

Supporting Assertion-Based Verification in VHDL

An Assertions Functional Team Working Meeting

Stephen Bailey

Technical Marketing

Chair, IEEE 1076

Model Technology

A MENTOR GRAPHICS COMPANY

Instructions for the WG Chair



(Not necessary to be shown)

- At Each Meeting, the Working Group Chair shall:
 - Show slides #1 and #2 of this presentation
 - Advise the WG membership that:
 - The IEEE’s Patent Policy is consistent with the ANSI patent policy and is described in Clause 6 of the *IEEE SA Standards Board Bylaws*;
 - Early disclosure of patents which may be essential for the use of standards under development is encouraged;
 - Disclosures made of such patents may not be exhaustive of all patents that may be essential for the use of standards under development, and that neither the IEEE, the WG nor the WG Chairman ensure the accuracy or completeness of any disclosure or whether any disclosure is of a patent that in fact may be essential for the use of standards under development.
- Instruct the WG Secretary to record in the minutes of the relevant WG meeting:
 - that the foregoing advice was provided and the two slides were shown;
 - that an opportunity was provided for WG members to identify or disclose patents that the WG member believes may be essential for the use of that standard;
 - any responses that were given, specifically the patents and patent applications that were identified (if any) and by whom.



IEEE-SA Standards Board Bylaws on Patents in Standards

Slide #1

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.

Approved by IEEE-SA Standards Board – December 2002

Inappropriate Topics for IEEE WG Meetings

Slide #2



- 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

Approved by IEEE-SA Standards Board – December 2002

4

ModelSim at DAC 2003-Anaheim



Fast-Track



Current Issues

- **Explicit/implicit operators: Proposal published**
- **Unary reduction operators: Proposal written; TBP**
 - Ensure that language does not require an error for null array case
 - Null array result will be consistent with how they will be defined for `std_logic`
- **Array/scalar logic operators:**
 - Will do as predefined operators
 - Of course the bit type must be the element type of the vector type
 - Behavior is as though the bit operand is replicated to create a vector of the same length as the vector operand and the logical operation applied to the two vectors
 - Null arrays as actual: returns null array



Current Issues

- **Min/Max operators:** David Bishop in progress
- **To_string:** Jim Lewis – in progress. Jim thinks he can reuse FMF code.
- **Signal Spy:** Mentor/MTI donation in legal
- **Formatted (and radix) IO:** Reuse work from other languages (Perl)



New Issues

- **Signal expressions in port maps**
 - Implies a delta cycle. Define in terms of equivalent concurrent signal assignment
- **Reading out mode ports**
 - Driving value of the out mode port (after resolution of local drivers and contributing values)
- **Integer (and other) types larger than 32-bits**
- **Auto-visibility of `std_logic_1164`, `textio` package**
- **Dynamic arrays**
- **Eliminate need for explicit type conversions between closely-related types**



New Issues

■ Process(<keyword>) sensitivity

– Three kinds of process sensitivity recognized:

- Comb: Asserts sensitivity on all signals and a combinatorial process (no latches or registers)
- Latch: Asserts sensitivity on all signals and a latch process
- Register: Asserts sensitivity on a clock signal edge and any asynchronous control signals
 - John will investigate 1076.6 discussion of register inference to see if we can reuse it to specify how the sensitivity list is inferred for register processes.
- Tools then infer the appropriate sensitivity list from the keyword

■ Expressions in sensitivity lists

- Allow @(<expression>) for an concurrent, behavioral statement – processes, signal assignments, assertions and procedure calls (presence of @ clause overrides implied sensitivity of concurrent statement):

```
process [( <sensitivity_list> | @<expression> )] [is]
q <= d [@<expression>] ;
assert <expr> ... [@<expression>] ;
foo(a, b, ...) [@<expression>] ;
```

- If either sensitivity list or @clause is present, no wait statement allowed



New Issues

- **Selected and conditional signal assignment constructs for sequential statement context**
 - Any gotchas? Should be straight-forward
 - Have parser guru ensure no parsing problems. If no problems use existing syntax
 - Of course, new @clause would not apply
- **Define rising and falling edge operators or attributes**
 - Do as unary operator (1st operator to require a signal parameter)
 - Align definition to Verilog posedge/negedge semantics? Desired, but 1164 issue.
 - Use rising/falling_edge names to rose/fell? posedge/negedge? (rose and fell is preferred as it is coming in for PSL anyway; concerns about new keywords)



New Issues

- **Implicit (boolean) type conversion (defining true and false values)**
 - To keep things simple, implicit boolean type conversion is only applied
 - Wherever the language allows a *condition* and a conversion from the expression result type to boolean is defined.
 - Can specify mapping via an attribute specification or a function.
 - As a function and attribute
 - Attribute `implicit_tcf : boolean;`
`function to_bool(P : std_ulogic) return boolean is ... end;`
`attribute implicit_tcf of to_bool is TRUE;`
 - This looks more general and is more desirable.
 - This doesn't address the case:
 - If (`addr = "1010"`) and `A = '1'` and `B = '1'` then (would like result of `addr = "1010"` to be promoted to `std_logic` so remaining expression can be evaluated
 - General problem is implicit type conversion from any type to another in any context without making existing overload resolution significantly more complex
 - Cannot result in existing expressions evaluating to a different result
 - Equivalent expressions in new short-hand vs. what would have been required without this change must return same result.
 - Type promotion (from lesser to more general representative type) is necessary for the more general case
 - Ask Chuck Swart to chew on this.



New Issues

- Terniary operator (based on PSL boolean implication)?
 - `<bool-cond> -> <true_exp> : <false_exp> ;`
 - NOTE: -> is being added with PSL addition and is equivalent to C's ?
 - `<true_exp> when <bool-cond> else <false_exp> ;`
 - Have a vote on which of these alternative solutions is desired by VHDL users.
- Add increment and decrement operators?
 - Not now. Issues with expressions as statements, signal object inc/dec etc.
- Subprogram calls: allow `,,` in parameter list to skip over parameters with default values
 - Can use `, open,` today.



New Issues

■ Definition of case comparison operator

- Define keyword case as the operator name. Define case for predefined types.
- Then allow overloading.
- Relax some restrictions (such as choice can only appear once)
- Full case: either always require others when case is overloaded or only guarantee full case if implicitly declared case is used
- Operator definition:
 - 1st operand is the case expression
 - 2nd operand is either a choice or an array of choices. In the 1st case, the return value is boolean. In the 2nd case, the return value is an index into the array of choices for the 1st choice that matches. 2nd option can be very slow for user-defined case overloaded functions.
 - 2nd choice doesn't scale well for vector expressions (arrays of unconstrained arrays means the case operator must be overloaded for every possible length of array.)
 - Use 1st choice (case expression, choice expression)
- Allow choices to be globally static

■ Eliminate need to type qualifiers in case statement expression

- Should fall out from case changes (eliminating locally static requirement)
- Except when concatenation can create a vector interpreted of several types