Advanced Digital Logic Design – EECS 303

http://ziyang.eecs.northwestern.edu/eecs303/

Teacher: Robert Dick
Office: L477 Tech
Email: dickrp@northwestern.edu
Phone: 847–467–2298

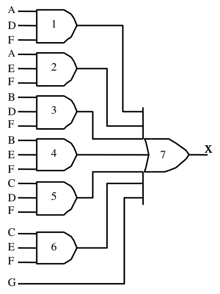
NORTHWESTERN UNIVERSITY

Multi-level example

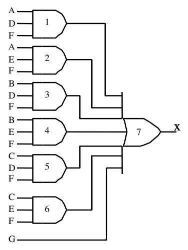
- So far, we have talked about two-level logic
  - Fewest number of levels that is fully general
- However, there are some circuits that can more efficiently be implemented using three or more levels
- Even basic functions, like NANDS and NORS, are commonly composed of multiple gates if the fan-in is high

Two-level form

$X(A, B, C, D, E, F, G) = ADF + AEF + BDF + BEF + CDF + CEF + G$



Two-level form

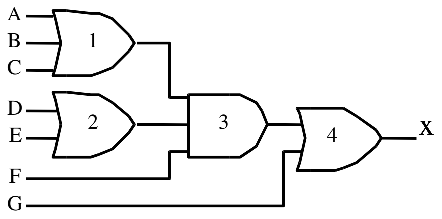


- 6 x 3-input AND gates
- 1 x 7-input OR gate (may not exist!)
- 25 wires (19 literals plus 6 internal wires)

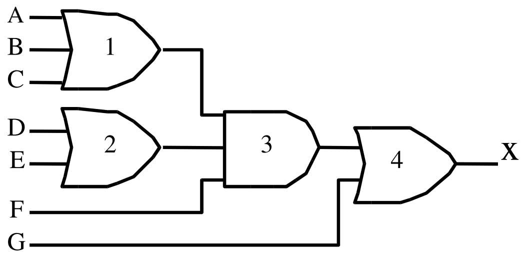
Multi-level form

Factor:

$$\begin{aligned} X(A, B, C, D, E, F, G) &= ADF + AEF + BDF + BEF + \\ &\quad CDF + CEF + G \\ &= (A + B + C)(D + E)F + G \end{aligned}$$

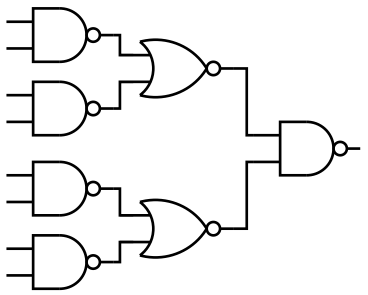


Multi-level form



- 1 x 3-input OR gate, 2 x 2-input OR gates
- 1 x 3-input AND gate
- 10 wires (7 literals plus 3 internal wires)

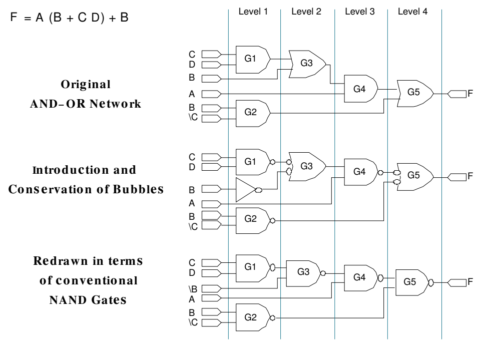
Multi-level implementation of NAND8



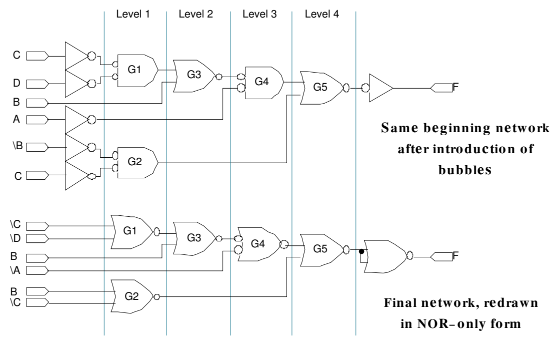
Convert middle ANDs to NORs with inverted inputs

