Introduction to CMOS VLSI Design

Lecture 14: CAMs, ROMs, and PLAs

David Harris

Harvey Mudd College

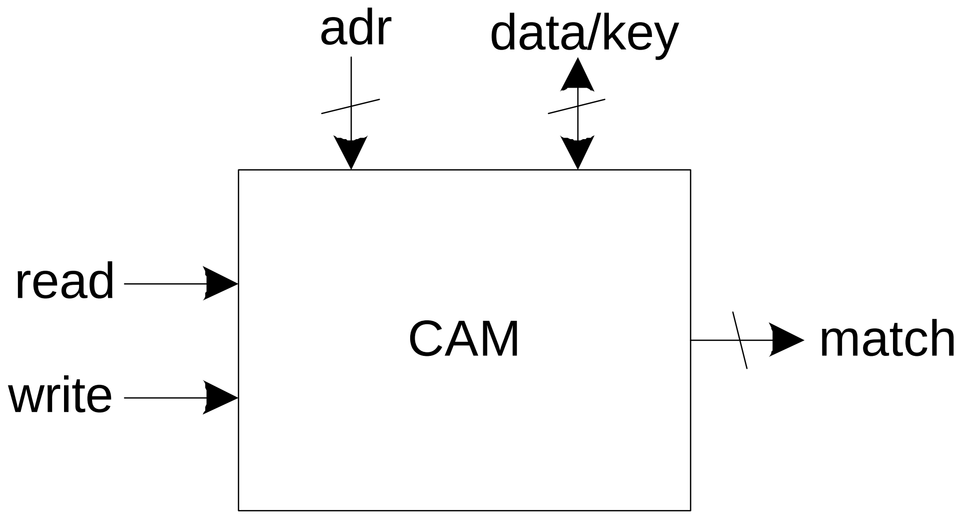
Spring 2004

Outline

- Content-Addressable Memories
- Read-Only Memories
- Programmable Logic Arrays

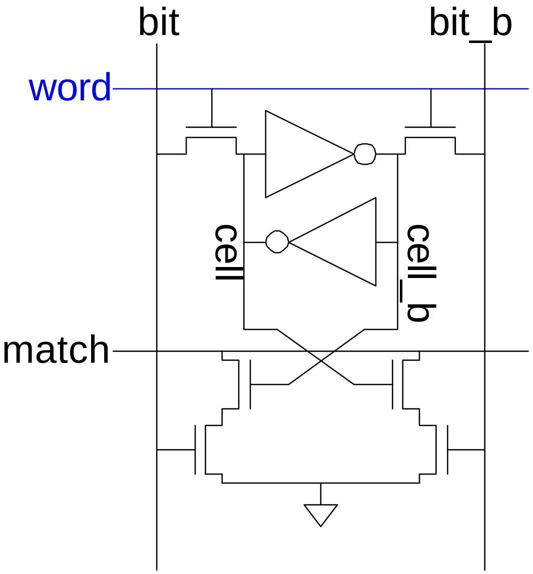
CAMs

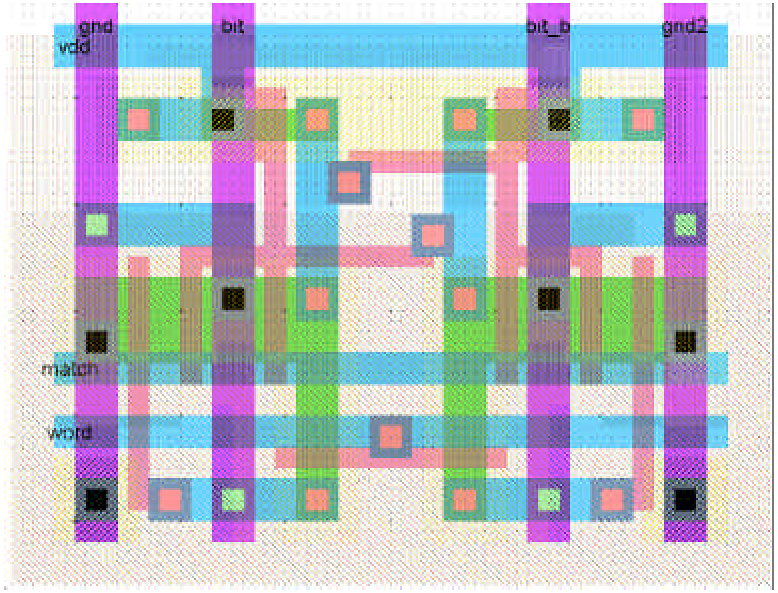
- ❑ Extension of ordinary memory (e.g. SRAM)
  - – Read and write memory as usual
  - – Also *match* to see which words contain a *key*



10T CAM Cell

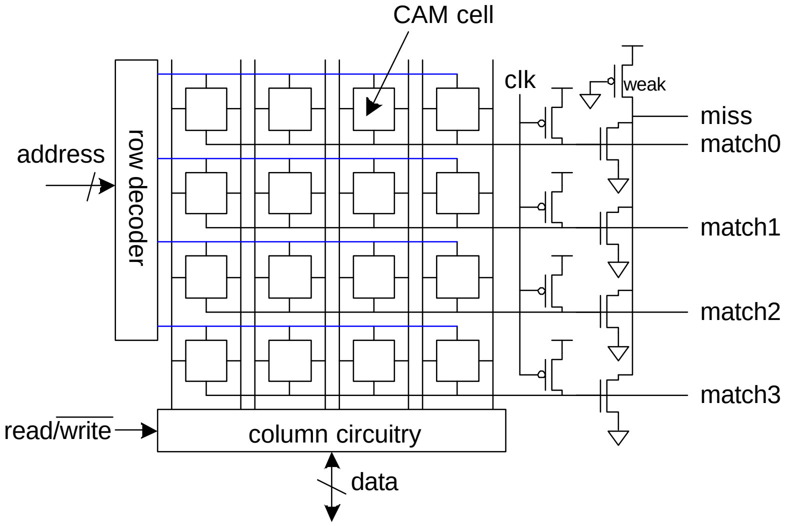
- Add four match transistors to 6T SRAM
  - – 56 x 43 λ unit cell





CAM Cell Operation

- ❑ Read and write like ordinary SRAM
- ❑ For matching:
  - – Leave wordline low
  - – Precharge matchlines
  - – Place key on bitlines
  - – Matchlines evaluate
- ❑ Miss line
  - – Pseudo-nMOS NOR of match lines
  - – Goes high if no words match



