VHDL 200X Fast Track
New Features being Standardized

By Jim Lewis
SynthWorks VHDL Training
Team Leader VHDL-200X Fast Track
Jim@SynthWorks.com

MARLUG - Mid-Atlantic Region Local Users Group
ANNUAL CONFERENCE - OCTOBER 5, 2004
Johns Hopkins University Applied Physics Lab – Laurel, MD

VHDL-200X, Goals / What

SynthWorks

- 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.

Caution: All activities here are work in progress.

VHDL-200X, Organization

SynthWorks

- VHDL-200X is being developed in a time phased effort.
  - The first phase is called Fast Track and is intended to be completed in Mid 2005
  - The remainder of the work will be sorted in a priority basis and will be developed in one or more following revisions.

- Work is divided into several sub-groups:
  - Modeling and Productivity
  - Assertions
  - Testbench / Verification
  - Data Types and Abstractions
  - Performance
  - Environment

VHDL-200X, Participation

SynthWorks

- Observer Participants
  - By IEEE rules, anyone (including non-IEEE members) with a vested interest may attend meetings, join reflectors, and comment on standards activity.

- Voting Members
  - Development: Member of IEEE-CS + DASC + IEEE
  - Balloting: Member of IEEE-SA

- Your input can make a difference. Participate.
  - See http://www.eda.org/vhdl-200x for details
  - Join main + individual reflectors

- Not all tasks are standards tasks are LRM writing
  - Some simple tasks: review of new packages

VHDL-200X, Sponsors

SynthWorks

- 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

VHDL-200X, Financials $$$

SynthWorks

- IEEE/IEEE-SA do not fund standards projects
  - They provide infrastructure for balloting and legal issues
- Most of the work is done by volunteers
  - Maintaining of reflectors, webpages
  - Writing and editing proposals
  - Technical meetings
- Current plan is to hire an LRM editor
  - For this we need funding ($200K - $300K over 3 years)
  - Some money will come from EDA vendors,
  - But we will need other sources …
    - Contact Stephen Bailey (stephen@srbailey.com)

Vendor Support of Standards

SynthWorks

- Business view of supporting EDA standards
  - Supporting a feature of a standard is an investment
  - Feature support is user 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
  - tell your friends (who can then tell their vendors)

VHDL-200X-FT: Proposals

SynthWorks

- Unary Reduction Operators
- Array/Scalar Logic Operators
- to_string, to_hstring, …
- hwrite, owrite, … hread, oread
- Hierarchical references of signals
- Sized bit string literals
- Nnary Expressions
- 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
- Simplified if expressions+
- Process_Comb, Process_latch, Process_ff
- Aggregates with slices
- Simplified Case Statements
- Don't Care in a Case Statement
- Fixed Point Packages
- Floating Point Packages
- Type Generics
- Generics on Packages
- PSL
- IP Protection / Encryption
- Std_logic_1164 Updates
- Numeric_Std Updates

Much of VHDL's cumbersome syntax issues can be fixed

Unary Reduction Operators

SynthWorks

- 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

SynthWorks

- 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

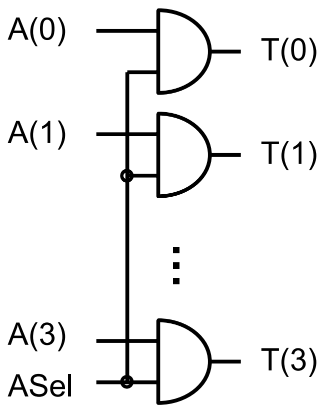
```
signal ASel, BSel, CSel, DSel : bit ;
signal Y, A, B, C, D : bit_vector(3 downto 0) ;
. . .
Y <= (A and ASel) or (B and BSel) or
     (Csel and C) or (DSel and D) ;
```

Array / Scalar Logic Operators

SynthWorks

- 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"

To_String, To_HString, ...

