ASIC Verification

Interfaces

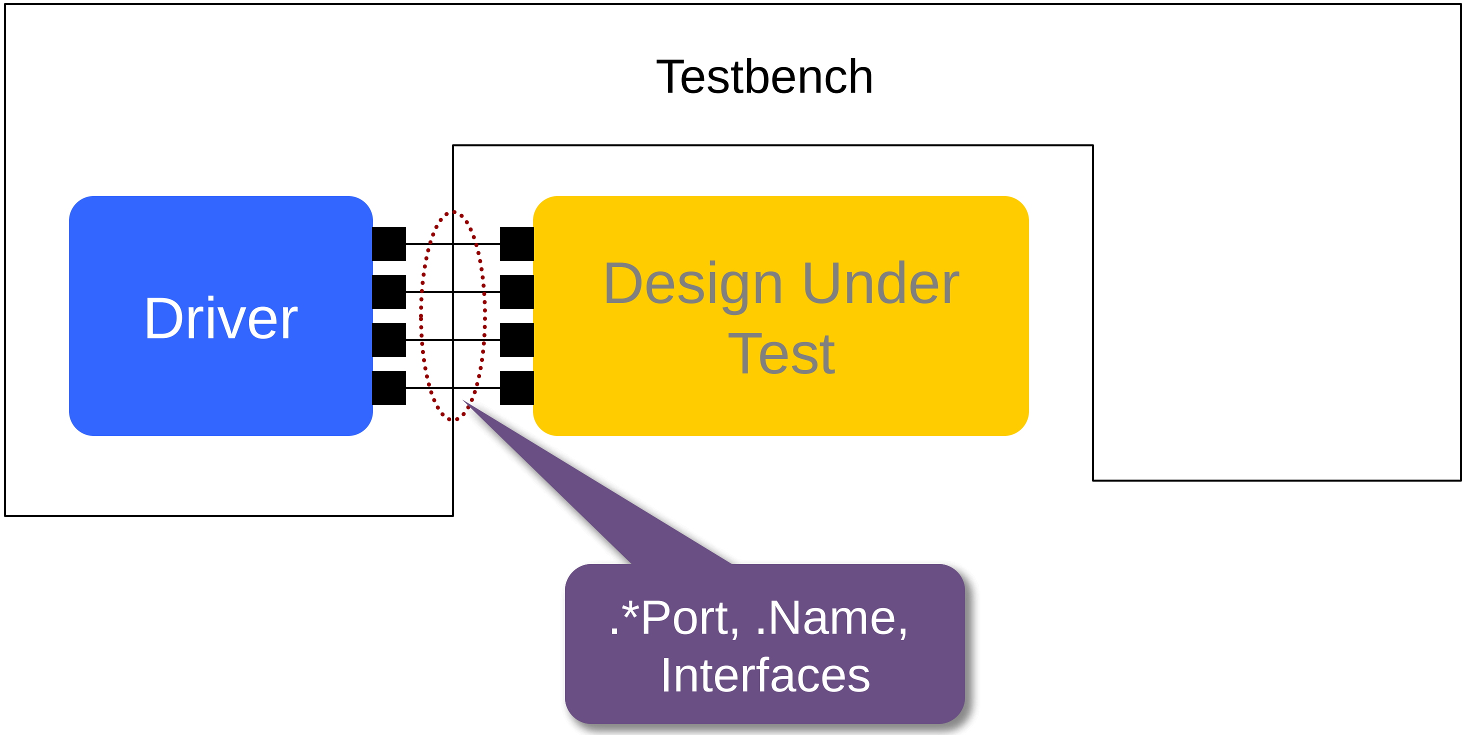
Fall 2011
Meeta Yadav

Outline

1. Connecting the testbench and the design
2. Verilog connection review
3. SystemVerilog Interfaces
4. Stimulus timing
5. Clocking Blocks
6. Timing regions
7. Program block

Connecting the Testbench and the Design

How do I connect my design to the testbench?

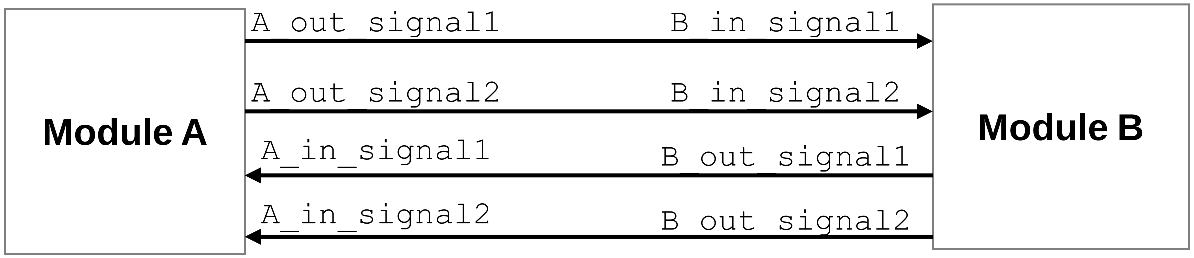


©2011, Dr. Meeta Yadav,

Connecting the Testbench and the Design

- One way to connect the testbench and the design is to use the conventional verilog module ports convention

