CS 61C: Great Ideas in Computer Architecture

Combinational and Sequential Logic, Boolean Algebra

Instructor: Justin Hsia
Review of Last Lecture

• OpenMP as simple parallel extension to C
  – During parallel fork, be aware of which variables should be shared vs. private among threads
  – Work-sharing accomplished with \texttt{for/sections}
  – Synchronization accomplished with \texttt{critical/atomic/reduction}

• Hardware is made up of transistors and wires
  – Transistors are voltage-controlled switches
  – \textit{Building blocks of all higher-level blocks}
Great Idea #1: Levels of Representation/Interpretation

**Higher-Level Language Program (e.g. C)**

- Compiler

**Assembly Language Program (e.g. MIPS)**

- Assembler

**Machine Language Program (MIPS)**

- Machine Interpretation

- Hardware Architecture Description (e.g. block diagrams)

- Architecture Implementation

- Logic Circuit Description (Circuit Schematic Diagrams)

- Register File

- ALU

- We are here

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw $t0, 0($2)  
lw $t1, 4($2)  
sw $t1, 0($2)  
sw $t0, 4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000  
1010 1111 0101 1000 0000 1001 1100 0110  
1100 0110 1010 1111 0101 1000 0000 1001  
0101 1000 0000 1001 1100 0110 1010 1111
```
Synchronous Digital Systems

Hardware of a processor, such as the MIPS, is an example of a Synchronous Digital System

**Synchronous:**

- All operations coordinated by a central clock
  - “Heartbeat” of the system!

**Digital:**

- Represent all values with two discrete values
- Electrical signals are treated as 1’s and 0’s
  - 1 and 0 are complements of each other
- High/Low voltage for True/False, 1/0
Signals and Waveforms: Clocks

- **Signals** transmitted over wires continuously
- Transmission is effectively instantaneous
  - Implies that any wire only contains one value at any given time
Signals and Waveforms

Stack signals on top of each other

All signals change after clock “triggers”
Signals and Waveforms: Grouping

A group of wires when interpreted as a bit field is called a *Bus*.
Agenda

• Combinational Logic
  – Truth Tables and Logic Gates
• Administrivia
• Boolean Algebra
• Sequential Logic
  – State Elements
• Bonus: Karnaugh Maps (Optional)
Type of Circuits

• *Synchronous Digital Systems* consist of two basic types of circuits:
  
  • Combinational Logic (CL)
    – Output is a function of the inputs only, not the history of its execution
    – e.g. circuits to add A, B (ALUs)
  
  • Sequential Logic (SL)
    • Circuits that “remember” or store information
    • a.k.a. “State Elements”
    • e.g. memory and registers (Registers)
Representations of Combinational Logic

• Circuit Diagram
  – Transistors and wires (Lec 17)
  – Logic Gates (Lec 18)
• Truth Table (Lec 18)
• Boolean Expression (Lec 18)

• All are equivalent

Right Now!
Truth Tables

• Table that relates the inputs to a CL circuit to its output
  – Output *only* depends on current inputs
  – Use abstraction of 0/1 instead of source/ground
  – Shows output for *every* possible combination of inputs

• How big?
  – 0 or 1 for each of N inputs, so $2^N$ rows
CL: General Form

If N inputs, how many distinct functions F do we have?

Function maps each row to 0 or 1, so \(2^{2^N}\) possible functions

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>F(0,0,0,0)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>F(0,0,0,1)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>F(0,0,1,0)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>F(0,0,1,1)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>F(0,1,0,0)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>F(0,1,0,1)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>F(0,1,1,0)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>F(0,1,1,1)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>F(1,0,0,0)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>F(1,0,0,1)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>F(1,0,1,0)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>F(1,0,1,1)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>F(1,1,0,0)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>F(1,1,0,1)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>F(1,1,1,0)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>F(1,1,1,1)</td>
</tr>
</tbody>
</table>
• For 3 outputs, just three separate functions:
  \( X(A,B,C,D), Y(A,B,C,D), \) and \( Z(A,B,C,D) \)
  – Can show functions in separate columns without adding any rows!
Logic Gates (1/2)

- Special names and symbols:
  - **NOT**
    
    ![NOT Diagram]
    
    \[\begin{array}{c|c}
    a & c \\
    \hline
    0 & 1 \\
    1 & 0 \\
    \end{array}\]

  - **AND**
    
    ![AND Diagram]
    
    \[\begin{array}{c|c|c}
    a & b & c \\
    \hline
    0 & 0 & 0 \\
    0 & 1 & 0 \\
    1 & 0 & 0 \\
    1 & 1 & 1 \\
    \end{array}\]

  - **OR**
    
    ![OR Diagram]
    
    \[\begin{array}{c|c|c}
    a & b & c \\
    \hline
    0 & 0 & 0 \\
    0 & 1 & 1 \\
    1 & 0 & 1 \\
    1 & 1 & 1 \\
    \end{array}\]
Logic Gates (2/2)

- Special names and symbols:

  **NAND**

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

  **NOR**

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

  **XOR**

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
# More Complicated Truth Tables

## 3-Input Majority

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

## 2-bit Adder

- **A**: $a_1$, $a_0$
- **B**: $b_1$, $b_0$
- **C**: $c_2$, $c_1$, $c_0$

3 separate functions

How many rows?

---

7/18/2012 Summer 2012 -- Lecture #18
Question: Convert the following statements into a Truth Table assuming the output D is whether Justin is comfortable (1) or uncomfortable (0).

