8. Design of Adders

Jacob Abraham

Department of Electrical and Computer Engineering
The University of Texas at Austin

VLSI Design
Fall 2020

September 22, 2020
Single-Bit Addition

Half Adder

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

<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>1</td>
<td>0</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>1</td>
</tr>
</tbody>
</table>

Full Adder

\[ S = A \oplus B \oplus C \]
\[ C_{out} = \text{MAJ}(A, B, C) \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</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>
Brute force implementation from equations

\[ S = A \oplus B \oplus C \]

\[ C_{\text{out}} = \text{MAJ}(A, B, C) \]
Factor $S$ in terms of $C_{out}$

$$S = A \cdot B \cdot C + (A + B + C) \cdot \overline{C_{out}}$$

Critical path is usually $C$ to $C_{out}$ in ripple adder
Clever layout circumvents usual line of diffusion
- Use wide transistors on critical path
- Eliminate output inverters
- Complementary Pass Transistor Logic (CPL)
  - Slightly faster, but more area
Simplest design: cascade full adders
- Critical path goes from $C_{in}$ to $C_{out}$
- Design full adder to have fast carry (small delay for carry signal)
- Critical path passes through majority gate
  - Built from minority + inverter
  - Eliminate inverter and use inverting full adder
N-bit adder called CPA
- Each sum bit depends on all previous carries
- How do we compute all these carries quickly?
Carry Propagate, Generate, Kill (P, G, K)

For a full adder, define what happens to carries

- **Generate**: \( C_{out} = 1 \), independent of \( C \)
  - \( G = A \cdot B \)
- **Propagate**: \( C_{out} = C \)
  - \( P = A \oplus B \)
- **Kill**: \( C_{out} = 0 \), independent of \( C \)
  - \( K = \overline{A} \cdot \overline{B} \)

Generate and Propagate for groups spanning \( i:j \)

- \( G_{i:j} = G_{i:k} + P_{i:k} \cdot G_{k-1:j} \)
- \( P_{i:j} = P_{i:k} \cdot P_{k-1:j} \)

Base Case

- \( G_{i:i} \equiv G_i = A_i \cdot B_i \), \( G_{0:0} = G_0 = C_{in} \)
- \( P_{i:i} \equiv P_i = A_i \oplus B_i \), \( P_{0:0} = P_0 = 0 \)

Sum: \( S_i = P_i \oplus G_{i-1:0} \)
Carry Propagate, Generate, Kill (P, G, K)

For a full adder, define what happens to carries

- **Generate**: $C_{out} = 1$, independent of $C$
  - $G = A \cdot B$
- **Propagate**: $C_{out} = C$
  - $P = A \oplus B$
- **Kill**: $C_{out} = 0$, independent of $C$
  - $K = \overline{A} \cdot \overline{B}$

Generate and Propagate for groups spanning $i:j$

- $G_{i:j} = G_{i:k} + P_{i:k} \cdot G_{k-1:j}$
- $P_{i:j} = P_{i:k} \cdot P_{k-1:j}$

Base Case

- $G_{i:i} \equiv G_i = A_i \cdot B_i$, $G_{0:0} = G_0 = C_{in}$
- $P_{i:i} \equiv P_i = A_i \oplus B_i$, $P_{0:0} = P_0 = 0$

Sum: $S_i = P_i \oplus G_{i-1:0}$
\[ G_{i:0} = G_i + P_i \cdot G_{i-1:0} \]
\[ t_{\text{ripple}} = t_{pg} + (N - 1)t_{AO} + t_{xor} \]
- Carry-ripple is slow through all $N$ stages
- Carry-skip allows carry to skip over groups of $n$ bits
  - Decision based on $n$-bit propagate signal
For $k$ $n$-bit groups ($N = nk$)

$$t_{skip} = t_{pg} + [2(n - 1) + (k - 1)] t_{AO} + t_{xor}$$
- Carry-lookahead adder computes $G^i:0$ for many bits in parallel
- Uses higher-valency cells with more than two inputs
Higher Valency Cells
Carry-Select Adder

- Trick for critical paths dependent on late input $X$
  - Precompute two possible outputs for $X = 0, 1$
  - Select proper output when $X$ arrives
- Carry-select adder precomputes $n$-bit sums for both possible carries into $n$-bit group
Tree Adders

- Tree structures can be used to speed up computations
- Look at computing the XOR of 8 bits using 2-input XOR-gates

If lookahead is good for adders, lookahead across lookahead!

- Recursive lookahead gives $O(\log N)$ delay
- Many variations on tree adders
Brent-Kung Adder
Kogge-Stone Adder
Tree Adder Taxonomy

- Ideal N-bit tree adder would have
  - \( L = \log N \) logic levels
  - Fanout never exceeding 2
  - No more than one wiring track between levels

- Describe adder with 3-D taxonomy \((l, f, t)\)
  - Logic levels: \( L + l \)
  - Fanout: \( 2f + 1 \)
  - Wiring tracks: \( 2^t \)

- Known tree adders sit on plane defined by \( l + f + t = L - 1 \)
Tree Adder Taxonomy, Cont’d

Diagram of different types of adders:
- Sklansky
- Kogge-Stone
- Brent-Kung

The diagram illustrates the hierarchy and connections between these adder types, showing how they are related in terms of logic levels and wire tracks.
Knowles [2,1,1,1] Adder
Adder architectures offer area/power/delay tradeoffs

Choose the best one for your application

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Classification</th>
<th>Logic levels</th>
<th>Max. fanout</th>
<th>Tracks</th>
<th>Cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ripple Carry</td>
<td>N − 1</td>
<td>1</td>
<td>1</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>Carry-skip(n=4)</td>
<td>N/4 + 5</td>
<td>2</td>
<td>1</td>
<td>1.25N</td>
<td></td>
</tr>
<tr>
<td>Carry-inc.(n=4)</td>
<td>N/4 + 2</td>
<td>4</td>
<td>1</td>
<td>2N</td>
<td></td>
</tr>
<tr>
<td>Brent-Kung</td>
<td>(L-1,0,0)</td>
<td>2log₂ N−1</td>
<td>2</td>
<td>2N</td>
<td></td>
</tr>
<tr>
<td>Sklansky</td>
<td>(0,L-1,0)</td>
<td>log₂ N</td>
<td>N/2 + 1</td>
<td>0.5Nlog₂ N</td>
<td></td>
</tr>
<tr>
<td>Kogge-Stone</td>
<td>(0,0,L-1)</td>
<td>log₂ N</td>
<td>2</td>
<td>N/2</td>
<td>Nlog₂ N</td>
</tr>
</tbody>
</table>