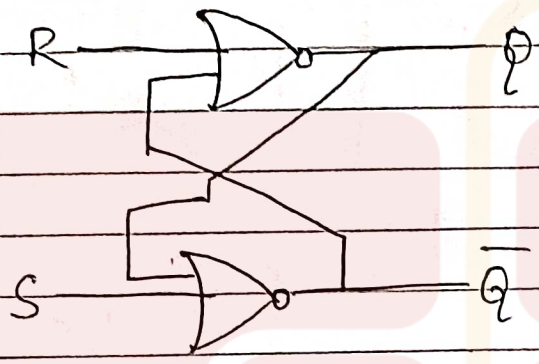
VHDL.

Module - 4
Flip Flops & Latches

The basic storage element is called latch.

As the time name suggest its latches (0, 1)

V.Imp * Design a ckt for S-R latch using NOR (+) gate



A	B	y
0	0	1
0	1	0
1	0	0
1	1	0

NOR gate truth table

Case 1 S=0, R=1 Q=0 $\overline{Q}$=1
S=0 R=0 Q=0 $\overline{Q}$=1 → memory

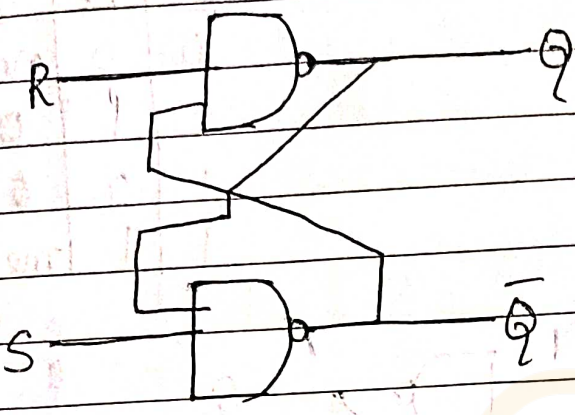
Case 2 S=1 R=0 Q=1 $\overline{Q}$=0
S=0 R=0 Q=1 $\overline{Q}$=0 → memory

Case 3 S=1 R=1 Q=0 $\overline{Q}$=0
Not used
S=0 R=0 Q=1 $\overline{Q}$=0 → memory

S	R	Q	$\overline{Q}$
0	0	memory	
0	1	0	1
1	0	1	0
1	1	Not used	

(*)

* Design a ckt for S-R latch using NAND gate



A	B	y
0	0	1
0	1	1
1	0	1
1	1	0

NAND gate

Case 1 S=0 R=1 Q=0 $\bar{Q}$=1
S=1 R=1 Q=0 $\bar{Q}$=1 → memory

Case 2 S=1 R=0 Q=1 $\bar{Q}$=0
S=1 R=1 Q=0 $\bar{Q}$=1 → memory

Case 3 S=1 R=1 Q=0 $\bar{Q}$=1
S=0 R=0 Q=1 $\bar{Q}$=1
not used

S	R	Q	$\bar{Q}$
0	0	Not used	
0	1	0	1
1	0	1	0
1	1	memory	

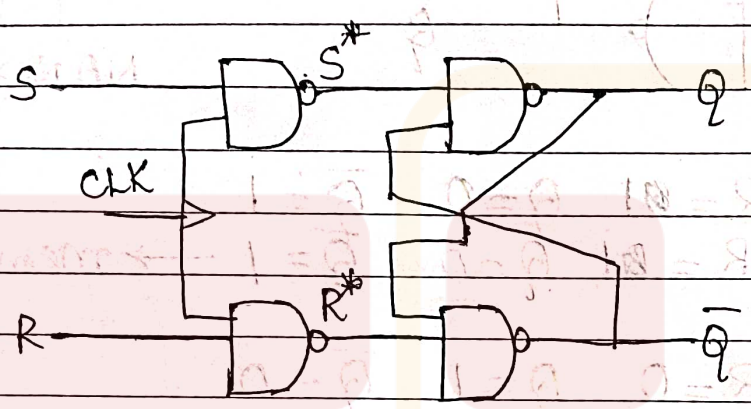
—>→ edge triggered / +ve
— → level —11—
—o>→ -ve —11—

VVIMP Pakka ✗ **SR Flip Flop**

$\bar{Q}$ na thagobeku

S^*	R^*	Q_{n+1} (NS)
0	0	not used
0	1	0
1	0	1
1	1	memory (Q_n) (PS)

I



II $S^* = (S.\overline{CLK}) = \bar{S} + \overline{CLK}$

$R^* = (R.\overline{CLK}) = \bar{R} + \overline{CLK}$

Case 1: $S=0, R=0, CLK=1$

$S^* = 1+0 = 1$

$R^* = 1+0 = 1$

Case 2: $S=0, R=1, CLK=1$

$S^* = 1+0 = 1$

$R^* = 0+0 = 0$

Case 3: $S=1, R=0, CLK=1$

$S^* = 0+0 = 0 \quad R^* = 1+0 = 1$

Case 4: $S=1, R=1, CLK=1$

$S^* = 0+0 = 0$

$R^* = 0+0 = 0$

V.Imp

* SR Flipflop

S	R	Q	$\bar{Q}$
0	0	Not used	
0	1	0	1
1	0	1	0
1	1	memory (Q_n)	

