

Some Aspects of Life in the Trenches

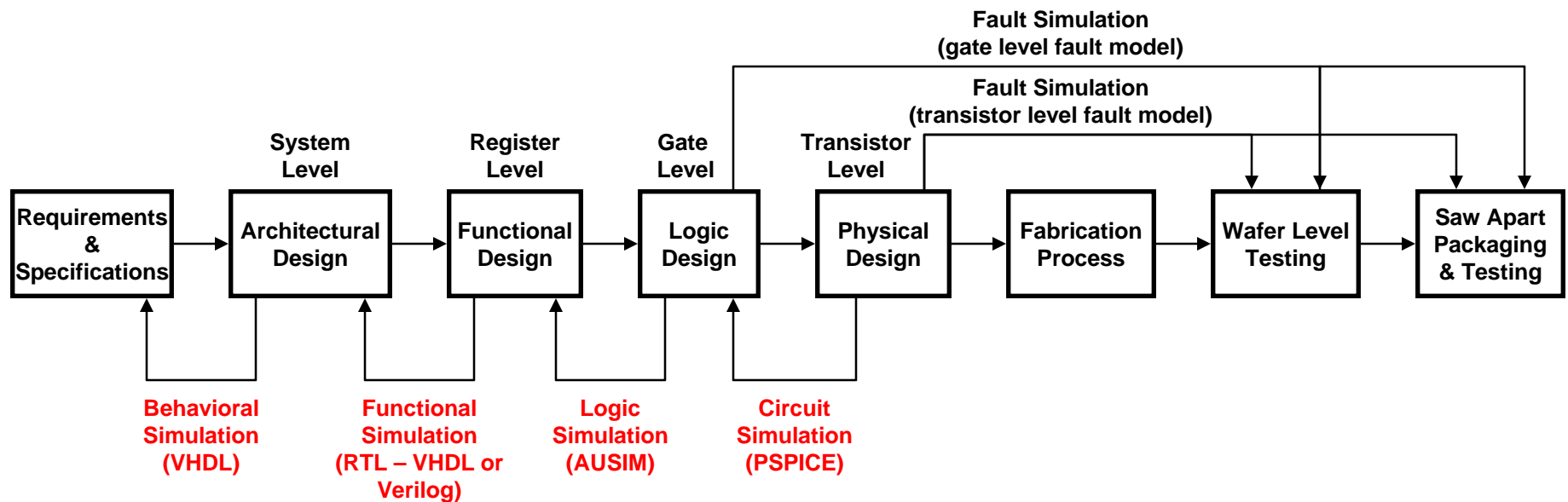
- From my experience, only 10% of the work is creative design
- The rest:
 - **Verifying & debugging your logic design via simulation**
 - **Analyzing whether your design works & meets requirements**
 - Developing & documenting system requirements
 - Risk analysis
 - determining whether and when to take risks
 - Communications
 - Within and between design groups
 - With management and customers
 - Documenting your work
- Good design practices ensure good designs
 - CAD tools are not a magic wand - they don't design the circuit for you!

More Aspects of Life in the Trenches

- Team-oriented development
 - 800 people designed Pentium 4
 - Only 100 people in redesign for cost reduction
- Insufficient design verification
 - Design error in floating-point division in Pentium processor in mid-1990's
 - Intel had to recall 5 million devices
 - It cost Intel \$500,000,000
 - It would be much higher today
 - More devices are shipped
 - It costs more per recall

Example of Digital System Design

- The integrated circuit design process



- Note all the simulation (design verification) - helps to ensure the design works and assists in debugging design errors
- In order to simulate a circuit, we must describe it in a manner that can be interpreted and understood by the simulator.

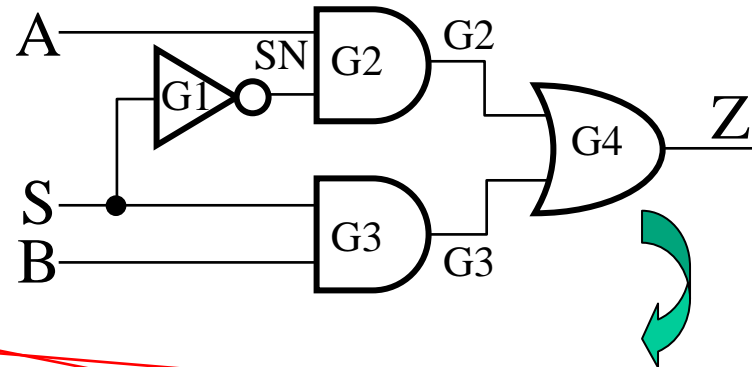
Design Capture with Hardware Description Languages