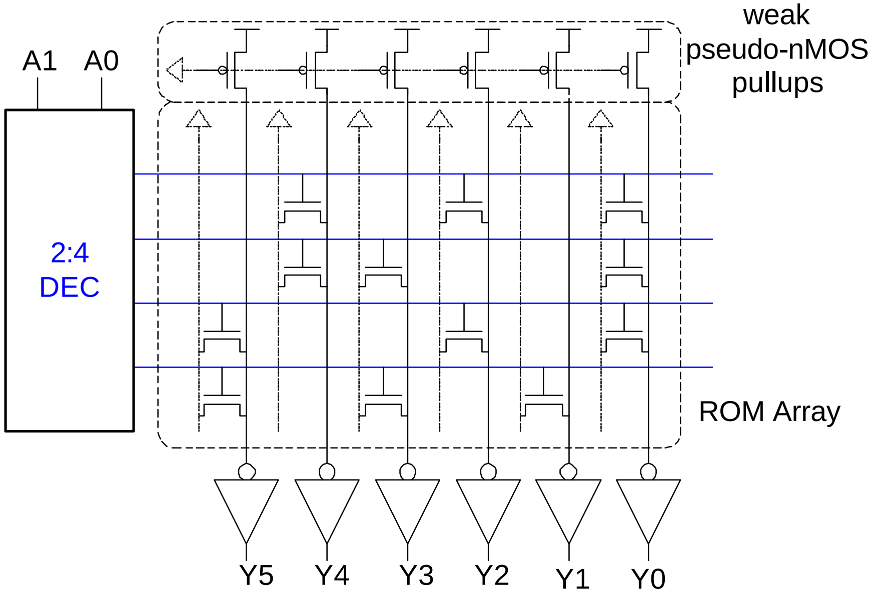
Read-Only Memories

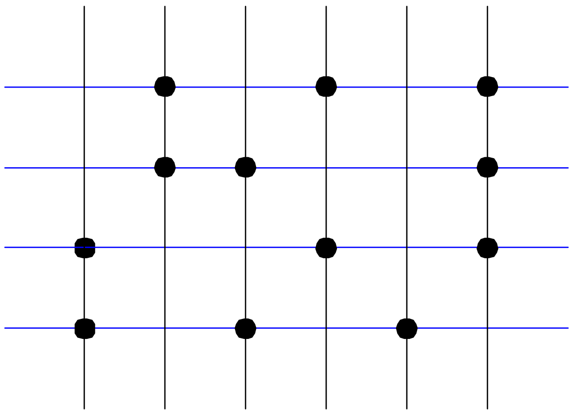
- ❑ Read-Only Memories are nonvolatile
  - – Retain their contents when power is removed
- ❑ Mask-programmed ROMs use one transistor per bit
  - – Presence or absence determines 1 or 0

ROM Example

- 4-word x 6-bit ROM
  - Represented with dot diagram
  - Dots indicate 1's in ROM

```
Word 0: 010101
Word 1: 011001
Word 2: 100101
Word 3: 101010
```

