Rutgers University
School of Engineering

Fall 2022

332:231 – Digital Logic Design

Sophocles J. Orfanidis
ECE Department
orfanidi@rutgers.edu

Unit 5 – arithmetic systems, comparators, adders, multipliers

Course Topics

1. Introduction to DLD, Verilog HDL, MATLAB/Simulink
2. Number systems
3. Analysis and synthesis of combinational circuits
4. Decoders/encoders, multiplexers/demultiplexers
5. Arithmetic systems, comparators, adders, multipliers
6. Sequential circuits, latches, flip-flops
7. Registers, shift registers, counters, LFSRs
8. Finite state machines, analysis and synthesis

Text: J. F. Wakerly, *Digital Design Principles and Practices*, 5/e, Pearson, 2018
additional references on Canvas Files > References

Components for arithmetic operations (Wakerly, Ch. 7 & 8).

Topics discussed are:

(a) Comparators for unsigned integers and for and two's complement signed integers

(b) half-adders

(c) full-adders

(d) ripple adders/subtractors for two's complement integers

(e) carry-lookahead adders

(f) overflow in addition/subtraction

(g) multiplier combinational circuits

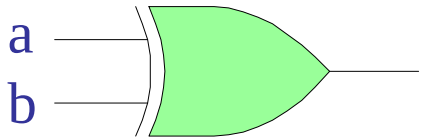
Unit-5 Contents:

1. Comparators for unsigned and signed 2's complement integers
2. Addition and subtraction
3. Half-adders
4. Full-adders
5. Full-adders – ripple adders
6. Full-adders – carry-look-ahead adders
7. Full-adders/subtractors for 2's complement integers
8. Overflow in 2's complement addition/subtraction
9. Multiplier combinational circuits

Comparators

A comparator compares two binary words and determines if they are equal, or, if one is greater or less than the other. Depending on the implementation, the binary words may be unsigned or signed 2's complement integers.

XOR

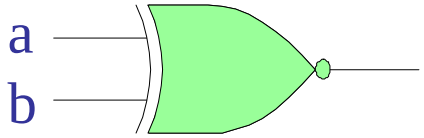


$D = a\,b' + a'\,b$

XOR		
a	b	D
0	0	0
0	1	1
1	0	1
1	1	0

XOR output D detects difference
D = 1, if $a \neq b$

XNOR



$E = a\,b + a'\,b'$

$E = D'$

XNOR		
a	b	E
0	0	1
0	1	0
1	0	0
1	1	1

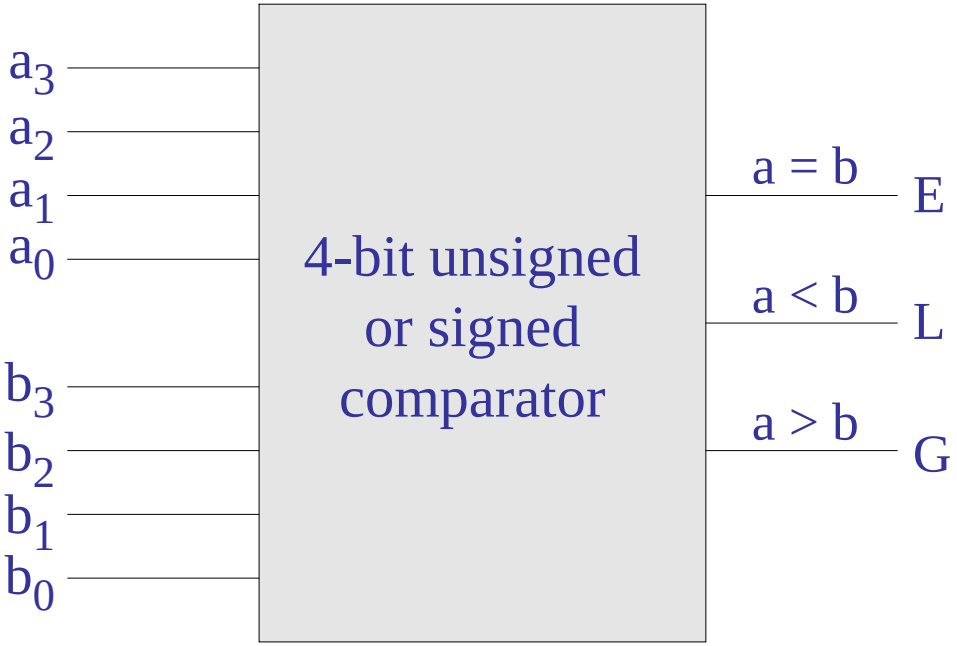
XNOR output E detects equality
E = 1, if a = b

