Ain Shams University
Faculty of Engineering
Computer and Systems Department
Spring 2024

CSE 313s
Selected Topics in Computer Engineering
Sheet 6

1. If a clocking block definition has timing specified as follows - what does it mean?

```
default input #2ns output #3ns
```

   a. All input signals in the clocking block are sampled 2ns before clocking event and all output signals are driven 3ns before clocking event
   b. All input signals in the clocking block are sampled 2ns after clocking event and all output signals are driven 3ns before clocking event
   c. All input signals in the clocking block are sampled 2ns after clocking event and all output signals are driven 3ns after clocking event
   d. All input signals in the clocking block are sampled 2ns before clocking event and all output signals are driven 3ns after clocking event

2. A single System Verilog “interface” can have multiple modports and multiple clocking blocks inside
   a. True
   b. False

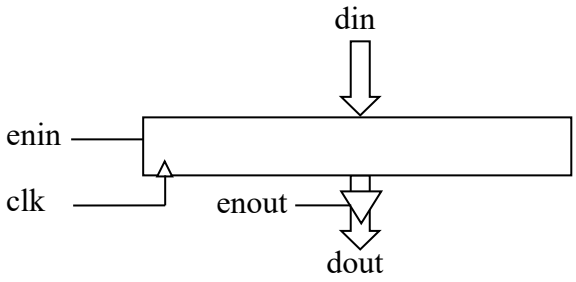
3. Modport is used to
   a. declare input and output skew
   b. declare direction of signals
   c. synchronize signals
   d. make interface complex

4. What is the use of modports?
   Modports restrict interface access within a module based on the direction declared. An example is shown below for a memory and its testbench.

```
interface intf (input clk);
  logic read, enable;
  logic [7:0] addr,data;

  modport dut (input read,enable,addr,output data);
  modport tb (output read,enable,addr,input data);

endinterface :intf
```

5. What is the need of clocking blocks?
   A clocking block is a construct that assembles all the signals that are sampled or synchronized by a common clock and define their timing behaviors with respect to the clock.
   - It is used to specify synchronization characteristics of the design
   - It Offers a clean way to drive and sample signals
   - Provides race-free operation if input skew > 0
   - Helps in testbench driving the signals at the right time
   - Features:
     - Clock specification
     - Input skew, output skew
     - Cycle delay (##)

Example:

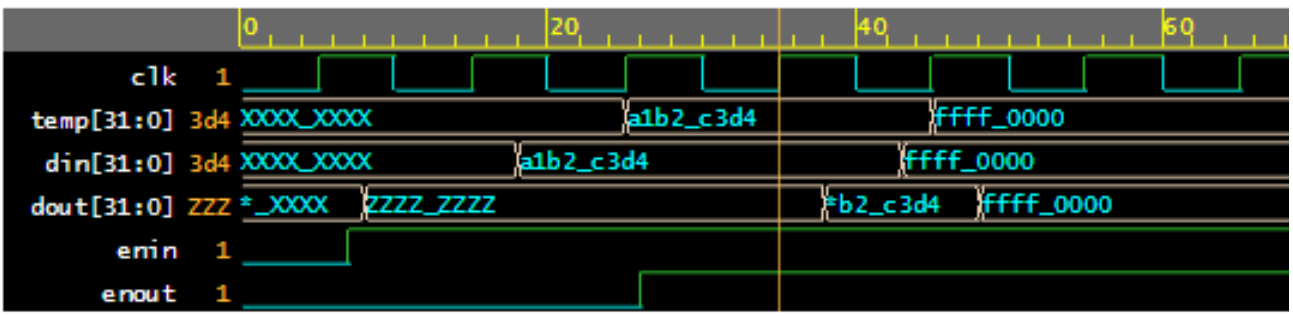



```
interface intf(input bit clk);
  logic enin,enout;
  logic [31:0] din, dout;

  clocking cb @(posedge clk);
    output #3ns dout;
  endclocking

  modport dut(input enin, enout, din, clk, clocking cb);
  modport tb(output enin, enout, din, input dout,clk);
endinterface
```

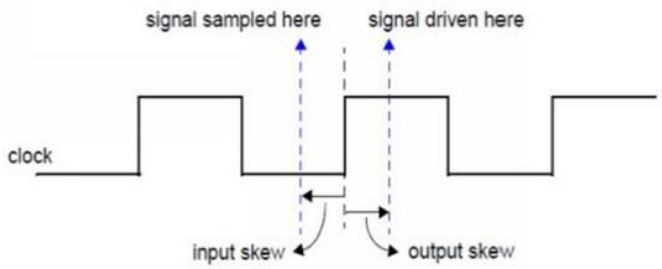
```
module test (intf.tb xyz);
  initial begin
    xyz.enin = 0;
    xyz.enout = 0;
    #7 xyz.enin = 1;
    #11 xyz.din = 32'hA1B2C3D4;
    #8 xyz.enout = 1;
    #17 xyz.din = 32'hFFFF0000;
    $display(xyz.dout);
  end
endmodule
```

```
module design (intf.dut abc);
  reg [31:0] temp;
  always @(abc.cb) begin
    if(abc.enin)
      temp = abc.din;
    if(abc.enout)
      abc.cb.dout <= temp;
    else
      abc.cb.dout <= 32'hZ;
  end
endmodule
```

```
module top;
  bit clk;
  always #5 clk = ~clk;
  intf f1(clk);
  design d1 (f1.dut);
  test t1 (f1);

  initial begin
    $dumpfile("test.vcd");
    $dumpvars;
    #200 $finish;
  end
endmodule
```