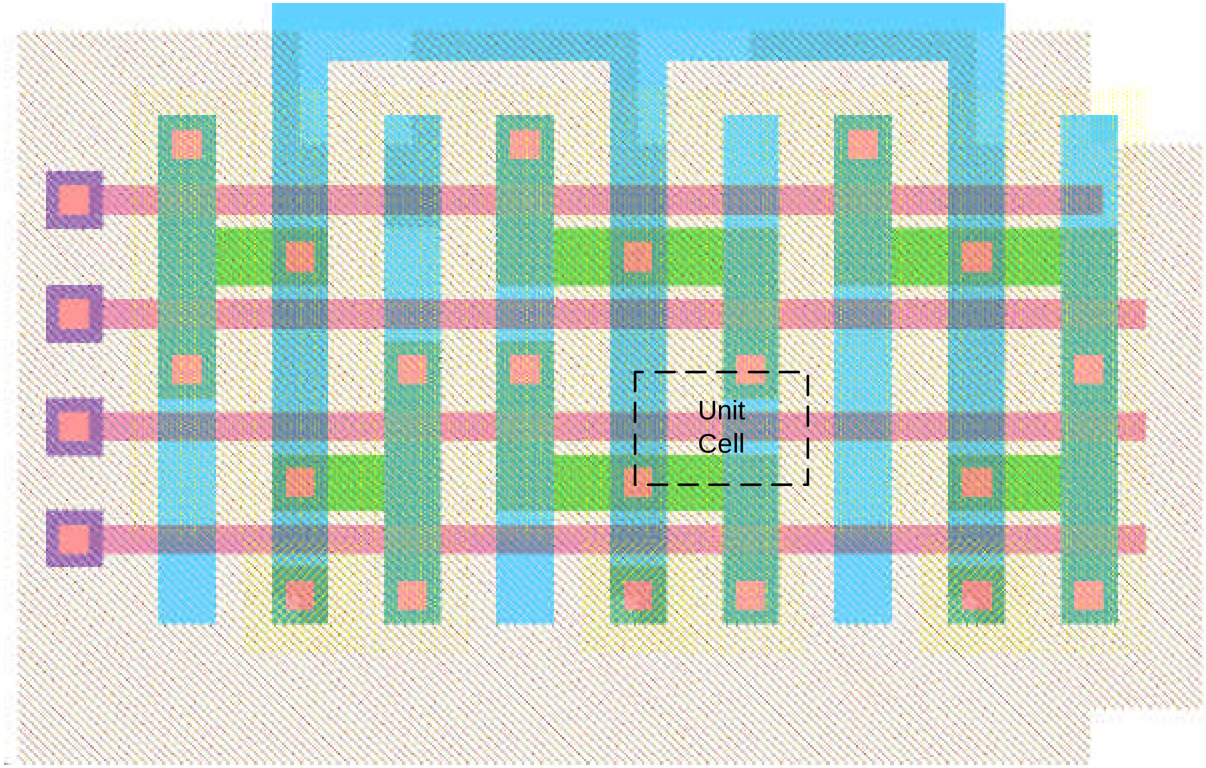




Looks like 6 4-input pseudo-nMOS NORs

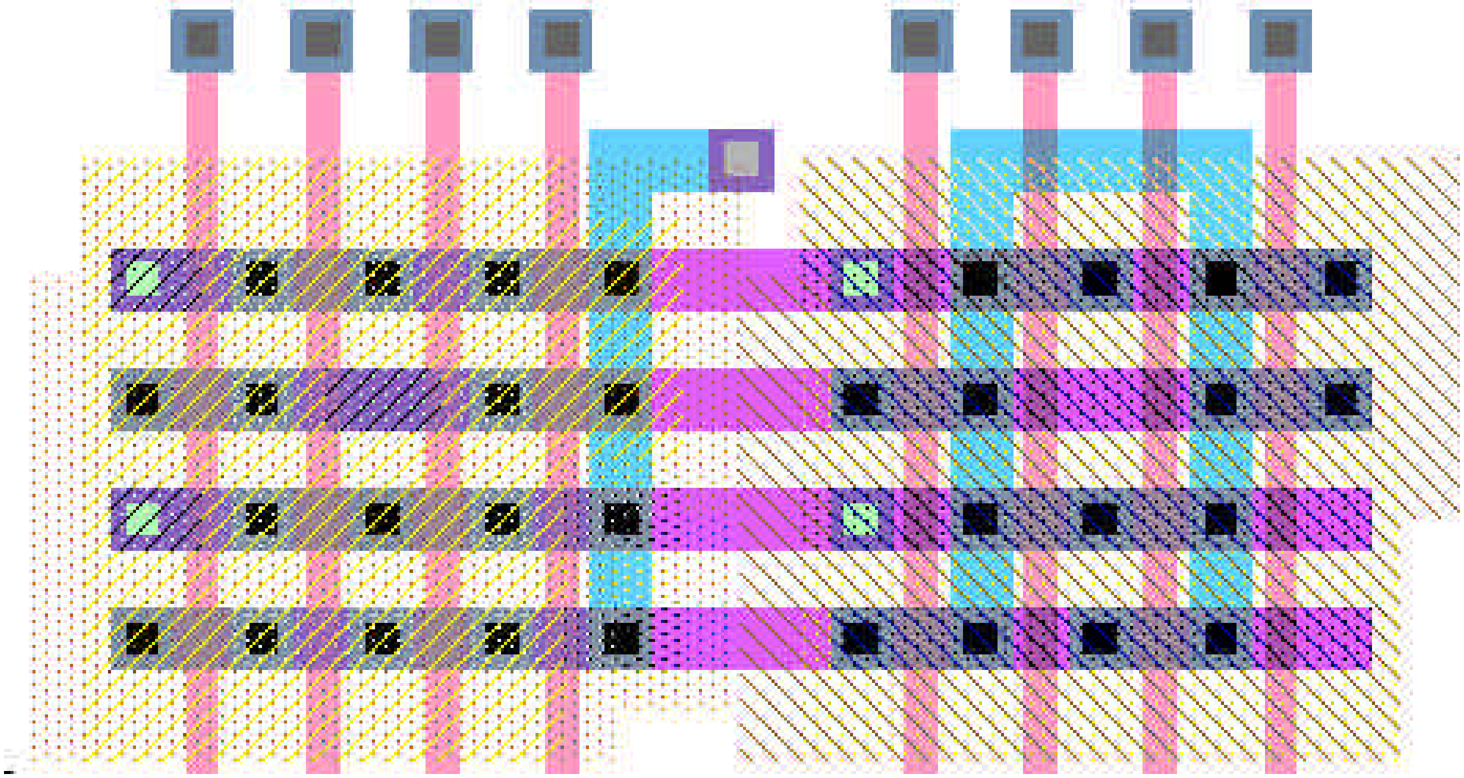
ROM Array Layout

- ❑ Unit cell is 12 x 8 λ (about 1/10 size of SRAM)

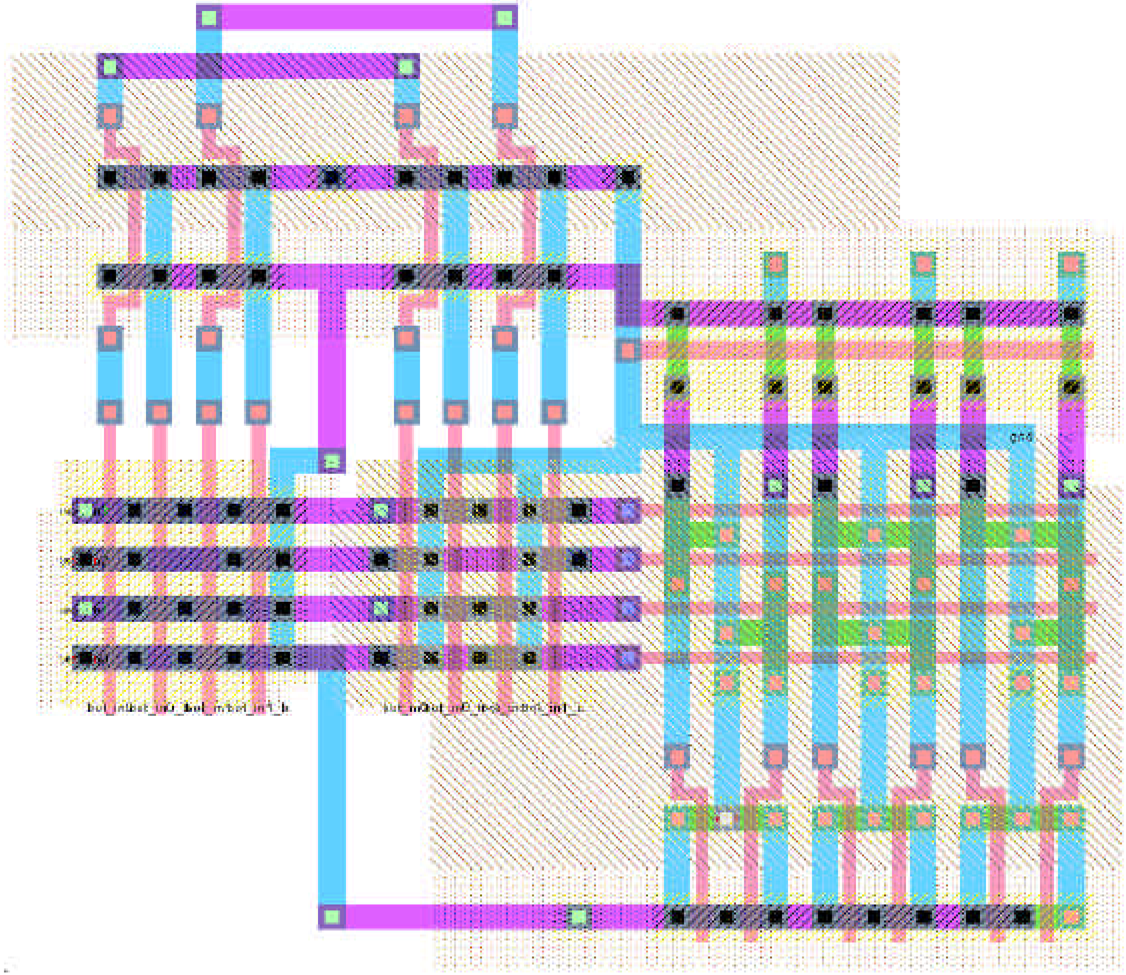


Row Decoders

- ❑ ROM row decoders must pitch-match with ROM
  - – Only a single track per word!

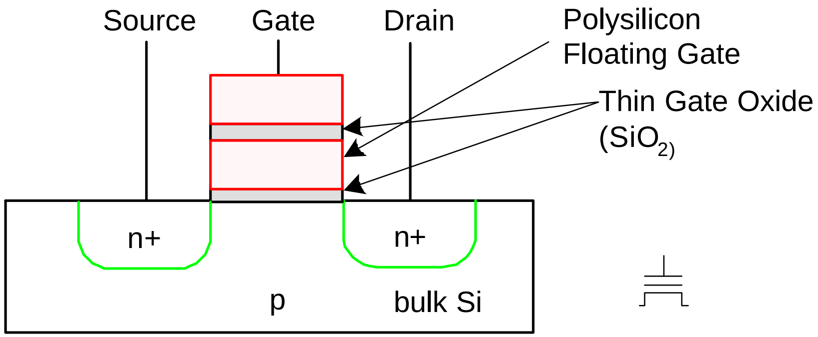


Complete ROM Layout



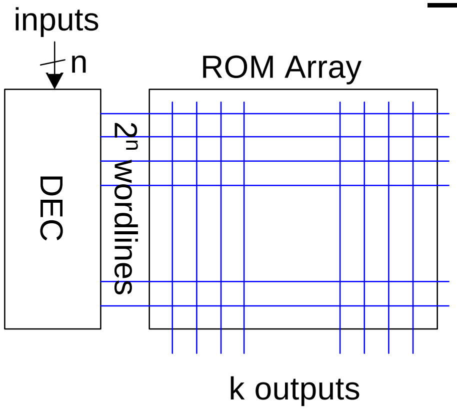
PROMs and EPROMs

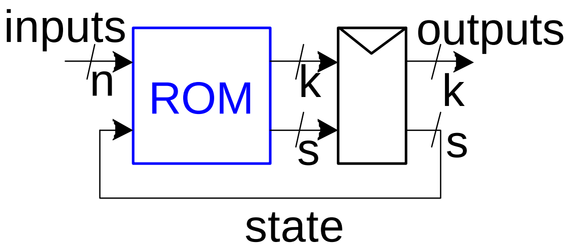
- ❑ Programmable ROMs
  - – Build array with transistors at every site
  - – Burn out fuses to disable unwanted transistors
- ❑ Electrically Programmable ROMs
  - – Use floating gate to turn off unwanted transistors
  - – EPROM, EEPROM, Flash



