Combinational Logic

CMPT 295 Week 10.1

Hardware Design

- Understand how processors work
  - Requires digital systems knowledge
- Understand how code is actually executed on a computer to analyze:
  - Reliability
  - Performance
  - Security
- Layered abstractions
  - Transistors → Combinational Logic → Sequential Logic → Processors → Machine Language → Assembly → High-level Programming Languages → Application programs
  - At each step we can “abstract away” the lower layers

Synchronous Digital Systems (SDS)

Hardware of a processor (e.g., RISC-V) is an example of a Synchronous Digital System

Synchronous:

- All operations coordinated by a central clock
  - “Heartbeat” of the system (processor frequency)

Digital:

- Represent all values with two discrete values
- Electrical signals are treated as 1’s and 0’s
  - High/Low voltage represent True/False, 1/0

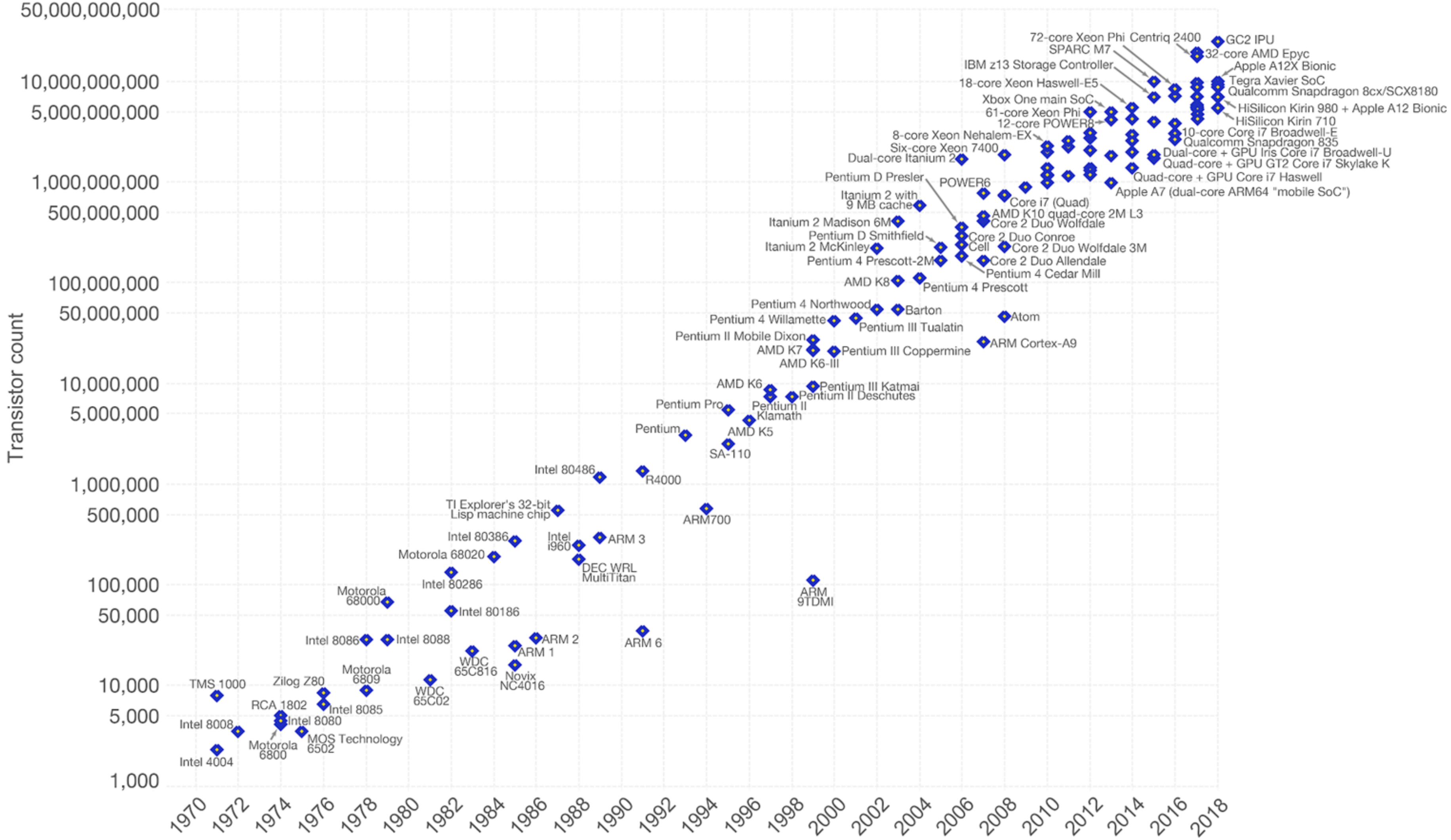
Moore’s Law

- **Original Version (1965):** Since the integrated circuit was invented, the number of transistors in an integrated circuit has roughly doubled every year; this trend would continue for the foreseeable future
- 1975: Revised - circuit complexity doubles every two years

Moore’s Law – The number of transistors on integrated circuit chips (1971-2018)

Our World in Data

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

Moore's Law

- **Original Version (1965):** Since the integrated circuit was invented, the number of transistors in an integrated circuit has roughly doubled every year; this trend would continue for the foreseeable future
- 1975: Revised - circuit complexity doubles every two years
- **Hardware Trend:** Hardware gets more powerful every year (due to technology advancement and the hard work of many engineers)
- **Software Trend:** Software gets faster and uses more resources (And has to keep up with ever-changing hardware)
- Digital circuits are used to build hardware

