Unit 11

Adders & Arithmetic Circuits
Learning Outcomes

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

- Addition is one of the most common operations performed by computer systems.
- We can use adders to build larger components like the counter to the right.
- Every clock cycle, the value $Q$ (let's say 4-bits: $Q[3:0]$), feeds back 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, 1, 2, 3, 4...
- Could you design what's inside the adder block? How would you do it?

$$0111 = \text{curr } Q$$
$$+ \quad 1$$
$$\underline{1000} = \text{next } Q$$
Adder Intro

• What if we had to add ANY two 4-bit numbers, X[3:0] and Y[3:0]? Do we have the techniques to build such a circuit directly?
  
• Yes and no
  
  – No. Not with K-maps since there are 8-inputs
  
  – Yes. We could use sum of minterms but that would take a long time, but it could be done

\[
\begin{align*}
0110 & = X \\
+ 0111 & = Y \\
\hline
1101 & \quad \text{result}
\end{align*}
\]
Adder Intro

- **Idea**: Build a circuit that performs one column of addition and then use 4 instances of those circuits to perform the overall 4-bit addition.

- Let's start by designing a circuit that adds 2-bits: X and Y that are in the same column of addition.

  
  
  \[
  \begin{array}{c c}
  0110 & = X \\
  + & 0111 & = Y \\
  \hline
  1101 & = \text{Sum} \\
  \end{array}
  \]

  

  ![Half Adder Diagram]
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 Carry-out of half adder to connect to the next

• Solution
  – Redesign adder circuit to include an additional input for the carry
Addition – Full Adders

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

<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>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>
Addition – Full Adders

- Find the minimal 2-level implementations for Cout and S...

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Cin</th>
<th>Cout</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>1</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>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>0</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>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>
Full Adder Logic

- **S = X xor Y xor Cin**
  - Recall: XOR is defined as true when an ODD number of inputs are true...exactly when the sum bit should be 1

- **Cout = XY + XCin + YCin**
  - 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

\[ \begin{array}{c}
0110 \\
+ \quad 0111 \\
\hline
0111
\end{array} \]
Addition – Full Adders (2)

- Connect bits of top number to X inputs

\[ \begin{array}{c}
0110 \\
+ \ 0111 \\
\hline
1011
\end{array} \]
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

\[
\begin{array}{c}
0110 \\
+ 0111 \\
= X
\end{array}
\begin{array}{c}
0111 \\
+ 0111 \\
= Y
\end{array}
\]

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
X & Y & \text{C}_{\text{out}} & \text{C}_{\text{in}} \\
\text{Full Adder} & \text{Full Adder} & S
\end{array}
\begin{array}{cccc}
1 & 1 & 0 & 1 \\
X & Y & \text{C}_{\text{out}} & \text{C}_{\text{in}} \\
\text{Full Adder} & \text{Full Adder} & S
\end{array}
\begin{array}{cccc}
1 & 1 & 0 & 0 \\
X & Y & \text{C}_{\text{out}} & \text{C}_{\text{in}} \\
\text{Full Adder} & \text{Full Adder} & S
\end{array}
\begin{array}{cccc}
0 & 1 & 0 & 1 \\
X & Y & \text{C}_{\text{out}} & \text{C}_{\text{in}} \\
\text{Full Adder} & \text{Full Adder} & S
\end{array}
\]
Addition – Full Adders (6)

• Use 1 Full Adder for each column of addition

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

\[
\begin{array}{cccc}
X & Y & C_{in} & S \\
0 & 0 &  &  \\
1 & 1 &  &  \\
1 & 1 &  &  \\
0 & 1 &  &  \\
\end{array}
\]
Addition – Full Adders (7)

• Use 1 Full Adder for each column of addition

\[
\begin{array}{c}
1100 \\
0110 = X \\
+ \quad 0111 = Y \\
\hline
101
\end{array}
\]

Diagram:

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

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

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

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

```
\[ 0110 \]
\[ + 0111 \]
\[ = 101 \]
```
Addition – Full Adders (8)

- Use 1 Full Adder for each column of addition

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

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

\[
\begin{array}{c|c|c}
0 & 1 & 1 \\
1 & 1 & 0 \\
\end{array} = 1101
\]
Performing Subtraction

- To subtract
  - Flip bits of Y
  - Add 1

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

\[
\begin{array}{c}
0101 \\
+ 1100 \\
\hline
10010
\end{array}
\]
4-bit Adders

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

\[
\begin{align*}
A_3A_2A_1A_0 &= A \\
+ B_3B_2B_1B_0 &= B \\
\underline{S_4S_3S_2S_1S_0} &= S
\end{align*}
\]
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 inside the block
    - Placeholders to indicate which input/output is which to the outside user
  - **System labels**: Signal names used outside the block
    - Actual signals from the circuit being built
    - Can have the same name as the device label if such a signal name exists out the outside level

**Analogy**: Formal and Actual parameters
1. `a` and `b` are like device labels and indicate the names used inside a block.
2. `x` and `y` are like system labels and represent the actual values to be used.

```c
int div(int a, int b)
{
    int s = a/b;
    return s;
}
int main()
{
    int x=10, y=2;
    int s = div(x,y);
}
```

**Device Labels**: Indicate which input/output is which inside the block.
**System Labels**: Actual signals from the circuit being built.
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 $X + Y + Ci$

• Use FA and/or HA components to add 4 individual bits:
  - $A + B + C + D$

• Solution:
  - 4 bits could yield sums from $000 - 100_2$. So we need 3 bits of output ($S_2, S_1, S_0$)
  - Be sure that bits you connect to a HA or FA are all from the same column (weight)
Adding 3 Numbers

- Solution: Adding (3) 4-bit numbers yields a sum of at most 45 = 15 + 15 + 15 which requires 6 bits of output (F[5:0]).
  - Be sure the bits you connect to the same adder column have the same significance/weight.
Mapping Algorithms to HW

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

• Wherever an if..then..else statement is used usually requires a mux
    • Z = A+2
  – else
    • Z = B+5
Adder / Subtractor

- If sub == 1
- Else
  - \( Z = X[3:0] + Y[3:0] \)
Adder / Subtractor

• Go back and optimize the muxes by determining what logic function they are actually performing
  • If sub == 1
  • Else

<table>
<thead>
<tr>
<th>SUB</th>
<th>Yi</th>
<th>Bi</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
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 \( A5, M1 = 0,0 \)

<table>
<thead>
<tr>
<th>A5</th>
<th>M1</th>
<th>B3</th>
<th>B2</th>
<th>B1</th>
<th>B0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
</tr>
</tbody>
</table>