VLSI Architectures for Digital Signal Processoring

Dr Noor Mahammad Sk

High Performance Reconfigurable Computing System Engineering Group
Indian Institute of Information Technology Design and Manufacturing (IIITDM) Kancheepuram, Chennai – 600127.

High Performance Circuit Design

- High speed Adder Circuits
  - Carry Ripple – Inherently Sequential
  - Carry Look ahead – Parallel version
  - You should understand the conversion portion
- Multipliers
  - Wallace-tree multipliers

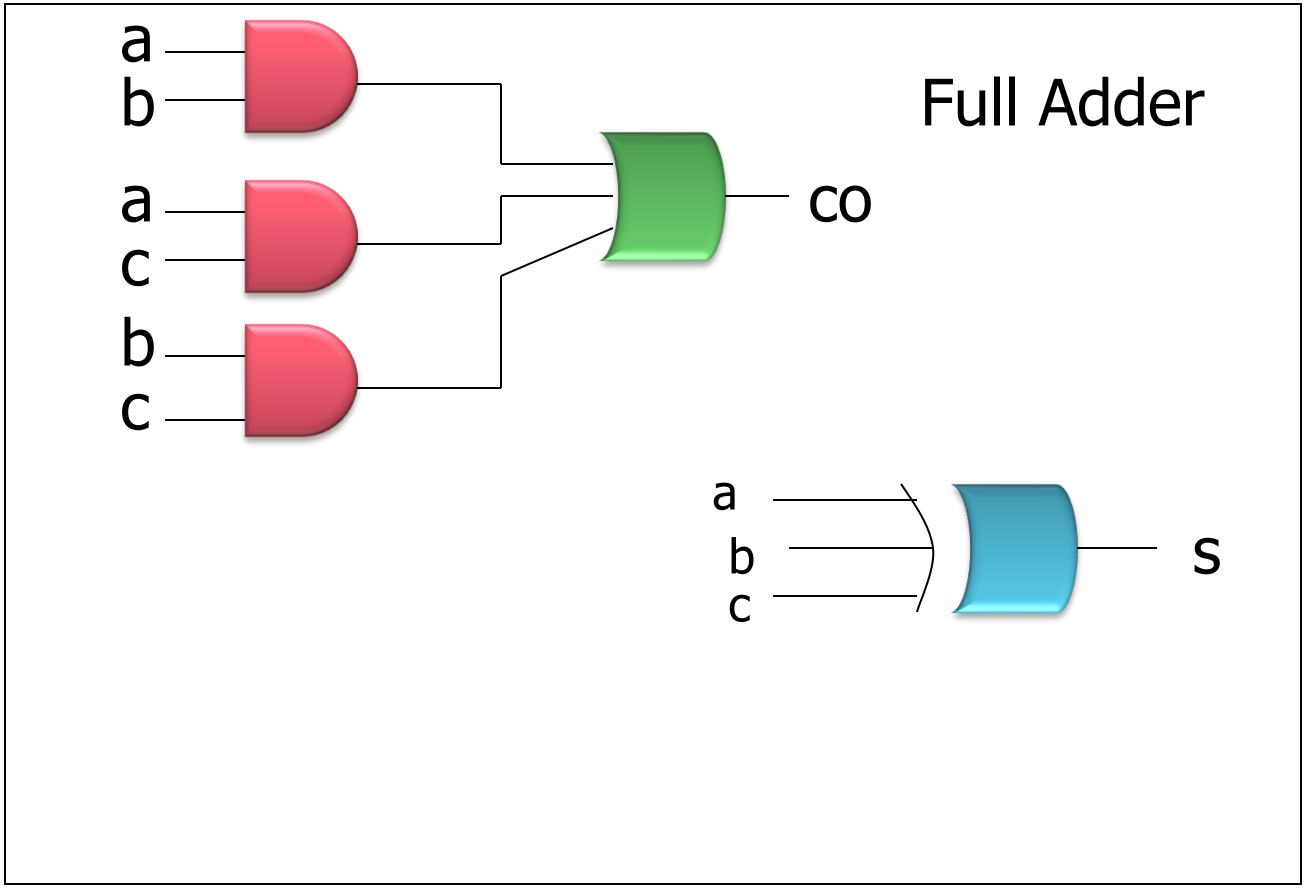
High Performance Circuit Design

- Performance of a circuit
  - Circuit depth – maximum level in the topological sort.
  - Circuit Size – Number of combinational elements.
- Optimize both for high performance.
- Both are inversely proportional – so a balance to be arrived.

Carry Ripple Adder

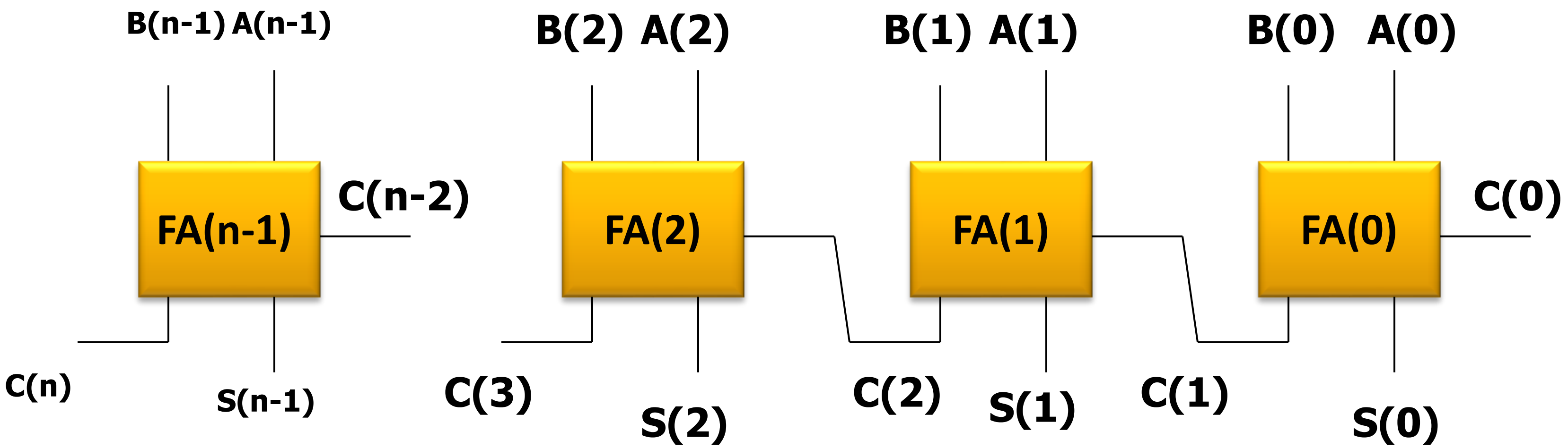
- Given two n-bit numbers
  - (a(n-1), a(n-2), a(n-3), …, a(0)) and
  - (b(n-1), b(n-2), b(n-3) ,…, b(0)).
- A full adder adds three bits (a, b, c), where ‘a’ and ‘b’ are data inputs and ‘c’ is the carry-in bit.
- It outputs a sum bit ‘s’ and a carry-out bit ‘co’

