Digital Logic and Computer Organization

Combinational Logic - Analysis and Design

Combinational Circuit

- A circuit consists of an inter-connection of logic gates.
- A logic circuit is combinational if its output(s) are a function of only the present inputs at any time.
- The inputs to a circuit can be viewed as binary variables from an external source.
- The outputs of a circuit are variables produced by the circuit based on the input signals and go to external destinations.
- For n input variables, there are 2^n possible combination of the binary inputs.
- For each distinct input combination, there is one value for each output variable.

Analysis of Combinational Circuits

- How to make sure the circuit to be analyzed is combinational? - In the schematic, there is no feedback paths or memory elements.
- Analyze by establishing a truth table: for each input combination, trace the schematic to establish its output.
- Literal analysis
    - Use meaningful symbols to name the output of each gate in the circuit;
    - Starting from the gates whose inputs are circuit inputs, find the Boolean functions for these gates;
    - For those gates whose inputs include the output of other gates whose functions are already established, find the Boolean functions for these gates;
    - Repeat the previous step until the Boolean function of the circuit's output is found.

Design Procedure

- From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each; (determine the interface of the circuit;)
- Derive the truth table that defines the required relationship between inputs and outputs; (determine the behaviour of the circuit;)
- Obtain the simplified Boolean function for each output as a function of the input variables; (determine the functionality of the circuit;)
- Draw the logic schematic; (determine the structure of the circuit;)
- Verify the correctness of the design by building a prototype or by simulation.

Design Examples/ Applications

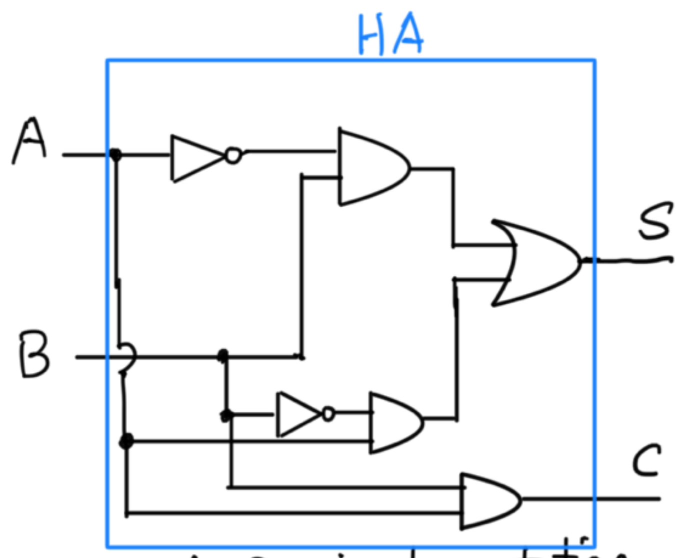
- Half Adders

Half Adder

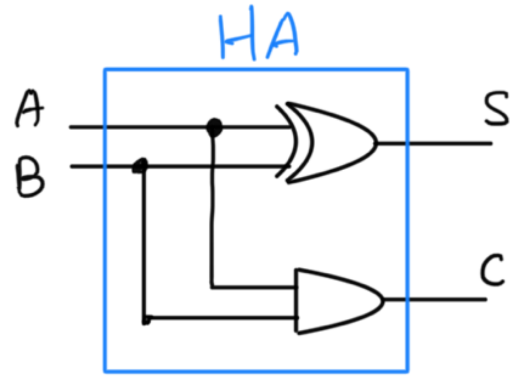
Augend	Addend	sum	carry
Input		output	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S = \bar{A}B + A\bar{B}$$
$$= A \oplus B$$

$$C = A \cdot B$$



And-Or implementation



XOR-And implementation

Full Adder

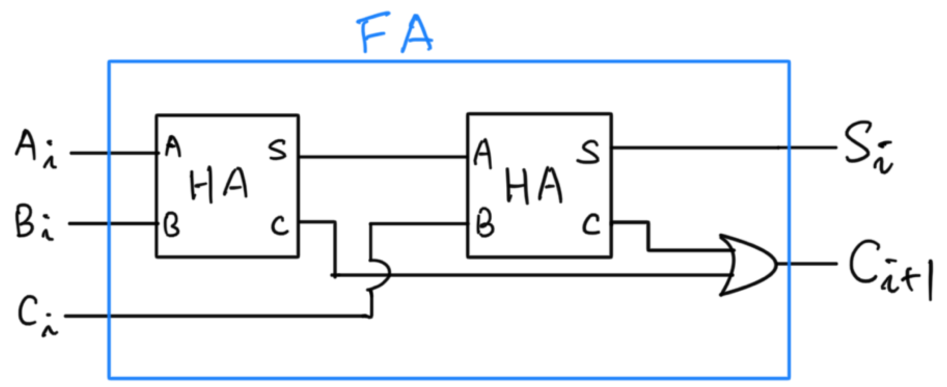
- Full Adder: Sum-of-Product; 2 Half Adders and an OR

Full Adder

Input			Output	
A_i	B_i	C_i	S_i	C_{i+1}
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

