הרצאת העשרה בנושא
VHDL

(or beyond LAB #1 experiments)
part II

Version 1.0 - May 2012

Amos Zaslavsky

Introduction

VHDL-2008 is here (alive and kicking..)

A Short **History** Lesson

- The Original VHDL-1987 standard was very powerful !
- Previous updated revisions were minor
  - VHDL-1993 (files, direct instantiation, extended naming of identifiers, some minor coding style change, shared variables ..)
  - VHDL-2002 (protected shared variables, improved buffer mode, bigger real ..)
- Because the basic version was very powerful and modifications were very minor, many tool vendors ignored modifications or upgraded their tools very slowly (why bother ..)
- Many people consider only Verilog 2001 to be a matching competitor to VHDL (this tie situation continues from 2001 until 2005)
- SystemVerilog (2005) is a superset of Verilog that improves it's verification capabilities. Some are beyond VHDL's verification capabilities.
- VHDL-2008 is a Major Revision with many improvement

Improvements in VHDL-2008 are in areas:

- Making VHDL language stronger (many more features..)
- Making VHDL language easier to use (less cumbersome..)

- Enhancements in the area of Verification (adding powerful features to existing capabilities + More coming soon..)

The final goal (מה מתבשל בתנור)

- Why update VHDL ? Instead, why not adopt SystemVerilog as verification Language ?
- VHDL already includes many system-level modeling features such as: records, access types (pointers), Protected types & shared variables, PSL and many many other advanced features that can be enhanced with relatively little effort, and new features can be added in a way that integrates cleanly with existing features.
- Organizations using VHDL already have significant experience using the language and a pool of people familiar with the language.
- Using SystemVerilog wold force users to learn a language that is different or have to manage multilingual environments (causing lower productivity)
- Final Goal: **VHDL** = **V**erification & **H**ardware **D**escription **L**anguage
- The next enhancements are mostly Verification focused:
  - Object Oriented Classes
  - Verification Data structures
  - More powerful Randomization (enhancing existing capabilities)
  - Functional Coverage
  - & Incorporating good things from SystemVerilog, SystemC, Vera & E

Presentation goals

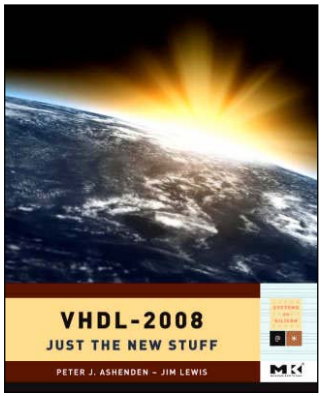
מטרה

ההצבעה על כיוונים

(מבלי להעמיק בחומר ומבלי לעבור על כל התוספות)

We will no go into all the modifications..

- For More information (or use the new LRM :-( )



Topics can all be found in

Topics that will be Mentioned here

- More reserved words — Simulation/Verification
- More convenient Comments
- More powerful & convenient constant string assignments
- More convenient mixed bit & vector Logical operations
- Unary reduction operations
- More built in arrays
- numeric_std_unsigned from IEEE replaces std_logic_unsigned from Synopsys
- Better relations
- New maximum(_), & minimum(_) functions
- New Fixed point and float vectors
- The Context design unit
- The condition operator
- More convenient Case operations
- conditional assignments & selected assignments in process

Topics that will be Mentioned here

- More convenient combinatorial descriptions
- More convenient sync descriptions
- New to_string(_) functions — Simulation/Verification only
- expressions written to ports when instantiation of components
- Revision of Modes (port directions) — Simulation/Verification directed
- More convenient & powerful generate statements
- Forcing signals through hierarchies using external names
- Forcing signals through hierarchies using external names — Simulation/Verification only
- Forcing ports: Driving value & Effective value
- Embedded PSL
- IP (Intellectual Property) protection
- VHDL Procedural Interface (VHPI) — Simulation/Verification directed

More reserved words

Reserved names VHDL87

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	in	inout
is	label	library	linkage	loop
map	mod	nand	new	next
nor	not	null	of	on
open	or	others	out	package
port	procedure	process	range	record
register	rem	report	return	select
severity	signal	subtype	then	to
transport	type	units	until	use
variable	wait	when	while	with
xor				

More reserved names VHDL-93/2002/2008

group	impure	inertial	literal	posponed
pure	reject	rol	ror	shared
sla	sll	sra	srl	unaffected
xnor				

VHDL-93

protected

VHDL-2002

VHDL-2008

assume	assume_guarantee	context	cover
default	fairness	force	parameter
property	release	restrict	restrict_guarantee
sequence	strong	vmode	vprop
vunit			

PSL words

More convenient Comments

Comments

```
-- this comment ends with <CR> <LF>

y <= a and b and c ; -- right part of this line is a comment

-- this
-- is
-- a
-- long
-- comment
```

```
/* this is a delimiter
   comment in VHDL-2008 */
```

More powerful & convenient constant string assignments

Old & New bit-string literals

```
y <= b"11110110" ;          -- binary

y <= x"F6" ;                -- hex

y <= "1111" & "011" & '0' ; -- default vector is bin

y <= x"f" & o"3" & '0' ;    -- octal & lowercase too
```

ok on older versions

```
y <= 246 ; -- error

y <= d"246" ;  -- ERROR d is not a legal prefix
```

```
y <= d"246" ;  -- OK in VHDL-2008 !
```

Specifying width in new bit-string literals

- Padding with leading zeros on the left

```
13d"246"    => "0000011110110"
13o"366"    => "0_000_011_110_110"
13x"f6"     => "0_0000_1111_0110"
10o"0366" => "0_011_110_110"
10x"0f6"  => "00_1111_0110"
```

- Signed values (Unsigned is the default)

```
12ux"f6" => "0000_1111_0110"  (unsigned extension)
12x"f6"  => "0000_1111_0110"  (unsigned extension)
12sx"f6" => "1111_1111_0110"  (signed extension)
```

- assigning std_logic_vector with meta-characters

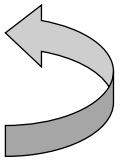
```
12o"01X" => "000_111_XXX"
```


More convenient
mixed bit & vector
Logical operations

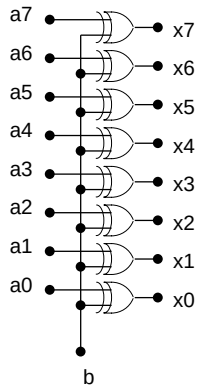
Mixed vector & bit operations

```
-- a polarity changer ?
entity op_bv1 is
   port ( x : out bit_vector(7 downto 0) ;
          a : in  bit_vector(7 downto 0) ;
          b : in  bit                    ) ;
end op_bv1 ;
architecture arc_op_bv1 of op_bv1 is
begin
   x <= a xor b ;
end arc_op_bv1 ;
```

Not legal on older Versions

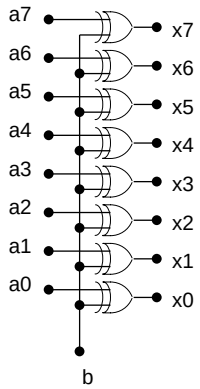


OK on VHDL-2008



Solution on older versions

```
-- a polarity changer
entity op_bv1 is
   port ( x : out bit_vector(7 downto 0) ;
          a : in  bit_vector(7 downto 0) ;
          b : in  bit                    ) ;
end op_bv1 ;
architecture arc_op_bv1 of op_bv1 is
   signal b_bv : bit_vector ( 7 downto 0 ) ;
begin
   b_bv <= ( others => b ) ;
   x <= a xor b_bv ;
end arc_op_bv1 ;
```



