



L10: Arithmetic Structures



Acknowledgements:

R. Katz, “*Contemporary Logic Design*”, Addison Wesley Publishing Company, Reading, MA, 1993.

J. Rabaey, A. Chandrakasan, B. Nikolic, “*Digital Integrated Circuits: A Design Perspective*” Prentice Hall, 2003.

Kevin Atkinson, Alice Wang



Number Systems Basics



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...
 - Leads to complicated addition/subtraction, but simple multiplication
- **Ones complement: if N is positive then its negative is $N -$**
 - Ex.: 0111 = 7, 1000 = -7
 - Range: $-(2^{N-1} - 1)$ to $+(2^{N-1} - 1)$
 - Still two representations for zero: 0000... & 1111...
 - Subtraction implemented as addition followed by ones complement – simpler than twos complement



Twos Complement Representation



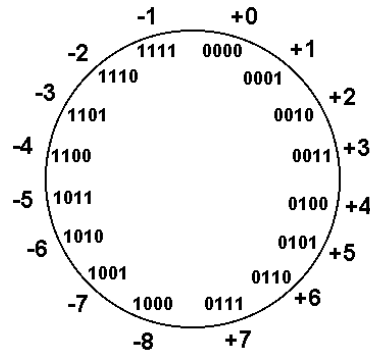
Twos complement = bitwise complement + 1

$$0111 \rightarrow 1000 + 1 = 1001 = -7$$

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

$$1001 \rightarrow 0110 + 1 = 0111 = 7$$

- 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
<u>+3</u>	<u>0011</u>	<u>+(-3)</u>	<u>1101</u>	<u>+(-3)</u>	<u>1101</u>	<u>+3</u>	<u>0011</u>
7	0111	-7	11001	1	10001	-1	1111

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

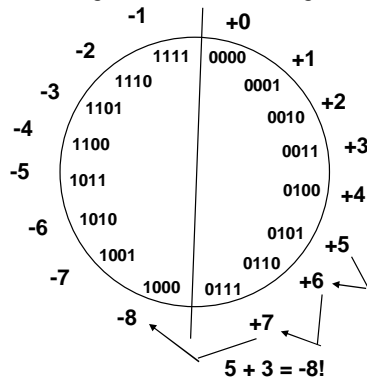


Overflow Conditions

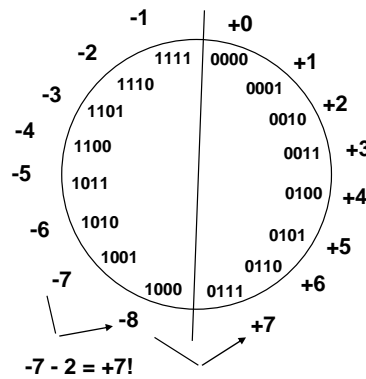


Add two positive numbers to get a negative number

or two negative numbers to get a positive number.



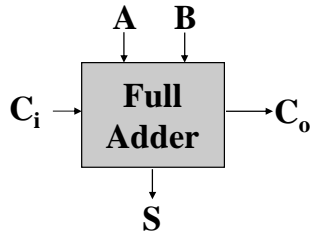
	0	1	1	1
5	0	1	0	1
<u>3</u>	<u>0</u>	<u>0</u>	<u>1</u>	<u>1</u>
-8	0	1	0	0



	1	0	0	0
-7	1	0	0	1
<u>-2</u>	<u>1</u>	<u>1</u>	<u>0</u>	<u>0</u>
7	1	0	1	1



Binary Full Adder



$$S = A \oplus B \oplus C$$

$$= \overline{A} \overline{B} C_i + \overline{A} B \overline{C}_i + A \overline{B} \overline{C}_i + A B C_i$$

$$C = AB + BC_i + AC_i$$

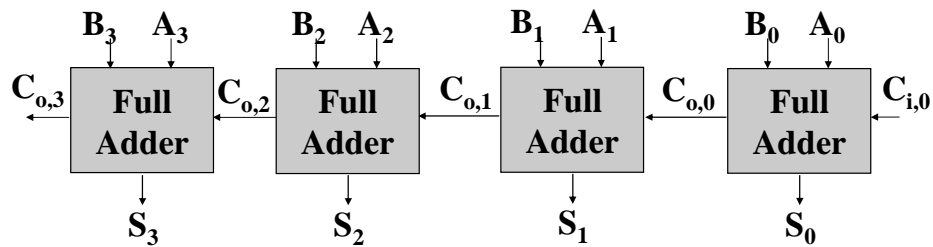
A	B	Ci	S	CO
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Ci	AB			
	00	01	11	10
0	0	1	0	1
1	1	0	1	0

