

Boolean Logic

- Axioms and Simplification
- Implicants, Minterms, etc.
- De Morgan's Theorem
- Karnaugh Maps

Combinational Logic

- Decoders
- Multiplexers
- Timing and Glitches
- Adders

Sequential Logic

- Bistables; SR Latch; D Latch
- D Flip-Flops
- Synchronous Digital Logic
- Shift Registers
- Counters

Finite State Machines

- Moore and Mealy Machines
- The Snail Example
- The TLC: One-Hot Encoding

CMOS Logic Gates

- The Inverter
- The CMOS NAND Gate
- The CMOS NOR Gate
- A CMOS AND-OR-INVERT Gate
- General Static CMOS Gates

Memories

- ROMs, EPROMs, and FLASH
- The SRAM Cell
- Dynamic RAM Cell
- PLAs and FPGAs

Datapaths and Control

- Datapaths
- Timing and Control FSM

Fundamentals of Computer Systems

Review for the Midterm

Stephen A. Edwards

Columbia University

Fall 2012



The Midterm

75 minutes

4–5 problems

Closed book

Simple calculators are OK, but unnecessary

One double-sided 8.5 × 11" sheet of your own notes

Anything discussed in class is fair game

Much like homework assignments

Problems will range from easy to difficult; do the easy ones first.

Historical developments & trivia will *not* be on the test.

The Axioms of (Any) Boolean Algebra

A Boolean Algebra consists of

A set of values A

An “and” operator \wedge

An “or” operator \vee

A “not” operator \neg

A “false” value $0 \in A$

A “true” value $1 \in A$

Axioms

$$a \vee b = b \vee a$$

$$a \wedge b = b \wedge a$$

$$a \vee (b \vee c) = (a \vee b) \vee c$$

$$a \wedge (b \wedge c) = (a \wedge b) \wedge c$$

$$a \vee (a \wedge b) = a$$

$$a \wedge (a \vee b) = a$$

$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$$

$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$$

$$a \vee \neg a = 1$$

$$a \wedge \neg a = 0$$

We will use the first non-trivial Boolean Algebra:

$A = \{0, 1\}$. This adds the law of excluded middle: if

$a \neq 0$ then $a = 1$ and if $a \neq 1$ then $a = 0$.

Simplifying a Boolean Expression

“You are a New Yorker if you were born in New York or were not born in New York and lived here ten years.”

Axioms

$$a \vee b = b \vee a$$

$$a \wedge b = b \wedge a$$

$$a \vee (b \vee c) = (a \vee b) \vee c$$

$$a \wedge (b \wedge c) = (a \wedge b) \wedge c$$

$$a \vee (a \wedge b) = a$$

$$a \wedge (a \vee b) = a$$

$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$$

$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$$

$$a \vee \neg a = 1$$

$$a \wedge \neg a = 0$$

$$x \vee ((\neg x) \wedge y)$$

Lemma:

$$\begin{aligned}x \wedge 1 &= x \wedge (x \vee \neg x) \\ &= x \wedge (x \vee y) \text{ if } y = \neg x \\ &= x\end{aligned}$$

Simplifying a Boolean Expression

“You are a New Yorker if you were born in New York or were not born in New York and lived here ten years.”

$$\begin{aligned}x \vee ((\neg x) \wedge y) \\ = (x \vee (\neg x)) \wedge (x \vee y)\end{aligned}$$

Axioms

$$\begin{aligned}a \vee b &= b \vee a \\ a \wedge b &= b \wedge a \\ a \vee (b \vee c) &= (a \vee b) \vee c \\ a \wedge (b \wedge c) &= (a \wedge b) \wedge c \\ a \vee (a \wedge b) &= a \\ a \wedge (a \vee b) &= a \\ a \wedge (b \vee c) &= (a \wedge b) \vee (a \wedge c) \\ a \vee (b \wedge c) &= (a \vee b) \wedge (a \vee c) \\ a \vee \neg a &= 1 \\ a \wedge \neg a &= 0\end{aligned}$$

Lemma:

$$\begin{aligned}x \wedge 1 &= x \wedge (x \vee \neg x) \\ &= x \wedge (x \vee y) \text{ if } y = \neg x \\ &= x\end{aligned}$$

Simplifying a Boolean Expression

“You are a New Yorker if you were born in New York or were not born in New York and lived here ten years.”

$$\begin{aligned}x \vee ((\neg x) \wedge y) \\ &= (x \vee (\neg x)) \wedge (x \vee y) \\ &= 1 \wedge (x \vee y)\end{aligned}$$

Axioms

$$\begin{aligned}a \vee b &= b \vee a \\ a \wedge b &= b \wedge a \\ a \vee (b \vee c) &= (a \vee b) \vee c \\ a \wedge (b \wedge c) &= (a \wedge b) \wedge c \\ a \vee (a \wedge b) &= a \\ a \wedge (a \vee b) &= a \\ a \wedge (b \vee c) &= (a \wedge b) \vee (a \wedge c) \\ a \vee (b \wedge c) &= (a \vee b) \wedge (a \vee c) \\ a \vee \neg a &= 1 \\ a \wedge \neg a &= 0\end{aligned}$$

Lemma:

$$\begin{aligned}x \wedge 1 &= x \wedge (x \vee \neg x) \\ &= x \wedge (x \vee y) \text{ if } y = \neg x \\ &= x\end{aligned}$$

Simplifying a Boolean Expression

“You are a New Yorker if you were born in New York or were not born in New York and lived here ten years.”

$$\begin{aligned}x \vee ((\neg x) \wedge y) \\&= (x \vee (\neg x)) \wedge (x \vee y) \\&= 1 \wedge (x \vee y) \\&= x \vee y\end{aligned}$$





Axioms

$$\begin{aligned}a \vee b &= b \vee a \\a \wedge b &= b \wedge a \\a \vee (b \vee c) &= (a \vee b) \vee c \\a \wedge (b \wedge c) &= (a \wedge b) \wedge c \\a \vee (a \wedge b) &= a \\a \wedge (a \vee b) &= a \\a \wedge (b \vee c) &= (a \wedge b) \vee (a \wedge c) \\a \vee (b \wedge c) &= (a \vee b) \wedge (a \vee c) \\a \vee \neg a &= 1 \\a \wedge \neg a &= 0\end{aligned}$$

Lemma:

$$\begin{aligned}x \wedge 1 &= x \wedge (x \vee \neg x) \\&= x \wedge (x \vee y) \text{ if } y = \neg x \\&= x\end{aligned}$$

Alternate Notations for Boolean Logic

Operator	Math	Engineer	Schematic
Copy	x	X	x — or x —  — x
Complement	$\neg x$	\bar{X}	x —  — \bar{x}
AND	$x \wedge y$	XY or $X \cdot Y$	X Y —  — XY
OR	$x \vee y$	$X + Y$	X Y —  — $X + Y$

Definitions

Literal: a Boolean variable or its complement

E.g., X \bar{X} Y \bar{Y}

Implicant: A product of literals

E.g., X XY $X\bar{Y}Z$

Minterm: An implicant with each variable once

E.g., $X\bar{Y}Z$ XYZ $\bar{X}\bar{Y}Z$

Maxterm: A sum of literals with each variable once

E.g., $X + \bar{Y} + Z$ $X + Y + Z$ $\bar{X} + \bar{Y} + Z$

Sum-of-minterms and Product-of-maxterms

Two mechanical ways to translate a function's truth table into an expression:

X	Y	Minterm	Maxterm	F
0	0	$\bar{X}\bar{Y}$	$X+Y$	0
0	1	$\bar{X}Y$	$X+\bar{Y}$	1
1	0	$X\bar{Y}$	$\bar{X}+Y$	1
1	1	XY	$\bar{X}+\bar{Y}$	0

The sum of the minterms where the function is 1:

$$F = \bar{X}Y + X\bar{Y}$$

The product of the maxterms where the function is 0:

$$F = (X+Y)(\bar{X}+\bar{Y})$$

Minterms and Maxterms: Another Example

The minterm and maxterm representation of functions may look very different:

X	Y	Minterm	Maxterm	F
0	0	$\bar{X}\bar{Y}$	$X+Y$	0
0	1	$\bar{X}Y$	$X+\bar{Y}$	1
1	0	$X\bar{Y}$	$\bar{X}+Y$	1
1	1	XY	$\bar{X}+\bar{Y}$	1

The sum of the minterms where the function is 1:

$$F = \bar{X}Y + X\bar{Y} + XY$$

The product of the maxterms where the function is 0:

$$F = X + Y$$

The Menagerie of Gates

Buffer



0		0
1		1

Inverter



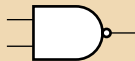
0		1
1		0

AND



.		0	1
0		0	0
1		0	1

NAND



.		0	1
0		1	1
1		1	0

OR



+		0	1
0		0	1
1		1	1

NOR



$\bar{+}$		0	1
0		1	0
1		0	0

XOR



\oplus		0	1
0		0	1
1		1	0

XNOR



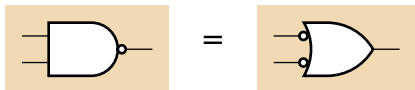
$\bar{\oplus}$		0	1
0		1	0
1		0	1

De Morgan's Theorem

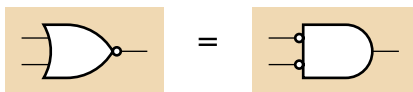
$$\neg(a \vee b) = (\neg a) \wedge (\neg b)$$

$$\neg(a \wedge b) = (\neg a) \vee (\neg b)$$

$$\overline{AB} = \overline{A} + \overline{B}$$



$$\overline{A + B} = \overline{A} \cdot \overline{B}$$



Karnaugh Map for Seg. a

W	X	Y	Z	a
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	0

		Z			
		1	0	1	1
X	{	0	1	1	1
		X	X	0	X
		1	1	X	X
				Y	
				W	

The Karnaugh Map Sum-of-Products Challenge

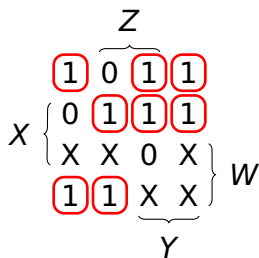
Cover all the 1's and none of the 0's using **as few literals** (gate inputs) as possible.

Few, large rectangles are good.

Covering X's is optional.

Karnaugh Map for Seg. a

W	X	Y	Z	a
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	0



The minterm solution: cover each 1 with a single implicant.

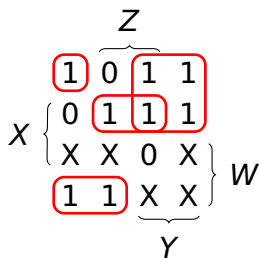
$$\begin{aligned}
 a = & \overline{W}\overline{X}\overline{Y}\overline{Z} + \overline{W}\overline{X}YZ + \overline{W}\overline{X}Y\overline{Z} + \\
 & \overline{W}X\overline{Y}\overline{Z} + \overline{W}X\overline{Y}Z + \overline{W}XY\overline{Z} + \\
 & W\overline{X}\overline{Y}\overline{Z} + W\overline{X}\overline{Y}Z
 \end{aligned}$$

$8 \times 4 = 32$ literals

4 inv + 8 AND4 + 1 OR8

Karnaugh Map for Seg. a

W	X	Y	Z	a
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	0



Merging implicants helps

Recall the distributive law:

$$AB + AC = A(B + C)$$

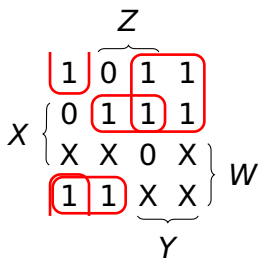
$$a = \overline{W}\overline{X}\overline{Y}\overline{Z} + \overline{W}Y + \overline{W}XZ + W\overline{X}\overline{Y}$$

$$4 + 2 + 3 + 3 = 12 \text{ literals}$$

$$4 \text{ inv} + 1 \text{ AND}_4 + 2 \text{ AND}_3 + 1 \text{ AND}_2 + 1 \text{ OR}_4$$

Karnaugh Map for Seg. a

W	X	Y	Z	a
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	0



Missed one: Remember this is actually a torus.

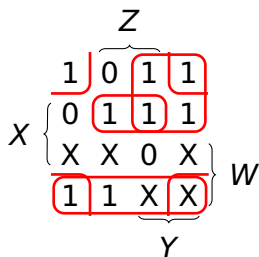
$$a = \overline{X}\overline{Y}\overline{Z} + \overline{W}Y + \overline{W}XZ + W\overline{X}\overline{Y}$$

3 + 2 + 3 + 3 = 11 literals

4 inv + 3 AND3 + 1 AND2 + 1 OR4

Karnaugh Map for Seg. a

W	X	Y	Z	a
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	0



Taking don't-cares into account, we can enlarge two implicants:

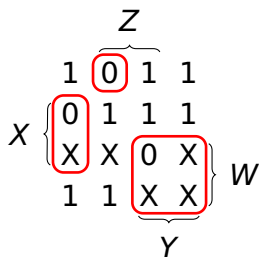
$$a = \bar{X}\bar{Z} + \bar{W}Y + \bar{W}XZ + W\bar{X}$$

2 + 2 + 3 + 2 = 9 literals

3 inv + 1 AND3 + 3 AND2 + 1 OR4

Karnaugh Map for Seg. a

W	X	Y	Z	a
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	0



Can also compute the complement of the function and invert the result.

Covering the 0's instead of the 1's:

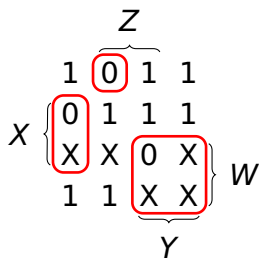
$$\bar{a} = \bar{W}\bar{X}\bar{Y}Z + X\bar{Y}\bar{Z} + WY$$

4 + 3 + 2 = 9 literals

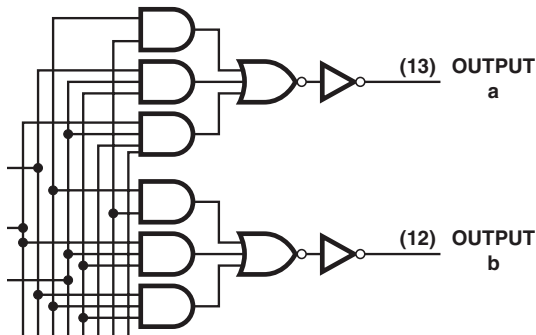
5 inv + 1 AND4 + 1 AND3 + 1 AND2
+ 1 OR3

Karnaugh Map for Seg. a

W	X	Y	Z	a
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	0



To display the score, PONG used a TTL chip with this solution in it:



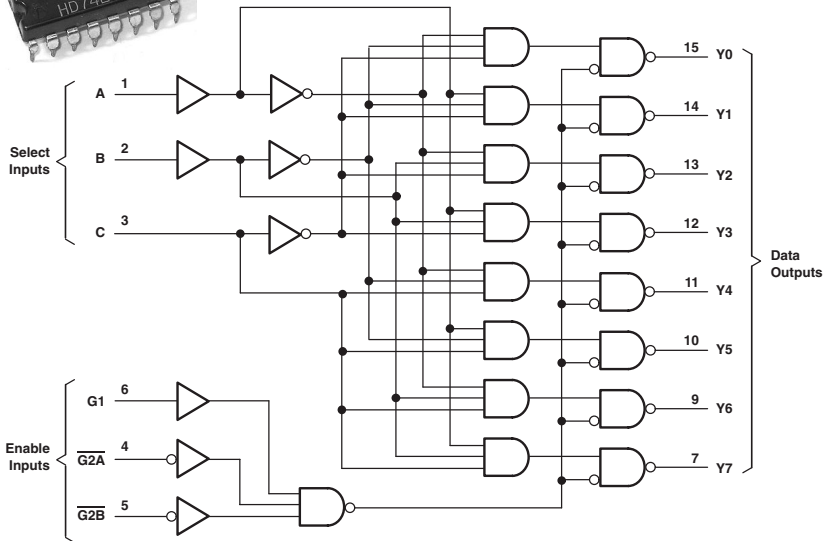
Decoders

Input: n -bit binary number

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

2-to-4		3-to-8 decoder		4-to-16 decoder	
in	out	in	out	in	out
00	0001	000	00000001	0000	0000000000000001
01	0010	001	00000010	0001	0000000000000010
10	0100	010	00000100	0010	0000000000000100
11	1000	011	00001000	0011	0000000000001000
		100	00010000	0100	0000000000010000
		101	00100000	0101	0000000001000000
		110	01000000	0110	0000000010000000
		111	10000000	0111	0000000100000000
				1000	0000000100000000
				1001	0000001000000000
				1010	0000010000000000
				1011	0000100000000000
				1100	0001000000000000
				1101	0010000000000000
				1110	0100000000000000
				1111	1000000000000000

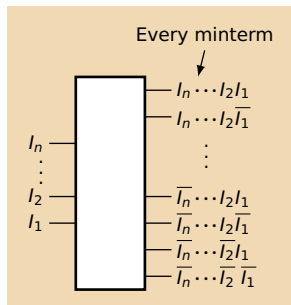
The 74138 3-to-8 Decoder



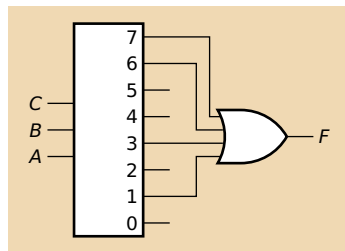
General n -bit Decoders

Implementing a function with a decoder:

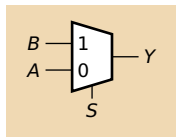
E.g., $F = A\bar{C} + BC$



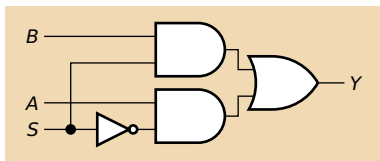
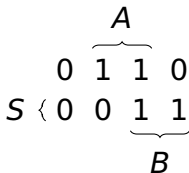
C	B	A	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



The Two-Input Multiplexer



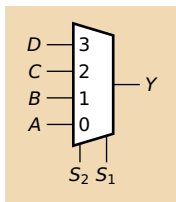
S	B	A	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



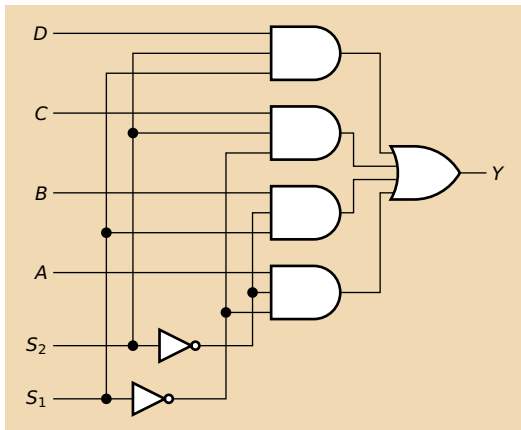
S	B	A	Y
0	X	0	0
0	X	1	1
1	0	X	0
1	1	X	1

S	Y
0	A
1	B

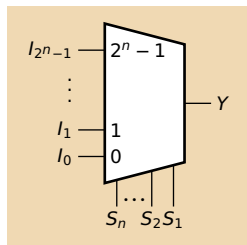
The Four-Input Mux



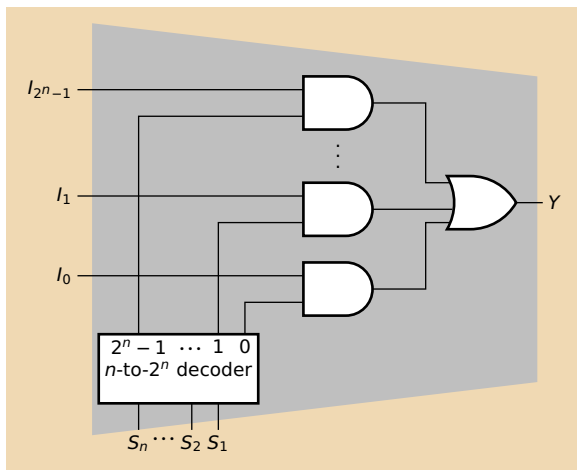
S_2	S_1	Y
0	0	A
0	1	B
1	0	C
1	1	D



General 2^n -input muxes



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



Using a Mux to Implement an Arbitrary Function

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

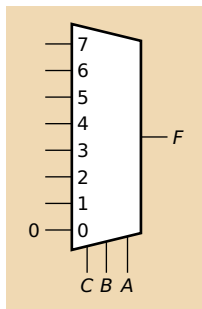
<i>C</i>	<i>B</i>	<i>A</i>	<i>F</i>
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Using a Mux to Implement an Arbitrary Function

Apply each value in the truth table:

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

<i>C</i>	<i>B</i>	<i>A</i>	<i>F</i>
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

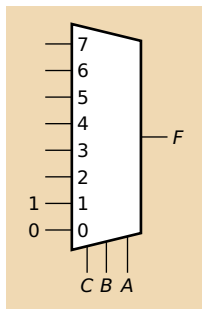


Using a Mux to Implement an Arbitrary Function

Apply each value in the truth table:

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

<i>C</i>	<i>B</i>	<i>A</i>	<i>F</i>
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

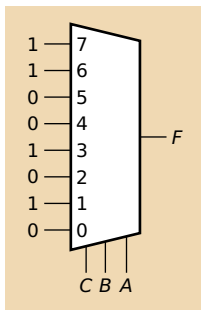


Using a Mux to Implement an Arbitrary Function

Apply each value in the truth table:

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

<i>C</i>	<i>B</i>	<i>A</i>	<i>F</i>
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

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

<i>C</i>	<i>B</i>	<i>A</i>	<i>F</i>
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

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

<i>C</i>	<i>B</i>	<i>A</i>	<i>F</i>
0	0	0	0
		1	1
0	1	0	0
		1	1
1	0	0	0
		1	0
1	1	0	1
		1	1

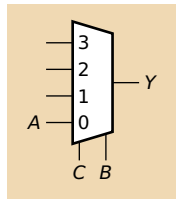
Using a Mux to Implement an Arbitrary Function

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

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 \bar{S} .

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



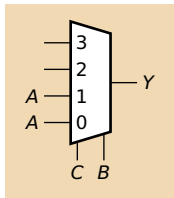
Using a Mux to Implement an Arbitrary Function

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

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 \bar{S} .

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



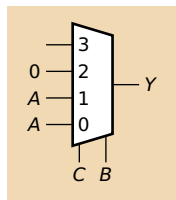
Using a Mux to Implement an Arbitrary Function

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

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 \bar{S} .

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



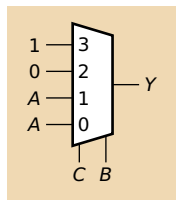
Using a Mux to Implement an Arbitrary Function

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

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 \bar{S} .

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



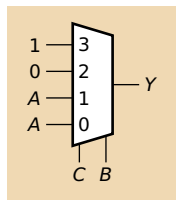
Using a Mux to Implement an Arbitrary Function

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

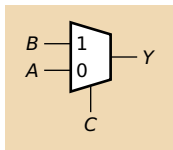
<i>C</i>	<i>B</i>	<i>A</i>	<i>F</i>
0	0	0	0
		1	1
0	1	0	0
		1	1
1	0	0	0
		1	0
1	1	0	1
		1	1

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

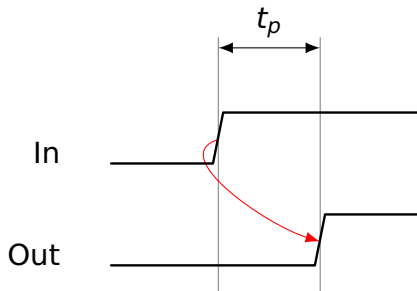
<i>C</i>	<i>B</i>	<i>F</i>
0	0	<i>A</i>
0	1	<i>A</i>
1	0	<i>B</i>
1	1	<i>B</i>



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

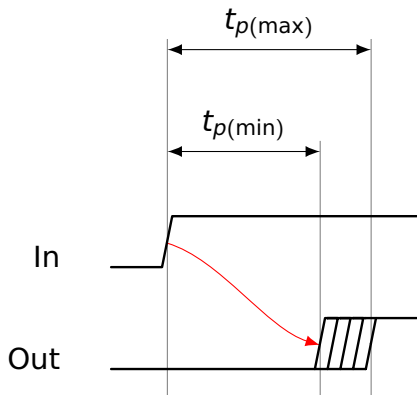


The Simplest Timing Model



- ▶ 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.

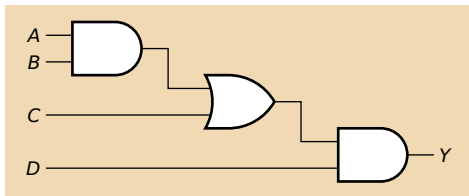
A More Realistic Timing Model



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

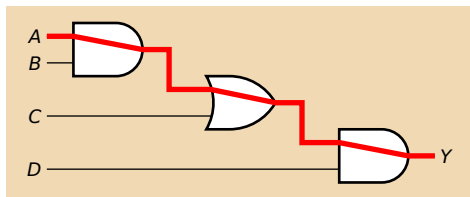
- ▶ Each gate has a minimum and maximum propagation delay $t_{p(\min)}$ and $t_{p(\max)}$.
- ▶ Outputs may start changing after $t_{p(\min)}$ and stabilize no later than $t_{p(\max)}$.

Critical Paths and Short Paths



How slow can this be?

Critical Paths and Short Paths

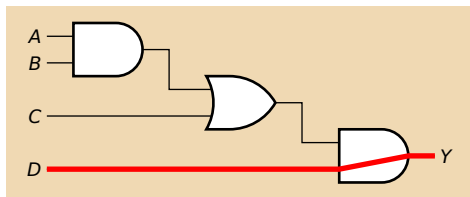


How slow can this be?

The **critical path** has the longest possible delay.

$$t_{p(\max)} = t_{p(\max, \text{AND})} + t_{p(\max, \text{OR})} + t_{p(\max, \text{AND})}$$

Critical Paths and Short Paths



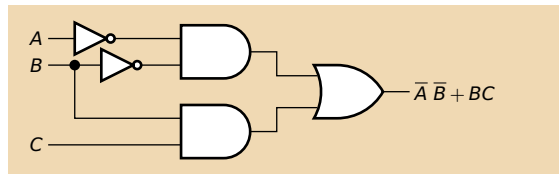
How fast can this be?

The **shortest path** has the least possible delay.

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

Glitches

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



	B			
	1	0	0	0
C	1	1	1	0
		A		

A _____

C _____

B _____

\bar{B} _____

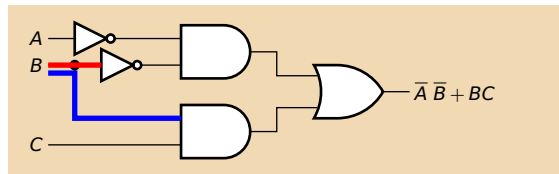
$\bar{A}\bar{B}$ _____

BC _____

$\bar{A}\bar{B} + BC$ _____

Glitches

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



	B			
	1	0	0	0
C	1	1	1	0
	A			

A _____

C _____

B _____

\bar{B} _____

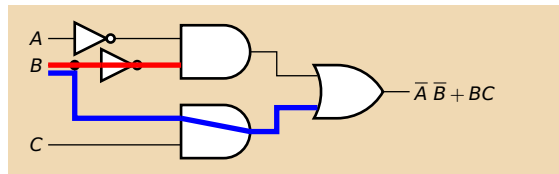
$\bar{A}\bar{B}$ _____

BC _____

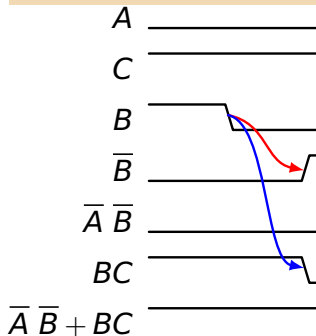
$\bar{A}\bar{B} + BC$ _____

Glitches

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

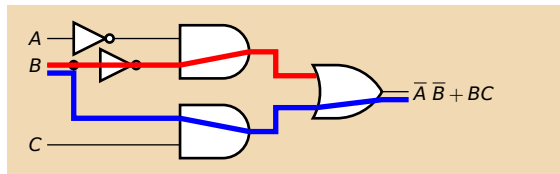


	B			
	1	0	0	0
C	1	1	1	0
		A		

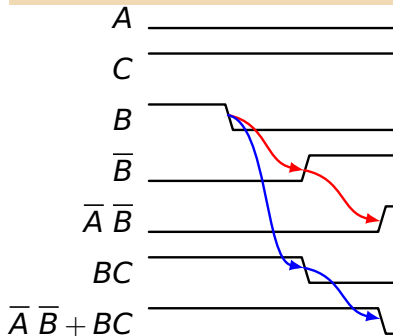


Glitches

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

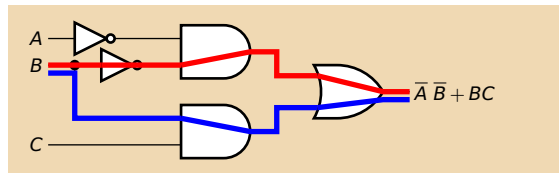


	B			
	1	0	0	0
C	1	1	1	0
	A			

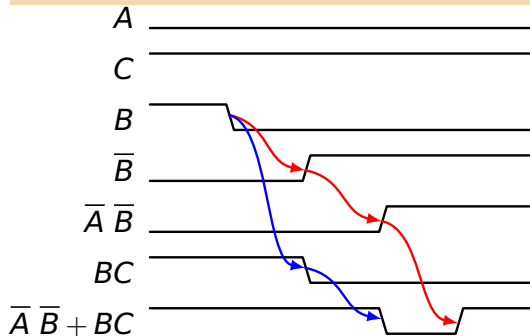


Glitches

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

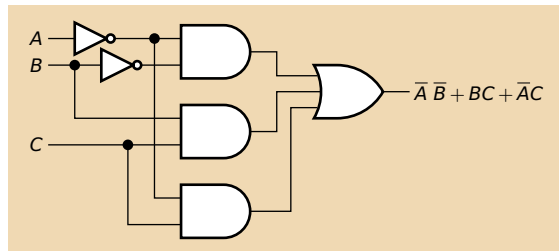


	<i>B</i>			
	1	0	0	0
<i>C</i>	1	1	1	0
	<i>A</i>			



Glitches

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



	B			
	┌───┴───┐			
	1	0	0	0
C	1	1	1	0
	└───┬───┘			
		A		

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

Arithmetic: Addition

Adding two one-bit numbers:

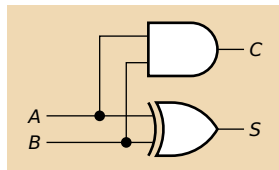
A and B

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



Male Adder

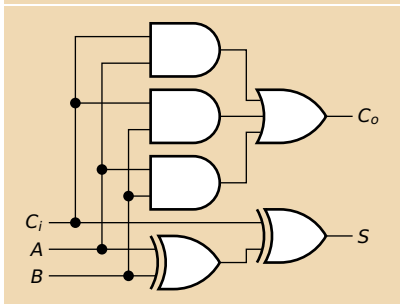
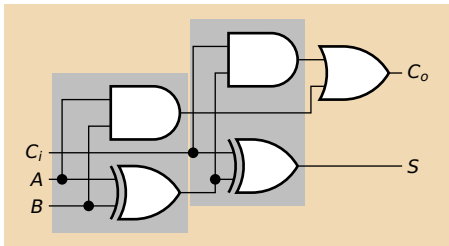
Full Adder

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

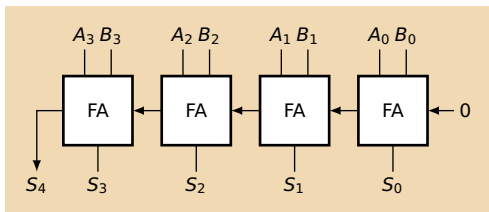
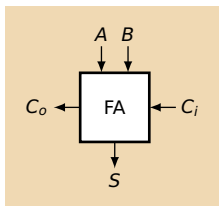
$$\begin{array}{r} 111000 \\ 111010 \\ + 11100 \\ \hline 1010110 \end{array}$$

$$\begin{array}{l} 0 + 0 = 00 \\ 0 + 1 + 0 = 01 \\ 0 + 0 + 1 = 01 \\ 0 + 1 + 1 = 10 \\ 1 + 1 + 1 = 11 \\ 1 + 1 + 0 = 10 \end{array}$$

C_iAB	C_oS
000	00
001	01
010	01
011	10
100	01
101	10
110	10
111	11

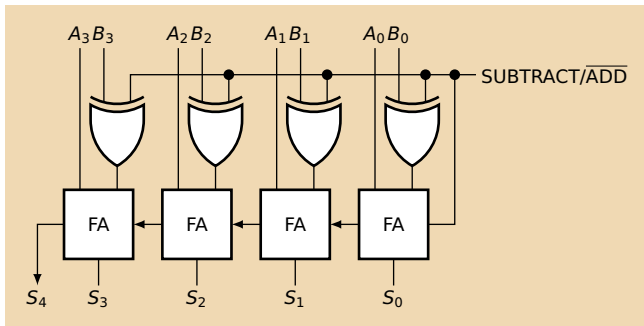


A Four-Bit Ripple-Carry Adder



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?

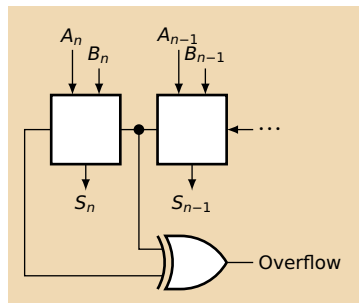
+	-2	-1	0	1
-2	$\begin{array}{r} 10 \\ 10 \\ +10 \\ \hline 00 \end{array}$			
-1	$\begin{array}{r} 10 \\ 10 \\ +11 \\ \hline 01 \end{array}$	$\begin{array}{r} 11 \\ 11 \\ +11 \\ \hline 10 \end{array}$		
0	$\begin{array}{r} 00 \\ 10 \\ +00 \\ \hline 10 \end{array}$	$\begin{array}{r} 00 \\ 11 \\ +00 \\ \hline 11 \end{array}$	$\begin{array}{r} 00 \\ 00 \\ +00 \\ \hline 00 \end{array}$	
1	$\begin{array}{r} 00 \\ 10 \\ +01 \\ \hline 11 \end{array}$	$\begin{array}{r} 11 \\ 11 \\ +01 \\ \hline 00 \end{array}$	$\begin{array}{r} 00 \\ 00 \\ +01 \\ \hline 01 \end{array}$	$\begin{array}{r} 01 \\ 01 \\ +01 \\ \hline 10 \end{array}$

Overflow in Two's-Complement Representation

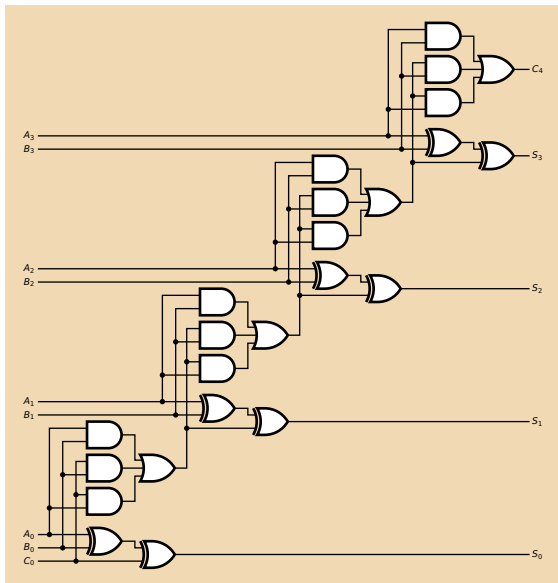
When is the result too positive or too negative?

	+	-2	-1	0	1
-2		$\begin{array}{r} 10 \\ 10 \\ +10 \\ \hline 00 \end{array}$ ✗			
-1		$\begin{array}{r} 10 \\ 10 \\ +11 \\ \hline 01 \end{array}$ ✗	$\begin{array}{r} 11 \\ 11 \\ +11 \\ \hline 10 \end{array}$		
0		$\begin{array}{r} 00 \\ 10 \\ +00 \\ \hline 10 \end{array}$	$\begin{array}{r} 00 \\ 11 \\ +00 \\ \hline 11 \end{array}$	$\begin{array}{r} 00 \\ 00 \\ +00 \\ \hline 00 \end{array}$	
1		$\begin{array}{r} 00 \\ 10 \\ +01 \\ \hline 11 \end{array}$	$\begin{array}{r} 11 \\ 11 \\ +01 \\ \hline 00 \end{array}$	$\begin{array}{r} 00 \\ 00 \\ +01 \\ \hline 01 \end{array}$	$\begin{array}{r} 01 \\ 01 \\ +01 \\ \hline 10 \end{array}$ ✗

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	
C	{	0	1	1	1
		B			

For bit i ,

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

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

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:

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

$$C_1 = G_0 + C_0 P_0$$

$$C_2 = G_1 + C_1 P_1$$

$$= G_1 + (G_0 + C_0 P_0) P_1$$

$$= G_1 + G_0 P_1 + C_0 P_0 P_1$$

$$C_3 = G_2 + C_2 P_2$$

$$= G_2 + (G_1 + G_0 P_1 + C_0 P_0 P_1) P_2$$

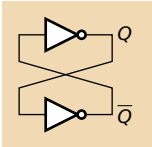
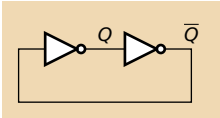
$$= G_2 + G_1 P_2 + G_0 P_1 P_2 + C_0 P_0 P_1 P_2$$

$$C_4 = G_3 + C_3 P_3$$

$$= G_3 + (G_2 + G_1 P_2 + G_0 P_1 P_2 + C_0 P_0 P_1 P_2) P_3$$

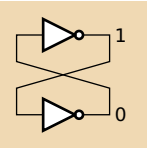
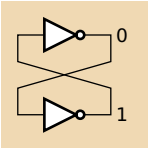
$$= G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3 + C_0 P_0 P_1 P_2 P_3$$

Bistable Elements

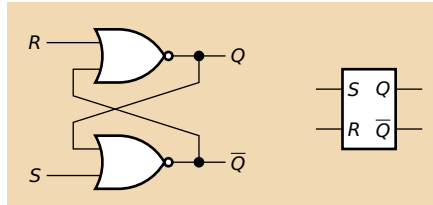


Equivalent circuits; right is more traditional.

