CSE471—Computer Design and Organization

Prof. Steven Burns *Autumn 1994*

Lecture Notes: Sixth Day

Clean Way to Look at Carry-Lookahead

- From *Cormen, Leiserson, and Rivest*
- We want a unit that given the *kill*, *propagate* and *generate* (`kpg`) status of each bit, predicts the carry at each bit.
- Let $x_1, x_2, x_3, \ldots, x_{32}$ be the input statuses of the bits 0 through 31. Let $y_0, y_1, y_2, \ldots, y_{31}, y_{32}$ be the output statuses of each bit.
- The x values are `k`, `p`, or `g`. The y values will be either `k` or `g`.

Computing the Statuses and Result

- We can easily compute the input status of a bit: it is `k` if a and b are 0, it is `g` if a and b are 1, and it is `p` if a and b are different.
- It is also easy to compute the CarryOut value from an output status: it is 1 if y_i is `g` and it is 0 if y_i is `k`.

Carry-Lookahead

- We compute the output values by the equations:

$$\begin{aligned} y_0 &= x_0 \\ y_i &= y_{i-1} \otimes x_i \\ &= x_0 \otimes x_1 \otimes \ldots \otimes x_i \end{aligned}$$

  where $w \otimes x$ corresponds to the table:

		x		
	$\otimes$	k	p	g
	k	k	k	g
w	p	k	p	g
	g	k	g	g

- We want to compute the y_i's quickly. We use something called the *parallel-prefix paradigm*.

Parallel Prefixes

- The trick is that $\otimes$ is a binary, associative operation. (Although it is not commutative).
- Let $[i,j]$ represent $x_i \otimes x_{i+1} \otimes \ldots \otimes x_j$.
- We can compute $[0,7]$ in different ways:

$$\begin{array}{rcl} [0,1] &=& x_0 \otimes x_1 \\ [0,2] &=& [0,1] \otimes x_2 \\ [0,3] &=& [0,2] \otimes x_3 \\ [0,4] &=& [0,3] \otimes x_3 \\ [0,5] &=& [0,4] \otimes x_3 \\ [0,6] &=& [0,5] \otimes x_3 \\ [0,7] &=& [0,6] \otimes x_7 \end{array} \qquad \begin{array}{rcl} [0,1] &=& x_0 \otimes x_1 \\ [2,3] &=& x_2 \otimes x_3 \\ [4,5] &=& x_4 \otimes x_5 \\ [6,7] &=& x_6 \otimes x_7 \\ [0,3] &=& [0,1] \otimes [2,3] \\ [4,7] &=& [4,5] \otimes [6,7] \\ [0,7] &=& [0,3] \otimes [4,7] \end{array}$$

- So we know how to compute any particular $y_j = [0,j]$ quickly.

Prefix Tree

- How do we compute all the $[0,j]$'s quickly using a small circuit?
- Assume we have computed:

$$[0,i-1], [i,j-1], \text{ and } [j,k]$$

  From these values we can compute:

$$\begin{array}{rcl} [i,k] &=& [i,j-1] \otimes [j,k] \\ [0,j-1] &=& [0,i-1] \otimes [i,j-1] \end{array}$$

  We direct the first value up the tree and the second value down to the right. We also pass $[0,j-1]$ down to the left.

- Using this tree structure, we can compute all the y_i's in $O(\log n)$ and using $2n-1$ circuits implementing $\otimes$.

Finishing Touches

- To finish off the design we need to know how to compute the sum value at each bit and how to implement the $\otimes$ operations.

- We start by encoding the three statuses `k`, `p`, and `g` on two wires named P and G.

	PG
`k`	00
`p`	10
`g`	01
$-$	11

Encodings

- We compute the P and G signals for x_i by

$$\begin{aligned} xP_i &= a_{i-1} \oplus b_{i-1} \\ xG_i &= a_{i-1}\, b_{i-1} \end{aligned}$$

- We compute the result (sum) for the i^{th} bit by

$$\begin{aligned} s_i &= xP_{i+1} \oplus yG_i \\ &= a_i \oplus b_i \oplus c_i \end{aligned}$$

Implementation of $\otimes$

- We can find an implementation of $y = w \otimes x$ by converting our "times table" into two Karnaugh maps:

yP		$xP\,xG$	00 `k`	10 `p`	11	01 `g`
	00	`k`	0	0	X	0
$wP\,wG$	10	`p`	0	1	X	0
	11		X	X	X	X
	01	`g`	0	0	X	0

yG		$xP\,xG$	00 `k`	10 `p`	11	01 `g`
	00	`k`	0	0	X	1
$wP\,wG$	10	`p`	0	0	X	1
	11		X	X	X	X
	01	`g`	0	1	X	1

- So, we can use

$$\begin{aligned} yP &= wP\,xP \\ yG &= wG\,xP + xG \end{aligned}$$

Other Types of Fast Adders

- Carry-select—duplicate adder circuitry, perform the addition for the different possible CarryIn's, decide afterward which result to use.
- Carry-bypass—if a block produces all propagates, bypass the CarryIn of the block directly to the CarryOut.
- Carry-save—use a redundant form to store the result. Useful if many numbers are to be added together. (Fast multipliers).