VHDL-2008,
The End of Verbosity!

by

Jim Lewis

SynthWorks VHDL Training

Jim@SynthWorks.com

SynthWorks

VHDL-2008, The End of Verbosity!

This material is derived from SynthWorks' VHDL classes

This material is updated from time to time and the latest copy of this is available at
http://www.SynthWorks.com/papers

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

www.SynthWorks.com

VHDL-2008, The End of Verbosity!

- Many think of VHDL as the Verbose HDL.
- With VHDL-2008 this is no longer the case.

Topics

- Simplified Sensitivity List
- Simplified Condition (if, while, …)
- Matching Relational Operators
- Simplified Case Statement
- Sequential Conditional Assignment
- Unary Reduction Logic Operators
- Array / Bit Logic Operators
- Array / Bit Addition Operators
- Simplified Printing with to_string
- Expressions In Port Maps
- Reading Out Ports
- Enhanced Bit String Literals
- Mod for Physical Types (Time)
- Context Declarations
- Slices in Array Aggregates

Simplified Sensitivity List - Process (all)

- Prior to 2008, all inputs to a combinational logic process need to be on the sensitivity list

```
Mux1_proc : process( MuxSel, A, B, C, D )
begin
. . .
```

- VHDL-2008 allows the use of keyword "all" in place of signals

```
Mux2_proc : process(all)
begin
```

Simplified Condition (if, while, …)

- Prior to 2008, a condition (expression in if, while, …) was required to have a boolean result.
  - Our code was plagued with "="

```
if  Cs1 = '1' and nCs2 = '0' and Cs3 = '1'  then
```

- VHDL-2008 allows condition to have a bit or std_logic result

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


Matching Relational Operators

- Prior to 2008, decoding arrays required conditionals

```
DevSel <= '1' when
   Addr = X"A5" and Cs1 = '1' and nCs2 = '0'
   else '0' ;
```

- 2008 adds relational operators: ?=, ?/=, ?>, ?>=, ?<, ?<=
  - return element values (bit, std_ulogic, …)
  - Simplifies decoders

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

  - Harmonizes with Simplified Condition

```
if (Addr ?= X"A5" and Cs1 and not nCs2) then
```

Matching Relational Operators

- Prior to 2008, there was no relational that understood '-'
  - As a result, '-' had to be factored out of an expression

```
DevSel <= '1' when
    Addr(7 downto 5) = "111" and
    Addr(2 downto 0) = "000"
    and Cs1 = '1' and nCs2 = '0'
    else '0' ;
```

- VHDL-2008, ?= and ?/= understand '-' as don't care
  - Defined for std_ulogic and 1 d arrays of std_ulogic

```
DevSel <= Addr ?= "111--000" and Cs1 and not nCs2 ;
```


Simplified Case Statement

- Prior to 2008, expressions required intermediate objects

```
Y := A xor B ;
case Y is
```

- With VHDL-2008, expressions are permitted

```
case  A xor B  is
```

- With VHDL-2008, locally static expressions now include
  - Operations on arrays (such as std_logic_vector)
  - Operators defined in std_logic_1164, numeric_std

```
constant CHOICE1 : std_logic_vector := "11" & "00" ;
. . .
case A xor B is
  when CHOICE1     =>     ...
  when "00" & "11" =>     ...
```

Sequential Conditional Assignment

- Prior to 2008, a conditional in a process required "if"

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

- VHDL-2008 simplifies code by allowing:
  - Conditional signal assignment in sequential code:

```
NextState <= FLASH when FP else IDLE ;
```

  - Conditional variable assignment in sequential code:

```
NextState := FLASH when FP else IDLE ;
```

Unary Reduction Logic Operators

- Prior to 2008, Calculating Parity required:

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

- VHDL-2008 adds Unary Reduction Operators of the form:

```
function "xor"  ( anonymous: BIT_VECTOR) return BIT;
```

  - Defined for arrays of bit and std_ulogic
  - Defined for all binary logic operators:
    - AND, OR, XOR, NAND, NOR, XNOR

- Simplifies Parity Calculation

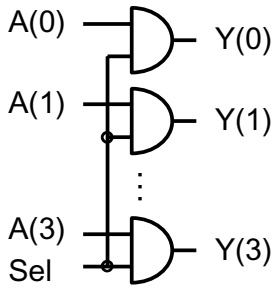
```
signal Data : std_logic_vector(7 downto 0) ;
signal Parity : std_logic ;
. . .
Parity <= xor Data ;
```

Array / Bit Logic Operators

- Prior to 2008, and'ing a bit with an array

```
signal Sel : std_logic ;
signal Y, A, vSel : unsigned(3 downto 0);
. . .
Y <=  A when Sel = '1' else "0000" ;
```

```
vSel <= (others => Sel) ;
Y    <=  A and vSel ;
```



- VHDL-2008 adds Array / Bit Logic Operators

```
function "and"( L: BIT_VECTOR; R: BIT) return BIT_VECTOR;
function "and"( L: BIT; R: BIT_VECTOR) return BIT_VECTOR;
```

  - Defined for arrays of bit and std_ulogic

- Simplifies to:

```
Y <=  A and Sel ;
```


Array / Bit Logic Operators

- Prior to 2008, data read back logic:

```
process ( A, ASel, B, BSel, C, CSel, D, DSel )
begin
  if ASel = '1' then
    DO <= A ;
  elsif BSel = '1' then
    DO <= B ;
  elsif CSel = '1' then
    DO <= C ;
  elsif DSel = '1' then
    DO <= D ;
  . . .
```

- VHDL-2008, simplifies data read back logic

```
DO <= (AReg and ASel) or (BReg and BSel) or
      (CSel and CReg) or (DSel and DReg) ;
```

Array / Bit Addition Operators

- Prior to 2008, addition only for arrays
  - May result in two separate adders

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

- 2008 adds Array / Bit Addition Operators

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

  - "+" and "-" defined for all array based math types

- Simplify coding Carry In

```
Y <= ('0' & A) + ('0' & B) + CarryIn ;
```

Simplified Printing with TO_String

- Prior to 2008, string conversion was handled with 'image
  - Only supported scalars. No overloading.

- VHDL-2008 adds string conversions for all types

```
function to_string ( VALUE : unsigned ) return string;
```

- Hex and Octal string conversions for bit based arrays
  - to_hstring, to_ostring

- Simplifies usage of report statement or built-in write

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

Expressions in Port Maps

- Prior to 2008, expressions on inputs require separate signals:

```
Temp <= Y and C ;

U_CHIP : CHIP port map ( A, Temp, B) ;
```

- VHDL-2008 allows expressions on inputs.
  - Executes same as a separate signal.

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

Reading Out Ports

- Prior to 2008, reading out ports required an extra signal:

```
i_Y  <= A and B ;
Y    <= i_Y ;   -- out port

W    <= i_Y and C ;
```

  - Viewed as an output after a chip IO cell
  - Provided minimal benefit at top level of design

- VHDL-2008 allows reading of Out Ports.

```
Y    <= A and B ; -- out port

W    <= Y and C ;
```

  - Value read will be locally driven value

Enhanced Bit String Literals

- Prior to 2008, hex bit string literals were a multiple of 4 bits

```
Addr(6 downto 0)  <=  "111" & X"F" ;
```

- VHDL-2008, simplifies bit string literals by adding

```
7X"7F"    = "1111111"

7UX"F"   = "0001111" -- extend
7UX"0F"  = "0001111" -- reduce
7UX"8F"  = "0001111" -- error

7SX"F"   = "1111111" -- extend
7SX"CF"  = "1001111" -- reduce
7SX"8F"  = "0001111" -- error

7SX"-X"  = "---XXXX"

7D"127" =  "1111111"
```

Lengths

Unsigned notation (default)
Extend: 0 fill LHS
Reduce: ok to drop 0 on LHS

Signed notation
Extend: replicate sign bit
Reduce: ok to drop sign bit

-, X, Z handling (*S)

Decimal values

Mod for Physical Types (time)

- Prior to 2008, Mod for Time Required a Calculation

```
Phase_int := (NOW/1 ns) mod (tperiod_wave/1 ns) ;
```

  - To big for type integer when time >= 2**31 * 1 ns

- 2008 defined Mod for Physical Types

```
function "mod" (anonymous : time) return time;
```

  - Implicitly defined for all physical types

- Simplifies periodic waveform phase calculation generation

```
phase     := NOW mod tperiod_wave ;
```

Context Declarations

- Prior to 2008, all designs referenced a set of packages

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

- VHDL-2008 adds context declarations

```
Context rtl_ctx is
  library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all ;
end ;
```

- Reference the named context unit

```
Library Lib_P1 ;
  context Lib_P1.rtl_ctx ;
```

Slices in Array Aggregates

- Prior to 2008, coding an adder with separate carry out:

```
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) ;
```

- 2008 allows slices in an array aggregate

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

. . .

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

VHDL-2008 Summary

- Readability and capability have increased
- Code is simpler and more concise.
- Verbosity is gone.
- Strong typing is still present.

- For more on VHDL-2008 see:
  - http://www.SynthWorks.com/blog

SynthWorks VHDL Training

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 bootcamp
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.

SynthWorks offers on-site, public venue, and on-line classes. See:
http://www.synthworks.com/public_vhdl_courses.htm