Ci	AB			
	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

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

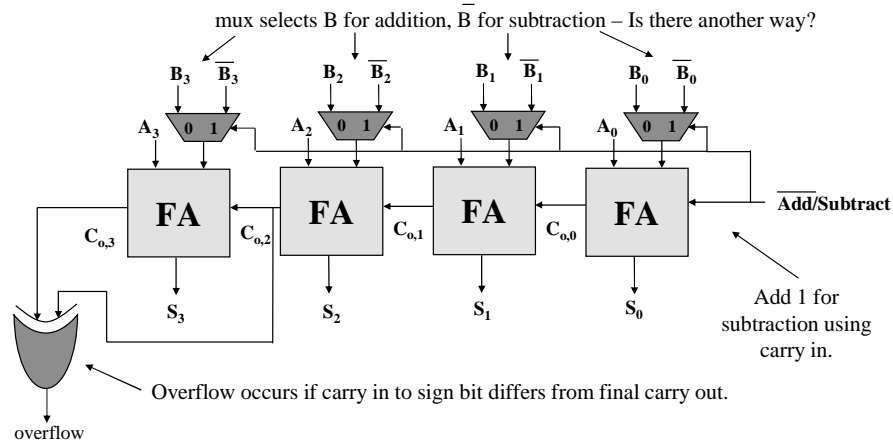


Extension to Subtraction



- Under two's complement, subtracting B is the same as adding the bitwise complement of B, then adding 1.

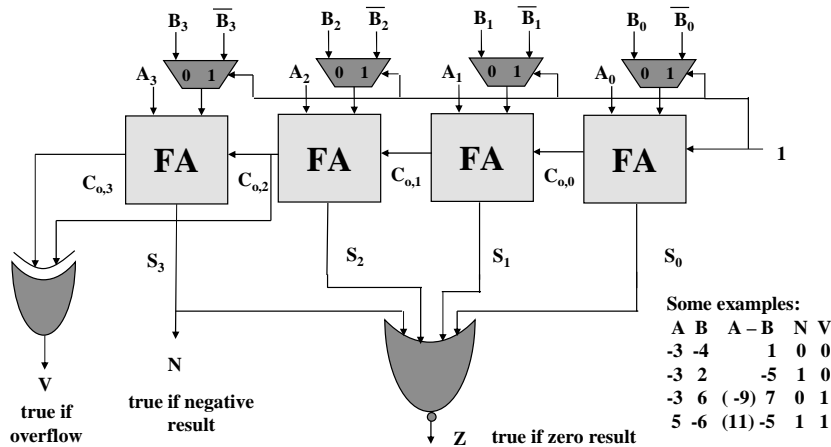
Combination addition/subtraction system:



7



Comparator (one approach)



