Fundamentals of Computer Systems

Combinational Logic

Martha A. Kim

Columbia University

Fall 2014

Combinational Circuits

Combinational circuits are stateless.

Their output is a function *only* of the current input.

Inputs $\longrightarrow$ Combinational Circuit $\longrightarrow$ Outputs

Basic Combinational Circuits

Enabler

Encoders and Decoders

Multiplexers

Shifters

Circuit Timing

Critical and Shortest Paths

Glitches

Arithmetic Circuits

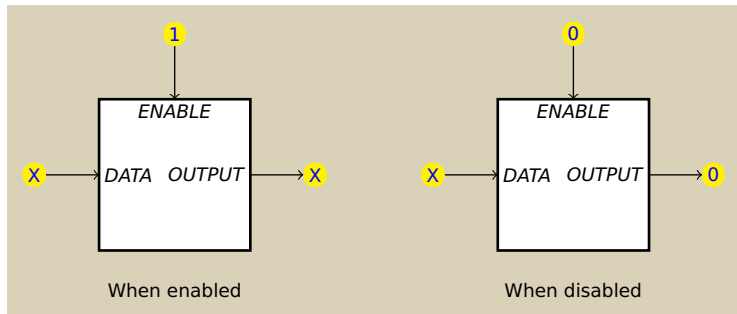Ripple Carry Adder

Adder/Subtractor

Carry Lookahead Adder

Enablers

Overview: Enabler

An enabler has two inputs:

- data: can be several bits, but 1 bit examples for now
- enable/disable: 1 bit on/off switch

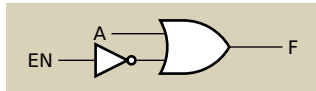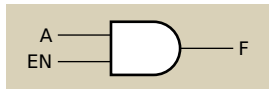When enabled, the circuit's output is its input data. When disabled, the output is 0.

When enabled   When disabled

Enabler Implementation

Note abbreviated truth table: input, A, listed in output column

EN	F
0	0
1	A

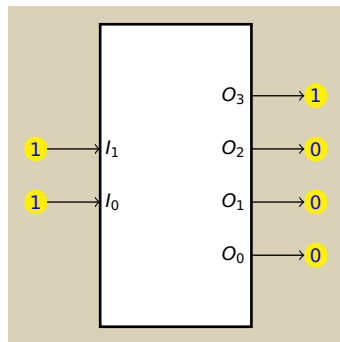| EN	F
0	1
1	A

In both cases, output is enabled when $EN = 1$, but they handle the disabled ($EN = 0$) cases differently.
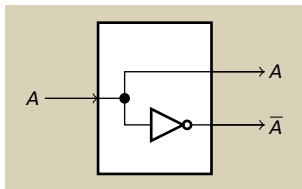
Encoders and Decoders

Overview: Decoder

A decoder takes a $k-bit$ input and produces 2^k single-bit outputs.

The input determines which output will be 1, all others 0. This representation is called *one-hot encoding*.
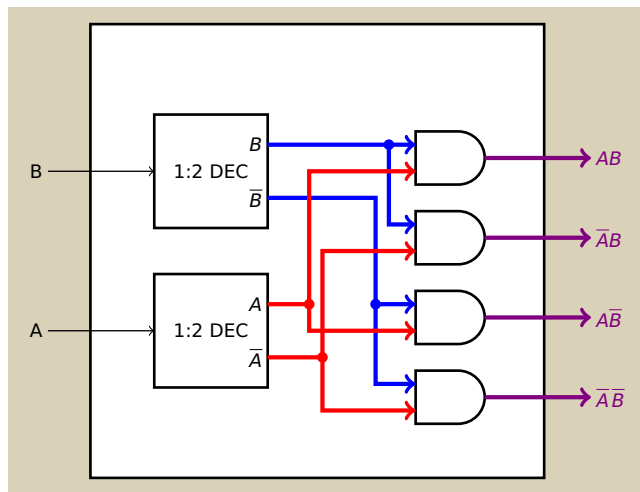
1:2 Decoder

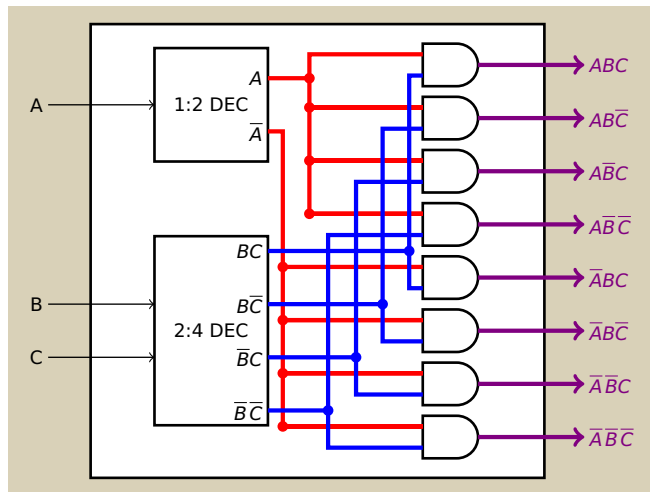The smallest decoder: one bit input, two bit outputs

2:4 Decoder

Decoder outputs are simply minterms. Those values can be constructed as a flat schematic (manageable at small sizes) or hierarchically, as below.
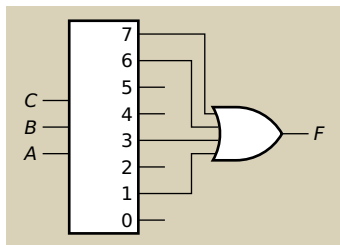
3:8 Decoder

Applying *hierarchical design* again, the 2:4 DEC helps construct a 3:8 DEC.
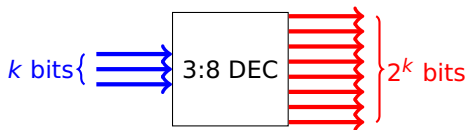
Implementing a function with a decoder

E.g., $F = A\overline{C} + BC$

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

Warning: Easy, but not a minimal circuit.

Encoders and Decoders

k bits $\{$ → 3:8 DEC → $\}$ 2^k bits

BCD			One-Hot							
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

k bits $\{$ ← 3:8 ENC ← $\}$ 2^k bits
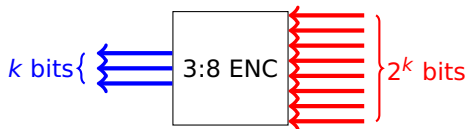
Priority Encoder

An encoder designed to accept any input bit pattern.

I_3	I_2	I_1	I_0	V	O_1	O_0
0	0	0	0	0	X	X
0	0	0	1	1	0	0
0	0	1	X	1	0	1
0	1	X	X	1	1	0
1	X	X	X	1	1	1

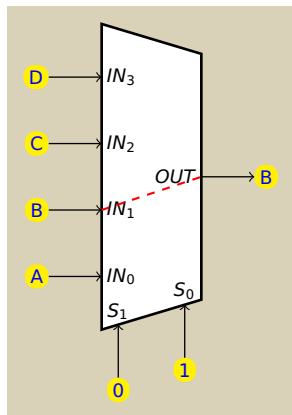$$V = I_3 + I_2 + I_1 + I_0$$
$$O_1 = I_3 + \overline{I_3}I_2$$
$$O_0 = I_3 + \overline{I_3}\,\overline{I_2}I_1$$

Multiplexers

Overview: Multiplexer (or Mux)

A mux has a $k - bit$ selector input and 2^k data inputs (multi or single bit).

It outputs a single data output, which has the value of one of the data inputs, according to the selector.

2:1 Mux Circuit

There are a handful of implementation strategies.

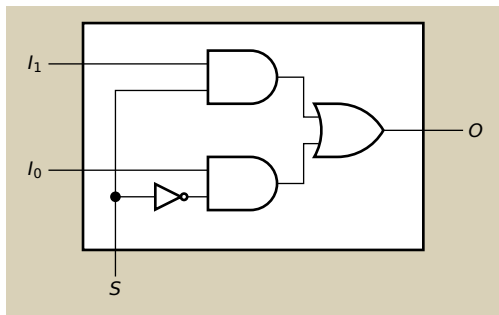E.g., a truth table and k-map are feasible for a design of this size.

| S	I_1	I_0	0
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

2:1 Mux Circuit

There are a handful of implementation strategies.

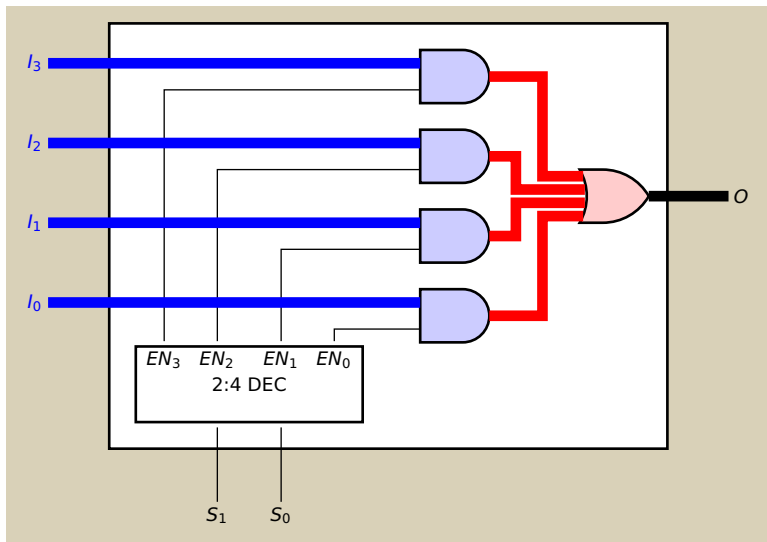E.g., a truth table and k-map are feasible for a design of this size.

S	I_1	I_0	O
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

2:1 Mux Circuit

There are a handful of implementation strategies.

E.g., a truth table and k-map are feasible for a design of this size.

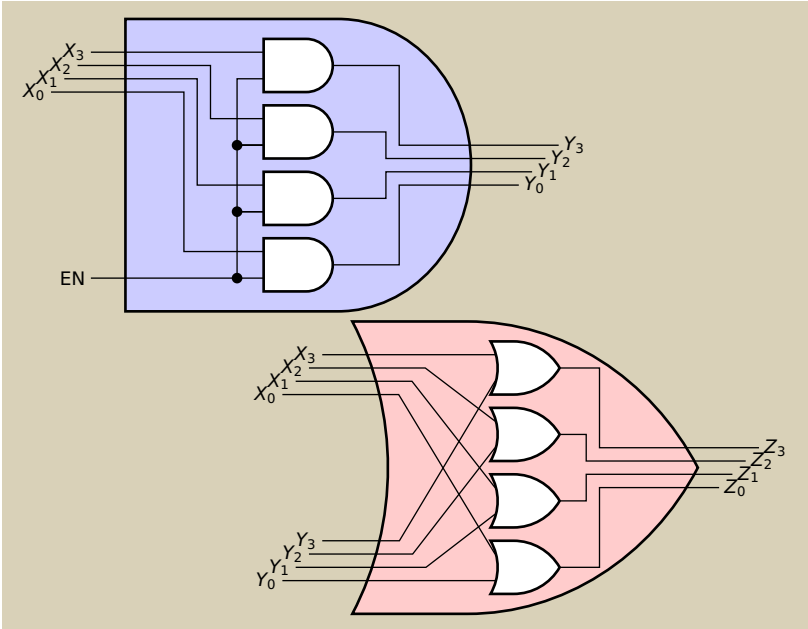| S	I_1	I_0	0
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1
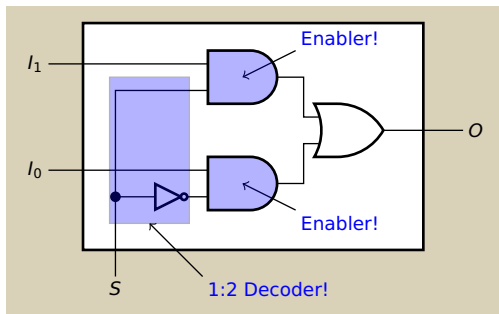
4:1 Mux Circuit
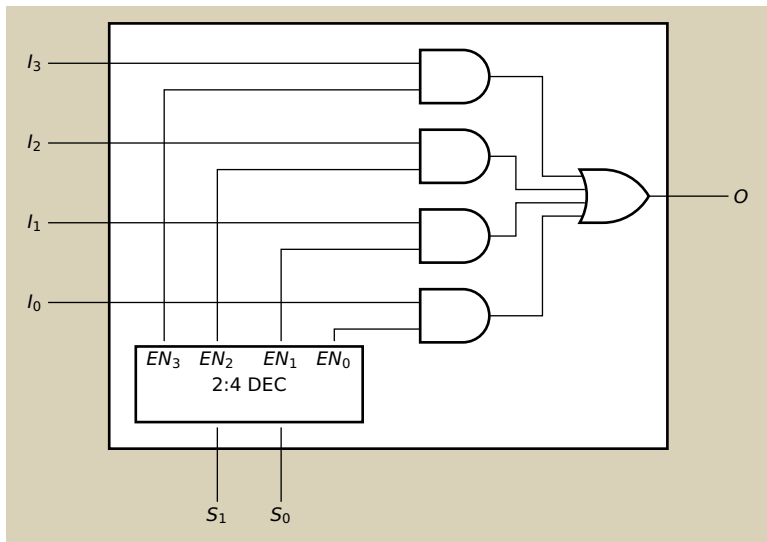
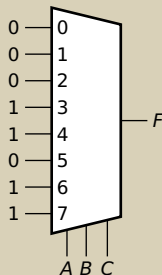# Muxing Wider Values (Overview)

Muxing Wider Values (Components)

