Lecture 18
EE 421 / CS 425
Digital System Design

Spring 2023

Shahid Masud

Topics

- Floating Point Adder and Multiplier Circuits

- Introduction to Programmable Logic Hardware Device

- Nomenclature and types

- Logic Array Circuits

- PLA and PAL

- Programmable Logic Devices – PLD

- A Programmable I/O Block
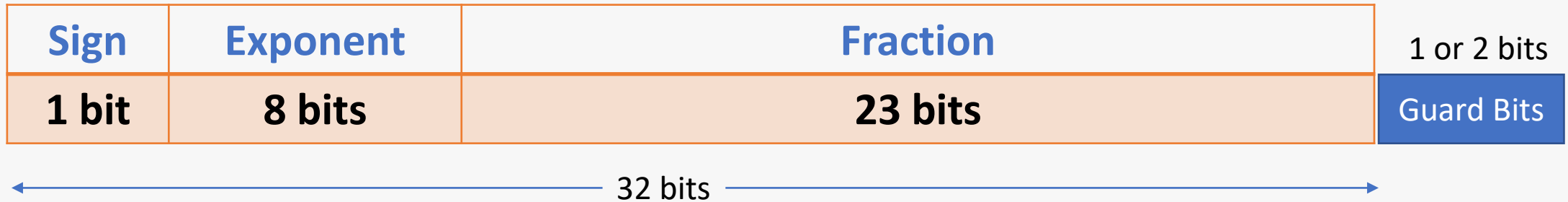
LUMS

Floating Point Arithmetic – Digital Design

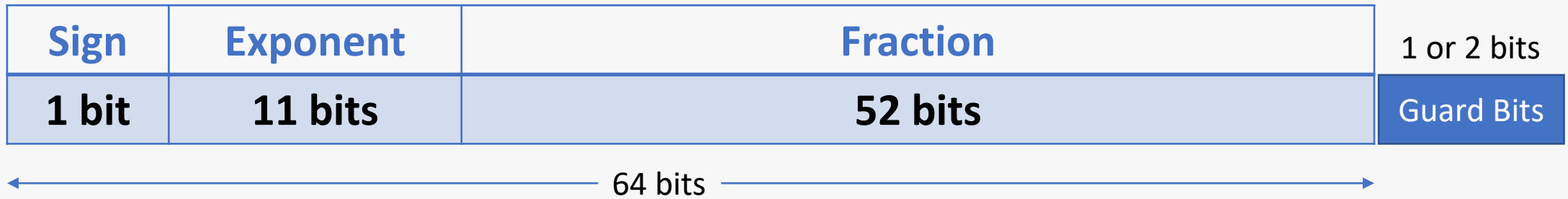$$N = (-1)^S \times (1+F) \times 2^E$$

E.g. $91.820734 \times 10^{-34}$

❖ A signed-magnitude system for the fractional part and a biased notation for the exponent
❖ Three subfields
  ❖ Sign S
  ❖ Fraction F (or Significand or Mantissa)
  ❖ Exponent E
❖ Sign bit is 0 for positive numbers, 1 for negative numbers
❖ Fractions always start from **1**.xxxx, hence the integer **1** is not written (register has xxxx)
❖ Exponent is biased by +127 (add 127 to whatever is in register bits)
❖ Normalize: Express numbers is the standard format by shifting of bits and adding / subtracting from Exponent register

LUMS

IEEE 754 Floating Point Representation

Single Precision IEEE 754

Sign	Exponent	Fraction	
1 bit	8 bits	23 bits	Guard Bits

1 or 2 bits

←————————————— 32 bits —————————————→

Double Precision IEEE 754

Sign	Exponent	Fraction	
1 bit	11 bits	52 bits	Guard Bits

1 or 2 bits

←————————————— 64 bits —————————————→

Examples of Floating Point Representation

$-(13.45)_{10}$

$= (1101.01\ 1100\ 1100\ 1100\ \ldots\ldots\ldots)^2$; this is un-normalized

$= (1.10101\ 1100\ 1100\ 1100\ 1100\ 1) \times 2^3$; normalized

Fraction part is 10101 1100 1100 1100 1

Biased Exponent is 3+127 = 130

Sign = 1

5.0345

$= 101\ .\ 0000\ 1000\ 1101\ 0100\ 1111\ 110$; this is un-normalized

$= 1.\ 01\ 0000\ 1000\ 1101\ 0100\ 1111\ 110 \times 2^2$; normalized

Biased Exponent = 2+127 = 129 = $(1000\ 0001)_2$

Fraction = 01 0000 1000 1101 0100 1111 110

Sign = 0

LUMS

Floating Point Multiplication

Consider two floating point numbers:
$(F_1 \times 2^{E1})$ and $(F_2 \times 2^{E2})$

The product of these two numbers is:
$= (F_1 \times 2^{E1}) \times (F_2 \times 2^{E2})$
$= (F_1 \times F_2) \times 2^{(E1+E2)}$
$= F \times 2^{E}$

Sign of result depends on Sign of the two numbers

LUMS

Floating Point Multiplication Steps

1. Normalize the two numbers if not done already

2. The exponents of the Multiplier (E1) and the multiplicand (E2) bits are added and the base value is subtracted from the added result. The subtracted result is put in the exponential field of the result block → E1+E2-bias

3. Multiply the two fractions (or significands)

4. S1, the signed bit of the multiplicand is XOR'd with the multiplier signed bit of S2. The result is put into the resultant sign bit.

