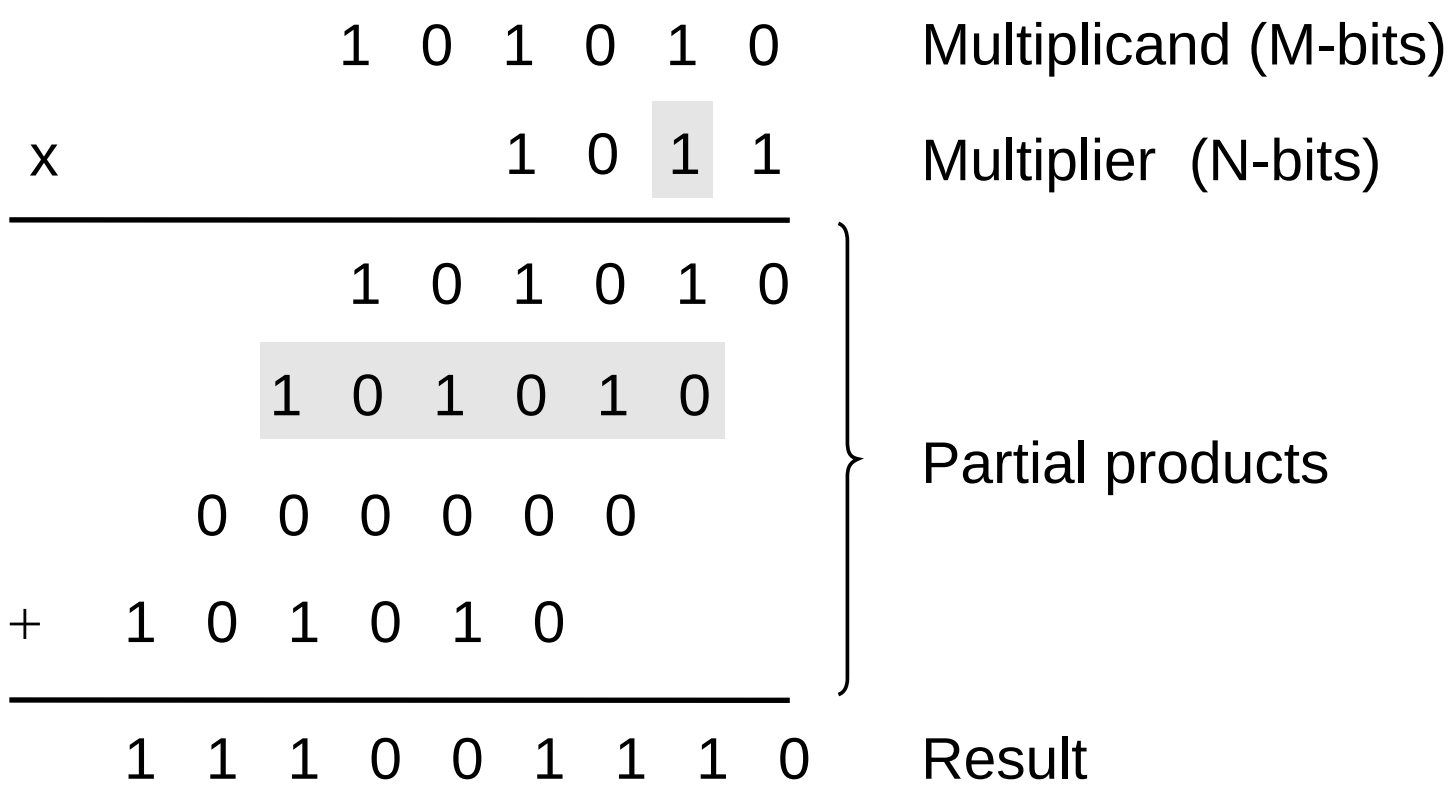
EECS 427
Lecture 12: Multipliers
Reading: 11.4

Lecture Overview

- Multiplier implementations
- Multipliers are vital in digital signal processing and standard desktop processors
- They are speed limiting – complex operations
- On modern CPUs they are quite fast
  - (at cost of large size) generally largest single logical element (memories bigger)

Binary Multiplication

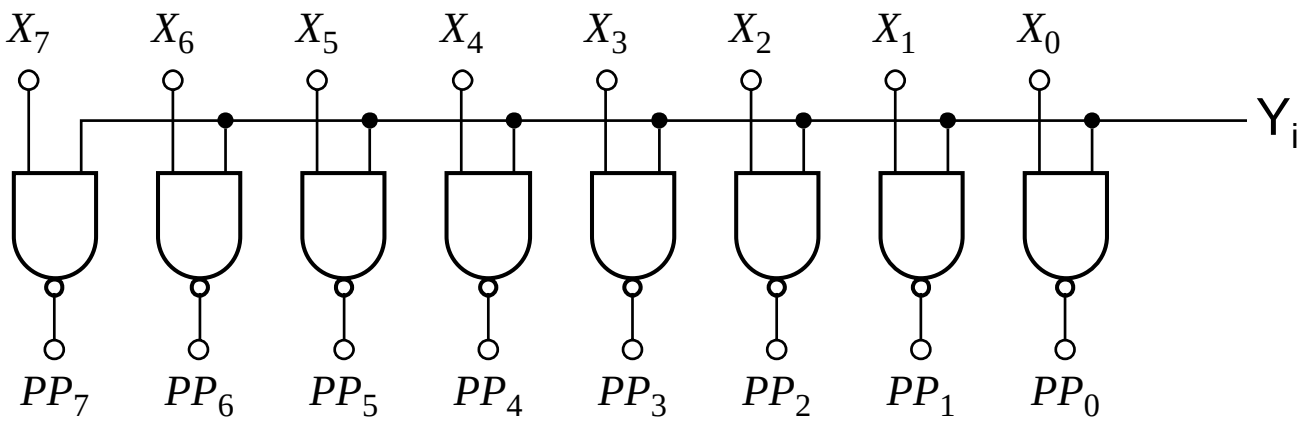


Key points

- NxM multiplication fits in N+M bits
- 2s complement multiplication is more difficult: Unlike addition, must think about sign in implementation, commonly:
  - Convert to **+** numbers and keep track of sign
  - Use Booth's algorithm
- Major steps are:
  1) Partial product generation
  2) Partial product accumulation
  3) Final addition (fast carry lookahead techniques)

Generating Partial Products

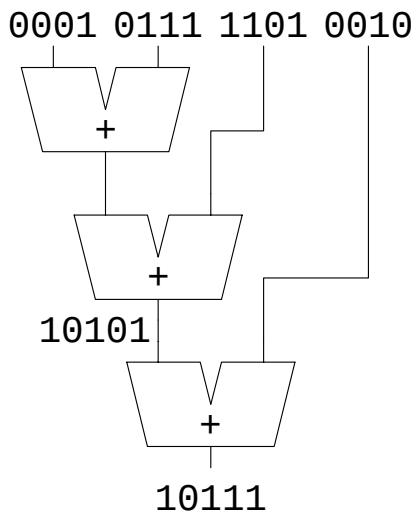
- All partial products: AND



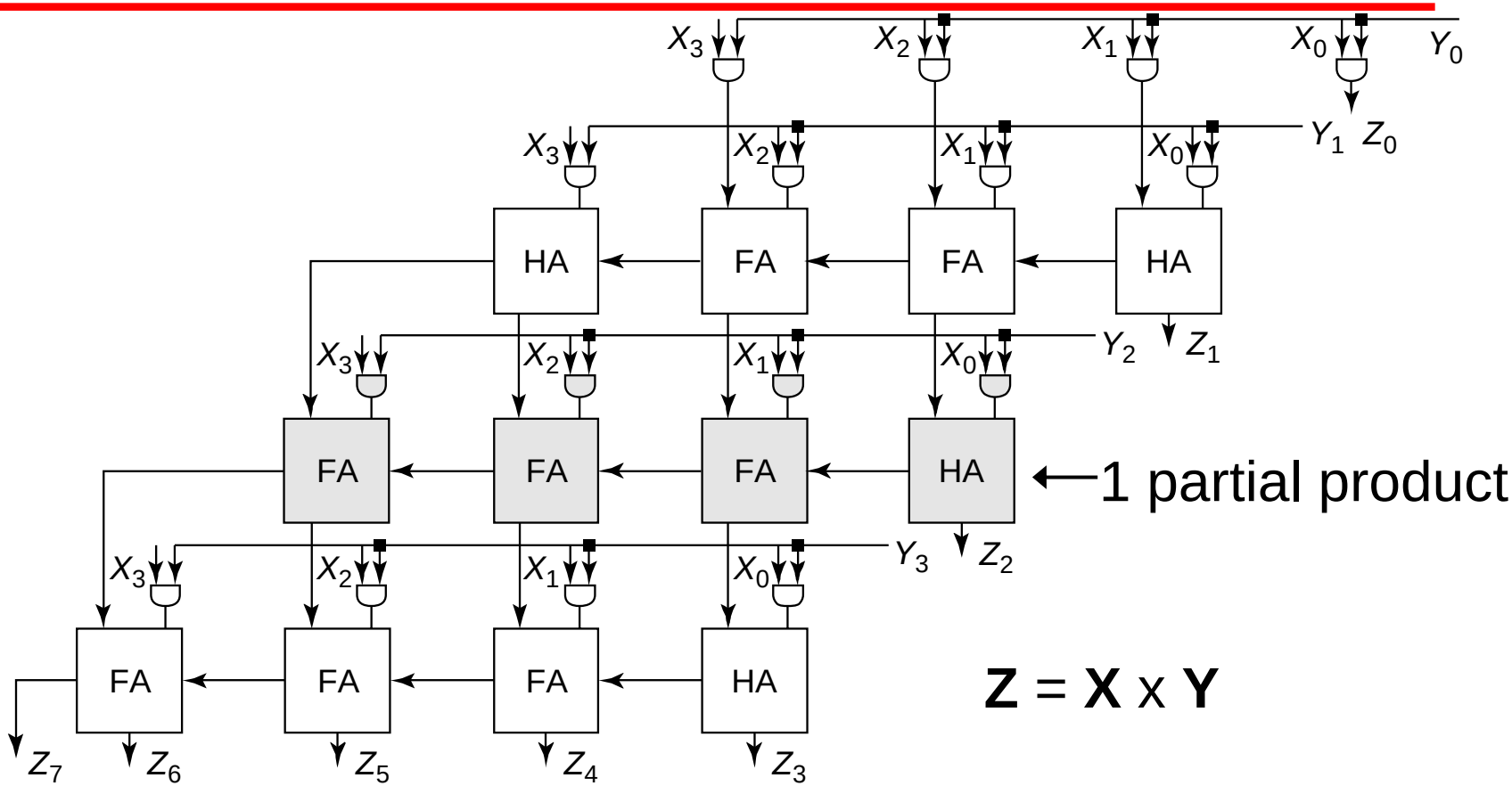
- Booth's recoding – reduction of partial product count (more later)

