Lesson 1: Introduction to Digital Logic Design

Why Digital Systems?
- Accurate — depending on number of digits used
  - CD Music is digital — Vinyl Records were analog
  - DVD Video and Audio
  - mp3 (quality depends on sampling/amount of bits)
- Reliable
  - Error Correction Capabilities
  - Discrete Values with Large Noise Margin
- Technology
  - can be implemented as fast, cheap CMOS semiconductors

Numbering systems
- Western World — Decimal or base 10
  - The system that we all know and take for granted
  - 10 probably picked because of the number of fingers on human hands
- Mayans — Vigesimal or base 20
  - had concept of zero and had a modern positional notation
  - Allowed for representing a large range (very small to very large numbers)
  - positional notation allows for long arithmetic
- Computers and Digital Systems — Binary or base 2
  - easy to implement physically — high or low voltage on wire
  - allows for use of Boolean math and philosopher’s logic
    - true or false = high or low = one or zero = on or off
- Hexadecimal System — Base 16
  - Can quickly convert large binary numbers to hexadecimal and back by inspection
  - One Hexadecimal digit represents four binary digits (0 – 9, A – F)

Binary Codes, BCD — clock example
- Binary Coded Decimal (8-4-2-1 weighted code)
  - used with simple LED displays (watch display, etc)

Sample Problems 1 – 5 min
- Convert 1000.012 to decimal
- Convert 10110011102 to Hex
- Convert 25.2510 to binary

Lesson 1-b: Logic Gates
Basic Operations - Inverter

- Inversion operation (AKA the complement)
  - operation performed on only single variable
  - indicated by a prime (') or overbar – (prime is easier to use)
  - the inversion of 1 is 0 and the inversion of 0 is 1
  - inverter consists of two transistors in CMOS (don’t need to know this for test)

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

Basic Operations – Logical AND

- AND function
  - operation performed on two or more boolean variables
  - output is one if and only if both inputs are one
  - indicated by a multiplication symbol (although not multiplication)
    - “*” can be used, or
    - two adjacent variables are assumed to be ANDed

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Z</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>

Basic Operations – Logical OR

- OR function
  - operation performed on two or more boolean variables
  - output is one if either or both of the inputs is one
  - indicated by a addition symbol (although not addition)
    - “+” can be used

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Z</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>

Sample Problems 2 – 5 min

- Convert a NAND gate into an INVERTER
  - Hint: no gates necessary

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Z</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>

Exclusive Or Logic gate

\[ X \oplus Y = X \cdot Y' + X' \cdot Y \]
Exclusive NOR Logic gate (XNOR)

\[ X \equiv Y = X \land Y + X' \land Y' = (X \oplus Y)' \]

A B Z
\begin{array}{ccc}
0 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}

Also known as an Equivalence Operation or Bit Compare

Sample Problem 3 – 5 min
- Convert a XOR gate into an INVERTER
- Hint: no gates necessary

A B Z
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
\end{array}

Sample Problem 4 – 5 min
- Convert a XOR gate into a BUFFER
- Hint: no gates necessary

A B Z
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
\end{array}

Logic Networks

\[ F(a,b,c) = a' \cdot b + a \cdot b \cdot c + b \cdot c' \]

Sample Problem 5 – 5 min
- Truth Table
- Logic Network Circuit Diagram

Truth Tables

\[ F(A,B,C) = AB + C \]

\[
\begin{array}{cccccc}
A & B & C & A \cdot B & F \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

Sample Problem 5 – 5 min
- 1. Truth Table
- 2. Logic Network Circuit Diagram
Lesson 1-c: Maxterms and Minterms

Design by Truth Table (based on 1’s of table)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Z</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>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>

\[ Z = A'BC + A'BC' + ABC \]
\[ Z = AB + BC \]

Minterm expansion in m-notation

\[ Z = A'BC + A'BC' + ABC \]
rewritten in m-notation
\[ Z(a,b,c) = m_3 + m_6 + m_7 \]

Maxterm expansion

\[ Z = (a+b+c)(a+b'+c)(a'+b+c)(a'+b+c') \]
rewritten as maxterm expansion
\[ Z(a,b,c) = M_0 M_1 M_4 M_5 \]

Lesson 1-d: Boolean Algebra Theorems

Theorems

- Basic Theorem:
  \[ X + 0 = X \]
  \[ X + 1 = X \]
- Idempotent Law:
  \[ X + X = X \]
  \[ X * X = X \]
- Involution Law:
  \[ (X')' = X \]
- Laws of Complementarity
  \[ X + X' = 1 \]
  \[ X * X' = 0 \]
Theorems (2)

- **Commutative Law:**
  \[ X \cdot Y = Y \cdot X \quad X + Y = Y + X \]

- **Associative Law:**
  \[ X \cdot (Y \cdot Z) = (X \cdot Y) \cdot Z \quad (X + Y) + Z = X + (Y + Z) \]

- **Distributive Law:**
  \[ X \cdot (Y + Z) = X \cdot Y + X \cdot Z \quad X + (Y \cdot Z) = (X + Y) \cdot (X + Z) \]

- **De Morgan's Law:**
  \[ (X_1 + X_2 + X_3)' = X_1' \cdot X_2' \cdot X_3' \]
  \[ (X_1 \cdot X_2 \cdot X_3)' = X_1' + X_2' + X_3' \]

Theorems (3) - Simplification

1. \[ X \cdot Y + X \cdot Y' = X \]
2. \[ X + X \cdot Y = X \]
3. \[ (X + Y)' = X' \cdot Y' \cdot X' \cdot Y' \]

Sample Problems 6 – 5 min

\[ F = A \cdot B \cdot C + C' + B' \]

\[ X \cdot Y' + Y = X + Y \]

\[ F = AB + C' + B' \]

Lesson 1-e: 4-bit Adder/Subtractor

Lab - Full Adder Cell

- adds three 1 bit numbers
- two numbers
- one carry-in from previous stage
- provides 1 bit sum and carry out

<table>
<thead>
<tr>
<th>A</th>
<th>B'</th>
<th>Cin</th>
<th>Sum</th>
<th>Cout</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>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</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>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>1</td>
</tr>
</tbody>
</table>

Binary Addition

Addition Table

\[
\begin{array}{c|c|c|c|c|}
0 + 0 = 0, c=0 & 0 & 0 & 0 & 0 \\
0 + 1 = 1, c=0 & 0 & 1 & 1 & 0 \\
1 + 0 = 1, c=0 & 1 & 0 & 0 & 0 \\
1 + 1 = 0, c=1 & 1 & 1 & 0 & 0 \\
\end{array}
\]

But Carry One to next column

\[ 12_{10} + 6_{10} = 18_{10} \]
**4-bit binary adder using Full adder**

\[
\begin{array}{c}
\text{A3 A2 A1 A0} \\
+ \text{B3 B2 B1 B0} \\
\hline
\text{S4 S3 S2 S1 S0}
\end{array}
\]

\[
\begin{array}{c}
110 \\
+ 011 \\
\hline
1010
\end{array}
\]

- Full Adder
- Sum
- Carry
- But Carry One to next column

**4-bit Binary Adder**

\[
\begin{array}{c}
\text{C3} \\
\text{C2} \\
\text{C1} \\
\text{C0}
\end{array}
\]

\[
\begin{array}{c}
\text{Full Adder} \\
\text{Adder} \\
\text{Adder} \\
\text{Adder}
\end{array}
\]

**Binary Subtraction**

- Subtraction Table
- \( 12_{10} = 1100 \)
- \( -6_{10} = 0110 \)
- \( 0110 = 6_{10} \)

- But borrow One from next column

**Sign and Magnitude system**

- \( 100110_2 = -6_{10} \)

- Sign bit 1 = -

- Magnitude (same in this case)

- \( 00110_2 = 6_{10} \)

**One’s complement**

- \( 000110_2 = 6_{10} \)
- \( 111010_2 = -6_{10} \)

- Formal conversion \( \Rightarrow N' = (2^n - 1) - N \)

- Example
  \[
  \begin{array}{c}
  (2^n-1) \\
  N = 6
  \end{array}
  \]

- \( 1111 \)
- \( -00110 \)
- \( 11001 \)

- Simple conversion \( \Rightarrow \) flip all bits

**Two’s complement**

- \( 000110_2 = 6_{10} \)
- \( 111010_2 = -6_{10} \)

- Formal conversion \( \Rightarrow N' = N' + 1 \)

- Example
  \[
  \begin{array}{c}
  (2^n-1) \\
  N = 6
  \end{array}
  \]

- \( 1111 \)
- \( -00110 \)
- \( 11001 \)
- \( \text{add 1} \)
- \( +00001 \)
- \( 11010 \)
**Binary Subtraction**

\[
12_{10} = 1100 - 6_{10} = -0110 +1010 = 0110 = 6_{10}
\]

2's Complement

\[
\begin{align*}
N &= 6 & \text{0110} \\
N' &= -0110 & \text{1001} \\
\text{add 1} &= +0001 & \text{1010}
\end{align*}
\]

- Ignore the last carry

**4-bit Binary Subtractor**

\[
\begin{align*}
\text{C3} &\quad \text{Full Adder} & \quad \text{B3} \quad \text{A3} \\
\text{C2} &\quad \text{Full Adder} & \quad \text{B2} \quad \text{A2} \\
\text{C1} &\quad \text{Full Adder} & \quad \text{B1} \quad \text{A1} \\
\text{C0} &\quad \text{Full Adder} & \quad \text{B0} \quad \text{A0}
\end{align*}
\]

- Ignore the last carry out

**Adder/Subtractor**

- Combine adder and subtractor with one control input
- Add/subtract = 1 – Adds B with A
- Add/subtract = 0 – Subtracts B from A

**Hints:**

- Use EX-OR Operation

<table>
<thead>
<tr>
<th>Add/Sub (S)</th>
<th>B</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</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>

**Lab Instructions:**

- Create new project for lab 1b
- When Adding Symbol -> use the symbol name (first lab)