Opening Remarks

• Announcements
  - HW1 due next Mon
  - Labs begin in week 4
  - No class next-next Mon -MLK Day
  - ECE230 Review HO_0 posted on web
  - Anyone need a syllabus?

• Today’s Objectives
  - Perform number base conversions for Dec, Hex, Bin
  - Identify value range as function of number of bits; identify out of range overflow for signed and unsigned binary numbers
  - Express numbers in signed 2’s complement (S2C) form, perform 2’s complement operation, and evaluate subtraction using S2C.
  - Identify value range in S2C and determine 2C overflow
  - Perform minimization of logic expressions using min/max terms, K-maps, and Boolean arithmetic
Homework Guidelines

• Be neat!
  - should represent a professional work product
• Show work
• Clearly indicate answers
• Give units in answer
• Grading
  - effort more than results
  - may not be “corrected” but solutions will be posted
• Homework Questions
  - come to office hours!

Number Systems

• Digital Bases
  - Decimal (Dec), base 10
  - Binary (Bin), base 2
  - Hexadecimal (Hex), base 16
• Base Conversions
  - Bin → Dec

  EX: \( 1010_2 \rightarrow 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \)
  = 8 + 0 + 2 + 0 = 10_{10}

  - Hex → Dec

  EX: \( 2EB5_{16} \rightarrow ?_{10} \)
  show work! = \( 2 \times 16^3 + E \times 16^2 + B \times 16^1 + 5 \)
  = 4096 + 256 + 176 + 5 = 4533_{10}

  - Bin → Hex

  EX: \( 101101001101_2 \rightarrow ?_{16} \)
  use calculator = \( 2 \times (4096) + 14 \times 256 + 11 \times 16 + 5 \)
  = 11857_{10}

  group by 4 from LSB

  1011 0100 1101 = B4D_{16}

TPS: Convert 111101101 to hex.
**Binary Addition**

**DEMO:** using PC Calculator to check conversions & math
- only for *checking* answers!
- always show work in homework

**Addition Examples**

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>+ 4</td>
<td>0100</td>
<td>+ 4</td>
</tr>
<tr>
<td>1110</td>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>10112</td>
<td>01112</td>
<td>7</td>
</tr>
<tr>
<td>01002</td>
<td>01002</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>+ 9</td>
<td>1001</td>
<td>+ 9</td>
</tr>
<tr>
<td>1016</td>
<td>01116</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>10016</td>
<td>9</td>
</tr>
<tr>
<td>1016</td>
<td>01116</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>10016</td>
<td>9</td>
</tr>
<tr>
<td>1016</td>
<td>01116</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>10016</td>
<td>9</td>
</tr>
</tbody>
</table>

**Value Range**
- $n$ unsigned binary bits can only express values 0 to $2^n - 1$
- $n$ unsigned hex bit express values 0 to $16^n - 1$
- larger values generate “overflow” → carry_out bit

---

**Binary Subtraction**

- Microprocessors do not subtract, they only add!
- 2’s Complement = 2C
  - $X - Y = X + (-Y) = X + [Y]^*, \text{ where } [ ]^* \text{ means } 2\text{’s complement}$
  - read extensive notes in HO_0a
- To subtract...
  - use numbers in signed 2’s complement (S2C) form
  - use 2C operation to negate subtracted/negative values
  - then ADD

**EX:** Evaluate 5 - 3 using 4-bit S2C #’s

\[
\begin{align*}
5 & \rightarrow -0101 \\
3 & \rightarrow -0011 \\
\text{S2C: } & \rightarrow 1100 + 1
\end{align*}
\]

value = 2
sign bit 0 = + (positive)
carry_out, ignore for subtraction
2's Complement Overflow

EX: Evaluate -7 - 6 using 4-bit S2C #'s

\[
\begin{align*}
-7 & \rightarrow 0111 \\
-6 & \rightarrow 0110 \\
\text{value} = 3 & \rightarrow 1001 \\
\text{sign bit 0} = + \text{ (positive)} & \\
\text{carry_out, ignore for subtraction} & \\
\end{align*}
\]

but does -7 - 6 = 3? No! should = -13

-13 is out of range for 4-bit S2C #

- **2's Complement Overflow**: when result of arithmetic is outside the value range of n-bit signed binary number
- Signed binary range: \(-2^{n-1}\) to \(2^{n-1}-1\)
- Detecting 2C Overflow: only overflow if sign of both numbers is same & is opposite sign of addition result

\[
\text{Overflow} = \overline{A_{n-1}} \cdot \overline{B_{n-1}} \cdot S_{n-1} + A_{n-1} \cdot \overline{B_{n-1}} \cdot \overline{S_{n-1}}
\]

Arithmetic with 2C Overflow

EX: Evaluate using 3-bit S2C #'s, detect 2C overflow

\[
\begin{align*}
\frac{011}{\text{3}} \rightarrow \frac{\overline{101}}{} & \rightarrow 011 + 1 = \frac{101}{\text{2}} \\
\text{negative} & \\
\end{align*}
\]

- **2C Overflow?**
  - sign of both #'s (at addition) = 0
  - sign of result = 1 (opposite of 0)
  - 2C overflow!

- More examples in HO_0a
Bin/Hex Math Examples

EX: Evaluate using 4-bit S2C #’s, detect 2C overflow

0110 → 0110 → 0110 → +6
-1110 → 0[0110] → 0010 → +2
1000 \[\begin{array}{c}
\text{no match}
\end{array}\]

• meets conditions for 2C overflow, so answer is not reliable

EX: Evaluate using 4-bit S2C #’s, check answer

1110 0101
-EE → 1110 1110 → +[110 111] = 001 0001 + 1 = 001 0010

\[\begin{array}{c}
\text{EX: Find the minimal SoP expression for } F = \sum_{x,y,z} (1,2,5,7)\\
\end{array}\]

Logic Minimization

• Often need to reduce a complex logic expression to its smallest form (i.e., fewest number of logic operations)

• Methods of logic minimization
  - Boolean arithmetic using Boolean properties
    • EX \[ F = \overline{X} \cdot \overline{Y} \cdot Z + \overline{X} \cdot \overline{Y} \cdot \overline{Z} = \overline{X} \cdot \overline{Y} \cdot (Z + \overline{Z}) = \overline{X} \cdot \overline{Y} \]
  - Karnaugh maps
    • EX: Find the minimal SoP expression for \[ F = \sum_{x,y,z} (1,2,5,7) \]

\[
\begin{array}{c|c|c|c|c}
xy & 00 & 01 & 11 & 10 \\
\hline
z & 0 & m_1 & m_2 & m_3 \\
0 & m_4 & m_5 & m_6 & m_7 \\
\end{array}
\]

• then reduce using Boolean arithmetic

\[ F = XZ + Y'Z + X'YZ' = Z(X+Y') + X'YZ' \] is the minimal form.
Basic Combinational Logic Gates

• **INV**

\[ A \rightarrow Y = A' \]

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

• **AND**

\[ A \land B \]

• **NAND**

\[ \overline{A \land B} \]

• **OR**

\[ A \lor B \]

• **NOR**

\[ \overline{A \lor B} \]

• inversion “bubbles”

<table>
<thead>
<tr>
<th>inputs</th>
<th>AND</th>
<th>NAND</th>
<th>OR</th>
<th>NOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>A*B</td>
<td>A*B</td>
<td>A+B</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>1</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>
</tbody>
</table>

More Combinational Logic

• **MUX**: select 1 from many
  - example 4:1 MUX

• **Buffer**: signal isolation & drive
  - inverter without the inversion

\[ A \rightarrow Y = A \quad A \rightarrow Y = A \]

• **Tri-State** (buffer/inverter)
  - buffer (or inverter) with disable (high impedance) state

<table>
<thead>
<tr>
<th>en</th>
<th>A</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>HI</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>HI</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>
And More…

• XOR & XNOR
  - true of both different (XOR) or both same (XNOR)

\[
\begin{array}{c|c|c}
A & B & A \oplus B \\
0 & 0 & 1 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

• Decoder
  - only one “active” output
  - example 4:1 decoder
  - difference from MUX?
  - active low concept

Sequential Logic & Latch

• Sequential logic
  - difference from combinational?
  - output based on input value at prior time

• Latch
  - 2-state storage circuit
    • can “hold” data
  - bi-stable circuit
  - D-type latch
  - D-type latch w/ enable
  - SR latch
Flip Flop

• Flip flop is a synchronous circuit
  – which means?
• D-type master-slave flip flop
  – latch level
  – logic gate level
• JK flip flop
• Toggle flip flop
• Convertability

State tables for D-type, JK, and T-type flip flops

Flip Flop Timing Diagrams

• Timing diagram: map of digital output (& input) vs. time
  – use state table to find output
  – look at proper clock transition
  – ignore input changes outside clock transitions
• Rising- vs. falling-edge triggered
  – triggered by clock
  – rising = positive = “up”
  – falling = negative = “down”
Registers

• Data registers
  - stores block (byte/word) of digital data
  - composed of flip flops; used as static memory
  - example: 4b parallel-in parallel-out register

• Shift register
  - can move data laterally between register bits
  - can input or output (or both) data serially
  - example: 4b serial-in parallel out shift register

Shift & Rotate

• Shift
  - move each bit (left or right) to adjacent register,
    load in preset value (normally 0) into open registers

• Rotate
  - move each bit (left or right) to adjacent register,
    rotate exiting bits back into other side of register

• Example:
  - shift right by 2
  - rotate left by 1
DeMorgans & Complete Set

- **DeMorgans Relations**
  \[
  \overline{A \cdot B} = \overline{A} + \overline{B} \\
  \overline{A + B} = \overline{A} \cdot \overline{B}
  \]

- **Complete Set Concept**
  - all logic functions can be implemented with **ONLY NAND** (or **ONLY NOR**)
  - example: INV with NAND
    
    \[
    A \implies \overline{A} = A \implies (A \cdot \overline{A}) = \overline{A}
    \]
  - similarly, can make AND, OR, NOR with only NAND

---

Logic Schematic Manipulation

- **Bubble pushing technique**
  - bubbles can be moved from input of a gate to the output of an attached gate (or from output to input)
  - 2 bubbles can be added to any node (like double negative)

- **Example**: convert to all NANDs using DeMorgans & bubble pushing
  - initial F
  
  \[
  \begin{align*}
  X & \implies \overline{X} \\
  X \cdot Y & \implies \overline{X} \cdot \overline{Y} \\
  X \oplus Y \oplus Z & \implies \overline{X} \cdot \overline{Y} \cdot \overline{Z} + \overline{X} \cdot \overline{Y} \cdot \overline{Z} \cdot \overline{X} \cdot \overline{Y} \cdot \overline{Z} \\
  \end{align*}
  \]
  - bubble pushing
  - DeMorgans
  - final