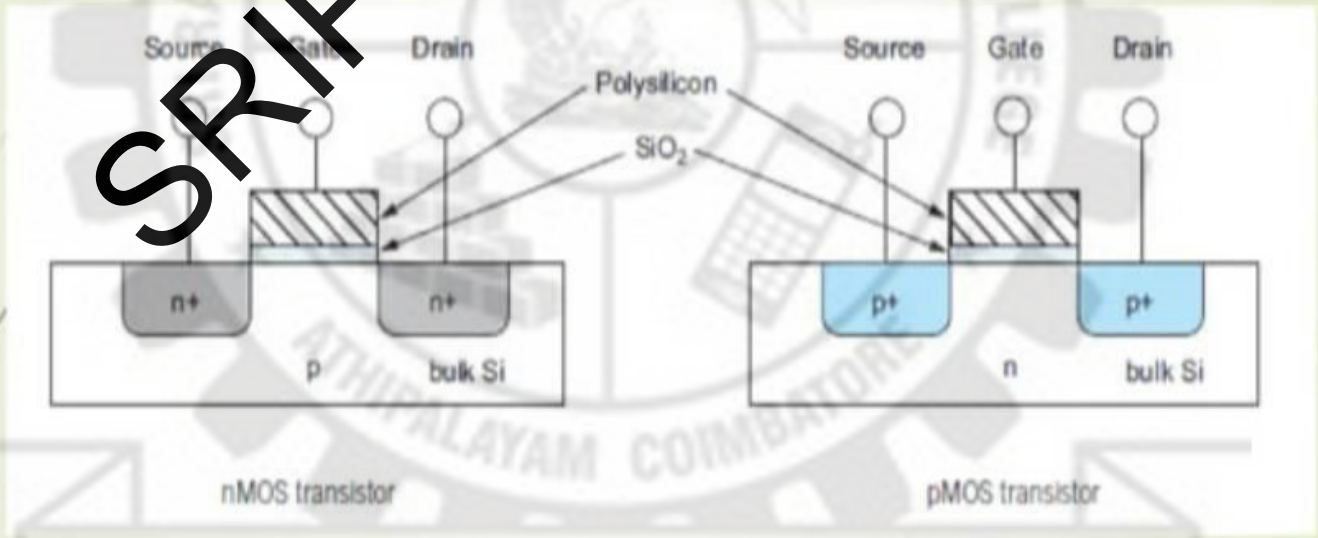
UNIT-I
INTRODUCTION TO VLSI

SWATHI E R
LECTURER/ECE

Introduction

- Metal Oxide-Semiconductor (*MOS) structure is created by superimposing several* layers of conducting and insulating materials to form a sandwich-like structure.
- These structures are manufactured using a series of chemical processing steps involving oxidation of the silicon, selective introduction of dopants, and deposition and etching of metal wires and contacts.
- CMOS technology provides two types of transistors: *an n-type transistor (nMOS) and a p-type* transistor (*pMOS).*
- *Transistor operation is controlled by electric fields so the devices are* also called Metal Oxide Semiconductor Field Effect Transistors (*MOSFETs)*



MOS Principle

- The gate is a control input: It affects the flow of electrical current between the source and drain.

N-channel MOSFET

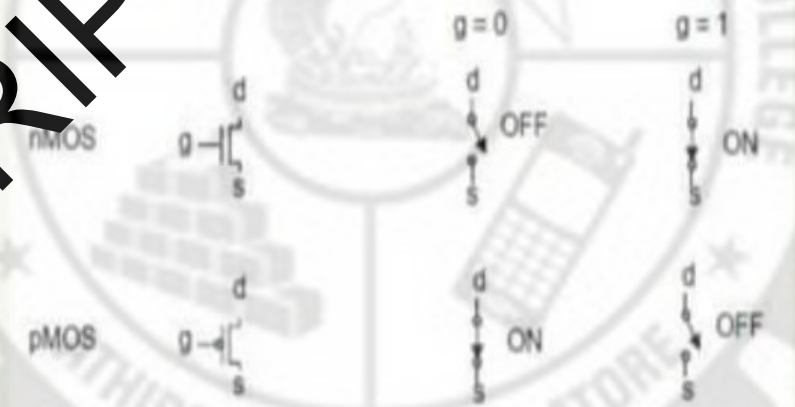
- Consider an nMOS transistor.
- The body is generally grounded so the p–n junctions of the source and drain to body are reverse-biased.
- If the gate is also grounded, no current flows through the reverse-biased junctions. Hence, we say the transistor is OFF.
- If the gate voltage is raised, it creates an electric field that starts to attract free electrons to the underside of the Si–SiO2 interface.
- If the voltage is raised enough, the electrons outnumber the holes and a thin region under the gate called the *channel is inverted to act as* an n-type semiconductor.
- Hence, a conducting path of electron carriers is formed from source to drain and current can flow. We say the transistor is ON.

P-channel MOSFET

- For a pMOS transistor, The body is held at a positive voltage.
- When the gate is also at a positive voltage, the source and drain junctions are reverse-biased and no current flows, so the transistor is OFF.
- When the gate voltage is lowered, positive charges are attracted to the underside of the Si–SiO2 interface.
- A sufficiently low gate voltage inverts the channel and a conducting path of positive carriers is formed from source to drain, so the transistor is ON.

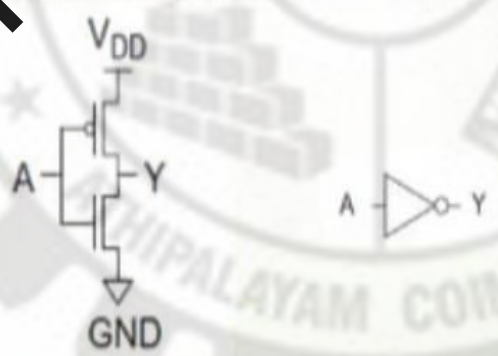
MOSFET as Switch

- The gate of an MOS transistor controls the flow of current between the source and drain
- Simplifying this to the extreme allows the MOS transistors to be viewed as simple ON/OFF switches.
- When the gate of an nMOS transistor is 1, the transistor is ON and there is a conducting path from source to drain.
- When the gate is low, the nMOS transistor is OFF and almost zero current flows from source to drain.
- A pMOS transistor is just the opposite, being ON when the gate is low and OFF when the gate is high.



CMOS Inverter

- When the input *A is 0, the nMOS transistor is OFF and* the pMOS transistor is ON. Thus, the output *Y is pulled up to 1 because it is connected to*
- *VDD but not to GND.*
- *Conversely, when A is 1, the nMOS is ON, the pMOS is OFF, and Y* is pulled down to 0.
- Schematic and Symbol for a CMOS inverter

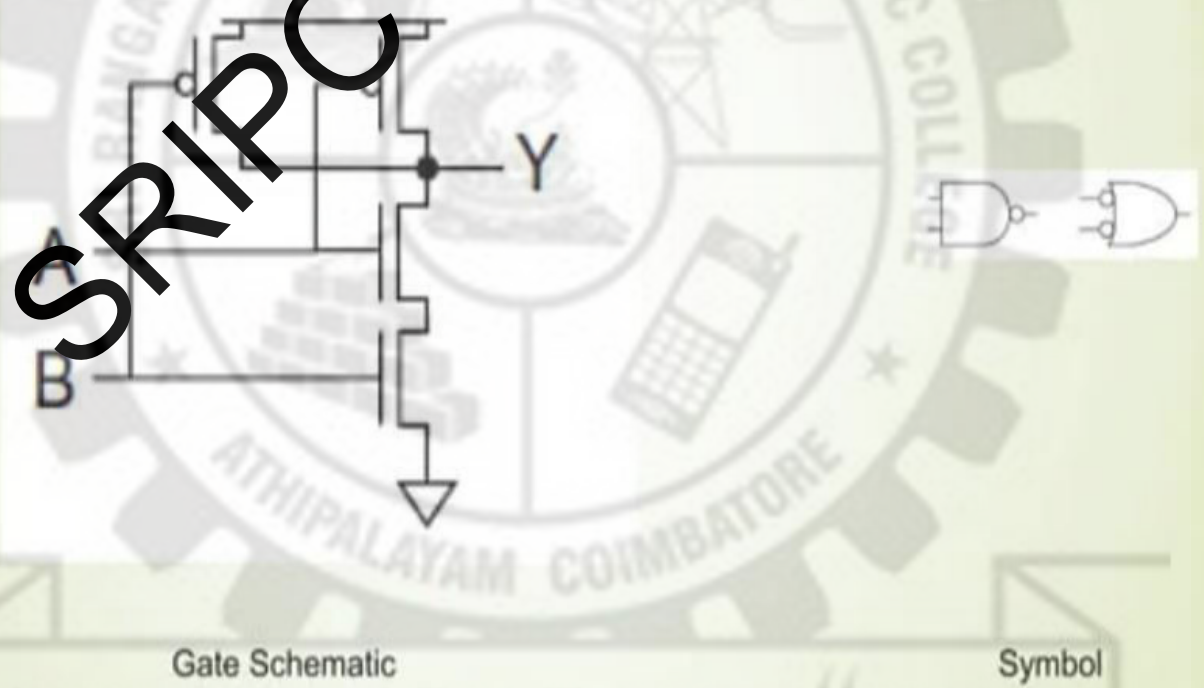


Inverter truth table

A	Y
0	1
1	0

- The bar at the top indicates *VDD and the triangle* at the bottom indicates GND.

CMOS NAND Gate

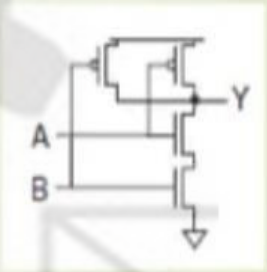


Gate Schematic

Symbol

CMOS NAND Gate

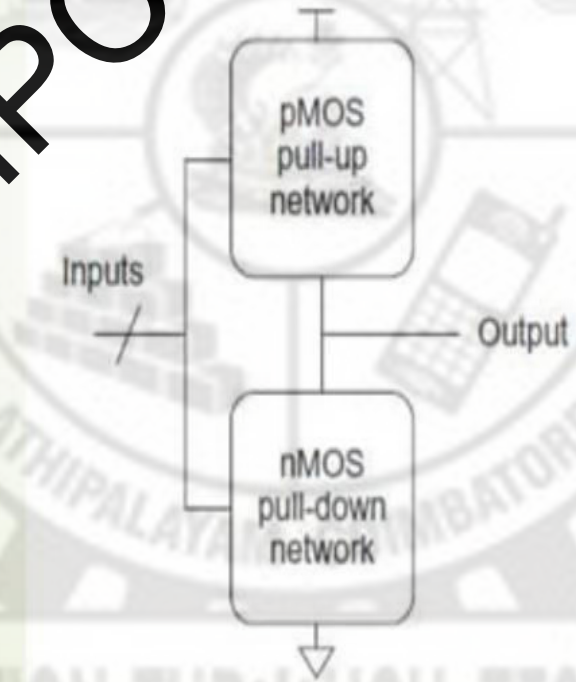
- It consists of two series nMOS transistors between *Y and GND and two parallel pMOS transistors between Y and VDD.*
- *If* either input *A or B is 0:*
  - *At least one of the nMOS transistors will be OFF, breaking the* path from *Y to GND.*
  - *But at least one of the pMOS transistors will be ON, creating a* path from *Y to VDD.*
  - *Hence, the output Y will be 1.*
- *If both inputs are 1,*
  - *both of the nMOS* transistors will be ON and
  - both of the pMOS transistors will be OFF.
  - Hence, the output will be 0.



CMOS NAND Gate

A	B	Pull-Down Network	Pull-Up Network	Y
0	0	OFF	ON	1
0	1	OFF	ON	1
1	0	OFF	ON	1
1	1	ON	OFF	0

CMOS Logic Structure

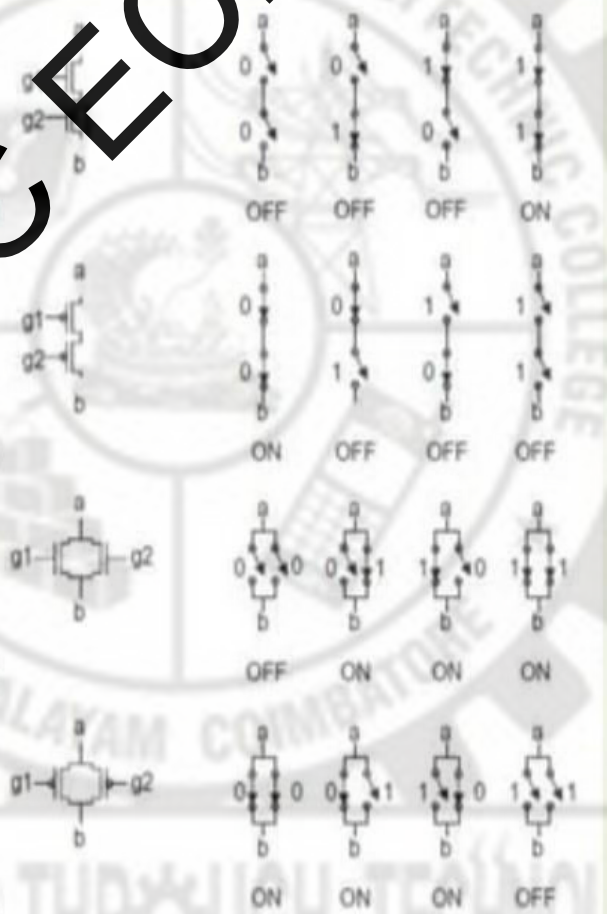


CMOS Logic Structure

- In general, a static CMOS gate has
  - an nMOS ***pull-down network*** *to* connect the output to 0 (GND) and
  - a PMOS ***pull-up network*** *to connect the output to 1 (VDD)*
- In general, when we join a pull-up network to a pull-down network to form a logic gate, they both will attempt to exert a logic level at the output.
- *The networks are arranged such that one is ON and the* other OFF for any input pattern.

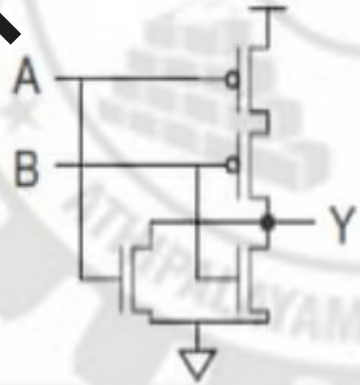
	pull-up OFF	pull-up ON
pull-down OFF	Z	1
pull-down ON	0	crowbarred (X)

Connection and behaviour of series and parallel transistors



CMOS NOR Gate

- The nMOS transistors are in parallel to pull the output low when either input is high.
- The pMOS transistors are in series to pull the output high when both inputs are low

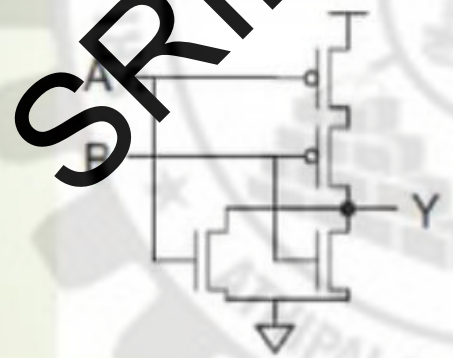


NOR gate truth table

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

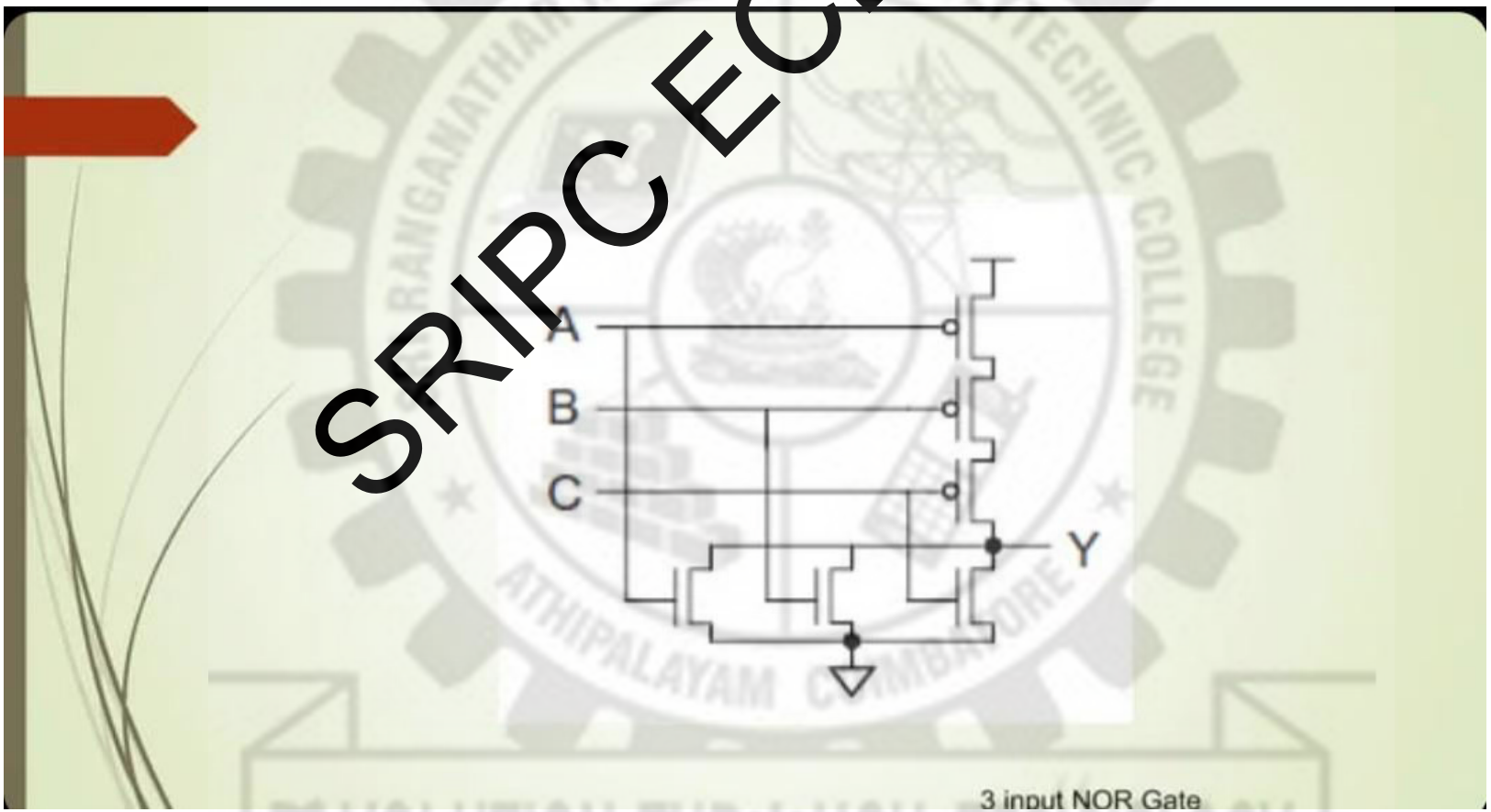
CMOS NOR Gate

- The nMOS transistors are in parallel to pull the output low when either input is high.
- The pMOS transistors are in series to pull the output high when both inputs are low

