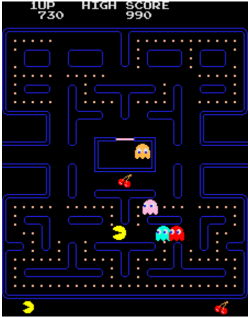
Understand Video Games; Understand Everything

Stephen A. Edwards

Columbia University



The Subject of this Lecture

0

The Subjects of this Lecture

0 1

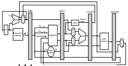
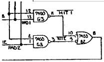
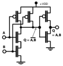
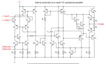
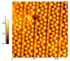
But let your communication be, Yea, yea; Nay, nay: for whatsoever is more than these cometh of evil.

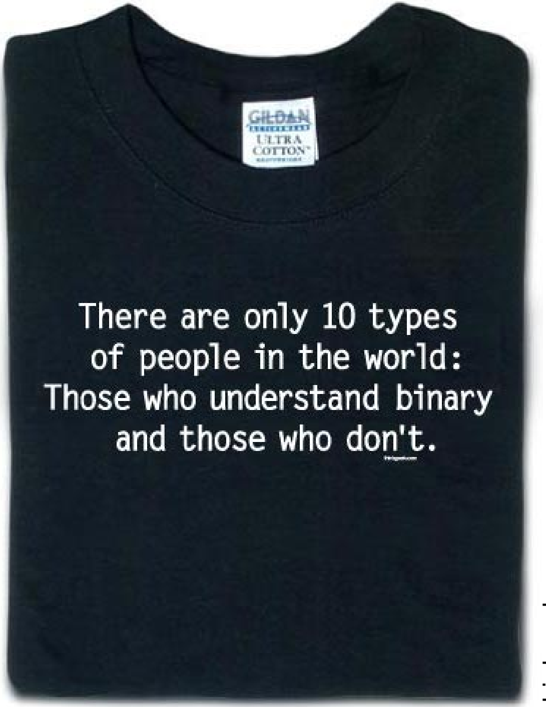
— Matthew 5:37

Engineering Works Because of Abstraction

	Application Software	COMS 3157, 4156, et al.
	Operating Systems	COMS W4118

`;; voice 1 wave select`
`ld a,(#CH1_W_NUM)`
`and a`
`ld a,(#CH1_W_SEL)`
`jr nz,#00b4`
`ld a,(#CH1_E_TABLE0)`
	Architecture	COMS W3827

	Micro-Architecture	COMS W3827
	Logic	COMS W3827
	Digital Circuits	COMS W3827
	Analog Circuits	ELEN 3331
	Devices	ELEN 3106
	Physics	ELEN 3106 et al.



thinkgeek.com

Boolean Logic

AN INVESTIGATION

OF

THE LAWS OF THOUGHT,

ON WHICH ARE FOUNDED

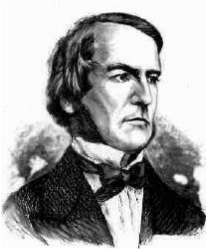
THE MATHEMATICAL THEORIES OF LOGIC
AND PROBABILITIES.

BY

GEORGE BOOLE, LL.D.

PROFESSOR OF MATHEMATICS IN QUEEN'S COLLEGE, CORK.

LONDON:
WALTON AND MABERLY,
UPPER GOWER-STREET, AND IVY-LANE, PATERNOSTER-ROW.
CAMBRIDGE: MACMILLAN AND CO.
1854.



George Boole
1815–1864

Boole's Intuition Behind Boolean Logic

Variables X, Y, ... represent classes of things

No imprecision: A thing either is or is not in a class

If X is "sheep" and Y is "white things," XY are all white sheep,

$$XY = YX$$

and

$$XX = X.$$

If X is "men" and Y is "women," $X + Y$ is "both men and women,"

$$X + Y = Y + X$$

and

$$X + X = X.$$

If X is "men," Y is "women," and Z is "European," $Z(X + Y)$ is "European men and women" and

$$Z(X+Y) = ZX+ZY.$$

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."

$X =$ born in New York
$Y =$ lived here ten years

$$X + (\overline{X} \cdot Y)$$

Axioms
$X + Y = Y + X$
$X \cdot Y = Y \cdot X$
$X + (Y + Z) = (X + Y) + Z$
$X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$
$X + (X \cdot Y) = X$
$X \cdot (X + Y) = X$
$X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z)$
$X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$
$X + \overline{X} = 1$
$X \cdot \overline{X} = 0$

Lemma:

$$\begin{aligned} X \cdot 1 &= X \cdot (X + \overline{X}) \\ &= X \cdot (X + Y) \text{ if } Y = \overline{X} \\ &= X \end{aligned}$$

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."

$X =$ born in New York
$Y =$ lived here ten years

$$X + (\overline{X} \cdot Y)$$
$$= (X + \overline{X}) \cdot (X + Y)$$

Axioms
$X + Y = Y + X$
$X \cdot Y = Y \cdot X$
$X + (Y + Z) = (X + Y) + Z$
$X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$
$X + (X \cdot Y) = X$
$X \cdot (X + Y) = X$
$X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z)$
$X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$
$X + \overline{X} = 1$
$X \cdot \overline{X} = 0$

Lemma:

$$X \cdot 1 = X \cdot (X + \overline{X})$$
$$= X \cdot (X + Y) \text{ if } Y = \overline{X}$$
$$= X$$

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."

$X =$ born in New York
$Y =$ lived here ten years

$$\begin{aligned} & X + (\overline{X} \cdot Y) \\ &= (X + \overline{X}) \cdot (X + Y) \\ &= 1 \cdot (X + Y) \end{aligned}$$

Axioms
$X + Y = Y + X$
$X \cdot Y = Y \cdot X$
$X + (Y + Z) = (X + Y) + Z$
$X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$
$X + (X \cdot Y) = X$
$X \cdot (X + Y) = X$
$X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z)$
$X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$
$X + \overline{X} = 1$
$X \cdot \overline{X} = 0$

Lemma:

$$\begin{aligned} X \cdot 1 &= X \cdot (X + \overline{X}) \\ &= X \cdot (X + Y) \text{ if } Y = \overline{X} \\ &= X \end{aligned}$$

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."

$X =$ born in New York
$Y =$ lived here ten years

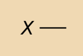
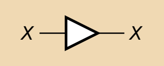
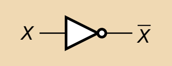
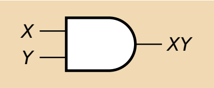
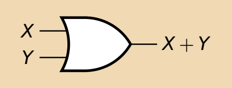
$$
\begin{aligned}
& X + (\overline{X} \cdot Y) \\
&\quad = (X + \overline{X}) \cdot (X + Y) \\
&\quad = 1 \cdot (X + Y) \\
&\quad = X + Y
\end{aligned}
$$

Axioms
$X + Y = Y + X$
$X \cdot Y = Y \cdot X$
$X + (Y + Z) = (X + Y) + Z$
$X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z$
$X + (X \cdot Y) = X$
$X \cdot (X + Y) = X$
$X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z)$
$X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$
$X + \overline{X} = 1$
$X \cdot \overline{X} = 0$

Lemma:

$$
\begin{aligned}
X \cdot 1 &= X \cdot (X + \overline{X}) \\
&= X \cdot (X + Y) \text{ if } Y = \overline{X} \\
&= X
\end{aligned}
$$

Alternate Notations for Boolean Logic

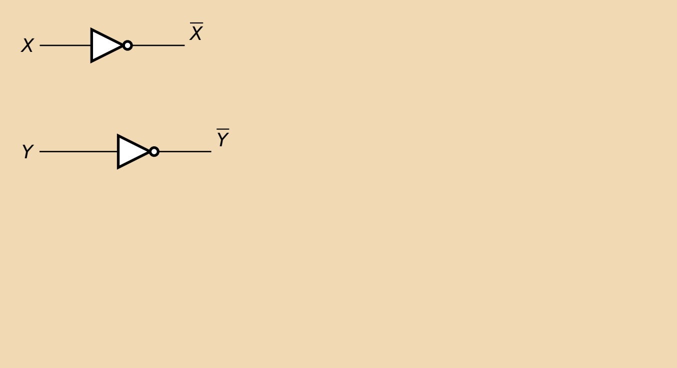
Operator	**Math**	**Engineer**	**Schematic**
Copy	x	X	 or 
Complement	$\neg x$	$\overline{X}$	
AND	$x \wedge y$	XY or $X \cdot Y$	
OR	$x \vee y$	$X + Y$	

Expressions to Schematics

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

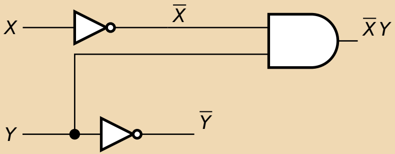
Expressions to Schematics

$$F = \overline{X}\,Y + X\,\overline{Y}$$



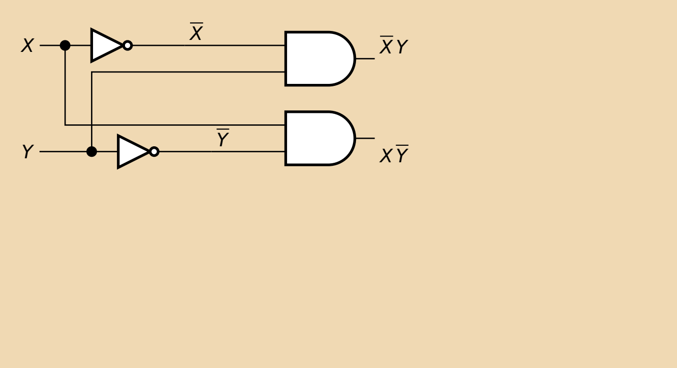
Expressions to Schematics

$$F = \overline{X}\,Y + X\,\overline{Y}$$



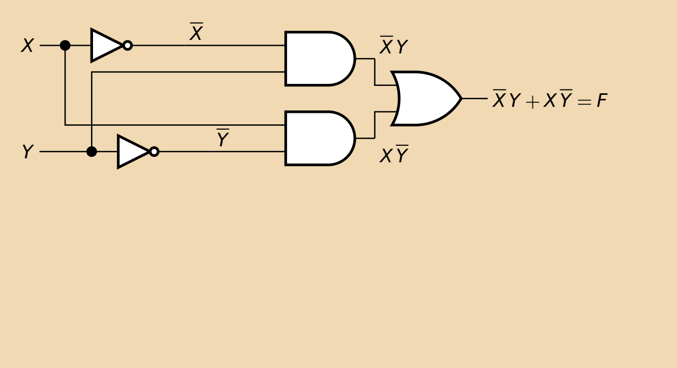
Expressions to Schematics

$$F = \overline{X}\,Y + X\,\overline{Y}$$



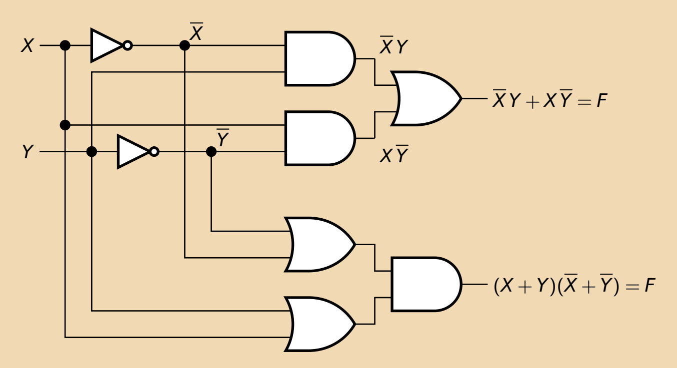
Expressions to Schematics

$$F = \overline{X}\,Y + X\,\overline{Y}$$



Expressions to Schematics

$$F = \overline{X}\,Y + X\,\overline{Y} = (X + Y)(\overline{X} + \overline{Y})$$



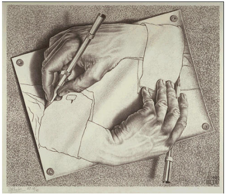
The Decimal Positional Numbering System

Ten figures: 0 1 2 3 4 5 6 7 8 9

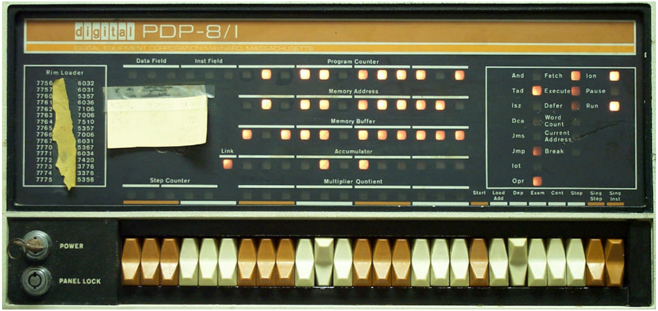
$7 \times 10^2 + 3 \times 10^1 + 0 \times 10^0 = 730_{10}$

$9 \times 10^2 + 9 \times 10^1 + 0 \times 10^0 = 990_{10}$

Why base ten?



Binary



DEC PDP-8/I, c. 1968

Dec	Bin
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010

$$\begin{aligned} PC &= 0 \times 2^{11} + 1 \times 2^{10} + 0 \times 2^{9} + 1 \times 2^{8} + 1 \times 2^{7} + 0 \times 2^{6} + \\ &\quad 1 \times 2^{5} + 1 \times 2^{4} + 1 \times 2^{3} + 1 \times 2^{2} + 0 \times 2^{1} + 1 \times 2^{0} \\ &= 1469_{10} \end{aligned}$$

Binary Addition Algorithm

$$\begin{array}{r} 10011 \\ +11001 \\ \hline \end{array}$$

$$1 + 1 = 10$$

+	0	1
0	00	01
1	01	10
10	10	11

Binary Addition Algorithm

$$\begin{array}{r} {\color{red}1} \\ 10011 \\ +11001 \\ \hline {\color{blue}0} \end{array}$$

$$1 + 1 = {\color{red}1}{\color{blue}0}$$

$$1 + 1 + 0 = {\color{red}1}{\color{blue}0}$$

+	0	1
0	00	01
1	01	10
10	10	11

Binary Addition Algorithm

$$\begin{array}{r} 11 \\ 10011 \\ +11001 \\ \hline 00 \end{array}$$

$$1 + 1 = 10$$

$$1 + 1 + 0 = 10$$

$$1 + 0 + 0 = 01$$

+	0	1
0	00	01
1	01	10
10	10	11

Binary Addition Algorithm

$$
\begin{array}{r}
011 \\
10011 \\
+11001 \\
\hline
100
\end{array}
$$

$$1 + 1 = 10$$

$$1 + 1 + 0 = 10$$

$$1 + 0 + 0 = 01$$

$$0 + 0 + 1 = 01$$

+	0	1
0	00	01
1	01	10
10	10	11

Binary Addition Algorithm

$$\begin{array}{r} 0011 \\ 10011 \\ +11001 \\ \hline 1100 \end{array}$$

$$\begin{aligned} 1+1 &= 10 \\ 1+1+0 &= 10 \\ 1+0+0 &= 01 \\ 0+0+1 &= 01 \\ 0+1+1 &= 10 \end{aligned}$$

+	0	1
0	00	01
1	01	10
10	10	11

Binary Addition Algorithm

$$
\begin{array}{r}
10011 \\
10011 \\
+11001 \\
\hline
101100
\end{array}
$$

$$
\begin{aligned}
1+1 &= 10 \\
1+1+0 &= 10 \\
1+0+0 &= 01 \\
0+0+1 &= 01 \\
0+1+1 &= 10
\end{aligned}
$$

+	0	1
0	00	01
1	01	10
10	10	11

Arithmetic Circuits

Arithmetic: Addition

Adding two one-bit numbers:

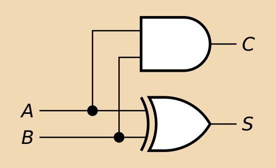
A and *B*

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



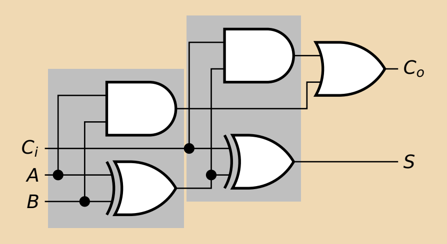
Half Adder

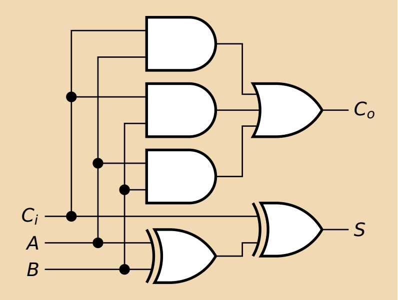
Full Adder

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

```
   111000
    111010
+    11100
  1010110
```

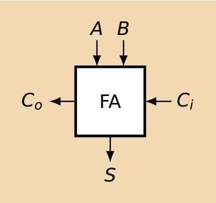
$C_i A B$	$C_o S$
0 0 0	0 0
0 0 1	0 1
0 1 0	0 1
0 1 1	1 0
1 0 0	0 1
1 0 1	1 0
1 1 0	1 0
1 1 1	1 1

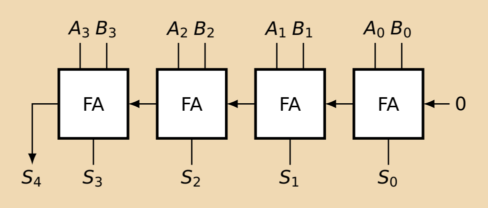




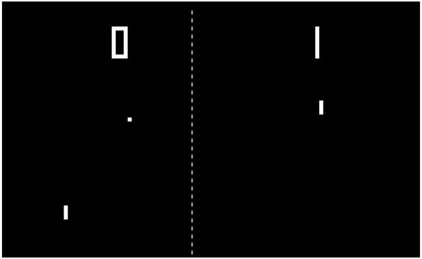
$$0+0=00$$
$$0+1+0=01$$
$$0+0+1=01$$
$$0+1+1=10$$
$$1+1+1=11$$
$$1+1+0=10$$

A Four-Bit Ripple-Carry Adder





PONG



PONG, Atari 1973

Built from TTL logic gates; no computer, no software

Launched the video arcade game revolution

Horizontal Ball Control in PONG

M	L	R	A	B
0	0	0	X	X
0	0	1	0	1
0	1	0	0	1
0	1	1	X	X
1	0	0	X	X
1	0	1	1	0
1	1	0	1	1
1	1	1	X	X

The ball moves either left or right.

Part of the control circuit has three inputs: M (“move”), L (“left”), and R (“right”).

It produces two outputs A and B.

Here, “X” means “I don’t care what the output is; I never expect this input combination to occur.”

Horizontal Ball Control in PONG

M	L	R	A	B
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	0	0
1	0	0	0	0
1	0	1	1	0
1	1	0	1	1
1	1	1	0	0

Assume all the X's are 0's:

$$A = M\overline{L}R + ML\overline{R}$$

$$B = \overline{M}\,\overline{L}R + \overline{M}\,L\overline{R} + ML\overline{R}$$

3 inv + 4 AND3 + 1 OR2 + 1 OR3

Horizontal Ball Control in PONG

M	L	R	A	B
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	1	0

Choosing better values for the X's:

$A = ML + MR$

$B = \overline{MR}$

3 NAND2

Horizontal Ball Control in PONG

M	L	R	A	B
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	1	1
1	0	1	1	0
1	1	0	1	1
1	1	1	1	0

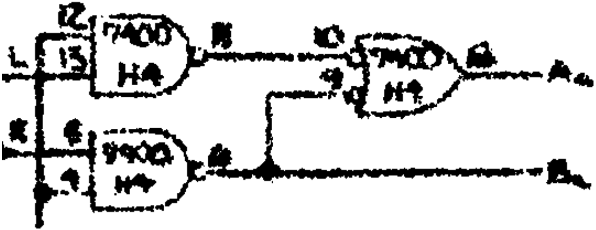
Being even more clever:

$A = M$

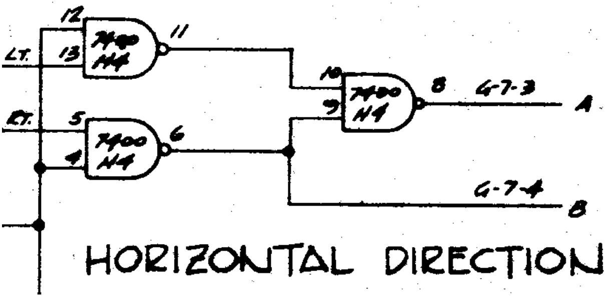
$B = \overline{MR}$

1 NAND2

The Actual Pong Circuit



Pong,
Atari, 1972



Winner,
Midway, 1973

Karnaugh Maps

Basic trick: put "similar" variable values near each other so simple functions are obvious

M	L	R	A	B
0	0	0	X	X
0	0	1	0	1
0	1	0	0	1
0	1	1	X	X
1	0	0	X	X
1	0	1	1	0
1	1	0	1	1
1	1	1	X	X

The M's are already arranged nicely

Karnaugh Maps

Basic trick: put “similar” variable values near each other so simple functions are obvious

M	*L*	*R*	*A*	*B*
0	0	0	X	X
0	0	1	0	1
0	1	0	0	1
0	1	1	X	X
1	0	0	X	X
1	0	1	1	0
1	1	0	1	1
1	1	1	X	X

Let’s rearrange the *L*’s by permuting two pairs of rows

Karnaugh Maps

Basic trick: put “similar” variable values near each other so simple functions are obvious

M	*L*	*R*	*A*	*B*
0	0	0	X	X
0	0	1	0	1
0	1	0	0	1
0	1	1	X	X
1	0	0	X	X
1	0	1	1	0
1	1	0	1	1
1	1	1	X	X

Let’s rearrange the *L*’s by permuting two pairs of rows

Karnaugh Maps

Basic trick: put "similar" variable values near each other so simple functions are obvious

M	L	R	A	B
0	0	0	X	X
0	0	1	0	1
0	1	0	0	1
0	1	1	X	X
1	0	0	X	X
1	0	1	1	0

Let's rearrange the L's by permuting two pairs of rows

1	1	0	1	1
1	1	1	X	X

Karnaugh Maps

Basic trick: put “similar” variable values near each other so simple functions are obvious

M	*L*	*R*	*A*	*B*
0	0	0	X	X
0	0	1	0	1
0	1	0	0	1
0	1	1	X	X
1	0	0	X	X
1	0	1	1	0

Let’s rearrange the *L*’s by permuting two pairs of rows

1	1	0	1	1
1	1	1	X	X

Karnaugh Maps

Basic trick: put "similar" variable values near each other so simple functions are obvious

M	*L*	*R*	*A*	*B*
0	0	0	X	X
0	0	1	0	1
0	1	0	0	1
0	1	1	X	X
1	1	0	1	1
1	1	1	X	X
1	0	0	X	X
1	0	1	1	0

Let's rearrange the *L*'s by permuting two pairs of rows

Karnaugh Maps

Basic trick: put “similar” variable values near each other so simple functions are obvious

M	*L*	*R*	*A*	*B*
0	0	0	X	X
0	0	1	0	1
0	1	0	0	1
0	1	1	X	X
1	1	0	1	1
1	1	1	X	X
1	0	0	X	X
1	0	1	1	0

Let’s rearrange the *L*’s by permuting two pairs of rows

Karnaugh Maps

Basic trick: put "similar" variable values near each other so simple functions are obvious

M	*L*	*R*	*A*	*B*
0	0	0	X	X
0	0	1	0	1
0	1	0	0	1
0	1	1	X	X
1	1	0	1	1
1	1	1	X	X
1	0	0	X	X
1	0	1	1	0

Let's rearrange the *L*'s by permuting two pairs of rows

Karnaugh Maps

Basic trick: put "similar" variable values near each other so simple functions are obvious

M	L	R	A	B
0	0	0	X	X
0	0	1	0	1
0	1	0	0	1
0	1	1	X	X
1	1	0	1	1
1	1	1	X	X
1	0	0	X	X
1	0	1	1	0

Let's rearrange the L's by permuting two pairs of rows

Karnaugh Maps

Basic trick: put “similar” variable values near each other so simple functions are obvious

M	L	R	A	B
0	0	0	X	X
0	0	1	0	1
0	1	0	0	1
0	1	1	X	X
1	1	0	1	1
1	1	1	X	X
1	0	0	X	X
1	0	1	1	0

The R’s are really crazy; let’s use the second dimension

Karnaugh Maps

Basic trick: put "similar" variable values near each other so simple functions are obvious

M	*L*	*R*	*A*	*B*
0	0	0	X	X
0	0	1	0	1
0	1	0	0	1
0	1	1	X	X
1	1	0	1	1
1	1	1	X	X
1	0	0	X	X
1	0	1	1	0

The *R*'s are really crazy; let's use the second dimension

Karnaugh Maps

Basic trick: put “similar” variable values near each other so simple functions are obvious

M	L	R	A	B
00	00	01	X0	X1
00	11	01	0X	1X
11	11	01	1X	1X
11	00	01	X1	X0

The R’s are really crazy; let’s use the second dimension

Karnaugh Maps

Basic trick: put “similar” variable values near each other so simple functions are obvious

M	*L*	*R*	*A*	*B*
00	00	01	X0	X1
00	11	01	0X	1X
11	11	01	1X	1X
11	00	01	X1	X0

MR

M

Maurice Karnaugh's Maps

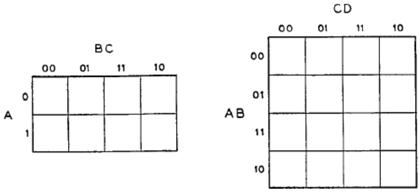
The Map Method for Synthesis of Combinational Logic Circuits

M. KARNAUGH
NONMEMBER AIEE

THE SEARCH for simple abstract techniques to be applied to the design of switching systems is still, despite some recent advances, in its early stages. The problem in this area which has been attacked most energetically is that of the synthesis of efficient combinational that is, nonsequential, logic circuits.

be convenient to describe other methods in terms of Boolean algebra. Whenever the term "algebra" is used in this paper, it will refer to Boolean algebra, where addition corresponds to the logical connective "or," while multiplication corresponds to "and."

The minimizing chart,[2] developed at

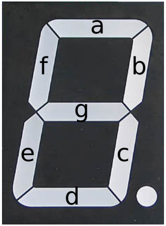


(A) (B)

Fig. 2. Graphical representations of the input conditions for three and for four variables

Transactions of the AIEE, 1953

The Seven-Segment Decoder Example



W	*X*	*Y*	*Z*	*a*	*b*	*c*	*d*	*e*	*f*	*g*
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	0	X	X	X	X	X	X	X
1	0	1	1	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X
1	1	1	1	0	0	0	0	0	0	0

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

Karnaugh map (*Z* spans columns 2–3, *Y* spans columns 3–4, *X* spans rows 2–3, *W* spans rows 3–4):

		Z	*Z*		
	1	0	1	1	
X	0	1	1	1	
X	X	X	0	X	*W*
	1	1	X	X	*W*
			Y	*Y*	

The Karnaugh Map
Sum-of-Products Challenge

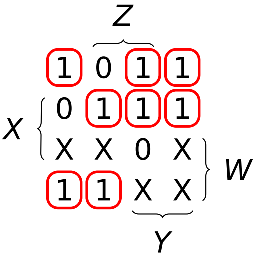
Cover all the 1's and none of the 0's using as few literals (gate inputs) as possible.

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.

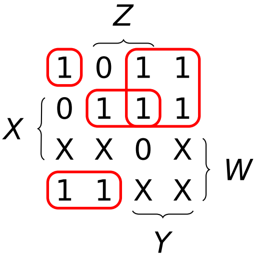
$$a = \overline{W}\,\overline{X}\,\overline{Y}\,\overline{Z} + \overline{W}\,\overline{X}\,Y\,Z + \overline{W}\,\overline{X}\,Y\,\overline{Z} + \overline{W}\,X\,\overline{Y}\,Z + \overline{W}\,X\,Y\,Z + \overline{W}\,X\,Y\,\overline{Z} + W\,\overline{X}\,\overline{Y}\,\overline{Z} + W\,\overline{X}\,\overline{Y}\,Z$$

$8 \times 4 = 32$ 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:
$AB + AC = A(B + C)$

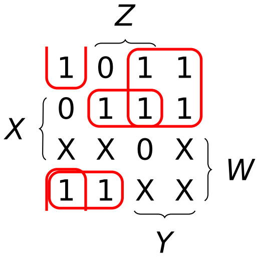
$$a = \overline{W}\,\overline{X}\,\overline{Y}\,\overline{Z} + \overline{W}Y + \overline{W}XZ + W\overline{X}\,\overline{Y}$$

$4 + 2 + 3 + 3 = 12$ literals

4 inv + 1 AND4 + 2 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



Missed one: Remember this is actually a torus.

