Chapter # 3: Multi-Level Combinational Logic

Contemporary Logic Design

Randy H. Katz
University of California, Berkeley

June 1993

Chapter Overview

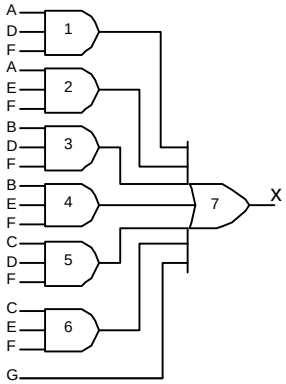
- ***Multi-Level Logic***
  - **Conversion to NAND-NAND and NOR-NOR Networks**
  - **DeMorgan's Law and Pushing Bubbles**
  - **AND-OR-Invert Building Blocks**
  - **CAD Tools for Multi-Level Optimization**
- ***Time Response in Combinational Networks***
  - **Gate Delays and Timing Waveforms**
  - **Hazards/Glitches and How To Avoid Them**

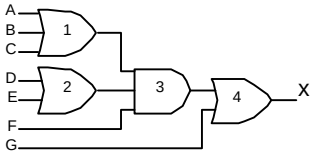
Multi-Level Logic: Advantages

Reduced sum of products form:

$x = A\,D\,F + A\,E\,F + B\,D\,F + B\,E\,F + C\,D\,F + C\,E\,F + G$

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





Factored form:

$x = (A + B + C)(D + E)\,F + G$

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 Logic: Conversion of Forms

NAND-NAND and NOR-NOR Networks

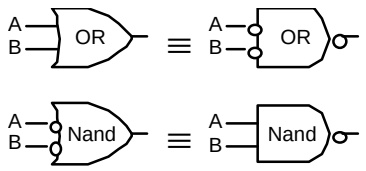
DeMorgan's Law: $(A + B)' = A' \cdot B'$; $(A \cdot B)' = A' + B'$

Written differently: $A + B = (A' \cdot B')'$; $(A \cdot B) = (A' + B')'$

In other words,
OR is the same as NAND with complemented inputs
AND is the same as NOR with complemented inputs
NAND is the same as OR with complemented inputs
NOR is the same as AND with complemented inputs

OR/NAND Equivalence

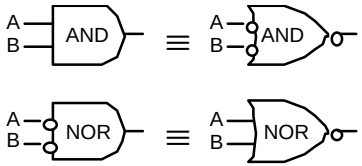
A	$\overline{A}$	B	$\overline{B}$	$A + B$	$\overline{A} \cdot \overline{B}$	$\overline{A} + \overline{B}$	$A \cdot B$
0	1	0	1	0	0	1	1
0	1	1	0	1	1	1	1
1	0	0	1	1	1	1	1
1	0	1	0	1	1	0	0



Mult-Level Logic: Conversion Between Forms

AND/NOR Equivalence

A	$\bar{A}$	B	$\bar{B}$	A • B	$\overline{A + B}$	$\bar{A} \cdot \bar{B}$	$\overline{A \cdot B}$
0	1	0	1	0	0	1	1
0	1	1	0	0	0	0	0
1	0	0	1	0	0	0	0
1	0	1	0	1	1	0	0

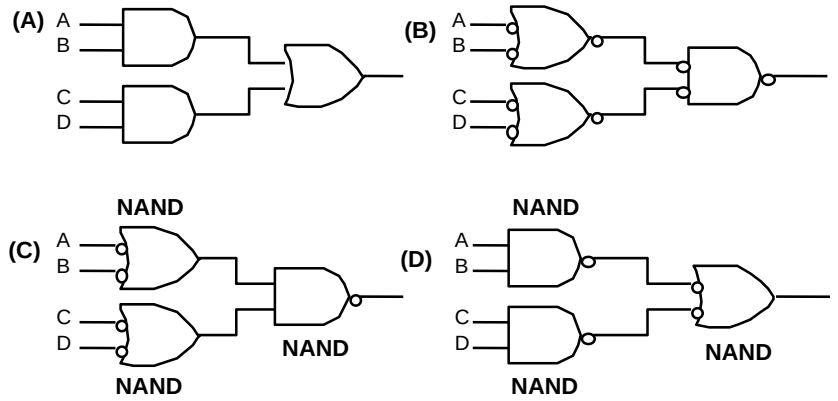


It is possible to convert from networks with ANDs and ORs to networks with NANDs and NORs by introducing the appropriate inversions ("bubbles")

To preserve logic levels, each introduced "bubble" must be matched with a corresponding "bubble"

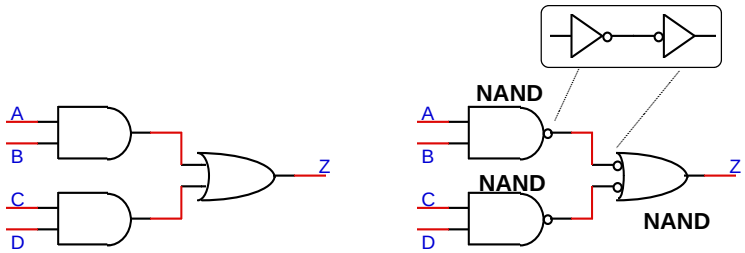
Multi-Level Logic: Conversion of Forms

Example: Map AND/OR network to NAND/NAND network



Multi-Level Logic: Conversion of Forms

Example: Map AND/OR network to NAND/NAND network



Verify equivalence of the two forms

Z = [(A•B)' (C•D)']'