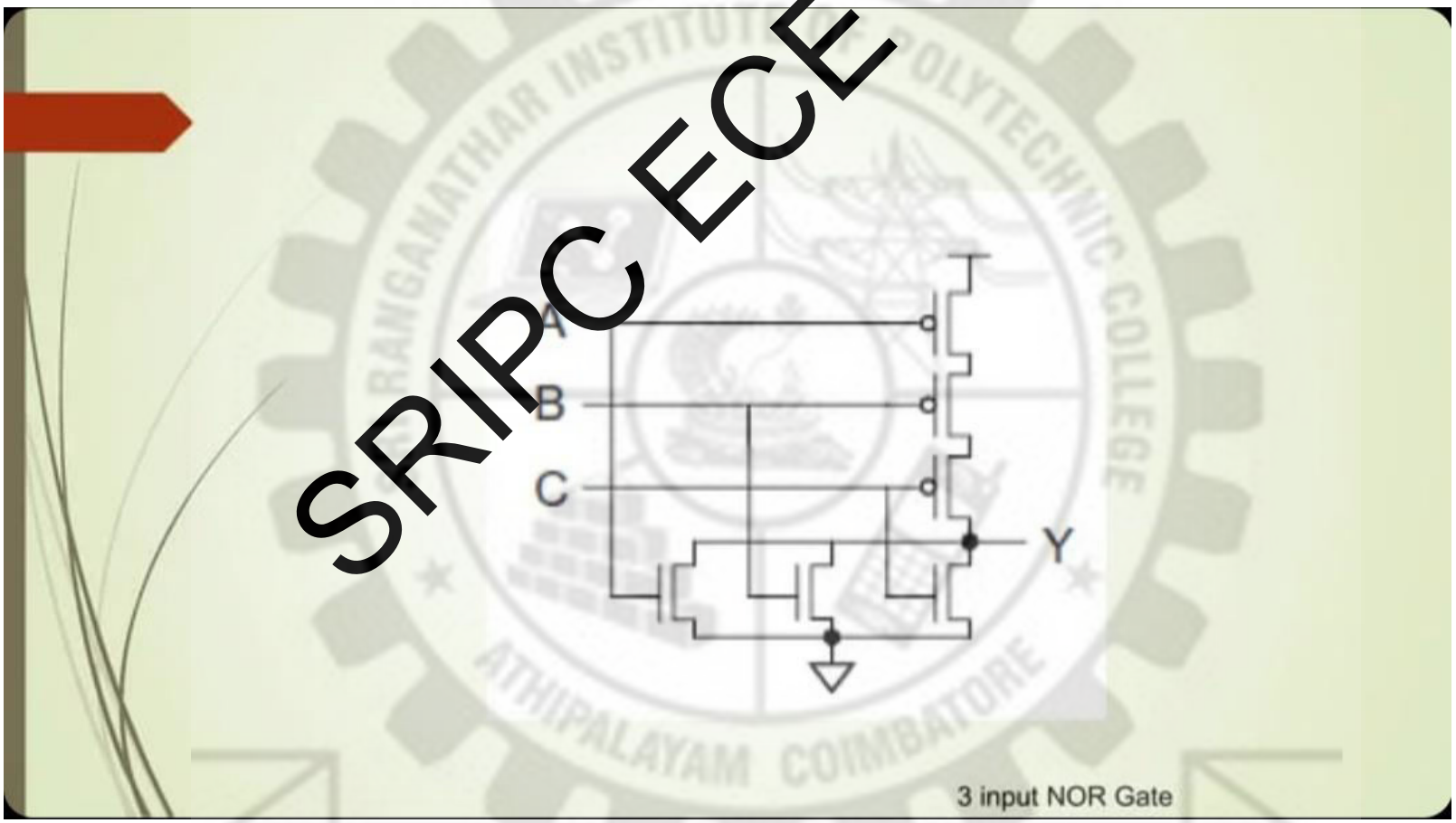


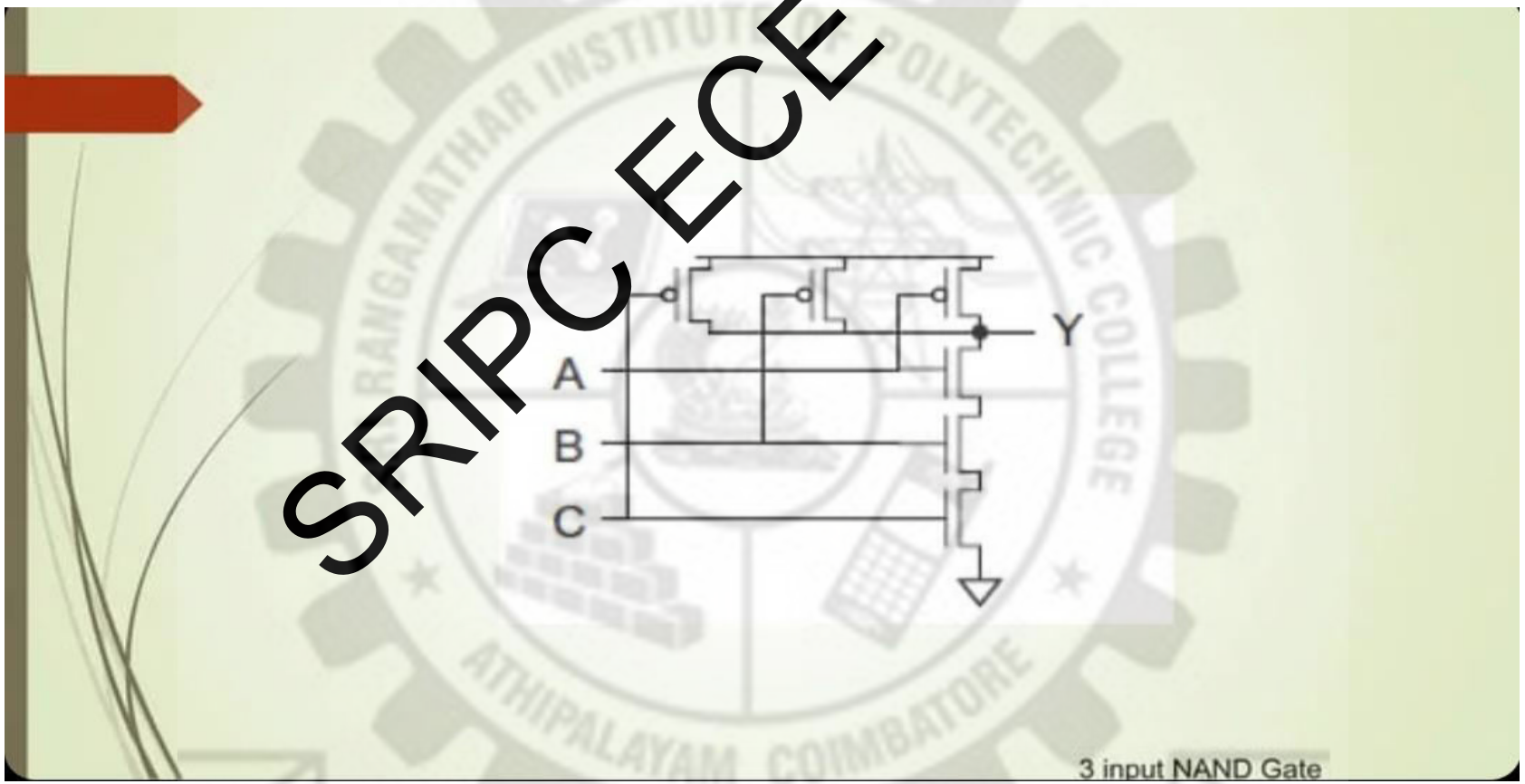
NOR gate truth table

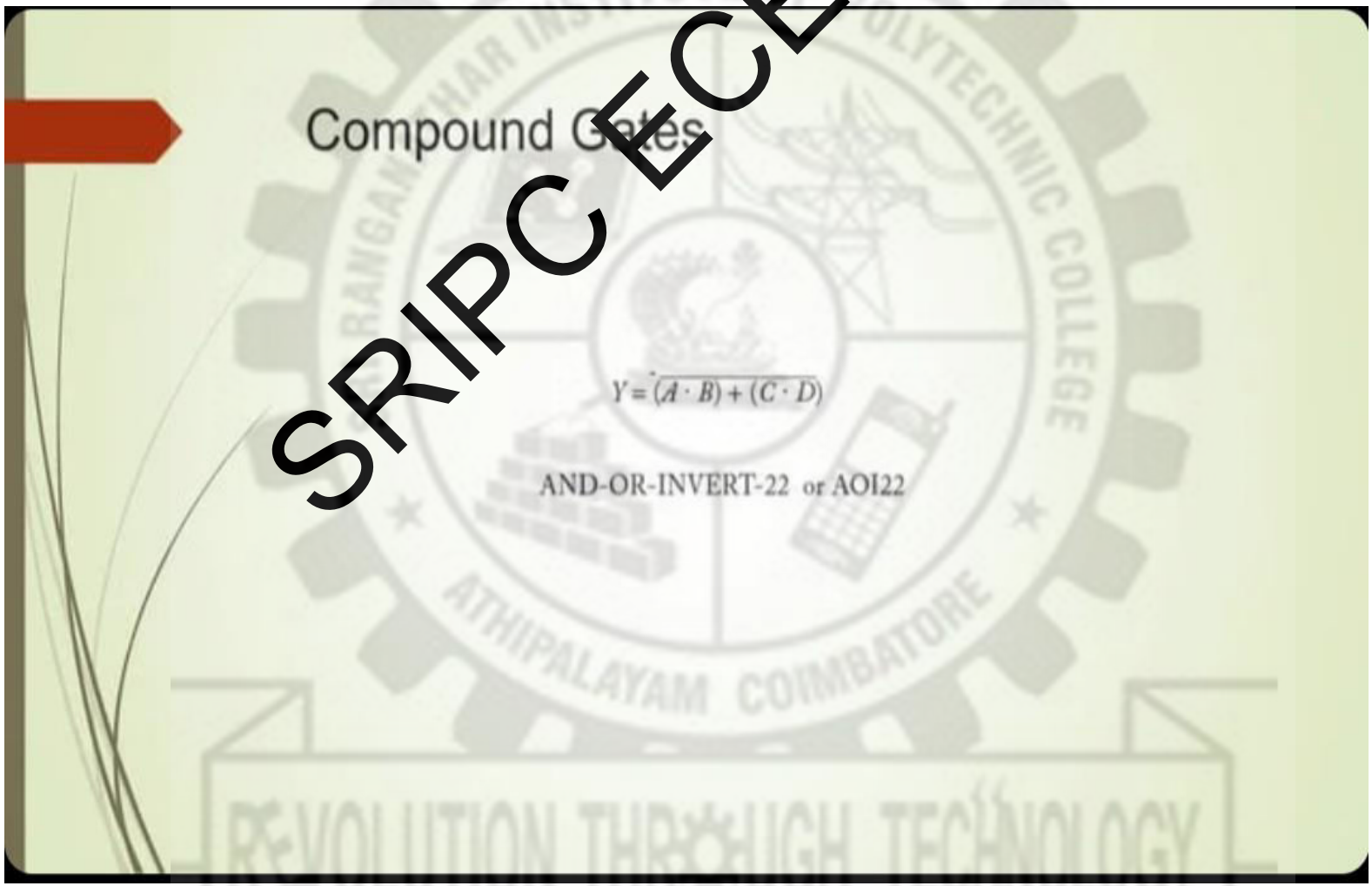
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

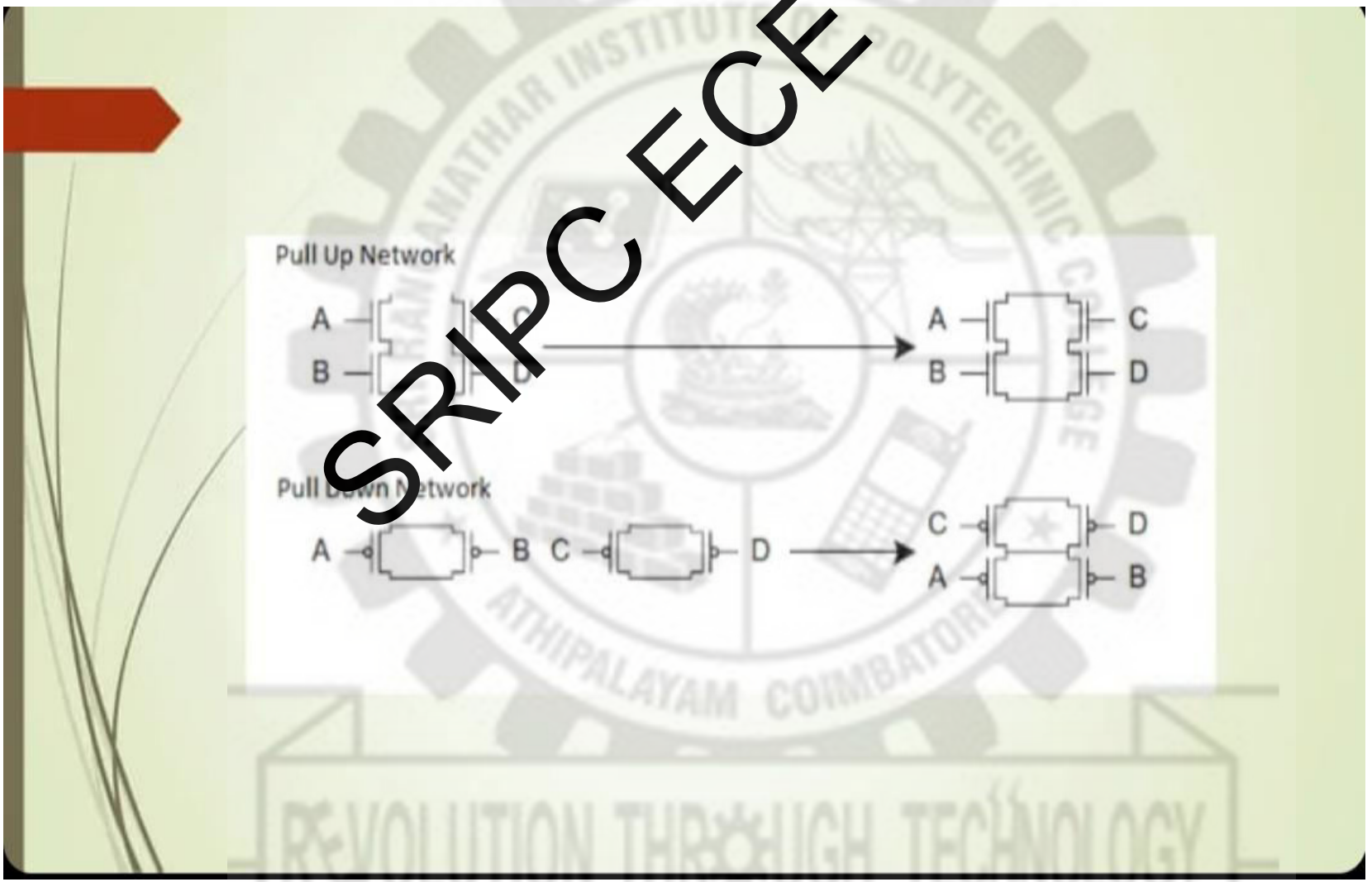


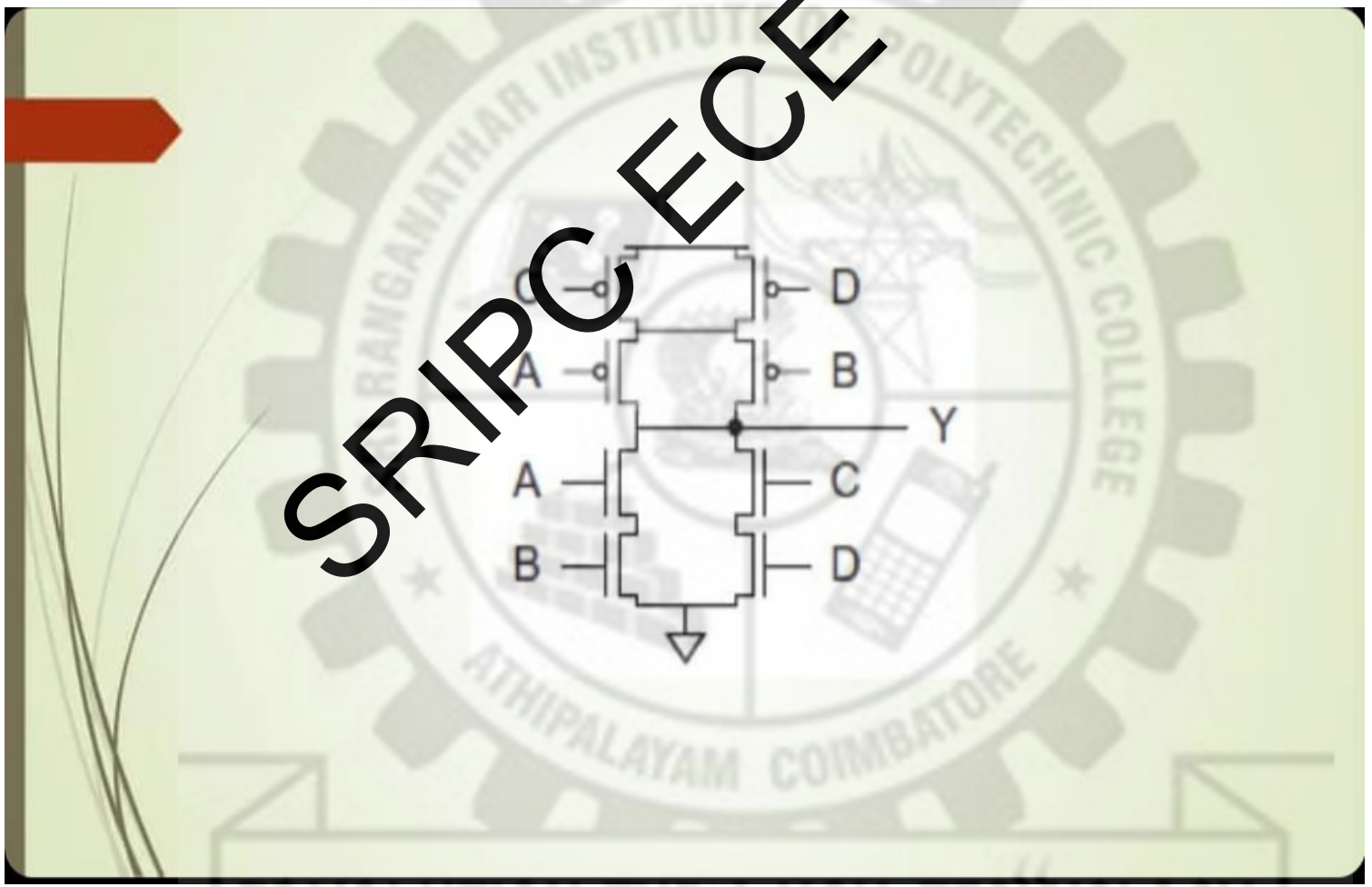
3 input NOR Gate







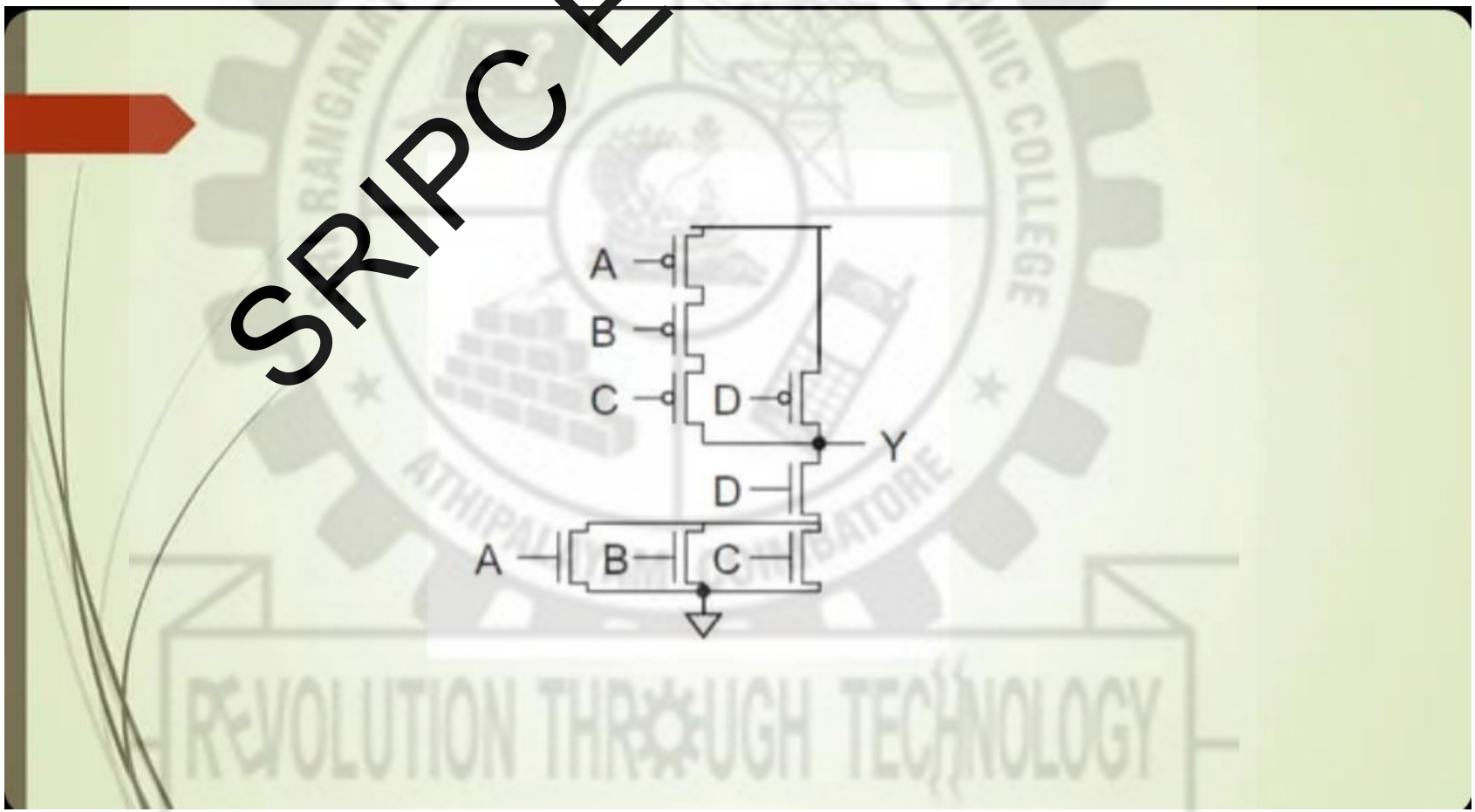




Compound Gates

$$Y = \overline{(A + B + C) \cdot D}$$

OR-AND-INVERT-3-1 (OAI31) gate.



Combinational Circuits

Sketch transistor-level schematics for the following logic functions. You may assume you have both true and complementary versions of the inputs available.

a) A 2:4 decoder defined by

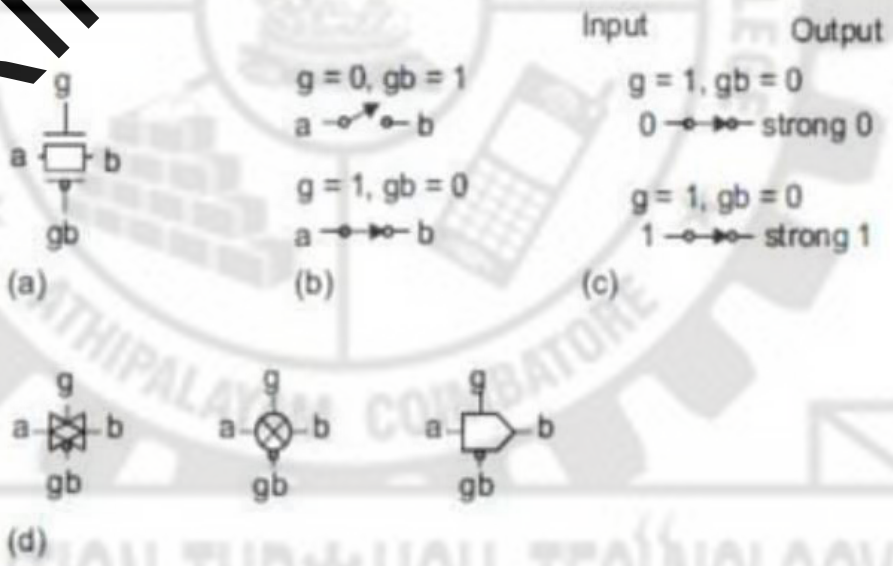
$Y0 = \overline{A0} \cdot \overline{A1}$
$Y1 = A0 \cdot \overline{A1}$
$Y2 = \overline{A0} \cdot A1$
$Y3 = A0 \cdot A1$

b) A 3:2 priority encoder defined by

$Y0 = \overline{A0} \cdot (A1 + \overline{A2})$
$Y1 = \overline{A0} \cdot \overline{A1}$

Pass Transistors and Transmission Gates

- *By combining a nMOS and a pMOS transistor in parallel,* we obtain a switch, which 0s and 1s are both passed in an acceptable fashion.
- We term this a ***transmission gate or pass gate.***



Pass Transistors and Transmission Gates

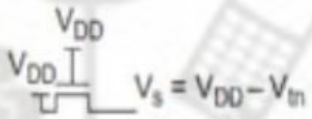
- The *strength of a signal is measured by how closely it approximates an ideal voltage source.*
- The power supplies, or *rails, (VDD and GND) are the source of the strongest 1s and 0s.*
- An nMOS transistor is an almost perfect switch when passing a 0 and thus we say it passes a ***strong 0***. *However, the nMOS transistor is imperfect at passing a 1. We say it* passes a *degraded or* ***weak 1***.
- An pMOS transistor is an almost perfect switch when passing a 1 and thus we say it passes a ***strong 1***. *However, the nMOS transistor is imperfect at passing a 1. We say it* passes a *degraded or* ***weak 0***

Pass Transistors and Transmission Gates

Concept of Weak 1:

Consider an nMOS transistor with the Gate and Drain tied to V_{DD}.

Imagine that the source is initially at $V_s = 0$. $V_{gs} > V_{tn}$, *so the transistor* is ON and current flows.



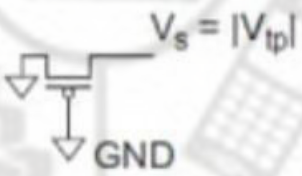
If the voltage on the source rises to $Vs = V_{DD} - V_{tn}$, V_{gs} *falls to* V_{tn} *and the transistor cuts itself OFF.*

Therefore, nMOS transistors attempting to pass a 1 never pull the source above $V_{DD} - V_{tn}$. *This loss is called threshold drop.*

Pass Transistors and Transmission Gates

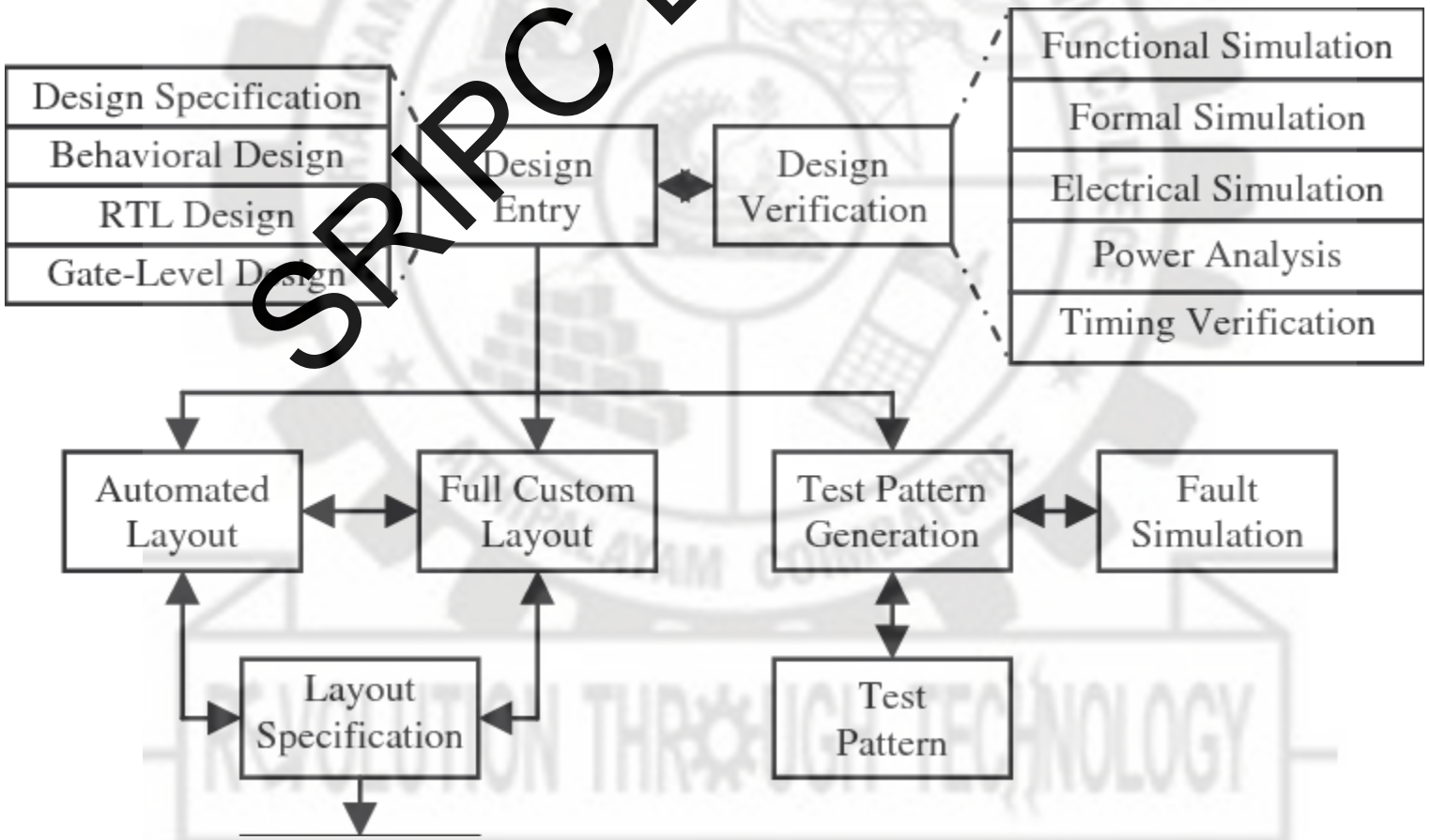
Concept of Weak 0:

Similarly, pMOS transistors pass 1s well but 0s poorly. If the pMOS source drops below |*Vtp*|, *the transistor cuts off*

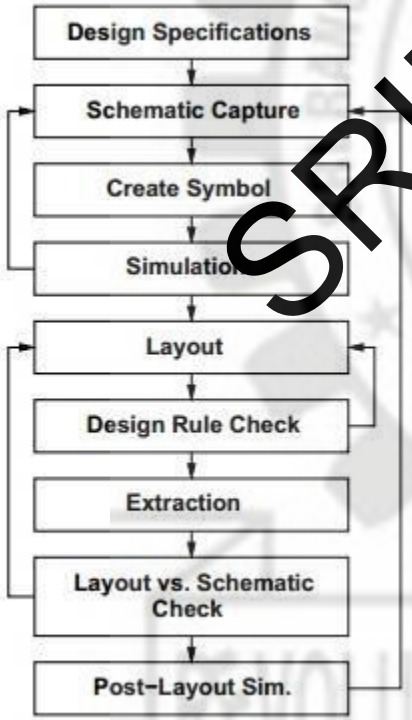


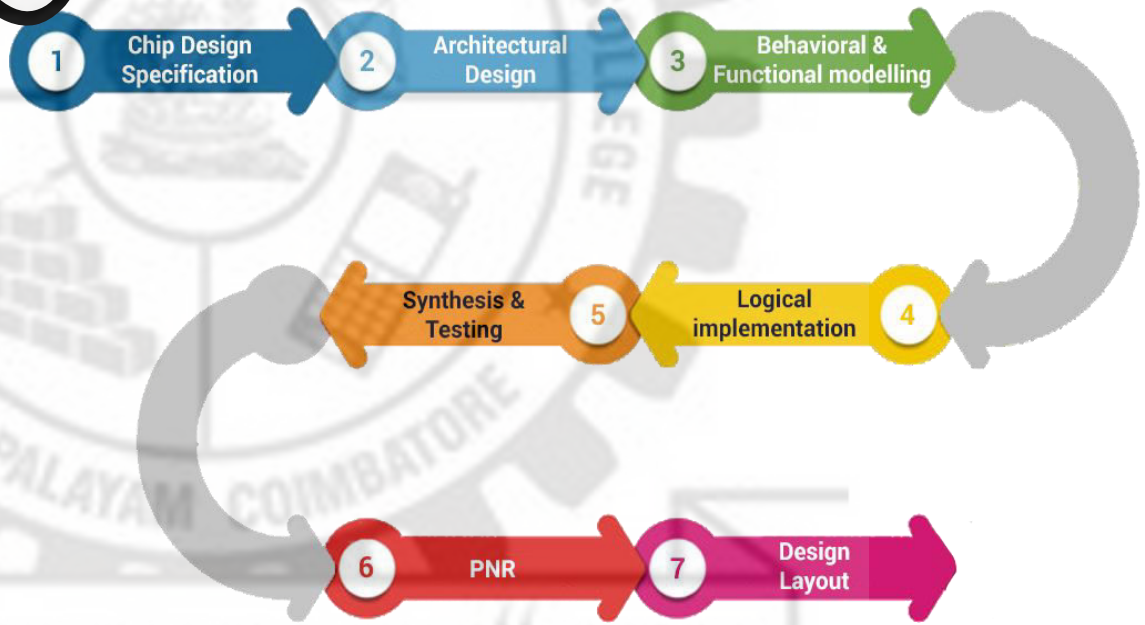
Hence, pMOS transistors only pull down to within a threshold above GND,

DIFFERENT LEVEL OF ABSTRACTIONS IN VLSI DESIGN



STEPS INVOLVED IN VLSI DESIGN PROCESS

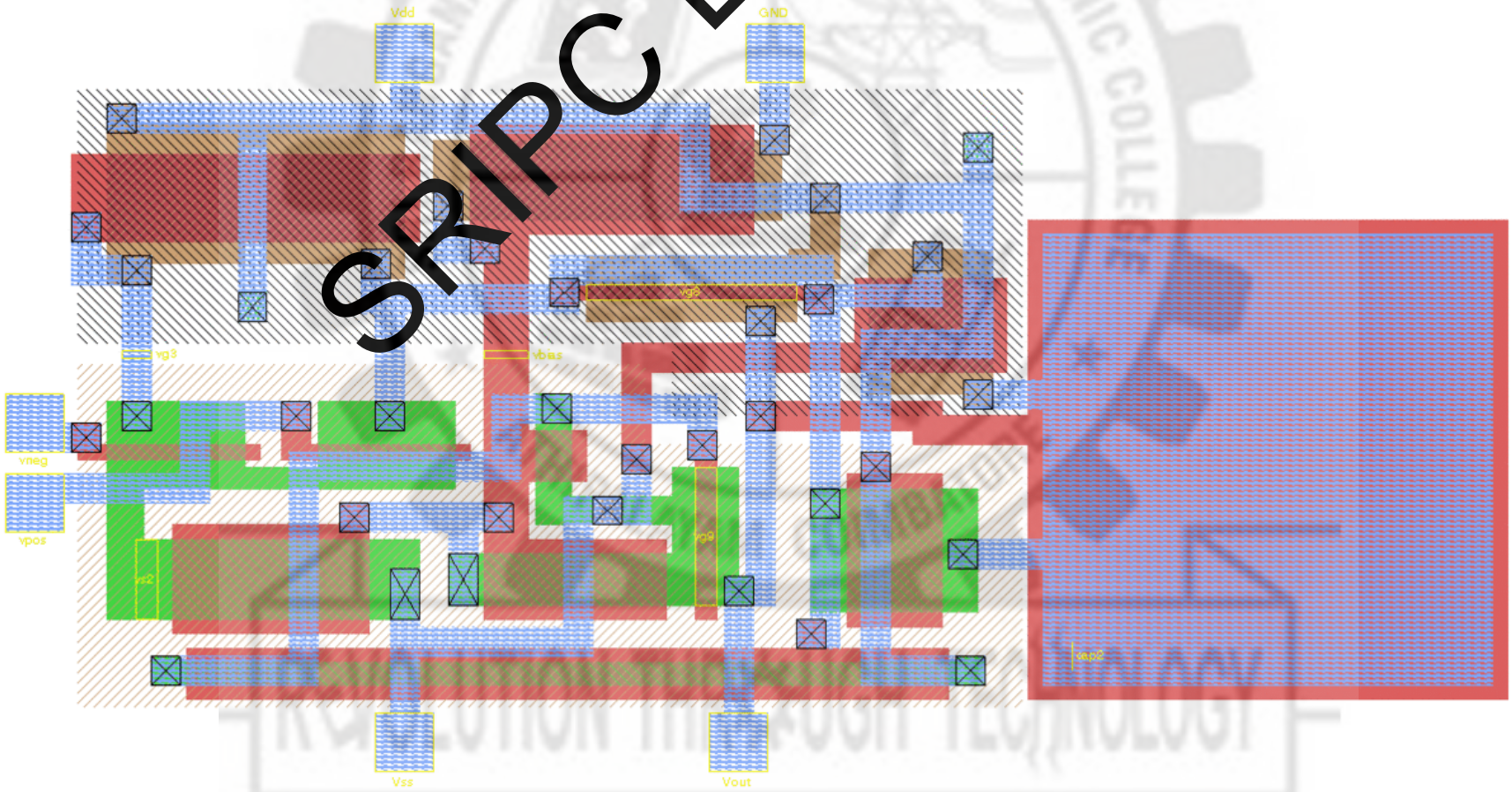


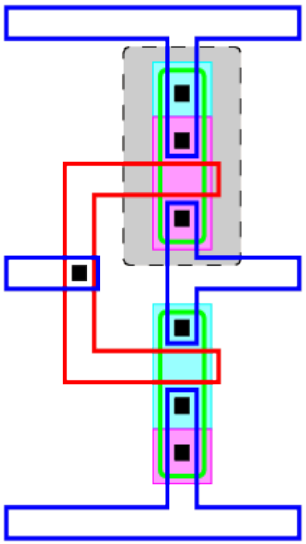


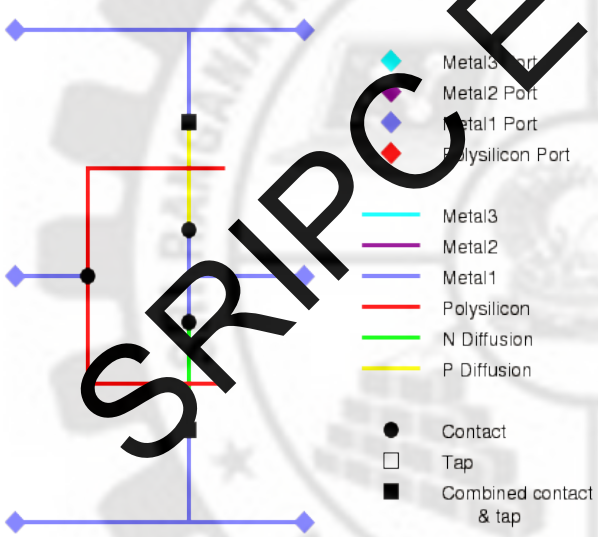
ECE

LAYOUT

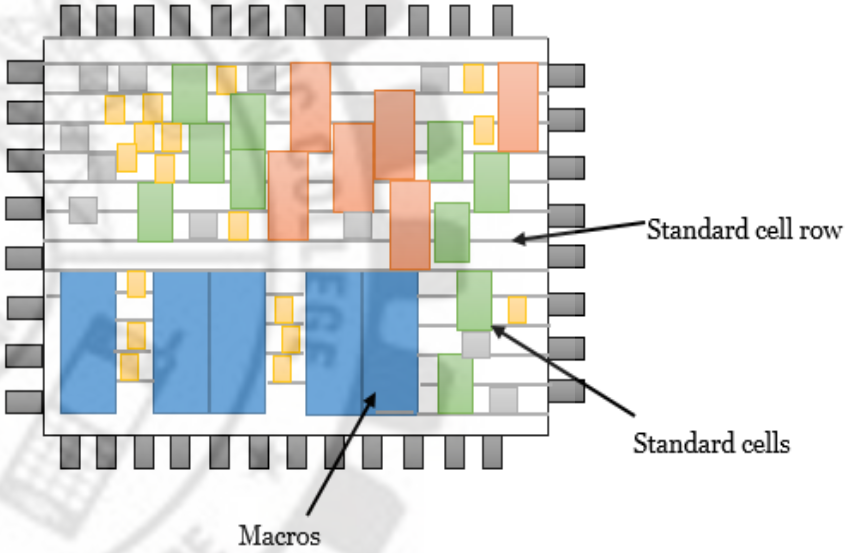
SRIPC ECE







STICK DIAGRAM



ROUTING

UNIT-II
INTRODUCTION TO VHDL

SWATHI E R
LECTURER/ECE

History of VHDL

- Designed by IBM, Texas Instruments, and Intermetrics as part of the DoD funded VHSIC program
- Standardized by the IEEE in 1987: IEEE 1076-1987
- Enhanced version of the language defined in 1993: IEEE 1076-1993
- Additional standardized packages provide definitions of data types and expressions of timing data
  - IEEE 1164 (data types)
  - IEEE 1076.3 (numeric)
  - IEEE 1076.4 (timing)

Traditional vs. Hardware Description Languages

- Procedural programming languages provide the *how* or recipes
  - for computation
  - for data manipulation
  - for execution on a specific hardware model
- Hardware description languages *describe* a system
  - Systems can be described from many different points of view
    - Behavior: what does it do?
    - Structure: what is it composed of?
    - Functional properties: how do I interface to it?
    - Physical properties: how fast is it?

Usage

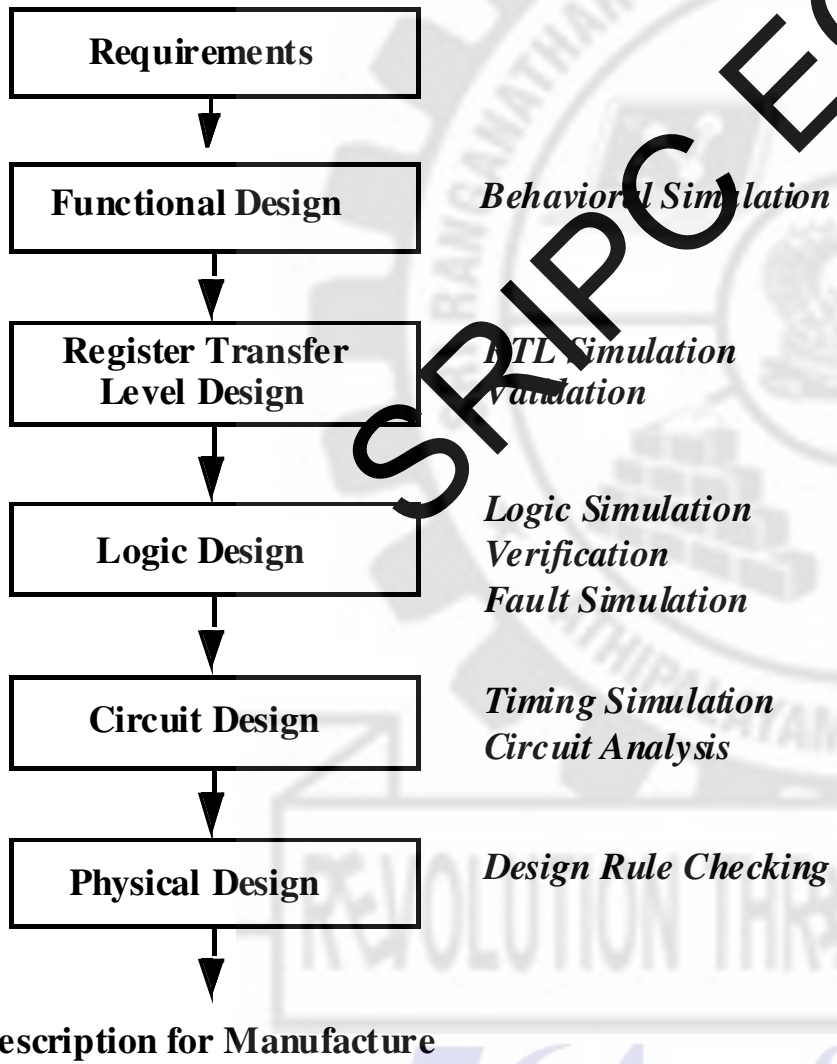
- Descriptions can be at different levels of abstraction
  - Switch level: model switching behavior of transistors
  - Register transfer level: model combinational and sequential logic components
  - Instruction set architecture level: functional behavior of a microprocessor

- Descriptions can used for
  - Simulation
    - Verification, performance evaluation
  - Synthesis
    - First step in hardware design

Why do we Describe Systems?

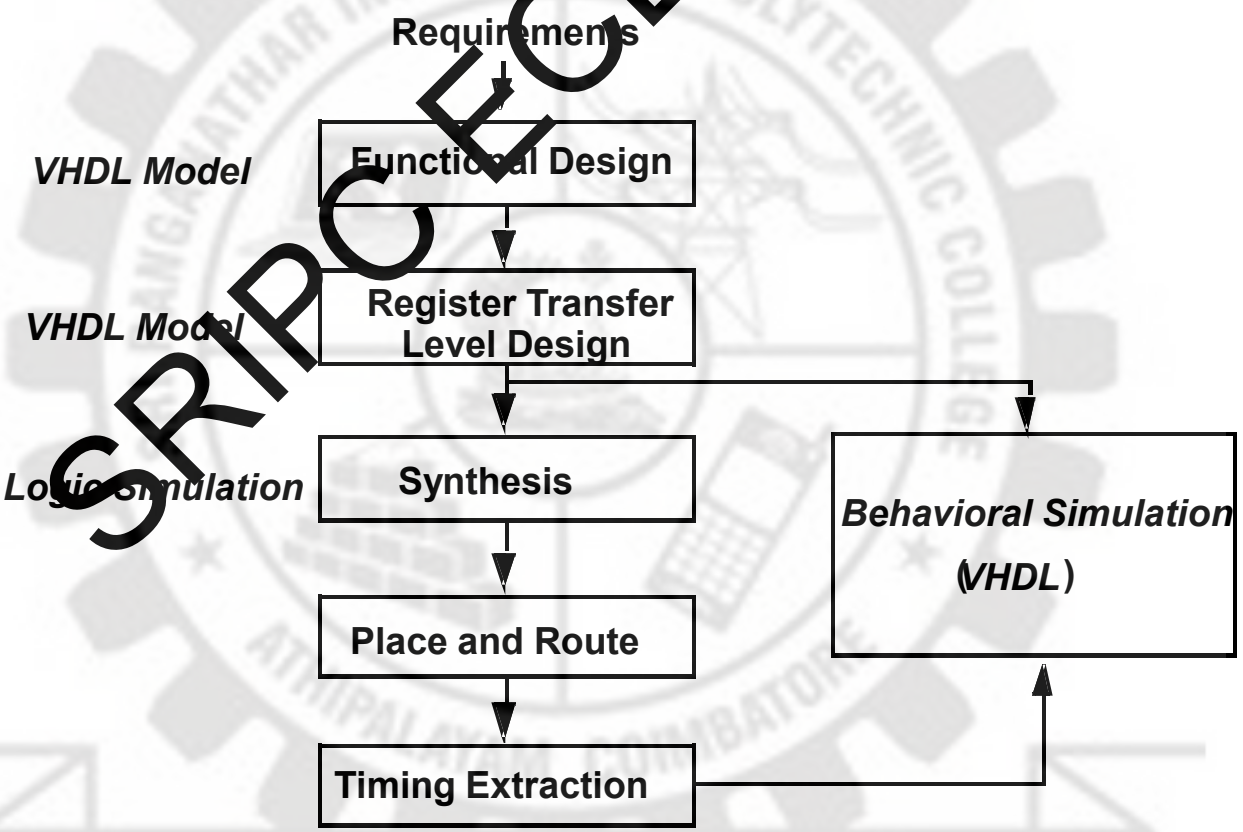
- Design Specification
  - unambiguous definition of components and interfaces in a large design
- Design Simulation
  - verify system/subsystem/chip performance prior to design implementation
- Design Synthesis
  - automated generation of a hardware design

Digital System Design Flow



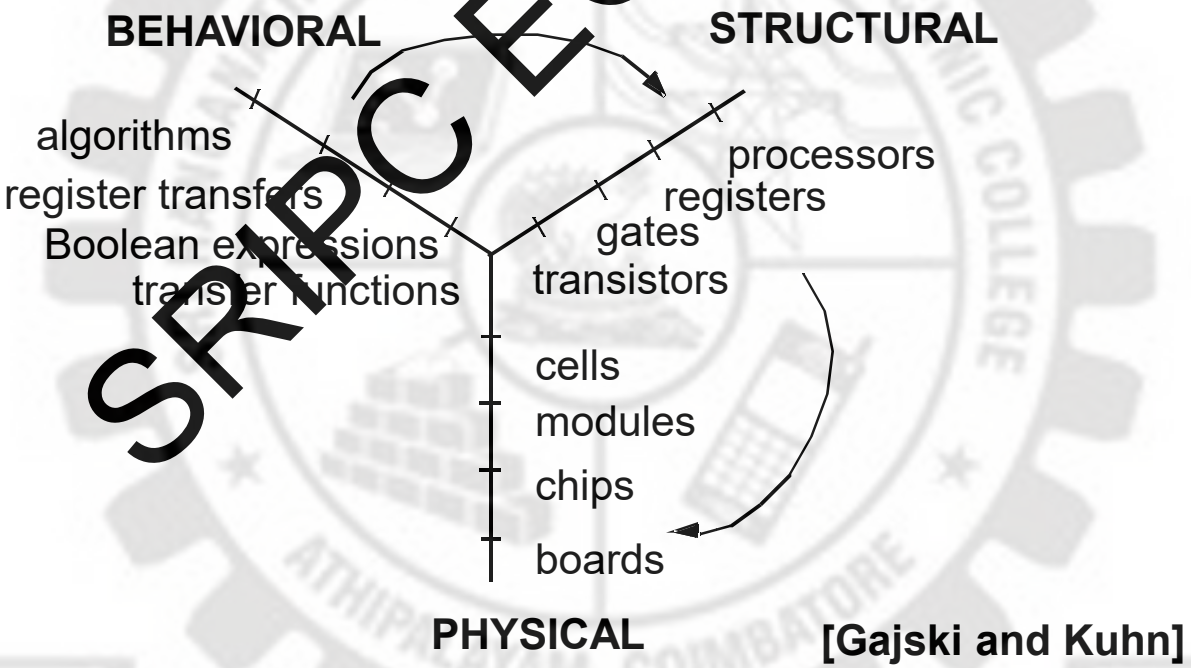
- Design flows operate at multiple levels of abstraction
- Need a uniform description to translate between levels
- Increasing costs of design and fabrication necessitate greater reliance on automation via CAD tools
  - $5M - $100M to design new chips
  - Increasing time to market pressures

A Synthesis Design Flow



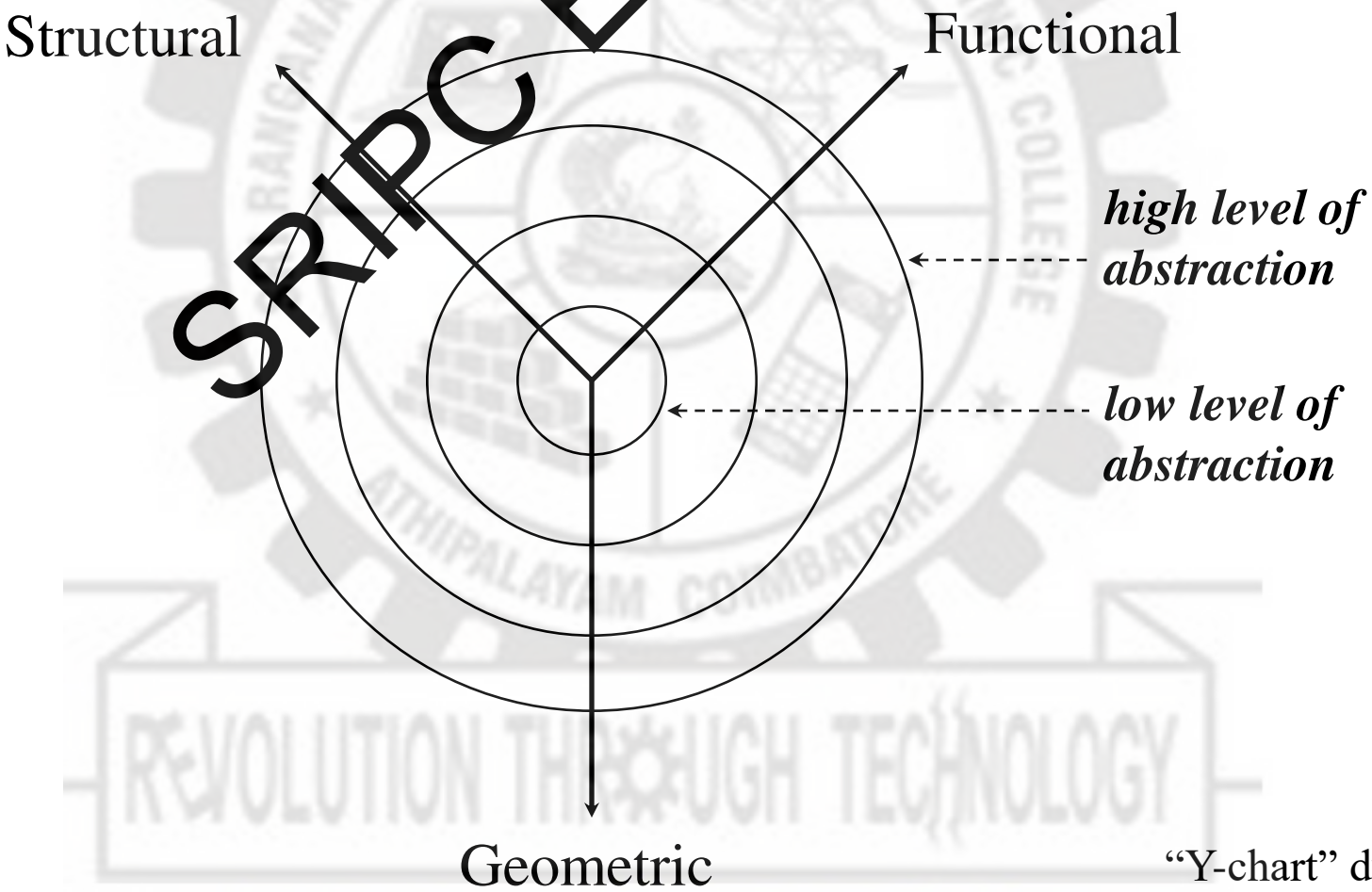
- Automation of design refinement steps
- Feedback for accurate simulation
- Example targets: ASICs, FPGAs

The Role of Hardware Description Languages



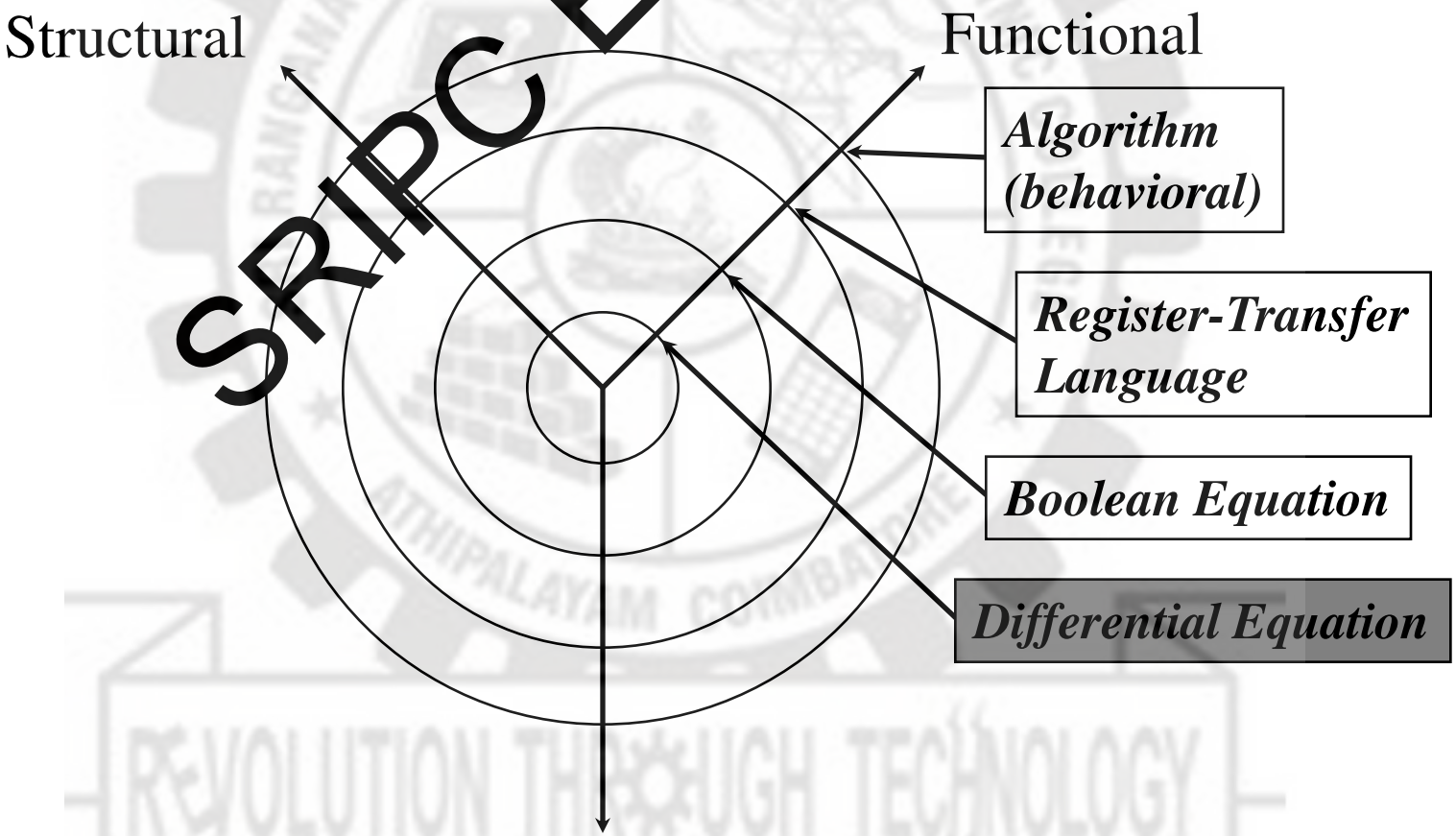
- Design is structured around a hierarchy of representations
- HDLs can describe distinct aspects of a design at multiple levels of abstraction

Domains and Levels of Modeling



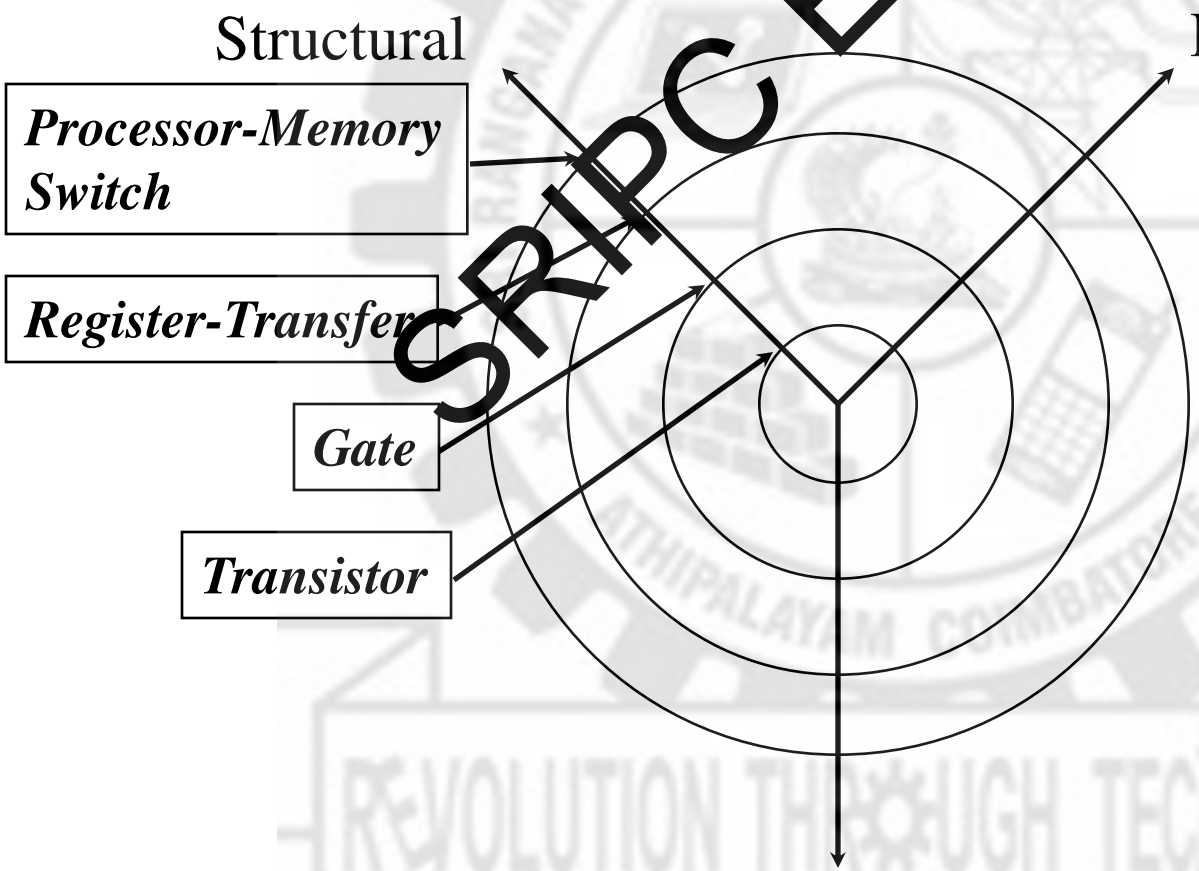
"Y-chart" due to Gajski & Kahn

Domains and Levels of Modeling



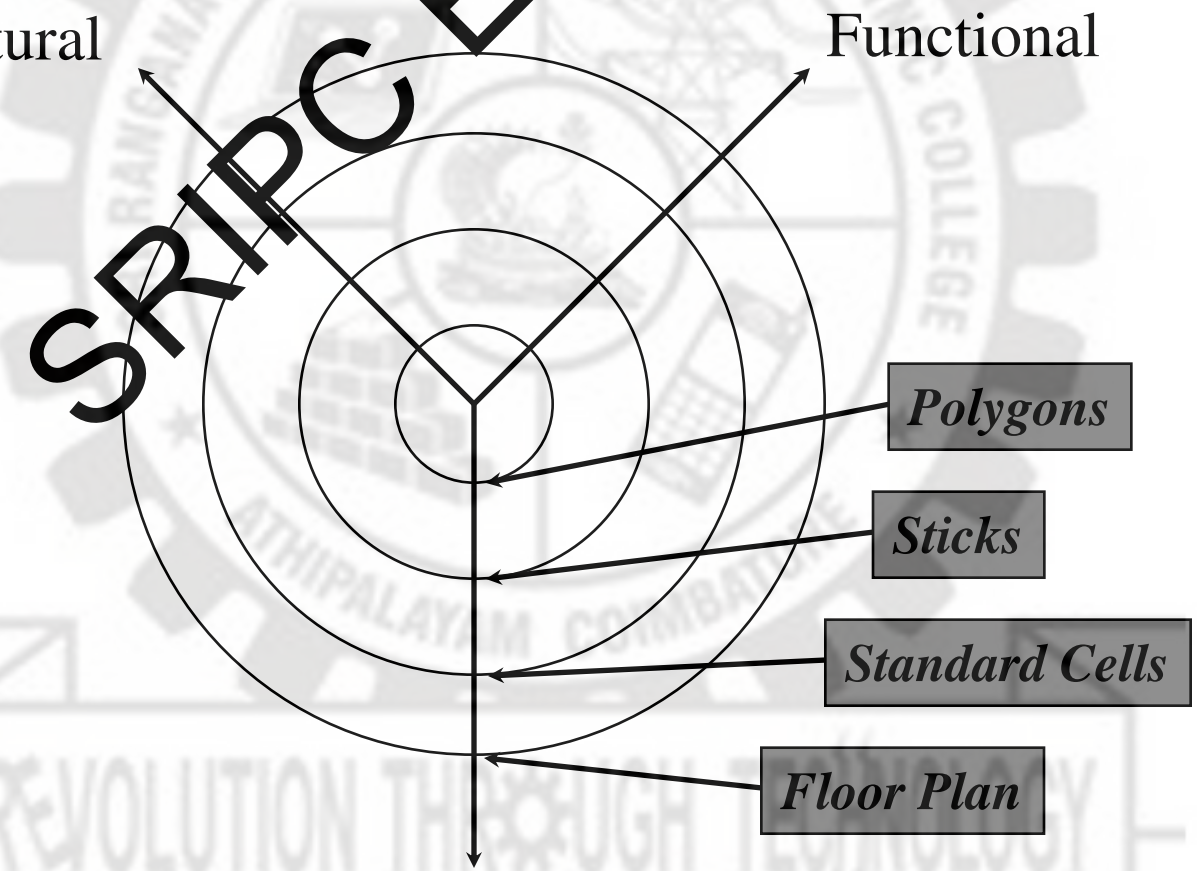
"Y-chart" due to Gajski & Kahn

Domains and Levels of Modeling



"Y-chart" due to Gajski & Kahn

Domains and Levels of Modeling



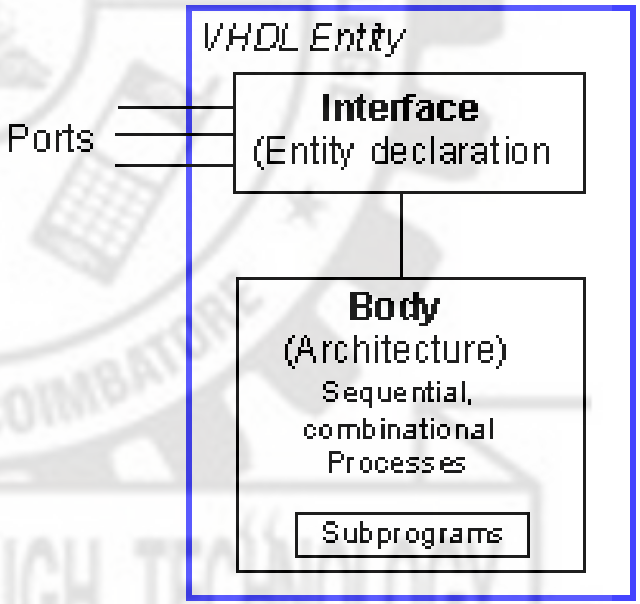
"Y-chart" due to Gajski & Kahn

Basic VHDL Concepts

- Interfaces
- Modeling (Behavior, Dataflow, Structure)
- Test Benches
- Analysis, elaboration, simulation
- Synthesis

Basic Structure of a VHDL File

- Entity
  - Entity declaration: interface to outside world; defines input and output signals
  - Architecture: describes the entity, contains processes, components operating concurrently



Entity Declaration

entity NAME_OF_ENTITY **is**
    port (*signal_names*: **mode** *type*;
        signal_names: **mode** *type*;
        .
        .
        signal_names: **mode** *type*);
end [NAME_OF_ENTITY] ;

MVL - 9			
Uninitialized	'U'	Weak 1	'H'
Don't Care	'-'	Weak 0	'L'
Forcing 1	'1'	Weak Unknown	'W'
Forcing 0	'0'	High Impedance	'Z'
Forcing Unknown	'X'		

- NAME_OF_ENTITY: user defined
- signal_names: list of signals (both input and output)
- mode: in, out, buffer, inout
- type: boolean, integer, character, std_logic

Architecture

- **Behavioral Model:**

```
architecture architecture_name of NAME_OF_ENTITY
    is
        -- Declarations
            .....
            .....

begin
        -- Statements
end architecture_name;
```

VHDL Process

- Group of Instructions that are executed sequentially
- Syntax

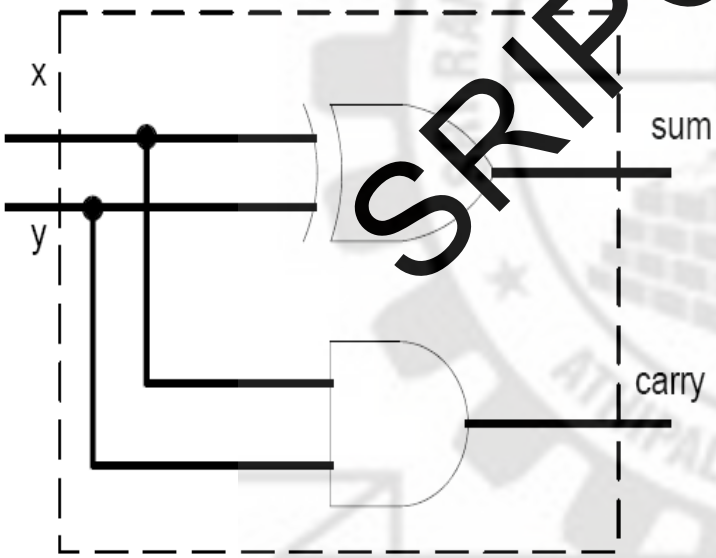
```
process
  declarations;
begin
  sequential statement;
  sequential statement;
 . . .
end process;
```

- The whole process is a concurrent statement

STATEMENTS

- An if…else statement is **a sequential statement** in VHDL which got executed depending on the value of the condition. The if condition tests each condition sequentially until the true condition is found.
- VHDL is a Hardware Description Language that is used to describe at a high level of abstraction a digital circuit in an FPGA or ASIC. **When we need to perform a choice or selection between two or more choices**, we can use the VHDL conditional statement.
- VHDL entity example. The entity syntax is **keyword "entity", followed by entity name and the keyword "is" and "port"**. Then inside parenthesis there is the ports declaration. In the port declaration there are port name followed by colon, then port direction (in/out in this example) followed by port type.

Half Adder




```
library ieee;
use  ieee.std_logic_1164.all;
entity half_adder is
port(
        x,y: in std_logic;
        sum, carry: out std_logic);
end half_adder;

architecture myadd of half_adder is
        begin
                sum <= x xor y;
                carry <= x and y;
end myadd;
```

Entity Examples …


```
entity half_adder is
port(
        x,y: in std_logic;
        sum, carry: out std_logic);
end half_adder;
```

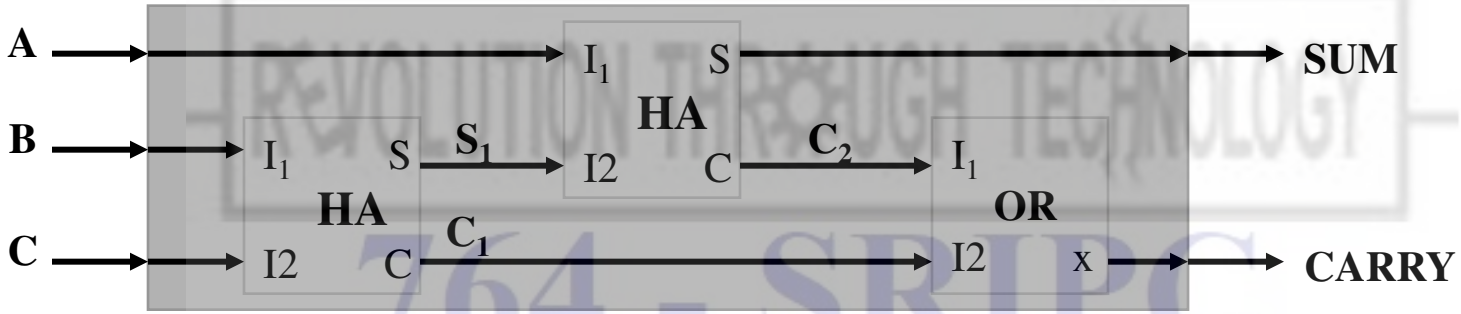
Architecture Examples: Behavioral Description

- Entity FULLADDER is
  port (A, B, C: in std_logic;
  SUM, CARRY: in std_logic);
  end FULLADDER;

- Architecture CONCURRENT of FULLADDER is
  begin
  SUM <= A xor B xor C after 5 ns;
  CARRY <= (A and B) or (B and C) or (A and C) after 3 ns;
  end CONCURRENT;

Architecture Examples: Structural Description …

- architecture STRUCTURAL of FULLADDER is
  signal S1, C1, C2 : bit;
  component HA
    port (I1, I2 : in bit; S, C : out bit);
  end component;
  component OR
    port (I1, I2 : in bit; X : out bit);
  end component;
  begin
  INST_HA1 : HA port map (I1 => B, I2 => C, S => S1, C => C1);
  INST_HA2 : HA port map (I1 => A, I2 => S1, S => SUM, C => C2);
  INST_OR : OR port map (I1 => C2, I2 => C1, X => CARRY);
  end STRUCTURAL;



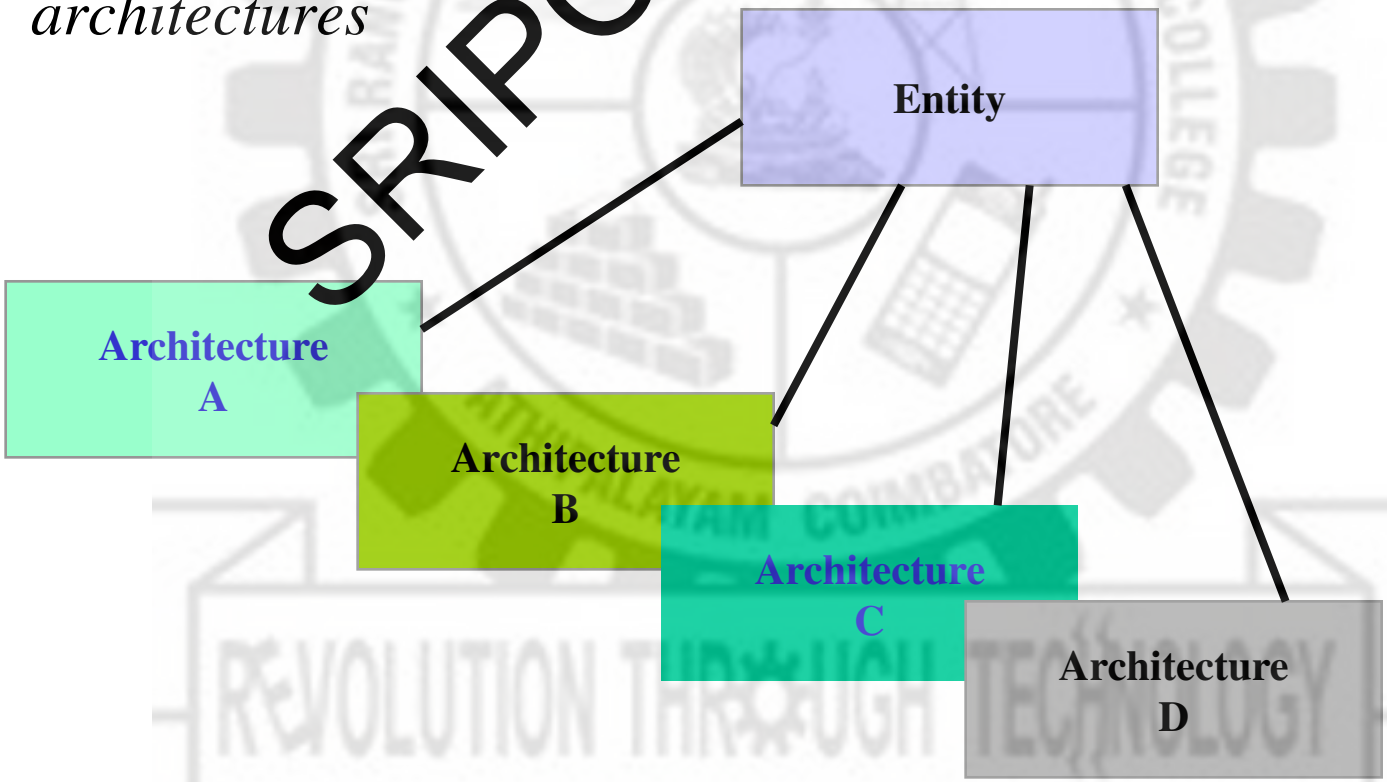
… Architecture Examples: Structural Description

```
Entity HA is
PORT (I1, I2 : in bit; S, C : out bit);
end HA ;
Architecture behavior of HA is
begin
   S <= I1 xor I2;
   C <= I1 and I2;
end behavior;
```

```
Entity OR is
PORT (I1, I2 : in bit; X : out bit);
end OR ;
Architecture behavior of OR is
begin
   X <= I1 or I2;
end behavior;
```

One Entity Many Descriptions

- A system (an *entity*) can be specified with different *architectures*



Test Benches

- Testing a design by simulation
- Use a *test bench* model
  - an architecture body that includes an instance of the design under test
  - applies sequences of test values to inputs
  - monitors values on output signals
    - either using simulator
    - or with a process that verifies correct operation

PROGRAM

- **VHDL CODES:**
- **OR gate program**
- library IEEE;
- use IEEE.STD_LOGIC_1164.ALL; use IEEE.STD_LOGIC_ARITH.ALL;
- use IEEE.STD_LOGIC_UNSIGNED.ALL;
- entity gate is
- Port (a : in STD_LOGIC; b : in STD_LOGIC;
- c : out STD_LOGIC); end gate;
- architecture Behavioral of gate is begin
- c <= a or b;
- end Behavioral;
- **ANDgate Program**
- library IEEE;
- use IEEE.STD_LOGIC_1164.ALL; use IEEE.STD_LOGIC_ARITH.ALL;
- use IEEE.STD_LOGIC_UNSIGNED.ALL;
- entity gate is
- Port (a : in STD_LOGIC; b : in STD_LOGIC;
- c : out STD_LOGIC); end gate;
- architecture Behavioral of gate is begin
- c <= a and b; end Behavioral;

PROGRAM

- **VHDL CODES:**
- **NOT gate program**
- library IEEE;
- use IEEE.STD_LOGIC_1164.ALL; use IEEE.STD_LOGIC_ARITH.ALL;
- use IEEE.STD_LOGIC_UNSIGNED.ALL;
- entity gate is
- Port (a : in STD_LOGIC); end gate;
- architecture Behavioral of gate is begin
- c <= a not b;
- end Behavioral;
- **NANDgate Program**
- library IEEE;
- use IEEE.STD_LOGIC_1164.ALL; use IEEE.STD_LOGIC_ARITH.ALL;
- use IEEE.STD_LOGIC_UNSIGNED.ALL;
- entity gate is
- Port (a : in STD_LOGIC; b : in STD_LOGIC;
- c : out STD_LOGIC); end gate;
- architecture Behavioral of gate is begin
- c <= a nand b; end Behavioral;

PROGRAM

- **VHDL CODES:**
- **Nor gate Program**
- library IEEE;
- use IEEE.STD_LOGIC_1164.ALL; use IEEE.STD_LOGIC_ARITH.ALL;
- use IEEE.STD_LOGIC_UNSIGNED.ALL;
- entity gate is
- Port (a : in STD_LOGIC; b : in STD_LOGIC;
- c : out STD_LOGIC); end gate;
- architecture Behavioral of gate is begin
- c <= a nand b; end Behavioral;
- **Exor gate Program**
- library IEEE;
- use IEEE.STD_LOGIC_1164.ALL; use IEEE.STD_LOGIC_ARITH.ALL;
- use IEEE.STD_LOGIC_UNSIGNED.ALL;
- entity gate is
- Port (a : in STD_LOGIC; b : in STD_LOGIC;
- c : out STD_LOGIC); end gate;
- architecture Behavioral of gate is begin
- c <= a exor b; end Behavioral;

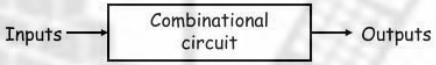
UNIT-III

COMBINATIONAL CIRCUIT DESIGN

SWATHI E R
LECTURER/ECE

Combinational Circuits

Combinational circuits

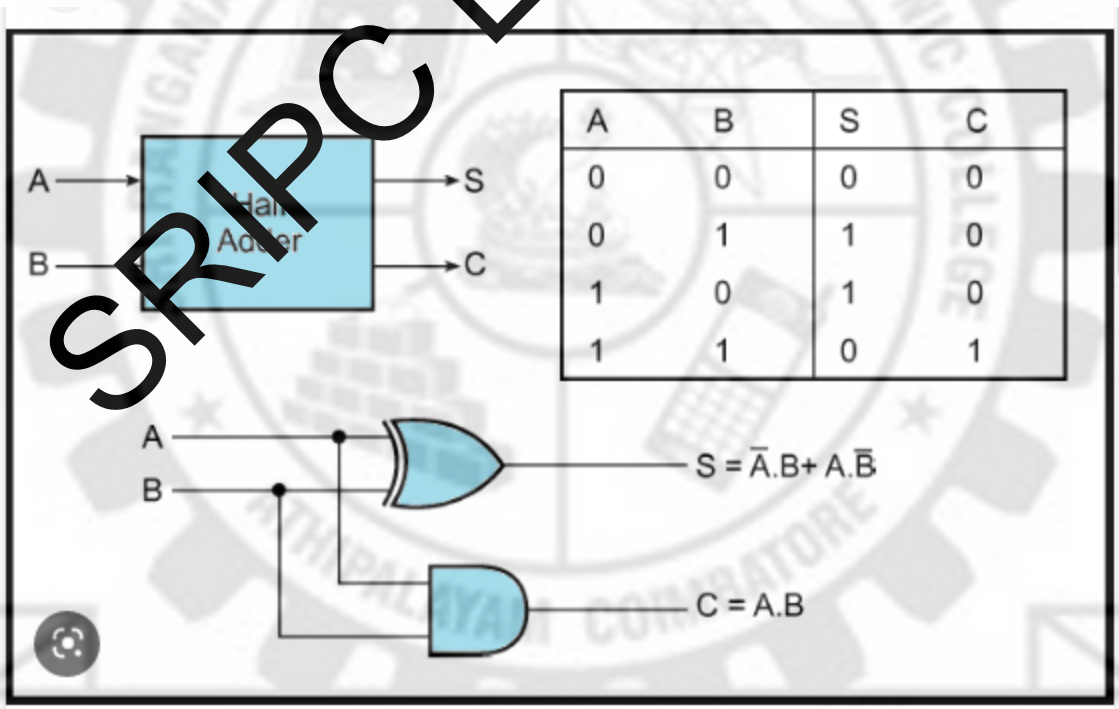


- So far we've just worked with combinational circuits, where applying the same inputs always produces the same outputs.
- This corresponds to a mathematical function, where every input has a single, unique output.
- In programming terminology, combinational circuits are similar to "functional programs" that do not contain variables and assignments.

