UMBC

Adders and other circuits

CMSC 313
Raphael Elspas

1-bit Adder logic

- Let's build a circuit which adds 2 bits: "a" & "b" together.
- We need two bits to store the output, so we'll use a sum bit "s" and a carry bit "c".

a	b	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$c = ab$$
$$s = a\bar{b} + \bar{a}b = a \oplus b$$

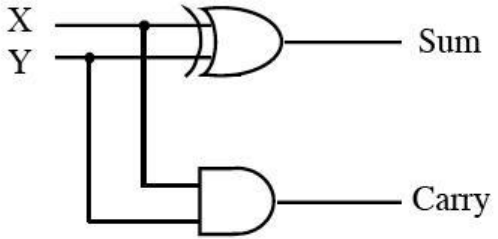
1-bit Adder circuits

- Input: a, b. Output: c, s

a	b	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

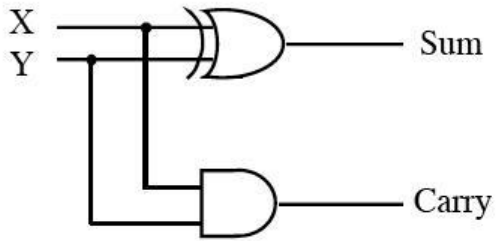
$c = ab$

$s = a\bar{b} + \bar{a}b = a \oplus b$

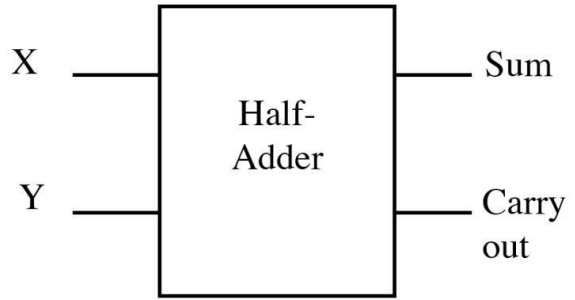


Abstract it

- Let’s put it in a box called a **block diagram** and call it a half-adder
- **Half-Adder** (HA) definition: adds two 1-bit values and produces a sum bit and a carry bit.
- A full adder should be able to have a carry in, not just a carry out



Half-Adder Using XOR and AND



Full adder motivation

- We want to attach multiple 1-bit adders together to create many-bit adders
- To do this we need a carry-in bit that the previous block's carry-out bit would feed into.

Full Adder

- A Full Adder (FA) has 3 inputs: two 1-bit values and a carry in, and 2 outputs: a sum bit and a carry bit.
- That means we need to add three 1-bit values

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

Full Adder

- Algebraically simplify
- Lets try to get similar statements to the Half adder to see if we can repurpose a half adder box.

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

$$S = \overline{A}\,\overline{B}\,C_{in} + \overline{A}\,B\,\overline{C_{in}} + A\,\overline{B}\,\overline{C_{in}} + A\,B\,C_{in}$$

$$= \overline{A}\,(\overline{B}\,C_{in} + B\,\overline{C_{in}}) + A\,(\overline{B}\,\overline{C_{in}} + B\,C_{in})$$

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

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

$$C_{out} = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

$$= C(\overline{A}B + A\overline{B}) + AB(\overline{C} + C)$$

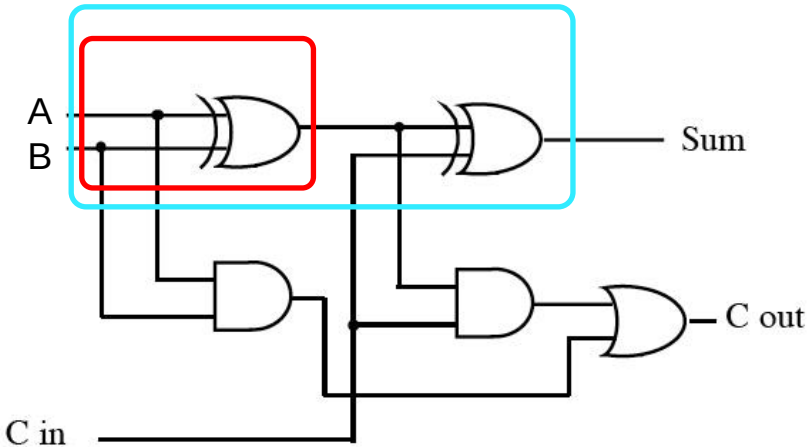
$$C_{out} = C(A \oplus B) + AB$$

Full adder realized circuit

- Convert to a circuit

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

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

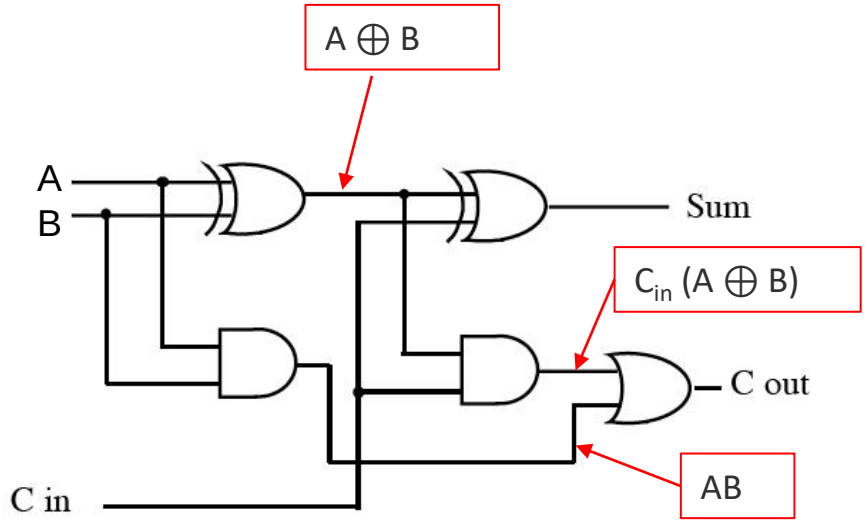


Full adder realized circuit

- Convert to a circuit

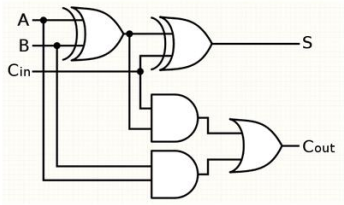
$S = A \oplus B \oplus C_{in}$

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

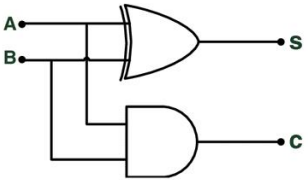


thanos

Full adder

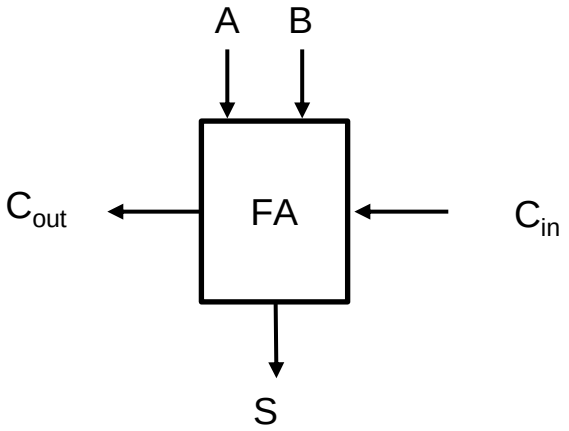


Half adder



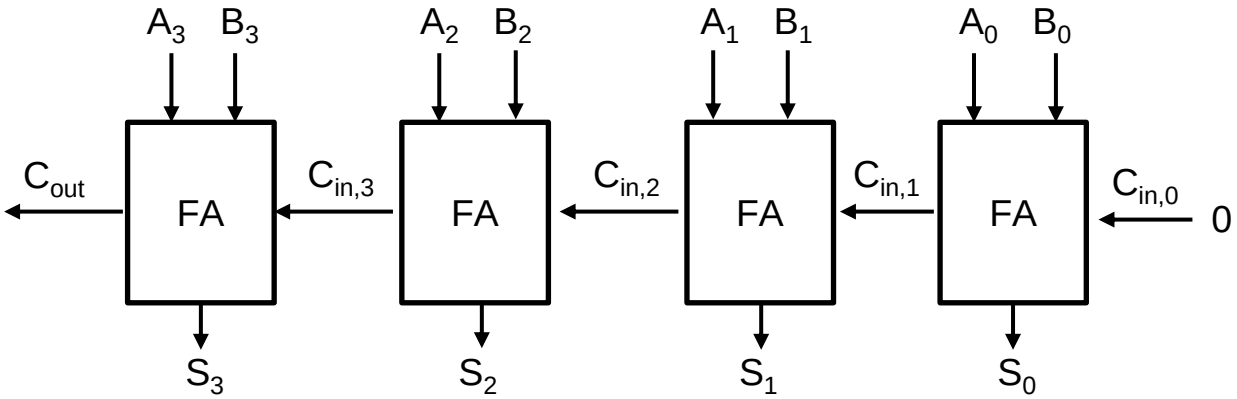
Full Adder

- 3 Inputs: A, B, Carry in
- 2 outputs: Sum, Carry out
- Put it in a box!



