Arithmetic Circuits

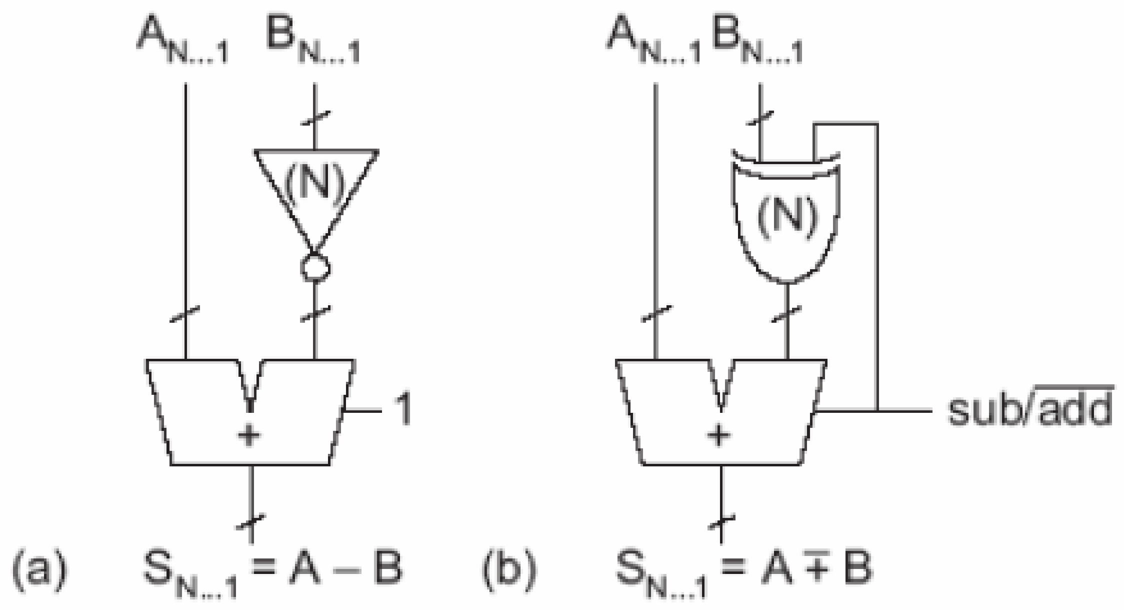
Dr. Shubhajit Roy Chowdhury,

Centre for VLSI and Embedded Systems Technology,
IIIT Hyderabad, India

Email: src.vlsi@iiit.ac.in

Subtractors

- Two's complement subtraction is easy:
  - Invert the second number and add

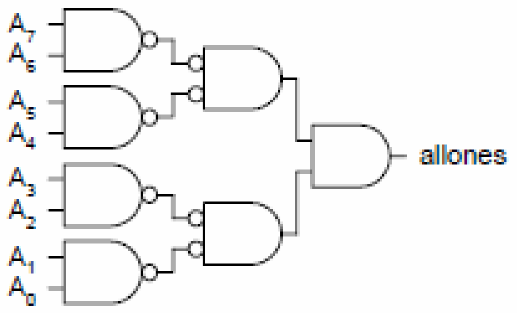


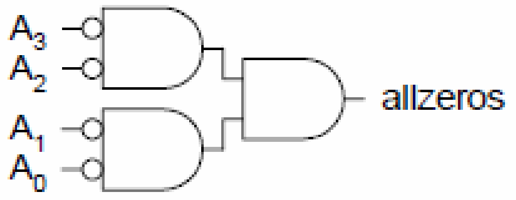
Comparators

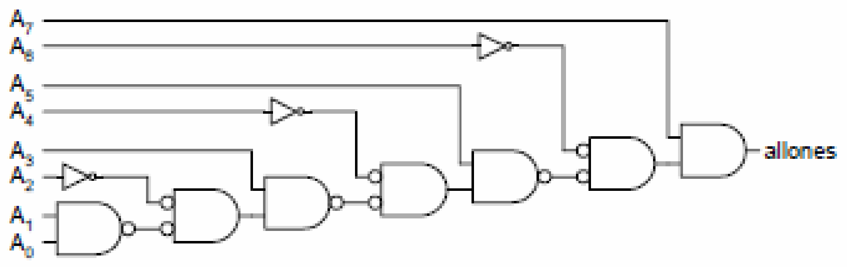
❑ 0's detector:	A = 00…000
❑ 1's detector:	A = 11…111
❑ Equality comparator:	A = B, A != B
❑ Magnitude comparator:	A < B, etc.

1’s and 0’s Detector

- 1’s detector: N-input AND gate
- 0’s detector: NOTs + 1’s detector (N-input NOR)

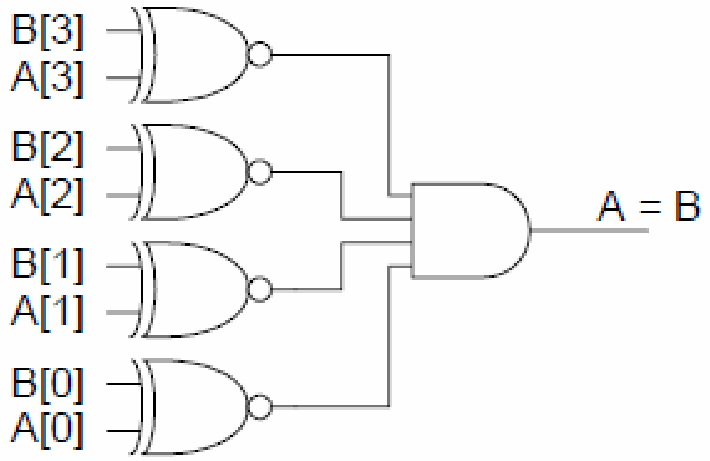






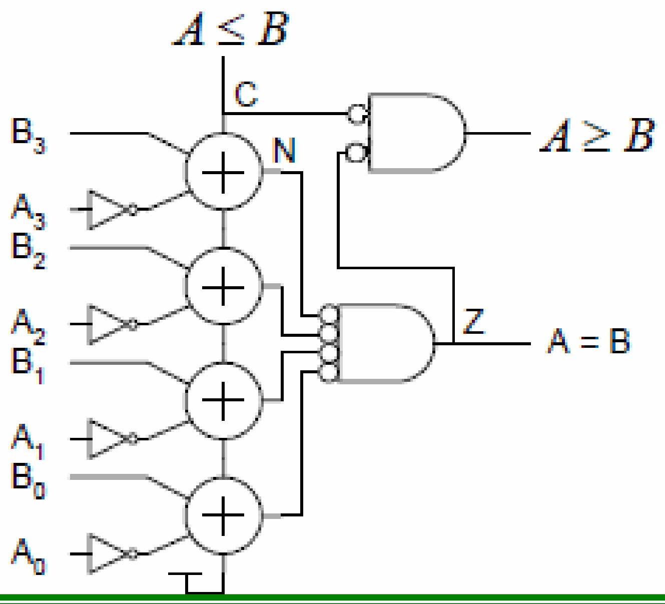
Equality detector

- Check if each bit is equal (XNOR, aka equality gate)
- 1's detect on bitwise equality



Magnitude Comparator

- Compute B-A and look at sign
- B-A = B + ~A + 1
- For unsigned numbers, carry out is sign bit



A+B=K Comparison

- Determining if A + B = K is easier than adding!
- Knowing bits, determine what carry in would be necessary if the sum is correct

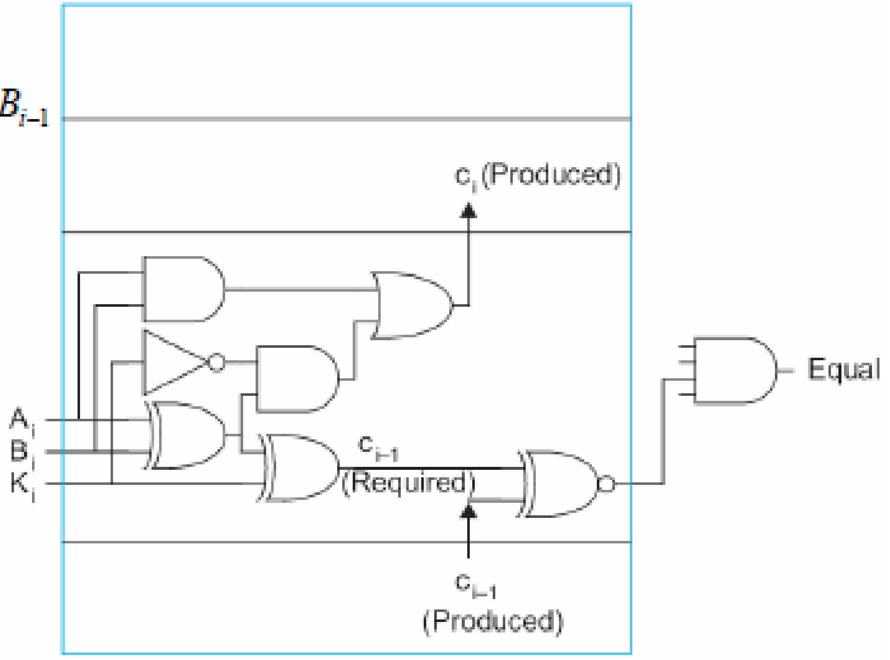
A_i	B_i	K_i	c_{i-1} (required)	c_i (produced)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	1
0	1	1	0	0
1	0	0	1	1
1	0	1	0	0
1	1	0	0	1
1	1	1	1	1

A+B=K Comparator Circuit

- Check if actual and required carries are equal

$$c_{i-1}(\text{required}) = A_i \oplus B_i \oplus K_i$$

$$c_{i-1}(\text{actual}) = (A_{i-1} \oplus B_{i-1})\overline{K}_{i-1} + A_{i-1}B_{i-1}$$

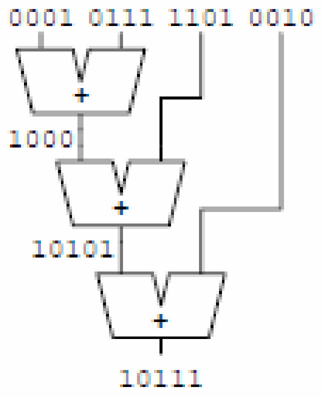


Multi-input Adders

- Suppose we want to add k N-bit words
  - Ex: 0001 + 0111 + 1101 + 0010 = _____

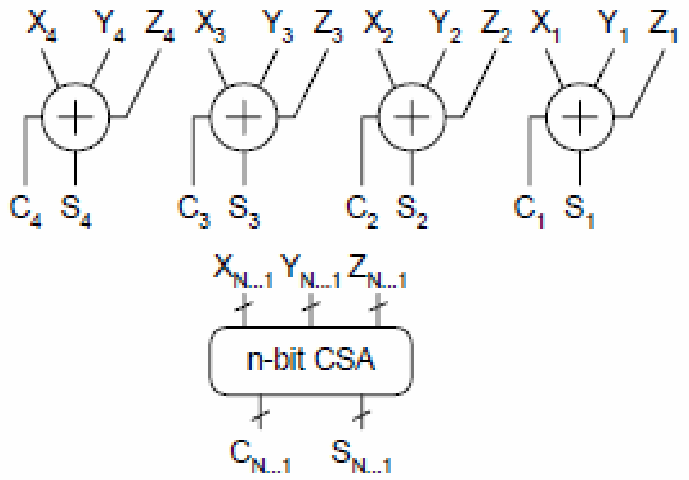
Multi-Input Adders

- ❑ 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



Carry Save Adders

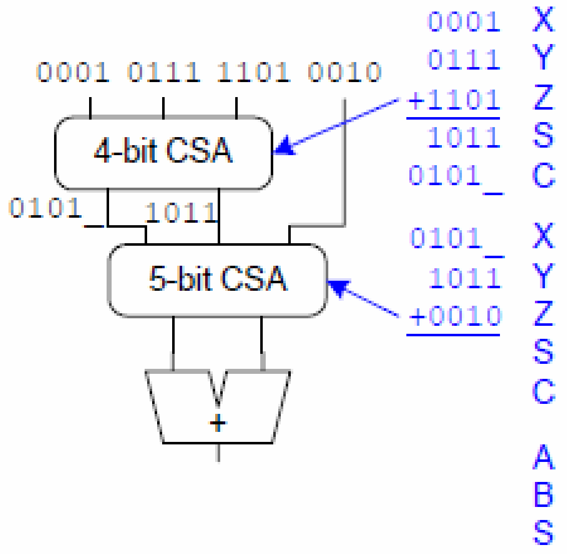
- A full adder sums 3 inputs and produces 2 outputs
  - Carry output has twice *weight* of sum output
- N full adders in parallel are called *carry save adder*
  - Produce N sums and N carry outs



X_4 Y_4 Z_4 X_3 Y_3 Z_3 X_2 Y_2 Z_2 X_1 Y_1 Z_1
C_4 S_4 C_3 S_3 C_2 S_2 C_1 S_1
$X_{N...1}$ $Y_{N...1}$ $Z_{N...1}$
n-bit CSA
$C_{N...1}$ $S_{N...1}$

Carry Save Adders

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



Multiplication

- Example:

$$\begin{array}{l} 1100 \ : \ 12_{10} \\ \underline{0101} \ : \ 5_{10} \end{array}$$

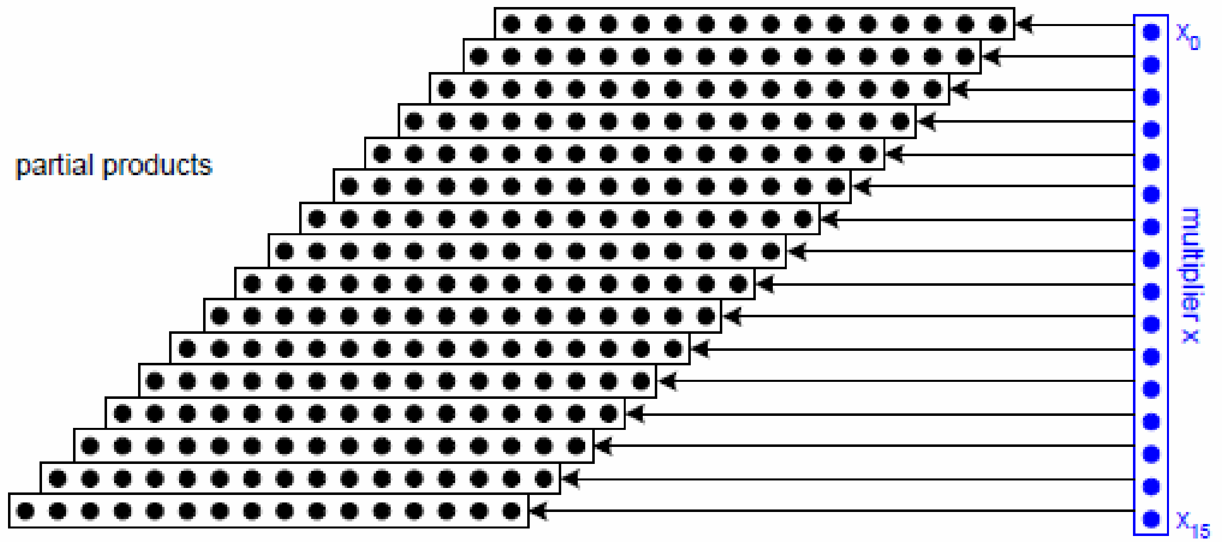
General Form of Multiplication

- Multiplicand: $Y = (y_{M-1}, y_{M-2}, \ldots, y_1, y_0)$
- Multiplier: $X = (x_{N-1}, x_{N-2}, \ldots, x_1, x_0)$
- Product: $P = \left(\sum_{j=0}^{M-1} y_j 2^j\right)\left(\sum_{i=0}^{N-1} x_i 2^i\right) = \sum_{i=0}^{N-1}\sum_{j=0}^{M-1} x_i y_j 2^{i+j}$

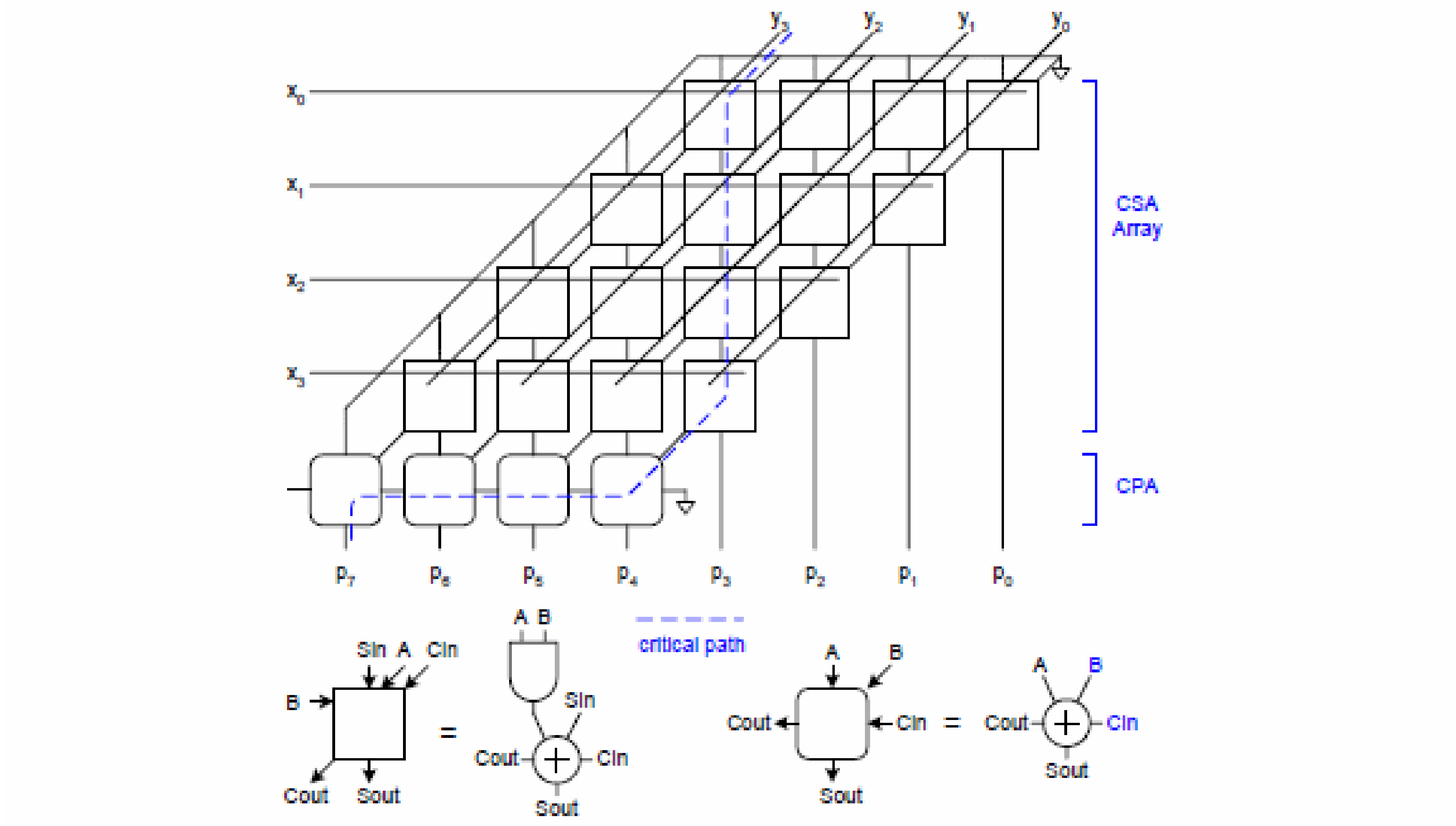
						y_5	y_4	y_3	y_2	y_1	y_0	multiplicand
						x_5	x_4	x_3	x_2	x_1	x_0	multiplier
						x_0y_5	x_0y_4	x_0y_3	x_0y_2	x_0y_1	x_0y_0	
					x_1y_5	x_1y_4	x_1y_3	x_1y_2	x_1y_1	x_1y_0		
				x_2y_5	x_2y_4	x_2y_3	x_2y_2	x_2y_1	x_2y_0			partial products
			x_3y_5	x_3y_4	x_3y_3	x_3y_2	x_3y_1	x_3y_0				
		x_4y_5	x_4y_4	x_4y_3	x_4y_2	x_4y_1	x_4y_0					
	x_5y_5	x_5y_4	x_5y_3	x_5y_2	x_5y_1	x_5y_0						
p_{11}	p_{10}	p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0	product

Dot diagram

- Each dot represents a bit

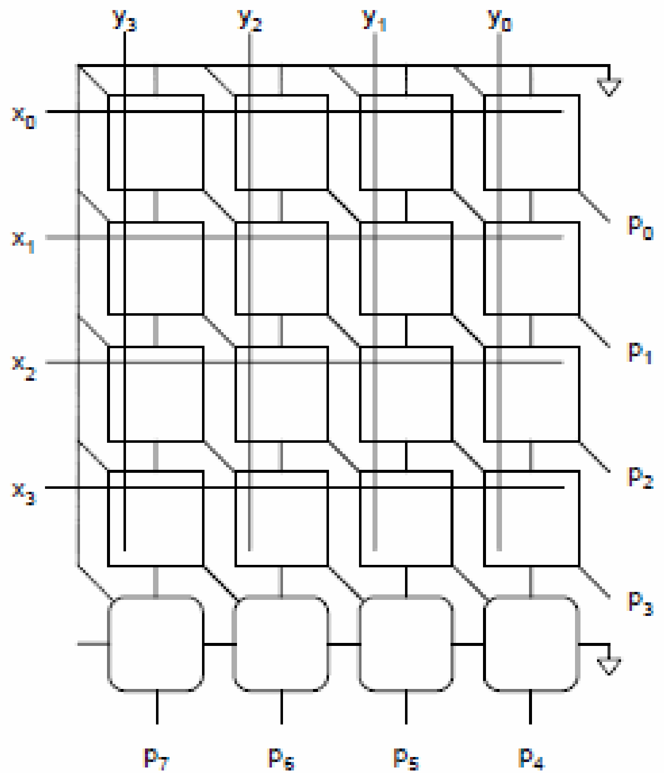


Array Multiplier



Rectangular Array

- Squash array to fit rectangular floorplan



Fewer Partial Products

- ❑ Array multiplier requires N partial products
- ❑ If we looked at groups of r bits, we could form N/r partial products.
  - – Faster and smaller?
  - – Called radix-2^r encoding
- ❑ Ex: r = 2: look at pairs of bits
  - – Form partial products of 0, Y, 2Y, 3Y
  - – First three are easy, but 3Y requires adder ☹

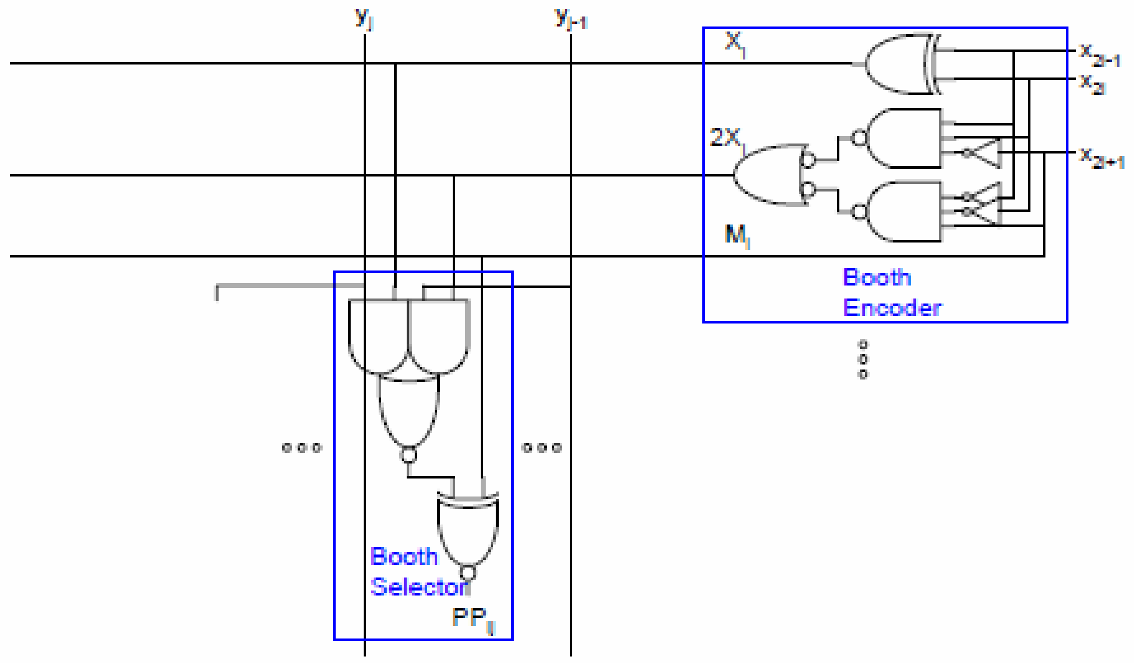
Booth Encoding

- Instead of 3Y, try –Y, then increment next partial product to add 4Y
- Similarly, for 2Y, try –2Y + 4Y in next partial product

Inputs			Partial Product	Booth Selects		
X_{2i+1}	X_{2i}	X_{2i-1}	PP_i	X_i	$2X_i$	M_i
0	0	0	0	0	0	0
0	0	1	Y	1	0	0
0	1	0	Y	1	0	0`
0	1	1	2Y	0	1	0
1	0	0	-2Y	0	1	1
1	0	1	-Y	1	0	1
1	1	0	-Y	1	0	1
1	1	1	-0	0	0	1

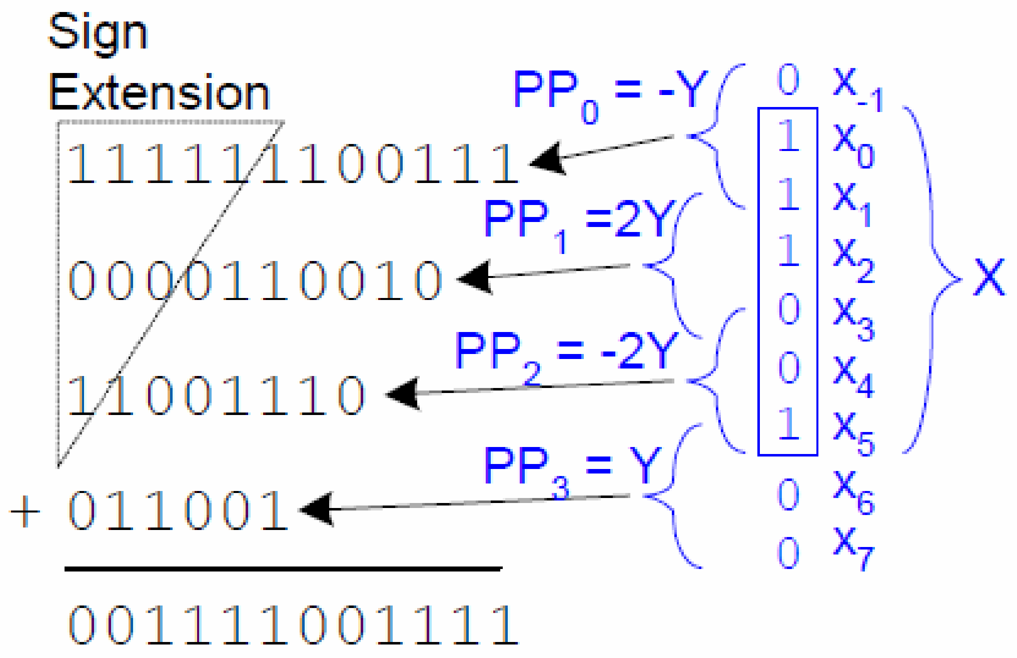
Booth Hardware

- Booth encoder generates control lines for each PP
  - Booth selectors choose PP bits



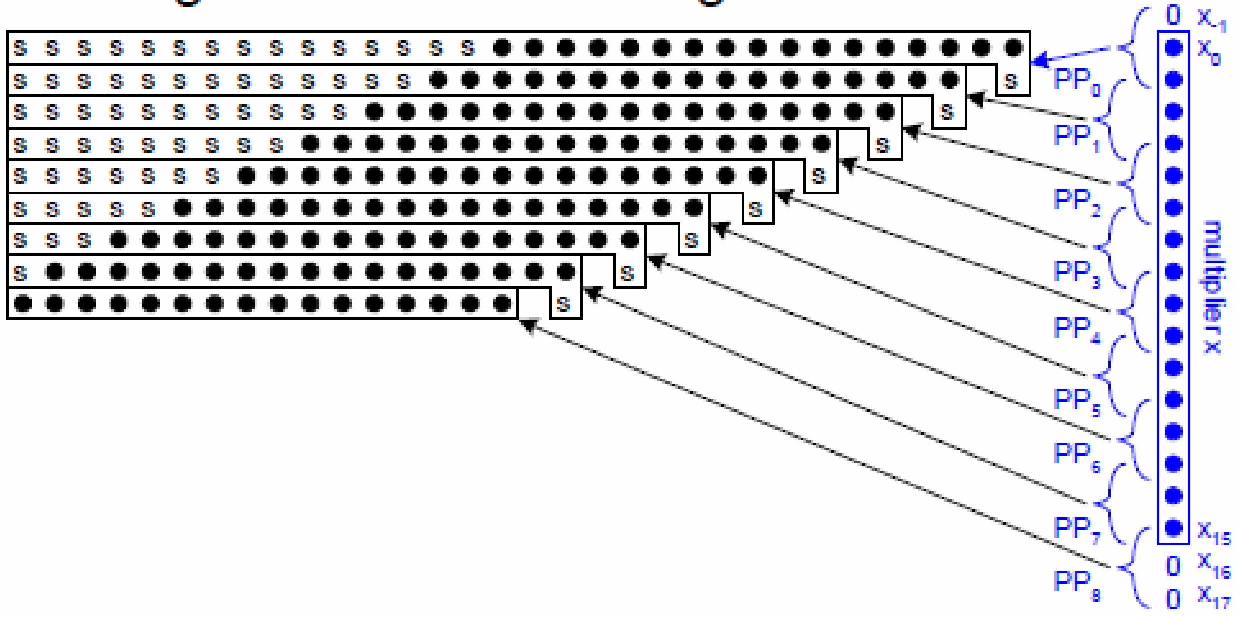
Booth Example

- 011001 x 100111



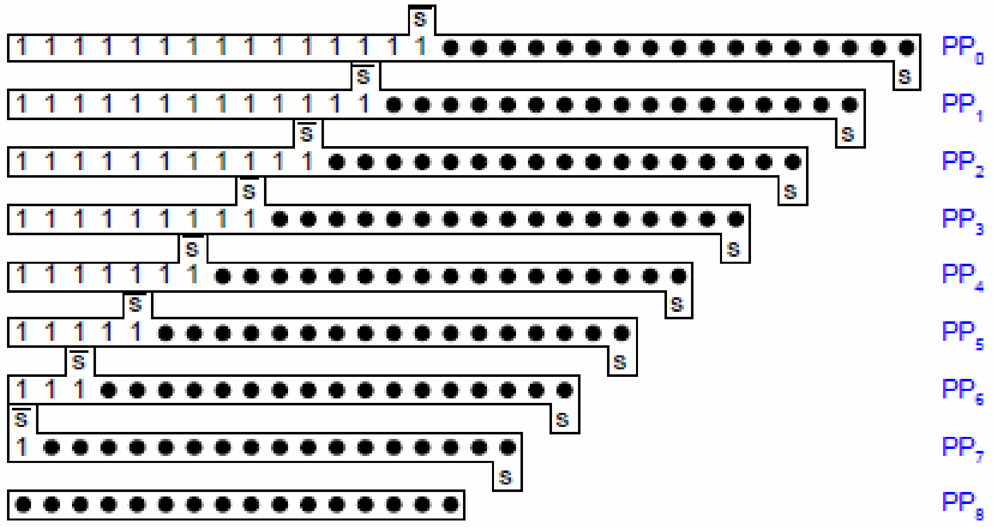
Sign Extension

- Partial products can be negative
  - Require sign extension, which is cumbersome
  - High fanout on most significant bit



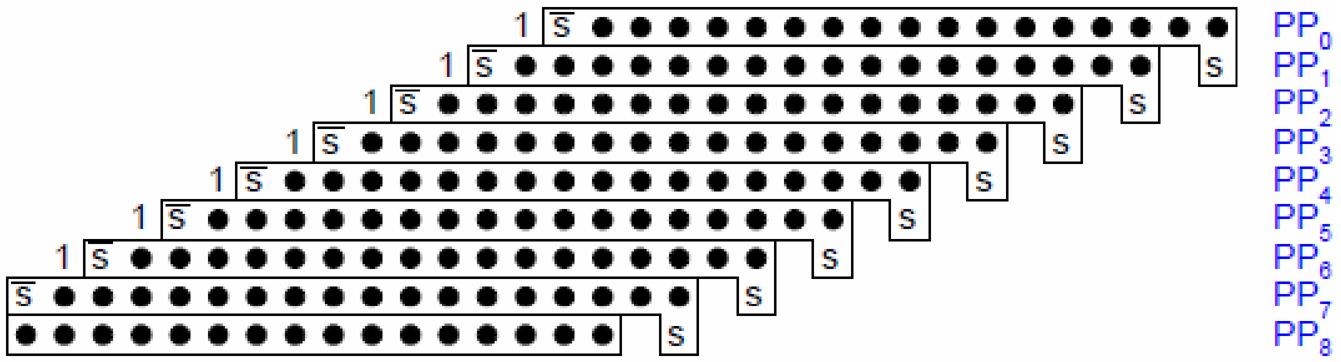
Simplified Sign Extension

- Sign bits are either all 0's or all 1's
  - Note that all 0's is all 1's + 1 in proper column
  - Use this to reduce loading on MSB



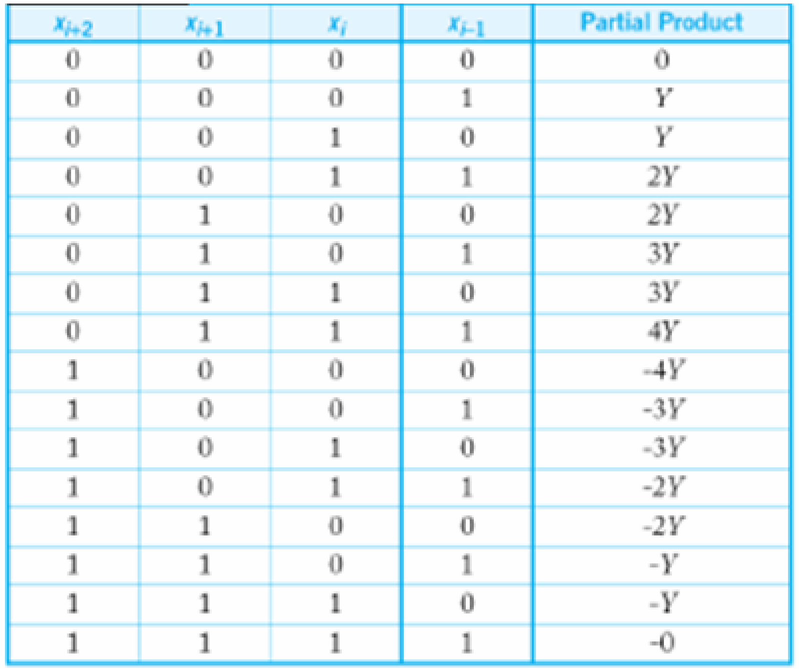
Even simpler sign extension

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



Higher Radix Booth Encoding

- Booth-8 reduces partial products by factor of 3
  - Requires +/- 3Y multiples which need a CPA



x_{i+2}	x_{i+1}	x_i	x_{i-1}	Partial Product
0	0	0	0	0
0	0	0	1	Y
0	0	1	0	Y
0	0	1	1	$2Y$
0	1	0	0	$2Y$
0	1	0	1	$3Y$
0	1	1	0	$3Y$
0	1	1	1	$4Y$
1	0	0	0	$-4Y$
1	0	0	1	$-3Y$
1	0	1	0	$-3Y$
1	0	1	1	$-2Y$
1	1	0	0	$-2Y$
1	1	0	1	$-Y$
1	1	1	0	$-Y$
1	1	1	1	-0

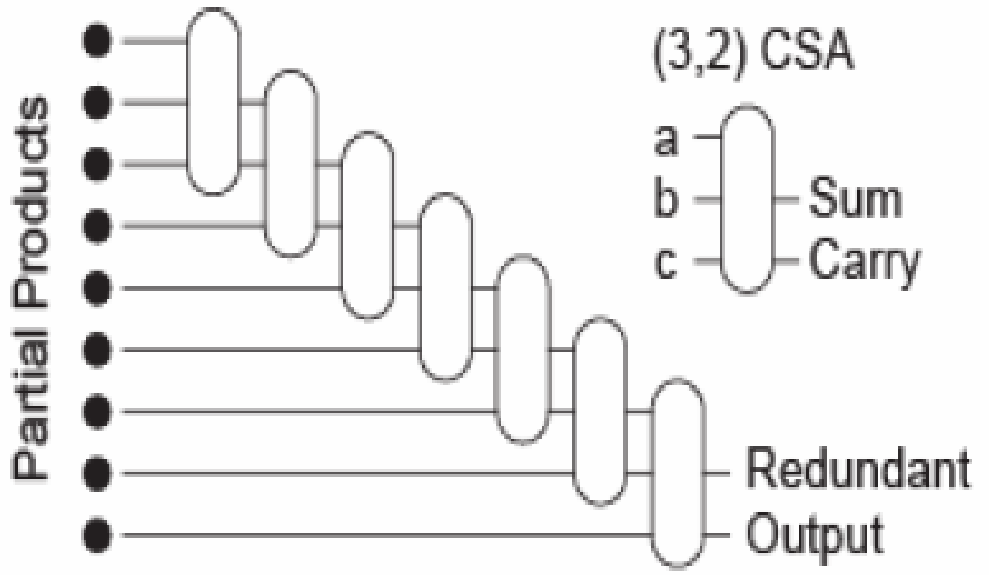
Column Addition

- Add up all the dots in a column in carry-save form
- Then use CPA to convert to nonredundant binary
- CSAs act as "1's counters"

A	B	C	Carry	Sum	Number of 1's
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	1
0	1	1	1	0	2
1	0	0	0	1	1
1	0	1	1	0	2
1	1	0	1	0	2
1	1	1	1	1	3

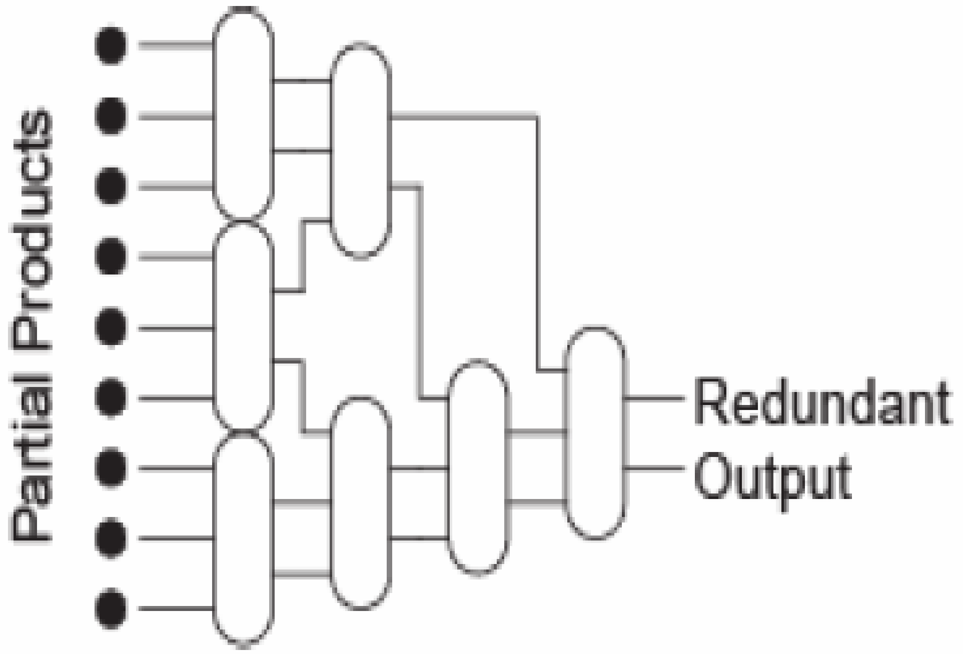
Array multiplier CSAs

- N-2 CSAs to sum N partial products
  - Ex: 16 x 16 Booth-4 multiplier with 9 PPs
  - Remember carries actually cross columns



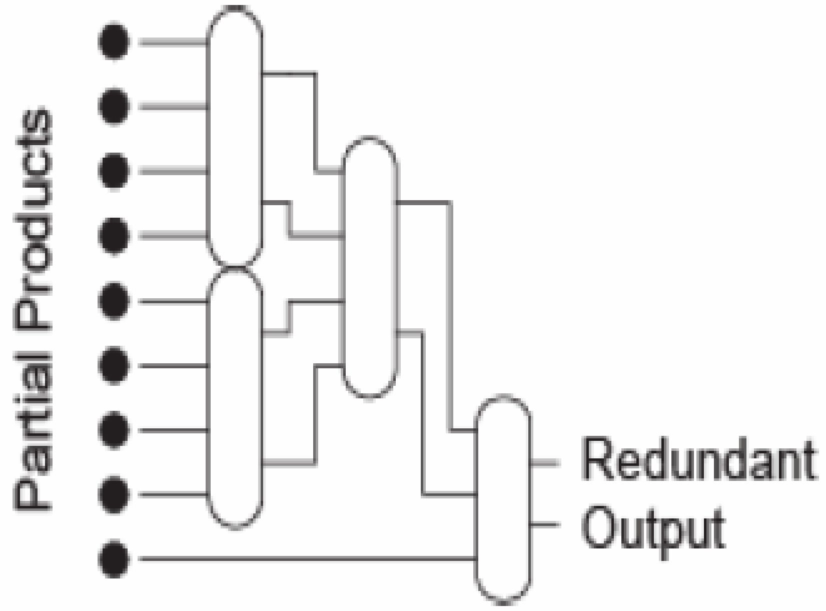
Wallace Tree

- Reduce number of levels using a tree
  - $\log_{3/2}N$ levels



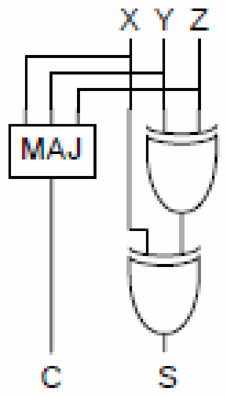
4:2 Compressor

- Wallace tree routing is irregular and long
  - Use 4:2 compressors instead

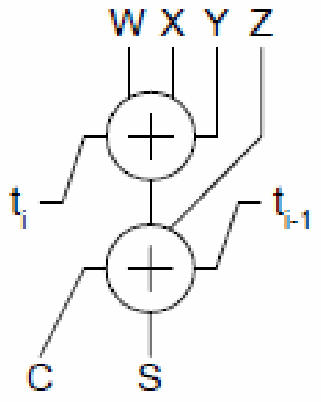


Compressor Design

- 3:2 CSA
  - 2 XOR delays
  - X is fast input
  - C is fast output

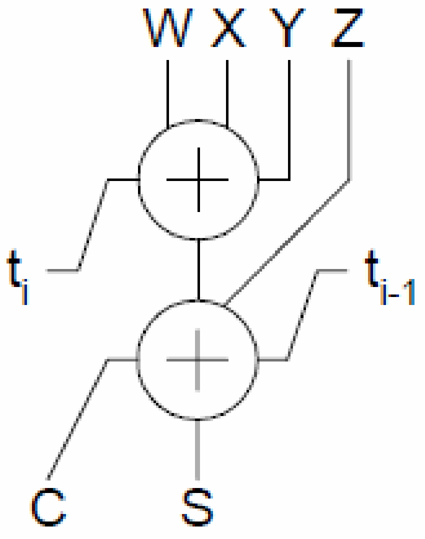


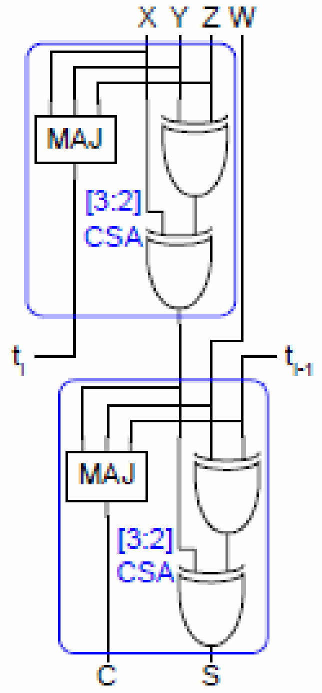
- 4:2 Compressor
  - 2 3:2 CSAs
  - Note horizontal carry t



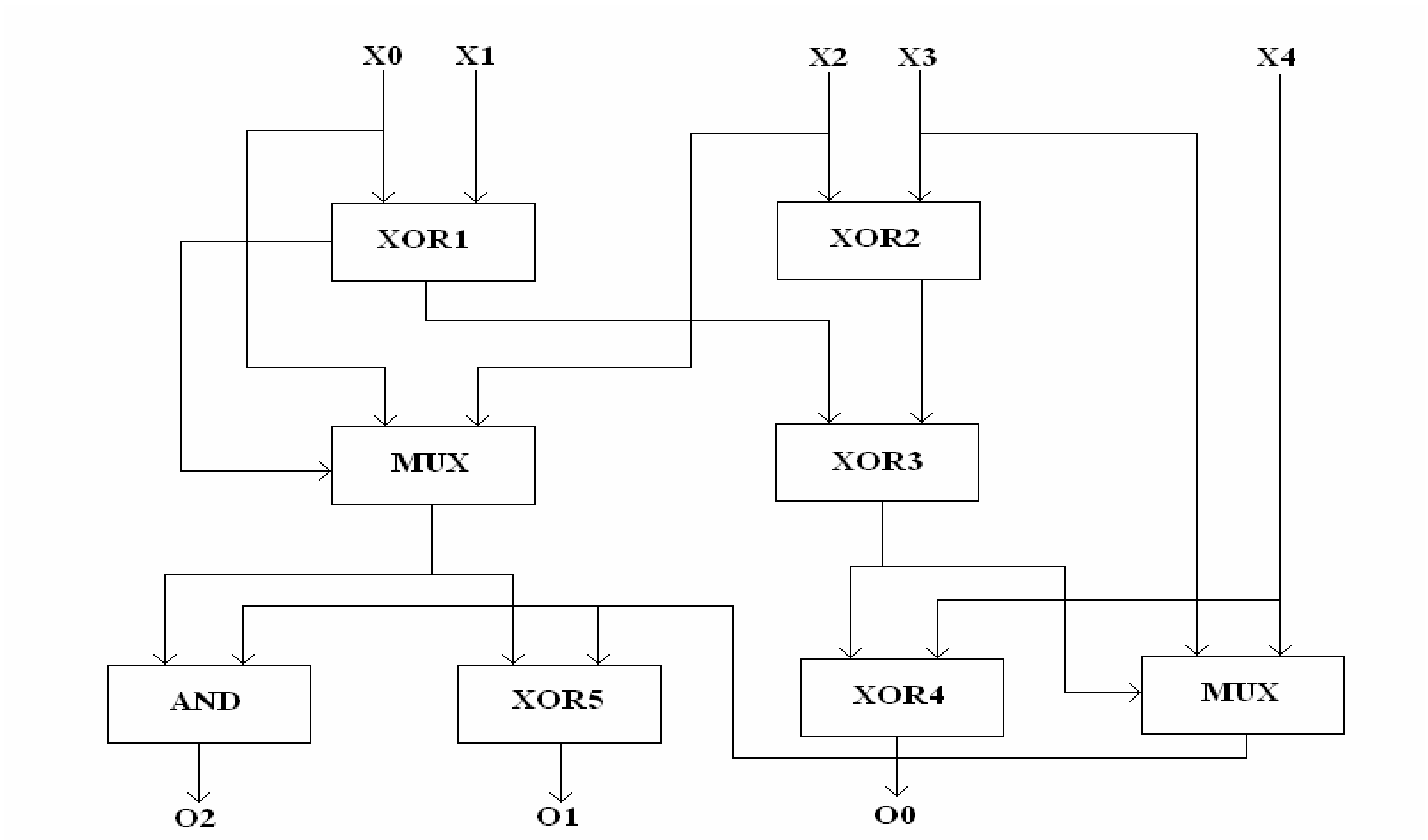
Improved 4:2 Compressor

- Arrange inputs carefully for only 3 XOR delays
  - Connect slow output to fast input



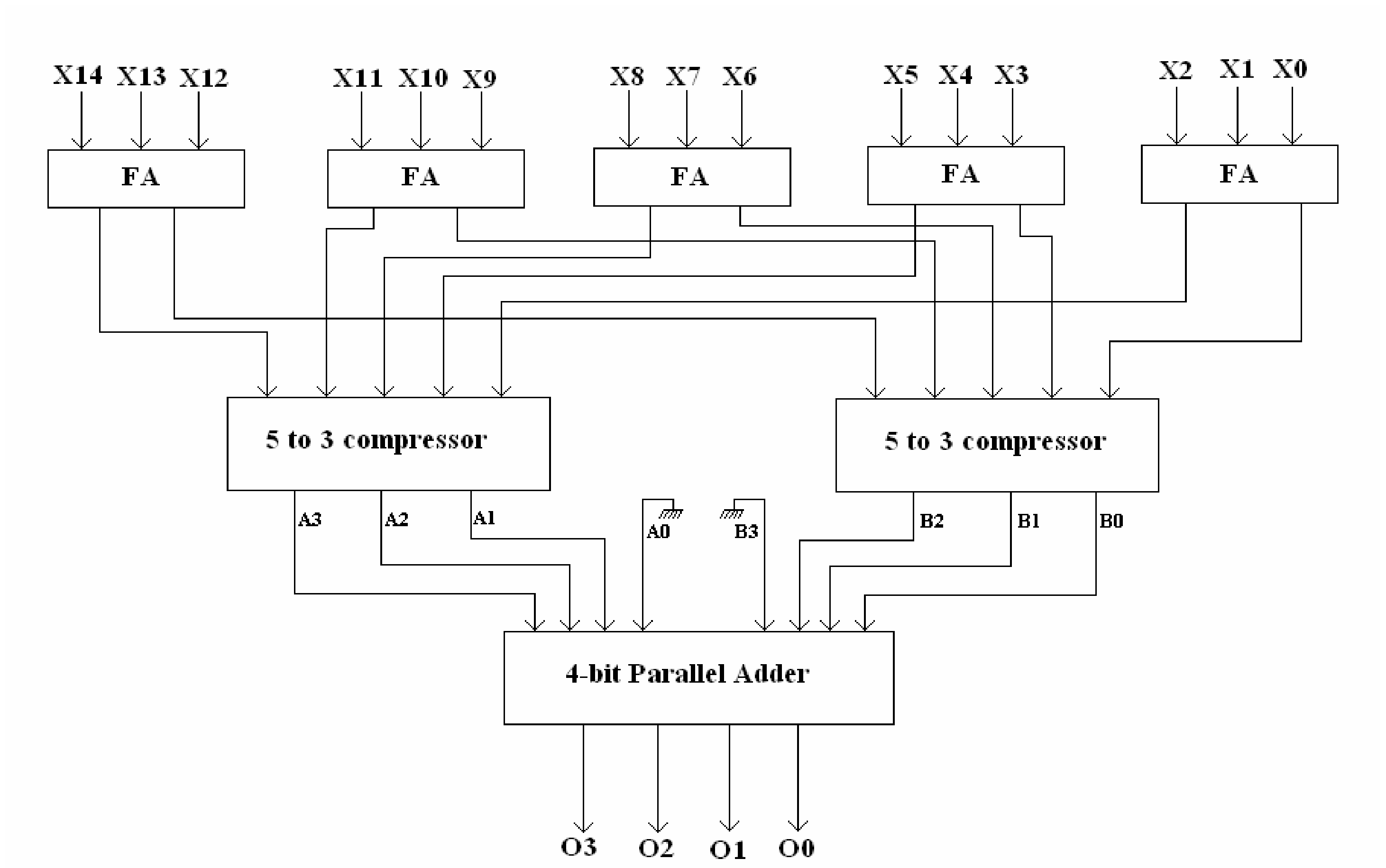


5:3 Compressor



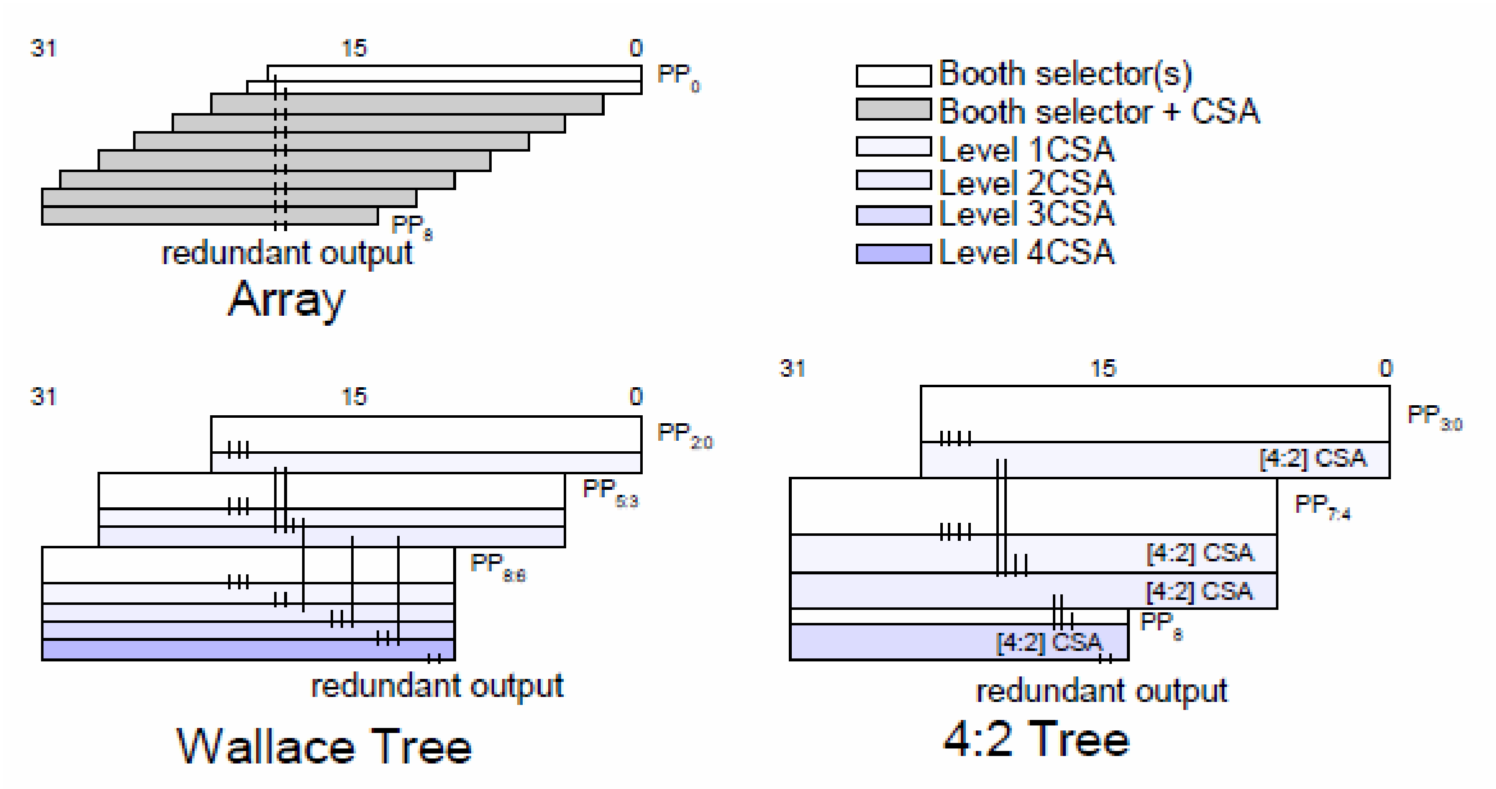
S. Roy Chowdhury, A. Banerjee, A. Roy, H. Saha, IEEE VDAT 2008

15:4 Compressor

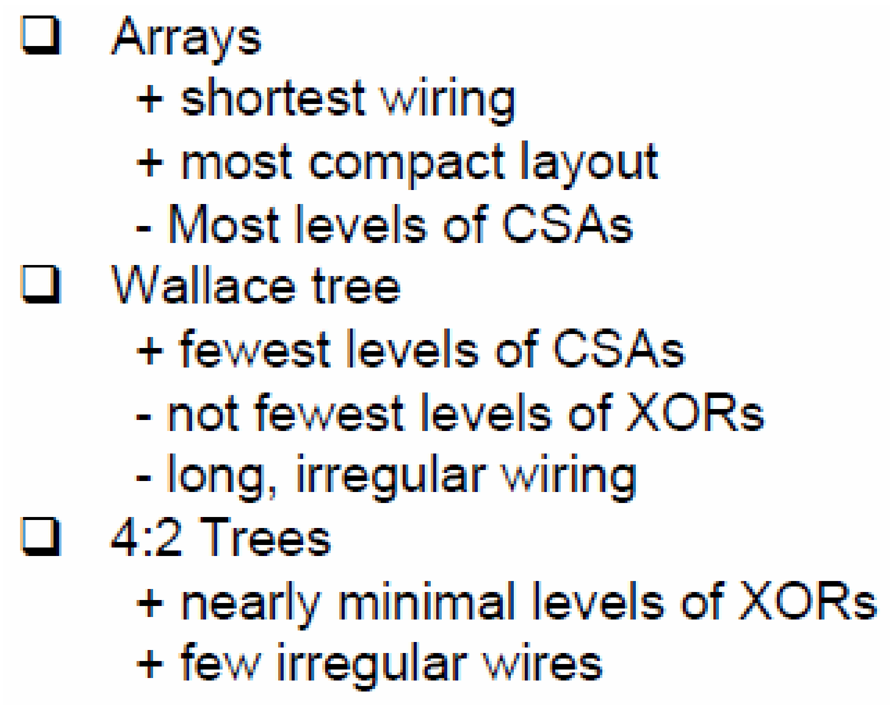


S. Roy Chowdhury, A. Banerjee, A. Roy, H. Saha, IEEE VDAT 2008

Multiplier Floorplans



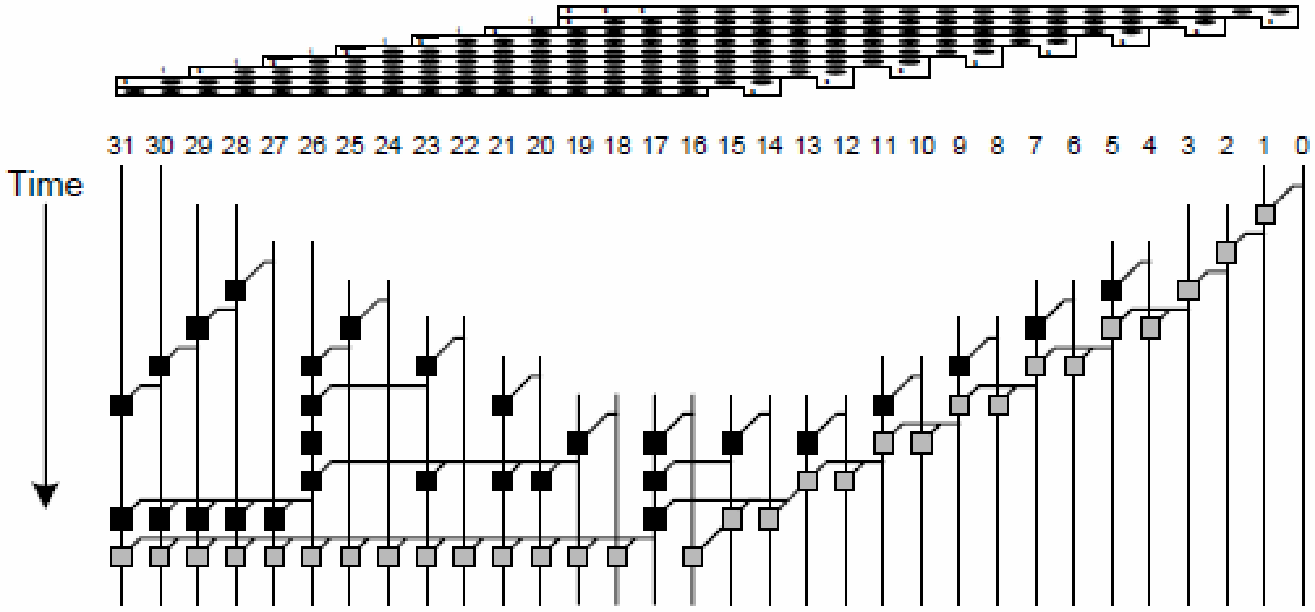
Comparison of different multiplier architectures



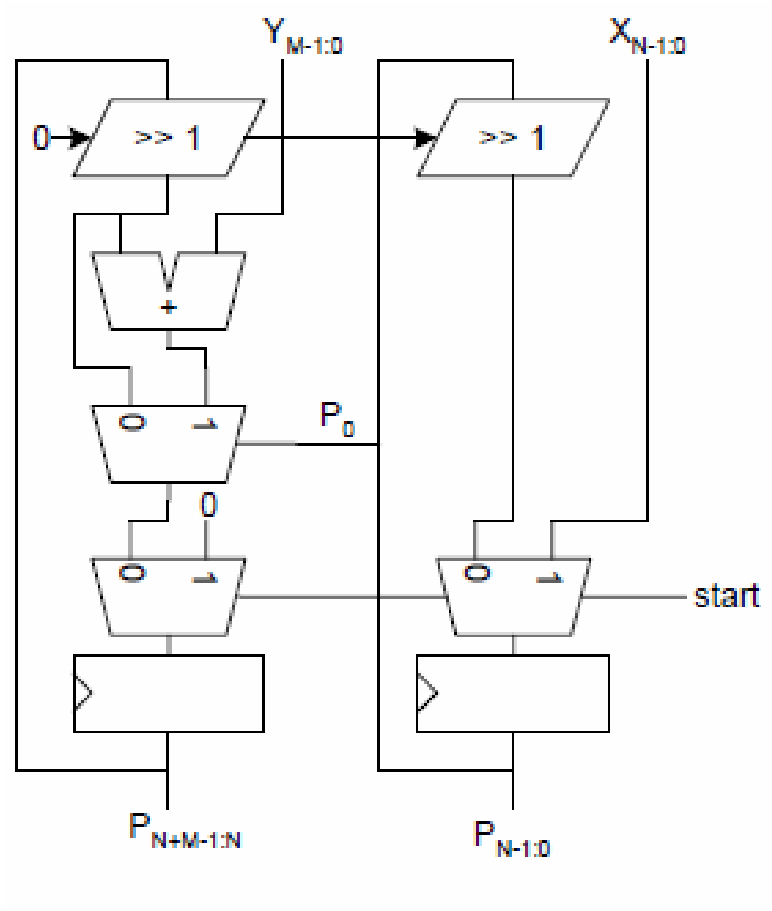
- ❑ Arrays
  - + shortest wiring
  - + most compact layout
  - - Most levels of CSAs
- ❑ Wallace tree
  - + fewest levels of CSAs
  - - not fewest levels of XORs
  - - long, irregular wiring
- ❑ 4:2 Trees
  - + nearly minimal levels of XORs
  - + few irregular wires

Final Addition

- Take advantage of irregular arrival times
  - First and last columns have fewer PPs
  - Reduce power of noncritical early inputs



Serial Multiplication

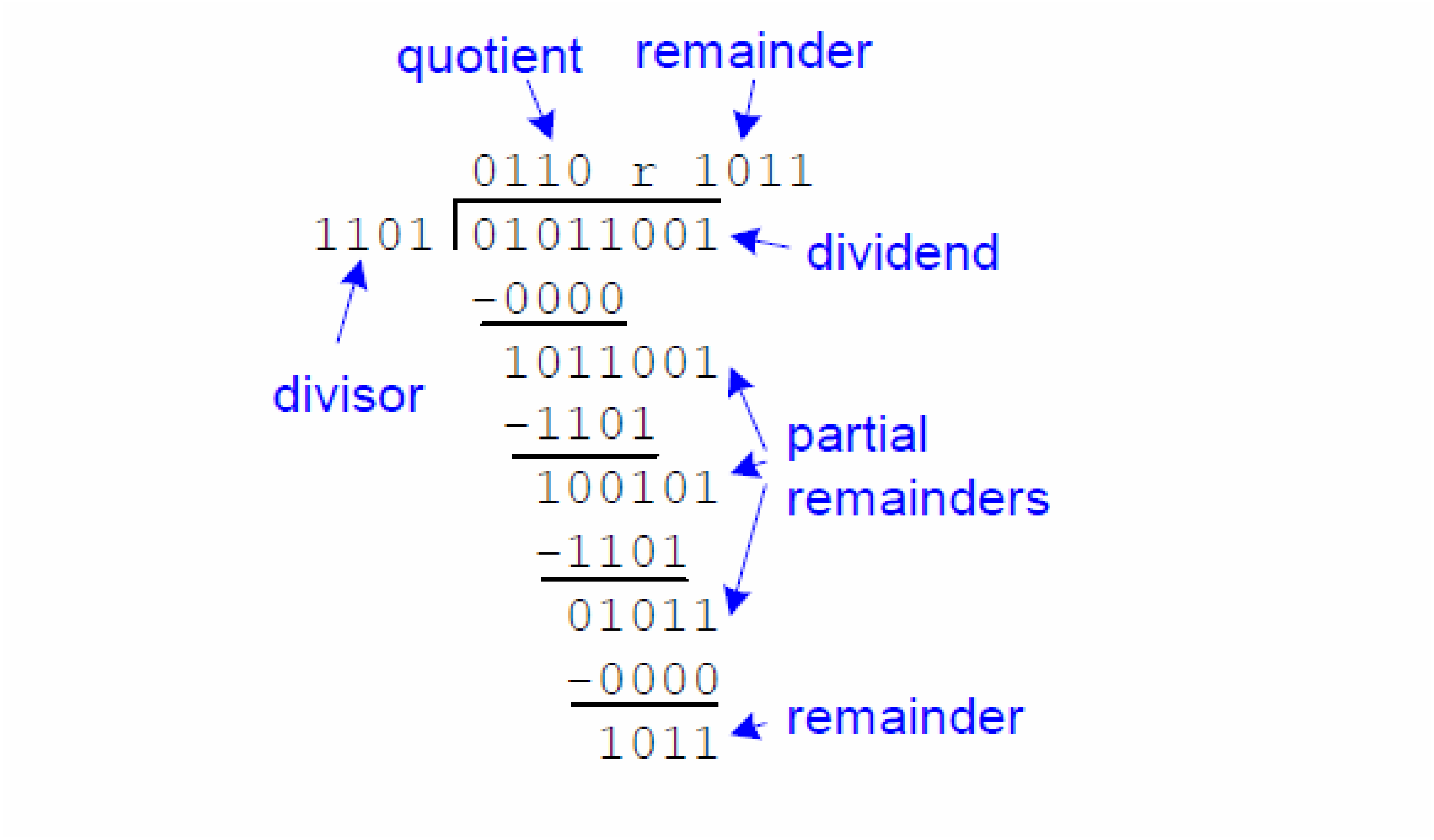


Step	Shift Reg	Notes
	0000|0101	initialize
0a	1100|0101	add 1*Y
0b	01100|010	shift right
1a	01100|010	add 0*Y
1b	001100|01	shift right
2a	111100|01	add 1*Y
2b	0111100|0	shift right
3a	0111100|0	add 0*Y
3b	00111100|	shift right

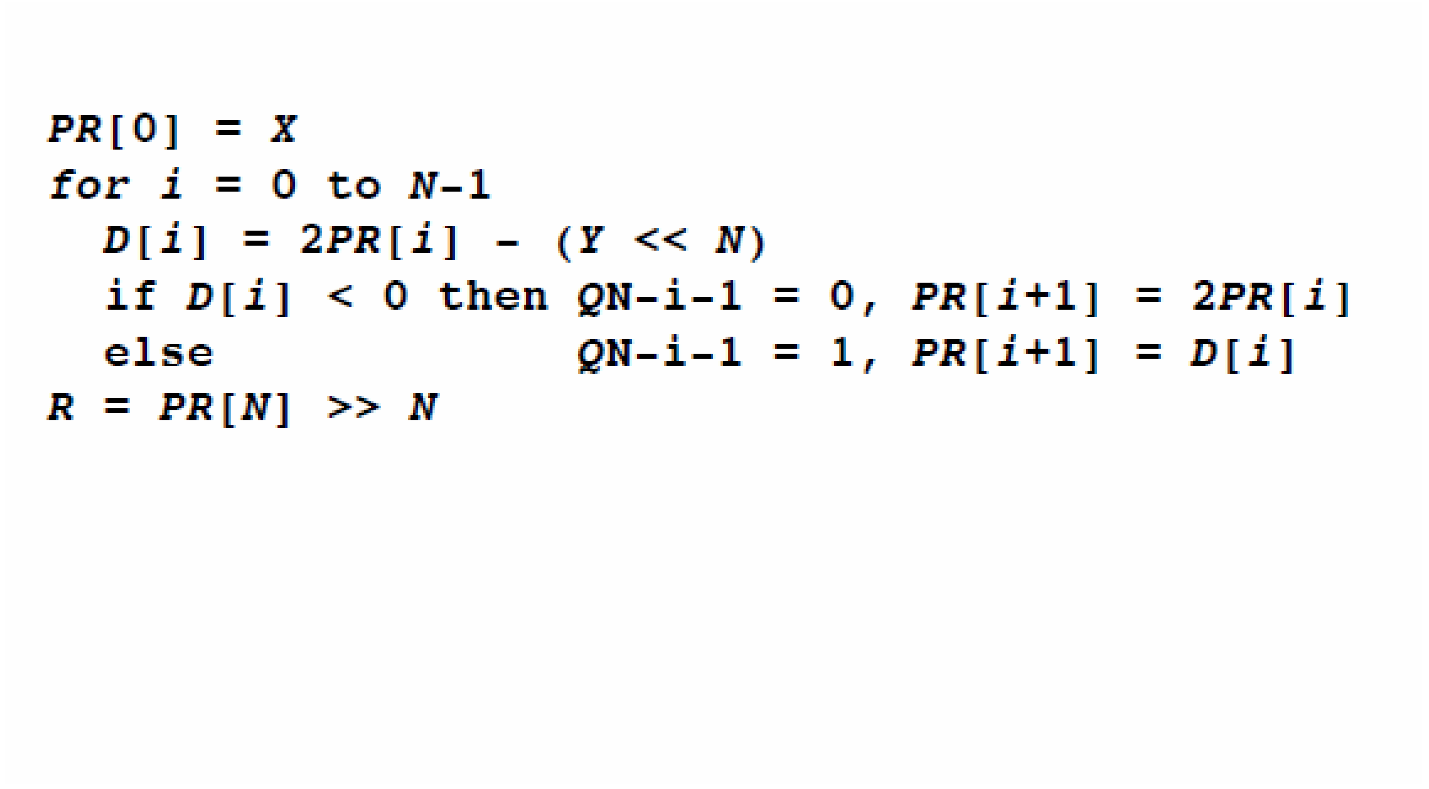
Division

- Compute X/Y = Q, R
  - Where X = Q Y + R
  - |R| < |Y|
- X: Dividend (M+N bits in range [0, $2^N Y$-1]
- Y: Divisor (M bits)
- Q: Quotient (N bits)
- R: Remainder (M bits)

Division Example

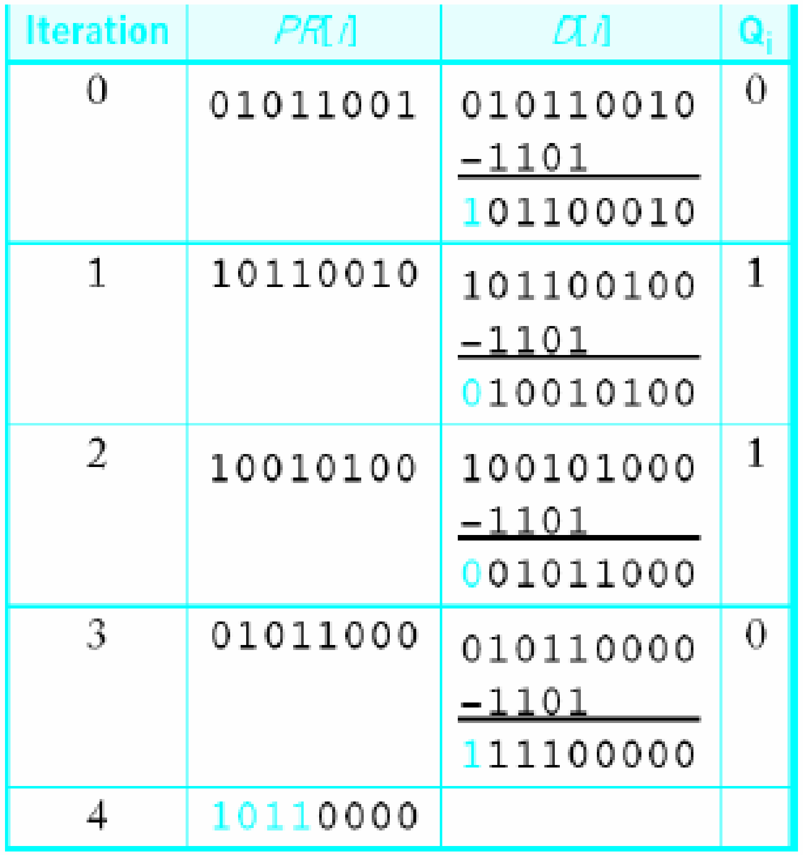


Division Algorithm



```
PR[0] = X
for i = 0 to N-1
  D[i] = 2PR[i] - (Y << N)
  if D[i] < 0 then QN-i-1 = 0, PR[i+1] = 2PR[i]
  else             QN-i-1 = 1, PR[i+1] = D[i]
R = PR[N] >> N
```

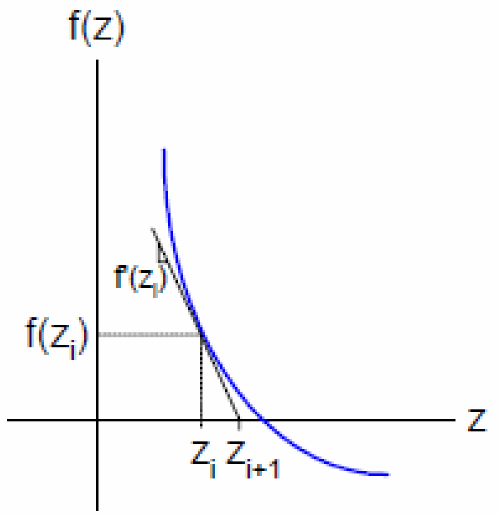
Division Example



Iteration	$PR[i]$	$D[i]$	Q_i
0	01011001	010110010 -1101 101100010	0
1	10110010	101100100 -1101 010010100	1
2	10010100	100101000 -1101 001011000	1
3	01011000	010110000 -1101 111100000	0
4	10110000		

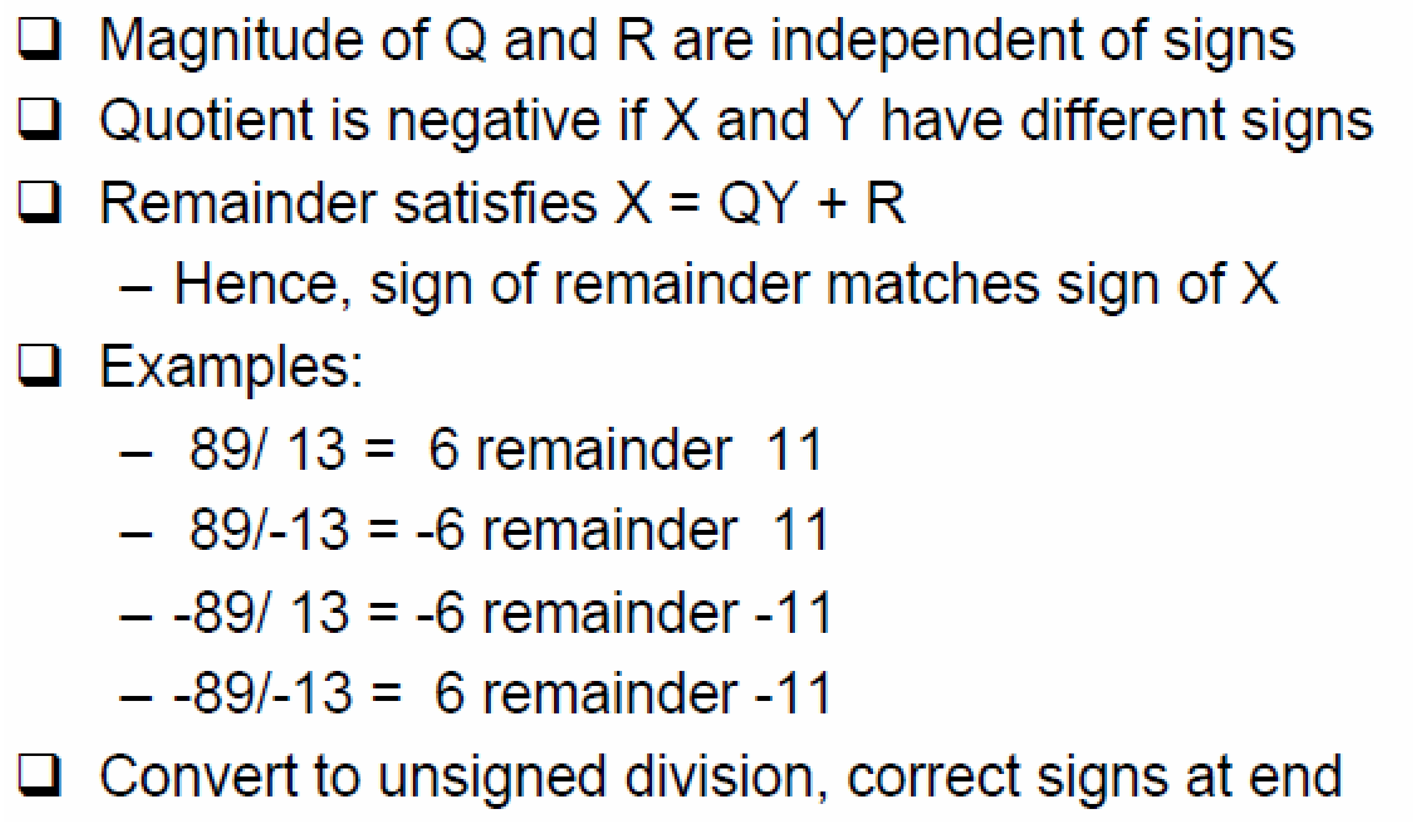
Iterative Approximation

- ❑ Make initial guess, then iteratively improve
- ❑ Common method: Newton-Ralphson
  - – Method of finding root of f(z)



$$z_{i+1} = z_i - \frac{f(z_i)}{f'(z_i)}$$

Signed Division



- Magnitude of Q and R are independent of signs
- Quotient is negative if X and Y have different signs
- Remainder satisfies X = QY + R
  - Hence, sign of remainder matches sign of X
- Examples:
  - 89/ 13 = 6 remainder 11
  - 89/-13 = -6 remainder 11
  - -89/ 13 = -6 remainder -11
  - -89/-13 = 6 remainder -11
- Convert to unsigned division, correct signs at end

Thank You