L8/9: Arithmetic Structures

<u>Lecture Material Adapted From:</u>

- **R. Katz, G. Borriello, "Contemporary Logic Design" (second edition), Copyright 2005 Prentice-Hall/Pearson Education.**

- **J. Rabaey, A. Chandrakasan, B. Nikolic, "Digital Integrated Circuits: A Design Perspective" Copyright 2003 Prentice Hall/Pearson Education.**

- **Special thanks to Kevin Atkinson, Alice Wang, Rex Min**

How to represent negative numbers?

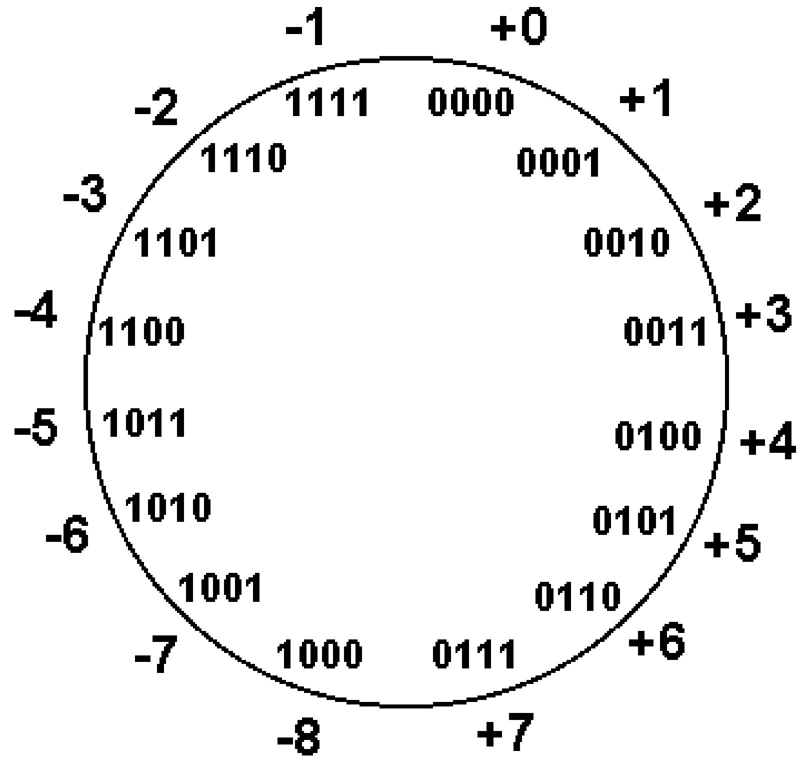
- Three common schemes: sign-magnitude, ones complement, twos complement
- Sign-magnitude: MSB = 0 for positive, 1 for negative
  - Range: $-(2^{N-1} - 1)$ to $+(2^{N-1} - 1)$
  - Two representations for zero: 0000… & 1000…
  - Simple multiplication but complicated addition/subtraction
- Ones complement: if N is positive then its negative is $\overline{N}$
  - Example: 0111 = 7, 1000 = -7
  - Range: $-(2^{N-1} - 1)$ to $+(2^{N-1} - 1)$
  - Two representations for zero: 0000… & 1111…
  - Subtraction implemented as addition and negation

Twos complement = bitwise complement + 1

0111 → 1000 + 1 = 1001 = -7

1001 → 0110 + 1 = 0111 = 7

- Asymmetric range: -2^{N-1} to $+2^{N-1}-1$
- Only one representation for zero
- Simple addition and subtraction
- Most common representation

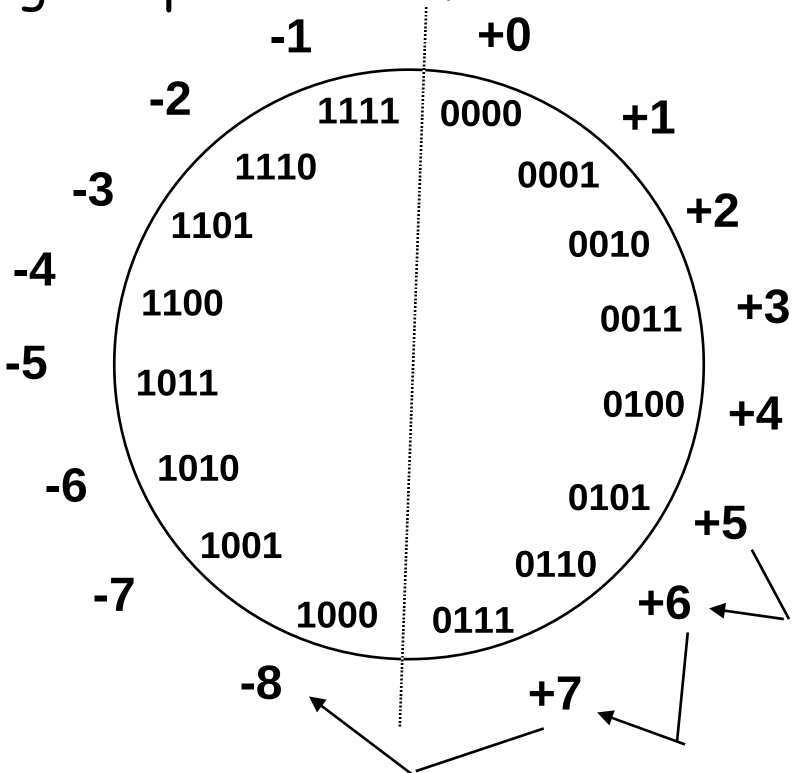


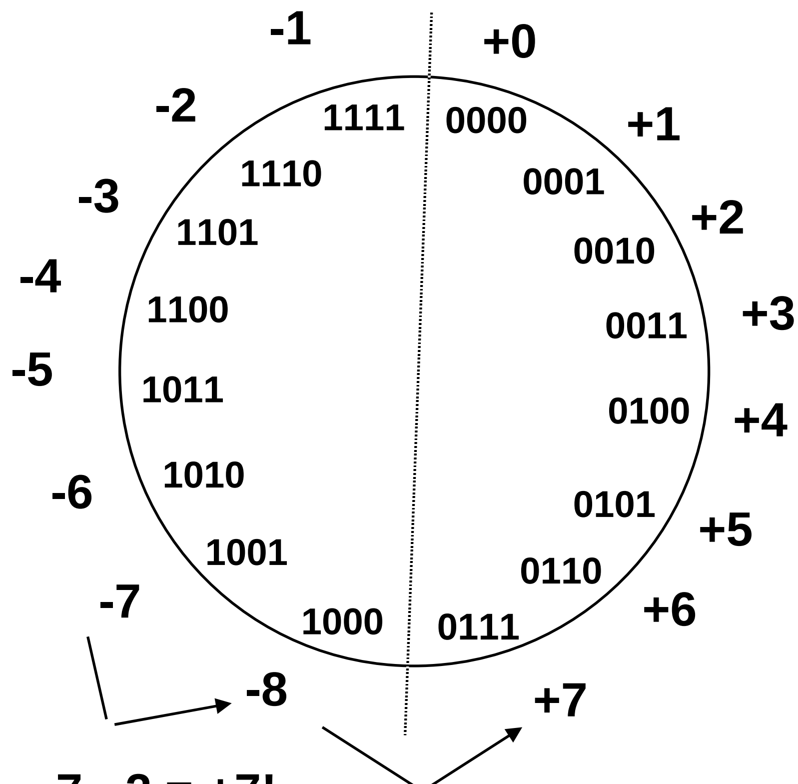
4	0100	-4	1100	4	0100	-4	1100
+ 3	0011	+ (-3)	1101	- 3	1101	+ 3	0011
7	0111	-7	11001	1	10001	-1	1111

[Katz05]

Overflow Conditions

Add two positive numbers to get a negative number or two negative numbers to get a positive number





5 + 3 = -8!

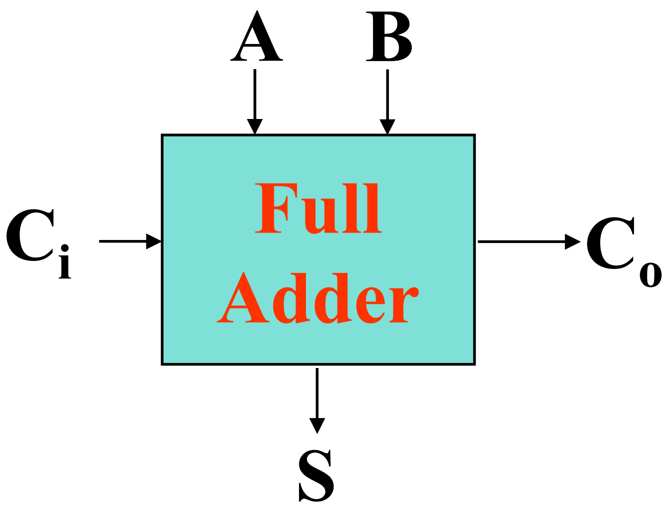
-7 - 2 = +7!

	0 1 1 1
5	0 1 0 1
3	0 0 1 1
-8	0 1 0 0 0

	1 0 0 0
-7	1 0 0 1
-2	1 1 0 0
7	1 0 1 1 1

If carry in to sign equals carry out then can ignore carry out, otherwise have overflow

Binary Full Adder



$$S = A \oplus B \oplus C_i$$
$$= A\overline{B}\,\overline{C_i} + \overline{A}B\overline{C_i} + \overline{A}\,\overline{B}C_i + ABC_i$$

$$C_o = AB + C_i\,(A+B)$$

A	B	CI	S	CO
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

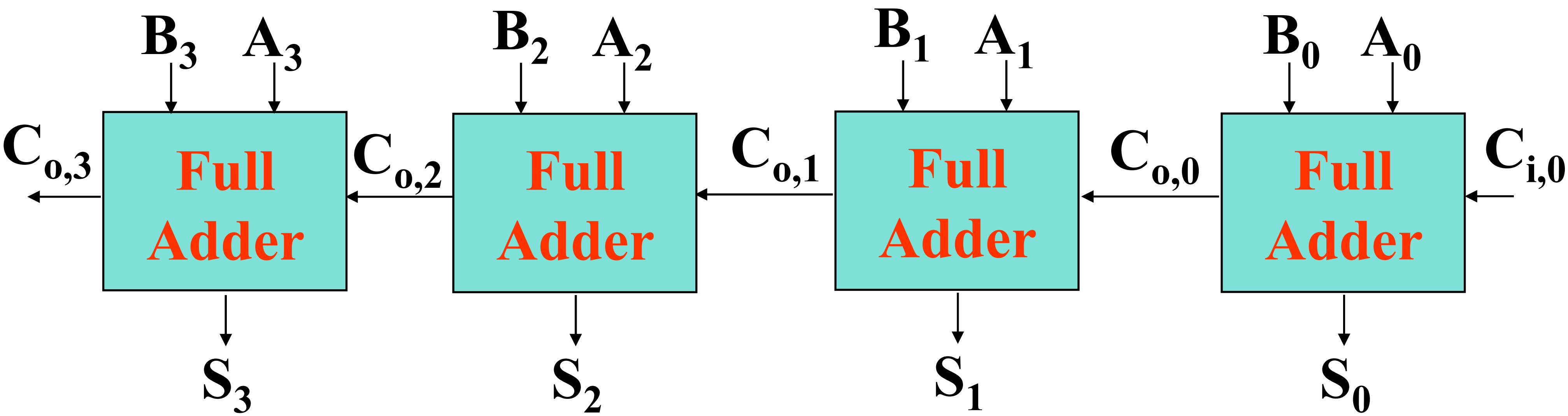
S

CI \ A B	00	01	11	10
0	0	1	0	1
1	1	0	1	0

CO

CI \ A B	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Ripple Carry Adder Structure



