CS 126 Lecture A3:
Boolean Logic

Outline

- **Introduction**
- Logic gates
- Boolean algebra
- Implementing gates with switching devices
- Common combinational devices
- Conclusions

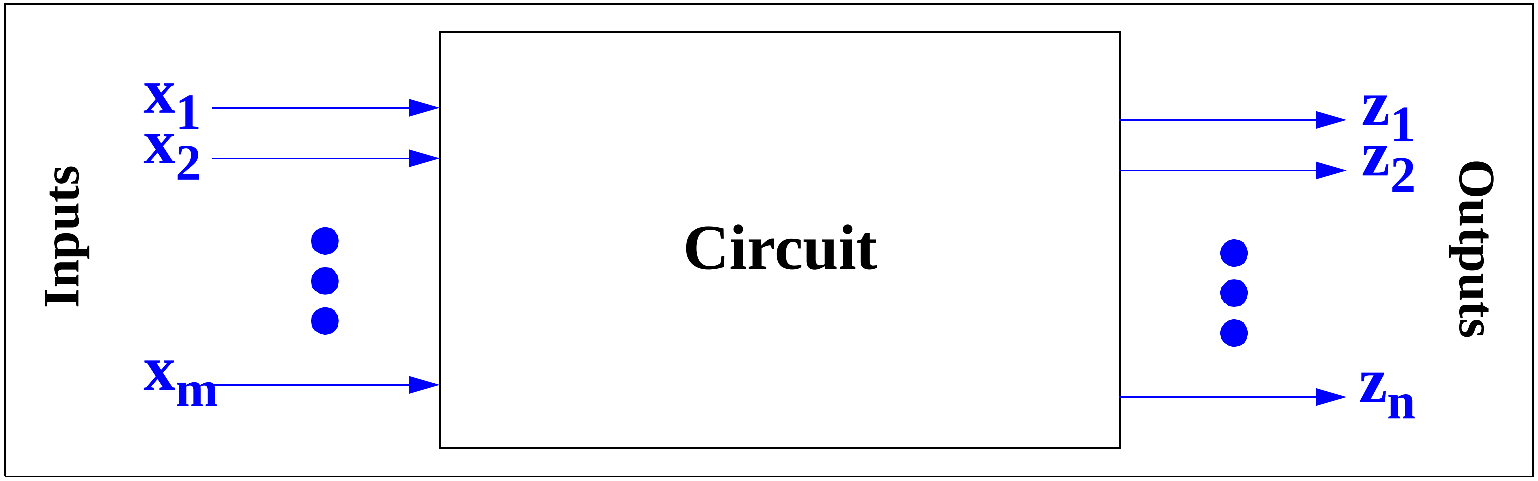
Where We Are At

- We have learned the abstract **interface** presented by a machine: the instruction set architecture
- What we will learn: the **implementation** behind the interface:
  - Start with switching devices (such as transistors)
  - Build logic gates with transistors
  - Build combinational circuit (memory-less) devices using gates
  - Next lecture: build sequential circuit (memory) devices
  - The one after: glue these devices into a computer

Digital Systems

- ... however, the application of digital logic extends way beyond just computers.
- Today, digital systems are replacing all kinds of analog systems in life (data processing, control systems, communications, measurement, ...)
- What is a digital system?
  - Digital: quantities or signals only assume discrete values
  - Analog: quantities or signals can vary continuously
- Why digital systems?
  - Greater accuracy and reliability

Digital Logic Circuits

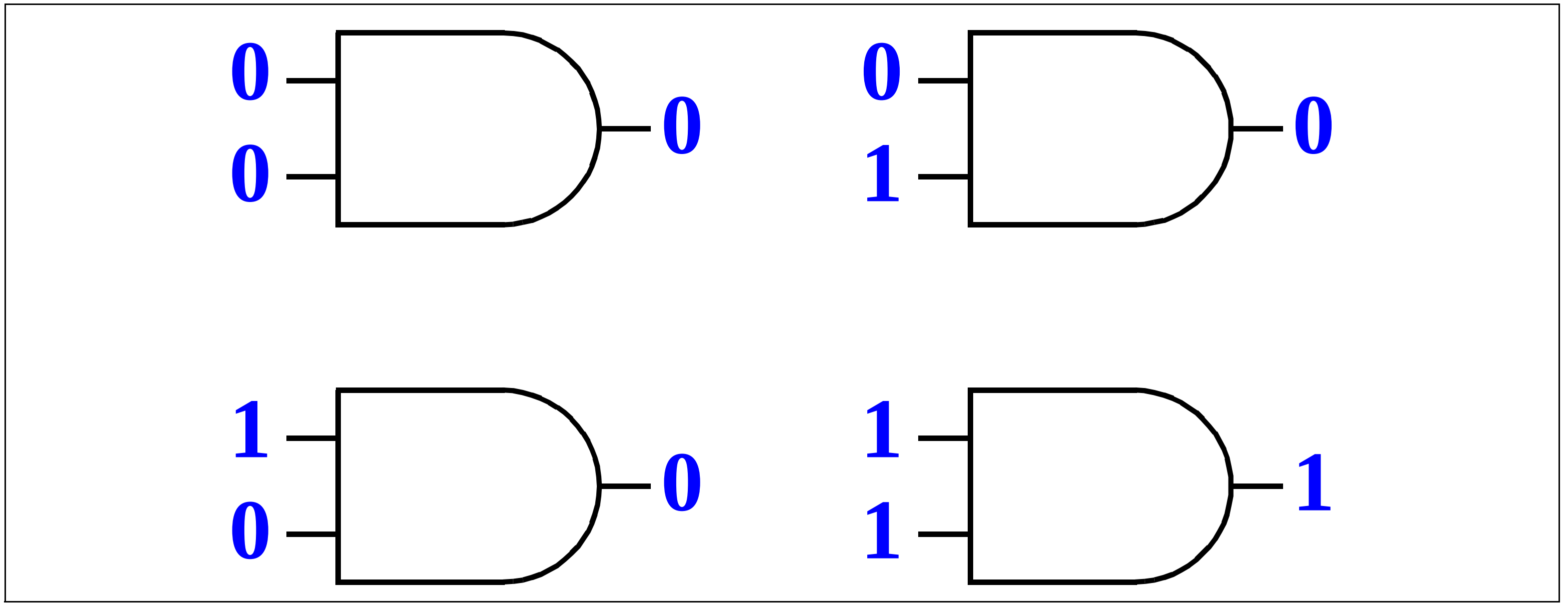


- The heart of a digital system is usually a digital logic circuit

Outline

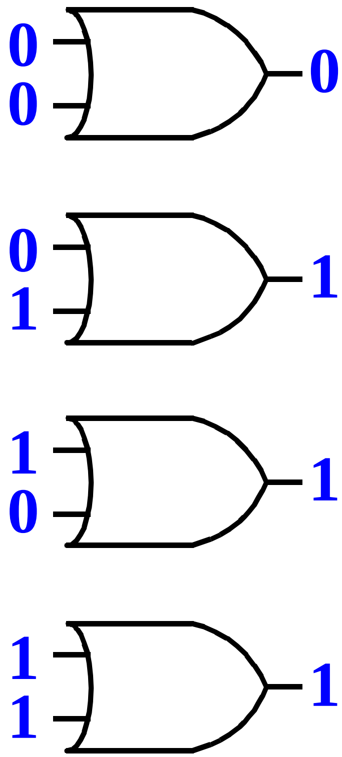
- ~~Introduction~~
- **Logic gates**
- Boolean algebra
- Implementing gates with switching devices
- Common combinational devices
- Conclusions

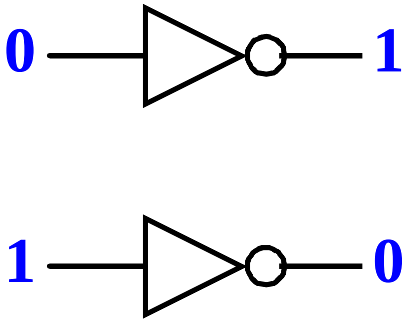
An AND-Gate



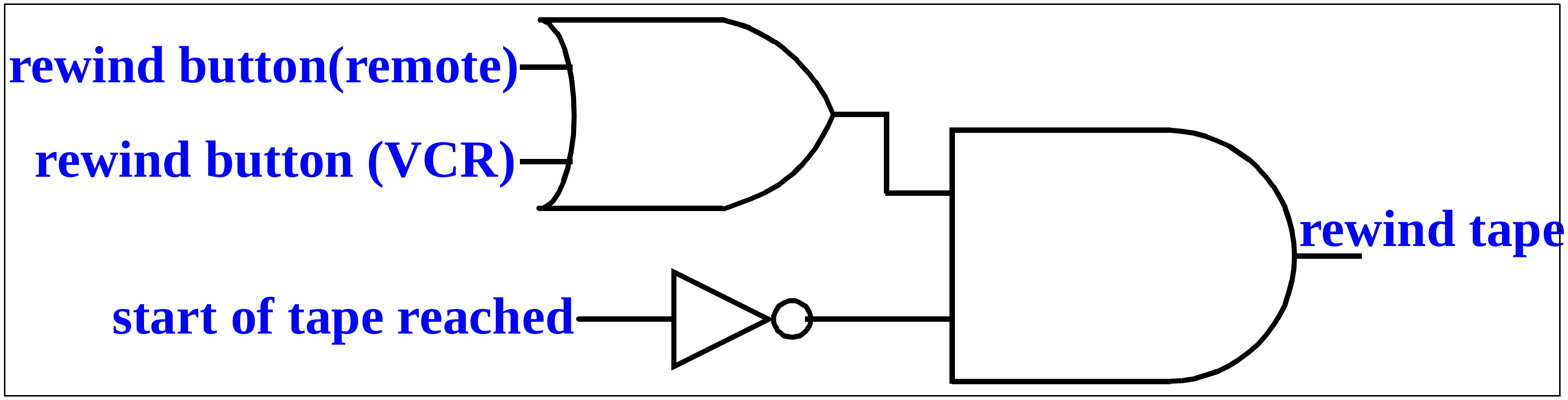
- A smallest useful circuit is a logic gate
- We will connect these small gates into larger circuits