Multi-level logic

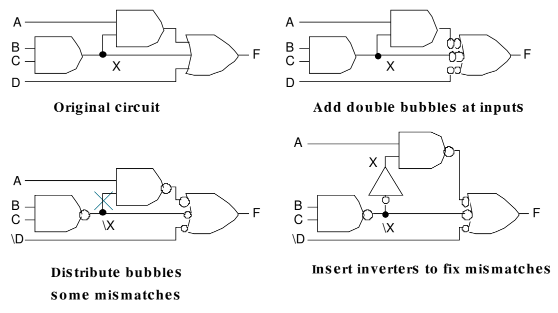
F = A (B + C D) + B



Multi-level logic


Level 1
Level 2
Level 3
Level 4
C
D
B
A
\B
C
G1
G3
G4
G5
G2
F
Same beginning network after introduction of bubbles
\C
\D
B
\A
B
\C
G1
G3
G4
G5
G2
F
Final network, redrawn in NOR-only form

NAND/NOR conversion for uneven paths


A
B
C
D
X
F
Original circuit
A
B
C
D
X
F
Add double bubbles at inputs
A
B
C
\D
\X
F
Distribute bubbles some mismatches
A
X
B
C
\D
\X
F
Insert inverters to fix mismatches

Multi-level minimization objectives

- Factor out common sub-logic (reduce fan-in, increase gate levels), subject to timing constraints
- Map factored form onto library of gates
- Minimize number of literals (correlates with number of wires)

$$X = (AB + \overline{B}\,C)(C + D(E + A\overline{C}\,)) + (D + E)(FG)$$

Multi-level minimization

Can interactively apply the following operations

- Decomposition
- Extraction
- Factoring
- Substitution
- Collapsing

Multi-level minimization

- Recall that two-level minimization is difficult
  - Quine-McCluskey is too slow for large problems
  - Resort to potentially sub-optimal heuristic
    - Espresso
- Multi-level minimization is much harder
  - No known efficient algorithm gives optimal solution
  - MIS uses heuristics to usually produce good solutions

Decomposition

Take a single Boolean expression and replace it with a collection of new expressions

$$L = ABC + ABD + \overline{A}\ \overline{C}\ \overline{D} + \overline{B}\ \overline{C}\ \overline{D} \qquad \text{12 literals}$$

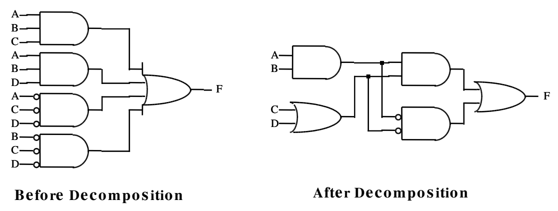
L rewritten as

$$L = MN + \overline{M}\ \overline{N} \qquad \text{8 literals}$$
$$M = AB$$
$$N = C + D$$

Decomposition

$$\begin{aligned} L &= ABC + ABD + \overline{A}\ \overline{C}\ \overline{D} + \overline{B}\ \overline{C}\ \overline{D} \\ &= AB(C + D) + (\overline{A} + \overline{B}\,)\overline{C}\ \overline{D} \\ &= AB(C + D) + \overline{(AB)}\,\overline{(C + D)} \\ M &= AB \\ N &= C + D \\ L &= MN + \overline{M}\ \overline{N} \end{aligned}$$

Decomposition



Extraction

- Extraction is decomposition applied to multiple functions
- The best decomposition for a single function may not be the best if there are multiple outputs
- Ideally, extracted sub-function can be re-used in producing many outputs