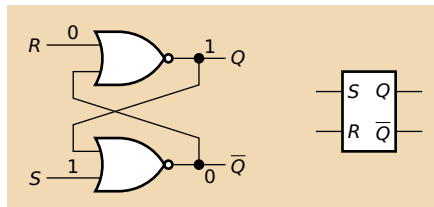
Two stable states:



SR Latch

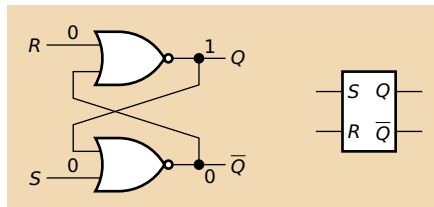


SR Latch



R —
 S —
 Q — Set
 \bar{Q} —

SR Latch



R _____

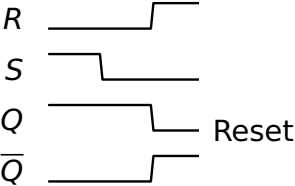
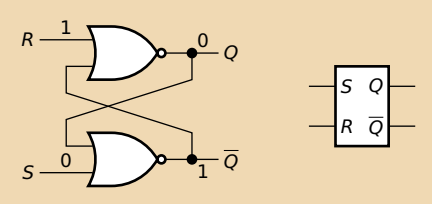
S _____

Q _____

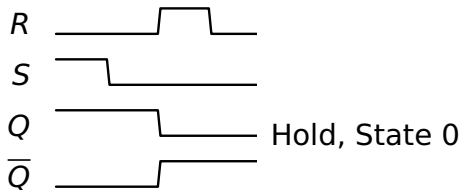
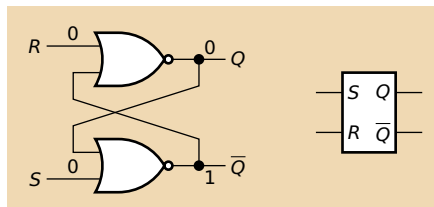
\bar{Q} _____

Hold, State 1

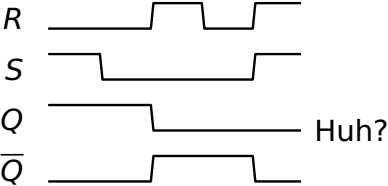
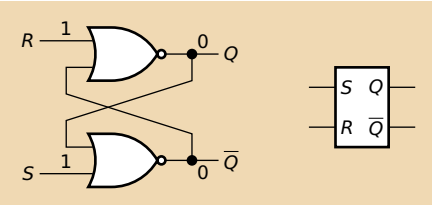
SR Latch



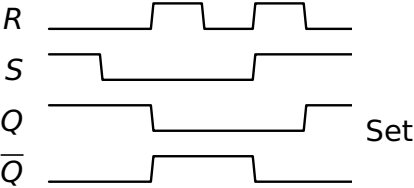
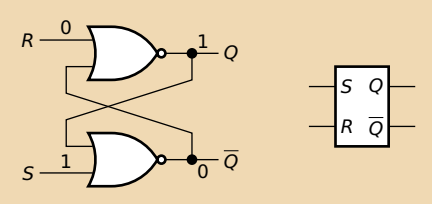
SR Latch



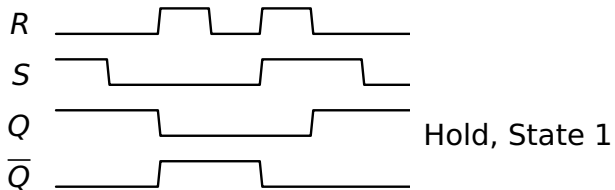
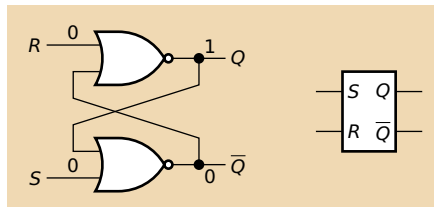
SR Latch



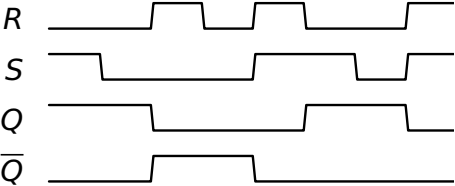
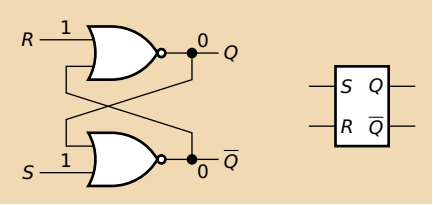
SR Latch



SR Latch

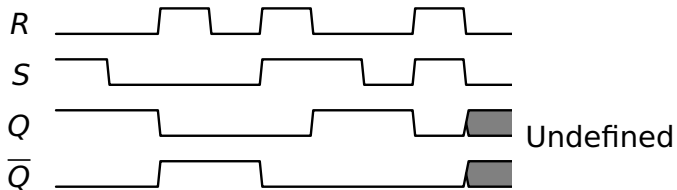
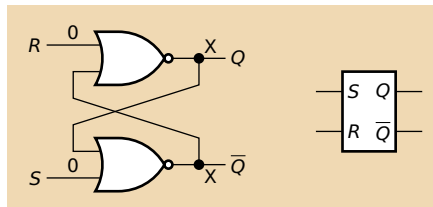


SR Latch

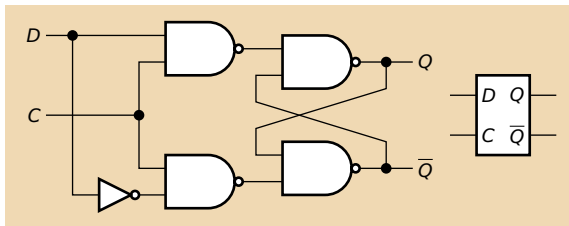


Huh?

SR Latch

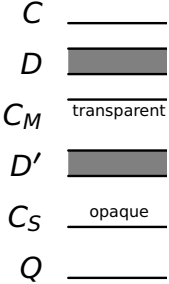
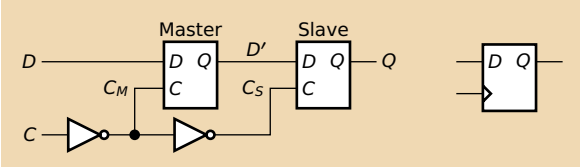


D Latch

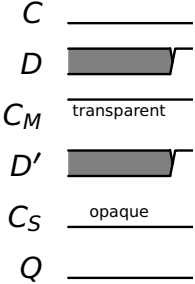
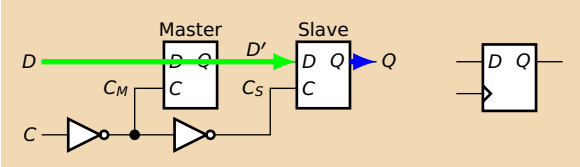


inputs		outputs	
C	D	Q	\bar{Q}
0	X	Q	\bar{Q}
1	0	0	1
1	1	1	0

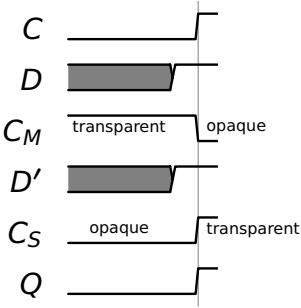
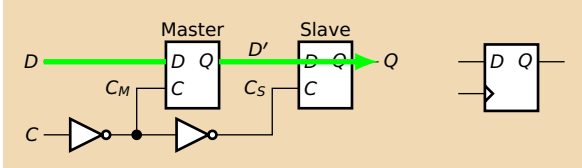
Positive-Edge-Triggered D Flip-Flop



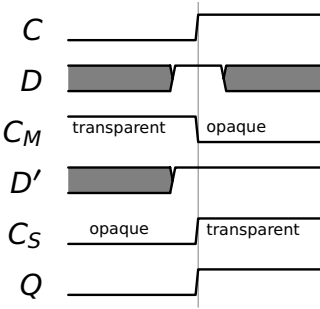
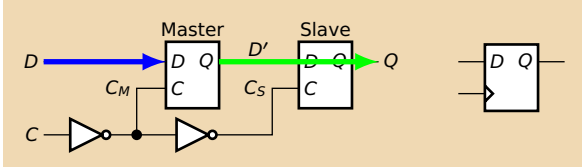
Positive-Edge-Triggered D Flip-Flop



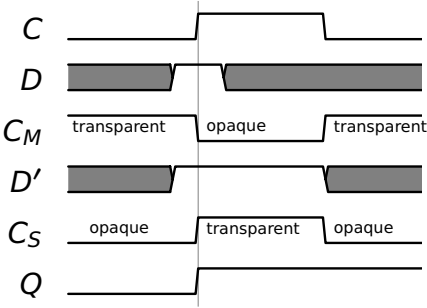
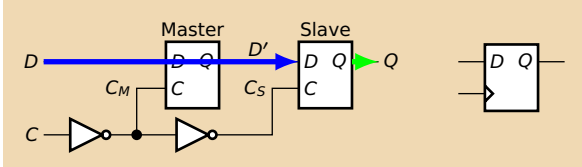
Positive-Edge-Triggered D Flip-Flop



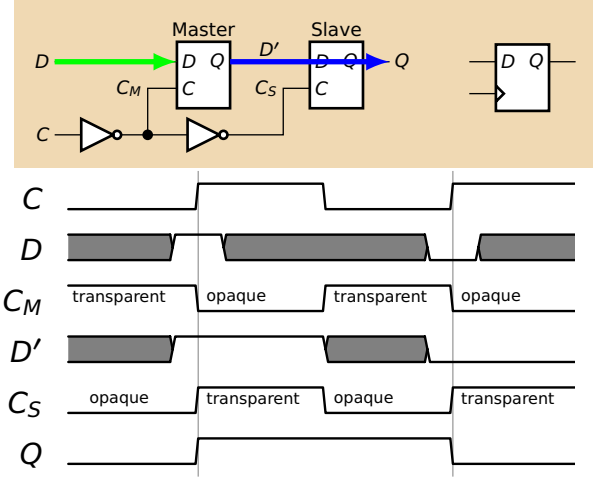
Positive-Edge-Triggered D Flip-Flop



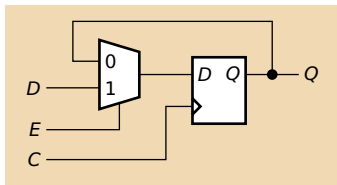
Positive-Edge-Triggered D Flip-Flop



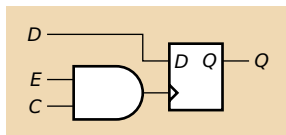
Positive-Edge-Triggered D Flip-Flop



D Flip-Flop with Enable

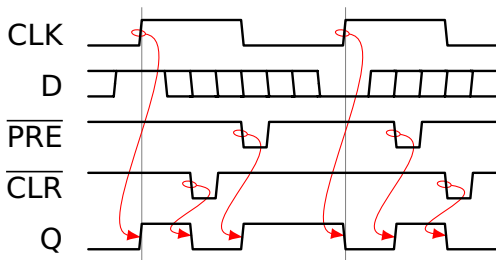
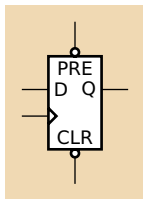


C	E	D	Q
\uparrow	0	X	Q
\uparrow	1	0	0
\uparrow	1	1	1
0	X	X	Q
1	X	X	Q



What's wrong with this solution?

Asynchronous Preset/Clear

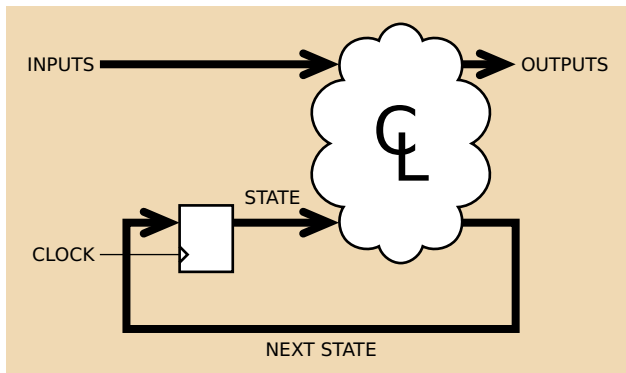


The Synchronous Digital Logic Paradigm

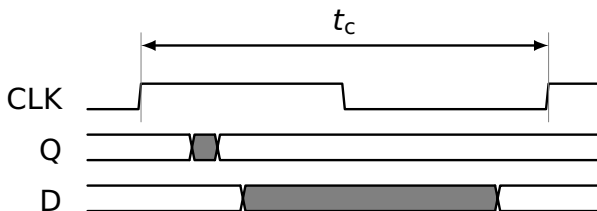
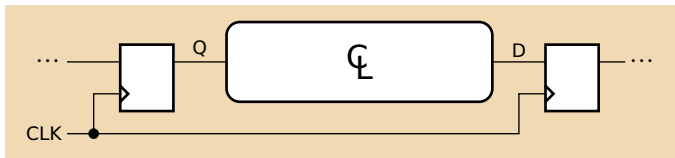
Gates and D
flip-flops only

Each flip-flop
driven by the
same clock

Every cyclic
path contains
at least one
flip-flop

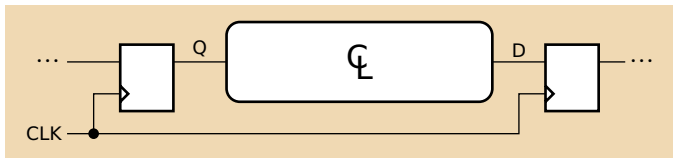


Timing in Synchronous Circuits

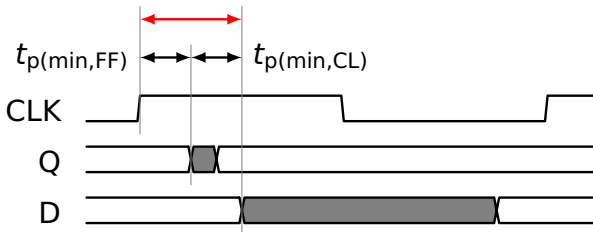


t_c : Clock period. E.g., 10 ns for a 100 MHz clock

Timing in Synchronous Circuits

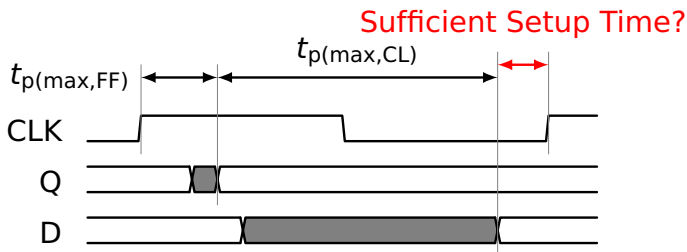
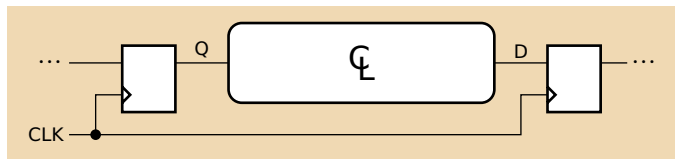


Sufficient Hold Time?



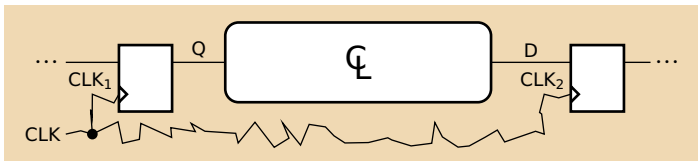
Hold time constraint: how soon after the clock edge can D start changing? Min. FF delay + min. logic delay

Timing in Synchronous Circuits

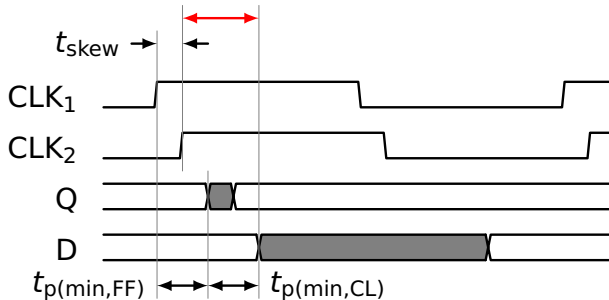


Setup time constraint: when before the clock edge is D guaranteed stable? Max. FF delay + max. logic delay

Clock Skew: What Really Happens

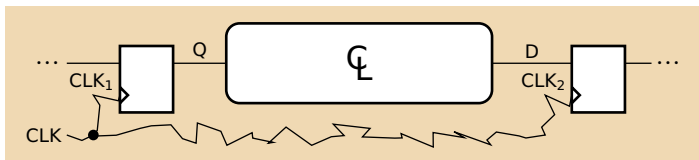


Sufficient Hold Time?

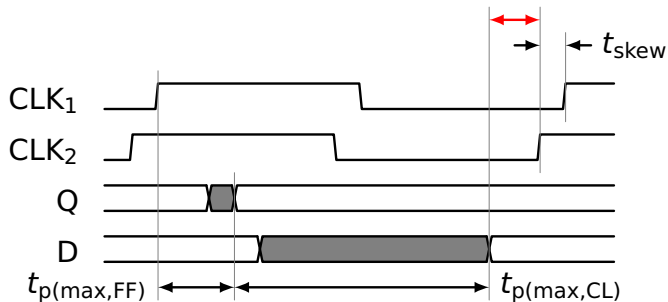


CLK_2 arrives late: clock skew reduces hold time

Clock Skew: What Really Happens

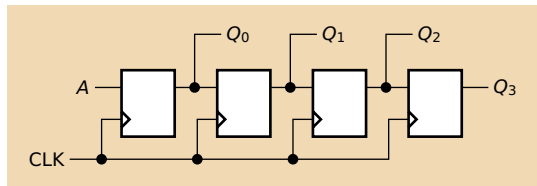


Sufficient Setup Time?



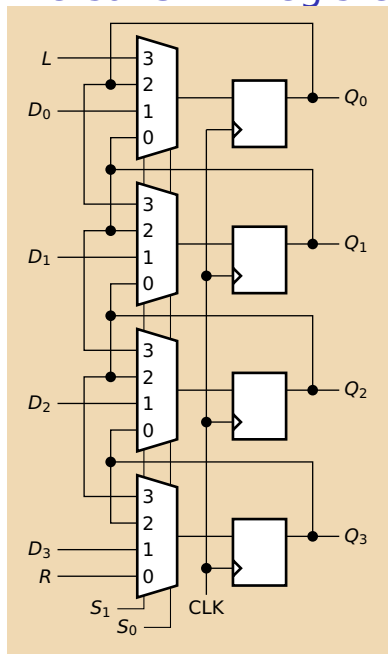
CLK_1 arrives early: clock skew reduces setup time

Cool Sequential Circuits: Shift Registers



A	Q_0	Q_1	Q_2	Q_3
0	X	X	X	X
1	0	X	X	X
1	1	0	X	X
0	1	1	0	X
1	0	1	1	0
0	1	0	1	1
0	0	1	0	1
0	0	0	1	0
1	0	0	0	1
0	1	0	0	0

Universal Shift Register

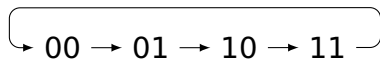


S_1	S_0	Q_3	Q_2	Q_1	Q_0
0	0	R	Q_3	Q_2	Q_1
0	1	D_3	D_2	D_1	D_0
1	0	Q_3	Q_2	Q_1	Q_0
1	1	Q_2	Q_1	Q_0	L

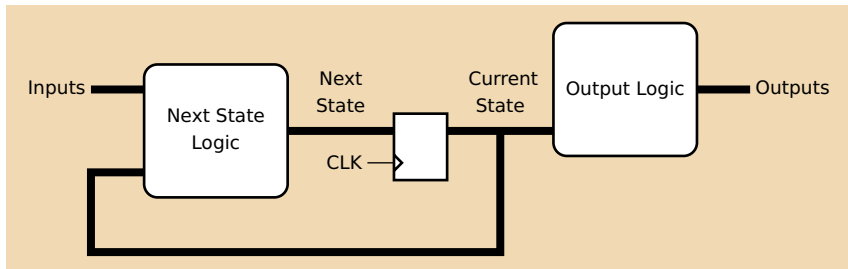
S_1	S_0	Operation
0	0	Shift right
0	1	Load
1	0	Hold
1	1	Shift left

Cool Sequential Circuits: Counters

Cycle through sequences of numbers, e.g.,



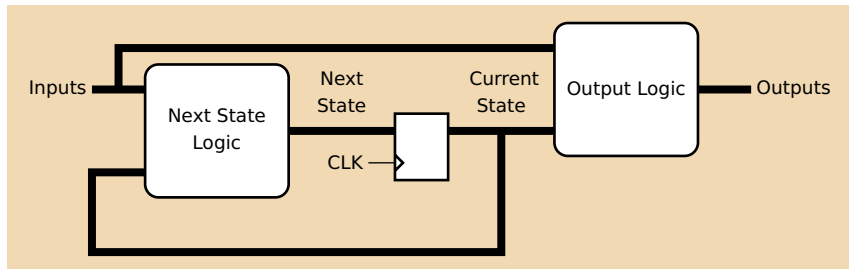
Moore and Mealy Machines



The Moore Form:

Outputs are a function of *only* the current state.

Moore and Mealy Machines

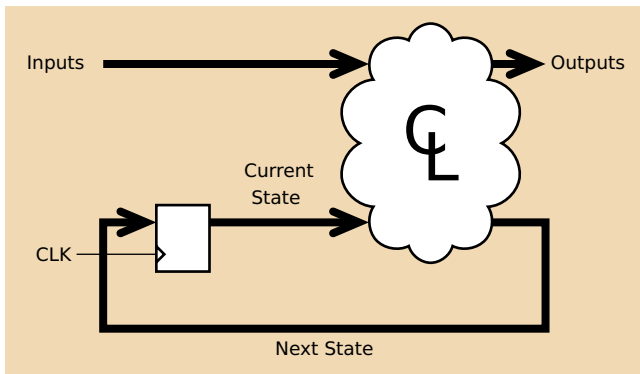


The Mealy Form:

Outputs may be a function of *both* the current state and the inputs.

A mnemonic: *Moore* machines often have *more* states.

Mealy Machines are the Most General



Another, equivalent way of drawing Mealy Machines

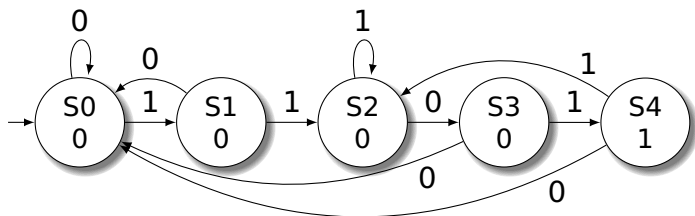
This is exactly the synchronous digital logic paradigm

Moore vs. Mealy FSMs

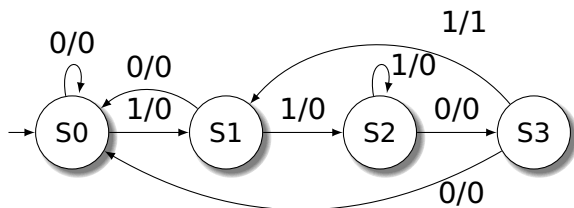
Alyssa P. Hacker has a snail that crawls down a paper tape with 1's and 0's on it. The snail smiles whenever the last four digits it has crawled over are 1101. Design Moore and Mealy FSMs of the snail's brain.