$$A < B = N \oplus V$$

$$A \leq B = Z + (N \oplus V)$$

$$A = B = Z$$

$$A > B = \overline{Z + (N \oplus V)}$$

$$A \geq B = \overline{N \oplus V}$$

$$A \neq B = \overline{Z}$$

8

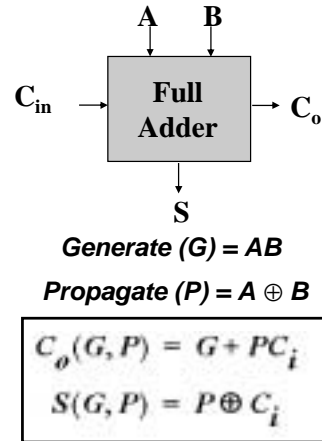


Alternate Adder Logic Formulation

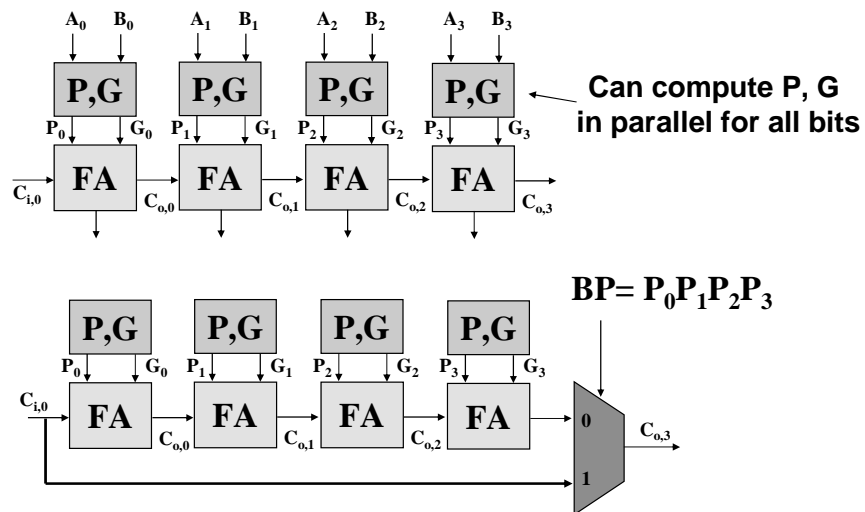


How to Speed up the Critical (Carry) Path? (How to Build a Fast Adder?)

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



Carry Bypass Adder



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



Carry Lookahead Adder



Re-express the carry logic as follows:

$$C1 = G0 + P0 C0$$

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

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

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

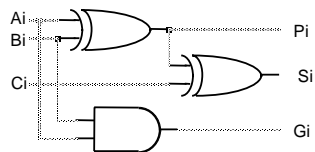
...

- 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!

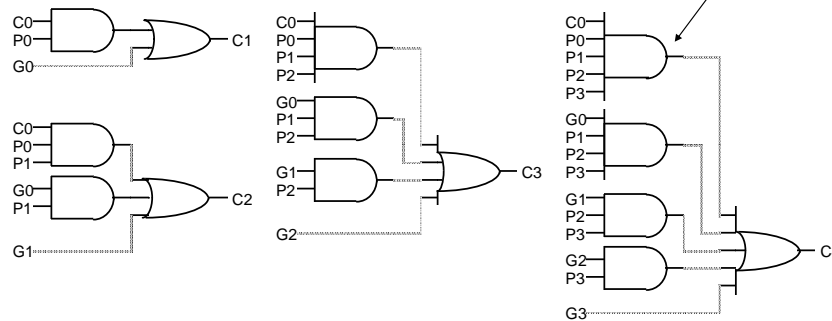


Carry Lookahead Logic



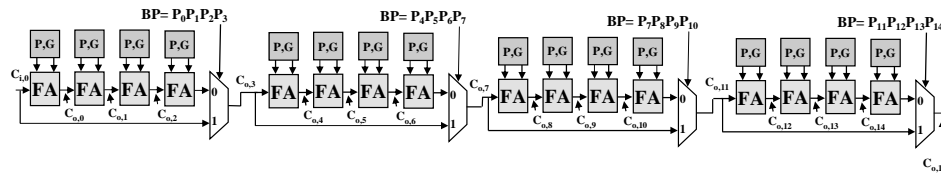
Adder with propagate and generate outputs

Later stages have increasingly complex logic.
Are later stages slower?





16-bit Carry Bypass Adder



Assume the following for delays of each block:

P, G from A, B: 1 delay unit

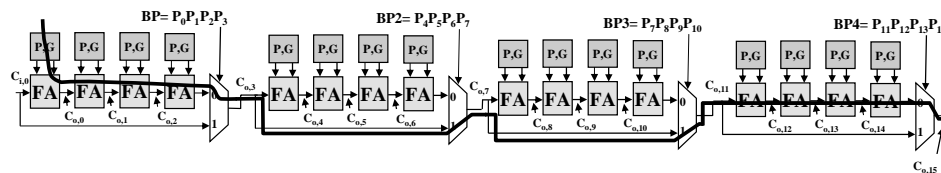
P, G, C_i to C_o or Sum for a FA: 1 delay unit

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***

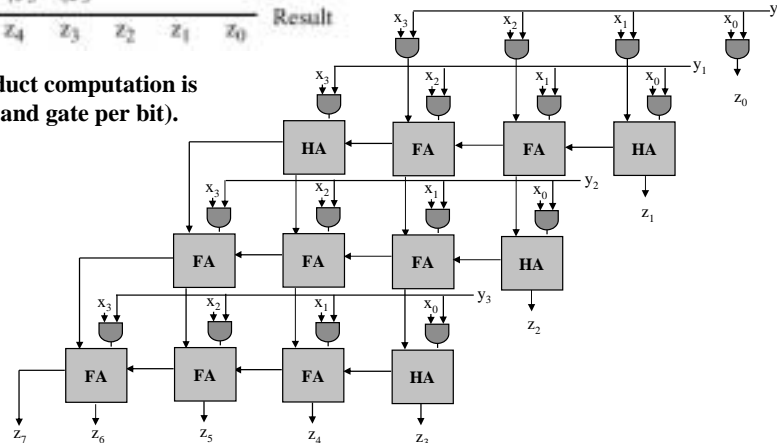


Binary Multiplication

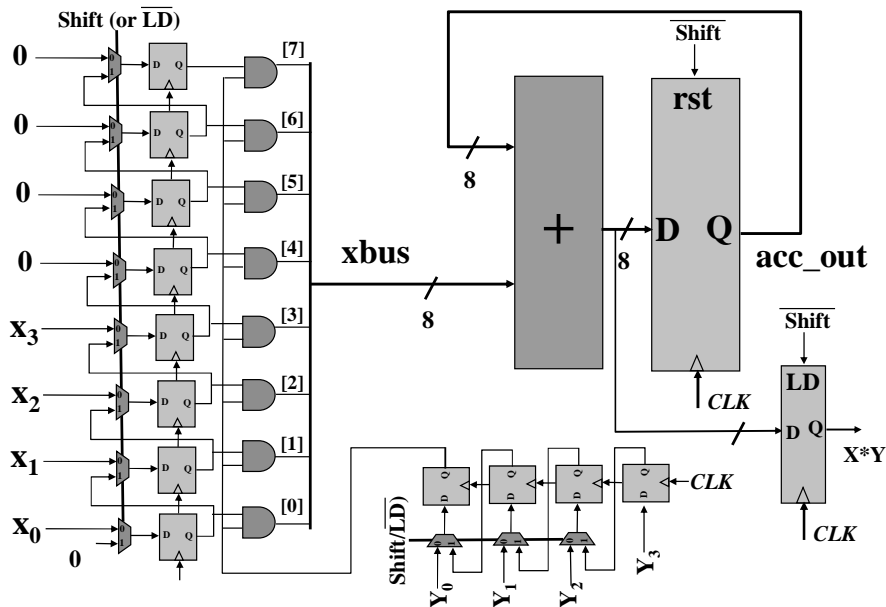


$$\begin{array}{r}
 \text{ Multiplicand} \\
 \times \text{ Multiplier} \\
 \hline
 x_3y_0 \ x_2y_0 \ x_1y_0 \ x_0y_0 \\
 x_3y_1 \ x_2y_1 \ x_1y_1 \ x_0y_1 \text{ Partial Product} \\
 x_3y_2 \ x_2y_2 \ x_1y_2 \ x_0y_2 \\
 + x_3y_3 \ x_2y_3 \ x_1y_3 \ x_0y_3 \\
 \hline
 z_7 \ z_6 \ z_5 \ z_4 \ z_3 \ z_2 \ z_1 \ z_0 \text{ Result}
 \end{array}$$

➤ Partial product computation is simple (single and gate per bit).

