VHDL-2008: Why It Matters

by Jim Lewis, SynthWorks VHDL Training

1. INTRODUCTION

VHDL-2008 (IEEE 1076-2008) is here! It is time to start using the new language features to simplify your RTL coding and facilitate the creation of advanced verification environments.

VHDL-2008 is the largest change to VHDL since 1993. An abbreviated list of changes includes:

- Enhanced Generics = better reuse
- Assertion language (PSL) = better verification
- Fixed and floating point packages = better math
- Composite types with elements that are unconstrained arrays = better data structures
- Hierarchical reference = easier verification
- Simplified Sensitivity List = less errors and work
- Simplified conditionals (if, ...) = less work
- Simplified case statements = less work

This article overviews the changes and the value they bring to your design process. Topics are categorized into three major sections: testbench, RTL, and packages/operators.

2. TESTBENCH

Through extended and new capability, VHDL-2008 enables the creation of advanced verification environments. The following subsections examine these changes and the value they deliver.

2.1 Enhanced Generics

Enhanced generics are one of the most significant changes to the language. Prior to 2008, generics could only be specified on an entity and were only allowed to be constants. VHDL-2008 allows specification of generics on packages and subprograms, and allows types, subprograms, and packages to be generics.

This means generics can be used to facilitate parameterization and reuse of packages and subprograms. This is particularly important for verification data structures, such as a scoreboard. Scoreboards keep an internal store of transmitted values to be compared with received values. In the code below, the package interface has a generic type for the expected (transmitted) value and actual (received) value, as well as a generic function to compare these values.

```
package ScoreBoardGenericPkg is
  generic (
    type ExpectedType ;
    type ActualType ;
    function check(A : ActualType; E: ExpectedType)
      return boolean ;
    . . .
  ) ;
  . . .
end ScoreBoardGenericPkg;
```

A generic package or subprogram must be instantiated before it can be referenced or used. The following package instance creates a scoreboard package for use with type std_logic_vector and comparison operator "?=" (see packages/operators).

```
library ieee ;
use ieee.std_logic_1164.all ;
package ScoreBoardPkg_slv is new
work.ScoreBoardGenericPkg
  generic map (
    ExpectedType    => std_logic_vector,
    ActualType      => std_logic_vector,
    check           => "?=",
    . . .
) ;
```

2.2 Assertion Language (PSL)

An assertion language improves verification by providing expressive syntax to detect design or interface conditions that either must happen (coverage) or must not happen (assertions). The conditions may be either static (things happening during a single clock period) or dynamic (sequences of events over multiple clock periods). These conditions are checked either dynamically during simulation or statically using formal verification techniques. By specifying the conditions within the design, visibility into the design's internal state can be gained.

Rather than develop a VHDL specific assertion language, VHDL integrates IEEE standard 1850, Property Specification Language (PSL). Since PSL has standard "flavors" for other HDL and verification languages, it simplifies mixed language environments.

As a result, PSL declarations (sequences and properties) are VHDL block declarations and may be put into packages, and the declarative part of an entity, architecture, or block statement. PSL directives (assert and cover) are VHDL statements and are permitted in any concurrent statement part. PSL design units (vunit, vprop, and vmode) are VHDL primary units and may include a context clause prior to the vunit.

Currently QuestaSim supports PSL within comment fields and not directly within VHDL code.

2.3 Hierarchical Reference

VHDL-2008 external names simplify verification by providing both observation and control access to signals, shared variables or declared constants in other portions of the design. This allows a testbench to supply a value for a missing feature, or read and check values in an embedded memory.

Objects can either be accessed directly or with aliases, such as the one shown below. Note that the object being referenced (.tb_top.u_ioc.int1) must be elaborated before the external name reference (alias) is elaborated.

```
alias int1  <<signal .tb_top.u_ioc.int1 : std_logic>>;
```

2.4 Force / Release

VHDL-2008 adds force and release. Force and release are modifiers to an assignment that allow the value supplied to override a value driven by other parts of a design. These are intended to be temporary overriding values used by a testbench while debugging a design or waiting for a fix to a design, and not a permanent part of a design or testbench.

```
int1 <= force '1' ;
…
int1 <= force '0' ;
…
Int1 <= release ;
```

2.5 Composites with Unconstrained Arrays

To facilitate reusable and/or standard matrix (or multidimensional) operations, VHDL-2008 extends composites (arrays or records) to allow their elements to be unconstrained arrays.

