

## Preamble

The Champions provided feedback on this proposal on 2007-08-18. Below are responses to the points made by the Champions.

- The proposal doesn't discuss what the global clock is with respect to compilation units

Global clocking is not directly related to compilation units. The requirement is that there be at most one global clocking declaration after the elaboration stage. The global clocking declaration applies to the entire elaborated SystemVerilog model, regardless of the particular compilation unit in which it appears.

- There is no concept of globals in SystemVerilog

The concept of globals is mentioned in the LRM. See for example Clause 3.12.3 "Simulation time unit". It states:

"The global time precision, also called the simulation time unit, is the minimum of all the timeprecision statements and the smallest time precision argument of all the `timescale compiler directives in the design. The step time unit is equal to the global time precision."

There is also a notion of a global symbol/task/function in Clause 34.

- It should be a tool option

This seems to be problematic since a transition to a new tool becomes unpredictable. We should take advantage of standardizing it, to make this concept well-defined.

- Strange that it is global and needs to be in a module, it should be in compilation unit space

Since the global clocking defines a clocking event, it makes sense to consider it as a kind of clocking, and not as an absolutely new construct. The LRM forbids the declaration of a clocking block in compilation unit space. This is why the global clocking declaration rules have been defined in a similar way. The consistency is crucial here.

- Not sure it fits in with the general usage of the language

In the introduction to the LRM it is written:

"SystemVerilog enables the use of a unified language for abstract and detailed specification of the design, specification of assertions, coverage, and testbench verification that is based on manual or automatic methodologies."

This implies that SystemVerilog should answer the needs of assertion-based formal verification. For formal verification of synchronous systems the notion of the reference clock is important, especially for multiclock

design and for building abstract formal models of the system. The detailed description of the motivation of this feature may be found in the Rationale below.

Since formal verification of RTL is in scope of the language, the notion of the reference clock needs to be covered by the language.

- The LRM is based on event based simulation.

SystemVerilog provides a unified language for writing both design models and associated assertion-based specifications. These are used not only in simulation, but also in formal verification, where the tractability of the verification computation usually hinges on abstraction and simplification of the event-based semantics. The purpose of the global clocking construct is to provide a way to specify the primary clock that synchronizes transitions in formal models and thereby obtain efficiencies in formal verification computations.

- If in a package would need to be imported

Global clocking cannot be declared in a package. Global clocking may be referenced by items declared within a package.

- Not sure why need this global thing, we already have clocking blocks.

The purpose of the global clocking construct is to provide a way to specify the primary clock that synchronizes transitions in formal models and thereby obtain efficiencies in formal verification computations. Using clocking events defined by regular clocking blocks, even the default for a particular scope, does not accomplish specification of the global primary clock for the entire formal model.

- The proposal says that global clock is for the design. Can the testbench use the global clock? (proposal seems to not say)

The proposal language has been changed to say that a global clock applies to an entire elaborated SystemVerilog model and that `$global_clock` may be invoked anywhere that a clocking event may be specified.

- Proposal shouldn't use the term design if it isn't clear

The proposal language has been changed to say that a global clock applies to an entire elaborated SystemVerilog model.

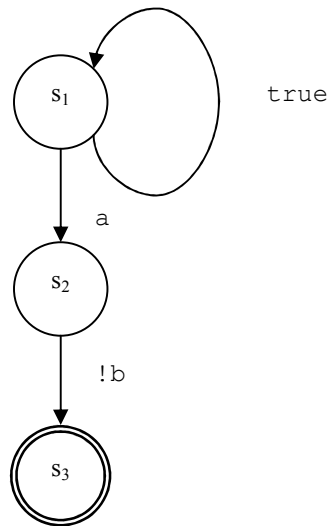
## Rationale

The goal of the global clocking concept proposed here is to provide a definition of the primary clock synchronizing transitions in formal models.

For formal verification purposes DUTs are usually considered as a set of states and transitions between the states. In verification of RTL models, the transitions are synchronous. In other words, at each step one and only one transition (possibly idle) is taken. This sequence of steps defines the primary clock in a natural way, and all system transitions are synchronized by it. The primary clock is fair, meaning that it never stops ticking. In the metalanguage describing the formal semantics of SystemVerilog Assertions (see Annex F), the primary clock is denoted as `@1`, but this notation cannot be used in the language itself because the event `@1` never happens. To be able to specify the primary clock in the language, a special notation is needed, and this proposal suggests that the notation be `$global_clock`.