Full Adder



n-bit Carry Ripple Adder

- Circuit Depth is ‘n’.
- Circuit area is ‘n’ times size of a Full Adder



Carry Lookahead Adder

- The depth is 'n' because of the carry.
- Some interesting facts about carry

a(j)	b(j)	c(j)	c(j+1)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

If a(j) = b(j) then

c(j+1) = a(j) = b(j)

If a(j) <> b(j) then

c(j+1) = c(j)

Carry Lookahead Circuit

a(j)	b(j)	c(j)	Status x(j+1)
0	0	0	Kill (k)
0	0	1	
0	1	c(j) = 0	Propagate (p)
0	1	c(j) = 1	
1	0	c(j) = 0	Propagate (p)
1	0	c(j) = 1	
1	1	0	Generate (g)
1	1	1	

Carry Lookahead Circuit

x(j+1)

(*)	k	p	g
k	k	k	g
p	k	p	g
g	k	g	g

x(j)

New (j+1)th carry status as influenced by x(j)

(*) is associative

y(j) = x(0) (*) x(1) (*) … x(j)

x(0) = k

If y(j) = k then c(j) = 0

If y(j) = g then c(j) = 1

Note that y(j) <> p

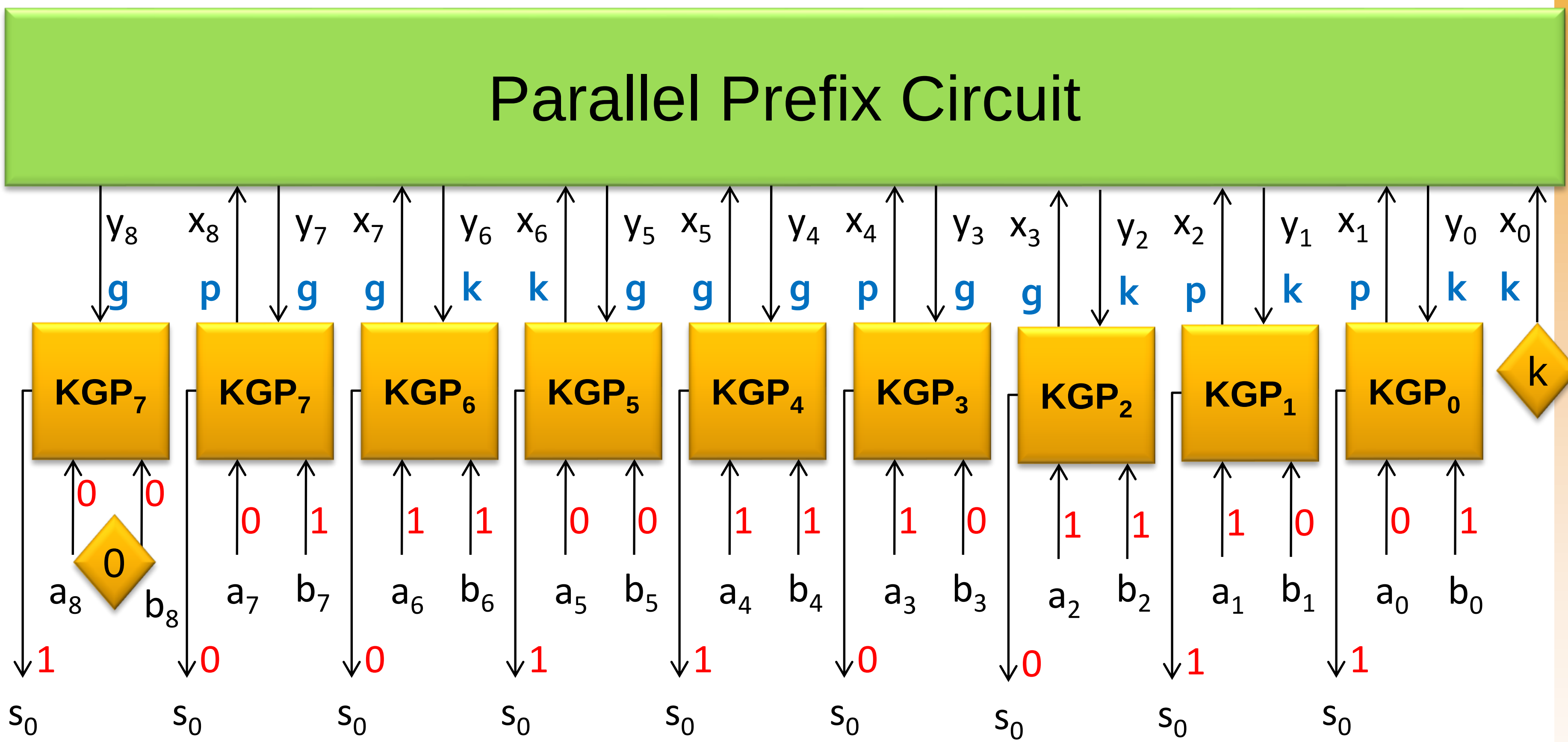
Carry Calculations

- A prefix computation
  - y(0) = x(0) = k
  - y(1) = x(0) (*) x(1)
  - y(2) = x(0) (*) x(1) (*) x(2)
  - …….
  - y(n) = x(0) (*) x(1) (*) …. x(n)
- Let [i,j] = x(i) (*) x(i+1) (*) … x(j)
- [i,j] (*) [j+1,k] = [i,k]
  - By associative property

Parallel Prefix Circuit

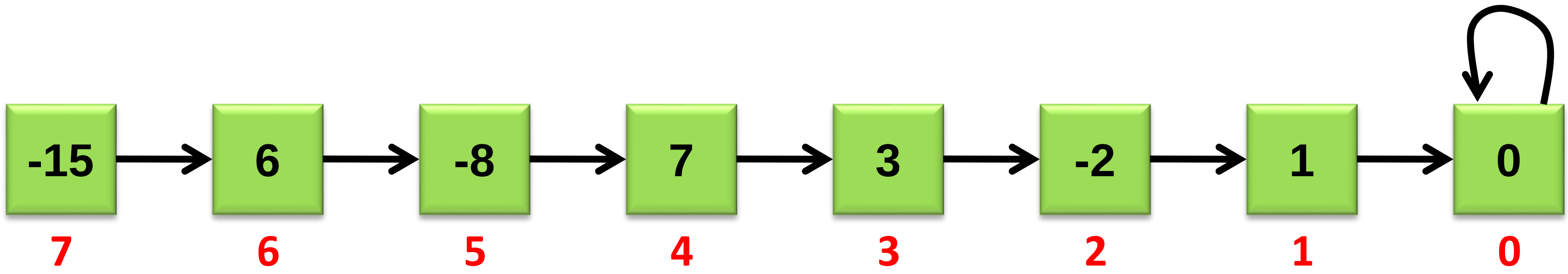
- Input x(0), x(1), … x(n) – for an n-bit CLA, where x(0) = k.
- Each x(i) is a 2-bit vector
- To compute the prefix (*) and pipeline the same.
- y(i) = x(0) (*) x(1) (*) … x(i)
- We can use the Recursive Doubling Technique described as follows