Solution on older versions

```
-- some other solutions (less elegant)

x <= a xor ( b & b & b & b & b & b & b & b ) ;

x <= a xor ( b , b , b , b , b , b , b , b ) ;

x <= ( a(7) xor b , a(6) xor b , a(5) xor b , a(4) xor b ,
       a(3) xor b , a(2) xor b , a(1) xor b , a(0) xor b ) ;

x(7) <= a(7) xor b ;
x(6) <= a(6) xor b ;
x(5) <= a(5) xor b ;
x(4) <= a(4) xor b ;
.
..
```


Unary reduction operations

What had to be done in older versions

```
-- no reduced operations on old VHDL
entity reduced_vec is
   port ( din    : in  bit_vector(7 downto 0) ;
          parity : out bit                    ) ;
end reduced_vec ;
architecture arc_reduced_vec of reduced_vec is
begin
   parity <= din(7) xor din(6)
                    xor din(5) xor din(4)
                    xor din(3) xor din(2)
                    xor din(1) xor din(0);
end arc_reduced_vec ;
```

Unary reduction operations in VHDL-2008

```
-- reduced operations on a vector (VHDL-2008)
entity reduced_vec is
   port ( din    : in  bit_vector(7 downto 0) ;
          parity : out bit                    ) ;
end reduced_vec ;
architecture arc_reduced_vec of reduced_vec is
begin
   parity <= xor din ;
end arc_reduced_vec ;
```


More built in arrays

boolean, integer, real & time arrays

```
-- decoder 3->8 using numeric relations
entity dec3t8 is
   port ( din    : in  integer range 0 to 7       ;
          enable : in  boolean                    ;
          d      : out boolean_vector(7 downto 0) ) ;
end dec3t8 ;
architecture arc_dec3t8 of dec3t8 is
begin
   -- positional association
   d <= ( ( din=7 ) and enable ,
          ( din=6 ) and enable ,
          ( din=5 ) and enable ,
          ( din=4 ) and enable ,
          ( din=3 ) and enable ,
          ( din=2 ) and enable ,
          ( din=1 ) and enable ,
          ( din=0 ) and enable ) ;
end arc_dec3t8 ;
```

in old VHDL only bit_vector (or std_logic_vector) and strings

boolean_vector, integer_vector, real_vector, time_vector

boolean, integer, real & time arrays

```
-- decoder 3->8 using numeric relations
entity dec3t8 is
   port ( din    : in  integer range 0 to 7       ;
          enable : in  boolean                    ;
          d      : out boolean_vector(7 downto 0) ) ;
end dec3t8 ;
architecture arc_dec3t8 of dec3t8 is
begin
   -- named association
   d <= ( 7 => ( din=7 ) and enable ,
          6 => ( din=6 ) and enable ,
          5 => ( din=5 ) and enable ,
          4 => ( din=4 ) and enable ,
          3 => ( din=3 ) and enable ,
          2 => ( din=2 ) and enable ,
          1 => ( din=1 ) and enable ,
          0 => ( din=0 ) and enable ) ;
end arc_dec3t8 ;
```

in old VHDL only bit_vector (or std_logic_vector) and strings

boolean_vector, integer_vector, real_vector, time_vector

std_logic_unsigned from Synopsys is replaced by numeric_std_unsigned from IEEE

New: numeric_std_unsigned

- The most popular 3 declarations (math operations on vectors)

```
-- Using the classic package of Synopsys in old VHDL
library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_unsigned.all ;
```

- Replaced by:

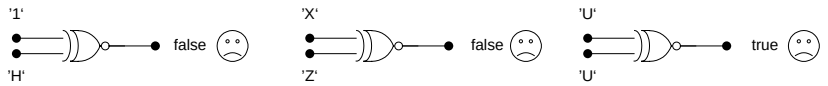
```
-- Same thing with new package of IEEE in VHDL-2008
library ieee ;
use ieee.std_logic_1164.all ;
use ieee.numeric_std_unsigned.all ; -- a new package !
```

- More operations (divisions,power)

Better relations

Problems with old = relations

=	'U'	'X'	'0'	'1'	'Z'	'W'	'L'	'H'	'-'
'U'	true	false	false	false	false	false	false	false	false
'X'	false	true	false	false	false	false	false	false	false
'0'	false	false	true	false	false	false	false	false	false
'1'	false	false	false	true	false	false	false	false	false
'Z'	false	false	false	false	true	false	false	false	false
'W'	false	false	false	false	false	true	false	false	false
'L'	false	false	false	false	false	false	true	false	false
'H'	false	false	false	false	false	false	false	true	false
'-'	false	false	false	false	false	false	false	false	true



Not behaving like real hardware

Return Boolean

New (matched) ?= relation

?=	'U'	'X'	'0'	'1'	'Z'	'W'	'L'	'H'	'-'
'U'	'U'	'U'	'U'	'U'	'U'	'U'	'U'	'U'	'1'
'X'	'U'	'X'	'X'	'X'	'X'	'X'	'X'	'X'	'1'
'0'	'U'	'X'	'1'	'0'	'X'	'X'	'1'	'0'	'1'
'1'	'U'	'X'	'0'	'1'	'X'	'X'	'0'	'1'	'1'
'Z'	'U'	'X'	'X'	'X'	'X'	'X'	'X'	'X'	'1'
'W'	'U'	'X'	'X'	'X'	'X'	'X'	'X'	'X'	'1'
'L'	'U'	'X'	'1'	'0'	'X'	'X'	'1'	'0'	'1'
'H'	'U'	'X'	'0'	'1'	'X'	'X'	'0'	'1'	'1'
'-'	'1'	'1'	'1'	'1'	'1'	'1'	'1'	'1'	'1'

Any compare to '-' (don't care) is '1' → Any compare to 'U' is 'U' → Strong '0' & '1' equal weak 'L' & 'H' → 'W' 'X' & 'Z' is same as 'X'

Vectors are and-ed together, with any 'X' producing 'X' result and any '0' producing '0' result

Behaving like real hardware

Problems with old relations - (<)

<	right								
left	'U'	'X'	'0'	'1'	'Z'	'W'	'L'	'H'	'-'
'U'	false	true	true	true	true	true	true	true	true
'X'	false	false	true	true	true	true	true	true	true
'0'	false	false	false	true	true	true	true	true	true
'1'	false	false	false	false	true	true	true	true	true
'Z'	false	false	false	false	false	true	true	true	true
'W'	false	false	false	false	false	false	true	true	true
'L'	false	false	false	false	false	false	false	true	true
'H'	false	false	false	false	false	false	false	false	true
'-'	false	false	false	false	false	false	false	false	false

'U' < 'X' < '0' < '1' < 'Z' < 'W' < 'L' < 'H' < '-'

No Meaningful hardware behavior

Return Boolean

New (matched) ?< relation

?<	right								
left	'U'	'X'	'0'	'1'	'Z'	'W'	'L'	'H'	'-'
'U'	'U'	'U'	'U'	'U'	'U'	'U'	'U'	'U'	'X'
'X'	'U'	'X'	'X'	'X'	'X'	'X'	'X'	'X'	'X'
'0'	'U'	'X'	'0'	'1'	'X'	'X'	'0'	'1'	'X'
'1'	'U'	'X'	'0'	'0'	'X'	'X'	'0'	'0'	'X'
'Z'	'U'	'X'	'X'	'X'	'X'	'X'	'X'	'X'	'X'
'W'	'U'	'X'	'X'	'X'	'X'	'X'	'X'	'X'	'X'
'L'	'U'	'X'	'0'	'1'	'X'	'X'	'0'	'1'	'X'
'H'	'U'	'X'	'0'	'0'	'X'	'X'	'0'	'0'	'X'
'-'	'X'	'X'	'X'	'X'	'X'	'X'	'X'	'X'	'X'

Any compare to '-' (don't care) is error → Any compare to 'U' is 'U' → Strong '0' & '1' equal weak 'L' & 'H' → 'W' 'X' & 'Z' is same as 'X'