An OR-Gate and a NOT-Gate



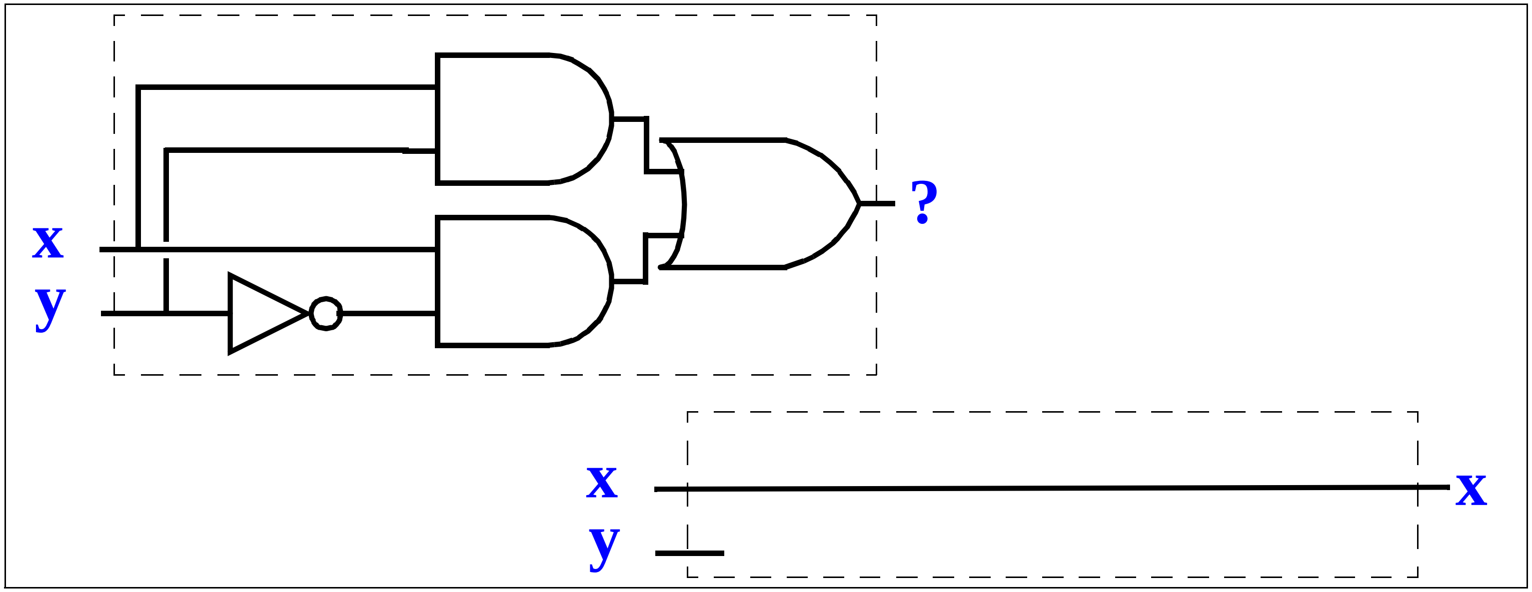


Building Circuits Using Gates



- Can implement **<u>any</u>** circuit using **<u>only</u>** AND, OR, and NOT gates
- But things get complicated when we have lots of inputs and outputs...

Problems



- Many different ways of implementing a circuit (the two above circuits turn out to be the same!)
- How do we find the best implementation? Need better formalism
- Also need more compact representation
- This leads to the study of boolean algebra

Outline

- ~~Introduction~~
- ~~Logic gates~~
- **Boolean algebra**
- Implementing gates with switching devices
- Common combinational devices
- Conclusions

Boolean Algebra

- History
  - Developed in 1847 by Boole to solve mathematic logic problems
  - Shannon first applied it to digital logic circuits in 1939
- Basics
  - **Boolean variables**: variables whose values can be 0 or 1
  - **Boolean functions**: functions whose inputs and outputs are boolean variables
- Relationship with logic circuits
  - Boolean variables correspond to signals
  - Boolean functions correspond to circuits

Defining a Boolean Function with a Truth Table

x	0	0	1	1
y	0	1	0	1
AND(x,y)	0	0	0	1

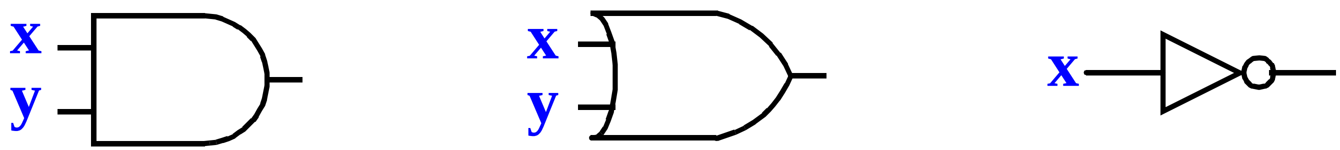
- A systematic way of specifying a function value for **all** possible combination of input values
- A function that takes 2 inputs has 2x2 columns
- A function that takes n inputs has 2^n columns
- This particular example is the AND-function

OR and NOT Truth Tables

x	0	0	1	1
y	0	1	0	1
OR(x,y)	0	1	1	1

x	0	1
NOT(x)	1	0

Defining a General Boolean Function Using Three Basic Boolean Functions



AND(x,y)=xy=x*y **OR(x,y)=x+y** **NOT(x)=x'**

- The three basic functions have short-hand notations
- Can compose the three basic boolean functions to form arbitrary boolean functions [such as **g(x,y)=xy+z'**]

Two Ways of Defining a Boolean Function

x	0	0	1	1
y	0	1	0	1
XOR(x,y)=x^y	0	1	1	0

XOR(x,y) = x^y = x'y + xy'

- We have learned that any function can be defined in these two ways: truth table and composition of basic functions
- Why do we need all these different representations?
  - Some are easier than others to begin with to design a circuit
  - Usually start with truth table (or variants of it)
  - Derive a boolean expression from it (perhaps including simplification)
  - Straightforward transformation from boolean expression to circuit

More Examples of Boolean Functions

Sixteen different functions

0	0	1	1	x	
0	1	0	1	y	
0	0	0	0	constant 0	
0	0	0	1	AND (xy)	[decode 11 = 3]
0	0	1	0		[decode 10 = 2]
0	0	1	1	x	
0	1	0	0		[decode 01 = 1]
0	1	0	1	y	
0	1	1	0	XOR (x^y)	
0	1	1	1	OR (x+y)	
1	0	0	0	NOR ("not or")	[decode 00 = 0]
1	0	0	1	== ("not xor")	
1	0	1	0	NOT y (y')	
1	0	1	1		
1	1	0	0	NOT x (x')	
1	1	0	1		
1	1	1	0	NAND ("not and")	
1	1	1	1	constant 1	

Gluing the truth tables of all functions of two variables into one table

For n variables, there are a total of 2^{2^n} functions!

So How to Translate a Truth Table to a Boolean Expression (Sum-of-Products)?

- form AND terms for each 1 in the function
  - use v if it corresponds to v = 1
  - use v' (NOT v) if it corresponds to v = 0
- OR the terms together

Ex: majority function

x:	0	0	0	0	1	1	1	1
y:	0	0	1	1	0	0	1	1
z:	0	1	0	1	0	1	0	1
m:	0	0	0	1	0	1	1	1

m = x'yz + xy'z + xyz' + xyz

Another Example

Example: odd parity function

x:	0	0	0	0	1	1	1	1
y:	0	0	1	1	0	0	1	1
z:	0	1	0	1	0	1	0	1
p:	0	1	1	0	1	0	0	1

$$p = x'y'z + x'yz' + xy'z' + xyz$$

001 010 100 111

Parity Function Construction Demo

```
x: 0 0 0 0 1 1 1 1
y: 0 0 1 1 0 0 1 1
z: 0 1 0 1 0 1 0 1
p: 0 1 1 0 1 0 0 1
```

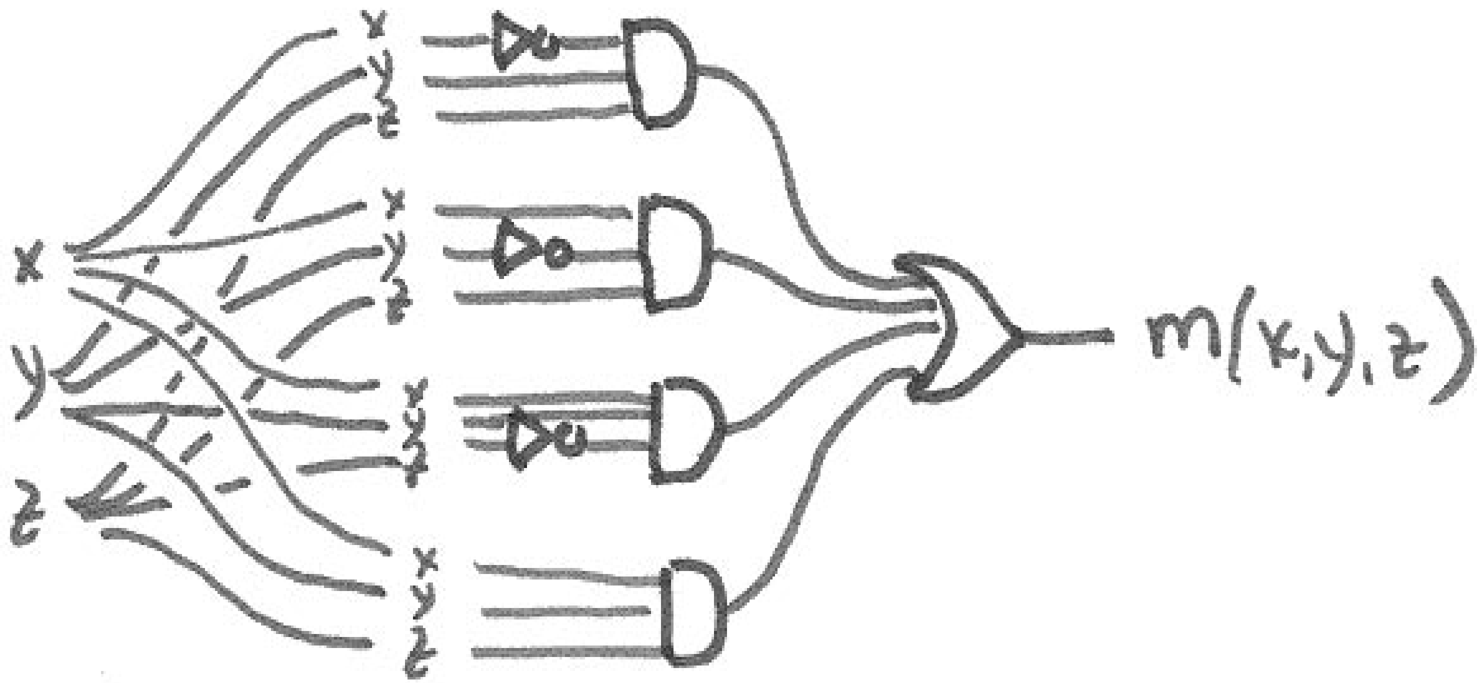
x'y'z   x'y z'   x y'z'+ x y z

Transform a Boolean Expression into a Boolean Circuit

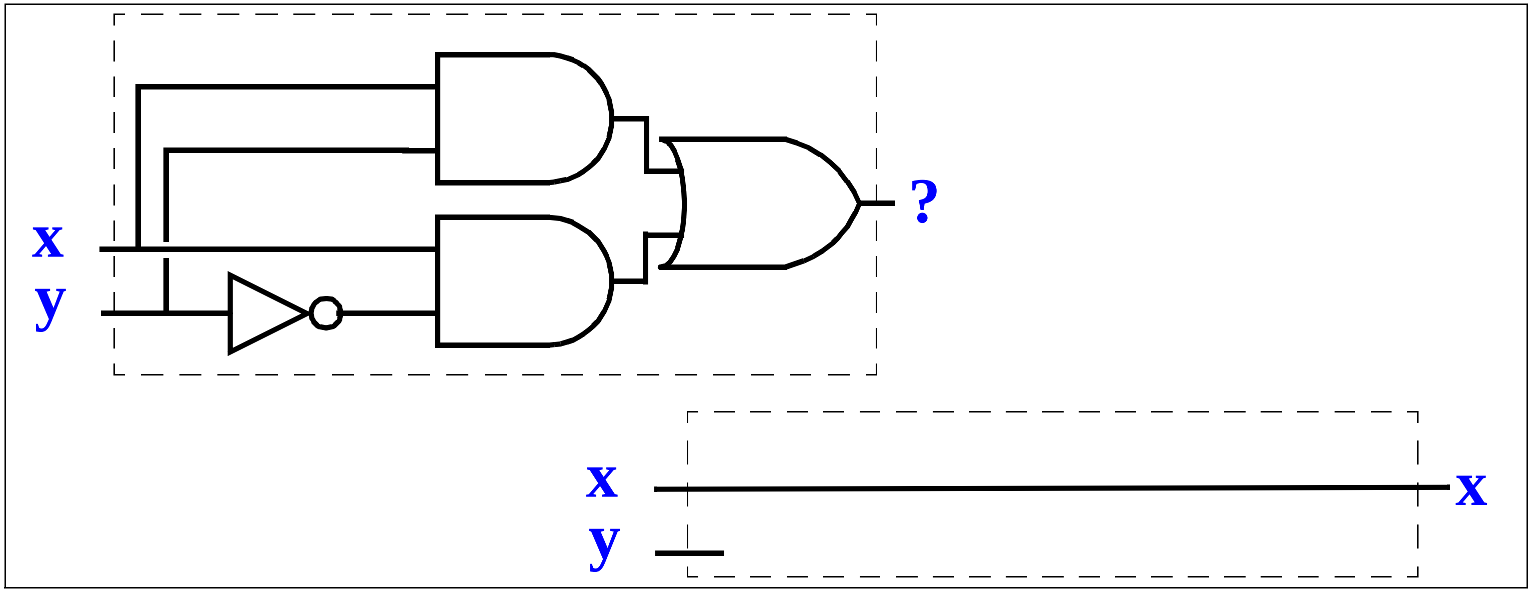
Use sum-of-products form of function

Example: majority

$$m = x'yz + xy'z + xyz' + xyz$$



Simplification Using Boolean Algebra



- Large body of boolean algebra laws can be employed to simplify circuits
- The previous example:
  xy + xy’ = x(y+y’) = x*1 = x
- Much more, but you don’t have to know any of this...

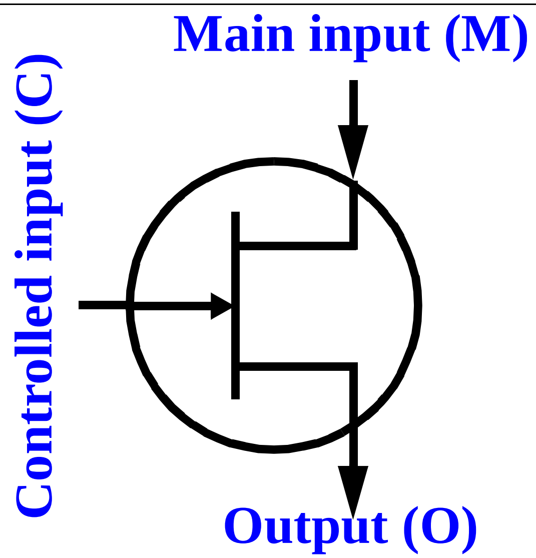
Mini-Summary: How Do We Make a Combinational Circuit

- Represent input signals with input boolean variables, represent output signals with output boolean variables
- Construct truth table based on what we want the circuit to do
- Derive (simplified) boolean expression from the truth table
- Transform boolean expression into a circuit by replacing basic boolean functions with primitive gates

Outline

- ~~Introduction~~
- ~~Logic gates~~
- ~~Boolean algebra~~
- **Implementing gates with switching devices**
- Common combinational devices
- Conclusions

