CSEE 3827: Fundamentals of Computer Systems, Spring 2022

Lecture 4

Prof. Dan Rubenstein (danr@cs.columbia.edu)

Agenda (M&K 3.1-3.12, 8.4)

- **Combinatorial Circuit Design**
  - Multi-bit output functions
  - Standard combinatorial circuits
    - enabler
    - decoder
    - MUX (multiplexer)
    - shifter
    - Code converter
    - Addition / Subtraction
    - half and full adders
    - ripple carry adder
    - carry lookahead adder

Quick Review: K-map with don't care conditions

f(W,X,Y,Z):

← Input → Output

W	X	Y	Z	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

Quick Review: K-map with don't care conditions

f(W,X,Y,Z):

← Input → Output

W	X	Y	Z	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

As Minterms:

$\bar{W}\bar{X}\bar{Y}\bar{Z}$ +

$\bar{W}X\bar{Y}\bar{Z}$ +

$\bar{W}\bar{X}Y\bar{Z}$ +

$\bar{W}XY\bar{Z}$ +

$W\bar{X}\bar{Y}\bar{Z}$ +

$W\bar{X}\bar{Y}Z$ +

Could optionally include Don't Cares

Quick Review: K-map with don't care conditions

f(W,X,Y,Z):

← Input → Output

W	X	Y	Z	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

As Minterms:

$\bar{W}\bar{X}\bar{Y}\bar{Z}$ +

$\bar{W}X\bar{Y}\bar{Z}$ +

$\bar{W}\bar{X}Y\bar{Z}$ +

$\bar{W}XY\bar{Z}$ +

$\bar{W}X\bar{Y}$

$W\bar{X}\bar{Y}\bar{Z}$ +

$W\bar{X}Y\bar{Z}$ +

Could optionally include Don't Cares

$WX\bar{Y}$

Quick Review: K-map with don't care conditions

f(W,X,Y,Z):

← Input → Output

W	X	Y	Z	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

As Minterms:

$\bar{W}\bar{X}\bar{Y}\bar{Z}$ +

$\bar{W}X\bar{Y}\bar{Z}$ +
$\bar{W}X\bar{Y}Z$ +
$\bar{W}XY\bar{Z}$ +

$\bar{W}X\bar{Y}$

$W\bar{X}\bar{Y}\bar{Z}$ +
$W\bar{X}\bar{Y}Z$ +

$X\bar{Y}$

Could optionally include Don't Cares

$WX\bar{Y}$

Quick Review: K-map with don't care conditions

f(W,X,Y,Z):

← Input → Output

W	X	Y	Z	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

As Minterms:

$\bar{W}\bar{X}\bar{Y}\bar{Z}$ +

$\bar{W}X\bar{Y}\bar{Z}$ +
$\bar{W}X\bar{Y}Z$ +
$\bar{W}XY\bar{Z}$ +

$\bar{W}X\bar{Y}$

$W\bar{X}\bar{Y}\bar{Z}$ +
$W\bar{X}\bar{Y}Z$ +

$X\bar{Y}$

Could optionally include Don't Cares

$WX\bar{Y}$

What about X?

covers a 0, can't use as product term

Quick Review: K-map with don't care conditions

f(W,X,Y,Z):

W	X	Y	Z	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

Input → Output

As Minterms:

$\bar{W}\bar{X}\bar{Y}\bar{Z}$ +

$\bar{W}X\bar{Y}\bar{Z}$ +
$\bar{W}X\bar{Y}Z$ +
$\bar{W}XY\bar{Z}$ +

$\bar{W}X\bar{Y}$

$W\bar{X}\bar{Y}\bar{Z}$ +
$W\bar{X}\bar{Y}Z$ +

$X\bar{Y}$

Could optionally include Don't Cares

$WX\bar{Y}$

What about $\bar{Y}$?

covers a 0, can't use as product term

Quick Review: K-map with don't care conditions

f(W,X,Y,Z):

← Input → Output

W	X	Y	Z	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

As Minterms:

$\bar{W}\bar{X}\bar{Y}\bar{Z}$ +

$\bar{W}\bar{X}\bar{Y}\bar{Z}$ +
$\bar{W}\bar{X}\bar{Y}Z$ +
$\bar{W}\bar{X}Y\bar{Z}$ +

$\bar{W}\bar{X}\bar{Y}$

$\bar{W}\bar{X}\bar{Y}\bar{Z}$ +
$\bar{W}X\bar{Y}\bar{Z}$ +

$X\bar{Y}$

Could optionally include Don't Cares

$WX\bar{Y}$

What about $\bar{Y}$?

covers a 0, can't use as product term

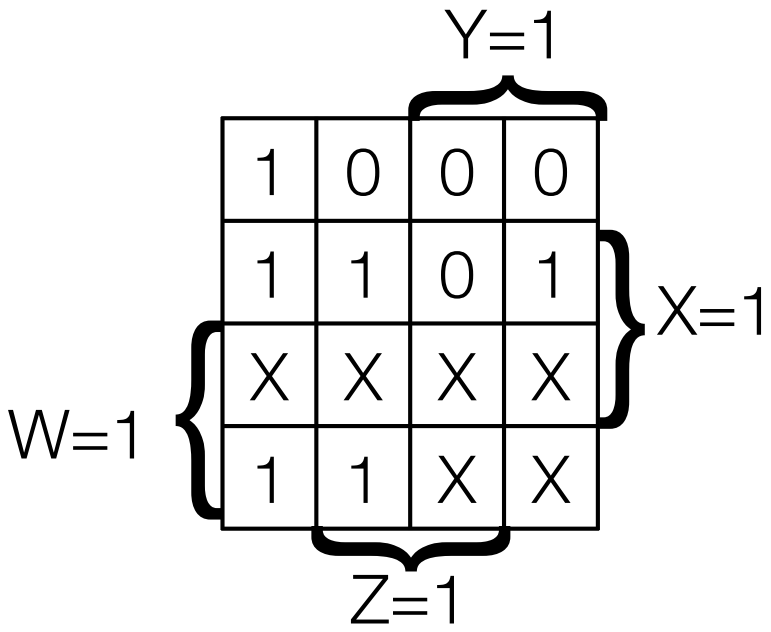
$X\bar{Y}$ is a prime implicant

Quick Review: K-map with don't care conditions

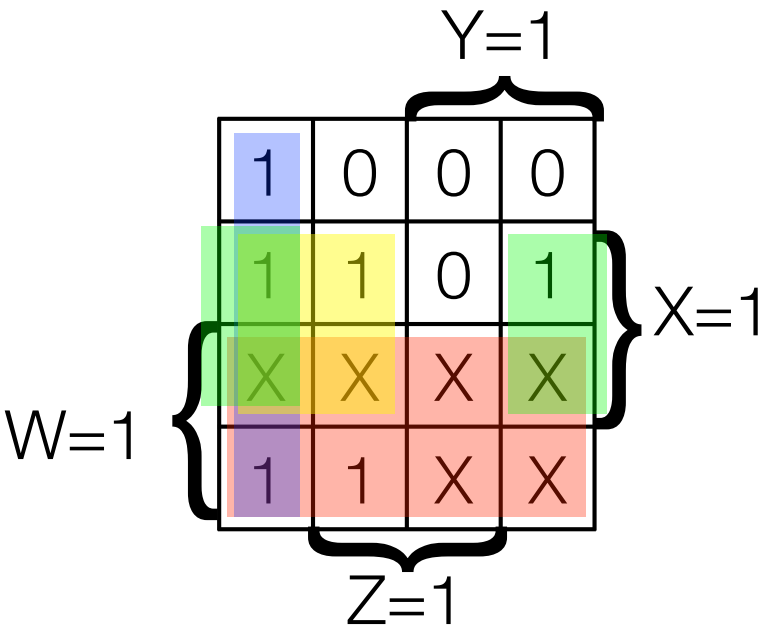
f(W,X,Y,Z):

← Input → Output

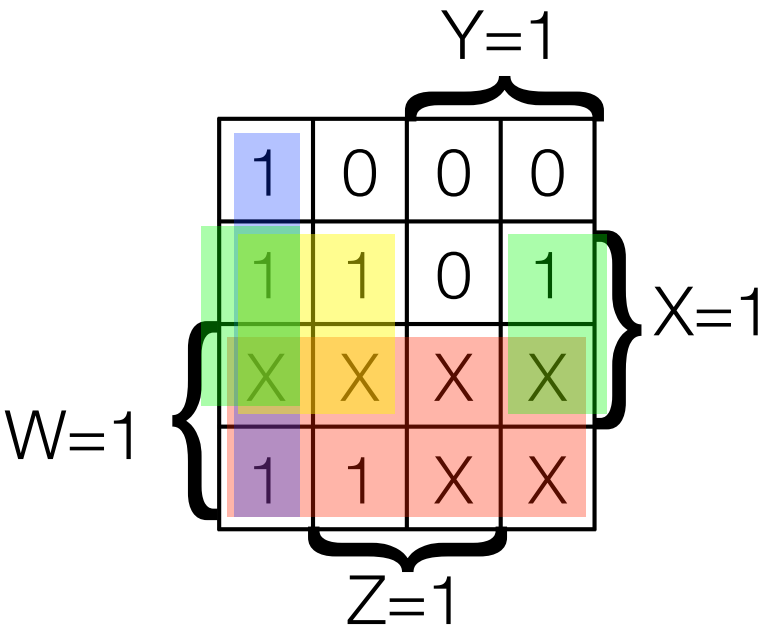
W	X	Y	Z	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X



Algebra and Circuit for “f”

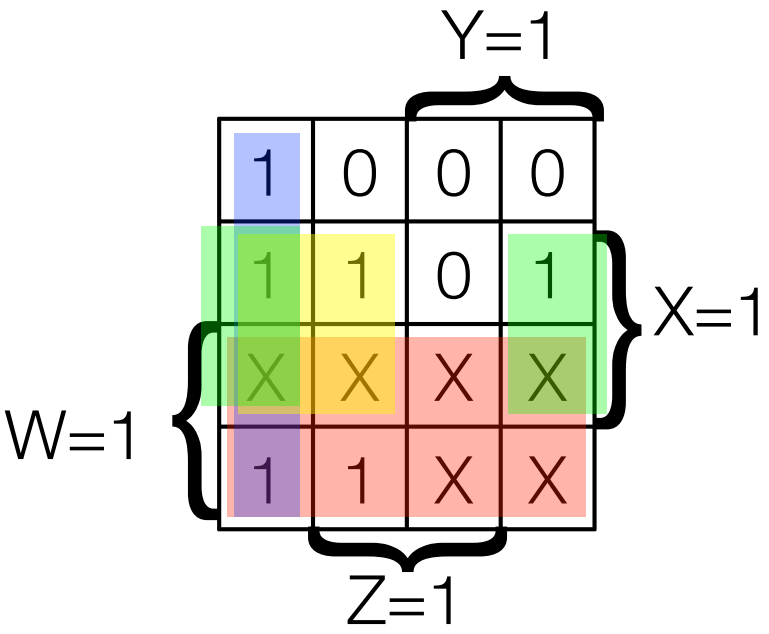


Algebra and Circuit for “f”

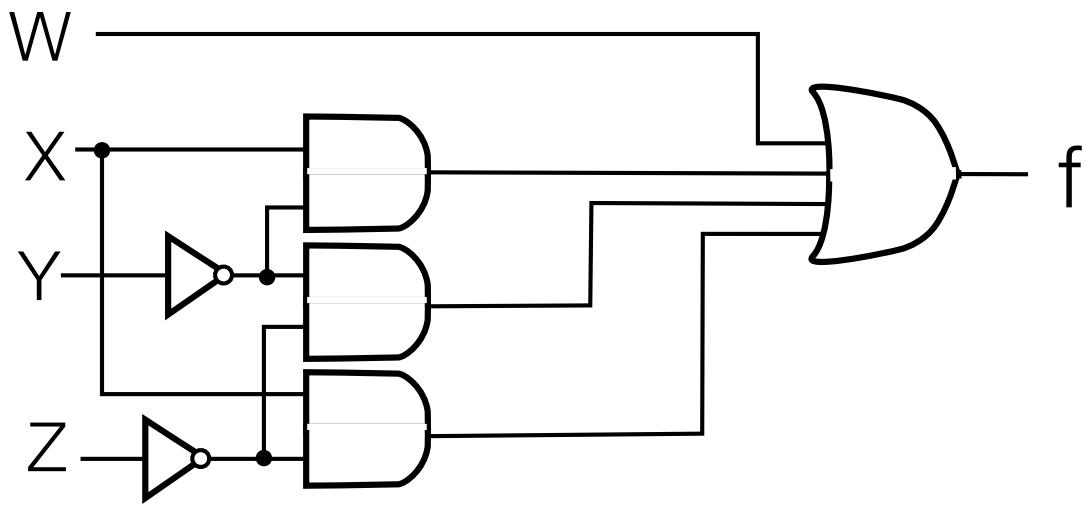


$$f = W + X\bar{Y} + \bar{Y}\bar{Z} + X\bar{Z}$$

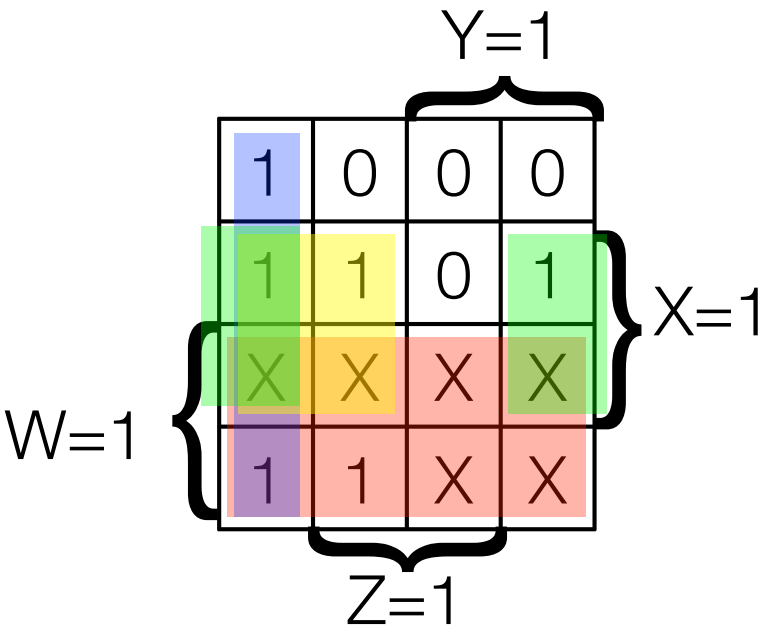
Algebra and Circuit for “f”



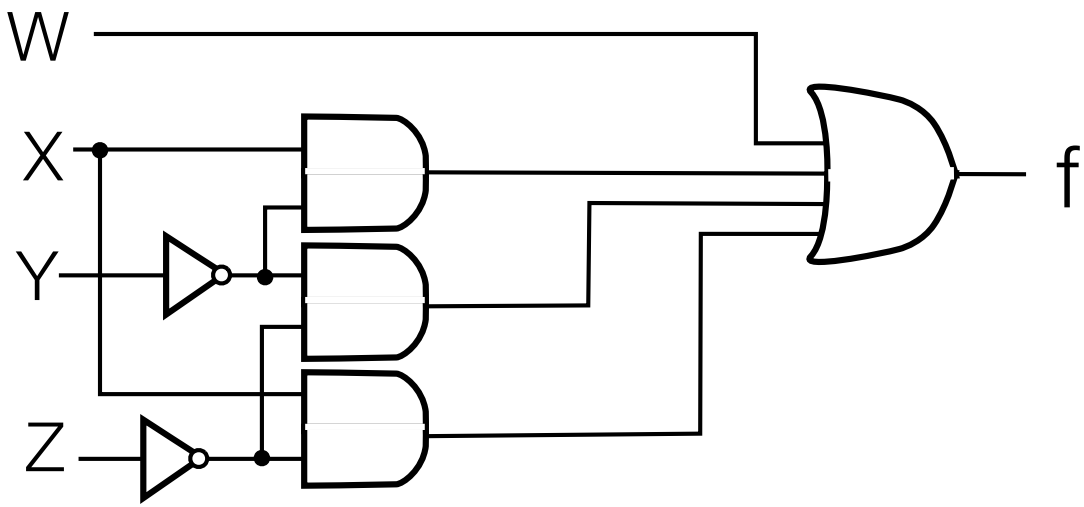
$$f = W + X\bar{Y} + \bar{Y}\bar{Z} + X\bar{Z}$$



Algebra and Circuit for “f”



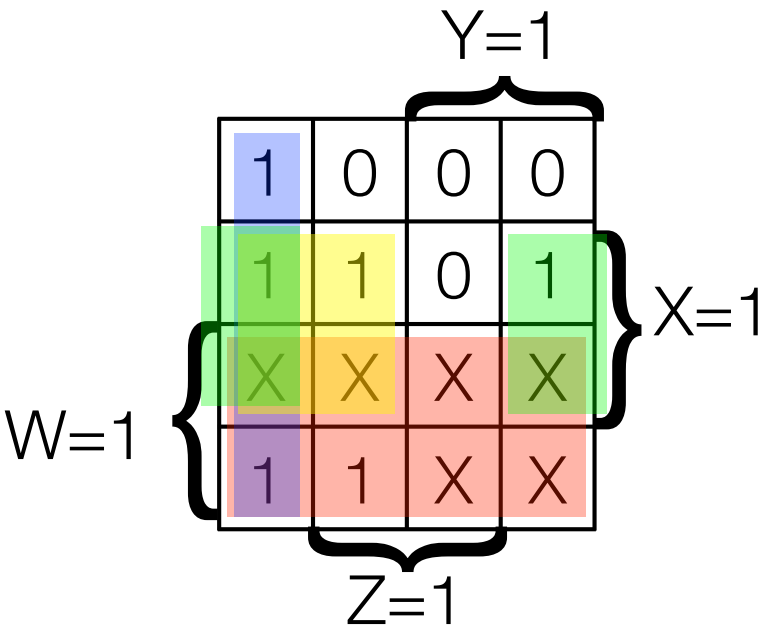
$f = W + X\bar{Y} + \bar{Y}\bar{Z} + X\bar{Z}$



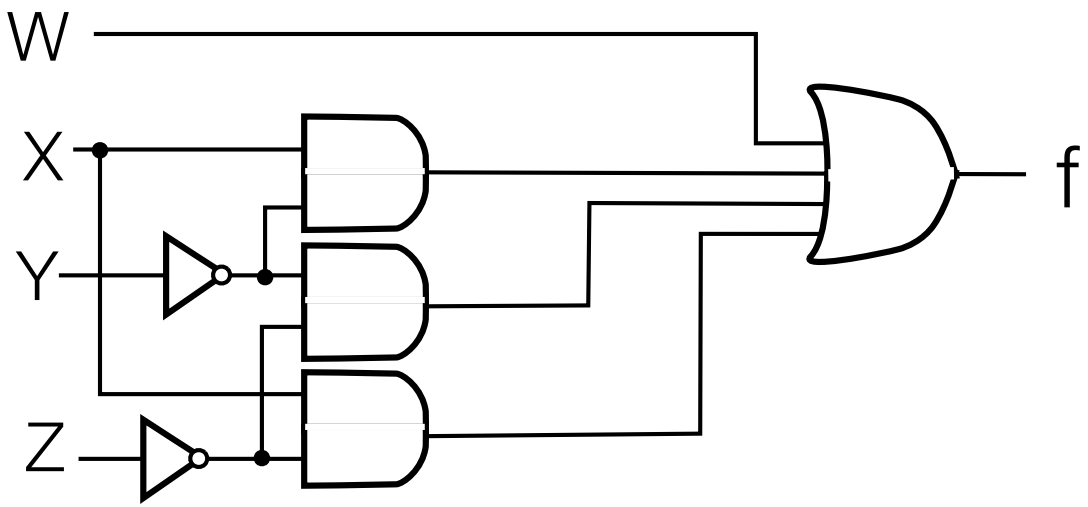
Note $X\bar{Y}$ is essential PI: only PI that covered $\bar{W}X\bar{Y}Z$

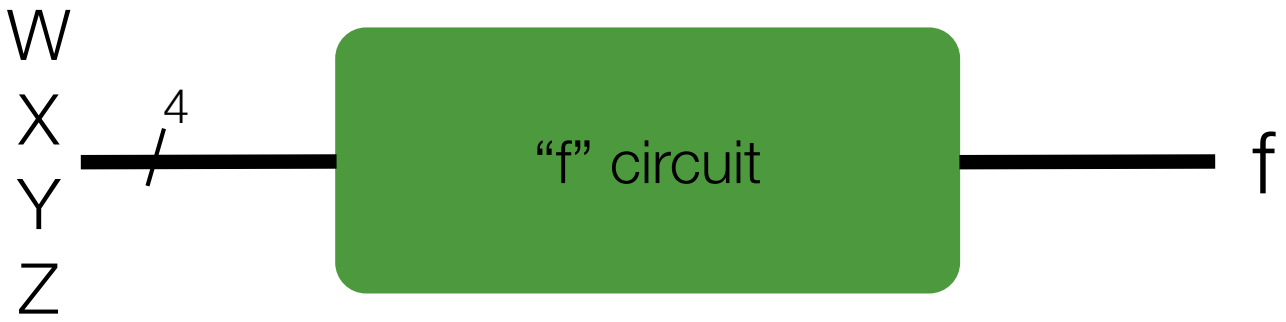
So are the other terms (W, $\bar{Y}\bar{Z}$, $X\bar{Z}$)

Algebra and Circuit for “f”



$$f = W + X\bar{Y} + \bar{Y}\bar{Z} + X\bar{Z}$$

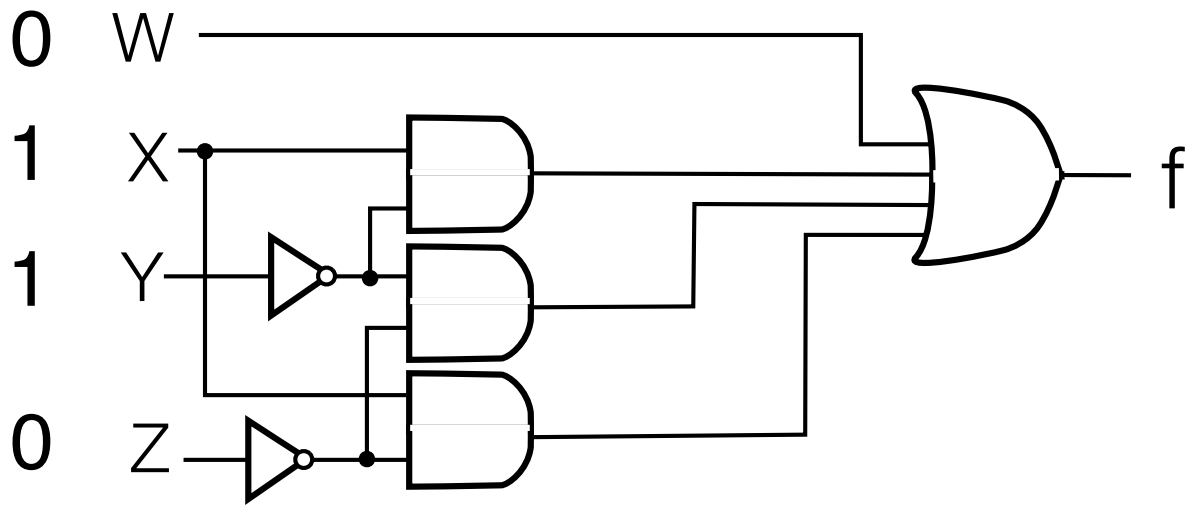




When we know the internals, we can simplify our picture of the circuit (i.e., with a labelled box)

Input Flow… circuit needs time to stabilize

$$f = W + X\bar{Y} + \bar{Y}\bar{Z} + X\bar{Z}$$

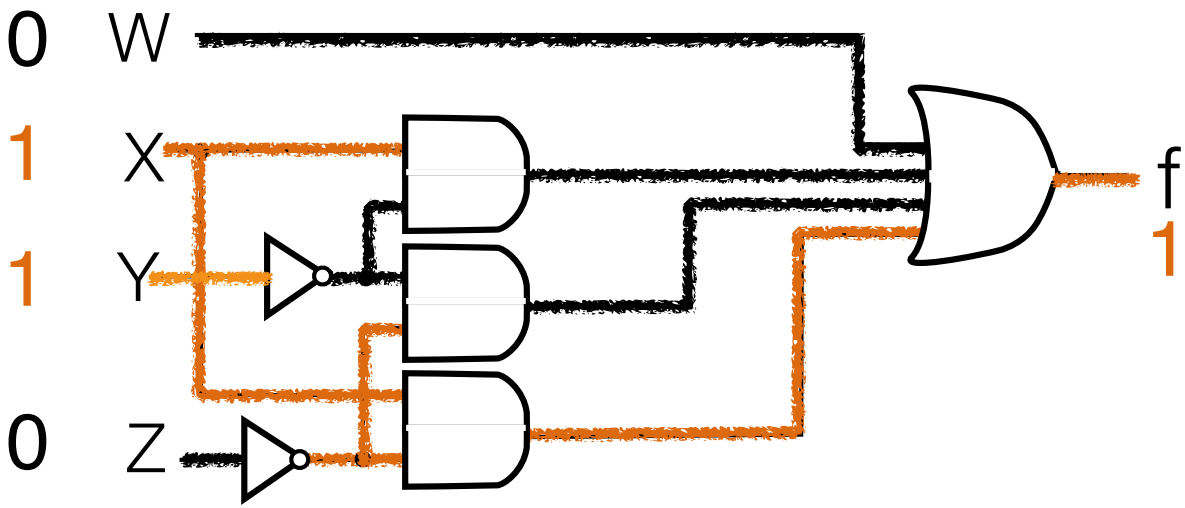


Input Flow… circuit needs time to stabilize

Wire carries “0”

Wire carries “1”

$f = W + X\bar{Y} + \bar{Y}\bar{Z} + X\bar{Z}$

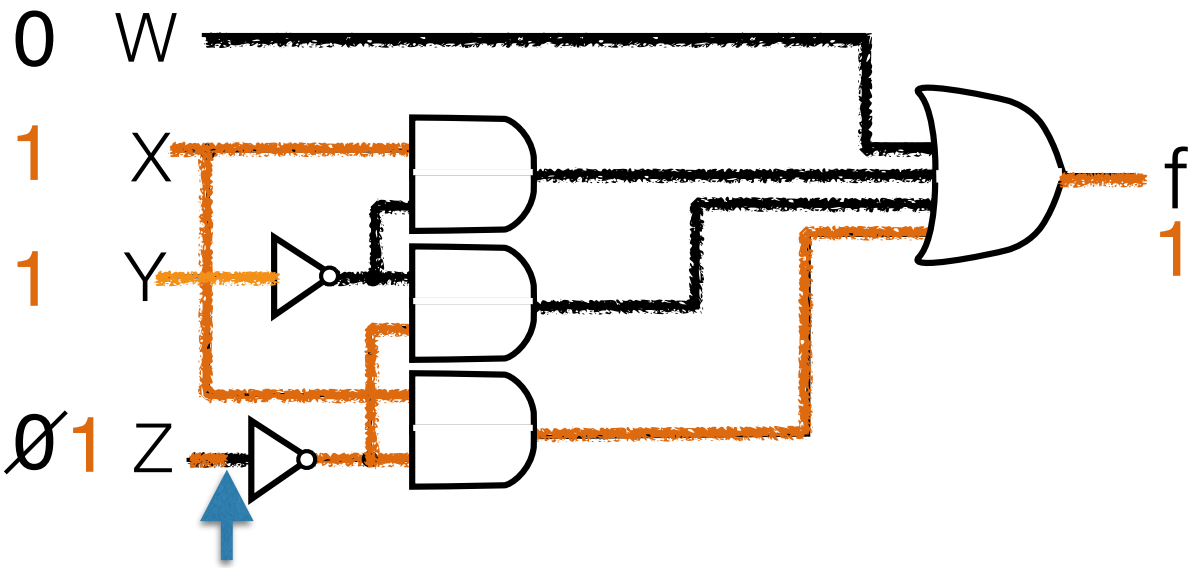


Input Flow… circuit needs time to stabilize

Wire carries “0”

Wire carries “1”

$f = W + X\bar{Y} + \bar{Y}\bar{Z} + X\bar{Z}$



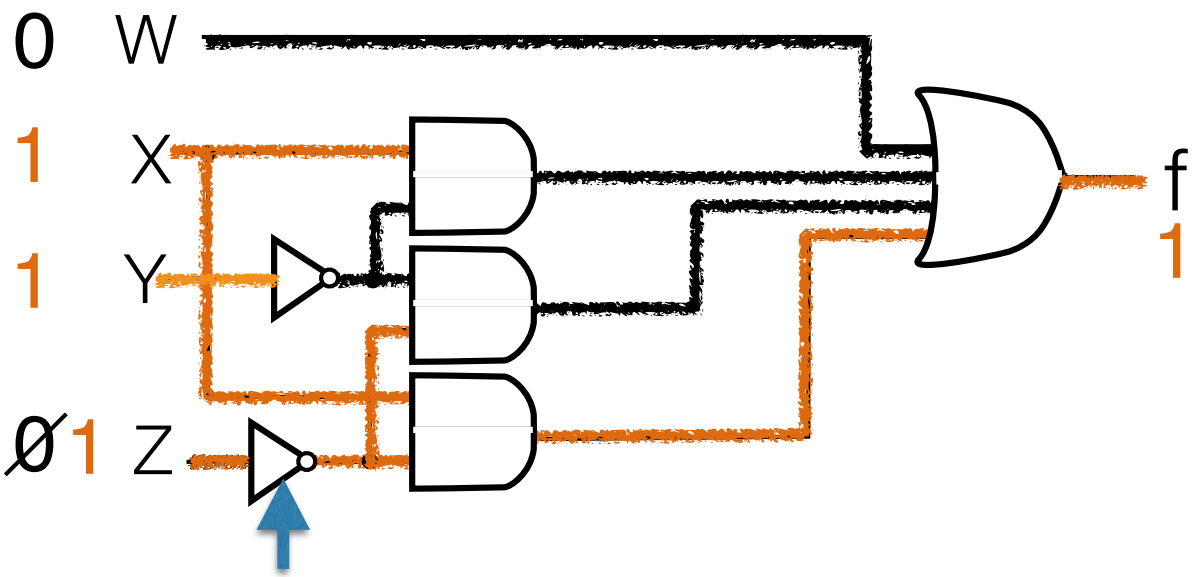
Updated info (Z=1) starts to flow quickly down wire (speed of light)…

Input Flow… circuit needs time to stabilize

Wire carries “0”

Wire carries “1”

