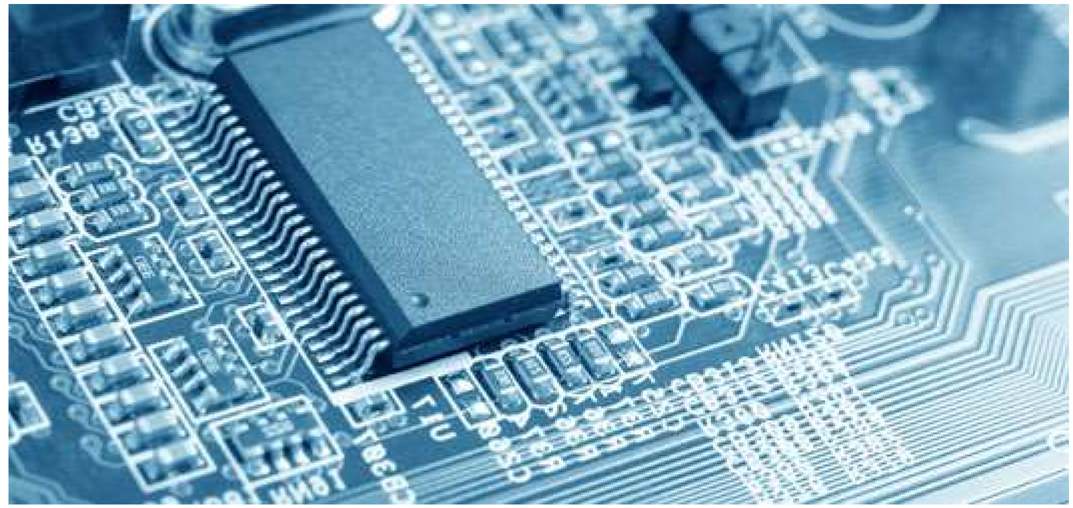
NAZARBAYEV UNIVERSITY
SCHOOL OF SCIENCE AND TECHNOLOGY



ROBT206 – Microcontrollers with Lab

Lectures 12 – Arithmetic Functions

20 February, 2018

Topics

Today's Topics

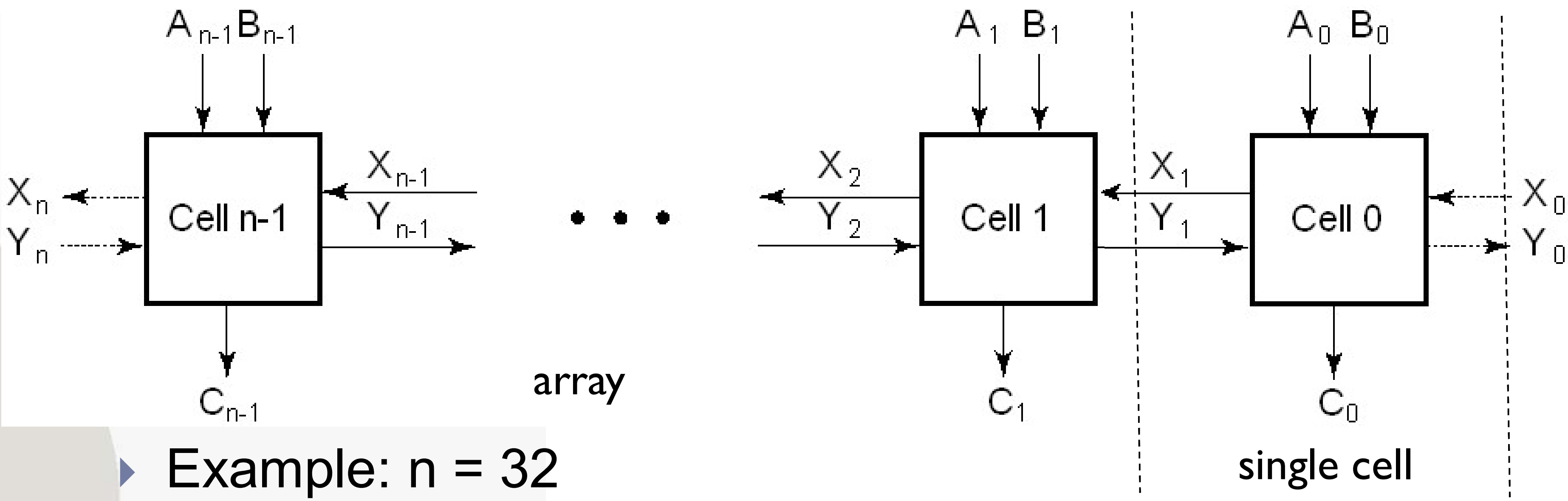
Iterative circuits
Binary Adders
    Half and full adders
    Ripple carry and carry look-ahead adders

Binary Subtraction

Iterative Combinational Circuits

- Arithmetic functions
  - Operate on binary vectors
  - Use the same subfunction in each bit position
- One can design a functional block for the subfunction and repeat it to obtain functional block for overall function
- *Iterative array* - an array of interconnected cells (1-D or 2-D arrays)

Block Diagram of a 1D Iterative Array



- Example: n = 32
  - Number of inputs = ?
  - Truth table rows = ?
  - Equations with up to ? input variables
  - Equations with huge number of terms
  - Design impractical!
- Iterative array takes advantage of the regularity to make design feasible: Divide and Conquer!