Switching Devices

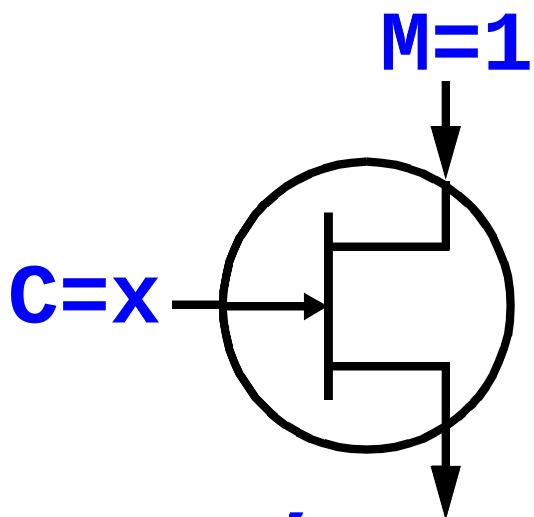


C	0	0	1	1
M	0	1	0	1
O	0	1	0	0

O = M C′

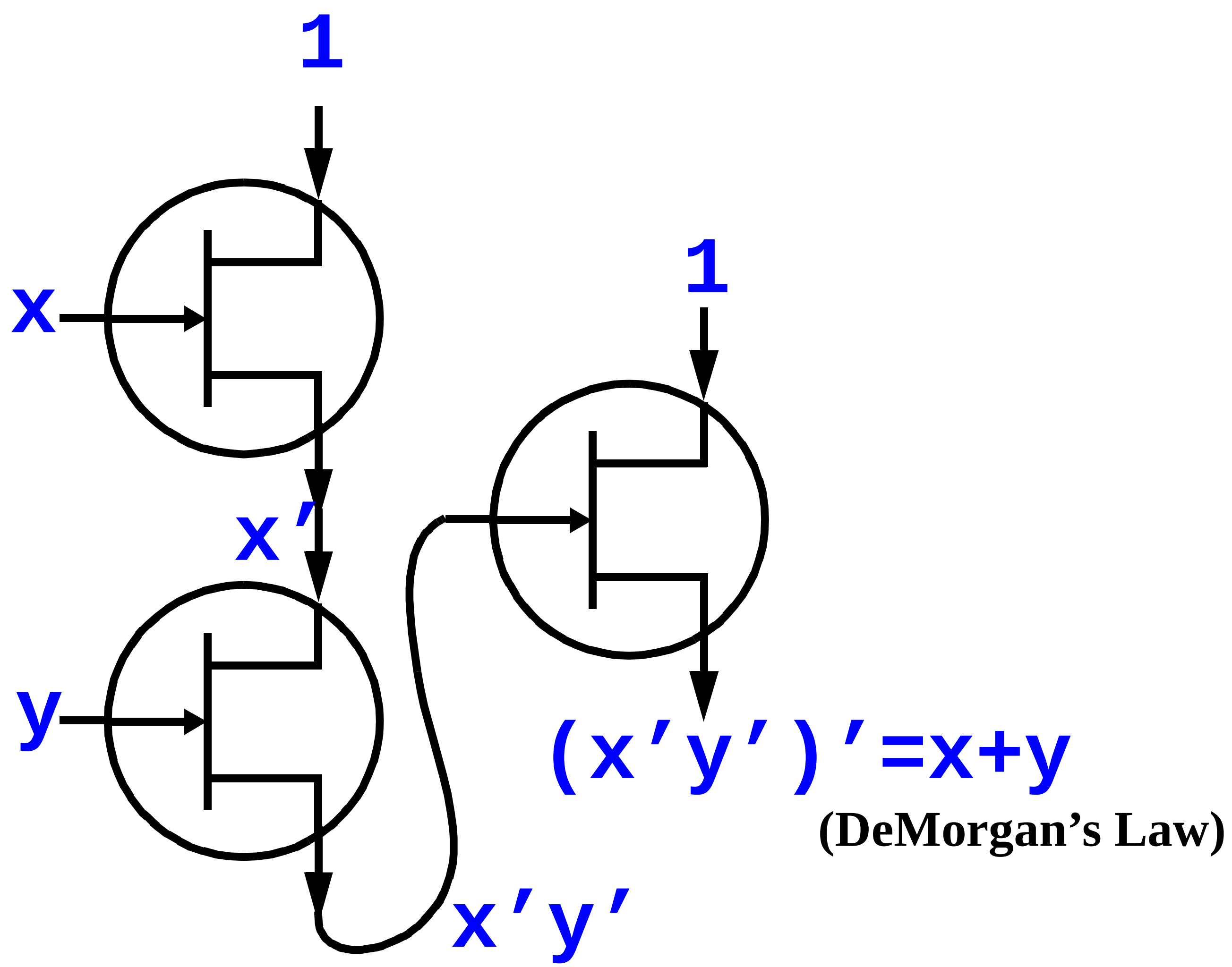
- Any two-state device can be a switching device, examples are relays, diodes, transistors, and magnetic cores
- A transistor example
- Any boolean function can be implemented by wiring together transistors

Make a NOT-gate Using a Transistor

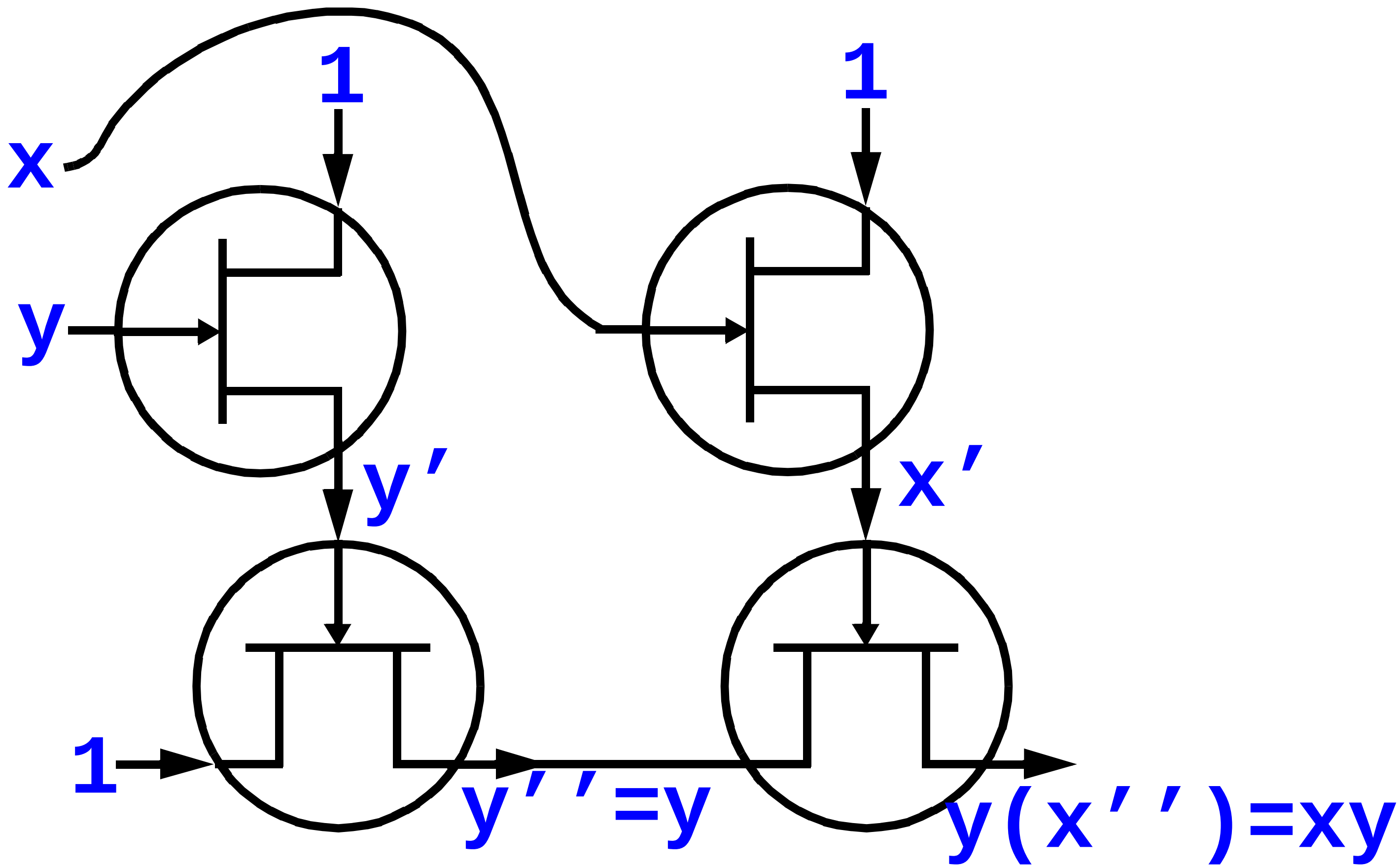


O = MC' = 1*x' = x'