Multi-Input Adder

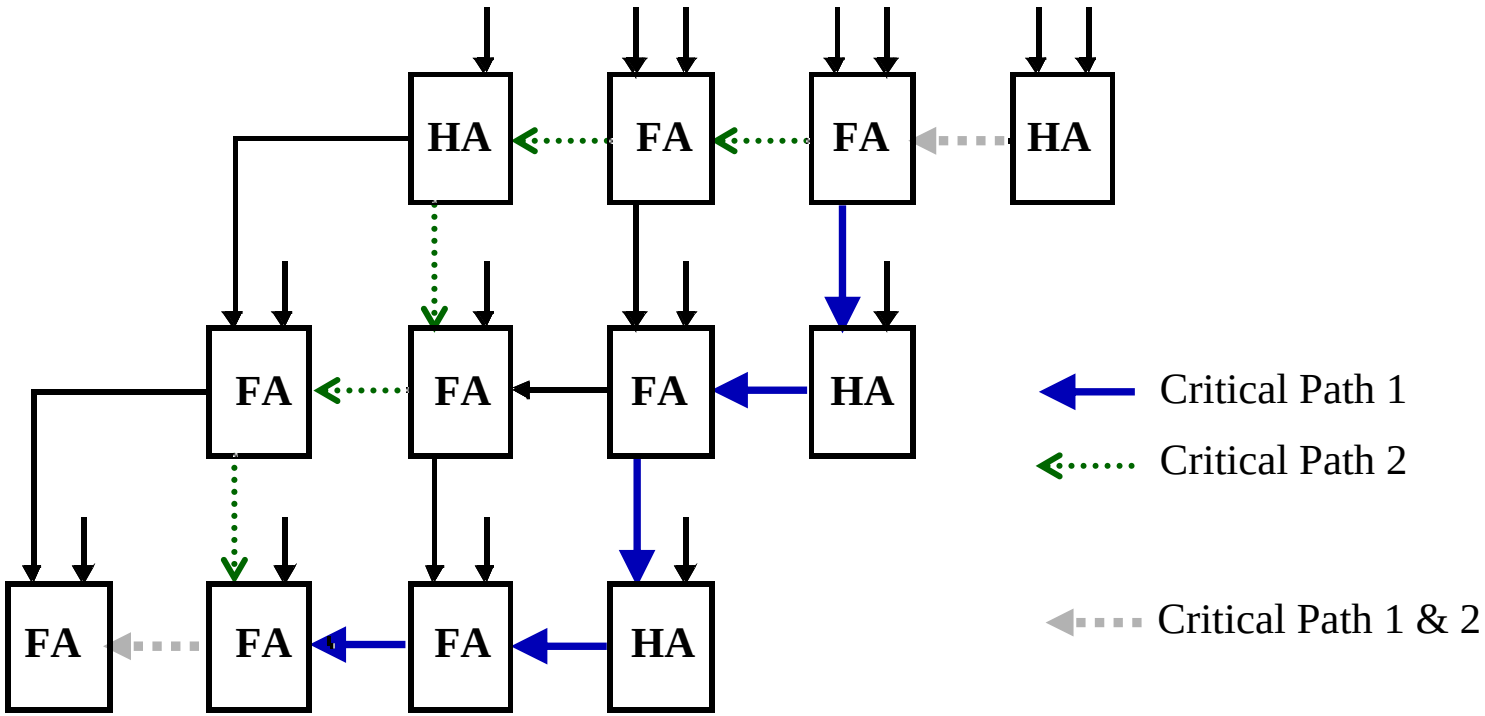
- Suppose we want to add k N-bit words
  - Ex: 0001 + 0111 + 1101 + 0010 = 10111
- Straightforward solution: k-1 N-input CPAs
  - Large and slow



The Array Multiplier



MxN Array Multiplier — Critical Path

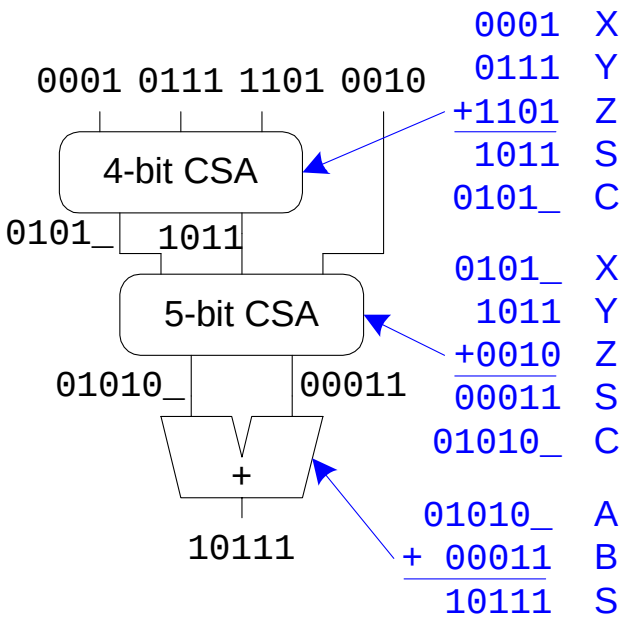


$$t_{mult} = [(M-1)+(N-2)]t_{carry} + (N-1)\,t_{sum} + t_{and}$$

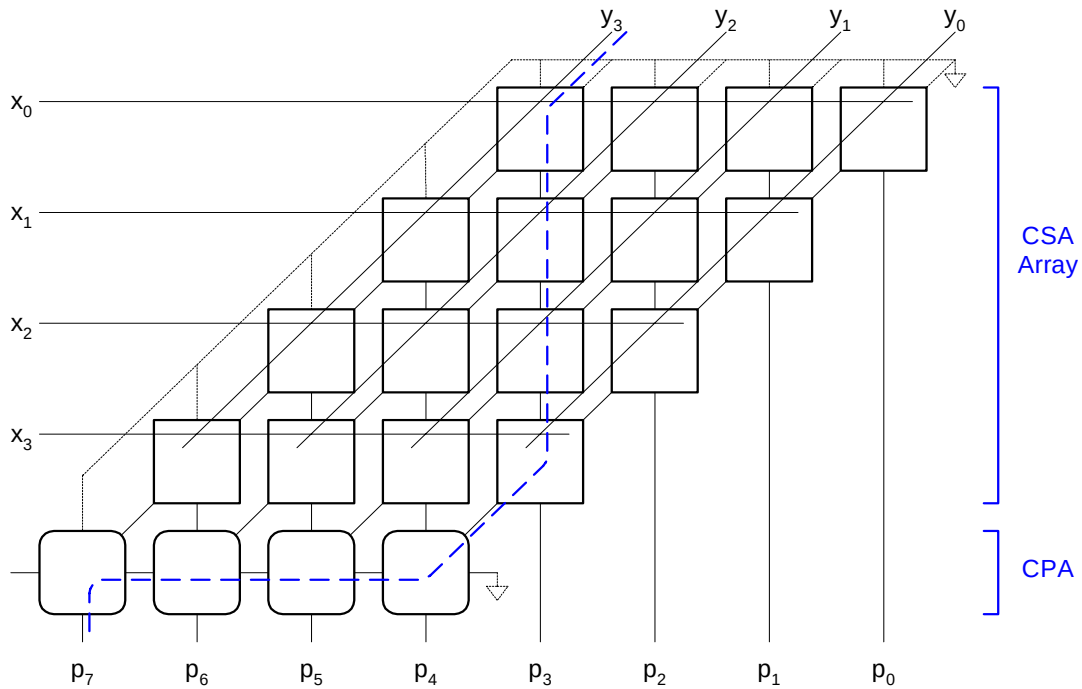
Both carry and sum delays important: T-gate adder cell…

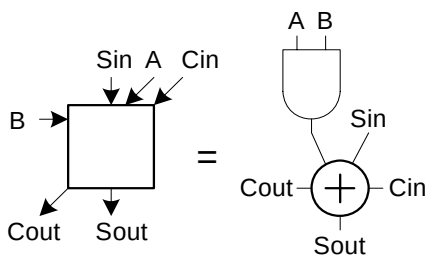
Carry-Save Addition

- Use k-2 stages of CSAs
  - Keep result in carry-save redundant form
- Final CPA computes actual result

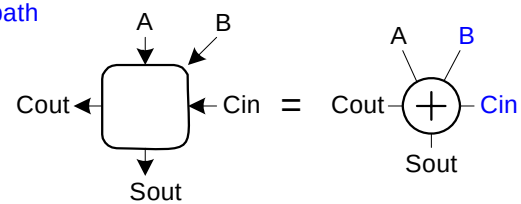


Carry-Save Multiplier





critical path



$$t_{mult} = (N-1)t_{carry} + t_{and} + t_{merge}$$

Tree Multiplier

- Just a carry save adder, but instead of a linear array, use a tree of carry save adders
- Connection diagram is awful
- Strange tree : each node has 3 in and 2 out
- Can get 4->2 (binary) rather than 3->2 via specially designed 5->3 adder (5 bits in, 1 sum bit, 2 carry bits)

More on 4-2 compressor

- Inputs a,b,c,d,cin
- Outputs out0, out1, cout
- Cout must be high if 3+ of a,b,c,d are high and must be low if 3+ are low but can go either way if 2 are high.
- Cout could thus be majority of a,b,c (cout of full adder) then out0, out1 outputs of full adder of cin, d, sum from first full adder
- Essentially just 2 3-2 compressors in series but can be sized for even delay
- Other logic is also possible e.g. cout = (a+b)(c+d)

Booth Recoding

- To implement 2s complement multiplication, modified Booth recoding is typically used
- Idea: Recode the multiplier value in a higher radix in order to reduce the # of partial products
- Ex:

$$\begin{array}{rl} 010111 & (23) \\ \underbrace{0}_{1}\underbrace{1}_{3}\underbrace{1}_{2}\underbrace{0}_{} & \underline{(30)} \\ & 690 \end{array}$$

Booth Recoding

- Now we need to be able to multiply by 0,1,2, or 3
  - +3 is not easy to implement; requires two stages (+4X – 1X OR +2X + 1X)
- Go with +4x – 1X, and "carry" the 4x to the next PP
- Ex: 010111 (23)
  011110 (30)
  2 -1 2 690

Modified Booth recoding

- Now we have a new linear path through the recodings. Completely unacceptable for tree multipliers e.g.
- We need to make the recoding local
- Instead look at three bits at a time (our two, and previous high order bit) and use negative 2 as well:
  - ±2X, ±1X, 0
  - Must be able to multiply by 0, 1, 2, -1, -2
  - 0 and 1 are easy, 2X involves a shift left by 1 bit position, -1X: invert all bits and set Cin = 1, -2X: invert all bits, carry in a 1, and shift left by 1 bit

Recoding table

- Note that when high bit 1 always a negative (i.e. “carry” of 4Y)
- So just that high bit of last PP used as “carry in”

$x_{i+2}x_{i+1}x_i$	Add to partial product
000	+0Y
001	+1Y
010	+1Y
011	+2Y
100	-2Y
101	-1Y
110	-1Y
111	-0Y

Example

- 010111 (23)
  011110 (30)
  690

Originally: 011110

011 → +2

111 → 0

100 → -2

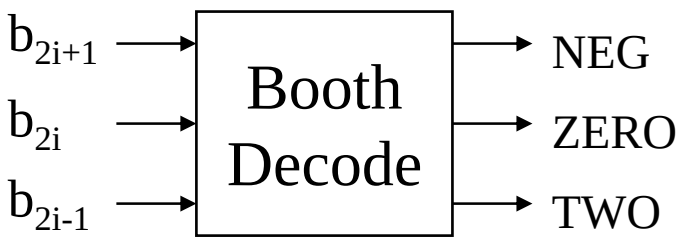
LSB extends with 0s

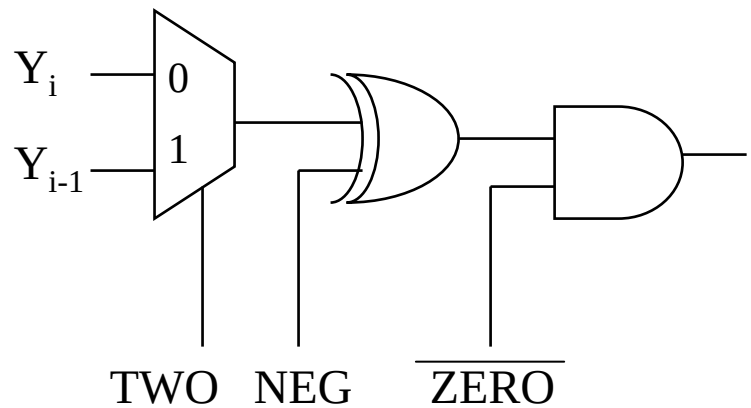
So we have:

(+2)(0)(-2)

Booth Decoding and Partial Product Generation

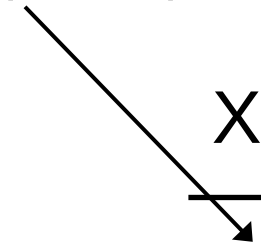
Operation	NEG	ZERO	TWO
x 0	0	1	0
x 1	0	0	0
x (-1)	1	0	0
x 2	0	0	1
x (-2)	1	0	1





Example, two 8-bit negative #'s

Extend sign in partial products



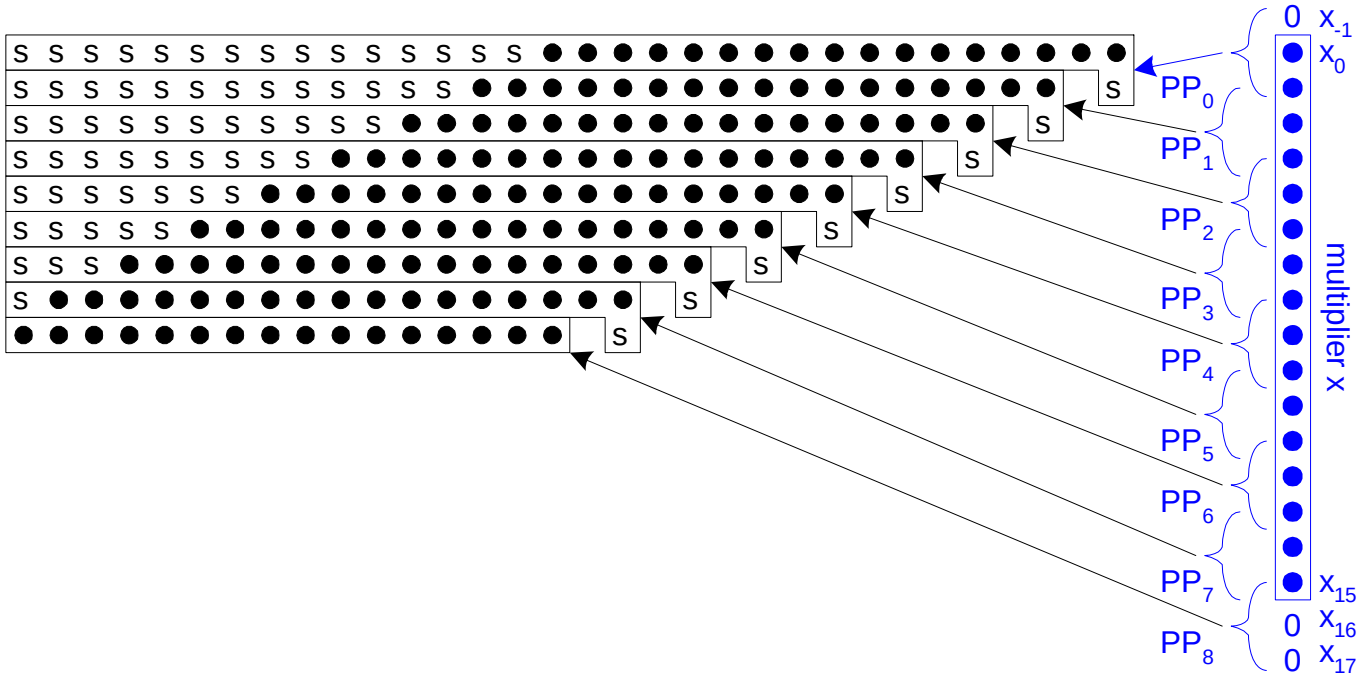
```
                10110010   = -78
        X       10011101   = -99
  ------------------------------
  1111111110110010
    00000001001110
     111101100100
      0010011100
  ------------------------------
 1 1000111000101010   = 7722
```

Ignore carry out into 17th place

Recode:
10011101

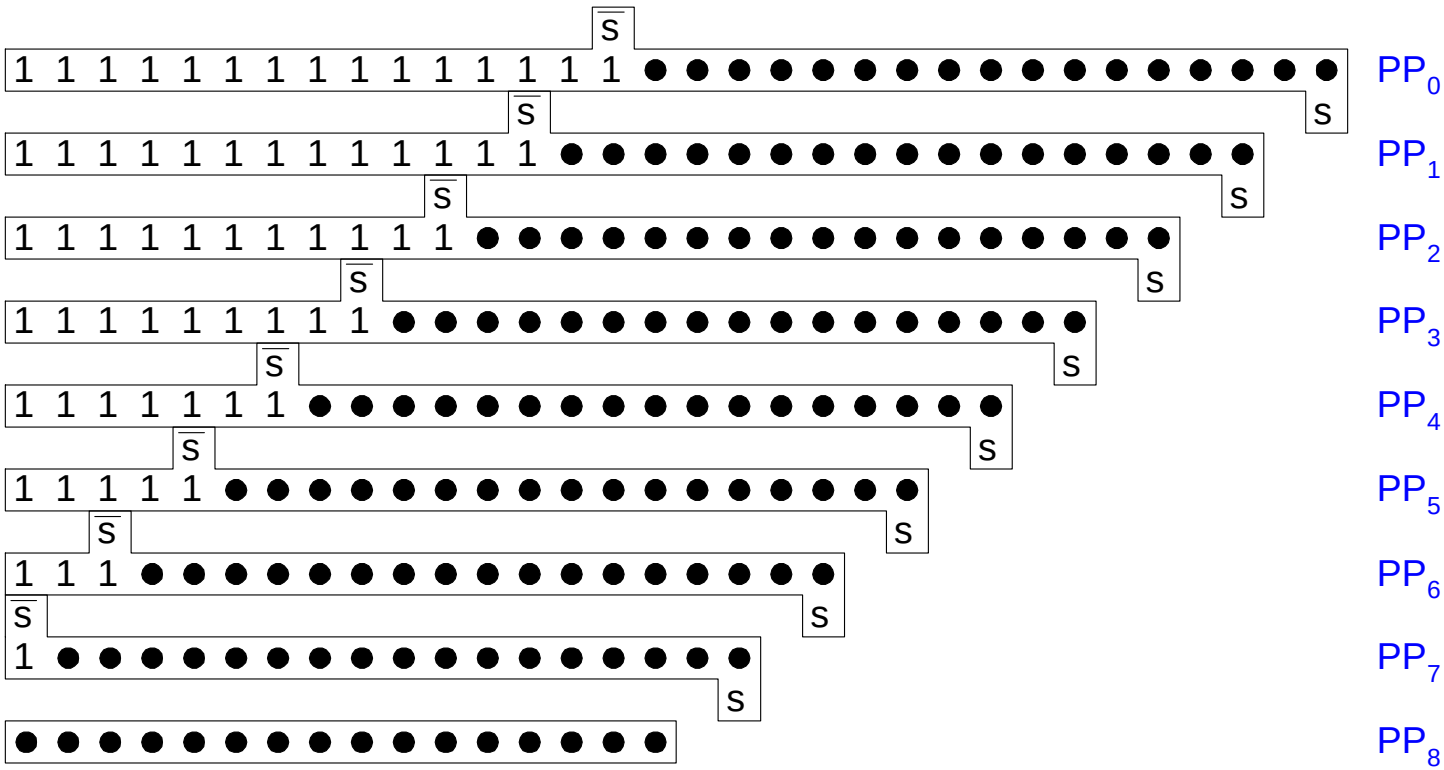
Sign Extension

- Partial products can be negative
  - Require sign extension, which is cumbersome



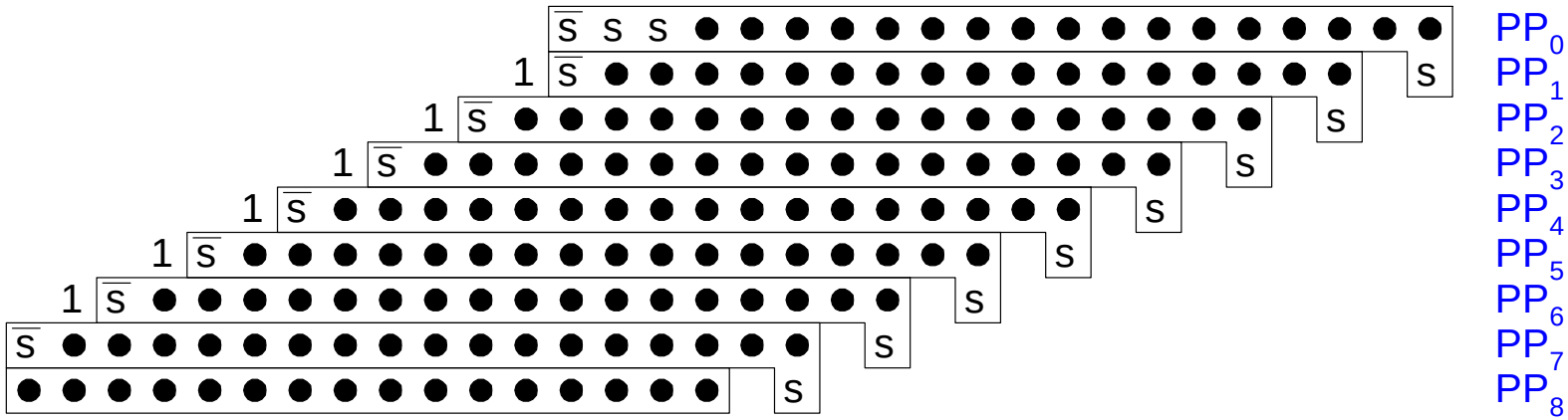
Simplified Sign Ext.

- Sign bits are either all 0's or all 1's
  - Note that all 0's is all 1's + 1 in proper column



Even Simpler Sign Ext.

- No need to add all the 1's in hardware
  - Precompute the answer!



Summary

- Generally speaking, multiply function consists of AND functions to generate the partial products and lots of addition
  - Carry and sum delays of adder cells can be equally critical
- Modified Booth recoding reduces the # of partial products to be added, improves speed
  - Also suitable for 2s complement addition
- Other topics:
  - Can pipeline within the multiplier unit to improve throughput
  - Tree structures to reduce the # of adders needed and speed the result (speed becomes logarithmic in # of bits)