SynthWorks

- 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." &
  NL & "  Actual value = " & to_hstring(Data) &
  NL & "  Expected value = " & to_hstring(ExpData) &
  NL & "  at time:  " & to_string(now, right, 12)) ;
```

To_String, To_HString, ...

SynthWorks

- 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 ;
```

Hwrite, Dwrite, Owrite, Bwrite

SynthWorks

- 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

Hread, Dread, Oread, Bread

SynthWorks

- 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 ( . . . ) ;
```

Hierarchical Reference

SynthWorks

- 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" ;
```

- Still have problems to solve to make this a reality.

Hierarchical Reference *

SynthWorks

- Near Term Alternative, Package based approach

```
Drive(
  source_signal       : IN STRING ;
  destination_signal  : IN STRING ;
  delay               : IN TIME := 0 ns ;
  delay_mode          : IN delay_mode_type := DEPOSIT;
  verbose             : IN integer ) ;

Probe(
  source_signal       : IN STRING ;
  destination_signal  : IN STRING ;
  verbose             : IN integer ) ;
```

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

Hierarchical Reference*

SynthWorks

- Temporary read / write signal with procedures

```
procedure signal_force (
  destination_signal  : IN STRING;
  force_value         : IN STRING;
  delay               : IN TIME := 0 ns ;
  delay_mode          : IN delay_mode_type := DEPOSIT;
  cancel_period       : IN DELAY_LENGTH := 0 ns ;
  delta_event         : IN BOOLEAN := FALSE );

procedure signal_release (
  destination_signal  : IN STRING;
  verbose             : IN integer) ;

get_value(
  destination_signal  : IN STRING;
  verbose             : IN integer) ;
```

- *Work inspired by donations from both Mentor and Cadence

Sized Bit String Literals

SynthWorks

- Currently hex bit string literals are a multiple of 4 in size

```
X"AA"  =  "10101010"
```

- Allow specification of size (and decimal bit string literals):

```
7X"7F"  =  "1111111"
7D"127" =  "1111111"
```

- Allow specification of signed vs unsigned (for extension of value):

```
9UX"F"  =  "000001111"    Unsigned 0 fill
9SX"F"  =  "111111111"    Signed: left bit = sign
9X"F"   =  "000001111"    Defaults to unsigned
```

- Allow Replication of X and Z

```
7X"XX"  =  "XXXXXXX"
7X"ZZ"  =  "ZZZZZZZ"
```

N-Nary Expressions

SynthWorks

- Similar to conditional signal assignment …

```
Y <=  A and B  if S = '1',  C and D ;
Y <= (A and B) if S = '1', (C and D) ;
```

- … except it is an expression:

```
Y <= A and (B if S = '1', C) and D ;
```

- And it can be used anywhere an expression can:

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

with MuxSel select
  Y <= A if Asel='1', B when '0',
       C if Csel='1', D when '1',
      'X' when others ;
```

Allow Conditional Assignments for Signals and Variables in a Process

SynthWorks

- Statemachine code:

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

- Simplification:

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

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

- Also support conditional variable assignment:

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

Allow Selected Assignments for Signals and Variables in a Process

SynthWorks

```
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 ;
```

Signal Expressions in Port Maps

SynthWorks

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

- Needed for 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.
- Also facilitates mapping bits to arrays

```
U_Decoder : Decoder
  port map (
    Addr   => A2 & A1 & A0,
    Sel    => Sel
  );
```

```
U_Decoder : Decoder
  port map (
    Addr   => (A2, A1, A0),
    Sel    => Sel
  );
```

Read Output Ports

SynthWorks

- Read output ports
  - Value read will be locally driven value
- Assertions need to be able to read output ports

Stop and Finish

SynthWorks

- 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.

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

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

Arrays of Unconstrained Arrays

SynthWorks

```
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) ;
   . . .
) ;
```

Records of Unconstrained Arrays

SynthWorks

```
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)) ;
```

Context Clause Design Unit

SynthWorks

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
  - . . .

Context Clause Design Unit

SynthWorks

- Create a named context design unit that references packages to use

```
Context project1_Ctx is
 library ieee  ;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all ;
  use std.textio.all ;
  use ieee.numeric_unsigned.all ;

  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 ;
```

Simplified If Expressions+

SynthWorks

- 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 nCs2"

```
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
```

Simplified If Expressions+

SynthWorks

- 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_logic 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" ;
```

Process_Comb, ...

SynthWorks

- 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, Process_ff

SynthWorks

- Process_latch
  - Indicates all signal assignments in a process create only latches
  - Automatically creates a sensitivity list with all signals read in the process on the 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 all signal assignments in a process create only registers
  - Automatically creates 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.

Slices in Array Aggregates

SynthWorks

- Allow slices in an Array Aggregate

```
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:

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

Simplified Case Statement

SynthWorks

- Allow non-scalars to be locally static
- Integrate packages 1164, 1076.2, and 1076.3 into 1076
  - Make the types and operands in these packages locally static.

```
signal A, B     : unsigned (3 downto 0) ;
. . .

process (A, B)
begin
  case A xor B 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 ;
```

Simplified Case Statement

SynthWorks

- In some cases will still need a type qualifier, but not a constrained type
  - When both std_logic_1164 and numeric_std are visible, concatenating std_logic objects can result in std_logic_vector, signed, or unsigned.

```
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

SynthWorks

- Create new form of case: Case?
- Allow use of '-' in targets provided targets are non-overlapping

```
-- 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 ;
```

Fixed Point Types

SynthWorks

- Definitions in package, ieee.fixed_pkg.all

```
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.

```
constant A : ufixed (3 downto -3) := "0011010000" ;

  3210 -3
  IIIIFFF
  0110100  = 0110.100 = 6.5
```

- Math is full precision math:

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

Floating Point Types

SynthWorks

- Definitions in package, ieee.fixed_pkg.all

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

- Format is Sign Bit, Exponent, Fraction

```
signal A, B, Y : fp (8 downto -23) ;

  8  76543210  12345678901234567890123
  S  EEEEEEEE  FFFFFFFFFFFFFFFFFFFFFFF

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
```

Type Generics + Generics on Packages

SynthWorks

- Packages get instantiated to customize them for a particular type

PSL

SynthWorks

- PSL will be incorporated directly into VHDL
- Define and Specify properties in VHDL

IP Protection and Encryption

SynthWorks

- Makes IP model encryption methodology independent of EDA tool vendors

Std Logic_1164 Updates

SynthWorks

- 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

SynthWorks

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

SynthWorks

Operator	Left	Right	Result
Logic	TypeA	TypeA	TypeA
Numeric	Array	Array	Array*
	Array	Integer	Array*
	Integer	Array	Array*
Logic, Addition	Array	Std_ulogic	Array
	Std_ulogic	Array	Array
Logic Reduction		Array	Std_ulogic

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

VHDL-200X, Summary

SynthWorks

- Fast Track work is the first phase of VHDL-200X
  - Expected completion of current work is mid-2005.
- Lots of more work to be done
  - Goal: Transition VHDL into a full Hardware Description and Verification Language (HDVL).
  - Integrate good features of 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 ...

Appendix VHDL-200X, Subgroups SynthWorks

- The primary intent of this presentation was to cover near term work that is being delivered with the VHDL-200X fast track.

- This appendix covers the charter of the following subgroups:
  - Performance
  - Modeling and Productivity
  - Testbench / Verification
  - Assertions
  - Data Types and Abstractions
  - Environment

- Each VHDL-200x subgroup has its own webpage, reflector, and team leader
  - To be fully vested in the process, one would have to sign up for the VHDL-200X reflector and the reflector of each subgroup.

VHDL-200X, Performance SynthWorks

- 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/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

VHDL-200X, Assertions

SynthWorks

- 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

VHDL-200X, Environment

SynthWorks

- 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

Other VHDL Standards Work

SynthWorks

- 1076.6-2004 VHDL RTL Synthesis Standard
  - Enhanced synthesis coding styles to accept a wider set of synthesizable objects

- A few items updated are:
  - Broader register coding styles
  - Multiple clocked and multiple edged registers
  - Support synthesis of registers in subprograms
  - Support registers and latches in concurrent assignments

- See DVCon 2004 paper, "IEEE 1076.6: VHDL Synthesis Coding Styles for the Future," and HDLCon 2002 paper, "Extensions to the VHDL RTL Synthesis Standard," which are at http://www.synthworks.com/papers