Lecture 8

- Logistics
  - Midterm 1 week from today in class. Closed book/closed notes
  - Sample midterm linked in on calendar
  - Review session Thursday Jan 29, 4:30. Location TBA
- Last lecture
  - Verilog
- Last last lecture
  - K-map examples
  - K-map high dimension example
  - Don't cares
- Today
  - Don't care (review)
  - POS minimization with K-map
  - Design examples with K-map
  - "Switching-network" logic blocks (multiplexers/demultiplexers)

The "WHY" slide

- Don't cares
  - Sometimes the logic output doesn't matter. When we don't care if the output is 0 or 1, rather than assigning random outputs, it is best to denote it as "Don't care." If you learn how to use the "don't care's", you will be able to build even more efficient circuits than without them.
- Design examples with K-map
  - Doing K-map is fun, but when it is combined with an actual design problem you will see how k-map fits into the whole scheme of logic design.
- "Switching network" blocks (multiplexers)
  - Routing method for data paths helps to structure logic design

Revisit Don't cares example: Truth table for a BCD increment-by-1

INPUTS				OUTPUTS			
A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

- Function F computes the next number in a BCD sequence

  If the input is 0010_2, the output is 0011_2

- BCD encodes decimal digits 0–9 as 0000_2–1001_2

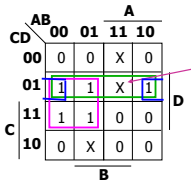
  Don't care about binary numbers 1010_2–1111_2

Notation

- Don't cares in canonical forms
  - Three distinct logical sets: {on}, {off}, {don't care}
- Canonical representations of a BCD increment-by-1
  - Minterm expansion
    - ⇨ W = m7+m8+d10+d11+d12+d13+d14+d15
      = Σm(7,8) + d(10,11,12,13,14,15)
  - Maxterm expansion
    - ⇨ W = M0•M1•M2•M3•M4•M5•M6•M9•D10•D11•D12•D13•D14•D15
      = ΠM(0,1,2,3,4,5,6,9) • D(10,11,12,13,14,15)
- In K-maps, can treat 'don't cares' as 0s or 1s
  - Depending on which is more advantageous

Example: with don't cares

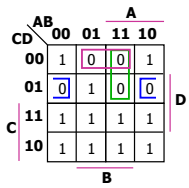
- F(A,B,C,D) = Σm(1,3,5,7,9) + d(6,12,13)
  - F = A'D + B'C'D *without using* don't cares
  - F = A'D + C'D *using* don't cares



Assign X == "1"
⇒ allows a 2-cube rather than a 1-cube

POS minimization using k-maps

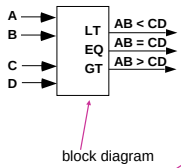
- Using k-maps for POS minimization
  - Encircle the zeros in the map
  - Interpret indices complementary to SOP form



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

Same idea as the Truth Table

Design example: a two-bit comparator



A	B	C	D	LT	EQ	GT
0	0	0	0	0	1	0
		0	1	1	0	0
		1	0	1	0	0
		1	1	1	0	0
0	1	0	0	0	0	1
		0	1	0	1	0
		1	0	1	0	0
		1	1	1	0	0
1	0	0	0	0	0	1
		0	1	0	0	1
		1	0	0	1	0
		1	1	1	0	0
1	1	0	0	0	0	1
		0	1	0	0	1
		1	0	0	0	1
		1	1	0	1	0

Need a 4 -variable Karnaugh map for each of the 3 output functions

Design example: a two-bit comparator (con't)

K-map for LT

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

K-map for EQ

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

K-map for GT

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

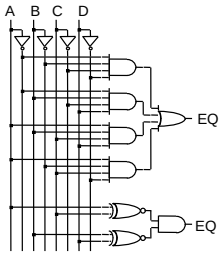
LT = A'B'D+A'C+B'CD

GT = BC'D'+AC'+ABD'

EQ = A'B'C'D'+A'BC'D+ABCD+AB'CD'
= (A xnor C)•(B xnor D)

Design example: a two-bit comparator (con't)

