An Introduction to Hazard-Free Logic Synthesis (Fundamental Mode)

Steven M. Nowick

Columbia University

(nowick@cs.columbia.edu)

July 15, 2002

Goal

Given: a Boolean function

Design: a *hazard-free* circuit implementation

Goal

Given: a Boolean function

Design: a *hazard-free* circuit implementation

Outline

Basics: Hazards

Part I. 2-Level Logic

Part II. Multi-Level Logic and Technology Mapping

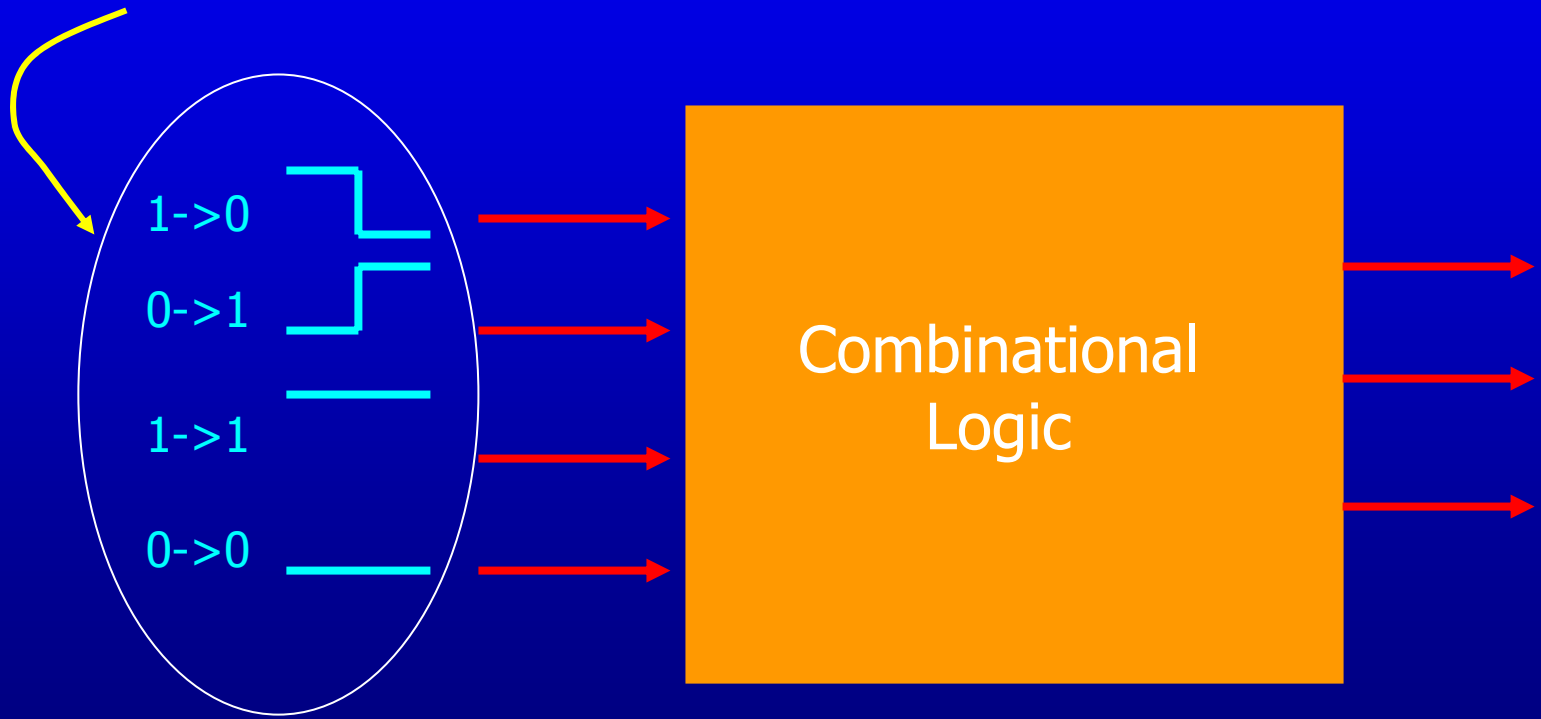
Conclusions

Basics

1) "Input Transition"

= "multiple-input transition", "multiple-input change" [*MIC*]

= a *change* from one input vector to another



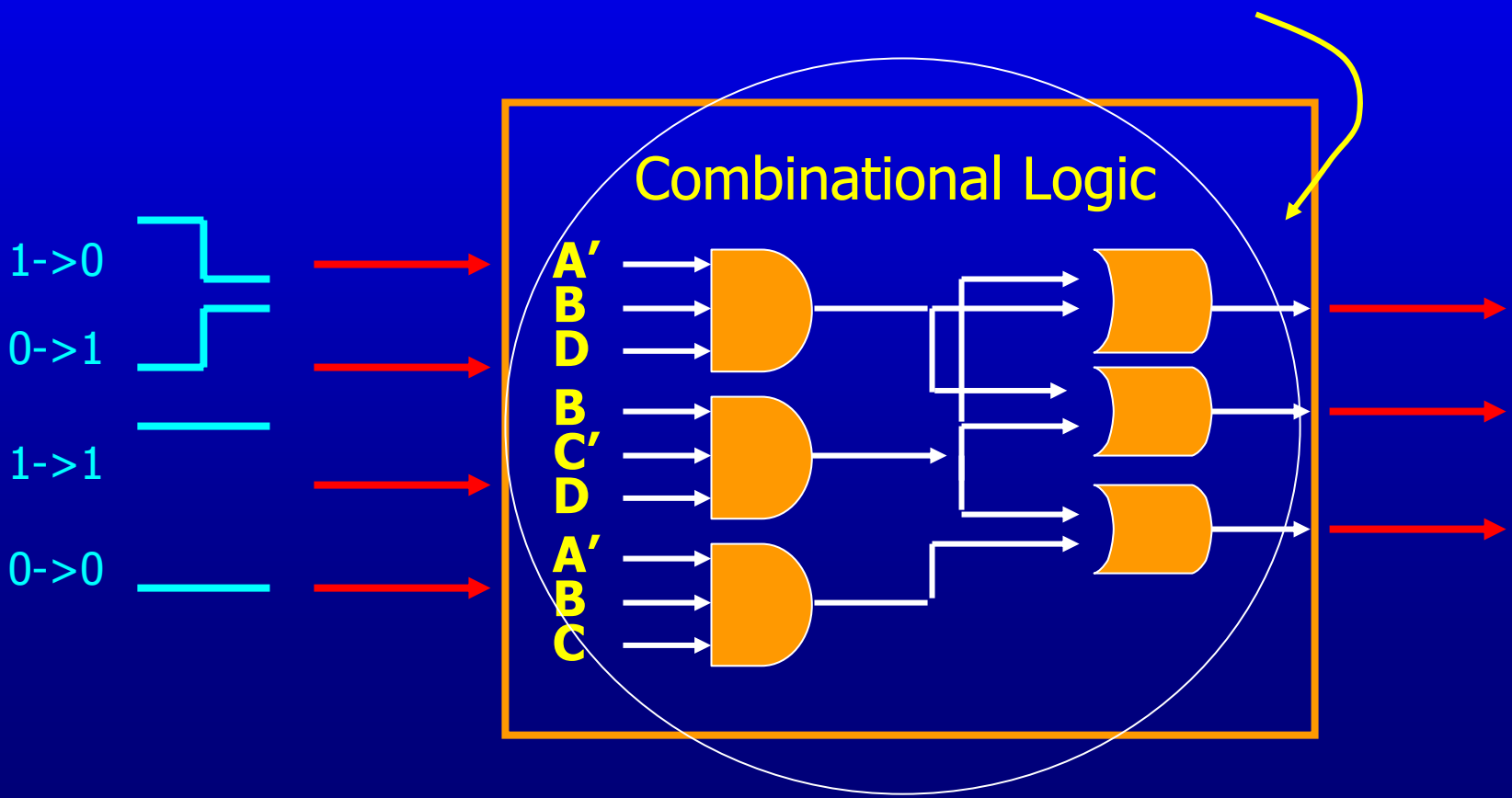
Assume: "*clean*" input transitions => no glitches!

Basics

2) Circuit Model

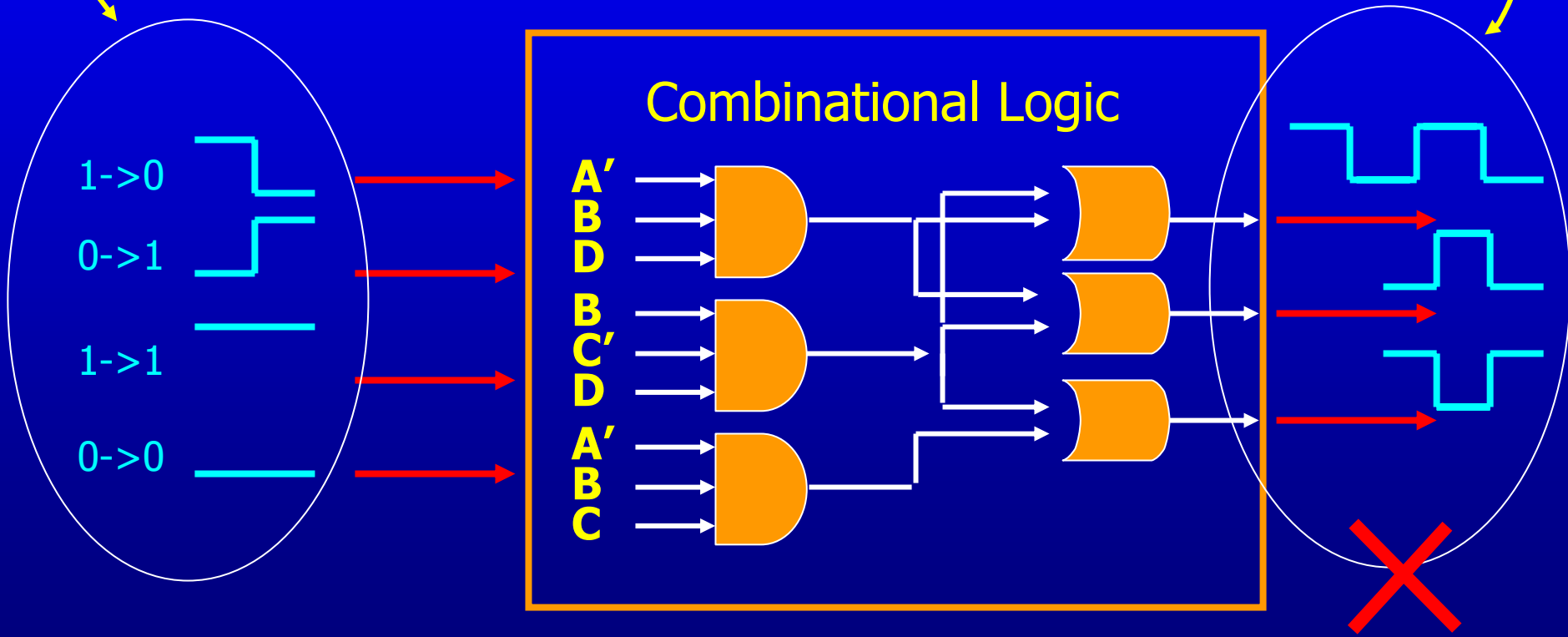
Assume an *"unbounded wire delay" model*

… gates and wires may have *arbitrary (finite) delays!*



The Goal

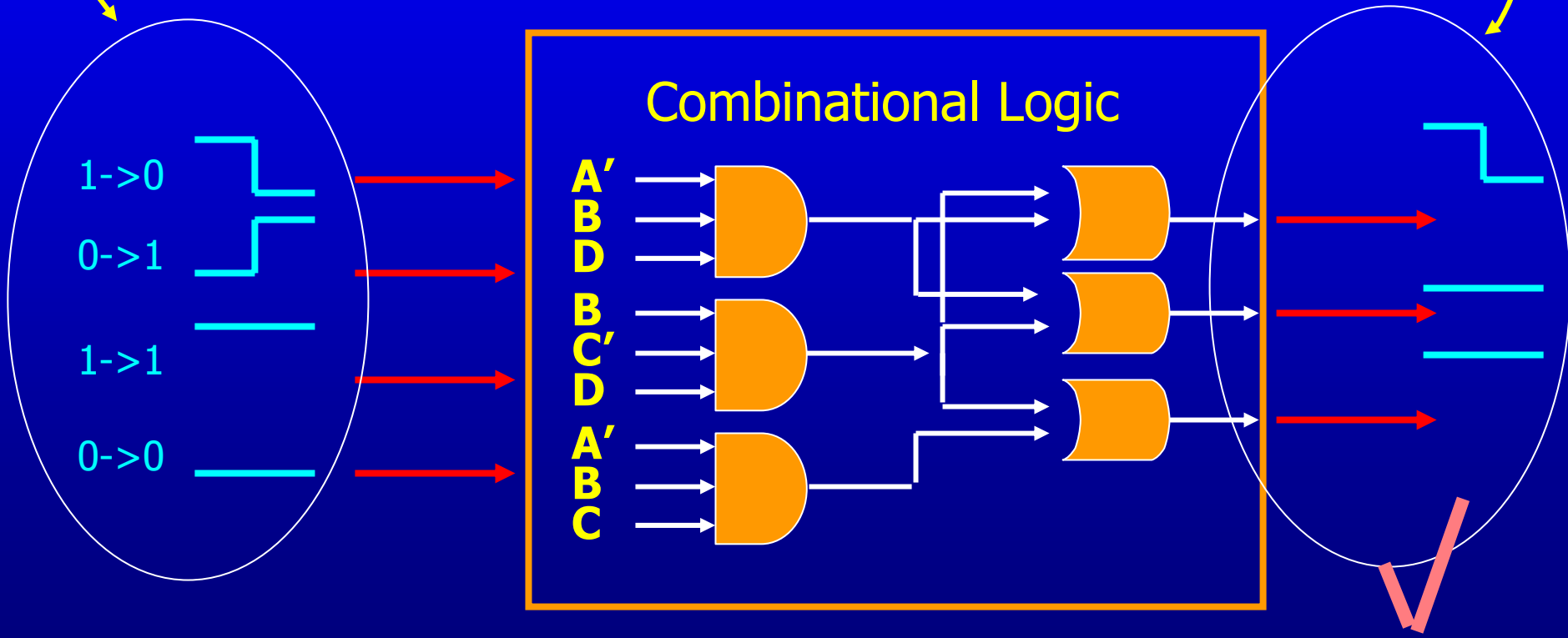
Given a *specified input transition,* synthesize a circuit impltn. with *no "combinational hazards"* for this transition (i.e. *no possible glitch on outputs*!)



... assuming above *circuit* & *environmental models*....

The Goal

Given a *specified input transition,* synthesize a circuit impltn. with *no "combinational hazard"* for this transition (i.e. *no possible glitch on outputs*!)



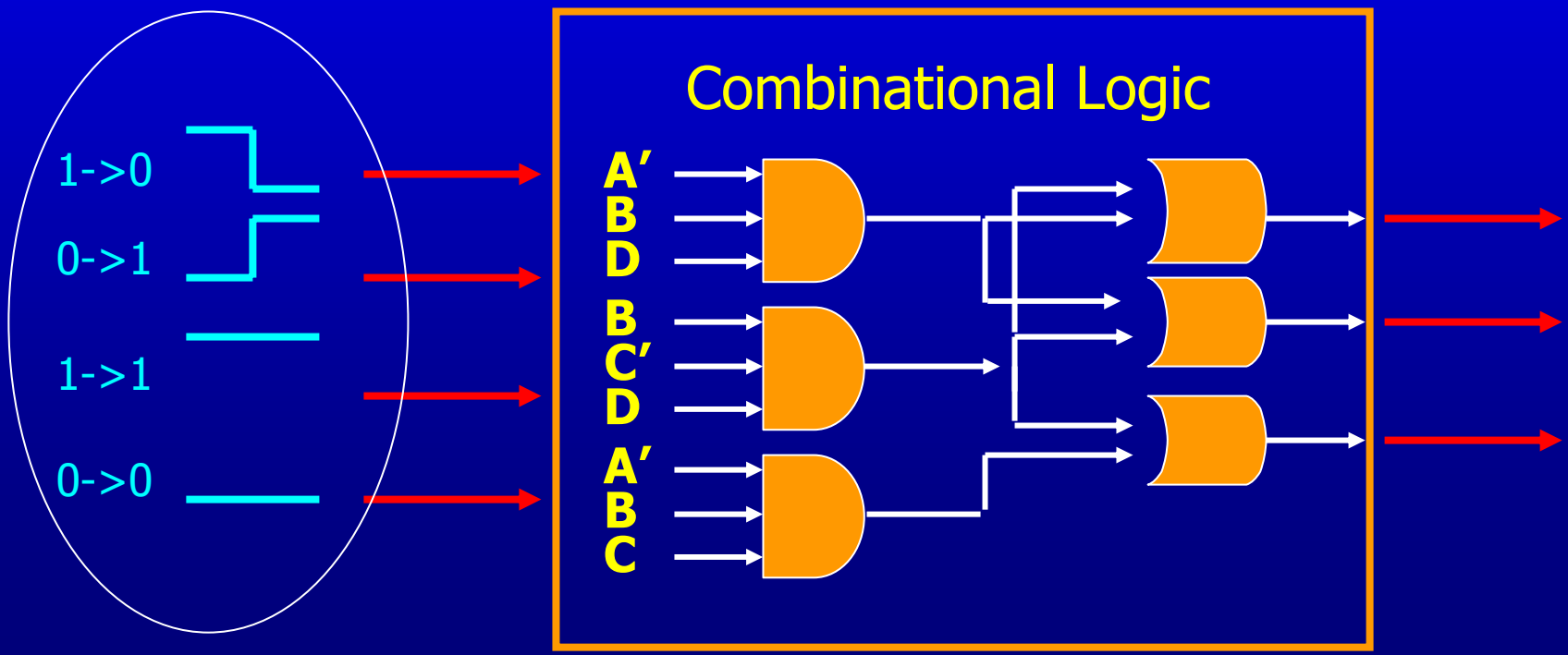
... assuming above *circuit* & *environmental models*....

Basics

3) Environmental Model

Assume *"generalized fundamental mode"*

… after an input transition, *no new inputs may arrive* until *the circuit has stabilized!*



Key Differences from "QDI" Hazard-Free Design

1. Combinational Circuit Model: *now more robust!*

   - *circuits correct for arbitrary gate + wire delays*

   - *… vs. QDI: uses "isochronic fork" assumption*

2. Environmental Model: "*generalized fundamental mode*

   - now, *timing assumptions on environment (1-sided*

   - *… vs. QDI: "input/output mode" (= none)*

Basics: Combinational Hazards

Two types of combinational hazards:

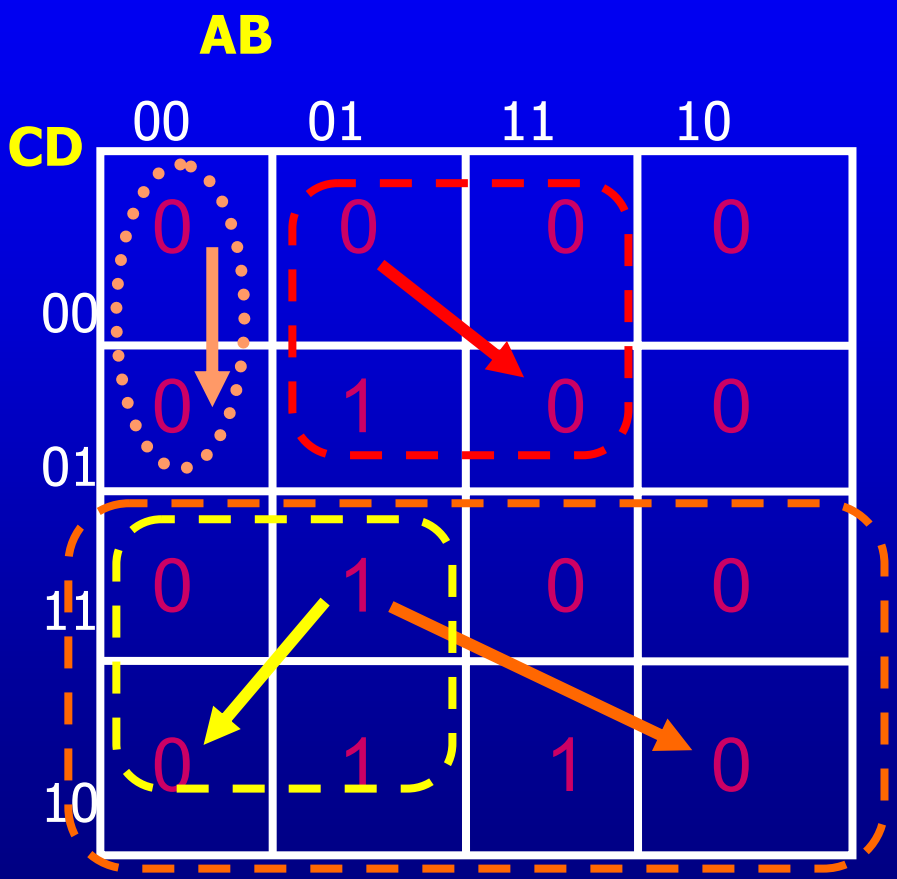
1. Function Hazard:

   - *inherent in combinational function*

2. Logic Hazard:

   - *inherent in circuit implementation*

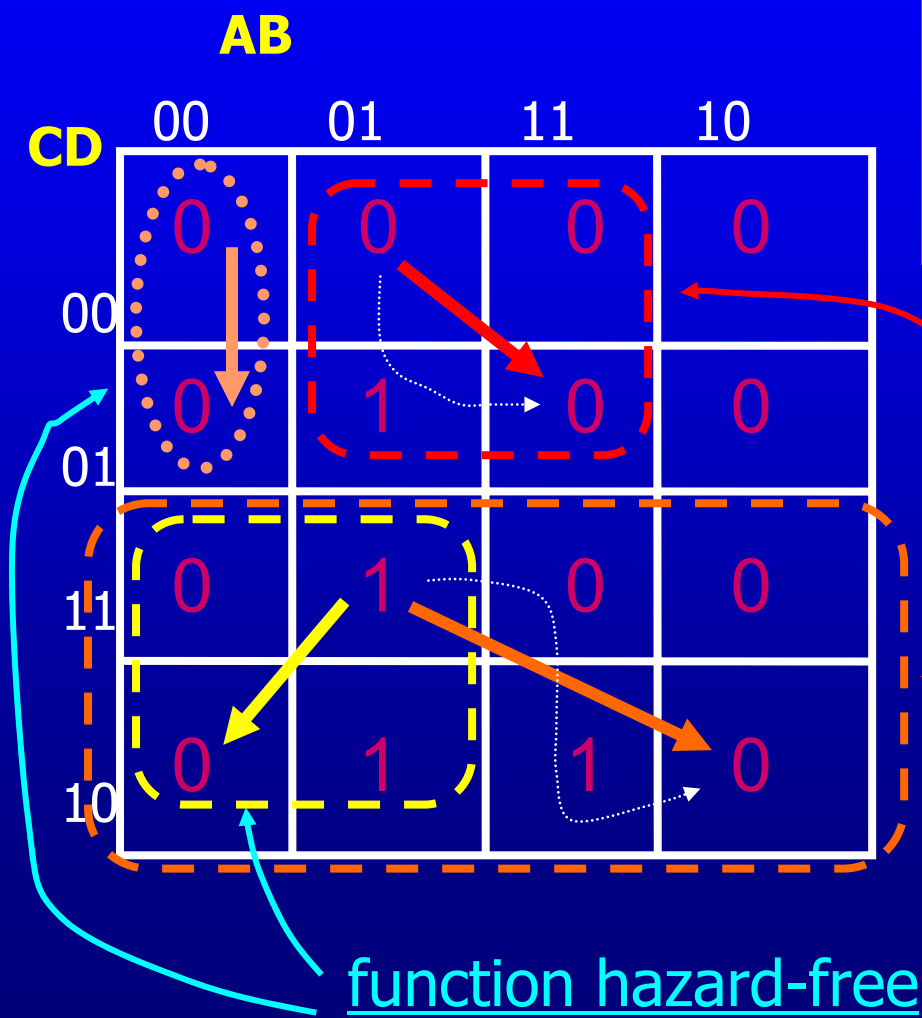
Function Hazards



Non-monotonic changes on output function during an input transition (i.e., >1 change on output!)

Function + 4 input transitions

Function Hazards



Non-monotonic changes on output function during an input transition (>1 change on output!)

Function Hazards: Summary

Function hazards: *cannot be removed*

- inherent in function itself
- cannot guarantee glitch-free logic implementation [Unger]

Therefore, *only consider function hazard-free transitions*:

- most "specified behaviors" = naturally monotonic (not glitchy)

Sequential synthesis methods:

- *must not introduce function hazards*

Burst-mode: uses ...

- constrained 'state minimization' + 'state assignment' steps
- always succeeds: no undesired function hazards introduced...

Logic Hazards

Now, assume *function hazard-free* input transitions....

Logic Hazard = property of a given *circuit implementation*

Def. Logic Hazard: Given combinational function f, circuit implementation C, and an input transition t.

If f is *function hazard-free* for input transition t,

but *implementation C may glitch* during transition t,

then *circuit C has a logic hazard* for transition t.

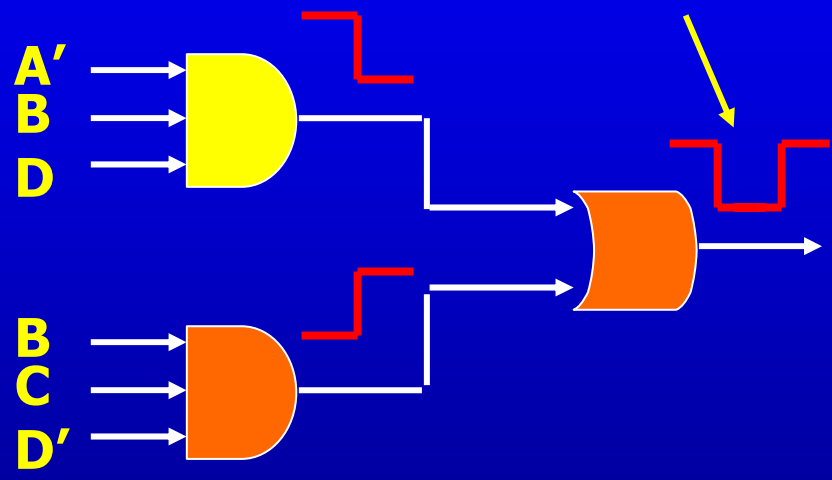
Otherwise, *circuit C is logic hazard-free* for transition t.