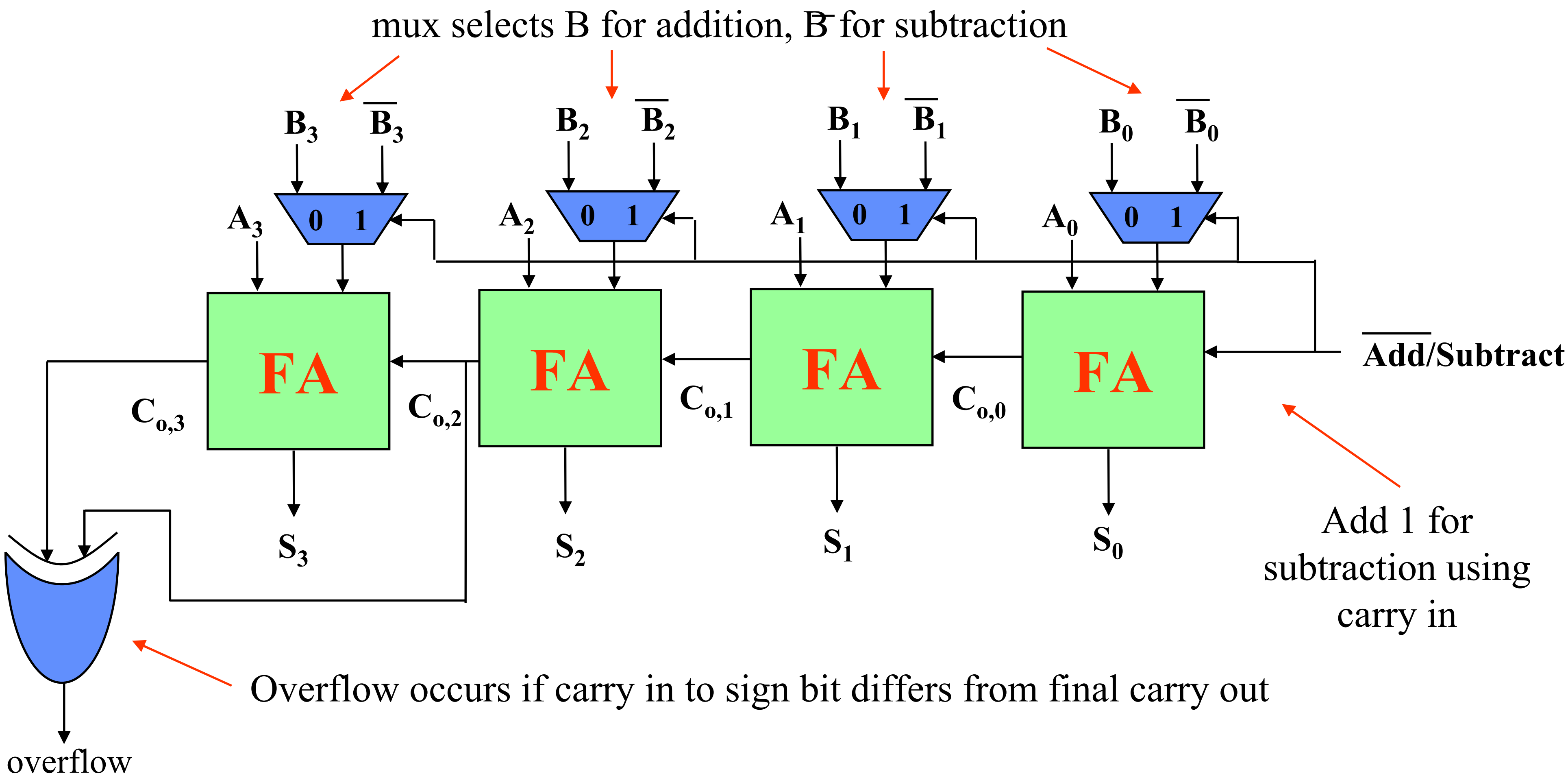
Worst case propagation delay linear with the number of bits

$$t_{adder} = (N-1)t_{carry} + t_{sum}$$

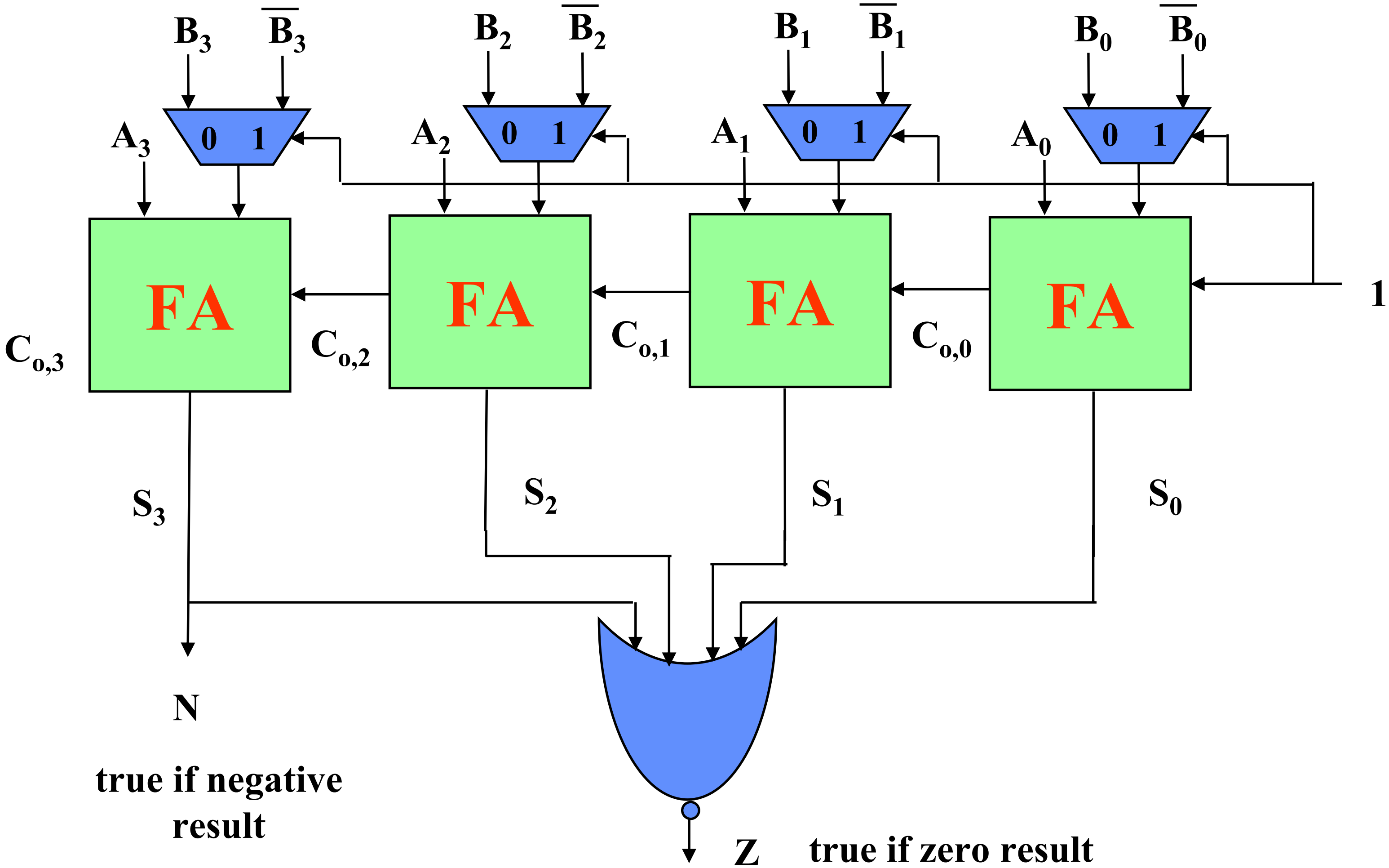
Extension to Subtraction

- Under twos complement, subtracting B is the same as adding the bitwise complement of B then adding 1

Combination addition/subtraction system:



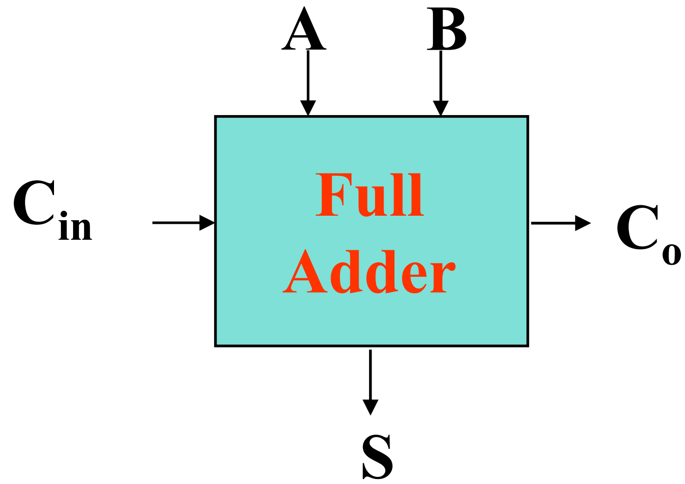
Comparator (one approach)



$$A < B \quad = \quad N$$
$$A = B \quad = \quad Z$$
$$A \leq B \quad = \quad Z + N$$

How to Speed up the Critical (Carry) Path?
(How to Build a Fast Adder?)

A	B	C_i	S	C_o	*Carry status*
0	0	0	0	0	delete
0	0	1	1	0	delete
0	1	0	1	0	propagate
0	1	1	0	1	propagate
1	0	0	1	0	propagate
1	0	1	0	1	propagate
1	1	0	0	1	generate
1	1	1	1	1	generate



