Digital Logic: From Transistors to Gates

Textbook Chapter 3
Announcements

- Auxiliary Website is up
  - https://classes.soe.ucsc.edu/cmpe012/Fall15/
- HW #1 extended until Tuesday
- Too Many Lab Swaps for me to handle myself
  - Find a partner and fill out google form. I will send confirmation for the switch
  - http://goo.gl/forms/d5ul67AwaO
- If you are still on the waitlist fill out the form up front before you leave
MSI

Mon 2-3:10 PM
Tu 12-1:15 PM
Wed 12:30-1:40 PM, 3:30-4:40 PM
Th 4-5:15 PM
Fri 3:30-4:40 PM
The Transistor

- Transistor: building block of computers
- Microprocessors contain tons of transistors
  - Intel Core i7-5960X (2015): 2.6 Billion
  - Qualcomm Snapdragon 810 (2015): multi billions
  - AMD 6-core Opteron (2009): 904 million
  - Intel Core i7 Quad (2008): 731 million
  - Intel Itanium 2 (2003): 220 million
  - Intel Pentium 4 (2000): 42 million
  - Intel 4004 (1971): 2300
The Transistor: Past and Present

- 1947 first point-contact transistor
Moore's Law

Microprocessor Transistor Counts 1971-2011 & Moore's Law

"The number of transistors on a chip will double about every 18 months."

Maxwell James Dunne – Fall 2015
What Is a Transistor?

- A switch, which can close between the source and the drain
- Changing the voltage of the gate lets you change the current flow between the source and drain (closing or opening the switch)
- Think of a light switch, the gate is the switch that allows electricity to flow from the source to the drain
Metal-Oxide-Semiconductor transistor

NMOS Transistor (n-channel MOSFET)

Silicon Dioxide (insulator)

source

gate

drain

gate electrode

p-type silicon

n-type silicon

Silicon Substrate
FinFET

- Higher performance at lower voltages
- Less power

(a) Conventional planar transistor

(b) FinFET
What is a transistor?

- Logically, each transistor is used as a switch
- Combined to implement logic functions
  - AND, OR, NOT
- Combined to build higher-level structures
  - Adder, multiplexer, decoder, register, ...
- Combined to build a processor
  - LC-3, Core i7, A9, etc
n-type MOS transistor

n-type MOS (nMOS)

- when Gate has **positive** voltage, short circuit between #1 and #2 (switch **closed**)
- when Gate has **zero** voltage, open circuit between #1 and #2 (switch **open**)

Terminal #2 must be connected to GND (0V).
p-type MOS transistor

p-type is *complementary* to n-type

- when Gate has positive voltage, open circuit between #1 and #2 (switch open)
- when Gate has zero voltage, short circuit between #1 and #2 (switch closed)

Terminal #1 must be connected to +2.9V in this example.
Digital Values for Analog Signals

- Use the switch behavior of MOS transistors to implement logical functions: AND, OR, NOT
- Digital symbols:
  - We assign a range of analog voltages to each digital (logic) symbol
  - Assignment of voltage ranges depends on electrical properties of transistors being used
CMOS circuit

- CMOS is Complementary Metal Oxide Semiconductor
- Uses both n-type and p-type MOS transistors
  - p-type (pMOS)
    - Attached to + voltage
    - Pulls output voltage UP when input is zero
  - n-type (nMOS)
    - Attached to GND
    - Pulls output voltage DOWN when input is one
- Faster than using just one type
Truth Table

- The most basic representation of a logic function
- It is a perfect induction proof - Lists the output for all possible input combinations
- How many rows of the truth table needed?

\[ 2^n \]
Truth Table: Inverter

- Inverted signals are denoted with an overbar
- Or with a prime symbol
  - $A'$
- Or with a bubble in a circuit diagram

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$Y = A'$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Maxwell James Dunne – Fall 2015
Inverter (NOT gate)

\[
\begin{array}{c|c}
\text{In} & \text{Out} \\
\hline
0 \text{ V} & 2.9 \text{ V} \\
2.9 \text{ V} & 0 \text{ V} \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{In} & \text{Out} \\
\hline
0 & 1 \\
1 & 0 \\
\end{array}
\]
Truth Table: AND Gate

- The result of an AND operation is 1 if and only if all inputs are 1.
- Depict AND by the multiplication symbol
  - $A \cdot B$
- Or by lumping the signals together
  - $AB$
- We don’t really build these gates...