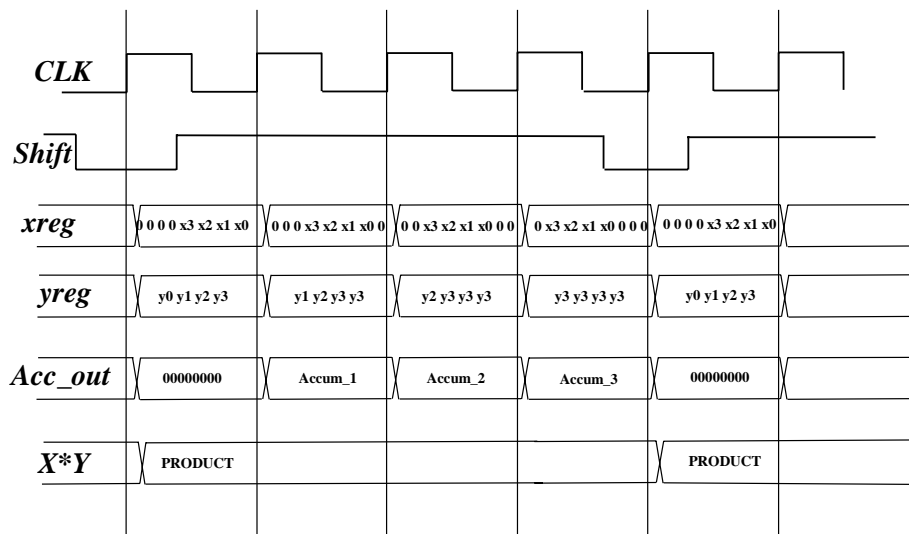


A Serial (Magnitude) Multiplier





Timing Diagram



VHDL of Serial Multiplier



```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity serialmult is
  port (SHIFT, CLK: in std_logic;
        X, Y : in std_logic_vector (3 downto 0);
        XY: out std_logic_vector (7 downto 0));
end serialmult ;
architecture behavioral of serialmult is
  signal XREG, XBUS, acc_out, XY_int, add_out:
    std_logic_vector (7 downto 0);
  signal YREG: std_logic_vector (3 downto 0);
begin
  add_out <= XBUS + acc_out;
  XY <= XY_int;
  XBUS <= (others => '0') when YREG(0) = '0' else XREG;

```



VHDL of Serial Multiplier (cont.)



```

process (CLK)
begin
  if (rising_edge(CLK)) then
    if (SHIFT = '0') then
      XREG <= "0000" & X;
      YREG <= Y;
      acc_out <= (others => '0');
      XY_int <= add_out;
    else
      XREG <= XREG (6 downto 0) & '0';
      YREG <= Y(3) & YREG (3 downto 1);
      acc_out <= add_out;
      XY_int <= XY_int; -- This is optional.
    end if;
  end if;
end process;
end architecture behavioral;

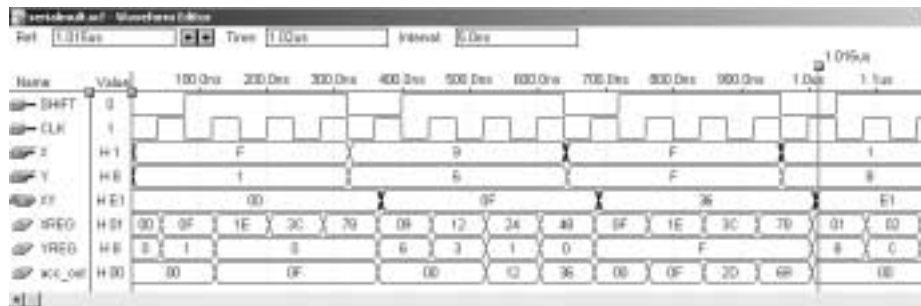
```



Simulation of Serial Multiplier



- Notice that SHIFT specifies a load followed by three shifts and brackets the rising edge of the CLK.
- The values for X and Y are loaded into XREG and YREG respectively.
- XREG shifts left while YREG shifts right.
- The value of XY is the product (in HEX) of X and Y.





Baugh Wooley Formulation



Assuming X and Y are 4-bit two's complement numbers:

$$X = -2^3x_3 + \sum_{i=0}^2 x_i2^i \quad Y = -2^3y_3 + \sum_{i=0}^2 y_i2^i$$

The product of X and Y is:

$$XY = x_3y_32^6 - \sum_{i=0}^2 x_iy_32^{i+3} - \sum_{j=0}^2 x_3y_j2^{j+3} + \sum_{i=0}^2 \sum_{j=0}^2 x_iy_j2^{i+j}$$

Recall for two's complement:

$$-\sum_{i=0}^2 z_i2^i = -2^3 + \sum_{i=0}^2 \bar{z}_i2^i + 1$$

The product then becomes:

$$\begin{aligned} XY &= x_3y_32^6 + \sum_{i=0}^2 x_iy_32^{i+3} + 2^3 - 2^6 + \sum_{j=0}^2 x_3y_j2^{j+3} + 2^3 - 2^6 + \sum_{i=0}^2 \sum_{j=0}^2 x_iy_j2^{i+j} \\ &= x_3y_32^6 + \sum_{i=0}^2 x_iy_32^{i+3} + \sum_{j=0}^2 x_3y_j2^{j+3} + \sum_{i=0}^2 \sum_{j=0}^2 x_iy_j2^{i+j} + 2^4 - 2^7 \\ &= -2^7 + x_3y_32^6 + (\bar{x}_2y_3 + \bar{x}_3y_2)2^5 + (\bar{x}_1y_3 + \bar{x}_3y_1 + x_2y_2 + 1)2^4 \\ &\quad + (\bar{x}_0y_3 + \bar{x}_3y_0 + x_1y_2 + x_2y_1)2^3 + (x_0y_2 + x_1y_1 + x_2y_0)2^2 + (x_0y_1 + x_1y_0)2^1 \\ &\quad + (x_0y_0)2^0 \end{aligned}$$



Two's Complement Multiplication