Building Logic with ROMs

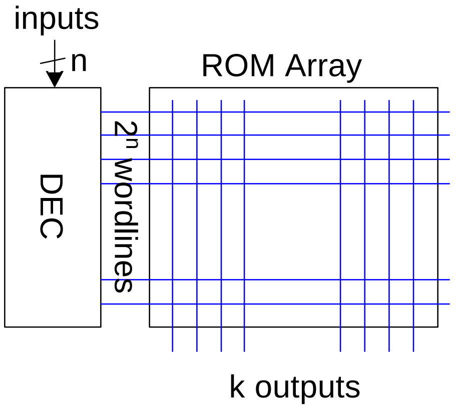
- ❑ Use ROM as lookup table containing truth table
  - – n inputs, k outputs requires __ words x __ bits
  - – Changing function is easy – reprogram ROM
- ❑ Finite State Machine
  - – n inputs, k outputs, s bits of state
  - – Build with _________ bit ROM and ____ bit reg

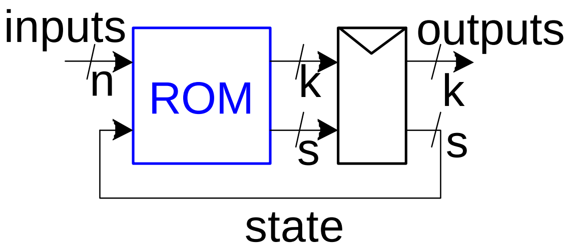




Building Logic with ROMs

- Use ROM as lookup table containing truth table
  - n inputs, k outputs requires 2^n words x k bits
  - Changing function is easy – reprogram ROM
- Finite State Machine
  - n inputs, k outputs, s bits of state
  - Build with 2^{n+s} x (k+s) bit ROM and (k+s) bit reg

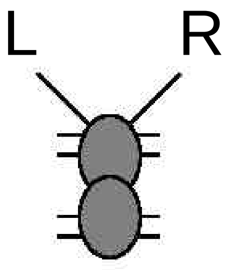




Example: RoboAnt

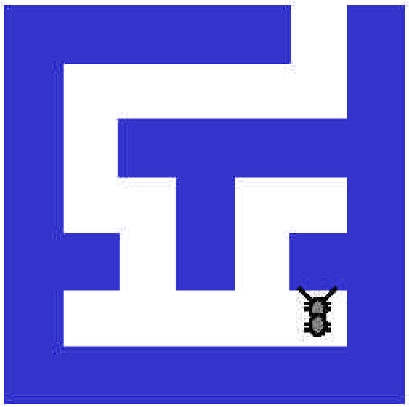
Let’s build an Ant

Sensors: Antennae
(L,R) – 1 when in contact



Actuators: Legs
Forward step F
Ten degree turns TL, TR

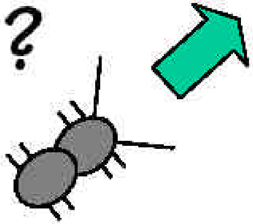
Goal: make our ant smart enough to get out of a maze



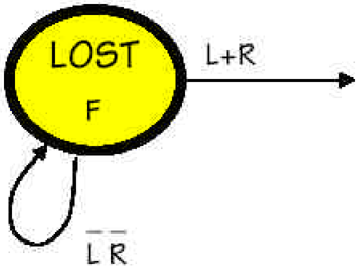
Strategy: keep right antenna on wall

(RoboAnt adapted from MIT 6.004 2002 OpenCourseWare by Ward and Terman)

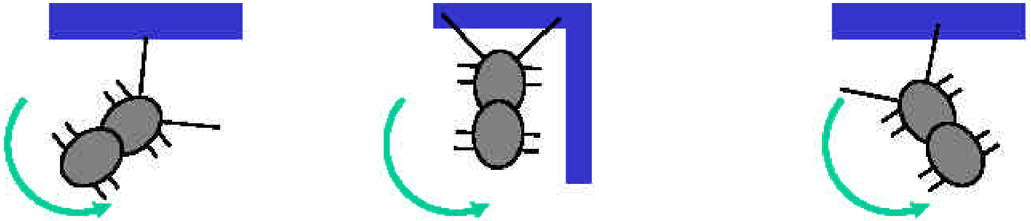
Lost in space



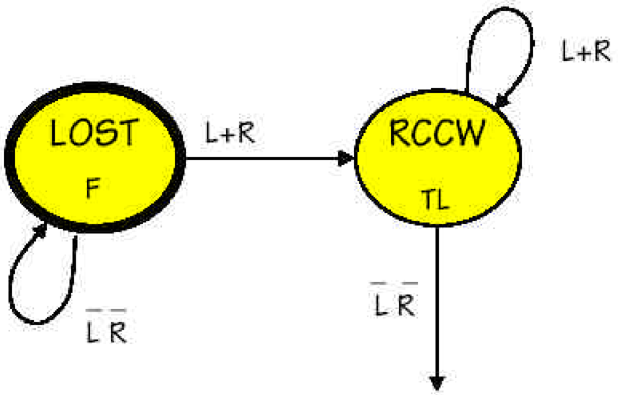
- ❑ Action: go forward until we hit something
  - – Initial state



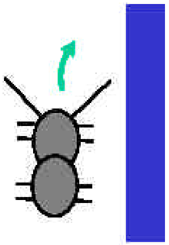
Bonk!!!



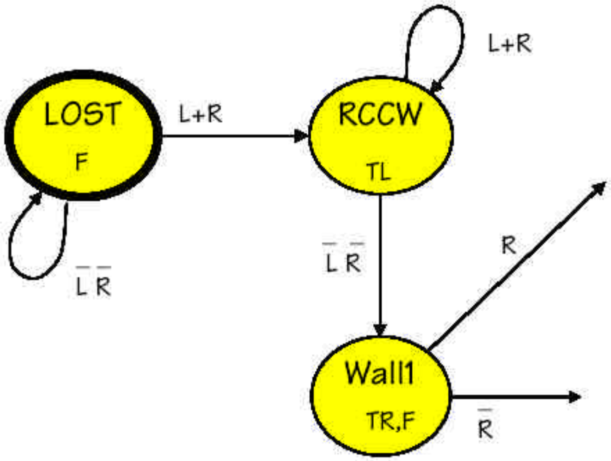
- □ Action: turn left (rotate counterclockwise)
  - – Until we don't touch anymore



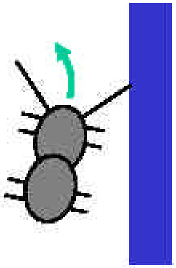
A little to the right



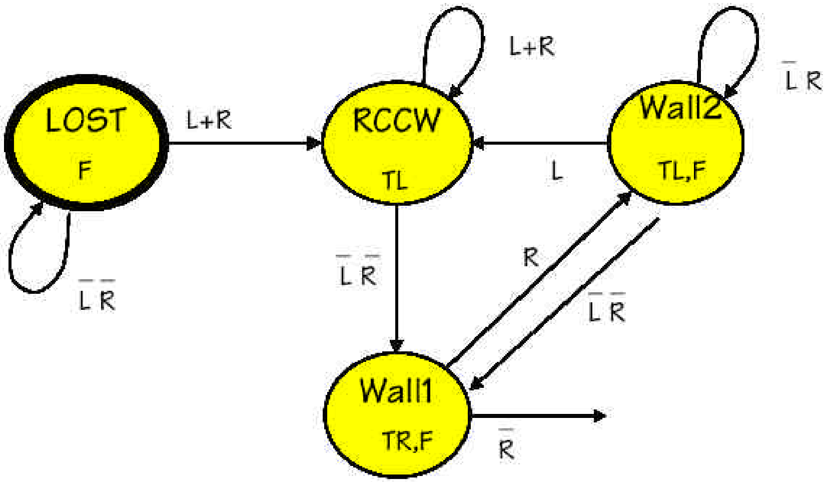
- Action: step forward and turn right a little
  - Looking for wall



