More Arithmetic Circuits

CS255

Chris Pollett

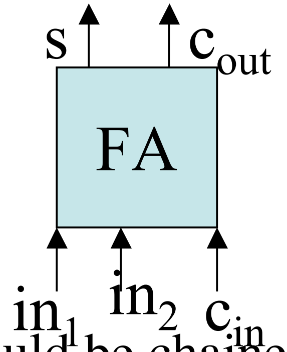
Feb. 27, 2006.

Outline

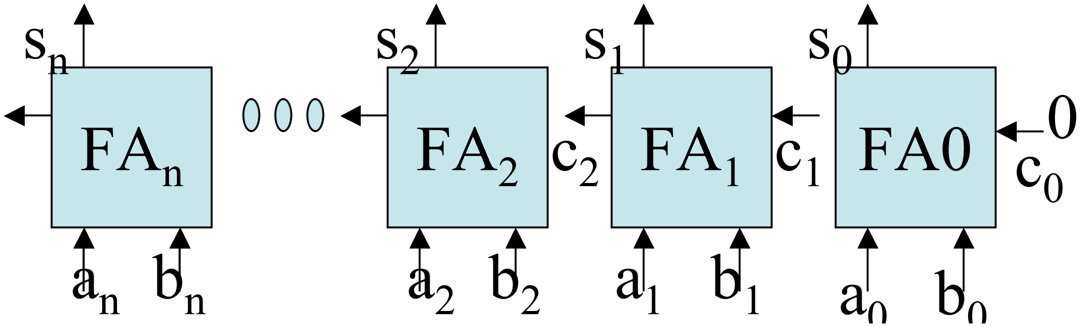
- Circuits for Addition

Ripple Carry Addition

- Last day on the board we considered circuits to count the number of `on’ bits in an n bit number.
- Today, we’ll look at circuits for adding two n-bit numbers.
- We’ll make use of a two bit full adders to do this:



- These could be chained together to do addition as follows:



Carry Lookahead Addition

- Ripple-carry addition has both size and depth O(n).
- We now look at how to reduce the depth.
- We can make a table of the carry status versus the status on the inputs to FA_{i-1}.

a_{i-1}	b_{i-1}	c_i	status
0	0	0	k
0	1	c_{i-1}	p
1	0	c_{i-1}	p
1	1	1	g

k - kill

p - propagate

g - generate

The Carry Status Operator

- Notice just from the carry status of FA_i and FA_{i-1} we can determine the carry status that will be output from the combined circuit according to the following table:

		FA_i		
	⊗	k	p	g
FA_{i-1}	k	k	k	g
	p	k	p	g
	g	k	g	g

- This operation called the **carry-status operator** and is associative.

An Faster Algorithm

- This suggests an algorithm to do addition:
  1. Compute the carry status operator of each full adder: x_i := k if $a_{i-1}=b_{i-1}=0$; x_i := p if $a_{i-1} \neq b_{i-1}$; x_i := g if $a_{i-1}=b_{i-1}=1$.
  2. Determine the value of $y_i = x_0 \otimes x_1 \otimes \ldots \otimes x_i$ for each i this is called a **prefix computation**.
  3. Use this to determine the value of c_i in constant size and depth.
  4. From the value of c_i, a_i, b_i figure out the given output bit of the circuit.
- Steps 1,3,4 can obviously be done in parallel. It turns out so can step 2. The next lemma says why step 3 is possible.

Lemma 29.1

Define x_i and y_i as above. For i=0,…,n the following conditions hold:

1. y_i=k implies c_i=0,
2. y_i=g implies c_i=1, and
3. y_i=p does not occur.

