Fundamentals of Computer Systems

Combinational Logic

Stephen A. Edwards

Columbia University

Spring 2012

Encoders and Decoders

Decoders

Input: n-bit binary number

Output: 1-of-2^n one-hot code

2-to-4	
in	**out**
00	0001
01	0010
10	0100
11	1000

Decoders

Input: n-bit binary number

Output: 1-of-2^n one-hot code

2-to-4	
in	**out**
00	0001
01	0010
10	0100
11	1000

3-to-8 decoder	
in	**out**
000	00000001
001	00000010
010	00000100
011	00001000
100	00010000
101	00100000
110	01000000
111	10000000

Decoders

Input: n-bit binary number

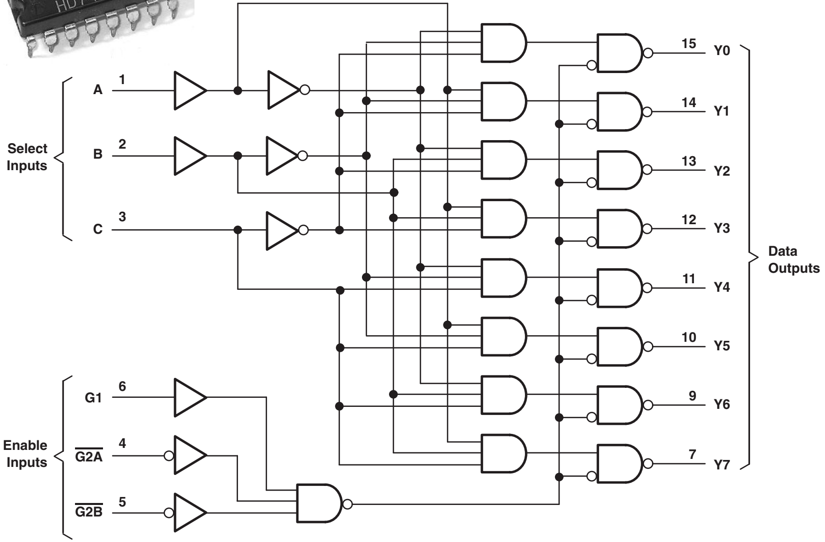
Output: 1-of-2^n one-hot code

2-to-4 in	2-to-4 out
00	0001
01	0010
10	0100
11	1000

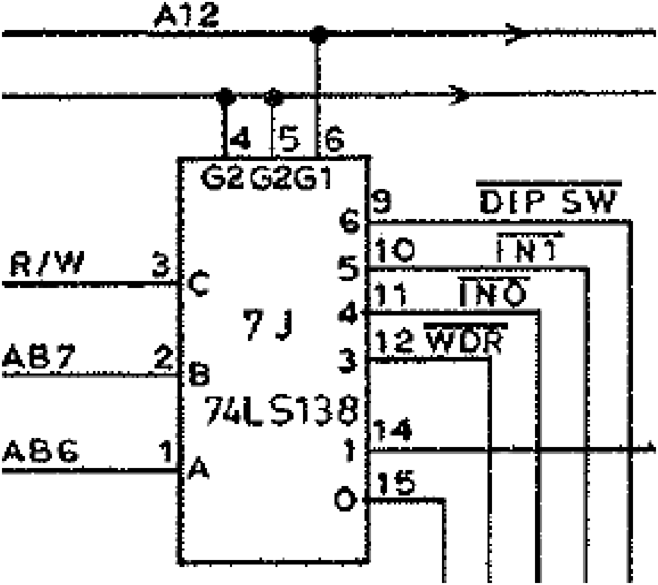
3-to-8 decoder in	3-to-8 decoder out
000	00000001
001	00000010
010	00000100
011	00001000
100	00010000
101	00100000
110	01000000
111	10000000