Comparators

Design a comparator for 4-bit unsigned integers, and then modify it for 2's complement integers

$a = (a_3\, a_2\, a_1\, a_0)$

$b = (b_3\, b_2\, b_1\, b_0)$



see also, https://en.wikipedia.org/wiki/Digital_comparator

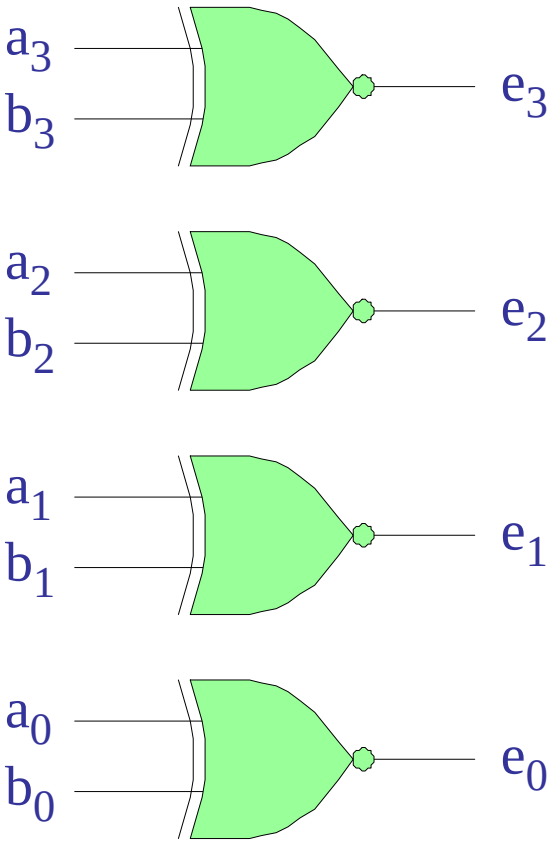
Comparators

Design a comparator for 4-bit unsigned integers, and then modify it for 2's complement integers

$a = (a_3\, a_2\, a_1\, a_0)$

$b = (b_3\, b_2\, b_1\, b_0)$

we may use four XNORs to detect equality of the individual bits,



equality of all four bits is indicated by the AND operation,

$E_{a=b} = e_3\, e_2\, e_1\, e_0$

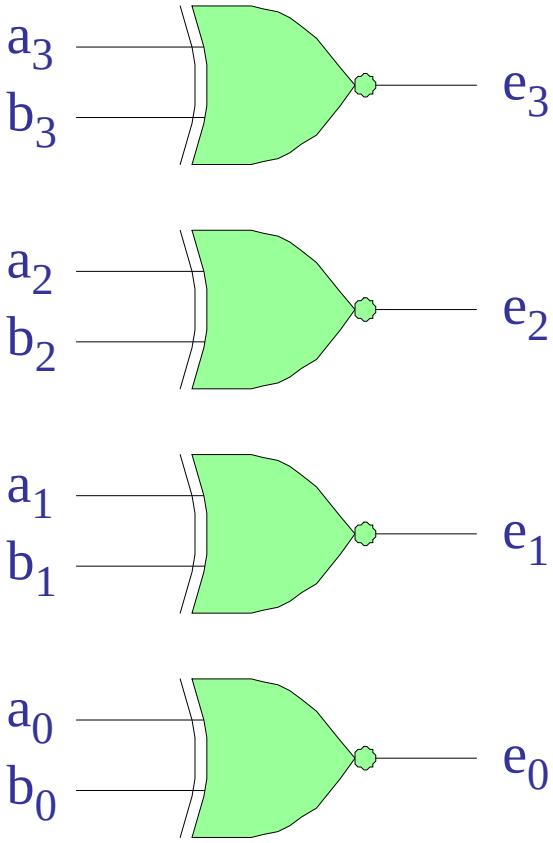
see also, https://en.wikipedia.org/wiki/Digital_comparator

