START
RECORDING

Circuits

CMSC250

What if T=1 and False =0?

- This is useful when we get to circuits
- What is AND, OR, and NOT?
- NOT = 1-x

x	$\sim x$
F	***T***
T	***F***

x	$1 - x$
0	***1***
1	***0***

What if T=1 and False =0?

- What is AND, OR, NOT?
- AND = xy

x	y	$x \wedge y$
F	*F*	*F*
F	*T*	*F*
T	*F*	*F*
T	*T*	*T*

x	y	xy
0	*0*	*0*
0	*1*	*0*
1	*0*	*0*
1	*1*	*1*

What if T=1 and False =0?

- What is AND, OR, and NOT?
- OR = x+y? NO!

x	y	$x \vee y$
F	*F*	*F*
F	*T*	*T*
T	*F*	*T*
T	*T*	*T*

x	y	$x + y$
0	*0*	*0*
0	*1*	*1*
1	*0*	*1*
1	*1*	*10*

What if T=1 and False =0?

- What is AND, OR, and NOT?
- OR = x+y-xy

x	y	$x \vee y$
F	***F***	***F***
F	***T***	***T***
T	***F***	***T***
T	***T***	***T***

x	y	$x + y$	$x + y - \mathrm{xy}$
0	***0***	***0***	***0***
0	***1***	***1***	***1***
1	***0***	***1***	***1***
1	***1***	***10***	***1***

Circuits

- We can build circuits for addition, multiplication, division, bit shifting…
- Every logical operation we have learned $(\sim,\wedge,\vee)$ maps straightforwardly to a tiny piece of hardware called a *logical gate.*
- These gates connect to each other to make arbitrarily complicated circuits!

From a truth table to a formula

p	q	r	output
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

From a truth table to a formula

p	q	r	output
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

- Let us focus entirely on the rows that output 1!

p	q	r	output
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Focusing on the 1st row…

p	q	r	output
0	0	0	1

- Write a formula that is ’1’ only on inputs p =0, q = 0, r = 0.

p	q	r	output
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Focusing on the 1st row…

p	**q**	**r**	**output**
0	0	0	1

- Write a simple formula that is ’1’ only on inputs p =0, q = 0, r = 0.

$$\sim p \land \sim q \land \sim r$$

p	q	r	output
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Focusing on the 4th row…

p	q	r	output
0	1	1	1

- Same deal

p	q	r	output
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Focusing on the 4th row…

p	q	r	output
0	1	1	1

- Same deal

$$\sim p \wedge q \wedge r$$

p	q	r	output
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Focusing on the 5th row…

p	q	r	output
1	0	0	1

$$p \land {\sim}q \land {\sim}r$$

p	q	r	output
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Focusing on the 8th row…

p	q	r	output
1	1	1	1

$$p \wedge q \wedge r$$

How do we combine those simple formulae?

$\sim p \land \sim q \land \sim r$

$\sim p \land q \land r$

$p \land \sim q \land \sim r$

$p \land q \land r$

How do we combine those simple formulae?

$(\sim p \wedge \sim q \wedge \sim r) \vee$

$(\sim p \wedge q \wedge r) \vee$

$(p \wedge \sim q \wedge \sim r) \vee$

$(p \wedge q \wedge r)$

p	q	r	output
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

- Outputs 1 if and only if the truth table outputs 1!

How do we combine those simple formulae?

$(\sim p \wedge \sim q \wedge \sim r)\ \vee$

$(\sim p \wedge q \wedge r) \vee$

$(p \wedge \sim q \wedge \sim r) \vee$

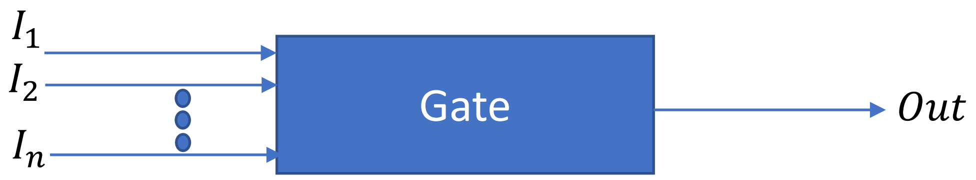
$(p \wedge q \wedge r)$

p	**q**	**r**	**output**
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

- Outputs 1 if and only if the truth table outputs 1!
- **We want to do this in *hardware!***

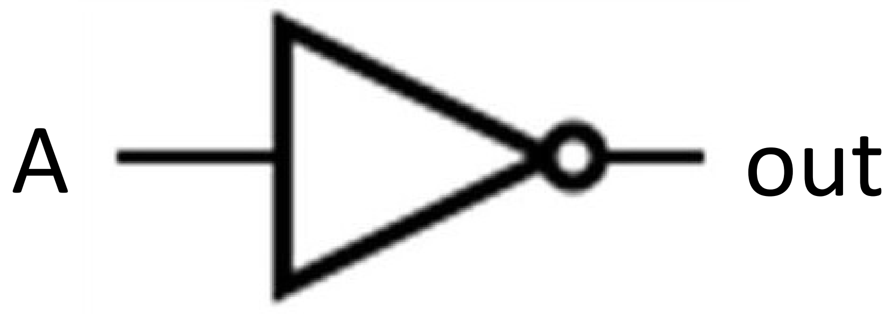
Logical gates

- The smallest pieces of hardware that we will examine are called ***logical gates.***
- Most gates for this course will take **bits** as inputs and will emit one **bit** as output. (Not all gates have this property)



- Those gates can connect to each other in various different ways in order to create more complex circuits

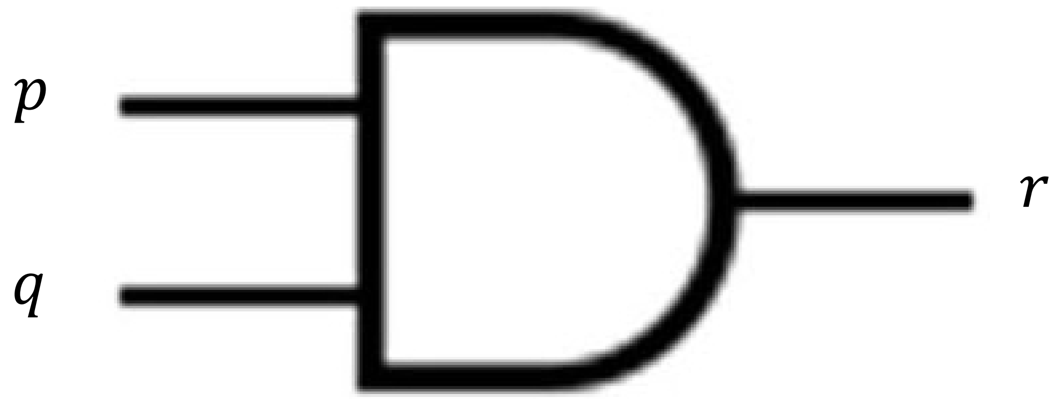
Our first gate



A	out
0	1
1	0

- This gate is known as the **inverter.**
- It corresponds **exactly** to the negation operation in propositional logic!
  - Where 1, set True.
  - Where 0, set False

Our second gate



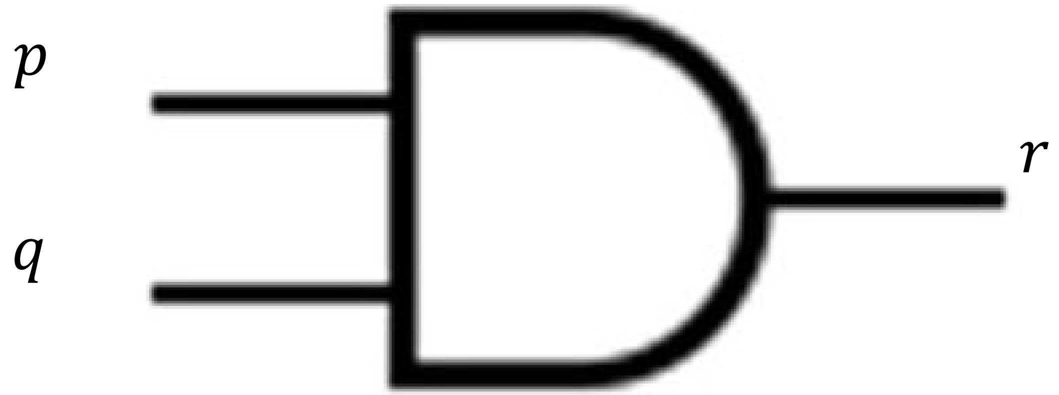
p	q	r
0	0	0
0	1	0
1	0	0
1	1	1

- Corresponds to:

Conjunction

Disjunction

Our second gate (AND gate)

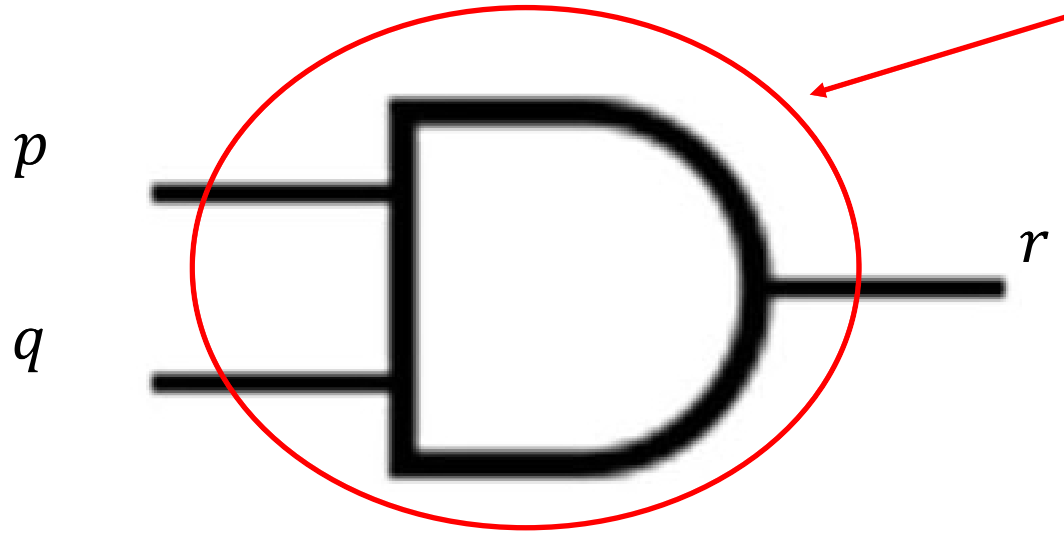


p	q	r
0	0	0
0	1	0
1	0	0
1	1	1

- Corresponds to:

[✓ Conjunction] Disjunction

Our second gate (AND gate)



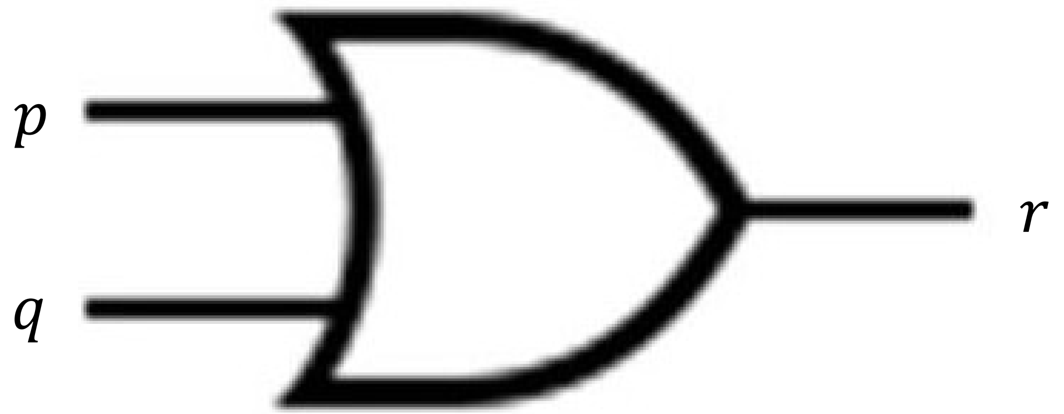
p	q	r
0	0	0
0	1	0
1	0	0
1	1	1

- Corresponds to:

Conjunction

Disjunction

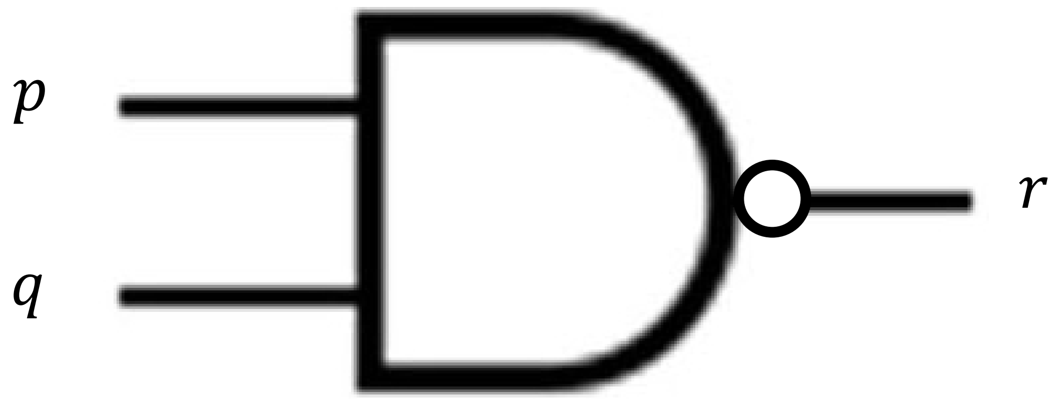
Our third gate (OR gate)