- Netlist
 - Connections via signal name (like ASL)
- Schematic
 - Connections either explicit or via signal name
 - Produces a netlist for simulation
- Higher level language (VHDL or Verilog)
 - Synthesis to gate level netlist
- All of these design descriptions can go to simulation for design verification

Key Ingredients of Hardware Description Languages (HDLs)

- Circuit statement:

- Circuit name
- Inputs
- Outputs



- Component statements:

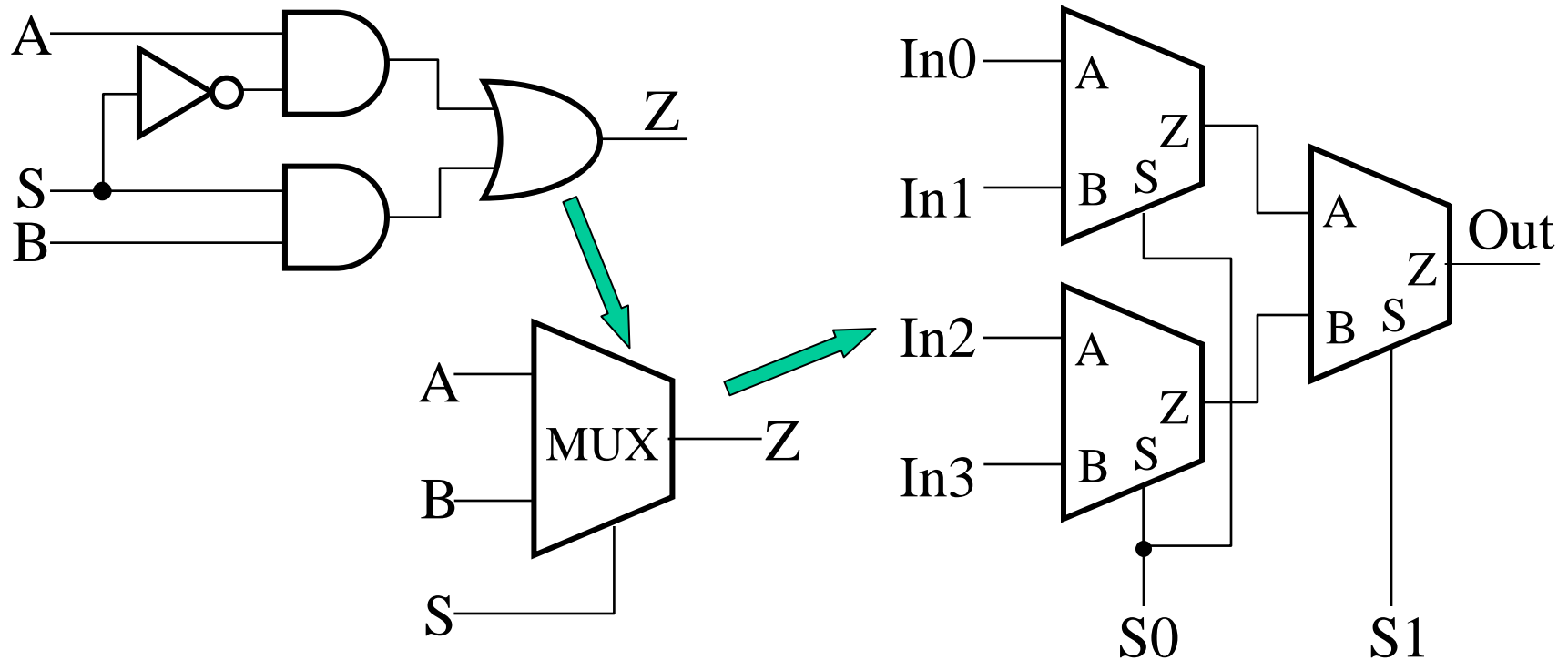
- Component type
- Component instantiation
- Inputs signals
- Output signals

```

ckt: MUX in: A B S out: Z ;
not: G1 in: S out: SN ;
and: G2 in: A SN out: G2 ;
and: G3 in: S B out: G3 ;
or: G4 in: G2 G3 out: Z ;
    
```

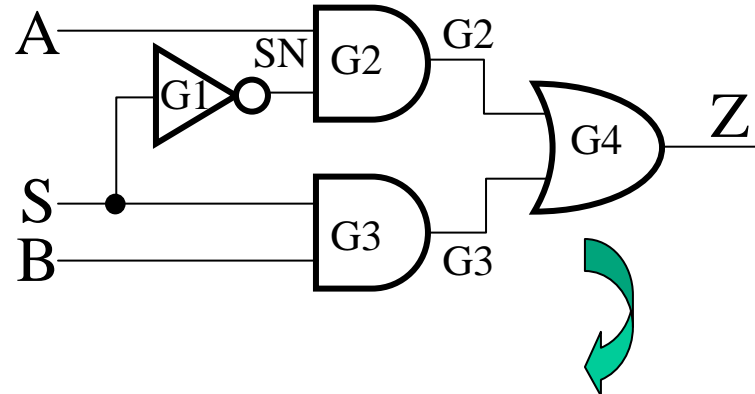
Hierarchical Design

- Hierarchical design saves time & design errors
 - Once a subcircuit has been simulated & is known to work
 - Most HDLs support hierarchical design



Key Ingredients of HDLs

- Hierarchy (subcircuits)
- Connection via signal names (or netnames)



- Keyword notation
 - Positional notation
 - Bus notation
- ```

subckt: MUX in: A B S out: Z ;
not: G1 in: S out: SN ;
and: G2 in: A SN out: G2 ;
and: G3 in: S B out: G3 ;
or: G4 in: G2 G3 out: Z ;
ckt: MUX4 in: In[0:3] S[0:1] out: Out ;
MUX: M1 in: In0 In1 S0 out: M1 ;
MUX: M2 in: In2 In3 S0 out: M2 ;
MUX: M3 in: M1 M2 S1 out: Out ;

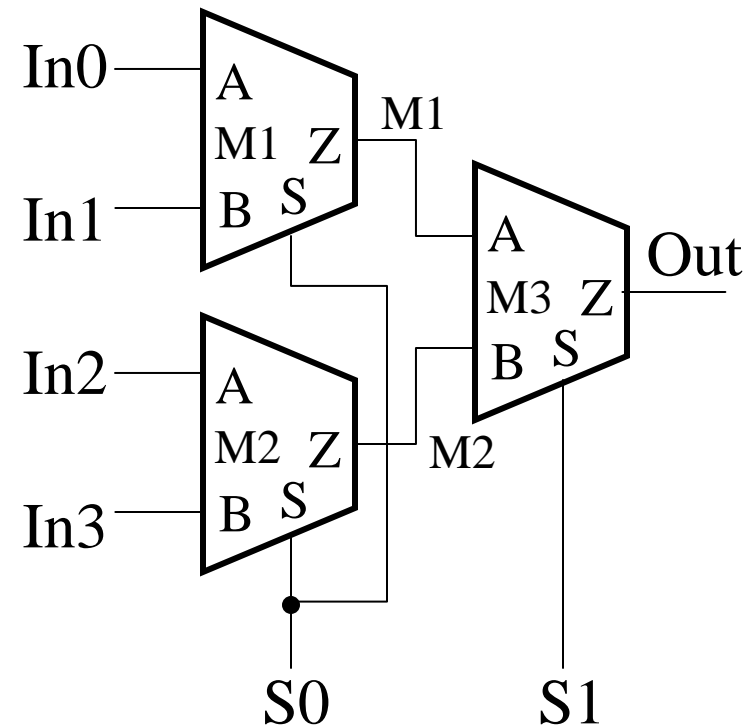
```

# Hierarchical Design

- Hierarchical design supported in most HDLs
  - VHDL, Verilog, ASL

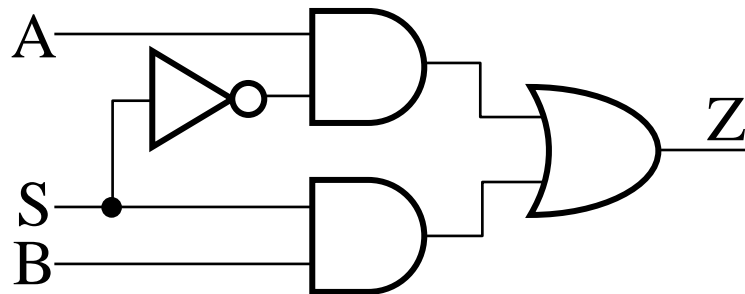
## *Netlist*

```
subckt: MUX in: A B S out: Z ;
not: G1 in: S out: SN ;
and: G2 in: A SN out: G2 ;
and: G3 in: S B out: G3 ;
or: G4 in: G2 G3 out: Z ;
ckt: MUX4 in: In[0:3] S[0:1]
out: Out ;
MUX: M1 in: In0 In1 S0 out: M1 ;
MUX: M2 in: In2 In3 S0 out: M2 ;
MUX: M3 in: M1 M2 S1 out: Out ;
```



# Key Ingredients of Logic Design

*Schematic (or logic diagram)*

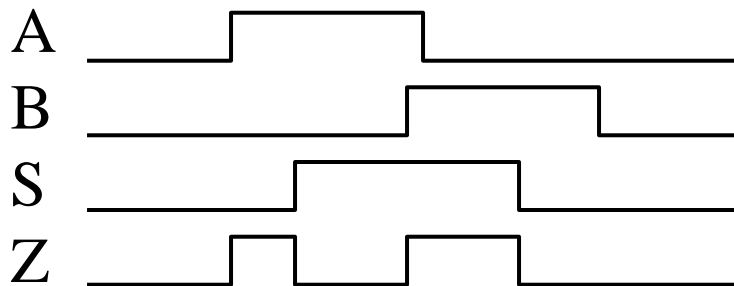


*Truth Table*

| A | B | S | Z |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

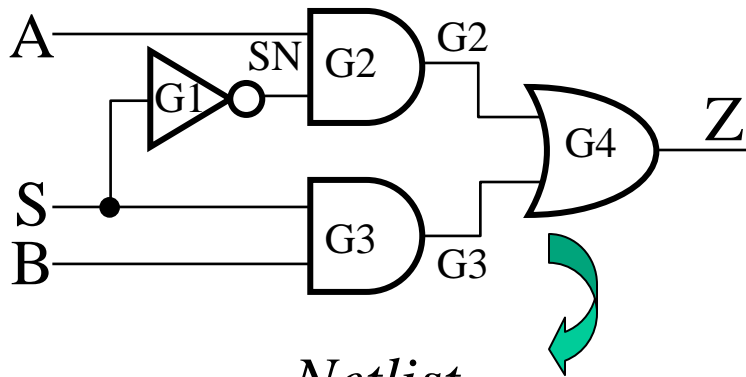
*Logic Equation*  $Z = A \cdot S' + B \cdot S$

*Timing diagram*



# Design Verification

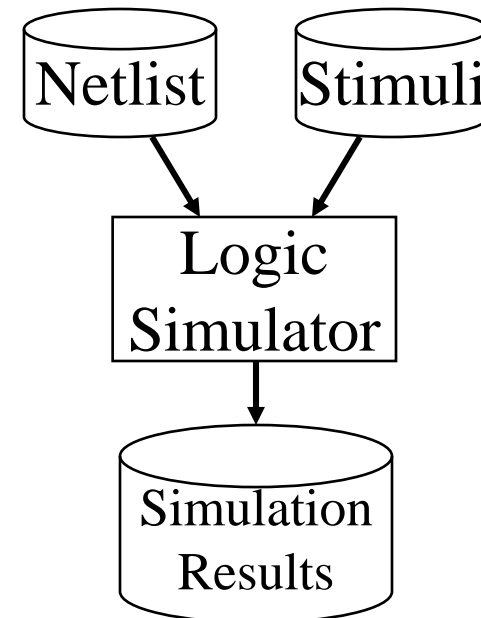
*Schematic (or logic diagram)*



*Netlist*

```
ckt: MUX in: A B S out: Z ;
not: G1 in: S out: SN ;
and: G2 in: A SN out: G2 ;
and: G3 in: S B out: G3 ;
or: G4 in: G2 G3 out: Z ;
```

*Logic Simulation*



# Design Verification

- Logic simulation results used
  - To verify proper operation of design
  - To find and fix problems (errors) in design
    - aka *debugging*

| <i>Input Stimuli</i> |     |  |          | <i>Simulation</i> |     |   |          |
|----------------------|-----|--|----------|-------------------|-----|---|----------|
| <i>(or vectors)</i>  |     |  |          | <i>Results</i>    |     |   |          |
| #                    | ABS |  | <i>i</i> | #                 | ABS | Z | <i>i</i> |
|                      | 000 |  |          |                   | 000 | 0 |          |
|                      | 001 |  |          |                   | 001 | 0 |          |
|                      | 010 |  |          |                   | 010 | 0 |          |
|                      | 011 |  |          |                   | 011 | 1 |          |
|                      | 100 |  |          |                   | 100 | 1 |          |
|                      | 101 |  |          |                   | 101 | 0 |          |
|                      | 110 |  |          |                   | 110 | 1 |          |
|                      | 111 |  |          |                   | 111 | 1 |          |

