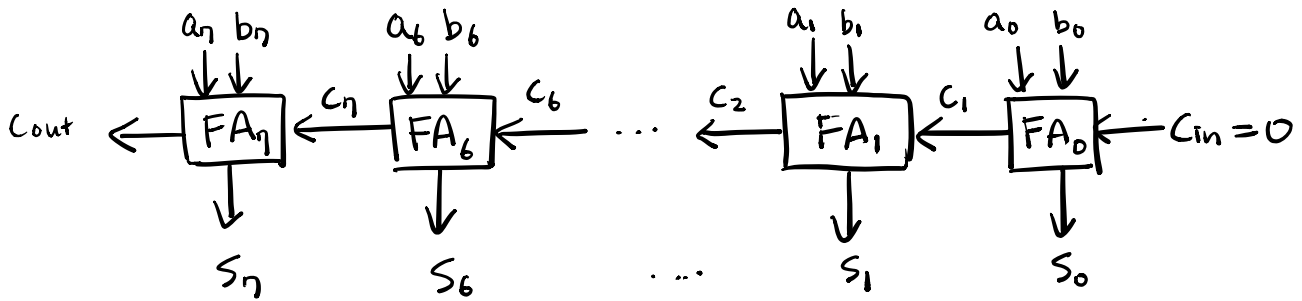


#1. 8-bit ripple carry adder

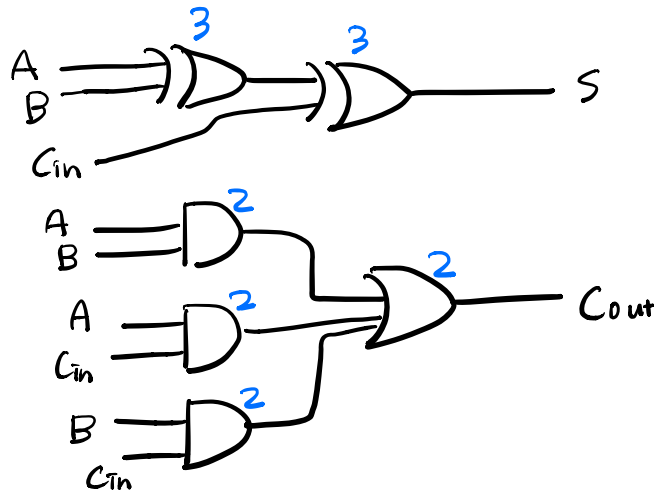
(FA:
Full Adder)



worst case path in ripple carry adder:

Inputs to FA_0 propagating all the way to FA_7
(as the carry bit)

full adder:



(written in blue
are delays for
each gate)

In each full adder, the worst case path goes through the AND gate and the OR gate (the circuit that computes the carry out bit). The delay in this path is

$$2 + 2 = \underline{4}$$

But in FA_7 (the full-adder that computes the most significant bit), the worst case path to the sum output (S_7) will go through only an XOR gate.

Since there are 8 full-adders in an 8-bit ripple carry adder, the worst case delay for any sum output would be

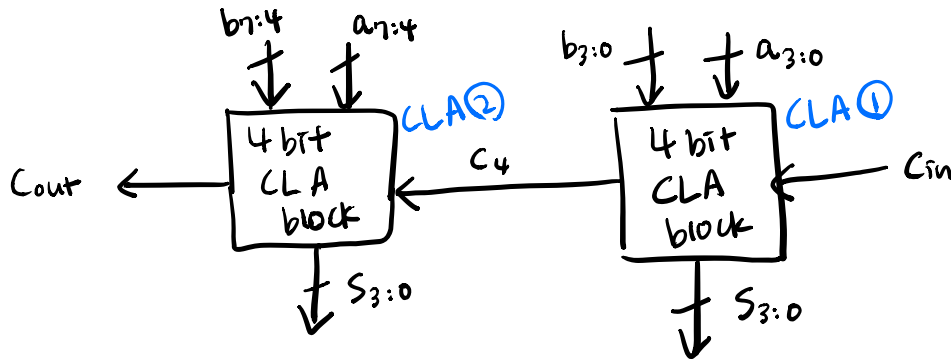
$$4 \times 7 + 3 = 31$$

path to carry-out

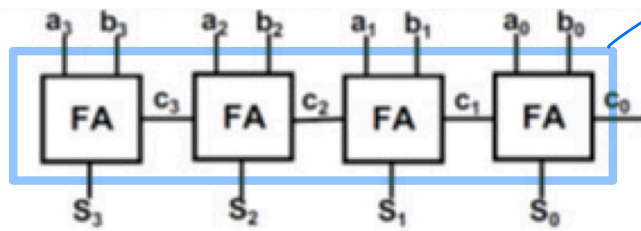
C_{in} XORed to compute Sum

The worst-case delay for the carry out would be $4 \times 8 = 32$

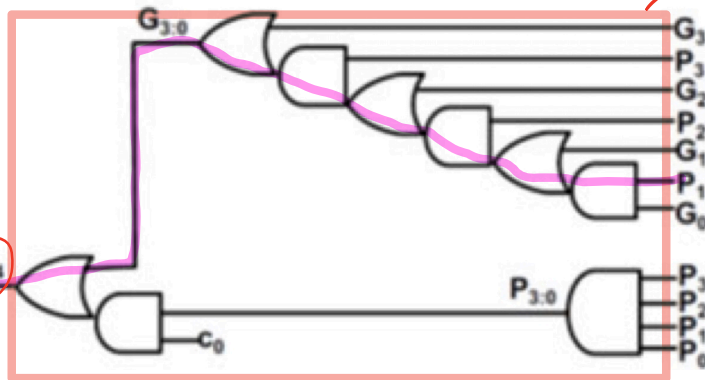
#2. 8-bit carry lookahead adder w/ 4-bit ripple-carry



4-bit CLA block:





4-bit ripple carry adder



carry lookahead logic
(computes C_4 faster, so that the next CLA block can generate its output faster)

worst-case delay for carry bit in 4-bit CLA block

G_i  $G_i = a_i b_i$ (adding bits a, b generate a new carry)

P_i  $P_i = a_i \oplus b_i$ (adding bits a, b will propagate the carry-in bit the addition receives)

(In the Figure above that connects the two 4-bit CLA blocks. I denoted the CLA block for least significant 4 bits as CLA ①, and the CLA block for most significant 4 bits as CLA ②.)

$$\begin{aligned} & \text{worst-case delay for } \underline{\text{carry}} \text{ bit in 8-bit CLA} \\ &= (\text{worst-case delay for carry bit in CLA ①}) \\ & \quad + (t_{\text{AND}} + t_{\text{OR}} \text{ in CLA ②}) \\ &= (t_{\text{XOR}} + 3 \cdot t_{\text{AND}} + 4 \cdot t_{\text{OR}}) + (t_{\text{AND}} + t_{\text{OR}}) \\ &= 3 + 6 + 8 + 2 + 2 = \textcircled{21} \end{aligned}$$

$$\begin{aligned} & \text{worst-case delay for } \underline{\text{sum}} \text{ bit in 8-bit CLA} \\ &= (\text{worst-case delay for carry-bit in CLA ①}) \\ & \quad + (\text{delay from } C_0 \text{ to } S_3 \text{ in CLA ②}) \\ &= (t_{\text{XOR}} + 3 \cdot t_{\text{AND}} + 4 \cdot t_{\text{OR}}) + (3 \cdot (t_{\text{AND}} + t_{\text{OR}}) + t_{\text{XOR}}) \\ &= (3 + 6 + 8) + (12 + 3) = \textcircled{32} \end{aligned}$$

#3. The ripple-carry adder will be advantageous when

- ① the number of bits being added together is so small that there is no advantage in the extra logic to compute carry-out bits faster
- ② the goal is to minimize chip area (and don't care about delay)

#4. [4x4 multiplier using ROM]

The address input would be 8-bits. with the 4 most significant bits being the multiplicand and the 4 least significant bits being the multiplier. (could be vice versa)

There will be $2^8 = 256$ words in this ROM.

The word activated by the input address would store the multiplication result.

Each word would be 8 bits long.

The size of the ROM array would be

$$256 \times 8 = \underline{2048 \text{ bits}}$$

#5. [4x4 carry-save multiplier]

Let the two values being multiplied be denoted as

$$A = a_3 a_2 a_1 a_0 \text{ (multiplicand) and } B = b_3 b_2 b_1 b_0 \text{ (multiplier)}$$

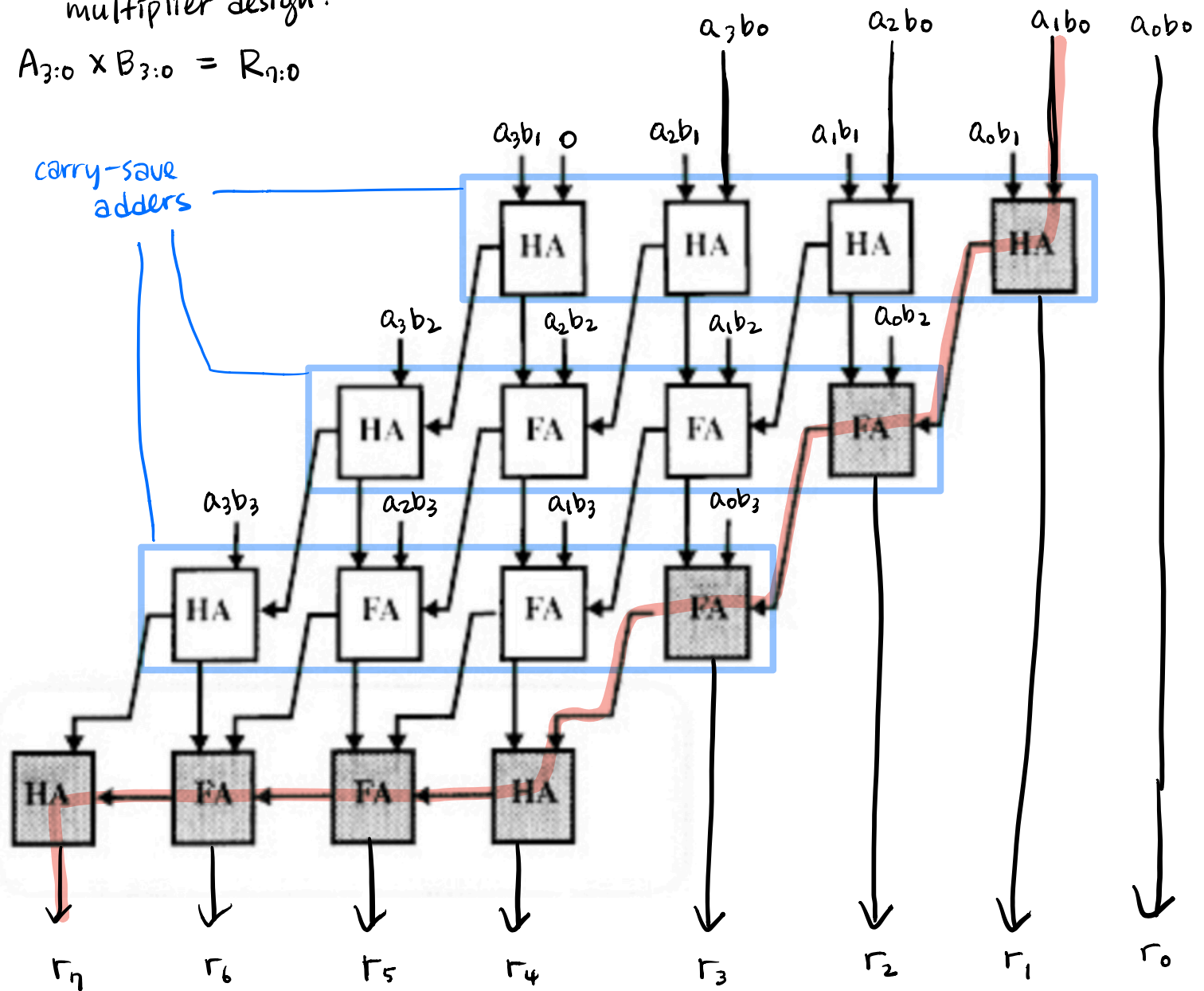
The multiplication result can be thought of as:

$$\begin{array}{r} b_0 \times a_3 a_2 a_1 a_0 \\ + b_1 \times a_3 a_2 a_1 a_0 0 \\ + b_2 \times a_3 a_2 a_1 a_0 0 0 \\ + b_3 \times a_3 a_2 a_1 a_0 0 0 0 \end{array}$$

instead of using a ripple carry adder for this sum, (or other fast adder) the carry-save multiplier uses the carry-save adder

Carry-save multiplier design:

$$A_{3:0} \times B_{3:0} = R_{7:0}$$



The critical path is through the shaded half adders and full adders. (propagated carry-outs that affect r_7).

The worst case delay in this would be:

$$\begin{aligned}
 & \text{(delay of AND gate)} \\
 & + (\text{delay in generating carry-out in HA}) \times 2 \\
 & + (\text{delay in generating carry-out in FA}) \times 4 \\
 & + (\text{delay in generating sum in HA}) \\
 & = t_G + 2t_G + 4 \times 2t_G + t_G = \boxed{12 t_G}
 \end{aligned}$$

#6. As the number of bits in the multiplicand and multiplier increase, the critical path delay in the carry-save multiplier will increase.

If the decoder logic for ROM addressing has smaller worst-case delay, the ROM-based multiplier could be faster.

However, the ROM-based multiplier will require much more transistors to implement (proportional to array size).

#7. (a) 1011_2

$$\text{unsigned: } 2^3 + 2^1 + 2^0 = 11$$

$$1\text{'s complement: } -1 \times (100)_2 = -4$$

$$2\text{'s complement: } -2^3 + 2^1 + 2^0 = -5$$

(b) 1100_2

$$\text{unsigned: } 2^3 + 2^2 = 12$$

$$1\text{'s complement: } -1 \times (011)_2 = -3$$

$$2\text{'s complement: } -2^3 + 2^2 = -4$$

(c) 10100100_2

$$\text{unsigned: } 2^7 + 2^5 + 2^2 = 164$$

$$1\text{'s complement: } -1 \times (1011011)_2 = -91$$

$$2\text{'s complement: } -2^7 + 2^5 + 2^2 = -92$$

(d) 10011110_2

$$\text{unsigned: } 2^7 + 2^4 + 2^3 + 2^2 + 2^1 = 158$$

$$1\text{'s complement: } -1 \times (1100001)_2 = -97$$

$$2\text{'s complement: } -2^7 + 2^4 + 2^3 + 2^2 + 2^1 = -98$$

(e) 01001001_2

$$\text{unsigned: } 2^6 + 2^3 + 2^0 = 173$$

1's complement, 2's complement are the same as unsigned for positive numbers.