I

CLK	S	R	Q_{n+1}
0	X	X	Q_n
1	0	0	Q_n
1	0	1	0
1	1	0	1
1	1	1	Not ~~used~~ used

$S^* = (S \cdot \overline{CLK}) = \bar{S} + \overline{CLK}$

$R^* = (R \cdot \overline{CLK}) = \bar{R} + \overline{CLK}$

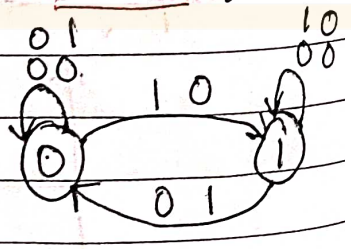
CI

Q_n	S	R	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	X
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	X

ET

Q_n	Q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

STD ($Q_n \rightarrow Q_{n+1}$)



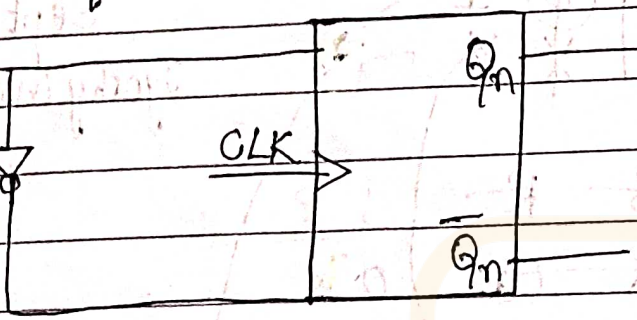
Q_n \ SR	$\bar{S}\bar{R}$	$\bar{S}R$	SR	$S\bar{R}$
$\bar{Q}_n$	0 0	0 1	X 3	1 2
Q_n	1 4	0 5	X 7	1 6

$$Q_{n+1} = S + Q_n \bar{R}$$

Pakka

D - Flipflop D → data

I



CLK	S	R	Qn+1
0	X	X	Qn
0	0	0	Qn
1	0	1	0
1	1	0	1
1	1	1	Invalid

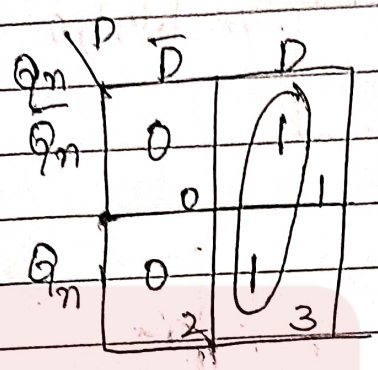
CLK	D	Qn+1
0	X	Qn
1	0	0
1	1	1

CT

Qn	D	Qn+1
0	0	0
0	1	1
1	0	0
1	1	1

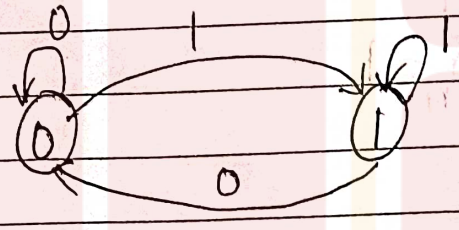
CT → characteristic table
ET → esitation table

ET	Q_n	Q_{n+1}	D
	0	0	0
	0	1	1
	1	0	0
	1	1	1



$$Q_{n+1} = D$$

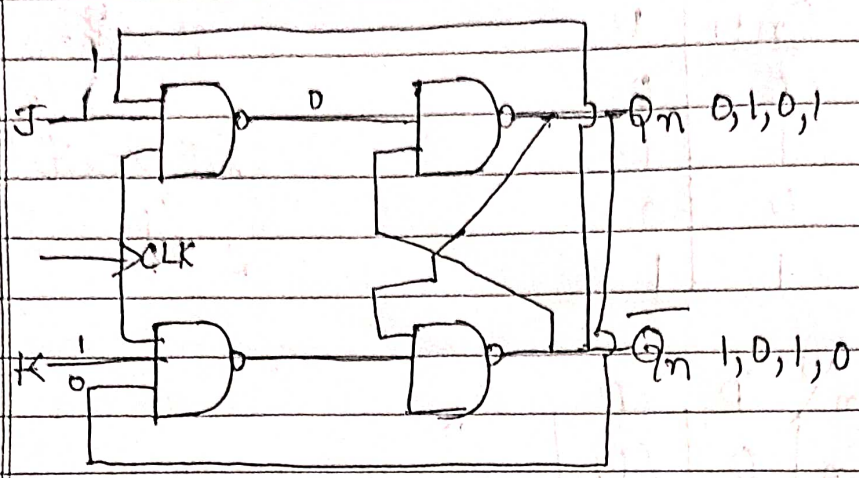
STD $Q_n \rightarrow Q_{n+1}$



v.v.imp
Pakka ✗ JK Flip Flop

CLK	S	R	Q_{n+1}
0	X	X	Q_n
1	0	0	Q_n
1	0	1	0
1	1	0	1
1	1	1	Invalid

X



CLK	J	K	Qn+1
0	X	X	Qn
1	0	0	Qn
1	0	1	0
1	1	0	1
1	1	1	Toggle($\overline{Q_n}$)

