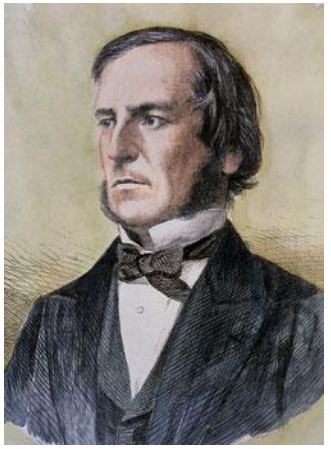
# FROM LAST TIME…



Computer Systems

- Why do Computer Systems matter?
- Boolean algebra
- Combinational logic

| *B1* | *B2* | *L* |
|---|---|---|
| 0 | 0 | **0** |
| 0 | 1 | **0** |
| 1 | 0 | **1** |
| 1 | 1 | **0** |

$$y = (B1 \cdot B2) + (B1 \cdot \overline{B2}) + (\overline{B1} \cdot B2) + (\overline{B1} \cdot \overline{B2})$$

$$y = (B1 + B2) \cdot (B1 + \overline{B2}) \cdot (\overline{B1} + B2) \cdot (\overline{B1} + \overline{B2})$$

# UNIT 3:
# COMBINATIONAL LOGIC

## PART B:
## BOOLEAN SIMPLIFICATION

# DEVICE OPERATION CAN OFTEN BE DEFINED IN A TRUTH TABLE

| Thermal Sensor | Front Sensor | Rear Sensor | Drive Motor | Drive Direction |
|---|---|---|---|---|
| OFF | OFF | OFF | ON | FORWARD |
| OFF | OFF | ON | ON | FORWARD |
| OFF | ON | OFF | ON | REVERSE |
| OFF | ON | ON | OFF | DON'T CARE |
| ON | OFF | OFF | OFF | DON'T CARE |
| ON | OFF | ON | OFF | DON'T CARE |
| ON | ON | OFF | OFF | DON'T CARE |
| ON | ON | ON | OFF | DON'T CARE |

Approaches:

1. Black Box (hand off problem to somebody else's software/compiler)
2. Brute Force (write software or build circuitry that checks all conditions)
3. Reduce Logic (write less complex software or circuitry)

# SUM-OF-PRODUCTS AND PRODUCT-OF-SUMS MAY NOT MINIMIZE COMPLEXITY

| *B1* | *B2* | *L* |
|---|---|---|
| 0 | 0 | **0** |
| 0 | 1 | **0** |
| 1 | 0 | **1** |
| 1 | 1 | **1** |

$$y = (B1 \cdot \overline{B2}) + (\mathrm{B1} \cdot B2)$$

$$y = (B1 + B2) \cdot (B1 + \overline{B2})$$

Might there be a way to reduce the complexity in a systematic fashion?

# BOOLEAN LOGIC CAN BE SIMPLIFIED IN SEVERAL WAYS

**Why?**

- Reduce number of gates and connections
- Limit propagation delays
- Make function logic more evident

**How?**

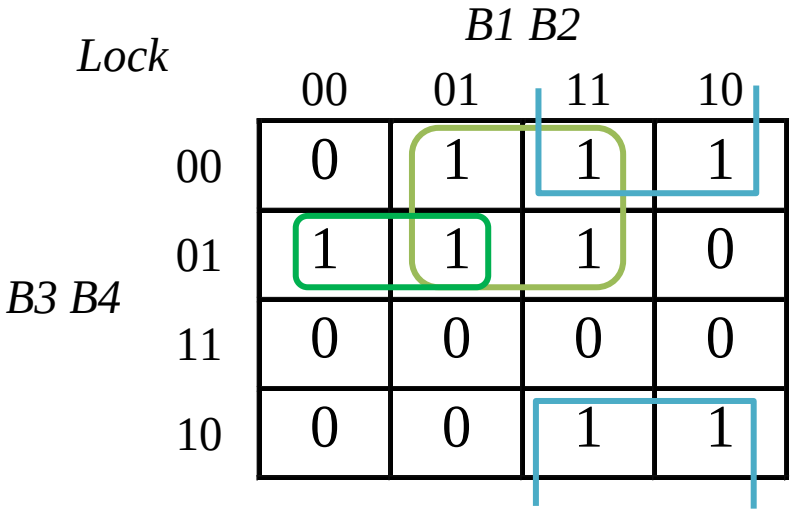
- Boolean algebra (apply axioms and theorems)
- Karnaugh maps (graphical method)
- Quine-McCluskey method (deterministic algorithm)

# SIMPLIFICATION MAY BE INCLUDED IN ADVANCED DIGITAL SYSTEMS

Modern synthesis code handles logic simplification for programmable devices. However, K-maps remain useful for minimizing circuitry when designing and building simple circuits.

# KARNAUGH MAPS PROVIDE A SIMPLE MINIMIZATION TECHNIQUE

- Graphical minimization technique for three to six variables
- Construct a table with adjacent boxes using Gray code
- Adjacency is established along interior and exterior boundaries



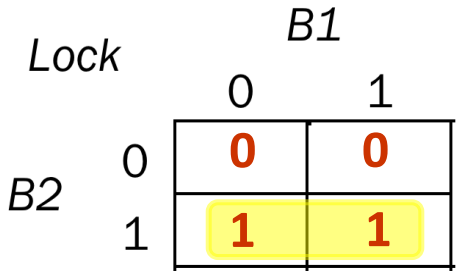
# EXAMPLE: GRAPHICAL MINIMIZATION

**Example: A two button door lock**

- A two button door lock with two possible unlock combinations:

| B1 | B2 | L |
|---|---|---|
| 0 | 0 | **0** |
| 0 | 1 | **1** |
| 1 | 0 | **0** |
| 1 | 1 | **1** |

- Adjacent 1's are collected
- Keep only terms that don't change for the final expression




$$y = (\overline{B1} \cdot B2) + (B1 \cdot B2)$$

$$y = (\overline{B1} + B1) \cdot B2$$

$$y = B2$$

# HOW TO EXTEND GRAPHICAL MINIMIZATION?

We want to add more variables…

- 3-variable map is three dimensional!
- Need to map 3-D maps to 2-D AND preserve adjacency
- Need a different way than the “natural” binary counting order

