VHDL-200X: The Future of VHDL

Jim Lewis, SynthWorks Design Inc. Jim@SynthWorks.com

Abstract

The VHDL-200X revision effort is currently finalizing its Fast Track phase. This paper summarizes the new features proposed to be released as part of this effort.

VHDL-200X is a comprehensive revision with a goal to enhance VHDL to improve performance, modeling capability, ease of use, verification capability, simulation control, and the type system. At the same time, it strives to maintain VHDL style, nature, and backward capability. For many features the plan is to leverage concepts proven useful by other industry efforts (such as PSL, System Verilog, Vera, and E).

VHDL-200X Fast Track discussed in this paper is the first phase of the VHDL-200X effort. VHDL-200X is a continuing effort that will include at least one additional set of revisions to the language.

1. Introduction

This paper gives a short overview and example of most of the features being added in VHDL-200X Fast Track. The features are not described in any particular order. If you need more details on a particular feature, see the VHDL-200X Fast Track webpage: http://www.eda.org/vhdl-200x/vhdl-200x-ft. Note this is a work in progress and may change up until balloting is completed.

2. Integration of PSL

Assertions are a concise way to specify static and dynamic (sequences of events) conditions. They can be used to specify design and interface requirements that must or must not happen. They can be checked either dynamically during simulation or statically using formal verification techniques. By validating their conditions with the design, they add visibility into the design's internal state.

Rather than develop a VHDL specific syntax for assertions, it was decided to integrate IEEE P1850 Property Specification Language (PSL) into VHDL. As a result, PSL declarations may be put into packages, and the declarative part of an entity, architecture, or block statement. PSL directives become a VHDL statement and are permitted in any concurrent statement part. PSL vunits become VHDL primary units and may include a context clause prior to the vunit.

By integrating PSL, VHDL leverages a standard assertion language that can be used with other HDLs and verification languages and at the same time has the same capabilities offered by a custom assertion language like SystemVerilog assertions (SVA).

3. IP Protection

A mechanism to protect VHDL source code has been added. This mechanism defines rules to encrypt the VHDL source and the format of the encrypted VHDL file. The syntax for specifying the encryption rules is identical to the syntax used for Verilog/System Verilog to facilitate tool development.

4. Type Generics & Generics on Packages

Generics have been extended to allow types and subprograms to be passed as generics. Packages and subprograms have been extended to allow generics to be mapped to them. The following shows a generic package.

```
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 ;
```

A generic package or subprogram must be instantiated before it can be referenced or used. The following package instantiations create packages that create the Mux4 for std_logic_vector and unsigned.

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

5. Context Unit

A context unit is a named primary unit that allows groups of library and use clauses to be referenced by a single name. An example of a context unit is shown below.

```
Context IEEE_CTX is
      use std.textio.all ;
   library ieee  ;
      use ieee.std_logic_1164.all;
      use ieee.numeric_std.all ;
      use ieee.numeric_unsigned.all ;
end ;
```

Rather than being overwhelmed by a large group of package references, a design unit can now reference a group of packages by referencing a single context unit as follows.

```
context work.IEEE_CTX ;
```

6. Composites with Unconstrained Arrays

Extend composites to allow arrays and records to contain unconstrained arrays.

```
type std_logic_matrix is array (natural range <>)
  of std_logic_vector ;
. . .
signal A : std_logic_matrix(7 downto 0)(5 downto 0) ;
. . .
A(5) <= "111000" ;    -- Accessing a Row
A(7)(5) <= '1' ;          -- Accessing an Element
. . .
entity e is
port (
   A : std_logic_matrix(7 downto 0)(5 downto 0) ;
   . . .
) ;
```

```
type complex is record
  a  : std_logic ;
  re : signed ;
  im : signed ;
end record ;
. . .
signal B : complex (re(7 downto 0), im(7 downto 0)) ;
```

7. Fixed Point Packages

The new package, ieee.fixed_pkg, defines the types ufixed and sfixed. To support fractional parts, negative indicies are used. The range downto is required. The whole number is on the left and includes the zero index. The fractional part is to the right of the zero index.

```
type ufixed is array (integer range <>) of std_logic;
type sfixed is array (integer range <>) of std_logic;
. . .
constant A : ufixed (3 downto -3) := "0110100" ;
. . .
--  3210    -1-2-3
--  IIII    FFF
--  0110    100   = 0110.100     = 6.5
```

A ufixed (or sfixed) addition/subtraction operation has a full precision result (unlike numeric_std which does modulo addition/subtraction – result is the same size as the largest array operand).

```
signal A, B : ufixed (3 downto -3) ;
signal Y    : ufixed (4 downto -3) ;
. . .

Y <= A + B ;
```