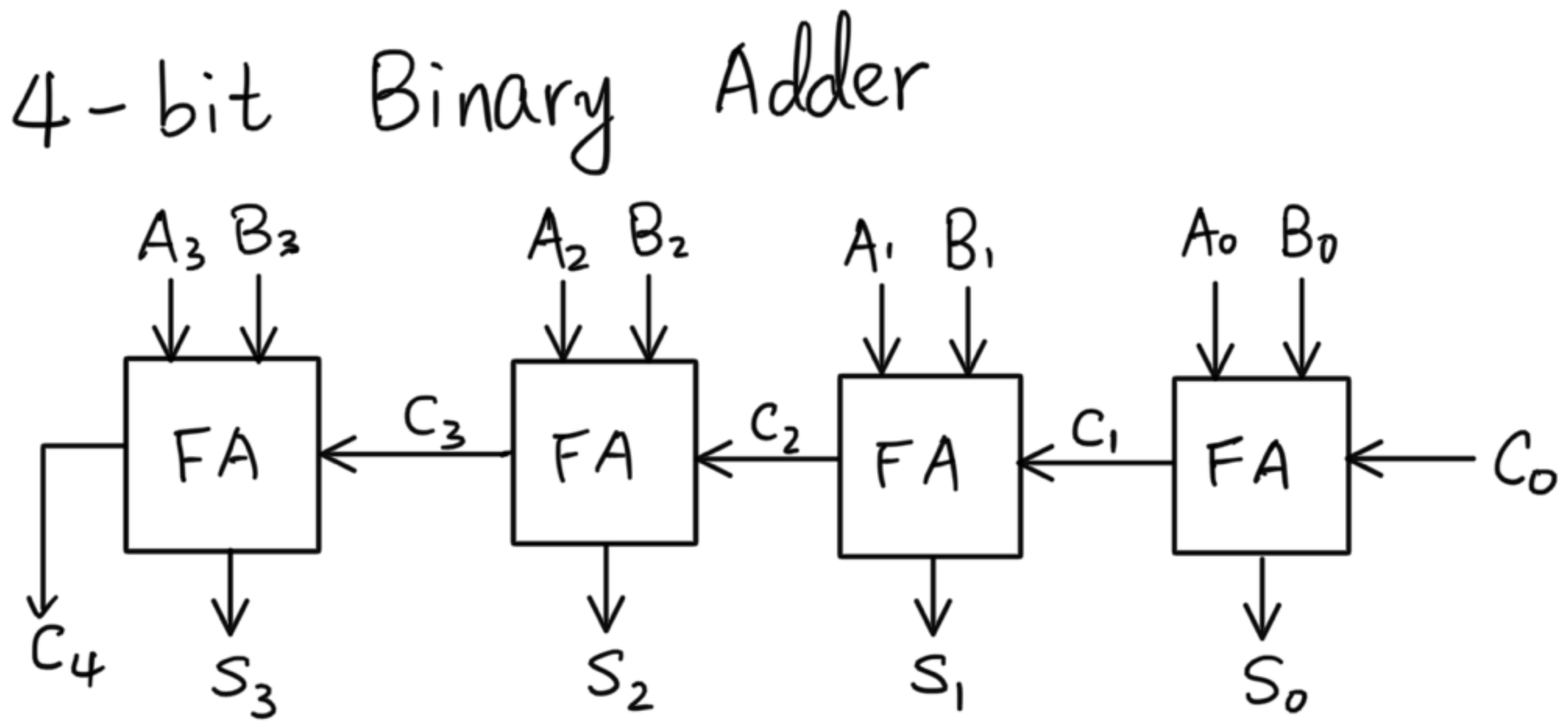
$$S_i = \bar{A}_i \bar{B}_i C_i + \bar{A}_i B_i \bar{C}_i + A_i \bar{B}_i \bar{C}_i + A_i B_i C_i$$

$$C_{i+1} = A_i B_i + A_i C_i + B_i C_i$$



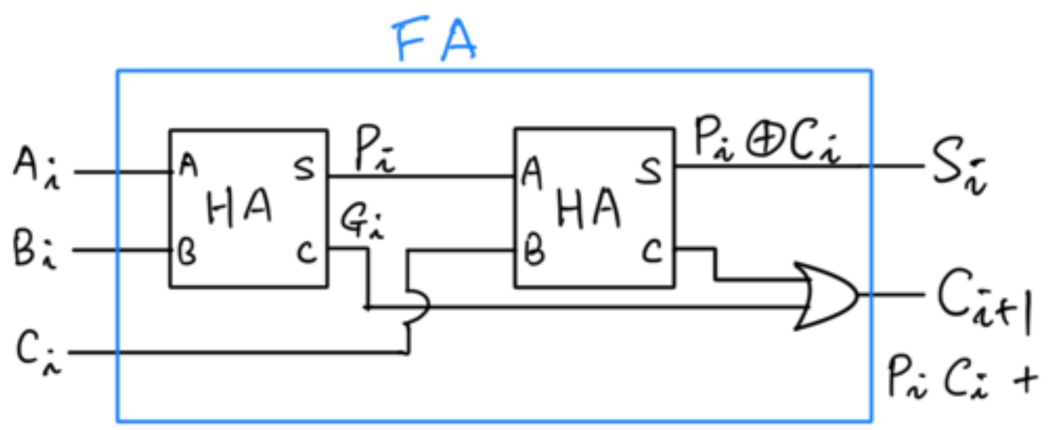
Binary Adder

- Binary Adder: Multi-bit ripple carry Adder



Binary Adder

- Issue: Carry propagation causing delays
- Solution: Carry lookahead



Carry Propagate $P_i = A_i \oplus B_i$

Carry Generate $G_i = A_i B_i$

C_0 = input Carry

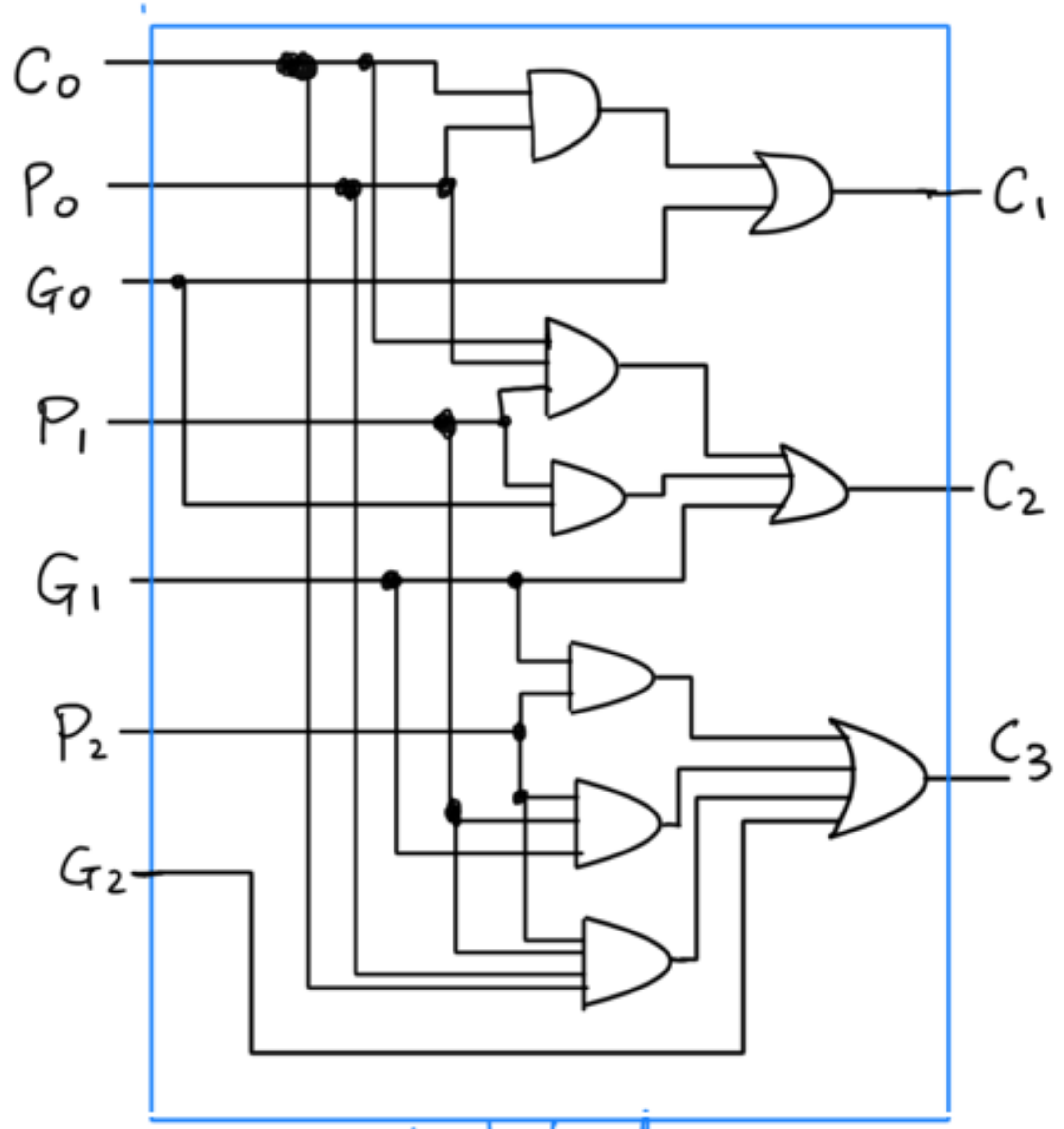