4-to-16 decoder in	4-to-16 decoder out
0000	0000000000000001
0001	0000000000000010
0010	0000000000000100
0011	0000000000001000
0100	0000000000010000
0101	0000000000100000
0110	0000000001000000
0111	0000000010000000
1000	0000000100000000
1001	0000001000000000
1010	0000010000000000
1011	0000100000000000
1100	0001000000000000
1101	0010000000000000
1110	0100000000000000
1111	1000000000000000

The 74138 3-to-8 Decoder

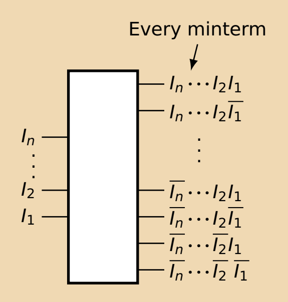


A ’138 Spotted in the Wild

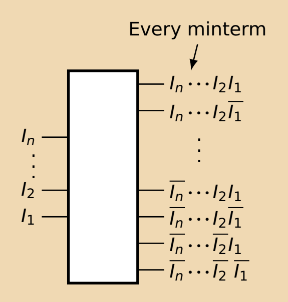


Pac-Man (Midway, 1980)

General n-bit Decoders



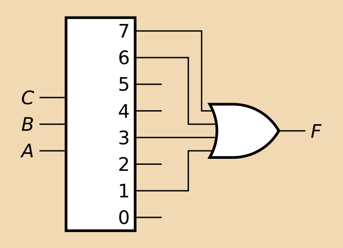
General n-bit Decoders



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

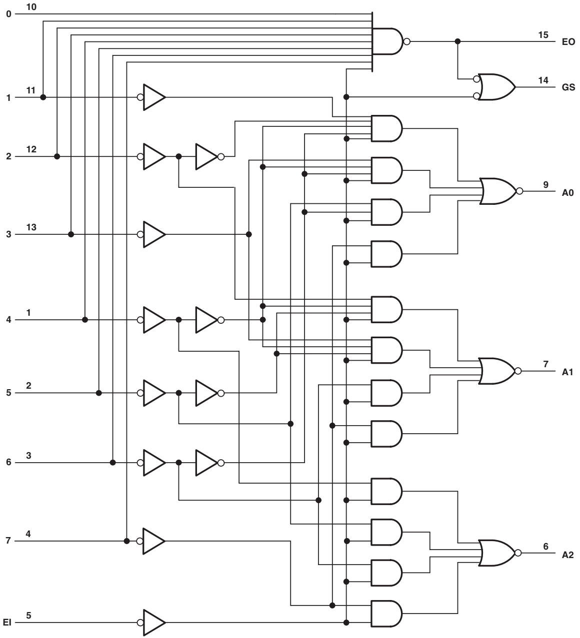


The 74148 Priority Encoder

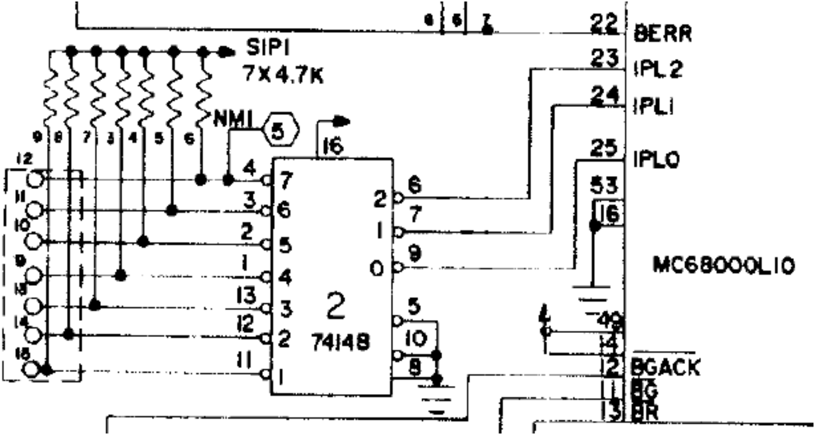
Input: 1-of-2^n

Output: *n*-bit binary number for *highest priority* input

	Inputs	Outputs	
E	01234567	210	G E
1	XXXXXXXX	111	1 1
0	11111111	111	1 0
0	XXXXXXX0	000	0 1
0	XXXXXX01	001	0 1
0	XXXXX011	010	0 1
0	XXXX0111	011	0 1
0	XXX01111	100	0 1
0	XX011111	101	0 1
0	X0111111	110	0 1
0	01111111	111	0 1