Generate (G) = AB

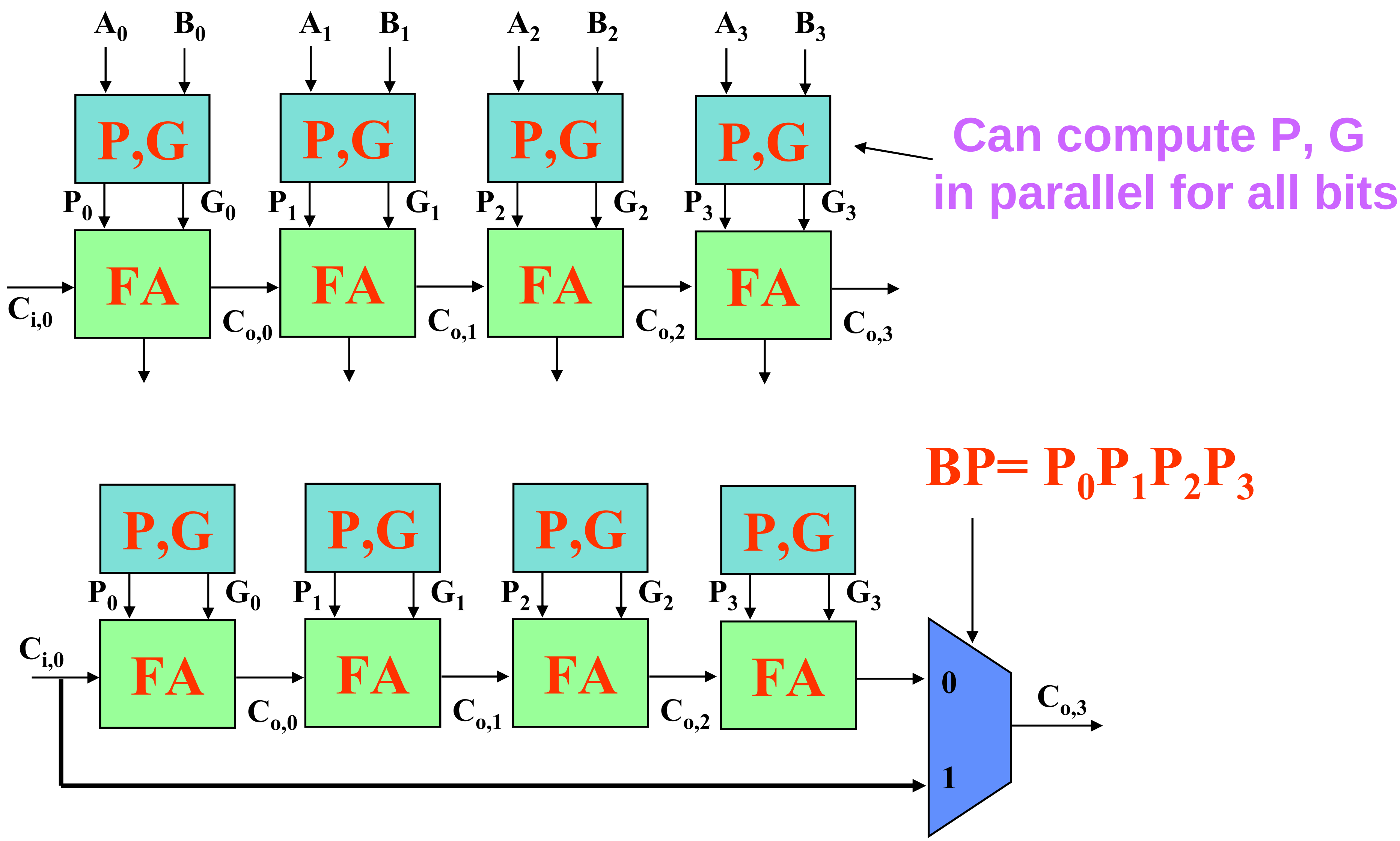
Propagate (P) = A ⊕ B

$$C_o(G,P) = G + PC_i$$

$$S(G,P) = P \oplus C_i$$

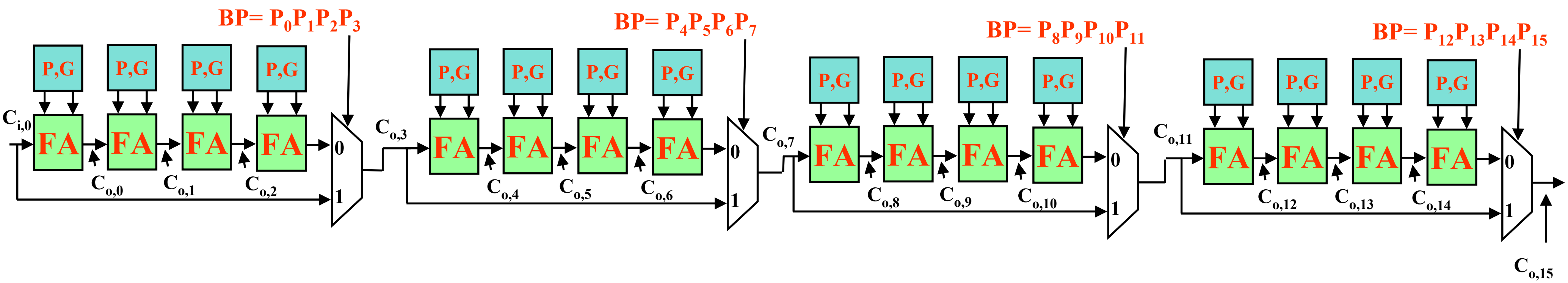
Note: can also use P = A + B for C_o

Carry Bypass Adder



Key Idea: if (P_0 P_1 P_2 P_3) then $C_{o,3} = C_{i,0}$

16-bit Carry Bypass Adder



Assume the following for delay each gate:

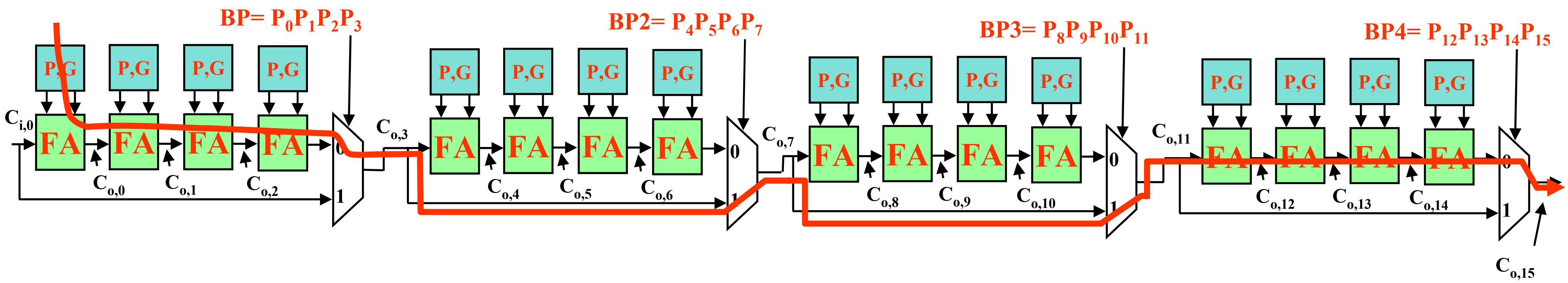
P, G from A, B: 1 delay unit

P, G, C_i to C_o or Sum for a FA: 1 delay unit

2:1 mux delay: 1 delay unit

What is the worst case propagation delay for the 16-bit adder?

Critical Path Analysis



For the second stage, is the critical path:

BP2 = 0 or BP2 = 1?

Message: Timing Analysis is Very Tricky – Must Carefully Consider Data Dependencies For *False Paths*

Carry Lookahead Adder

Re-express the carry logic as follows:

C1 = G0 + P0 C0

C2 = G1 + P1 C1 = G1 + P1 G0 + P1 P0 C0

C3 = G2 + P2 C2 = G2 + P2 G1 + P2 P1 G0 + P2 P1 P0 C0

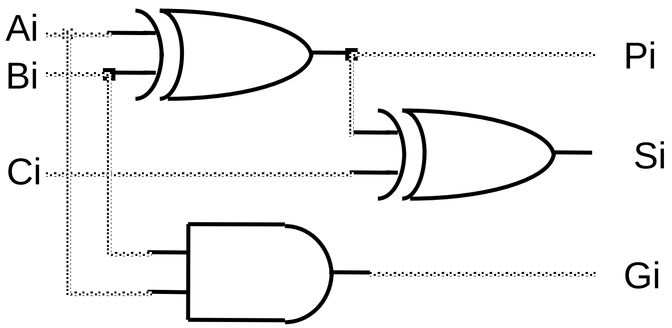
C4 = G3 + P3 C3 = G3 + P3 G2 + P3 P2 G1 + P3 P2 P1 G0 + P3 P2 P1 P0 C0

…

- **Each of the carry equations can be implemented in a two-level logic network**
- **Variables are the adder inputs and carry in to stage 0**

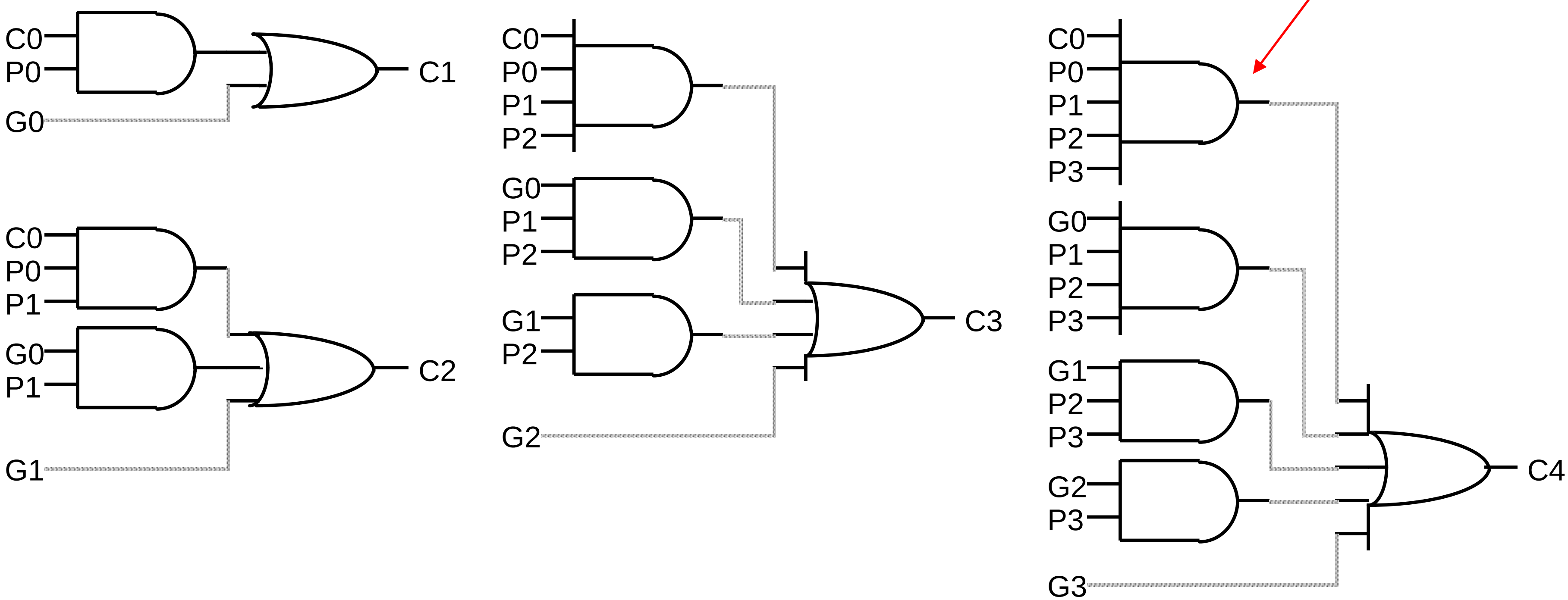
Ripple effect has been eliminated!

Carry Lookahead Logic



Adder with propagate and generate outputs

Later stages have increasingly complex logic



Block Generate and Propagate

$G_{j:i}$ and $P_{j:i}$ denote the Generate and Propagate functions, respectively, for a group of bits from positions i to j. We call them Block Generate and Block Propagate. $G_{j:i}$ equals 1 if the group generates a carry independent of the incoming carry. $P_{j:i}$ equals 1 if an incoming carry propagates through the entire group. For example, $G_{3:2}$ is equal to 1 if a carry is generated at bit position 3, or if a carry out is generated at bit position 2 and propagates through position 3. $G_{3:2} = G_3 + P_3G_2$. $P_{3:2}$ is true if an incoming carry propagates through both bit positions 2 and 3. $P_{3:2} = P_3P_2$

$$\mathbf{C_2 = (G_1 + P_1 G_0) + (P_1 P_0)C_0 = G_{1:0} + P_{1:0} C_0}$$

$$\mathbf{C_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0}$$

$$\mathbf{= (G_3 + P_3 G_2) + (P_3 P_2)C_{0,1} = G_{3:2} + P_{3:2} C_2}$$

$$\mathbf{= G_{3:2} + P_{3:2}(G_{1:0} + P_{1:0} C_0) = G_{3:0} + P_{3:0} C_0}$$

The carry out of a 4-bit block can thus be computed using only the block generate and propagate signals for each 2-bit section, plus the carry in to bit 0. The same formulation will be used to generate the carry out signals for a 16-bit adder using the block generate and propagate from 4-bit sections.