Behaving like real hardware

Other Matched relations

- Are all created by operations on the previous relations
  - **?/= is an inverted ?=**
  - **?>= is iverted ?<**
  - **?<= is ?< or-ed with ?=**
- Example of usage:

```
library ieee ;
use ieee.std_logic_1164.all ;
use ieee.numeric_std_unsigned.all ; -- present for math meaning
entity good_compare2 is
   port ( a       : in  std_logic_vector(3 downto 0) ;
          b       : in  std_logic_vector(2 downto 0) ;
          a_gte_b : out std_logic                    ) ;
end good_compare2 ;
architecture arc_good_compare2 of good_compare2 is
begin
a_gt_b <= ( a ?>= b ) ; -- matching directional compare
end arc_good_compare2 ;
```


Order of precedence of operators

Order of Precedence of operators

operator level (low to high)	Operators
condition	??
logical	and, or, nand, nor, xor, xnor
relational	=, /=, <, <=, >, >=, ?=, ?/=, ?<, ?<=, ?>, ?>=
shift	sll , slr , sla , sra, rol, ror
adding	+, - , &
sign	+, -
multiplying	*, /, mod, rem
miscellaneous	**, abs, not
parentheses	( )

New maximum(_), & minimum(_) functions

minimum(_) & maximum(_) functions

- Can be used with the following data types: character, bit, integer, real, time, string, boolean_vector, bit_vector, integer_vector, real_vector, time_vector, std_logic_vector and many more types .. (any data types that relations can be operated on)

- Usage:

```
maximum( object1 , object2 ) -- max between 2
maximum ( array_type ) -- returns max array element
```

- Example of usage:

```
constant w1 : integer := 3 ;
constant w2 : integer := 2 ;
signal y : std_logic_vector(maximum(w1,w2) downto 0) ;
```

New Fixed point and float vectors

More Powerful vector data types

- The new Powerful data types
  - Unsigned fixed - ufixed
  - Signed fixed - sfixed
  - Floating Point - float
- They are all vectoric data types !
- Useful for:
  - Large dynamic range (including fractions)
  - Complex Arithmetic and DSP applications
- fixed types consume less hardware resources but have lower dynamic range, float has more dynamic range but hardware is more complex
- Support for synthesis: In a short time
  - For example, Altera has already the flowing floating point macrofunctions: ALTFP_ABS, ALTFP_ADD_SUB, ALTFP_COMPARE, ALTFP_CONVERT, ALTFP_DIV, ALTFP_EXP, ALTFP_INV, ALTFP_INV_SQRT, ALTFP_LOG, ALTFP_MULT, ALTFP_SQRT, and more coming soon... & fixed = integers

ufixed & sfixed declarations & assignments

- Declarations & assignments (negative index range is fraction)

```
signal u : ufixed (1 downto -4) ;
signal s : sfixed (3 downto -1) ;
...
u <= "101010" ; -- 10.1010 = 2.625
s <= "11100"  ; -- 1110.0  = -2
```

- Equivalent Assignments (parameters are index ranges)

```
u <= to_ufixed(2.625,1,-4) ; -- 10.1010 = 2.625
s <= to_sfixed(  -2 ,3,-1) ; -- 1110.0  = -2
```

- Equivalent Assignments (parameter copies ranges from declarations of u and s respectivly)

```
u <= to_ufixed(2.625,u) ; -- 10.1010 = 2.625
s <= to_sfixed(  -2 ,s) ; -- 1110.0  = -2
```


ufixed & sfixed declarations & assignments

- Code example of an adder

```
library ieee ;
use ieee.fixed_pkg.all ;
entity fixed_vec is
   port ( a,b : in  ufixed(3 downto -4) ;   -- width 8
          y   : out ufixed(4 downto -4) ) ; -- width 9
end fixed_vec ;
architecture arc_fixed_vec of fixed_vec is
begin
   y <= a + b ; -- both sides are width 9
end arc_fixed_vec ;
```

Example

c	c	c	c							
	1	1	1	1	.	0	0	0	0
	1	1	1	1	.	0	0	0	0
1	1	1	1	0	.	0	0	0	0

- Operands are 8 bit wide & Result is 9 bits (Full Precision math with no automatic modulo-truncation)

with old vector (integer) math

- Both sides of assignments are not equal so it is not a legal code

```
-- error
library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_unsigned.all ;
entity vec_math is
   port ( a,b : in  std_logic_vector(3 downto 0) ;
          y   : out std_logic_vector(4 downto 0) ) ;
end vec_math ;
architecture arc_vec_math of vec_math is
begin
   y <= a + b ; -- error
end arc_vec_math ;
```


with old vector (integer) math

- Now you get automatic modulo-truncation

```
-- OK but with different results
library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_unsigned.all ;
entity vec2_math is
   port ( a,b : in  std_logic_vector(3 downto 0) ;
          y   : out std_logic_vector(3 downto 0) ) ;
end vec2_math ;
architecture arc_vec2_math of vec2_math is
begin
   y <= a + b ;
end arc_vec2_math ;
```

Example

c	c	c	c
	1	1	1	1
	1	1	1	1
	1	1	1	0

with old vector (integer) math

- Handling the truncated bit

```
-- Handling truncation bit
library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_unsigned.all ;
entity vec3_math is
   port ( a,b : in  std_logic_vector(3 downto 0) ;
          y   : out std_logic_vector(4 downto 0) ) ;
end vec3_math ;
architecture arc_vec3_math of vec3_math is
begin
   y <= ('0' & a) + ('0' & b) ;
end arc_vec3_math ;
```


Other operations on ufixed

- Result size

operation	result range
A + B , A - B	Max(A'left,B'left)+1 downto Min(A'right,B'right)
A * B	A'left+B'left+1 downto A'right+B'right
A / B	A'left-B'right downto A'right-B'left-1
A rem B	Min(A'left,B'left) downto Min(A'right,B'right)
A mod B	B'left downto Min(A'right,B'right)

- example:

```
signal A : ufixed (7 downto -4) ;
signal B : ufixed (3 downto -9) ;
```

- Width of multiplication result

```
ufixed(7+3+1 downto (-4)+(-9))
```

- No meaning for abs(_) and unary -(_) with ufixed

Other operations on sfixed

- Result size

operation	result range
A + B , A - B	Max(A'left,B'left)+1 downto Min(A'right,B'right)
A * B	A'left+B'eft+1 downto A'right + Bright
A / B	A'left-B'right+1 downto A'right-B'left
A rem B	Min(A'left,B'left) downto Min(A'right,B'right)
A mod B	Min(A'left,B'left) downto Min(A'right,B'right)
abs(A)	A'left+1 downto A'right
-A	A'left+1 downto A'right

- Added operations meaning for abs(_) and unary -(_) with sfixed

More operations

- Mixing with integer and real

```
signal a : sfixed (8 downto -5);
signal b : real ;
...
y <= a + b ;
```

- Equivalent to last assignment (b converted to ranges of a)

```
y <= a + sfixed(b,8,-5) ;
```

- Relation (Compare operations)

```
= , /= , < , > , >= , <= , ?= , ?/= , ?< , ?> , ?>= , ?<=
```

- Logical operations

```
and, or, nand, nor, xnor, not
```

- Conversion functions

```
sll, srl, rol, ror, sla, sra
```

- Conversion functions

to_sfixed(_),to_ufixed(_),to_real(_),to_integer(_),to_std_logic(_), to_string(_),resize(_)

New Floating point vectors

Beyond fixed - float

- Float ⇔ Fixed
  - more (dynamic range + hardware) ⇔ less (dynamic range + hardware)
  - constant relative accuracy ⇔ constant absolute accuracy
- What's wrong with real
  - Fixed range of 64 bit not suitable for all applications
  - No easy and direct access to bits of number

