

CET 141: Day 10

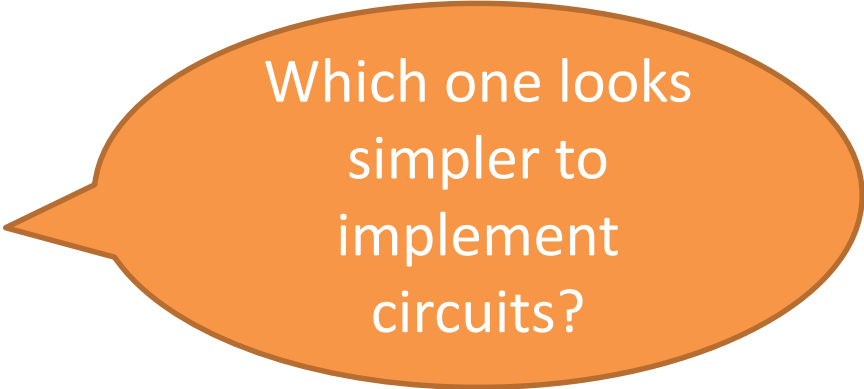
Dr. Noori KIM

Recap: DeMorgan's Theorem

- Used to simplify circuits containing NAND and NOR gates

- $\overline{A B} = \overline{A} + \overline{B}$

- $\overline{A + B} = \overline{A} \overline{B}$



Which one looks simpler to implement circuits?

- DeMorgan's Theorem in circuits → Bubble Pushing
 - Change the logic gate
 - (AND to OR or OR to AND)
 - Add bubbles to the inputs and outputs where there were none and remove original bubbles

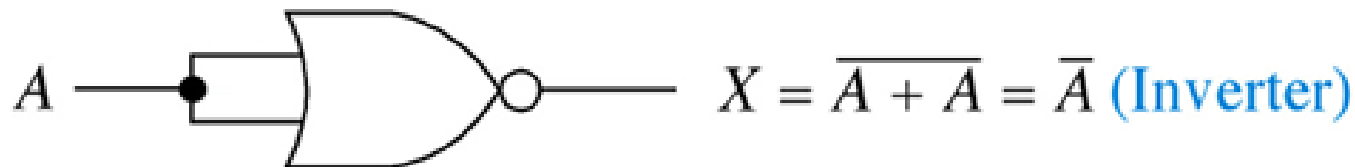
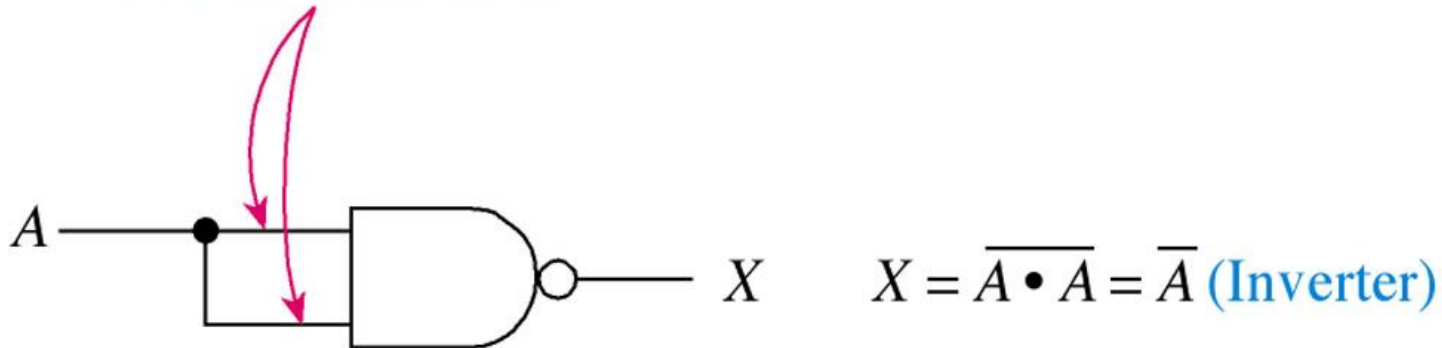
- Bubble Pushing

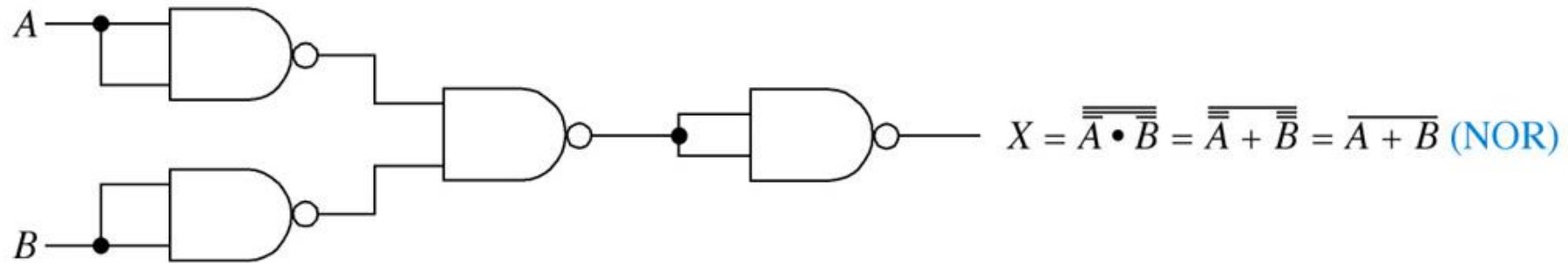
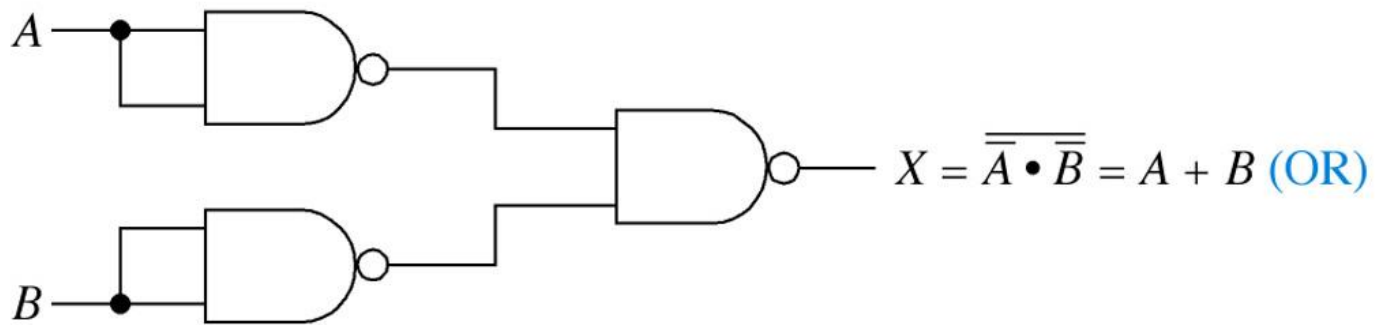
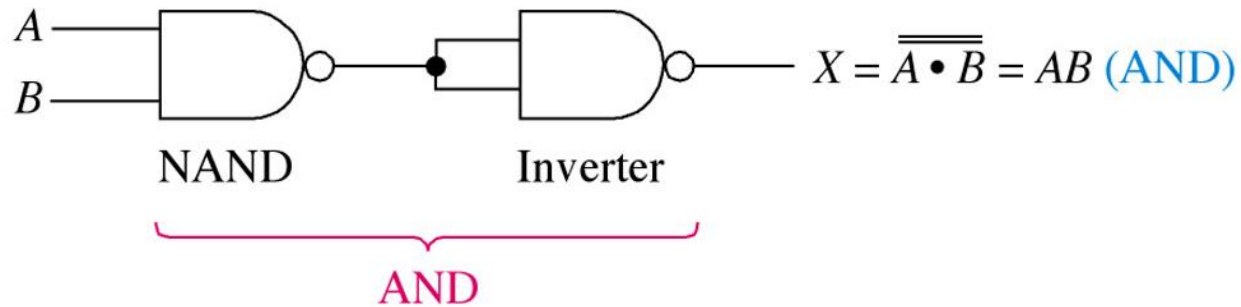


Recap: The Universal Capability of NAND and NOR Gates

- The NAND/NOR as inverters.

Connect both inputs to A to form an Inverter.





Agenda

- Lecture
 - Adder/Subtractor
 - Encoder/Decoder

Combinational Logics

- Using two or more logic gates to perform a more useful, complex function
- A combination of logic functions

ex) $B = KD + HD = D(K+H)$



Equation 1



Equation 2

Are they same, mathematically?

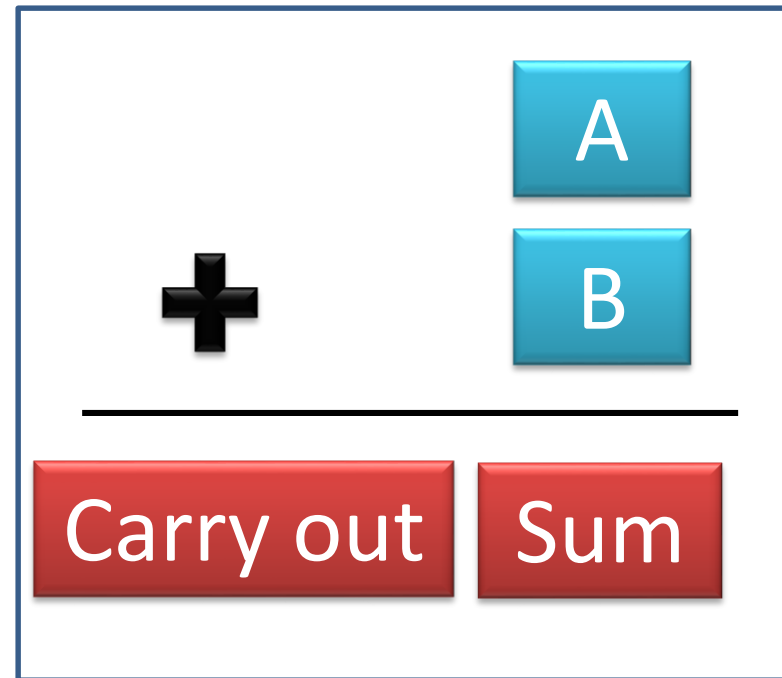
, in circuit implementation?

