Unit 2

Digital Circuits (Logic)
Moving from voltages to 1's and 0's...

**ANALOG VS. DIGITAL**
Signal Types

• Analog signal
  – Continuous time signal where each voltage level has a unique meaning
  – Most information types are inherently analog

• Digital signal
  – Continuous signal where voltage levels are mapped into 2 ranges meaning 0 or 1
  – Possible to convert a single analog signal to a set of digital signals
Signals and Meaning

Each voltage value has unique meaning.

Analog

- 0.0 V
- 5.0 V

Digital

- 0.0 V
- 8.0 V
- 2.0 V
- 5.0 V

Logic 0
Logic 1
Illegal
Threshold Range

Each voltage maps to '0' or '1'
(There is a small illegal range where meaning is undefined since threshold can vary based on temperature, small variations in manufacturing, etc.)
Analog vs. Digital

USC students used to program analog computers!
Analog vs. Digital

• Analog Advantages
  – Can be easier to build simple systems
   • AM Radio

• Digital Advantages
  – Not affected by small changes (noise) in the signal
  – Repeatable/Reproducible
  – The above make it easier to build large, complex systems
A Brief History

COMPUTERS AND SWITCHING TECHNOLOGY
Electronic Computers

- Replaced mechanical computation engines
- Electronic computers require some kind of "switching" technology
  - Initially that was the vacuum tube
- 1945 - ENIAC was one of the first, fully electronic computers
- Used thousands of vacuum tubes as fundamental switching (on/off) technology
- Weighed 30 tons, required 15,000 square feet, and maximum size number was 10 decimal digits (i.e. ±9,999,999,999)
- Still required some patch panels (wire plugs) to configure it
Vacuum Tube Technology

- Digital, electronic computers use some sort of voltage controlled switch (on/off)
- Looks like a light bulb
- Usually 3 nodes
  - 1 node serves as the switch value allowing current to flow between the other 2 nodes (on) or preventing current flow between the other 2 nodes (off)
  - Example: if the switch input voltage is 5V, then current is allowed to flow between the other nodes

![Diagram of Vacuum Tube with switch input and current flow](image)
Vacuum Tube Disadvantages

• Relatively large
  – Especially when you need 19,000 to make 1 computer

• Unreliable
  – Can burn out just like a light bulb

• Dissipate a lot of heat (power)
Transistor

- Another switching device
- Invented by Bell Labs in 1948
- Uses semiconductor materials (silicon)
- Much smaller, faster, more reliable (doesn't burn out), and dissipated less power

Individual Transistors
(About the size of your fingertip)
Moore's Law & Transistors

• **Moore's Law** = Number of transistors able to be fabricated on a chip will double every 1.5 – 2 years (i.e. exponential growth)
  – Achieved by shrinking the transistor structure
  – However, we are approaching the physical limitations of this shrinking
  – 53% compound annual growth rate over 50 years
    • No other technology has grown so fast so long

• Transistors are the fundamental building block of computer HW
  – Switching devices: Can conduct [on = 1] or not-conduct [off = 0] based on an input voltage
How Does a Transistor Work

• Transistor inner workings
  – http://www.youtube.com/watch?v=IcrBqCFLHIY
NMOS Transistor Physics

- Let's review what we saw in the video...
- Transistor is started by implanting two n-type silicon areas, separated by p-type.
NMOS Transistor Physics

- A thin, insulator layer (silicon dioxide or just "oxide") is placed over the silicon between source and drain.
NMOS Transistor Physics

- A thin, insulator layer (silicon dioxide or just "oxide") is placed over the silicon between source and drain.
- Conductive polysilicon material is layered over the oxide to form the gate input.
NMOS Transistor Physics

- Positive voltage (charge) at the gate input repels the extra positive charges in the p-type silicon.
- Result is a negative-charge channel between the source input and drain.
NMOS Transistor Physics

- Electrons can flow through the negative channel from the source input to the drain output.
- The transistor is "on".
NMOS Transistor Physics

- If a low voltage (negative charge) is placed on the gate, no channel will develop and no current will flow.
- The transistor is "off".