$f = W + X\bar{Y} + \bar{Y}\bar{Z} + X\bar{Z}$



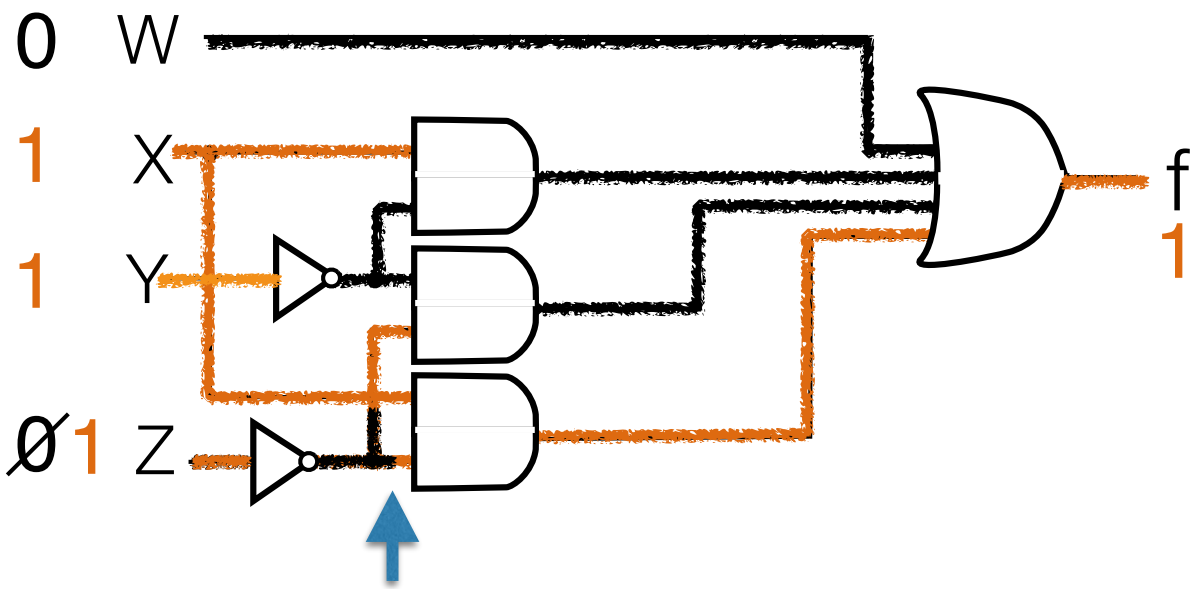
Reaches NOT gate, takes a bit of time to come out the other side…

Input Flow… circuit needs time to stabilize

Wire carries “0”

Wire carries “1”

$f = W + X\bar{Y} + \bar{Y}\bar{Z} + X\bar{Z}$



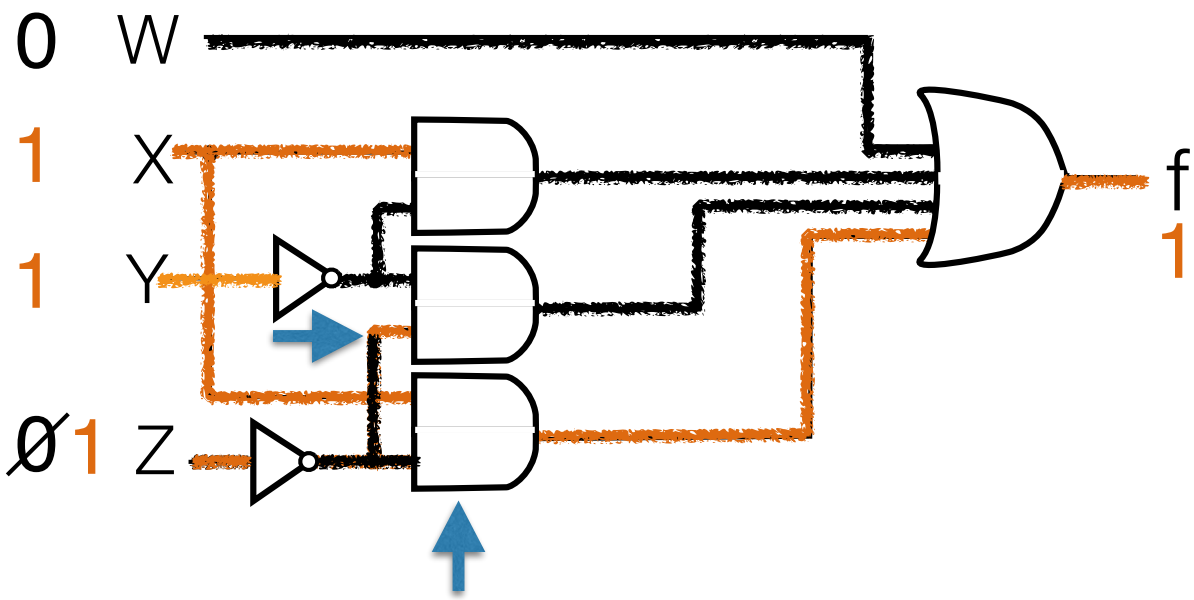
Down the next wire (quickly)

Input Flow… circuit needs time to stabilize

Wire carries “0”

Wire carries “1”

$f = W + X\bar{Y} + \bar{Y}\bar{Z} + X\bar{Z}$



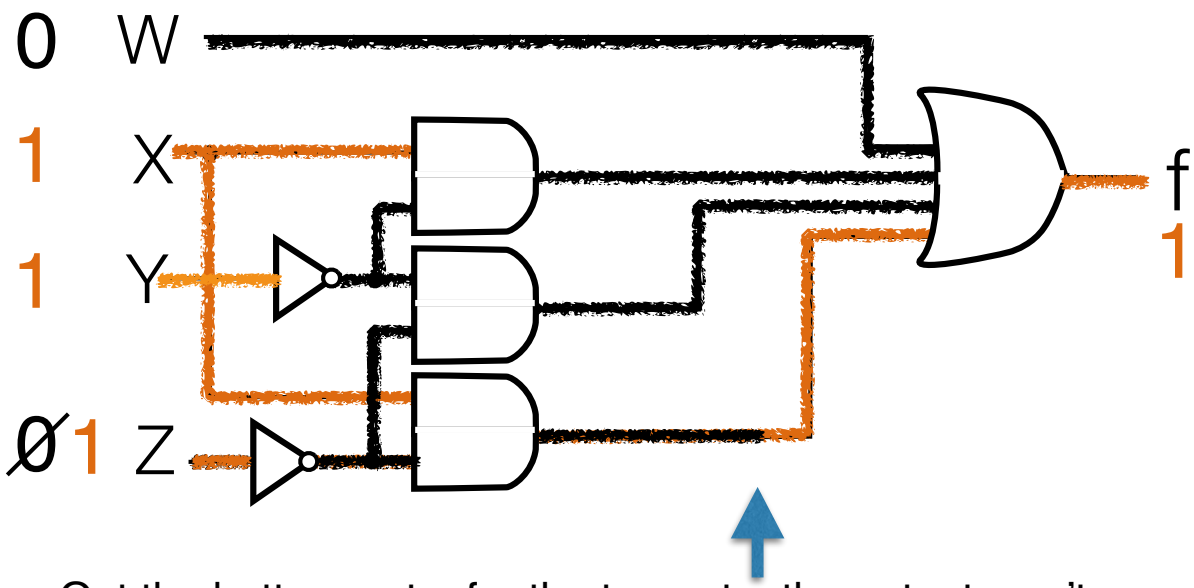
In one gate (for a while), almost to other gate

Input Flow… circuit needs time to stabilize

Wire carries “0”

Wire carries “1”

$f = W + X\bar{Y} + \bar{Y}\bar{Z} + X\bar{Z}$



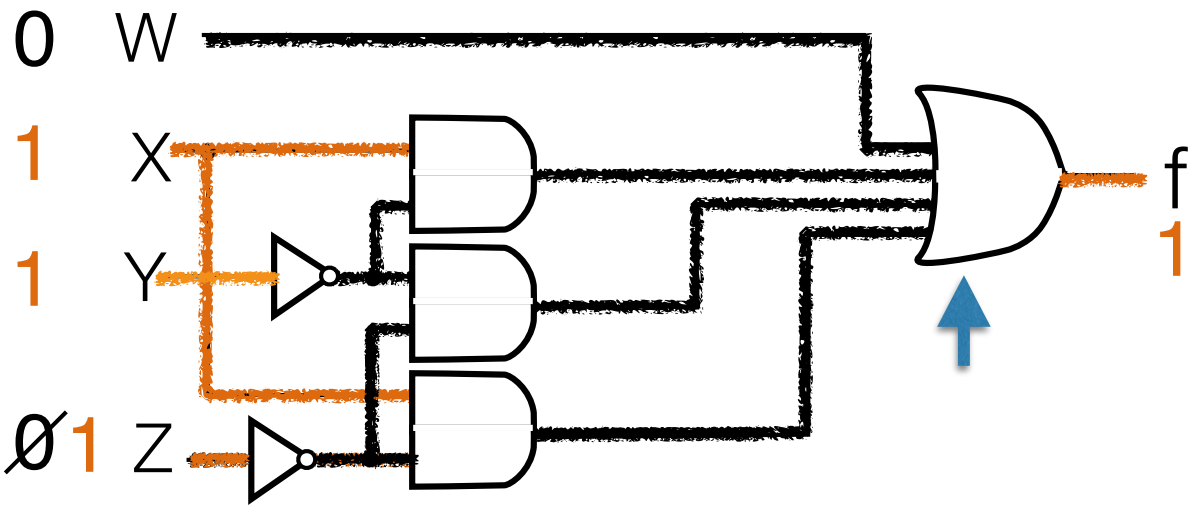
Out the bottom gate, for the top gate, the output won’t change (output stays 0)

Input Flow… circuit needs time to stabilize

Wire carries “0”

Wire carries “1”

$f = W + X\bar{Y} + \bar{Y}\bar{Z} + X\bar{Z}$



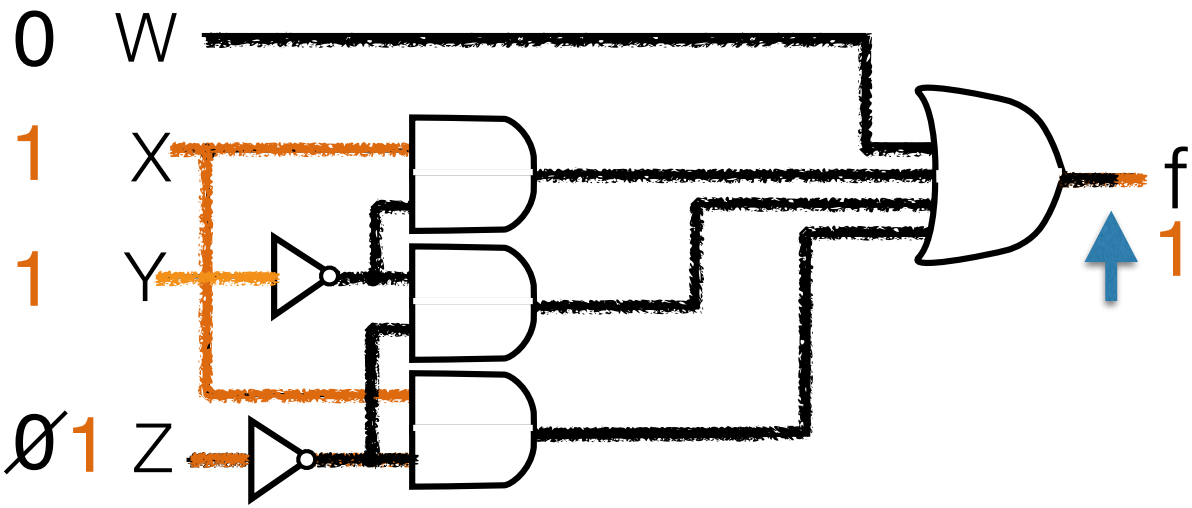
Enters last gate in the sequence dependent on Z takes some time (relative to wire speed)

Input Flow… circuit needs time to stabilize

Wire carries “0”

Wire carries “1”

$f = W + X\bar{Y} + \bar{Y}\bar{Z} + X\bar{Z}$



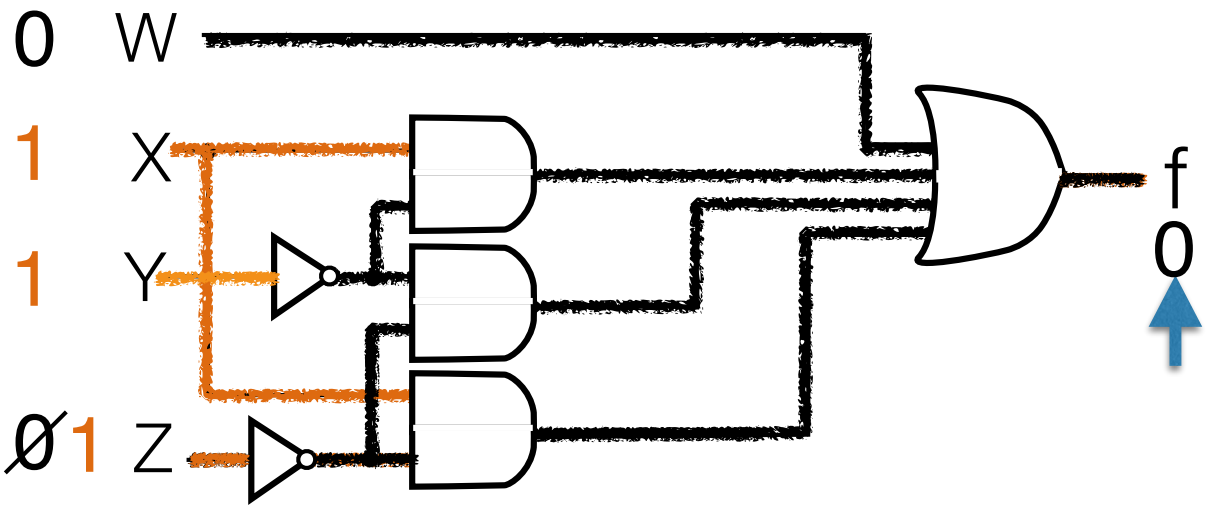
Finally out the final gate, output soon to change…

Input Flow… circuit needs time to stabilize

Wire carries “0”

Wire carries “1”

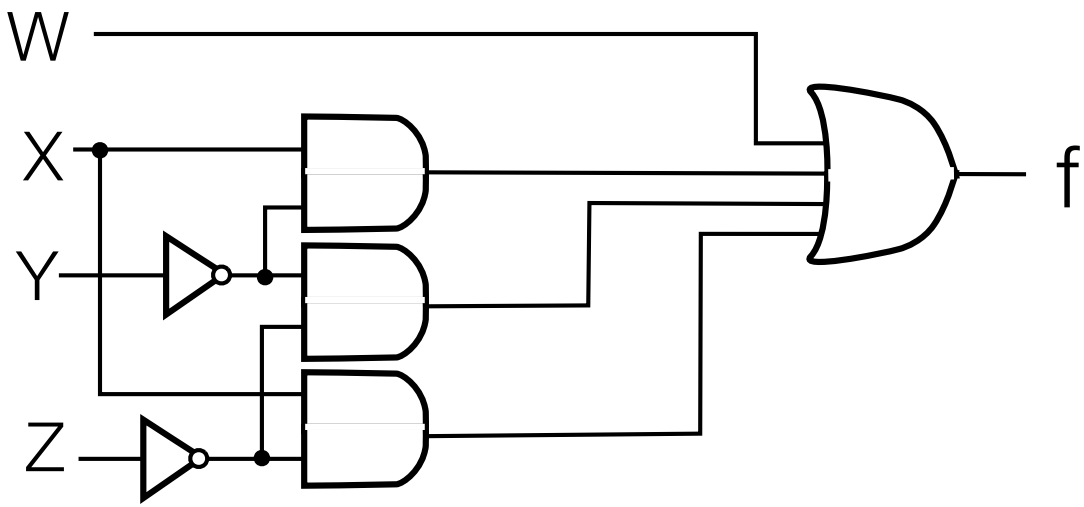
$f = W + X\bar{Y} + \bar{Y}\bar{Z} + X\bar{Z}$

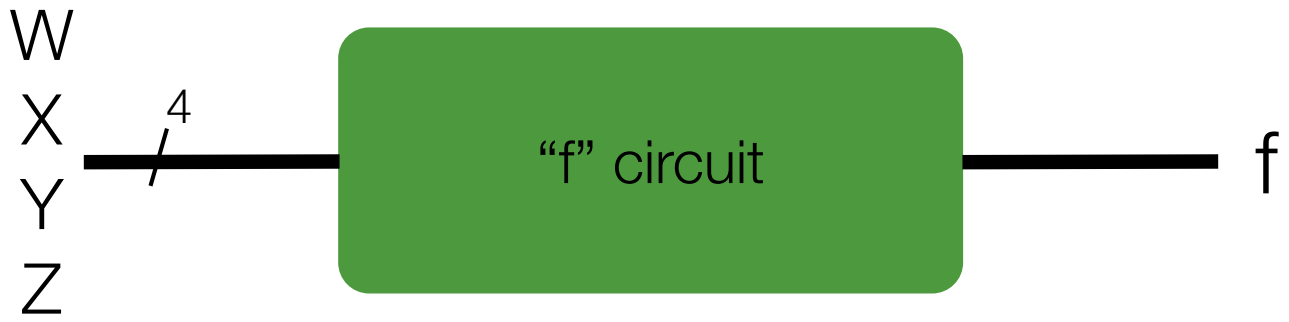


Circuit has “stabilized” - output corresponds to f(inputs)

Algebra and Circuit for “f”

$$f = W + X\bar{Y} + \bar{Y}\bar{Z} + X\bar{Z}$$





When we use this “box” simplifying representation, still need to understand how changing inputs takes time to propagate to output.

Combinational circuits

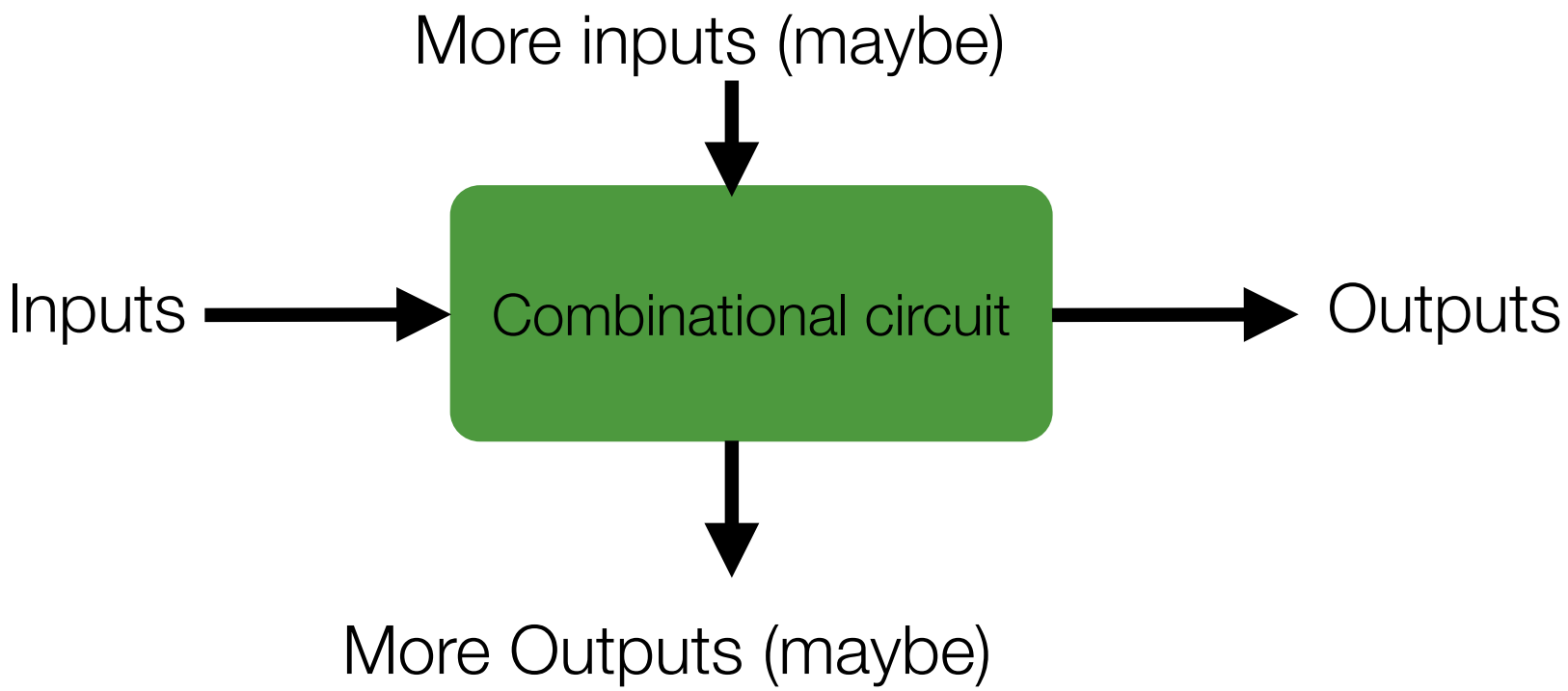
- Combinational circuits are stateless

- The outputs are functions only of the inputs

Combinational circuits

- Combinational circuits are stateless

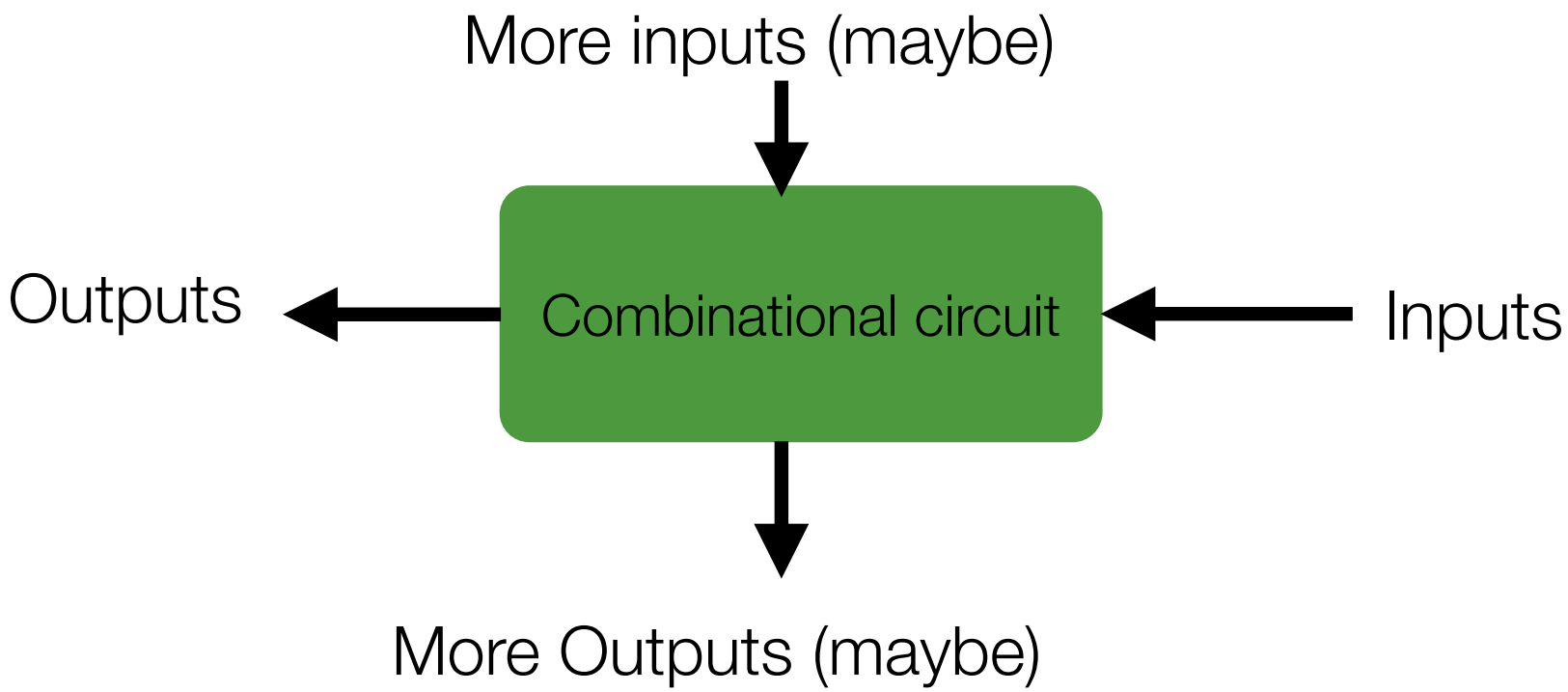
- The outputs are functions only of the inputs



Combinational circuits

- Combinational circuits are stateless

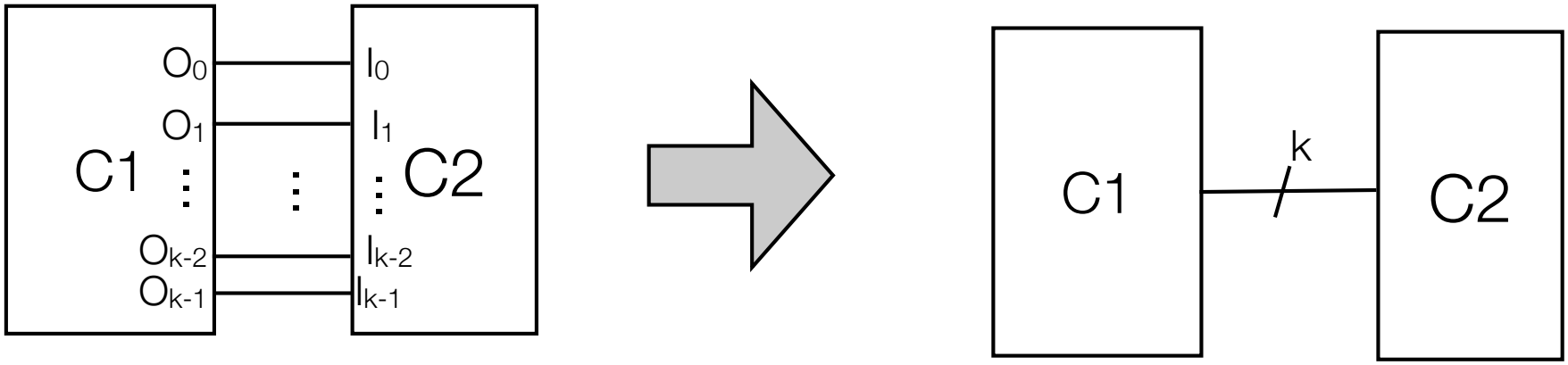
- The outputs are functions only of the inputs



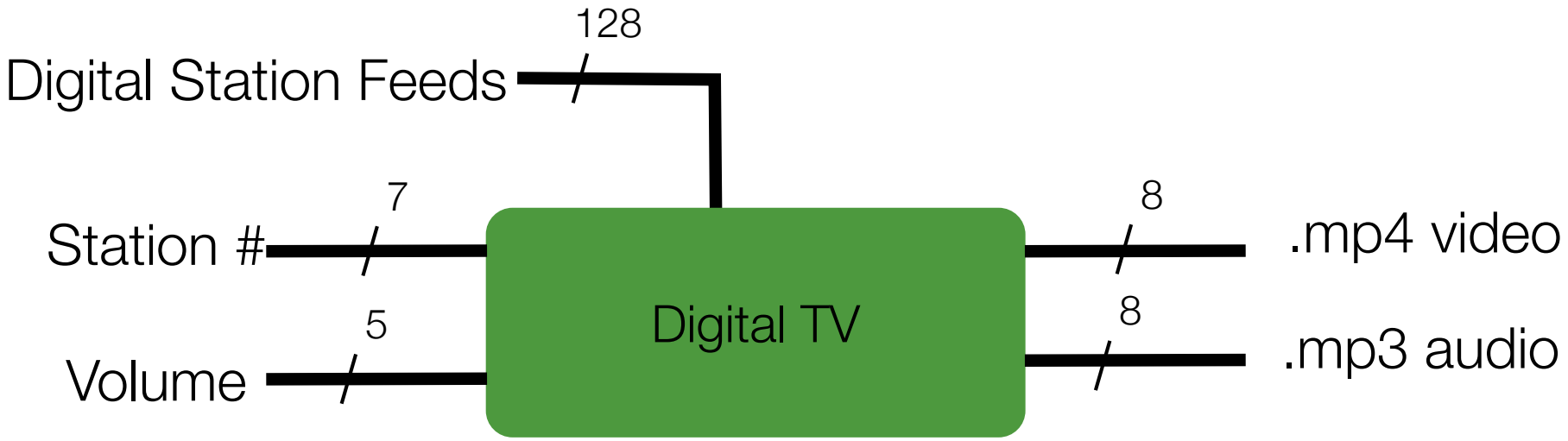
You can tell what are inputs and what are outputs either by the arrow directions, or by context (when obvious)

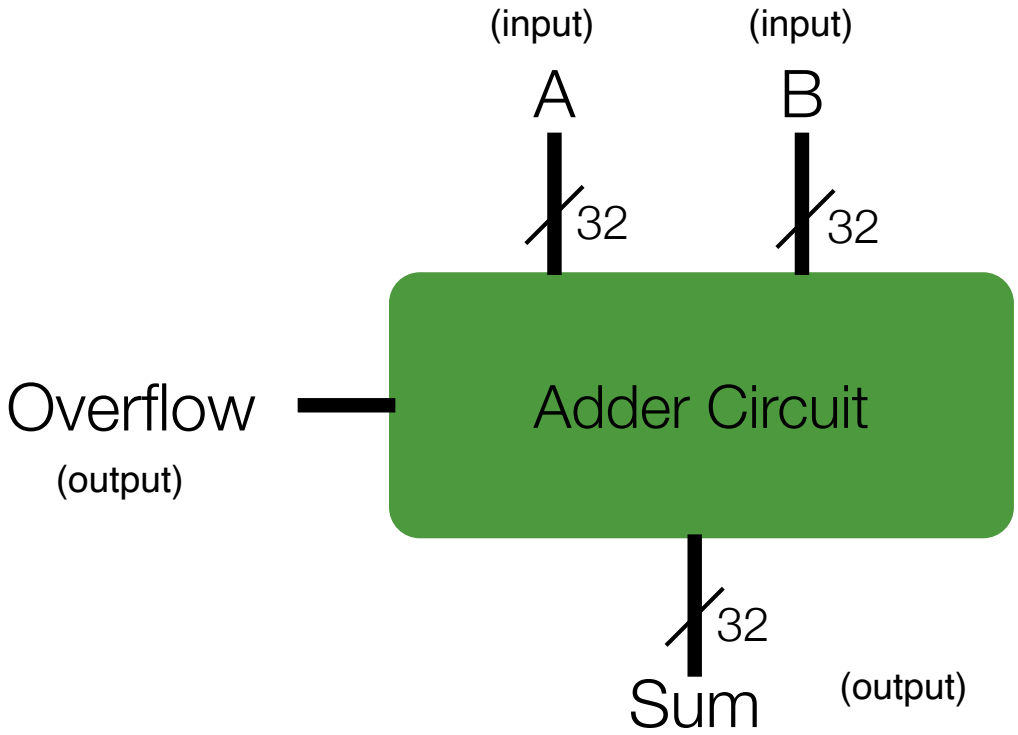
Review: Parallel Multi-wire notation

- Useful when running a bunch of bits in parallel to the same (similar place)



Multi Input and/or Output Circuits





Sum = A+B (mod 2^{32}),
Overflow is overflow bit

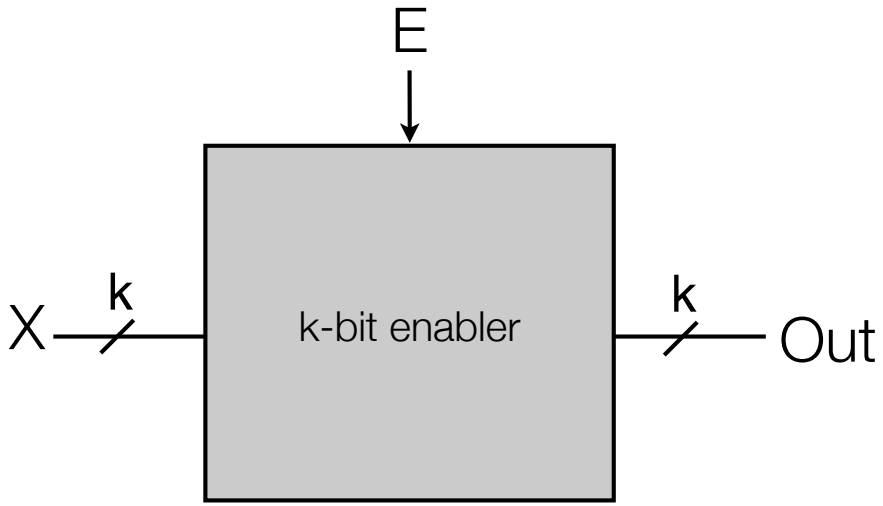
“Standard” Circuits

“Standard Circuits”

