

Understand Video Games; Understand Everything

Stephen A. Edwards

Columbia University



The Subject of this Lecture

0

The Subjects of this Lecture

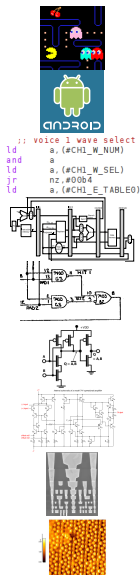
0

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

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.

GILDAN
ULTRA
COTTON

There are only 10 types
of people in the world:
Those who understand binary
and those who don't.

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, COBK.

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, \dots 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

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

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 —  — XY Y —
OR	$x \vee y$	$X + Y$	X —  — $X + Y$ Y —

Expressions to Schematics

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

x

y

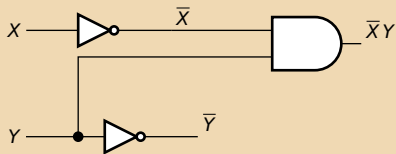
Expressions to Schematics

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



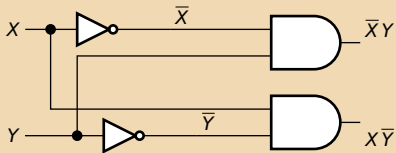
Expressions to Schematics

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



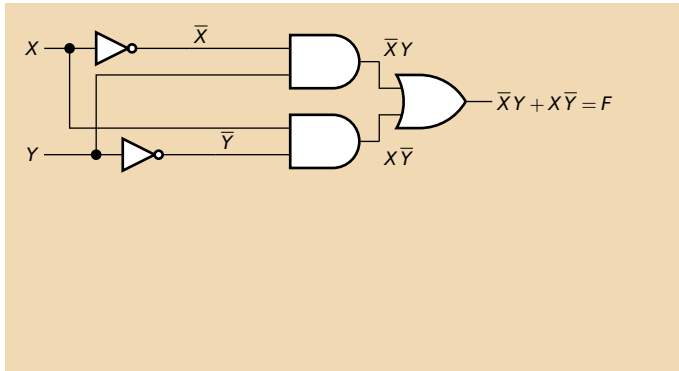
Expressions to Schematics

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



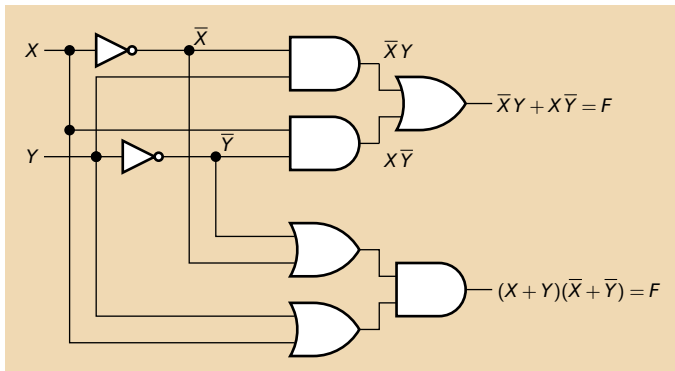
Expressions to Schematics

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



Expressions to Schematics

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



The Decimal Positional Numbering System

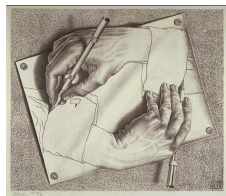


Ten figures: 0 1 2 3 4 5 6 7 8 9

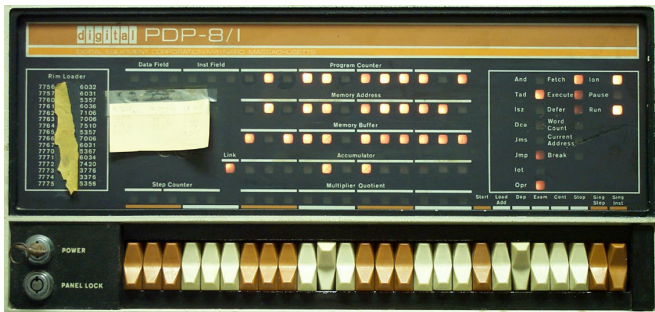
$$7 \times 10^2 + 3 \times 10^1 + 0 \times 10^0 = 730_{10}$$

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

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

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

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

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

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

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

Binary Addition Algorithm

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

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

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

Binary Addition Algorithm

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

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

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

Binary Addition Algorithm

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

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

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

Binary Addition Algorithm

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

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

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

Binary-Coded Decimal



thinkgeek.com

Humans prefer reading decimal numbers; computers prefer binary.

BCD is a compromise: every four bits represents a decimal digit.