Arithmetic Functional Blocks: Addition

- Binary addition used frequently
- Addition Development:
  - *Half-Adder* (HA), a 2-input bit-wise addition functional block,
  - *Full-Adder* (FA), a 3-input bit-wise addition functional block,
  - *Ripple Carry Adder*, an iterative array to perform binary addition, and
  - *Carry-Look-Ahead Adder* (CLA), a hierarchical structure to improve performance.

Arithmetic Functional Block: Half-Adder

- A 2-input, 1-bit width binary adder that performs the following computations:
-

X	0	0	1	1
+Y	+ 0	+ 1	+ 0	+ 1
C S	0 0	0 1	0 1	1 0

- A half adder adds two bits to produce a two-bit sum
- The sum is expressed as a <u>sum bit</u> , S and a <u>carry bit</u>, C
- The half adder can be specified as a truth table for S and C $\Rightarrow$

X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Logic Simplification: Half-Adder

- The K-Map for S, C is:
- This is a pretty trivial map! By inspection:

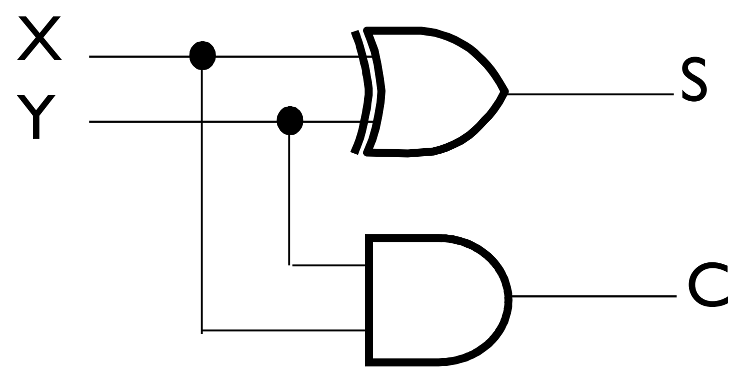
S		Y
	$_0$	1_1
X	1_2	$_3$

C		Y
	$_0$	$_1$
X	$_2$	1_3

$$\mathbf{S} = \mathbf{X} \cdot \overline{\mathbf{Y}} + \overline{\mathbf{X}} \cdot \mathbf{Y} = \mathbf{X} \oplus \mathbf{Y}$$

- and

$$\mathbf{C} = \mathbf{X} \cdot \mathbf{Y}$$



Arithmetic Functional Block: Full-Adder

- A full adder is similar to a half adder, but includes a carry-in bit from lower stages. Like the half-adder, it computes a sum bit, S and a carry bit, C.
  - For a carry-in (Z) of 0, it is the same as the half-adder:

Z	0	0	0	0
X	0	0	1	1
+Y	+ 0	+ 1	+ 0	+ 1
C S	0 0	0 1	0 1	1 0

  - For a carry- in (Z) of 1:

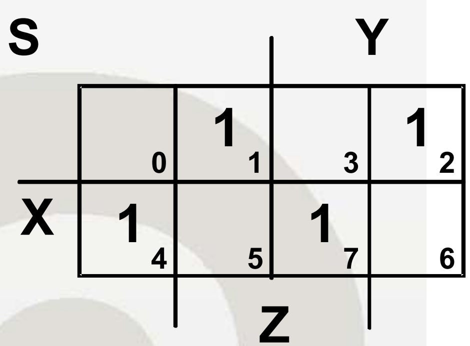
Z	1	1	1	1
X	0	0	1	1
+Y	+ 0	+ 1	+ 0	+ 1
C S	0 1	1 0	1 0	1 1

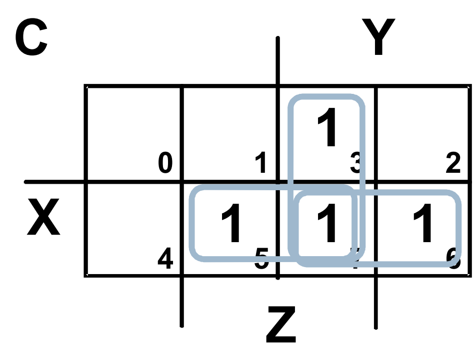
Logic Optimization: Full-Adder

- Full-Adder Truth Table:

X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

- Full-Adder K-Map:





Equations: Full-Adder

- From the K-Map, we get:

$$\mathbf{S} = \mathbf{X}\overline{\mathbf{Y}}\,\overline{\mathbf{Z}} + \overline{\mathbf{X}}\mathbf{Y}\,\overline{\mathbf{Z}} + \overline{\mathbf{X}}\,\overline{\mathbf{Y}}\mathbf{Z} + \mathbf{X}\mathbf{Y}\mathbf{Z}$$

$$\mathbf{C} = \mathbf{X}\mathbf{Y} + \mathbf{X}\mathbf{Z} + \mathbf{Y}\mathbf{Z}$$

- The S function is the three-bit XOR function (Odd Function):

$$\mathbf{S} = \mathbf{X} \oplus \mathbf{Y} \oplus \mathbf{Z}$$

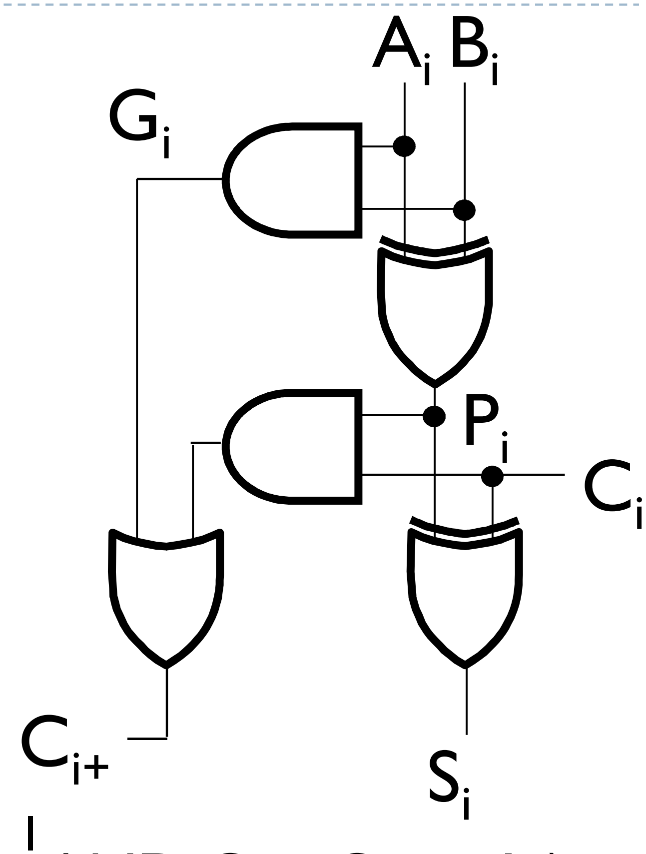
- The Carry bit C is 1 if both X and Y are 1 (the sum is 2), or if the sum is 1 and a carry-in (Z) occurs. Thus C can be re-written as:

$$\mathbf{C} = \mathbf{X}\mathbf{Y} + (\mathbf{X} \oplus \mathbf{Y})\mathbf{Z}$$

- The term $X \cdot Y$ is ***carry generate***.
- The term $X \oplus Y$ is ***carry propagate***.

Implementation: Full Adder

- Full Adder Schematic
- Here X, Y, and Z, and C (from the previous pages) are A, B, C_i and C_o, respectively. Also,
  G = generate and
  P = propagate.
- Note: This is really a combination of a 3-bit odd function (for S)) and Carry logic (for C_o):



(G = Generate) OR (P =Propagate AND C_i = Carry In)