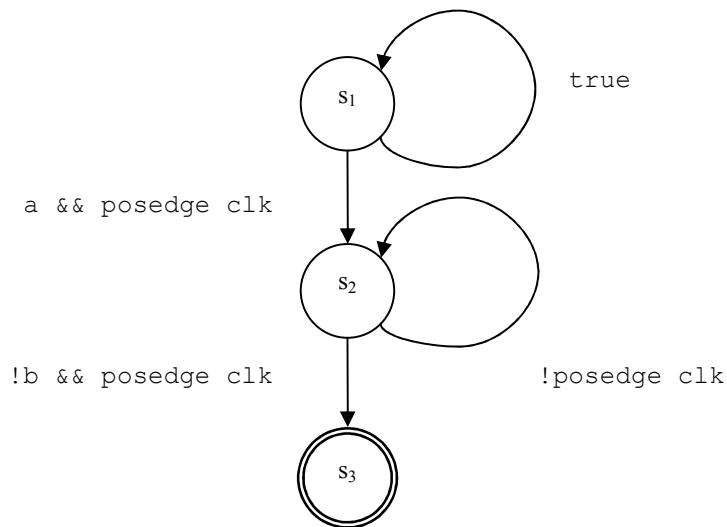
Formal verification of assertions controlled by the primary clock is more efficient than the verification of assertions controlled by any other clock. Compare the failure automata generated for the following two assertions:

```
always_p1: assert property (@$global_clock a |=> b);
```



and

```
always_p2: assert property (@(posedge clk) a |=> b);
```



The first automaton is simpler: it does not have clocking conditions along its edges, and it does not have an extra idle edge for awaiting the clock. These assertions are not, of course, equivalent, but often for practical purposes either of them is acceptable.

The notion of the primary clock allows specification of the next value of a variable – the value of the variable assigned at the end of a given transition. This value is set at the next tick of the primary clock `$future_gclk(v)` (this function is not part of this proposal and is referenced here only for the purpose of illustration). Using the primary clock and next value functions is very natural for specifying stability properties. Consider the assertion saying that some signal `sig` should always be stable. The natural way to write this assertion is:

```
assert property(@$global_clock sig == $future_gclk(sig));
```

Without the notion of global clock this assertion has to be rewritten as:

```
assert property(@clk ##1 sig == $past(sig));
```

which has the following drawbacks (I intentionally use `$past` and not `$stable` here to avoid a discussion about future stability functions):

1. Most users will write the above assertion as `assert property(@clk sig == $past(sig));` without skipping the initial phase, and thus will check that `sig` is always 0 (if `sig` has a two-value type), instead of the intended assertion.
2. The automaton of the second assertion is more expensive than that of the first assertion since it contains more states.
3. The first assertion is more robust since it doesn't need to be changed when a faster clock is introduced in the design.

As in the previous example, these assertions are not equivalent, but often for practical purposes either of them is acceptable.

In order to use the primary clock in simulation, a real clocking event must be associated with it. Therefore the global clocking construct is suggested. There is no obligation that all property clocks be synchronized with the global clock in simulation – this is purely an issue of methodology or tool implementation.

Thus the introduction of global clocking and the `$global_clock` construct is backward compatible and implies no penalty in simulation.

### 14.3 Clocking block declaration

REPLACE

```
clocking_declaration ::=  
    [ default ] clocking [ clocking_identifier ] clocking_event ;  
        { clocking_item }  
    endclocking [ : clocking_identifier ]
```

WITH

```
clocking_declaration ::=  
    [ default ] clocking [ clocking_identifier ] clocking_event ;  
        { clocking_item }  
    endclocking [ : clocking_identifier ]  
    | global clocking [ clocking_identifier ] clocking_event ; endclocking [ : clocking_identifier ]
```

ADD subclause 14.13 Global clocking

### 14.13 Global clocking

Note to editor: Shift the numeration of the following sections accordingly.

One clocking block may be declared as the *global clocking* for an entire elaborated SystemVerilog model.

The syntax for the `global clocking` declaration is as follows:

---

```
clocking_declaration ::=  
    ...  
    | global clocking [clocking_identifier] clocking_event ; endclocking [ : clocking_identifier ]
```

---

*Syntax 14-1—Clocking block syntax (excerpt from Annex A)*

There shall be at most one `global clocking` declaration anywhere in an entire elaborated SystemVerilog model. It shall be an error if there is more than one such `global clocking` declaration. It shall be an error to place a `global clocking` declaration within a program block.

The system function `$global_clock` returns the event expression specified in the unique `global clocking` declaration. The function has no arguments. It shall be an error to invoke the `$global_clock` system function if there is no `global clocking` declaration in the elaborated SystemVerilog model. Otherwise, `$global_clock` may be invoked anywhere that a clocking event may be specified.

The main purpose of global clocking is to specify which clocking event corresponds to the primary clock used in formal verification.