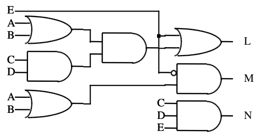
Extraction

$L = (A + B)CD + E$ 11 literals
$M = (A + B)\overline{E}$
$N = CDE$

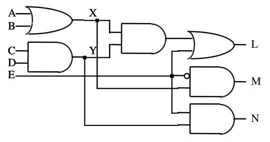
Can be re-written as

$L = XY + E$ 11 literals
$M = X\overline{E}$
$N = YE$
$X = A + B$
$Y = CD$

Extraction



Before Extraction



After Extraction

Gate count has improved, literals is unchanged

Extraction example

Given equations of a Boolean Network

$L = AF + BF + AG + CG + ADE + BDE + CDE$ 33 literals
$M = AF + BF + ACE + BCE$
$N = ADE + CDE$

Find the best kernel of these functions. If the kernel is $(A + B)$, then

$L = DEM + FP + AG + CG$ 22 literals
$M = CEP + FP$
$N = ADE + CDE$
$P = A + B$

Cube extraction

Can restrict divisors to single cubes

$L = ABC + ABD + EG$ 16 literals
$M = ABFG$
$N = BD + EF$

We can obtain the best cube AB to divide the functions

$L = PC + PD + EG$ 15 literals
$M = PFG$
$N = BD + EF$
$P = AB$

Factoring

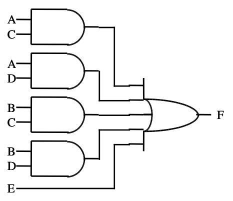
Expression in two-level form re-expressed in multi-level form without introducing intermediate functions

$L = AC + AD + BC + BD + E$ 9 literals

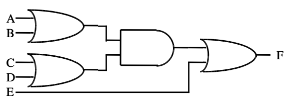
Can be rewritten as

$L = (A + B)(C + D) + E$ 5 literals

Factoring



Before Factoring



After Factoring

Substitution

Express L in terms of M

$L = A + BC$ 5 literals
$M = A + B$

L rewritten in terms of M

$L = M(A + C)$ 5 literals
$M = A + B$

Collapsing

Reverse of substitution. Sometimes eliminates levels to meet timing constraints.

$$L = M(A + C) \qquad \text{5 literals}$$
$$M = A + B \qquad \text{only 2 additional}$$

$$\begin{aligned} L &= (A + B)(A + C) \\ &= AA + AC + AB + BC \\ &= A + BC \end{aligned}$$

Redundancy

- Minimizing gate count and literals isn't always good for performance or power consumption
- Using redundant sub-circuits can result in improvements
- The wiring required to re-use sub-functions can result in larger delays than redundant sub-functions

Boolean division

$L = PQ + R$

P divisor

Q quotient

R remainder

$X = AC + AD + BC + BD + E$

$Y = A + B$

X divided by Y expressed as

$$X = Y(C + D) + E$$

P and Q are symmetrical

Boolean division

Finding divisors that lead greatest number of common subexpressions is difficult

$$L = AD + BCD + E$$
$$M = A + B$$

M does not divide L under algebraic division rules

M does divide L under Boolean rules (very large number of these!)

$$L/M = (A + C)D$$

Boolean division

$$\begin{aligned} L &= AD + BCD + E \\ M &= A + B \\ L &= (A + B)(AD + CD) + E \\ &= (A + B)(A + C)D + E \end{aligned}$$

Boolean division

This follows from writing L in $MQ + R$ form

$$\begin{aligned} L &= M((A + C)D) + E \\ &= (A + B)(A + C)D + E \\ &= (AA + AC + AB + BC)D + E \\ &= (A + BC)D + E \\ &= AD + BCD + E \end{aligned}$$