Logic Hazards

CD \ AB	00	01	11	10
00	0	0	0	0
01	0	1	0	0
11	0	1	0	0
10	0	1	–	0

Function hazard-free input transition

Logic hazard



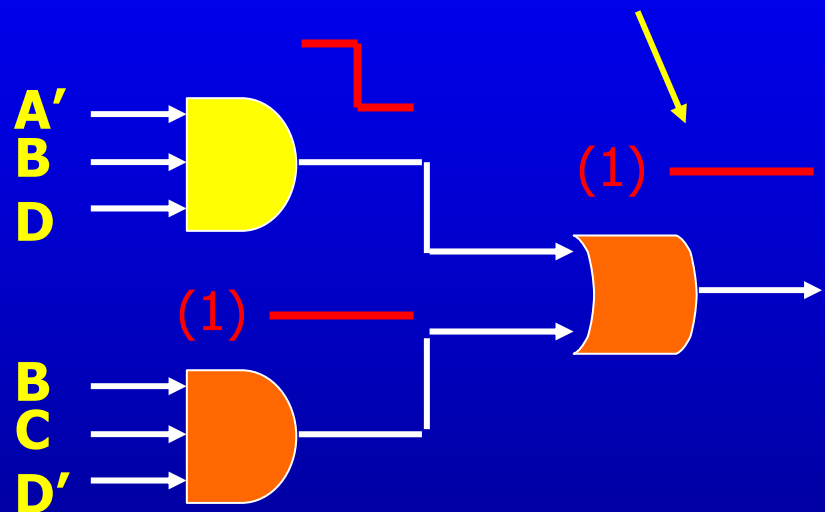
"input change": *D: 1-->0*
(ABC=011)

Logic Hazards

CD \ AB	00	01	11	10
00	0	0	0	0
01	0	1	0	0
11	0	1	0	0
10	0	1	_	0

A Different Implementation

No logic hazard



"input change": *D: 1-->0*
(ABC=011)

Part I

Two-Level Logic

Part I: Outline

- **Problem #1:** Eliminating Logic Hazards for *One* Input Transition
- **Problem #2:** Eliminating Logic Hazards for *Several* Input Transitions
- **2-Level Hazard-Free Logic Minimization:** *a Complete Example*
- **Existence of a Hazard-Free Solution**
- **An Alternative Approach:** *Using GC-Elements*

PROBLEM #1: Eliminating Logic Hazards for *One* Input Transition

Given: a combinational function f,

and a *function hazard-free* input transition t.

Goal: find a 2-level (AND-OR) implementation of f

which is *logic hazard-free* for input transition t.

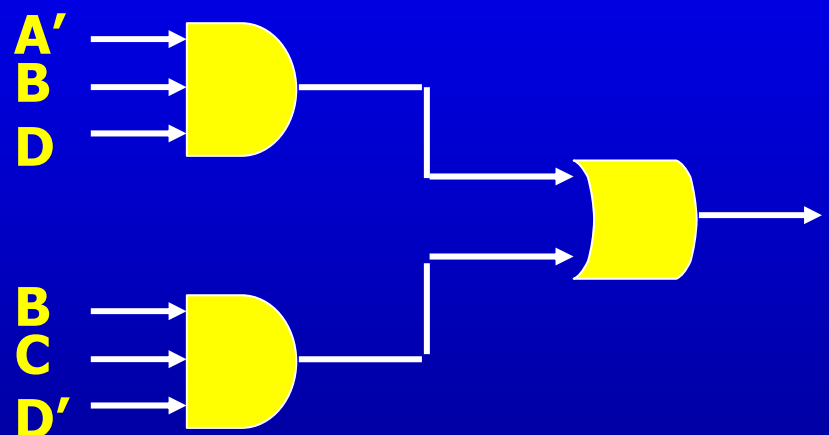
SUMMARY: Eliminating Logic Hazards for One Input Transition

transition type	hazard-free requirements
0 --> 0	?
1 --> 1	?
1 --> 0, 0 --> 1	?

Eliminating Hazards: "Static Transition" (0->0)

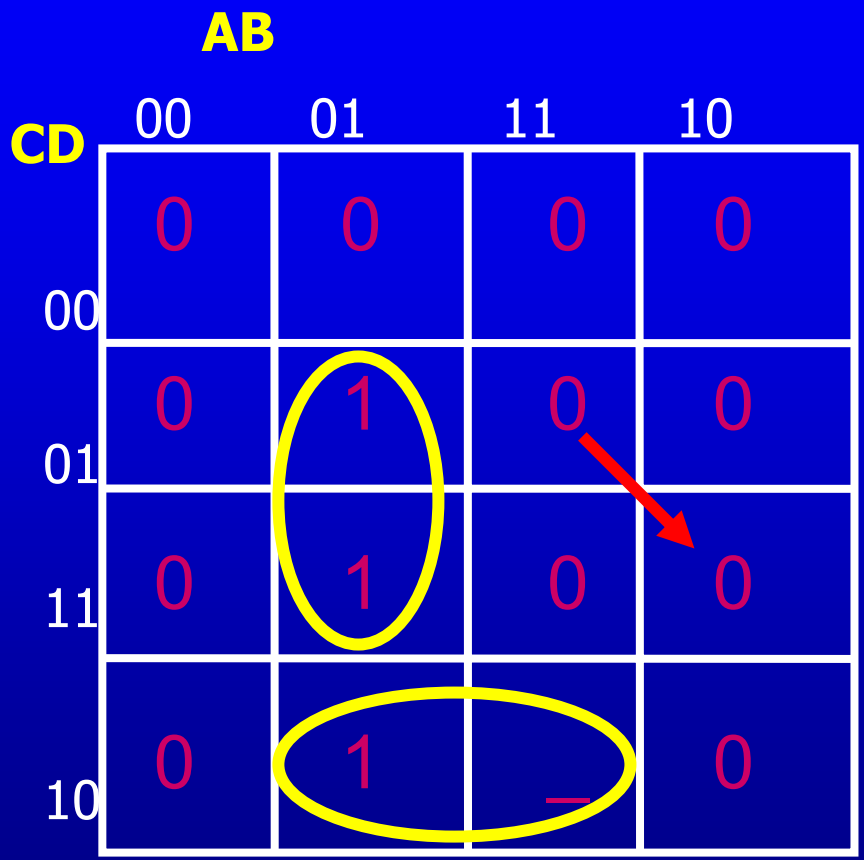
CD \ AB	00	01	11	10
00	0	0	0	0
01	0	1	0	0
11	0	1	0	0
10	0	1	—	0

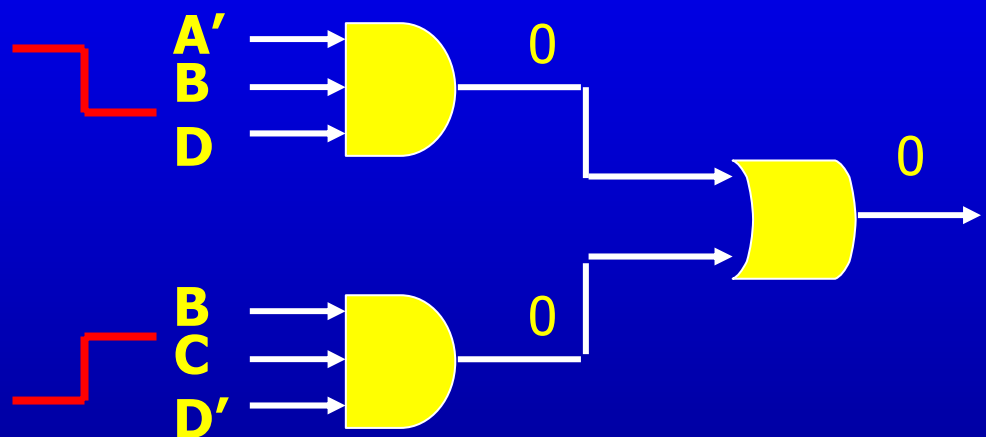
"input change": *BC: 10-->01*
(AD=11)



Example Circuit

Eliminating Hazards: 0->0 Transition





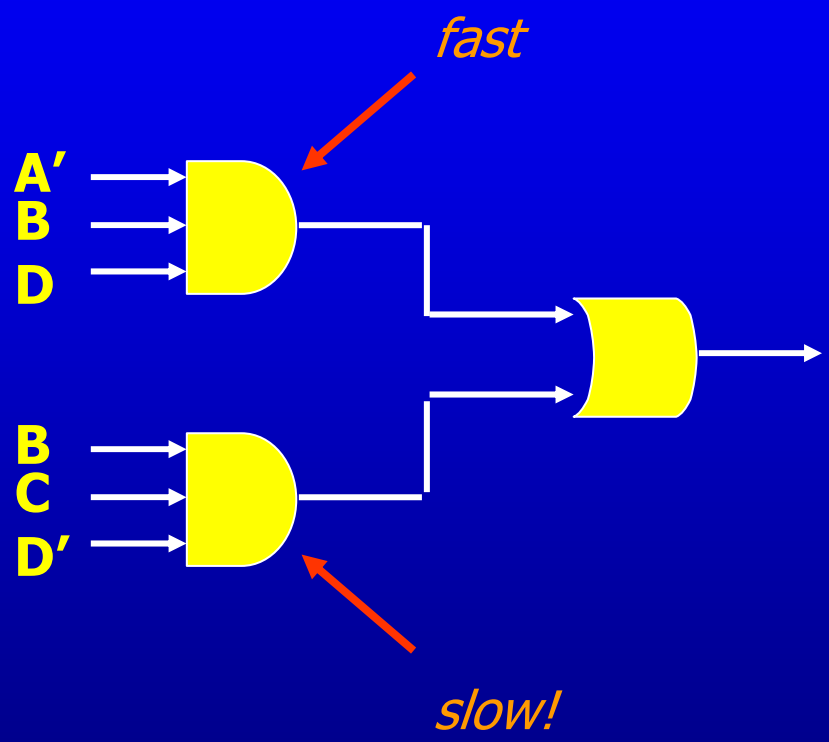
No Requirement!

If no function hazard,
then every 2-level implementation
is *free of logic hazards* [Unger]

Eliminating Hazards: "Static Transition" (1->1)

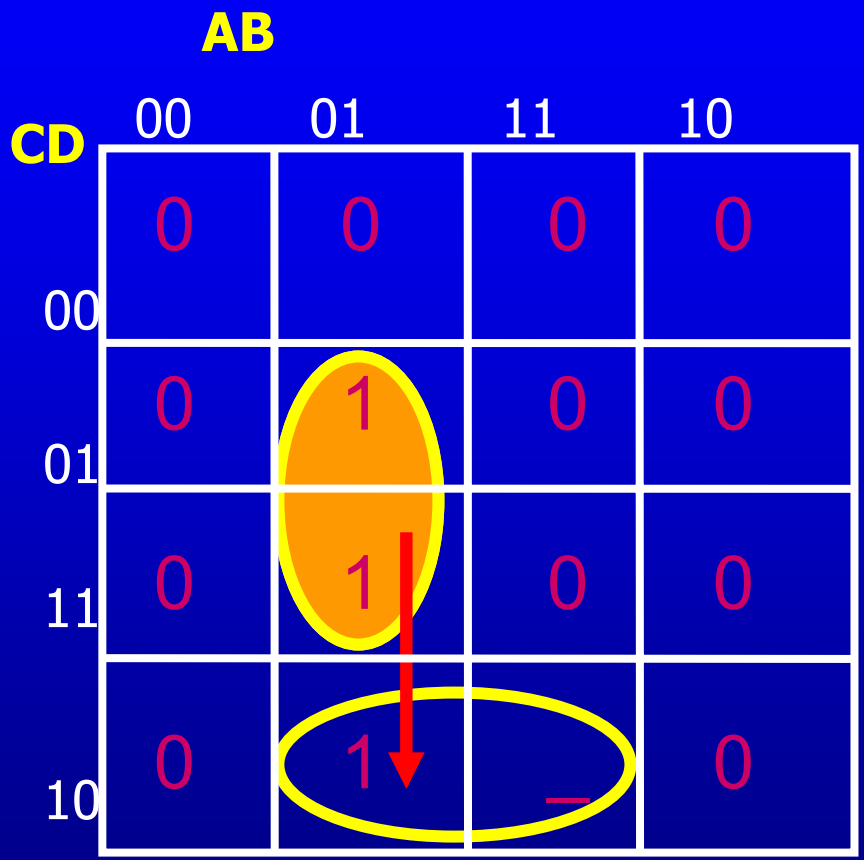
CD \ AB	00	01	11	10
00	0	0	0	0
01	0	1	0	0
11	0	1	0	0
10	0	1	–	0

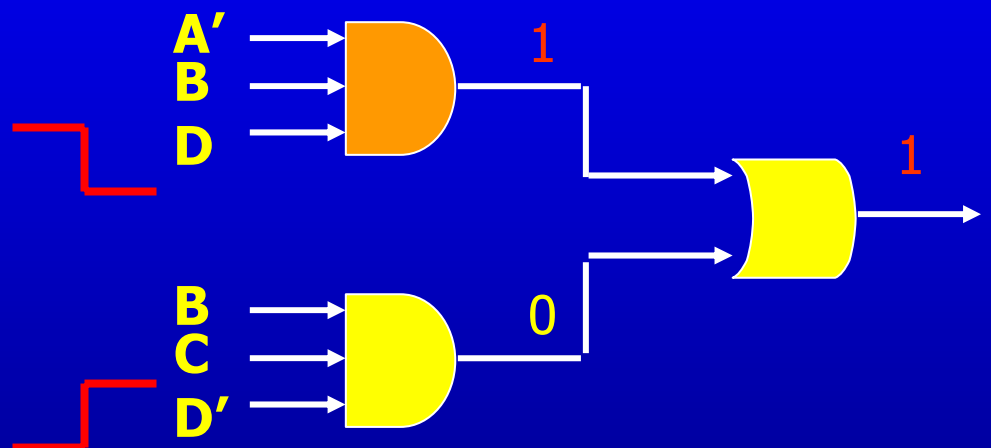
"input change": *D: 1-->0*
(ABC=011)



