VHDL-2019: Just the New Stuff

Part 3: RTL Enhancements

by
Jim Lewis
VHDL Training Expert at SynthWorks
IEEE 1076 Working Group Chair
OSVVM Chief Architect
Jim@SynthWorks.com

SynthWorks

1

VHDL-2019

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

www.SynthWorks.com

2

2

SynthWorks

VHDL-2019: Just the New Stuff, Part 3: RTL Enhancements

- Agenda
  - VHDL is #1
  - What to Change and What not to Change
  - Conditional expressions
  - Conditional return
  - Inferring signal and variable constraints from initial values
  - All interface lists are ordered
  - Allow functions to know the output vector size
  - Optional semicolon at the end of interface list
  - Component declaration syntax regularization
  - Empty Records
  - Attributes of Enumerated Types
  - Range Expressions

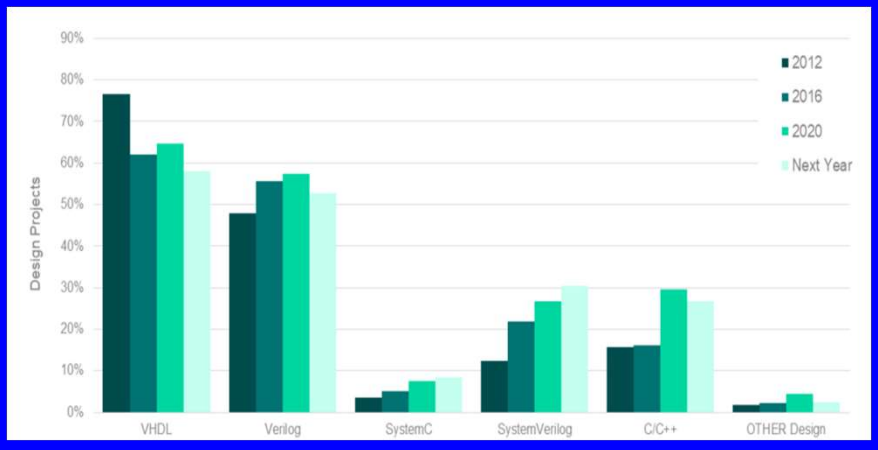
3

3

SynthWorks

VHDL is the #1 FPGA Design (RTL) Language

- 2020 Wilson Research Functional Verification Survey



- © Siemens 2020 https://blogs.sw.siemens.com/verificationhorizons/2020/12/16/part-6-the-2020-wilson-research-group-functional-verification-study/

4

4

SynthWorks

What to Change & What not to Change

- Rules
  - Don't break old code
  - … at least not intentionally and not without notification
  - Mistakes happen – less will happen if more people participate
- Some RTL things we talked about in Part 1
  - Interfaces
  - Conditional Analysis
  - Using Date and Time to create a hardware build time register (slide 17)
- Good things that will not change
  - Uniformity and Consistency
  - Strong Typing
  - Rich Math Capability
  - Conciseness (2008)

5

5

SynthWorks

Strong Typing (1987)

Strong Typing = Strong error checking built into the compiler

- 60-70% of the time a strong typing violation is a bug
- Rules are easy.
  - However, some are package specific (Numeric_Std vs Fixed_Pkg)

Operation	Size of Y = Size of Expression
`Y <= "10101010" ;`	number of digits in literal
`Y <= X"AA" ;`	4 * (number of digits)
`Y <= A ;`	A'Length = Length of array A
`Y <= A and B ;`	A'Length = B'Length
`W <= A > B ;`	Boolean
`Y <= A + B ;`	Maximum (A'Length, B'Length)
`Y <= A + 10 ;`	A'Length
`V <= A * B ;`	A'Length + B'Length

6

6

SynthWorks

Rich Math Capability (87, 93, 96, 97, 08)

VHDL is Superior at Math

Package	Contents
standard	math for integer and real
std_logic_1164	std_logic family + logic operators
math_real	log, trig, and random functions
math_complex	math and types for complex numbers
numeric_std	unsigned and signed
numeric_std_unsigned	unsigned math for std_ulogic_vector
fixed_generic_pkg	Generic unsigned and signed fixed point
fixed_pkg	Instance of the generic fixed package
float_generic_pkg	Generic floating point
float_pkg	Instance of the generic float package