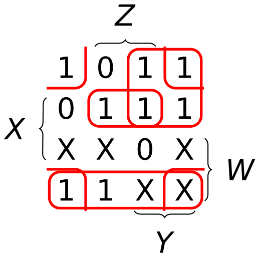
$$a = \overline{X}\,\overline{Y}\,\overline{Z} + \overline{W}Y + \overline{W}XZ + W\overline{X}\,\overline{Y}$$

$3 + 2 + 3 + 3 = 11$ 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:

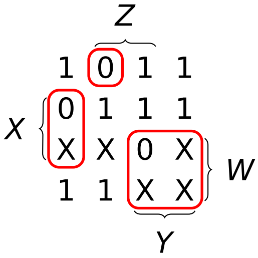
$$a = \overline{X}\,\overline{Z} + \overline{W}Y + \overline{W}XZ + W\overline{X}$$

$2 + 2 + 3 + 2 = 9$ 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:

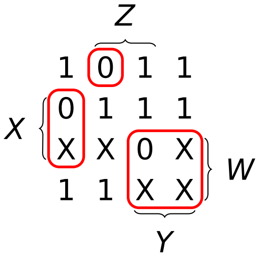
$$\overline{a} = \overline{W}\,\overline{X}\,\overline{Y}Z + X\overline{Y}\,\overline{Z} + WY$$

$4 + 3 + 2 = 9$ literals

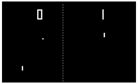
5 inv + 1 AND4 + 1 AND3 + 1 AND2 + 1 OR3

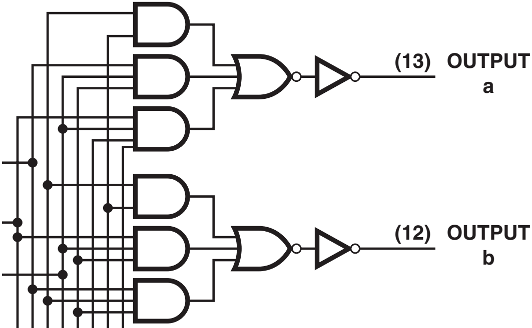
Karnaugh Map for Seg. a

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 chip with this:





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

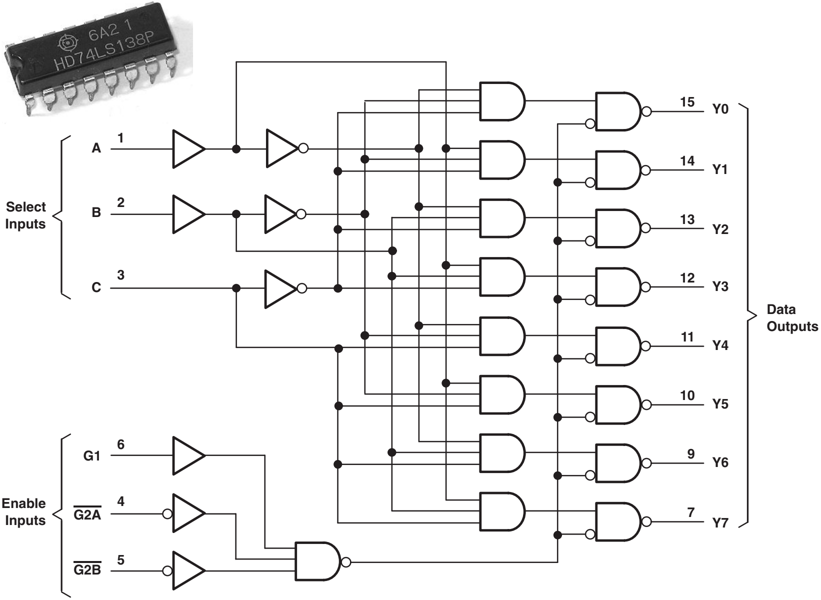
3-to-8 decoder

in	out
000	00000001
001	00000010
010	00000100
011	00001000
100	00010000
101	00100000
110	01000000
111	10000000

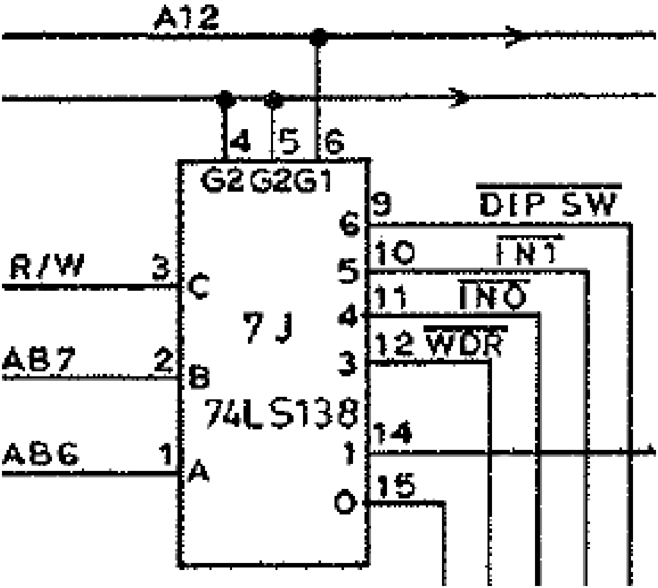
4-to-16 decoder

in	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



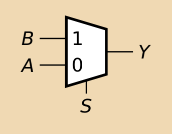
A ’138 Spotted in the Wild



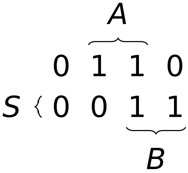
Pac-Man (Midway, 1980)

Multiplexers

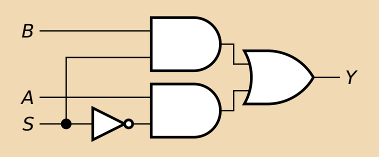
The Two-Input Multiplexer



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

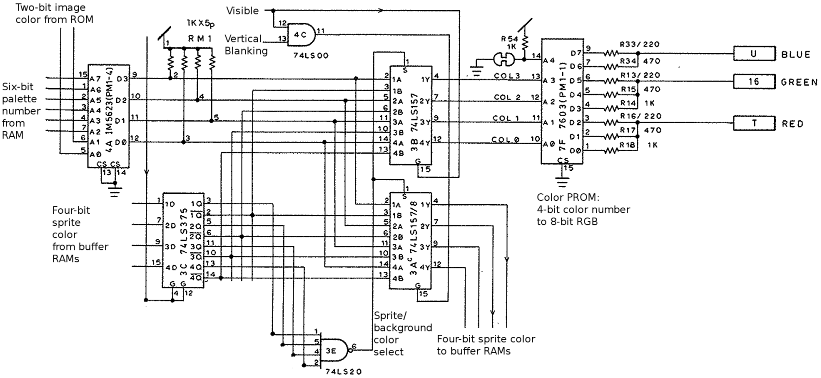


S	B	A	Y
0	X	0	0
0	X	1	1
1	0	X	0
1	1	X	1



S	Y
0	A
1	B

Two-input Muxes in the Wild

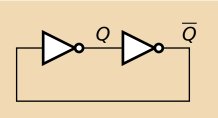


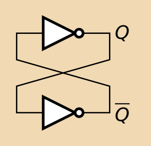
Quad 2-to-1 mux 3B selects color from a sprite or the background

Pac-Man (Midway, 1980)

State-Holding Elements

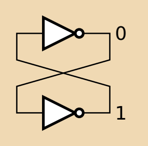
Bistable Elements

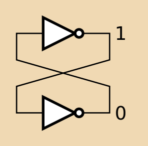




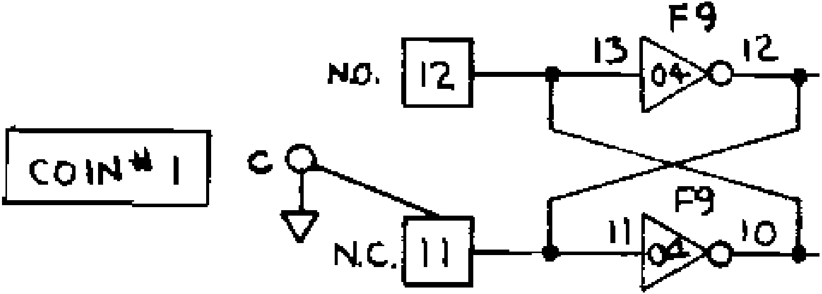
Equivalent circuits; right is more traditional.