Comparators

$a = (a_3\, a_2\, a_1\, a_0)$

$b = (b_3\, b_2\, b_1\, b_0)$

inequality (for unsigned, non-negative, integers) can be tested with the help of the individual XNOR outputs



$a > b$ is true if any of the following is true,

$a_3 = 1, b_3 = 0$

$a_3 = b_3$, and, $a_2 = 1, b_2 = 0$

$a_3 = b_3$, $a_2 = b_2$, and, $a_1 = 1, b_1 = 0$

$a_3 = b_3$, $a_2 = b_2$, $a_1 = b_1$, and, $a_0 = 1, b_0 = 0$

or, phrased in terms of the XNOR outputs,

$a_3 = 1, b_3 = 0$

$e_3 = 1$, and, $a_2 = 1, b_2 = 0$

$e_3 = 1, e_2 = 1$, and, $a_1 = 1, b_1 = 0$

$e_3 = 1, e_2 = 1, e_1 = 1$, and, $a_0 = 1, b_0 = 0$

these can be combined into an overall expression,

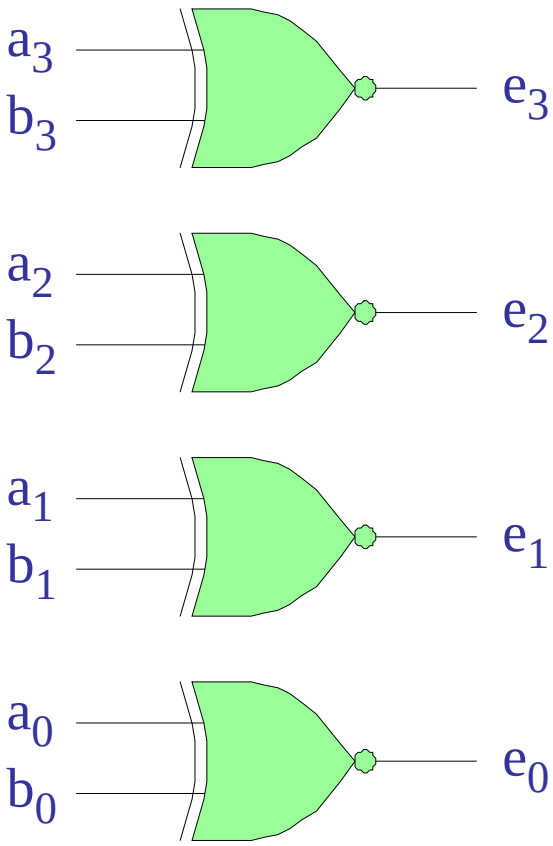
$$G_{a>b} = a_3\, b_3' + e_3\, a_2\, b_2' + e_3\, e_2\, a_1\, b_1' + e_3\, e_2\, e_1\, a_0\, b_0'$$

Comparators

$a = (a_3\, a_2\, a_1\, a_0)$

$b = (b_3\, b_2\, b_1\, b_0)$

inequality (for unsigned, non-negative, integers) can be tested with the help of the individual XNOR outputs



$a > b$ is true if

$a_3 = 1, b_3 = 0$

worst case:

$a = (1\ 0\ 0\ 0)$
$b = (0\ 1\ 1\ 1)$

$a = 2^3$
$b = 2^2 + 2^1 + 2^0 = (2^3 - 1) / (2 - 1) = 2^3 - 1$

thus, $a > b$

and, $G_{a>b} = 1 \cdot 1 + 0 + 0 + 0 = 1$

$$G_{a>b} = a_3\, b_3' + e_3\, a_2\, b_2' + e_3\, e_2\, a_1\, b_1' + e_3\, e_2\, e_1\, a_0\, b_0'$$