- **Input A:** Justin wears light (0) or heavy (1) clothing
- **Input B:** It is cold (0) or hot (1) outside
- **Input C:** Justin spends the day indoors (0) or outdoors (1)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
• Truth tables are huge
  – Write out EVERY combination of inputs and outputs (thorough, but inefficient)
  – Finding a particular combination of inputs involves scanning a large portion of the table
• There must be a shorter way to represent combinational logic
  – Boolean Algebra to the rescue!
Agenda

• Combinational Logic
  – Truth Tables and Logic Gates
• Administrivia
• Boolean Algebra
• Sequential Logic
  – State Elements
• Bonus: Karnaugh Maps (Optional)
Administrivia

• Midterm re-grade requests due tomorrow
• Project 2: Matrix Multiply Performance Improvement
  – In partners
  – Part 1: Due July 22 (this Sunday)
  – Part 2: Due July 29
• HW 4 also due July 25
  – Will be released tonight
• Justin moving OH permanently to 200 SDH
Agenda

• Combinational Logic
  – Truth Tables and Logic Gates

• Administrivia

• Boolean Algebra

• Sequential Logic
  – State Elements

• Bonus: Karnaugh Maps (Optional)
Boolean Algebra

• Represent inputs and outputs as variables
  – Each variable can only take on the value 0 or 1
• Overbar is NOT: “logical complement”
  – e.g. if A is 0, then \( \bar{A} \) is 1 and vice-versa
• Plus (+) is 2-input OR: “logical sum”
• Product (\( \cdot \)) is 2-input AND: “logical product”
  – All other gates and logical expressions can be built from combinations of these
    (e.g. \( A \text{ XOR } B = \bar{A}B + \bar{B}A = A'B + AB' \))
Truth Table to Boolean Expression

• Read off of table
  – For 1, write variable name
  – For 0, write complement of variable

• *Sum of Products (SoP)*
  – Take rows with 1’s in output column, sum products of inputs
  – \( c = \overline{a}b + \overline{b}a \)

• *Product of Sums (PoS)*
  – Take rows with 0’s in output column, product the sum of the complements of the inputs
  – \( c = (a + b) \cdot (\overline{a} + \overline{b}) \)

We can show that these are equivalent!

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Manipulating Boolean Algebra

• SoP and PoS expressions can still be long
  – We wanted to have shorter representation than a truth table!

• Boolean algebra follows a set of rules that allow for simplification
  – Goal will be to arrive at the simplest equivalent expression
  – Allows us to build simpler (and faster) hardware
Faster Hardware?

• **Recall:** Everything we are dealing with is just an abstraction of transistors and wires
  – Inputs propagating to the outputs are voltage signals passing through transistor networks
  – There is always some *delay* before a CL’s output updates to reflect the inputs

• Simpler Boolean expressions $\leftrightarrow$ smaller transistor networks $\leftrightarrow$ smaller circuit delays $\leftrightarrow$ faster hardware
Combinational Logic Circuit Delay

A = \{a_3, a_2, a_1, a_0\}
B = \{b_3, b_2, b_1, b_0\}

Symbol for a bus (and width)

Combinational Logic delay
Laws of Boolean Algebra

These laws allow us to perform simplification:

\[
\begin{align*}
    x \cdot \overline{x} &= 0 &
    x + \overline{x} &= 1 \\
    x \cdot 0 &= 0 &
    x + 1 &= 1 \\
    x \cdot 1 &= x &
    x + 0 &= x \\
    x \cdot x &= x &
    x + x &= x \\
    x \cdot y &= y \cdot x &
    x + y &= y + x \\
    (xy)z &= x(yz) &
    (x + y) + z &= x + (y + z) \\
    x(y + z) &= xy + xz &
    x + yz &= (x + y)(x + z) \\
    xy + x &= x &
    (x + y)x &= x \\
    \overline{xy} + x &= x + y &
    (\overline{x} + y)x &= xy \\
    x \cdot \overline{y} &= \overline{x + \overline{y}} &
    x + \overline{y} &= \overline{x \cdot \overline{y}}
\end{align*}
\]

complementarity
laws of 0’s and 1’s identities
idempotent law
commutativity
associativity
distribution
uniting theorem
uniting theorem v.2
DeMorgan’s Law
Boolean Algebraic
Simplification Example

\[ y = ab + a + c \]
Circuit Simplification

1) original circuit (Transistors and/or Gates)

\[ y = ((ab) + a) + c \]
\[ = ab + a + c \]
\[ = a(b + 1) + c \]
\[ = a(1) + c \]
\[ = a + c \]

2) equation derived from original circuit

3) algebraic simplification

4) simplified circuit
Converting Combinational Logic

Try all input combinations

Circuit Diagram

Truth Table

Boolean Expression

Try all input combinations

Propagate signals through gates

Wire inputs to proper gates (easiest to use AND, OR, and NOT)

This is difficult to do efficiently!
Circuit Simplification Example (1/4)

• Simplify the following circuit:

• Options:

  1) Test all combinations of the inputs and build the Truth Table, then use SoP or PoS

  2) Write out expressions for signals based on gates
     • Will show this method here
Circuit Simplification Example (2/4)

• Simplify the following circuit:

\[
\begin{align*}
    A & \quad \neg B \quad C \\
    B & \quad \neg B' \quad C \\
    \end{align*}
\]

• Start from left, propagate signals to the right
• Arrive at \( D = (AB)'(A + B'C) \)
Circuit Simplification Example (3/4)

- Simplify Expression:

\[ D = (AB)'(A + B'C) \]

\[ = (A' + B')(A + B'C) \quad \text{DeMorgan's} \]

\[ = A'A + A'B'C + B'A + B'B'C \quad \text{Distribution} \]

\[ = 0 + A'B'C + B'A + B'B'C \quad \text{Complementarity} \]

\[ = A'B'C + B'A + B'C \quad \text{Idempotent Law} \]

\[ = (A' + 1)B'C + AB' \quad \text{Distribution} \]

\[ = B'C + AB' \quad \text{Law of 1's} \]

\[ = B'(A + C) \quad \text{Distribution} \]

Which of these is “simpler”?
Circuit Simplification Example (4/4)

• Draw out final circuit:
  - \( D = B'C + AB' = B'(A + C) \)

- Reduction from 6 gates to 3!

• Simplified Circuit:

- How many gates do we need for each?

- Reduction from 6 gates to 3!
Karnaugh Maps (Optional)

