System Verilog v3.1
Overview, Semantics and Scheduling

Peter Flake, Arturo Salz, Surrendra Dudani, Phil Moorby
Synopsys, Inc.

Agenda

- Motivation
- New Constructs
- Lexical Containment
- Program block
- Clocking domains
- Sequences and properties
- Expects
- Coverage
- IP packaging
- Scheduling semantics

Motivation

- Combine HDL and HVL and property language into a single language
- Synthesis, simulation and formal property checking driven by a single language
- Design and verification IP in a single delivery and usage mechanism
- Single language means common syntax, semantics and API for common operations

Testbench and Property Needs

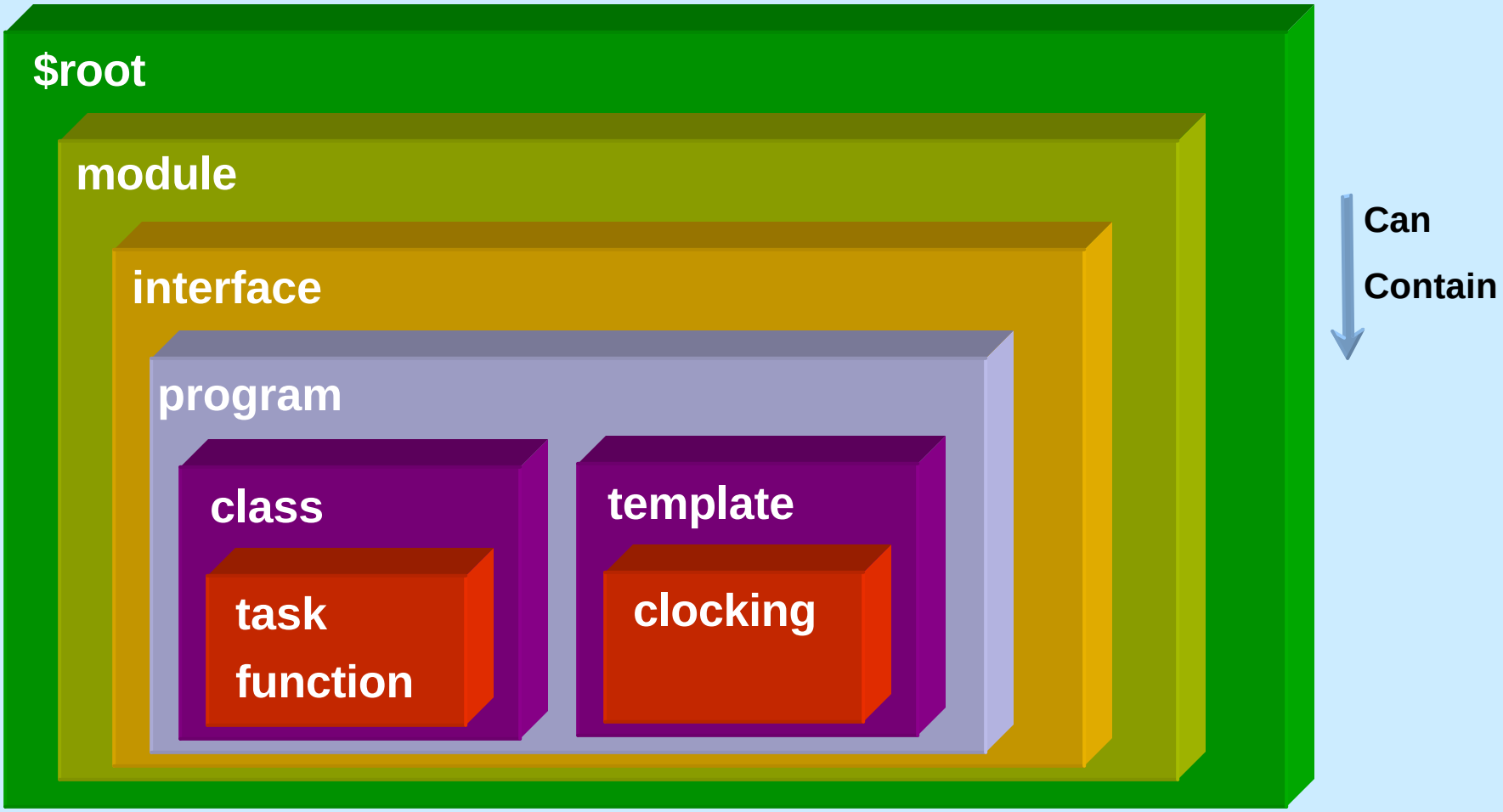
- Synchronous (cycle) semantics
  - Clock specification
  - Sampling of signals
- Named sequences and properties
  - Libraries of properties
  - Properties for assertions and coverage
  - Properties for environmental assumptions and constraints
- Assertions and coverage in either testbench or design
- Coverage feedback to testbench

