Computer Science 210
Computer Systems 1
Lecture Notes

Lecture 7
Digital Logic Structures

Credits: Slides prepared by Gregory T. Byrd, North Carolina State University

Building Functions from Logic Gates

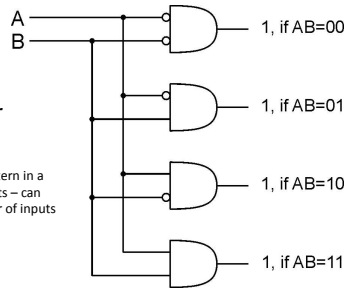
Combinational Logic Circuit
- output depends only on the current inputs
- stateless

Sequential Logic Circuit
- output depends on the sequence of inputs (past and present)
- stores information (state) from past inputs

We'll first look at some useful combinational circuits,
then show how to use sequential circuits to store information.

Decoder

- n inputs, 2^n outputs
  - exactly one output is 1 (true) for each possible input pattern



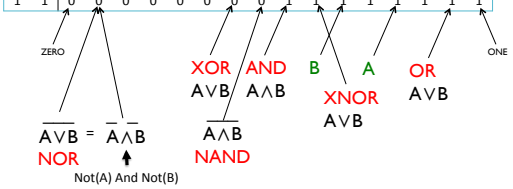
2-bit decoder

Can detect a pattern in a string of input bits – can have any number of inputs

Two Variables

16 possible unique functions

A	B	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1



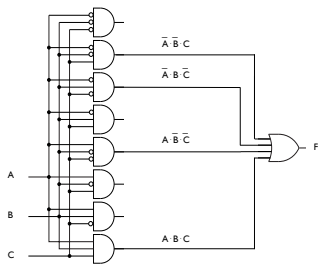
Truth Table for Function F

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

XOR
F is True is 1 or 3 inputs are true
F is False if zero or 2 inputs are true
More generally F is true if an odd number of inputs are true

Creating a Circuit from a Truth Table

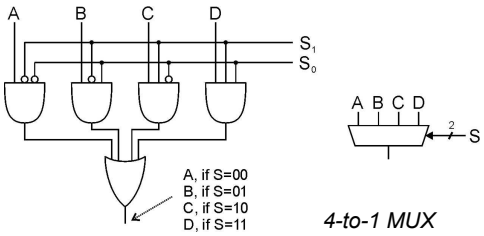
A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



Multiplexer (MUX)

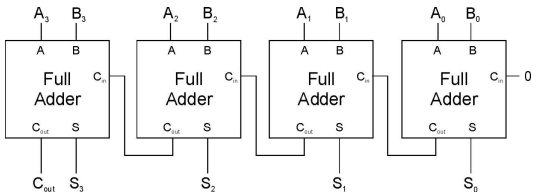
n-bit selector and 2^n inputs, one output

– output equals one of the inputs, depending on selector S_1 & S_2



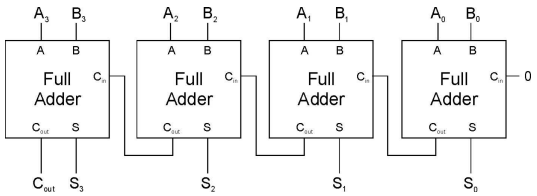
4-to-1 MUX

Four-bit Adder



This is a ripple-carry adder – operates from right to left

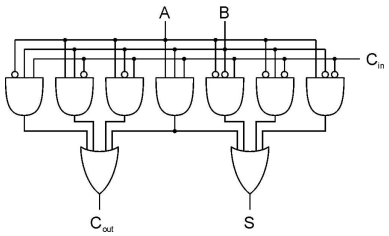
Four-bit Adder



Each full adder is implemented with an XOR gate with three inputs A, B & C
The output S is True (1) when an odd number of inputs is true
The C_{out} is True (1) if 2 or 3 inputs are true (the majority)

Full Adder

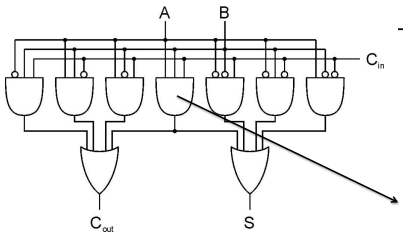
Add two bits and carry-in,
produce one-bit sum and carry-out



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

Full Adder

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

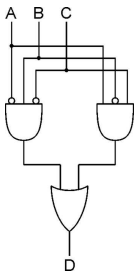


A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Logical Completeness

•Can implement *ANY* truth table with AND, OR, NOT

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



1. AND combinations that yield a "1" in the truth table.

2. OR the results of the AND gates.

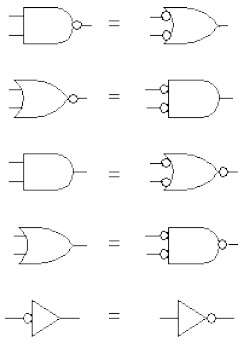
Logical Completeness

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

Can implement *ANY* truth table with just NAND or NOR

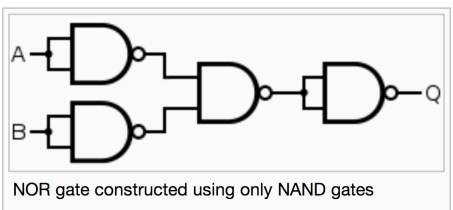
We might not want to for reasons of simplicity

DeMorgan's Law



De Morgan's Theorem

Using Only NAND gates to create a NOR gate

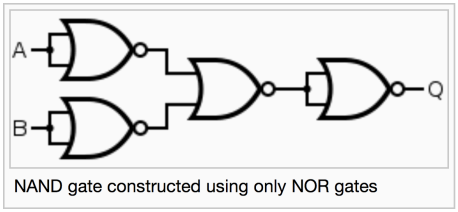


NOR gate constructed using only NAND gates

credit: http://en.wikipedia.org/wiki/NOR_gate

De Morgan's Theorem

Using Only NOR gates to create a NAND gate



NAND gate constructed using only NOR gates

credit: http://en.wikipedia.org/wiki/NOR_gate

Combinational vs. Sequential

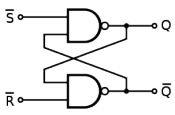
Combinational Circuit
- always gives the same output for a given set of inputs
  - ex: adder always generates sum and carry, regardless of previous inputs

Sequential Circuit
- stores information
- output depends on stored information (state) plus input
  - so a given input might produce different outputs, depending on the stored information
- *example:* ticket counter
  - advances when you push the button
  - output depends on previous state
- useful for building "memory" elements and "state machines"

SR-Latch: Simple Storage Element

Flip-Flop

S is used to "set" the element – set output Q to one
R is used to "reset" or "clear" the element – set output Q to zero



NAND SR-Latch

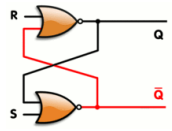
$\overline{SR}$ latch operation		
$\overline{S}$	$\overline{R}$	Action
0	0	Restricted combination
0	1	Q = 1
1	0	Q = 0
1	1	No Change

This gives us the ability to store a bit (either 0 or 1)

Watch the video http://youtu.be/ti5jD7Q7BSA

SR-Latch: Simple Storage Element
Flip-Flop

S is used to "set" the element – set output Q to one
R is used to "reset" or "clear" the element – set output Q to zero



NOR SR-Latch

Characteristic table			
S	R	Q_{next}	Action
0	0	Q	hold state
0	1	0	reset
1	0	1	set
1	1	X	not allowed

This gives us the ability to store a bit (either 0 or 1)