Cliff Cummings - SystemVerilog 3.1-Draft 4 Review

First: I would again like to propose that "logic" be renamed to "ulogic" and "bit" be renamed to "ubit." This has had some support in the committees.

The latter names are less likely to appear as identifiers in existing models, plus, ulogic and ubit are unresolved, so the naming convention is more consistent for anyone with a VHDL background looking to adopt SystemVerilog. VHDL-types are going to think "logic" and "bit" are resolved types (ala std_logic & std_bit), which they are not.

DWS: This is not the place to make a change like this. Your committee has already voted on it and it is closed.

Consistency issue: Both testbench and test-bench are used extensively throughout the document. I recommend doing a global search-replace of test-bench with testbench.

Testbench will be flagged as a spelling error, but workbench is not a spelling error (they are roughly equivalent but spell-checkers do not know about HDL testbenches). The two best Verilog books on the market both use "testbench" as one word. (Writing Testbenches - Functional Verification of HDL Models by Janick Bergeron, and Principles of Verifiable RTL Design by Bening and Foster - a quick scan of the index shows multiple entries for testbench and none for either test bench or test-bench).

DWS: Done in LRM-238

Section 1 - no changes recommended

13.1 - 1st paragraph:
WAS: High-level and easy-to-use synchronization and communication mechanism are essential ...
PROPOSED: High-level and easy-to-use synchronization and communication mechanisms are essential ...

DWS: Done in LRM-239

13.1 - end of 2nd paragraph (noted in SystemVerilog-EC meeting)
DELETE: Lastly, SystemVerilog adds the **wait_var** mechanism that can be used to synchronize processes using dynamic data.

DWS: Done in LRM-2

13.2 - 2nd paragraph - different word???
WAS: Semaphore is a built-in class that provides
BETTER??: Semaphore is a pre-defined class that provides

DWS: We have used built-in throughout the LRM and never used pre-defined. Leave as is.

13.2.1, 31.2.2, 13.2.3, 13.2.4 - consistency
The examples and text use both "*key_count*" and "*keyCount*." Choose one or the other.
Per the SystemVerilog-EC meeting, delete most of 13.8 and move the rest to section 8.9.
(I did not review the rest of section 13)

DWS: Done in LRM-239

14.1 - 1st paragraph - better wording
WAS: Although SystemVerilog is used for more than simulation, the semantics of the language are defined for event directed simulation, and everything else is abstracted from this base definition.
PROPOSED: Although SystemVerilog is used for more than simulation, the semantics of the language are defined for event directed simulation, and all other event ordering is abstracted from this base definition.

DWS: This changes the meaning of the sentence in a way that is incompatible with what was voted on. No change.

14.2 - 1st paragraph - awkward wording
WAS: The SystemVerilog language is defined in terms of a discrete event execution model. The discrete event simulation is described in more detail in this section to provide a context to describe the meaning and valid interpretation of SystemVerilog constructs. These resulting definitions provide the standard SystemVerilog reference algorithm for simulation, which all compliant simulators shall implement. Note, though, that there is ...
PROPOSED: The SystemVerilog language is defined in terms of a discrete event execution model. The discrete event simulation is explained in more detail in this section to provide a context to describe the meaning and valid interpretation of SystemVerilog constructs. These resulting definitions provide the standard SystemVerilog reference algorithm for simulation, which all compliant simulators shall implement. Note, *deleted* that there is ...

DWS: I believe that it is defined in terms and not just explained. It is fundamental to the language and its definition. Removed the "though" in LRM-240.

14.3 - 5th paragraph after numbered time-slot list - wrong word??? (device does not make sense)
WAS: The sampling time of sampled data ... This `#1step` construct is a conceptual device that provides a method of defining when sampling takes place, and it is not creating a requirement that an event be created in this previous time slot. ...
CORRECTED???: The sampling time of sampled data ... This `#1step` construct is a conceptual delay that provides a method of defining when sampling takes place, and it is not creating a requirement that an event be created in this previous time slot. ...

DWS: Actually device is correct. It is not a conceptual delay it is a device or mechanism that provides a method etc… No Change.

