Last time….

- Who can remind us what we talked about last time?
  - Electric circuit basics
    - $V_{cc} = 1$
    - Ground = 0
  - Transistors
    - PMOS
    - NMOS
  - Gates
    - Complementary PMOS + NMOS
    - Output is logical function of inputs
Boolean Gates

- Saw these gates
  - Mnemonic to remember them
**Boolean Functions, Gates and Circuits**

- **Circuits** are made from a network of gates.

\[(\neg A \land \neg C) \lor (A \land C)\]

![Circuit diagram](image)
A few more words about gates

- Gates have inputs and outputs
  - If you try to hook up two outputs, get short circuit
    (Think of the transistors each gate represents)

- If you don’t hook up an input, it behaves kind of randomly
  (also not good, but not set-your-chip-on-fire bad)
Introducing the Multiplexer (“mux”)

Selector (S)

Input A

Input B

Output

"A"
Introducing the Multiplexer ("mux")

Selector (S):

Input A:

Input B:

Output:

"A"
Introducing the Multiplexer ("mux")
Introducing the Multiplexer ("mux")
Let’s Make a Useful Circuit

• Pick between 2 inputs (called 2-to-1 MUX)
  • Short for multiplexor

• What might we do first?
  • Make a truth table?
    • S is selector:
      • S=0, pick A
      • S=1, pick B

• Next: how to get formula?
  Sum-of-products

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>S</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Sum of Products

- Find the rows where the output is 1
- Write a formula that exactly specifies each row
- OR these all together
  - Possible ways to get 1.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>S</th>
<th>Output</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>1</td>
<td>0</td>
<td>0</td>
<td>1</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>

\[
\begin{align*}
!A \land B \land S & \Rightarrow 0 \ 1 \ 1 \ 1 \\
A \land !B \land !S & \Rightarrow 1 \ 0 \ 0 \ 1 \\
A \land B \land !S & \Rightarrow 1 \ 1 \ 0 \ 1 \\
A \land B \land S & \Rightarrow 1 \ 1 \ 1 \ 1
\end{align*}
\]
Let’s Make a Useful Circuit

• Sum-of-products:
  
  (!A & B & S) | 
  (A & !B & !S) | 
  (A & B & !S) | 
  (A & B & S)

• This is long, though. Need to simplify.
Simplifying The Formula

• Simplifying this formula:

\[(\neg A \land B \land S) \lor (A \land \neg B \land \neg S) \lor (A \land B \land \neg S) \lor (A \land B \land S)\]

B doesn’t matter
Simplifying this formula:

\((\neg A \land B \land S) \lor (A \land \neg S) \lor (A \land B \land S)\)

A doesn’t matter
Simplifying The Formula

- Simplifying this formula:

\[(A \& !S) \mid (B \& S)\]
Let’s Make a Useful Circuit

- **Simplified formula:**
  
  \[(A \& !S) \mid (B \& S)\]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>S</th>
<th>Output</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>1</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>
Circuit Example: 2x1 MUX

Draw it in gates:

MUX(A, B, S) = (A & !S) | (B & S)

So common, we give it its own symbol:
Example 4x1 MUX

The / 2 on the wire means “2 bits”
Boolean Function Simplification

• Boolean expressions can be simplified by using the following rules (bitwise logical):

  • \( A \& A = A \)                     \( A \mid A = A \)
  • \( A \& 0 = 0 \)                     \( A \mid 0 = A \)
  • \( A \& 1 = A \)                     \( A \mid 1 = 1 \)
  • \( A \& !A = 0 \)                   \( A \mid !A = 1 \)
  • \(!!A = A\)

• \& and \mid are both commutative and associative
• \& and \mid can be distributed: \( A \& (B \mid C) = (A \& B) \mid (A \& C) \)
• \& and \mid can be subsumed: \( A \mid (A \& B) = A \)

• Can typically just let synthesis tools do this dirty work, but good to know
DeMorgan’s Laws

• Two (less obvious) Laws of Boolean Algebra:
  • Let us push negations inside, flipping & and |

  \[ \neg (A \& B) = (\neg A) \| (\neg B) \]

  \[ \neg (A \| B) = (\neg A) \& (\neg B) \]

  Alluded to these last time
    Very good rules to know in general
Next: logic to work with numbers

- Computers do one thing: math
  - And they do it well/fast
  - Fundamental rule of computation: “Everything is a number”
    - Computers can only work with numbers
    - Represent things as numbers
  - Specifically: good at binary math
    - Base 2 number system: matches circuit voltages
      - 1 (Vcc)
      - 0 (Ground)
    - Use fixed sized numbers
      - How many bits
  - Quick primer on binary numbers/math
    - Then how to make circuits for it
Numbers for computers

- We usually use base 10:
  - \(12345 = 1 \times 10^4 + 2 \times 10^3 + 3 \times 10^2 + 4 \times 10^1 + 5 \times 10^0\)
  - Recall from third grade: 1’s place, 10’s place, 100’s place...
    - Yes, we are going to re-cover 3rd grade math, but in binary
    - What is the biggest digit that can go in any place?

- Base 2:
  - 1’s place, 2’s place, 4’s place, 8’s place, ....
  - What is the biggest digit that can go in any place?
Basic Binary

- Advice: memorize the following
  - $2^0 = 1$
  - $2^1 = 2$
  - $2^2 = 4$
  - $2^3 = 8$
  - $2^4 = 16$
  - $2^5 = 32$
  - $2^6 = 64$
  - $2^7 = 128$
  - $2^8 = 256$
  - $2^9 = 512$
  - $2^{10} = 1024$
Binary continued:

- Binary Number Example: 101101
  - Take a second and figure out what number this is
Binary continued:

- Binary Number Example: 101101
  - Take a second and figure out what number this is
    1 in 32’s place = 32
    0 in 16’s place
    1 in 8’s place = 8
    1 in 4’s place = 4
    0 in 2’s place
    1 in 1’s place = 1
    —
    45
Converting Numbers

• Converting Decimal to Binary
  Suppose I want to convert 457 to binary
  Think for a second about how to do this
### Decimal to binary using remainders

<table>
<thead>
<tr>
<th>?</th>
<th>Quotient</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>457 ÷ 2 =</td>
<td>228</td>
<td>1</td>
</tr>
<tr>
<td>228 ÷ 2 =</td>
<td>114</td>
<td>0</td>
</tr>
<tr>
<td>114 ÷ 2 =</td>
<td>57</td>
<td>0</td>
</tr>
<tr>
<td>57 ÷ 2 =</td>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>28 ÷ 2 =</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>14 ÷ 2 =</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>7 ÷ 2 =</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>3 ÷ 2 =</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 ÷ 2 =</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Binary representation: 111001001
Decimal to binary using comparison

<table>
<thead>
<tr>
<th>Num</th>
<th>Compare $2^n$</th>
<th>$\geq ?$</th>
</tr>
</thead>
<tbody>
<tr>
<td>457</td>
<td>256</td>
<td>1</td>
</tr>
<tr>
<td>201</td>
<td>128</td>
<td>1</td>
</tr>
<tr>
<td>73</td>
<td>64</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>32</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$111001001$
Hexadecimal: Convenient shorthand for Binary

- Binary is unwieldy to write
  - 425,000 decimal = 1100111110000101000 binary
  - Generally about 3x as many binary digits as decimal
  - Converting (by hand) takes some work and thought

- Hexadecimal (aka “hex”)—base 16—is convenient:
  - Easy mapping to/from binary
  - Same or fewer digits than decimal
  - 425,000 decimal = 0x67C28
  - Generally write “0x” on front to make clear “this is hex”
  - Digits from 0 to 15, so use A—F for 10—15.
Binary ↔ Hex conversion is straightforward. Every 4 binary bits = 1 hex digit. If # of bits not a multiple of 4, add implicit 0s on left as needed.

<table>
<thead>
<tr>
<th>Hex digit</th>
<th>Binary</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>A</td>
<td>1010</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>1011</td>
<td>11</td>
</tr>
<tr>
<td>C</td>
<td>1100</td>
<td>12</td>
</tr>
<tr>
<td>D</td>
<td>1101</td>
<td>13</td>
</tr>
<tr>
<td>E</td>
<td>1110</td>
<td>14</td>
</tr>
<tr>
<td>F</td>
<td>1111</td>
<td>15</td>
</tr>
</tbody>
</table>

Indicates a hex number

0xDEADBEEF

1101 1110 1010 1101 1011 1110 1110 1111

0x02468ACE

0000 0010 0100 0110 1000 1010 1100 1110

0x13579BDF

0001 0011 0101 0111 1001 1011 1101 1111
Binary to/from hexadecimal

- $010110110010011_2$ -->
- $0101 \ 1011 \ 0010 \ 0011_2$ -->
- $5 \ B \ 2 \ 3_{16}$

1 F 4 B$_{16}$ -->

0001 1111 0100 1011$_2$ -->

0001111101001011$_2$
Binary Math : Addition

• Suppose we want to add two numbers:

  00011101

  + 00101011

  00101011

• How do we do this?
Binary Math : Addition

• Suppose we want to add two numbers:

\[
\begin{array}{c}
00011101 & 695 \\
+ 00101011 & + 232 \\
\end{array}
\]

• How do we do this?
  • Let’s revisit decimal addition
  • Think about the process as we do it
Binary Math : Addition

• Suppose we want to add two numbers:

\[
\begin{array}{c}
00011101 \\
+ 00101011 \\
\hline
00101011 \\
\end{array}
\hspace{1cm}
\begin{array}{c}
695 \\
+ 232 \\
\hline
77 \\
\end{array}
\]

• First add one’s digit \(5+2 = 7\)
Binary Math : Addition

• Suppose we want to add two numbers:

\[
\begin{array}{c}
1 \\
00011101 \\
+ 00101011 \\
\hline
27
\end{array}
\quad
\begin{array}{c}
695 \\
+ 232 \\
\hline
927
\end{array}
\]

• First add one’s digit 5+2 = 7
• Next add ten’s digit 9+3 = 12 (2 carry a 1)
Binary Math : Addition

• Suppose we want to add two numbers:

\[
\begin{array}{c}
00011101 \\
+ 00101011 \\
\hline
00101011
\end{array}
\]

1

\[
\begin{array}{c}
695 \\
+ 232 \\
\hline
927
\end{array}
\]

• First add one’s digit \(5+2 = 7\)
• Next add ten’s digit \(9+3 = 12\) (2 carry a 1)
• Last add hundred’s digit \(1+6+2 = 9\)
Binary Math : Addition

• Suppose we want to add two numbers:

\[
\begin{array}{c}
00011101 \\
+ 00101011 \\
\hline
00101011
\end{array}
\]

• Back to the binary:
• First add 1’s digit 1+1 = ...?
• Suppose we want to add two numbers:

\[
\begin{align*}
1 \\
00011101 \\
+ 00101011 \\
\hline \\
00101011
\end{align*}
\]

• Back to the binary:
• First add 1’s digit 1+1 = 2 (0 carry a 1)
• Suppose we want to add two numbers:

\[
\begin{array}{c}
  11 \\
  0011101 \\
+ 0010111 \\
\hline
  00
\end{array}
\]

• Back to the binary:
• First add 1’s digit 1+1 = 2 (0 carry a 1)
• Then 2’s digit: 1+0+1 = 2 (0 carry a 1)
• You all finish it out....
Binary Math : Addition

• Suppose we want to add two numbers:

\[
\begin{array}{c}
111111 \\
00011101 \quad = \quad 29 \\
+ \quad 00101011 \quad = \quad 43 \\
\hline
01001000 \quad = \quad 72
\end{array}
\]

• Can check our work in decimal
Negative Numbers

• May want negative numbers too!

• Many ways to represent negative numbers:
  • Sign/magnitude
  • Biased
  • 1’s complement
  • 2’s complement
2’s Complement Integers

- To negate, flip bits, add 1:
  - 1’s complement + 1

- Pros:
  - Easy to compute with
  - One representation of 0

- Cons:
  - More complex negation
  - Extra negative number (-8)

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>-8</td>
</tr>
<tr>
<td>1001</td>
<td>-7</td>
</tr>
<tr>
<td>1010</td>
<td>-6</td>
</tr>
<tr>
<td>1011</td>
<td>-5</td>
</tr>
<tr>
<td>1100</td>
<td>-4</td>
</tr>
<tr>
<td>1101</td>
<td>-3</td>
</tr>
<tr>
<td>1110</td>
<td>-2</td>
</tr>
<tr>
<td>1111</td>
<td>-1</td>
</tr>
</tbody>
</table>

EVERYBODY DOES THIS
Binary Math: Addition

• Revisit binary math for a minute:

\[
\begin{array}{c}
01011101 \\
+ 01101011 \\
\hline
01101011
\end{array}
\]
Binary Math: Addition

• What about this one:

\[
\begin{align*}
1111111 & \\
01011101 & = 93 \\
+ 01101011 & = 107 \\
\hline
11001000 & = -56
\end{align*}
\]

• But... that can’t be right?
  • What do you expect for the answer?
  • What is it in 8-bit signed 2’s complement?
Integer Overflow

- Answer should be 200
  - Not representable in 8-bit signed representation
  - **No** right answer

- Called Integer Overflow
  - Signed addition: CI $\neq$ CO of last bit
  - Unsigned addition: CO $\neq$ 0 of last bit

- Can detect in hardware
  - Signed: XOR CI and CO of last bit
  - Unsigned: CO of last bit
  - What processor does: depends
Subtraction

• 2’s complement makes subtraction easy:
  • Remember: A - B = A + (-B)
  • And: -B = ~B + 1
      ✤ that means flip bits (“not”)
  • So we just flip the bits and start with CI = 1
  • Fortunate for us: makes circuits easy (next time)

\[
\begin{array}{c}
1 \\
0110101 \rightarrow 0110101 \\
1010010 + 0101101
\end{array}
\]
Signed and Unsigned Ints

- Most programming languages support two int types
  - Signed: negative and positive
  - Unsigned: positive only, but can hold larger positive numbers

- Addition and subtraction:
  - Same, except overflow detection
  - x86: one add instruction, sets two different flags for overflows

- Inequalities
  - Different operations for signed/unsigned
  - Can someone give an example? (Let’s say 4-bit numbers)
One hot representation

• Binary representation convenient for math
• Another representation:
  • One hot: one wire per number
  • At any time, one wire = 1, others = 0

• Very convenient in many cases (e.g., homework 1)
Converting to/from one hot

- Converting from $2^N$ bits one hot to $N$ bits binary = **encoder**
  - E.g., “an 8-to-3 encoder”
- Converting from $N$ bits binary to $2^N$ bits one hot = **decoder**
  - E.g., “a 4-to-16 decoder”
  (which may be quite useful on hwk1)
Let's build a 4-to-2 encoder

• Start with a truth table
  • Input constrained to 1-hot: don’t care about invalid inputs
    • Can do anything we want

<table>
<thead>
<tr>
<th>In0</th>
<th>In1</th>
<th>In2</th>
<th>In3</th>
<th>Out0</th>
<th>Out1</th>
</tr>
</thead>
<tbody>
<tr>
<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>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

• Simplest formulas:
  • Out0 = In1 or In3
  • Out1 = In2 or In3

[alternatively: Out0 = In0 nor In2]
[alternatively: Out1 = In0 nor In1]
• Our 4-to-2 encoder
  • Note: the dots here show connections
  • Don’t confuse with open circles which mean NOT

In0  In1  In2  In3

Out0  Out1
• Our 4-to-2 encoder
  • Note: the dots here show connections
  • Don’t confuse with open circles which mean NOT

In0 didn’t actually figure into the logic. That’s ok
Synthesis will eliminate it if its not used elsewhere…
And whatever logic creates it, etc..
Let's build a 2-to-4 decoder

- Start with a truth table
  - Now input unconstrained

<table>
<thead>
<tr>
<th>In1</th>
<th>In0</th>
<th>Out0</th>
<th>Out1</th>
<th>Out2</th>
<th>Out3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<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>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Now sum-of-powers more useful, do for each of the 4 outputs:

- Out0 = (Not In1) and (Not In0)
- Out1 = (Not In1) and In0
- Out2 = In1 and (Not In0)
- Out3 = In1 and In0
2-to-4 decoder

In0

In1

Out0

Out1

Out2

Out3
• 2-to-4 decoder

In0

In1

Out0

Out1

Out2

Out3
2-to-4 decoder

- 2-to-4 decoder
Delays

• Mentioned before: switching not instant
  • Not going to try to calculate delays by hand (tools can do)
  • But good to know where delay comes from, to tweak/improve

• Gates:
  • Switching the transistors in gates takes time
  • More gates (in series) = more delay

• Fan-out: how many gates the output drives
  • Related to capacitance
  • High fan-out = slow
  • Sometimes better to replicate logic to reduce its fan-out

• Wire delay:
  • Signals take time to travel down wires
Wrap Up

• Combinatorial Logic
  • Putting gates together
  • Sum-of-products
  • Simplification
  • Muxes, Encoders, Decoders

• Number Representations
  • One Hot
  • 2’s complement