- Package float_pkg enables usage of
  - Standard 32 bit & 64 bit numbers (IEEE-754)
  - Standard 128 bit numbers
  - Any width (compliant with IEEE-854)
- 3 Parts (mandatory)

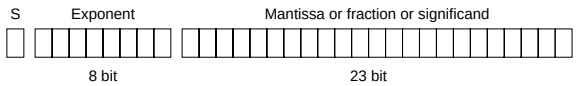
```
sign.exponent.fraction
```

- Examples in next foils..

float32 example (normalized)

- Structure

Structure of standard 32 bit floating point vector = float32



- Normalization method of fraction is 1.fraction (1 not represented)
- Calculation of a normalized number

$$NORMILIZED\_NUM = (-1)^{SIGN} \cdot 2^{EXPONENT-127} \cdot (1.0 + FRACTION)$$

- example:

```
use ieee.float_pkg.all ;
..
signal f : float (8 downto -23) ; -- bit 8 is sign
..
begin
   f <= to_float(6.5,8,23) ;
```


float32 example (normalized)

- the previous function to_float(_) converts real to float
- Bit 8 (MSB) is the sign bit
- bits (7 downto 0) are for exponent
- bits (-1 downto 23) are for mantissa=fraction=significand
- Direction of vector must always be downto
- Equivalent declaration to the previous one:

```
signal f : float32 ; -- a shorter way
```

- Equivalent assignments:

```
f <= to_float(6.5,f) ;
f <= '0' & "10000001" & "10100000000000000000000" ;
```

- Calculation of last assignment:

$$(-1)^{SIGN} \cdot 2^{EXPONENT-127} \cdot (1.0 + FRACTION)$$
$$= 2^{129-127} \cdot (1.0 + 0.625) = (4) \cdot (1.625) = 6.5$$

float32 example (normalized)

- The smallest possible normalized values (exponent must not be zero)

```
f <= '0' & "00000001" & "00000000000000000000000" ;
f <= '1' & "00000001" & "00000000000000000000000" ;
```

- Calculation of the smallest normalized positive value

$$(-1)^{SIGN} \cdot 2^{EXPONENT-127} \cdot (1.0 + FRACTION)$$
$$= 2^{1-127} \cdot (1.0 + 0.0) = 2^{-126}$$
$$\approx (1.175494350) \cdot 10^{-38}$$

- exponent 00000000 is considered un-normalized and is calculated differently (next foils)

float32 example (un-normalized)

- Un-normalized (De-Normalized) values have a zero exponent

$$DENORMILIZED\_NUMBER = (-1)^{SIGN} \cdot 2^{-126} \cdot (FRACTION)$$

- Biggest values are & calculation of biggest positive value:

```
f <= '0' & "00000000" & "11111111111111111111111" ;
f <= '1' & "00000000" & "11111111111111111111111" ;
```

$$DENORMILIZED\_NUMBER = (-1)^{SIGN} \cdot 2^{-126} \cdot (FRACTION)$$
$$= (-1)^{SIGN} \cdot 2^{-126} \cdot (1 - 2^{-23})$$
$$\approx (1.175494210) \cdot 10^{-38}$$

- Mid values are & calculation of mid positive value:

```
f <= '0' & "00000000" & "10000000000000000000000" ;
f <= '1' & "00000000" & "10000000000000000000000" ;
```

$$DENORMILIZED\_NUMBER = (-1)^{SIGN} \cdot 2^{-126} \cdot (FRACTION)$$
$$= 2^{-126} \cdot (0.5) = 2^{-127}$$
$$\approx (5.87747..) \cdot 10^{-39}$$

float32 example (un-normalized)

- Smallest values are & calculation smallest positive value:

```
f <= '0' & "00000000" & "00000000000000000001" ;
f <= '1' & "00000000" & "00000000000000000001" ;
```

$$DENORMILIZED\_NUMBER = (-1)^{SIGN} \cdot 2^{-126} \cdot (FRACTION)$$
$$= 2^{-126} \cdot (2^{-23}) = 2^{-149}$$
$$\approx (1.401298..) \cdot 10^{-45}$$

- Any smaller number then that, is considered zero & calculation of it's positive value (zero) is:

```
f <= '0' & "00000000" & "00000000000000000000" ;
f <= '1' & "00000000" & "00000000000000000000" ;
```

$$DENORMILIZED\_NUMBER = (-1)^{SIGN} \cdot 2^{-126} \cdot (FRACTION)$$
$$= (-1)^{SIGN} \cdot 2^{-126} \cdot (0)$$
$$= 0$$

float32 example (big-normalized)

- The biggest possible normalized values (exponent no zeros) & calculation of it's value

```
f <= '0' & "11111110" & "11111111111111111111" ;
f <= '1' & "11111110" & "11111111111111111111" ;
```

$$(-1)^{SIGN} \cdot 2^{EXPONENT-127} \cdot (1.0 + FRACTION)$$
$$= 2^{254-127} \cdot (1.0 + (1 - 2^{-23}))$$
$$= 2^{+127} \cdot (2.0 - 2^{-23}))$$
$$\approx (3.402823...) \cdot 10^{38}$$

- Values beyond that are considered infinite:

```
f <= '0' & "11111111" & "00000000000000000000" ;
f <= '1' & "11111111" & "00000000000000000000" ;
```

- Infinite is result of division of non zero by zero.

float32 example (NaN)

- Exponent 11111111 and fraction that is not all zeros is NaN
- Result of:
  - division of 0 by 0
  - Square root of -1
  - Addition of +infinity and -infinity
  - division of infinity by infinity
  - others..
- The fraction is considered a Payload that specifies code of problem
- Quiet NaN or qNaN (MSB of fraction is 1) ⇔ Signaling NaN or sNaN (MSB of fraction is 0)

```
f <= '0' & "11111110" & "1??????????????????????" ;
f <= '1' & "11111110" & "0??????????????????????" ;
```

- After Quiet NaN or qNaN calculations can continue
- After Signaling NaN or sNaN calculations are stopped

Example of floating point divider

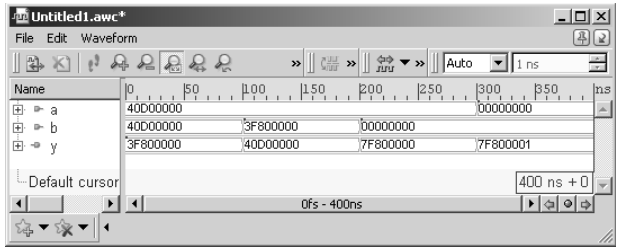
```
-- example of a floating point divider
library ieee ;
use ieee.float_pkg.all ;
entity fdiv is
   port ( a,b : in  float32 ;
          y   : out float32 ) ;
end fdiv ;
architecture arc_fdiv of fdiv is
begin
   y <= a / b ;
end arc_fdiv ;
```

Non Standard floating points can be declared too

```
standard 64 & 128 bit
signal a,b : float64  ;
signal a,b : float128 ;
-- non standard 16 bit
signal a,b : float(5 downto -10) ;
-- non standard 7 bits (minimum)
signal a,b : float(3 downto  -3) ;
```

Example of floating point divider

```
force a x"40d00000" ;# 6.5
force b x"40d00000" ;# 6.5
run 100 ns          ;# expected x3f800000 = 1.0
force b x"3f800000" ;# 1.0
run 100 ns          ;# expected x40d00000 = 6.5
force b x"00000000" ;# 0.0
run 100 ns          ;# expected x7f800000 = inf
force a x"00000000" ;# 0.0
run 100 ns          ;# expected x7f800001 = NaN
```



More standard float formats (64 & 128 bits)

- Declaring standard 64 & 128 bit standard floats

```
signal f1 : float(11 downto  -52) ; -- standard 64
signal f2 : float(15 downto -112) ; -- standard 128
```

- An alternative declarations

```
signal f1 : float64  ; -- standard 64  bit
signal f2 : float128 ; -- standard 128 bit
```

