5. SystemVerilog 3.1 scheduling semantics

Extracted from IEEE Std. 1364-2001
Edited and expanded by Phil Moorby Feb 2003
Edited by David Smith Feb 2003
Version 6.

5.1 Execution of a hardware model and its verification environment

The balance of the sections of this standard describes the behavior of each of the elements of the language. This section gives an overview of the interactions between these elements, especially with respect to the scheduling and execution of events.

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.

5.2 Event simulation

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 a great deal of choice in the definitions that follow, and differences in some details of execution are to be expected between different simulators. In addition, SystemVerilog simulators are free to use different algorithms than those described in this section, provided the user-visible effect is consistent with the reference algorithm.

A SystemVerilog description consists of connected threads of execution or processes. Processes are objects that can be evaluated, that may have state, and that can respond to changes on their inputs to produce outputs. Processes are concurrently scheduled elements, such as initial blocks. Example of processes include, but are not limited to, primitives, initial and always procedural blocks, continuous assignments, asynchronous tasks, and procedural assignment statements.

Every change in state of a net or variable in the system description being simulated is considered an *update event*.

Processes are sensitive to update events. When an update event is executed, all the processes that are sensitive to that event are considered for evaluation in an arbitrary order. The evaluation of a process is also an event, known as an *evaluation event*.

Evaluation events also include PLI callbacks, which are points in the execution model where user-defined external routines can be called from the simulation kernel.

In addition to events, another key aspect of a simulator is time. The term *simulation time* is used to refer to the time value maintained by the simulator to model the actual time it would take for the system description being simulated. The term *time* is used interchangeably with simulation time in this section.

To fully support clear and predictable interactions, a single time slot is divided into multiple regions where events may be scheduled that provide for an ordering of particular types of execution. This allows properties and checkers to sample data when the design under test is in a stable state. Property expressions can be safely evaluated, and testbenches can react to both properties and checkers with zero delay, all in a predictable manner. This same mechanism also allows for non-zero delays in the design, clock propagation, and/or stimulus and response code to be mixed freely and consistently with cycle accurate descriptions.

5.3 The stratified event scheduler

A compliant SystemVerilog simulator must maintain some form of data structure that allows events to be dynamically scheduled, executed and removed as the simulator advances through time. The data structure is normally implemented as a time ordered set of linked lists, which are divided and sub-divided in a well defined manner.

The first division is by time. Every event has one and only one simulation execution time, which at any given point during simulation may be the current time or some future time. All scheduled events at a specific time define a *time slot*. Simulation proceeds by executing and removing all events in the current simulation time slot before moving on to the next non-empty time slot, in time order. This procedure guarantees that the simulator never goes backwards in time.

A time slot is divided into a set of ordered regions:

1) preponed
2) pre-active
3) active
4) inactive
5) pre-NBA
6) NBA
7) post-NBA
8) observed
9) post-observed
10) reactive
11) postponed

The purpose of dividing a time slot into these ordered regions is to provide predictable interactions between the design and testbench code.

Except for the observed and reactive regions and the post-observed PLI region, these regions essentially encompass the Verilog 1364-2001 standard reference model for simulation, with exactly the same level of determinism. This means that legacy Verilog code will continue to run correctly without modification within the new mechanism. The postponed region is where the monitoring of signals, and other similar events, takes place. No new value changes are allowed to happen in the time slot once the postponed region is reached.

The observed and reactive regions are new in the SystemVerilog 3.1 standard, and events are only scheduled into these new regions from new language constructs.

The observed region is for the evaluation of the property expressions when they are triggered. It is essential that the signals feeding and producing *all* the clocks to the property expressions have stabilized, so that the next state of the property expressions can be calculated deterministically. A criterion for this determinism is that the property evaluations must only occur once in any clock triggering time slot. During the property evaluation, pass/fail code will be scheduled to be executed in the reactive region of the current time slot.