More Definitions…

$$(g, p) \bullet (g', p') = (g + p g', pp')$$

The above dot operator obeys the associative property, but it is not commutative

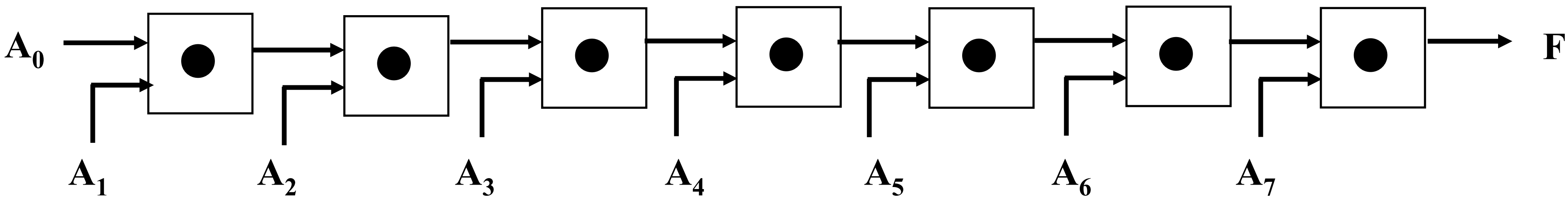
$$(G_{3:2}, P_{3:2}) = (G_3, P_3) \bullet (G_2, P_2)$$

$$(C_{o,\,3}, 0) = ((G_3, P_3) \bullet (G_2, P_2) \bullet (G_1, P_1) \bullet (G_0, P_0)) \bullet (C_{i,\,0}, 0)$$

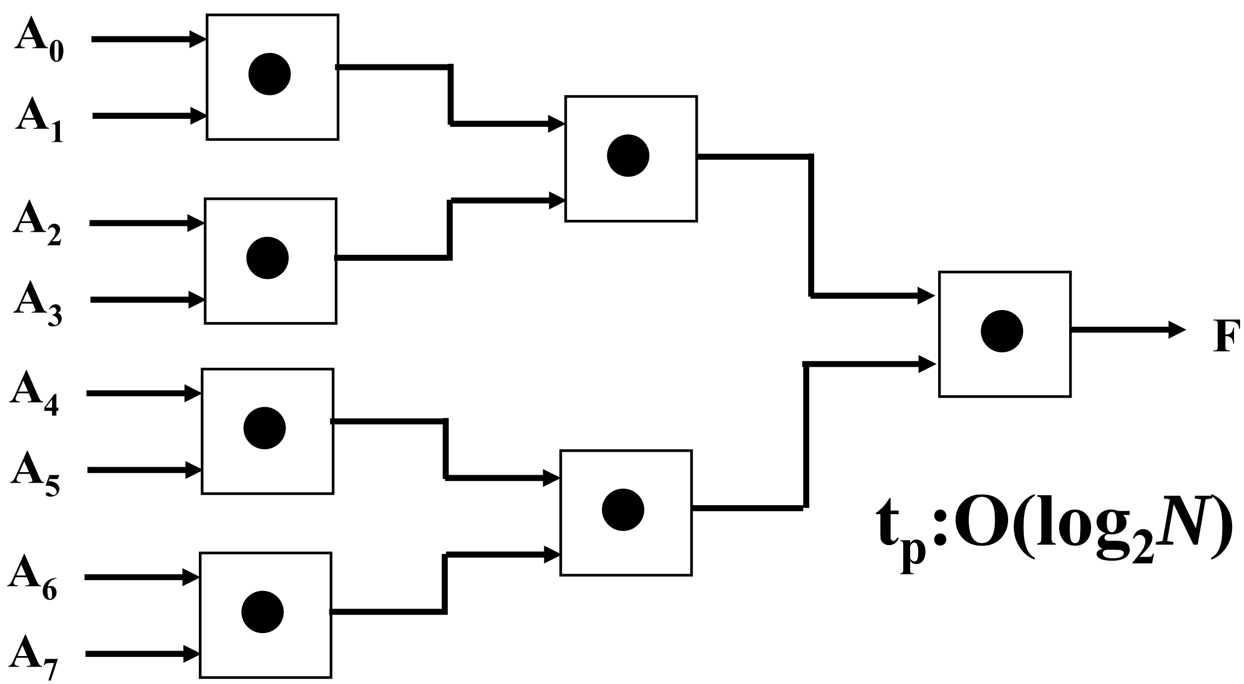
$$\begin{aligned}(G_{3:0}, P_{3:0}) &= [(G_3, P_3) \bullet (G_2, P_2)] \bullet [(G_1, P_1) \bullet (G_0, P_0)] \\ &= (G_{3:2}, P_{3:2}) \bullet (G_{1:0}, P_{1:0})\end{aligned}$$

$$(C_{o,\,k}, 0) = ((G_k, P_k) \bullet (G_{k-1}, P_{k-1}) \bullet \ldots \bullet (G_0, P_0)) \bullet (C_{i,\,0}, 0)$$

Logarithmic Look-Ahead Adder

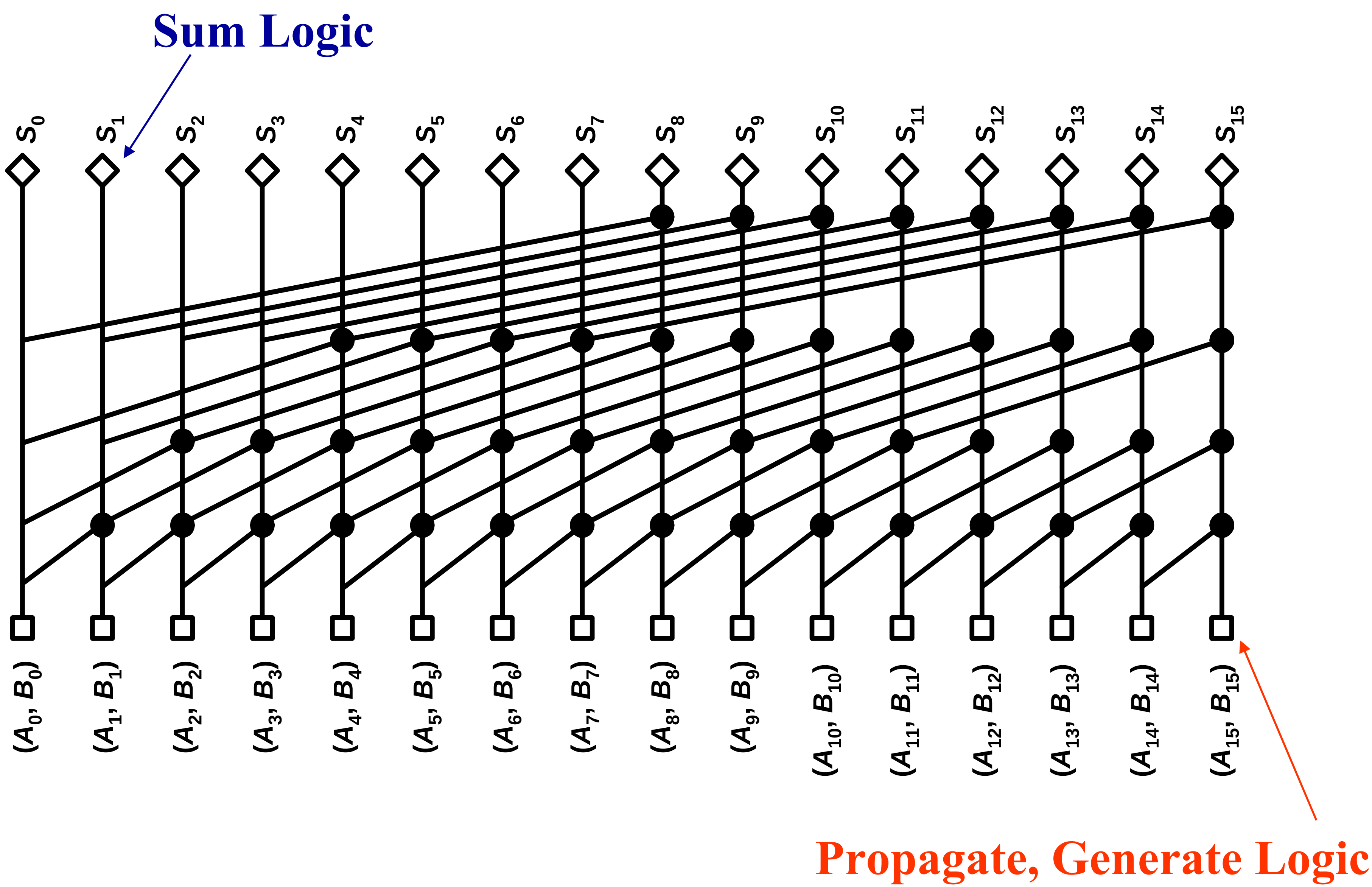


t_p: O(N)

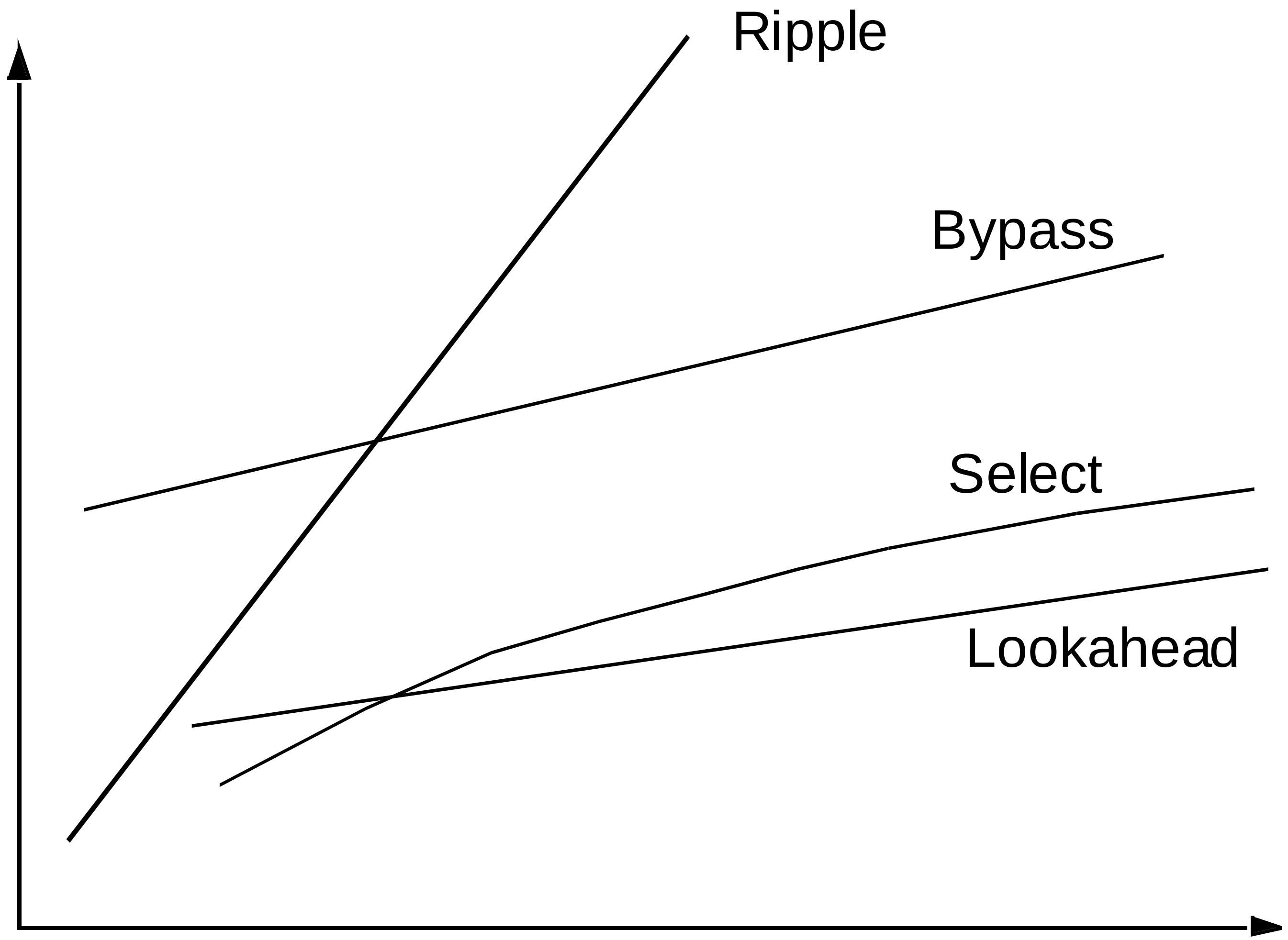


t_p:O($\log_2 N$)