$$C_o = G + P \cdot C_i$$

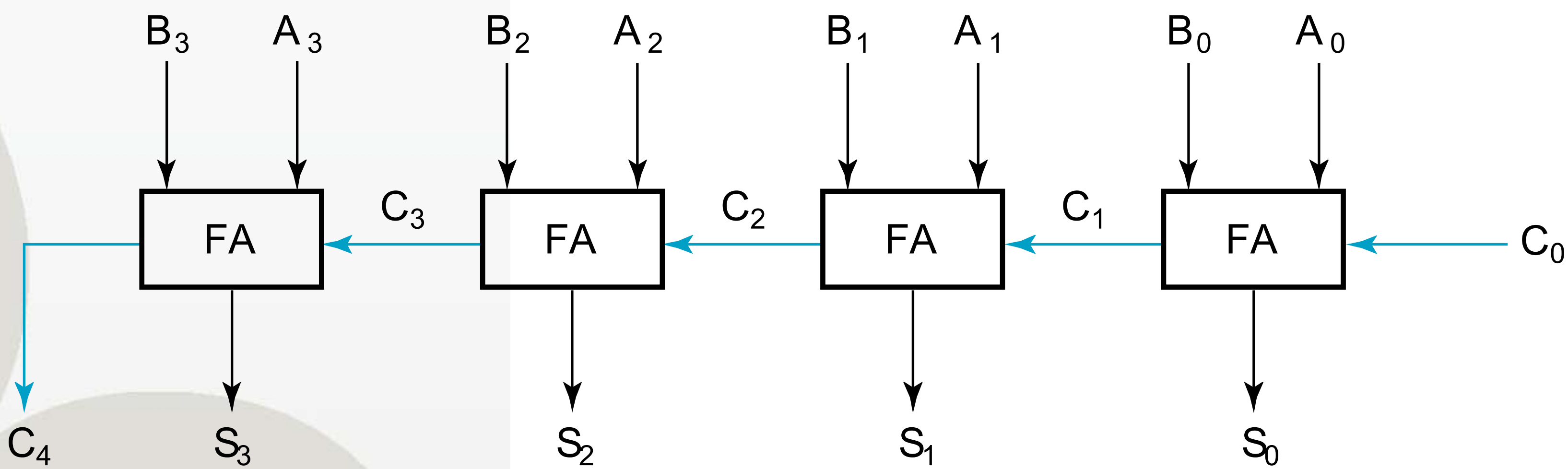
Binary Adders

- To add multiple operands, we "bundle" logical signals together into vectors and use functional blocks that operate on the vectors
- Example: 4-bit ripple carry adder: Adds input vectors A(3:0) and B(3:0) to get a sum vector S(3:0)
- Note: carry out of cell i becomes carry in of cell i + 1

Description	Subscript 3 2 1 0	Name
Carry In	0 1 1 0	C_i
Augend	1 0 1 1	A_i
Addend	0 0 1 1	B_i
Sum	1 1 1 0	S_i
Carry out	0 0 1 1	C_i+1

4-bit Ripple-Carry Binary Adder

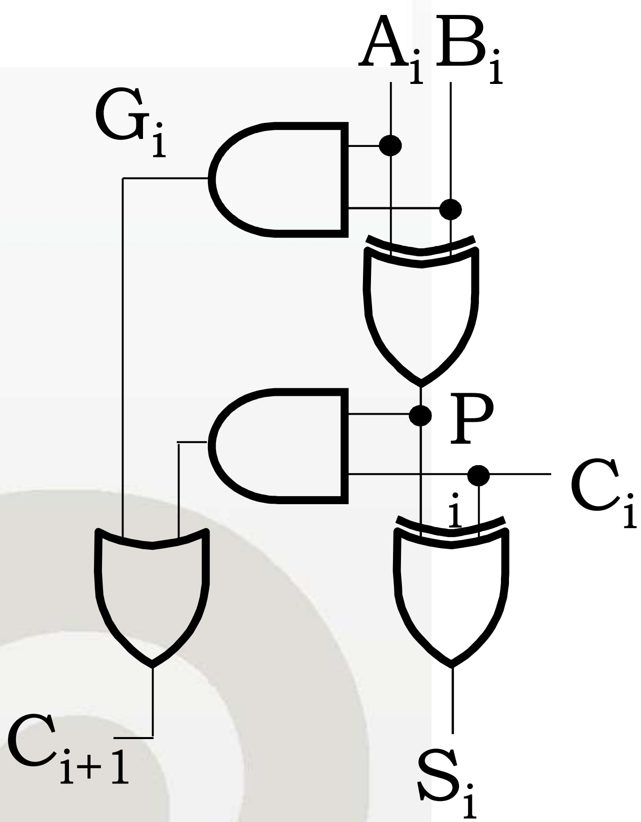
- A four-bit Ripple Carry Adder made from four 1-bit Full Adders:



Slow adder: many delays from input to output

Delay of a Full Adder

- Assume that AND, OR gates have 1 gate delay and the XOR has 2 gate delays
- Delay of the Sum and Carry bit:



$$S_i = A_i \oplus B_i \oplus C_i$$

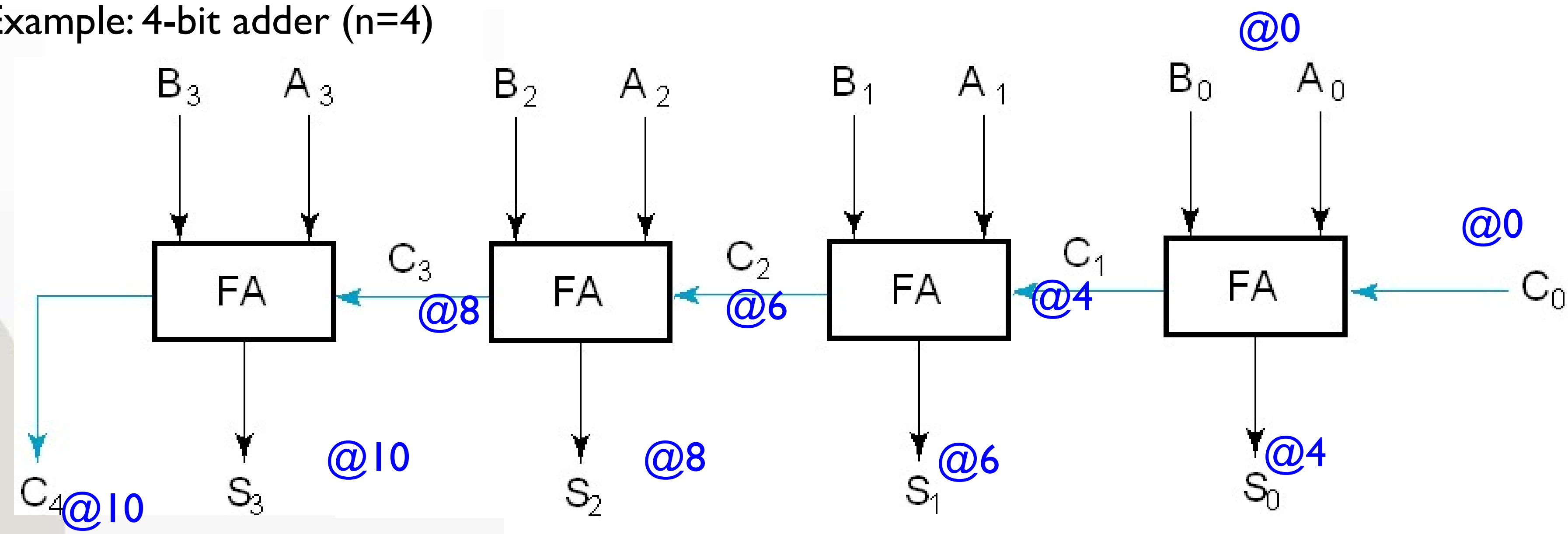
$$S_0 = \underbrace{\underbrace{A_0 \oplus B_0}_{\text{2 delays}} \oplus C_0}_{2+2=4\text{ delays}}$$

$$C_{i+1} = A_i\, B_i + (A_i \oplus B_i)\, C_i$$

$$C_1 = \underbrace{A_0\, B_0 + \underbrace{\underbrace{(A_0 \oplus B_0)}_{@2}\, C_0}_{@3}}_{2+2=4\text{ delays}}$$

Delay in a Ripple-carry adder

Example: 4-bit adder (n=4)



One problem with the addition of binary numbers is the length of time to propagate the ripple carry from the least significant bit to the most significant bit.

Example: 32-bit Ripple-carry has a unit gate delay of 1ns.

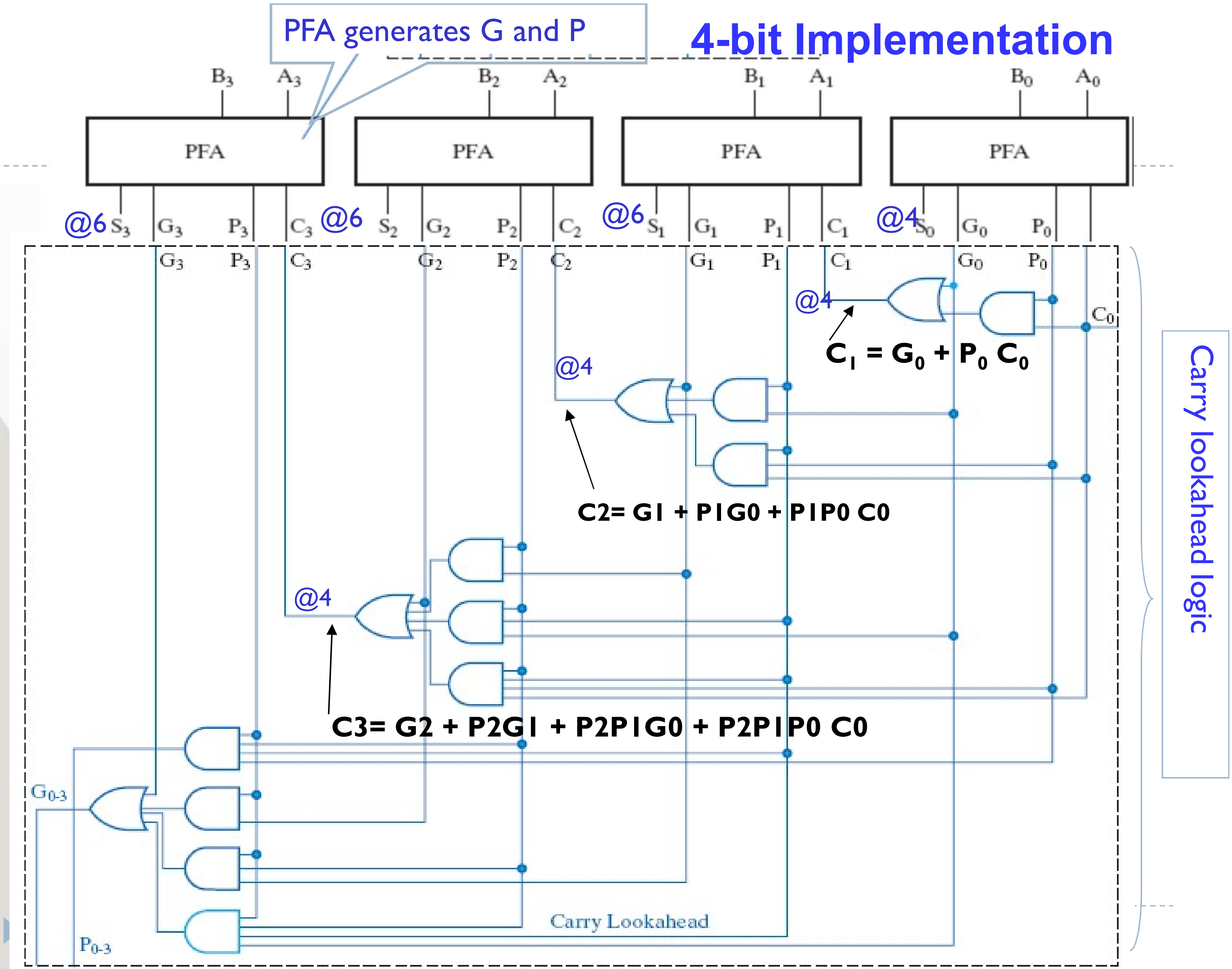
•What is the total delay of the adder?

