20070205

The new scheduling semantics address the first half of Mantis item 549. I anticipate that Doug Warmke's 890 proposal will address the rest of the issues and then we should be able to close this item.

Regards - Cliff Cummings

Issues with Section 15 "Scheduling Semantics"

Chapter 15 of the LRM describes the new scheduling semantics of a SystemVerilog simulator. There are a number of inconsistencies and incorrect statements in this chapter. However, when analyzing how new time queues are used by assertions, clocking blocks, and programs it seems clear how the new scheduler should operate despite the inaccuracies.

1 Inconsistencies

1.1 Missing pre-active region in section 15.3.1

The pre-active region described as the 2nd ordered region in section 15.3 is missing from the reference algorithm in section 15.3.1 on page 179. We believe it should immediately follow the line

```
      execute_region(preponed);
```

and should read

```
      execute_region(pre-active);
```

Completed as described.

1.2 Iterative Regions

The set of regions that are defined as the iterative regions on the bottom of page 178 and again on page 179 are described by the reference algorithm in section 15.3.1 as placing their events in the active region and then running the active region followed by a rescan of all regions. However, the diagram in figure 15-1 only shows that certain regions are capable of returning to just before the preactive region. In particular, the regions for pre-NBA, post-NBA, and post-Observed do not appear to be able to return. We believe the diagram in figure 15-1 is in error and should be updated to show that these regions also return.

Completed as described.

2 Inaccuracies

2.1 Preponed Region Definition

In section 15.3 on the bottom of page 178, the preponed region is defined as a region for PLI callbacks that allow a user to access data in the current time slot before any net or variable has changed state. This appears to be more closely related to the definition for the pre-active region, not the definition of the preponed region.

The preponed region is used by clocking blocks and assertions as a sampling point to obtain consistent sampled values before any net or variable has changed state. In order to achieve this, the preponed region can not allow any modification of simulation state during its execution by either the simulation or PLI. Further, PLI constructs need not use this region as they can use the region immediately following (pre-active) for the same purpose.

We believe the wording of this definition should change to say that

"The Preponed region is specifically for sampling by SystemVerilog constructs to access data in the current time slot before any net or variable has changed state. This region is not a PLI region, and PLI constructs may not schedule into it."

If it is decided to allow PLI into the proponed region, despite having the preactive region for the same purpose, then wording similar to that used for the postponed region regarding the illegality of writing values must be added to the preponed region's definition in order to ensure that the preponed region is able to provide its intended function for assertions and clocking blocks.

Not completely accurate. Both Preponed and Pre-Active regions are required to allow PLI commands to sample values in the Preponed region before the PLI is allowed to make updates in the Pre-Active region. For this reason, the Preponed region description indicates that PLI commands are only allowed to read and not write in this region.

Preponed is now described as both a simulation and PLI region.

2.2 Observed Region Definition

Similar to the preponed region, the observed region is used for sampling by new SystemVerilog constructs. In order to be successful in this endeavor, this region must either disallow PLI callbacks during its run or disallow those pli operations from writing to any net or variable. We recommend that it be modified consistent with the modifications necessary for the preponed region as described by section 2.1 above.

Observed region now indicates that "PLI callbacks are not allowed in the Observed region."

2.3 Postponed Region Definition

The wording indicates that the postponed region is only used for PLI callbacks. This, however, is not the case, as this region is also used for non-PLI activities such as $monitor. This wording should be updated to indicate its multi-purpose usage.

Description has been updated to be consistent with this request.

2.4 cbAtEndOfSimTime Definition

Although the name of this callback region indicates it should be at the end of the simulation time, it is also expected to be a read-write region. As such the table 15-3 in section 15.4 which places it in the postponed region is inaccurate. Originally this region was targeted for the post-NBA location which would have been the last read-write region. However, with the addition of the SystemVerilog constructs it is unclear where this belongs. Possible answers include post-NBA, post-Observed, and Reactive. We believe the most correct answer is in the Reactive region as this is the last read-write region of the new simulation cycle.