Dec	BCD
0	0000 0000
1	0000 0001
2	0000 0010
⋮	⋮
8	0000 1000
9	0000 1001
10	0001 0000
11	0001 0001
⋮	⋮
18	0001 1000
19	0001 1001
20	0010 0000
⋮	⋮



Arithmetic Circuits

Arithmetic: Addition

Adding two one-bit numbers:

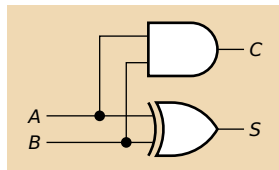
A and B

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



Male Adder

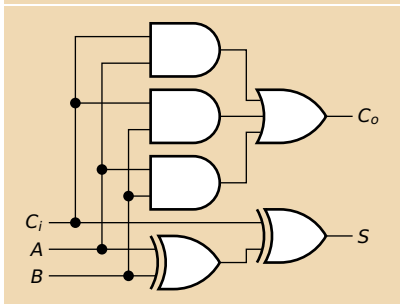
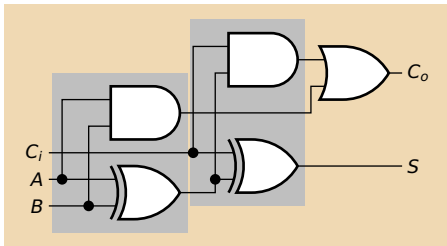
Full Adder

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

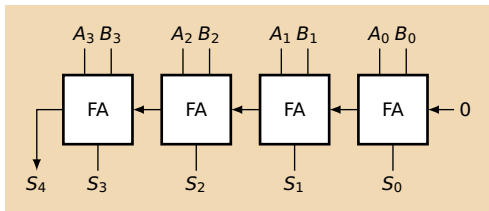
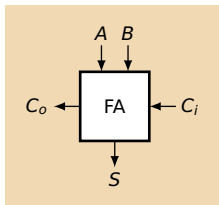
$$\begin{array}{r} 111000 \\ 111010 \\ + 11100 \\ \hline 1010110 \end{array}$$

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

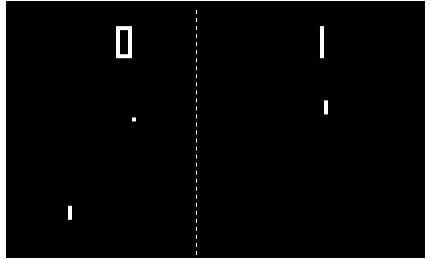
C_i	A	B	C_o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



A Four-Bit Ripple-Carry Adder



PONG



PONG, Atari 1973

Built from TTL logic gates; no computer, no software

Launched the video arcade game revolution

Horizontal Ball Control in PONG

<i>M</i>	<i>L</i>	<i>R</i>	<i>A</i>	<i>B</i>
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

<i>M</i>	<i>L</i>	<i>R</i>	<i>A</i>	<i>B</i>
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:

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

$$B = \bar{M}\bar{L}R + \bar{M}L\bar{R} + ML\bar{R}$$

3 inv + 4 AND3 + 1 OR2 + 1 OR3

Horizontal Ball Control in PONG

<i>M</i>	<i>L</i>	<i>R</i>	<i>A</i>	<i>B</i>
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

<i>M</i>	<i>L</i>	<i>R</i>	<i>A</i>	<i>B</i>
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

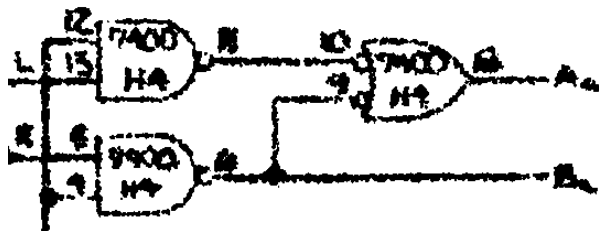
Being even more clever:

$$A = M$$

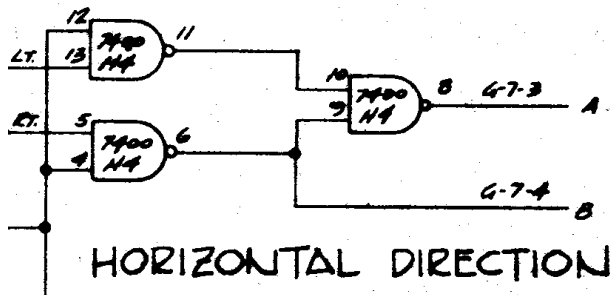
$$B = \overline{MR}$$

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

<i>M</i>	<i>L</i>	<i>R</i>	<i>A</i>	<i>B</i>
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

<i>M</i>	<i>L</i>	<i>R</i>	<i>A</i>	<i>B</i>
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

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

1	1
X	X

Karnaugh Maps

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

<i>M</i>	<i>L</i>	<i>R</i>	<i>A</i>	<i>B</i>
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

<i>M</i>	<i>L</i>	<i>R</i>	<i>A</i>	<i>B</i>
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

<i>M</i>	<i>L</i>	<i>R</i>	<i>A</i>	<i>B</i>
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

<i>M</i>	<i>L</i>	<i>R</i>	<i>A</i>	<i>B</i>
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

<i>M</i>	<i>L</i>	<i>R</i>	<i>A</i>	<i>B</i>	
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	0	0	X	X	
1	0	1	1	0	

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

1	1
X	X

Karnaugh Maps

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

<i>M</i>	<i>L</i>	<i>R</i>	<i>A</i>	<i>B</i>		
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

<i>M</i>	<i>L</i>	<i>R</i>	<i>A</i>	<i>B</i>
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

<i>M</i>	<i>L</i>	<i>R</i>	<i>A</i>	<i>B</i>
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_0	0_1	X_0	X_1
0_0	1_1	0_1	0_X	1_X
1_1	1_1	0_1	1_X	1_X
1_1	0_0	0_1	X_1	X_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