A ’148 Spotted in the Wild

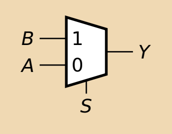


Users would connect wires to interrupt sources; pull-ups quiet unconnected interrupts

OB68K1A Single-board Computer (Omnibyte 1983)

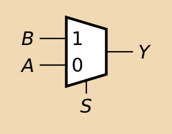
Multiplexers

The Two-Input Multiplexer



S	B	A	Y
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

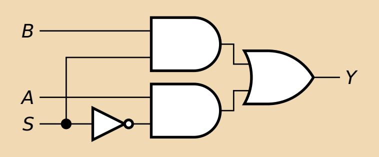
The Two-Input Multiplexer



S	B	A	Y
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

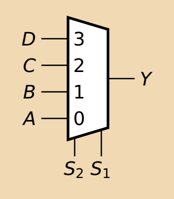
$$0\ \overbrace{1\ \ 1}^{A}\ 0$$
$$S\ \{\ 0\ \ 0\ \underbrace{1\ \ 1}_{B}$$

S	B	A	Y
0	X	0	0
0	X	1	1
1	0	X	0
1	1	X	1

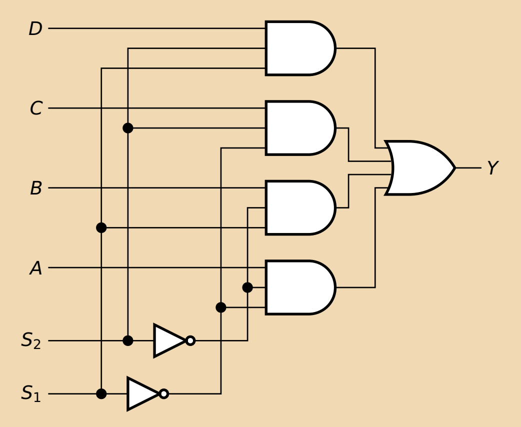


S	Y
0	A
1	B

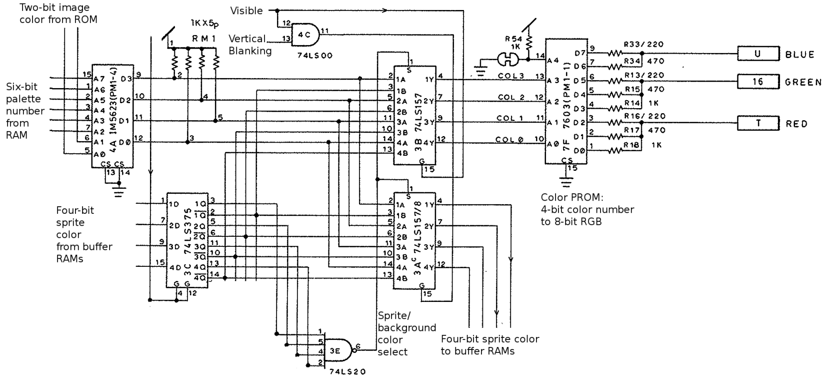
The Four-Input Mux



S_2	S_1	Y
0	0	A
0	1	B
1	0	C
1	1	D



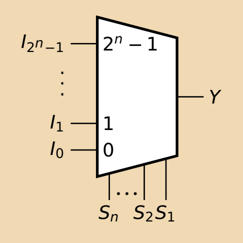
Two-input Muxes in the Wild

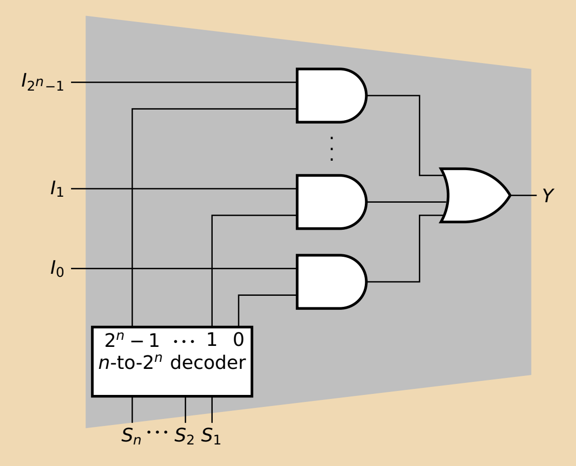


Quad 2-to-1 mux 3B selects color from a sprite or the background

Pac-Man (Midway, 1980)

General 2^n-input muxes





$$
\begin{aligned}
Y = \; & I_0\overline{S_n}\cdots\overline{S_2}\;\overline{S_1} \;+\\
& I_1\overline{S_n}\cdots\overline{S_2}\;S_1 \;+\\
& I_2\overline{S_n}\cdots S_2\;\overline{S_1} \;+\\
& \vdots\\
& I_{2^n-2}S_n\cdots S_2\;\overline{S_1} \;+\\
& I_{2^n-1}S_n\cdots S_2\;S_1
\end{aligned}
$$

Using a Mux to Implement an Arbitrary Function

$$F = A\overline{C} + BC$$

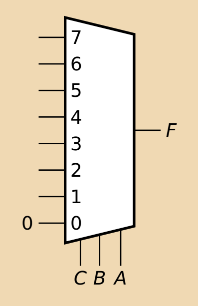
C	B	A	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Using a Mux to Implement an Arbitrary Function

$$F = A\overline{C} + BC$$

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

Apply each value in the truth table:

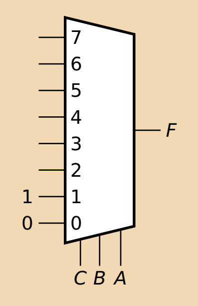


Using a Mux to Implement an Arbitrary Function

$$F = A\overline{C} + BC$$

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

Apply each value in the truth table:

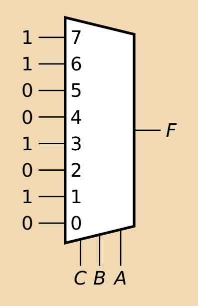


Using a Mux to Implement an Arbitrary Function

$$F = A\overline{C} + BC$$

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

Apply each value in the truth table:



Using a Mux to Implement an Arbitrary Function

$$F = A\overline{C} + BC$$

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

Using a Mux to Implement an Arbitrary Function

$$F = A\overline{C} + BC$$

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

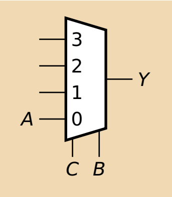
Using a Mux to Implement an Arbitrary Function

$F = A\overline{C} + BC$

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

Can always remove a select and feed in 0, 1, S, or $\overline{S}$.

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



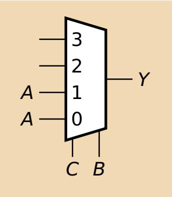
Using a Mux to Implement an Arbitrary Function

$F = A\overline{C} + BC$

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

Can always remove a select and feed in 0, 1, S, or $\overline{S}$.

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



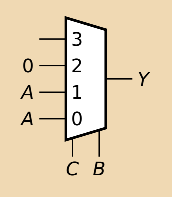
Using a Mux to Implement an Arbitrary Function

$F = A\overline{C} + BC$

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

Can always remove a select and feed in 0, 1, S, or $\overline{S}$.

