EECS 151/251A Spring 2023 Digital Design and Integrated Circuits

Instructor:
John Wawrzynek

Lecture 7: Combinational Logic part 2, FSMs part 1

Announcements

- HW2 being graded.
- HW 3 posted.
- *Wawrzynek office hours moving to Thursday 12:30 starting this week.*

Outline

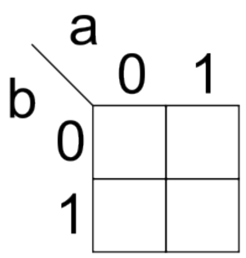
Combinational Logic:

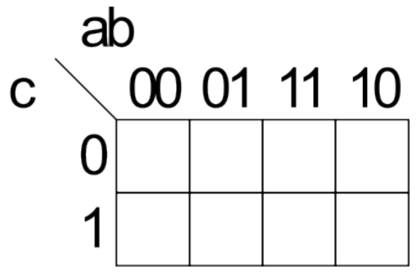
- *Boolean Simplification*
- *Multi-level Logic,*
- *NAND/NOR*
- *XOR*

Finite State Machines:

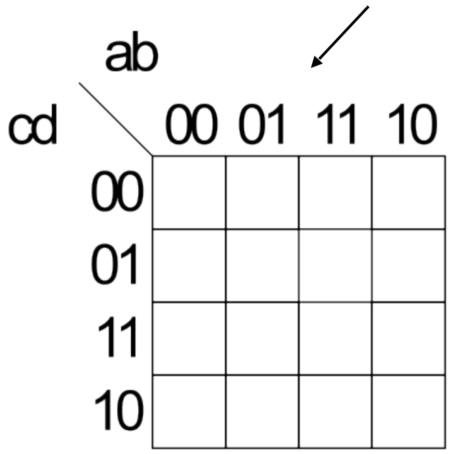
Karnaugh Map Method

- K-map is a method of representing the TT leading to simplification.





Note: "gray code" labeling.



5 & 6 variable k-maps possible

K-map Simplification

1. Draw K-map of the appropriate number of variables (between 2 and 6)
2. Fill in map with function values from truth table.
3. Form groups of 1's.
   - ✓ Dimensions of groups must be even powers of two (1x1, 1x2, 1x4, …, 2x2, 2x4, …)
   - ✓ <u>Form as large as possible groups and as few groups as possible</u>.
   - ✓ Groups can overlap (this helps make larger groups)
   - ✓ Remember K-map is periodical in all dimensions (groups can cross over edges of map and continue on other side)
4. For each group write a product term.
   - the term includes the "constant" variables (use the uncomplemented variable for a constant 1 and complemented variable for constant 0)
5. Form Boolean expression as sum-of-products.

Product-of-Sums K-map

1. Form groups of 0’s instead of 1’s.
2. For each group write a sum term.
   - the term includes the “constant” variables (use the uncomplemented variable for a constant 0 and complemented variable for constant 1)
3. Form Boolean expression as product-of-sums.

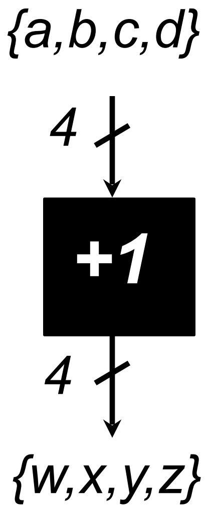
cd \ ab	00	01	11	10
00	1	0	0	1
01	0	1	0	0
11	1	1	1	1
10	1	1	1	1

$$f = (b' + c + d)(a' + c + d')(b + c + d')$$

BCD incrementer example

Binary Coded Decimal

	a	b	c	d	w	x	y	z
0	0	0	0	0	0	0	0	1
1	0	0	0	1	0	0	1	0
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	1	0	0
4	0	1	0	0	0	1	0	1
5	0	1	0	1	0	1	1	0
6	0	1	1	0	0	1	1	1
7	0	1	1	1	1	0	0	0
8	1	0	0	0	1	0	0	1
9	1	0	0	1	0	0	0	0
	1	0	1	0	-	-	-	-
	1	0	1	1	-	-	-	-
	1	1	0	0	-	-	-	-
	1	1	0	1	-	-	-	-
	1	1	1	0	-	-	-	-
	1	1	1	1	-	-	-	-



BCD Incrementer Example

- Note one map for each output variable.
- Function includes “don’t cares” (shown as “-” in the table).
  - These correspond to places in the function where we don’t care about its value, because we don’t expect some particular input patterns.
  - We are free to assign either 0 or 1 to each don’t care in the function, as a means to increase group sizes.
- In general, you might choose to write product-of-sums or sum-of-products according to which one leads to a simpler expression.