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$B$</td>
<td>$Y = A \cdot B$</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>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
NAND gate (NOT-AND)

Note: Parallel structure on top, serial on bottom.
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>

Add an inverter to a NAND.
Truth Table: OR Gate

- The result of an OR operation is 1 if and only if any inputs are 1
- Depict OR by the addition symbol
  \[ A + B \]

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B</td>
<td>Y = A + B</td>
</tr>
<tr>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>1</td>
</tr>
<tr>
<td>1 0</td>
<td>1</td>
</tr>
<tr>
<td>1 1</td>
<td>1</td>
</tr>
</tbody>
</table>
NOR Gate: NOT-OR

Note: Serial structure on top, parallel on bottom.
OR 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>

Add an inverter to a NOR gate.
Digital Logic: Boolean Algebra and Gates

Textbook Chapter 3
Basic Logic Gates

- NOT
- OR
- NOR
- AND
- NAND
- XOR
\[
\begin{array}{c}
\text{a} \\
\text{b} \\
\text{c} \\
\text{d}
\end{array}
\]

\[
\begin{array}{c|c|c|c|c}
\text{a} & \text{b} & \text{c} & \text{d} & < \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 & 1 \\
1 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 0 \\
\end{array}
\]
Truth Table

- The most basic representation of a logic function
- Lists the output for all possible input combinations
- How many rows of the truth table needed?
- Can get big very fast
Truth Table: Inverter

- Inverted signals are denoted with an overbar
- Or with a prime symbol
  - $A'$

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Truth Table: AND Gate

- The result of an AND operation is 1 if and only if all inputs are 1
- Depict AND by the multiplication symbol
  \(- A \cdot B\)
- Or by lumping the signals together
  \(- A \bar{B} \bar{C}\)
- We don’t really build these gates...

\[
(\overline{A + \overline{B}})(A + \overline{B})(\overline{A + B})
\]

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Truth Table: OR Gate

- The result of an OR operation is 1 if and only if any inputs are 1.
- Depict OR by the addition symbol: \( A + B \)

<table>
<thead>
<tr>
<th>( F = A + B )</th>
<th>( A )</th>
<th>( B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</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>
Sum of Products

• How do you get from a truth table to a logic expression?
• Sum of products is standard way of synthesizing simple circuits
• Procedure:
  1. Find the rows with the ‘1’ output
  2. Write the product-form expression for the inputs in that row (0= inverted, 1= normal)
  3. Combine the products in step 2 into a sum (OR the results of step 2)
Sum of Products

1. Find the rows with the ‘1’ output
2. Write the product-form expression for the inputs in that row (0=inverted, 1=normal)
3. Combine the products in step 2 into a sum (OR the results of step 2)

\[ Y = A'B + AB' \]
Product of Sums

• Procedure:
  1. Find the rows with the ‘0’ output
  2. Write the sum-form expression for the inputs in that row (0=normal, 1=inverted)
  3. Combine the sums in step 2 into a product (AND the results of step 2)

• Note: we treat 0 and 1 reverse than for SoP
Examples of SoP and PoS

\[
\begin{align*}
A & B & C & F \\
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 \\
\end{align*}
\]

\[
\bar{A} \bar{B} \bar{C} + A \bar{B} \bar{C} + \bar{A} B \bar{C} + \\
A B C \quad (A + B + \bar{C}) \quad (A + \bar{B} + C) \quad (A + \bar{B} + \bar{C}) \\
\bar{A} + \bar{B} + C
\]
Examples of SoP and PoS

\[ \begin{array}{c|ccc}
A & B & C & \text{Output} \\
\hline
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 \\
\end{array} \]

\[ \overline{AB} + \overline{A\bar{B}} \]

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

\[ (A + \bar{B}) \]

\[ (\bar{A} + \bar{B}) \]
De Morgan’s Laws

• “Break the line, change the sign”