Two stable states:





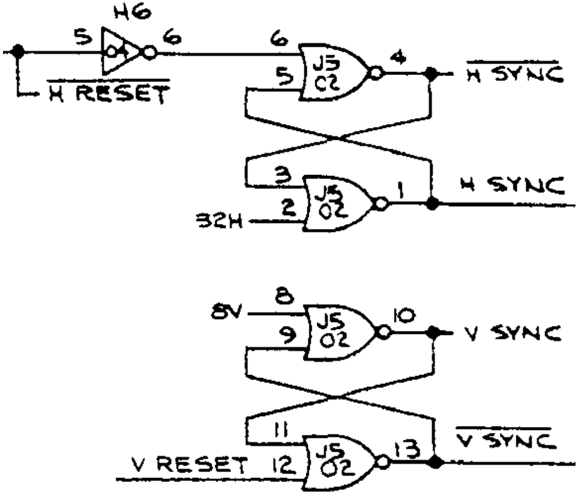
A Bistable in the Wild



This “debounces” the coin switch.

Breakout, Atari 1976.

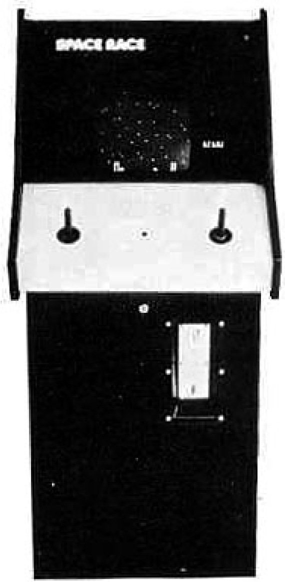
SR Latches in the Wild

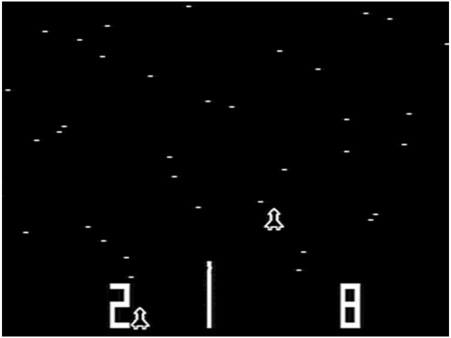


Generates horizontal and vertical synchronization waveforms from counter bits.

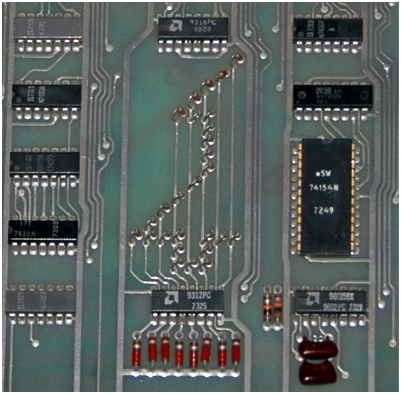
Stunt Cycle, Atari 1976.

Atari Space Race, 1973

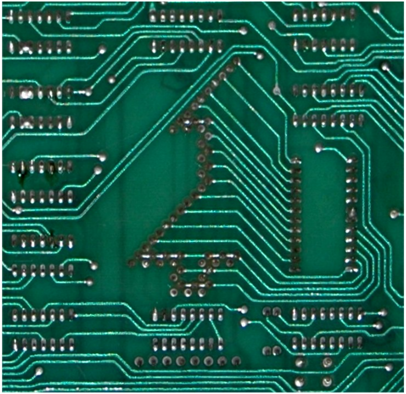




Atari Space Race PCB



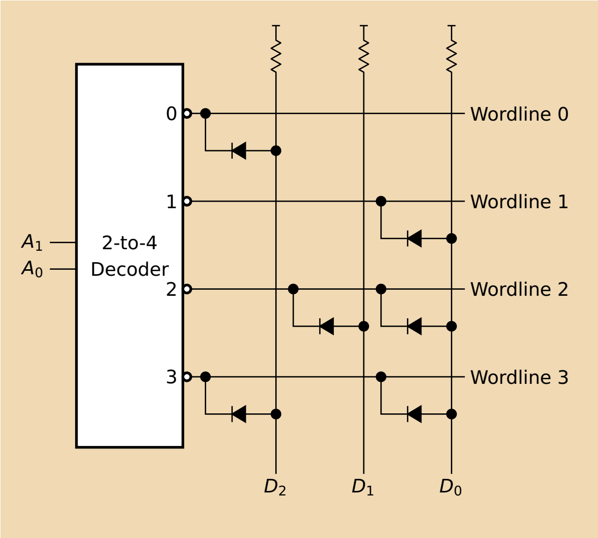
Front



Back (mirrored)

