VHDL 200X: The Future of VHDL

By Jim Lewis
SynthWorks VHDL Training
Team Leader VHDL-200X Fast Track
jim@synthworks.com
VHDL-200X: The Future of VHDL

- Currently the work is being funded through Accellera membership and the work is being transitioned to an Accellera VHDL Working Group.

- VHDL-200X is a multi-phased effort.
  - In the first phase, the Accellera WG will be considering the IEEE VHDL-200X-FT work plus some additional testbench enhancements.

- Goals:
  - Maintain VHDL style, nature, and backward compatibility
  - Leverage Industry Efforts (PSL, SystemVerilog, …)

**Caution:** Changes presented here are a work in progress
VHDL-200X-FT: Proposals

- PSL
- IP Protection
- Type Generics
- Generics on Packages
- Context Unit
- Arrays with unconstrained arrays
- Records with unconstrained arrays
- Fixed Point Packages
- Floating Point Packages
- Process(all)
- Case Statement Updates
- Don't Care in a Case Statement
- Conditional Expression Updates
- Expressions in port maps
- Read out ports
- Stop and Finish
- Conditional and Selected assignment in sequential code
- If Expressions
- hwrite, owrite, … hread, oread
- to_string, to_hstring, …
- Sized bit string literals
- Unary Reduction Operators
- Array/Scalar Logic Operators
- Hierarchical references of signals
- Slices in array aggregates
- Std_logic_1164 Updates
- Numeric_Std Updates

Much of VHDL's cumbersome syntax issues can be fixed
PSL

- PSL will be incorporated directly into VHDL

- Implications
  - PSL Vunits are a VHDL Design Unit
  - PSL declarations (properties) can go in:
    - Packages
    - Declarative regions of entity, architecture, and block.
  - PSL directives (assert, cover, …) are VHDL statements
    - Can be placed in any concurrent statement part.

**Note:** PSL code will not need to be placed in comments
package MuxPkg is
    generic( type array_type) ;

    function Mux4 (  
        Sel : std_logic_vector(1 downto 0);  
        A   : array_type ;  
        B   : array_type ;  
        C   : array_type ;  
        D   : array_type 
    ) return array_type ;  
end MuxPkg ;

package body MuxPkg is  
    . . .
end MuxPkg ;
Type Generics + Generics on Packages

- Making the Mux4 function available for both std_logic_vector and unsigned.

```vhdl
library ieee;
package MuxPkg_slv is new work.MuxPkg
  Generic map (    
    array_type => ieee.std_logic_1164.std_logic_vector
  ) ;

library ieee;
package MuxPkg_unsigned is new work.MuxPkg
  Generic map (    
    array_type => ieee.numeric_std.unsigned
  ) ;
```
Arrays of Unconstrained Arrays

```vhdl

type std_logic_matrix is array (natural range <>)
of std_logic_vector;

-- constraining in declaration
signal A : std_logic_matrix(7 downto 0)(5 downto 0);

entity e is
port (   A : std_logic_matrix(7 downto 0)(5 downto 0);
    . . .
);
```

```
Records of Unconstrained Arrays

type complex is record
    a : std_logic ;
    re : signed ;
    im : signed ;
end record ;

-- constraining in declaration
signal B : complex (re(7 downto 0), im(7 downto 0)) ;
Fixed Point Types

- Definitions in package, ieee.fixed_pkg.all

```vhdl
type ufixed is array (integer range <>) of std_logic;
type sfixed is array (integer range <>) of std_logic;
```

- For downto range, whole number is on the left and includes 0.

```vhdl
constant A : ufixed (3 downto -3) := "0011010000" ;
3210 -3
IIIIFFF
0110100 = 0110.100 = 6.5
```

- Math is full precision math:

```vhdl
signal A, B : ufixed (3 downto -3) ;
signal Y : ufixed (4 downto -3) ;
...
Y <= A + B ;
```
Floating Point Types

- Definitions in package, ieee.fphdl_pkg.all

```vhdl
type fp is array (integer range <>) of std_logic;
```

- Format is Sign Bit, Exponent, Fraction

```vhdl
signal A, B, Y : fp (8 downto -23) ;
8 76543210 12345678901234567890123
S EEEEEEEE FFFFFFFFFFFFFFFFFFFFFFFF

E = Exponent has a bias of 127
F = Fraction has an implied 1 in leftmost bit

0 10000000 00000000000000000000000 = 2.0
0 10000001 10100000000000000000000 = 6.5
0 01111100 00000000000000000000000 = 0.125 = 1/8

Y <= A + B ; -- FP numbers must be same size
Hierarchical Reference *

- For the near term use a package based approach

```haskell
Procedure Drive(
    source: in string;
    target: in string;
    force_mode: in force_mode_type := DEFAULT;
    verbose: in boolean := FALSE)
);

Procedure Probe(
    observer: in string;
    source: in string;
    verbose: in boolean := FALSE)
);
```

- * Inspired by donations from Mentor (Signal Spy) and Cadence (NCMirror)
Hierarchical Reference

```vhdl
procedure force (  
    target        : in string ;  
    value         : in string ;  
    delay         : in time := 0 ns ;  
    delay_mode    : in delay_mode_type := TRANSPORT_DELAY ;  
    force_mode    : in force_mode_type := DEFAULT ;  
    cancel_period : in time := -1 fs ;  
    verbose       : in boolean := FALSE  
) ;

procedure release (  
    target        : in string ;  
    keep_value    : in boolean := FALSE ;  
    verbose       : in boolean := FALSE  
) ;

function get_value(  
    target        : in string ;  
    verbose       : in boolean := FALSE  
) return string ;
```
Simplified Case Statement

- Allow locally static expressions to contain:
  - implicitly defined operators that produce composite results
  - operators and functions defined in std_logic_1164, numeric_std, and numeric_unsigned.

```vhdl
constant ONE1    : unsigned := "11" ;
constant CHOICE2 : unsigned := "00" & ONE1 ;
signal A, B      : unsigned (3 downto 0) ;

process (A, B)
begin
  case A xor B is
    when "0000"      =>    Y <= "00" ;
    when CHOICE2     =>    Y <= "01" ;
    when "0110"      =>    Y <= "10" ;
    when ONE1 & "00" =>    Y <= "11" ;
    when others      =>    Y <= "XX" ;
  end case ;
end process ;
```
Simplified Case Statement

- Although concatenation is specifically allowed, some cases will still require a type qualifier.

```vhdl
signal A, B, C, D : std_logic;
...

process (A, B, C, D)
begin
  case std_logic_vector'(A & B & C & D) is
    when "0000" =>   Y <= "00" ;
    when "0011" =>   Y <= "01" ;
    when "0110" =>   Y <= "10" ;
    when "1100" =>   Y <= "11" ;
    when others =>   Y <= "XX" ;
  end case ;
end process ;
```
Case With Don't Care

- Allow use of '-' in targets provided targets are non-overlapping

```vhdl
-- Priority Encoder
process (Request)
begin
    case Request is
    when "1---" => Grant <= "1000" ;
    when "01--" => Grant <= "0100" ;
    when "001-" => Grant <= "0010" ;
    when "0001" => Grant <= "0001" ;
    when others => Grant <= "0000" ;
    end case ;
end process ;
```

**Note:** Only '-' in the case target is treated as a don't care. A '-' in the case? Expression will not be treated as a don't care.
Simplified Conditional Expressions

- Applies to conditional of: if, elsif, wait until, when, while

```vhdl
if (Cs1 and not nCs2 and Cs3 and Addr=X"A5") then
if (Cs1 and nCs2='0' and Cs3 and Addr=X"A5") then
if (not nWe) then
  Sel <= Cs1 and not nCs2 and Cs3 ;
```

- Active low is represented with "not nCs2"

- Backward compatible with current VHDL syntax:

```vhdl
if nWe = '0' then
if (Cs1='1' and nCs2='0' and Addr=X"A5") then
if ((Cs1 and not nCs2)='1' and Addr=X"A5") then
```
Simplified Conditional Expressions

- Implementation Part 1:
  If top level of a conditional expression is either bit or std_ulogic, then call the function condition? to do the conversion.
    - This makes the following legal:
      ```
      if (Cs1 and not nCs2 and Cs3) then
      ```

- Implementation Part 2:
  Overload logic operators so if boolean used with bit/std_ulogic, then the result will be bit/std_ulogic
    - Maintains accuracy by promoting true to '1' and false to '0'
    - This makes the following legal:
      ```
      if (Cs1 and not nCs2 and Cs3 and Addr=X"A5") then
      DevSel1 <= Cs1 and not nCs2 and Cs3 and Addr=X"A5" ;
      ```
Process (all)

- Creates a sensitivity list with all signals on sensitivity list

```vhdl
Mux3_proc : process(all)
begin
  case MuxSel is
    when "00" =>      Y <= A ;
    when "01" =>      Y <= B ;
    when "10" =>      Y <= C ;
    when others =>    Y <= 'X' ;
  end case ;
end process
```

- Benefit: Reduce mismatches between simulation and synthesis
Hwrite, Hread, Owrite, Oread