- A few types of circuits you will need to become very familiar with
  - how to build
  - how to use
- They are:
  - Enabler
  - Decoder
  - MUX (multiplexer) (also DeMux)
  - (General Purpose) Code Converter
  - Shifter
  - Adder/Subtractor

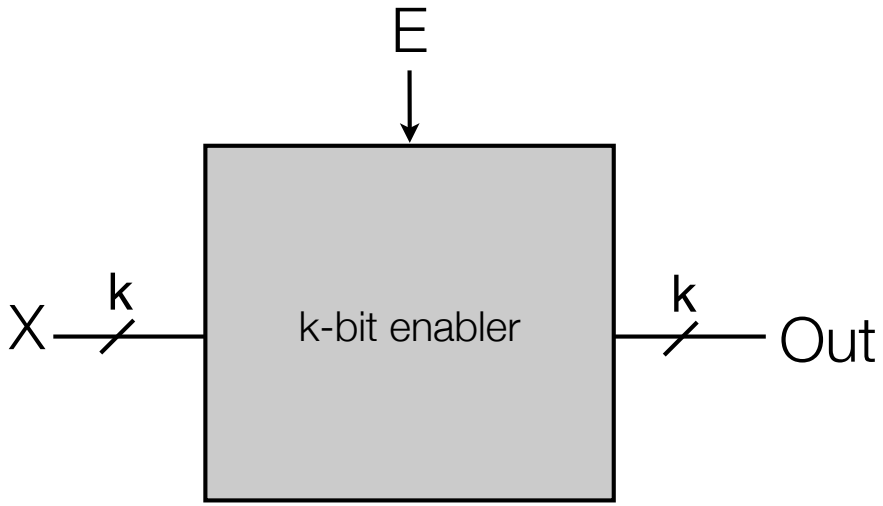
Enabler Circuit (High-level view):

- Enabler circuit has 2 inputs
  - data (can be k bits in parallel)
  - enable/disable (i.e., on/off) input
- Output:
  - If E=1, output = data input
  - If E=0, output = k 0's

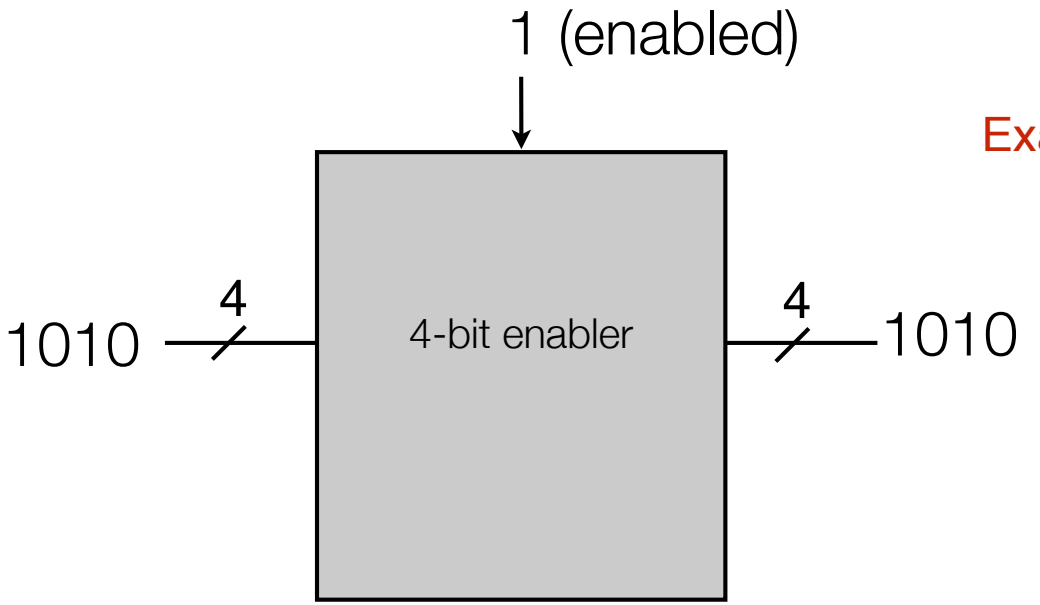


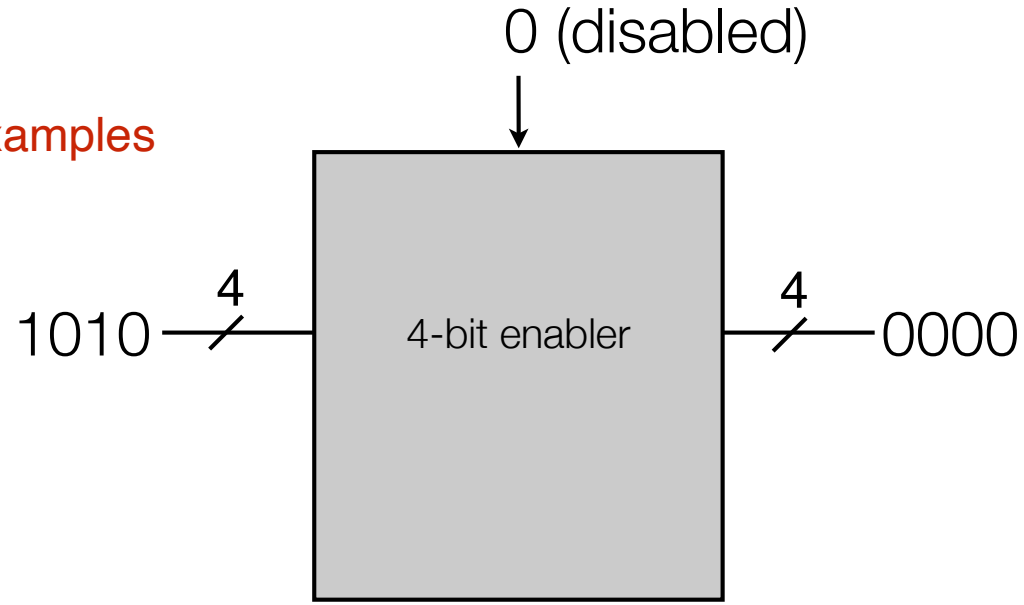
Enabler Circuit (High-level view):

- Enabler circuit has 2 inputs
  - data (can be k bits in parallel)
  - enable/disable (i.e., on/off) input
- Output:
  - If E=1, output = data input
  - If E=0, output = k 0's



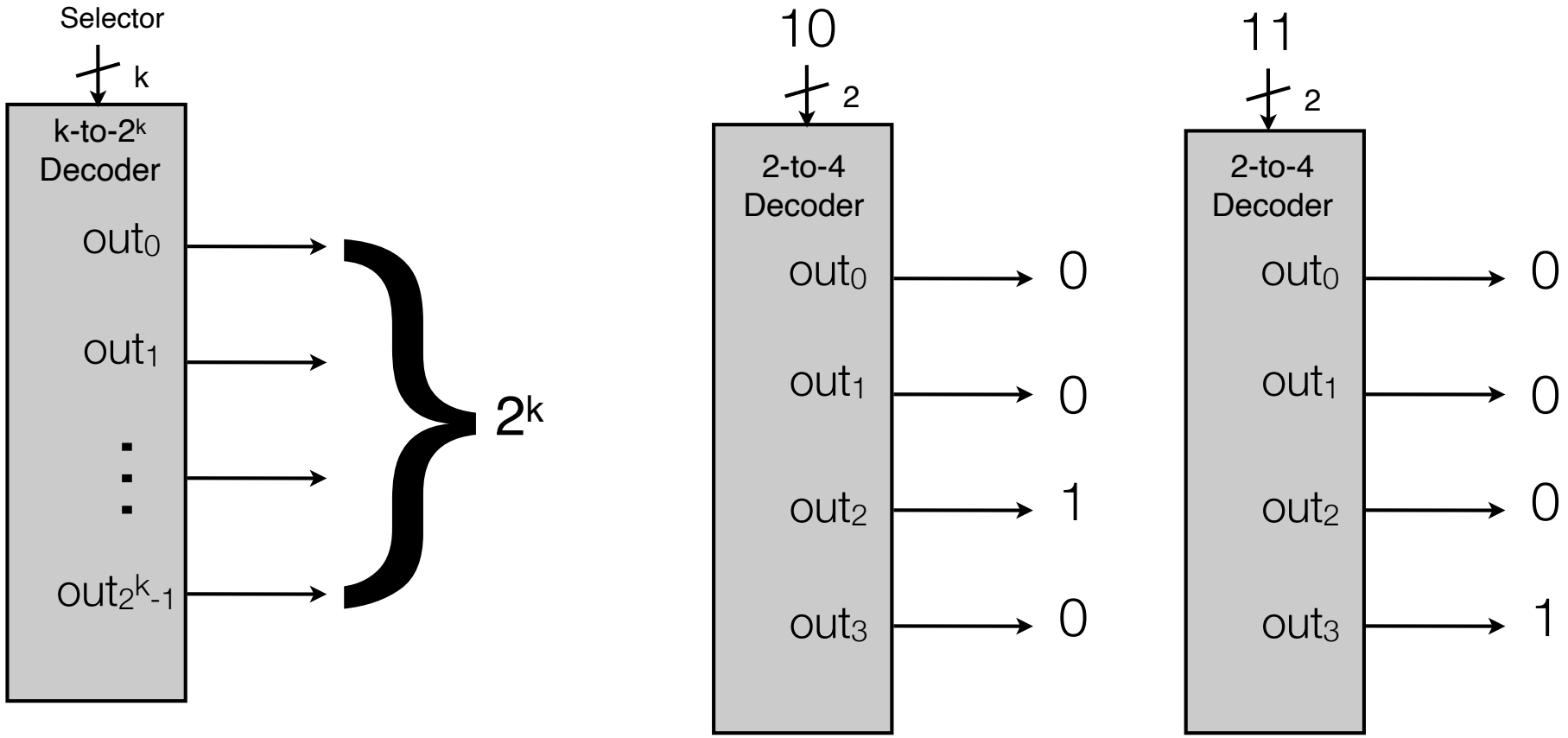
Examples





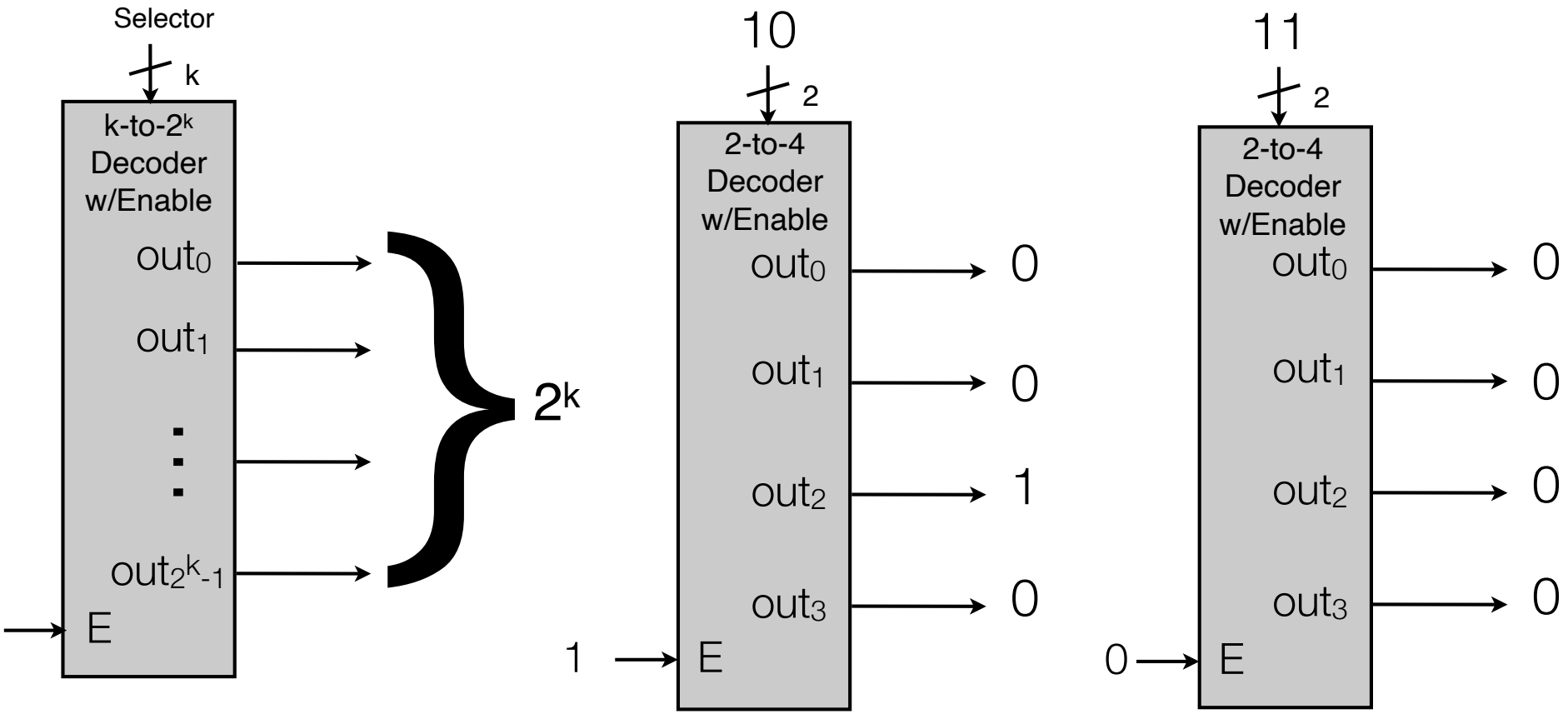
Standard Decoder Circuit (high-level view):

- No “DATA" inputs, just a k-bit “selector” input
- 2^k 1-bit outputs
- Selector input chooses which output = 1, all other outputs = 0



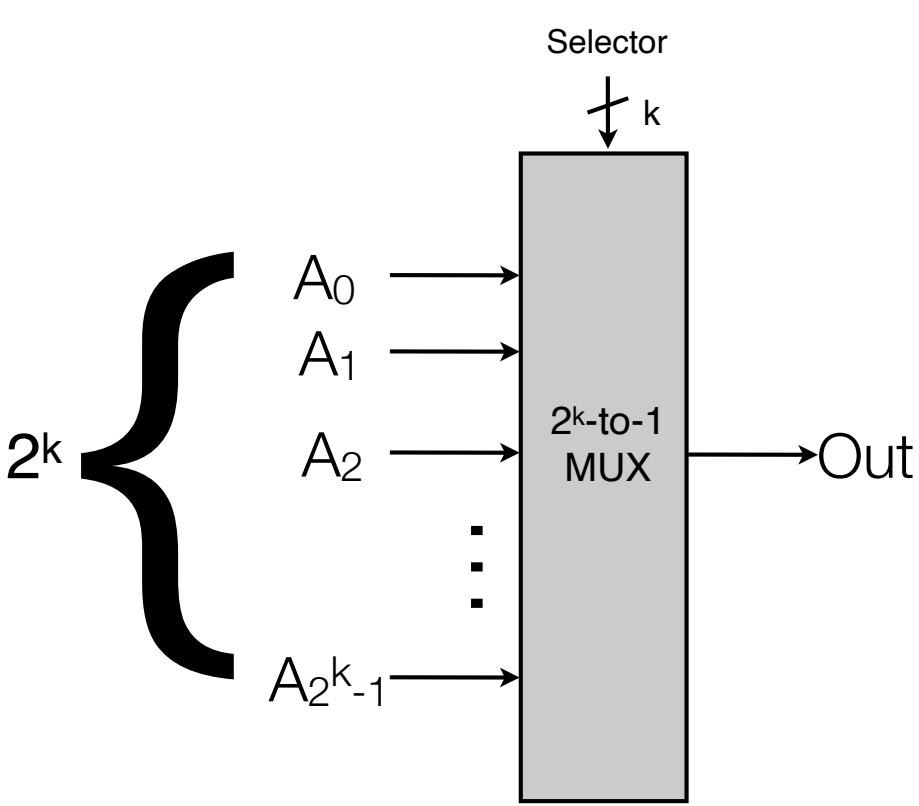
Decoder with Enable

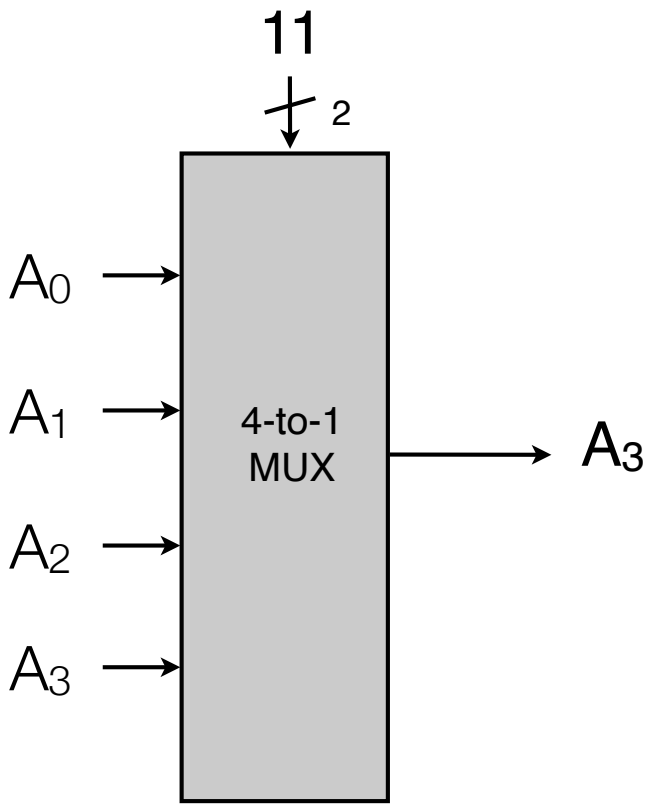
- One additional input, E
  - E=1: behaves like standard decoder
  - E=0: all outputs are 0



MUX Circuit (High-level)

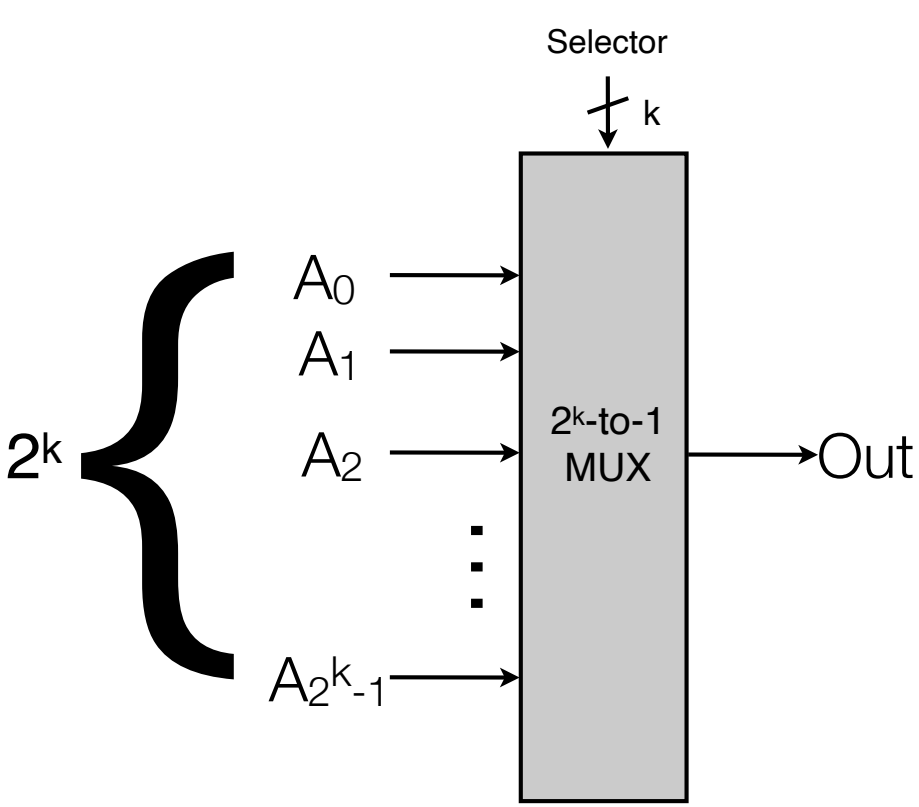
- 2^k Data values enter as input
- k-bit selector chooses which one comes out
- Like the A_i are digital music signals (over time) and i chooses the station

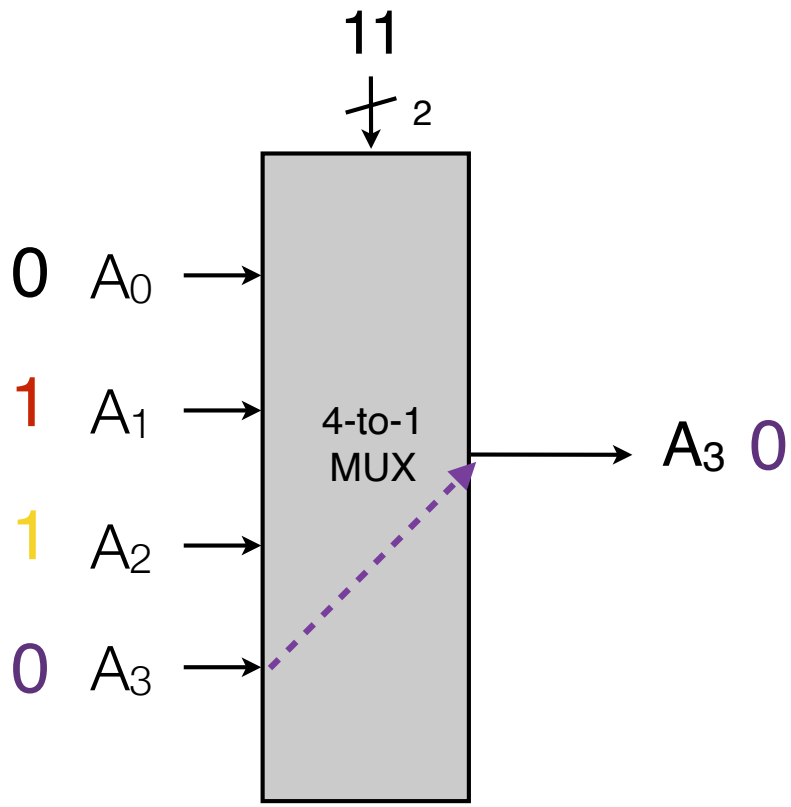




MUX Circuit (High-level)

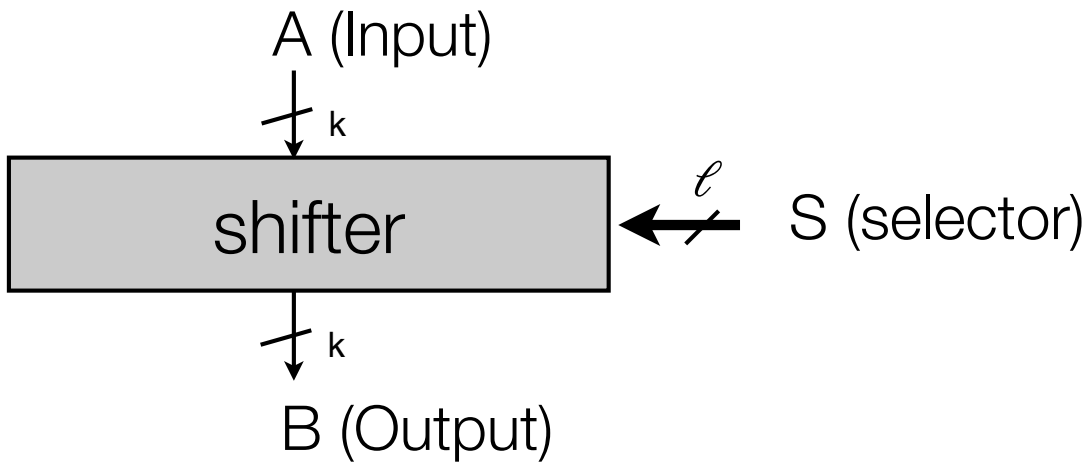
- 2^k Data values enter as input
- k-bit selector chooses which one comes out
- Like the A_i are digital music signals (over time) and i chooses the station





Shifter Circuit (High-level)

- k-bit data value entered as input
- ℓ-bit selector gives instructions how to shift



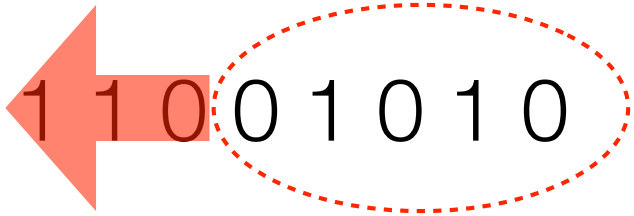
Barrel Shifter: 8 bits w/ rollout Example

e.g., shift-left-3 11001010

1 1 0 0 1 0 1 0

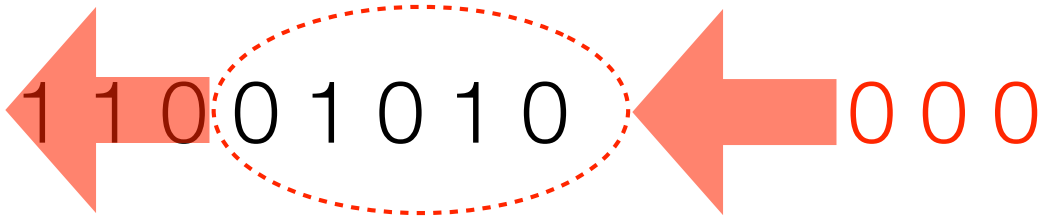
Barrel Shifter: 8 bits w/ rollout Example

e.g., shift-left-3 11001010



Barrel Shifter: 8 bits w/ rollout Example

e.g., shift-left-3 11001010



result: 01010000

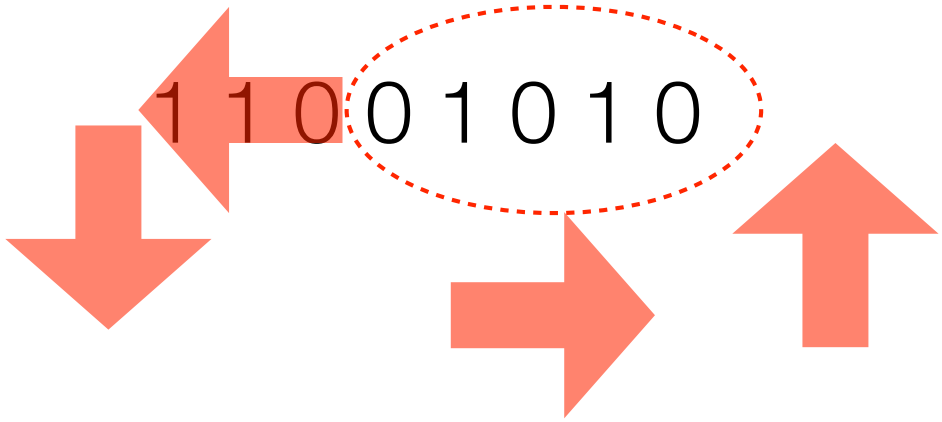
Barrel Shifter: 8 bits w/ wraparound Example

e.g., shift-left-3 11001010

1 1 0 0 1 0 1 0

Barrel Shifter: 8 bits w/ wraparound Example

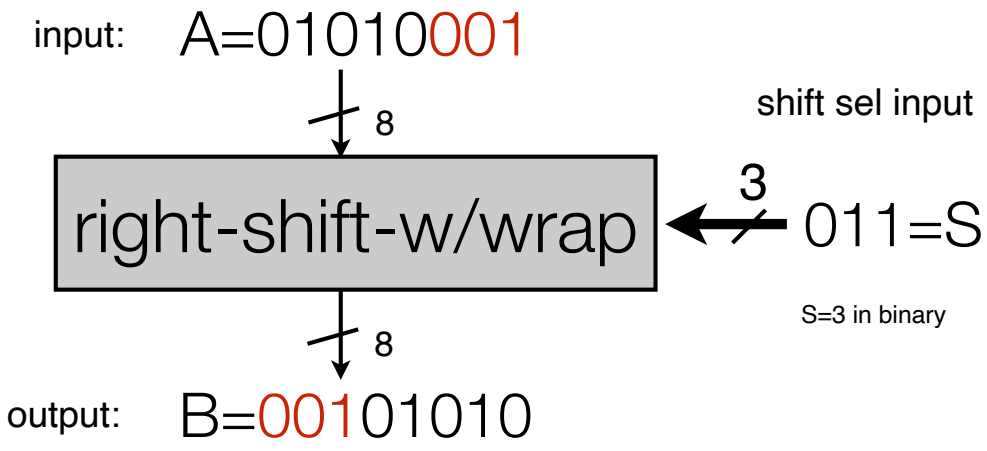
e.g., shift-left-3 11001010



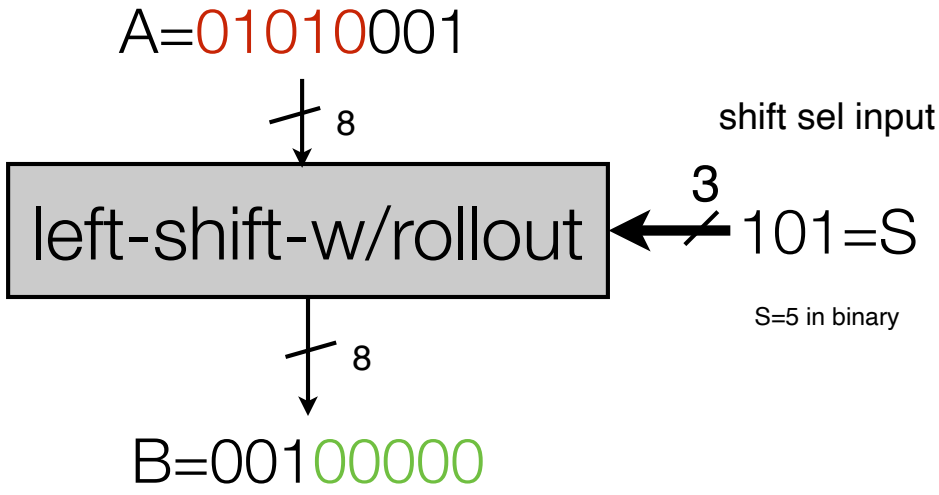
result: 01010110

(Barrel) Shifter Circuit (High-level)

- k-bit data value entered as input
- ℓ = $\log_2$k-bit selector
  chooses how far bits should be shifted
- Example 1: k=8, right shifter with "wraparound", asked to shift by 3 bits