Comparators

x	bits
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
10	1 0 1 0
11	1 0 1 1
12	1 1 0 0
13	1 1 0 1
14	1 1 1 0
15	1 1 1 1

$a = (a_3\, a_2\, a_1\, a_0)$

$b = (b_3\, b_2\, b_1\, b_0)$

$a > b$ is true any of the following is true,

$a_3 = 1,\ b_3 = 0$

$a_3 = b_3$, and, $a_2 = 1,\ b_2 = 0$

$a_3 = b_3,\ a_2 = b_2$, and, $a_1 = 1,\ b_1 = 0$

$a_3 = b_3,\ a_2 = b_2,\ a_1 = b_1$, and, $a_0 = 1,\ b_0 = 0$

or, phrased in terms of the XNOR outputs,

$a_3 = 1,\ b_3 = 0$

$e_3 = 1$, and, $a_2 = 1,\ b_2 = 0$

$e_3 = 1,\ e_2 = 1$, and, $a_1 = 1,\ b_1 = 0$

$e_3 = 1,\ e_2 = 1,\ e_1 = 1$, and, $a_0 = 1,\ b_0 = 0$

these can be combined into an overall expression,

$$G_{a>b} = a_3\, b_3' + e_3\, a_2\, b_2' + e_3\, e_2\, a_1\, b_1' + e_3\, e_2\, e_1\, a_0\, b_0'$$

Comparators

x	bits			
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

$a = (a_3\, a_2\, a_1\, a_0)$

$b = (b_3\, b_2\, b_1\, b_0)$

$a > b$ is true any of the following is true,

$a_3 = 1,\ b_3 = 0$

$a_3 = b_3$, and, $a_2 = 1,\ b_2 = 0$

$a_3 = b_3,\ a_2 = b_2$, and, $a_1 = 1,\ b_1 = 0$

$a_3 = b_3,\ a_2 = b_2,\ a_1 = b_1$, and, $a_0 = 1,\ b_0 = 0$

or, phrased in terms of the XNOR outputs,

$a_3 = 1,\ b_3 = 0$

$e_3 = 1$, and, $a_2 = 1,\ b_2 = 0$

$e_3 = 1,\ e_2 = 1$, and, $a_1 = 1,\ b_1 = 0$

$e_3 = 1,\ e_2 = 1,\ e_1 = 1$, and, $a_0 = 1,\ b_0 = 0$

these can be combined into an overall expression,

$$G_{a>b} = a_3\, b_3' + e_3\, a_2\, b_2' + e_3\, e_2\, a_1\, b_1' + e_3\, e_2\, e_1\, a_0\, b_0'$$

Comparators

x	bits			
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

$a = (a_3\, a_2\, a_1\, a_0)$

$b = (b_3\, b_2\, b_1\, b_0)$

$a > b$ is true any of the following is true,

$a_3 = 1, b_3 = 0$

$a_3 = b_3$, and, $a_2 = 1, b_2 = 0$

$a_3 = b_3, a_2 = b_2$, and, $a_1 = 1, b_1 = 0$

$a_3 = b_3, a_2 = b_2, a_1 = b_1$, and, $a_0 = 1, b_0 = 0$

or, phrased in terms of the XNOR outputs,

$a_3 = 1, b_3 = 0$

$e_3 = 1$, and, $a_2 = 1, b_2 = 0$

$e_3 = 1, e_2 = 1$, and, $a_1 = 1, b_1 = 0$

$e_3 = 1, e_2 = 1, e_1 = 1$, and, $a_0 = 1, b_0 = 0$

these can be combined into an overall expression,

$$G_{a>b} = a_3\, b_3' + e_3\, a_2\, b_2' + e_3\, e_2\, a_1\, b_1' + e_3\, e_2\, e_1\, a_0\, b_0'$$