= [(A' + B') (C' + D')]'

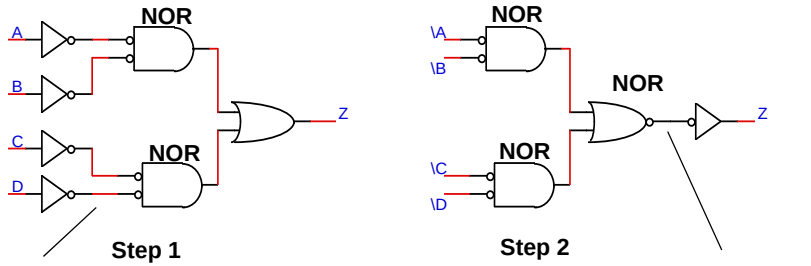
= [(A' + B')' • (C' + D')']

= (A • B) + (C • D)

This is the easy conversion!

Multi-Level Logic: Mapping Between Forms

Example: Map AND/OR network to NOR/NOR network

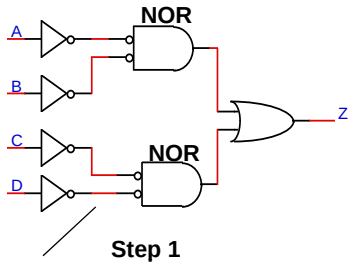


Z =

Verify equivalence of the two forms

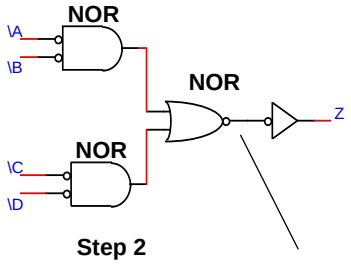
Multi-Level Logic: Mapping Between Forms

Example: Map AND/OR network to NOR/NOR network



Step 1

Conserve "Bubbles"



Step 2

Conserve "Bubbles"

Verify equivalence of the two forms

Z = {[(A' + B')' + (C' + D')']'}'

= {(A' + B') • (C' + D')}'

= (A' + B')' + (C' + D')'

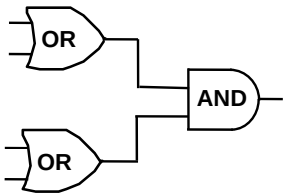
= (A • B) + (C • D)

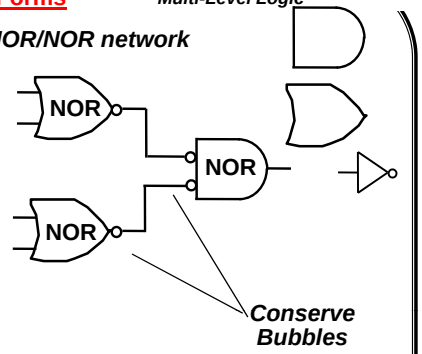
This is the hard conversion!

AND/OR to NAND/NAND more natural

Multi-Level Logic: Mapping Between Forms

Example: Map OR/AND network to NOR/NOR network





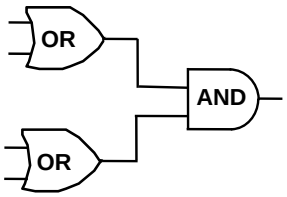
Conserve Bubbles

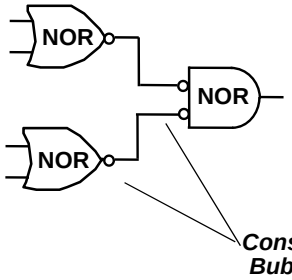
Verify equivalence of the two forms

Z =

Multi-Level Logic: Mapping Between Forms

Example: Map OR/AND network to NOR/NOR network





Conserve Bubbles

Verify equivalence of the two forms

Z = [(A + B)' + (C + D)']'

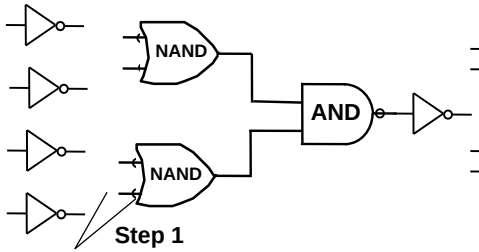
= {(A + B)'}' • {(C + D)'}'

= (A + B) • (C + D)

This is the easy conversion!

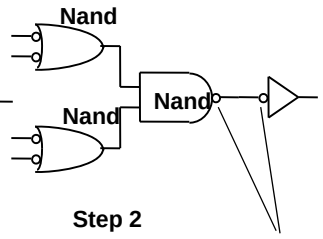
Multi-Level Logic: Mapping Between Forms

Example: Map OR/AND network to NAND/NAND network



Step 1

Conserve Bubbles!