Multi-level minimization definitions

- A variable is a symbol representing a single coordinate in a Boolean space, e.g., A or B
- Literal is a variable or its negation, e.g., A or $\overline{A}$
- A cube, C, is a set of literals, e.g., $A\ B\ \overline{C}$ or $A\ \overline{B}$
- Consider a sum-of-products expression
  $$L = AF + BF + AG + CG + ADE + BDE + CDE$$
- The primary divisors of an expression form a set of expressions
  $$D(L) = \{L/C | C \text{ is a cube}\}$$

Multi-level minimization definitions

- Equation is cube-free if no cubes divide it evenly
- $AB + C$ is cube-free
- $AB + AC$ is not cube-free
  - Divided evenly by A

Multi-level minimization definitions

- The kernels of an expression are sets of expressions
  $$K(L) = \{G | G \in D(L) \text{ and } G \text{ is cube-free}\}$$
- In other words, the kernels are divisors of the function for cube quotients that can not be evenly divided (remainder-free) by other cubes
- The cube C used to divide L to obtain the kernel $K(L)$ is called the co-kernel

Kernel

$$L = AF + BF + AG + CG + ADE + BDE + CDE$$

Cube	Primary divisor	Kernel (cube free)
A	(F + G + DE)	k_1
B	(F + DE)	k_2
C	(G + DE)	k_3
D	(AE + BE + CE)	no
E	(AD + BD + CD)	no
F	(A + B)	k_4
G	(A + C)	k_5
DE	(A + B + C)	k_6

Divisor selection

Kernels are often good divisors, extract and substitute

$$L = AF + BF + AG + CG + ADE + BDE + CDE \qquad \text{17 literals}$$

Divide by $k_6 = (A + B + C)$

$$L = AF + BF + AG + CG + DEk_6$$

Remaining kernels: $k_1 = (F + G)$, $k_4 = (A + B)$, $k_5 = (A + C)$

Pick one

$$L = Fk_4 + AG + CG + DEk_6$$

Divisor selection

$$L = Fk_4 + AG + CG + DEk_6$$

Remaining kernel: $k_5 = (A + C)$

$$L = Fk_4 + AG + CG + DEk_6$$
$$L = Fk_4 + Gk_5 + DEk_6$$

End result

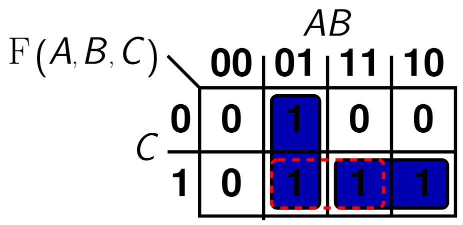
$$L = F(A + B) + G(A + C) + DE(A + B + C) \qquad \text{11 literals}$$

Node simplification

- During multi-level logic synthesis, representation broken into sub-functions
- Each sub-function (e.g., SOP form) needs to be minimized
  $$L = AF + BF + AG + CG + ADE + BDE + CDE$$
- Similar to using Karnaugh Maps, Quine McCluskey, or Espresso to simply expressions

$$L = \overline{A}\ B + AC + BC$$
$$= \overline{A}\ B + AC$$

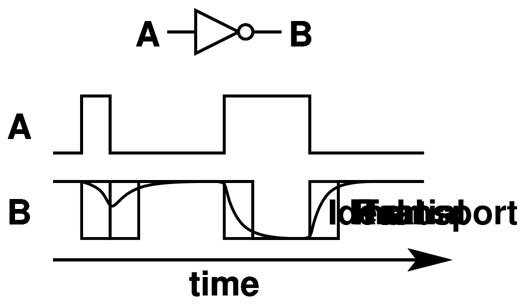
Node simplification



Node simplification

- Recall that two-level simplification can be improved using Don't Care information
- Need to find Don't Care automatically from Boolean network
- Take advantage of satisfiability Don't Cares
  - Input can never occur
- Take advantage of observability Don't Cares
  - Output is ignored for certain inputs

Delay models



Delay models

- Ideal – No delay, outputs respond instantly to inputs
- Transport – Output shifted by some fixed duration
  - All input changes reflected at output

