Proposed Scheduling Semantics for SystemVerilog

Scheduling Semantics Working Group

February 2003

Requirements

- **100 % Backward Compatibility w/ Verilog-1364**
- **Testbenches must be portable between**
  - System-level, RTL, gate-level, and gate-level with timing levels
  - Delays may need to be back-annotated
- **Assertions must be capable of deterministic semantics between simulation, formal methods, and pre- and post-synthesis.**
- **Enable SystemVerilog to access all regions that PLI/VPI can access**
- **Non-Verilog models and models of computation supported**
  - C/C++, digital HDLs, mixed signal simulation, emulators, and acceleration
- **No adverse performance impact for models not using the extended features.**

Proposal Does not Attempt To

- **Add determinism in order to**
  - Eliminate design races
  - Eliminate testbench races
- **Modify existing Verilog semantics**
  - Legacy code is undisturbed
- **Favor a specific methodology or style**
  - Place undue restrictions on coding style

Can IEEE 1364-2001 do the job?

- **Problems**
  - Verilog zero-delay simulation races
    - Lack of predictability
  - Lack of consistency across design and verification tools
    - Different semantics between event-driven and cycle-accurate
- **Proposed solution**
  - Extend the scheduling semantics of the Verilog 2K1 standard
  - Apply partial ordering of design, testbench and assertion-based code using 3 new event scheduling regions

Models Synchronization Needs

- **Model often needs to know that its inputs are stable before calculating its response.**
  - **Need to synchronize to a stable, write-able slot (after design).**
- **Model often needs to sample previous values before new values appear.**
  - **Need to sample values just before the current time step.**

Clocked Assertions Synchronization Needs

- **Need to detect the clock event that triggers their evaluation.**
  - **Need to synchronize to the trigger event.**
- **Need the steady-state values of their state variables (inputs).**
  - **Need to sample values just before the current time step.**
  - **Need to sample values after the design is in steady state.**

Overall Testbench Needs

- Test programs may need to examine:
  - DUT (outputs and internal state)
  - Models states (RTL, behavioral, C, …)
  - Assertion status
- In order to
  - Assert correct behavior
  - Compute next stimulus
  - and Present it to DUT

