

HARDWARE DESCRIPTION LANGUAGE FOR DIGITAL DESIGN

數位設計硬體描述語言

Arithmetic Modules

Materials partly adapted from “Digital System Designs and Practices Using Verilog HDL and FPGAs,” M.B. Lin.

OUTLINE

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

ADDER AND SUBTRACTOR

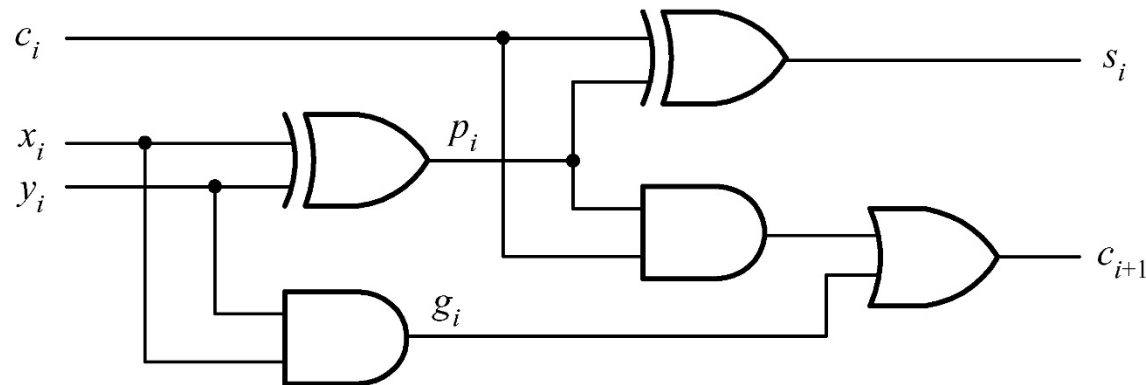
BOTTLENECK OF RIPPLE-CARRY ADDER

- 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): $p_i = x_i \oplus y_i$



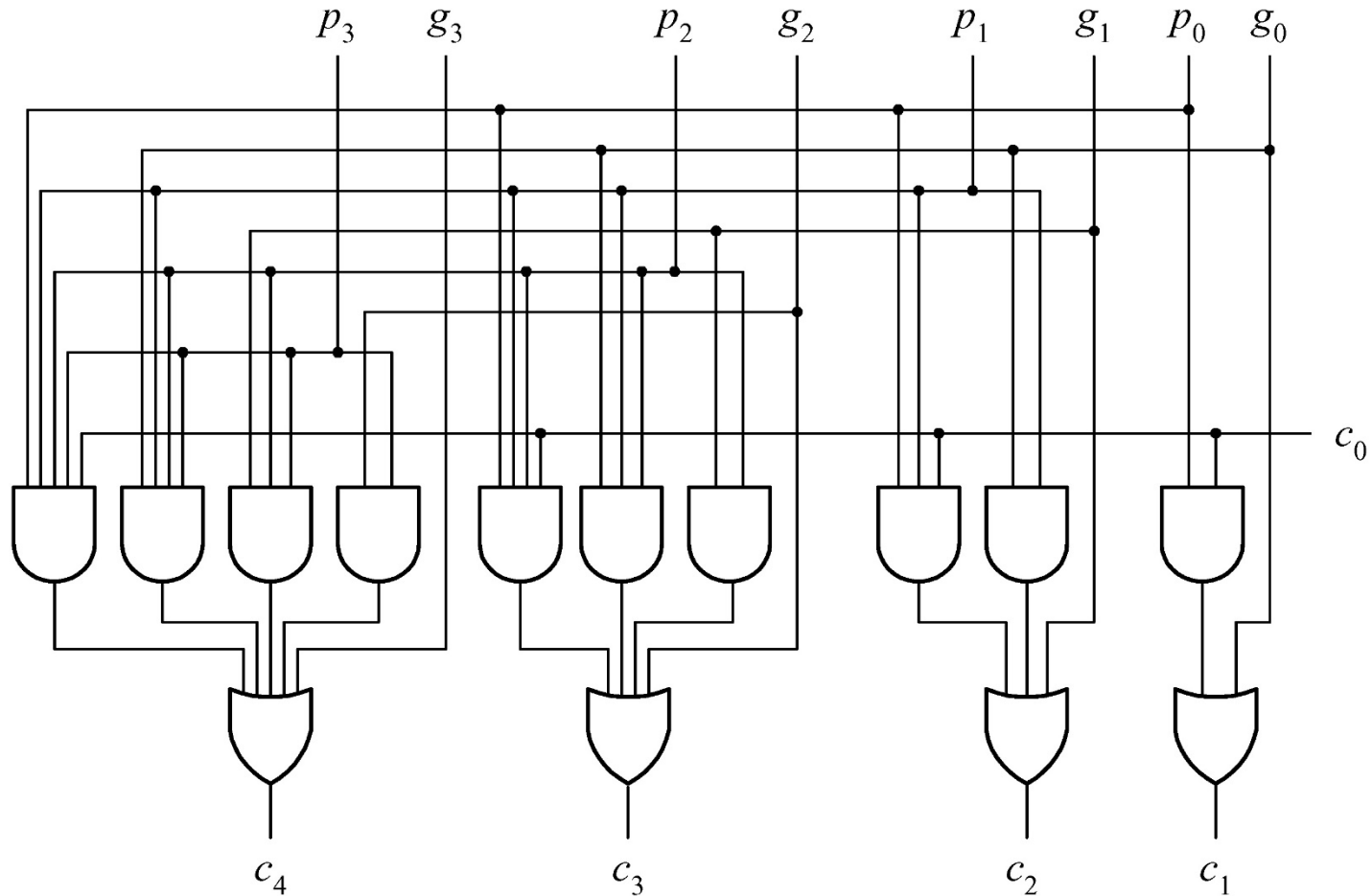
$$s_i = p_i \oplus c_i$$

$$c_1 = g_0 + p_0 c_0$$

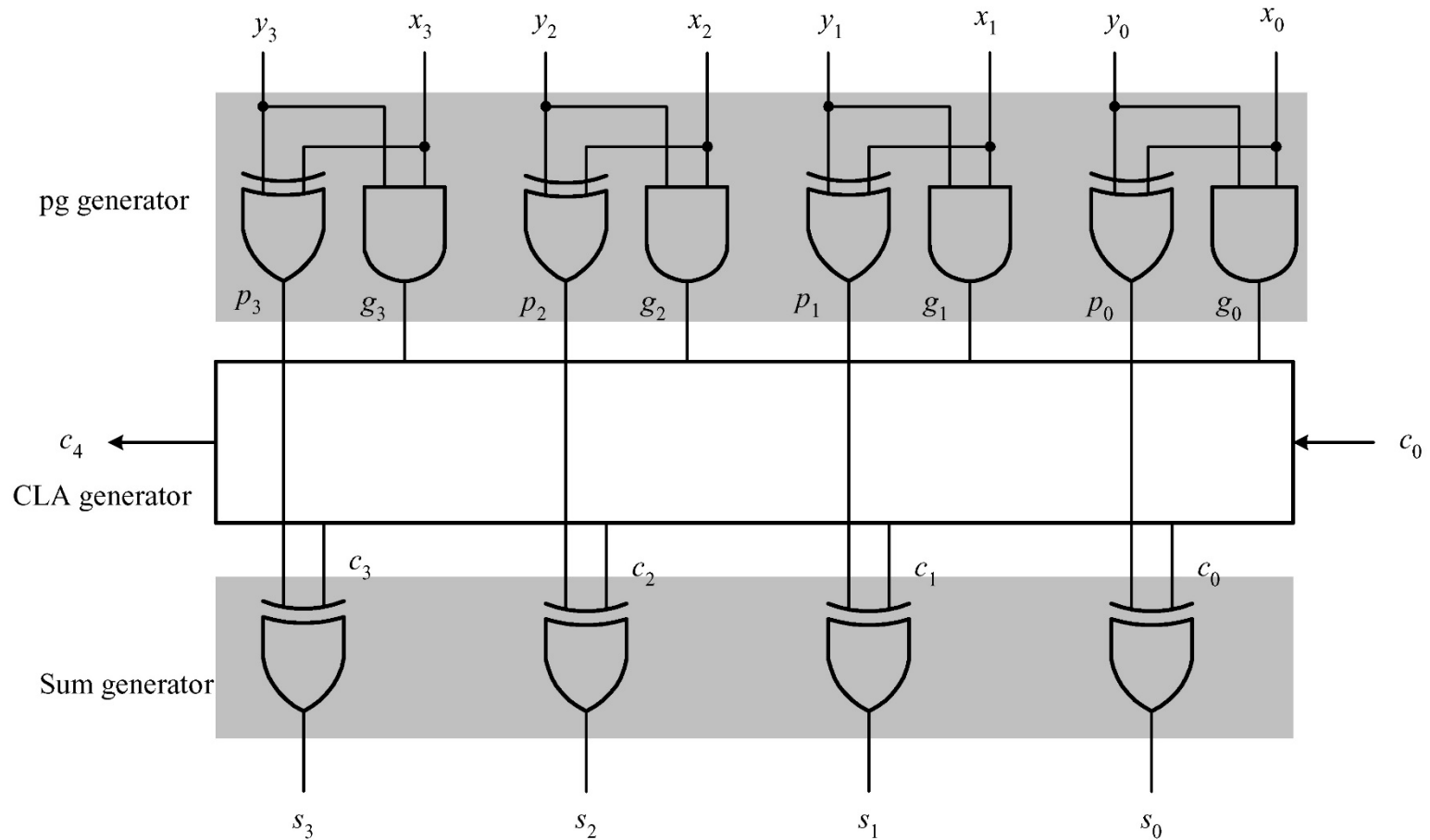
$$c_{i+1} = g_i + p_i \cdot c_i$$

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

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
```

...

PARALLEL-PREFIX ADDERS

- The prefix sums

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

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 can be written as $c_i = g_{i-1} + p_{i-1} \cdot c_{i-1}$

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

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

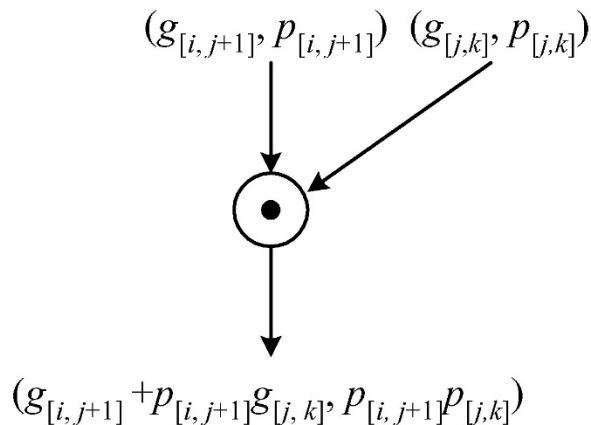
$$w_{[i,j]} = (g_{[i,j]}, p_{[i,j]})$$

PARALLEL-PREFIX ADDERS

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

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

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

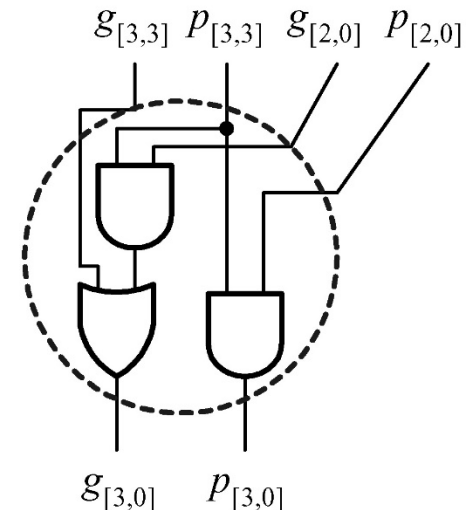


$$g_{[i,i]} = x_{[i]}y_{[i]}$$

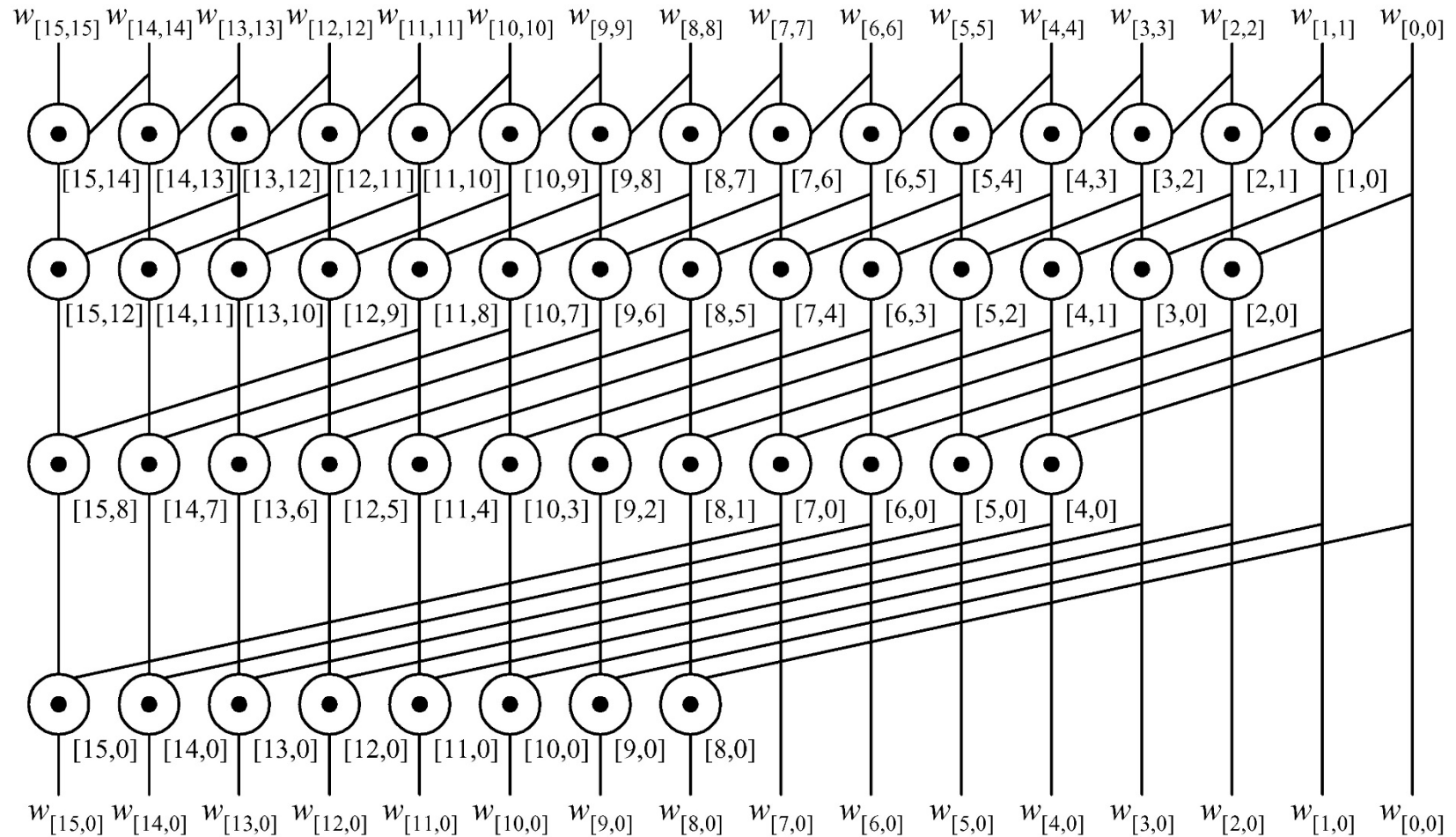
$$p_{[i,i]} = x_{[i]} \oplus y_{[i]}$$

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

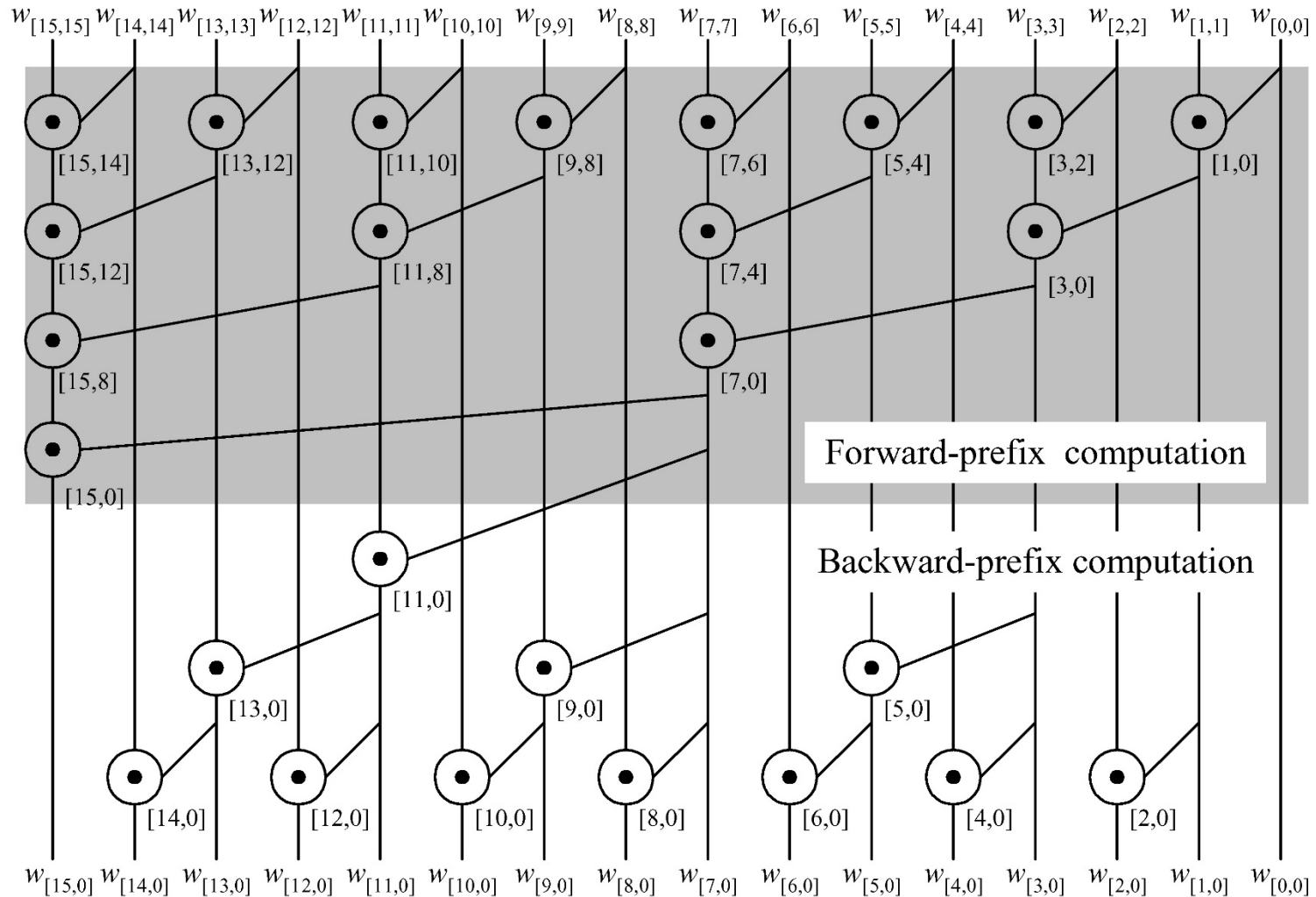
$$w_{[3,0]} = w_{[3,3]} \odot w_{[2,0]}$$



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

MULTIPLICATION

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 $CNT = CNT - 1$;

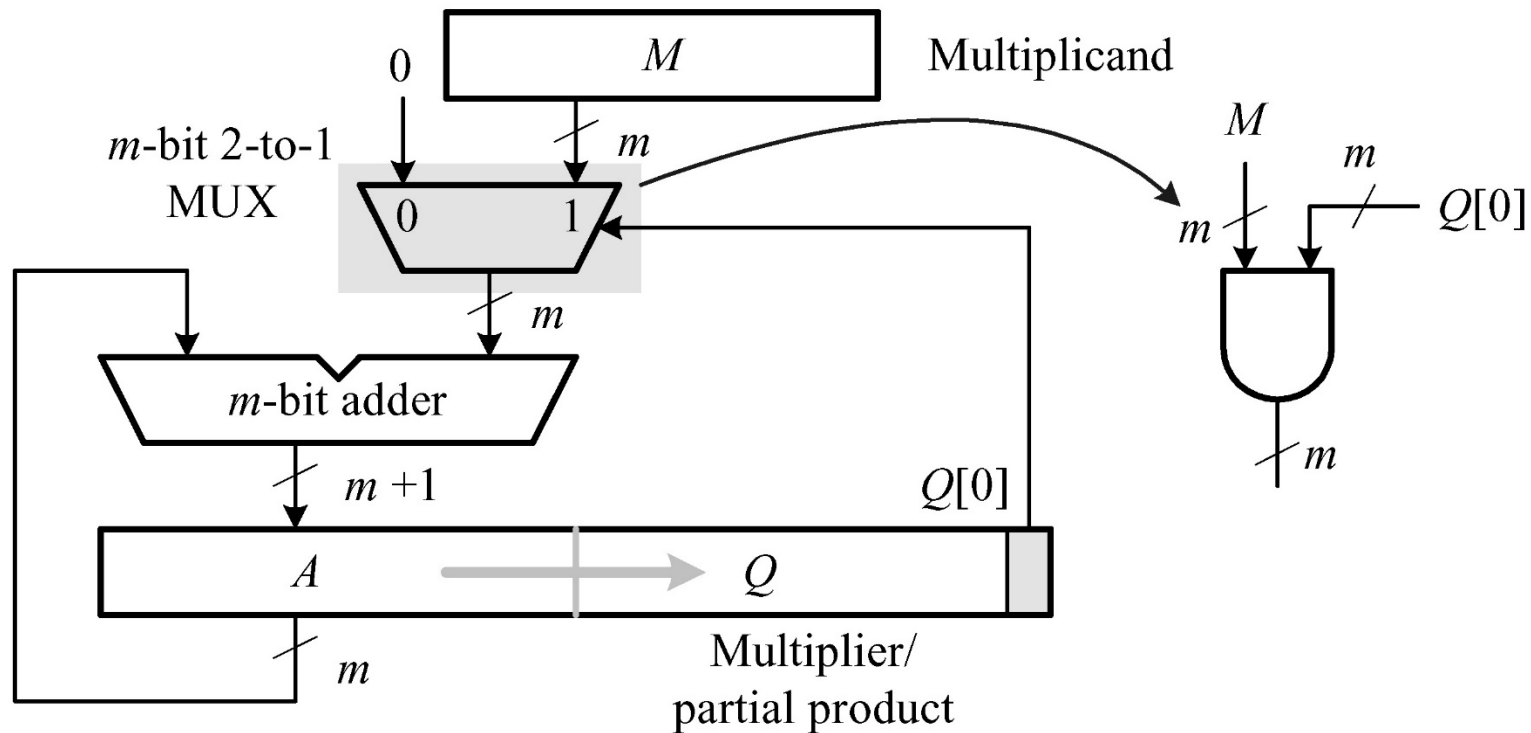
until ($CNT == 0$);

End



SHIFT-AND-ADD MULTIPLICATION

- A sequential implementation



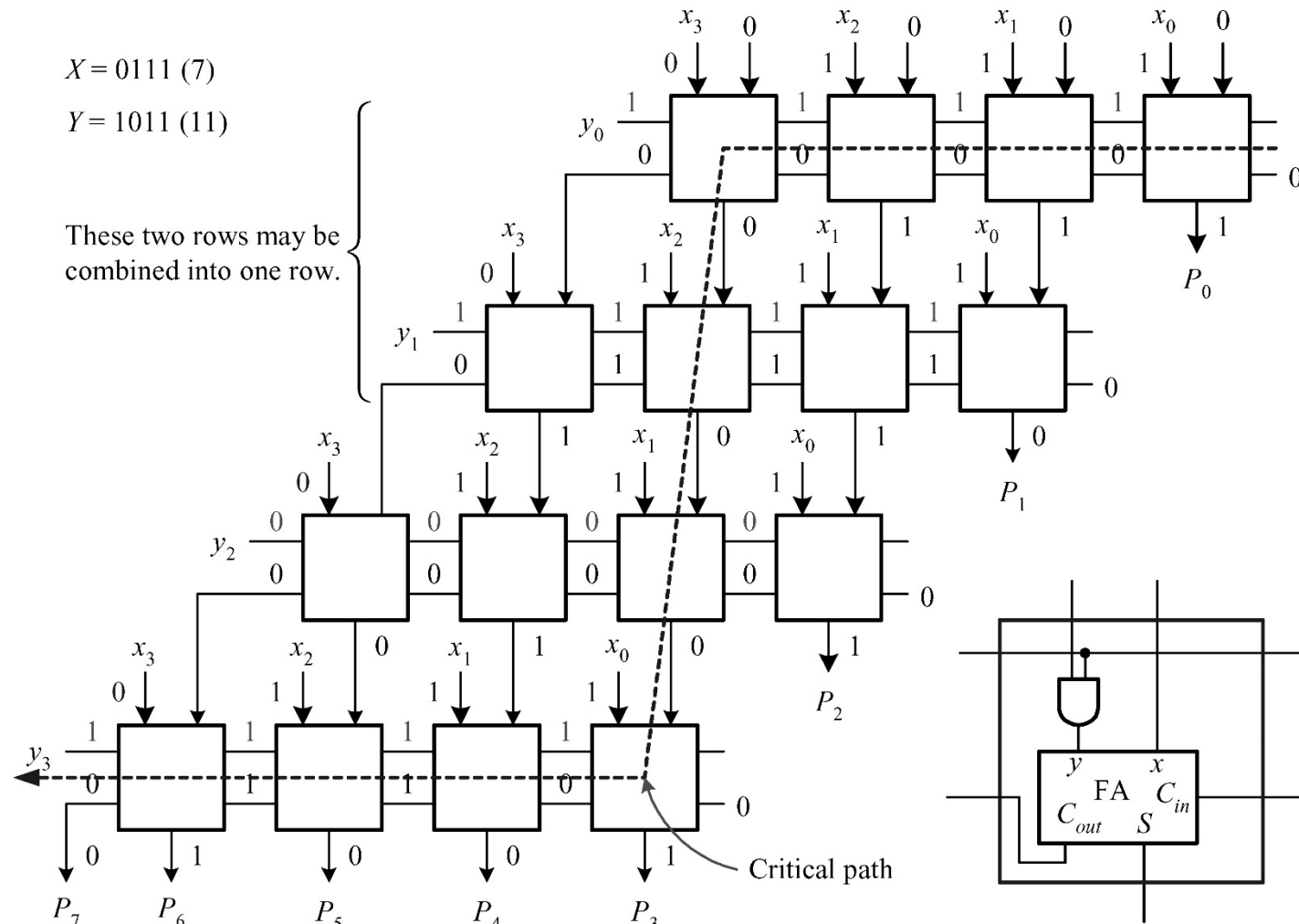
A BASIC ARRAY MULTIPLIER

- An iterative logic structure

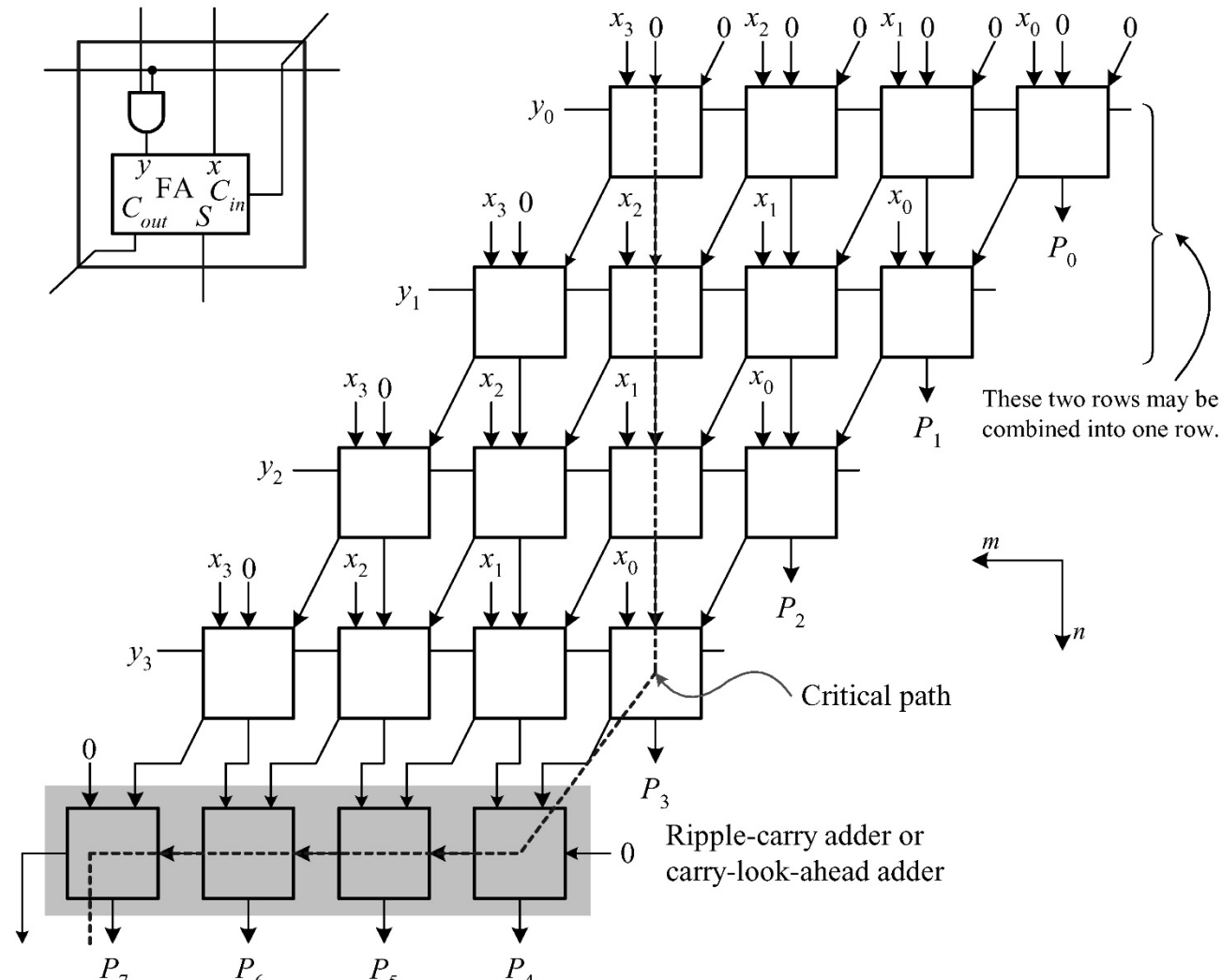
$$\begin{array}{rcccccc}
 & & x_3 & x_2 & x_1 & x_0 & = X \text{ (multiplicand)} \\
 \times & y_3 & y_2 & y_1 & y_0 & = Y \text{ (multiplier)} \\
 \hline
 & & x_3y_0 & x_2y_0 & x_1y_0 & x_0y_0 & \\
 & & & x_3y_1 & x_2y_1 & x_1y_1 & x_0y_1 \\
 & & & & x_3y_2 & x_2y_2 & x_1y_2 & x_0y_2 \\
 + & x_3y_3 &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

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

A BASIC UNSIGNED ARRAY MULTIPLIER



AN UNSIGNED CSA ARRAY MULTIPLIER



The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

A SIGNED ARRAY MULTIPLIER

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

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

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

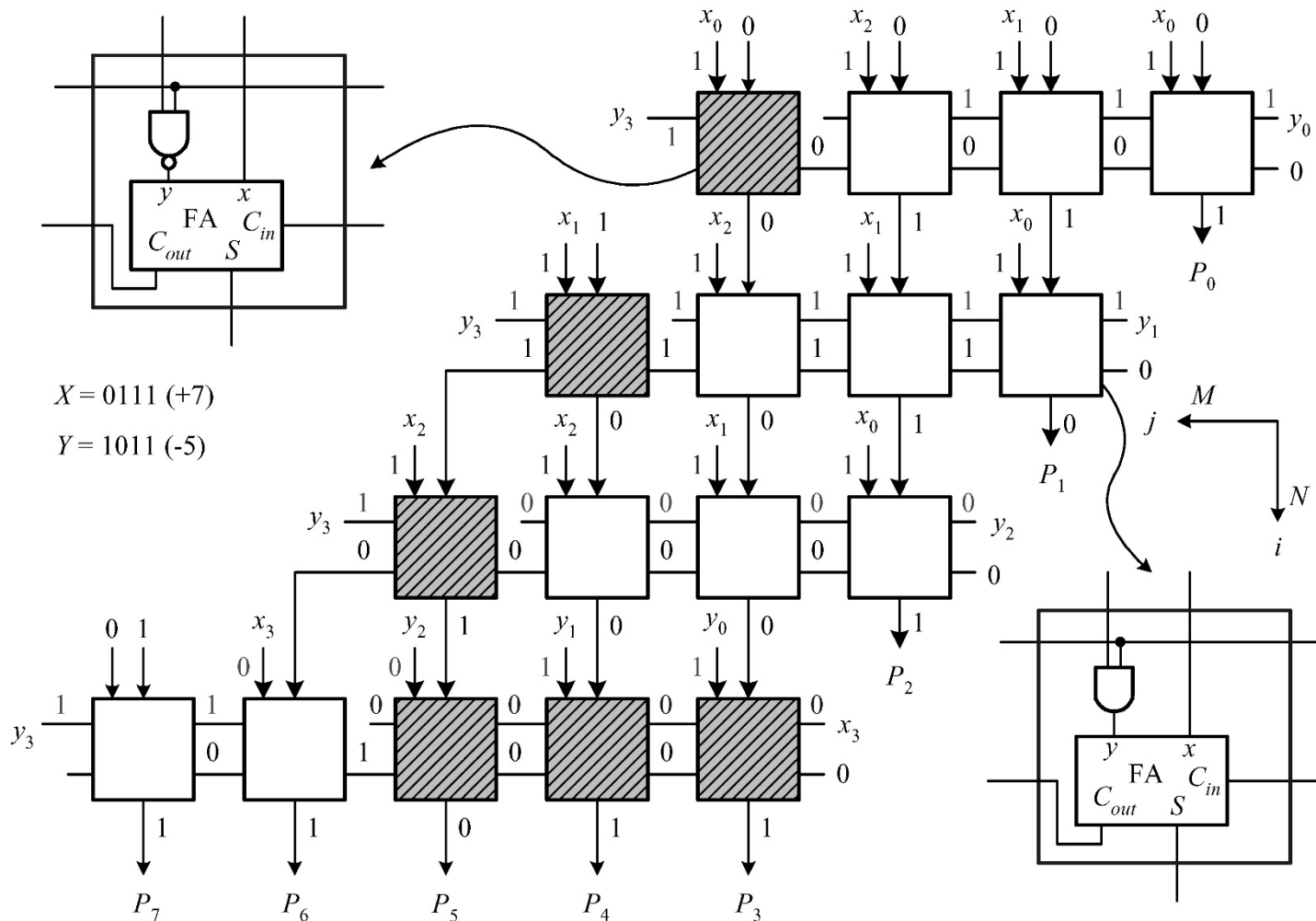
$$P = XY$$

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

A SIGNED ARRAY MULTIPLIER

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

A SIGNED ARRAY MULTIPLIER



DIVISION

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
$$D[0] = 0; \text{ left shift } R : D \text{ one bit; } R = R + M; \text{ end}$$
else begin
$$D[0] = 1; \text{ left shift } R : D \text{ one bit; } R = R - M; \text{ end}$$
 - 4.2 $CNT = CNT - 1$;
 - until $(CNT == 0)$
 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 ② 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

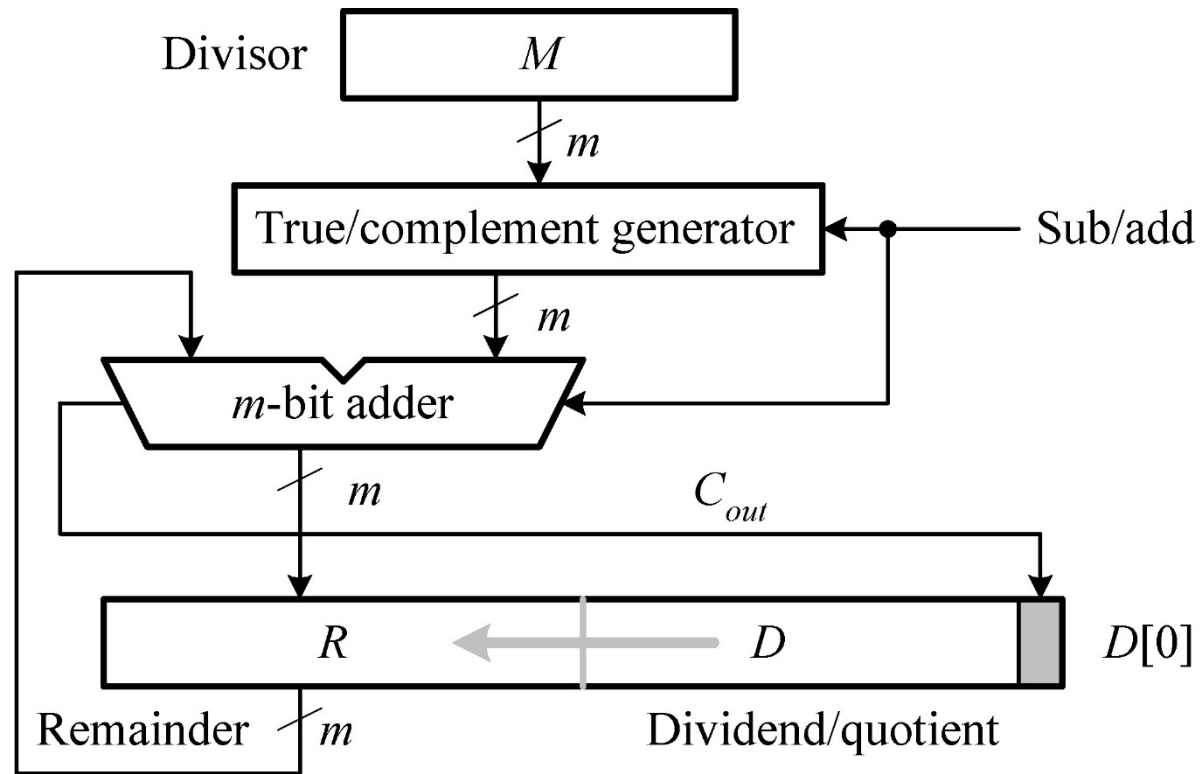
0 1 1 0

Remainder → 0 0 0 1

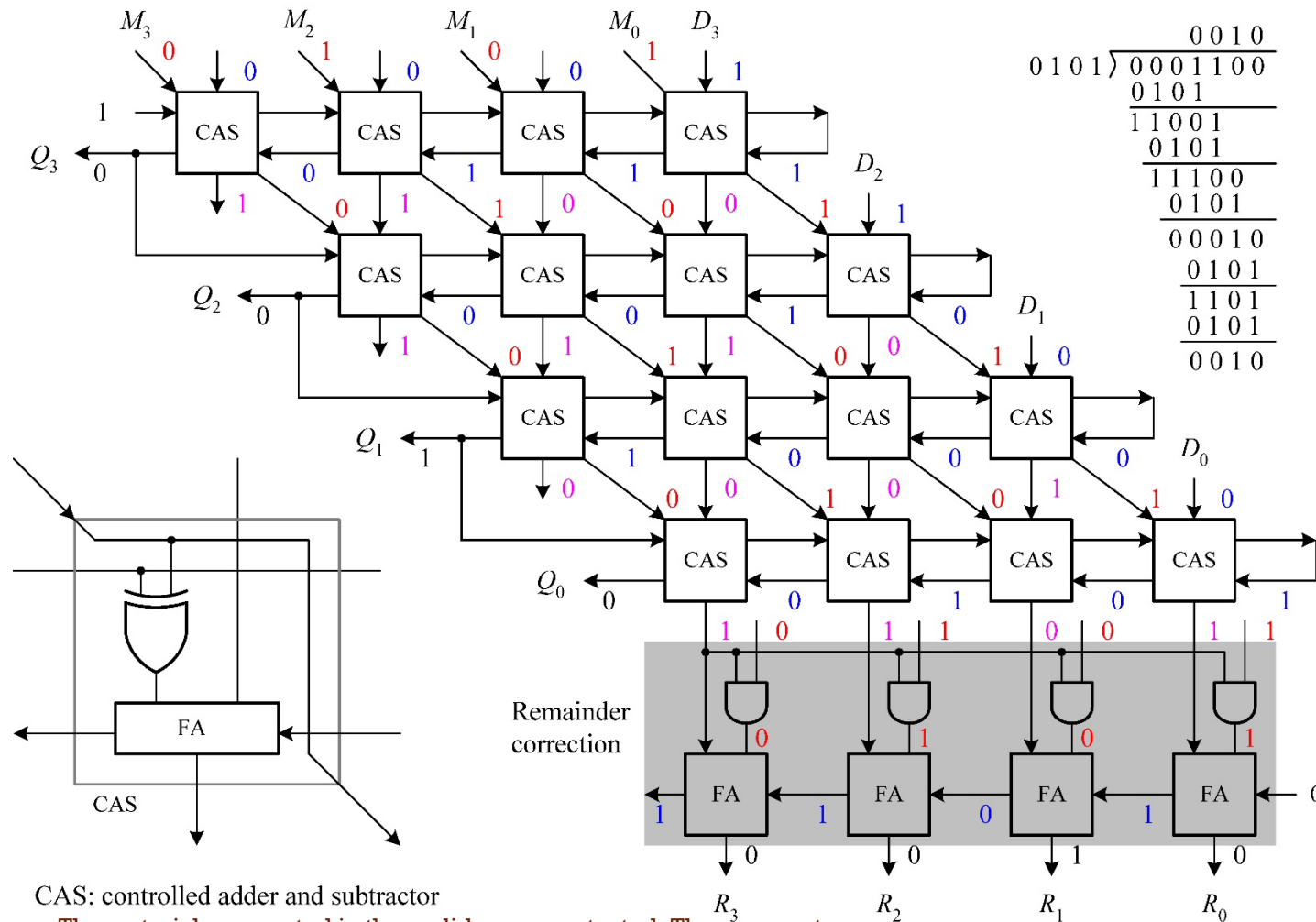
$$\begin{array}{l}
D - M \\
< 0, \quad Q = 0 \\
\text{right shift } M, \quad D + M \\
< 0, \quad Q = 0 \\
\text{right shift } M, \quad D + M \\
< 0, \quad Q = 0 \\
\text{right shift } M, \quad D + M \\
< 0, \quad Q = 0 \\
\text{right shift } M, \quad D + M \\
> 0, \quad Q = 1 \\
\text{right shift } M, \quad D - M \\
> 0, \quad Q = 1 \\
\text{right shift } M, \quad D - M \\
> 0, \quad Q = 1 \\
\text{right shift } M, \quad D - M \\
< 0, \quad Q = 0 \\
D + M
\end{array}$$

A SEQUENTIAL UNSIGNED NON-RESTORING DIVISION

- A sequential implementation



AN UNSIGNED ARRAY NON-RESTORING DIVIDER



CAS: controlled adder and subtractor

The materials presented in these slides are protected. They may not be redistributed, reproduced, or used in any form without the express consent of the instructor, Prof. Chiou, at NCKU EE.

ARITHMETIC AND LOGIC UNIT

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

ARITHMETIC-LOGIC UNITS