$C_1 = G_0 + P_0 C_0$

$C_2 = G_1 + P_1 C_1 = G_1 + P_1(G_0 + P_0 C_0)$

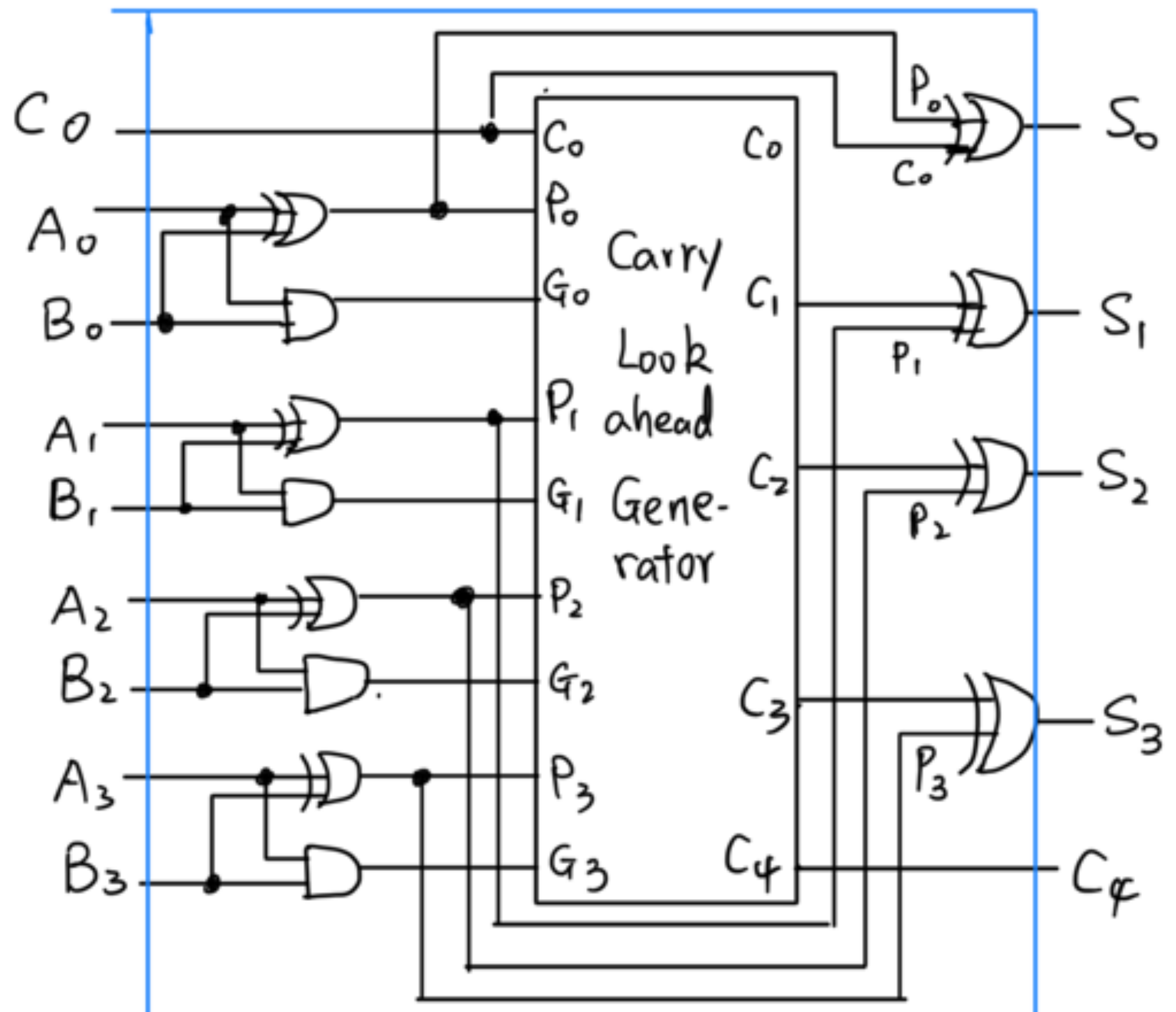
$= G_1 + P_1 G_0 + P_1 P_0 C_0$

$C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$

Binary Adder with Lookahead

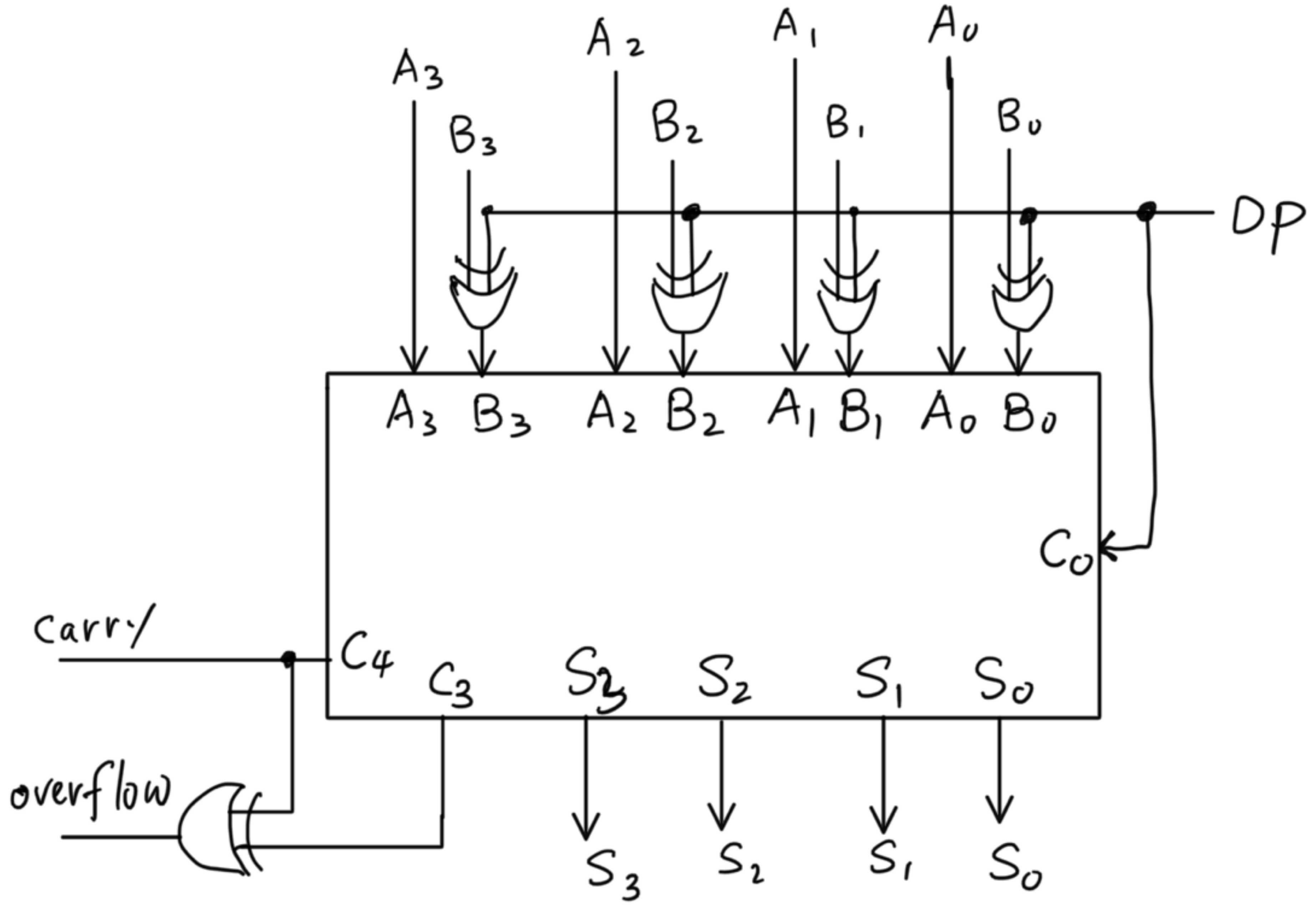


Carry Lookahead Generator



4 bit adder with carry lookahead