# GRAY CODE IS A DIFFERENT "COUNTING" SEQUENCE

| Decimal | Binary | Gray Code |
|---|---|---|
| 0 | 000 | 000 |
| 1 | 001 | 001 |
| 2 | 010 | 011 |
| 3 | 011 | 010 |
| 4 | 100 | 110 |
| 5 | 101 | 111 |
| 6 | 110 | 101 |
| 7 | 111 | 100 |

# MULTI-BIT CHANGES MAY CAUSE DATA PROBLEMS

- "Straight" binary counting – two transitions involve changing of both bits:

$$00 \rightarrow \underbrace{01 \rightarrow 10}_{\substack{Both\ bits \\ change \\ simultaneously}} \rightarrow 11$$

- Depends on which bit changes first, the intermediate values will come from the following sequence:

$$01 \rightarrow 00 \rightarrow 10 \quad \text{or} \quad 01 \rightarrow 11 \rightarrow 10$$

- Alternative counting order:

$$00 \rightarrow 01 \rightarrow 11 \rightarrow 10$$

# REFLECTED GRAY CODE IS EASILY CREATED

Reflect and prefix method

- Reflect down
- Prefix top half with 0
- Prefix bottom half with 1

$n = 1$

$$0$$
$$1$$

$n = 2$

$$0 \rightarrow 00$$
$$1 \rightarrow 01$$
$$1 \rightarrow 11$$
$$0 \rightarrow 10$$

$n = 3$

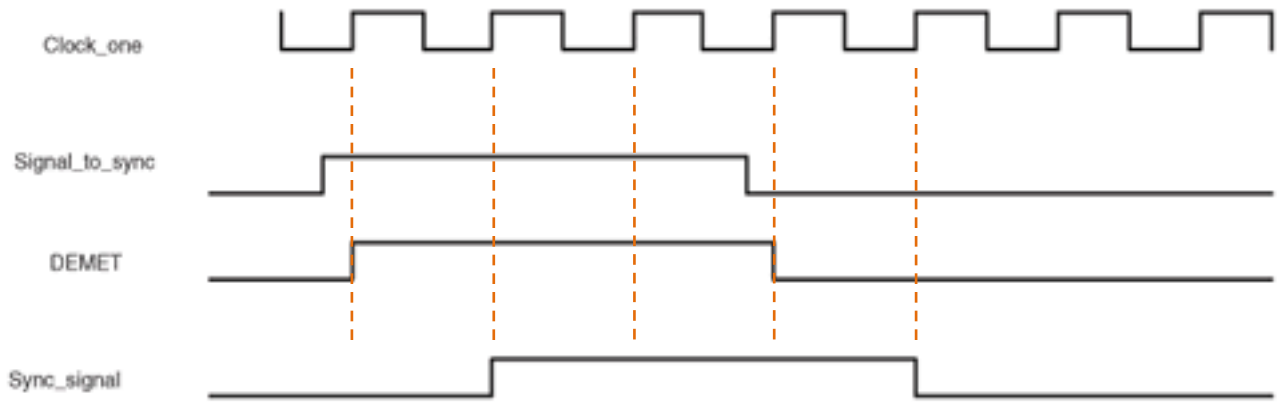
$$00 \rightarrow 000$$
$$01 \rightarrow 001$$
$$11 \rightarrow 011$$
$$10 \rightarrow 010$$
$$10 \rightarrow 110$$
$$11 \rightarrow 111$$
$$01 \rightarrow 101$$
$$00 \rightarrow 100$$

# REFLECTED GRAY CODE CAN BE CONVERTED BACK TO BINARY CODE

- The bits of an $n$-bit binary or Gray-code code word are numbered from right to left, from 0 to $n - 1$
- Bit $i$ of a Gray-code word is 0 if bits $i$ and $i + 1$ of the corresponding binary code word are the same, else bit $i$ is 1. (When $i + 1 = n$, bit $n$ of the binary code is considered to be 0.)

| Decimal Number | Binary Code | Gray Code |
|---|---|---|
| 0 | 000 | 000 |
| 1 | 001 | 001 |
| 2 | 010 | 011 |
| 3 | 011 | 010 |
| 4 | 100 | 110 |
| 5 | 101 | 111 |
| 6 | 110 | 101 |
| 7 | 111 | 100 |

# ASYNCHRONOUS INPUTS MAY REQUIRE A HANDSHAKE PROTOCOL




- **Clocked I/O** – All registers and memories are updated at either the rising edge or the falling edge of the clock signal and are ready to be read at the next transition of the clock signal
- **Handshake** – Protocol used to "broadcast" when output values are ready to be read

# NO SYNCHRONIZER IS NEEDED WITH REFLECTED GRAY CODE

$$00 \rightarrow 01 \rightarrow 11 \rightarrow 10$$

- Input must go through all intermediate values to get from one value to another
- Reading at any time will give valid result
- If value is changing, result will be either the old or the new value; never a spurious value

