CS 135: Computer Architecture I

Digital Logic Circuits

Instructor: Prof. Bhagi Narahari
Dept. of Computer Science
Course URL: www.seas.gwu.edu/~bhagiweb/cs135/

Summary of Number Representation

- Binary representation
  - Why binary
  - Unsigned integers
  - 2's complement
  - ASCII
  - Floating point
  - Hex notation
- Other: Bit Vector, Color
  - Position of bit *i* indicates set membership of element *i*
  - *Color expressed in RGB values (Red,Green,Blue)*
- Arithmetic and Logic operations
  - *Addition, multiplication, overflow, sign extension*
- *Machine dependence*
  - *Eg: integers represented and stored using big endian or little endian*

What next: Digital Logic Structures

- Chapter 3 of text [P&P]
- Also read the online links posted on lecture notes webpage
- The hardware building blocks and their operations
- Digital Logic structures
  - Basic device operations: CMOS transistor as switch
  - Combinational Logic circuits
    - Gates (NAND, OR, NOT), Decoder, Multiplexer
    - Adders, multipliers
  - Sequential circuits– concept of memory
    - Finite state machines, memory organization
    - Basic storage elements: latches, flip-flops
    - Memory organization basics

Recall: what are Computers meant to do ?

- We will be solving problems that are describable in English (or Greek or French or Hindi or Chinese or ...) and using a box filled with electrons and magnetism to accomplish the task.
  - This is accomplished using a system of well defined (sometimes) transformations that have been developed over the last 50+ years.

Problem Transformation - levels of abstraction

The desired behavior: the application

- Natural Language
- Algorithm
- Program
- Machine Architecture
- Micro-architecture
- Logic Circuits
- Devices

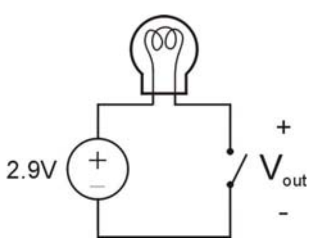
The building blocks: electronic devices

Recall: Why use Binary and How to represent data in a computer?

- At the lowest level, a computer is an electronic machine.
  - works by controlling the flow of electrons
  - Electrons flowing on the wire when voltage exists
- Easy to recognize two conditions:
  1. presence of a voltage – we'll call this state "1"
  2. absence of a voltage – we'll call this state "0"

  More complex to base state on value of voltage
- Think of the two states 0,1 as states of a switch
  - Changing from 0 to 1 means throwing the switch to turn on the light
  - Presence of voltage on the wire means value of bit = 1 else 0

Simple Switch Circuit



- Switch open:
  - No current through circuit
  - Light is off
  - V_{out} is +2.9V
- Switch closed:
  - Short circuit across switch
  - Current flows
  - Light is on
  - V_{out} is 0V

Switch-based circuits can easily represent two states: on/off, open/closed, voltage/no voltage.

Digital Circuits: It's all about switching...

- Action at a distance
- Come here, Watson! I need you!
- Tubes
- Transistors
- CMOS FET

Switching Devices

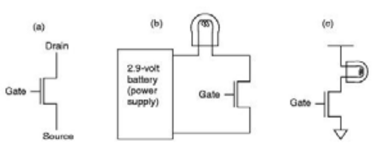
- Vacuum Tubes:
  - Also known as valves because they control the flow of electrons
    - Flow from Cathode to Anode
  - First computer built using vacuum tubes
    - ENIAC
- Transistors - brought about a big change
  - Size
  - Speed
  - Precision
- transistors viewed in digital circuits as a "switch"
  - Transistors used in analog circuits
    - Stereos, recorders, Image proc., etc.