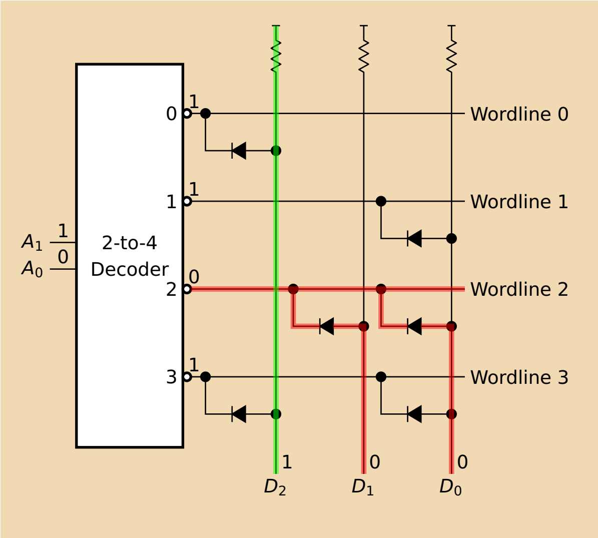
Implementing ROMs

Add.	Data
00	011
01	110
10	100
11	010

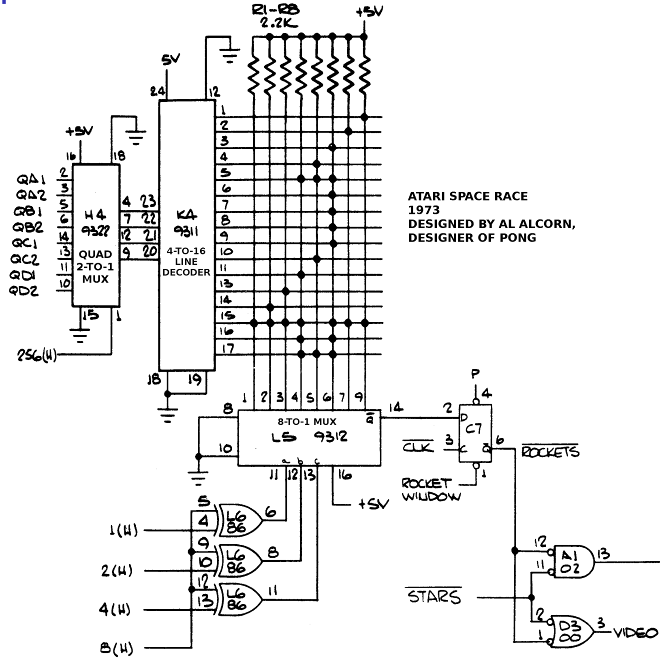


Implementing ROMs

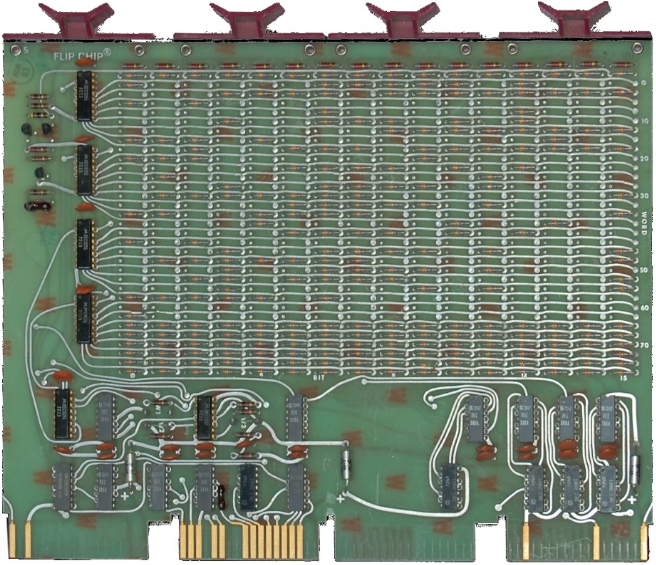
Add.	Data
00	011
01	110
10	100
11	010



Atari Space Race Schematic

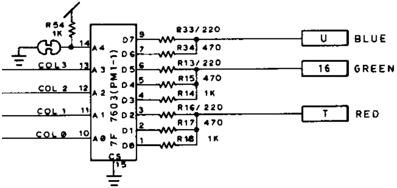


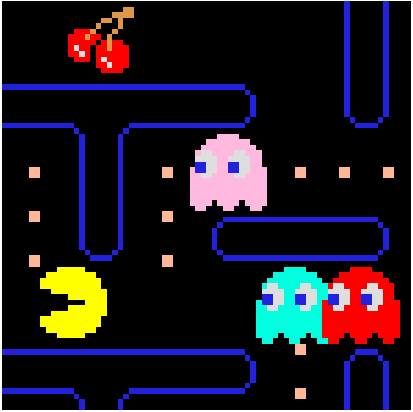
The 1971 DEC M792-YB Bootstrap Diode Matrix



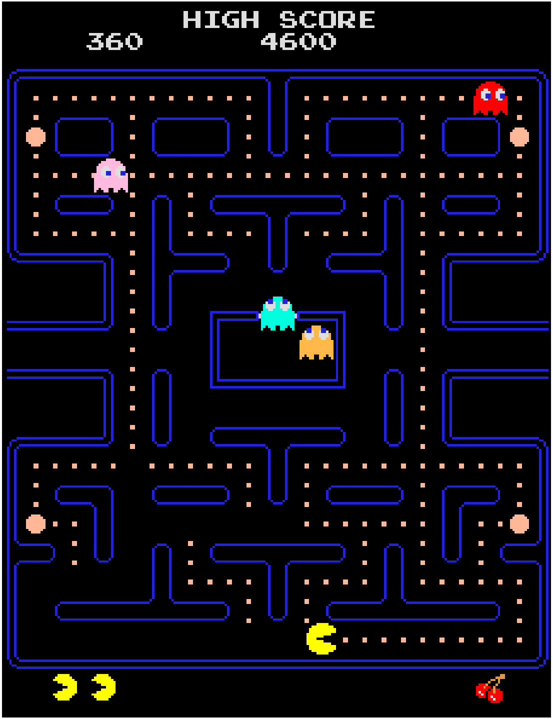
32-word, 16-bit (64-byte) ROM diode matrix

