Karnaugh Maps

□ Objectives

This section presents a technique for simplifying logical expressions. It will:

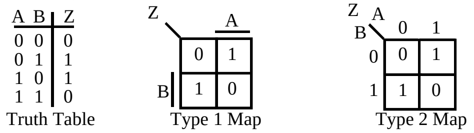
- Define Karnaugh and establish the correspondence between Karnaugh maps and truth tables and logical expressions.
- Show how to use Karnaugh maps to derive minimal sum-of-products and product-of-sums expressions.
- Introduce the concept of "don't care" entries and show how to extend Karnaugh map techniques to include maps with don't care entries.

□ Reading Assignment

- Sections 2.6 and 2.7 from the text

Karnaugh Map Definitions

□ A Karnaugh map is a two-dimensional truth-table. Unlike ordinary (*i.e.*, one-dimensional) truth tables, however, certain logical network simplifications can be easily recognized from a Karnaugh map.



A	B	Z
0	0	0
0	1	1
1	0	1
1	1	0

Truth Table

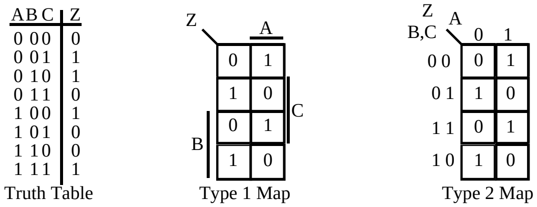
Z		A
	0	1
B	1	0

Type 1 Map

Z B\A	0	1
0	0	1
1	1	0

Type 2 Map

Two-Variable Maps



A B C	Z
0 0 0	0
0 0 1	1
0 1 0	1
0 1 1	0
1 0 0	1
1 0 1	0
1 1 0	0
1 1 1	1

Truth Table

Z		A	
	0	1	
	1	0	C
B	0	1	C
B	1	0	

Type 1 Map

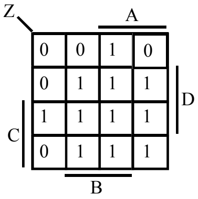
Z B,C\A	0	1
0 0	0	1
0 1	1	0
1 1	0	1
1 0	1	0

Type 2 Map

Three-Variable Maps

A	B	C	D	Z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Truth Table



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

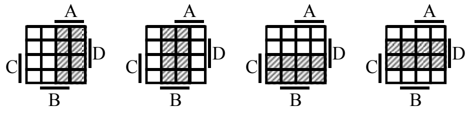
Type 1 Map

Z A,B C,D	00	01	11	10
00	0	0	1	0
01	0	1	1	1
11	1	1	1	1
10	0	1	1	1

Type 2 Map

Four-Variable Maps

- □ The interpretation of a type 1 map is that the rows or columns labeled with a variable correspond to region of the map where that variable has value 1.



A = 1 region　B = 1 region　C = 1 region　D = 1 region

- □ Numbering of Karnaugh Map Squares.

Z A B	0	1
0	0	2
1	1	3

Z A B,C	0	1
00	0	4
01	1	5
11	3	7
10	2	6

Z A,B C,D	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

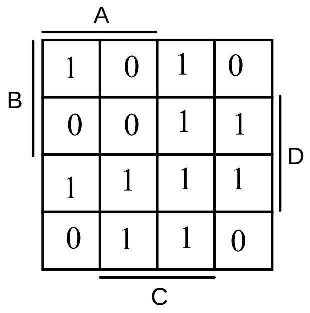
- □ Exercise: Plot the following expression on a Karnaugh map.

$$Z = (A \bullet B) \oplus (C+D)$$

$Z = A \bullet B \bullet (C+D)' + (A \bullet B)' \bullet (C+D)$

$\quad = A \bullet B \bullet C' \bullet D' + (A'+B') \bullet (C+D)$

$\quad = A \bullet B \bullet C' \bullet D' + A' \bullet C + A' \bullet D + B' \bullet C + B' \bullet D$

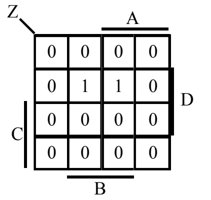


1	0	1	0
0	0	1	1
1	1	1	1
0	1	1	0

Minimal Sum-Of-Products Expressions

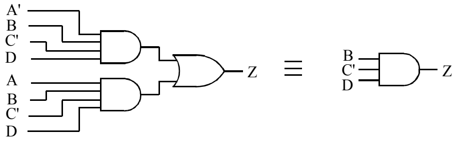
- □ Ordering of Squares
  - ■ The important feature of the ordering of squares is that the squares are numbered so that the binary representations for the numbers of two adjacent squares differ in exactly one position.
    - ◆ This is due to the use of a Gray code (one in which adjacent numbers differ in only one position) to label the edges of a type 2 map.
    - ◆ The labels for the type 1 map must be chosen to guarantee this property.
  - ■ Note that squares at opposite ends of the same row or column also have this property (i.e., their associated numbers differ in exactly one position).