Combinational vs. Sequential Logic

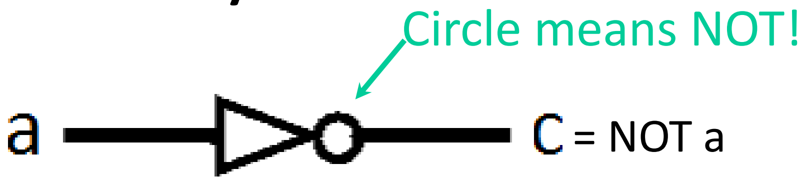
- *Digital Systems* consist of two basic types of circuits:
  - Combinational Logic (CL)
    - Output is a function of the inputs only, not the history of its execution
    - Example: add A, B (ALUs)
  - Sequential Logic (SL)
    - Circuits that “remember” or store information
    - Also called “State Elements”
    - Example: Memory and registers

Simple Logic Gates

- Special names and symbols:

Truth Table

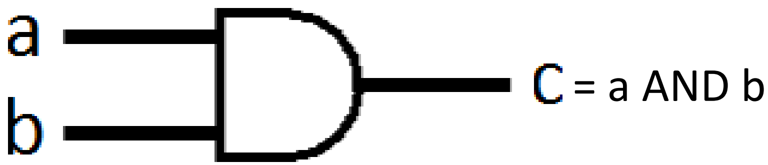
NOT



a	NOT a
0	1
1	0

True if input is false

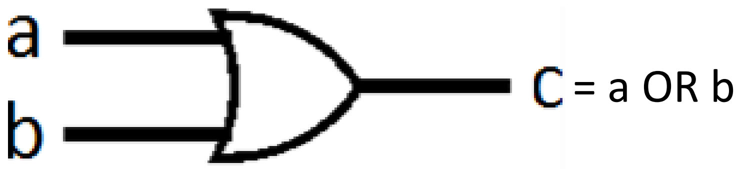
AND



a	b	a AND b
0	0	0
0	1	0
1	0	0
1	1	1

True if both inputs are true

OR



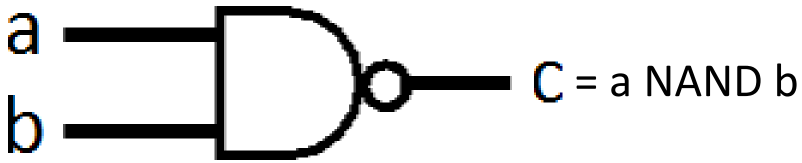
a	b	A OR b
0	0	0
0	1	1
1	0	1
1	1	1

True if at least one input is true

More Simple Logic Gates

Inverted versions are easier to implement in CMOS

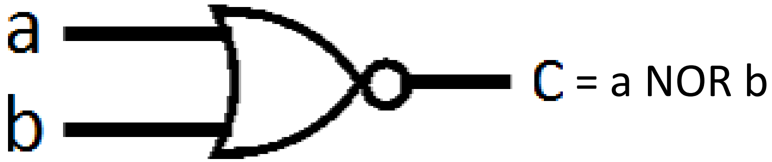
NAND



True if at least one input is false

a	b	a NAND b
0	0	1
0	1	1
1	0	1
1	1	0

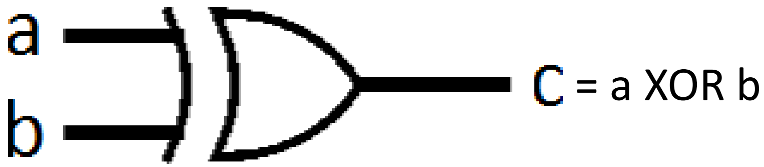
NOR



True if both inputs are false

a	b	a NOR b
0	0	1
0	1	0
1	0	0
1	1	0

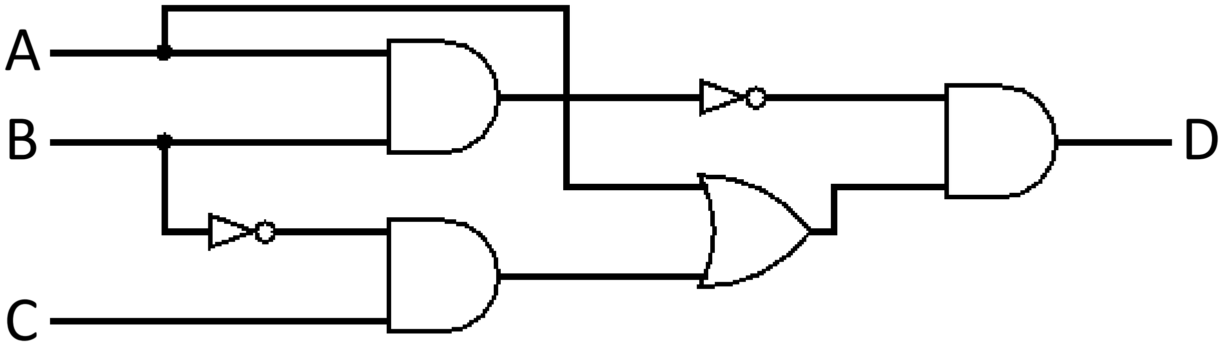
XOR



True if exactly one input is true

a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

Combining Multiple Logic Gates



D = (NOT(**A** AND **B**)) AND (**A** OR (NOT **B** AND **C**))

How to Represent Combinational Logic?

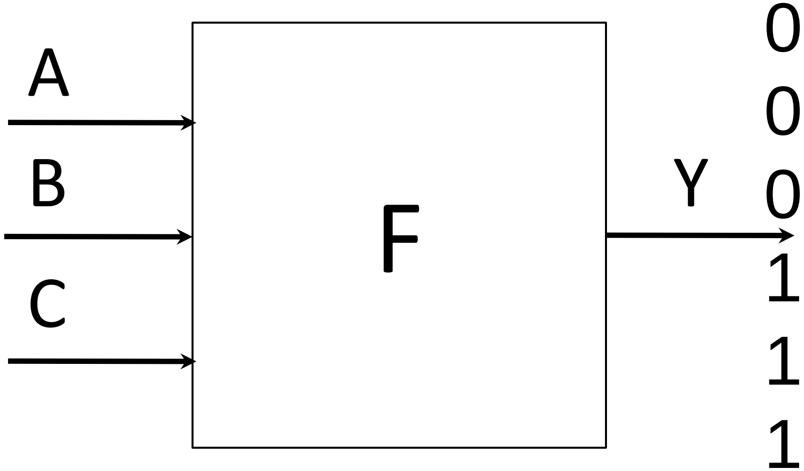
- ✓ Text Description
- ✓ Circuit Diagram
  - – Transistors and wires
  - – Logic Gates
- ✓ Truth Table
- ✓ Boolean Expression