Binary Subtractor



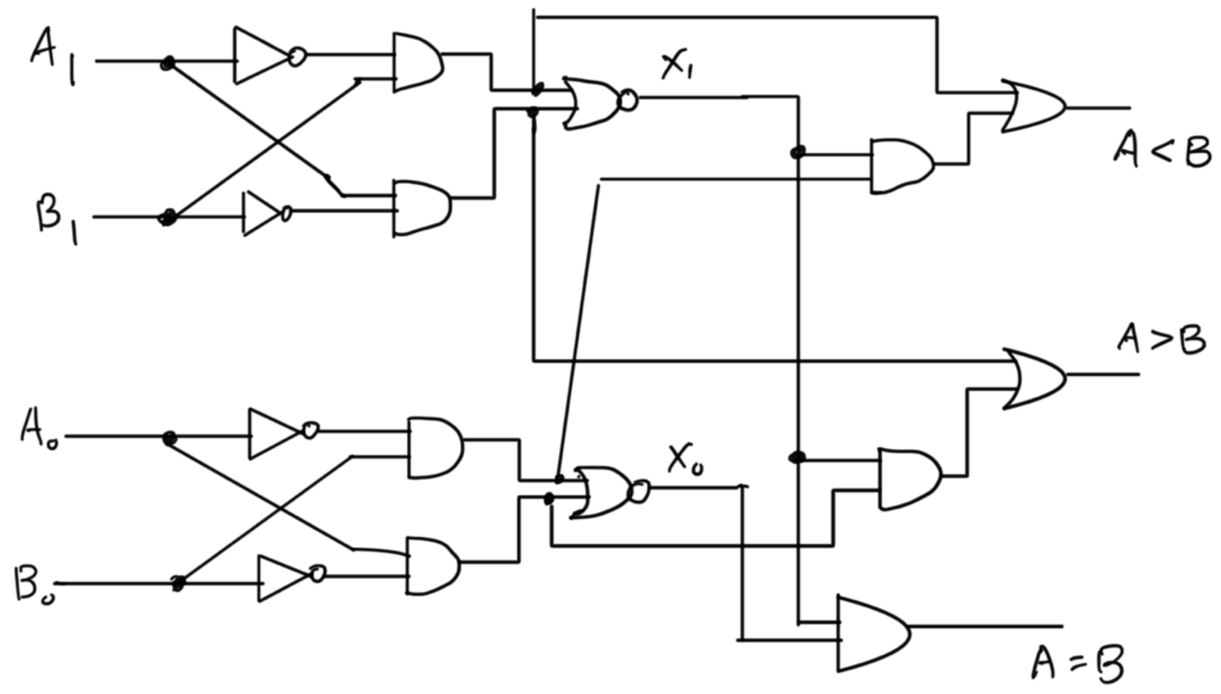
Magnitude Comparator

$x_i = A_i B_i + \overline{A_i}\,\overline{B_i}$ for $i = 0, 1, 2, 3$

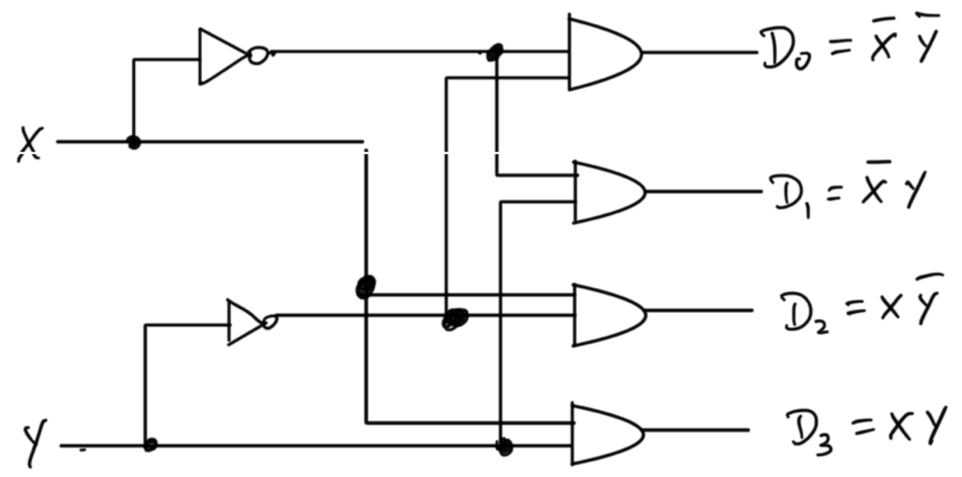
$F_{A=B} = x_3 \cdot x_2 \cdot x_1 \cdot x_0$

$F_{(A>B)} = A_3 \overline{B_3} + x_3 A_2 \overline{B_2} + x_3 x_2 A_1 \overline{B_1} + x_3 x_2 x_1 A_0 \overline{B_0}$

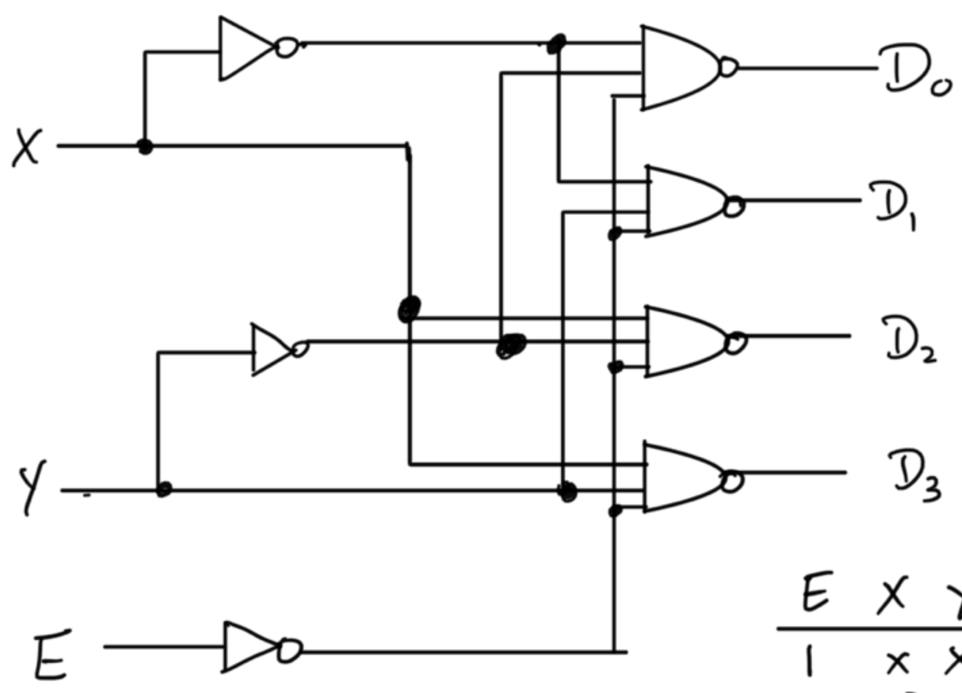
$F_{(A<B)} = \overline{A_3} B_3 + x_3 \overline{A_2} B_2 + x_3 x_2 \overline{A_1} B_1 + x_3 x_2 x_1 \overline{A_0} B_0$



Decoder

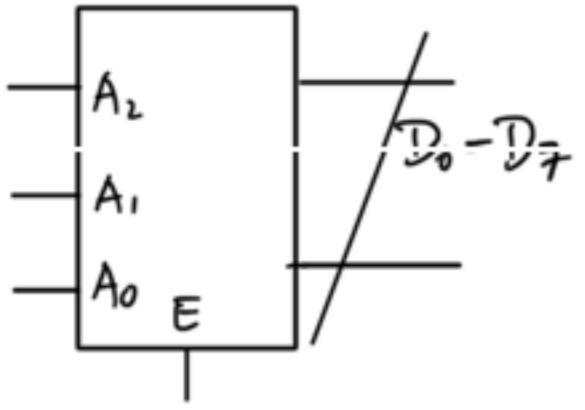


2-to-4 decoder



2-to-4 decoder
with active-low enable
(0 means selected)

E	X	Y	D_0	D_1	D_2	D_3
1	x	x	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0



3-to-8
decoder
with enable

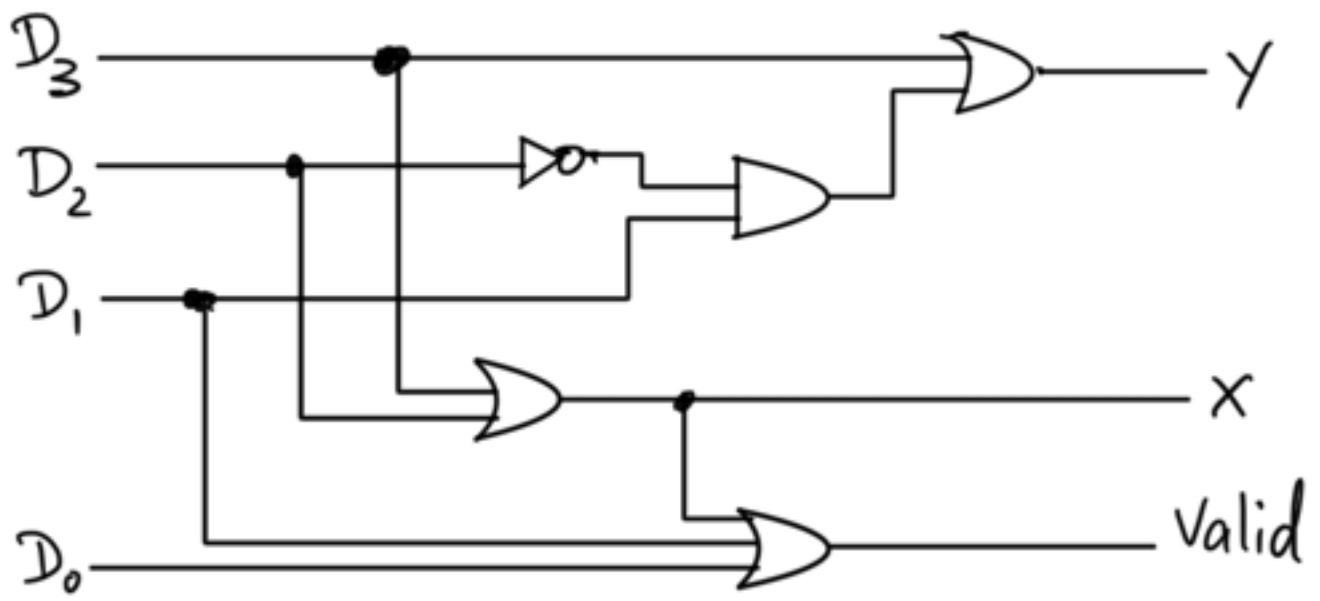
Encoder & Priority Encoder

Input				output	
D_0	D_1	D_2	D_3	X	Y
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