- Made feasible by operator overloading

7

7

SynthWorks

Conciseness (2008)

- Simple sensitivity list with keyword "all"

```
Mux2_proc : process(all)  . . .
```

- Conditionals allow a bit or std_logic result

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

- Matching relational operators: ?=, ?/=, ?>, ?>=, ?<, ?<= return std_ulogic

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

- Case statements allow expressions (with globally static types)

```
case A xor B is
```

- Port maps allow expressions on inputs

```
U_DataBlk : DataBlk
  port map ( I1 => A and B, I2 => C, O1 => Y ) ;
```

8

8

Conciseness (2008)

SynthWorks

- Sized Literals

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

- Logic Reduction Operations

```
Parity_sl <= xor Data8 ;
```

- Array - Bit Operations

```
Y8 <= (A8 and Asel_sl) or (B8 and Bsel_sl) ;
```

```
Y8 <= A8 + B8 + Cin_sl ;
```

- We are not necessarily done with conciseness updates, however,

Now VHDL is more concise than Verilog / SV

It is certainly that VHDL is more self-consistent

9

9

Conditional Expressions

SynthWorks

- VHDL-2008 allows conditional assignments in a process

```
sNextState <= FLASH when (FP) else IDLE ;
. . .
vNextState := FLASH when (FP) else IDLE ;
```

- However, conditionals in declarations still required custom functions

```
constant MODEL_INSTANCE_NAME : string :=
    ifelse( gMODEL_NAME /= "", gMODEL_NAME, "X86_Lite" ) ;
```

- VHDL-2019 implements conditional expressions

```
constant MODEL_INSTANCE_NAME : string :=
    gMODEL_NAME when gMODEL_NAME /= "" else "X86_Lite" ;
```

10

10

Conditional Expressions

SynthWorks

- Basic Conditional Expressions. "else" is required in this form

```
conditional_expression ::=
    expression { when condition else expression }
```

- Initializer in constant, signal, variable, and interface object declarations

```
constant period : time := 10 ns when gSLOW else 1 ns ;
```

- Actual designator in an association list (port, parameter, or generic)

```
generic map (
   delay => 10 ns when gSLOW else 1 ns,
```

- Attribute specifications

```
attribute RAM_STYLE : string;
attribute RAM_STYLE of RAM : signal is
  "block" when gLARGE_RAM else "distributed" ;
```

11

11

Conditional Expressions

SynthWorks

With Parentheses, any expression can have a conditional

```
primary ::=
    . . .
    | ( conditional_expression )
```

- A few places where we need parentheses

```
with MuxSel select
  (A + B when Nom else C + D) when "00",
  . . .
```

  - Without parentheses "when" would belong to "with – select" = error

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

  - With parentheses = Synchronous reset flip-flop
  - Without parentheses = Asynchronous reset flip-flop

12

12

SynthWorks

Conditional Expressions

- Extended Form: Conditional or Unaffected Expression (Limited Use)

```
conditional_or_unaffected_expression ::=
    expression_or_unaffected { when condition else
        expression_or_unaffected } [ when condition ]

expression_or_unaffected ::=
  expression | unaffected
```

- Variable assignment – subsumes conditional variable assignment

```
vNextState := FLASH when (FP) else IDLE ;

vNextState := FLASH when (FP) ; -- else don't change
```

- Signal assignment with a force expression

```
Sig <= force A when (En) ;

Sig <= force unaffected when (not Enable) else A ;
```

13

13

SynthWorks

Conditional Return

- Conditional Return from a Function

```
value_return_statement ::=
  [ label : ] return conditional_or_unaffected_expression ;
```

```
return R1 when A <= 16 else R2 when A <= 32 ;
```

  - Returns a value.
  - If unaffected or no final else, execution continues after return

- Conditional Return from a Procedure

```
plain_return_statement ::=
  [ label : ] return [ when condition ] ;
```

```
return when A <= 32 ;
```

  - Returns execution control.