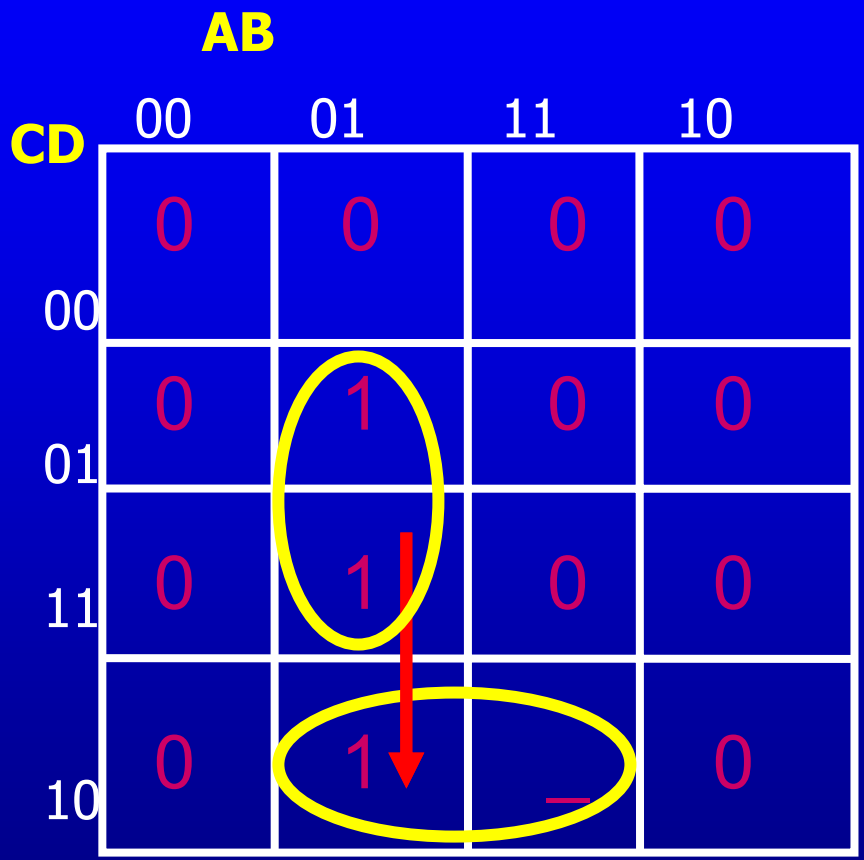
Example Circuit

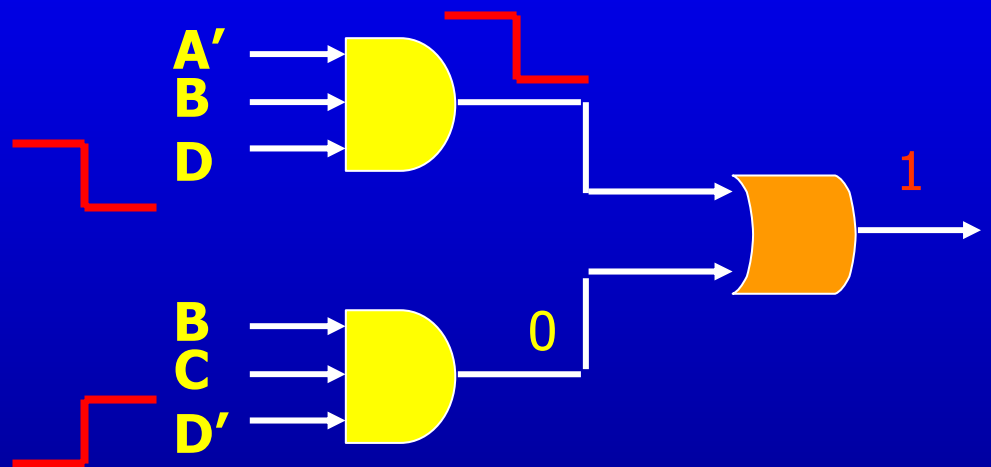
Eliminating Hazards: 1->1 Transition



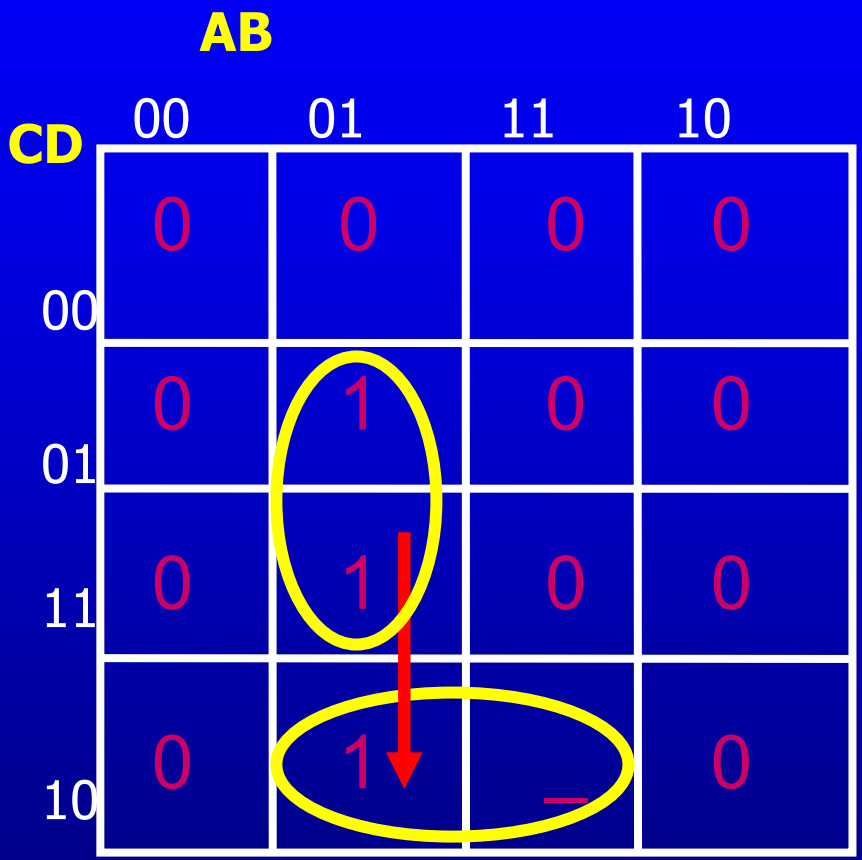


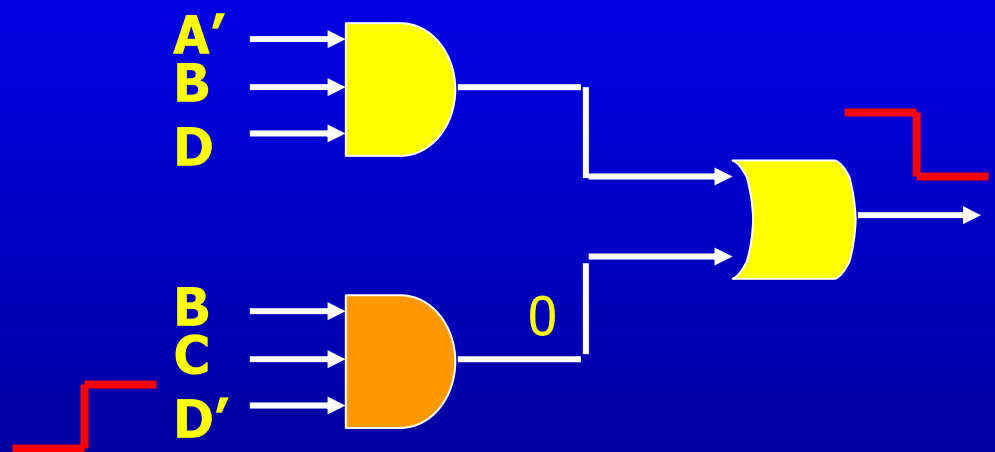
Eliminating Hazards: 1->1 Transition



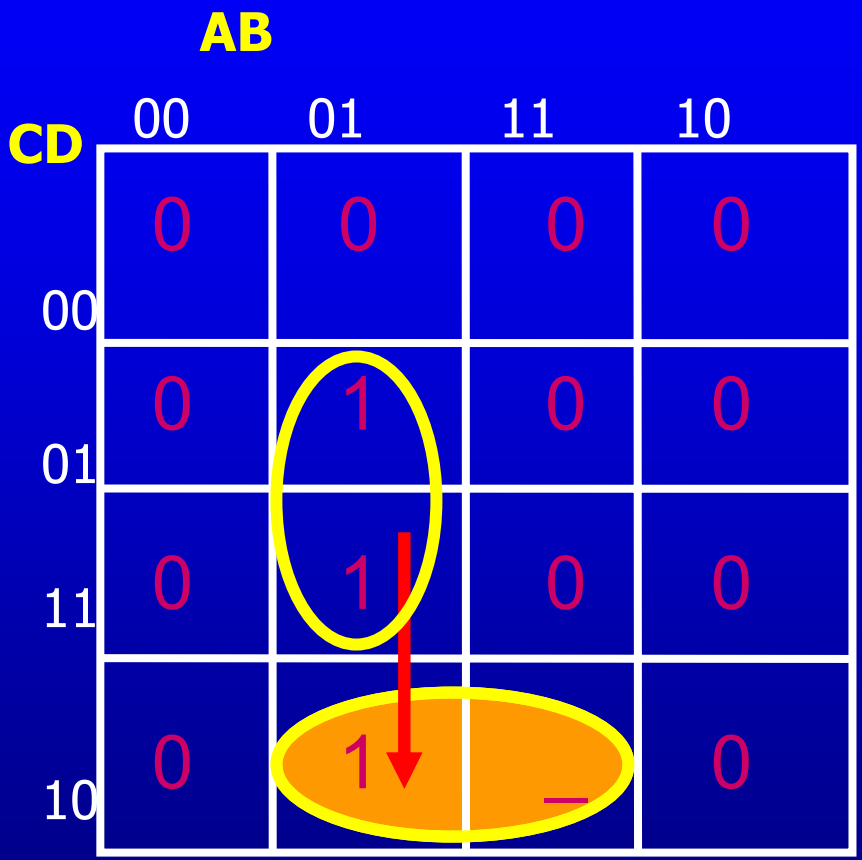


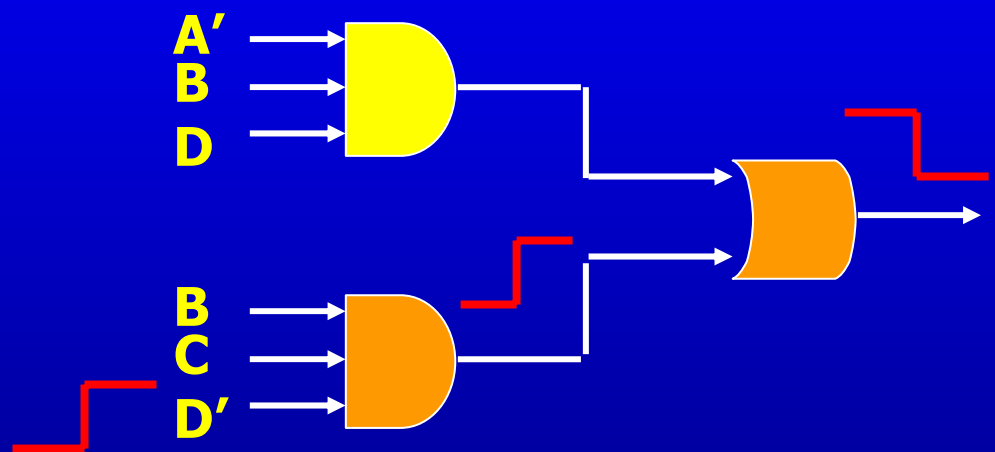
Eliminating Hazards: 1->1 Transition



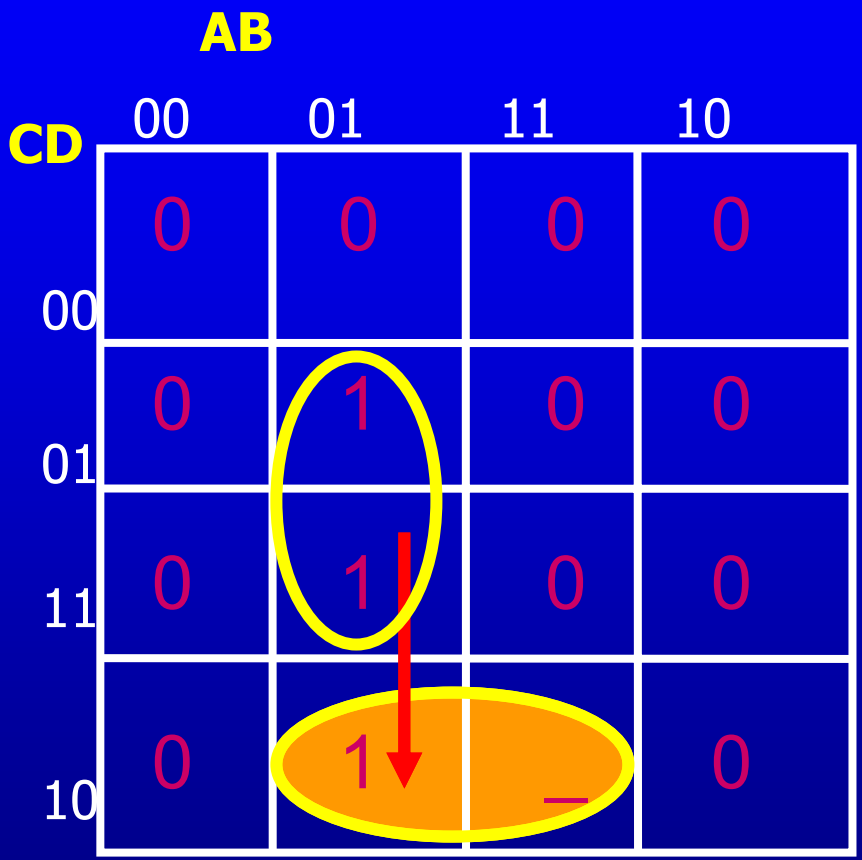


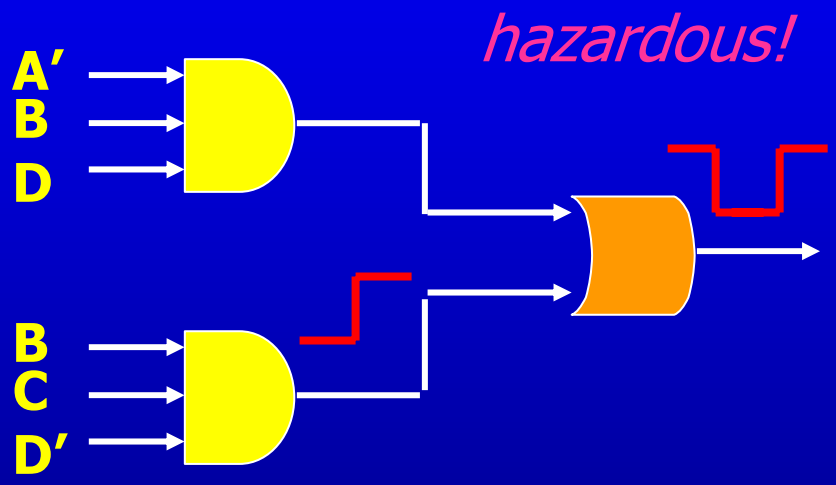
Eliminating Hazards: 1->1 Transition



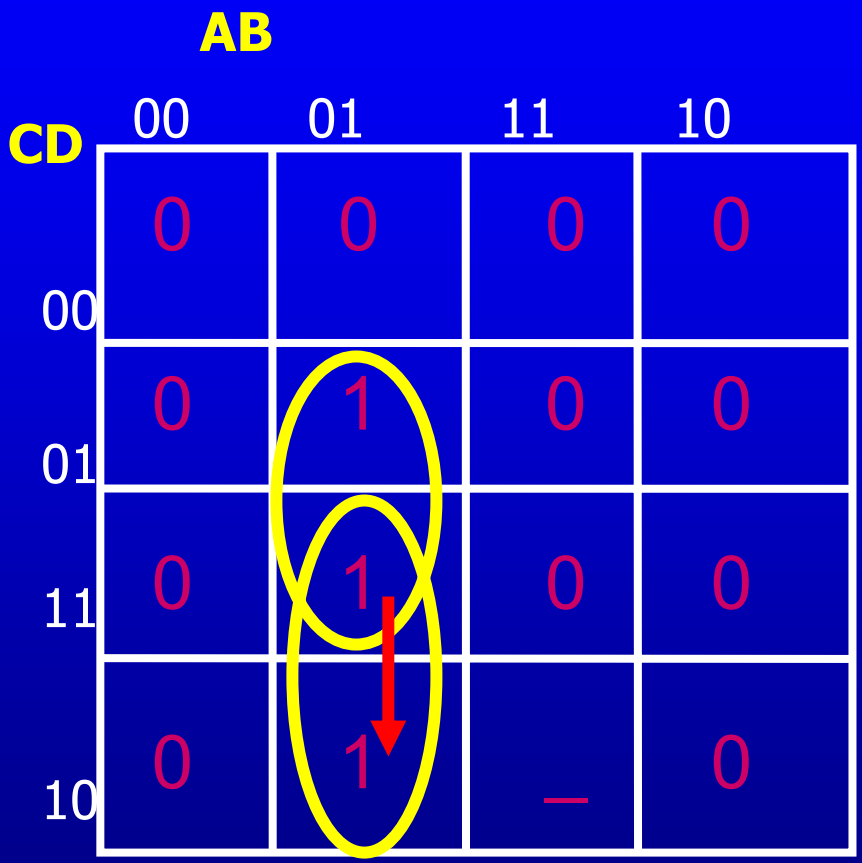


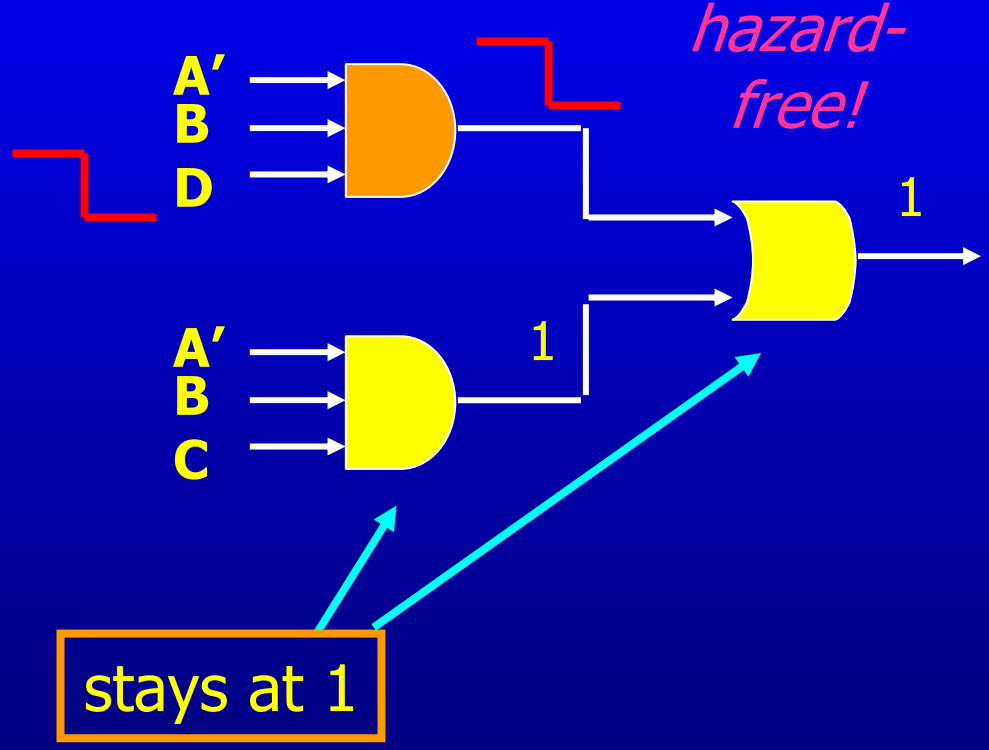
Eliminating Hazards: 1->1 Transition





Eliminating Hazards: 1->1 Transition

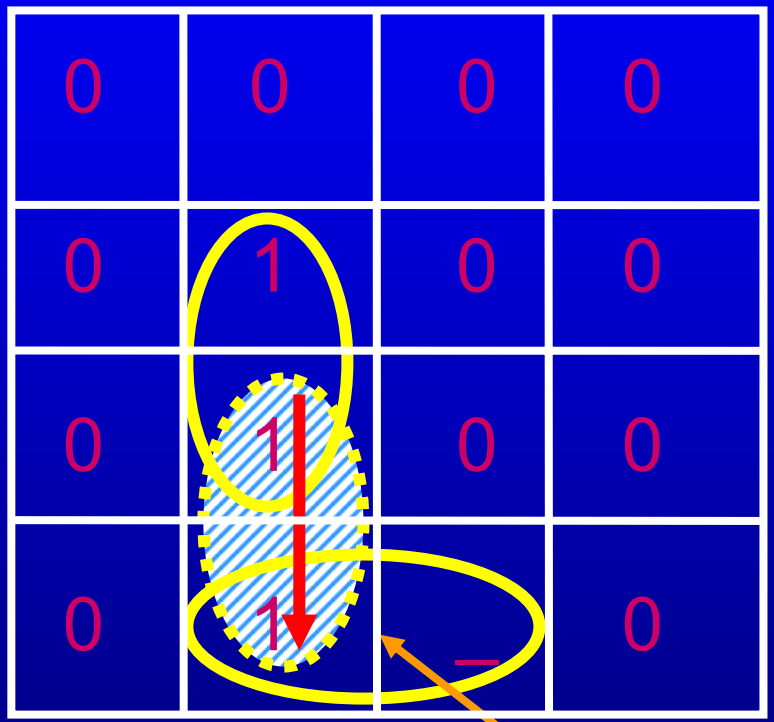




Alternative Circuit

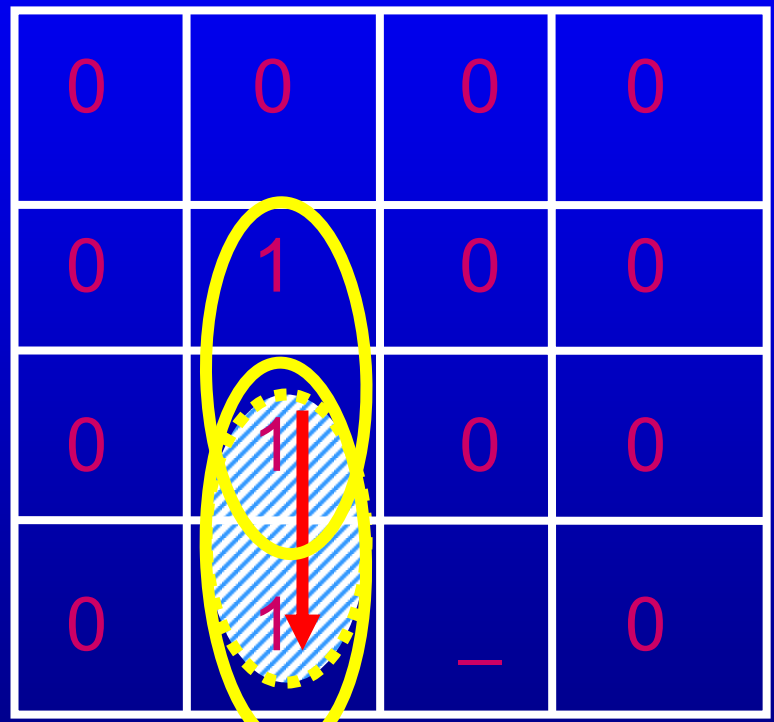
Eliminating 1->1 Hazard: Summary

hazardous



NO

hazard-free



YES

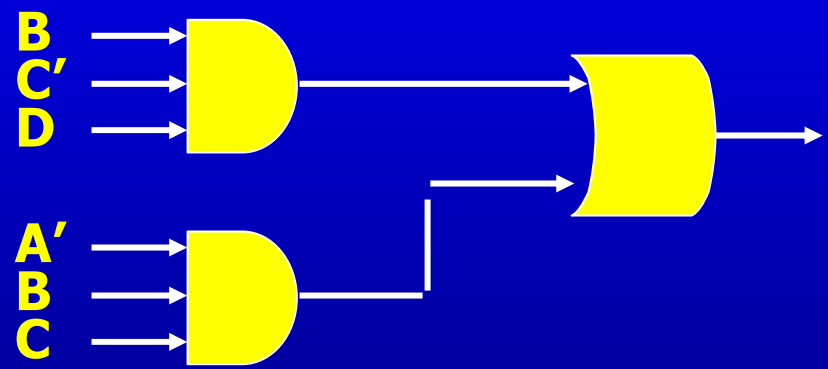
Requirement

"required cube": must be <u>completely contained</u> in some product

Eliminating Hazards: "Dynamic Transition" (1->0 or 0->1)

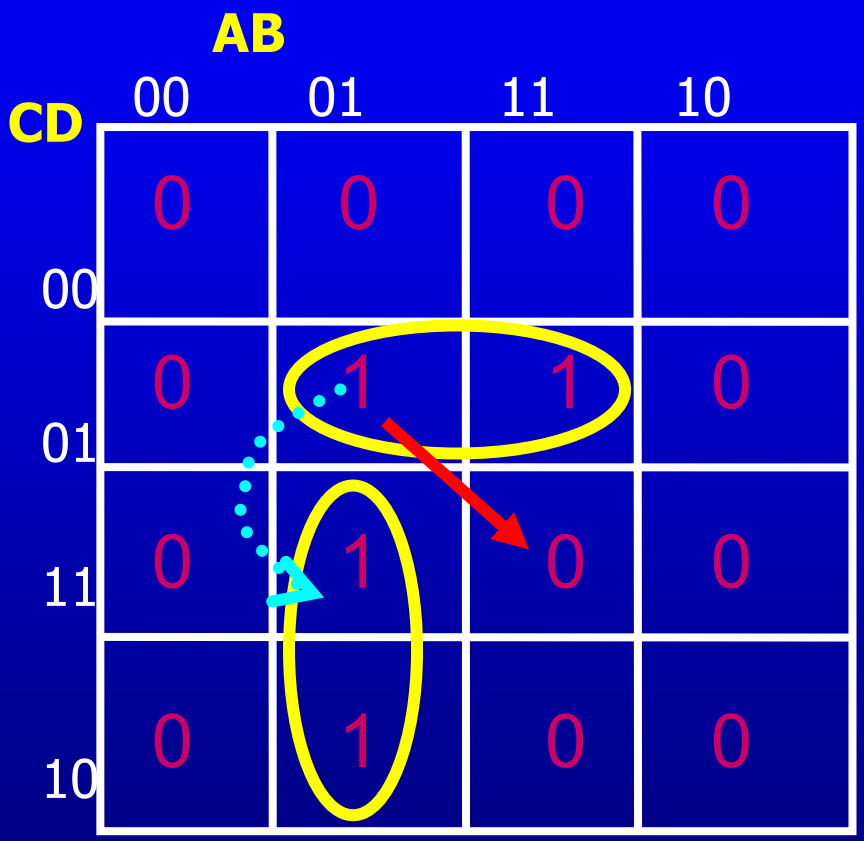
CD \ AB	00	01	11	10
00	0	0	0	0
01	0	1	1	0
11	0	1	0	0
10	0	1	0	0

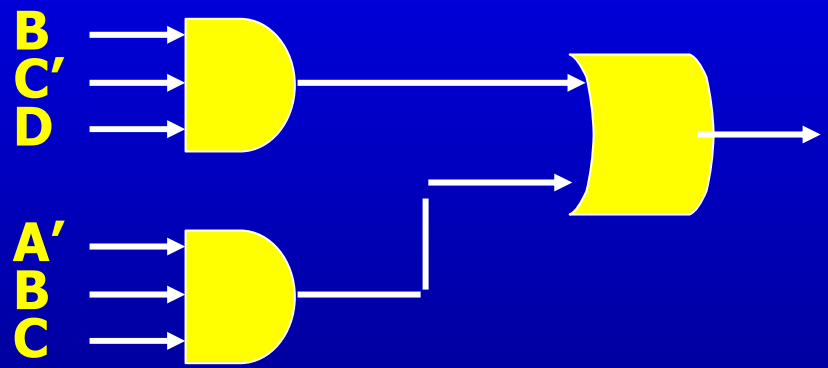
"input change": *AC: 00->11*
(BD=11)



Example Circuit

Eliminating Hazards: 1->0 Transition

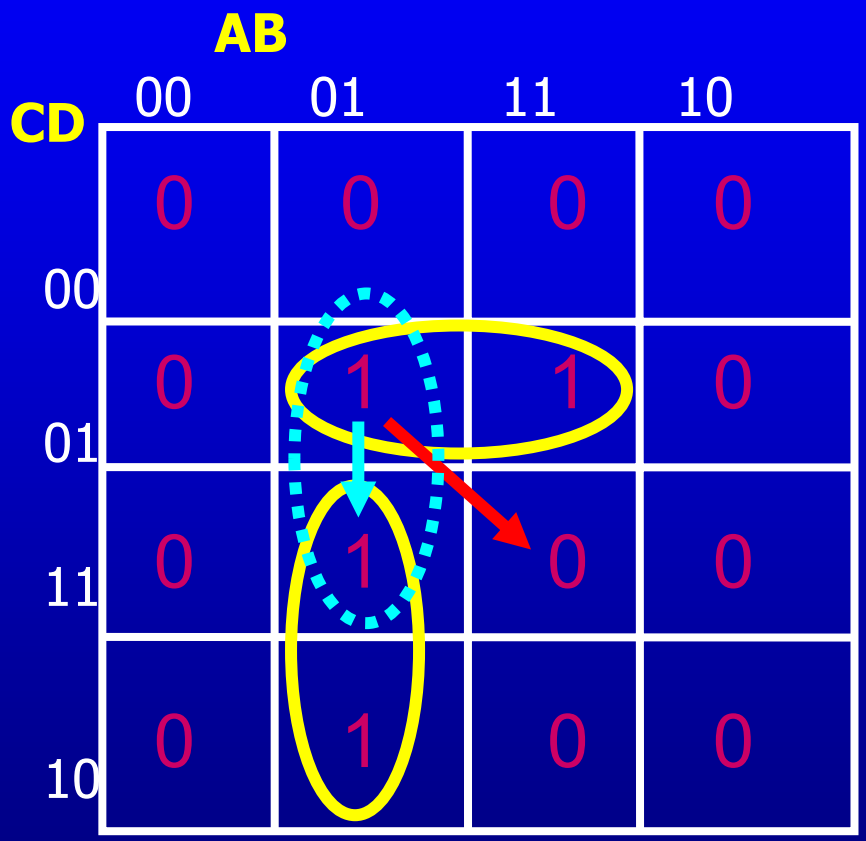


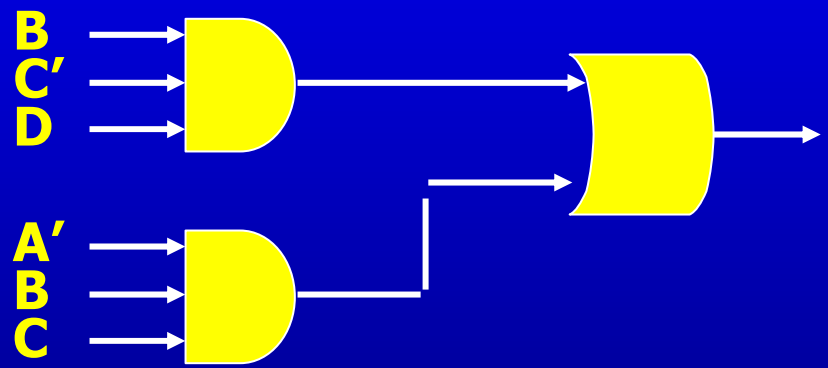


Problem #1: 1-to-1 "partial transition" is *hazardous:*

- violates 1->1 covering requirement

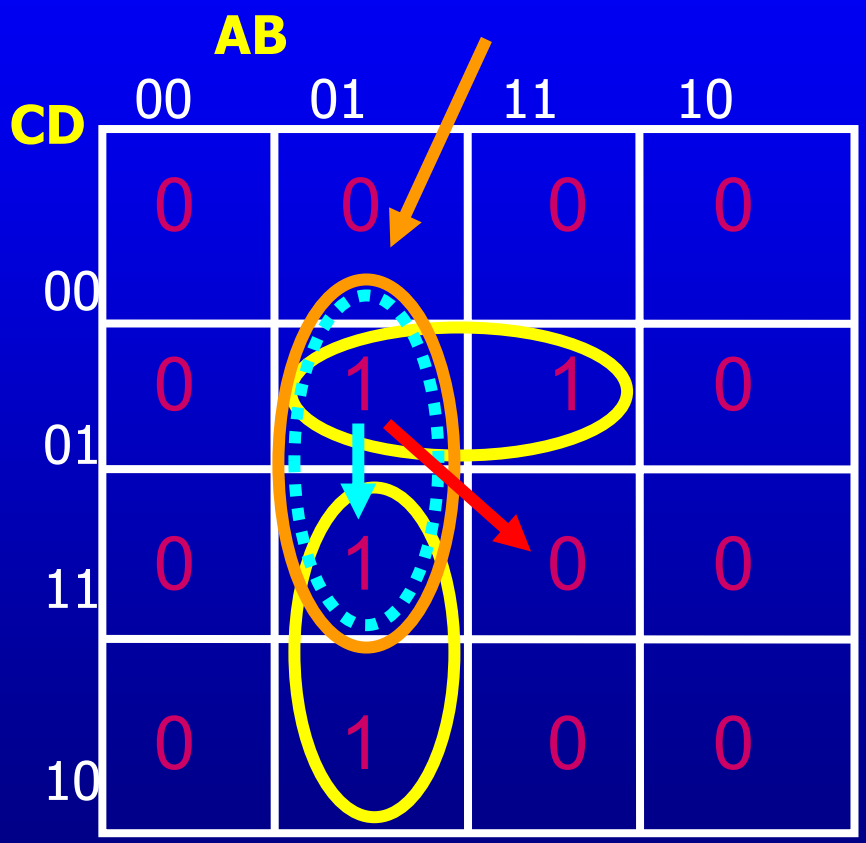
Eliminating Hazards: 1->0 Transition

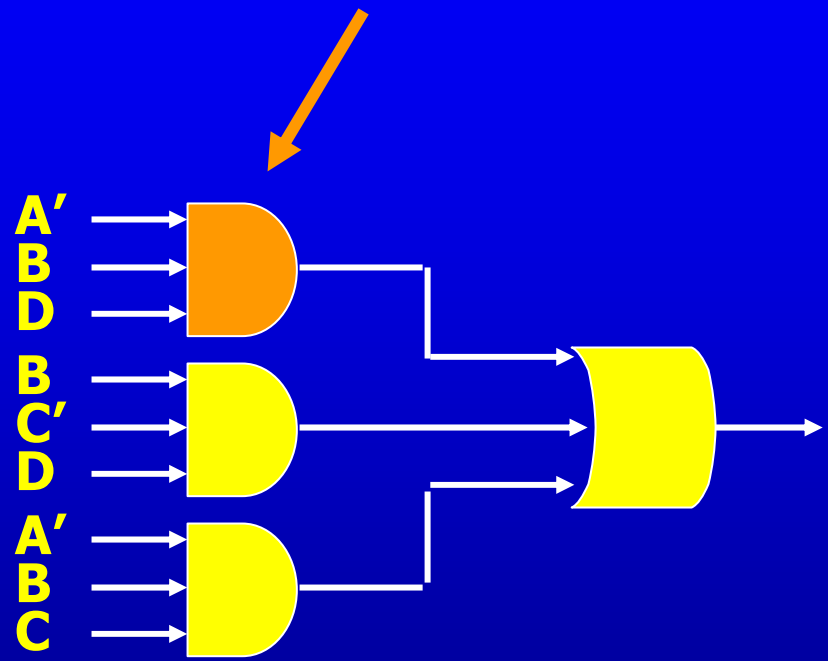




Problem #1: "required cube" for partial transition
- … *not covered by any product!*