Next Topic...Basics of Digital Circuit Design

- How to build a switch ?
  - Transistors
- How to build basic logic functions – gates using transistors ?
  - Build simple gates (AND, NOT, OR, …) using transistors
- How to build more complex combinational logic using gates
  - Build Adders, multiplexer, decoder, storage devices using simple gates (AND,NOT, OR..)
- Build a whole computer using complex logic devices

Transistors

- An electronic switch that is open or closed between the *source* and the *drain* depending on the voltage on the *gate*.



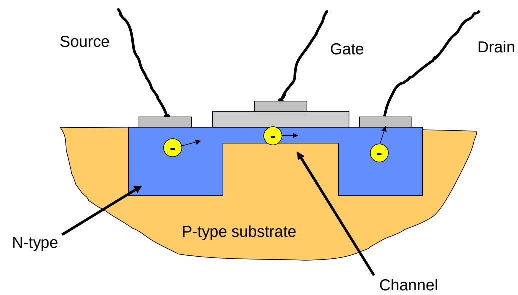
Transistor: Building Block of Computers

- Microprocessors contain millions of transistors
  - Intel Pentium 4 (2000): **48 million**
  - IBM PowerPC 750FX (2002): **38 million**
  - IBM/Apple PowerPC G5 (2003): **58 million**
- Logically, each transistor acts as a switch
- Transistor built using Semiconductor
- Combined to implement logic functions
  - AND, OR, NOT
- Combined to build higher-level structures
  - Adder, multiplexer, decoder, register, …
- Combined to build processor
  - LC-3

Semiconductor

- Most materials are either insulators or conductors
  - They don't "change" their properties
- Semiconductors: between insulator and conductor
- Semiconductors: Based on voltage applied to "gate" it is either insulator or a conductor
  - Electric field creates a circuit
  - Changes the device from an insulator to a conductor

MOS FET (Metal Oxide SemiConductor)



CMOS Transistors

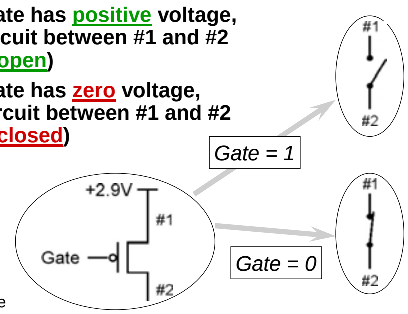
- CMOS
  - = Complementary Metal-Oxide Semiconductor
  - Standard type for digital applications
  - Two versions: P-type (positive) and N-type (negative)
  - P and N-type transistors operate in inverse modes

Open (insulating) if gate is "on" = 1
Closed (conducting) if gate is "off" = 0

Open (insulating) if gate is "off" = 0
Closed (conducting) if gate is "on" = 1

p-type MOS Transistor

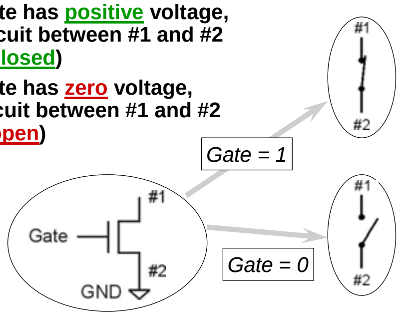
- p-type
  - when Gate has positive voltage, open circuit between #1 and #2 (switch open)
  - when Gate has zero voltage, short circuit between #1 and #2 (switch closed)



Terminal #1 must be connected to +2.9V.

n-type MOS Transistor

- **n-type** ***complementary*** **to p-type**
  - **when Gate has positive voltage, short circuit between #1 and #2 (switch closed)**
  - **when Gate has zero voltage, open circuit between #1 and #2 (switch open)**



Terminal #2 must be connected to GND (0V).