Comparators

$a < b$ can be constructed in terms of the $a = b$ and $a > b$ expressions,

$$L_{a<b} = E_{a=b}' \; G_{a>b}' = (E_{a=b} + G_{a>b})' = \text{NOR operation}$$

Summary – 4-bit unsigned comparator

$$e_3 = (a_3 \oplus b_3)'$$
$$e_2 = (a_2 \oplus b_2)'$$
$$e_1 = (a_1 \oplus b_1)'$$
$$e_0 = (a_0 \oplus b_0)'$$

math notation:

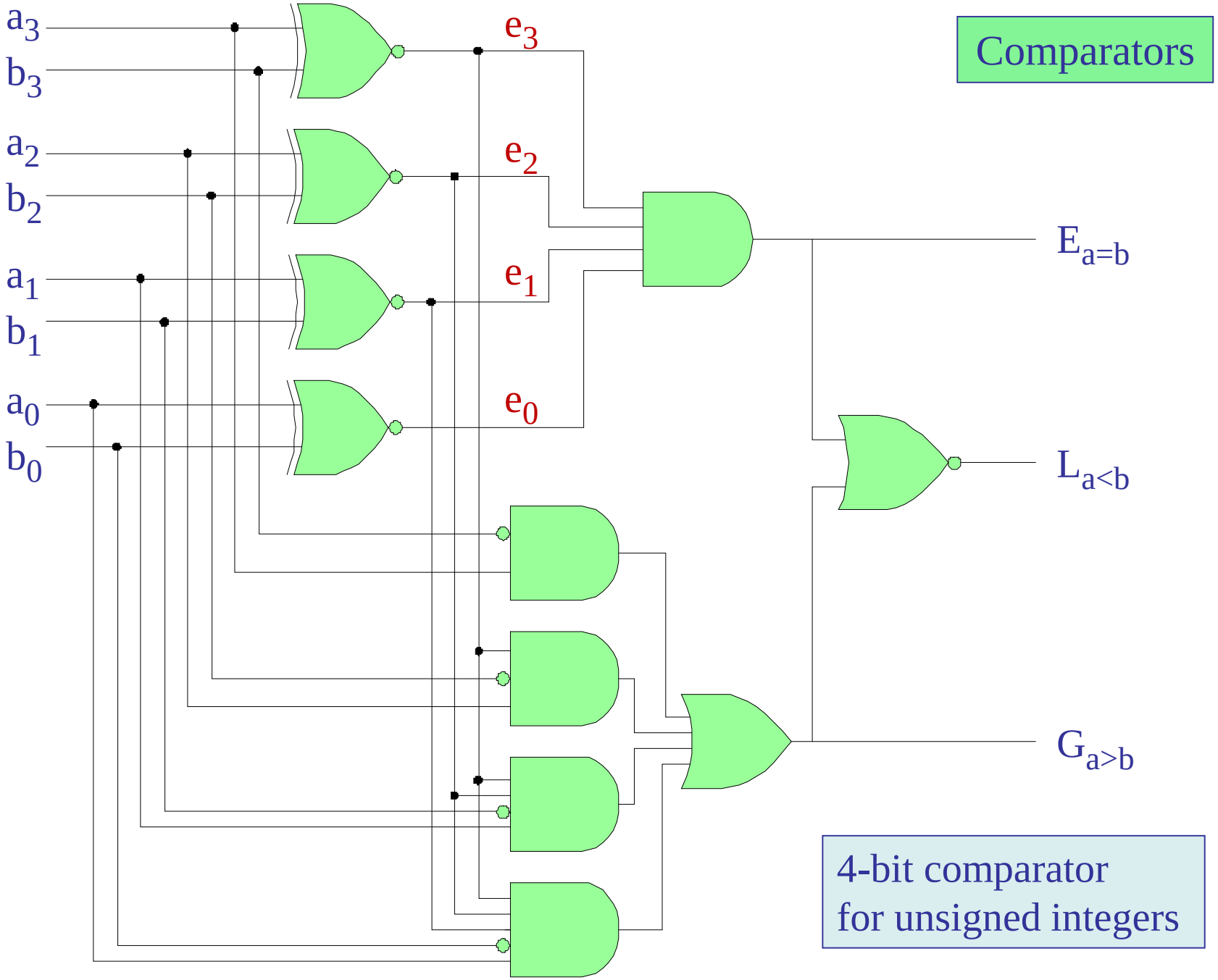
XOR(a, b) = $a \oplus b$

XNOR(a, b) = $(a \oplus b)'$

$$E_{a=b} = e_3 \, e_2 \, e_1 \, e_0$$
$$G_{a>b} = a_3 \, b_3' + e_3 \, a_2 \, b_2' + e_3 \, e_2 \, a_1 \, b_1' + e_3 \, e_2 \, e_1 \, a_0 \, b_0'$$
$$L_{a<b} = (E_{a=b} + G_{a>b})'$$

overall realization shown on next page

$$E + G + L = 1, \quad \text{and,} \quad E\,G = E\,L = G\,L = 0$$


