CMPEN 411
VLSI Digital Circuits
Spring 2012

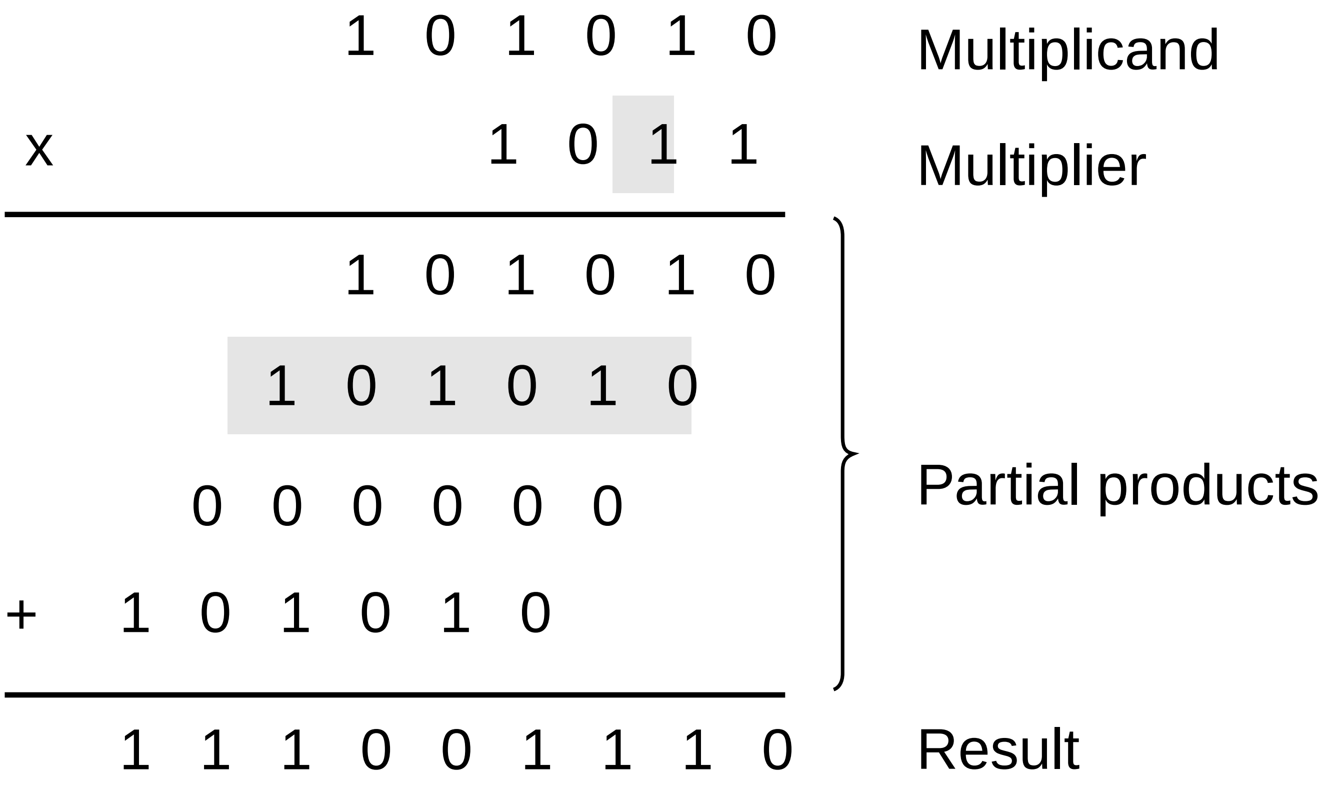
Lecture 20: Multiplier Design

[Adapted from Rabaey's *Digital Integrated Circuits*, Second Edition, ©2003 J. Rabaey, A. Chandrakasan, B. Nikolic]

Review: Basic Building Blocks

- Datapath
  - Execution units
    - Adder, multiplier, divider, shifter, etc.
  - Register file and pipeline registers
  - Multiplexers, decoders
- Control
  - Finite state machines (PLA, ROM, random logic)
- Interconnect
  - Switches, arbiters, buses
- Memory
  - Caches (SRAMs), TLBs, DRAMs, buffers

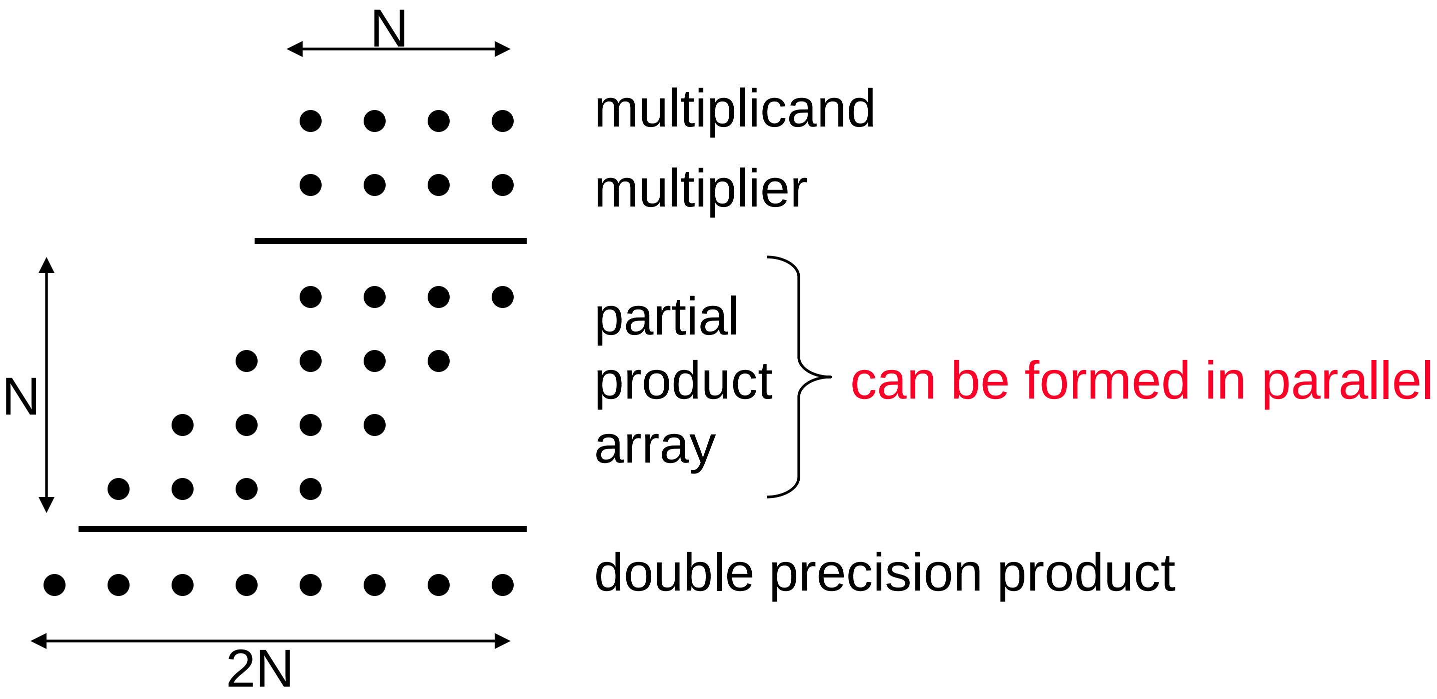
The Binary Multiplication



```
              1   0   1   0   1   0        Multiplicand
x                     1   0   1   1        Multiplier
  ---------------------------------
              1   0   1   0   1   0
          1   0   1   0   1   0
      0   0   0   0   0   0                Partial products
+ 1   0   1   0   1   0
  ---------------------------------
  1   1   1   0   0   1   1   1   0        Result
```

Multiply Operation

- Multiplication is just a a lot of additions



multiplicand

multiplier

partial product array

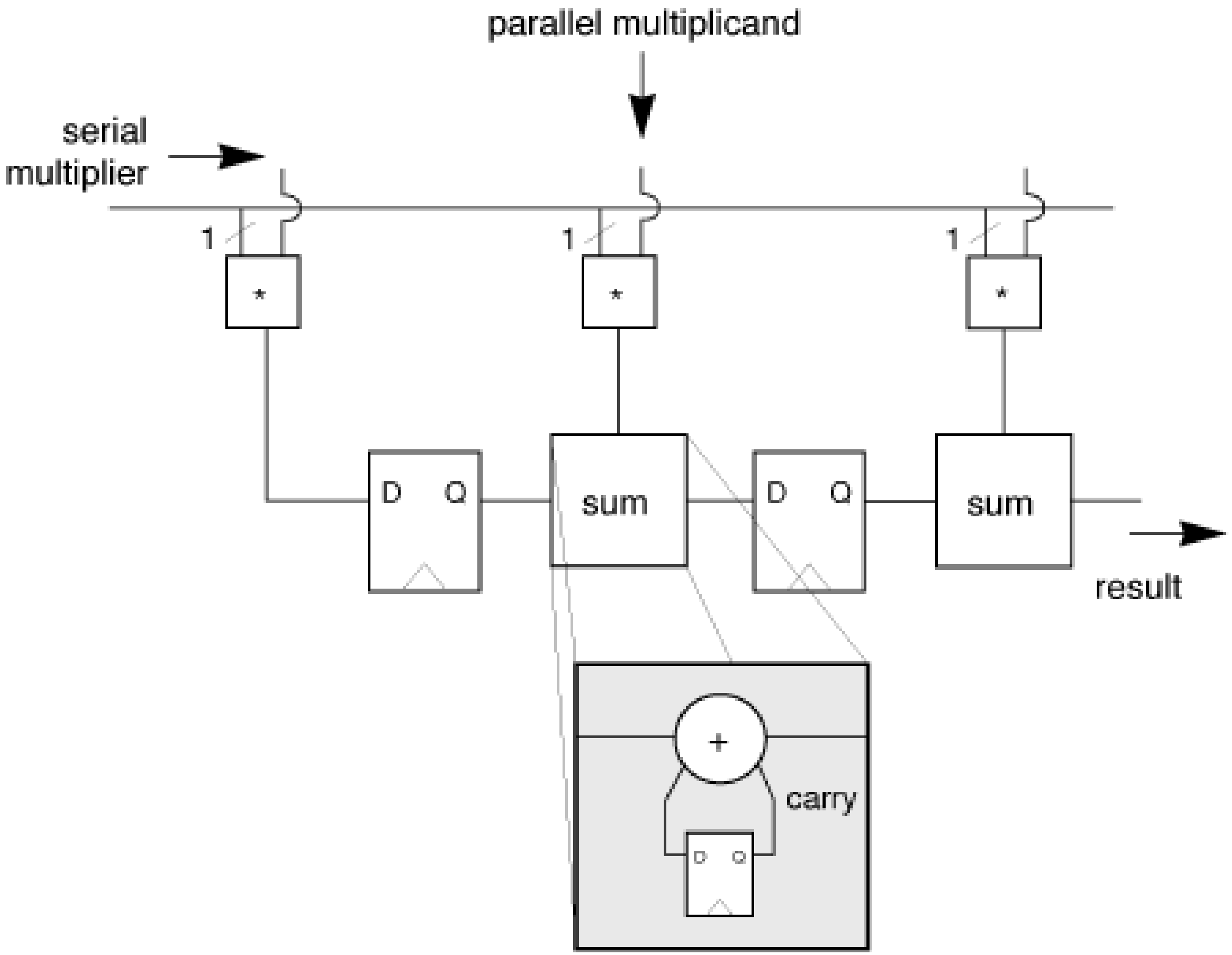
can be formed in parallel

double precision product

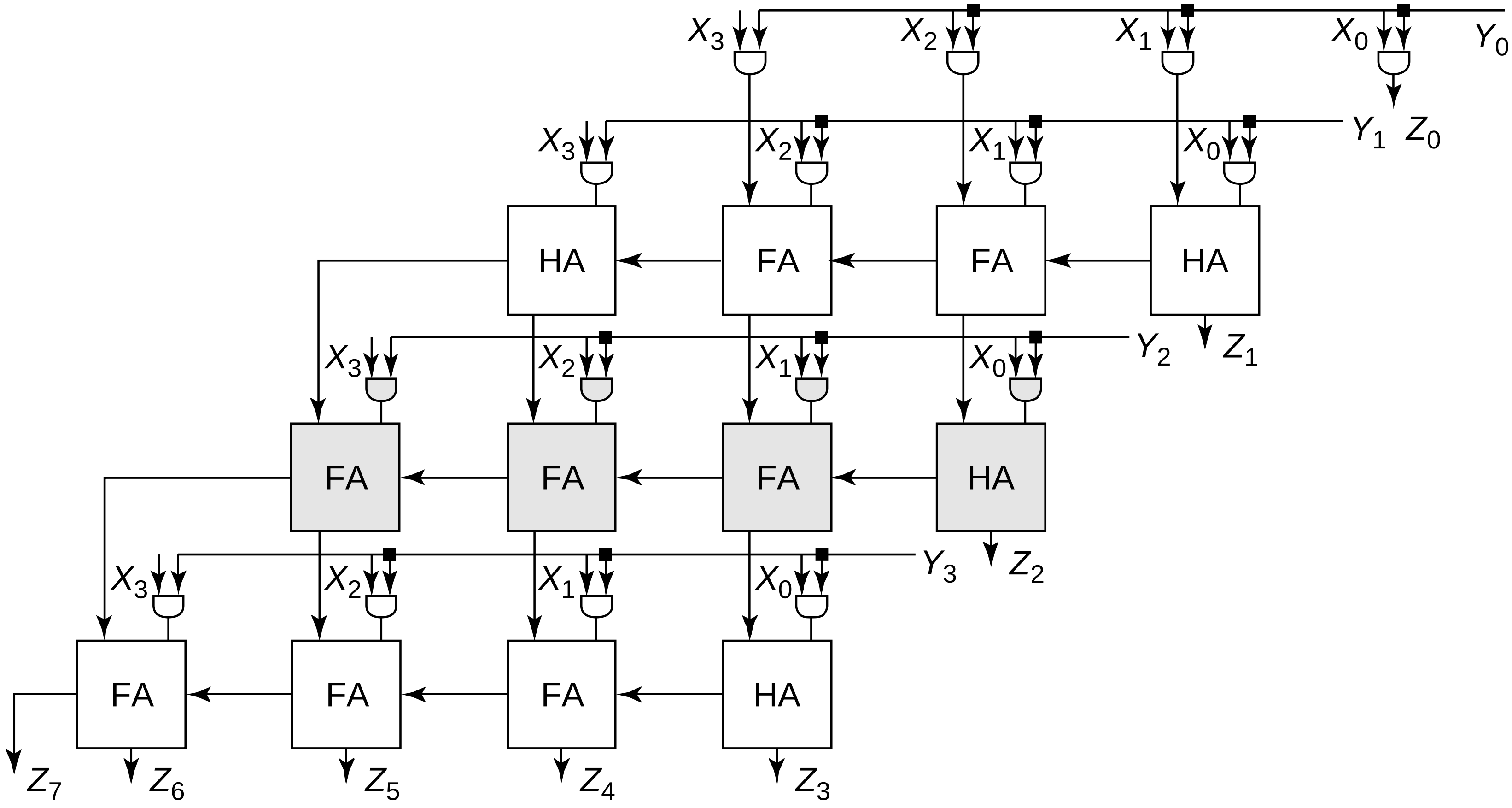
Multiplication Approaches

- Right shift and add
  - Partial product array rows are accumulated from top to bottom on an N-bit adder
    - After each addition, right shift (by one bit) the accumulated partial product to align it with the next row to add
  - Time for N bits $T_{serial\_mult} = O(N\ T_{adder}) = O(N^2)$ for a RCA
- Making it faster
  - Use a faster adder
  - Use higher radix (e.g., base 4) multiplication – $O(N/2\ T_{adder})$
    - Use multiplier recoding to simplify multiple formation (booth)
  - Form the partial product array in parallel and add it in parallel
- Making it smaller (i.e., slower)
  - Use serial-parallel mult
  - Use an array multiplier
    - Very regular structure with only short wires to nearest neighbor cells. Thus, very simple and efficient layout in VLSI Can be easily and efficiently pipelined

Serial-parallel multiplier structure



The Array Multiplier



Booth multiplier

- Encoding scheme to reduce number of stages in multiplication.
- Performs two bits of multiplication at once—requires half the stages.
- Each stage is slightly more complex than simple multiplier, but adder/subtracter is almost as small/fast as adder.

Booth encoding

- Two's-complement form of multiplier:
  - $y = -2^n y_n + 2^{n-1} y_{n-1} + 2^{n-2} y_{n-2} + \ldots$ (first bit is the sign bit)

    (example, y=18=010010 y= -18 = 101110)
- Rewrite using $2^a = 2^{a+1} - 2^a$:
  - $y = 2^n(y_{n-1} - y_n) + 2^{n-1}(y_{n-2} - y_{n-1}) + 2^{n-2}(y_{n-3} - y_{n-2}) + \ldots$
- Consider first two terms: by looking at three bits of y, we can determine whether to add *x*, *2x* to partial product.

Booth actions

- $y = 2^n(y_{n-1} - y_n) + 2^{n-1}(y_{n-2} - y_{n-1}) + 2^{n-2}(y_{n-3} - y_{n-2}) + \ldots$

❑ Consider first two terms: by looking at three bits of y, we can determine whether to add *x*, *2x* to partial product.

y_i y_{i-1} y_{i-2}	increment
0 0 0	0
0 0 1	x
0 1 0	x
0 1 1	2x
1 0 0	-2x
1 0 1	-x
1 1 0	-x
1 1 1	0

Booth example

- $x = 1001\ (9_{10})$, $y = 0111\ (7_{10})$.
- $P_0 = 00000000$
- $y_3y_2y_1$=011 $y_1y_0y_{-1}$=11(0)
- $y_1y_0y_{-1}$ = 110, $P_1 = P_0$ - (1001) = 11110111
- x shift left for 2 bits to be 100100
- $y_3y_2y_1$ = 011, $P_2 = P_1$+ (10*100100) =

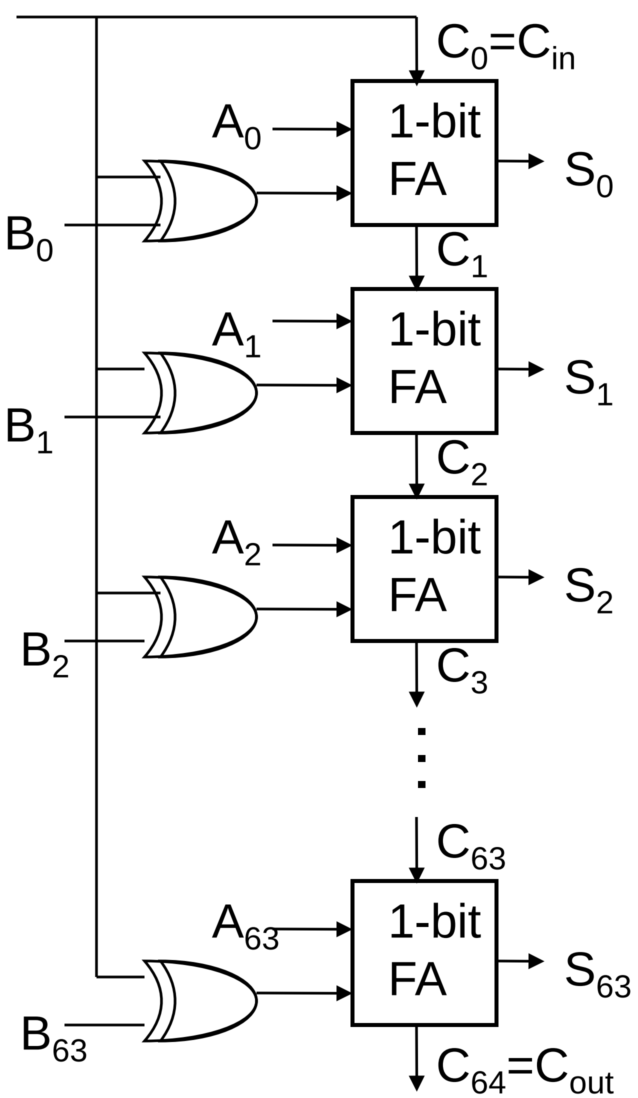
  11110111+01001000 = 001111111 (63_{10})
- An array multiplier needs N addtions, booth multiplier needs only N/2 additions

Review: A 64-bit Adder/Subtractor

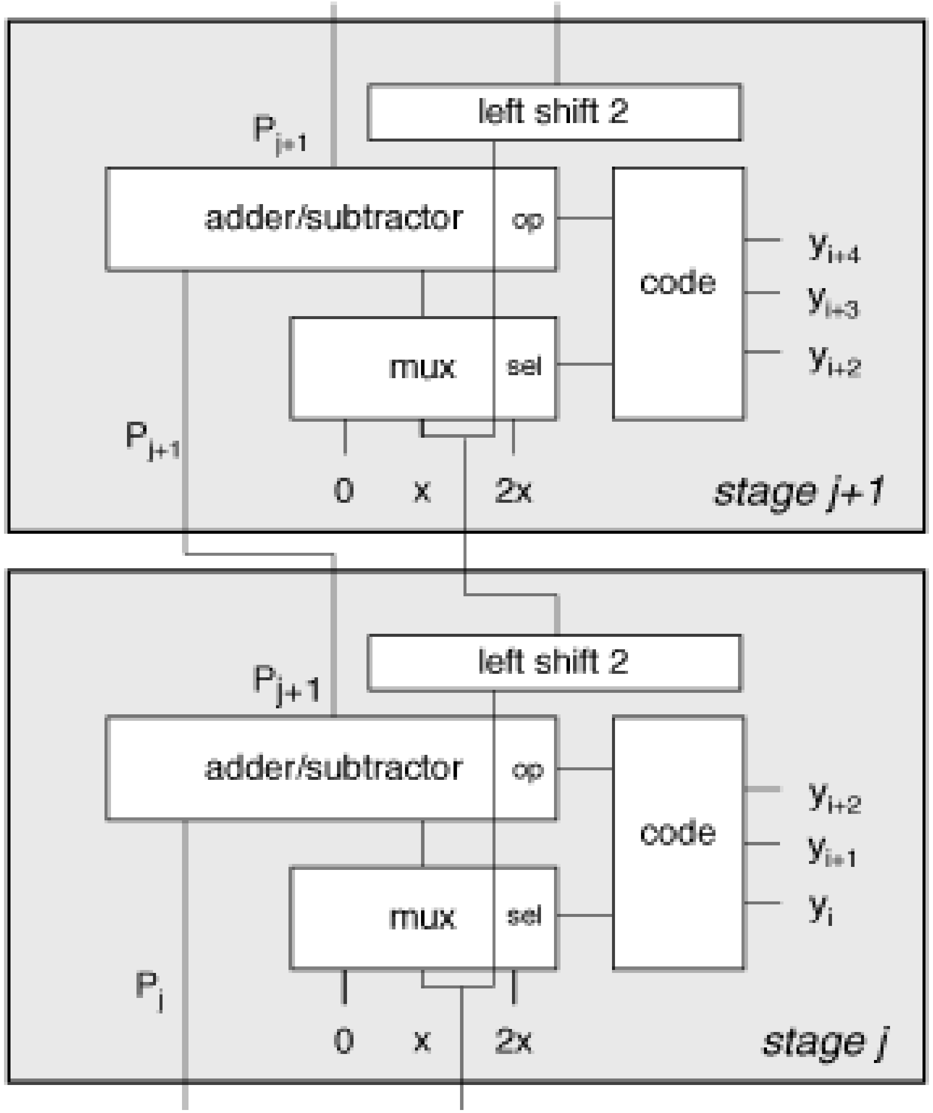
- Ripple Carry Adder (RCA) built out of 64 FAs

- Subtraction – complement all subtrahend bits (xor gates) and set the low order carry-in

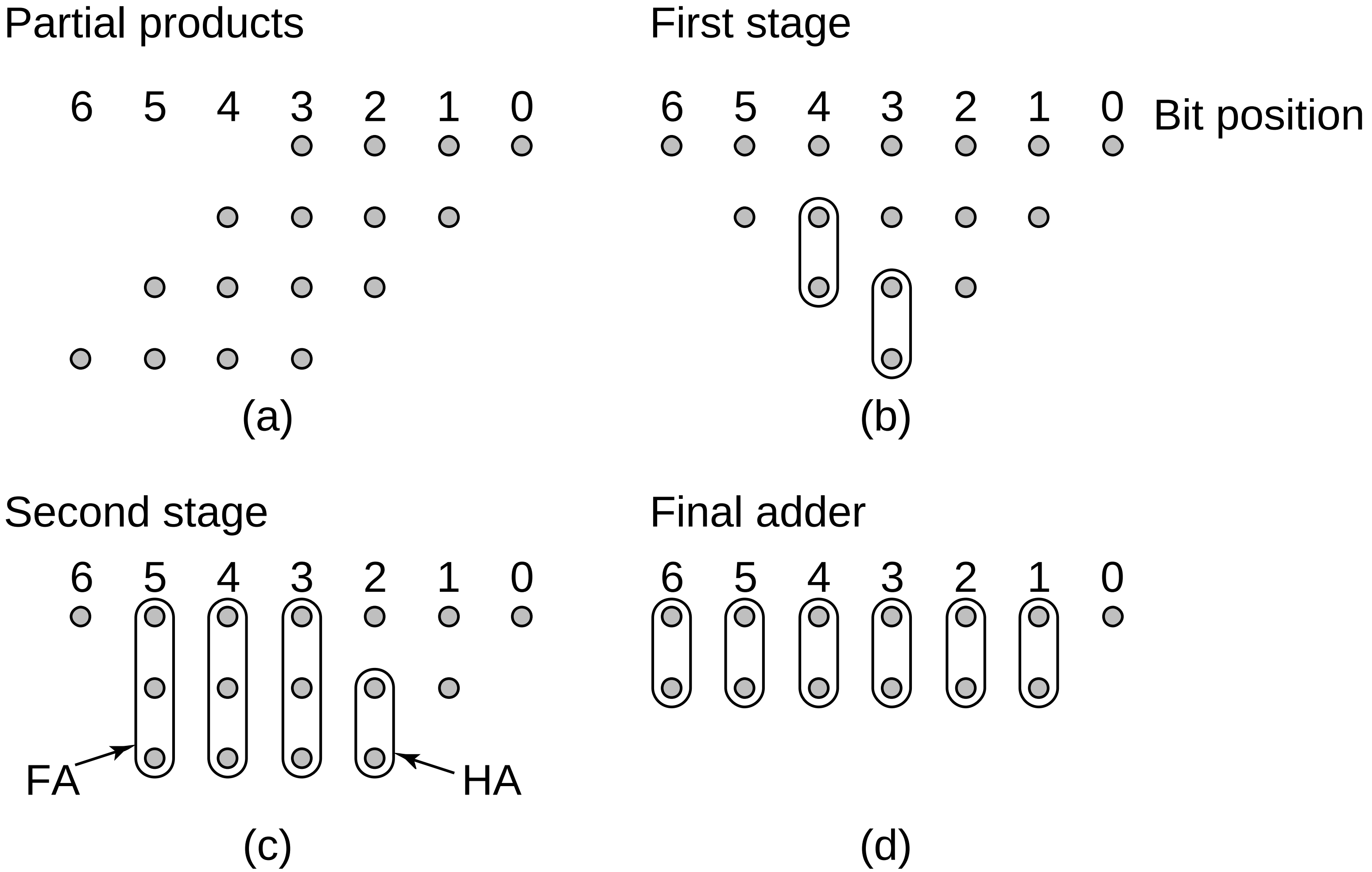
- RCA
  - advantage: simple logic, so small (low cost)
  - disadvantage: slow (O(N) for N bits) and lots of glitching (so lots of energy consumption)



Booth structure



Wallace-Tree Multiplier



Partial products

(a)

First stage

(b)

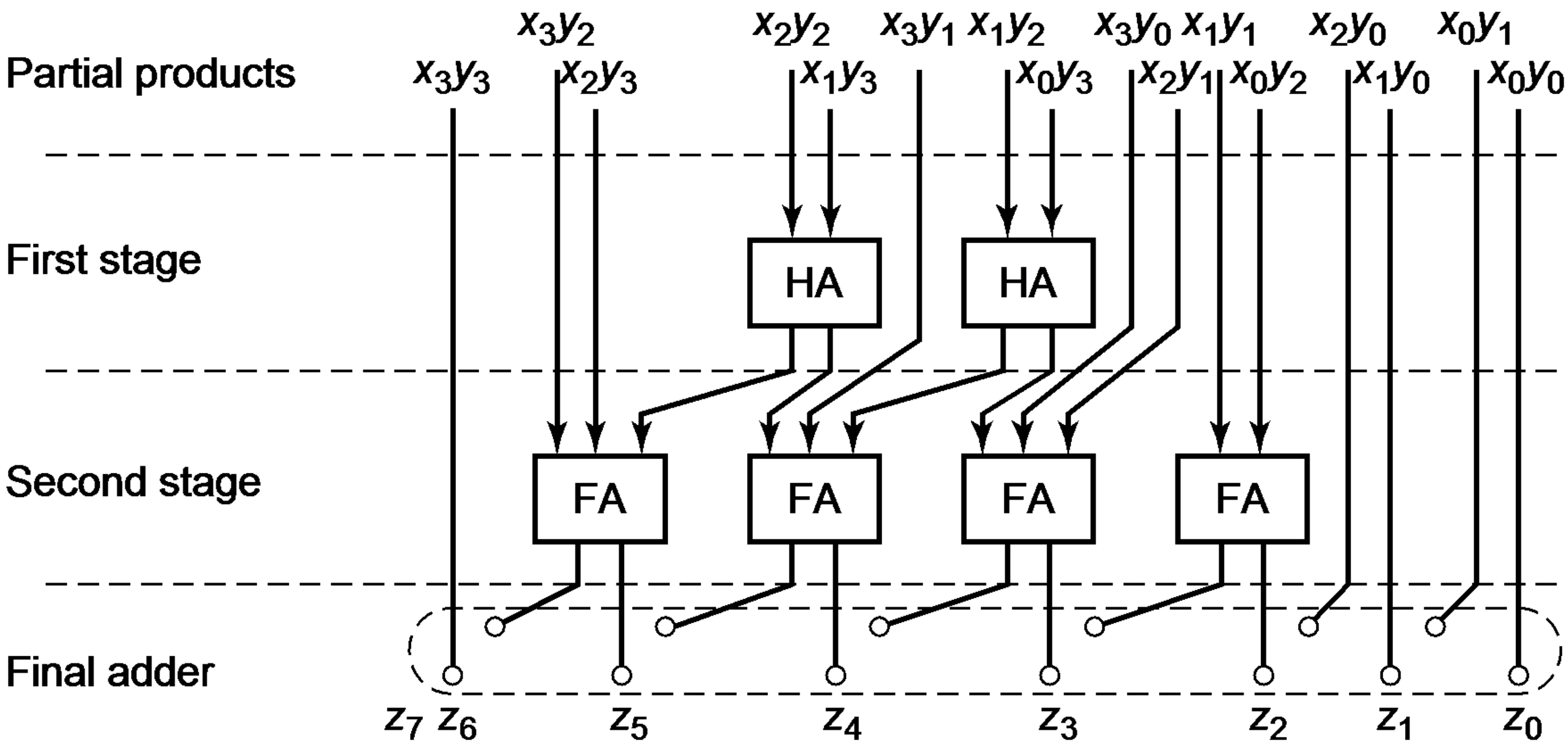
Second stage

(c)

Final adder

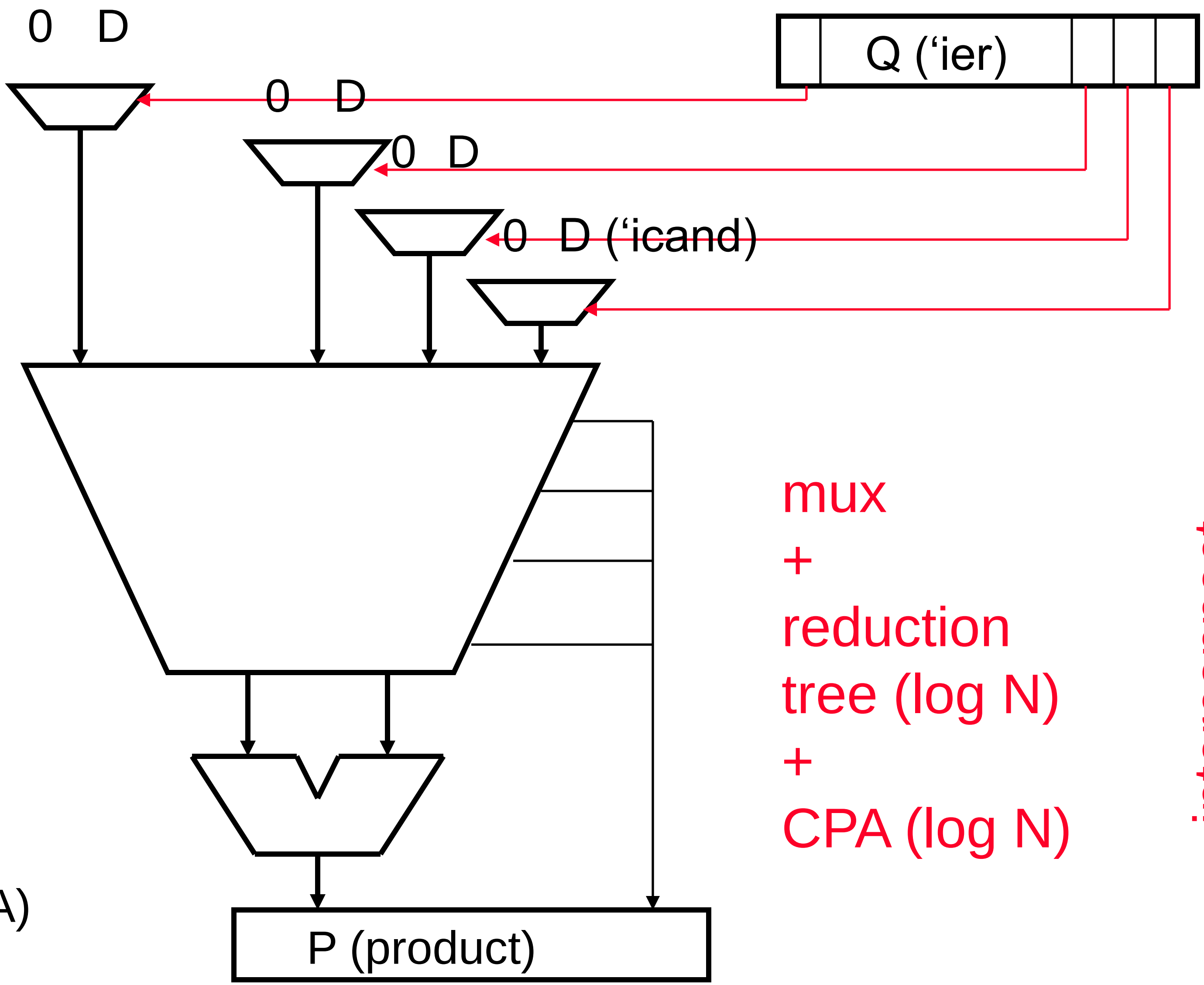
(d)

Wallace-Tree Multiplier



Full adder = (3,2) compressor

Making it Faster: Tree Multiplier Structure



multiple
forming
circuits

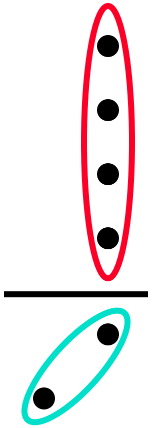
partial
product
array
reduction
tree

fast carry
propagate
adder (CPA)

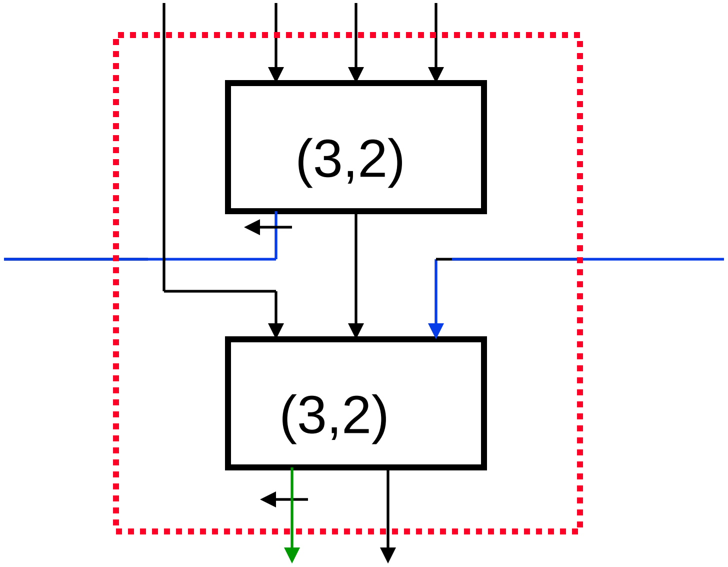
mux
+
reduction
tree (log N)
+
CPA (log N)

interconnect

(4,2) Counter

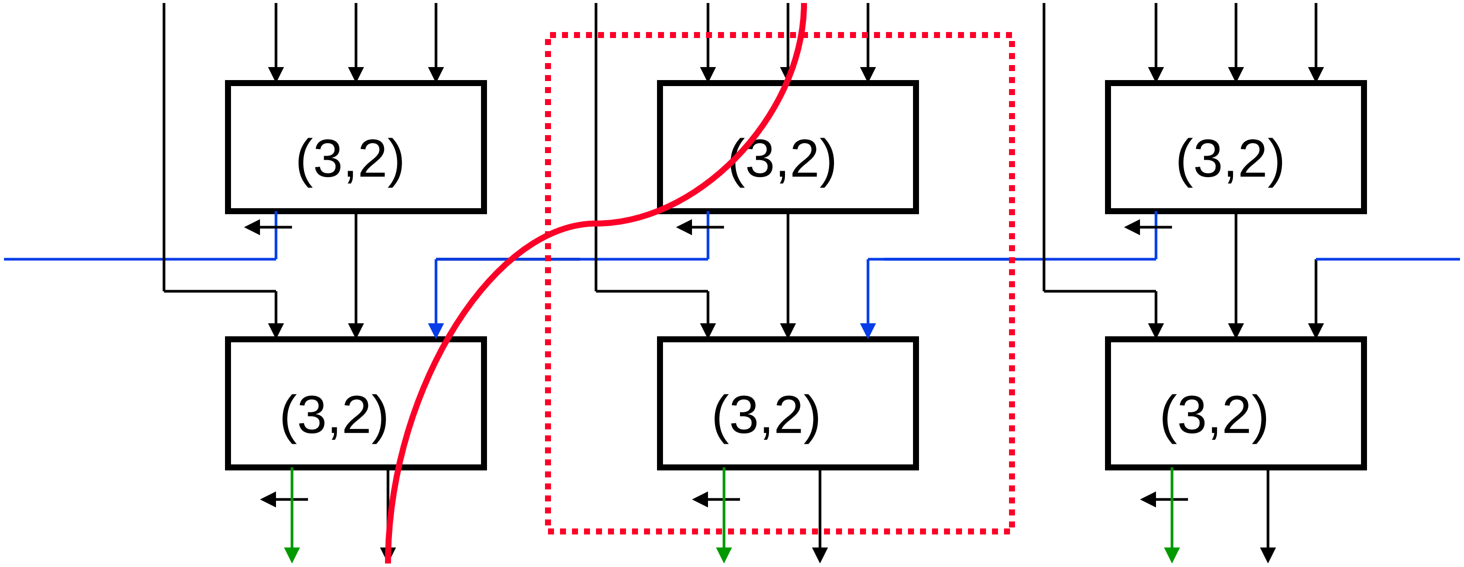


- Built out of two (3,2) counters (just FA's!)
  - all of the inputs (4 external plus one internal) have the same weight (i.e., are in the same bit position)
  - the internal carry output is fed to the next higher weight position (indicated by the ←)



Note: Two carry outs - one "internal" and one "external"

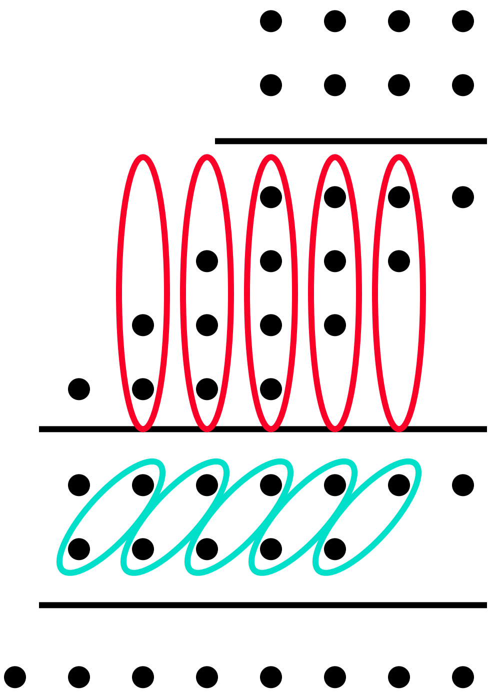
Tiling (4,2) Counters



- Reduces columns four high to columns only two high
  - Tiles with neighboring (4,2) counters
  - Internal carry in at same “level” (i.e., bit position weight) as the internal carry out

4x4 Partial Product Array Reduction

- ❑ Fast 4x4 multiplication using (4,2) counters



multiplicand

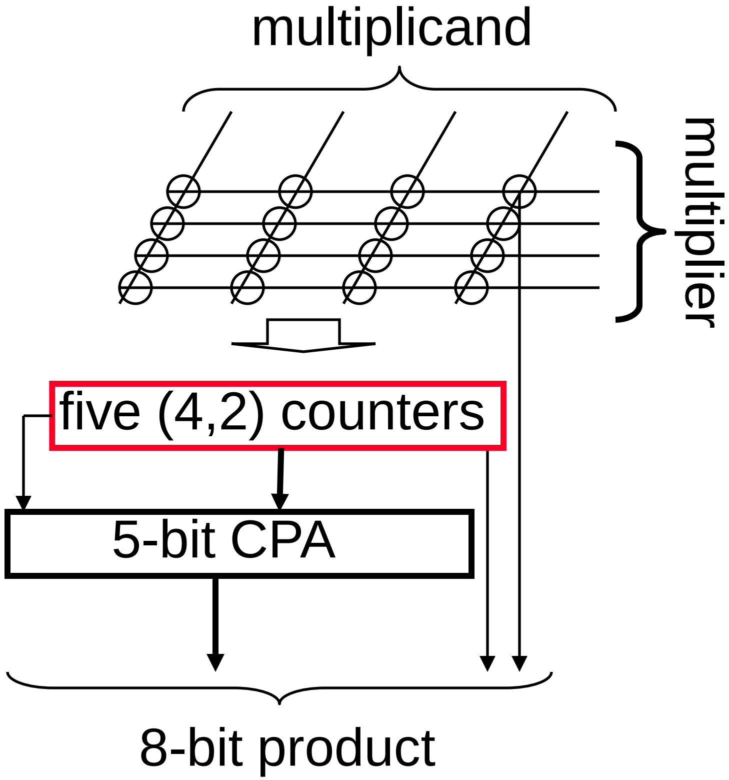
multiplier

partial product array

reduced pp array (to CPA)

