EECS 151/251A Homework 11

Due Friday, May 3rd, 2024

Introduction

This homework is meant to test your understanding adders, multipliers and shifters. This is the last homework! Rejoice :-)!

Problem 1: CLA Latency

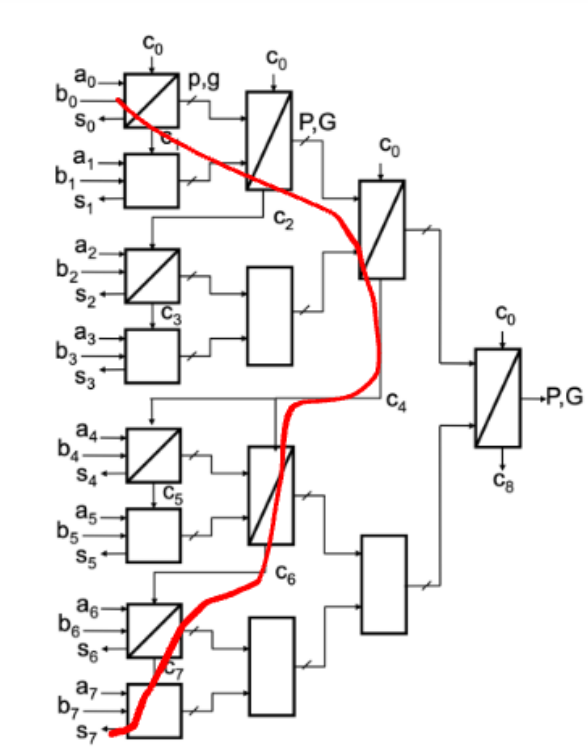
Given the gate delays presented below, trace out and calculate the delay on the critical path for the 8-bit CLA presented in slide 22 from the week 14 Adder lecture.

GATE	Delay(ps)
OR	25
AND	19
XOR	37

Solution:

It is critical to understand *how* a carry-lookahead adder reduces latency of addition. A CLA produces the carry-out from the addition for a bit-pair using an earlier carry-out (benefit only comes from a bit-pair ealier than the immediate preceding bit-pair). This is implemented hierarchical by combining multiple the generate and propagate terms from multiple bit-pairs. By grouping bit pairs of long bit inputs, for example groups 4-bits, the carry-out of a 4-bit group is available fast and allows the subsequent 4-bit group to start its addition.

From the above, The critical path does not deal with carry-out generation because that's the whole point of a carry lookahead adder! The critical path the longest path to the last sum bit. This represents the case where the final carry-in must wait for all previous carry bits. The diagram below shows the critical path.



The idea is that you need the carry bit from the second-to-last bit pair to generate the last

sum bit. Therefore, just trace the path to generate c_7.

- The propagate and generate terms for all bit-pairs are generated in parallel. Furthermore, all hierarchical propagate and generate terms are generated in parallel. Therefore, all blocks in the same column have the same latency.
- c_6 must be available to generate c_7
- c_4 must be available to generate c_6
- To generate c_4 The propagate and generate terms (including their hierarchies) from bit-pairs 0 to 3 must be available
- To generate c_6 The propagate and generate terms (including their hierarchies) from bit-pairs 4 to 5 must be available

From the notes above four latencies need to be known: (1) the latency through 3 levels of hierarchy, (2) the latency through 2 levels of hierarchy, (3) the latency of a single carry-out generation, and (4) the latency for a single sum bit. These latencies represent the latency to generate c_4, the latency to generate c_6, the latency to generate c_7, and the latency to generate s_7. For single bit-pair, the propagate term takes the longest. For hierarchical generate and propagate terms, the generate term dominates $(Delay_{OR} + Delay_{AND})$ if the carry-out is not on the path, and is $(2(Delay_{OR} + Delay_{AND}))$ if the carry-out is on the critical path.

$$\begin{aligned} Latency =& \big(Delay_{XOR} + (Delay_{OR} + Delay_{AND}) + 2(Delay_{OR} + Delay_{AND})\big) + \\ & 2(Delay_{OR} + Delay_{AND}) + 2Delay_{XOR} \\ =& \ 331ps \end{aligned}$$

Problem 2: CSA Latency

Suppose you want to design a fast 64-bit Wallace Tree multiplier using CSAs for the partial product reduction.