Proof of Lemma 29.1

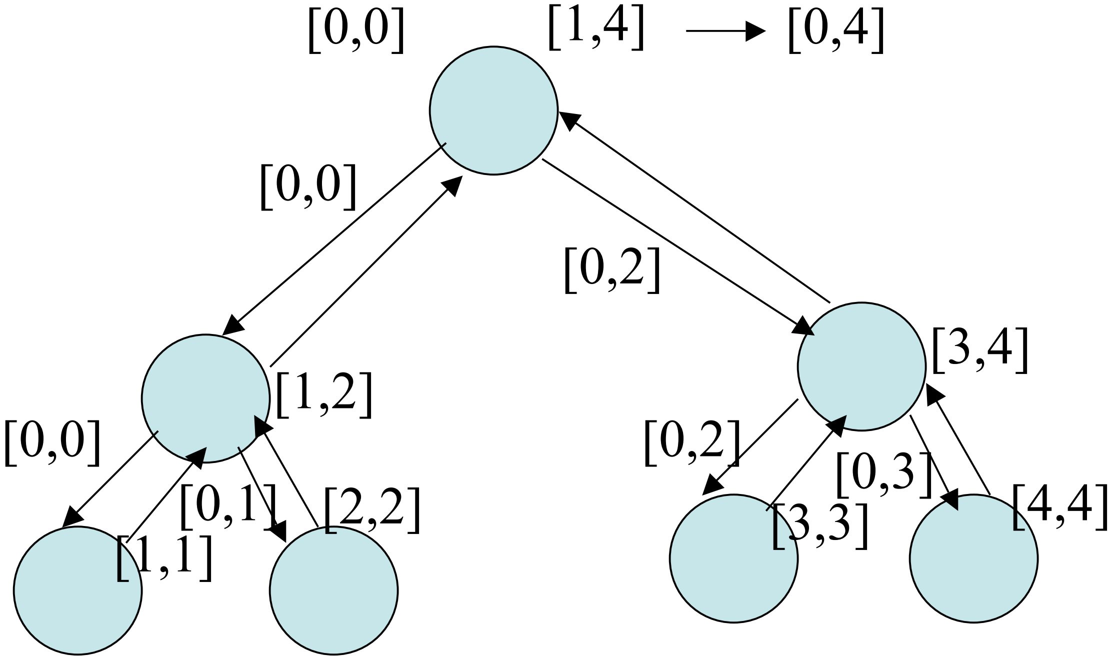
The proof is by induction on i. When i=0, we have $y_0=x_0=k$ by definition, and so $c_0=0$ too. For the inductive step, assume the lemma hold for i-1. There are three possible cases:

1. $y_i=k$, then since $y_i= y_{i-1}\otimes x_i$, either $x_i=k$ or $x_i=p$ and $y_{i-1}=k$. If $x_i=k$ then $a_{i-1}=b_{i-1}=0$, so $c_i=0$. If $x_i=p$ and $y_{i-1}=k$, then $a_{i-1}\neq b_{i-1}$ and by induction $c_{i-1}=0$. Thus, $c_i=\text{majority}(a_{i-1},b_{i-1,},c_{i-1})=0$.
2. If $y_i=g$, then either we have $x_i=g$ or we have $x_i=p$ and $y_{i-1}=g$. If $x_i=g$, then $a_{i-1}=b_{i-1}=1$, so $c_i=1$. If $x_i=p$ and $y_{i-1}=g$, then $a_{i-1}\neq b_{i-1}$ and by induction $c_{i-1}=1$. So $c_i=1$.
3. If $y_i=p$, then we must have $y_{i-1}=p$, but this contradicts the inductive hypothesis.

Determining the Value of y_i

- So to complete our description of our circuit we need to say how to compute the value of the y_i's.
- Let $[i,j] = x_i \otimes x_1 \otimes \ldots \otimes x_j$. So $y_i = [0, i]$
- Since the carry status operator is associative we have $[i,k] = [i,j-1] \otimes [j,k]$.
- The next slide give an illustrative example of the general divide and conquer circuit we'll use.

Circuit for y_i



The tree has log depth need to compute up and then down the tree so have twice this total depth. So overall circuit will be log depth.

Carry-Save Addition

- Suppose wanted to add three n-bit numbers x,y,z together.
- We could do this with only constant overhead by using the full adder on three inputs to reduce the situation to the two n-bit number case.
- We make an n bit number u and an n+1 bit number v such that u+v = x+y+z.
- u_i = parity(x_i, y_i, z_i) , v_0 =0, v_{i+1}= majority(x_i, y_i, z_i)
- Then u, v are added with the carry-lookahead adder.