PROBLEM WITH #1step DESCRIPTIONS??
According to 14.3, #1step is identical to taking the sample in the preponed region. Why not take the sample in the preponed region?
DWS: It is NOT identical. It is conceptually identical. Where it is taken is a performance and implementation issue. It may not even be taken, it may already exist and just be used.
According to 15.2, #1step forces sampling on a non-cycle boundary. Do we really want this? Does this cause simulation inefficiencies for cycle-based simulation?
DWS: This has been discussed and indicated that the existing implementations do not have an efficiency problem with this. We agreed to it as well in committee.
According to 15.2, sampling occurs in the read-only-synchronize phase of the previous event time step?
According to 15.3, 1step forces sampling at the end of the previous time step (event time step?)
According to 18.6, step is equal to the global time precision. What if there is not "timeprecision" and no "`timescale" directives?
Seems like a lot of descriptions for the same thing that could be quite confusing to implementers and users alike.
DWS: The concept of the global time precision was added to be consistent with IEEE 1364-2001 and is defined later taking into account missing directives. No problem here. There reason there are multiple

descriptions is that it is really not a simple concept. The whole goal of Section 15 was to define things in a clean way that the other chapters could reference (which they do). This was all discussed and voted on as is. No change.

14.3 - 5th paragraph after numbered time-slot list #1step definitions
... The new `#1step` sampling delay has been added to provide the ability to sample data immediately before entering the current time slot, ... Conceptually this `#1step` sampling is identical to taking the data samples in the preponed region of the current time slot.

15.2 - 1st & 2nd paragraphs after "clocking bus ..." example
In the above example, the first line declares a clocking domain called `bus` that is ... The last line adds the signal `addr` and overrides the default input skew so that `addr` is sampled one step before the positive edge of the clock.

Unless otherwise specified, the default **`input`** skew is `1`**`step`** and the default **`output`** skew is **`0`**. A **`step`** is a special time unit whose value is defined in Section 18.6. A `1`**`step`** input skew allows input signals to sample their steady-state values immediately before the clock event (i.e., at read-only-synchronize immediately before time advanced to the clock event). Unlike other time units, which represent physical units, a step cannot be used to set or modify either the precision or the timeunit.

15.3 - 1st & 2nd paragraphs after "clocking dram ..." example
An input skew of 1**`step`** indicates that the signal is to be sampled at the end of the previous time step. That is, the value sampled is always the signal's last value immediately before the corresponding clock edge.

18.6 - last paragraph
The global time precision is the minimum of all the **`timeprecision`** statements and the smallest time precision argument of all the **`\`timescale`** compiler directives (known as the precision of the time unit of the simulation in Section 19.8 of the IEEE 1364-2001 standard) in the design. The **`step`** time unit is equal to
the global time precision.

DWS: See above responses.

14.3 - pre-active, pre-NBA, post-NBA paragraphs - update references in three places ("below" is two pages away)
WAS: (see PLI callback control points, below).
PROPOSED: (see section 14.4, The PLI callback control points).

DWS: Done in LRM-241

14.3.1 - Somewhat confusing algorithm
This is a description with a routine (execute_simulation) that calls another routine (exectute_time_slot) that calls a third routine (execute_region), twice. This will be confusing to readers. Even though it is slightly more verbose, I recommend that the routines be combined into one large routine.

DWS: No change. This was thoroughly reviewed and approved.

15.1 - end or 3rd paragraph - missing word
WAS: SystemVerilog adds ... Depending on the environment, a test-bench may contain one or more clocking domains, each containing its own clock plus an arbitrary number signals.
CORRECTED: SystemVerilog adds ... Depending on the environment, a testbench may contain one or more clocking domains, each containing its own clock plus an arbitrary number of signals.

DWS: Fixed in LRM-237

15.3 - 2nd paragraph after "clocking dram ..." example - is this right??? What are we thinking???
An input skew of `#0` forces a skew of zero. Inputs with zero skew are sampled at the same time as their corresponding clocking event, but to avoid races, they are sampled in the observed region. Likewise, outputs with zero skew are driven at the same time as their specified clocking event, as nonblocking assignments (in the NBA region).