Using a Mux to Implement an Arbitrary Function Version 1

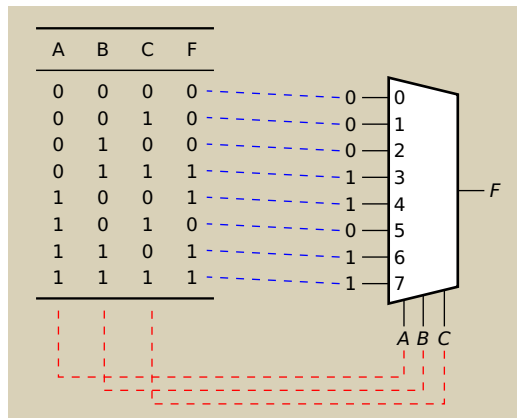Think of a function as using k input bits to choose from 2^k outputs.

E.g., $F = BC + A\overline{C}$

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

Using a Mux to Implement an Arbitrary Function Version 1

Think of a function as using k input bits to choose from 2^k outputs.

E.g., $F = BC + A\overline{C}$

Using a Mux to Implement an Arbitrary Function Version 2

Can we use a smaller MUX?

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

Using a Mux to Implement an Arbitrary Function Version 2

Can we use a smaller MUX?

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

Using a Mux to Implement an Arbitrary Function Version 2

Can we use a smaller MUX?

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

Using a Mux to Implement an Arbitrary Function Version 2

Can we use a smaller MUX?

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

A	B	F
0	0	0
0	1	C
1	0	$\overline{C}$
1	1	1

Using a Mux to Implement an Arbitrary Function Version 2

Can we use a smaller MUX?

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

A	B	F
0	0	0
0	1	C
1	0	$\overline{C}$
1	1	1
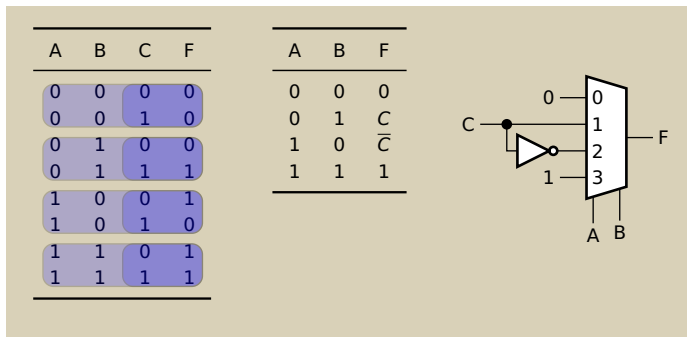
Instead of feeding just 0 or 1 into the mux, as in Version 1, one can remove a bit from the select, and feed it into the data ports along with the constant.

Shifters

Overview: Shifters

A shifter shifts the inputs bits to the left or to the right.
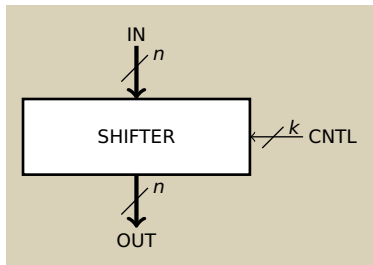
There are various types of shifters.

- ▶ Barrel: Selector bits indicate (in binary) how far to the left to shift the input.
- ▶ L/R with enable: Two control bits (upper enables, lower indicates direction).

In either case, bits may "roll out" or "wraparound"

Example: Barrel Shifter with Wraparound

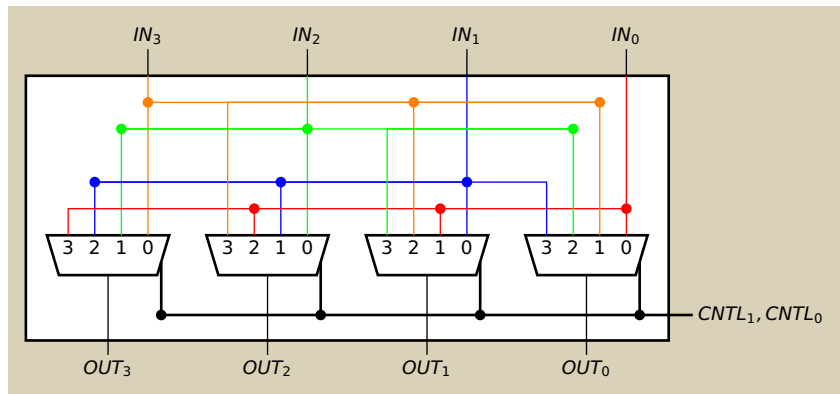# Implementation of Barrel Shifter with Wraparound (Part 2)

Main idea: wire up all possible shift amounts and use muxes to select correct one.

Implementation of Barrel Shifter with Wraparound (Part 2)

Main idea: wire up all possible shift amounts and use muxes to select correct one.

Implementation of Barrel Shifter with Wraparound (Part 2)

Main idea: wire up all possible shift amounts and use muxes to select correct one.

Implementation of Barrel Shifter with Wraparound (Part 2)

Main idea: wire up all possible shift amounts and use muxes to select correct one.

Implementation of Barrel Shifter with Wraparound (Part 2)

Main idea: wire up all possible shift amounts and use muxes to select correct one.

Implementation of Barrel Shifter with Wraparound (Part 2)

Main idea: wire up all possible shift amounts and use muxes to select correct one.

Circuit Timing

Computation Always Takes Time
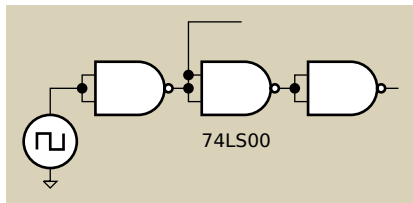
There is a delay between inputs and outputs, due to:

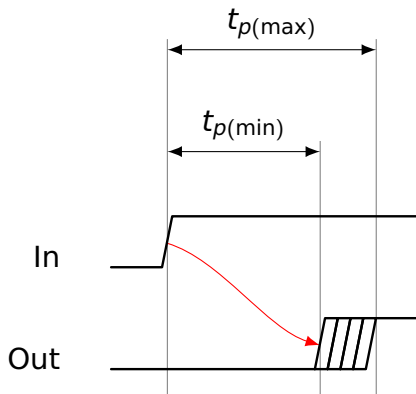· Limited currents charging capacitance

· The speed of light
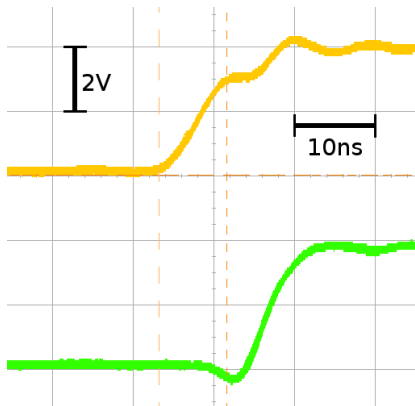
The Simplest Timing Model
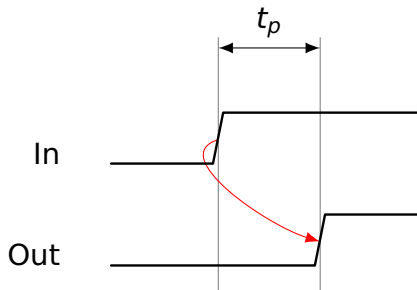
- ▶ Each gate has its own propagation delay t_p.
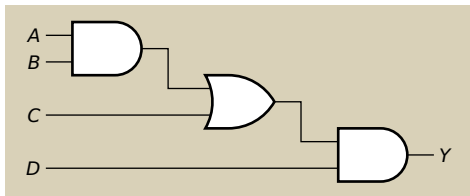- ▶ When an input changes, any changing outputs do so after t_p.
- ▶ Wire delay is zero.

A More Realistic Timing Model

It is difficult to manufacture two gates with the same delay; better to treat delay as a range.
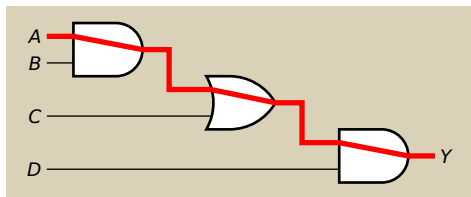
- Each gate has a minimum and maximum propagation delay $t_{p(\min)}$ and $t_{p(\max)}$.
- Outputs may start changing after $t_{p(\min)}$ and stablize no later than $t_{p(\min)}$.

Critical Paths and Short Paths

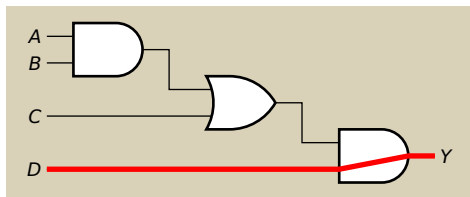How slow can this be?

Critical Paths and Short Paths

How slow can this be?