*compare simulation results to truth table*

# Key Ingredients of Logic Design

- Logic simulation results used

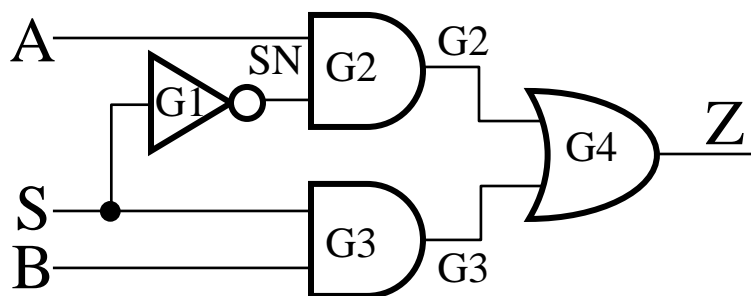
- To verify proper operation of design (design verification)
- To find and fix problems (errors) in design (aka *debugging*)

| <i>Input Stimuli</i><br>(or vectors) |     |          |  | <i>Simulation Results</i> |     |   |          |
|--------------------------------------|-----|----------|--|---------------------------|-----|---|----------|
| #                                    | ABS | <i>i</i> |  | #                         | ABS | Z | <i>i</i> |
|                                      | 000 |          |  |                           | 000 | 0 |          |
|                                      | 001 |          |  |                           | 001 | 0 |          |
|                                      | 010 |          |  |                           | 010 | 0 |          |
|                                      | 011 |          |  |                           | 011 | 1 |          |
|                                      | 100 |          |  |                           | 100 | 1 |          |
|                                      | 101 |          |  |                           | 101 | 0 |          |
|                                      | 110 |          |  |                           | 110 | 1 |          |
|                                      | 111 |          |  |                           | 111 | 1 |          |

# Simulation

## Design Capture Input

### *Schematic diagram*



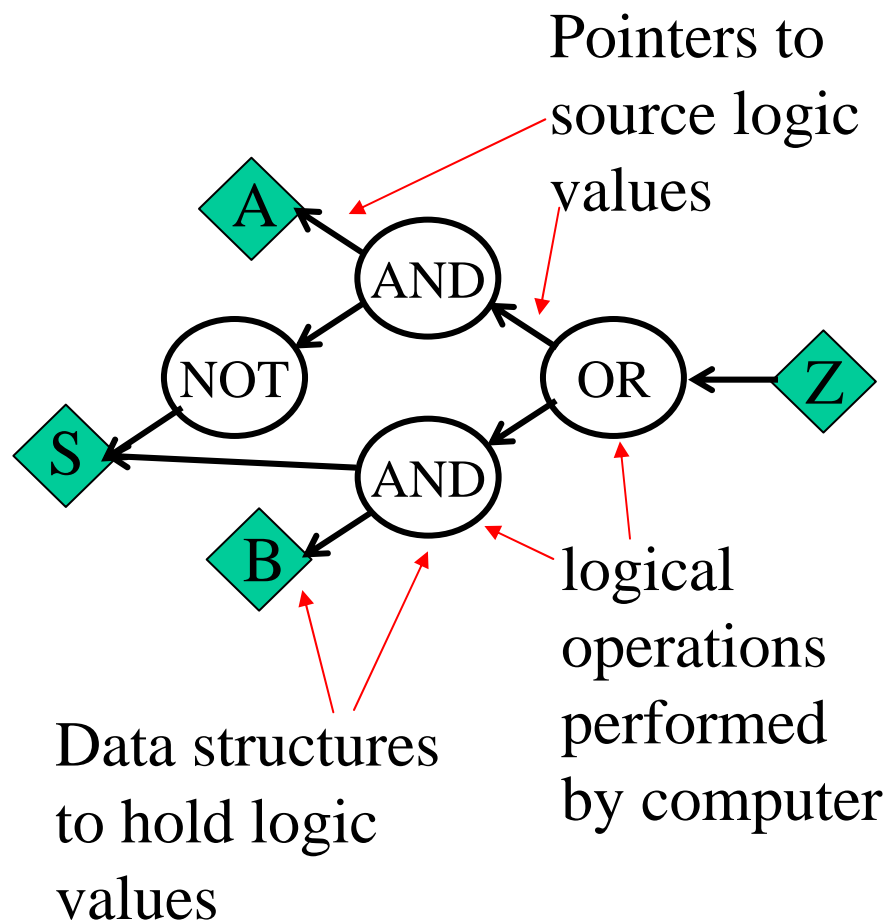
### *Netlist*

```

ckt: MUX in: A B S out: Z ;
not: G1 in: S out: SN ;
and: G2 in: A SN out: G2 ;
and: G3 in: S B out: G3 ;
or: G4 in: G2 G3 out: Z ;