Adders

Half adder

Inputs

Outputs



<i>A</i>	<i>B</i>	<i>Sum</i>	<i>Carry-Out</i>
0	0		
0	1		
1	0		
1	1		

A	B	Sum	Carry-Out
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

A\B	0	1
0	0	1
1	1	0

K map for Sum

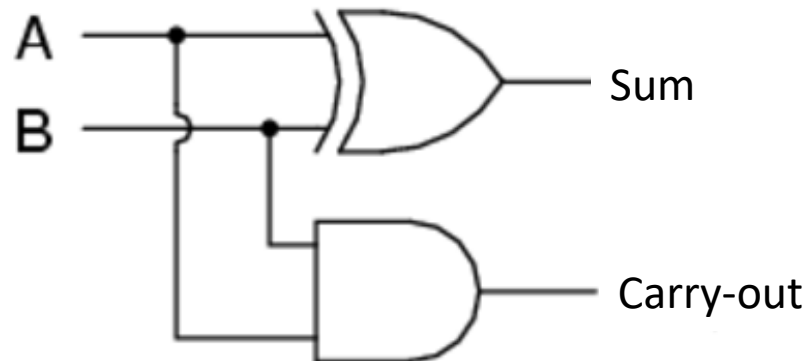
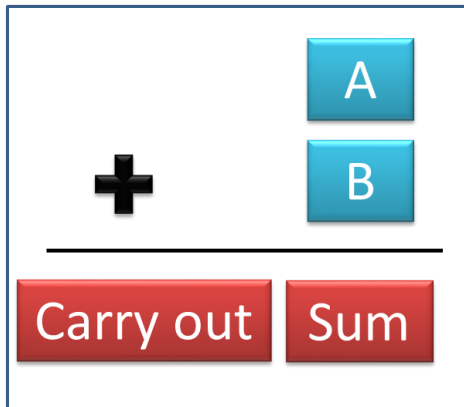
$$Sum = A \cdot \bar{B} + \bar{A} \cdot B \\ = A \oplus B$$

A\B	0	1
0	0	0
1	0	1

K map for Carry-out

$$Carry-out = A \cdot B$$

Half adder



Full adder

Inputs

Outputs

A	B	Carry-In	Sum	Carry-Out
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

+

A

B

Temp-Carry out =
Generate

Temp-Sum =
Propagate

Carry-In

+

Carry-out

Sum

A	B	Carry-In	Sum	Carry-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

AB\Carry-In	0	1
00	0	1
01	1	0
11	0	1
10	1	0

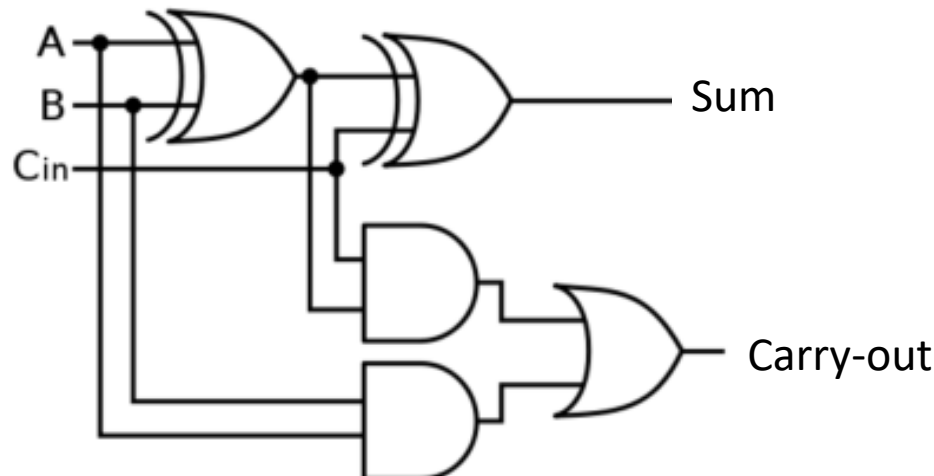
K map for Sum

$$\begin{aligned}
 \text{Sum} &= A'B'C_{in} + A'BC'_{in} + ABC_{in} + AB'C'_{in} \\
 &= A'(B \oplus C_{in}) + A(B \oplus C_{in})' \\
 &= A \oplus B \oplus C_{in}
 \end{aligned}$$

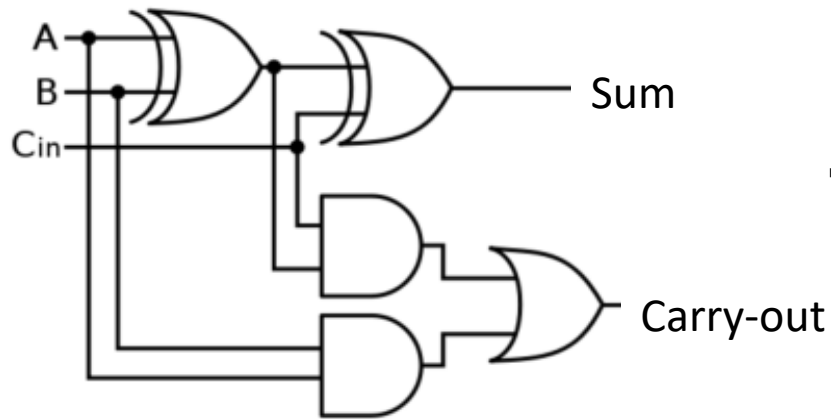
AB\Carry-In	0	1
00	0	0
01	0	1
11	1	1
10	0	1

K map for Carry-out

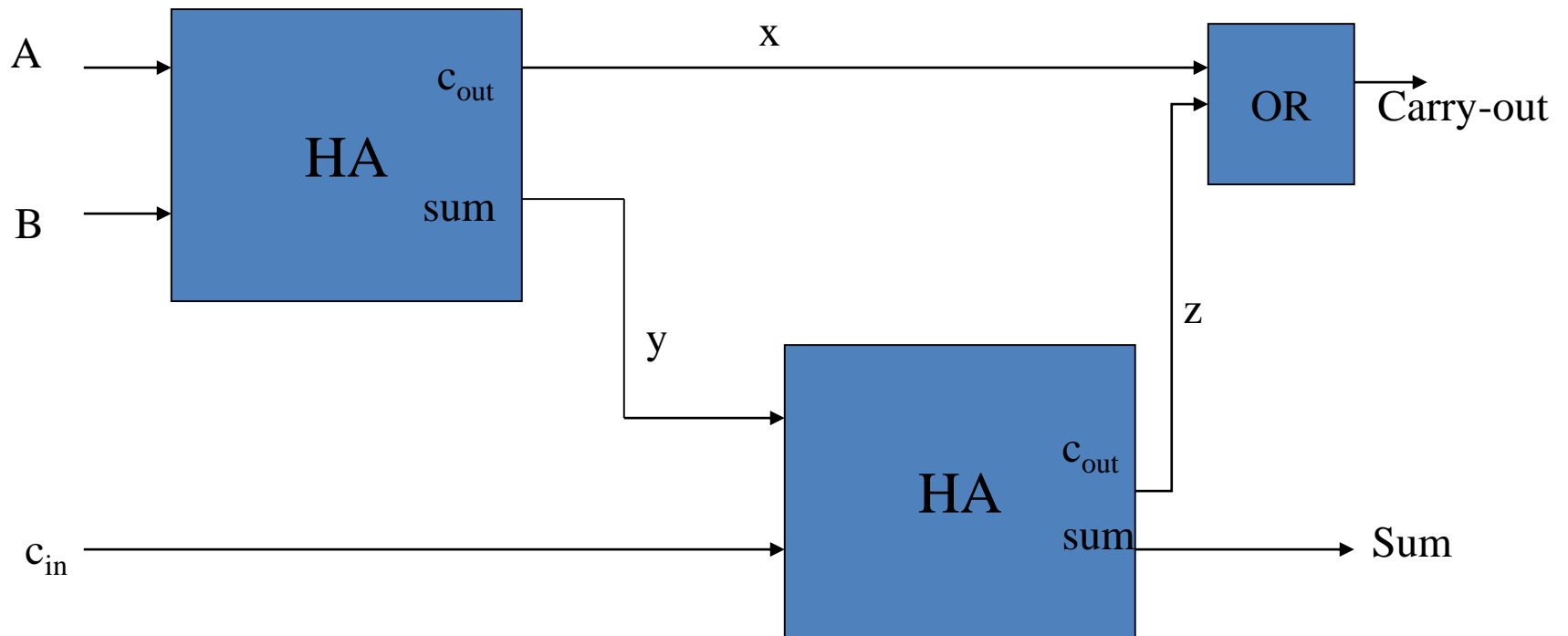
$$\begin{aligned}
 \text{Carry-out} &= AB + BC_{in} + AC_{in} \\
 &= AB + A'BC_{in} + AB'C_{in} \\
 &= AB + C_{in}(B \oplus A)
 \end{aligned}$$



Think: where are the temp carry out and temp sum?



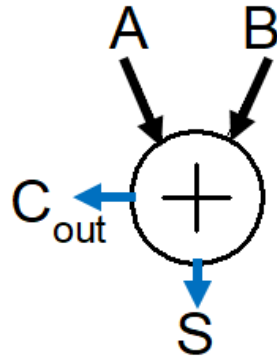
➔ Cascading two Half Adders (HA) ?