Eliminating Hazards: 1->0 Transition



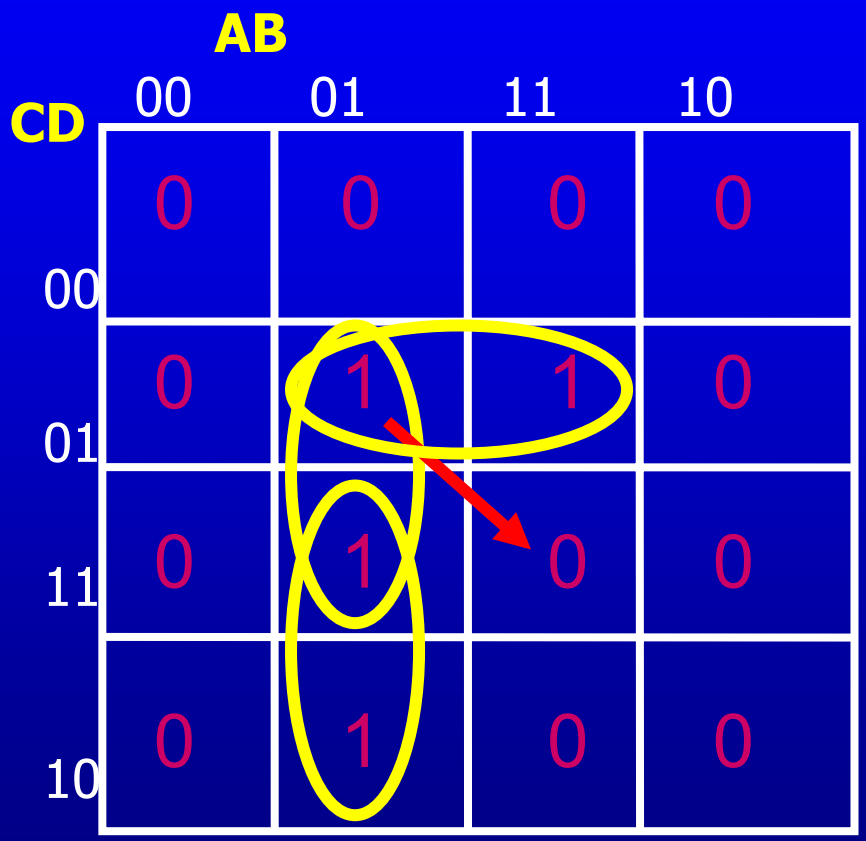


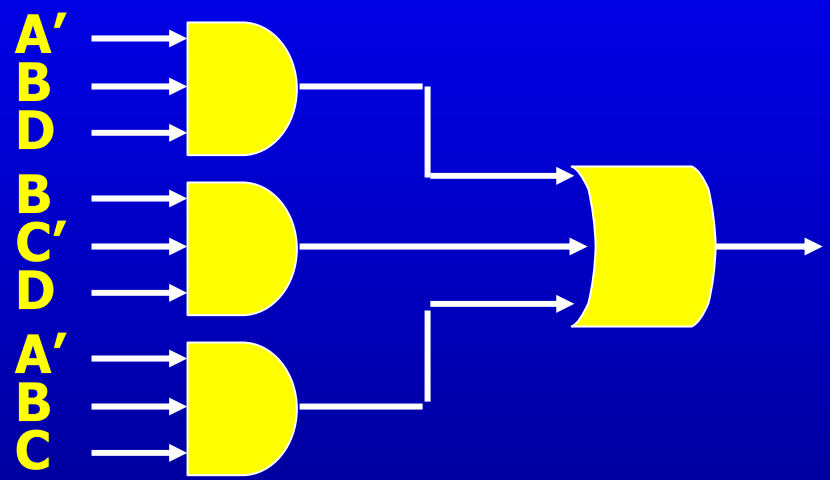
Requirement #1

Solution: cover the "required cube" for each partial transition

- … by some product

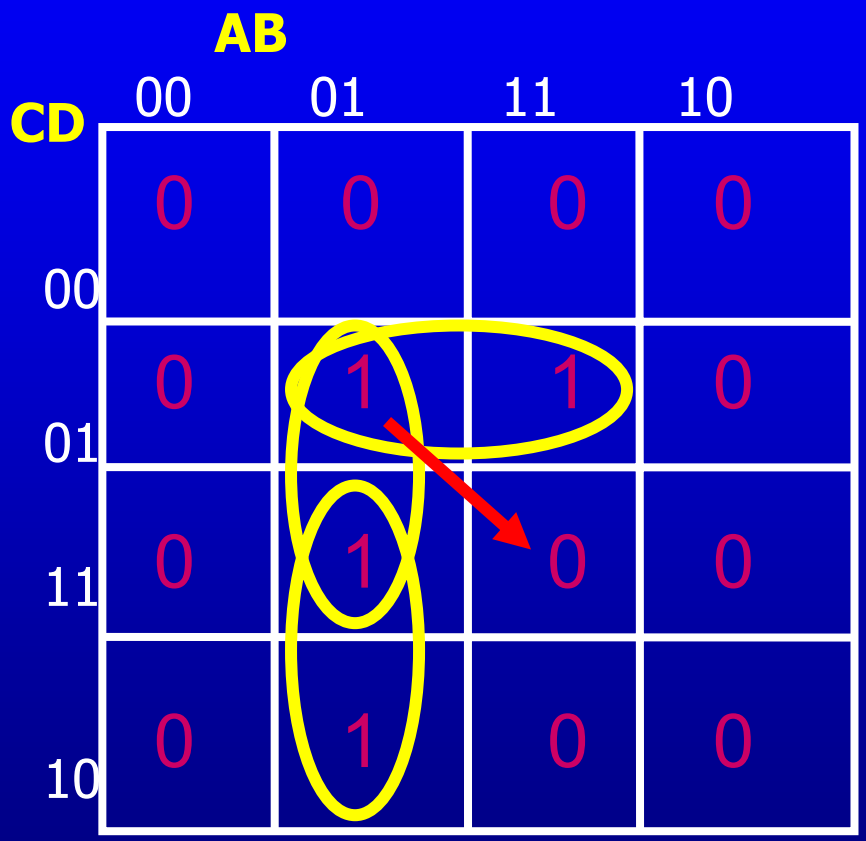
Eliminating Hazards: 1->0 Transition

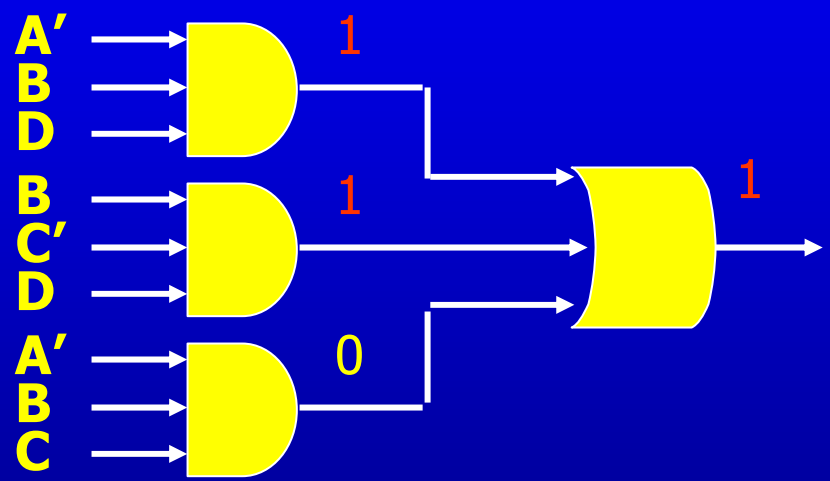




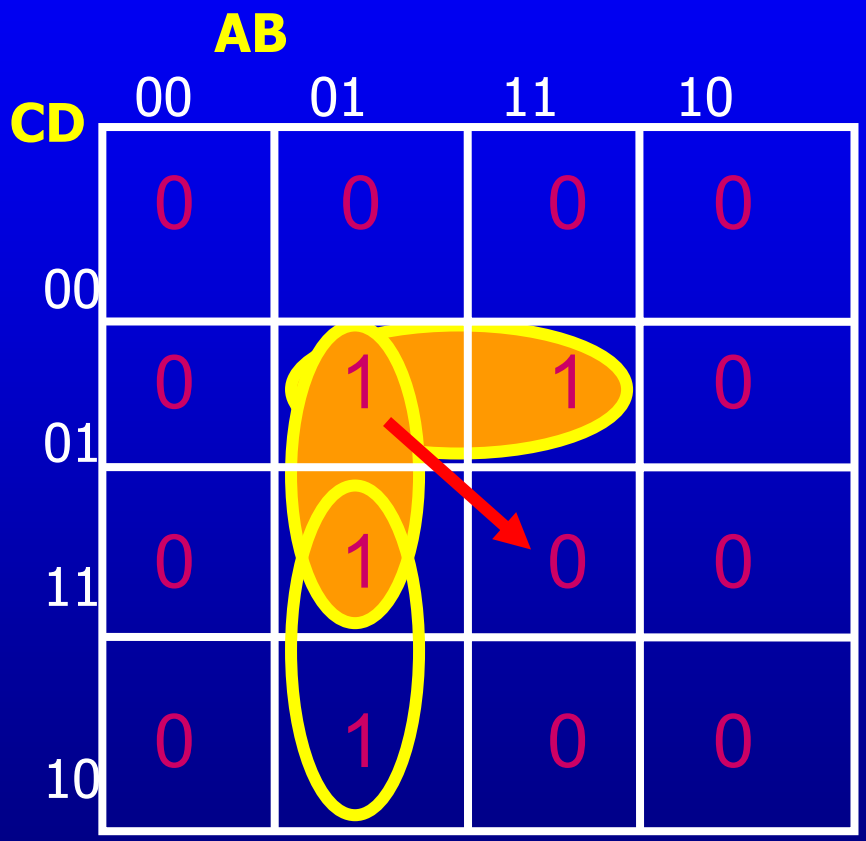
Problem #2: *entire* dynamic 1-to-0 transition *still hazardous!*

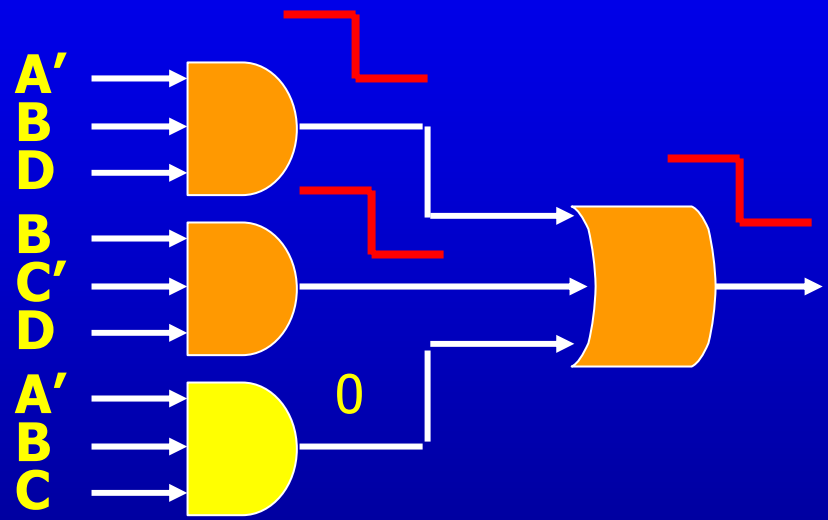
Eliminating Hazards: 1->0 Transition



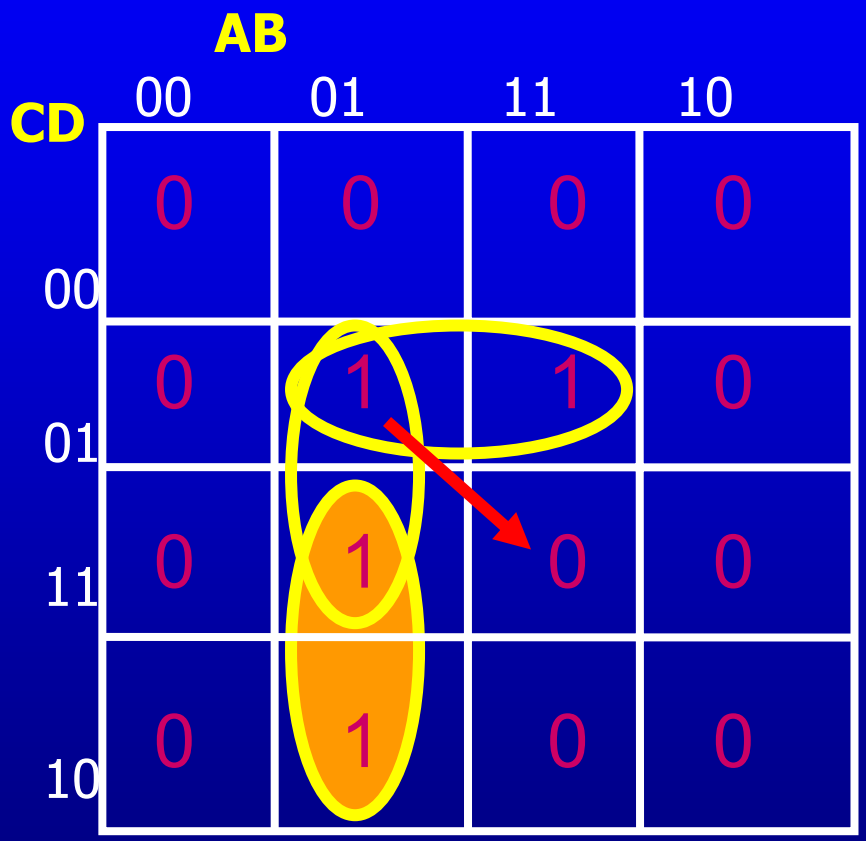


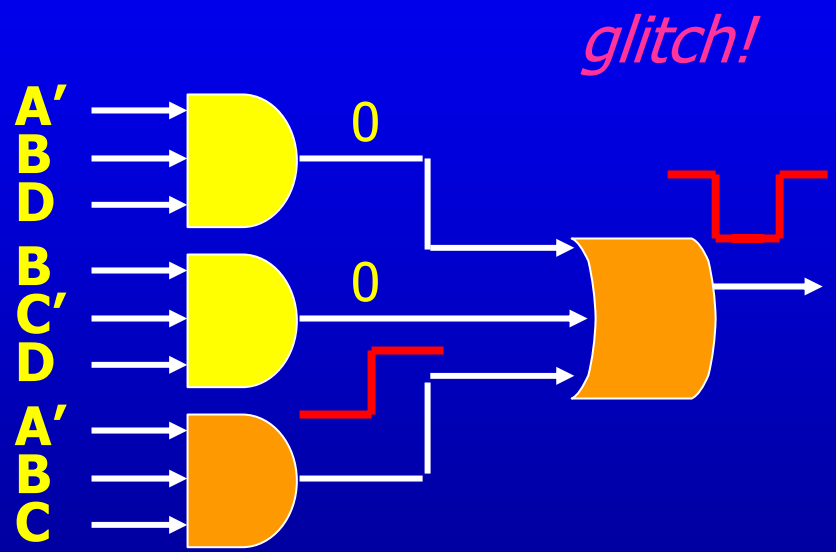
Eliminating Hazards: 1->0 Transition



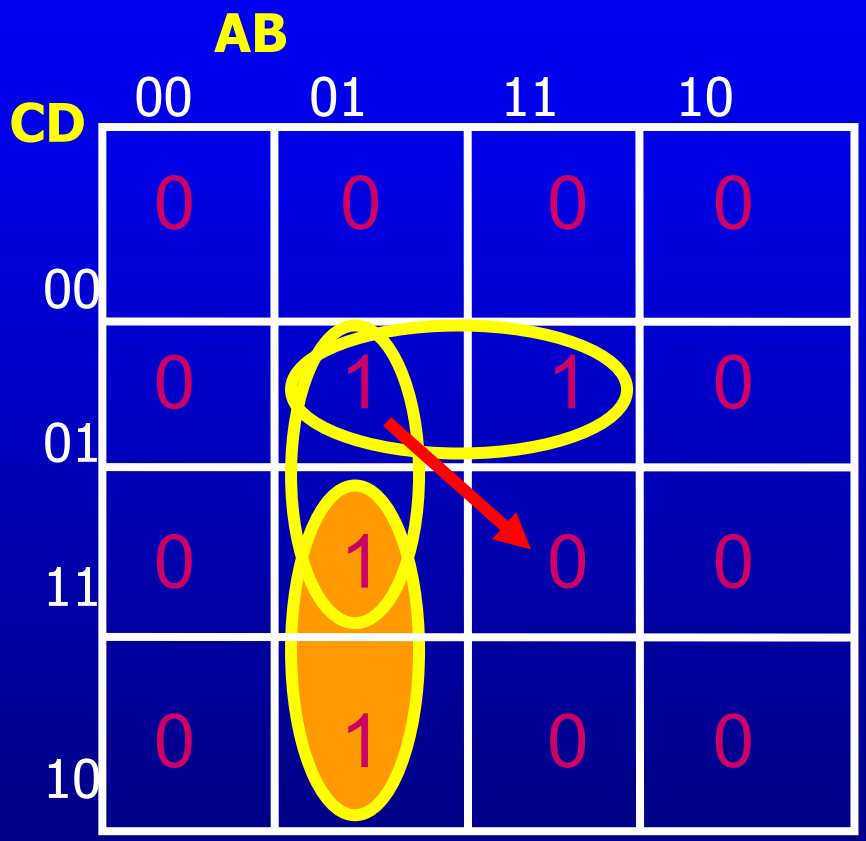


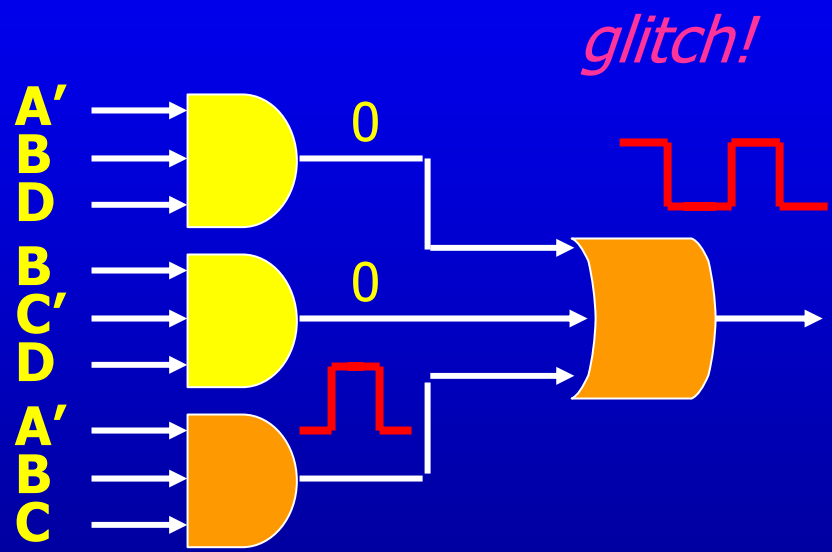
Eliminating Hazards: 1->0 Transition



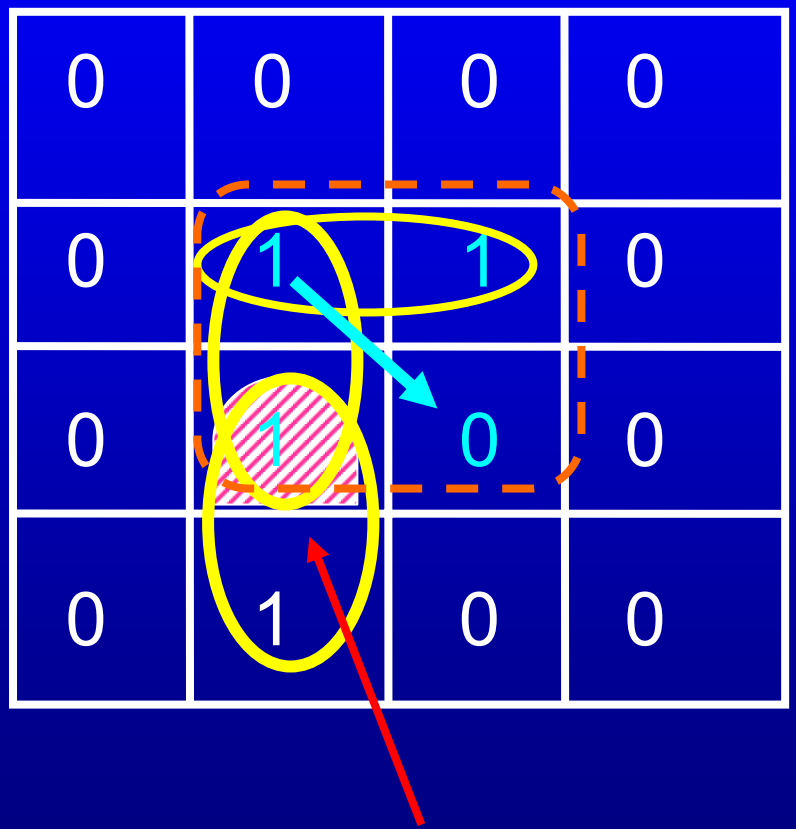


Eliminating Hazards: 1->0 Transition

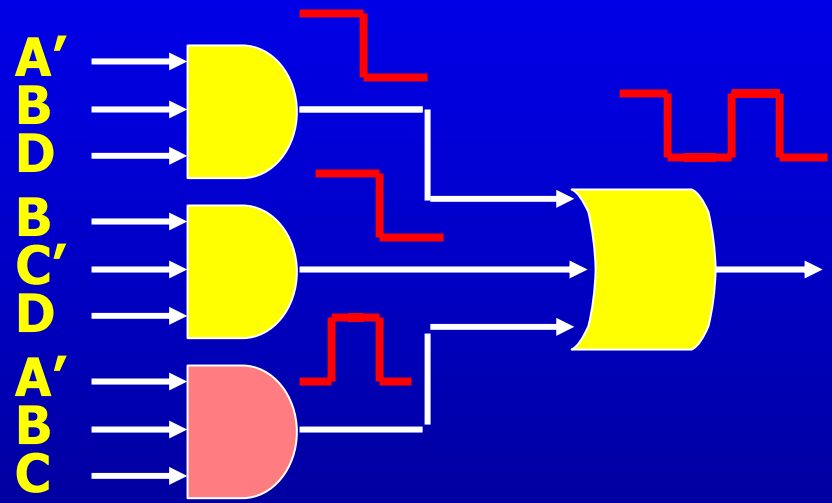




Eliminating 1->0 Hazard: Summary



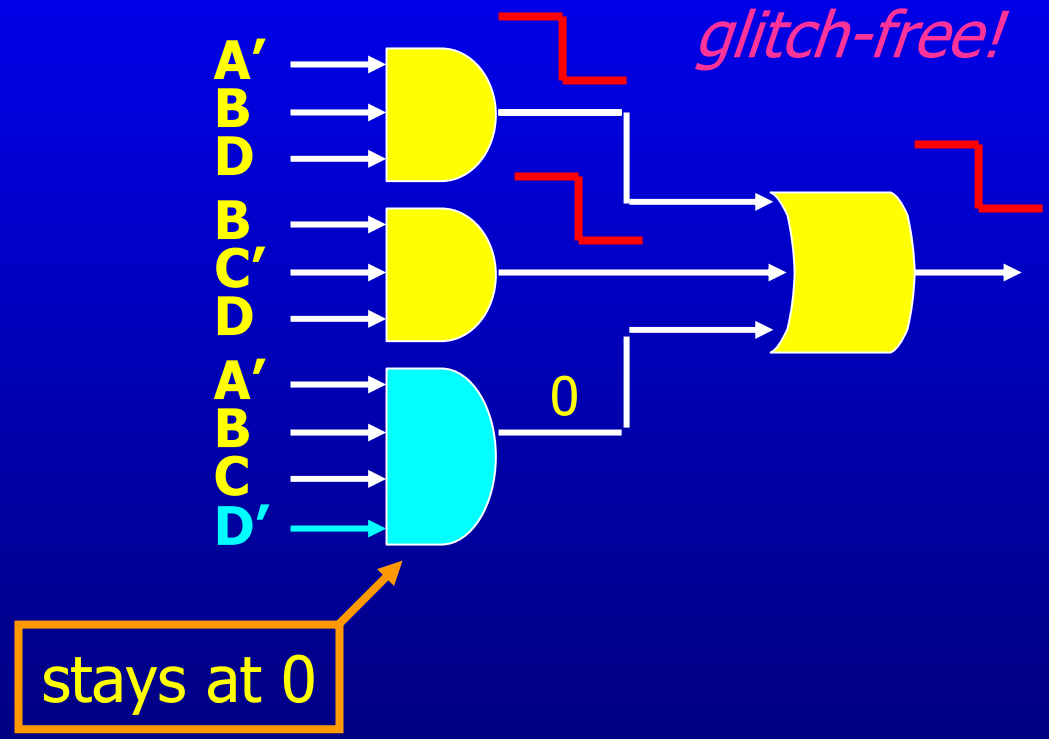
"illegal intersection"



Eliminating 1->0 Hazard: Summary

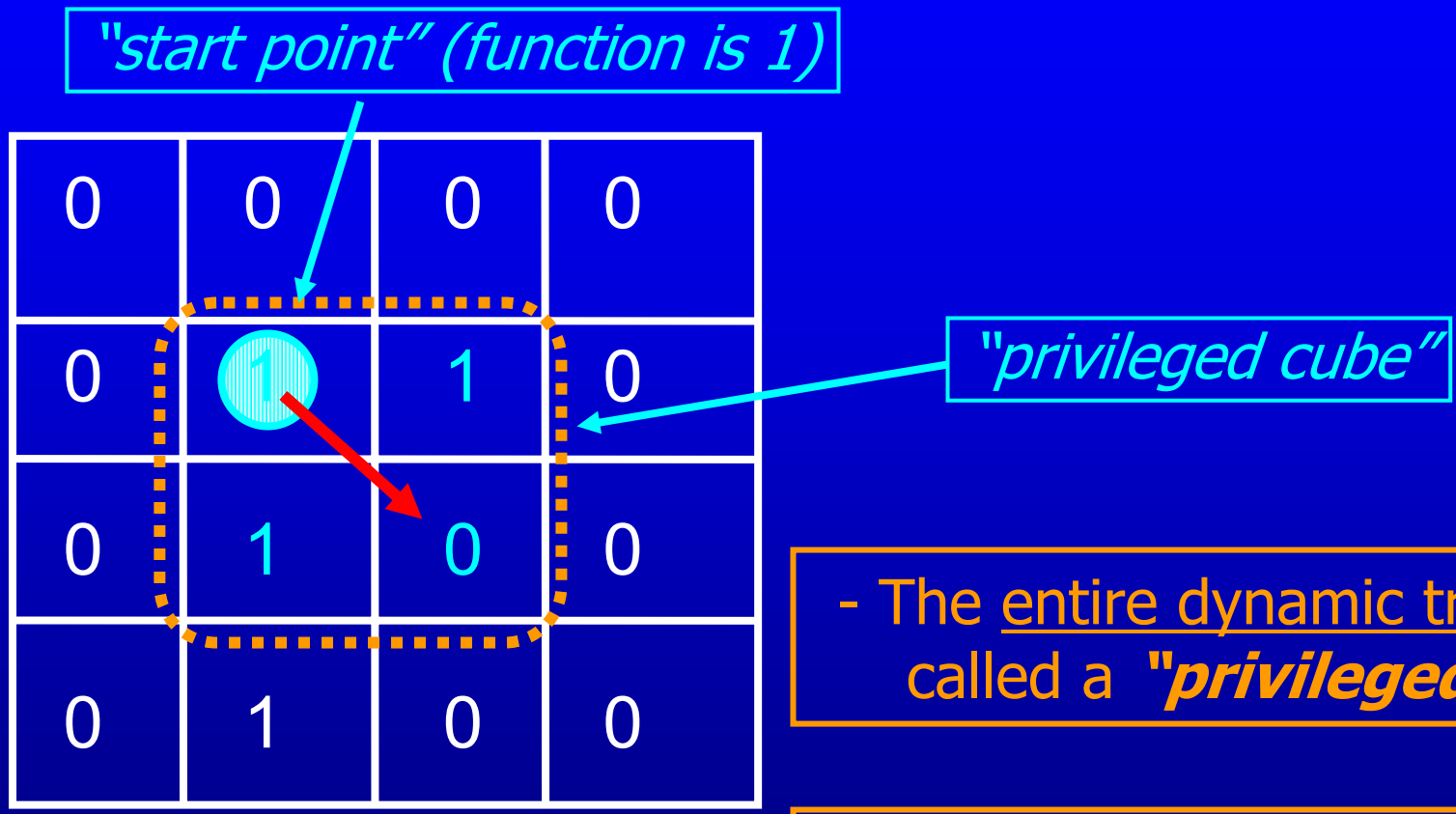
0	0	0	0
0	1	1	0
0	1	0	0
0	1	0	0

no "illegal intersection"



Alternative Circuit: Hazard-Free

Eliminating 1->0 Hazard: "Privileged Cubes"

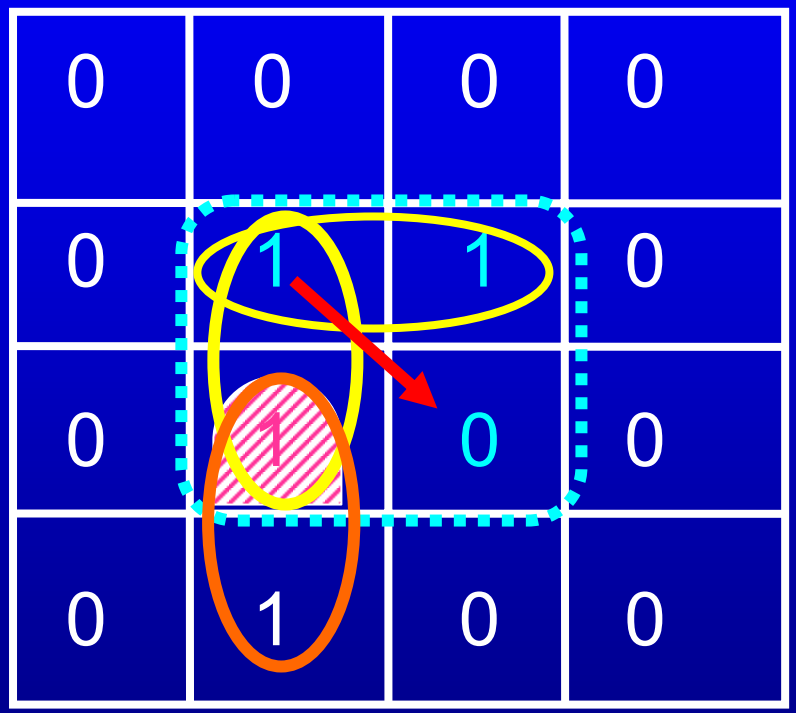


- The entire dynamic transition is called a ***"privileged cube"***

- **No implicant can intersect** any *"privileged cube"* **unless it also contains its *"start point"***

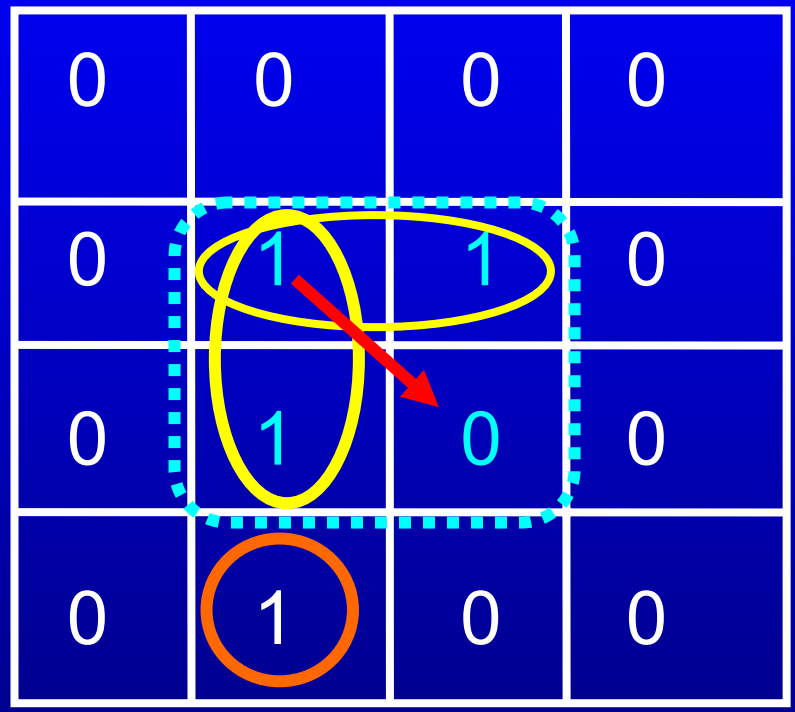
Eliminating 1->0 Hazard: Summary

hazardous



illegal intersection

hazard-free



legal intersection

Requirement #2

"privileged cube": must not be illegally intersected by any product

FINAL SUMMARY: Eliminating Logic Hazards for One Input Transition

transition type	hazard-free requirements
0 --> 0	- *[none]*
1 --> 1	- required cube: must be covered by some implicant
1 --> 0, 0 --> 1	- required cubes: each must be covered by some implicant - privileged cube: must not be illegally intersected

PROBLEM #2: Eliminating Logic Hazards for *Several* Input Transitions

"2-Level Hazard-Free Logic Minimization Problem"

Given:

- a Boolean function
- **a specified set of input transitions**

Find:

- **a minimum-cost 2-level implementation** which is hazard-free for each specified input transition (i.e, guaranteed not to glitch)

Goals and Assumptions:

- produce hazard-free combinational circuit:
  - guaranteed glitch-free, regardless of gate+wire delays
- inputs: assumed to be glitch-free

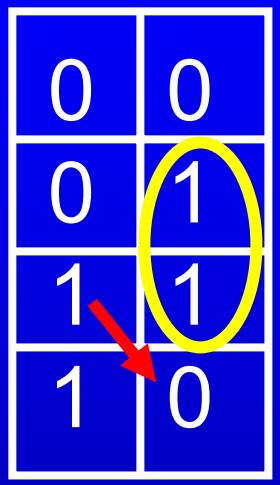
2-Level Hazard-Free Logic Minimization Problem

Equivalent Goal

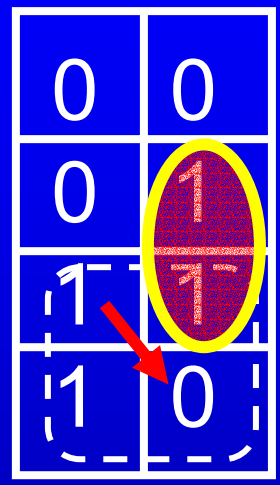
Find a 2-level circuit implementation, where:

- no privileged cube is *"illegally intersected"* by a product; and
- each required cube is *completely contained* in some product.

"Dynamic Hazard-Free (DHF) Prime Implicants"



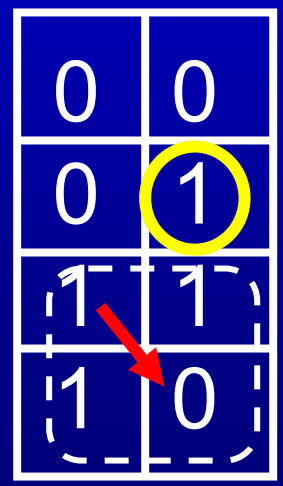
Prime Implicant



NOT DHF-Prime: has illegal intersection

DHF-Prime Implicant =

a maximal implicant which has *no "illegal intersections"* with any privileged cubes



DHF-Prime Implicant: no illegal intersections

2-Level Hazard-Free Logic Minimization Problem (cont.)

Revised Goal (version #2):

Find a 2-level circuit implementation:

- … using *only* DHF-prime implicants,
- … where each required cube is *completely covered* by some product.

2-Level Logic Minimization: a Comparison (Classic vs. Hazard-Free)

In each case, solve a *"covering problem":*

<"objects to be covered", "covering objects">

- Classic (Quine-McCluskey method, espresso-exact, …):

  <on-set minterms, prime implicants>

- Hazard-Free (Nowick/Dill [92]):

  <required cubes, DHF-prime implicants>

2-Level Logic Minimization: a Comparison

Classic Method: ***Non-Hazard-Free***

Step 1: Generate All Prime Implicants

Step 2: Generate Prime Implicant Table

Step 3: Solve Covering Problem

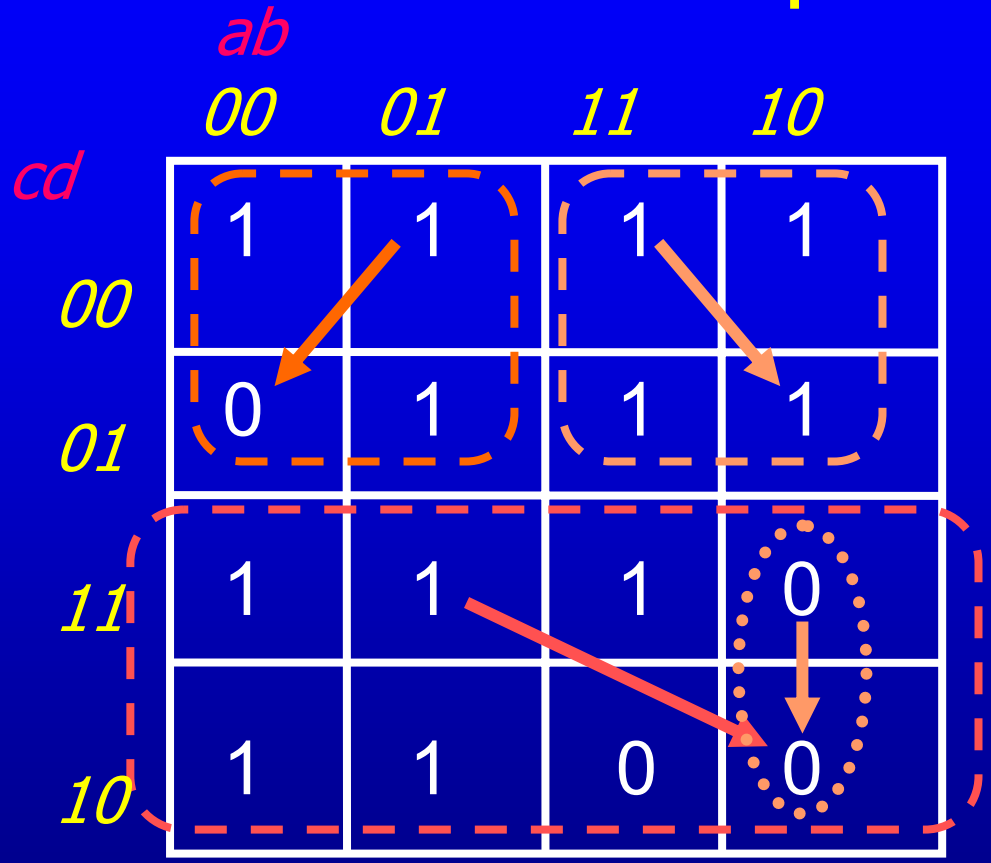
New Method: ***Hazard-Free*** [Nowick/Dill '92]

Step 1: Generate All DHF-Prime Implicants

Step 2: Generate DHF-Prime Implicant Table

Step 3: Solve Covering Problem

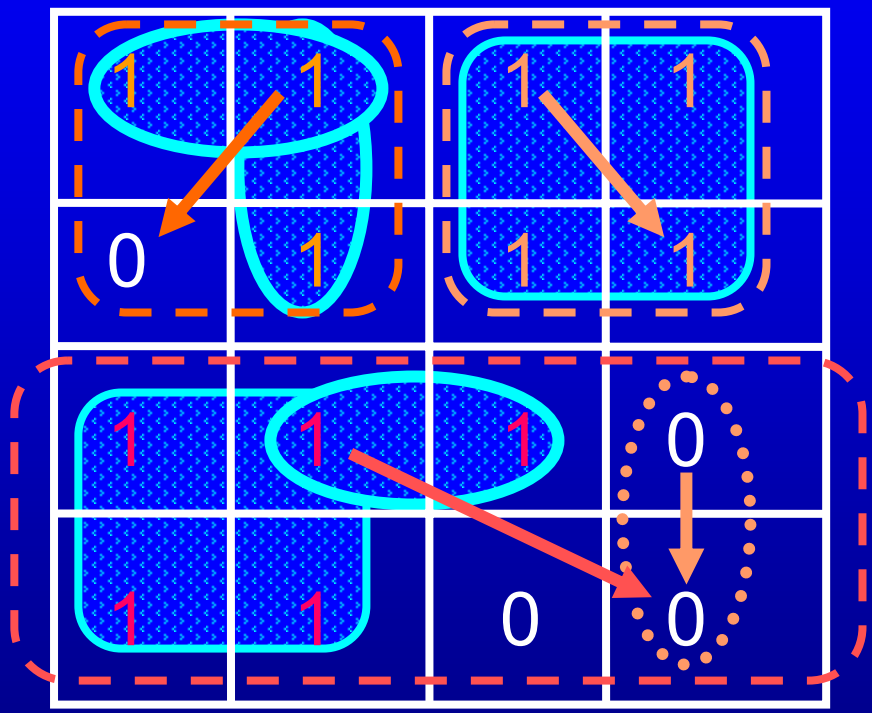
2-Level Hazard-Free Logic Minimization: a Complete Example



[from: Nowick/Dill, ICCAD'92; IEEE Trans. On CAD Aug.'95]

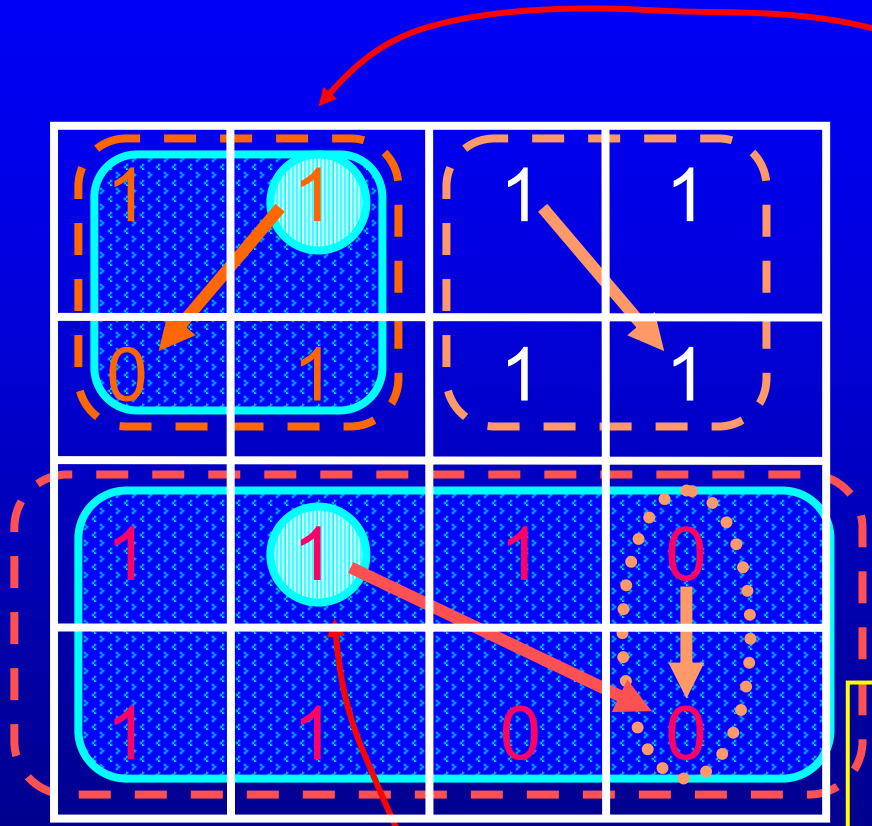
Boolean Function +
4 (function hazard-free) input transitions

2-Level Hazard-Free Logic Minimization: a Complete Example



Required Cubes:
Each required cube must be *completely contained* in some product

2-Level Hazard-Free Logic Minimization: a Complete Example



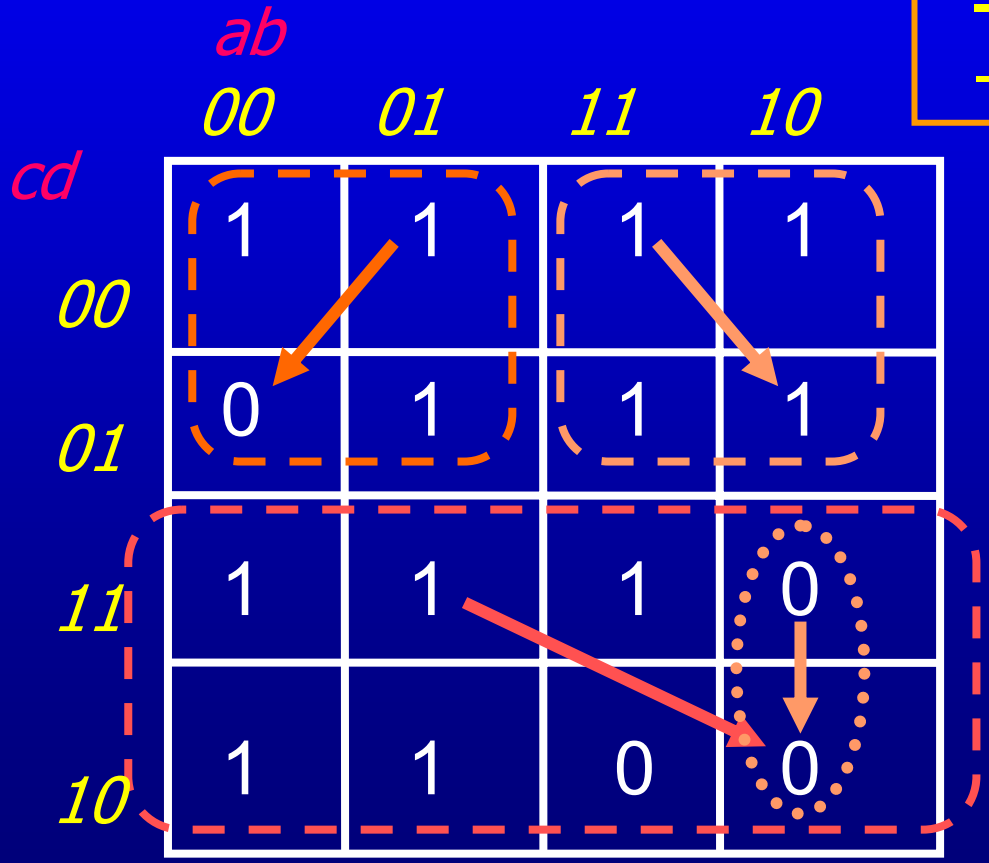
Privileged Cubes:

If any product intersects a *privileged cube*,
it must also intersect its *start point*

2-Level Hazard-Free Logic Minimization: a Complete Example

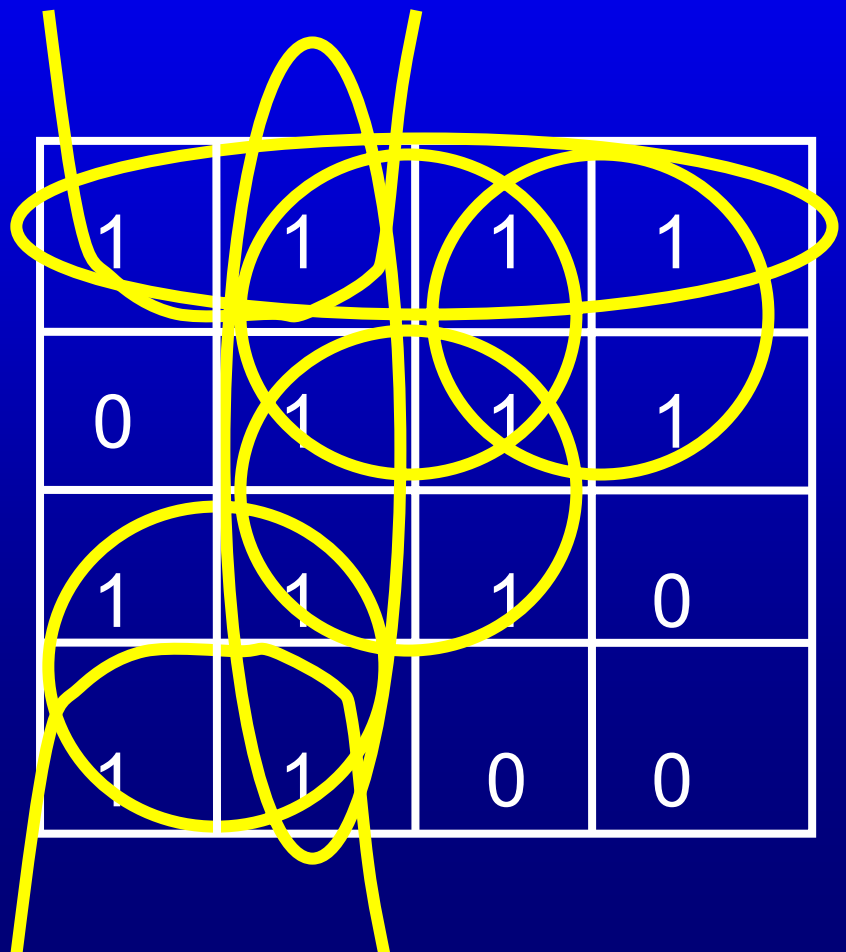
Step 1: Generate All DHF-Prime Implicants

Approach: 2 steps
- Generate All Prime Implicants
- Reduce to DHF-Prime Implicants



2-Level Hazard-Free Logic Minimization: a Complete Example

Step 1: Generate All DHF-Prime Implicants



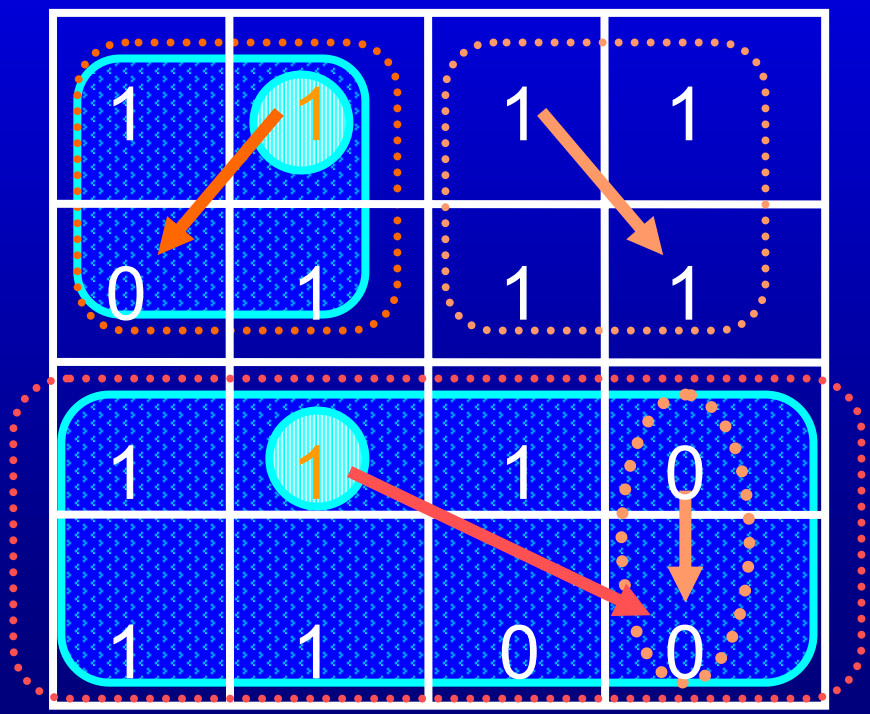
Approach: 2 steps

- Generate All Prime Implicants
- Reduce to DHF-Prime Implicants

Total: 7 Prime Implicants

2-Level Hazard-Free Logic Minimization: a Complete Example

Step 1: Generate All DHF-Prime Implicants



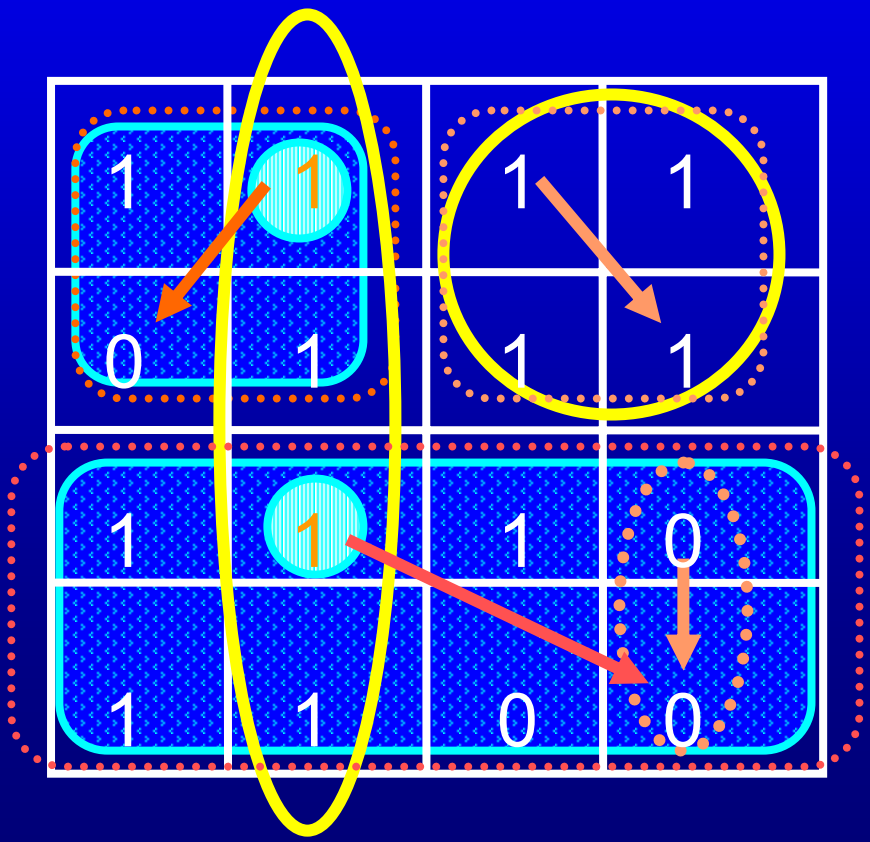
Approach: 2 steps

- Generate All Prime Implicants
- Reduce to DHF-Prime Implicants

= privileged cube

2-Level Hazard-Free Logic Minimization: a Complete Example

Step 1: Generate All DHF-Prime Implicants



Approach: 2 steps

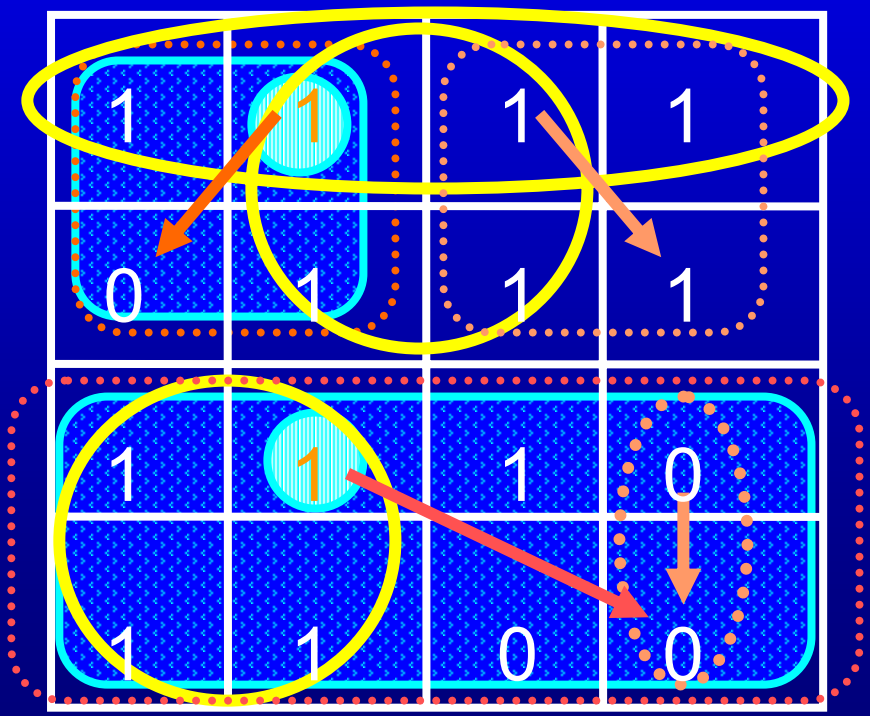
- Generate All Prime Implicants
- ⇒ Reduce to DHF-Prime Implicants

Some primes have no illegal intersections

=> they are DHF-primes

2-Level Hazard-Free Logic Minimization: a Complete Example

Step 1: Generate All DHF-Prime Implicants



Approach: 2 steps

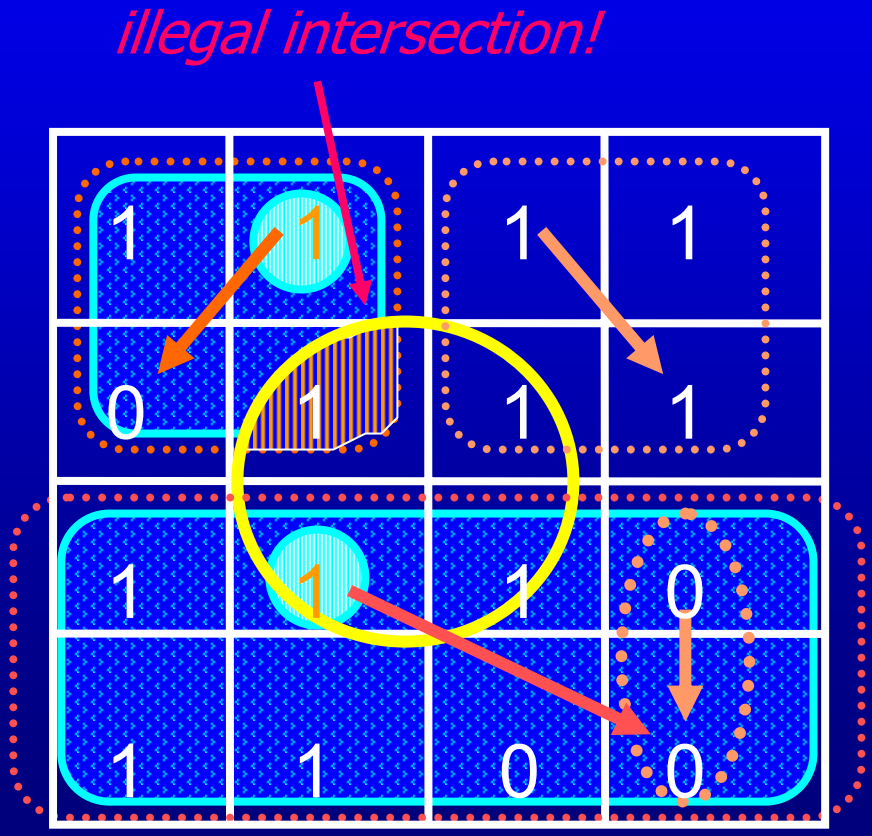
- Generate All Prime Implicants
- Reduce to DHF-Prime Implicants