In above example, we have defined a clocking block with name *cb* and the clock associated with this clocking block is *clk*. We specify that skew for output is 3ns. The output skew defines how many time units after the clock event the signal will be driven. We could also define an input skew (but we didn't). The input skew defines how many time units before the
clock event the signal is sampled.



6. What are interfaces in SystemVerilog?
   An interface is a way to encapsulate signals into a block. All related signals. Are grouped together to form an interface block so that the same interface can be re-used for other projects. Also it becomes easier to connect with the DUT and other verification components.
   An interface can be instantiated in a design and can be connected using a single name instead of having all the port names and connections.

7. What is a modport construct in an interface?
   modport (short form for module port) is a construct in an interface that lets you group signals and specify directions. Following is an example of how an interface can be further grouped using modports for connecting to different components.

```
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);
  modport MONITOR (input request, grant, reset, clk);
endinterface
```

   In this example, you can see that the same signals are given different directions in different modports. A monitor component needs all signals as input and hence the modport MONITOR of interface can be used to connect to monitor. A test or a driver will need to drive some signals and sample other signals and above example shows a modport TEST that can be used

8. Are interfaces synthesizable?
   Yes, interfaces are synthesizable.

9. What is the difference between the following two ways of specifying skews in a clocking block?

```
input #1 step req1;
input #1ns req1;
```

The clocking skew determines how many time units away from the clock event a signal is to be sampled (input skew) or driven (output skew).
A skew can be specified in two forms:

1. either explicitly in terms of time as in the second line above, where the signal is sampled 1ns before the clock.
2. or in terms of time step as in the first line above, where the step corresponds to global time precision (defined using `timescale directive).

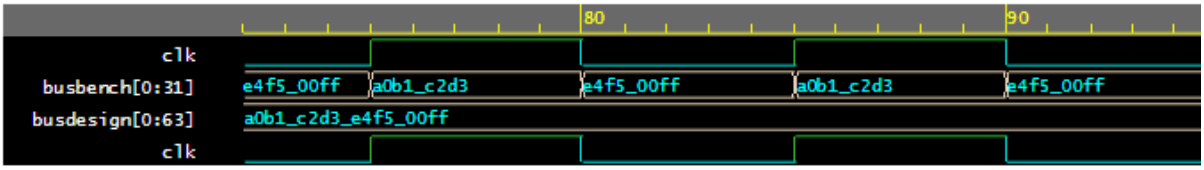
10. Write code for the below requirement
DUT has data bus of 64 bit which is driven on positive edge of clock.
TESTBENCH has data bus of 32 bit which can sample on both positive edge and negative edge.
On positive edge, drive [0 to 31] bits of DUT to TESTBENCH bus and on negative edge drive [32 to 63] bits of DUT to TESTBENCH bus.
All the above logic should be done inside the interface itself.

```
interface intf(input bit clk);
  logic [0:63] busdesign;
  logic [0:31] busbench;

  clocking cb1 @(posedge clk);
    output busdesign;
  endclocking

  clocking cb2 @(posedge clk or negedge clk);
    output busbench;
  endclocking

  modport dut(clocking cb1, input clk);
    modport tb (clocking cb2, input busdesign, clk);
endinterface
```

```
module design(intf.dut abc);
  always @(abc.cb1)
    abc.cb1.busdesign <= 64'hA0B1C2D3E4F500FF;
endmodule
```

```
module test(intf.tb xyz);
  always @(xyz.cb2) begin
    if (xyz.clk == 1)   // rising edge
      xyz.cb2.busbench <= xyz.busdesign[0:31];
    else                // falling edge
      xyz.cb2.busbench <= xyz.busdesign[32:63];
  end
endmodule
```

```
module top;
  bit clk;
  always #5 clk = ~clk;
  intf f1(clk);
  design d1(f1.dut);
  test t1(f1.tb);

  initial begin
    $dumpfile("test.vcd");
    $dumpvars;
    #200 $finish;
  end
endmodule
```



11. If clocking block is not used, then what happens?

The usage of clocking block is majorly to reduce the timing efforts in the testbench.

They allow you to collect all the timing related to a set of synchronous lines and put it all in one place. Then, you can use that information to write clean testbench stimuli based on the clocking and skewing defined in the clocking block.

So not using clocking blocks will just increase your coding efforts.