

VHDL-200X-FT

Meeting Dec 4, 2003

Jim Lewis

Director of Training, SynthWorks
VHDL-200X-FT, Co-Team Leader

SynthWorks

IEEE-SA Standards Board Bylaws on Patents in Standards

6. Patents

IEEE standards may include the known use of patent(s), including patent applications, provided the IEEE receives assurance from the patent holder or applicant with respect to patents essential for compliance with both mandatory and optional portions of the standard. This assurance shall be provided without coercion and prior to approval of the standard (or reaffirmation when a patent becomes known after initial approval of the standard). This assurance shall be a letter that is in the form of either

a) A general disclaimer to the effect that the patentee will not enforce any of its present or future patent(s) whose use would be required to implement the proposed IEEE standard against any person or entity using the patent(s) to comply with the standard or

b) A statement that a license will be made available without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination

This assurance shall apply, at a minimum, from the date of the standard's approval to the date of the standard's withdrawal and is irrevocable during that period.

- Don't discuss licensing terms or conditions
- Don't discuss product pricing, territorial restrictions or market share
- Don't discuss ongoing litigation or threatened litigation
- Don't be silent if inappropriate topics are discussed... do formally object.

**If you have questions,
contact the IEEE Patent Committee Administrator
at patcom@ieee.org**

- Unconstrained arrays of unconstrained arrays
- Records of unconstrained arrays
- Interfaces w/ Records
- Context Clause Design Unit
- FT-05 TO_String and &
- FT-08 Hwrite, Dwrite, Owrite, Bwrite, Hread, Dread, Oread, Bread, Swrite, SRead
- FT-07 Signal Spy
- FT-10 Conditional and Selected Signal Assignment
- Ternary Operator with when else
- Add if else (deprecate when else)
- FT-11 Signal expressions in port maps +
 - what is a conversion function vs. what is an expression in the ports
- FT-12 Read Out ports
- Boolean Equivalence

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) ;
    . . .
) ;
```

5

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)) ;
```

6

Records & Interfaces

- Problem: Record ports have no way to identify direction of a field



```
type IfRecType is record
  Sig1      : std_logic ;
  Sig2      : std_logic_vector (7 downto 0);
end record ;
```

- With a method to identify direction of fields, record ports would be more useful.
 - Field that is input to one entity would be output of another.
- Current work arounds:
 - Use std_logic_vector and initialize to 'Z' (not great)
 - Use two records (yuck)

Records & Interfaces

- Current KLUDGE, requires std_logic resolution:

```
constant InitModel1_IfRecType : IfRecType := (
  Sig1      => 'Z' ;
  Sig2      => (others => '0') ;
end record ;

constant InitModel2_IfRecType : IfRecType := (
  Sig1      => '0' ;
  Sig2      => (others => 'Z') ;
end record ;
```

Records & Interfaces

- Proposal 1: add initialization

```
constant InitModel1_IfRecType : IfRecType := (  
  Sig1    => off ;    -- ?null?  
  Sig2    => (others => '0') ;  
end record ;  
  
constant InitModel2_IfRecType : IfRecType := (  
  Sig1    => '0' ;  
  Sig2    => off ;    -- ?null?  
end record ;
```

Records & Interfaces

- Proposal 2: New subtype or interface (similar to SV's modports)

```
subtype Model1_IfRecType of IfRecType is record  
  Sig1    : in ;  
  Sig2    : out ;  
  Sig3    : inout ;  
  Sig4    : in ;  
end record ;  
  
subtype Model2_IfRecType of IfRecType is record  
  Sig1    : out ;  
  Sig2    : in ;  
  Sig3    : inout ;  
  Sig4    : in ;  
end record ;
```

Records & Interfaces (after meeting)

- Proposal 3: add mode

```
Mode IfRecType_Mode_Model1 : IfRecType := (  
  Sig1    => in,  
  Sig2    => out  
end record ;
```

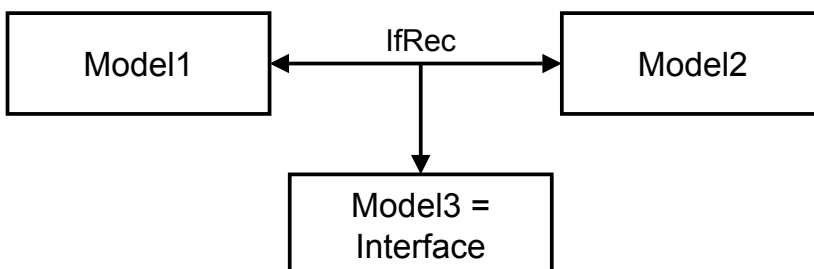
```
Mode IfRecType_Mode_Model2 : IfRecType := (  
  Sig1    => out,  
  Sig2    => in  
end record ;
```

```
Entity Model1 is  
port (  
  IfRec1 : IfRecType_Model_Model1 : IfRecType ;  
  . . .
```

VHDL "Interfaces"

SynthWorks

- VHDL's answer to System Verilog Interfaces



- Interface is nothing more than an entity that has the record as an input only
 - Model does protocol checks, timing checks, ...