16-bit Kogge-Stone Tree Adder

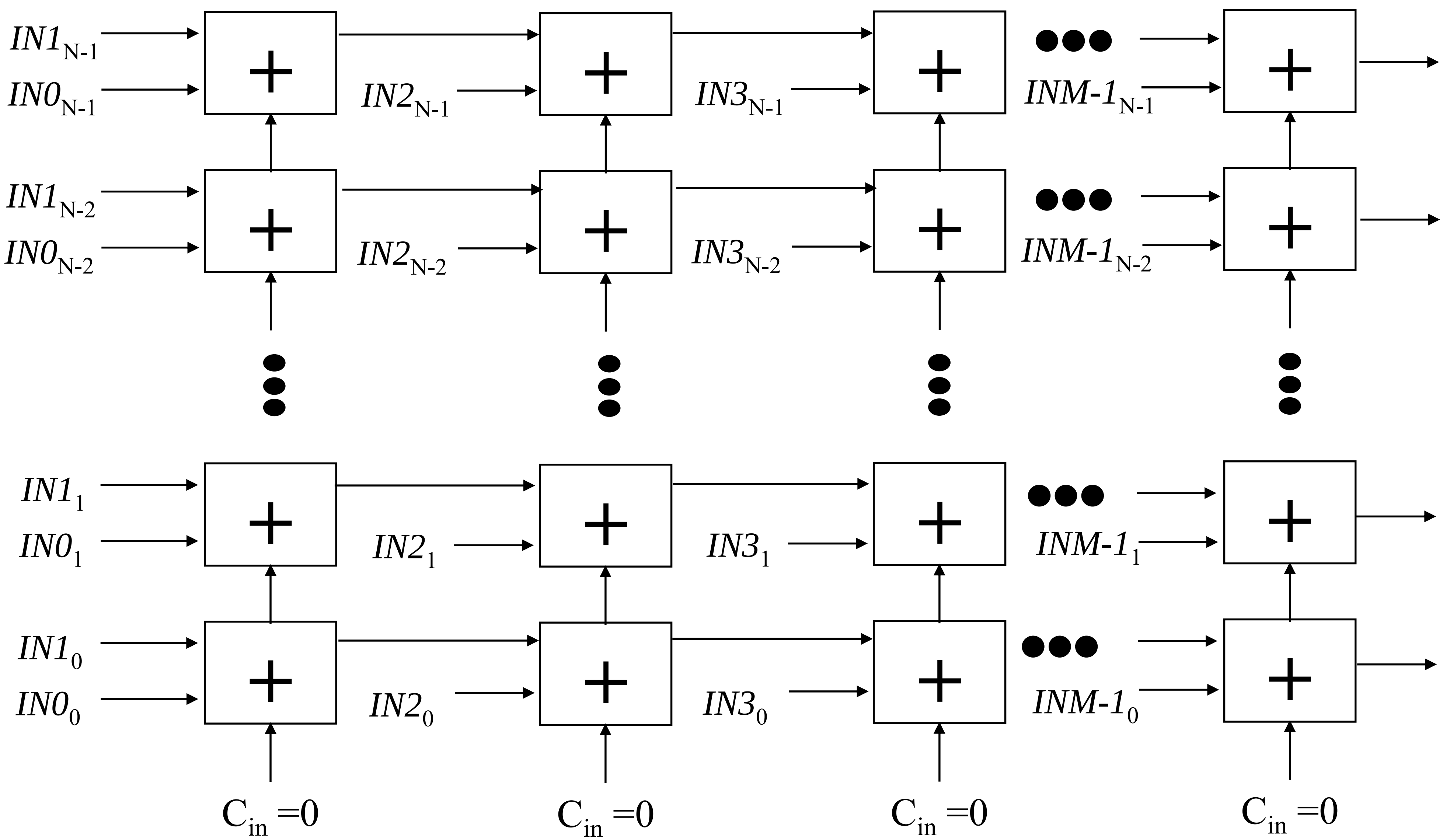


Adder Performance

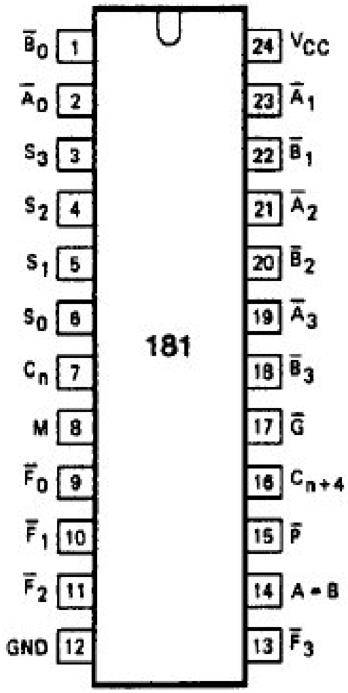


Delay vs. number of bits

Addition of M, N-bit Numbers



74181 TTL 4-bit ALU (TI)

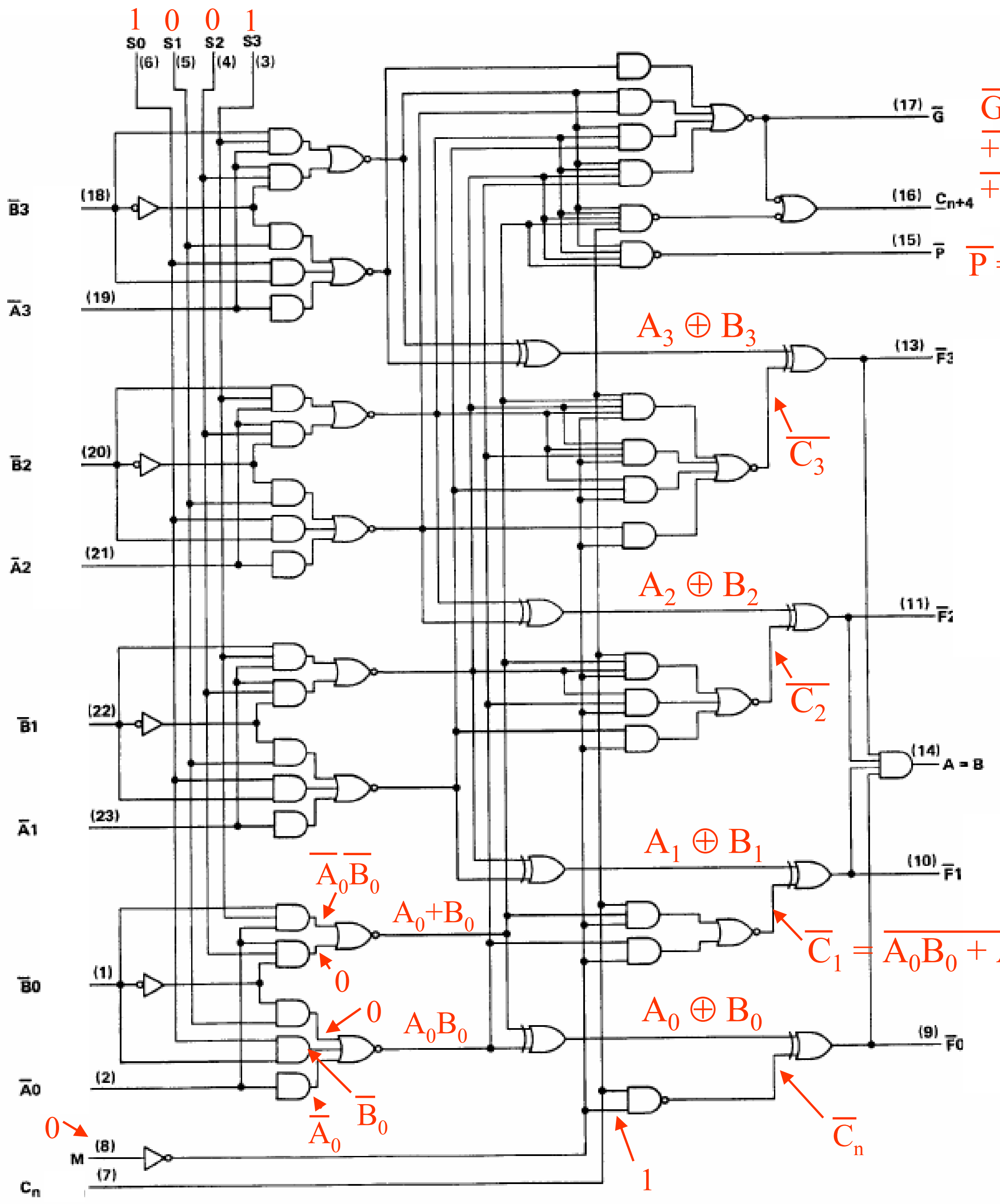


SELECTION				ACTIVE-LOW DATA		
				M = H	M = L; ARITHMETIC OPERATIONS	
S3	S2	S1	S0	LOGIC FUNCTIONS	Cn = L (no carry)	Cn = H (with carry)
L	L	L	L	$F = \overline{A}$	F = A MINUS 1	F = A
L	L	L	H	$F = \overline{AB}$	F = AB MINUS 1	F = AB
L	L	H	L	$F = \overline{A} + B$	$F = A\overline{B}$ MINUS 1	$F = A\overline{B}$
L	L	H	H	F = 1	F = MINUS 1 (2's COMP)	F = ZERO
L	H	L	L	$F = \overline{A + B}$	$F = A$ PLUS $(A + \overline{B})$	$F = A$ PLUS $(A + \overline{B})$ PLUS 1
L	H	L	H	$F = \overline{B}$	$F = AB$ PLUS $(A + \overline{B})$	$F = AB$ PLUS $(A + \overline{B})$ PLUS 1
L	H	H	L	$F = \overline{A \oplus B}$	F = A MINUS B MINUS 1	F = A MINUS B
L	H	H	H	$F = A + \overline{B}$	$F = A + \overline{B}$	$F = (A + \overline{B})$ PLUS 1
H	L	L	L	$F = \overline{A}B$	F = A PLUS (A + B)	F = A PLUS (A + B) PLUS 1
H	L	L	H	$F = A \oplus B$	F = A PLUS B	F = A PLUS B PLUS 1
H	L	H	L	F = B	$F = A\overline{B}$ PLUS (A + B)	$F = A\overline{B}$ PLUS (A + B) PLUS 1
H	L	H	H	F = A + B	F = (A + B)	F = (A + B) PLUS 1
H	H	L	L	F = 0	F = A PLUS A‡	F = A PLUS A PLUS 1
H	H	L	H	$F = A\overline{B}$	F = AB PLUS A	F = AB PLUS A PLUS 1
H	H	H	L	F = AB	$F = A\overline{B}$ PLUS A	$F = A\overline{B}$ PLUS A PLUS 1
H	H	H	H	F = A	F = A	F = A PLUS 1

‡Each bit is shifted to the next more significant position.

- 16 logic functions and 16 arithmetic operations
- Internal 4-bit carry lookahead adder
- Inputs can be active high or active low (active low is shown here)
- Carry in and out are opposite polarity from other inputs/outputs

74181 Addition (Active Low)



$$\overline{G} = \overline{A_3B_3+(A_3+B_3)A_2B_2 +(A_3+B_3)(A_2+B_2)\,A_1B_1 +(A_3+B_3)(A_2+B_2)\,(A_1+B_1)\,A_0B_0}$$

$$\overline{P} = \overline{(A_3+B_3)(A_2+B_2)(A_1+B_1)(A_0+B_0)}$$

A	B	AB	A+B	(AB)⊕(A+B)
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

A⊕B

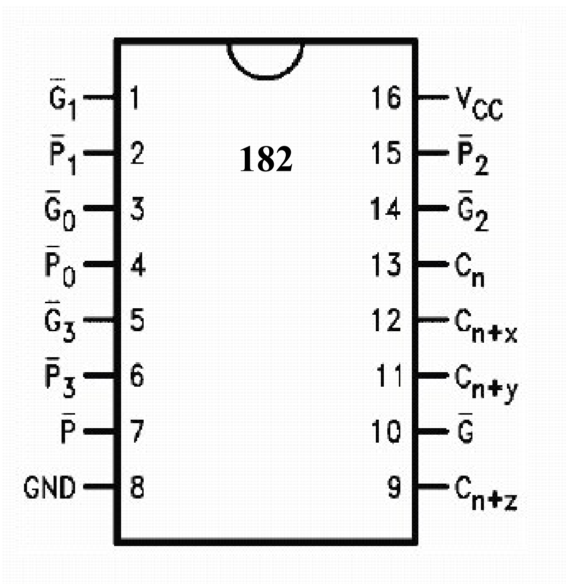
$$\overline{F_1} = \overline{A_1 \oplus B_1 \oplus C_1}$$

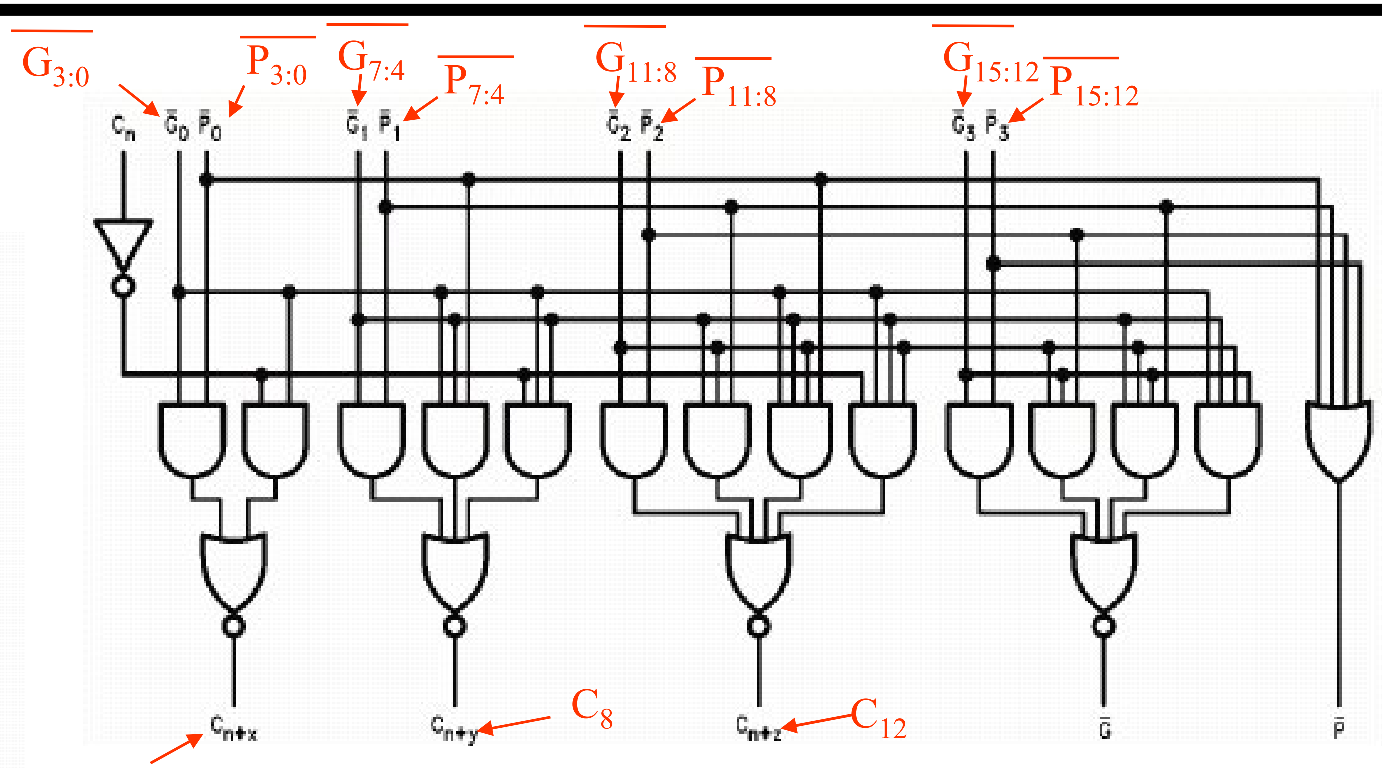
$$\overline{C_1} = \overline{A_0B_0 + A_0C_n + B_0C_n}$$

$$\overline{F_0} = A_0 \oplus B_0 \oplus \overline{C_n}$$
$$= \overline{A_0 \oplus B_0} \oplus C_n$$
$$= \overline{A_0 \oplus B_0 \oplus C_n}$$

74182

74182 carry lookahead unit





- high speed carry lookahead generator
- used with 74181 to extend carry lookahead beyond 4 bits
- correctly handles the carry polarity of the 181

Active low example:

$$C_{n+x} = \overline{\overline{G0}\cdot\overline{P0} + \overline{G0}\cdot\overline{C_n}}$$

$$= \overline{\overline{G0}\cdot\overline{P0}}\cdot\overline{\overline{G0}\cdot\overline{C_n}}$$

$$= (G0+P0)\cdot(G0+C_n) \;=\; G0 + P0C_n$$

- $C_4 = G_{3:0} + P_{3:0}C_n$

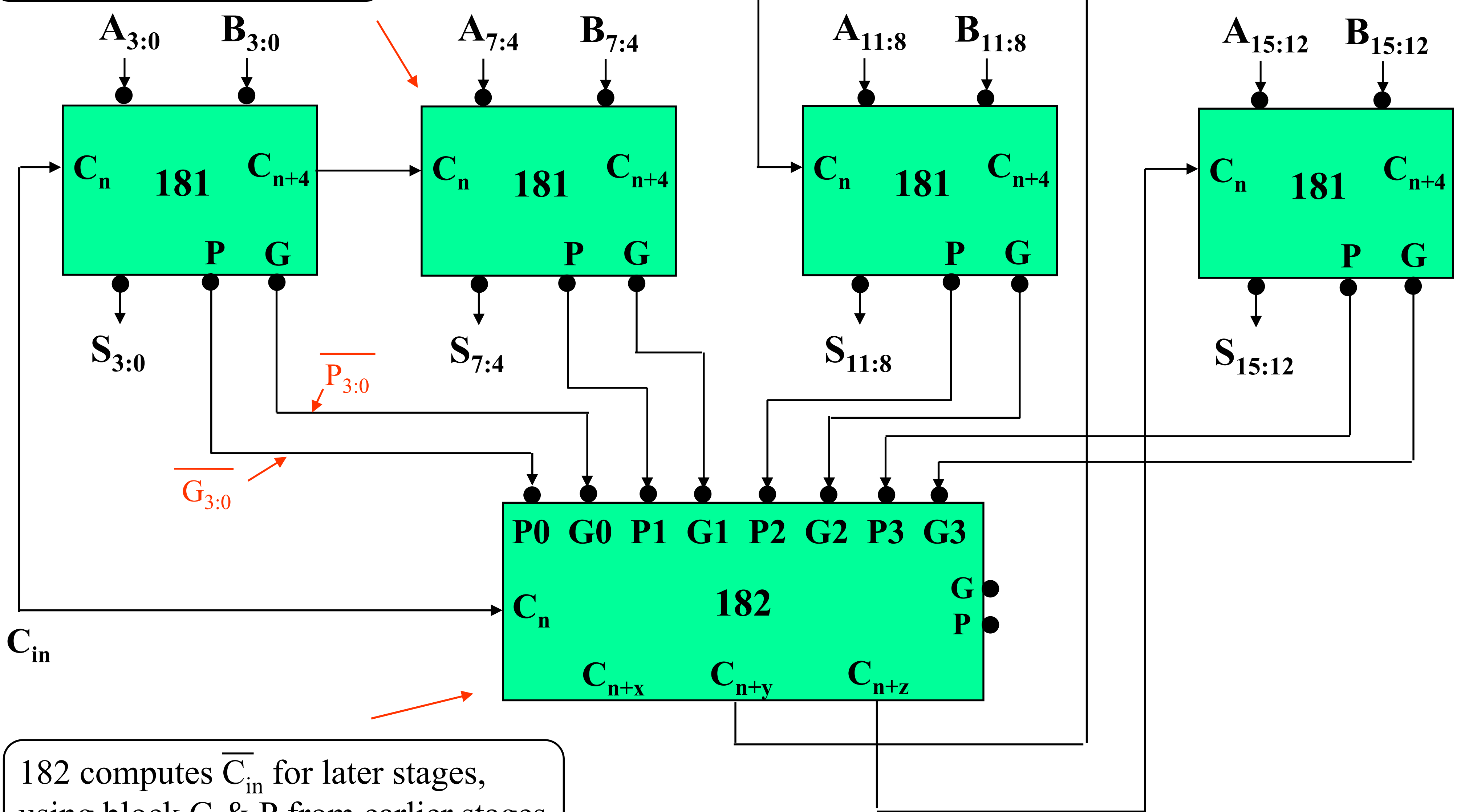
$$C_{n+y} = C_8 = G_{7:4} + P_{7:4}G_{3:0} + P_{7:4}P_{3:0}C_{i,0} = G_{7:0} + P_{7:0}C_n$$

$$C_{n+z} = C_{12} = G_{11:8} + P_{11:8}G_{7:4} + P_{11:8}P_{7:4}G_{3:0} + P_{11:8}P_{7:4}P_{3:0}C_n$$