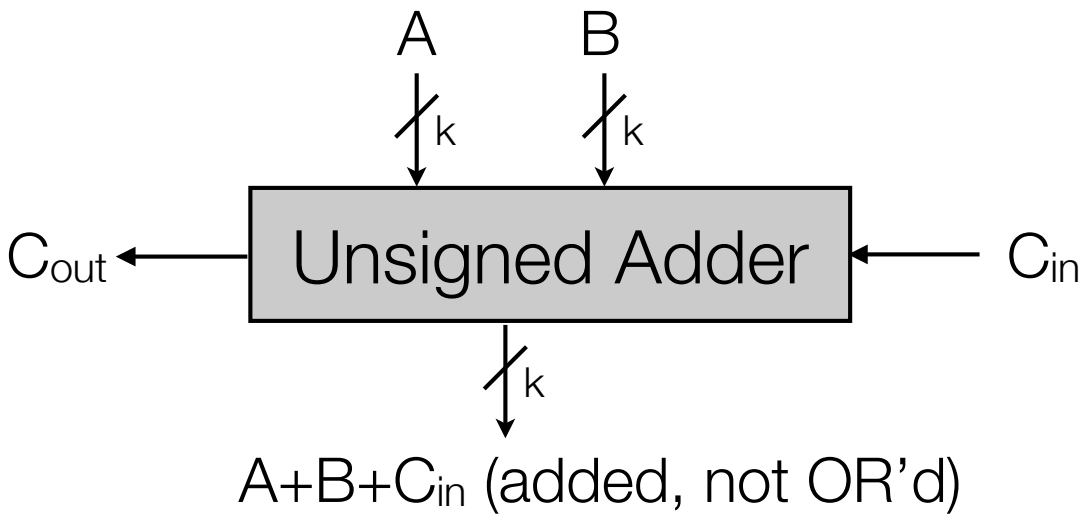


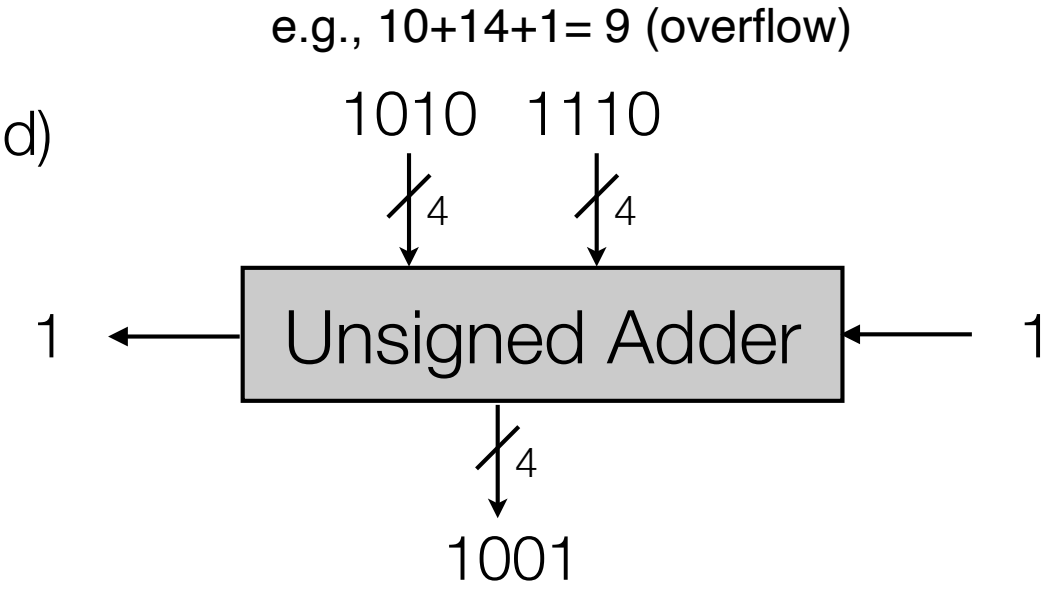
- Example 2: k=8, left shifter with "rollout", asked to shift by 5 bits (fill missing bits with 0's



(Unsigned) Adder Circuit (High-level)

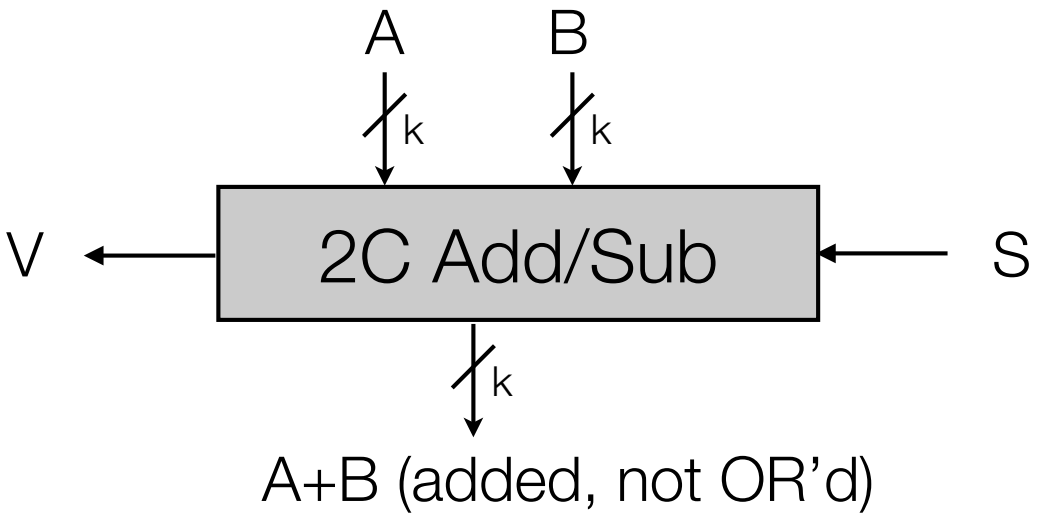
- Two k-bit data values, A and B, and 1 bit C_{in} (carry-in) entered as input
- output is k-bit A+B + C_{in} (unsigned binary addition), and 1-bit C_{out} (carry-out), i.e., the overflow



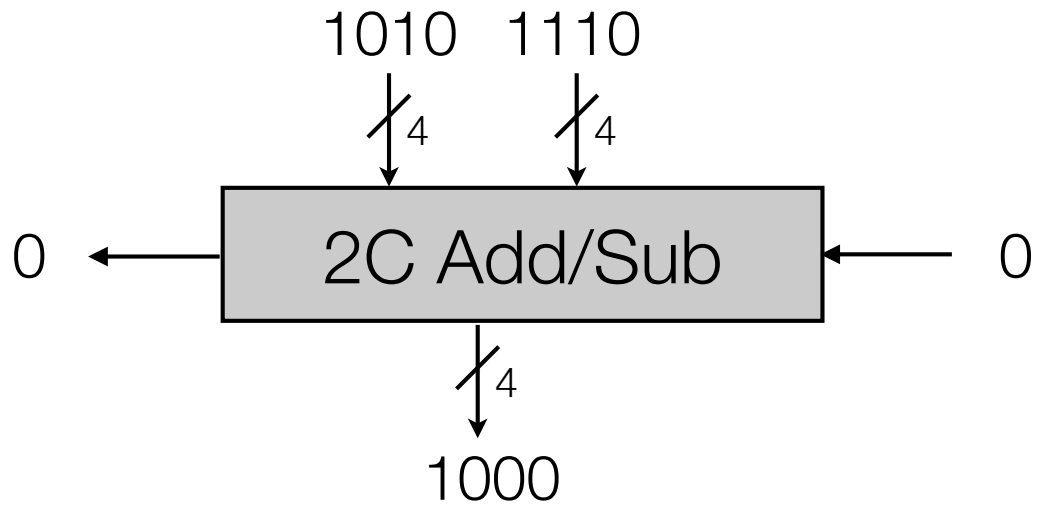


(Signed 2's-C) Adder/Subtractor Circuit (High-level)

- Two k-bit data values, A and B, and 1 bit S=0 for add, S=1 for subtract
- Output is k-bit A+B when S=0, A-B when S=1 (2's comp binary add/sub), and 1-bit overflow, V



e.g., -6 + (-2) = -8 (no overflow)



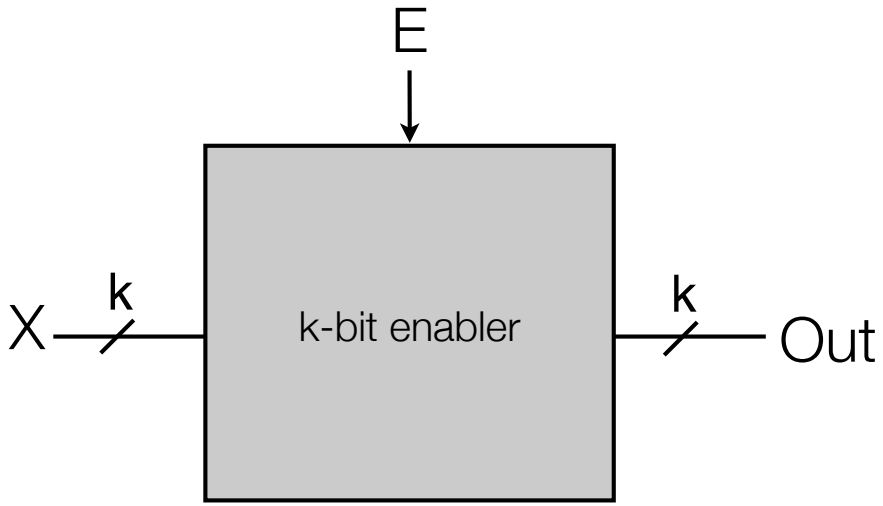
“Standard” Circuit Details

(how to build from basic boolean ops)

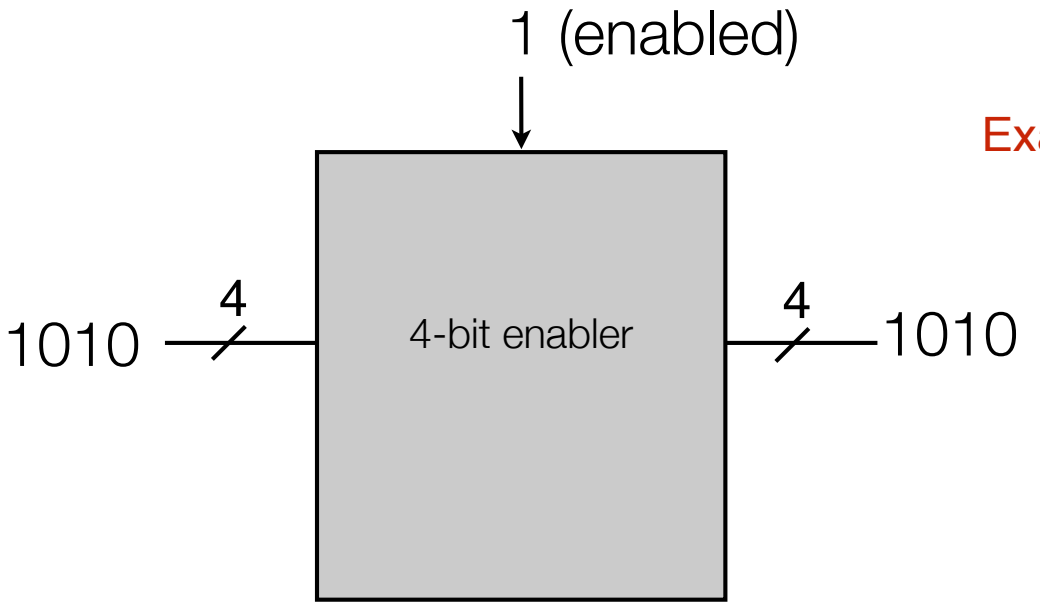
Enabler

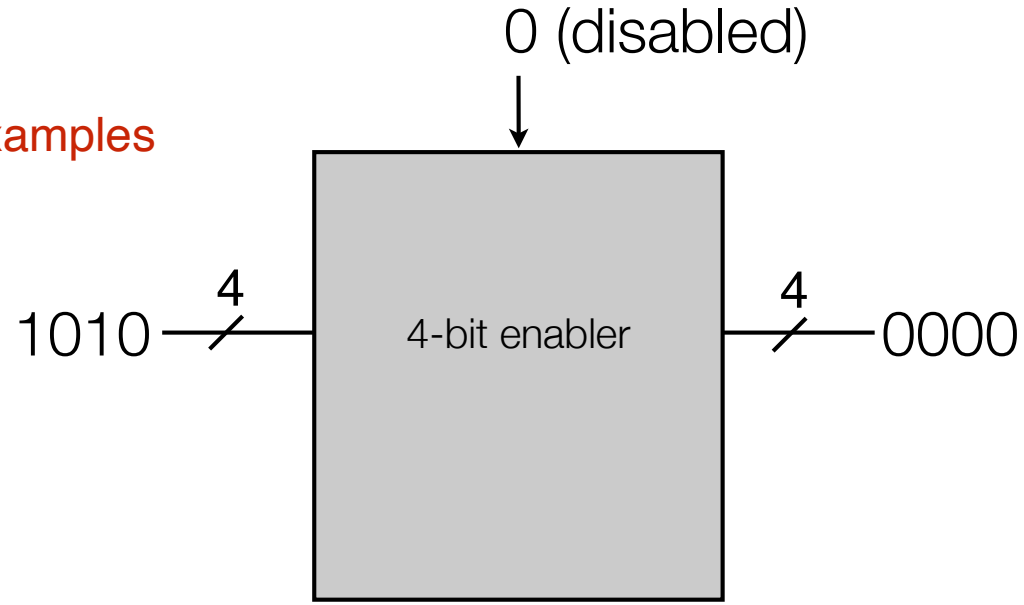
Enabler Circuit (High-level view):

- Enabler circuit has 2 inputs
  - data (can be k bits in parallel)
  - enable/disable (i.e., on/off) input
- Output:
  - If E=1, output = data input
  - If E=0, output = k 0's



Examples

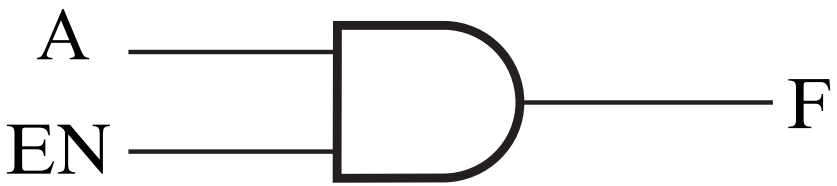




Enabler circuits: 1-bit Data input “A” (k=1)

3-15

Abbreviated truth table (input listed in the output)



(a)

EN	F
0	0
1	A

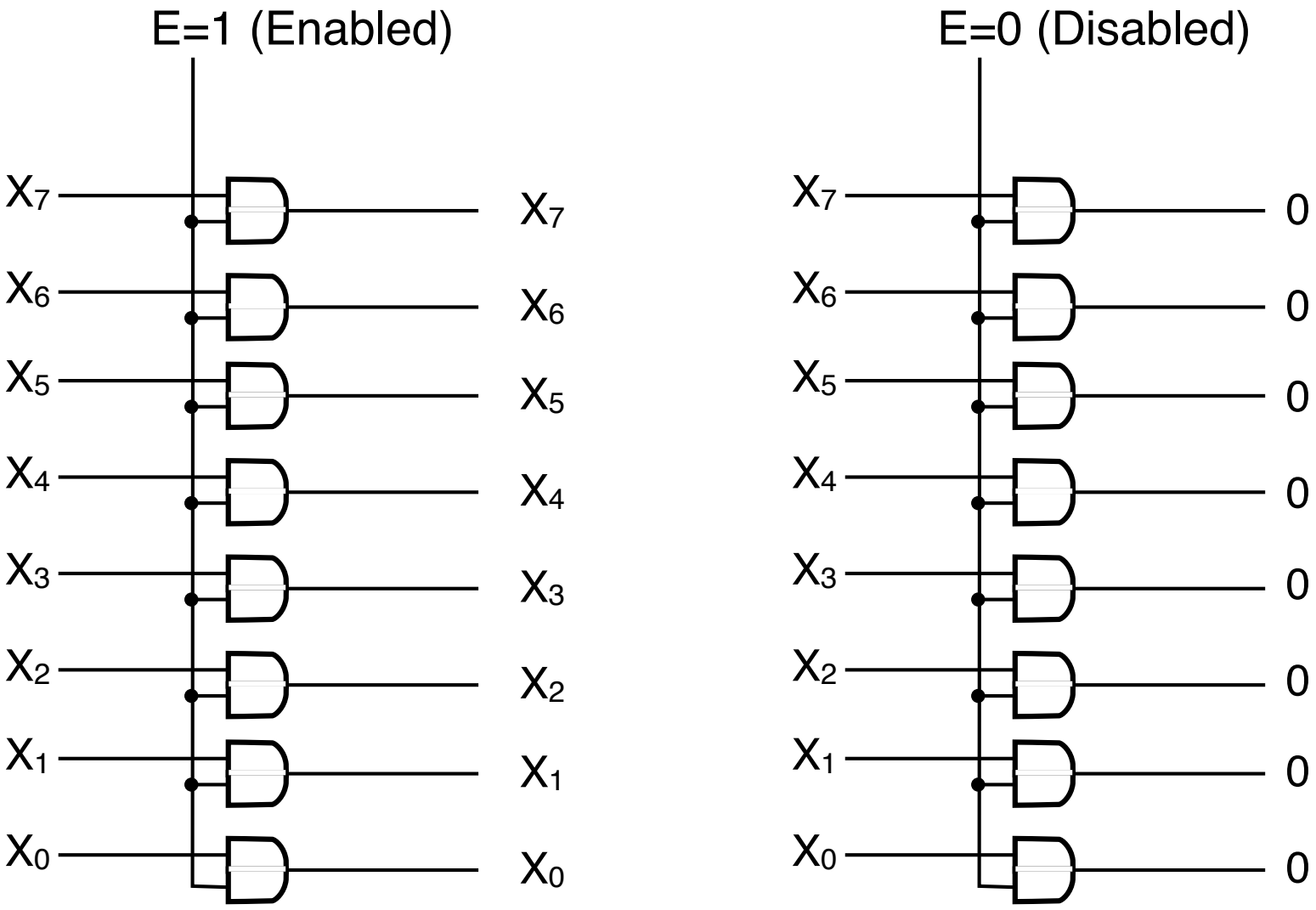
A “reverse” enabler (outputs 1 when disabled):

(b)

EN	F
0	1
1	A

For both enabler circuits above, output is “enabled” (F=X) only when input ‘ENABLE’ signal is asserted (EN=1). Note the different output when DISABLED

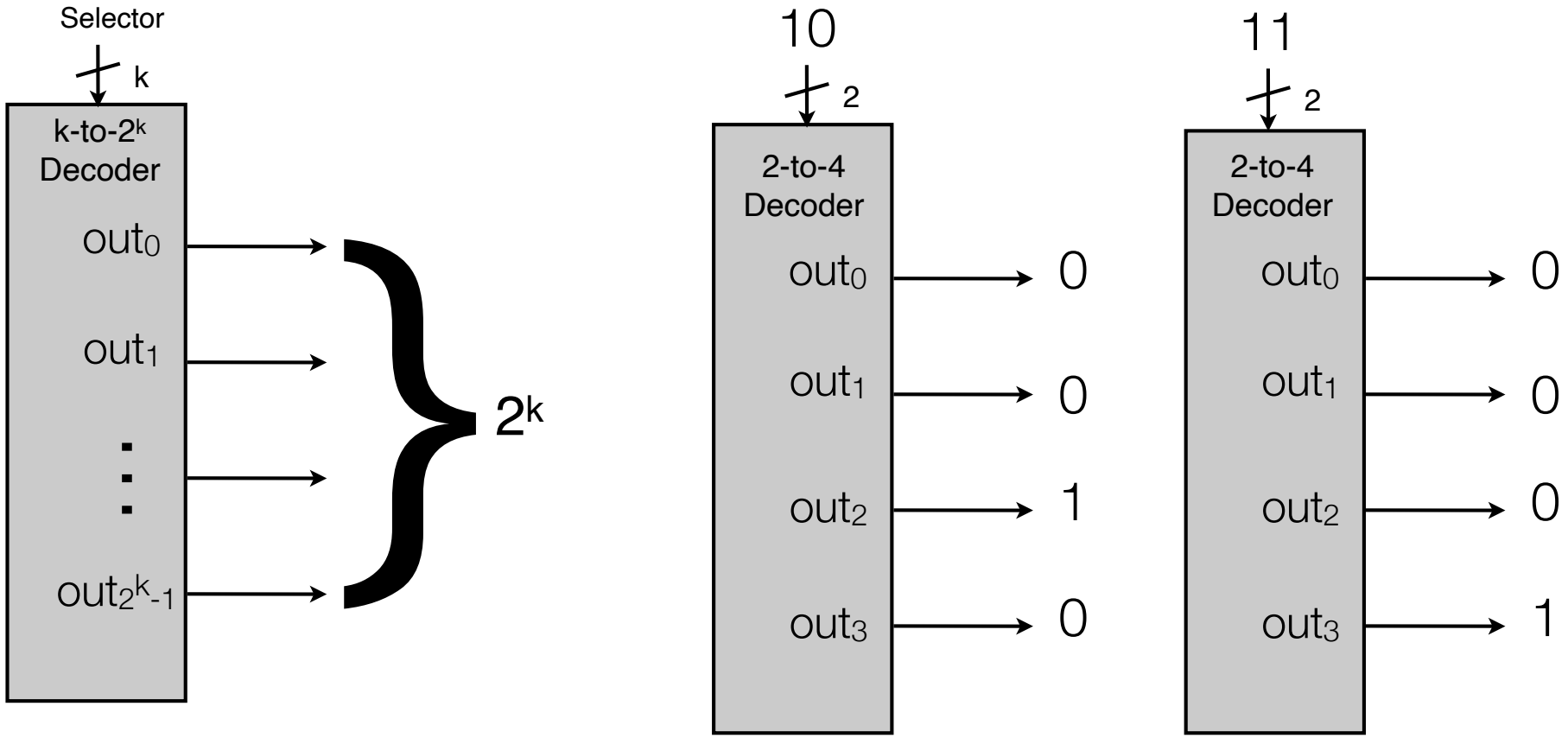
k-bit enabler (e.g., k=8)



Decoder

Standard Decoder Circuit (high-level view):

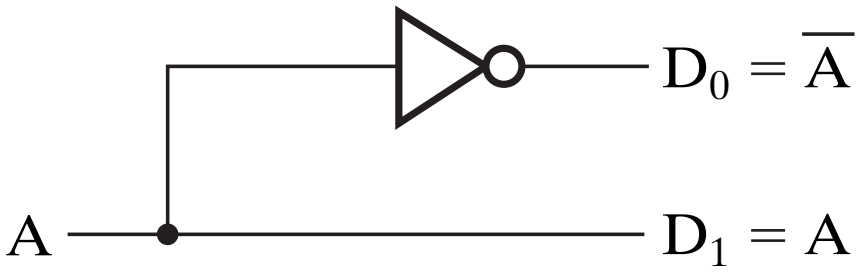
- No “DATA" inputs, just a k-bit “selector” input
- 2^k 1-bit outputs
- Selector input chooses which output = 1, all other outputs = 0



Decoder (1:2) Internal Design

A	D_0	D_1
0	1	0
1	0	1

(a)



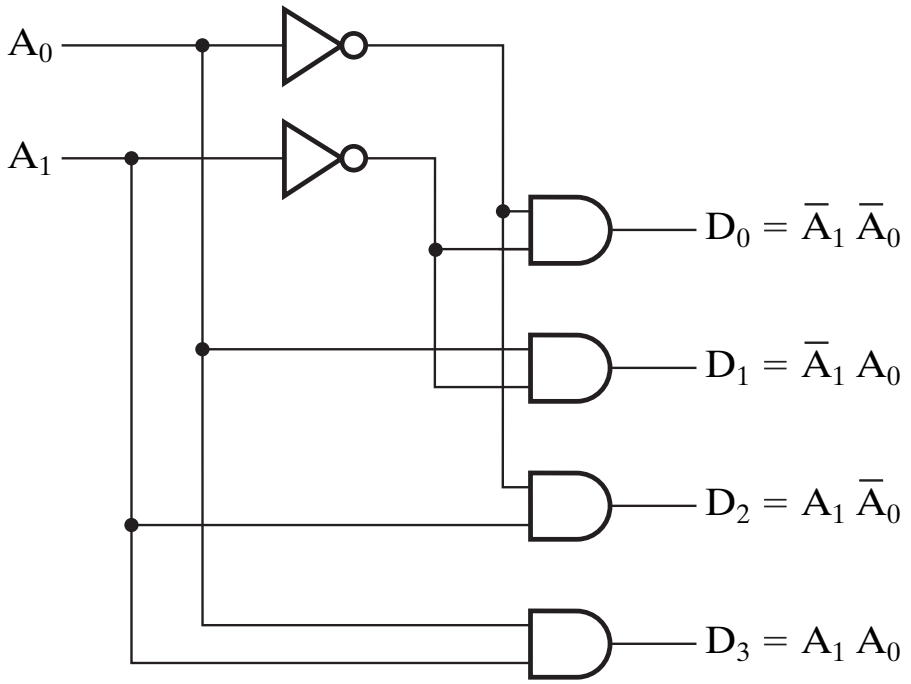
(b)

Decoder (2:4)

"Standard" Decoder: i^{th} output = 1, all others = 0, where i is the binary representation of the input

A_1	A_0	D_0	D_1	D_2	D_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

(a)

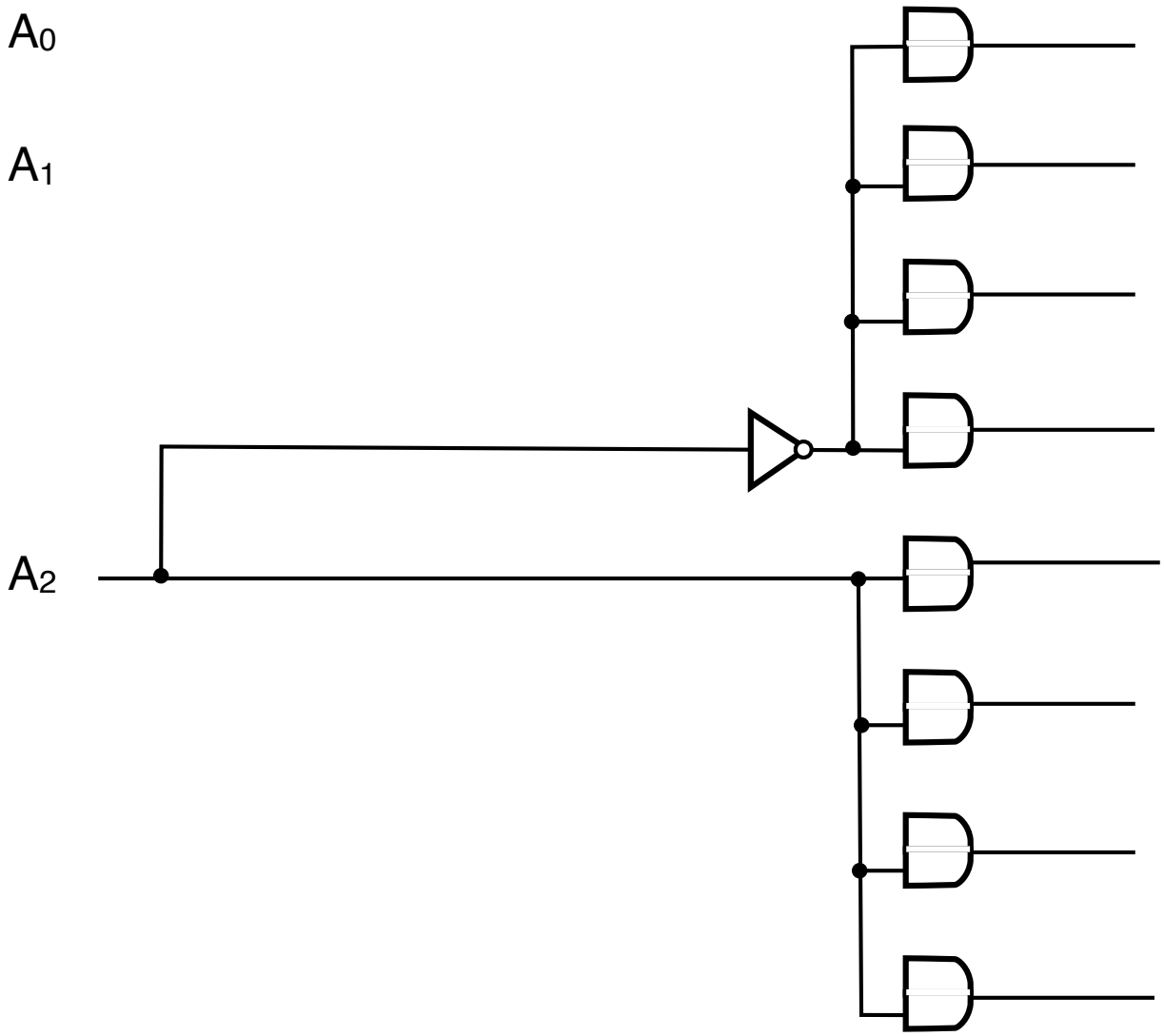


(b)

i^{th} output is just the i^{th} midterm!

Building a Decoder from scratch (building Minterms)

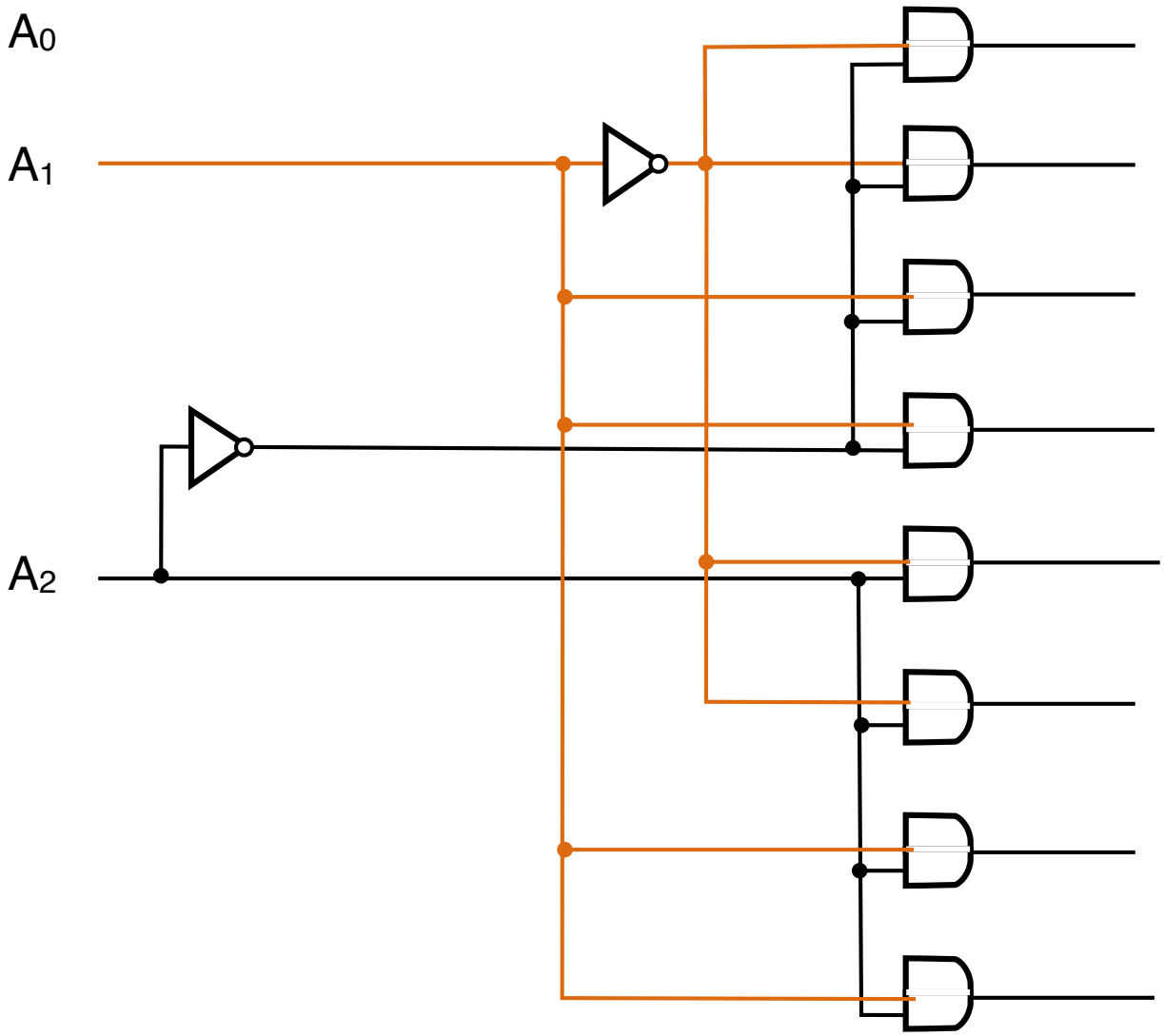
- A_2 is 0 for top 4 gates, 1 for bottom 4



A_2	A_1	A_0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Building a Decoder from scratch (building Minterms)

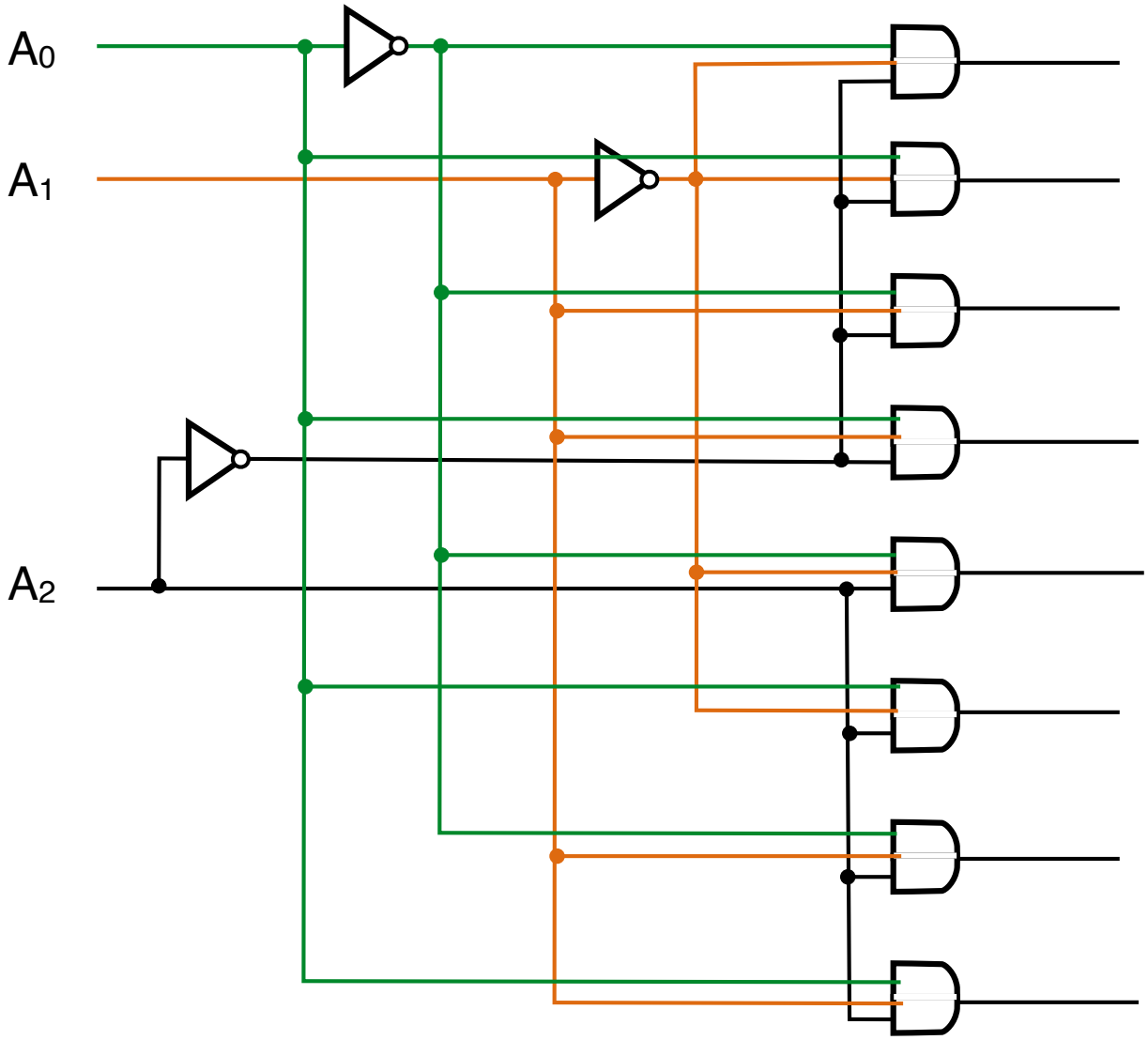
- A_1 is 0 for gates 0,1,4,5, is 1 for gates 2,3,6,7



A_2	A_1	A_0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

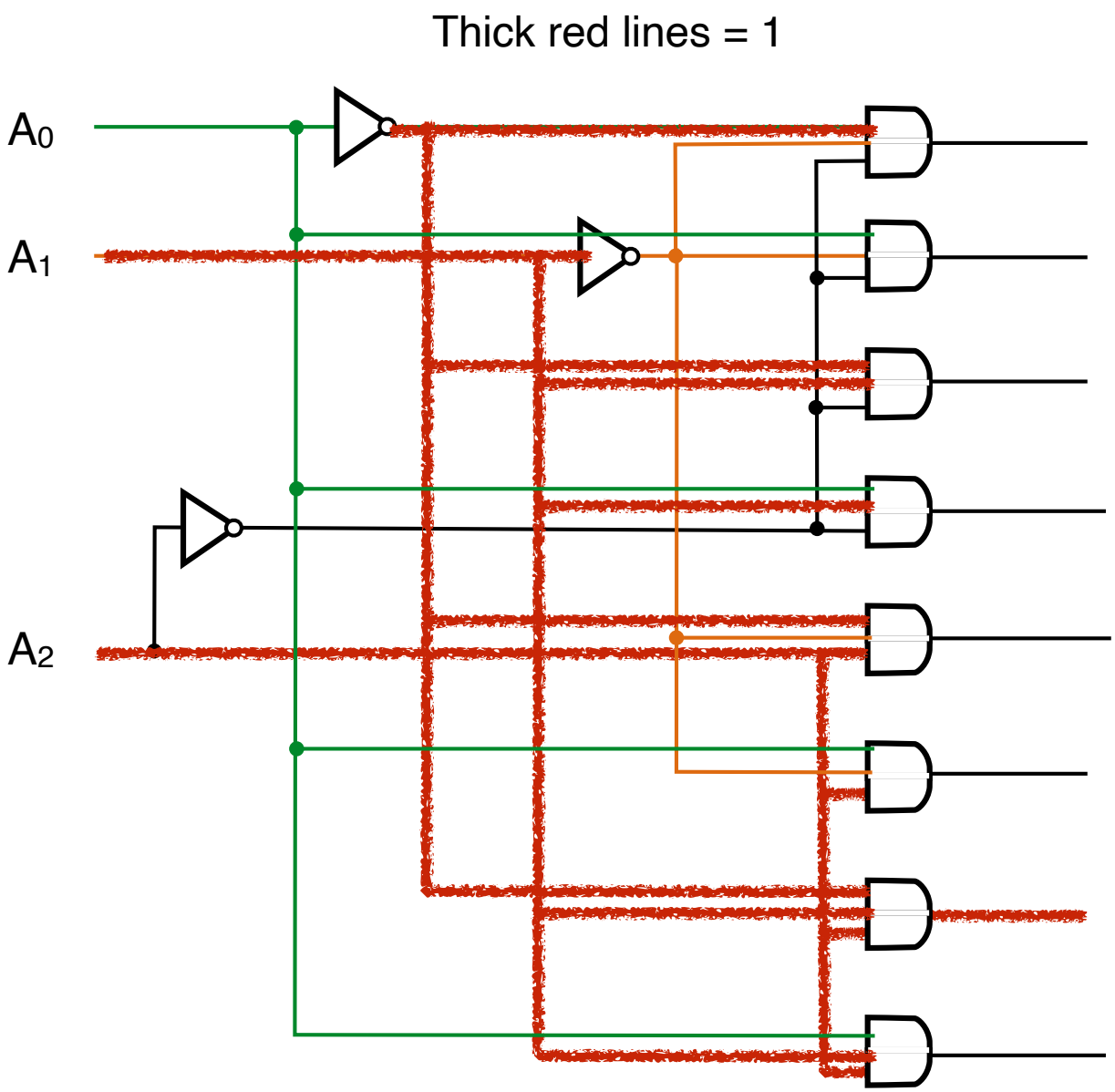
Building a Decoder from scratch (building Minterms)

- A_0 is 0 even gates (0,2,4,6), 1 for odd



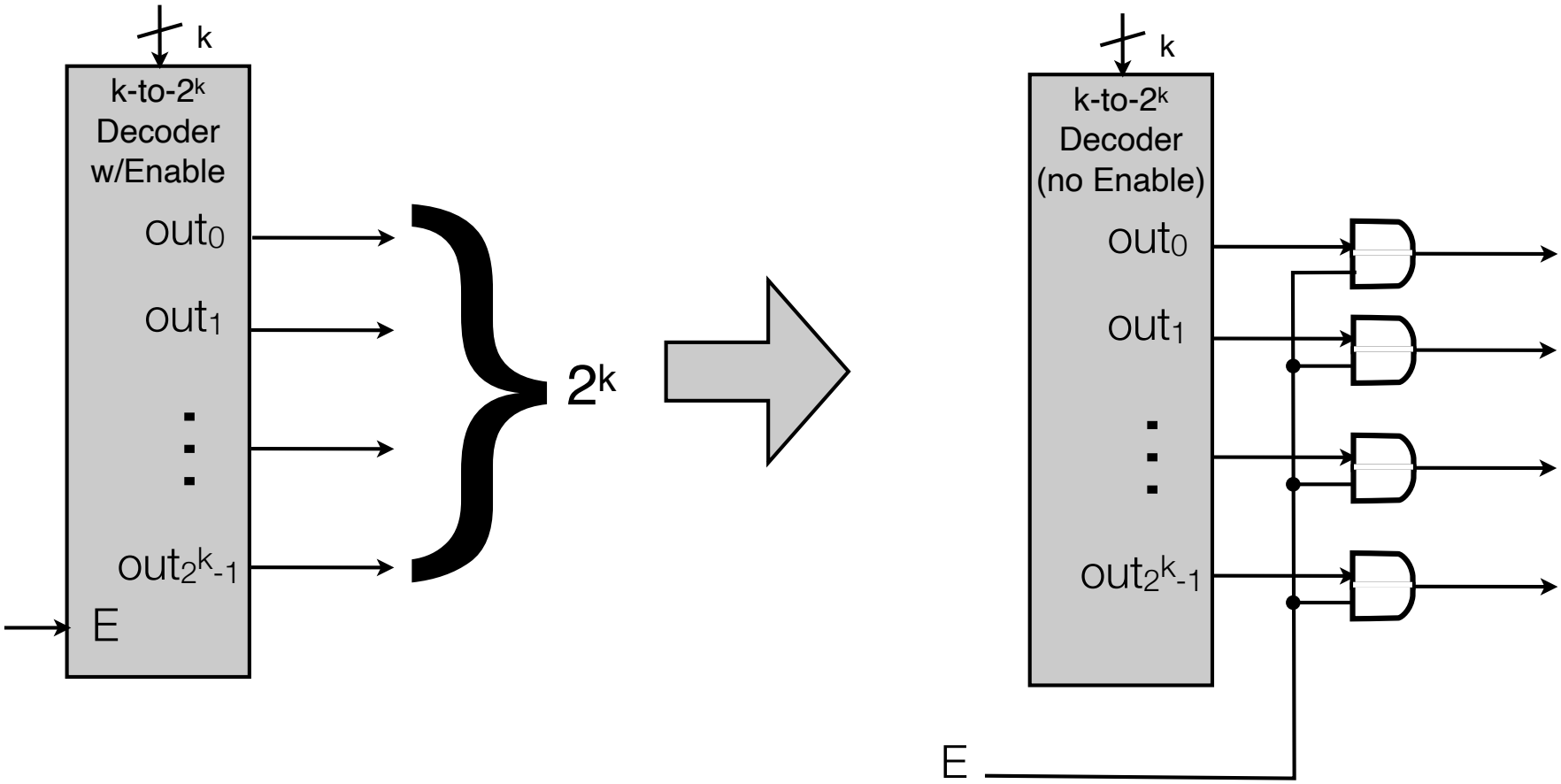
A_2	A_1	A_0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Example: A = $A_2A_1A_0$ = 1 1 0



A_2	A_1	A_0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

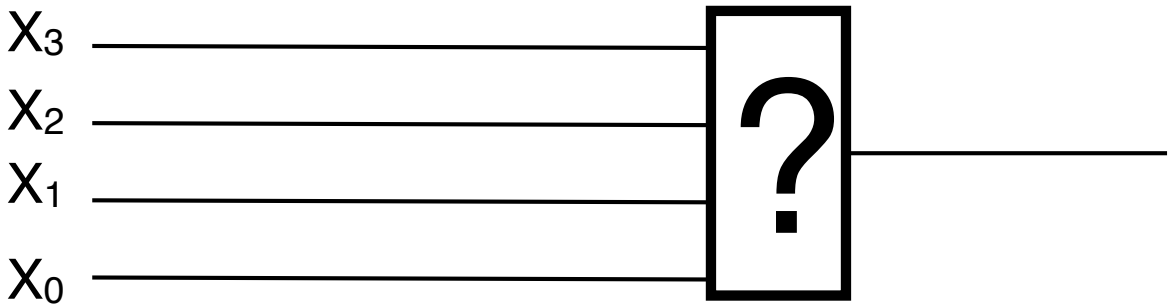
Decoder with Enable from Decoder



Merging a signal with known 0 signals

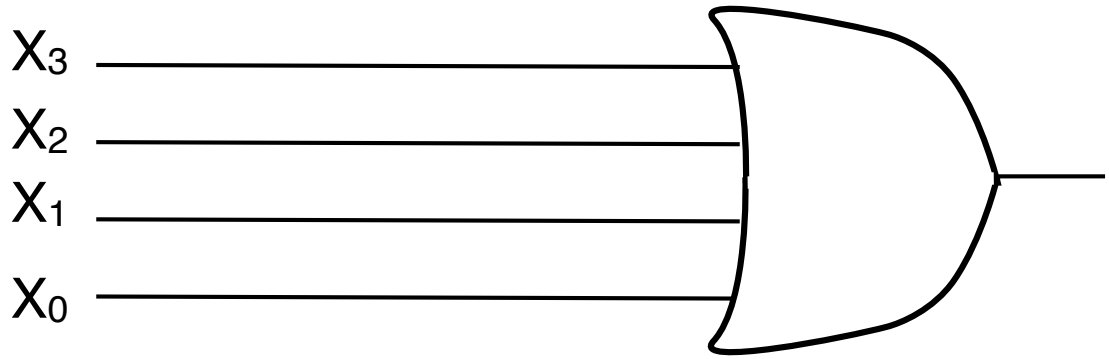
Merge-with-known-0's

- Suppose have m input signals
- We know all of them = 0 except maybe one X_j. (but we don't know which Xj is possibly non-0)
- Circuit to retrieve the value of the one unknown



Merge-with-known-0's

- Suppose have m input signals
- All of them = 0 except maybe one X_j.
- Circuit to retrieve the value of the one unknown

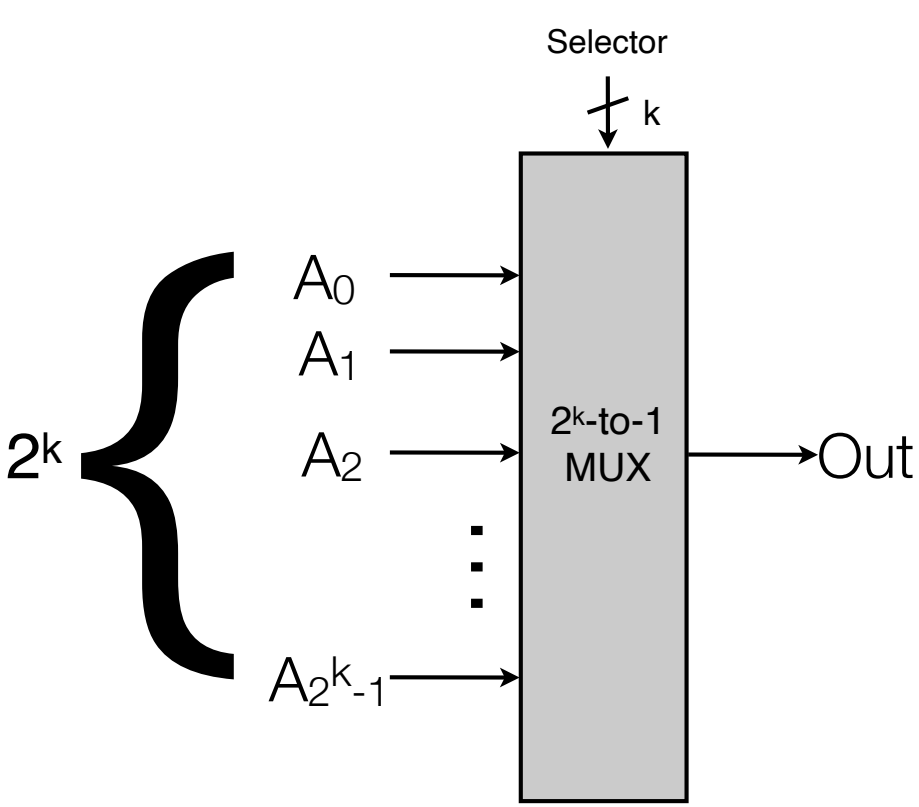


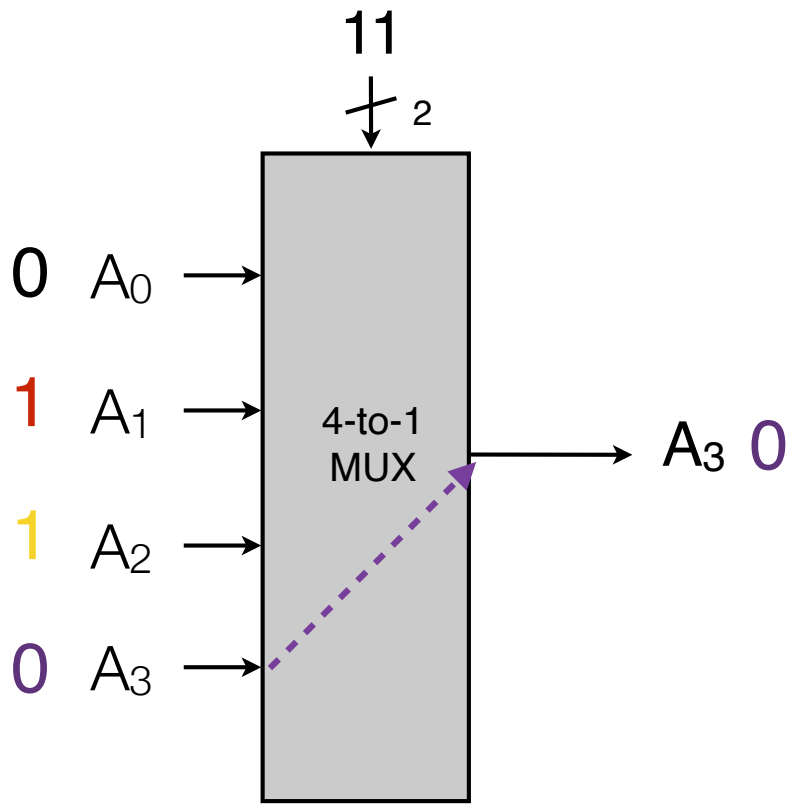
- If all X_i=0 for i≠j, then the OR over all X_i (including X_j) = X_j

Multiplexer (MUX)

MUX Circuit (High-level)

- 2^k Data values enter as input
- k-bit selector chooses which one comes out
- Like the A_i are digital music signals (over time) and i chooses the station





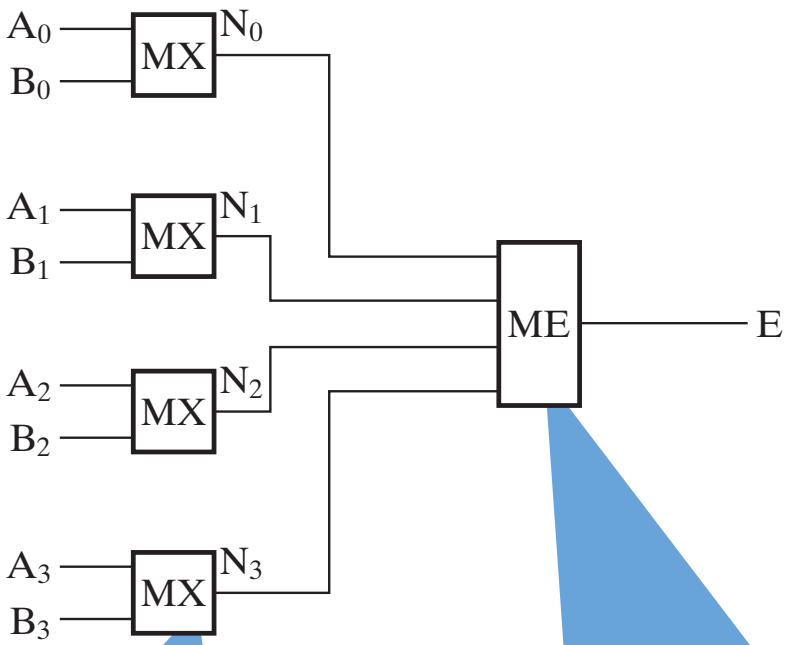
Hierarchical Design

Recall

- AND, OR, NOT gates are really built from NANDs (or NORs)
- When we show a circuit in terms of AND,OR,NOT, the real underlying design is hierarchically built from NANDS

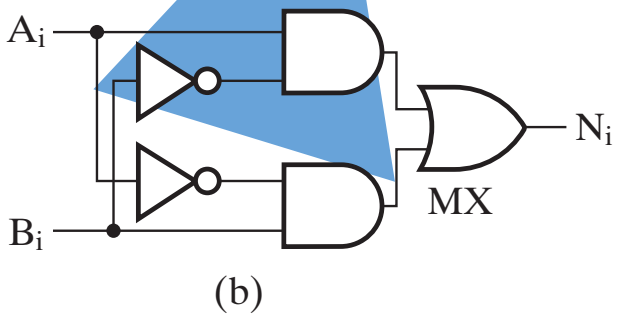
Similarly, big circuits built from “smaller parts”

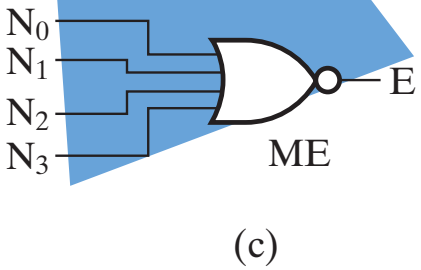
“Big”Circuit



Big circuit can be built by combining smaller circuits, which are themselves built from even smaller circuits

Smaller Circuits

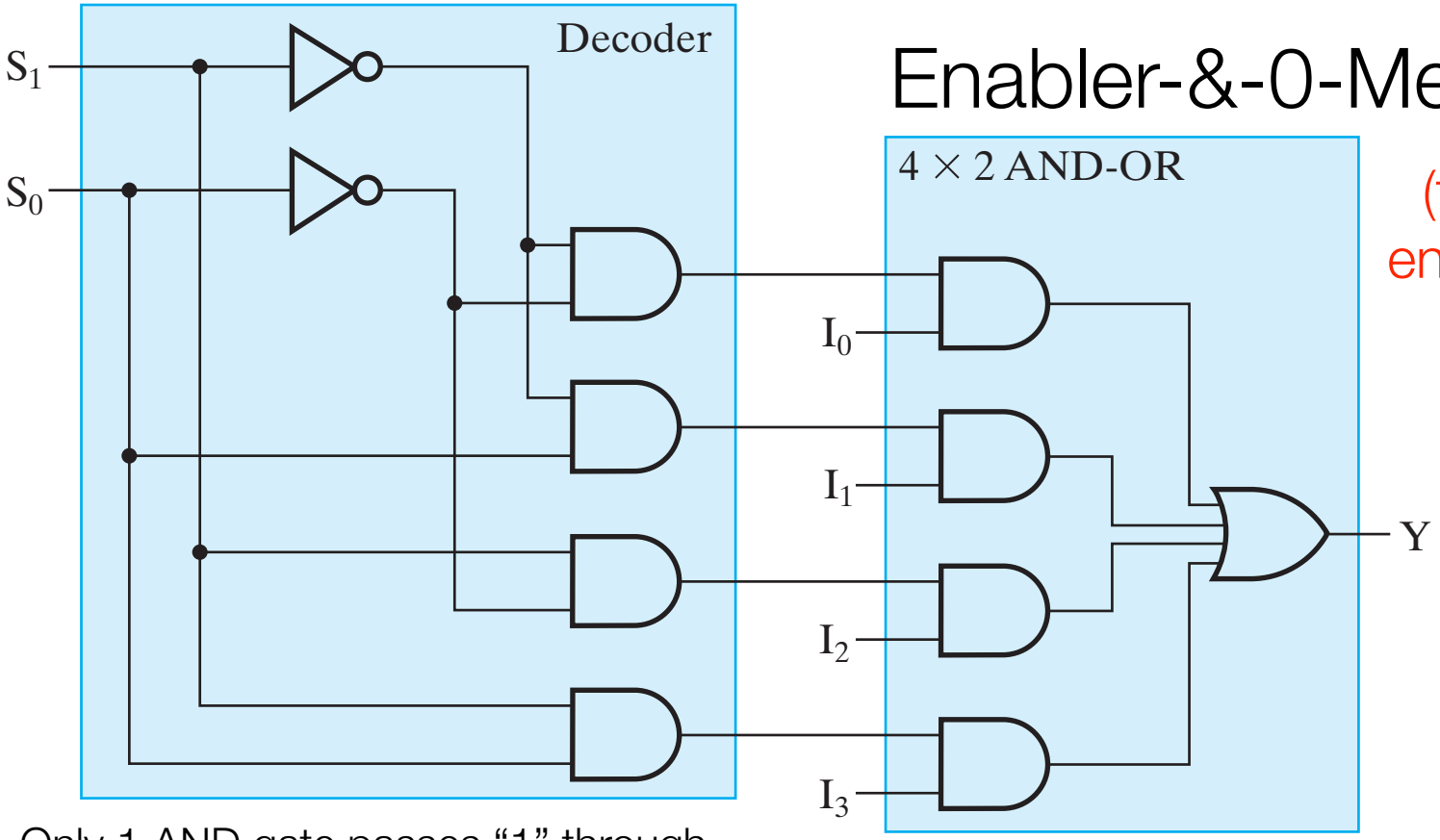




MUX internals (hierarchical design example)

Selector Logic

(selects which inputs "flows through")
and which get "zeroed out"



Only 1 AND gate passes "1" through

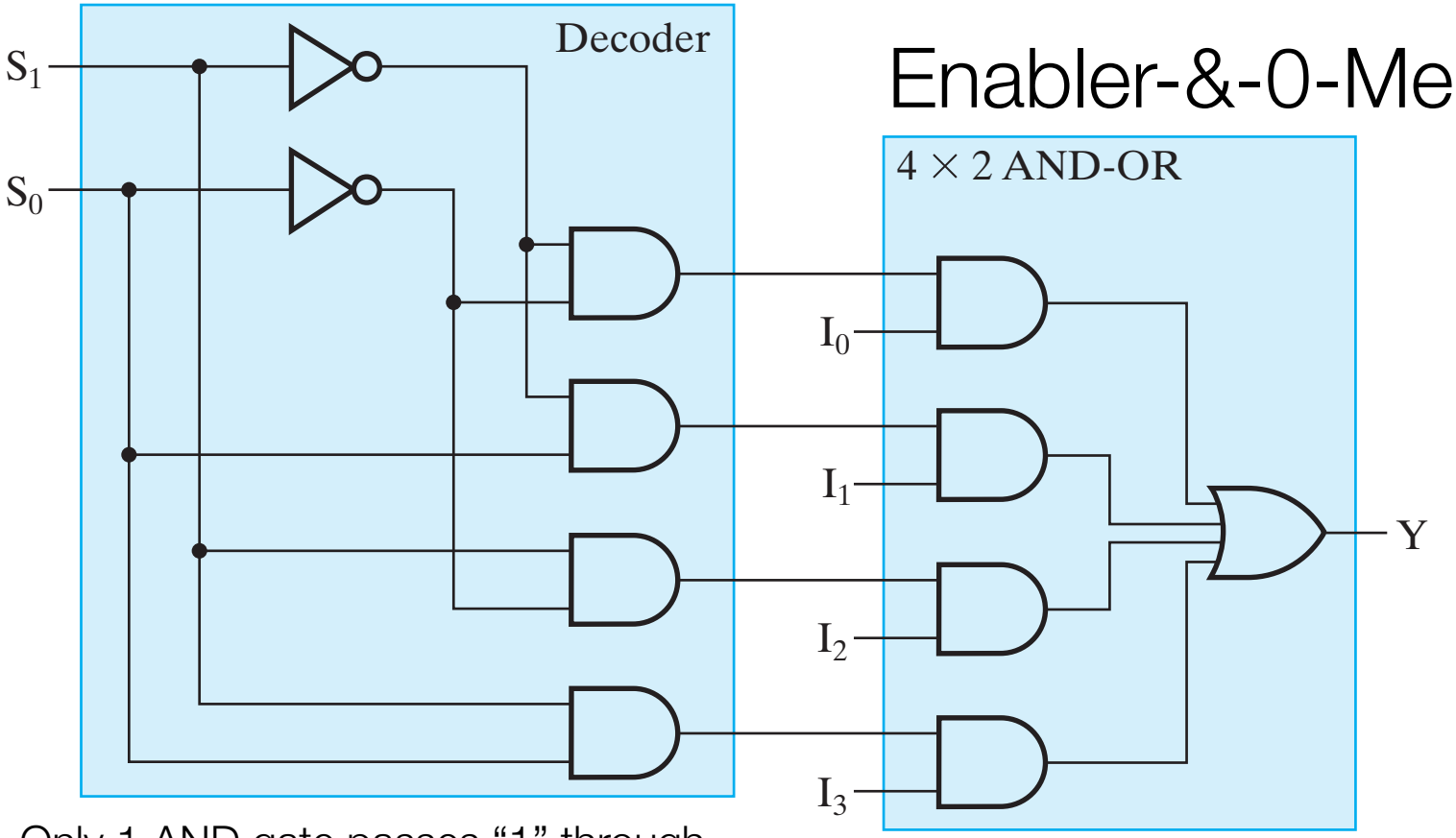
Enabler-&-0-Merge logic

(takes inputs, enables one, 0's out the rest)