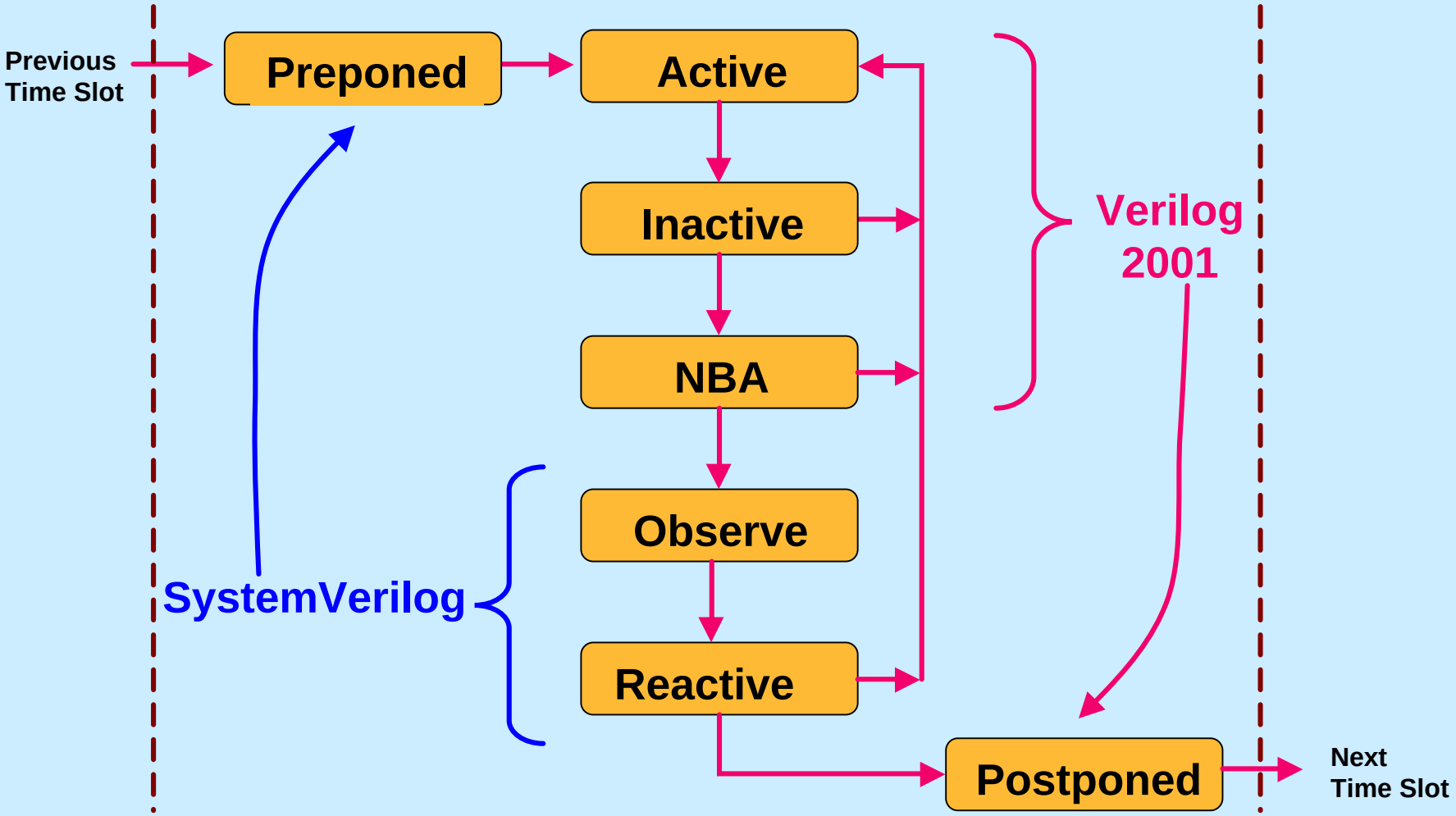
Additional Requirements

- **Assertions should not interfere with DUT**
  - Avoid **-** DUT/assertion races - Functional errors (false positives)
  - Pass/Fail should not change DUT state
- **Testbench may examine multiple clocks**
  - From various assertions, models, DUT, …
  - Needs deterministic sampling
- **Test should not introduce additional races**
  - De-couple test from DUT
- **A clocked assertion is evaluated only once per time step.**
  - Error for clock to trigger more than once per time step
- **Variables are sampled at most once per time step.**
- **Language constructs enable access as powerful as VPI**

Scheme Imposes a Partial Ordering

- **Sampling (previous steady state)**
- **DUT execution (always, =, #0, <=, …)**
  - Clock Detection (interleaved)
- **Property Evaluation**
  - Pass/Fail statements delayed
- **#0 Sampling (current steady state)**
- **Property Evaluation**
- **Testbench**
- **Monitors**

SystemVerilog Event Regions



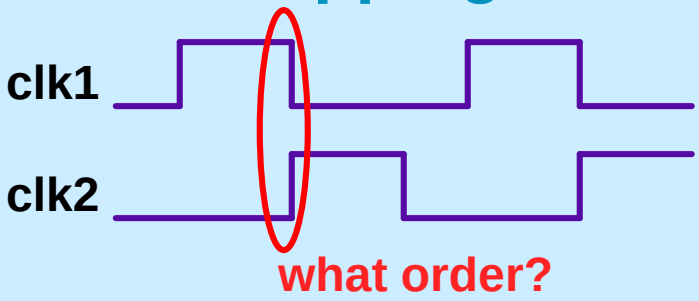
Scheduling Region Language Mapping

(To be confirmed by technical committees)

