Lecture 7: Digital Logic

• Last time we introduced the concept of digital electronics
  – i.e., one identifies a range of voltages with the value “1”, and another range with the value “0”
• But we didn’t specify these ranges
  – or how fast we can expect the output to switch between 0 and 1
  – or how much power it takes to do so
  – or how much power the circuit outputs
  – or…
• All of these parameters depend on how the digital signals are generated
• Different ways of doing this are called logic “families”
• Currently, there are two families in common use:
  – TTL (Transistor-transistor logic)
  – CMOS (Complementary MOSFET)
TTL

- This family is built around the bipolar junction transistor
- Start with a common-emitter circuit:
  
  - If the input voltage is 0:
    - transistor is cut off
    - output voltage is
      \[ V_{out} = \frac{R_L}{R_L + R_C} V_{CC} \]
  
  - If the input voltage is high (~\(V_{CC}\))
    - transistor is saturated
    - output voltage is ~0.7V
• So this looks like a useful digital circuit
  – if performs the logical operation NOT
• But there are problems
• we want the “high” output to be close to $V_{CC}$
  – means $R_C$ should be small
• we don’t want the circuit to consume a lot of power
  – meaning we don’t want to pump a lot of current through it
  – means $R_C$ should be large!
• A somewhat more complicated circuit is required to meet our design goals
Real TTL circuit

• Here’s how to really make a NOT circuit in TTL

• Note that the input goes into the emitter, not base, of transistor Q₁
  • Q₁ acts like a pair of diodes
• If In is 1 (~V_{CC}), the base-emitter diode will be reverse-biased
  • Current flows toward Q₂’s base, turning it ON
• If In is 0, no current flows toward Q₂
  • so Q₂ is OFF
• If $Q_2$ is OFF, current from $R_2$ must flow toward $Q_3$, turning it ON
  – $Q_4$ is OFF, since no current can flow into its base
• If $Q_2$ is ON, then the effective resistance of $Q_2$ becomes small
  – which means the base-emitter voltage for $Q_3$ is below cutoff, so $Q_3$ is OFF
  – but $Q_4$ is ON
• With $Q_4$ ON, the output is close to 0V (plus a diode drop)
• With $Q_4$ OFF, the output is close to $V_{CC}$ (minus two diode drops)
• We can summarize the behavior as:

<table>
<thead>
<tr>
<th></th>
<th>$Q_1$</th>
<th>$Q_2$</th>
<th>$Q_3$</th>
<th>$Q_4$</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ON</td>
<td>OFF</td>
<td>ON</td>
<td>OFF</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>OFF</td>
<td>ON</td>
<td>OFF</td>
<td>ON</td>
<td>0</td>
</tr>
</tbody>
</table>
• As advertised, this is a NOT operation
• But due to the diode drops, the output isn’t 0 or $V_{CC}$ (usually 5V)
  – more like 0.6V for 0, and 3.6V for 1
• Note also this this circuit will draw current
• This constant power draw is characteristic of TTL
• But TTL has good features as well
  – fast switching between 0 and 1
  – ability for output to drive ~10 other TTL devices
• TTL devices typically have part numbers beginning with 7400. Some variants are:
  – 74S04: Schottky TTL: faster than standard TTL, but draws more power
  – 74LS04: Low-power Schottky
  – 74ALS04: Advanced low-power Schottky: faster and lower power than standard TTL
CMOS

- CMOS uses MOSFETs rather than bipolar junction transistors to implement logic
- As we learned earlier, FETs draw no current
  - so CMOS devices use no power except transient surges when switching output levels
    - that’s important for battery-powered devices!
  - output levels are very close to 0V and 5V
- Downside:
  - early versions were very sensitive to static (it’s easier to break a MOSFET than a bipolar junction transistor)
    - now diodes offer some input protection
  - also was originally much slower than TTL
• Part numbers for a few CMOS variants:
  – 4000: Original line
  – 74C: pin compatible with 7400 TTL, but slower
  – 74HC: as fast as TTL, but less power
  – 74HCT: interchangeable with TTL (pins and output levels)
  – 74ALCV: Advanced low voltage CMOS. Runs on 3.3V.
  – 74AVC: Advanced very low voltage CMOS. Runs on 2.5V
  – 74AUC: Advanced ultra low voltage CMOS. Runs on 1.8V
Digital Logic

• Circuits that perform logical operations (logic gates) are at the heart of much of digital electronics
  – computers, for example, make heavy use of logic gates

• The input/output characteristics and schematic symbols for some logic gates are:
  – AND gate:

<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>
- OR gate

- NOT gate:

- NAND gate:

<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>

<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>

<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 gate:**

```
A
B
C
```

<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 gate:**

```
A
B
C
```

<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>
Boolean Algebra

• In the mathematics of digital electronics, there are only two numbers (0 and 1)
• This mathematics is called “Boolean algebra”, and it has its own special notation:
• AND is represented by “•” or no symbol at all
  – $A \text{ AND } B = A \cdot B = AB$
• OR is represented by “+”
• NOT is represented by an overline: $\text{NOT } A = \overline{A}$
• Boolean algebra is:
  – communicative: $A \cdot B = B \cdot A$
    $A + B = B + A$
  – distributive: $A \cdot (B + C) = A \cdot B + A \cdot C$
and associative:

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

\[ A \cdot (B \cdot C) = (A \cdot B) \cdot C \]

- Some other rules:

\[ A \cdot 0 = 0 \]

\[ A \cdot 1 = 1 \]

\[ A + 0 = A \]

\[ A + 1 = 1 \]

\[ A \cdot A = A \]

\[ A + A = A \]

\[ A \cdot \bar{A} = 0 \]

\[ A + \bar{A} = 1 \]
DeMorgan’s Theorems

• We can often simplify Boolean expressions by using DeMorgan’s Theorems, which state that:

\[ \overline{A \cdot B} = \overline{A} + \overline{B} \]

\[ \overline{A + B} = \overline{A} \cdot \overline{B} \]

• We can prove these by using “truth tables” to compare the results of the two operations, for example:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>( \overline{A \cdot B} )</td>
<td>( \overline{A} + \overline{B} )</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
• DeMorgan’s Theorems are true no matter how many inputs are involved
  – i.e., \( A \cdot B \cdot C = \overline{A} + \overline{B} + \overline{C} \)

• They can be summarized as
  – for expressions with a bar over a combination of inputs:
    1. break up the bar to go over each input individually
    2. convert OR to AND and AND to OR

• In terms of gates, we can express DeMorgan’s Theorems as:

  NOT   \[\begin{array}{c}
  \text{NOT} \\
  \end{array}\]   =   \[\begin{array}{c}
  \text{AND} \\
  \end{array}\]

  NAND \[\begin{array}{c}
  \text{NAND} \\
  \end{array}\]   =   \[\begin{array}{c}
  \text{NAND} \\
  \end{array}\]

  NOR \[\begin{array}{c}
  \text{NOR} \\
  \end{array}\]   =   \[\begin{array}{c}
  \text{NOR} \\
  \end{array}\]
Universal Gates

• All logical operations can be performed using combinations of NAND and NOR gates
  – these are called “universal gates”

• This fact can be useful if your circuit needs an inverter (NOT gate), but all you have available is a NAND
  – you can just do:
Karnaugh Mapping

- Another way to discover simplified Boolean expressions is Karnaugh mapping
- As an example, we’ll apply this method to the following truth table:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>Output</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
• Start with a table listing some of the inputs across columns, and the rest down the rows
  – can be in any order we want, except that only one bit can change between adjacent rows or columns
• Place the output of the truth table in each space in the table
• The following will work:

<table>
<thead>
<tr>
<th>$AB \to C$ ↓</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
• The shaded cells in the map show that all the 1’s are grouped into a cluster
  – don’t be confused by the fact that there seem to be two clusters on different sides of the table – Karnaugh maps don’t have any edges, so we just wrap around the ends of the table
• Whenever this happens, it means we can simplify the logic
• We can divide our cluster into three smaller clusters, shown below

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

Cluster 1

Cluster 2

Cluster 3
• From cluster 1, we see that the output is 1 whenever $A$ and $B$ are both 0
  – i.e., $\text{Output} = \overline{A} \cdot \overline{B}$
• From cluster 2, we see that the output is 1 whenever $B$ is 0 and $C$ is 0
  – i.e., $\text{Output} = \overline{B} \cdot \overline{C}$
• Finally from cluster 3, we see that the output is 1 whenever $A$ is 1 and $C$ is 0
  – i.e., $\text{Output} = A \cdot \overline{C}$
• We can put this together to find the Boolean expression corresponding to our truth table:

\[
\text{Output} = \overline{A} \cdot \overline{B} + \overline{C} \cdot \overline{B} + A \cdot \overline{C}
\]

\[
= (\overline{A} + \overline{C}) \cdot \overline{B} + A \cdot \overline{C}
\]
Now that we’ve found a Boolean expression for our truth table, we can readily design a circuit that behaves the way we want: