4. Assigning Values
A_sl <= '1';  -- Character literal
B_slv <= "1111";  -- String literal
C_slv := X"3F";  -- hex. 4 bits per character
D_slv5 := '1' & X"F";  -- 5 bit object
E_slv5 := "5X1F";  -- VHDL-2008, 5 bit object
F_slv := (others => '1');  -- aggregate
G_slv(0) := '0';  -- index
H_slv(0 to 1) := "01";  -- slice
L_int <= 15;  -- universal integer
M_int <= 16#F#;  -- base 16 integer
N_bool <= TRUE;  -- Boolean only true or false

5. Common Attributes
signal A : unsig(7 downto 0);

Array Attribute  Result  Value / Meaning
A length        value     8
A range         range     3 downto 0
A reverse_range range     0 to 7
A left          value     7
A right         value     0

6. VHDL Operators
Logic and, or, and, nor, xor, xnor
Comp <=, /=, <, <=, >=
Shift sll, srl, sla, sra, rol,ror
Add +, -
Sign +, -
Mult *, /, mod, rem
Misc **, abs, not, and, or, nand, nor, xor, xnor

Precedence increases from logic to misc. Underlined items are VHDL-2008.

7. Strong Typing and Assignments
Size and type of target (right) must equal size and type of expression (left). For the array-based types, each operation has a specific sized result. VHDL operators allow multiple implementations for different types (overloading).

8. Logic operators = Logic Gates
Separate operators with parentheses when using different operators (except not) or non-associative operators (nand, nor).

Legal:  Illegal:
Z <= A and B and C ;
Z <= (A and B) or C ;
Z <= (A and B) and C ;
Z <= not A and B ;
Z <= (not A) and B ;

Array must be the same length. Arrays are handled bit-wise from left to right, independent of indices.

Overloading

<table>
<thead>
<tr>
<th>Left</th>
<th>Right</th>
<th>Return</th>
<th>Package</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>bool</td>
<td>bool</td>
<td>std</td>
<td></td>
</tr>
<tr>
<td>sul</td>
<td>sul</td>
<td>sul</td>
<td>1164</td>
<td>1 also sl</td>
</tr>
<tr>
<td>slv</td>
<td>slv</td>
<td>slv</td>
<td>1164</td>
<td></td>
</tr>
<tr>
<td>sulv</td>
<td>sulv</td>
<td>sulv</td>
<td>1164</td>
<td></td>
</tr>
<tr>
<td>uv</td>
<td>uv</td>
<td>uv</td>
<td>ns</td>
<td>not in sla</td>
</tr>
<tr>
<td>sv</td>
<td>sv</td>
<td>sv</td>
<td>ns</td>
<td>not in sla</td>
</tr>
</tbody>
</table>

* Not operator only has a right value.

9. General Numeric Overloading
The general sense of overloading for Comparison, Addition, and Multiplication is summarized below. The arrays must be the same type (either uv, sv, or slv).

10. Comparison

10.1 Comparisons Return Boolean
Use a conditional to create an appropriate value:

Z <= '1' when (A > B) else '0';

10.2 Comparisons only match identical values
Hence, comparing to '-' is invalid:

Dec1 <= '1' when Addr = "11---0" else '0';  --illegal

10.3 Overloaded Comparison Operators
When using packages ns, sla, and slu, array comparisons are treated numerically.

10.4 * Implicit Definitions
In addition to the general numeric overloading for sv, uv, and slv, the following overloading also exists:

<table>
<thead>
<tr>
<th>Left</th>
<th>Right</th>
<th>Return</th>
<th>Package</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>sul</td>
<td>sul</td>
<td>bol</td>
<td>implicit</td>
<td>1,*</td>
</tr>
<tr>
<td>sulv</td>
<td>sulv</td>
<td>bol</td>
<td>implicit</td>
<td></td>
</tr>
<tr>
<td>slv</td>
<td>slv</td>
<td>bol</td>
<td>implicit</td>
<td></td>
</tr>
</tbody>
</table>

For the ordering operators (<=, /=, <, <=, >=), the left most type values are smaller than right most values. Hence, implicitly for std_logic, '0' < '1' < 'L' < 'H'

Use type unsigned or signed, package slu, or functions TO_X01 to avoid this issue.

10.5 Std_Match = comparison with don't care
Std_match does comparisons and understands '1'. It is in package NS and is supported for sul, slv, sulv, uv, sv.

Dec1 <= '1' when std_match(Addr,"11---0") else '0';
11. Shift Operators
With VHDL-2008, shift operators are implemented for std_logic family of types.

\[
Y_{uv} \leftarrow A_{uv} \text{sl} 1; \\
Y_{slv} \leftarrow A(6\, \text{downto}\, 0) \& '0'; \quad \text{-- SLL 1} \\
Z_{uv} \leftarrow A_{uv} \text{rl} 1; \\
Z_{slv} \leftarrow '0' \& A(7\, \text{downto}\, 1); \quad \text{-- SRL 1} \\
SIR \leftarrow S1_{sl} \& A(7\, \text{downto}\, 1); \quad \text{-- Shift In} \\
\]

12. Addition
For addition and subtraction, always use the + and - operators (never gate logic). The size of the result equals the size of the largest array input. A short input array is extended appropriately for the type.

**Overloading beyond general numeric overloading**

<table>
<thead>
<tr>
<th>Function</th>
<th>Left</th>
<th>Right</th>
<th>Return</th>
<th>Package</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>sul</td>
<td>array</td>
<td>sla, ns, nsu</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>sul</td>
<td>array</td>
<td>sla, ns, nsu</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Facilitates Add with Carry or counter with count enable:

\[
Y_{uv} \leftarrow A_{uv} + B_{uv} + \text{Cin}_sl; \\
\text{IncVal} \leftarrow \text{IncReg} + \text{CountEn}; \\
\]

13. Multiplication
NS supports the general numeric overloading. SLA and slu only support array with array. For array with array, the size of result equals the sum of the size of the two inputs. Array with integer is not useful since the size of the result is 2X the array input.

14. Misc Operators: - abs

Function | In | Rtn | Pkg | Notes |
---------|----|-----|-----|-------|
-         | sv | sv  | ns, sla | 2       |
abs       | sv | sv  | ns, sla | 2       |

15. Summary of Result Sizes

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;10101010&quot;</td>
<td>number of bits in array</td>
</tr>
<tr>
<td>X&quot;AA&quot;</td>
<td>4 x number of characters</td>
</tr>
</tbody>
</table>
A_slv8     | A_slv8'length |
C5 & D3    | C5'length + D3'length |
A8 and B8  | D8'length (= E8'length) |
A8 > B8    | Boolean  |
A8 > 10    | Boolean |
A8 + B8    | maximum(A8'length, B8'length) |
A8 + 10    | A8'length |
A8 + B8    | A8'length + B8'length |
A8 * 10    | 2 * A8'length |

16. Notes for Overloaded Operators

16.1 std_logic
Since std_logic automatically converts to std_ulogic, there is no overloading for std_logic.

16.2 numeric_std and std_logic_arith
Do not use together in the same entity / package.

16.3 Use Integer Literals

\[
Z_{uv} \leftarrow A_{uv} + 16\#5F#; \\
\]

16.4 SLA: Signed & Unsigned Arguments
Std_logic_arith overloads addition, comparison, and multiplication to allow signed and unsigned operands in the same expression. Requires type qualifiers when using string literals. Does not apply to NS.

\[
Z_{uv} \leftarrow A_{uv} + \text{unsigned}('0101'); \\
\]

16.5 SLA: Std_logic_vector Return Values
sla overloads functions to return slv as well as unsigned or signed. Requires type qualifiers with sla and slu in the case shown below. Does not apply to NS and slu.

\[
Z_{slv} \leftarrow \text{signed}'(A_{sv} + B_{sv}) + \text{signed}'(C_{sv} + D_{sv}); \\
\]

17. Ambiguous Expressions
A statement is ambiguous if more than one operator symbol or function can match its arguments. VHDL type qualifiers (TypeName) are a mechanism that specifies the type of an argument or return value of a subprogram. See notes 15.4 and 15.5.