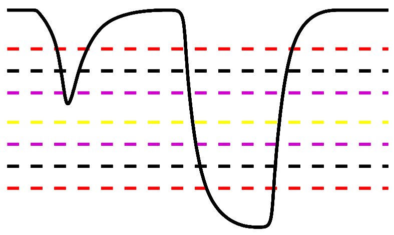
Delay models

- Inertial – Transitions that are not stable for more than some threshold period of time are absorbed by the gate
  - Provides a coarse approximation to real digital logic gate behavior
- Real – Outputs quickly respond to input changes
  - However, output changes have limited slew-rates

RC delay

- What happens when a transistor drives another transistor?
- Recall how CMOS transistors work

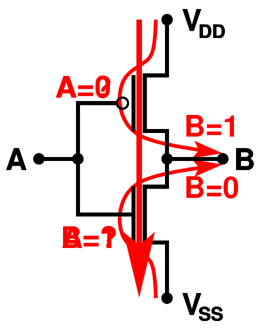
IO thresholds



IO thresholds

- *Digital* is a simplifying concept in a (macroscopically) *analog* world
- To treat signals as digital, need to define a dividing line between false and true
- What actually happens to the next stage when the voltage is at that line?
  - NMOS $V_{GS} > V_T$
  - PMOS $V_{GS} < -V_T$

Molten transistor



IO thresholds

- Define valid ranges of logic high and logic low signals
- Undefined signals considered invalid

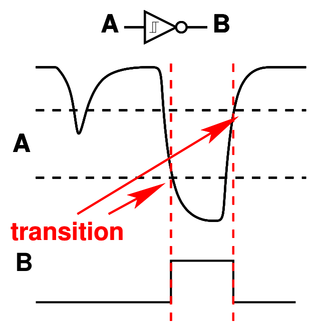
IO thresholds

- Better – However, still have a problem
- What happens if the output of a gate is near that threshold?
- Slight variation between gates might result in crossing this threshold for next gate
- How to compensate?
- Use separate input and output thresholds

Separate IO thresholds

- Now, we can safely treat the system as digital
- However, digital systems talk with analog systems
- It is necessary to deal with noisy signals
- Real slew isn't ideal

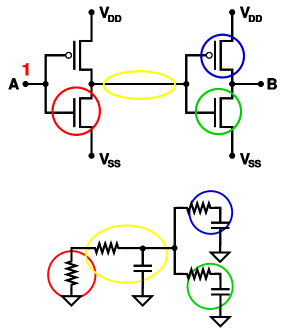
Schmitt triggers



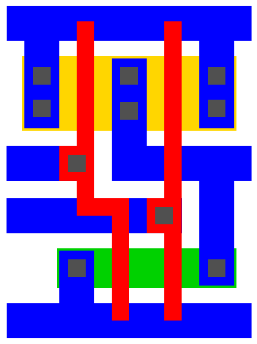
Reason for gradual transition

- A logic stage is an RC network
- Whenever a transition occurs, capacitance is driven through resistance
- Consider the implementation of a CMOS inverter

CMOS NOT is RC network



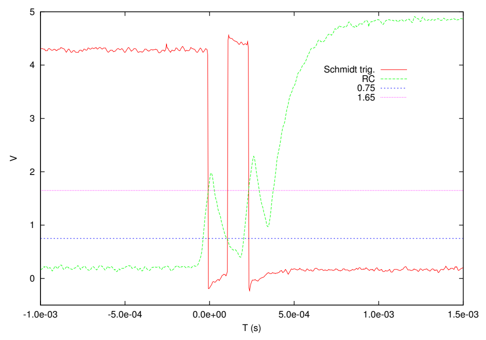
What is driven in NAND2?