<i>M</i>	<i>L</i>	<i>R</i>	<i>A</i>	<i>B</i>
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

<i>M</i>	<i>L</i>	<i>R</i>	<i>A</i>	<i>B</i>
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

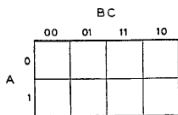
M. KARNAUGH

NONMEMBER AIEE

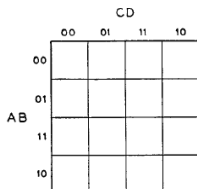
THE SEARCH for simple abstract techniques to be applied to the design of switching systems is still, despite some recent advances, in its early stages. The problem in this area which has been attacked most energetically is that of the synthesis of efficient combinational that is, nonsequential, logic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



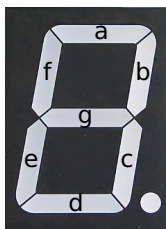
(A)



(B)

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

The Seven-Segment Decoder Example



<i>W</i>	<i>X</i>	<i>Y</i>	<i>Z</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>
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

		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

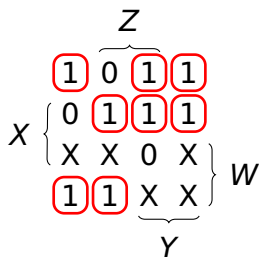
Cover all the 1's and none of the 0's using **as few literals** (gate inputs) as possible.

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.

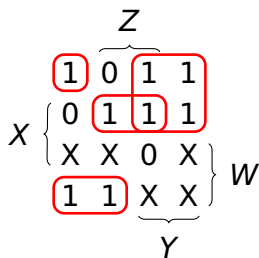
$$\begin{aligned}
 a = & \overline{W}\overline{X}\overline{Y}\overline{Z} + \overline{W}\overline{X}YZ + \overline{W}\overline{X}Y\overline{Z} + \\
 & \overline{W}X\overline{Y}Z + \overline{W}XYZ + \overline{W}XY\overline{Z} + \\
 & W\overline{X}\overline{Y}\overline{Z} + W\overline{X}\overline{Y}Z
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

$8 \times 4 = 32$ 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:

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

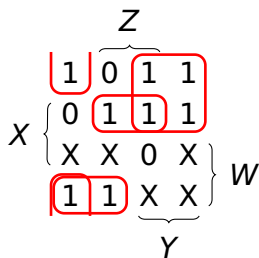
$$a = \overline{W}\overline{X}\overline{Y}\overline{Z} + \overline{W}Y + \overline{W}XZ + W\overline{X}\overline{Y}$$

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

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

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.

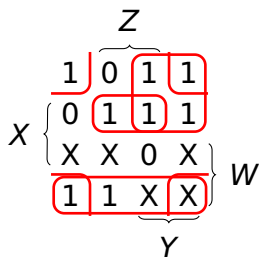
$$a = \overline{X}\overline{Y}\overline{Z} + \overline{W}Y + \overline{W}XZ + W\overline{X}\overline{Y}$$

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:

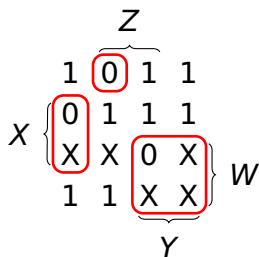
$$a = \bar{X}\bar{Z} + \bar{W}Y + \bar{W}XZ + W\bar{X}$$

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

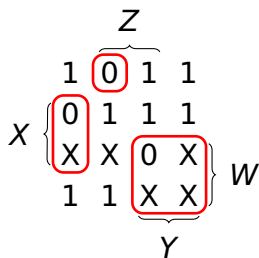
$$\bar{a} = \bar{W}\bar{X}\bar{Y}Z + X\bar{Y}\bar{Z} + WY$$

4 + 3 + 2 = 9 literals

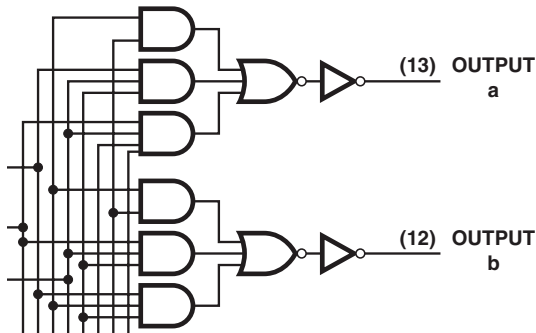
5 inv + 1 AND4 + 1 AND3 + 1 AND2
+ 1 OR3

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 chip with this:



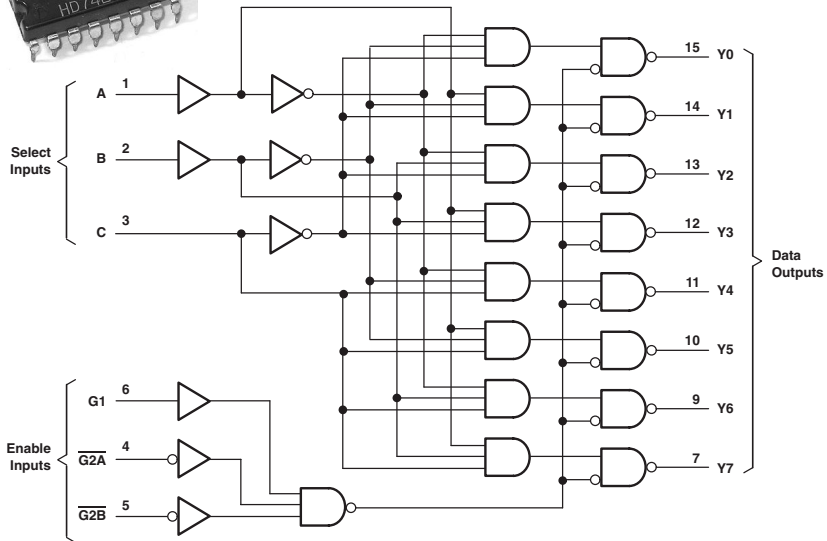
Decoders

Input: n -bit binary number

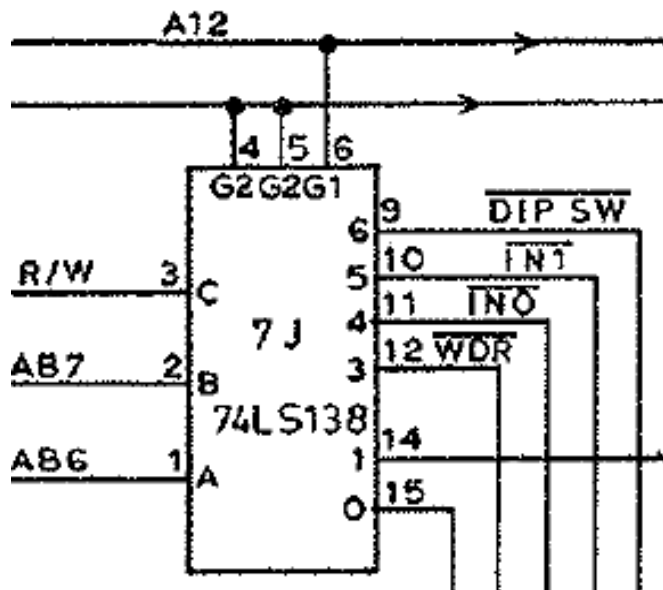
Output: 1-of- 2^n one-hot code

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



A '138 Spotted in the Wild

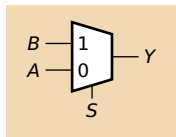


Pac-Man (Midway, 1980)

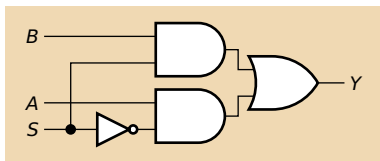
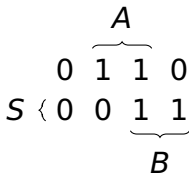


Multiplexers

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



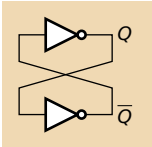
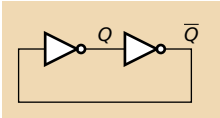
S	B	A	Y
0	X	0	0
0	X	1	1
1	0	X	0
1	1	X	1

S	Y
0	A
1	B



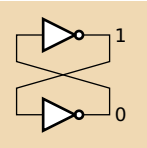
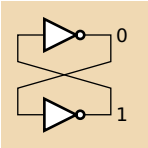
State-Holding Elements

Bistable Elements

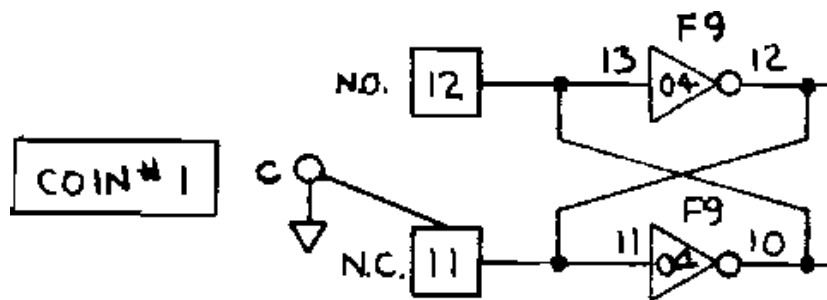


Equivalent circuits; right is more traditional.

Two stable states:



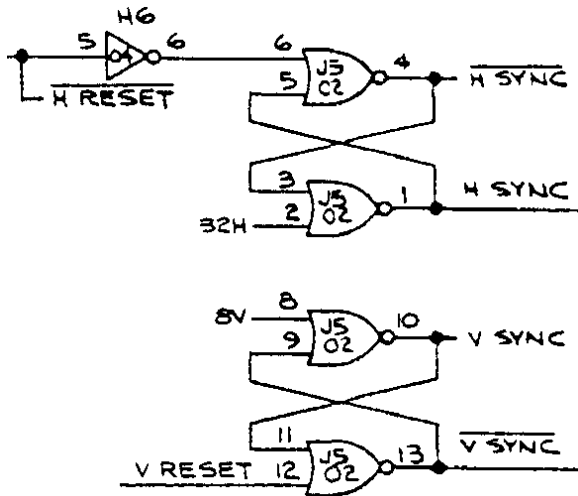
A Bistable in the Wild



This "debounces" the coin switch.

Breakout, Atari 1976.