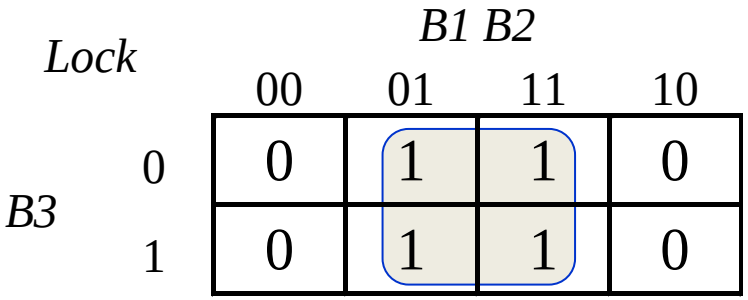
# KARNAUGH MAPS PROVIDE A SIMPLE MINIMIZATION TECHNIQUE

- Graphical minimization technique for three to six variables
- Construct a table with adjacent boxes using **Gray code**
- Adjacency is established along interior and exterior boundaries

| Lock | B1 B2 = 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| B3 B4 = 00 | 0 | 1 | 1 | 1 |
| 01 | 1 | 1 | 1 | 0 |
| 11 | 0 | 0 | 0 | 0 |
| 10 | 0 | 0 | 1 | 1 |

# K-MAPPING REDUCES ADJACENT PAIRS OF PAIRS

K-Map



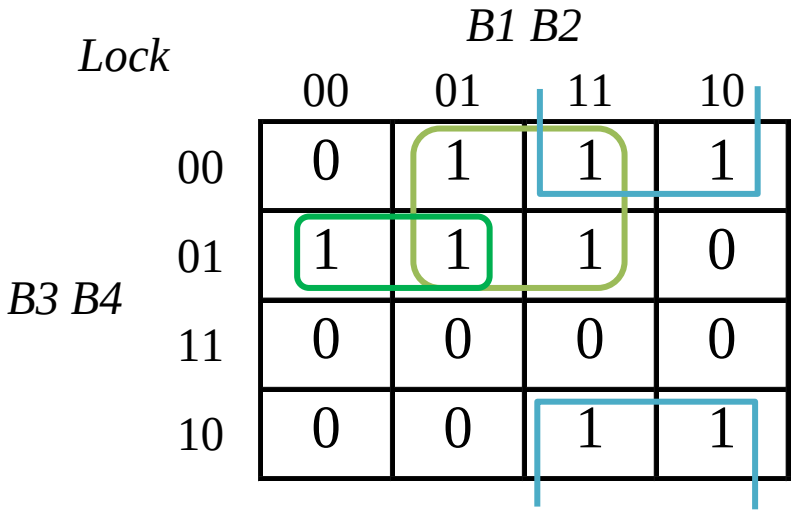

Truth Table

| B1 | B2 | B3 | L |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

- Both B1 and B3 change values while B2 remains fixed
- The result of the minimization is L = B2

# KARNAUGH MAPS ARE EASILY EXTENDED TO 4 VARIABLES

$$L = (B2 \cdot \overline{B3}) + (B1 \cdot \overline{B4}) + (\overline{B1} \cdot \overline{B3} \cdot B4)$$




- Can be extended to 6 variables using multiple tables.
- Beyond 6 variables, other minimization methods must be used.

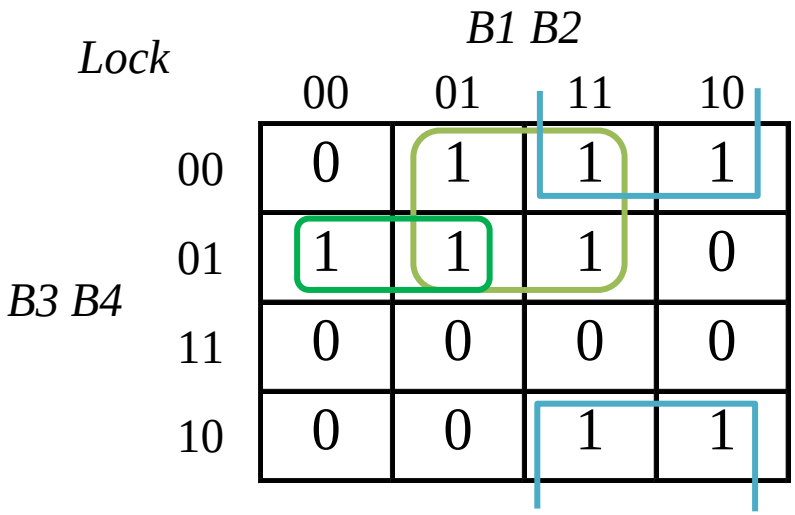
# IMPOSSIBLE INPUT CONDITIONS CALLED "DON'T CARES"

"Don't Care" conditions can be used for:

- Error Checking
  - Illegal or impossible input conditions can generate additional outputs to signal potential malfunction
- Circuit Minimization
  - Cells corresponding to don't care inputs can be set to either 1 or 0 in such a way that the size of the design grouping is increased

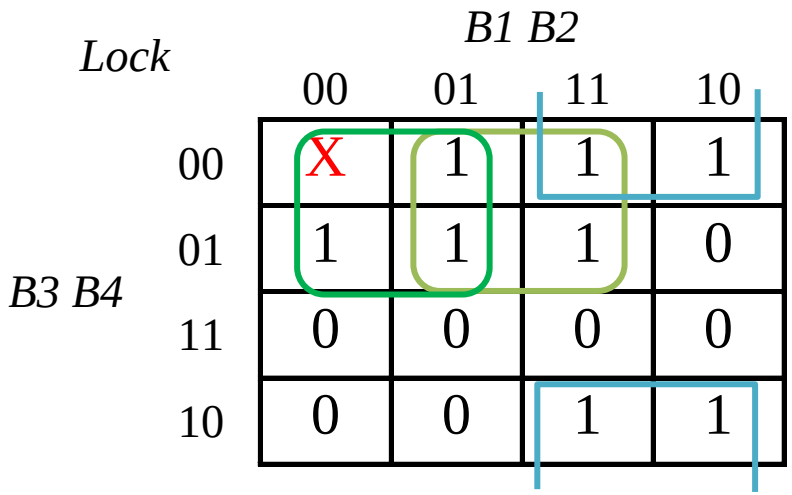
# WITHOUT A DON'T CARE STATE, MINIMIZATION IS LIMITED

$$L = (B2 \cdot \overline{B3}) + (B1 \cdot \overline{B4}) + (\overline{B1} \cdot \overline{B3} \cdot B4)$$