• Two laws:
  - $A' + B' = (AB)'$
    - This is the NAND gate
  - $A' B' = (A+B)'$
    - This is the NOR gate
De Morgan’s Laws

\[(A + B)' = A'B' \quad \text{conversely} \quad (AB)' = A' + B'\]

“Break the line, change the sign”

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>A+B</th>
<th></th>
<th>A</th>
<th>B</th>
<th>A⋅B</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>1</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>1</td>
<td>0</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>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
De Morgan’s Laws

\[(A + B)' = A'B' \quad \text{conversely} \quad (AB)' = A' + B'\]

“Break the line, change the sign”

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>AB</th>
<th>(\overline{AB})</th>
<th>(\overline{A})</th>
<th>(\overline{B})</th>
<th>(\overline{A} + \overline{B})</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>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
De Morgan’s Laws and SOP

- Generate equivalent circuits
  - NAND/NAND
  - NOR/NOR
- We prefer NAND/NAND circuits
  - Same transistor count as NOR
  - NANDs are faster
Logical Completeness

Can implement **ANY** truth table with AND, OR, NOT.

<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>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>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>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>0</td>
</tr>
</tbody>
</table>

1. AND combinations that yield a "1" in the truth table.
2. OR the results of the AND gates.
3. Is not necessarily a minimal solution.
More Than Two Inputs?

- AND and OR gates can take any number of inputs
  - AND gives 1 if all inputs are 1
  - OR gives 1 if any input is 1
- NAND?? NOR??
  - Not associative!
Two-Way Multiplexer
Two-Way Multiplexer

2-way multiplexer: the output is equal to one of the two inputs, based on a selector

<table>
<thead>
<tr>
<th>$S$</th>
<th>$A$</th>
<th>$B$</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>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>

Sum of Products Expression

$$Y = S'A'B' + S'A'B + S'AB' + S'AB + SA'B' + SA'B + SAB' + SAB$$
Four-Way Multiplexer

- \( n \)-bit selector and \( 2^n \) inputs, one output
  - output equals one of the inputs, depending on selector
- "Four-to-one mux"
Two-to-Four Decoder

- $n$ inputs, $2^n$ outputs
  - exactly one output is 1 for each possible input pattern
- Generates a walking-ones pattern
- Uses:
  - Convert memory or register address to a control line
  - Convert an opcode to one of $n$ control lines
  - We will get to this in the LC-3 material
Binary Addition and Half-Adder

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 0 = 1$
- $1 + 1 = ...$

A half-adder can add 2 bits and produces a sum and carry signal

- Sum = $A \ xor \ B$
- Carry = $AB$
One-Bit Full Adder

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C&lt;sub&gt;in&lt;/sub&gt;</th>
<th>C&lt;sub&gt;out&lt;/sub&gt;</th>
<th>S</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>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</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>
<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>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Four-Bit Full Adder

Ripple-carry adder
Masking

- Want to look only at certain bits of a binary word
- Use a mask to remove the uninteresting bits
- It is a bitwise AND operation with the MASK
- Example:
  
  | Value | 1001 1101 |
  | MASK  | 0000 1111 |
  | Result| 0000 1101 |
Axioms of Boolean Algebra

- \(0 \cdot 0 = 0\)
- \(1 + 1 = 1\)
- \(1 \cdot 1 = 1\)
- \(0 + 0 = 0\)
- \(0 \cdot 1 = 1 \cdot 0 = 0\)
- \(1 + 0 = 0 + 1 = 1\)
- if \(x = 0\) then \(x' = 1\)
- if \(x = 1\) then \(x' = 0\)
Single-Variable Theorems

- \( x \cdot 0 = 0 \)
- \( x + 1 = 1 \)
- \( x \cdot 1 = x \)
- \( x + 0 = x \)
- \( x \cdot x = x \)
- \( x + x = 0 \)
- \( x \cdot x' = 1 \)
- \( x + x' = x \)
- \( (x')' = \)
Properties of Boolean Algebra

- **Commutative**
  - $x \cdot y = y \cdot x$
  - $x + y = y + x$

- **Associative**
  - $x \cdot (y \cdot z) = (x \cdot y) \cdot z$
  - $x + (y + z) = (x + y) + z$