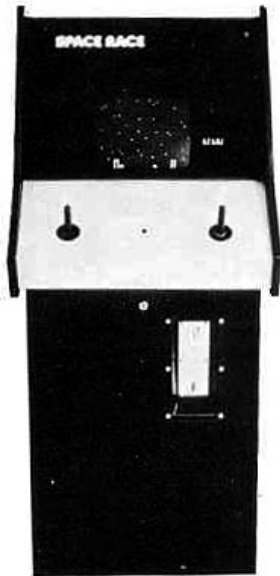
SR Latches in the Wild



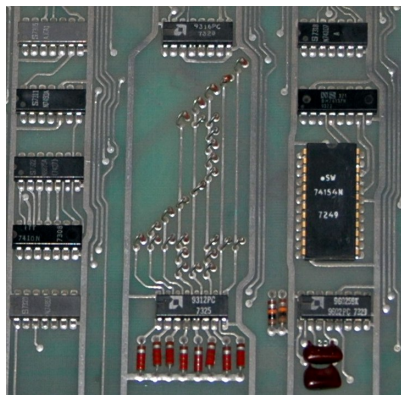
Generates horizontal and vertical synchronization waveforms from counter bits.

Stunt Cycle, Atari 1976.

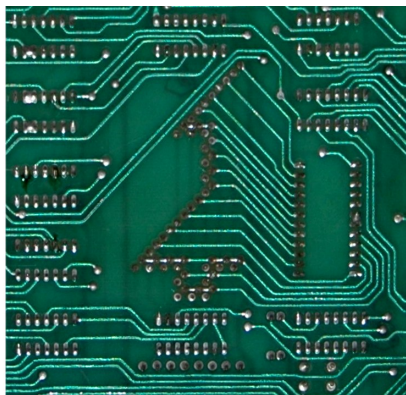
Atari Space Race, 1973



Atari Space Race PCB



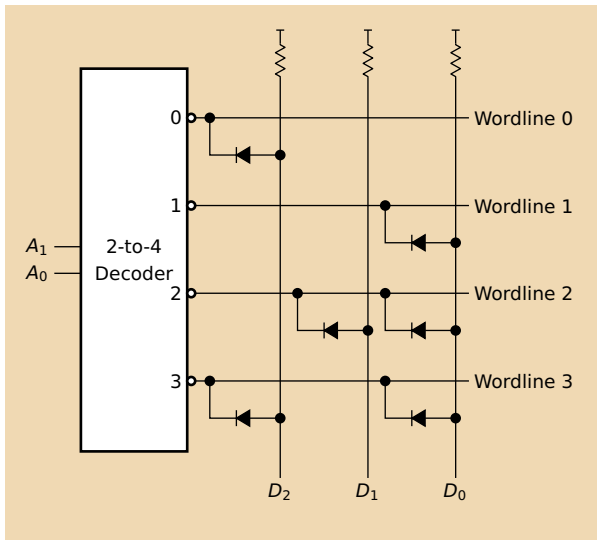
Front



Back (mirrored)