14

14

SynthWorks

Inferring Signal and Variable Constraints

- VHDL-87 allows ports to be unconstrained – like subprogram parameters

```
entity E is
  port(
    A : in  ufixed ;
    B : in  ufixed ;
```

- Deriving Signal Dimensions: VHDL-2008

```
constant CalcResultSize  : ufixed := A + B ;
signal Result            : CalcResultSize'subtype ;
```

- Deriving Signal Dimensions: VHDL-2019

```
signal Result : ufixed := A + B ;
```

- VHDL-202X?

```
signal Result : (A + B)'subtype ;
```

15

15

SynthWorks

All Interface Lists are Ordered

- VHDL-2008 made generic lists ordered
- VHDL-2019 makes Port Lists ordered

```
entity E is
  port(
    A : in  ufixed ;
    B : in  A'subtype ;
    Y : out ufixed(A'left + 1 downto A'right) ;
```

- VHDL-2019 makes Parameter Lists ordered

```
function Mux4 (
  Sel        : std_logic_vector (1 downto 0) ;
  A          : std_logic_vector ;
  B, C, D    : A'subtype ;
) return std_logic_vector is
```

16

16

SynthWorks

Functions Know Output Subtype

- VHDL-2019 syntax adds "return_identifier of"

```
function_specification ::=
  [ pure | impure ] function designator
  subprogram_header
  [ [ parameter ] ( formal_parameter_list ) ]
    return [ return_identifier of ] type_mark
```

- Potential update to "to_unsigned"

```
function TO_UNSIGNED (ARG : NATURAL)
  return result of UNRESOLVED_UNSIGNED is
begin
  return to_unsigned( ARG, result'length ) ;
end function TO_UNSIGNED;
```

17

17

SynthWorks

Functions Know Output Subtype

- Legal Usage

```
constant TEN  : unsigned(7 downto 0) := to_unsigned(10) ;
. . .
process
  variable   Y, A : unsigned(7 downto 0) ;
begin
  A <= to_unsigned(10) ;
  wait for 1 ns ;
  Y <= A + to_unsigned(10) ;
```

- Subtype (type + constraints) of "return_identifier" is determinable by:
  - Subtype of constant, signal, or variable declaration
  - Target of an assignment (A and Y above)
  - Formal if used as an actual
  - Actual if used as a conversion of the formal
  - Subtype of type qualifier

Works well if expression argument size = target size

18

18

SynthWorks

Functions Know Output Subtype

- Allows to_unsigned to be a conversion function

```
E_1 : E
 port map (
   A_slv                =>  to_unsigned(X_Int),
   to_unsigned(B_Int)   =>  Y_slv,
   . . .
```

  - VHDL conversion functions can only have 1 parameter

- To be legal to use to_unsigned here,
  - Conversion on input
    - Formal port A_slv must have constraints
    - IE: A_slv must be constrained
  - Conversion on output
    - Actual port Y_slv must have constraints
    - Always TRUE since Y_slv is a signal that has constraints.

19

19

SynthWorks

Functions Know Output Subtype

```
signal   A_uf  : ufixed(3 downto -3) ;
signal   Y_uf  : ufixed(4 downto -3) ;
. . .
Y_uf <= A_uf + to_ufixed(2.5) ;
```

- Size of to_ufixed matches the target, Y = (4 downto -3)
- Illegal since size of result is (5 downto -3)

```
E_1 : E
 port map (
   A_uv    =>  to_unsigned(X_Int) * B_uv,
```

- Size of to_unsigned matches size of input port A_uv.
- Illegal since size of result = A'length + B'length

```
(A_slv, B_slv) <= to_unsigned(X_Int) ;
```

- LRM explicitly makes this illegal when aggregate on LHS

20

20

Regularize Component Declarations

SynthWorks

- In component, now end by itself is allowed.

```
entity AxiStreamTransmitter is
port (
    Clk       : in  std_logic ;
    AxiStream : view AxiStreamTxView of AxiStreamRecType ;
    TransRec  : inout StreamRecType
  ) ;
end entity AxiStreamTransmitter ;
```

```
component AxiStreamTransmitter is
port (
    Clk       : in  std_logic ;
    AxiStream : view AxiStreamTxView of AxiStreamRecType ;
    TransRec  : inout StreamRecType
  ) ;
end component AxiStreamTransmitter ;
```

- Simplifies remembering syntax and automating component generation

21

21

Allow Optional Semicolon on Interface Lists

SynthWorks

- Entity (and Component) Interface Lists

```
entity AxiStreamTransmitter is
port (
    Clk       : in  std_logic ;
    AxiStream : view AxiStreamTxView of AxiStreamRecType ;
    TransRec  : inout StreamRecType ;
  ) ;
end entity AxiStreamTransmitter ;
```

```
component AxiStreamTransmitter is
port (
    Clk       : in  std_logic ;
    AxiStream : view AxiStreamTxView of AxiStreamRecType ;
    TransRec  : inout StreamRecType ;
  ) ;
end component AxiStreamTransmitter ;
```

- Brings consistency with record syntax
- Necessary if conditional analysis modifies the number of ports

22

22

Allow Optional Semicolon on Interface Lists

SynthWorks

- Subprogram Interface Lists

```
function Mux4 (
  Sel         : std_logic_vector (1 downto 0) ;
  A           : std_logic_vector ;
  B, C, D     : A'Subtype  ;
) return A'Subtype is
```

```
procedure AffirmIfEqual (
  AlertLogID : AlertLogIDType ;
  Received, Expected : std_logic_vector ;
  Message : string := "" ;
  Enable : boolean := FALSE  ;
) is
```

23

23

Empty Records

SynthWorks

- 2019 makes element declarations optional

```
record_type_definition ::=
  record
    { element_declaration }
  end record [ record_type_simple_name ]

element_declaration ::=
    identifier_list : subtype_indication ;
```

- After conditionals, the record may end up empty

```
type AbstractRecType is record
`if (TOOL_VENDOR = "Xilinx") then
   ID : integer ;
`elsif (TOOL_VENDOR = "Intel") then
   ID : std_logic_vector(7 downto 0) ;
`end if
end record AbstractRecType ;
```

24

24

SynthWorks

Attributes for Enumerated Types

- Use with any prefix P that is appropriate for either
  - An object with a scalar type or subtype T, or an alias thereof, or
  - Any scalar type or subtype T.

```
P'range              Range of the type T
P'reverse_range      Opposite of the range of type T
P'length             Length of type T  (ie: number of values)
```

- Extended to allow objects (subbullet 1 above)

```
P'base               Base type of T.  Only used with
                     other attributes
P'left               Leftmost value of type T
P'right              Rightmost value of type T
P'high               Upper (largest) bound of type T
P'low                Lower (smallest) bound of type T
P'ascending          TRUE if range is ascending (aka "to")
```

25

25

SynthWorks

Range Expressions

- With Ranges we can

```
constant Aval : unsigned(7 downto 0) := X"4A" ;
signal   A    : unsigned(Aval'range) ;
```

- However, a range can not be passed.
- As a result, conversions in IEEE Fixed_Generic_Pkg either
  - Pass the indices or
  - Pass an object of the desired size
  - Both to some degree are ugly

```
signal A4    : ufixed (3 downto -3) ;

A4 <= to_ufixed( 6.5, 3, -3 ) ;      -- pass indices

A4 <= to_ufixed( 6.5, A4 ) ;         -- pass an object
```

26

26

SynthWorks

Range Expressions

- VHDL-2019 adds range expressions

```
range ::=
  . . .
  | range_expression
```

  - Type = the implicitly defined range record for the indices
  - Can be used anywhere a range is used

- A range record is implicitly defined for all scalar types

```
type RANGE_DIRECTION is (
  ASCENDING, -- The range is ascending.
  DESCENDING -- the range is descending.
);

type <unnamed_range_record> is record
  Left      : <scalar_type>;
  Right     : <scalar_type>;
  Direction : RANGE_DIRECTION;
end record;
```

27

27

SynthWorks

Range Expressions

- For type integer, a range record type is implicitly defined:

```
type <unnamed_INTEGER_range_record> is record
  Left      : INTEGER;
  Right     : INTEGER;
  Direction : RANGE_DIRECTION;
end record;
```

- We access the implicitly defined range record type for integer by:

```
subtype INTEGER_range_record is integer'range'record ;
```

- For VHDL-2019, the following are equivalent

```
constant INTEGER_RANGE_1 : INTEGER_range_record := (
    Left       =>  -2**63,
    Right      =>  2**63 - 1,
    Direction  =>  ASCENDING ) ;

constant INTEGER_RANGE_2 : INTEGER_range_record :=
       integer'range'value ;
```

28

28

Range Expressions

SynthWorks

- Back to ufixed:

```
signal A4    : ufixed (3 downto -3) ;
```

- With range expressions we can also write:

```
constant B4_RANGE : INTEGER_range_record := (
    Left       =>  3,
    Right      =>  -3,
    Direction  =>  DESCENDING ) ;

signal B4    : ufixed (B4_RANGE) ;
```

- To_ufixed can be overloaded for INTEGER_range_record

```
B4 <= to_ufixed(6.5, B4_RANGE) ;
```

- If we defined "+" for INTEGER_range_record, then

```
signal A4, B4   : ufixed (B4_RANGE) ;
signal Y5       : ufixed (B4_RANGE + B4_RANGE) ;
```

29

29

Trying Out VHDL-2019

SynthWorks

- Clone the VHDL-2019 presentation repositories:

```
git clone --recursive https://gitlab.com/synthworks/VHDL_2019
```

- In RivieraPRO, cd to sim directory and initialize scripting environment

```
cd VHDL_2019/_sim
source ../OsvvmLibraries/Scripts/StartUp.tcl
```

  - See Documentation/Scripts_user_guide.pdf for Aldec ActiveHDL

- Build the OSVVM Libraries

```
build ../OsvvmLibraries
```

- Run a test

```
build ../ConditionalExpressions
build ../InterfaceLists
build ../ConversionPkg
build ../EmptyRecord
build ../RangeExpressions
```

30

30

IEEE VHDL Working Group

SynthWorks

- Participation
  - Individual, volunteer participation based.
  - No fees or membership required – except for officers
  - Meetings are on-line, on GitLab issues, on TWIKI, and email
- We seek volunteers experienced in one or more:
  - VHDL design or verification
  - Language design (VHDL, Ada)
  - Programming Language Interfaces
  - Digital design experience (DSP, Floating-Point, …)
  - Latex
  - Commercial, University, or retired
- We seek input for 202X
  - Please share with us 10 capabilities you would like to see
  - https://gitlab.com/IEEE-P1076/VHDL-Issues/-/issues
    - Please search the list before adding new items
    - If your idea is already there, add a comment "me too"

31

31

Summary

- VHDL-2019 was
  - Requested by users
  - Ranked by users
  - Scrutinized by users
  - Written by Users
  - Balloted by the VHDL community
- It should be clear that the VHDL user community wants these features
  - I am happy to see that Aldec is well into their implementation
  - Hopefully, others will be soon
- Be sure to talk to your simulation vendors about VHDL-2019
- VHDL-202X work is already in process. Join us.
  - See: http://www.eda-twiki.org/cgi-bin/view.cgi/P1076/WebHome
  - See: https://ieee-p1076.gitlab.io/About/index.html

32

32

SynthWorks VHDL Classes

Comprehensive VHDL Introduction 4 Days / 9 on-line sessions
https://synthworks.com/comprehensive_vhdl_introduction.htm
Learn VHDL for FPGA and ASIC design and verification. Class covers syntax, RTL coding, and testbenches. Class comes with your choice of an Altera or Xilinx FPGA board to make sure you understand the whole process from simulation to chip

VHDL Coding for Synthesis 4 Days / 8 on-line sessions
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.

Advanced VHDL Testbenches and Verification 5 days / 10 on-line sessions
http://www.synthworks.com/vhdl_testbench_verification.htm
Learn the latest VHDL verification techniques including transaction based modeling, self-checking, scoreboards, memory modeling, functional coverage, directed, algorithmic, constrained random, and intelligent testbench test generation. Create a VHDL testbench environment that is competitive with other verification languages, such as SystemVerilog or 'e'. Our techniques work on VHDL simulators without additional licenses and are accessible to RTL engineers.

33