Debouncing

- Mechanical switches bounce!
- What happens if multiple pulses?
  - Mutliple state transitions
- Need to clean up signal

Debouncing



Improving device delay

- Decrease driven resistance and load
  - Use smaller transistors
  - Use shorter wire
  - Use wider wire (can increase load)

Improving device delay

- Increase drive by increasing width of driving transistors
  - Causes problems
  - Larger transistors provide larger loads to their inputs
  - In general, keep transistors small unless they absolutely need to drive large loads

Driving large loads

- Sometimes large loads need to be driven
- Long wires
- Output pads
- What happens if we go from a minimum-size inverter to a huge inverter?
  - Huge delay driving huge inverter's gate

Tapered buffer chain

- Instead, gradually increase buffers in chain
- Optimal number of stages: $\ln\left(C_{BIG}/C_{SMALL}\right)$
- Stage width exponentially increases in α

$$W_1 = W_{MIN} \cdot \alpha^0$$
$$W_2 = W_{MIN} \cdot \alpha^1$$
$$W_3 = W_{MIN} \cdot \alpha^2$$
$$\ldots$$

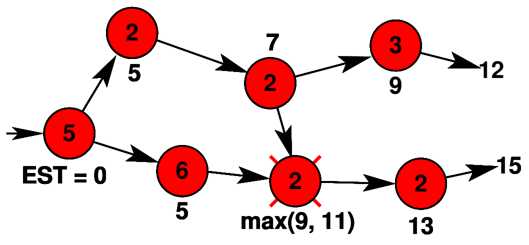
Useful delay: Debouncing

- Mechanical switches bounce
  - Noisy transition
- Use RC delay network to decrease transition speed
- Convert multiple noisy to single smooth transition
- Use Schmitt trigger to clean signal

Delay estimation in multi-level circuits

- Can get delay for a gate by knowing its drive (resistance) and the load it drives (RC)
  - Gate libraries will have this information
- Still need to get network delay
- Conduct topological sort of network to find earliest start times (EST)
- Always visit a node's parent before visiting it
- EST is maximum any parent's EST plus it's delay

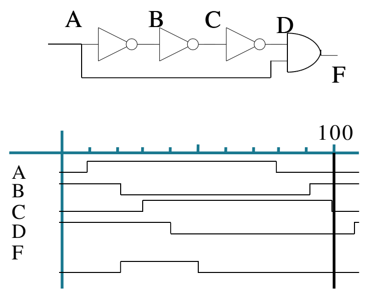
Topological sort



Slack

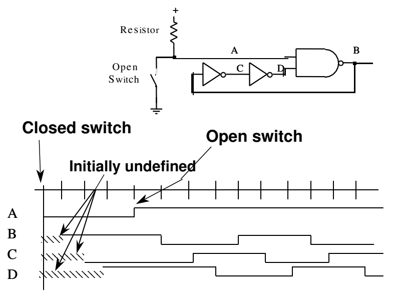
- Can use reverse topological sort from end nodes (given some target delay) to get latest start time (LST)
- Subtract delays and take minimum over children instead of adding delays and taking maximum over parents
- Subtract EST from LST to get node slack
- Gates with lowest slack are on critical path
- Make this path faster. . .
- . . . or save area (at the expense of speed) on non-critical paths

Useful delay: Rising edge pulse shaping



Problem: Pulse width poorly controlled

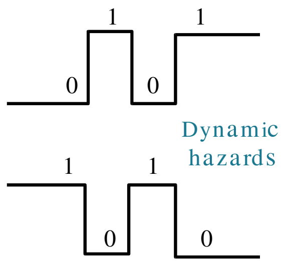
Useful delay: Pulse rising/falling edge pulse shaping



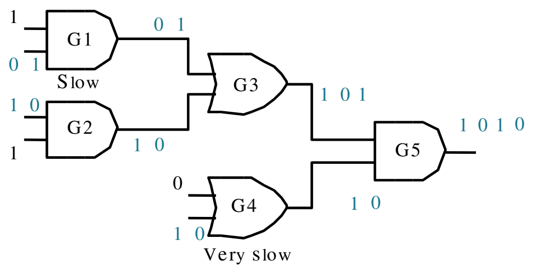
Problem: Pulse width poorly controlled

Dynamic hazards

- Potential for two or more spurious transitions before intended transition
- Results from uneven path delays in some multi-level circuits



Dynamic hazards

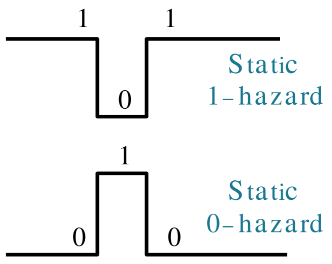


Eliminating dynamic hazards

- Some approaches allow preservation of multi-level structure
  - Quite complicated to apply
- Simpler solution – Convert to two-level implementation

Static hazards

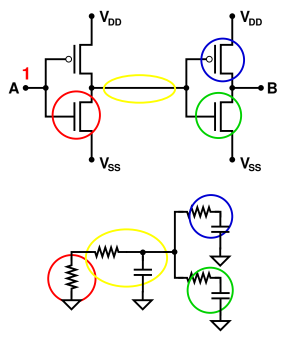
- Still have static hazards
- Potential for transient change of output to incorrect value



Problems with glitches

- These transitions result in incorrect output values at some times
- Also result in uselessly charging and discharging wire and gate capacitances through wire, gate, and channel resistances
  - Increase power consumption

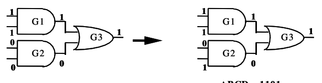
Glitches increase power consumption



Detecting hazards

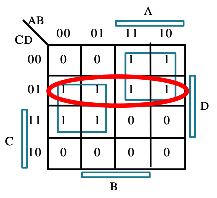
- The observable effect of a hazard is a glitch
  - A circuit that might exhibit a glitch has a hazard
- Whether or not a hazard is observed as a glitch depends on relative gate delays
- Relative gate delays change depending on a number of factors – Conditions during fabrication, temperature, age, etc.
- Best to use abstract reasoning to determine whether hazards might be observed in practice, under some conditions

Eliminating static hazards



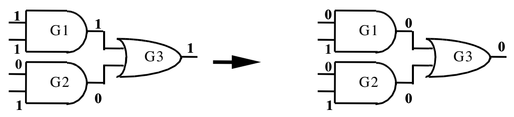
ABCD = 1100 ABCD = 1101

input change within product term

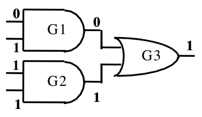


F = $\overline{A}$ D + A $\overline{C}$

Eliminating static hazards



ABCD = 1101 ABCD = 0101 ($\overline{A}$ is still 0)



ABCD = 0101 ($\overline{A}$ is 1)

Eliminating static hazards

- Add redundant primes covering all 1-1 transitions in SOP
- Add redundant primes covering all 0-0 transitions in POS
- Clearly primes can be used, consider contradiction stemming from assumption that non-prime is necessary to cover a transition

Where do static hazards really come from?

- Static-0: $A\,\overline{A}$
- Static-1: $A + \overline{A}$
- Assume SOP form has no product terms containing a variable in complemented and uncomplemented forms
  - Reasonable assumption, if true, drop product term

Where do static hazards really come from?

- Assume POS form has no sum terms containing a variable in complemented and uncomplemented forms
  - Reasonable assumption, if true, drop sum term
- Assume only one input switches at a time
- Conclusion: SOP has no 0-hazards and POS has no 1-hazards
  - In other words, if you are doing two-level design, you need not analyze the other form for hazards

Living with hazards

Sometimes hazards can be tolerated

- Combinational logic whose outputs aren't observed at all times
- Synchronous systems
- Systems without tight power consumption limits

Summary

- Brief review of cascaded carry lookahead adder
- Common ALU operations
- Overview of memory types
- Timing behavior