

VHDL-200X & The Future of VHDL

by

Jim Lewis, SynthWorks

Steve Bailey, Model Technology

Erich Marschner, Cadence Design Systems

J. Bhasker, eSilicon Corp.

Peter Ashenden, Ashenden Designs Pty. Ltd.

DesignCon 2004

1

The Future of VHDL

SynthWorks

With all the media hype about languages such as Verilog/SystemVerilog, Vera, and specman e where does the future of VHDL lie? 2003 has heard many claims about VHDL being "The New Latin" and it is dead.

Fortunately these claims were made by people or companies who have very little interest (or market share) in VHDL and are looking to push the market in their direction.

The VHDL designer and vendor community are actively working on revisions to both VHDL and the packages that support the language. These revisions are integrating the newest features of verification languages and assertion languages as well as adding features such as a programming language interface (VHPI) and a simulation control interface. In addition, changes are being made to improve performance and ease of use.

To ensure wide support of the new features, EDA vendors are being actively engaged to participate in the standardization efforts and only features that have both designer and vendor support will be integrated into the language.

DesignCon 2004

2

- Significant enhancements are currently being made to VHDL.
- This paper summarizes the efforts being made by the following IEEE working groups:

Agenda

VHDL-200X

IEEE 1164

IEEE 1076.3/numeric std

IEEE 1076.3/floating point

IEEE 1076.6

VHDL-200X Fast Track

Website

<http://www.eda.org/vhdl-200x>

<http://www.eda.org/vhdl-std-logic>

<http://www.eda.org/vhdlsynth>

<http://www.eda.org/fphdl>

<http://www.eda.org/siwg>

<http://www.eda.org/vhdl-200x/vhdl-200x-ft>

Caution: All activities here are work in progress.

VHDL-200X, Goals / What

- Enhance/update VHDL to improve performance, modeling capability, ease of use, verification features, simulation control, and the type system.
- Maintain VHDL style, nature, and backward compatibility
- Leverage industry efforts
 - Spring board off of efforts by PSL assertions, Verisity E, Vera, and SystemVerilog
- Focus on features sponsored and prototyped by both users and vendors to ensure quick adoption and that features are both cool and useful.

- Kick-Off: Feb 2003 (at DVCon)
- VHDL-200X is being developed in a time phased effort.
 - The first phase is called Fast Track and is intended to be completed in early 2004 (goal DAC 2004)
 - The remainder of the work will be sorted in a priority basis and will be developed in one or more following revisions.

VHDL-200X, Sponsors

- IEEE
 - IEEE-CS: IEEE Computer Society
 - DASC: Design Automation Standards Committee
 - VASG: VHDL Analysis and Standardization Group
 - VHDL-200X: Current Development Work
- IEEE-SA: IEEE Standards Association
 - Coordinates balloting on all IEEE Standards.

Websites:

IEEE:	http://www.ieee.org
DASC:	http://www.dasc.org
VASG:	http://www.eda.org/vasg
Accellera:	http://www.accellera.org

- Observer Participants
 - By IEEE rules, anyone (including non-IEEE members) with a vested interest may attend meetings, join reflectors, and comment on standards activity (at meetings and on reflectors) .
- Voting Members
 - Development: Member of IEEE-CS + DASC + IEEE
 - Balloting: Member of IEEE-SA
- Your input can make a difference. Participate.

VHDL-200X, Groups

- Work on VHDL has been tasked to the following subgroups:
 - Fast Track
 - Performance
 - Modeling and Productivity
 - Testbench/Verification
 - Assertions
 - Data Types and Abstraction
 - Environment
- Each group is supported by its own separate reflector.
 - To keep up with all, join each reflector plus the general reflector
 - See <http://www.eda.org/vhdl-200x> for details

- Goals:
 - Critical updates needed for other standards (such as IEEE P1164, IEEE P1076.3, and Assertions).
 - Other high priority items
- A few items on the list are:
 - Unary Reduction Operators
 - Array/Scalar Logic Operators
 - Hierarchical signal access
 - Formatted IO: hwrite, hread, owrite, oread ...
 - String Conversions: to_string, to_hstring, to_ostring ...
 - Conditional + selected signal assignment in a process
 - Details follow ...
- Fast track items must be non-controversial

- Goals:
 - Make language changes that facilitate enhanced tool performance, primarily for, but not only for simulation.

VHDL-200X, Modeling and Productivity

- Goals:
 - Improve designer productivity through
 - enhancing conciseness,
 - simplifying common occurrences of code, and
 - improving capture of intent.
 - Facilitate modeling of functionality that is currently difficult or impossible.
- A few items on the list are:
 - Case expressions.
 - Case/If Generate
 - Pick up where fast track leaves off

VHDL-200X, Testbench and Verification

- Goals:
 - Ease the job of the verification engineer.
 - Give VHDL similar functionality to Vera and Verisity E.
- A few items on the list are:
 - Constrained Random stimulus generation with optional and dynamic weighting
 - Associative arrays
 - Queues/FIFOs
 - Memory implementation and loading & dumping

- Goals:
 - Define support for temporal expressions and assertion-based verification in VHDL.
 - Consider formal, synthesis, and coverage implications.
- Approach:
 - Exploit work of others.
 - Current plan is to integrate PSL by reference

VHDL-200X, Data Types and Abstractions

- Goals:
 - Enhancements centered on the type system.
 - Higher abstraction level constructs
- A few items on the list are:
 - Generics for Packages (including types)
 - Object-orientation
 - Greater than 32-bit range for integers (infinite range)
 - Sparse / Associative Arrays
 - User-defined floating point mantissa/exponent
 - User-defined positional values of enum literals

- Goals:
 - Simulation control environment.
 - Standard interfaces to other languages.
 - Additional support packages.
- A few items on the list are:
 - Simulation control (like \$stop, ... in Verilog)
 - Direct C and Verilog calls with well defined mapping of data objects (VHDL integer to C int)
 - Conditional compilation
 - VCD for VHDL
 - TEE functionality to STD.OUTPUT
 - Verilog and C Foreign interfaces

- With VHDL-200X, VHDL will transition to an HDVL
 - Will be integrating the good features from Vera, SystemC, specman e, and SystemVerilog
- End result
 - Get full verification capabilities
 - Language consistency of VHDL
 - Not necessary to use other languages or switch
- VHDL-200X = 200 X better than ...

- 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
 - Provide text I/O package for standard logic (similar to Synopsys' std_logic_textio)
- Website: <http://www.eda.org/vhdl-std-logic>
- See also DVCon 2003 paper, "Enhancements to VHDL's Packages" which is available at:
<http://www.synthworks.com/papers>

- Goals:
 - Enhance current numeric_std package.
 - Unsigned math with std_logic_vector/std_ulogic_vector
 - Add a package for floating point
- 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
- Website: <http://www.eda.org/vhdlsynth>
<http://www.eda.org/fphdl> -- floating point

- Goals: Enhance synthesis coding styles to accept a wider set of synthesizable objects
- A few items on the list are:
 - Broader register coding styles
 - Multiple clocked and multiple edged registers
 - Support synthesis of registers in subprograms
 - Support registers and latches in concurrent assignments
- Website: <http://www.eda.org/siwg>
- See DVCon 2004 session, "IEEE 1076.6: VHDL Synthesis Coding Styles for the Future"
- See HDLCon 2002 paper, "Extensions to the VHDL RTL Synthesis Standard", at <http://www.synthworks.com/papers>

- Business view of supporting EDA standards
 - Supporting a feature of a standard is an investment
 - Feature support is market driven
 - If you don't ask, they don't support it.
- As a result, if you see new features you want to use, tell your EDA vendor.

- Unary Reduction Operators
- Array/Scalar Logic Operators
- to_string, to_hstring, ...
- hwrite, owrite, ... hread, oread
- Hierarchical references of signals, ...
- Sized bit string literals
- Conditional and Selected assignment in sequential code
- Expressions in port maps
- Read out ports
- Add Stop, Finish, and Restart as callable routines
- Unconstrained arrays of unconstrained arrays
- Records of unconstrained arrays
- Context clause design unit
- Simplified if expressions+
- Individual IO mode specification for records
- Process_Comb, Process_latch, Process_ff
- Concatenation on LHS

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) ;
```

- Proposal: Create symmetric array/scalar overloading of all binary logic operators for bit_vector, std_logic_vector, ...

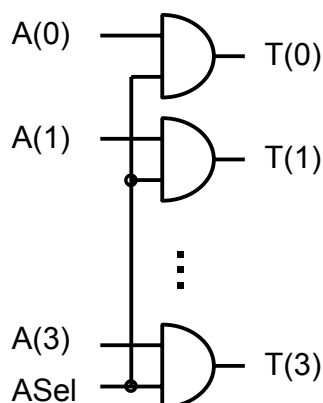
```
function "and"(anonymous: BIT_VECTOR; anonymous : BIT)
    return BIT_VECTOR;
function "and"(anonymous: BIT;          anonymous : BIT_VECTOR)
    return BIT_VECTOR;
. . .
```

- Application

```
signal ASel, BSel, CSel, DSel : bit ;
signal Y, A, B, C, D : bit_vector(3 downto 0) ;
. . .
Y <= (A and ASel) or (B and BSel) or
      (CSel and C) or (DSel and D) ;
```

- In this context, the following code implies the hardware below:

```
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"

- Report requires string values. To_string would make report more useful:

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

- Furthermore, to_string permits a usage of vhdl-93 write:

```
-- write(<file_handle>, <string>) ;
write(Output, "%%ERROR data value miscompare." &
  CRLF & "  Actual value = " & to_hstring(Data) &
  CRLF & "  Expected value = " & to_hstring(ExpData) &
  CRLF & "  at time:  " & to_string(now, right, 12)) ;
```

- Support hex, octal, binary, and decimal for all types (integer, bit_vector, ...)

```
function to_string (
  VALUE          : in integer;
  JUSTIFIED      : in SIDE := RIGHT;
  FIELD         : in WIDTH := 0
) return string ;
function to_hstring ( . . . ) return string ;
function to_ostring ( . . . ) return string ;
function to_bstring ( . . . ) return string ;
function to_dstring ( . . . ) return string ;
```

- Support write with radix, similar to std_logic_textio, for all types

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

- Work inspired by Synopsys' donation of std_logic_textio to IEEE for IEEE 1164 efforts.
 - Goal to stay compatible with std_logic_textio

- Support write with radix, similar to std_logic_textio, for all types

```
function hread (  
    Buf          : inout Line ;  
    VALUE        : out integer;  
    Good         : out boolean  
);  
function dread ( . . . ) ;  
function oread ( . . . ) ;  
function bread ( . . . ) ;
```

- Permanent connection to object by expanding upon alias.
 - Mode specifies in (read), out (drive), or inout
 - Path to signal specified in the format of path_name (see attribute 'path_name')
 - Currently objects envisioned to be signals, constants (and hence generics), and shared variables

```
Alias addr : std_logic_vector signal is
  out ":tb:u_uut:u_mem_ctrl:addr" ;

Alias addr : std_logic_vector(7 downto 0) signal is
  out ":tb:u_uut:u_mem_ctrl:addr" ;

Alias data : std_logic_vector(7 downto 0) signal is
  inout ":tb:u_uut:u_mem_ctrl:data" ;
```

Hierarchical Reference*

- Temporary connection to signal with procedures
 - signal_force and signal_release:

```
procedure signal_force (
  destination_signal : IN STRING;
  force_value       : IN STRING;
  force_delay       : IN TIME := 0 ns ;
  force_how         : IN force_mode := DEPOSIT;
  cancel_period     : IN DELAY_LENGTH := 0 ns ;
  delta_event       : IN BOOLEAN := FALSE );

procedure signal_release (
  destination_signal : IN STRING );
```

- For more details see the proposal on the fast track website.
- *Work inspired by donations from both Mentor/ModelTech and Cadence

Enhance Conditional Assignment SynthWorks

- Proposal: Allow use of conditional assignment in sequential code
- Common branching in a statemachine:

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

- Simplifying this code by using conditional signal assignment

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

- Note: the new part is doing this in a process

Enhance Conditional Assignment SynthWorks

- Further enhance conditional assignment to do the following:
 - Ternary Operation, Synchronous Reset

```
AReg <=
    ('0' when nReset = '1' else A) when rising_edge(Clk) ;
```

- Initialization:

```
Signal A : integer := 7 when GEN_VAL = 1 else 15 ;
```

- Currently supported coding styles (1076.6-2004):
 - Register with conditional signal assignment (currently supported)

```
AReg <= A when rising_edge(Clk) ;
```

- Asynchronous Reset (not ternary):

```
AReg <= '0' when nReset = '1' else
    A when rising_edge(Clk) ;
```

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

- Need in PSL and OVL to avoid creating an extra signal assignment
- Semantics of expressions in port map:
 - convert to an equivalent concurrent signal assignment
 - if expression is not a single signal, constant, or does not qualify as a conversion function, then 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

- Currently one way to stop a simulation is:

```
Report "Just Kidding. Test Passed" Severity Failure ;
```

- Which produces the message:

```
# ** Failure: Just Kidding. Test Passed
#   Time: 1060 us Iteration: 4 Instance: ...
```

- Create procedures STOP and FINISH that either bind to VHPI calls or internal simulator routines.

```
procedure STOP;
-- Stop a simulation in the manner that breakpoint does

procedure FINISH;
-- Terminate a simulation and exit to the simulator prompt
```

Arrays of Unconstrained Arrays

```
type std_logic_matrix is array of std_logic_vector ;

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

-- Accessing a Row
A(5) <= "111000" ;

-- Accessing an Element
A(7)(5) <= '1' ;

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 ;

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

Problem:

Currently users have to specify a large collection of packages before an entity and there is no way to abstract this.