Adding another ugly #0 rule is abhorrent. What is intuitive about a #0 input sample being taken AFTER a nonblocking assignment?? #0 delayed assignments are scheduled before nonblocking assignments. This syntax will probably meet stiff resistance in the IEEE VSG. Why would I ever sample an input after nonblocking assignment updates? Could someone show me a useful example? Even if there is a useful example, the syntax should be changed.

I guess this means you can drive outputs (nba region) and sample those outputs if they are also tied to the inputs (observe region) and cause this to trigger always blocks that schedule more blocking and nonblocking assignments?

DWS: This is not another ugly #0 rule. It is replacing an ugly #0 rule with a well-defined behavior. You were there. This has been thoroughly discussed. No change.

15.4 - 1st paragraph - poor cross reference (above?? - this is described two section earlier)
WAS: Any signal in a clocking domain can be associated with an arbitrary hierarchical expression. As described above, a hierarchical expression is introduced by appending an equal sign (=) followed by the hierarchical expression:
BETTER??: Any signal in a clocking domain can be associated with an arbitrary hierarchical expression. As described in section 15.2, a hierarchical expression is introduced by appending an equal sign (=) followed by the hierarchical expression:

DWS: Fixed in LRM-242

15.4 2nd paragraph (after the "clocking cd1 ..." example) - wrong use of terms and confusing terms
In the IEEE Verilog Standard, a *part-select* is a range of bits within a word. A *slice* is something like a row of words out of an array. What is a "*combination of signals*??"
WAS: However, hierarchical expressions are not limited to simple names or signals in other scopes. They can be used to declare slices, concatenations, or combinations of signals in other scopes or in the current scope.
SEMI-BETTER (still needs correction): However, hierarchical expressions are not limited to simple names or signals in other scopes. They can be used to declare part-selects, concatenations, or ???combinations of signals??? in other scopes or in the current scope.

DWS: Slice is defined in 4.4 and is part of SystemVerilog 3.0. The other question I am too tired to figure out. Hopefully Arturo will respond.

15.7 - Section title typo?? (to match section text)
WAS: **Multiple clocking domain example**
WAS: **Multiple clocking domains example**

DWS: Fix in LRM-244

15.7 - first example - example implies that the write output is assigned in the initial block, therefore, the example header must be changed to include either the logic or reg data type. Also, cmd is bidirectional, so the body should also show a continuous assignment driving the cmd port (cannot be driven from a procedural block) and declare the cmd_reg shadow register.

```
program test( input phi1, input [15:0] data, output logic write,
              input phi2, inout [8:1] cmd, input enable
             );
       reg [8:1] cmd_reg;

       clocking cd1 @(posedge phi1);
              input data;
              output write;
              input state = top.cpu.state;
       endclocking

       clocking cd2 @(posedge phi2);
              input #2 output #4ps cmd;
              input enable;
       endclocking

       initial begin
              // program begins here
              // ...
              // user can access cd1.data , cd2.cmd , etc…
       end
       assign cmd = enable ? cmd_reg : 'z;
endprogram
```

DWS: Fixed in LRM-245

15.7 - last example - cmd & data both are vectors and must be declared. Use named port connections to make the example more clear, plus it is a better style

```
module top;
       logic phi1, phi2;
       wire [ 8:1] cmd; // cannot be logic (two bidir drivers)
       logic [15:0] data;

       test main( .phi1(phi1), .data(data), .write(write), .phi2(phi2),
                  .cmd(cmd), .enable(enable));
       cpu cpu1( .phi1(phi1), .data(data), .write(write));
       mem mem1( .phi2(phi2), .cmd(cmd), .enable(enable));
endmodule
```

DWS: Fixed in LRM-245 (although I did not do the named ports – in this case I find it no clearer and actually two much text – not to mention I am lazy).

15.8 - first example - wire declarations inside of an interface are almost illegal. Wires are only used if there are multiple drivers and NO procedural assignments. Procedural assignments are the most common testbench assignment type. The interface declarations should probably be of type "logic." Note that the comments in the program initial block indicate that access will be made to the interface variable names, implies procedural (reg or logic) assignments.

