

Verilog HDL

1. Bitwise operators

2. Parameter

3. Different data-types reg, wire (ip, op or implicitly wire)

4. Comparison operators

5. \$finish

6. Blocking, Non-blocking, continuous assignment

7. Vector [msb : lsb]

8. Arithmetic operators.

9. Concatenation operator.

10. always @ block

11. *missing sensitivity list

else

default -> missing case condition

12. if, case, for

13. *nested if else

14. \$display, \$monitor

15. posedge, negedge

16. Order in blocking assignment.

17. Replication operator.

18. Ternary operator

19. Function & task (static / automatic)

20. Stratified event queue

21. primitive (Built-in)

22. while, forever

26-09-2024

Hardware modeling using verilog

- Indranil Deygupta

Verilog → hardware description language

behavioral, structural

Area, speed, energy consumption

1961 - First IC

Technology CMOS, FinFET (14nm)
(22nm)

Design flow Specification, Synthesis, Simulation, Layout, Testing & analysis
Use of CAD tool

Behavioral → RTL → gate → transistor.

Design Idea → Behavioral Design

Data path Design

netlist (blocks connected) → Structural

Logic Design

gate level, standard cell

Physical Design

regular polygon.

Manufacturing

1. Introduction about lecture
2. Design Representation.
- 3 Getting started with verilog.
4. VLSI Design Styles.

Design Representation

Verilog

System Verilog

VHDL

Timing Constraints

Timing Analysis

Timing Driven Placement & Routing

Timing Driven Placement

Timing Driven Routing

Timing Driven Placement & Routing

30/09/2024

Book- Digital logic design using
verilog.

Verilog HDL

→ IEE 1495 1364
1995, 2001, 2005

- Vivado

1983-84 gate automation

→ Phil morby

- Vivado

- Questasim

Why HDL?

- in C++, designing a counter in it or any high level language.

- not hardware counter

- no description abt hardware

- i/o port

- no direction control

- no trigger (rising edge posedge
falling edge Negedge)

- to

1) Model digital design

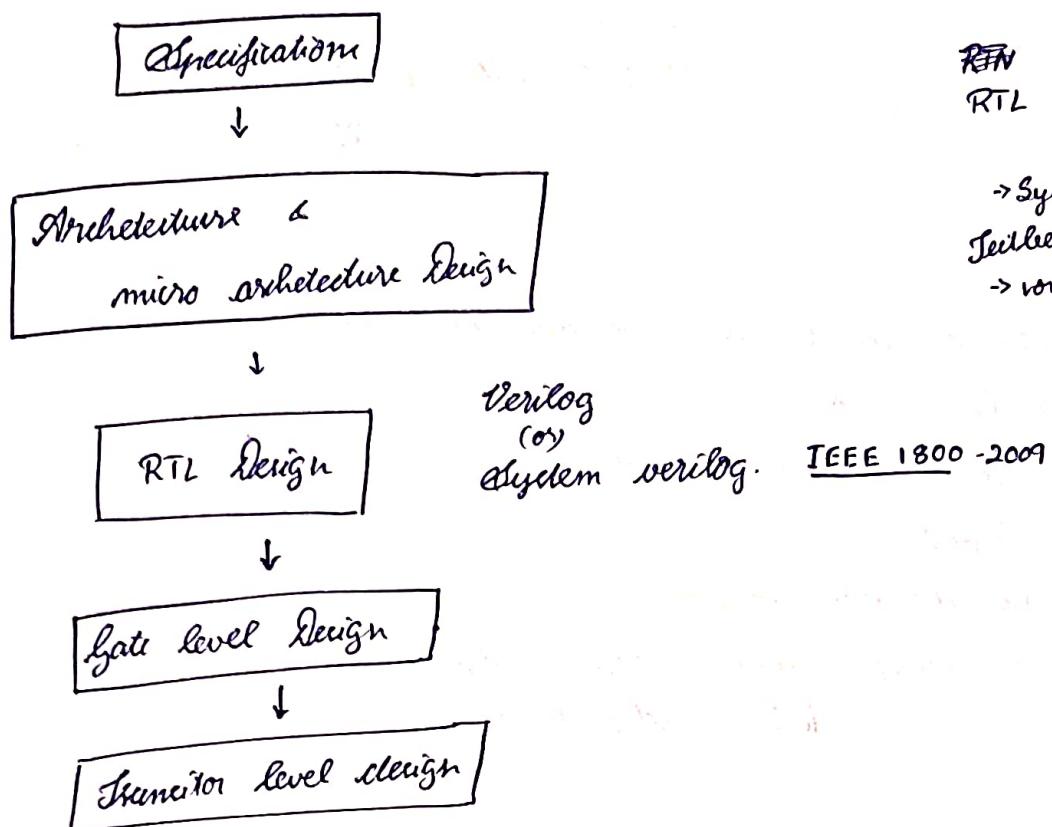
2) direction control input
output
input output in out.

3) Edge sensitive
pos edge
neg edge

4) Time control generate signal after delay
in a sequence.

5) Switch level modeling.

Design flow



Verilog HDL

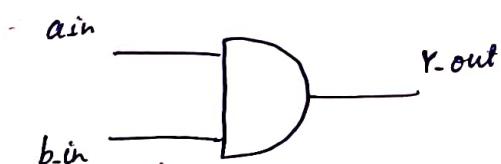
Synthesizable & non-synthesizable construct.

↳ RTL Design.

↳ Behavioral
used in verification

similar generation

intention → not to infer logic.



Syntax Verilog HDL explained.

Dont use red for signals

X → red in for dont care

```
module <module-name>(  
    input <port-name>,  
    ...  
    ...  
    output <port-name>  
)
```

// functionality of block

endmodule

Combinational logic

assign

↳ all are assigned concurrently (not line by line as in any compiled or interpreted language)

Forced level simulation

→ force clock on each i/p

complexity increase 2^n → no. of i/p's

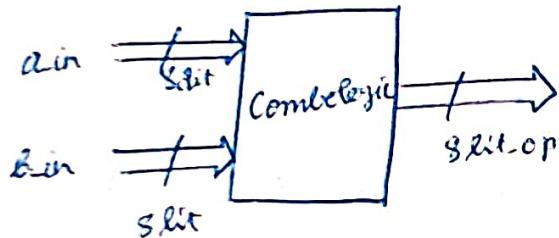
01-10-2024

Vector n-bit ifo

Bitwise operators & their application in RTL Design

Almost inputs & outputs are vector (ie) multi-bit system conv o/p
as a bus. Logical operators are not suitable, Bitwise operator
to be used, following are bitwise operator

- 0 - Bitwise and
- 1 - Bitwise or
- ~ - Bitwise not
- \wedge - Bitwise xor
- $\wedge\wedge$ } Bitwise xnor
- $\wedge\sim$



module logic_gates

input [7:0] a-in,

input [7:0] b-in,

output [7:0] y-out

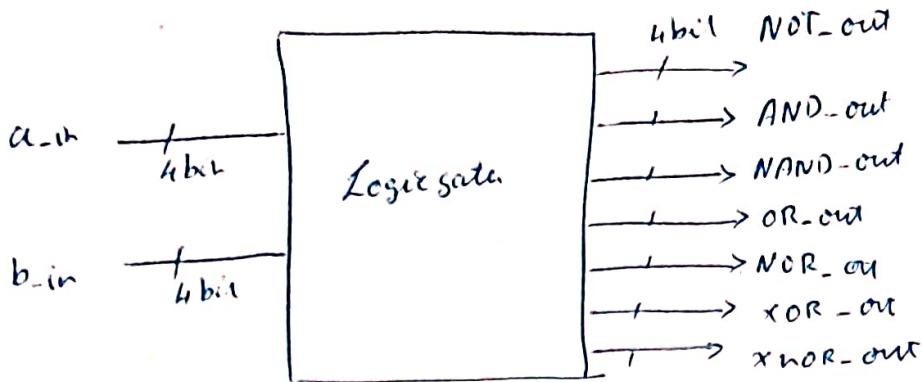
;

assign y-out = a-in & b-in

end module.

Exercise

RTL Design



parameter (parameters) (Name usually capital letter)

```
module add_bit #(parameter DATA_WIDTH=4)
  input [DATA_WIDTH-1:0] a-in,
  input [DATA_WIDTH-1:0] b-in,
  output [DATA_WIDTH+1:0] Y-out;
  assign Y-out = a-in + b-in;
end module.
```

Exercise 3

RTL Design. use parameter DATA_WIDTH = 4

Test Bench \Rightarrow user non synthesizable construct

\hookrightarrow tb-name

C.DATA_WIDTH(4)

* test bench uses non-synthesizable construct

module tb.add_bit();

```
input <reg>#(4) a-in;
input <reg>#(4) b-in;
output <wire>#(4) Y-out;
add-bit #(.DATA_WIDTH(4)) DUT
```

```

  begin
    a-in(a-in),
    b-in(b-in);
    Y-out(Y-out));
  end
```

\rightarrow execute only once
procedural block used in test bench
initial to generate a stimulus.
begin
 a-in=0; b-in=0;
 #10 a-in=4'b100; b-in=4'b1010;
 #20 a-in=4'b1100; b-in=4'b1011;
 #200 final.
end

* a-in, b-in need to be assign specific value
within initial block there should be if

type reg, not wire
Error: procedural assignment to
non-register not permitted)

reg/integer/time/generator

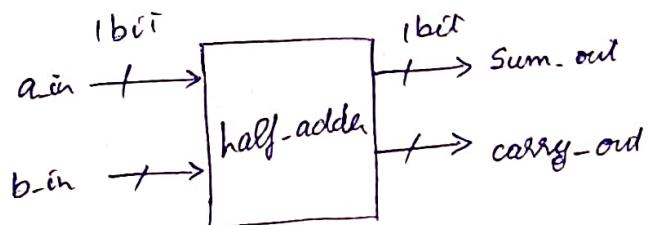
Hyper industrial standard & requirement.

- usage of naming convention - in input
 - out output.
 - io inputoutput
- improve readability of design & testbench i.e. use comments wherever required.
- use synthesizable constructs in RTL design file.
- use non-synthesizable constructs in testbenches
- to check for functional correctness of design use testbench not logic level simulation.

Evening session

Exercise 6

Complete RTL design of half adder



Exercise 7

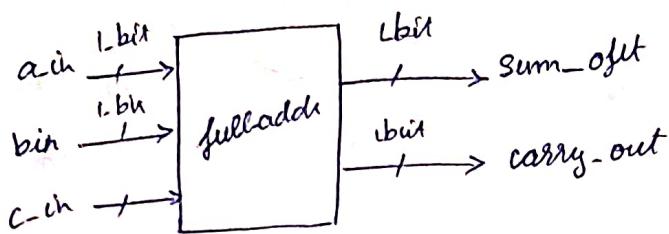
tb_half-adder

Complete a testbench to check functional correctness of exer.

~~Exercise 8~~ Relaunch → to see waveform.

Exercise 8

use min no of half adder & OR gate to have 1-bit full adder



instance of half adder.

Exercise 9

Complete a test bench of exercise 8.

Exercise 10

RTL Design 1-bit comparator



Exercise 11

Testbench exercise 10

\$finish

>
<
==
>=
<=

} comparison operator.

03-10-2024

Exercise 12

Check order of continuous assign

→ Blocking

→ Non-blocking.

(BA) =

(NBA) <=



↓

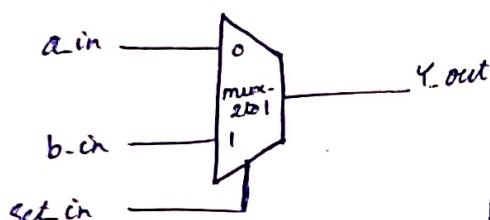
used for assignment
in procedural block

Procedural blocks are

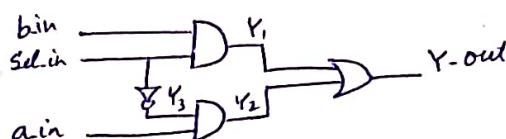
- initial, always.

Continuous assign → in module

Consider RTL design of 2:1 MUX



Sel.in	Y.out
0	a-in
1	b-in



module mux2to1(

input a-in, b-in, sel-in,
output Y-out);

wire Y1, Y2, Y3;

assign Y1 = sel-in & b-in;

assign Y2 = ~~sel-in~~ ^{Y3} & a-in;

assign Y3 = !sel-in;

assign Y-out = Y1 | Y2;

} Here Y3 is used in previous line
(i.e. before assign to Y3 that execution
order doesn't matter.
all execute parallelly.

end module

Exercise 13

test bench for exercise 12.

Port select

module --

(input [7:0] a_in,

:

);

wire [3:0] Y1;

wire Y2;

assign Y1 = a_in[3:0];

assign Y2 = a_in[7];

:

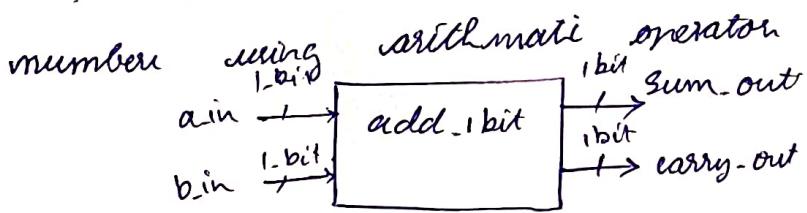
end module

Arithmetic operators

+ , - , * , / , % , **
(power)

Exercise 14

Complete RTL design to perform addition of 2-bit binary



concordation operator

{ ... , - , ... , - } \Rightarrow to assign value or for overflow

[7:0]
D₇ ... D₀
MSB LSB

{0:1}
D₀ ... D₇
MSB LSB

Exercise 16

add_n_bit

Exercise 17

last bench for exercise for above

Evening session

Exercise 18, 19 → Subtractor n-bit

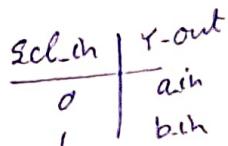
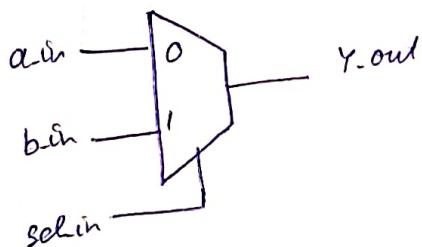
Exercise 20, 21 → 2:4 decoder

always @, if...else

always @ (K sensitivity list variables)

begin
!
end.

(eg) consider 2:1 MUX



module mux_2to1(input a-in,
input b-in,
input sel-in,
output y-out);

always @ (a-in, b-in, sel-in)

begin
if (sel-in == 1b1)

y-out = b-in;

else
y-out = a-in;

end.

Inside if block all assignment are blocking in combinational logic

non-blocking assignment are used to model sequential logic \leftarrow

\Rightarrow Why?

In this design-

y-out is output & can't be of wire type as it is assigned within always procedural block, so y-out should be of a reg type

Exercise 22

Complete RTL design of 2:1 max & test bench. use always procedural block with if...else

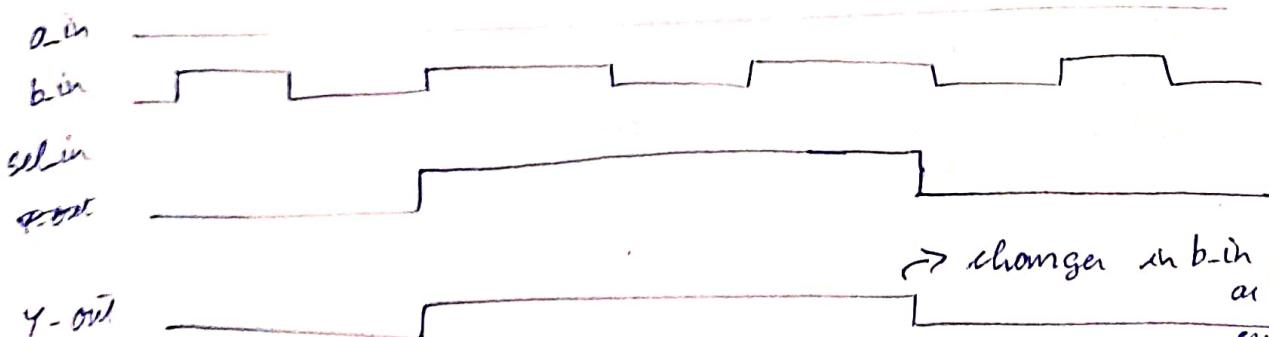
* missing sensitivity list

for example if b-in is not included in the sensitivity list tool will through a warning of missing sensitivity list signal \hookrightarrow nota error

(ii) reference signal b-in should be on a sensitivity list.

\Rightarrow schematic synthesis will work as expected

if b-in is missing always procedural block will not be invoked & there will be simulation / synthesis mismatch.



\Rightarrow changes in b-in not reported as b-in missing in sensitivity list (11)

always @ (*) \Rightarrow introduced in IEE 1364-2001

\rightarrow indicates include all required inputs in sensitivity list



+ missing else

\hookrightarrow will generate a latch & tool will generate a warning.

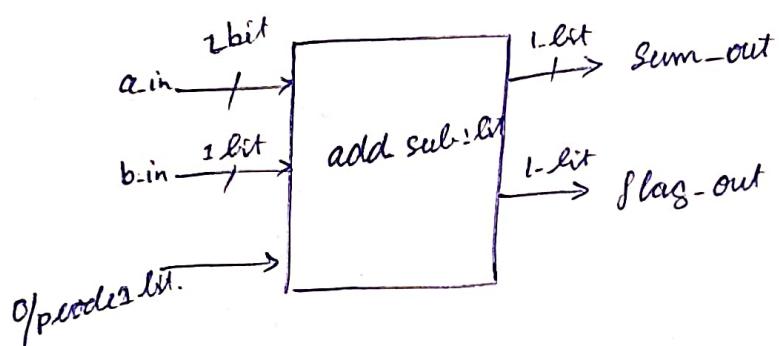
\hookrightarrow intelligent tool will flag a warning on intelligent latch in a design

04-10-2024

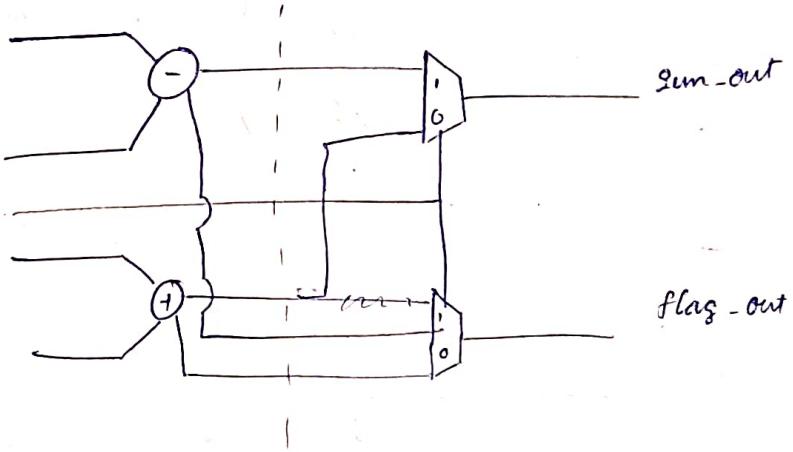
Exercise 23

RTL design for following:

Design 1-bit adder subtractor



opcode	
0	add
1	sub



multiple always @ block execute concurrently

Nested if... else

always @ *

begin

if (condition)

begin

:

end

else if (condition2)

begin

:

end

:

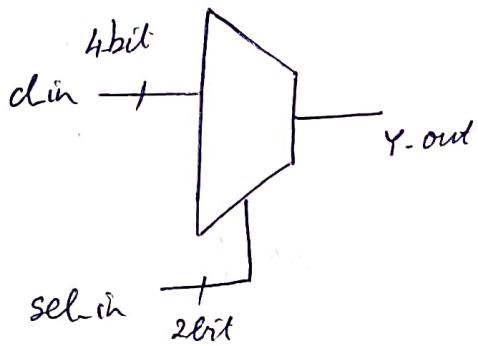
else

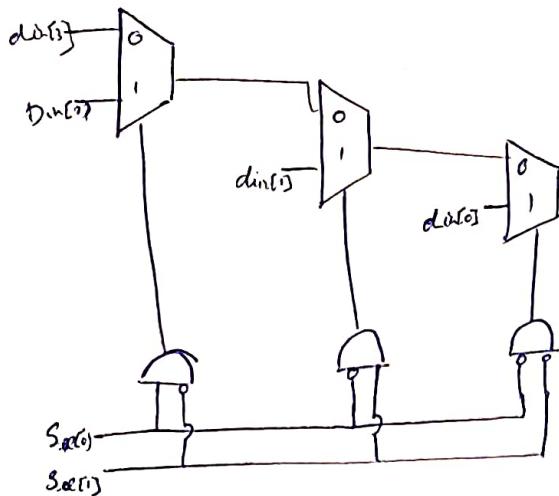
begin

:

end

Exercise 26
PT2 design of 4:1 mux needed if else





boxed: nested if else will infer priority logic] so never use nested if else

dangling if generates latch in design generated.

case C?... end case

case (var1)

(condition-1) : <statement to execute if condition-1>;

: begin

: end.

default : < >

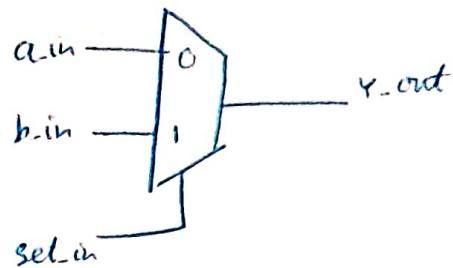
endcase

boxed: missing case condition will infer a latch]

casez, casez are two other variants \Rightarrow discuss subsequently.

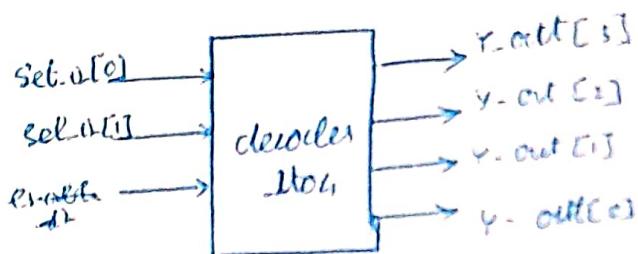
Evening session

~~Exercise 27~~ RTL design of 2:1 MUX using case construct



Exercise 28

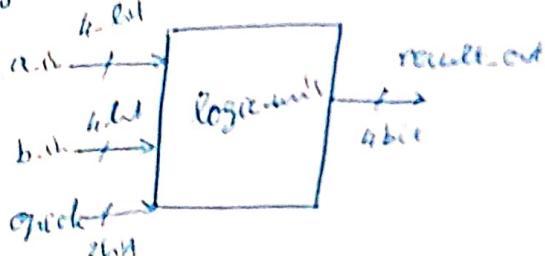
Complete RTL design of 2:4 decoder using case construct



Exercise 29

Design logic unit to perform following operations on two

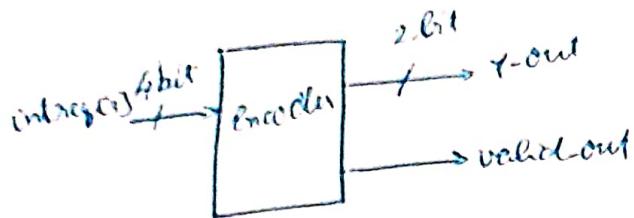
4-bit binary numbers



opcode	operation
0	NOT
1	OR
2	AND
3	XOR

Exercise 30

Design priority encoder to



int_req(3) has highest priority

& int_req(0) has lowest priority

valid_out = 0 when all int_req

Exercise 31

Create a testbench.

[time unit / time precision]

[monitor] → used to display the required ips and oips at various

time (eg)

05-10-2024

always @*

} \Rightarrow always @ (posedge clock)
 begin
 $q_{\text{out}} \leq d_{\text{in}}$; // non-blocking assignment
 end

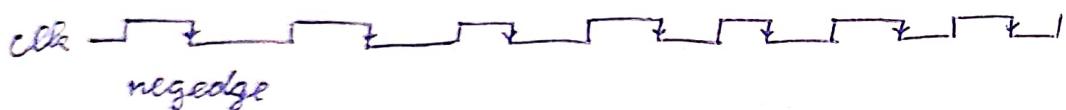
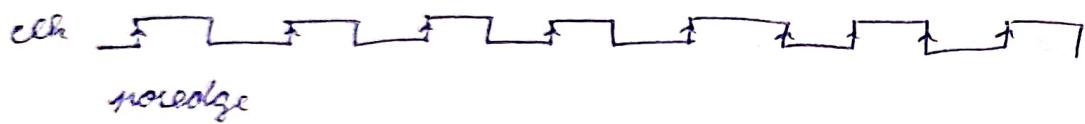
@ (posedge)

@ (negedge)

Blocking assignments, Non-Blocking assignments. BA & NBA

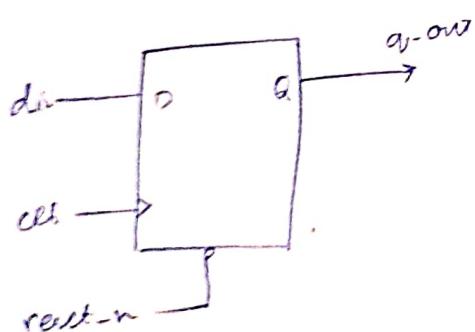
The discussed earlier blocking assignments are used to model
 a combinational logic '=' is used in the assignment within
 procedural block

It is recommended to use NBA within sequential logic (register). NBP assignments are using ' \leftarrow '. Consider positive edge sensitive D-flip flop. we can use posedge clock where posedge indicates low to high transition. To have negative edge sensitive flip flop we can use negedge of clock which is high to low transition.



Exercise -

Rising active low edge sensitive flip flop with a assign active low reset.



Now

Consider clk, reset_n
mix of level sensitive & edge sensitive are not synthesizable
by
edge re
trigger

\$ display

\$ monitor

Exercise 33

Synchronous reset D-SI

Exercise 34

Test bench.

Active low sync reset

always @ (posedge clk)

begin

if (!reset-n)

q_out <= 1'b0;

else

q_out <= d_in;

end

Exercise 35

negedge D SI active low aync reset

Exercise 36

modify RTL of above to have active low sync reset.

Exercise 37

Find out logic inferred by following design

always @ (posedge clk, negedge reset_n)

begin

if (reset_n)

begin

q1_out <= 0;

q2_out <= 0;

end

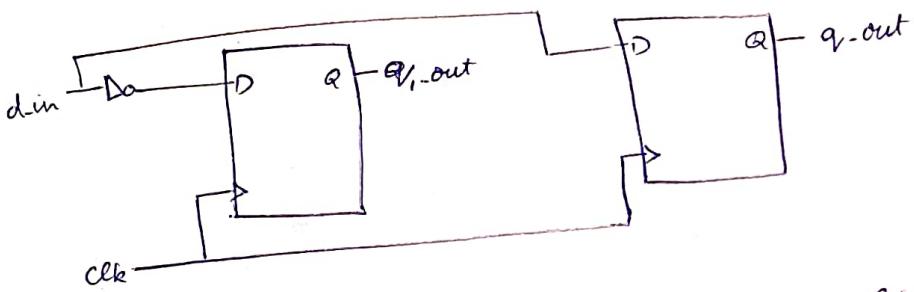
else

begin

q1_out <= d_in;

q2_out <= !d_in;

end end



If designer intention is to have shift register having 3FF, and considers the following design which use blockers assignment as shown, this will infer a single FF not a shift register.

module shift-reg (input din
input reset_n, clk,
output reg q_out)

Change this using NBA it
will infer shift register.

reg temp1, temp2;

always @ (posedge clk, negedge reset_n)

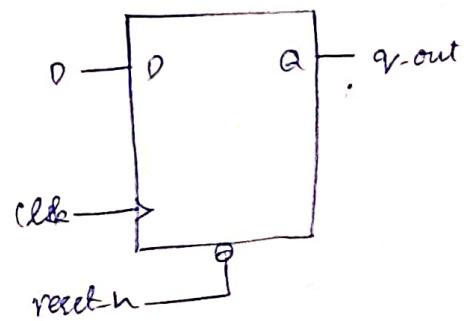
begin

temp1 = d_in;

temp2 = temp1;

q_out = temp2;

end



always @ (posedge clk, negedge reset_n)

begin

if (reset_n)

begin

temp_1 <= 1'b0;

temp_2 <= 1'b0;

q_out <= 1'b0;

end

else

begin

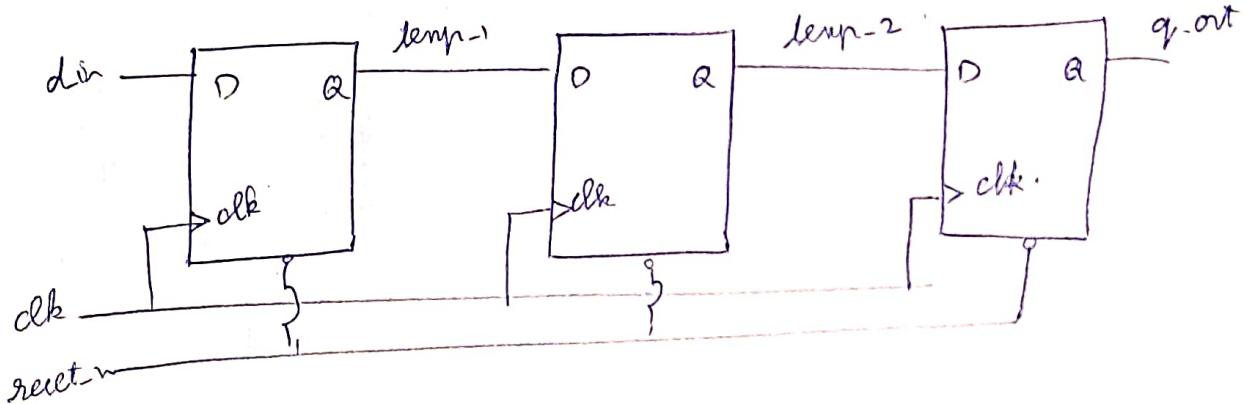
temp_1 <= d_in;

temp_2 <= temp_1;

q_out <= temp_2;

end

end



if any NFA is change in order no change in logic inferred

* Reordering of blocking assignment

changes the number of flip flop from no of assignment $\rightarrow 1$

* Don't mix blocking, non-blocking assignments.

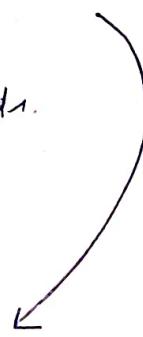
③ FF

$q_{\text{out}} = \text{temp}_2;$
 $\text{temp}_2 = \text{temp}_1;$
 $\text{temp}_1 = d_{\text{in}}$

② FF

$q_{\text{out}} = \text{temp}_2;$ $\text{temp}_1 = d_{\text{in}};$
 $\text{temp}_1 = \text{temp}_2;$ $\text{temp}_2 = \text{temp}_1;$
 $\text{temp}_2 = \text{temp}_1;$ $q_{\text{out}} = \text{temp}_2$

① FF



given in verilog

active

assign BA, RHS
NBA

inactive

#odelay

NBA

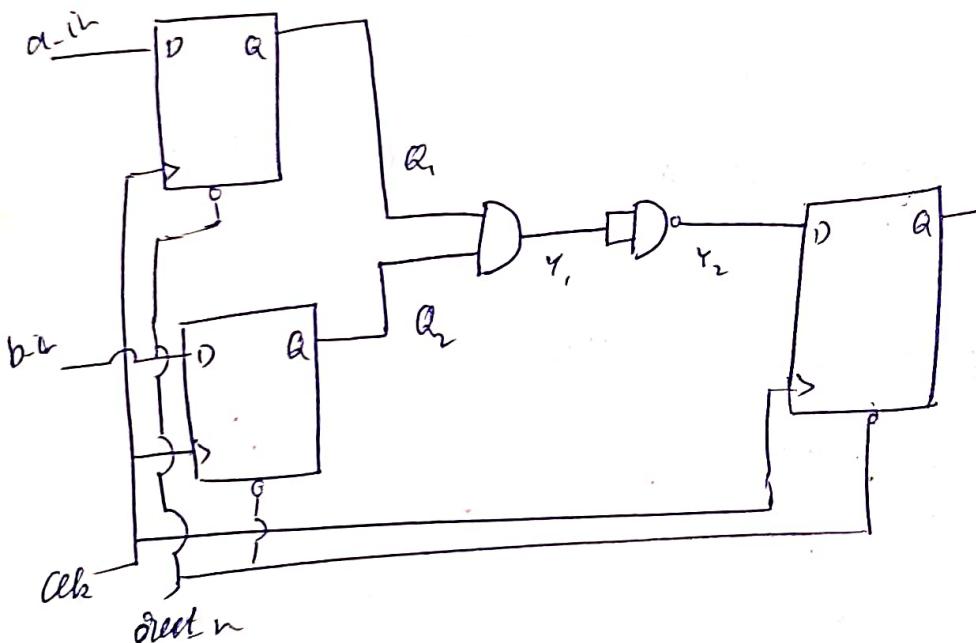
LHS of NBA

monotonic

monoton
ic delay

Exercise 3S

Complete RTL design for the following.



Exercise 40 counter 2bit up async reset

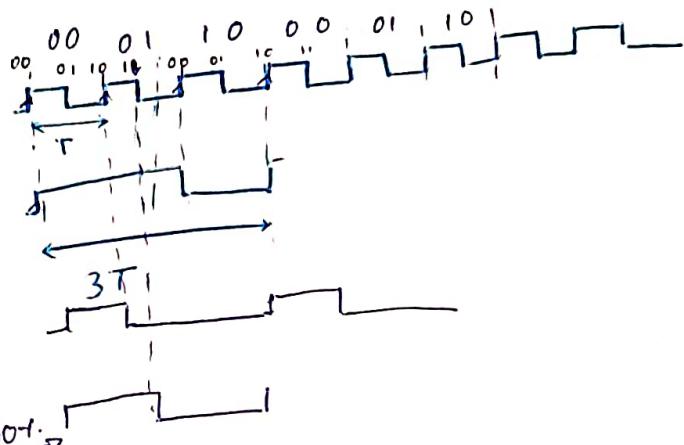
Exercise 42 counter n bit up/down async reset

replication operator `{.dDATA_WIDTH{`b0} };` \Rightarrow gives warning

Exercise 44 with load

Exercise 46 Y-2

Exercise 48 Y-3 counter



Exercise 50 50% duty cycle. $50\% \Rightarrow$

07-10-2024

Exercise 51 Mod 5 counter, load, async active low reset (up-down-in)

Exercise 52 Test bench

€

Exercise 53 Ring counter 4-bit, load 8 at initial async active low reset

Exercise 54 Test bench. $q_{out} \leftarrow \{q_{-out}[0], q_{-out}[3:1]\};$

1000
0100
0010
0001

Exercise 55 0, 8, 12, 14, 15, 7, 3, 1, 0 ... Sequence generator.

\downarrow
 \downarrow
 \downarrow
 \downarrow
 \downarrow
 \downarrow
 \downarrow
 \downarrow

Exercise 56 Test bench

Sequence generator.
Shannon counter
(twisted ring counter)

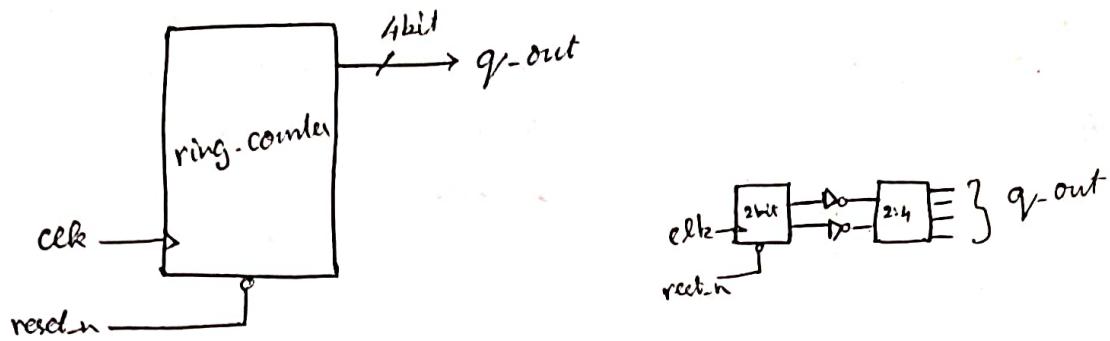
$q_{-out} \leftarrow \{q_{-out}[0], q_{-out}[3:1]\}$

0000
1000
1100
1110
1111
0111
0011
0001
0000

Evening session Exercise 57

RTL Design of 4 bit ring counter using 2:4 decoder &

2 bit up counter



00, 01, 10, 11 \rightarrow 1000, 0100, 0010, 0001

J
11 10 01 00

Exercise 58

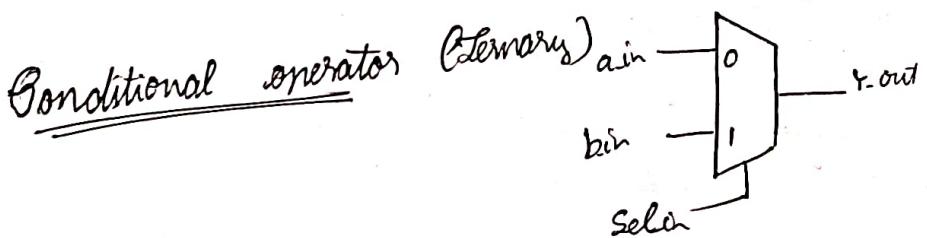
Test Bench

Exercise 59

Shift register, 4 bit left shift, right shift;

Exercise 60

Test Bench & monitor.



assign y-out = (sel==1)? b-in : a-in

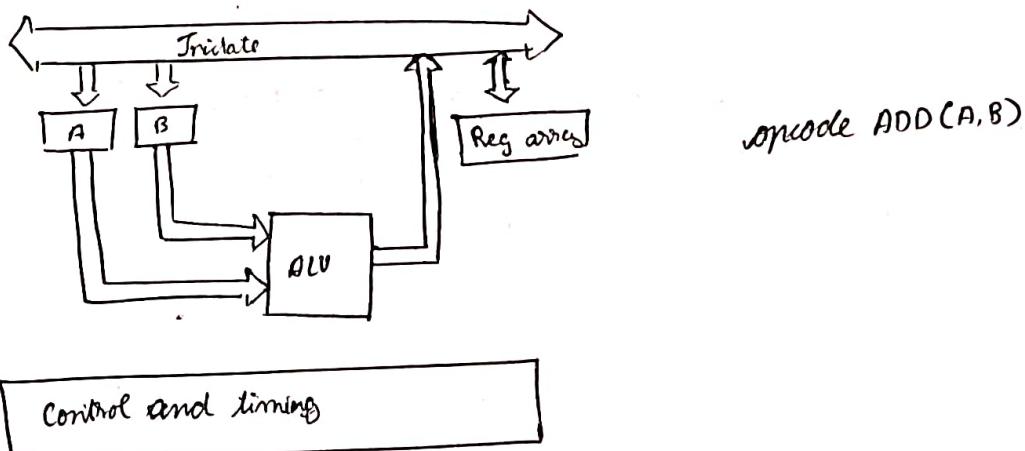
08-10-2024

4-to-1 MUX using ternary

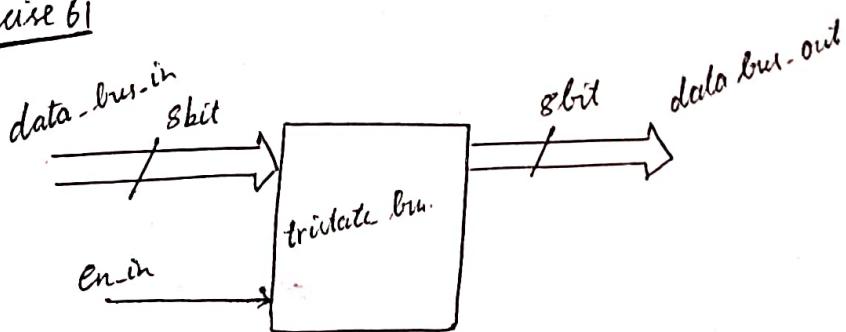
1-bit adder, subtractor conditional operators.

a-in result-out
b-in flag-out
opcode

ALU



Exercise 61

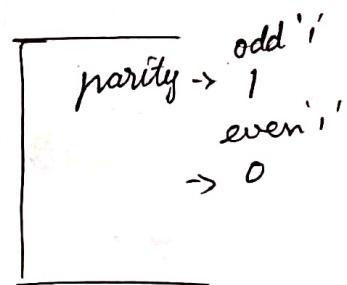
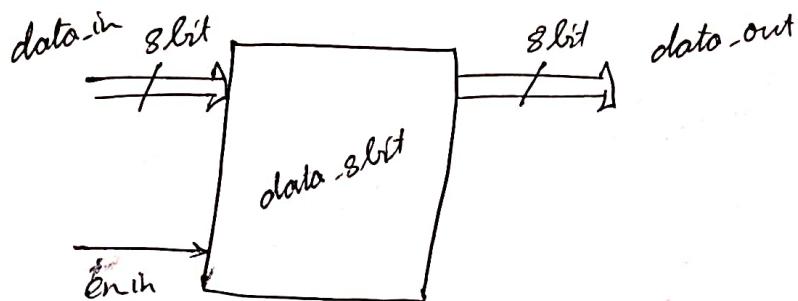


en-in	operation
1	data-bus-in = data-bus-out
0	high impedance = $Z(0)Z$

Exercise 62

Intentional latch

positive level sensitive 8-bit latch



Test bench

09-10-2024

- ✓ assign
- ✓ always @ +
- ✓ always @ (posedge or negedge) ↪ sync, async reset
- ✓ Ternary operators.

Function & Task

function <return type> <name of function> <arguments>
(poslist)

endfunction

#,@, wait
posedge, negedge

Rules..

- fn should have atleast one i/p arg
- fn can call another function but can't call task
- fn can't control of time control construct
- fn can't control of output or input arg
- fn cannot have non-blocking assignment, deassign, force, release

→ sometime we need to perform some operation multiple times in such scenario we can use a function call (function, endfunction) are keywords

(eg) function [3:0] addition (input [3:0] a_in, b_in);

begin

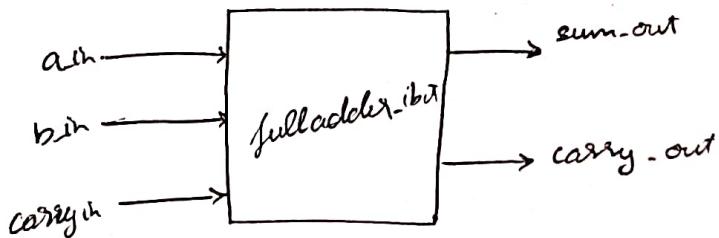
$$\text{addition} = \text{a_in} + \text{b_in};$$

end

endfunction

Exercise 63 addition 4 bit using junction

Exercise 64 1 bit full adder using 1 bit half adder.

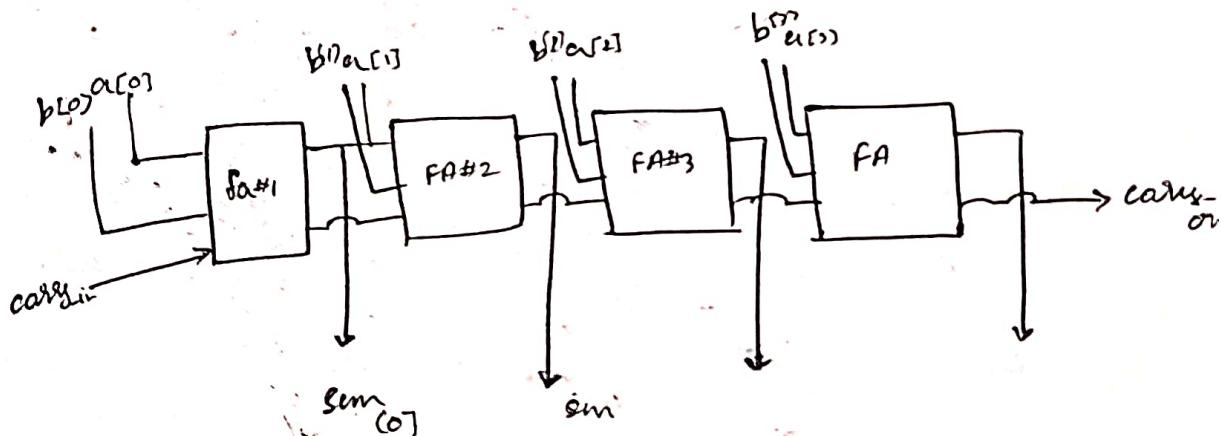
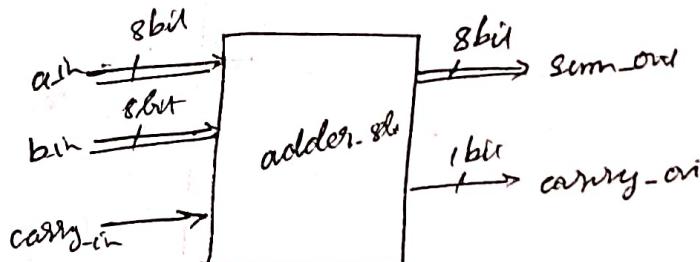


For more instance see

generate
genvar i;

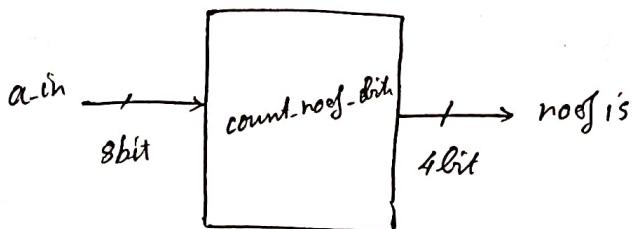
for C : ; ;

endgenerate



Exercise 65

RTL design to find no of 1's in ip use function & for loop



counter (a-in)

```
for (i=0; i<8; i=i+1)
begin
  if (ain[i])
    end count+1;
```

Evening session

Verilog 95 module declaration
function declaration

function <name>;	2
<post-assignment> (4p<0/p)	
endfunction	

```

module <name>(<parameter>);
  output ...;
  output ...;
endmodule
  
```

Task

task declaration

```
task <name of task> < arg, ip, op, inout >; }
```

....
....
....

Rules for a task

- task can have ip, op, inout arg
- task can enable another task or function.
- task will not, won't return any value
but, same result is achieved using output arg.
- task can consist of time control construct
(#, @, posedge, negedge, wait)
- we have static task, automatic task
↳ task is re-enterant.

Tcl → used in legacy system

puts

set <variable-name> <values>

for {set i >} { \$i < \$num } {incr i} {

[expr \$sum + \$i]

}

[format %c <var>]

task call inside always @*

Global task can be declared outside module

Exercise 67 RTL design of posedge D-ff having active low sync reset (reset_n) using static task as well as global task

10-10-2024

Exercise 68 RTL Design to find no of zeros from 8 bit ip using task

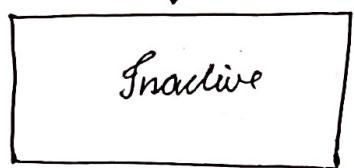
Stratified Event Queue

From previous time slot

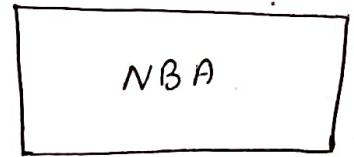


- continuous assignment
- \$display
- primitives →

and (conzine);
or (cp, ip);
not (cp, ip);
- BA
- RHS of NBA



- #0 delay assignments



- LHS of NBA



- \$monitor
- \$smon

To next time slot.

must 2 to 1 using primitive

full adder using primitive

→ with instance of half adder with primitive

Fork

...

Join

Evening session

* Static task vs automatic task

```

module tb_design;
initial
    print();
initial
    print();
:
task print();
    : for (current)
endtask
task automatic print();

```

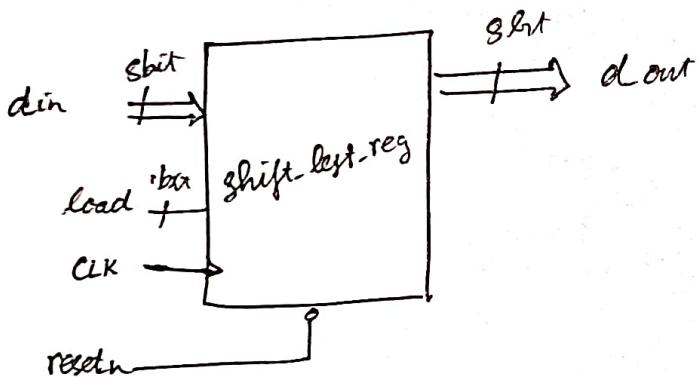
* Loops in design

→ for

Exercise 70 RTL design to have 8-bit shift 1-bit

1) without for loop

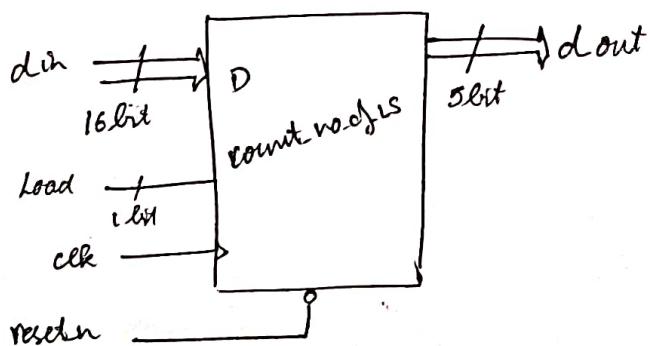
2) with for loop.



8 →

nrst
ceil(log₂(nof bit))

Exercise 71 RTL design to find no of 1's in 16 bit



11.10.2024

while loop

while (condition)

begin

end

→ carefull about condition

→ not advised to use loop in synthesis

infer cascaded logic, for combo design.

(e.g.) for loop.

for ever loop → infinite loop need with time controlled construct

always vs forever

always #5 clk = ~clk;

initial

begin

clk=0;

#250 \$finish;

end.

100MHz clk

2 timer assignment
of variable in 2
different procedural blocks.

non-deterministic
race condition

initial

begin

clk=0;

forever # 5 clk = ~clk;

end

initial

begin

#250 \$finish;

end.

multidriven net

→ if assigned twice in
always procedural block

intera & interdelay

→ difference in BA, NBA

fork intell begin

join

end

↳ all events concurrently.

Exercise 7c

fork, join (half adder → tb)

Evening session

Repeat loop

repeat (number)

begin

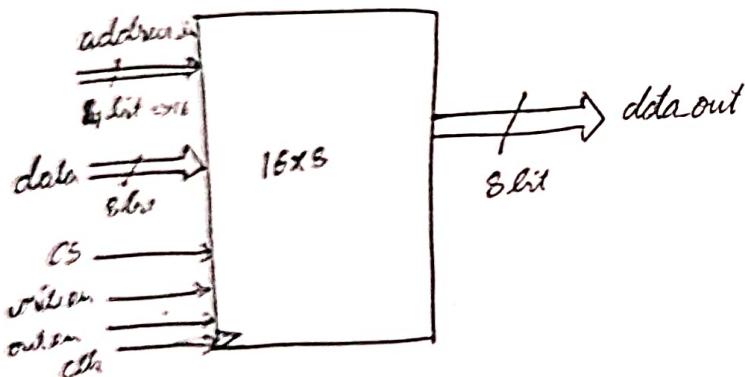
end

16 bit - 64 kB

32 bit - 4 GB

64 bit - 16 Exa B

ROM/RAM operation



CS	write-en	ctrl	operation
↑	0	1	0 → write to memory → off bypass
↑	1	0	1 → off per addr.
↑	0	x	x 000
↑	1	0	store mem. byp.

Memory modeling - consider 1, location of memory

ext to datain

address	2-bit	0
	2-bit	1
	2-bit	2
	2-bit	3

if data.out

else

reg[1:0] memory[0:3];

:

memory[address-in] <= data-in;

:

data-out <= memory[address-in];

Exercise 7

RAM 16x8bit table & i/p from above.

13-10-2024

Test bench for above

- Clk - period 10ns (ie) clock toggle for 5ns
- to write 16 memory location we need 160ns similarly to read.
- address to be incremented by '1' for every clock cycle (ii)
for every 10ns
- when select 'cs' atleast 320ns (ie) 160 for read & 160 for write
- following are test case.

```
always #5 clk = ~clk;
always #10 address_in = address_in + 1;
always #160 write_en = ~write_en;
always #400 cs = ~cs; // > 320
always #10 data_in = data_in + 1;
always #160 out_en = ~out_en;
```

initial

begin

data_in = 0

address_in = 0

write_en = 1

cs = 1

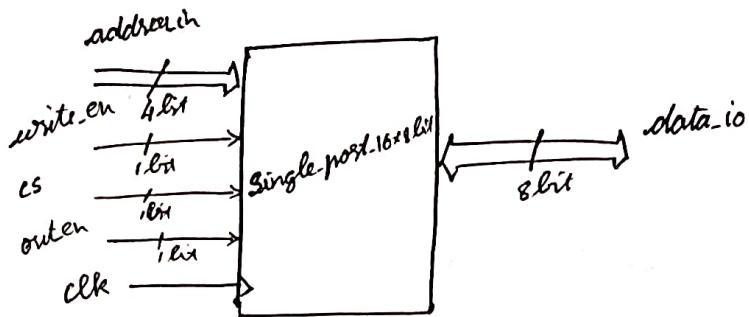
out_en = 0

clk = 0

end

Exercise 78

RTL Design of single port RAM 16x8 bit having bidirectional inout port



cs	write_en	out_en	operator
0	x	x	No operation
1	1	x	write to memory
1	0	0	
1	0	1	

```

wire data-io [7:0];
reg temp-data [7:0];
assign data-io = (!out_en & write_en & (S)) ? temp-data : 8'bzzzz-zzzz

```

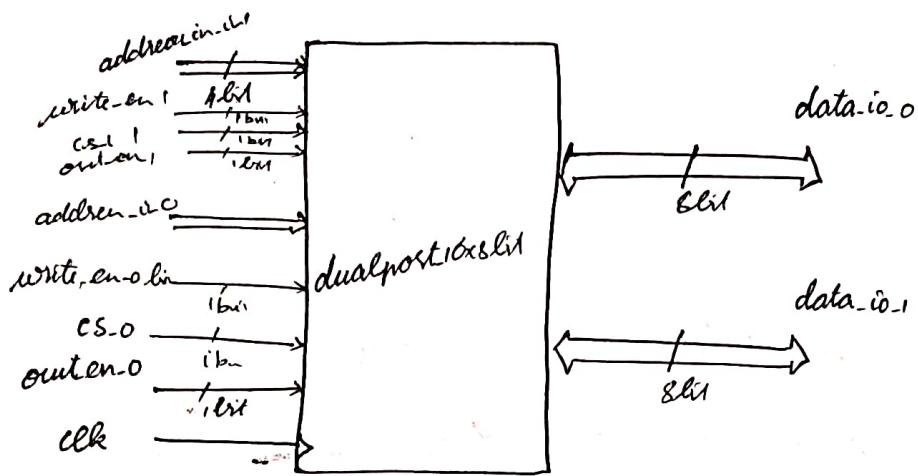
\$random

For inout data bus initially write a data in data memory
for that consider temp-data & assign that to data-io for write condition

13-10-2024

Exercise 79

RTL design of dual port RAM 16x8 bit having



Same operation as before.

Delay modeling

Functional verification → without delay

→ to check functional correctness

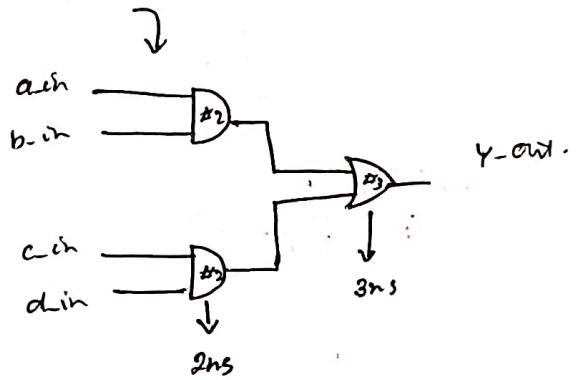
Delay modeling.

/

\

distributed

lumped delay.



→ delay distributed

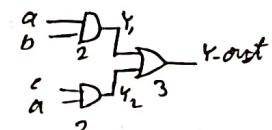
```
module delay_modeling_distributed(a.in, b.in, c.in, d.in,
                                  output r.out);
```

```
wire r, r1;
```

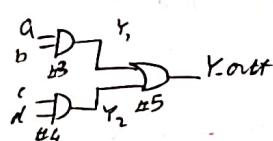
```
assign #2 r2 = a.in && b.in;
assign #2 r1 = c.in && d.in;
assign #3 r.out = r1|r2;
```

→ Functional verification is without delays. goal is to check functional correctness of design. But, in delay modeling we can expect output of gate after some delay. we have 2 types of delay 1. distributed delay & 2. lumped delay. // In distributed delay delays are specified per gate basis. they are distributed across design // When in lumped delays are specified as per model basis i.e. single delay on op gate.

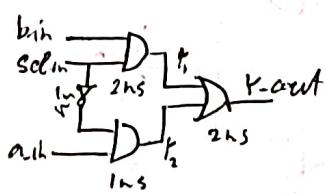
Exercise 80 ⇒ RTL design for distributed delay of



Exercise 81 ⇒ using gate primitive

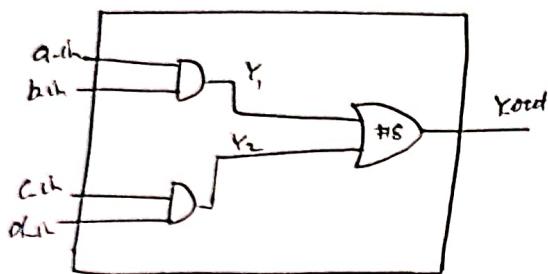


Exercise 81 ⇒ RTL design of 2:1 MUX



Lumped delay

There are module level delays (ie) delays are specified per module. $\approx 8 \text{ ns}$ is specified.

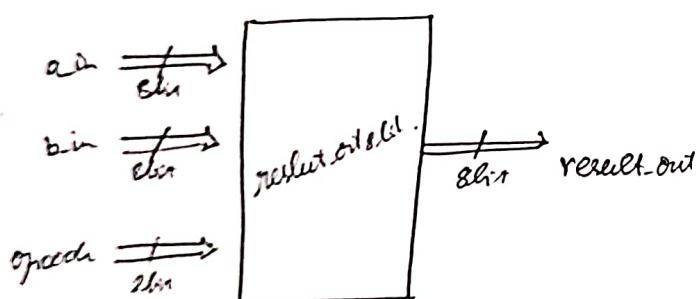


'define <name> values

'(name)



RTL design of 8-bit logic unit to perform following operation.



opcode	operation
00	NOT
01	X OR
10	AND
11	OR

Exercise 8.4
Consider lumped delay of module

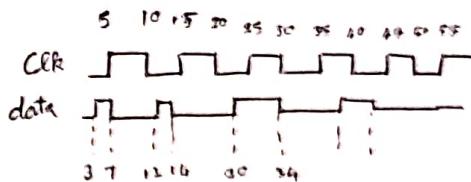
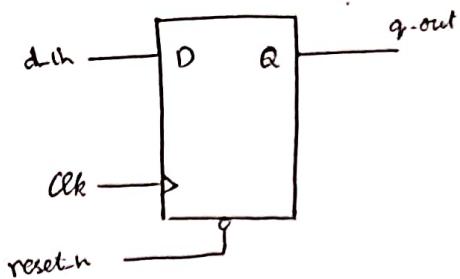
⇒ 'include " <name>.vh"
↳ verilog headerfiles

Timing check ⇒

\$# setup

\$# hold.

\$# width



assign D_i = din_i;

assign CLK_i = clk_i;

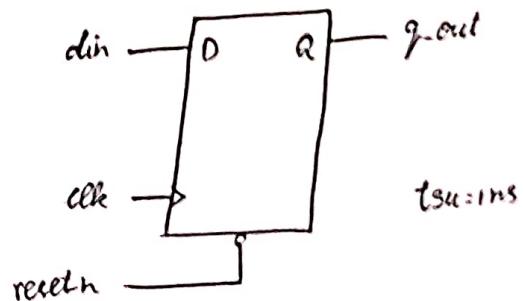
specify

specparam t_{su} = 2ns;

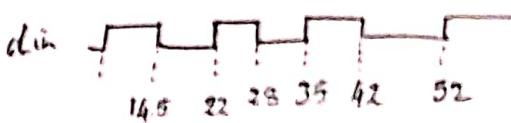
\$# setup (D_i, CLK_i, t_{su});

end specify

11.10.2024
use timing check setup & check for timing violation



\$# setup (data_change, reference, limit);



$\$setup$ in timing check system task all timing check must be within specify block.

data change is signal changes that is checked against reference.
reference is clk signal or any other signal that is used as reference.

limit is time between two events.

Violation is reported when

$$T_{reference} - T_{data\ change} < \text{limit}$$

in $\$setup \Rightarrow$ if data changes within given time limit it reports timing violation.

Following are scenarios it capture

\Rightarrow if data change & event occurs at the same time there is no-violation

$\$hold \Rightarrow$ it checks whether data is stable in specified interval of time after clock edge

\Rightarrow Violation reported if data event & reference event occurs at some time

if the following is true then tool will report

$$T_{data\ event} - T_{reference} < \text{limit}$$

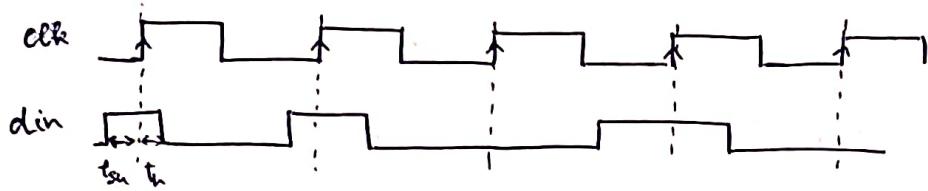
$\$setuphold(\text{reference_event}, \text{data_event}, \text{tol}, \text{th});$

22-10-2024

→ # width (reference, time limit)

↓
first transition of signal → min time req. between $T_1 \& T_2$

Violation occurs if $T_{ref2} - T_{ref1} < \text{limit}$



$$t_{su} = 1\text{ns}$$

$$t_h = 0.5\text{ns}$$

$$\Rightarrow t_{su} + t_h = 1 + 0.5 \\ = 1.5\text{ns}$$

width (posedge din, 1.5);

Queste

New project →
library name (work)

Limitation.

Code coverage report

create new file dff.v
tb-dff.v

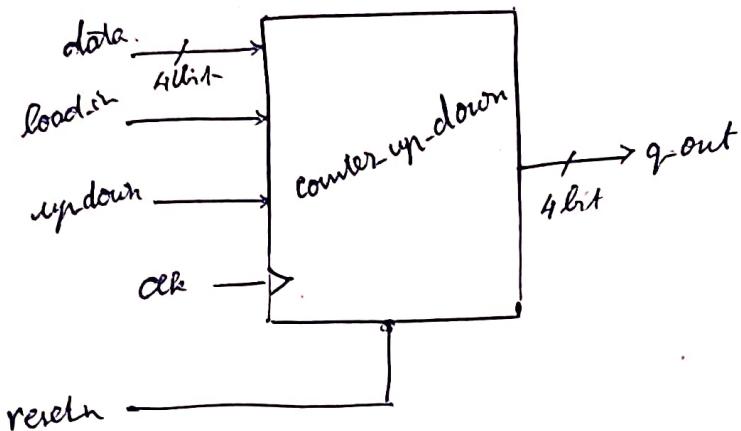
tools view code cov.

Select files and compile.

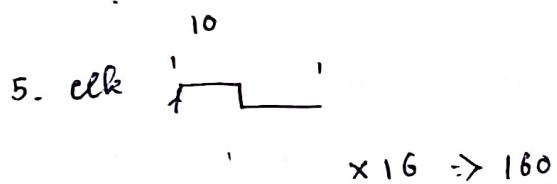
simulate
work

23-10-2024

RTL design of 4 bit binary up down counter & get 100% code coverage



reinth	clk	loadin	up-down	operation
0	x	x	x	$q_out \leq 0$
1	1	1	x	$q_out = din$
1	1	0	1	up counting
1	1	0	0	down counting



160 updown

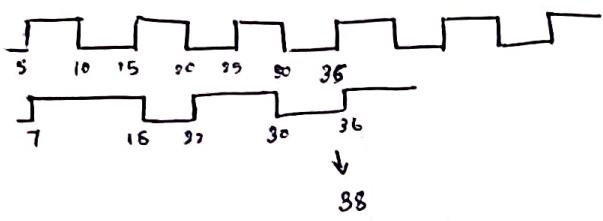
→ toggle coverage of data-in

`f skew (<signal to compare>, < >, tskew);`

It is used to check the timing violation depending on the arrival of active edge of clk. will know the clk skew or difference between arrival of active edge of clk if diff clk are used in a design which are not sync f skew will

report timing violation when active edge of clk occur outside the time limit allowed for the other clk. When both events will occur at the same time & skew will not report timing violation.

Tb generate clk₁, clk₂



```
wire D1;
wire E1;
assign D1 = din;
assign E1 = en;
specify
  $recoveron trecover = 2;
  $recover (posedge E1, D1, trecover);
end specify
```

Tomorrow to see

\$nochange \$period \$recovery

24-10-2024

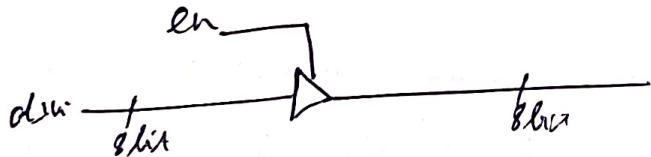
\$recover (posedge en, data, limit);

reference event - edge triggered (pos-edge or neg-edge)

→ violation if time interval between edge triggered reference exceed the lim, if both occurs at same time then violation reported

* NOTE: If reference event arg without edge a error is reported.

Consider tri-state buffer.



\$no_change (ref.event, data.event, start-time, end-time);



→ checks if data signal is stable in an interval of start time or start offset & end time or end offset. If signal has changed timing violation reported, reference event arrival can be pos-edge or neg-edge.

To understand this consider z-if and gate as shown in following

start edge offset $\Rightarrow t_{start} = -2 \quad t_{end} = 2$

\$period (ref.event, limit);



→ to check if period of event sufficiently long or not reference event is edge specific, data event is not specified directly. By defining as ref event, \$period report violation when time of ref event \sim in less limit.

Consider a buffer

25/10/2024

Conditional compilation

'ifdef

'else

:

'endif.

Consider PIP0 register without reset, that can be included by 'ifdef'.

```
'define INCLUDE_RST  
module PIP0_8bit #(input clk,  
    'ifdef INCLUDE_RST  
        input resetn,  
    'endif  
        input [7:0] din,  
        output [7:0] dout);  
    );
```

always @ (posedge clk)

begin

'ifdef INCLUDE_RST

if (!resetn)

begin

d_out <= 8'b0000_0000;

end

'endif

begin

d_out <= din;

end

end

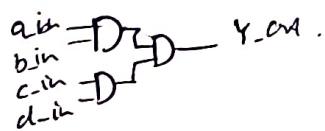
endmodule

Exercise 95

async reset 16 bit PIP0

Exercise 96

'define synThesis.



'if' nodes have exactly one

26-10-2024

Generate block

→ no specify
specparam.

Multiply module instantiation

- Verilog construct, generate block which allow to multiply module
i.e. to perform conditional instantiation of any module.
- generate block cannot contain
port, parameters, specparam, specify block.
- all generate instance are written within a module
within generate, ... endgenerate. Generated instances can have
assignment, module, always or initial block & user define
primitive

Generate construct are of two types loop using generate
for loop. Conditional construct generate if-else or case.
Loop variable can be declared using key word genvar.
which gives information to compiler or tool that
variable to be used during elaboration of generate
block

Application consider replication of instances of AND gate
N times. Generate for loop can be used.

a.in
b.in - D - y.out

S D

Code coverage report → Expression coverage Branch coverage
Statement coverage FSM coverage
Toggle coverage

Exercise 98

PSPO.sbit generate diff

Exercise 99

full adder n-bit

conditional generate

generate if → a name indicate we of if else inside generate constraint & objective is to select between two different implementations (e.g. 2*o/p* NAND based implementation or 2*o/p* NOR based implementation)

in top module a parameter NAND-~~SELECT~~^{SELECT}=1 to select a choice of 2*o/p* NAND consider the following design

Exercise 100 parameter NAND-sel=1;

Generate

of (NAND.SEL)

design_and y, (a.in, b.in, y, a1)

case ()

else

design_nor y, (a.in, b.in, y, a1)

end case

Seleno LD_LIBRARY_PATH /home/stud1/greentrial

vlib work

design

vlog -work workv

...v

tb

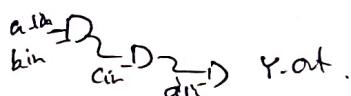
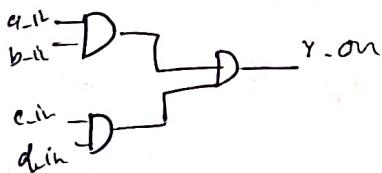
... .v

runsim

vsim -c -quiet work.<modulename> -do "run -all; quit"

Excerin 103

In need to choose a instance of a module depending on
the expression. e.g. consider 2 i/p NAND-gate, 2 i/p NOR gate & if
gate-type = 1 NAND instance should be selected otherwise NOR
instance. In such scenario we use generate case.



Excerin 104

diff w/ diff & diff res.

Excerin 105



if else
case.

already discussed about inbuilt primitive like and, or, not, nand,
but not always useful in complex logic. So, to model
complex logic we may have to use VDP (user defined primitive)
we can have combinational or seq VDP.

- 1. All VDP should have exactly 'i' op can be 0, 1, x
never Z
- 2. Verilog VDP are written in same level in module definition
but never between module .. endmodule

2. VDP can have many i/p ports. Bidirectional ports are not
allowed. All port signals should ^{be} scalar type.

Hardware behaviour is described in primitive state table
which consists of various combinator i/p & output within
table ... endtable.

Cfg for combinational VDP following can be various value
0 - logic 0, 1 - logic 1, x unknown either 0 or 1, used as
i/p or o/p, ? -> logic 0, 1, or x cannot be o/p of VDP. Some other
inbuilt gate level primitive and where single op & multiple
i/p
and (4-out, a-in, b-in)

Combinational VDP of and we can model using following

simulate it & instantiation in test bench in same as model instantiation.

primitive and primitive (Y-out, a-in, b-in);

output Y-out;

input a-in, b-in;

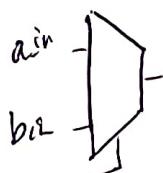
table

a-in	b-in	:	Y-out
0	0	:	0;
0	1	:	0;
1	0	:	0;
1	1	:	1;

endtable

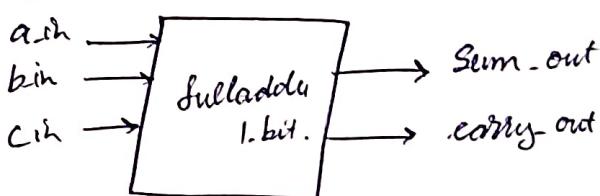
end-primitive

Exercice 107

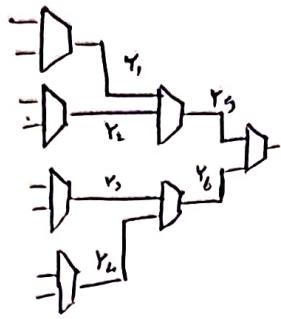
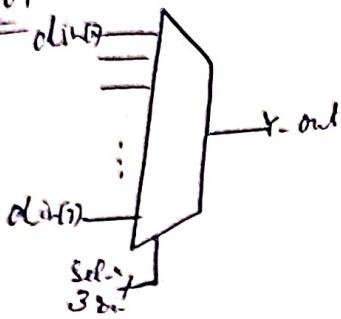


sel	a-in	b-in	Y.out
0	1	?	: 1 ;
0	0	?	: 0 ;
1	?	0	: 0 ;
1	?	1	: 1 ;

21-10-2024 Design full adder 1-bit using combinational CD P



Exercise 109



Sequential VDP

Luenkenheiter

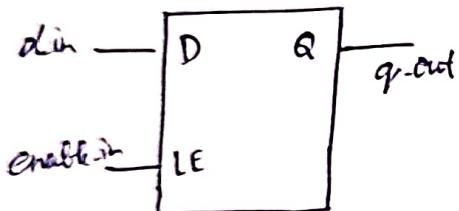
• Latch

Edge sensitive

- Shift flop
- Counter

Exercise 110

Positive Luenkenheiter



primitive latch-primitive

// enablein din : q-out : qout#
table

end table
end primitive.

* some as ?? indicate any change in ip

- at o/p only indicates no change

↑ rising (0→1)

↓ falling edge (1→0)

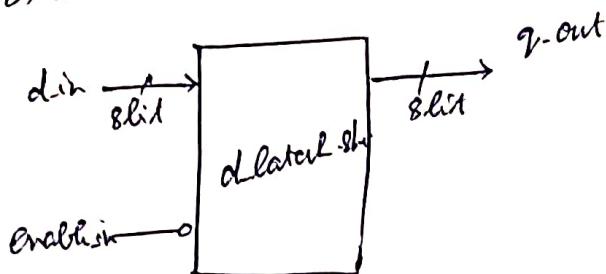
(ab) → a table

(10) → transition from 1 to 0

(01) → transition from 0 to 1

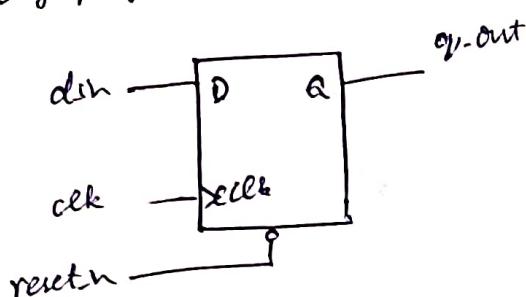
Exercise 11

8-bit D latch having active low enable



Exercise 11²

D flip-flop having active low asyne reset.



Exercise 11³.
VDP of negative edge sensitive dff having asyne active low reset

complete tb & get 100% code coverage.

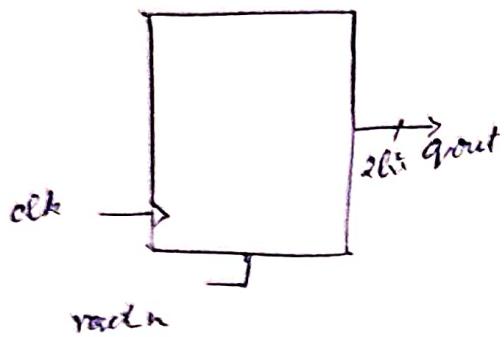
Exercise 11⁴
Modify rising edge sensitive dff (Exercise 11²) to have sync-active
low reset-n complete tb & get 100% code coverage

Exercise 11⁵
using VOP sequential of rising edge sensitive D-ff complete
8-bit PIPPO register. consider asyne reset-n complete

28-10-2024

Exercise 11b

asynchronous 2-bit up counter

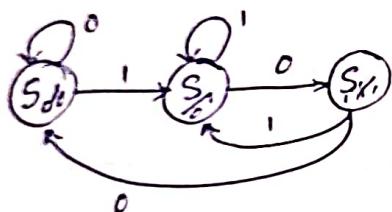


Moore sequence detector

$$\begin{array}{c} \text{Din} = 0100001010 \\ \downarrow \quad \downarrow \\ 2 \quad 2 \end{array}$$

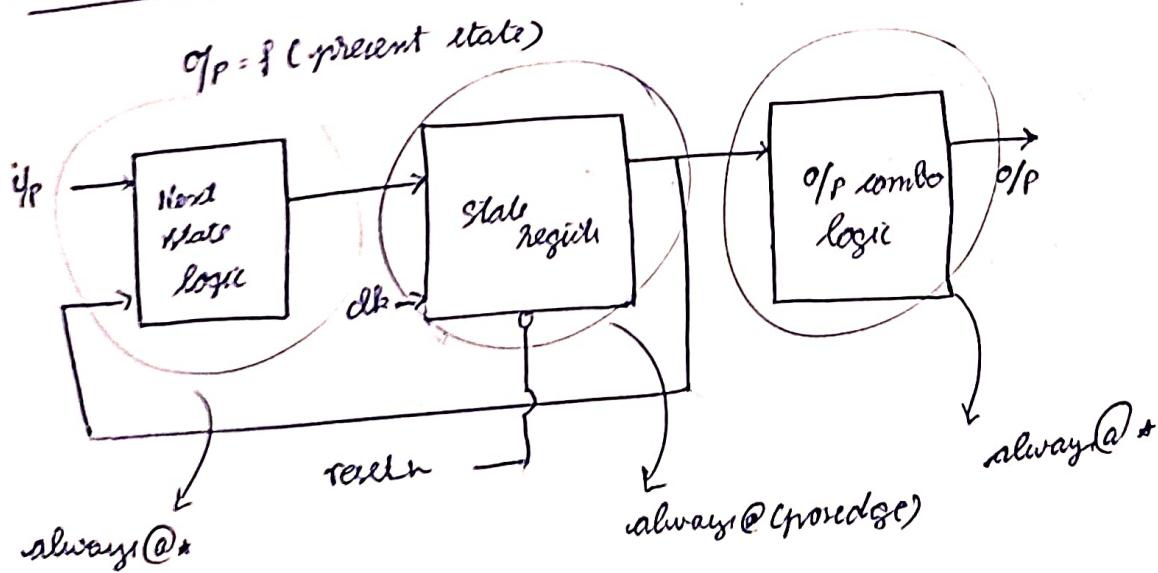
$\text{qOut} = 0010001001$

10 sequence detector



Mealy sequence detector

Mealy sequence detector



FSM sequence detect using verilog. 10

parameters $S_0 = 2^b000$

$S_1 = 2^b01$;

$S_2 = 2^b10$;

reg [1:0] present-state, next-state;

//State register

always @ (posedge clk, negedge reset-n)

begin

if (reset-n)

present-state <= 0;

else

present-state <= next-state;

end

//next state logic

always @ *

begin

case (present-state)

S_0 : if (din) next-state = S_1 ;
else S_0 ;

S_1 : if (!din) next-state = S_2 ;
else S_1 ;

S_2 : if (!din) next-state = S_0 ;
else S_1 ;

default: next-state = S_0 ;

//All will not all binary of met

endcase

//o/p logic

always @ (present-state)

begin

case (present-state)

S_0 : q_out = 1'b0;

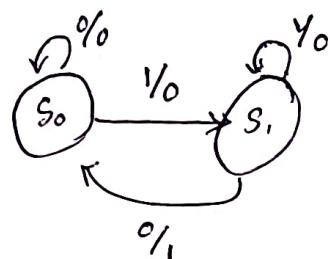
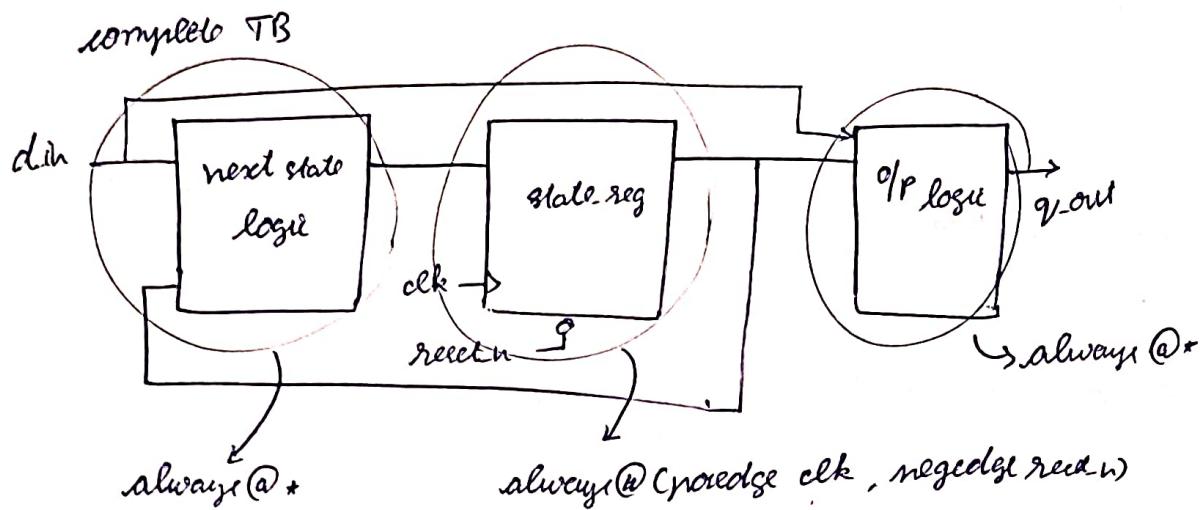
S_1 : q_out = 1'b0;

S_2 : q_out = 1'b0;

default q_out = 1'b0;

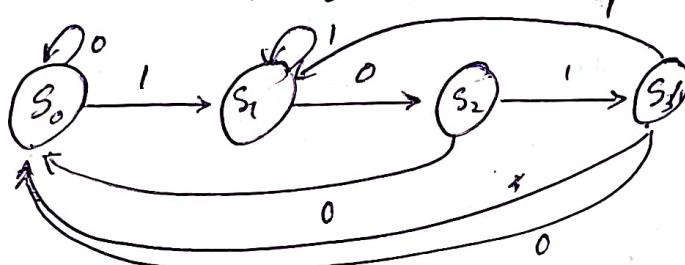
Exercise 117

RTL design sequence detector to detect 10 sequence



Exercise 118

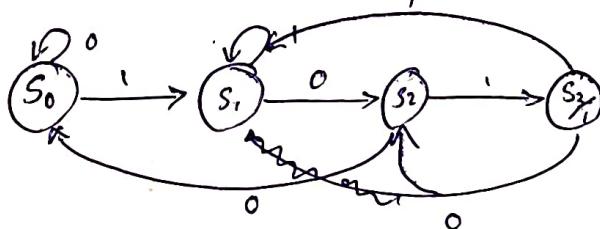
RTL design 101 non overlapping sequence detector. moore



29-10-2024

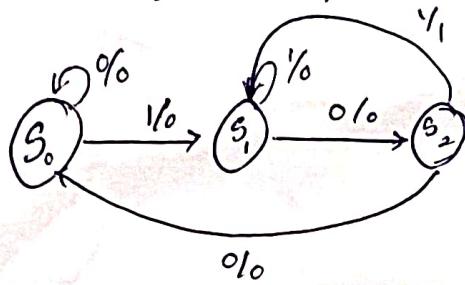
Exercise 119

RTL Design of moore overlapping sequence detector 101



Exercise 120

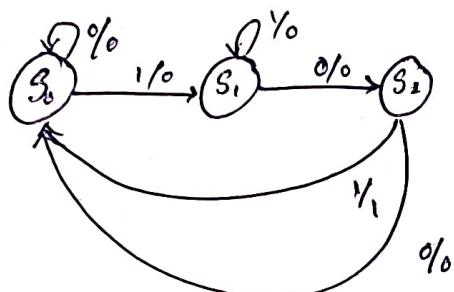
RTL Design of Mealy overlapping sequence detector to detect 101



Exercise 121

RTL design

101 mealy non-overlapping sequence detector.



→ Exercise 122

RTL design of mealy overlapping 10101

Exercise 123

RTL

design of mealy overlapping 101 sequence detector.

→ gray encoders

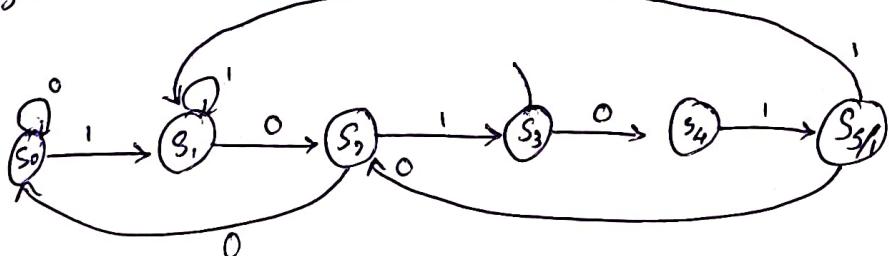
Exercise 124

RTL

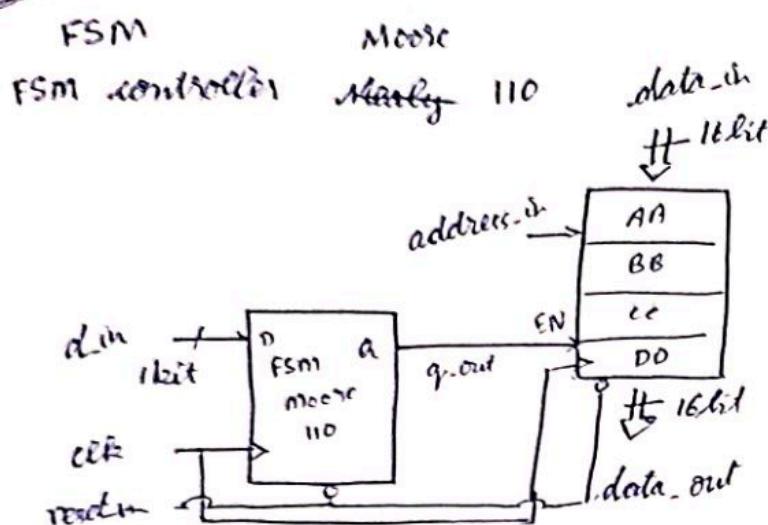
design moore one-bit overlapping

→ one-hot encoding

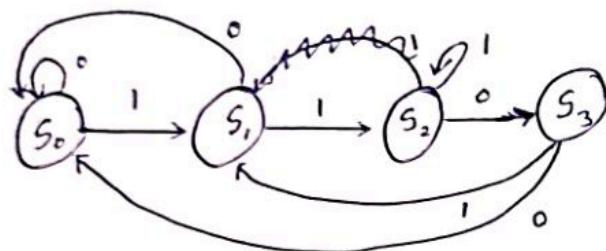
10101



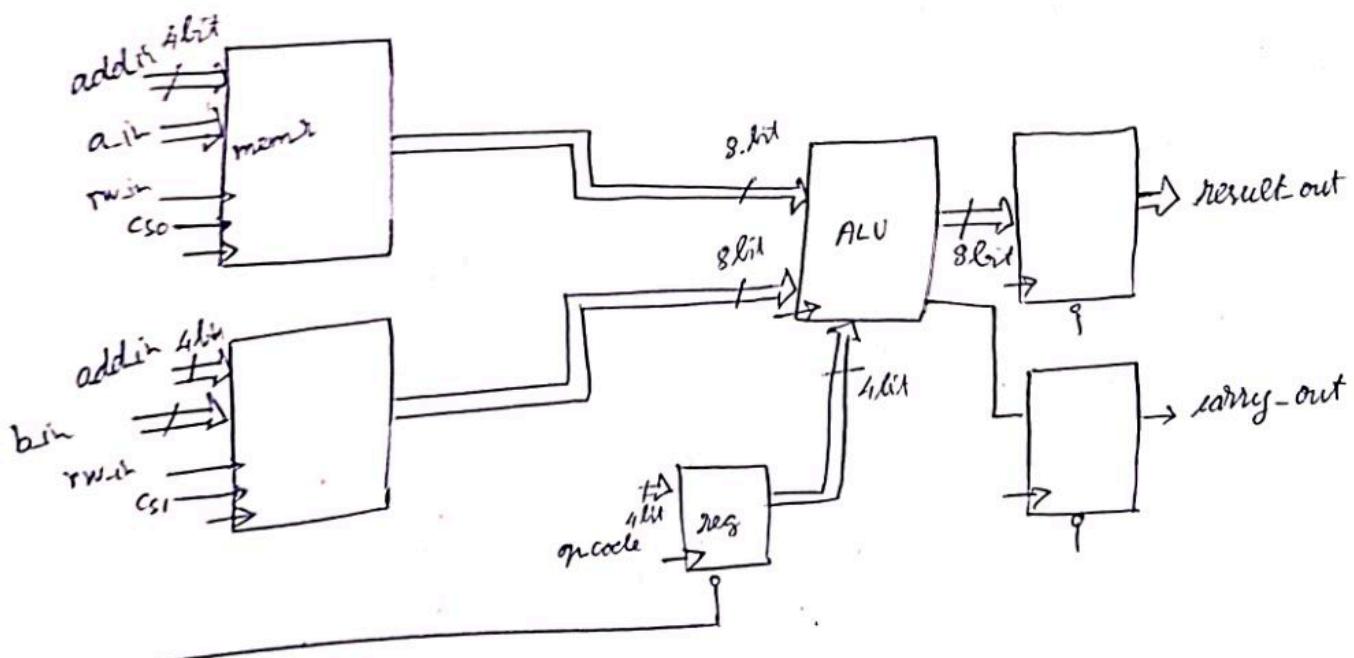
04-11-2024



data-in
data-out



Complex examples and RTL design

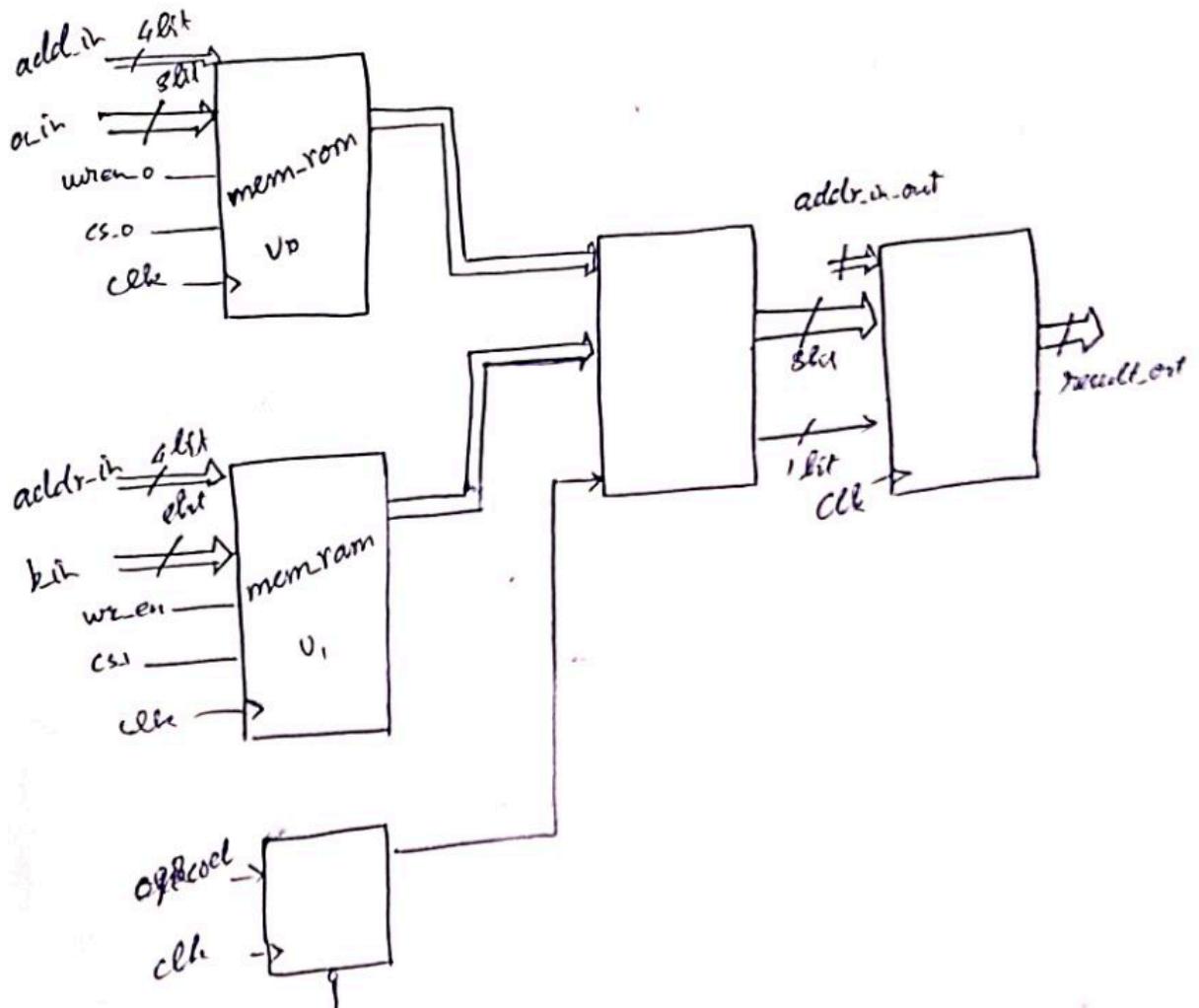


memory

CS	wr-en	operation
1	1	write
1	0	read
0	x	No operation

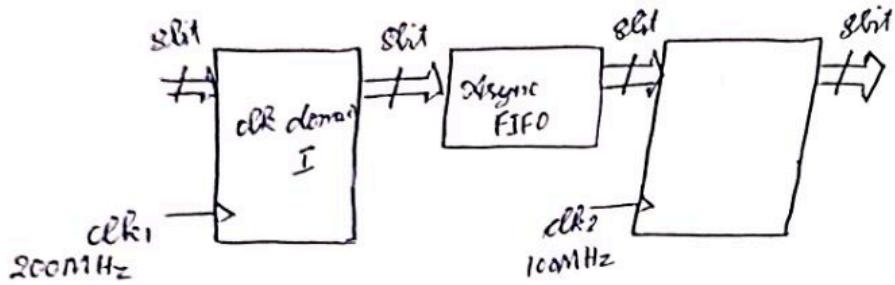
opcode	operation
0000	Transfer a-in
0001	a-in + b-in
0010	a-in + b-in + 1
0011	a-in - b-in
0100	a-in - b-in - 1
0101	a-in + 1
0110	a-in - 1
0111	Transfer b-in
1000	a-in OR b-in
1001	a-in XOR b-in
1010	a-in AND b-in
1011	NOT of a-in

Exercise 131



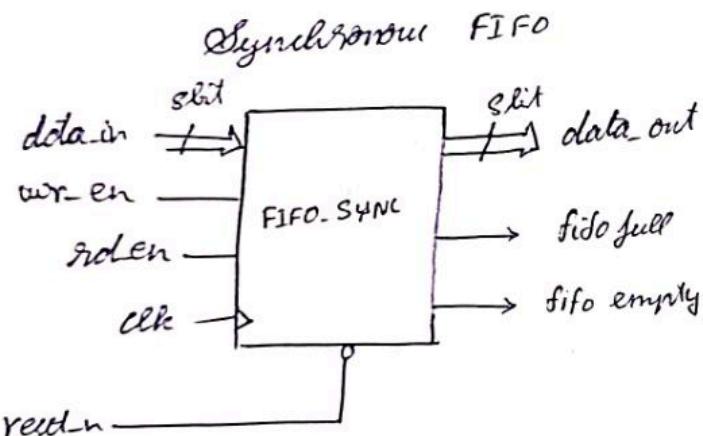
05-10-2024

FIFO design

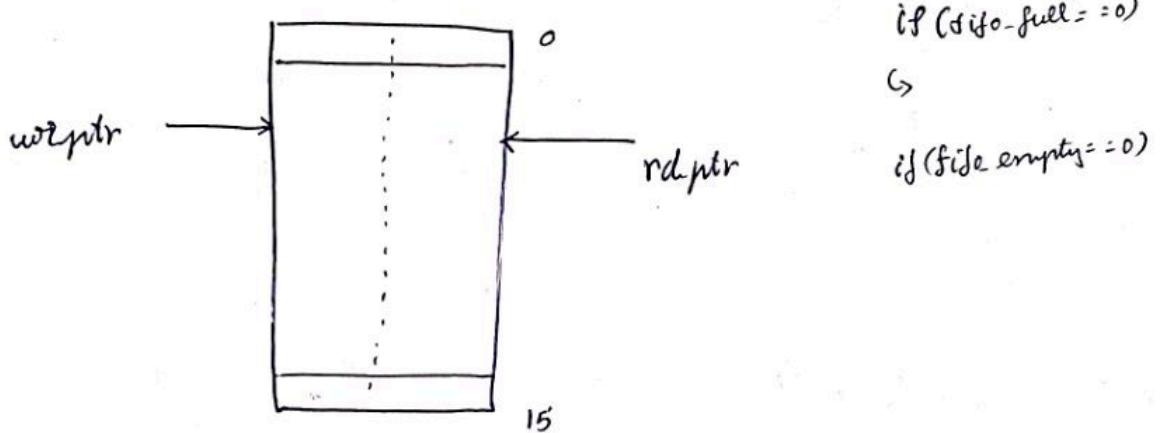


In multiple clock domain design

\begin{array}{c}
 \text{FIFO} \\
 \swarrow \quad \searrow \\
 \text{Asynchronous} \quad \text{Synchronous}
 \end{array}



1. Design the memory 16×8 bit & develope rd.ptr & wr ptr logic
2. Develop a logic for FIFO full, FIFO empty
3. Complete RTL Design
4. Complete TB
5. 100% code coverage.



if (fifo_full == 0)

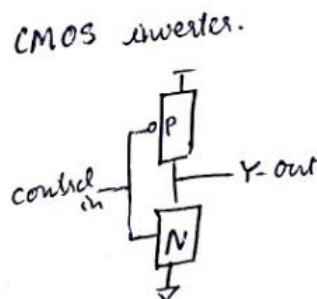
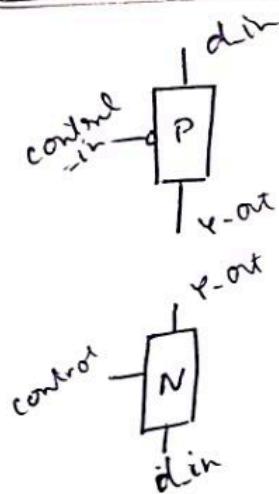
↳

if (file_empty == 0)

\$close();

06-11-2024

Switch level modeling



p-mos & n-mos switcher can be used in switch level modelling

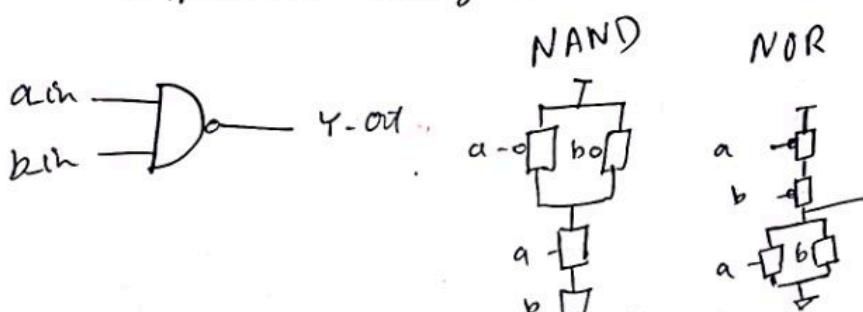
n-mos can be used to pass strong 0, pmos to pass strong 1

pmos v1 (y-out, d-in, control-in);

nmos v2 (y-out, d-in, control-n);

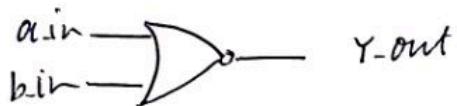
Exc 133

Implement 2 i/p NAND using switch level modeling

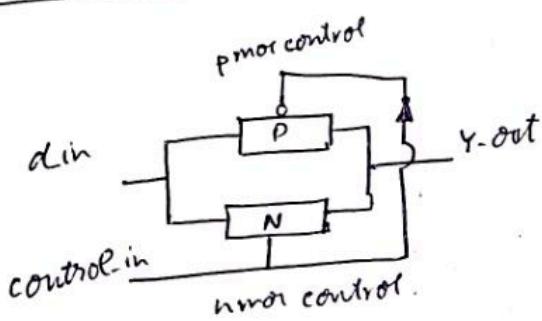


Exc 134

Implement 2 i/p NOR using switch level modeling

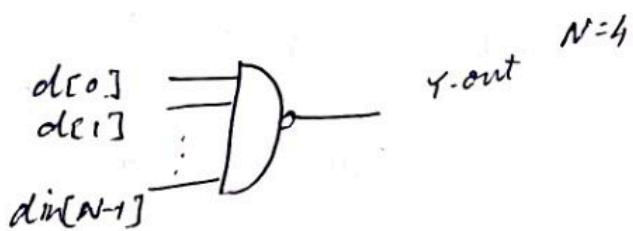


CMOS switch



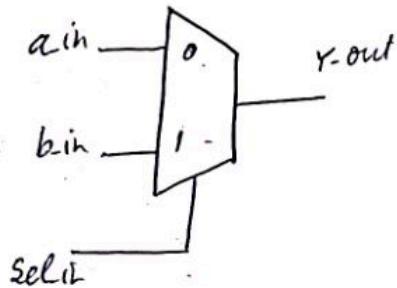
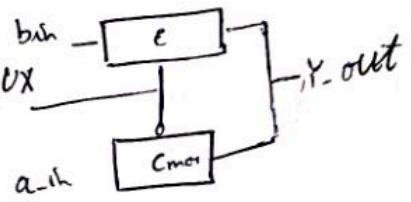
Switch level modeling of CMOS

Exercise 136 Implement N i/p NAND using switch level modeling



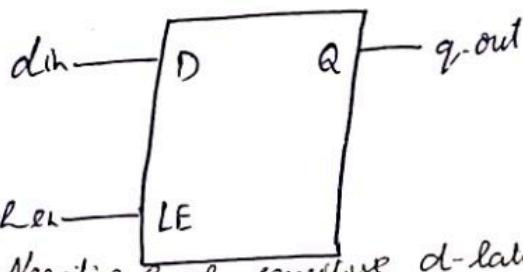
06-10-2024

Exercise 138 switch level modeling of 2:1 MUX



Sel-in	Y-out
0	a-bin
1	b-bin

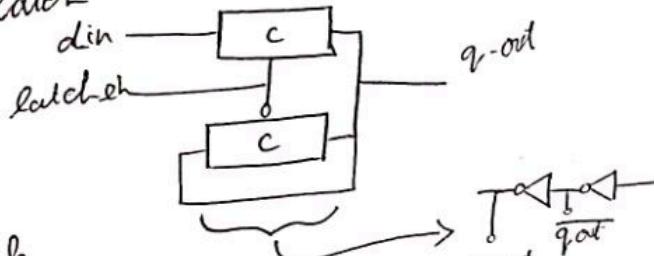
Exercise 139 switch level modeling of D-latch positive level sensitive



latchen	q-out
0	No change
1	d-in

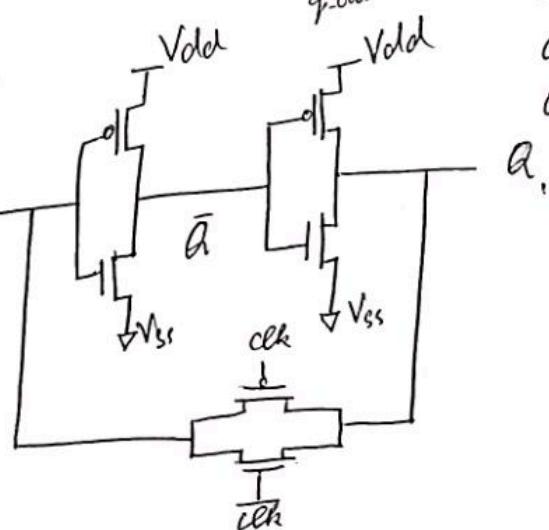
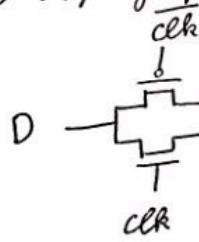
Exercise 140 → Negative level sensitive d-latch.

D-latch



Homework

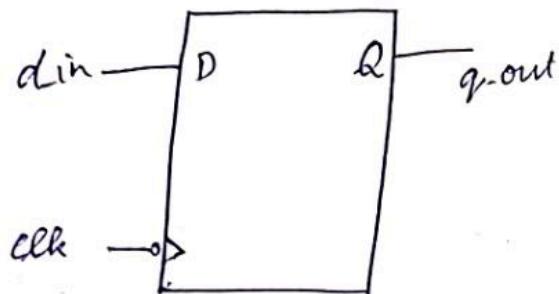
D-Slip-flip-



data-input - d-in
clock - clk
output - q-out

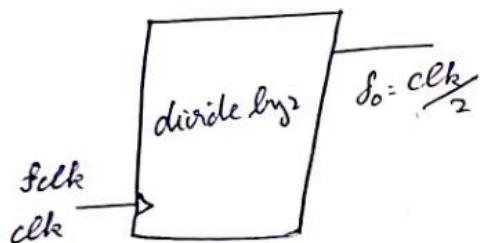
07-11-2024

Exercise 141 Switch level modeling of negative edge sensitive D flip flop.

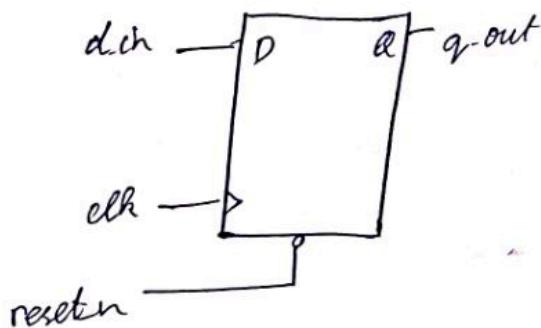


clk	q-out
+	d-in
-	No change

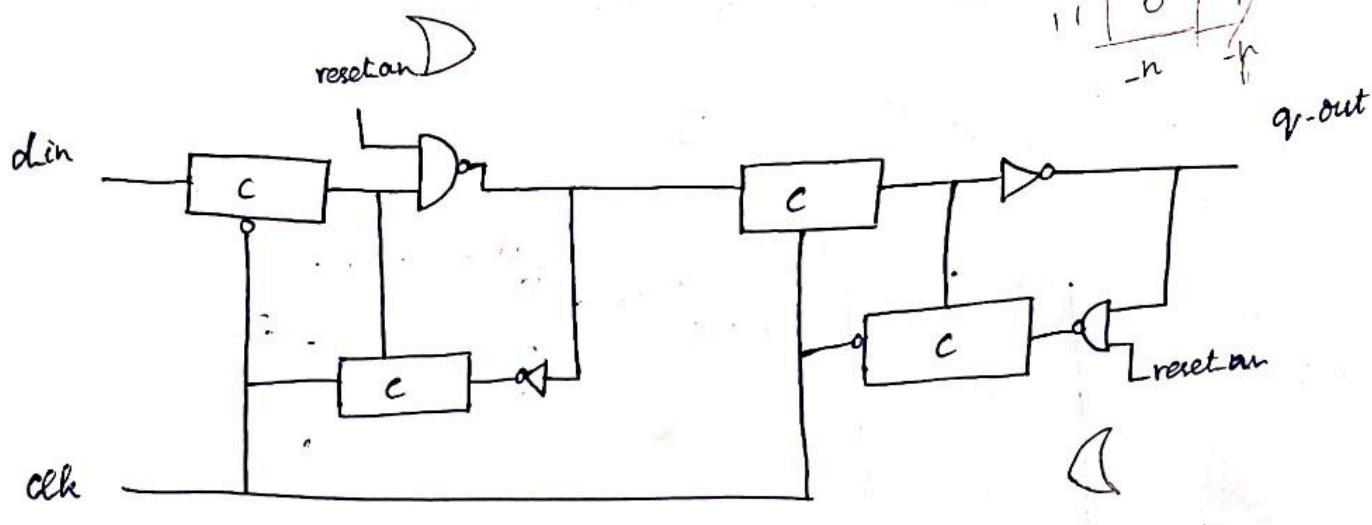
Exercise 142 CMOS switch level modeling



Exercise 143 Include active low asynchronous reset in positive edge triggered D flip flop.



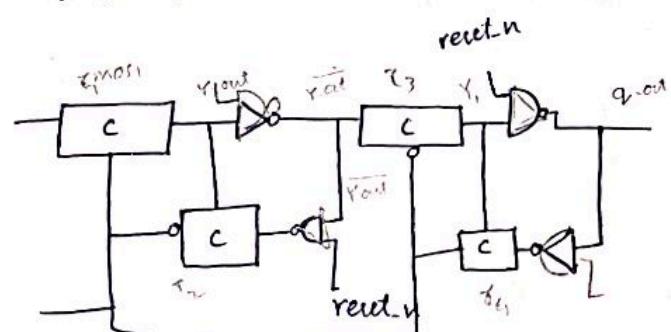
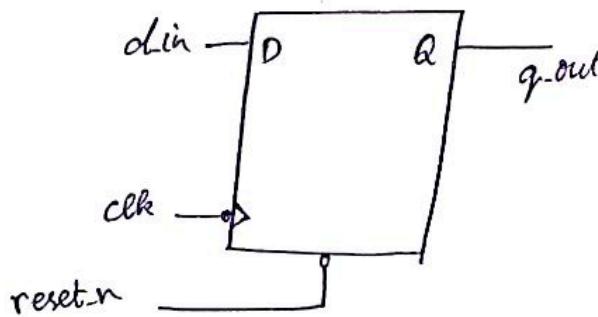
A	B	$\overline{A \cdot B}$	$\overline{A} \cdot \overline{B}$
0	0	1	1
0	1	1	0
1	0	1	0
1	1	0	1



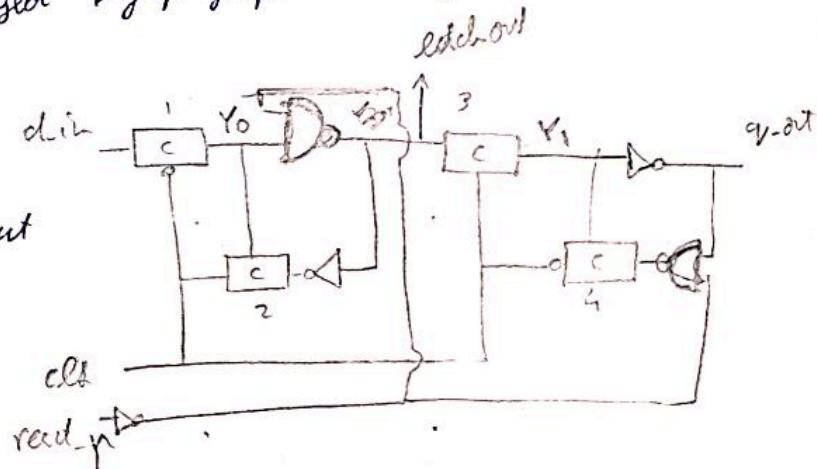
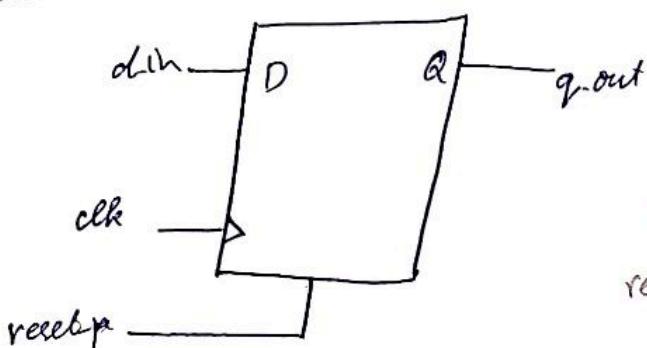
Synchronous reset

08-11-2024

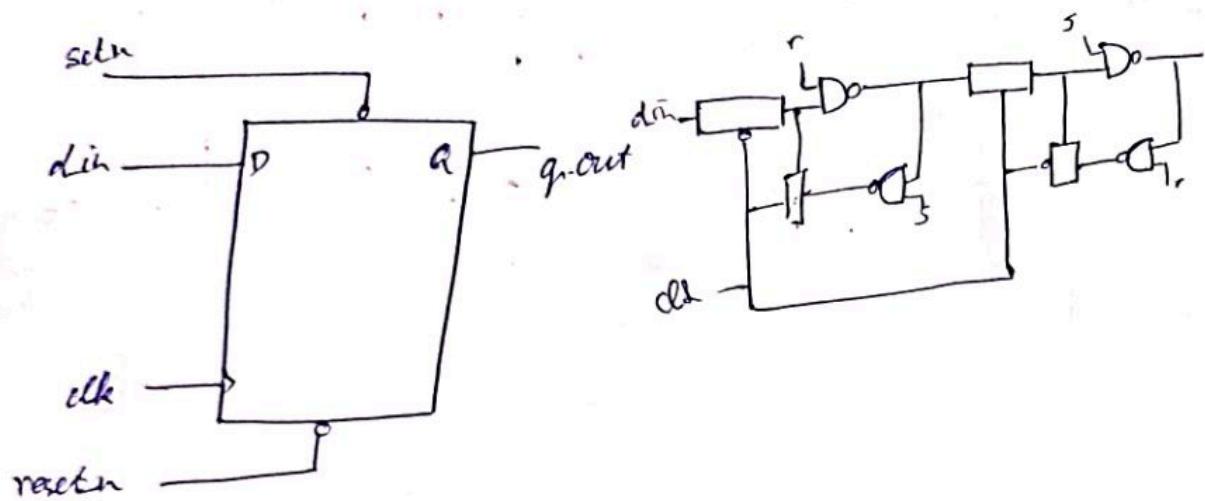
Exercise 144 → negative edge triggered D flip flop with active low reset-n



Exercise 145 → positive edge triggered D flip flop having active high asynchronous reset

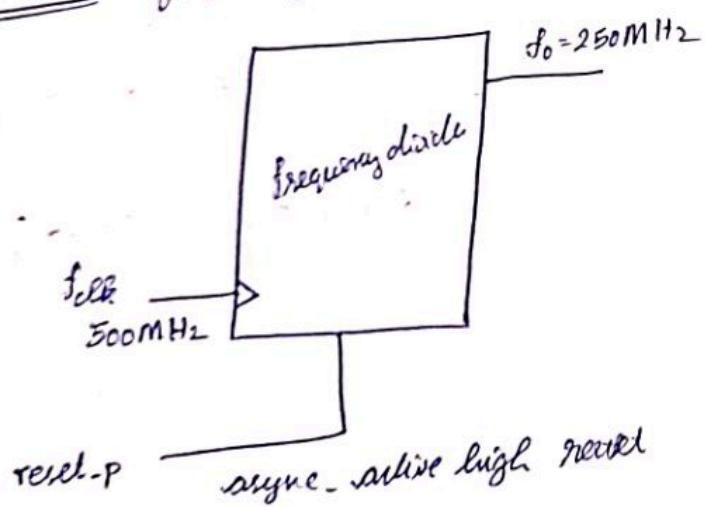


Exercise 146 positive edge triggered Dff having sync-active low reset & asynd active low set h

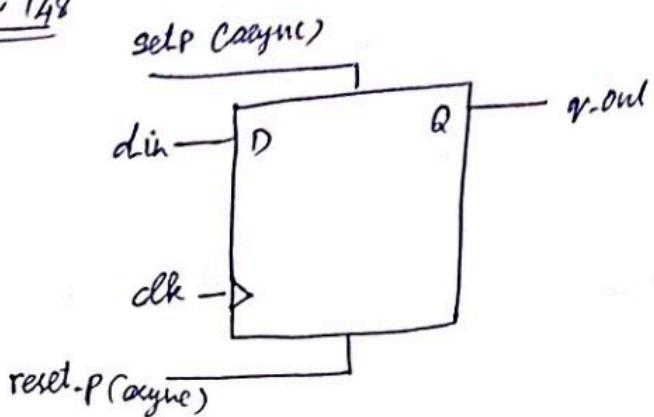


→ Events Session.

Exercise 147 frequency divider



Exercise 148



09-11-2024

Application of Verilog constructs

→ Inter delay assignment

#5 a = 0;

#10 a <= 0;

$\overline{\downarrow}$
LHS side

→ Inter delay assignment

a = #5 0;

b = #10 1;

$\overline{\downarrow}$
RHS

Difference between Blocking assignment
Non-Blocking assignment
in Test Benel.

Delay assignment

Verilog has two type of delay assignment inter delay assignment & inter delay assignment

Inter delay - delay value on LHS of operator, which indicates the statement is executed after delay exprn.

e.g. Consider initial procedural block having inter

delay assignment (mostly used). These assignment will

execute when delay expires. →

initial begin

#5 a = 0;

#10 a <= 0;

end

Inter delay.

In this delay is on RHS of operator. Here statement is evaluated and values of all signals on RHS are captured first. Then assigned to resultant signal after delay expires.

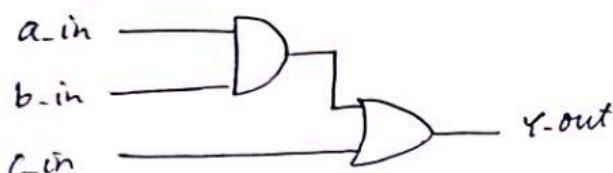
(eg) initial begin

a = #5 0;

a <= #10 1;

end

RTL design



t	a-in	b-in	c-in	y-out
0	0	0	0	0
10	1	0	1	0
15	1	0	1	1

Testbench NBA

a-in <= 0, b-in <= 0, ac-in <= 0;

#10 a-in <= 1, b-in <= 1, c-in <= 1;

#50 \$finish;

Both NBA & BA give same delay

Consider simulation source. having inter-delay assignment with NBA within initial procedural block checking for y-out

```
module beh-design;
```

```
reg a-in;  
reg b-in;  
reg c-in;  
reg Y-out;
```

```
initial begin
```

```
$monitor C ...
```

```
a-in <= 0; b-in <= 0 c-in <= 0
```

```
#10 a-in <= 1; c-in <= 1
```

```
#5 Y-out <= (a-in & b-in) | c-in;
```

```
#60 $finish;
```

```
end
```

```
endmodule
```

t	a-in	b-in	c-in	Y-out
0	0	0	0	0
10	1	0	0	0
15	1	0	1	1

Index assignment & inter assignment delay & NBA in a Test Bench

↳ $Y\text{-out} \leftarrow \boxed{\#5} (a\text{-in} \& b\text{-in}) | c\text{-in}$

t	a-in	b-in	c-in	Y-out
0	0	0	0	x
10	1	0	0	x
15	1	0	1	0

```
#5 a-in <= 1;
```

$c\text{-in} = 1$ ↗ NBA
 $c\text{-in} = 0$ ↗ BA

t	a-in	b-in	c-in	Y-out
0	0	0	0	0
10	1	0	0	0
15	1	0	1	1

#5 a-in <= 0; b-in <= 0; c-in <= 0; Y-out <= 0

#10 a-in = 1; b-in = 0; c-in = 0; Y-out = 0

#25 a-in <= 0; b-in = 1; c-in = 1; Y-out = 0

a-in = 1; b-in = 0; c-in = 1; Y-out = 1

#50

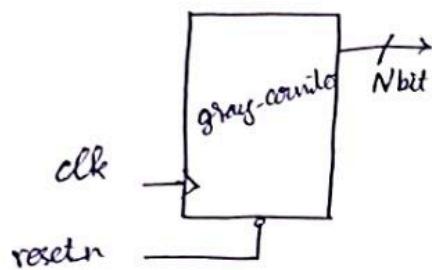
fork

:

join

a-in	b-in	c-in	y-out
0	0		0
5	1		0
10	0		1
25	1		1

Exercise 149 RTL Design of N bit gray counter, use sync active low resetn



Complete TB

Have 100% Code coverage.

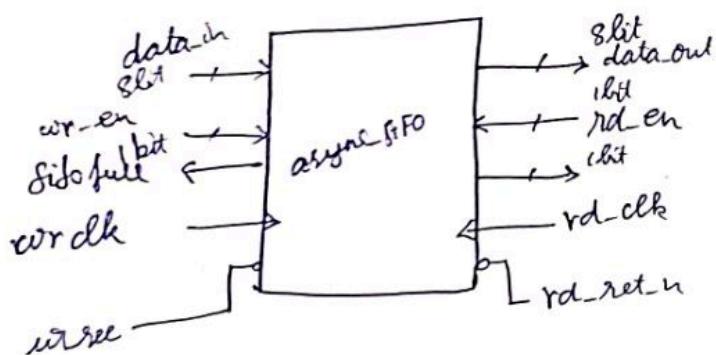
Application Multiple Clock domain data path

$$\text{wr-clk} = 100 \text{ MHz}$$

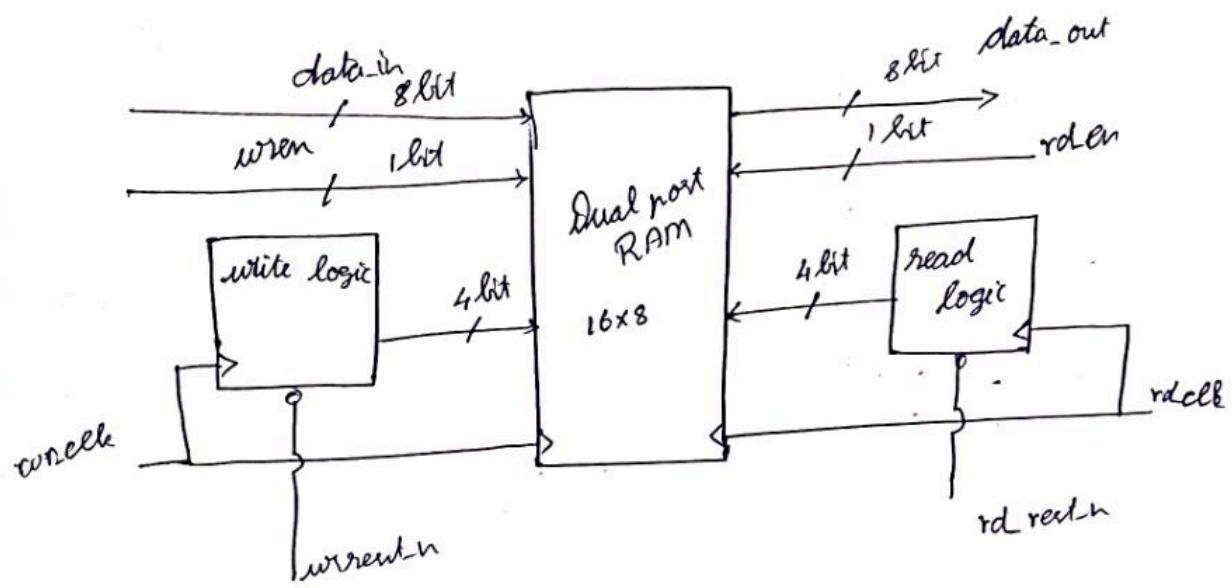
$$\text{rd-clk} = 50 \text{ MHz}$$

$$\text{Burst size} = 32 \text{ byte}$$

$$\text{Depth of FIFO} = ? \Rightarrow 16 \text{ byte}$$

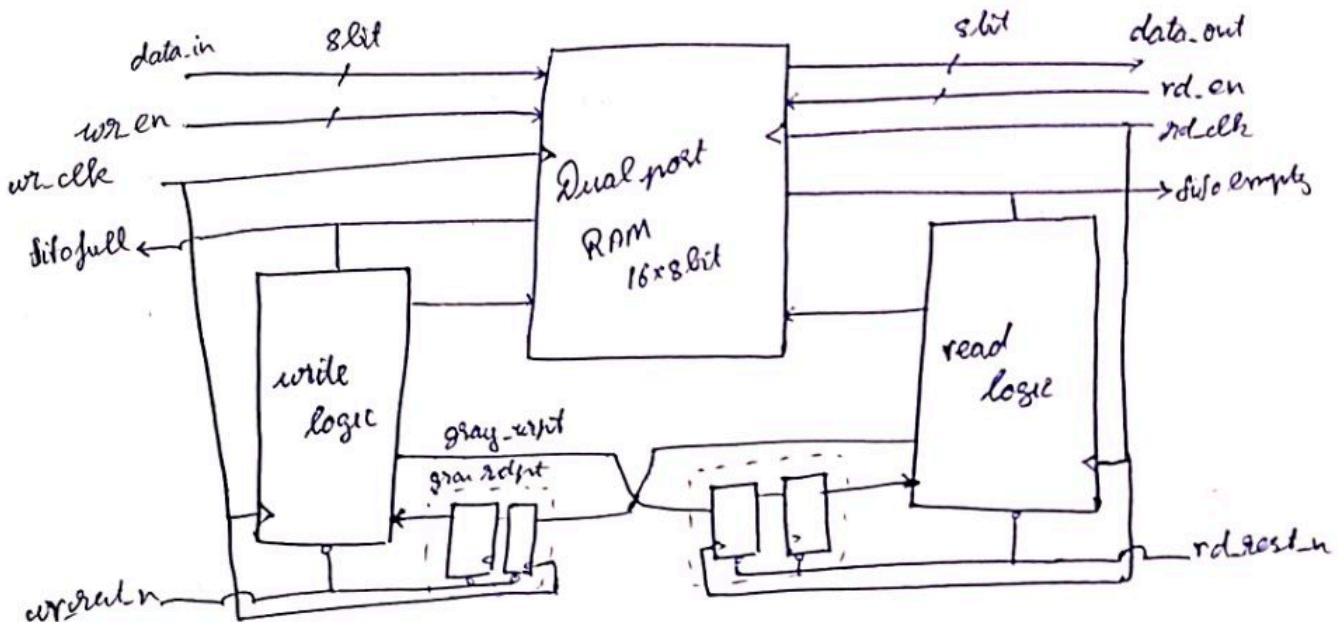


- Dual port RAM
- Synchronous
- write logic
- read logic



FIFO full
 $\text{gray_wr_ptr} := \{\neg(\text{gray_rd_ptr}[\text{WIDTH:WIDTH-1}], \text{gray_rd_ptr}[\text{WIDTH-2:0}]\}$

FIFO empty
 $\text{gray_wr_ptr} := \text{gray_rd_ptr}$



11-11-2024

Three variants of race

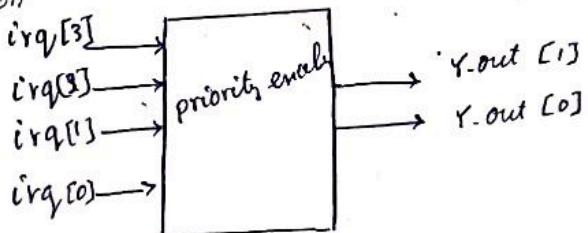
case
case X
case Z

→ X propagation.

$2^b1Z =$

Priority encoder

Higher Priority



Exercise 152

↳ Priority encoder using nested if else

Exercise 153

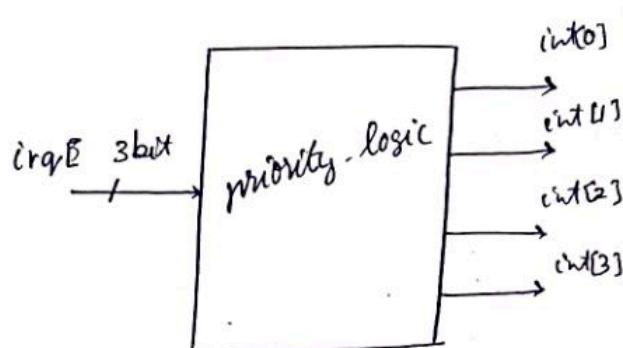
... case X

Exercise 154

... case Z

Verilog defines 3 version of race statement case, casec, & casez
case X it allows $Z, \text{?}$ to be treated as a don't care
 in either race expression or race item. But, in race X
 statement X propagation can cause problem. Hence
 not advised to use in synthesizable design.
case Z in $Z ; Z\text{?}$ to be treated as don't care
 in either race statement or race item is $2^b1Z = 2^b1?$
 $2^b10 = 2^b2$
 2^b11
 2^b1X

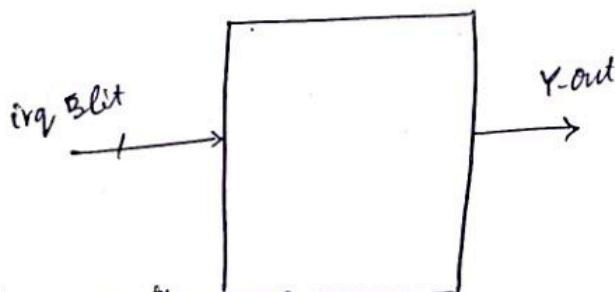
In this if more than one expression matches the case item the first matching case is taken. \rightarrow By using this we can infer a priority logic.



casez (irq)
 $3'b1?? : \text{out}[2] = 1'b1;$
 $3'b?1? : \text{out}[1] = 1'b1;$
 $3'b??0 : \text{out}[0] = 1'b1;$
endcase

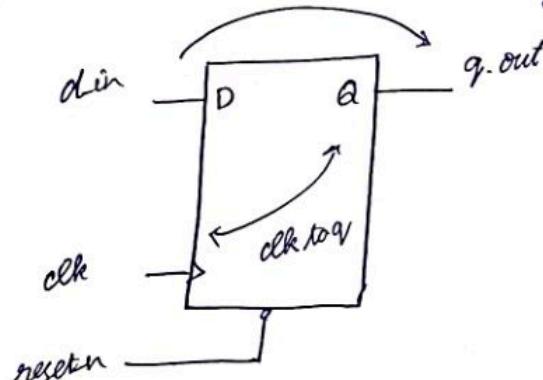
Full case & parallel case

\hookrightarrow // synonymous full-case



casez // synonymous parallel
 $3'b1?? : Y-out = irq[2];$
 $3'b?1? : Y-out = irq[1];$
 $3'b??1 : Y-out = irq[0];$
endcase

12-11-2024 Edge sensitive paths full parallel connector.



Specify
specparam
 $d_{toq} = 2\text{ns};$
 $d_{tck} = 1\text{ns};$
 $(D \Rightarrow q_{.0}) = d_{toq}$
 $(clk \Rightarrow q_{.0}) = d_{tck}$
endspecify

Practically we use ~~or~~ special parameter inside specify block
 (eg) d_in, clk, reactn. q \Rightarrow here D to Q delay are specified in specify block

\Rightarrow RTL design, check simulation get 100% code coverage.

Difference between \$finish & \$stop

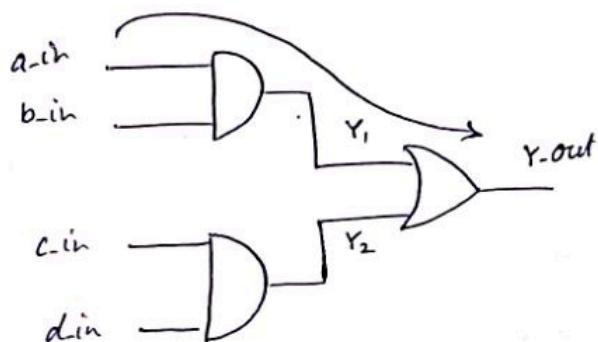
Full connection & Parallel connection

$$(a_{in} \Rightarrow y_{out}) = \text{delay};$$

Condition delay.

$$\text{if } (a_{in}) \\ (a_{in} \Rightarrow y_{out}) = 5;$$

$$\text{if } (\neg a_{in}) \\ (a_{in} \Rightarrow y_{out}) = 6;$$



Consider above circuit \Rightarrow we can specify pin \rightarrow pin delay.
 Practically, depending on state of ip signal pin \rightarrow pin delay

may change. Verilog allows path delays based on condition.

\Rightarrow are called as conditional path delays (or) state dependent path delays

Objective is understand parallel connection for different pin to pin depending on state of ip state.