IEEE 1076-2008
VHDL-200X

By

Jim Lewis

SynthWorks VHDL Training

jim@synthworks.com

IEEE VHDL Working Group Chair

VHDL-2008: Powerful, Easier to Use VHDL

Contact Information
Jim Lewis, President
SynthWorks Design Inc
11898 SW 128th Avenue
Tigard, Oregon 97223
503-590-4787
800-505-VHDL
jim@SynthWorks.com

http://www.SynthWorks.com

IEEE 1076-2008 = VHDL-200X

- IEEE 1076-2008 is a work product of IEEE VASG and Accellera VHDL working group

- History:
  - Feb 2003, started as VHDL-200X by IEEE VASG
  - Sept 2005, Accellera provides a funding and a separate working group
  - July 2006, VHDL Draft 3.0 becomes an Accellera standard
  - Summer 2008, released back to IEEE
  - September 2008, approved as IEEE 1076-2008

- Standard available at http://www.ieee.org/go/shop

IEEE 1076-2008

- Biggest Language change since 1076-1993

- PSL, IP Protection, VHPI
- Fixed and Floating Point Packages
- Records and Arrays with unconstrained elements
- Process(all)
- New Types: Integer_vector …
- ENV package: STOP
- Package Integration
- New and Enhanced Operators
- Simplified Conditional (IF, While)
- Simplified Case Statements
- Don't Care in a Case
- Enhanced bit string literals
- Better Printing
- Extended Assignments
- Enhanced Port Maps
- Context Declarations and clause
- Enhanced Generics

PSL, IP Protection, VHPI

PSL = Property Specification Language (IEEE 1850)

- Assertion language integrated directly into VHDL
  - Properties are VHDL block (concurrent) declarations
  - Assert and cover are VHDL concurrent statements
  - Vunit, Vmode, Vprop are VHDL Design Units

IP Protection and Encryption

- A pragma-based approach
  - Keywords and constructs specify algorithms and keys
  - Constructs demarcated protected envelopes of VHDL code

VHDL Procedural Interface - VHPI

- Standardized Procedural Programming Interface to VHDL
  - Gives tools access to information about a VHDL model during analysis, elaboration, and execution

---

Fixed Point Types

- Definitions in package: ieee.fixed_pkg.all (instance of fixed_generic_pkg)

```
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) := "0110100" ;

  3210 -3
  IIII FFF
  0110 100  = 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

- Definitions in package: ieee.float_pkg.all (instance of float_generic_pkg)

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

- Format is Sign Bit, Exponent, Fraction

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

  8  76543210  12345678901234567890123
  S  EEEEEEEE  FFFFFFFFFFFFFFFFFFFFFFF

E = Exponent has a bias of 127
F = Fraction with implied 1 left of the binary point

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
```

Composites with Unconstrained Elements

Arrays with Unconstrained Array Elements

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

```
signal A : std_logic_matrix(5 downto 0)(7 downto 0) ;
```

Records with Unconstrained Array Elements

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

```
signal B : complex (re(7 downto 0), im(7 downto 0)) ;
```

Process (all)

- Creates a sensitivity list with all signals on 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 ;
```

- Benefit: Reduce mismatches between simulation and synthesis

---

Types: New Array Types

```
type integer_vector is array (natural range <>) of integer ;
type real_vector is array (natural range <>) of real ;
type time_vector is array (natural range <>) of time ;
type boolean_vector is array (natural range <>) of boolean ;
```

- Allows emulation of argv by use of unconstrained arrays

```
function sum ( A : integer_vector ) return integer is
  variable result : integer := 0 ;
begin
  for I in A'range loop
    result := result + A(I) ;
  end loop ;
  return result ;
end function sum ;
```

```
Signal A, B : integer ;
. . .
A := Sum ( ( 1, 5, 9) ) ;
B := Sum ( ( 7, 15, 2, 23, 4, 8) ) ;
```

Types: Enhanced Std_logic_vector

```
subtype std_logic_vector is (resolved) std_ulogic_vector ;
```

- Allows easy connection between std_ulogic_vector and std_logic_vector

```
signal A_slv  : std_logic_vector(7 downto 0) ;
signal B_sulv : std_ulogic_vector(7 downto 0) ;
. . .