# ADDITIONAL MINIMIZATION IS POSSIBLE WITH "DON'T CARE"

$$L = (B2 \cdot \overline{B3}) + (B1 \cdot \overline{B4}) + (\overline{B1} \cdot \overline{B3})$$



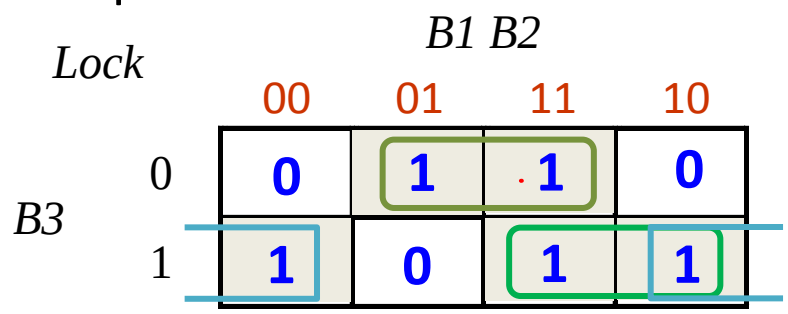
# EXAMPLE: THREE BUTTON DOOR LOCK

Recall $x \cdot y + x \cdot \bar{y} = x \cdot (y + \bar{y}) = x$

Truth Table

| B1 | B2 | B3 | L |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

K-Map




$$y = (\overline{B1} \cdot B2 \cdot \overline{B3}) + (B1 \cdot B2 \cdot \overline{B3}) + (B1 \cdot B2 \cdot B3) + (B1 \cdot \overline{B2} \cdot B3) + (\overline{B1} \cdot \overline{B2} \cdot B3)$$

$$y = (B2 \cdot \overline{B3}) + (B1 \cdot B3) + (\overline{B2} \cdot B3)$$

# EXAMPLE: ROBOT OPERATION

| Thermal Sensor | Front Sensor | Rear Sensor | Drive Motor | Drive Direction |
|---|---|---|---|---|
| OFF | OFF | OFF | ON | FORWARD |
| OFF | OFF | ON | ON | FORWARD |
| OFF | ON | OFF | ON | REVERSE |
| OFF | ON | ON | OFF | DON'T CARE |
| ON | OFF | OFF | OFF | DON'T CARE |
| ON | OFF | ON | OFF | DON'T CARE |
| ON | ON | OFF | OFF | DON'T CARE |
| ON | ON | ON | OFF | DON'T CARE |

ON = 1, OFF = 0
FORWARD = 1, REVERSE = 0
DON'T CARE = X

When drive motor is off, the drive direction is irrelevant!

# EXAMPLE: ROBOT OPERATION

| TS | FS | RS | DM | DD |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | X |
| 1 | 0 | 0 | 0 | X |
| 1 | 0 | 1 | 0 | X |
| 1 | 1 | 0 | 0 | X |
| 1 | 1 | 1 | 0 | X |

ON = 1, OFF = 0
FORWARD = 1, REVERSE = 0
DON'T CARE = X

# EXAMPLE: ROBOT OPERATION

**DM**

| TS \ FS RS | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 |

$$DM = (\overline{TS} \cdot \overline{RS}) + (\overline{TS} \cdot \overline{FS})$$

**DD**

| TS \ FS RS | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 1 | 0 | X | X |
| 1 | 1 | X | X | X |

$$DD = \overline{FS}$$

| TS | FS | RS | DM | DD |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | X |
| 1 | 0 | 0 | 0 | X |
| 1 | 0 | 1 | 0 | X |
| 1 | 1 | 0 | 0 | X |
| 1 | 1 | 1 | 0 | X |

# NON-REDUCIBLE MAPS CAN SOMETIMES BE SIMPLIFIED

- Door Lock example (alternate combination)

| Lock | B1 = 0 | B1 = 1 |
|---|---|---|
| B2 = 0 | **0** | **1** |
| B2 = 1 | **1** | **0** |

$$L = (\overline{B1} \cdot B2) + (B1 \cdot \overline{B2})$$

- Diagonal patterns indicate an XOR relationship:

$$L = B1 \oplus B2$$

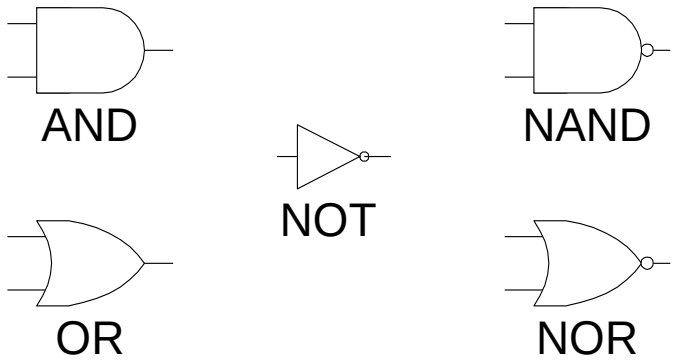
# PHYSICAL REALIZATION OF COMBINATIONAL LOGIC

- Because of digital quantization, relatively easy to implement combinational logic
- Circuits are “almost exact” representations of the mathematics (Boolean algebra)

# PSEUDO CIRCUITS ARE A FIRST STEP INTO CIRCUIT DESIGN

- Convert Boolean equations into logic block diagrams
- Very close to actual circuit elements

- Standard symbols:



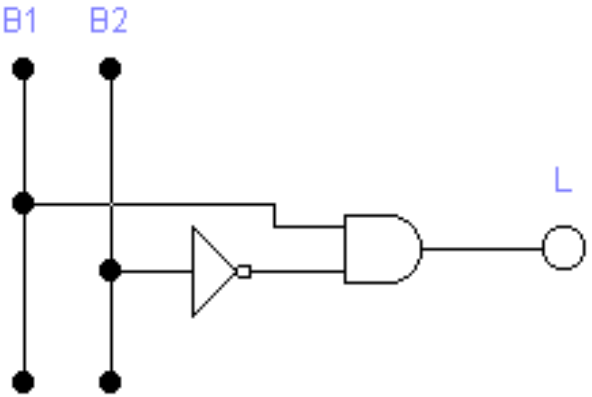

  - NAND gate is NOT(AND): $NAND = \overline{(x \cdot y \cdot \ldots)}$
  - NOR gate is NOT(OR): $NOR = \overline{(x + y + \cdots)}$
  - NAND and NOR are the most common gates (as a result of how semiconductor logic gates are constructed with transistors)

# PSEUDO CIRCUITS PRESENT BOOLEAN LOGIC AS A SCHEMATIC

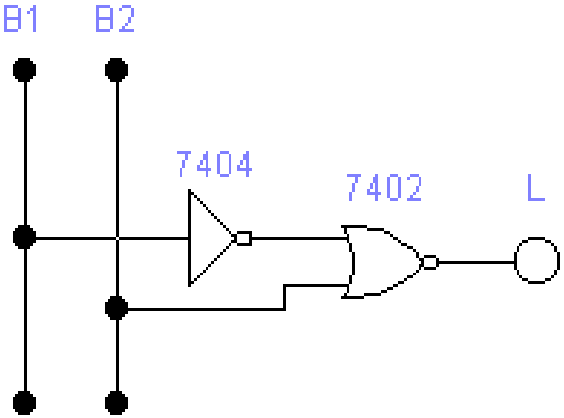
Door lock example:

- Boolean Equation: $L = B1 \cdot \overline{B2} = \overline{(\overline{B1} + B2)}$

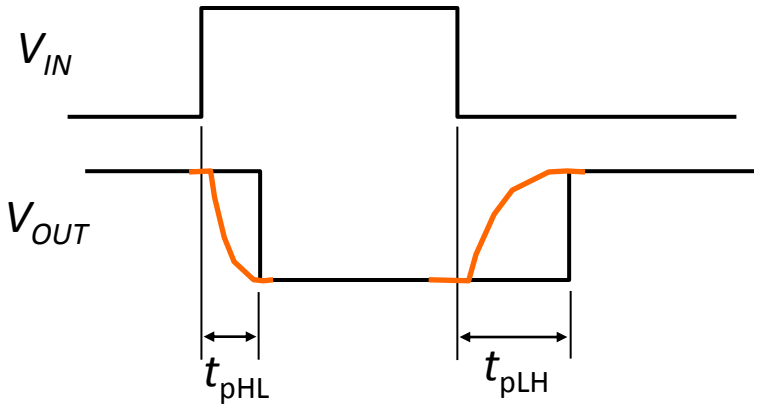
- Pseudo Circuit:




- Convert to NAND/NOR form:



# REAL DEVICES HAVE PROPAGATION DELAYS




- Propagation delay has minimal impact for combinational (Boolean) logic circuits
- Propagation delay is very important when the combinational logic becomes the part of a sequential logic system

# GLITCHES ARE SPURIOUS SIGNALS GENERATED AS INPUTS CHANGE

**Example:**

$$L = (\overline{B1} \cdot \overline{B3} \cdot B4) + (B2 \cdot \overline{B3}) + (B1 \cdot B3 \cdot B4)$$

| *Lock* | *B1 B2* 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 1 | 1 | 0 |
| 01 | 1 | 1 | 1 | 0 |
| 11 | 0 | 0 | 1 | 1 |
| 10 | 0 | 0 | 0 | 0 |