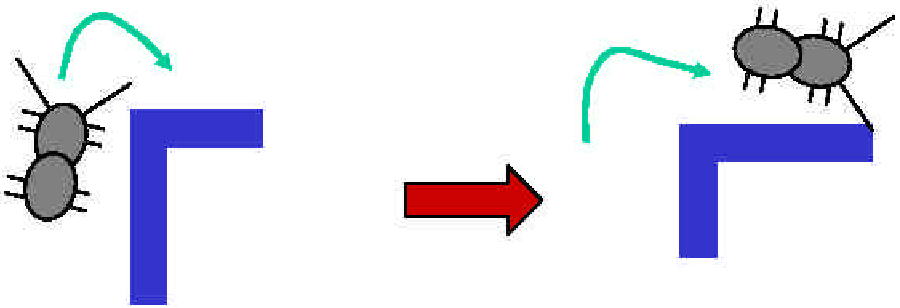
Then a little to the right



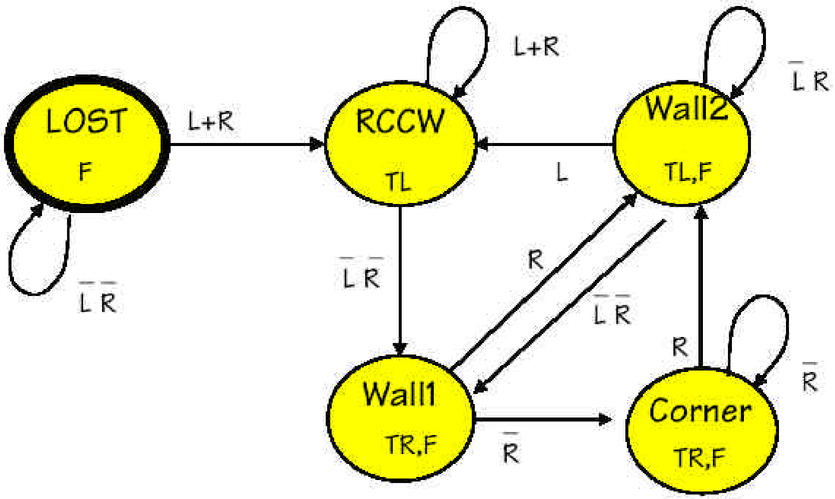
- ❑ Action: step and turn left a little, until not touching


LOST
F
L+R
RCCW
TL
L+R
Wall2
TL,F
$\overline{L}R$
L
$\overline{L}\,\overline{R}$
$\overline{L}\,\overline{R}$
R
$\overline{L}\,\overline{R}$
Wall1
TR,F
$\overline{R}$

Whoops – a corner!

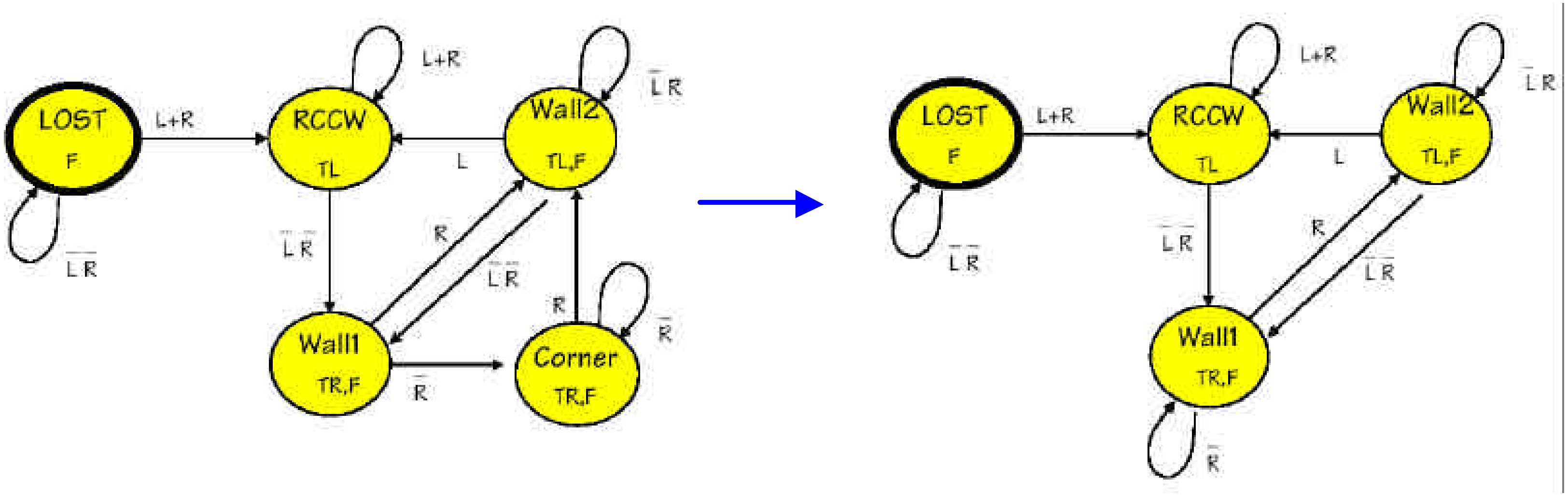


- Action: step and turn right until hitting next wall



Simplification

- ❑ Merge equivalent states where possible

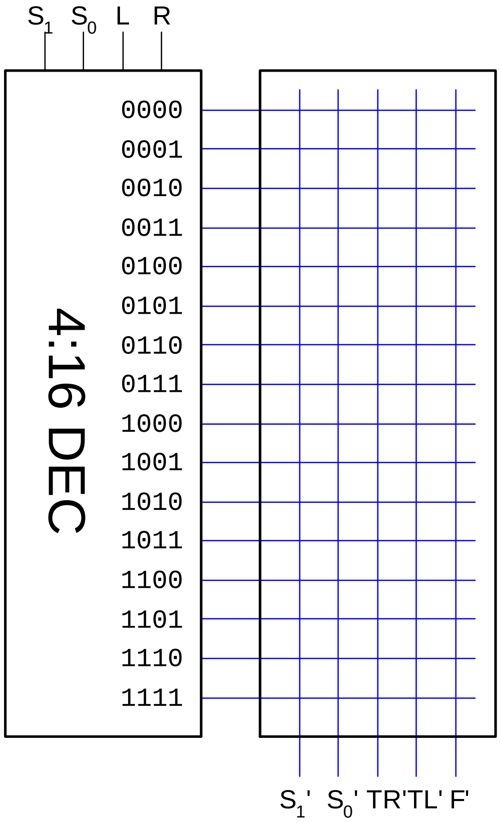

LOST
F
L+R
$\overline{L}\,\overline{R}$
RCCW
TL
L+R
Wall2
TL,F
$\overline{L}R$
L
$\overline{L}\,\overline{R}$
R
$\overline{L}\,\overline{R}$
R
Wall1
TR,F
$\overline{R}$
Corner
TR,F
$\overline{R}$
LOST
F
L+R
$\overline{L}\,\overline{R}$
RCCW
TL
L+R
Wall2
TL,F
$\overline{L}R$
L
$\overline{L}\,\overline{R}$
R
$\overline{L}\,\overline{R}$
Wall1
TR,F
$\overline{R}$