□ Merging Adjacent Product Terms

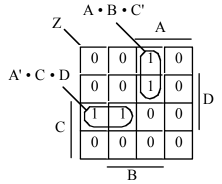


0	0	0	0
0	1	1	0
0	0	0	0
0	0	0	0

$$\begin{aligned} Z &= m_5 + m_{13} \\ &= A'\bullet B\bullet C'\bullet D + A\bullet B\bullet C'\bullet D \\ &= (A'+A)\bullet B\bullet C'\bullet D \\ &= 1\bullet B\bullet C'\bullet D \\ &= B\bullet C'\bullet D \end{aligned}$$

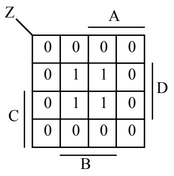


- Example



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

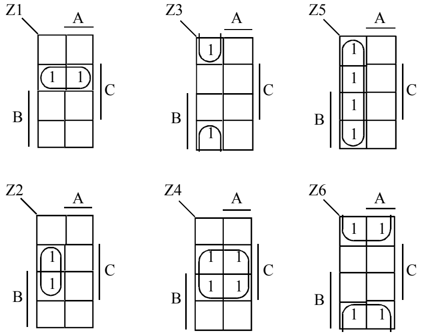
□ For k-variable maps, this reduction technique can also be applied to groupings of 4,8,16,...,2k rectangles all of whose binary numbers agree in (k-2),(k-3),(k-4),...,0 positions, respectively.



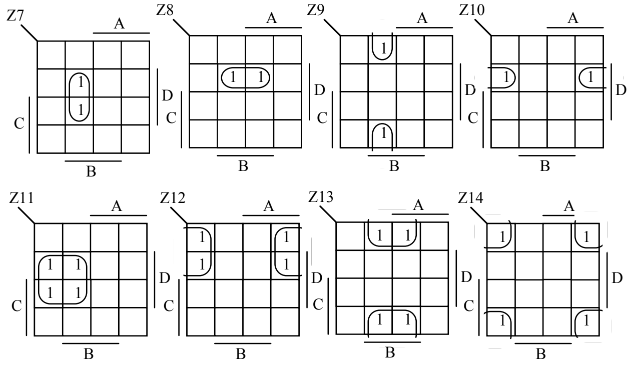
0	0	0	0
0	1	1	0
0	1	1	0
0	0	0	0

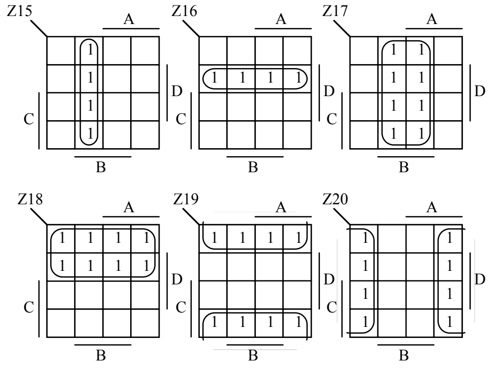
$$\begin{aligned} Z = {} & m_5 + m_7 + m_{13} + m_{15} \\ = {} & A'\bullet B\bullet C'\bullet D + A'\bullet B\bullet C\bullet D + A\bullet B\bullet C'\bullet D + A\bullet B\bullet C\bullet D \\ = {} & (A'\bullet C' + A'\bullet C + A\bullet C' + A\bullet C) \bullet (B\bullet D) \\ = {} & (A'\bullet(C' + C) + A\bullet(C' + C)) \bullet (B\bullet D) \\ = {} & (A' + A) \bullet (B\bullet D) \\ = {} & B\bullet D \end{aligned}$$

□Basic Karnaugh Map Groupings for Three-Variable Maps.



□Basic Karnaugh Map Groupings for Four-Variable Maps.