HALF ADDER

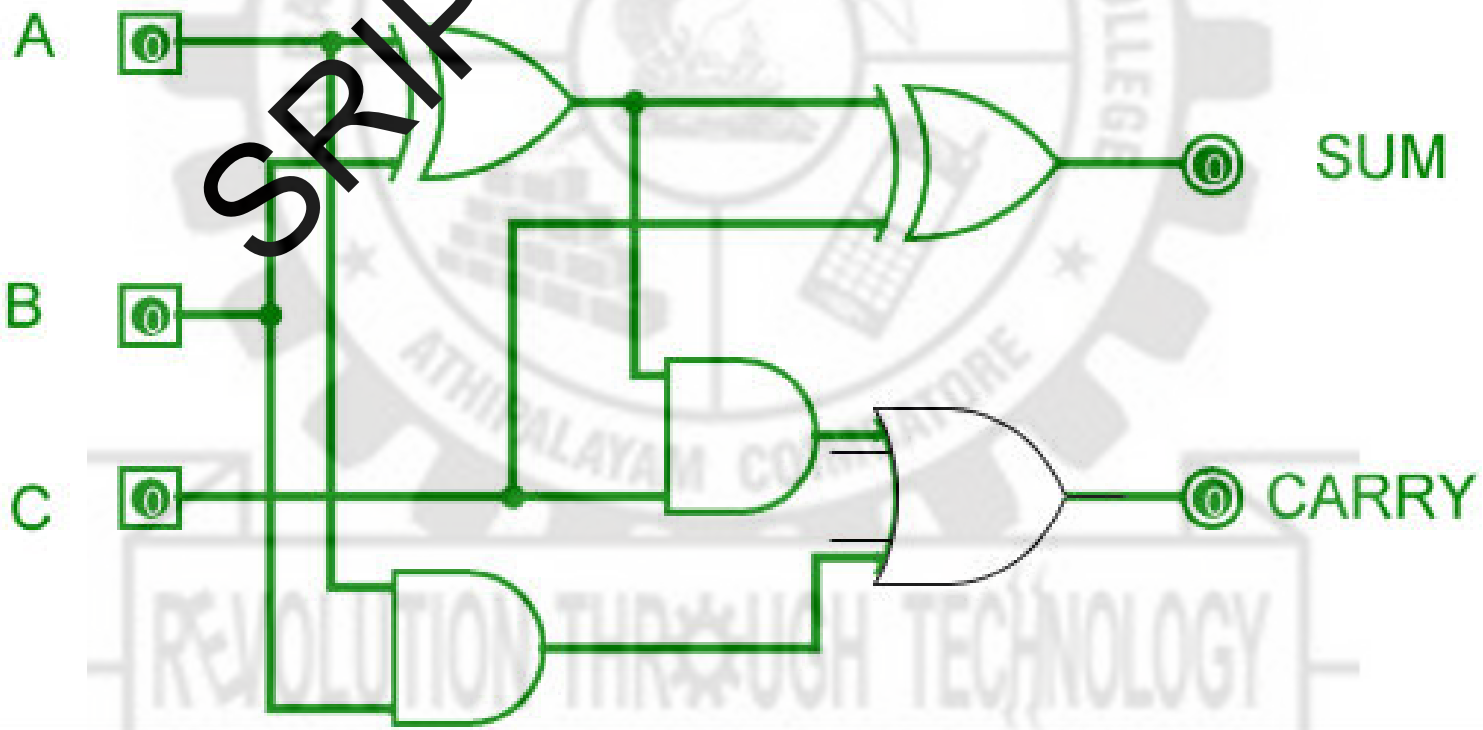
Half Adder

TRUTH TABLE

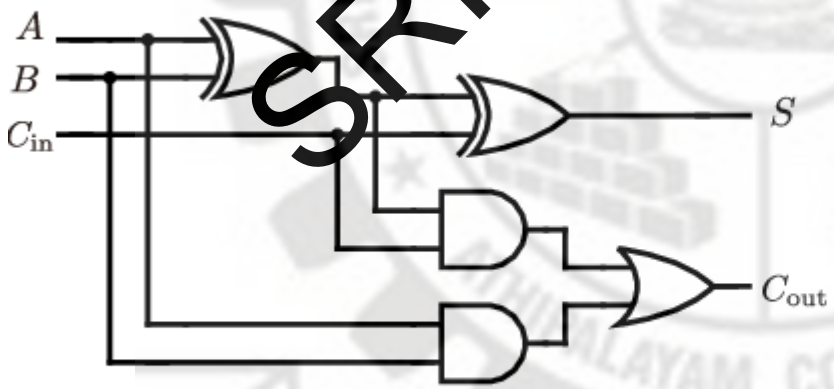


A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

FULL ADDER



TRUTH TABLE



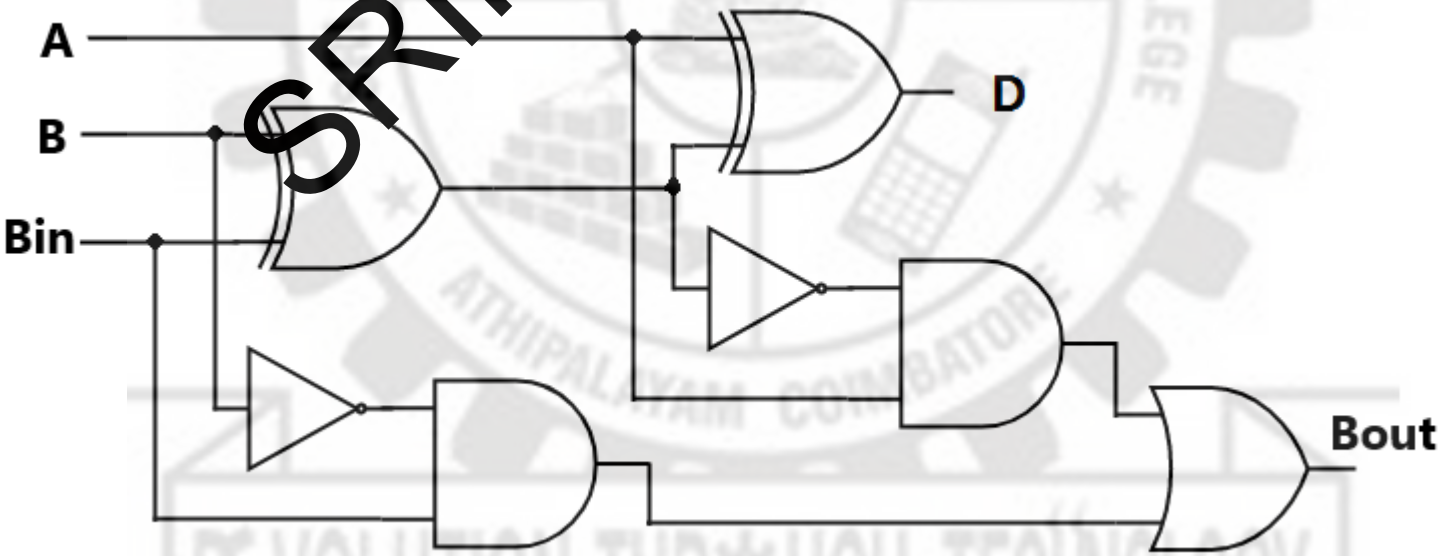
Inputs			Outputs	
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

HALF SUBTRACTOR

TRUTH TABLE

Inputs		Outputs	
A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

FULL SUBTRACTOR



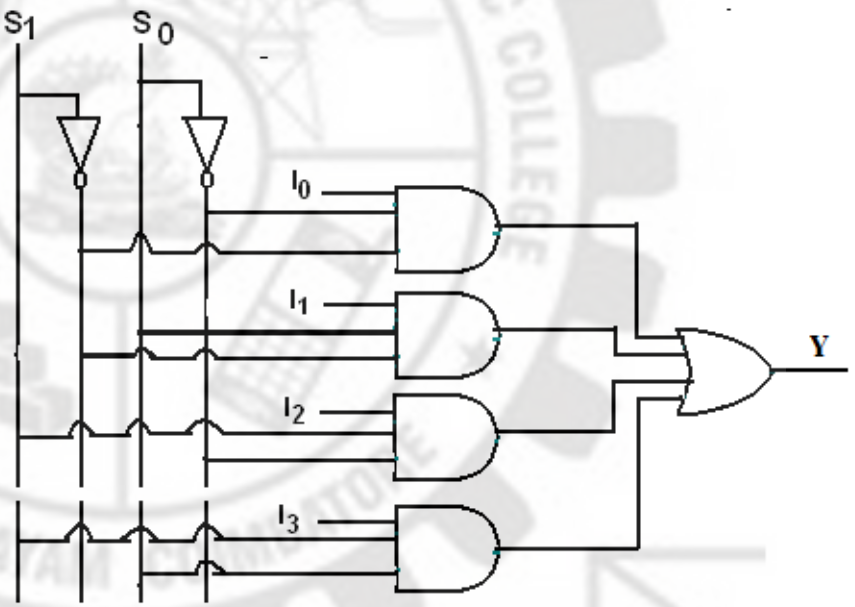
TRUTH TABLE

INPUT			OUTPUT	
A	B	Bin	D	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

4 TO 1 MUX

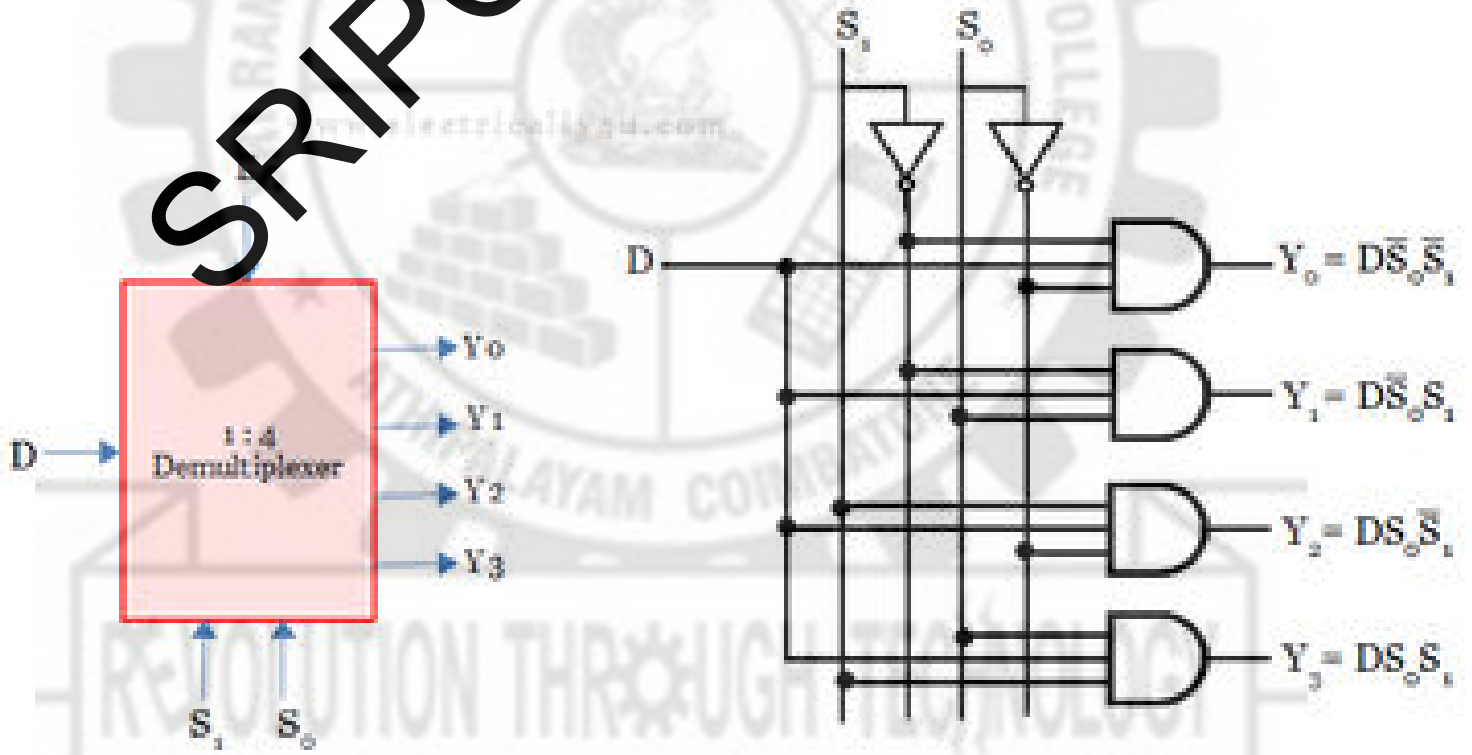
Input	S1	S0	Y
I_0	0	0	I_0
I_1	0	1	I_1
I_2	1	0	I_2
I_3	1	1	I_3

$$Y = S_1 S_0 I_3 + S_1 \overline{S_0} I_2 + \overline{S_1} S_0 I_1 + \overline{S_1}\,\overline{S_0} I_0$$

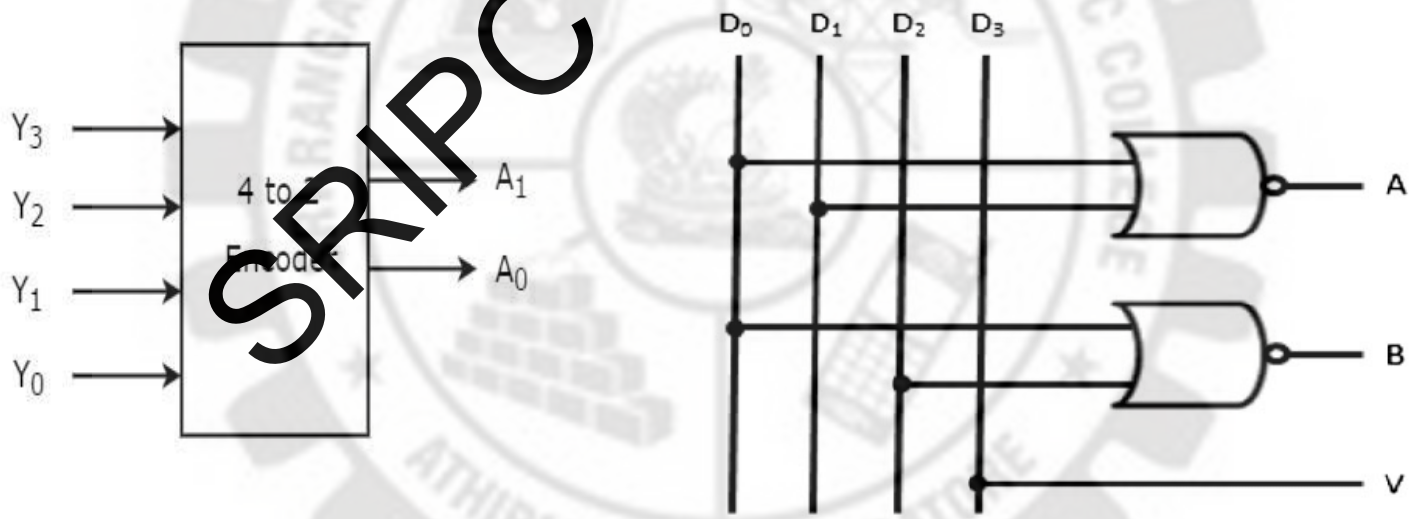


4 to 1 Multiplexer and its truth table

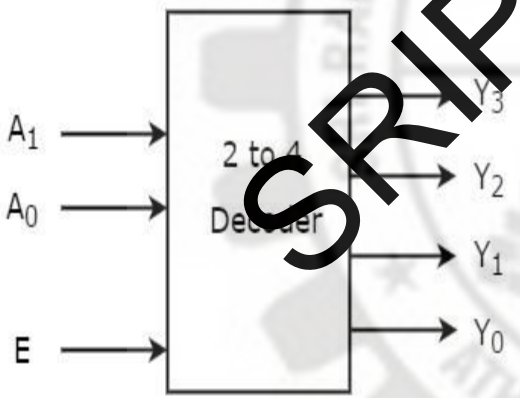
1 TO 4 DEMUX

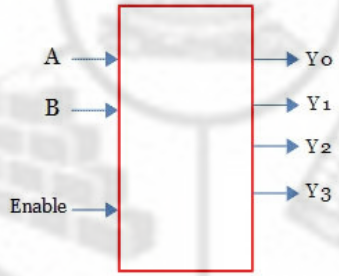


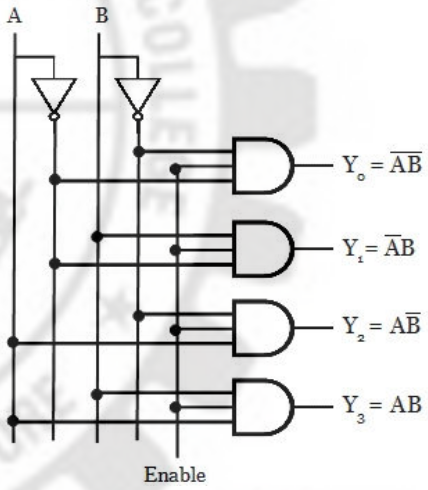
4 TO 2 ENCODER



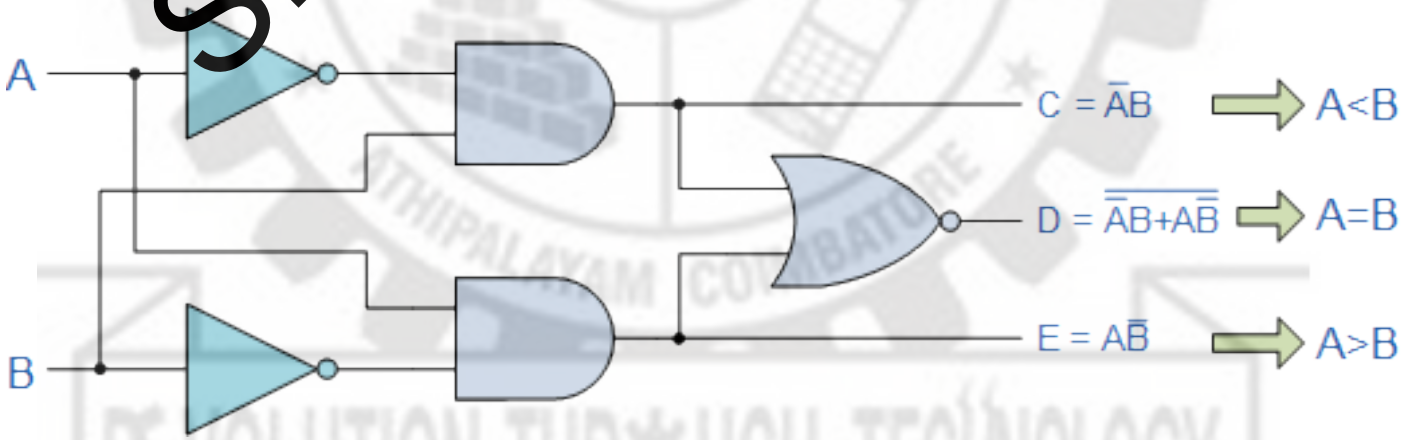
2 TO 4 DECODER



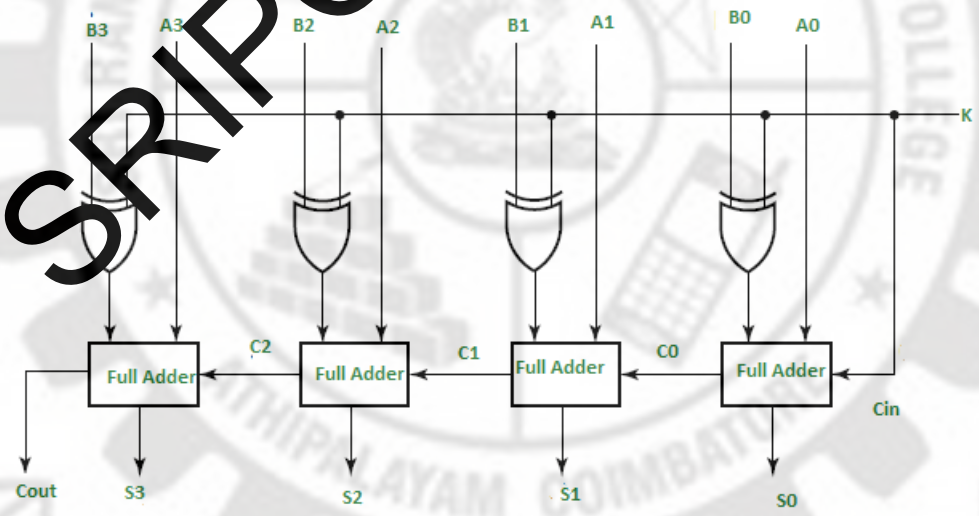




COMPARATOR

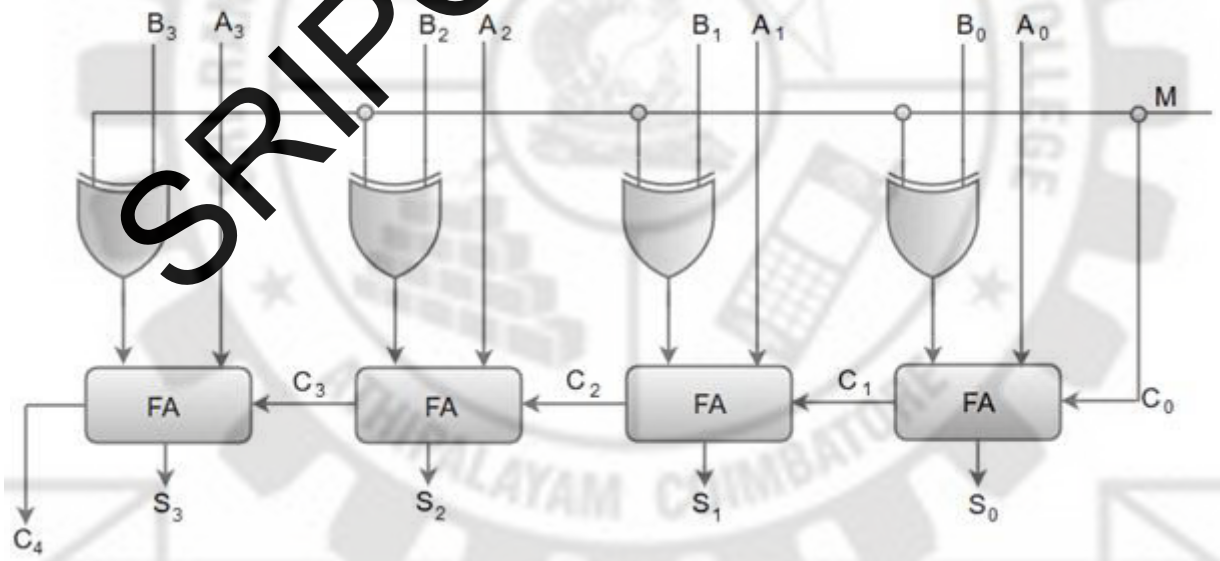


4 BIT ADDER



4 BIT SUBTRACTOR

4 bit adder-subtractor:



VHDL PROGRAM FOR HALF ADDER

VHDL Code Half adder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity half_adder is
   port(a,b:in bit; sum,carry:out bit);
end half_adder;

architecture data of half_adder is
begin
   sum<= a xor b;
   carry <= a and b;
end data;
```

VHDL PROGRAM FOR FULL ADDER

Full adder Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity full_adder is port
(a,b,c:in bit; sum,carry:out bit);
end full_adder;

architecture data of full_adder is
begin
   sum<= a xor b xor c;
   carry <= ((a and b) or (b and c) or (a and c));
end data;
```

VHDL PROGRAM FOR HALF SUBTRACTOR

- library IEEE;
- use IEEE.STD_LOGIC_1164.ALL; use IEEE.STD_LOGIC_ARITH.ALL;
- use IEEE.STD_LOGIC_UNSIGNED.ALL;
-
- Library ieee;
- use ieee.std_logic_1164.all;
-
- entity half_sub is
- port (a,b : in std_logic; dif,bo: out std_logic);
- end half_sub;
-
- architecture sub_arch of half_sub is begin
- dif <= a xor b;
- bo <= (not a) and b; end sub_arch;

VHDL PROGRAM FOR FULL SUBTRACTOR

- library IEEE;
- use IEEE.STD_LOGIC_1164.ALL; use IEEE.STD_LOGIC_ARITH.ALL;
- use IEEE.STD_LOGIC_UNSIGNED.ALL;
-
- entity full_sub is
- port(a,b,c: in bit; sub, borrow:out bit); end full_sub;
- architecture data of full_sub is beginsub<= a xor b xor c;
- borrow <= ((b xor c) and (not a))or (b and c);
- end data;

VHDL PROGRAM FOR SINGLE BIT DIGITAL COMPARATOR

- library IEEE;
- use IEEE.STD_LOGIC_1164.ALL;
- entity comparator_1bit is
- Port (A,B : in std_logic; G,S,E: out std_logic);
- end comparator_1bit;
- architecture comp_arch of comparator_1bit is begin
- G <= A and (not B); S <= (not A) and B; E <= A xnor B;
- end comp_arch;

VHDL PROGRAM FOR ENCODER

- library IEEE;
- use IEEE.STD_LOGIC_1164.ALL; use IEEE.STD_LOGIC_ARITH.ALL;
- use IEEE.STD_LOGIC_UNSIGNED.ALL;
- entity first is
- port (input : in std_logic_vector(3 downto 0); output : out std_logic_vector(1 downto 0)); end first;
- architecture Behavioral of first is begin
- process(input) begin
- case input is
- when "0001" =>output <= "00";
- when "0010" =>output <= "10";
- when "0100" =>output <= "01";
- when "1000" =>output <= "11"; when others =>null;
- end case; end process;
- end Behavioral;

VHDL PROGRAM FOR DECODER

- Libraryieee;
- use ieee.std_logic_1164.all; useieee.std_logic_arith.all; useieee.std_logic_unsigned.all;
- entityobject_co is port(clk : in std_logic;
- sw : in std_logic_vector(3 downto 0); y : out std_logic_vector(7 downto 0);
- -- sel : out std_logic_vector(5 downto 3) sel : out std_logic_vector(5 downto 0)
-);
- endobject_co;

Four bit Arithmetic adder

- LIBRARY IEEE;
- USE IEEE STD_LOGIC_1164.ALL;
- USE IEEE STD_LOGIC_ARITH.ALL;
- ENTITY ADDER_4 BIT IS
- PORT(A:IN STD _LOGIC_VECTOR(3 DOWN TO 0);
  (B:IN STD _LOGIC_VECTOR(3 DOWN TO 0);
- CARRY:OUT STD_LOGIC;
- S:OUT STD_LOGIC _VECTOR(3 DOWN TO 0);
- END ADDER_4 BIT;

- ARCHITECTURE ADDER_ARC OF ADDER_4 BIT IS
- COMPONENT FA
- PORT(X:INSTD_LOGIC;
- Y:IN STD_LOGIC;
- CI:IN STD_LOGIC;
- SUM:OUT STD_LOGIC;
- CY:OUT STD_LOGIC);
- END COMPONENT;
- SIGNAL C0,C1,C2,C3:STD_LKOGIC;
- BEGIN;

- BBAR<=NOT B;
- FA0:FA PORT MAP(A(0)),BBAR(0),'1',D(0)C(0);
- FA1:FA PORT MAP(A(1)),BBAR(0),'1',D(0)C(0);
- FA2:FA PORT MAP(A(2)),BBAR(0),'1',D(0)C(0);
- FA3:FA PORT MAP(A(3)),BBAR(0),'1',D(0)C(0);
- CARRY<=C3;
- END ARCH_4;

Four bit Arithmetic SUBTRACTOR

- LIBRARY IEEE;
- USE IEEE STD_LOGIC_1164.ALL;
- USE IEEE STD_LOGIC_ARITH.ALL;
- ENTITY SUB_4 BIT IS
- PORT(A:IN STD _LOGIC_VECTOR(3 DOWN TO 0);
  (B:IN STD _LOGIC_VECTOR(3 DOWN TO 0);
- D:OUT STD_LOGIC;
- OV:OUT STD_LOGIC _VECTOR(3 DOWN TO 0);
- END SUB_4 BIT;