The following example shows a multidimensional array structure (implemented as an array of an array type) being defined as MatrixType within a package, and then used on an entity interface and within the architecture.

```
package MatrixPkg is
  type MatrixType is array (natural range <>)
    of std_logic_vector ;
  …
end package MatrixPkg ;
use work.MatrixPkg.all ;
entity e is
port (
  A : out MatrixType (7 downto 0)(5 downto 0) ;
  . . .
) ;
Architecture a of e is
signal B : MatrixType (7 downto 0)(5 downto 0) ;
begin
. . .
B(5) <= "111000" ;    -- Accessing a Row
A(7)(5) <= '1' ;       -- Accessing an Element
. . .
```

Records with unconstrained elements are useful for creating reusable data structures. The Open Source VHDL Verification Methodology (OSVVM) package, CoveragePkg, uses this to create the base type for functional coverage (point and cross) modeling.

2.6 Better Printing: Write, Read, …

To facilitate printing using TEXTIO, read and write procedures have been added for all standard types. Except for the package std.standard, overloading for read and write is in the package that defines the type. This way it is not necessary to include additional packages to be able to print.

Hexadecimal and octal printing procedures (hwrite, hread, owrite, and oread) were added for types that are arrays of a bit type (std_logic or bit).

To enable printing to both a file and the screen (OUTPUT), the procedure tee was added.

To improve string handling, the procedures sread and swrite were added. Sread reads non-space tokens. It skips any leading white space. It stops reading when either argument'length characters or white space is read. Swrite allows strings to be written without the necessity to use a type qualifier.

2.7 Better Printing: To_string functions

To facilitate printing using report statements or VHDL's built-in write statement, string conversions, named to_string, were created for all types. In addition, for logic based array types hexadecimal and octal string conversions were created (to_hstring and to_ostring). Using to_string with VHDL's built in write provides a simpler call syntax, much more similar to that provided by "C".

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

2.8 Stop

To enable a testbench to stop the simulation when it finishes generating and checking stimulus a stop procedure was added. Stop removes the necessity to keep track of how long a simulation runs. Stop is in the VHDL-2008 env package in the library std. The following example shows a call to stop where the package was not referenced in a use clause.

```
std.env.stop(0) ;
```

2.9 Context Unit

A context unit enables a design to reference a set of packages with a single reference. When used by an entire design team, a context unit can ensure that all designs use the same set of project approved packages. An example of a context unit is shown below.

```
Context ProjectCtx is
      use std.textio.all ;
      use std.env.all ;
   library ieee  ;
      use ieee.std_logic_1164.all;

      use ieee.numeric_std.all ;
end ;
```

A design references a context unit as follows.

```
context work.ProjectCtx ;
```

2.10 Expressions in Port Maps

To simplify entity/component instances, port associations now allow expressions in a port map. The expression eliminates the need to create extra signal assignments.

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

From a language perspective, the expression is treated as if there were a signal assignment and it incurs a simulation cycle delay (same as would happen with an explicit signal assignment).

3. RTL

VHDL-2008 enhancements simplify RTL coding.
Among these are changes to sensitivity lists, conditionals (if statements), and case statements. The following subsections examine the RTL changes and the value they deliver. Note while these will work in your simulator, also take care to try these in your synthesis tool(s) before using them extensively.

