Hi, All - (proposals also attached as Enumeration_proposals_20020328)

This email contains seven proposals to clarify and enhance SystemVerilog enumerated types. See details below.

Section 3.6 - Paragraph #1 Clarification Proposal
Section 3.6 - Paragraph #2 Clarification Proposal
Section 3.6 - Proposal to add a new paragraph #3 for clarification
Section 3.6 - Example #3 Clarification Proposal
Section 3.6 - Enumeration Range Proposal
Section 3.6 - Part-Select of Enumerated Variable Proposal
Section 3.6 - X & Z-Assignment of Enumerated Variable Proposal

Regards - Cliff Cummings

Draft 4, Section 3.6 - page 8
Section 3.6 - Paragraph #1 Clarification Proposal
PROPOSAL: expand the first paragraph as follows:
An enumerated type has one of a set of named values. In the following example, "light1" and "light2" are defined to be variables of the anonymous (unnamed) enumerated type that includes the three members: "red", "yellow" and "green."

Draft 4, Section 3.6 - page 8
Section 3.6 - Paragraph #2 Clarification Proposal
PROPOSAL: modify the second paragraph as follows:
The named values can be cast to integer types, and increment from an initial value of 0. This can be over-ridden. If integers are assigned to some but not all of the named values, unassigned named values are implicitly assigned an increment of the previous explicitly or implicitly assigned integer.

```
enum {bronze=3, silver, gold} medal; // silver=4, gold=5
enum (a=3, b=7, c) alphabet; // c=8
```

Draft 4, Section 3.6 - page 8
Section 3.6 - Proposal to add a new paragraph #3 for clarification
PROPOSAL: add a new paragraph and example after second example as follows:
It shall be illegal to explicitly or implicitly assign the same integer to more than one named value.

```
enum (a=0, b=7, c, d=8) alphabet; // c must be 8 and d is 8 - syntax error
```

Draft 4, Section 3.6 - page 9
Section 3.6 - Example #3 Clarification Proposal
PROPOSAL: for the third existing example of section 3.6, change the comment to:

```
// silver=4'h4, gold=4'h5 (all are 4 bits wide)
enum {bronze=4'h3, silver, gold} medal4;
```

Draft 4, Section 3.6 - page 9
Section 3.6 - Enumeration Range Proposal
PROPOSAL: permit enum to have a range to set a common size.
Proposed wording: Add the following before the paragraph starting with "The type name can be given ...," just before the typedef example.

Adding a constant range to the enum declaration can be used to set the size of the type. If any of the enum members are defined with a different sized constant, this shall be a syntax error.

```
// Error in the bronze and gold member declarations
enum [3:0] {bronze=5'h13, silver, gold=3'h5} medal4;
// Correct declaration - bronze and gold sizes are redundant
enum [3:0] {bronze=4'h13, silver, gold=4'h5} medal4;
```

Draft 4, Section 3.6 - page 9

Section 3.6 - Part-Select of Enumerated Variable Proposal

PROPOSAL: Add the ability to index, bit-select and part-select enumerated types:

Proposed wording: Add the following before the paragraph starting with "The type name can be given ...," just before the typedef example.

Enumerated types declared with a range shall permit bit-selection or part-selection access to the bits of the enumerated variables.

```
// state and next both have the range [1:0]
enum [1:0] { S0=2'b00, S1=2'b01, S2=2'b11 } state, next;

assign out1 = state[1]; // the out1 output is tied to bit[1] of the state variable
```

Reason: Some of the most efficient FSM designs utilize a technique called "output encoded FSMs" where one or more of the state bits are explicitly mapped to output bits. This creates registered outputs without any additional logic. Fast and small.

One of the following proposals are required to make enumerated types usable for efficient FSM synthesis.

Draft 4, Section 3.6 - pages 8-9

Section 3.6 - X & Z-Assignment of Enumerated Variable Proposal

2nd paragraph states:

"The values can be cast to integer types, and increment from an initial value of 0. This can be over-ridden."

PROPOSAL-A: add this as the last paragraph in section 3.6

The values of enumerated types can also be cast to all X's ('x) or all Z's ('z).

```
enum { S0=2'b00, S1=2'b01, S2=2'b11, XX='x } state, next;
```

-OR-

PROPOSAL-B: add this as the last paragraph in section 3.6

Any enumerated type can be assigned a value of all X's ('x) or all Z's ('z).

```
enum { S0=2'b00, S1=2'b01, S2=2'b11 } state, next;

always @(posedge clk or posedge reset)
  if (reset) state <= S0;
  else       state <= next;

always @* begin
  next = 2'bx;  // (SystemVerilog) next = 'x
  found_101 = 0;
  case (state)
    S0: if ( serial)                next = S2;
        else                        next = S0;
    S2: if (!serial)                next = S1;
        else                        next = S0;
    S1: begin
                                    next = S0;
          if (serial) found_101 = 1;
        end
```

```
  endcase
end
```

REASON: Efficient FSM coding style uses a default assignment of all X's to the next state variable for two very good reasons:

(1) it helps debug the FSM design. By making an all X's assignment to the next variable before entering the case statement, if the designer forgot to add one of the state transition assignments (for example, delete the else statement for the S2 case item) it will become very obvious during simulation that the RTL code is missing a state transition (the simulation will suddenly go to all X's (bleed-red in the waveform viewer) precisely when the missing transition occurred). This helps to quickly identify defects in the RTL code.

(2) it helps the synthesis tool optimize the design. X-assignments are treated as "don't-cares" for synthesis purposes, so the synthesis tool will optimize away the unused state encodings. Without x-assignments, we would require the very ugly a problematic (* synthesis, full_case *) attribute to accomplish the same synthesis-goal.

In the absence of the ability to make X-assignments to enumerated types, I would counsel students to ignore enumerated types for abstract FSM design because it will be easier to debug the FSM using parameters and X-assignments, to control state assignments and to enable synthesis optimization. This is a glaring hole in VHDL FSM design using enumerated types (trading off waveform enumerations for debug-ease and synthesis optimization).

The following example demonstrates the use of the X-assignment.

```
parameter S0=2'b00,
          S1=2'b01,
          S2=2'b11;

reg [1:0] state, next;

always @(posedge clk or posedge reset)
  if (reset) state <= S0;
  else       state <= next;

always @* begin
  next = 2'bx;  // (SystemVerilog) next = 'x
  found_101 = 0;
  case (state)
    S0: if ( serial)                next = S2;
        else                        next = S0;
    S2: if (!serial)                next = S1;
        else                        next = S0;
    S1: begin
                                    next = S0;
          if (serial) found_101 = 1;
        end
  endcase
end
```