State Transition Diagrams: Looking for “1101”



Moore Machine: States indicate output



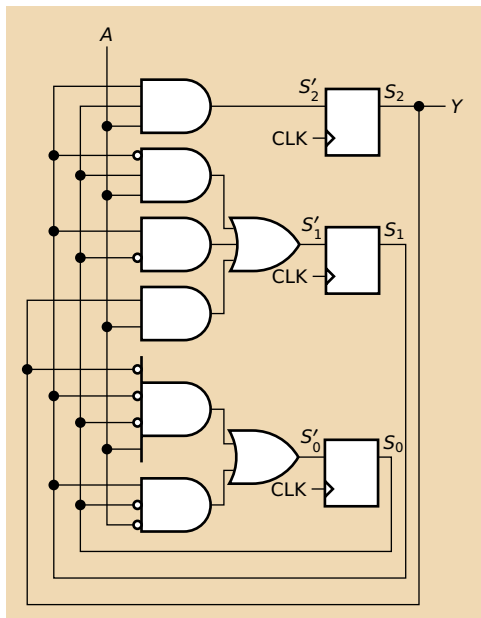
Mealy Machine: Arcs indicate input/output

Moore Machine

Next State			Output	
S	A	S'	S	Y
S0	0	S0	S0	0
S0	1	S1	S1	0
S1	0	S0	S2	0
S1	1	S2	S3	0
S2	0	S3	S4	1
S2	1	S2		
S3	0	S0		
S3	1	S4		
S4	0	S0		
S4	1	S2		

Moore Machine

Next State			Output	
S	A	S'	S	Y
000	0	000	000	0
000	1	001	001	0
001	0	000	010	0
001	1	010	011	0
010	0	011	100	1
010	1	010		
011	0	000		
011	1	100		
100	0	000		
100	1	010		

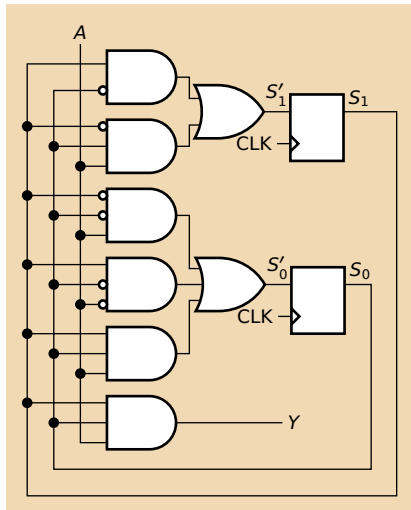


Mealy Machine

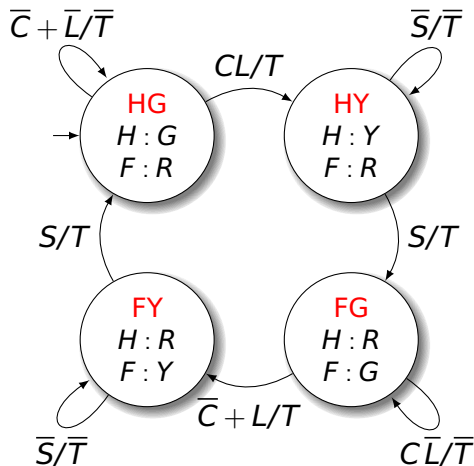
S	A	S'	Y
S0	0	S0	0
S0	1	S1	0
S1	0	S0	0
S1	1	S2	0
S2	0	S3	0
S2	1	S2	0
S3	0	S0	0
S3	1	S1	1

Mealy Machine

S	A	S'	Y
00	0	00	0
00	1	01	0
01	0	00	0
01	1	10	0
10	0	11	0
10	1	10	0
11	0	00	0
11	1	01	1



State Transition Diagram for the TLC



Inputs:

C: Car sensor

S: Short Timeout

L: Long Timeout

Outputs:

T: Timer Reset

H: Highway color

F: Farm road color

S	C	S	L	T	S'
HG	0	X	X	0	HG
HG	X	X	0	0	HG
HG	1	X	1	1	HY
HY	X	0	X	0	HY
HY	X	1	X	1	FG
FG	1	X	0	0	FG
FG	0	X	X	1	FY
FG	X	X	1	1	FY
FY	X	0	X	0	FY
FY	X	1	X	1	HG

S	H	F
HG	G	R
HY	Y	R
FG	R	G
FY	R	Y

State and Output Encoding

S	C	S	L	T	S'
HG	0	X	X	0	HG
HG	X	X	0	0	HG
HG	1	X	1	1	HY
HY	X	0	X	0	HY
HY	X	1	X	1	FG
FG	1	X	0	0	FG
FG	0	X	X	1	FY
FG	X	X	1	1	FY
FY	X	0	X	0	FY
FY	X	1	X	1	HG

S	H	F
HG	G	R
HY	Y	R
FG	R	G
FY	R	Y

A one-hot
encoding:

HG	0001
HY	0010
FG	0100
FY	1000
G	001
Y	010
R	100

State and Output Encoding

S	C	S	L	T	S'
0001	0	X	X	0	0001
0001	X	X	0	0	0001
0001	1	X	1	1	0010
0010	X	0	X	0	0010
0010	X	1	X	1	0100
0100	1	X	0	0	0100
0100	0	X	X	1	1000
0100	X	X	1	1	1000
1000	X	0	X	0	1000
1000	X	1	X	1	0001

S	H	F
0001	001	100
0010	010	100
0100	100	001
1000	100	010

$$T = S_0CL + S_1S + S_2(\overline{C} + L) + S_3S$$

$$S'_3 = S_2(\overline{C} + L) + S_3\overline{S}$$

$$S'_2 = S_1S + S_2\overline{(\overline{C} + L)}$$

$$S'_1 = S_0CL + S_1\overline{S}$$

$$S'_0 = S_0\overline{CL} + S_3S$$

$$H_R = S_2 + S_3$$

$$H_Y = S_1$$

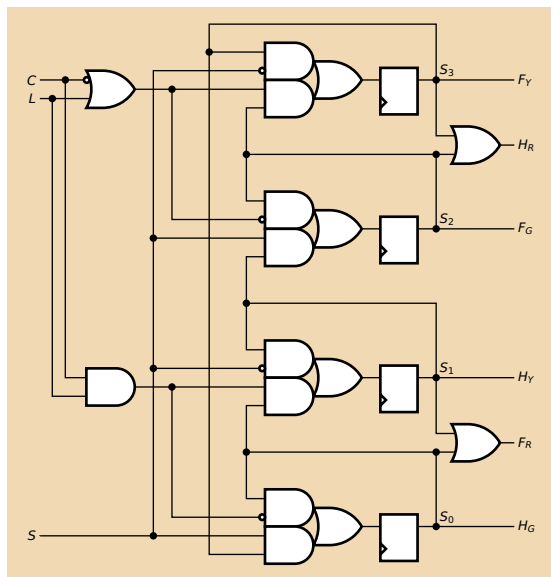
$$H_G = S_0$$

$$F_R = S_0 + S_1$$

$$F_Y = S_3$$

$$F_G = S_2$$

State and Output Encoding



$$T = S_0CL + S_1S + S_2(\overline{C} + L) + S_3S$$

$$S'_3 = S_2(\overline{C} + L) + S_3\overline{S}$$

$$S'_2 = S_1S + S_2\overline{(\overline{C} + L)}$$

$$S'_1 = S_0CL + S_1\overline{S}$$

$$S'_0 = S_0\overline{(CL)} + S_3S$$

$$H_R = S_2 + S_3$$

$$H_Y = S_1$$

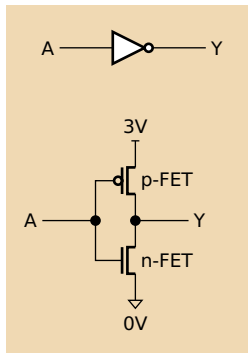
$$H_G = S_0$$

$$F_R = S_0 + S_1$$

$$F_Y = S_3$$

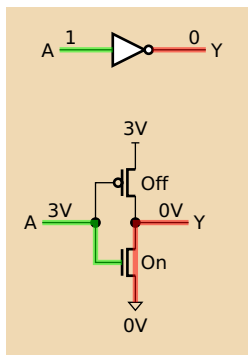
$$F_G = S_2$$

The CMOS Inverter



- An inverter is built from two MOSFETs:
- An n-FET connected to ground
- A p-FET connected to the power supply

The CMOS Inverter



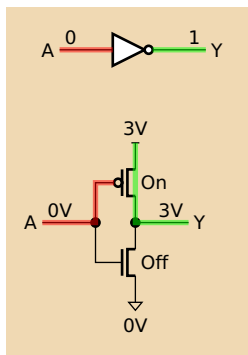
When the input is near the power supply voltage ("1"),

the p-FET is turned off;

the n-FET is turned on, connecting the output to ground ("0").

n-FETs are only good at passing 0's

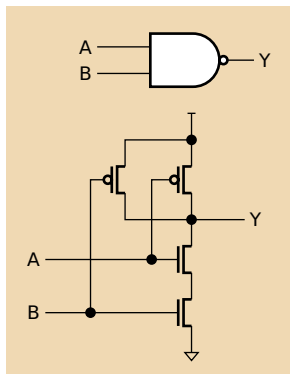
The CMOS Inverter



When the input is near ground (“0”),
the p-FET is turned on, connecting the
output to the power supply (“1”);
the n-FET is turned off.

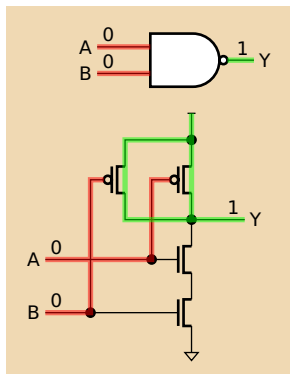
p-FETs are only good at passing 1’s

The CMOS NAND Gate



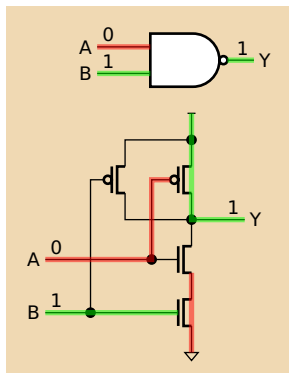
Two-input NAND gate:
two n-FETs in series;
two p-FETs in parallel

The CMOS NAND Gate



Both inputs 0:
Both p-FETs turned on
Output pulled high

The CMOS NAND Gate



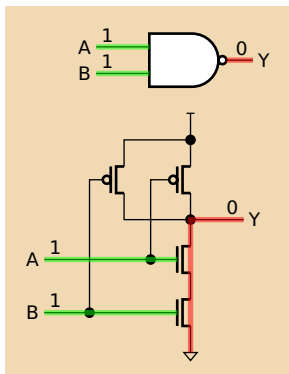
One input 1, the other 0:

One p-FET turned on

Output pulled high

One n-FET turned on, but does not control output

The CMOS NAND Gate



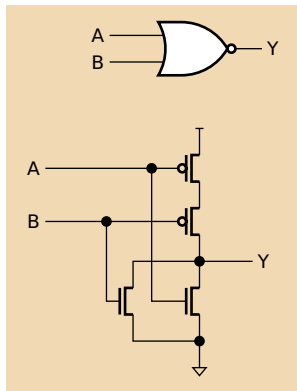
Both inputs 1:

Both n-FETs turned on

Output pulled low

Both p-FETs turned off

The CMOS NOR Gate



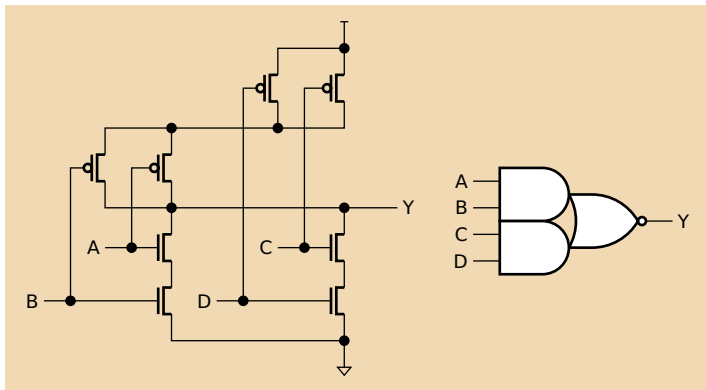
Two-input NOR gate:

two n-FETs in parallel;

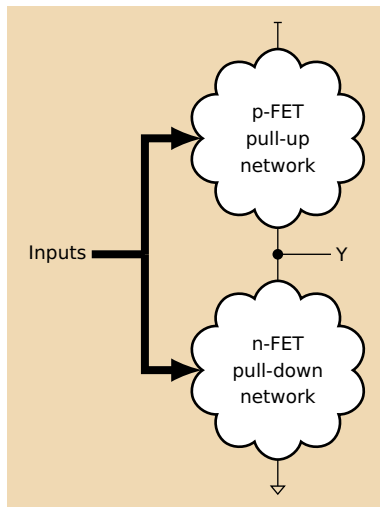
two p-FETs in series.

Not as fast as the NAND gate
because n-FETs are faster than
p-FETs

A CMOS AND-OR-INVERT Gate



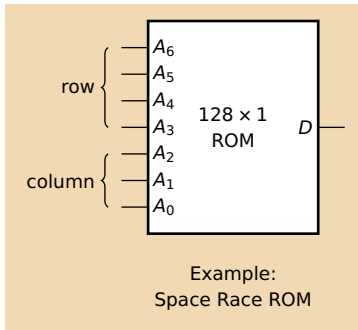
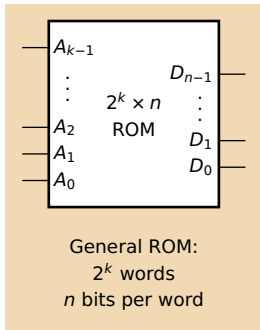
Static CMOS Gate Structure



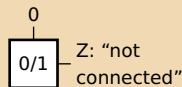
Pull-up and Pull-down networks must be complementary; exactly one should be connected for each input combination.

