Lecture 4
EE 421 / CS 425
Digital System Design

Fall 2024

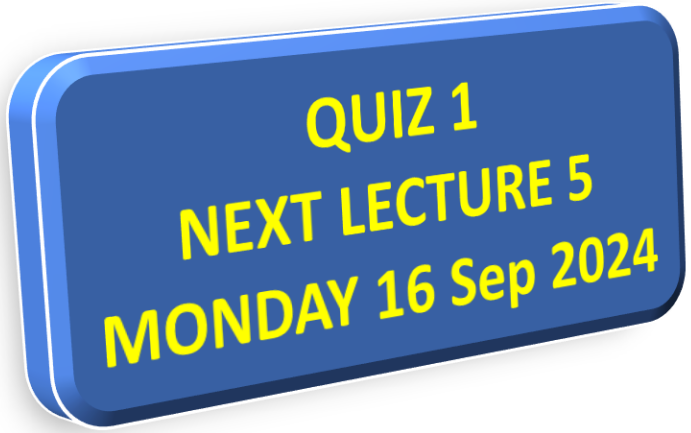
Shahid Masud

Topics

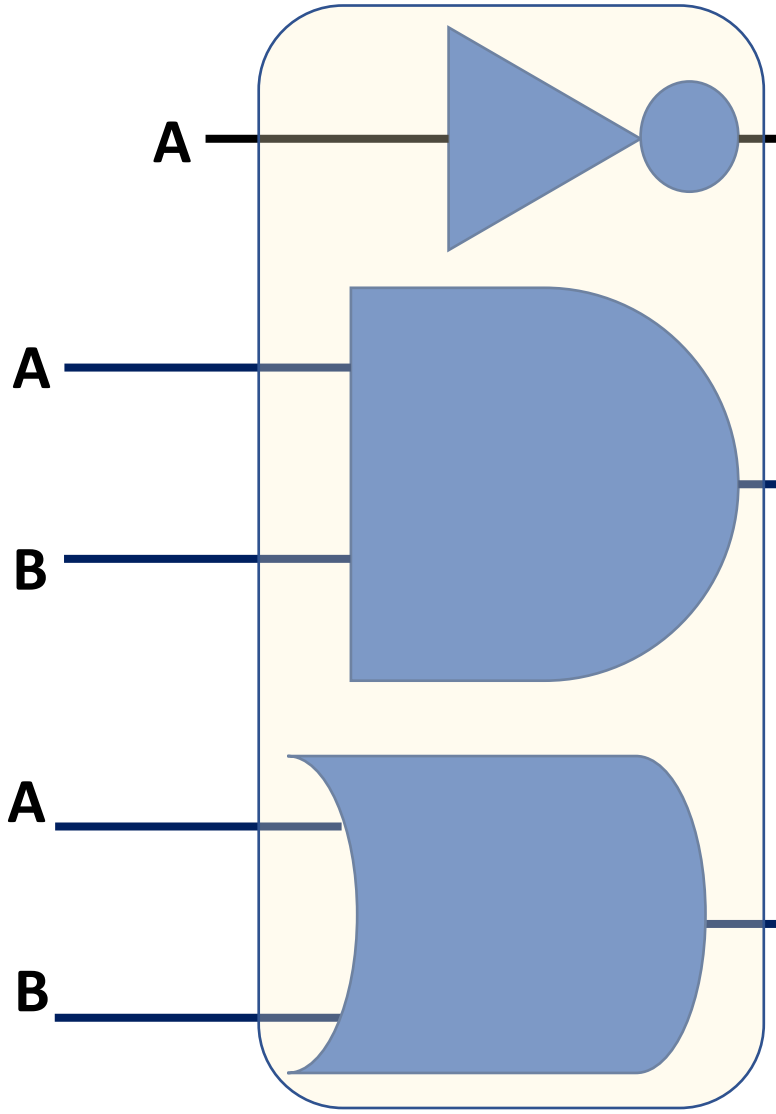
Some examples from WinLogiLab (from last time)

FROM BEHAVIOUR TO IMPLEMENTATION

- Connecting logic expression to physical circuits: Boolean Algebra, K-Maps to logic gates, XOR
- Combinational Logic Implementation using functional mapping to Decoders, Memory and Multiplexers
- Implementation Constraints



Gates (Primary Gates)



PHYSICAL GATES

BEHAVIOURAL MODELS

Inverter

A	F
0	1
1	0

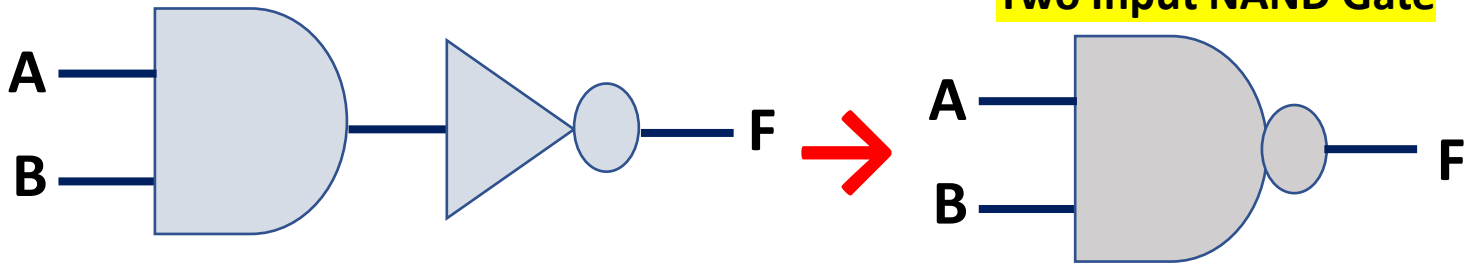
Two Input AND Gate

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

Two Input OR Gate

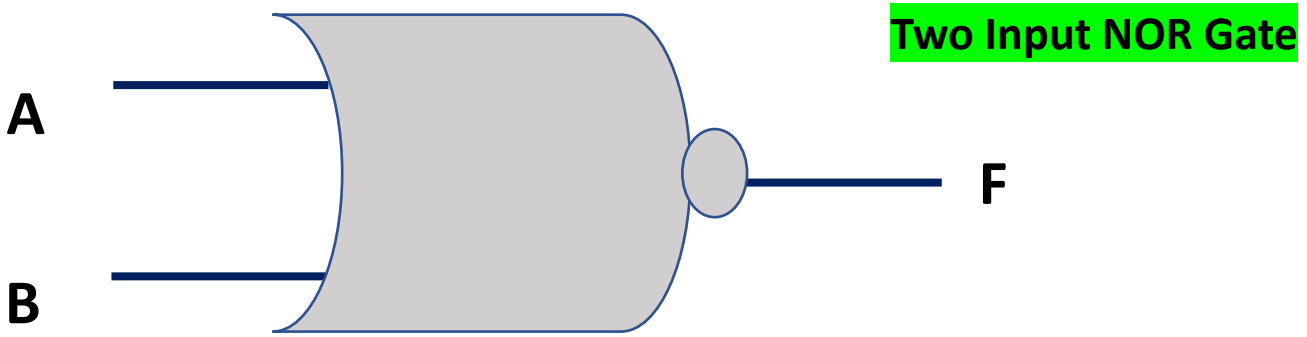
A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

Gates (Compound Gates)



AND Gate followed by Inverter

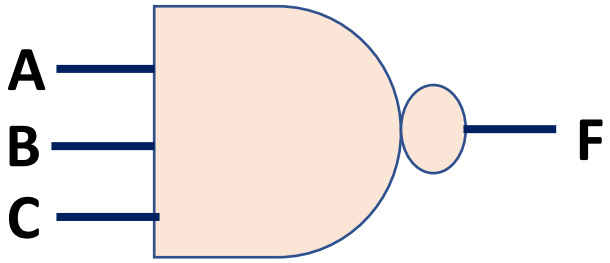
A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