5. The mantissa of the Multiplier (M1) and multiplicand (M2) are multiplied and the result is placed in the resultant field of the mantissa (truncate/round the result for 24 bits) → M1 * M2

6. If the product is 0, adjust the proper representation of answer to 0

7. If the product fraction is too big, normalize by shifting it right and incrementing the exponent

8. If the product fraction is too small, normalize by shifting left and decrementing the exponent

9. Round to appropriate number of bits. If rounding results in loss of normalization, then first normalize and then do the rounding

10. If an exponent underflow (below -127) or overflow (above +127) occurs then generate an error condition

LUMS

Bit-widths required

- Multiply the mantissa values including the hidden bits in rounding register. The resultant product of the 24 bits mantissas (M1 and M2) is 48 bits (2 bits are to the left of binary point)

- 8 bit adder (subtractor) needed for Exponent

LUMS

Floating Point Multiplier Block Diagram - Example

Definitions:

St: Start of Floating Point multiplication operation

Mdone: Fraction multiplication is complete

FZ: Fraction is Zero

FV: Fraction overflow (too small or too big)

Fnorm: F is normalized

EV: Exponent overflow

Load: Load F1, E1, F2, E2 into respective registers

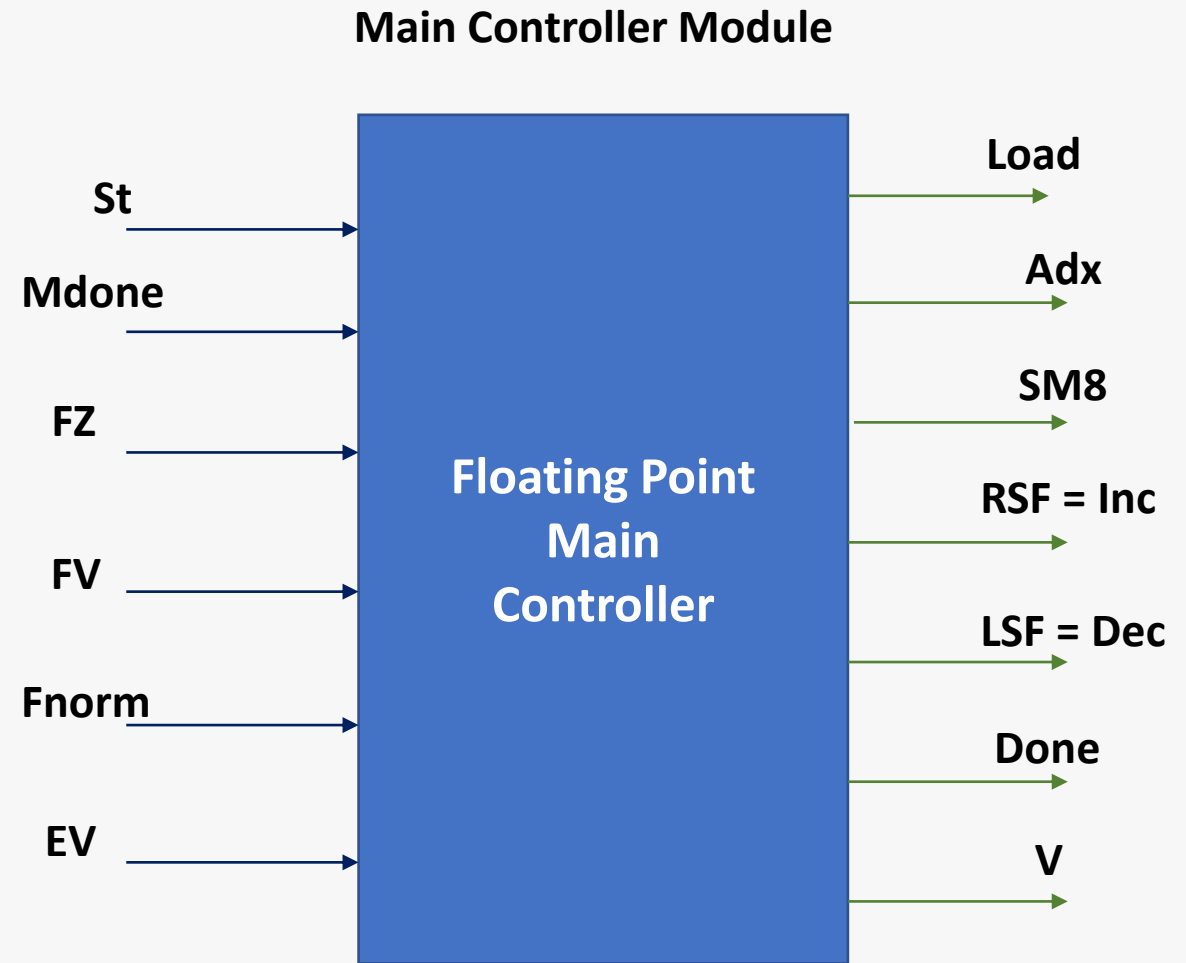**Adx:** Add Exponent; also start of multiplication