The sampling time of sampled data for property expressions is controlled in the clock domain block. The new #1step sampling delay has been added to provide the ability to sample data immediately before entering the current time slot, and is a preferred construct over other equivalent constructs in order to allow the 1step time delay to be parameterized. 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. Conceptually this #1step sampling is identical to taking the data samples in the preponed region of the current time slot.

Code specified in the program block, and pass/fail code from property expressions, are scheduled to occur in the reactive region.

The pre-active, pre-NBA, and post-NBA are new in the SystemVerilog 3.1 standard but support existing PLI callbacks. The post-observed region is new in the SystemVerilog 3.1 standard and has been added for PLI support.

The pre-active region is specifically for a PLI callback control point that allows for user code to read and write values and create events before events in the active region are evaluated (see PLI callback control points below).

The pre-NBA region is specifically for a PLI callback control point that allows for user code to read and write values and create events before the events in the NBA region are evaluated (see PLI callback control points below)..

The post-NBA region is specifically for a PLI callback control point that allows for user code to read and write values and create events after the events in the NBA region are evaluated (see PLI callback control points below)..

The post-observed region is specifically for a PLI callback control point that allows for user code to read values after properties are evaluated (in observed or earlier region).

The flow of execution of the event regions is specified in figure 5.1.

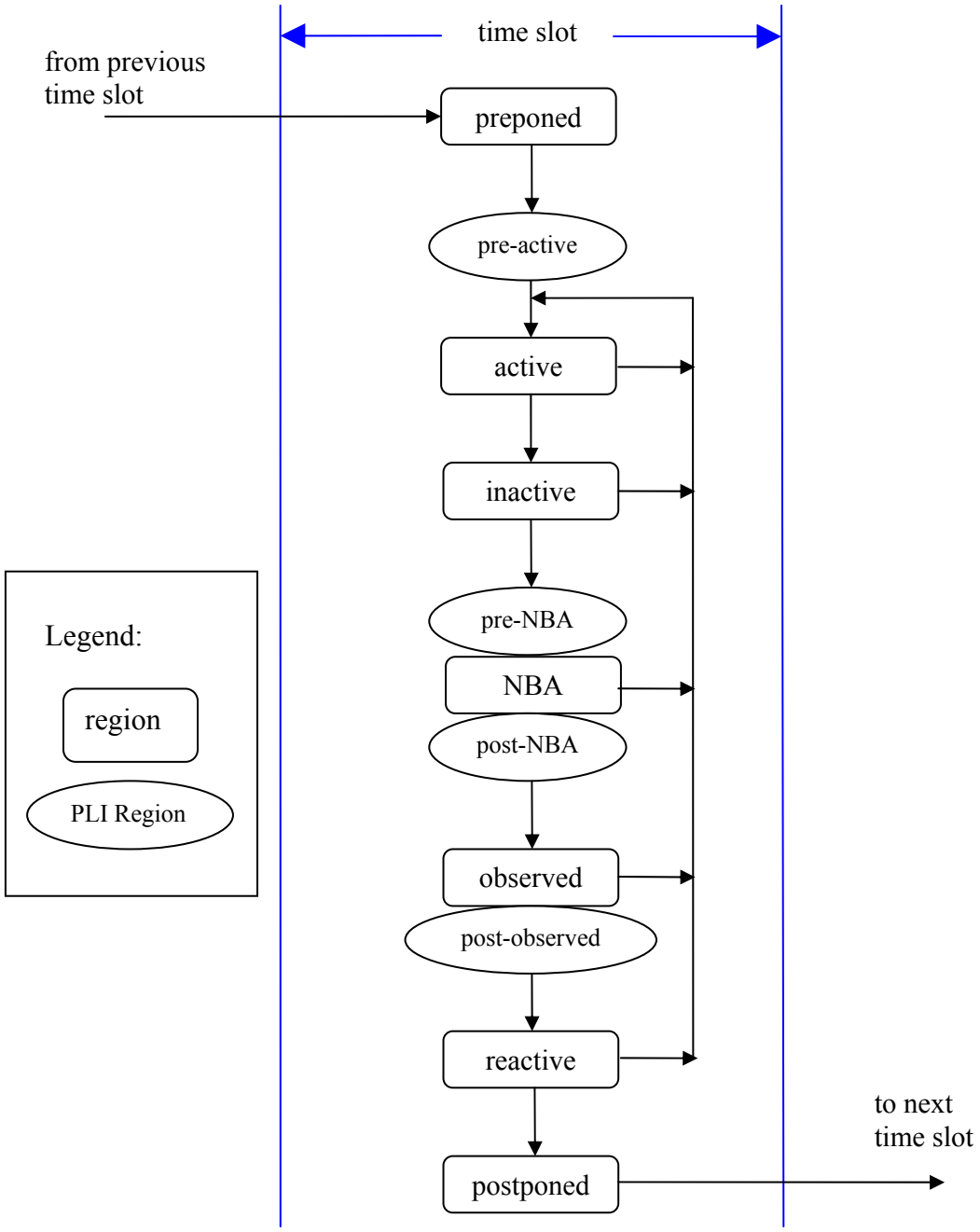

Figure 5.1. The SystemVerilog flow of time slots and event regions

The active, inactive, pre-NBA, NBA, post-NBA, observed, post-observed and reactive regions are known as the *iterative* regions.

The *preponed* region is specifically for a PLI callback control point that allows for user code to access data at the current time slot before any net or variable has changed state.

The *active* region holds current events being evaluated and can be processed in any order.

The *inactive* region holds the events to be evaluated after all the active events are processed.

An *explicit zero delay* (#0) requires that the process be suspended and an event scheduled into the inactive region of the current time slot so that the process can be resumed in the next inactive to active iteration.

A non blocking assignment creates an event in the NBA region, scheduled for current or a later simulation time.

