



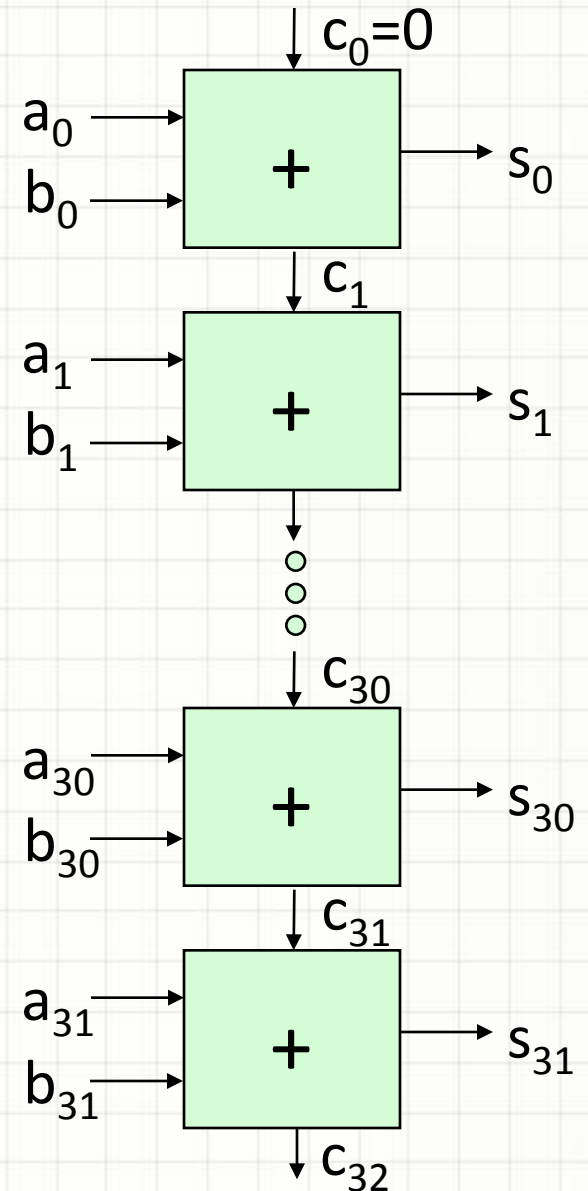
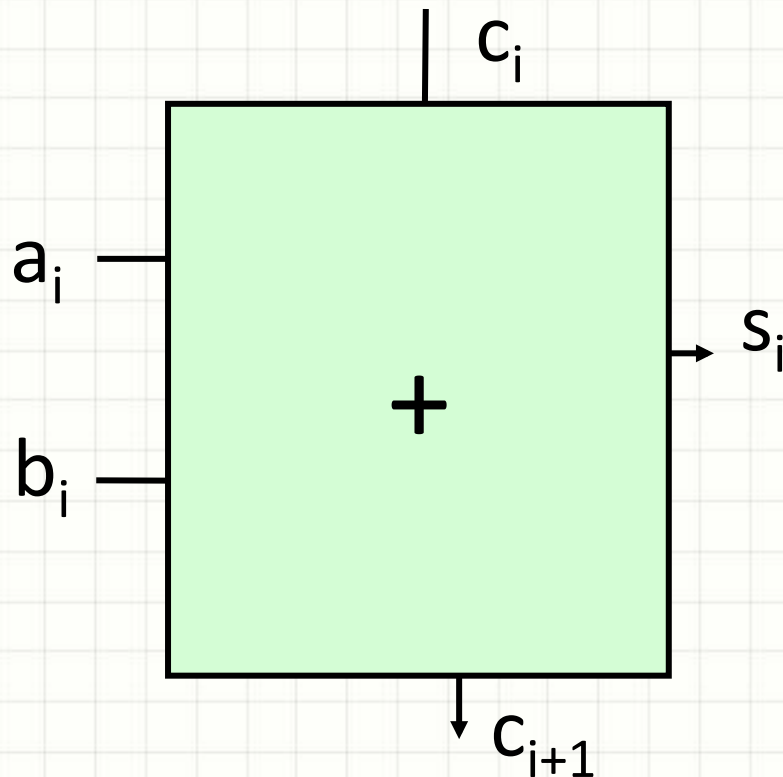
COL215 DIGITAL LOGIC AND SYSTEM DESIGN

Fast Adder, Multiplier Design

08 September 2017

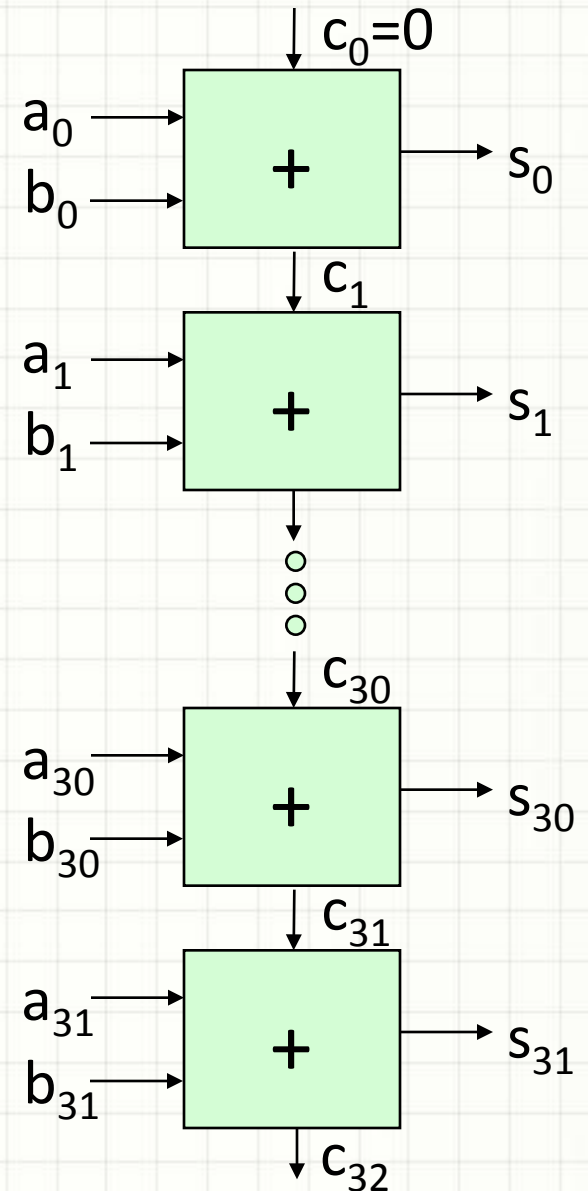
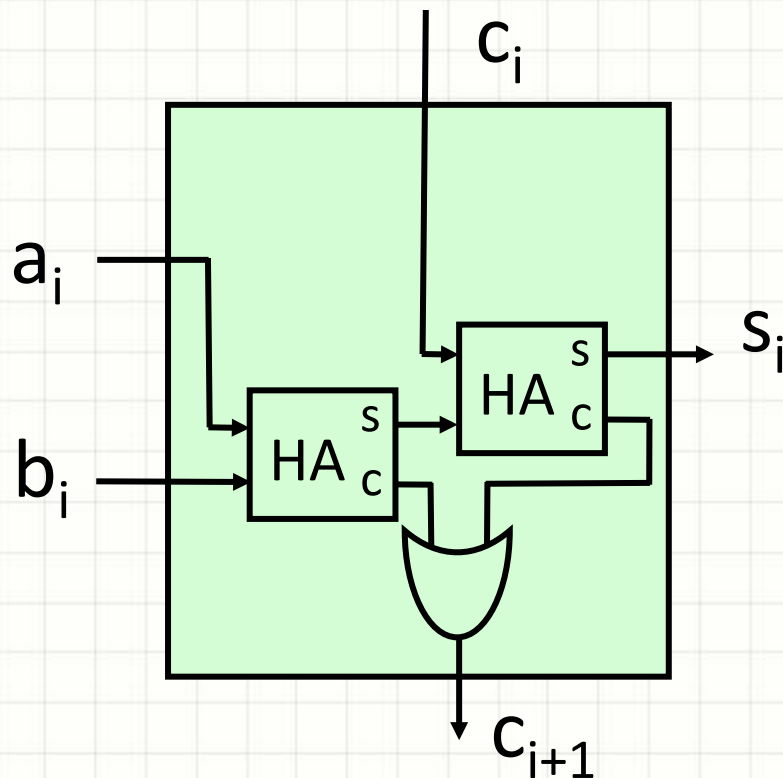
Adder circuit