✓ *All are equivalent*

Truth Tables

- Table that relates the inputs to a combinational logic circuit to its output
  - Output *only* depends on current inputs
  - Use abstraction of 0/1 (F/T) instead of high/low Voltage
  - Shows output for *every* possible combination of inputs
- How big is a truth table with N inputs?
  - 0 or 1 for each of N inputs, so 2^N rows

N-input Truth Tables

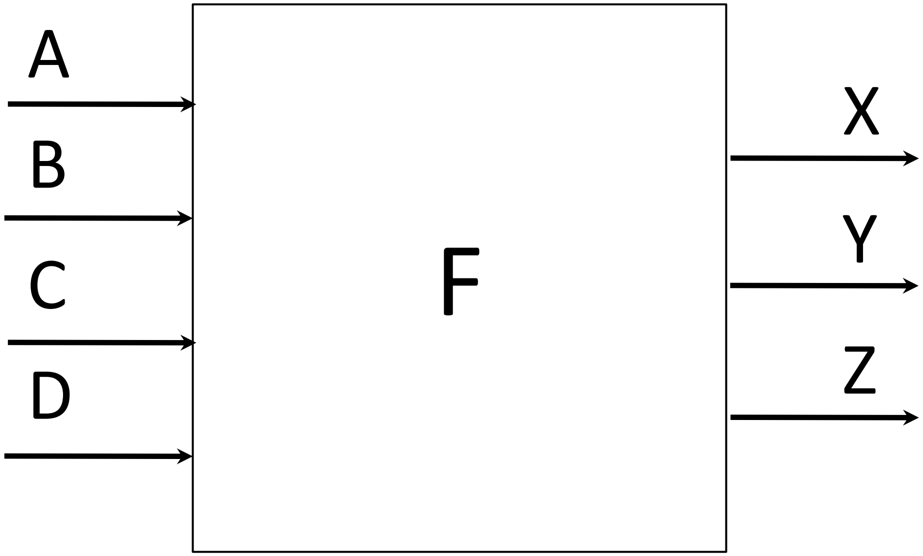


A	B	C		Y
0	0	0	F(0,0,0)	0
0	0	1	F(0,0,1)	1
0	1	0	F(0,1,0)	0
0	1	1	F(0,1,1)	0
1	0	0	F(1,0,0)	0
1	0	1	F(1,0,1)	1
1	1	0	F(1,1,0)	1
1	1	1	F(1,1,1)	0

For N inputs, how many distinct functions F do we have?

Function maps each row to 0 or 1, so 2^{2^N}possible functions

Truth Tables with Multiple Outputs



- For 3 outputs, just three indep. functions: X(A,B,C,D), Y(A,B,C,D), and Z(A,B,C,D)
  - Can show functions in separate columns (no additional rows)

Question: Which of the columns A-D is the correct output of the Truth Table for: (X XOR Y) OR (NOT Z)

X	Y	Z	(A)	(B)	(C)	(D)
0	0	0	1	1	1	1
0	0	1	0	0	0	0
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	0	1	1	1
1	0	1	1	1	0	1
1	1	0	1	1	1	0
1	1	1	1	0	1	1

Question: Which of the columns A-D is the correct output of the Truth Table for: (X XOR Y) OR (NOT Z)

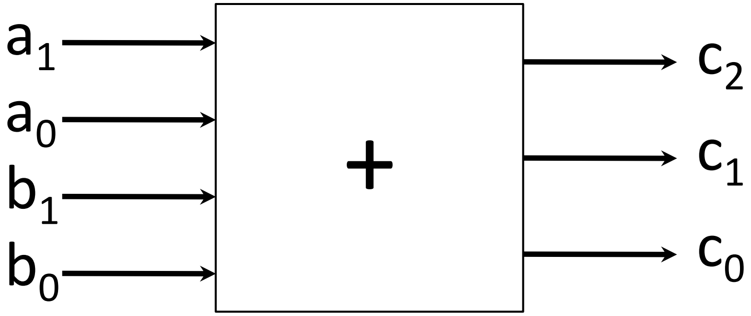
X	Y	Z	(A)	(B)	(C)	(D)
0	0	0	1	1	1	1
0	0	1	0	0	0	0
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	0	1	1	1
1	0	1	1	1	0	1
1	1	0	1	1	1	0
1	1	1	1	0	1	1

More Complex Truth Tables

3-Input Majority

a	b	c	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

2-bit Adder



How many rows?

A		B		C		
a_1	a_0	b_1	b_0	c_2	c_1	c_0
			.			
			.			
			.			