- Support Hex and Octal read & write for all bit based array types

```pascal
procedure hwrite (            Buf             : inout Line ;      VALUE           : in integer;      JUSTIFIED       : in SIDE  := RIGHT;      FIELD           : in WIDTH := 0  ) ;

procedure hread (            Buf             : inout Line ;      VALUE           : out integer;      Good            : out boolean)

procedure oread ( . . . ) ;
procedure owrite ( . . . ) ;
```

- No new packages. Supported in base package
  - For backward compatibility, std_logic_textio will be empty
To_String, To_HString, To_OString

- Create to_string for all types.
- Create hex and octal functions for all bit based array types

```vhdl
function to_string ( 
  VALUE : in std_logic_vector; 
  JUSTIFIED : in SIDE := RIGHT; 
  FIELD : in WIDTH := 0 
) return string;
function to_hstring ( ) return string;
function to_ostring ( ) return string;
```

- Formatting Output with Write (not write from TextIO):

```vhdl
write(Output, "%%%ERROR data value miscompare." & NL & "  Actual value = " & to_hstring (Data) & NL & "  Expected value = " & to_hstring (ExpData) & NL & "  at time:  " & to_string (now, right, 12));
```
Sized Bit String Literals

- Currently hex bit string literals are a multiple of 4 in size
  
  \[
  \text{X"AA"} = \text{"10101010"}
  \]

- Allow specification of size (and decimal bit string literals):
  
  \[
  \begin{align*}
  \text{7X"7F"} & = \text{"1111111"} \\
  \text{7D"127"} & = \text{"1111111"}
  \end{align*}
  \]

- Allow specification of signed vs unsigned (extension of value):
  
  \[
  \begin{align*}
  \text{9UX"F"} & = \text{"000001111"} \quad \text{Unsigned 0 fill} \\
  \text{9SX"F"} & = \text{"111111111"} \quad \text{Signed: left bit = sign} \\
  \text{9X"F"} & = \text{"000001111"} \quad \text{Defaults to unsigned}
  \end{align*}
  \]

- Allow Replication of X and Z
  
  \[
  \begin{align*}
  \text{7X"XX"} & = \text{"XXXXXXXX"} \\
  \text{7X"ZZ"} & = \text{"ZZZZZZZ"}
  \end{align*}
  \]
Signal Expressions in Port Maps

```
U_UUT : UUT
    port map ( A, Y and C, B) ;
```

- Needed to avoid extra signal assignments with OVL

- If expression is not a single signal, constant, or does not qualify as a conversion function, then
  - convert it to an equivalent concurrent signal assignment
  - and it will incur a delta cycle delay
Read Output Ports

- Read output ports
  - Value read will be locally driven value

- Assertions need to be able to read output ports
Allow Conditional Assignments for Signals and Variables in Sequential Code

- Statemachine code:

```
if (FP = '1') then
    NextState <= FLASH ;
else
    NextState <= IDLE ;
end if ;
```

- Simplification (new part is that this is in a process):

```
NextState <= FLASH when (FP = '1') else IDLE ;
```

- Also support conditional variable assignment:

```
NextState := FLASH when (FP = '1') else IDLE ;
```
Allow Selected Assignments for Signals and Variables in Sequential Code

```vhdl
signal A, B, C, D, Y : std_logic;
signal MuxSel : std_logic_vector(1 downto 0);

Process(clk)
begin
  wait until Clk = '1';
  with MuxSel select
    Mux :=
      A when "00",
      B when "01",
      C when "10",
      D when "11",
      'X' when others;
    Yreg <= nReset and Mux;
end process;
```
IF Expressions

- Similar to conditional signal assignment ...

\[
Y \leq \ A \ and \ B \ if \ S = '1', \ C \ and \ D ; \\
Y \leq \ (A \ and \ B) \ if \ S = '1', \ (C \ and \ D) ;
\]

- ... except it is an expression:

\[
Y \leq \ A \ and \ (B \ if \ S = '1', \ C) \ and \ D ;
\]

- And it can be used anywhere an expression can:

\[
\text{Signal A : integer := 7 if GEN\_VAL = 1, 15 ;} \\
\text{with MuxSel select} \\
\text{Y \leq (A \ if \ Asel='1', B) \ when \ '0',} \\
\text{\ (C \ if \ Csel='1', D) \ when \ '1',} \\
\text{'X' \ when \ others ;}
\]
Unary Reduction Operators