New Constructs in SV 3.1

- Program
- Clocking domain
- Class
- Template

- Property
- Sequence
- Boolean expression
- Clocked assign
- Constraint
- Dynamic memory
- Event variable
- Semaphore
- Mailbox
- Expect
- Cycle delay

Lexical Containment



Program Block

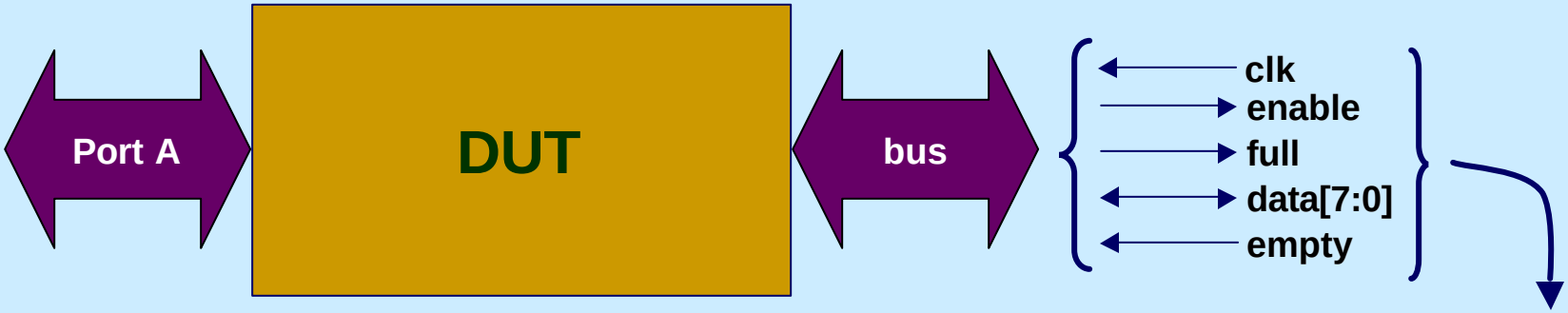
- Purpose: distinguish verification code
- **program** is similar to a **module**
  - Only one implicit initial block
  - Special semantics
    - Execute in verification phase

```
program name ( port_list );
    declarations (class, type, function, clocking...)
    statements
endprogram
```

Clocking Domain

- A clocking domain defines a synchronous interface for testbench and properties
- Every clocking domain has only one clock event
- Sample and drive timing specified with respect to clock
- A signal may appear in multiple clocking domains
  - Input - multiple samples
  - Output – default bus resolution
- Clocking domain creates a scope

Connecting to a Synchronous Interface



Synchronous Interface

```
clocking bus @(posedge clk);
  default input #1ns output #2ns;

  input          enable, full;
  inout          data;
  output         empty;
  output  #6ns   reset = top.u1.reset;
endclocking
```

Clocking Event "clock"

Default I/O Skew

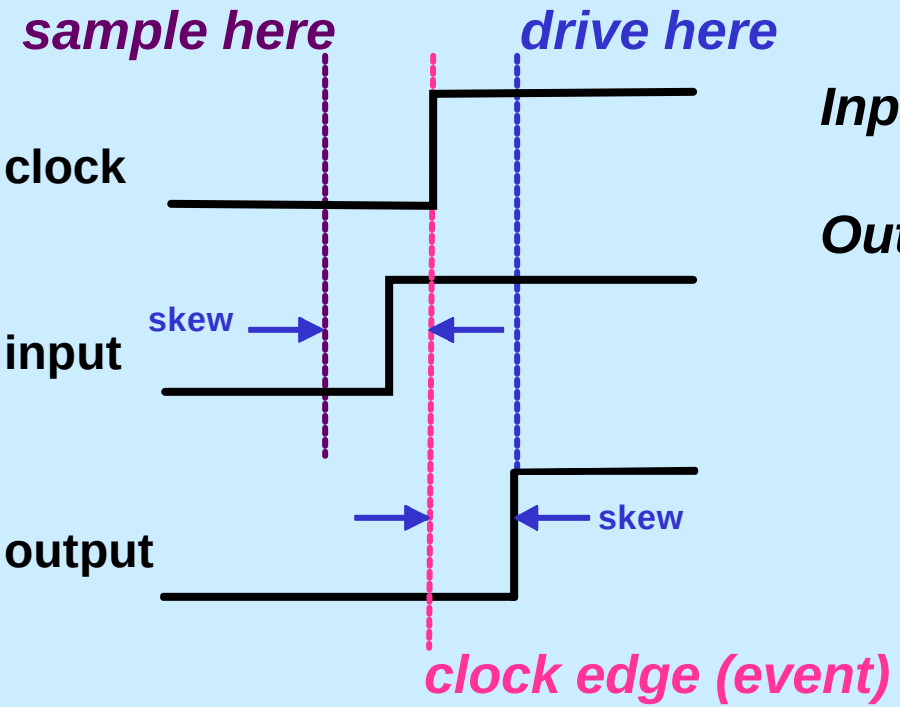
Hierarchical signal

Testbench Uses:

```
bus.enable
bus.data
  ...
```

Skew Declaration

- Specify Synchronous Sample / Drive Times



Input skew is for sampling

Output skew is for driving

Built-in default skews:

input #1step
output #0

Default Clocking

- Designate one clocking as default

```
default clocking interface1.bus;
```

- One default per module, interface, or program
- Cycle Delay Syntax: *## integer_expression*

```
## 5;              // wait 5 cycles
## j + 4;          // wait j+4 cycles
```

Sequences and Properties

- Declarations of named sequences
- Declaration of named properties
  - Using sequences in properties

```
module flashMem (...);

 ...
 seq @clk mem_read = (mem_req ;[1] mem_done);
 property rd_p = (if (req == read) mem_read);

 ...
endmodule
```

Default Clocking and Properties

- Properties with no clock use the default clocking

```
module example1 (…)
   reg  p,q,r;
   default clocking interface1.bus ;
   property p1 = (p; q ; r);
   property p2 = (p;[1:5] r == 0);
   property p3 = (if (r) ([3:6] q));
endmodule
```

Clocking and Properties

- Clocking can include SystemVerilog Properties

```
module example2 (clk,…)
  reg  a,b,c;
  clocking bus1 @(posedge clk);
     property p1 = (a; b;[1:3] c);
     property p2 = (if (b) (c;[2] !a));
  endclocking
   ...
  property p3 = (bus1.a;[2:5] bus1.c);
endmodule
```

a b c : Implicit input Declaration

Expects

- Enables the testbench to react to sequences
  - Useful for transactors
- Blocking statement
  - Blocks until sequence succeeds or fails
- Issues error when it fails

Expect Example

```
program test (…)
  clocking p @(negedge clk1 );
    input  a, b, c;
  endclocking

  ...  // drive stimulus
  expect (p.a ; [1:5]p.b; rose p.c);
  ...  // check response
endprogram
```

Coverage Specification

- Supports white box coverage for design internals
- Supports black box coverage for testbench
- Identical syntax and semantics
- Allows use of design and testbench variables in coverage specification
- Provides features to customize coverage analysis

Coverage Example

Basic coverage - report using label

```
trans: cover (req ;[4:7] ack ;[1] grant);
```

Customized coverage

```
trans: cover (req ;[4:7] ack ;[1] grant)
          do begin
             trans_count++;
             store(req_fifo_state);
          end
```

Coverage Feedback

Design code

```
event read_mem_e;
read_c: cover ((req==read);[1:4] mem_read)
   do begin
      $display("read memory transaction");
      -> read_mem_e;
   end
```

Testbench code

```
int read_count;
@(read_mem_e) gen_rand_req(read_count++);
```

IP Packaging

- Support IP to include testbench and assertions
- Create independent module or interface with testbench and assertions to allow
  - VIP creation independent of implementation
  - Mixed language simulation
  - Formal and simulation verification
- Construct parameterizable VIP

Verification IP Packaging Example

interface busP

ASSERTIONS

```
seq read_t = req ;[1:4] grant;
 t1: assert read_t;[1] !rd_r *[8];
 c1: cover (req;[4] ack;[1] grant);
 …
```

TESTBENCH

```
program test (…)
 …
 default clocking clk;
 …
 gen_trans(par1);
 expect ([4:7] ack;[1] grant);
 …
endprogram
```

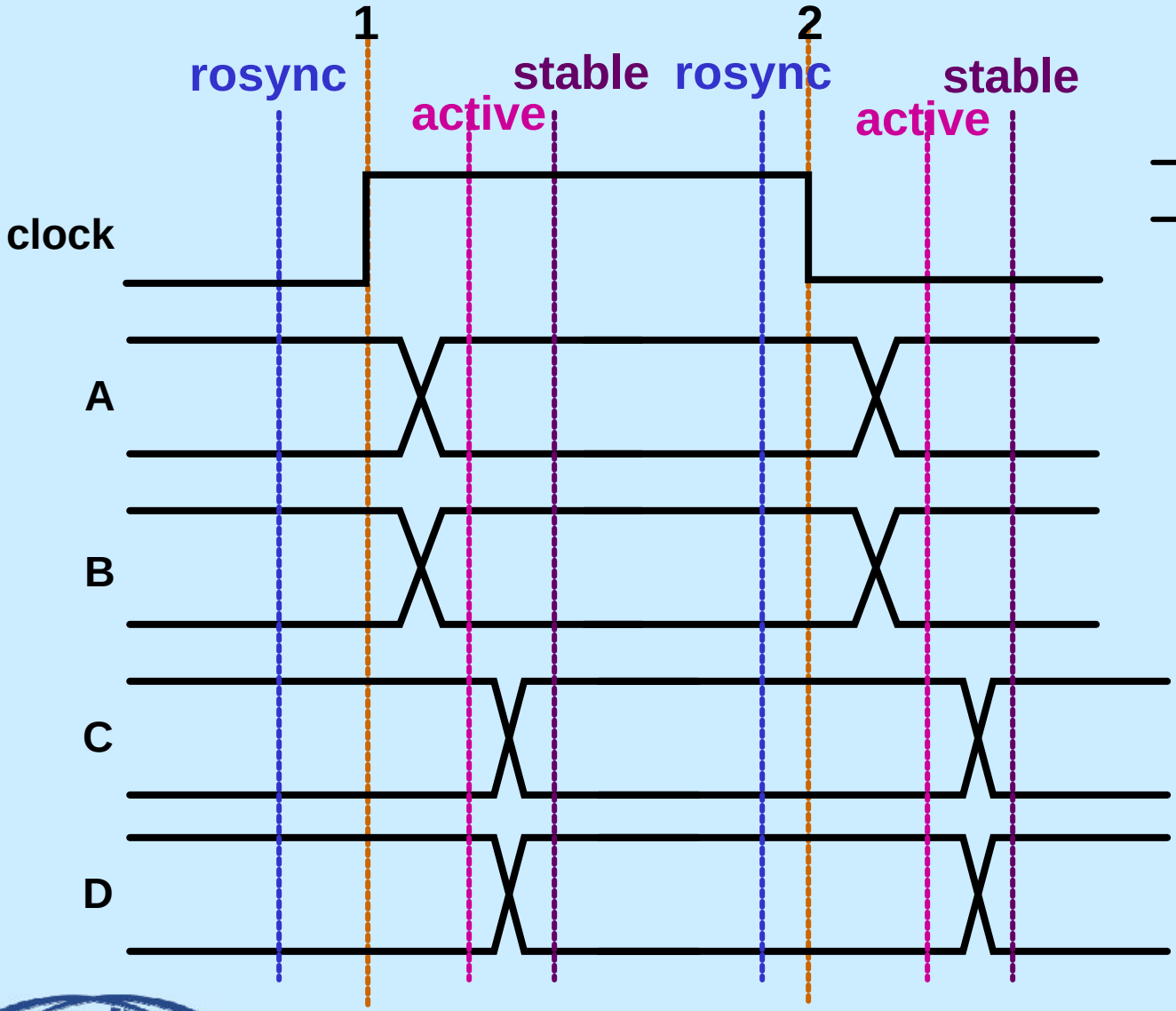
Instantiate protocol as master and slave