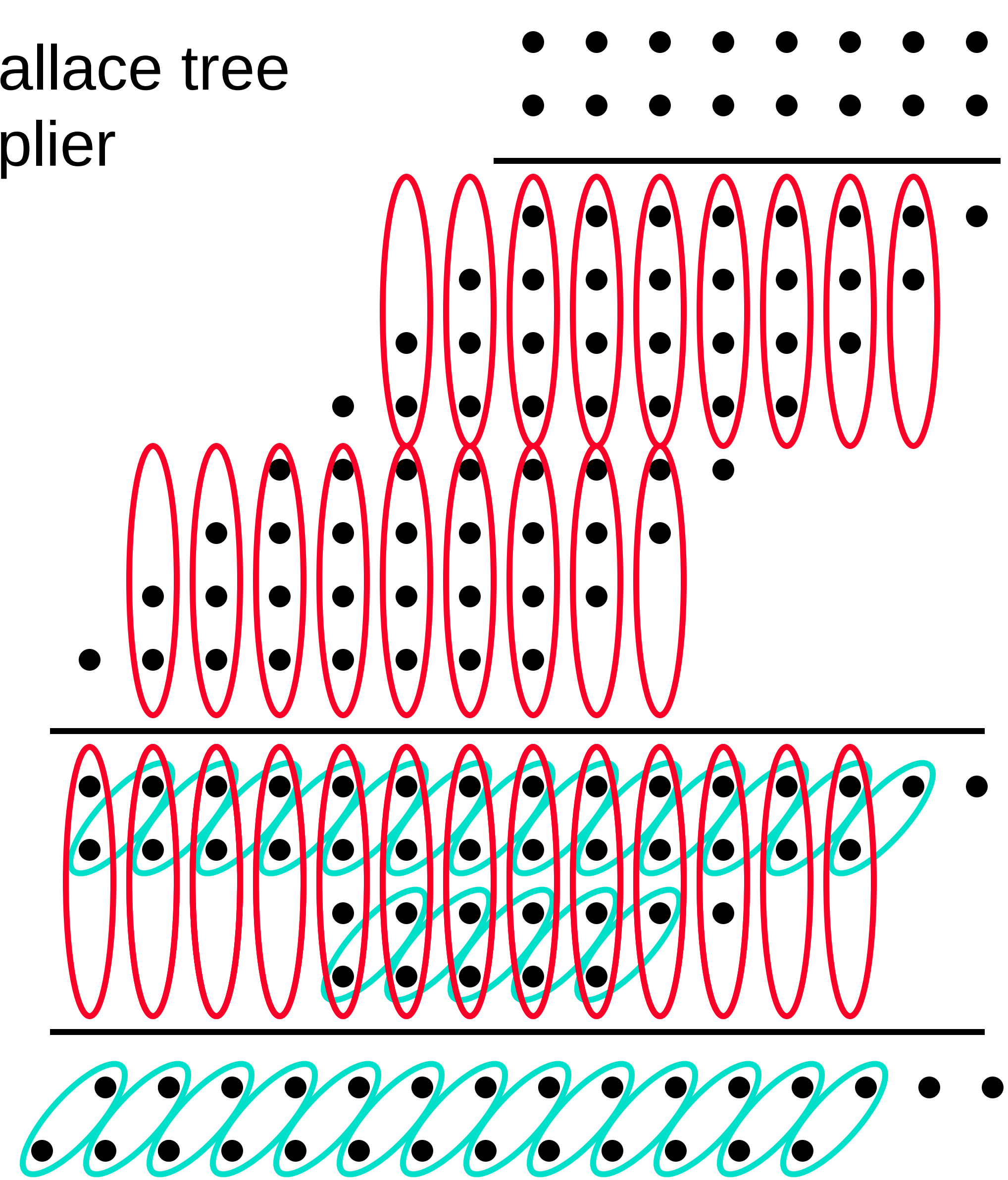
double precision product

- ❑ How would you lay it out?



8x8 Partial Product Array Reduction

- Wallace tree multiplier



‘icand

‘ier

partial product array

two rows of nine (4,2) counters

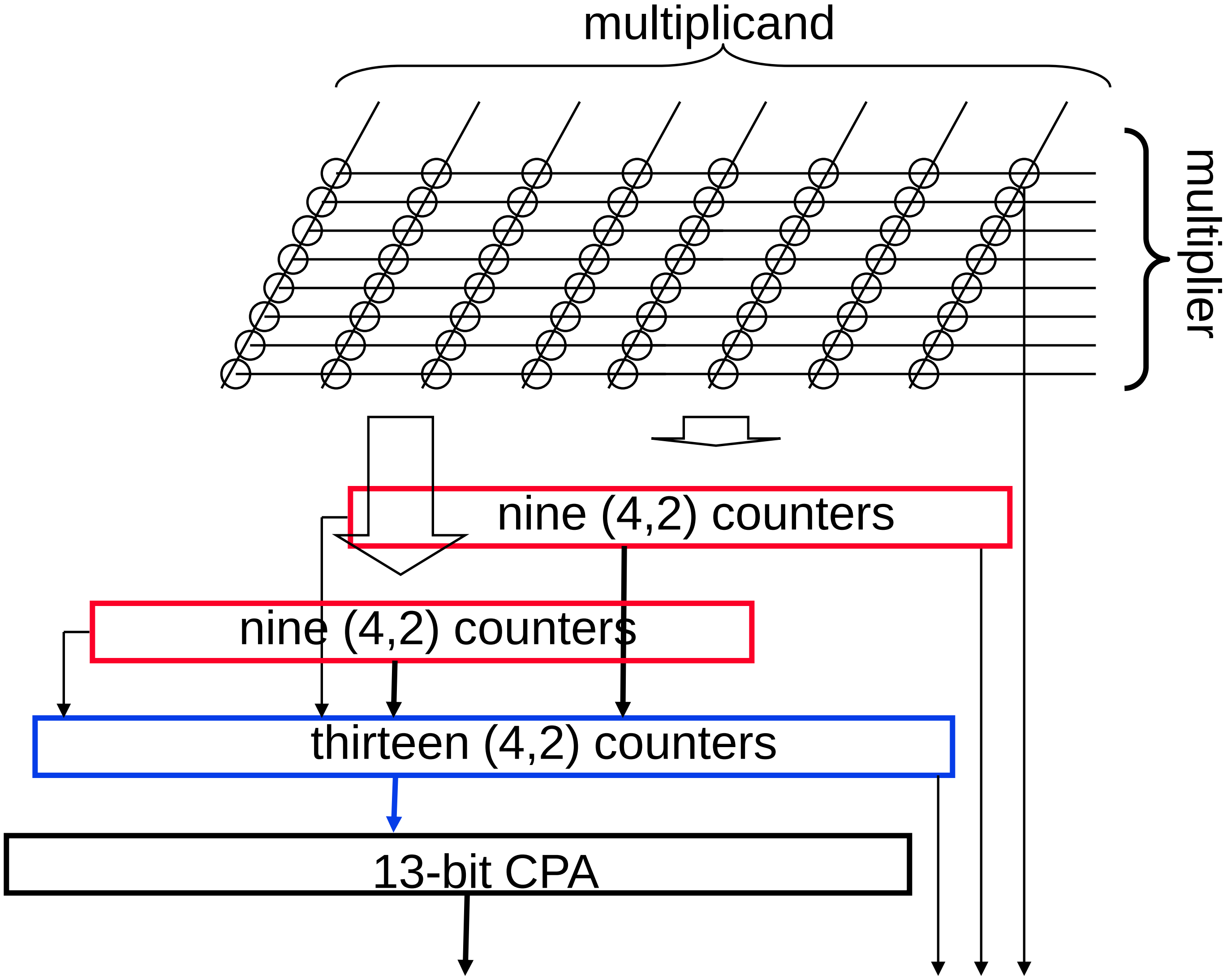
reduced partial product array

one row of thirteen (4,2) counters

to a 13-bit fast CPA

An 8x8 Multiplier Layout

- How should it be laid out?



Multipliers —Summary

- Optimization Goals Different Vs Binary Adder
- Once Again: Identify Critical Path
- Other possible techniques
  - Logarithmic versus Linear (Wallace Tree Mult)
  - Data encoding (Booth)
  - Pipelining

FIRST GLIMPSE AT SYSTEM LEVEL OPTIMIZATION

Next Lecture and Reminders

- Next lecture
  - Shifters, decoders, and multiplexers
    - Reading assignment – Rabaey, et al, 11.5-11.6