Verilog Connection Review

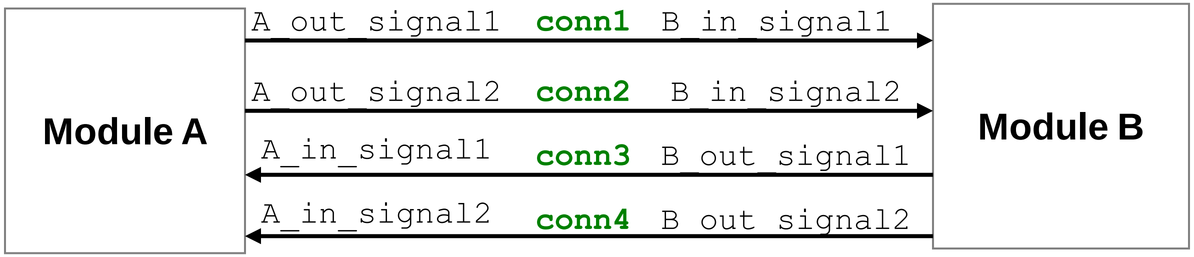


```
module A (
input A_in_signal1,
input A_in_signal2
output A_out_signal1,
output A_out_signal2,
);
…
endmodule
```

```
module B (
input B_in_signal1,
input B_in_signal2
output B_out_signal1,
output B_out_signal2,
);
…
endmodule
```

Connecting the Testbench and the Design

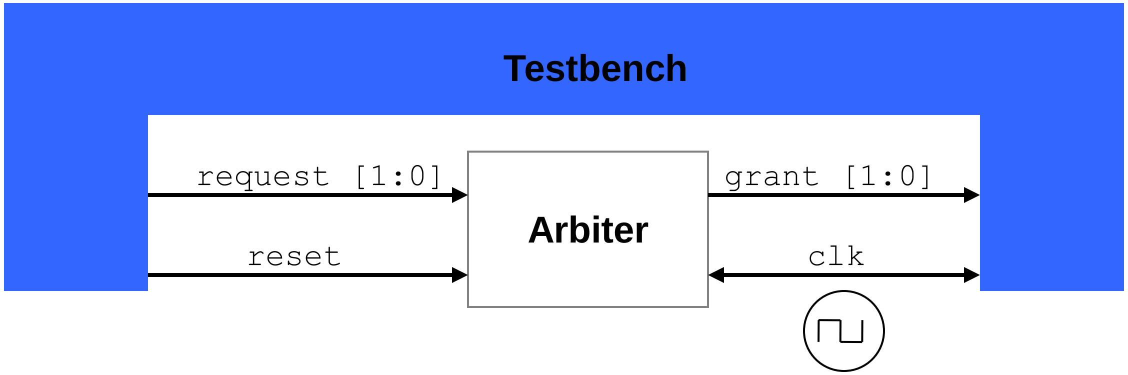
- Verilog connection review
  - Verilog language connects modules together through module ports



```
module top (
wire conn1,
wire conn2
wire conn3,
wire conn4,
);
A A_instance1(.A_out_signal1(conn1),
              .A_out_signal2(conn2),
              .A_in_signal1(conn3),
              .A_in_signal2(conn4));
B B_instance1(.B_in_signal1(conn1),
              .B_in_signal2(conn2),
              .B_out_signal1(conn3),
              .B_out_signal2(conn4));
endmodule
```

Connecting the Testbench and the Design

Verilog Connection Review:



```
module arb_port (
output logic [1:0] grant,
input logic [1:0] request,
input logic reset,
input logic clk
);
…
endmodule
```

```
module test (
input logic [1:0] grant,
output logic [1:0] request,
output logic reset,
input logic clk
);
…
endmodule
```

Connecting the Testbench and the Design

SystemVerilog Connections (Same as Verilog):

Verilog Module Ports

```
module top
logic [1:0] grant,request;
logic clk, reset;
arb_port a1(.grant(grant), .request(request), .reset(reset), .clk(clk));
test t1(.grant(grant), .request(request), .reset(reset), .clk(clk));
…
endmodule
```

SystemVerilog: .*Port Connections

- **Implicit .* port connections**
  - .* infers connections of all nets and ports of the same name
  - For a connection to be inferred the name and the vector sizes should be the same
  - Types connected together should be compatible

```
module top
logic [1:0]
grant,request;
logic clk, reset;
arb_port a1(.*);
test t1(.*);
…
endmodule
```

.*Port Connections

SystemVerilog: .Name Connections

- Implicit .name connections
  - .name is an abbreviation of named port connections
  - .name infers a connection of a net and port of the same name and same vector sizes
  - .name simplifies connections to module instances
  - .name can be combined with named port connections

```
module top
logic [1:0] grant,request;
logic clk, reset;
arb_port a1(.grant, .request, .reset, .clk);
test t1(.grant, .request, .reset, .clk);
…
endmodule
```

Downside of Verilog Connection Conventions

Verilog module port conventions are cumbersome

Why?