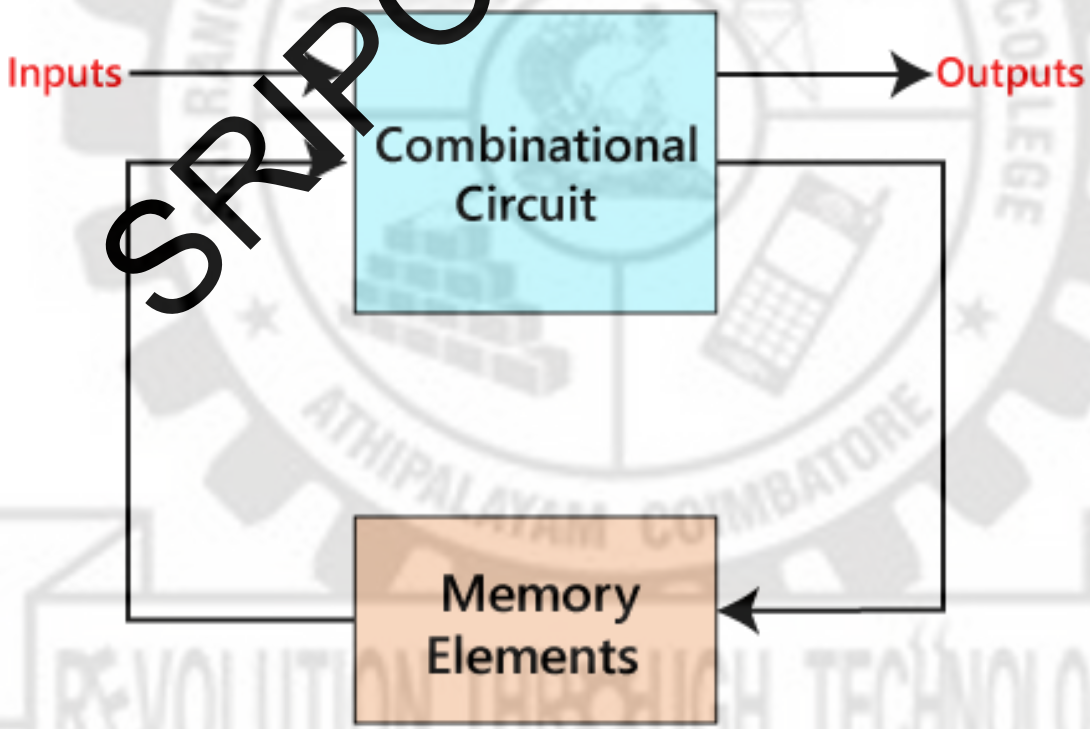
- ARCHITECTURE SUB_ARC OF ADDER_4 BIT IS
- COMPONENT FA
- PORT(X:INSTD_LOGIC;
- Y:IN STD_LOGIC;
- CI:IN STD LOGIC;
- SUM:OUT STD_LOGIC;
- CY:OUT STD_LOGIC);
- END COMPONENT;
- SIGNAL C0,C1,C2,C3:STD_LKOGIC;
- BEGIN;

- BBAR<=NOT B;
- FA0:FA PORT MAP(A(0)),BBAR(0),'1',D(0)C(0);
- FA1:FA PORT MAP(A(1)),BBAR(0),'1',D(0)C(0);
- FA2:FA PORT MAP(A(2)),BBAR(0),'1',D(0)C(0);
- FA3:FA PORT MAP(A(3)),BBAR(0),'1',D(0)C(0);
- OV<=C2 XOR C3;
- END SUB_4;

UNIT-IV
SEQUENTIAL CIRCUIT DESIGN

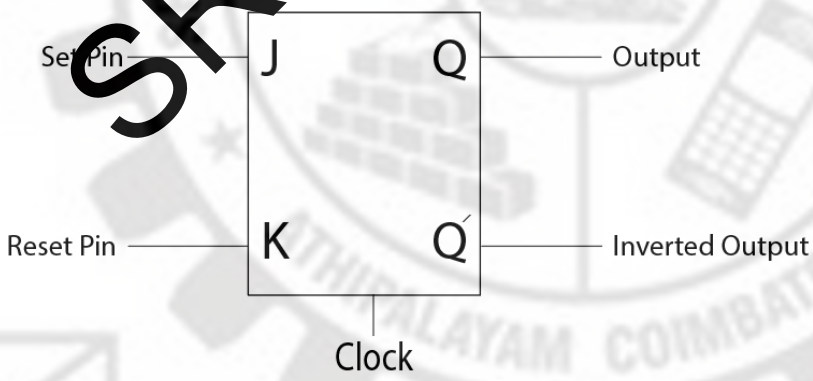
SWATHI E R
LECTURER/ECE

SEQUENTIAL CIRCUIT DESIGN



JK FLIP FLOPS

JK Flip-Flop Symbol



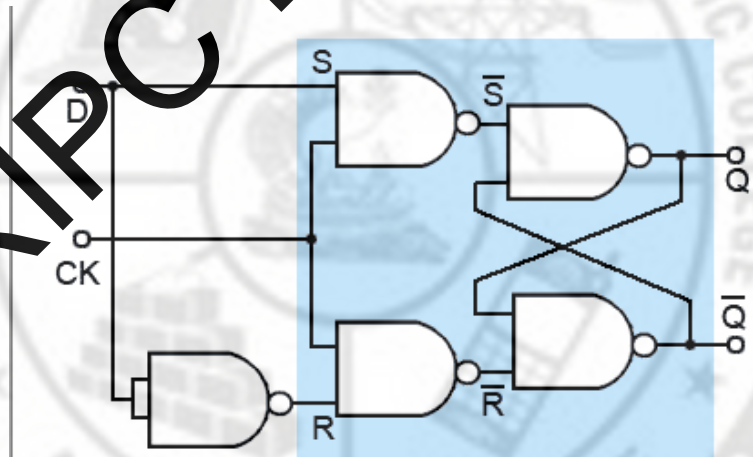
JK Flip-Flop Logic Table

C	J	K	Q	Q`
HIGH	0	0	Latch	Latch
HIGH	0	1	0	1
HIGH	1	0	1	0
HIGH	1	1	Toggle	Toggle
LOW	0	0	Latch	Latch
LOW	0	1	Latch	Latch
LOW	1	0	Latch	Latch
LOW	1	1	Latch	Latch

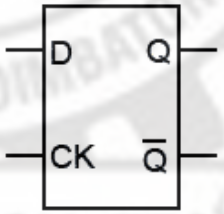
Introduction to JK Flip Flop

www.TheEngineeringProjects.com

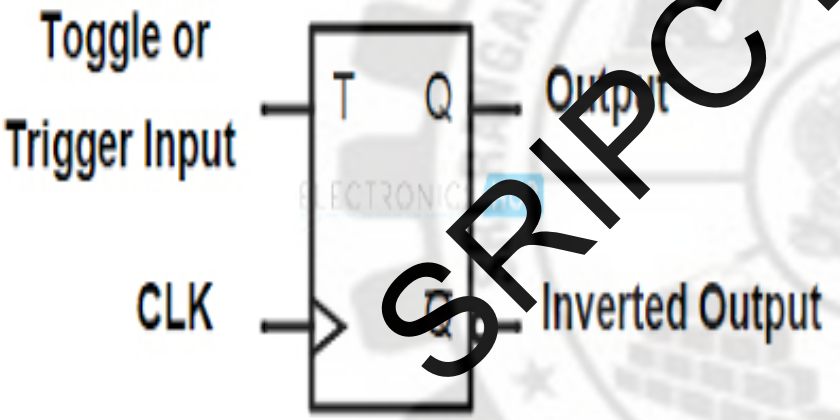
D FLIP FLOPS



Inputs		Outputs	
CK	D	Q	$\overline{Q}$
0	X	No change	
1	0	0	1
1	1	1	0

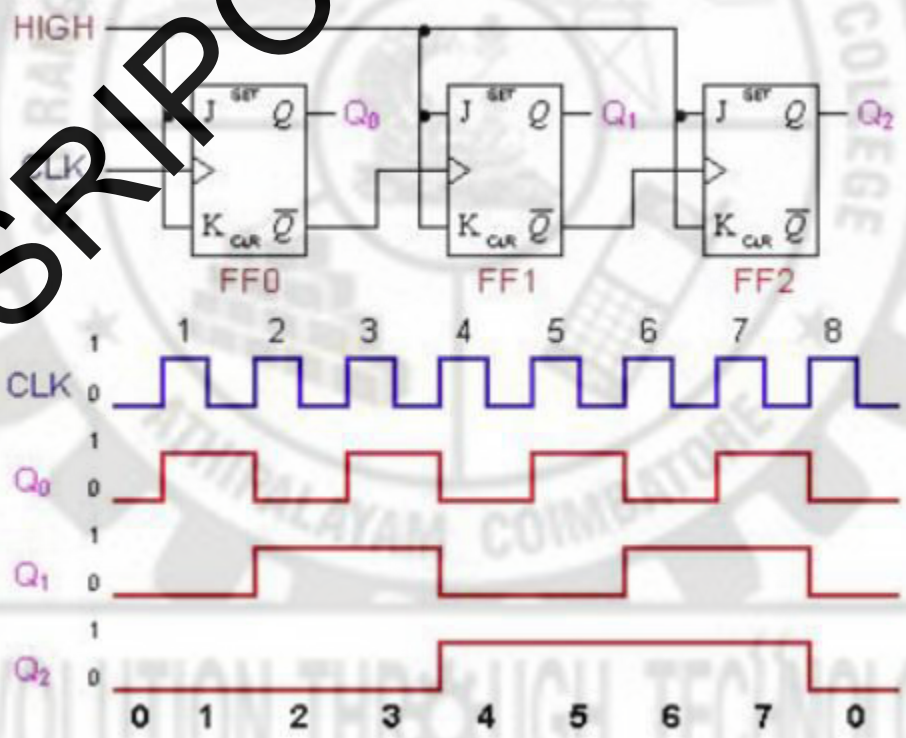


T FLIPFLOPS



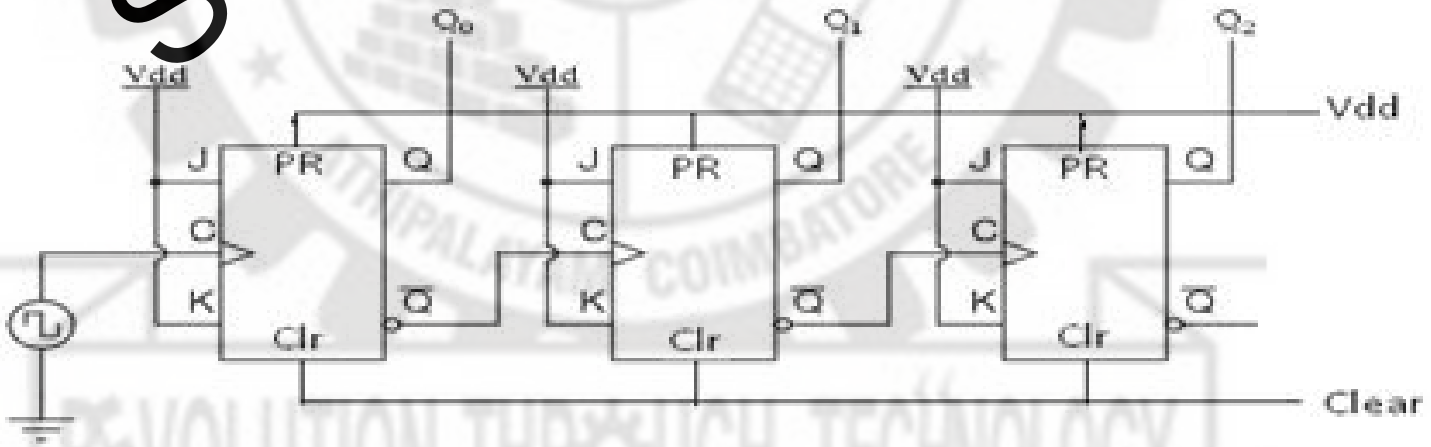
T	Qn	Qn+1	J	K
0	0	0	0	x
0	1	1	x	0
1	0	1	1	x
1	1	0	x	1

3 BIT SYNCHRONOUS UP COUNTER

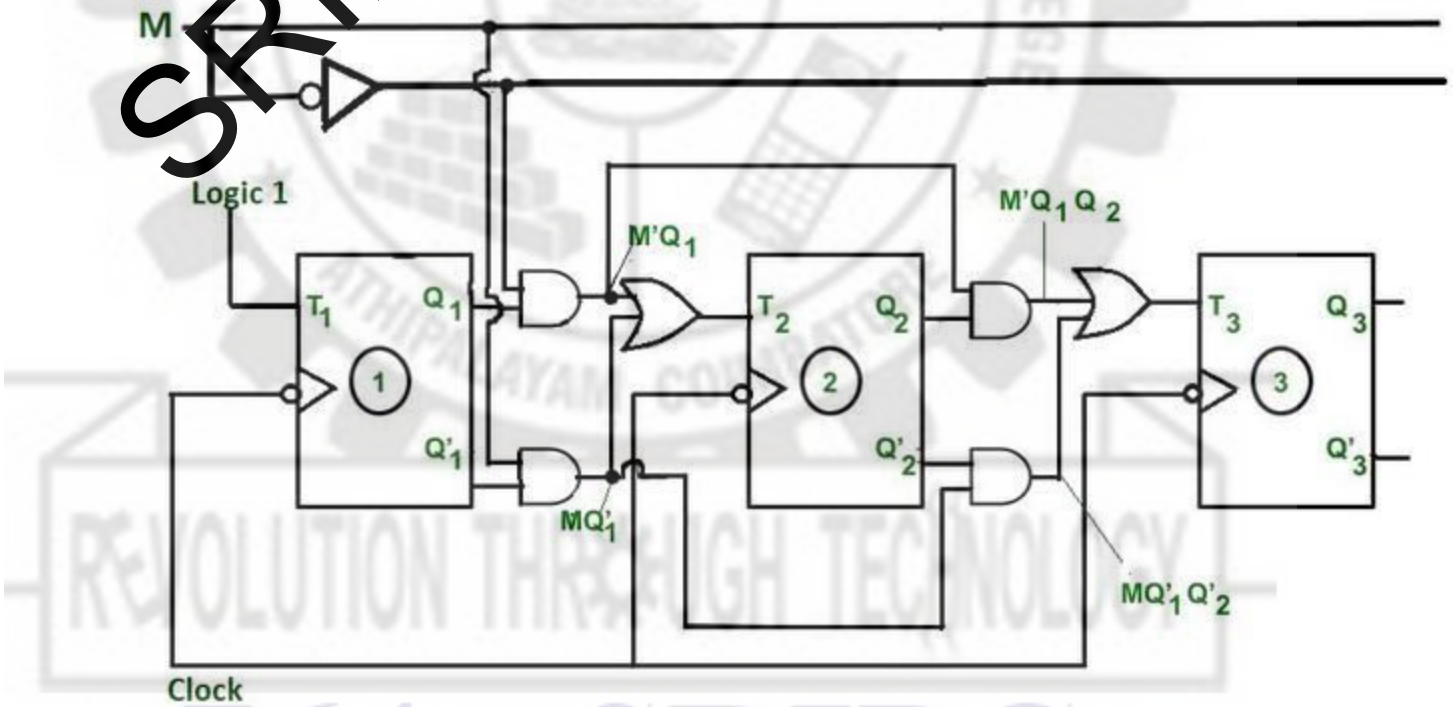


3 BIT SYNCHRONOUS DOWN COUNTER

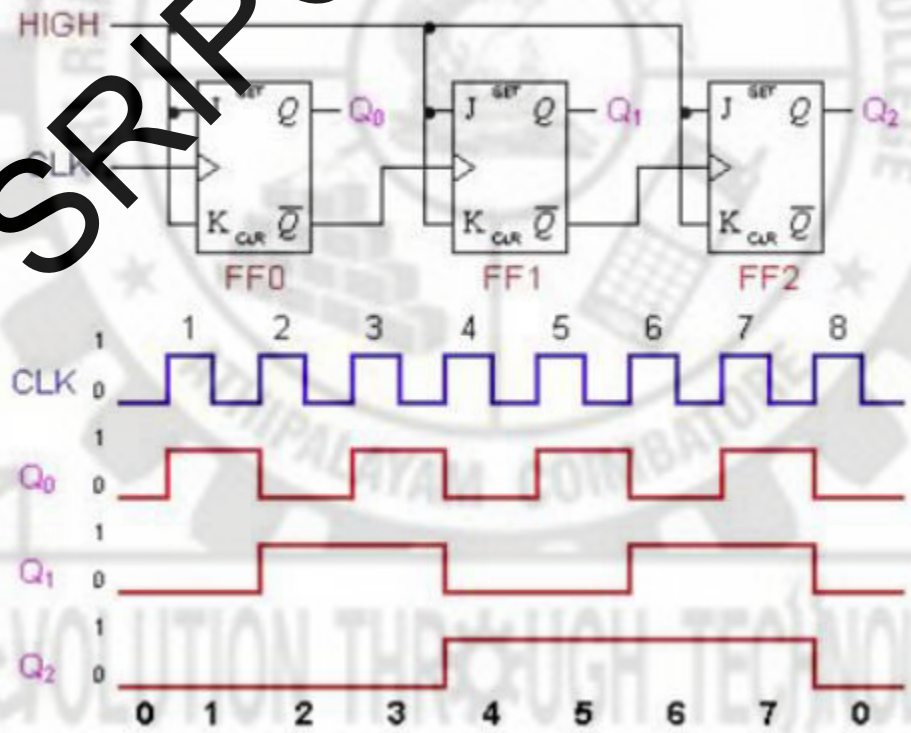
Logic Diagram for 3-bit Down Counter:



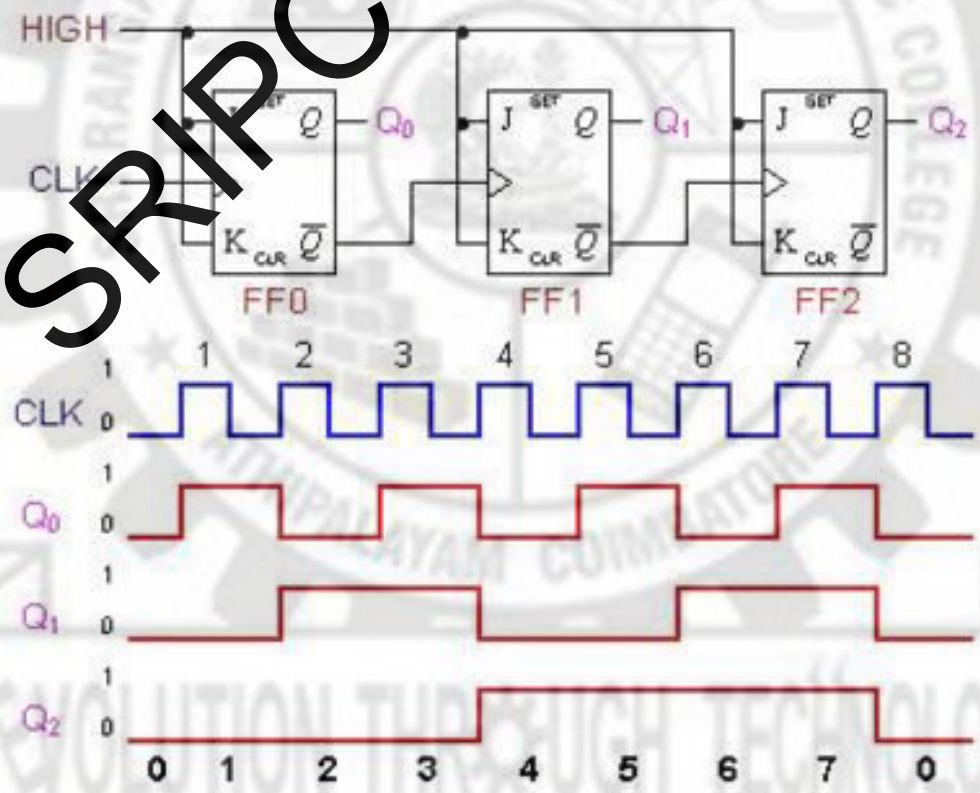
3 BIT SYNCHRONOUS UP/DOWN



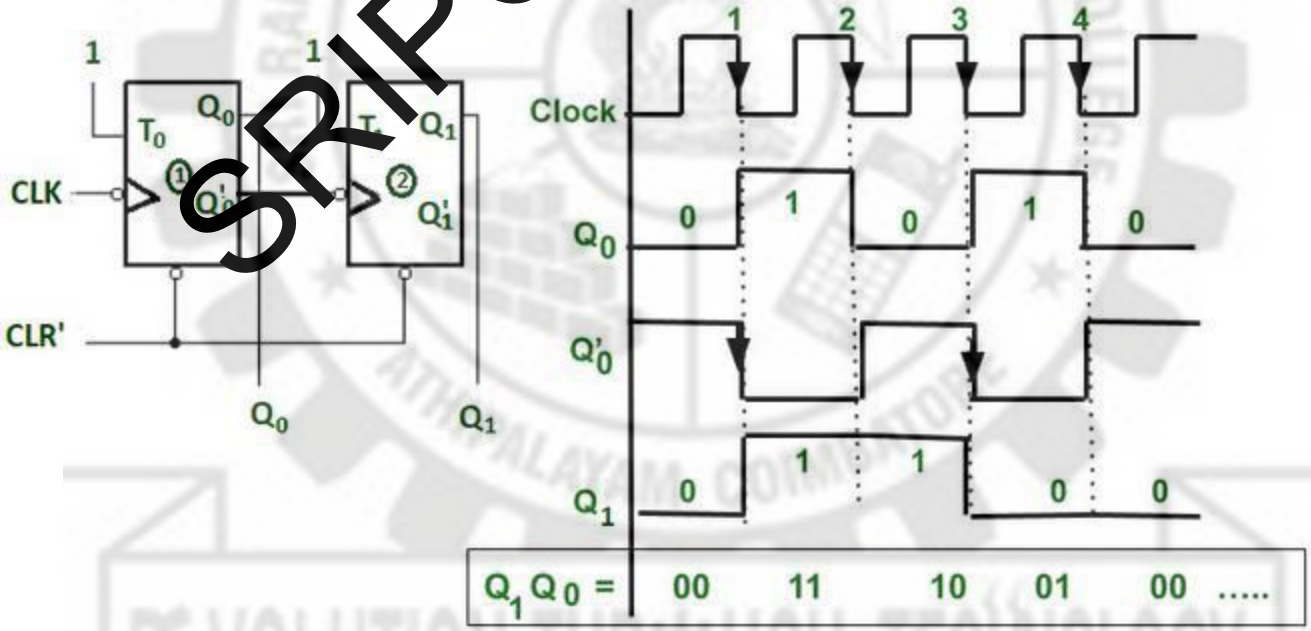
3 BIT ASYNCHRONOUS UP COUNTER



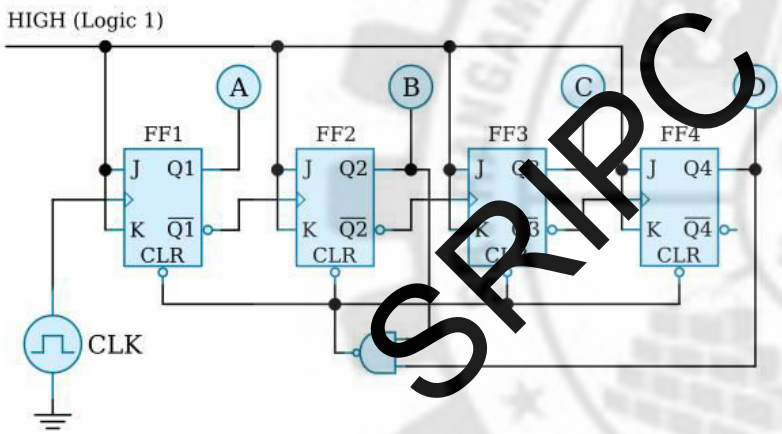
3 BIT ASYNCHRONOUS DOWN COUNTER



3 BIT ASYNCHRONOUS UP/DOWN COUNTER

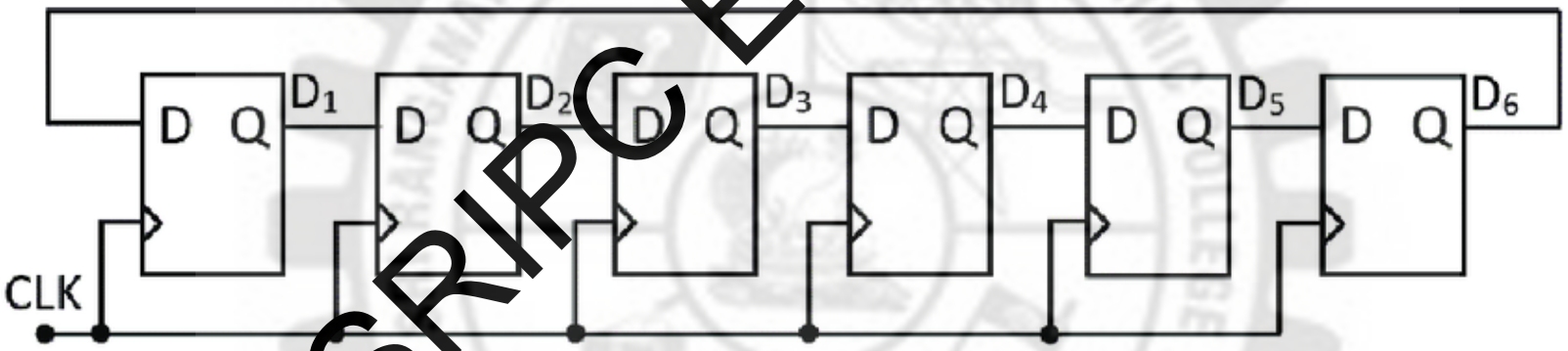


Decade counter

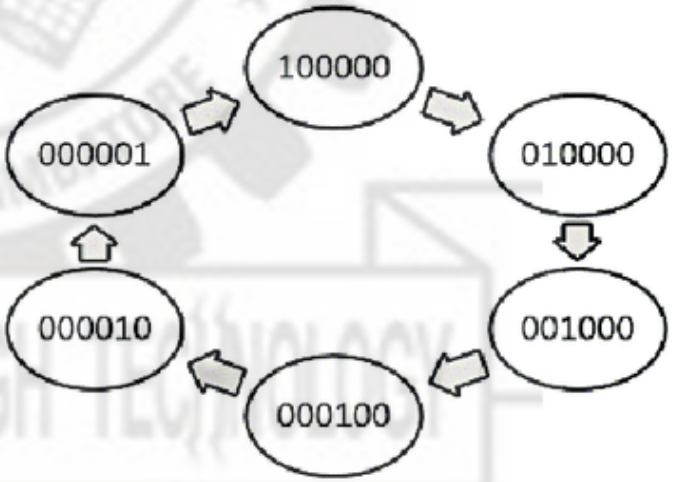


Input Pulses	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
0	0	0	0	0 (resets)

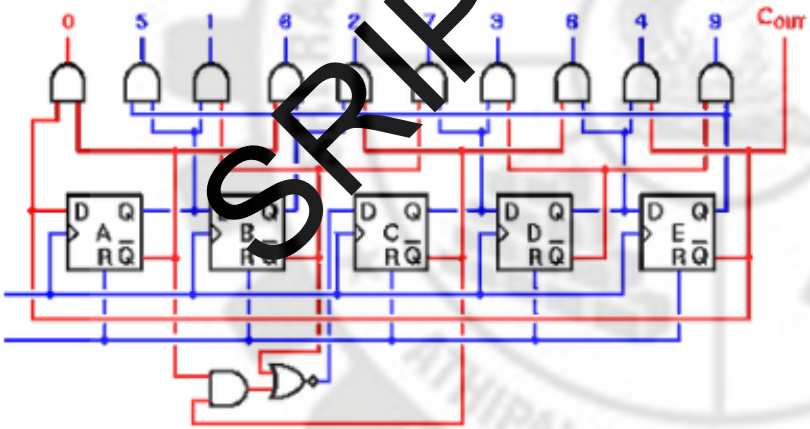
RING COUNTER

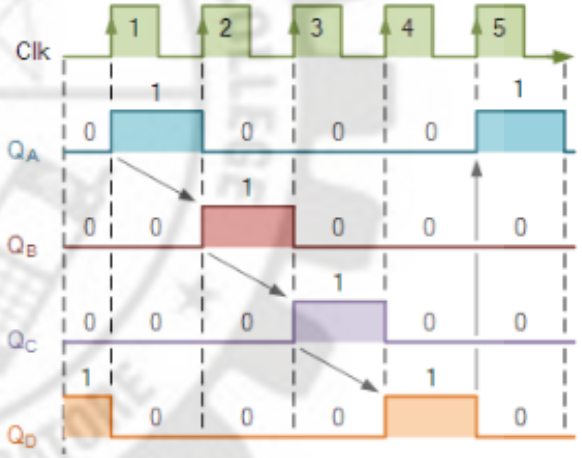


CLK	D_1	D_2	D_3	D_4	D_5	D_6
1	1	0	0	0	0	0
2	0	1	0	0	0	0
3	0	0	1	0	0	0
4	0	0	0	1	0	0
5	0	0	0	0	1	0
6	0	0	0	0	0	1



What is Johnson Counter?