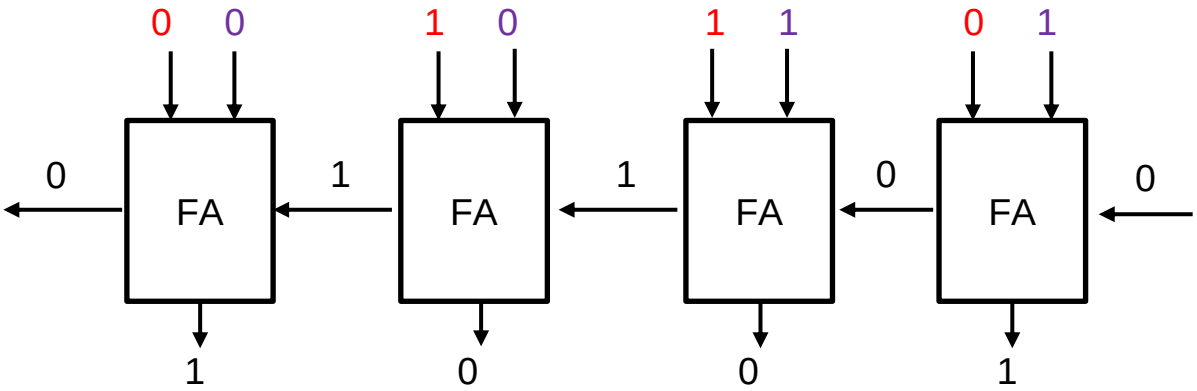
Ripple Carry Adder

- Chain together Full Adders to add multiple bits
- $C_{in,0}$ is set to 0, since nothing is being carried in



Ripple Carry Adder example

- $A_3A_2A_1A_0$ = 0110 (6)
- $B_3B_2B_1B_0$ = 0011 (3)



Result: 1001, Carry 0

Check: 6+3 = 9

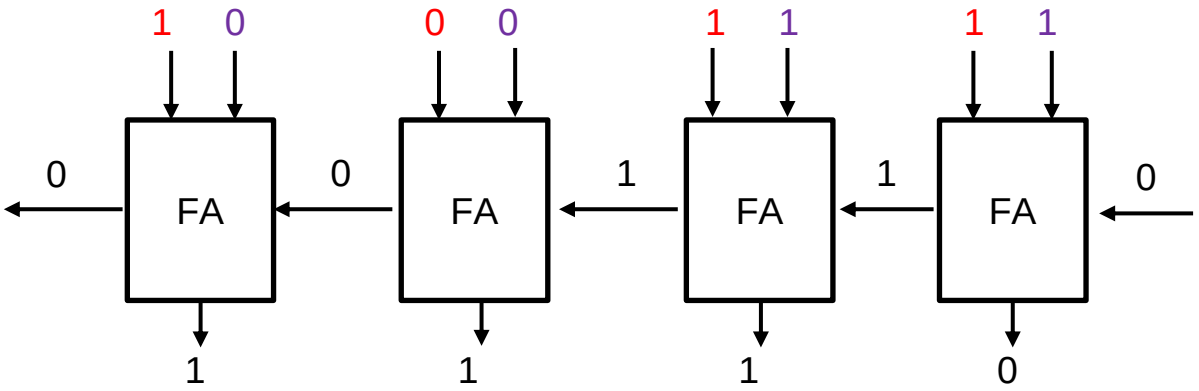
Ripple Carry Adder - 2s complement

- Does this module work for 2s complement?
- Yes! Because 2s complement uses regular addition
- But the number has to already be in 2s complement form
- Lets try -5 + 3

Ripple Carry Adder - 2s complement example

(5) 0101 → (inv) → 1010 → (+1) → 1011

- $A_3A_2A_1A_0$ = 1011 (-5)
- $B_3B_2B_1B_0$ = 0011 (3)

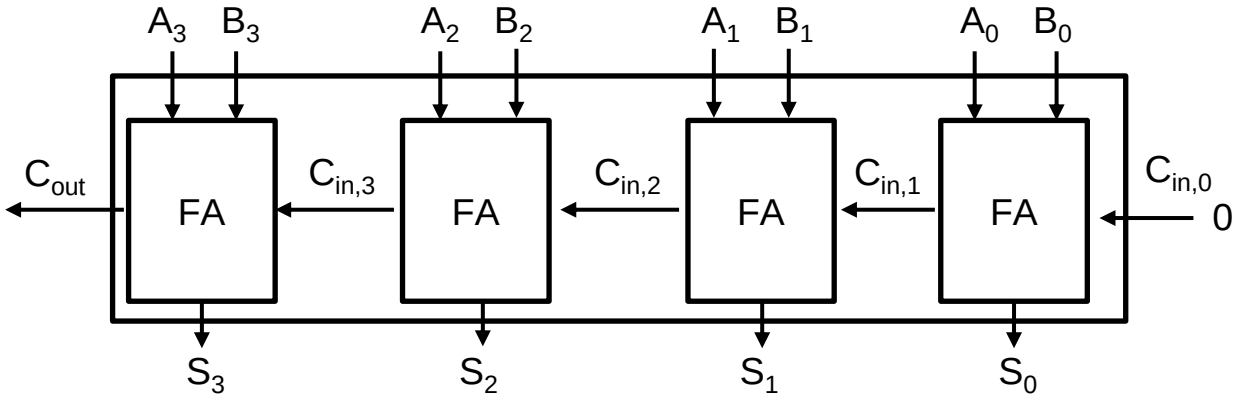


Result: 1110, Carry 0

Check: 1110 (2s complement)
1101 (-1)
0010 (invert bits)
= -2

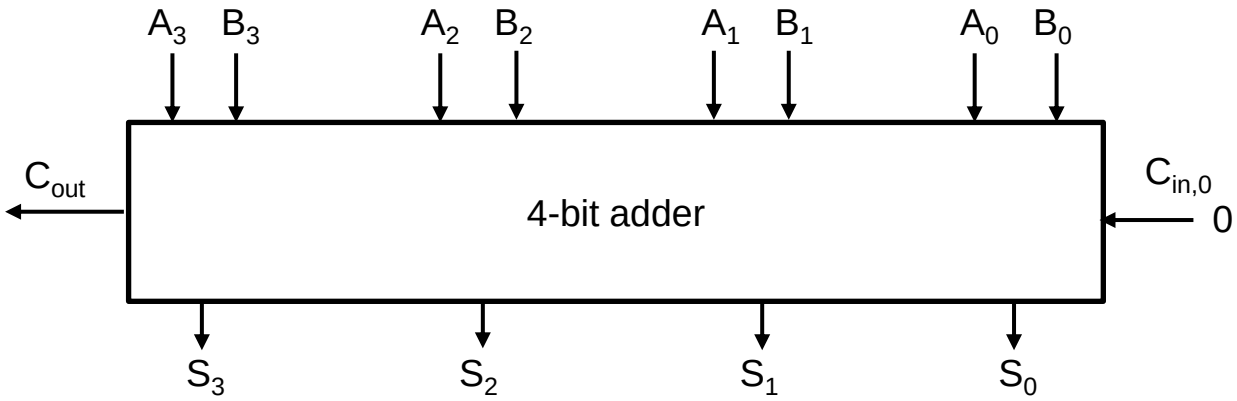
It works!