```
library ieee ;
  use ieee.std_logic_1164.all ;
  use ieee.numeric_std.all ;
  use std.textio.all ;
```

- This problem will continue to grow with additional standards packages
 - Floating Point
 - Unsigned math with std_logic_vector
 - Assertion Libraries
 - . . .

Proposal:

- Create a design unit which is the context clause
- Create a default context clause and allow users to override it.

Current default context clause:

```
Context default_context is
  library std ;
    use std.standard.all ;
  library work ;
end ;
```

Proposed default context clause:

```
Context default_context is
  library std ;
    use std.standard.all ;
  library work ;
  library ieee ;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all ;
  use std.textio.all ;
end;
```

- The default_context is referenced by default

- The default _context can be overridden by compiling it a new version into the work library.
 - Intent is for the default_context to define a project/company wide standard usage of packages.
- Also proposed is the ability to create named contexts and

```
Context project1_Ctx is
  library Lib_p1 ;
    use Lib_P1.P1Pkg.all ;
    use Lib_P1.P1_Defs.all ;
    ...
end ;
```

- Reference named contexts:

```
Library Lib_P1 ;
context Lib_P1.project1_ctx ;
```

- Enable simplified conditional expressions
 - if, elsif, wait until, when, while

```
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
```

- Be consistent with std_ulogic, so active low is represented with "not nWe"

```
Sel <= Cs1 and not nCs2 and Cs3 ;
```

- Backward compatible with current VHDL syntax:

```
if (Cs1='1' and nCs2='0' and Cs3='0' and Addr=X"A5") then
if ((Cs1 and not nCs2 and Cs3)='1' and Addr=X"A5") then
if nWe = '0' then
```

- Implementation Part 1:
At top level of an conditional expression, if the resulting expression is not boolean, call the function condition? to convert to boolean (if it exists)

- With Part 1: the following will work:

```
if (Cs1 and not nCs2 and Cs3) then
```

- Intended types to create overloading for are bit types (bit, std_ulogic)

- Implementation Part 2:
Create overloaded logic operators that allow boolean to be used with bit/std_ulogic and result in bit/std_ulogic

- This promotes true to '1' and false to '0' to maintain accuracy
- This enables the following two examples:

```
if (Cs1 and not nCs2 and Cs3 and Addr=X"A5") then
DevSel1 <= Cs1 and not nCs2 and Cs3 and Addr=X"A5" ;
```

- Specify mode of each field of a record

```
Mode IfRecType_Mode_Model1 : IfRecType := (  
  Sig1    => in,  
  Sig2    => out  
end mode ;
```

```
Mode IfRecType_Mode_Model2 : IfRecType := (  
  Sig1    => out,  
  Sig2    => in  
end mode ;
```

```
Entity Model1 is  
port (  
  IfRec1 : IfRecType_Mode_Model1 : IfRecType ;  
  . . .
```

Process Comb, ...

- Process_Comb
 - Indicates a process only contains combinational logic
 - Automatically create a sensitivity list with all signals on sensitivity list
 - If process creates a latch or register, synthesis tools shall generate an error and not produce any netlist results.

```
Mux3_proc : process_comb  
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 errors in creating combinational logic, particularly, statemachines.

- Process_latch
 - Indicates a process only contains only latches
 - Automatically create a sensitivity list with all signals on sensitivity list
 - If the process creates combinational logic or a register, synthesis tools shall generate an error and not produce any netlist results.
- Process_ff
 - Indicates a process only contains only registers
 - Automatically create a sensitivity list with the clock signal and any asynchronous signals on the sensitivity list.
 - If the process creates combinational logic or a latch, synthesis tools shall generate an error and not produce any netlist results.

Concatenation on LHS

- Permit concatenation to be on the LHS

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

. . .

CarryOut & Result  <= ('0' & A) + ('0' & B) ;
```

This page is intentionally left blank

Overloading "&" and To String?

SynthWorks

```
function "&" (  
    l      : in string;  
    r      : in unsigned  
    ) return string ;  
function "&" (  
    l : in unsigned ; r : in string  
    ) return string ;  
function "&" (  
    l : in unsigned; r : in unsigned  
    ) return string ;
```

```
constant A : unsigned := "1010" ;  
signal Y : unsigned(7 downto 0) := ("0000" & A) + 1 ;  
. . .  
Y <= (A & A) + A ;  
Write(Output, "A twice: " & A & A & " Y = " & Y) ;
```