BCD incrementer example

w

cd \ ab	00	01	11	10
00	0	0	-	1
01	0	0	-	0
11	0	1	-	-
10	0	0	-	-

x

cd \ ab	00	01	11	10
00	0	1	-	0
01	0	1	-	0
11	1	0	-	-
10	0	1	-	-

y

cd \ ab	00	01	11	10
00	0	0	-	0
01	1	1	-	0
11	0	0	-	-
10	1	1	-	-

z

cd \ ab	00	01	11	10
00	1	1	-	1
01	0	0	-	0
11	0	0	-	-
10	1	1	-	-

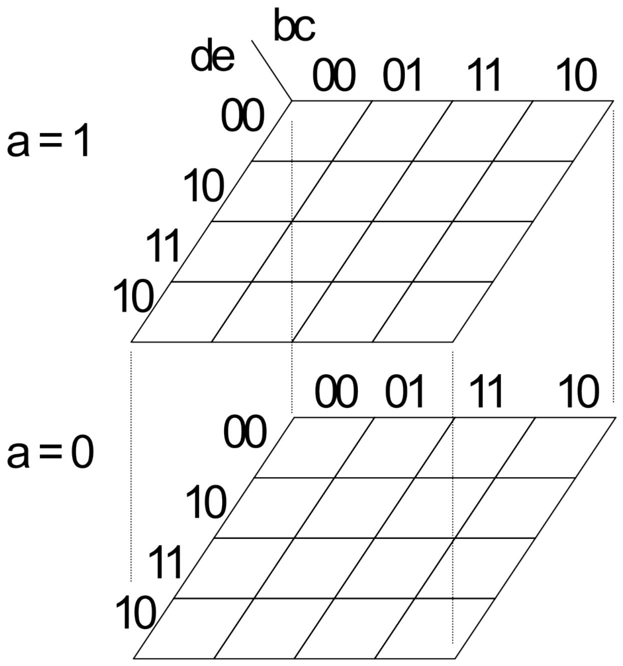
$w =$

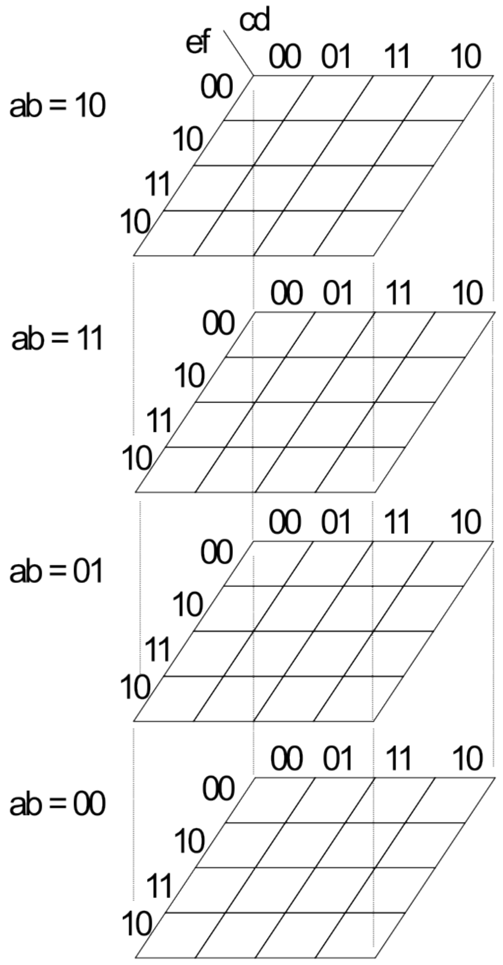
$x =$

$y =$

$z =$

Higher Dimensional K-maps





Boolean Simplification – Multi-level Logic

Multi-level Combinational Logic

- Example: reduced sum-of-products form

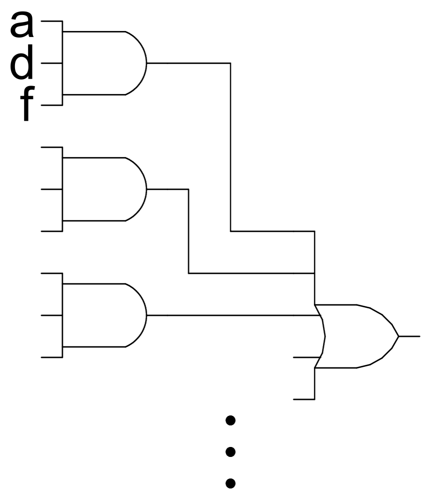
  `x = adf + aef + bdf + bef + cdf + cef + g`

- Implementation in 2-levels with gates:

  cost: 1 7-input OR, 6 3-input AND

  => ~50 transistors

  delay: 3-input AND gate delay + 7-input OR gate delay



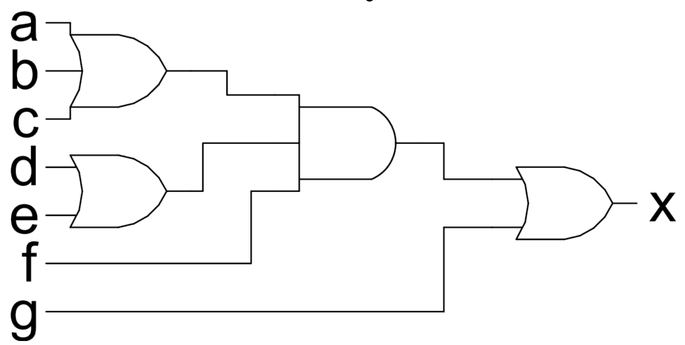
- Factored form:

  `x = (a + b +c)(d + e)f + g`

  cost: 1 3-input OR, 2 2-input OR, 1 3-input AND

  => ~20 transistors

  delay: 3-input OR + 3-input AND + 2-input OR



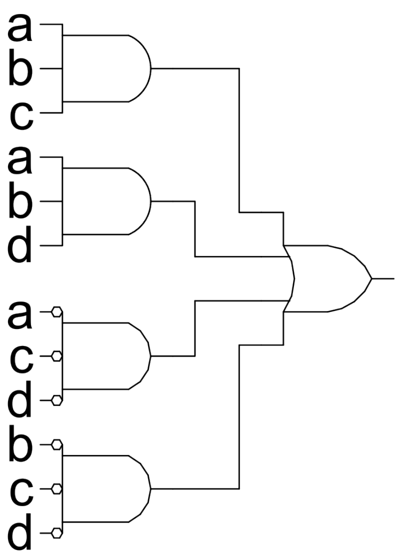
Footnote: NAND would be used in place of all ANDs and ORs.

Which is faster?

In general: Using multiple levels (more than 2) will reduce the cost. Sometimes also delay.
Sometimes a tradeoff between cost and delay.

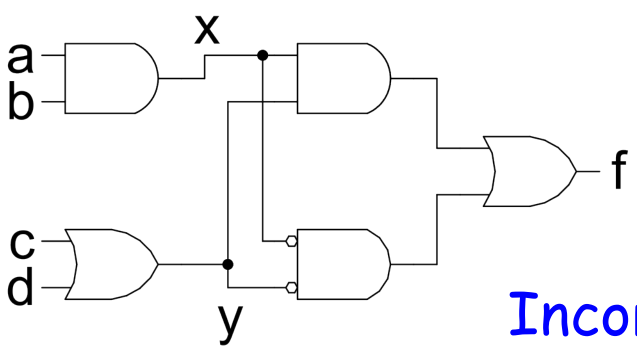
Multi-level Combinational Logic

Another Example: `F = abc + abd +a'c'd' + b'c'd'`



let `x = ab   y = c+d`

`f = xy + x'y'`



Incorporates fanout.