Some primes have no illegal intersections

=> they are DHF-primes

2-Level Hazard-Free Logic Minimization: a Complete Example

Step 1: <u>Generate All DHF-Prime Implicants</u>



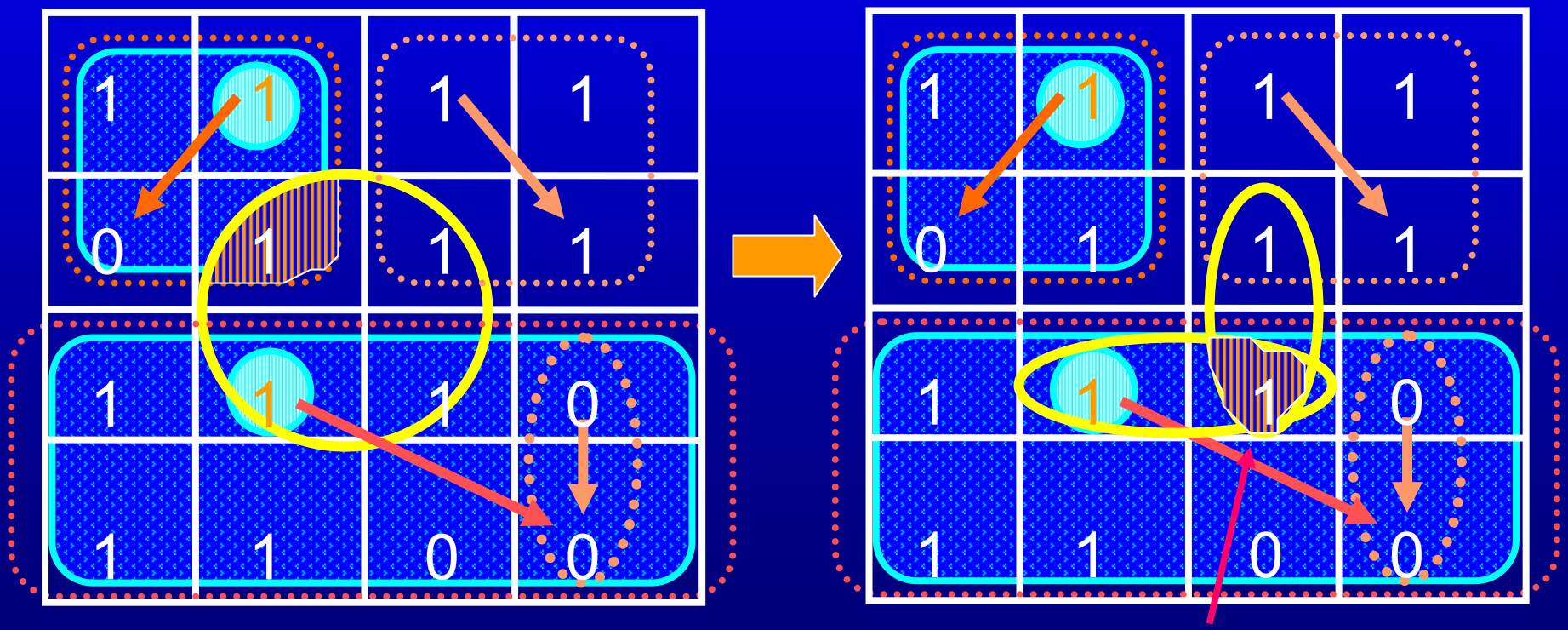
<u>Approach: 2 steps</u>
- Generate All Prime Implicants
- ⇒ Reduce to DHF-Prime Implicants

<u>Other primes have illegal intersections</u>
=> they must be reduced

2-Level Hazard-Free Logic Minimization: a Complete Example

Step 1: Generate All DHF-Prime Implicants

First reduction of prime implicant

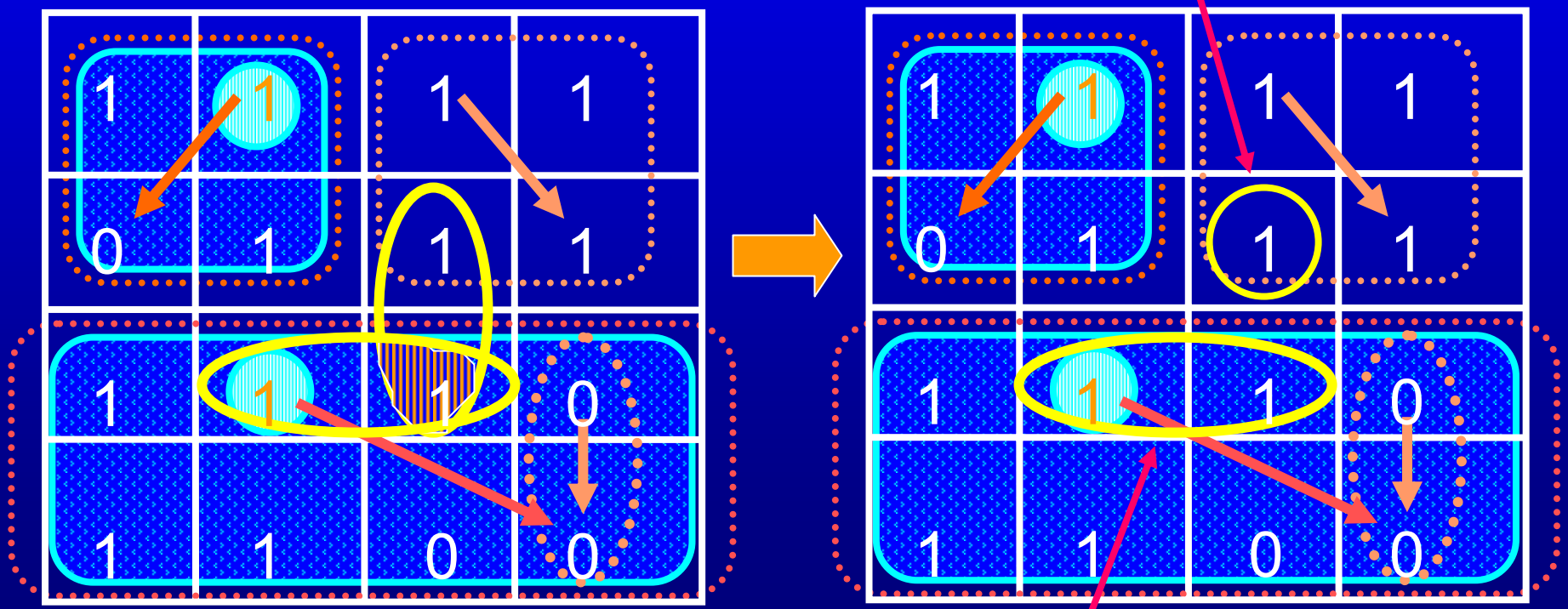


2-Level Hazard-Free Logic Minimization: a Complete Example

Step 1: Generate All DHF-Prime Implicants

Second reduction of prime implicant

DISCARD: contained in a DHF-Prime

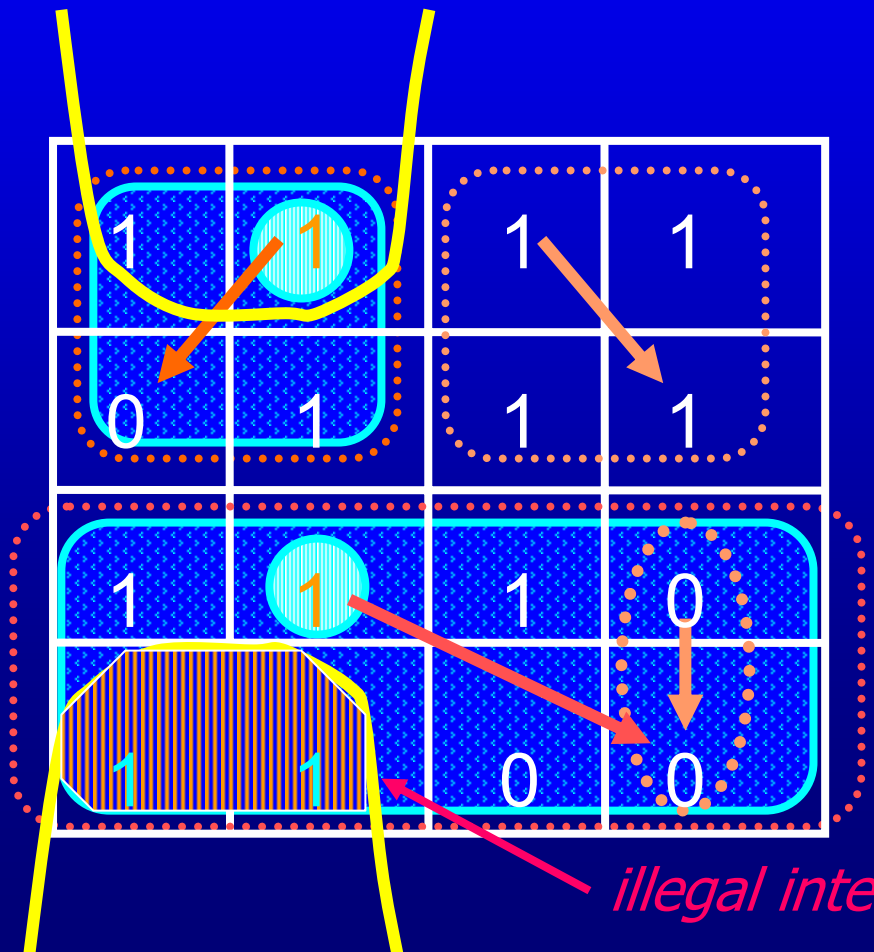


All illegal intersections eliminated

KEEP: new DHF-Prime

2-Level Hazard-Free Logic Minimization: a Complete Example

Step 1: Generate All DHF-Prime Implicants



Approach: 2 steps
- Generate All Prime Implicants
- Reduce to DHF-Prime Implicants

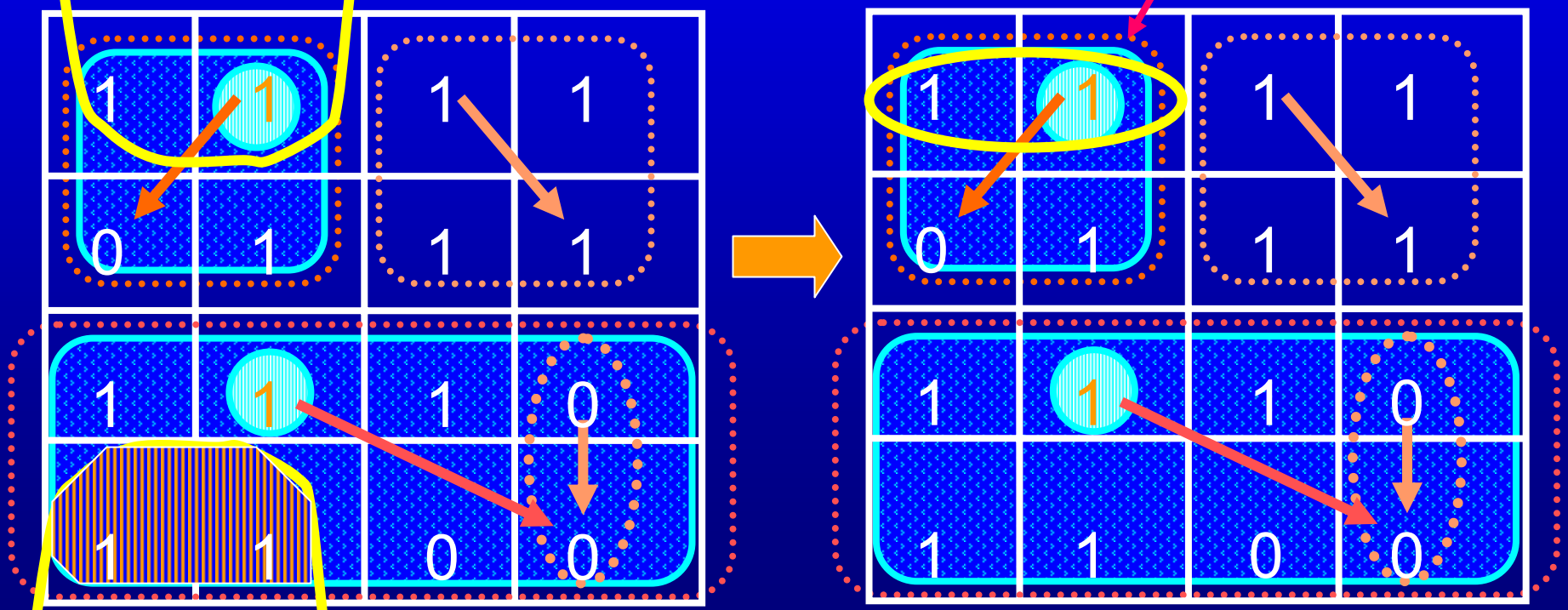
Other primes have illegal intersections
=> they must be reduced

2-Level Hazard-Free Logic Minimization: a Complete Example

Step 1: Generate All DHF-Prime Implicants

First reduction of prime implicant

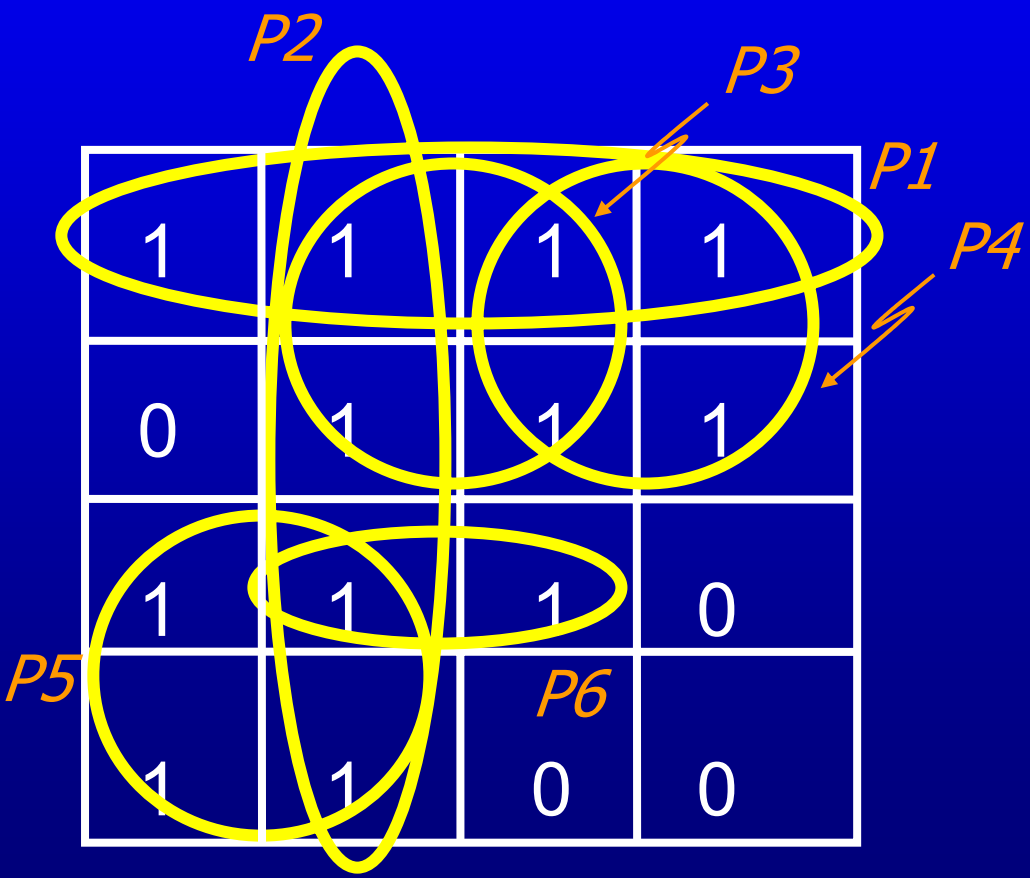
DISCARD: contained in a DHF-Prime



All illegal intersections eliminated

2-Level Hazard-Free Logic Minimization: a Complete Example

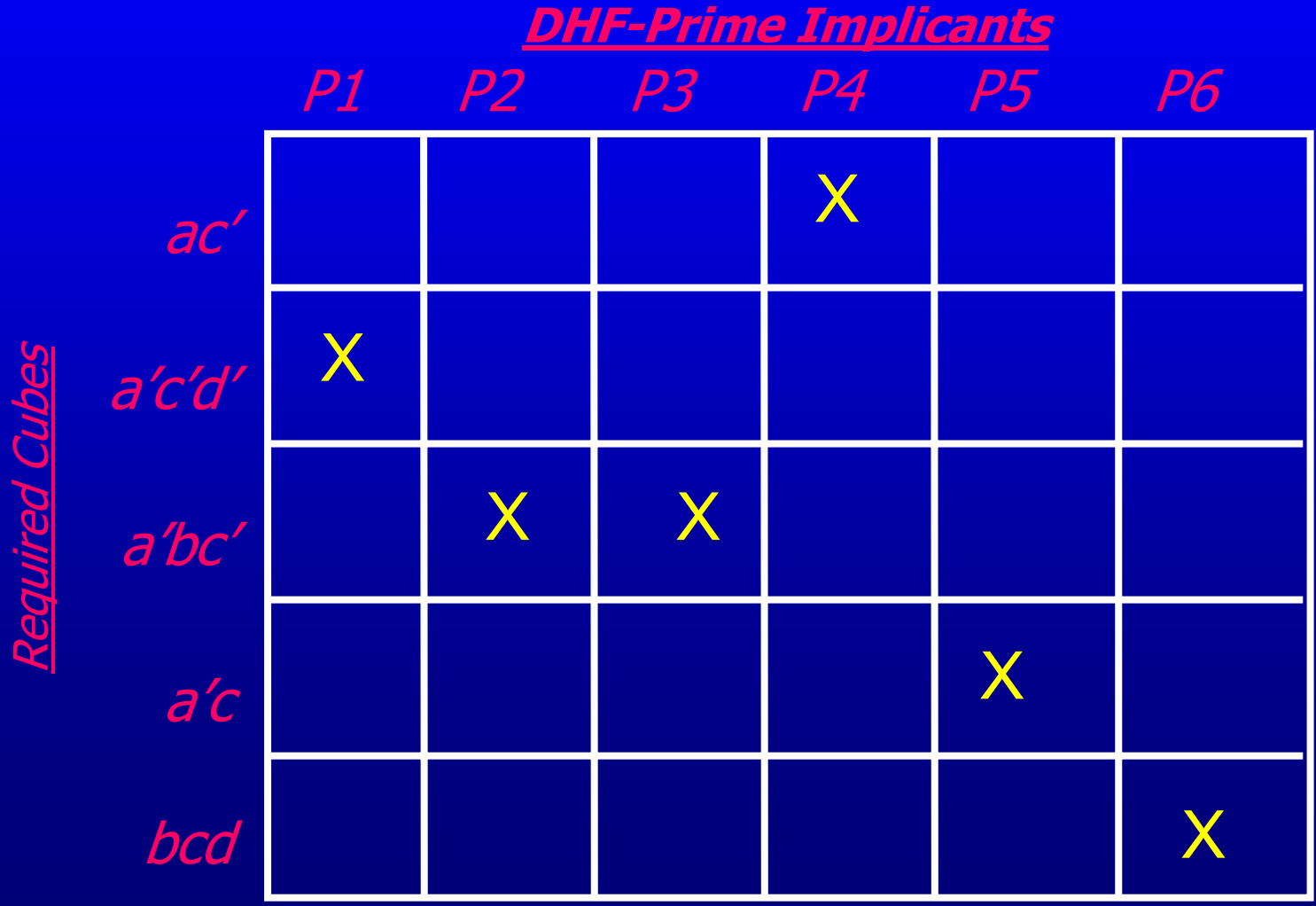
Step 1: <u>Generate All DHF-Prime Implicants</u>



<u>Final Result:</u>
6 DHF-Prime Implicants

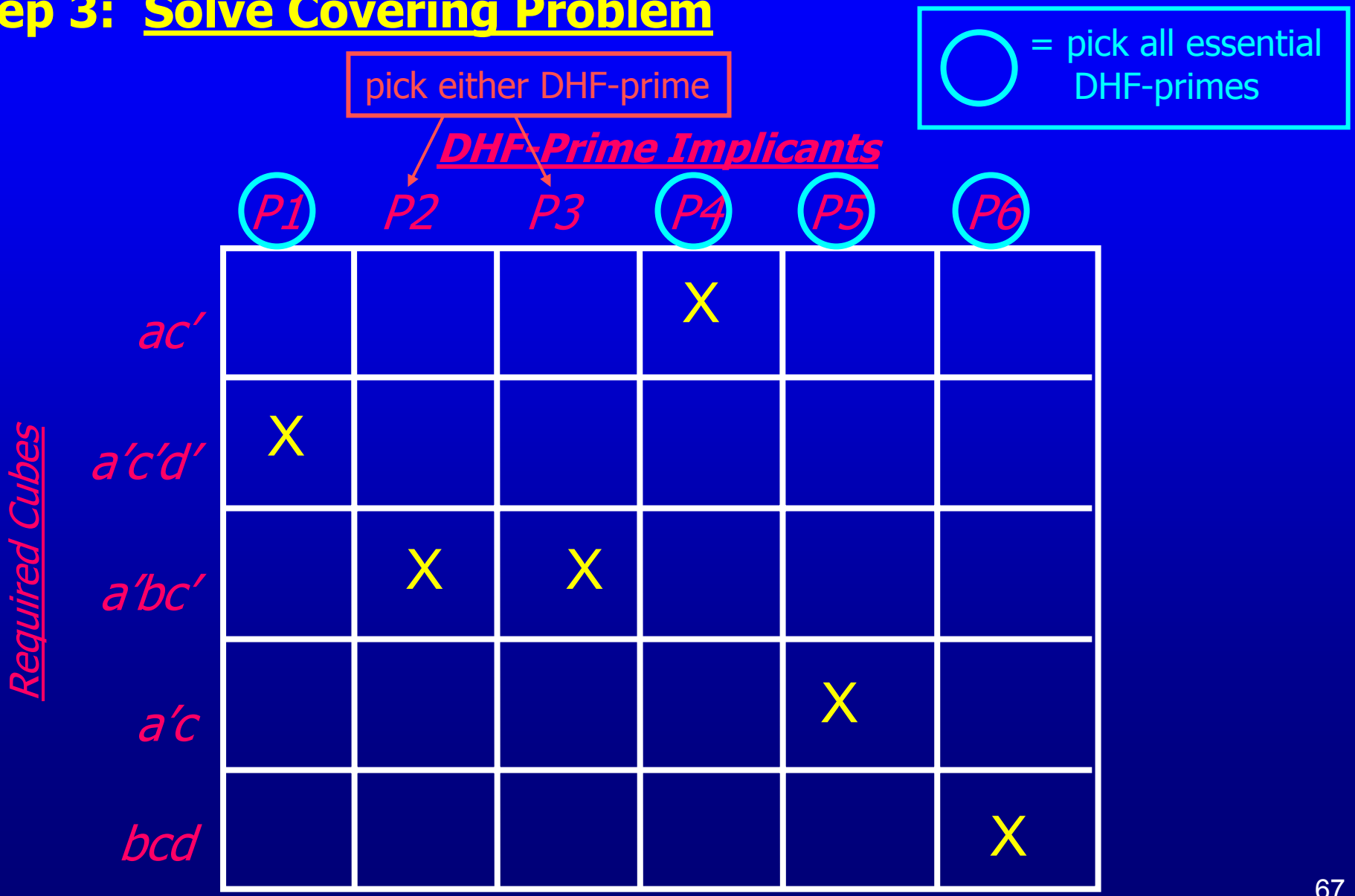
2-Level Hazard-Free Logic Minimization: a Complete Example

Step 2: Generate DHF-Prime Implicant Table



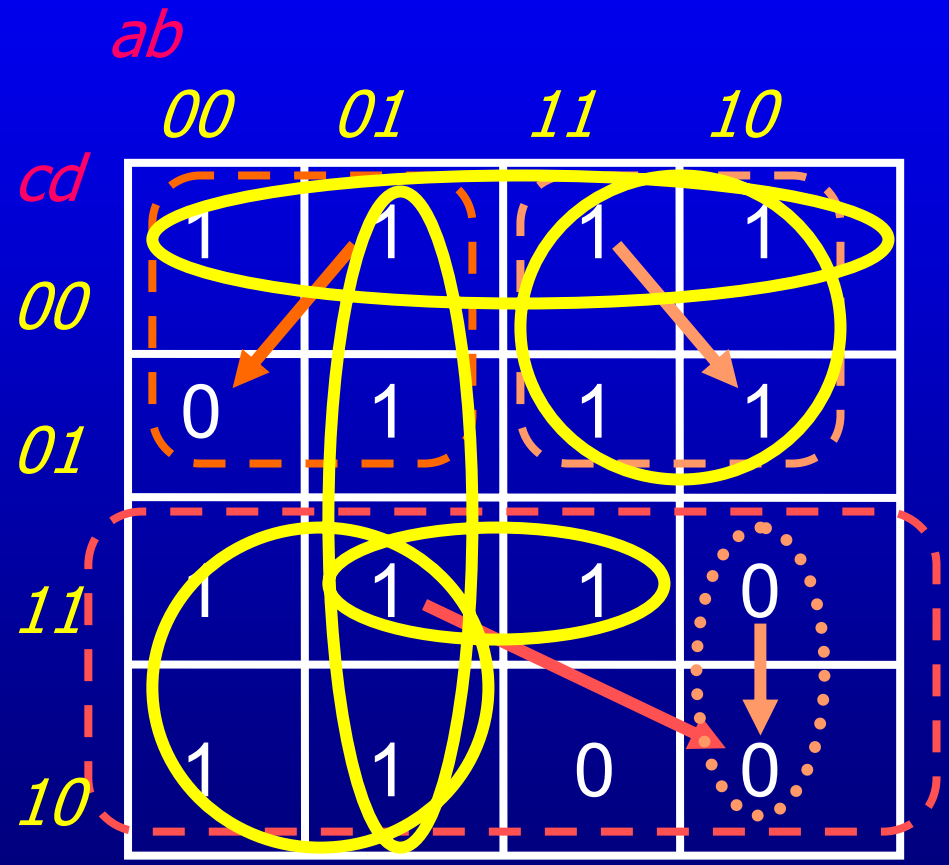
2-Level Hazard-Free Logic Minimization: a Complete Example