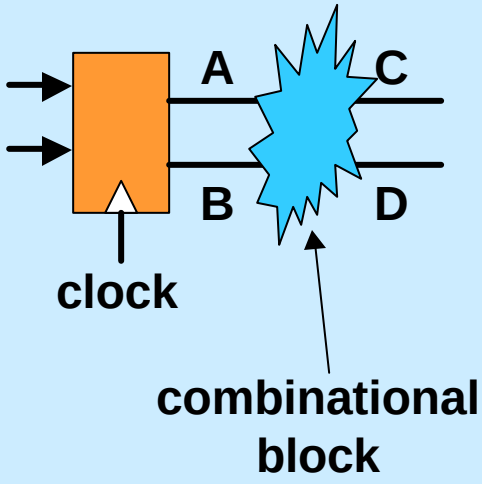
```
busP bus_master(…,clock);
busP bus_slave(…,clock);
```

Sampling for Verification

- Races between design and testbench are a common source of bugs
- Properties must only be checked on steady state

Steady State - When?



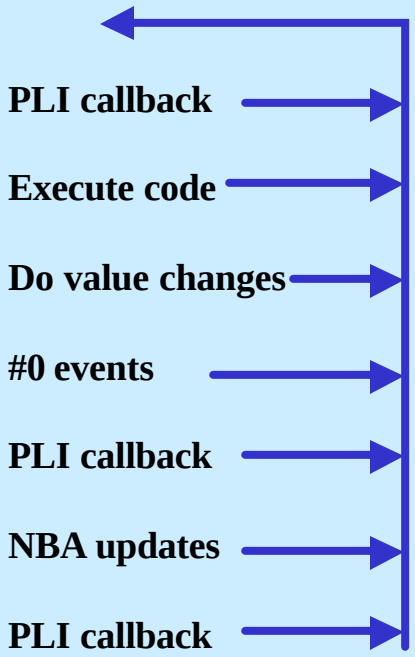


rosync == stable

Scheduling semantics

- Goal: Mix design and verification code and avoid common race situations between the two
- Time slot divided into 4 ordered phases:
  - Design, clocking, verification and read-only
  - There is no going back to a previous phase within the same time slot

4 phases of a time slot

Design	Clocking	Verification	Read-only
	PLI callback(1) Detect clocks Sample design variables Evaluate next state of properties	Mirror of design phase Detect design errors Do coverage actions	PLI callback(2). Strobes Monitors Special dumping