No convenient hand methods exist for multi-level logic simplification:

a) CAD Tools use sophisticated algorithms and heuristics

Guess what? These problems tend to be NP-complete

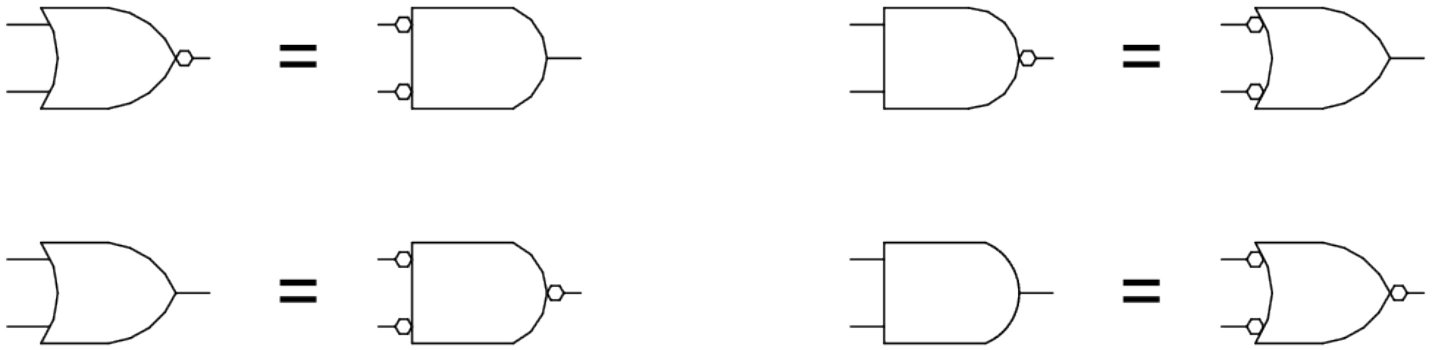
b) Humans and tools often exploit some special structure (example adder)

NAND-NAND & NOR-NOR Networks

DeMorgan's Law Review:

$(a + b)' = a'\,b'$ $\qquad$ $(a\,b)' = a' + b'$

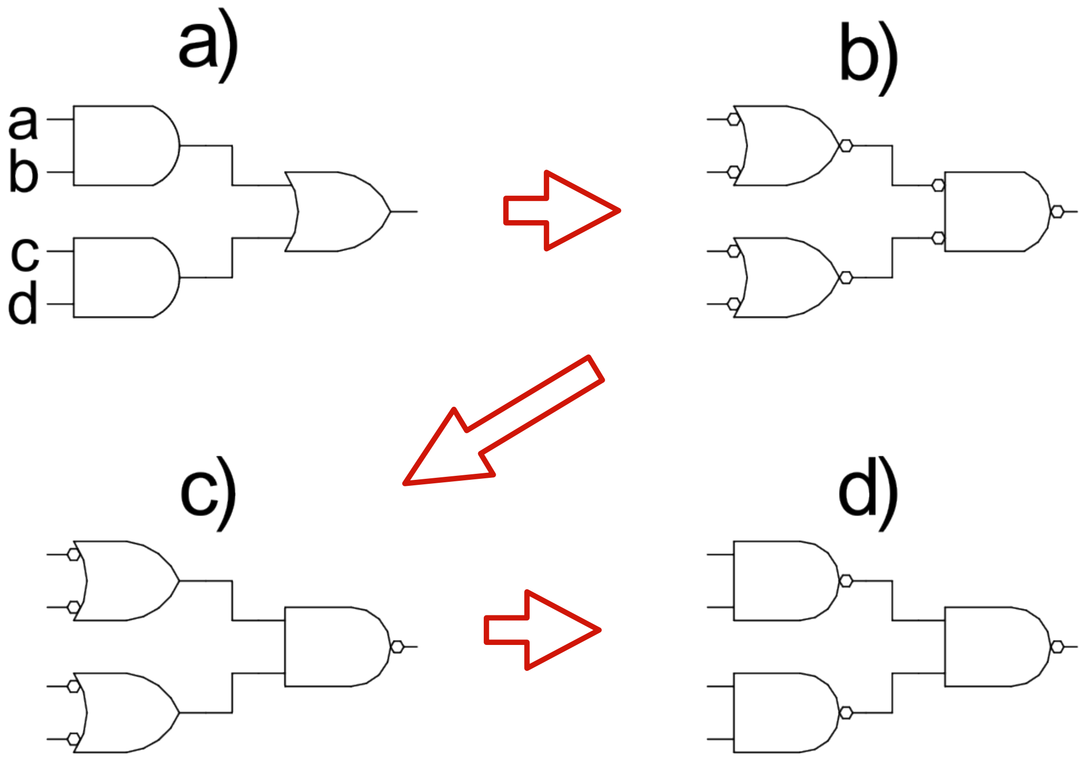
$a + b = (a'\,b')'$ $\qquad$ $(a\,b) = (a' + b')'$



push bubbles or *introduce in pairs* or *remove pairs:* *(x')' = x.*

NAND-NAND & NOR-NOR Networks

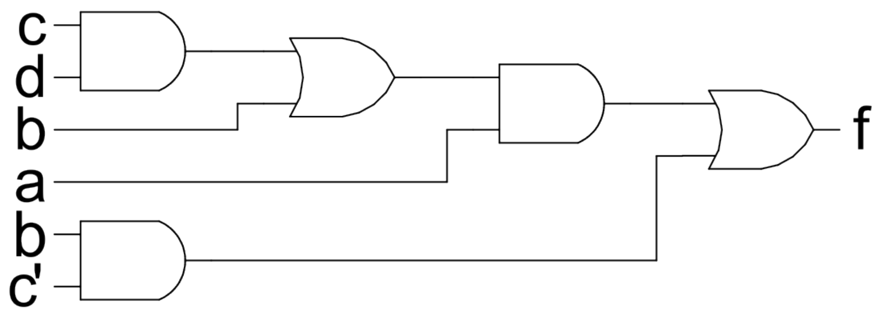
- Mapping from AND/OR to NAND/NAND



Multi-level Networks

Convert to NANDs:

F = a(b + cd) + bc'

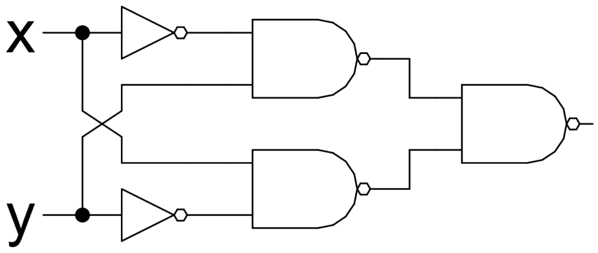


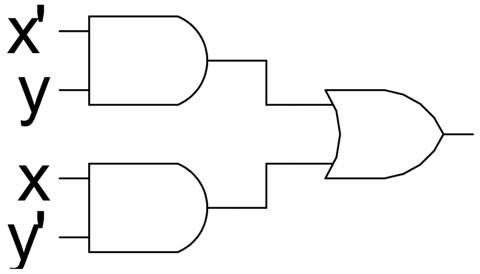
EXOR Function Implementations

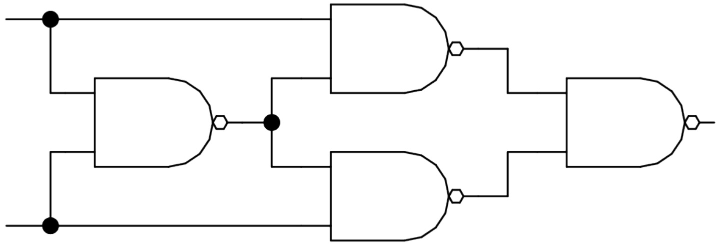
Parity, addition mod 2

$x \oplus y = x'y + xy'$

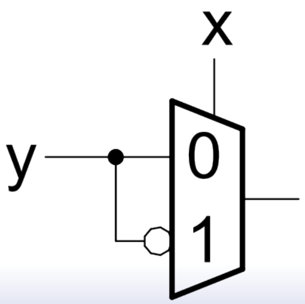
x y	xor	xnor
0 0	0	1
0 1	1	0
1 0	1	0
1 1	0	1







Another approach:

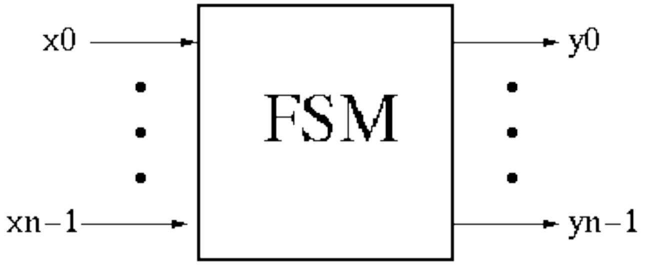


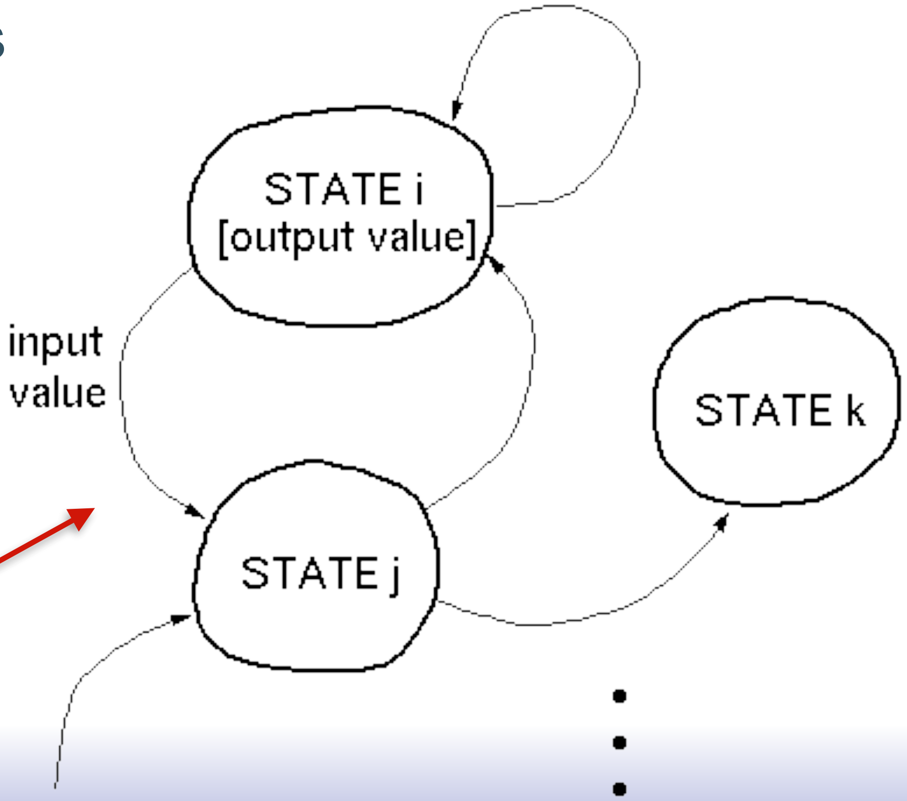
if x=0 then y else y'

Finite State Machines

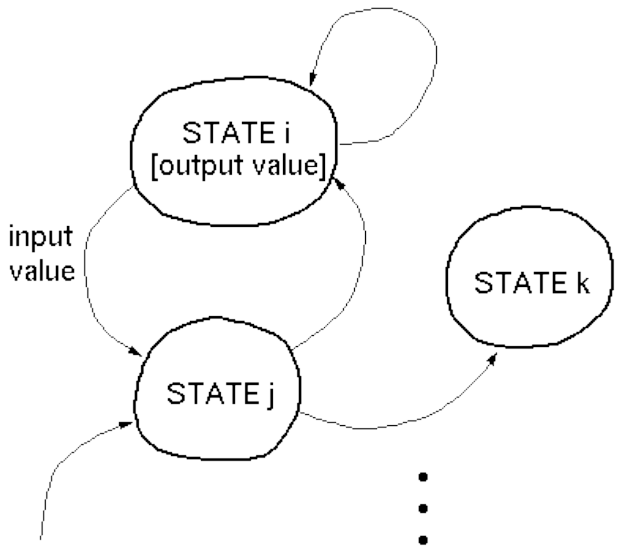
Finite State Machines (FSMs)