No negative channel between source and drain = No current flow
View of a Transistor

• Cross-section of transistors on an IC
• Moore's Law is founded on our ability to keep shrinking transistor sizes
  – Gate/channel width shrinks
  – Gate oxide shrinks
• Transistor feature size is referred to as the implementation "technology node"
Minimum Feature Size
Intel Processor Trends

1971 – Intel 4004
1000 transistors
Max 4K-bits addressable memory
1 MHz operation

- 2nd Gen. Intel Core i7 Extreme Processor for desktops launched in Q4 of 2012
- #cores/#threads: 6/12
- Technology node: 32nm
- Clock speed: 3.5 GHz
- Transistor count: Over one billion
- Cache: 15MB
- Addressable memory: 64GB
- Size: 52.5mm by 45.0mm mm²
ARM Cortex A15

ARM Cortex A15 in 2011 to 2013

- 4 cores per cluster, two clusters per chip
- Technology node: 22nm
- Clock speed: 2.5 GHz
- Transistor count: Over one billion
- Cache: Up to 4MB per cluster
- Addressable memory: up to 1TB
- Size: 52.5mm by 45.0mm
DIGITAL LOGIC GATES
Transistors and Logic

• Transistors act as switches (on or off)
• Logic operations (AND / OR) formed by connecting them in specific patterns
  – Series Connection
  – Parallel Connection

Series Connection
S1 AND S2 must be on for A to be connected to B

Parallel Connection
S1 OR S2 must be on for A to be connected to B
Digital Logic

- Forms the basic processing circuits for digital signals (i.e. 1's and 0's)
- Digital Logic still abstracts many of the physical issues (voltage, current, parasitics, etc.) dealt with in the study of integrated circuits
  - An alarm should sound if the key is in the ignition AND your seatbelt is NOT fastened
  - If the sensor voltage rises above 3 volts create a conductive channel to excite the ignition sensor...

<table>
<thead>
<tr>
<th>Computer Architecture (Functional Blocks)</th>
<th>Digital Logic</th>
<th>Integrated Circuits (Transistors)</th>
</tr>
</thead>
</table>

**AND**

**OR**

**NOT**
Gates

- Each logical operation (AND, OR, NOT) can be implemented in circuit form using the corresponding logic gate.
AND Gates

- An AND gate outputs a '1' (true) if ALL inputs are '1' (true)
- Gates can have several inputs
- Behavior can be shown in a truth table (listing all possible input combinations and the corresponding output)

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>F</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>

2-input AND

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>F</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>0</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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

3-input AND
OR Gates

• An OR gate outputs a '1' (true) if ANY input is '1' (true)
• Gates can also have several inputs

\[
\begin{array}{ccc}
X & Y & F \\
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{ccc}
X & Y & F \\
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

2-input OR

\[
\begin{array}{ccc}
X & Y & Z & F \\
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 \\
1 & 1 & 1 & 1 \\
\end{array}
\]

3-input OR
Buffer & NOT (Inverter) Gate

- A Buffer simply passes a digital value
  - But strengthens it electrically (e.g. boosts 3.7V closer to 5V)
- A NOT (aka "inverter") gate inverts a digital signal to its opposite value (i.e. flips a bit)

\[
\begin{array}{c|c}
X & F \\
\hline
0 & 0 \\
1 & 1 \\
\end{array}
\]

\[
\begin{array}{c|c}
X & F \\
\hline
1 & 0 \\
0 & 1 \\
\end{array}
\]

the "bubble" (logically performs the inversion)
How Do You Build an Inverter (1)?

- A simple (though maybe not ideal) method to build an inverter is to place a transistor in series with a resistor
  - Input to inverter is input to transistor
  - Output of inverter is node between resistor and transistor
- We can model the transistor as a resistor
- Develop an equation for $V_{out}$

$$V_{out} = V_{dd} \cdot \frac{R_{trans}}{R_{trans} + R_{pullup}}$$
How Do You Build an Inverter (2)?