A half Adder

$$S = A \oplus B$$

$$C_{out} = A \cdot B$$



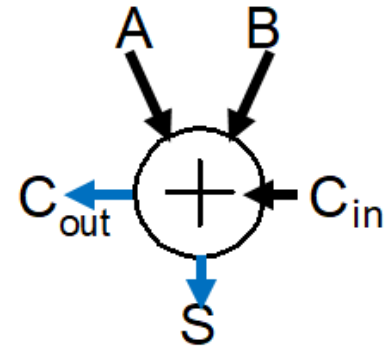
A	B	C_{out}	S
0	0		
0	1		
1	0		
1	1		

A full Adder

$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = MAJ(A, B, C_{in})$$

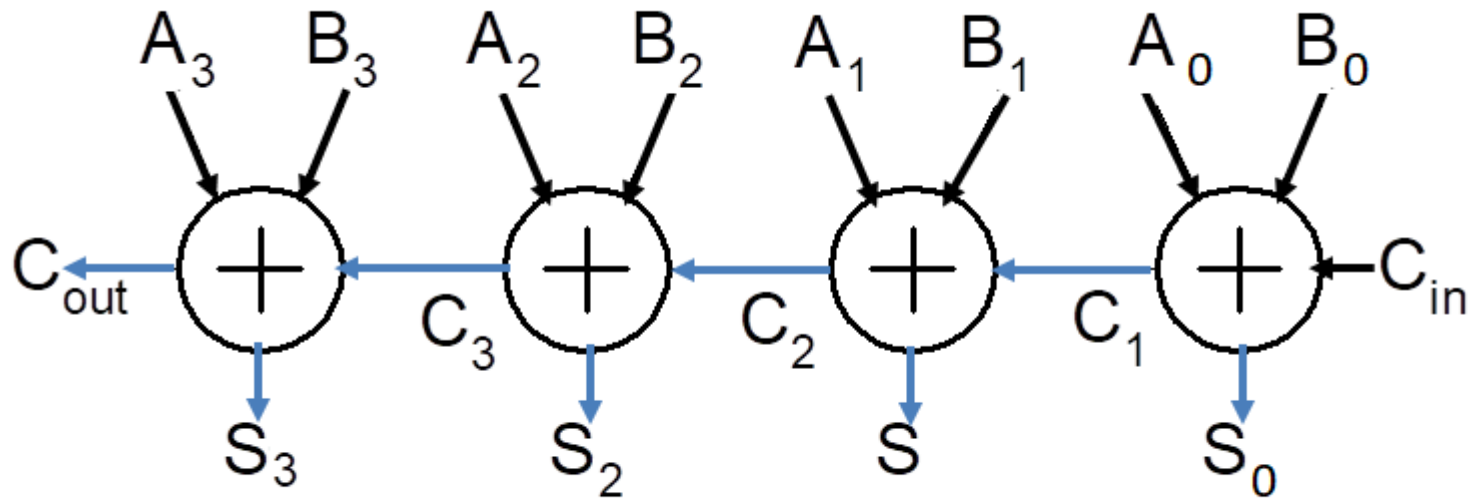
$$= AB + C_{in}(B \oplus A)$$



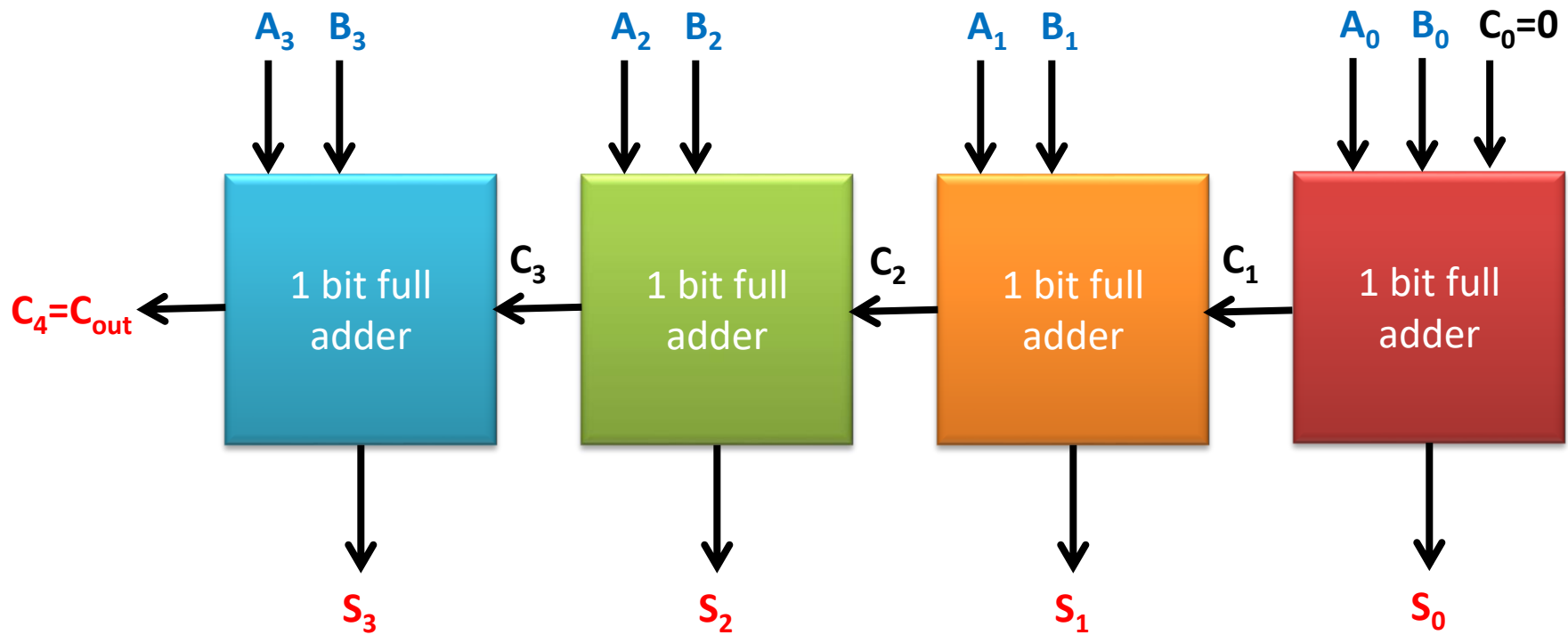
A	B	C_{in}	C_{out}	S
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Ripple-Carry Adders

- Simplest design: cascade full adders
 - Critical path goes from C_{in} to C_{out}



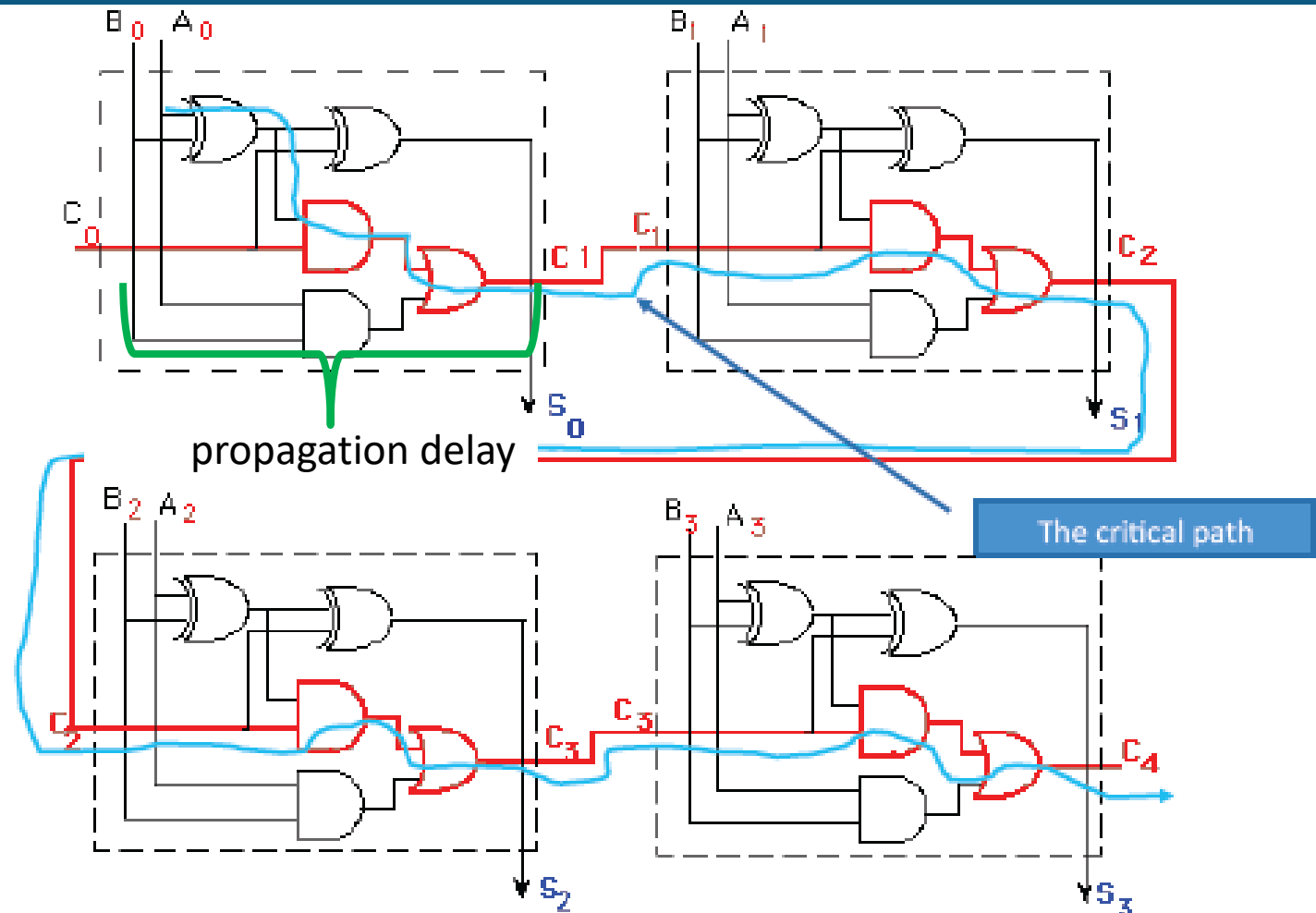
Inputs
Outputs



$$S_i = A_i \oplus B_i \oplus C_i$$
$$C_{i+1} = A_i B_i + C_i (A_i \oplus B_i)$$

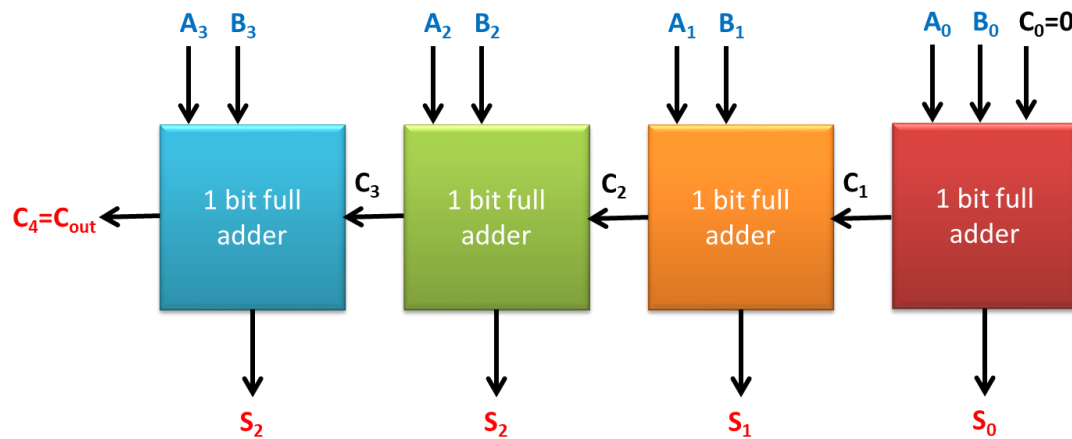


Ripple-Carry Adder



Assume the 2-input XOR has 2 gate delays, and all other 2-input gates have 1 gate delay:

Gate delay for C₄: $4 + (n-1) \cdot 2 = 2n + 2$

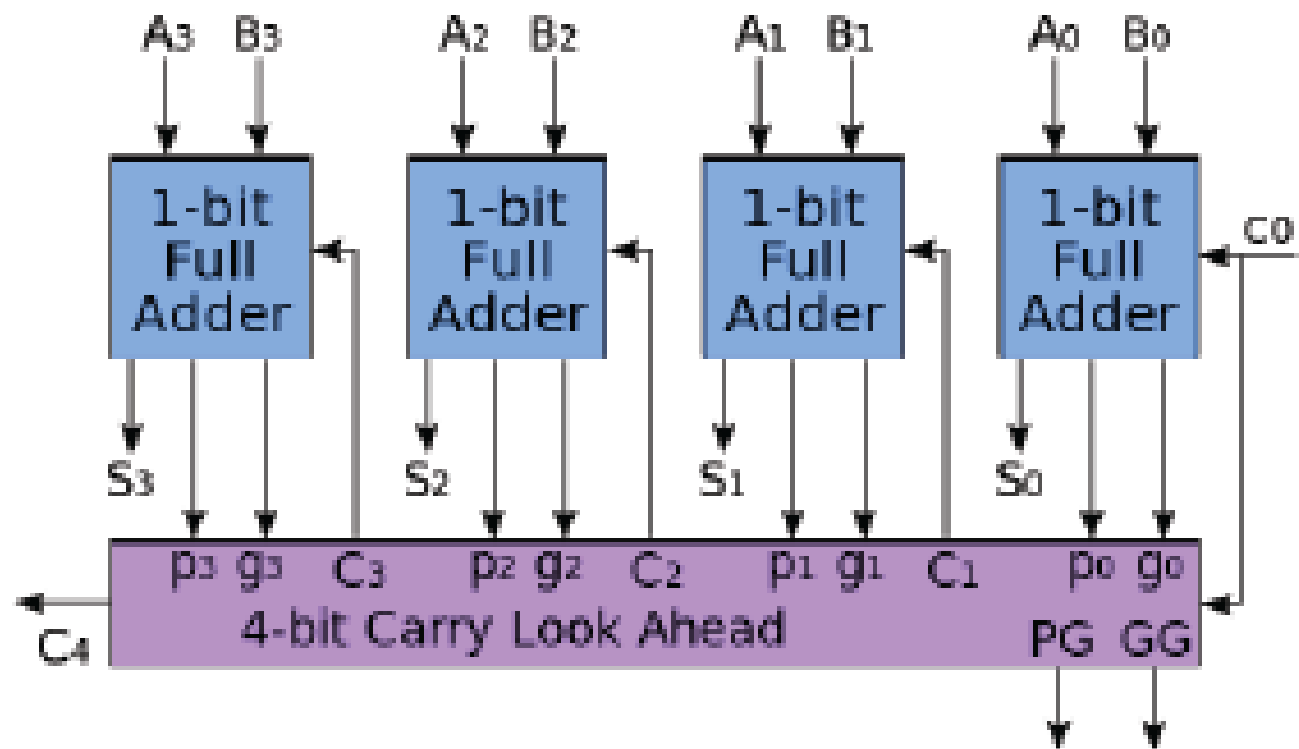


- When A_0 and $B_0=1 \rightarrow C_1=1$
 - However, it takes some time to settle down logic, so C_1 shows up after A_1 and B_1 inputs.
 - Thus, before C_1 shows up, the second full adder does not produce its outputs.
- The worst case, C_4 is not correctly computed until
 - $4 \times$ propagation delay
 - C_n is not computed until $n \times$ propagation delay.

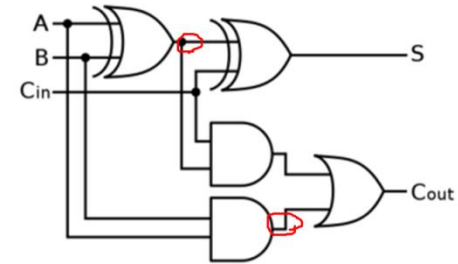
- The limiting factor of ripple-carry adder is the time it takes to propagate the carry.
- The **carry look-ahead adder (CLA)** solves this problem by calculating the carry signals in advance, based on the input signals.
- The result is a reduced carry propagation time.

Carry-Lookahead Adder

Carry-lookahead adders.



- P and G (at half adders, HA, logics)
 - $P_i = A_i \oplus B_i$: Carry propagate (Sum at HA)
 - $G_i = A_i \bullet B_i$: Carry generate (Carry-out at HA)



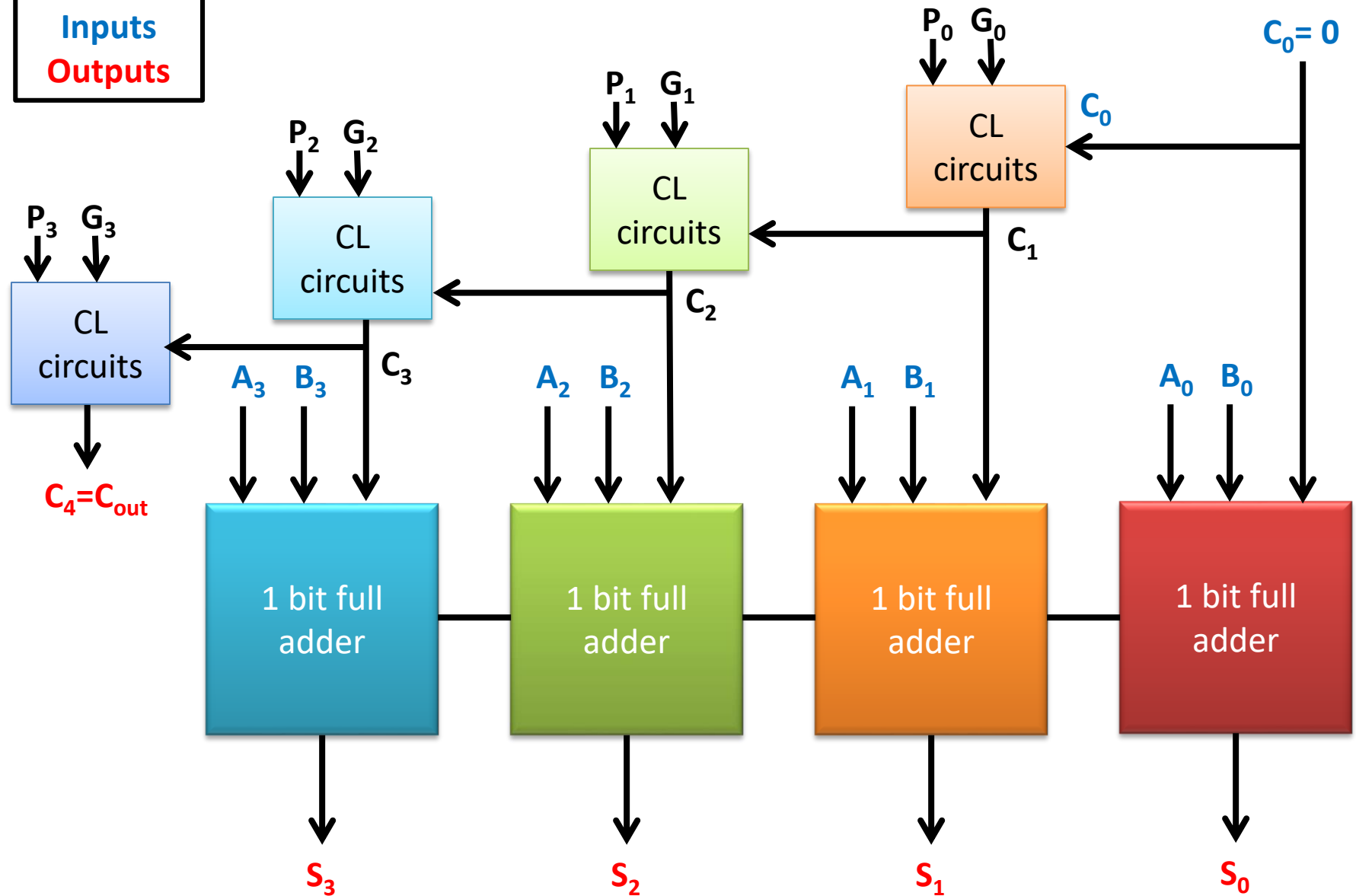
– ***Both propagate and generate signals depend only on the input bits***

- The new expressions for Sums and Carry-outs:

- $S_i = P_i \oplus C_i$
- $C_{i+1} = G_i + P_i C_i$

$S = A \oplus B \oplus C_{in}$ $C_{out} = AB + C_{in}(B \oplus A)$	Recall!
--------------------------------------------------------------------	---------

Inputs
Outputs



$$- S_i = P_i \oplus C_i$$

$$P_i = A_i \oplus B_i$$

$$- C_{i+1} = G_i + P_i C_i$$

$$G_i = A_i B_i$$

• A carry signal will (C_{i+1}) be generated in two cases:

– if both bits A_i and B_i are 1

– if either A_i or B_i is 1 and the carry-in C_i is 1.

