Last Time

• Introduction to embedded systems
  – Properties
  – Examples

• Class structure
Administrivia

Lab hours proposal
- Monday/Wednesday: 1-5
- Tuesday/Thursday: 10:30-12:30 and 1:30-4:30

Lab location: EL 124
- If you do not find the lab open, find one of us (in 141 or 152)
Administrivia

Office hours (until lab opens):

• Andrew Fagg (EL 152):
  – Monday: 1-2
  – Wednesday: 4-5
  – Or by appointment

• Mark Woehrer (EL 141):
  – Tuesday: 10:30-11:30
  – Wednesday: 3:30-4:30
  – Or by appointment
Schedule Notes

• Our class schedule is fluid
  – Changes will be announced in class and on the Blackboard
Today: Introduction to Digital Logic

• Binary encoding
• Boolean algebra
• Transistors to logic gates
Encoding Information

In your ‘circuits and sensors’ class: how did you encode information?
• e.g., the acceleration measured by your accelerometer?
• or the rate of bend of a piezoelectric device?
Encoding Information

Following some form of (implementation dependent) analog conditioning:

• Acceleration (or bend rate) is encoded in the voltage that is output from the circuit
• As acceleration increases, the voltage also increases
Encoding Information

Following some form of (implementation dependent) analog conditioning:

• Acceleration (or bend rate) is encoded in the voltage that is output from the circuit
• As acceleration increases, the voltage also increases

• We say that this is an analog or continuous encoding of the information
Analog Encoding

What is the problem with analog encoding?
Analog Encoding

What is the problem with analog encoding?

• Small injections of noise – either in the sensor itself or from external sources – will affect this analog signal

• This leads to errors in how we interpret the sensory data

How do we fix this?
Digital Encoding

How do we fix this?

• At any instant, a single signal encodes one of two values:
  – A voltage around 0 (zero) Volts is interpreted as one value
  – A voltage around +5 V is interpreted as another value
Binary Encoding

• Binary digits can have one of two values: 0 or 1
• We call 0V a binary “0” (or FALSE)
• And +5V a binary “1” (or TRUE)
Binary Encoding

• Exactly what these levels are depends on the technology that is used (it is common now to see +1.8V as a binary 1 in low-power processors)

• This encoding is much less sensitive to noise: small changes in voltage do not affect how we interpret the signal
Transistors

What do transistors do for us?
Transistors

What do transistors do for us?

• They act as current amplifiers

• But if we operate below the threshold and above the saturation level, we can use them to process digital signals.
Transistors to Digital Processing

Consider the following circuit:

- What is the output given an input of 0V?
- An input of +5V?
Transistors to Digital Processing

- Input: 0V -> Output +5V
- Input: +5V -> Output 0V
- We call this a “NOT” gate
The NOT Gate

- Logical Symbol: \[ A \quad \neg \quad B \]

- Algebraic Notation: \[ B = \overline{A} \]

- Truth Table:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
A Two-Input Gate

What does this circuit compute?

- A and B are inputs
- C is the output
The “NAND” Gate

- Logical Symbol:

- Algebraic Notation: \( C = \overline{A} \overline{B} = \overline{AB} \)

- Truth Table:

<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>
The “AND” Gate

- Logical Symbol:

- Algebraic Notation: $C = A \cdot B = AB$

- Truth Table:

<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>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Yet Another Gate

What does this circuit compute?
The “NOR” Gate

- Logical Symbol:

- Algebraic Notation: $C = \overline{A+B}$

- Truth Table:

<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>
The “OR” Gate

• Logical Symbol:

• Algebraic Notation: \( C = A + B \)

• Truth Table:

<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>1</td>
</tr>
</tbody>
</table>
N-Input Gates

Gates can have an arbitrary number of inputs (2, 3, 4, 8, 16 are common)
Exclusive OR ("XOR") Gates

- Logical Symbol:
- Algebraic Notation: $C = A \oplus B$
- Truth Table:

<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>

How would we implement this with AND/OR/NOT gates?
An XOR Implementation
An Example

Problem: implement an alarm system

• There are 3 inputs:
  – Door open (1 represents open)
  – Window open
  – Alarm active (1 represents active)

• And one output:
  – Siren is on (1 represents on) when either the door or window are open – but only if the alarm is active

What is the truth table?
### Alarm Example: Truth Table

<table>
<thead>
<tr>
<th>D</th>
<th>W</th>
<th>A</th>
<th>Siren</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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\overline{D} \ W \ A + \overline{D} \overline{W} \ A + D \ W \ A
\]
Alarm Example: Circuit

