EE486: Advanced Computer Arithmetic
Homework #3 Solutions (35 pts + 5 pts for the extra problem)

(15pts) Problem 1 - Adder Design and Performance (n=32, r=3)

(5pts) a) Conditional Sum Adder (with 3-bit slice)

$C_3 = C_{n3} + C_{e3}$

$C_6 = C_{n6} + C_{e6}C_{n3} + C_{e6}C_{e3}C_0$

...

$C_{30} = C_{n30} + C_{e30}C_{n27} + C_{e30}C_{e27}C_{n24} + ... + C_{e30}C_{e27}C_{e24}C_{e21}C_{e18}C_{e15}C_{e12}C_{e9}C_{e6}C_{e3}C_0$

Although there are many terms in C_{30} and a simple AND-OR tree would give 3 levels for the ANDing and 3 levels for the ORing but since some of the terms are simple they can be combined in the OR tree earlier and the delay for the whole thing is 5 gate delays instead of 6 gate delays. Look at the discussion of preferred sites in the canonic adder section in the book.

3 gate delays to calculate C_{n3} and C_{e3} (1 for G, P and T, 1 for AND, 1 for OR)

5 gate delays to calculate C_{30}, C_{27}, C_{24}, ... C_3

2 gate delays to calculate C_{32}, C_{31}, ...

$Total\ Gate\ Delay = 10$

(5pts) b) Single Level Carry Skip Adder (fixed block size = r-1)

The worst path is to ripple through the first group (2 gates per bit for r-1 bits), to ripple through the most significant group (again 2 gates per bit for r-1 bits) and to skip for the $\lceil n/(r-1) \rceil - 2$ groups that are between them.

$$\begin{aligned} Total\ Gate\ Delay &= 2(r-1) + 2(r-1) + 2(\lceil n/(r-1) \rceil - 2) \\ &= 4(r-2) + 2\lceil n/(r-1) \rceil = 4 + 32 = 36 \end{aligned}$$

(5pts) c) Carry Look Ahead (CLA)

Generate p and g 1 *gate delay*
Generate P' and G' +2 *gate delay*, across 3 bits
Generate P'' and G'' +2 *gate delay*, across 9 bits
Generate P''' and G''' +2 *gate delay*, across 27 bits
Generate C_{27} +1 *gate delay* (P''' is early, see below)
Generate C_{30} +2 *gate delay*
Generate C_{31} +2 *gate delay*
Generate S_{31} +1 *gate delay*

$Total\ Gate\ Delay = 13$

Notice that $G''' = G''_2 + P''_2 G''_1 + P''_2 P''_1 G''_0$, $P''' = P''_2 P''_1 P''_0$ and $C_{27} = G''' + P''' C_0$

Note also that here you can not directly apply the formula from the book giving the delay as $4\lceil \log_r(n) \rceil$. In deriving that formula, it was assumed that you pass through (2(number of CLA levels)-1) = 7 levels. Here we are passing only through 4 levels up the tree to generate C_{27} then 2 down the tree to get C_{31}. We are also saving one gate at C_{27}. The formula thus gives you the upper bound of a CLA adder but you can do better sometimes like here.

(10pts) Problem 3.2 - Adders

(4pts) a) $r = 4$ and dot-or limited to 4

The Ling adder is dot-or limited. We need $2^3 = 8$ fan-in for the dot-or, but we only have 4. Therefore, we can not generate the first level of H_i in only 1 gate delay. Instead we need 2 gate delays.

$$\begin{aligned} t &= \lceil \log_r n/2 \rceil + 1 + 1 \\ &= \lceil \log_4 32 \rceil + 2 \\ &= 5 \end{aligned}$$

For the canonic adder,

$$\text{delay} = 2\lceil \log_r (n-1) \rceil + 2 - \delta$$

Since

$$\lceil \log_4 64 \rceil = 3 > 1$$

but

$$\begin{aligned} n = 64 &\quad ? \quad 2(4)^2 - \lceil \log_4 (64 - 16) \rceil \\ &\quad ? \quad 32 - 3 \\ 64 &\quad > \quad 30 \end{aligned}$$

and thus $\delta = 0$. Therefore,

$$\begin{aligned} t &= 2(3) + 2 \\ &= 8 \end{aligned}$$

(3pts) b) Num of CSA delays to reduce six 32-bit operands

Using a wallace tree, this can be done using 3 levels of CSAs.