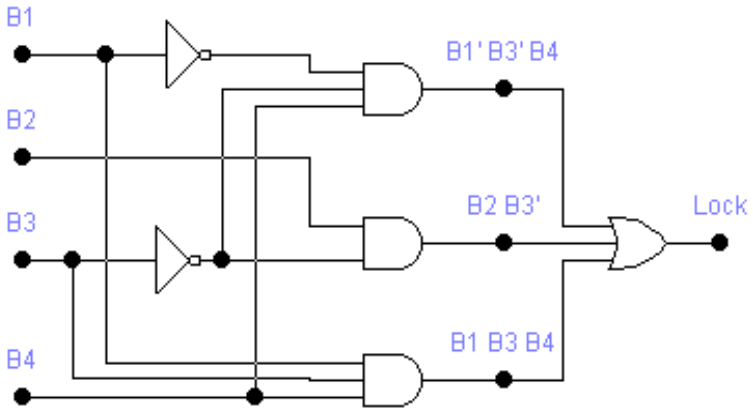
*B3 B4*

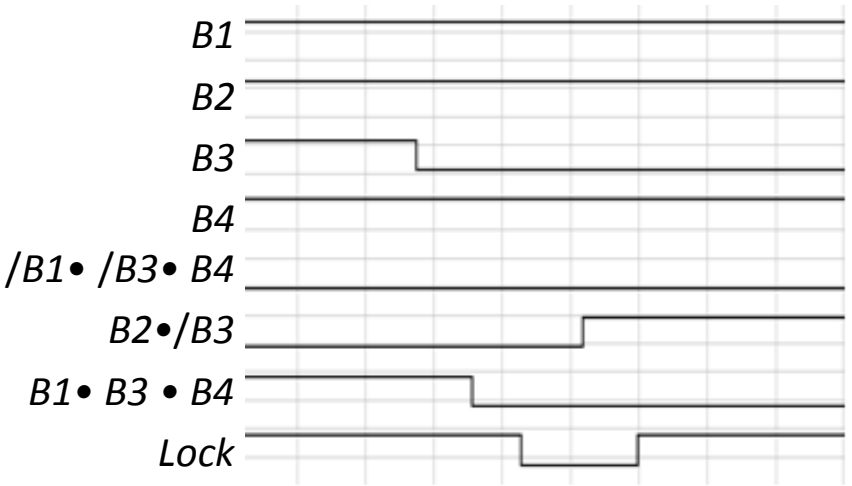
# GLITCHES ARE SPURIOUS SIGNALS GENERATED AS INPUTS CHANGE

## Example:

$$L = (\overline{B1} \cdot \overline{B3} \cdot B4) + (B2 \cdot \overline{B3}) + (B1 \cdot B3 \cdot B4)$$

| Lock | B1 B2 = 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| B3 B4 = 00 | 0 | 1 | 1 | 0 |
| 01 | 1 | 1 | 1 | 0 |
| 11 | 0 | 0 | 1 | 1 |
| 10 | 0 | 0 | 0 | 0 |



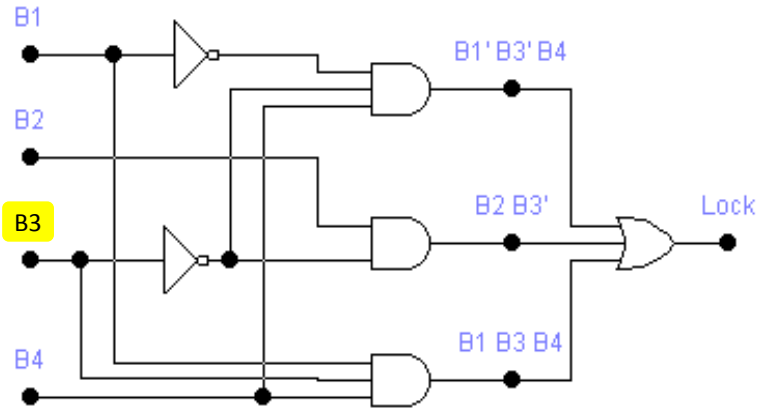
# GLITCHES ARE SPURIOUS SIGNALS GENERATED AS INPUTS CHANGE

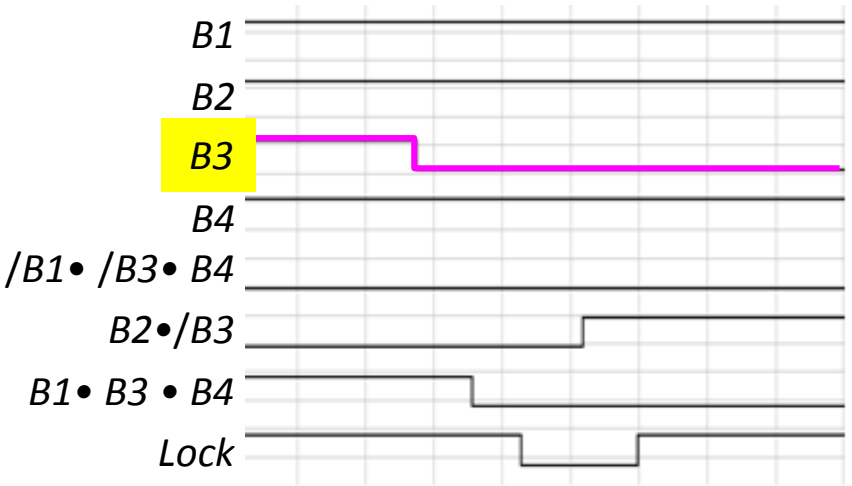
## Example:

$$L = (\overline{B1} \cdot \overline{B3} \cdot B4) + (B2 \cdot \overline{B3}) + (B1 \cdot B3 \cdot B4)$$

| Lock | B1 B2 = 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 1 | 1 | 0 |
| 01 | 1 | 1 | 1 | 0 |
| 11 | 0 | 0 | 1 | 1 |
| 10 | 0 | 0 | 0 | 0 |

(B3 B4 = 00, 01, 11, 10)