- Two ways to implement EQ:

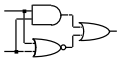


Option 1:
EQ = A'B'C'D'+A'BC'D+ABCD+AB'CD'

5 gates but they require lots of inputs

Option 2
EQ = (A xnor C) •(B xnor D)

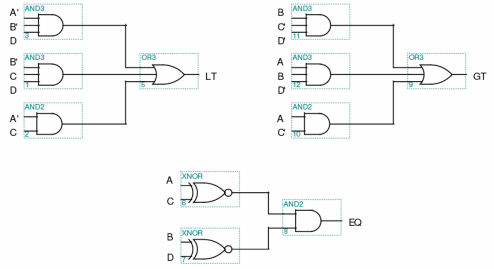
XNOR is constructed from 3 simple gates



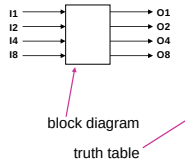
7 gates but they all have 2 inputs each

Design example: a two-bit comparator (con't)

Circuit schematics



Design example: BCD increment by 1



I8	I4	I2	I1	O8	O4	O2	O1
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

Need a 4 -variable Karnaugh map for each of the 4 output functions

Design example: BCD increment by 1 (con't)

$O_8 = I_4I_2I_1 + I_8I_1$`

$O_4 = I_4I_2' + I_4I_1' + I_4'I_2I_1$

O_8				
I_2 \ I_8I_1	00	01	11	10
00	0	0	X	1
01	0	0	X	0
11	0	1	X	X
10	0	0	X	X

O_4				
I_2 \ I_8I_1	00	01	11	10
00	0	1	X	0
01	0	1	X	0
11	1	0	X	X
10	0	1	X	X

$O_2 = I_8'I_2'I_1 + I_2I_1$`

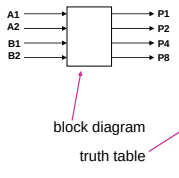
$O_1 = I_1'$

O_2				
I_2 \ I_8I_1	00	01	11	10
00	0	0	X	0
01	1	1	X	0
11	0	0	X	X
10	1	1	X	X

O_1				
I_2 \ I_8I_1	00	01	11	10
00	1	1	X	1
01	0	0	X	0
11	0	0	X	X
10	1	1	X	X

We greatly simplify the logic by using the don't cares

Design example: a two-bit multiplier



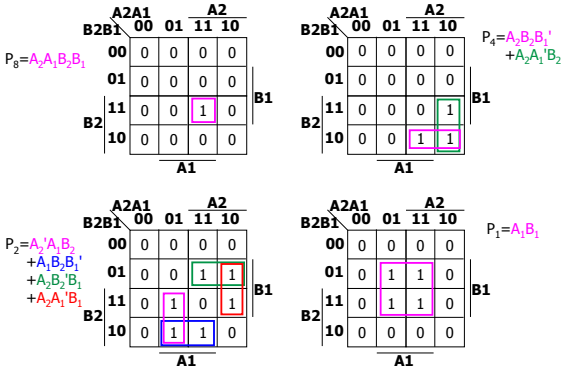
block diagram

truth table

A2	A1	B2	B1	P8	P4	P2	P1
0	0	0	0	0	0	0	0
		0	1	0	0	0	0
		1	0	0	0	0	0
		1	1	0	0	0	0
0	1	0	0	0	0	0	0
		0	1	0	0	0	1
		1	0	0	0	1	0
		1	1	0	0	1	1
1	0	0	0	0	0	0	0
		0	1	0	0	1	0
		1	0	0	1	0	0
		1	1	0	1	1	0
1	1	0	0	0	0	0	0
		0	1	0	0	1	1
		1	0	0	1	1	0
		1	1	1	0	0	1

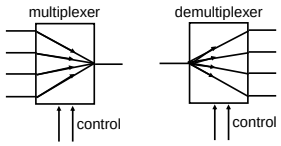
Need a 4 -variable Karnaugh map for each of the 4 output functions

Two-bit multiplier (cont'd)