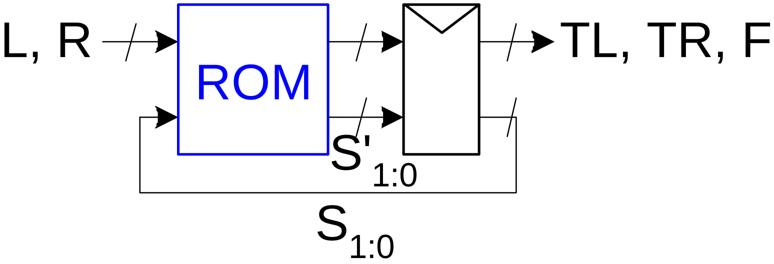
State Transition Table

	$S_{1:0}$	L	R	$S_{1:0}'$	TR	TL	F
Lost	00	0	0	00	0	0	1
	00	1	X	01	0	0	1
	00	0	1	01	0	0	1
RCCW	01	1	X	01	0	1	0
	01	0	1	01	0	1	0
	01	0	0	10	0	1	0
Wall1	10	X	0	10	1	0	1
	10	X	1	11	1	0	1
Wall2	11	1	X	01	0	1	1
	11	0	0	10	0	1	1
	11	0	1	11	0	1	1

ROM Implementation

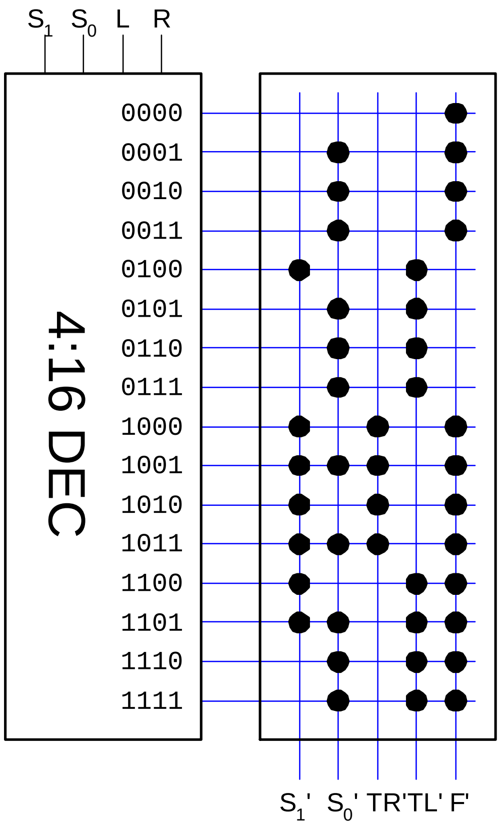
- ❑ 16-word x 5 bit ROM



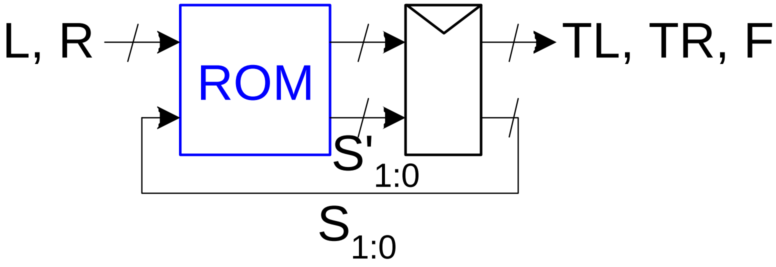


ROM Implementation

- 16-word x 5 bit ROM



	S_1'	S_0'	TR'	TL'	F'
0000					●
0001		●			●
0010		●			●
0011		●			●
0100	●			●	
0101		●		●	
0110		●		●	
0111		●		●	
1000	●		●		●
1001	●	●	●		●
1010	●		●		●
1011	●	●	●		●
1100	●			●	●
1101	●	●		●	●
1110		●		●	●
1111		●		●	●



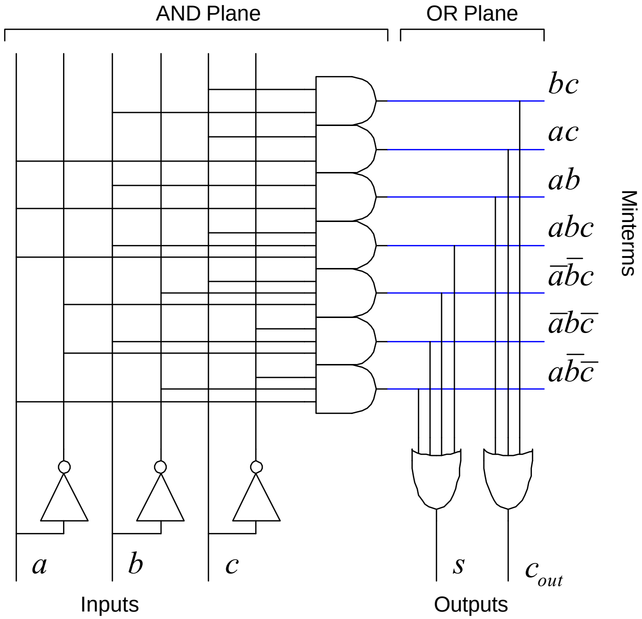
PLAs

- A *Programmable Logic Array* performs any function in sum-of-products form.
- *Literals*: inputs & complements
- *Products / Minterms*: AND of literals
- *Outputs*: OR of Minterms

- Example: Full Adder

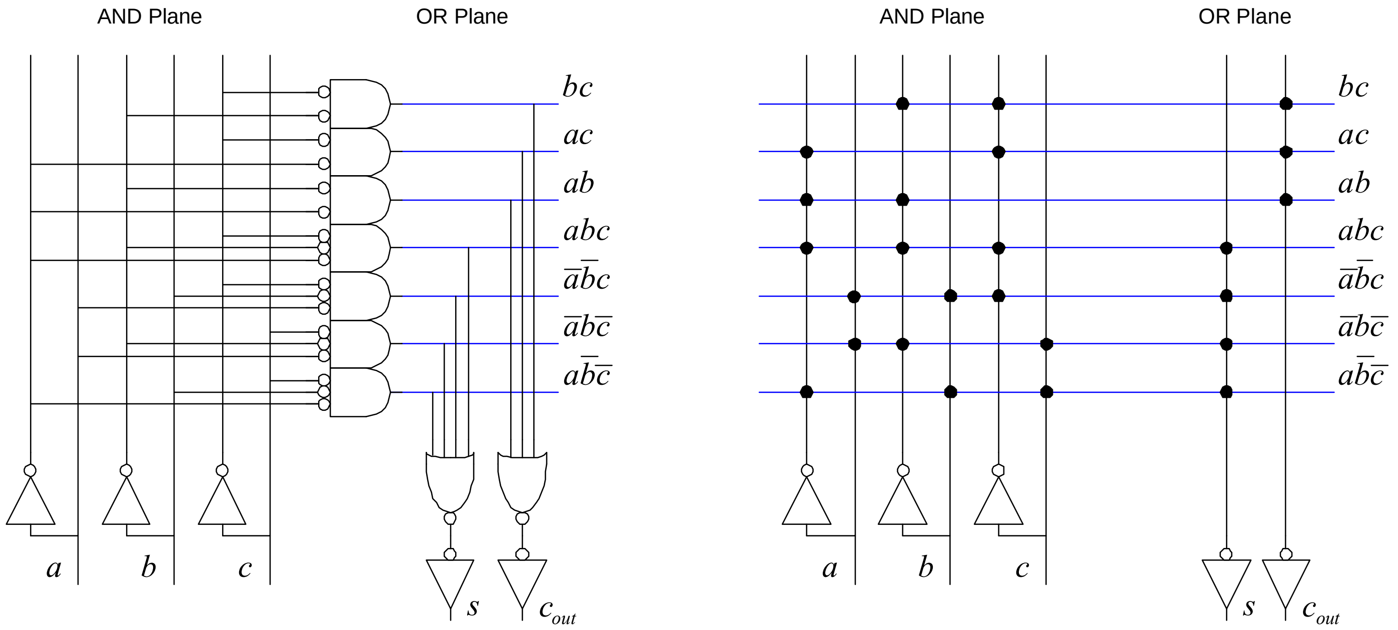
$$s = a\overline{b}\overline{c} + \overline{a}b\overline{c} + \overline{a}\overline{b}c + abc$$