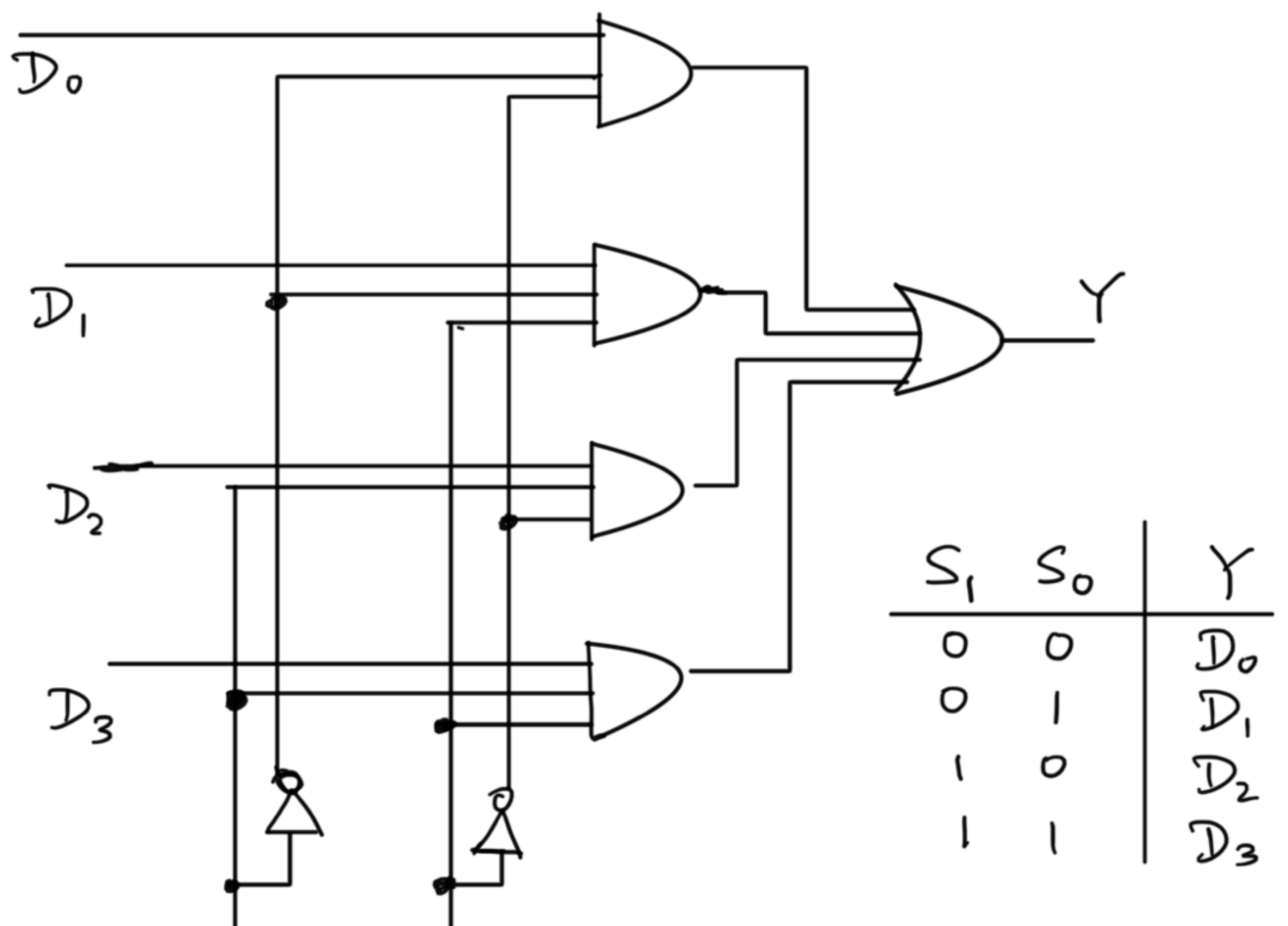
$X = D_2 + D_3$

$Y = D_1 + D_3$

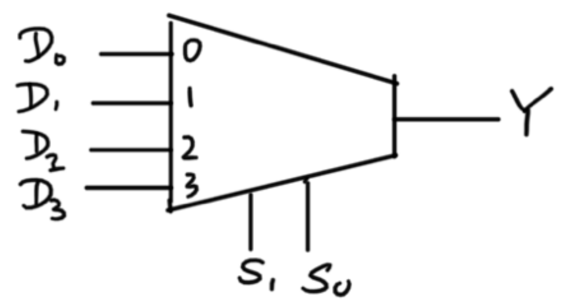
Input				output		
D_0	D_1	D_2	D_3	X	Y	Valid
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	1	1	1	1	1