Series connection in one should be parallel in the other

Read-Only Memories: Combinational Functions

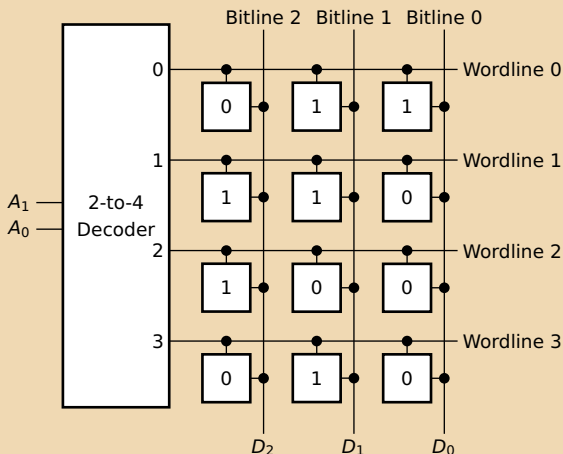


Implementing ROMs

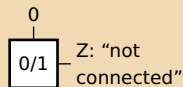


Add. Data

00	011
01	110
10	100
11	010

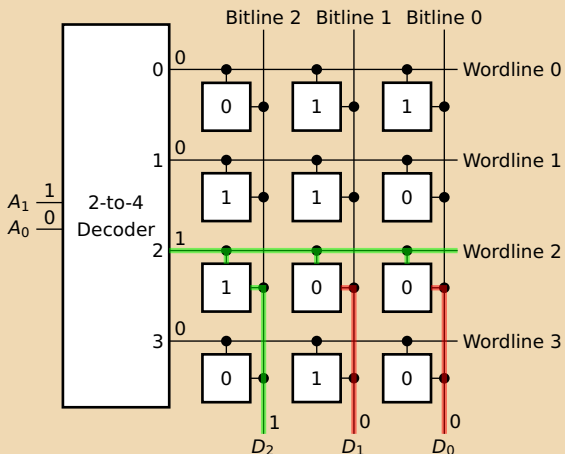


Implementing ROMs

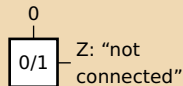


Add. Data

00	011
01	110
10	100
11	010

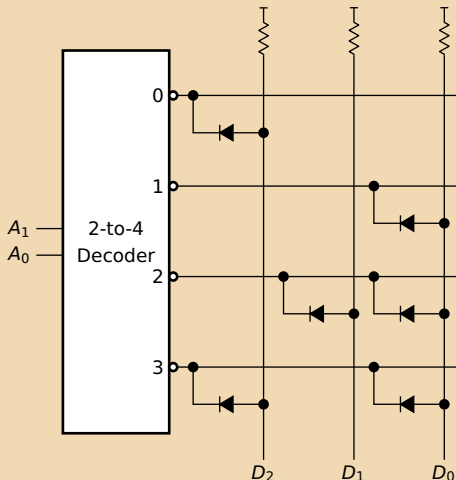


Implementing ROMs

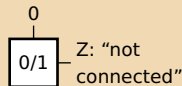


Add. Data

00	011
01	110
10	100
11	010

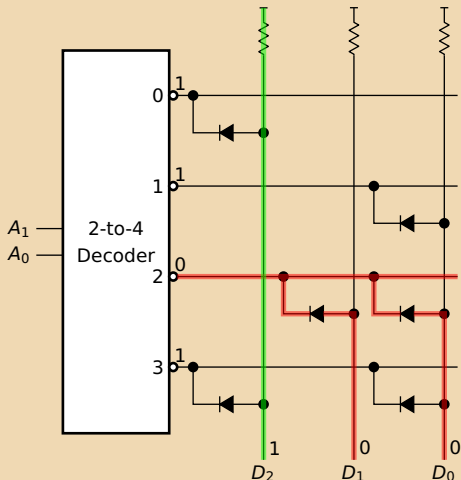


Implementing ROMs



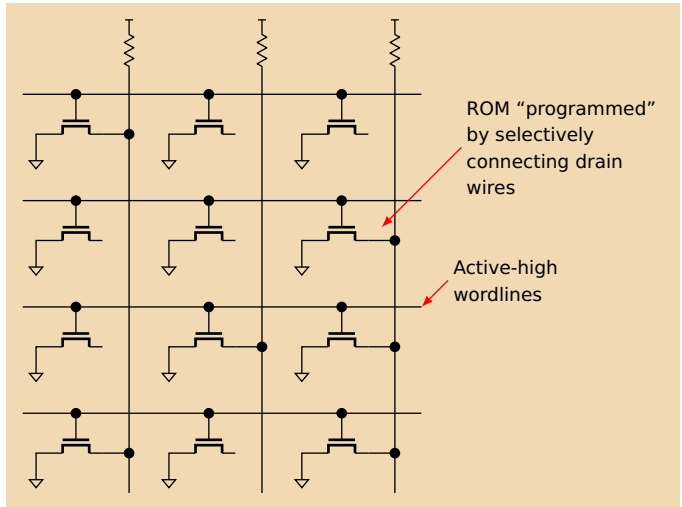
Add. Data

00	011
01	110
10	100
11	010

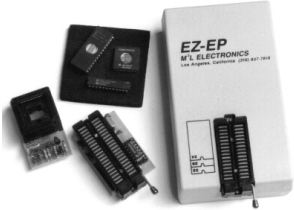
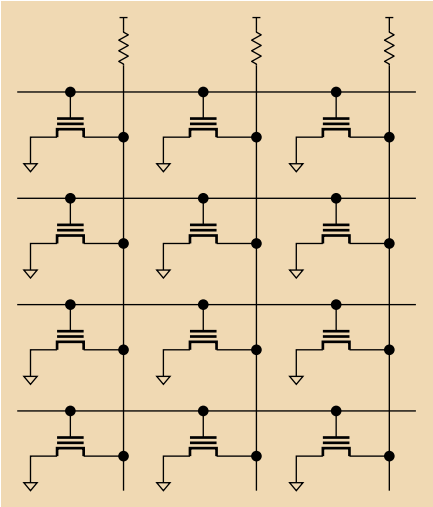


CMOS Mask-Programmed ROMs

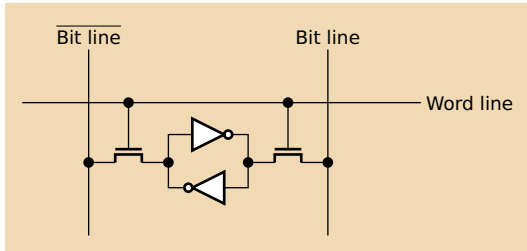
Add. Data	
00	011
01	110
10	100
11	010



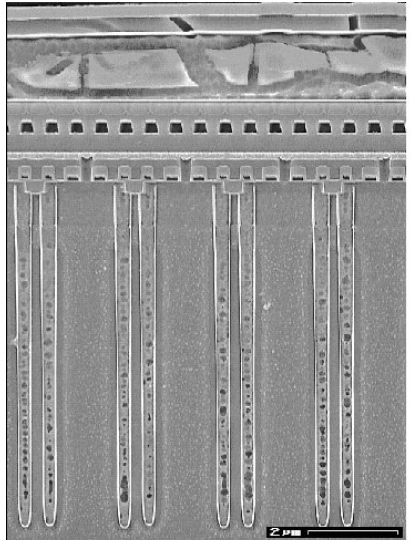
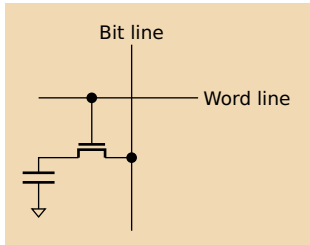
EPROMs and FLASH use Floating-Gate MOSFETs



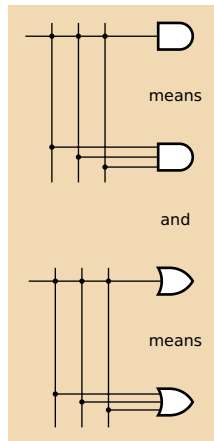
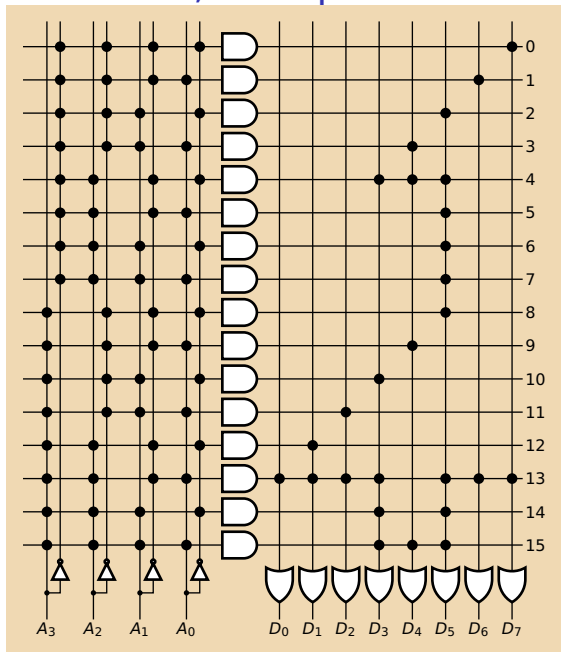
Static Random-Access Memory Cell



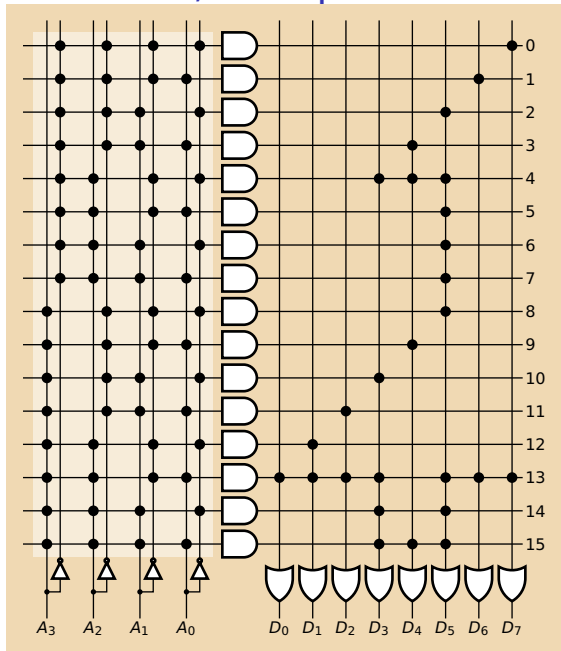
Dynamic RAM Cell



Our Old Pal, the Space Race ROM



Our Old Pal, the Space Race ROM

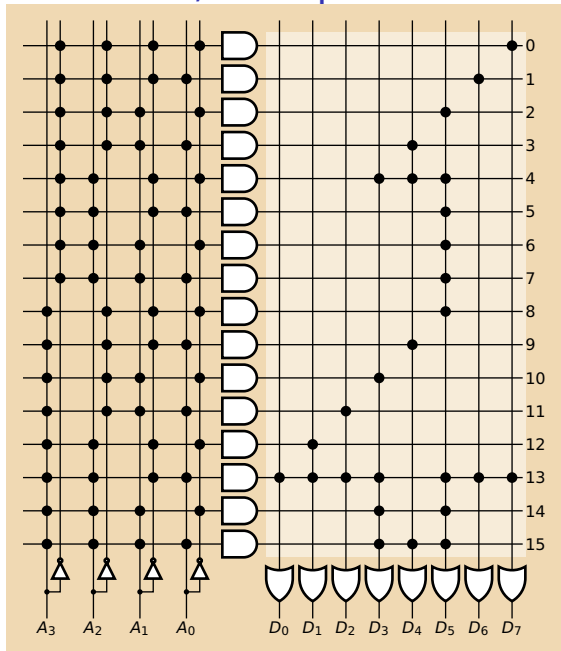


The decoder or
"AND plane"

In a RAM or ROM,
computes every
minterm

Pattern is not
programmable

Our Old Pal, the Space Race ROM

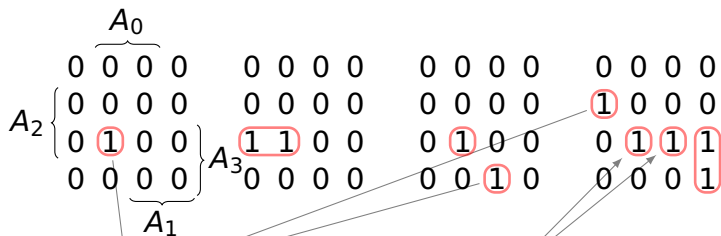


The decoder or
“OR plane”

