Making Circuits Faster

- Carry lookahead
- Other circuits with linear structure
- Multiplication

Jon Turner
Size vs. Speed

- What’s the “best circuit” for a given application?
  - to reduce cost, want circuit that uses least amount of resources
  - but for best performance (highest clock rate) want circuits with low delay – short circuit paths, limited fanout
  - often small circuits are also fast, but not always

- Focus on arithmetic circuits
  - often key to good overall system performance
  - well-developed methods for making them faster
  - general approaches can be applied in other settings
Increment Circuit

- Ripple-carry incrementer based on standard addition algorithm
  - small circuit but long carry-chain limits performance
  - alternate version (omitting carry-in) uses tree-structured carry logic
    - short input-to-output paths, but large fanout
Better Lookahead Carry Circuit

- $n$ bit version has $\lg n$ rows
  $\approx n \lg n$ gates, fanout=2
  - for $n=64$, longest path is 6 gates instead of 64
  - but also $\approx 342$ gates vs 64

General construction

- define $A(i,j)=a_i, a_{i-1}, \ldots, a_j$
- 1st row implements $A(i,i-1)$
- 2nd row implements $A(i,i-3)=A(i,i-1)A(i-2,i-3)$
- 3rd row implements $A(i,i-7)=A(i,i-3)A(i-4,i-7)$
- $k$th row implements
  $$A(i,i-(2^k-1))=A(i,i-(2^{k-1}-1))A(i-2^{k-1},i-(2^{k-1}))$$
Exercises

1. Recall that we can compute the 2s-complement for an input signal X, by finding the rightmost 1 in X and flipping all the bits to the left of this position. Use this to design a 4 bit 2s-complement circuit that is similar to the ripple-carry increment circuit.

2. How could you modify the lookahead carry circuit for the incrementer so that it could be used in a 2s-complement circuit?

3. Consider a binary number 10101000. If we subtract 1 from this, we get 10100111. Notice that the four rightmost bits have been flipped in the result, while the others remain unchanged. In general, to subtract 1 from a binary value, we can just flip the bits up to and including the rightmost 1. Use this to design a decrement circuit that is similar to the ripple-carry decrement circuit.

4. How could you modify the lookahead carry circuit for the incrementer so that it could be used in a decrement circuit?
## Binary Addition

- Binary long addition similar to decimal long addition

<table>
<thead>
<tr>
<th></th>
<th>decimal</th>
<th>binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>carry</td>
<td>110</td>
<td>111100</td>
</tr>
<tr>
<td>augend</td>
<td>2565</td>
<td>[10110 propagates]</td>
</tr>
<tr>
<td>addend</td>
<td>6754</td>
<td>[10111 generates carry]</td>
</tr>
<tr>
<td>sum</td>
<td>9319</td>
<td>110001</td>
</tr>
</tbody>
</table>

- Binary addition algorithm - add \( a_{n-1}...a_0 \) to \( b_{n-1}...b_0 \) giving \( s_{n}...s_0 \)

  \( c_0 = 0 \)  
  \[
  s_i = a_i \oplus b_i \oplus c_i \\
  c_{i+1} = a_i b_i + a_i c_i + b_i c_i = a_i b_i + ((a_i \oplus b_i) c_i) \\
  s_n = c_n
  \]  
  // \( c_i \) are carry bits
Ripple-Carry Adder

- Generate signal \( g_i = a_i b_i \), propagate signal \( p_i = a_i \oplus b_i \)
- generalize generate/propagate to span ranges
  - \( P(i,j) = p_j p_{j-1} \ldots p_i \) can be implemented using lookahead circuit from incrementer
  - \( G(i,i-(2^k-1)) = G(i,i-(2^{k-1}-1)) + (G(i-(2^k-1),i-(2^{k-1}-1)))P(i,i-(2^{k-1}-1)) \)
Binary Coded Decimal (BCD)

- Some systems represent data in BCD to facilitate data conversion
  - standard BCD: 946 ⇒ 1001 0100 0110
  - excess-3 form: 946 ⇒ 1100 0111 1001
- Implement addition using ripple-carry circuit
Lookahead Adder
Linear Circuit Patterns

- Ripple-carry increment and addition circuits are examples of a common linear circuit pattern
  - copies of a common “block” with one or more signals between adjacent blocks

- Other circuits with similar pattern
  - 2s-complemeneter, maximum, comparison, tally,...

- Propagation delay for such circuits typically grows in proportion to number of blocks

- Look-ahead versions can have propagation delays that grow with logarithm of number of blocks
Comparison Circuit

- Compare bits from most-significant down to least-significant
  - first position where bits differ determines which is larger
    - so if $A=010110$ and $B=010011$, $A$ is larger because it is 1, in first bit where the two differ

- Circuit constructed from repeated elements in linear array
Binary Multiplication

- Binary multiplication is done much like decimal multiplication

  \[
  \begin{array}{c}
  \text{multiplicand} \\
  1101 \\
  1010 \\
  \hline
  \text{multiplier} \\
  0000 \\
  1101 \\
  0000 \\
  1101 \\
  \hline
  \text{product} \\
  10000010
  \end{array}
  \]

- Works for signed values too
  > first extend signs of operands

- Requires 1 bit multipliers (AND gates) and adders

- With ripple-carry adders, time is roughly 4 times adder delay
Making Multiplication Faster

- Replacing ripple-carry adder with lookahead adder “should” speedup multiplier, but doesn’t
  - increases length of worst-case path

- To get faster circuit use “adder-tree” to combine partial products
  - using lookahead adders, about 10x faster than basic multiplier for $n=64$
  - using ripple-carry adders, it’s worse than original version
Exercises

1. The binary subtraction algorithm subtracts bits from right-to-left, computing both difference bits and borrow bits as it goes. Draw a circuit that implements a ripple-carry subtracter based on this algorithm. Under what conditions does a bit position in the subtracter generate a borrow. When does it propagate a borrow.

2. What is the length of the longest circuit path in a 16 digit BCD adder, assuming the conventional encoding of the digits. Just count the total number of gates on the path. Also assume that the >9 comparator has a worst-case path that passes through 6 gates. How does this change if we use the excess-three representation instead?

3. To apply the lookahead carry circuit to a BCD adder, we need a generate and a propagate signal for each BCD adder block. Under what circumstances does a BCD adder generate a carry? Under what circumstances does it propagate a carry?

4. Let $A$ and $B$ be 8 bit logic vectors. Observe that
   - if $A(7 .. 4) > B(7 .. 4)$ then $A > B$ if and only if $A(7 .. 4) > B(7 .. 4)$ and
   - if $A(7 .. 4) = B(7 .. 4)$ then $A > B$ if and only if $A(3 .. 0) > B(3 .. 0)$

   Use this to design a “recursive circuit” that compares $A$ and $B$. Show that the delay for a general version of this circuit grows as the logarithm of the number of bits.