Multiplexer

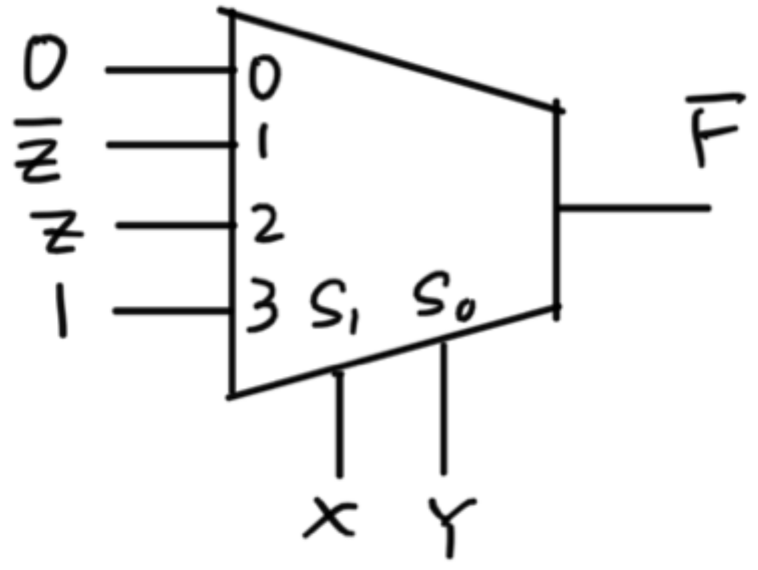


S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3



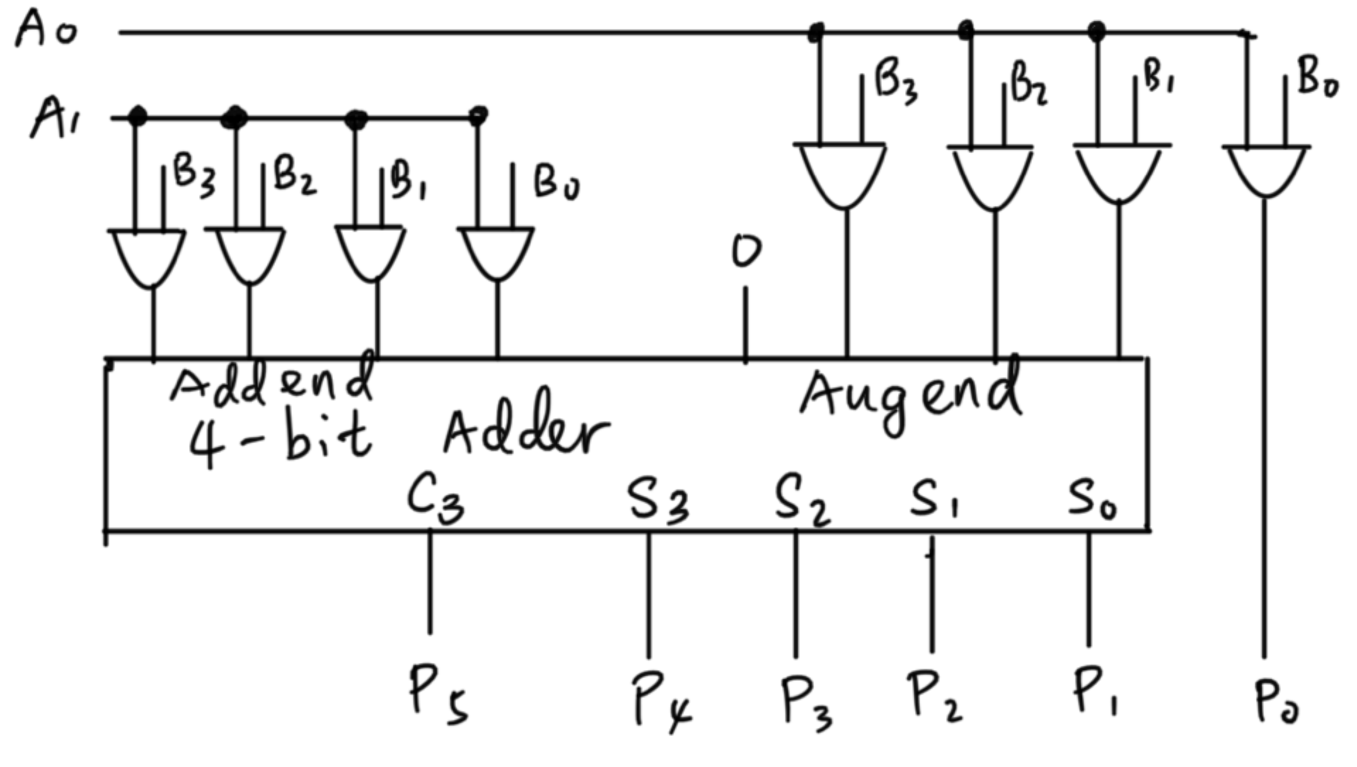
Implement Function using Multiplexer

Input			output	
X	Y	Z	F	
0	0	0	0	$F = 0$
0	0	1	0	
0	1	0	1	$F = \overline{Z}$
0	1	1	0	
1	0	0	0	$F = Z$
1	0	1	1	
1	1	0	1	$F = 1$
1	1	1	1	



Binary Multiplier

$$\begin{array}{cccccc}
 & & B_3 & B_2 & B_1 & B_0 \\
\times & & & & A_1 & A_0 \\
\hline
 & & A_0B_3 & A_0B_2 & A_0B_1 & A_0B_0 \\
+ & A_1B_3 & A_1B_2 & A_1B_1 & A_1B_0 & \\
\hline
P_5 & P_4 & P_3 & P_2 & P_1 & P_0
\end{array}$$



A_0
A_1
B_3
B_2
B_1
B_0
B_3
B_2
B_1
B_0
0
Addend
4-bit Adder
Augend
C_3
S_3
S_2
S_1
S_0
P_5
P_4
P_3
P_2
P_1
P_0