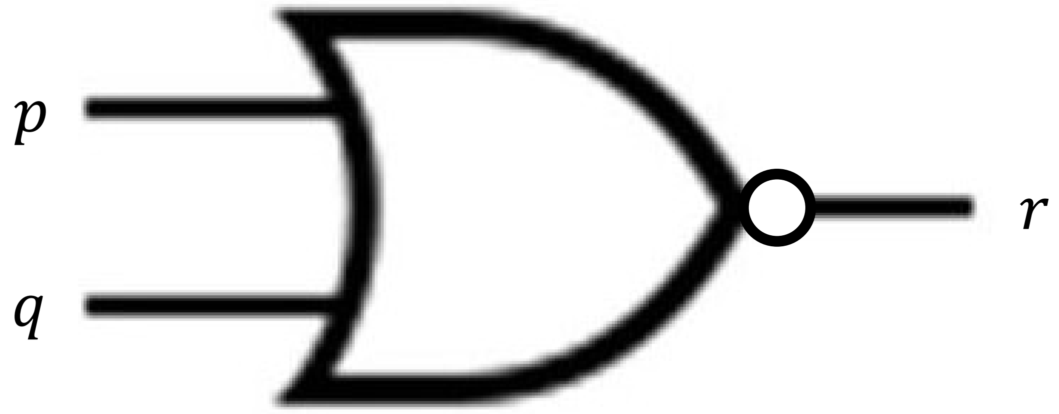
p	q	r
0	0	0
0	1	1
1	0	1
1	1	1

- Corresponds to logical disjunction (OR)

Our fourth and fifth gate (NAND and NOR gate)



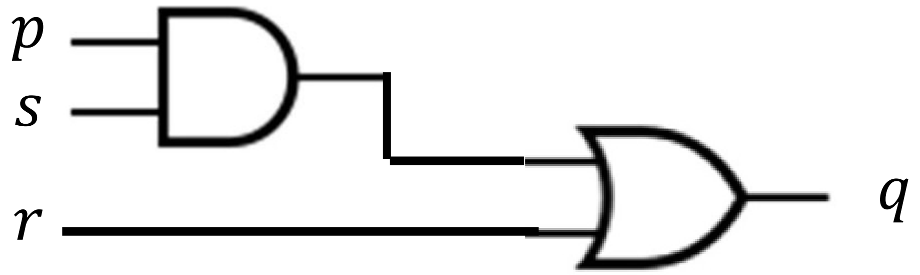
p	q	r
0	0	1
0	1	1
1	0	1
1	1	0



p	q	r
0	0	1
0	1	0
1	0	0
1	1	0

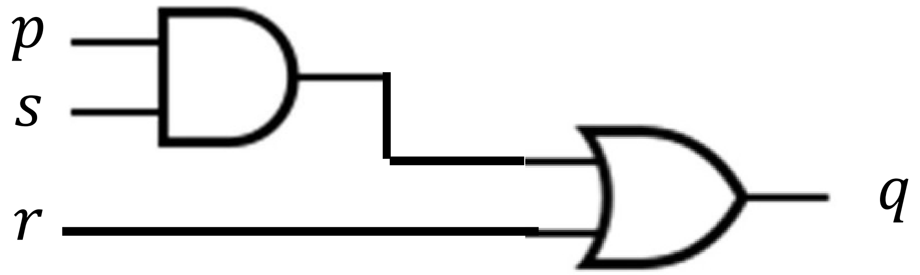
Exercises

- Which boolean function does this circuit correspond to?



Exercises

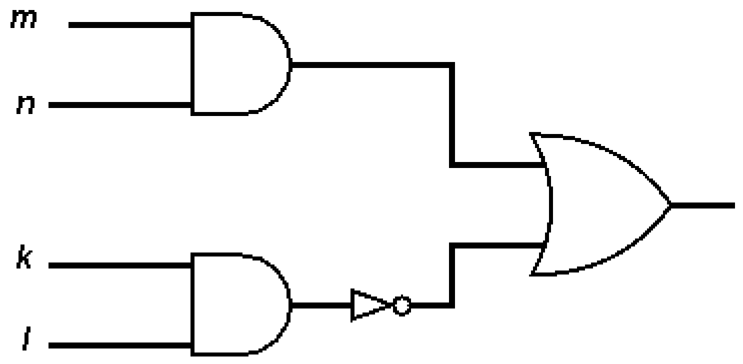
- Which boolean function does this circuit correspond to?



$$(p \wedge s) \vee r$$

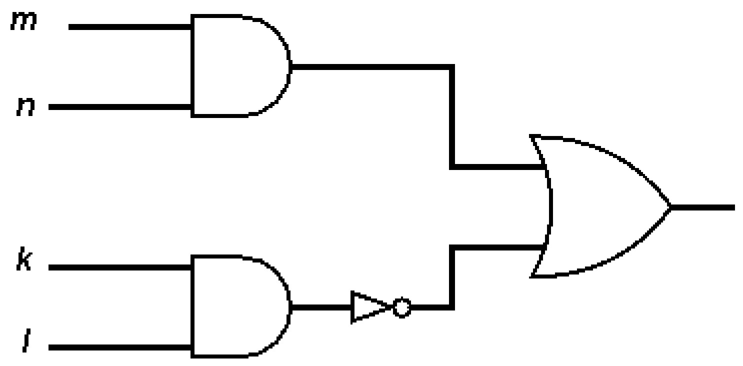
Exercises

- And this?



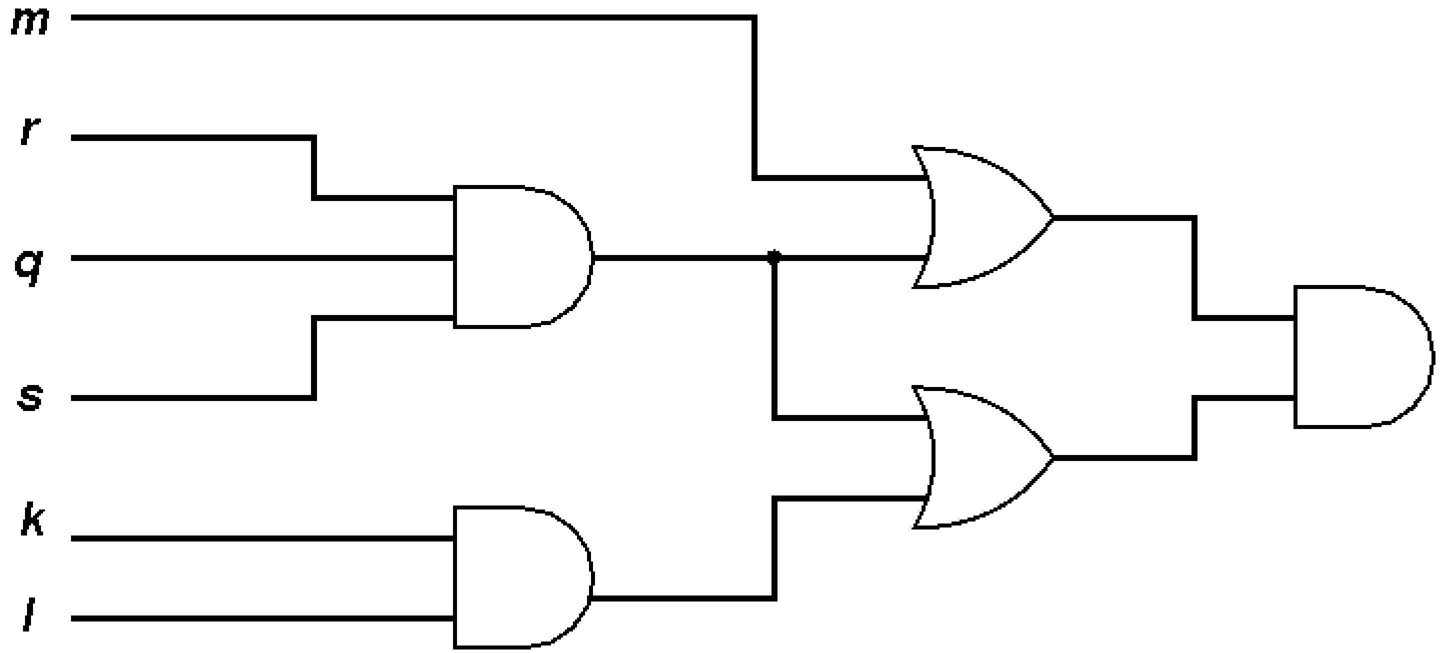
Exercises

- And this?

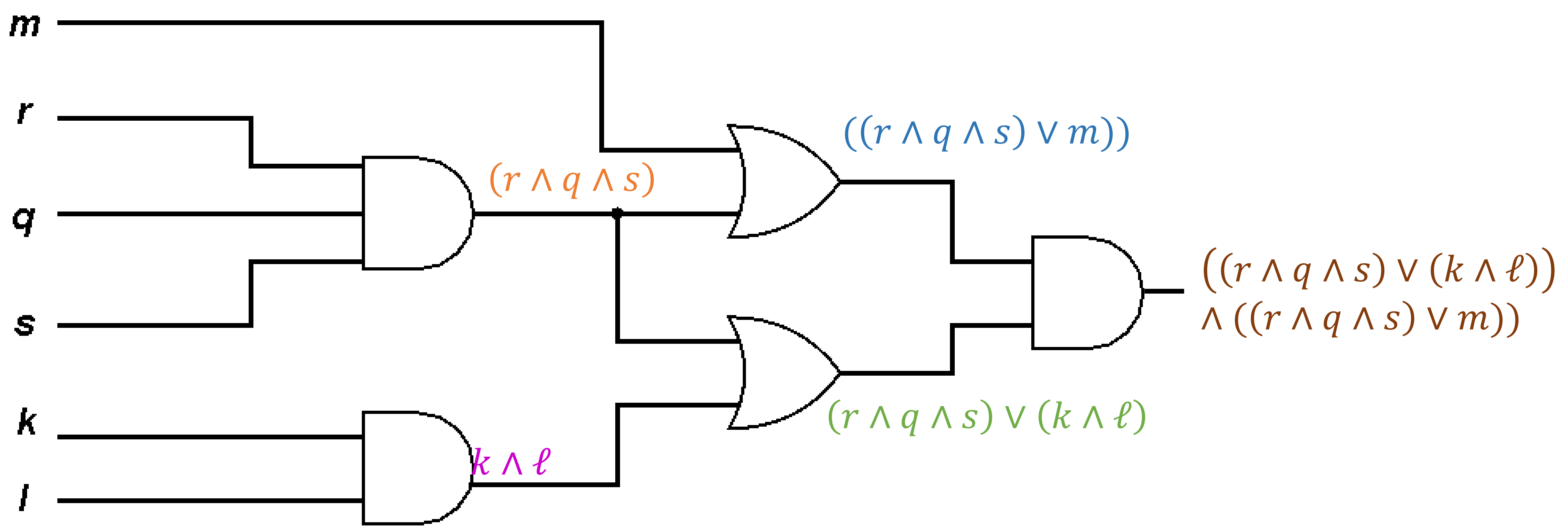


$$(m \wedge n) \vee (\sim (k \wedge l))$$

And this?



And this?

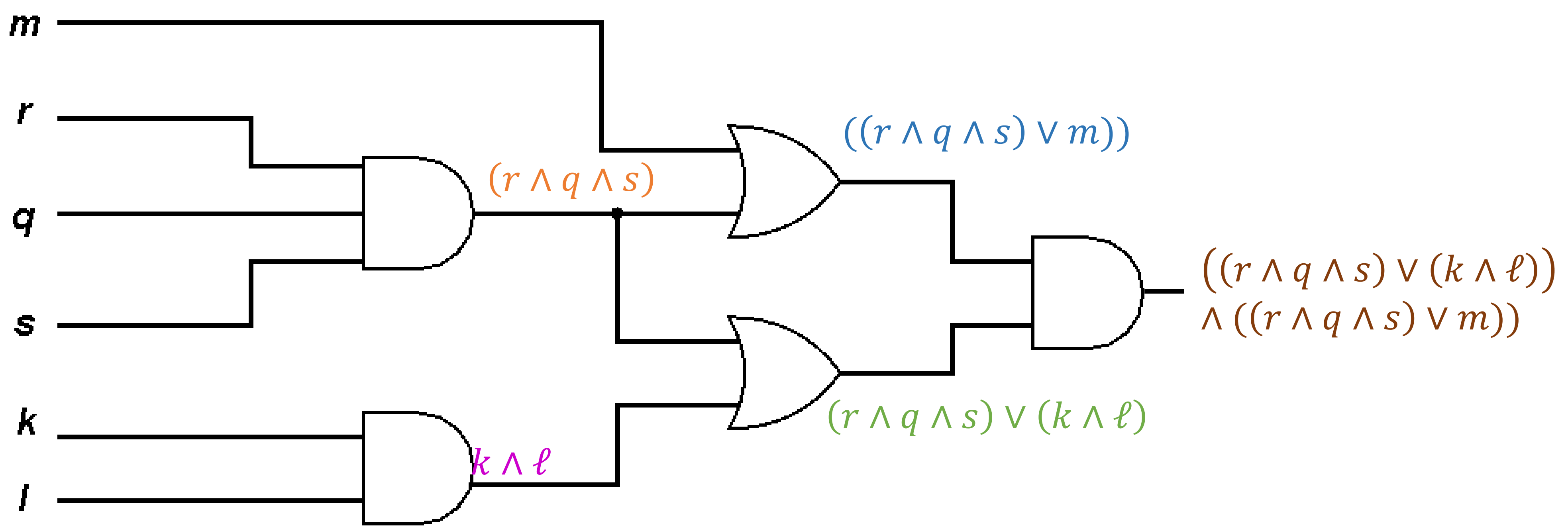


m
r
q
s
k
l
$(r \wedge q \wedge s)$
$k \wedge \ell$
$((r \wedge q \wedge s) \vee m))$
$(r \wedge q \wedge s) \vee (k \wedge \ell)$
$((r \wedge q \wedge s) \vee (k \wedge \ell))$
$\wedge ((r \wedge q \wedge s) \vee m))$

$$((r \wedge q \wedge s) \vee (k \wedge \ell)) \wedge ((r \wedge q \wedge s) \vee m))$$

And this?