Step 2

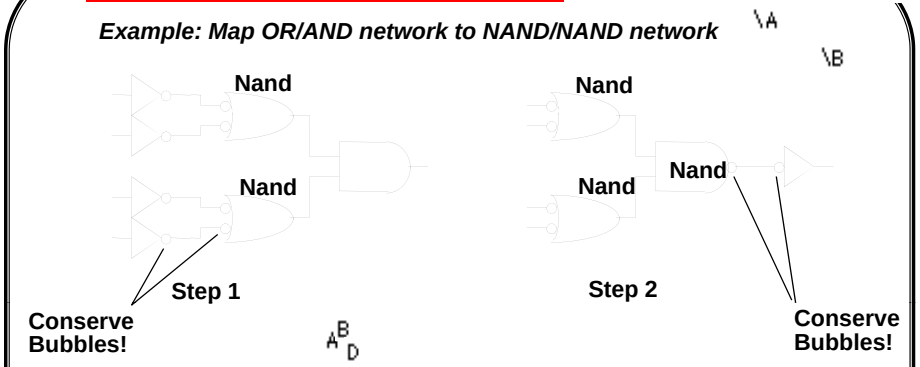
Conserve Bubbles!

Z =

Verify equivalence of the two forms

Multi-Level Logic: Mapping Between Forms

Example: Map OR/AND network to NAND/NAND network



Verify equivalence of the two forms

$Z = \{[(A' \cdot B')' \cdot (C' \cdot D')']'\}'$

$= \{(A' \cdot B') + (C' \cdot D')\}'$

$= (A' \cdot B')' \cdot (C' \cdot D')'$

$= (A + B) \cdot (C + D)$

This is the hard conversion!

OR/AND to NOR/NOR more natural

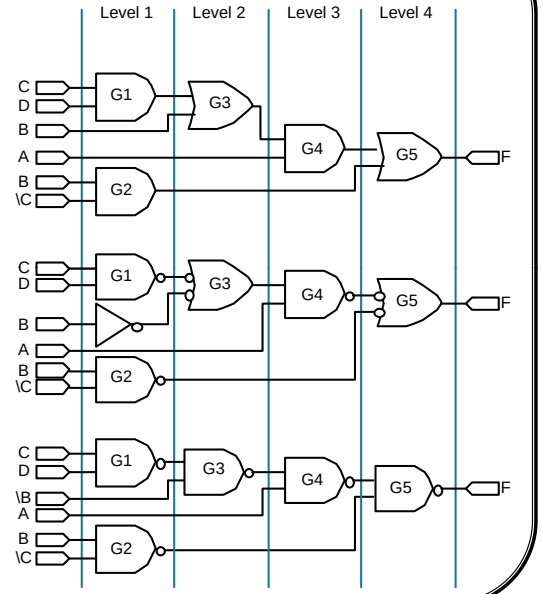
Multi-Level Logic: More than Two Levels

$f = A\,(B + C\,D) + B\,C'$

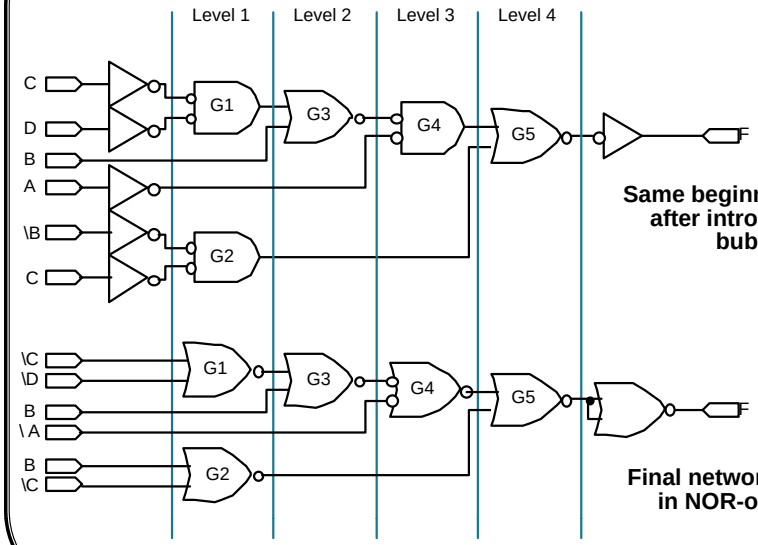
Original AND-OR Network

Introduction and Conservation of Bubbles

Redrawn in terms of conventional NAND Gates



Multi-Level Logic: More than Two Levels

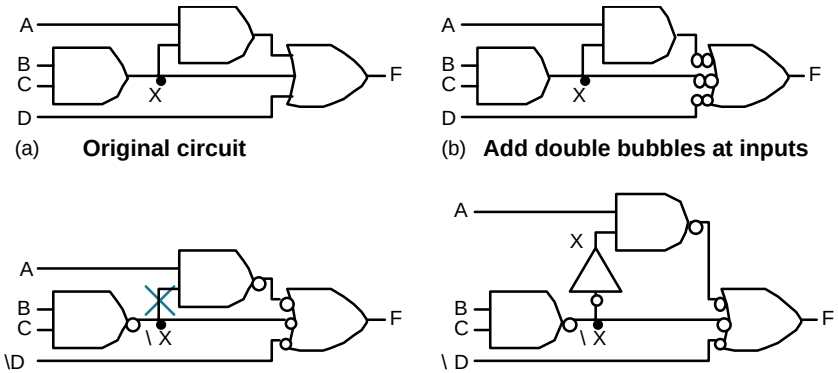


Same beginning network after introduction of bubbles

Final network, redrawn in NOR-only form

Multi-Level Logic: More than Two-Levels

Conversion Example



(a) **Original circuit**

(b) **Add double bubbles at inputs**

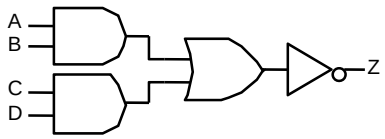
(c) **Distribute bubbles some mismatches**

