Lecture 12

- Time/space trade offs
- Adders

Time vs. speed: Linear chain
- 8-input OR function with 2-input gates
  - Gates: 7
  - Max delay: 7

Time vs. speed: Tree
- Gates: 7
- Max delay: 3

Modified circuit
- Calculate the OR of the first 2 inputs, the OR of the first 3, and so on, up to the OR of all 8
  - Gates: 7
  - Max delay: 7

Parallel version
- Gates: 12
- Max delay: 3

Binary half adder
- 1-bit half adder
  - Computes sum, carry-out
  - No carry-in
  - Sum = A'B + AB' = A xor B
  - Cout = AB
  - A B | S | Cout
  - 0 0 | 0 | 0
  - 0 1 | 1 | 0
  - 1 0 | 1 | 0
  - 1 1 | 0 | 1
**Binary full adder**

- 1-bit full adder
  - Computes sum, carry-out
  - Carry-in allows cascaded adders
  - Sum = Cin xor A xor B
  - Cout = ACin + BCin + AB

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>Cin</th>
<th>Sum</th>
<th>Cout</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>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</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>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Full adder: Using 2 half adders**

- Multilevel logic
  - Slower
  - Fewer gates: 5 vs. 6 (2 XORs, 2 ANDs, 1 OR)

Sum = (A ⊕ B) ⊕ Cin
Cout = ACin + BCin + AB

\[(A ⊕ B)Cin + AB\] Distributive law?

**Full adder: Using 2 half adders**

**4-bit ripple-carry adder**

Also subtracts!
- Twos complement: \(A - B = A + (-B) = A + B' + 1\)

**Problem: Ripple-carry delay**

- Carry propagation limits adder speed

0111 + 0001 = 1000

**Ripple-carry delay**

\(C_n\) arrives late
\(C_{out}\) takes two gate delays
Ripple-carry delay

Can we be more clever?
- Let’s compute all the carries in parallel
  - Derive carries from the data inputs
    - Not from intermediate carries
  - Use two-level logic
  - Compute all sums in parallel
  - How do we do that???

Speeding up the adder
- Need to find a way to “predict” Cout for all bits without knowing what Cin is
- Call this PROPAGATE
- Call this GENERATE

Lookahead logic: Pi and Gi
- Step 1: Getting Pi and Gi
  - Carry generate: Gi = AiBi
  - Generate carry when A = B = 1
  - Carry propagate: Pi = Ai xor Bi
  - Propagate carry-in to carry-out when (A xor B) = 1

Lookahead logic: Sum and carry
- Step 2: Calculate Sum and Cout
  - Si = Ai xor Bi xor Ci = Pi xor Ci
  - Ci+1 = AiBi + Ci(Ai xor Bi) = Gi + CiPi

Lookahead logic: Carries
- Step 3: Express all carries in terms of C0, G, and P
  - Derive intermediate results directly from inputs rather than from carries
  - Allows “sum” computations to proceed in parallel
  
\[
\begin{align*}
C_1 &= G_0 + P_0C_0 \\
C_2 &= G_1 + P_0C_1 = G_1 + P_1G_0 + P_1P_0C_0 \\
C_3 &= G_2 + P_1C_2 = G_2 + P_1G_1 + P_1P_0G_0 + P_1P_1P_0C_0 \\
C_4 &= G_3 + P_0C_4 = G_3 + P_0G_2 + P_0P_1G_1 + P_0P_0P_0G_0 + P_0P_0P_1P_0C_0 \\
\end{align*}
\]
Lookahead logic: Carries

Logic complexity increases with adder size

Lookahead logic: Sum

Summary: Lookahead logic

- Compute all the carries in parallel
  - Derive carries from data inputs not from intermediate carries
  - Compute all sums in parallel using two-level logic
- Cascade simple adders to make large adders
- Speed improvement
- Complex combinational logic