A_slv <= B_sulv ;
```

- Also removes the need to provide overloading for both std_ulogic_vector and std_logic_vector in the packages

---

ENV package library STD

```
package ENV is
  procedure STOP ( STATUS: INTEGER );
  procedure STOP ;

  procedure FINISH ( STATUS: INTEGER );
  procedure FINISH ;

  function RESOLUTION_LIMIT return DELAY_LENGTH;

end package ENV;
```

Stop simulator like breakpoint

Stop simulator and do not continue

Simulator resolution

- Usage:

```
use std.env.all ;
. . .
TestProc : process
begin
  . . .
  stop(0) ;
end process TestProc ;
```

```
TestProc : process
begin
  . . .
  std.env.stop(0) ;
end process TestProc ;
```

Package Integration

- Following packages integrated into IEEE 1076
  - std.standard
  - std.env - new
  - ieee.std_logic_1164 - updated
  - ieee.math_real
  - ieee.math_complex
  - ieee.numeric_std - updated
  - ieee.numeric_std_unsigned - new, unsigned math for std_ulogic_vector
  - ieee.fixed_generic_pkg - new
  - ieee.fixed_pkg - new, an instance of the generic fixed package
  - ieee.float_generic_pkg - new
  - ieee.float_pkg - new, an instance of the generic float package

---

Functions: IS_X, TO_X01

- IS_X for all std_ulogic based types

```
function IS_X (S : T) return BOOLEAN;
```

- Strength strippers for all std_ulogic based types

```
-- originally only in std_logic_1164
function TO_X01  (S : T)  return T;
function TO_X01Z (S : T)  return T;
function TO_UX01 (S : T)  return T;

-- originally only in numeric_std
function TO_01 (S : T; XMAP : STD_ULOGIC := '0') return T;
```

Operators: Unary Reduction

- 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:

```
signal Data : std_logic_vector(7 downto 0) ;
signal Parity : std_logic ;
. . .
Parity <= xor Data ;
```

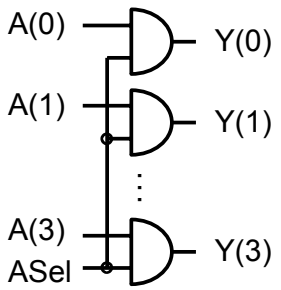
- Calculating Parity without reduction operators:

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

Operators: Array / Bit Logic

- For all binary logic operators (and, or, …)

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



When ASel = '0', it represents "0000"
When ASel = '1', it represents "1111"

- Application: Data read back logic

```
signal ASel, BSel, CSel, DSel : std_logic ;
signal DataOut, AReg, BReg, CReg, DReg
       : std_logic_vector(3 downto 0) ;
. . .
DataOut <= (AReg and ASel) or (BReg and BSel) or
           (CSel and CReg) or (DSel and DReg) ;
```

Operators: Array / Bit Addition

- Overload "+" and "-" for all math types:

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

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

Slices in Array Aggregates

- 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 by either of the following:

```
signal Y9 : unsigned(8 downto 0) ;
. . .
Y9 <= ('0' & A) + ('0' & B) ;
Y <= Y9(7 downto 0) ;
CarryOut <= Y9(8) ;
```

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

Operators: Maximum / Minimum

- Defined for all scalar, discrete array types, and numeric std_logic type*

```
function minimum (L, R: T) return T;
function maximum (L, R: T) return T;
```

  - All types in std.standard except real_vector or time_vector
  - * numeric std_logic type = unsigned, signed, sfixed, ufixed, float

- Defined for single dimensional array types, T, whose element, E, is a scalar

```
function minimum (A: AT) return ET;
function maximum (A: AT) return ET;
```

  - Integer_vector, real_vector, time_vector, ...

- Used in a constant:

```
procedure MemInit (AddrBits, DataBits : integer) is
  constant BLK_ADJ : integer := minimum(BLK_BITS, AddrBits);
  subtype AT is MemArrayType(0 to 2**(AddrBits-BLK_ADJ)-1) ;
begin
```

Operators: Matching Relational

- New relational operators: ?=, ?/=, ?>, ?>=, ?<, ?<=
  - return element values (bit, std_ulogic, …)
  - Understands std_ulogic values and returns UX01

- ?=, ?/=
  - understands '-' as don't care
  - defined for std_ulogic & 1 dimensional arrays of std_ulogic

```
DevSel1 <= Addr?=X"A5" and Cs1 and not nCs2 ;
```

- ?>, ?>=, ?<, ?<=
  - Defined for bit and std_ulogic
  - Not implicitly defined like >, >=, <, <=
  - Overloaded in numeric packages

Simplified Conditional Expressions

- For all conditional expressions (if, when, exit, …)
  - If entire conditional is bit or std_ulogic, then implicitly call the condition operator: ??

