CS255 Homework 3

Student Generated Solutions

April 8, 2006

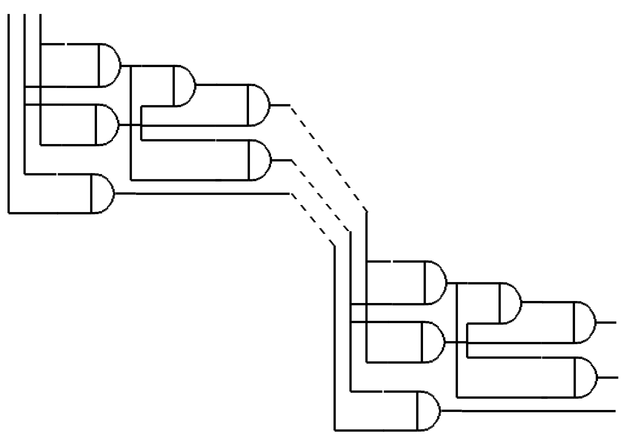
29.1-6 Let C be an n-input, n-output combinational circuit of depth d. If two copies of C are connected, with the outputs of one feeding directly into the inputs of the other, what is the maximum possible depth of this tandem circuit? What is the minimum possible depth?

Assumption : If circuit C has elements x_1,x_2,...,x_n at depths d_1,d_2,...,d_n respectively, then the max$\{d_1,d_2,...,d_n\}$ is depth d.

The maximum possible depth : 2d.

It can be achieved when maximum depth path of the first C which is max$\{d_1,d_2,...,d_n\}$ is connected with maximum depth of the second C which is max$\{d_1,d_2,...,d_n\}$.

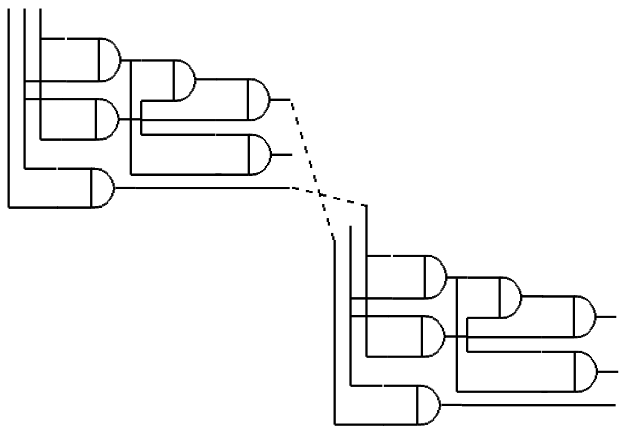
The following figure shows an example configuration where we acheive a maximum possible depth:



The minimum possible depth : d + min$\{d_1,d_2,...d_n\}$.

It can be achieved when maximum depth path of the first C which is max$\{d_1,d_2,...,d_n\}$ connected with minimum depth of the second C which is min$\{d_1,d_2,...d_n\}$. Where the min could be as small as 1.

The following figure shows an example configuration where we acheive a minimum possible depth:



29.2-1 let a= (0 1 1 1 1 1 1 1) , b (0 0 0 0 0 0 0 1), and n= 8 . Show the sum and carry bits output by full adders when ripple-carry addition is performed on these two sequences. Show the carry status x_0,x_1,...,x_8 corresponding to a and b, label each wire of the parallel prefix circuit of Figure 29.9 with the value it has given these x; inputs, and show the resulting outputs y_0,y_1,...,y_8

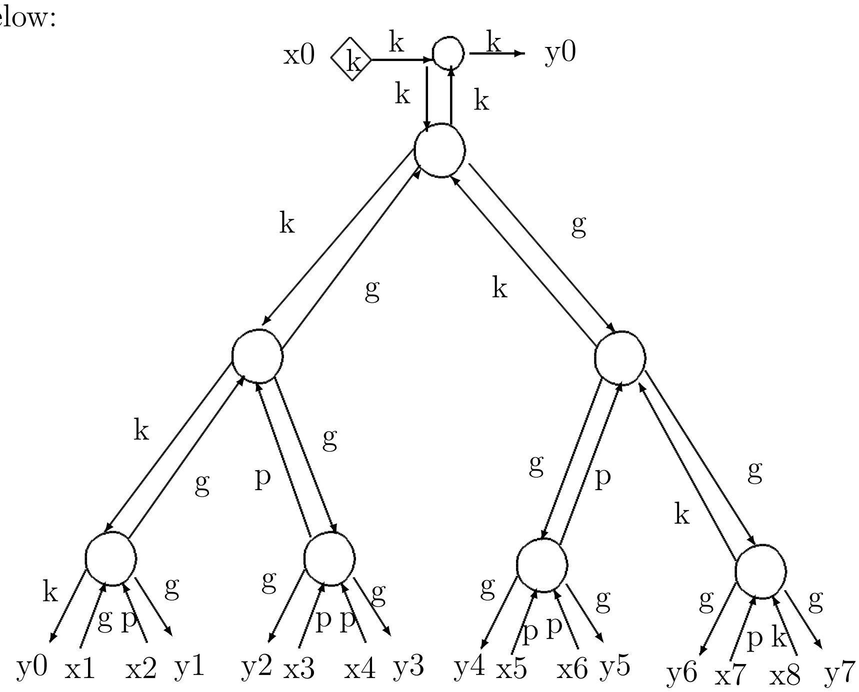
The sum and the carry bits output by full adders are as follows:

8 7 6 5 4 3 2 1 0		i
0 1 1 1 1 1 1 1 0	=	carry
- 0 1 1 1 1 1 1 1	=	a
- 0 0 0 0 0 0 0 1	=	b
0 1 0 0 0 0 0 0 0	=	sum

The values of x_i and y_i for i= 0,1,...,8 that correspond to the values of a_i,b_i,and c_i are as shown below:

a_i		0	1	1	1	1	1	1	1
b_i		0	0	0	0	0	0	0	1
x_i	k	p	p	p	p	p	p	g	k
y_i	k	g	g	g	g	g	g	g	k
c_i	0	1	1	1	1	1	1	1	0

A parallel prefix circuit that correspond to the figures above is shown below:



29.3-5 Describe an efficient circuit to compute the quotient when a binary number x is divided by 3. Consider the equation $\frac{4}{3} = \frac{1}{1-u}$. Solving for u gives $u = 1/4$. Recall

$$1 + u + u^2 + \cdots u^k = \frac{1 - u^{k+1}}{1 - u}.$$

We would like $\frac{x}{4(1-u)} - \frac{x(1-u^{k+1})}{4(1-u)}$ to be less than 1. This implies $x \cdot u^{k+1} < 4 - 1 = 3$. Since $u = 1/4$ and x has n bits. This will happen when $2^n/2^{2k+1} < 3$ which will sure;y hold if $k > \lceil n/2 \rceil$. Notice $1 + u + u^2 + \cdots u^k$ with be thus the string 10101.. of length n. So our circuit for division by 3 consists of our log-depth circuit multiplication of x times this hard-coded bit pattern. We then discard the two low order bits of the ouput/

12.2 Devise a PRAM algorithm by which, given b_i, the S_i can be computed (with the result contained in P_i) in O(log n) steps. Using this, show how Stage 3 of the algorithm can be implemented in O(log n) steps.

To compute S_i in O(log n) steps, we simulate arranging the processors P_j, $j = 1 \ldots i$, in a binary tree fashion. Each processor is essentially a node in this tree, each non-leaf node will receive two numbers, one from each child, calculates the sum and passes the result to its parent. By $\lceil logn \rceil$ steps, the sum is contained at the root of the tree. We accomplish this with a parallel algorithm that has the processors communicating in a binary tree fashion, using array locations to store intermediate sums.

The folowing PRAM algorithm, using i processors, takes as input an array $b[1 \ldots i]$ and terminates with S_i stored in $b[j = i]$

for $k \leftarrow 0 \quad to \quad \lceil logn \rceil - 1$
  for $all \quad j \quad in \quad parallel$
    $incr \leftarrow 2^k$
    if $(j \ mod \ (2 \cdot incr)) = 0$
      $b[j] \leftarrow b[j - incr] + b[j]$

This S_i result can be used to implement stage 3 of the PRAM quicksort algorithm in O(log n) steps. For a processor P_i, S_i indicates the number of processors from 1 *to* i that has a value less than P_{pivot}'s, including itself. For all $i > pivot$, if P_i's value is greater than P_{pivot}'s value, then P_i does nothing. Otherwise, P_i does a binary search on the values contained in the processors with indeces less than pivot, looking for a value that is larger than the pivot's value.

We handle contentions by calculating the difference between S_i and S_{pivot}, let that difference be called x. So to avoid having 2 or more "right hand" processors swapping with the same "left hand" processor we swap with the x^{th} left hand processor found to have a value greater than the pivot.