D FLIP FLOP


```
Library IEEE;
 USE IEEE.Std_logic_1164.all;
entity RisingEdge_DFlipFlop is
port( Q : out std_logic; Clk :in std_logic;
 D :in std_logic );
end RisingEdge_DFlipFlop;
 architecture Behavioral of RisingEdge_DFlipFlop is
 begin
 process(Clk)
begin if(rising_edge(Clk))
 then Q <= D;
end if;
end process;
 end Behavioral;
```

D FLIPFLOP with RESET


```
Library IEEE;
 USE IEEE.Std_logic_1164.all;
entity RisingEdge_DFlipFlop is
port( reset,Q : out std_logic; Clk :in std_logic;
 D :in std_logic );
end RisingEdge_DFlipFlop;
 architecture Behavioral of RisingEdge_DFlipFlop is
 begin
 process(Clk)
Reset="1 then temp='0';"
begin if(rising_edge(Clk))
 then Q <= D;
end if;
end process;
 end Behavioral;
```

T FLIP FLOPS


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity T_FF is
port( T: in std_logic;
Clock: in std_logic;
Q: out std_logic);
end T_FF;
architecture Behavioral of T_FF is
signal tmp: std_logic;
begin
process (Clock)
begin
if Clock'event and Clock='1' then
 if T='0' then
tmp <= tmp;
elsif T='1' then
tmp <= not (tmp);
end if;
end if;
end process;
Q <= tmp;
end Behavioral;
```

T FLIP FLOPS with RESET


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity T_FF is
port( reset,T: in std_logic;
Clock: in std_logic;
Q: out std_logic);
end T_FF;
architecture Behavioral of T_FF is
signal tmp: std_logic;
begin
process (Clock)
begin
if Clock'event and Clock='1' then
Reset=“1 then temp=‘0’;”
 if T='0' then
tmp <= tmp;
elsif T='1' then
tmp <= not (tmp);
end if;
end if;
end process;
Q <= tmp;
end Behavioral;
```

ECE

J K FLIP FLOPS


```
library ieee;
use ieee. std_logic_1164.all;
use ieee. std_logic_arith.all;
use ieee. std_logic_unsigned.all;
entity JK_FF is
PORT( J,K,CLOCK: in std_logic;
Q, QB: out std_logic);
end JK_FF;
Architecture behavioral of JK_FF is
begin
PROCESS(CLOCK)
variable TMP: std_logic;
begin
if(CLOCK='1' and CLOCK'EVENT) then
if(J='0' and K='0')then
TMP:=TMP;
elsif(J='1' and K='1')then
TMP:= not TMP;
elsif(J='0' and K='1')then
TMP:='0';
else
TMP:='1';
end if;
end if;
Q<=TMP;
Q <=not TMP;
end PROCESS;
end behavioral;
```

J K FLIP FLOPS with RESET


```
library ieee;
use ieee. std_logic_1164.all;
use ieee. std_logic_arith.all;
use ieee. std_logic_unsigned.all;
entity JK_FF is
PORT(reset   J,K,CLOCK: in std_logic;
Q, QB: out std_logic);
end JK_FF;
Architecture behavioral of JK_FF is
begin
PROCESS(CLOCK)
variable TMP: std_logic;
begin
if(CLOCK='1' and CLOCK'EVENT) then
Reset=“1 then temp=‘0’;”
if(J='0' and K='0')then
TMP:=TMP;
elsif(J='1' and K='1')then
TMP:= not TMP;
elsif(J='0' and K='1')then
TMP:='0';
else
TMP:='1';
end if;
end if;
Q<=TMP;
Q <=not TMP;
end PROCESS;
```

VHDL code for synchronous up-counter


```
library IEEE;
 use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL
; use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity SOURCE is
Port ( CLK,RST : in  STD_LOGIC;
   COUNT : in out  STD_LOGIC_VECTOR (3 downto 0));
end SOURCE;
architecture Behavioral of SOURCE is
begin
process (CLK,RST)
begin if (RST = '1')
then COUNT <= "0000";
elsif(rising_edge(CLK))
then COUNT <= COUNT+1;
end if;
end process;
end Behavioral;
```

VHDL code for synchronous down-counter


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity is
  Port ( clk,rst : in  STD_LOGIC;
 count : out  STD_LOGIC_VECTOR (3 downto 0));
end;
architecture Behavioral of down_count is
signal temp:std_logic_vector(3 downto 0);
begin process(clk,rst)
begin if(rst='1')
then temp<="1111";
elsif(rising_edge(clk))
then temp<=temp-1;
 end if;
end process;
 count<=temp;
end Behavioral;
```

VHDL code for synchronous updown-counter


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
 use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity updown_count is
    Port ( clk,rst,updown : in  STD_LOGIC;
 count : out  STD_LOGIC_VECTOR (3 downto 0));
end updown_count;
architecture Behavioral of updown_count is
signal temp:std_logic_vector(3 downto 0):="0000";
Begin
 process(clk,rst)
begin if(rst='1')
then temp<="0000“
; elsif(rising_edge(clk))
then if(updown='0')
then temp<=temp+1;
else temp<=temp-1;
end if; end if;
end process;
count<=temp;
end Behavioral;
```

VHDL CODE FOR DECADE COUNTER


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity decade is
   Port ( CLOCK : in  STD_LOGIC;
          RESET : in  STD_LOGIC;
          Q : out  STD_LOGIC_VECTOR (3 downto 0));
end decade;
architecture Behavioral of decade is
signal q_tmp: std_logic_vector(3 downto 0):= "0000";
begin
process(CLOCK,RESET)
begin
if RESET = '1' then
   count <= "0000";
Else if clock'event AND clock='1' then
If count<count+'1';
Else
  count <= "0000";
  end if
Q <= count;
end process;
end Behavioral;
```

VHDL CODE FOR RING COUNTER


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Ring_counter is
    Port ( CLOCK : in  STD_LOGIC;
           RESET : in  STD_LOGIC;
           Q : out  STD_LOGIC_VECTOR (3 downto 0));
end Ring_counter;

architecture Behavioral of Ring_counter is
signal q_tmp: std_logic_vector(3 downto 0):= "0000";
begin
process(CLOCK,RESET)
begin
if RESET = '1' then
    q_tmp <= "0001";
elsif Rising_edge(CLOCK) then
    q_tmp(1) <= q_tmp(0);
    q_tmp(2) <= q_tmp(1);
    q_tmp(3) <= q_tmp(2);
    q_tmp(0) <= q_tmp(3);
end if;
end process;
Q <= q_tmp;
end Behavioral;
```

VHDL CODE FOR JOHNSON COUNTER

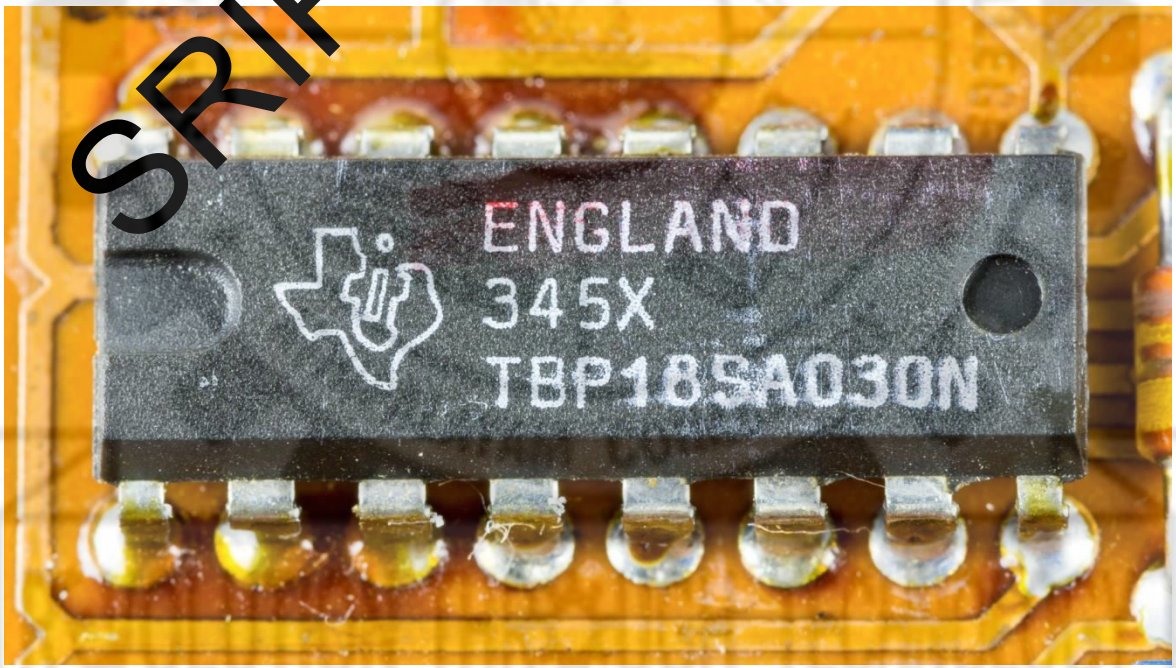

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Johnson_counter is
Port ( clk : in STD_LOGIC;
rst : in STD_LOGIC;
Q : out STD_LOGIC_VECTOR (3 downto 0));
end Johnson_counter;