- Define unary AND, OR, XOR, NAND, NOR, XNOR

```
function "and" ( anonymous: BIT_VECTOR) return BIT;
function "or"   ( anonymous: BIT_VECTOR) return BIT;
function "nand" ( anonymous: BIT_VECTOR) return BIT;
function "nor" ( anonymous: BIT_VECTOR) return BIT;
function "xor"  ( anonymous: BIT_VECTOR) return BIT;
function "xnor" ( anonymous: BIT_VECTOR) return BIT;
```

- Calculating Parity with reduction operators:

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

- Calculating Parity without reduction operators:

```
Parity <= Data(7) xor Data(6) xor Data(5) xor Data(4) xor Data(3) xor Data(2) xor Data(1) xor Data(0) ;
```
Array / Scalar Logic Operators

- Overload logic operators to allow:

```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 Logic Operators

- A common application is to data read back logic

```
signal Sel1, Sel2, Sel3, Sel4 : std_logic;
signal DO, Reg1, Reg2, Reg3, Reg4 : std_logic_vector(3 downto 0);
...
DO <= (Reg1 and Sel1) or (Reg2 and Sel1) or (Sel3 and Reg3) or (Sel4 and Reg4);
```
Slices in Array Aggregates

- Allow slices in an Array Aggregate

```vhdl
Signal A, B, Y : unsigned (7 downto 0);
signal CarryOut : std_logic;

... (CarryOut, Y) <= ('0' & A) + ('0' & B);
```

- Currently, this would have to be written as:

```vhdl
(CarryOut, Y(7), Y(6), Y(5), Y(4), Y(3), Y(2), Y(1), Y(0))
<= ('0' & A) + ('0' & B);
```
Std Logic_1164 Updates

- Goals: Enhance current std_logic_1164 package

- A few items on the list are:
  - Uncomment xnor operators
  - Add shift operators for vector types
  - Add logical reduction operators
  - Add array/scalar logical operators
  - Match Function
  - Provide text I/O package for standard logic (similar to Synopsys' std_logic_textio)

- See also DVCon 2003 paper, "Enhancements to VHDL's Packages" which is available at:
  http://www.synthworks.com/papers
Numeric Std Updates

Goals:
- Enhance current numeric_std package.
- Unsigned math with std_logic_vector/std_ulo_logic_vector

A few items on the numeric_std list are:
- Logic reduction operators
- Array / scalar logic operators
- Array / scalar addition operators
- TO_X01, IS_X for unsigned and signed
- TextIO for numeric_std
# Resulting Operator Overloading

<table>
<thead>
<tr>
<th>Operator</th>
<th>Left</th>
<th>Right</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic TypeA</td>
<td>TypeA</td>
<td>TypeA</td>
<td>TypeA</td>
</tr>
<tr>
<td>Numeric Array</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_ulogic_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
Stop and Finish

- Add procedures STOP and FINISH:
  - STOP - Stops like breakpoint
  - FINISH - Terminate the simulation

```verbatim
procedure STOP;
procedure FINISH;
```

- Benefit:
  - Just stop. No annoying messages.
IP Protection and Encryption

- A pragma-based approach

- Allows IP authors to mark specific areas of VHDL code for encryption using standard algorithms.

The proposal:

- Defines constructs to demarcate protected envelopes in VHDL source code.
- Defines keywords to specify algorithms and keys.

- Tools that work with encrypted IP must not reveal any details through any interface or output it generates.
  - For example, a synthesis tool should generate an encrypted netlist for any portion of a design that is encrypted.
Context Unit = Primary Design Unit

- Allows a group of packages to be referenced by a single name

```vhdl
Context project1_Ctx is
  use std.textio.all;
  library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;
  use ieee.fixed_pkg.all;
end;
```

- Reference the named context unit

```vhdl
Library Lib_P1;
  context Lib_P1.project1_Ctx;
```

- Benefit increases as additional standard packages are created
  - Fixed Point, Floating Point, Assertion Libraries, . . .
VHDL-200X, Summary

● Fast Track work is the first phase of VHDL-200X
  ● Expected completion of current work is mid-2006.

● Next Steps are to add:
  ● Verification Data Structures such as
    ● associative arrays, queues, FIFOs, and memories
  ● Object Orientation
  ● Constrained Random Stimulus Generation
  ● Direct C and Verilog/SystemVerilog Calls

● Goal = HDVL: Hardware Description and Verification Language
  ● Full verification capabilities in one consistent language

● VHDL-200X = 200 X better than …  

    cont
VHDL-200X: Summary

- We need your help:
  - Participate! Don't sit on the bench and wait and watch.
    - See http://www.accellera.org/activities/vhdl/
  - Ask your colleagues and vendors to participate
  - Tell your vendors about features you want supported.
    - Be specific and prioritize your requests