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



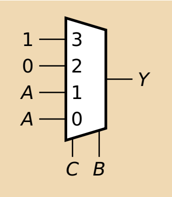
Using a Mux to Implement an Arbitrary Function

$F = A\overline{C} + BC$

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

Can always remove a select and feed in 0, 1, S, or $\overline{S}$.

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



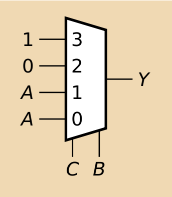
Using a Mux to Implement an Arbitrary Function

$F = A\overline{C} + BC$

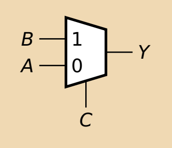
C	B	A	F
0	0	0	0
		1	1
0	1	0	0
		1	1
1	0	0	0
		1	0
1	1	0	1
		1	1

Can always remove a select and feed in 0, 1, S, or $\overline{S}$.

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

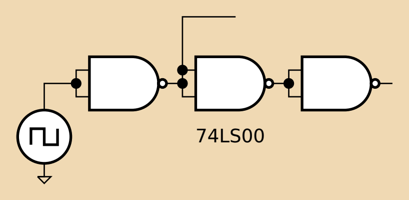


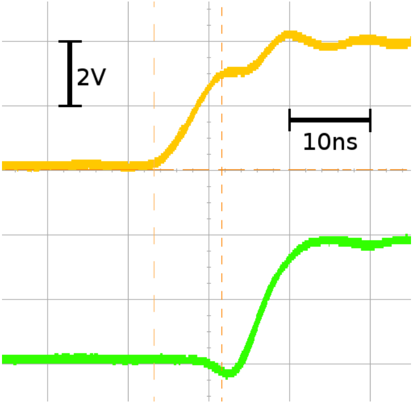
In this case, the function just happens to be a mux: (not always the case!)



Timing

Computation Always Takes Time

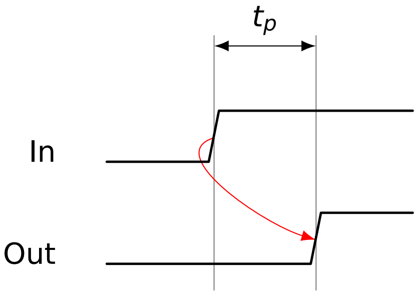




Always a delay between inputs and outputs. Causes:

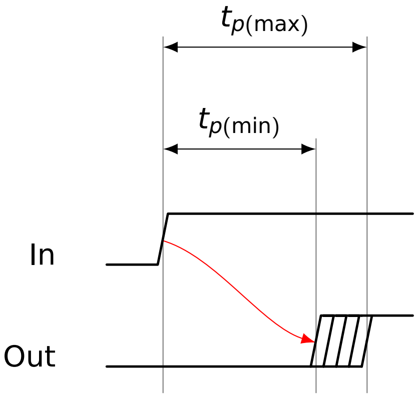
- Limited currents charging capacitance
- The speed of light

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.

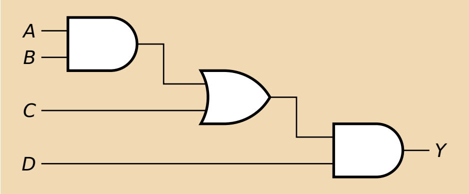
A More Realistic Timing Model



It is difficult to manufacture two gates with the same delay; better to treat delay as a range.

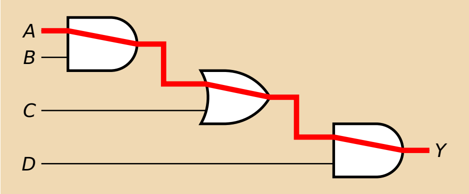
- Each gate has a minimum and maximum propagation delay $t_{p(\text{min})}$ and $t_{p(\text{max})}$.
- Outputs may start changing after $t_{p(\text{min})}$ and stablize no later than $t_{p(\text{min})}$.

Critical Paths and Short Paths



How slow can this be?