Computer hardware –what's up

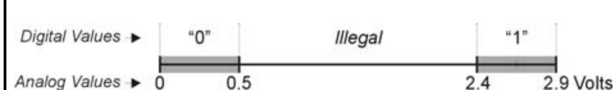
- **Let's look back at what we learnt last week**
  - **Numbers can be represented as 0s and 1s**
    - **1 is presence of voltage on line, 0 is no voltage on line**
  - **Arithmetic operations on these numbers**
  - **Logical operations on these numbers**
- **Starting point: how to implement the basic logic operators using transistors/switches ?**
  - **NOT, AND, OR**
- **Next: how to implement arithmetic operations and other functions**
  - **Combinational circuits; example: adder**

Logical Operations

- **NOT, AND, OR, NAND, NOR, XOR**
- **These are binary functions**
  - **Input is binary, output is binary**
- **Boolean function – operates on boolean variables**
  - **Boolean function can be expressed using truth table**
  - **Eg: addition can be represented as a boolean function**
- **Recall: can implement a boolean function using AND, OR, NOT, etc.**
- **Start by building these logical operator "gates" using transistors**

Logic Gates

- **Use switch behavior of MOS transistors to implement logical functions: AND, OR, NOT.**
- **Digital symbols:**
  - **recall that we assign a range of analog voltages to each digital (logic) symbol**

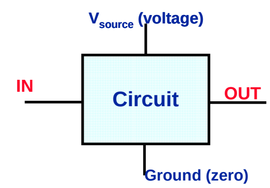


  - **assignment of voltage ranges depends on electrical properties of transistors being used**
    - **typical values for "1": +5V, +3.3V, +2.9V**
    - **from now on we'll use +2.9V**

Ok....start building logic gates

- Use N type and P type transistors
- 'signal' is a 1 or 0 and nothing else
- Output value will be voltage measured at some point in the "circuit"
  - Need to determine where to designate the output point (i.e., where to measure)
- Inputs will be applied to the transistor gate
  - A line in the circuit always tied to 1 (voltage source) and one always tied to 0 (ground)
- Start by looking at the truth table for the logic function

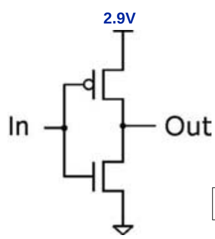
NOT Gate

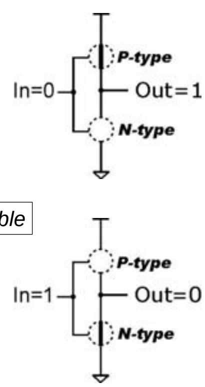


In	Out
0 V	2.9 V
2.9 V	0 V

In	Out
0	1
1	0

Inverter (NOT Gate)



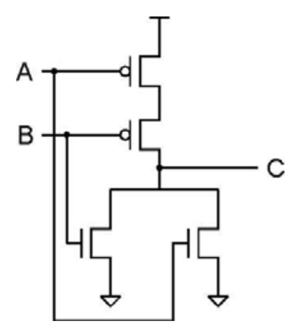


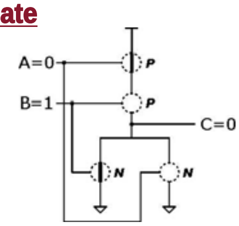
Truth table

In	Out
0 V	2.9 V
2.9 V	0 V

In	Out
0	1
1	0

NOR Gate

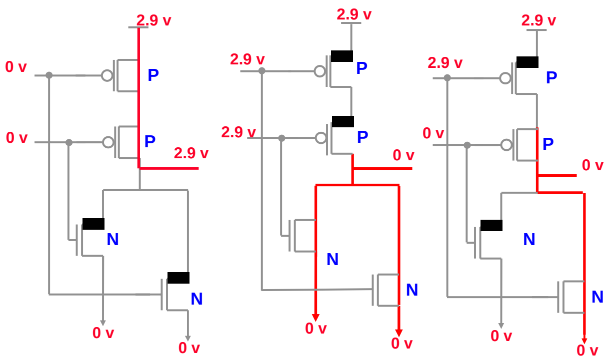




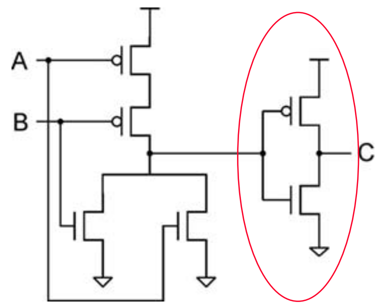
A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

Note: Serial structure on top, parallel on bottom.

NOR Gate - Operation



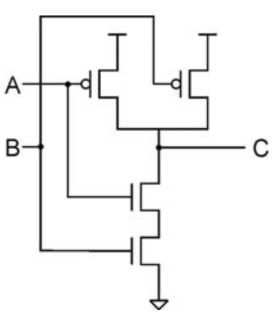
OR Gate

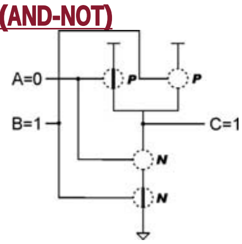


A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

Add inverter to NOR.

NAND Gate (AND-NOT)

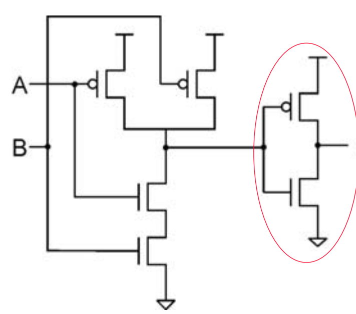




A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

Note: Parallel structure on top, serial on bottom.

AND Gate



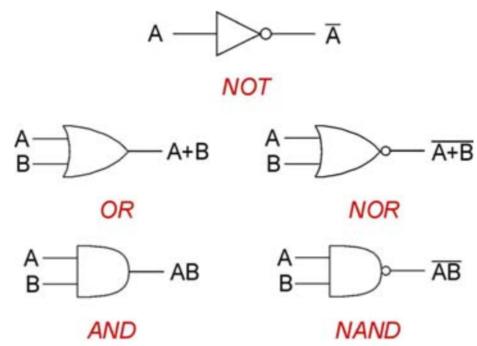
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

Add inverter to NAND.

Digital Logic Design – Current Summary

- MOS transistors used as switches to implement logic functions.
  - n-type: connect to GND, turn on (with 1) to pull down to 0
  - p-type: connect to +2.9V, turn on (with 0) to pull up to 1
- Basic gates: NOT, NOR, NAND
  - Logic functions are usually expressed with AND, OR, and NOT
- Next: a little theory behind combinational logic design and some basic combinational devices
  - DeMorgan's Law
  - Decoder, Multiplexer, Adder, PLA
  - Boolean Algebra

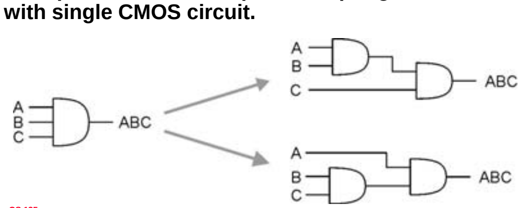
Basic Logic Gates



More than 2 Inputs?

- AND/OR can take any number of inputs.
  - AND = 1 if all inputs are 1.
  - OR = 1 if any input is 1.
  - Similar for NAND/NOR.

- Can implement with multiple two-input gates, or with single CMOS circuit.

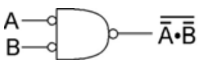


Digital Logic Circuits

- We saw how we can build the simple logic gates using transistors
- Use these gates as building blocks to build more complex combinational circuits
  - Decoder: based on value of n-bit input control signal, select one of 2^N outputs
  - Multiplexer: based on value of N-bit input control signal, select one of 2^N inputs.
  - Adder: add two binary numbers
  - …any boolean function

DeMorgan's Law

- Converting AND to OR (with some help from NOT)
- Consider the following gate:



A B	$\bar{A}$	$\bar{B}$	$\bar{A} \cdot \bar{B}$	$\overline{\bar{A} \cdot \bar{B}}$
0 0	1	1	1	0
0 1	1	0	0	1
1 0	0	1	0	1
1 1	0	0	0	1

Same as A+B

To convert AND to OR (or vice versa), invert inputs and output.

CS 135

DeMorgan's Law

- NOT(A AND B) = (NOT A) OR (NOT B);
  - i.e. NOT(NOT A AND NOT B) = A OR B
- NOT(A OR B) = (NOT A) AND (NOT B);
  - i.e., NOT(NOT A OR NOT B) = A AND B
- In C syntax:
  - ~(~A&~B) = A|B
  - ~(~A|~B) = A&B

A	B	~A	~B	~A&~B	~(~A&~B)	~A|~B	~(~A|~B)	A|B	A&B
0	0	1	1	1	0	1	0	0	0
0	1	1	0	0	1	1	0	1	0
1	0	0	1	0	1	1	0	1	0
1	1	0	0	0	1	0	1	1	1

CS 135

A note on De-Morgan's Law

- Where have you seen this before ?
  - In a different context ?
- Set operations: Union, Intersect, Comp.
  - $(A^c \cup B^c) = (A \cap B)^c$
  - $(A^c \cap B^c) = (A \cup B)^c$
- De-Morgan's law applies to any boolean algebra

CS 135

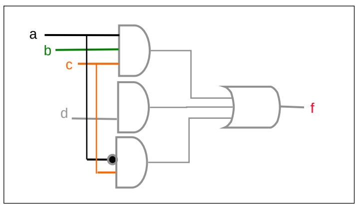
Completeness: Very Important Concept

- It can be shown that any truth table (i.e. any binary function of binary variables) can be reduced to combinations of the AND & NOT functions, or of the OR & NOT functions.
  - This result extends also to functions of more than two variables
  - Methodology: Karnaugh Maps
- In fact, it turns out to be convenient to use a basic set of three logic gates:
  - AND, OR & NOT or NAND, NOR & NOT
  - In fact, can implement all logic functions using just NAND!

CS 135

Representation of Boolean Logic Functions

- What is this circuit doing ?



CS 135

Boolean Logic Function

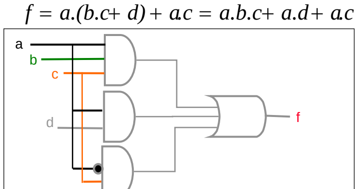
- Output(s) is a function of input boolean variables
- *y = f(x,y,…)*
- The “operators” used are any of the boolean logic operators
  - AND, OR, NOT, NAND, etc.
- How do we represent boolean functions?

CS 135

Representation of Logic Functions

- A logic function can be represented as
  - a truth table
  - a logic expression
  - a logic circuit
- Example

$$f = a.(b.c + d) + \bar{a}c = a.b.c + a.d + \bar{a}c$$



a	b	c	d	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

CS 135

Truth tables and Boolean functions

A	B	C	x_1	x_2
0	0	0	0	1
0	0	1	0	0
0	1	0	1	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	1
1	1	0	0	0
1	1	1	1	0

CS 135

Truth Table to Digital Circuit design…

- Look at all rows with a 1 in the output
  - ALL conditions in the input must hold for the output to be 1 = AND of all the input conditions
  - Any row can be 1 = do an OR of all the row conditions
- When is x_1 = 1 ?
  - Look at all rows where x_1= 1
  - What are the values of inputs for each of these rows ?

Canonical Boolean expressions with Minterms

- Each row in a truth table specifies values of all the input variables
  - What is the value of each input variable – 0 or 1
  - i.e., y=1 or y'=1 for each input variable y
- For the specific output, what row(s) are we interested ?
- When is the value of the output =1
  - When the value in the row of the truth table =1
  - What are the values of the input variables ?
- Canonical boolean expression
  - Any boolean expression can be converted to an equivalent two level AND-OR expression
    - an OR of AND terms,
    - each AND term corresponds to a 1 in the truth table row,
    - each AND term contains all input variables exactly once – i.e., a *minterm*
  - Canonical expression: OR of minterms

Boolean function from a Truth Table

- OR of minterms
- Each minterm is an AND of the input variables y
  - y or y' is true in each row

Circuits and Boolean Functions

- How to get boolean function represented by a circuit ?
- Trace each output line through the gates

Combinational and Sequential Circuits

- A circuit is a collection of devices that are physically connected by wires
  - Combinational circuit
  - Sequential circuit
- In Combinational circuit the input determines output
- In sequential circuit, the input and the previous 'state' (previous values) determine output and next 'state'
  - Need circuit to implement concept of storage

Combinational Devices

- Use basic gates to build more complex combinational logic functions

Problem

- You have an *n* bit binary number assigned as unique ID to each student. How do we select & physically connect to a specific student with ID *y* ?
- Example:
  - 00 is Sam, 01 is Krista, 10 is Zach, 11 is Alex
  - We want to select Zach:
  - Give binary 10 as input, and the output line to Zach is set to high - 110 volts ☺

Boolean function for "selector"

- Need to select one of four:
  - 2 bits needed to encode the four outcomes
  - $a_1 a_0$
- 4 outputs – 1 associated with each signal
  - $x_3 x_2 x_1 x_0$
- What is the boolean function ?
- When is each x_i set to 1:
  - $x_0 = a_1'.a_0'$ (NOT a_1 AND NOT a_0)
  - $x_1 = a_1'.a_0$
  - $x_2 = a_1.a_0'$
  - $x_3 = a_1.a_0$

Truth table

a_1	a_0	x_0	x_1	x_2	x_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

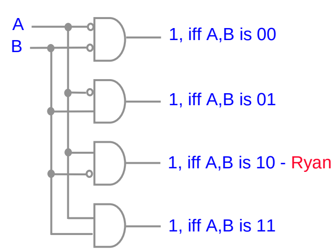
Truth Table to Digital Circuit design…

- Look at all rows with a 1 in the output
  - ALL conditions in the input must hold for the output to be 1 = AND of all the input conditions
  - Any row can be 1 = do an OR of all the row conditions
- When is $x_2 = 1$?

Notation shortcuts…

- What happens when circle is put at input of AND gate ?
- Invert signal – same as adding a NOT gate in the input line and then sending it to input of AND gate

Decoder



2-bit decoder
(4 input decoder)

- An n input decoder has 2^n outputs.
- Output_i is 1 iff the binary value of the n-bit input is i.
- At any time, exactly one output is 1, all others are 0.

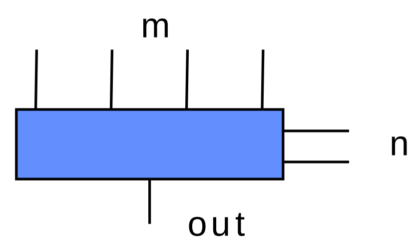
Truth Table to Digital Circuit design…

- Look at all rows with a 1 in the output
  - ALL conditions in the input must hold for the output to be 1 = AND of all the input conditions
  - Any row can be 1 = do an OR of all the row conditions

CS 135

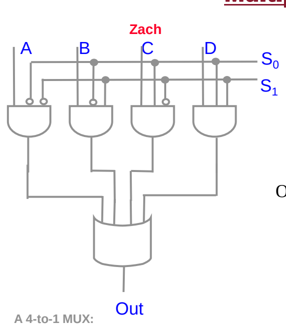
Problem: Selecting one of many

- You have *m* input signals and you want to use the logical value on one of them determined by a set control signals/wires – *n* control signals
  - Each student sends a signal (0 or 1)
  - I want to select Zach's signal – so I can give him an A grade…
    - Need to give Zach's code of 10 to select his answer