```
signal Cs1, nCs2, Cs3 : std_logic ;
. . .
if (Cs1 and not nCs2 and Cs3) then
```

- Use matching relationals with arrays: ?=, ?/=, ?>, ?>=, ?<, ?<=

```
signal Addr : std_logic_vector(7 downto 0) ;
. . .
if (Addr?=X"A5" and Cs1 and not nCs2) then
```

- Condition operator can be called directly:

```
signal Clk  : std_logic ;
signal edge : boolean ;
. . .
Edge <= rising_edge(Clk) and ?? (Cs1) ;
```

Simplified Case Statement

```
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 ;
```

Now a Globally Static Type

Still a Locally Static expression, however

- Locally static now includes operators and functions that:
  - have composite results and/or
  - are defined in std_logic_1164, numeric_std, or numeric_std_unsigned

Simplified Case Statement

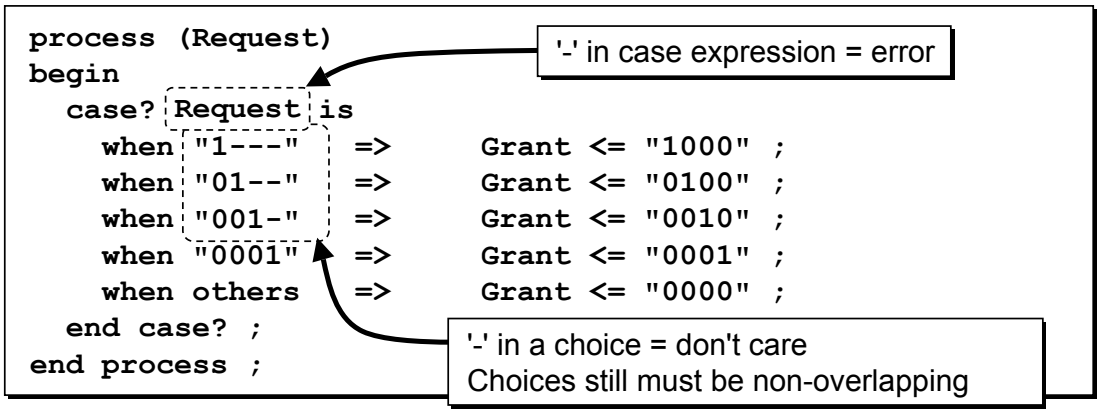
- Although concatenation is allowed, some cases still require a type qualifier.

```
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? = Case With Don't Care

- For std_ulogic or arrays of std_ulogic, '-' represents don't care



- There is a corresponding select?

```
with Request select?
  Grant <= "1000" when "1---",
           "0100" when "01--",
           "0010" when "001-",
           "0001" when "0001",
           "0000" when others ;
```

Enhanced Bit String Literals

- 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 (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

```
7SX"XX"  =  "XXXXXXX"
9UX"ZZ"  =  "0ZZZZZZZZ"
9SX"ZZ"  =  "ZZZZZZZZZ"
```

Printing: Hwrite, Owrite, Swrite, Hread, ...

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

```
procedure hwrite (
      Buf              : inout Line ;
      VALUE            : in bit_vector ;
      JUSTIFIED        : in SIDE  := RIGHT;
      FIELD            : in WIDTH := 0
) ;
procedure hread (
      Buf              : inout Line ;
      VALUE            : out bit_vector ;
      Good             : out boolean
) ;
procedure owrite ( . . . ) ;
procedure oread ( . . . ) ;
procedure swrite ( . . . ) ;  -- string
procedure sread ( . . . ) ;
```

- No new packages.
  - Supported in base packages (std.standard, ieee.std_logic_1164, …)
  - For backward compatibility, std_logic_textio contains aliases

Printing: To_String, To_HString, To_OString

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

```
function to_string (
      VALUE           : in std_logic_vector;
) return string ;
function to_hstring ( . . . ) return string ;
function to_ostring ( . . . ) return string ;
```

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

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

Assignments: Extended Conditional

- In VHDL-2002, a conditional in a process requires an if statement:

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

- VHDL-2008 allows:
  - Conditional signal assignment in sequential code:

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

  - Conditional variable assignment in sequential code:

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

Assignments: Extended Selected

- VHDL-2008 allows selected assignment in sequential code:

```
signal A, B, C, D, YReg : std_logic ;
signal MuxSel : std_logic_vector(1 downto 0) ;
. . .
Process(clk)
begin
  if rising_edge( Clk ) then
    with MuxSel select
      Mux :=
        A when "00",
        B when "01",
        C when "10",
        D when "11",
       'X' when others ;

    YReg <= nReset and Mux ;
  end if ;
end process ;
```

Assignments: Force and Release

- Forcing a port or signal:

```
A <= force '1' ;
```

  - For in ports and signals this forces the effective value
  - For out and inout ports this forces the driving value

- Releasing a signal:

```
A <= release ;
```

Assignments: Hierarchical Reference

- Direct hierarchical reference:

```
A <= <<signal .tb_top.u_comp1.my_sig : std_logic_vector >>;
```

  - Specifies object class (signal, shared variable, constant)
  - path (in this case from top level design)
  - type (constraint not required)

- Note an object must be elaborated before the reference is elaborated
  - Designs are elaborated in order of instantiation
  - As a result, later designs may reference into earlier ones