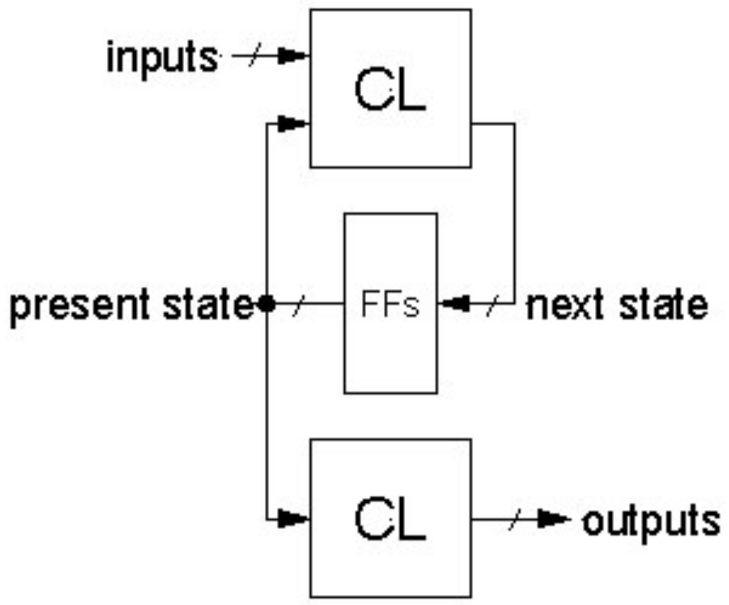
- **FSMs:**
  - Can model behavior of *any sequential circuit*
  - Useful representation for designing sequential circuits
  - As with all sequential circuits: output depends on present *and* past inputs
    - effect of past inputs represented by the current *state*
- Behavior is represented by ***State Transition Diagram***:
  - traverse one edge per clock cycle.





FSM Implementation





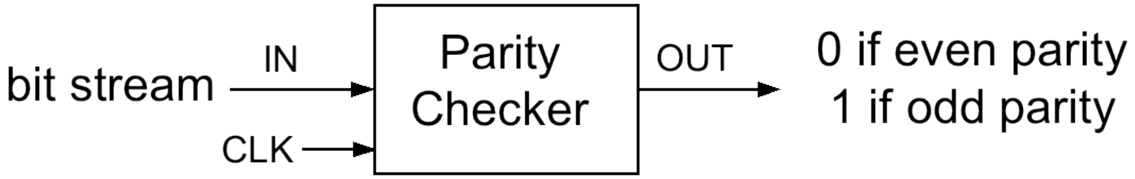
- Flip-flops form *state register*
- number of states ≤ $2^{\text{number of flip-flops}}$
- CL (combinational logic) calculates next state and output
- Remember: The FSM follows exactly one edge per cycle.

Later we will learn how to implement in Verilog. Now we learn how to design "by hand" to the gate level.

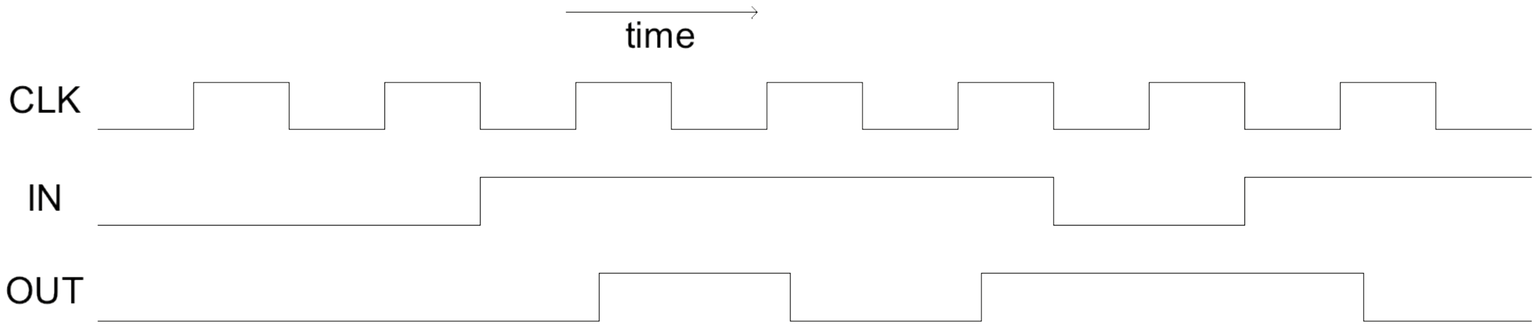
FSM Example: Parity Checker

A string of bits has "even parity" if the number of 1's in the string is even.

- Design a circuit that accepts a infinite serial stream of bits, and outputs a 0 if the parity thus far is even and outputs a 1 if odd:

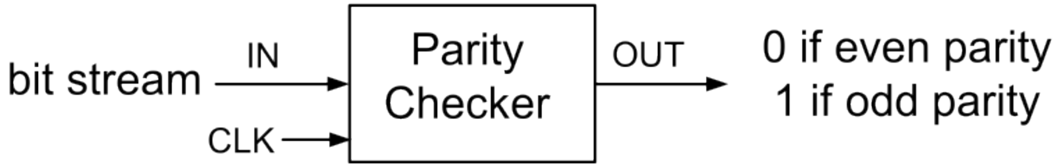


example:	0	0	1	1	1	0	1
	even	even	odd	even	odd	odd	even



Next we take this example through the "formal design process". But first, can you guess a circuit that performs this function?

By-hand Design Process (a)

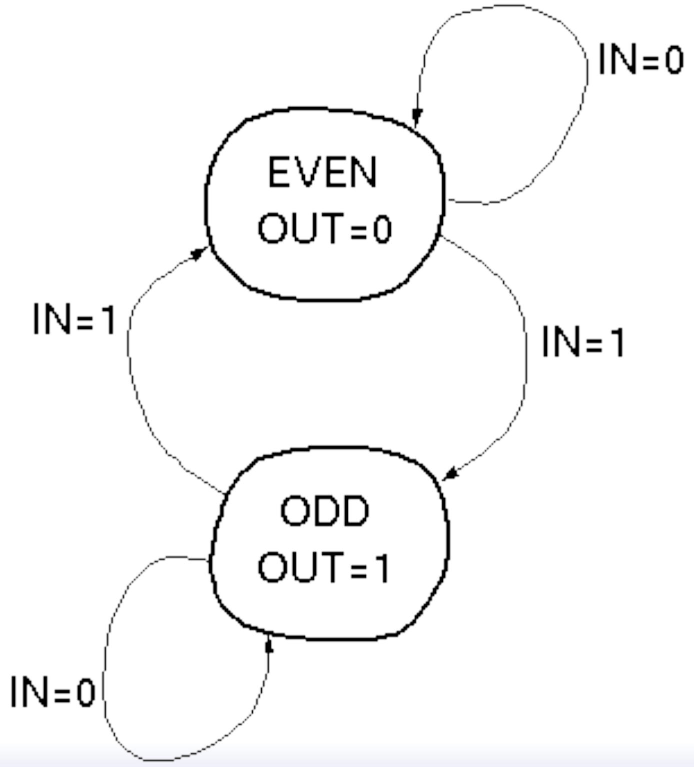


example:	0	0	1	1	1	0	1
	even	even	odd	even	odd	odd	even

$\xrightarrow{\text{time}}$

"State Transition Diagram"

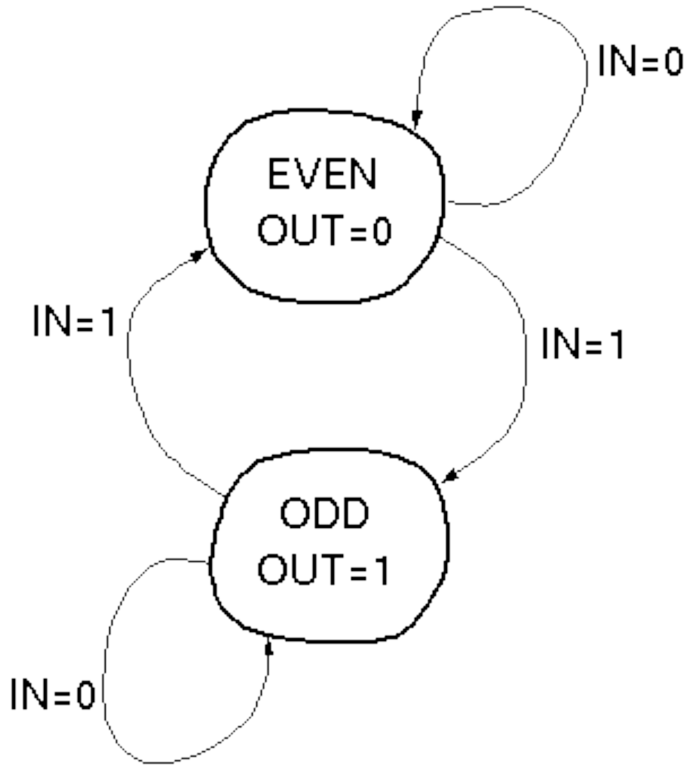
- circuit is in one of two "states".
- transition on each cycle with each new input, over exactly one arc (edge).
- Output depends on which state the circuit is in.



By-hand Design Process (b)

State Transition Table:

present state	*OUT*	*IN*	*next state*
EVEN	*0*	*0*	*EVEN*
EVEN	*0*	*1*	*ODD*
ODD	*1*	*0*	*ODD*
ODD	*1*	*1*	*EVEN*



Invent a code to represent states:

Let 0 = EVEN state, 1 = ODD state

present state (ps)	*OUT*	*IN*	*next state (ns)*
0	*0*	*0*	*0*
0	*0*	*1*	*1*
1	*1*	*0*	*1*
1	*1*	*1*	*0*

Derive logic equations from table (how?):

OUT = PS

NS = PS xor IN

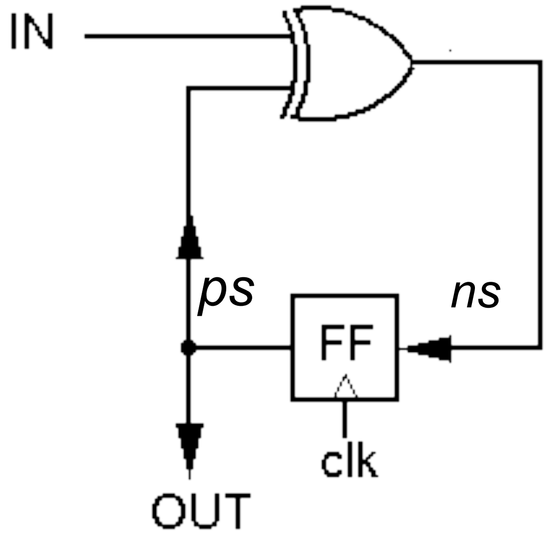
By-hand Design Process (c)

Logic equations from table:

OUT = PS

NS = PS xor IN

- Circuit Diagram:
  - XOR gate for NS calculation
  - Flip-Flop to hold present state
  - no logic needed for output in this example.



“Formal” By-hand Design Process

Review of Design Steps:

*1. Specify **circuit function** (English)*

*2. Draw **state transition diagram***

*3. Write down **symbolic state transition table***

*4. Write down **encoded state transition table***

*5. Derive **logic equations***

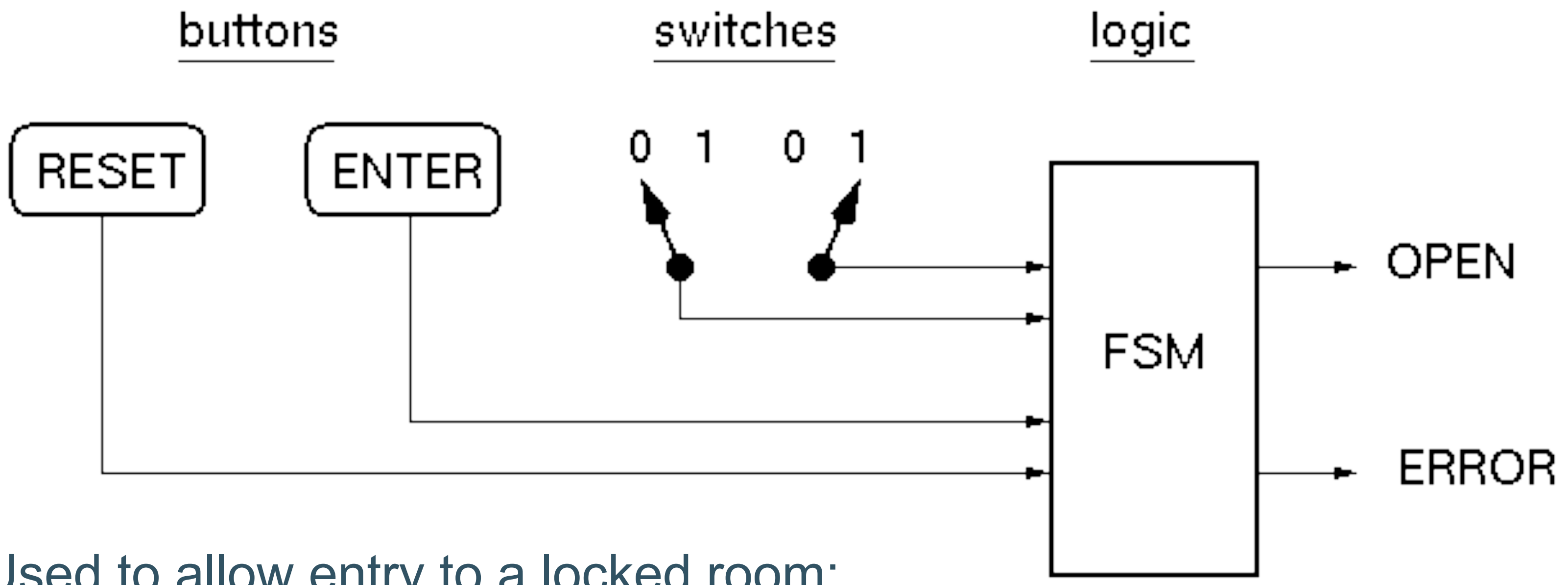
*6. Derive **circuit diagram***

Register to hold state

Combinational Logic for Next State and Outputs

Another FSM Design Example

Combination Lock Example



- Used to allow entry to a locked room:

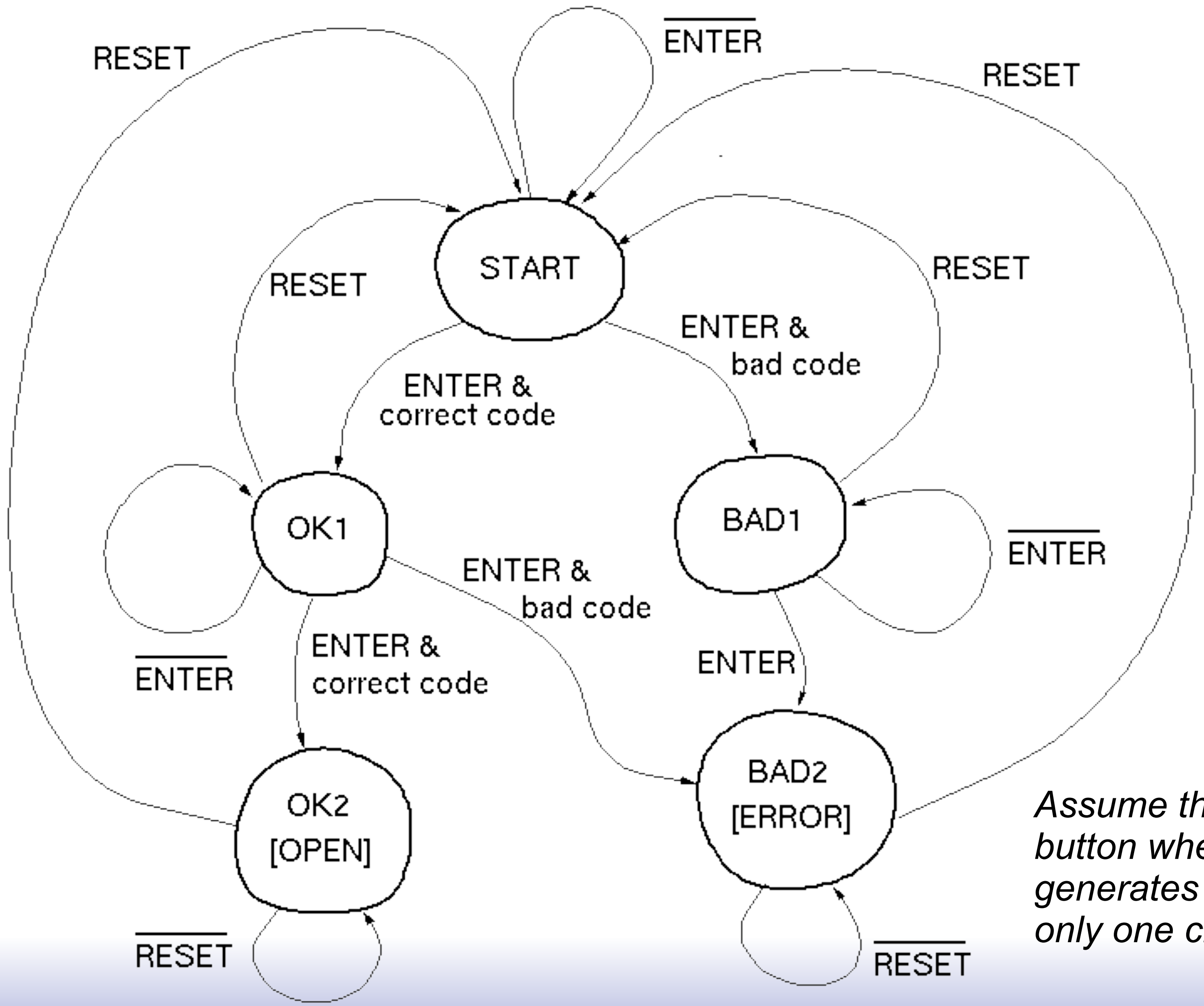
  2-bit serial combination. Example 01,11:

  1. Set switches to 01, press ENTER
  2. Set switches to 11, press ENTER
  3. OPEN is asserted (OPEN=1).

     If wrong code, ERROR is asserted (after second combo word entry).

     Press Reset at anytime to try again.

Combinational Lock STD

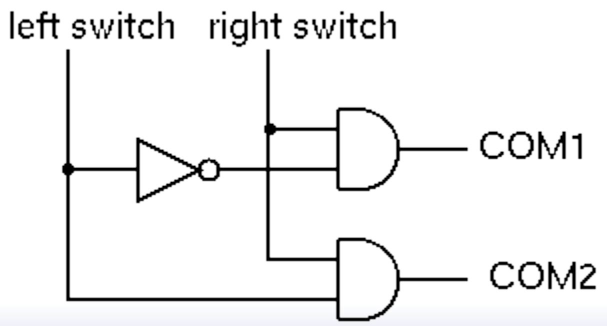


Assume the ENTER button when pressed generates a pulse for only one clock cycle.