Better yet, give me the enhancement I requested months ago. Allow procedural assignments to wire-types!! Who knows how many other 3.1 interface examples use illegal wire declarations?

```
interface bus_A (input clk);
       logic [15:0] data;
```

```
        logic write;
        modport test (input data, output write);
        modport dut (output data, input write);
endinterface

interface bus_B (input clk);
        logic [8:1] cmd;
        logic enable;
        modport test (input enable);
        modport dut (output enable);
endinterface
```

DWS: Fixed in LRM-246

15.8 - module top example - to make the example clear, use named port connections. By the way, what is generating the phi1 and phi2 clocks??

```
module top;
        logic phi1, phi2;

        bus_A a (.clk(phi1));
        bus_B b (.clk(phi2));
        test main (.a(a), .b(b));
        cpu cpu1 (.a(a));
        mem mem1 (.b(b));
endmodule
```

DWS: No change since what is there is correct.

15.8 - paragraph before last example and last example. "Alternatively" is confusing. Dangling clocking blocks are confusing.

The clocking domains of program test could have also been written using both interfaces and hierarchicalexpressions as shown program test2 below:

```
program test2 (bus_A.test a, bus_B.test b);

        clocking cd1 @(posedge a.clk);
                input data = a.data;
                output write = a.write;
                inout state = top.cpu.state;
        endclocking

        clocking cd2 @(posedge b.clk);
                input #2 output #4ps cmd = b.cmd;
                input enable = b.enable;
        endclocking
        // ...
endprogram
```

DWS: True but what is there is not unclear. Made a change to clarify the sentence in LRM-246.

15.10 - The paragraph before the example says that, "What constitutes a cycle is determined by the default clocking in effect" ?? is it only determined by the default clocking??

DWS: That is what it says and it is an error if there is no default clocking.

15.10 - example comments - misleading??

Example:

```
##5; // wait 5 cycles using the default clocking
##j + 1; // wait j+1 cycles using the default clocking
```

Does this wait for 5 and (j+1) "cycles" or "clocking events" (it would be different for @(clk) and @(posedge clk))??

DWS: The whole section is using clocking events and clock cycles the same here. What is the confusion?

15.11 - paragraph before "program test ..." example. Unnecessary word - "other"
WAS: A default clocking is valid only within the scope containing the default clocking specification. This scope includes the module, interface, or program that contains the declaration as well as any nested modules or interfaces. It does not include other instantiated modules or interfaces.
BETER: A default clocking is valid only within the scope containing the default clocking specification. This scope includes the module, interface, or program that contains the declaration as well as any nested modules or interfaces. It does not include *deleted* instantiated modules or interfaces.

DWS: Fixed in LRM-247

15.11 - last example - Enclose example within "module top ...example... endmodule, otherwise, the clocking declarations look like illegal $root declarations.

DWS: Fixed in LRM-194

15.12 - example synchronization events - Consistency: "clock domain" or "clock-domain?" (with or without hyphen?)

DWS: clock-domain is used consistently in the LRM

15.12 - more consistency issues - better wording to be consistent with earlier examples??

— Wait for the positive edge of the signal `enable` of clock domain `ram_bus`.

```
@(posedge ram_bus.enable);
```

DWS: But it can also be called the signal ram_bus.enable. No Change.

— Wait for the falling edge of the specified 1-bit slice `sign[a]` of clock domain `dom`. Note that the index `a` is evaluated at runtime.

```
@(negedge dom.sign[a]);
```

DWS: What is wrong with what is there?

— Wait for either the next positive edge of `sig1` or the next change of `sig2` of clock domain `dom`, whichever happens first.

```
@(posedge dom.sig1 or dom.sig2);
```

DWS: What is wrong with what is there?

— Wait for the either the negative edge of `sig1` or the positive edge of `sig2` of clock domain `dom`, whichever happens first.

```
@(negedge dom.sig1 or posedge dom.sig2);
```

DWS: What is wrong with what is there?