architecture Behavioral of Johnson_counter is
signal temp: std_logic_vector(3 downto 0):= "0000";
begin
process(clk,rst)
begin
if rst = '1' then
temp <= "0000";
elsif Rising_edge(clk) then
temp(1) <= temp(0);
temp(2) <= temp(1);
temp(3) <= temp(2);
temp(0) <= not temp(3);
end if;
end process;
Q <= temp;
end Behavioral;
```

UNIT-V
PROGRAMMABLE LOGIC DEVICES

BY
SWATHI E R
LECTURER
ECE

PROM

PROM chips have several different applications, including **cell phones, video game consoles, RFID tags, medical devices, and other electronics**. They provide a simple means of programming electronic devices. Standard PROM can only be programmed once.



PAL

Programmable Array Logic (PAL) is a **type of semiconductor used to implement logic functions in digital circuits.** PAL is a type of programmable logic device, which is a term for an integrated circuit that can be programmed in a laboratory to perform complex functions

COMPARE PROM,PAL PLA

Ans :

P(ROM)	PAL	PLA
1. AND array is fixed, OR array is programmable.	1.OR array is fixed, AND is array programmable.	1.Both arrays are programmable.
2.Only SSOP type Boolean function or expression can be implement.	2.Any SOP type Boolean expression can be implement.	2.Any SOP type expression can be implement.
3. Cost is low.	3. Cost is low.	3. Costlier.
4. Simple to construct.	4. Simple to construct.	4. Complex to construct.

- Available choice for digital designer
- FPGA – A detailed look
- Interconnection Framework
  - FPGAs and CPLDs
- Field programmability and programming technologies
  - SRAM, Anti-fuse, EPROM and EEPROM

Designer's Choice

- Digital designer has various options
  - SSI (small scale integrated circuits) or MSI (medium scale integrated circuits) components
    - Difficulties arises as design size increases
    - Interconnections grow with complexity resulting in a prolonged testing phase
  - Simple programmable logic devices
    - PALs (programmable array logic)
    - PLAs (programmable logic array)
      - Architecture not scalable; Power consumption and delays play an important role in extending the architecture to complex designs
      - Implementation of larger designs leads to same difficulty as that of discrete components

Designer's Choice

- Quest for high capacity; Two choices available
  - MPGA (Masked Programmable Logic Devices)
    - Customized during fabrication
    - Low volume expensive
    - Prolonged time-to-market and high financial risk
  - FPGA (Field Programmable Logic Devices)
    - Customized by end user
    - Implements multi-level logic function
    - Fast time to market and low risk

FPGA – A Quick Look

- Two dimensional array of customizable logic block placed in an interconnect array
- Like PLDs programmable at users site
- Like MPGAs implements thousands of gates of logic in a single device
  - Employs logic and interconnect structure capable of implementing multi-level logic
  - Scalable in proportion with logic removing many of the size limitations of PLD derived two level architecture
- FPGAs offer the benefit of both MPGAs and PLDs!

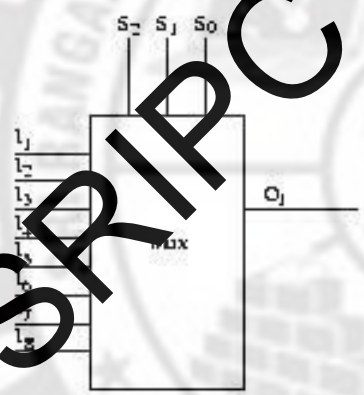
FPGA – A Detailed Look

- Based on the principle of functional completeness
- FPGA: Functionally complete elements (Logic Blocks) placed in an interconnect framework
- Interconnection framework comprises of wire segments and switches; Provide a means to interconnect logic blocks
- Circuits are partitioned to logic block size, mapped and routed

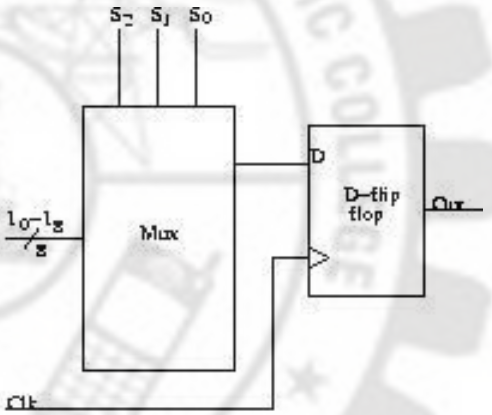
A Fictitious FPGA Architecture

(With Multiplexer As Functionally Complete Cell)

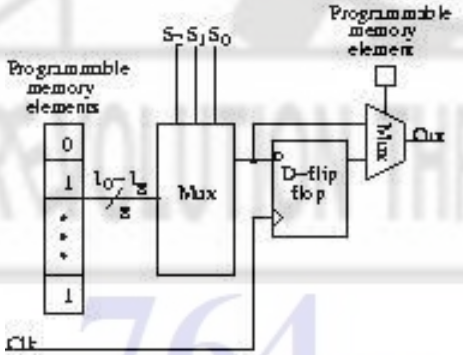
- Basic building block



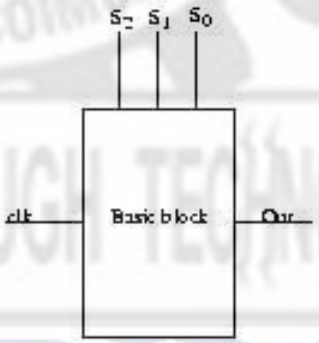
(A) Mux with 3 select inputs



(B) Mux with output driving D-flipflop



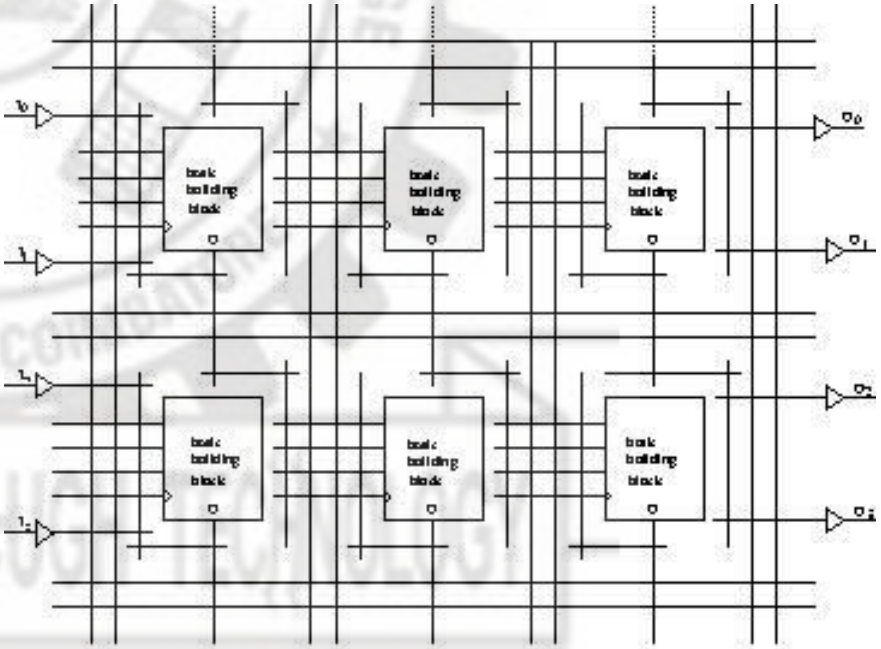
(C) Selection of combinational or sequential output



(D) Shorthand image of basic building block

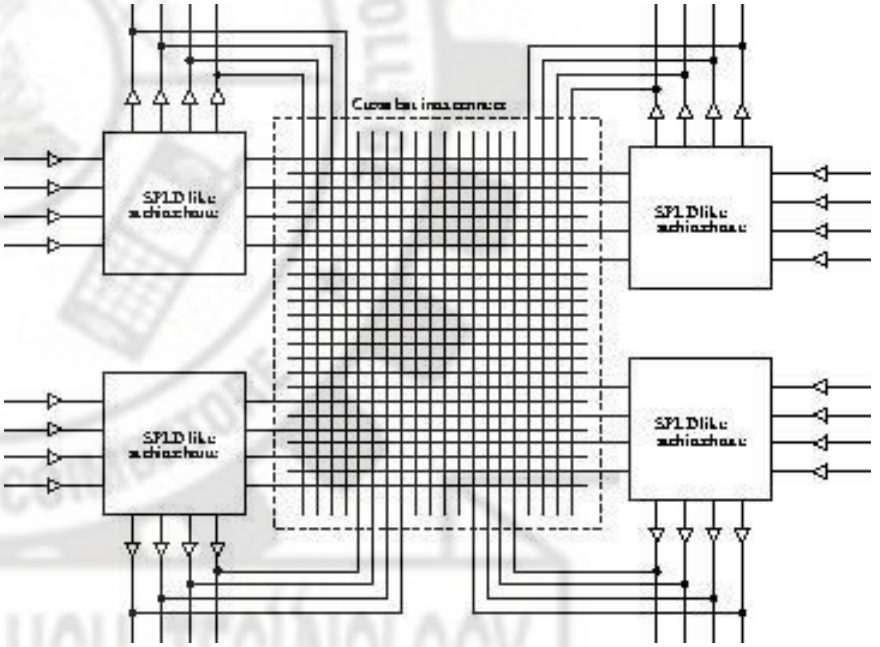
Interconnection Framework

- Granularity and interconnection structure has caused a split in the industry
  - FPGA
    - Fine grained
    - Variable length interconnect segments
    - Timing in general is not predictable; Timing extracted after placement and route



Interconnection Framework

- CPLD
  - Coarse grained (SPLD like blocks)
  - Programmable crossbar interconnect structure
  - Interconnect structure uses continuous metal lines
  - The switch matrix may or may not be fully populated
  - Timing predictable if fully populated
  - Architecture does not scale well



Field Programmability

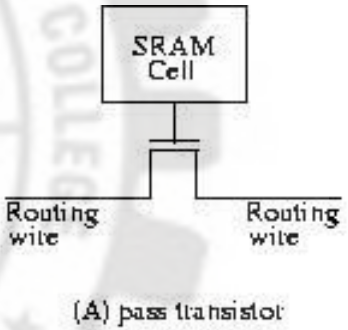
- Field programmability is achieved through switches (Transistors controlled by memory elements or fuses)
- Switches control the following aspects
  - Interconnection among wire segments
  - Configuration of logic blocks
- Distributed memory elements controlling the switches and configuration of logic blocks are together called "Configuration Memory"

Technology of Programmable Elements

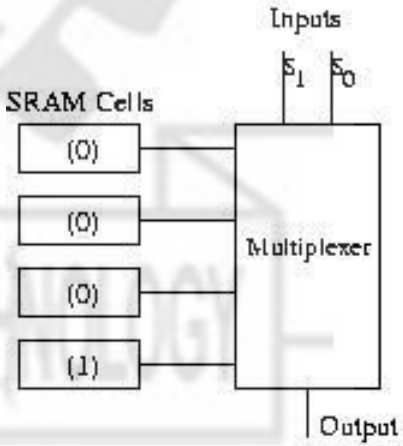
- Vary from vendor to vendor. All share the common property: Configurable in one of the two positions – 'ON' or 'OFF'
- Can be classified into three categories:
  - SRAM based
  - Fuse based
  - EPROM/EEPROM/Flash based
- Desired properties:
  - Minimum area consumption
  - Low on resistance; High off resistance
  - Low parasitic capacitance to the attached wire
  - Reliability in volume production

SRAM Programming Technology

- Employs SRAM (Static RAM) cells to control pass transistors and/or transmission gates
- SRAM cells control the configuration of logic block as well
- Volatile
  - Needs an external storage
  - Needs a power-on configuration mechanism
  - In-circuit re-programmable
- Lesser configuration time
- Occupies relatively larger area

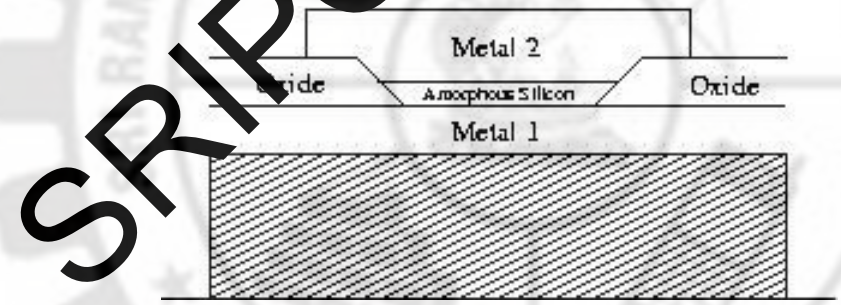


(A) pass transistor



(B) Multiplexer

Anti-fuse Programming Technology

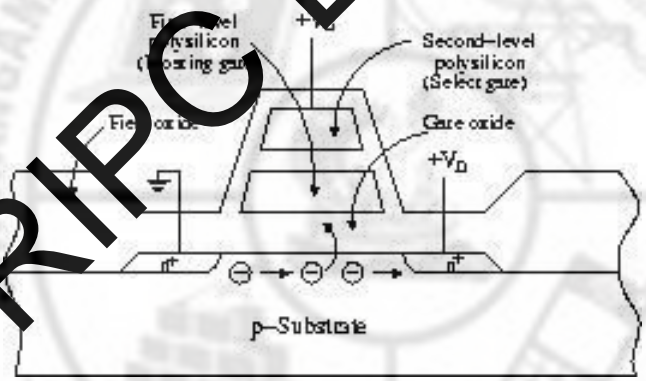


- Though implementation differ, all anti-fuse programming elements share common property
  - Uses materials which normally resides in high impedance state
  - But can be fused irreversibly into low impedance state by applying high voltage

Anti-fuse Programming Technology

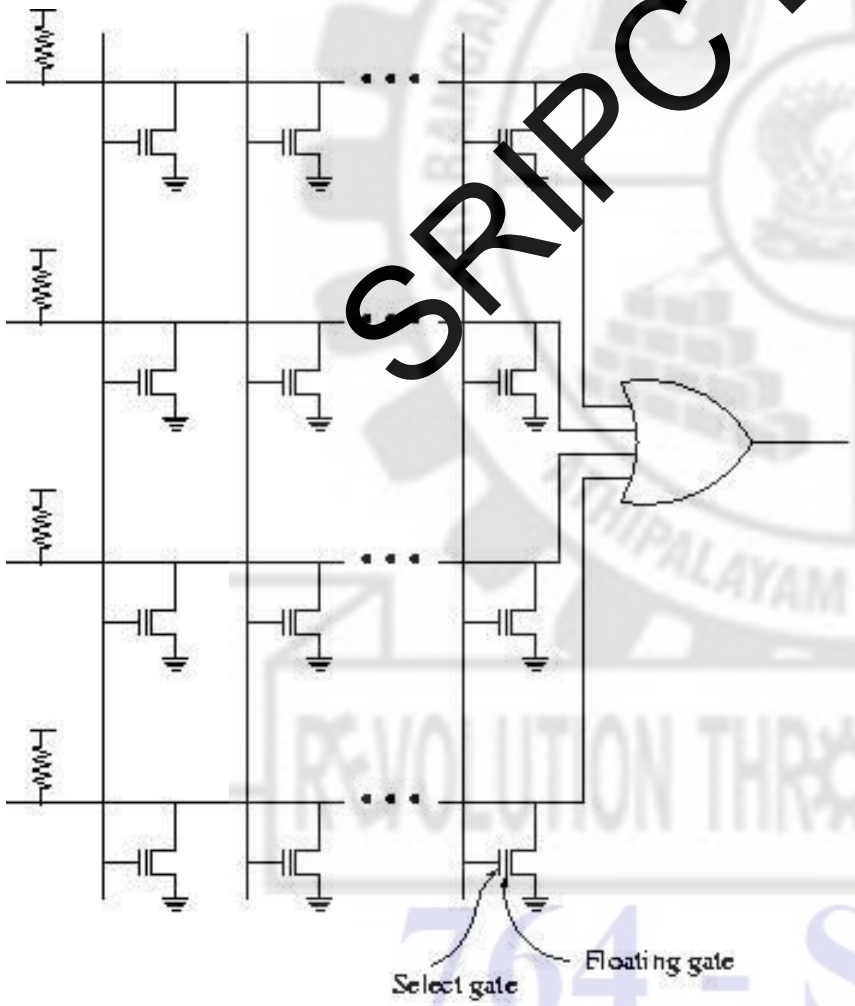
- Very low ON Resistance (Faster implementation of circuits)
- Limited size of anti-fuse elements; Interconnects occupy relatively lesser area
  - Offset : Larger transistors needed for programming
- One Time Programmable
  - Cannot be re-programmed
    - (Design changes are not possible)
  - Retain configuration after power off

EPROM, EEPROM or Flash Based Programming Technology



- EPROM Programming Technology
  - Two gates: Floating and Select
  - Normal mode:
    - No charge on floating gate
    - Transistor behaves as normal n-channel transistor
  - Floating gate charged by applying high voltage
    - Threshold of transistor (as seen by gate) increases
    - Transistor turned off permanently
  - Re-programmable by exposing to UV radiation

EPROM Programming Technology



- Used as pull-down devices
- Consumes static power

EPROM Programming Technology

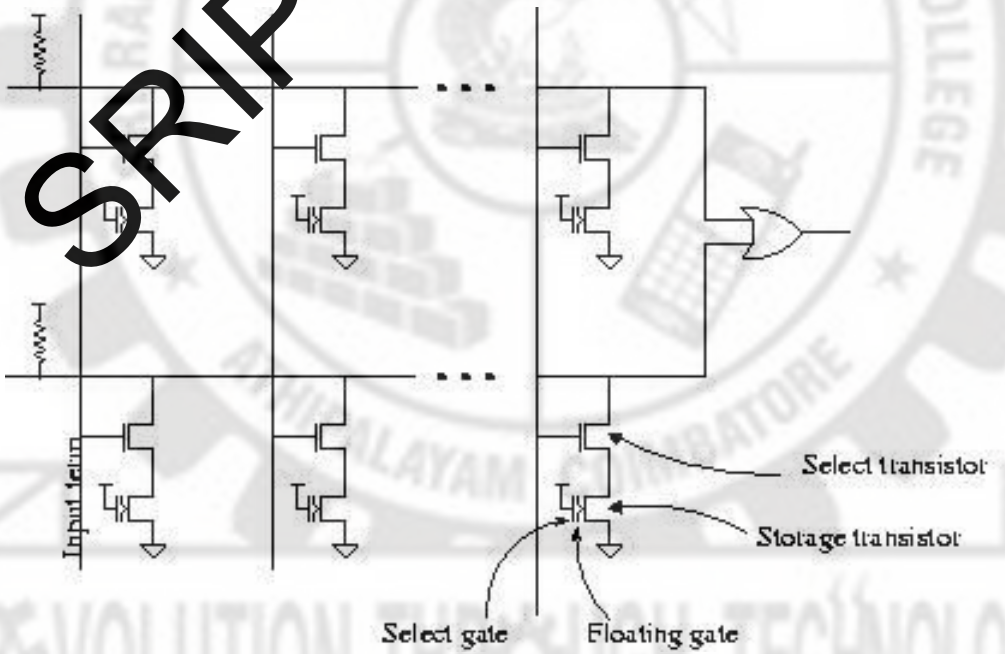
- No external storage mechanism
- Re-programmable (Not all!)
- Not in-system re-programmable
- Re-programming is a time consuming task

EEPROM Programming Technology

- Two gates: Floating and Select
- Functionally equivalent to EPROM; Construction and structure differ
- Electrically Erasable: Re-programmable by applying high voltage

  (No UV radiation expose!)
- When un-programmed, the threshold (as seen by select gate) is negative!

EEPROM Programming Technology

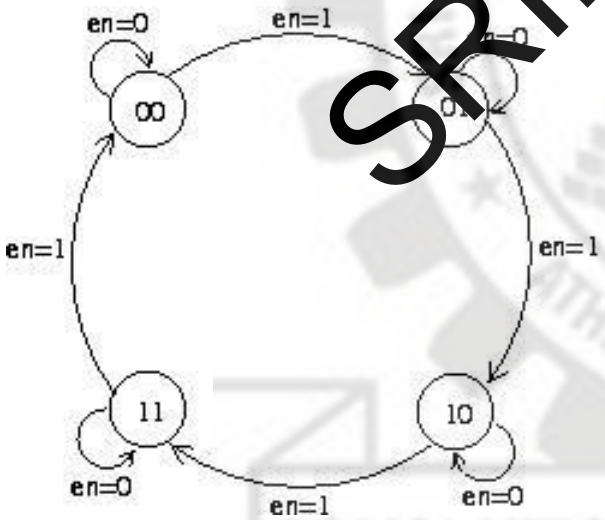


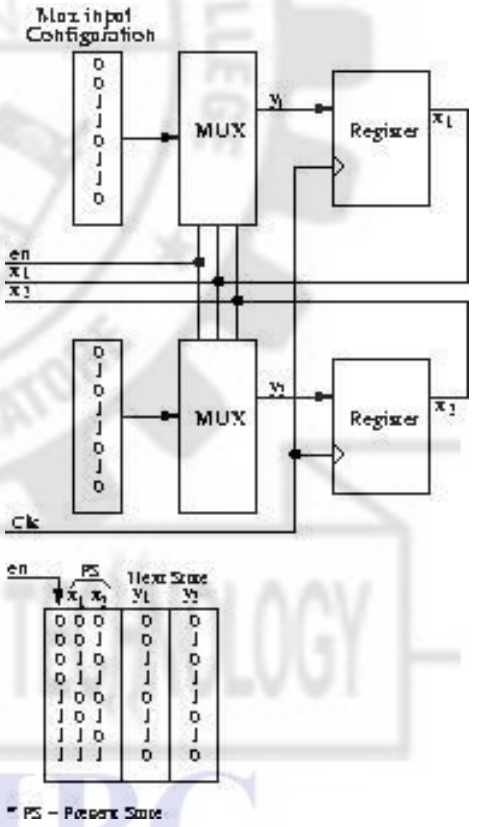
EEPROM Programming Technology

- Re-programmable, In general, in-system re-programmable
- Re-programming consumes lesser time compared to EPROM technology
- Multiple voltage sources may be required
- Area occupied is twice that of EPROM!

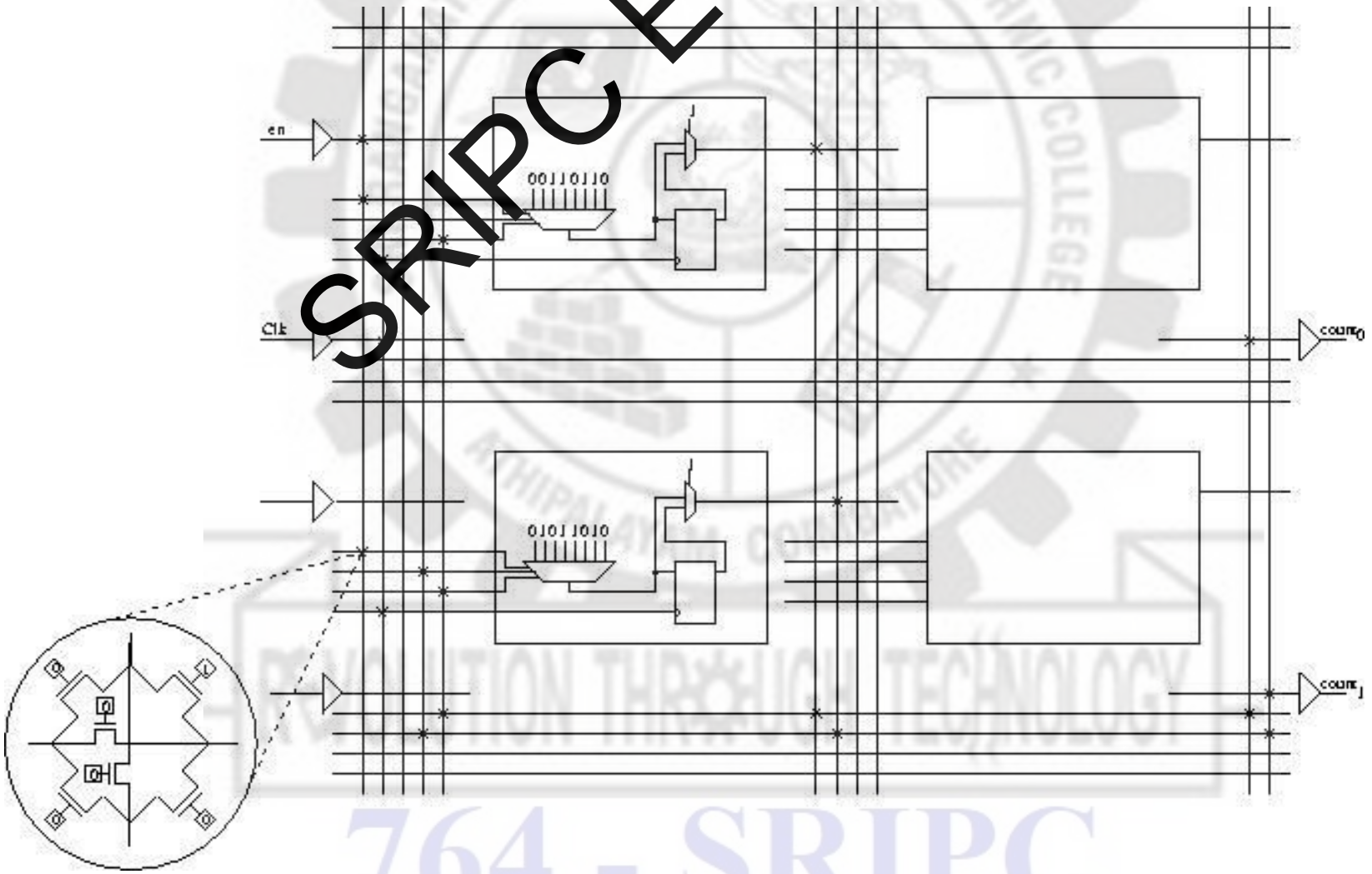
An Example

- Modulo-4 counter: Specification
- Modulo-4 counter: Logic Implementation

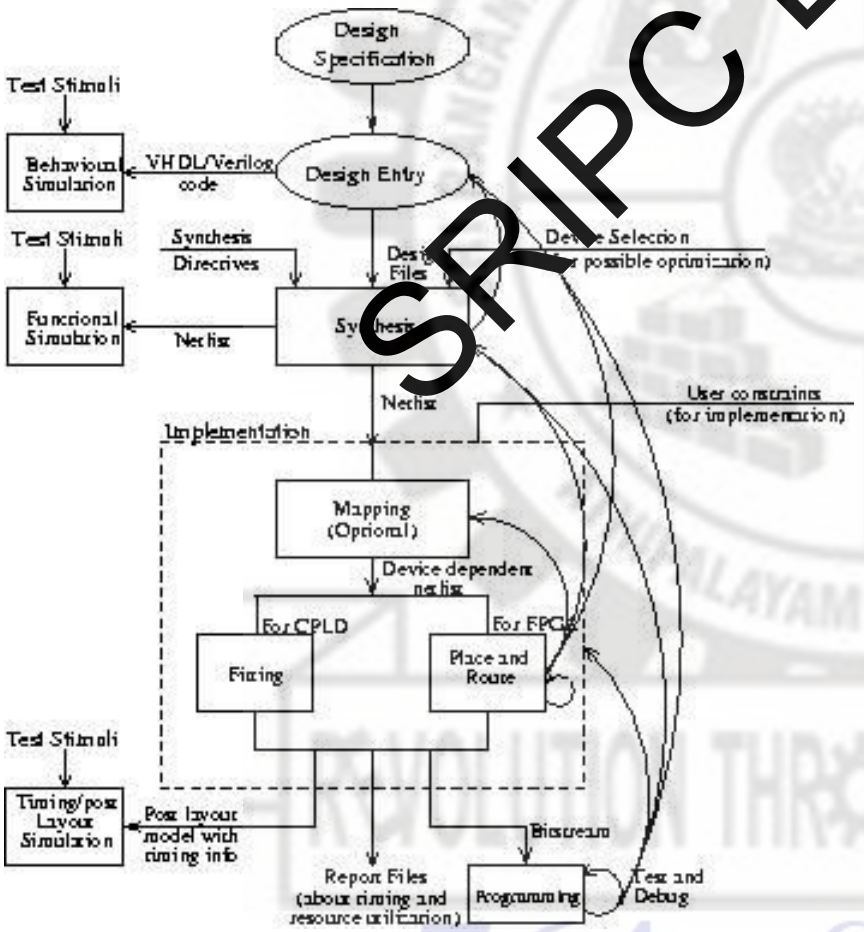




FPGA Implementation of Modulo-4 Counter



Design Steps Involved in Designing With FPGAs



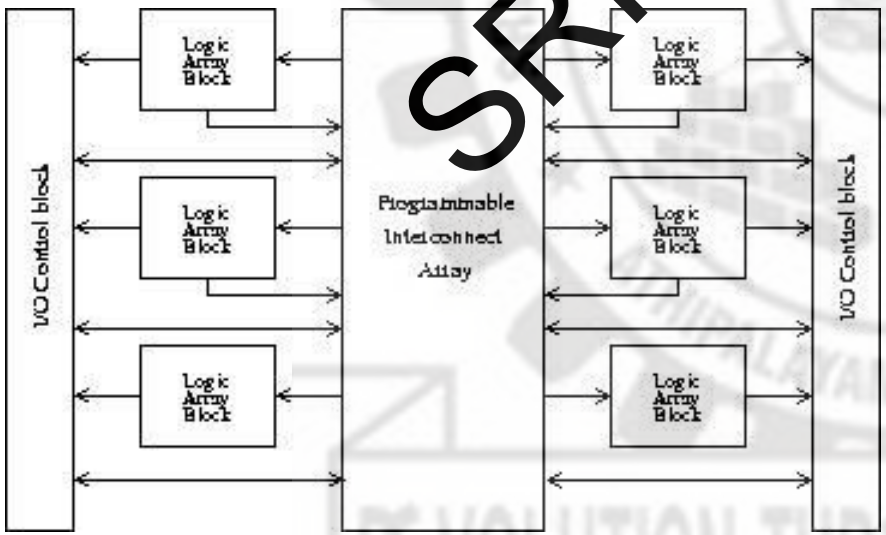
- Understand and define design requirements
- Design description
- Behavioural simulation (Source code interpretation)
- Synthesis
- Functional or Gate level simulation
- Implementation
  - Fitting
  - Place and Route
- Timing or Post layout simulation
- Programming, Test and Debug

Commercially Available Devices

- Architecture differs from vendor to vendor
- Characterized by
  - Structure and content of logic block
  - Structure and content of routing resources
- To examine, look at some of available devices
  - FPGA: Xilinx (XC4000)
  - CPLD: Altera (MAX 5K)

ALTERA CPLDS

- Altera generic architecture



- Hierarchical PLD structure
  - First level: LABs (Functional blocks); LAB is similar to SPLDs
  - Second Level: Interconnections among LABs
- LAB consists of
  - Product term array
  - Product term distribution
  - Macro-cells
  - Expander product terms
- Interconnection region: PIA
- EPROM/EEPROM based
- Example: MAX5K, MAX7K

SRAM FPGA -- EEPROM FPGA

- An FPGA is similar to several other types of devices which have been around for quite a while, the difference being that an FPGA is simply much more expandable and versatile. The devices which FPGAs get compared to most often are CPLDs (Complex Programmable Logic Devices), which are similar in function but typically have way less logic gates inside them; Customizable CPU design is much more feasible with an FPGA. Once upon a time, CPLDs also had the distinct advantage of retaining their configuration even

SRAM FPGA -- EEPROM FPGA

- when turned off; When FPGAs first came out, they used simple SRAM to hold their configuration, which of course would be lost when the device lost power. Back then, the FPGA had to be programmed from scratch every time it was turned on, usually from a separate serial ROM chip. But today, FPGAs come in Flash, EPROM, and EEPROM variants, which will retain configuration, and which can also be re-programmed. (Fuse and anti-fuse FPGAs also exist, which act like PROMs in that they are one-time programmable, and cannot be reprogrammed

SRAM FPGA -- EEPROM FPGA

- afterward.) Despite this, however, most FPGAs still use SRAM for reasons of simplicity (when you need to reprogram it, it's easier to re-encode a small ROM chip than to reprogram a large FPGA chip), so count on having to use a separate boot ROM for the FPGA.
- Use of an FPGA is broadly divided into two main stages: The first is "configuration mode", the mode in which the FPGA is when you first power it up. Configuration mode is, as you may have guessed, where you configure the FPGA; That is,

SRAM FPGA -- EEPROM FPGA

- this is when you load your code into it, dictating how the pins behave. Once configuration is complete, the FPGA goes into "user mode", its main mode of operation, where the programmed circuit actually starts functioning.

Product – FPGA vs ASIC

Comparison:

- FPGA benefits vs ASICs:
  - Design time: 9 month design cycle vs 2-3 years
  - Cost: No $3-5 M upfront (NRE) design cost.
    No $100-500K mask-set cost
  - Volume: High initial ASIC cost recovered only in very high volume products

- Due to Moore's law, many ASIC market requirements now met by FPGAs
  - Eg. Virtex II Pro has 4 processors, 10 Mb memory, IO

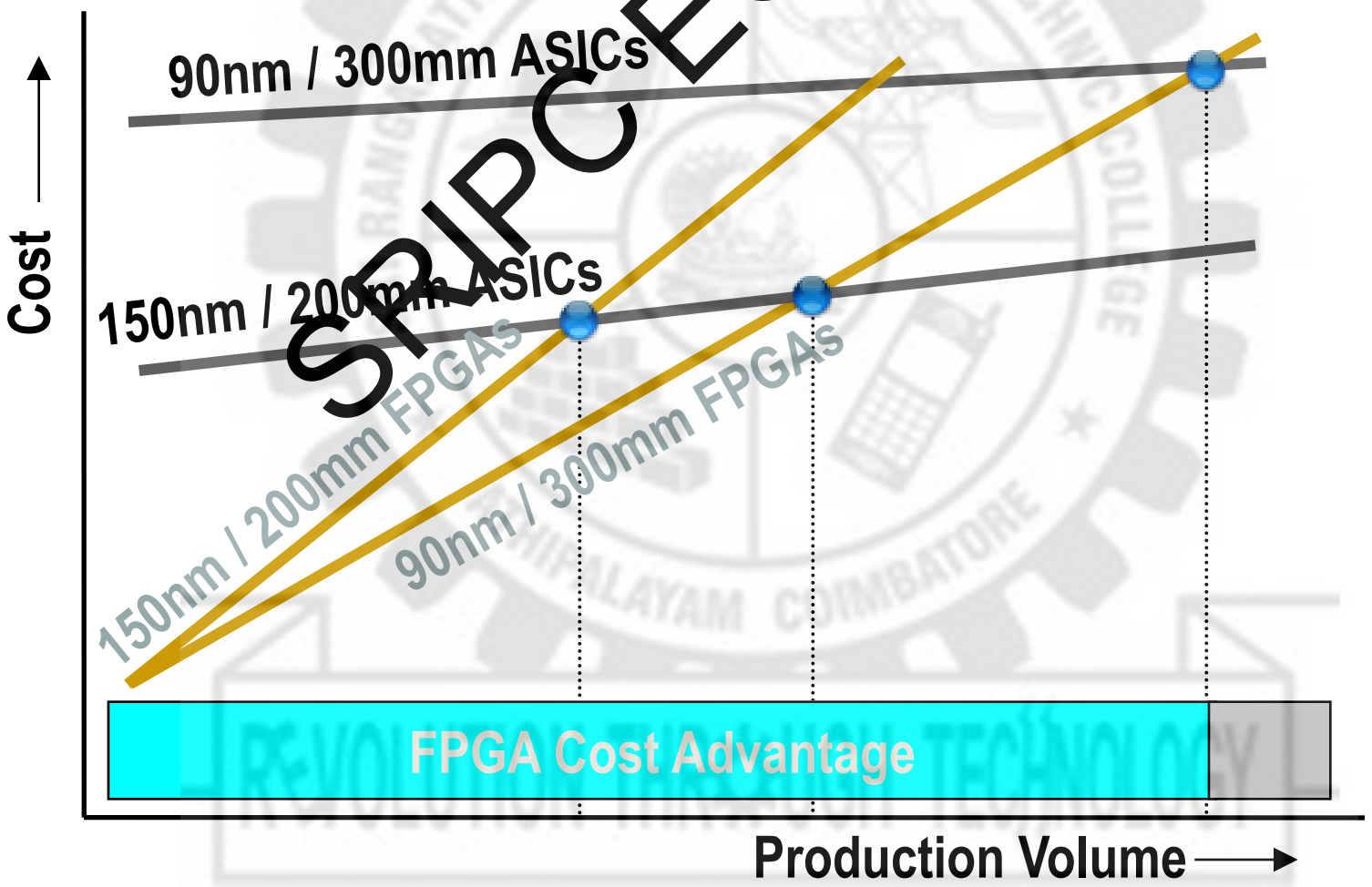
Resulting Market Shift:

- Dramatic decline in number of ASIC design starts:
  - 11,000 in '97
  - 1,500 in '02

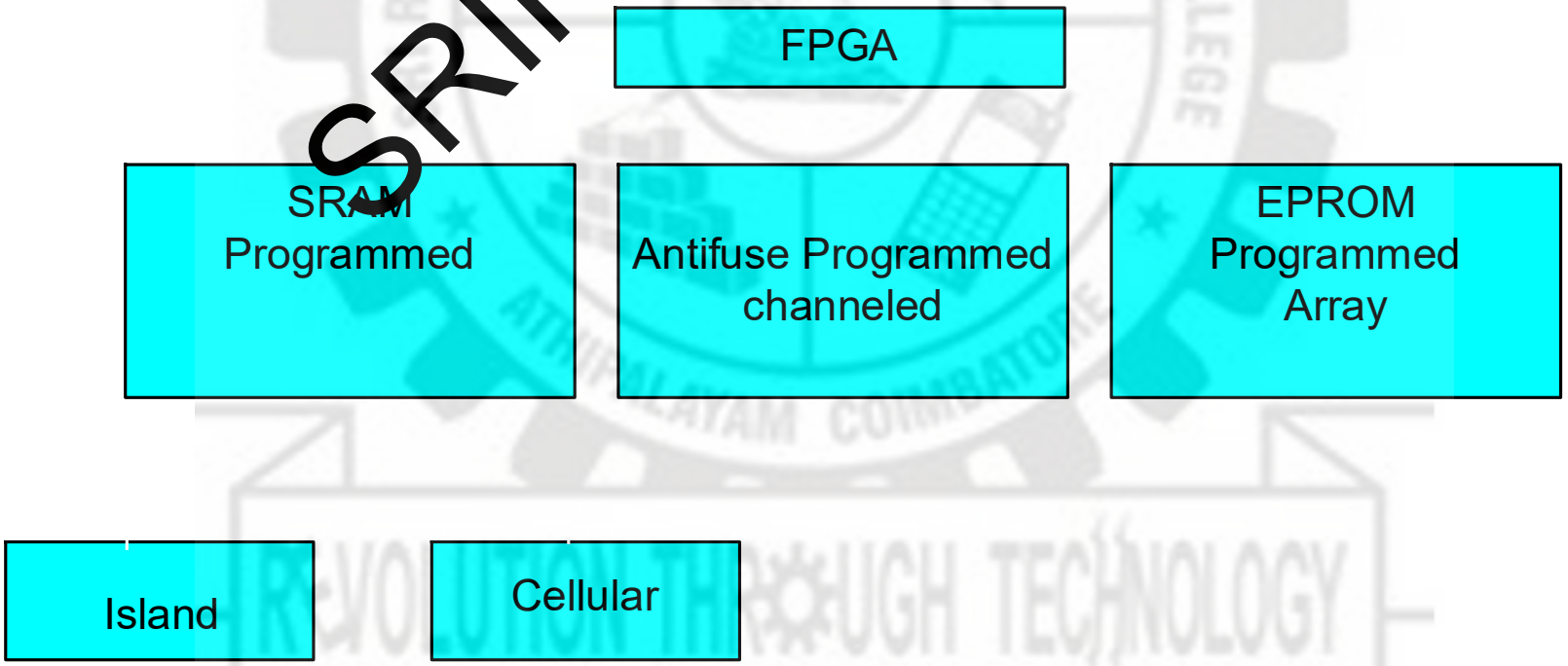
- FPGAs as a % of Logic market:
  - Increase from 10 to 22% in past 3-4 years

- FPGAs (or programmable logic) is the fastest growing segment of the semiconductor industry!!

FPGA/ASIC Crossover Changes



Taxonomy of FPGAs



ASICS

- An ASIC (application-specific integrated circuit) is a microchip designed for a special application, such as a particular kind of transmission protocol or a hand-held computer. You might contrast it with general integrated circuits, such as the microprocessor and the random access memory chips in your PC. ASICs are used in a wide-range of applications, including auto emission control, environmental monitoring, and personal digital assistants (PDAs).