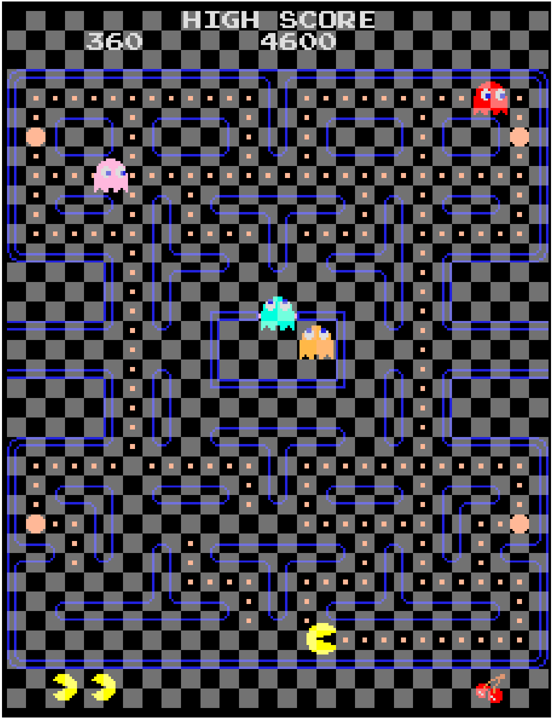
Color PROM in Pac-Man



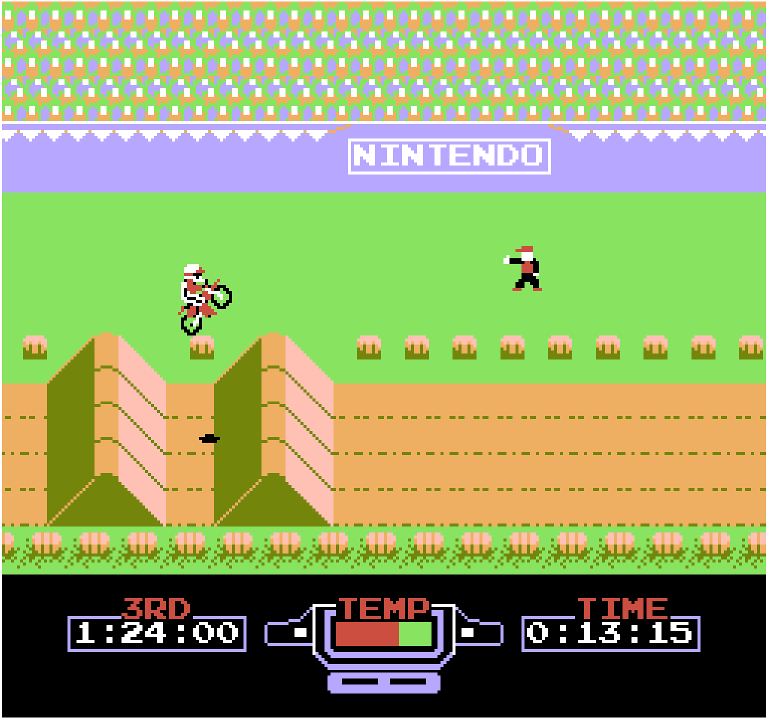


00	00
01	07
02	66
03	EF
04	00
05	F8
06	EA
07	6F
08	00
09	3F
0A	00
0B	C9
0C	38
0D	AA
0E	AF
0F	F6
10	00
⋮	⋮
1F	00


HIGH SCORE
360
4600

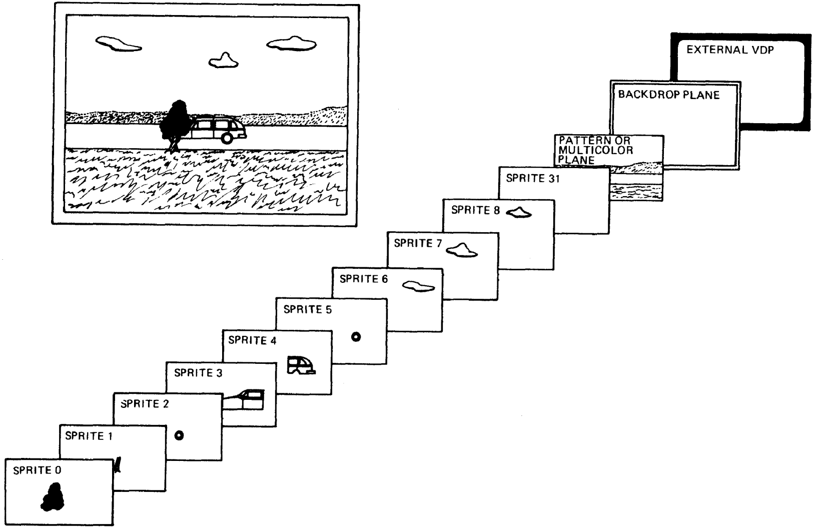

HIGH SCORE
360
4600


MARIO
000700
x01
WORLD
8-1
TIME
242

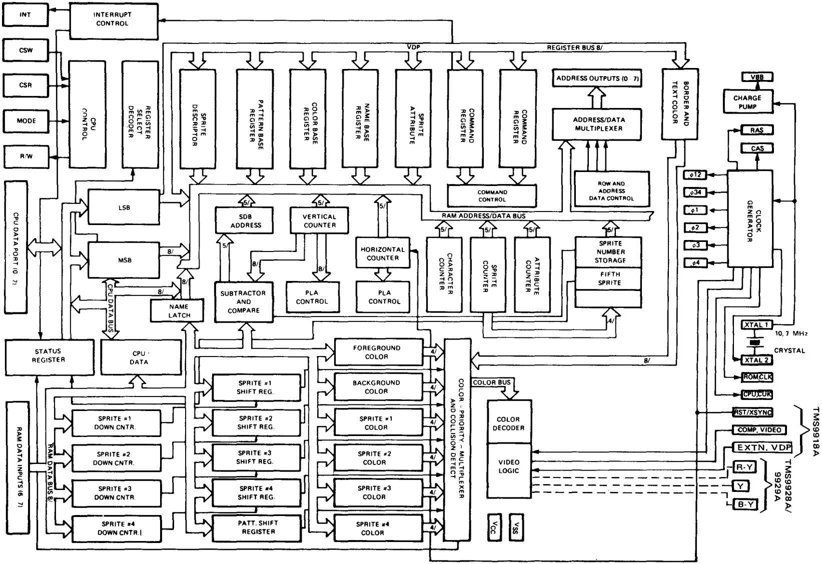

NINTENDO
3RD
1:24:00
TEMP
TIME
0:13:15


TUNNELS
OF
DOOM
©1982 TEXAS INSTRUMENTS

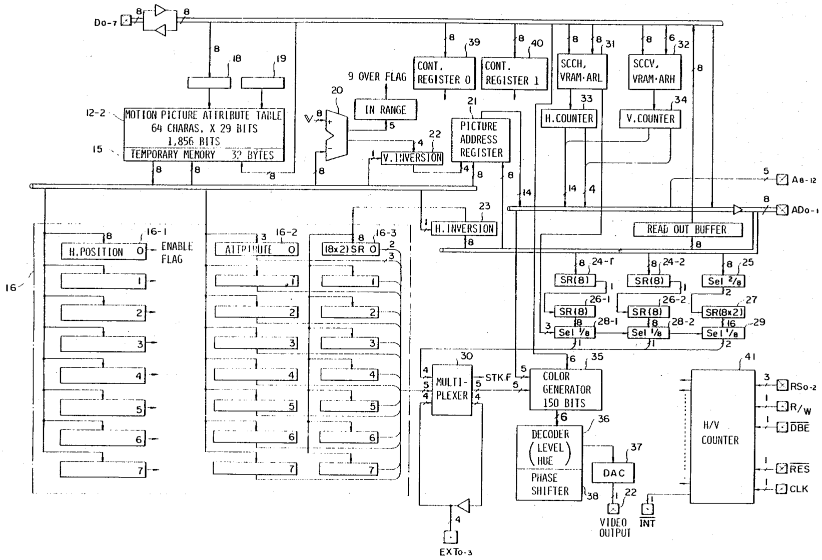
TMS9918 Video Display Processor



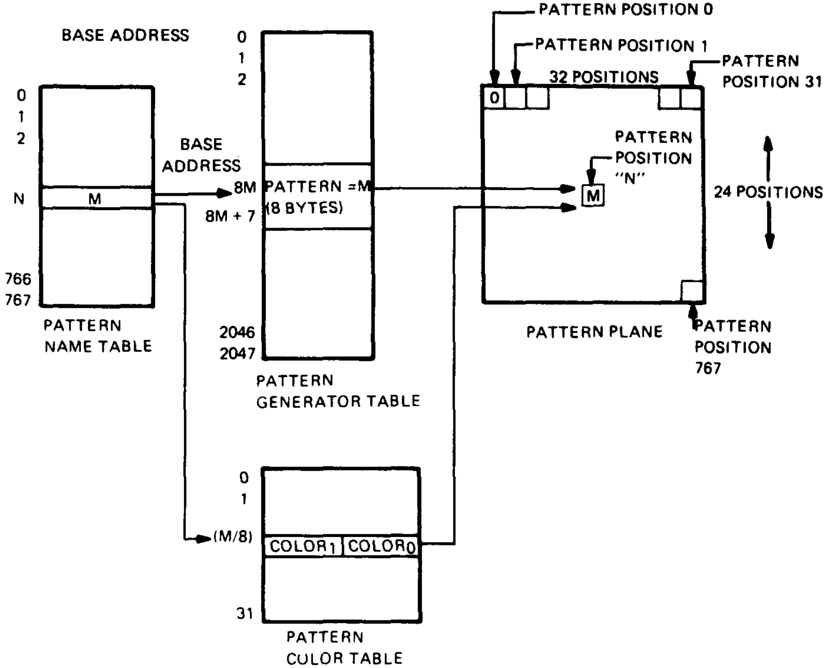
TMS9918 Video Display Processor



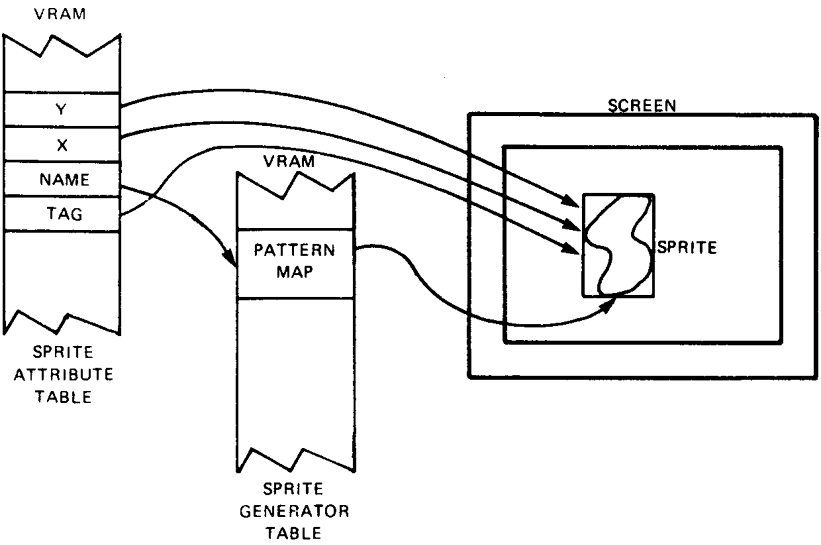
Nintendo NES/Famicom



TMS9918 Pattern Generation



TMS9918 Sprite Generation



TMS9918 Sprite Attribute Table Entry

BIT

BYTE	0	1	2	3	4	5	6	7
0	VERTICAL POSITION							
1	HORIZONTAL POSITION							
2	NAME							
3	EARLY CLOCK BIT	0	0	0	COLOR CODE			