Time response in combinational networks

- Waveforms to visualize what is happening

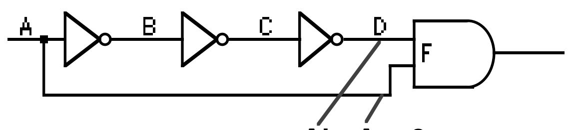
- Simulation to create these waveforms

- Momentary change of signals at the outputs are of interest: hazards
  - can be useful — pulse shaping circuits
  - can be a problem — incorrect circuit operation (glitches)

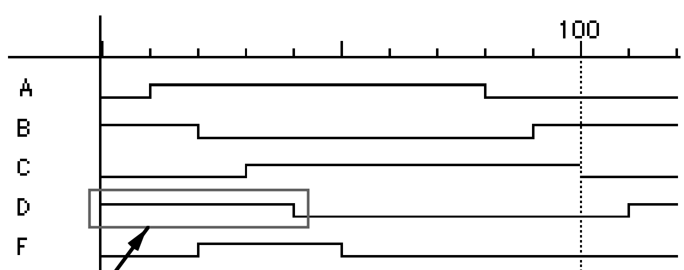
- Terms:
  - gate delay — time for change at input to cause change at output
    - min delay vs. typical/nominal delay vs. max delay
    - careful designers design for the worst case!
  - rise time — time for output to transition from low to high voltage
  - fall time — time for output to transition from high to low voltage

Time response in combinational networks

- Pulse shaping circuit



A' • A = 0



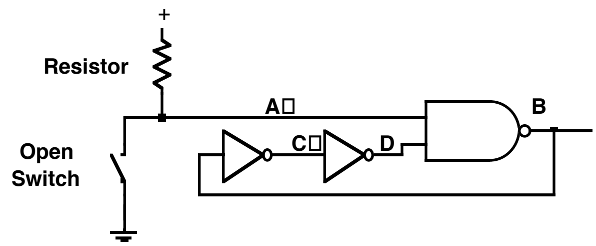
3 gate delays

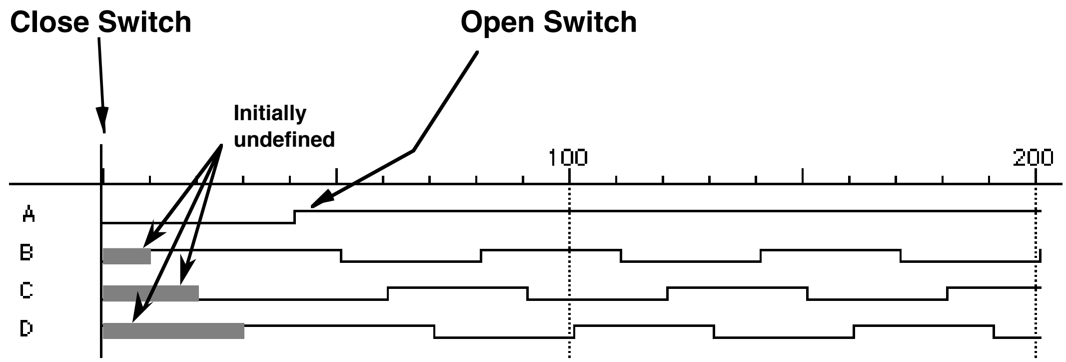
D remains high for
three gate delays after
A changes from low to high

F is not always 0!

Time response in combinational networks

- Another pulse shaping circuit





Time response in combinational networks

- Hazards/glitches: unwanted switching at the outputs

  - occur because different paths through the circuit
    - have different propagation delays

  - danger arises 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)

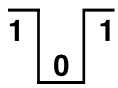
- Usual solutions

  - 1) wait until signals are stable (by using a clock)
  - 2) design hazard-free circuits

  - the first approach is preferable (easiest to design), however, we'll
  - talk about hazard-free design anyway because there are times when
  - we must consider asynchronous inputs to our designs

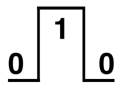
Time response in combinational networks

- Kinds of hazards



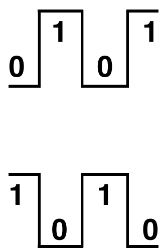
static
1-hazard

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



static
0-hazard

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



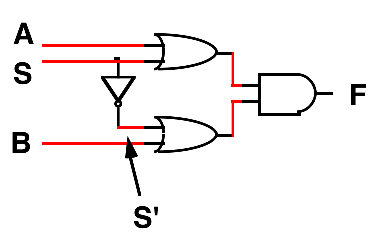
dynamic
hazards

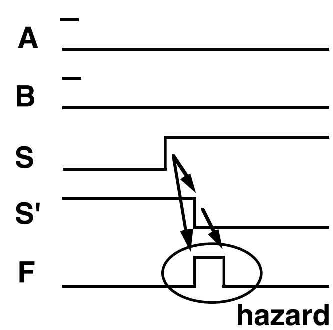
input change causes
a double change
from 0 to 1 to 0 to 1 OR
from 1 to 0 to 1 to 0

Time response in combinational networks

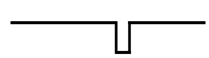
- Static hazards: due to a literal and its complement momentarily taking on the same value (through different paths with different delays and reconverging)

may cause and output that should have stayed the same to momentarily take on the wrong value





static-0 hazard



static-1 hazard

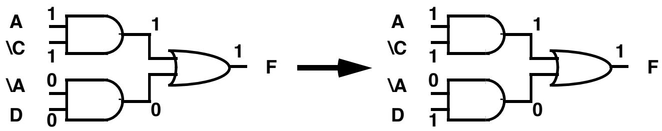
Time response in combinational networks

- Glitch example

F = A'D + AC'

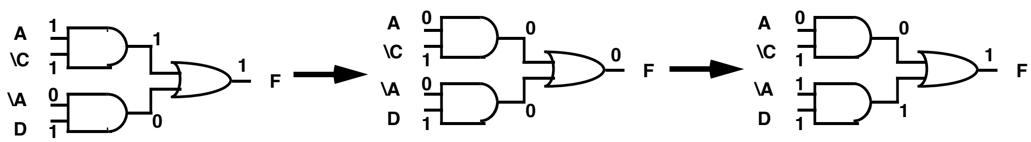
0	0	1	1
1	1	1	1
1	1	0	0
0	0	0	0

A, B, C, D



ABCD = 1100 ABCD = 1101

input change within product term
output will remain a constant one



ABCD = 1101 ABCD = 0101 (A' is still 0) ABCD = 0101 (A' is finally 1)

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

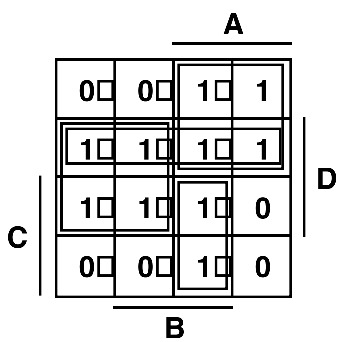
Time response in combinational networks

- Glitch example continued
  - general Strategy: add redundant terms
    - F = A' D + A C' becomes A' D + A C' + C' D
  - so that all constant-one single input changes are covered by a cube.
- Expressions in sum of products form cannot have static 0-hazards.
  (Ignore what Katz says on page 143.)
  You don't need to do anything special to eliminate them.

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

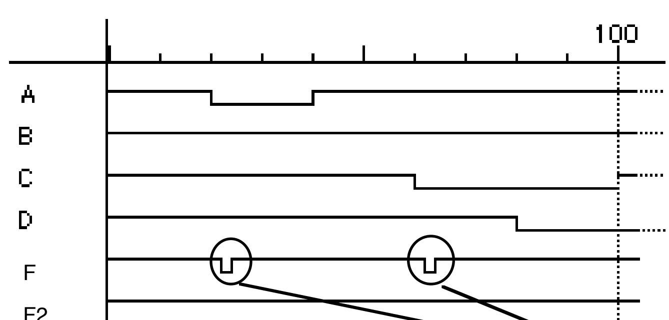
ABCD: 1110 to 1100, term ABC goes low while term AC' goes high

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 + $\boxed{\textbf{A B + B D}}$



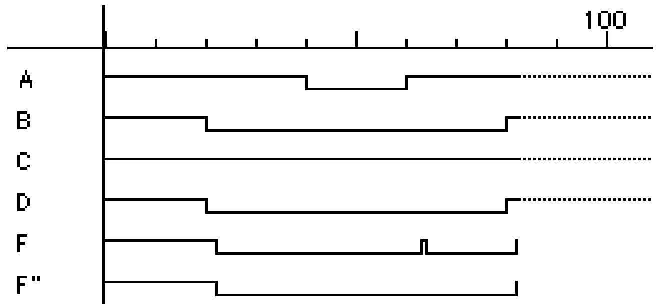
1-hazards in F corrected in F2

Time response in combinational networks

- Static 0-hazards are possible if the transient output function includes a term containing both a variable and its complement

F = A B C + (A + D) (A' + C')
F1 = A B C + $\boxed{\text{A A'}}$ + A C' + A' D + C' D 2-level form

F = (A' + B + C') D + A (B + C') Static hazard-free
= A' D + B D + C' D + A B + A C'



Time response in combinational networks

- Designing networks for hazard-free operation

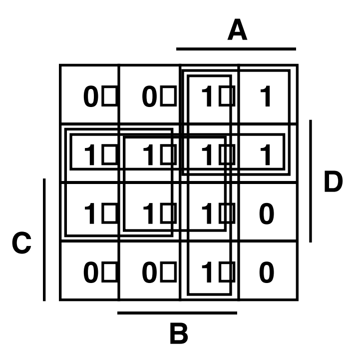
simply place transient output function in a form that
guarantees that all adjacent 1s are covered by a term

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

F(A,B,C,D) = $\sum$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')

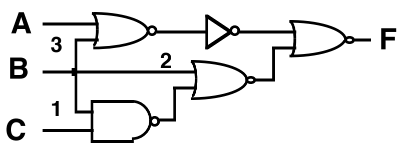
(factored by distributive law, which does
not introduce hazards since it does not
depend on the complementarity laws
for its validity)

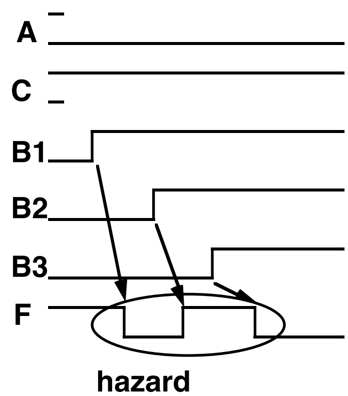


Time response in combinational networks

- Dynamic hazards: due to the same versions of a literal taking on opposite values (through different paths with different delays and reconverging)

may cause an output to change 3 times instead of once





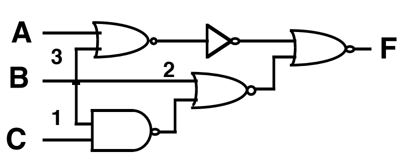
dynamic hazards

Time response in combinational networks

- Dynamic hazard example (cont'd):

$F = [(A + B_3) + (B_2 + (B_1C)')']' = A'B_3'B_2 + A'B_3'B_1' + A'B_3'C'$

to detect: don't use <u>complementarity</u> or <u>idempotency</u> axioms of Boolean algebra



set A=0, C=1, and change B from 0 to 1

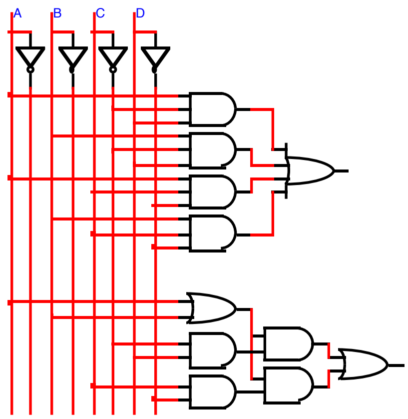
$F(A=0, C=1) = B_3'B_2 + B_3'B_1'$

F is initially 1 ($B_3'B_1' = 1$)
then B_1 changes and F = 0
then B_2 changes and F = 1 ($B_3'B_2 = 1$)
then B_3 changes and F = 0

handling dynamic hazards is very complex in general and is beyond the scope of this course

Multi-level logic: CAD tools for simplification

- Exploit common subexpressions (less gates)
- Minimize number of literals rather than terms
- Trade more levels of logic for reduced fan-in (may also be faster)
- No systematic minimizaton procedure exists as in the two-level case



X = AC'D + BC'D + ACD' + BCD'
(12 literals and 4 wires
max fan-in = 4)

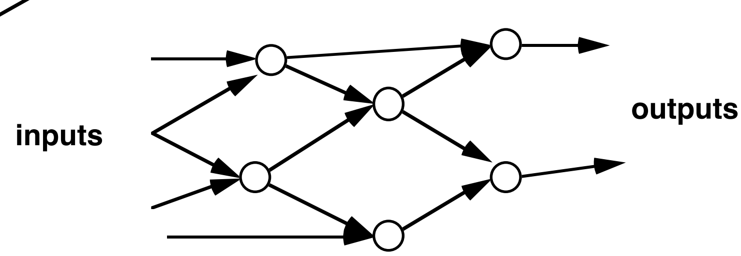
X = (A+B)C'D + (A+B)CD'
(8 literals and 6 wires
max fan-in = 2)