An 8-node Carry Lookahead Adder



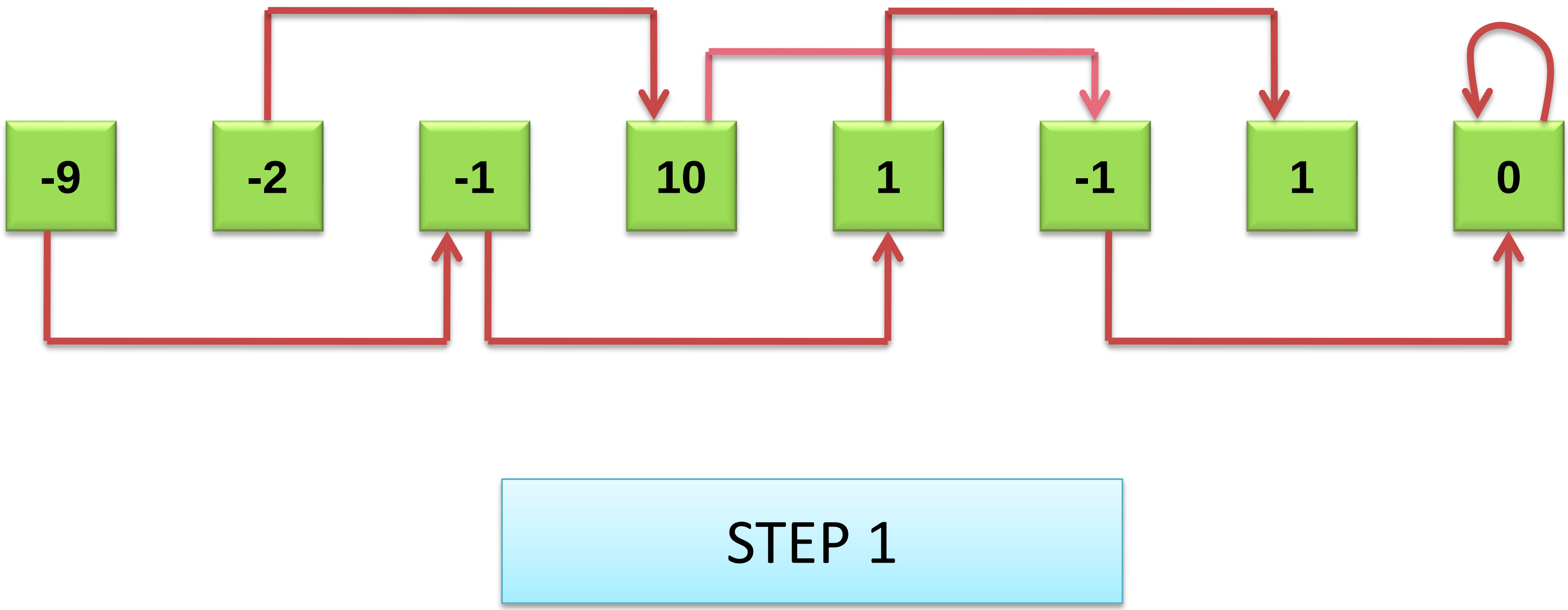
Parallel Techniques: Recursive Doubling

- Finding Prefix sum of ‘8’ numbers

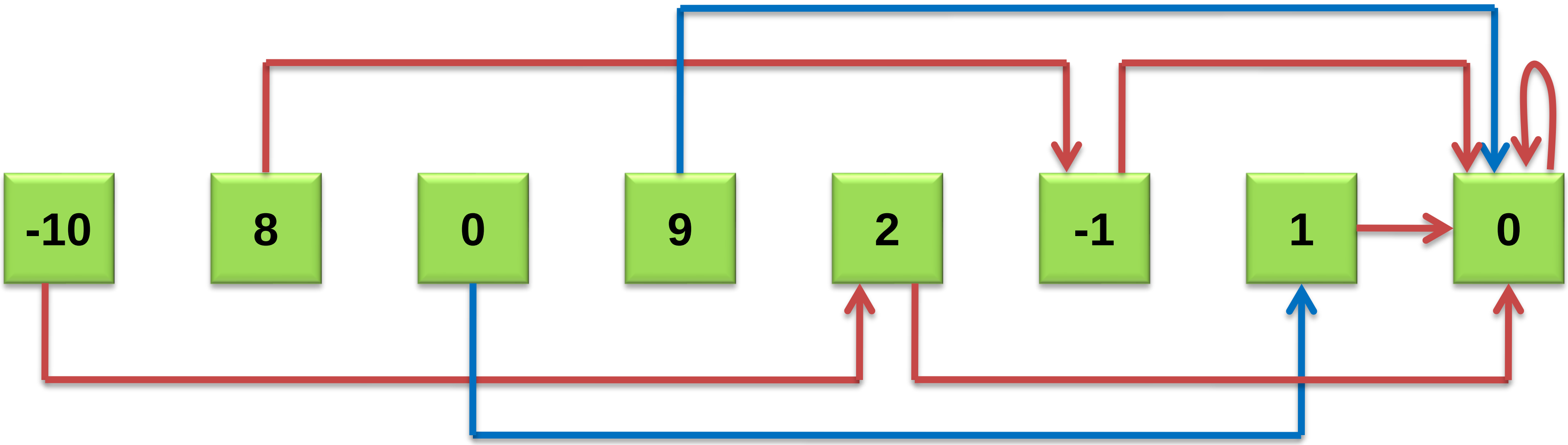


Recursive Doubling (Step 1)