AND-OR gate combo "zeros out" unselected I_i

MUX Example: S_1=1, S_0=0

Selector Logic

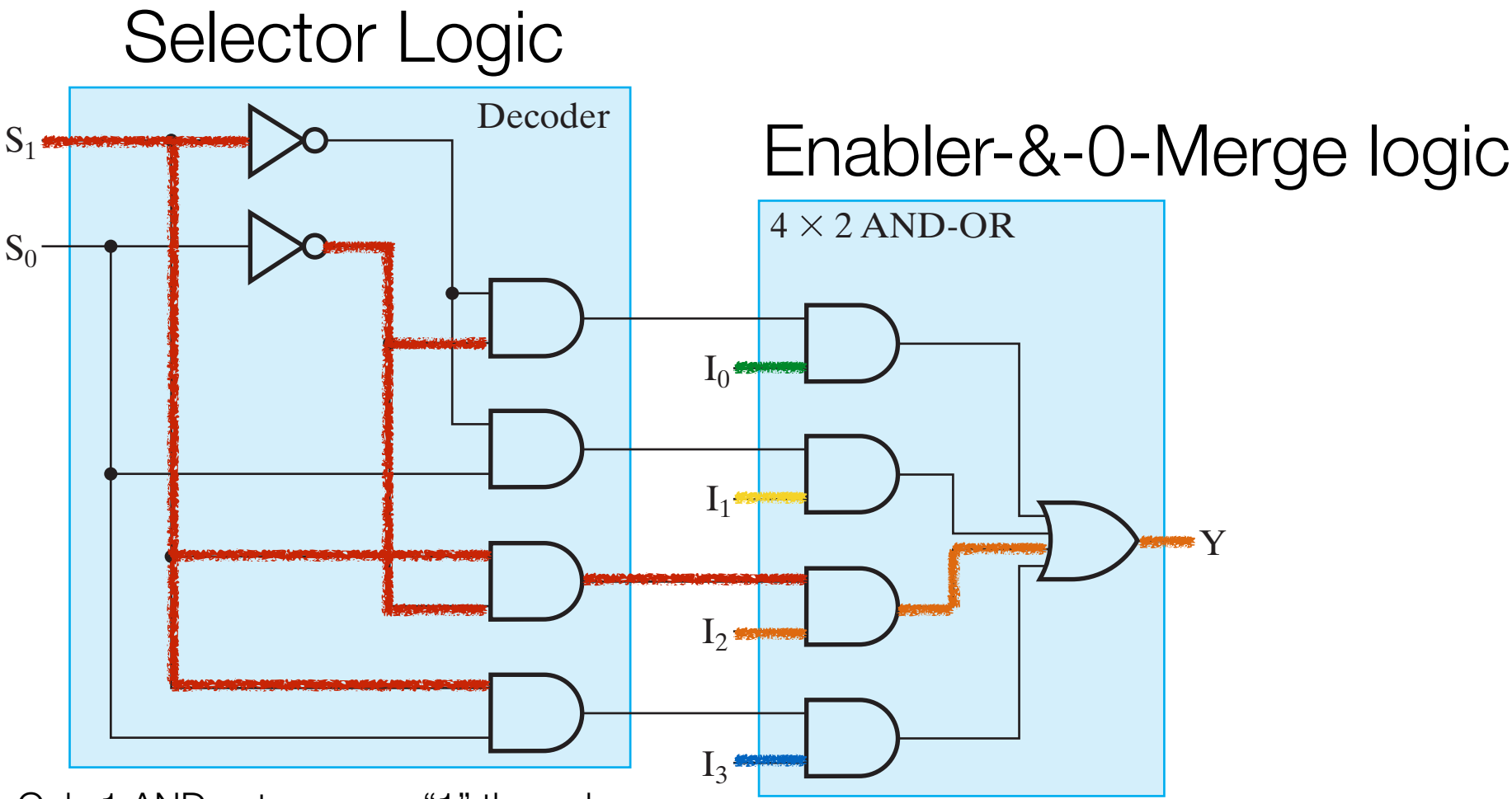


Only 1 AND gate passes "1" through

Enabler-&-0-Merge logic

AND-OR gate combo "zeros out" unselected I_i

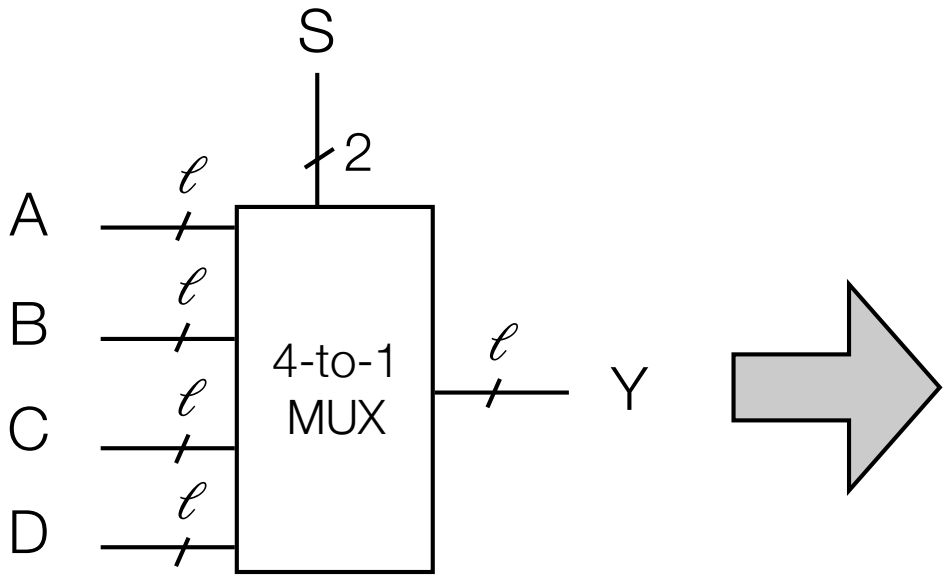
MUX Example: S_1=1, S_0=0 (i.e., S = 2)

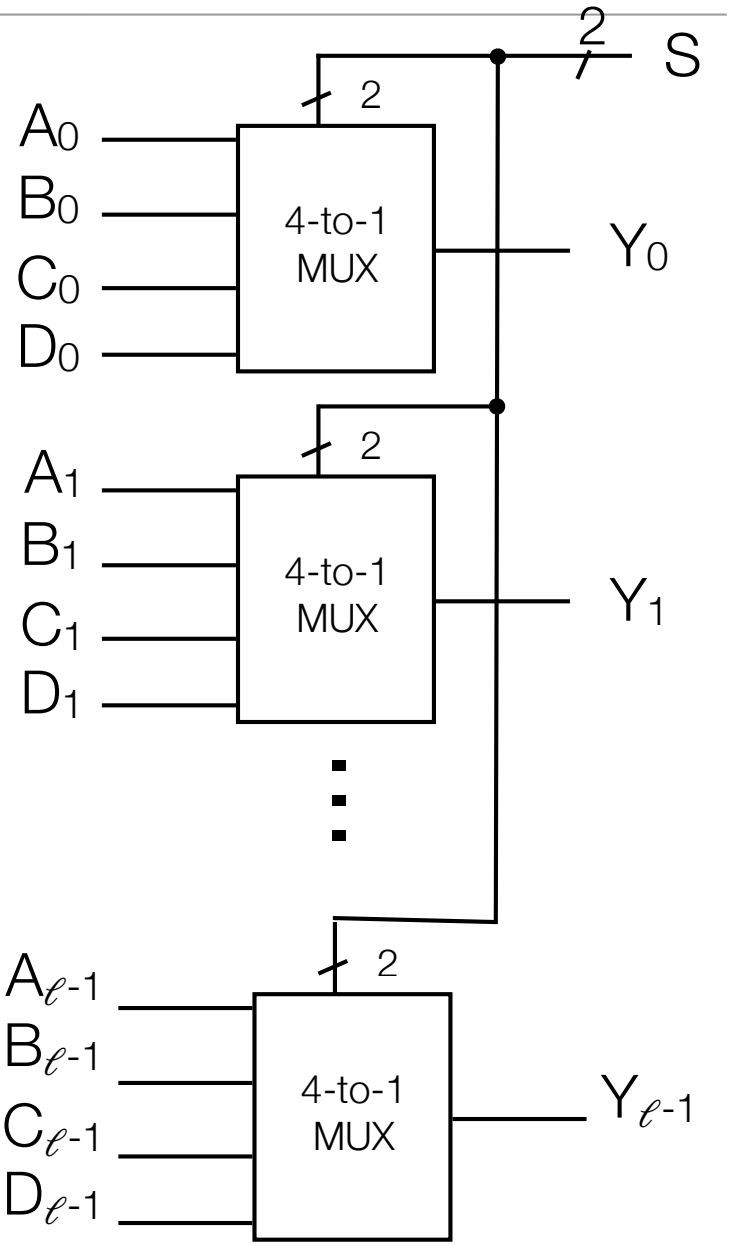


Only 1 AND gate passes "1" through

AND-OR gate combo "zeros out" unselected I_i

Multi-bit Data Input MUX shorthand notation





- ℓ-bit 4-to-1 MUX: ℓ-bit quantities fed into a multiplexer, which selects one of those quantities

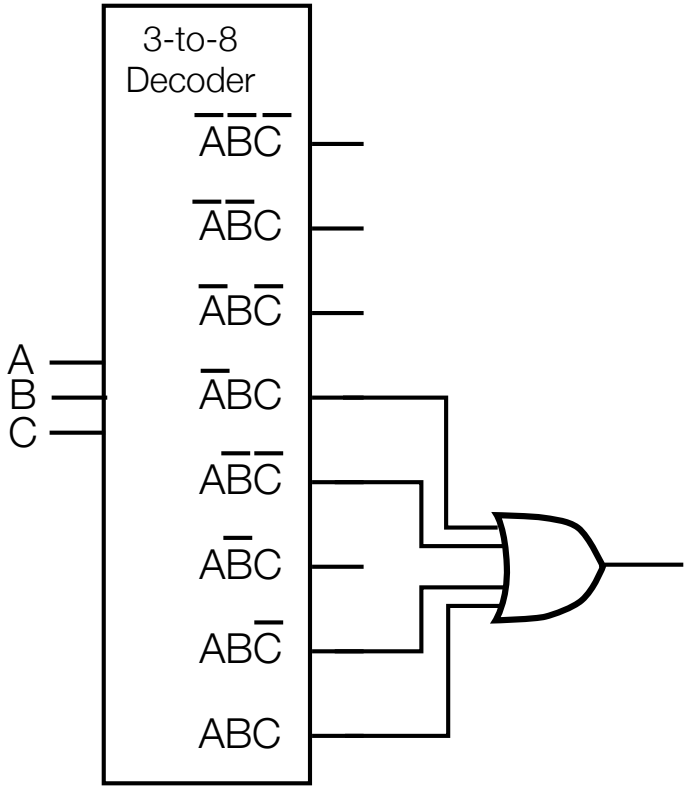
- Implemented by feeding each of k bits into k separate 1-bit 4-to-1 MUX, same selector bits fed into each MUX

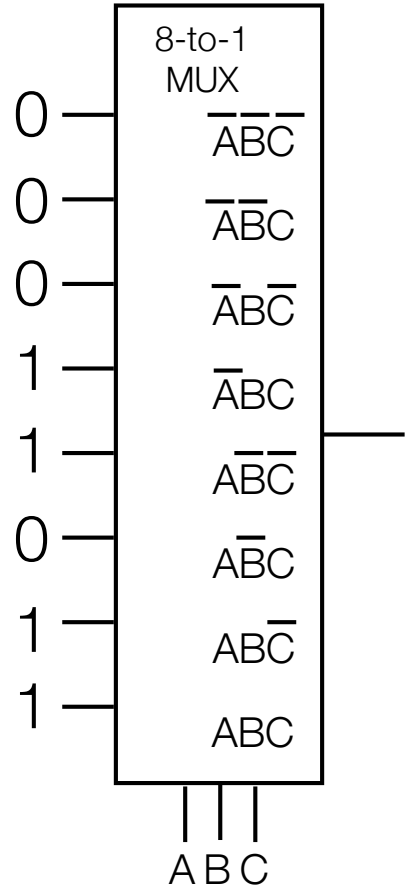
Building Functions from Decoders and MUXes

Representing Functions with Decoders and MUXes

- e.g., $F = A\bar{C} + BC$

A	B	C	minterm	F
0	0	0	$\bar{A}\bar{B}\bar{C}$	0
0	0	1	$\bar{A}\bar{B}C$	0
0	1	0	$\bar{A}B\bar{C}$	0
0	1	1	$\bar{A}BC$	1
1	0	0	$A\bar{B}\bar{C}$	1
1	0	1	$A\bar{B}C$	0
1	1	0	$AB\bar{C}$	1
1	1	1	ABC	1





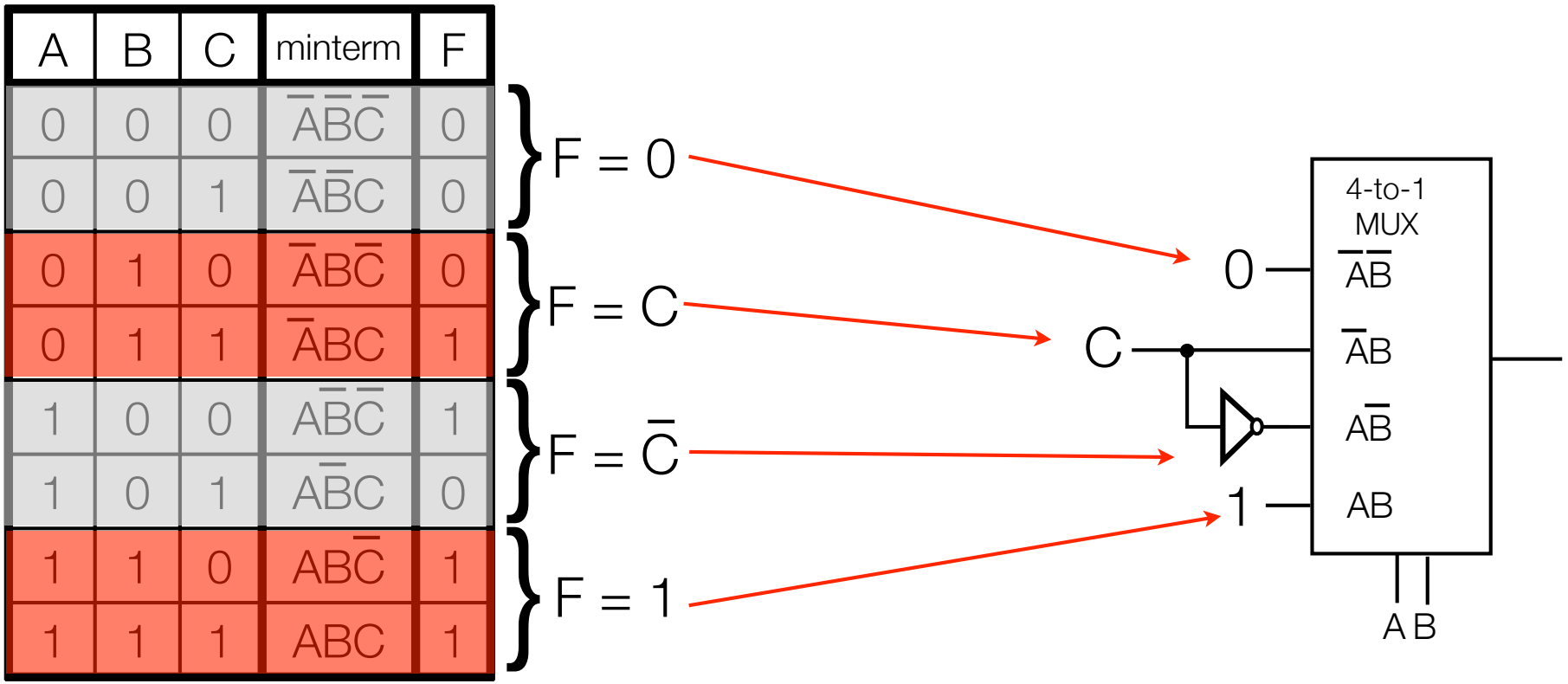
- Decoder: OR minterms for which F should evaluate to 1

- MUX: Feed in the value of F for each minterm

A Slick MUX trick

- Can use a smaller MUX with a little trick e.g., F = AC + B$\bar{C}$
- Note for rows paired below, A&B have same values, C iterates between 0&1
- For the pair of rows, F either equals 0, 1, C or $\bar{C}$

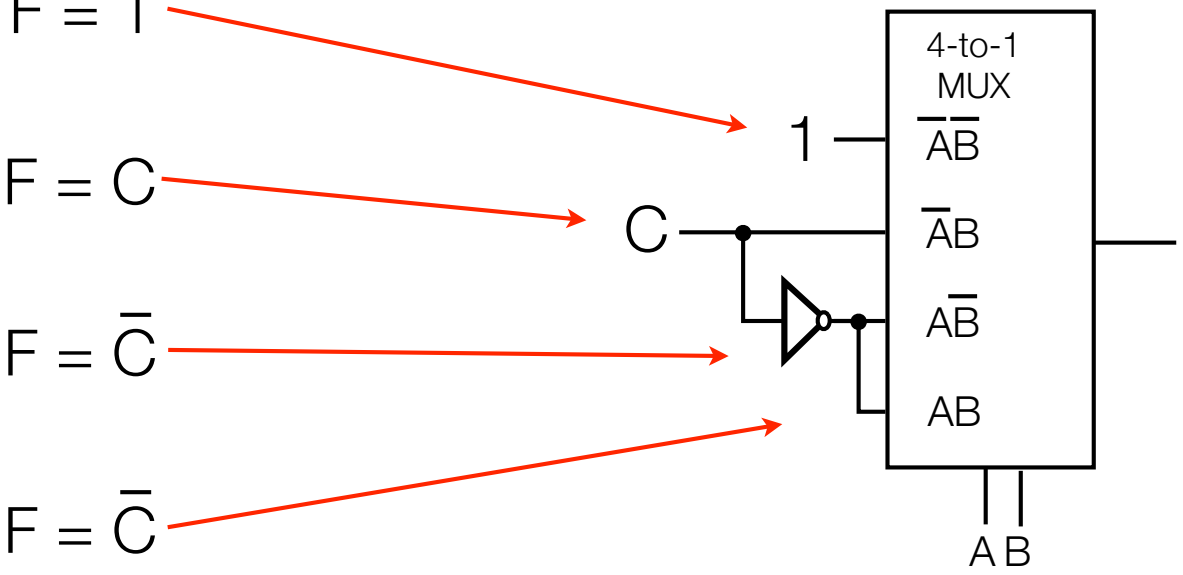
A	B	C	minterm	F
0	0	0	$\bar{A}\bar{B}\bar{C}$	0
0	0	1	$\bar{A}\bar{B}C$	0
0	1	0	$\bar{A}B\bar{C}$	0
0	1	1	$\bar{A}BC$	1
1	0	0	$A\bar{B}\bar{C}$	1
1	0	1	$A\bar{B}C$	0
1	1	0	$AB\bar{C}$	1
1	1	1	ABC	1



Slick MUX trick: Example

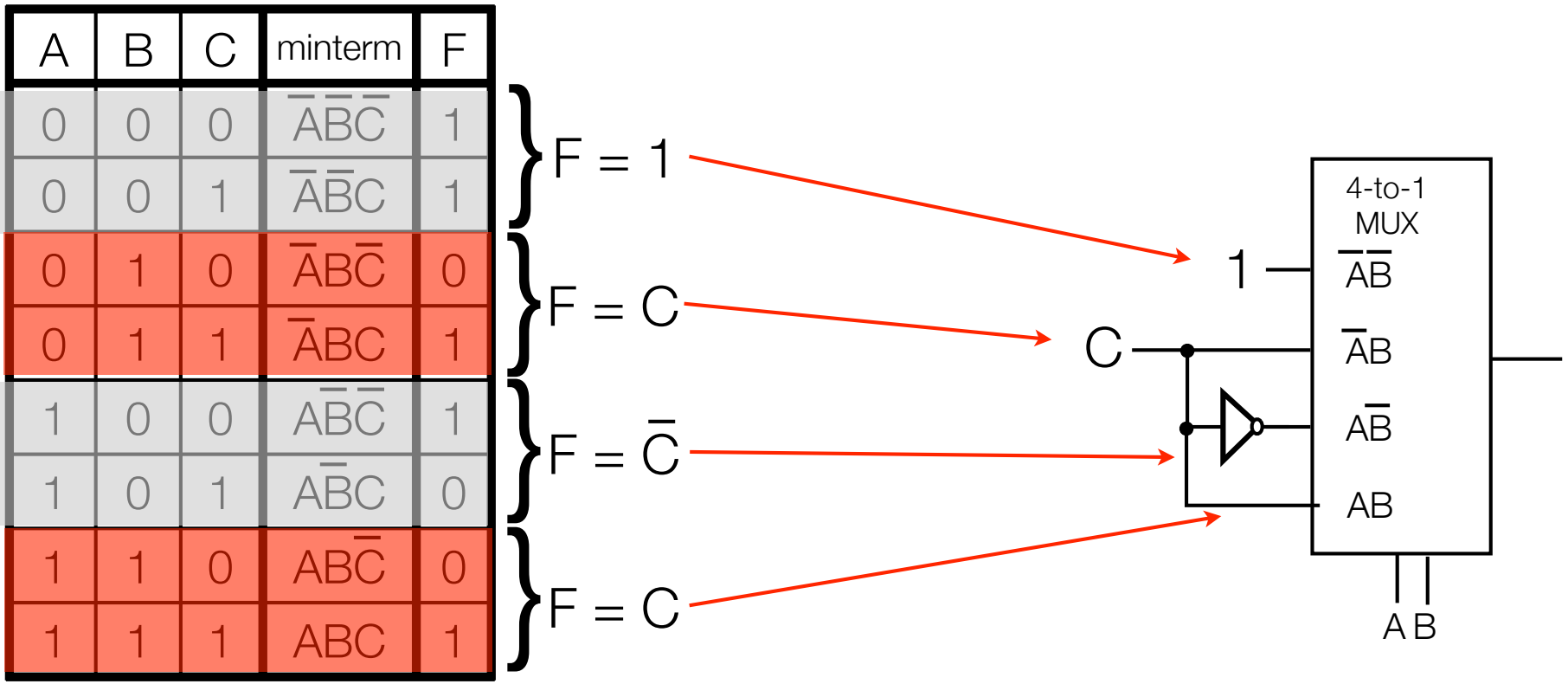
- e.g., F = $\bar{A}C + \bar{B}\bar{C} + A\bar{C}$

A	B	C	minterm	F
0	0	0	$\bar{A}\bar{B}\bar{C}$	1
0	0	1	$\bar{A}\bar{B}C$	1
0	1	0	$\bar{A}B\bar{C}$	0
0	1	1	$\bar{A}BC$	1
1	0	0	$A\bar{B}\bar{C}$	1
1	0	1	$A\bar{B}C$	0
1	1	0	$AB\bar{C}$	1
1	1	1	ABC	0



Slick MUX trick: Example 2

- e.g., $F = \bar{A}C + \bar{B}\bar{C} + BC$

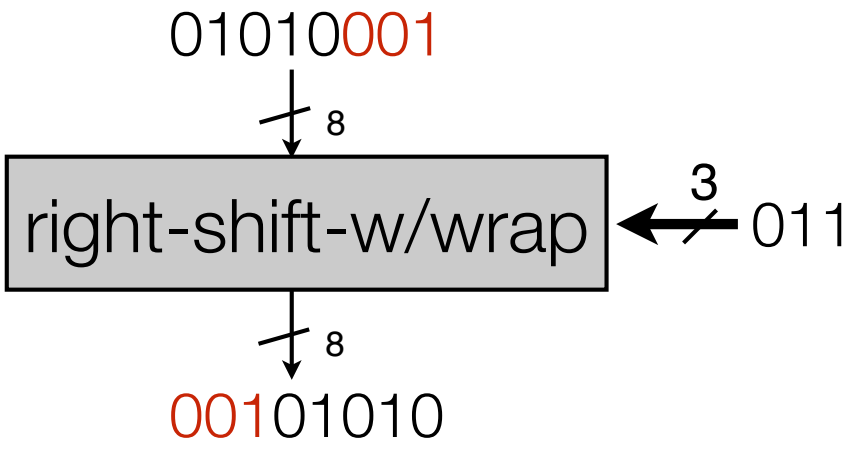


Shifter

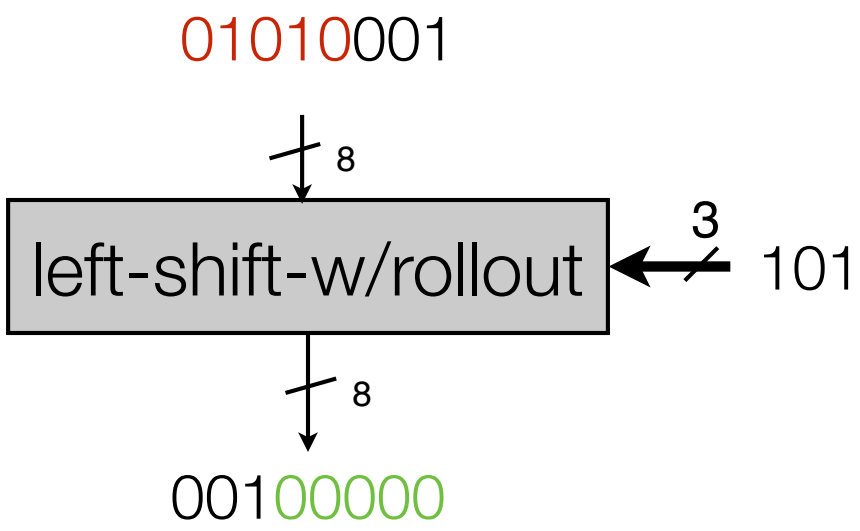
(Barrel) Shifter Circuit (High-level)

- k-bit data value entered as input
- ℓ = $\log_2$k-bit selector

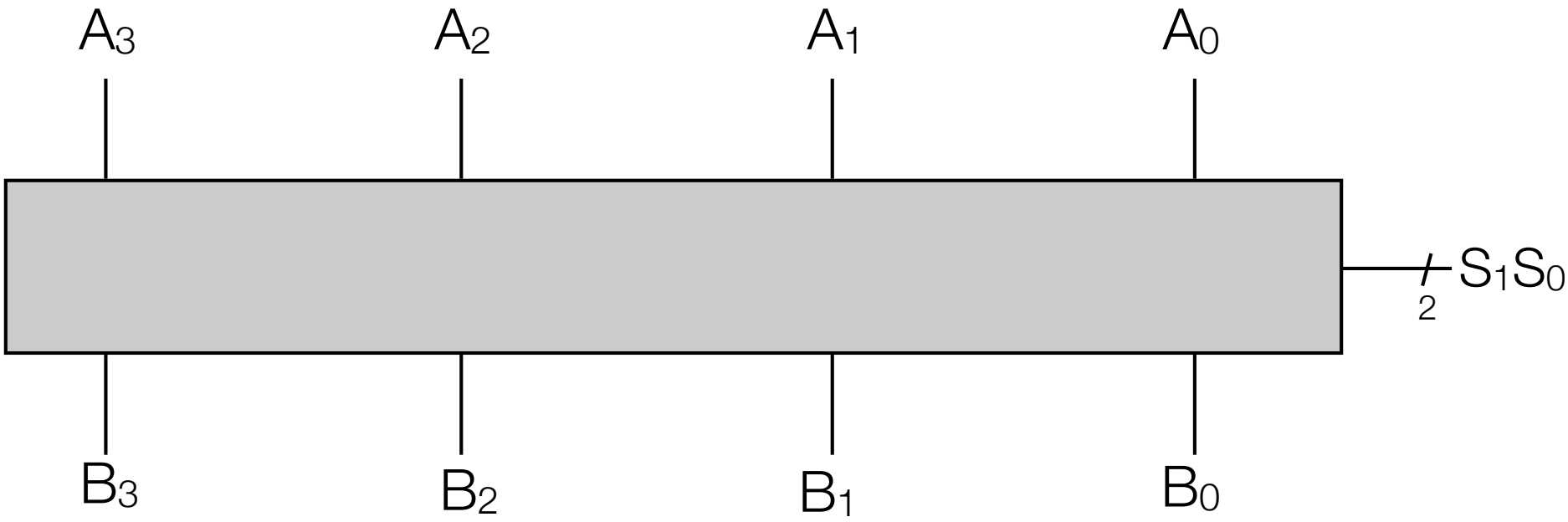
  chooses how far bits should be shifted
- Example 1: k=8, right shifter with "wraparound", asked to shift by 3 bits



- Example 2: k=8, left shifter with "rollout", asked to shift by 5 bits (fill missing bits with 0's



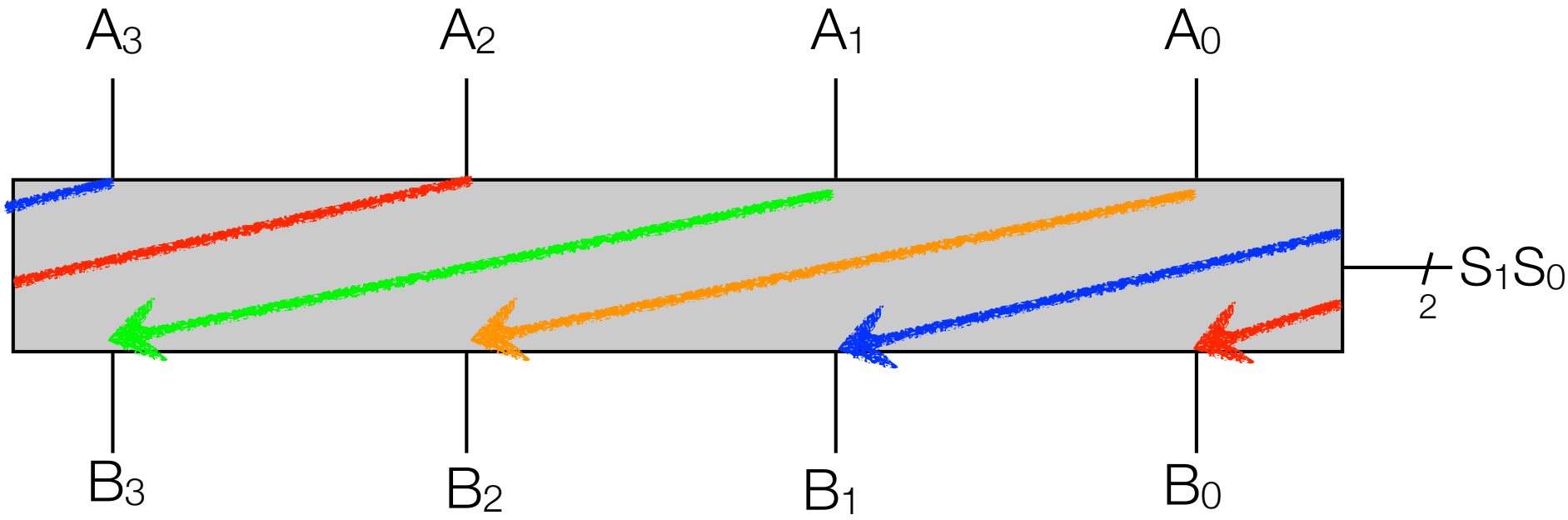
Barrel Shift-Left Design with wraparound (using MUXs)



- EXAMPLE: S = 2 (S_1S_0 = 10)

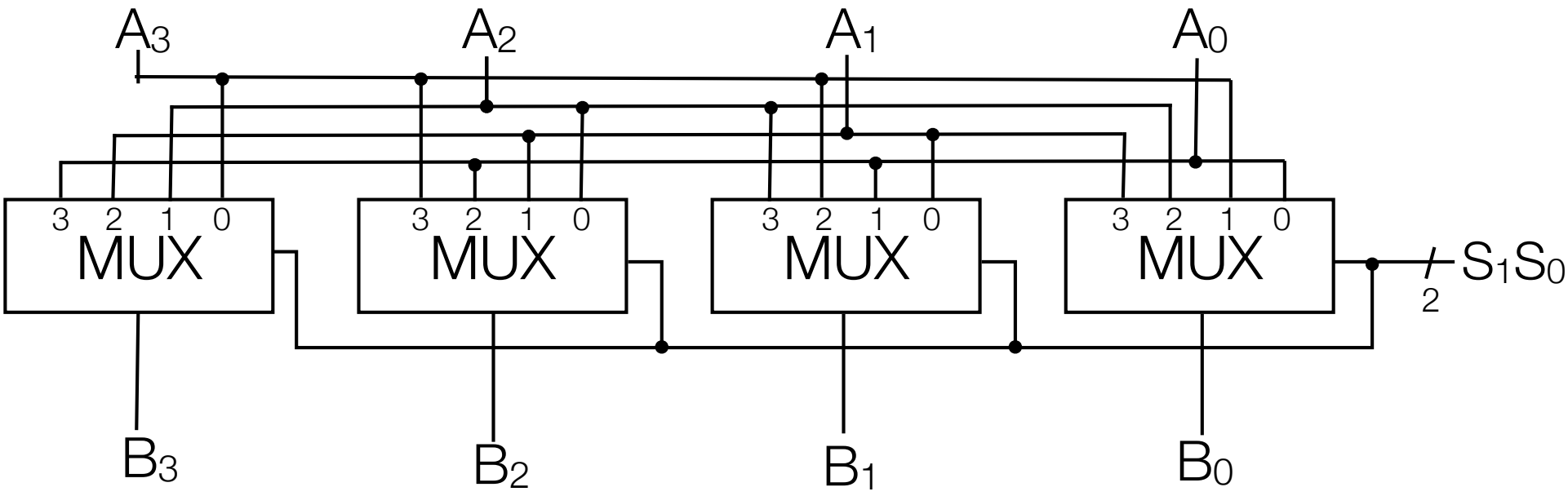
- $B_{i+2} = A_i$ (mod 4)

Barrel Shift-Left Design with wraparound (using MUXs)



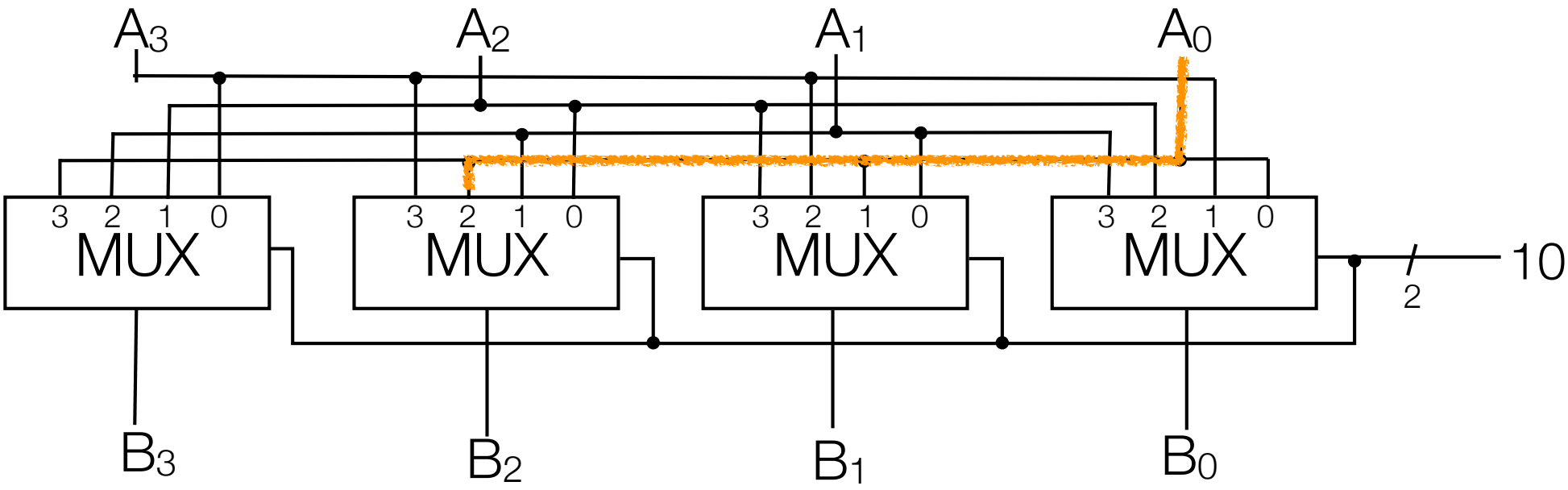
- EXAMPLE: S = 2 (S_1S_0 = 10)
- $B_{i+2} = A_i$ (mod 4)
- In general, if shifter = j, $B_{i+j} = A_i$ (mod # data bits)

Barrel Shift-Left Design with wraparound (using MUXs)



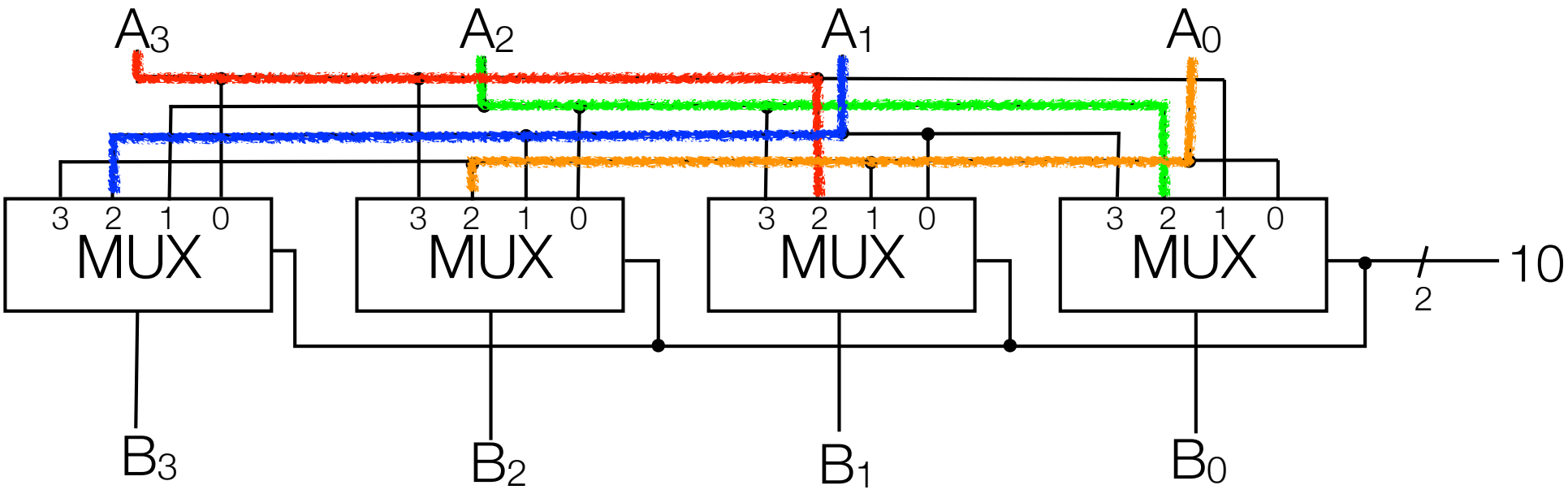
- Basic form of design: Each A_i feeds into each MUX connecting to B_j into input (j-i) mod 4

Barrel Shift-Left Design with wraparound (using MUXs)



- Basic form of design: Each A_i feeds into each MUX connecting to B_j into input (j-i) mod 4
- Selector is 10 (i.e., 2 binary):
- each MUX entry 2 is selected
  - A_0 flows into the '2' input of the MUX whose output is B_2

Barrel Shift-Left Design with wraparound (using MUXs)

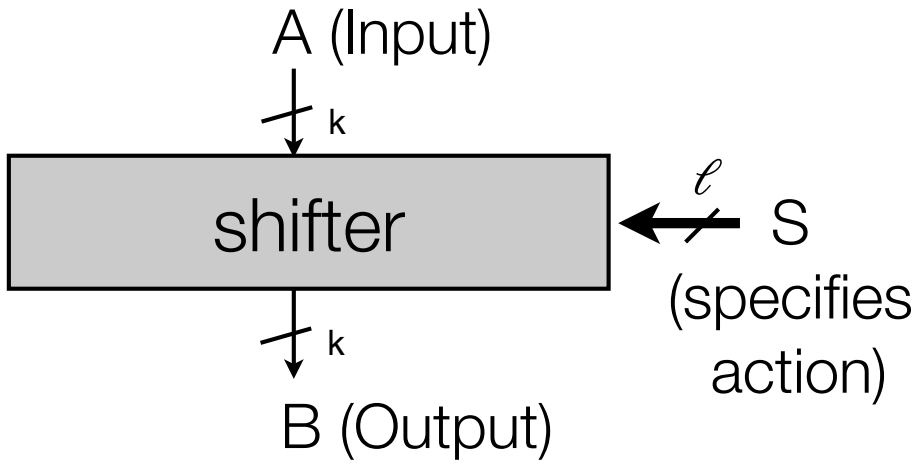


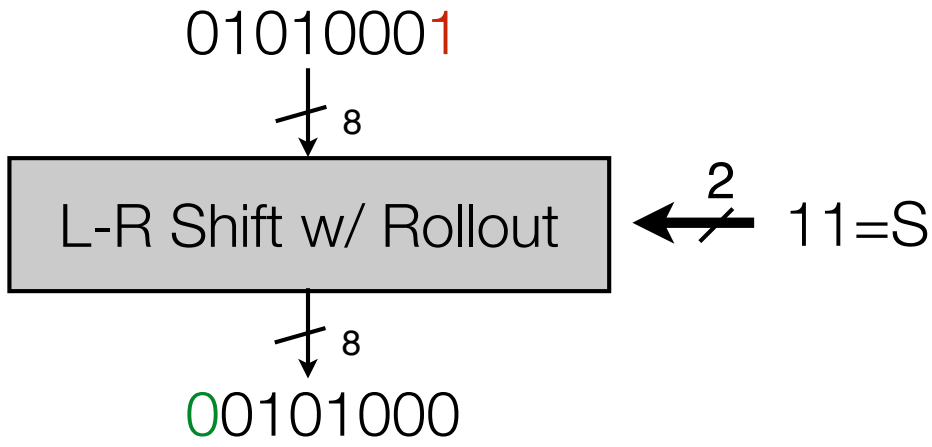
- Basic form of design: Each A_i feeds into each MUX connecting to B_j into input (j-i) mod 4
- Selector is 10 (i.e., 2 binary):
- each MUX entry 2 is selected
  - $B_3\ B_2\ B_1\ B_0 = A_1\ A_0\ A_3\ A_2$

L-R Shift Circuit with Rollout

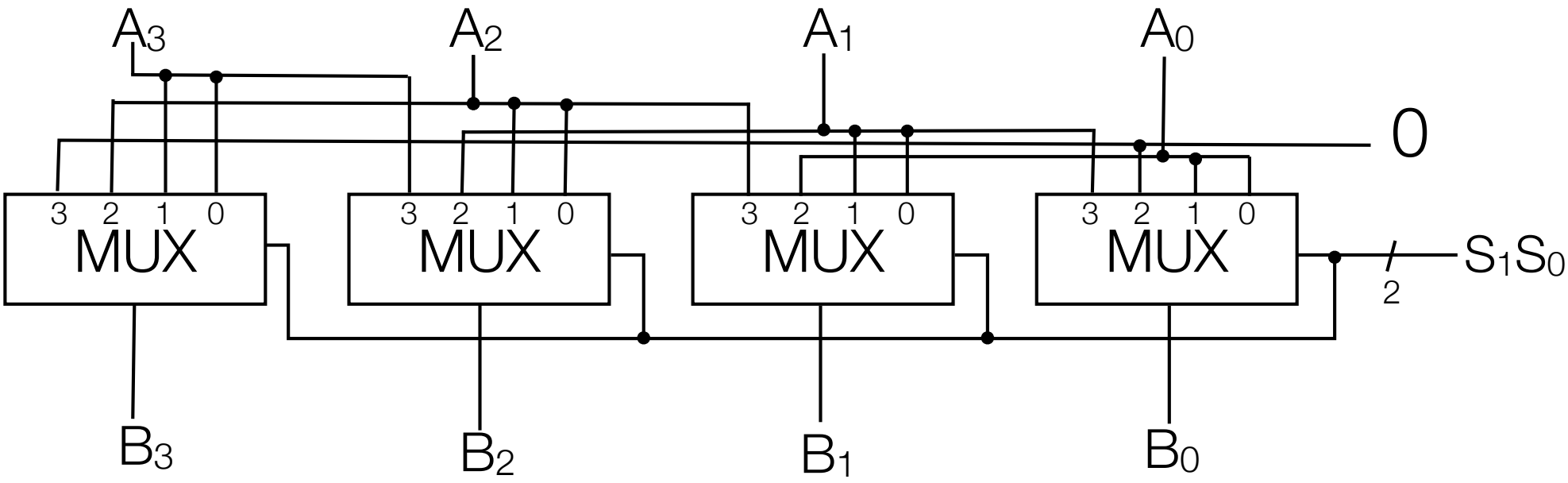
- k-bit data value entered as input
- ℓ = 2 Specifies action

S_1	S_0	Action
0	0	none
0	1	none
1	0	shift-left (by 1 bit)
1	1	shift-right (by 1 bit)



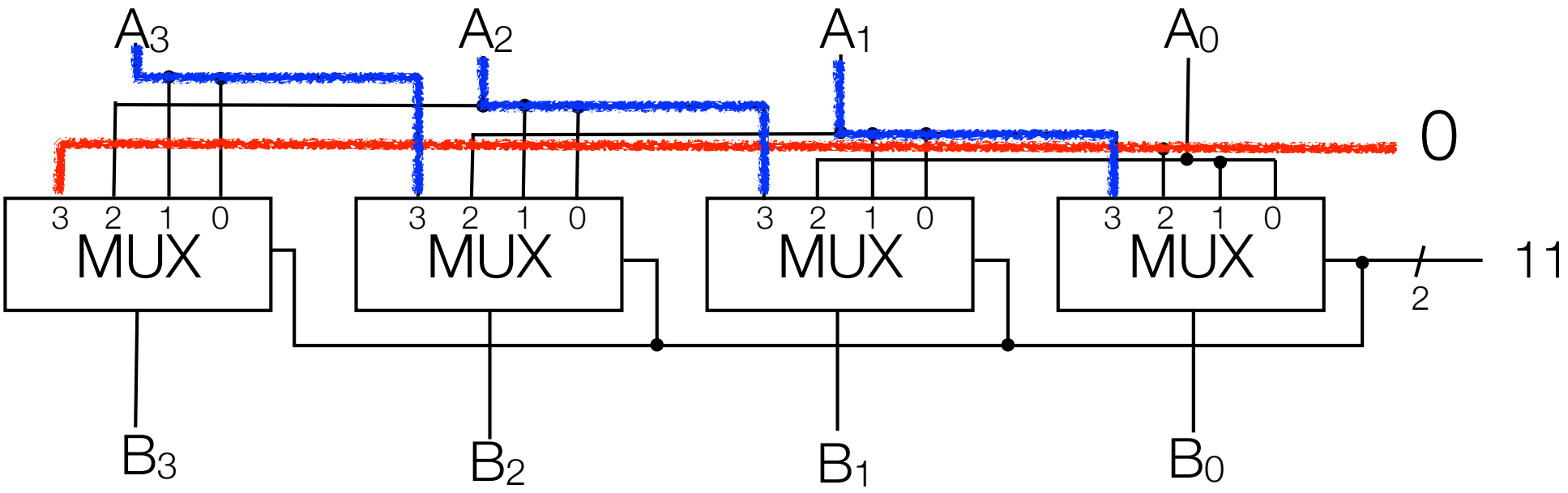


L/R Shift w/ Rollout



- Basic form of design:
  - 0 & 1 MUX selectors ($S_1 = 0$) feed A_i to B_i
  - 2 MUX selector feeds from left ($B_i = A_{i-1}$), 3 MUX from right ($B_i = A_{i+1}$)
  - Note 0 feeds (0's roll in when bits rollout)

L/R Shift w/ Rollout

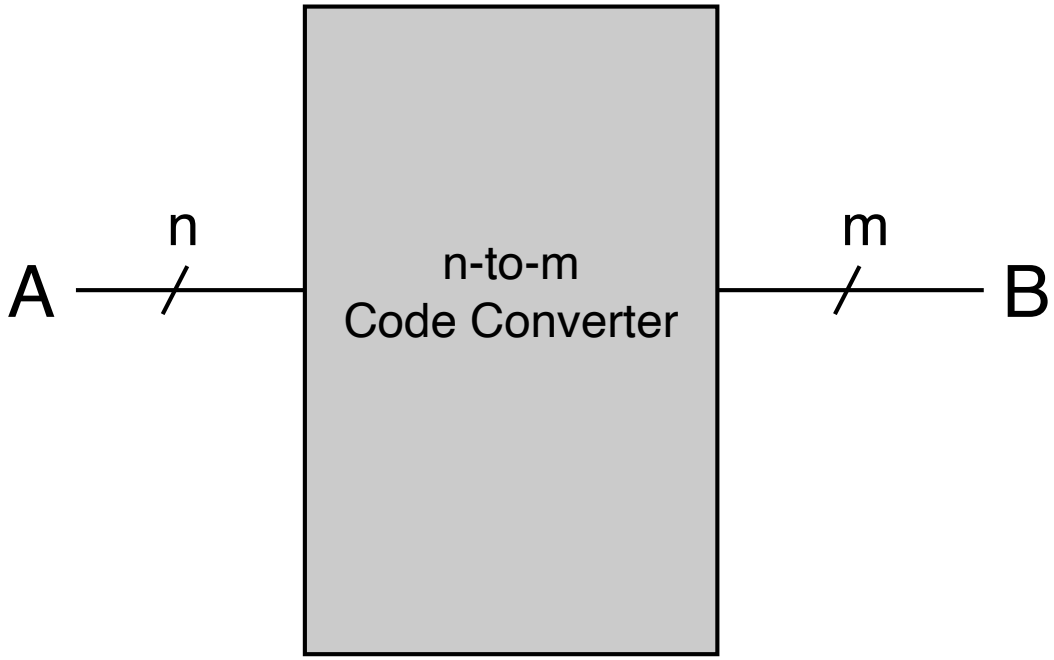


- Basic form of design:
  - 0 & 1 MUX selectors ($S_1 = 0$) feed A_i to B_i
  - 2 MUX selector feeds from left ($B_i = A_{i-1}$), 3 MUX from right ($B_i = A_{i+1}$)
  - Note 0 feeds (0's roll in when bits rollout)

General Code Converters

Code Converter

- General term for a circuit that takes n input A, produces m output B
- n,m can be any values
- Each of the m outputs is a function of the n inputs

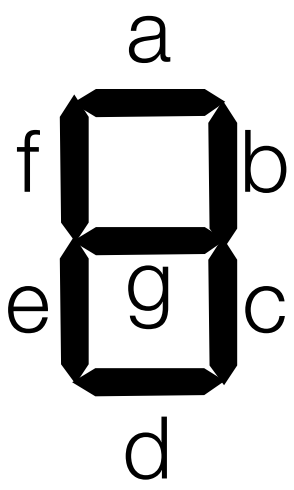


General Code Converter Example: LED digits

3-3

(a) Segment designation (b) Numeric designation for display

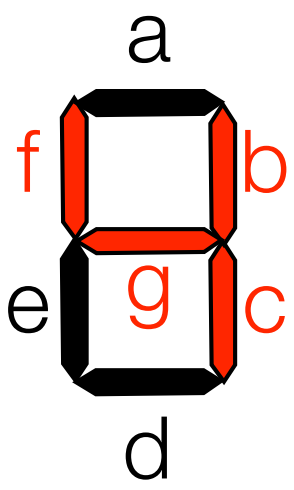
Code conversion



Input is 4 variables WXYZ, representing Va, an unsigned binary value (only care about 0-9)

	Input				Output						
Val	W	X	Y	Z	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1
X	1	0	1	0	X	X	X	X	X	X	X
X	1	0	1	1	X	X	X	X	X	X	X
X	1	1	0	0	X	X	X	X	X	X	X
X	1	1	0	1	X	X	X	X	X	X	X
X	1	1	1	0	X	X	X	X	X	X	X
X	1	1	1	1	X	X	X	X	X	X	X

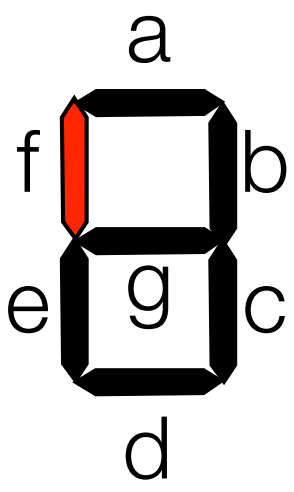
Code conversion



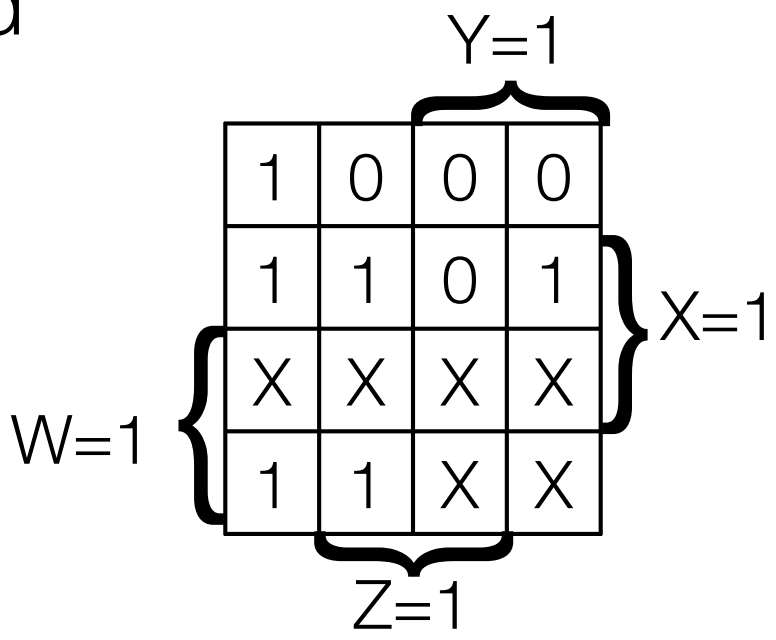
e.g., what outputs "light up" when input Va=4?

	Input				Output						
Val	W	X	Y	Z	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1
X	1	0	1	0	X	X	X	X	X	X	X
X	1	0	1	1	X	X	X	X	X	X	X
X	1	1	0	0	X	X	X	X	X	X	X
X	1	1	0	1	X	X	X	X	X	X	X
X	1	1	1	0	X	X	X	X	X	X	X
X	1	1	1	1	X	X	X	X	X	X	X

Code conversion

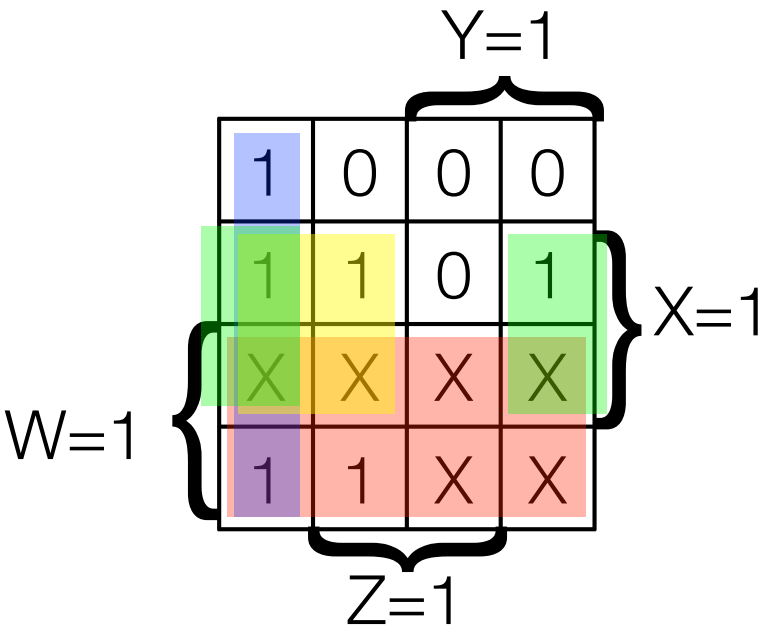


For what values does output f “light up” for?

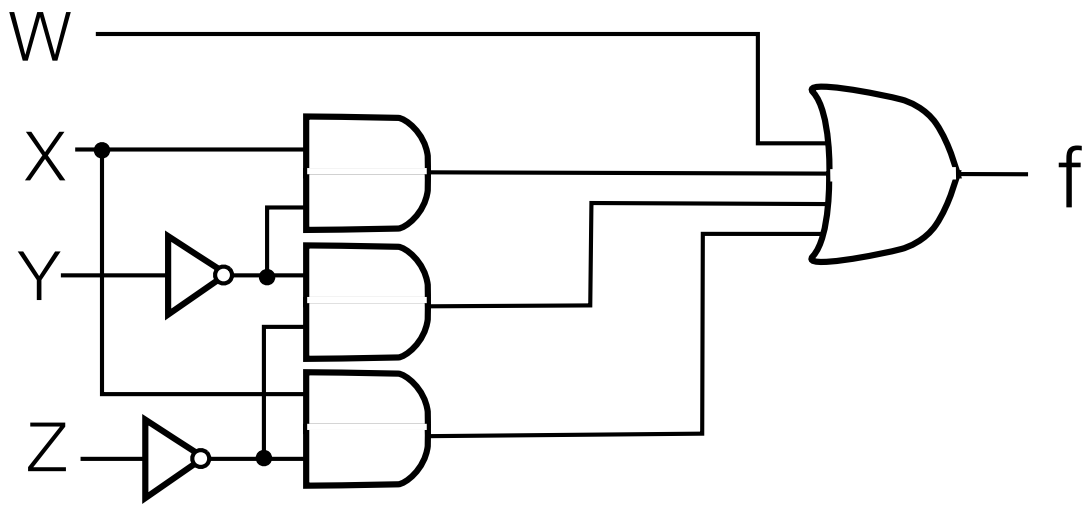


	Input				Output						
Val	W	X	Y	Z	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1
X	1	0	1	0	X	X	X	X	X	X	X
X	1	0	1	1	X	X	X	X	X	X	X
X	1	1	0	0	X	X	X	X	X	X	X
X	1	1	0	1	X	X	X	X	X	X	X
X	1	1	1	0	X	X	X	X	X	X	X
X	1	1	1	1	X	X	X	X	X	X	X

Algebra and Circuit for “f”



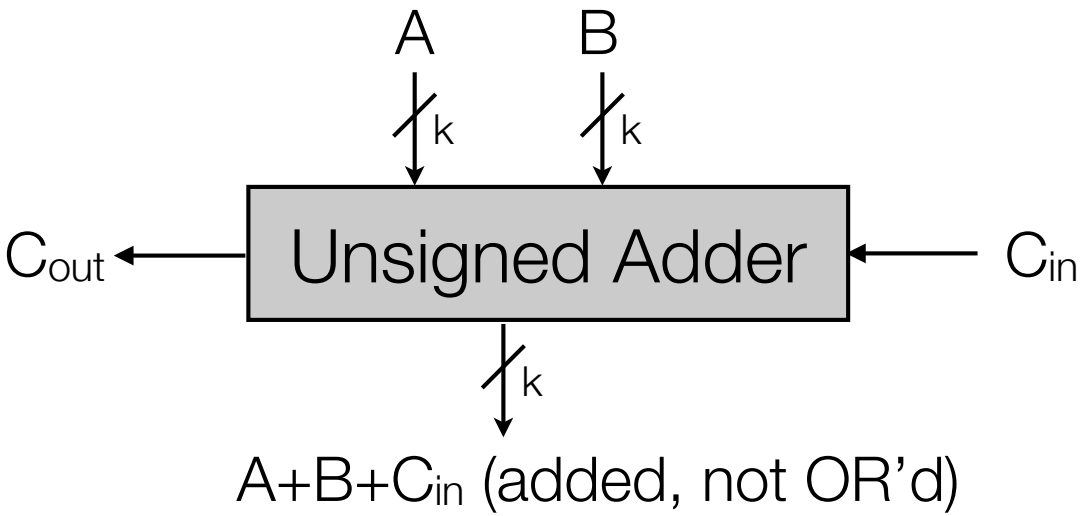
$$f = W + X\bar{Y} + \bar{Y}\bar{Z} + X\bar{Z}$$

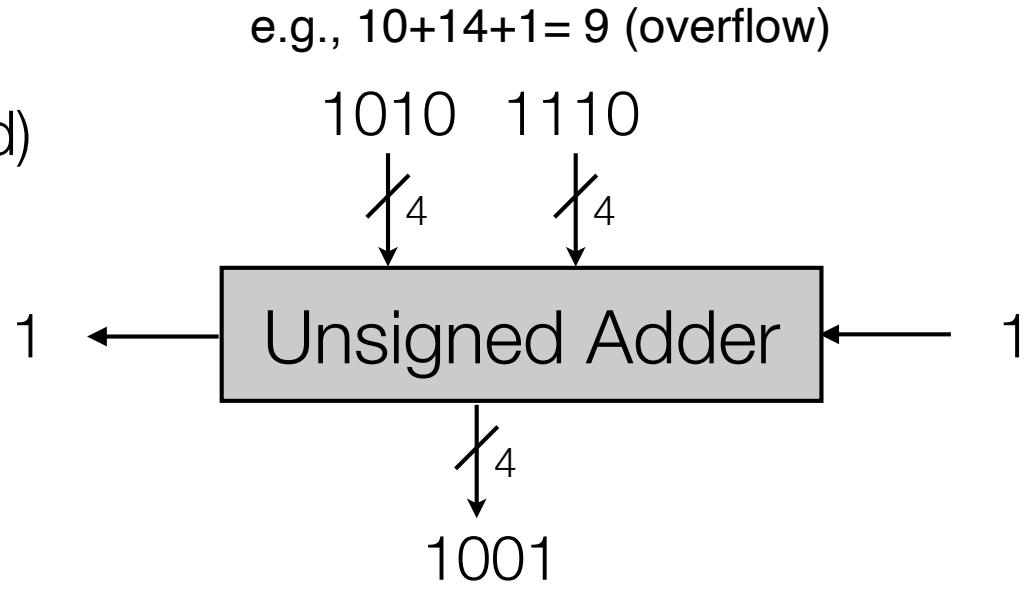


Adder Circuits
(Hierarchical Approach)

(Unsigned) Adder Circuit (High-level)

- Two k-bit data values, A and B, and 1 bit C_{in} (carry-in) entered as input
- output is k-bit A+B + C_{in} (unsigned binary addition), and 1-bit C_{out} (carry-out), i.e., the overflow





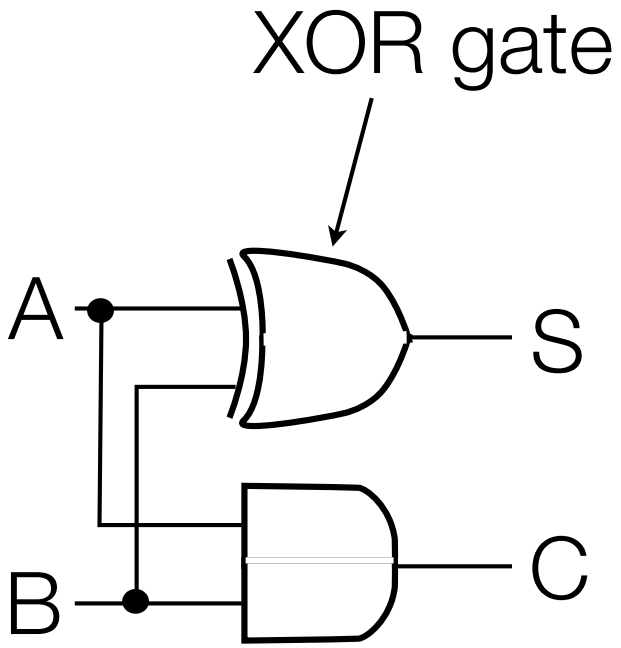
Addition: The Half-Adder

- Addition of 2 bits: A & B produces a summand (S) and carry (C)

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S = A \oplus B$$

$$C = AB$$



- But to do addition, we really need to add 3 bits at a time (to account for carries), e.g.,

```
  011   <-- carry bits
+ 1011
  1001
 10100
```