The following is an example of a `global clocking` declaration:

```
module top;
  logic clk1, clk2;
  global clocking sys @(clk1 or clk2); endclocking
  // ...
endmodule
```

In this example, `sys` is declared as the `global clocking` event and is defined to occur if, and only if, there is a change of either of two signals, `clk1` and `clk2`. Specification of the name `sys` in the `global clocking` declaration is optional since the `global clocking` event may be referenced by `$global_clock` from anywhere in the SystemVerilog model.

Any `clocking_event` may be specified in a `global clocking` declaration.

## 16.4 Concurrent assertions overview

REPLACE

The clock expression that controls evaluation of a sequence can be more complex than just a single signal name. Expressions such as `(clk && gating_signal)` and `(clk iff gating_signal)` can be used to represent a gated clock. Other more complex expressions are possible. However, in order to ensure proper behavior of the system and conform as closely as possible to truly cycle-based semantics, the signals in a clock expression must be glitch-free and should only transition once at any simulation time.

WITH

The clock expression that controls evaluation of a sequence can be more complex than just a single signal name. Expressions such as `(clk && gating_signal)` and `(clk iff gating_signal)` can be used to represent a gated clock. Other more complex expressions are possible. However, in order to ensure proper behavior of the system and conform as closely as possible to truly cycle-based semantics, the signals in a clock expression must be glitch-free and should only transition once at any simulation time.

In simulation, a reference to `$global_clock` (see [Note to editor: insert a reference to Global clocking subclause here](#)) is understood to be a reference to the `clocking_event` defined in the `global clocking` declaration. In formal verification `$global_clock` is considered to be the primary system clock (see F.3.1). Thus, in the following example

```
global clocking @clk; endclocking
...
assert property (@$global_clock a);
```

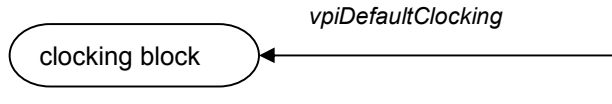
the assertion states that `a` is true at each tick of the global clock. In simulation, this assertion is equivalent to

```
assert property (@clk a);
```

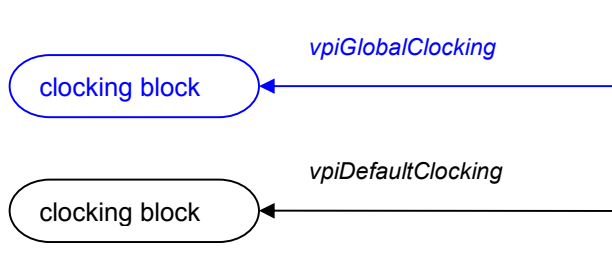
## 36.4 Module

[Note to editor: in the diagram](#)

REPLACE



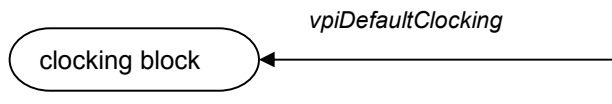
WITH



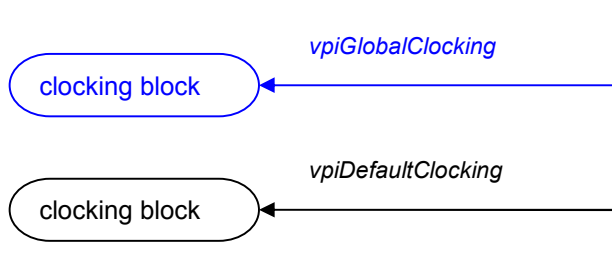
### 36.5 Interface

Note to editor: in the diagram

REPLACE



WITH



### A.6.11 Clocking block

REPLACE

clocking\_declaration ::=  
    [ **default** ] **clocking** [ clocking\_identifier ] clocking\_event ;  
        {clocking\_item}  
    **endclocking** [ : clocking\_identifier ]

WITH

clocking\_declaration ::=  
    [ **default** ] **clocking** [ clocking\_identifier ] clocking\_event ;  
        { clocking\_item }  
    **endclocking** [ : clocking\_identifier ]  
    | **global clocking** [ clocking\_identifier ] clocking\_event ; **endclocking** [ : clocking\_identifier ]

## Annex B

### Keywords

Note to editor: Insert a new keyword [global](#) into Table B1 (Reserved keywords).

#### F.3.1 Rewrite rules for clocks

REPLACE

The semantics of clocked sequences and properties is defined in terms of the semantics of unlocked sequences and properties. The following rewrite rules define the transformation of a clocked sequence or property into an unlocked version that is equivalent for the purposes of defining the satisfaction relation. In this transformation, it is required that the conditions in event controls not be dependent upon any local variables.