I & III $J=1, K=1, CLK=1$

Assume $Q_n=0, \overline{Q_n}=1$

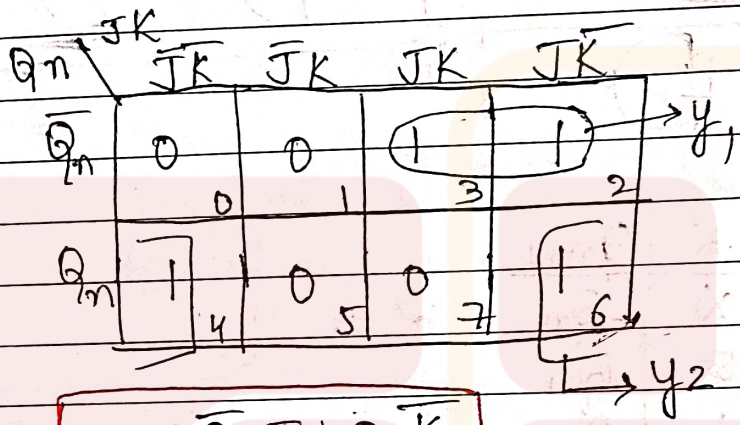
II CT (TT)

Q_n	J	K	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

III ET (CT)

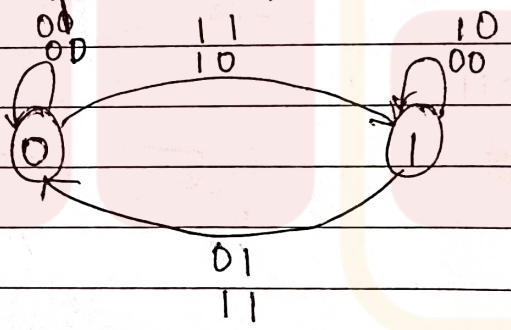
Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

IV



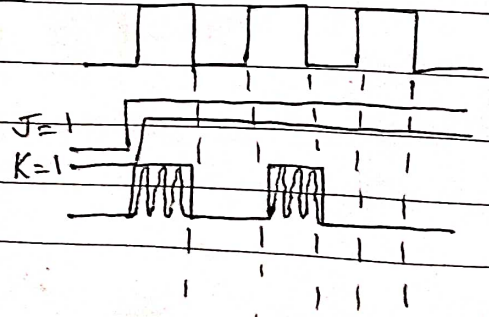
$$Q_{n+1} = \bar{Q}_n J + Q_n \bar{K}$$

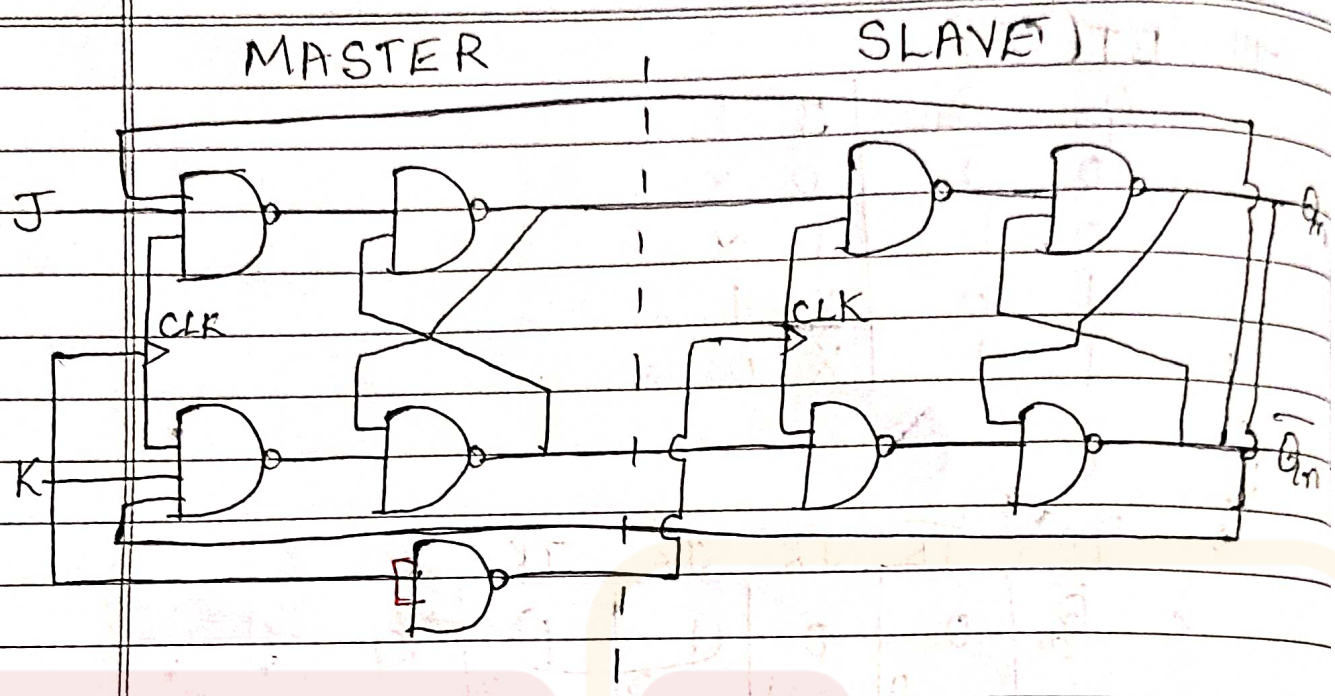
V STD ($Q_n \rightarrow Q_{n+1}$)