•What is the max frequency at which it can be clocked?

Carry Lookahead Adder

- Uses a different circuit to calculate the carry out (calculates it ahead), to speed up the overall addition
- Requires more complex circuits.
- Trade-off: speed vs. area (complexity, cost)

4-bit Implementation



Unsigned Subtraction

- Algorithm:
  - Subtract N from M
  - If no end borrow occurs, then $M \geq N$, and the result is a non-negative number and correct.
  - If an end borrow occurs, then $N > M$ and the difference

  $\mathbf{M - N + 2^n}$ is subtracted from 2^n, and a minus sign is appended to the result.

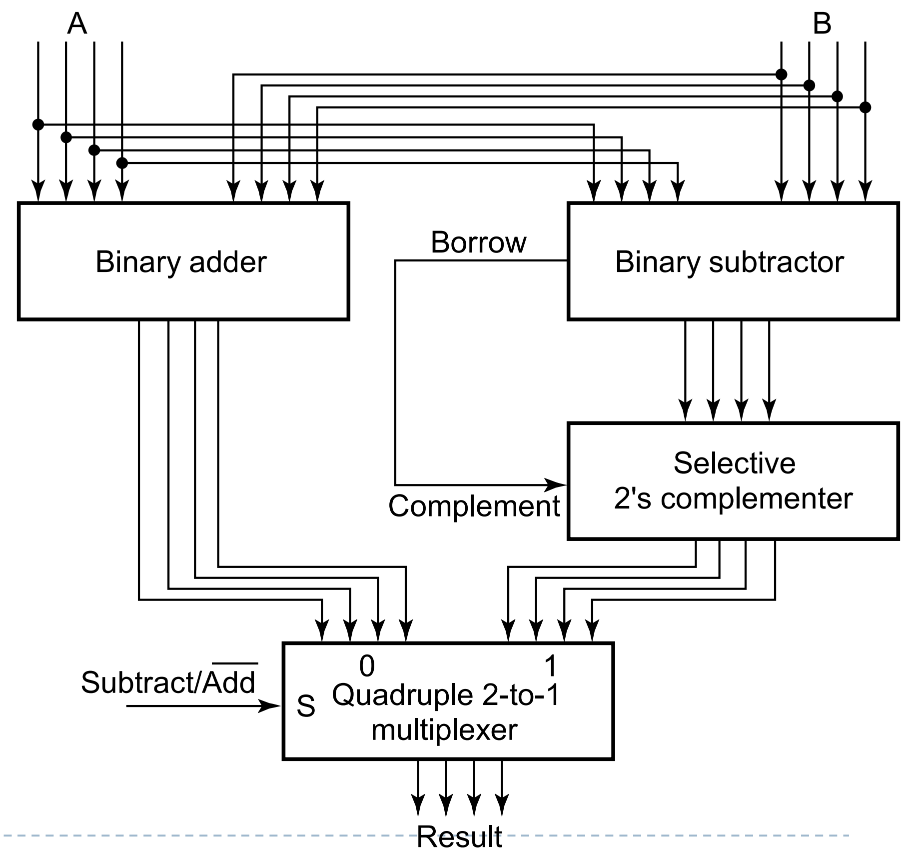
- Examples:

$$\begin{array}{r} 0 \quad\; \\ 1001 \\ -\underline{0111} \\ 0010 \end{array} \qquad \begin{array}{r} 1 \quad\; \\ 0100 \\ -\underline{0111} \\ 1101 \end{array}$$

$$\begin{array}{r} 10000 \\ -\underline{1101} \\ (-)\ 0011 \end{array}$$

Unsigned Subtraction (continued)

