FASTER CARRY-PROPAGATE ADDERS

Faster Carry-Propagate Adders

- The entire goal to make faster adders is to resolve the carry across the entire adder structure more quickly
- It should be perplexing at first glance how this could be possible given the dependence of *every* output bit on the *LSB input bits*
- A few common faster CPAs:

1) Carry Select
   - Speculatively add and select later
2) Carry Lookahead
   - Look at how a carry propagates through a group of bits
3) Conditional-sum (recursive carry select)
4) Carry skip
5) Other parallel prefix adders
   - Kogge-Stone, 1973
   - Brent-Kung, 1982
   - etc.

1) Carry Select Adder

- Break ripple adder into pieces
- Compute each sub-block (except the one covering the least-significant bits) twice: once assuming the carry input is a "0" and once assuming the input is a "1"
- Each sub-block computes 1) sum bits and 2) a single carry bit
- A mux selects correct sum+carry bits when the previous block's carry-out (the carry-in of the block containing the mux) is known
- This method can be sped up further with a hierarchical structure (conditional-sum)

2) Carry Lookahead Adder

- Break ripple adder into pieces
- Look at the bits inside of each piece and decide two things *based only on the input operands and independent of the sub-block's carry-in*
  - Will this sub-block generate a carry-out regardless of the carry-in (*generate*)
  - Will the carry-out be equal to the value of the carry-in (*propagate*)
  - Other variations include a condition to stop a carry (*kill*)
- In the simplest form, the carry-out can be calculated by,

  $c_{out} = Generate \text{ OR } (Propagate \text{ AND } c_{in})$
- Key point: Each sub-block pre-examines the input operand bits and gets ready for fast carry-out calculation
- There are a number of more complicated variations
- This method can be sped up further with a hierarchical structure

2) Carry Lookahead Adder

- When is Generate = 1?
  - When c_{in} = 0 and c_{out} = 1
  - When a + b = {c_{out} , sum} = {1xxx…xxx} with c_{in} = 0
  - For example, a[3:0] + b[3:0] = 1xxxx with c_{in} = 0
  - It will be true that Generate = 1 when c_{in} = **1** and c_{out} = 1, but that is not sufficient to show the cases when Generate = 1
- When is Propagate = 1?
  - When a + b = {c_{out} , sum} = {0111…111}
  - For example, a[3:0] + b[3:0] = 01111