- **Distributive**
  - $x \cdot (y + z) = x \cdot y + x \cdot z$
  - $x + y \cdot z = (x + y) \cdot (x + z)$
Properties of Boolean Algebra

- **Absorption**
  
  \[ x + x \cdot y = x(1+y) = x \]
  
  \[ x \cdot (x + y) = xx + xy = x + xy = x(1+y) = x \]

- **Combining**
  
  \[ x \cdot y + x \cdot y' = x(y + y') = x \]
  
  \[ -(x + y) \cdot (x + y') = xx + xy' + yx + yy' \]
  
  \[ = x + xy' + yx + 0 \]
  
  \[ = x + x(y' + y) \]
  
  \[ = x + x(1) \]
  
  \[ = x + x = x \]
Properties of Boolean Algebra

- **De Morgan’s Laws**
  - \(-(x \cdot y)’ = x’ + y’\)
  - \-(x + y)’ = x’ \cdot y’\)

- **Other**
  - \-x + x’ \cdot y = (x + x’) \cdot (x + y) = x + y\)
  - \-x \cdot (x’ + y) = x \cdot x’ + x \cdot y = x \cdot y\)
### Logic Minimization

<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>Y</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>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</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>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>

Sum of products

\[ Y = A'B'C' + A'BC + AB'C' + ABC' \]

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

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

\[ = A'B(1) + AC'(1) \]

\[ = A'B + AC' \]
Combinational vs. Sequential

Two types of “combination” locks

**Combinational**
Success depends only on the values, not the order in which they are set.

**Sequential**
Success depends on the sequence of values (e.g., R-13, L-22, R-3).
Combinational vs. Sequential

• Combinational circuit
  – Always gives the same output for a given set of inputs
  – Example: Adder always generates sum and carry, regardless of previous inputs

• Sequential circuit
  – Remembers previous input
  – Output depends on state and input
Reset-Set (RS) Latch – or SR

- Two inputs: Set and Reset
- Set to 0 one of the two inputs at a time to store a value, S sets, R clears
- The transition to 00 generates an undefined output

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Restricted combination</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Q = 1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Q = 0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Keep state</td>
</tr>
</tbody>
</table>
R-S Latch
D-Latch

- D-latch (D for data) is a gated RS latch
- Two inputs: D (data) and WE (write enable)
D-Latch: Timing Diagram
D-Flip-Flop

- Two D-latches hooked together
- Connect one latch to the inverted clock
- D-flip-flop is edge-triggered (changes only on the edge of the clock)
- Also called “edge-triggered d-latch”
- This is an example of an edge triggered DFF with an enable
D-Flip Flop: Timing Diagram

Diagram showing the timing of Ck, WE, D, and Q signals over time.
D-Flip Flops in a Pipeline

D

Q1

Q2

Ck

WE

E

D

Q

Q1

Q2

Ck

time

D

Q

Q1

Q2

Ck

time

D

Q

Q1

Q2

Ck

time

D

Q

Q1

Q2

Ck

time

D

Q

Q1

Q2

Ck

time

D

Q

Q1

Q2

Ck

time
Register

- A register stores a multi-bit value
- Common WE which latches the n-bit value
Memory

Now that we know how to store bits, we can build a memory – a logical $k \times m$ array of stored bits.

**Address Space:**
number of locations
(usually a power of 2)

**Addressability:**
number of bits per location
(e.g., byte-addressable)
2^2 x 3 Memory

- address
- word select
- word WE
- input bits
- write enable
- address decoder
- output bits
State Machine

The basic type of sequential circuit

- Combines combinational logic with storage
- “Remembers” state, and changes output (and state) based on inputs and current state
Example of sequential machine

Combinational lock R13, L22, R3:
Representing Multi-bit Values

- Number bits from right (0) to left (n-1)
  - just a convention -- could be left to right, but must be consistent
- Use brackets to denote range:
  D[l:r] denotes bit l to bit r, from left to right

\[
A = 0101001101010101
\]

\[
\]

May also see A<14:9>, especially in hardware block diagrams.