Lets change the name of a port request to request 1

```
module arb_port (
output logic [1:0] grant,
input logic [1:0] request1,
input logic reset,
input logic clk
);
…
endmodule
```

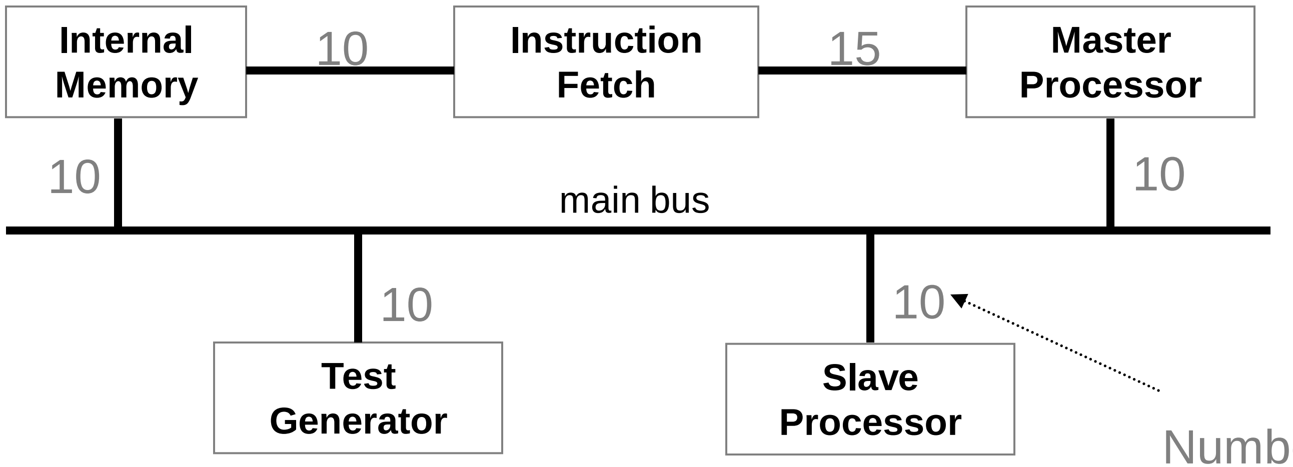
```
module test (
input logic [1:0] grant,
output logic [1:0] request1,
output logic reset,
input logic clk
);
…
endmodule
```

- Need to change the port list of each module
- Need to change the port list of the connecting module
- Need to change the name in the instantiation of the modules
- Need to change the name everywhere in the hierarchy

What if you forget to change it in someplace?? --> Compilation error!!!!

Downside of Verilog Connection Conventions

Verilog connections become especially tedious and cumbersome for large designs



Number of ports

©2011, Dr. Meeta Yadav,

Disadvantages of Verilog Connection Conventions

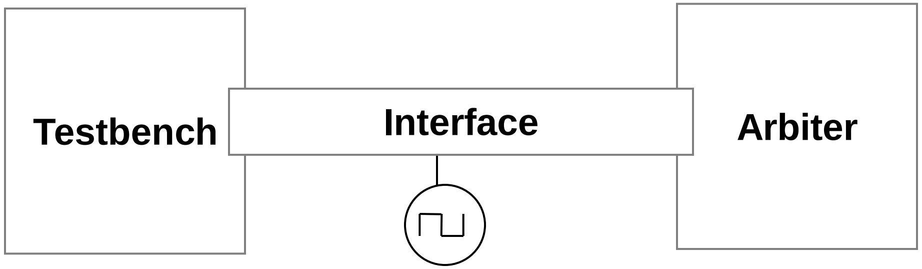
- Disadvantage of Verilog Module Connections
  - ◆ Declarations must be duplicated in multiple modules
  - ◆ Communication protocols must be duplicated in several modules
  - ◆ Risk of mismatched declarations
  - ◆ A change in design specifications can require modifications in multiple modules

Solution!!!!

SystemVerilog introduces a powerful new port type called: Interface

SystemVerilog Interfaces

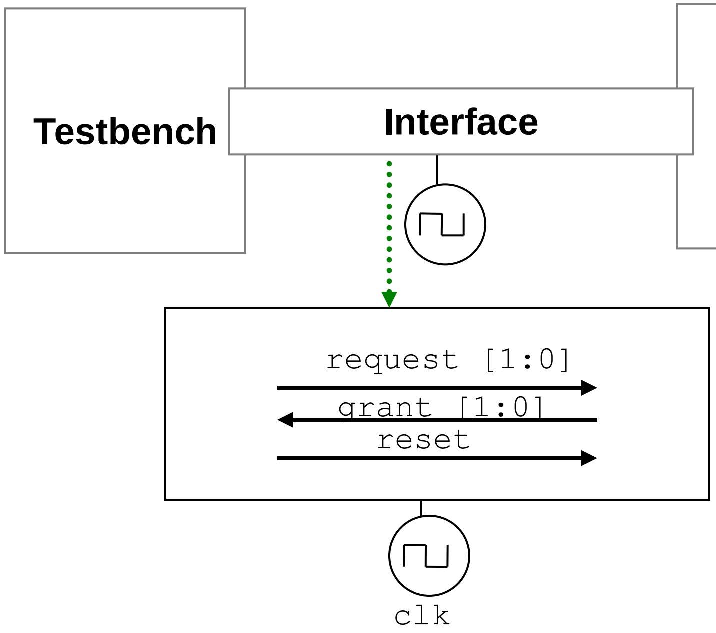
- **SystemVerilog Interfaces**
  - SystemVerilog adds a powerful new port type to Verilog, called an **`interface`**.
  - An **`interface`** allows a number of signals to be grouped together and represented as a single port
  - The declarations of the signals that make up the **`interface`** are contained in a single location.
  - Each module that uses these signals then has a single port of the **`interface`** type, instead of many ports with the discrete signals.



All the signals that are common between the major blocks of the design are encapsulated in one location- the interface declaration

SystemVerilog Interfaces

Using an interface to simplify connections



Interface

```
interface arb_if (input bit clk);
logic [1:0] grant, request;
logic reset;
endinterface
```

Interface Declaration

SystemVerilog Interfaces

- Using an interface to simplify connections

```
interface arb_if (input bit clk);
logic [1:0] grant, request;
logic reset;
endinterface
```

Interface Declaration

```
module test (arb_if arbif);
…
  initial begin
     @(posedge arbif.clk);
     arbif.request<=2’b01;
     $display (“@%0d: Drove req=01”, $time);
     repeat(2) @(posedge arbif.clk);
     if(arbif.grant!=2’b01)
       $display (“@%0d: a1: grant !=2’b01”, $time);
     $finish
  end
endmodule: test
```

`test` module using a simple arbiter interface

SystemVerilog Interfaces

- Using an interface to simplify connections

```
interface arb_if (input bit clk);
logic [1:0] grant, request;
logic reset;
endinterface
```

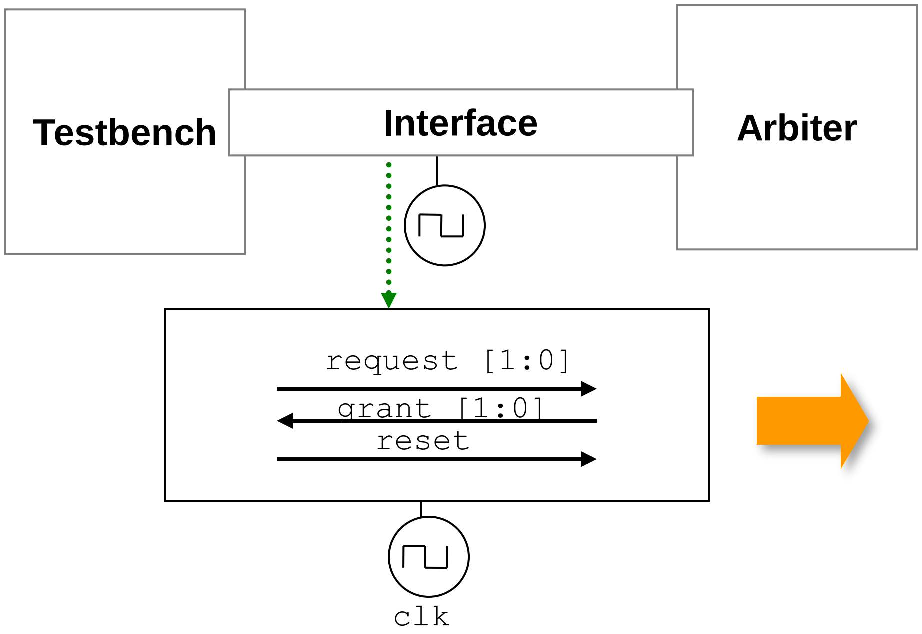
Interface Declaration

```
module arb(arb_if arbif);
…
endmodule:
```

arb module using a simple arbiter interface

SystemVerilog Interfaces

Using an interface to simplify connections



Interface

```
interface arb_if (input bit clk);
logic [1:0] grant, request;
logic reset;
endinterface
```

Interface Declaration

```
module top;
bit clk;
always #5 clk=~clk;
arb_if arbif(clk);
arb a1(arbif);
test t1(arbif);
endmodule: top
```

Top module using a simple arbiter interface

SystemVerilog Interfaces

- Connecting interfaces and ports
  - >Signals in an interface are referenced using the port name

```
<port_name>.<internal_interface_signal_name>
```

```
interface arb_if (input bit clk);
logic [1:0] grant, request;
logic reset;
endinterface
```

Interface Declaration

```
module top;
bit clk;
always #5 clk=~clk;
arb_if arbif(clk);
arb a1(.grant(arbif.grant),
       .request(arbif.request),
       .reset(arbif.reset),
       .clk(arbif.clk));
test t1(arbif);
endmodule: top
```

Connecting the arbiter module using ports to the test module using an interface

SystemVerilog Interfaces: `modports`

- Interface `modports`
  - SystemVerilog interfaces provide a means to define different views of the interface signal
  - `modport` is an abbreviation for module port
  - An interface can have any number of modport definitions
  - The modport declaration only defines whether the connecting module sees a signal as an input, output or bidirectional

```
interface arb_if (input bit clk);
logic [1:0] grant, request;
logic reset;
modport TEST(output request,reset,
             input grant, clk);
modport DUT(input request,reset, clk
             output grant);
endinterface
```

1. Interface Declaration using `modports`

```
module arb (abf_if.DUT arbif);
…
endmodule
```

2. arbiter module with interface using modports

```
module test (abf_if.TEST arbif);
…
endmodule
```

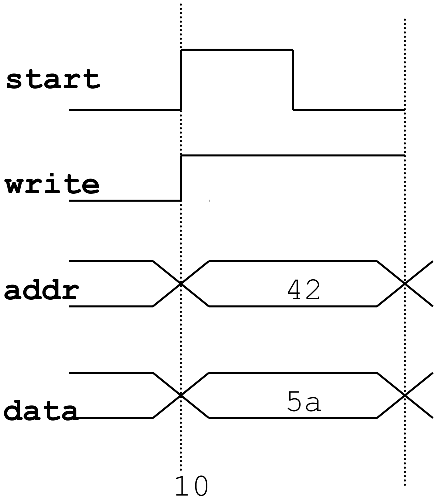
3. test module with interface using modports

SystemVerilog Interfaces

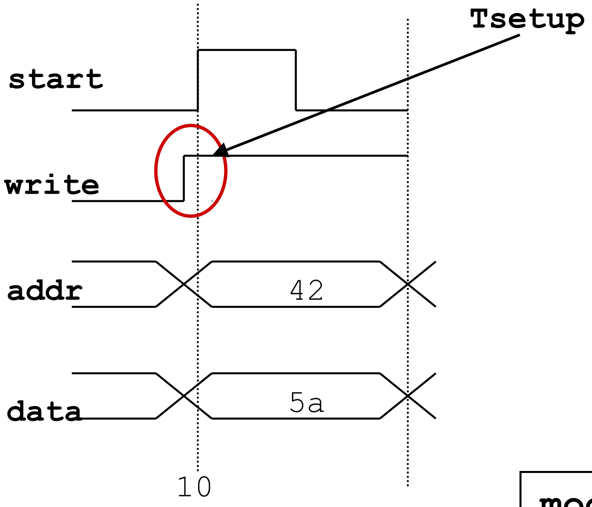
- SystemVerilog interfaces overview
  - SystemVerilog interfaces allow bundling of signals
  - SystemVerilog interfaces cannot contain design hierarchy
    - cannot contain instances of modules
  - SystemVerilog interfaces can be used as a module port
    - `modports` allow modules to see interface differently

Stimulus Timing

The timing between the testbench and the design should be maintained to avoid race conditions



Driving the signal too late and sampling it too early causes the race condition to occur



Solution: Drive the signal Tsetup early to avoid race condition

```
module test(…)
initial begin
start=0;mwrite=0;
#10
start=1; write=1;
addr=8’h42;data=8’h5a;
…
end
endmodule
```

Testbench

```
module memory(…)
always @(posedge start) begin
if (write)
mem [addr]=data;
…
end
endmodule
```

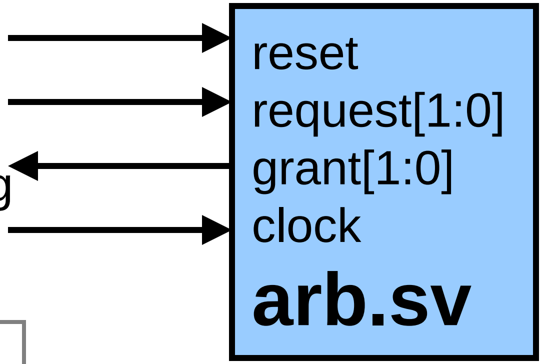
Design

Stimulus Timing: Clocking Blocks

- Clocking Blocks Overview
  - ◆ Use in the interface, just for testbench
  - ◆ Benefits:
    - ► Creates explicit synchronous timing domains
    - ► Provides race-free operation
    - ► Your testbench will always drive the signals at the right time!
  - ◆ Functionality:
    - ► An interface can contain multiple clocking blocks
    - ► There is one clock per clocking block.
    - ► Default is "`default input #1step output #0;`"
      - – The 1 step delay specifies that signals be sampled in the postpone region before any design activity
      - – Directions are with-respect-to the testbench

Stimulus Timing: Clocking Blocks

- Clocking Block
  - Can be declared as `@(posedge clk)`
  - An interface can use a clocking block to control timing
    - Directions are relative to program block



```
interface arb_if (input bit clk);
logic [1:0] grant, request;
logic reset;
clocking cb @(posedge clk);
output request
input grant
endclocking
modport TEST(clocking cb,output reset);
modport DUT(input request, reset, output grant);
endinterface
```

Example of clocking block

Declare the clocking block and indicate that the signals will be active on positive edge of the clock

Using the clocking block, (request is output and grant is input)