The *postponed* region is specifically for a PLI callback control point that allows for user code to be suspended until after all the active, inactive and non blocking assign update regions have completed. Within this region, it is illegal to write values to any net or variable, or to schedule an event in any previous region within the current time slot.

5.3.1 The SystemVerilog simulation reference algorithm

```
execute_simulation {
        T = 0;
        initialize the values of all nets and variables;
        schedule all initialization events into time 0 slot;
        while (some time slot is non-empty) {
                move to the next future non-empty time slot and set T;
                execute_time_slot (T);
        }
}

execute_time_slot {
        execute_region (preponed);
        while (some iterative region is non-empty) {
                execute_region (active);
                scan iterative regions in order {
                        if (region is non-empty) {
                                move events in region to the active region;
                                break from scan loop;
                        }
                }
        }
```

```
        execute_region (postponed);
}

execute_region {
        while (region is non-empty) {
                E = any event from region;
                remove E from the region;
                if (E is an update event) {
                        update the modified object;
                        evaluate processes sensitive to the object and possibly schedule
                        further events for execution;
                } else { /* E is an evaluation event */
                        evaluate the process associated with the event and possibly
                        schedule further events for execution;
                }
        }
}
```

The iterative regions and their order are: active, inactive, pre-NBA, NBA, post-NBA, observed, post-observed and reactive.

5.4 The PLI callback control points

There are two kinds of PLI callbacks, those that are executed immediately when some variable changes value in an update event, and those that are explicitly registered as a one-shot evaluation event.

It is possible to explicitly schedule a PLI callback event in any region. Thus, an explicit PLI callback registration is identified by a tuple: (time, region).

The following list provides the mapping from the various current PLI callbacks:

tf_synchronize	(time, pre-NBA)
tf_isynchronize	(time, pre-NBA)
tf_rosynchronize	(time, postponed)
tf_irosynchronize	(time, postponed)
cbReadWriteSynch	(time, post-NBA)
cbAtStartOfSimTime	(time, pre-active)
cbReadOnlySynch	(time, postponed)
cbNBASynch	(time, pre-NBA)
cbAtEndOfSimTime	(time, postponed)
cbNextSimTime	(time, pre-active)
cbAfterDelay	(time, pre-active)