

Digital Circuit Basic

Dr. Shamim Akhter

Fundamental Elements

Combinational Elements

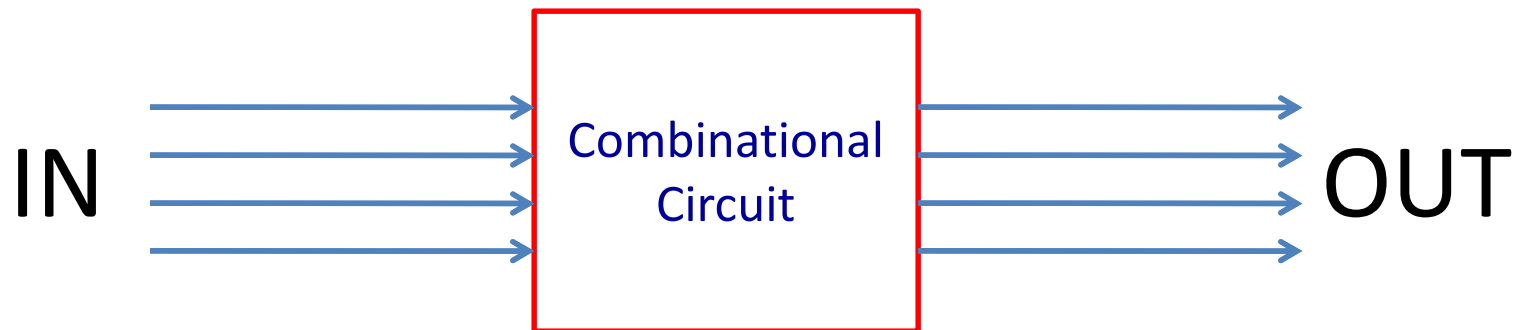
- Combinational Digital Circuit
- Gates, Multiplexors, Decoder, Encoder

Sequential Elements

- Sequence of States
- FF/Latch/Registers

Combinational Circuit

- No Internal Memory (Storage)
- Outputs are produced from current inputs only
- $OUT = F(IN)$



Timing and Combinational Circuit

- All logic devices are infinitely fast?
- Outputs immediately reflect input?
- Real World **Not TRUE!**

Glitching- A fault-Unpredictable Output
- May/May not have disastrous effect

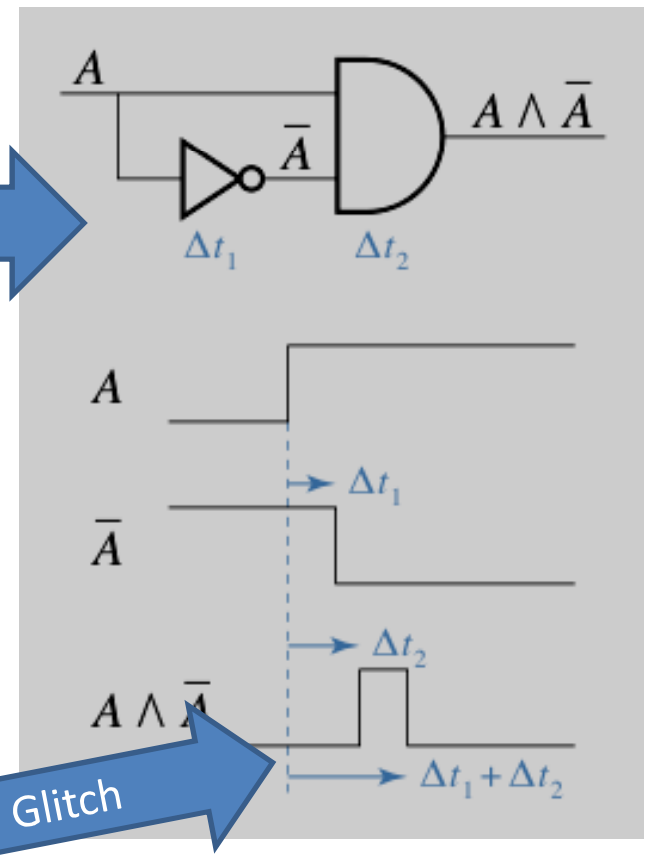
Possible to design a circuit with no glitch.

How to remove the glitch problem?

- design circuit insensitive to glitch (hard and costly)
- wait a fixed time after a change
(setup time delay and holding time delay).

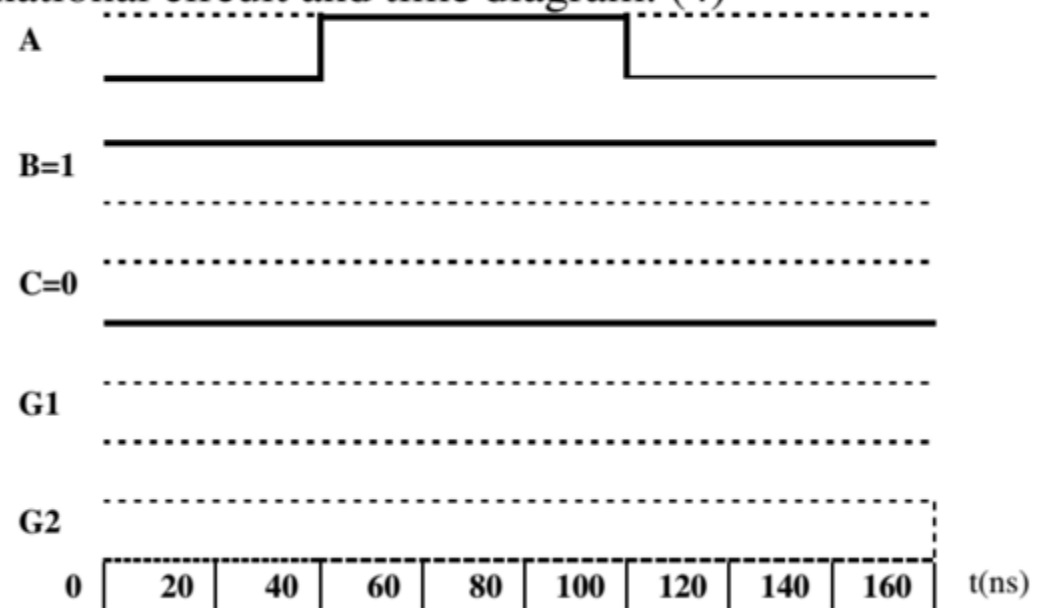
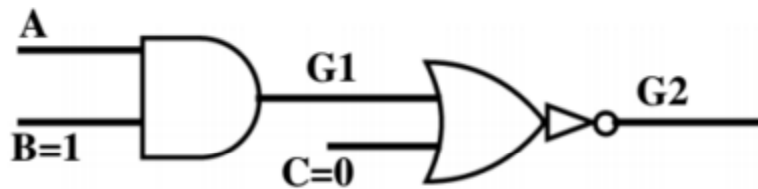
Basic Idea of clocked circuit (synchronous circuit).

Gate Delay



Experiment 1

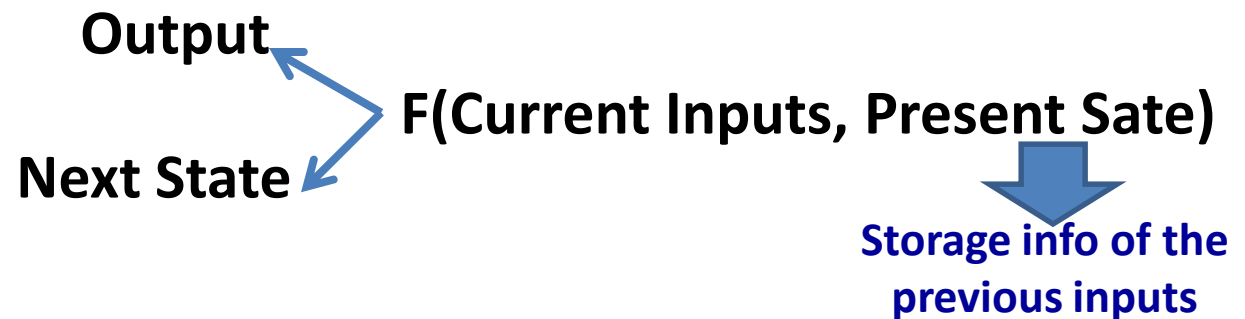
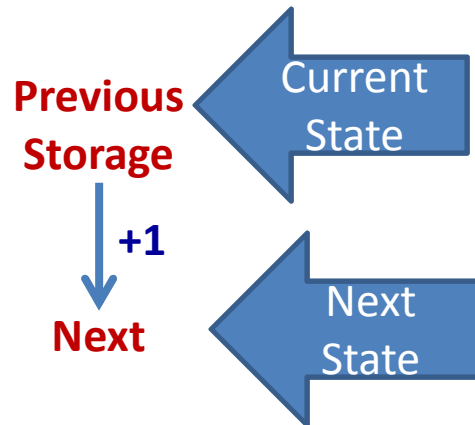
Assume the propagation delay for both AND gate and NOR gate is 20ns. Draw the signal flow for G1 and G2 by using the below combinational circuit and time diagram. (4)



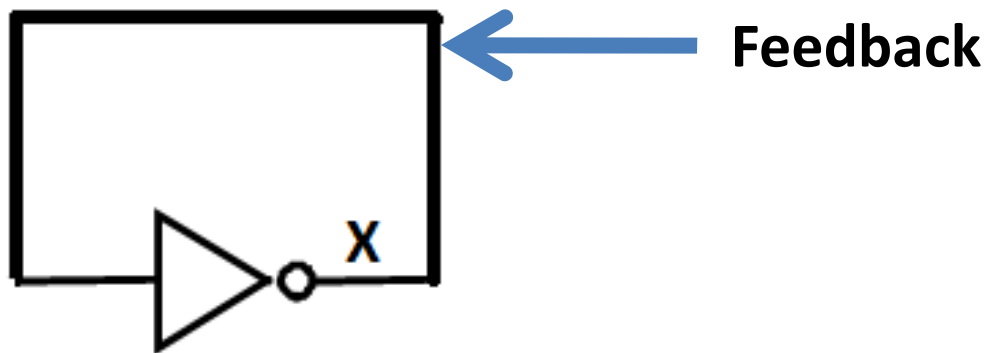
Sequential Circuit

- If we add the output of a combinational circuit to its input → Sequential Circuit
- Why do we need output to input?
 - Example: Counter [0-9]

0+1=1
1+1=2
2+1=3
3+1=4
.....
8+1=9



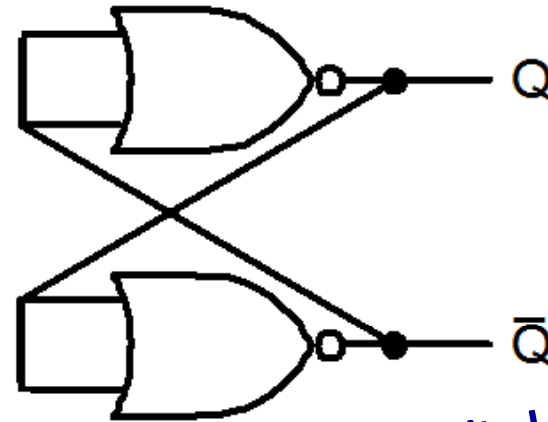
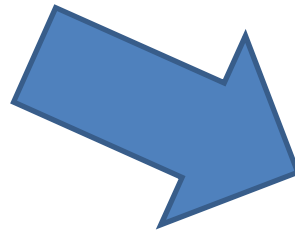
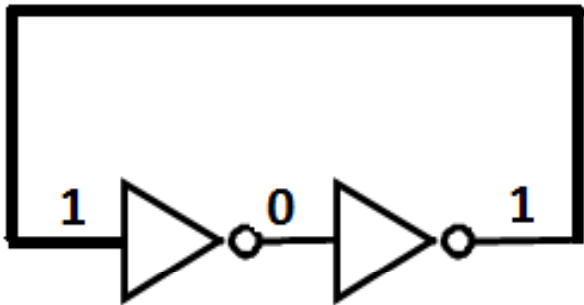
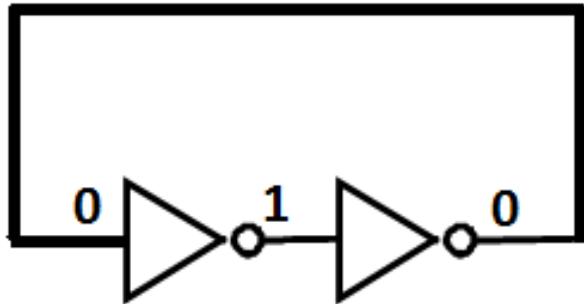
What is the purpose to use feedback in Sequential Circuit ?



- Input 0 propagates to output 1 and feedback as input
- Input 1 propagates to output 0 and feedback as input
- Result :
 - Continuous oscillation of output between 0 & 1
 - Not stable state (for same input-> output change overtime)

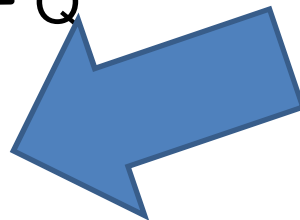
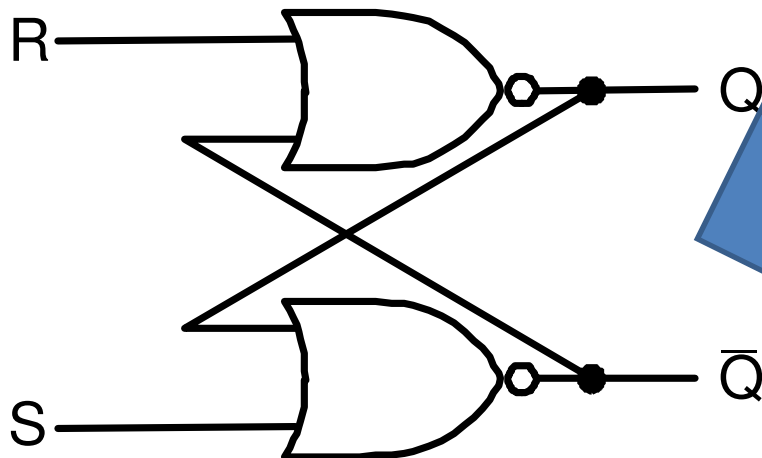
- Stable Circuit (same input always same output)

- 1 input -> 1 output
- 0 input -> 0 output



Storing Capability but Infinite Looping

How do we control this circuit??



S-R Latch

Sequential Circuit

Synchronous Circuit

controlled & synchronized by the periodic pulses of the clock

Asynchronous

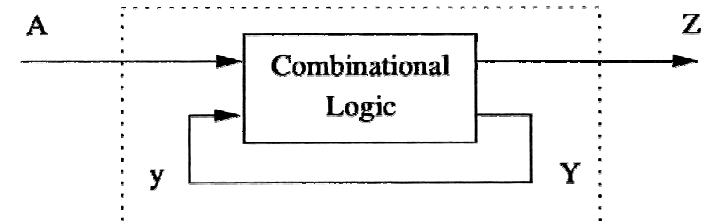
Only input event drive the circuit
No clock
Diffⁿ memory element can change state at diffⁿ time

Clocked

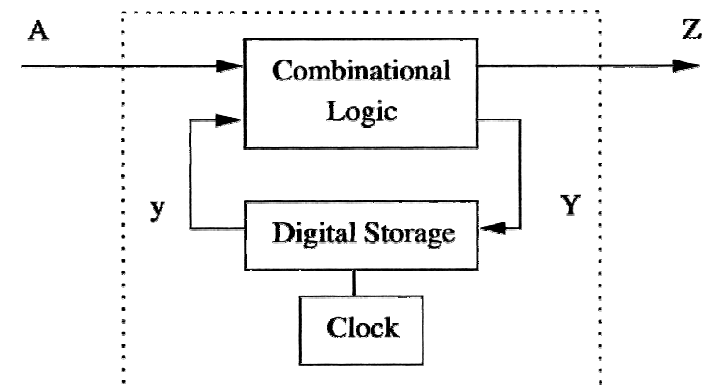
Edge Trigger

Level Trigger

Asynchronous



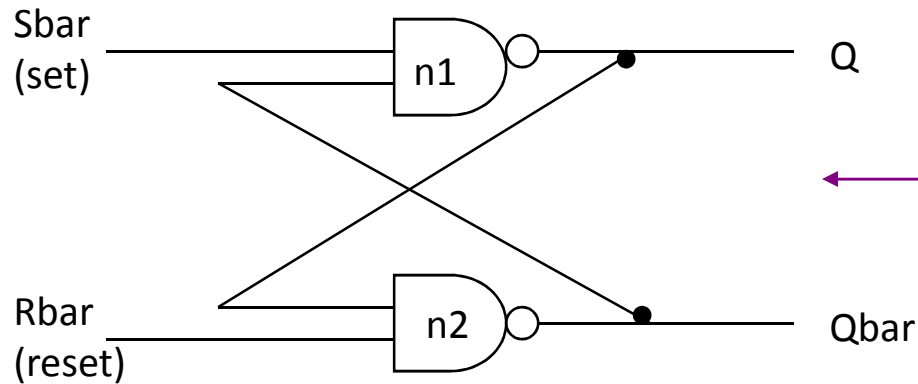
Synchronous



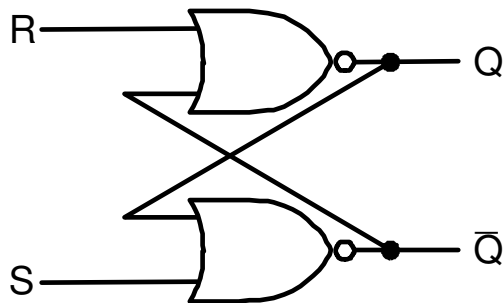
- Asynchronous to Synchronous

Set-Reset (SR-) latch (unclocked)

Think of S_{bar} as \overline{S} , the inverse of set (which sets Q to 1), and R_{bar} as \overline{R} , the inverse of reset.



equivalently with nor gates



A set-reset latch made from two cross-coupled *nand* gates is a basic memory unit.

When both S_{bar} and R_{bar} are 1, then either *one of the following two states is stable*:

- a) $Q = 1$ & $Q_{\text{bar}} = 0$
- b) $Q = 0$ & $Q_{\text{bar}} = 1$

and the latch will *continue* in the current stable state.

If S_{bar} changes to 0 (while R_{bar} remains at 1), then the latch is forced to the *exactly one* possible stable state (a). If R_{bar} changes to 0 (while S_{bar} remains at 1), the latch is forced to the *exactly one* possible stable state (b).

So, the latch *remembers* which of S_{bar} or R_{bar} was last 0 *during* the time they are both 1.

When both S_{bar} and R_{bar} are 0 the *exactly one* stable state is $Q = Q_{\text{bar}} = 1$. However, if after that both S_{bar} and R_{bar} return to 1, the latch must then *jump non-deterministically* to one of stable states (a) or (b), which is undesirable behavior.

S-R Latch VHDL Code

```
module sr_latch(Q, Qbar, Sbar, Rbar);  
  
  //Port declarations  
  output Q, Qbar;  
  input Sbar, Rbar;  
  
  // Instantiate lower level modules  
  // In this case, instantiate Verilog primitive "nand" gates  
  // Note, how the wires are connected in a cross coupled fashion.  
  nand n1(Q, Sbar, Qbar);  
  nand n2(Qbar, Rbar, Q);  
  
  // endmodule statement  
endmodule
```

```
// Module name and port list// Stimulus module
```

```
module Top;
```

```
// Declarations of wire, reg and other variables
```

```
wire q, qbar;
```

```
reg set, reset;
```

```
// Instantiate lower level modules
```

```
// In this case, instantiate SR_latch
```

```
sr_latch l1(q, qbar, ~set, ~reset);
```

```
// Behavioral block, initial
```

```
initial
```

```
begin
```

```
$monitor($time, "set = %b, reset= %b, q= %b, qbar= %b\n", set, reset, q, qbar);
```

```
set = 0; reset = 0;
```

```
#5 set = 0; reset = 1;
```

```
#5 set = 0; reset = 0;
```

```
#5 set = 1; reset = 0;
```

```
#5 set = 0; reset = 0;
```

```
// Uncomment the following two statements *one after another*
```

```
// to see a problem with non-deterministic behavior
```

```
#5 set = 1; reset = 1;
```

```
#5 set = 0; reset = 0;
```

```
end
```

```
// endmodule statement
```

```
endmodule
```

TERMINAL OUTPUT

```
0set = 0, reset= 0, q= x, qbar= x
```

```
5set = 0, reset= 1, q= 0, qbar= 1
```

```
10set = 0, reset= 0, q= 0, qbar= 1
```

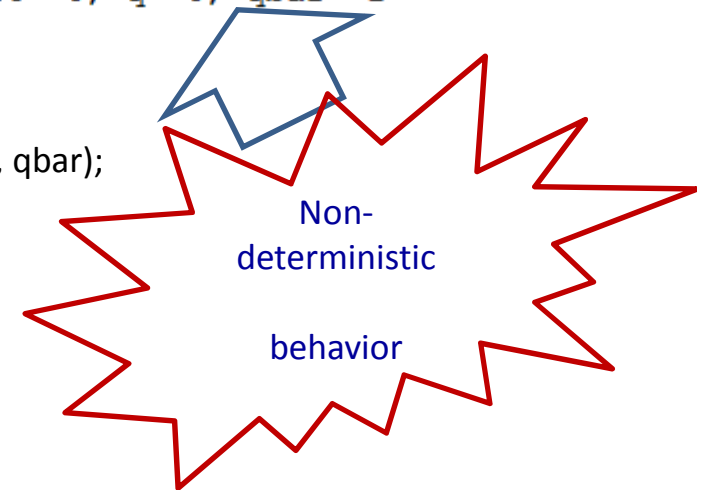
```
15set = 1, reset= 0, q= 1, qbar= 0
```

```
20set = 0, reset= 0, q= 1, qbar= 0
```

```
25set = 1, reset= 1, q= 1, qbar= 1
```

```
30set = 0, reset= 0, q= 0, qbar= 1
```

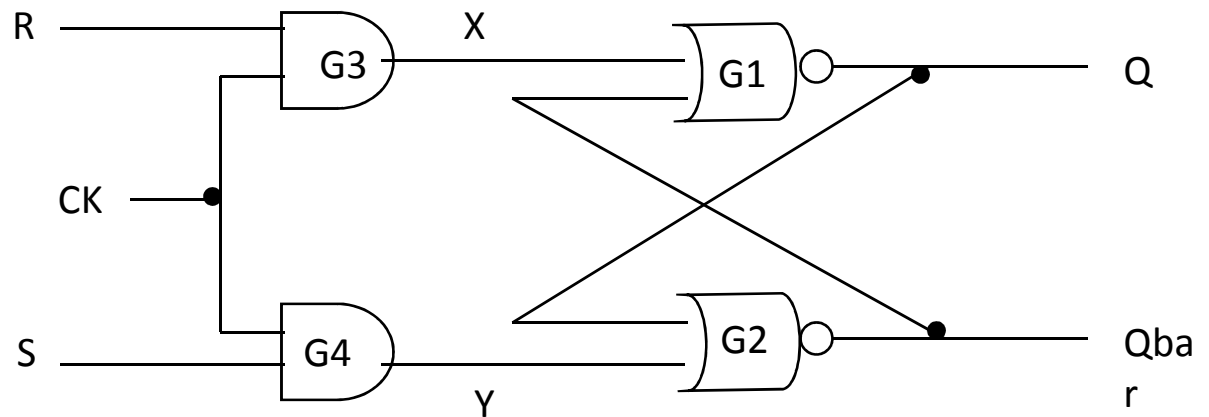
Forbidden
state



Clocked SR-latch

- State will unchanged when clock is *low* (CK=0)
- If clock pulse is present (CK=1), it works exactly same as previous slide
- Potential problem** : both inputs $S = 1$ & $R = 1$ will cause non-deterministic/undefined/ambiguous behavior

S_n	R_n	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0
0	0	?	?



S_n	R_n	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	?

Reduced Characteristic Table

Clock S-R Latch VHDL Code

```
module clockGenerator(clk);
    output clk;
    reg clk;

    initial
        begin
            clk = 0;
            // Uncomment the next line if you run this
            // module stand-alone or it will never exit!
            #30 $finish;
        end

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

```
module clock_sr_latch(Q, Qbar, S, R, clk);

//Port declarations
output Q, Qbar;
input S, R, clk;
wire X, Y;

and a1(X, R, clk);
and a2(Y, S, clk);

nor n1(Q, X, Qbar);
nor n2(Qbar, Y, Q);

// endmodule statement
endmodule
```

```

module Top;
// Declarations of wire, reg and other variables
wire q, qbar;
input clk;
reg set, reset;

```

```

// Instantiate lower level modules
// In this case, instantiate SR_latch
clock_sr_latch l1(q, qbar, set, reset, clk);
clockGenerator s1(clk);
// Behavioral block, initial
Initial
begin

```

```

$monitor($time, " clk= %b, set = %b, reset= %b, q= %b,
qbar= %b\n", clk, set, reset, q, qbar);

```

```

set = 0; reset = 0;
#2 set = 0; reset = 1;
#2 set = 0; reset = 0;
#2 set = 1; reset = 0;
#2 set = 0; reset = 1;
#2 set = 0; reset = 0;
#2 set = 1; reset = 0;

```

```

// Uncomment the following two statements *one after another*

```

```

// to see a problem with non-deterministic behavior

```

```

#2 set = 1; reset = 1;
#2 set = 0; reset = 0;

```

```

end

```

TERMINAL OUTPUT

```

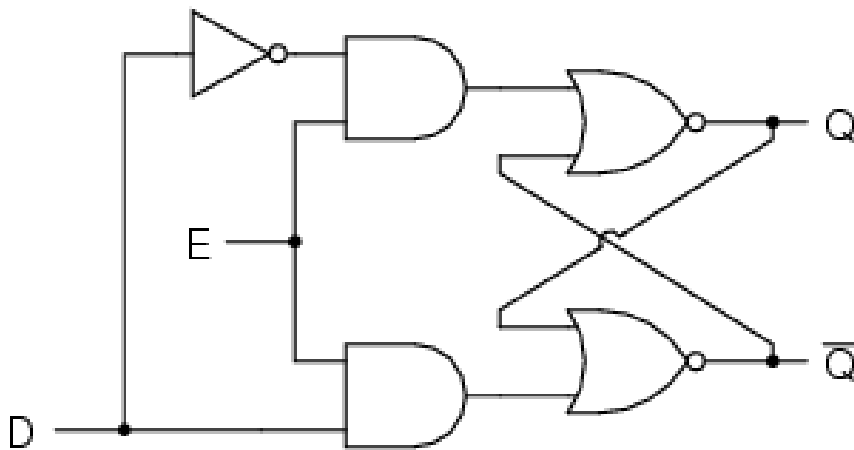
0 clk= 0, set = 0, reset= 0, q= x, qbar= x
2 clk= 0, set = 0, reset= 1, q= x, qbar= x
4 clk= 0, set = 0, reset= 0, q= x, qbar= x
5 clk= 1, set = 0, reset= 0, q= x, qbar= x
6 clk= 1, set = 1, reset= 0, q= 1, qbar= 0
8 clk= 1, set = 0, reset= 1, q= 0, qbar= 1
10 clk= 0, set = 0, reset= 0, q= 0, qbar= 1
12 clk= 0, set = 1, reset= 0, q= 0, qbar= 1
14 clk= 0, set = 1, reset= 1, q= 0, qbar= 1
15 clk= 1, set = 1, reset= 1, q= 0, qbar= 0
16 clk= 1, set = 0, reset= 0, q= 0, qbar= 1
17 clk= 0, set = 0, reset= 0, q= 0, qbar= 1
20 clk= 0, set = 0, reset= 0, q= 0, qbar= 1
25 clk= 1, set = 0, reset= 0, q= 0, qbar= 1
30 clk= 0, set = 0, reset= 0, q= 0, qbar= 1

```

Non-
determinis
tic
behavior

Clocked D-latch

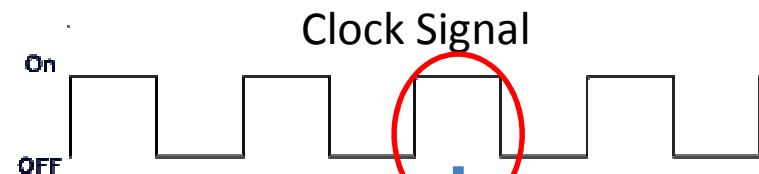
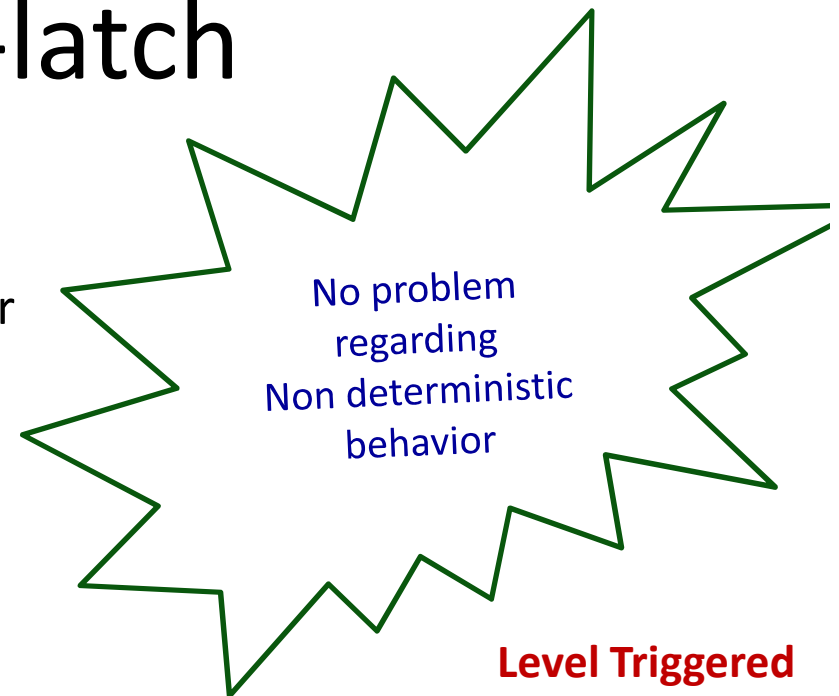
- State can change only when *clock is high*
- Only *single* data input (compare SR-latch)
- **No problem** with non-deterministic behavior



D	Clk (E)	Q_n	Q_{n+1}
0	1	X	0
1	1	X	1
x	0	Q_n	Q_n

Reduced Characteristic Table

D_n	Q_{n+1}
0	0
1	1



Clock on

But input forces output to change

May create unstable/unpredictable result

- 1) - Need D must be stable critical change in the clock signal takes place.
- 2) - Input and clock change at the same time
Need storage elements can't change their state more than one during one clock cycle

```
module clockGenerator(clk);
    output clk;
    reg clk;

    initial
        begin
            clk = 0;
//    Uncomment the next line if you run this
//    module stand-alone or it will never exit!
            #10 $finish;
        end

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

```
module clock_D_latch(Q, Qbar, D, clk);

//Port declarations
output Q, Qbar;
input D, clk;
wire X, Y, Dbar;

// Instantiate lower level modules
// Note, how the wires are connected in a
//cross coupled fashion.

not n1(Dbar, D);
and a1(X, Dbar, clk);
and a2(Y, D, clk);

nor n1(Q, X, Qbar);
nor n2(Qbar, Y, Q);

// endmodule statement
endmodule
```

```
// Stimulus module
module Top;

// Declarations of wire, reg and other variables
wire q, qbar;
input clk;
reg d;

// Instantiate lower level modules
// In this case, instantiate D_latch
clock_D_latch l1(q, qbar, d, clk);
clockGenerator s1(clk);

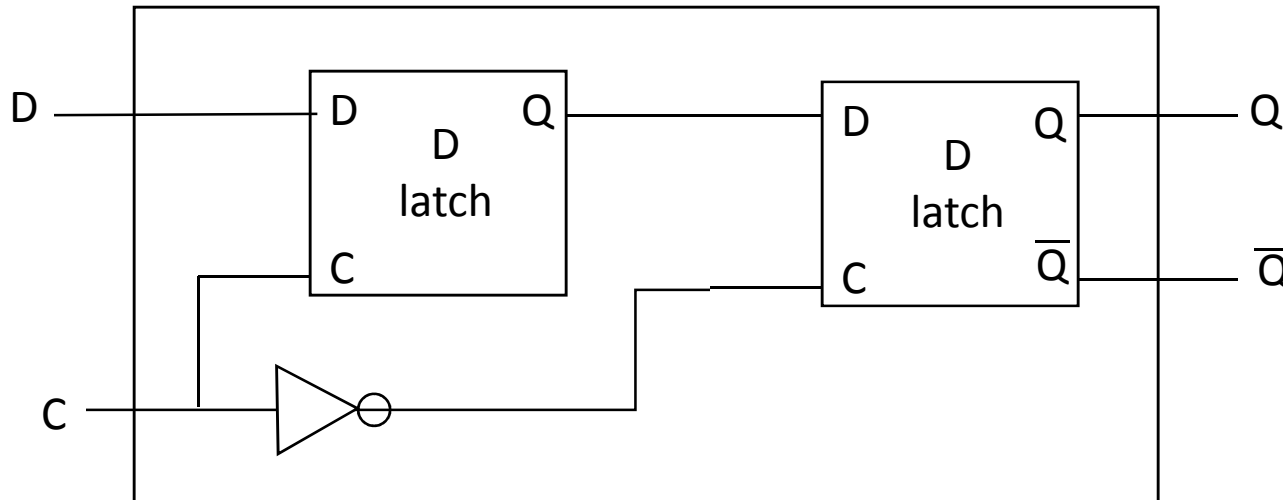
// Behavioral block, initial
initial
    begin
        $monitor($time, " clk= %b, D = %b,q= %b, qbar= %b\n", clk, d, q, qbar);
        #2 d=0;
        #2 d=1;
        #2 d=0;
        #2 d=1;
        #2 d=0;
    end
endmodule
```

**Within
clk state
changes**

```
0 clk= 0, D = x,q= x, qbar= x
2 clk= 0, D = 0,q= x, qbar= x
4 clk= 0, D = 1,q= x, qbar= x
5 clk= 1, D = 1,q= 1, qbar= 0
6 clk= 1, D = 0,q= 0, qbar= 1
8 clk= 1, D = 1,q= 1, qbar= 0
10 clk= 0, D = 0,q= 1, qbar= 0
```

Clocked D-flipflop

- **Negative edge/falling-edge-triggered**
- Made from *Two D-latches*



Data is consistent for whole cycle.

We can store data still input clock is low.

Lots of latches and a big decoder equals one computer memory.

The first latch, called master, is open and follows the input D, when the clock input, C, is asserted.

When clock input C falls, the first latch is closed, but the second latch, called slave, is open and gets the input from the output of the master latch.

Edge interrupt gets fired only on changing edges, while level interrupts gets fired as long as the pulse is low or high.

How to convert positive edge trigger?

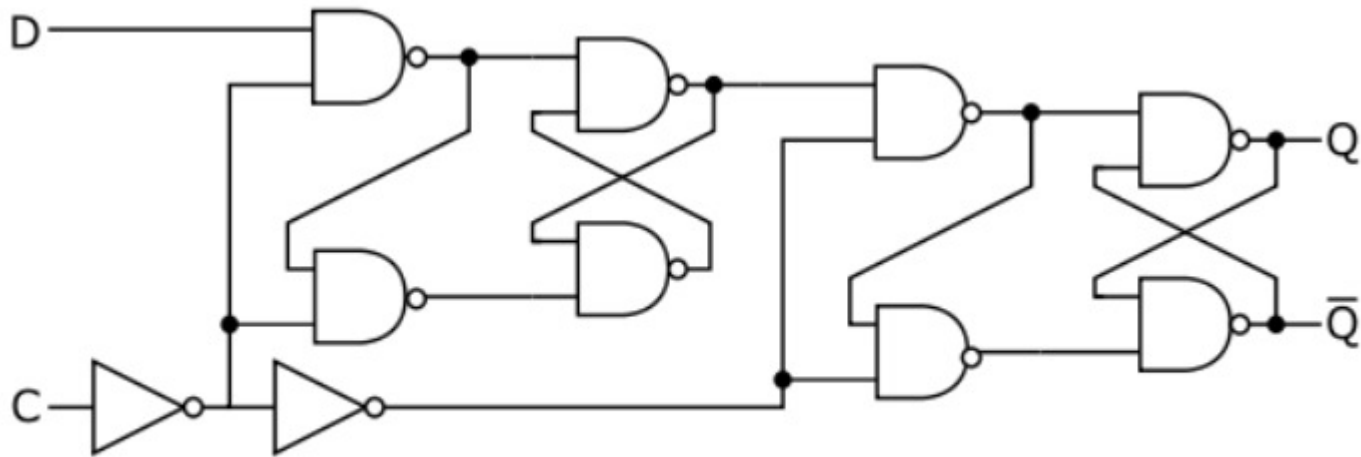
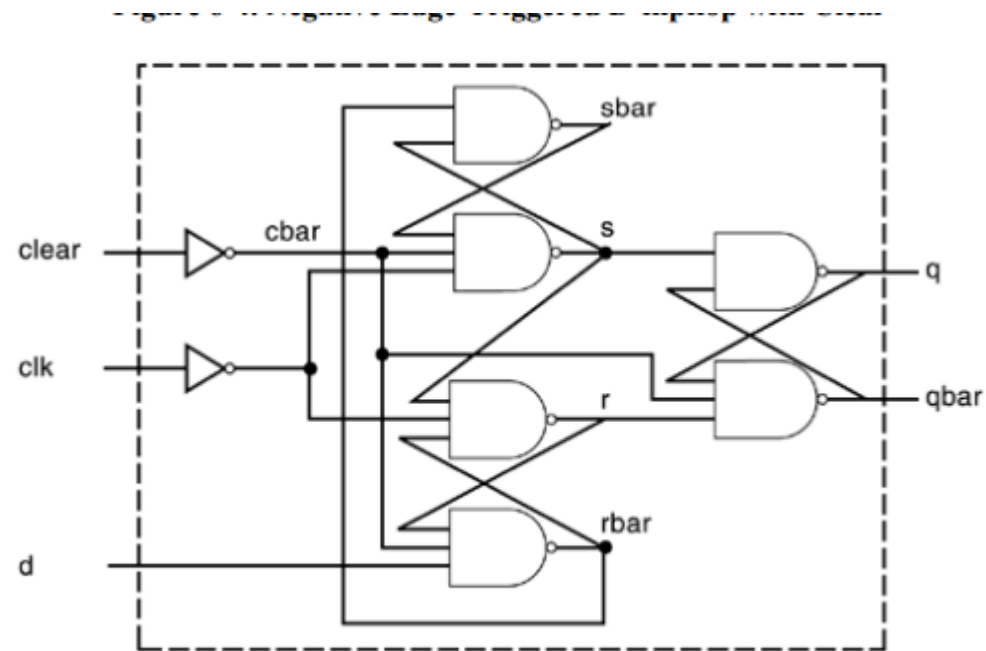


Figure 6 A Positive Edge Triggered Master-Slave D Flip-Flop



Negative Edge-triggered D-flipflop with Clear

```
module edge_dff( q, d, enable, clock, clear );
```

```
// Inputs and outputs
```

```
output q;
```

```
input d, enable, clock, clear;
```

```
// Internal wires
```

```
wire clkbar, qbar, s, sbar, r, rbar, cbar;
```

```
// The enable signal is anded with the clock  
and ( clk, clock, enable );
```

```
//Create complements of clear and clk
```

```
not ( cbar, clear );
```

```
not ( clkbar, clk );
```

```
// See circuit Fig.6-4 in Palnitkar Ch. 6
```

```
nand ( sbar, rbar, s );
```

```
nand ( s, sbar, cbar, clkbar );
```

```
nand ( r, rbar, clkbar, s );
```

```
nand ( rbar, r, cbar, d );
```

```
nand ( q, s, qbar );
```

```
nand ( qbar, q, r, cbar );
```

```
endmodule
```

**D Negative edge clock
With clear and enable signal**

```

module Top;

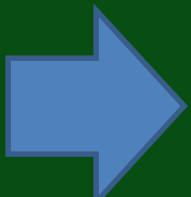
wire q, qbar, clk;
reg d, clear, enable;

edge_dff ff1(q,d,enable, clk, clear);
clockGenerator cg(clk);

initial
begin
    clear = 1;
    enable=1;

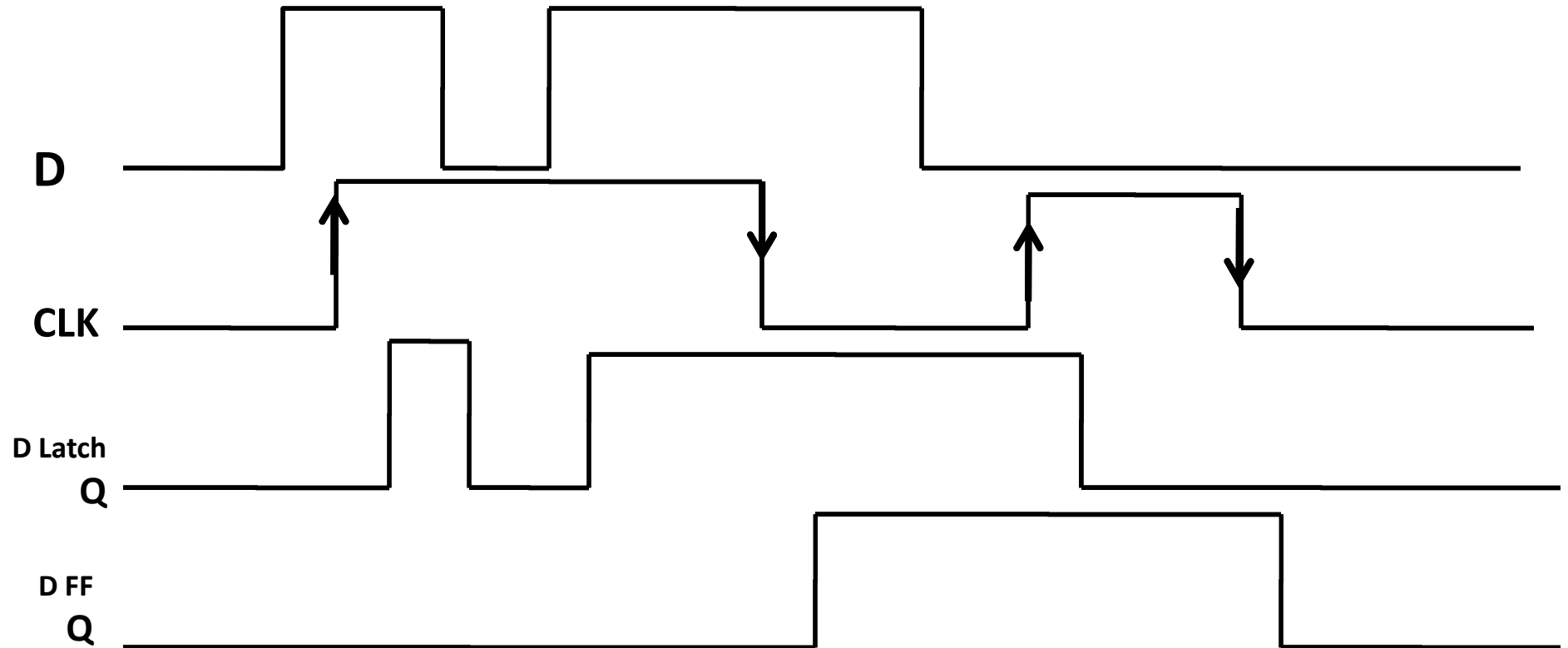
    #2 d = 0;
    #1 $display(" time = %d, clk= %b, d = %b, q= %b\n", $time, clk, d, q);
    #1 d = 1;clear = 0;
    #1 $display(" time = %d, clk= %b, d = %b, q= %b\n", $time, clk, d, q);
    #1 d = 0;
    #1 $display(" time = %d, clk= %b, d = %b, q= %b\n", $time, clk, d, q);
    #1 d = 1;
    #1 $display(" time = %d, clk= %b, d = %b, q= %b\n", $time, clk, d, q);
    #3 d = 0;
    #1 $display(" time = %d, clk= %b, d = %b, q= %b\n", $time, clk, d, q);
end
initial
begin
    #100$finish;

```



time =	3, clk= 0, d = 0, q= 0
time =	5, clk= 1, d = 1, q= 0
time =	7, clk= 1, d = 0, q= 0
time =	9, clk= 0, d = 1, q= 1
time =	13, clk= 1, d = 0, q= 1

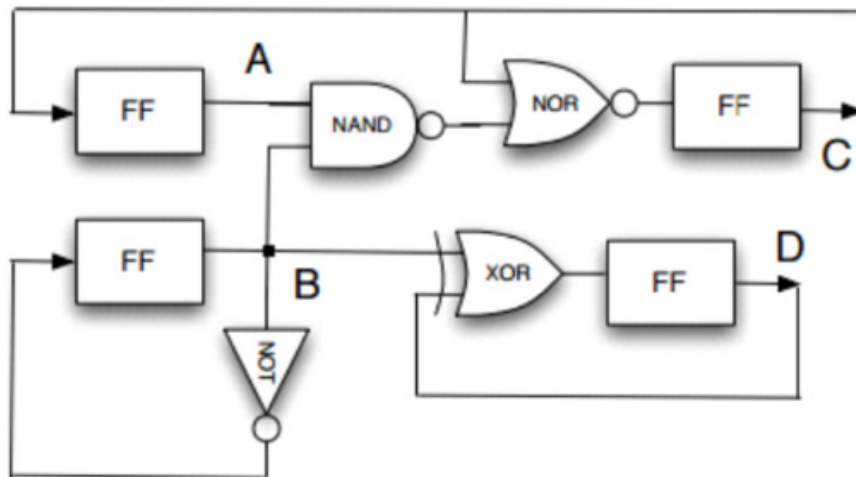
Timing diagram of D latch and D FF



Operations of D latch and negative-edge-trigger D flip-flop

Experiment 2

Consider the following sequential circuit (assume all flip-flops are **negative edge triggered**):

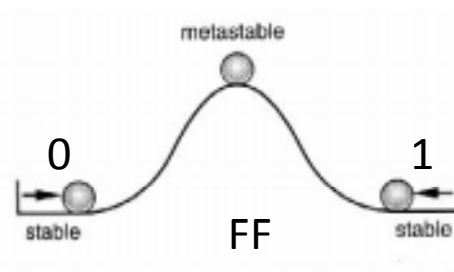


Clk	0	1	0	1	0	1	0
A	1	1	0	0	0	0	0
B	0	0	1	1	0	0	1
C	0	0	0	0	0	0	0
D	1	1	1	1	0	0	0

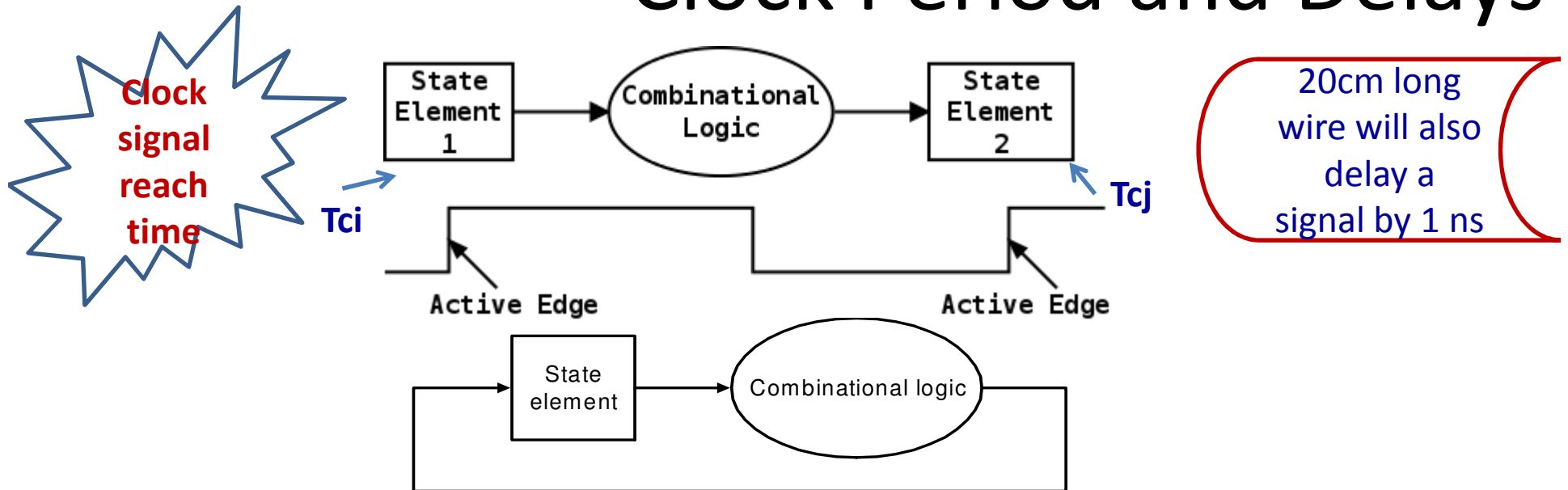
Show the values of the indicated signals at each time step. The clock is shown on the top line of the above table, and the initial values are shown in the first column. (4)

Concern with Edge Trigger FFs

- Correct operation of the circuit
 - ensure by making the clock period long enough to accommodate the worst case delay
 - Otherwise
 - Metastability will rise
 - setup time and hold time of a FF are not met
 - delaying the signal by a cycle is usually sufficient
- Clock period duration is important



Clock Period and Delays



$$\text{Clock periods} \geq t_{\text{pop}} + t_{\text{comb}} + t_{\text{setup}} + t_{\text{skew}}$$

Propagation delay (t_{pop}): delay associated with transferring an input change to output

Combinational Logic Delay (t_{comb}): delay to transfer at combinational circuit

Set up time (t_{setup}): Give enough time to setup a state element

Minimum amount of time input must be stable before clock tick

Clock skew (t_{skew}) positive/negative. $t_{\text{skew } i,j} = T_{ci} - T_{cj} > 0$ positive otherwise 0 or negative

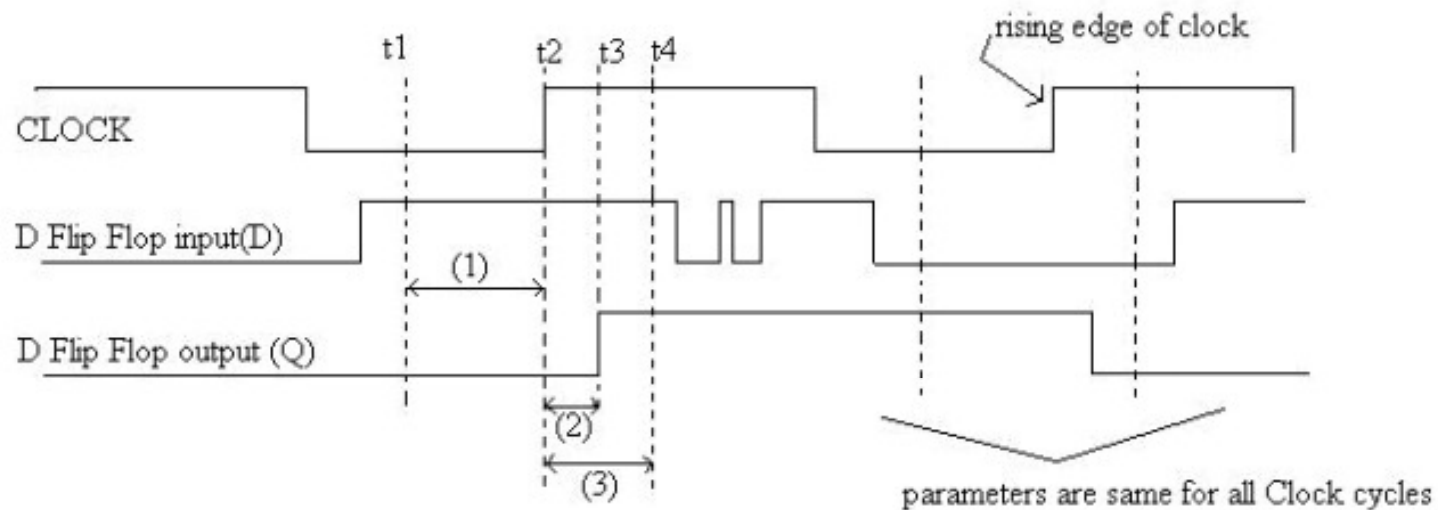
(Special) Holdtime t_{hold} : minimum amount of time after clock pulse, input must not change.

Input signal must be stable after clock tick -> will guarantee to remove metastability.

Experiment 3

Q8. What do Setup and Hold time look like on a timing diagram?

[Ans] Observe the waveform below:



The timing diagram above illustrates three signals: the Clock, the Flip Flop Input (D) and the Flip Flop output (Q).

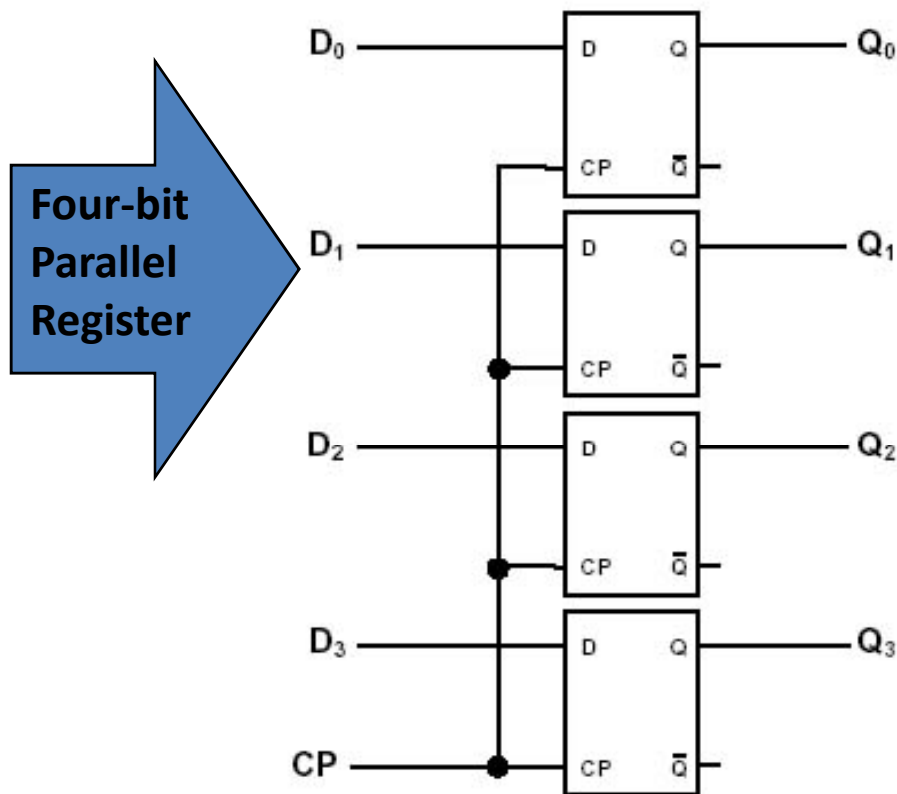
(1) is the Setup Time $[t_2 - t_1]$: the minimum amount of time Input must be held constant BEFORE the clock tick. Note that D is actually held constant for somewhat longer than the minimum amount. The extra “constant” time is sometimes called the setup margin.

(2) is the Propagation delay of the Flip Flop $[t_3 - t_2]$: this is the time that it takes for the new input to be to propagate and influence the output.

(3) is the Hold time $[t_4 - t_3]$: the minimum amount of time the Input is held constant AFTER the clock tick. Note that Q is actually held constant for somewhat longer than the minimum amount. The extra “constant” time is sometimes called the hold margin.

State Elements on the Datapath: Register File

- A register is simply a collection of **edge triggered D flip-flops**.



```
module register4bits( dataOut, dataIn, enable,  
clock, clear );
```

```
// Inputs and outputs
```

```
output [3:0] dataOut;
```

```
input [3:0] dataIn;
```

```
input enable, clock, clear;
```

```
// 4 D-flipflops
```

```
edge_dff ff0( dataOut[0], dataIn[0], enable,  
clock, clear );
```

```
edge_dff ff1( dataOut[1], dataIn[1], enable,  
clock, clear );
```

```
edge_dff ff2( dataOut[2], dataIn[2], enable,  
clock, clear );
```

```
edge_dff ff3( dataOut[3], dataIn[3], enable,  
clock, clear );
```

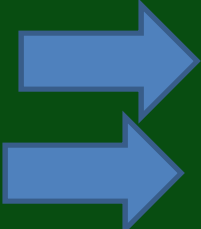
```
endmodule
```

```

module Top;

wire [3:0] dataOut;
reg [3:0] dataIn;
wire clk;
reg d, clear, enable;

```



time =	clk	dataIn[0-4]	dataOut[0-4]
5	1	0000	0000
9	0	0110	0110
13	1	1001	0110
17	0	1111	1111
21	1	0000	1111

```

register4bits rg4(dataOut, dataIn, enable, clk, clear );
clockGenerator cg(clk);

```

Clock changing after 4 pulse
Negative Edge trigger FFs

```

initial
begin
    clear = 1;
    enable=1;

```

```

#4 dataIn[0] = 0; dataIn[1] = 0; dataIn[2] = 0; dataIn[3] = 0;
#1 $display(" time = %d, clk= %b, dataIn[0-4] = %b%b%b%b,dataOut[0-4]= %b%b%b%b\n",
$time, clk,dataIn[0],dataIn[1],dataIn[2],dataIn[3],dataOut[0],dataOut[1],dataOut[2],dataOut[3]);
#3 dataIn[0] = 0; dataIn[1] = 1; dataIn[2] = 1; dataIn[3] = 0; clear = 0;
#1 $display(" time = %d, clk= %b, dataIn[0-4] = %b%b%b%b,dataOut[0-4]= %b%b%b%b\n",
$time, clk,dataIn[0],dataIn[1],dataIn[2],dataIn[3],dataOut[0],dataOut[1],dataOut[2],dataOut[3]);
#3 dataIn[0] = 1; dataIn[1] = 0; dataIn[2] = 0; dataIn[3] = 1;
#1 $display(" time = %d, clk= %b, dataIn[0-4] = %b%b%b%b,dataOut[0-4]= %b%b%b%b\n",
$time, clk,dataIn[0],dataIn[1],dataIn[2],dataIn[3],dataOut[0],dataOut[1],dataOut[2],dataOut[3]);
#3 dataIn[0] = 1; dataIn[1] = 1; dataIn[2] = 1; dataIn[3] = 1;

```

```

#1 $display(" time = %d, clk= %b, dataIn[0-4] = %b%b%b%b,dataOut[0-4]= %b%b%b%b\n",
    $time, clk, dataIn[0], dataIn[1], dataIn[2], dataIn[3], dataOut[0], dataOut[1],
    dataOut[2],dataOut[3]);
#3 dataIn[0] = 0; dataIn[1] = 0; dataIn[2] = 0; dataIn[3] = 0;
#1 $display(" time = %d, clk= %b, dataIn[0-4] = %b%b%b%b,dataOut[0-4]= %b%b%b%b\n",
    $time, clk,
    dataIn[0],dataIn[1],dataIn[2],dataIn[3],dataOut[0],dataOut[1],dataOut[2],dataOut[3]);
end

initial
begin
    #100$finish;
end

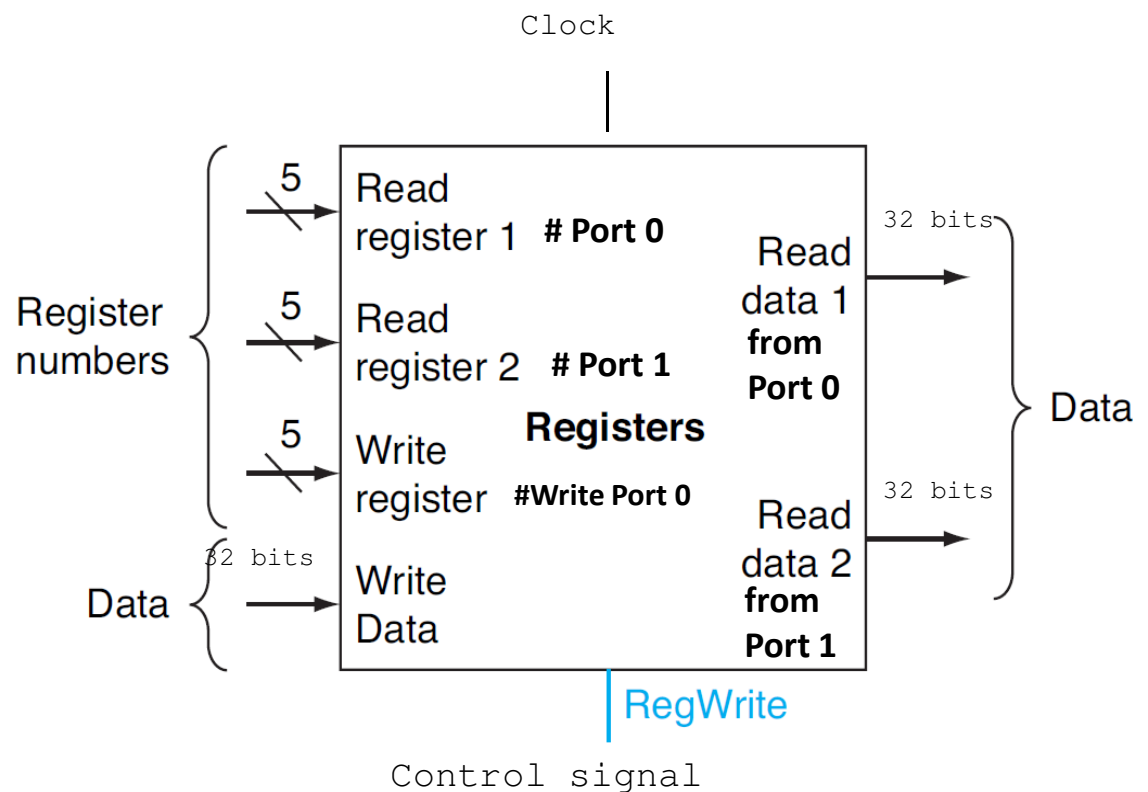
endmodule

```

State Elements on the Datapath: Register File

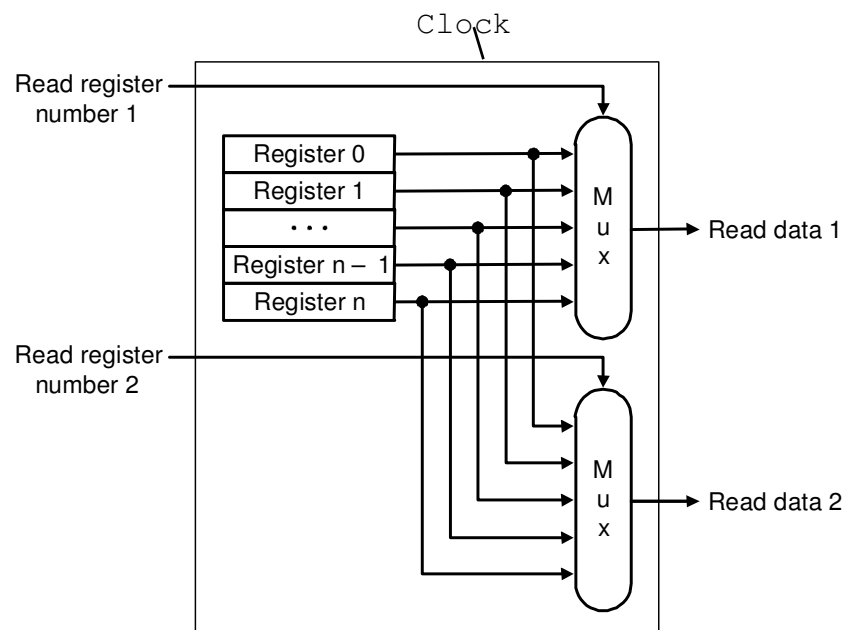
A register-file block-diagram assuming:

- 32 registers numbered from 0 to 31 (00000_2 to 11111_2 in binary)
- 32-bit registers
- two read ports
- one write port



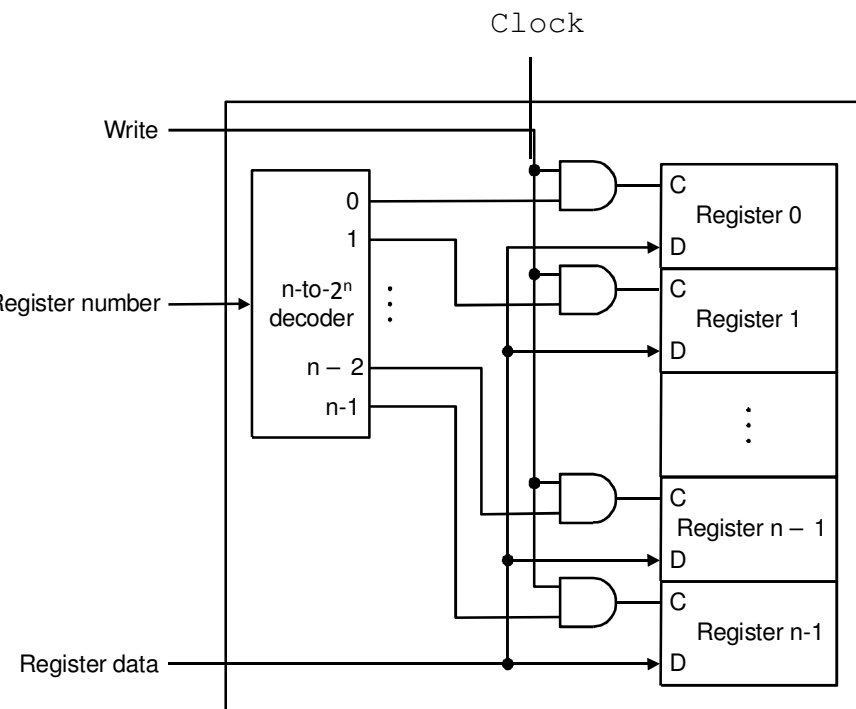
State Elements on the Datapath: Register File

- Port implementation:



Read ports are implemented with a pair of multiplexors – 32-to-1 multiplexors each 32 bits wide

Add \$t3, \$t2, \$t1



Write port is implemented using a decoder – 5-to-32 decoder for 32 registers. Clock is relevant to write as register state may change only at clock edge

```
module memory4registers( readData, writeData, readAddress, writeAddress, MemWrite, clock,
clear );
```

```
// Inputs and outputs
output [3:0] readData;
input [3:0] writeData;
input [1:0] readAddress, writeAddress;
input MemWrite, clock, clear;
```

```
// Internal wires
wire [3:0] dataOut0, dataOut1, dataOut2, dataOut3;
wire [3:0] dataIn0, dataIn1, dataIn2, dataIn3;
wire y0, y1, y2, y3;
wire writeEnable0, writeEnable1, writeEnable2, writeEnable3;
```

```
// Four 4-bit registers
register4bits register0( dataOut0, writeData, writeEnable0, clock, clear );
register4bits register1( dataOut1, writeData, writeEnable1, clock, clear );
register4bits register2( dataOut2, writeData, writeEnable2, clock, clear );
register4bits register3( dataOut3, writeData, writeEnable3, clock, clear );
```

Register File

```

// Four 4-to-1 multiplexors, one to read each bit of data based on address input
mux4_to_1 muxMemory0( readData[0], dataOut0[0], dataOut1[0], dataOut2[0],
dataOut3[0], readAddress[1], readAddress[0] );
mux4_to_1 muxMemory1( readData[1], dataOut0[1], dataOut1[1], dataOut2[1],
dataOut3[1], readAddress[1], readAddress[0] );
mux4_to_1 muxMemory2( readData[2], dataOut0[2], dataOut1[2], dataOut2[2],
dataOut3[2], readAddress[1], readAddress[0] );
mux4_to_1 muxMemory3( readData[3], dataOut0[3], dataOut1[3], dataOut2[3],
dataOut3[3], readAddress[1], readAddress[0] );

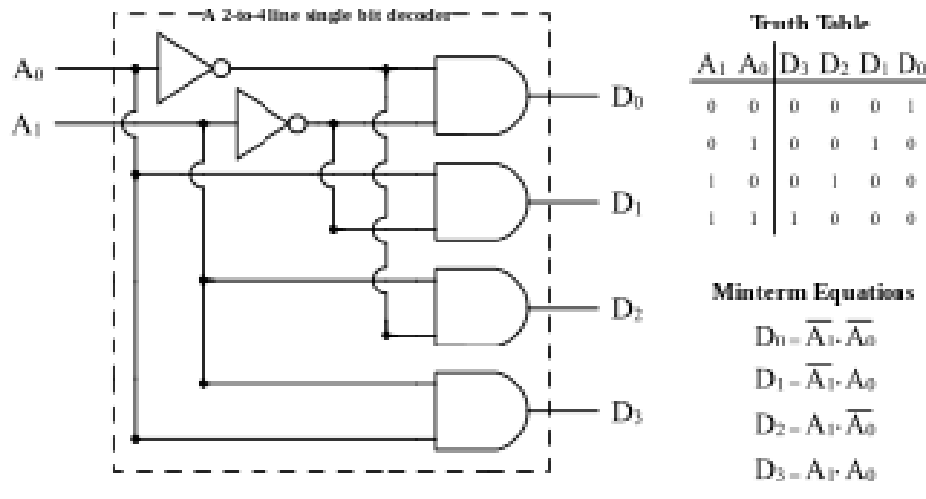
// One 2-to-4 decoder to select register for write
decoder2_to_4 decoderMemory( y0, y1, y2, y3, writeAddress[1], writeAddress[0] );

// Decoder output is anded with MemWrite to give enable signal for each register
and( writeEnable0, MemWrite, y0 );
and( writeEnable1, MemWrite, y1 );
and( writeEnable2, MemWrite, y2 );
and( writeEnable3, MemWrite, y3 );

endmodule

```

2 to 4 Decoder



```
module decoder2_to_4 ( y0, y1, y2, y3, s1, s0 );
```

```
// Inputs and outputs
```

```
output y0, y1, y2, y3;
```

```
input s1, s0;
```

```
// Internal wires
```

```
wire s1n, s0n;
```

```
// Create complements of s1 and s0
```

```
not ( s1n, s1 );
```

```
not ( s0n, s0 );
```

```
// See multiplexor circuit Fig. 5-5 in Palnitkar
```

```
//Ch. 5 (without the i inputs and final or gate)
```

```
and ( y0, s1n, s0n );
```

```
and ( y1, s1n, s0 );
```

```
and ( y2, s1, s0n );
```

```
and ( y3, s1, s0 );
```

```
endmodule
```

Experiment 4

Suppose a computer has a **24-bit instruction format** with the following fields:

<u>opcode</u>	<u>ra</u>	<u>rb</u>	<u>rc</u>	<u>rd</u>
8 bits	4 bits	4 bits	4 bits	4 bits

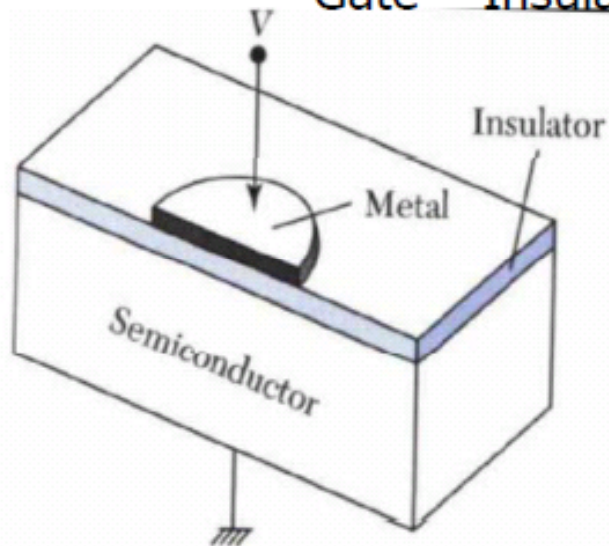
ra, rb, rc, specify the input registers and rd specifies the destination register. **Draw the register file block diagram** to represent the answer of the following questions (3.a and 3.b).

- If there is a single register file, how many registers should there be in that register file?
- How many data input lines? How many data output lines?

Switch Level Modeling

- MOS (Metal, Oxide, Silicon)

Gate Insulation Semiconductor



MOSFET Type	Logic Circuit Symbol	$A = 0$ Approximation	$A = 1$ Approximation
NMOS			
PMOS			

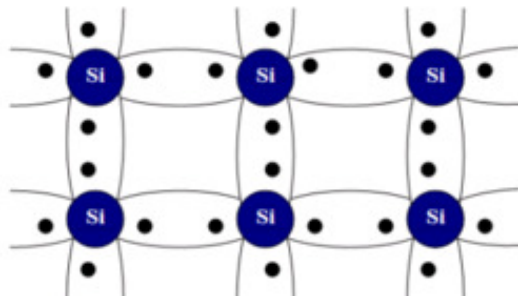
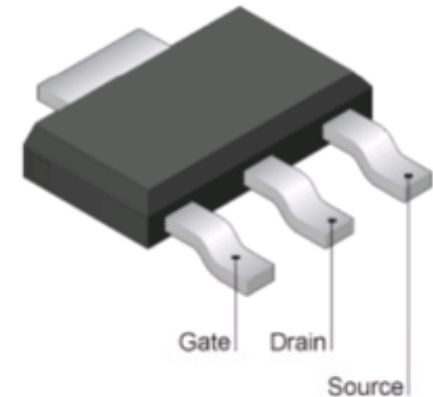
What is MOSFET?- MOS Field Effect Transistor

Field (current) plays a central role in the operation of the device

How Does Transistor Work?

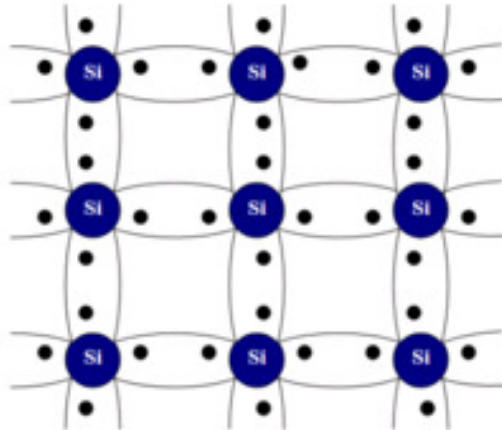
- Transistor -> two states switch (ON/OFF)

- No human control activity
- Incredibly tiny around 22nm wide
- Voltage regulates electricity
- Semiconductor -Silicon
 - conduct electricity better than insulator but not good as silver (highest conductivity)

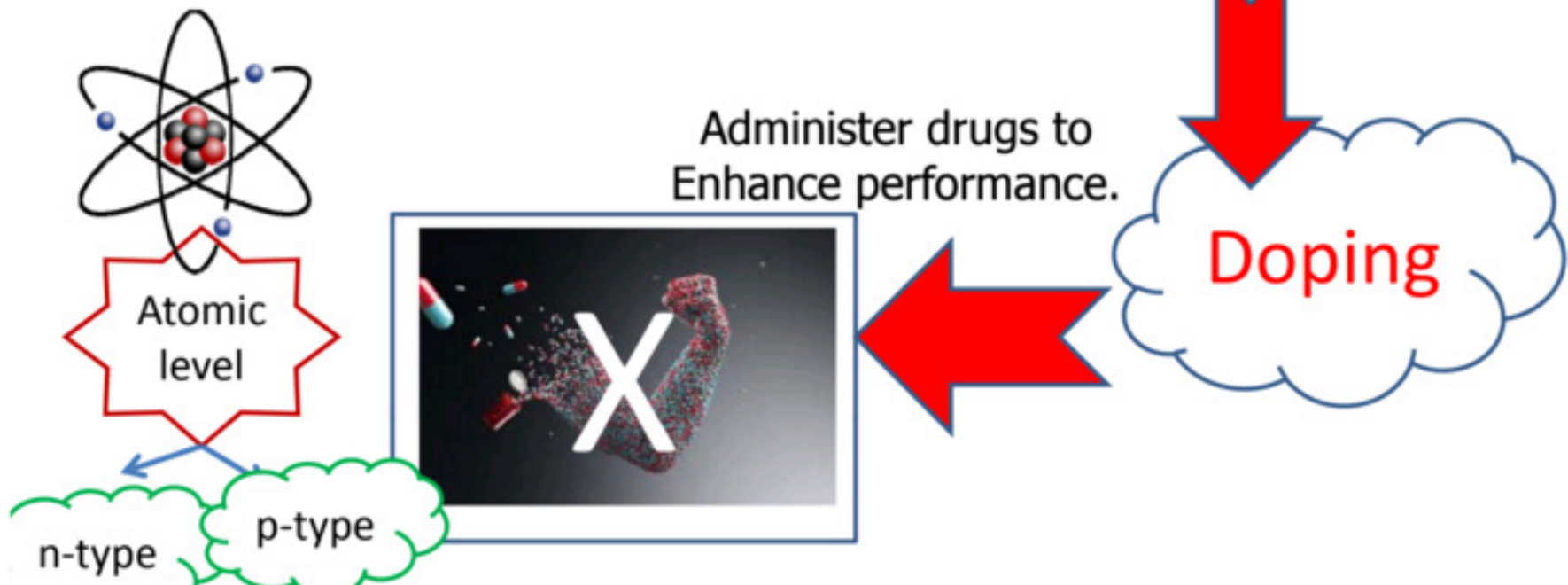


Atom of silicon has two (2) inner, eight(8) middle and four (4) electrons at outer

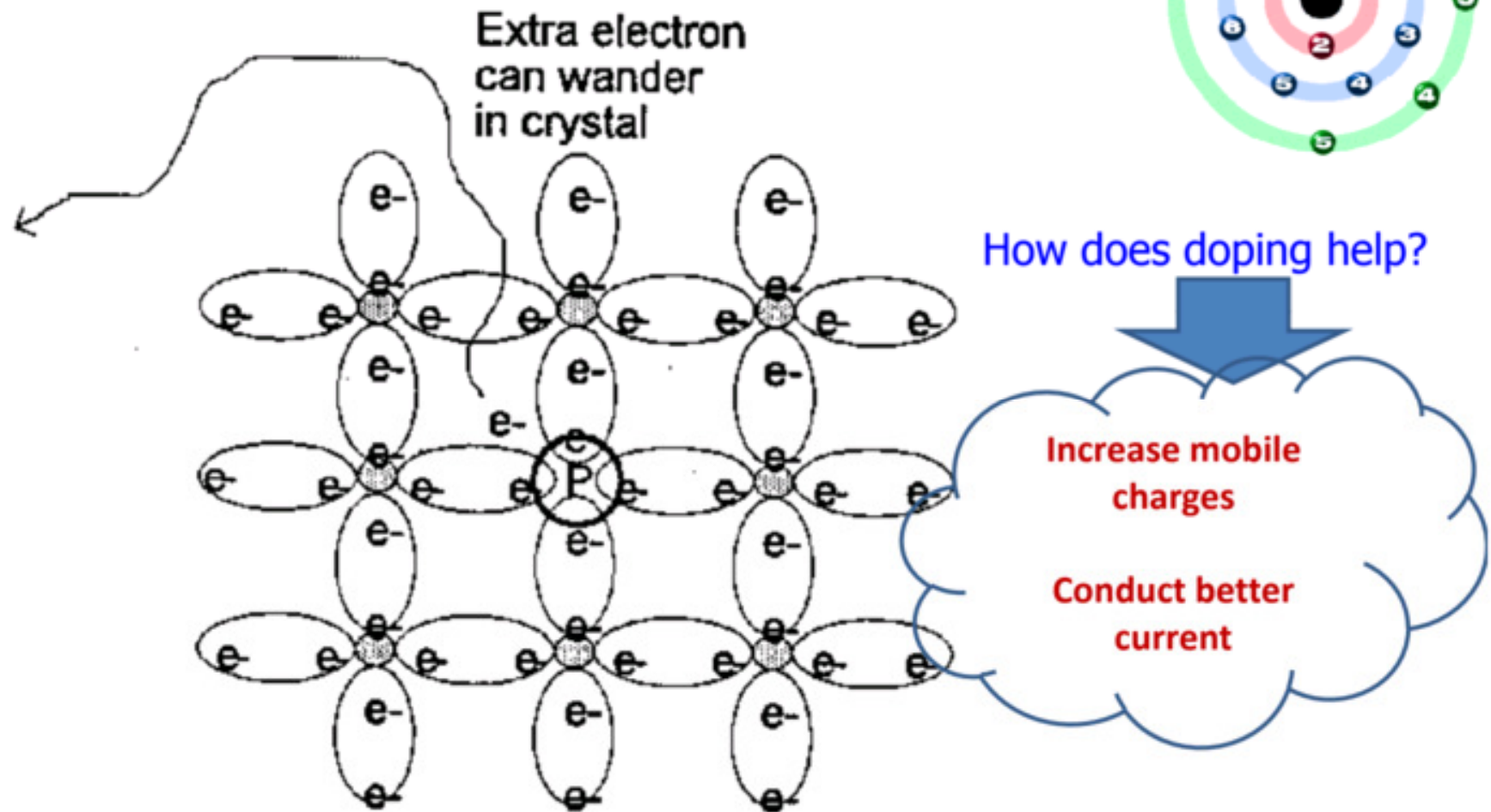
Why is Silicon called semi-conductor?



- Few electrons from the bond get potential energy to escape away.
 - Creates small# of mobile charges
 - Creates opportunity for doping



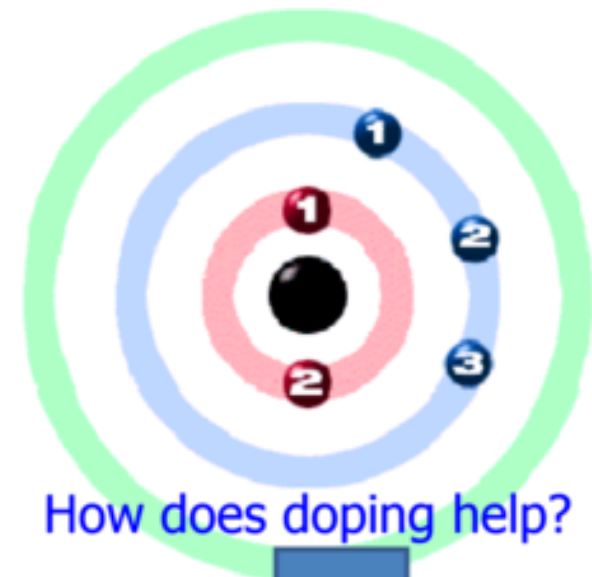
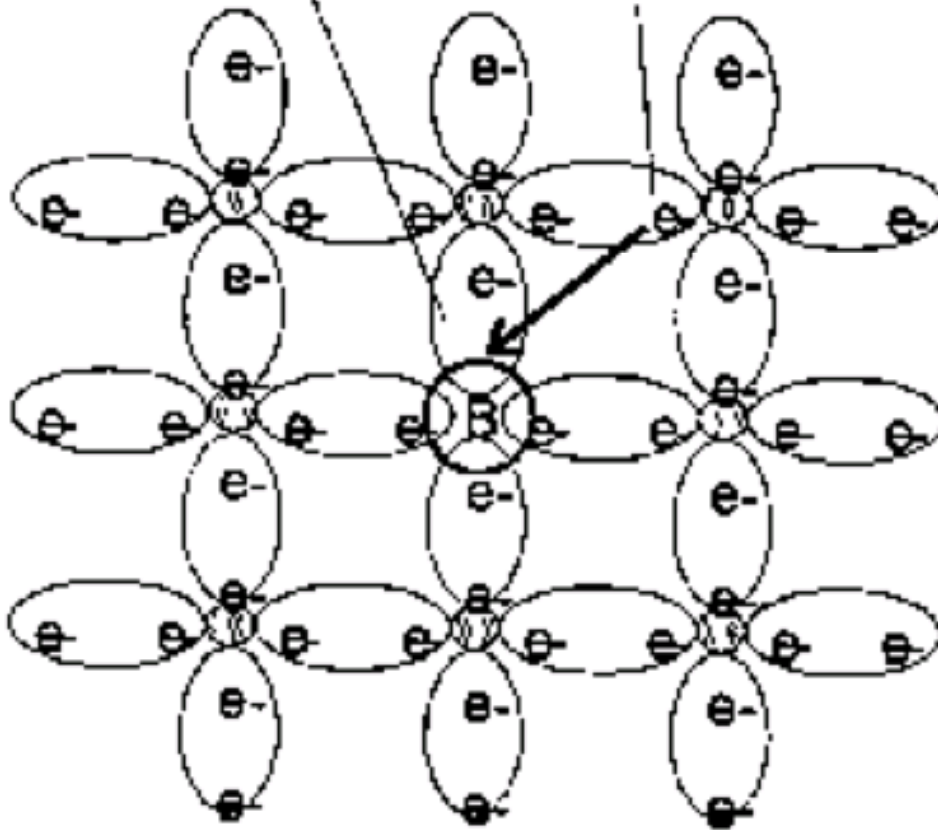
Phosphorous N-Type Doping



Boron P-Type Doping

Deficit of one electron leaves "hole"

Nearby electrons can move in and "fill" hole.

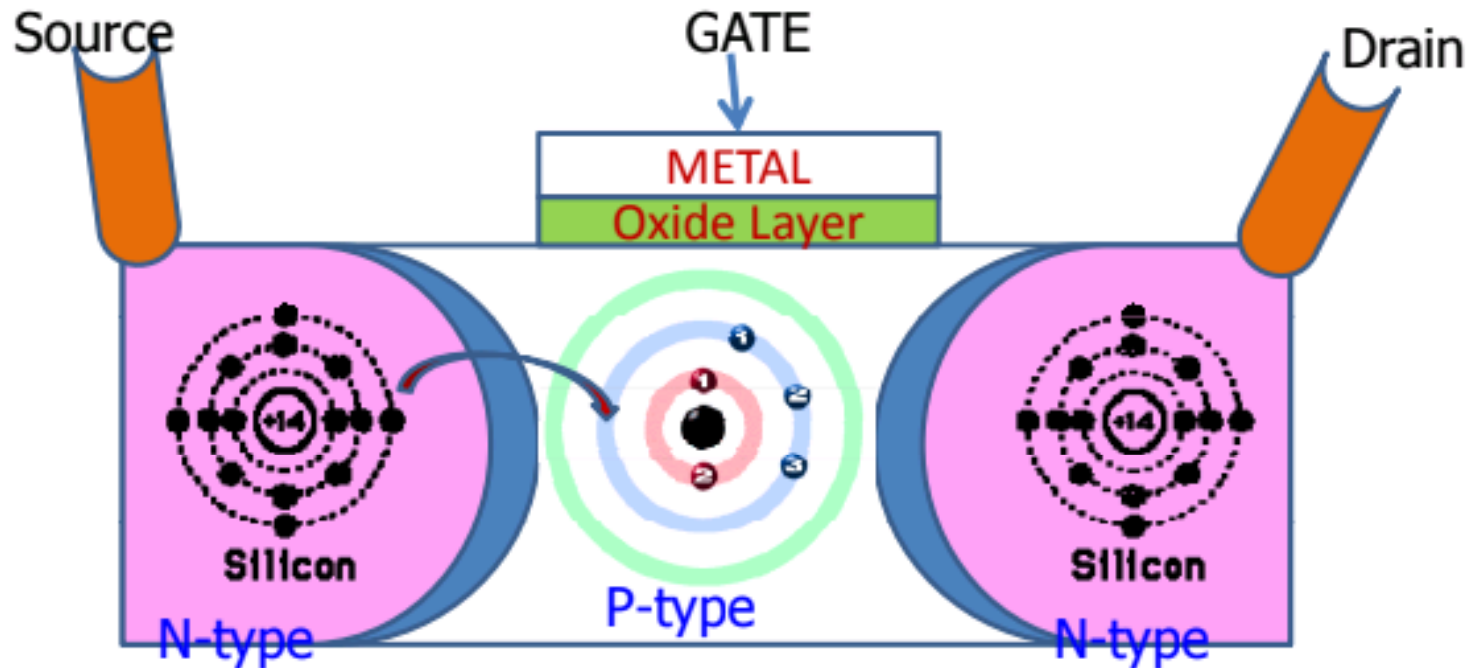


How does doping help?

Not only electrons (-) move but also holes (+) can also move

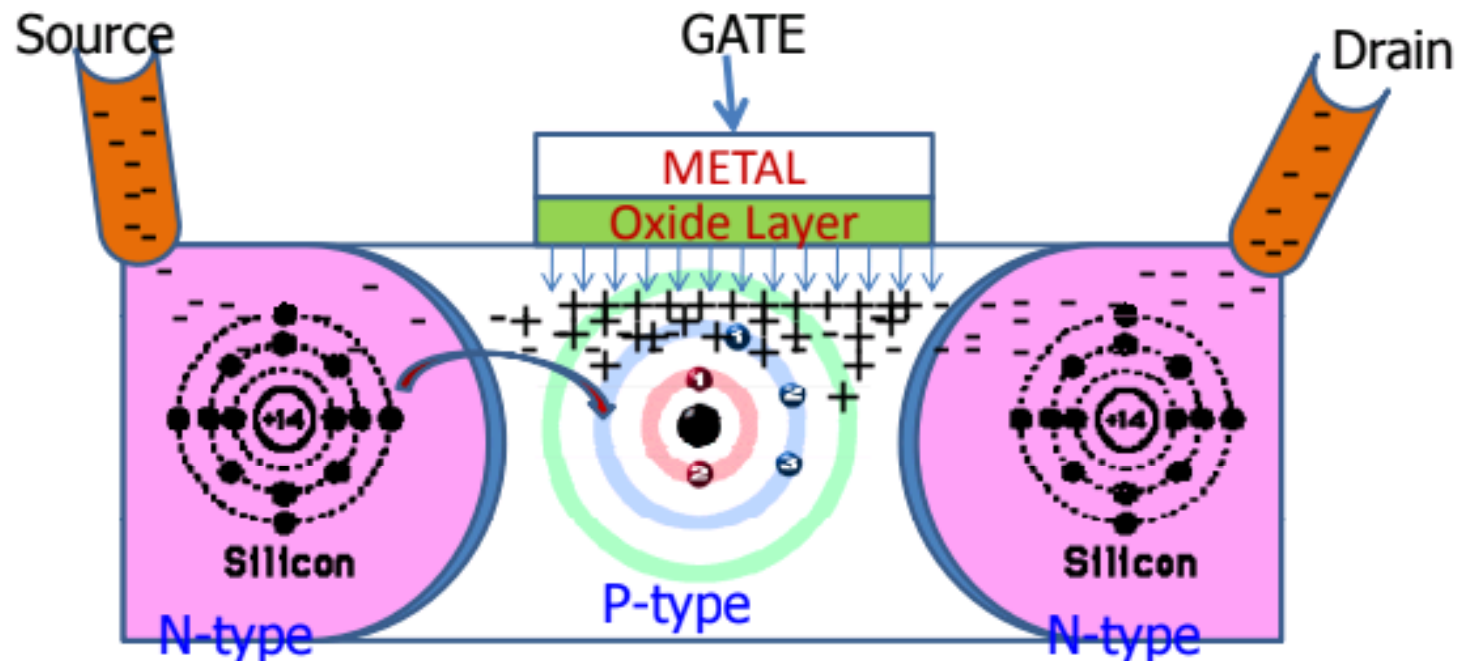
Conduct better electricity

Transistor State OFF



- Electron flow from N-type to P-type and filled the holes
- called Depletion layer
 - Makes P-type negative (-) by adding more electrons
 - So no more electrons passing (similar state)

Transistor State ON



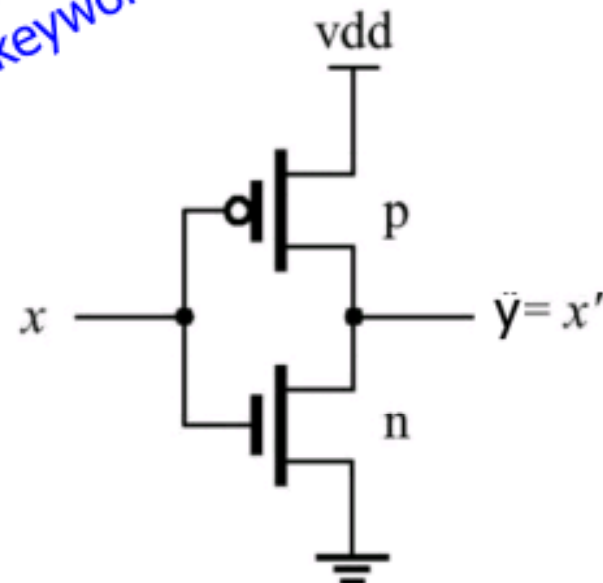
- Gate ON – flow +ve voltage to gate
 - Creates holes in p-type
 - attracts electrons to move
- Shrinks Depletion layer (electrons can move)
 - Conducting electrical flow from source to Drain

CMOS-Complementary MOS

```
module cmos_inverter (input x, output y);  
  supply1 vdd;  
  supply0 gnd;  
  pmos p1(y, vdd, x);  
  nmos n1(y, gnd, x);  
endmodule
```

*Power and Ground sources are
defined with supply keyword*

```
module Top;  
  reg x;  
  wire y;  
  cmos_inverter c1 (x, y);  
  initial  
  begin  
    #1 x = 0;  
    #2 $display(" x = %b, y= %b\n", x, y);  
    #3 x = 1;  
    #4 $display(" x = %b, y= %b\n", x, y);  
  end  
endmodule
```



(a) Circuit

CMOS Nand Gate

```
module cmos_nand (Out,A,B);
```

```
input A; input B;
```

```
output Out;
```

```
wire C;
```

```
supply1 Vdd;
```

```
supply0 Vss;
```

```
pmos p1(Out,A,Vdd);
```

```
pmos p2(Out,B,Vdd);
```

```
nmos n1(Out,A,C);
```

```
nmos n2(C,Vss,B);
```

```
endmodule
```

```
module Top;
```

```
reg A, B;
```

```
wire Out;
```

```
cmos_nand c1(Out, A, B);
```

```
initial
```

```
begin
```

```
    A = 1'b0; B = 1'b0;
```

```
    #5 A = 1'b0; B = 1'b1;
```

```
    #5 A = 1'b1; B = 1'b0;
```

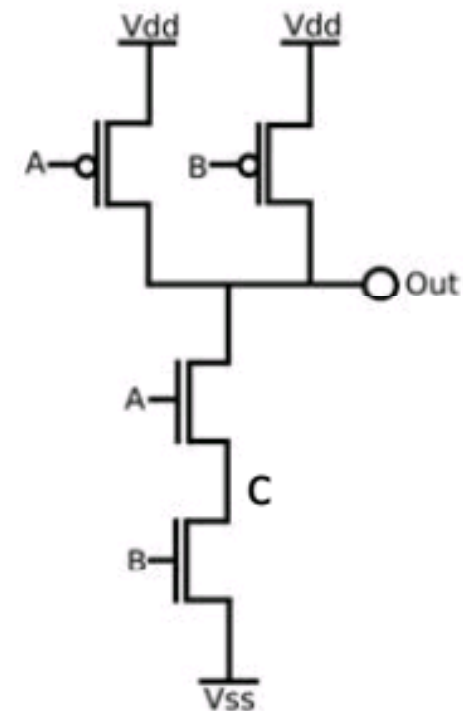
```
    #7 A = 1'b1; B = 1'b1;
```

```
end
```

```
initial
```

```
    $monitor("out=%b, A = %b, B= %b\n", Out, A, B);
```

```
endmodule
```



CMOS NOR Gate

```
module cmos_nor (out,a,b);
```

```
input a,b;  
output out;  
wire c;
```

```
supply1 Vdd;  
supply0 Vss;
```

```
pmos (c,Vdd,b);  
pmos (out,c,a);  
nmos (out,Vss,a);  
nmos(out,Vss,b);
```

```
endmodule
```

```
module Top;
```

```
reg A, B;
```

```
wire Out;
```

```
cmos_nor c1(Out, A, B);
```

```
initial
```

```
begin
```

```
A = 1'b0;
```

```
B = 1'b0;
```

```
#5 A = 1'b0; B = 1'b1;
```

```
#5 A = 1'b1; B = 1'b0;
```

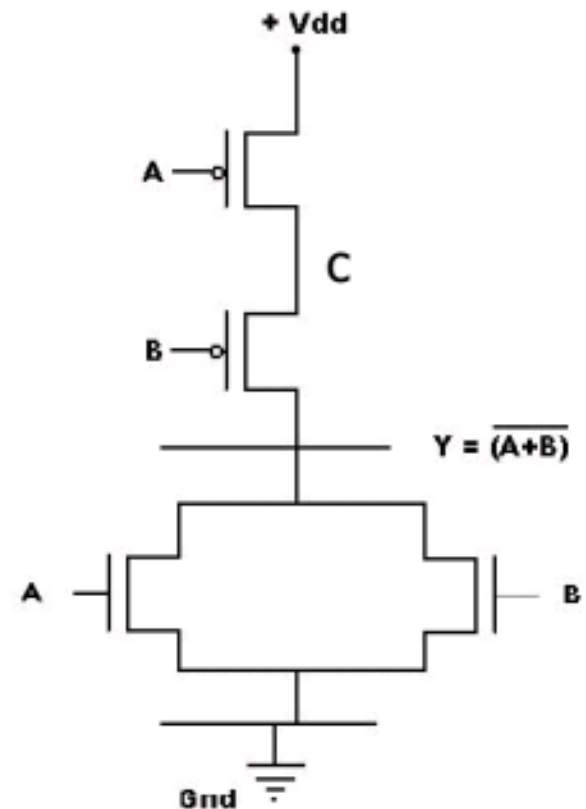
```
#7 A = 1'b1; B = 1'b1;
```

```
end
```

```
initial
```

```
$monitor("out=%b, A = %b, B= %b\n", Out, A, B);
```

```
endmodule
```



CMOS 2-to-1 Multiplexor

```
module cmos_mux (out, s, i0, i1);
```

```
output out;  
input s, i0, i1;
```

```
wire sbar;
```

```
cmos_nor nt(sbar, s, s);
```

```
cmos (out, i0, sbar, s);  
cmos (out, i1, s, sbar);
```

```
endmodule
```

```
module Top;
```

```
reg i0, i1, s;
```

```
wire out;
```

```
cmos_mux cm(out, s, i0, i1)
```

```
initial
```

```
begin
```

```
    i0 = 1'b0;
```

```
    i1 = 1'b1;
```

```
    #5 s = 1'b0;
```

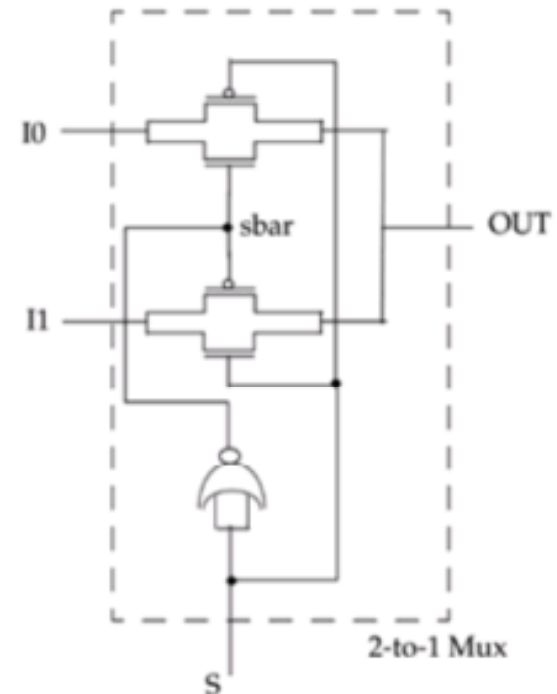
```
    #5 s = 1'b1;
```

```
end
```

```
initial
```

```
    $monitor("out=%b, S = %b\n", out, s);
```

```
endmodule
```



CMOS Latch

```
module cmos_latch (q, qbar, d, clk);  
    output q, qbar;  
    input d, clk;  
    wire e;  
    wire nclk;  
        cmos_inverter nt(nclk,clk);  
        cmos cm1(e, d, clk, nclk);  
        cmos cm2(e, q, nclk, clk);  
        cmos_inverter nt1(qbar,e);  
        cmos_inverter nt2(q,qbar);  
endmodule
```