Make an OR-gate Using Transistors



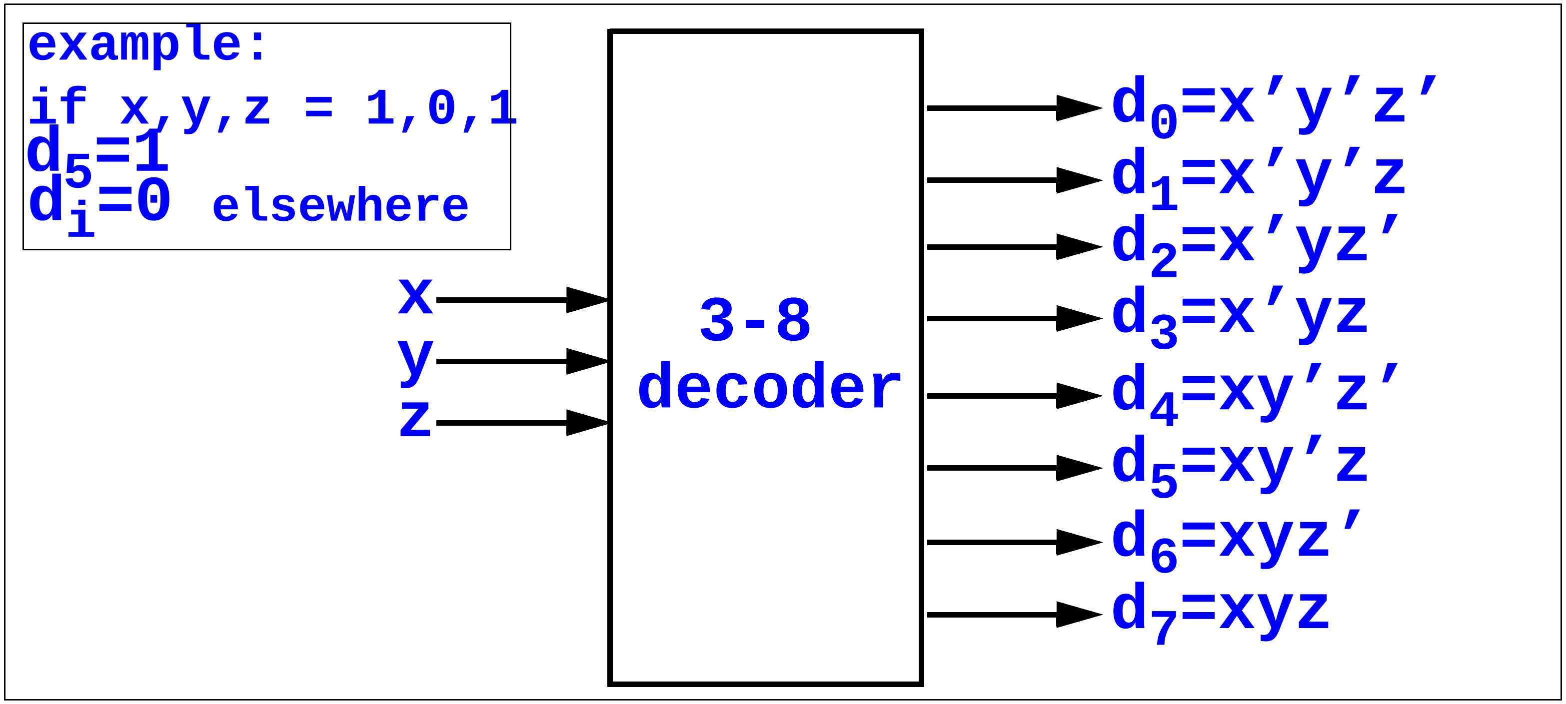
Make an AND-gate Using Transistors



Outline

- ~~Introduction~~
- ~~Logic gates~~
- ~~Boolean algebra~~
- ~~Implementing gates with switching devices~~
- **Common combinational devices**
- Conclusions

Decoder Interface



DECODER

N "inputs"

2^N "outputs"

2^N boolean functions

- Turns on precisely one "output"

  address is encoded in "inputs"

Deriving Decoder Boolean Expressions

x	0	0	0	0	1	1	1	1
y	0	0	1	1	0	0	1	1
z	0	1	0	1	0	1	0	1
d_0	1	0	0	0	0	0	0	0

$$d_0 = x'y'z'$$

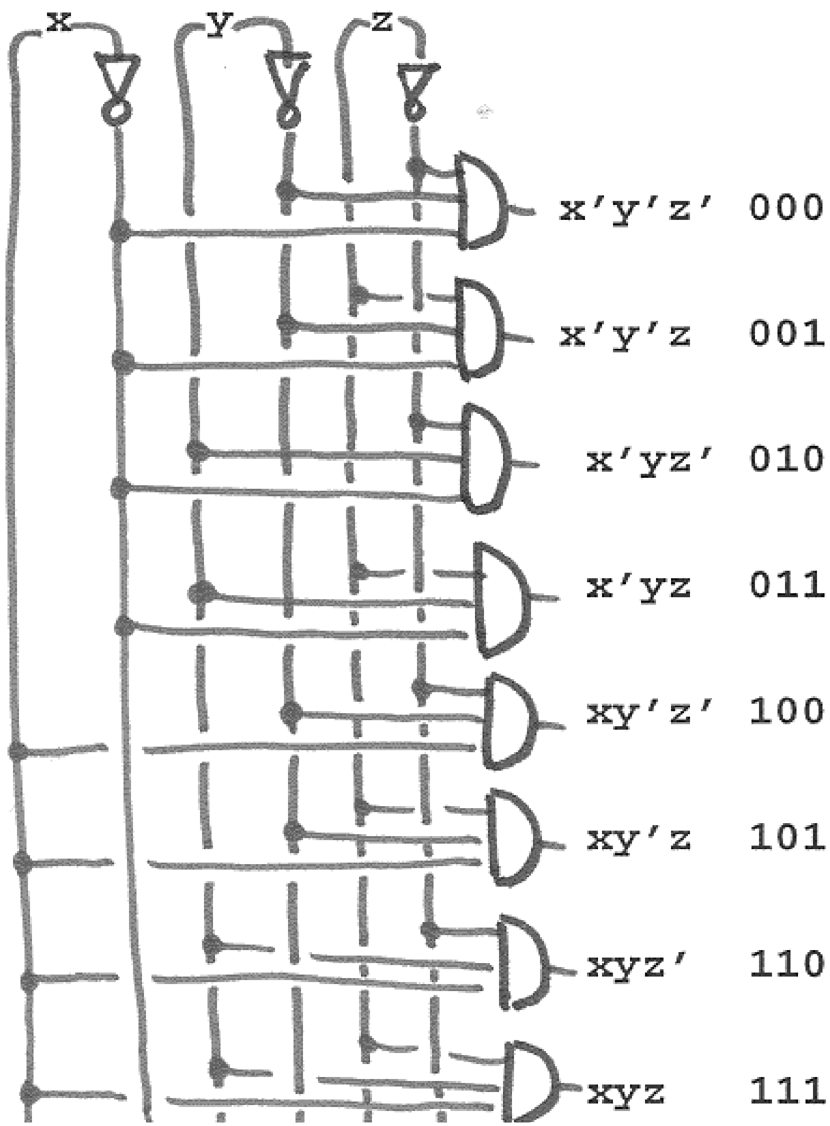
x	0	0	0	0	1	1	1	1
y	0	0	1	1	0	0	1	1
z	0	1	0	1	0	1	0	1
d_1	0	1	0	0	0	0	0	0

$$d_1 = x'y'z$$

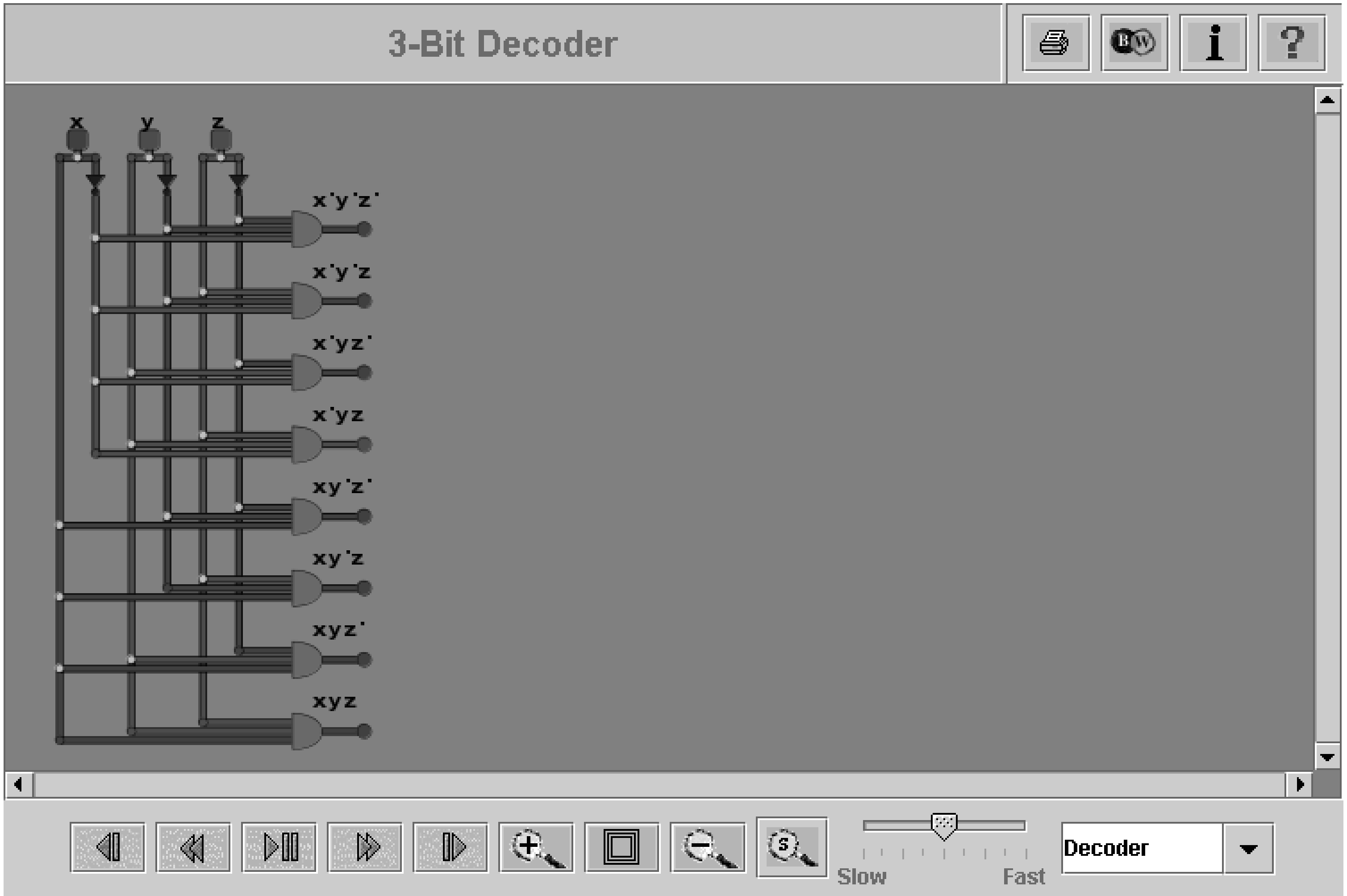
......