Context Clause Design Unit

- Problem: Currently users have to specify a large collection of packages before an entity and there is no way to abstract this.
- Proposal:
 - Create a design unit which is the context clause
 - Create a default context clause and allow users to override it.

Current default context clause:

```
Context default_context is
  library std ;
    use std.standard.all ;
  library work ;
end ;
```

Proposed default context clause:

```
Context default_context is
  library std ;
    use std.standard.all ;
  library work ;
  library ieee ;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all ;
  use std.textio.all ;
end;
```

- Intent is that the standard default context clause references all standard packages - allows us in some ways to drive the methodology

Context Clause Design Unit

- User can overload default context clause and compile it into work.
- User can create user defined context clause and reference it in the design

```
Context project1_Ctx is
  library Lib_p1 ;
    use Lib_P1.P1Pkg.all ;
    use Lib_P1.P1_Defs.all ;
    ...
end ;
```

- The context clause, default_context is referenced by default
- User context clauses can be referenced with a use clause:

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

- ?Simplify by allowing the named library to be implicitly referenced by the use clause?

```
Library Lib_P1 context project_ctx ;
```

- Report requires string values. To_string would make report more useful:

```
assert (ExpectedVal = ReadVal)
  report "Expected Value /= Actual Value. Expected = " &
    to_string(Expected) & "      Read = " & to_string(ReadVal))
  severity error ;
```

- Furthermore, to_string permits a usage of vhd-93 write:

```
-- write(<file_handle>, <string>) ;
write(Output, "%%ERROR data value miscompare." &
  CRLF & "  Actual value = " & to_hstring(Data) &
  CRLF & "  Expected value = " & to_hstring(ExpData) &
  CRLF & "  at time: " & to_string(now, right, 12) ) ;
```

- Support Radix as a parameter?

```
function to_string (
  VALUE          : in integer;
  radix          : in radix_type := DEC ;
  JUSTIFIED      : in SIDE := RIGHT;
  FIELD         : in WIDTH := 0
) return string ;
```

- Support Radix by having multiple procedures (like std_logic_textio)?

```
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 ;
```

- Use shorter names?

```
Consider alt shorter names:  str, hstr
```

Overloading "&" and To String? SynthWorks

```
function "&" (  
    l      : in string;  
    r      : in unsigned  
    ) return string ;  
function "&" (  
    l : in unsigned ; r : in string  
    ) return string ;  
function "&" (  
    l : in unsigned; r : in unsigned  
    ) return string ;
```

```
constant A : unsigned := "1010" ;  
signal Y : unsigned(7 downto 0) := ("0000" & A) + 1 ;  
. . .  
Y <= (A & A) + A ;  
Write(Output, "A twice: " & A & A & " Y = " & Y) ;
```

17

Hwrite, Dwrite, Owrite, Bwrite SynthWorks

```
procedure hwrite (  
    Buf          : inout Line ;  
    VALUE        : in integer;  
    JUSTIFIED    : in SIDE := RIGHT;  
    FIELD        : in WIDTH := 0  
    ) ;  
procedure dwrite (  
    Buf          : inout Line ;  
    VALUE        : in integer;  
    JUSTIFIED    : in SIDE := RIGHT;  
    FIELD        : in WIDTH := 0  
    ) ;  
procedure owrite ( . . . ) ;  
procedure bwrite ( . . . ) ;
```

18

Hread, Dread, Oread, Bread

SynthWorks

```
function hread (
    Buf           : inout Line ;
    VALUE         : out integer;
    Good          : out boolean
) ;

function dread (
    Buf           : inout Line ;
    VALUE         : out integer;
    Good          : out boolean
) ;

function oread ( . . . ) ;
function bread ( . . . ) ;
```

19

Signal Spy (FT-07)

SynthWorks

- Signal Spy donation from MTI:

```
init_signal_driver(src, dest, delay, delay_type, verbose) ;
init_signal_spy(src_object, dest_object, verbose) ;
signal_force( dest, value, rel_time, force_type,
    cancel_period, verbose ) ;
signal_release( dest_object, verbose ) ;
```

- Signal names and value specified as a string
- Some limitations when reading Verilog values in VHDL.
- Functionality is appropriate, perhaps new names?
 - Probe, Drive, Force, release?
- Extensions?

```
Get_Value( dest, verbose) ; -- function that returns value
```

- Problematic wrt type limitation
- Type limitations?