• Lots of Boolean Algebra laws for simplification
  – Difficult to memorize and spot applications
  – When do you know if in simplest form?

• Karnaugh Maps (K-maps) are an alternate way to simplify Boolean Algebra
  – This technique is normally taught in CS150
  – We will never ask you to use a K-map to solve a problem, but you may find it faster/easier if you choose to learn how to use it (see Bonus Slides)
Get To Know Your Staff

• Category: Television
Agenda

• Combinational Logic
  – Truth Tables and Logic Gates
• Administrivia
• Boolean Algebra
• Sequential Logic
  – State Elements
• Bonus: Karnaugh Maps (Optional)
Type of Circuits

• *Synchronous Digital Systems* consist of two basic types of circuits:
  • Combinational Logic (CL)
    – Output is a function of the inputs only, not the history of its execution
    – e.g. circuits to add A, B (ALUs)
  • Sequential Logic (SL)
    • Circuits that “remember” or store information
    • a.k.a. “State Elements”
    • e.g. memory and registers (Registers)
Uses for State Elements

• Place to store values for some amount of time:
  – Register files (like in MIPS)
  – Memory (caches and main memory)

• *Help control flow of information between combinational logic blocks*
  – State elements are used to hold up the movement of information at the inputs to combinational logic blocks and allow for orderly passage
Accumulator Example

An example of why we would need to control the flow of information.

\[ X_i \rightarrow \text{SUM} \rightarrow S \]

Want: \( S=0; \)

For \( X_1, X_2, X_3 \) over time...

\[ S = S + X_i \]

Assume:

- Each \( X \) value is applied in succession, one per cycle
- The sum since time 1 (cycle) is present on \( S \)
First Try: Does this work?

No!

1) How to control the next iteration of the ‘for’ loop?
2) How do we say: ‘S=0’?
Second Try: How About This?

A Register is the state element that is used here to hold up the transfer of data to the adder.

Delay through Register and Adder

Rough timing ...
Register Internals

- n instances of a “Flip-Flop”
  - Output flips and flops between 0 and 1
- Specifically this is a “D-type Flip-Flop”
  - D is “data input”, Q is “data output”
  - In reality, has 2 outputs (Q and \(\bar{Q}\)), but we only care about 1

Flip-Flop Timing Behavior (1/2)

- Edge-triggered D-type flip-flop
  - This one is “positive edge-triggered”
- “On the rising edge of the clock, input d is sampled and transferred to the output. At other times, the input d is ignored and the previously sampled value is retained.”
- Example waveforms:
Flip-Flop Timing Terminology (1/2)

• Camera Analogy: Taking a photo
  – *Setup time:* don’t move since about to take picture (open camera shutter)
  – *Hold time:* need to hold still after shutter opens until camera shutter closes
  – *Time to data:* time from open shutter until image appears on the output (viewfinder)
Flip-Flop Timing Terminology (2/2)

• Now applied to hardware:
  – *Setup Time*: how long the input must be stable *before* the CLK trigger for proper input read
  – *Hold Time*: how long the input must be stable *after* the CLK trigger for proper input read
  – “*CLK-to-Q*” *Delay*: how long it takes the output to change, measured from the CLK trigger
Flip-Flop Timing Behavior (2/2)

- Edge-triggered d-type flip-flop
  - This one is “positive edge-triggered”
- “On the rising edge of the clock, input d is sampled and transferred to the output. At other times, the input d is ignored and the previously sampled value is retained.”

![Flip-Flop Timing Diagram](image)
Accumulator Revisited
Proper Timing (2/2)

- reset signal shown
- Also, in practice $X_i$ might not arrive to the adder at the same time as $S_{i-1}$
- $S_i$ temporarily is wrong, but register always captures correct value
- In good circuits, instability never happens around rising edge of CLK
Summary

• Hardware systems are constructed from **Stateless** Combinational Logic and **Stateful** “Memory” Logic (registers)

• Voltages are analog, but quantized to represent logical 0’s and 1’s

• Combinational Logic: equivalent circuit diagrams, truth tables, and Boolean expressions
  – Boolean Algebra allows minimization of gates

• State registers implemented from Flip-flops
Special Bonus Slides: You are NOT responsible for the material contained on the following slides!!! You may, however, find it useful to read anyway.
Agenda

• Combinational Logic
  – Truth Tables and Logic Gates
• Administrivia
• Boolean Algebra
• Sequential Logic
  – State Elements
• Bonus: Karnaugh Maps (Optional)
Karnaugh Maps (Optional)

• Again, this is completely OPTIONAL material
  – Recommended you use .pptx to view animations

• Karnaugh Maps (K-maps) are an alternate way to simplify Boolean Algebra
  – This technique is normally taught in CS150
  – We will never ask you to use a K-map to solve a problem, but you may find it faster/easier if you choose to learn how to use it

• For more info, see:
  http://en.wikipedia.org/wiki/Karnaugh_map
Underlying Idea

• Using Sum of Products, “neighboring” input combinations simplify
  – “Neighboring”: inputs that differ by a single signal
  – e.g. \( ab + a'b = b \), \( a'bc + a'bc' = a'b \), etc.
  – **Recall**: Each product only appears where there is a 1 in the output column

• **Idea**: Let’s write out our Truth Table such that the neighbors become apparent!
  – Need a Karnaugh map for *EACH* output
Reorganizing the Truth Table

- Split inputs into 2 *evenly-sized* groups
  - One group will have an extra if an odd # of inputs
- Write out all combinations of one group horizontally and all combinations of the other group vertically
  - Group of n inputs → $2^n$ combinations
  - Successive combinations change only 1 input

2 Inputs:

```
A | B
---|---
0 | 0 1
1 |   
```

3 Inputs:

```
A | B | C
---|---|---
0 | 00 | 01 11 10
0 |    |   
1 |    |   
```
K-map: Majority Circuit (1/2)

- Filling in the Karnaugh map:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- Each row of truth table corresponds to ONE cell of Karnaugh map
- Recommended you view the animation on this slide on the Powerpoint (pptx)
- Note the funny jump when you go from input 011 to 100 (most mistakes made here)
K-map: Majority Circuit (2/2)

• Group neighboring 1’s so all are accounted for:
  – Each group of neighbors becomes a product term in output

• \( y = bc + ab + ac \)

• Larger groups become smaller terms
  – The single 1 in top row \( \rightarrow \) \( abc' \)
  – Vertical group of two 1’s \( \rightarrow \) \( ab \)
  – If entire lower row was 1’s \( \rightarrow \) \( c \)
General K-map Rules

- Only group in powers of 2
  - Grouping should be of size $2^i \times 2^j$
  - Applies for both directions
- Wraps around in all directions
  - “Corners” case is extreme example
- Always choose largest groupings possible
  - Avoid single cells whenever possible

- $y = bd + b'd' + acd$

<table>
<thead>
<tr>
<th></th>
<th>ab</th>
<th>cd</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>1 0 0 1</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>0 1 1 0</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0 1 1 1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1 0 0 1</td>
<td></td>
</tr>
</tbody>
</table>

1) NOT a valid group
2) IS a valid group
3) IS a valid group
4) “Corners” case
5) 1 of 2 good choices here (spot the other?)