3.1 Simplified Sensitivity List: Process (all)

One of the most common errors in combinational logic is forgetting a signal in a process sensitivity list. “Process(all)” eliminates this problem by implicitly including all signals that are read in the process on the sensitivity list. For subprograms called by the process, this includes signals read by side-effect when the subprogram is declared in the same design unit as the process. 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
```

3.2 Simplified Conditionals (If, While, …)

Prior to 2008, conditional expressions were required to be boolean and were plagued by relational (comparison) operations, as shown below:

```
-- Old, Prior to 2008 code:
if (Cs1=’1’ and nCs2=’0’ and Cs3=’1) then
```

Conditional expressions were simplified by also allowing the entire expression to evaluate to a bit type such as std_logic or bit. Hence the above code simplifies to:

```
-- New
if (Cs1 and nCs2 and Cs3) then
```

3.3 Matching Relational Operators

The new matching relational operators (“?=”, “?/=”, “?<”, “?<=”, “?>”, and “?>=”) return bit values (bit or std_ulogic), and as a result, are better suited for hardware design than the older ordinary relational operators.

Using “?=”, decoding of mixed bit and array signals can be simplified to the following. Note that in addition to returning bit values, “?=” and “?/=” operators also understand ‘-’ as don’t care.

```
Reg1Sel <= Cs1 and not nCs2 and Addr?= “1010--” ;
```

Matching relational operators further simplify conditional expressions by facilitating array comparisons such as the following.

```
if Cs1 and not nCs2 and Addr?= “1010--” then
   …
```

The matching ordering operators ("?<", "?<=", "?>", and "?>=") are only defined for array types that also support numeric operations. This avoids errors caused by implicit dictionary style relationals ("<", "<=", ">", and ">=") that are present with std_logic_vector when the package numeric_std_unsigned is not used.

3.4 Simplified Case Statements

Prior to 2008, case statement rules made most expressions within either the case select or choice expressions illegal. VHDL-2008 removes many of these limitations and simplifies the usage of case statements. The changes are illustrated in the following example.

```
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   -- 2008
    when "0000" =>            Y <= "00" ;
    when CHOICE2 =>           Y <= "01" ;  -- 2008
    when "0110" =>            Y <= "10" ;
    when ONE1 & "00 =>        Y <= "11" ;  -- 2008
    when others =>            Y <= "XX" ;
  end case ;
end process ;
```

The following is what has changed. Case select expressions now only need to have a globally static type. The definition of locally static no longer excludes operations on arrays (such as std_logic_vector or unsigned). All the operators from the standard packages (that are part of 1076) can be part of locally static expressions.

3.5 Case With Don't Care

Some use models of case statements benefit from the usage of don't care characters. VHDL-2008 adds a matching case statement, "case?" that uses "?=" to determine equality, and hence, understands '-' as don't care. Using "case?", basic arbitration logic can be created as follows. Note that each case choice must still be non-overlapping.

```
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 ;
```

Note that the ordinary case statement handles '-' as an ordinary character which is important for many applications.

3.6 Extended Conditional Assignment

Prior to 2008, evaluation of a condition within a process required an if statement such as shown below.

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

VHDL-2008 simplifies the above code by allowing conditional assignment, such as the one shown below, to be used with either signals or variables. The result is shorter, more readable code, such as statemachines.

```
NS1 <= FLASH when (FP = '1') else IDLE ;
```

3.7 Extended Selected Assignment

Selected assignment provides a shorthand for a case statement when only one data object is being targeted. Prior to 2008, selected assignment could only be used for signals in a concurrent code region. VHDL-2008 allows it to be used within a process with either signals or variables (as shown below).

```
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 ;
```

3.8 Enhanced Bit String Literals

Prior to 2008, hexadecimal bit string literals values were always multiples of 4 bits, and hence, challenging to work with.

VHDL-2008 simplifies working with hexadecimal bit string literals values by adding an integer length value prior to the base specifier. Bit string values can either be extended or reduced by the length specified provided that the numeric value does not change. By default, bit string literals are unsigned. An additional prefix character of S for signed or U for unsigned can also precede the base specifiers B, O, and H. If a non-hexadecimal character, such as '-' is included in the string, it will be replicated four times in a hexadecimal. A few examples are shown below. Also added is a decimal base specifier (D).