```

## Computer Emulation



# Types of Simulators

- **Compiled Simulator (AUSIM)**
  - Simulation continues until circuit is stable
    - No changing logic values within circuit
  - Aka: unit delay or logic simulator
    - All gates in circuit have a finite unit delay
  - Good for initial design verification
    - Short simulation times
- **Event-Driven Simulator**
  - Simulation events scheduled in time
    - Circuit may not be stable when input changes
  - Aka: timing simulator
    - Gates have real delays base on intrinsic & extrinsic factors
  - More accurate for real circuits
    - longer simulation times and more computer intensive

# Other CAD Tools in Logic Design

- Audits for potential design problems
  - Such as “no-connects” or multiple gates driving same net
    - Usually part of another tool (schematic capture or simulator)
- Logic minimization tools
  - Handles combinational logic circuits too big for K-maps
- Timing analysis
  - Finds and reports worst case timing delay path
    - Like  $P_{del}$  but uses actual timing parameters per gate

# Timing Issues

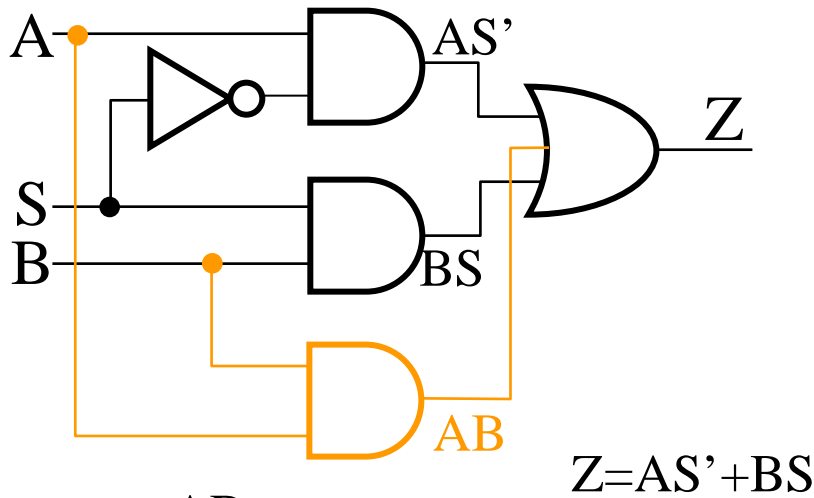
- Normally only steady-state behavior considered
- But transient behavior can also be important
  - Behavior may differ from steady state expectations
- Gates have finite delays
  - Unit gate delay (1 unit of delay per gate)
  - Propagation delay (gate delay  $\propto$  gate size)
- Delays differ through various paths
  - Can lead to ‘glitches’ on combinational logic outputs
    - Called **timing hazards** (or just hazards, with timing implied)
      - Static-1 hazard: produces a glitch to logic 0
      - Static-0 hazard: produces a glitch to logic 1

# Static Hazards

- Definitions:
  - Static-1 hazard caused by a pair of input values that:
    1. Differ by only one input variable (distance = 1),
    2. Both produce a logic 1 output, *and*
    3. Cause a momentary logic 0 (also called a 0 glitch) at the output during a transition from one input value to the other
  - Static-0 hazard caused by a pair of input values that:
    1. Differ by only one input variable (distance = 1),
    2. Both produce a logic 0 output, *and*
    3. Cause a momentary logic 1 (also called a 1 glitch) at the output during a transition from one input value to the other
- Important properties:
  - A properly designed 2-level AND-OR (SOP) circuit has no static-0 hazards, but may have static-1 hazards
  - A properly designed 2-level OR-AND (POS) circuit has no static-1 hazards, but may have static-0 hazards

# Static-1 Hazard Example

*Multiplexer logic diagram*



| S \ AB | 00 | 01 | 11 | 10 |
|--------|----|----|----|----|
| 0      | 0  | 0  | 1  | 1  |
| 1      | 0  | 1  | 1  | 0  |

$Z = AS' + BS + AB$

Note: this group removes hazard

*Timing diagram*

