Enhancements to VHDL's Packages

by

Jim Lewis
Director of Training, SynthWorks Design Inc
Jim@SynthWorks.com

http://www.SynthWorks.com

Enhancements to VHDL's Packages

- IEEE 1076.3 (numeric_std) and IEEE 1164 (std_logic_1164).
  - Logic reduction operators
  - Array / scalar logic operators
  - Array / scalar addition operators
  - TO_X01, IS_X for unsigned and signed
  - Shift operators for std_logic_vector and std_ulogic_vector
  - Unsigned arithmetic for std_logic_vector & std_ulogic_vector
  - TextIO for std_logic_1164 and numeric_std
  - Floating point arithmetic
- Review of strong typing, overloading, type conversions

Caution:
Work in progress. Some items presented may change
Logic Reduction Operators

- For std_logic_vector, std_ulogic_vector, unsigned and signed.

- Possible forms (being coordinated with VHDL-200X*):

```vhdl
function and_reduce(arg : std_logic_vector) return std_ulogic;
function and(arg : std_logic_vector) return std_ulogic;
```

- Calculating Parity with reduction operators:

```vhdl
Parity <= xor Data;
```

- Calculating Parity without reduction operators:

```vhdl
Parity <= Data(7) xor Data(6) xor Data(5) xor Data(4)
          Data(3) xor Data(2) xor Data(1) xor Data(0);
```

- *VHDL-200X may fasttrack overloading unary logic operators.

Array / Scalar Logic Operators

- Proposal: Create symmetric overloading for bit_vector*, std_logic_vector, std_ulogic_vector, unsigned, and signed

```vhdl
function "and"( l : std_logic_vector; r : std_ulogic )
return std_logic_vector;
function "and"( l : std_ulogic; r : std_logic_vector )
return std_logic_vector;
```

- * Proposed as a VHDL-200X activity

- Application

```vhdl
signal ASel, BSel, CSel, DSel : std_logic;
signal Y, A, B, C, D : std_logic_vector(3 downto 0);
  . . .
Y <= (A and ASel) or (B and BSel) or
    (C and CSel) or (D and DSel);
```
Array / Scalar Logic Operators

What is the hardware implication of the following:

```vhdl
signal ASel : std_logic;
signal T, A : std_logic_vector(3 downto 0);
. . .
T <= (A and ASel);
```

The value of ASel will replicated to form an array.

- When ASel = '0', value expands to "0000"
- When ASel = '1', value expands to "1111"

Array / Scalar Addition Operators

Mix an array with a scalar for unsigned and signed addition.

```vhdl
function +"(L: unsigned; R: std_ulogic) return unsigned;
function +"(L: std_ulogic; R: unsigned) return unsigned;
```

```vhdl
signal Cin : std_logic;
signal A, B : unsigned(7 downto 0);
signal Y : unsigned(8 downto 0);
. . .
Y <= A + B + Cin;
```

The value of Cin will be expanded to be "0" & Cin and typed appropriately:
- When Cin = '0', value expands to "0000"
- When Cin = '1', value expands to "0001"
TO_X01, TO_X01Z, TO_UX01, and IS_X

- For types unsigned and signed

  ```
  function To_X01 ( s : unsigned ) return unsigned;
  function Is_X ( s : unsigned ) return boolean;
  ```

- Motivation: consistency with std_logic_1164

Shift Operators

- For std_logic_vector and std_ulogic_vector

  ```
  function "sll" ( l : std_logic_vector; r : integer )
  return std_logic_vector;

  function "sll" ( l : std_ulogic_vector; r : integer )
  return std_ulogic_vector;
  ```
Unsigned Arithmetic for std_logic_vector & std_ulogic_vector

- Add new package named numeric_std_unsigned?

- Rationale: Testbench code

```vhdl
constant CHIP1_RAM_BASE : std_logic_vector(31 downto 0)
  := X"40000000";
constant ZERO_DATA : std_logic_vector(31 downto 0)
  := (others => '0');
  . .
for i in 0 to 31 loop
  CpuWrite(CpuRec, CHIP1_RAM_BASE + i, ZERO_DATA + 2**i);
end loop;
```

- Note: requires minor VHDL LRM update to make this "LRM-legal". VHDL ISAC is currently working on this.

TextIO

- IEEE 1076.3 (numeric_std) and IEEE 1164 (std_logic_1164).
  - Synopsys has donated the package std_logic_textio, which is planned to be updated in a compatible fashion to become the standard.