Multi-level logic: CAD tools for simplification

- Operations on factored forms
  - factoring
  - decomposition
  - extraction
  - substitution
  - collapsing

manipulate network by interactively issuing the appropriate instructions

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



each node is an arbitrarily complex gate

Multi-level logic: CAD tools for simplification

- Factorization: "division" of Boolean functions

  F = DQ + R   D = divisor
  Q = quotient
  R = remainder

- Example:

  X = ac + ad + bc + bd + e
  Y = a + b

  X/Y = c + d   X = Y (c + d) + e

  divisor   quotient   remainder

interesting divisors are called kernels and cubes

Multi-level logic: CAD tools for simplification

- Algebraic division – use rules of polynomial division (previous example)

- Boolean division – use rules of Boolean algebra

F = ad + bcd + e
G = a + b

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

F/G = (a + c) d

F = GQ + R = [G (a + c) d] + e
(a + b) (a + c) d + e
(aa + ac + ab + bc) d + e
(a + bc) d + e
ad + bcd + e

the key here is the absorption theorem of Boolean algebra

Multi-level logic: CAD tools for simplification

- Factoring — expression in two-level form is re-expressed in multi-level form

  F = a c + a d + b c + b d + e (9 literals)

  can be re-written as:

  F = (a + b) (c + d) + e (5 literals)

- Usually we only perform algebraic factorization

  — Boolean factorization is still too computationally expensive

Multi-level logic: CAD tools for simplification

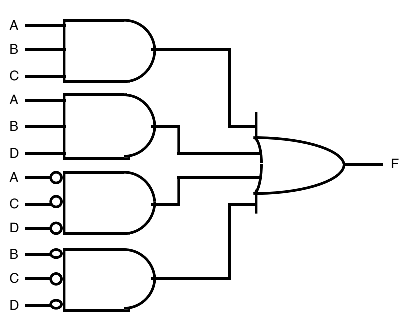
- Decomposition — take a single factored Boolean expression and replace it with a collection of new expressions and update circuit

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

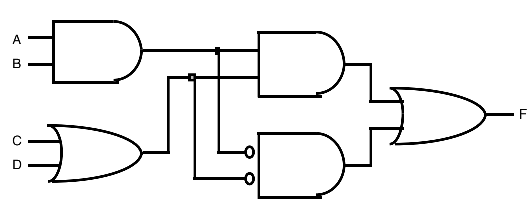
  F rewritten as: F = X Y + X' Y'

  X = A B

  Y = C + D (8 literals)



before decomposition



after decomposition

Multi-level logic: CAD tools for simplification

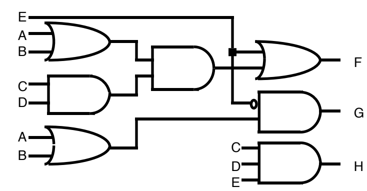
- Extraction — look at two expressions and factor out common intermediate subfunctions and reflect their sharing in the circuit

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

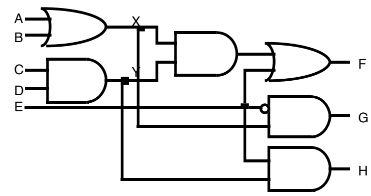
can be re-written as:

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

before extraction



after extraction



Multi-level logic: CAD tools for simplification

- Substitution — an expression is substituted into another thus changing the circuit topology

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

F rewritten in terms of G:

F = G (A + D) (3 literals)

- Collapsing — reverse of substitution it is used to eliminate levels so as to meet timing constraints

F = G (A + D)
= (A + B + C) (A + D)
= A A + A C + A B + A D + B D + C D
= A + B D + C D √

Multi-level logic: CAD tools for simplification

- Other network operations —

  Simplification — two-level simplification of a node (espresso algorithm)

  Phase-assignment — decide on cheaper implementation of a node (complement the function or not)

  Sweep — remove all inverters and buffers (nodes where output = input can occur as a result of simplification)

  Eliminate — eliminate nodes of too low a cost by collapsing basically, the number of literals that will be saved by removing the node from the graph (high fanout nodes are usually not eliminated because they are heavily re-used)

  And many manipulations . . .

Multi-level logic: CAD tools for simplification

- Minimization procedures
  - heuristic application of the operations we just listed
  - no guarantee of finding an optimal realization
  - does quite well in a practical amount of time
- Everything up to this point has been technology independent
  - just considering literal count
  - not the types of gates available to actually implement the circuit
- Technology mapping
  - process of converting circuit graph into one where each node
  - is directly implementable with an available gate

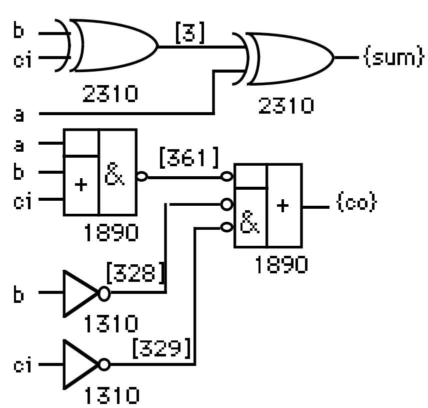
Multi-level logic: CAD tools for simplification

- Implementation library – catalog of available components
- Try to find minimum cost cover of the circuit graph
- Common methods make use of modern compiler technology

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

- A mapped circuit example: full adder



note: OAI equivalent to IAO using de Morgan's law

Multi-level logic: CAD tools for simplification

- Example session with MIS (a multi-level optimization tool)

```
% 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 equations

two level minimization

good decomposition

Multi-level logic: CAD tools for simplification

- Example session with MIS continued

```
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

- **Multi-level circuit before mapping (14 literals)**

```
{co} = a [2] + b ci
{sum} = a' [3]' + a [3]
[2] = ci + b
[3] = b' ci' + b ci
```

- **Multi-level circuit after mapping (16 literals)**

```
[361] = b' ci' + a'
[328] = b'
[329] = ci'
{co} = [328]' [329]' + [361]'
[3] = b ci' + b' ci
{sum} = [3] a' + [3]' a
```


Multi-level logic: CAD tools for simplification

- **Two-level circuit before mapping (26 literals)**

```
[437] = a b' ci'
[438] = a' b ci'
[439] = a' b' ci
[440] = a b ci
[441] = b ci
[442] = a ci
[443] = a b
{co} = [440] + [441] + [442] + [443]
{sum} = [437] + [438] + [439] + [440]
```

- **Two-level circuit after mapping (29 literals)**

```
[499] = a'
[500] = b'
[501] = ci'
[437] = a' + [499]' + [501]'
[438] = [499]' + b' + [501]'
[439] = [499]' + [500]' + ci'
[440] = a' + b' + ci'
[441] = b' + ci'
[442] = a' + ci'
[443] = a' + b'
{co} = [440]' + [441]' + [442]' + [443]'
{sum} = [437]' + [438]' + [439]' + [440]'
```

Multi-level logic: CAD tools for simplification

- Standard script: seems to work most of the time

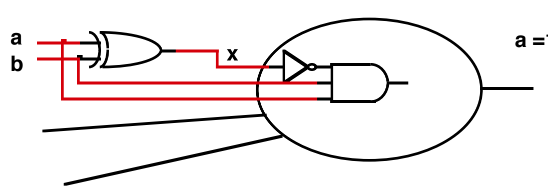
```
sweep                         flatten inverters and buffers
elim 5                        eliminate small (provide more factors for later)
sim1 *                        espresso on all nodes
gkx -abt 30 *
gcx -bt 30 *                  find largest divisors and resubstitute
resub -a *
gkx -abt 10 *
gcx -bt 10 *                  find more divisors and resubstitute
resub -a *
gkx -abt 0 *
gcx -bt 0 *                   consider all remaining divisors and resubstitute
resub -a *
sweep                         flatten any inverters and buffers generated
elim 0                        eliminate any obviously expensive nodes
decomp -g *                   reuse common subexpressions as much as possible

then perform technology mapping (map)
```


Multi-level logic: CAD tools for simplification

- Don't cares come from two sources in multi-level circuits

- From specification (external explicit don't cares)
  - in terms of circuit inputs and outputs

- From structure of circuit graph (internal implicit don't cares)
  - a combination of input and internal values cannot occur or
  - an internal node output is irrelevant for some input combinations
    - depending on how it is used by its fanout



a =1, b = 1, x =1 can never occur

- Both are critical in arriving at minimal circuits

- Must be maintained throughout all graph operations

Chapter review

- Conversion from AND/OR, OR/AND to NAND/NAND, NOR/NOR
- Transition from simple gates to more complex gate building blocks
- Multi-level logic: reduced gate count, fan-ins, potentially faster,
  □ □ □ □ more levels, harder to design
- Time response in combinational logic
- Hazards and hazard-free design
- Optmization and mapping tools for multi-level logic