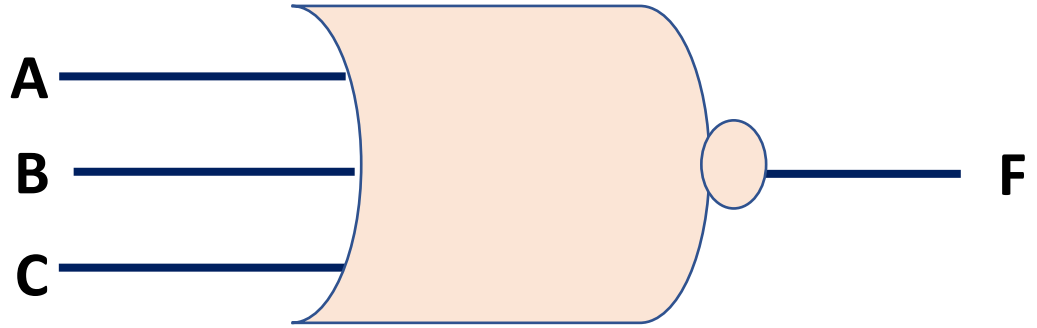


OR Gate followed by Inverter

A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

3 Input Compound Gates





Three Input NAND Gate

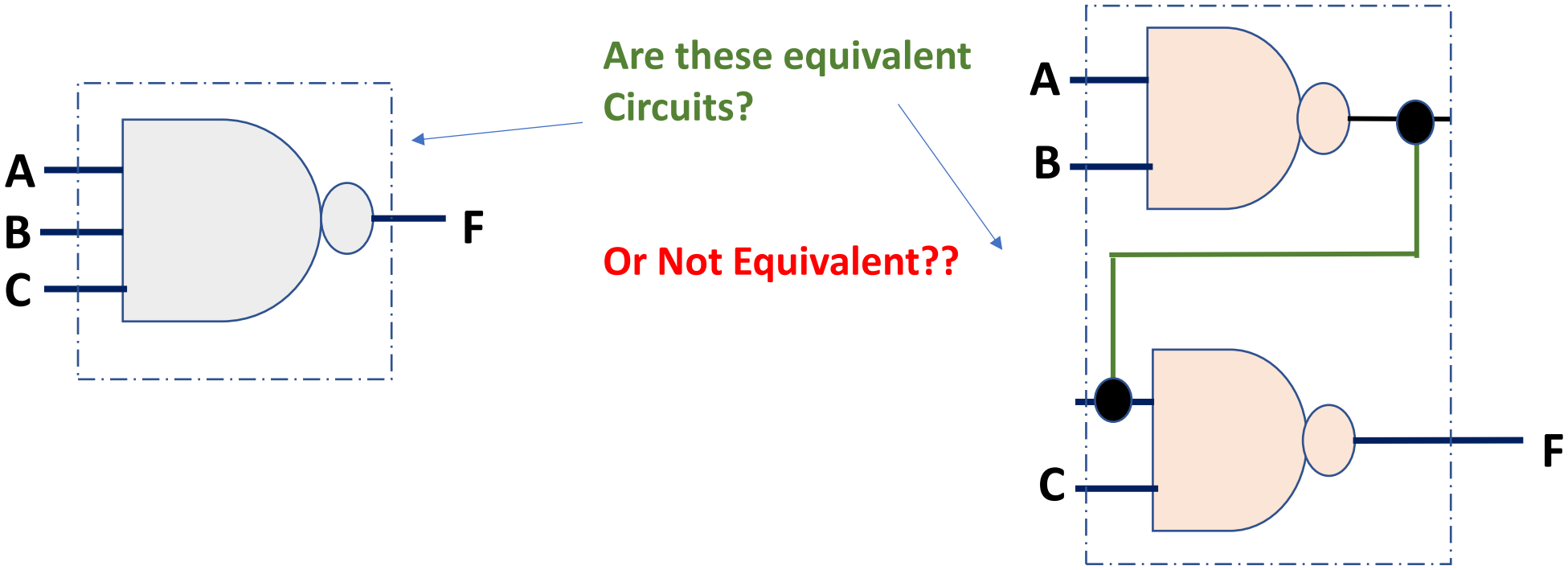
A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Three Input NOR Gate

A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

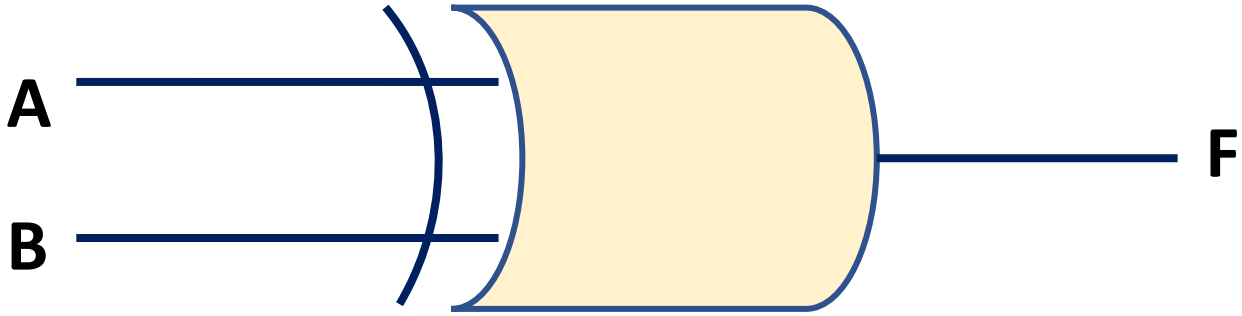
A Question about Compound Gates

Is this 3 input Nand gate same as the configuration on the right?



Complex Gates – the XOR and XNOR

Two Input Exclusive OR Gate



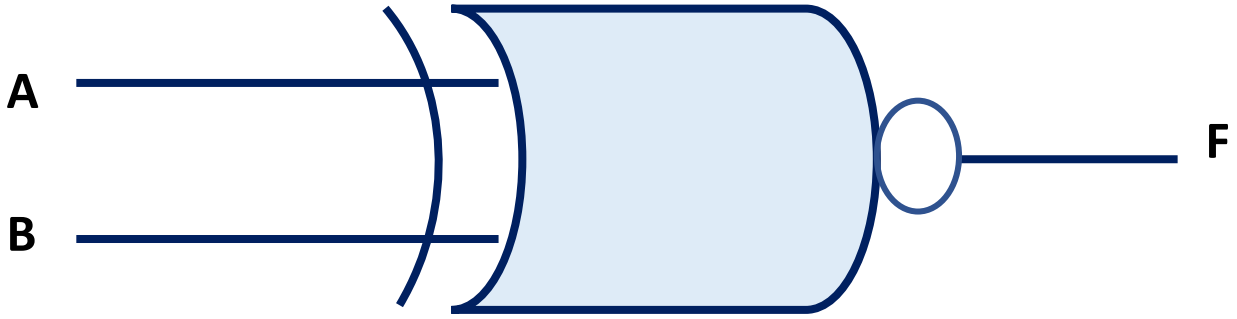
$$F = \bar{A}B + A\bar{B}$$

$$F = A \oplus B$$

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

The XNOR Complex gate

Two Input Exclusive NOR Gate



A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

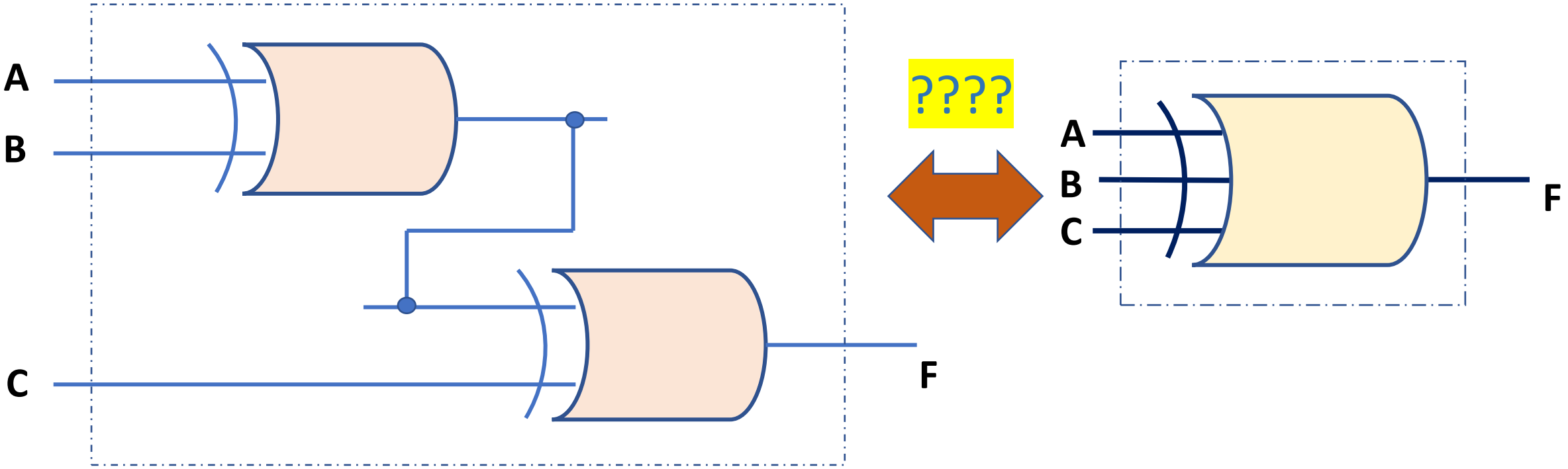
$$F = AB + \overline{A}\,\overline{B}$$

$$F = \overline{A \oplus B}$$

Or $F=(A \oplus B)'$

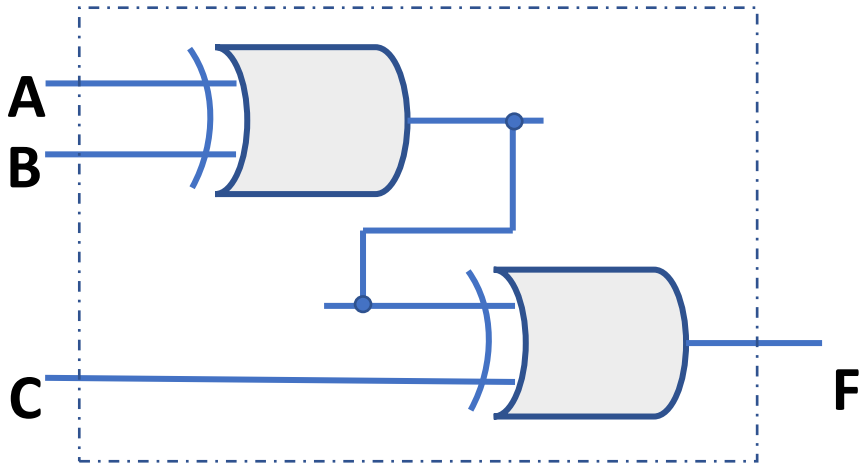
Question?

Are these two gates equivalent, Yes? Or No?



$$F = A \oplus B \oplus C$$

Three Input XOR Gate



$$F = A \oplus B \oplus C$$

A	B	C	$(A \oplus B)$	$(A \oplus B) \oplus (C)$	F
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	1	1	1
0	1	1	1	0	0
1	0	0	1	1	1
1	0	1	1	0	0
1	1	0	0	0	0
1	1	1	0	1	1

$$K\ Map\ of\ (A \oplus B) \oplus (C)$$

This is the right way to work with XOR Gates

C, AB	00	01	11	10
0	0	1	0	1
1	1	0	1	0

(but not the NAND or NOR gates)

Functional Digital Circuits – MSI (Medium Scale Integration)

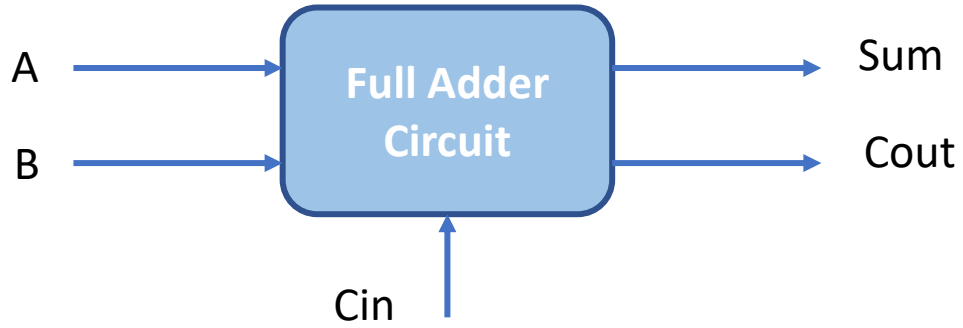
Circuits that are formed from simple, compound or complex gates

Examples:

- **Adder/Subtractor circuits**
- **Decoders**
- **Multiplexers**
- **Parity Checkers**
- **Memory Elements**
- **Counters, Modules**
- **SIPO, PISO, or FIFO, LIFO,– Require REGISTERS**
- **Etc.**

Direct Implementation of Complex Logic Functions

1 Bit Full Adder Circuit



Truth table represents behaviour of inputs and outputs

A	B	Cin	Sum	Cout
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

K-Map for Sum Function

Cin, AB → / ↓ Cin	00	01	11	10
0	0	1	0	1
1	1	0	1	0

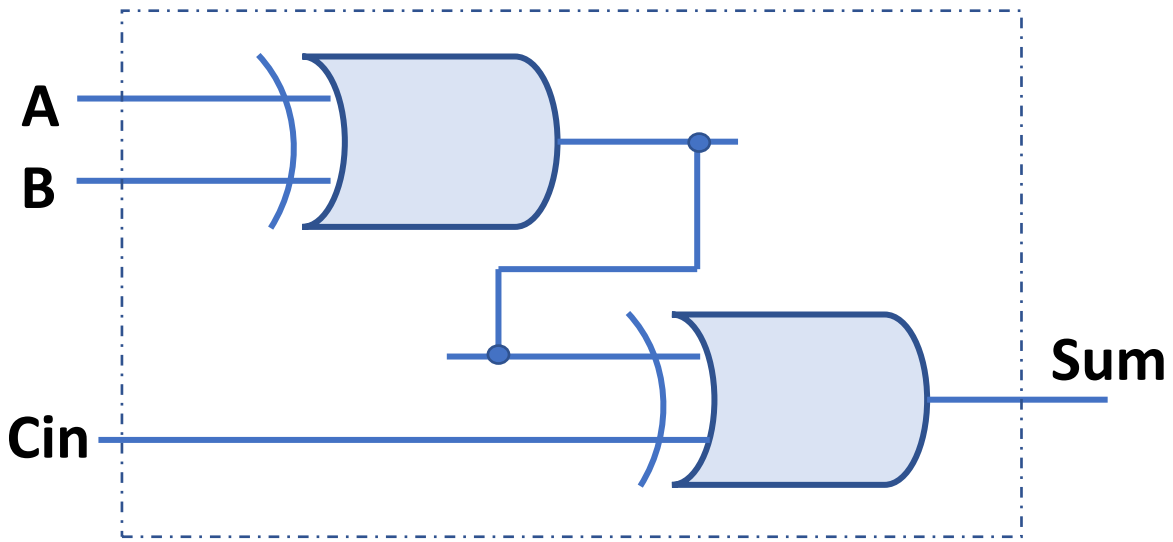
$$Sum = A \oplus B \oplus Cin$$

K-Map for Cout Function

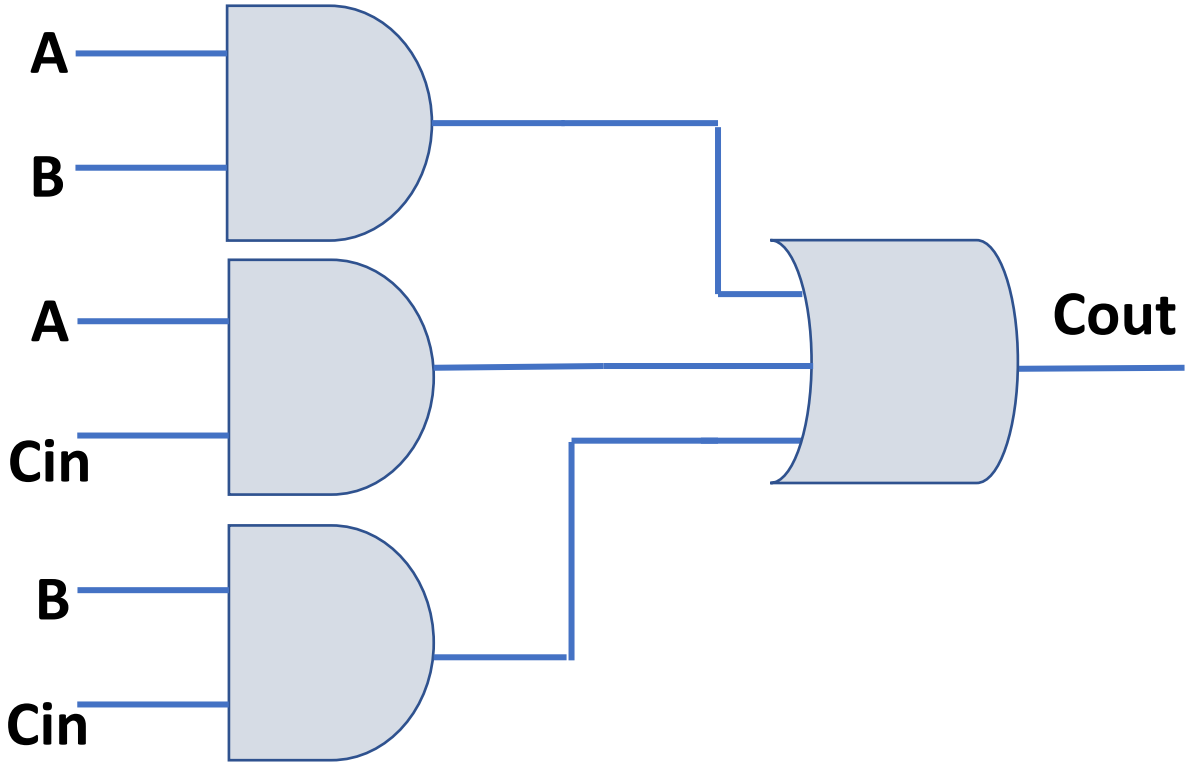
Cin, AB → / ↓ Cin	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$Cout = AB + ACin + BCin$$