```
--  Expanding a value
7X"F"   =  "0001111"   -- unsigned fill with 0
7UX"F"            =  "0001111"   -- same as above
7SX"F"            =  "1111111"   -- signed replicate sign
-- Reducing a value
7UX"0F"           =  "0001111"   -- ok.  Same value
7SX"CF"           = "1001111"    -- ok.  Same value
7UX"8F"           =  "0001111"   -- error.  Value changed
7SX"8F"           =  "001111"    -- error.  Value change.
-- repeating X and - characters:
X"-X"   =  "----XXXX"
-- Decimal values.  Requires length.  Always unsigned
8D"15"            =  "00001111"
```

3.9 Slices and Array Aggregates

VHDL-2008 allows array aggregates to include array slices. Hence, the following is legal

```
signal A, B, Sum : unsigned(7 downto 0) ;
signal CarryOut : std_logic ;
…
(CarryOut, Sum) <= ('0' & A) + B ;
```

3.10 Generate: Else and Case

VHDL-2008 extends the "if generate" statement to support "else" and "elsif" clauses and adds a "case" generate statement.

These features will be supported in an upcoming version of QuestaSim in 2013.

3.11 Block Comments

VHDL-2008 adds "C" like multiline comments that start with "/*" and end with "*/".

3.12 Read Out Ports

Prior to 2008, out ports of an entity could not be read. This restriction was intended to prevent reading the output side of a chip output. As a result, it provided a minimal benefit at the top level of a design that also instantiates IO cells (common in ASIC design flows, but not FPGA).

RTL designers have been working around this issue for years by adding extra internal signals to specifically read a value internally. As testbench designers add assertions, however, often they are not permitted to modify the RTL design. As a result, this rule was neither useful nor practical to maintain.

3.13 IP Protection

Intellectual Property (IP) protection simplifies the process of distributing protected source code. This mechanism allows IP providers to provide source code that is hidden from viewing, but is still able to be processed by EDA tools. In addition while the code is being handled within the tools, its intermediate form is restricted from being viewed by users. This minimizes IP supplier concerns of the IP being reverse engineered while being used in EDA tools. The approach is based on and consistent with work in the IEEE P1735 working group.

4. PACKAGE AND OPERATOR UPDATES

VHDL's support for math types and operations is unmatched by other languages. With VHDL-2008, VHDL becomes the only RTL language supporting fixed and floating point types and operations.

In addition there were new operators added, and tune ups to the packages and how the packages are integrated into the language. The following subsections explore the updates as well as the value they deliver.

4.1 Fixed Point Packages

The new package, fixed_generic_pkg, defines fixed point math types and operations. It defines the types ufixed and sfixed. To support fractional parts, negative indices are used. The index 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. A fixed point number may contain only a fraction or a whole number. The diagram below illustrates how values in a fixed point number work.

```
Format of ufixed (3 downto -3)
Integral part    = bits 3 downto 0
Fractional part  = bits -1 downto -3

constant A : ufixed (3 downto -3) := "0110100" ;
Integral part    = "0110"   = 6
Fractional part  = "100"    =  0.5
Value "0110100"             = 6.5
```

A fixed point addition/subtraction operation has a full precision result. Hence, when adding two numbers with a 4 bit integral part, the result will have a 5 bit integral part. This is shown below.

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

The fixed point package has generics to parameterize the rounding style (round or truncate), overflow style (saturate or wrap), and number of guard bits (for division). The package instance, ieee.fixed_pkg, selects generics round for rounding style, saturate for overflow style, and 3 guard bits. If you need something different, you will need to create your own package instance.

4.2 Floating Point Packages

The new package, float_generic_pkg, defines floating point math types and operations. It defines the type float as well as subtypes for single, double, and extended precision numbers. The index range downto is required. The sign bit is the left most bit. The exponent contains the remaining non-negative indices (including zero). The mantissa (fractional part) is to the right of the zero index. The floating point format can be visualized as follows:

```
Format of float (8 downto -23)
Sign Bit        = Bit 8
Exponent        = Bits 7 downto 0
   has a bias of 127 (2**E'length-1)
Fraction        = Bits -1 to -23
   has an implied 1 in leftmost bit

Float(8 downto -23) value                  Number
0  10000000  00000000000000000000000  =  2.0
0  10000001  10100000000000000000000  =  6.5
0  01111100  00000000000000000000000  =  0.125
```

Floating point operations always produce a result that is the size of its largest operands.