SM8: Set exponent to -8 to handle special case of 0
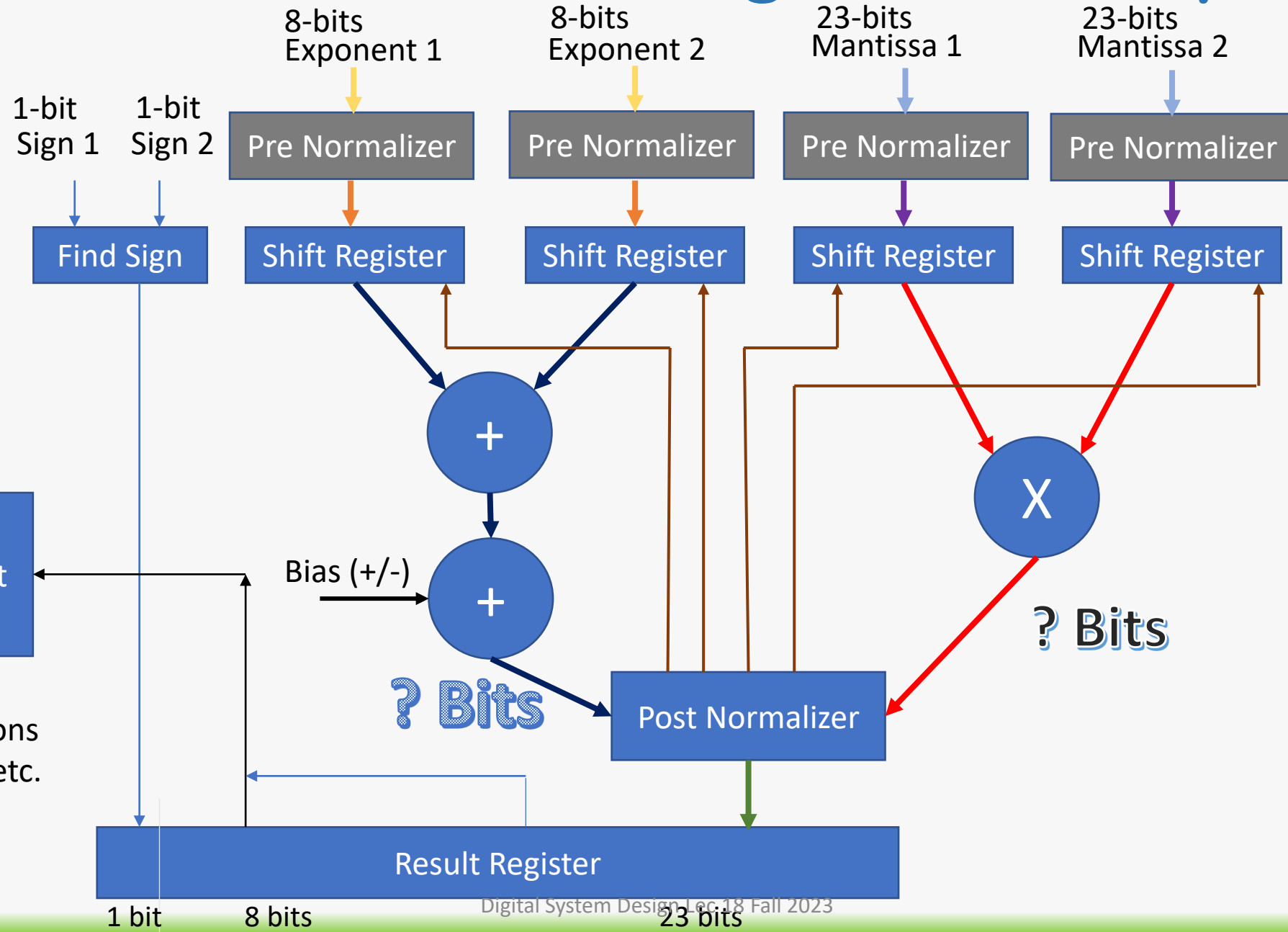
RSF: Shift fraction right; also increment E

LSF: Shift fraction left; also decrement E

V: overflow indicator

Done: Floating Point multiplication is complete

Main Controller Module

Floating Point Main Controller

St, Mdone, FZ, FV, Fnorm, EV → Load, Adx, SM8, RSF = Inc, LSF = Dec, Done, V

LUMS

Data Path of Floating Point Multiplier

Operation Details of Datapath

Pre Normalize Block for Adder and Subtractor : Calculate the difference between the smaller and larger exponent. Adjust the smaller fraction by right shifting it, determine if the operation is an add or subtract after resolving the sign bits. Check for NaNs on inputs.

Pre Normalize Block for Multiplication: Computes the sum/difference of exponents, checks for exponent overflow, underflow condition and INF value on an input.

Post Normalize and Round Unit - Normalize fraction and exponent. Also do all the roundings in parallel and then pick the output corresponding to the chosen rounding mode.

Exceptions Unit – This unit Generates the exception signals like sNAN, qNAN, Inf and Ine The IEEE standard defines two classes of NaNs(non numbers): i. quiet NaNs (qNaNs) : A qNaN is a NaN with the most significant fraction bit set. ii. signaling NaNs (sNaNs): A sNaN is a NaN with the most significant fraction bit clear.

The single precision floating point format is divided into three main parts corresponding to the sign , exponent and mantissa.

Mutiplication of the two operands is done in three parts and thereby obtaining the Product.

The first part of the product which is the sign is determined by an exclusive OR function of the two input signs.

The exponent of the product which is the second part is calculated by adding the two input exponents. The third part which is the significand of the product is determined by multiplying the two input significands each with a '1'concatenated to it.

11

A multiplication of two floating-point numbers is done in four steps: • non-signed multiplication of mantissas: it must take account of the integer part, implicit in normalization. The number of bits of the result is twice the size of the operands (48 bits). • normalization of the result, the exponent can be modified accordingly . • addition of the exponents, taking into account the bias. • calculation of the sign.

Programmable Logic
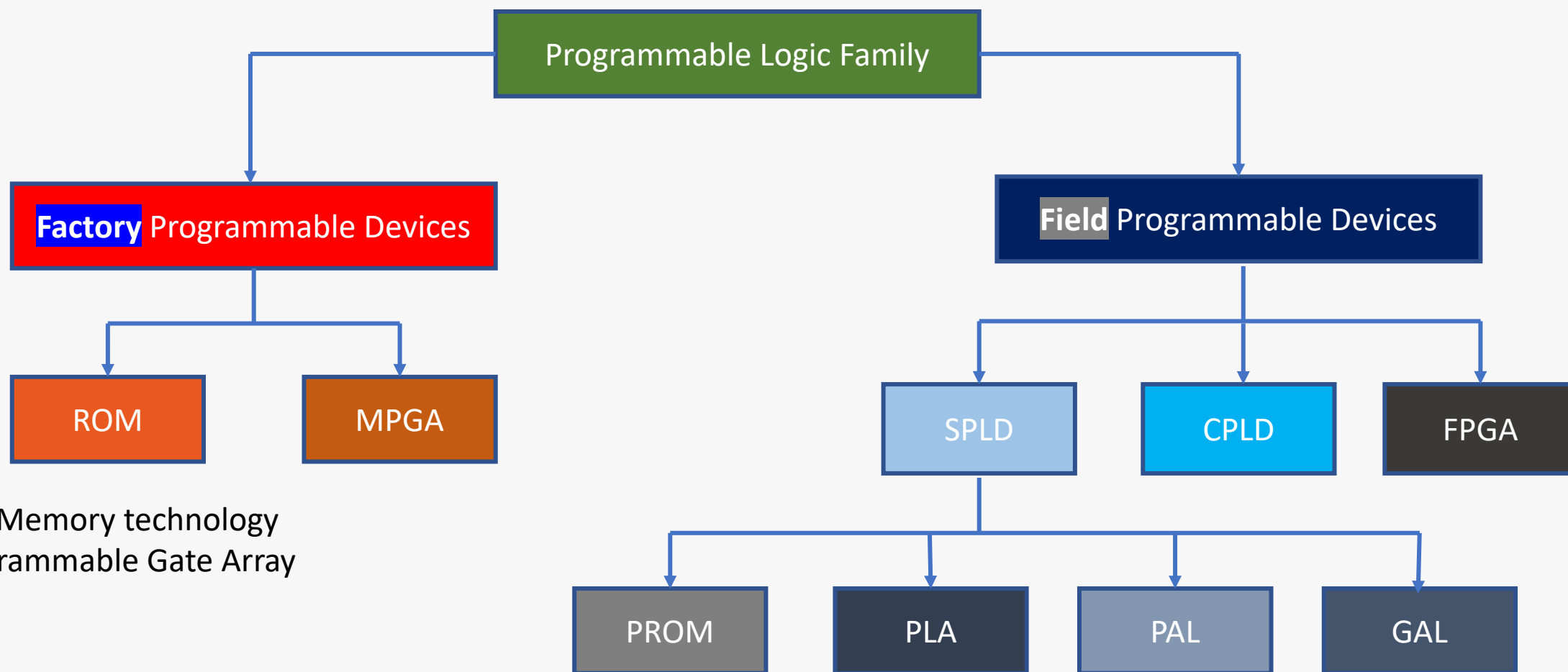# Reconfigurable Logic

LUMS

Definition of Programmable Logic

- A family of semiconductor integrated circuits that can be reconfigured through:
  - special hardware-description-languages, and
  - associate Electronic Design Automation tools
  - Special configuration equipment

to produce a variety of hardware functionality from the same integrated circuit chip.

- The functionality in hardware can be erased and reconfigured in a manner analogous to design of a software system using microprocessor.

- All other types of semiconductors have fixed and pre-defined functionality.

- Performance vs Flexibility tradeoffs exist, as with any other digital system.

LUMS

Programmable Logic Broad Classification

Programmable Logic Family

Factory Programmable Devices

Field Programmable Devices

ROM

MPGA

SPLD

CPLD

FPGA

Read only Memory technology
Mask Programmable Gate Array

PROM

PLA
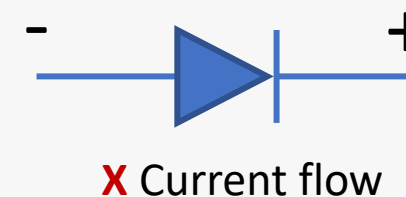
PAL

GAL

Simple and Complex Programmable Logic Devices
Field Programmable Gate Array
Generic Array Logic (complex PAL by lattice semi), Programmable Logic Array
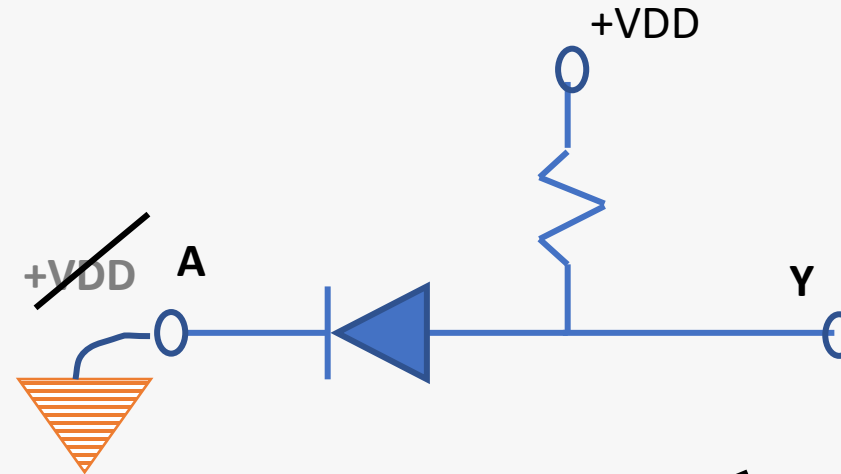
LUMS