Switching-network logic blocks

- Multiplexer (MUX)
  - Routes one of many inputs to a single output
  - Also called a *selector*
- Demultiplexer (DEMUX)
  - Routes a single input to one of many outputs
  - Also called a *decoder*



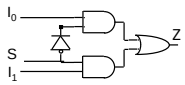
We construct these devices from:
- logic gates
- networks of transistor switches

Multiplexers

- Basic concept
  - 2^n data inputs; n control inputs ("selects"); 1 output
  - Connects one of 2^n inputs to the output
  - "Selects" decide which input connects to output
  - Two alternative truth-tables: Functional and Logical

Example: A 2:1 Mux

$Z = SIn_1 + S'In_0$



Functional truth table

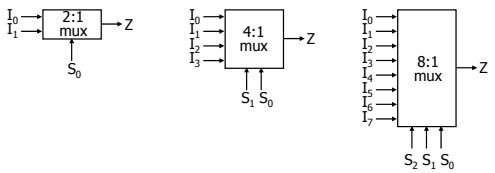
S	Z
0	In_0
1	In_1

Logical truth table

In_1	In_0	S	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Multiplexers (con't)

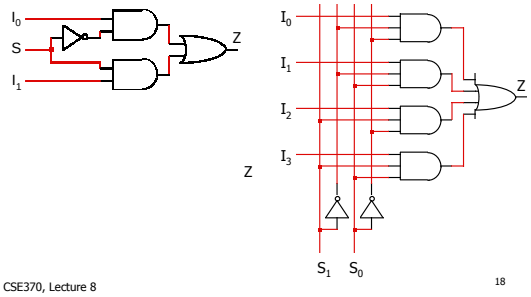
- 2:1 mux: $Z = S_0'In_0 + S_0In_1$
- 4:1 mux: $Z = S_1'S_0'In_0 + S_1'S_0In_1 + S_1S_0'In_2 + S_1S_0In_3$
- 8:1 mux: $Z = S_2'S_1'S_0'In_0 + S_2'S_1'S_0In_1 + \ldots$



Logic-gate implementation of multiplexers

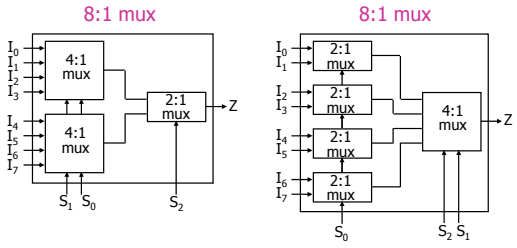
2:1 mux

4:1 mux



Cascading multiplexers

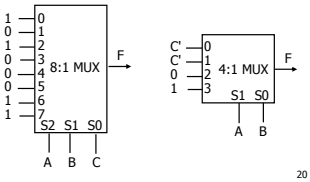
- Can form large multiplexers from smaller ones
  - Many implementation options



Multiplexers as general-purpose logic

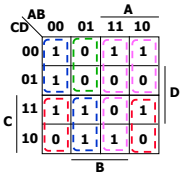
- A 2^n:1 mux can implement any function of n variables
  - A lookup table
  - A 2^{n-1}:1 mux also can implement any function of n variables
- Example: F(A,B,C) = m0 + m2 + m6 + m7
  = A'B'C' + A'BC' + ABC' + ABC
  = A'B'(C') + A'B(C') + AB(0) + AB(1)

A	B	C	F	
0	0	0	1	C'
0	0	1	0	
0	1	0	1	C'
0	1	1	0	
1	0	0	0	0
1	0	1	0	
1	1	0	1	1
1	1	1	1	



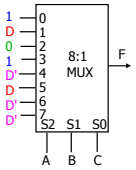
Multiplexers as general-purpose logic

- Implementing a 2^{n-1}:1 mux as a function of n variables
  - (n-1) mux control variables $S_0 - S_{n-2}$
  - One data variable S_{n-1}
  - Four possible values for each data input: 0, 1, S_{n-1}, S_{n-1}'
  - Example: F(A,B,C,D) implemented using an 8:1 mux



Choose A,B,C as control variables

Choose D as a data variable



Demultiplexers (DEMUX)

- Next class