Critical Paths and Short Paths

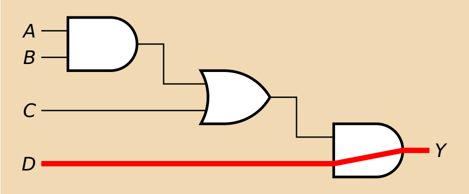


How slow can this be?

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



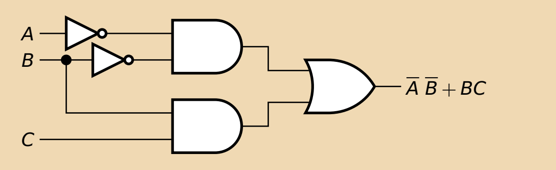
How fast can this be?

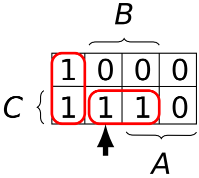
The shortest path has the least possible delay.

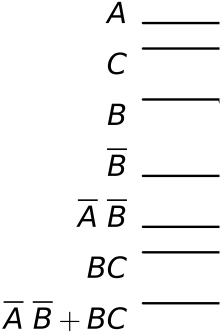
$$t_{p(\text{min})} = t_{p(\text{min, AND})}$$

Glitches

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

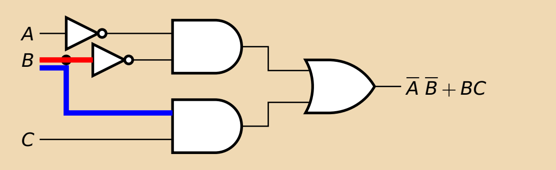


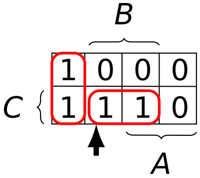


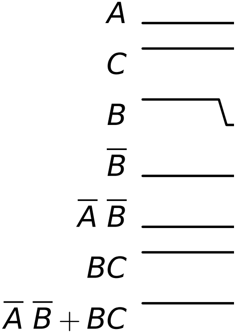


Glitches

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

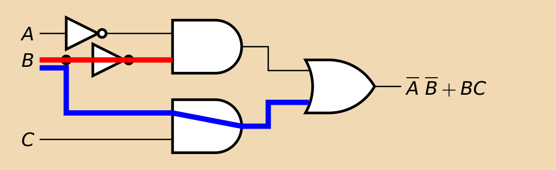


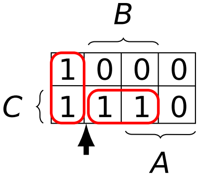


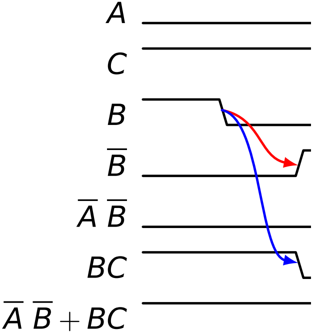


Glitches

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

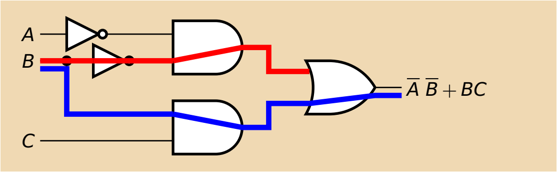


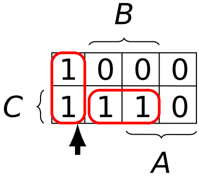


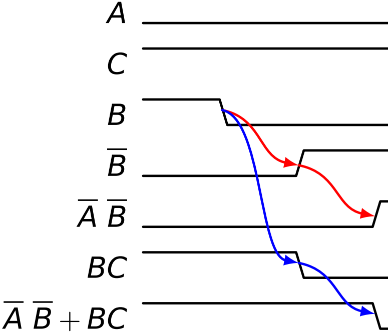


Glitches

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

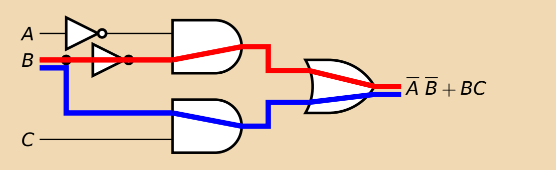