Truth Tables Don’t Scale

- Truth tables are huge
  - Write out EVERY combination of inputs and outputs (thorough, but inefficient)
  - Finding a particular combination of inputs involves scanning a large portion of the table
- Boolean Algebra is a shorter way to represent combinational logic

Boolean Algebra

- Represent inputs and outputs as variables
  - Each variable can only take on the value 0 or 1
- Overbar or ¬ is NOT: “logical complement”
  - e.g. if A is 0, $\overline{A}$ is 1. If A is 1, then ¬A is 0
- Plus (+) is 2-input OR: “logical sum”
- Product (·) is 2-input AND: “logical product”
  - Sometimes omitted
- All other gates and logical expressions can be built from combinations of these

$\overline{\mathbf{AB}} + \mathbf{A}\overline{\mathbf{B}}$ == (NOT(**A** AND **B**)) OR (**A** AND (NOT **B**))

Laws of Boolean Algebra

These laws allow us to simplify Boolean expressions:

$x \cdot \overline{x} = 0$	$x + \overline{x} = 1$	complementarity
$x \cdot 0 = 0$	$x + 1 = 1$	laws of 0's and 1's
$x \cdot 1 = x$	$x + 0 = x$	identities
$x \cdot x = x$	$x + x = x$	idempotent law
$x \cdot y = y \cdot x$	$x + y = y + x$	commutativity
$(xy)z = x(yz)$	$(x + y) + z = x + (y + z)$	associativity
$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$	distribution
$xy + x = x$	$(x + y)x = x$	uniting theorem
$\overline{x}y + x = x + y$	$(\overline{x} + y)x = xy$	uniting theorem v.2
$\overline{x \cdot y} = \overline{x} + \overline{y}$	$\overline{x + y} = \overline{x} \cdot \overline{y}$	DeMorgan's Law

Converting Truth Table to Boolean Expression

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

- Read off of table
  - For 1, write variable name
  - For 0, write complement of variable
- *Sum of Products (SoP)*
  - Take rows with 1’s in output column, sum products of inputs
  - $c = \overline{a}\,b + a\,\overline{b}$
- *Product of Sums (PoS)*
  - Take rows with 0’s in output column, product the sum of the *complements of the inputs*
  - $c = (a + b) \cdot (\overline{a} + \overline{b})$

We can show that these are equivalent!

Simplifying Boolean Expressions

- **Logic Delay:** Everything we are dealing with is just an abstraction of transistors and wires
  - Inputs propagating to the outputs are voltage signals passing through transistor networks
  - There is always some *delay* before a CL's output updates to reflect the inputs
  - Critical Path is longest delay from any input to output. Could be represented as "n gate delays"
- Simpler Boolean expressions ↔ smaller transistor networks ↔ smaller circuit delays ↔ faster hardware

Simplifying Boolean Expressions: Example

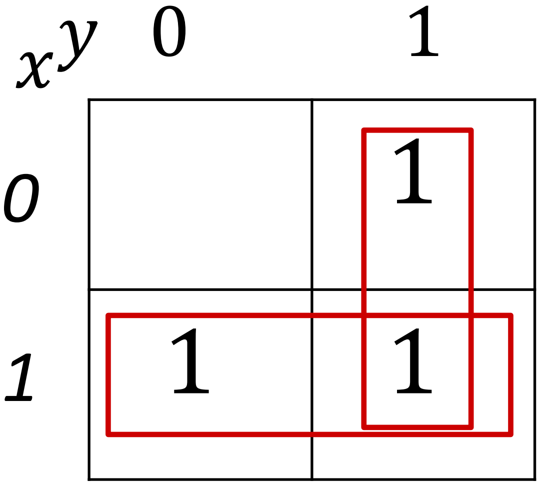
$$y \quad = ab + a + c$$

Karnaugh Maps

- Used to simplify Boolean expressions of 2-4 variables
- Table composed of squares each representing a unique combination of all variable (1 if true, else blank)
- Two variable Map:

x y	0	1
0	$\bar{x}\bar{y}$	$\bar{x}y$
1	$x\bar{y}$	xy

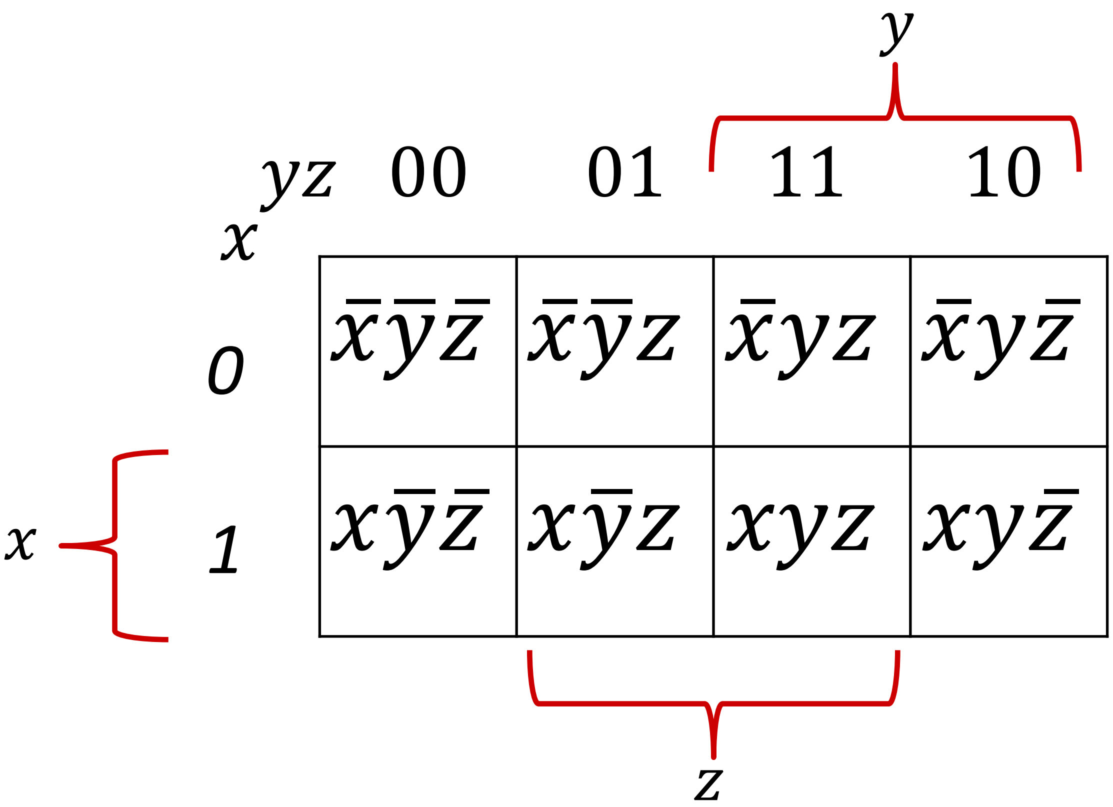
Example: Boolean Expression?



x y	0	1
0		1
1	1	1

$x + y$

Three Variable Karnaugh Maps

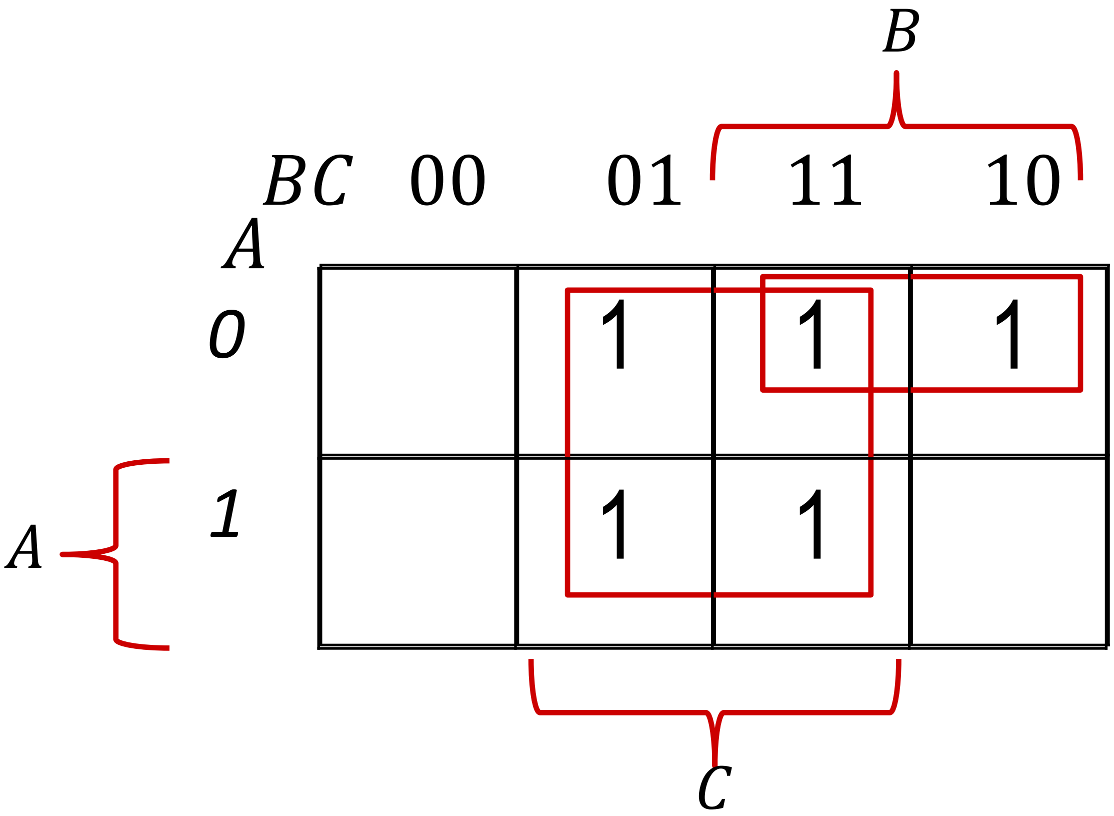


$x \backslash yz$	00	01	11	10
0	$\bar{x}\bar{y}\bar{z}$	$\bar{x}\bar{y}z$	$\bar{x}yz$	$\bar{x}y\bar{z}$
1	$x\bar{y}\bar{z}$	$x\bar{y}z$	xyz	$xy\bar{z}$

Question: Simplify $\bar{A}C + \bar{A}B + A\bar{B}C + BC$

Example: Simplify 3-Variable Expression

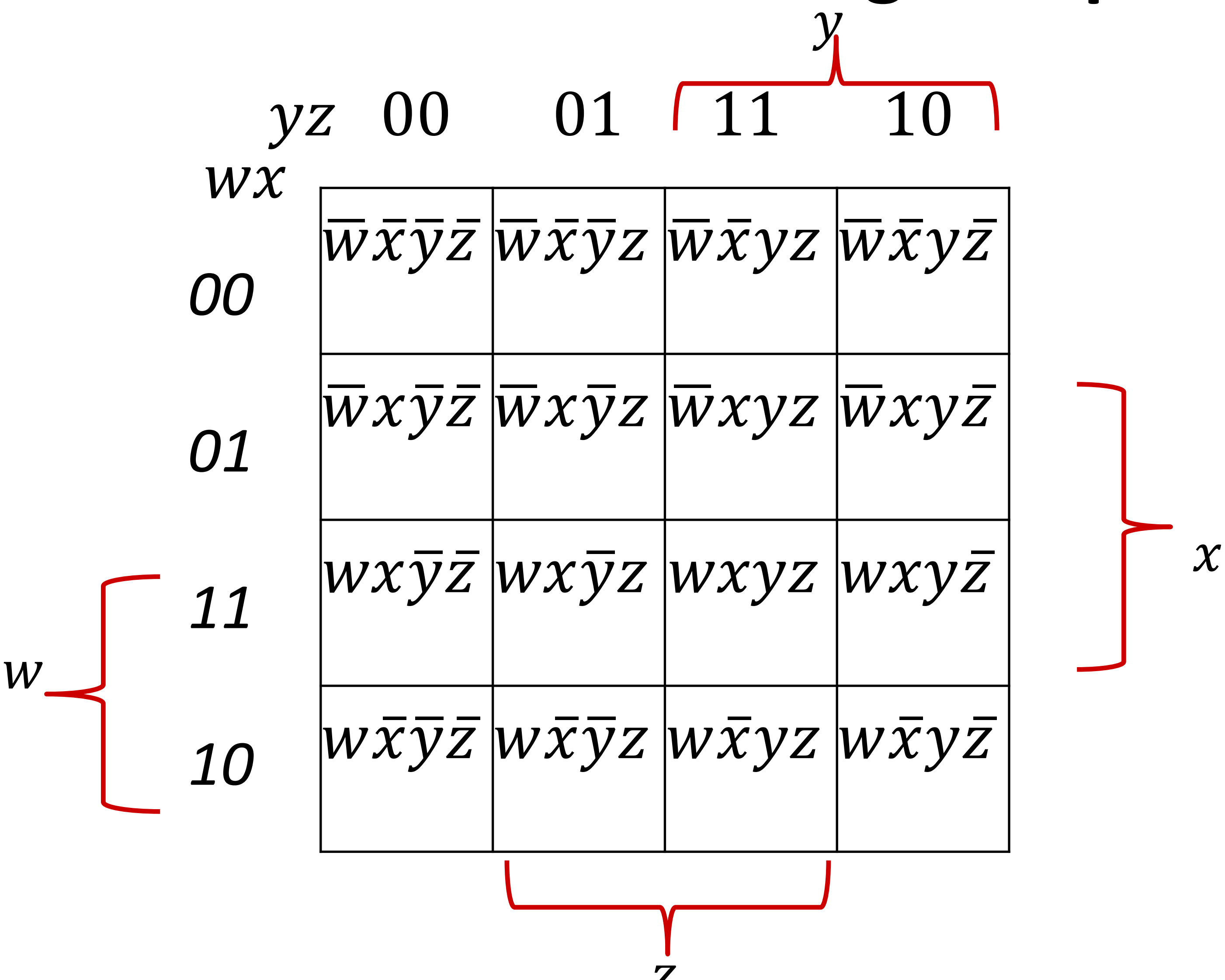
Question: Simplify $\bar{A}C + \bar{A}B + A\bar{B}C + BC$



