Lecture 15: Combinational Circuits

- Complete logic functions

- Some combinational logic functions
  - Half adders
  - Adders
  - Multiplexers
  - Decoders
  - Parity generators

- Signal propagation in combinational logic

- ALUs
Complete Logic Functions

• A logic gate is considered to be a complete logic function if we can use just that one type of gate to implement any logic function (i.e., NOT, AND, OR, NAND, and NOR)

• NAND and NOR functions (each by themselves) are complete logic functions

• Why would you be interested in complete logic functions or using just one type of logic gate in your design?
NAND Represents a Complete Logic Function

**NOT**

**AND**

**NAND**
NAND Represents a Complete Logic Function (cont.)

OR

NOR
Example

- $A' + B' + C$
- Circuit design using NOTs and ORs
- “Brute force” substitution of NAND gates
- We can make it a little simpler - how about applying DeMorgan’s Law first?
  \[ A' + B' = (AB)' \]
Another Example

\[ AB + B'C \]
Combinational Logic and Digital Devices

• Now that we’ve learned how to implement a logic expression in a logic-gate-level design, we can start putting those functions together to form more complex (and useful!) functions

• Digital logic that is designed so that the outputs are dependent only upon the current set of inputs is called **COMBINATIONAL LOGIC**
Examples of Combinational Digital Logic Functions

- Multiplexers
- Decoders
- Parity Generators
- Adders
  - Half adders
  - Full adders
  - 32-bit adders
- Shifters
- Comparators
- Arithmetic Logic Units
Half Adder

- Consists of:
  - 2 inputs
  - 2 outputs
    - Sum
    - Carry

- Used for basic integer addition (1 bit)

<table>
<thead>
<tr>
<th>Input A</th>
<th>Input B</th>
<th>Sum</th>
<th>Carry</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>1</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>
</tbody>
</table>
Half Adder Logic Design

A

B

Sum

Carry
Full Adder

- **Consists of:**
  - 3 inputs
    - A
    - B
    - Carry in
  - 2 outputs
    - Sum
    - Carry out

- **Typical application:** link together multiple full adders to add integers containing more than one bit

<table>
<thead>
<tr>
<th>Input A</th>
<th>Input B</th>
<th>Carry In</th>
<th>Sum</th>
<th>Carry Out</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>1</td>
<td>0</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>0</td>
<td>1</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>0</td>
<td>1</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>
Full Adder Logic Design

A

B

Sum

Carry In

Carry Out
Integer Addition Using Full Adder Circuits

Adding two binary numbers, each containing two bits

\[
\begin{align*}
C_1 & \quad C_0 \\
A & \quad (A_1, A_0) \\
+ & \quad (B_1, B_0) \\
\text{Sum} & \quad (C_1, S_1, S_0)
\end{align*}
\]
Longer Chains of Adders
Add Longer Integers

Full Adder

\[ \text{Sum}_3 \]

\[ A_3 \quad B_3 \]

\[ \text{Carry} \]

Full Adder

\[ \text{Sum}_2 \]

\[ A_2 \quad B_2 \]

\[ \text{Carry} \]

Full Adder

\[ \text{Sum}_1 \]

\[ A_1 \quad B_1 \]

\[ \text{Carry} \]

Full Adder

\[ \text{Sum}_0 \]

\[ A_0 \quad B_0 \]

\[ \text{Carry} \]

“0”
Multiplexers

- Consist of:
  - $2^n$ data inputs
  - One output
  - $n$ control inputs
- Used to control the selection of a single output from $n$ different inputs
- Typical application: parallel to serial converter
Decoders

- Consist of:
  - $n$ inputs
  - $2^n$ outputs
- Used to select a single (one) output line
- Typical application: selecting a memory chip

2 to 4 decoder

L15: Combinational Circuits

Fall 2005
Parity Generator

- Consist of:
  - n inputs
  - 1 output

- Used to create a parity bit for a given set of inputs

Even parity for 4 inputs
Arithmetic Logic Units (ALUs)

- The ALU is the math engine for modern digital computers
A Very Simple ALU

• Performs four functions:
  A .AND. B   (A & B)  //bitwise AND
  A .OR. B    (A | B)  //bitwise OR
  NOT.B       (~ B)    //bitwise NOT
  A + B       (A + B)  //addition operator

• Operates on single bits “A” and “B”
Block Diagram for a Very Simple, 1-Bit ALU

Logic Unit (AND, OR, NOT)

2 to 4 Decoder

Full Adder

Carry In

Output

Carry Out

A

B

A

B

F0

F1

Inv A

Enable A

Enable B

En OR

En AND

En NOT

En ADD
1-Bit Slice ALU Example

• Four Functions (AND, OR, NOT B, ADDITION) for single bits \( A_i \) and \( B_i \)

• Inputs
  - INV A  Inverse of \( A_i \)
  - A  The single bit \( A_i \)
  - ENA  Enable the A input
  - B  The single bit \( B_i \)
  - ENB  Enable the B input
  - FO  Control signal 0
  - F1  Control signal 1
  - Carry In  Carry-in signal (usually the carry out signal from the previous state)

• Outputs
  - Output  Result of ALU computation
  - Carry Out  Carry out signal from full adder circuit
Control Signals F0 and F1

- What do the control signals F0 and F1 do?
- Both signals go into the 2 to 4 decoder
- Between the two signals, one of the four ALU functions is selected to be the ALU’s output

<table>
<thead>
<tr>
<th>F0</th>
<th>F1</th>
<th>Output Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>A AND B</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>A OR B</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>NOT B</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>A + B</td>
</tr>
</tbody>
</table>
Putting together digital logic functions
Connect “n” 1-bit ALUs to create
1 n-bit ALU
More Complex ALUs

- ALUs in modern CPUs can include more functions than the four we just put together.

- For example, other functions that might be useful:
  - multiply
  - shift
  - compare
How would you get the ALU to subtract two integers?

• \( A - B \)

• Think about the two's complement procedure
  1. NOT B (take the complement of B)
  2. NOTB + 1 (add one to the result of step 1)
  3. (NOTB + 1) + A (add A to the result of step 2)
  4. Ignore any carry out signals
What additional function would we need to be able to multiply and divide integers?

- Shifters (left shift and right shift)

- Multiplication and division can be thought of as “shift and add” or “shift and subtract” procedures

We don’t have time to build a multiplier, but this should give you the intuition for how it could be done.