Diode – Principle of operation

P | N

+ ____ -

Current flow

Forward Biased

- ____ +

X Current flow
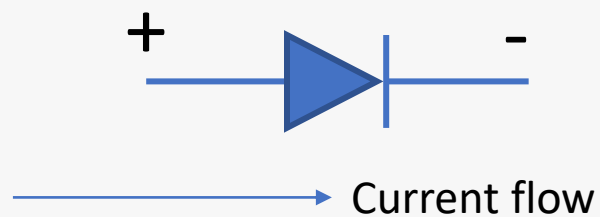
Reverse Biased

Diode in logic gates

+VDD

+VDD   **A**   **Y**
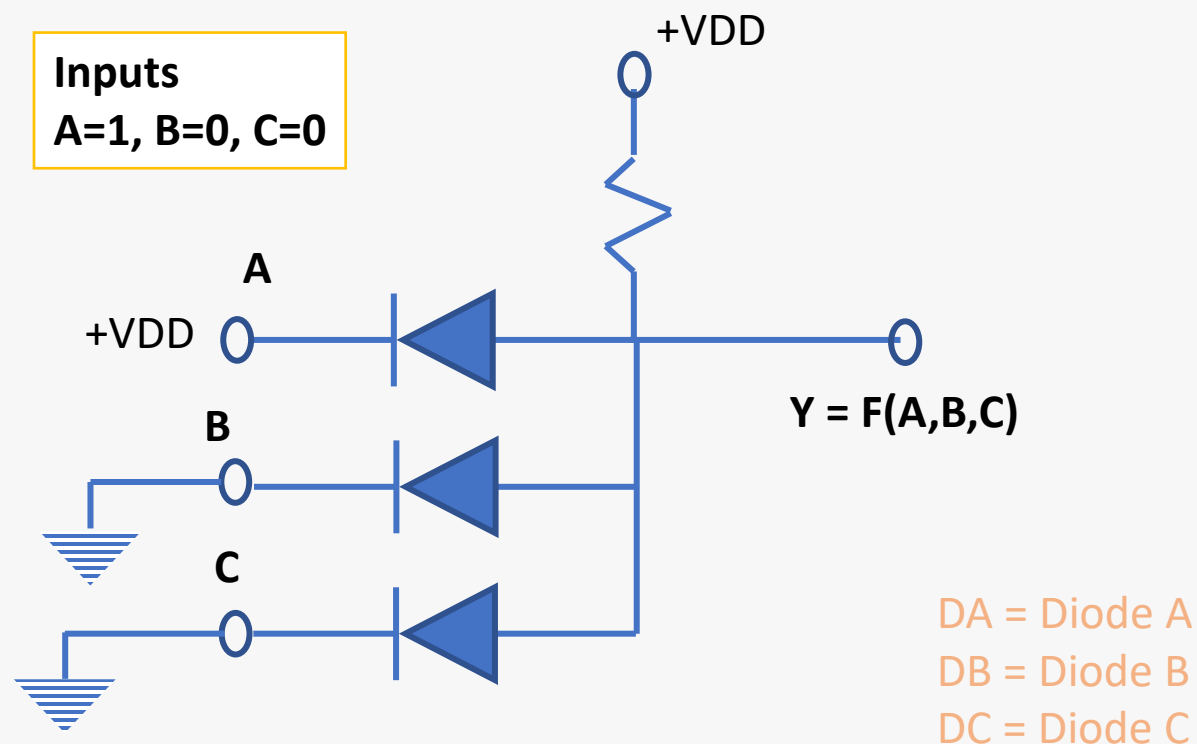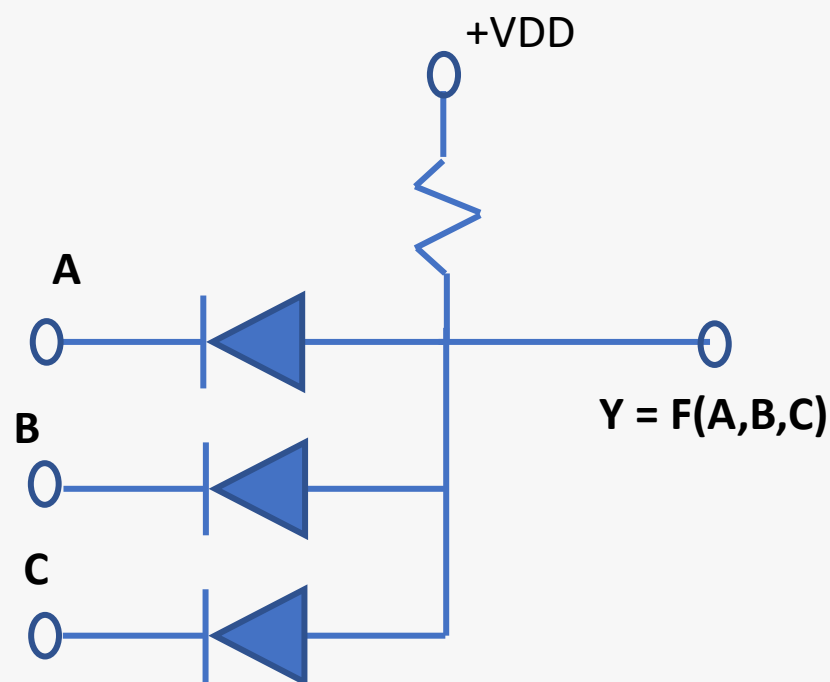
**When A=1, Diode is Reverse Biased
Hence Output Y = 1 through +VDD**

**When A=0, Diode is Forward Biased
Hence Output Y = 0 through A = 0 Volt**
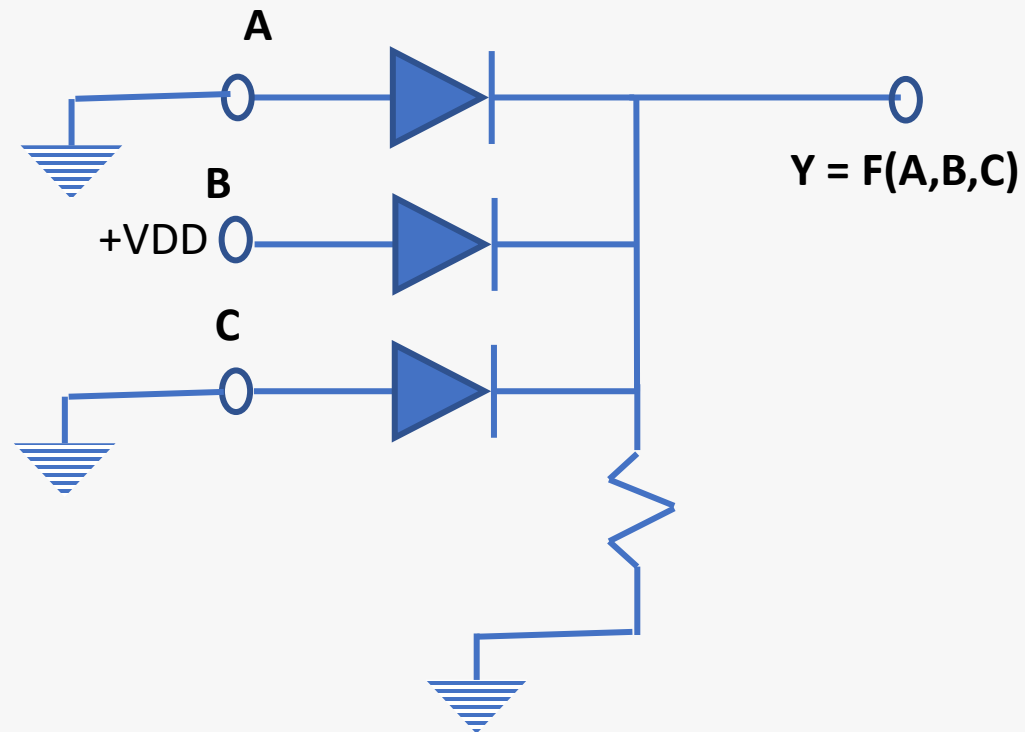
LUMS