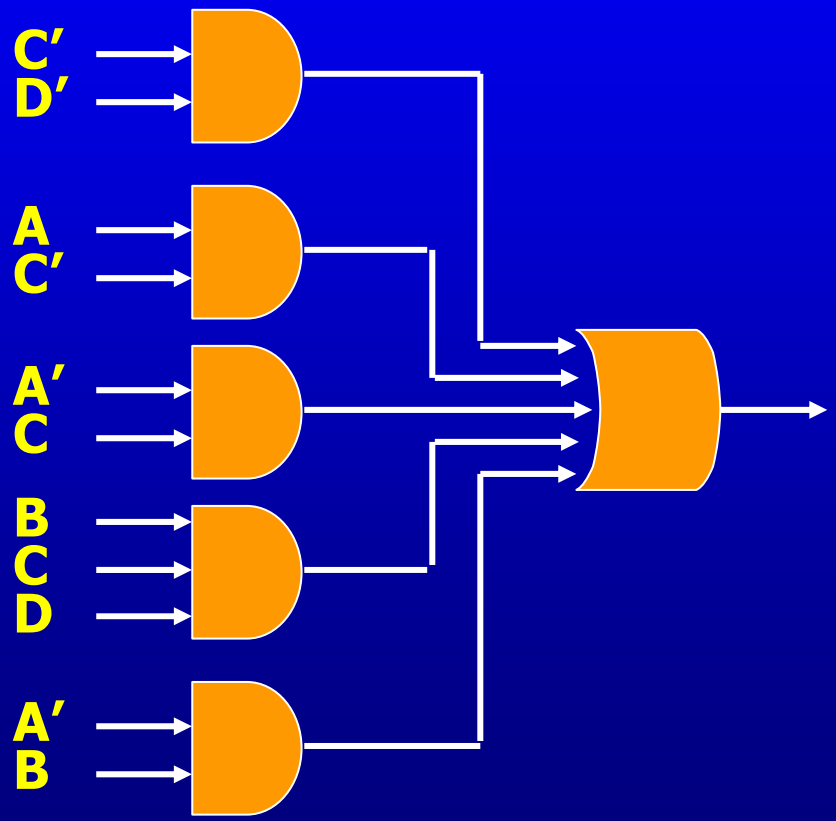
Step 3: <u>Solve Covering Problem</u>



2-Level Hazard-Free Logic Minimization: a Complete Example

Final Hazard-Free Circuit (minimum-cost):

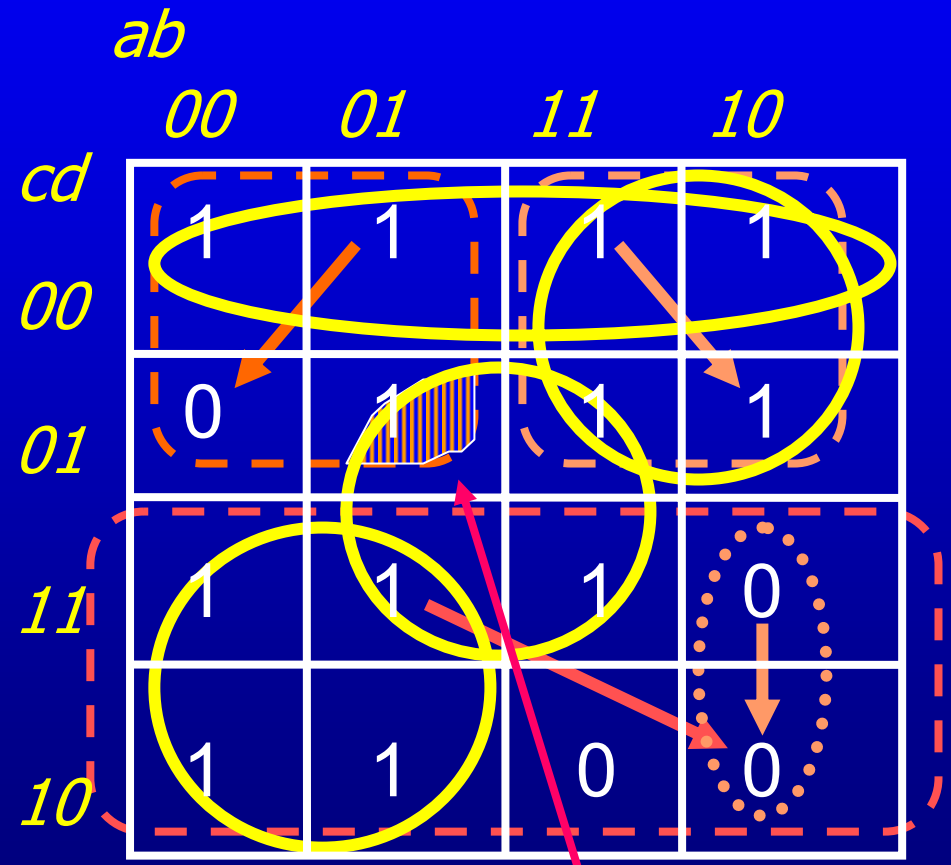


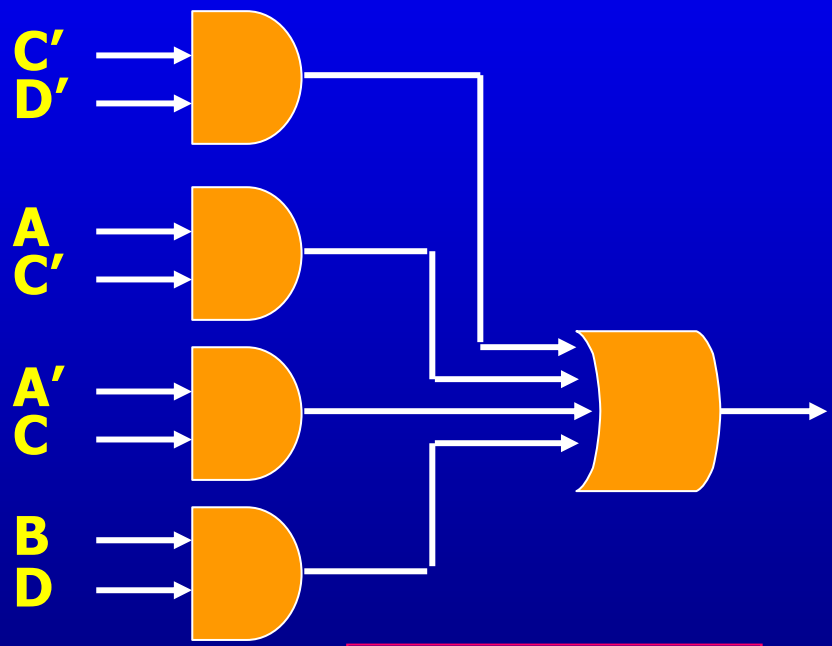


5 DHF-Prime Implicants

2-Level Hazard-Free Logic Minimization: a Complete Example

Final *Non-Hazard-Free* Circuit (minimum-cost):

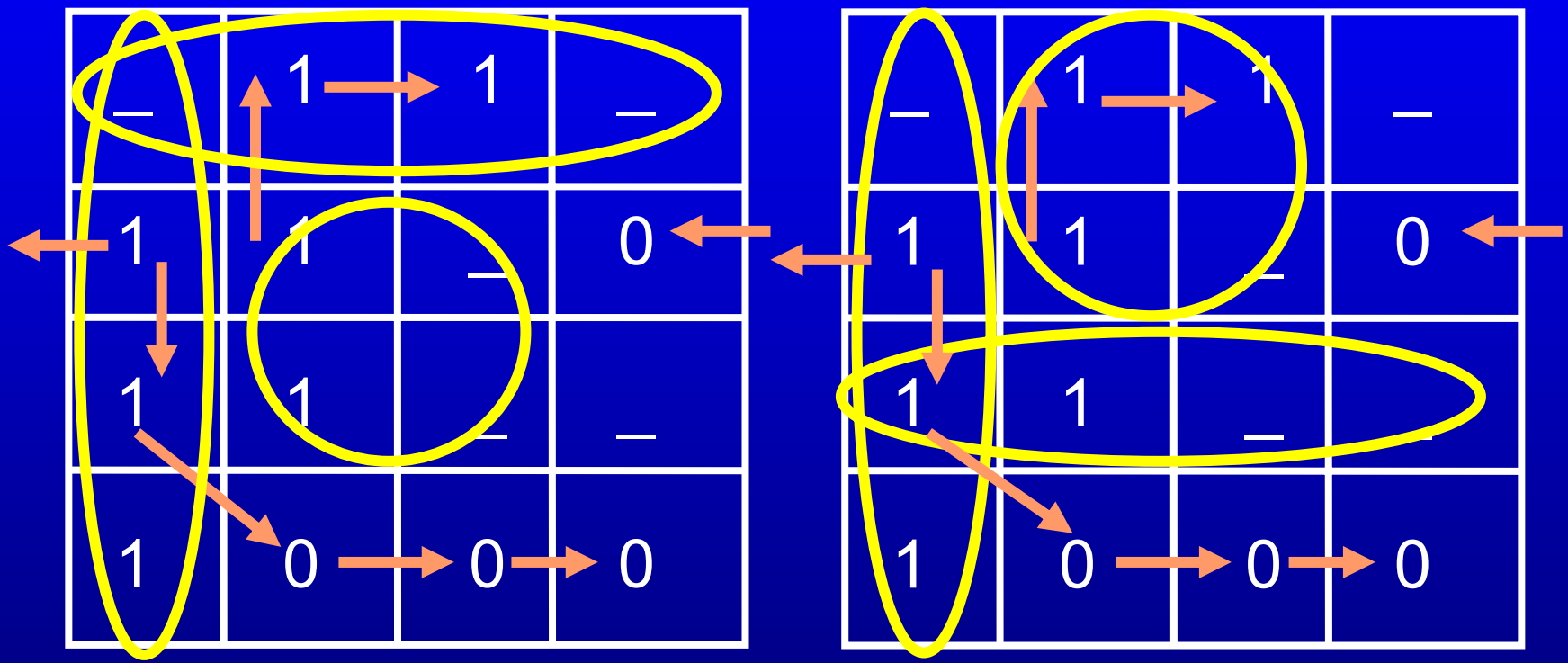




4 Prime Implicants

2-Level Hazard-Free Logic Minimization: Another Example

same # of products

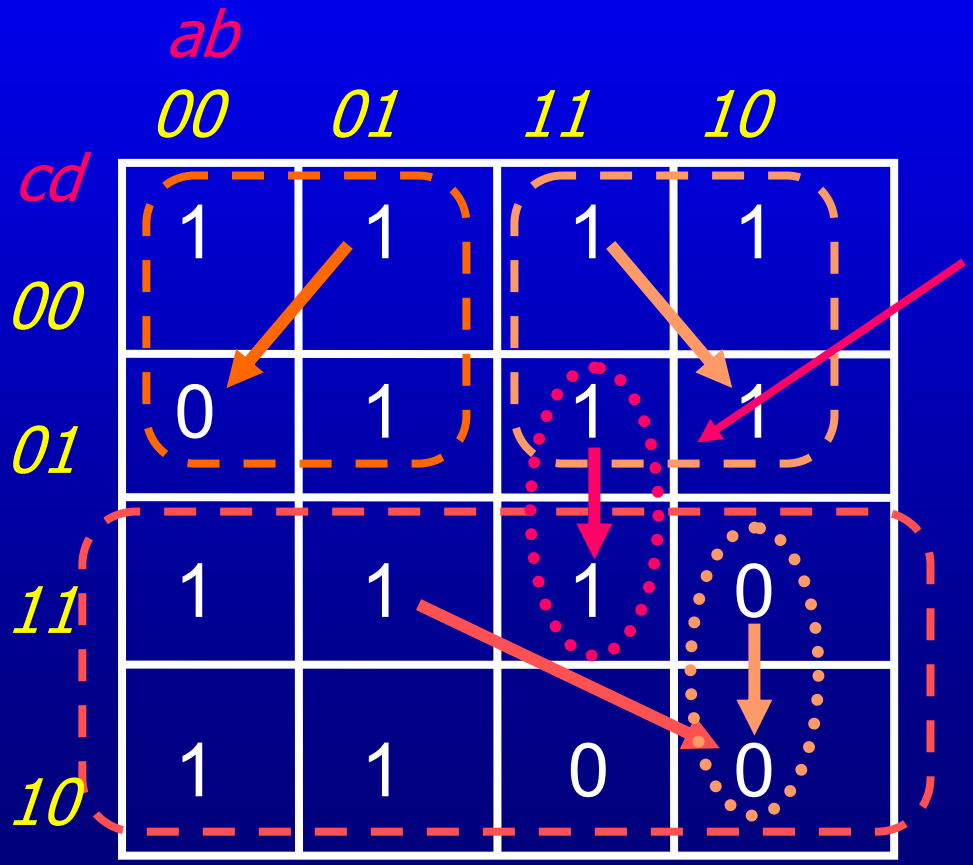


Non-hazard-free:
min cost = 3 products

Hazard-free:
min cost = 3 products

Existence of a Hazard-Free Solution

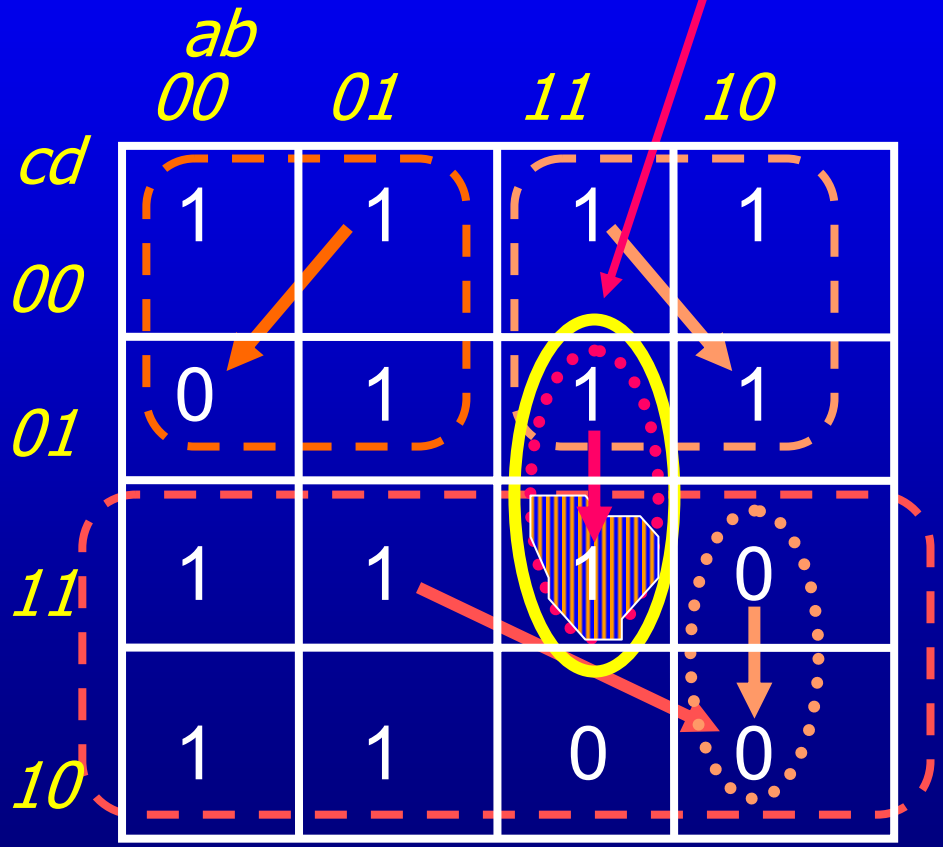
Challenge: a hazard-free 2-level implementation does not always exist!

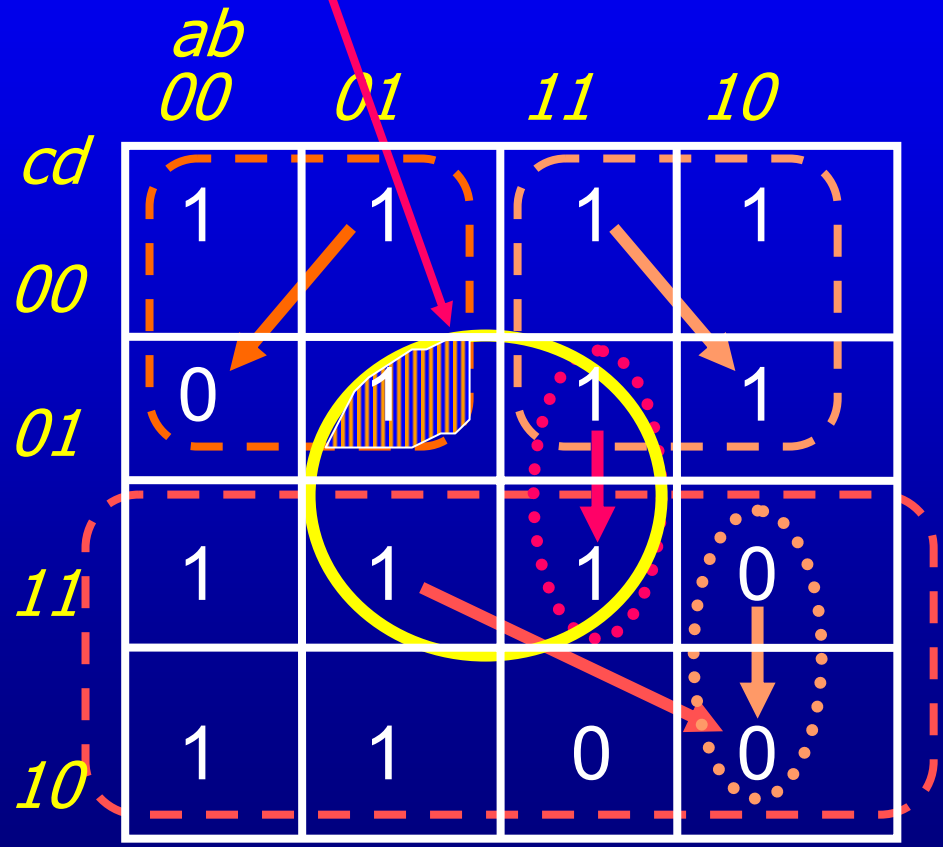


New Example: add 1 more input transition

Existence of a Hazard-Free Solution

Every implicant containing the new required cube also has an *illegal intersection*!

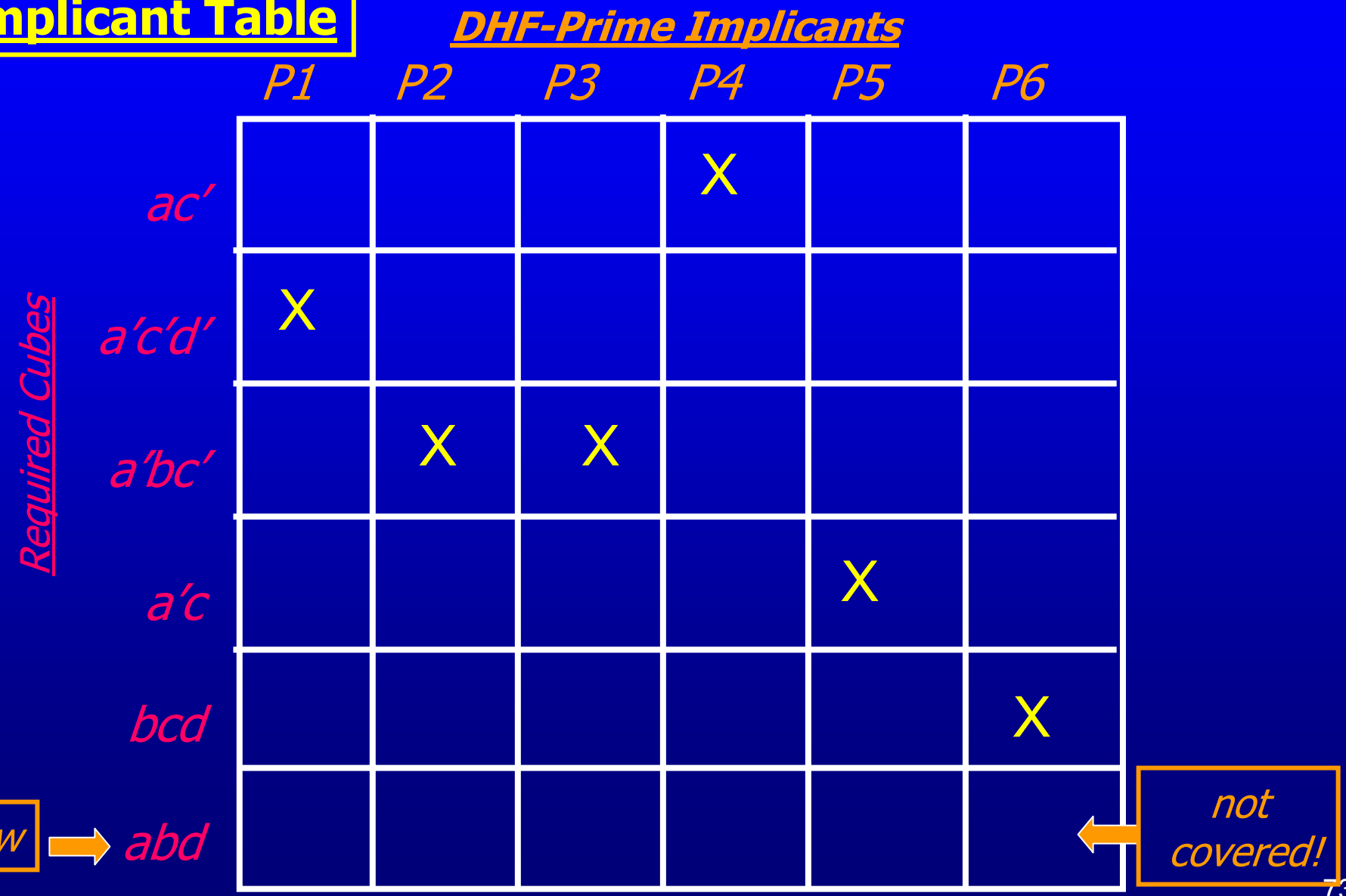




No hazard-free 2-level implementation exists

Existence of a Hazard-Free Solution

DHF-Prime Implicant Table



Existence of a Hazard-Free Solution

Conclusion:

- asynchronous sequential synthesis methods:

  must produce functions *for which hazard-free implementations exist!*

Burst-Mode Synthesis Methods: impose constraints on

- state minimization
- state assignment

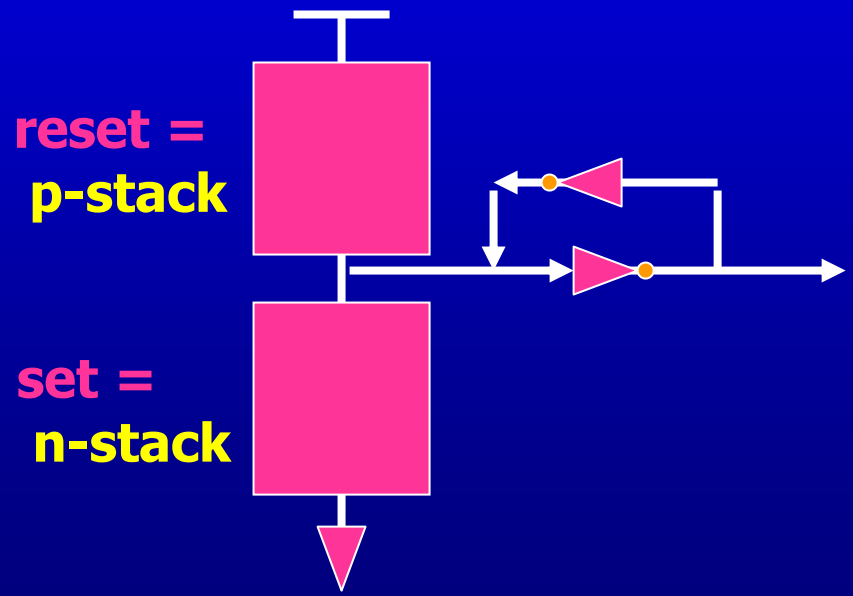
... to generate Boolean functions where all logic hazards can *always* be eliminated

Always *guarantee* a hazard-free solution exists

An Alternative Approach: using "Generalized C-Elements"

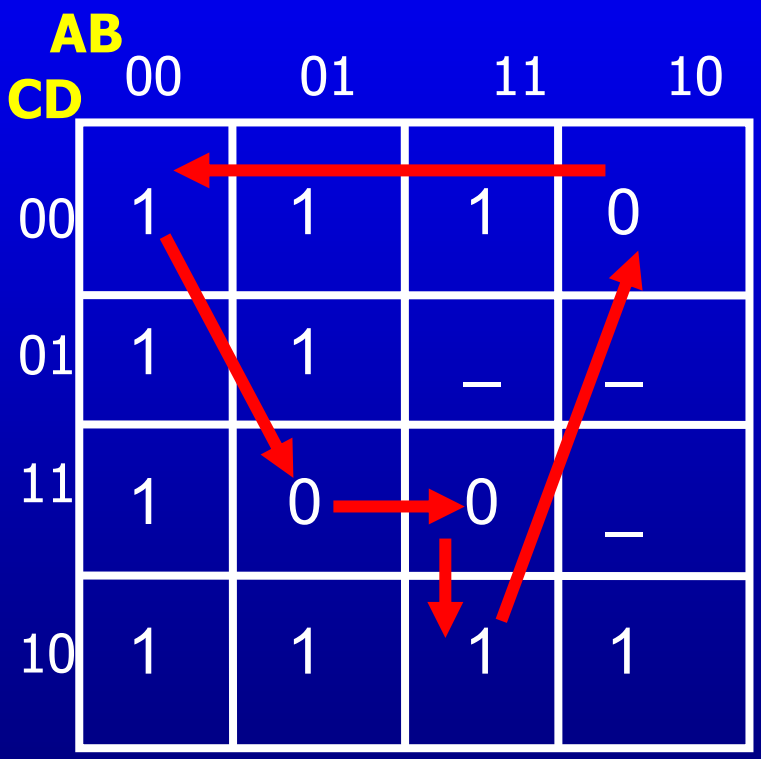
Alternative to 2-Level Logic

Target = "Generalized C-element (*GC*)":

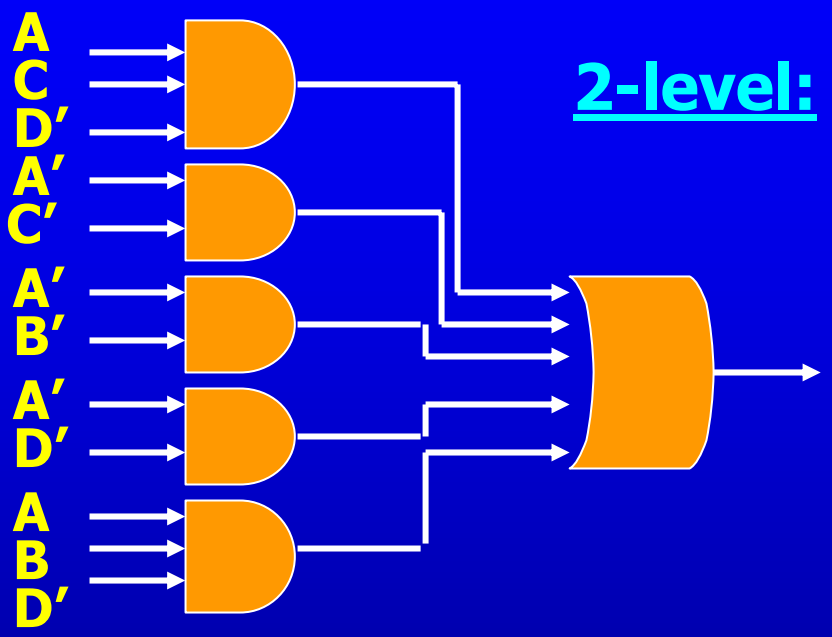


Hazard-Free Logic Using GC-Elements

gC-Based Mapping: an Example

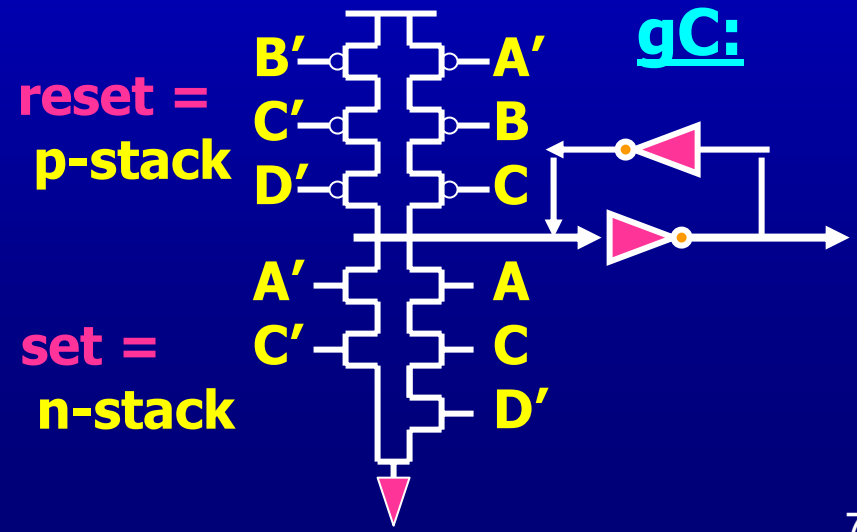


Function



2-level:

gC:



A Reading List

Hazard Basics:

- S. Unger, *Asynchronous Sequential Switching Circuits,* Wiley Interscience, 1969
- J. Beister, "A Unified Approach to Combinational Hazards", IEEE Transactions on Computers, vol C-23, no. 6, 1974
- S.M. Nowick, *Automatic Synthesis of Burst-Mode Asynchronous Controllers,* PhD Thesis, Stanford University, March 1993 (revised technical report, Stanford Computer Systems Lab CSL-TR-95-686, Dec. 1995).
- S.M. Nowick and D.L. Dill, "Exact Two-Level Minimization of Hazard-Free Logic with Multiple-Input Changes", IEEE Transactions on Computer-Aided Design, vol. 14, pp. 986-997, August 1995

Two-Level Hazard-Free Logic Minimization:

Basic Method: (first complete solution) exact hazard-free minimization

- S.M. Nowick and D.L. Dill, "Exact Two-Level Minimization of Hazard-Free Logic with Multiple-Input Changes", IEEE Transactions on Computer-Aided Design, vol. 14, pp. 986-997, August 1995

A Reading List (cont.)