Imp Pakka * JK Master/Slave

Graph





I

CLK	J	K	Q_{n+1}
0	X	X	Q_n
1	0	0	Q_n
1	0	1	0
1	1	0	1
1	1	1	$\overline{Q_n}$

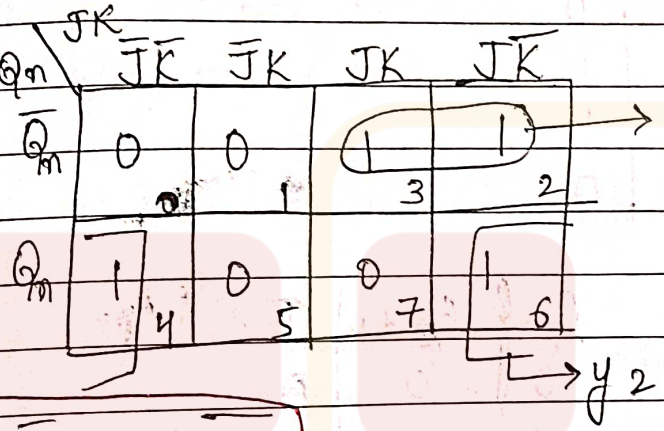
II CT

Q_n	J	K	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

III ET(CT)

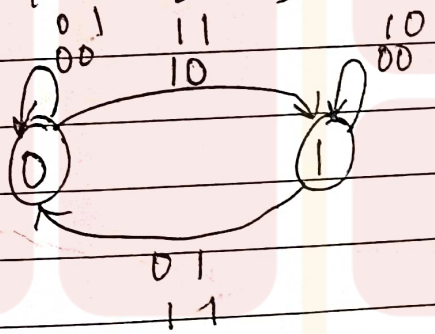
Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

IV

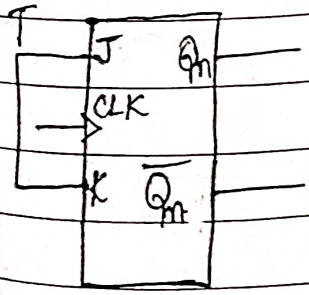


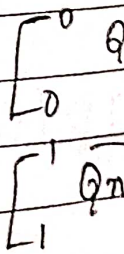
$$Q_{n+1} = \bar{Q}_n J + Q_n \bar{K}$$

V STD ($Q_n \rightarrow Q_{n+1}$)



* T - Flipflop





J K

0 0 → Q_n

1 1 → $\bar{Q}_n$

CLK	T	Q_{n+1}
0	X	Q_n
1	0	Q_n
1	1	$\overline{Q_n}$

CT

Q_n	T	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

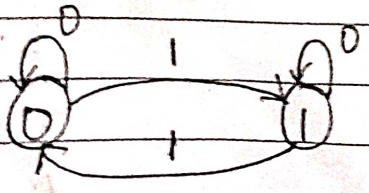
ET :-

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Q_n \ T	$\overline{T}$	T
$\overline{Q_n}$	0 (0)	1 (1)
Q_n	1 (2)	0 (3)

$$Q_{n+1} = Q_n\overline{T} + \overline{Q_n}T$$
$$= Q_n \oplus T$$

STD ($Q_n \rightarrow Q_{n+1}$)



Dr KISHORE G R
Associate Professor
Dept-of ISE
Jyothy Institute of Technology

Conversion of Flipflops

Convert JK to D

Step-1 Identify available & required flipflop
2 Make a characteristic table for required flip flop
3 Make a ET for available flipflop
4 Combine require flipflop with available flipflop
5 Write a boolean expression for available flipflop
6 Draw a circuit

I Available FF → JK
Required FF → D

II

Qn	D	Qn+1
0	0	0
0	1	1
1	0	0
1	1	1

III

Qn	Qn+1	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

IV

Required FF		Available FF		
Q_n	D	Q_{n+1}	J	K
0	0	0	0	X
0	1	1	1	X
1	0	0	X	1
1	1	1	X	0

V K-MAP (AV FF)

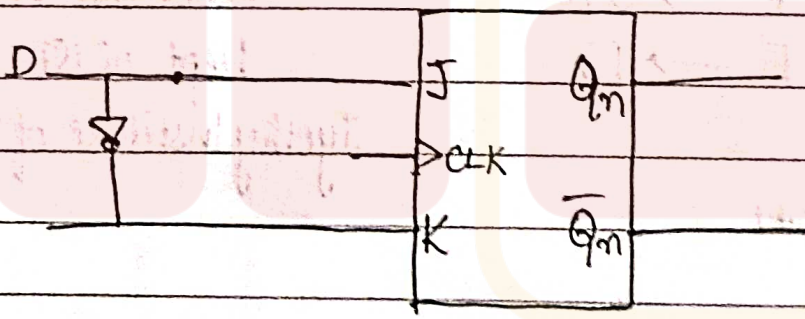
Q_n \ D	$\bar{D}$	D
$\bar{Q}_n$	0	1
Q_n	X	X

$J = D$

Q_n \ D	$\bar{D}$	D
$\bar{Q}_n$	X	X
Q_n	1	0

$K = \bar{D}$

VI



* Convert SR to JK.