Full Adder Implementation using Logic Gates

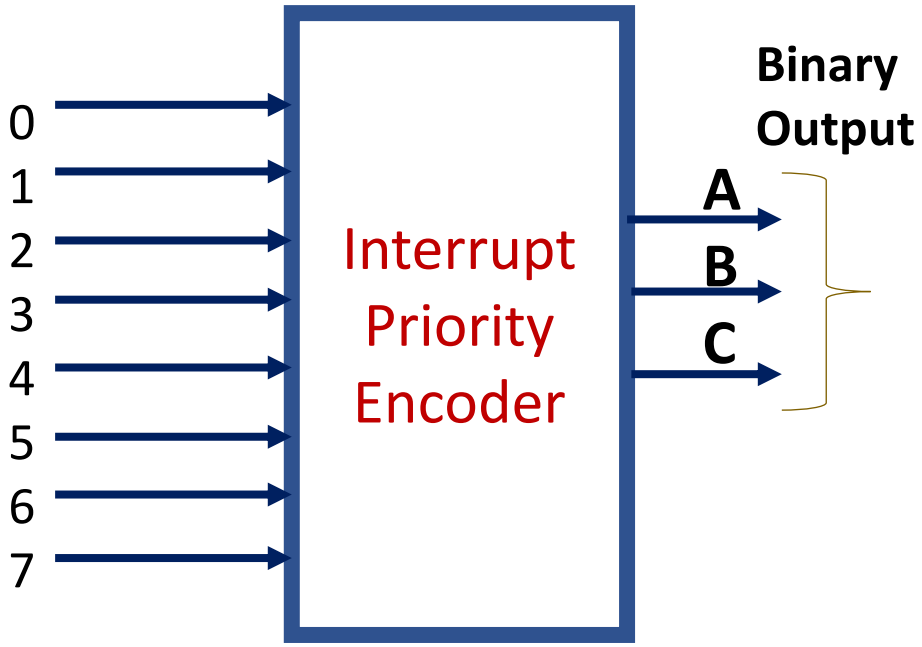


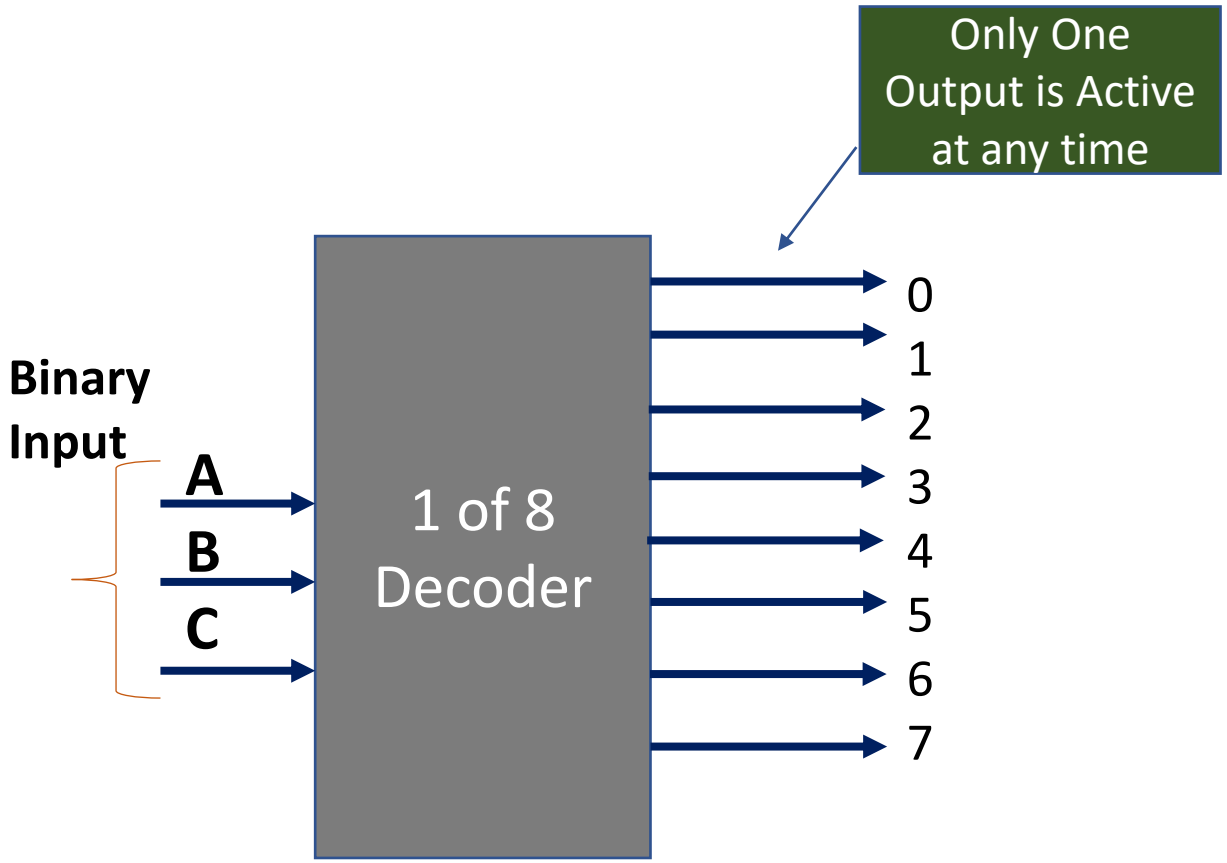
$Sum = A \oplus B \oplus Cin$



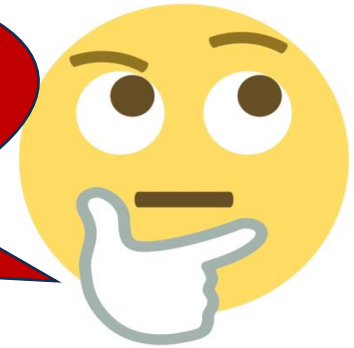
$Cout = AB + ACin + BCin$

Encoders and Decoders Functional Chips

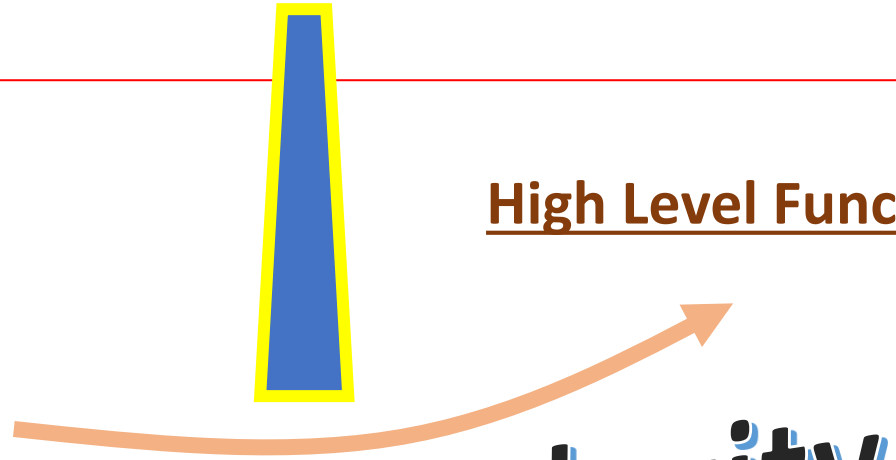




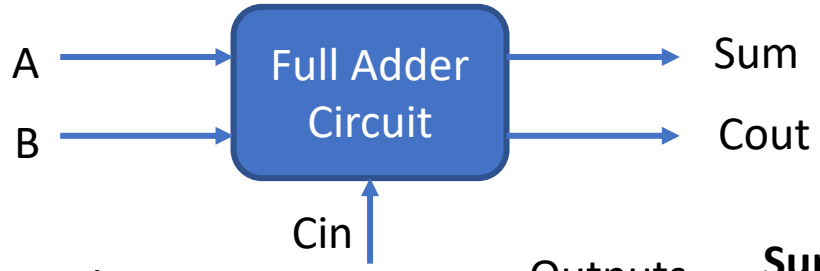
Remember: We are dealing with 'Complex' and 'High Speed Circuits'



Logic Implementation vs Functional Mapping



Full Adder Mapping on Decoder

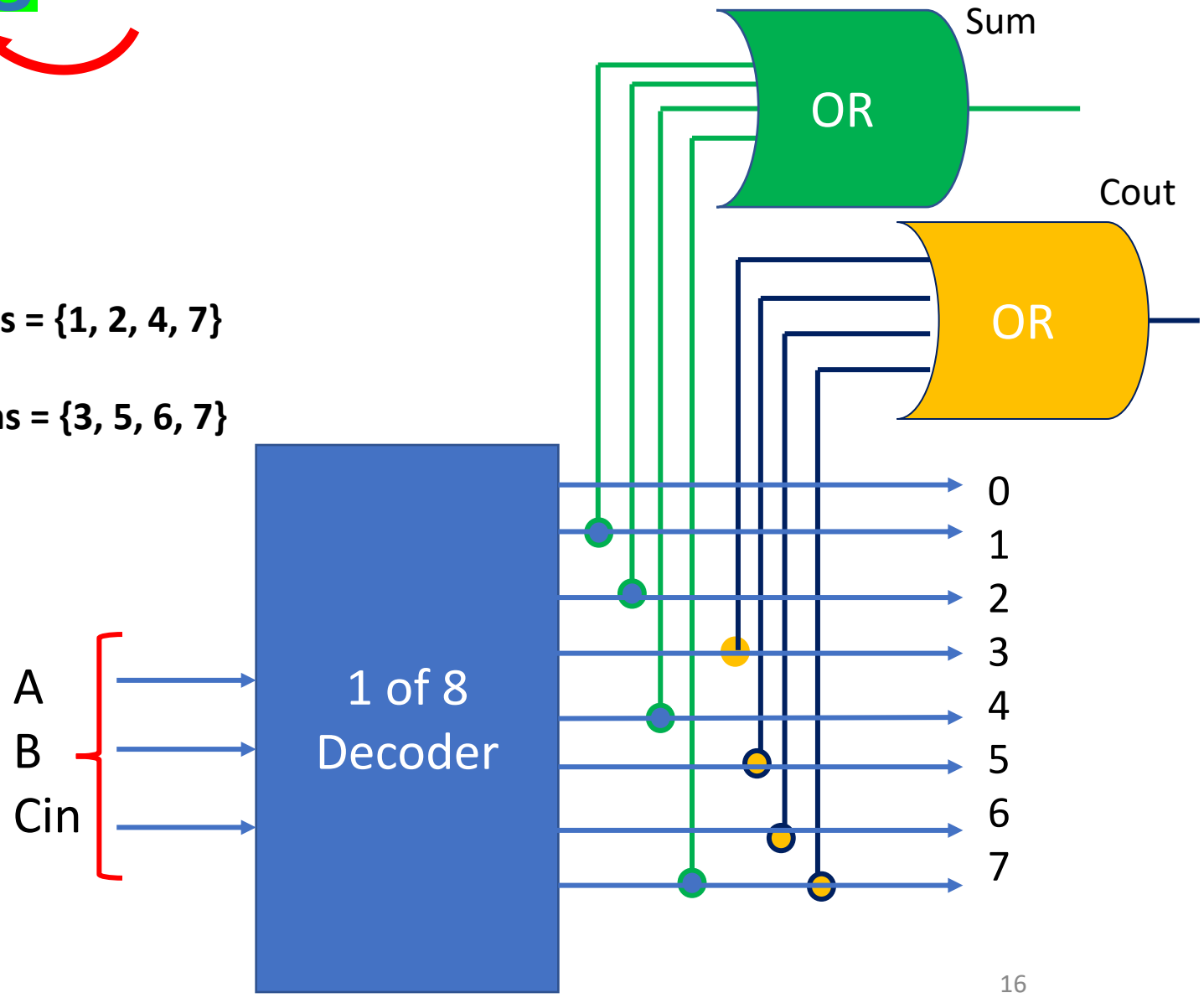


Inputs Outputs

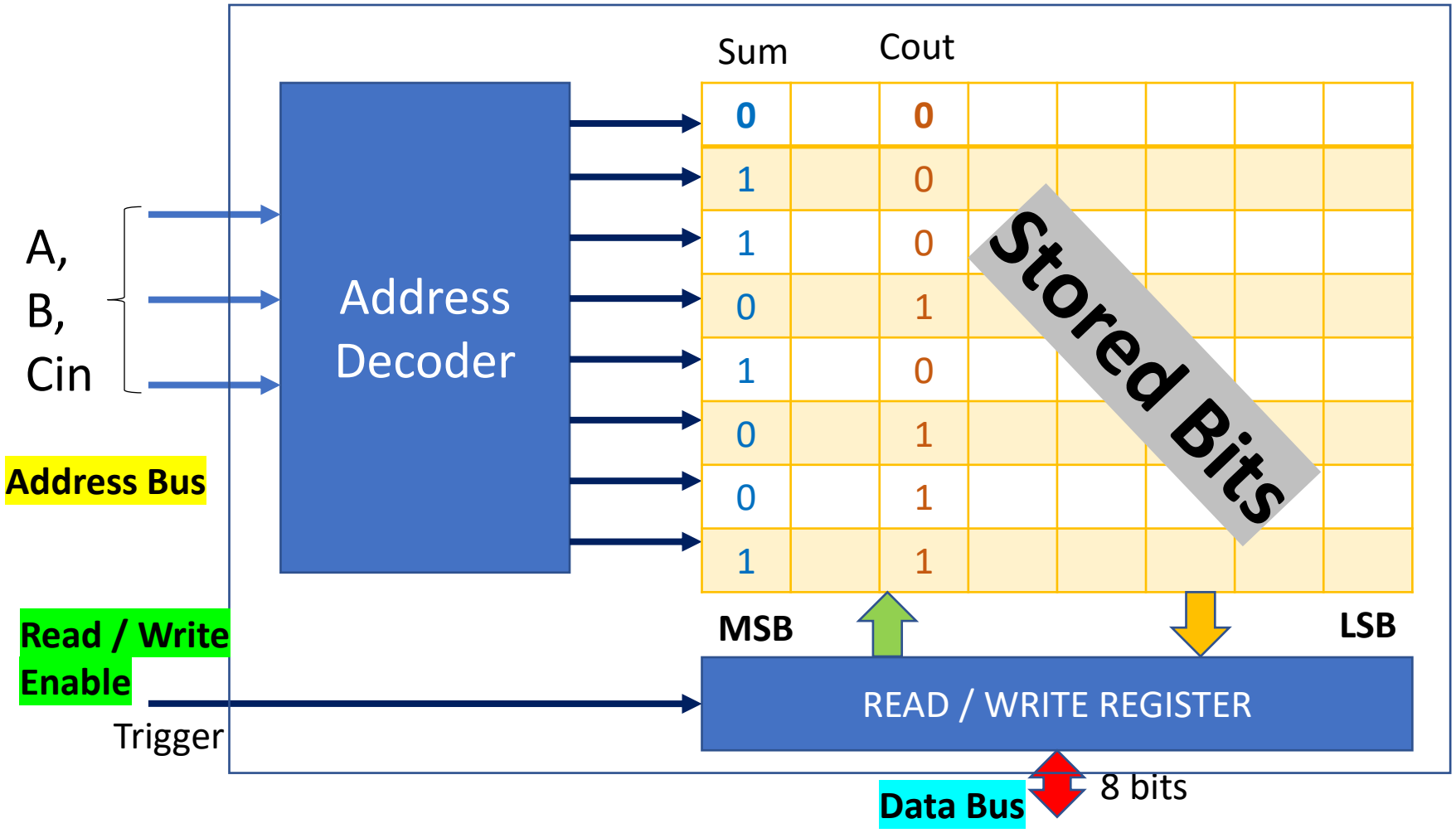
A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Sum Minterms = {1, 2, 4, 7}