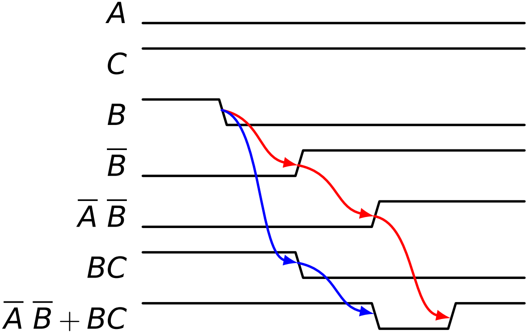


Glitches

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

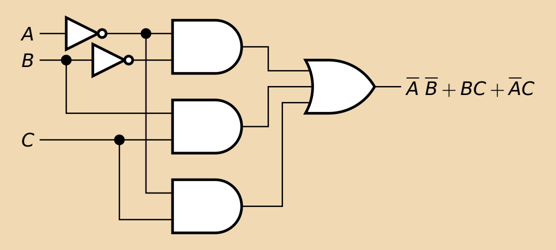


		B	B	
	1	0	0	0
C	1	1	1	0
			A	A



Glitches

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



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

Adding such redundancy only works for single input changes; glitches may be unavoidable when multiple inputs change.

Arithmetic Circuits

Arithmetic: Addition

Adding two one-bit numbers:

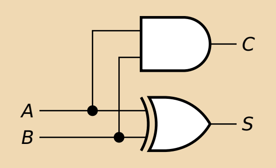
A and *B*

Produces a two-bit result:

C 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

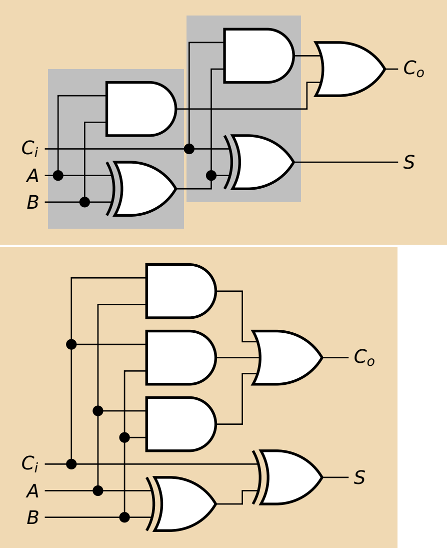
Full Adder

In general, you need to add *three bits*:

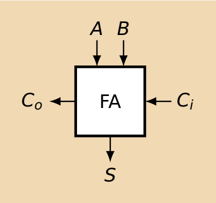
$$\begin{array}{r} \color{red}{111000} \\ 111010 \\ +\quad 11100 \\ \hline 1010110 \end{array}$$

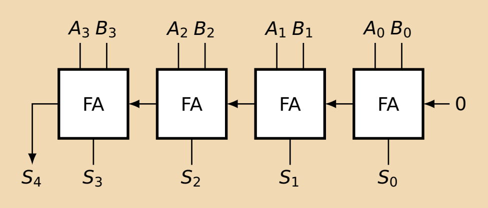
$C_i A B$	$C_o S$
0 0 0	0 0
0 0 1	0 1
0 1 0	0 1
0 1 1	1 0
1 0 0	0 1
1 0 1	1 0
1 1 0	1 0
1 1 1	1 1

$$0 + 0 = 00$$
$$0 + 1 + 0 = 01$$
$$0 + 0 + 1 = 01$$
$$0 + 1 + 1 = 10$$
$$1 + 1 + 1 = 11$$
$$1 + 1 + 0 = 10$$