Region	VPI	SystemVerilog	Notes
Preponed	cbPrePoned	None	We should considered adding language access to this region
Active	cbStartOfTime, cbAfterDelay, cbNextSimTime	Initial, Always, Primitive, Continuous Assign, Assertions	Assertions can read sampled values to get cycle accurate results.
Inactive	None	#0 Delays	
NBA	?	NBA, Sampling Drives	
Observe	cbNBASync	None	
Reactive	?	Program Block, Pass/Fail Code	
Postponed	cbReadOnlySync		

Continuous invariant: Assertion on non-overlapping clocks

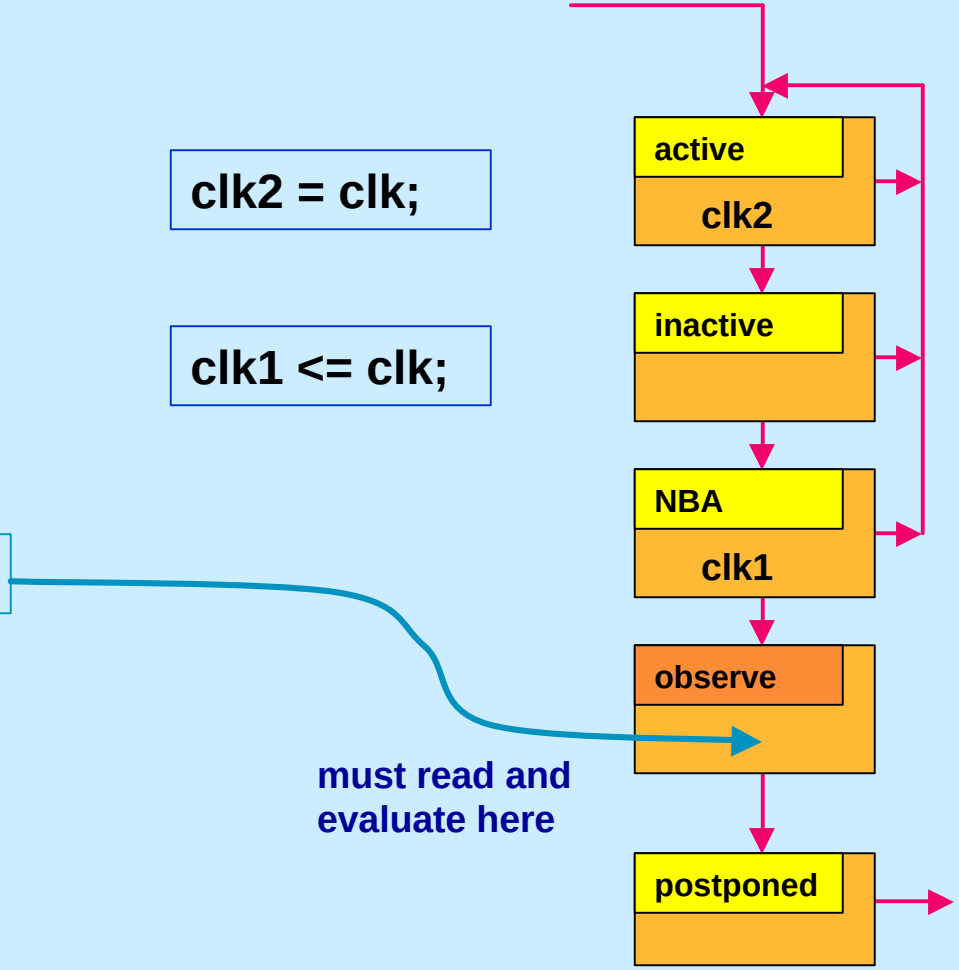
non-overlapping clocks

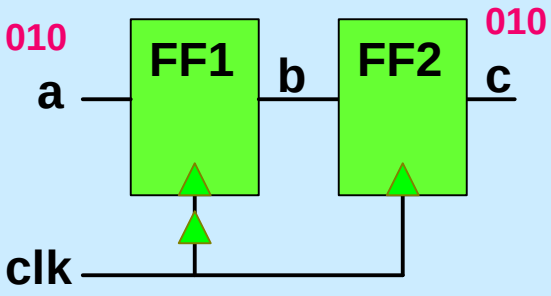


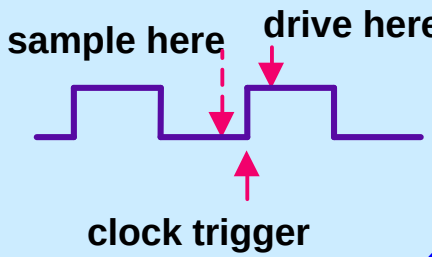

```
clk2 = clk;
```

```
clk1 <= clk;
```

```
assert ((clk1 && clk2) == 0);
```





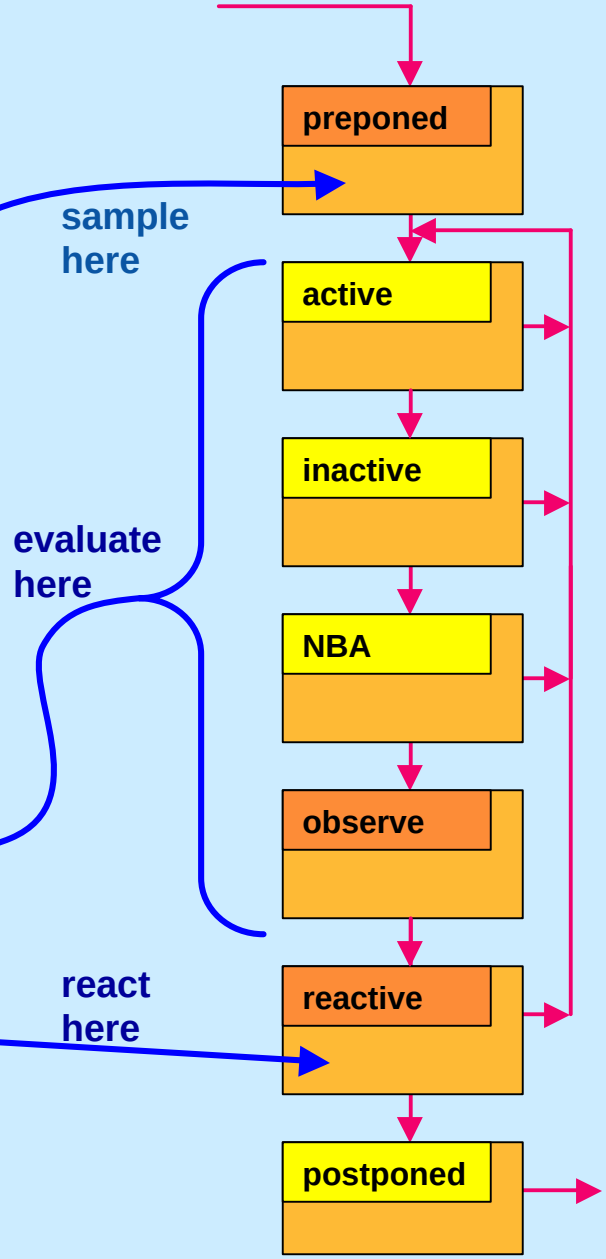



```
assign #0 gclk = clk;
always @(posedge gclk) b = a; // FF1
always @(posedge clk) c = b;   // FF2
```