- The general Normalized formula:

$$NORMILIZED\_NUM = (-1)^{SIGN} \cdot 2^{EXPONENT-(EXPONENT\_BIAS)} \cdot (1.0 + FRACTION)$$

$$EXPONENT\_BIAS = 2^{EXPONENT\_WIDTH-1} - 1$$

- The General De-Normalized Formula:

$$DENORMILIZED\_NUM = (-1)^{SIGN} \cdot 2^{-(EXPONENT\_BIAS-1)} \cdot (FRACTION)$$

$$EXPONENT\_BIAS = 2^{EXPONENT\_WIDTH-1} - 1$$

More standard float formats (64 & 128 bits)

- The Standard 64 float Formula:

$$NORMILIZED\_N_{64} = (-1)^{SIGN} \cdot 2^{EXPONENT-1023} \cdot (1.0 + FRACTION)$$

$$DENORMILIZED\_N_{64} = (-1)^{SIGN} \cdot 2^{-(1022)} \cdot (FRACTION)$$

- The Standard 64 float Formula:

$$NORMILIZED\_N_{128} = (-1)^{SIGN} \cdot 2^{EXPONENT-16383} \cdot (1.0 + FRACTION)$$

$$DENORMILIZED\_N_{128} = (-1)^{SIGN} \cdot 2^{-(16382)} \cdot (FRACTION)$$

More on float format

- Non standards of 16 bits & 7 bits examples or others possible too..

```
signal f3 : float(5 downto -10) ; -- 16 bit
signal f4 : float(3 downto  -3) ; -- minimum 7 bits
```

- Operations supported

```
+, -, *, /, rem, mod, abs(_), -(unary)
```

- Ranges are adjusted to the widest exponent & fraction

```
-- adjusting floats with different ranges
library ieee ; use ieee.float_pkg.all ;
entity fadd is
port ( a : in  float( 8 downto -23) ;
       b : in  float(11 downto -52) ;
       y : out float(11 downto -52) ) ;
end fadd ;
architecture arc_fadd of fadd is
begin
   y <= a + b ;
end arc_fadd ;
```

Adjusted to this

More on float format

- Last assignment is equivalent to:

```
y <= resize(a,11,52) + b ;
y <= resize(a,b) + b ;
```

- type real is automatically adjusted to the float operand:

```
-- real type adjusted to float
library ieee ; use ieee.float_pkg.all ;
entity fadd2 is
   port ( a : in  float(8 downto -23) ;
          b : in  real                ;
          y : out float(8 downto -23) ) ;
end fadd2 ;
architecture arc_fadd2 of fadd2 is
begin
   y <= a + b ;
end arc_fadd2 ;
```

Adjusted to this

More on float format

- Last assignment is equivalent to:

```
y <= a + to_float(b,a) ;
```

- If any operand has non strong '0' or '1' or non strong 'L' or 'H' result is "XX..X"
- Relations are all supported

```
= , /= , < , > , >= , <=
?= , ?/= , ?< , ?> , ?>= , ?<=
```

- Logical operations are all supported

```
and, or, nand, nor, xnor, not
```

- Many functions supported:
  - to_float(_), to_real(_), to_integer(_), to_stdlogic(_), to_stdlogic_vector(_)
  - resize(_)
  - to_string(_)
  - more...
- Will soon be supported by synthesis !

The Context design unit

The context design unit

- Makes including a group of libraries and packages easier using a single name reference
- Example of declaration (and then declaration to library proj_lib):

```
context easy is
   library ieee ;
   use ieee.std_logic_1164.all ;
   use ieee.std_logic_unsigned.all ;
   library proj_lib ;
   use proj_lib.proj_pack.all ;
   use std.textio.all ;
   use ieee.std_logic_textio.all ;
end context ;
```

- Example of usage:

```
library proj_lib ;
context proj_lib.easy ;
entity . . . is
. . .
```

The condition operator

The ?? condition Operator

```
-- Introducing the condition operator "??"
library ieee ;
use ieee.std_logic_1164.all ;
entity condition is
   port ( din  : in  std_logic ;
          dout : out boolean   ) ;
end condition ;
architecture arc_condition of condition is
begin
   dout <= ?? din ; -- convert std_logic to boolean
end arc_condition ;
```

- '1' or 'H' is converted to true
- other values ('0','L','X','Z','U','W'-') are converted to false

Old style (not convenient)

```
entity condition1 is
   port( cs0,cs1,pol : in  bit                    ;
         din         : in  bit_vector(3 downto 0) ;
         dout        : out bit_vector(3 downto 0) ) ;
end condition1 ;
architecture arc_condition1 of condition1 is
begin
   process (din,cs0,cs1,pol)
   begin
      if    cs1 = '1' and cs0 = '0' and pol = '1' then
         dout <= din ;
      elsif cs1 = '1' and cs0 = '0' and pol = '0' then
         dout <= not din ;
      else
         dout <= ( others => '0' ) ;
      end if ;
   end process ;
end arc_condition1 ;
```

boolean

Old style (using the ?? operator)

```
-- using the ?? operator
architecture arc_condition2 of condition2 is
begin
   process (din,cs0,cs1,pol)
   begin
      if    ?? ( cs1 and not cs0 and      pol ) then
         dout <= din ;
      elsif ?? ( cs1 and not cs0 and not pol ) then
         dout <= not din ;
      else
         dout <= ( others => '0' ) ;
      end if ;
   end process ;
end arc_condition2 ;
```

Not boolean

Old style (?? implicitly implied)

```
-- the ?? operator implicitly implied
architecture arc_condition3 of condition3 is
begin
   process (din,cs0,cs1,pol)
   begin                                    OK too
      if    cs1 and not cs0 and     pol then
         dout <= din ;
      elsif cs1 and not cs0 and not pol then
         dout <= not din ;
      else
         dout <= ( others => '0' ) ;
      end if ;
   end process ;
end arc_condition3 ;
```


Where can this implicitly implied ?? be used

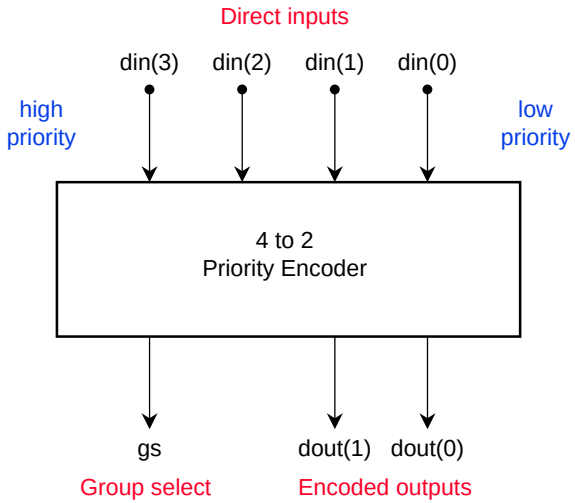
- After an if clause (as in previous example) in an if statement
- After an elsif clause in an if statement
- After an until clause in a wait statement
- After an assert statement
- After a while clause in a while loop statement
- After a when clause in a next statement
- After a when clause in an exit statement
- After an if clause (as in previous example) in an if generate statement
- After an elsif clause in an if generate statement
- In a boolean expression in a PSL declaration
- In a boolean expression in a PSL statement

There must be no ambiguity allowed

```
-- implicit conversion NOT applied
-- because of ambiguous interpretation !
architecture arc_condition4 of condition4 is
begin
   process (din,cs0,cs1,pol)
   begin
      if    cs1 and not cs0 and pol = '1' then -- error
         dout <= din ;
      elsif cs1 and not cs0 and pol = '0' then -- error
         dout <= not din ;
      else
         dout <= ( others => '0' ) ;
      end if ;
   end process ;
end arc_condition4 ;
```


More convenient
Case operations

Priority encoder example (again)