- Can bypass truth table when you're comfortable with this

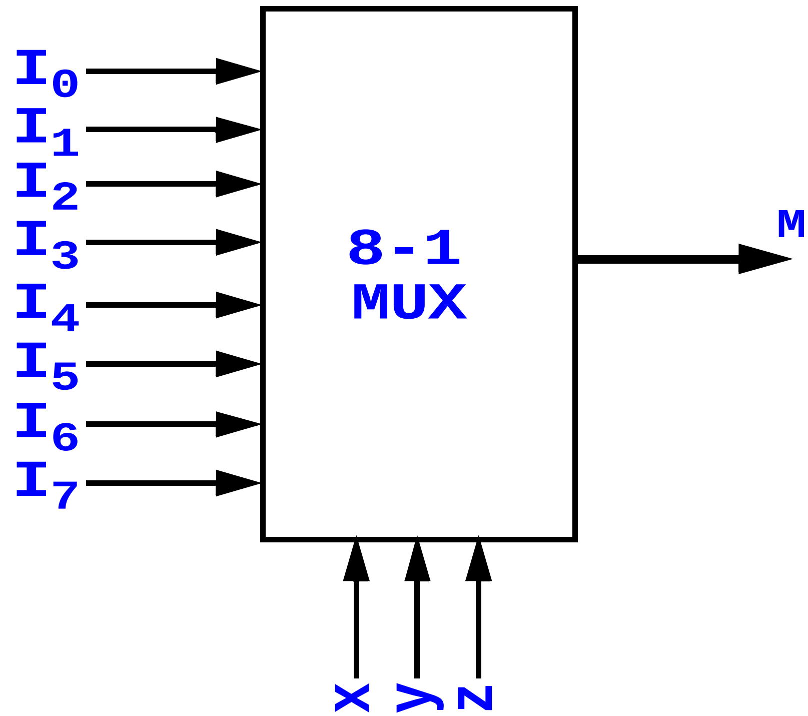
Decoder Implementation



Decoder Demo



Multiplexer Interface



- I_0-I_7 are the "data inputs", x,y,z form the "control" inputs and are interpreted together as one binary number
- One data input is selected by the control and becomes output
- For example, if x,y,z are 1,0,1, then $M=I_5$

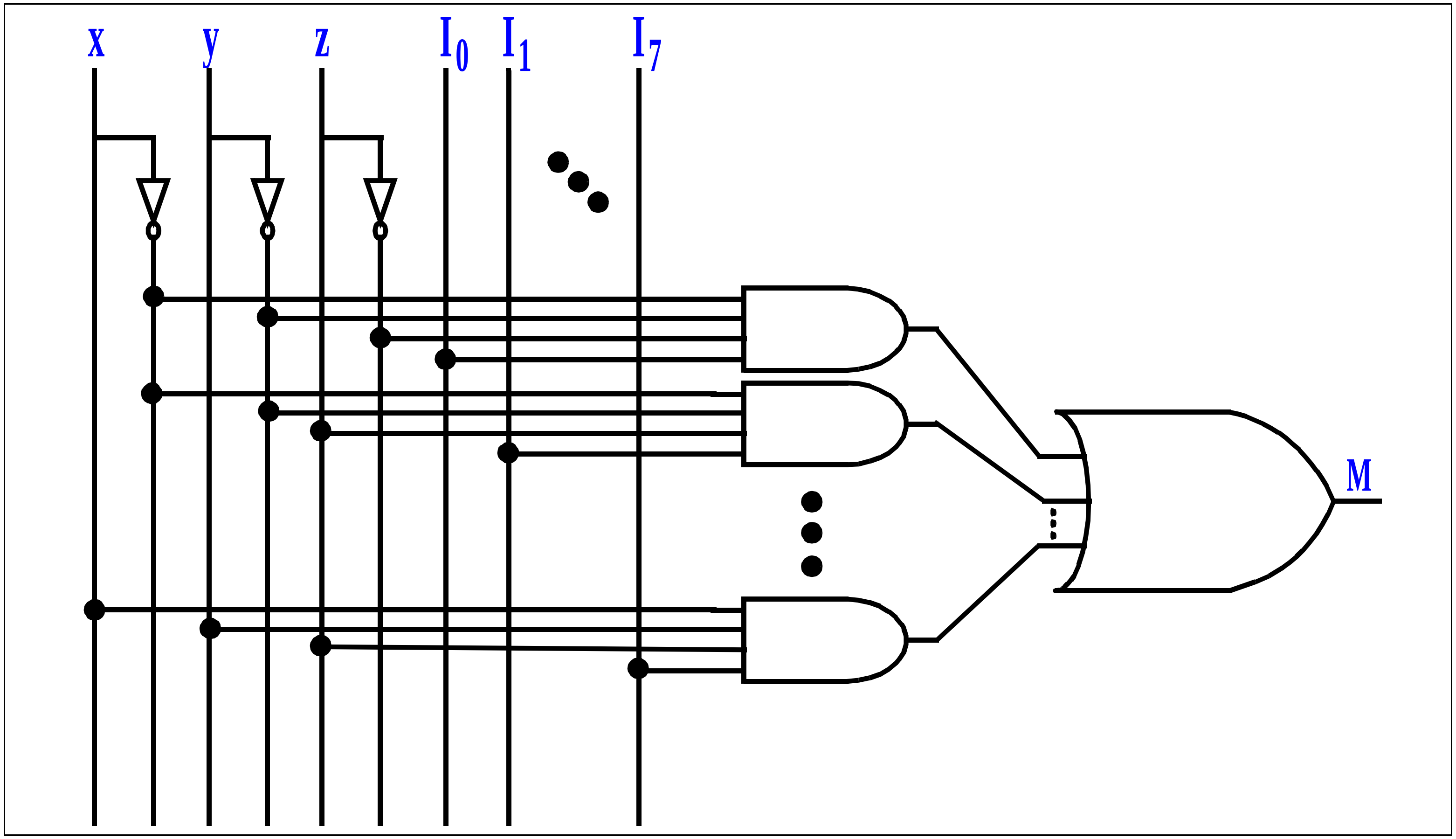
Multiplexer Boolean Expression

x	0	0	0	0	...	1	1
y	0	0	0	0	...	1	1
z	0	0	1	1	...	1	1
I_7	0	0	0	0	...	0	1
...	...	...	...	...	...	...	...
I_1	0	0	0	1	...	0	0
I_0	0	1	0	0	...	0	0
M	0	1	0	1	...	0	1