(d) **Insert inverters to fix mismatches**

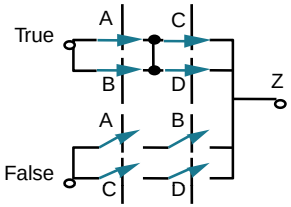
Multi-Level Logic: AND-OR-Invert Block

AOI Function: **Three stage logic— AND, OR, Invert**
Multiple gates "packaged" as a single circuit block

logical concept

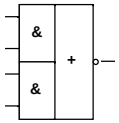


possible switch implementation

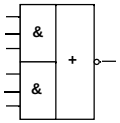


AND OR Invert
two-input two-stack

2x2 AOI Schematic Symbol



3x2 AOI Schematic Symbol



Multi-Level Logic: AND-OR-Invert

Example: XOR implementation

A xor B = A' B + A B'

= (?)'

AOI form

(A' B + A B')'

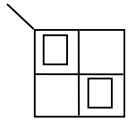
(A + B') (A' + B)

(A B + A' B')

General procedure to place in AOI form:

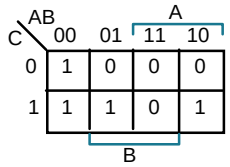
Compute the complement in Sum of Products form by circling the 0's in the K-map!

***f* = (A' B' + A B)'**



Multi-Level Logic: AND-OR-Invert

Example:



F' K-map

F = B C' + A C' + A B

F' = A' B' + A' C + B' C

Implemented by 2-input 3-stack AOI gate

F = (A + B) (A + C') (B + C')

F' = (B' + C) (A' + C) (A' + B')

Implemented by 2-input 3-stack OAI gate

Example:

4-bit Equality Function

Z = (A0 B0 + A0' B0') (A1 B1 + A1' B1') (A2 B2 + A2' B2') (A3 B3 + A3' B3')

Each implemented in single 2x2 AOI gate

Multi-Level Logic: AND-OR-Invert

Example: AOI Implementation of a 4-Bit Equality Tester

High if A0 ° B0, Low if A0 = B0
A = B active low

Conservation of bubbles

NOR

If all inputs are low
(asserted in negative logic)
then Ai = Bi, i=0,...,3
Output Z asserted

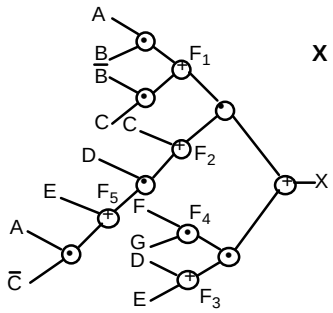
Multi-Level Logic: CAD Tools for Simplification

Multi-Level Optimization:

1. **Factor out common sublogic (reduce fan-in, increase gate levels), subject to timing constraints**
2. **Map factored form onto library of gates**
3. **Minimize number of literals (correlates with number of wires)**

Factored Form:

sum of products of sum of products . . .



X = (A B + B' C) (C + D (E + A C')) + (D + E)(F G)

Multi-Level Logic: CAD Tools for Simplification

Operations on Factored Forms:

- **Decompostion**
- **Extraction**
- **Factoring**
- **Substitution**
- **Collapsing**

Manipulate network by interactively issuing the appropriate instructions

There exists no algorithm that guarantees "optimal" multi-level network will be obtained

Multi-Level Logic: CAD Tools for Simplification

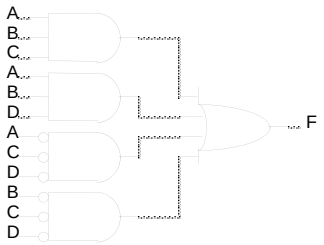
Decomposition:

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

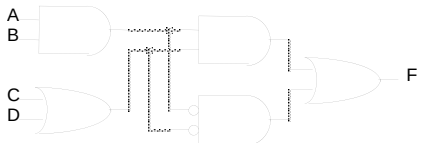
F = A B C + A B D + A' C' D' + B' C' D' (12 literals)

F rewritten as:

F = X Y + X' Y' **(4 literals)**
X = A B
Y = C + D



Before Decomposition



After Decomposition

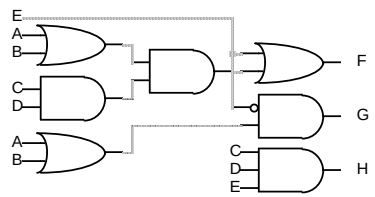
Multi-Level Logic: CAD Tools for Simplification

Extraction: **common intermediate subfunctions are factored out**

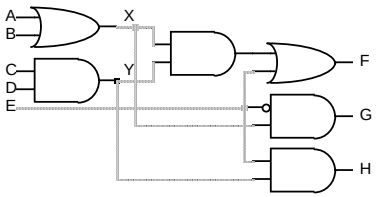
F = (A + B) C D + E **(11 literals)**
G = (A + B) E'
H = C D E

can be re-written as:

F = X Y + E **(7 literals)**
G = X E'
H = Y E
X = A + B **"Kernels": primary divisors**
Y = C D



Before Extraction



After Extraction

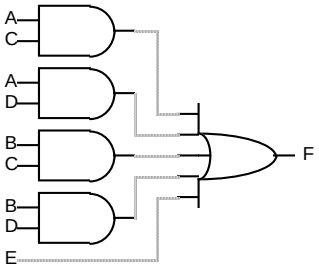
Multi-Level Logic: CAD Tools for Simplification

Factoring: **expression in two level form re-expressed in multi-level form**

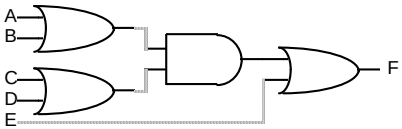
F = A C + A D + B C + B D + E (9 literals)

can be rewritten as:

F = (A + B) (C + D) + E (5 literals)



Before Factoring



After Factoring

Multi-Level Logic: CAD Tools for Simplification

Substitution: **function G into function F, express F in terms of G**

F = A + B C (5 literals)
G = A + B

F rewritten in terms of G:

F = G (A + C) (2 literals)

Collapsing: **reverse of substitution; use to eliminate levels to meet timing constraints**

F = G (A + C)
= (A + B) (A + C)
= A A + A C + A B + B C
= A + B C ¦

Multi-Level Logic: CAD Tools for Simplification

Key to implementing these operations: "division" over Boolean functions

F = P Q + R

divisor — quotient — remainder

example:

X = A C + A D + B C + B D + E
Y = A + B

X "divided" by Y is—

X = Y (C + D) + E

Complexity: finding suitable divisors

F = A D + B C D + E
G = A + B

G does not divide F under algebraic division rules
G does divide F under Boolean rules (very large number of these!)

F/G = (A + C) D

F written as G Q + R

F = [G (A + C) D] + E
= (A + B) (A + C) D + E
= (A A + A C + A B + B C) D + E
= (A + B C) D + E
= A D + B C D + E ¦

Multi-Level Logic: CAD Tools for Simplification

misII Session with the Full Adder

```
% misII
UC Berkeley, MIS Release #2.1 (compiled 3-Mar-89 at 5:32 PM)
misII> re full.adder
misII> p
    {co} = a b ci + a b ci' + a b' ci + a' b ci
    {sum} = a b ci + a b' ci' + a' b ci' + a' b' ci
misII> pf
   {co} = a b' ci + b (ci (a' + a) + a ci')
   {sum} = ci (a' b' + a b) + ci' (a b' + a' b)
misII> sim1 *
misII> p
    {co} = a b + a ci + b ci
    {sum} = a b ci + a b' ci' + a' b ci' + a' b' ci
misII> pf
   {co} = ci (b + a) + a b
   {sum} = ci (a' b' + a b) + ci' (a b' + a' b)
misII> gd *
misII> pf
   {co} = a [2] + b ci
   {sum} = a' [3]' + a [3]
   [2] = ci + b
   [3] = b' ci' + b ci
```

read eqntott equations

two level minimization

good decomposition

technology independent up to this point

Multi-Level Logic: CAD Tools for Simplification

```
misII> rlib msu.genlib
misII> map
misII> pf
   [361] = b' ci' + a'
   [328] = b'
   [329] = ci'
   {co} = [328]' [329]' + [361]'
   [3] = b ci' + b' ci
   {sum} = [3] a' + [3]' a
misII> pg
[361]     1890:physical 32.00
[328]     1310:physical 16.00
[329]     1310:physical 16.00
{co}      1890:physical 32.00
[3]       2310:physical 40.00
{sum}     2310:physical 40.00
misII> pat
 ... using library delay model
{sum}     : arrival=( 2.2  2.2)
{co}      : arrival=( 2.2  2.2)
[328]     : arrival=( 1.2  1.2)
[361]     : arrival=( 1.2  1.2)
[329]     : arrival=( 1.2  1.2)
[3]       : arrival=( 1.2  1.2)
ci        : arrival=( 0.0  0.0)
b         : arrival=( 0.0  0.0)
a         : arrival=( 0.0  0.0)
misII> quit
%
```

read library & perform technology mapping

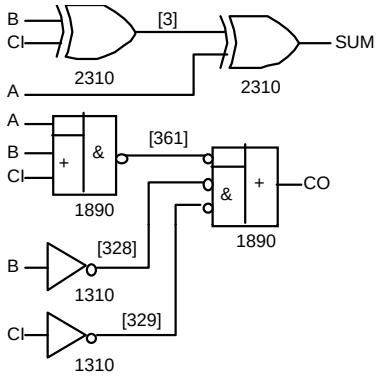
gates that implement the various nodes and their relative areas