I Available FF → SR
Required FF → JK

II

Qn	J	K	Qn+1
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

III

Qn	Qn+1	S	R
0	0	X	0
0	1	0	1
1	0	1	0
1	1	0	X

IV

Qn	J	K	Qn+1	S	R
0	0	0	0	X	0
0	0	1	0	X	0
0	1	0	1	0	1
0	1	1	1	0	1
1	0	0	1	0	X
1	0	1	0	1	0
1	1	0	1	0	X
1	1	1	0	1	0

V K-MAP

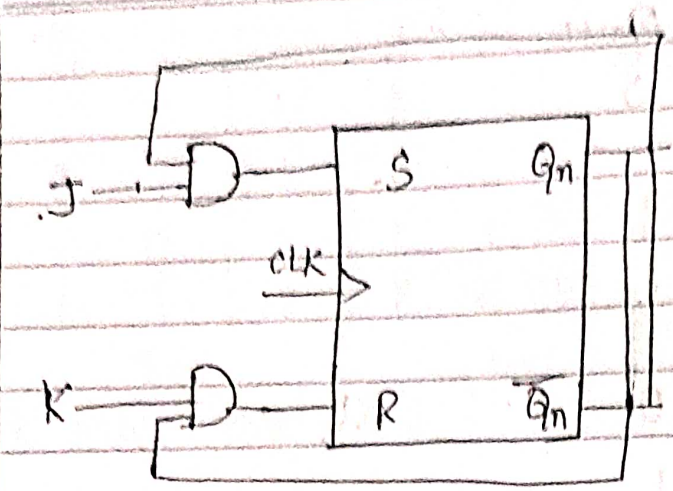
Qn \ JK	$J\bar{K}$	$\bar{J}\bar{K}$	$\bar{J}K$	JK
$\bar{Q}n$	X_0	X_1	0_3	0_2
Qn	0_4	1_5	1_7	0_6

$S = Q_n \cdot \bar{K}$

Qn \ JK	$\bar{J}\bar{K}$	$\bar{J}K$	JK	$J\bar{K}$
$\bar{Q}n$	0_0	0_1	1_3	1_2
Qn	X_4	0_5	0_7	X_6

$R = \bar{Q}_n J$

VI



* Convert SR to D

I Available FF → SR
Required FF → D

II

Qn	D	Qn+1
0	0	0
0	1	1
1	0	0
1	1	1

III

Qn	Qn+1	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

IV

Qn	D	Qn+1	S	R
0	0	0	0	X
0	1	1	1	0
1	0	0	0	1
1	1	1	X	0

V

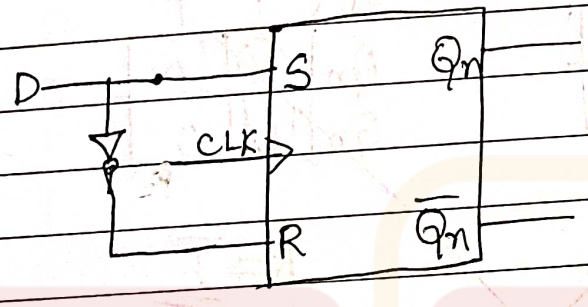
Q_n \ D	$\bar{D}$	D
$\bar{Q}_n$	0	1
Q_n	0	X

$S = D$

Q_n \ D	$\bar{D}$	D
$\bar{Q}_n$	X	0
Q_n	1	0

$R = \bar{D}$

VI



✗ SR to T:

I Avai FF → SR
Req FF → T

II

Q_n	T	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

III

Q_n	Q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

IV

Q_n	T	Q_{n+1}	S	R
0	0	0	0	X
0	1	1	1	0
1	0	1	X	0
1	1	0	0	1

V

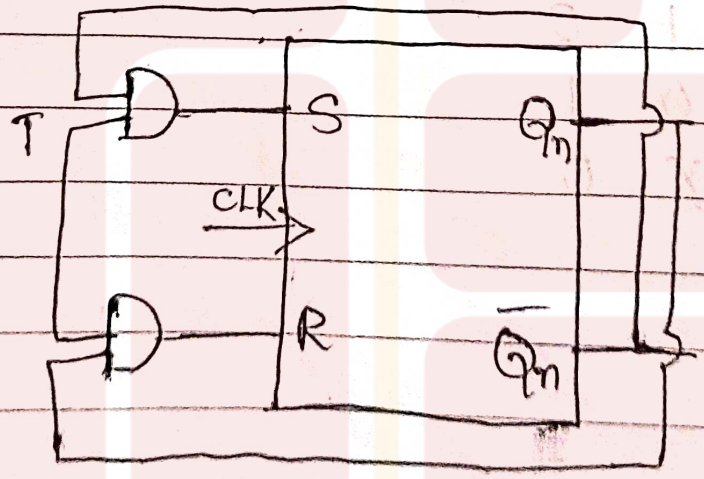
Q_n \ T	$\bar{T}$	T
$\bar{Q}_n$	0	(1)
Q_n	X	0

$S = \bar{Q}_n T$

Q_n \ T	$\bar{T}$	T
$\bar{Q}_n$	X	0
Q_n	0	(1)

$R = Q_n T$

VI



Date
Page

* D to JK

I Avai FF → D
Req FF → JK

II

Qn	J	K	Qn+1
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

III

Qn	Qn+1	D
0	0	0
0	1	1
1	0	0
1	1	1

IV

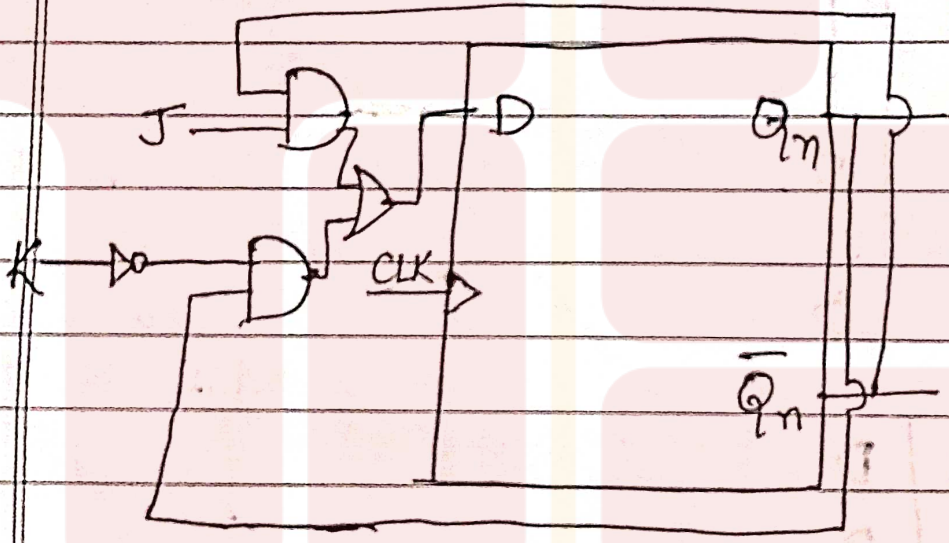
Qn	J	K	Qn+1	D
0	0	0	0	0
0	0	1	0	0
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	0	0
1	1	0	1	1
1	1	1	0	0

V

Q_n \ JK	$\bar{J}\bar{K}$	$\bar{J}K$	$J\bar{K}$	JK
$\bar{Q}_n$	0 0	0 1	1 3	1 2
Q_n	1 4	0 5	0 7	1 6

$$D = \bar{Q}_n J + Q_n \bar{K}$$

VI

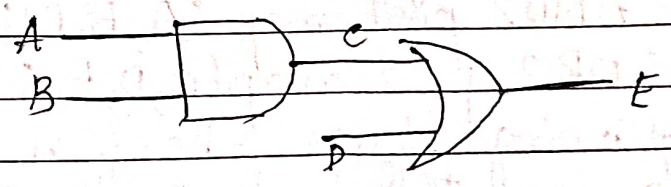


Introduction to VHDL

* VHDL stands for Very High Speed Integrated Circuit - Hardware Description Language.
* VHDL is a hardware description language i.e used to describe the behaviour & structure of digital system.
* VHDL is a genral - purpose hardware description language which can be used to describe & simulate the operation of the vaide variety of digital systems, ranging in complexity from a few gates to an interconnection of many complex Integrated circuits.

VHDL Description of Combinational Circuits

* In VHDL, a signal assignment statement of the form : signal name <= expression;
* The expression is evaluated when the statement is executed, & the signal on the left side is scheduled to change after delay.
* The square brackets indicate that after delay is optional. If after delay is ommited, then the signal is scheduled to be updated after the delta delay, Δ (Infinitesimal delay).
* A VHDL ~~delay~~ signal is used to describe a signal in a physical system.
* The VHDL language also includes variables, similar to variables in programming languages.



Gate Circuit.

Data Flow Description

* The two assignment statements give a data flow description of the above circuit, where it is assumed that each gate has a 5-ns propagation delay
* When this statements are simulated, the first statement will be evaluated any time A or B changes, & the second statement will be evaluated any time C or D changes.
* Suppose that initially A = 1, & B = C = D = E = 0; and if B changes to 1 at time 0, C will change to 1 at the time equals to 5ns. Then, E will change to 1 at time = 10ns. C <= A & B after 5ns; E <= C or D after 5ns;

Behavioural Description:

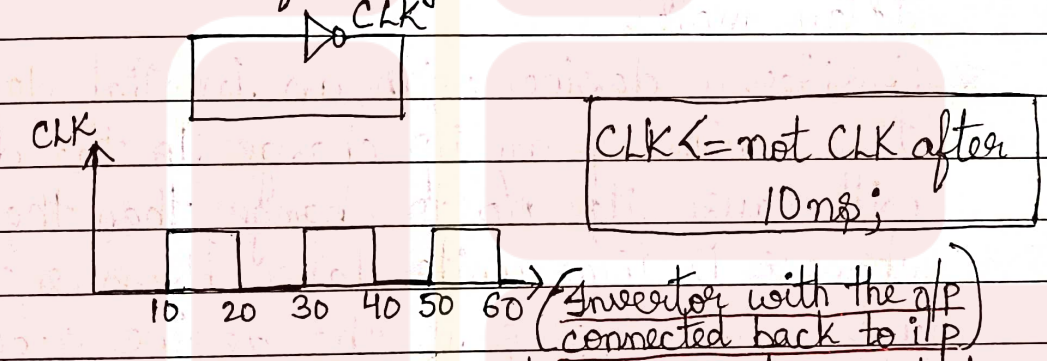
* A behavioural description of the above circuit shown is E <= D or (A & B); paranthesis are used to specify the order of operator execution

Structural Description

* The above ckt shown can also be described using structural VHDL code. To do so requires that a two-i/p AND gate component & a 2-input OR gate component be declared & defined
* The ckt shown is described by instantiating the AND & OR gate as follows:
Gate1: AND2 port map (A, B, D);
Gate2: OR2 port map (C, D, E);
* The following figure shows an inverter with the o/p connected back to the i/p. If the o/p

is "0", then this "0" feeds back to the i/p & the inverter o/p changes to "1" after the inverter delay, assumed to be 10 ns. Then, the "1" feeds back to the i/p, & the o/p changes to "0" after the inverter delay.

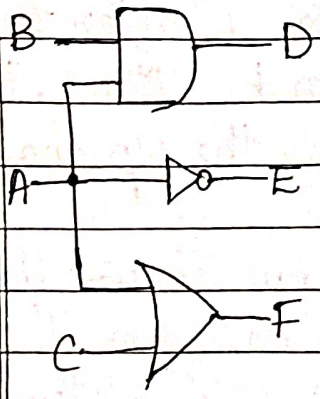
☆ The signal CLK will continue to oscillate b/w "0" & "1", as shown in the waveform. The corresponding concurrent VHDL statement will produce the same result. If CLK is initialized to "0", the statement executes & CLK changes to "1", after 10 ns. Because CLK has changed, the statement executes again, & CLK will change back to "0" after another 10 ns. This process will continue indefinitely.



☆ The following figure shows 3 gates that have the signal A as a common i/p & the corresponding VHDL code. The 3 concurrent statements execute simultaneously whenever a changes, just as the 3 gates start processing the signal change at the same time. However, if the gates have different delays, the gate o/p can change at different times.

☆ If the gates have delays of 2 ns, 1 ns & 3 ns, respectively & 'A' changes at time 5 ns, then the gate o/p 'D', 'E' & 'F' can change at time 7 ns, 6 ns, & 8 ns, respectively. However, if no delays were specified, then D, E & F would all be updated

at time 5+Δ




```
-- when A changes, these
concurrent statements all
execute at the same time
D<= A & B after 2ns;
E<= not A after 1ns;
F<= A or C after 3ns;
```

3 gates that have signal A as a common i/p & the corresponding VHDL code

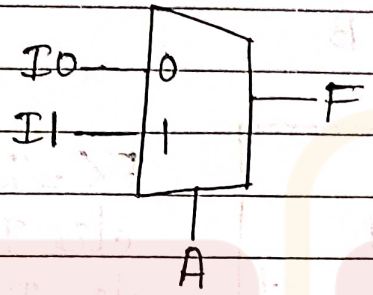
* Inertial delay model
* Signal assignment statements containing "after delay" create ~~what~~ is called an inertial delay model
* Consider a device with an inertial delay of D time units. If an i/p change to the device will cause its o/p to change, then the o/p changes D time units later. However, this is not what happens if the device receives 2 i/p changes within a period of D time units & both i/p changes should cause the o/p to change. In this case the device o/p does not change in response to either i/p change

* Ideal (Transport) delay
* VHDL can also model devices with an ideal (transport) delay
* O/p changes caused by i/p changes to a device exhibiting an ideal (transport) delay of D time units are delayed by D time units, & the o/p changes occur even if they occur within D time units

☆ VHDL signal assignment statement models ideal (transport) delay is signal name <= transport expression after delay

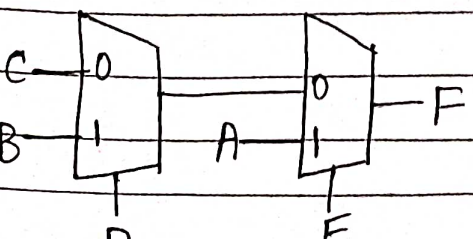
* VHDL Models for Multiplexers

i 2:1 multiplexer with 2 data i/p & one control i/p



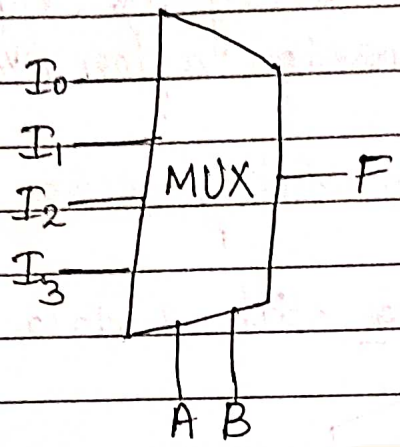
- The MUX o/p is = 'O+I. The corresponding VHDL statement is F<=(not A & I0) or (A & I1);
- Alternatively, we can represent the MUX by a conditional signal assignment statement, F<= I0 when A="0" else I1;
- This statement executes whenever A, I0 or I1 changes the MUX o/p is I0 when A="0", & else it is I1. In the conditional statement, I0, I1 & F can either be bits or bits-vectors
- Signal name <= expression 1 when condition 1 else expression 2 when condition 2
.
.
else expression N

* Cascaded MUX representation



F<= A when E="1"
else B when D="1"
else C;

ii 4:1 multiplexer




```
sel <= A & B;
--selected signal
assignment statement
with sel select
F <= I0 when "00";
     I1 when "01";
     I2 when "10";
     I3 when "11";
```

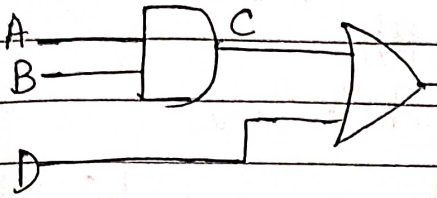
```
F <= I0 when A='0' & B='0'
else I1 when A='0' & B='1'
else I2 when A='1' & B='0'
else I3;
```

★ The general form of a selected signal assignment statement is

```
with expression_& select
signal_& <= expression 1 [after delay-time]
            when choice 1,
            expression 2 [after delay-time]
            when choice 2,
            ...
            [expression_n [after delay-time]
            when others];
```

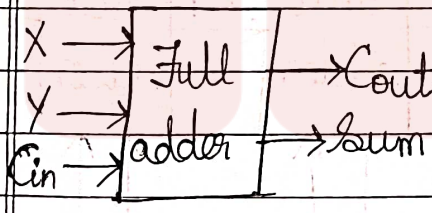
* VHDL Modules

To write a complete VHDL module, we must declare all of the i/p & o/p signals using an entity declaration, & then specify the internal operation of the module using an architecture declaration

Ex 


```
entity two_gates is
port (A, B, D: in bit; E:
      out bit);
end two_gates;
architecture gates of two_
gates is signal C: bit;
begin
C <= A & B; --concurrent
E <= C or D; --statements
end gates;
```

* Full adder architecture




```
entity FullAdder is
port (X, Y, Cin: in bit; --i/p
      Cout, Sum: out bit; --o/p
end FullAdder
```

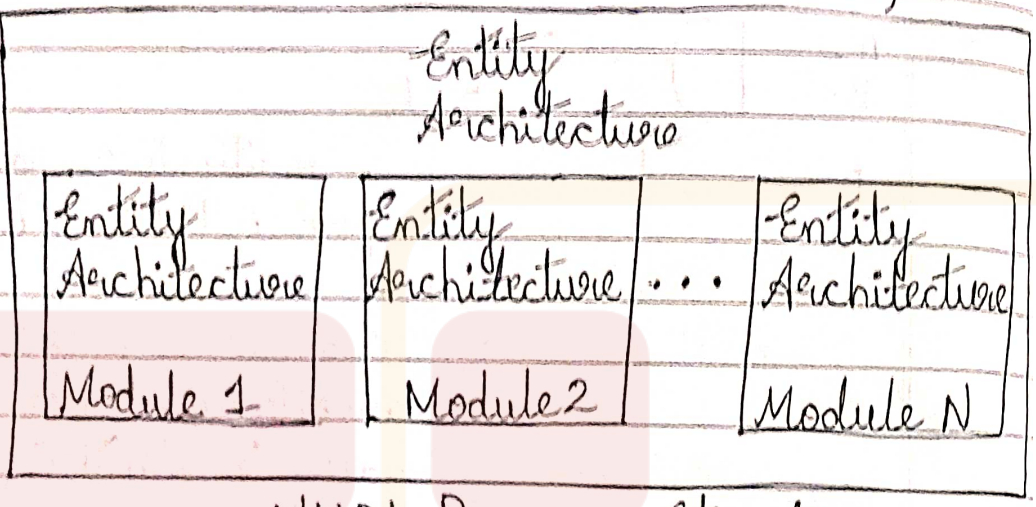
* Entity declaration of the form

```
entity entity-name is
[port (interface-signal-declaration);]
end [entity] [entity-name];
```

The items enclosed in [ ] are optional

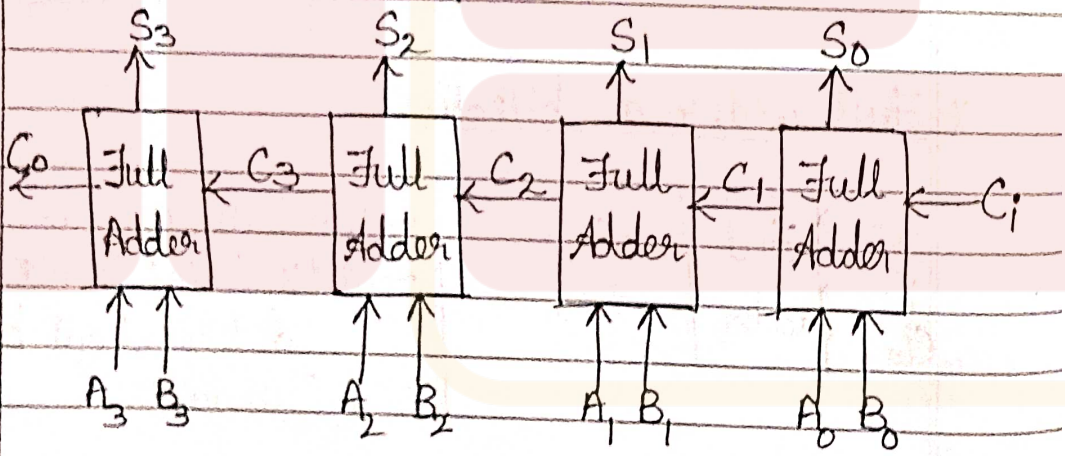
The interface signal declaration normally has the following form:

list-of-interface-signals : mode type [:= initial-value]
{; list-of-interface-signals : mode type [:= initial-value]};



VHDL Program Structure

* Four-Bit Full Adder




```
entity Adder4 is
  port(A,B: in bit_vector(3 downto 0); Ci: in bit;
                                        --i/p
       S: out bit_vector(3 downto 0); Co: out bit);
                                        --o/p
end Adder4;
architecture Structure of Adder4 is
```

```
component FullAdder
  port(X, Y, Cin: in bit; --I/p
    Cout, Sum: out bit); --o/p
end component;
signal C: bit_vector(3 downto 1);
begin --instantiate four copies of FullAdder
  FA0: FullAdder port map(A(0), B(0), Ci, C(1),
                          S(0));
  FA1: FullAdder port map(A(1), B(1), C(1), C(2),
                          S(1));
  FA2: FullAdder port map(A(2), B(2), C(2), C(3),
                          S(2));
  FA3: FullAdder port map(A(3), B(3), C(3), Co
                          S(3));
end Structure;
```