Two-Level Hazard-Free Logic Minimization (cont.):

HFMIN: binary & symbolic (exact) hazard-free minimization

- R.M. Fuhrer and S.M. Nowick, *Sequential Optimization of Asynchronous and Synchronous Finite-State Machines: Algorithms and Tools.* Kluwer Academic, 2001.

Recent Methods: Exact Solutions

- "IMPYMIN": M. Theobald and S.M. Nowick, "Fast Heuristic and Exact Algorithms for Two-Level Hazard-Free Logic Minimization", IEEE Transactions on Computer-Aided Design, vol. 17, pp. 1130-1147, November 1998
- C. Myers and H. Jacobson, "Efficient Exact Two-Level Hazard-Free Logic Minimization", Async-01 Symposium (IEEE Int. Symp. On Advanced Rsrch. In Asynchronous Circuits and Systems), pp. 64-73, March 2001
- J. Rutten, M. Berkelaar, et al., "An Efficient Divide and Conquer Algorithm for Exact Hazard-Free Logic Minimization", Design, Automation and Test in Europe Conference (DATE), pp. 749-754, February 1998.

Recent Methods: Heuristic Solutions

- "ESPRESSO-HF": M. Theobald and S.M. Nowick, "Fast Heuristic and Exact Algorithms for Two-Level Hazard-Free Logic Minimization", IEEE Transactions on Computer-Aided Design, vol. 17, pp. 1130-1147, November 1998

Part II

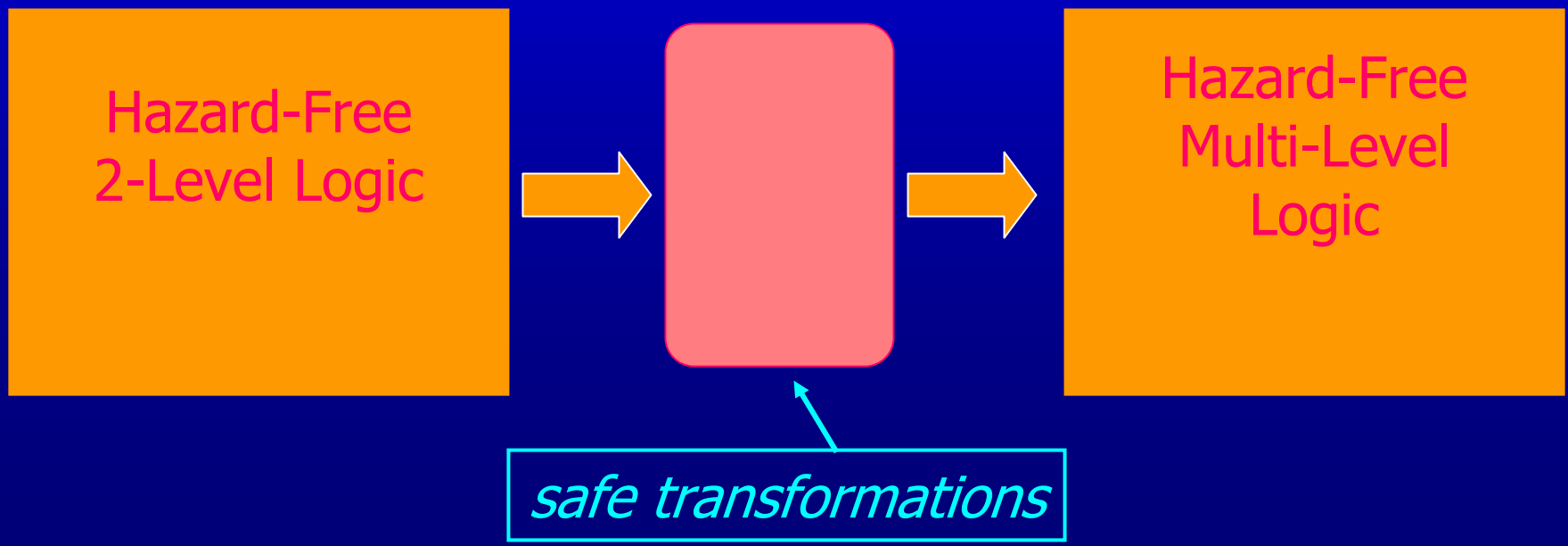
Multi-Level Logic and Technology Mapping

Goal: Hazard-Free Multi-Level Logic

Strategy

Start with: hazard-free 2-level logic

Apply: *hazard-non-increasing* multi-level transformations



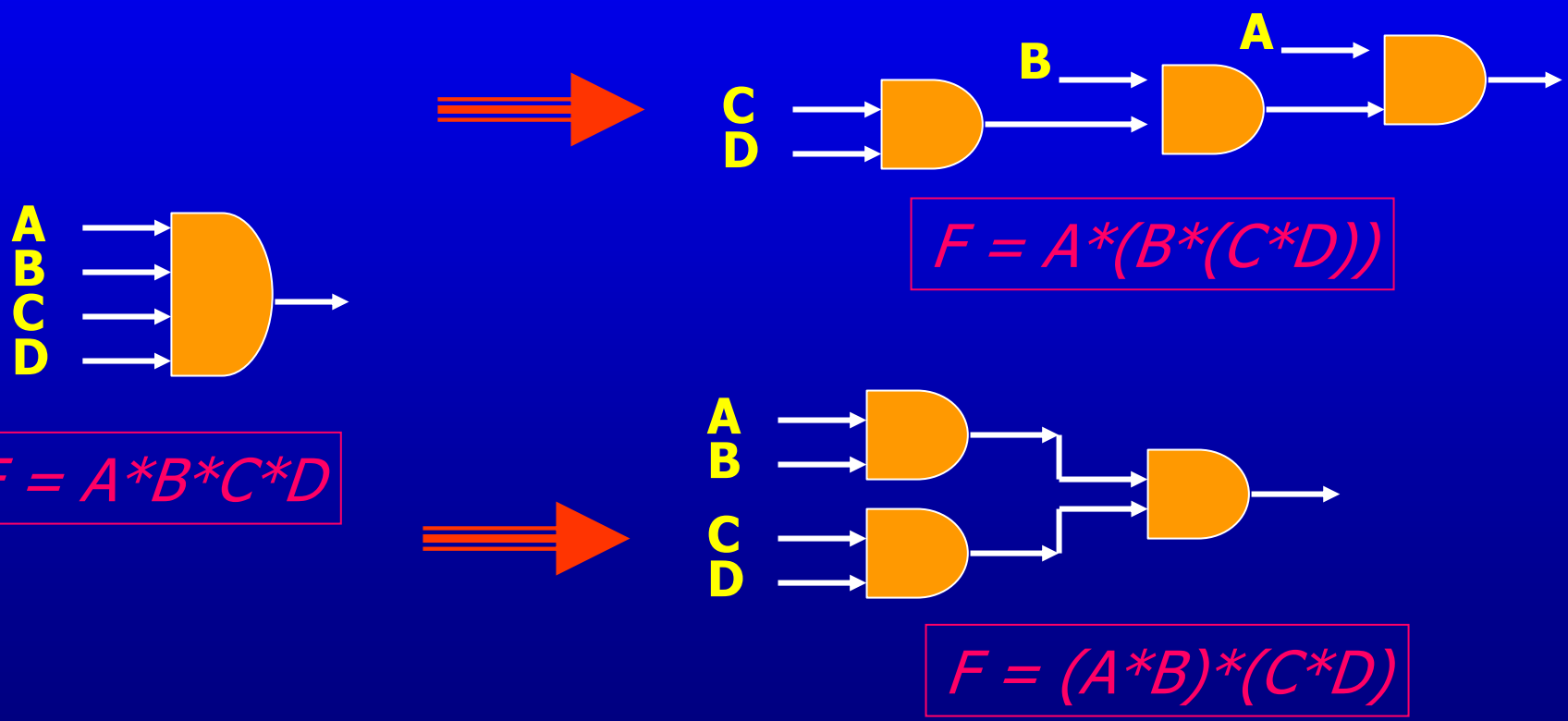
Hazard-Non-Increasing Multi-Level Transforms

A Large Menu of "Safe Transforms": [Unger, Kung]

- Associative Law
- Factoring
- DeMorgan's Law
- … Many others:
  - Kernel & Cube Factoring
  - Dual Global Flow
  - Double Inversion
  - Tree Decomposition of a Gate

Associative Law (1)

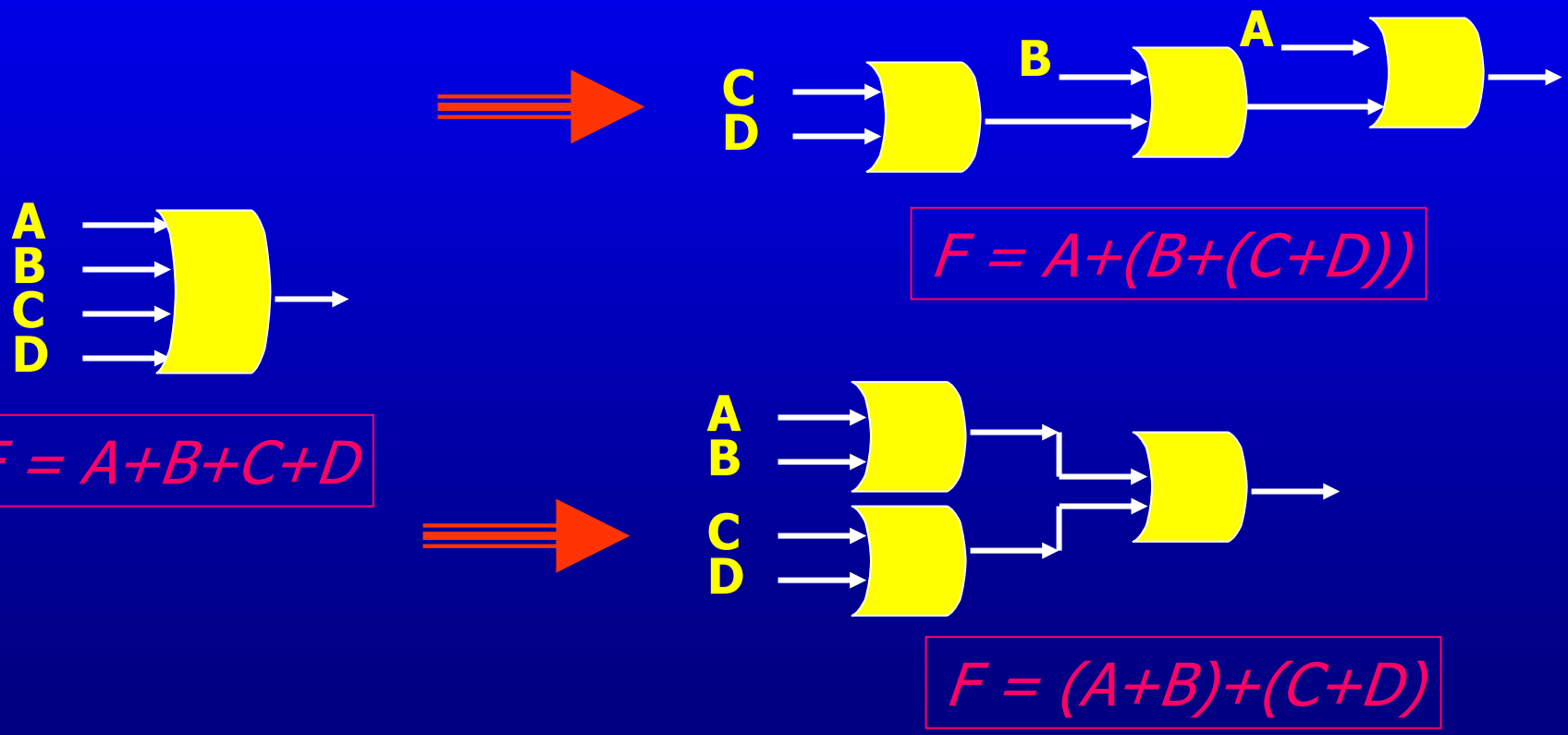
Example: *decomposing large fan-in gates*



This transform may introduce hazards in 'speed-independent' or 'QDI' circuits.

Associative Law (2)

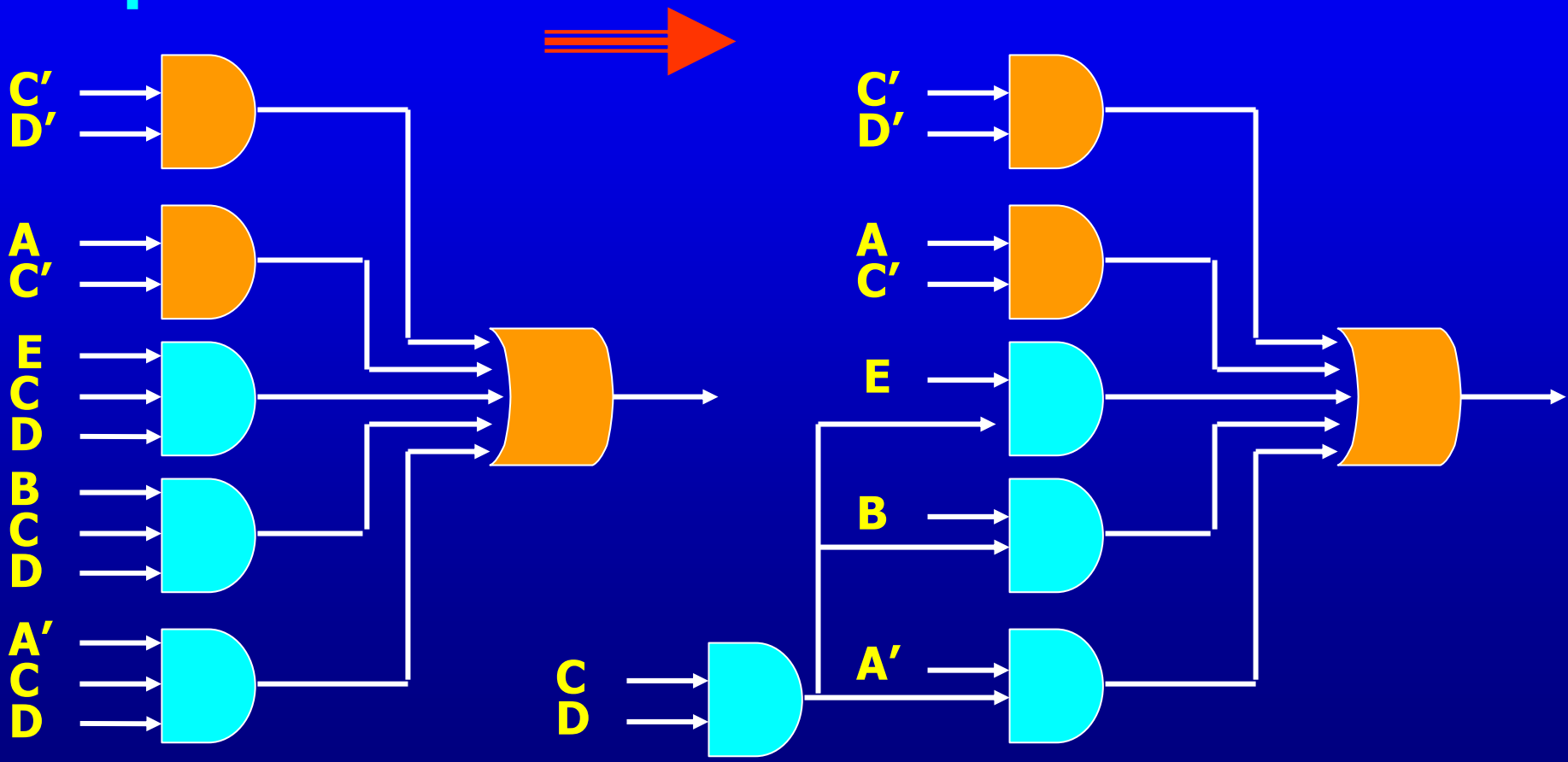
Example: *decomposing large fan-in gates*



This transform may introduce hazards in 'speed-independent' or 'QDI' circuits.

Factoring

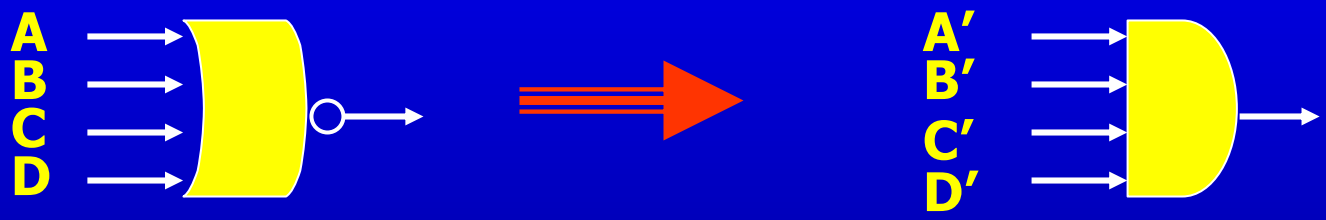
Example:



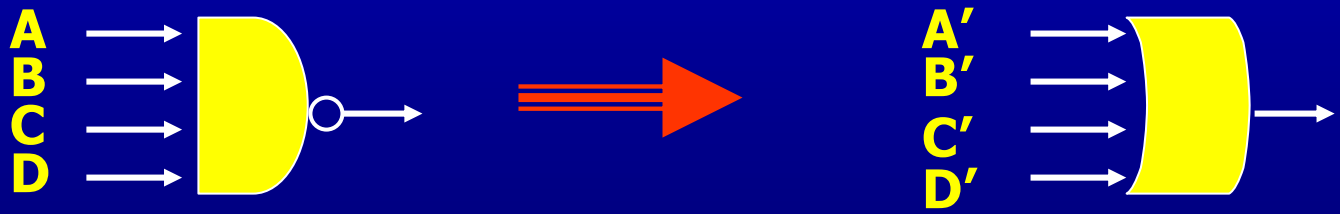
DeMorgan's Law

Example:

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

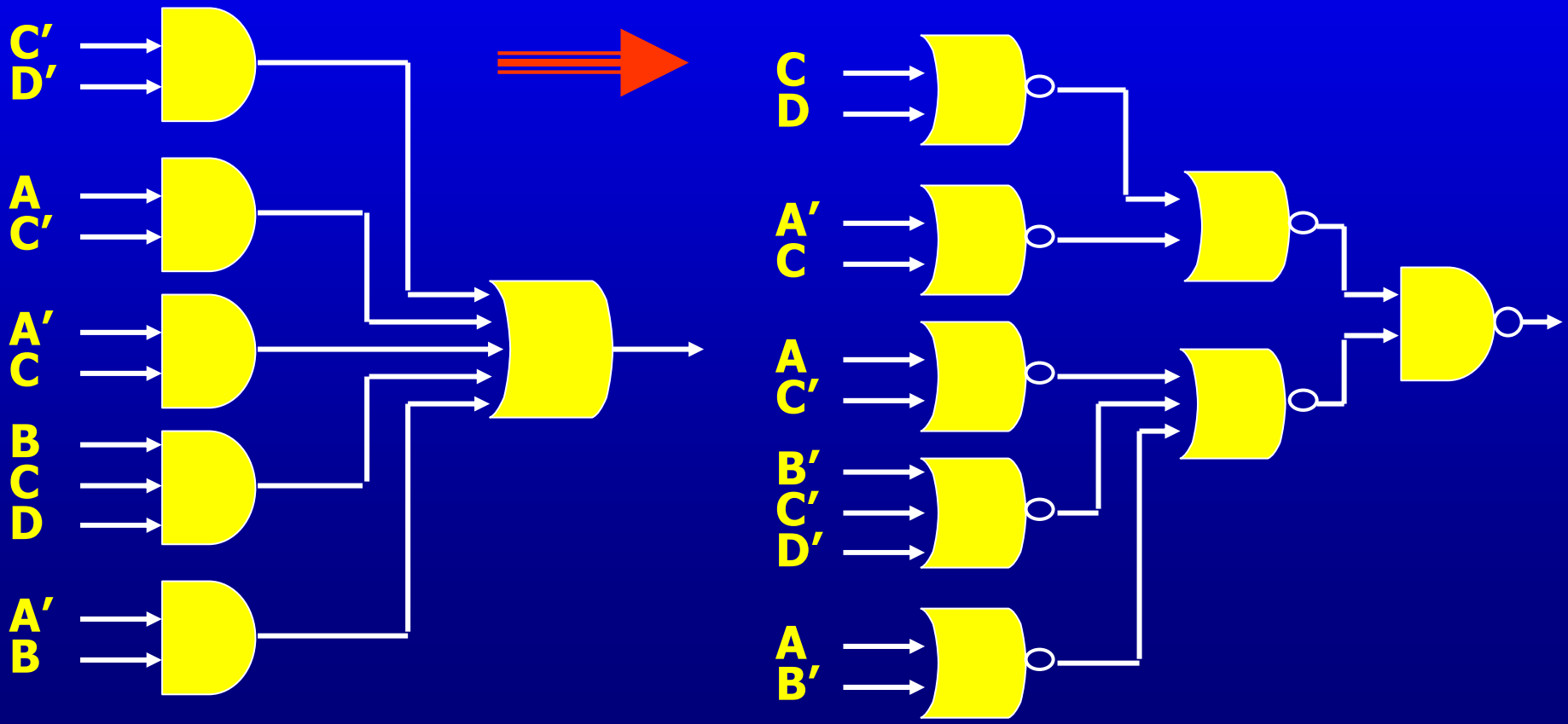


$(A*B*C*D)' = A'+B'+C'+D'$



Allows replacement of AND/OR gates by NAND (NOR) gates

Example: 2-Level Circuit (cont.)



Summary

Hazard-Non-Increasing Transforms:

- Allow hazard-free decomposition into simple gates *(always!)*
- Wide & flexible range of safe transforms:
  - much overlap with 'scripts' of Synopsys Design Compiler
- Less restrictive than *QDI* or speed-independent transforms:
  - many safe "fundamental mode" multi-level transforms fail with QDI
    [e.g. associative law]

Hazard-Free Technology Mapping

1. Basic approach:
   - Siegel, De Micheli [DAC'93]
2. For improved "average-case performance":
   - basic: Beerel et al. [Async'96]
   - transistor-level optimization: James, Yun [Async'98]
3. For complex CMOS gates:
   - Kudva et al. [DAC'96]

A Reading List

Hazard-Free Multi-Level Logic:

- S. Unger, *Asynchronous Sequential Switching Circuits,* Wiley Interscience, 1969
- D. Kung, "Hazard-non-increasing gate-level optimization algorithms", IEEE International Conference on CAD (ICCAD), pp. 631-634, Nov. 1992
- B. Lin and S. Devadas, "Synthesis of Hazard-Free Multi-Level Logic Under Multiple-Input Changes from Binary Decision Diagrams", IEEE Transactions on Computer-Aided Design, vol. 14:8, pp. 974-985, August 1995

Hazard-Free Technology Mapping:

- P. Siegel, G. De Micheli, and D. Dill, "Automatic Technology Mapping for Generalized Fundamental Mode Asynchronous Designs," IEEE Design Automation Conference (DAC), pp. 61-67, June 1993
- P.A. Beerel, K.Y. Yun, and W.C. Chou, "Optimizing Average-Case Delay in Technology Mapping of Burst-Mode Circuits", Async Symposium (IEEE Intl. Symposium on Advanced Research in Async. Circuits and Systems), PP. 244-259, March 1996.

A Reading List (cont.)

Hazard-Free Technology Mapping (cont.):

- K. James and K.Y. Yun, "Average-case optimized transistor-level technology mapping of extended burst-mode circuits", Async Symposium (IEEE Intl. Symposium on Advanced Research in Async. Circuits and Systems), PP. 70-79, April 1998.
- P. Kudva, G. Gopalakrishnan, H. Jacobson, and S. Nowick, "Synthesis of Hazard-Free Customized CMOS Complex-Gate Networks Under Multiple-Input Changes", IEEE Design Automation Conference (DAC), pp. 77-82, June 1996