$$M=x'y'z'I_0 + x'y'zI_1 +...+ xyzI_7$$

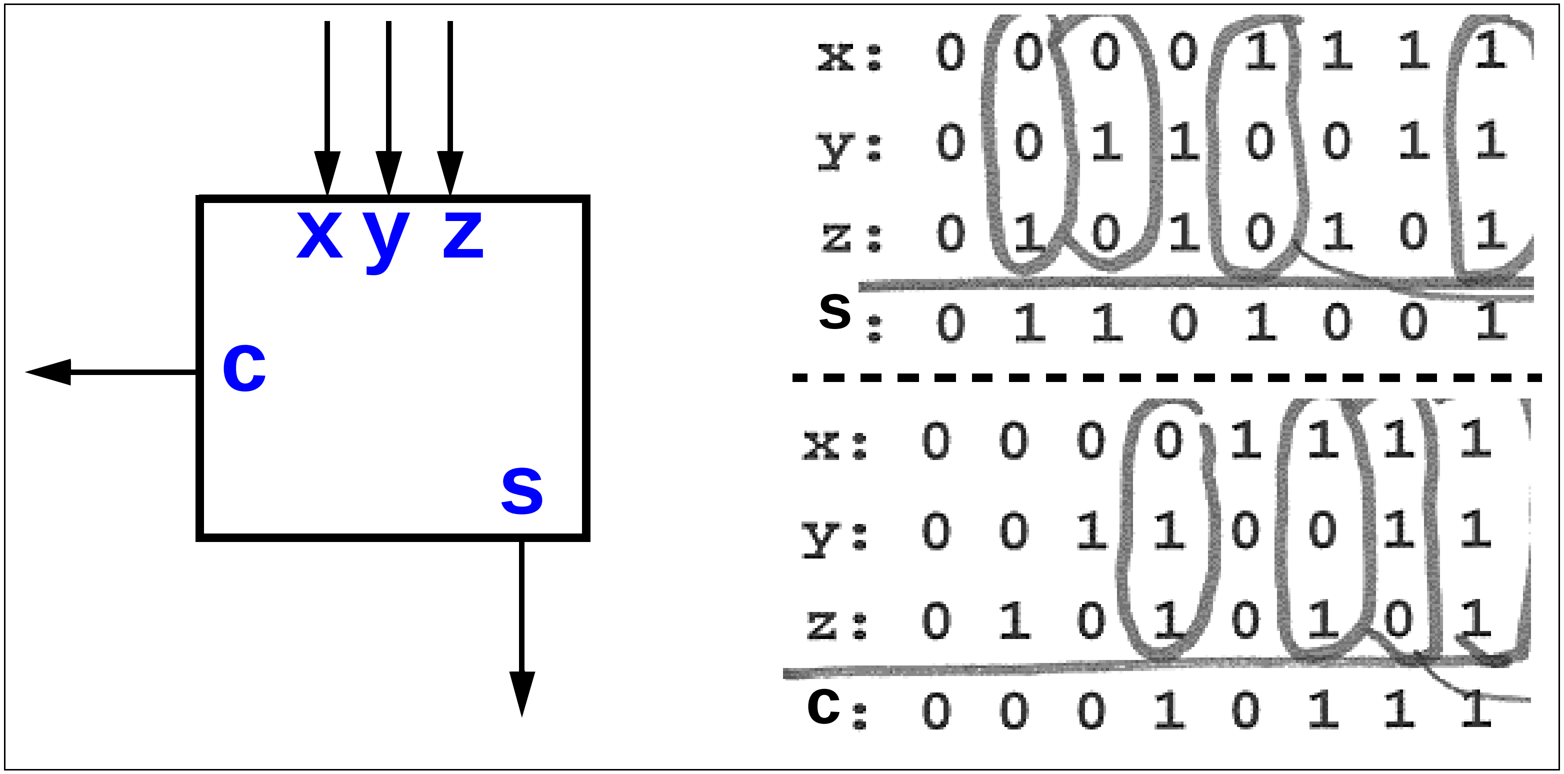
- A lot easier in this case to directly derive the boolean expression instead of starting with a truth table

Multiplexer Implementation



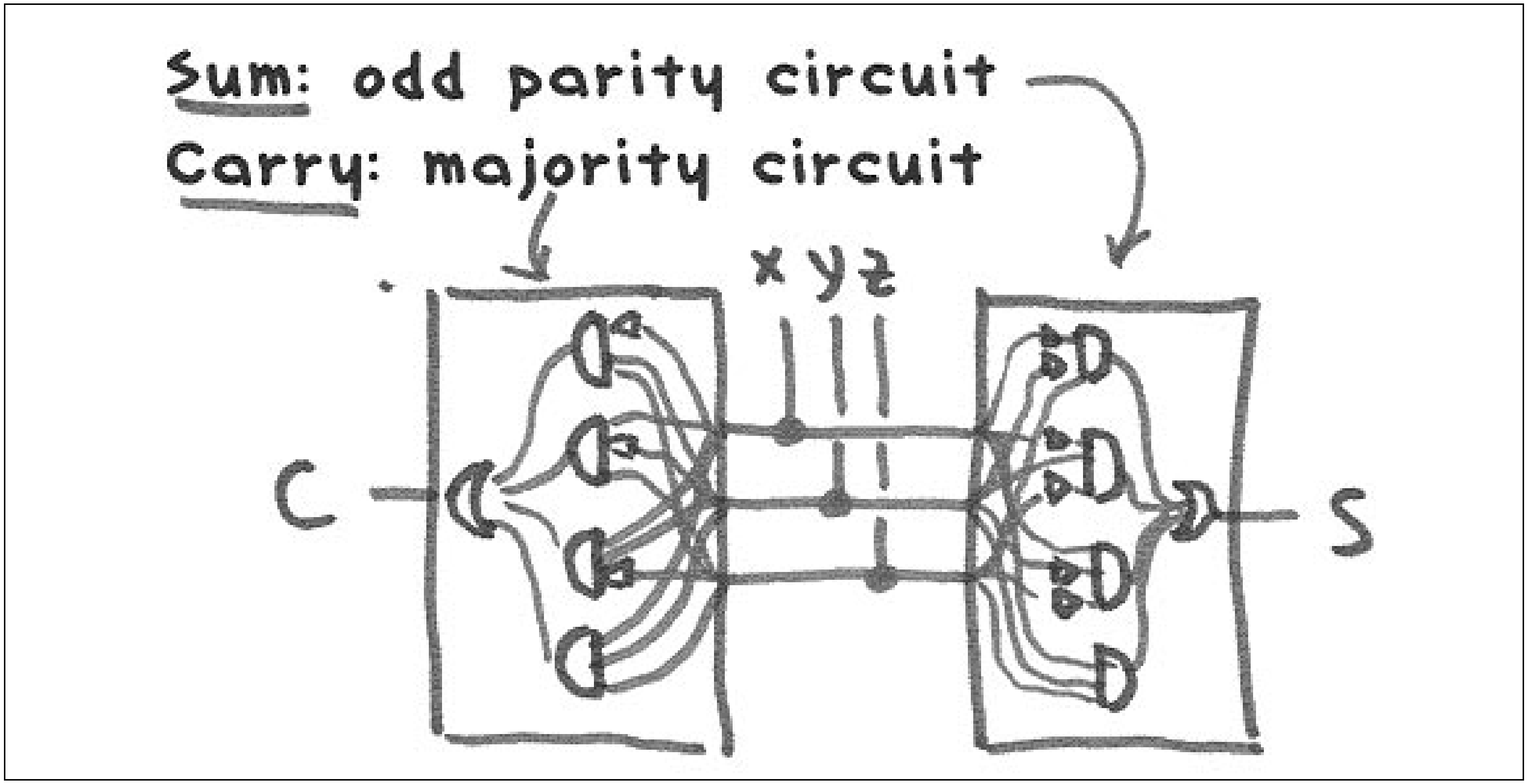
- $M = x'y'z'I_0 + x'y'zI_1 + x'yz'I_2 + x'yzI_3 + xy'z'I_4 + xy'zI_5 + xyz'I_6 + xyzI_7$

An Adder Bit-Slice Interface



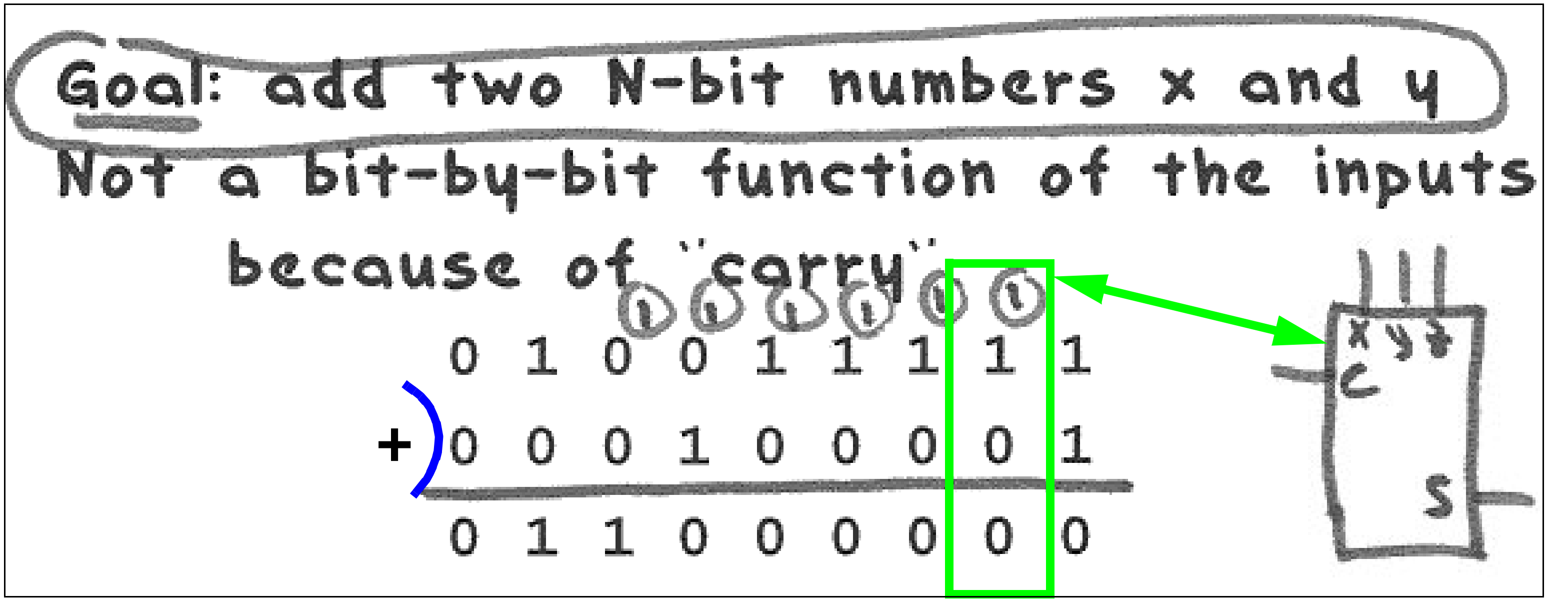
- Add three 1-bit numbers x, y, z
- s is the 1-bit sum
- c is the 1-bit carry