• Apply these equations for a 4-bit adder:

$$C_1 = G_0 + P_0 C_0$$

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

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$

$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$$

Note that $P_i = A_i \oplus B_i$ and $G_i = A_i \bullet B_i$

$$C_1 = G_0 + P_0C_0$$

$$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0) = G_1 + P_1G_0 + P_1P_0C_0$$

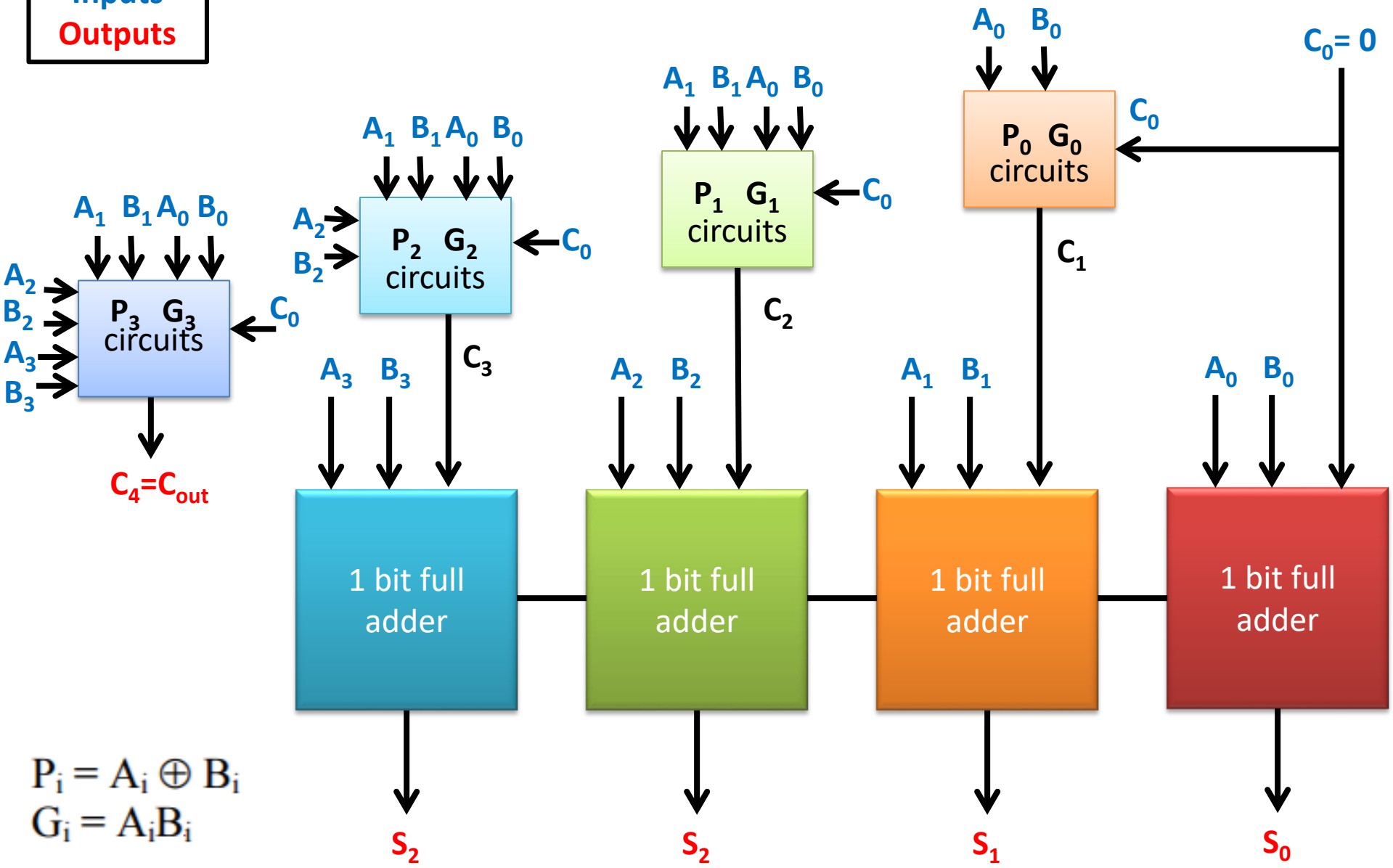
$$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$$

$$C_4 = G_3 + P_3C_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_0$$

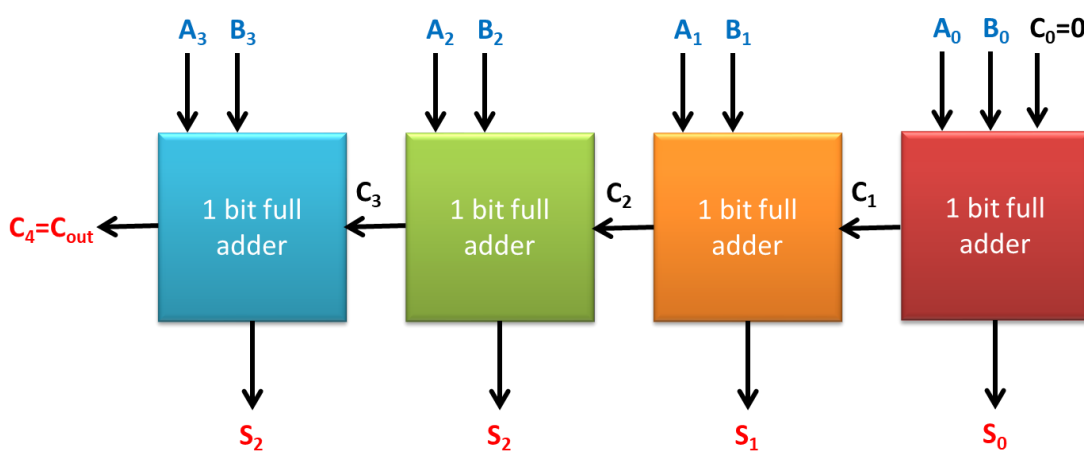
- C_2 , C_3 and C_4 do not depend on its previous carry-in.
- C_4 does not need to wait for C_3 to propagate.
 - As soon as C_0 is computed, C_4 can reach steady state.
 - The same is also true for C_2 and C_3
- The general expression is

$$C_{i+1} = G_i + P_iG_{i-1} + P_iP_{i-1}G_{i-2} + \dots \dots P_iP_{i-1} \dots P_2P_1G_0 + P_iP_{i-1} \dots P_1P_0C_0.$$

Inputs
Outputs

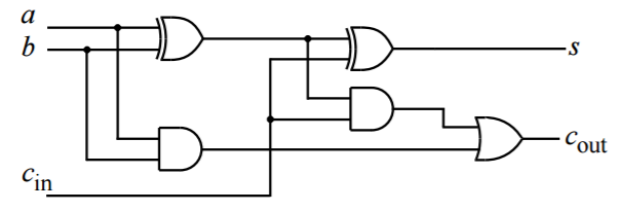


Reduce delay but circuit complexity increases as the circuit size grows.

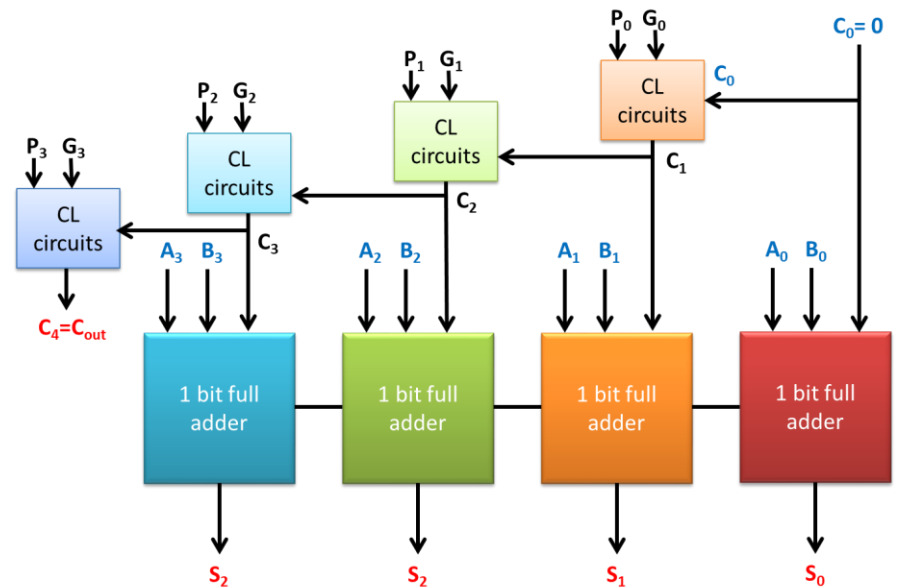
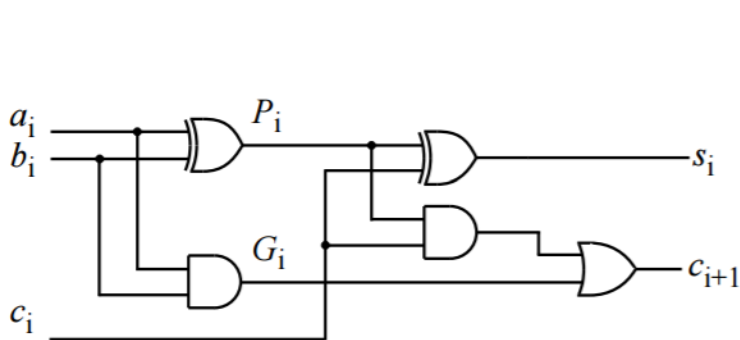


$$S_i = A_i \oplus B_i \oplus C_i$$

$$C_{i+1} = A_i B_i + C_i (A_i \oplus B_i)$$



Delay for carry: $O(n)$



To calculate the carry signals

- No need to wait for the carry to ripple through all the previous stages to find its proper value

Delay for carry: $O(\log(n))$

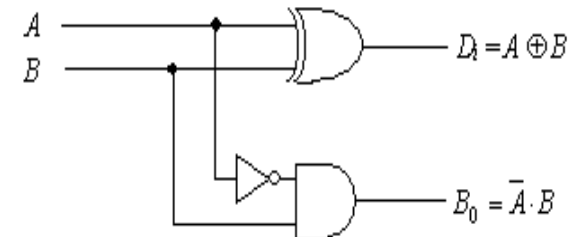
Subtractors

- Half subtractor accepts two binary digits as input (Minuend and Subtrahend, A and B) and produces two outputs, a Difference bit (D_i) and Borrow bit (B_0).