18. Conversions

18.1 Std_logic <= Signed(i), Unsigned(i)
Two objects that are subtypes of the same type covert automatically and do not need a conversion function:

\[
A_{sul} \leftarrow B_{sl}; \\
C_{sl} \leftarrow D_{sv}(1); \\
E_{uv}(1) \leftarrow F_{sl}; \\
G_{sv}(1) \leftarrow H_{uv}(1); \\
\]

18.2 Signed, Unsigned <= Std_logic_vector
Arrays with a common base type and indices that have a common base type can be converted by type casting:

\[
A_{slv} \leftarrow \text{std_logic_vector}(B_{uv}); \\
C_{slv} \leftarrow \text{std_logic_vector}(D_{sv}); \\
G_{uv} \leftarrow \text{unsigned}(H_{slv}); \\
J_{sv} \leftarrow \text{signed}(K_{slv}); \\
\]

18.3 Unsigned, Signed <= Integer
Converting between either signed or unsigned and integer requires a conversion function:

Function | In | Rtn | Pkg |
---------|----|-----|-----|
to_integer(val) | uv | int0+ | nsu |
to_integer(val) | sv | int  | ns  |
to_integer(val) | slv | int  | ns  |
to_unsigned(val, len) | int0+ | int0+ | uv  |
to_signed(val, len) | int, int0+ | sv | ns  |
to_slv(val, len) | int0+ | int0+ | slv | nsu |
conv_integer(val) | uv | int  | sla |
conv_integer(val) | sv | int  | sla |
conv_unsigned(val, len) | int, int | uv | sla |
conv_signed(val, len) | int, int | sv | sla |

18.4 std_logic_vector <= Integer
\[
A_{slv} \leftarrow \text{std_logic_vector}(\text{to_unsigned}(B_{int}, 8)); \\
C_{int} \leftarrow \text{to_integer}(\text{unsigned}(D_{slv})); \\
\]
Alternately use conversions in Numeric_std_unsigned.

18.5 Resizing Arrays
Resizing can be done manually, with operator overloading (when only one operand is smaller), or with the following functions:

Function | In | Rtn | Pkg |
---------|----|-----|-----|
resize(val, len) | uv | int0+ | uv  |
resize(val, len) | sv | int0+ | sv  |
conv_unsigned(val, len) | uv | int | uv  |
conv_signed(val, len) | uv | int | sv  |

19. Strength Strippers
Strength strippers are used in behavioral models and IO pads to map values of a type to a simpler set (X, 0, 1). Strength strippers TO_X01, TO_X01Z, TO_UX01 convert the input values to X01, X01Z, and UX01 respectively. Maps: 'H' to '1', 'L' to '0' and others to 'X'.

Function | In | Rtn | Pkg |
---------|----|-----|-----|
to_01    | s  | 0   | 1164|
conv_integer | s, xmap := '0' | array | sl  |

With VHDL-2008, array is slv, sulv, sv, uv, ...

Strength stripper TO_01 maps: 'H' to '1', 'L' to '0' and others to XMAP value.

Function | In | Rtn | Pkg |
---------|----|-----|-----|
to_01    | s  | 0   | 1164|

With VHDL-2008, array is slv, sulv, sv, uv, ...

20. X detection
The function is_X detects 'X' on inputs of models.

Function | In | Rtn | Pkg |
---------|----|-----|-----|
to_01    | s  | 0   | 1164|

With VHDL-2008, array is slv, sulv, sv, uv, ...

21. Edge detection
Functions rising_edge and falling_edge.

Function | In | Rtn | Pkg |
---------|----|-----|-----|
to_01    | s  | 0   | 1164|

http://www.SynthWorks.com     jim@SynthWorks.com  
VHDL Hardware Synthesis and Verification Training  
11898 SW 128th Ave.  Tigard OR 97223  
(800)-505-8435  
Reproduction of entire document in whole permitted. All other rights reserved.