Cout Minterms = {3, 5, 6, 7}



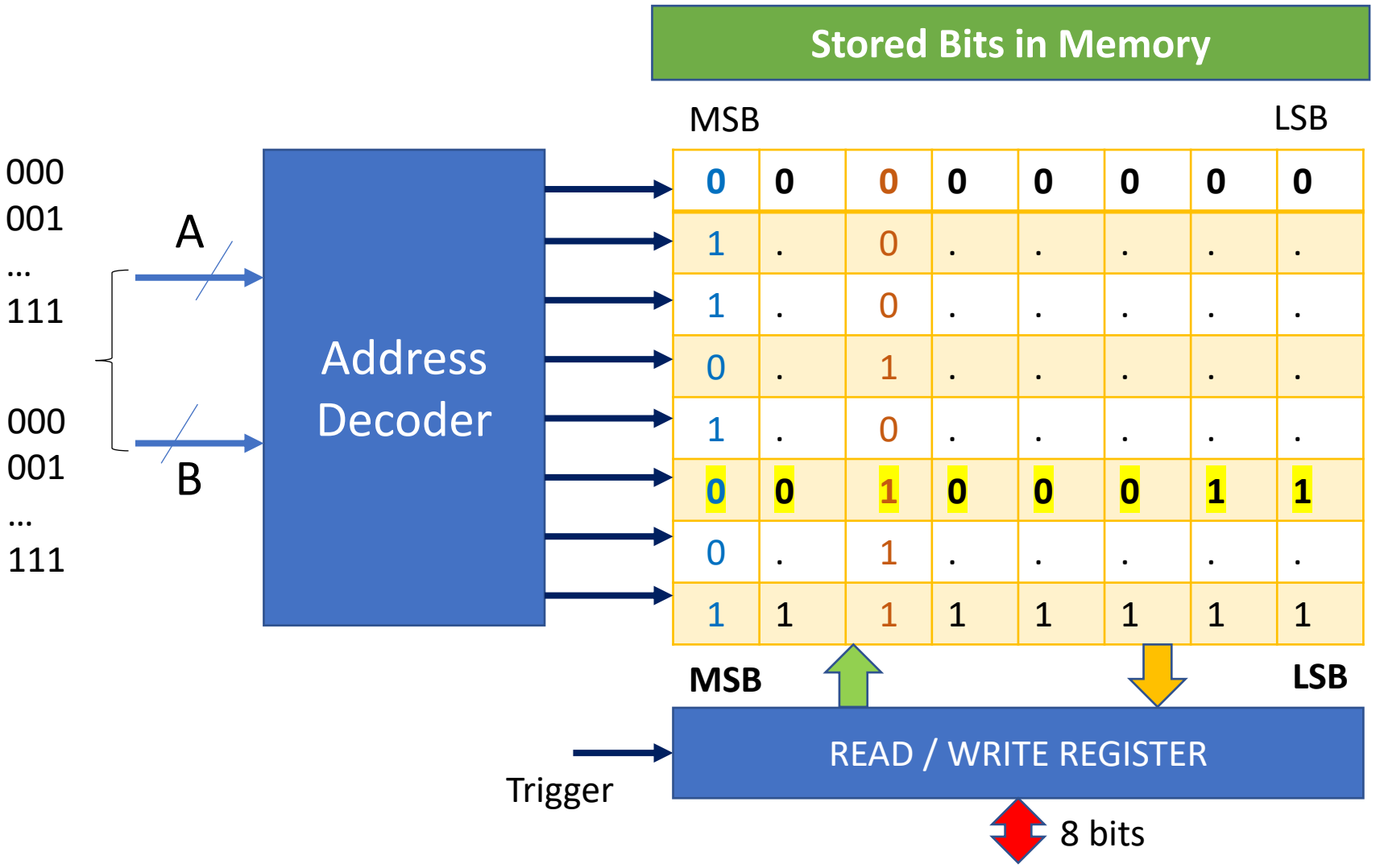
Memory Storage (RAM or ROM) as a functional logic block – Lookup Tables



Truth Table Outputs are Programmed in the memory Lookup Table

MSB output (Bit 7) is Sum
MSB-2 output (Bit 5) is Cout

Implement 3 bit x 3 bit Multiplier using Memory



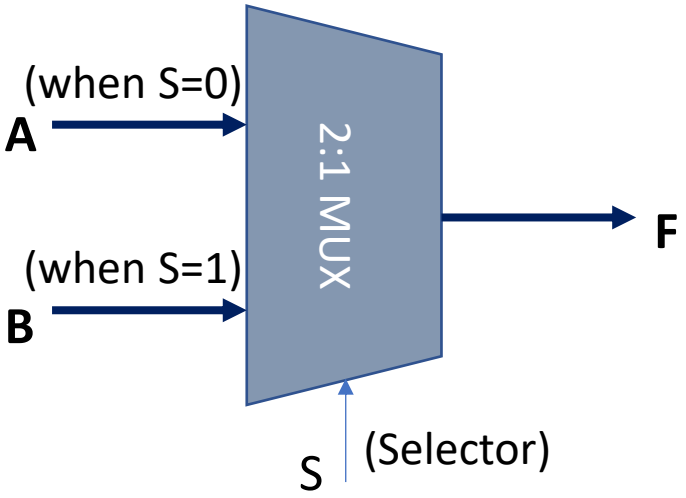
5x7

Truth Table Outputs are Programmed in the memory Lookup Table

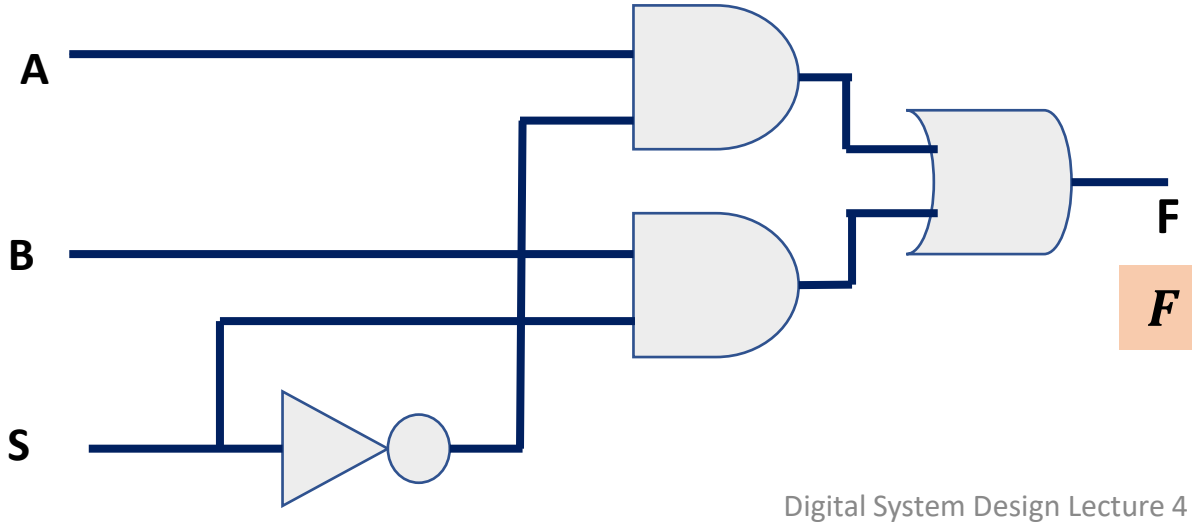
A	B	AxB	Data
0	0	0	00 H
0	1	0	00 H
....			
0	7	0	00 H
0	1	0	00 H
1	1	1	01 H
2	1	2	02 H
....			
3	2	6	06 H
.....			
5	7	35	23 H
....	....	....	....

Multiplexer MUX Functional Module

Selects one of many inputs at the output

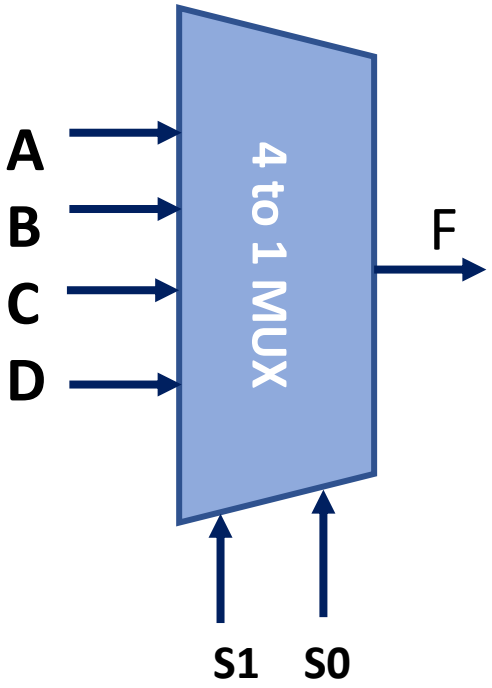


Inputs		Select	Output
A	B	S	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1



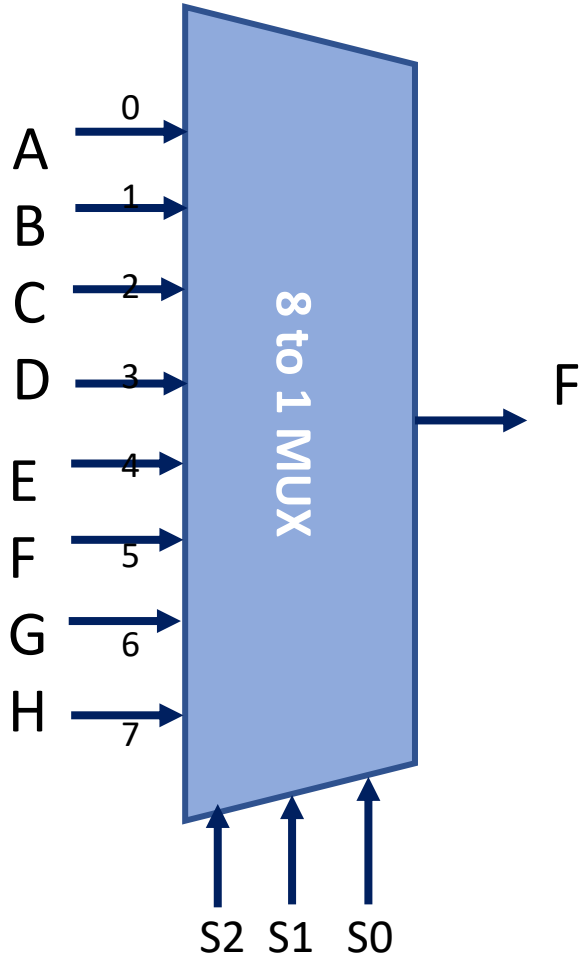
$$F = \bar{S}A + SB$$

4 to 1 MUX



Inputs				Select		Output
A	B	C	D	S1	S0	F
A	B	C	D	0	0	A
A	B	C	D	0	1	B
A	B	C	D	1	0	C
A	B	C	D	1	1	D

8 to 1 MUX



Input Lines								Select lines			Output
A	B	C	D	E	F	G	H	S2	S1	S0	F
A	B	C	D	E	F	G	H	0	0	0	A
A	B	C	D	E	F	G	H	0	0	1	B
A	B	C	D	E	F	G	H	0	1	0	C
A	B	C	D	E	F	G	H	0	1	1	D
A	B	C	D	E	F	G	H	1	0	0	E
A	B	C	D	E	F	G	H	1	0	1	F
A	B	C	D	E	F	G	H	1	1	0	G
A	B	C	D	E	F	G	H	1	1	1	H

Map a full adder to a 4 to 1 MUX

Full Adder Truth Table

A	B	Cin	Sum	Cout
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

MUX Selector Groups (2 Bits, 4 groups)

Observations

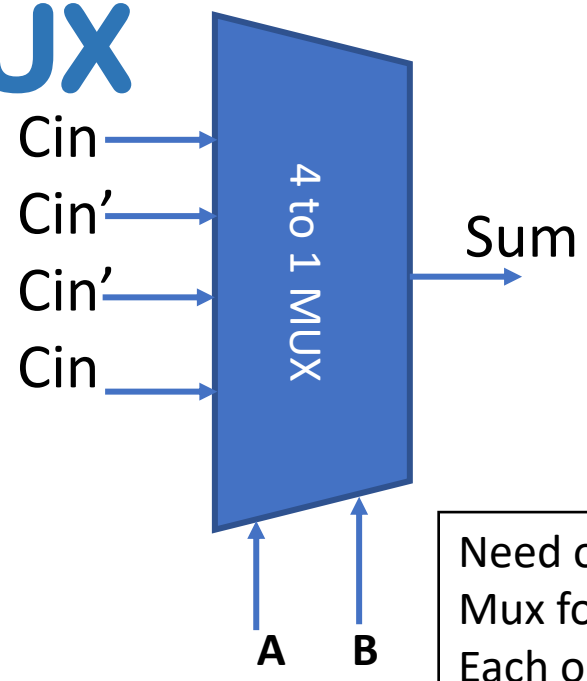
Select = 00 — Sum output follows Cin; Cout remains 0

Select = 01 — Sum output follows Cin'; Cout follows Cin

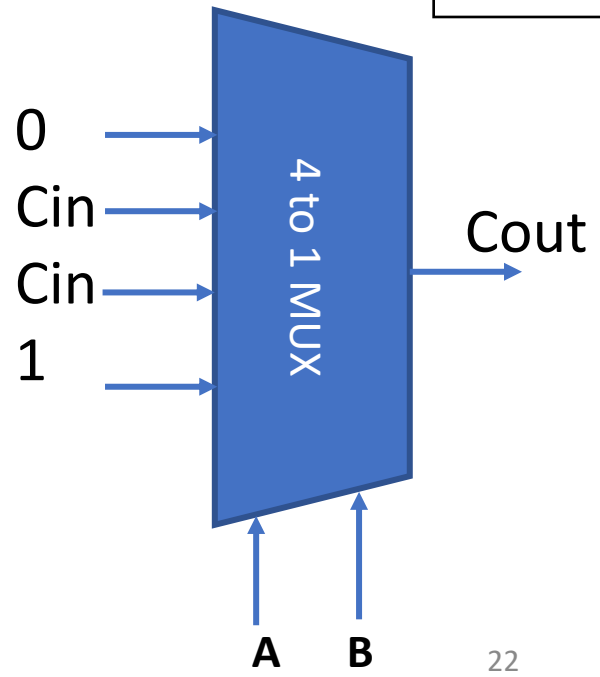
Select = 10 — Sum output follows Cin'; Cout follows Cin

Select = 11 — Sum output follows Cin; Cout remains 1

AND
Selector is same as inputs
AB



Need one Mux for Each output



Mapping 16 Minterms to a 8 to 1 MUX

A	B	C	D	Output F	
0	0	0	0	0	
0	0	0	1	1	Select = 000
0	0	1	0	0	
0	0	1	1	1	Select = 001
0	1	0	0	1	
0	1	0	1	0	Select = 010
0	1	1	0	0	
0	1	1	1	0	Select = 011
1	0	0	0	0	
1	0	0	1	0	Select = 100
1	0	1	0	0	
1	0	1	1	1	Select = 101
1	1	0	0	1	
1	1	0	1	1	Select = 110
1	1	1	0	1	
1	1	1	1	1	Select = 111

3 Select Lines so 8 groups

Observations

F follows D

F follows D

F follows D’