$$c_{\text{out}} = ab + bc + ac$$

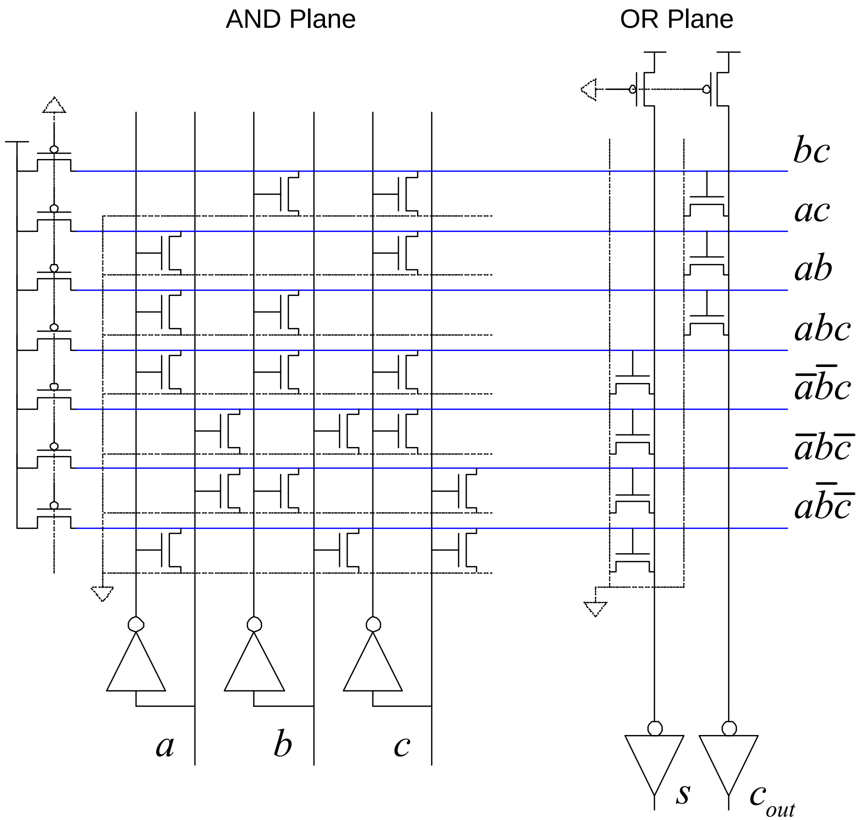

AND Plane
OR Plane
bc
ac
ab
abc
$\overline{a}\overline{b}c$
$\overline{a}b\overline{c}$
$a\overline{b}\overline{c}$
Minterms
a
b
c
Inputs
s
c_{out}
Outputs

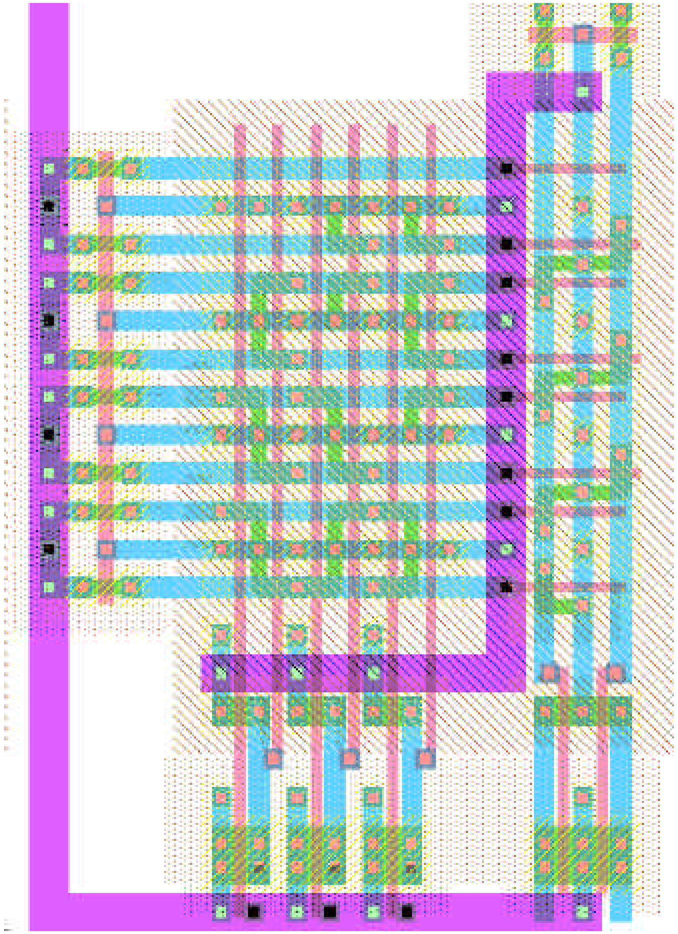
NOR-NOR PLAs

- ANDs and ORs are not very efficient in CMOS
- Dynamic or Pseudo-nMOS NORs are very efficient
- Use DeMorgan’s Law to convert to all NORs



PLA Schematic & Layout


AND Plane
OR Plane
bc
ac
ab
abc
$\overline{a}\overline{b}c$
$\overline{a}b\overline{c}$
$a\overline{b}\overline{c}$
a
b
c
s
c_{out}



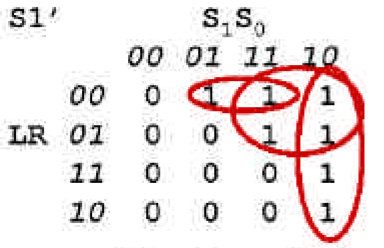
PLAs vs. ROMs

- ❑ The OR plane of the PLA is like the ROM array
- ❑ The AND plane of the PLA is like the ROM decoder
- ❑ PLAs are more flexible than ROMs
  - – No need to have 2^n rows for n inputs
  - – Only generate the minterms that are needed
  - – Take advantage of logic simplification

Example: RoboAnt PLA

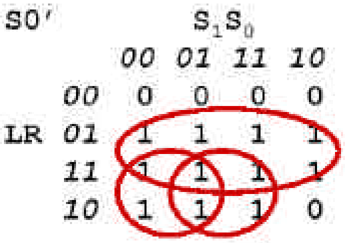
- ❑ Convert state transition table to logic equations

$S_{1:0}$	L	R	$S_{1:0}'$	TR	TL	F
00	0	0	00	0	0	1
00	1	X	01	0	0	1
00	0	1	01	0	0	1
01	1	X	01	0	1	0
01	0	1	01	0	1	0
01	0	0	10	0	1	0
10	X	0	10	1	0	1
10	X	1	11	1	0	1
11	1	X	01	0	1	1
11	0	0	10	0	1	1
11	0	1	11	0	1	1