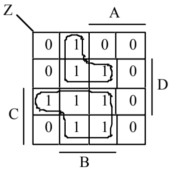
□ Rules for Grouping:

- ■ The number of squares in a grouping is 2^i for some i such that $1 \leq i \leq k$.
- ■ There are exactly k-i variables that have constant value for all squares in the grouping.

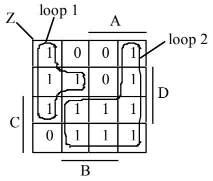
□ Resulting Product Terms:

- ■ If X is a variable that has value 0 in **all** of the squares in the grouping, then the literal X' is in the product term.
- ■ If X is a variable that has value 1 in all of the squares in the grouping, then the literal X is in the product term.
- ■ If X is a variable that has value 0 for some squares in the grouping and value 1 for others, then neither X' nor X are in the product term.

□ Invalid Karnaugh Map Groupings.



Violates Rule 1

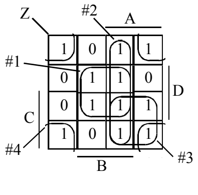


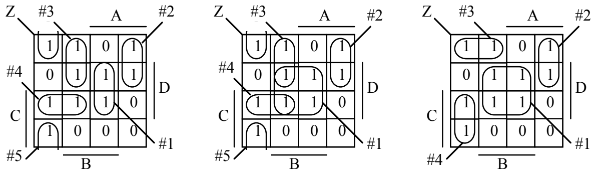
Violates Rule 2

□ In order to minimize the resulting logical expression, the groupings should be selected as follows:

- Identify those groupings that are maximal in the sense that they are not contained in any other possible grouping. The product terms obtained from such groupings are called *prime implicants*.
  - ◆A *distinguished 1-cell* is a cell that is covered by only one prime implicant.
  - ◆An *essential prime implicant* is one that covers a distiquished 1-cell.
- Use the fewest possible number of maximal groupings needed to cover all of the squares marked with a 1.

□ Examples:





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

a. Non-minimal Product Terms

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

b. Non-minimal Selection of Prime Implicants

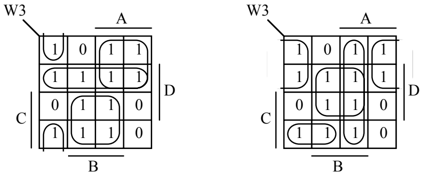
1	1	0	1
0	1	1	1
1	1	1	0
1	0	0	0

c. Minimal Grouping

a. Z = A•B•D+A•B'•C'+A'•B•C'+A'•C•D+A'•B'•D'

b. Z = B•D+A•B'•C'+A'•B•C'+A'•C•D+A'•B'•D'

c. Z = B•D+A•B'•C'+A'•C'•D'+A'•B'•C



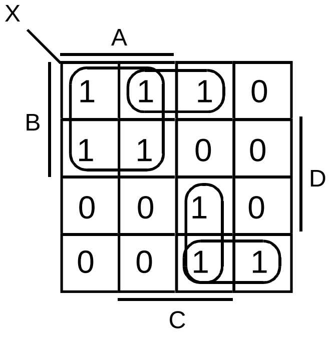
1	0	1	1
1	1	1	1
0	1	1	0
1	1	1	0

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

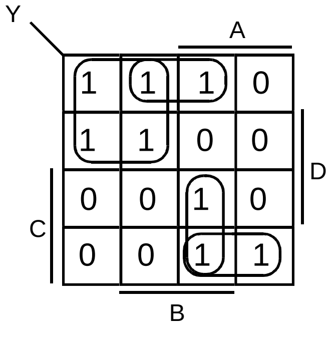
a. W3 = C'D+AC'+BC+A'B'D'

b. W3 = BD+AB+B'C'+A'CD'

□ Exercise: Derive minimal sum-of-products logical expressions from the following Karnaugh maps.



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



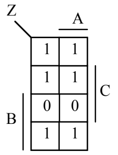
1	1	1	0
1	1	0	0
0	0	1	0
0	0	1	1

X = A•B + A'•B'•D' + A'•B'•C + B•C•D'

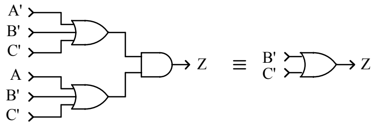
Y = A'•C' + A•C•D' + A•B•C + B•C'•D'

Minimal Product-Of-Sums Expressions

□ Merging Adjacent Product Terms.



Karnaugh Map



OR-AND Networks

$$\begin{aligned} M3 \bullet M7 &= (A+B'+C') \bullet (A'+B'+C') \\ &= (A \bullet A')+(B'+C') \\ &= 0 + (B'+C') \\ &= B' + C' \end{aligned}$$

□ Rules for Grouping:

- Same as for sum-of-products, except that zero's are grouped instead of ones.

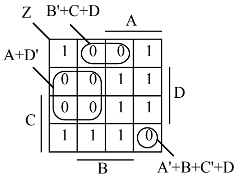
□ Resulting Sum Terms:

- If variable X has value 0 for **all** squares in the group, then the literal X is in the sum term.
- If variable X has value 1 for **all** squares in the group, then the literal X' is in the sum term.
- If variable X has value 0 for some squares in the group and value 1 for the others, then that variable does not appear in the sum term.

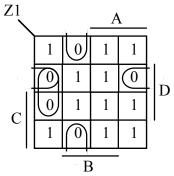
□ Prime Implicate:

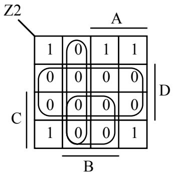
- Maximal grouping of zeros.

□ Examples:



Z = (A+D')•(B'+C+D)•(A'+B+C'+D)

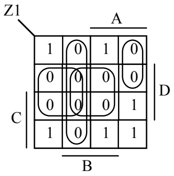


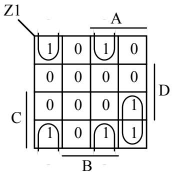


Z1 = (A+B'+D)•(B+C+D')•(A+B+D')

Z2 = D'•(A+B')•(B'+C')

□ Comparison of Sum-of-Products and Product-of-Sums Expressions.

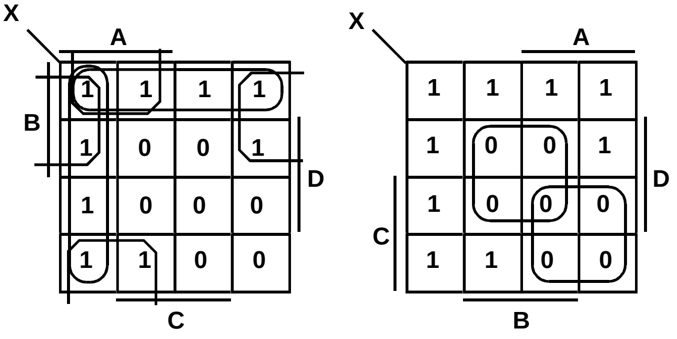




Z1 = (A+B')•(A+D')•(B'+D')•(A'+B+C)

Z1 = (A•B•D') + (A•B'•C) + (A'•B'•D')

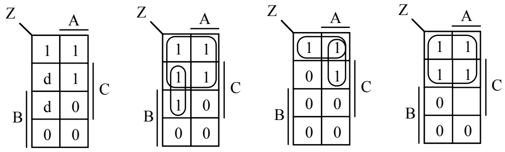
□ Exercise: Derive minimal SOP and POS expressions from the following Karnaugh maps.



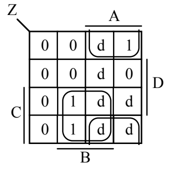
X = B•D' + A•C' + A•D' + B•C'   Y = (B' + D') • (A' + C')

Don't Care Entries

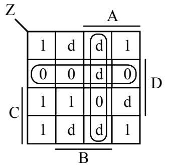
□ Assignment of Values to Don't Care Entries.



Z = B'+A'•C   Z = A•B'+B'•C'   Z = B'



Z = B•C + A•D'



Z = (C+D')•(A'+B')

□ Minterm and Maxterm Lists with Don't Care Entries.

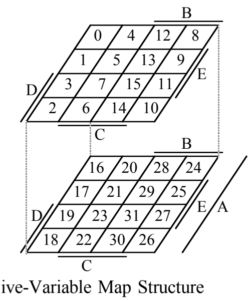
$\Sigma_{A,B,C}(0,d1,d3,4,5)$ and $\Pi_{A,B,C}(d1,2,d3,6,7)$

$\Sigma_{A,B,C}(6,7,8,d10,d11,d12,d13,d14,d15)$

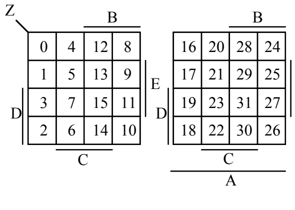
$\Pi_{A,B,C}(0,1,2,3,4,5,9,d10,d11,d12,d13,d14,d15)$

Five- and Six-Variable Maps

□ Five-Variable Maps.



Five-Variable Map Structure

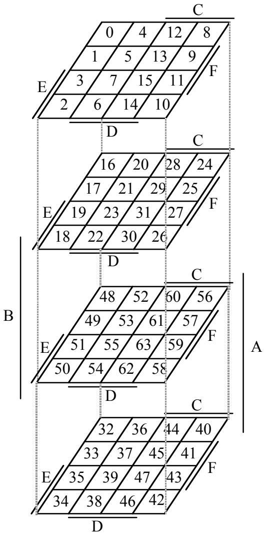


0	4	12	8
1	5	13	9
3	7	15	11
2	6	14	10

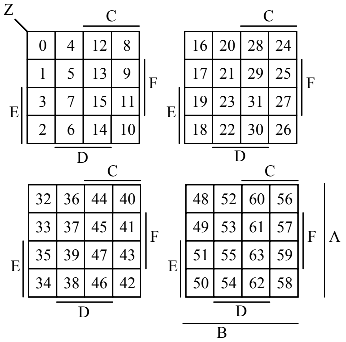
16	20	28	24
17	21	29	25
19	23	31	27
18	22	30	26

Alternate Version of Five-Variable Map

□ Six-Variable Maps



Six Variable Map Structure



0	4	12	8
1	5	13	9
3	7	15	11
2	6	14	10

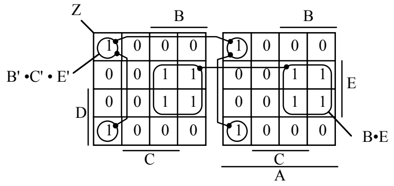
16	20	28	24
17	21	29	25
19	23	31	27
18	22	30	26

32	36	44	40
33	37	45	41
35	39	47	43
34	38	46	42

48	52	60	56
49	53	61	57
51	55	63	59
50	54	62	58

Alternate Version of Six-Variable Map

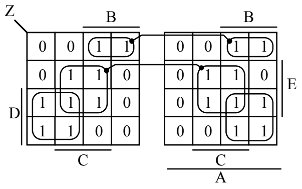
□ Examples.



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

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

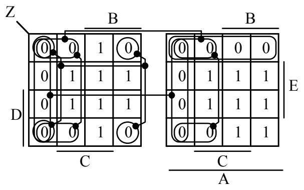
Z = B'•C'•E'+B•E



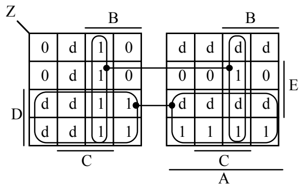
0	0	1	1
0	1	1	0
1	1	1	0
1	1	0	0

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

A'B'D + CE + ABD + BD'E'



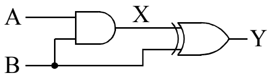
Z = (B+C)•(B+E•(A'+D+E)•(A+C+E)

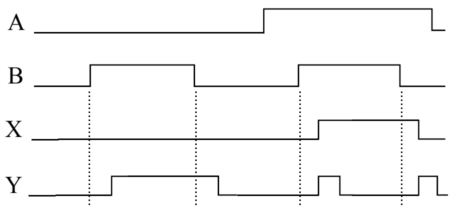


Z = B•C + D

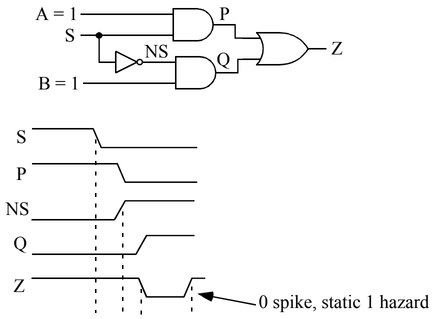
Timing Hazards

□ Exercise: Draw a timing diagram for the following circuit assuming an equal unit delay for each gate.



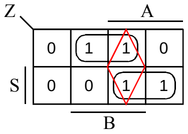


□ Static 1 hazards



- Without the delay in the gates, the output Z would be constant at 1.
- The spike is due to the gate delay and the fact that not every path from on input to the output is the same length.

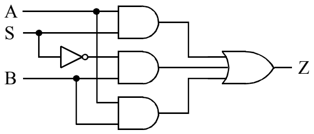
□ There is a simple way to prevent the hazards using Karnaugh maps.



0	1	1	0
0	0	1	1

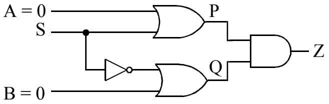
- The hazard occurs when the output is 1 before and after the transition.
- It happens because one of the AND gates is holding the output 1 before the transition and the other AND gate holds it 1 after the transition. The spike occurs if it is possible for the first to turn off before the second turns on.
- This can happen anytime there are to adjacent squares on the Karnaugh map that are both 1 and not both covered by a common loop. The hazard occurs in making a transition from one square to the other.
- The way to prevent it is to put in an extra product term that is 1 on both sides of the transition. On the Karnaugh map this is a loop covering the two adjacent one squares.

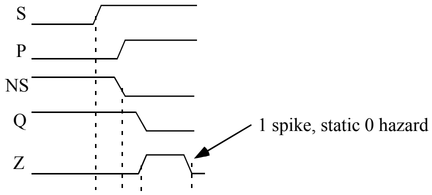
- The loop added to prevent the hazard is redundant and not needed to realize the logical expression. Its only purpose is to prevent the hazard.



- This technique will avoid all static 1 hazards if only one variable is changed at a time.
- It is not possible to get a 1 spike from a 2-level AND-OR network by changing only one variable. Why?

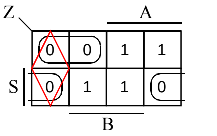
□ Static 0 hazard



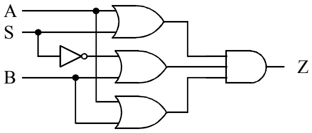


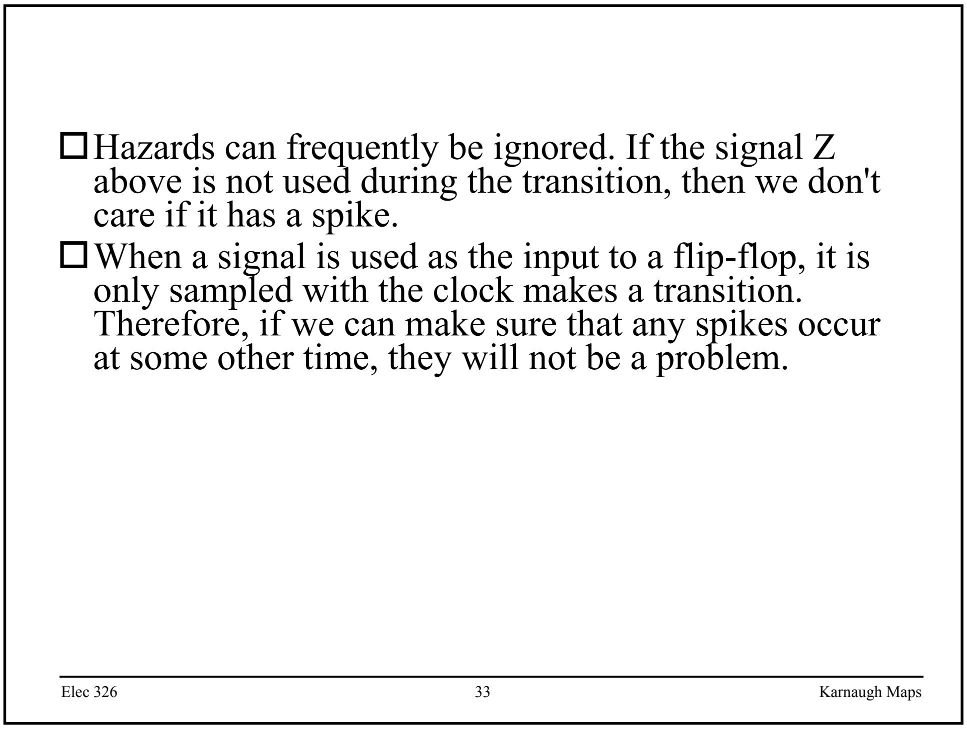
1 spike, static 0 hazard

- The static 0 hazard is caused by two OR gates where one holds the output zero before the transition and the other after the transition, and it is possible for them to both be 1 during the transition
- The fix is to put in an extra OR gate to hold the output zero during the transition



	0	0	1	1
S	0	1	1	0





- □ Hazards can frequently be ignored. If the signal Z above is not used during the transition, then we don't care if it has a spike.
- □ When a signal is used as the input to a flip-flop, it is only sampled with the clock makes a transition. Therefore, if we can make sure that any spikes occur at some other time, they will not be a problem.