Subject: VOTE - Two Deprecation Proposals - Votes due Wed., April 17

E-MAIL VOTES - DUE BY END OF DAY ON WEDNESDAY, APRIL 17TH

Any person who has attended three of the last four SystemVerilog committee meetings or 75% of all of the SystemVerilog committee meetings is eligible to vote.

All eligible SystemVerilog committee members should separately vote "yes," "no" or "abstain" on the following TWO proposals.

PROPOSAL #1 - Deprecate defparam
PROPOSAL #2 - Deprecate procedural assign & deassign

Regards - Cliff

PROPOSAL #1: Deprecate defparam

Modify the first paragraph of section 14, page 73 of draft 5 as shown:
Add a section (14.1.1) just before section 14.2, page 73 of draft 5 as shown:
Add sections 19 & 19.1, starting on page 81 of draft 5, as shown:

14.1 Introduction (informative)

Verilog-2001 has parameters, localparams and specparams, which are typically used either for controlling the dimensions of arrays, or for controlling delays. The parameter values can be set in three ways. They must be given a default value when declared. The default value of parameters can be overridden by the instantiation of the module, and this in turn can be overridden by positional parameter redefinition, named parameter redefinition or **`defparam`** statements. The latter can have a hierarchical name so that it does not need to be in the same module as the instantiation.

14.1.1 Defparam removal

The **`defparam`** statement may be removed from a future version of the language. See section 19.1.

Section 19
Verilog features under consideration for removal from SystemVerilog

The following Verilog language features are simulation inefficient, easily abused and the source of numerous design problems and are therefore being considered for removal from the SystemVerilog language.

19.1 Defparam statements (see section 14.1, and section 14.1.1)

Prior to the acceptance of the Verilog-2001 Standard (IEEE Std 1364-2001), it was common practice to change one or more parameters of instantiated modules using a separate **`defparam`** statement. **`Defparam`** statements are frequently the source of needless tool complexity and design problems.

When a **`defparam`** statement is placed in the Verilog code adjacent to the instance whose parameters are to be modified, and if the **`defparam`** statement makes a simple modification to just the parameters of the instantiated module and not hierarchically to any other parameters, the **`defparam`** statement has been used as intended.

The problem with **defparam** is that a **defparam** statement can precede the instance to be modified, can follow the instance to be modified, can be at the end of the file that contains the instance to be modified, can be in a whole separate file from the instance to be modified, can modify parameters hierarchically that in turn must again be passed to other **defparam** statements to modify parameters for a specific instance and can modify the same parameter from two different **defparam** statements placed in the same or in two separate files (with undefined results). Due to the many ways that a **defparam** can modify parameters, a Verilog compiler cannot insure the final parameter values for an instance until after all of the design files are compiled. This abuse of the **defparam** statement causes compiler tool inefficiencies and difficult debugging of convoluted parameterized Verilog designs.

Prior to Verilog-2001, the only other method available to change the values of parameters on instantiated modules was to use the **#(parameter)** redefinition syntax included as part of the instantiation itself. The disadvantage to using the **#(parameter)** redefinition syntax was that all parameters up to and including the parameter to be changed had to be placed in the correct order and assigned values, enclosed within the parentheses.

Verilog-2001 introduced the capability to pass parameters by name in the instantiation, which supplied all of the necessary parameter information to the model in the instantiation itself.

The practice of adding **defparam** statements to Verilog source files is highly discouraged and is being considered for removal from the SystemVerilog Standard. Engineers are encouraged to take advantage of the new named parameter passing capability introduced by and added to the IEEE Verilog-2001 Standard.

PROPOSAL #2 - Deprecate procedural assign & deassign

Add a section (8.10) at the end of the existing section 8, page 28 of draft 5 as shown:
Add sections 19(same as PROPOSAL #1) & 19.2, starting on page 81 of draft 5, as shown:

8.10 Procedural assign and deassign removal

Verilog and SystemVerilog support the procedural **assign** and **deassign** statements.

The procedural **assign** and **deassign** statements may be removed from a future version of the language. See section 19.2.

Section 19
Verilog features under consideration for removal from SystemVerilog

The following Verilog language features are simulation inefficient, easily abused and the source of numerous design problems and are therefore being considered for removal from the SystemVerilog language.

19.2 Procedural assign and deassign statements (see section 8.10)

Verilog has long had the confusing capability to do procedural **assign** and **deassign** assignments (not to be confused with a continuous assignment, which is placed outside of procedural blocks). In particular, the **assign** statement has a strange behavior that most Verilog users don't understand and engineers either find the behavior confusing or they get lucky and the **assign** statement is coded in such a way as to yield correct simulation results, despite the incorrect usage of the statement.

Most designs that include an **assign** statement inside of a procedural block could have been coded without the **assign** statement.

The problem with the procedural **assign** and **deassign** statements is that they are often used to control the value of a variable from multiple procedural blocks. Although simple **assign** and **deassign** statements can be understood, multiple **assign** statements are sometimes place in multiple procedural blocks to control the same variable and in large designs, it can be either difficult or impossible for a user to understand the priority of the multiple **assign** statements. Synthesis tools do not support the procedural **assign** and **deassign** statements because they can be coded in a manner that is impossible to implement in hardware.

The procedural **assign** statement has the strange behavior that it will update the LHS procedural variable anytime the RHS variable changes, even if the RHS variable is not in the sensitivity list. The concept that last **assign** wins is also confusing and restoring other non-**assign** procedural assignments to activity after a **deassign** statement is similarly confusing.

The **force**-**release** commands, have the same functionality (on both procedural and net variables) but they are rarely abused, are not synthesizable by any tool, and are clearly documented to be intended mostly for debugging purposes. Having both **assign**-**deassign** and **force**-**release** where the latter has higher priority and can also assign to wire types is confusing to most Verilog users.

The continuous assignment statement, placed outside of a procedural block, is still highly recommended for RTL coding of combinational logic.

The practice of using the **assign** and **deassign** statements inside of procedural blocks is highly discouraged and being considered for removal from the SystemVerilog Standard.