8. Floating Point Packages

The new package, ieee.fphdl_base_pkg, defines the type float. The range downto is required. The sign bit is the left most bit. The exponent contains the remaining bits on the left downto and including the zero index. The mantissa (fractional part) is to the right of the zero index.

```
type float is array (integer range <>) of std_logic;
. . .
signal A, B, Y : float (8 downto -23) ;
. . .
Y <= A + B ;  -- float numbers must be same size
```

The floating point format can be visualized as follows:

```
Formatting Examples:
                00000000011111111112222
8   76543210    12345678901234567890123
S   EEEEEEEE    FFFFFFFFFFFFFFFFFFFFFFF

E = Exponent has a bias of 127 (2**E'length-1)
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
```

This example only shows a portion of the floating point capability. More on the fixed and floating point packages will be presented by David Bishop in the paper,

"Fixed- and floating-point packages for VHDL 2005 " at DVCon 2005.

9. Process (all)

Process (all) indicates that all signals that are read in the process will implicitly be included on the sensitivity list. The following example uses the keyword all instead of including A, B, C, and MuxSel on the sensitivity list.

```
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
```

10. Case Statement Updates

The rule requiring the case expression to have a locally static subtype has been removed.

Case choices must still be locally static expressions, however, the rules for locally static expressions have been extended. Now locally static expressions are permitted to contain implicit operators that produce composite results. In addition, locally static expressions are permitted to contain operators and functions defined in the packages, ieee.std_logic_1164, ieee.numeric_std, ieee.numeric_bit, ieee.numeric_unsigned, or ieee.numeric_bit_unsigned.

```
constant ONE1 : unsigned := "11"  ;
constant CHOICE2 : unsigned := "00" & ONE1 ;
constant CHOICE3 : unsigned := "0" & ONE1 & "0";
constant CHOICE4 : unsigned :=  ONE1 & "00";
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 CHOICE3 =>         Y <= "10" ;
    when CHOICE4 =>         Y <= "11" ;
    when others =>          Y <= "XX" ;
  end case ;
end process ;
```

Although concatenation is specifically allowed, when joining elements of type std_ulogic or std_logic a type qualifier will be necessary as shown below to distinguish between types std_logic_vector and std_ulogic_vector. However, the code is still far simpler to both write and understand than the previous rules. Note no type qualifier would have been necessary if one of the objects being concatenated were std_logic_vector.

```
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 ;
```

11. Case With Don't Care

A new form of case has been created that allows use of a don't care character in the case choice. Note the don't care is not treated as a don't care in the case expression.

```
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 ;
```

12. Conditional Expression Updates

Conditional expressions have been expanded to allow bit and std_ulogic/std_logic as well as boolean results. In addition overloading has been added to the logic operators that yields a bit (std_ulogic) result when boolean and bit (std_ulogic) are used together. As a result, the following are all valid.

```
-- new
if (Cs1 and not nCs2 and Cs3 and Addr=X"A5")
-- backward compatible
if (Cs1='1' and nCs2='0' and Cs3='0' and Addr=X"A5")
if ((Cs1 and not nCs2 and Cs3)='1'   and Addr=X"A5")
-- Mixed, new and old
if (Cs1 and nCs2='0' and Cs3 and Addr=X"A5")
```

The new overloading can also be used in signal assignments.

```
Sel <= Cs1 and not nCs2 and Cs3 and Addr=X"A5" ;
```

13. Permit Expressions in Port Maps

Entity/component port associations have been extended to allow expressions in a port map. The expression eliminates the need to create an extra signal assignment. Semantically an expression in a port map that cannot be interpreted as a conversion function creates an implicit signal and incurs a delta cycle delay (the same as an explicit assignment would have).

```
U_E : E   port map ( A, Y and C, B) ;
```

14. Read Out Ports

The interface object rules have been updated to allow an interface object of mode out to be read. Reading out ports eliminates the need to create temporary internal signals within an entity or subprogram to read signal output ports and parameters. Allowing the reading of the driving value will facilitate design and verification capabilities such as referencing a ports driving value in an assertion.

15. Stop and Finish

The capability to stop a simulation with calls to procedures stop or finish has been added. The procedures stop and finish are bound to VHPI internal procedures.

16. Conditional & Selected Assignment

Conditional and selected assignment have been enhanced so that they may be used in sequential code for both signal and variable assignment. This proposal makes the code for NS1 equivalent to the code shown for NS2.

```
NS1 <= FLASH when (FP = '1') else IDLE ;
. . .
if (FP = '1') then
   NS2  <= FLASH ;
else
   NS2  <= IDLE ;
end if ;
```

Selected assignment provides a more concise form of case when only one data object is being targeted.

```
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 ;
```

17. If Expressions

If expressions have been added. An if expression is similar to conditional assignment, however, it can be used anywhere an expression is used. The following are examples

```
Signal A : integer := 7 if GEN_VAL = 1, 15 ;

Y1 <= A if SelA,  B if SelB, C if SelC, D ;
Y2 <=  (A and B) if S = '1',  (C and D) ;
Y3 <=  (A if S, B) + (C if S, D) ;
```

N-Nary expressions have low precedence, so parentheses are only required for Y3 above. N-Nary expressions use a ‘,’ as a separator similar to selected assignment. The expectation being that long term N-Nary expressions will replace most applications of conditional assignment.