• First, estimate the resistance of $R_{\text{trans}}$ if $V_{\text{in}}$ is 0 (low) then again if $V_{\text{in}}$ is 1 (high-voltage).
• Use that estimate in your equation for $V_{\text{out}}$ to determine the output voltage.

\[ V_{\text{out}} = \frac{V_{\text{dd}}}{R_{\text{pullup}} + R_{\text{trans}}} \]

\[ V_{\text{out}} = \frac{(V_{\text{dd}})}{R_{\text{pullup}} + R_{\text{trans}}} \]
NAND and NOR Gates

- Inverted versions of the AND and OR gate

\[ Z = \overline{X \cdot Y} \]

\[ Z = \overline{X + Y} \]

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</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>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</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>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

AND
True if NOT ALL inputs are true

NAND

OR

NOR
True if NOT ANY input is true
XOR and XNOR Gates

➢ Exclusive OR gate. Outputs a '1' if either input is a '1', but not both.

\[
F = X \oplus Y
\]

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>F</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>

XOR

True if an odd # of inputs are true
= True if inputs are different

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>F</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>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

XNOR

\[
F = \overline{X \oplus Y}
\]

True if an even # of inputs are true
= True if inputs are same

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>F</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>1</td>
</tr>
</tbody>
</table>
Logic Example

0 A
0 B
1 C
0 D

1
0
0
1

F 1
Logic Example

Diagram:

- A is 0
- B is 1
- C is 0
- D is 0
- F is 0
DIGITAL DESIGN GOALS

Speed, area, and power
Digital Design Goals

• When designing a circuit, we want to optimize for the following three things:
  – Area or Circuit Size (minimize)
  – Speed (maximize) or Delay (minimize)
  – Power (minimize)

• Can usually only optimize one or two of the 3
  – There is a huge trade space! This is what engineering is all about!
Minimizing Circuit Area

• Approaches:
  – Reduce the number of gates used to implement a circuit
  – Reduce the number of inputs to each gate
    • In general a gate with n inputs requires 2*n transistors to implement

• Simplify logic expressions (usually by factoring and then canceling terms) to reduce the number of gates
  – We'll learn more about this soon
Maximizing Speed

• Speed is affected by:
  – Levels of logic (path length)
  – Gate type
  – Number of inputs (fan-in) to the gate
  – Number of outputs a gate connects to (fan-out)
  – Feature size and implementation technology
Delay Example

• Levels of Logic = Max. # of gates on any path from input to output

Change in D, C, or A must propagate through 4 levels of gates
Gate Delays

- Order the gate types in terms of fastest to slowest?
- Typical gate delay for a 2-input NAND or NOR is under a 100 ps.
Logical Operations Summary

• All digital circuits can be described using AND, OR, and NOT
  – Note: You'll learn in future courses that digital circuits can be described with any of the following sets:
    • \{AND, NOT\}, \{OR, NOT\}, \{NAND only\}, or \{NOR only\}

• Normal convention: 1 = true / 0 = false

• A logic circuit takes some digital inputs and transforms each possible input combination to a desired output values

Trivia-of-the-day: The Apollo Guidance Computer that controlled the lunar spacecraft in 1969 was built out of 8,400 3-input NOR gates.
Storing bits

SEQUENTIAL LOGIC
Combinational vs. Sequential Logic

- All logic is categorized into 2 groups
  - Combinational logic:
    - Outputs = f(current inputs)
  - Sequential Logic
    - Outputs = f(current + past inputs)
    - Sequential logic has the notion of “memory” (remembering inputs or events that happened in the past)
Combinational vs. Sequential

Outputs depend only on current outputs

Outputs depend on current inputs and previous inputs (previous inputs summarized via state)
Combinational Example: Staircase Light Switch

Whether or not the light is on is only dependent on the current position of the switches

<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
<th>Light</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>
Water Tank Problem

• Build a control system for a pump to keep the tank from going empty
Sequential Devices (Registers)

- AND, OR, NOT, NAND, and other gates are known as **combinational logic**
  - Outputs only depend on what the inputs are **right now**, not one second ago
  - This implies they have no "memory" (can't remember a value)

- **Sequential logic** devices provide the ability to retain or "remember" a value by itself (even after the input is changed or removed)
  - Outputs can depend on the current inputs, and previous states of the circuit (stored values.)
  - Usually have a controlling signal that indicates when the device should update the value it is remembering vs. when it should simply remember that value
  - This controlling signal is usually the "clock" signal
 Registers

- Registers are the most common sequential device
- Registers sample the data input (D) on the edge of a clock pulse (CP) and stores that value at the output (Q)
- Analogy: Taking a picture with your phone...when you press a button (clock pulse) the camera samples the scene (input) and remembers/saves it as a snapshot (output). A register can only save 1 value (i.e. the last picture taken)

\[
\begin{align*}
&\text{D} & \text{Q} \\
&\text{CP} & \\
&\text{Data Input} & \text{Data Output} \\
&\text{(could be many bits)} & \text{(could be many bits)}
\end{align*}
\]

The clock pulse (positive edge) here...

...causes \( q(t) \) to sample and hold the current \( d(t) \) value

\[
\begin{align*}
&d(t) & q(t) \\
&t = 0 \text{ ns} & \text{unk} & d(1) & d(2) & d(3) & d(4) & d(5) & d(6) & d(7) & d(8) & d(9) & d(10) & d(11) & d(12) \\
&t = 1 \text{ ns} & d(1) & d(2) & d(3) & d(4) & d(5) & d(6) & d(7) & \text{unk} & d(9) & d(10) & d(11) & d(12) \\
&t = 5 \text{ ns} & d(5) & d(6) & d(7) & d(8) & d(9) & d(10) & d(11) & d(12) & \text{unk} & d(1) & d(2) & d(3) \\
&t = 7 \text{ ns} & \text{unk} & d(1) & d(2) & d(3) & d(4) & d(5) & d(6) & d(7) & d(8) & d(9) & d(10) & d(11) \\
&t = 10 \text{ ns} & d(1) & d(2) & d(3) & d(4) & d(5) & d(6) & d(7) & d(8) & d(9) & d(10) & d(11) & d(12)
\end{align*}
\]
Flip-Flops

- Flip-flops are the building blocks of registers
  - 1 Flip-flop PER bit of input/output
  - There are many kinds of flip-flops but the most common is the D- (Data) Flip-flop (a.k.a. D-FF)

- D Flip-flop triggers on the clock edge and captures the D-value at that instant and causes Q to remember it until the next edge
  - Positive Edge: instant the clock transition from low to high (0 to 1)
Registers and Flip-flops

• A register is simply a group of D flip-flops that all trigger on a single clock pulse.

<table>
<thead>
<tr>
<th>CLK</th>
<th>( Q_{t+1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( Q_t )</td>
</tr>
<tr>
<td>1</td>
<td>( Q_t )</td>
</tr>
<tr>
<td>↑</td>
<td>( D_t )</td>
</tr>
</tbody>
</table>

Steady level of 0 or 1
Positive Edge
Pulses and Clocks

- Registers need an edge to trigger.
- We can generate pulses at specific times (creating an irregular pattern) when we know the data we want has arrived.
- Other registers in our hardware should trigger at a regular interval.
- For that we use a clock signal...
  - Alternating high/low voltage pulse train.
  - Controls the ordering and timing of operations performed in the processor.
  - 1 cycle is usually measured from rising/positive edge to rising/positive edge.
- Clock frequency (F) = # of cycles per second.
- Clock Period (T) = 1 / Freq.

![Clock Pulses](image)

Clock Pulsed

Clock Signal

1 (5V)
0 (0V)

1 cycle
Op. 1
Op. 2
Op. 3

2.8 GHz
= 2.8*10^9 cycles per second
= 0.357 ns/cycle

Processor

Clock Pulses
Combinational vs. Sequential

• Sequential logic (i.e. registers) is used to store values
  – Each register is analogous to a variable in your software program (a variable stores a number until you need it)

• Combinational logic is used to process bits (i.e. perform operations on values
  – Analogous to operators (+, -, *) in your software program