Introduction to Digital Logic

Lecture 15:
Comparators
Adding Many Bits

- You know that an FA adds $X + Y + Ci$
- Use FA and/or HA components to add 4 individual bits: $W + X + Y + Z$
Adding 3 Numbers

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/\sim\text{add} = 1
- Else
  - \( Z = X[3:0] + Y[3:0] \)
Adder / Subtractor

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

<table>
<thead>
<tr>
<th>SUB/~ADD</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>

4-bit Binary Adder

SUB/~ADD

X0 A0
X1 A1
X2 A2
X3 A3

Y0 B0
Y1 B1
Y2 B2
Y3 B3

Z0 C0
Z1 C1
Z2 C2
Z3 C3

© Mark Redekopp, All rights reserved
Adder / Subtractor

\[ Z = \begin{cases} 
X + 5, & \text{when } A5,M1 = 1,0 \\
X - 1, & \text{when } A5,M1 = 0,1 \\
X, & \text{when } A5,M1 = 0,0 
\end{cases} \]
Multiplication By Powers of 2

• In decimal, multiplication by $10^n$ can easily be achieved by adding $n$ 0’s to the right of the number
  – $937 \times 100 = 937 \times 10^2 = 93,700$
  – $145 \times 10000 = 145 \times 10^4 = 1,450,000$

• In binary, multiplication by $2^n$ can easily be achieved by adding $n$ 0’s to the right of the number
  – $3_{10} = 11_2$
  – $6_{10} = 3 \times 2^1 = 110_2$
  – $12_{10} = 3 \times 2^2 = 1100_2$
  – $24_{10} = 3 \times 2^3 = 11000_2$
Division By Powers of 2

• In decimal, division by $10^n$ can easily be achieved by shifting the decimal point $n$ places to the left
  – $937 / 100 = 937 / 10^2 = 9.37$
  – $145,768 * 10000 = 145,768 / 10^4 = 14.5768$

• In binary, division by $2^n$ can easily be achieved by shifting the decimal point $n$ places to the left
  – $12_{10} = 1100 \Rightarrow 12/2^1 = 110.0_2 = 6$
  – $12_{10} = 1100 \Rightarrow 12/2^2 = 11.00_2 = 3$
  – $7_{10} = 111 \Rightarrow 7/2^1 = 11.1_2 = 3.5$
Constant Multiplication w/ Adders

- Design a circuit to calculate $Y = 10X$, where $X$ is a 4-bit unsigned number ($10_{10}$ is a constant)
  
  - Break it down to powers of 2 (e.g. $10X = 8X + 2X$)
  
  - Recall, multiplying by a power of 2 is equal to shifting $X$ to the left that number of bits (i.e. $8X$ is just $X$ shifted to the left 3 places)
  
  - if $X = 0011_2 = 3_{10}$
    
    - $8X = 0011000_2 = 24_{10}$
    - $2X = 0000110_2 = 6_{10}$

\[
\begin{align*}
0011000 &= 8X \\
+ 0000110 &= 2X \\
\hline
0011110 &= 10X
\end{align*}
\]
Constant Multiplication w/ Adders

• Design a circuit to calculate $Y = 10X$, where $X$ is a 4-bit unsigned number ($10_{10}$ is a constant)
  – Break it down to powers of 2 (e.g. $10X = 8X + 2X$)
  – Recall, multiplying by a power of 2 is equal to shifting $X$ to the left that number of bits (i.e. $8X$ is just $X$ shifted to the left 3 places)
  – if $X = 0011_2 = 3_{10}$
    • $8X = 0011000_2 = 24_{10}$
    • $2X = 0000110_2 = 6_{10}$

\[
\begin{align*}
0011000 &= 8X \\
+ 0000110 &= 2X \\
\hline
0011110 &= 10X
\end{align*}
\]
Performing Multiplication w/ Adders

• How many 4-bit adders do we need?

\[
\begin{align*}
X_3 & \quad X_2 & \quad X_1 & \quad X_0 & \quad 0 & \quad 0 & \quad 0 & = & \quad 8X \\
+ & \quad X_3 & \quad X_2 & \quad X_1 & \quad X_0 & \quad 0 & & = & \quad 2X \\
\hline
Y_7 & \quad Y_6 & \quad Y_5 & \quad Y_4 & \quad Y_3 & \quad Y_2 & \quad Y_1 & \quad Y_0 & = & \quad 10X
\end{align*}
\]
Performing Multiplication w/ Adders

- To implement using 4-bit adders,
  - Assume 4-bit number \( X = X_3X_2X_1X_0 \)

\[
\begin{array}{cccc}
X_3 & X_2 & X_1 & X_0 \\
+ & X_3 & X_2 & \\
Y_7 & Y_6 & Y_5 & Y_4 & Y_3
\end{array}
\]

\[
\begin{align*}
0 & 0 & 0 & 0 = 8X \\
X_1 & X_0 & 0 = 2X \\
Y_2 & Y_1 & Y_0 = 10X
\end{align*}
\]

We don’t need an adder to figure out what the sum of these 3 columns are because we’re adding with 0.
Y = 10*X Circuit
COMPARATORS
Comparators

• Compare two numbers and produce relational conditions $A < B$, $A = B$, $A > B$, $A \leq B$, and $A \geq B$

• Suppose we want to compare two, 4-bit numbers: $A[3:0]$ and $B[3:0]$
  – Too big to build directly…decompose the problem

Compare:

  A3 A2 A1 A0
  B3 B2 B1 B0
Start Easy...

- Build a circuit to compare a bit, A, with a bit, B
- The only way one bit, A, can be greater than another bit, B, is if A=1 and B=0
- The only way one bit, A, can be less than another bit, B, is if A=0 and B=1
- A and B are equal when they are the same

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>A=B</th>
<th>A&lt;B</th>
<th>A&gt;B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</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>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

![Logic Gates Diagram](attachment:logic_gates.png)
4-bit Comparator

A3 B3
Comp.
GT EQ LT

A2 B2
Comp.
GT EQ LT

A1 B1
Comp.
GT EQ LT

A0 B0
Comp.

Merger

A>B
A=B
A<B
Merging Comparison Results

• Suppose we have the comparison results of individual digits of a 2-digit number, how can we combine them to produce the overall result?
  – Key: More significant digits/bits fully determine results of overall comparison unless they are equal…in which case we use the result of the less significant digits

\[
\begin{align*}
A &= (9 \ 3)_{10} \\
B &= (7 \ 8)_{10}
\end{align*}
\]

\[
\begin{align*}
\text{MSD:} & \quad 9 > 7 \\
\text{LSD:} & \quad 3 < 8
\end{align*}
\]

Overall?

\[A > B \quad (9x > 7x)\]
Greater-Than, Less-Than, Equal

- Find an algorithm to check whether one decimal number is greater than, less than, or equal to another
  - $837 > 756$
    - Start with the MSD, $8 > 7$ and you’re done…any number with 8 in the 100’s place is greater than a number with 7 in the 100’s place
  - $621 < 649$
    - Start with the MSD, $6 = 6$…you can’t tell yet which is greater or less than so move to the next digit
    - $2 < 4$ and you’re done $62x$ is always less than $64x$

- Algorithm: Start with the MSD and keep comparing digits to the right. Once any digit of A is $>$ or $<$ corresponding digit of B then you know the whole number A $>$ B or A $<$ B, respectively. If all digits are equal then both number are equal.
## Merging Results

<table>
<thead>
<tr>
<th>MS Digit Results</th>
<th>LS Digit Results</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A&gt;B (MS)</td>
<td>A&gt;B (LS)</td>
<td>A &gt; B</td>
</tr>
<tr>
<td>A&lt;B (MS)</td>
<td>A&lt;B (LS)</td>
<td>A &lt; B</td>
</tr>
<tr>
<td>A=B (MS)</td>
<td>A=B (LS)</td>
<td>A = B</td>
</tr>
</tbody>
</table>

\[
A = (9 \ 3)_{10} \quad \text{MSD:} \quad 9 > 7 \\
B = (7 \ 8)_{10} \quad \text{LSD:} \quad 3 < 8 \\
Overall? \]
# Merging Results

<table>
<thead>
<tr>
<th>MS Digit Results</th>
<th>LS Digit Results</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A&gt;B (MS)</td>
<td>A&gt;B (LS)</td>
<td>A&gt;B</td>
</tr>
<tr>
<td>A&lt;B (MS)</td>
<td>A&lt;B (LS)</td>
<td>A&lt;B</td>
</tr>
<tr>
<td>A=B (MS)</td>
<td>A=B (LS)</td>
<td>A=B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1 0 0</th>
<th>X X X</th>
<th>1 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 0</td>
<td>X X X</td>
<td>0 1 0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>1 0 0</td>
<td>1 0 0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0 1 0</td>
<td>0 1 0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0 0 1</td>
<td>0 0 1</td>
</tr>
</tbody>
</table>

A= (9 3) <sub>10</sub>  
B= (7 8) <sub>10</sub>  

MSD: 9 > 7  
LSD: 3 < 8  
Overall?
# Merging Logic

<table>
<thead>
<tr>
<th>MS Digit Results</th>
<th>LS Digit Results</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>MGT</td>
<td>MLT</td>
<td>MEQ</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\text{GT} = \text{MGT} + \text{MEQ} \cdot \text{LGT} \\
\text{LT} = \text{MLT} + \text{MEQ} \cdot \text{LLT} \\
\text{EQ} = \text{MEQ} \cdot \text{LEQ}
\]
4-bit Comparator

Delay: Linear in the length of the chain

Can we try to implement the merging in 2-levels?
4-bit Equality Check

- XNOR gate outputs 1 when two bits are equal

- To check whether two 4-bit numbers are equal
  - use 4 XNOR gates to XNOR each pair of bits
  - only if all the pairs are equal, are the 2 numbers equal (i.e. AND the outputs together)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>XNOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</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>
4-bit Greater-Than

Consider 2 numbers $A (A_3A_2A_1A_0)$ and $B (B_3B_2B_1B_0)$

There are 4 cases for when $A > B$

- $A_3 > B_3$ and who cares about the other bits
- $A_3 = B_3$, $A_2 > B_2$ and the other bits don’t matter
- $A_3 = B_3$, $A_2 = B_2$, $A_1 > B_1$ and the other bits don’t matter
- $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$, $A_0 > B_0$

<table>
<thead>
<tr>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
<th>Case 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1xxx = A</td>
<td>01xx = A</td>
<td>111x = A</td>
<td>0001 = A</td>
</tr>
<tr>
<td>0xxx = B</td>
<td>00xx = B</td>
<td>110x = B</td>
<td>0000 = B</td>
</tr>
</tbody>
</table>
### 4-bit Less-Than

- The same style algorithm can be used to check for less-than.

- There are 4 cases for when $A < B$
  - $A_3 < B_3$ and who cares about the other bits
  - $A_3 = B_3$, $A_2 < B_2$ and the other bits don’t matter
  - $A_3 = B_3$, $A_2 = B_2$, $A_1 < B_1$ and the other bits don’t matter
  - $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$, $A_0 < B_0$

<table>
<thead>
<tr>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
<th>Case 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xxx = A</td>
<td>00xx = A</td>
<td>110x = A</td>
<td>00000 = A</td>
</tr>
<tr>
<td>1xxx = B</td>
<td>01xx = B</td>
<td>111x = B</td>
<td>00001 = B</td>
</tr>
</tbody>
</table>
4-bit Comparator

A3  B3
-<
X  Y
Comp.
GT  EQ  LT
GT3  LT3  EQ3

A2  B2
-<
X  Y
Comp.
GT  EQ  LT

A1  B1
-<
X  Y
Comp.
GT  EQ  LT

A0  B0
-<
X  Y
Comp.
GT  EQ  LT

A>B  <-
A=B  <-
A<B  <-
4-bit Cascabable Comparator (74LS85)

4-bit Magnitude Comparator (Non-cascadable)

1-bit comparator

A0

A0

A0

A0

1-bit comparator

A1

A1

A1

A1

1-bit comparator

A2

A2

A2

A2

1-bit comparator

A3

A3

A3

A3

1-bit comparator

B0

A0 = B
A0 > B
A0 < B
LT0

B0 = B
EQ0

B0

A1 = B
A1 > B
A1 < B
LT1

B1 = B
EQ1

B1

A2 = B
A2 > B
A2 < B
LT2

B2 = B
EQ2

B2

A3 = B
A3 > B
A3 < B
LT3

B3 = B
EQ3

B3

A < B of LSB’s
A > B of LSB’s
A = B of LSB’s
Less-Than

• Rather than using the previous logic, put less-than in terms of equal and greater-than

• \( A < B = (A > B) \cdot (A = B) \)
4-bit Magnitude Comparator

\[ A_0 = B_0 \]
\[ A_1 = B_1 \]
\[ A_2 = B_2 \]
\[ A_3 = B_3 \]
4-bit Magnitude Comparator
4-bit Magnitude Comparator

A0
A1
A2
A3

4-bit Magnitude Comparator

B0
B1
B2
B3

A<B
A>B
A=B
Building Larger Comparators

- Try to build an 8-bit comparator
- Can we chain cascade (chain) together 4-bit magnitude comparators?
  - Try to connect 4-bits of each input to a 4-bit comparator
  - Problem: No way to communicate results of lower bits to higher bit comparator
Building Larger Comparators

• Remember if the MSB’s of A are Greater-Than or Less-Than MSB’s of B we don’t need to look at LSB’s
• Only when MSB’s of A = MSB’s of B do we need to look at the relationship of the LSB’s

<table>
<thead>
<tr>
<th>MSB’s of A &gt; MSB’s of B</th>
<th>MSB’s of A &lt; MSB’s of B</th>
<th>MSB’s of A = MSB’s of B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1100 xxxx = A</td>
<td>0100 xxxx = A</td>
<td>0110 1001 = A</td>
</tr>
<tr>
<td>1010 xxxx = B</td>
<td>0111 xxxx = B</td>
<td>0110 0010 = B</td>
</tr>
</tbody>
</table>

A > B…LSB’s don’t matter
A < B…LSB’s don’t matter
Relationship of LSB’s is now needed to find out whether A>B, A<B, A=B
Building Larger Comparators

• Just add one of our merger circuits
Building Larger Comparators

• Just add one of our merger circuits

To make a block that can be repeated and chained arbitrarily, cut the design as shown:
Building Larger Comparators

- **Solution:**
  - Add cascade inputs $I_{A<B}$, $I_{A>B}$, $I_{A=B}$, to each 4-bit comparator where the comparison of LSB’s can be connected.
  - Cascaded inputs will only be passed when the MSB’s are equal.

![Diagram of comparator circuit](Image)
4-bit Cascadable Comparator

- Below is the logic needed to add these cascaded inputs
4-bit Cascadable Comparator

- Below is the logic needed to add these cascaded inputs

Notice that all 3 of the cascaded inputs are qualified with A=B from the MSB’s. When A=B = 0, the cascaded inputs are forced to 0
4-bit Cascadable Comparator

- Below is the logic needed to add these cascaded inputs

If the MSB’s of A are > or < the MSB’s of B then the overall outputs are fully determined. Thus those outputs go into the final OR gate. If they are 1 then the output will be 1 no matter what the other input to the OR gate is.
Since the MSB’s of A > MSB’s of B we don’t need the results of the LSB comparator.
Cascaded Comparator Example

\[ A = 1011 \ 0010 \]
\[ B = 1011 \ 1100 \]

Since the MSB’s of \( A \) = MSB’s of \( B \) the comparison of the LSB’s determines the final output.
XOR APPLICATIONS
XOR Gate Review

XOR

\[ Z = X \oplus Y \]

<table>
<thead>
<tr>
<th>X</th>
<th>Y</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>0</td>
</tr>
</tbody>
</table>

True if an odd # of inputs are true

2 input case: True if inputs are different
XOR Conditional Inverter

- If one input to an XOR gate is 0, the other input is passed.
- If one input to an XOR gate is 1, the other input is inverted.
- Use one input as a control input which can conditionally pass or invert the other input.

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
XOR / XNOR Equivalents

- XOR gate with...
  - Odd number of input/output inversions => XNOR gate
  - Even number of input/output inversions => XOR gate

<table>
<thead>
<tr>
<th>X</th>
<th>Y</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>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>
Parity & Error Detection

• When digital data is transmitted there is some probability that a bit will be corrupted (flipped)
• We’d like a way to detect an error so that the receiver can ask for a retransmission rather than using the wrong data
• We will attach an extra “parity” bit, P, to the data to help determine if there is an error
Even & Odd Parity

- Even Parity: All transmissions will contain an even number of 1’s
  - We will choose even parity in example below
- Odd Parity: All transmissions will contain an odd number of 1’s

A

1101 1

Make P=1 to ensure even # of 1’s

B

1001 1

Receiver sees odd # of 1’s and knows there is an error
Generating & Checking Parity

- To generate even parity bit P at sender:
  - P = 1 when odd number of 1’s in data
  - This is nothing more than an XOR function
- To check if error at receiver
  - Error = 1 when odd number of 1’s in data and parity bit message
  - Again, nothing more than an XOR function

A

B

© Mark Redekopp, All rights reserved