Table 15-3 now shows cbAtEndOfSimTime executes in the Pre-Postponed region.

Doug Warmke to address the remaining issues.

3 Interpretation

3.1 Program Blocks

The specification in the reference algorithm in section 15.3.1 that events are moved from their region of origin to the active region can be somewhat misleading. Although this accurately describes the complexities around the triggering of NBA's and the events and subsequent process wakeups that they cause, such is not the case for other regions, in particular the reactive region.

Program blocks are specified to run in the reactive region. If an event in a program block were to trigger an event that caused another process to wakeup in the program block there are two possible interpretations. One interpretation is that since we are now on the active region, this event and the corresponding wakeup should wait until the next reactive region. Another possible interpretation is that since we were executing a portion of the program block we should execute the process in the current region rather than wait for the next region.

Chapter 15 is worded such that the first interpretation is correct. However, Chapter 17 is worded such that the second interpretation is correct. We believe users would find the second interpretation more to their liking and thus propose that it be used.

However, in so interpreting it, the LRM's specification of chapter 15 is no longer accurate. The paragraph that describes the interaction with program blocks would need to change from "are scheduled" to "run" to be consistent. Some additional changes may also be necessary in section 15.3.1 in the reference algorithm. We believe the most logical change would simply be to remove the portion about moving events to the active region and running the active region. Instead we should simply run the first non-empty region. Any additional wording that specifies what gets scheduled into the active region should also change to specify the current region, whatever it is. This should accurately describe the level of indeterminism within any given simulation while preserving the correct functionality for events triggered in the NBA region as well as events triggered in the Reactive region.

3.2 Missing Callbacks

A number of the new regions specify that PLI can schedule into them. However, at no place in the LRM is there any definition of the callbacks that could be used for the preponed, observed, post-observed, or reactive regions. We interpret this to mean that there is no such callback, and user code can not interact with these regions. Although this satisfies the special requirements of the preponed and observed regions, this does make the need for the post-Observed region somewhat unclear as it is impossible to schedule anything into it. It seems logical that we should define new callback names for each of these regions.

4 Backwards Compatibility

4.1 #0 and the inactive region

SystemVerilog declares that an explicit #0 requires a delay into the inactive region. This may create a backwards compatibility issue as historically simulations treated #0 differently depending on the context in which it was used. For example, if used to describe the delay on a gate or wire, the #0 delay means the same thing as no delay. Similarly, the #0 on the RHS of an NBA (and even a regular blocking assign) would be the same as no delay. Only as a delay control on a statement in procedural code does #0 cause a delay to the inactive queue (actually meaning the next "active" queue).

In certain SystemVerilog situations, #0 means to perform an action in the Observed region, or delay till the next Reactive region. We believe this wording needs to be modified to either allow for implementation dependent variations or construct dependent variations. A better solution would be to simply remove the inactive queue from the definition and define that the particular #0 delay in procedural RTL code delay to the next appropriate simulation region of the appropriate type. This could then be done in the section on statement delays rather than in the section on time queues.

4.2 cbReadWriteSynch Callback Definition

Table 15-3 in section 15.4 specifies the translation from historic VPI cbReadWriteSynch callback to the new SystemVerilog regions. Although this is accurate for some simulators, it is inaccurate for certain others, which had been using the pre-NBA region. Rather than define which region this callback occurs in for SystemVerilog, this callback should be deprecated.

5 Organization

5.1 Region Defintions

The region definitions as found in section 15.3 are somewhat difficult to piece together. Some regions have more than one defining paragraph spread across multiple pages. We believe it would be advisable to organize these paragraphs in some manner. One possible organization is to have a single defining paragraph per region that provides all the necessary information.

5.2 Inter-document references

Section 15.3 has a couple of paragraphs that allude to the time queues usage in assertions, clocking blocks, and programs. These references are incomplete to describe the functionality of these constructs, and yet they provide more detail than is really necessary. It may be advantageous to simply refer the reader to the other sections of the document that describe the functionality of these constructs in more detail.