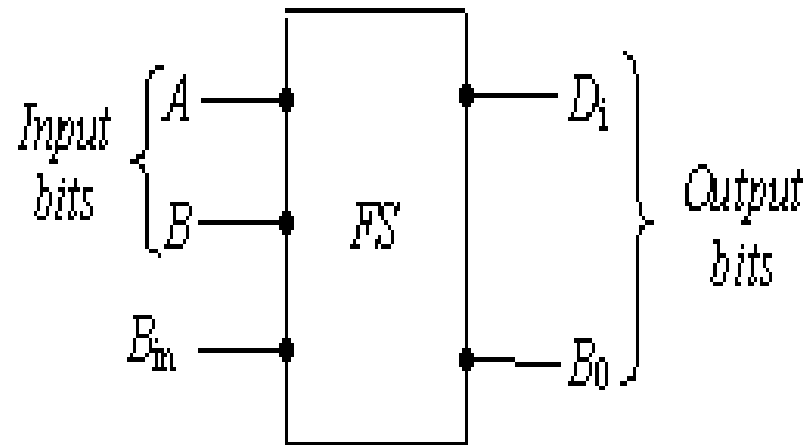
A	B		Difference	Borrow
0	0	=	0	0
0	1	=	1	1
1	0	=	1	0
1	1	=	0	0

$A'B$

AB'



The *full-subtractor* accepts three inputs including a borrow input (B_{in}) and produces a difference output (D_i) and a borrow output (B_o).



Full-Subtractor

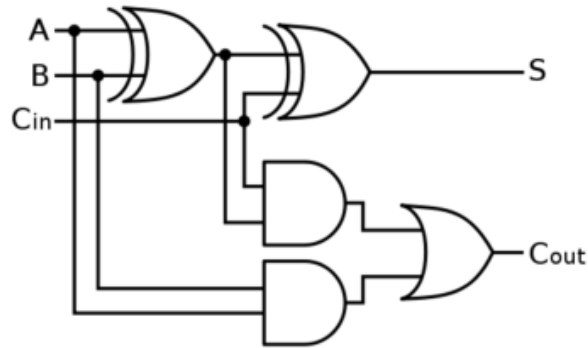
Symbol	Truth Table				
	B-in	Y	X	Diff.	B-out
	0	0	0		
	0	0	1		
	0	1	0		
	0	1	1		
	1	0	0		
	1	0	1		
	1	1	0		
	1	1	1		

$$Diff = A \oplus B \oplus Bin$$

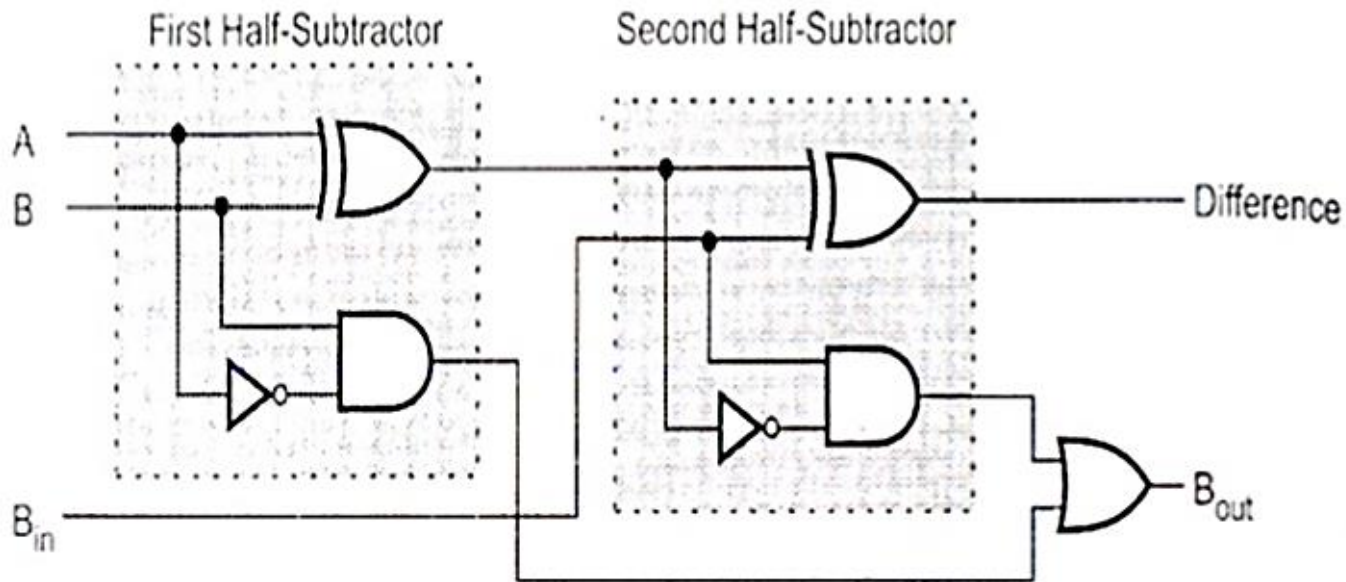
$$B_0 = \overline{A} \cdot B + \overline{A \oplus B} \cdot B_{in}$$

Try it!

Do K-map for verification



A full adder



A full subtractor

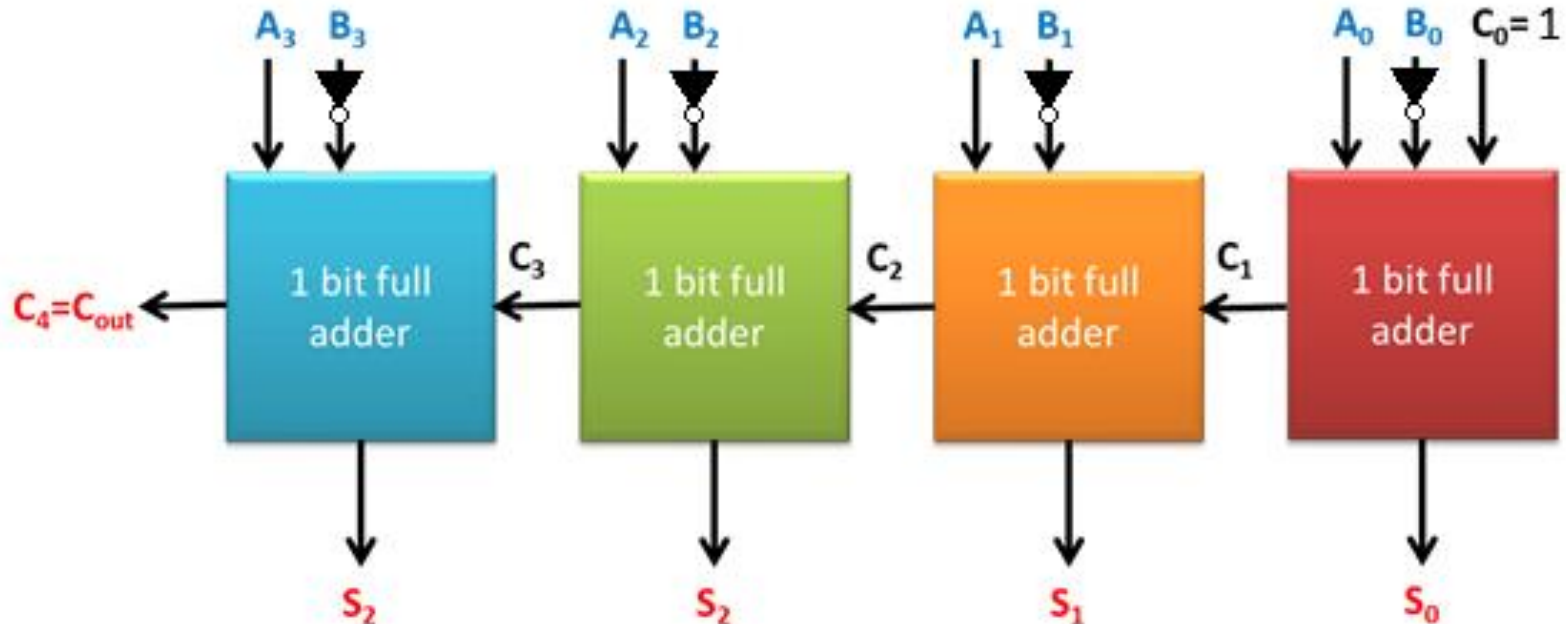
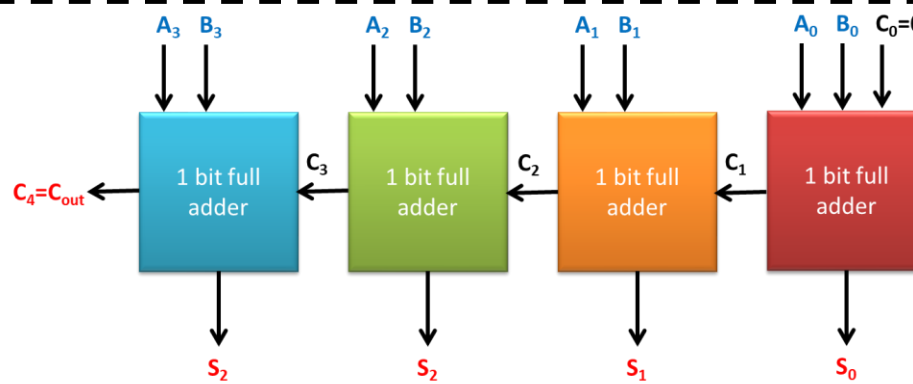
$$Diff = A \oplus B \oplus B_{in}$$

$$B_{out} = A' B + (A \oplus B)' B_{in}$$

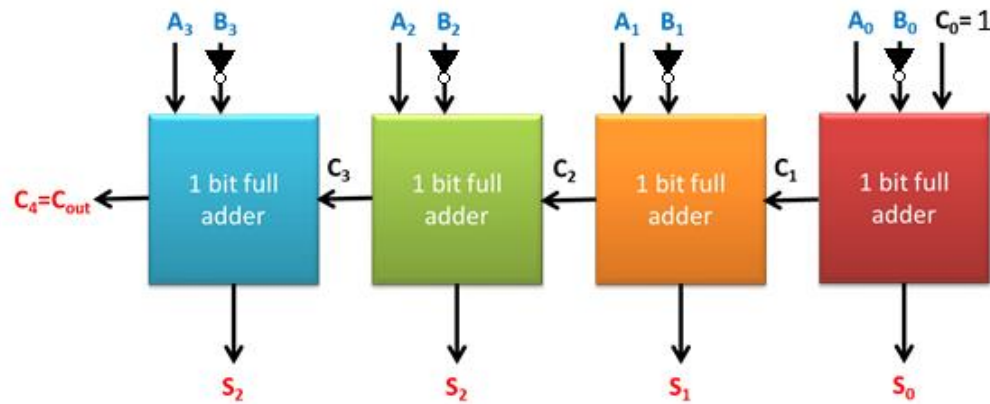
Four Bit Parallel Subtractor using Full Adders