18. TextIO enhancements

Overloaded read and write procedures have been added for std_logic. For all bit based array types (bit_vector, std_logic_vector, …) overloaded write, owrite, hwrite, read, oread, and hread procedures have been created.

A procedure tee has been created that write the line to both a file (like writeline) and to the file OUTPUT (defined in std.textio). A string constant named NL has been defined. NL will contain the correct characters to produce a carriage return and line feed.

A procedure sread has been created to read string based tokens. It skips leading white space and reads consecutive non-space characters up to its argument length long. If less than argument length characters are read, then the remaining characters of the argument will be filled with blanks. A procedure swrite has been created to write strings (so you will not longer need to use a type qualifier).

19. To_string functions

An overloaded function named to_string has been created for all types. For all logic based array types the functions to_ostring and to_hstring have been created. One expected use of these functions is with the report statement.

```
assert (ExpectedVal = ReadVal)
  report "Expected Val /= Actual Val.  Expected = " &
    to_string (Expected) & "   Actual = " &
    to_string (ReadVal)
  severity error ;
```

The following example writes to the console using VHDL’s built in write procedure.

```
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)) ;
```

20. Sized Bit String Literals

Specification of bit string literals has been extended to allow an optional size to be specified. Notation for signed (S) and unsigned (U) literal values has been created. The default literal is unsigned. If the string value does not specify enough characters, the value is extended based on the numeric value specified. If extra characters are specified, it is an error if the removal of them changes the value of the object. A few examples:

```
X"-X"       = "----XXXX"
7X"F"       = "0001111"   -- unsigned fill with 0
7UX"F"      = "0001111"   -- same as above
7SX"F"      = "1111111"   -- signed replicate sign
7UX"0F"     = "0001111"   -- ok.  Same value
7UX"8F"     = "0001111"   -- error.  Value changed
7SX"8F"     = "1001111"   -- error.  Value change.
7SX"CF"     = "1001111"   -- ok.  Same value
```

21. Unary Reduction Operators

Unary versions of AND, OR, NOR, NAND, XOR, and XNOR have been created for logic array types (bit_vector, std_logic_vector, …). The operator is applied to each element of the array and produces an element result (a reduction operation). These operators will have the same precedence as the miscellaneous operators (**, ABS, and NOT). This proposal makes the code for Parity1 equivalent to the code for Parity2.

```
Parity1 <= xor Data and ParityEnable;

Parity2 <=
  (data(7) xor data(6) xor data(5) xor data(4) xor
   data(3) xor data(2) xor data(1) xor data(0))
   and ParityEnable ;
```

22. Array/Scalar Logic Operations

The binary logic operators have been overloaded to produce an array (bit_vector, std_logic_vector, …) when the one operand is an element (bit or std_ulogic) and the other is an array. This proposal makes the Data1 and Data2 equivalent.

```
Data1 <= A and Asel ;

GenLoop : for I in Data2'Range loop
begin
   Data2(I) <= A(I) and Asel ;
end generate;
```

Creation AND-OR logic is also greatly simplified:

```
DataOut1 <=
    (A and ASel) or (B and BSel) or
    (C and CSel) or (D and DSel) ;
```

Without these operators, a common mistake is to write the above code as follows. Although functionally correct when the select signals (Asel, …) are mutually exclusive, it results in an inefficient hardware implementation.

```
Y <=
   A when ASel = '1' else  B when BSel = '1' else
   C when CSel = '1' else  D when DSel = '1' else
  (others => '0') ;
```

23. Hierarchical Reference

A missing feature from VHDL is the ability for an entity (such as a testbench) to probe and force signals that are defined in another part of the design hierarchy (such as a tri-state enable output of the chip). Some simulator vendors have created their own proprietary package of probe and force functions. The standards group has received donations and is creating a standard package based on these.

24. Std Logic Family Updates

The package ieee.std_logic_1164 has been made part of IEEE 1076 and is being updated as part of this effort. The following actions have been done for the std_logic_1164 package: uncomment xnor operators, add shift operators, add logic reduction operators, add array/scalar logical operators, add a match function (that understands don't care characters), add to_string functions, and provide overloaded read and write procedures.

25. Numeric Standard Updates

The package ieee.numeric_std has been made part of IEEE 1076 and is being updated as part of this effort. The following actions have been done for the numeric_std package: add logic reduction operators, add array/scalar logical operators, add array/scalar addition operators, add a match function, add functions to_x01 and is_x, add to_string functions, and provide overloaded read and write procedures.