```
signal A, B, Y : float (8 downto -23) ;
. . .
Y <= A + B ;  -- float numbers must be same size
```

The floating point package has generics to parameterize the rounding styles (nearest, positive infinity, negative infinity, zero), denormalized number handling (enabled or not), NAN handling (enabled or not), and number of guard bits (for all operations). The package instance, ieee.float_pkg, selects generics round nearest, denormalize true, NAN handling true, and 3 guard bits. If you need something different (recommended for RTL), then you will need to create your own package instance.

4.3 Package Integration

With VHDL-2008, the packages std_logic_1164, numeric_std, numeric_bit, math_real, and math_complex are part of IEEE 1076. Seems like a trivial change, however, it allows the operators in these packages to be part of locally static expressions. This allows std_logic_vector values or constants to be used in an expression within a case statement choice.

4.4 Package: Numeric_Std_Unsigned

The new package, ieee.numeric_std_unsigned, defines numeric operations for std_ulogic_vector (std_logic_vector). It provides std_ulogic_vector (std_logic_vector) with the same operations that are available in numeric_std for type unsigned.

4.5 Package: Env

The new package, std.env, defines the procedures stop, finish, and the function resolution_limit. The procedure stop stops a simulation and leaves it in a state that can be continued. The procedure finish stops a simulation and leaves it in a state that cannot be continued. The function resolution_limit returns the value of the smallest representable time (set by the simulator).

4.6 Types: Integer_vector, …

VHDL-2008 adds the types integer_vector, real_vector, time_vector, and boolean_vector to the package std.standard. These types are arrays of integer, real, time, and boolean respectively.

Usage of a type like integer_vector on a subprogram interface allows a variable number of integer values to be passed to the subprogram. This is much like "C"s argc/argv capability. This capability is used extensively in the OSVVM packages for coverage modeling (CoveragePkg) and randomization (RandomPkg).

4.7 Resolution Functions & Std_logic_vector

VHDL-2008 enhanced resolution functions so that a resolution function for an element type can be applied to an array of that type. Using this enhancement, std_logic_vector is now a subtype of std_ulogic_vector as shown below.

```
subtype STD_LOGIC_VECTOR is
   (resolved) STD_ULOGIC_VECTOR;
```

With this change, std_logic_vector will automatically convert to std_ulogic_vector (or vice versa). Hence, while numeric_std_unsigned only created overloading for std_ulogic_vector, it supports both std_ulogic_vector and std_logic_vector.

In addition, all of the numeric packages now create both unresolved and resolved numeric array type declarations.

4.8 Unary Reduction Operators

VHDL-2008 creates unary versions of AND, OR, NOR, NAND, XOR, and XNOR for logic array types (bit_vector, std_logic_vector, …). The operators are applied to each element of the array argument (a reduction operation) and produce an element result. Unary operators have the same precedence as the miscellaneous operators (**, ABS, and NOT).

In the following, Parity1 and Parity2 both produce the same result. Parity1 uses the unary "XOR" operator.

```
-- with VHDL-2008
Parity1 <= xor Data and ParityEnable;

-- without
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 ;
```

4.9 Array/Scalar Logic Operations

VHDL-2008 overloads logic operators to support mixing an array argument (std_logic_vector, …) with a scalar (std_ulogic, …) argument. With logic operations, the scalar argument is applied with each element of the array argument. The size of the result matches the size of the array argument. In the following example, the assignments to D1 and D2 are equivalent.

```
signal A, D1, D2 : std_logic_vector(7 downto 0) ;
signal Asel : std_ulogic ;
…
-- with VHDL-2008
D1 <= A and Asel ;

-- without
GenLoop : for I in D2'Range loop -- without
begin
   D2(I) <= A(I) and Asel ;
end generate;
```

These operators also simplify the creation of multiplexers using AND-OR logic as shown below.