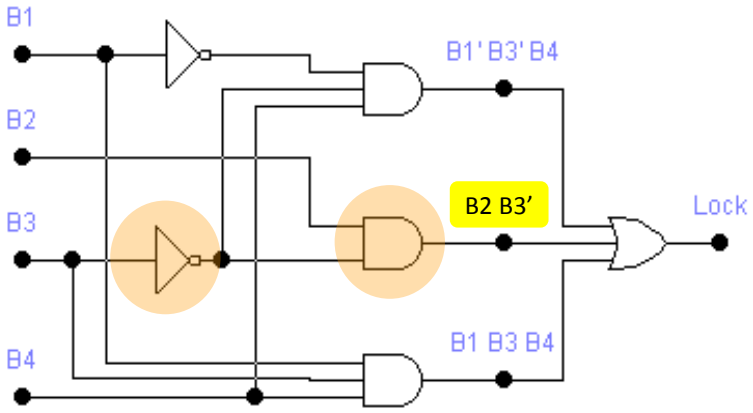
# GLITCHES ARE SPURIOUS SIGNALS GENERATED AS INPUTS CHANGE

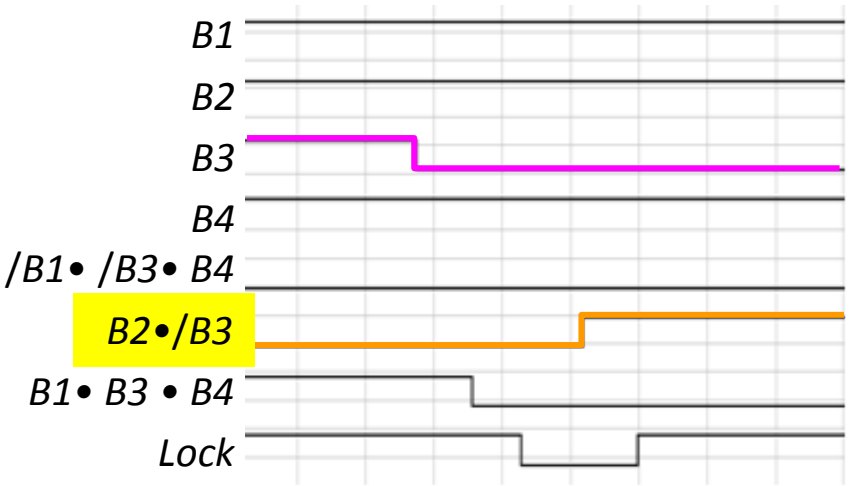
## Example:

$$L = (\overline{B1} \cdot \overline{B3} \cdot B4) + (B2 \cdot \overline{B3}) + (B1 \cdot B3 \cdot B4)$$

Lock

| B3 B4 \ B1 B2 | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 1 | 1 | 0 |
| 01 | 1 | 1 | 1 | 0 |
| 11 | 0 | 0 | 1 | 1 |
| 10 | 0 | 0 | 0 | 0 |



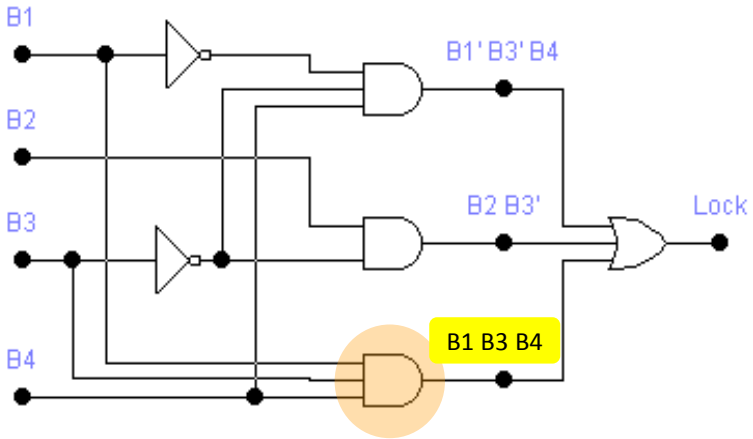
# GLITCHES ARE SPURIOUS SIGNALS GENERATED AS INPUTS CHANGE

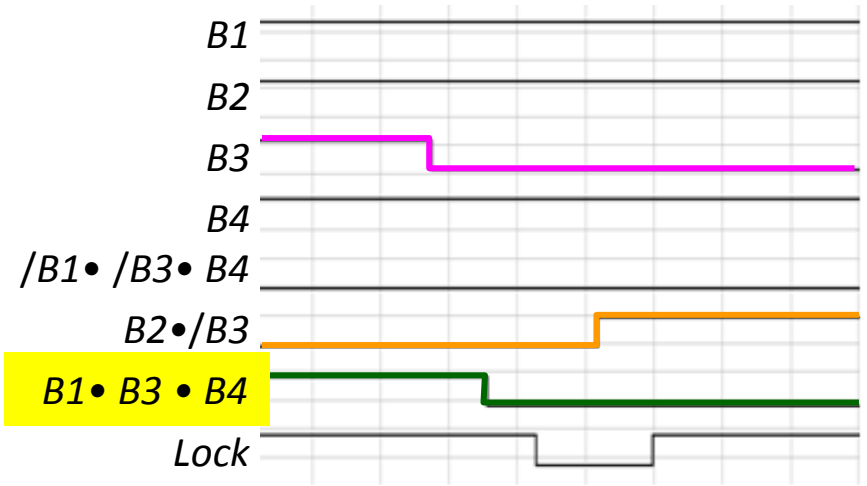
## Example:

$$L = (\overline{B1} \cdot \overline{B3} \cdot B4) + (B2 \cdot \overline{B3}) + (B1 \cdot B3 \cdot B4)$$

Lock

| B3 B4 \ B1 B2 | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 1 | 1 | 0 |
| 01 | 1 | 1 | 1 | 0 |
| 11 | 0 | 0 | 1 | 1 |
| 10 | 0 | 0 | 0 | 0 |