26. Revision Plans Following Fast Track

VHDL-200X Fast Track effort is the first step in updating VHDL. As soon as Fast Track revisions are completed, the plan is to start the next set of revisions.

Some of the items that are planned to be part of the revision following Fast Track are:

- Constrained Random Stimulus Generation
- Verification Data Structures such as associative arrays
- Queues, FIFOs, and Memories
- Object Orientation
- Direct C and Verilog/SystemVerilog Calls

With these additions, VHDL will become a full HDVL (hardware design and verification language) and design teams will no longer need to consider supplementing VHDL with a separate verification language or go through

an expensive transition (in terms of training and experience) to an alternate HDVL such as SystemVerilog.

27. Summary

The VHDL-200X Fast Track effort is planned to become the standard IEEE 1076-2005. It is the intent to bring both enhanced capability and simplified usage of the language.

There is still much work to be done by the VHDL-200X working group and still plenty opportunity to participate. You can participate by requesting enhancements, working as part of the standards group, testing packages written by the standards group, lobbying your EDA vendors to support the standard, and requesting your company to financially support the VHDL LRM editing. More details of each of these follow.

28. Requesting Enhancements

You might note that we may have missed the changes you have been wishing for (no matter how simple or obivious). To ensure we don't miss them in the future, make sure to submit them via the enhancement request link on the webpage: http://www.eda.org/vhdl-200x.

Note also that the standards organization is a volunteer run organization and people tend to work on what interests them. If no one but you is interested in your suggestion, it may sit on the back burner. Your only insurance is to participate.

29. Participating In VHDL Standards

VHDL standards are IEEE standards. As a VHDL community member it is both your right and responsibility to join IEEE committees and participate in VHDL standards. You can participate at different levels. As a non-voting member, in which no organizational membership is required, you can join the working group email reflectors, attend teleconferences, attend in person meetings and participate in all discussions. To have an offical vote on issues, you must become a voting member. To learn more about VHDL standards see the following websites.

EDA Standards:	http://www.eda.org
VHDL-200X:	http://www.eda.org/vhdl-200x
DASC:	http://www.eda.org
RTL Synthesis:	http://www.eda.org/siwg.

30. Funding VHDL Standards

Much of the work done on a standards project is done on a volunteer basis. This includes maintaining email reflectors and webpages, writing and editing proposals, and participation in technical meetings (both in person and teleconferences).

The task of integrating all of the proposals into changes to the LRM (IEEE 1076 standard) and editing the LRM is undertaken by a single, paid technical editor. This person is a VHDL expert with deep knowledge. Due to the amount of time this effort takes, this work is done by a paid consultant.

Unfortunately none of our working group funding comes from IEEE or IEEE-SA. Some of the money will be coming from EDA vendors, however, we are seeking industry and government sponsors for the balance of the funding. If your organization can help with funding, please contact the VASG chair, Steve Bailey, at stephen@srbailey.com.

31. Vendor Support

Writing a standard is the first step toward getting new features in the language. The next step is vendor implementation. Supporting a standard is a business decision. To make this an easy decision for the vendors, please be vocal in letting them know that you want them to support the new features of IEEE 1076. Often the most effective person to pass this information to the vendors is the person who is purchasing your EDA tools. A particularly effective time to do this is when buying new licenses or renewing your current licenses.

32. Acknowlegements

This paper is based on the hardwork of people supporting the VHDL-200X effort. Numerous proposals were either written by or refined by John Ries. David Bishop developed the fixed and floating point packages as well as updated numerous of the other packages. Peter Ashenden wrote the generic enhancement proposal and is the LRM editor. Erich Marschner worked on the PSL integration. Deepak Pant and Ajayhash Varikat worked on the IP protection. Ryan Hinton worked on the composites of unconstrained arrays proposal. Steve Bailey has chaired the group responsible for the VHDL-200X effort.

33. References

David Bishop, "Fixed- and floating-point packages for VHDL 2005", to be published in DVCon 2005.

Ryan Hinton and Brad Davis, "Data abstraction is the beginning of algorithm abstraction", to be published in DVCon 2005

VHDL-200X proposals webpage, http://www.eda.org/vhdl-200x/vhdl-200x-ft/proposals/proposals.html

34. About the Author

Jim Lewis, the founder of SynthWorks, has nineteen years of design, teaching, and problem solving experience. In addition to working as a VHDL course developer and trainer for SynthWorks, Mr. Lewis does ASIC and FPGA design, custom model development, and consulting. Mr. Lewis is an active member of IEEE 1076 / VHDL-200X standards group and is the team leader of the Fast Track effort. He is also an active member of the RTL Synthesis (IEEE 1076.6). Mr. Lewis can be reached at jim@SynthWorks.com