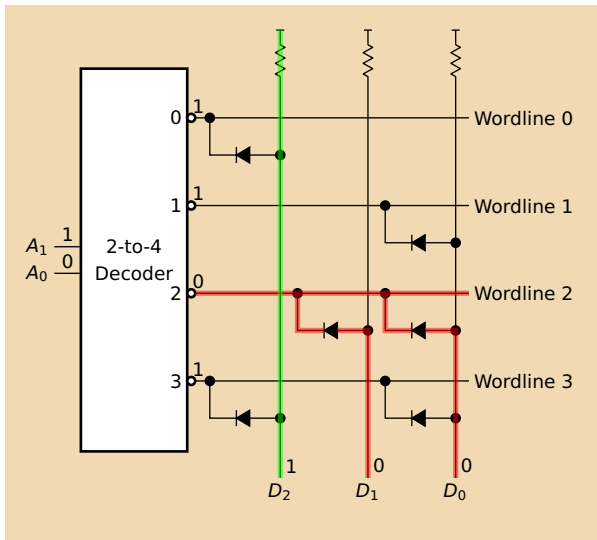
Implementing ROMs

Add. Data	
00	011
01	110
10	100
11	010

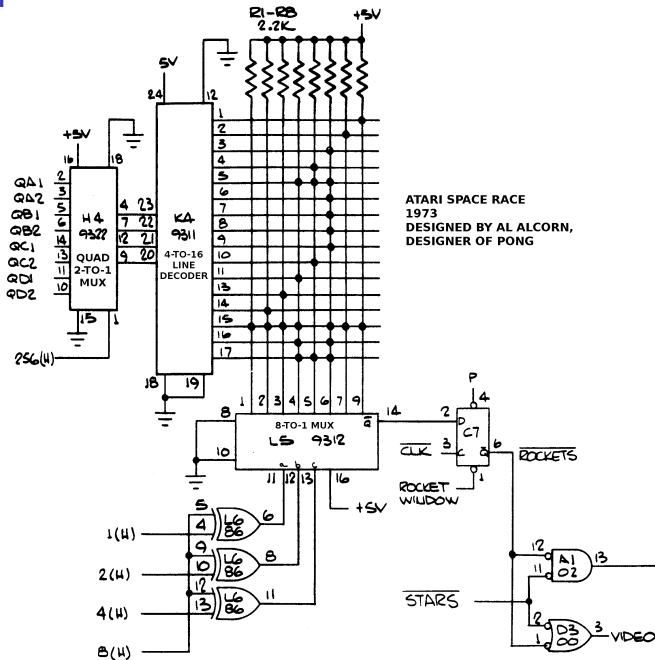


Implementing ROMs

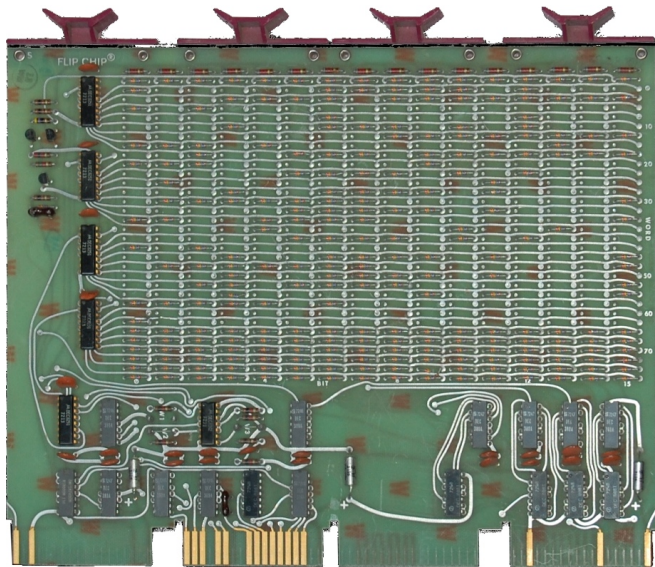
Add. Data	
00	011
01	110
10	100
11	010



Atari Space Race Schematic

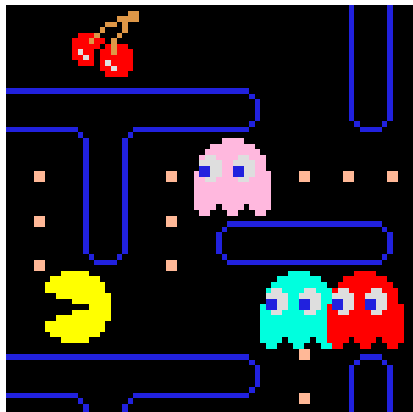
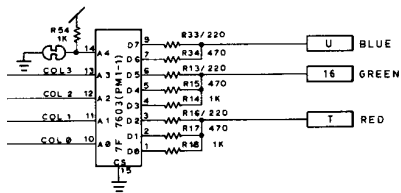


The 1971 DEC M792-YB Bootstrap Diode Matrix



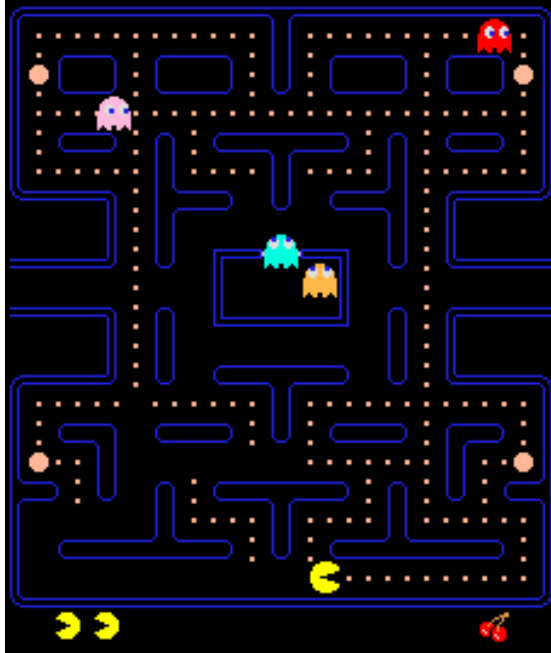
32-word, 16-bit (64-byte) ROM diode matrix

Color PROM in Pac-Man

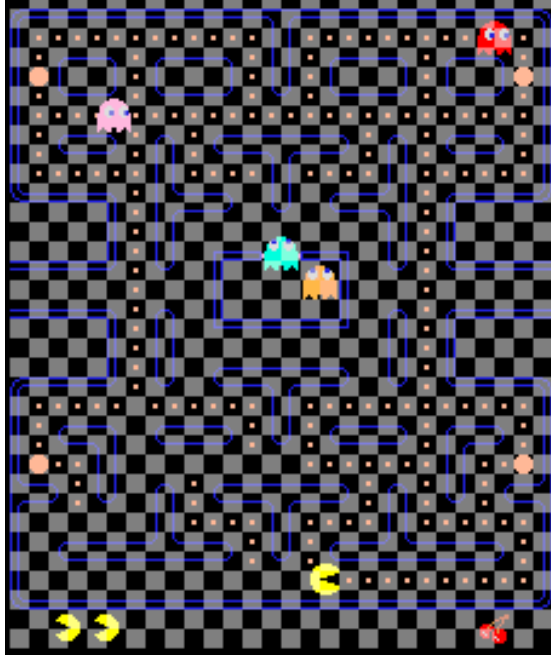


00	00	Black
01	07	Red
02	66	Brown
03	EF	Pink
04	00	Black
05	F8	Cyan
06	EA	Light Blue
07	6F	Orange
08	00	Black
09	3F	Yellow
0A	00	Black
0B	C9	Blue
0C	38	Bright Green
0D	AA	Teal
0E	AF	Light Orange
0F	F6	Light Purple
10	00	Black
:	:	
1F	00	Black