- Finding Prefix sum of ‘8’ numbers

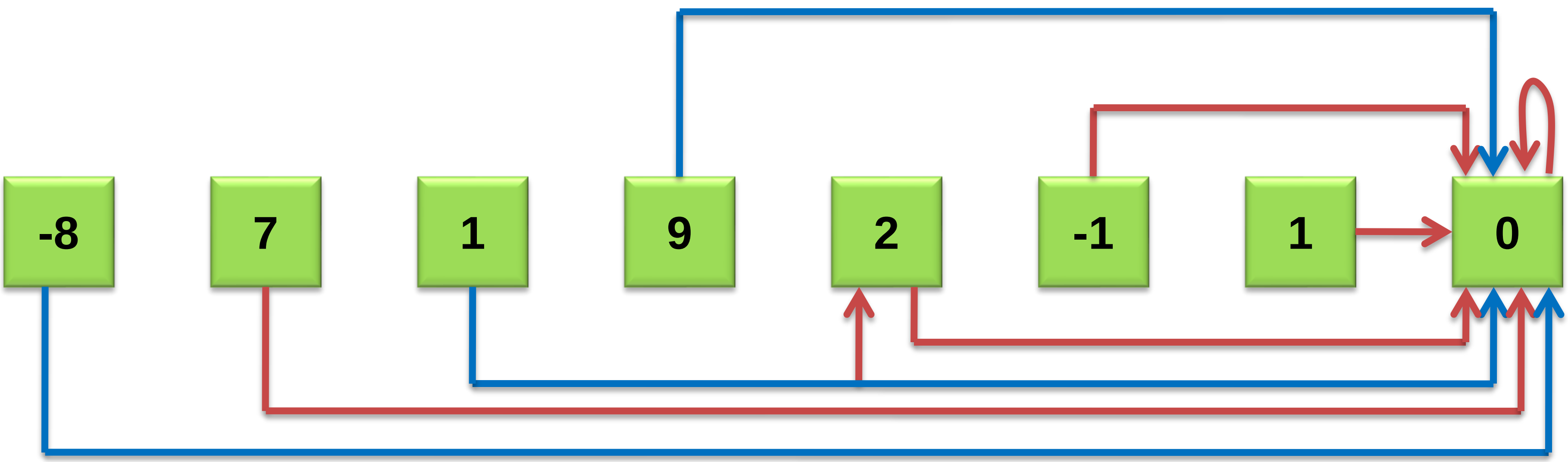


Recursive Doubling (Step 2)



STEP 2

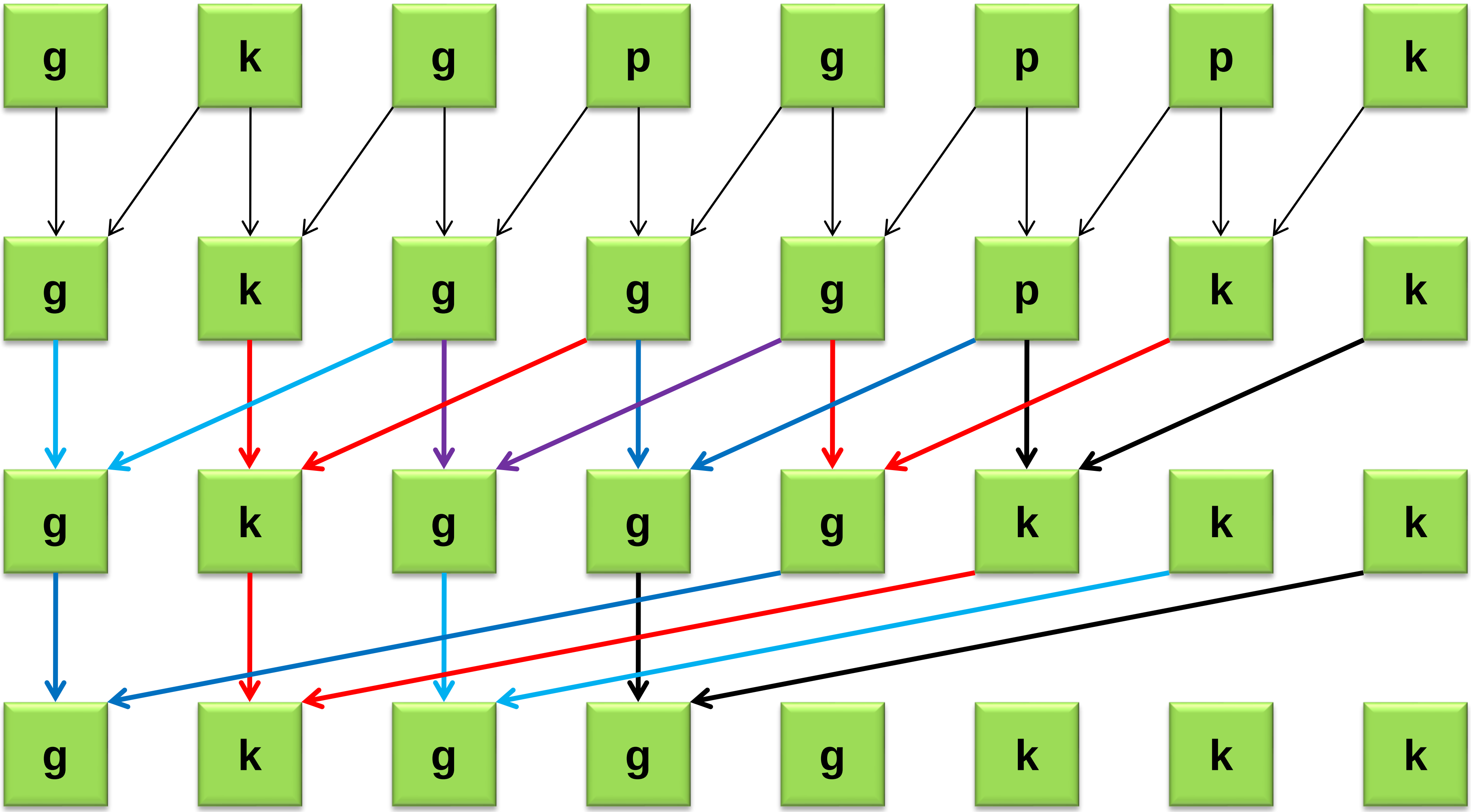
Recursive Doubling (Step 3)



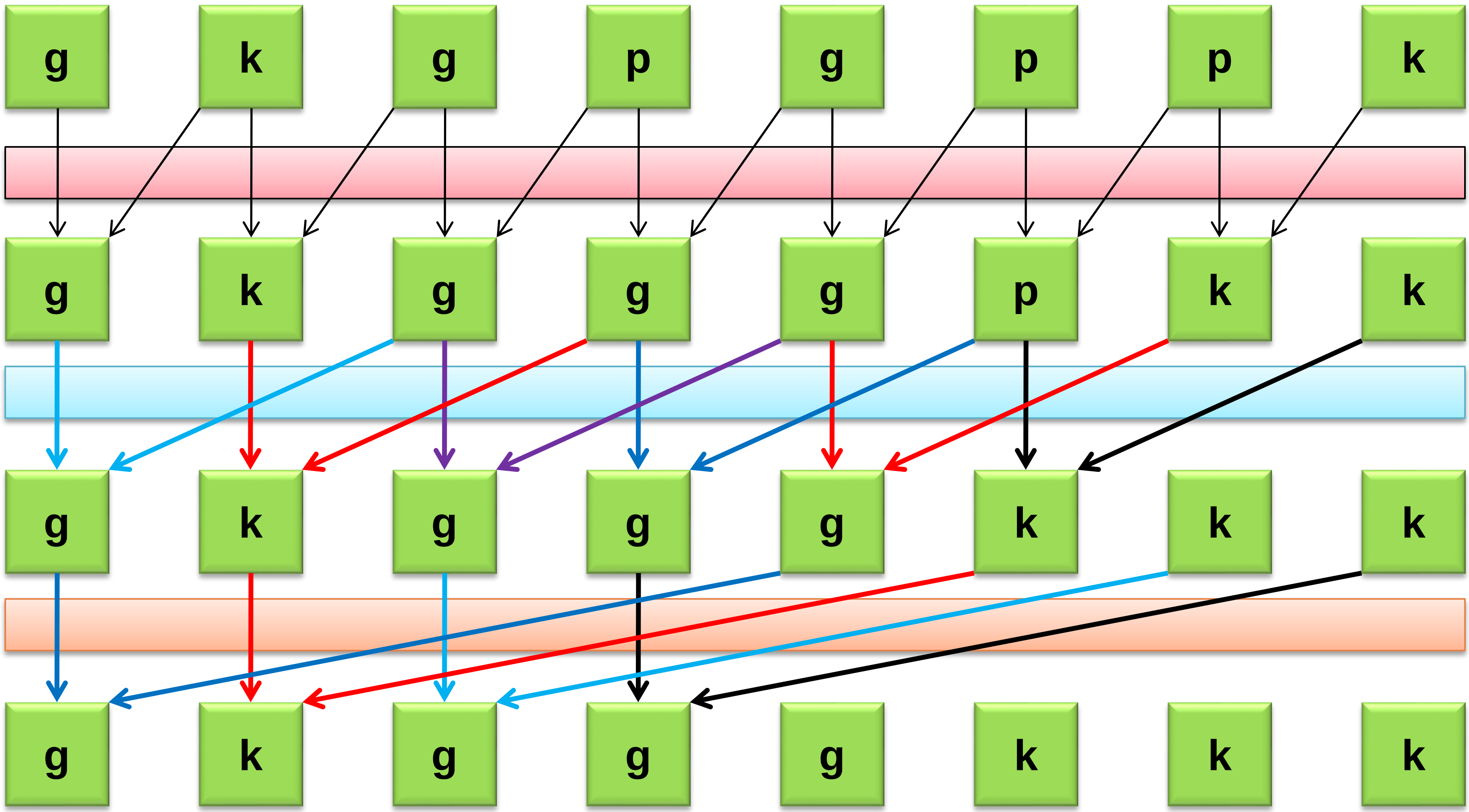
STEP 3

- Prefix Sum of n numbers in log n steps
- Recursive Doubling is applicable:
  - Operators like Min, Max, Mul etc. that is associative

Prefix Calculation Based on Recursive Doubling Technique



Pipeline Prefix Calculation Based on Recursive Doubling Technique



Outcome of Above Method

- Depth of the circuit reduced from O(n) to O($\log_2$ n)
- Size is still O(n)
- This results in a Fast Adder

Multipliers

- Simple grade-school multiplication method.
  - Concept of partial-products
- Partial products generated in parallel and carry save addition results in faster array multiplier

Grade School Multiplication

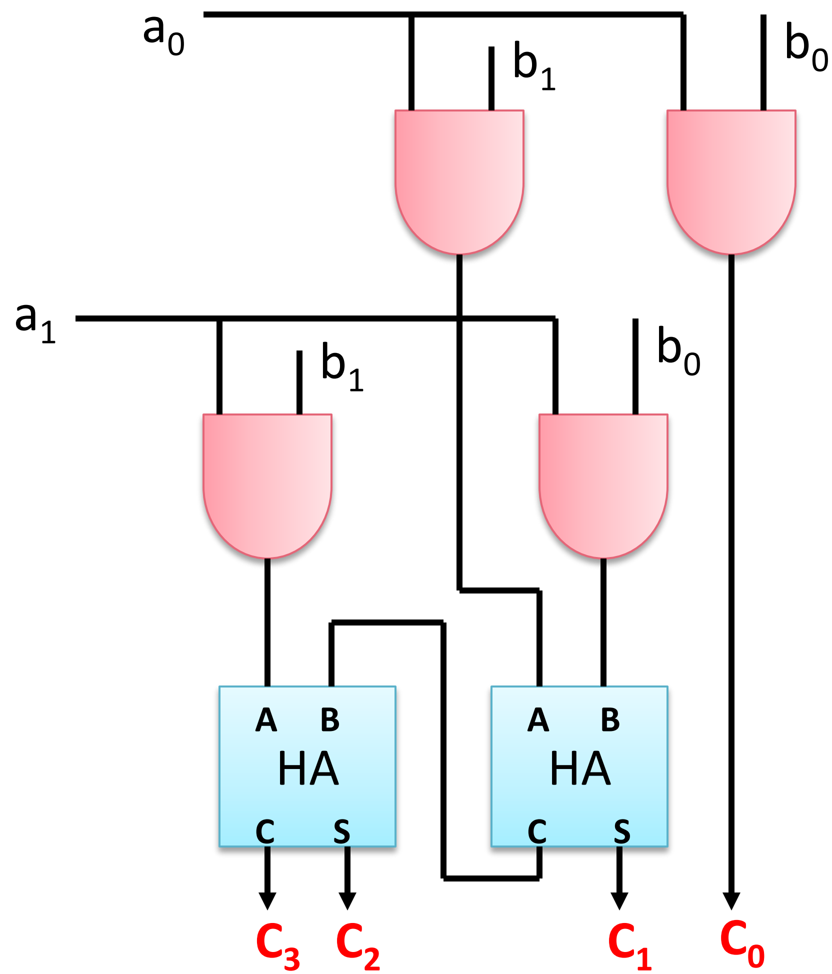
```
        1 1 1 0 = a
        1 1 0 1 = b
------------------------------
        1 1 1 0 = m(0)
      0 0 0 0   = m(1)
    1 1 1 0     = m(2)
  1 1 1 0       = m(3)
-------------------------------
 1 0 1 1 0 1 1 0 = p
```

Array Multiplier

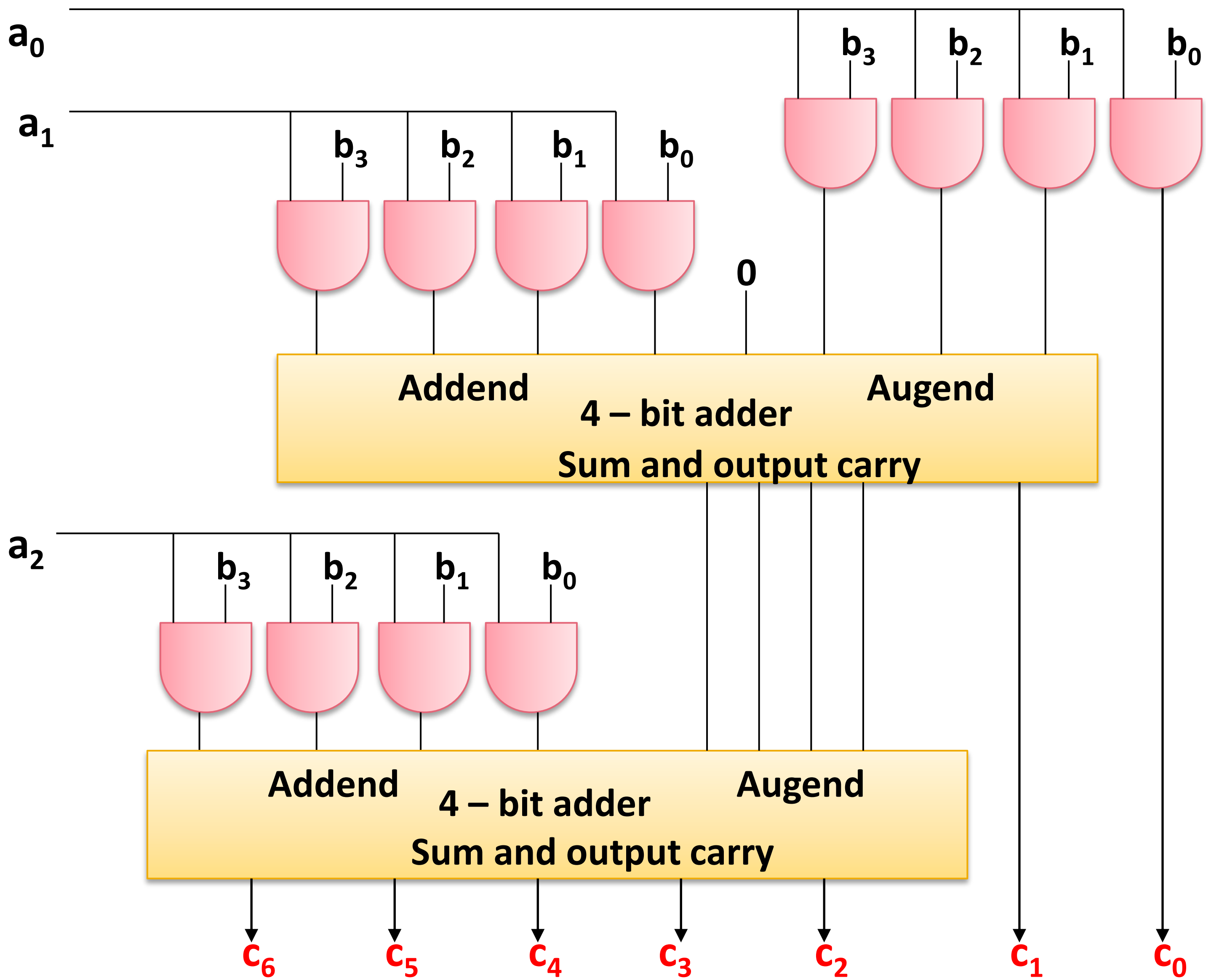
- Combination circuit
- Product generated in one micro-operation
- Requires large number of gates
- Became feasible after integrated circuits developed
- Needed for 'j' multiplier and 'k' multiplicand bits
  - j x k – AND gates
  - j – 1 k-bit adders to produce product of j +k bits

2-bit by 2-bit Array Multiplier

		b_1	b_0
		a_1	a_0
		a_0b_1	a_0b_0
	a_1b_1	a_1b_0	
c_3	c_2	c_1	c_0



4–bit by 3-bit array multiplier



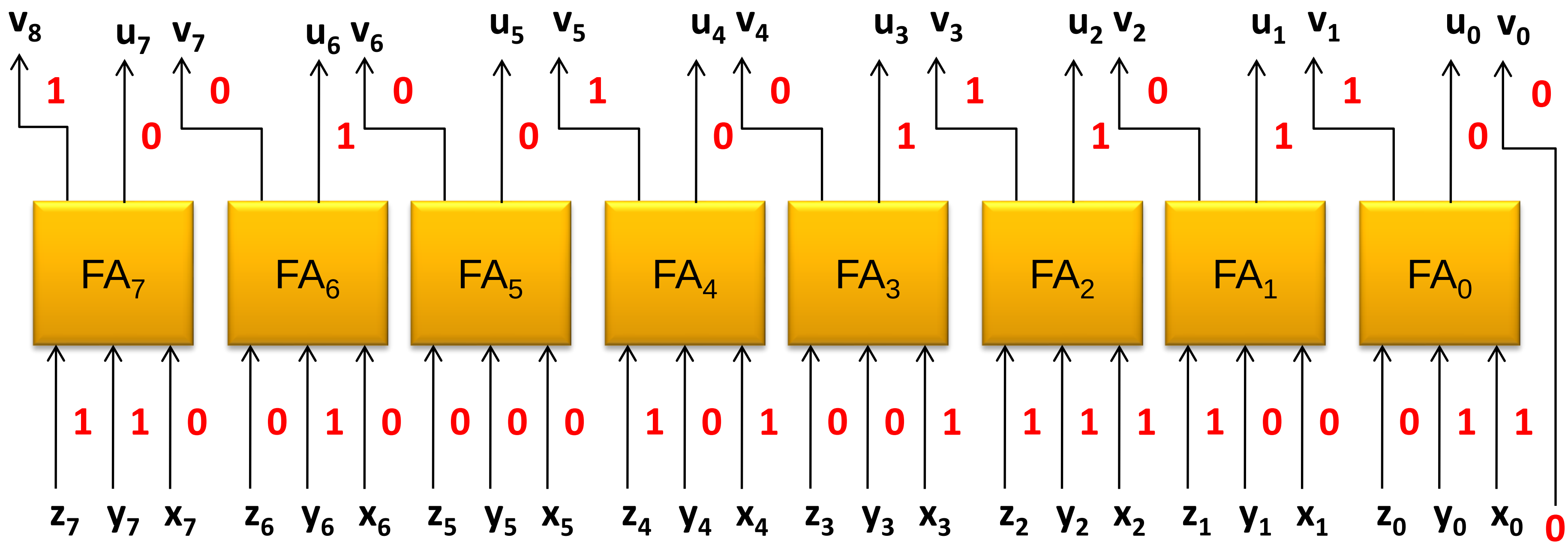
Carry Save Addition

- Given three n-bit numbers x, y and z.
- The circuit computes a n-bit number ‘u’ and a (n+1)-bit number ‘v’ such that
  - $x+y+z = u + v$

Carry Save Addition Example

8	7	6	5	4	3	2	1	0		i
	0	0	0	1	1	1	0	1	=	x
	1	1	0	0	0	1	0	1	=	y
	1	0	0	1	0	1	1	0	=	z
	0	1	0	0	1	1	1	0	=	u
1	0	0	1	0	1	0	1	0	=	v

Carry Save Adder Circuit



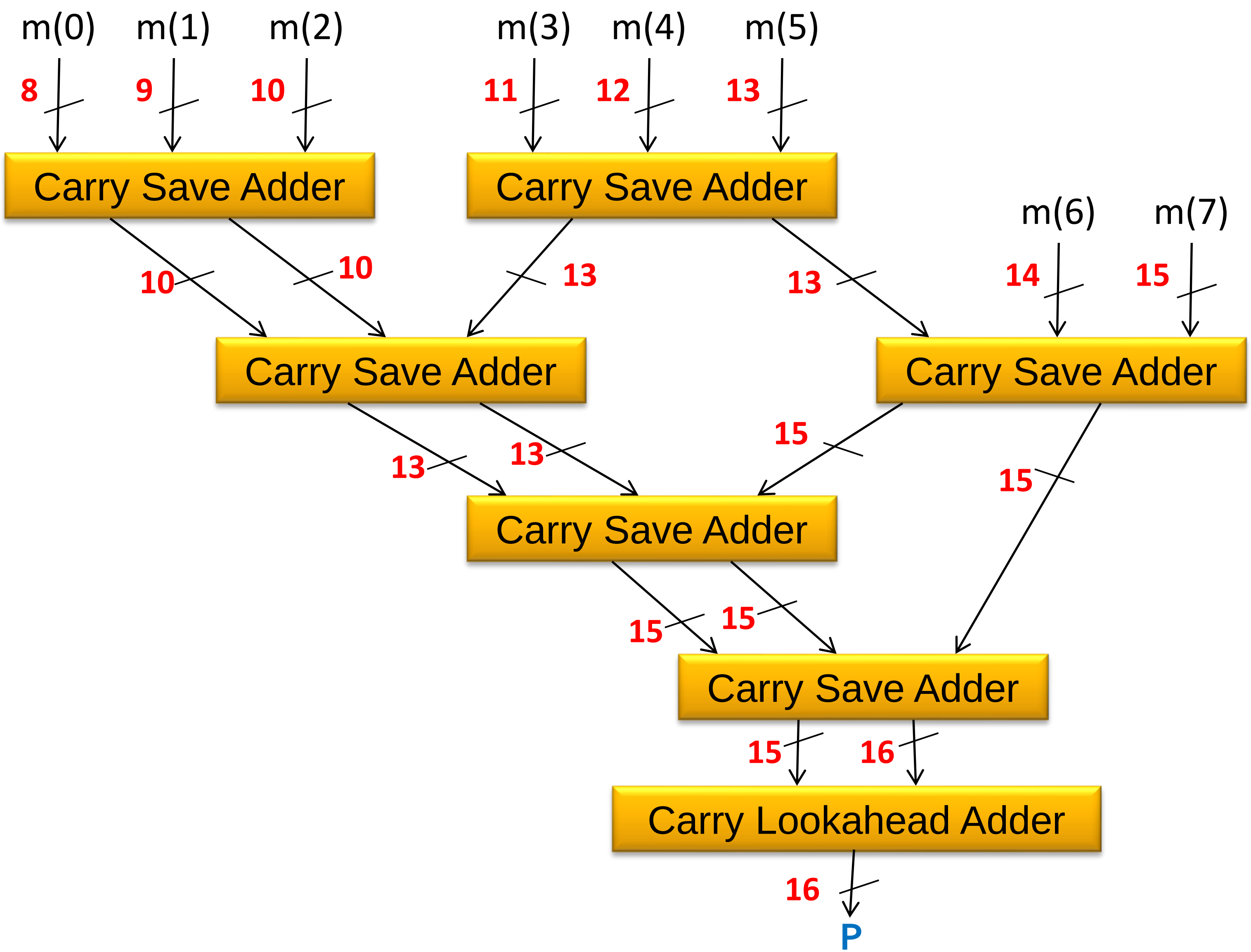
Carry Save Addition based Multiplication

				0	0	0	0	=	0
				1	1	1	0	=	m(0)
			0	0	0	0		=	m(1)
			0	1	1	1	0	=	u(1)
			0	0	0			=	v(1)
		1	1	1	0			=	m(2)
		1	1	0	1	1	0	=	u(2)
		0	1	0					v(2)
	1	1	1	0				=	m(3)
	1	0	1	0	1	1	0	=	u(3)
	1	1	0					=	v(3)
1	0	1	1	0	1	1	0	=	p

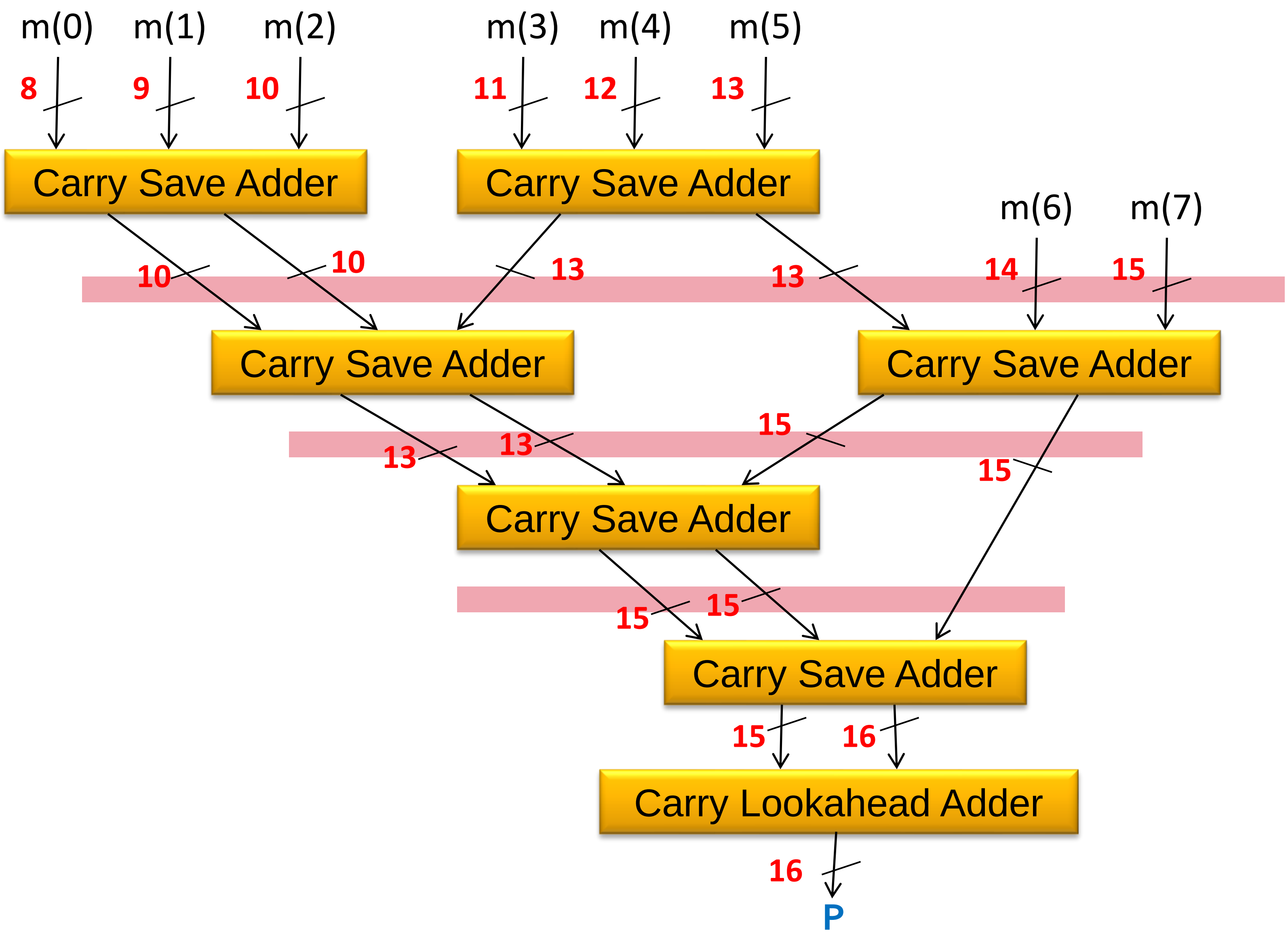
Wallace Tree Multipliers

- While multiplying two n-bit numbers a total of n partial products have to be added.
- Use floor(n/3) carry save adders and reduce the number to ceil(2n/3).
- Apply it recursively.
- O(log n) depth circuit of size $O(n^2)$.

8-bit Wallace Tree Multiplication



8-bit Wallace Tree Multiplication



Floating Point Arithmetic

Dr Noor Mahammad Sk

Floating Point Numbers

- $B = b_0 . b_{-1} b_{-2} \ldots b_{-(n-1)}$
- $F(B) = -b_0 \times 2^0 + b_{-1} \times 2^{-1} + b_{-2} \times 2^{-2} + \ldots + b_{-(n-1)} \times 2^{-(n-1)}$
- IEEE Standards for floating point numbers
  - $\pm X_1 . X_2 X_3 X_4 X_5 X_6 X_7 \times 10^{\pm Y1Y2}$
  - Single Precision (32 – bits)
    - Value represented = $\pm 1. M \times 2^{E' - 127}$ $(E' = E + 127)$
    - Where E stands for Exponent
  - Double Precision (64 – bits)
    - Value represented = $\pm 1. M \times 2^{E' - 1023}$ $(E' = E + 1023)$

Single Precision

- Not Normalized
  - $0.0010110 \ldots \times 2^{9}$
- Normalized
  - $1.0110 \ldots \times 2^{6}$

32 bits

S	E'	M

Sign of Number
0 signifies +
1 signifies -

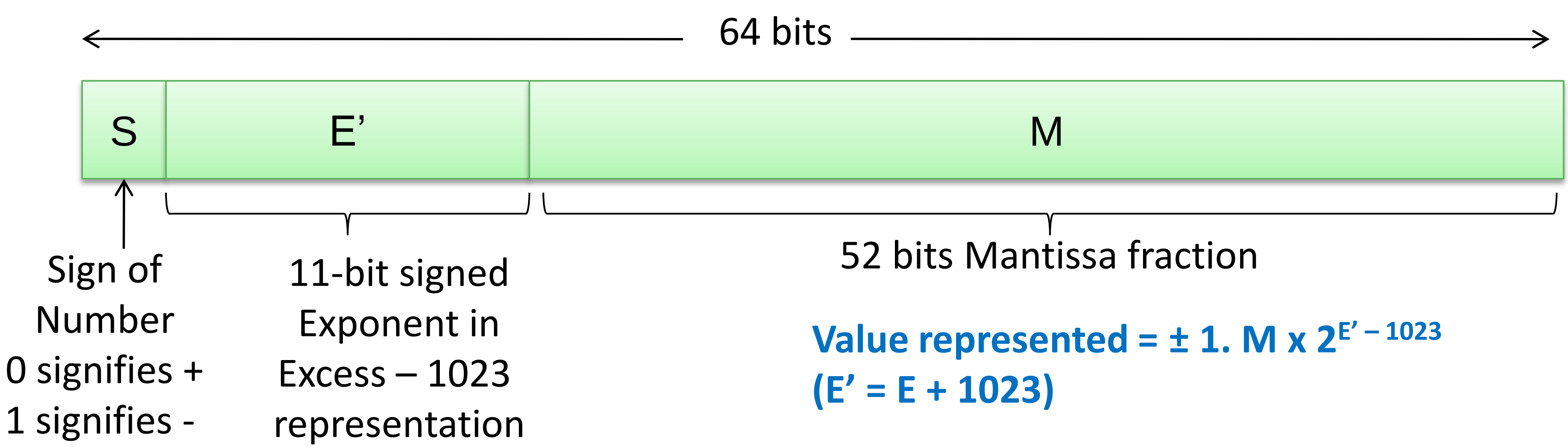
8-bit signed Exponent in Excess – 127 representation

23 bits Mantissa fraction

Value represented = $\pm\, 1.M \times 2^{E' - 127}$
($E' = E + 127$)

Double Precision

- Single Precision
  - $1 \le E' \le 254$ $(-126 \le E \le 127)$
- Double Precision
  - $1 \le E' \le 2046$ $(-1022 \le E \le 1023)$



Value represented = $\pm$ 1. M x $2^{E' - 1023}$
(E’ = E + 1023)

Example

0	0 0 1 0 1 0 0 0	0 0 1 0 1 0 . . .	0

Value represented = $+ 1.0010110 \ldots 0 \times 2^{-87}$

0	1 0 0 0 0 1 0 1	0 1 1 0 . . .

Value represented = $+ 1.0110 \ldots \times 2^{6}$

IEEE 754 Format Interpretation

			Single Precision		Double Precision	
	Sign	Fraction	Biased Exponent	Value	Biased Exponent	Value
Positive Zero	0	0	0	0	0	0
Negative Zero	1	0	0	0	0	0
Plus Infinity	0	0	255 (all 1's)	$+\infty$	2047 (all 1's)	$+\infty$
Minus Infinity	1	0	255 (all 1's)	$-\infty$	2047 (all 1's)	$-\infty$
Not a Number	0 or 1	$\neq 0$	255 (all 1's)	NaN	2047 (all 1's)	NaN
Positive normalized nonzero	0	M	$0 < E' < 255$	$2^{E'-127}$ (1.M)	$0 < E' < 2047$	$2^{E'-1023}$ (1.M)
Negative normalized nonzero	1	M	$0 < E' < 255$	$-2^{E'-127}$ (1.M)	$0 < E' < 2047$	$-2^{E'-1023}$ (1.M)
Positive denormalized	0	$M \neq 0$	0	$-2^{E'-127}$ (0.M)	0	$-2^{E'-1023}$ (0.M)
Negative denormalized	1	$M \neq 0$	0	$-2^{E'-127}$ (0.M)	0	$-2^{E'-1023}$ (0.M)

Floating Point Add/ Subtract

- Check for zeros
- Align the mantissa
- Add or subtract mantissas
- Normalize the result

Multiplication

- Check for zeros
- Add the exponents
- Multiply the mantissas
- Normalize the product

Floating Point Division

- Check for zeros
- Initialize registers and evaluate the sign
- Align the dividend
- Subtract the exponents
- Divide the mantissas

Thank You!!

noor@iiitdm.ac.in