Diode based AND Logic Array

+VDD

A

B

C

Y = F(A,B,C)

Inputs
A=1, B=0, C=0

+VDD

A

+VDD

B

C

Y = F(A,B,C)

DA = Diode A
DB = Diode B
DC = Diode C

DA=Open Circuit, DB = Forward Biased Short, DC = Forward Biased Short
Y= Logic '0' as it is Shorted to Zero due to DB or DC

LUMS

Diode based OR array
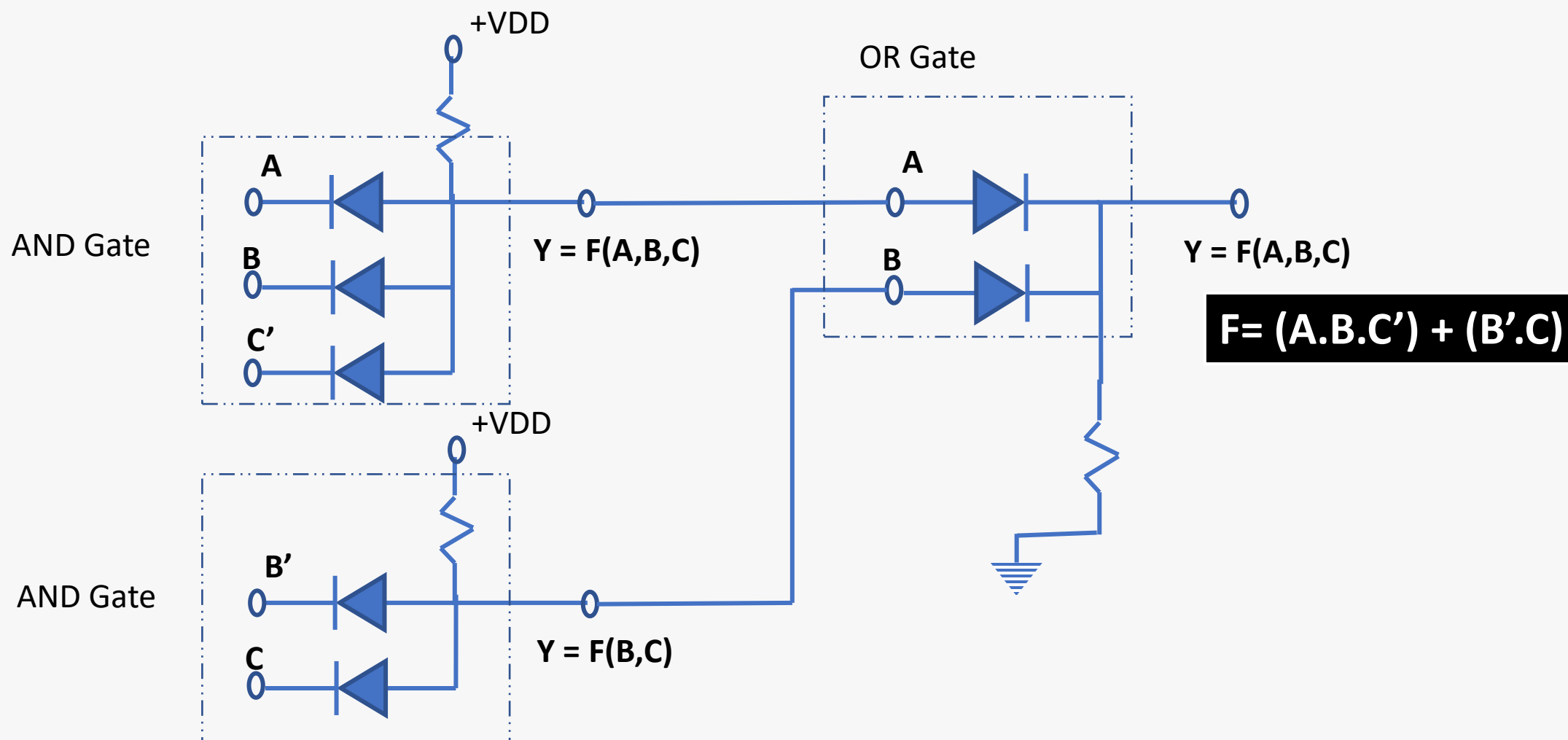
A

B

C

Y = F(A,B,C)

Inputs A = 0, B = 1, C = 0

A

B

+VDD

C

Y = F(A,B,C)
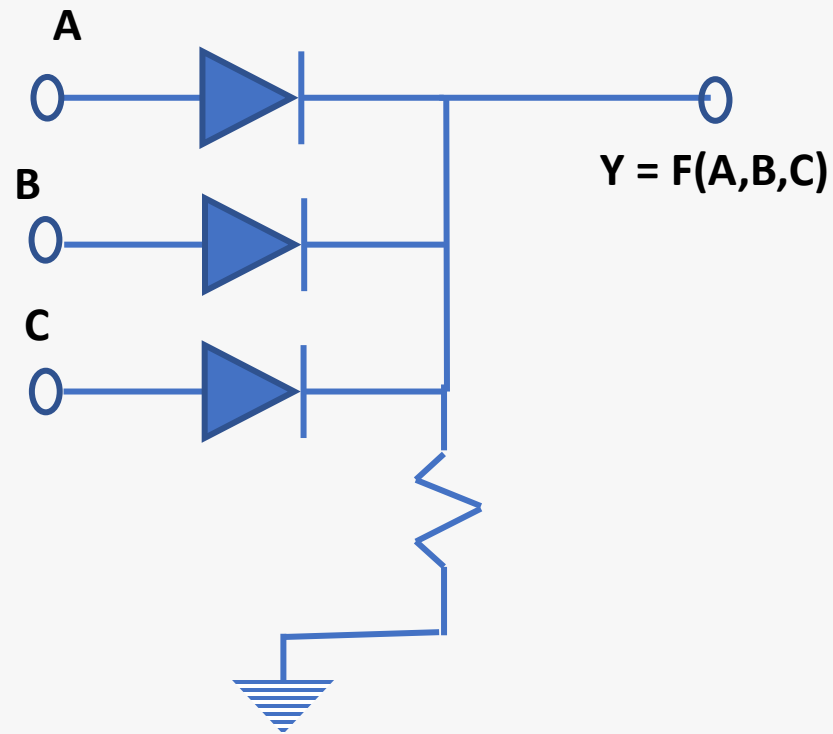
DA and DC = Open Circuit due to Zero input, DB = Short Circuit due to +VDD input
Y = +VDD Logic '1' due to DB being short circuit, other paths are open circuit

Diode based AND-OR Array

+VDD

OR Gate

A

AND Gate

A

B

C'

Y = F(A,B,C)

B

Y = F(A,B,C)

F= (A.B.C') + (B'.C)

+VDD

AND Gate

B'

C

Y = F(B,C)

Diode as fuse

Momentarily apply a higher voltage where open circuit path is required

+20V

-20V

⟷

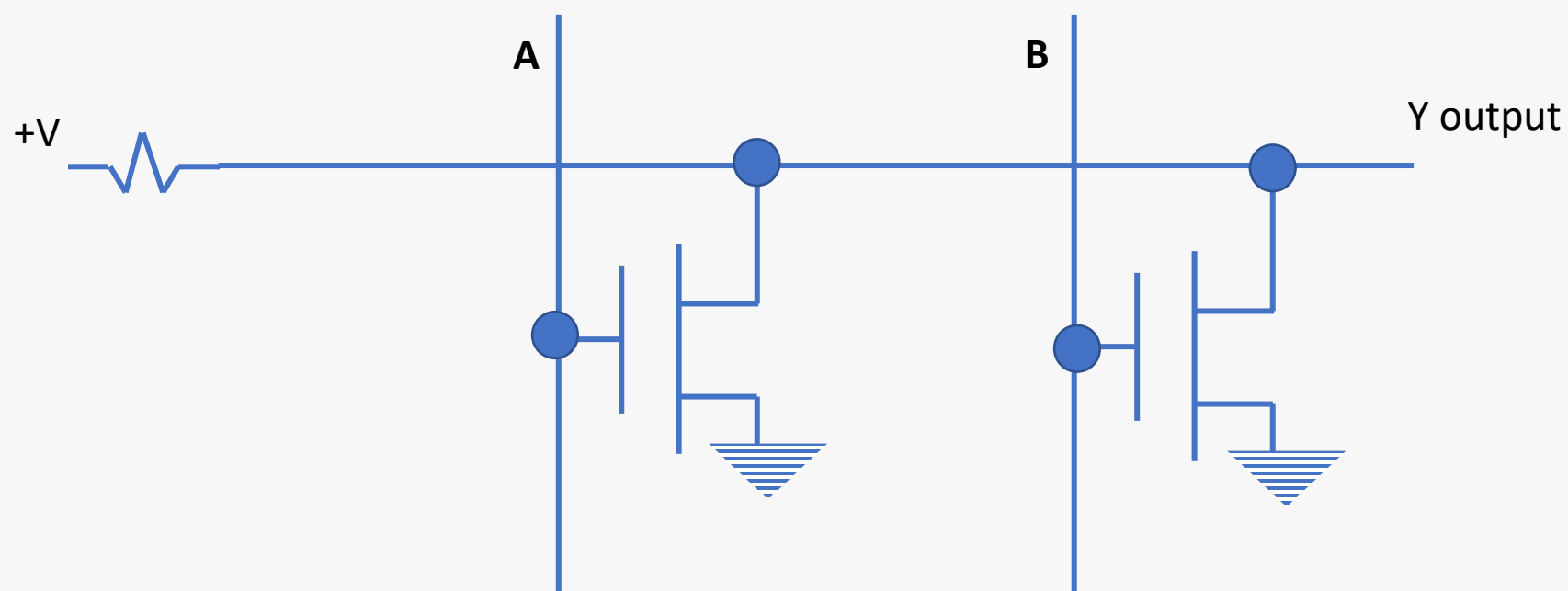**Desired paths from AND gates and OR gates will be retained, others will be blown out**

Desired Paths are selected through a MASK during final stages of FABRICATION PROCESS

LUMS

MOS Transistor as Switch
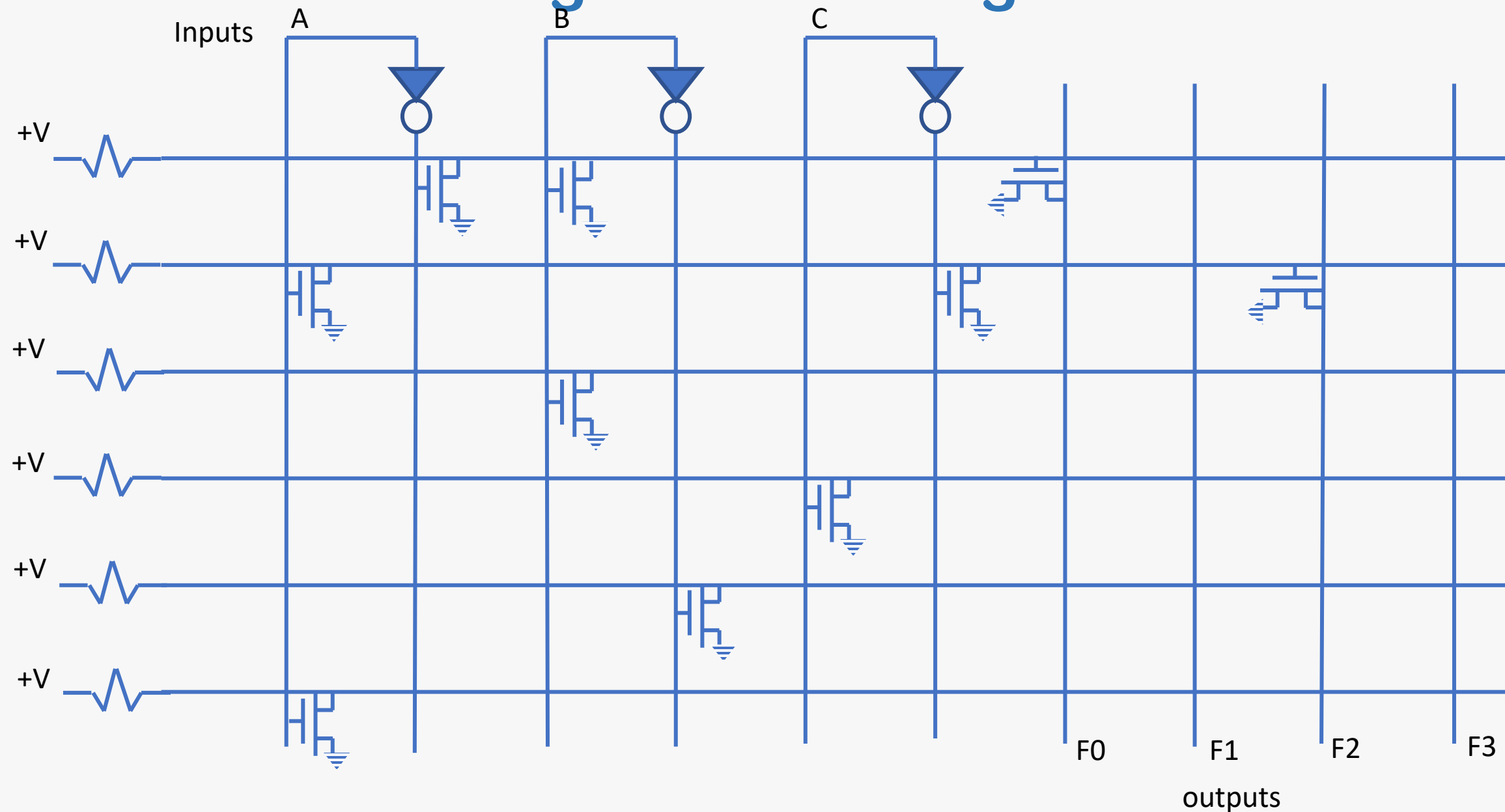
Drain

D

Gate

G

Source

S
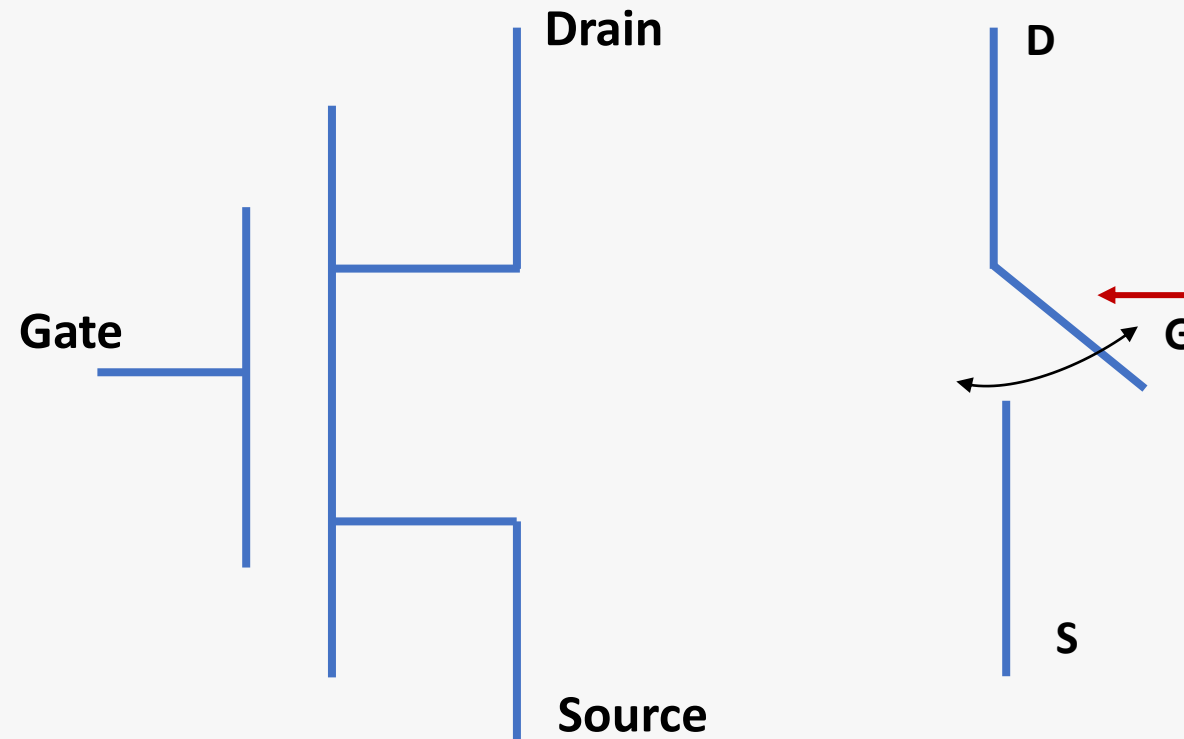
Programming is achieved through controlling GATE Voltage
High GATE Voltage will ensure flow of current, like a short circuit
Low GATE Voltage will ensure no flow of current, like an open circuit

LUMS

Logic Gates Design with Transistor Switches

Transistors as Programmable Logic Switches

Inputs

A   B   C

F0   F1   F2   F3

outputs

Different logic functions are realized through transistor switches

LUMS

AND-OR Logic Array

PLA = Programmable Logic Array Comprises Programmable AND-OR Gates

A

B

C

Y output

LUMS