- Let's put it in a box

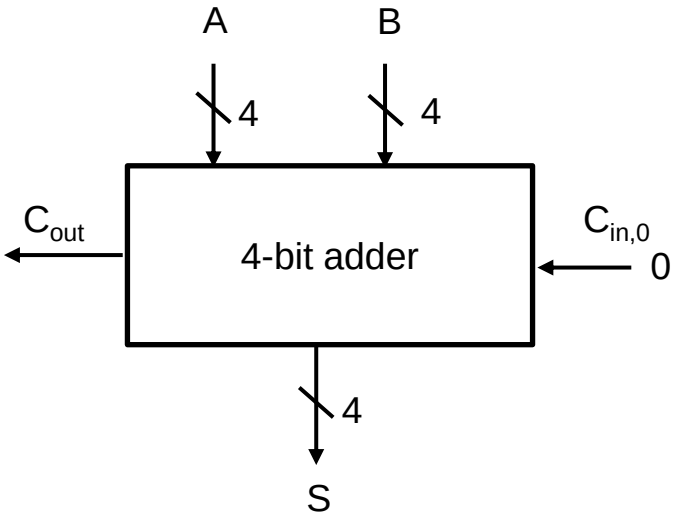


- $C_{in,1}$-$C_{in,3}$ are internal variables, so they disappear

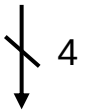
4-bit adder



4-bit adder another representation

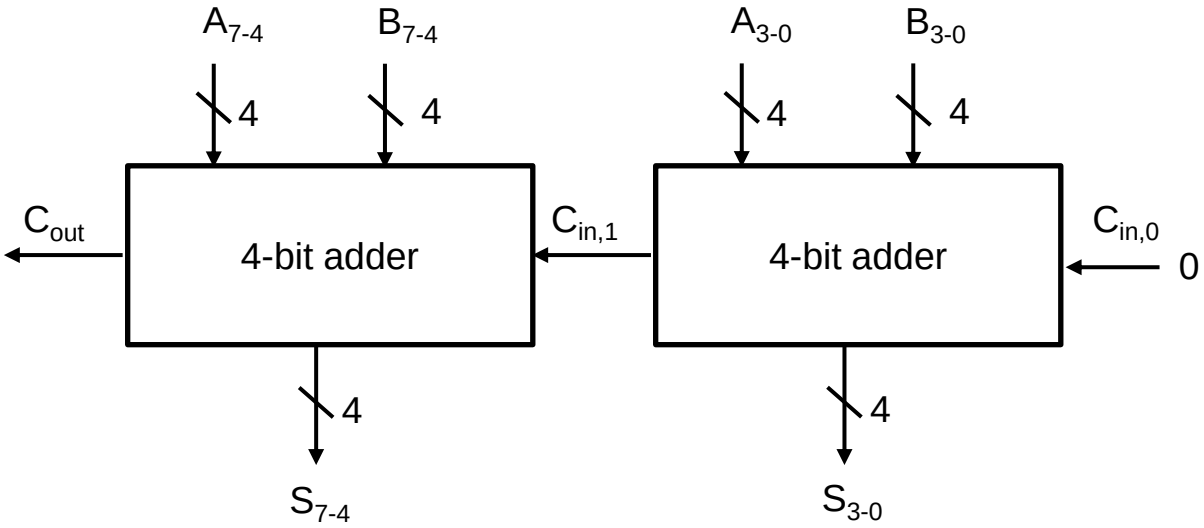


Note



Means there are four wires here– called a “bus”