One term for every
output

Pattern is
programmable =
the contents of the
ROM

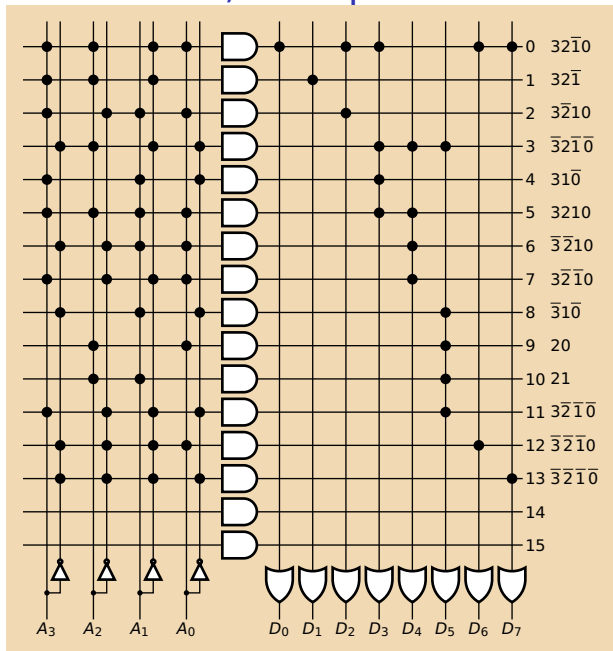
Simplifying the Space Race ROM



Essential minterms mean don't expand these



Our New PAL, the Space Race ROM



$$D_0 = 32\bar{1}0$$

$$D_1 = 32\bar{1}$$

$$D_2 = 3\bar{2}\bar{1}0 + 32\bar{1}0$$

$$D_3 = \bar{3}\bar{2}\bar{1}\bar{0} + 31\bar{0} + 32\bar{1}0 + 3210$$

$$D_4 = \bar{3}\bar{2}10 + \bar{3}\bar{2}\bar{1}\bar{0} + 3\bar{2}\bar{1}0 + 3210$$

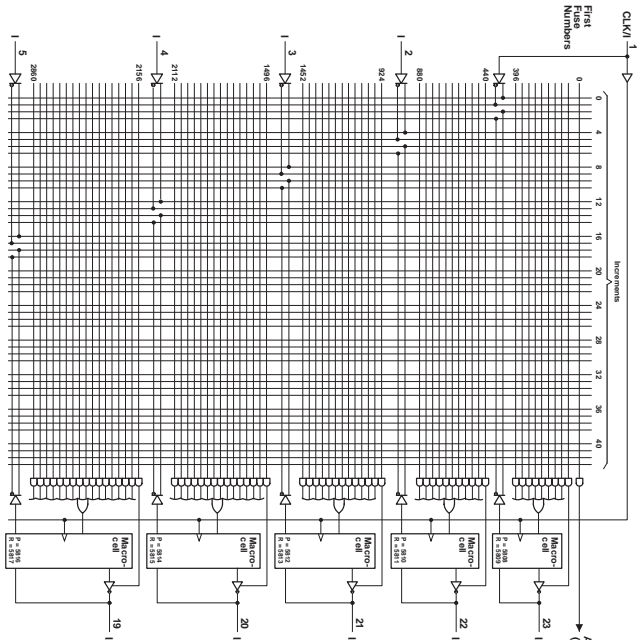
$$D_5 = \bar{3}\bar{1}\bar{0} + 20 + 21 + \bar{3}\bar{2}\bar{1}\bar{0} + 3\bar{2}\bar{1}\bar{0}$$

$$D_6 = \bar{3}\bar{2}\bar{1}0 + 32\bar{1}0$$

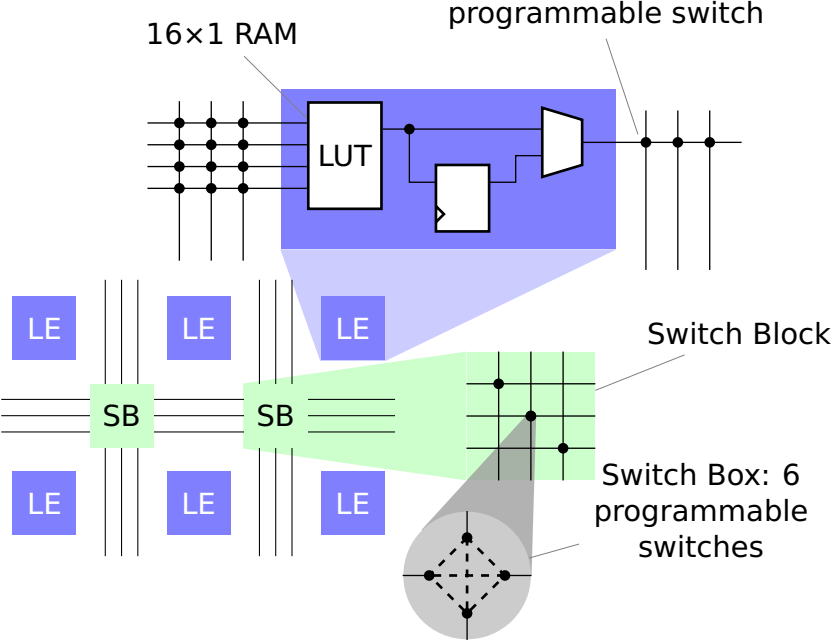
$$D_7 = \bar{3}\bar{2}\bar{1}\bar{0} + 32\bar{1}0$$

Saved two ANDs

A 22V10 PAL: Programmable AND/Fixed OR



Field-Programmable Gate Arrays (FPGAs)



Datapath for dx , dy , *right*, and *down*

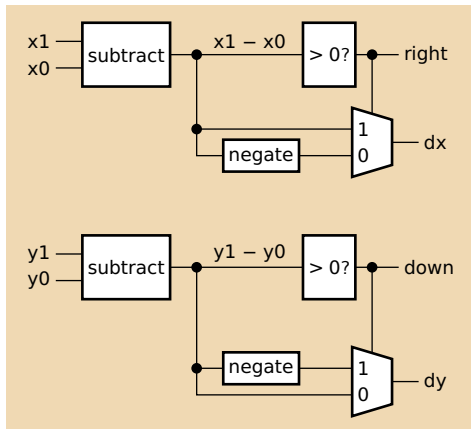
```
void line(Uint16 x0, Uint16 y0,
         Uint16 x1, Uint16 y1)
{
    Sint16 dx;    // Width of bounding box
    Sint16 dy;    // Height of BB (neg)
    Uint16 x, y;  // Current point
    Sint8  sx, sy; // -1 or 1
    Sint16 err;   // Loop-carried value
    Sint16 e2;    // Temporary variable
    int right;    // Boolean
    int down;     // Boolean
```

```
    dx = x1 - x0;
    right = dx > 0;
    if (!right) dx = -dx;
    dy = y1 - y0;
    down = dy > 0;
    if (down) dy = -dy;
```

```
    err = dx + dy;
    x = x0; y = y0;
```

```
    for (;;) {
        plot(x, y);
        if (x == x1 && y == y1)
            break;
        e2 = err << 1;
        if (e2 > dy) {
            err += dy;
            if (right) x++;
            else x--;
        }
        if (e2 < dx) {
            err += dx;
            if (down) y++;
            else y--;
        }
    }
}
```


Datapath for dx , dy , *right*, and *down*

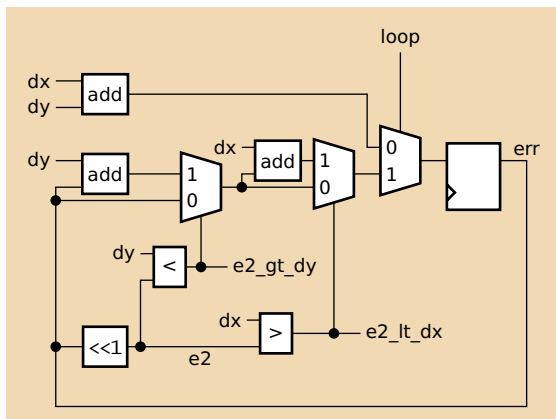


```
dx = x1 - x0;  
right = dx > 0;  
if (!right) dx = -dx;  
dy = y1 - y0;  
down = dy > 0;  
if (down) dy = -dy;
```

```
err = dx + dy;  
x = x0; y = y0;
```

```
for (;;) {  
    plot(x, y);  
    if (x == x1 && y == y1)  
        break;  
    e2 = err << 1;  
    if (e2 > dy) {  
        err += dy;  
        if (right) x++;  
        else x--;  
    }  
    if (e2 < dx) {  
        err += dx;  
        if (down) y++;  
        else y--;  
    }  
}
```

Datapath for *err*

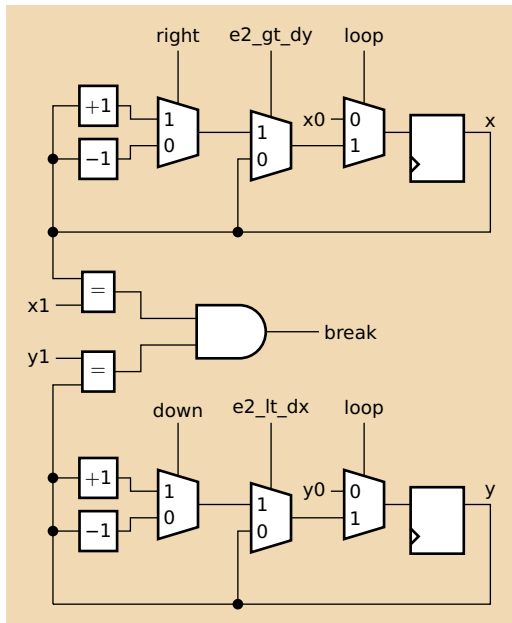


```
dx = x1 - x0;
right = dx > 0;
if (!right) dx = -dx;
dy = y1 - y0;
down = dy > 0;
if (down) dy = -dy;
```

```
err = dx + dy;
x = x0; y = y0;
```

```
for (;;) {
    plot(x, y);
    if (x == x1 && y == y1)
        break;
    e2 = err << 1;
    if (e2 > dy) {
        err += dy;
        if (right) x++;
        else x--;
    }
    if (e2 < dx) {
        err += dx;
        if (down) y++;
        else y--;
    }
}
```

Datapath for x and y

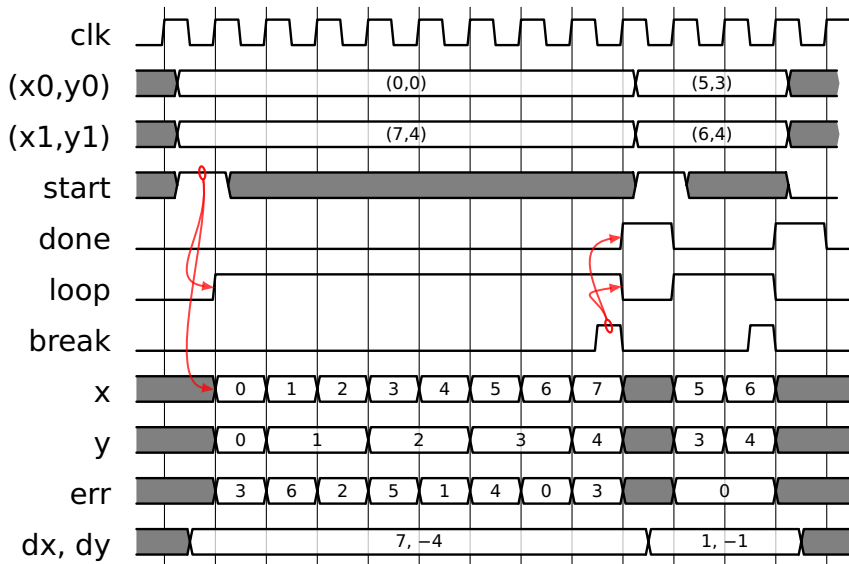


```
dx = x1 - x0;
right = dx > 0;
if (!right) dx = -dx;
dy = y1 - y0;
down = dy > 0;
if (down) dy = -dy;
```

```
err = dx + dy;
x = x0; y = y0;
```

```
for (;;) {
    plot(x, y);
    if (x == x1 && y == y1)
        break;
    e2 = err << 1;
    if (e2 > dy) {
        err += dy;
        if (right) x++;
        else x--;
    }
    if (e2 < dx) {
        err += dx;
        if (down) y++;
        else y--;
    }
}
```

Timing



Control FSM