Siren = \overline{D} \ W \ A + D \ \overline{W} \ A + D \ W \ A
Alarm Example: Circuit

Is a simpler circuit possible?
## Alarm Example: Truth Table

<table>
<thead>
<tr>
<th>D</th>
<th>W</th>
<th>A</th>
<th>Siren</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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Andrew H. Fagg: Embedded Real-Time Systems: Digital Logic
Alarm: An Alternative Circuit

D
W
A

Siren
Next Time

• Some notes on Boolean Algebra
• DeMorgan’s Laws
• Multiplexers
• Demultiplexers
• Circuit reduction with Karnaugh Maps
Last Time

• Transistors to digital logic
• Basic gates: AND, OR, NOT
• Other gates: XOR, NOR, NAND
Today

Circuit reduction tools:
• Boolean Algebra
• DeMorgan’s Laws
• Karnaugh Maps
Administrivia

• Project 1 is delayed until next week
• You should have received the test email to the class mailing list: if not we need to fix it ASAP
Administrivia

Lab hours

• Monday/Wednesday: 9-11 and 2-5
• Tuesday/Thursday: 10:30-12:30 and 1:30-4:30

Until lab is handed out next week:

• Fagg: Monday: 10-11, Wednesday: 4-5
• Woehrer: Tuesday: 10:30-11:30, Thursday: 3:30-4:30
Boolean Algebra

• There are exactly two numbers in Boolean System: “0” and “1”
• You are already familiar with the “integers”: {... -2, -1, 0, 1, 2, ...}
  – (and Integer Algebra)
Boolean Algebra

- Like the integers, Boolean Algebra has the following operators:

<table>
<thead>
<tr>
<th></th>
<th>Integers</th>
<th>Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
<td>OR</td>
</tr>
<tr>
<td>*</td>
<td>product</td>
<td>AND</td>
</tr>
<tr>
<td>inverse</td>
<td>negation</td>
<td>NOT</td>
</tr>
</tbody>
</table>
NOT Operator

Definition:

- $\overline{0} = 0' = 1$
- $\overline{1} = 1' = 0$

NOTE: this is identical to our truth table (just a slightly different notation)

Suppose that “X” is a Boolean variable, then:

- $\overline{\overline{X}} = X'' = X$
OR (+) Operator

Definition:

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 0 = 1$
- $1 + 1 = 1$
OR (+) Operator

Suppose “X” is a Boolean variable, then:

- $0 + X = X$
- $1 + X = 1$
- $X + X = X$
- $X + X' = 1$
AND (*) Operator

Definition:

- \(0 \times 0 = 0\)
- \(0 \times 1 = 0\)
- \(1 \times 0 = 0\)
- \(1 \times 1 = 1\)
AND (*) Operator

Suppose “X” is a Boolean variable, then:

• 0 * X = 0
• 1 * X = X
• X * X = X
• X * X’ = 0
Boolean Algebra Rules: Precedence

The AND operator applies before the OR operator:

\[ A \times B + C = (A \times B) + C \]

\[ A + B \times C = A + (B \times C) \]
Boolean Algebra Rules: Association Law

If there are several AND operations, it does not matter which order they are applied in:

\[ A \ast B \ast C = (A \ast B) \ast C = A \ast (B \ast C) \]
Boolean Algebra Rules: Association Law

Likewise for the OR operator:

\[ A + B + C = (A + B) + C = A + (B + C) \]
Boolean Algebra Rules: 
Distributive Law

AND distributes across OR:

\[ A \times (B + C) = (A \times B) + (A \times C) \]

\[ A + (B \times C) = (A + B) \times (A + C) \]
Boolean Algebra Rules: Commutative Law

Both AND and OR are symmetric operators (the order of the variables does not matter):

\[ A + B = B + A \]

\[ A \times B = B \times A \]
DeMorgan’s Laws

\[(A \ast B)' = A' + B'\]

How do we convince ourselves that this is true?
Proof by Truth Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>(A * B)'</th>
<th>A' + B'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<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>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

NOTE: change in the NOT notation
DeMorgan’s Laws (cont)

\[(A + B)' = A' \times B'\]
Proof by Truth Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>(A + B)'</th>
<th>A' * B'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
### Alarm Example: Truth Table

<table>
<thead>
<tr>
<th>D</th>
<th>W</th>
<th>A</th>
<th>Siren</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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- $D'$ $W$ $A$  
  $+$  
- $D$ $W'$ $A$  
  $+$  
- $D$ $W$ $A$
### Reduction with Algebra

<table>
<thead>
<tr>
<th>Expression</th>
<th>Simplification</th>
</tr>
</thead>
<tbody>
<tr>
<td>D’ W A + D W’ A + DWA</td>
<td>X + X = X</td>
</tr>
<tr>
<td>= D’ W A + D W’ A + DWA + DWA</td>
<td></td>
</tr>
<tr>
<td>= D’ WA + DWA + DW’A + DWA</td>
<td>Commutative Law</td>
</tr>
<tr>
<td>= D’ WA + DWA + W’DA + WDA</td>
<td></td>
</tr>
<tr>
<td>= (D’ + D) WA + (W’ + W) DA</td>
<td>Associative Law</td>
</tr>
<tr>
<td>= 1* W A + 1* DA</td>
<td>X + X’ = 1</td>
</tr>
</tbody>
</table>
Reduction with Algebra (cont)

<table>
<thead>
<tr>
<th>1* WA + 1* DA</th>
<th>X * 1 = X</th>
</tr>
</thead>
<tbody>
<tr>
<td>= WA + DA</td>
<td>Associative Law</td>
</tr>
<tr>
<td>= (W + D) * A</td>
<td></td>
</tr>
</tbody>
</table>

We have the same circuit as before!
Karnaugh Maps

Geometric technique for reducing circuits

- Step 1: Construct the map
- Step 2: Fill in the output
- Step 3: Identify “clusters” in the map
- Step 4: Read out the terms
Suppose we have the following truth table:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</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>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</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>0</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>

3 Inputs
1 Output
Karnaugh Maps
Step 1: Construct Map

<table>
<thead>
<tr>
<th>C</th>
<th>AB</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Karnaugh Maps
Step 1: Construct Map

<table>
<thead>
<tr>
<th>AB</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Karnaugh Maps
Step 1: Construct Map

<table>
<thead>
<tr>
<th>AB</th>
<th>C</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A*B*C
Karnaugh Maps
Step 1: Construct Map

<table>
<thead>
<tr>
<th></th>
<th>AB</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A' B C'
### Karnaugh Maps

#### Step 2: Insert Output Values

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</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>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</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>0</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>

Enter output values into map.
Karnaugh Maps
Step 2: Insert Output Values

<table>
<thead>
<tr>
<th>AB</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</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>
</tr>
</tbody>
</table>

A B
Karnaugh Maps
Step 3: Identify Clusters of 1’s

<table>
<thead>
<tr>
<th>AB</th>
<th>C</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00</td>
<td>01</td>
<td>11</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Andrew H. Fagg: Embedded Real-Time Systems: Digital Logic
Karnaugh Maps
Step 3: Identify Clusters of 1’s

AB

C

0 0 1 1 0
1 1 1 0

A’ B
Karnaugh Maps
Step 3: Identify Clusters of 1’s

<table>
<thead>
<tr>
<th>AB</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</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>
</tr>
</tbody>
</table>
Karnaugh Maps

Step 3: Identify Clusters of 1’s

<table>
<thead>
<tr>
<th>AB</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</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>
</tr>
</tbody>
</table>
Karnaugh Maps
Step 3: Identify Clusters of 1’s

Clustering Rules:
• A cluster may not contain a ‘0’
• All ‘1’s must be covered

• But: it is OK for a ‘1’ to be covered by more than one cluster
Karnaugh Maps
Step 3: Identify Clusters of 1’s

<table>
<thead>
<tr>
<th>AB</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</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>
</tr>
</tbody>
</table>

Clusters: A' C, B
Karnaugh Maps
Step 4: Read Out the Terms

“OR” the clusters together

Resulting logical rule:
B + A’C
The Alarm Example (again)

<table>
<thead>
<tr>
<th>D</th>
<th>W</th>
<th>A</th>
<th>Siren</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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

D’ W A + D W’ A + D W A
Karnaugh Maps

Step 1: Construct Map

<table>
<thead>
<tr>
<th></th>
<th>DW</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Andrew H. Fagg: Embedded
Real-Time Systems: Digital Logic
Karnaugh Maps

Step 2: Insert Output Values

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

Andrew H. Fagg: Embedded
Real-Time Systems: Digital Logic
### Karnaugh Maps

**Step 3: Identify Clusters of 1’s**

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>DW</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Clusters of 1’s are circled in red.
Karnaugh Maps

Step 4: Read Out the Terms

Resulting logical rule:

WA + DA
Karnaugh Maps
Step 4: Read Out the Terms

Resulting logical rule:
WA + DA

Which can be simplified to:
A * (W + D)
Next Time

- More on Karnaugh Maps
- Multiplexers
- Demultiplexers
A New K-Map Example

• Four input variables: A, B, C, D
• One output variable: E
# Truth Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</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>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Step 1: Construct Map

\[\begin{array}{cccc}
AB & 00 & 01 & 11 & 10 \\
\hline
CD & 00 &   &   &   \\
    & 01 &   &   &   \\
    & 11 &   &   &   \\
    & 10 &   &   &   \\
\end{array}\]

Andrew H. Fagg: Embedded Real-Time Systems: Digital Logic
Step 2: Insert Output Values

<table>
<thead>
<tr>
<th>AB</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Step 3: Identify Clusters

<table>
<thead>
<tr>
<th>CD</th>
<th>AB</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Andrew H. Fagg: Embedded Real-Time Systems: Digital Logic
### Step 3: Identify Clusters

![Table and Diagram]

<table>
<thead>
<tr>
<th>CD</th>
<th>AB</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Andrew H. Fagg: Embedded Real-Time Systems: Digital Logic
Step 3: Identify Clusters

<table>
<thead>
<tr>
<th>AB</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>CD</td>
<td>00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>01</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

B' D

Andrew H. Fagg: Embedded Real-Time Systems: Digital Logic
### Step 3: Identify Clusters

<table>
<thead>
<tr>
<th>CD</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Step 4: Read Out the Terms

C A' + B' D + A B D' C'

What does the circuit look like?
Resulting Circuit

\[ C \cdot A' + B' \cdot D + A \cdot B \cdot D' \cdot C' \]
“Don’t Care” Cases

• For some functions that we will implement, some of the input cases will never occur

• How does this affect our Karnaugh Map?
## Modified Truth Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</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>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
## Step 2: Insert Output Values

<table>
<thead>
<tr>
<th>AB</th>
<th>CD</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>

Andrew H. Fagg: Embedded Real-Time Systems: Digital Logic
Step 3: Identify Clusters

<table>
<thead>
<tr>
<th>AB</th>
<th>CD</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>

ABD’C’
## Step 3: Identify Clusters

### ABC’

<table>
<thead>
<tr>
<th>CD</th>
<th>AB</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Andrew H. Fagg: Embedded Real-Time Systems: Digital Logic
Last Time

- Boolean Algebra
- Karnaugh Maps
Today

• More on Karnaugh Maps
• Multiplexers
• Demultiplexers
• Tristate buffers
Administrivia

Current office hours:

• Fagg: Monday: 10-11, Wednesday: 4-5
• Woehrer: Tuesday: 10:30-11:30, Thursday: 3:30-4:30
Step 3: Identify Clusters

<table>
<thead>
<tr>
<th>AB</th>
<th>CD</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>1</td>
<td>0</td>
<td>x</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>x</td>
</tr>
</tbody>
</table>

ABC'
Step 4: Read Out the Terms

C A’ + B’ D + A B C’

The resulting circuit is simpler (but just a little in this case)
A Diagonal Example

<table>
<thead>
<tr>
<th>C</th>
<th>AB</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Andrew H. Fagg: Embedded
Real-Time Systems: Digital Logic
### Working Through the Algebra…

<table>
<thead>
<tr>
<th>Expression</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A'B'C + ABC + A'BC' + AB'C'$</td>
<td></td>
</tr>
<tr>
<td>$(A'B' + AB)C + (A'B + AB')C'$</td>
<td>Associative Law</td>
</tr>
<tr>
<td>$(A'B' + AB)C + (A \oplus B)C'$</td>
<td>Commutative Law; XOR</td>
</tr>
<tr>
<td>$(A'B + AB')'C + (A \oplus B)C'$</td>
<td>DeMorgan</td>
</tr>
<tr>
<td>$(A \oplus B)'C + (A \oplus B)C'$</td>
<td>XOR Definition</td>
</tr>
<tr>
<td>$=A \oplus B \oplus C$</td>
<td>“</td>
</tr>
</tbody>
</table>
### Diagonal Example II

<table>
<thead>
<tr>
<th>AB</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Circles indicate specific values for the variables.
Diagonal Example II

$BA' + CB + AB'C'$

Can we get away with fewer terms?
Diagonal Example II

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

Andrew H. Fagg: Embedded Real-Time Systems: Digital Logic
Diagonal Example II

CB + C’(A XOR B)

We can also arrive at this through algebraic manipulation of our previous solution. How?
### Diagonal Example II

<table>
<thead>
<tr>
<th>Expression</th>
<th>Simplification</th>
</tr>
</thead>
<tbody>
<tr>
<td>( BA' + CB + AB'C' )</td>
<td>( X + X' = 1 )</td>
</tr>
<tr>
<td>( = BA'(C + C') + CB + AB'C' )</td>
<td>( = BA'C + BA'C' + CB + AB'C' )</td>
</tr>
<tr>
<td>( = BA'C + CB + C'(BA' + AB') )</td>
<td>( = BA'C + CB + C'(B XOR A) )</td>
</tr>
<tr>
<td>( = CB(A' + 1) + C'(B XOR A) )</td>
<td>( = CB + C' (B XOR A) )</td>
</tr>
<tr>
<td>( = CB + C' (B XOR A) )</td>
<td>( X + 1 = 1 )</td>
</tr>
</tbody>
</table>
Multiple Output Variables

Suppose we have a function with multiple output variables?

- How do we handle this with Karnaugh Maps?
Multiple Output Variables

How do we handle this with Karnaugh Maps?

• We produce one Karnaugh Map for each output variable

• But: in the final implementation, some sub-circuits may be shared
Karnaugh Maps: Final Notes

• Always use clusters that are as large as possible

• Possible to extend technique to 5 and 6 inputs (and more), but it starts to get hairy
  – Tend to make use of circuit design “assistants”
More Logical Components

• Multiplexer
• Demultiplexer
• Tristate buffer
A multiplexer is a device that selects between two input lines

- A & B are the inputs
- S is the selection signal (also an input)
- C is a copy of A if S=0
- C is a copy of B if S=1
What would the Karnaugh map look like?

<table>
<thead>
<tr>
<th>S</th>
<th>A</th>
<th>B</th>
<th>C</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>1</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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Multiplexer K-Map

<table>
<thead>
<tr>
<th>S</th>
<th>AB</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</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>
</tr>
</tbody>
</table>
Multiplexer Implementation

\[ C = S'A + SB \]
N-Input Multiplexer

Suppose we want to select from between $N$ different inputs.

- This requires more than one select line. How many?
N-Input Multiplexer

How many select lines?

- $M = \log_2 N$
- $N = 2^M$

What would the $N=8$ implementation look like?
8-Input Multiplexer Implementation

\[ C = I_0 S'_2 S'_1 S'_0 + I_1 S'_2 S'_1 S_0 + I_2 S'_2 S_1 S'_0 + \\
I_3 S'_2 S_1 S_0 + I_4 S'_2 S'_1 S'_0 + I_5 S_2 S'_1 S_0 + \\
I_6 S_2 S_1 S'_0 + I_7 S_2 S_1 S_0 \]

Note that we have one of each possible select line combination (or addressing terms)
Last Time

- Karnaugh Maps
- Multiplexers
Today

• Demultiplexers
• Tristate buffers
• Practical issues in digital logic implementation

• Project 1
Administrivia

Make sure you fill out and hand-in group placement forms today
What Is It?
A Mechanical Implementation of an OR Gate

goldfish.ikaruga.co.uk/logic.html
A Mechanical Implementation of an OR Gate
A Mechanical Implementation of an OR Gate
A Mechanical Implementation of an OR Gate
Demultiplexer

- The multiplexer reduces $N$ signals down to 1 (with $M$ select lines)
- A demultiplexer routes a data input ($D$) to one of $N$ output lines ($A_s$)
  - Which $A$ depends on the select lines
# 2-Input Demultiplexer Truth Table

<table>
<thead>
<tr>
<th>$S_1$</th>
<th>$S_0$</th>
<th>$D$</th>
<th>$A_3$</th>
<th>$A_2$</th>
<th>$A_1$</th>
<th>$A_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<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>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Inputs**

**Outputs**
Demultiplexer Implementation

- What does it look like?
Demultiplexer Implementation

Andrew H. Fagg: Embedded Real-Time Systems: Digital Logic
M-Input (Select) Demultiplexer

- Will have $2^M$ output lines
- Useful for converting a binary address into select lines for devices and memory elements
  - (more on binary soon)
Tristate Buffers

• Until now: the output line(s) of each device are driven either high or low
  – So the line is either a source or a sink of current
• Tristate buffers can do this or leave the line floating (as if it were not connected to anything)
Tristate Buffers

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>floating</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>floating</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

How are tristate buffers useful?
Tristate Buffers

We can wire the outputs of multiple tristate buffers together without any other logic
Tristate Buffers

• We must guarantee that only one select line is active at any one time

• Tristate buffers will turn out to be useful when we start building data and address buses
Another Tristate Buffer

What does the truth table look like?
Another Tristate Buffer

<table>
<thead>
<tr>
<th>S</th>
<th>A</th>
<th>B</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>floating</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>floating</td>
</tr>
</tbody>
</table>

A → B

S
Next Time

• Practical issues in digital logic implementation
• Project 1 details