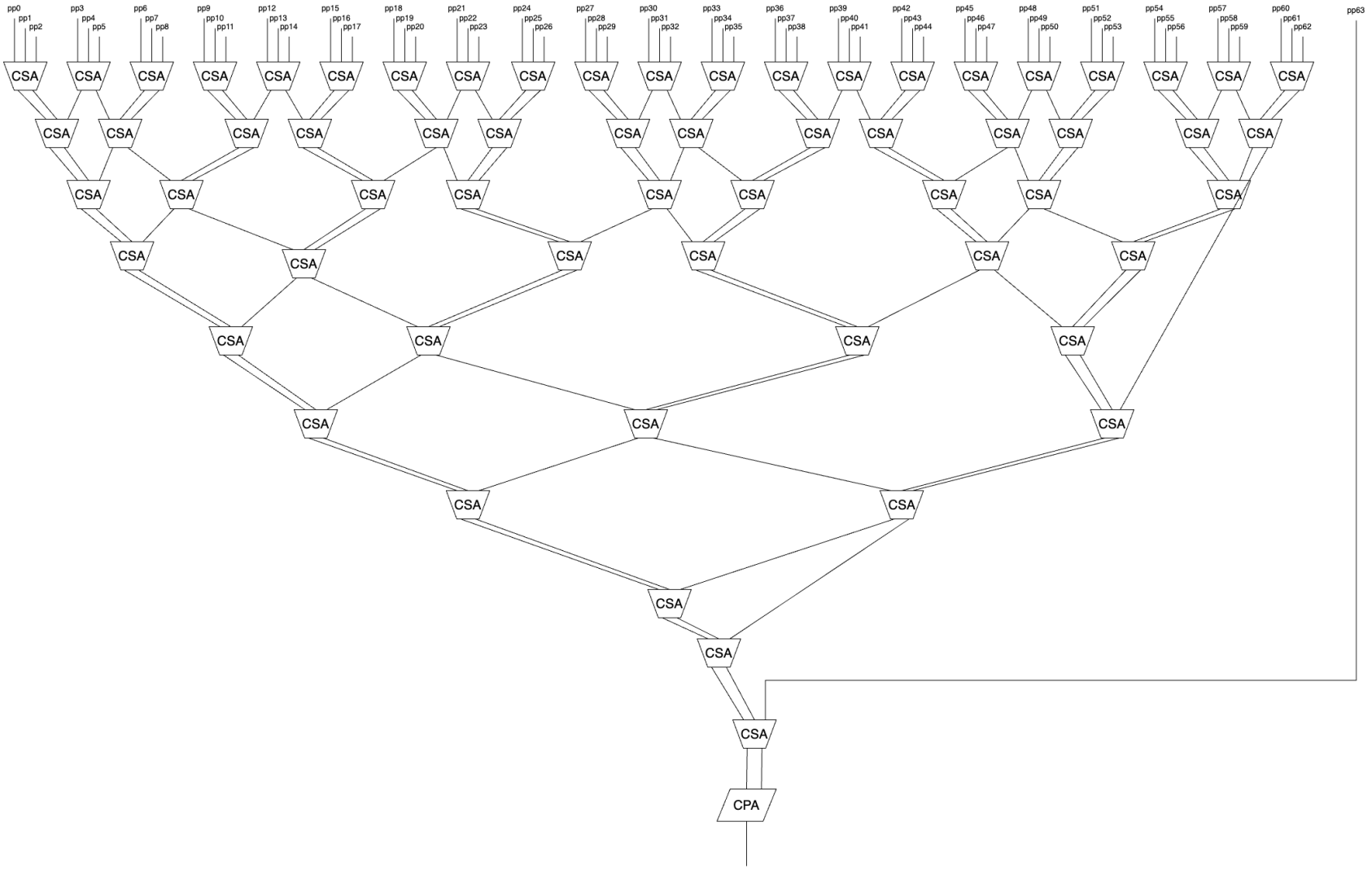
1. How many CSAs are needed?
2. Suppose the delay for a CSA and a CPA, are 1ns and 4ns respectively. What is the minimal delay for product reduction to final result (summing together the partial products)?

Solution:

The following answers based upon the diagram on the following page.

1. 62 CSAs. The number of CSAs could be solved mathematically by repeated dividing the partial products by 3. The diagram is a visual confirmation.
2. There are 10 levels of CSA and one CPA. Therefore, the total latency is $10\tau_{CAS} + \tau_{CPA}$.

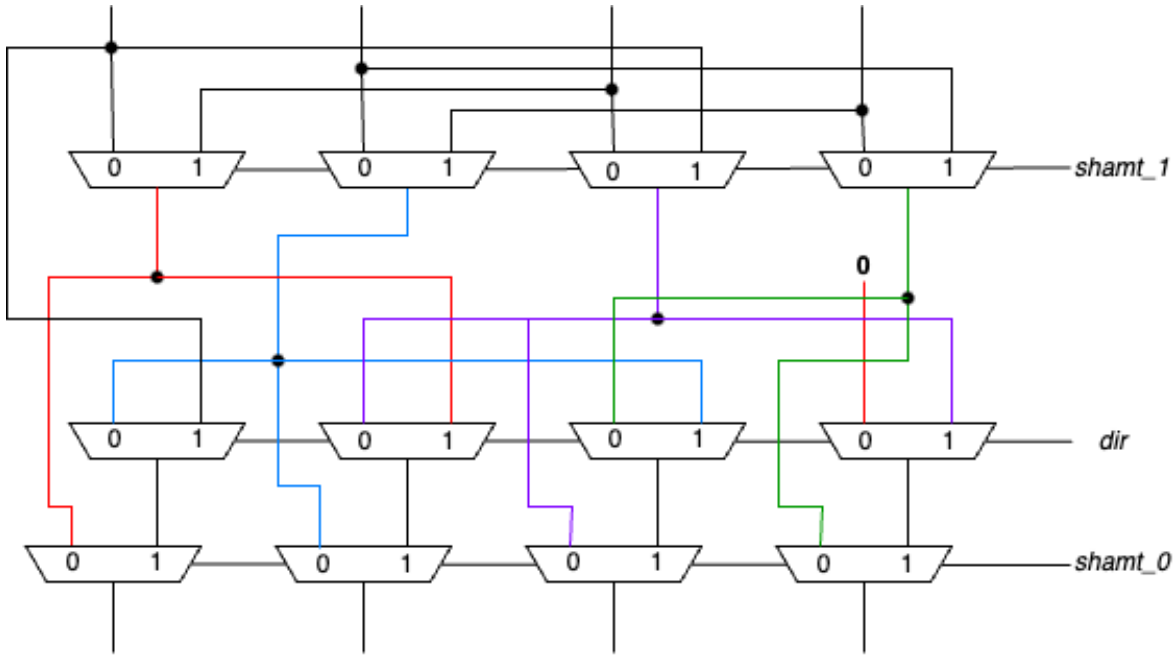
Solution:


pp0 pp1 pp2 pp3 pp4 pp5 pp6 pp7 pp8 pp9 pp10 pp11 pp12 pp13 pp14 pp15 pp16 pp17 pp18 pp19 pp20 pp21 pp22 pp23 pp24 pp25 pp26 pp27 pp28 pp29 pp30 pp31 pp32 pp33 pp34 pp35 pp36 pp37 pp38 pp39 pp40 pp41 pp42 pp43 pp44 pp45 pp46 pp47 pp48 pp49 pp50 pp51 pp52 pp53 pp54 pp55 pp56 pp57 pp58 pp59 pp60 pp61 pp62 pp63
CSA
CPA

Problem 3: Log-Shifter

Design and draw a 4-bit arithmetic log-shifter capable of performing **both** logical left shifts and right shifts. In addition to the shift-amount input (*shamnt*), there is an input named *dir* which is set to 1 for right shift and 0 for left shift.

Solution:



Problem 4: Practice Base 2 Multiplication

Compute the products for the following base 10 numbers as 4-bit 2's-complement binary numbers: (a) -7 x 3, (b) 7 x -3, and (c) -7 x -3. **Show your work for each step in the boxes below.**

a	b	c

Solution:

a

```
    11111001
  + 00000011
  ----------
    11111001
   11111001
  00000000
 00000000
00000000
00000000
00000000
----------------
1111111111101011
```

b

```
    00000111
  + 11111101
  ----------
    00000111
   00000000
  00000111
 00000111
00000111
00000111
00000111
----------------
1111111111101011
```

c

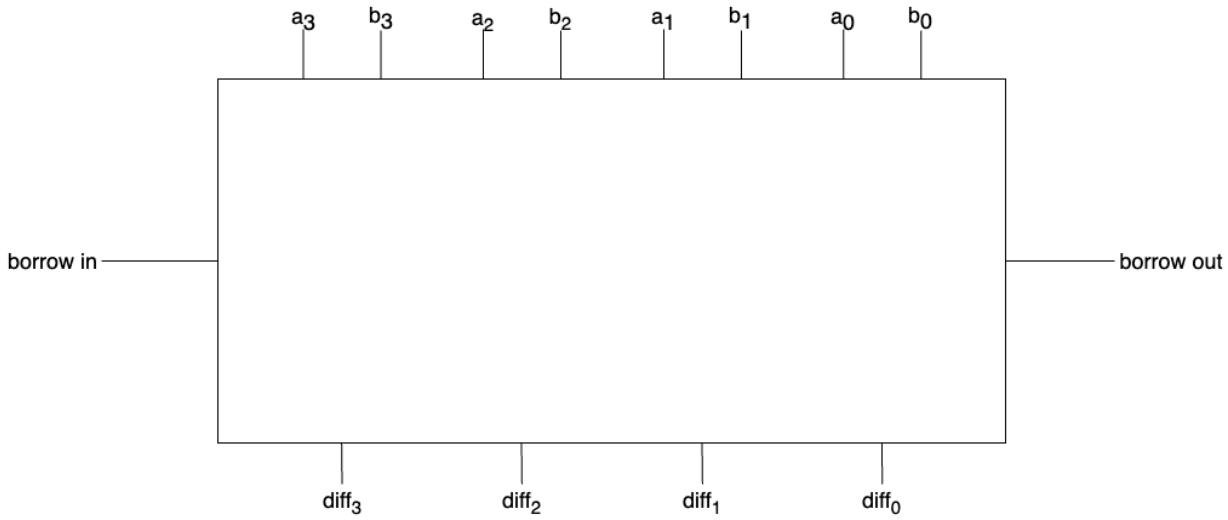
```
    11111001
  + 11111101
  ----------
    11111001
   00000000
  11111001
 11111001
11111001
11111001
11111001
----------------
0000000000010101
```

Problem 5: Ripple Borrow Subtractor

In lecture, we discussed the design for a ripple carry adder. Your task is to design a ripple borrow subtractor which has the same structure of the adder except it performs subtraction.

(a) Draw a single stage at the logic-gate level.

(b) Draw a diagram for a ripple borrow subtractor for 4-bit 2's complement inputs at the stage level.



Solution:

The subtractor circuit is fairly similar to the adder circuit. However, it is to build the subtractor circuit from scratch rather than translating the adder circuit into a subtractor circuit (plus you gain intuition).

The rules for binary subtraction are as such:

- 0 – 0 = 0
- 0 – 1 = 1

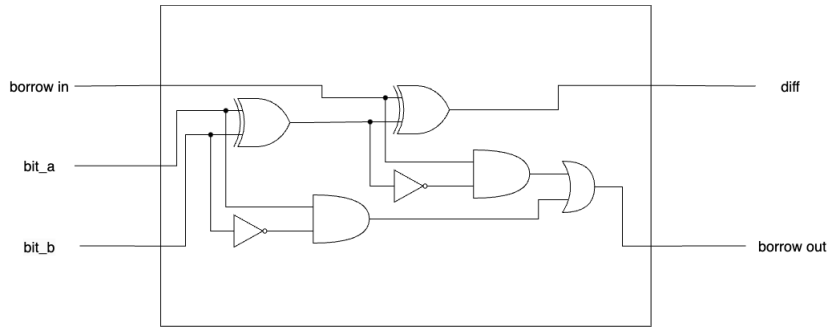
- $1 - 0 = 1$
- $1 - 1 = 0$

In subtraction, you have a borrow bit (which is similar to subtraction in base 10). A pair of bits can only lend a borrow bit in the case of $1 - 0$. Furthermore, a borrow bit is required only for the case $0 - 1$ (this is important later).

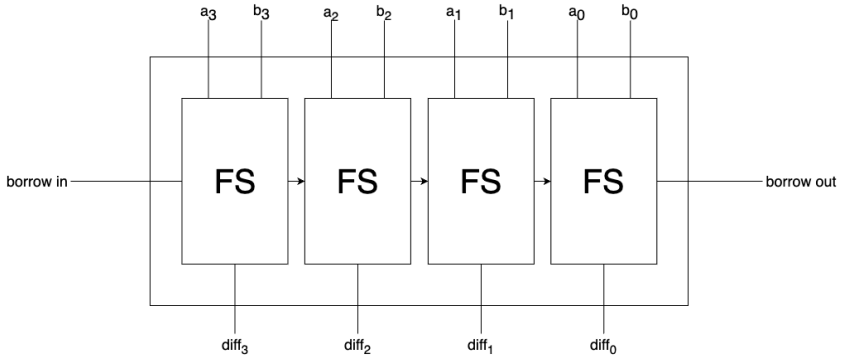
From the above the truth table for a 2-bit subtractor, a half-subtractor, is:

A	B	DIFF	BORROW
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	0

Similar to how a full-adder is composed of two half-adder, a full-subtractor (3-bit subtractor) is composed of two half-subtractors. The circuit for the half-subtractor is not shown since it can be easily inferred from the table table.



At this point combining full subtractors is exactly the same as full adders. The important aspect to take note is the bit being transferred is the borrow out which flows from higher order bits to lower order bits.



Problem 6: Cross-bar Switch

Based on the cross-bar switch presented in lecture:

For the scenarios below, provide the hexadecimal value to represent the decoder input for a 4-bit inputs. For example, given an implementation for a 16-bit input the first 4-bits of the hex value would control the decoder of the output's LSB, and the next consecutive 4-bits control the decoder for the output's second to last bit.

1. Bit-Reversal (ex. $a_0a_1a_2a_3 \rightarrow a_3a_2a_1a_0$)
2. Pairwise Reversal (ex. $a_0a_1a_2a_3 \rightarrow a_1a_0a_3a_2$)
3. Random Permutation: $a_0a_1a_2a_3 \rightarrow a_1a_3a_2a_0$

Solution:

Note that the numbering of the bits is reverse relative to the slide. This was intentional. Final exams often have similar problems with slight modifications. Do not solely rely on your cheat sheet!

On the other hand, the final exams often make problems easier to solve if it is understood correctly. For example, the backwards numbering of bits allows the decoder inputs to simply be the numbering of the bit :-).

1. Bit-Reversal: 8'he4 (as decimal(LSb → MSb): 3|2|1|0)
2. Pairwise Reversal: 8'h4e (as decimal(LSb → MSb): 1|0|3|2)
3. Random Permutation: 8'h78 (as decimal(LSb → MSb): 1|3|2|0)

Problem 7: 64-bit Adder

An application you are designing hardware for requires a 64-bit adder. You could: (1) design and build a 64-bit adder, or (2) construct a 64-bit using very fast 32-bit adders you've already designed and built. You decide to save time and construct a 64-bit adder using 32-bit adders.

Your 32-bit adder has the following specifications:

Inputs: 1-bit $carry_{in}$, two 32-bit operands

Outputs: 1-bit $carry_{out}$, a 33-bit sum

Latency: 64ps from $carry_{in}$ to $carry_{out}$

Perform an analysis for two design points: (1) carry-select architecture, and (2) carry-lookahead architecture. Design each architecture and compare the total delay and HW cost (number of each adder, gates, multiplexors, etc) between the architectures.

Solution:

Carry-Select
A carry select architecture requires three 32-bit adders: one adder for the 32 LSbs, one for the upper 32 MSbs with $carry_{in}$ = 0, one for the upper 32 MSbs with $carry_{in}$ = 1. Furthermore, you would need a mux to selector which of the two upper 32-bit partial sum to use.

Latency: The latency would be 64ps + the delay of a mux. Since all the adders execute in parallel, the final sum is known once the $carry_{out}$ from the 32 LSb adder is known.

Hardware Cost: The hardware cost is another adder and a mux. The cost of the mux is negligible. The cost of a third adder is higher. Again, the third adder is necessary to allow parallel computation of the upper 32-bits which is the reason why latency decreases.

Carry-Lookahead
The benefit of the carry lookahead adder is that the $carry_{out}$ bit is determined without rippling through full adders. Additional logic, external to the adders, can be used to generate the $carry_{out}$ of the 32 LSbs to start the addition of the upper 32 MSbs earlier.

Latency: The latency is 6τ + 64ps, where τ is the latency of the propagate and generate term production. If a tree structure is implement to produce the $carry_{out}$, then there will be $log_2(32)$ = 5 levels of hierarchy. The delay of the generate signal dominates because it requires two cascaded gates. Therefore, the total delay through the hierarchy to produce the final propagate and generate is 5τ, where τ is the delay to produce a generate term. The very last level of the hierarchy contains one unit which combines all previous propagate and generate into P and G respectively then uses these terms to produce a carry out ($C + c_{in}P$)). Note the delay to produce the $carry_{out}$ is the same as the generate delay. Therefore, the total latency is 6τ + 64ps.

Hardware Cost: The total hardware cost is two 32-bit adders and the logic carry lookahead logic. Referencing Problem 1, half the units on each hierarchy level generate a carry out. However, this carry out is used to calculate the sum bits only which we do not care about because the 32-bit adder is doing this for us. The carry lookahead logic only needs to produce the final carry out. Therefore, the addtional cost is $\sum_{i=0}^{5} 2^i$ 3-input logic gate (2 AND gates and 1 OR gate).

Problem 8: Baugh-Wooley Annotation

We would like to prove that the Baugh-Wooley approach to signed multiplication is correct. Annotate the corresponding circuit on slide 22 from the "Multipliers & Shifters" lecture for X = -3, Y = -5, by writing down the inputs and outputs for each FA and HA blocks. Verify that the results match what you expect as a result (Z = 15).

A labelled diagram of the multiplier is provided below. The naming convention is $ELEMENT\_[ROW][INDEX]$ ($ELEMENT$ can be HA, FA, NAND or AND, ROW is zero-indexed, and $INDEX$ is zero-indexed and is agnostic to the $ELEMENT$). Fill in the tables below for the output of each element. **Only the values in the table will be graded.** However, labelling may be instructive and easier to visualize.

NAND_03	AND_02	AND_01	AND_00

FA_03	FA_02	FA_01	HA_00

NAND_13	AND_12	AND_11	AND_10

FA_13	FA_12	FA_11	HA_10

NAND_23	AND_22	AND_21	AND_20

HA_24	FA_23	FA_22	FA_21	HA_20

AND_33	NAND_32	NAND_31	NAND_30

Solution:

Note that that adders have two outputs: the sum and carry. The entries for the adders are in the format $carry \mid sum$.

NAND03	AND02	AND01	AND00
0	1	0	1

FA_03	FA_02	FA_01	HA_00
0|1	0|1	0|1	0|1

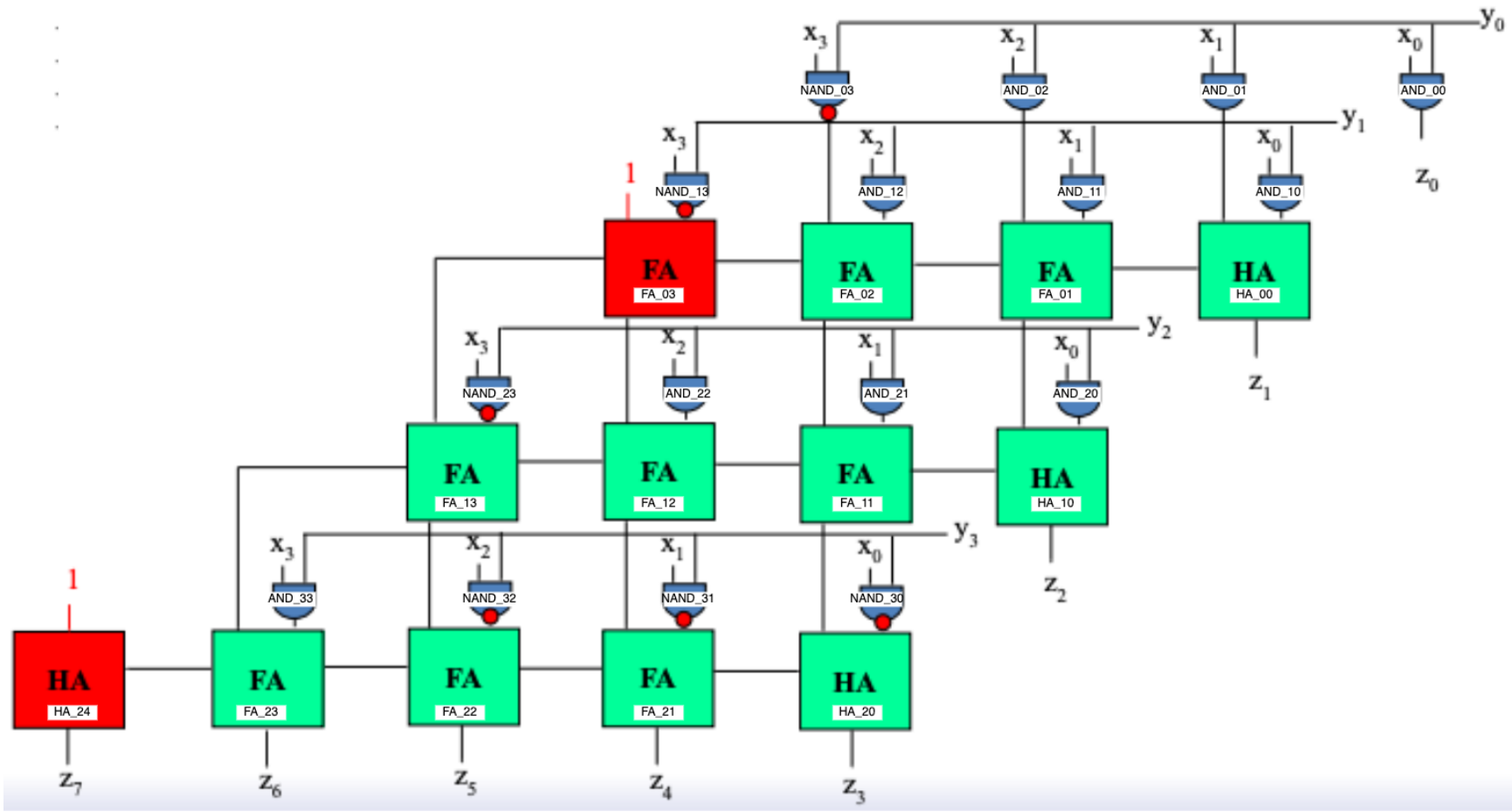
NAND13	AND12	AND11	AND10
0	1	0	1

FA_13	FA_12	FA_11	HA_10
0|1	0|1	0|1	0|1

NAND23	AND22	AND21	AND20
1	0	0	0

HA_24	FA_23	FA_22	FA_21	HA_20
1|0	1|0	1|0	1|0	0|1

AND33	NAND32	NAND31	NAND30
1	0	1	0


y0
x3
x2
x1
x0
NAND_03
AND_02
AND_01
AND_00
z0
y1
1
NAND_13
AND_12
AND_11
AND_10
FA
FA_03
FA
FA_02
FA
FA_01
HA
HA_00
z1
y2
NAND_23
AND_22
AND_21
AND_20
FA
FA_13
FA
FA_12
FA
FA_11
HA
HA_10
z2
y3
1
AND_33
NAND_32
NAND_31
NAND_30
HA
HA_24
FA
FA_23
FA
FA_22
FA
FA_21
HA
HA_20
z7
z6
z5
z4
z3

Problem 9: Constant Multiplication

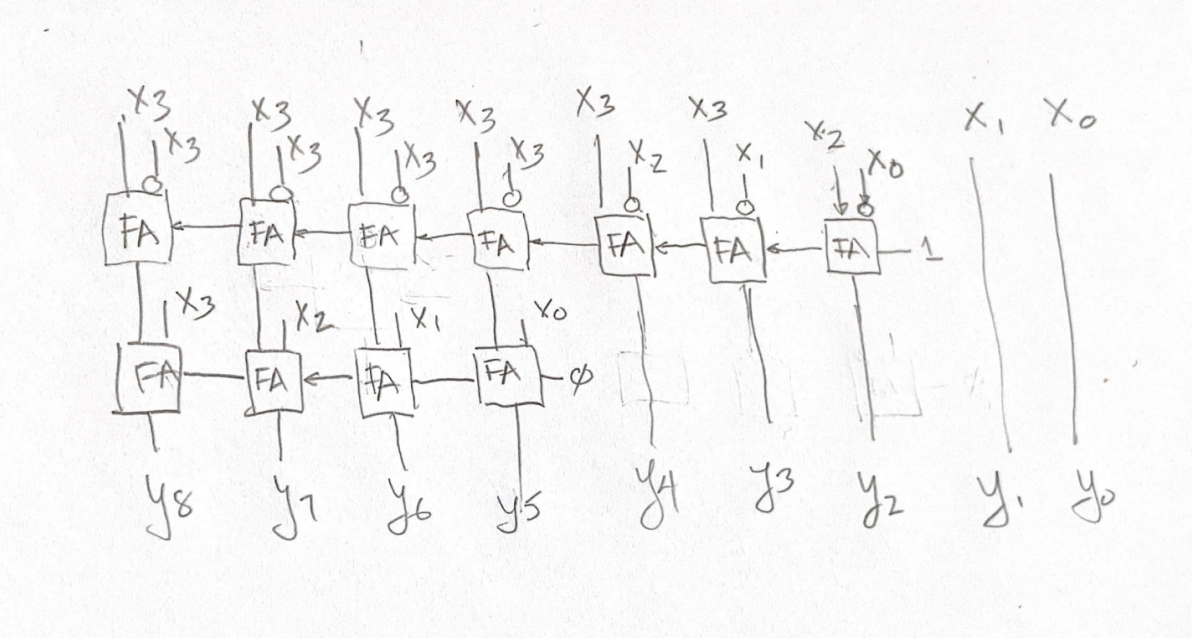
Using the CSD representation, design a multiplier for $C \cdot X$, where C is a constant and X is a signed 4-bit input to the circuit. For your circuit, let C = 29. **Show your circuit to the FA level of detail.**

Solution:

Quickly, let's review what CSD is doing. It reduces that number of bits in a bit string to represent a number. This correlates directly hardware necessary to perform an operation like multiplication. It is based upon the principle that any string of 1's with length greater than 2 can be represented by a bit string with the MSb set with a positive weight, and the LSb set witha negative weight. A simple example is 7 (0111 in binary). Mathemically, 7 = 8-1. 8 is 1000 in binary and 1 is 0001 in binary. Therefore, $100\bar{1}$ represents 7 in CSD encoding. This represents for all numbers regardless of position in bit string i.e. (3, 15, 127, etc).

The binary representation for 29 is 00011101. In CSD encoding, 29 is $0010\bar{0}\bar{1}01$. See the figure below which visually shows the alignment of the partial products. Therefore, we only need three full subtractors versus 8 full adders if CSD encoding was not used.

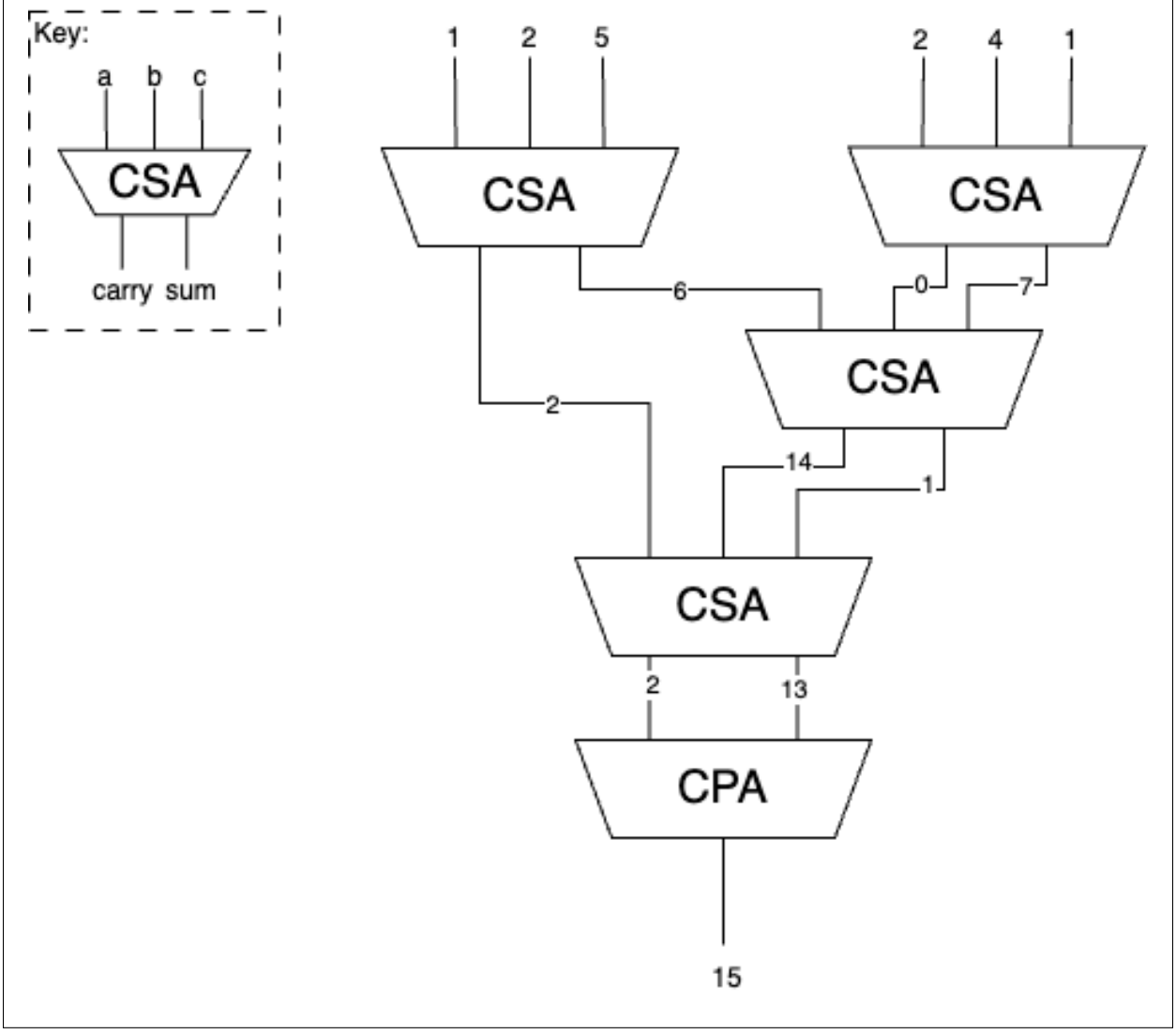
Since X is signed, it must be signed extended the full bit width of the product. A naive solution would be to negate $4 * X$ and sum all the partial products. However, since we already need to perform an addition we can cleverly combine 2's complement step with the addition by negating the bits of $4 * X$ and let the very first carry in for the adder be 1.



Problem 10: CSA + CPA Annotation

Using CSAs and a final CPA step, design and draw a circuit to calculate the following: $1 + 2 + 5 + 2 + 4 + 1$. **You must annotate your circuit showing all intermediate values.**

Solution:



Problem 11: Parallel Prefix Adder

Draw out the circuit diagram for a 7-bit parallel prefix adder. Drawn in the style of the example on slide 26 from the Adder lecture.

Solution: