Outline for Today's Lecture

- Digital Building Blocks
- Full Adder
- Ripple Carry Adder
- Carry Look Ahead Adder
- Prefix Adder
- Subtractor
Introduction

• **Digital building blocks:**
  – Gates, multiplexers, decoders, registers, arithmetic circuits, counters, memory arrays, logic arrays

• **Building blocks demonstrate hierarchy, modularity, and regularity:**
  – Hierarchy of simpler components
  – Well-defined interfaces and functions
  – Regular structure easily extends to different sizes

• **Will use building and hierarchy to build microarchitecture in lab**
1-Bit Adders

Half Adder

\[
\begin{array}{cccc}
A & B & C_{\text{out}} & S \\
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 \\
\end{array}
\]

Full Adder

\[
\begin{array}{cccc}
C_{\text{in}} & A & B & C_{\text{out}} \\
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 \\
\end{array}
\]

S = 
C_{\text{out}} =
1-Bit Adders

Half Adder

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C\text{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>

S = C\text{out} + A B

Full Adder

<table>
<thead>
<tr>
<th>C\text{in}</th>
<th>A</th>
<th>B</th>
<th>C\text{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>

S = C\text{out} + A B + C\text{in}

ENGR 303
1-Bit Adders

**Half Adder**

\[ S = A \oplus B \]
\[ C_{out} = AB \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</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>

**Full Adder**

\[ S = A \oplus B \oplus C_{in} \]
\[ C_{out} = AB + AC_{in} + BC_{in} \]

**Verilog**

module fulladder (input a, b, cin, output reg s, cout);
reg p, g;
always @ (*)
begin
    p <= a^ b;
    g <= a & b;
    s <= p ^ cin;
    cout <= g | (p & cin);
end
endmodule
Multibit Adders (CPAs)

- Types of carry propagate adders (CPAs):
  - Ripple-carry (slow)
  - Carry-lookahead (fast)
  - Prefix (faster)

- Carry-lookahead and prefix adders faster for large adders but require more hardware
Ripple-Carry Adder

- Chain 1-bit adders together
- Carry ripples through entire chain
- Disadvantage: slow
Ripple-Carry Adder Delay

\[ t_{\text{ripple}} = N t_{FA} \]

where \( t_{FA} \) is the delay of a 1-bit full adder
Carry-Lookahead Adder

• Compute carry out ($C_{out}$) for $k$-bit blocks using *generate* and *propagate* signals

• Some definitions:
  – Column $i$ produces a carry out by either *generating* a carry out or *propagating* a carry in to the carry out
  – Generate ($G_i$) and propagate ($P_i$) signals for each column:
    • Column $i$ will generate a carry out if $A_i$ AND $B_i$ are both 1.
      \[
      G_i = A_i B_i
      \]
    • Column $i$ will propagate a carry in to the carry out if $A_i$ OR $B_i$ is 1.
      \[
      P_i = A_i + B_i
      \]
    • The carry out of column $i$ ($C_i$) is:
      \[
      C_i = A_i B_i + (A_i + B_i)C_{i-1} = G_i + P_i C_{i-1}
      \]
Carry-Lookahead Addition

- **Step 1**: Compute $G_i$ and $P_i$ for all columns
- **Step 2**: Compute $G$ and $P$ for $k$-bit blocks
- **Step 3**: $C_{in}$ propagates through each $k$-bit propagate/generate block
Carry-Lookahead Adder

• **Example:** 4-bit blocks \((G_{3:0} \text{ and } P_{3:0})\):

\[
G_{3:0} = G_3 + P_3 \left( G_2 + P_2 \left( G_1 + P_1 G_0 \right) \right)
\]

\[
P_{3:0} = P_3 P_2 P_1 P_0
\]

• **Generally,**

\[
G_{i:j} = G_i + P_i \left( G_{i-1} + P_{i-1} \left( G_{i-2} + P_{i-2} G_j \right) \right)
\]

\[
P_{i:j} = P_i P_{i-1} P_{i-2} P_j
\]

\[
C_i = G_{i:j} + P_{i:j} C_{j-1}
\]
32-bit CLA with 4-bit Blocks

ENGR 303
Carry-Lookahead Adder Delay

For $N$-bit CLA with $k$-bit blocks:

$$t_{CLA} = t_{pg} + t_{pg\_block} + (N/k - 1)t_{AND\_OR} + kt_{FA}$$

- $t_{pg}$: delay to generate all $P_i, G_i$
- $t_{pg\_block}$: delay to generate all $P_{i:j}, G_{i:j}$
- $t_{AND\_OR}$: delay from $C_{in}$ to $C_{out}$ of final AND/OR gate in $k$-bit CLA block

An $N$-bit carry-lookahead adder is generally much faster than a ripple-carry adder for $N > 16$
Prefix Adder

- Computes carry in \((C_{i-1})\) for each column, then computes sum:
  \[ S_i = (A_i \oplus B_i) \oplus C_i \]

- Computes \(G\) and \(P\) for 1-, 2-, 4-, 8-bit blocks, etc. until all \(G_i\) (carry in) known

- \(\log_2 N\) stages
Prefix Adder

- Carry in either *generated* in a column or *propagated* from a previous column.
- Column -1 holds $C_{in}$, so 
  $$G_{-1} = C_{in}, \ P_{-1} = 0$$
- Carry in to column $i = \text{carry out of column } i-1$:
  $$C_{i-1} = G_{i-1:-1}$$
  $G_{i-1:-1}$: generate signal spanning columns $i-1$ to -1
- Sum equation:
  $$S_i = (A_i \oplus B_i) \oplus G_{i-1:-1}$$
- **Goal:** Quickly compute $G_{0:-1}, G_{1:-1}, G_{2:-1}, G_{3:-1}, G_{4:-1}, G_{5:-1}, \ldots$ (called *prefixes*)
Prefix Adder

• Generate and propagate signals for a block spanning bits $i:j$:
  
  \[ G_{i:j} = G_{i:k} + P_{i:k} G_{k-1:j} \]
  
  \[ P_{i:j} = P_{i:k} P_{k-1:j} \]

• In words:
  
  – **Generate**: block $i:j$ will generate a carry if:
    
    • upper part ($i:k$) generates a carry or
    
    • upper part propagates a carry generated in lower part ($k-1:j$)
  
  – **Propagate**: block $i:j$ will propagate a carry if *both* the upper and lower parts propagate the carry
Prefix Adder Schematic

Legend

A_i B_i

P_{i,i} G_{i,i}

P_{i,j} P_{k-1,j} G_{i,k} G_{k-1,j}

G_{i-1:j} A_i B_i

S_i
Prefix Adder Delay

\[ t_{PA} = t_{pg} + \log_2 N(t_{pg\_prefix}) + t_{XOR} \]

- \( t_{pg} \): delay to produce \( P_i \) \( G_i \) (AND or OR gate)
- \( t_{pg\_prefix} \): delay of black prefix cell (AND-OR gate)
Adder Delay Comparisons

Compare delay of: 32-bit ripple-carry, carry-lookahead, and prefix adders

- CLA has 4-bit blocks
- 2-input gate delay = 100 ps; full adder delay = 300 ps
Adder Delay Comparisons

Compare delay of: 32-bit ripple-carry, carry-lookahead, and prefix adders

- CLA has 4-bit blocks
- 2-input gate delay = 100 ps; full adder delay = 300 ps

\[ t_{\text{ripple}} = N t_{FA} = 32(300 \text{ ps}) \]
\[ = 9.6 \text{ ns} \]

\[ t_{CLA} = t_{pg} + t_{pg\_block} + (N/k - 1)t_{\text{AND\_OR}} + kt_{FA} \]
\[ = [100 + 600 + (7)200 + 4(300)] \text{ ps} \]
\[ = 3.3 \text{ ns} \]

\[ t_{PA} = t_{pg} + \log_2 N(t_{pg\_prefix}) + t_{\text{XOR}} \]
\[ = [100 + \log_2 32(200) + 100] \text{ ps} \]
\[ = 1.2 \text{ ns} \]
Subtractor

Symbol | Implementation
--- | ---

\[ \begin{align*}
A & \quad B \\
\text{Symbol} & \quad \text{Implementation}
\end{align*} \]

Verilog
module subtractor #(parameter N = 8) (input [N-1:0] a, b, output [N-1:0] y);
assign y = a – b;
endmodule