S1′	LR \ S_1S_0	00	01	11	10
	00	0	1	1	1
LR	01	0	0	1	1
	11	0	0	0	1
	10	0	0	0	1

$$S_1' = S_1\overline{S_0} + \overline{L}S_1 + \overline{L}\,\overline{R}S_0$$



S0′	LR \ S_1S_0	00	01	11	10
	00	0	0	0	0
LR	01	1	1	1	1
	11	1	1	1	1
	10	1	1	1	0

$$S_0' = R + L\overline{S_1} + LS_0$$

$$TR = S_1\overline{S_0}$$

$$TL = S_0$$

$$F = S_1 + \overline{S_0}$$

RoboAnt Dot Diagram

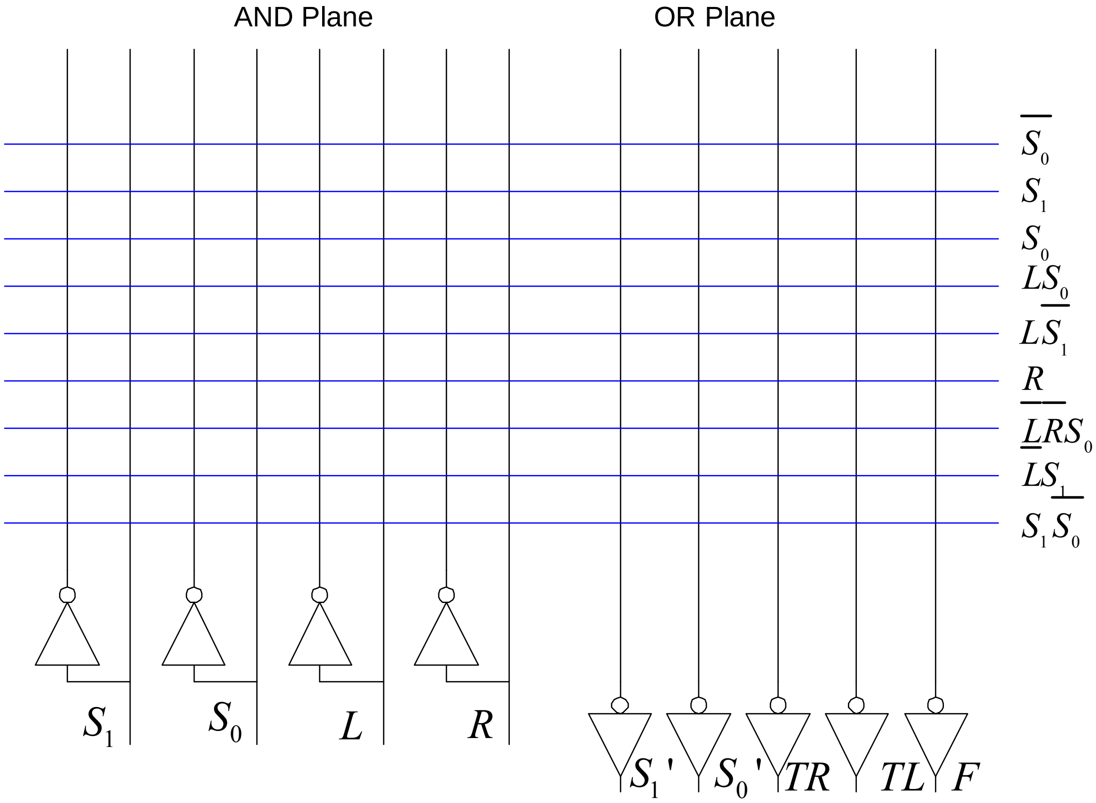
$$S1' = S_1\overline{S_0} + \overline{L}S_1 + \overline{L}\,\overline{R}S_0$$

$$S0' = R + L\overline{S_1} + LS_0$$

$$TR = S_1\overline{S_0}$$

$$TL = S_0$$

$$F = S_1 + \overline{S_0}$$



RoboAnt Dot Diagram

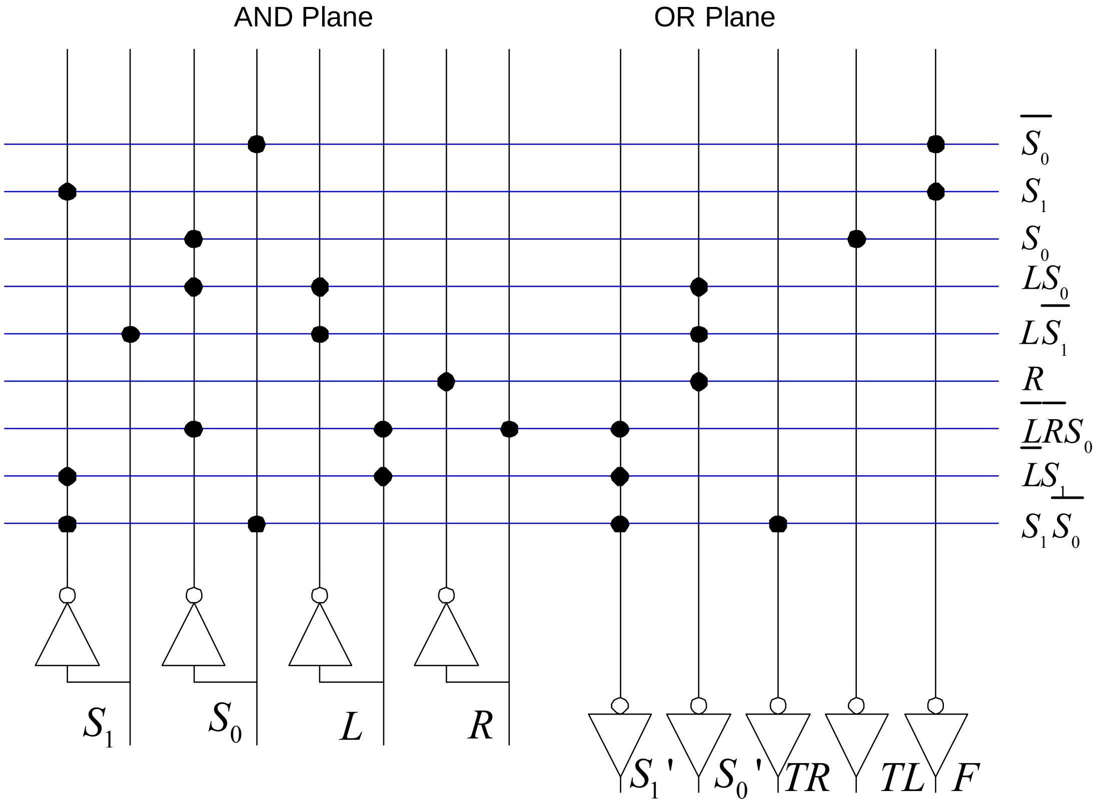
$$S1' = S_1\overline{S_0} + \overline{L}S_1 + \overline{L}\,\overline{R}S_0$$

$$S0' = R + L\overline{S_1} + LS_0$$

$$TR = S_1\overline{S_0}$$

$$TL = S_0$$

$$F = S_1 + \overline{S_0}$$


AND Plane
OR Plane
$\overline{S_0}$
S_1
S_0
LS_0
$L\overline{S_1}$
R
$\overline{L}\,\overline{R}S_0$
$\overline{L}S_1$
$S_1\overline{S_0}$
S_1
S_0
L
R
S_1'
S_0'
TR
TL
F