The critical path has the longest possible delay.

$$t_{p(\text{max})} = t_{p(\text{max, AND})} + t_{p(\text{max, OR})} + t_{p(\text{max, AND})}$$
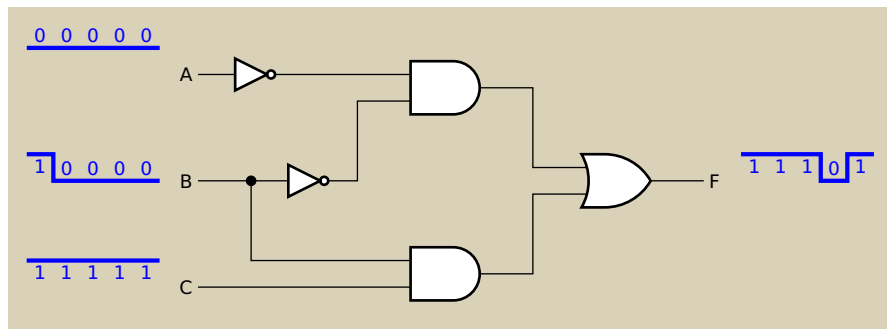
Critical Paths and Short Paths

How fast can this be?

The shortest path has the least possible delay.

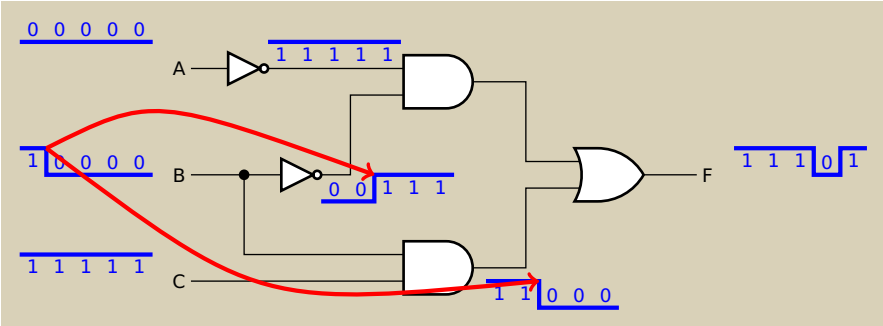$$t_{p(\text{min})} = t_{p(\text{min, AND})}$$
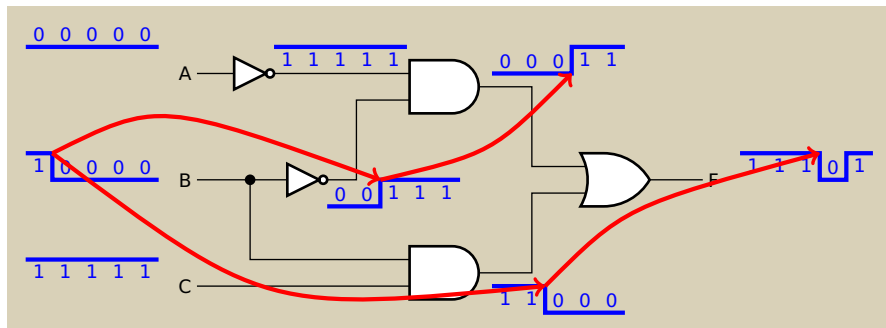
Glitches

A glitch is when a single change in input values can cause multiple output changes.

Glitches *may* occur when there are multiple paths of different length from input *I* to output *O*.
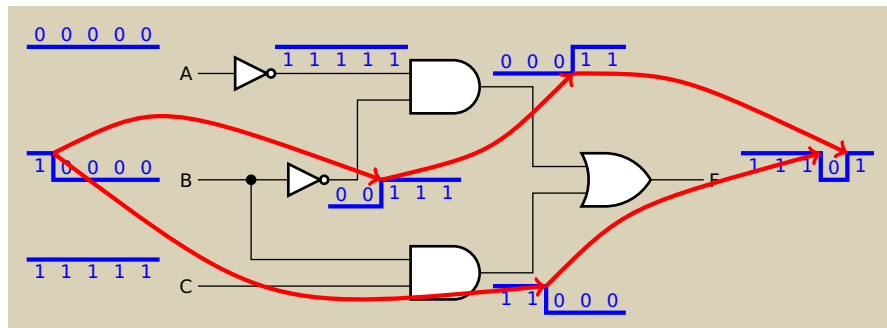
Glitches

A glitch is when a single change in input values can cause multiple output changes.

Glitches *may* occur when there are multiple paths of different length from input *I* to output *O*.

Glitches

A glitch is when a single change in input values can cause multiple output changes.

Glitches *may* occur when there are multiple paths of different length from input *I* to output *O*.

Glitches

A glitch is when a single change in input values can cause multiple output changes.
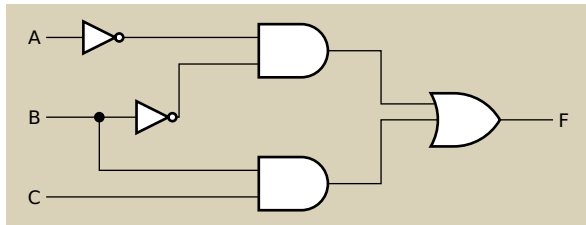

Glitches *may* occur when there are multiple paths of different length from input *I* to output *O*.

Preventing Single Input Glitches

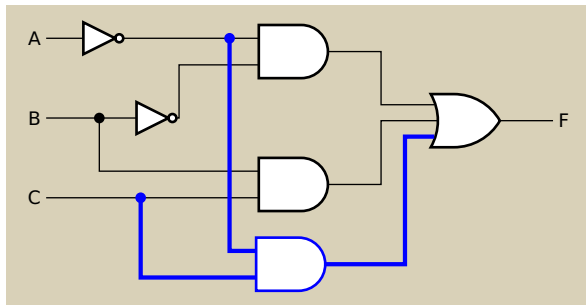Additional terms can prevent single input glitches (at a cost of a few extra gates).

Preventing Single Input Glitches

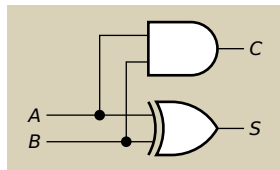Additional terms can prevent single input glitches (at a cost of a few extra gates).

Arithmetic Circuits

Arithmetic: Addition

Adding two one-bit numbers: A and B

Produces a two-bit result: C and S (carry and sum)

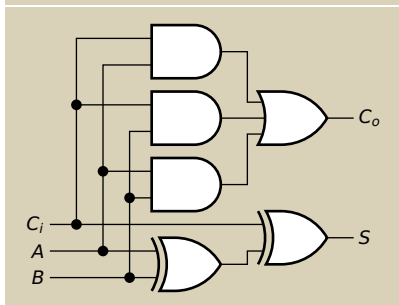| A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Half Adder

Full Adder

In general, due to a possible carry in, you need to add *three bits*:

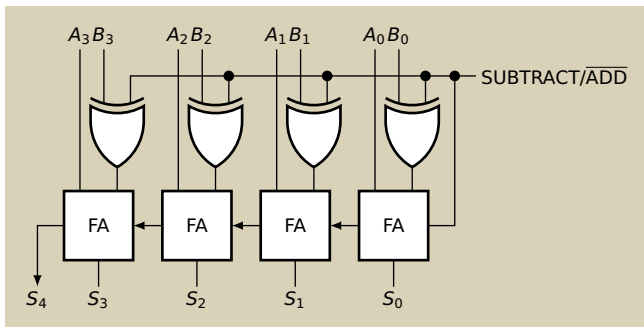| $C_i A B$	$C_o S$
0 0 0	0 0
0 0 1	0 1
0 1 0	0 1
0 1 1	1 0
1 0 0	0 1
1 0 1	1 0
1 1 0	1 0
1 1 1	1 1
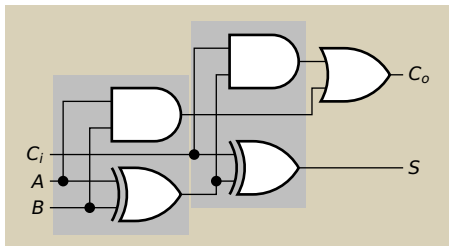
A Four-Bit Ripple-Carry Adder
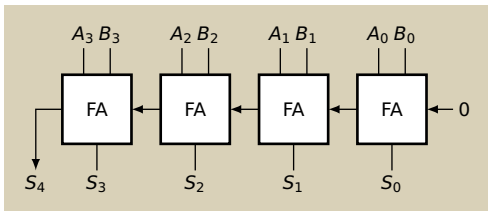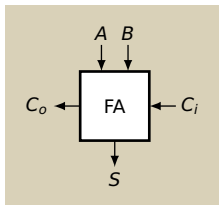
A Two's Complement Adder/Subtractor

To subtract B from A, add A and $-B$.
Neat trick: carry in takes care of the $+1$ operation.

Overflow in Two's-Complement Representation

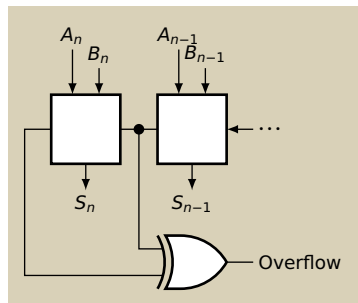When is the result too positive or too negative?

+	−2	−1	0	1

| −2 | 10
10
+10
00 | | | |

| −1 | 10
10
+11
01 | 11
11
+11
10 | | |

| 0 | 00
10
+00
10 | 00
11
+00
11 | 00
00
+00
00 | |

| 1 | 00
10
+01
11 | 11
11
+01
00 | 00
00
+01
01 | 01
01
+01
10 |

Overflow in Two's-Complement Representation

When is the result too positive or too negative?

+	−2	−1	0	1
−2	10 10 +10 00 ✗			
−1	10 10 +11 01 ✗	11 11 +11 10		
0	00 10 +00 10	00 11 +00 11	00 00 +00 00	
1	00 10 +01 11	11 11 +01 00	00 00 +01 01	01 01 +01 10 ✗

The result does not fit when the top two carry bits differ.

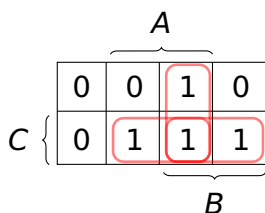# Ripple-Carry Adders are Slow
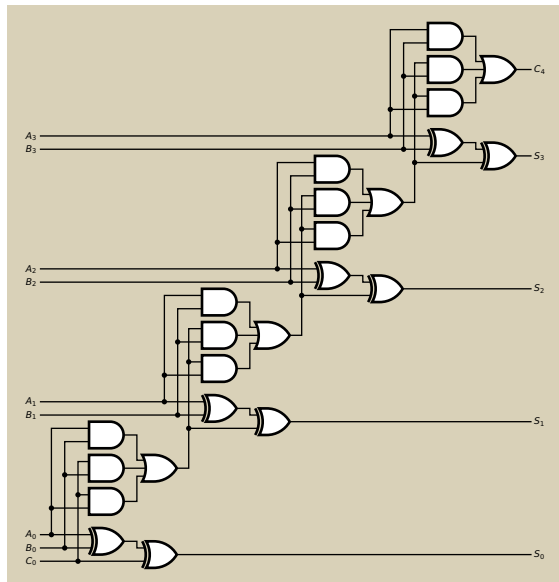
The *depth* of a circuit is the number of gates on a critical path.

This four-bit adder has a depth of 8.

n-bit ripple-carry adders have a depth of $2n$.

Carry Generate and Propagate

The carry chain is the slow part of an adder;
carry-lookahead adders reduce its depth using the
following trick:

A

0	0	1	0
0	1	1	1

C brackets the rows; B brackets the lower columns

K-map for the
carry-out
function of a full
adder

For bit i,

$$
\begin{aligned}
C_{i+1} &= A_i B_i + A_i C_i + B_i C_i \\
&= A_i B_i + C_i (A_i + B_i) \\
&= G_i + C_i P_i
\end{aligned}
$$

Generate $G_i = A_i B_i$ sets carry-out
regardless of carry-in.

Propagate $P_i = A_i + B_i$ copies carry-in to
carry-out.

Carry Lookahead Adder

Expand the carry functions into sum-of-products form:

$$C_{i+1} = G_i + C_i P_i$$

$$
\begin{aligned}
C_1 &= G_0 + C_0 P_0 \\
C_2 &= G_1 + C_1 P_1 \\
    &= G_1 + (G_0 + C_0 P_0) P_1 \\
    &= G_1 + G_0 P_1 + C_0 P_0 P_1 \\
C_3 &= G_2 + C_2 P_2 \\
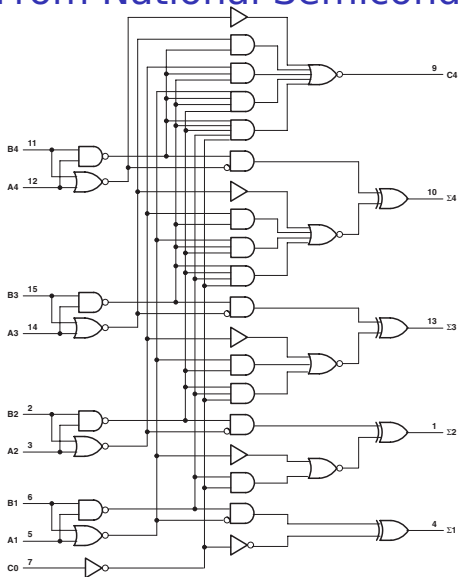   &= G_2 + (G_1 + G_0 P_1 + C_0 P_0 P_1) P_2 \\
    &= G_2 + G_1 P_2 + G_0 P_1 P_2 + C_0 P_0 P_1 P_2 \\
C_4 &= G_3 + C_3 P_3 \\
    &= G_3 + (G_2 + G_1 P_2 + G_0 P_1 P_2 + C_0 P_0 P_1 P_2) P_3 \\
    &= G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3 + C_0 P_0 P_1 P_2 P_3
\end{aligned}
$$

The 74283 Binary Carry-Lookahead Adder (From National Semiconductor)

Carry out i has $i+1$ product terms, largest of which has $i+1$ literals.

If wide gates don't slow down, delay is independent of number of bits.

More realistic: if limited to two-input gates, depth is $O(\log_2 n)$.