

## **1. Approach**

### **1.1 Central Bug/Enhancement for all of VHDL**

To virtually integrate all dot standards and associated standards, I would like to have one central place for submitting proposals and one methodology for formatting proposals. With just the current reflectors it is difficult to track proposals, proposals get submitted to the wrong reflector, and in the wrong format.

### **1.2 Long Range Vision with Short Term Milestones**

Depending on how much revision to the language is desired, we may wish to have a long range vision with short term milestones. The short term milestones would be implemented by vendors, but not necessarily a full IEEE standard. Final deliverable will be the next generation IEEE VHDL standard.

### **1.3 User + Vendor Buy In**

Need to have vendor and user buy in for new features. Need to market new features to users (similar to what the Verilog groups are doing).

I volunteer myself to do presentations to market / evangelize new features.

### **1.4 Goals to modifications**

Faster sims  
Increased designer productivity  
Verification productivity  
Assertions  
Additional Packages

## **2. TextIO + Printing**

Printing currently takes 6-10 lines where one two line printf statement would get the job done.

### **2.1 Printf and Fprintf**

VHDL Code:  
Write (WriteBuf, string'("Current Value of A = "));  
Write (WriteBuf, A\_slv) ;  
Write (WriteBuf, string'("Current time = "));  
Write (WriteBuf, now, 12, Right, ns) ;  
Writeline(Output, WriteBuf) ;

-- Equivalent Printf, fprintf Examples:  
Printf("Current Value of A = %b Current time = %12ns \n", A\_slv, now) ;

```
fprintf(Output, "Current Value of A = %b   Current time = %12ns \n", A_slv, now);
```

## 2.2 Swrite, Hwrite, Owrite, Hread, Oread

Formatted write and read. Swrite (String write) to avoid string':

```
Swrite(WriteBuf, "Current Value of A = ") ;
```

Equivalent write:

```
Write (WriteBuf, string'("Current Value of A = "));
```

## 2.3 TO\_String, Str, Image, ...

Expose string conversion procedures from textio. Facilitates using VHDL-93 built in write. Extend write/VHDL such that LF is always a newline + carriage return when used with write (all versions).

```
Write(Output, "Current Value of A = " & to_string(A_slv) &  
        "current time = " & to_string(now, 12, right, ns) & LF) ;
```

Octal, decimal, and hex output would be desirable, so to\_hstring, to\_ostring, and to\_dstring would be desirable.

## 2.4 Add TEE to std.textio.all

```
procedure tee (  
    file OutFile : text ;  
    variable WriteBuf : inout line  
) is  
    variable CopiedBuf : line ;  
begin  
    CopiedBuf := new string'(WriteBuf.all) ;  
    writeline(OutFile, WriteBuf) ;  
    writeline(Output, CopiedBuf) ;  
end;
```

### **3. Language Interfaces**

#### **3.1 Read Verilog Gate-level Netlists**

Verilog netlists simulate fast. Would permit silicon vendors to support one netlist format and one sign-off library

#### **3.2 Direct C Calls**

System design organizations (defense contractors) want to be able to reuse their C code in testbenches and initial behavior modeling. Calls to C concurrently should be fairly straight forward.

#### **3.3 Direct Verilog Calls**

### **4. Predefined environment**

Predefined constants to define if tool is a simulation, cycle simulator, synthesis tool, formal verification, ...

### **5. Simulation Environment**

\$STOP, . . . from verilog

### **6. Language Tweaks / Extensions**

#### **6.1 Allow concurrent signal assignments sequential code**

Allow conditional signal assignment and selected signal assignment in sequential code. For simple statemachines:

```
LedNext <= LED_OFF when Pending = '1' else LED_IDLE;
```

Selected signal assignment for similar reasoning.

#### **6.2 1 dimensional array aggregates**

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

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

### **6.3 Graceful degradation of types**

For enumerated types such as `std_logic`, specify the primary values and how to degrade the number of values used with the type to increase the simulation speed.

### **6.4 Report Severity**

Stopping Testbenches: Done, Stop

To get a breakpoint with report, currently any message a user prints is prefixed with, "# Failure "

### **6.5 Case Statement Expressions**

Constructing a useful case statement expression is beyond the typical VHDL user.

### **6.6 Sensitivity Lists**

Include all signals on sensitivity list with short hand: \* -or- all

### **6.7 Read Output ports**

### **6.8 Assign Image values for identifier based enumerated types**

Type StateType is (Invalid, S0, S1, S2) ;

**attribute** ENUM\_ENCODING of StateType : **type is "-- 00 01 10" ;**

An attribute such as ENUM\_ENCODING does not allow the code to access values of the object. Perhaps we could build in a language feature which facilitates this?