Can we make this circuit *cheaper*?

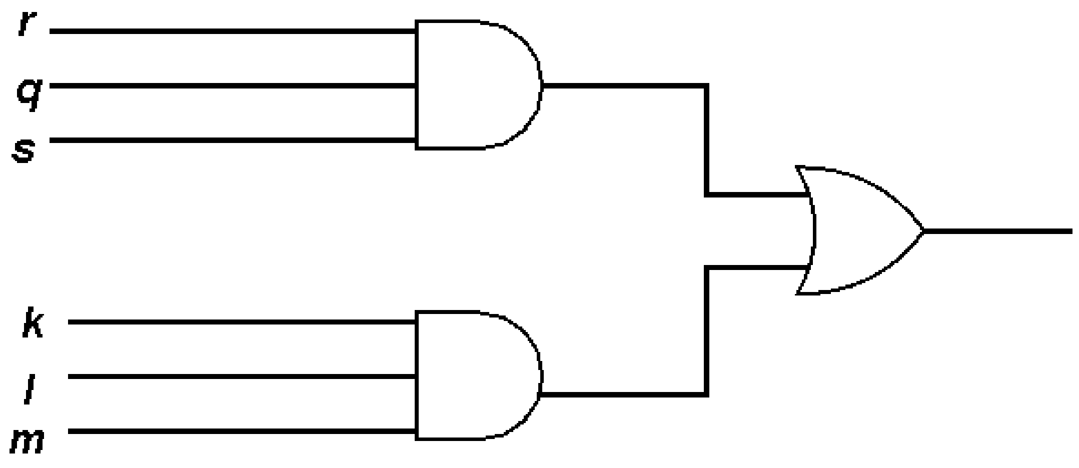


m
r
q
s
k
l
$(r \wedge q \wedge s)$
$k \wedge \ell$
$((r \wedge q \wedge s) \vee m))$
$(r \wedge q \wedge s) \vee (k \wedge \ell)$
$((r \wedge q \wedge s) \vee (k \wedge \ell))$
$\wedge ((r \wedge q \wedge s) \vee m))$

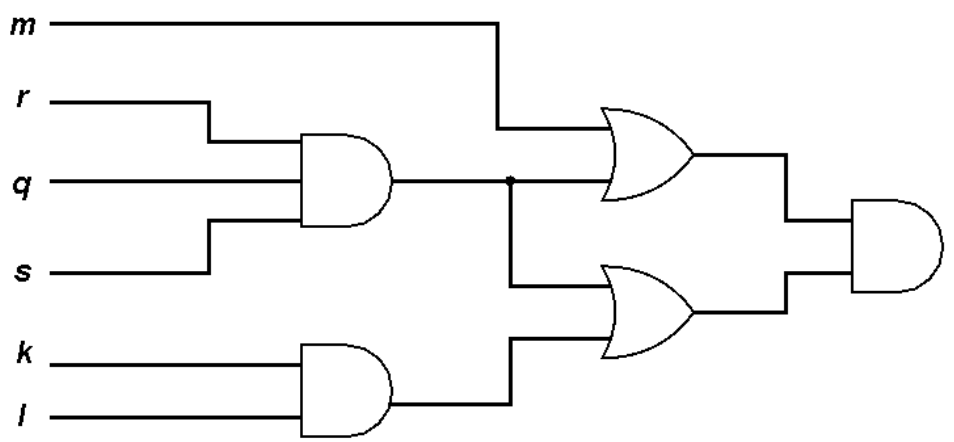
$$((r \wedge q \wedge s) \vee (k \wedge \ell)) \wedge ((r \wedge q \wedge s) \vee m))$$

Simplifying the circuit…

$$((r \wedge q \wedge s) \vee (k \wedge \ell)) \wedge ((r \wedge q \wedge s) \vee m))$$
$$\equiv (r \wedge q \wedge s) \vee ((k \wedge \ell) \wedge m)$$



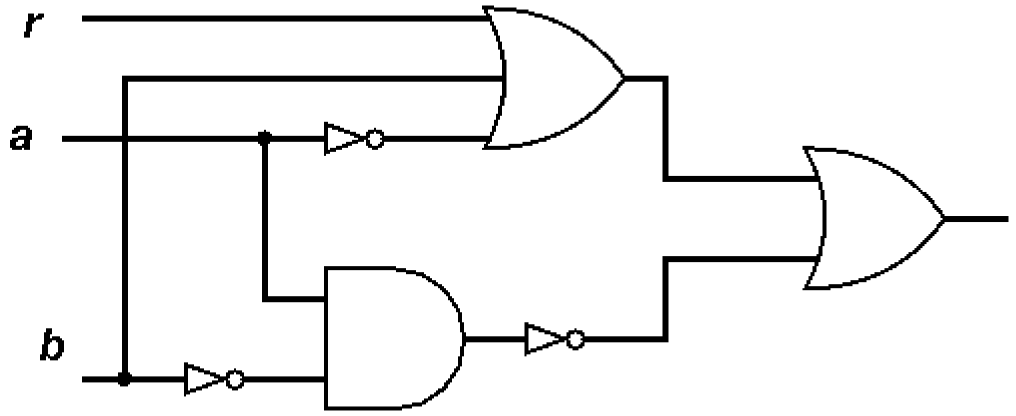
New circuit: Three gates



Old circuit: Five gates

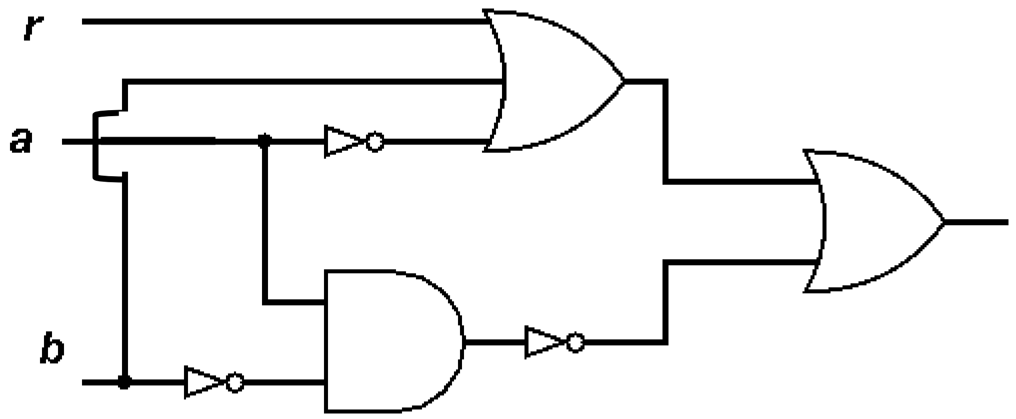
Exercises

1. Which logical expression is computed by the following circuit?



Exercises

1. Which logical expression is computed by the following circuit?
2. ***Simplify*** the circuit as much as possible!



Coming back to our original formula…

$$(\sim p \wedge \sim q \wedge \sim r) \vee (\sim p \wedge q \wedge r) \vee (p \wedge \sim q \wedge \sim r) \vee (p \wedge q \wedge r)$$

Coming back to our original formula…

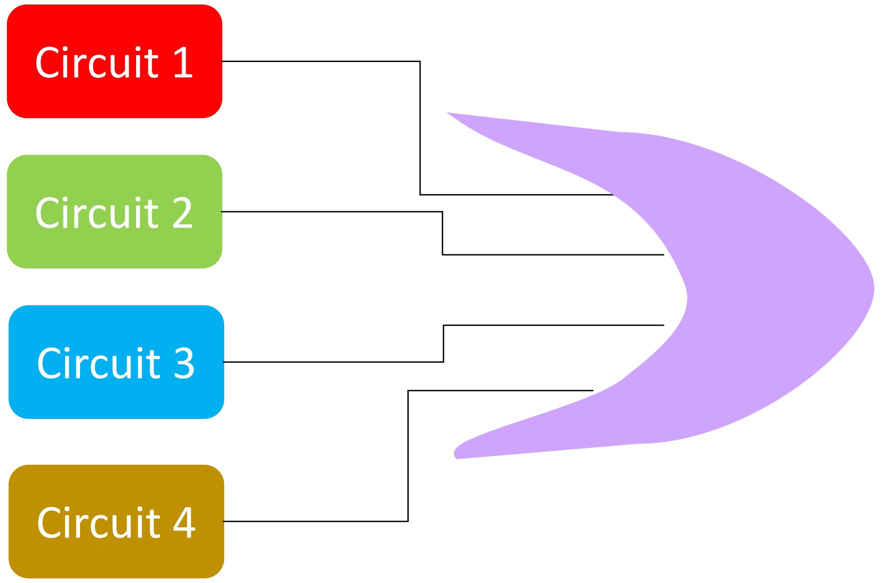
$$(\sim p \wedge \sim q \wedge \sim r) \vee (\sim p \wedge q \wedge r) \vee (p \wedge \sim q \wedge \sim r) \vee (p \wedge q \wedge r)$$

- For each small formula we have a circuit, and we will combine with a 4-input OR gate!

Coming back to our original formula…

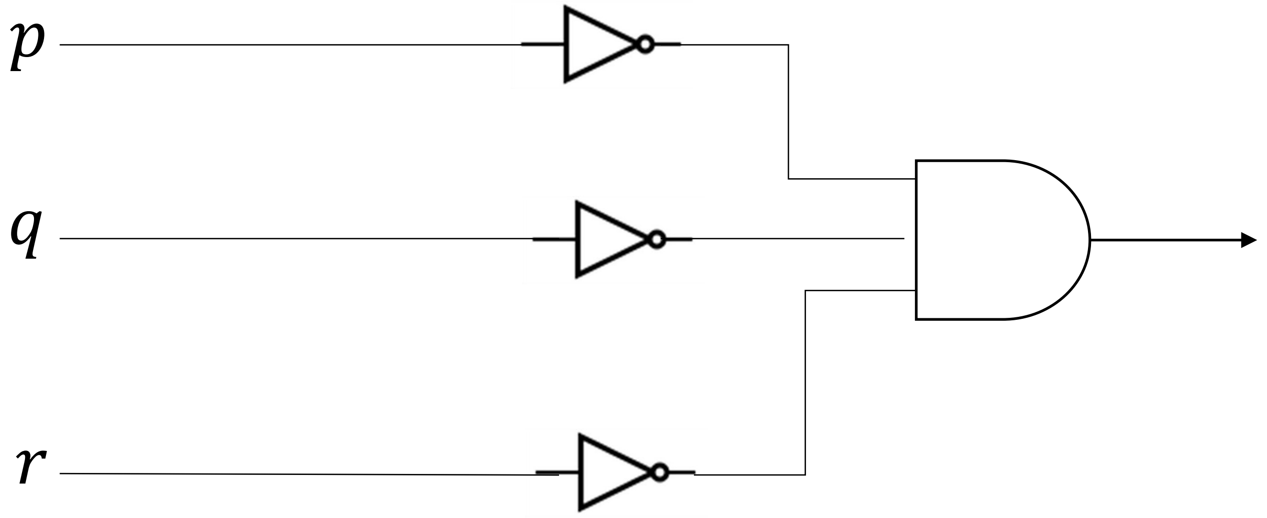
$$(\sim p \land \sim q \land \sim r) \lor (\sim p \land q \land r) \lor (p \land \sim q \land \sim r) \lor (p \land q \land r)$$

- For each small formula we have a circuit, and we will combine with a 4-input OR gate!