- The subtraction, $2^n - N$, is taking the 2's complement of N
- To do both unsigned addition and unsigned subtraction requires:
- Quite complex!
- Goal: Shared simpler logic for both addition and subtraction
- Introduce complements as an approach



Complements

- Two complements:
  - Diminished Radix Complement of N
    - $(r - 1)$'s complement for radix r
    - 1's complement for radix 2
    - Defined as $(r^n - 1) - N$
  - Radix Complement
    - r's complement for radix r
    - 2's complement in binary
    - Defined as $r^n - N$
- Subtraction is done by adding the complement of the subtracted number
- If the result is negative, takes its 2's complement

Binary 1's Complement

- For $r = 2$, $N = 01110011_2$, $n = 8$ (8 digits):
  $(r^n - 1) = 256 - 1 = 255_{10}$ or 11111111_2
- The 1's complement of 01110011_2 is then:

$$\begin{array}{r} 11111111 \\ -\ \underline{01110011} \\ 10001100 \end{array}$$

- Since the $2^n - 1$ factor consists of all 1's and since $1 - 0 = 1$ and $1 - 1 = 0$, the 1's complement is obtained by complementing each individual bit (bitwise NOT).

Binary 2's Complement

- For $r = 2$, $N = 01110011_2$, $n = 8$ (8 digits), we have:
  $(r^n) = 256_{10}$ or 100000000_2
- The 2's complement of 01110011 is then:

$$\begin{array}{r} 100000000 \\ -\underline{01110011} \\ 10001101 \end{array}$$

- Note the result is the 1's complement plus 1, a fact that can be used in designing hardware

Alternate 2's Complement Method

- Given: an *n*-bit binary number, beginning at the least significant bit and proceeding upward:
  - Copy all least significant 0's
  - Copy the first 1
  - Complement all bits thereafter.
- 2's Complement Example:

  $10010\underline{100}$

  - Copy underlined bits:

    $\underline{100}$

  - and complement bits to the left:

    $\underline{01101}100$

Subtraction with 2's Complement

- For n-digit, unsigned numbers M and N, find M – N in base 2:
  - Add the 2's complement of N to M:
    $$M + (2^n - N) = M - N + 2^n$$
  - If $M \geq N$, the sum produces end carry 2^n which is discarded; from above, M – N remains.
  - If M < N, the sum does not produce an end carry, since it is equal to $2^n - (N - M)$, the 2's complement of (N– M).
  - To obtain the result – (N – M) , take the 2's complement of the sum and place a – to its left.

2's Complement Subtraction Example 1

- Find $01010100_2 - 01000011_2$

$$
\begin{array}{r} 01010100 \\ -\ \underline{01000011} \end{array}
\xrightarrow{\textbf{2's comp}}
\begin{array}{rr} 1 & 01010100 \\ + & \underline{10111101} \\ & 00010001 \end{array}
$$

- The carry of 1 indicates that no correction of the result is required.

2's Complement Subtraction Example 2

- Find $01000011_2 - 01010100_2$

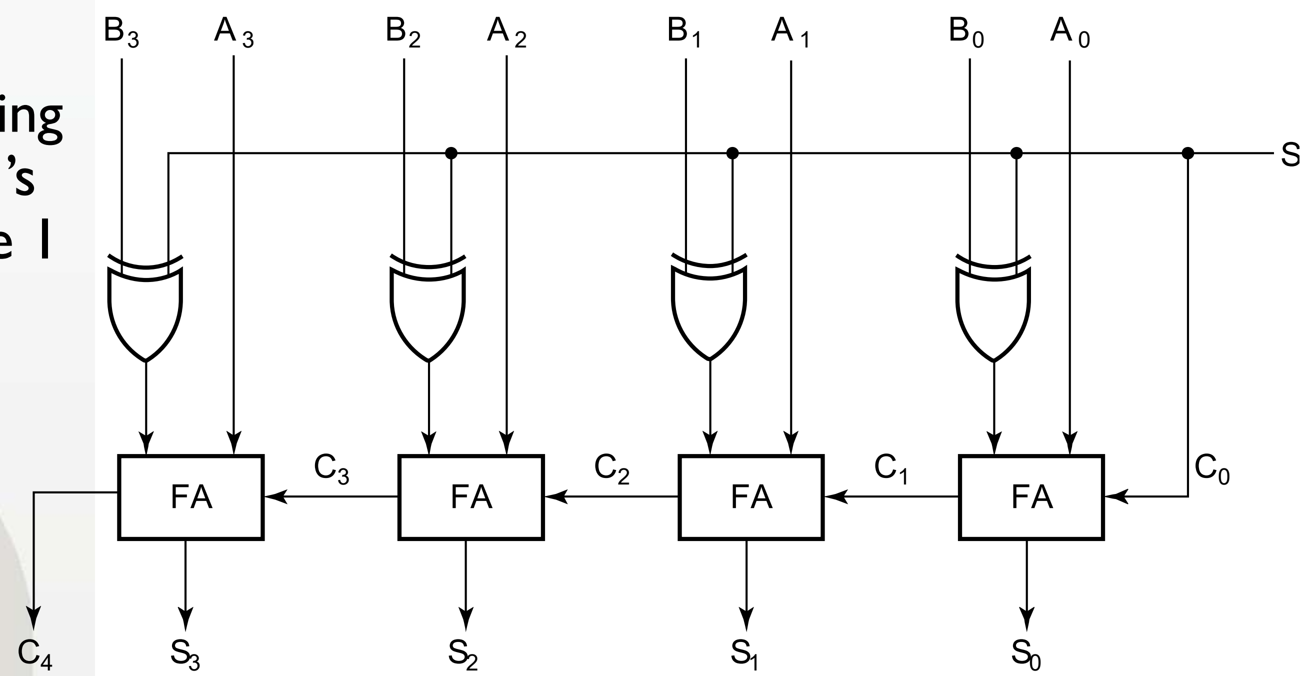
$$
\begin{array}{r}
01000011 \\
-\ \underline{01010100}
\end{array}
\xrightarrow{\text{2's comp}}
\begin{array}{r}
{}^{0}\ \ 01000011 \\
+\ \underline{10101100} \\
11101111
\end{array}
$$

$$
\xrightarrow{\text{2's comp}} 00010001
$$

- The carry of 0 indicates that a correction of the result is required.
- Result = – (00010001)

2's Complement Adder/Subtractor

- Subtraction can be done by addition of the 2's Complement.
  1. Complement each bit (1's Complement.)
  2. Add 1 to the result.
- The circuit shown computes A + B and A – B:
- For S = 1, subtract, the 2's complement of B is formed by using XORs to form the 1's comp and adding the 1 applied to C_0.
- For S = 0, add, B is passed through unchanged



Signed Integers

- Positive numbers and zero can be represented by unsigned n-digit, radix *r* numbers. We need a representation for negative numbers.
- To represent a sign (+ or –) we need exactly one more bit of information (1 binary digit gives $2^1 = 2$ elements which is exactly what is needed).
- Since computers use binary numbers, by convention, the most significant bit is interpreted as a sign bit:

$$s\ a_{n-2}\ \ldots\ a_2 a_1 a_0$$

where:

$s = 0$ for Positive numbers

$s = 1$ for Negative numbers

and $a_i = 0$ or 1 represent the magnitude in some form.

Exercise

- Give the sign+magnitude, 1's complement and 2's complement of (using minimal required bits):

	Sign+Mag	One's compl.	Two's compl.
+2	010	010	010
- 2	110	101	110
+3	011	011	011
- 3	111	100	101
+0	000	000	000
- 0	100	111	000

Signed 2's complement system

- Positive numbers are unchanged
- Negative numbers: take 2's complement
- Example for 4-bit word:

0	0000		
+1	0001	-1	1111
+2	0010	-2	1110
+3	0011	-3	1101
+4	0100	-4	1100
+5	0101	-5	1011
+6	0110	-6	1010
+7	0111	-7	1001
		-8	1000

•0 indicates positive and 1 negative numbers

•7 positive numbers and 8 negative ones

2's Complement Arithmetic

- Addition: Simple rule
  - Represent negative number by its 2's complement. Then, add the numbers including the sign bits, discarding a carry out of the sign bits (2's complement):
  - Indeed, e.x. $M+(-N) \rightarrow M + (2^n-N)$
    - If $M \geq N$: $(M-N) + \cancel{2^n}$ ignore carry out: $M-N$ is the answer in two's complement)
    - If $M \leq N$: $(M-N) + 2^n = 2^n - (N-M)$ which is 2's complement of the (negative) number $(M-N)$: $-(N-M)$.
- Subtraction: $M-N \rightarrow M + (2^n-N)$

  Form the complement of the number you are subtracting and follow the rules for addition.

Signed 2's Complement Examples

- Example 1: $\begin{array}{r} 1101 \\ +\underline{0011} \end{array}$

- Example 2: $\begin{array}{r} 1101 \\ -\underline{\ 0011} \end{array}$

- Example 3: $(5 - 11)_{10}$ (using 2's compl.)

Any Questions?