- Using an alias to create a local short hand:

```
alias u1_my_sig is <<signal u1.my_sig : std_logic_vector >>;
```

  - Here, path refers to component instance u1 (subblock of current block).
  - Can also go up from current level of hierarchy using "^"

Enhanced Port Maps

Expressions in Port Maps

```
U_CHIP : CHIP port map ( A, Y and C, B) ;
```

- If the expression contains a signal and is not a conversion,
  - it is converted to a concurrent signal assignment
  - and it will incur a delta cycle delay
- Needed to avoid extra signal assignments with OVL

Reading Output Ports

- Value read will be locally driven value
- Allows assertions to read out ports without creation of additional signals

Context Declaration

- Primary design unit that groups packages references into a single name

```
Context project1_Ctx is
  library ieee, YYY_math_lib ;
  use std.textio.all ;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all ;
  use YYY_math_lib.ZZZ_fixed_pkg.all ;
end ;
```

- Reference the named context unit

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

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

Enhanced Generics

- Formal Type and Subprogram Generics + Packages with Generic Clause

```
package ScoreBoardPkg is
  generic (
    type BaseType ;
    function check(A, E : BaseType) return boolean
  ) ;
  . . .
end ScoreBoardPkg ;
```

- Specify generics in a package instance to create a new package

```
library IEEE ;
  use ieee.std_logic_1164.all ;
package ScoreBoardPkg_slv8 is new work.ScoreBoardPkg
  generic map (
    BaseType => std_logic_vector(7 downto 0),
    check => std_match ) ;
```

Block Comments

- Added "C" block comments: "/*" and "*/"

```
/* Remove the following code
if (FP = '1') then
    NextState  <= FLASH ;
else
    NextState  <= IDLE ;
end if ;
*/
```

- Recommendation, only use this for temporary edits.
- For permanent edits, clarity is more important, so instead,
  - Use a good editor
  - Select the region of code
  - Comment out selected region using "--"

Resulting Operator Overloading

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

Std Logic_1164 Updates

- A few items that were updated:
  - std_logic_vector is now subtype of std_ulogic_vector
  - Uncomment xnor operators
  - Add logical shift operators for vector types
  - Add logical reduction operators
  - Add array/scalar logical operators
  - Added text I/O read, oread, hread, write, owrite, hwrite
    - No longer need ieee.std_logic_textio
    - Std_logic_textio modified to be peacefully co-exist if referenced

---

Numeric Std Updates

- A few items that were updated in numeric_std are:
  - Array / scalar addition operators
  - TO_X01, IS_X for unsigned and signed
  - Logic reduction operators
  - Array / scalar logic operators
  - TextIO for numeric_std
- Added numeric_std_unsigned package
  - Replaces std_logic_unsigned
  - Language modified to allow explicit operators to always overload implicit ones - even if in a different package
  - Overloads for std_ulogic_vector/std_logic_vector to have all of the operators defined for ieee.numeric_std.unsigned
  - Subprograms (especially conversions and extensions) are consistent with numeric_std

Next Steps for VHDL

- Encourage your EDA vendor(s) to support VHDL-2008 and beyond.
  - Many are well into their implementation - keep encouraging them
- Next Steps, add:
  - Functional coverage
  - Constrained random stimulus generation
  - Verification data structures (FIFOs, scoreboards, memories, …)
  - Direct C and Verilog/SystemVerilog Calls
  - Object oriented constructs - extend protected types - have initial proposal
- VHDL community needs the next steps to
  - Re-use and extend our current testbenches
  - Keep our design and verification teams using the same language
  - Use a language with consistent syntax
- Participate in VHDL standards. See: http://www.eda.org/vasg
- Seeking funding and/or funding model.

SynthWorks VHDL Training

SynthWorks is committed to supporting VHDL standards
Support us by buying your training from us.

Comprehensive VHDL Introduction 4 Days
http://www.synthworks.com/comprehensive_vhdl_introduction.htm
A design and verification engineer's introduction to VHDL syntax, RTL coding, and testbenches. Students get VHDL hardware experience with our FPGA based lab board.

VHDL Testbenches and Verification 5 days - OS-VVM Boot Camp
http://www.synthworks.com/vhdl_testbench_verification.htm
Learn the latest VHDL verification techniques including transaction-based testing, bus functional modeling, self-checking, data structures (linked-lists, scoreboards, memories), directed, algorithmic, constrained random and coverage driven random testing, and functional coverage.

VHDL Coding for Synthesis 4 Days
http://www.synthworks.com/vhdl_rtl_synthesis.htm
Learn VHDL RTL (FPGA and ASIC) coding styles, methodologies, design techniques, problem solving techniques, and advanced language constructs to produce better, faster, and smaller logic.

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