(If you know RC adder and 2's complements this part is a piece of cake)

- Parallel adders can be used to perform binary subtraction because the subtraction is addition in the 2's complement form of binary number.
- In this case, the outputs are result and Carry-out which are more intuitive than the previous case

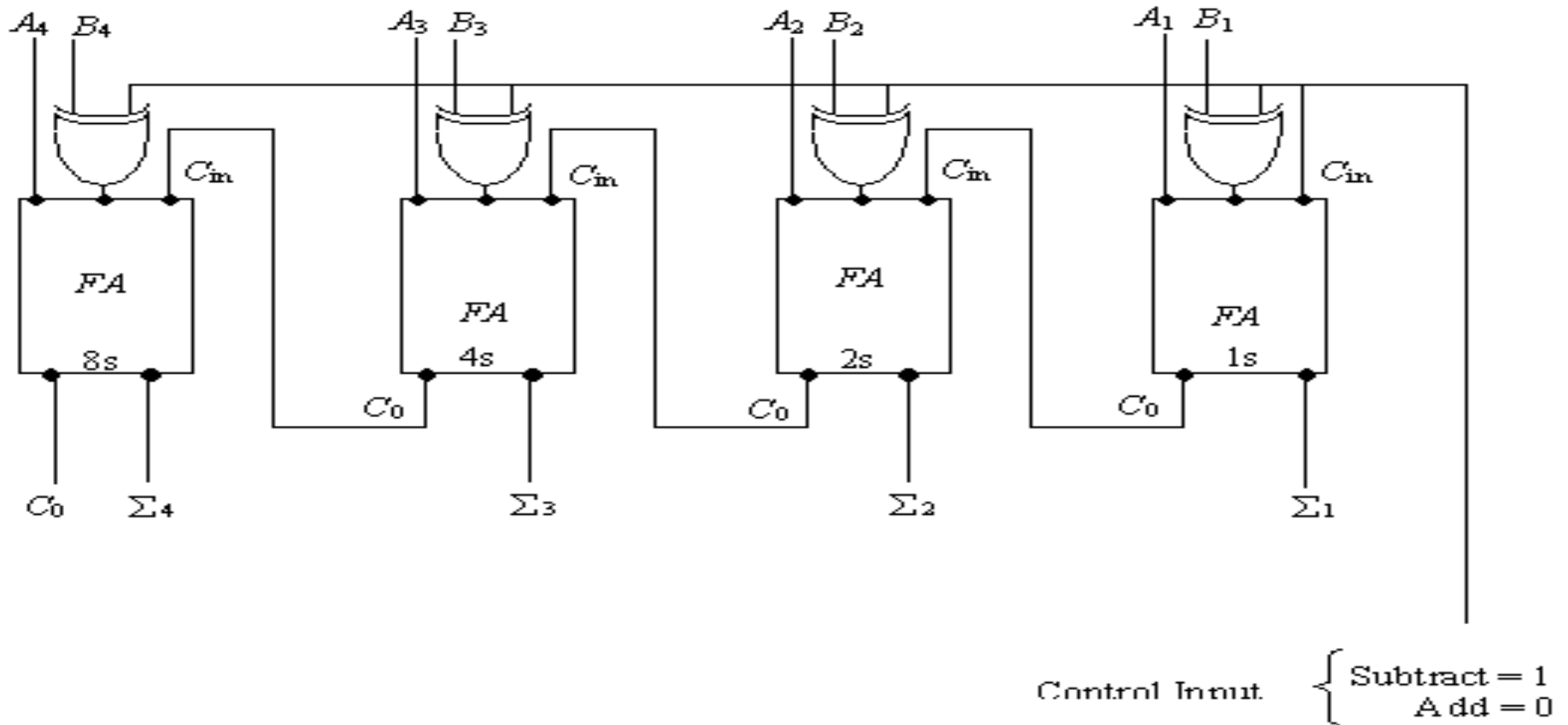


**Ripple-carry adder extension
(2's complement based subtractor)**



- The four inverters invert subtrahends
- The high input at $C_0=1$ LSB makes the binary subtrahend to 2's complement form (+1)
- The C_{out} of fourth Full adder is negation of B_{out} (when if carry bit is 1, borrow bit is 0)

4 bit RC Adder / Subtractor Circuit



Q. Consider a two-bit subtractor, which has four inputs: A_1 , A_0 , B_1 and B_0 .

- This subtractor has three outputs: subtraction result R_1 , R_0 , and carry-out C_{out} .
- This subtractor performs the $A_1, A_0 - B_1, B_0$ binary subtraction.

Show the truth table for the R_1 , R_0 and C_{out} .

Hint, use 2's complement method

A_1	A_0	B_1	B_0	R_1	R_0	C_{out}
0	0	0	0	0	0	1
0	0	0	1	1	1	0
0	0	1	0	1	0	0
0	0	1	1	0	1	0
0	1	0	0	0	1	1
0	1	0	1	0	0	1
0	1	1	0	1	1	0
0	1	1	1	1	0	0
1	0	0	0	1	0	1
1	0	0	1	0	1	1
1	0	1	0	0	0	1
1	0	1	1	1	1	0
1	1	0	0	1	1	1
1	1	0	1	1	0	1
1	1	1	0	0	1	1
1	1	1	1	0	0	1

Try it

Encoder/Decoder


Reference: www.allaboutcircuits.com
<http://www.electronics-tutorials.ws>

# of Input	# of Output	Name of I/O converting system
2^n	n	Encoder
n	2^n	Decoder
More than 1 Data In: 2^n Control In: n	1 (mostly)	Multiplexer
1 (mostly)	More than 1	Demultiplexer

Decoder

- What is a code? What does that mean?

code


/kəʊd/ 

noun

1. a system of words, letters, figures, or symbols used to represent others, especially for the purposes of secrecy.
"the Americans cracked their diplomatic code"
synonyms: [cipher](#), secret language, secret writing, set of symbols, [key](#), hieroglyphics; [More](#)
2. **COMPUTING**
program instructions.
"assembly code"

Then how about decode?

decode

/di:'kəʊd/ 

verb

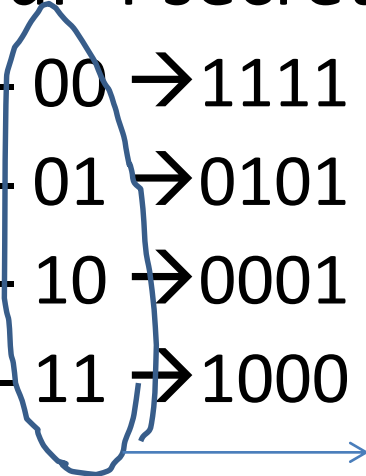
1. convert (a coded message) into intelligible language.

"he put down the phone and decoded the message"

synonyms: [decipher](#), [decrypt](#), [unravel](#), [untangle](#), [work out](#), [sort out](#), [piece together](#), [solve](#),
[interpret](#), [translate](#), [construe](#), [explain](#), [understand](#), [comprehend](#), [apprehend](#), [grasp](#);
[More](#)

Example of decoder

- Assuming that we have 2 bit decoder, we can generate 4 secret codes
- Assuming that students has following IDs
 - Haseem:15, Aina:7, Rachel:5, Afrina: 1
- Our 4 secret codes indicate following IDs



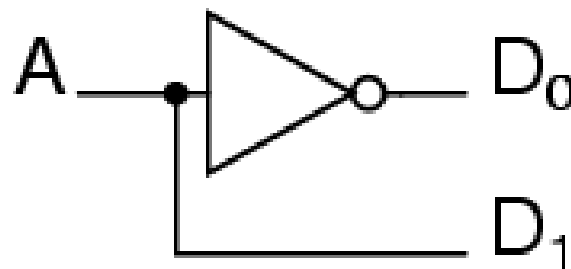
- 00 → 1111
- 01 → 0101
- 10 → 0001
- 11 → 1000

We decode this secret numbers to a proper ID by means of a decoder

- A decoder is a circuit that changes a code into a set of signals.
- A common example: a line decoder
 - 1-to-2 line decoder.

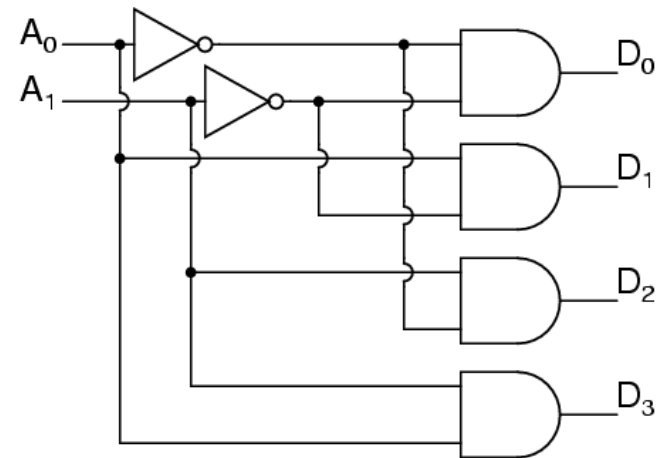
A	D ₁	D ₀
0	0	1
1	1	0

- A is the address and D is the dataline.
- D₀ is NOT A and D₁ is A.