- **Referencing signals in the Clocking Block**

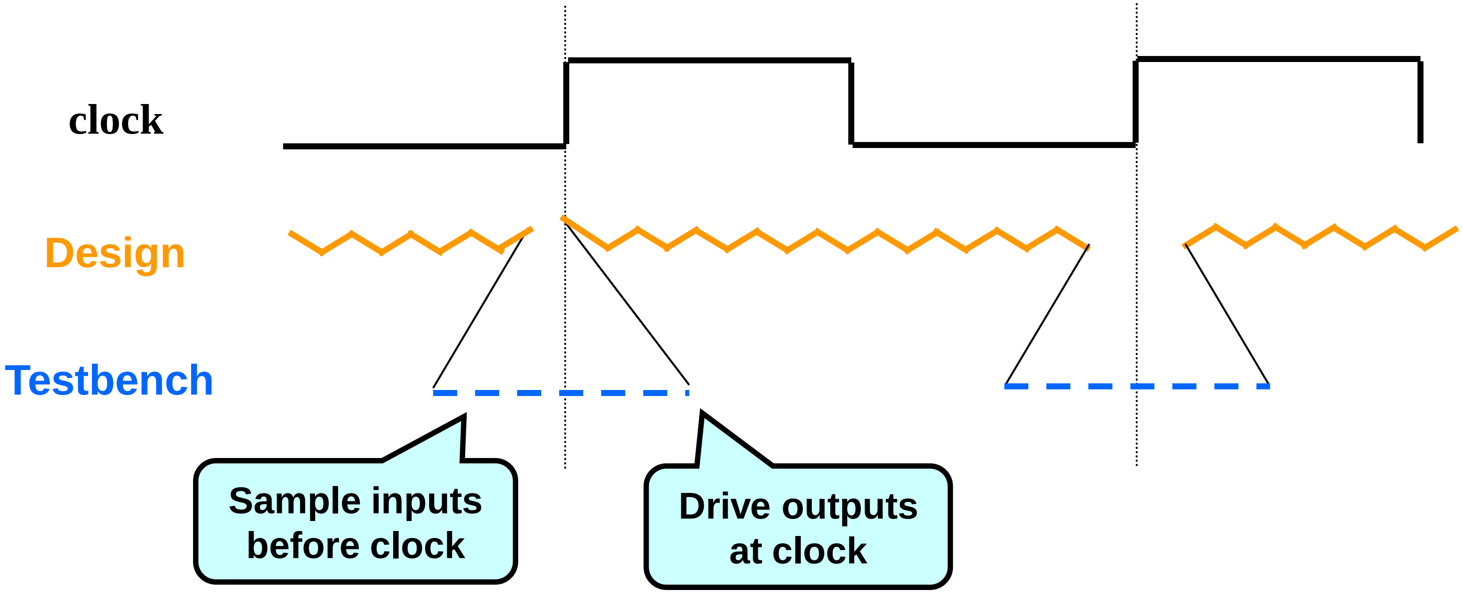
```
<my_interface.cb.signal_name>
```

```
arb_if arbif;
arbif.cb.request<=2’b01;
if(arbif.cb.grant!=2’b01)
@arbif.cb
```

Example of refrencing signals in the clocking block

System Verilog Testbench in Simulation

```
default input #1step output #0;
```



Clocking Block: Signal Synchronization

- **Synchronize to active clock edge specified in clocking block**

```
@arbif.cb;            // continue on clock edge from arb_if
repeat (3) @arbif.cb; // Wait for 3 posedges
```

- **Synchronize to any edge of signal**

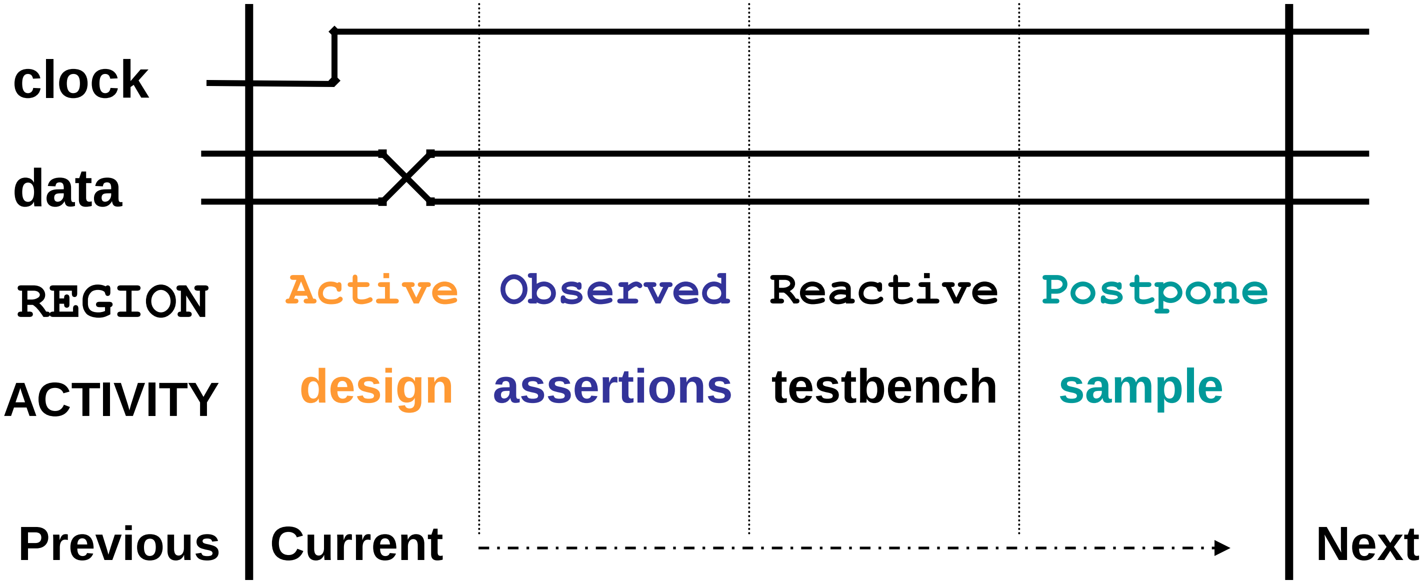
```
@arbif.cb.grant;           // continue on any edge of grant
@(posedge arbif.cb.grant); // continue on posedge
@(negedge arbif.cb.grant); // continue on negedge
wait (arbif.cb.grant==1);  // wait for expression
                           // no delay if already true
```

- **Wait for N clock cycles with ##n - blocking**

```
##2 arbif.cb.request <= 0;    // Wait 2 cycles then assign
```

Stimulus Timing: Timing Regions

- **Timing Regions**
  - Race conditions are caused by mixing design and testbench events during the same time slot
  - SystemVerilog introduces division of time slots
    - **Active Region:** Simulation of design code in modules
    - **Observed Region:** Assertions evaluated after design executes
    - **Reactive Region:** Execution of testbench
    - **Postpone Region:** Sampling signals after all design activity

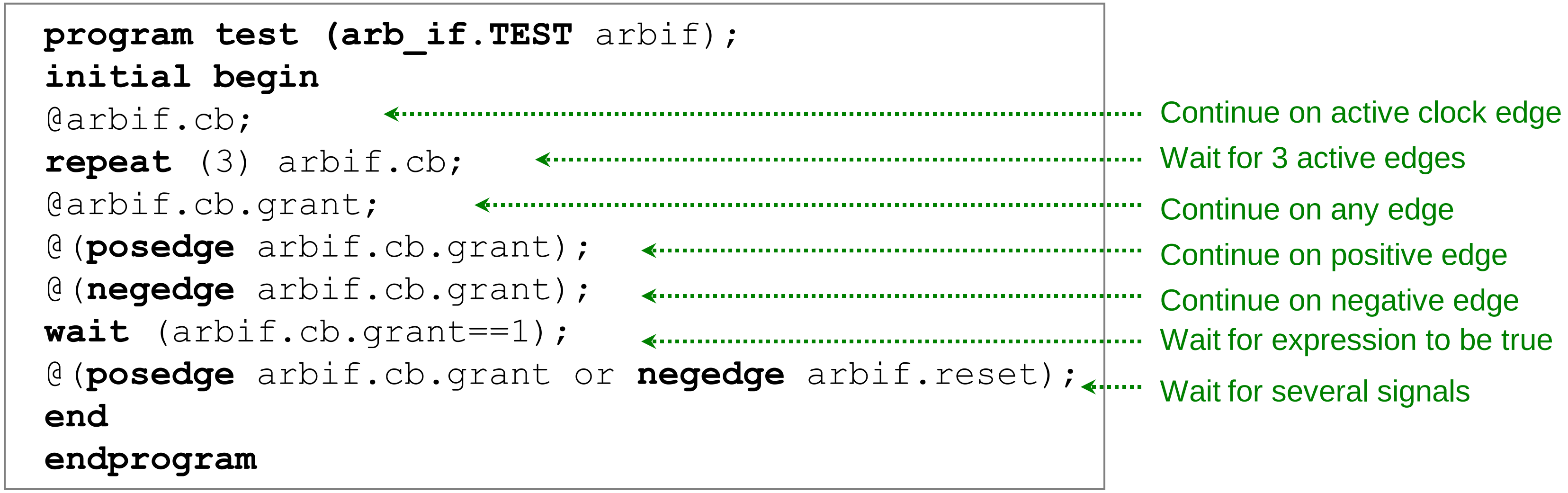


Program Block Overview

- **Benefits:**
  - Separates the testbench from the DUT
  - Reduces race conditions by running in separate region
  - Provides an entry point for execution
  - Creates a scope to encapsulate program-wide data
- **Functionality:**
  - Can be instantiated in any hierarchical location
    - Typically at the top level
  - Interfaces and ports can be connected in the same manner as any other module
  - Code goes in initial blocks & routines, no always blocks
  - Executes in the Reactive region
  - Implicit $finish when all initial blocks end in program

Program Block

- **Program Block**
  - In Systemverilog the test bench code is in a program block
    - Program block is similar to a module and can contain code and variables and be instantiated in other modules
    - A program cannot have hierarchy such as instances of modules or interfaces



```
program test (arb_if.TEST arbif);
initial begin
@arbif.cb;
repeat (3) arbif.cb;
@arbif.cb.grant;
@(posedge arbif.cb.grant);
@(negedge arbif.cb.grant);
wait (arbif.cb.grant==1);
@(posedge arbif.cb.grant or negedge arbif.reset);
end
endprogram
```

Continue on active clock edge

Wait for 3 active edges

Continue on any edge

Continue on positive edge

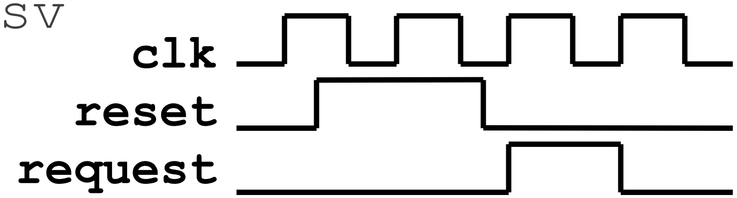
Continue on negative edge

Wait for expression to be true

Wait for several signals

Program Block

- Create testbench program: `test.sv`



```
program test(arb_if.TB arbif);
  initial begin
    // Asynch drive reset
        arbif.reset <= 0;
    #15ns arbif.reset <= 1;
    #35ns arbif.reset <= 0;

    // Synch drive request
    ##1 arbif.cb.request <= 1;
    ##1 arbif.cb.request <= 0;
    wait (arbif.cb.grant == 1);
  end
endprogram
```

ns!