CS 135

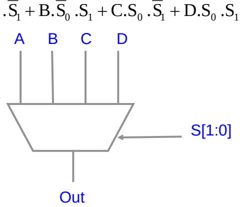
Multiplexer (MUX)



A 4-to-1 MUX:
Out takes the value of A,B, C or D depending on the value of S (00, 01, 10, 11)

- In general, a MUX has
  - 2^n data inputs
  - n select (or control) lines
  - and 1 output.
- It behaves like a channel selector.

$$\text{Out} = A.\overline{S_0}.\overline{S_1} + B.\overline{S_0}.S_1 + C.S_0.\overline{S_1} + D.S_0.S_1$$



CS 135

Problem

- No one will buy your new computer design unless it can do at least some math, say, like adding!
- How to build hardware for adding 2 binary numbers using what we have learnt so far ?
- First look at the function performed by addition – we saw this last week
  - Bit by bit addition, right to left, propagate carry
  - Inputs: A, B and Carry-in
  - Output: sum bit and carry-out (to next bit position)

CS 135

Truth table for Addition

A	B	Carry In	Out	Carry Out
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

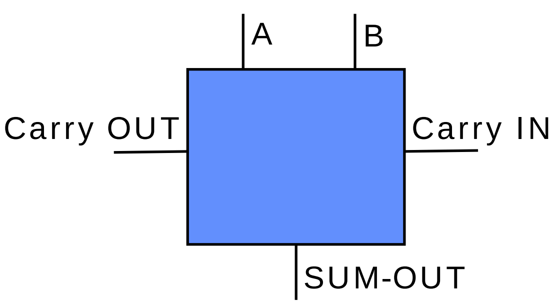
CS 135

Truth

A	B	Carry In	Out	Carry Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

CS 135

Full Adder



CS 135

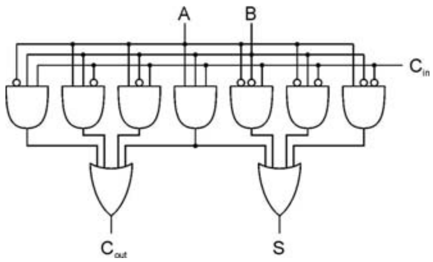
Boolean function for Addition

- When is Sum =1, as a function of A, B, C_{in}
  - Write function for S
- $S = A'B'C_{in} + A'BC'_{in} + AB'C_{in}' + ABC_{in}$
- When is C_{out} = 1, as a function of A, B, C_{in}
  - Write function for C_{out}
- $C_{out} = A'BC_{in} + AB'C_{in} + ABC_{in}' + ABC_{in}$

CS 135

Full Adder

- Add two bits and carry-in, produce one-bit sum and carry-out.

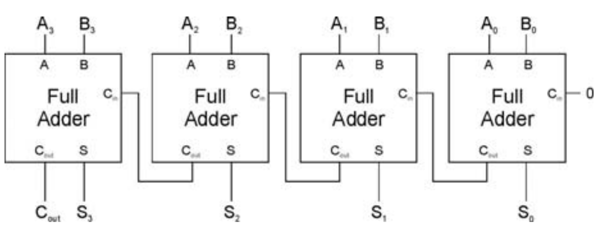


A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

N-bit Adder

- Use the building block of the full-adder to build N-bit adder
  - Need to connect carry-out to carry-in of next significant bit

Four-bit Adder



Is there a theory behind design of combinational logic devices ?

- To describe behavior of combinational circuit
  - Truth table
  - Boolean expressions
  - Digital logic circuit/diagram
- Equivalent circuits ?
- Efficient design ?
  - Fewest gates used
- Boolean Algebra: algebraic expression written according to laws of boolean algebra specifies not only what a combinational circuit does, but also how it does it!