The old case statement

```
case to_x01(din) is
   when  "1000" | "1001" | "1010" | "1011"
       | "1100" | "1101" | "1110" | "1111" =>
      dout <= "11" ;  gs <= '1' ;
   when  "0100" | "0101" | "0110" | "0111" =>
      dout <= "10" ;  gs <= '1' ;
   when "0010" | "0011" =>
      dout <= "01" ;  gs <= '1' ;
   when "0001" =>
      dout <= "00" ;  gs <= '1' ;
   when "0000" =>
      dout <= "00" ;  gs <= '0' ;
   when others =>
      dout <= "XX" ;  gs <= 'X' ; report "unknown din" ;
end case ;
```

An new elegant matching case? statement

```
case? din is  -- using the ?= (built in matching) operator
   when "1---" =>
      dout <= "11" ;  gs <= '1' ;
   when "01--" =>
      dout <= "10" ;  gs <= '1' ;
   when "001-" =>
      dout <= "01" ;  gs <= '1' ;
   when "0001" =>
      dout <= "00" ;  gs <= '1' ;
   when "0000" =>
      dout <= "00" ;  gs <= '0' ;
   when others =>
      dout <= "XX" ;  gs <= 'X' ; report "unknown din" ;
end case ;
```

'-' in code is treated as Φ

'H' is treated as '1'
'L' is treated as '0'
no need for to_x01()

Works with selected assignments too

```
-- direct priority filter with
-- matching selected assignment
library ieee ;
use ieee.std_logic_1164.all ;
entity priority is
   port ( din  : in  std_logic_vector(3 downto 0) ;
          dout : out std_logic_vector(3 downto 0) ) ;
end priority ;
architecture arc_priority of priority is
begin
   with din select?                -- VHDL-2008
   dout <= "1000" when "1---" ,
           "0100" when "01--" ,
           "0010" when "001-" ,
           "0001" when "0001" ,
           "0000" when others ;
end arc_priority ;
```

conditional assignments & selected assignments in process

Complex assignments in process (1 of 2)

```
-- mixed 4=>1 mux with enable
entity mixedmux is
   port ( din  : in  bit_vector(3 downto 0) ;
          sel  : in  bit_vector(1 downto 0) ;
          ena  : in  bit                    ; -- ena
          dout : out bit                    ) ;
end mixedmux ;
architecture arc_mixedmux of mixedmux is
begin
   process (din,sel,ena)
```


Complex assignments in process (2 of 2)

```
process (din,sel,ena)
   begin
      if    sel = "00" then
         -- conditional assignment in process
         dout <= din(0) when ena = '1' else '0' ;
      elsif sel = "01" then
         -- conditional assignment in process
         dout <= din(1) when ena = '1' else '0' ;
      elsif sel = "10" then
         -- selected assignment in process
         with ena select
         dout <= din(2) when '0' ,
                 '0' when others ;
      else  -- sel="11"
         -- selected assignment in process
         with ena select
         dout <= din(1) when '0' ,
                 '0' when others ;
      end if ;
   end process ;
end arc_mixedmux ;
```


More convenient combinatorial descriptions

Complete sensitivity list in comb system

```
-- complete sensitivity list for VHDL-2008
architecture arc_comrule2 of comrule2 is
begin
   process ( all )  -- very elegant solution !
   begin
      y <= a or b or c ;
   end process ;
end arc_comrule2 ;


-- good for latches too !
```

More convenient
sync descriptions

using rising_edge(_) with bit too, no dilemma

```
-- using the rising_edge(_) function with bit too
entity dff7 is
   port ( clk : in  bit ;
          d   : in  bit ;
          q   : out bit ) ;
end dff7 ;
architecture arc_dff7 of dff7 is
begin
   process ( clk )
   begin
      if rising_edge(clk)then -- no more dilemma !
         q <= d ;
      end if ;
   end process ;
end arc_dff7 ;
```

Improved I/O operations
with the 'image(_) &
'value(_) attributes
(VHDL-93)

The 'image( ) attribute - VHDL-93

- Returns a string representing the value
  data_type_name'**image**(*data_value*)
- Usable only with scalar data types (not vectors)
- Useful for simulation or Verification (send strings to console)
- Here is an example

```
process ( din ) -- din is integer
   begin
      assert false
        report " [time=" & time'image(now) & "]" &
               " [din="  & integer'image(din) & "]"
        severity note ;
end process ;
```

- results

```
Note: [time=100 ns] [din=0]
Note: [time=200 ns] [din=3]
```


New to_string(_) functions

The to_string(_) function - VHDL-2008

- Here is an example

```
process ( din ) -- din is std_logic_vector(11 downto 0)
   begin
      assert false
         report " [time=" & to_string(now) & "]" &
                " [din="  & to_string(din) & "]"
         severity note ;
end process ;
```

- results

```
Note: [time=100 ns] [din=101011001111]
Note: [time=200 ns] [din=00001111HLXZ]
```

- All other old built in data types are supported too
- Data types ufixed, sfixed and float are supported too

to_string(_) function with more parameters

- Shown: units to use with time
- Shown: width of the right side of decimal point with real

```
process ( din ) -- din is now real
   begin
      assert false
         report " [time=" & to_string(now,ps) & "]" &
                " [din="  & to_string(din,10) & "]"
         severity note ;
end process ;
```

- results

```
Note: [time=100000 ps] [din=1.0000000000]
Note: [time=200000 ps] [din=6.5000000000]
```

- With type real you can also use other format strings like in printf of C: E%, e%, f%, g%

to_hstyring(_) & to_ostyring(_) functions

- Special functions to be used on vector types:
  `to_hstring( vector_type )`
  `to_ostring( vector_type )`
- Example with binary hex & octal presentation shown:

```
process ( din ) -- din is std_logic_vector(11 downto 0)
begin
   assert false
      report " [bin=" & to_string(din)  & "]" &
             " [hex=" & to_hstring(din) & "]" &
             " [oct=" & to_ostring(din) & "]"
      severity note ;
end process ;
```

- results on “101011001111” and “00001111HLXZ”

```
Note: [bin=101011001111] [hex=ACF] [oct=5317]
Note: [bin=00001111HLXZ] [hex=0FX] [oct=03XX]
```


expressions written to ports when instantiation of components

wiring of signals constants & expressions

- in VHDL-87 only clean signals are possible

```
u1: or_gate  port map ( s , q , x1 ) ;
```

- In VHDL-93 wiring of constants is possible

```
u1: and_gate8
   port map(output => y                          ,
            input(3 downto 0) => x(3 downto 0)   ,
            input(7 downto 4) => '1'&'1'&'1'&'1' ) ;
```

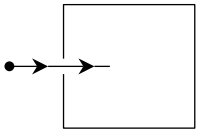
- in VHDL-2008 expressions are possible

```
u1: or_gate  port map ( not s , q , x1 ) ;
```

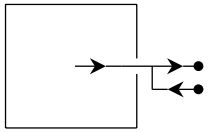

Revision of Modes (port directions)

Old - Modes (port directions)

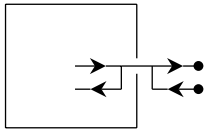
- in
  - The default mode
  - In signal can not be written (LHS) internally
  - In signal can only be read (RHS) internally



- out
  - Out signal can be externally bussed
  - Out signal can not be read back (RHS) internally !
  - Out signal can Only be written (LHS) internally

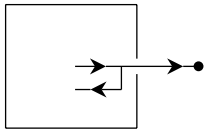


- inout
  - inout signal can be externally bussed
  - inout signal can be read back (RHS) internally
  - No internal restrictions (read write)



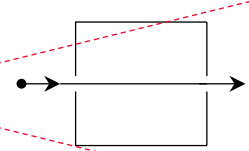
Old - Modes (port directions)

- buffer
  - Out signal can not be externally bussed
  - Out signal can be written (LHS) internally
  - Out signal can also be read (RHS) internally
  - Should be used more often !