timing simulation
unit delay plus 0.2 time units per fan-out

Multi-Level Logic: CAD Tools for Simplification

misII and the MSU gate library

VLSI Standard Cells



NOTE: OR-AND-INVERT equivalent to INVERT-AND-OR

Number	Name	Function
1310	inv	A'
1120	nor2	(A+B)'
1130	nor3	(A+B+C)'
1140	nor4	(A+B+C+D)'
1220	nand2	(A•B)'
1230	nand3	(A•B•C)'
1240	nand4	(A•B•C•D)'
1660	and2/nand2	[A•B, (A•B)']
1670	and3/nand3	[A•B•C, (A•B•C)']
1680	and4/nand4	[A•B•C•D, (A•B•C•D)']
1760	or2/nor2	[A+B, (A+B)']
1770	or3/nor3	[A+B+C, (A+B+C)']
1780	or4	(A+B+C+D)
1870	aoi22	(A•B + C•D)'
1880	aoi21	(A + B•C)'
1860	oai22	[(A + B)(C + D)]'
1890	oai21	[A (B + C)]'
1970	ao22	A•B + D•E
1810	ao222	A•B + C•D + E•F
1910	ao2222	A•B + C•D + E•F + G•H
1930	ao33	A•B•C + D•E•F
2310	xor2	A•B' + A'•B
2350	xnor2	A•B + A'•B'

Multi-Level Logic: CAD Tools for Simplification

More Examples

mis with standard simplification script:

```
misII -f script -t pla <espresso truth table file>
```

Full Adder:

```
.model full.adder
.inputs a b ci
.outputs sum co
.names a b ci co sum
1--0 1
-1-0 1
--10 1
111- 1
.names a b ci co
11- 1
1-1 1
-11 1
.end
```

mis pla style outputs

input variables

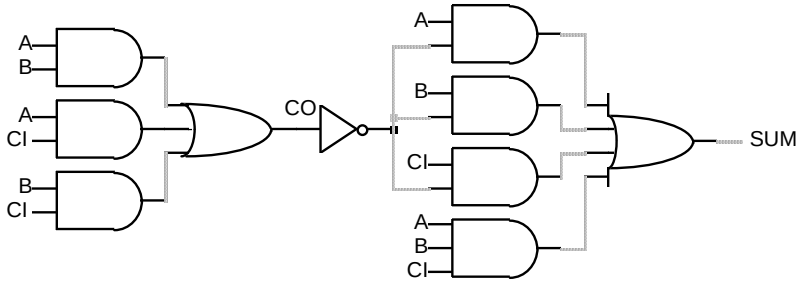
output variable

SUM = A CO' + B CO' + CI CO' + A B CI (9 literals)

CO = A B + A CI + B CI (6 literals)

Note that A xor B xor CI = A' B' CI + A B' CI' + A' B CI' + A B CI (12 literals!)

Multi-Level Logic: CAD Tools for Simplification



Multilevel Implementation of Full Adder: 5 Logic Levels!

Multi-Level Logic: Tools for Simplication

Two-bit Adder

```
.inputs a b c d
.outputs x y z
.names a c z [22] x
---1 1
11-- 1
-10- 1
.names a b c d x z [22] y
1---0-- 1
--1---1 1
-11-0-- 1
--110-- 1
---100- 1
.names a b c d z
-0-1 1
-1-0 1
0-10 1
.names a d z [22]
110 1
.end
```

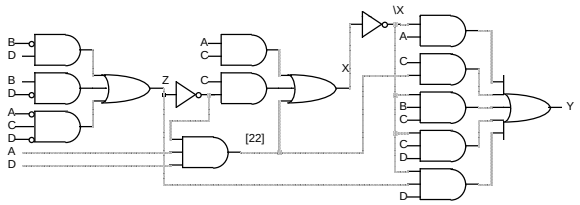
Mis Output

Z = B' D + B D' + A' C D'

[22] = A D Z'

X = [22] + A C + C Z'

Y = A X + C [22] + B C X' + C D X' + D X' Z'



8 logic levels!

Multi-Level Logic: CAD Tools for Simplication

BCD Increment By 1

```
.model bcd.increment
.inputs a b c d
.outputs w x y z
.names a b c d z w
1---1 1
0111- 1
.names a b c w z x
01-0- 1
0-100 1
.names a c z y
-11 1
000 1
.names a b c d z
0--0 1
-000 1
.end
```

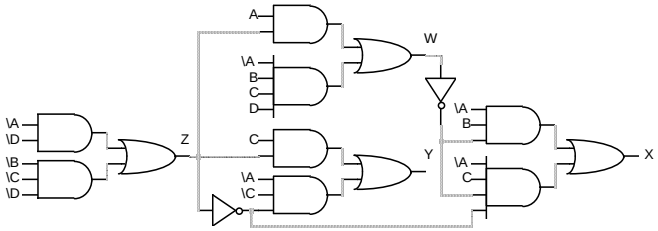
Mis Output

Z = A' D' + B' C' D'

Y = C Z + A' C' Z'

W = A Z + A' B C D

X = A' B W' + A' C W' Z'



Time Response in Combinational Networks

- **emphasis on timing behavior of circuits**
- **waveforms to visualize what is happening**
- **simulation to create these waveforms**
- **momentary change of signals at the outputs: *hazards***
  can be useful— pulse shaping circuits
  can be a problem — *glitches:* incorrect circuit operation

Terms:

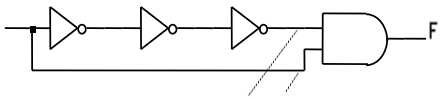
gate delay— time for change at input to cause change at output
minimum delay vs. typical/nominal delay vs. maximum delay
careful designers design for the worst case!

rise time— time for output to transition from low to high voltage

fall time— time for output to transition from high to low voltage

Time Response in Combinational Networks

Pulse Shaping Circuit



A' • A = 0

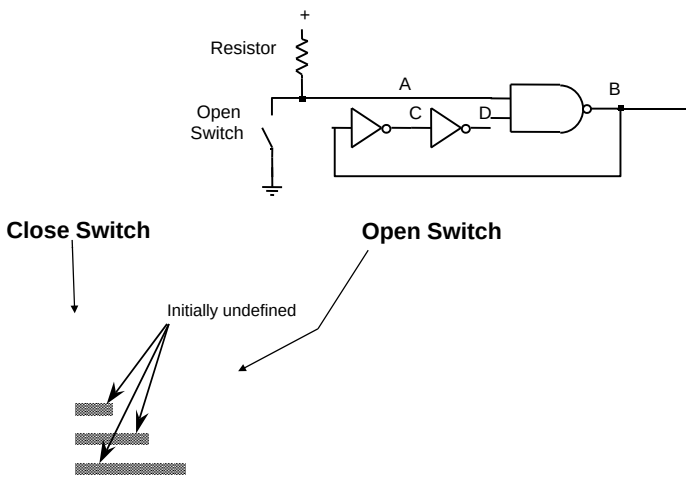
3 gate delays

D remains high for three gate delays after A changes from low to high

F is not always 0!

Time Response in Combinational Networks

Another Pulse Shaping Circuit



Time Response in Combinational Networks

Hazards/Glitches and How to Avoid Them

Unwanting switching at the outputs

Occur because delay paths through the circuit experience different propagation delays

Danger if logic "makes a decision" while output is unstable OR hazard output controls an *asynchronous* input (these respond immediately to changes rather than waiting for a synchronizing signal called a *clock*)

Usual solutions:
wait until signals are stable (by using a clock)

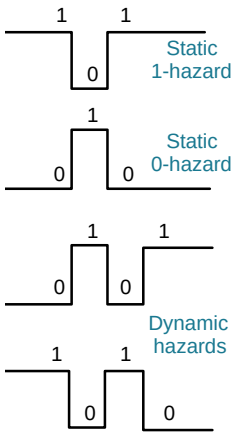
never, never, never use circuits with asynchronous inputs

design hazard-free circuits

Suggest that first two approaches be used, but we'll tell you about hazard-free design anyway!

Time Response in Combinational Networks

Hazards/Glitches and How to Avoid Them



Input change causes output to go from 1 to 0 to 1

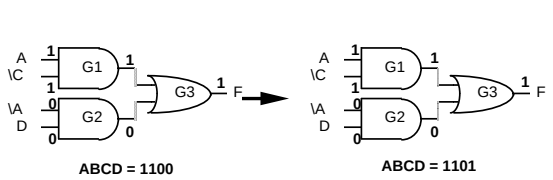
Input change causes output to go from 0 to 1 to 0

Input change causes a double change from 0 to 1 to 0 to 1 OR from 1 to 0 to 1 to 0

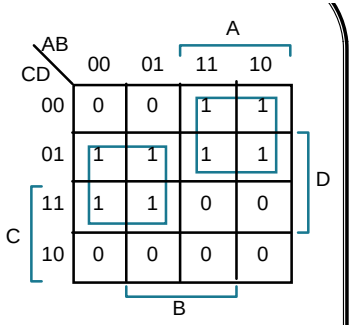
Kinds of Hazards

Time Response in Combinational Circuits

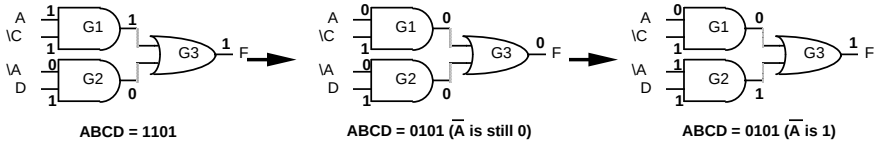
Glitch Example



input change within product term



F = A' D + A C'



input change that spans product terms output changes from 1 to 0 to 1

Time Response in Combinational Networks

Glitch Example

General Strategy: add redundant terms

F = A' D + A C' becomes A' D + A C' + C' D

This eliminates 1-hazard? How about 0-hazard?

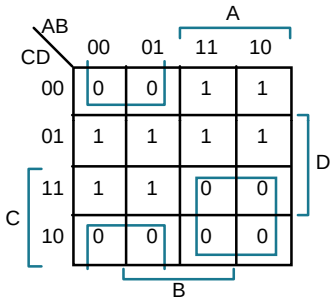
Re-express F in PoS form:

F = (A' + C')(A + D)

Glitch present!

Add term: (C' + D)

This expression is equivalent to the hazard-free SoP form of F



Time Response in Combinational Networks

Glitch Example

Start with expression that is free of static 1-hazards

F = A C' + A' D + C' D

Work with complement:

F' = (A C' + A' D + C' D)'

= (A' + D) (A + D') (C + D')

= A C + A C D' + C D' + A' C D' + A' D'

= A C + C D' + A' D'

covers all the adjacent 0's in the K-map

free of static-1 and static-0 hazards!