$$= G_{11:0} + P_{11:0}C_n$$

16-bit Carry Lookahead Schematic

181 configured for A+B:
M = 0, $S_{3\text{-}0}$ = 1001

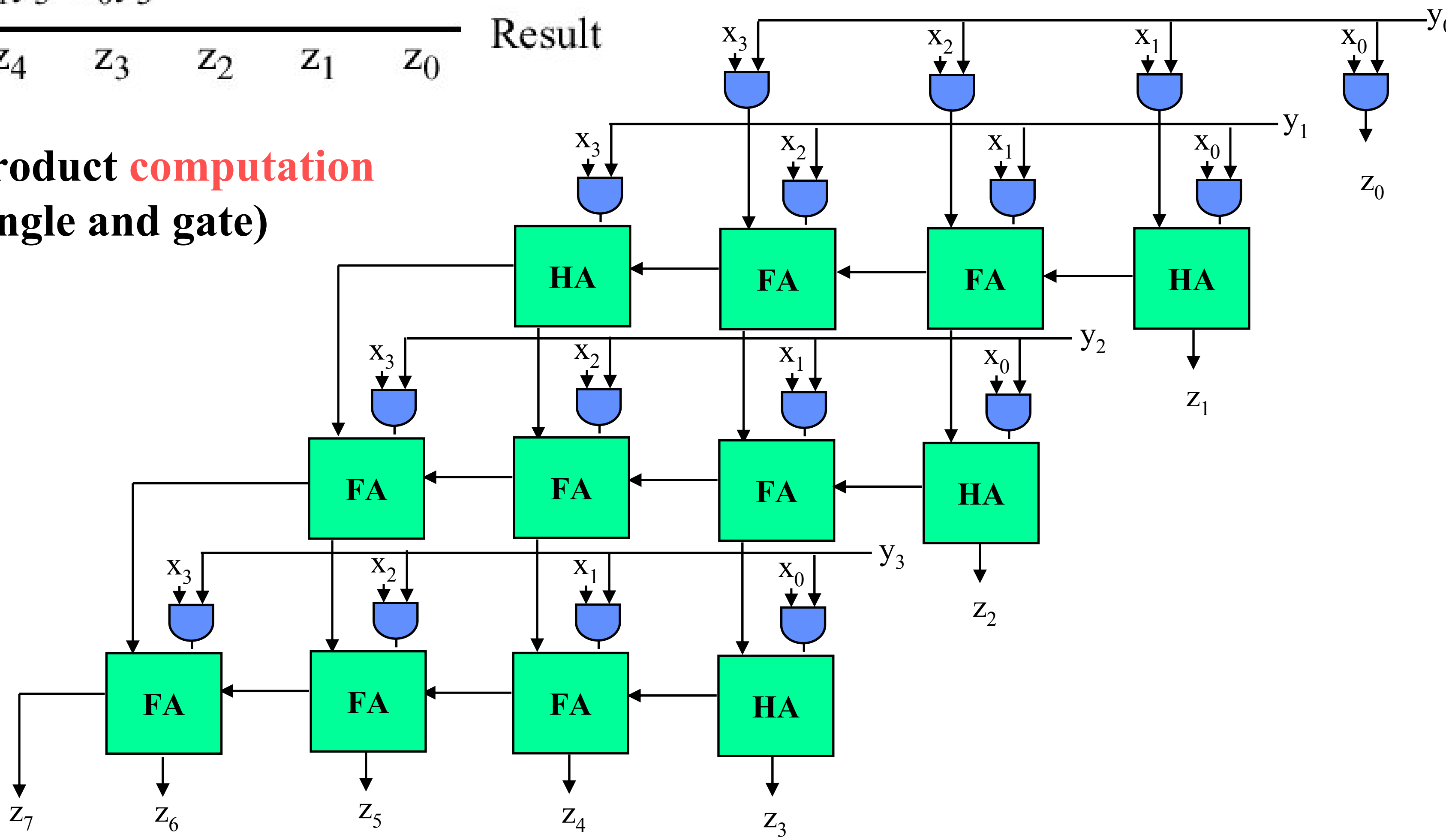


182 computes $\overline{C}_{in}$ for later stages, using block G & P from earlier stages

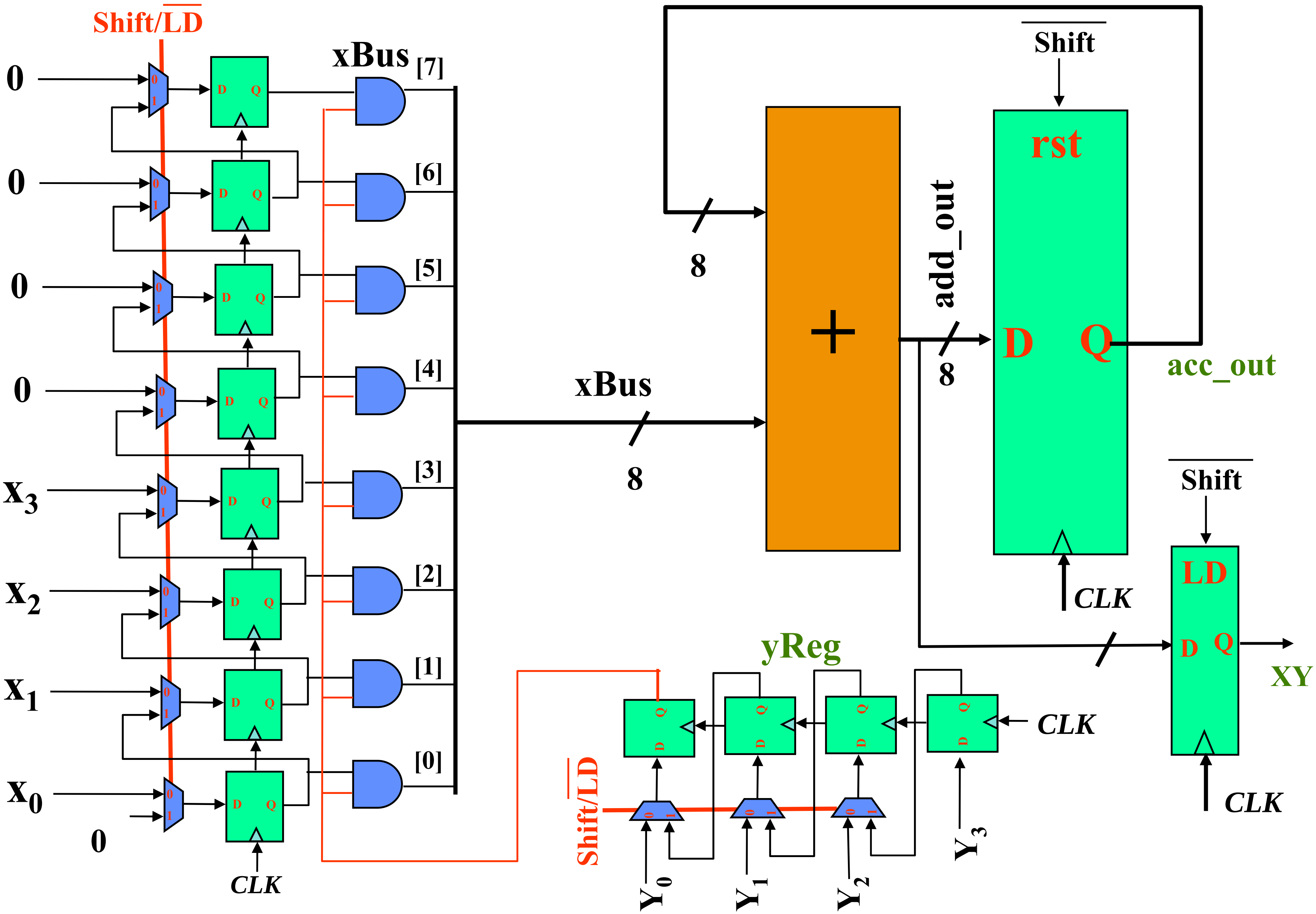
Binary Multiplication

				x_3	x_2	x_1	x_0	Multiplicand
×				y_3	y_2	y_1	y_0	Multiplier
				x_3y_0	x_2y_0	x_1y_0	x_0y_0	
			x_3y_1	x_2y_1	x_1y_1	x_0y_1		Partial Product
		x_3y_2	x_2y_2	x_1y_2	x_0y_2			
+	x_3y_3	x_2y_3	x_1y_3	x_0y_3				
z_7	z_6	z_5	z_4	z_3	z_2	z_1	z_0	Result

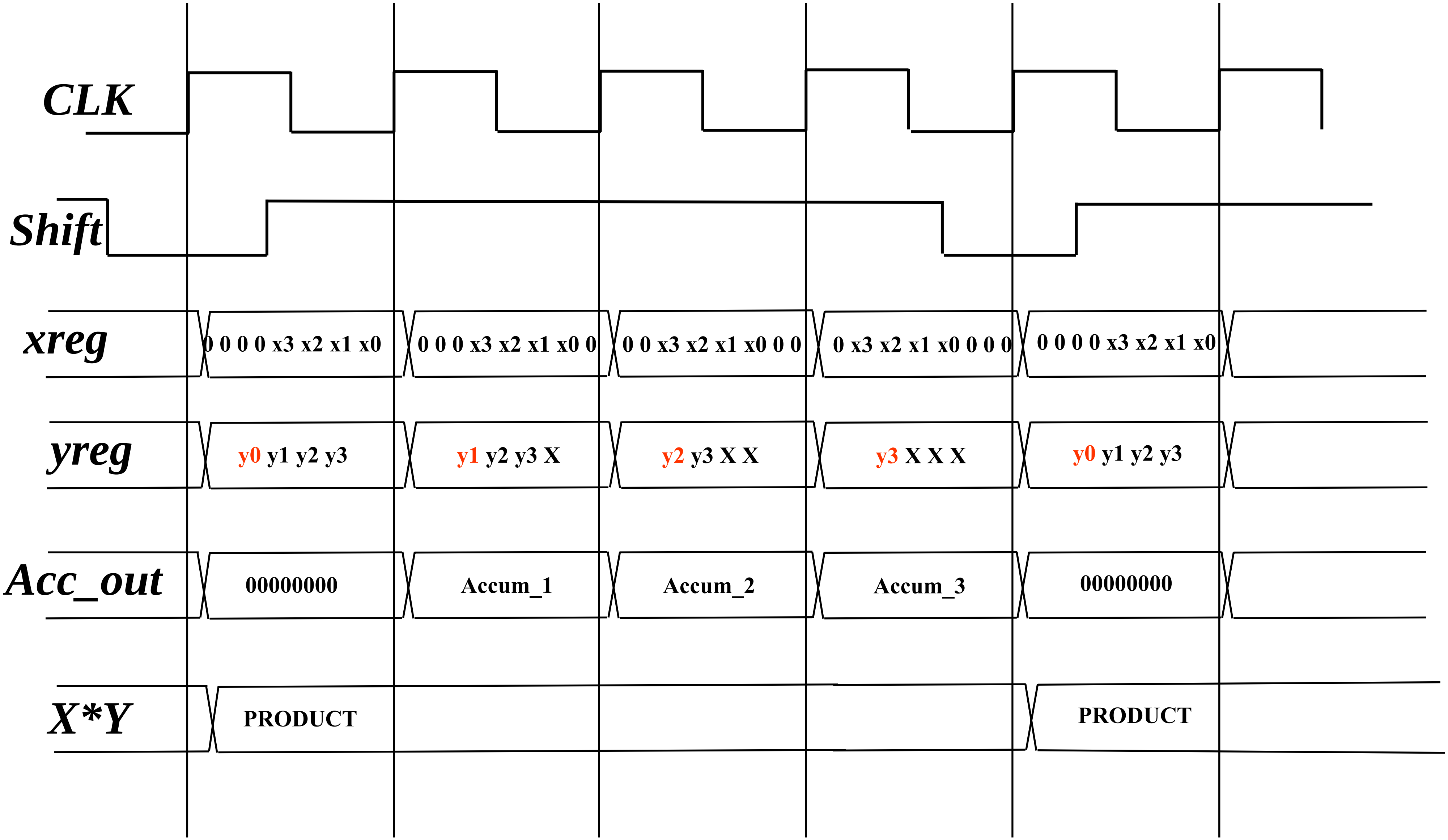
- **Partial product computation is simple (single and gate)**