### **6.9 Constraining size of Hex, Octal, Decimal string literals**

5X"1F" = 5 bits, all one in hex

5D"255" = 5 bits, all one in decimal

### **6.10 Sizing of Integers 32, 64, 128, ?**

Currently integers beyond the size of 32 are not portable.

### **6.11 Size Real Numbers**

To be able to synthesize real numbers, we must be able to support something other than just the standard sizes. To facilitate this, add a mechanism (attributes) to constrain the size of the exponent and mantissa of a real number. Add an additional mechanism that allows a simulator to simulate with the sized real number for accuracy of the model or to simulate with the standard sized real number (if it is larger) for simulation speed.

## **6.12 Std\_logic extensions??**

Add forcing strength values to std\_logic, or provide a simulation environment that permits this. Use for stubbing out logic during debug?

## **6.13 Make transport the default delay model**

It is not clear this can be done, but many users would benefit from this. Inertial delay is useful for gate level modeling. Testbench modeling would be facilitated by this.

## **6.14 Reduction Operators**

## **6.15 Bit Vector Array w/ Scalar logic operators**

Similar to Std\_logic\_1164 extensions

## **6.16 Record Templates with Unconstrained Arrays:**

Declaration:

Type Floating is Record template

  Mantissa : signed ;

  Exponent : signed ;

End record ;

Usage (conceptual):

  Signal Float32 : Floating (Mantissa(23 downto 0), Exponent(7 downto 0)) ;

## **6.17 Rising\_edge for bit**

## **6.18 Unsigned numeric understanding of bit\_vector**

## **6.19 Bidirectional Connections**

A <=> B ; no delta cycles, immediate transmission

## **6.20 Physical Literals and Units**

5 ns = 5ns

## **6.21 Permit ";" to follow last Interface\_Element**

## **6.22 Allow min syntax**

VHDL-93 made much syntax similar if the full syntax is specified.

For example, entity and configuration declaration. However, it would be nice if they were identical also with minimum syntax.

## 6.23 Allow Subprogram Implementations in package declarations

A small win. Average user will be a user of a package and not a developer.

## 6.24 Else Clause for IF-Generate

## 6.25 EndIf (flows with ElseIf)

## 6.26 OrIf, OrEls, ErrEls

From: Weng Tianxiang

For expressing mutual exclusive conditions in a statemachine:

```
If (quarter = '1') then
  nextState <= GOT_QUARTER ;
Orif (Dime = '1') then
  nextState <= GOT_DIME ;
Orif (Nickle = '1') then
  nextState <= GOT_NICKLE ;
OrErr
  Report "Decoder failure. More than one input set at a time" severity failure ;
Else
  nextState <= IDLE ;
End if ;
```

Alternatively, do not support erreles and optionally cover it with an assert statement:

```
Assert MutuallyExclusive(Quarter & Dime & Nickle)
  Report "Quarter, Dime, and Nickle are not mutually exclusive"
  Severity failure ;
```

## 7. Additional Packages

### 7.1 Comparisons that return std\_logic

Replace: `Y <= '1'` when `std_match(A, B)` else `'0'` ;

With: `Y <= EQ (A, B) ;` -- understand '-'. `rtn 'X'` when either A or B is 'X'  
`Z <= GT (X, Y) ;`

Table Functions, ... IBM has some of these. Talk to Lance Thompson at IBM Rochester.

### 7.2 Extended Hardware Functions

Mux2, Mux4, Decode, ...

AddCi, AddCiCo, AddSub, ... Similar to current DW functions

### 7.3 Testbench Utilities

There seem to be many requests for fork and join. To me fork and join are for starting and stopping concurrent actions from a sequential language. What I think we really need (and I actually use) are procedures that allow synchronizing multiple processes and handshaking between models:

```
-----  
procedure SyncTo (  
-- Synchronize two processes until both have called SyncTo  
-----  
    signal SyncIn      : in  std_logic ;  
    signal SyncOut     : out std_logic  
) is  
begin  
    -- Activate Rdy  
    SyncOut <= '1' ;  
  
    -- Make sure our Rdy is seen  
    wait for 0 ns ;  
  
    -- Wait until other process' Rdy is at level 1  
    if SyncIn /= '1' then  
        wait until SyncIn = '1' ;  
    end if ;  
  
    -- Deactivate Rdy  
    SyncOut <= '0' ;  
end ; -- procedure SyncTo
```

#### Usage:

```
Proc1 : process  
Begin  
    . . .  
    -- Proc1 & Proc2 stop until both reach this point  
    SyncTo( SyncIn => Proc2Rdy, SyncOut => Proc1Rdy) ;  
    . . .  
end process ;  
  
Proc2 : process  
Begin  
    . . .  
    -- Proc1 & Proc2 stop until both reach this point  
    SyncTo( SyncIn => Proc1Rdy, SyncOut => Proc2Rdy) ;  
    . . .  
end process ;
```

There are a number of additional procedures that also belong here:

```
Expect("Error in CpuRead", ExpectedValue, ReadValue, ErrorCount);  
Expect("Error in CpuRead", ExpectedValue, ReadValue, FoundError);
```

RequestAction  
WaitForRequest

## **8. Verification Support**

### **8.1 Issue: Subprogram IO**

Often with procedure that are for testbenches, much of the IO is a port of the design under test. The ports are required for the procedure to drive, but do not provide any great value to the test writer. As a result, a testbench procedure call for a CpuWrite of X86-like CPU looks like the following:

```
CpuWrite (Clk, nAds, Addr, Data, Read, nRdy, X"A0000", X"5555");
```

What the testbench writer would like to specify is:

```
CpuWrite (X"A0000", X"5555");
```

With side-effects, this is feasible, but the subprogram must be coded in the testbench architecture itself. For most system level tests, there is more than one testbench architecture, hence, the subprogram definition must be repeated.

To overcome this, usage of records is possible:

```
CpuWrite (CpuRec, X"A0000", X"5555");
```

The issue with records is that they are InOut of more than one entity. As a result, the fields of the record must be resolved. My simple minded approach is to use the std\_logic family and drive 'Z' on unused fields.

The following proposals address these issues.

#### **8.1.1 Record IO**

Record IO typically has multiple drivers which must be initialized and resolved.

Current methodology, only use std\_logic and initialize to 'Z':

```
Constant INIT_CPU_MODEL : CpuRecType := (  
  CmdRdy    => 'Z',  
  CmdAck    => '0',  
  Address   => (others => 'Z'),  
  Data      => (others => 'Z')  
);  
  
entity CpuModel is  
port (  
  CpuRec : InOut CpuRecType : INIT_CPU_MODEL ;  
  . . .
```

Always only one model is driving at a time. It is feasible to set up fields so that only one model ever drives the field. As a result, use of null or undriven here would remove the necessity of a resolved type:

```
Constant INIT_CPU_MODEL : CpuRecType := (  
  CmdRdy    => UNDRIVEN,  
  CmdAck    => '0',  
  Address   => UNDRIVEN,  
  Data      => UNDRIVEN  
) ;  
  
entity CpuModel is  
port (  
  CpuRec : InOut CpuRecType : INIT_CPU_MODEL ;  
  . . .
```

Again the intent of UNDRIVEN is so it is not necessary to use a resolved type if only one object is driving. Here it only shows Address and Data in the record with which is easy to use a std\_logic based type. However, it is not uncommon to pass objects of type time across the interface. In addition, it would be more natural to deal with error counters as an integer/natural rather than type unsigned.

### **8.1.2 Subprogram: Macro**

A macro is a template and can be declared in a package. A reference to a macro causes the call to be replaced with the code in the macro (the macro expands into the code rather than being called). Macro formal parameters do not have a type, instead, they take on the type of the object with which they are called.

Assignments in the macro can target the formal parameters or objects in the design in which the macro expands. Naming would have to be consistent.

The intent of a macro is to allow the following to be put into a package without having to use a record to simplify the subprogram interface:

```
CpuWrite (X"A0000", X"5555");
```

The types in a macro do not get checked when the package is compiled, they get checked when the macro is expanded in a design.

## **8.2 Hierarchical References**

In a testbench reference a element in another part of the design.

## **9. Synthesis Support**

### **9.1 Specify Attributes in code space:**

Specify synthesis code intent with the code

## **10. Assertions??**

### **10.1 OVL**

Current approach would be easier if expressions could be mapped to entity ports. Is it possible to somehow "qualify" an expression to make it work as a signal input.

Y => signal'(A and B),

### **10.2 OVL: Subprograms vs. Entities**

Currently OVL is using entities. Either VHDL needs to be fixed to allow expressions to be put in the actual of a port map, or convince OVL to use subprograms.

### **10.3 Subprogram for assertions**

MutuallyExclusive(Quarter, Dime, Nickle)