n bit parallel adder =
array of n full adders



Adder circuit

n bit parallel adder =
array of n full adders



Boolean expressions for HA & FA

HA:

$$s = a' b + a b'$$

$$= a \oplus b$$

$$c = a b$$

FA:

$$s_i = a_i' b_i' c_i + a_i' b_i c_i' + a_i b_i' c_i' + a_i b_i c_i$$

$$= a_i \oplus b_i \oplus c_i$$

$$c_{i+1} = a_i b_i + a_i c_i + b_i c_i = a_i b_i + (a_i + b_i) c_i$$

Carry-lookahead adder

Express carries using p's and g's

$$p_i = a_i + b_i$$

$$g_i = a_i \cdot b_i$$

$$c_1 = p_0 c_0 + g_0$$

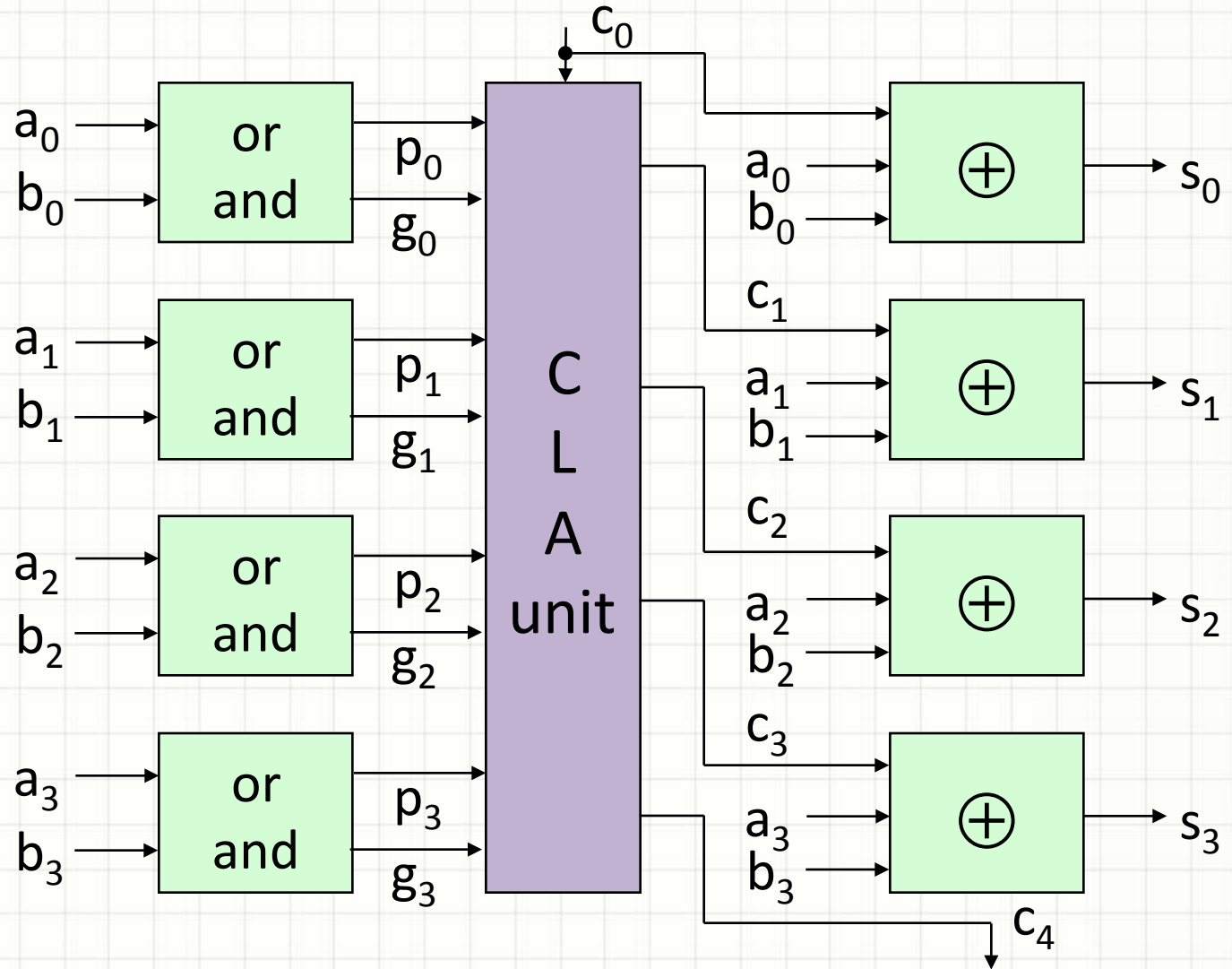
$$c_2 = p_1 c_1 + g_1 = p_1 p_0 c_0 + p_1 g_0 + g_1$$

$$c_3 = p_2 c_2 + g_2 = p_2 p_1 p_0 c_0 + p_2 p_1 g_0 + p_2 g_1 + g_2$$

$$c_4 = p_3 c_3 + g_3 =$$

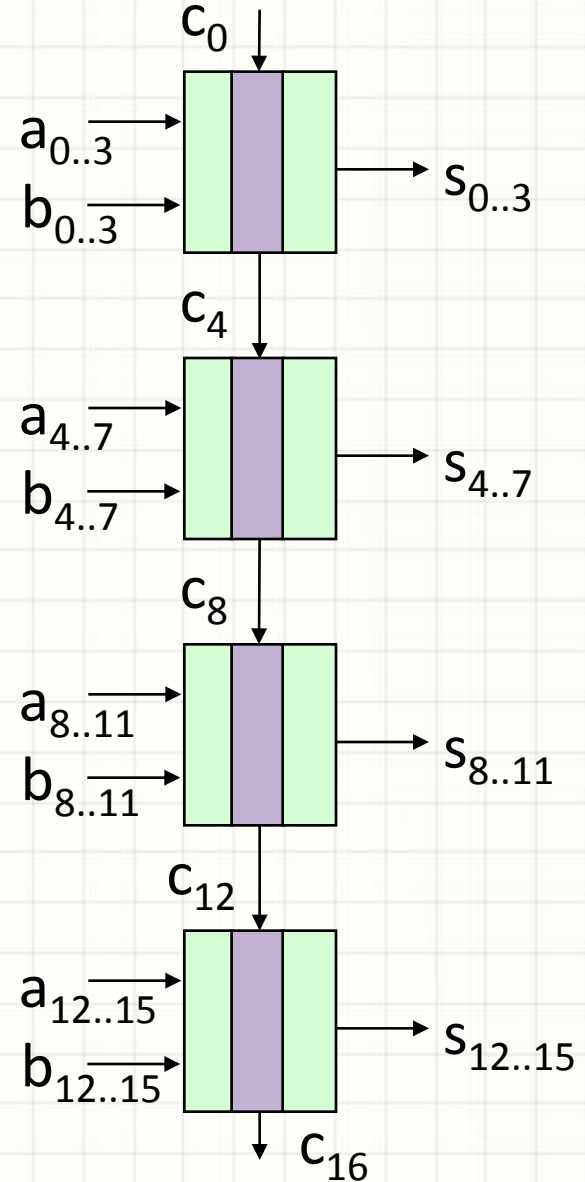
$$p_3 p_2 p_1 p_0 c_0 + p_3 p_2 p_1 g_0 + p_3 p_2 g_1 + p_3 g_2 + g_3$$

4 bit CLA adder



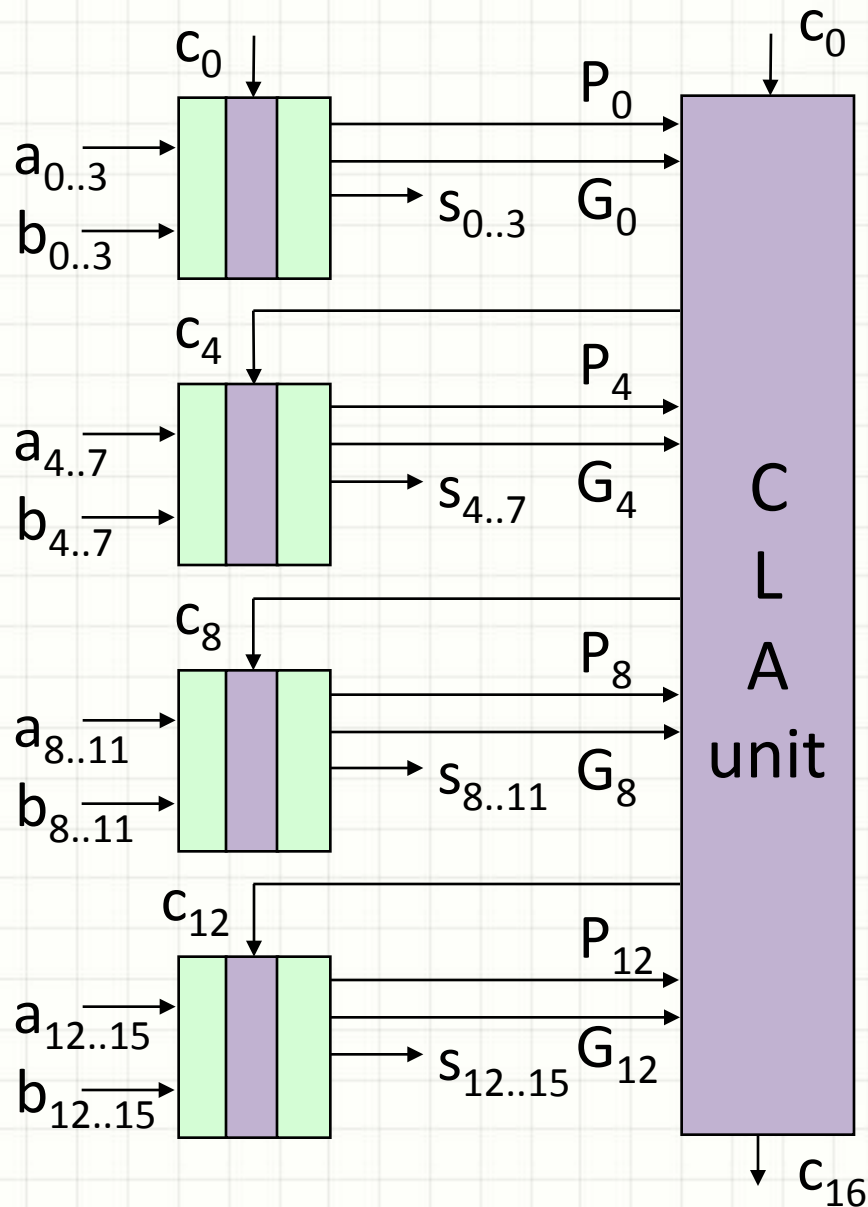
16 bit addition with 4 bit CLAs

partial rippling of carry



2 levels of look ahead

no rippling
of carry



Group propagate & generate

$$P_i = p_{i+3} p_{i+2} p_{i+1} p_i$$

$$G_i = p_{i+3} p_{i+2} p_{i+1} g_i + p_{i+3} p_{i+2} g_{i+1} + p_{i+3} g_{i+2} + g_{i+3}$$

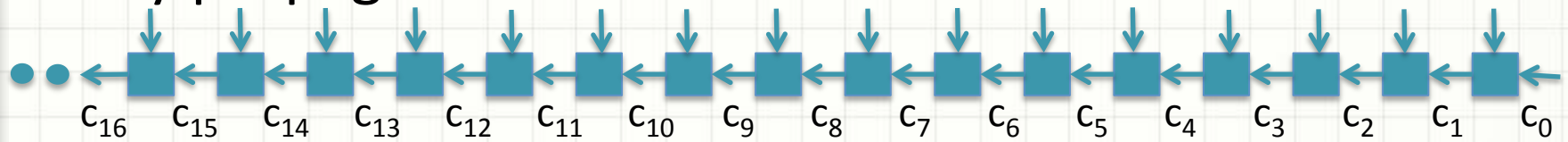
$$c_4 = P_0 c_0 + G_0$$

$$c_8 = P_4 P_0 c_0 + P_4 G_0 + G_4$$

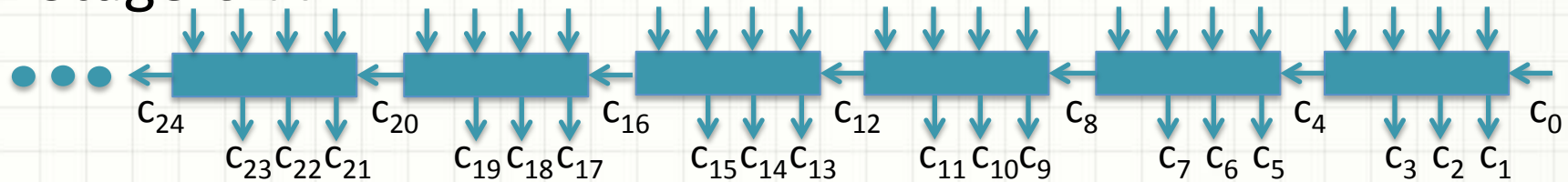
$$c_{12} = P_8 P_4 P_0 c_0 + P_8 P_4 G_0 + P_8 G_4 + G_8$$