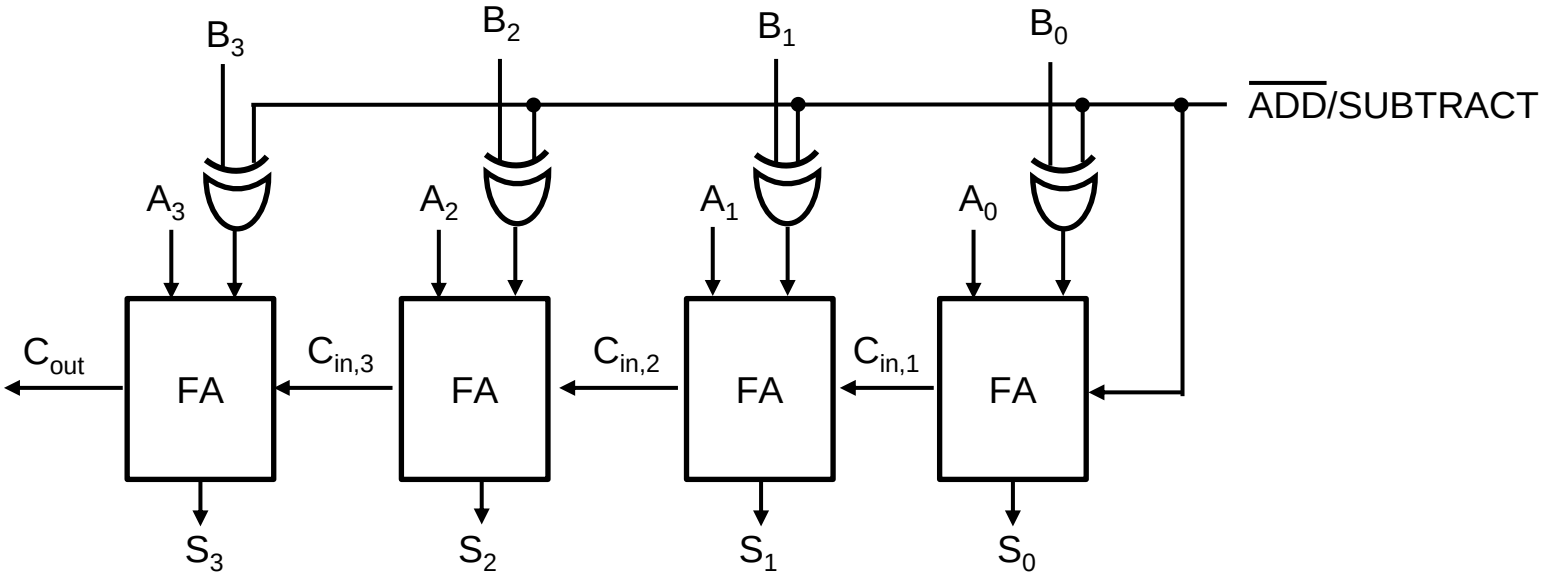
8-bit adder from 4 bit adder



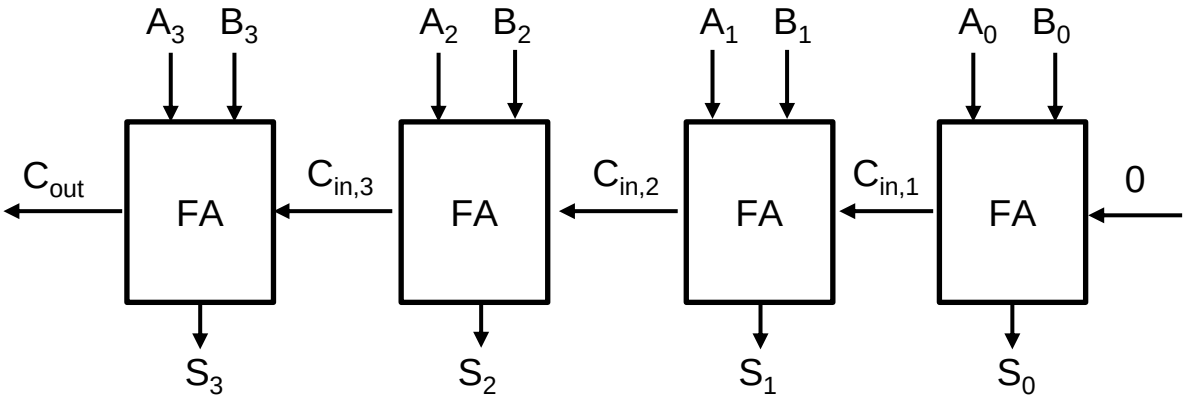
Adder-Subtractor

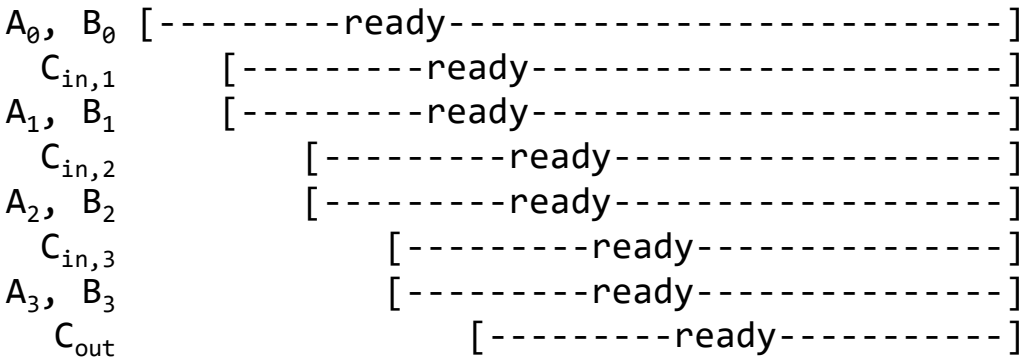
- In order to make a subtractor, we have to make a 2's complement converter.
- Then we'll reuse the adding circuit as before
- In order to do A – B, we'll convert the B to 2's complement – but we also want to be able to add A + B. We need a switch to turn subtracting on and off.
- In subtractor mode, we need to invert the bits in B and add 1
  - We can make a controllable inverter with an XOR gate
  - We can add 1 with the Carry-in to the first FA.

Adder-Subtractor implementation



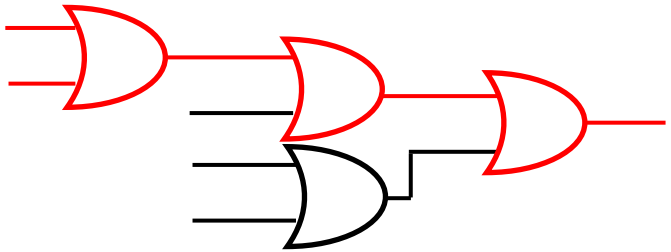
Ripple Carry adder delay





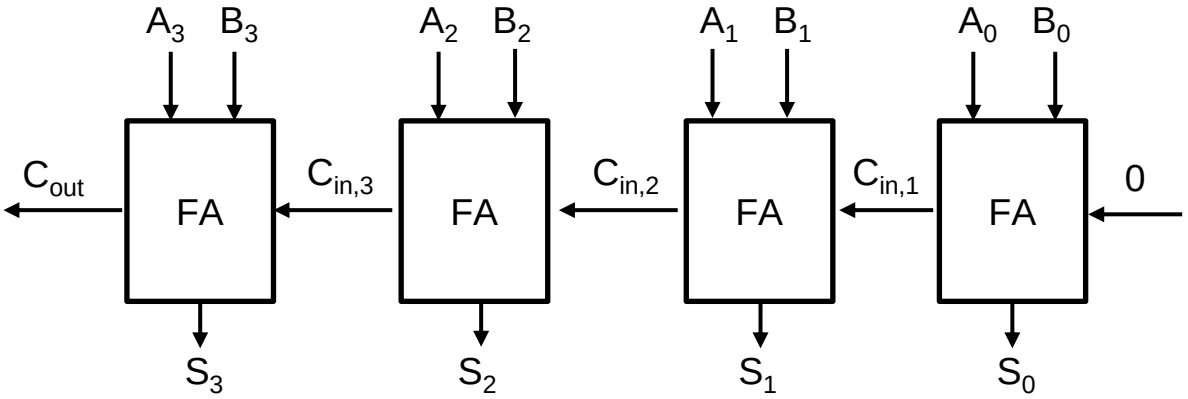
Gate Delay

- Every gate incurs some delay as information passes through.
- We want to minimize circuit delay so that circuits are faster
- We measure worst delay from the **critical path**. This is the longest path from input to output in terms of number of gates traversed.

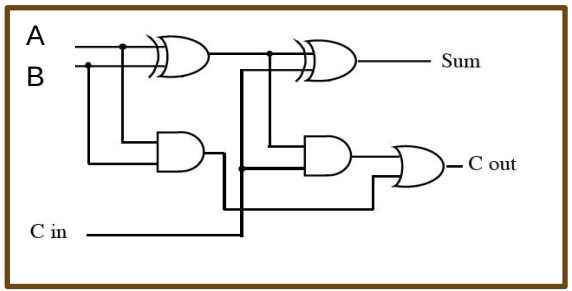


The two critical paths pass through 3 OR gates

Ripple Carry adder delay

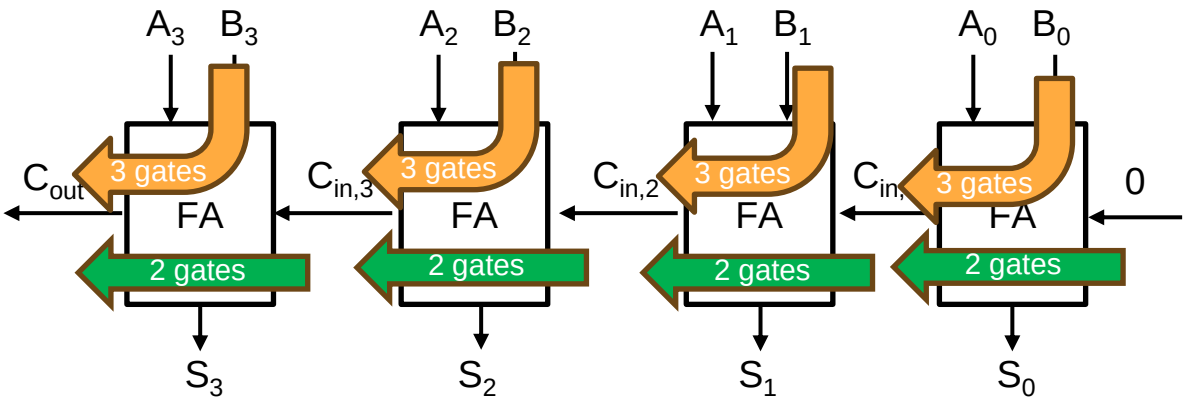


Full Adder



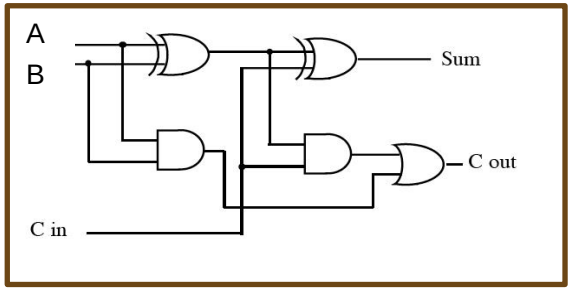
Critical Paths for Full Adder:
A,B → Cout : 3 gates
Cin → Cout: 2 gates
A,B → Sum: 2 gates
Cin → Sum: 1 gate

Ripple Carry adder delay



All paths displayed

Full Adder



Paths for Full Adder:

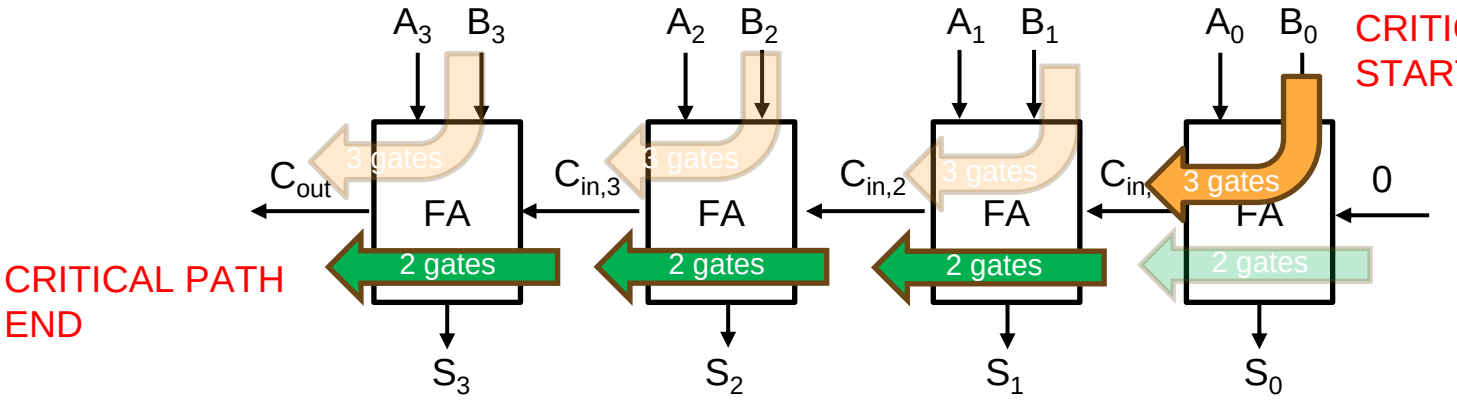
A,B → Cout : 3 gates

Cin → Cout: 2 gates

A,B → Sum: 2 gates

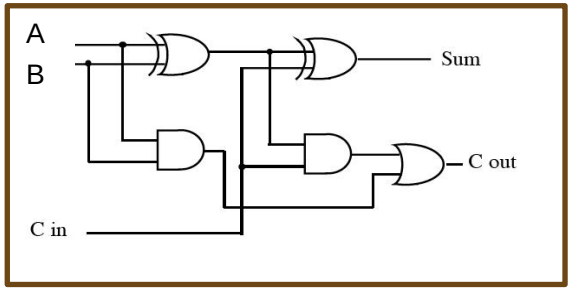
Cin→ Sum: 1 gate

Ripple Carry adder delay



Critical Path:
9 gates

Full Adder



Paths for Full Adder:

A,B → Cout : 3 gates

Cin → Cout: 2 gates

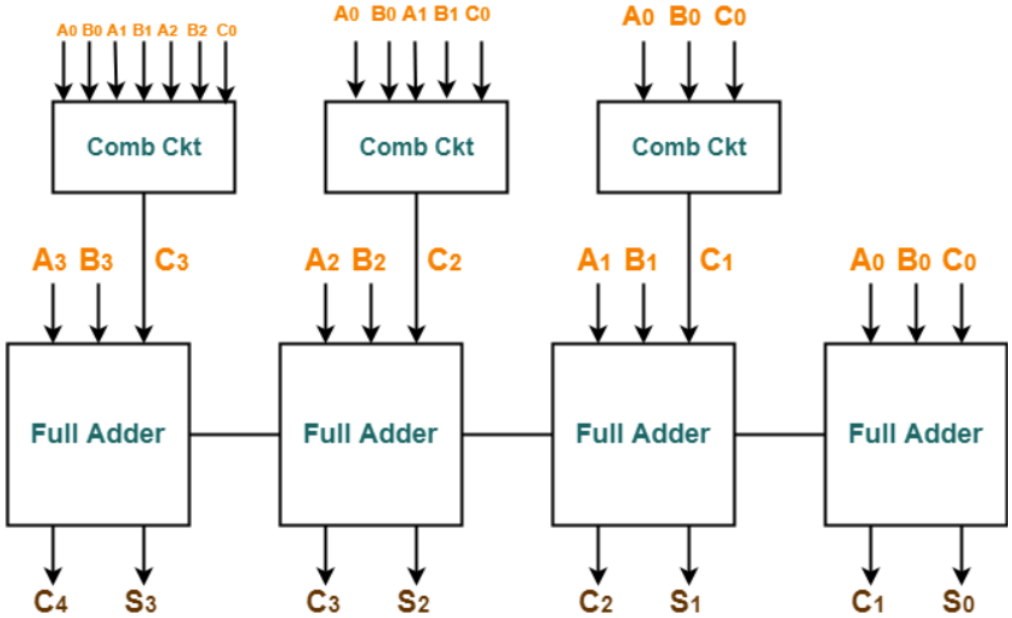
A,B → Sum: 2 gates

Cin→ Sum: 1 gate

Idea: predict carry in advance

- Predict carry-in value in advance so numbers can be added in parallel
- $C_{in,i}$ for Full adder i relies on all input bits before it including A_0 through A_{i-1}, B_0 through B_{i-1}, and $C_{in,0}$ through $C_{in,i-1}$.
- All Carry in values are dependent on either A or B, so we can leave them out as part of the predictor logic.

Carry lookahead adder idea

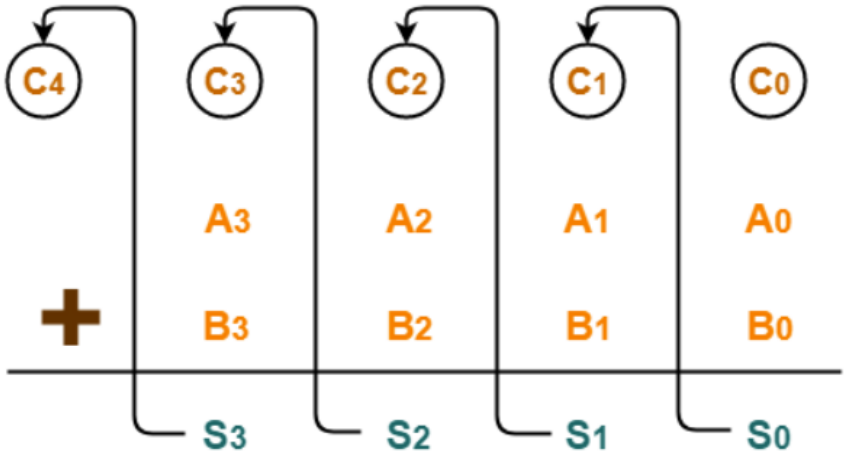


Calculating each carry

- Remember how we calculate carry out in the Full Adder

  $C_{out} = C_{in} (A \oplus B) + AB$
- We can find each of the carry bits this way, and create expressions that are dependent on previous values

- $C_1 = C_0 (A_0 \oplus B_0) + A_0B_0$
- $C_2 = C_1 (A_1 \oplus B_1) + A_1B_1$
- $C_3 = C_2 (A_2 \oplus B_2) + A_2B_2$
- $C_4 = C_3 (A_3 \oplus B_3) + A_3B_3$



Carry algebra

For simplicity, Let-

- $G_i = A_iB_i$ where G is called carry generator
- $P_i = A_i \oplus B_i$ where P is called carry propagator

Therefore, we'll rewrite each $C_{i+1} = C_i (A_i \oplus B_i) + A_iB_i$ as

- $C_1 = C_0P_0 + G_0$
- $C_2 = C_1P_1 + G_1$
- $C_3 = C_2P_2 + G_2$
- $C_4 = C_3P_3 + G_3$

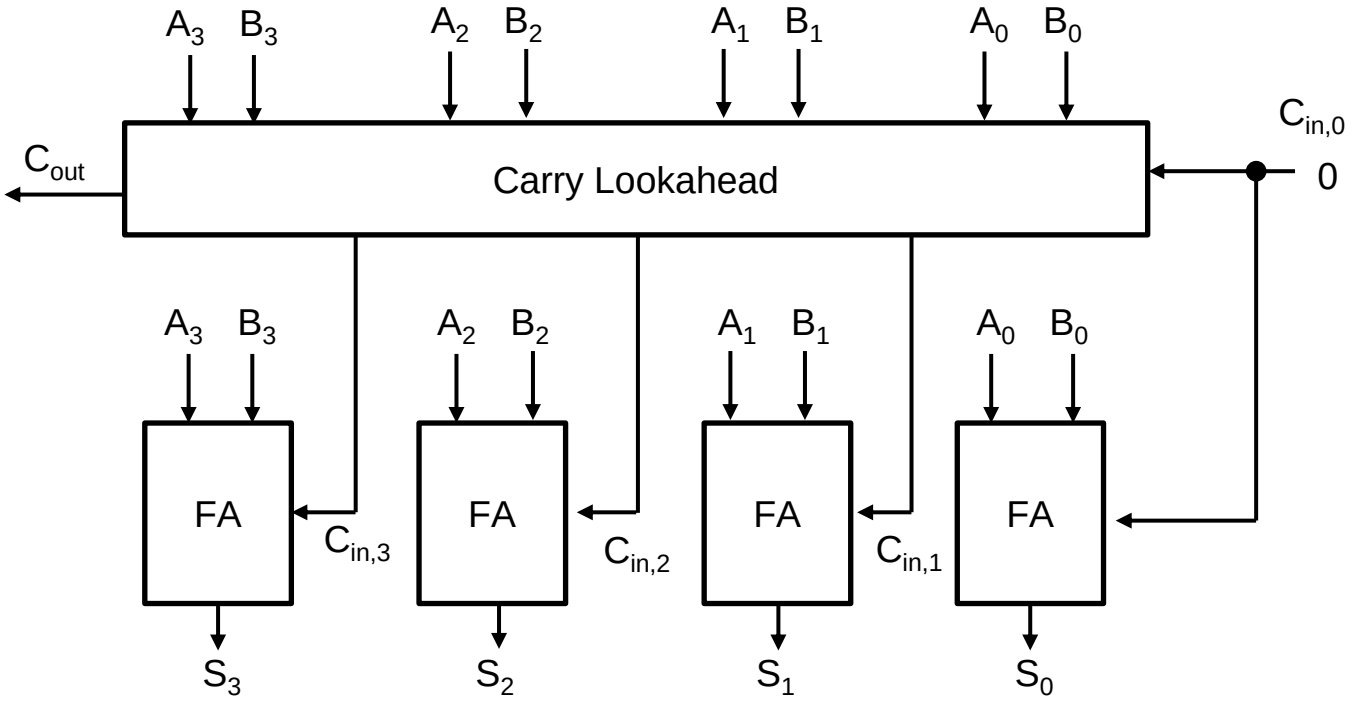
Carry algebra

- C_1, C_2 and C_3 are intermediate carry bits.
- So, let's remove C_1, C_2 and C_3 from RHS of every equation.
- Substituting $C_1 = C_0P_0 + G_0$ in for $C_2 = C_1P_1 + G_1$ we get C_2 in terms of C_0.
- We can continue substituting so we get C_3 in terms of C_0 and so on.

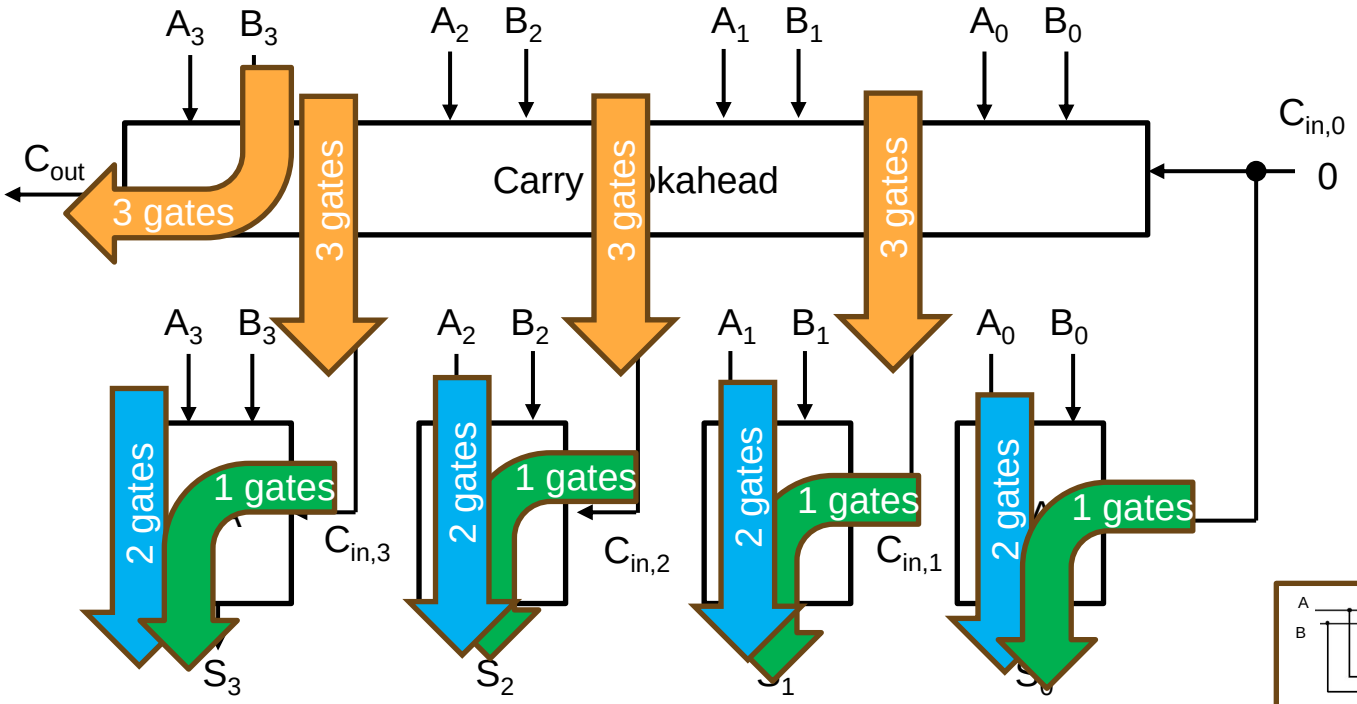
Therefore we'll get

- $C_1 = C_0P_0 + G_0$ ← 3 gates delay
- $C_2 = C_0P_0P_1 + G_0P_1 + G_1$ ← 3 gates delay
- $C_3 = C_0P_0P_1P_2 + G_0P_1P_2 + G_1P_2 + G_2$ ← 3 gates delay
- $C_4 = C_0P_0P_1P_2P_3 + G_0P_1P_2P_3 + G_1P_2P_3 + G_2P_3 + G_3$ ← 3 gates delay

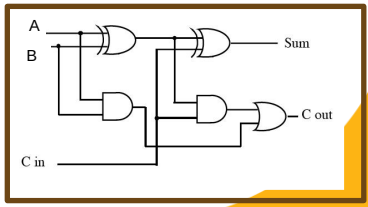
Carry Lookahead adder



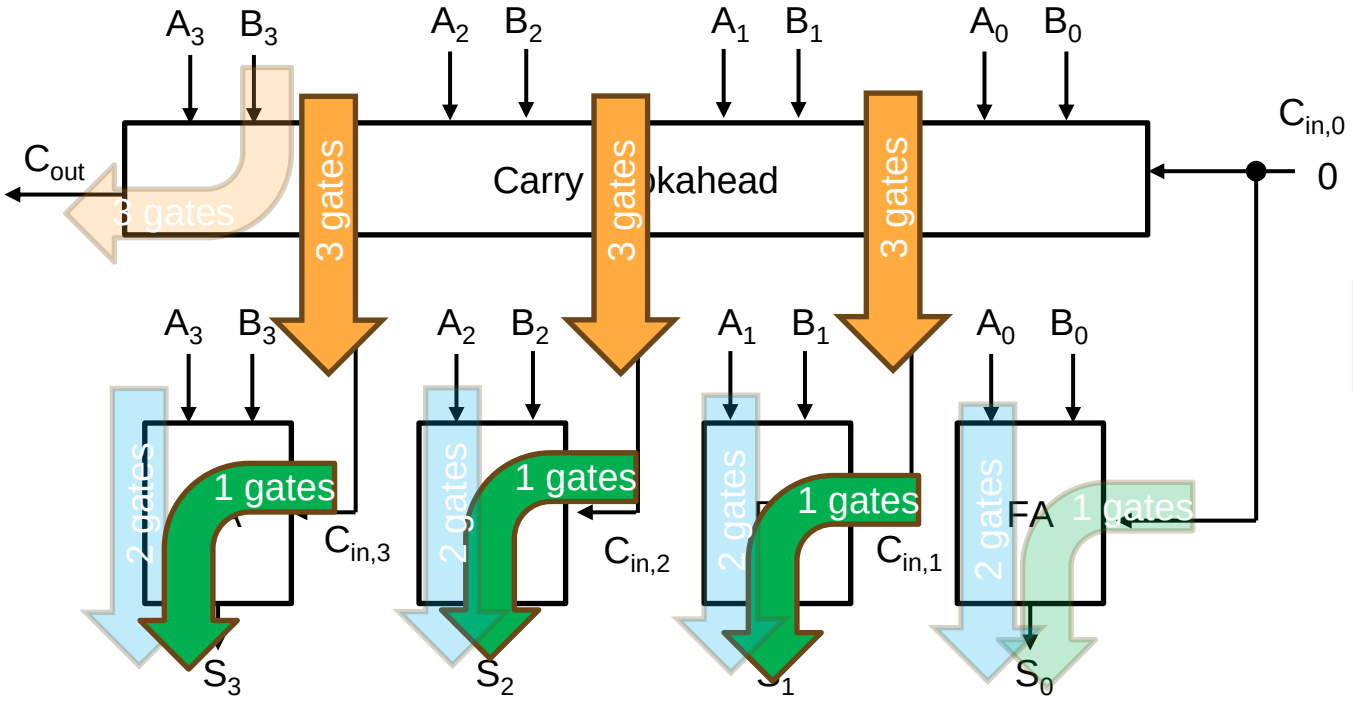
Carry Lookahead adder



Full Adder



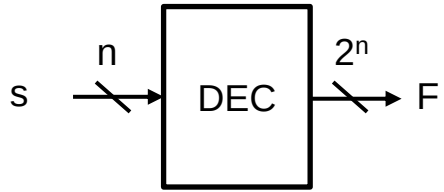
Carry Lookahead adder



Critical Path:
4 gates

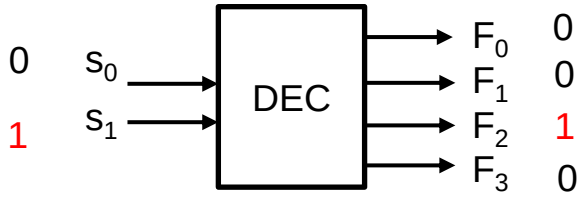
Decoder (DEC)

- A Decoder is a circuit that has n inputs and 2^n outputs.
- A decoder converts a binary number $s = s_{n-1}s_{n-2}\ldots s_1s_0$, and produces a “1” on the decoded line F_s, and “0” on all other F output lines.
- Decoders are a combinational circuit that appear frequently in computer hardware and ALUs

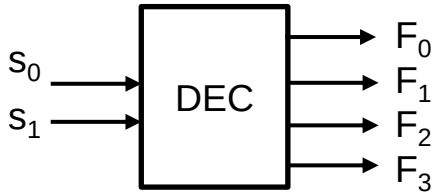


Decoder example

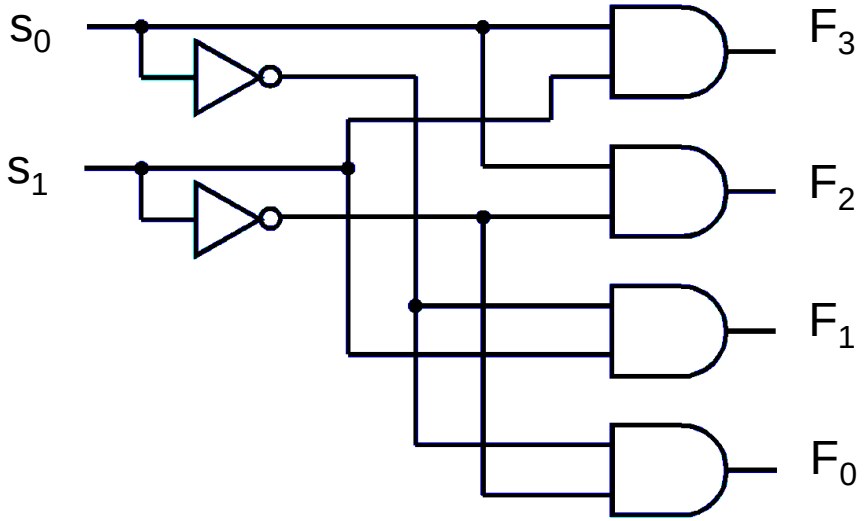
If s_1s_0 equals 10, (or 2 in decimal), then F_2 will equal 1 and the other Fs equal 0.



2-bit Decoder (DEC)

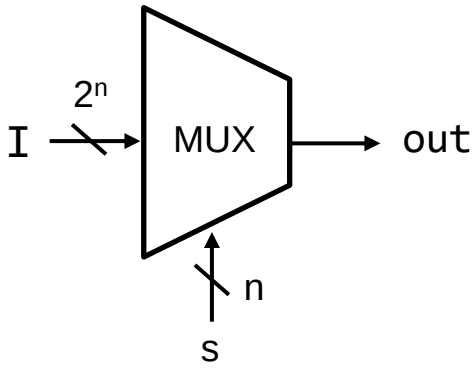


s_0	s_1	F_0	F_1	F_2	F_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



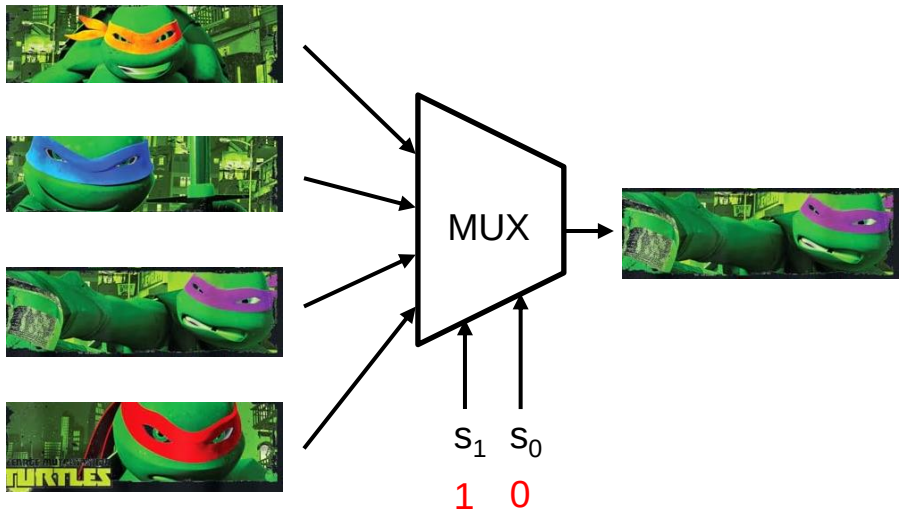
Multiplexer (MUX)

- A multiplexer (MUX) has n select bits “s” which allow you to select one of the 2^n input lines “`I`” to duplicate on the output line “out”
- Basically allows different inputs to share the same wire as an output
- Use a trapezoid to represent MUX



4-input MUX example

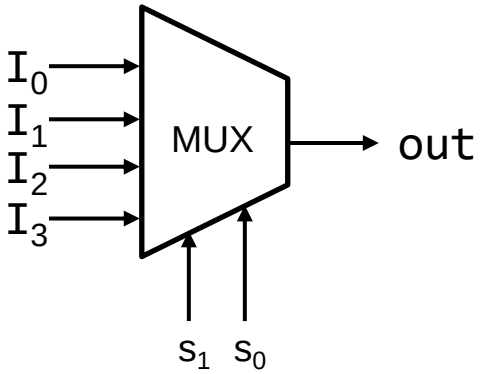
- If I select s=10, then I'm selecting the second line to be displayed on the output



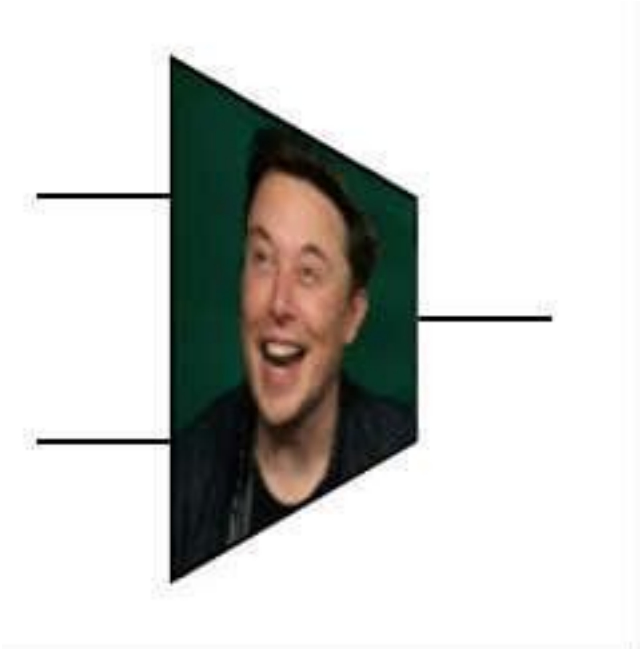
4-input MUX example explanation

- 6 inputs, so truth table would have 64 rows
- We need an efficient way to design this

s_1	s_0	out
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

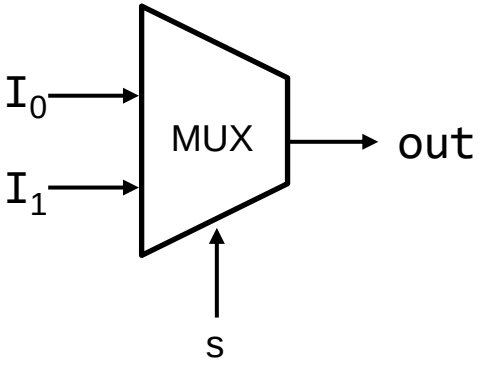


Elon MUX



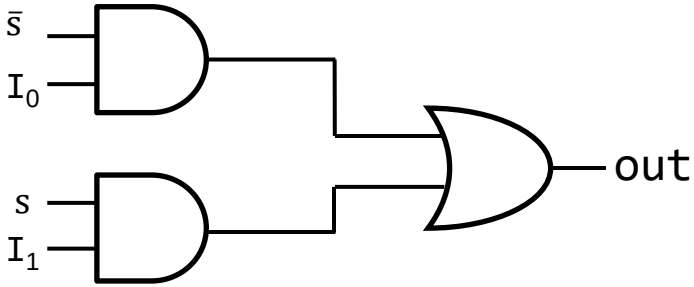
2-input gate level implementation MUX

s	I_0	I_1	out
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



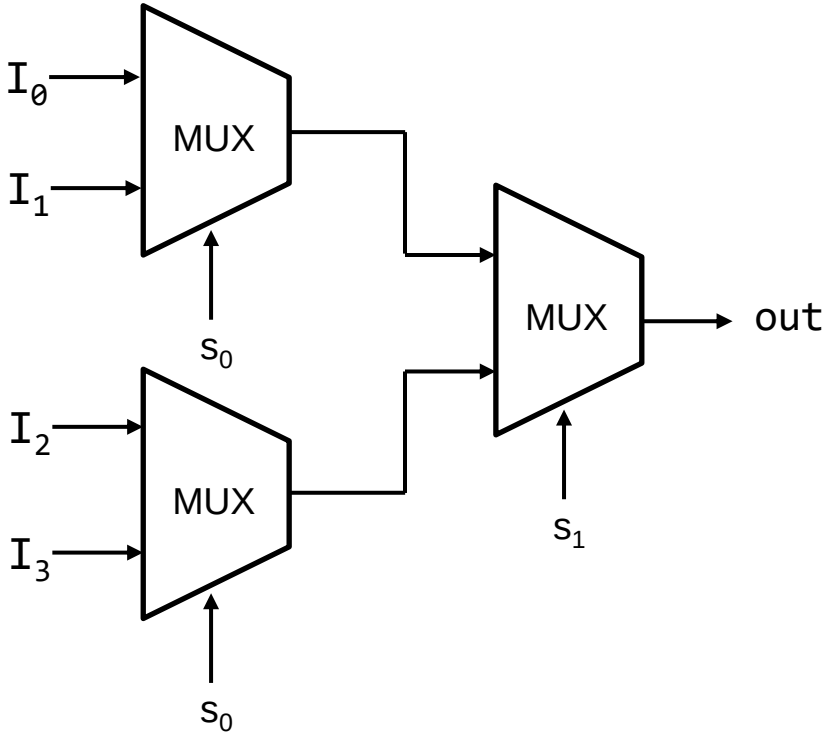
s \ I_0I_1	00	01	11	10
0	0	0	1	1
1	0	1	1	0

$$F = \bar{s}I_0 + sI_1$$



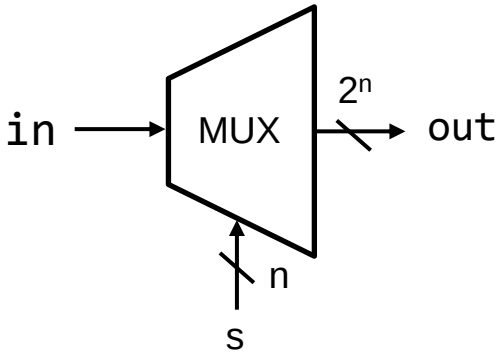
Chain MUXes

Easier than making 6 input Kmap (4 input, 2 select).



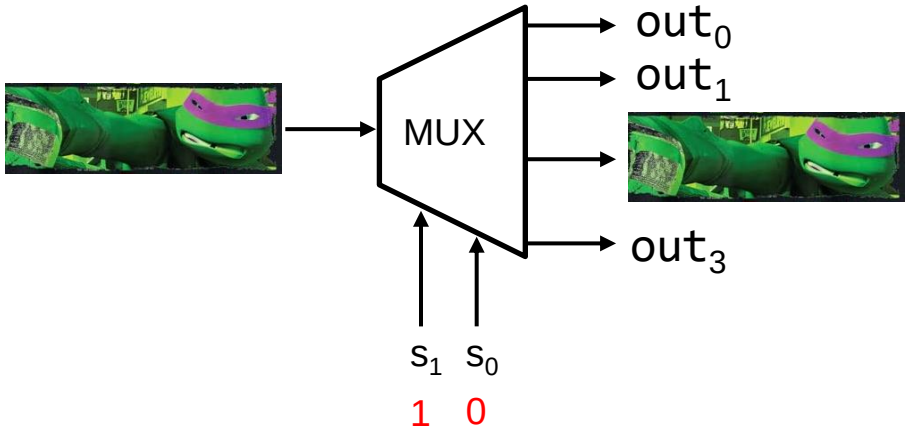
Demultiplexer (DEMUX)

- A demultiplexer (DEMUX) does the inverse of a MUX.
- A DEMUX has n select bits “s” which allow you to select one of the 2^n output lines to copy the input line to
- Basically allows an input to split an output on different wires.



DEMUX example

- If I select s=10, then I'm selecting the second line I_2 to be displayed on the output
- All other outputs have nothing sent over them



DEMUX implementation

- Two different representation of the truth table

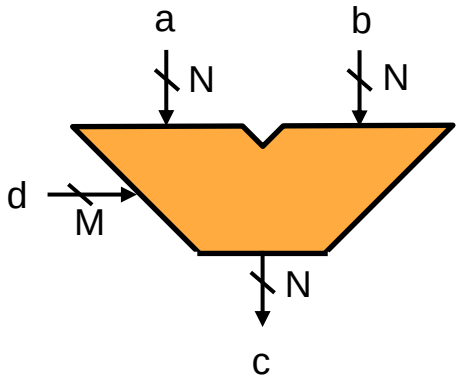
in	s_1	s_0	o_0	o_1	o_2	o_3
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

s_1	s_0	o_0	o_1	o_2	o_3
0	0	in	0	0	0
0	1	0	in	0	0
1	0	0	0	in	0
1	1	0	0	0	in

ALU

- **Arithmetic logic unit (ALU)** is a combinational digital circuit that performs arithmetic and bitwise operations on integer binary numbers.

a,b are integer operands
c is an integer result
d is an operation selector



N: the width of the inputs and outputs of the ALU.
M: the width of the operation selector. There are 2^M operations for a selector that has a width of M.

Summary

- Half adder IN(a,b) OUT(c_{out}, sum)
- Full adder IN(a,b,c_{in}) OUT(c_{out}, sum)
- Carry lookahead adder reduces cumulative gate delay
- Decoder outputs a 1 on a specific wire, and 0 everywhere else
- Multiplexer and Demultiplexer convert multiple wires to share one line and the reverse respectively.

References

- https://www.gatevidyalay.com/tag/disadvantages-of-carry-look-ahead-adder/
- https://www.allaboutelectronics.org/half-adder-and-full-adder-explained/
- https://redirect.cs.umbc.edu/~tsimo1/CMSC313/topics/Slides21.pdf