360 HIGH SCORE 4600



360 HIGH SCORE 4600



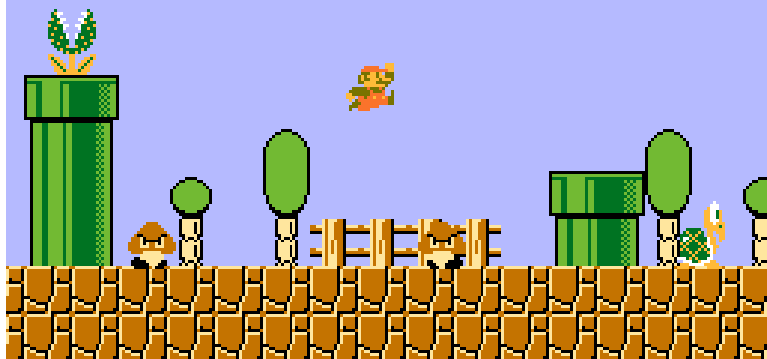
MARIO
000700

0 x 01

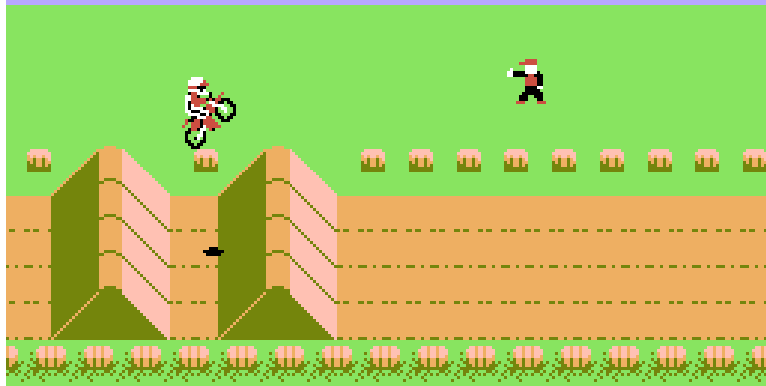
WORLD
8-1

TIME
242

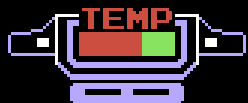
0 0



NINTENDO



3RD
1:24:00



TIME
0:13:15

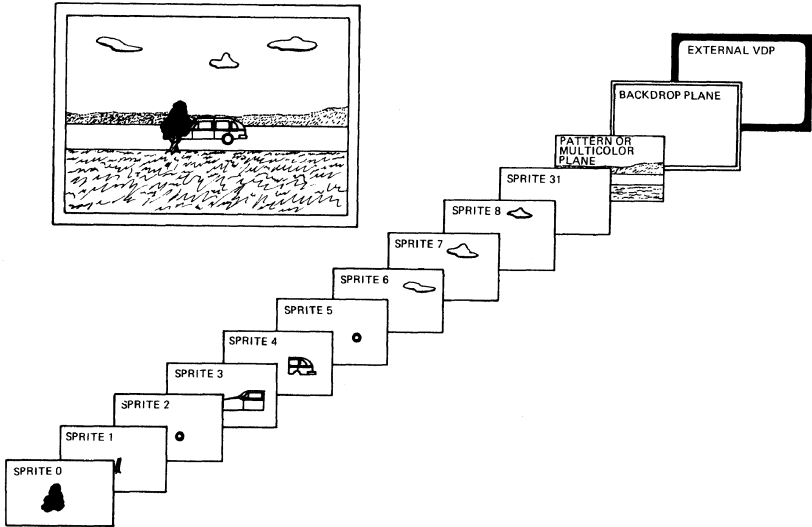
TUNNELS



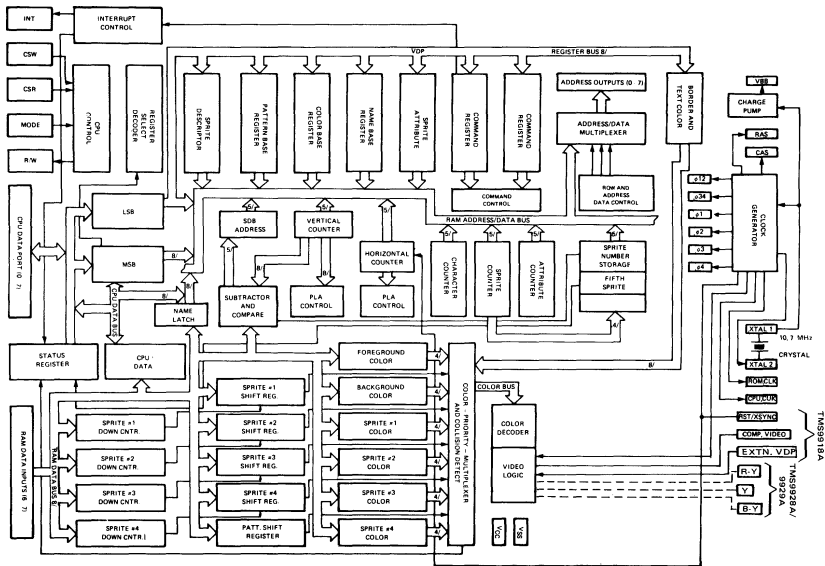
