Clarification for SV-BC 18h and 18i

Rev 0.2

3.1 (Informative) 2nd paragraph

OLD:
See section 3.3.1.
New
See sections 3.3.1 and 5.6.

5.1 (Informative) 3rd paragraph

OLD:
Verilog 2001 constants are literals, parameters, localparams and specparams. Verilog 2001 also has variables and nets. Variables must be written by procedural statements, and nets must be written by continuous assignments or ports.

NEW:
Verilog 2001 constants are literals, parameters, localparams and specparams. Verilog 2001 also has variables and nets. Variables must be written by procedural statements, and nets must be driven by continuous assignments or ports. SystemVerilog extends the functionality of variables by allowing them to either be written by procedural statements or driven by a single continuous assignment. For legacy behavior, a **`reg`** variable retains its Verilog-2001 functionality, whereas a **`logic`** variable is its SystemVerilog replacement.

5.6 Nets, regs, and Other Variables

Replace with:

In Verilog 2001, a net can only be written by one or more continuous assignments, primitive outputs or through module ports. The resultant value of multiple drivers is determined by the resolution function of the net type. If a net on one side of a port is driven by a variable on the other side, a continuous assignment is implied. A **`force`** statement can override the value of a net. When released, it returns to resolved value.

Verilog 2001 also states that only procedural statements can write to **`reg`** variables, including procedural continuous assignments. The last write determines the value. The **`force`** statement overrides the procedural assign statement, which in turn overrides the normal assignments. A **`reg`** variable cannot be written through a port, it must go through an implicit continuous assignment to a net.

In SystemVerilog, all data types except **`reg`** can be written either by one continuous assignment, or by one or more procedural statements, including procedural continuous assignments. It shall be an error to have multiple continuous assignments or a mixture of procedural and continuous assignments writing to the same variable. All data types except **`reg`** may write through a port

SystemVerilog variables may be packed or unpacked aggregates of other types. The assignments made to each element of a variable are independently examined using the longest static prefix rules. (See section TDB- SV-BC21) *[Note: This will define an assignment like* a[i] = expr; *to be treated as an assignment to all elements of an array]* It shall be an error to have a packed structure or array type written with a mixture of procedural and continuous assignments. Thus, an unpacked structure or array can have one element assigned procedurally, and another element assigned continuously. And, each element of a packed structure or array may each have a single continuous assignment. For example, assume the following structure declaration

```
struct {
        bit [7:0] A;
        bit [7:0] B;
        char C;
} abc;
```

The following statements are legal assignments to struct abc:

```
assign abc.C = sel ? 8'hBE : 8'hEF;
not(abc.A[0],abc.B[0]), (abc.A[1],abc.B[1]), (abc.A[2],abc.B[2]),
(abc.A[3],abc.B[3]);
always @(posedge clk) abc.B <= abc.B + 1;
```

The following additional statements are illegal assignments to struct abc:

```
// Multiple continuous assignments to abc.C
assign abc.C = sel ? 8'hDE : 8'hED;
// Mixing continuous and procedural assignments to abc.A
always @(posedge clk) abc.A[7:4] <= !abc.B[7:4];
```

For the purposes of the preceding rule, a declared variable initialization or a procedural continuous assignment is considered a procedural assignment. A **`force`** statement is neither a continuous or procedural assignment. A **`release`** statement will not change the variable until there is another procedural assignment, or a continuous assignment re-evaluates. A single **`force`** or **`release`** statement shall not be applied to a whole or part of a variable that is being assigned by a mixture of continuous and procedural assignments.

A continuous assignment is implied when a variable is connected to an **`input`** port declaration. Assignments to variable declared as an **`input`** port are illegal. A continuous assignment is implied when a variable is connected the **`output`** port of an instance. Assignments to a variable connected to the **`output`** port of an instance are illegal.

SystemVerilog variables of the same type may be connected to both sides of an **`inout`** port. In that case, variables on both the instance and declaration side of the **`inout`** port are treated as if they are a single variable. Procedural assignments may be made to either side of an **`inout`** port; last write wins. Continuous assignments may be made to different elements of a variable on either side of an **`inout`** port. See section 12.8 (port connection rules) for more information about port connection rules.

The compiler may issue a warning if a continuous assignment drives a strength other then St0, St1, StX, or HiZ.

Note that a SystemVerilog type cannot have an implicit continuous assignment as part of its declaration, the way a net data type can. An assignment as part of the logic declaration is a variable initialization, not a continuous assignment. For example:

```
wire w = vara & varb; // continuous assignment
logic v = consta & constb; // initial assignment, can have other
procedural assignments
logic l; // no initial assignment
assign l = vara & varb; // continuous assignment to a logic
real circ;
assign circ = 2.0 * PI * R; // continuous assignment to a real
```