```
signal A, B, C, D, Y1 : std_logic_vector(7 downto 0) ;
signal ASel, BSel, CSel, DSel : std_ulogic ;
…
Y1 <=
     (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.

```
Y2 <=
    A when ASel = '1' else  B when BSel = '1' else
    C when CSel = '1' else   D when DSel = '1' else
   (others => '0') ;
```

4.10 Array/Scalar Addition Operations

VHDL-2008 overloads addition operators to support mixing an array argument (unsigned, …) with a scalar (std_ulogic, …) argument. With addition and subtraction operations, the scalar is interpreted as either a numeric 0 or 1. This simplifies operations such as the Add with Carry shown below.

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

Without this, CarryIn must be modeled as an unsigned value and, in synthesis, may result in an implementation with two adders rather than one.

4.11 Maximum / Minimum

VHDL-2008 predefines functions maximum and minimum with arguments that are either scalars or arrays (whose elements are either an enumerated type or integer). Hence, the following is supported.

```
constant V : integer := minimum(10, V_GENERIC) ;
```

VHDL-2008 also predefines maximum and minimum for arrays whose elements are scalars that return the element type. This is illustrated below.

```
constant V : integer_vector := (10, 20, 30) ;
constant V_Min : integer := minimum( V ) ;
```

4.12 IS_X, TO_X01, …

Prior to 2008, the strength reduction functions in std_logic_1164 were different from the ones in numeric_std.

With VHDL-2008 the functions IS_X, TO_X01, TO_X01Z, TO_UX01, and TO_01 are now defined for all standard types based on std_ulogic.

4.13 Shift Operations

Shift operators were added in VHDL-1993 and are implicitly defined for all arrays of bit and boolean. However, due to the interesting definition of sra and sla, the shift operators were never widely implemented in the std_logic family.

VHDL-2008 implements the ror, rol, srl, and sll for arrays of std_logic (in their corresponding package). Sra and sla are only defined when an array of std_logic has a numeric interpretation. For unsigned operations, sra (sla) produces the same result as srl (sll). For signed operations, an sra (or sla with a negative shift) shifts right with the left most value matching the current left most value (replicate the sign bit). For signed operations, an sla (or sra with a negative shift) produces a result that is consistent with sll.

4.14 Mod for Physical Types (Time)

VHDL-2008 implicitly defines mod for all physical types. The following calculation uses mod with type time to calculate the phase of a periodic waveform (testbench application).

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

5. SUMMARY

VHDL-2008 brings new and enhanced features that increase reuse, capability, and productivity, and as a result, simplify coding and facilitate the creation of advanced verification environments.

At this point what is most exciting is that all of the features, except generate, are implemented in QuestaSim release 10.2 and many of them are already available in release 10.1.

6. REVISION PLANS

VHDL-2008 is a good step forward. Work is currently in progress for the next revision. The standard is being revised by the IEEE VHDL Analysis and Standardization Group' (VASG). There is also a VHDL package open source group currently forming.

Some of the work being considered includes.

- Direct Programming Interface
- Simplified Interfaces
- Functional Coverage
- Random Stimulus Generation
- Verification Data Structures (Scoreboards, …)

7. PARTICIPATING IN VHDL STANDARDS

VHDL standards are open for all to participate. Make sure your voice is heard. Come join us. See http://www.eda.org/twiki/bin/view.cgi/P1076/WebHome or alternately start at http://www.eda.org/ and follow the links. There is also an open source VHDL package group that is currently forming.

8. ACKNOWLEDGEMENTS

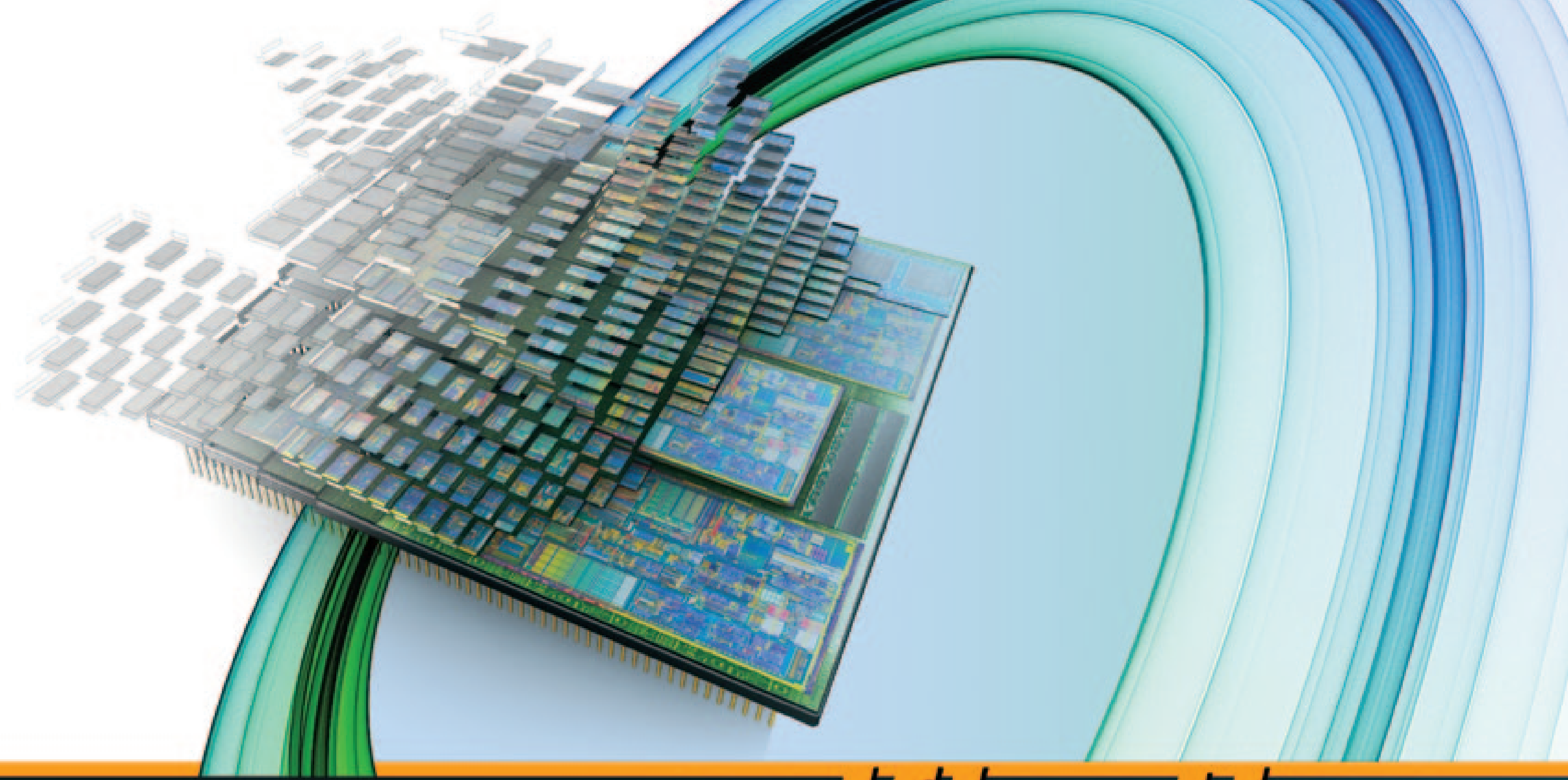
VHDL-2008 was made possible by the hard work of the VASG and Accellera VHDL teams. Thanks to everyone who participated.

9. REFERENCES

Peter Ashenden, Jim Lewis, VHDL-2008: *Just the New Stuff*, Morgan Kaufmann/Elsevier Burlington, MA ISBN 978-0123742490

10. ABOUT THE AUTHOR

Jim Lewis, the founder of SynthWorks, has twenty-eight years of design, teaching, and problem solving experience. Mr. Lewis participated in the VHDL-2008 standards effort and is the current IEEE VHDL Analysis and Standards Group (VASG) chair.



verification HORIZONS

Editor: Tom Fitzpatrick
Program Manager: Rebecca Granquist

Wilsonville Worldwide Headquarters
8005 SW Boeckman Rd.
Wilsonville, OR 97070-7777
Phone: 503-685-7000

To subscribe visit:
www.mentor.com/horizons

To view our blog visit:
VERIFICATIONHORIZONSBLOG.COM

Mentor Graphics®

www.mentor.com