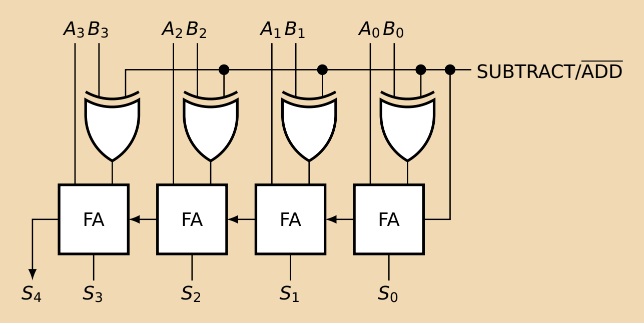
A Four-Bit Ripple-Carry Adder





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?

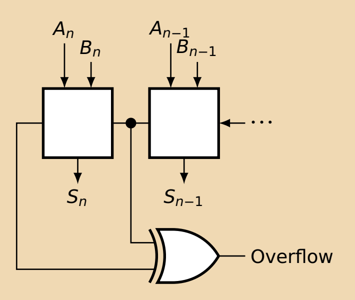
+	−2	−1	0	1
−2	10 10 +10 00			
−1	10 10 +11 01	11 11 +11 10		
0	00 10 +00 10	00 11 +00 11	00 00 +00 00	
1	00 10 +01 11	11 11 +01 00	00 00 +01 01	01 01 +01 10

Overflow in Two's-Complement Representation

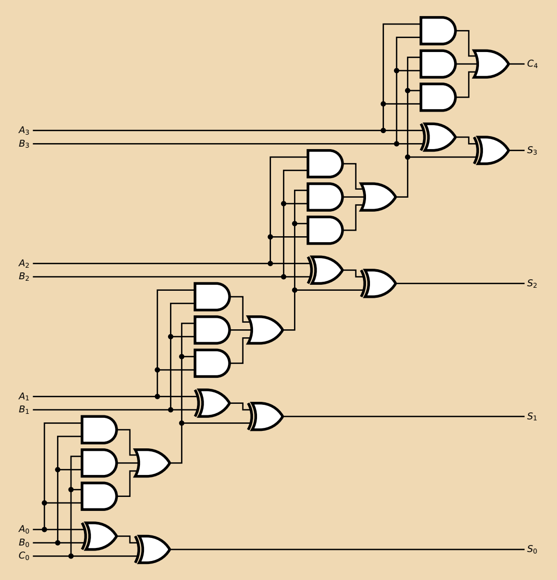
When is the result too positive or too negative?

+	−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 2*n*.

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

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

For bit i,

$$
\begin{aligned}
C_{i+1} &= A_iB_i + A_iC_i + B_iC_i \\
&= A_iB_i + C_i(A_i + B_i) \\
&= G_i + C_iP_i
\end{aligned}
$$

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

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

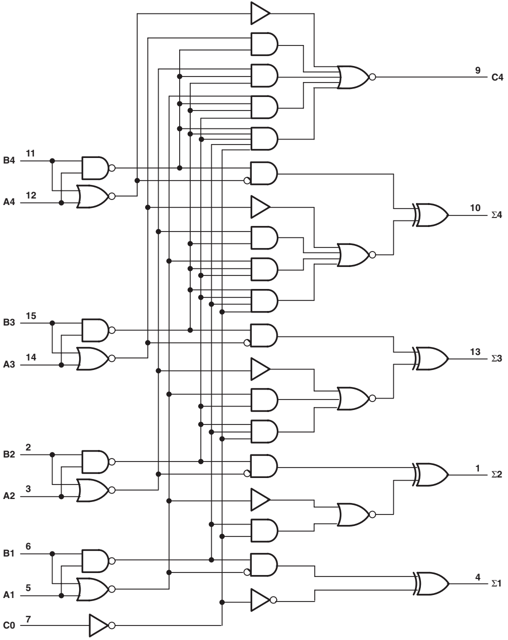
Carry Lookahead Adder

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



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 $O(\log_2 n)$.