

Chapter 14: Arithmetic Modules

Prof. Ming-Bo Lin

Department of Electronic Engineering

National Taiwan University of Science and Technology

Syllabus

- ❖ Objectives
- ❖ Addition and subtraction
- ❖ Multiplication
- ❖ Division
- ❖ Arithmetic and logic unit

Objectives

After completing this chapter, you will be able to:

- ❖ Describe both addition and subtraction modules
- ❖ Understand the principles of carry-look-ahead (CLA) adder
- ❖ Understand the essential ideas of parallel-prefix adders
- ❖ Describe the basic operations of multiplication
- ❖ Describe the basic operations of division
- ❖ Describe the designs of arithmetic-logic unit (ALU)

Syllabus

- ❖ Objectives
- ❖ Addition and subtraction
 - Carry-look-ahead (CLA) adders
 - Parallel-prefix adders
- ❖ Multiplication
- ❖ Division
- ❖ Arithmetic and logic unit

Bottleneck of Ripple-Carry Adder

❖ Bottleneck of n -bit ripple-carry adder

- The generation of carries

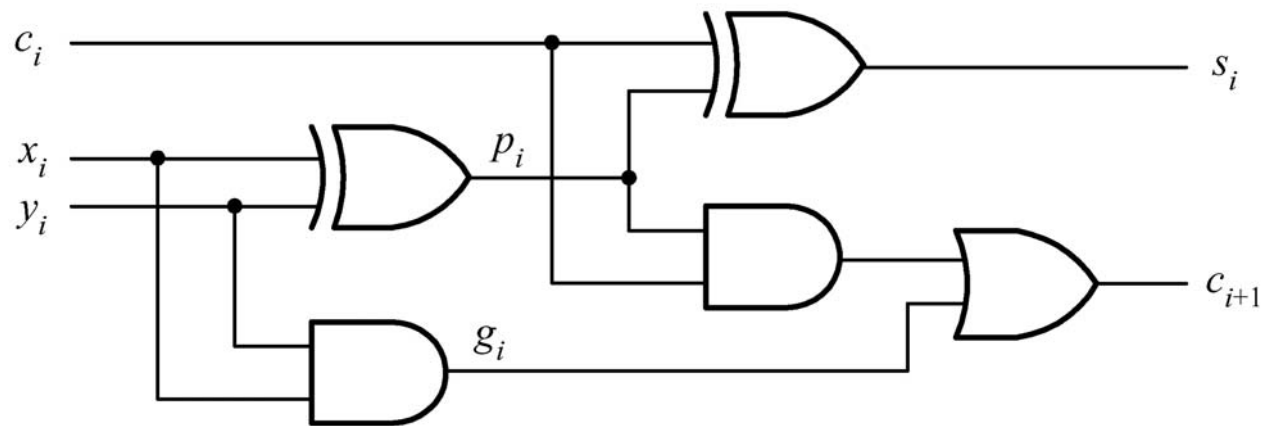
❖ Ways of carry generation

- carry-look-ahead (CLA) adder
- parallel-prefix adders:
 - Kogge-Stone adder
 - Brent-Kung adder
- others

A CLA adder

❖ Definition

- carry generate (g_i): $g_i = x_i \cdot y_i$
- carry propagate (p_i): $p_i = x_i \oplus y_i$



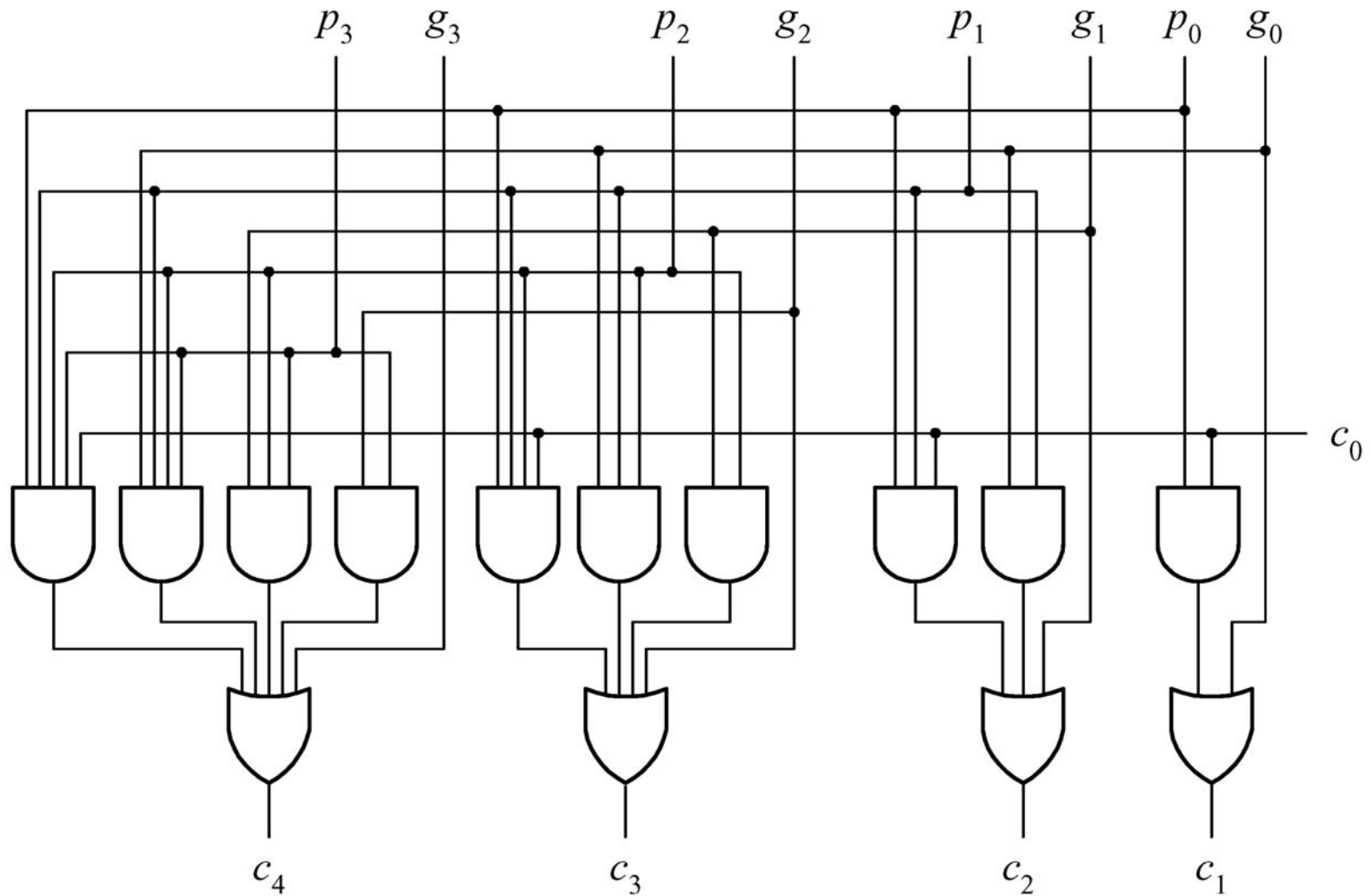
$$s_i = p_i \oplus c_i$$

$$c_1 = g_0 + p_0 c_0$$

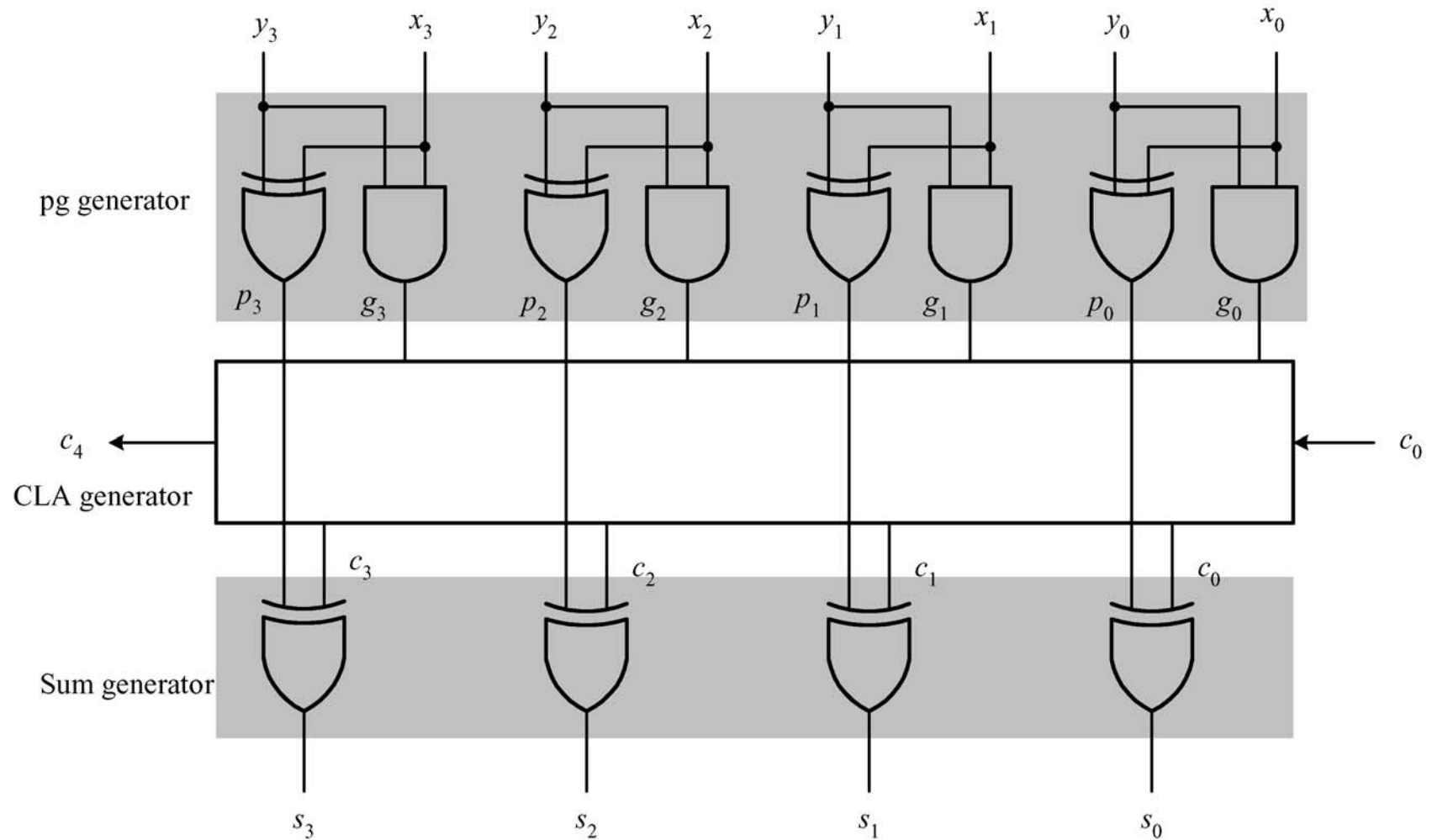
$$c_{i+1} = g_i + p_i \cdot c_i$$

$$\begin{aligned} c_2 &= g_1 + p_1 c_1 = g_1 + p_1 (g_0 + p_0 c_0) \\ &= g_1 + p_1 g_0 + p_1 p_0 c_0 \end{aligned}$$

A Carry-Lookahead Generator



A CLA Adder



A CLA Adder

```
// a 4-bit CLA adder using assign statements
module cla_adder_4bits(x, y, cin, sum, cout);
// inputs and outputs
input  [3:0] x, y;
input  cin;
output [3:0] sum;
output cout;

// internal wires
wire p0,g0, p1,g1, p2,g2, p3,g3;
wire c4, c3, c2, c1;

// compute the p for each stage
assign p0 = x[0] ^ y[0], p1 = x[1] ^ y[1],
       p2 = x[2] ^ y[2], p3 = x[3] ^ y[3];
```

A CLA Adder

```
// compute the g for each stage
assign g0 = x[0] & y[0], g1 = x[1] & y[1],
       g2 = x[2] & y[2], g3 = x[3] & y[3];
// compute the carry for each stage
assign c1 = g0 | (p0 & cin),
       c2 = g1 | (p1 & g0) | (p1 & p0 & cin),
       c3 = g2 | (p2 & g1) | (p2 & p1 & g0) | (p2 & p1 & p0 & cin),
       c4 = g3 | (p3 & g2) | (p3 & p2 & g1) | (p3 & p2 & p1 & g0) |
           (p3 & p2 & p1 & p0 & cin);
// compute Sum
assign sum[0] = p0 ^ cin, sum[1] = p1 ^ c1,
       sum[2] = p2 ^ c2, sum[3] = p3 ^ c3;
// assign carry output
assign cout = c4;
endmodule
```

A CLA Adder --- Using generate statements

```
// an n-bit CLA adder using generate loops
module cla_adder_generate(x, y, cin, sum, cout);
// inputs and outputs
parameter N = 4; //define the default size
input  [N-1:0] x, y;
input  cin;
output [N-1:0] sum;
output cout;
```

```
// internal wires
wire [N-1:0] p, g;
wire [N:0]   c;
// assign input carry
assign c[0] = cin;
```

Virtex 2 XC2V250 FG456 -6

n	4	8	16	32
f (MHz)	104.3	78.9	53.0	32.0
LUTs	8	16	32	64

A CLA Adder --- Using generate statements

```
genvar i;
generate for (i = 0; i < N; i = i + 1) begin: pq_cla
    assign p[i] = x[i] ^ y[i];
    assign g[i] = x[i] & y[i];
end endgenerate // compute generate and propagation

generate for (i = 1; i < N+1; i = i + 1) begin: carry_cla
    assign c[i] = g[i-1] | (p[i-1] & c[i-1]);
end endgenerate // compute carry for each stage

generate for (i = 0; i < N; i = i + 1) begin: sum_cla
    assign sum[i] = p[i] ^ c[i];
end endgenerate // compute sum

assign cout = c[n]; // assign final carry

...
```

Syllabus

- ❖ Objectives
- ❖ Addition and subtraction
 - Carry-look-ahead (CLA) adder
 - **Parallel-prefix adders**
- ❖ Multiplication
- ❖ Division
- ❖ Arithmetic and logic unit

Parallel-Prefix Adders

❖ The prefix sums

$$\blacksquare s_{[i,0]} = x_i \odot x_{i-1} \odot \dots \odot x_1 \odot x_0$$

where $0 \leq i < n$

❖ From bits k to i

$$g_{[i,k]} = g_{[i,j+1]} + p_{[i,j+1]} \cdot g_{[j,k]}$$

$$p_{[i,k]} = p_{[i,j+1]} \cdot p_{[j,k]}$$

where $0 \leq i < n, k \leq j < i, 0 \leq k < n$

$$g_{[i,i]} = x_i \cdot y_i$$

$$p_{[i,i]} = x_i \oplus y_i$$

Parallel-Prefix Adders

- ❖ The carry of i th-bit adder $c_i = g_{i-1} + p_{i-1} \cdot c_{i-1}$ can be written as

$$g_{[i,0]} = g_{[i,j+1]} + p_{[i,j+1]} \cdot g_{[j,0]}$$

- ❖ Define group $g_{[i,j]}$ and $p_{[i,j]}$ as a group and denoted as

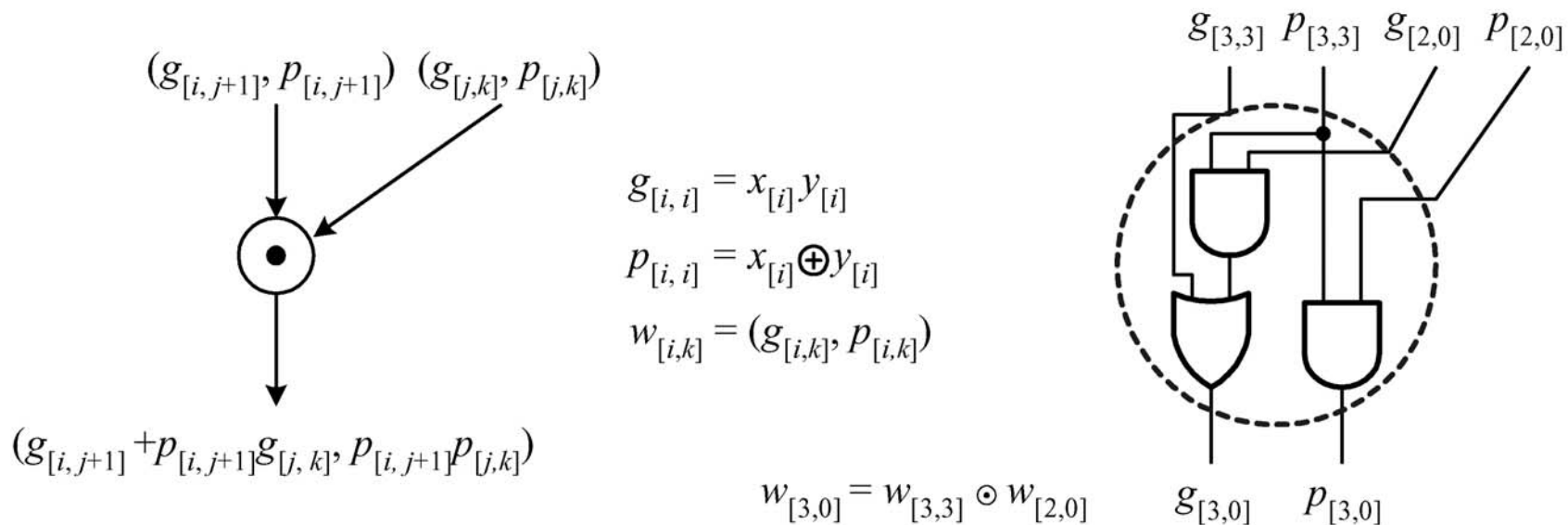
$$w_{[i,j]} = (g_{[i,j]}, p_{[i,j]})$$

Parallel-Prefix Adders

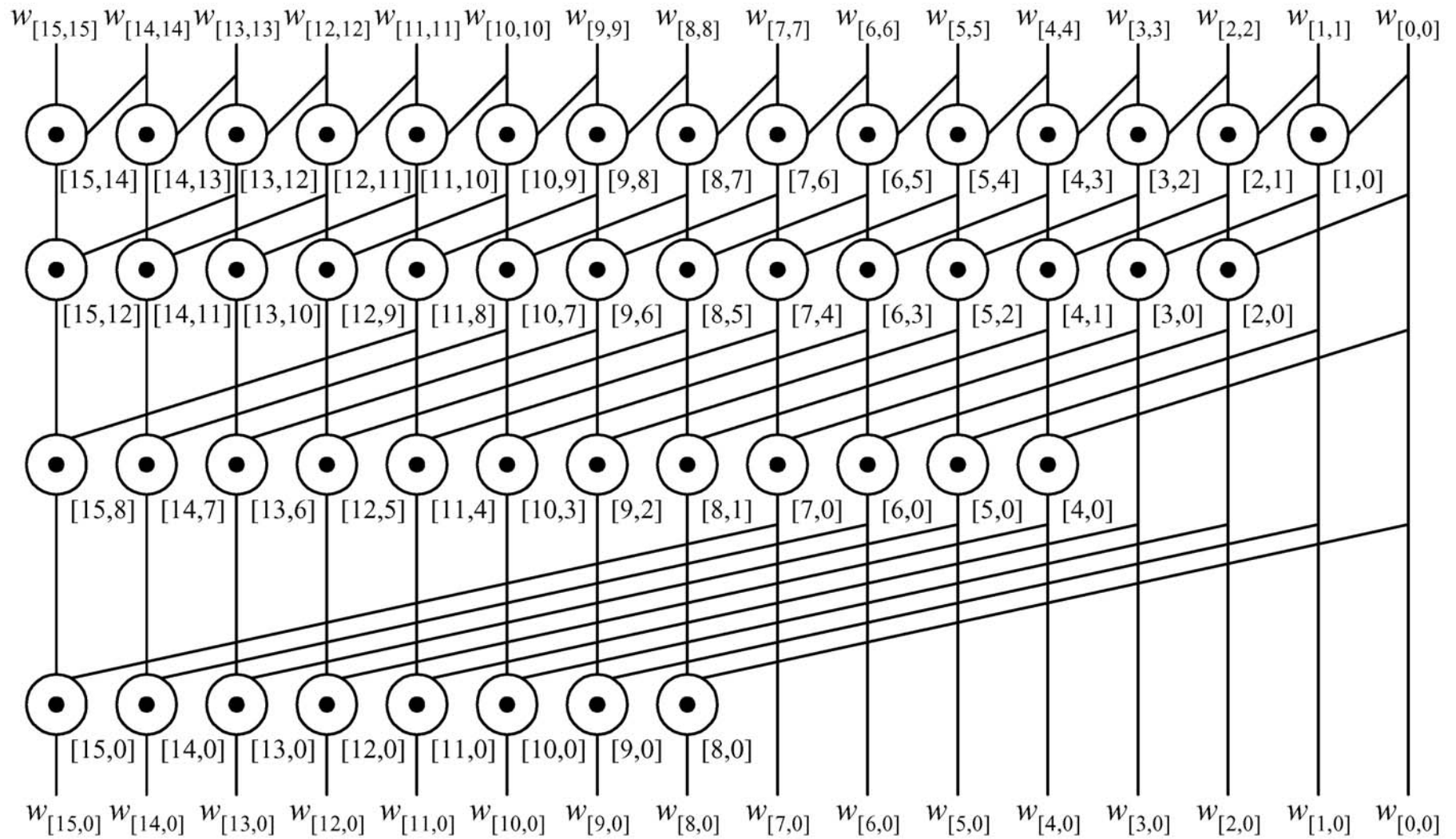
- ❖ The operator \odot in $w_{[i,k]} = w_{[i,j+1]} \odot w_{[j,k]}$ is a binary associative operator.

$$c_i = g_{i-1} + p_{i-1} \cdot c_{i-1}$$

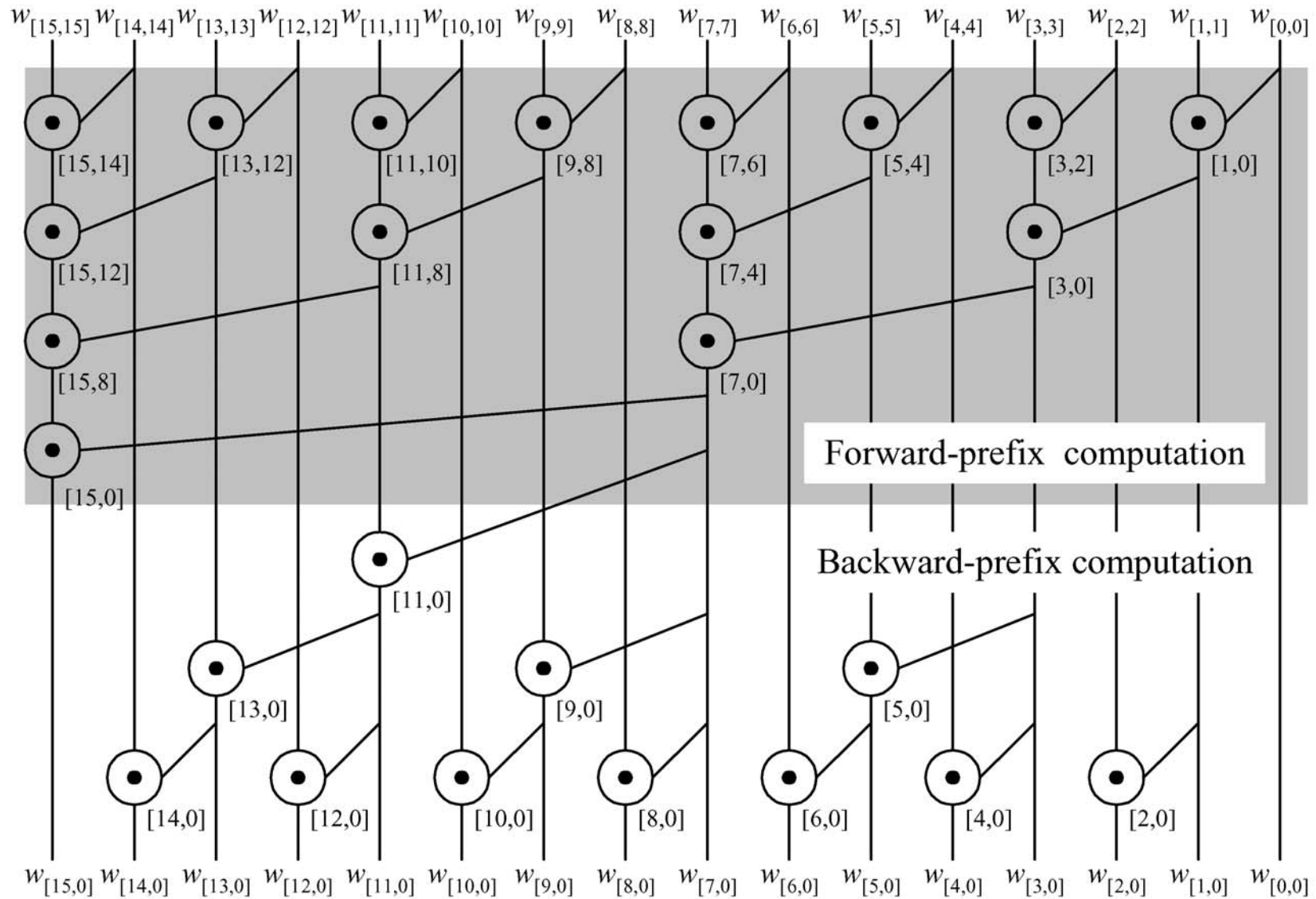
$$g_{[i,0]} = g_{[i,j+1]} + p_{[i,j+1]} \cdot g_{[j,0]}$$



Kogge-Stone Adder



Brent-Kung Adder



Parallel-Prefix Adders

- ❖ An n -input **Kogge-Stone parallel-prefix network**
 - a propagation delay of $\log_2 n$ levels
 - a cost of $n \log_2 n - n + 1$ cells
- ❖ An n -input **Brent-Kung parallel-prefix network**
 - a propagation delay of $2 \log_2 n - 2$ levels and
 - a cost of $2n - 2 - \log_2 n$ cells

Syllabus

- ❖ Objectives
- ❖ Addition and subtraction
- ❖ Multiplication
 - Shift-and-add multiplication
 - Basic array multipliers
 - A signed array multiplier
- ❖ Division
- ❖ Arithmetic and logic unit

Shift-and-Add Multiplication

Algorithm: Shift-and-add multiplication

Input: An m -bit multiplicand and an n -bit multiplier.

Output: The $(m + n)$ -bit product.

Begin

1. Load multiplicand and multiplier into registers M and Q , respectively;
clear register A and set loop count CNT equal to n .

2. **repeat**

2.1 if ($Q[0] == 1$) **then** $A = A + M$;

2.2 Right shift register pair $A : Q$ one bit;

2.3 $CNT = CNT - 1$;

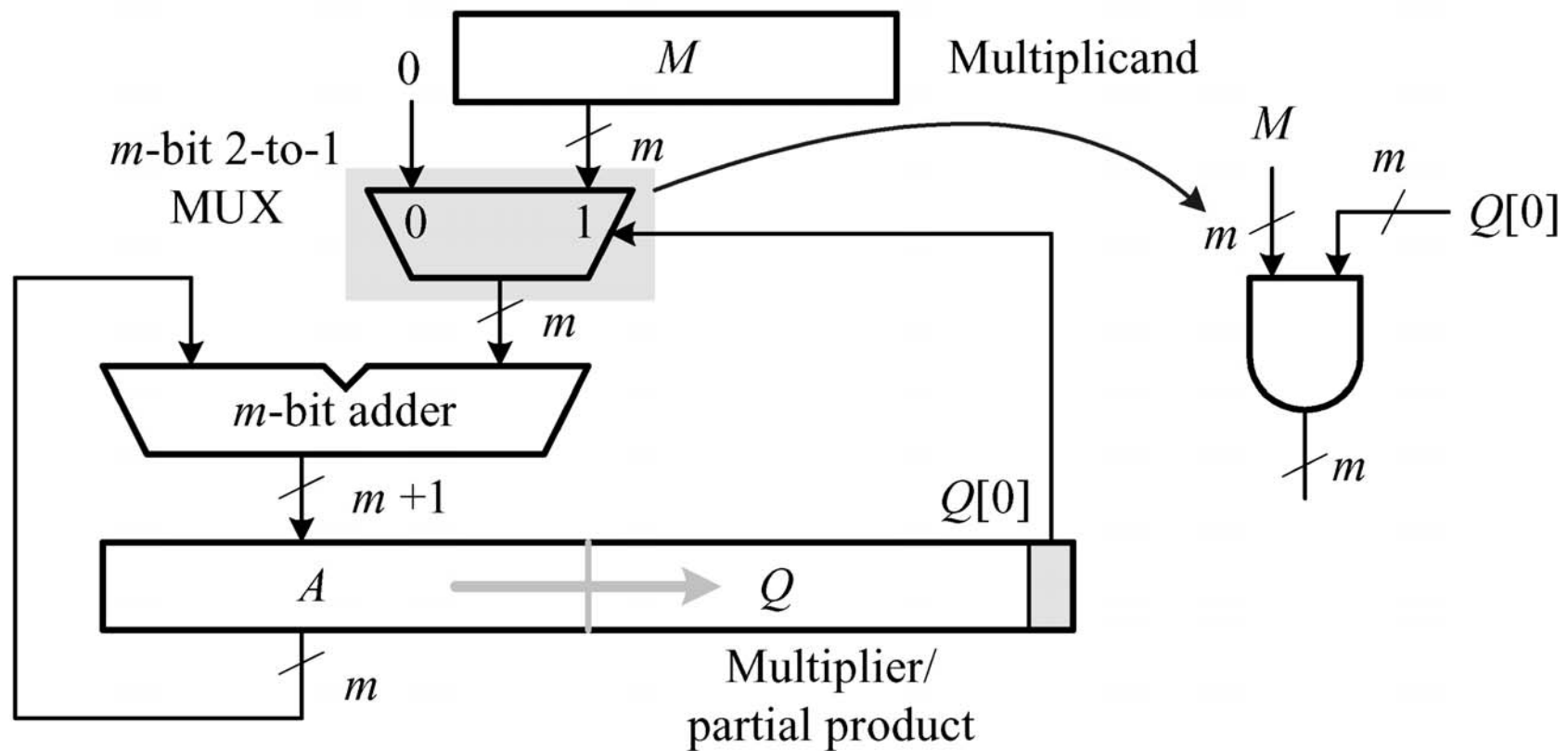
until ($CNT == 0$);

End



Shift-and-Add Multiplication

❖ A sequential implementation



Syllabus

- ❖ Objectives
- ❖ Addition and subtraction
- ❖ Multiplication
 - Shift-and-add multiplication
 - Basic array multipliers
 - A signed array multiplier
- ❖ Division
- ❖ Arithmetic and logic unit

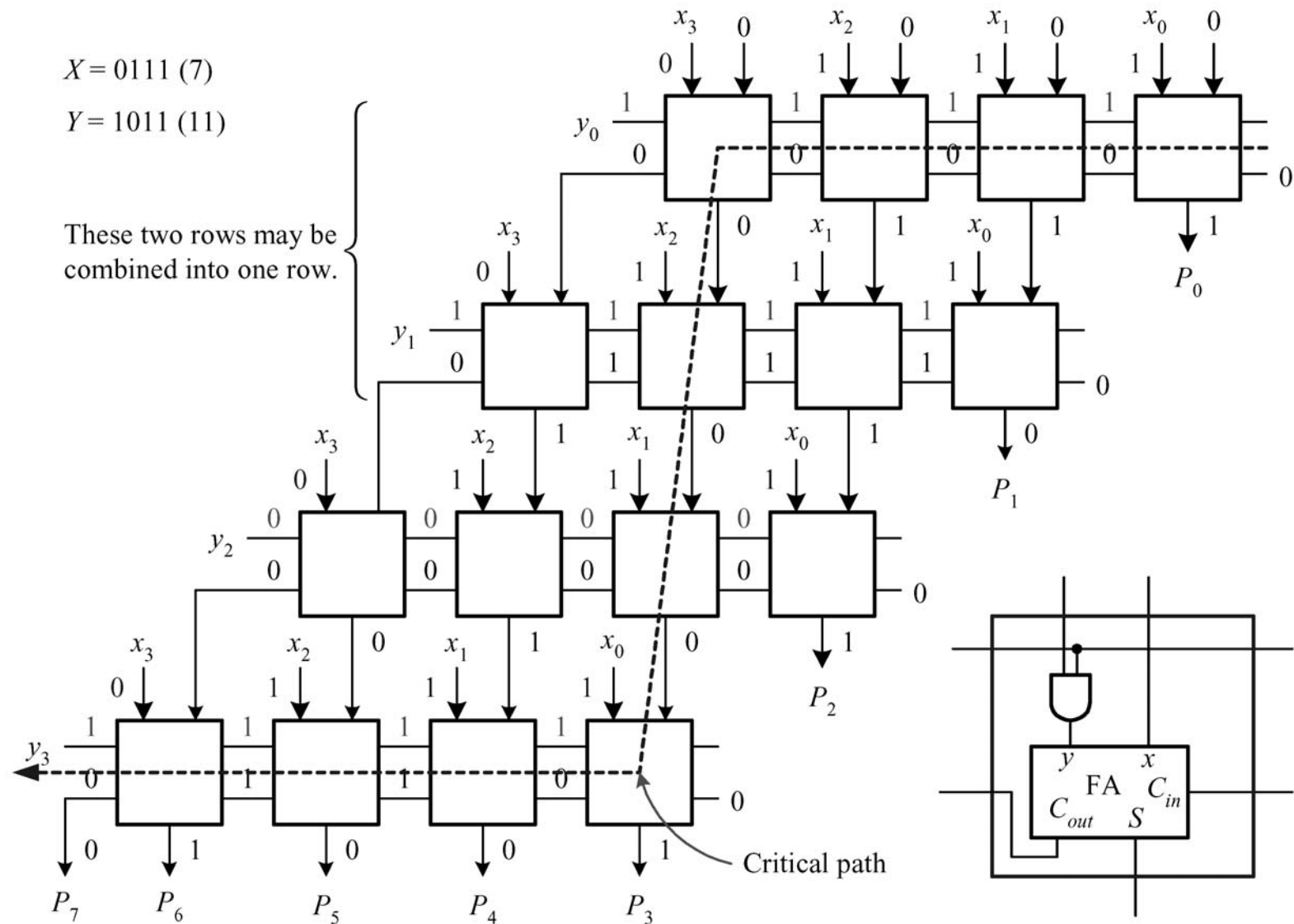
A Basic Array Multiplier

❖ An iterative logic structure

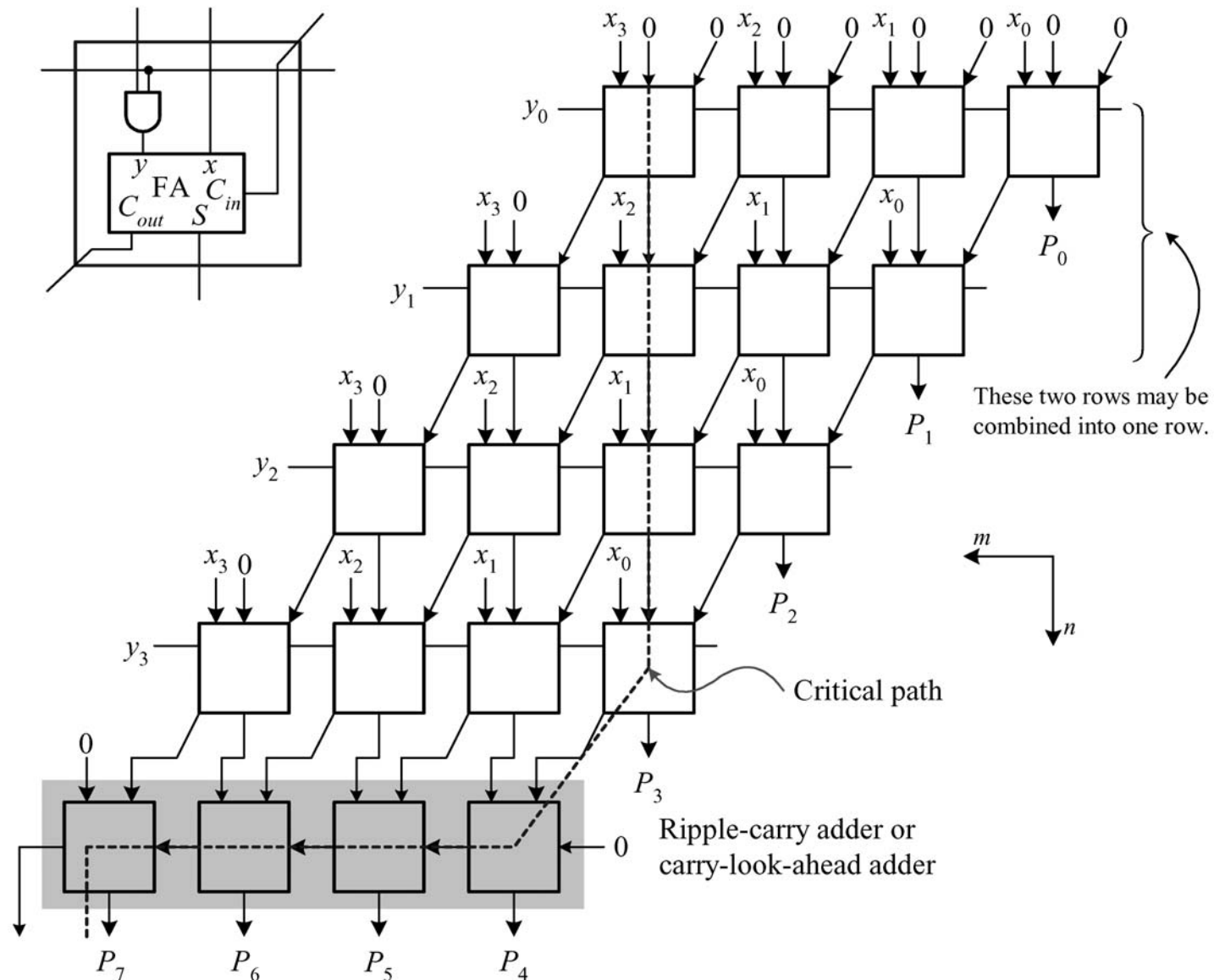
$$\begin{array}{rcccccc}
 & & x_3 & x_2 & x_1 & x_0 & = X \text{ (multiplicand)} \\
 \times & & y_3 & y_2 & y_1 & y_0 & = Y \text{ (multiplier)} \\
 \hline
 & & x_3y_0 & x_2y_0 & x_1y_0 & x_0y_0 & \\
 & & & x_3y_1 & x_2y_1 & x_1y_1 & x_0y_1 \\
 & & & & x_3y_2 & x_2y_2 & x_1y_2 & x_0y_2 \\
 + & & & & & x_3y_3 & x_2y_3 & x_1y_3 & x_0y_3 \\
 \hline
 & P_6 & P_5 & P_4 & P_3 & P_2 & P_1 & P_0 & \text{Product}
 \end{array}
 \left. \begin{array}{l} x_3y_0 \ x_2y_0 \ x_1y_0 \ x_0y_0 \\ x_3y_1 \ x_2y_1 \ x_1y_1 \ x_0y_1 \\ x_3y_2 \ x_2y_2 \ x_1y_2 \ x_0y_2 \end{array} \right\} \text{Partial product}$$

$$P = X \times Y = \sum_{i=0}^{m-1} x_i \cdot 2^i \cdot \sum_{j=0}^{n-1} y_j \cdot 2^j = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (x_i \cdot y_j) \cdot 2^{i+j} = \sum_{k=0}^{m+n-1} (P_k) \cdot 2^k$$

A Basic Unsigned Array Multiplier



An Unsigned CSA Array Multiplier



Syllabus

- ❖ Objectives
- ❖ Addition and subtraction
- ❖ Multiplication
 - Shift-and-add multiplication
 - Basic array multipliers
 - A signed array multiplier
- ❖ Division
- ❖ Arithmetic and logic unit

A Signed Array Multiplier

❖ Let X and Y be two two's complement number

$$X = -x_{m-1}2^{m-1} + \sum_{i=0}^{m-2} x_i 2^i$$

$$Y = -y_{n-1}2^{n-1} + \sum_{j=0}^{n-2} y_j 2^j$$

$$P = XY$$

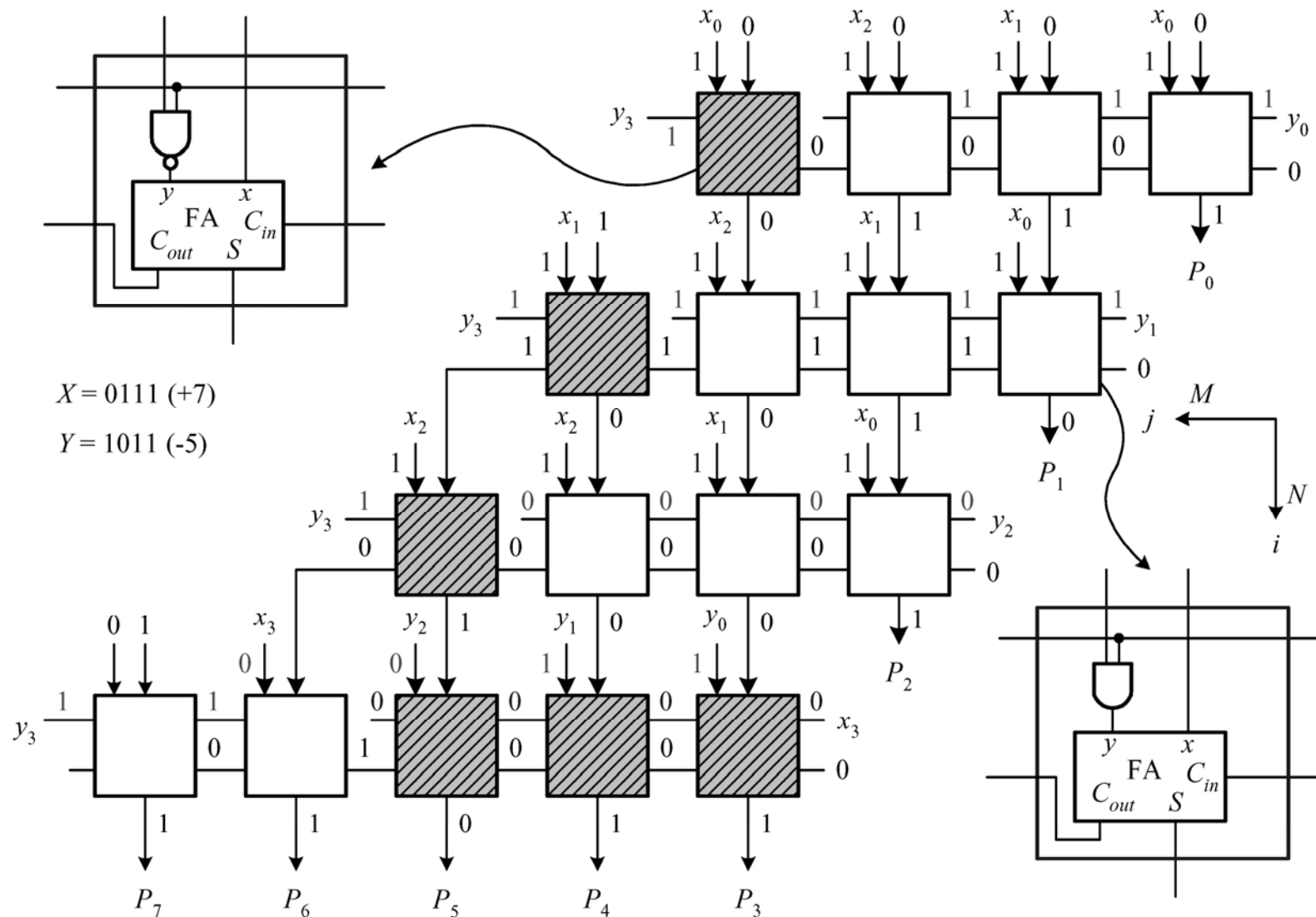
$$= \left(-x_{m-1}2^{m-1} + \sum_{i=0}^{m-2} x_i 2^i \right) \left(-y_{n-1}2^{n-1} + \sum_{j=0}^{n-2} y_j 2^j \right)$$

$$= \sum_{i=0}^{m-2} \sum_{j=0}^{n-2} x_i y_j 2^{i+j} + x_{m-1} y_{n-1} 2^{m+n-2} - \left(\sum_{i=0}^{m-2} x_i y_{n-1} 2^{i+n-1} + \sum_{j=0}^{n-2} x_{m-1} y_j 2^{j+m-1} \right)$$

A Signed Array Multiplier

$$\begin{array}{rccccccccc}
 & & & & x_3 & x_2 & x_1 & x_0 & = X & \text{(multiplicand)} \\
 & & & & \times & y_3 & y_2 & y_1 & y_0 & = Y & \text{(multiplier)} \\
 & & & 1 & \overline{y_3 x_0} & x_2 y_0 & x_1 y_0 & x_0 y_0 & & & \\
 & & \overline{y_3 x_1} & x_2 y_1 & x_1 y_1 & x_0 y_1 & & & & & \\
 & & \overline{y_3 x_2} & x_2 y_2 & x_1 y_2 & x_0 y_2 & & & & & \\
 + & 1 & y_3 x_3 & \overline{y_2 x_3} & \overline{y_1 x_3} & \overline{y_0 x_3} & & & & & \\
 \hline
 & P_7 & P_6 & P_5 & P_4 & P_3 & P_2 & P_1 & P_0 & & \text{Product}
 \end{array}
 \left. \begin{array}{l} \\ \\ \\ \end{array} \right\} \text{Partial product}$$

A Signed Array Multiplier



Syllabus

- ❖ Objectives
- ❖ Addition and subtraction
- ❖ Multiplication
- ❖ Division
 - Nonrestoring division algorithm
 - Implementations
- ❖ Arithmetic and logic unit

An Unsigned Non-restoring Division Algorithm

Algorithm: Unsigned non-restoring division

Input: An n -bit dividend and an m -bit divisor.

Output: The quotient and remainder.

Begin


1. Load divisor and dividend into registers M and D , respectively;
clear partial-remainder register R and

An Unsigned Non-restoring Division Algorithm

set loop count CNT equal to $n - 1$.

2. Left shift register pair $R : D$ one bit.
3. Compute $R = R - M$;
4. **repeat**
 - 4.1 **if** ($R < 0$) **begin**
 $D[0] = 0$; left shift $R : D$ one bit; $R = R + M$; **end**
 else begin
 $D[0] = 1$; left shift $R : D$ one bit; $R = R - M$; **end**
 - 4.2 $CNT = CNT - 1$;
5. **if** ($R < 0$) **begin** $D[0] = 0$; $R = R + M$; **end else** $D[0] = 1$;

End



An Unsigned Nonrestoring Division Example

divisor(M)
 0 1 1 0

$85_{10} = 01010101$

$6_{10} = 0110$

2's complement of 6
 = 1010

① or ① represents quotient bit

Hence quotient = 00001110

remainder = 0001

dividend (D)

	0	0	0	0	1	0	1	0	1	0	1
	1	0	1	0	↓	↓	↓	↓	↓	↓	↓
①	1	0	1	0	1						
	0	1	1	0	↓						
①	1	0	1	1	0						
	0	1	1	0	↓						
①	1	1	0	0	1						
	0	1	1	0	↓						
①	1	1	1	1	0						
	0	1	1	0	↓						
①	0	1	0	0	1						
	1	0	1	0	↓						
①	0	0	1	1	0						
	1	0	1	0	↓						
①	0	0	0	0	1						
	1	0	1	0	↓						
①	1	0	1	1							
	0	1	1	0	↓						
	0	0	0	1							

Remainder → 0 0 0 1

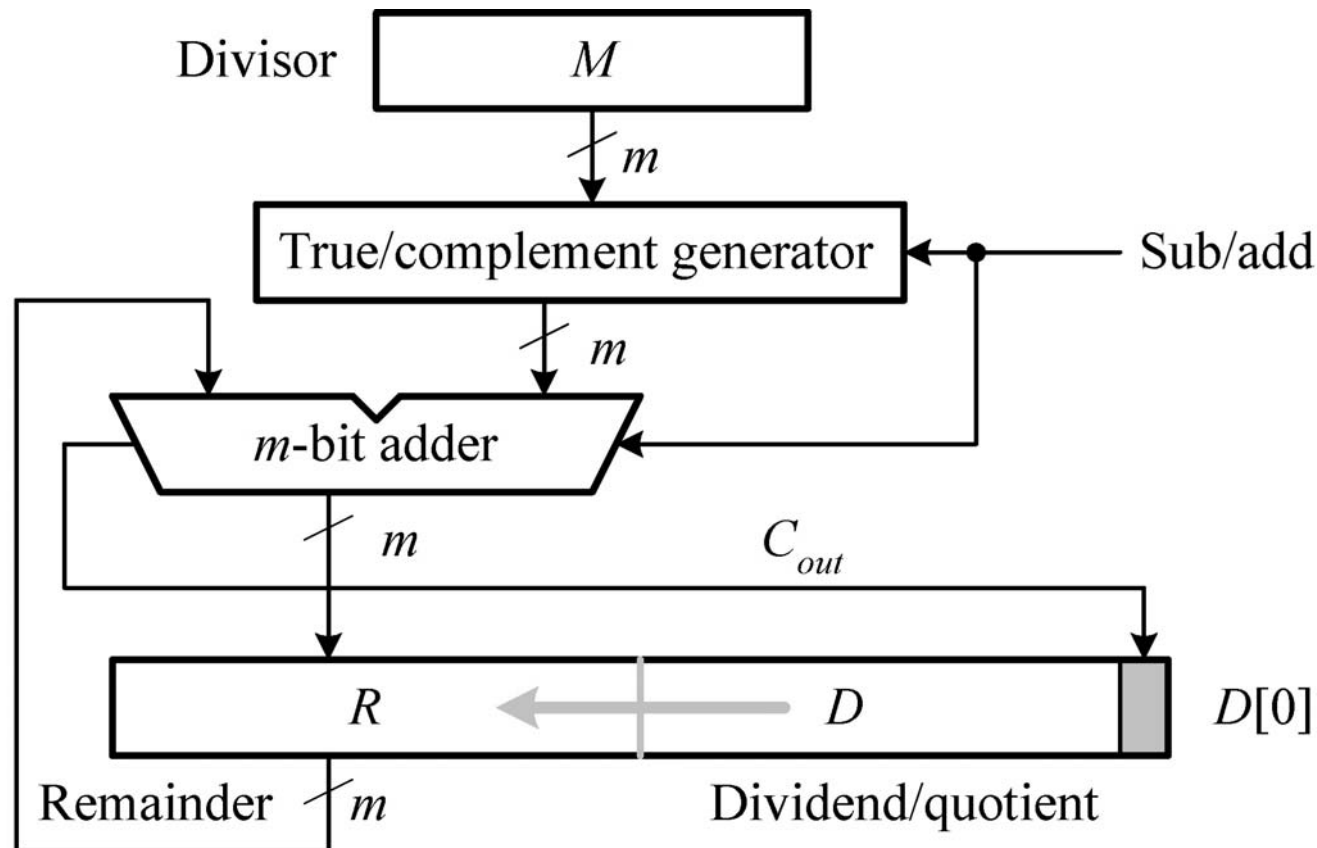
$D - M$
 $< 0, Q = 0$
 right shift $M, D + M$
 $< 0, Q = 0$
 right shift $M, D + M$
 $< 0, Q = 0$
 right shift $M, D + M$
 $< 0, Q = 0$
 right shift $M, D + M$
 $< 0, Q = 0$
 right shift $M, D + M$
 $> 0, Q = 1$
 right shift $M, D - M$
 $> 0, Q = 1$
 right shift $M, D - M$
 $> 0, Q = 1$
 right shift $M, D - M$
 $< 0, Q = 0$
 $D + M$

Syllabus

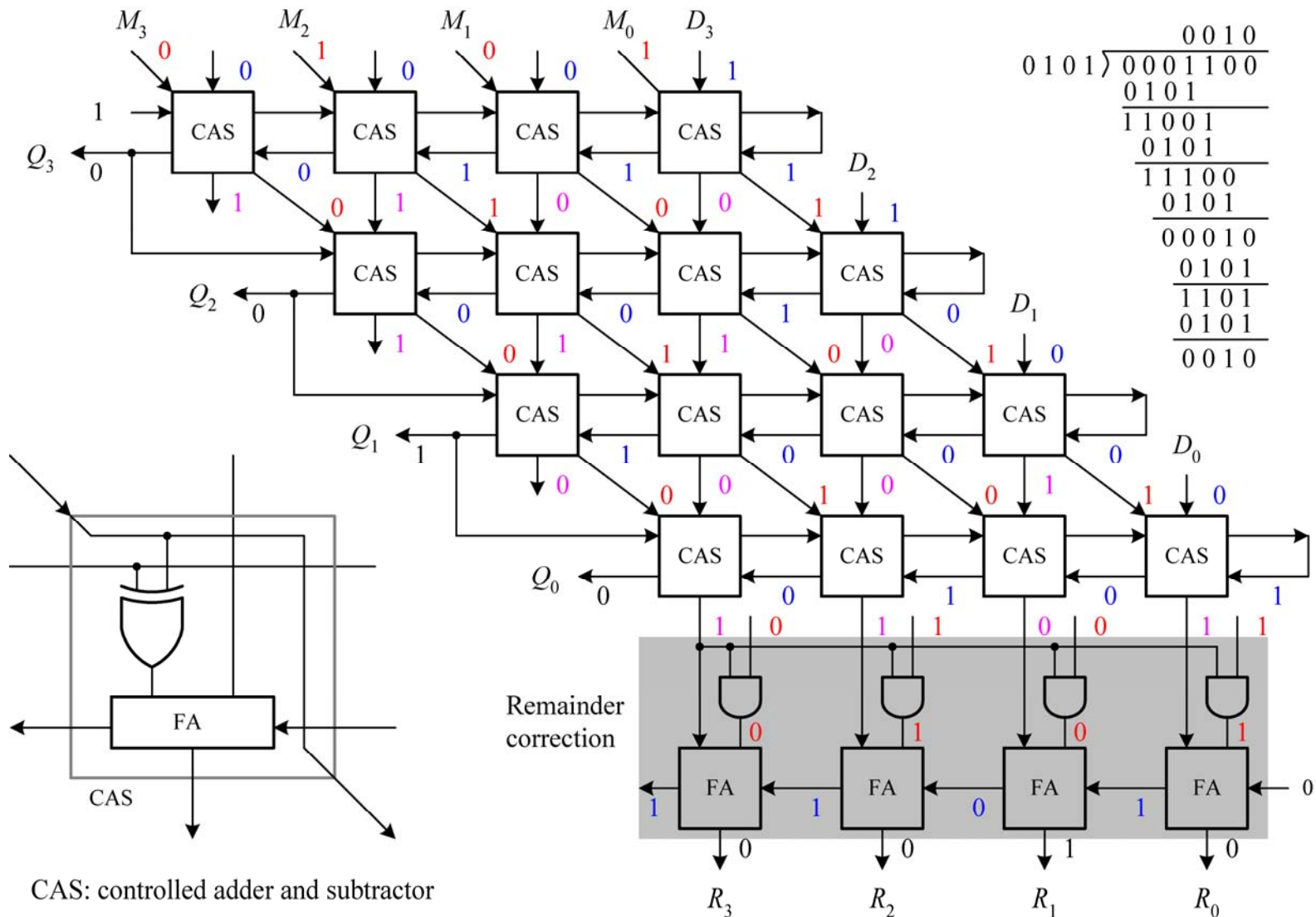
- ❖ Objectives
- ❖ Addition and subtraction
- ❖ Multiplication
- ❖ Division
 - Nonrestoring division algorithm
 - Implementations
- ❖ Arithmetic and logic unit

A Sequential Unsigned Non-restoring Division

❖ A sequential implementation



An Unsigned Array Non-restoring Divider



Syllabus

- ❖ Objectives
- ❖ Addition and subtraction
- ❖ Multiplication
- ❖ Division
- ❖ Arithmetic and logic unit
 - Basic functions
 - Implementations

Arithmetic-Logic Units

❖ Arithmetic unit

- addition
- subtraction
- multiplication
- division

❖ Logical unit

- AND
- OR
- NOT

Shift Operations

❖ Logical shift

- Logical left shift
- Logical right shift

❖ Arithmetic shift

- Arithmetic left shift
- Arithmetic right shift

Syllabus

- ❖ Objectives
- ❖ Addition and subtraction
- ❖ Multiplication
- ❖ Division
- ❖ Arithmetic and logic unit
 - Basic functions
 - Implementations

Arithmetic-Logic Units