- Additional functionality: Overloaded String functions

  ```vhdl```
  To_string(now, right, 12)
  ```vhdl```

- Usage (with VHDL's built-in write):

  ```vhdl```
  write(Output, "%%%ERROR data value miscompare." & LF &
    "  Actual data value = " & to_hstring(Data) & LF &
    "  Expected data value = " & to_hstring(ExpData) & LF &
    "  at time: " to_string(now, right, 12) );
  ```vhdl```
Floating Point

- Addition of a family of packages
- See David Bishop's paper later in the conference for details.
  - Session 6.2

Review

- Strong Typing
- Overloading
- Ambiguous Expressions
- Type Conversions
**Strong Typing**

- **Size and type of target** (left) = size and type of expression (right)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Size of Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y &lt;= A ;</td>
<td>A'Length</td>
</tr>
<tr>
<td>Y &lt;= A and B ;</td>
<td>A'Length = B'Length</td>
</tr>
<tr>
<td>W &lt;= A &gt; B ;</td>
<td>Boolean</td>
</tr>
<tr>
<td>Y &lt;= A + B ;</td>
<td>Maximum (A'Length, B'Length)</td>
</tr>
<tr>
<td>Y &lt;= A + 10 ;</td>
<td>A'Length</td>
</tr>
<tr>
<td>V &lt;= A * B ;</td>
<td>A'Length + B'Length</td>
</tr>
</tbody>
</table>

- Strong typing is like an assertion check for expressions:

```vhdl
signal A8, B8, Result8 : std_logic_vector(7 downto 0);
signal Result9  : std_logic_vector(8 downto 0);
signal Result7  : std_logic_vector(6 downto 0);

Result8 <= A8 + B8 ;
Result9 <= ('0' & A8) + ('0' & B8) ;
Result7 <= A8(6 downto 0) + B8(6 downto 0) ;
```

**Overloading**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Left</th>
<th>Right</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic</td>
<td>TypeA</td>
<td>TypeA</td>
<td>TypeA</td>
</tr>
<tr>
<td>Numeric</td>
<td>Array</td>
<td>Array</td>
<td>Array*</td>
</tr>
<tr>
<td></td>
<td>Array</td>
<td>Integer</td>
<td>Array*</td>
</tr>
<tr>
<td></td>
<td>Integer</td>
<td>Array</td>
<td>Array*</td>
</tr>
<tr>
<td>Logic, Addition</td>
<td>Array</td>
<td>Std_ulogic</td>
<td>Array</td>
</tr>
<tr>
<td></td>
<td>Std_ulogic</td>
<td>Array</td>
<td>Array</td>
</tr>
<tr>
<td>Logic Reduction</td>
<td>Array</td>
<td>Std_ulogic</td>
<td>Std_ulogic</td>
</tr>
</tbody>
</table>

**Notes:**
- **Array** = std_logic_vector, std_logic_vector, bit_vector, unsigned, signed,
- **TypeA** = boolean, std_logic, std_ulogic, Array
- For **Array** and **TypeA**, arguments must be the same.
- * for comparison operators the result is boolean
Ambiguous Expressions

- An expression / statement is ambiguous if more than one operator symbol or subprogram can match its arguments.

- **Std.Logic.Arith** defines the following two functions:

  ```
  function "+" (L, R: SIGNED) return SIGNED;
  function "+" (L: SIGNED; R: UNSIGNED) return SIGNED;
  ```

- The following expression is ambiguous and an error:

  ```
  Z_sv <= A_sv + "1010" ;
  ```

- Issues typically only arise when using literals.

- How do we solve this problem?

Std.Logic.Arith:
Ambiguous Expressions

- VHDL type qualifier (**type_name'**) is a mechanism that specifies the type of an operand or return value of a subprogram (or operator).

  ```
  Z_sv <= A_sv + signed'("1010") ;
  ```

- Leaving out the ' is an error:

  ```
  -- Z_sv <= A_sv + signed("1010") ;
  ```

- Without ', it is type casting. Use type casting for:

  ```
  Z_sv <= A_sv + signed(B_slv) ;
  ```

- Recommended solution, use integer:

  ```
  Z_sv <= A_sv - 6 ;
  ```
Type Conversions

- VHDL is dependent on overloaded operators and conversions
- What conversion functions are needed?
  - Signed & Unsigned (elements) <=> Std_Logic
  - Signed & Unsigned <=> Std_Logic_Vector
  - Signed & Unsigned <=> Integer
  - Std_Logic_vector <=> Integer
- VHDL Built-In Conversions
  - Automatic Type Conversion
  - Conversion by Type Casting
  - Conversion functions located in Numeric_Std

Automatic Type Conversion: Unsigned, Signed <=> Std_Logic

- Two types convert automatically when both are subtypes of the same type.
  - subtype std_logic is resolved std_ulogic;
- Converting between std_ulogic and std_logic is automatic
- Elements of Signed, Unsigned, and std_logic_vector = std_logic
- Elements of these types convert automatically to std_ulogic or std_logic

Legal Assignments

\[
\begin{align*}
A_{sl} & \leq J_{uv}(0) \\
B_{sul} & \leq K_{sv}(7) \\
L_{uv}(0) & \leq C_{sl} \\
M_{slv}(2) & \leq N_{sv}(2)
\end{align*}
\]

Implication:

\[
Y_{sl} \leq A_{sl} \text{ and } B_{sul} \text{ and } J_{uv}(2) \text{ and } K_{sv}(7) \text{ and } M_{slv}(2);
\]
Type Casting:
Unsigned, Signed <=> Std_Logic_Vector

- Use type casting to convert equal sized arrays when:
  - Elements have a common base type (i.e. std_logic)
  - Indices have a common base type (i.e. Integer)

- Unsigned, Signed <=> Std.Logic_Vector

  ```
  A_slv <= std_logic_vector( B_uv  ) ;
  C_slv <= std_logic_vector( D_sv  ) ;
  G_uv  <= unsigned( H_slv ) ;
  J_sv  <= signed( K_slv ) ;
  ```

- Motivation, Unsigned - Unsigned = Signed?

  ```
  signal X_uv, Y_uv  : unsigned (6 downto 0) ;
  signal Z_sv        : signed   (7 downto 0) ;
  . . .
  Z_sv <= signed('0' & X_uv) - signed('0' & Y_uv) ;
  ```

Numeric_Std Conversions:
Unsigned, Signed <=> Integer

- Converting to and from integer requires a conversion function.

  - Unsigned, Signed => Integer

    ```
    Unsigned_int  <= TO_INTEGER ( A_uv ) ;
    Signed_int    <= TO_INTEGER ( B_sv ) ;
    ```

  - Integer => Unsigned, Signed

    ```
    C_uv <= TO_UNSIGNED ( Unsigned_int, 8 ) ;
    D_sv <= TO_SIGNED ( Signed_int, 8 ) ;
    ```

- Motivation (indexing an array of an array):

  ```
  Data_slv <= ROM( TO_INTEGER( Addr_uv ) ) ;
  ```
Supporting Standards

- VHDL is an IEEE standard
- It is your right and responsibility to participate
- Join
  - IEEE
  - DASC: see [http://dasc.org](http://dasc.org)
  - VASG: see [http://www.eda.org/vasg](http://www.eda.org/vasg)
  - Accellera: see [http://www.accellera.org](http://www.accellera.org)
  - VHDL-200x: see [http://www.eda.org/vhdl-200x](http://www.eda.org/vhdl-200x)

Vendor Support of Standards

EDA vendor support of standards is not as simple as it may seem. For EDA vendors, supporting a standard is an investment. Hence, feature support is market driven. They don't support new features based on merit, they support them based on user requests.

As a result, if you see new features in a standard that you would like to use, make sure to request that your EDA vendor support the feature.
Author Biography

Jim Lewis, Director of Training, SynthWorks Design Inc.

Jim Lewis, the founder of SynthWorks, has seventeen years of design, teaching, and problem solving experience. In addition to working as a Principal Trainer for SynthWorks, Mr. Lewis does ASIC and FPGA design, custom model development, and consulting. Mr. Lewis is an active member of IEEE Standards groups including, VHDL (IEEE 1076), RTL Synthesis (IEEE 1076.6), Std_Logic (IEEE 1164), and Numeric_Std (IEEE 1076.3). Mr. Lewis can be reached at 1-503-590-4787, jim@SynthWorks.com, or http://www.SynthWorks.com

SynthWorks VHDL Training

Comprehensive VHDL Introduction 4 Days
http://www.synthworks.com/comprehensive_vhdl_introduction.htm
A design and verification engineers introduction to VHDL syntax, RTL coding, and testbenches.
Our designer focus ensures that your engineers will be productive in a VHDL design environment.

VHDL Coding Styles for Synthesis 4 Days
http://www.synthworks.com/vhdl_rtl_synthesis.htm
Engineers learn RTL (hardware) coding styles that produce better, faster, and smaller logic.

VHDL Testbenches and Verification 3 days
http://www.synthworks.com/vhdl_testbench_verification.htm
Engineers learn how create a transaction-based verification environment based on bus functional models.

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