

# CSCI-4320/6360 - Assignment 1: C Program for a Carry-Lookahead Adder

Christopher D. Carothers  
Department of Computer Science  
Rensselaer Polytechnic Institute  
110 8th Street  
Troy, New York U.S.A. 12180-3590

January 12, 2022

**DUE DATE: 11:59 a.m., Monday, January 31st, 2022**

## 1 Overview

You are to construct a C program that specifically “simulates” a 4,096 bit Carry Lookahead Adder using 8 bit blocks. We use the term “simulate” here because unlike in real hardware, your CLA adder will not run faster than the serial ripple-carry adder. Here, you will be doing the steps. However, keep in mind that if each level were done completely in parallel via hardware, then it clearly executes much, much faster.

For a list of bitwise operators and the associated C language syntax, please see: [https://en.wikipedia.org/wiki/Operators\\_in\\_C\\_and\\_C%2B%2B](https://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B). Make sure you used the older ANSI-C operators and not the newer operator forms/syntax (e.g., XOR is  $\wedge$  in C).

Recall, that a CLA adder can be constructed from the follow:

1. Calculate  $g_i$  and  $p_i$  for all 4,096 bits  $i$ .
2. Calculate  $gg_j$  and  $gp_j$  for all 512 groups  $j$  using  $g_i$  and  $p_i$ .
3. Calculate  $sg_k$  and  $sp_k$  for all 64 sections  $k$  using  $gg_j$  and  $gp_j$ .
4. Calculate  $ssg_l$  and  $spl$  for all 8 super sections  $l$  using  $sg_k$  and  $sp_k$ . *Note, it is at this point, we can shift to computing the top-level sectional carries. This is because the number of sections is less than or equal the block size which is 8 bits.*
5. Calculate  $ssc_l$  using  $ssg_l$  and  $spl$  for all  $l$  super sections and 0 for  $ssc_{-1}$ .
6. Calculate  $sc_k$  using  $sg_k$  and  $sp_k$  and correct  $ssc_l, l = k \text{ div } 8$  as super sectional carry-in for all sections  $k$ .
7. Calculate  $gc_j$  using  $gg_j, gp_j$  and correct  $sc_k, k = j \text{ div } 8$  as sectional carry-in for all groups  $j$ .

8. Calculate  $c_i$  using  $g_i, p_i$  and correct  $gc_j, j = i \text{ div } 8$  as group carry-in for all bits  $i$ .
9. Calculate  $sum_i$  using  $a_i \oplus b_i \oplus c_{i-1}$  for all  $i$  where  $\oplus$  is the exclusive-or or XOR operation.

You need to construct a C-program that executes on `submitty.cs.rpi.edu` that reproduces this algorithm. Each step in the above algorithm will be implemented as a separate function. Also, the arrays can be globally/statically allocated in memory. No need for dynamic memory allocation.

**A template is provided that parses the hex input data and outputs the *sum* result in a hex format. Additionally, the functions that perform the  $g_i, p_i$  and  $gg_j$  and  $gp_j$  calculations is provided.**

More specifically, your program will do the following:

1. Read two 1024 digit hex numbers as input. Add a leading hex “0” digit to input.
2. Convert the “hex” (base-16) string to a “binary” (base-2) string. NOTE: That hex[0] is the most significant digit, so when you convert to the “binary” string, you need to make sure that binary[4095] is the most significant digit. This can be done by translating each hex digit into it’s binary format (e.g.,  $B = 1011$ ) and then reversing the whole binary array to make sure that binary[4095] is the most significant digit and in the right order with respect to how the hex digits were translated.
3. Each bit can be a full integer and so you’ll have arrays of unsigned integers for each of the components parts of the adder.
4. Write functions for each step in the above algorithm. You can do it using `for` loops and do not have to perform the equation substitutions by hand as we did in class.
5. A master “cla” routine will run thru all the steps.
6. Your main routine will invoke your input reading function, followed by your master “cla” function and then reconvert the binary sum of the two numbers into hex output.
7. Last, you can check your answer by creating a simple ripple carry adder based on computing the  $c_i = g_i + p_i * c_{i-1}$  for all  $i$  and then computing the final sum, based on  $sum_i = a_i \oplus b_i \oplus c_{i-1}$ .
8. Convert the binary  $sum_i$  array to “hex” and output your result in 1024 digit “hex” format.

## 2 Example Test Case

To read the input data, just use `scanf` function call to obtain the two inputs number from the *Standard Input Stream: stdin*. In a file, place the following two 1024 character lines of input. Note, that you need space for 1025 characters in each input for the NULL termination character.

Line 1 is:

Line 2 is:

The output should be a lead “zero”, then “one” followed by 1023 zeros which is the following:

3

Note, above output line has a leading 0 digit pre-pended.

Your program will be run as follows: `./cla < test0.txt`. Note the input file name will change based on which test is being run from the Submittity rubric.

Please submit your C-code (both `main.h` and `cla.c`) to the `submittty.cs.rpi.edu` grading system. The grading rubric will be available to test your solution.