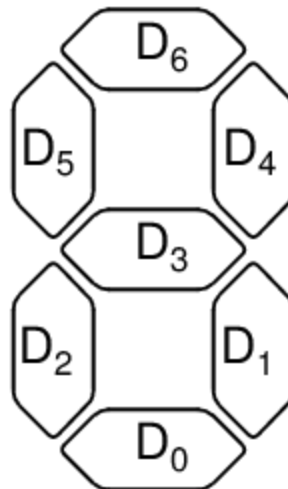
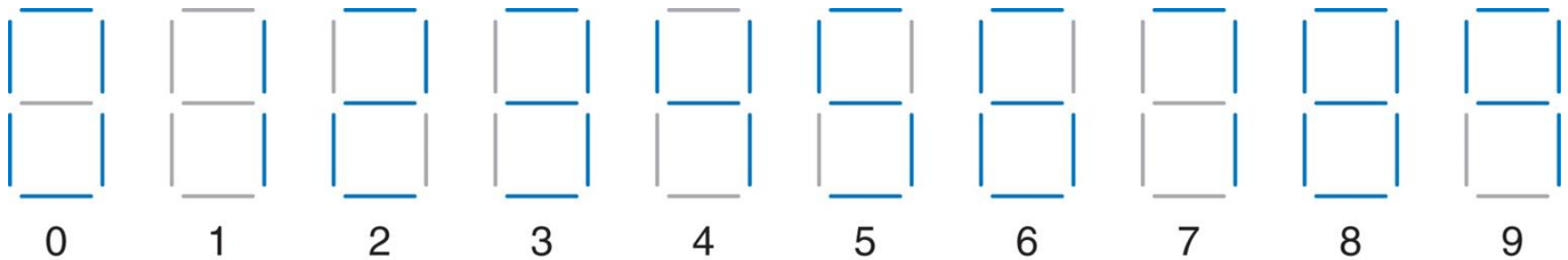
The 2-to-4 line decoder

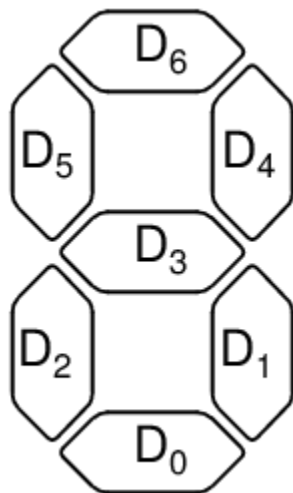
A_1	A_0	D_3	D_2	D_1	D_0
0	0				
0	1				
1	0				
1	1				



- A typical application of a line decoder circuit is to select among multiple devices.

A **d**ecoder application: a binary to 7-segment **d**ecoder.





I_3	I_2	I_1	I_0	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0							
0	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							
1	0	0	0							
1	0	0	1							

What about the remaining six entries of the truth table??

Can you make simplified Bool eq for each segment (D0-D6)?

- i.e., D0 and D1

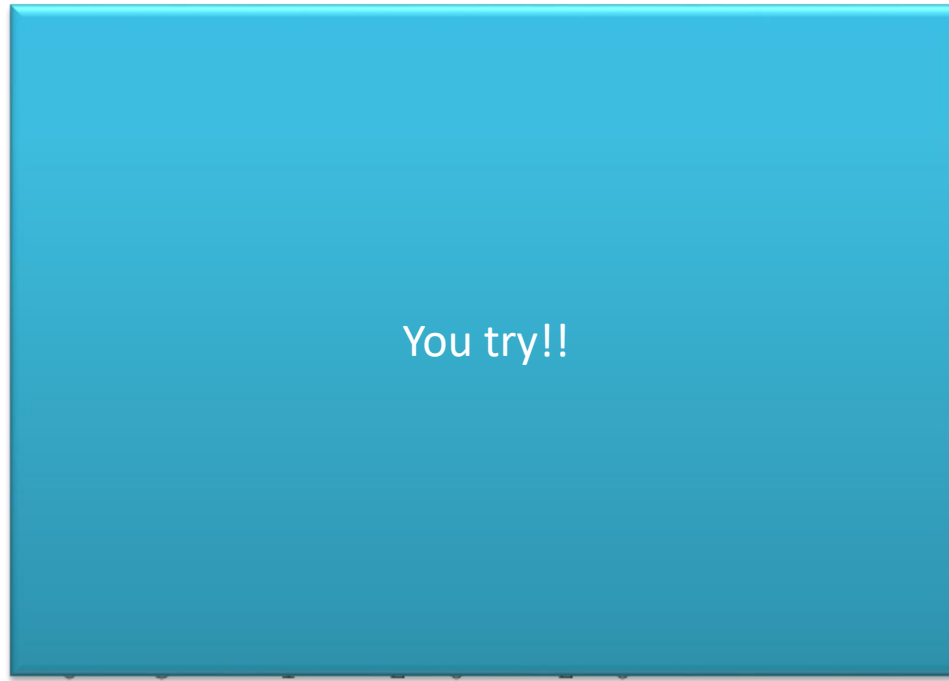
$$D_0 = I_3 + \bar{I}_2 I_1 + \bar{I}_2 \bar{I}_0 + I_1 \bar{I}_0 + I_2 \bar{I}_1 I_0$$

$I_3 \backslash I_2 I_1 I_0$	00	01	11	10
00	1	0	1	1
01	0	1	0	1
11	*	*	*	*
10	1	1	*	*

$$D_1 = I_3 + I_2 + \bar{I}_1 + I_0$$

$I_3 \backslash I_2 I_1 I_0$	00	01	11	10
00	1	1	1	0
01	1	1	1	1
11	*	*	*	*
10	1	1	*	*

- The collection of equations is summarized



Encoder

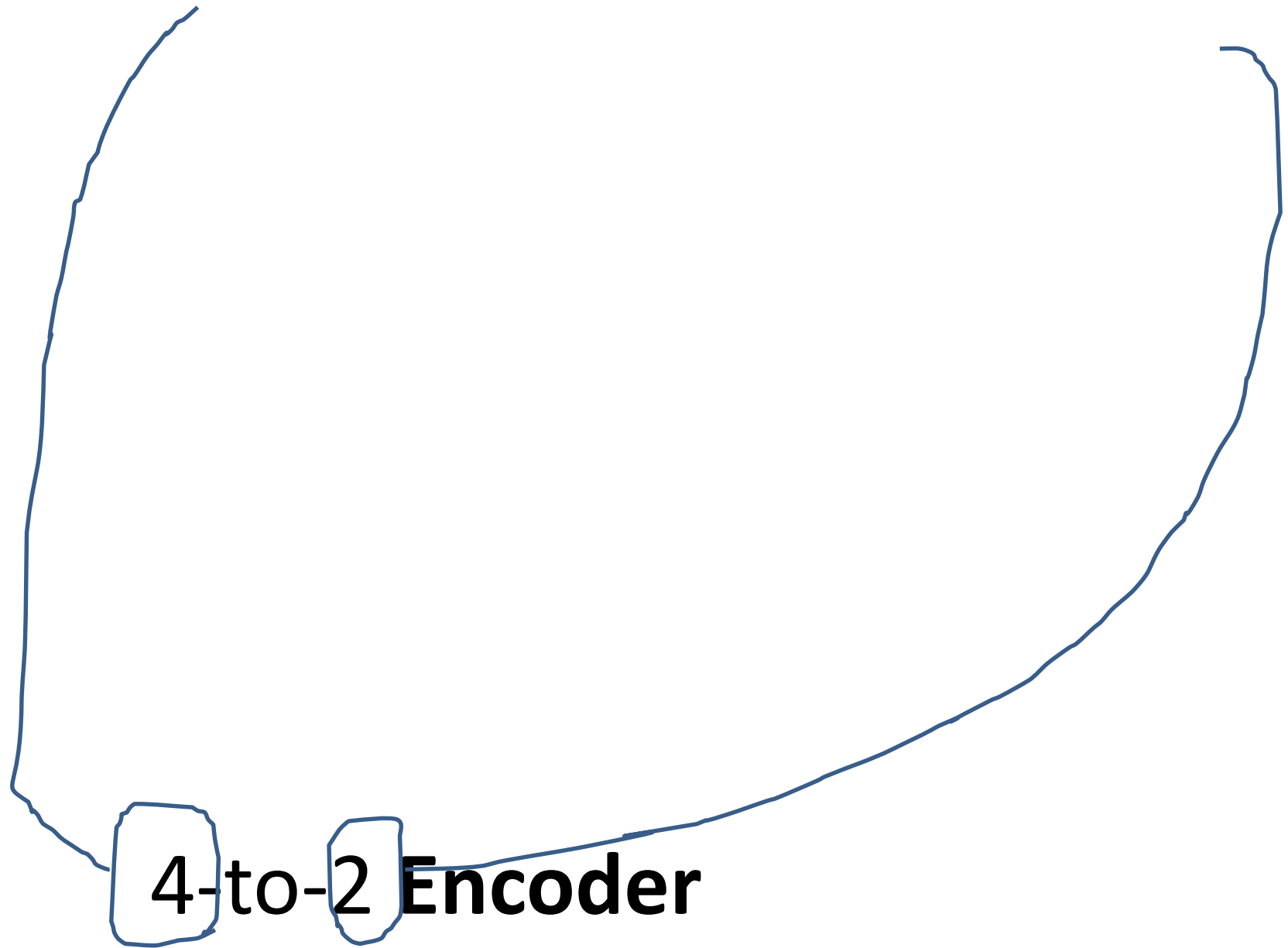
About 39,600,000 results (0.65 seconds)

An **encoder** is a device, circuit, transducer, software program, algorithm or person that converts information from one format or code to another, for the purposes of standardization, speed or compressions.

[Encoder - Wikipedia](https://en.wikipedia.org/wiki/Encoder)

<https://en.wikipedia.org/wiki/Encoder>

- An encoder is a circuit that changes a set of signals into a code.



2-to-1 line encoder truth table

- By reversing the 1-to-2 decoder truth table.

D_1	D_0	A
0	1	0
1	0	1

- A complete truth table would be

D_1	D_0	A
0	0	
0	1	0
1	0	1
1	1	

One question we need to answer is what to do with those other inputs?