$$c_{16} = P_{12} P_8 P_4 P_0 c_0 + P_{12} P_8 P_4 G_0 + P_{12} P_8 G_4 + P_{12} G_8 + G_{12}$$

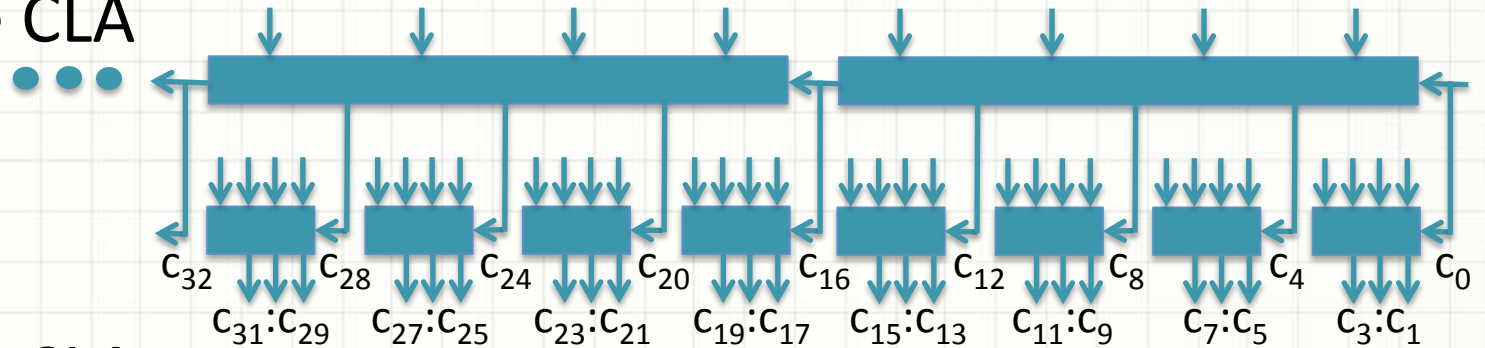
Carry propagation



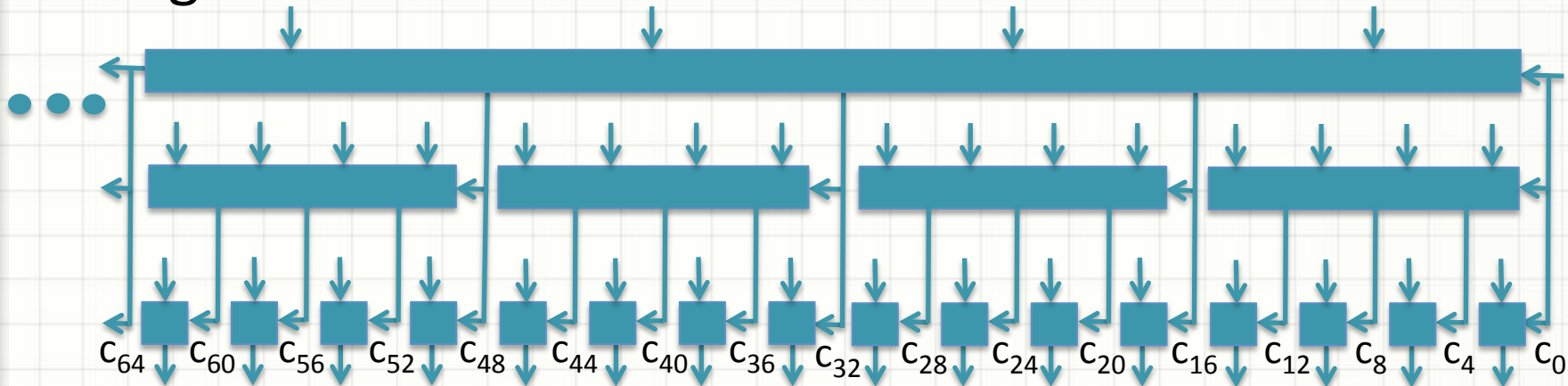
1 Stage CLA



2 Stage CLA



3 Stage CLA



Multiplication: paper - pencil method

The diagram illustrates the paper-pencil method for binary multiplication. It shows the multiplication of two 4-bit numbers, A and B, resulting in an 8-bit product AB. The partial products are shown with their respective shifts, and the final sum is calculated using XOR operations.

A					1	0	1	1									
B					<u>1</u>	<u>1</u>	<u>0</u>	<u>1</u>									
					1	0	1	1									
				0	0	0	0	x									
		1	0	1	1	x	x										
	1	0	1	1	x	x	x										
AB	1	0	0	0	1	1	1	1									

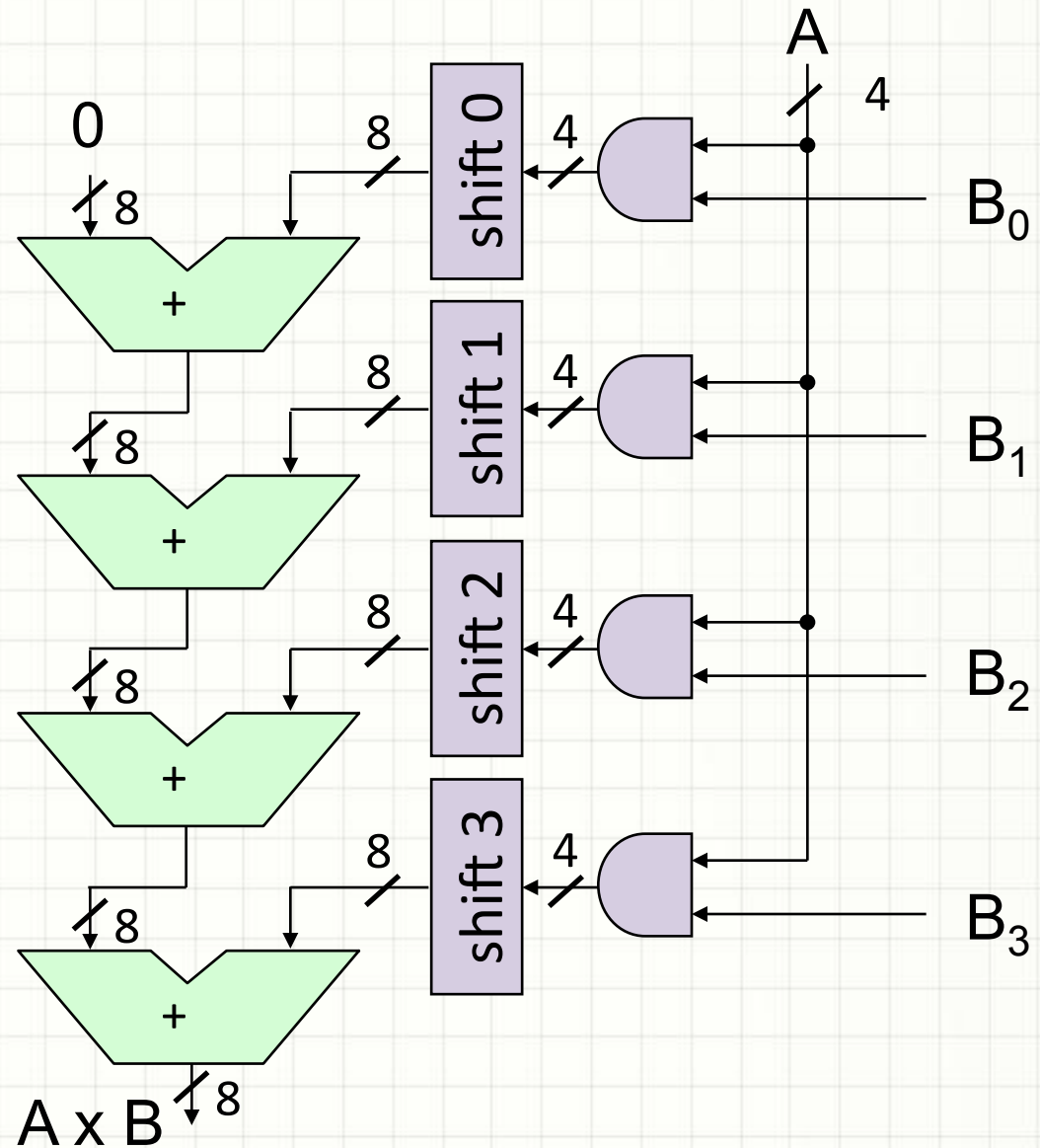
The partial products are shifted and summed using XOR operations:

- Partial product 1 (1011) is shifted 0 positions.
- Partial product 2 (0000x) is shifted 1 position.
- Partial product 3 (1011xx) is shifted 2 positions.
- Partial product 4 (1011xxx) is shifted 3 positions.

The final result is the sum of these partial products, which is 10001111.

Shift add multiplier

$$A \times B = \sum_{i=0}^{n-1} A \cdot B_i \times 2^i$$



Shift operations

shift left logical 3 bits

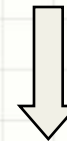
$a_{31} a_{30} \dots a_1 a_0$



$a_{28} a_{27} \dots a_1 a_0 0 0 0$

shift right logical 3 bits

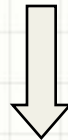
$a_{31} a_{30} \dots a_1 a_0$



$0 0 0 a_{31} a_{30} \dots a_4 a_3$

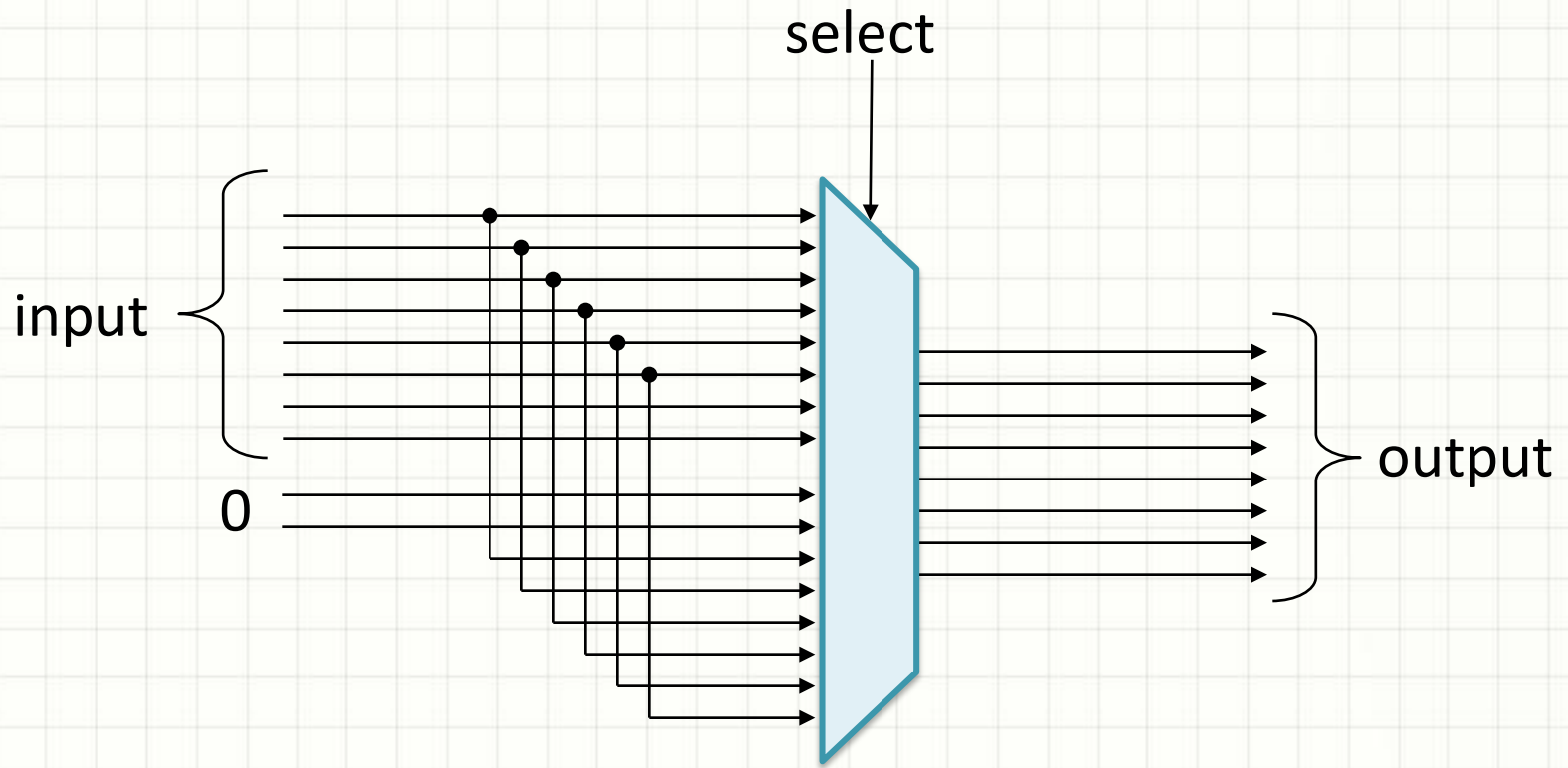
shift right arithmetic 2 bits

$a_{31} a_{30} \dots a_1 a_0$

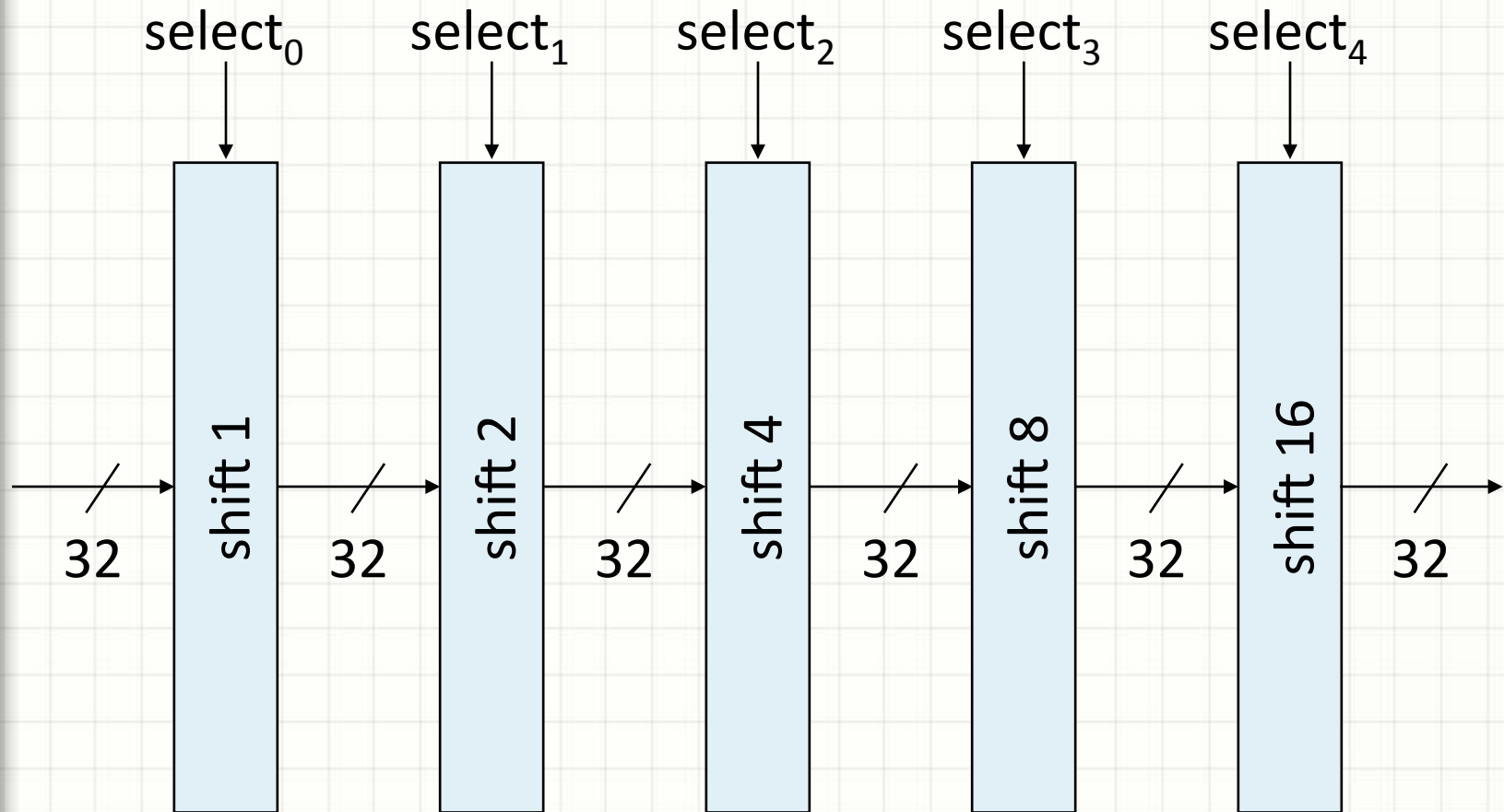


$a_{31} a_{31} a_{31} a_{30} \dots a_3 a_2$

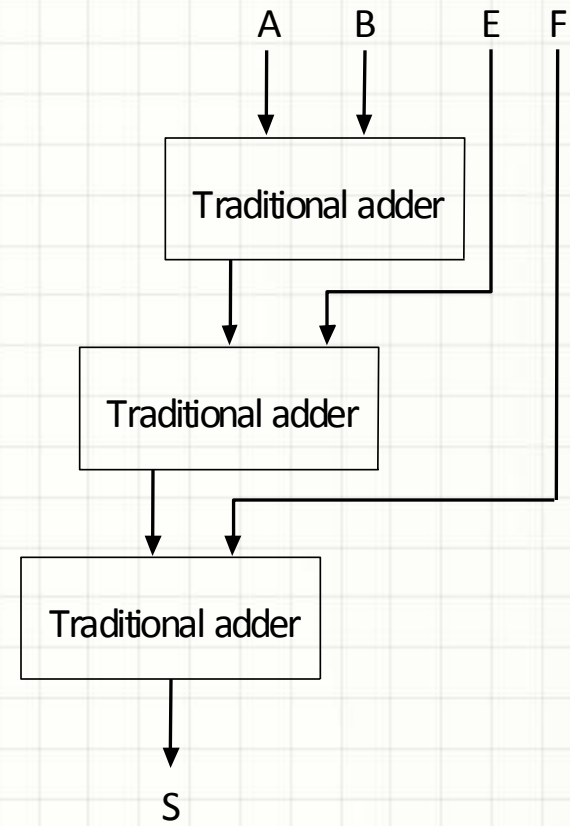
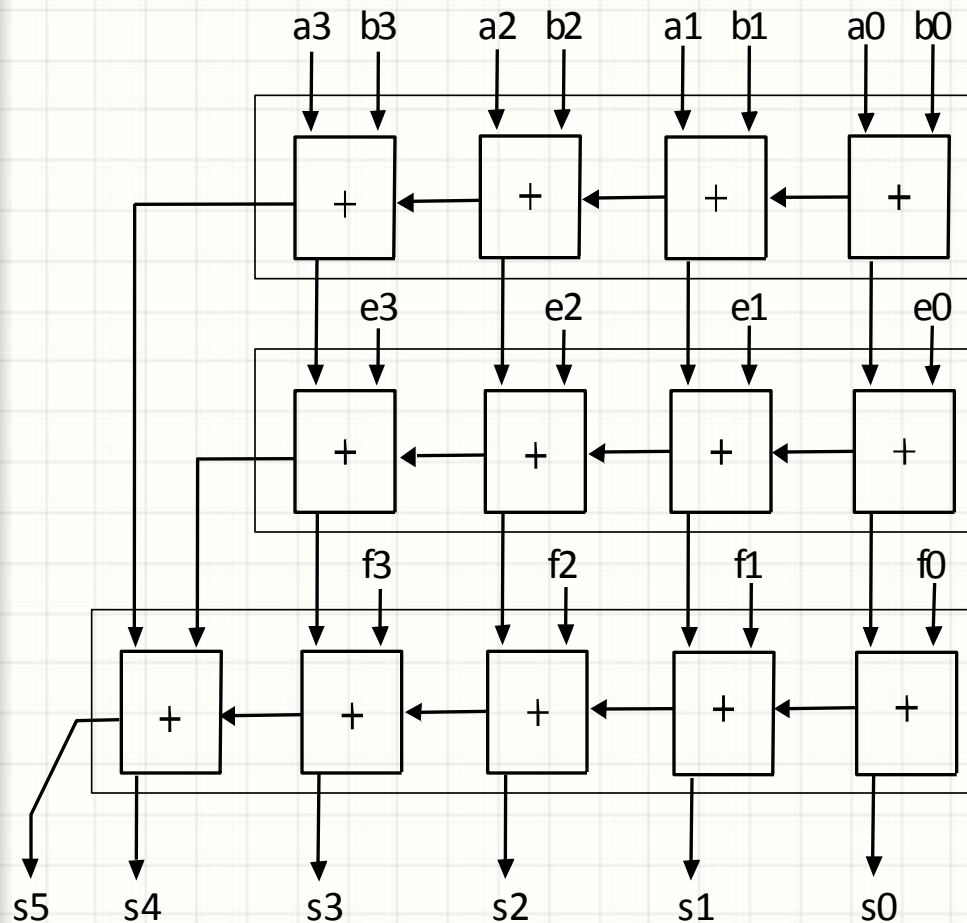
Circuit for shifting by 2 bits



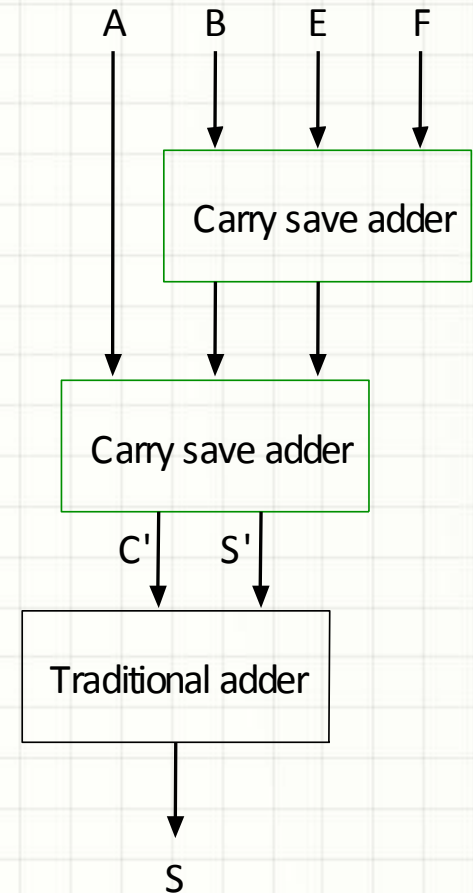
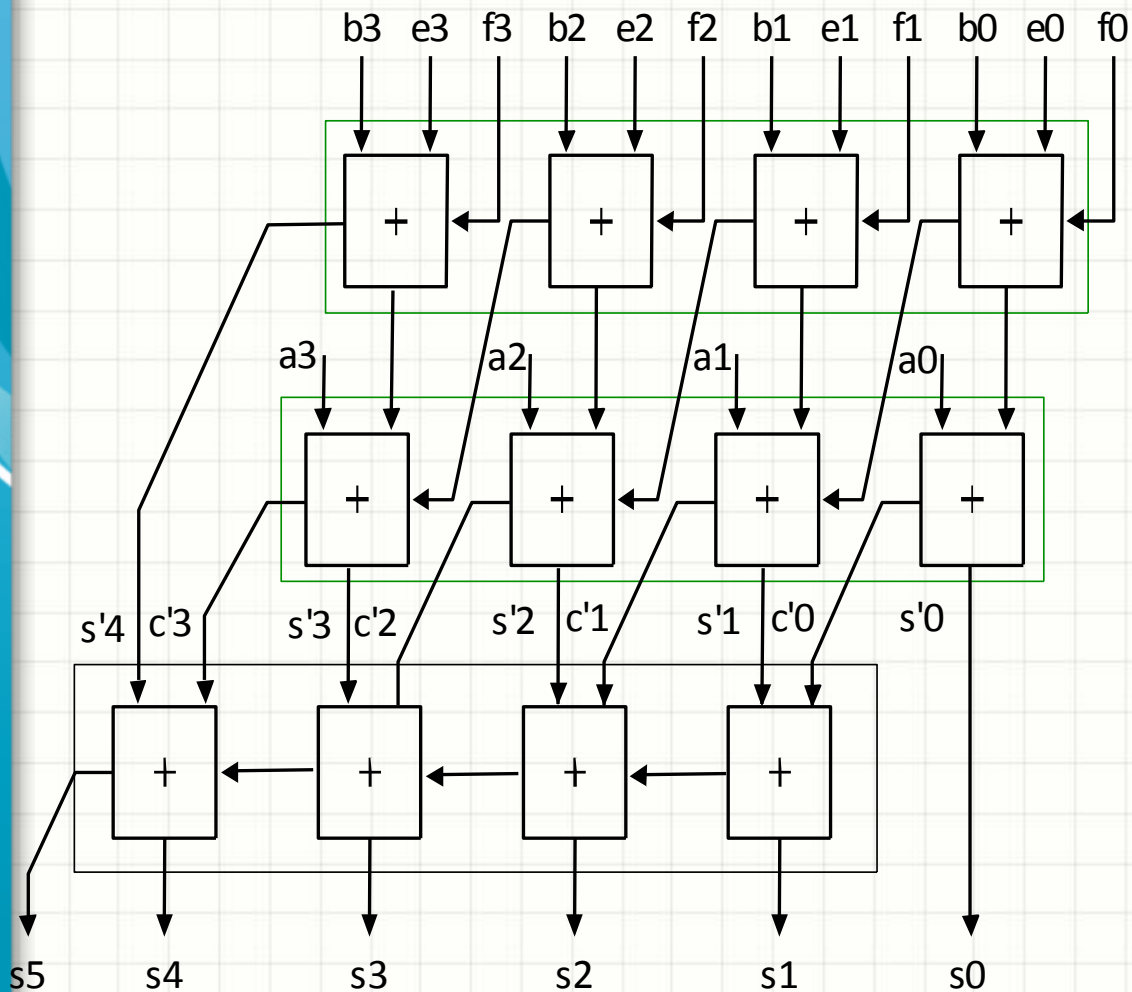
Circuit for shifting by 0 to 31 bits



Multiple operand addition

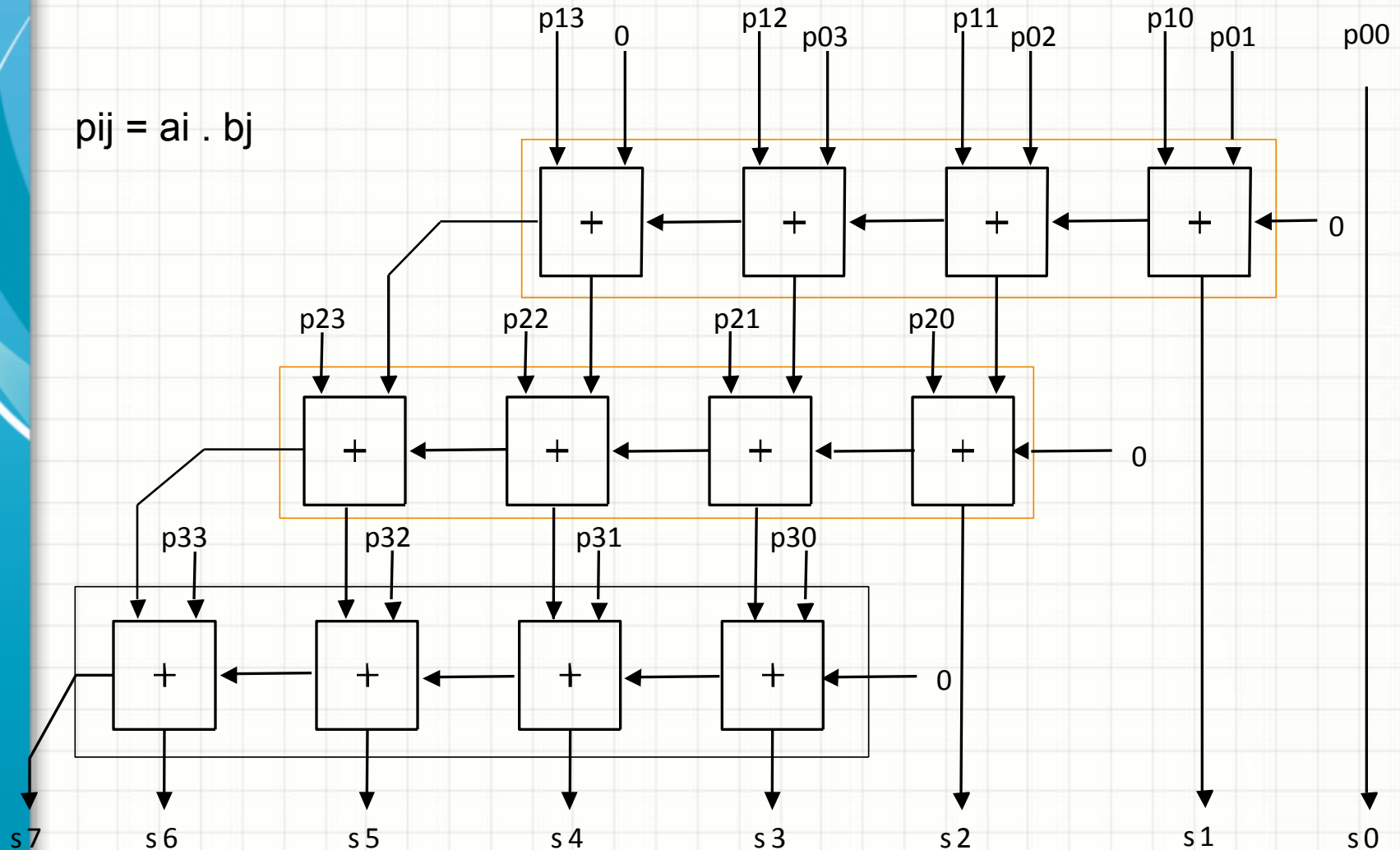


Carry save addition

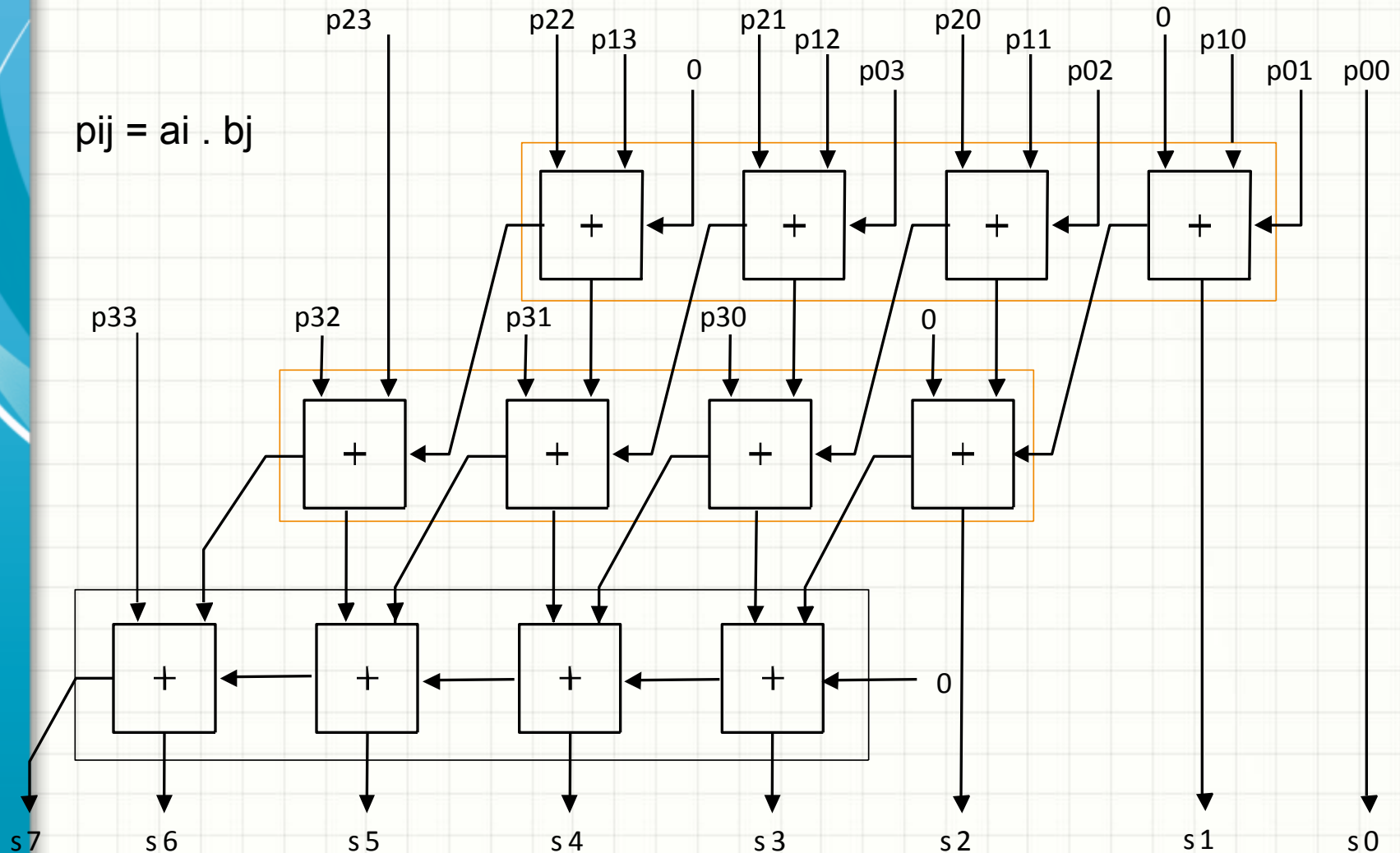


Array mult with carry propagate

$$p_{ij} = a_i \cdot b_j$$



Array multiplier with carry save



Delay of array multipliers

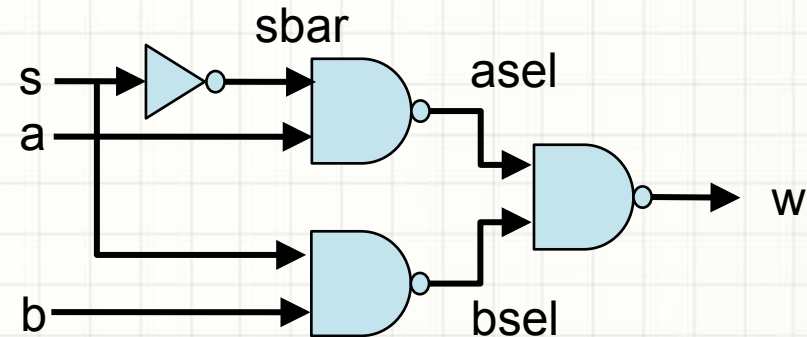
- Using carry propagate adders: $\sim 3 n d$
- Using carry save adders: $\sim 2 n d$
(here d = delay of 1 bit adder)

Structural design in VHDL

ENTITY multiplexer IS

PORT (a, b, s : IN BIT; w : OUT BIT);

END ENTITY;



ARCHITECTURE direct OF multiplexer IS

SIGNAL sbar, asel, bsel : BIT;

BEGIN

U1: ENTITY WORK.inv (simple) PORT MAP (s, sbar);

U2: ENTITY WORK.nand2 (simple) PORT MAP (a, sbar, asel);

U3: ENTITY WORK.nand2 (simple) PORT MAP (b, s, bsel);

U4: ENTITY WORK.nand2 (simple) PORT MAP (asel, bsel, w);

END ARCHITECTURE direct;

Structural design in VHDL

ARCHITECTURE gates OF multiplexer IS

```
COMPONENT n1 PORT ( i1: IN BIT; y: OUT BIT); END COMPONENT;
```

```
COMPONENT n2 PORT ( i1, i2: IN BIT; y: OUT BIT); END COMPONENT;
```

```
FOR ALL : n1 USE ENTITY WORK.inv;
```

```
FOR ALL : n2 USE ENTITY WORK.nand2;
```

```
SIGNAL sbar, asel, bsel : BIT;
```

BEGIN

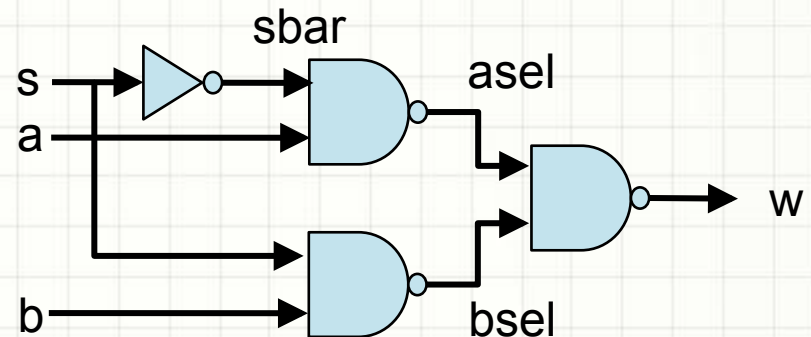
```
U1: n1 PORT MAP (s, sbar);
```

```
U2: n2 PORT MAP (a, sbar, asel);
```

```
U3: n2 PORT MAP (b, s, bsel);
```

```
U4: n2 PORT MAP (asel, bsel, w);
```

```
END ARCHITECTURE gates;
```



Building blocks

ENTITY inv IS

PORT (i1 : IN BIT; y : OUT BIT);

END ENTITY;

ARCHITECTURE simple OF inv IS

...

END ARCHITECTURE simple;

ENTITY nand2 IS

PORT (i1, i2 : IN BIT; y : OUT BIT);

END ENTITY;

ARCHITECTURE simple OF nand2 IS

...

END ARCHITECTURE simple;

Generate statement

- Creating an array of component instances
FOR i IN 1 TO n GENERATE ...
- Instantiating optional components
IF condition GENERATE ...
- These constructs can be nested

8 bit Multiplexer

ENTITY multiplexer8 IS

PORT (a, b : IN BIT_VECTOR (7 DOWNT0 0); s : IN BIT;
w : OUT BIT_VECTOR (7 DOWNT0 0));

END ENTITY;

ARCHITECTURE direct OF multiplexer8 IS

BEGIN

U0TO7: FOR i IN 0 TO 7 GENERATE

Ui: ENTITY WORK.multiplexer (gates)

PORT MAP (a(i), b(i), s, w(i));

END GENERATE;

END ARCHITECTURE direct;

8 bit Mux with comp decl

ARCHITECTURE iterative OF multiplexer8 IS

 COMPONENT mux PORT (a, b, s : IN BIT; w : OUT BIT);

 END COMPONENT;

 FOR ALL : mux USE ENTITY WORK.multiplexer (gates);

BEGIN

 U0TO7: FOR i IN 0 TO 7 GENERATE

 Ui: mux PORT MAP (a(i), b(i), s, w(i));

 END GENERATE;

END ARCHITECTURE iterative;



THANKS