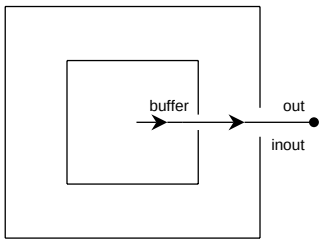
- linkage
  - Signal passes through
  - Signal can not be read from
  - Signal can not be written
  - Rarely used (usually for power supplies)

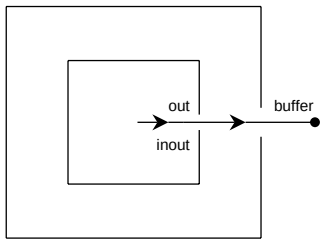
VHDL-2002 recommends removing this mode



Old - Modes (port directions)

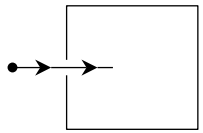
- buffer connected to out or inout from within a hierarchy
  - Warning: Can not associate buffer signal with out or inout signal
  - Restriction removed already on VHDL-2002
  - Many people did not know about it
  - Many tools ignored the 2002 revision





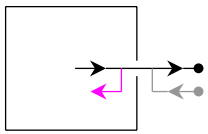
New - Modes (port directions)

- in
  - The default mode
  - In signal can not be written (LHS) internally
  - In signal can only be read (RHS) internally

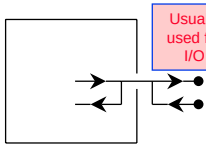


Will be used as normal output (not I/O) BUS

- out
  - Out signal can be externally bussed
  - Out signal can be read back (RHS) internally !
  - Out signal can be written (LHS) internally



- inout
  - inout signal can be externally bussed
  - inout signal can be read back (RHS) internally
  - No internal restrictions (read write)



Usually used for I/O

Will be used as I/O BUS (synthesis tools prefer this mode)

People will probably stop using buffer

Old coding style

```
library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_unsigned.all ;
entity count16 is
   port ( clk   : in  std_logic                      ;
          count : out std_logic_vector(3 downto 0) ) ;
end count16 ;
architecture arc_count16 of count16 is
   signal cnt : std_logic_vector(3 downto 0) ;
begin
   process (clk)
   begin
      if rising_edge(clk) then
        cnt <= cnt + 1 ;
      end if ;
   end process ;
   count <= cnt ;
end arc_count16 ;
```


New coding style

```
library ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_unsigned.all ;
entity count16 is
   port ( clk   : in  std_logic                      ;
          count : out std_logic_vector(3 downto 0) ) ;
end count16 ;
architecture arc_count16 of count16 is
   -- signal cnt : std_logic_vector(3 downto 0) ;
begin
   process (clk)
   begin
      if rising_edge(clk) then
        count <= count + 1 ;   -- now you can read mode out
      end if ;
   end process ;
   -- count <= cnt ;
end arc_count16 ;
```


More convenient & powerful generate statements

More powerful if-generate statements

```
-- old coding style
label1: if condition
   generate
   -- first alternative
   ...
end generate ;

label2: if not condition
   generate
   -- second alternative
   ...
end generate ;
```

```
-- same thing in new style
label: if condition generate
   -- first alternative
   ...
else generate
   -- second alternative
   ...
end generate ;
```

Full If-generate and case-generate syntax

```
-- syntax similar to
-- regular if clause
label: if condition1
   generate
    -- first alternative
    ...
elsif condition2 generate
    -- second alternative
    ...
    ...
else generate
    -- last default
   alternative
    ...
end generate ;
```

```
-- syntax similar to
-- regular case clauses
label: case
   tested_expression
   generate
    when choice1 =>
        -- first alternative
        ...
    when choice2 =>
        -- second alternative
        ...
    when others =>
        -- last default
   alternative
        ...
end generate ;
```


Example of case-generate statement

```
    -- constant version_number : integer := 1 ;
    constant version_number : integer := 2 ;
    ...
    ...
begin
    gencase: case version_number generate
      when 1 =>
        U1: comp1 port map( resetN , clk, din , dout ) ;
      when 2 =>
        U2: comp2 port map( resetN , clk, din , dout ) ;
      when others =>
        assert false
           report "Incorrect version number"
           severity error ;
    end generate ;
```


Monitoring signals from lower hierarchies using Modelsim's signal_spy utility (pre VHDL-2008)

The signal_spy utility

- The Modelsim signal_spy utility can mirror signal values from a lower hierarchy (source_object) into an existing signal in the current hierarchy (destination_object)
- It works only in Modelsim versions later then version 5.5
- Activation is done through calling a procedure - once (see example later)
- Syntax is

```
init_signal_spy (src_object, dest_object [,verbose] ) ;
```

- *src_object* and *dest_object* are strings that represent signal names. Whole signals should be used (no slices or vector elements)
- The verbose value is an optional integer and it's values can be be: either 0 (default) or 1
- You also need to make the following library & package visible:

```
library modelsim_lib ;
use modelsim_lib.util.all;
```

Example of using the signal_spy utility

```
-- monitoring a signal from a lower hierarchy
library ieee ;
use ieee.std_logic_1164.all ;
library modelsim_lib ;
use modelsim_lib.util.all ;
entity tbspy is
   -- no ports in test bench
end tbspy ;
architecture arc_tbspy of tbspy is
   component counter
      port ( resetN , clk : in     std_logic ;
             tc           : buffer std_logic ) ;
   end component ;
   signal resetN, clk, tc : std_logic ;
   signal cnt_des : std_logic_vector(7 downto 0) ;
begin
```

making libaray "modelsim_lib" & package "util" visible

A signal named "count" is an internal signal to this counter component

Destination object

Example of using the signal_spy utility

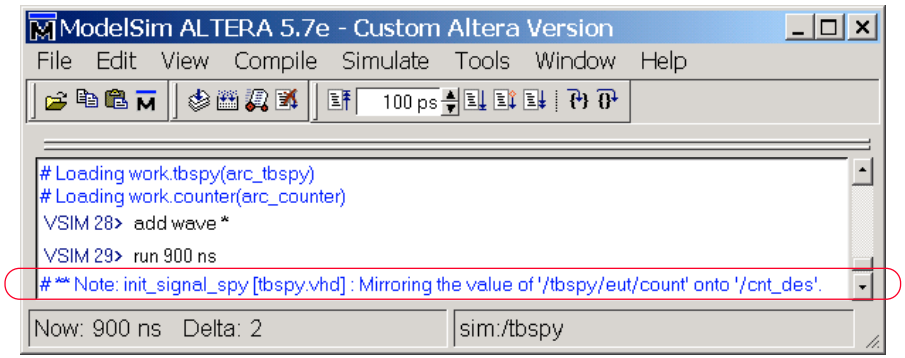
```
begin
   eut: counter port map ( resetN , clk , tc ) ;
   process
   begin
      clk <= '0' ;  wait for 50 ns ;
      clk <= '1' ;  wait for 50 ns ;
   end process ;
   resetN <= '0' , '1' after 100 ns ;
   -- a spy process recognized by modelsim (5.5 < ver)
   process
   begin           -- "eut/count" is ok too
      init_signal_spy("/tb/eut/count","/cnt_des") ;
      wait ;
   end process;
end arc_tbspy ;
```

path of spy object

Destination object

Outcome of a verbose activation

```
init_signal_spy( "/tb/eut/count","/cnt_des", 1 ) ;
```



Monitoring (Accessing) signals through hierarchies using external names

Probing

Accessing external names

- Access data objects (signals,constants & shared variables) directly and not through ports
- Full syntax:

```
<< object_class_type pathname : subtype_indicator >>
```

- *object_class_type* can be: signal, constant or shared variable. *pathname* can be: absolute, relative or start from package. *subtype_indicator*'s role is the same as it's role in alias declarations (can also be unconstrained)

- Example #1 (Absolute path)

```
ext_sig <= <<signal .tb.top_u.bot_u.int_sig : bit>> ;
```

- Example #2 (relative path - used in an assert statement):

```
assert << signal u_top.u_mid.u_bottom.count :
                std_logic_vector(3 downto 0) >> /= "1111"
 report "count is not legal" ;
```


Accessing external names (using alias)

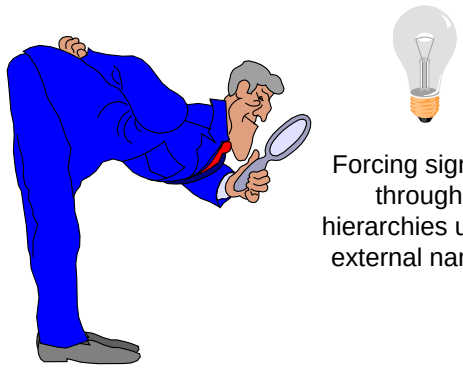
- Special path characters are: ^ - go up from current hierarchy ,
  @ - start absolute hierarchy from package
- Example #3 (pakage with hierarchy & start from package)

```
alias data_width is << constant @work.cpu_pack.alu_pack.width : natural >> ;
signal data_bus : std_logic_vector( data_width -1 downto 0) ;
```

- Example #4 using the alias command to make a compact name

```
alias count is <<signal .eut.u_top.cnt : std_logic_vector>> ;
assert count(3 downto 0) = "1001"
   report "got: "& to_string(count) & "/= 1001 expected" ;
```

- Results of Example #4 (when count is not 1001)
  Error: got: 0000 /= 1001 expected



Forcing signals through hierarchies using external names

Force & Release assignments

- Override all normal resolved values assigned to a signal (similar to script force command)
  - Initialize signals quickly bypassing complex initialization sequences
  - Inject erroneous values into design to ensure that detection & response are appropriate
- A verification feature
- Force Syntax:

```
signal_name <= force expression ;
```

signal_name can be a regular signal name but also can also be (and usually is) an external name or alias

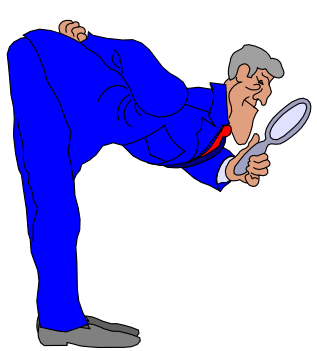
- Release syntax:

```
signal_name <= release ;
```

- Stop forcing & design takes back control

Force & Release assignment example

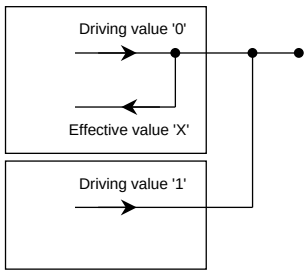
```
process
   use work.control_pkg.all ;
   alias clk is << signal eut.clk : std_logic >> ;
   alias present_state is
      << signal eut.sm.present_state :
                              std_logic_vector(3 downto 0) >>;
begin
   . . .
   -- force a bad (corrupt) state to state machine
   wait until falling_edge(clk) ;
   current_state <= force "1111" ; -- bad (corrupt) state
   wait until falling_edge(clk) ;
   current_state <= release ;
   -- monitor recovery activity
   . . .
end process ;
```

Forcing ports:
Driving value &
Effective value

Force & Release assignments of ports

- Driving value
  - The value presented externally by the entity
  - Determined by internal sources within the entity
- Effective value
  - The value seen internally by an entity
  - Determined by whatever is externally connected to port



VHDL-2008 allows us to force the driving and effective values of a port independently !

Force & Release assignments of ports

- VHDL-2008 allows us to force the driving and effective values of a port independently !
- Forcing driving value (can be done to any port of mode out or inout or buffer but can not be done to port of mode in)

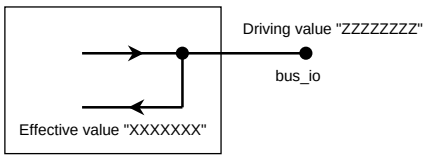
```
bus_io <= force out "ZZZZZZZZ" ;
```

- Forcing effective value (can be done to port of any mode)

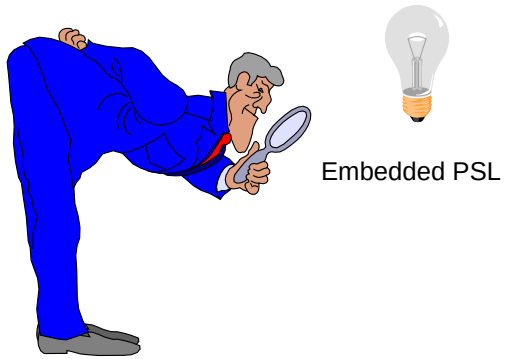
```
bus_io <= force in  "XXXXXXXX" ;
```

- Releasing previous ports

```
bus_io <= release out ;
bus_io <= release in ;
```



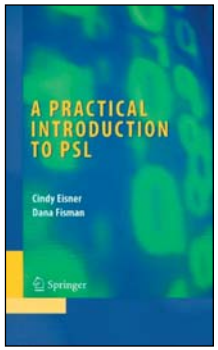
- Default forcing assignment of port signals of mode in is force in (forcing effective value)
- Default forcing assignment of port signals of mode inout,out,buffer is force out (forcing driving value)

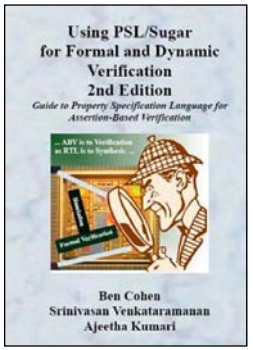


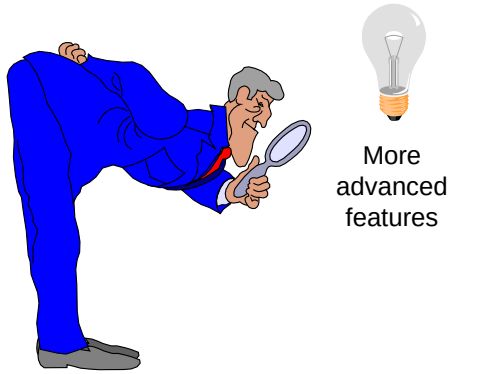
Embedded PSL

General Information on PSL

- PSL = Property Specification Language
- Origin: Sugar (IBM) => IEEE std 1850
- Allows specification of temporal properties of a model
  - Enables Static Verification (using formal proof tools)
  - Enables Dynamic Verification (using simulation checkers)
- For full information refer to:
  - *"Practical Introduction to PSL",* Cindy Eisner & Dana Fisman
  - *"Using PSL/Sugar for Formal and Dynamic Verification",* multiple authors







More advanced features

More advanced topics

- IP (Intellectual Property) protection
  - Pragma based approach
  - Allows IP authors to mark specific areas of VHDL code for encryption
  - using standard cipher algorithms (symmetric & asymmetric ciphers such as: DES,AES,blowfish,twofish,serpent, cast,RSA,elgama, and others)
  - using standard encoding methods (such as: uudecode, base64 & others)
  - Using standard hash functions (sha1,md5,md2,ripemod-160)
  - Support is embedded in standard simulation & Synthesis tools
- VHDL Procedural Interface (VHPI)
  - application-programming interface (API) to VHDL tools
  - Tools can gain acess about VHDL model during analysis, elaboration or execution of model
  - Can help development of add-in tools such as: Linters, Code coverage analyzer tools, Timing and Power analyzers
  - Can help usage of external simulation models

Open Source VHDL Verification Methodology

Coffee Break