— @ ( c ) b  $\mapsto$  ( ! c [\*0:\$] ##1 c & b ) .  
— @ ( c ) ( 1, v = e )  $\mapsto$  ( @ ( c ) 1 ##0 ( 1, v = e ) ) .  
— @ ( c ) ( P )  $\mapsto$  ( @ ( c ) P ) .  
— @ ( c ) ( R1 ##1 R2 )  $\mapsto$  ( @ ( c ) R1 ##1 @ ( c ) R2 ) .  
— @ ( c ) ( R1 ##0 R2 )  $\mapsto$  ( @ ( c ) R1 ##0 @ ( c ) R2 ) .  
— @ ( c ) ( R1 **or** R2 )  $\mapsto$  ( @ ( c ) R1 **or** @ ( c ) R2 ) .  
— @ ( c ) ( R1 **intersect** R2 )  $\mapsto$  ( @ ( c ) R1 **intersect** @ ( c ) R2 ) .  
— @ ( c ) **first\_match** ( R )  $\mapsto$  **first\_match** ( @ ( c ) R ) .  
— @ ( c ) R [\*0]  $\mapsto$  ( @ ( c ) R ) [\*0] .  
— @ ( c ) R [\*1:\$]  $\mapsto$  ( @ ( c ) R ) [\*1:\$] .  
— @ ( c ) **disable iff** ( b )  $\mapsto$  P **disable iff** ( b ) @ ( c ) P .  
— @ ( c ) **not** P  $\mapsto$  **not** @ ( c ) P .  
— @ ( c ) ( R | -> P )  $\mapsto$  ( @ ( c ) R | -> @ ( c ) P ) .  
— @ ( c ) ( P1 **or** P2 )  $\mapsto$  ( @ ( c ) P1 **or** @ ( c ) P2 ) .  
— @ ( c ) ( P1 **and** P2 )  $\mapsto$  ( @ ( c ) P1 **and** @ ( c ) P2 ) .

WITH

The semantics of clocked sequences and properties is defined in terms of the semantics of unlocked sequences and properties. The following rewrite rules define the transformation of a clocked sequence or property into an unlocked version that is equivalent for the purposes of defining the satisfaction relation. In this transformation, it is required that the conditions in event controls not be dependent upon any local

variables.

```
—@($global_clock) b  $\mapsto$  b .
—@(c) b  $\mapsto$  ( !c [*0:$] ##1 c & b ) for c  $\neq$  $global_clock.
—@(c) ( 1, v = e )  $\mapsto$  ( @(c) 1 ##0 ( 1, v = e ) ) .
—@(c) ( P )  $\mapsto$  ( @(c) P ) .
—@(c) ( R1 ##1 R2 )  $\mapsto$  ( @(c) R1 ##1 @(c) R2 ) .
—@(c) ( R1 ##0 R2 )  $\mapsto$  ( @(c) R1 ##0 @(c) R2 ) .
—@(c) ( R1 or R2 )  $\mapsto$  ( @(c) R1 or @(c) R2 ) .
—@(c) ( R1 intersect R2 )  $\mapsto$  ( @(c) R1 intersect @(c) R2 ) .
—@(c) first_match ( R )  $\mapsto$  first_match ( @(c) R ) .
—@(c) R [*0]  $\mapsto$  ( @(c) R ) [*0] .
—@(c) R [*1:$]  $\mapsto$  ( @(c) R ) [*1:$] .
—@(c) disable iff ( b ) P  $\mapsto$  disable iff ( b ) @(c) P .
—@(c) not P not  $\mapsto$  @(c) P .
—@(c) ( R |-> P )  $\mapsto$  ( @(c) R |-> @(c) P ) .
—@(c) ( P1 or P2 )  $\mapsto$  ( @(c) P1 or @(c) P2 ) .
—@(c) ( P1 and P2 )  $\mapsto$  ( @(c) P1 and @(c) P2 ) .
```

The rewrite rule `@($global_clock) b  $\mapsto$  b` shall be applied only in formal verification. In simulation `$global_clock` shall be replaced by the *clocking\_event* defined in the `global_clocking` declaration, and the rewrite rule `@(c) b  $\mapsto$  ( !c [*0:$] ##1 c & b )` shall be applied for all clocks.

## M.2 Source code

REPLACE

```
#define vpiInputSkew 706
#define vpiOutputSkew 707
#define vpiDefaultClocking 709
```

WITH

```
#define vpiInputSkew 706
#define vpiOutputSkew 707
#define vpiGlobalClocking editor to fill
#define vpiDefaultClocking 709
```