A \ BC	00	01	11	10
0		1	1	1
1		1	1	

Answer: $C + \bar{A}B$

Four Variable Karnaugh Maps

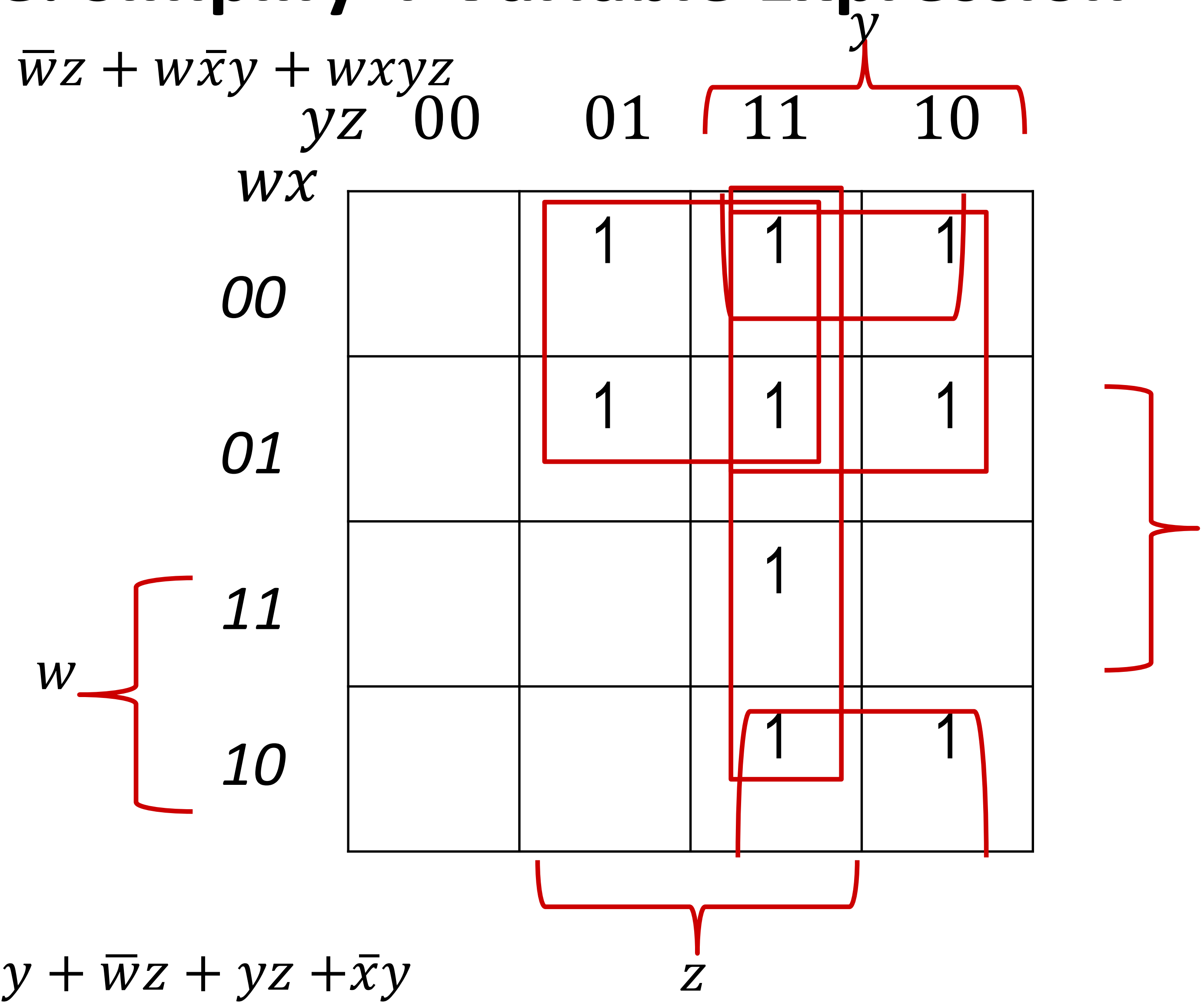


$wx \backslash yz$	00	01	11	10
00	$\bar{w}\bar{x}\bar{y}\bar{z}$	$\bar{w}\bar{x}\bar{y}z$	$\bar{w}\bar{x}yz$	$\bar{w}\bar{x}y\bar{z}$
01	$\bar{w}x\bar{y}\bar{z}$	$\bar{w}x\bar{y}z$	$\bar{w}xyz$	$\bar{w}xy\bar{z}$
11	$wx\bar{y}\bar{z}$	$wx\bar{y}z$	$wxyz$	$wxy\bar{z}$
10	$w\bar{x}\bar{y}\bar{z}$	$w\bar{x}\bar{y}z$	$w\bar{x}yz$	$w\bar{x}y\bar{z}$