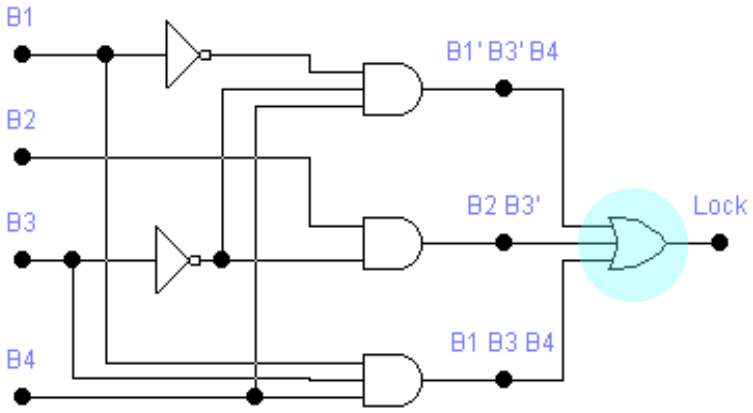
# GLITCHES ARE SPURIOUS SIGNALS GENERATED AS INPUTS CHANGE

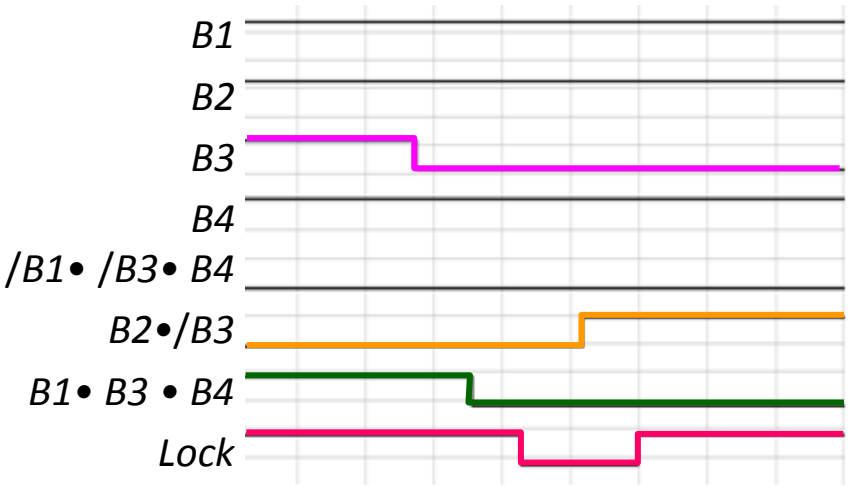
## Example:

$$L = (\overline{B1} \cdot \overline{B3} \cdot B4) + (B2 \cdot \overline{B3}) + (B1 \cdot B3 \cdot B4)$$

Lock

| B3 B4 \ B1 B2 | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 1 | 1 | 0 |
| 01 | 1 | 1 | 1 | 0 |
| 11 | 0 | 0 | 1 | 1 |
| 10 | 0 | 0 | 0 | 0 |



# HAZARDS (GLITCHES)

If device connected to output is static (its output depends only on the current value of the input), an input glitch will show up as an output glitch

- May not cause series problem, since glitches are usually very short.
- Low pass filtering devices (e.g., motors…) will filter out the glitches.

$\Rightarrow$ *Usually will not cause problem…*

# HAZARDS (GLITCHES)

If device connected to the output is dynamic (sequential), it may respond to the momentary change, if the change is long enough to switch one of its inputs

- A counter might respond to the glitch and increment its count

Remedy – "De-minimize" the circuit in a controlled manner, by introducing additional terms to cover the transition

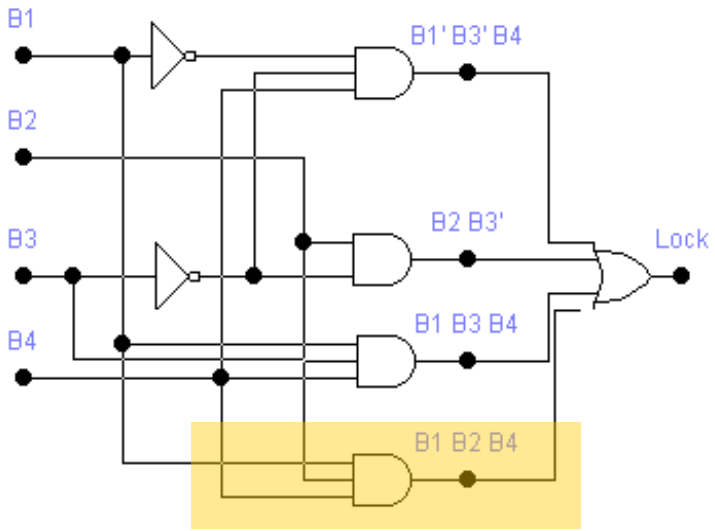
- All transitions between adjacent boxes should take place under a common term – there is always one term that does not change during the transition

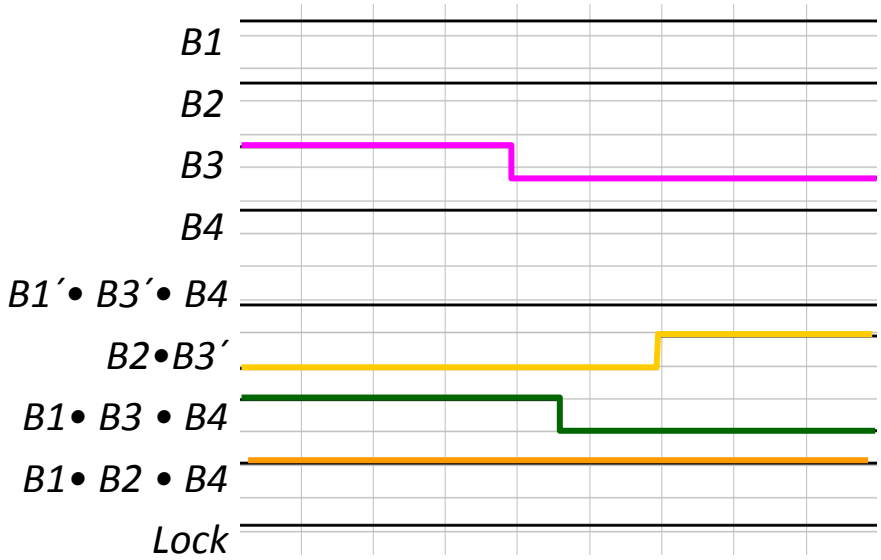
# DEMINIMIZE THE PRIOR EXAMPLE

Previous example:

$$L = (\overline{B1} \cdot \overline{B3} \cdot B4)$$
$$+(B2 \cdot \overline{B3})$$
$$+(B1 \cdot B3 \cdot B4)$$
$$+(B1 \cdot B2 \cdot B4)$$

| Lock | B1 B2 = 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| B3 B4 = 00 | 0 | 1 | 1 | 0 |
| 01 | 1 | 1 | 1 | 0 |
| 11 | 0 | 0 | 1 | 1 |
| 10 | 0 | 0 | 0 | 0 |



# COMING UP…

- Implementing combinational logic
- Sequential logic
- Finite State Machines (FSM)