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
- A
- B
- C
- D

### OUTPUTS
- W
- X
- Y
- Z

- Function F computes the next number in a BCD sequence
- If the input is 0010, the output is 0011

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

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

Example: with don’t cares

F(A,B,C,D) = Σm(1,3,5,7,9) + d(6,12,13)

F = AD + BCD, without using don’t cares
F = AD + CD, using don’t cares

POS minimization using k-maps

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

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+m6+d10+d11+d12+d13+d14+d15
    - = Σm(7,8) + ΠM(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) • ΣD(10,11,12,13,14,15)
- In K-maps, can treat 'don't cares' as 0s or 1s
  - Depending on which is more advantageous

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

Same idea as the Truth Table
Design example: a two-bit comparator

Block diagram

Truth table

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

K-map for LT

K-map for EQ

K-map for GT

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

Two ways to implement EQ:

Option 1:

\[ EQ = A'B'CD + A'BC'D + ABCD + AB'D' \]

5 gates but they require lots of inputs

Option 2:

\[ EQ = (A \text{xnor} C) \times (B \text{xnor} D) \]

XNOR is constructed from 3 simple gates

7 gates but they all have 2 inputs each

Design example: BCD increment by 1

Truth table

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

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

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

\[ O_3 = I_2I_4 + I_2I_4' \]

\[ O_4 = I_1I_4 + I_1I_4' + I_1I_4I_4' \]

\[ O_5 = I_1I_4I_4 + I_1I_4I_4' \]

\[ O_6 = I_1I_4 + I_1I_4' \]
Design example: a two-bit multiplier

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
0 1 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

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

Functional truth table

<table>
<thead>
<tr>
<th>S</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 In</td>
</tr>
<tr>
<td>1</td>
<td>1 In</td>
</tr>
</tbody>
</table>

Logical truth table

<table>
<thead>
<tr>
<th>S</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 In</td>
</tr>
<tr>
<td>1</td>
<td>1 In</td>
</tr>
</tbody>
</table>

Multiplexers (con’t)

- 2:1 mux: \( Z = S_0 I_0 + S_1 I_1 \)
- 4:1 mux: \( Z = S_0 S_1' I_0 + S_0 S_1 I_1 + S_1 S_0' I_2 + S_1 S_0 I_3 \)
- 8:1 mux: \( Z = S_0 S_1' S_2' I_0 + S_0 S_1 S_2' I_1 + S_1 S_0' S_2 I_2 + S_1 S_0 S_2 I_3 \)

Logic-gate implementation of multiplexers
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)

Demultiplexers (DEMUX)

- Next class