A Serial (Magnitude) Multiplier



Timing Diagram



Verilog of Serial Multiplier

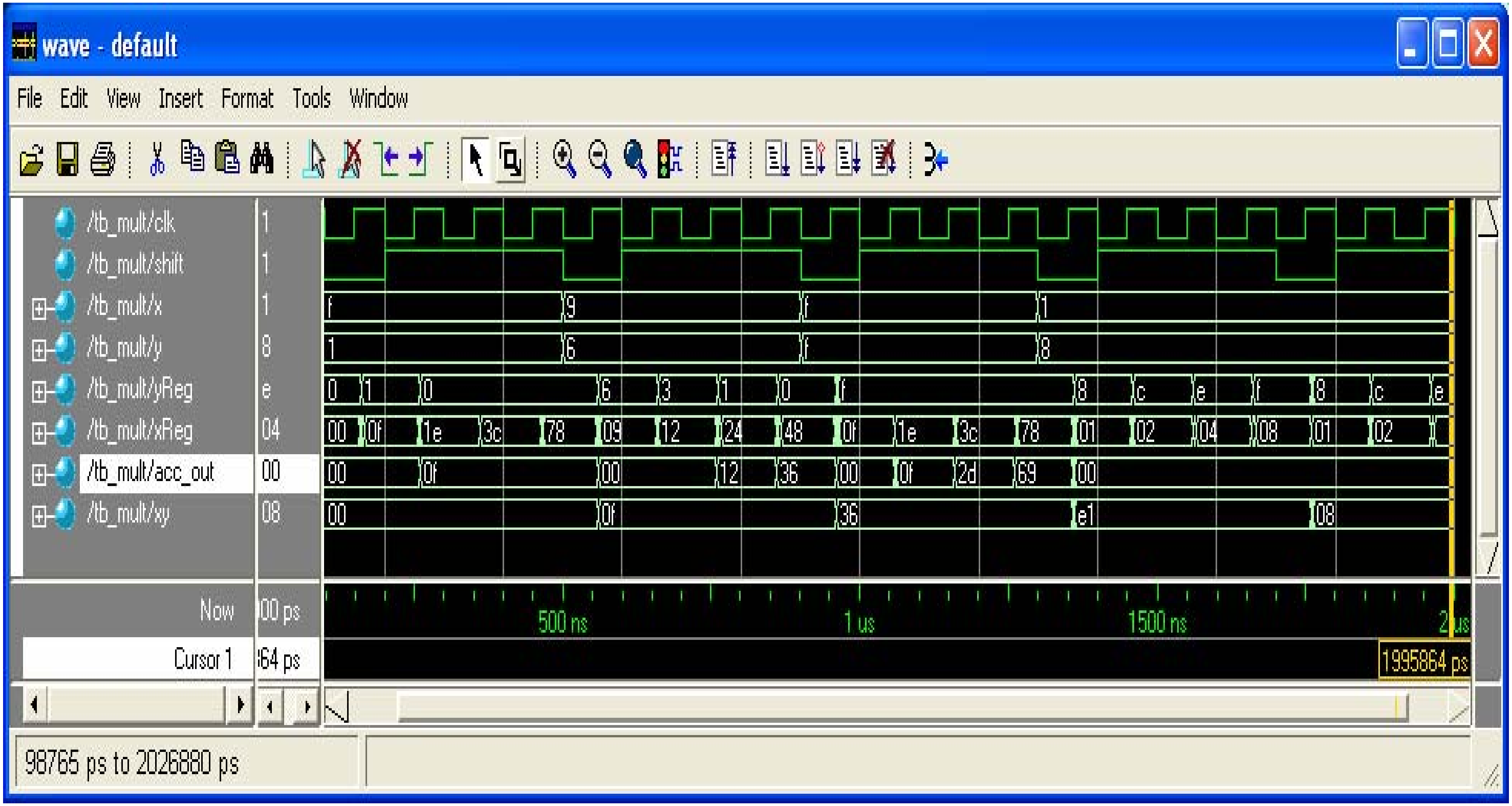
```
module serialmult(shift, clk,
x, y, xy);
input shift, clk;
input [3:0] x, y;
output [7:0] xy;
reg [7:0] xReg;
reg [3:0] yReg;
reg [7:0] xBus, acc_out,
xy_int;
wire[7:0] add_out;
assign add_out = xBus +
acc_out;
assign xy = xy_int;

always @ (yReg[0] or xReg)
begin
if (yReg[0] == 1'b0) xBus =
8'b0;
else xBus = xReg;
end
```

```
always @ (posedge clk)

begin
  if (shift == 1'b0)
     begin
       xReg <= {4'b0, x};
       yReg <= y;
       acc_out <= 8'b0;
       xy_int <= add_out;
     end
   else
     begin
       xReg <= {xReg[6:0], 1'b0};
       yReg <= {y[3], yReg[3:1]};
       acc_out <= add_out;
       xy_int <= xy;
     end // if shift
end // always
endmodule
```

Simulation



wave - default
File Edit View Insert Format Tools Window
/tb_mult/clk
/tb_mult/shift
/tb_mult/x
/tb_mult/y
/tb_mult/yReg
/tb_mult/xReg
/tb_mult/acc_out
/tb_mult/xy
Now
Cursor 1
500 ns
1 us
1500 ns
1995864 ps
98765 ps to 2026880 ps

Baugh Wooley Formulation

Assuming X and Y are 4-bit twos complement numbers:

$$X = -2^3 x_3 + \sum_{i=0}^{2} x_i 2^i \qquad Y = -2^3 y_3 + \sum_{i=0}^{2} y_i 2^i$$

The product of X and Y is:

$$XY = x_3 y_3 2^6 - \sum_{i=0}^{2} x_i y_3 2^{i+3} - \sum_{j=0}^{2} x_3 y_j 2^{j+3} + \sum_{i=0}^{2}\sum_{j=0}^{2} x_i y_j 2^{i+j}$$

For twos complement, the following is true:

$$-\sum_{i=0}^{3} x_i 2^i = -2^4 + \sum_{i=0}^{3} \overline{x}_i 2^i + 1$$

The product then becomes:

$$XY = x_3 y_3 2^6 + \sum_{i=0}^{2} \overline{x_i y_3} 2^{i+3} + 2^3 - 2^6 + \sum_{j=0}^{2} \overline{x_3 y_j} 2^{j+3} + 2^3 - 2^6 + \sum_{i=0}^{2}\sum_{j=0}^{2} x_i y_j 2^{i+j}$$

$$= x_3 y_3 2^6 + \sum_{i=0}^{2} \overline{x_i y_3} 2^{i+3} + \sum_{j=0}^{2} \overline{x_3 y_j} 2^{j+3} + \sum_{i=0}^{2}\sum_{j=0}^{2} x_i y_j 2^{i+j} + 2^4 - 2^7$$

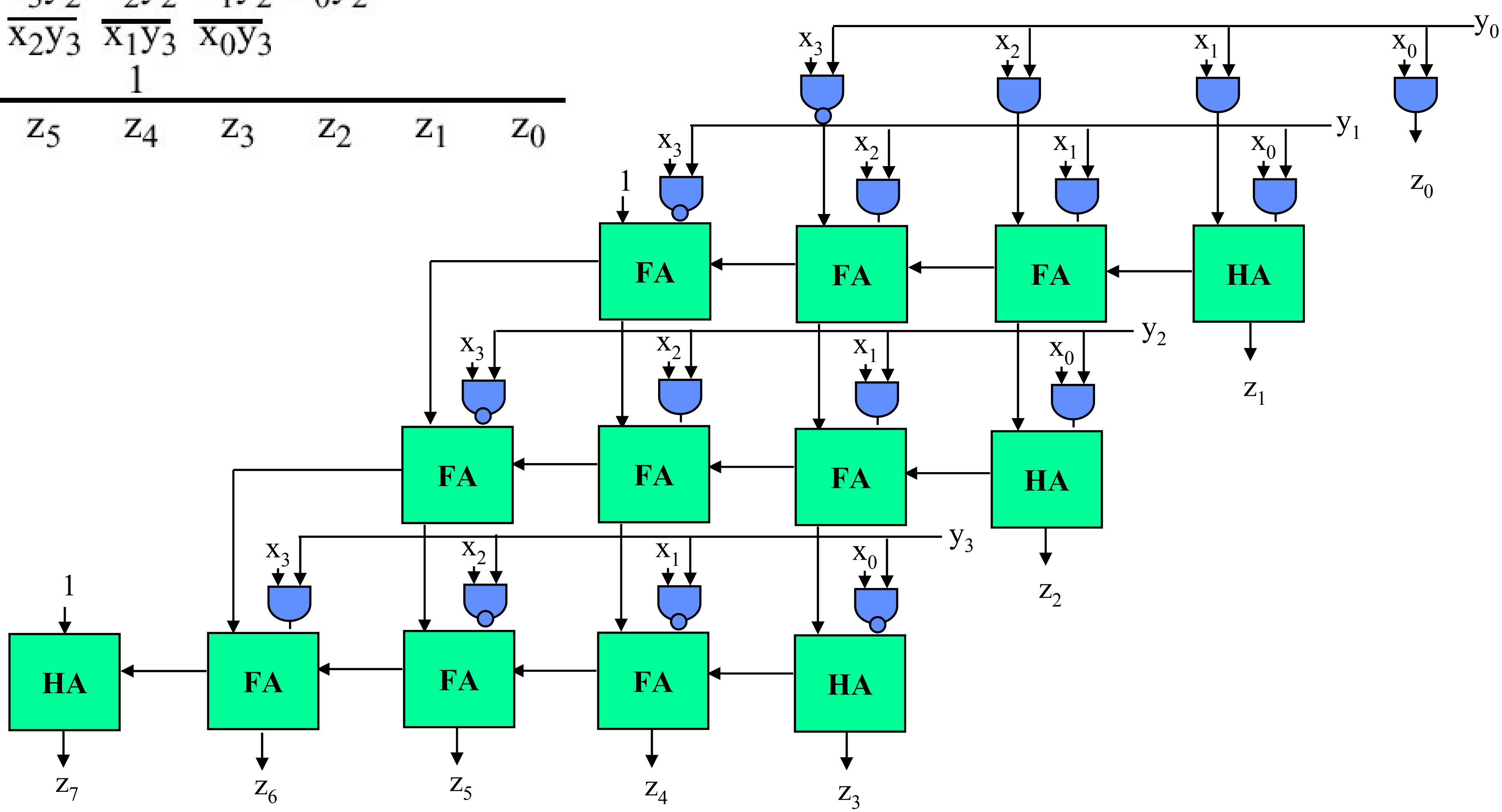
$$= -2^7 + x_3 y_3 2^6 + (\overline{x_2 y_3} + \overline{x_3 y_2})2^5 + (\overline{x_1 y_3} + \overline{x_3 y_1} + x_2 y_2 + 1)2^4$$

$$+ (\overline{x_0 y_3} + \overline{x_3 y_0} + x_1 y_2 + x_2 y_1)2^3 + (x_0 y_2 + x_1 y_1 + x_2 y_0)2^2 + (x_0 y_1 + x_1 y_0)2^1$$

$$+ (x_0 y_0)2^0$$

Twos Complement Multiplication

	z_7	z_6	z_5	z_4	z_3	z_2	z_1	z_0	
					x_3	x_2	x_1	x_0	Multiplicand
$\times$					y_3	y_2	y_1	y_0	Multiplier
					$\overline{x_3y_0}$	x_2y_0	x_1y_0	x_0y_0	
				$\overline{x_3y_1}$	x_2y_1	x_1y_1	x_0y_1		
			$\overline{x_3y_2}$	x_2y_2	x_1y_2	x_0y_2			
		x_3y_3	$\overline{x_2y_3}$	$\overline{x_1y_3}$	$\overline{x_0y_3}$				
$+$	1			1					
	z_7	z_6	z_5	z_4	z_3	z_2	z_1	z_0	



Summary

- Performance of arithmetic blocks dictate the performance of a digital system
- Architectural and logic transformations can enable significant speed up (e.g., adder delay from $O(N)$ to $O(\log_2(N))$)
- Similar concepts and formulation can be applied at the system level
- Timing analysis is tricky: watch out for false paths!
- Area-Delay trade-offs (serial vs. parallel implementations)