Learning Outcomes

- I understand what gates are used to design half and full adders
- I can build larger arithmetic circuits from smaller building blocks

Adder Intro

- Addition is one of the most common operations performed by computer systems
- We can use adders to build larger components like the [Diagram: Adder (+) and Register]
- Every clock cycle, the value Q (let's say 4-bits: Q[3:0]), _________ to the adder circuit which adds 1 to the value and the register captures that new value on the next clock edge
- The sequence on Q on each clock cycle would be: 0, _________...
- Could you design what's inside the adder block? How would you do it?

```
0111
+ 1
1000 = next Q
```
Adder Intro

• What if we had to add ______ two 4-bit numbers, X[3:0] and Y[3:0]? Do we have the techniques to build such a circuit directly?
  – ________________
  – ________________
  0110 = X
  + 0111 = Y
  1101

Addition – Half Adders

• Addition is done in columns
  – Inputs are the bit of X, Y
  – Outputs are the Sum Bit and Carry-Out (C_{out})
• Design a Half-Adder (HA) circuit that takes in X and Y and outputs S and C_{out}
• Use the truth table to find the gate implementation

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>C_{out}</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Problem With Half Adders

• We’d like to use one adder circuit for each column of addition
• Problem:
  – No place for _________ of half adder to connect to the _________
• Solution
  – Redesign adder circuit to include an _________ input for the _________
Addition – Full Adders

- Add a Carry-In input ($C_{in}$)
- New circuit is called a Full Adder (FA)

$$\begin{array}{cc}
0 & 0 \\
0 & 1 \\
0 & 1 \\
1 & 0 \\
1 & 1 \\
\end{array}$$

$$\begin{array}{cc}
0 & 0 \\
0 & 1 \\
0 & 1 \\
1 & 0 \\
1 & 1 \\
\end{array}$$

Addition – Full Adders

- Find the minimal 2-level implementations for $C_{out}$ and $S$...

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>$C_{in}$</th>
<th>$C_{out}$</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>1</td>
<td>0</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>1</td>
<td>0</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>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Recall a 2-input XOR can be written in SOP as $F = x'y + xy'$

XOR and XNOR Gates

- Recall a 2-input XOR can be written in SOP as $F = x'y + xy'$
- A 2-input XNOR can be written in SOP as $F = x'y' + xy$

<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>1</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>1</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>

$F = X \text{ xor } Y \text{ xor } Z$

<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>1</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>1</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>

$F = X \text{ xor } Y \text{ xor } Z$

A checkerboard K-map corresponds to either an XOR or XNOR

Full Adder Logic

- $S =$
  - Recall: XOR is defined as true when ODD number of inputs are true...exactly when the sum bit should be 1
- $C_{out} =$
  - Carry when sum is 2 or more (i.e. when at least 2 inputs are 1)
  - Circuit is just checking all combinations of 2 inputs
Addition – Full Adders (1)

- Use 1 Full Adder for each column of addition

\[0110\]
\[+\]
\[0111\]

Addition – Full Adders (2)

- Connect bits of top number to X inputs

\[0110\]
\[+\]
\[0111\]

Addition – Full Adders (3)

- Connect bits of bottom number to Y inputs

\[0110 = X\]
\[+\]
\[0111 = Y\]

Addition – Full Adders (4)

- Be sure to connect first \(C_{in}\) to 0

\[0110 = X\]
\[+\]
\[0111 = Y\]
Addition – Full Adders (5)

- Use 1 Full Adder for each column of addition

\[ 0110 = X \]
\[ + 0111 = Y \]
\[ = 1 \]

Addition – Full Adders (6)

- Use 1 Full Adder for each column of addition

\[ 0110 = X \]
\[ + 0111 = Y \]
\[ = 01 \]

Addition – Full Adders (7)

- Use 1 Full Adder for each column of addition

\[ 1100 = X \]
\[ 0110 = X \]
\[ + 0111 = Y \]
\[ = 101 \]

Addition – Full Adders (8)

- Use 1 Full Adder for each column of addition

\[ 0110 = X \]
\[ + 0111 = Y \]
\[ = 1101 \]
Performing Subtraction

• To subtract

\[ \begin{array}{c}
0101 = X \\
- 0011 = Y \\
\hline
0010 = X + Y
\end{array} \]

4-bit Adders

• We can create a component to perform 4-bit addition

\[ \begin{array}{c}
A_3A_2A_1A_0 = A \\
+ B_3B_2B_1B_0 = B \\
\hline
S_4S_3S_2S_1S_0 = S
\end{array} \]

Device vs. System Labels

• When using hierarchy (i.e. building blocks) to design a circuit be sure to show both device and system labels
  
  - Device Labels: Signal names used ________ the block
    
    - _________ to indicate which input/output is which to the outside user
  
  - System labels: Signal names used ________ the block
    
    - _________ signals from the circuit being built
    
    - Can have the same name as the device label if such a signal name exists at the outside level

EXERCISES
**Building an 8-bit Adder**

- Use (2) 4-bit adders to build an 8-bit adder to add \( X = X[7:0] \) and \( Y = Y[7:0] \) and produce a sum, \( S = [7:0] \) and a carry-out, \( C_8 \).
  - Label the inputs and outputs and make appropriate connections.

**Adding Many Bits**

- You know that an FA adds \( X + Y + C_i \)
- Use FA and/or HA components to add 4 individual bits:
  \( A + B + C + D \)

**Adding 3 Numbers**

- Add \( X[3:0] + Y[3:0] + Z[3:0] \) to produce \( F[?:0] \) using the adders shown plus any FA and HA components you need.

**Mapping Algorithms to HW**

- Wherever an if..then..else statement is used usually requires a mux
    - \( Z = A + 2 \)
  - else
    - \( Z = B + 5 \)
11.29 Mapping Algorithms to HW

- Wherever an if..then..else statement is used usually requires a mux
    - Z = A+2
  - else
    - Z = B+5

Comparison Circuit

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0101</td>
<td>0010</td>
<td>1</td>
</tr>
</tbody>
</table>

11.30 Adder / Subtractor

- If sub == 1
- Else
  - Z = X[3:0] + Y[3:0]

11.31 Another Example

- Design a circuit that takes a 4-bit binary number, X, and two control signals, A5 and M1 and produces a 4-bit result, Z, such that:
  - Z = X + 5, when A5,M1 = 1,0
  - Z = X - 1, when A5,M1 = 0,1
  - Z = X, when A5,M1 = 0,0

11.32 Adder / Subtractor

- Go back and optimize the muxes by determining what logic function they are actually performing
  - If sub == 1
  - Else
    - Z = X[3:0] + Y[3:0]