20

Signal Spy alternate with alias

- Signal Spy alternative with alias:

```
Alias A : std_logic is ":tb: uut:b:d:a_sig" ;  
  
Alias B : std_logic is ":tb: uut:b:" & "d:C_sig" ;  
  
Alias C : std_logic is MY_GENERIC & "d:C_sig" ;
```

- Signal Spy alternative with Signal Declaration

```
signal A : std_logic probe(. . .) ;  
signal B : integer drive(. . .) := 5 ;
```

Sequential usage of Conditional and Selected Signal Assignment

- Common branching in a statemachine

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

- Simplifying this code by using conditional signal assignment

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

Make conditional signal assignment a ternary operator

- Register with conditional signal assignment

```
AReg <= A when rising_edge(Clk) ;
```

- Ternary Operation, Synchronous Reset

```
AReg <=
    ('0' when nReset = '1' else A) when rising_edge(Clk) ;
```

- Initialization:

```
Signal A : integer := 7 when GEN_VAL = 1 else 15 ;
```

- Asynchronous Reset (not ternary):

```
AReg <= '0' when nReset = '1' else
    A when rising_edge(Clk) ;
```

23

Add "if else" for conditional signal (deprecate when else?)

- Register with conditional signal assignment

```
AReg <= A if rising_edge(Clk) ;
```

```
-- PrioritySel Logic
Y <=
    A if ASel = '1' else
    B if BSel = '1' else
    C if CSel = '1' else
    D ;
```

- Why?
 - Many confuse "when else" with selected signal assignment (see RMM RTL recommendations for multiplexers)
- Deprecate when else?

24

Signal Expressions in Port Maps SynthWorks

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

- For PSL and OVL to remove need for extra signal assignment
- Semantics of expressions in port map:
 - convert to concurrent signal assignment
 - delta cycle -- prefer no delta cycle
 - Type Conversion/Conversion Function = no delta cycle
 - If expression = signal, no delta cycle
 - Paradox: "not"(A) no delta cycle, whereas, not A has a delta cycle
- Also would facilitate 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
  );
```

Conversion Function Definition SynthWorks

- What defines a conversion function:
 - only one actual can be a signal
 - Allow additional inputs if they are globally constant expressions
- to_unsigned(int, 5)
 - sometimes problematic
 - returns an unconstrained array
 - Size cannot be determined statically
- If formal of entity port is constrained, then ok.
 - Can constrain formal in port map:
 - f(31:0) => to_unsigned(int, 32) -- permitted
 - unconstrained_f => to_unsigned(int, 32) -- not permitted
- Restrict: Must be a single signal name (slice, name, indexed, sig attribute)
 - F => CF(A(3), B(3)) --
 - F => CF(A(3), A(4)) --
- Delta cycle if > single signal name in the function

Conversion Function Definition

SynthWorks

- LRM Issue:
 - $P \Rightarrow CF(A)$ -- permitted
 - $P \Rightarrow CF(F_PARAM \Rightarrow A)$ -- not permitted
- Currently subprograms not allowed to have conversion functions
- Expanding signal rules to subprogram would have to be carefully analyzed

27

Read Output Ports

SynthWorks

- Read output ports
 - Value read will be locally driven value
 - "?Erroneous if additional drivers outside local block?"
- Helpful for writing assertions

28

- Std_logic expr in if statements
 - View where hide implicitly visible operators for a type (=, /=)
 - Provide own version of operators (= that sl in and sl out)
 - "condition"(Arg) return boolean
 - called whenever a condition can be used (wait, if, while, ...)
 - handles all simple cases
- Issues:
 - S1 or S2 - ok
 - S1 or not S2 -- X mapping problematic (fuzzy equals)
 - (V = "001") and S1
- Ideas:
 - Create SL with new so it does not inherit "=" operations -- not accepted
 - eq vs == to return std_logic
- Proposal
 - "??"(Arg) return boolean ;
 - implied in grammar whenever "condition" is used
 - in std: bit, boolean, ?integer, ?real, ?bit_vector
 - in 1164: define ?? For Std_ulogic, ?sulv, ?slv
 - Fuzzy equal? Not implicitly defined (that returns false if X) Accounting for X's ?=, ?/=?, ?<, ?>, ?<=, ?>= handle '-' correctly
 - in 1164 and for bit: for all logic operators overload: sul, sul, bool / sul, bool, bool / bool, sul, bool
 - case? Uses: "?=" requires mutually exclusivity