Wait 1 clock cycle

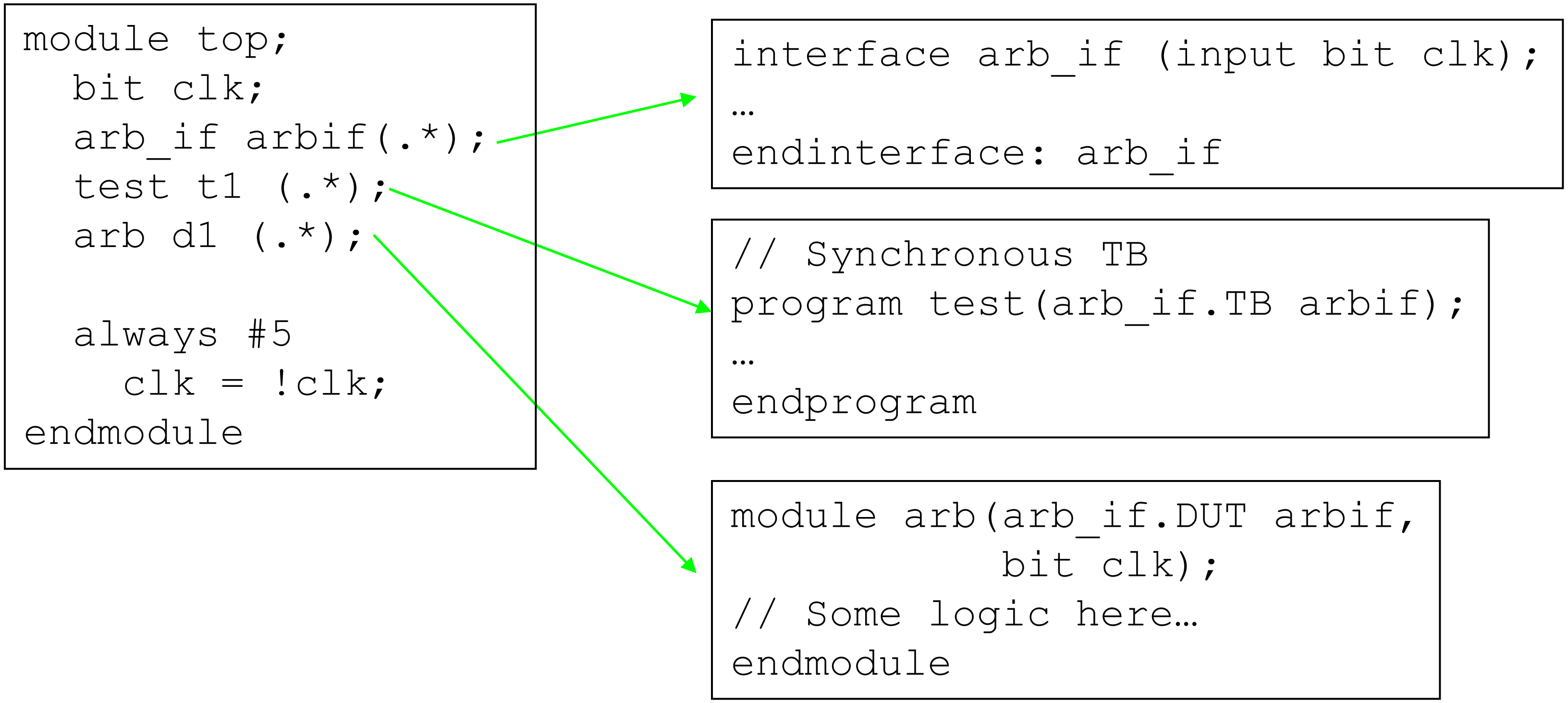
```
interface arb_if (input bit clk);
  logic grant, request, reset;
  clocking cb @(posedge clk);
    input grant;
    output request;
  endclocking
 modport TB (clocking cb,
             output reset);
endinterface: arb_if
```

Common mistake: forgot "cb." in signal reference `Error: arbif.request not visible via modport`

Testbench Environment – Top Block

- **Create top module**

**Implicit port connections:
The syntax .* connect ports
and signals with same names**



```
module top;
  bit clk;
  arb_if arbif(.*);
  test t1 (.*);
  arb d1 (.*);

  always #5
    clk = !clk;
endmodule
```

```
interface arb_if (input bit clk);
…
endinterface: arb_if
```

```
// Synchronous TB
program test(arb_if.TB arbif);
…
endprogram
```

```
module arb(arb_if.DUT arbif,
              bit clk);
// Some logic here…
endmodule
```

Thank You

Stimulus Timing

- The timing between the testbench and the design should be maintained
  - Ensure driving and receiving synchronous signals at the proper time in relation to the clock
  - Driving a signal too late or sampling it too early can cause the testbench to be off a cycle
  - To ensure proper sampling and signaling the testbench should be separate from the design