©1982

TEXAS INSTRUMENTS

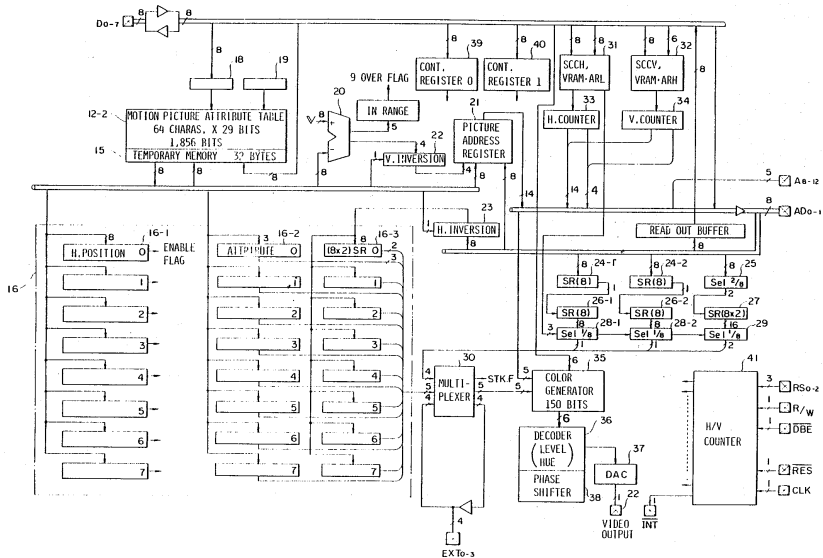
TMS9918 Video Display Processor



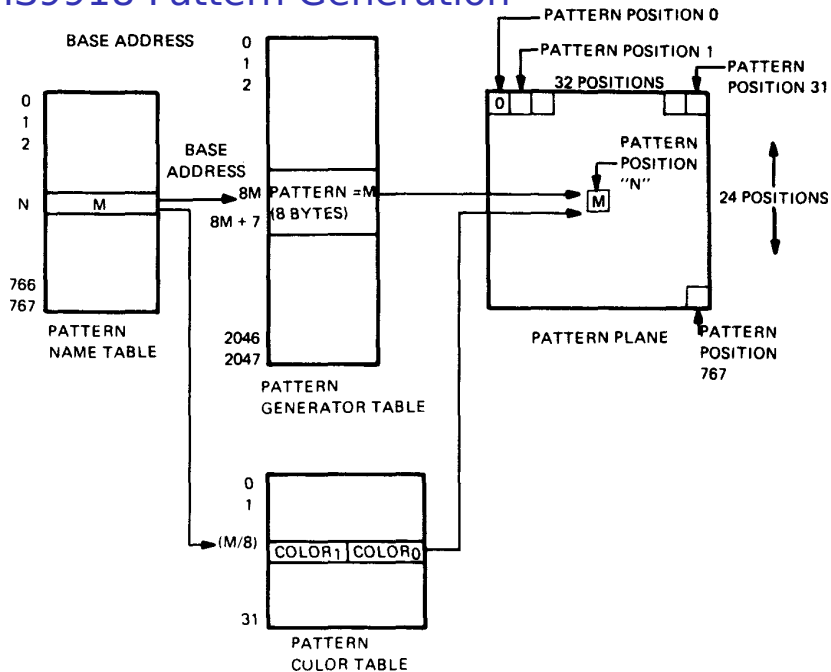
TMS9918 Video Display Processor



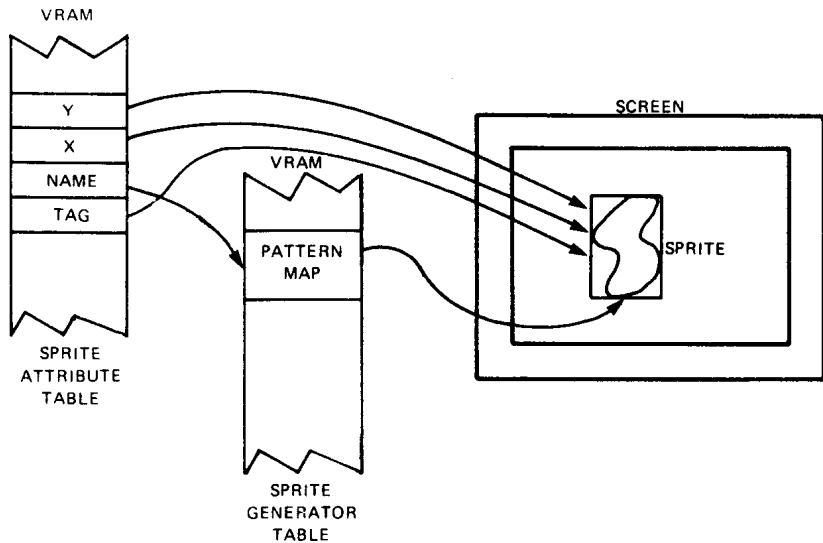
Nintendo NES/Famicom



TMS9918 Pattern Generation



TMS9918 Sprite Generation



TMS9918 Sprite Attribute Table Entry