Question: Simplify $\bar{w}y + \bar{w}z + w\bar{x}y + wxyz$

Example: Simplify 4-Variable Expression

Simplify $\bar{w}y + \bar{w}z + w\bar{x}y + wxyz$



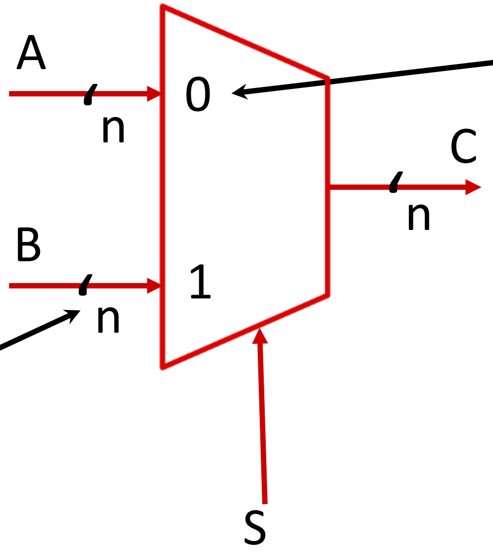
$wx \backslash yz$	00	01	11	10
00		1	1	1
01		1	1	1
11			1	
10			1	1

Solution: $\bar{w}y + \bar{w}z + yz + \bar{x}y$

Useful Combinational Circuits

Data Multiplexor (MUX)

- Multiplexor (“MUX”) is a *selector*
  - – Place one of multiple inputs onto output (N-to-1)
- Shown below is an n-bit 2-to-1 MUX
  - – Input S selects between two inputs of n bits each

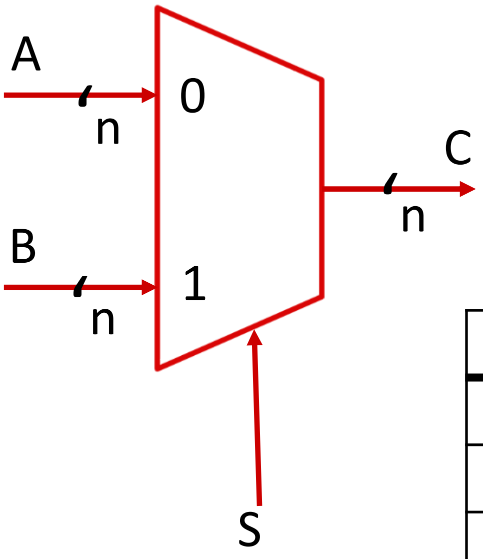


This input is passed to output if selector bits match shown value

Represents that this input has n bits

Implementing a 1-bit 2-to-1 MUX

- **Schematic:**



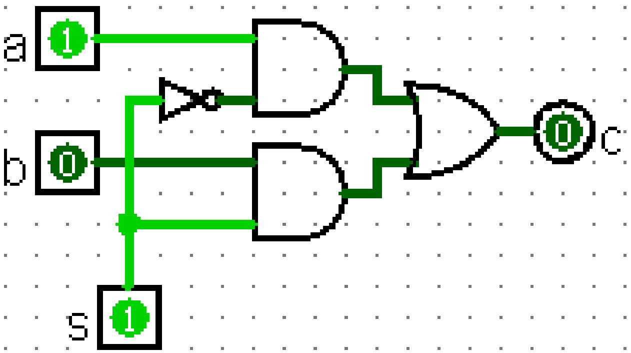
- **Truth Table:**

s	a	b	c
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

- **Boolean Algebra:**

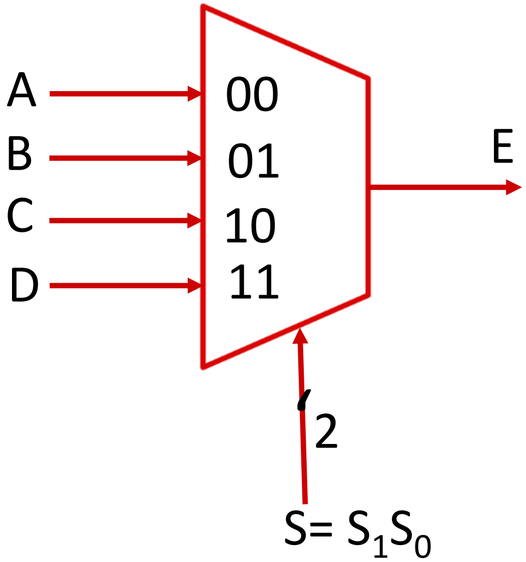
$$\begin{aligned} c &= \bar{s}a\bar{b} + \bar{s}ab + s\bar{a}b + sab \\ &= \bar{s}(a\bar{b} + ab) + s(\bar{a}b + ab) \\ &= \bar{s}(a(\bar{b} + b)) + s((\bar{a} + a)b) \\ &= \bar{s}(a(1) + s((1)b) \\ &\boxed{= \bar{s}a + sb} \end{aligned}$$