(3pts) c) FP Add vs FP Mul

There are several reasons why FP add could take more time than FP mul. The dataflow for FP addition can be very complicated in the worst case: swap operands, subtract exponents, shift smaller operand right, add mantissas, recomplement, left shift, round, final shift. In the worst case, it is possible for this process to contain three full-length carry-propagate additions and two full-length shifts, although this case is never actually seen in practice.

Multiply, on the other hand, is relatively straightforward: generate partial products, reduce partial products in a tree or array, perform final carry-propagate add, round. Thus, depending on the exact implementation of both, it is possible for the delay of FP add to be greater than that of FP mul due to its several shifts and full-length adds.

(5pts) Problem 3.3 - Fast Adder

(2pts) a) Fast Address Generation
One possible solution is to use 12 CSA's for the lower order 12 bits and 12 half adders for the higher order 12 bits. Then use a CPA to add the carry and sum bits together.

Another solution is to use 12 CSA's followed by a CPA for the low order 12 bits while a conditional sum adder is used for the high order 12 bits. The carry out of the low order CPA then selects final output out of the two outcomes from the conditional sum adder.
(3pts) b) Upper bound Check
2's complement the upper bound address and add that to the C and S bits of the first solution above. Use 24 CSA's to reduce the C, S and 2's Complement of the upper bound. Compute the final CPA and check if the result is greater than zero.

(5pts) Problem 3.5 - ROMs

Several different designs were possible. Some used the ROMs in the form of (2,2,2,2,2,6) counters as described in chapter 4 but the better designs in terms of performance used some form of carry-lookahead. The best was a design (by Alex) which had a delay of 4 ROMs with 10 total ROMs and 2 ROM types.

The first type of ROM took 4 bits of each input and a carry in signal. It produced 4 sum bits and the group P and G functions. Those group P and G signals are independent of the carry in signal, hence both memory locations with carry in 0 or 1 produce the same P and G. This fact is important for the delay estimation later on.

The second type of ROM took 4 of the group P and G together with a carry in signal to produce 4 carry out signals.

8 ROMs of the first type are in the first row. The second row has 2 ROMs of the second type. One of those two takes C_0 and the P and G from the 4 low order groups of bits to generate C_4, C_8, C_{12} and C_{16} which are fed back to the ROMs in the first row to get the sum bits. C_{16} is also fed to the second ROM in the second row together with the P and G from the 4 high order groups of bits to generate C_{20}, C_{24}, C_{28} and C_{32}. Those carries are also fed back to the ROMs in the first row to get the sum bits.

The worst case delay is from one of the low order ROMs in the first row (P and G) to the first ROM of the second row (C_{16}) to the second ROM in the second row (any of the carrys) to one of the high order ROMs in the first row (to get the sum bits). Since P and G are independent of the carry signal, the second row of ROMs gets its P and G inputs from the first row of ROMs despite the fact that the carries into that first row are not known till the second row produces them.

(5pts) Extra credit - Adder Design and Performance (n=32, r=3)

Design a fast multiple level carry skip adder with variable block size. (N=48, r=5)

One possible solution of the block sizes is:

Level 0: 1,3,4,4,4,4,4,4,4,4,4,4,3,1
Level 1: 1,4,4,4,1
Level 2: 1,3,1

The total delay depends on the adder design within each level 0 block. For example, a CLA is faster than a carry-ripple adder.

Critical Path (carry generate in bit 1 to sum bit in bit 46):

1) Propagate within level 0 blocks

2) Skip through three level 0 blocks

3) Skip through one level 1 block

4) Skip through three level 0 blocks

5) Propagate within level 0 block