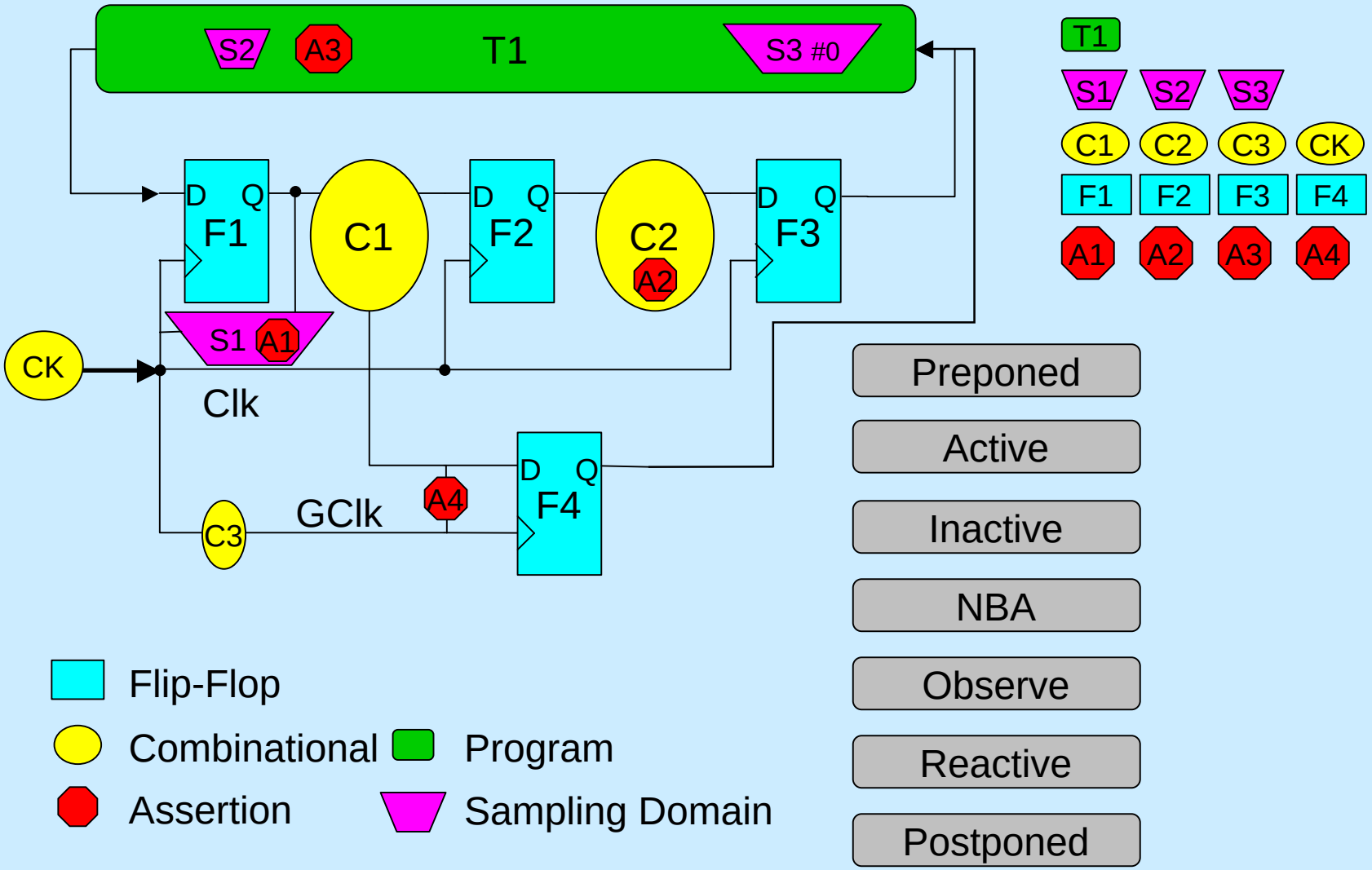
```
sequence @(posedge clk) sa = (!a ; a ; !a);
sequence @(posedge clk) sc = (!c ; c ; !c);
```

```
property p = (sa => [2] sc); // clocked assertion
```

```
assert (p)  pass_statement;
       else; fail_statement;
```



Simulation Cycle Animation



Conclusions

- The new scheduling semantics enables design, testbench, and assertion-based code in one language
- A new event scheduling algorithm has been proposed that enables design and verification code to consistently work together without the need to resort to PLI synchronization
- Backward Compatible with Verilog 1364-2001
- Consistent semantics and results across design and verification tools, including simulation, synthesis, and formal verification