Time Response in Combinational Networks

Detecting Static Hazards in Multi-Level Circuits

Calculate transient output function

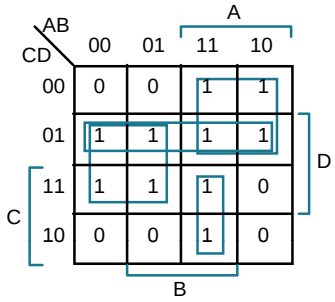
variables and complements are treated as independent variables

cannot use X + X' = 1 or X • X' = 0 for simplifications

Example:

F = A B C + (A + D) (A' + C')

F1 = A B C + A A' + A C' + A' D + C' D 2-level form



ABCD: 1111 to 1110, covered by term ABC, so no 1-hazard present

ABCD: 1110 to 1100, term ABC goes low while term AC' goes high

some static hazards are present!

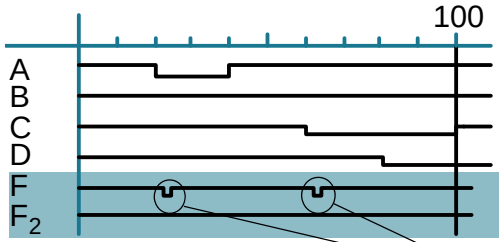
Time Response in Combinational Networks

Static 1-hazards

Solution:

Add redundant terms to insure all adjacent transitions are covered by terms

F2 = A C' + A' D + C' D + A B + B D



1's hazards in F corrected in F2

Time Response in Combinational Networks

Static 0-Hazards

Similar to previous case, but work with the complement of F

If terms of the transient output function cover all 0 transitions, then no 0-hazards are present

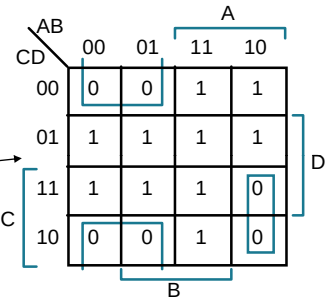
$\overline{F}$ = [A B C + (A + D) (A' + C')]'

= (A' + B' + C') (A' D' + A C)

= A' D' + A' B D' + A' C D' + A B' C

= A' D' + A B' C

+ B' C D'



F = (A + D) (A' + B + C') (B + C' + D)

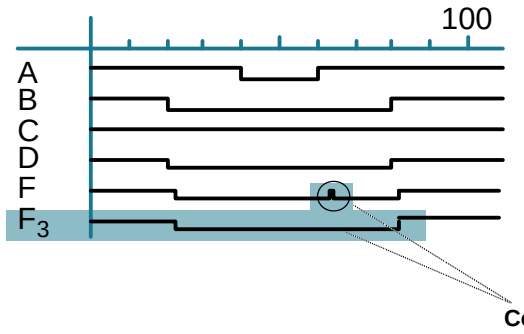
0-hazard free

equivalent to F2 on last slide

0-hazard on transition from 1010 to 0010

Time Response in Combinational Networks

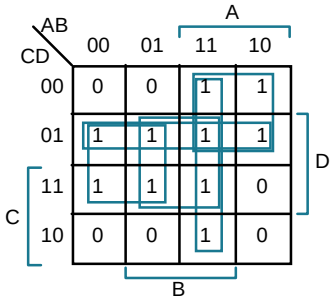
Static 0-Hazards



0-Hazard Corrected in F3

Time Response in Combinational Networks

Designing Networks for Hazard-free operation



Simply place transient output function in a form that guarantees that all adjacent ones are covered by a term

no term of the transient output function contains both a variable and its complement

F(A,B,C,D) = m(1,3,5,7,8,9,12,13,14,15)

F = A B + A' D + B D + A C' + C' D

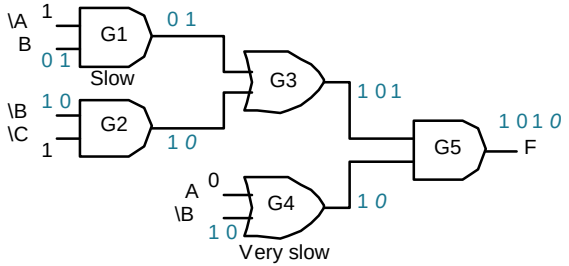
= (A' + B + C') D + A (B + C')

(factored by distributive law, which does not introduce hazards since it does not depend on the complementarity laws for its validity)

Time Response in Combinational Networks

Dynamic Hazards

Example with Dynamic Hazard



Three different paths from B or B' to output

ABC = 000, F = 1 to ABC = 010, F = 0

different delays along the paths:
G1 slow, G4 very slow

Handling dynamic hazards very complex

Beyond our scope

Chapter Review

- Transition from Simple Gates to more complex gate building blocks
- Conversion from AND/OR, OR/AND to NAND/NAND, NOR/NOR
- Multi-Level Logic: Reduced gate count, fan-ins, but increased delay
- Use of misII to optimize multi-level logic and to perform mappings
- Time Response in Combinational Logic:
  - Gate Delay, Rise Time, Fall Time
  - Hazards and Hazard-free Design