Symbolic State Transition Table

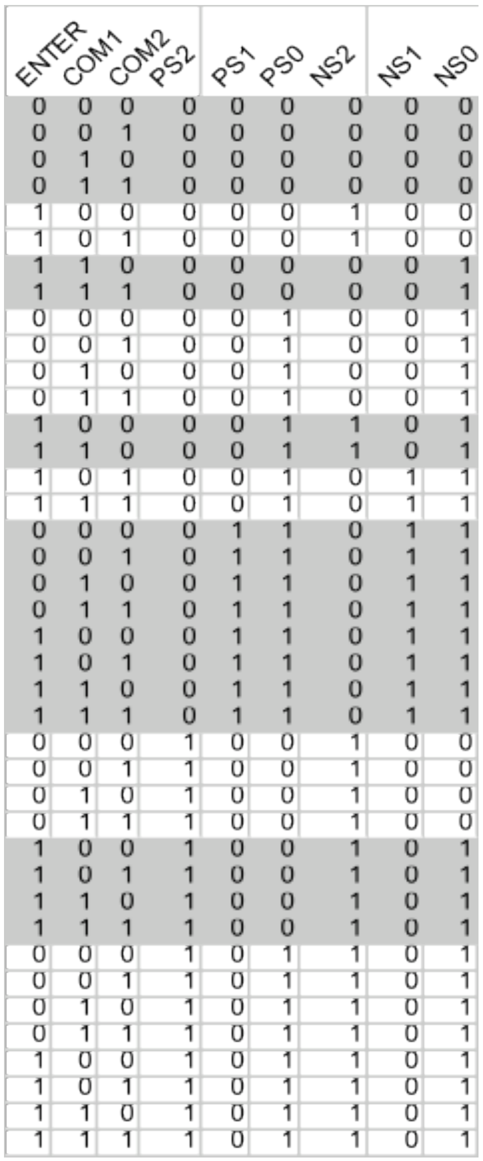
RESET	ENTER	COM1	COM2	Preset State	Next State	OPEN	ERROR
0	0	*	*	START	START	0	0
0	1	0	*	START	BAD1	0	0
0	1	1	*	START	OK1	0	0
0	0	*	*	OK1	OK1	0	0
0	1	*	0	OK1	BAD2	0	0
0	1	*	1	OK1	OK2	0	0
0	*	*	*	OK2	OK2	1	0
0	0	*	*	BAD1	BAD1	0	0
0	1	*	*	BAD1	BAD2	0	0
0	*	*	*	BAD2	BAD2	0	1
1	*	*	*	*	START	0	0

Decoder logic for checking combination (01,11):



** represents "wild card" - expands to all combinations*

Encoded ST Table



ENTER	COM1	COM2	PS2	PS1	PS0	NS2	NS1	NS0
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0
1	0	0	0	0	0	1	0	0
1	0	1	0	0	0	1	0	0
1	1	0	0	0	0	0	0	1
1	1	1	0	0	0	0	0	1
0	0	0	0	0	1	0	0	1
0	0	1	0	0	1	0	0	1
0	1	0	0	0	1	0	0	1
0	1	1	0	0	1	0	0	1
1	0	0	0	0	1	1	0	1
1	1	0	0	0	1	1	0	1
1	0	1	0	0	1	0	1	1
1	1	1	0	0	1	0	1	1
0	0	0	0	1	1	0	1	1
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	0	1	1	0	1	1
1	0	0	0	1	1	0	1	1
1	0	1	0	1	1	0	1	1
1	1	0	0	1	1	0	1	1
1	1	1	0	1	1	0	1	1
0	0	0	1	0	0	1	0	0
0	0	1	1	0	0	1	0	0
0	1	0	1	0	0	1	0	0
0	1	1	1	0	0	1	0	0
1	0	0	1	0	0	1	0	1
1	0	1	1	0	0	1	0	1
1	1	0	1	0	0	1	0	1
1	1	1	1	0	0	1	0	1
0	0	0	1	0	1	1	0	1
0	0	1	1	0	1	1	0	1
0	1	0	1	0	1	1	0	1
0	1	1	1	0	1	1	0	1
1	0	0	1	0	1	1	0	1
1	0	1	1	0	1	1	0	1
1	1	0	1	0	1	1	0	1
1	1	1	1	0	1	1	0	1

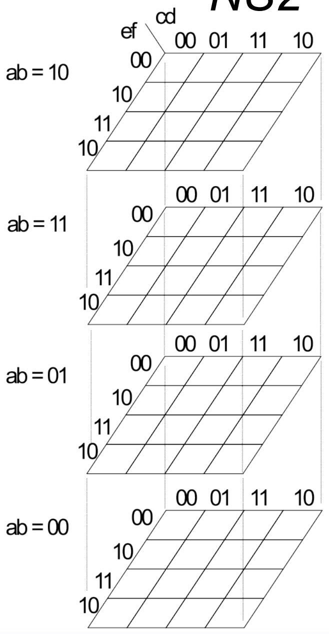
- *Assign states:*

START=000, OK1=001, OK2=011

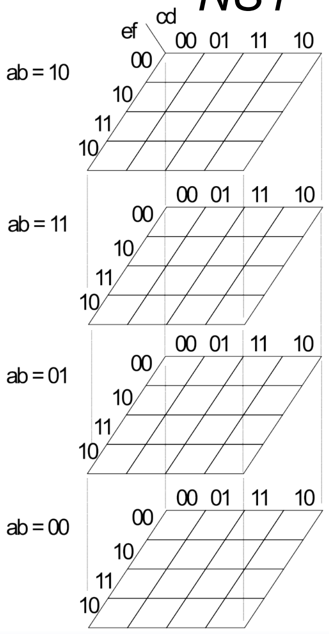
BAD1=100, BAD2=101

- *Omit reset. Assume that primitive flip-flops has reset input.*
- *Rows not shown have don't cares in output. Correspond to invalid PS values.*

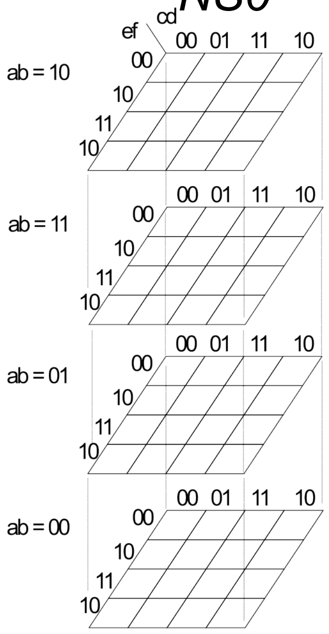
NS2



NS1



NS0



- *What are the output functions for OPEN and ERROR?*