CS/COE 0447

Logic Minimization and K-Maps

wilkie (with content borrowed from:
Jarrett Billingsley
Dr. Bruce Childers)

Karnaugh Maps

X Marks the Spot … Well it is actually a bunch of rectangles

Gray Code

- **Gray code** is a way of counting in binary where only **one bit** changes on each count
- For our purposes, just knowing the 2-bit code enough.

```
00
01
11
10
```

2-bit Gray code
(red bits are bits that change)

Karnaugh Maps (K-maps) – Setting up

- A **Karnaugh Map** is a tool for minimizing boolean functions
- Let's start with a function that has two inputs

Truth Table

A	B	Q
0	0	1
0	1	0
1	0	1
1	1	1

K-map

		0	1
		$\bar{B}$	B
0	$\bar{A}$	1	0
1	A	1	1

1. Write input values ***in Gray code*** along axes.
   - (there's only one input on each side here, it's easy)
2. Fill in cells from truth table.

Karnaugh Maps (K-maps) – Finding rects

K-map

	$\overline{B}$	B
$\overline{A}$	1	0
A	1	1

3. Find **rectangles of 1s** with these rules:
   - width and height can only be 1, 2, or 4
     - NEVER 3
   - overlapping is totally fine! it's *good!*
   - use **the <u>biggest</u> rectangles possible**
   - use **the <u>fewest</u> rectangles possible**

Karnaugh Maps (K-maps) – Interpreting rects

<u>**K-map**</u>

	$\overline{B}$	B
$\overline{A}$	1	0
A	1	1

Red: $\overline{B}$ **Blue: A**

$$Q = A + \overline{B}$$

4. For each rectangle, look at the values of the variables along the axes. some variables **change**, and others don't.
   - Which variable changes in the red rectangle? which doesn't?
   - What about the blue rectangle?
5. Each rectangle is an AND term
   - Write the variables **that stay the same for that rect** (keeping the NOT bars)
   - Ignore the variables that **change**
6. OR all the terms together
7. WHEW!

I'd like to place an order for the carry-out bit

C_i	A	B	C_o
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

- With more than 2 variables, put two along one axis (**GRAY CODE!**)

Try to make the rectangles **as big as possible.** overlap is goooood.

		00	01	11	10
		$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
0	$\bar{C}_i$	0	0	1	0
1	C_i	0	1	1	1

Red: AC_i

Green: AB

Blue: BC_i

$$C_o = AC_i + AB + BC_i$$

Just like a 2D RPG world map...

- rectangles on K-maps can *wrap around* (left-right AND top-bottom!)

		00	01	11	10
		$\overline{X}\overline{Y}$	$\overline{X}Y$	XY	$X\overline{Y}$
0	$\overline{Z}$	1	1	0	1
1	Z	1	0	0	1

Red: $\overline{Y}$

Blue: $\overline{X}\overline{Z}$

$$Q = \overline{Y} + \overline{X}\overline{Z}$$

this is really a **2x2 rectangle.**
it's just... doing its best.

Okay, maybe it's not perfect.

- let's try the Sum output of a full adder

a **1x1 rectangle** becomes a term that uses all the variables

		00	01	11	10
		$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
0	$\bar{C}_i$	0	1	0	1
1	C_i	1	0	1	0

Red: $\bar{A}\bar{B}C_i$

Green: $\bar{A}B\bar{C}_i$

Blue: ABC_i

Purple: $A\bar{B}\bar{C}_i$

$$\mathbf{Sum} = \bar{A}\bar{B}C_i + \bar{A}B\bar{C}_i + ABC_i + A\bar{B}\bar{C}_i$$

wait, didn't we say that we could do it as:

$$\mathbf{Sum} = A \oplus B \oplus C_i$$ *(that's xor!)*

Tradeoffs, tradeoffs

- There are extensions to K-maps to detect XORs
- but...
  - XOR gates are slower than AND/OR gates
  - if area is a concern, an XOR make sense
  - if speed is a concern, AND/OR gates make sense
- What do real hardware designers do?
  - They use programs to do this stuff for them lol
  - Things like FPGAs, CPLDs, and GALs are *reconfigurable hardware* which usually use "sum-of-products" to do logic, so ANDs and ORs are all you've got

(Pst. What if we have lots of 1's?)

C_i	A	B	C_o
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

When we select 0's, we get a **product of sums** that represents the inverse of our function.
It helps when we have less 0s than 1s.

		00	01	11	10
		$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
0	$\overline{C_i}$	0	0	1	0
1	C_i	0	1	1	1

$$\overline{C_o} = (\bar{B} + \overline{C_i})(\bar{A} + \bar{B})(\bar{A} + \overline{C_i})$$

$$C_o = \overline{(\bar{B} + \overline{C_i})(\bar{A} + \bar{B})(\bar{A} + \overline{C_i})}$$

(Doesn't help in the worst case)

- let's try the Sum output of a full adder

a **1x1 rectangle** becomes a term that uses all the variables

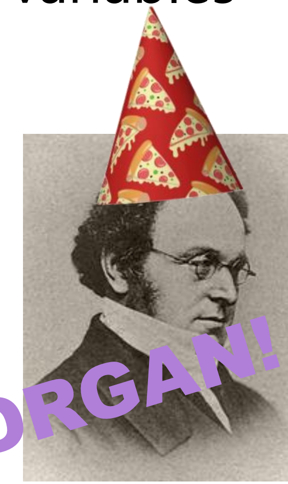
		00	01	11	10
		$\overline{A}\,\overline{B}$	$\overline{A}B$	AB	$A\overline{B}$
0	$\overline{C_i}$	0	1	0	1
1	C_i	1	0	1	0

Red: $\overline{A} + B + C_i$

Green: $\overline{A} + \overline{B} + \overline{C_i}$

Blue: $A + \overline{B} + C_i$

Purple: $A + B + \overline{C_i}$



THANKS DeMORGAN!

(Although everybody already knew this)

$$S = \overline{(\overline{A} + B + C_i)(\overline{A} + \overline{B} + \overline{C_i})(A + \overline{B} + C_i)(A + B + \overline{C_i})}$$

Same as: $S = \overline{A}\,\overline{B}C_i + \overline{A}B\overline{C_i} + ABC_i + A\overline{B}\,\overline{C_i}$

(Here's the thing, though)

- Both methods are equivalent.
  - You can get f(a,b,c,d) by selecting 1s
  - Or the inverse: not(f(a,b,c,d)) by selecting 0s.
- You can use DeMorgan's Law to get f(a,b,c,d) from its inverse
  - It will be the same.
- Just select the 1s.
  - No need to make things more complicated.
  - But it is nice to know that logic… works as intended.
  - And discrete math was not a waste of time ☺
    - It's actually really useful. Let's be honest.