- **Circuit Diagram:**



1-bit 4-to-1 MUX

- **Schematic:**

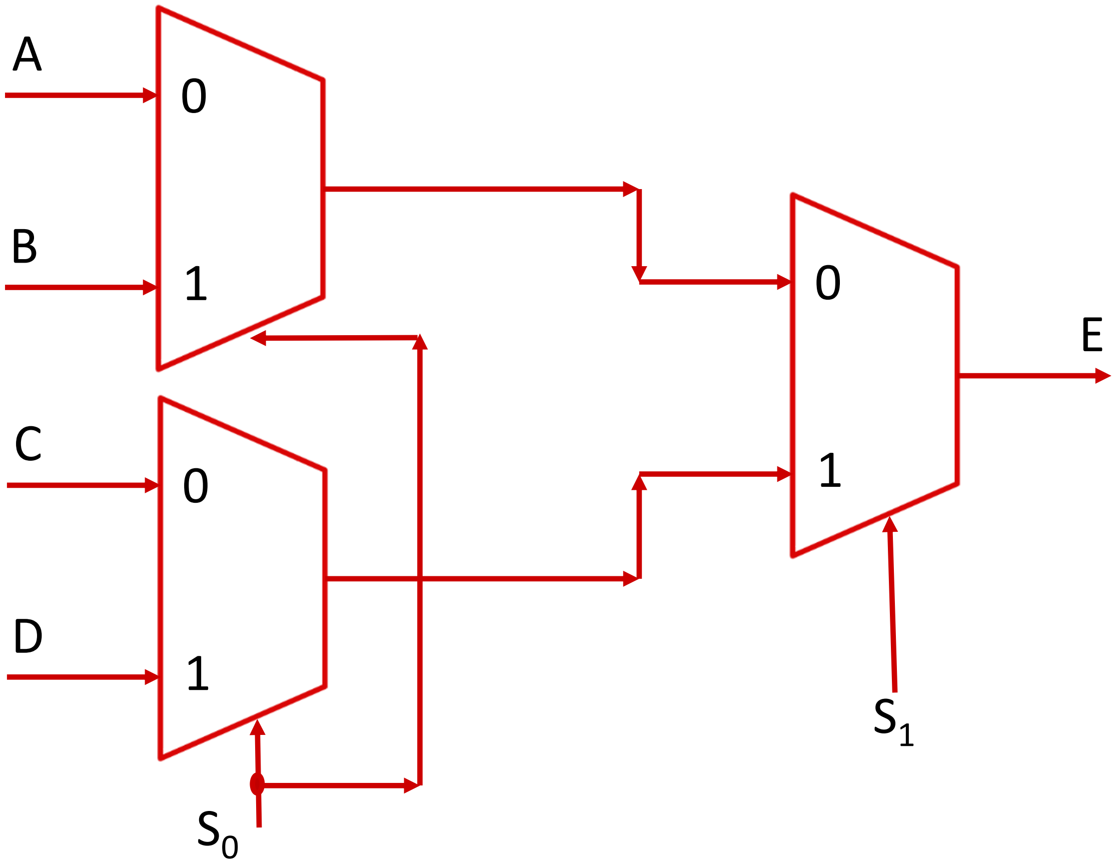


- **Truth Table:** How many rows? 2^6
- **Boolean Expression:**

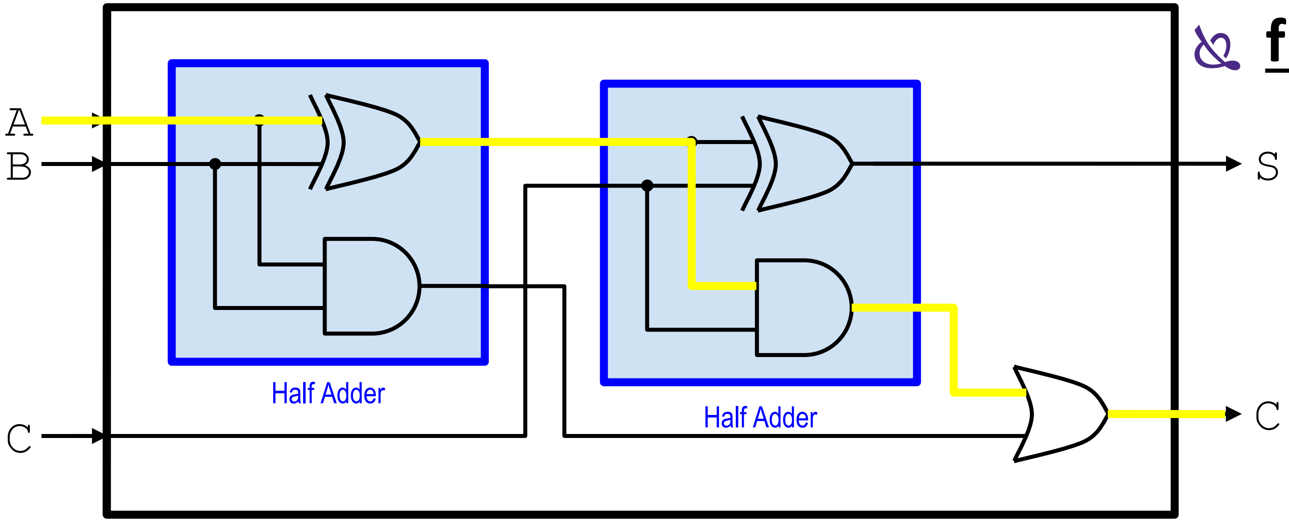
$$E = \overline{S_1S_0}A + \overline{S_1}S_0B + S_1\overline{S_0}C + S_1S_0D$$

Another Design for 4-to-1 MUX

- Can we leverage what we’ve previously built?
  - – Alternative hierarchical approach:



Full Adder



Full Adder

- **function table:**
  - basically a truth table

A B C	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

3-bit Addition!

Q. What's the propagation delay?

- **3 gate delays (highlighted)**

Q. What does the circuit accomplish?

- **Algebra:** S = A ^ B ^;CC = (A & B) | (C & (A ^ B))

; $C = AB + C(A \oplus B)$

$S = A \oplus B \oplus C$