DWS: No changes

15.12 - last paragraph - in which event region??
The values used by the synchronization event control are the synchronous values, that is, the values sampled at the corresponding clocking event **which queue??**.

DWS: The term queue is not specified here. The discussion of regions is in 15.13.

15.13 - 2nd paragraph - confusing text needs better explanation
Samples happen immediately (the calling process does not block **what does this mean??**). When a signal appears in an expression, it is replaced by the signal's sampled value, that is, the value that was sampled at the last sampling point.

DWS: It means it does not block.

15.14 - 1st paragraph and example - change "slice" to "part-select" (explained earlier - this should be checked globally).

DWS: Likewise see above response.

15.14 - Middle paragraph should "cycles" be "event transitions?"
WAS: The *event_count* is an integral expression that optionally specifies the number of clocking events (i.e. cycles) that must pass before the statement executes. Specifying ...
BETTER??: The *event_count* is an integral expression that optionally specifies the number of clocking events (i.e. event transitions) that must pass before the statement executes. Specifying ...

DWS: No. These are cycles not event transitions.

15.14 - Next paragraph - Do we need to specify which event region for clarity??
The second form of the synchronous drive uses the intra-assignment syntax. An intra-assignment eventcount specification also delays execution of the assignment. In this case the process does not block and the right-hand side expression is evaluated when the statement executes.

DWS: I do not think so.

15.14 - Next examples - Again, is "cycle" correct or should it be "event transitions??
WAS: Examples:

```
bus.data[3:0] <= 4'h5;      // drive data in current cycle
##1 bus.data <= 8'hz;       // wait 1 (bus) cycle and then drive data
##2; bus.data <= 2;               // wait 2 default clocking cycles, then
drive data
```

BETTER??: Examples:

```
bus.data[3:0] <= 4'h5;      // drive data in current cycle
##1 bus.data <= 8'hz;       // wait 1 (bus) event transition and then
drive data
```

```
##2; bus.data <= 2;              // wait 2 default clocking event
transitions,
                                // then drive data
```

DWS: I believe you are incorrect. See above.

15.14.2 - 1st paragraph - typo - bad hypehnation
WAS: When more than one synchronous drive is applied to the same clocking domain output (or inout) at the same simulation time, the driven values are checked for conflicts. When conflicting drives are detected a runt-ime error is issued, and each conflicting bit is driven to X (or 0 for a 2-state port).
CORRECTED: When more than one synchronous drive is applied to the same clocking domain output (or inout) at the same simulation time, the driven values are checked for conflicts. When conflicting drives are detected a run-time error is issued, and each conflicting bit is driven to X (or 0 for a 2-state port).

DWS: Fixed in LRM-243

15.14.2 - 2nd paragraph after 1st example - Is this statement true??
When the same variable is an output from multiple clocking domains, the last drive determines the value of the variable. This allows a single module to model multi-rate devices, such as a DDR memory, using a different clocking domain to model each active edge. For example:
(Is this true even when the drives happen in the same time-step?)

DWS: I believe it to be correct.

15.14.2 - last paragraph - no hyphen in nonblocking
WAS: Clock-domain outputs driving a net (i.e. through different ports) cause the net to be driven to its resolved signal value. When a clock-domain output corresponds to a wire, a driver for that wire is created that is updated as if by a continuous assignment from a register inside the clock-domain that is updated as a non-blocking assignment.
CORRECTED: Clock-domain outputs driving a net (i.e. through different ports) cause the net to be driven to its resolved signal value. When a clock-domain output corresponds to a wire, a driver for that wire is created that is updated as if by a continuous assignment from a register inside the clock-domain that is updated as a nonblocking assignment.

DWS: Fixed in LRM-248

18.6 - paragraph in the middle of the section (hard to be precise with all the deleted sections)
WAS: There can shall be only one time unit and one time precision for any module or interface definition, or in **`$root`**. ...
CORRECTED: There *deleted* shall be only one time unit and one time precision for any module or interface definition, or in **`$root`**. ...

DWS: Fixed in LRM-249