(1) PLI code is not allowed to create design phase events at current time

(2) PLI code is not allowed to create any events in current time

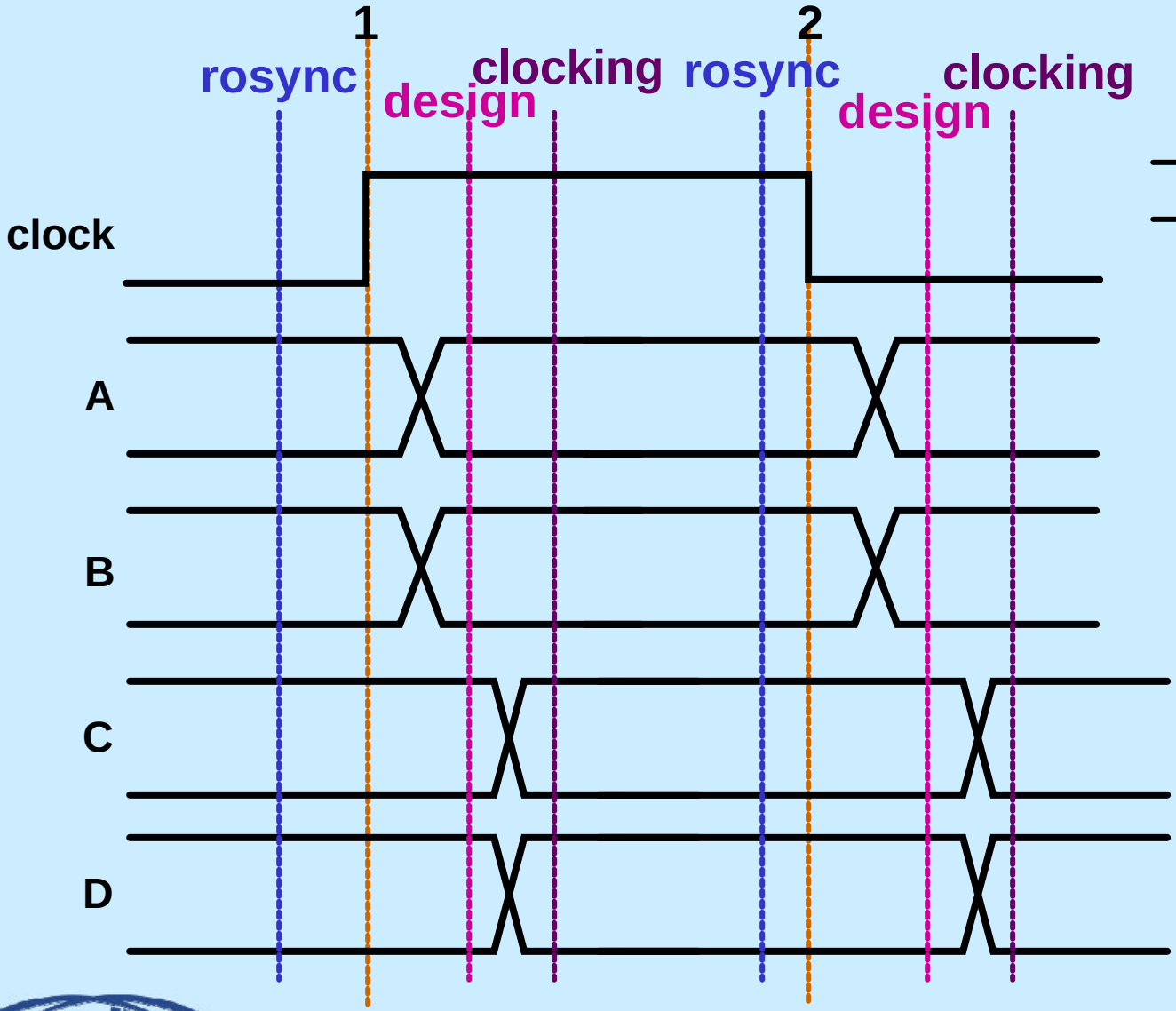
Clocking phase

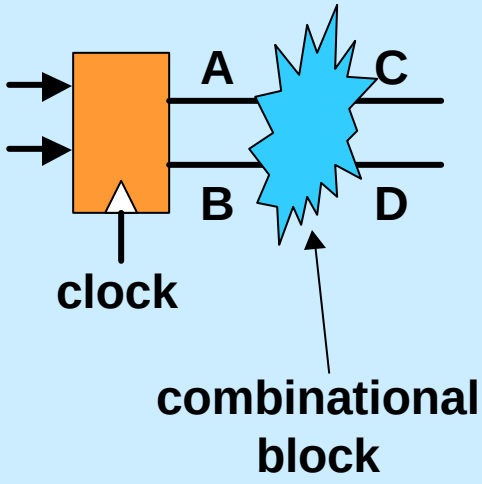
- Happens if some clock triggered in some clocking domain or sampling clock
- Sample design variables, evaluate properties and sequences
- For system processing only - no System Verilog code executed
- No loop back mechanism within phase

Verification phase

- Execute code within program blocks, coverage and assertion actions.
- Similar loop back semantics to the design phase for delayed, blocking and non-blocking statements
- Mirror of design phase
- Not allowed to trigger clocks at current time slot
- More useful than VHDL “postponed processes”

Steady State - When?





rosync == clocking

Summary

- Enhanced SystemVerilog 3.0 with Testbench and Property features
- Clarified semantics between design and testbench
- Minimized overall language complexity
  - keywords, syntax rules, semantics
- Maximized commonality of syntax and semantics in the spirit of Verilog