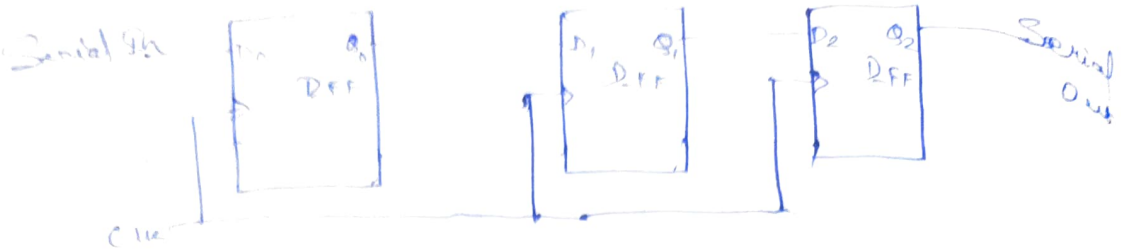
System Tasks:-

1) Shift registers




```
module Shift_reg_3stage (input in, output out, input clk);
reg Q1, Q0;
always @(posedge clk)
    Q0 <= in;
    Q1 <= Q0;
    out <= Q1;
end
```

Application

Shift registers are used in making memory,

→ In solving CDC, problems b/w modules.

→ Serial to parallel, parallel to serial conversi

→ Frequency divison.

→ Producing time delay.

→ High speed data transmission.

2) a) Synchroniser

→ A synchroniser is a technique of passing a signal coming from a module with ~~different~~ one clock to another module operating wrt another different clock.

→ This is necessary in order to prevent metastable ~~racing conditions~~ from arising in CDC.

→ As signal passes b/w different clock domains, they can give rise to ~~racing~~ metastable condition (due to different hold/setup time).

Application -

→ In CDC

→ Communication interfaces like SPI, I2C - etc.

b) Sometimes while signal crossing b/w domain, delay is not enough, i.e., it may arrive at uncertain time of hold/setup, causing metastability. In order to ensure reliability & give proper delay, multi-stage synchronisers are added.

(Signal stabilises before passing).

2) Timescale

→ `timescale is a macro / compiler directive in verilog that instructs the compiler as to what should be the individual time units & time precision for simulation.

→ Delays are interpreted on the basis of timescale of timescale

Eg - `timescale 1us/1ps.

#10 ⟶ means 10 ns delay.

→ Time precision helps in accurate timing analysis of circuit & affects simulation resolution results

Eg - #(11/2) = 5 ns for timeprecision = 1ns
(delay rounded up 1 ns)

& = 5.500 ps for timeprecision = 1ps
(delay rounded upto nearest 1ps).

Timestep - Smallest increment by which simulation time advances. (How often values are updated in simulation)

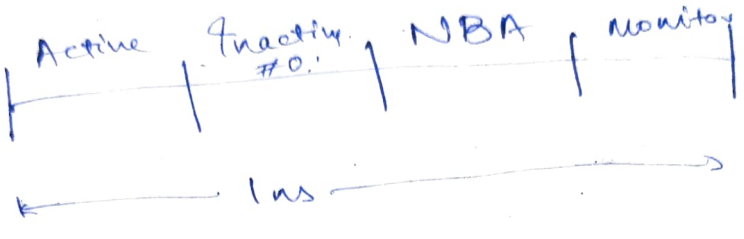
timeprecision ↑ more accurate result, more computation

timestep ↓ = faster simulation, lower accuracy

⇒ Default timescale 1ns/1ns in questasim.

4) Scheduling regions in Verilog :-

⇒ They define the order in which statement/a proc events/a are processed during simulation.



Active region

⇒ Here, is where blocking & NB assignment are evaluated. Also assigned in blocking statement.

Inactive

⇒ Events scheduled at #0, delta delay, execute here after active region, but before next time step.

⇒ helps in solving racing cond^n.

NBA

⇒ NBA assignments are evaluated in active / but assigned in NBA region.

Monitor

$monitor, $ strobe execute here, after all assignments are done, at the very end, conclusion of events in timestep.

⇒ No assignments, only observation.

5) Inter delay statements

Ex ~~#10 a = rhs~~ #10 a=10;

or

#10; a = 10;

or

#10; a <= 10;

begin

Intra delay statement (Ex- gate delay)

~~#10 a = b+c;~~ → a = assigned 10 after 10 sec but (b+c) evaluated at 0ns

a = #10 b+c;

a = declared at 0ns

(b+c) evaluated at 0ns.

assignment after 10 ns.

6) System tasks :-

display. $monitor, $display, $strobe, $write

~~random~~ file = $fopen, $fclose, $feof, $fdisplay, $fmonitor

reading args: $value $plusargs

ending sim = $stop, $finish

backdoor access = $readmemh, $writememh
$readmemb, $writememb

Seed

→ $random(seed) is a system function used to generate a random 32 bit value which can be assigned to variable.

→ After each consecutive run, $random generates a new value.

Seed - This value is used to generate a different sequence of bits than usual.

→ Each seed value, generates a different random no.

→ In real life seed can be generated using $time, or $realtime or by giving delays.

→ Also seed can be the o/p of previous execution.

→ This ensures time is a least overlap or random numbers.

8) Compiler directives :-

→ Compiler directives is a keyword in verilog that is used to instruct the compiler during compilation about aspects of simulation.

Ex - 'timescale tells compiler about timestep / time precision to be followed during simulation.

→ They are overridden in the run.do file during compilation.

⇒ They donot affect the hardware implementation directly.

Types

i) `define - To define constants / expressions for reducing repetitive code.

`define WIDTH 8

ii) `undef - Undefine a previously defined macro.

`undef WIDTH

iii) `include - include external files for modularity, reusability.

`include "design.sv"

iv) `ifdef - Conditional compilation (For implementing feature / testcases)

v) `timescale - set timing in simulation.

vi) `celldefine - mark a code block as a cell.

9) Parameter (local)

⇒ Declared inside module

⇒ Overridden in module instantiation

⇒ Can hold constants.

⇒ To define module parameters.

Macro (`define/global)

⇒ Declared outside module.

⇒ Overridden in compilation

`define can be overriden in (outside) compilation

⇒ Can hold constants/expressions

⇒ Used for debugging

→ Macro are generally preferred for conditional compilation, & debugging as they can store both constant & expressions.

10) XMR

XMR = Cross Module Reference.

→ XMR is a method of referencing one module inside another module. And declaring & passing argument through the current module to the other one.

→ We can access piece of code like variable, task, functions from different module to present module.

Ex - module tb;
integer a; task print();
$display("%d", a);
end task
end module

module top;
tb t(); - mod. instantiation.
t.a = 10;
t.print(); o/p = 10.
end module.

14) Vector good practice - use relevant words.
- scalar = small vector = small
- array = Capital.

Parameter good practice :- write in capital.