F remains 0

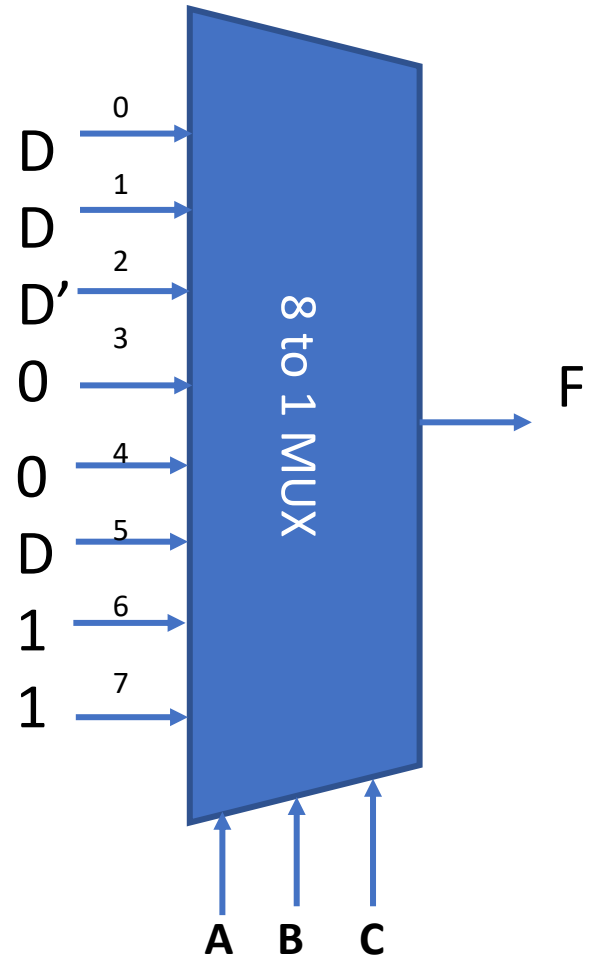
F remains 0

F follows D

F remains 1

F remains 1

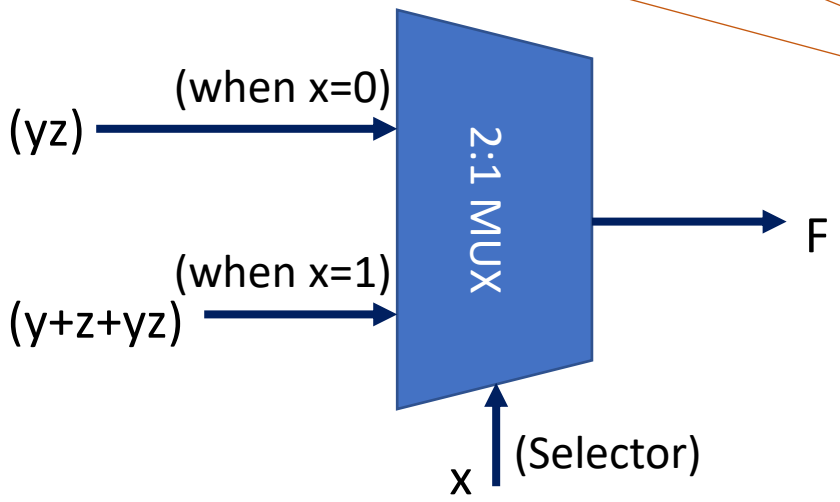
Mapping



Minterms {1, 3, 4, 11, 12, 13, 14, 15}

Example: Implement logic functions using only 2 to 1 MUXes

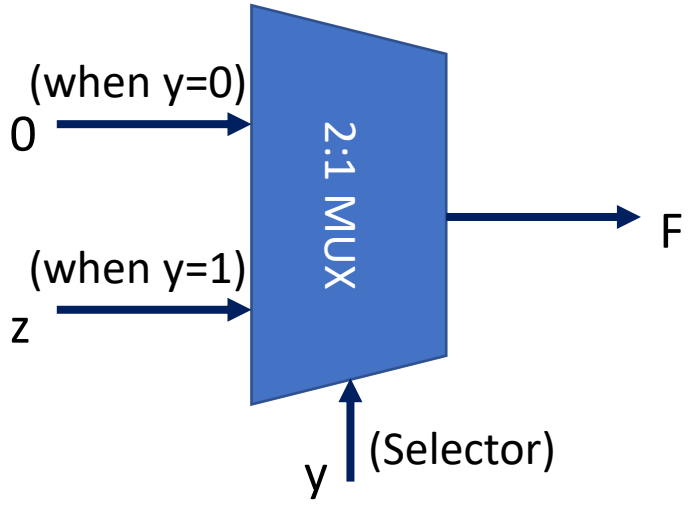
F=x'(yz) + x(y+z+yz), use x as selector input



Now we have to find a way to map (yz) and (y+z+yz) into 2 to 1 MUX

Use y as select lines

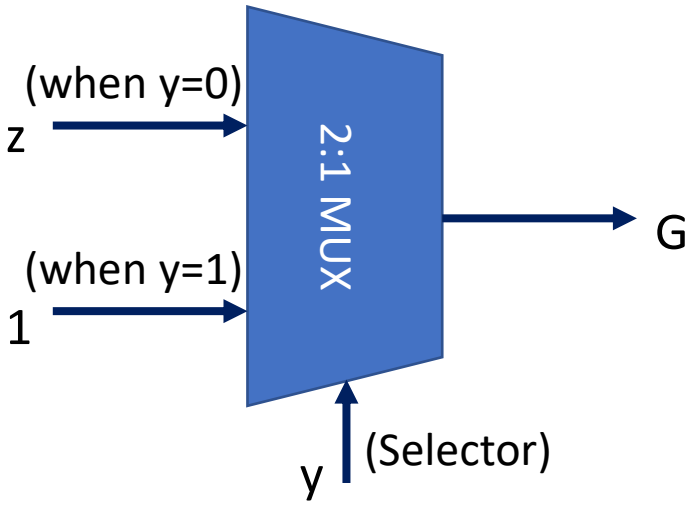
For (yz); when y = 0, output is 0
when y = 1, output is z



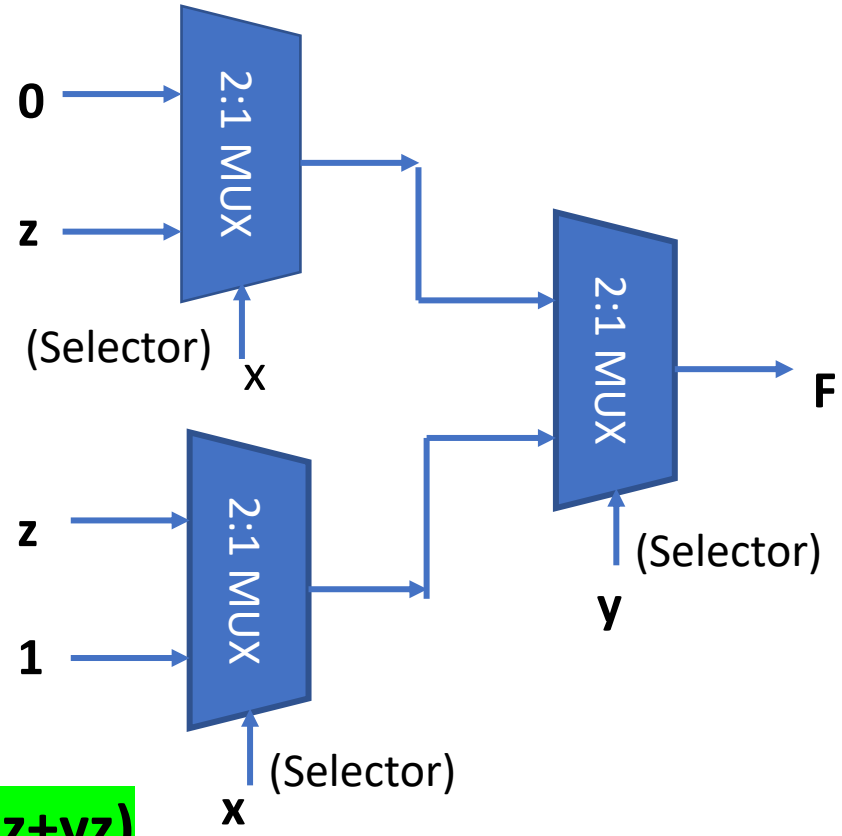
……. continued

Contd - Mapping only on 2 to 1 MUX

Let G = (y+z+yz)
Then G = y + z(y+y') + yz
G=y(1+z+y) + z(y')
G=y(1) + y'(z)



Full implementation using only 2 to 1 MUX



Implemention of F=x'(yz) + x(y+z+yz)

Conclusion

- Behaviour of Digital System is described in Truth Table or Boolean Algebra
- For Implementation of this Behaviour, we can use:
  - Logic Gates – **detailed and elaborate gate level design** - Complicated
  - Decoders
  - Multiplexers
  - Memory

High Level Mapping to Manage COMPLEXITY