The Full Adder

- Takes as input 2 digits (A&B) and a previous carry (P)

P	A	B	C	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:

P \ B	0	1	0	1
1	1	0	1	0

A

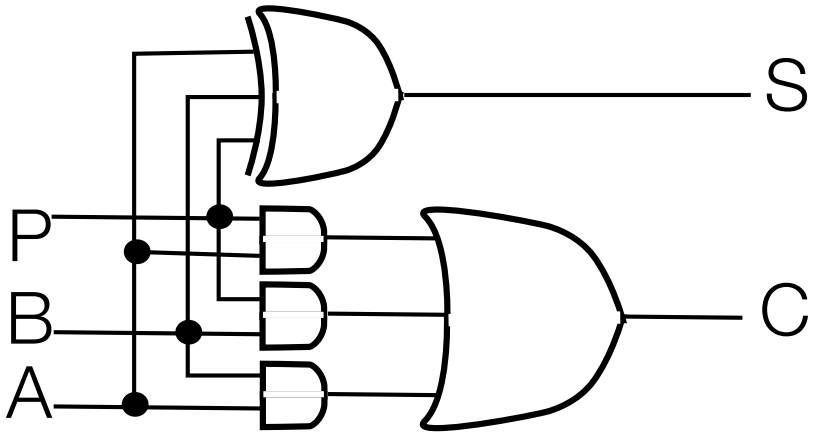
C:

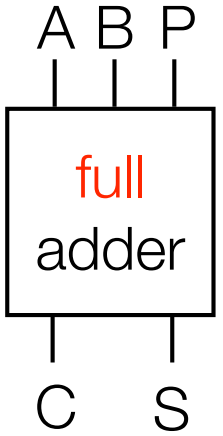
P \ B	0	0	1	0
0	0	1	1	1

A

$$S = A \oplus B \oplus P$$

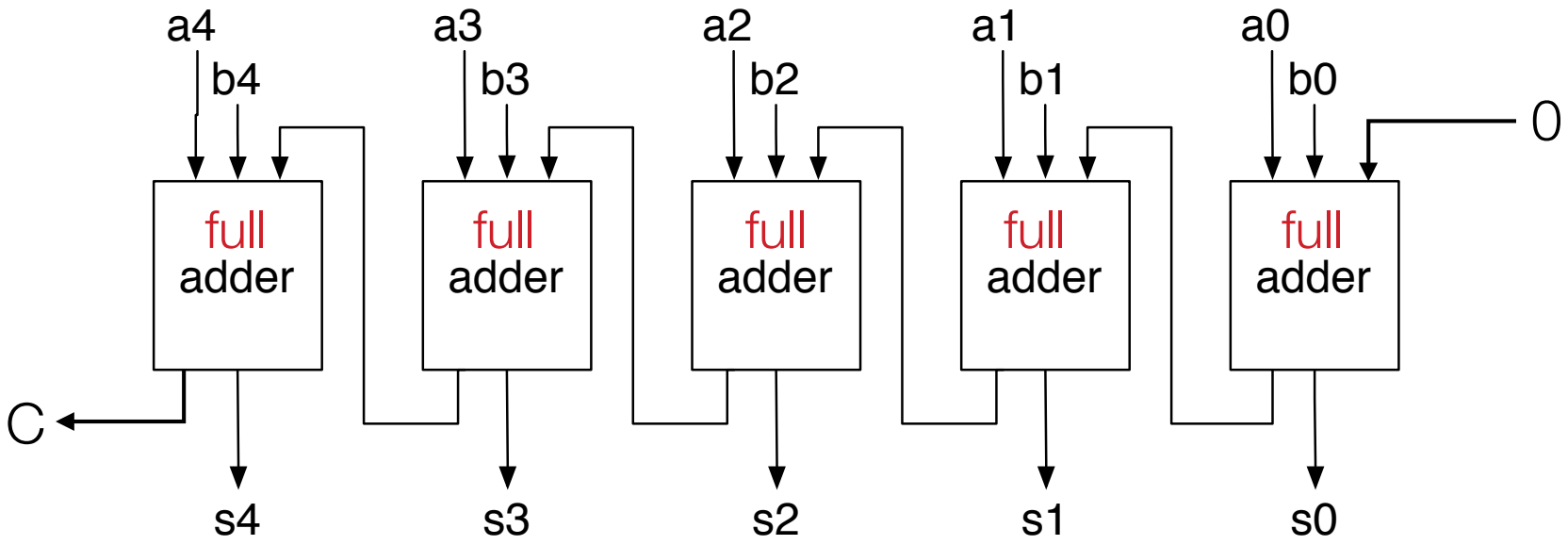
$$C = AB + AP + BP$$





5-bit ripple carry adder

Computes $a_4a_3a_2a_1a_0 + b_4b_3b_2b_1b_0$



- Note how computation "ripples" through adders from right to left
  - Each full adder's has depth 2 (inputs pass through 2 gates to reach output)
  - Full adder that computes s_i cannot "start" its computation until previous full adder computes carry
  - The longest depth in a k-bit ripple carry adder is 2k

Subtraction review (2's complement)

- Suppose A and B are 2's complement numbers:
- To perform A-B:
- Version 1:
  - Negate B:
    - B_F = Flip Bits in B
    - Y = B_F+1, Y=-B
  - return A+Y

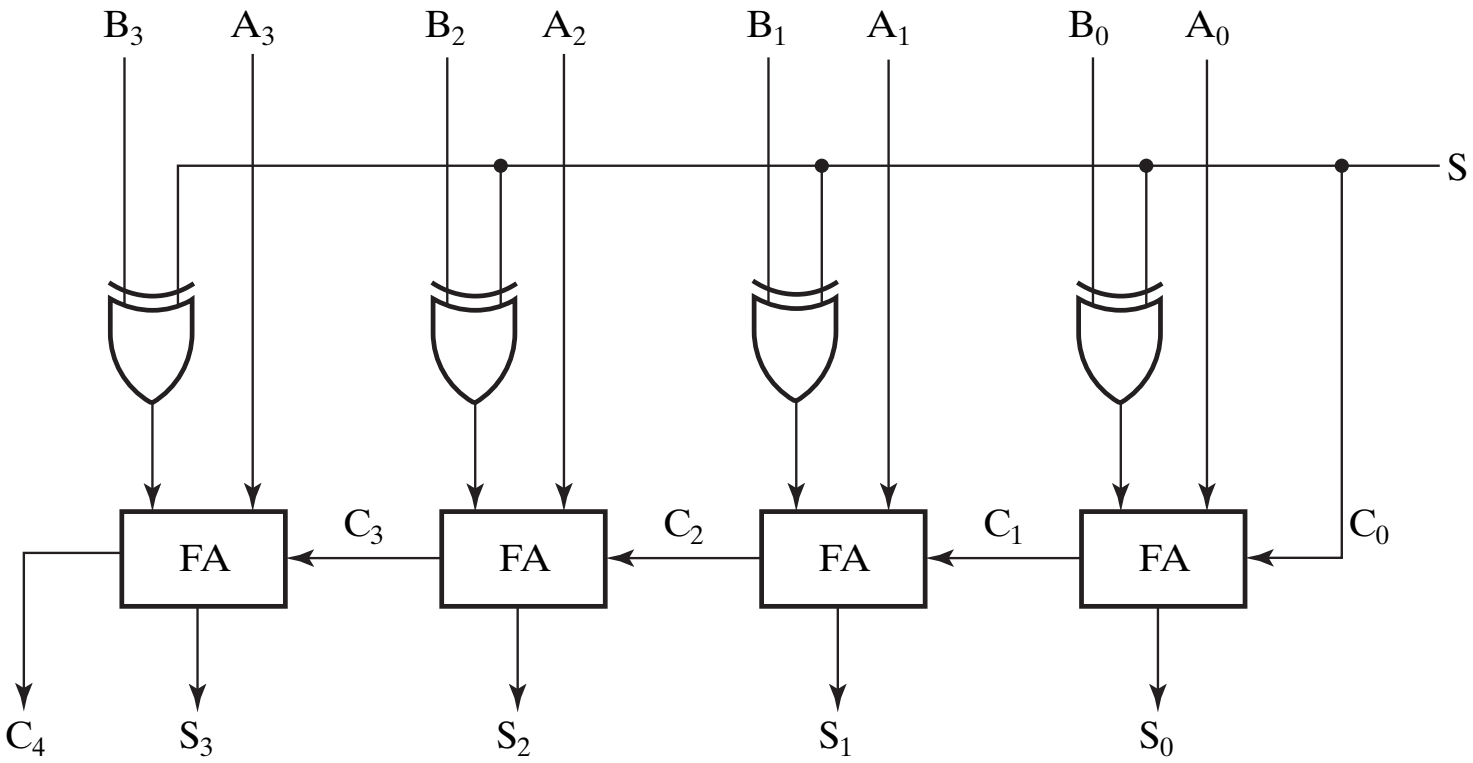
Subtraction review (2's complement)

- Suppose A and B are 2's complement numbers:
- To perform A-B:
- Version 1:
  - Negate B:
    - B_F = Flip Bits in B
    - Y = B_F+1, Y=-B
  - return A+Y
- Note: A+Y = A+(B_F+1) = (A+B_F)+1
- Could first compute B_F then just add A+B_F+1
- This is how our add/subtract circuit will do it…

XOR refresher

- $X \oplus 0 = X$
- $X \oplus 1 = \overline{X}$ (i.e., XOR'ing with 1 flips the bit)

Adder/subtractor for #'s in 2's complement form

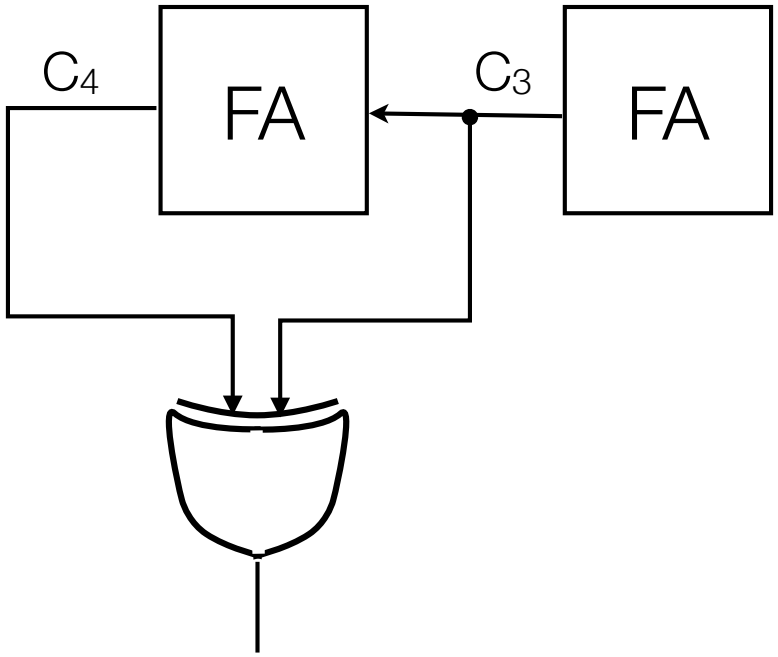


S=0: B unchanged, C_0=0: add

S=1: B complemented, C_0=1 (bits flipped and 1 added): subtract

Overflow computation in 2's comp add/subtractor

- Recall: for 2’s complement, overflow if 2 most significant carries differ



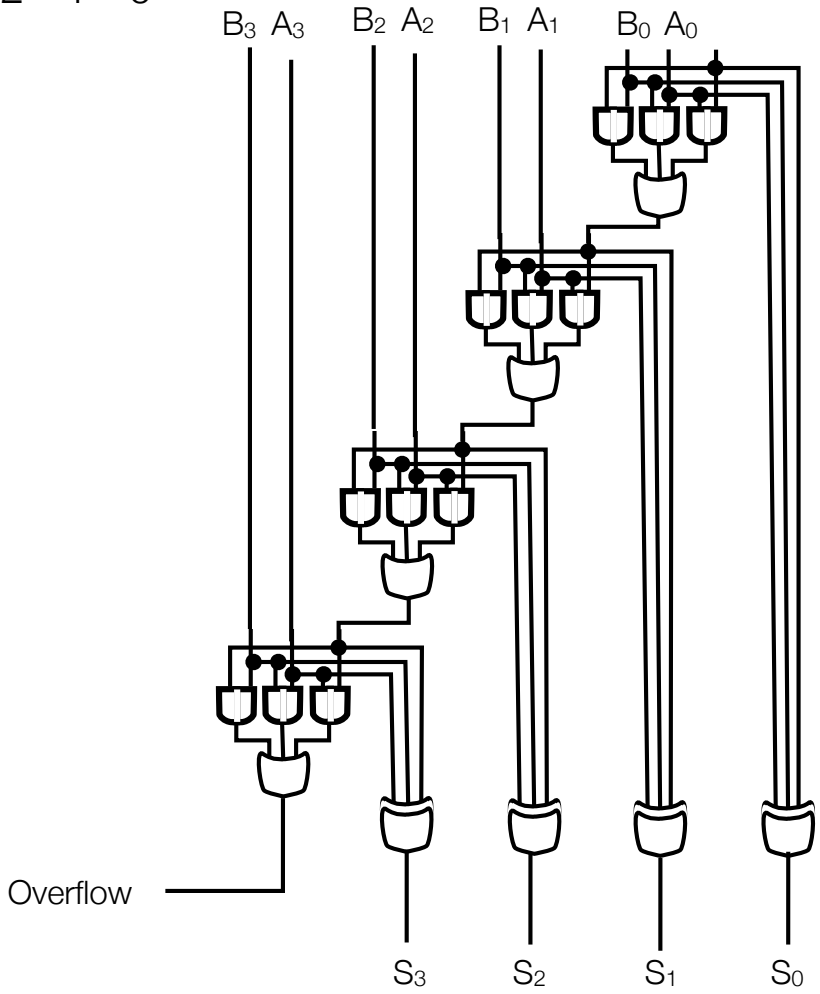
=0 then no overflow, =1 then overflow

Ripple Carry Adder
v
Carry Lookahead Adder

We built a Ripple-Carry adder: what is circuit depth?

$A_3A_2A_1A_0 + B_3B_2B_1B_0 = S_3S_2S_1S_0$

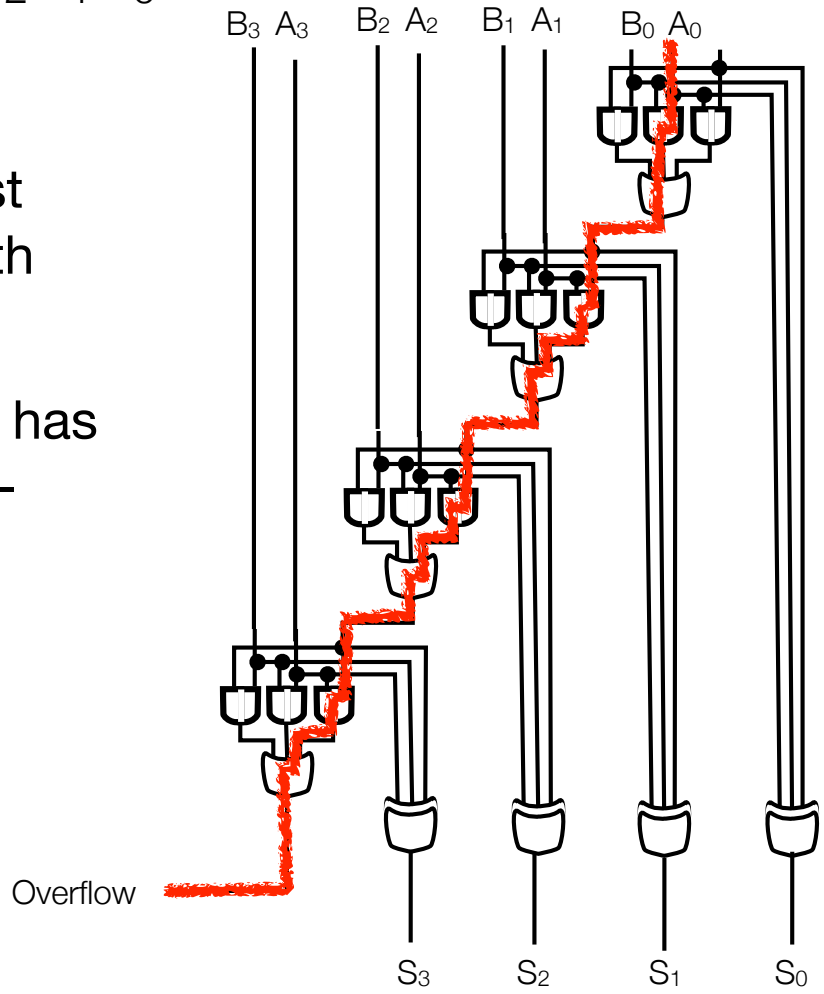
- Depth of a circuit is the longest (most gates to go through) path



We built a Ripple-Carry adder: what is circuit depth?

$A_3A_2A_1A_0 + B_3B_2B_1B_0 = S_3S_2S_1S_0$

- Depth of a circuit is the longest (most gates to go through) path
- Each carry computation stage has depth 3 (parallel ANDs, then 3-input OR)
- In general, C_k has depth 3k in Ripple-Carry Adder
- Linear depth!!!



Q: When does the i^{th} column yield a carry?

Column	i+1	i	i-1	i-2	▪ ▪ ▪
A					
B					

- When does column i produce a carry of 1?

Q: When does the i^{th} column yield a carry?

Column	i+1	i	i-1	i-2	▪ ▪ ▪
A		1			
B		1			

- When does column i produce a carry of 1?
- Both A and B are 1 in the i^{th} column: definitely have carry

Q: When does the i^{th} column yield a carry?

Column	i+1	i	i-1	i-2	▪ ▪ ▪
A		0			
B		0			

- When does column i produce a carry of 1?
- Both A and B are 0 in the i^{th} column: definitely no carry

Q: When does the i^{th} column yield a carry?

Column	i+1	i	i-1	i-2	▪ ▪ ▪
A		1 or 0			
B		0 or 1			

- When does column i produce a carry of 1?

- A and B differ in i^{th} column: column i carries if and only if column i-1 carries

$$\begin{matrix}1\\1\end{matrix} \quad \begin{matrix}0\\1\end{matrix}\ \text{or}\ \begin{matrix}1\\0\end{matrix} \quad \begin{matrix}1\\1\end{matrix} \qquad \begin{matrix}0\\1\end{matrix}\ \text{or}\ \begin{matrix}1\\0\end{matrix} \quad \begin{matrix}0\\1\end{matrix}\ \text{or}\ \begin{matrix}1\\0\end{matrix} \quad \begin{matrix}1\\1\end{matrix}$$

- i.e., $C_i = A_iB_i + (A_i \oplus B_i)A_{i-1}B_{i-1} + (A_i \oplus B_i)(A_{i-1} \oplus B_{i-1})A_{i-2}B_{i-2} + \ldots$

$$\begin{matrix}0\\1\end{matrix}\ \text{or}\ \begin{matrix}1\\0\end{matrix} \quad \begin{matrix}0\\1\end{matrix}\ \text{or}\ \begin{matrix}1\\0\end{matrix} \quad \begin{matrix}0\\1\end{matrix}\ \text{or}\ \begin{matrix}1\\0\end{matrix} \quad \begin{matrix}0\\1\end{matrix}\ \text{or}\ \begin{matrix}1\\0\end{matrix} \quad \begin{matrix}1\\1\end{matrix}$$

$$+ (A_i \oplus B_i)(A_{i-1} \oplus B_{i-1})\ldots(A_2 \oplus B_2)(A_1 \oplus B_1)\ A_0B_0$$

Depth to produce C_k?

- i.e., $C_k = A_kB_k + (A_k \oplus B_k)A_{k-1}B_{k-1} + (A_k \oplus B_k)(A_{k-1} \oplus B_{k-1})A_{k-2}B_{k-2} + \ldots$

  $+ (A_k \oplus B_k)(A_{k-1} \oplus B_{k-1})\ldots(A_2 \oplus B_2)(A_1 \oplus B_1)\, A_0B_0$

- What is depth of above circuit?

(3) XOR pairwise-AND (1)

AND at most k+1 of the above together ($\log_2(k+1)$)

OR at most k+1 of the above together ($\log_2(k+1)$)

$3 + \log_2(k+1) + \log_2(k+1) = 3 + 2\log_2(k+1)$

Final Thoughts on Carry Lookahead Adder

- Showed C_k can be computed with depth $3 + 2 \log_2(k+1)$
- Actually can be reduced to depth $1 + 2 \log_2(k+1)$
  - (XORs can be replaced by ORs,
  - e.g., consider $A_iB_i + (A_i \oplus B_i)A_{i-1}B_{i-1}$: replace with $A_iB_i + (A_i + B_i)A_{i-1}B_{i-1}$
  - The product term A_iB_i already forces expression true when $A_i=B_i=1$
- Depth of k-1st sum bit adds depth 2 to perform XOR
- Bottom line: depth $3 + 2 \log_2(k+1) << 3k$ for large k
- e.g., k=32, 15 vs 96
- e.g., k=64, 17 vs 192

Contraction

Contraction

- Simplification of a circuit through constant input values.

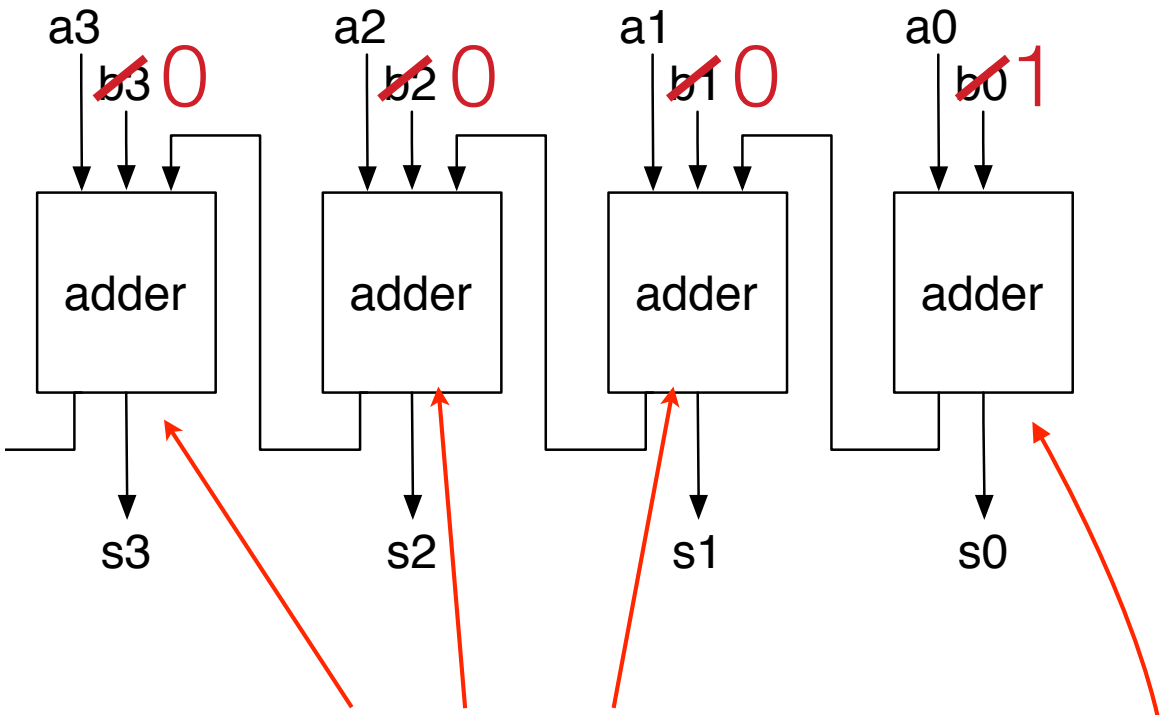
- e.g., suppose you want to build an “increment” circuit that adds 1 to an input value

  - step 1: take an all-purpose adder

  - step 2: fix one of the inputs to equal “1”

  - step 3: reduce the circuitry

Contraction example: adder to incrementer



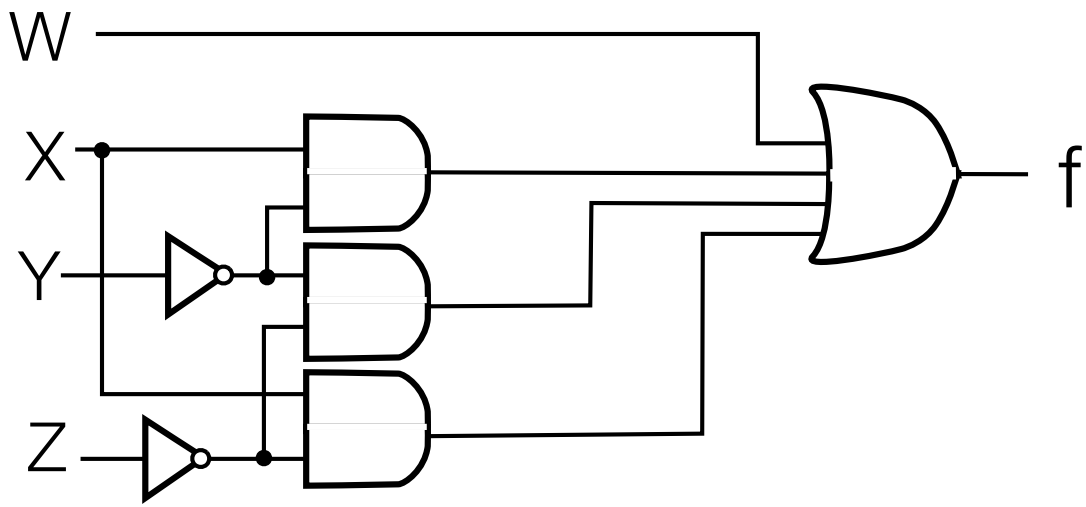
Incrementer circuit

a_0	S	C
0	1	0
1	0	1

$S_0=\overline{a_0}$, $C_0=a_0$

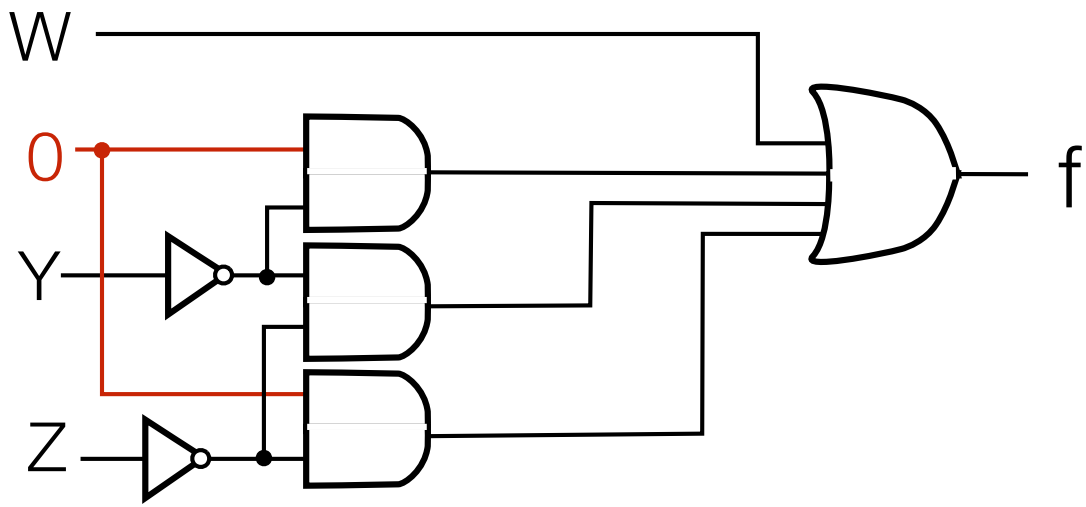
Example 2 of contraction

Suppose: use circuit below in cases where X=0 always



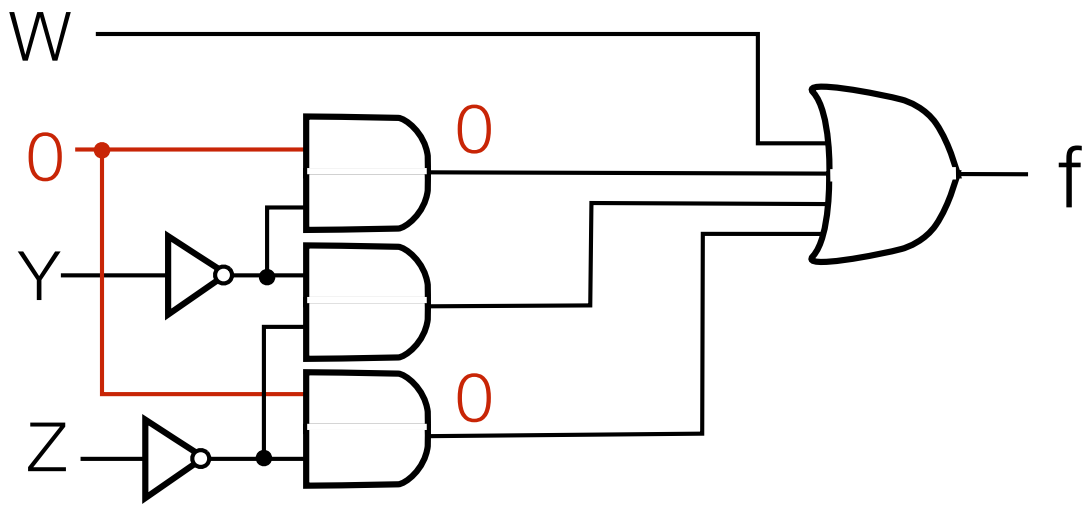
Example 2 of contraction

Suppose: use circuit below in cases where X=0 always



Example 2 of contraction

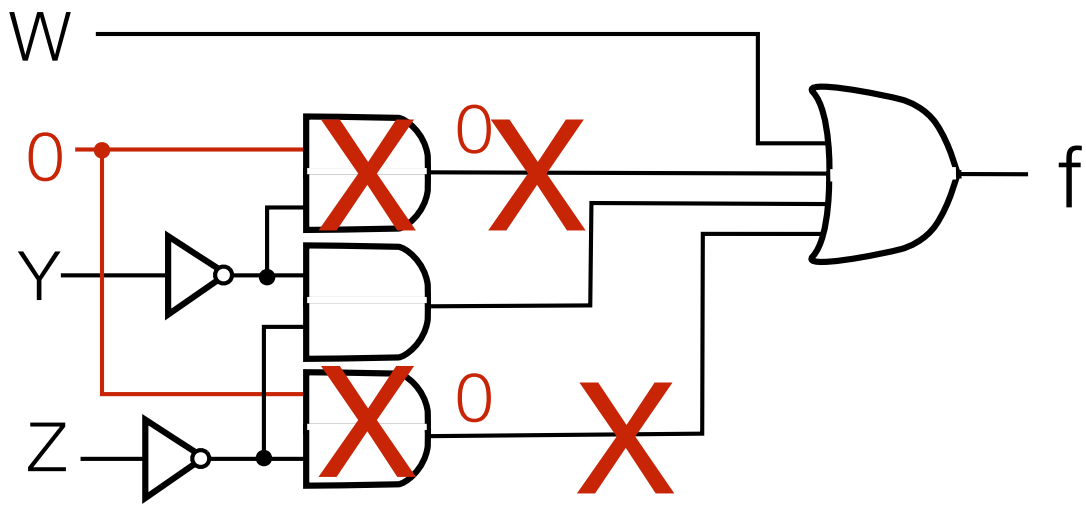
Suppose: use circuit below in cases where X=0 always



Example 2 of contraction

Suppose: use circuit below in cases where X=0 always

Can remove gates whose outcome is known (is constant)



Example 2 of contraction

Suppose: use circuit below in cases where X=0 always