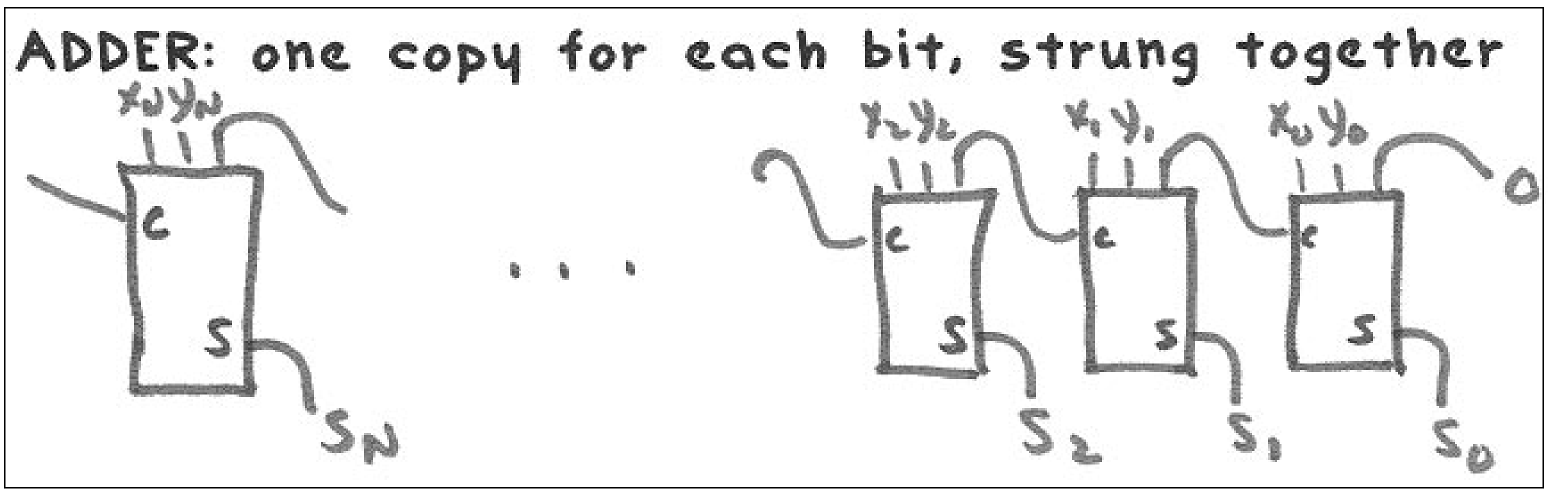
An Adder Bit-Slice Implementation



- See slides 11-16, 11-17, and 11-18 for details of the odd parity circuit and majority circuit

An N-bit Adder Made with Bit-Slices



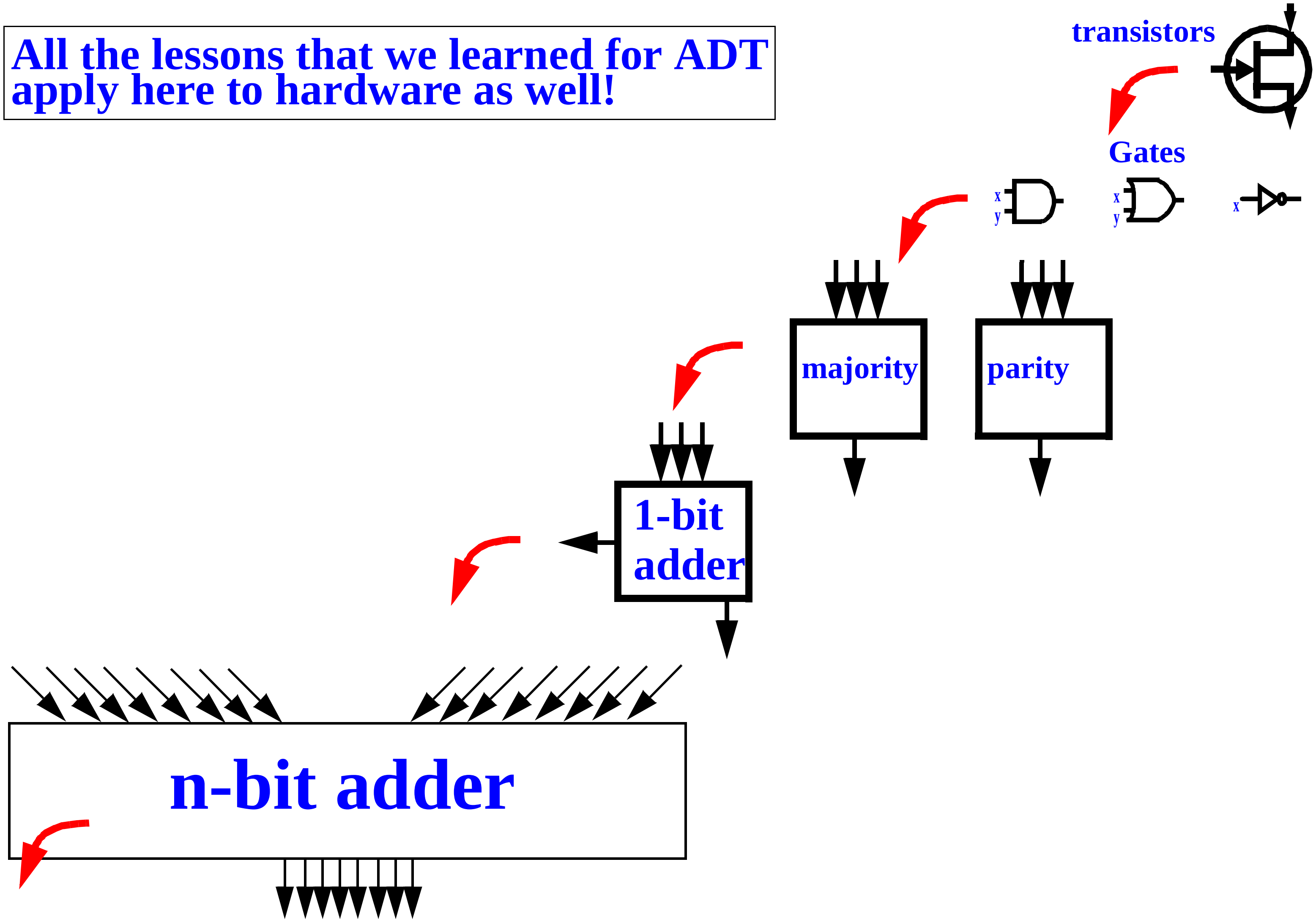


Outline

- ~~Introduction~~
- ~~Logic gates~~
- ~~Boolean algebra~~
- ~~Implementing gates with switching devices~~
- ~~Common combinational devices~~
- **Conclusions**

Abstractions and Encapusulation

All the lessons that we learned for ADT apply here to hardware as well!



Building a Computer Bottom Up

- **Circuit design**: specifying the interconnection of components such as resistors, diodes, and transistors to form logic building blocks
- **Logic design**: determining how to interconnect logic building blocks such as logic gates and flip-flops to form subsystems
- **System design** (or computer architecture): specifying the number, type, and interconnection of subsystems such as memory units, ALUs, and I/O devices

What We Have Learned

- How to build basic gates using transistors
- How to build a combinational circuit
  - Truth table
  - Sum-of-product boolean expression
  - Transform a boolean expression into a circuit of basic gates
- The functionality of some common devices and how they are made
  - Decoder
  - Multiplexer
  - Bit-slice adder
- You're **<u>not</u>** responsible for
  - Boolean algebra laws, or circuit simplification