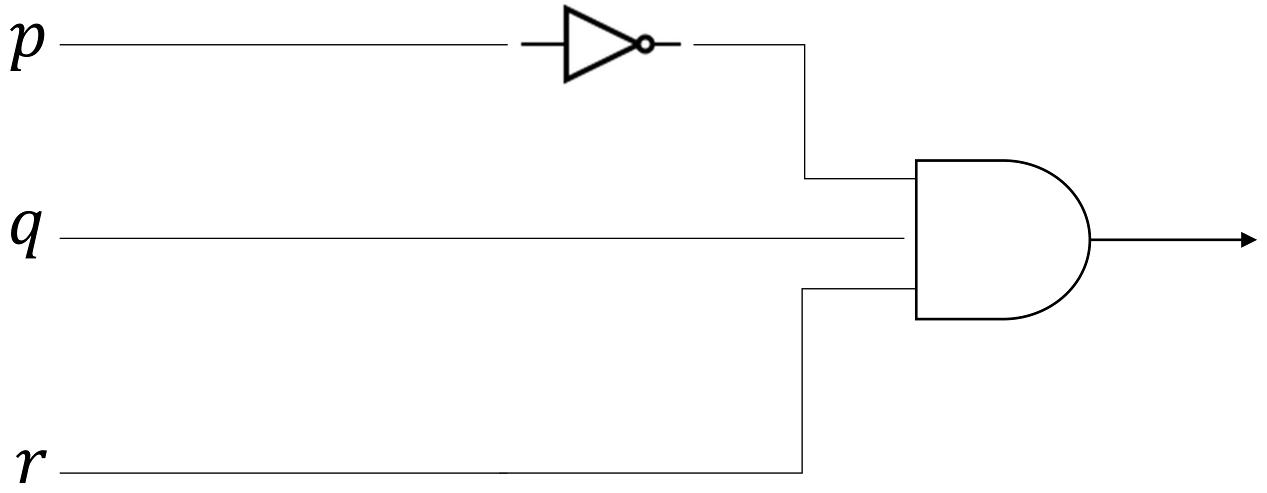
Circuit 1

$(\sim p \wedge \sim q \wedge \sim r)$



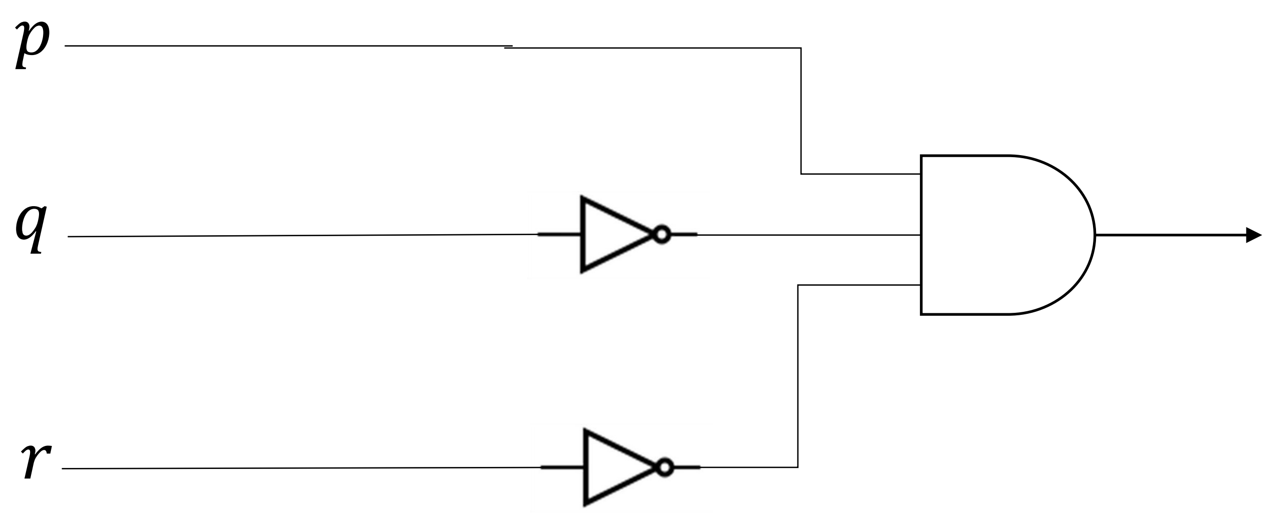
Circuit 2

$(\sim p \land q \land r)$



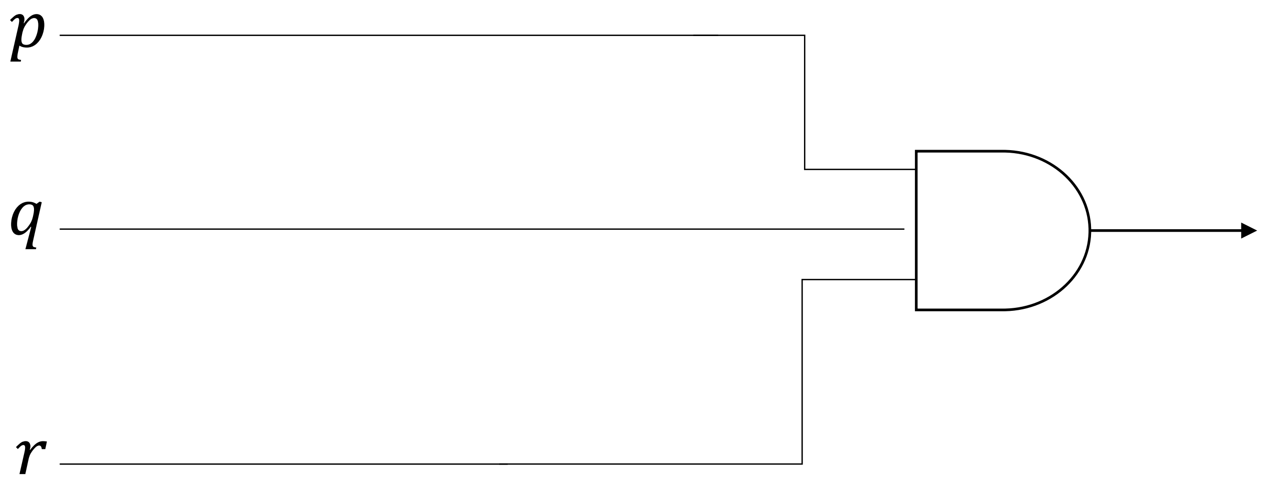
Circuit 3

$$(p \wedge \sim q \wedge \sim r)$$



Circuit 4

$(p \wedge q \wedge r)$



Building Adder Circuits

- We want to build circuits that add arbitrarily large binary numbers.
- E.g

$$\begin{array}{rl} \left.\begin{array}{r}10011001\\ +00110011\end{array}\right\} & \text{Inputs} \\ \hline 11001100 \longrightarrow & \text{Output}\end{array}$$

Half-Adder

- A half-adder is a circuit that adds **two bits** together!

$$\begin{array}{r} X \\ \underline{+\,Y} \\ C\ S \end{array}$$

- (Remember: C is the carry bit.)
- Let's try to build a circuit that computes both S and C!

Truth table

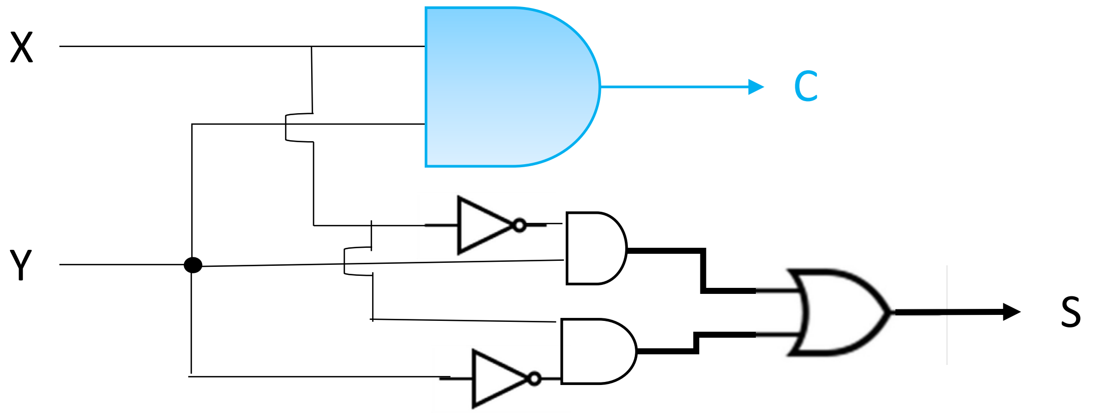
X	**Y**	**S**	**C**
0	0	?	?
0	1	?	?
1	0	?	?
1	1	?	?

Truth table

X	Y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

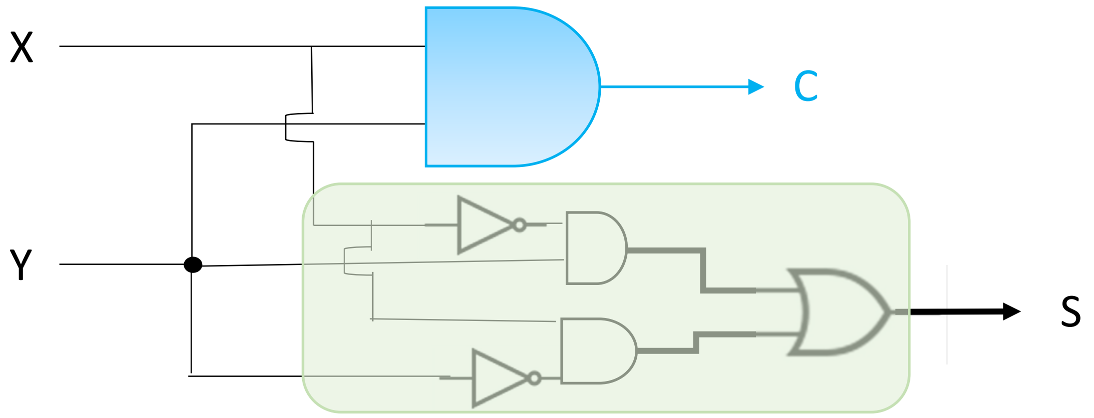
Truth table

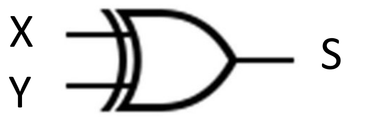
X	Y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Truth table

X	**Y**	**S**	**C**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1





XOR Gate
(“Exclusive OR)

Making XOR cheaper

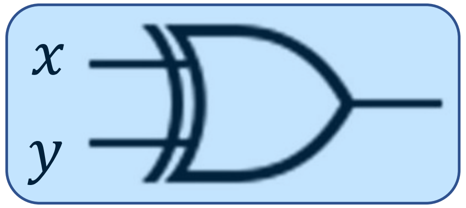
- First, let's convince ourselves that

$$(x \oplus y) \equiv (x \ \wedge (\sim y)) \vee \big((\sim x) \wedge y\big) \equiv (x \vee y) \wedge (\sim (x \wedge y))$$

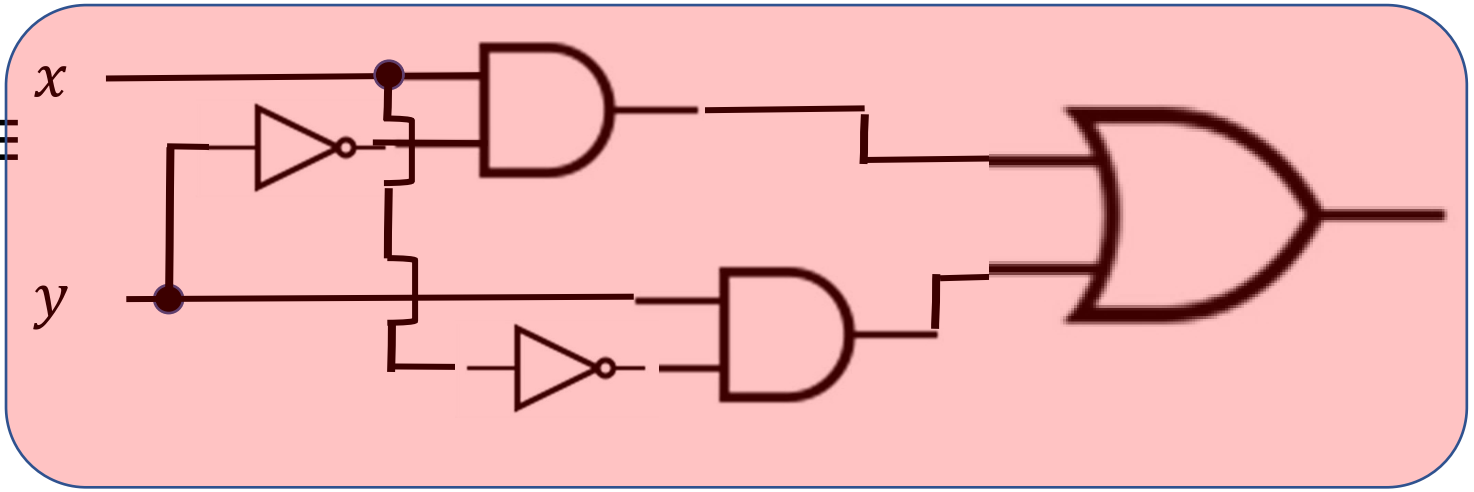
Making XOR cheaper

- First, let's convince ourselves that

$$(x \oplus y) \equiv (x \wedge (\sim y)) \vee ((\sim x) \wedge y) \equiv (x \vee y) \wedge (\sim (x \wedge y))$$



$\equiv$

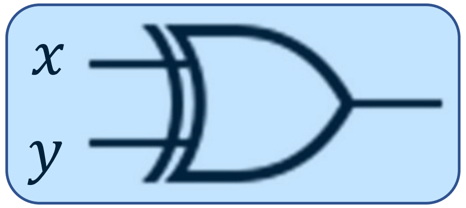


$\equiv$

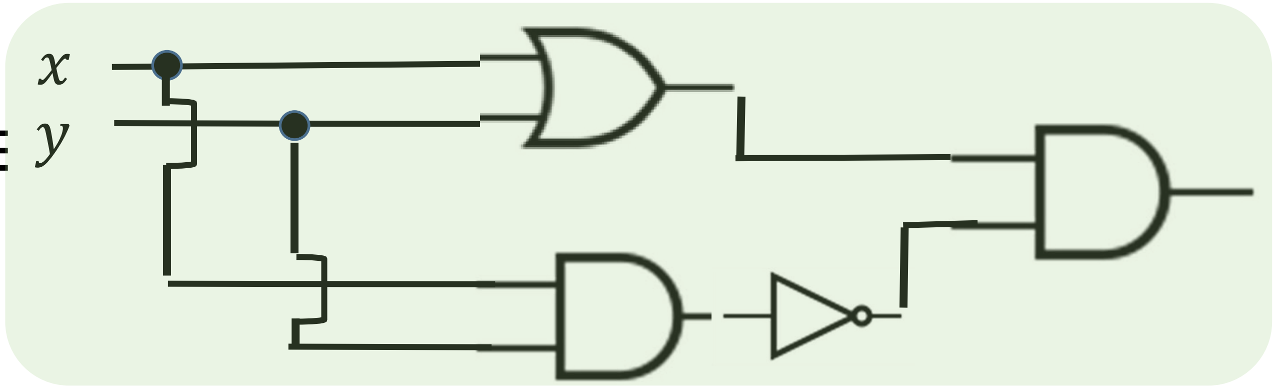
Making XOR cheaper

- First, let's convince ourselves that

$$(x \oplus y) \equiv (x \wedge (\sim y)) \vee ((\sim x) \wedge y) \equiv (x \vee y) \wedge (\sim(x \wedge y))$$



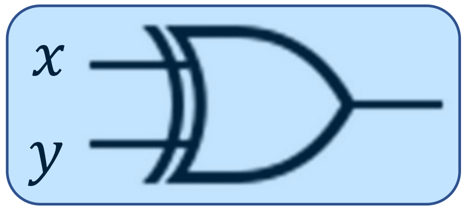
$\equiv$



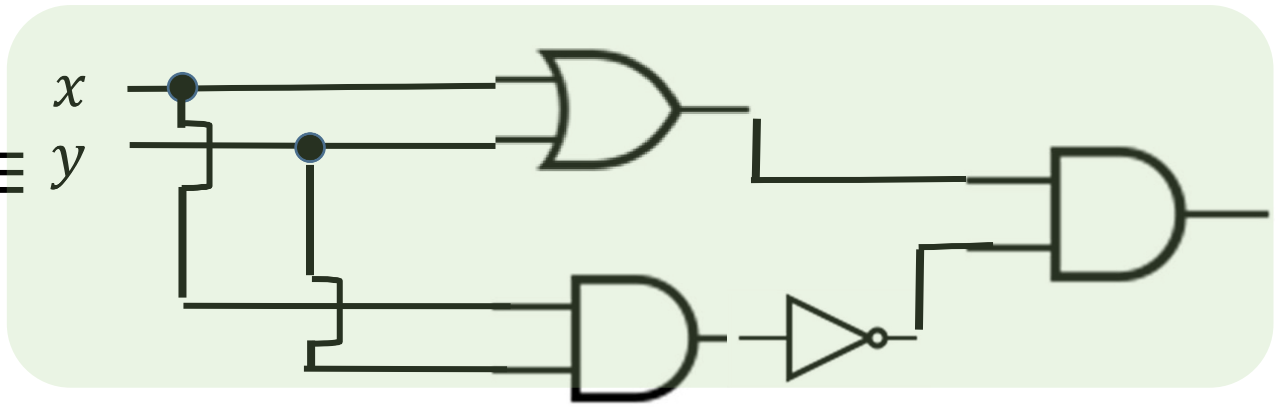
Making XOR cheaper

- First, let's convince ourselves that

$$(x \oplus y) \equiv (x \wedge (\sim y)) \vee ((\sim x) \wedge y) \equiv (x \vee y) \wedge (\sim(x \wedge y))$$



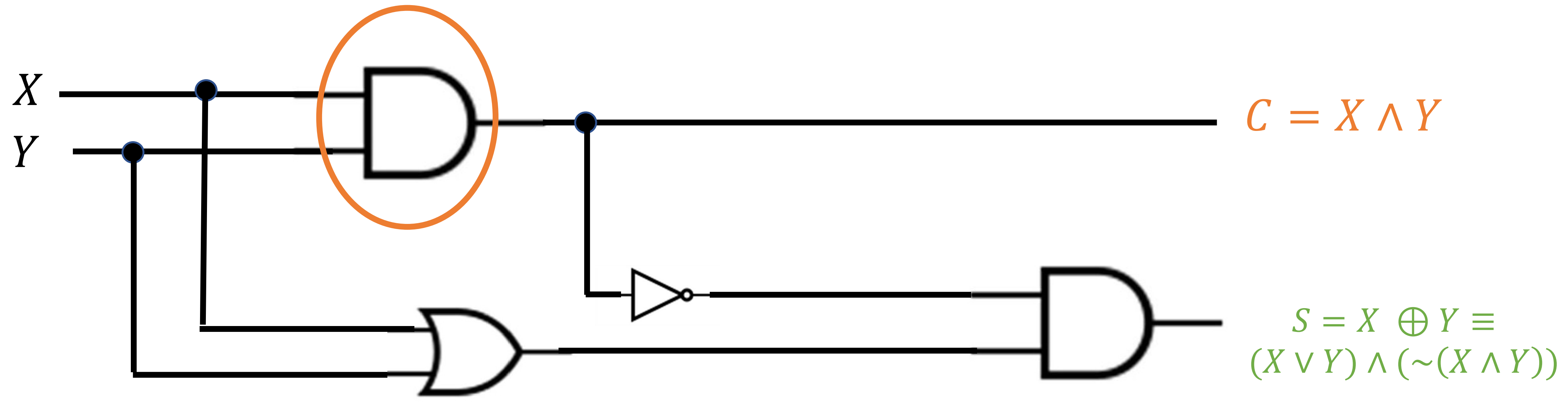
$\equiv$



From **five** gates to **four**!

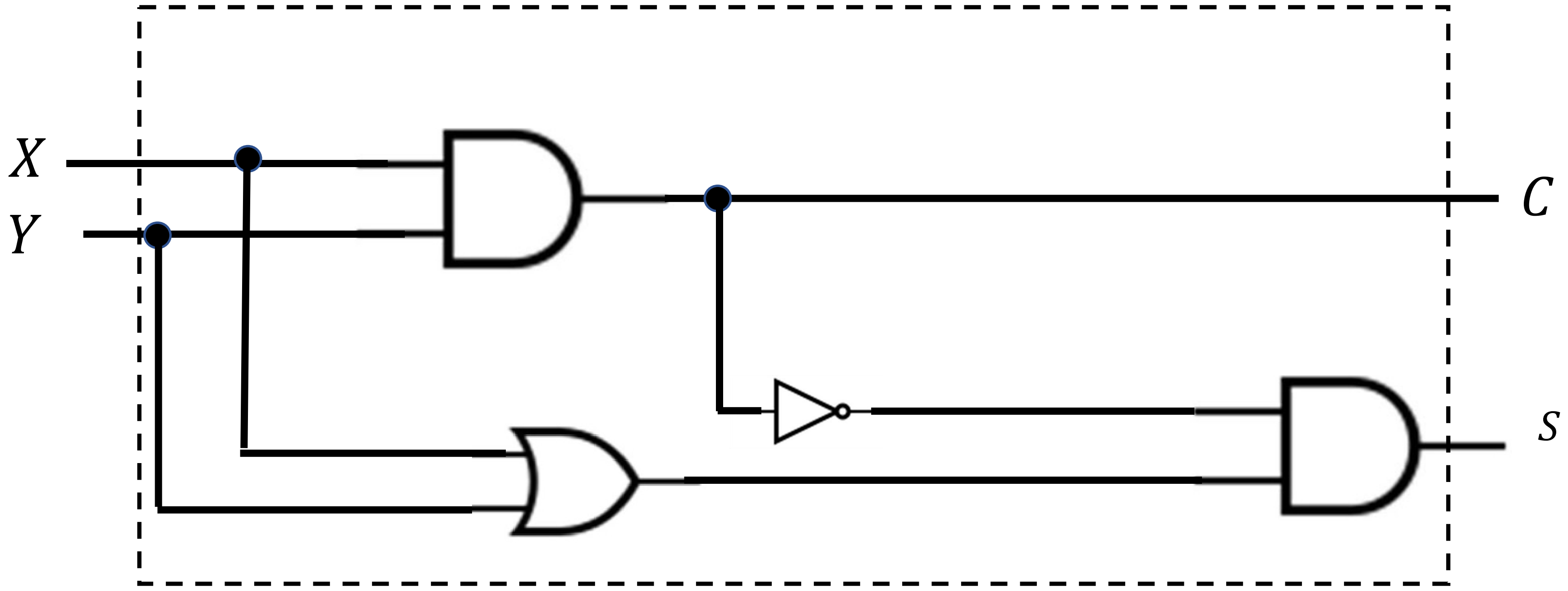
Optimizing Half Adder

- We can now optimize the Half Adder.
- We won't just use simplified XOR, but also leverage simplified XOR to **re-use** the AND gate used to compute the carry bit C!



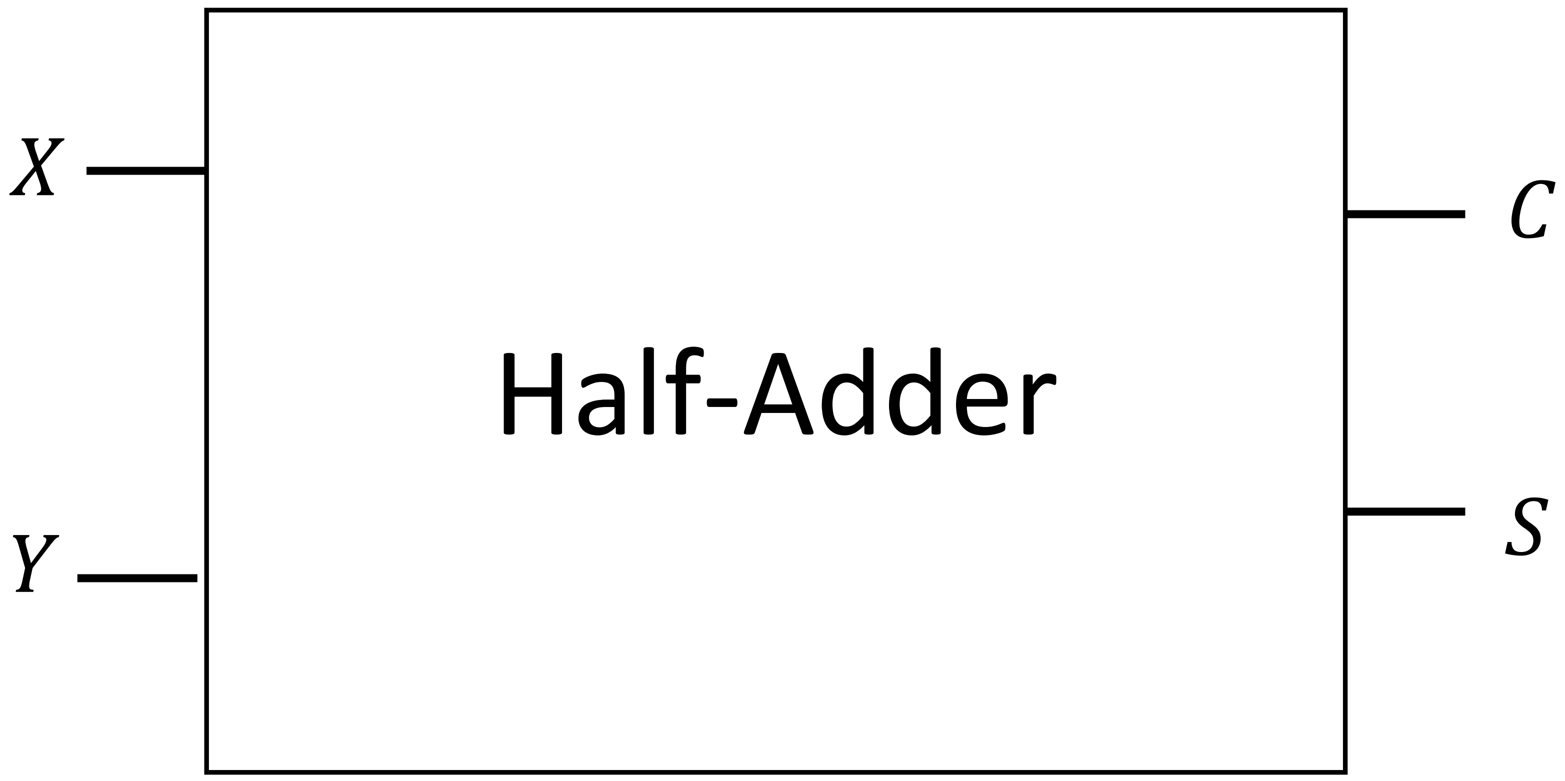
X
Y
$C = X \wedge Y$
$S = X \oplus Y \equiv (X \vee Y) \wedge (\sim(X \wedge Y))$

Half Adder Abstraction



4 gates, instead of 6 for the previous one!

Half Adder Abstraction



Full-Adder

- Now, let’s consider the complete case, where we want to build a circuit that computes the sum of two 2-digit binary numbers:

$$\begin{array}{rrr} & P & Q \\ + & \underline{W} & \underline{X} \\ C & S_1 & S_2 \end{array}$$

- To do this, we also need the ability to add 3 digits, because:

$$\begin{array}{rrr} & \underline{C_1} & \\ & P & Q \\ + & \underline{W} & \underline{X} \\ C & S_1 & S_2 \end{array}$$

Full-Adder

- Now, let's consider the complete case, where we want to build a circuit that computes the sum of two 2-digit binary numbers:

$$\begin{array}{rrr} & P & Q \\ + & \underline{W} & \underline{X} \\ C & S_1 & S_2 \end{array}$$

- To do this, we also need the ability to add <span>3</span> digits, because:

$$\begin{array}{rrr} & \underline{C_1} & \\ & P & Q \\ + & \underline{W} & \underline{X} \\ C & S_1 & S_2 \end{array}$$

We will call a circuit that adds 3 bits a full adder

We could do the truth table....

P Q

$+\underline{W\ X}$

C S1 S2

P	Q	W	X	C	S_1	S_2
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			
1	0	1	0			
1	0	1	1			
1	1	0	0			
1	1	0	1			
1	1	1	0			
1	1	1	1			

We could do the truth table….

P Q

+ W X

C S1 S2

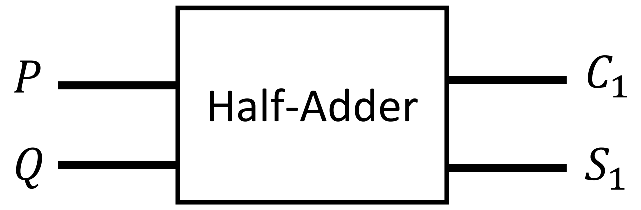
P	Q	W	X	C	S_1	S_2
0	0	0	0			
0	0	0	1			
0	0	1	0			
0	0	1	1			
0	1	0	0			
0	1	0	1			
0	1	1	0			
0	1	1	1			
1	0	0	0			
1	0	0	1			
1	0	1	0			
1	0	1	1			
1	1	0	0			
1	1	0	1			
1	1	1	0			
1	1	1	1			

But it’s time consuming and we are all busy people

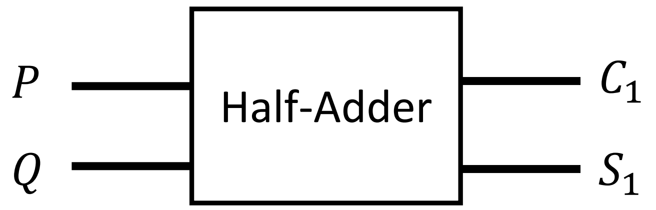
Constructing a Full-Adder in another way

- We need to build a circuit that computes the sum of 3 digits, e.g P + Q + R

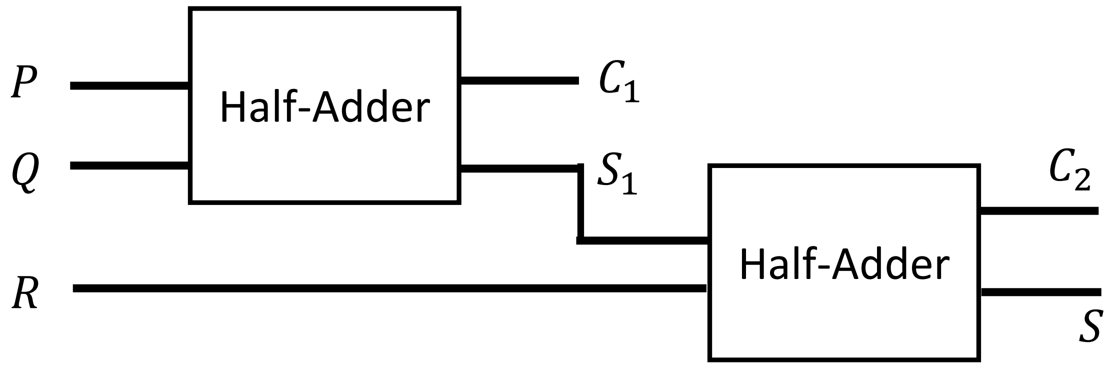
- Step 1: Compute $\begin{array}{r} P \\ +\ Q \\ \hline C_1S_1 \end{array}$ with a **half-adder**:



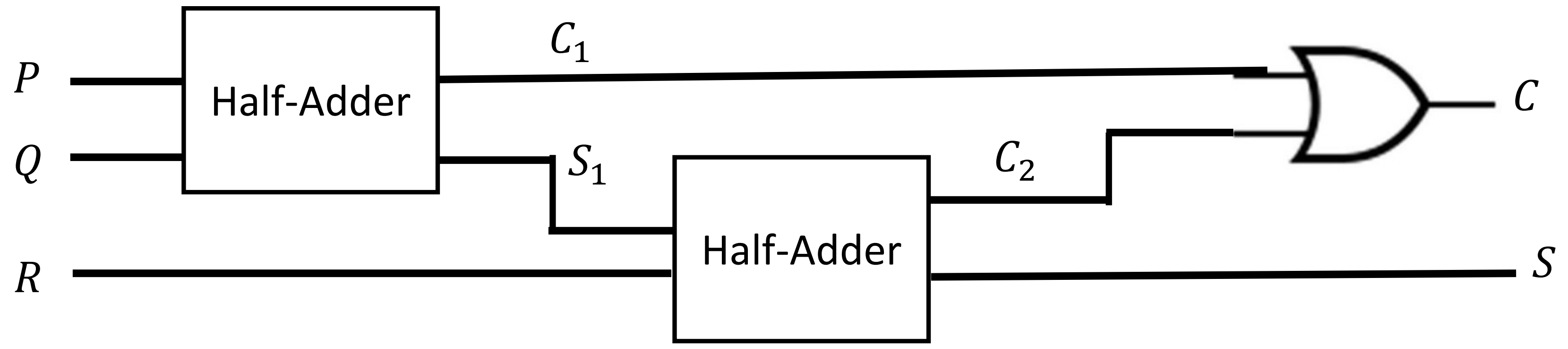
Constructing a Full Adder



- Step 2: Compute $\begin{array}{r} S_1 \\ +\ \underline{\ R} \\ C_2S \end{array}$ with another half-adder:

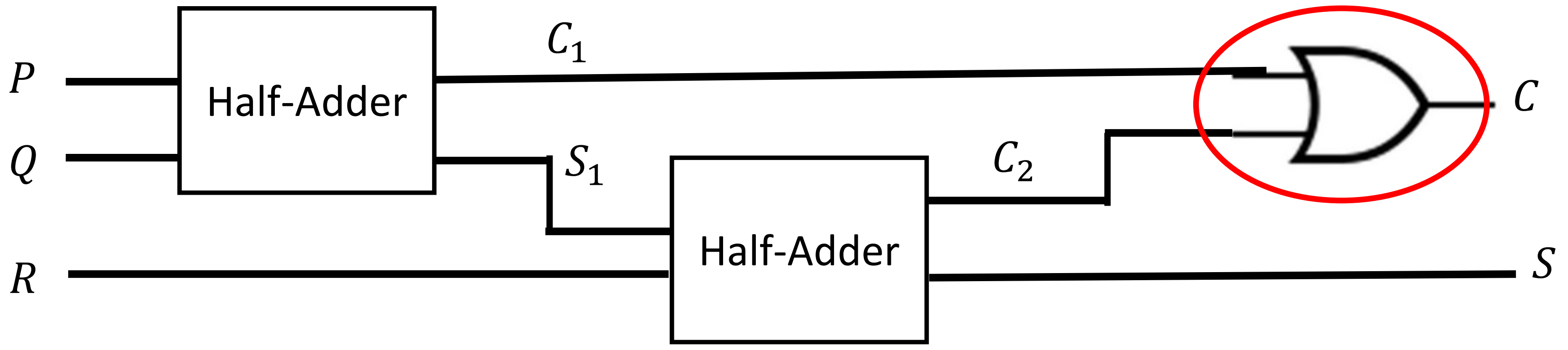


Constructing a full-adder



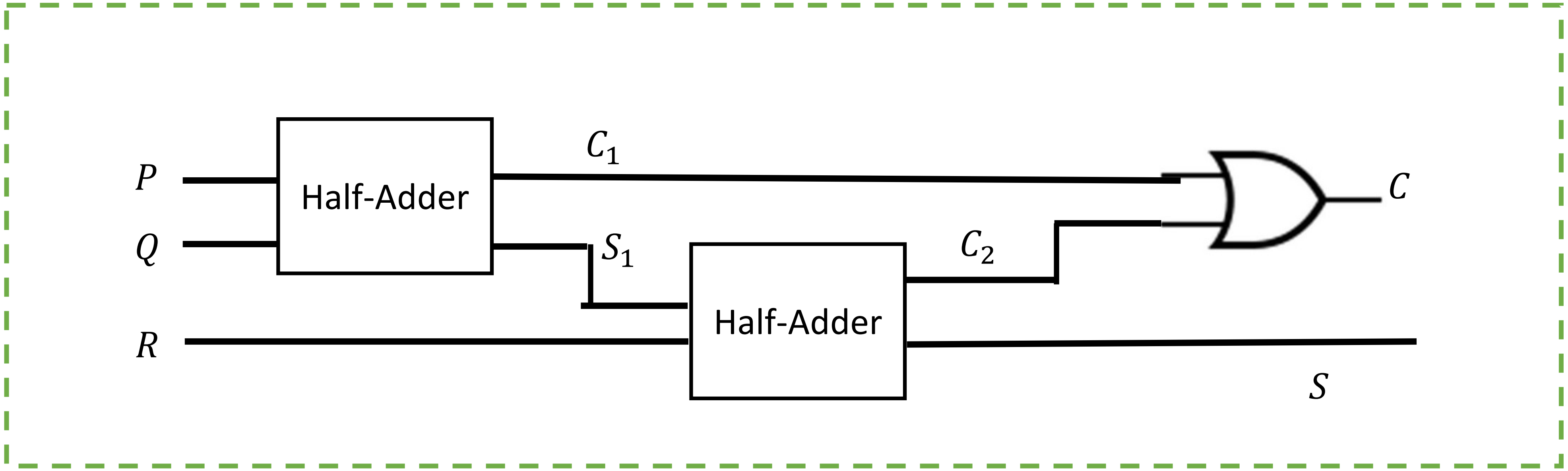
- Step 3: Combine C_1 and C_2 with an OR gate to yield the final carry bit C.

Constructing a full-adder



- Step 3: Combine C_1 and C_2 with an OR gate to yield the final carry bit C.
- Why did we choose an OR gate to combine the "intermediate" carries C_1 and C_2?

Constructing a full-adder

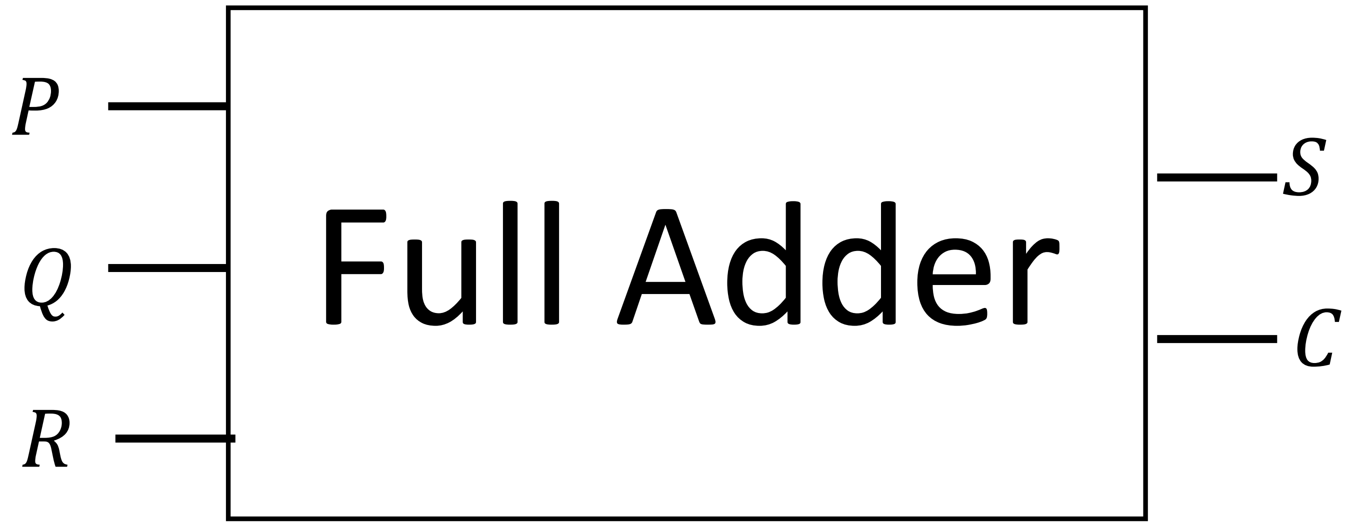


- Step 3: Combine C_1 and C_2 with an OR gate to yield the final carry bit C.

Abstraction time!

Full Adder Black Box

- 3 inputs, 2 outputs

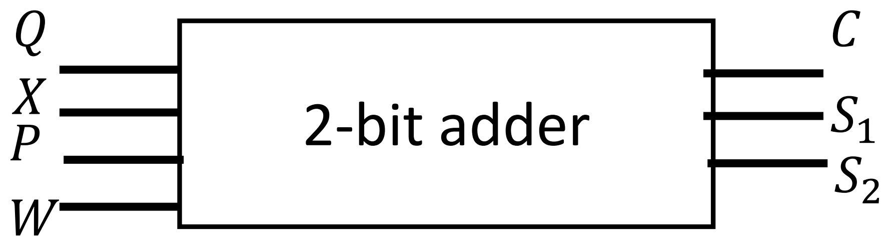


2-bit adder

- However, **we still have not solved our original problem**, which is to construct a circuit that adds 2-bit numbers!

$$\begin{array}{r} P\ Q \\ +\ \underline{W\ X} \\ C\ S_1\ S_2 \end{array}$$

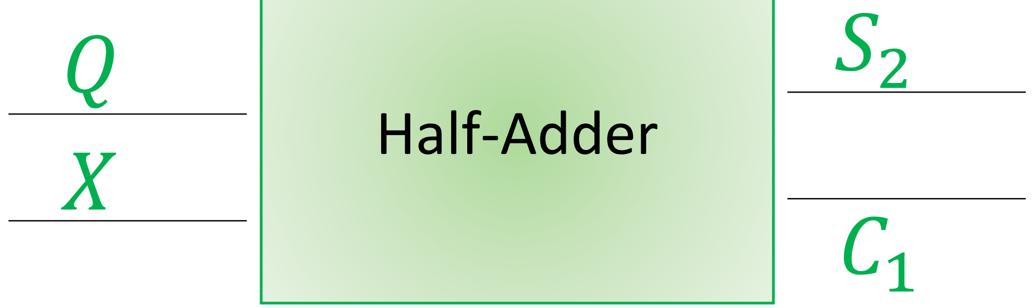
- So, we need a circuit that takes 4 inputs and emits 3 outputs:



Constructing a 2-bit adder

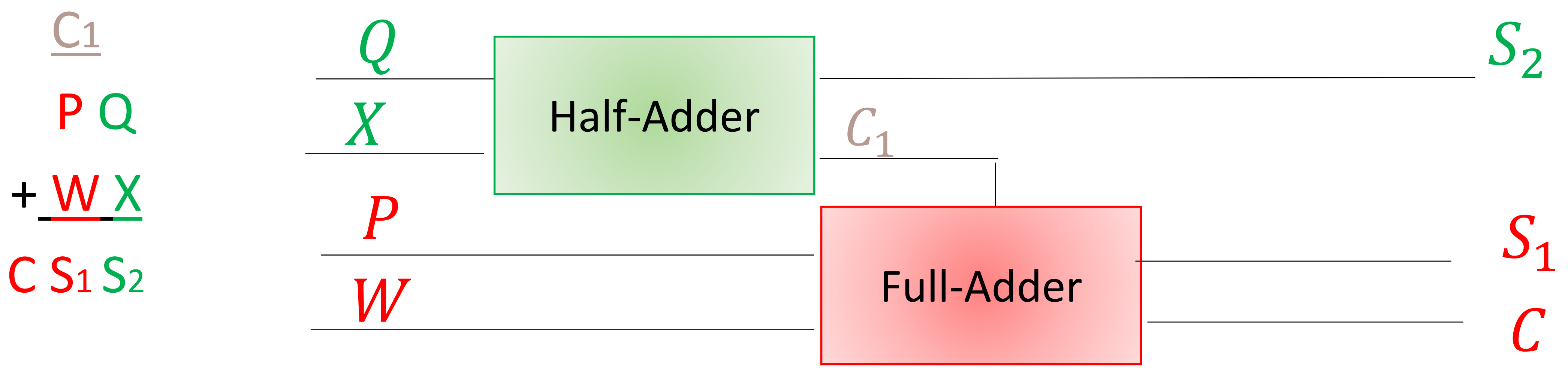
- Step 1: Take care of the right-most column with a half-adder:

$$\begin{array}{r} C_1 \\ P\ Q \\ +\ \underline{W\ X} \\ C\ S_1\ S_2 \end{array}$$



Constructing a 2-bit adder

- Step 1: Take care of the right-most column with a half-adder:

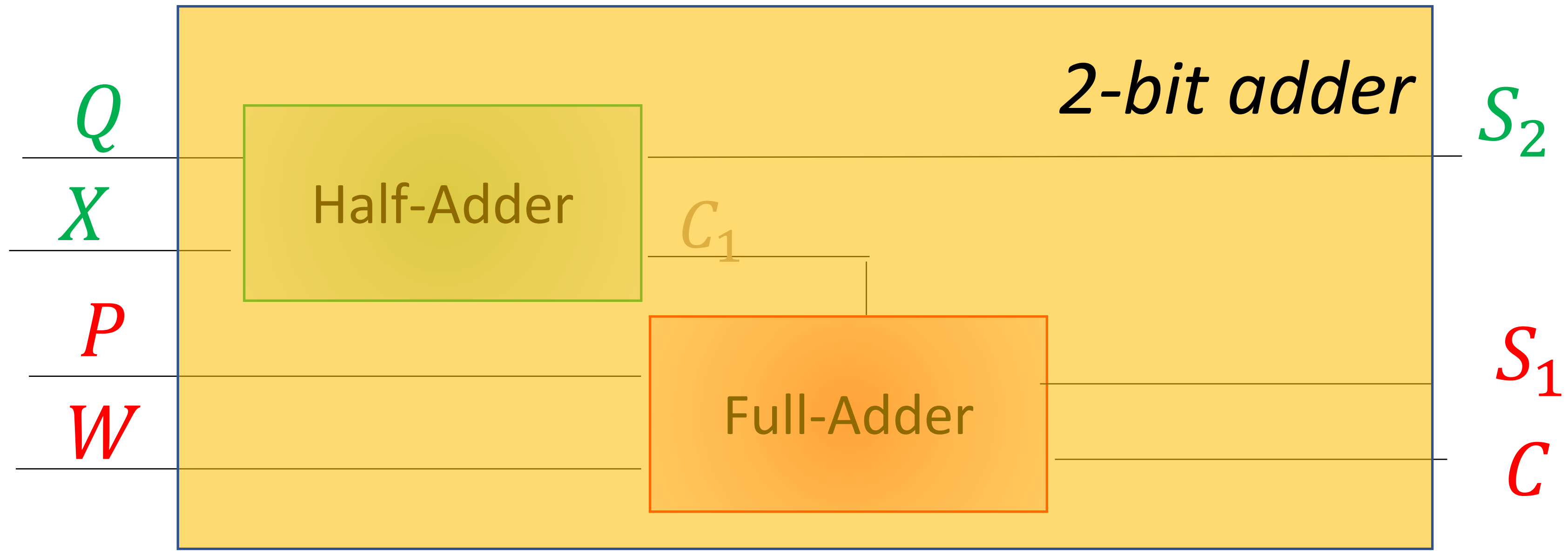


- Step 2 (and final): Connect Half-Adder and new inputs to Full-adder appropriately to produce final circuit.

Constructing a 2-bit adder

- Step 1: Take care of the right-most column with a half-adder:

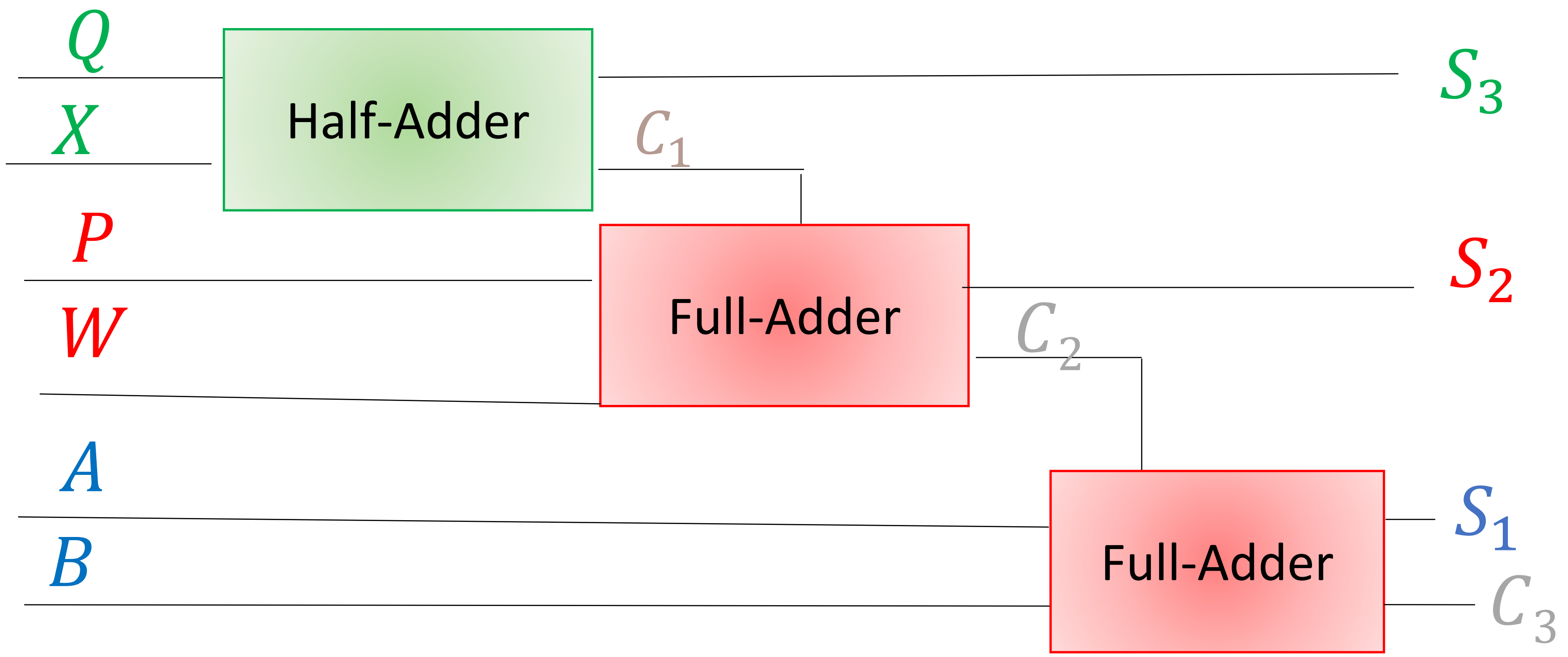
$$\begin{array}{rcc} & \underline{C_1} & \\ & P & Q \\ + & \underline{W} & \underline{X} \\ C & S_1 & S_2 \end{array}$$



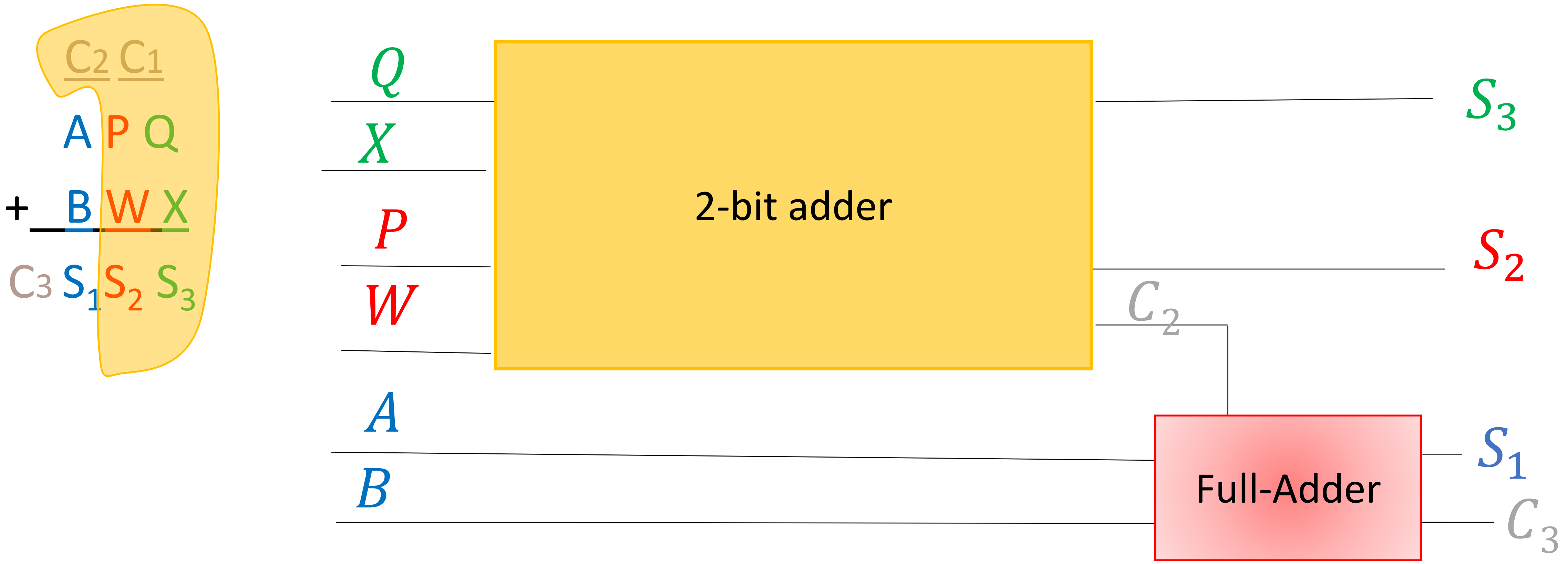
- Step 2 (and final): Connect Half-Adder and new inputs to Full-adder appropriately to produce final circuit.

Constructing a 3-bit adder (messy)

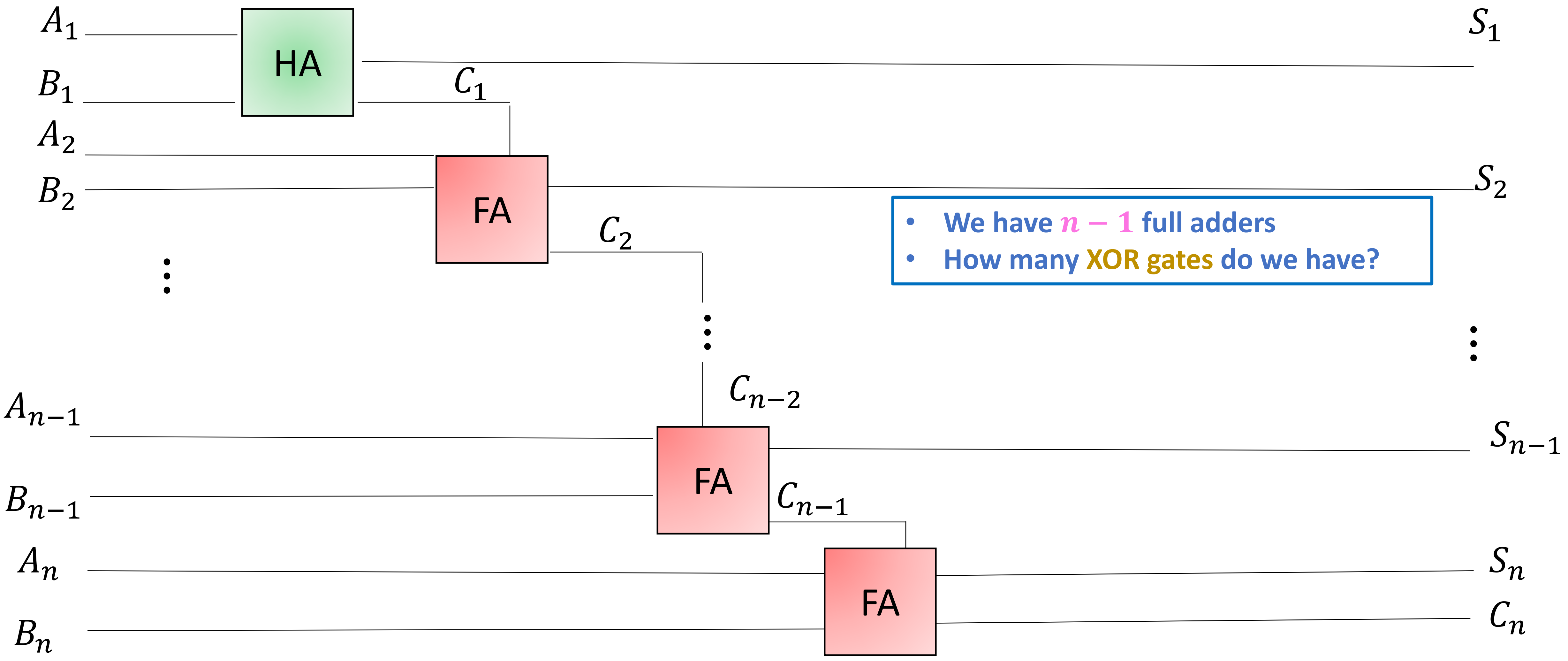
$$\begin{array}{rccc} & \underline{C_2} & \underline{C_1} & \\ & A & P & Q \\ + & \underline{B} & \underline{W} & \underline{X} \\ C_3 & S_1 & S_2 & S_3 \end{array}$$



Constructing a 3-bit adder (neat)

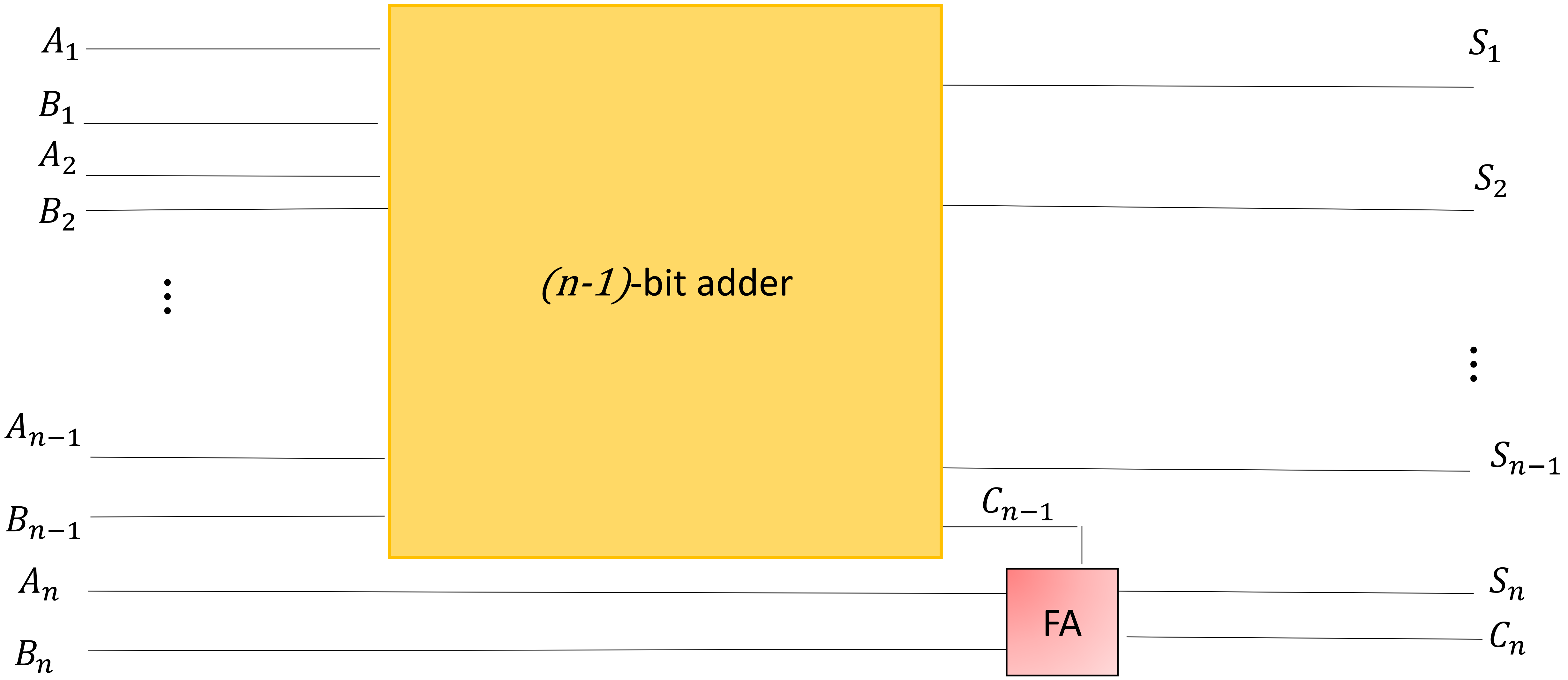


Constructing an n-bit adder (messy)



- We have $n-1$ full adders
- How many XOR gates do we have?

Constructing an n-bit adder (neat)



Other numeric functions

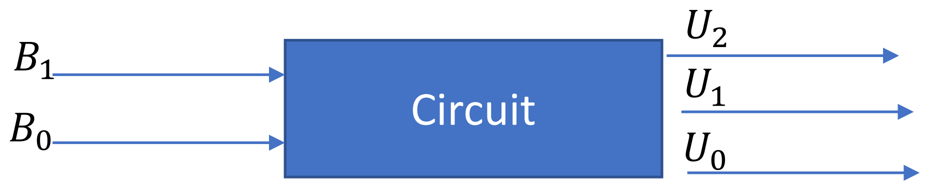
- Addition (have done)
- Multiplication
- Division
- Primality test (test whether a number is prime)

- There are circuits for all of these!
  - Computers actually work this way at the base level: they consist of gates.

Fun exercise

- Input: number in **binary**

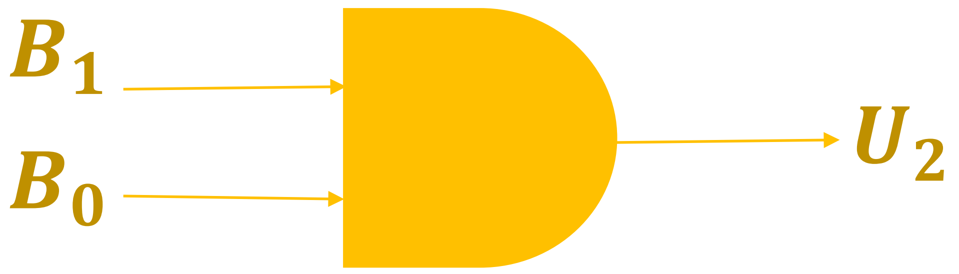
B_1	B_0	U_2	U_1	U_0
0	0	0	0	0
0	1	0	0	1
1	0	0	1	1
1	1	1	1	1



First micro-circuit

B_1	B_0	U_2	U_1	U_0
0	0	0	0	0
0	1	0	0	1
1	0	0	1	1
1	1	1	1	1

$$U_2 = B_1 \wedge B_0$$



Second micro-circuit

B_1	B_0	U_2	U_1	U_0
0	0	0	0	0
0	1	0	0	1
1	0	0	1	1
1	1	1	1	1

$$U_1 = (B_1 \wedge \sim B_0) \vee (B_1 \wedge B_0)$$

Second micro-circuit

B_1	B_0	U_2	U_1	U_0
0	0	0	0	0
0	1	0	0	1
1	0	0	1	1
1	1	1	1	1

$$U_1 = (B_1 \wedge \sim B_0) \vee (B_1 \wedge B_0) = B_1$$

(from distributive law of conjunction over disjunction!)

$$B_1 \xrightarrow{\qquad\qquad} U_1$$

Third micro-circuit

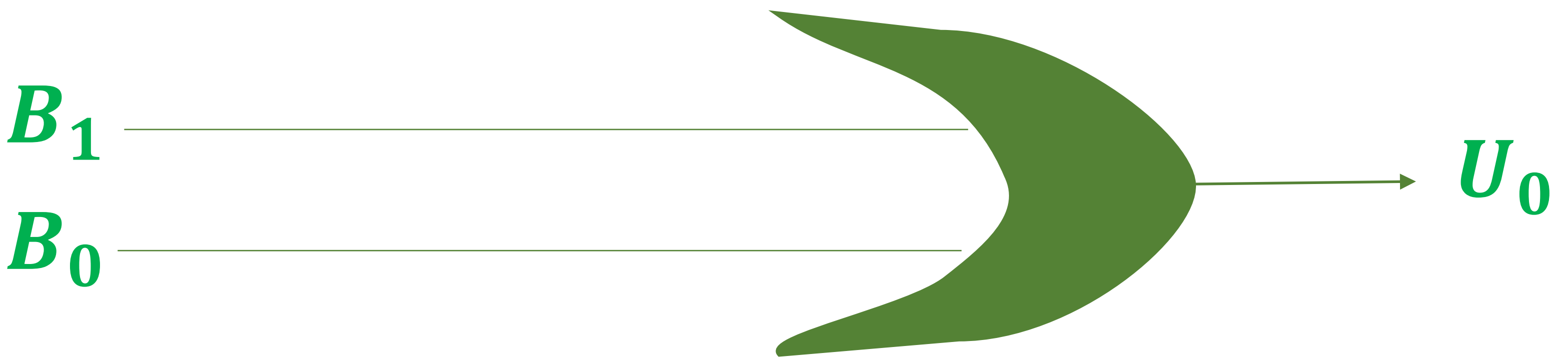
B_1	B_0	U_2	U_1	U_0
0	0	0	0	0
0	1	0	0	1
1	0	0	1	1
1	1	1	1	1

$$U_0 = (\sim B_1 \wedge B_0) \vee (B_1 \wedge \sim B_0) \vee (B_1 \wedge B_0)$$

Third micro-circuit

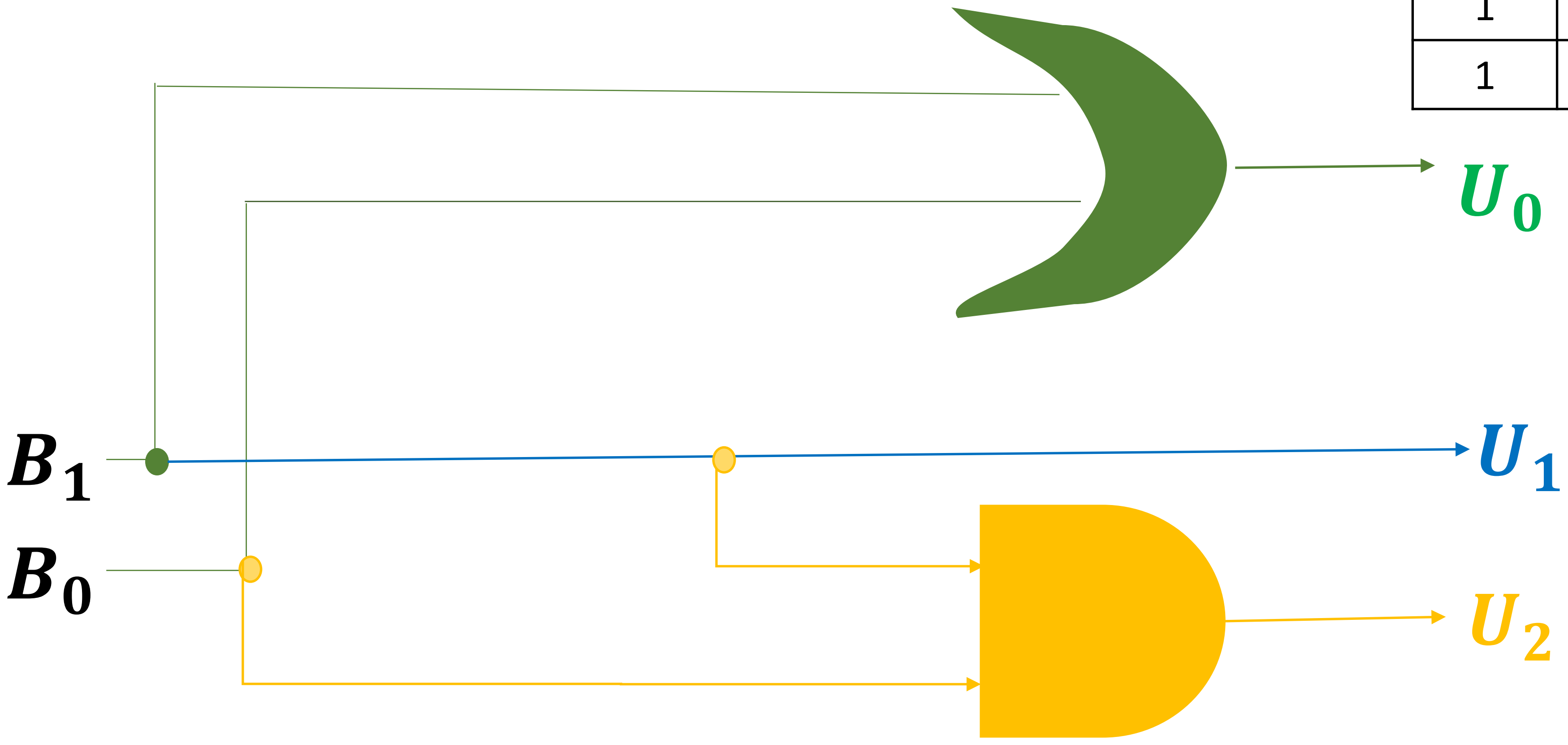
B_1	B_0	U_2	U_1	U_0
0	0	0	0	0
0	1	0	0	1
1	0	0	1	1
1	1	1	1	1

$U_0 = (\sim B_1 \wedge B_0) \vee (B_1 \wedge \sim B_0) \vee (B_1 \wedge B_0) = (\sim B_1 \wedge B_0) \vee B_1 = (\sim B_1 \vee B_1) \wedge (B_0 \vee B_1) = B_0 \vee B_1$



Final circuit

B_1	B_0	U_2	U_1	U_0
0	0	0	0	0
0	1	0	0	1
1	0	0	1	1
1	1	1	1	1



STOP

RECORDING