Comparators
a_3
b_3
a_2
b_2
a_1
b_1
a_0
b_0
e_3
e_2
e_1
e_0
$E_{a=b}$
$L_{a<b}$
$G_{a>b}$
4-bit comparator
for unsigned integers

Comparators

for the 2's complement case,

if $a_3 = b_3$, then use the unsigned version

if $a_3 \neq b_3$, then must invert both $L_{a<b}$ and $G_{a>b}$

and may introduce a selector input u for choosing the unsigned (u=1) or the signed case (u=0)

$c = u + e_3$

$L_{a<b} = (c \oplus L)'$

$G_{a>b} = (c \oplus G)'$

u	e_3	c	invert L,G
0	0	0	invert
0	1	1	do not invert
1	0	1	do not invert
1	1	1	do not invert

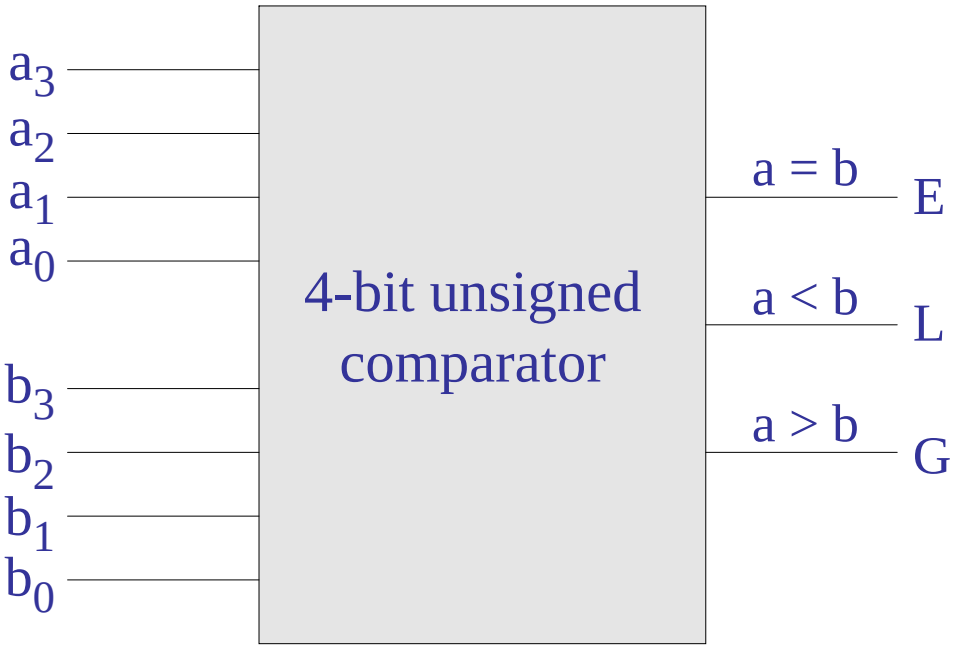
use XOR properties

$0 \oplus L = L$

$0 \oplus G = G$

$1 \oplus L = L'$

$1 \oplus G = G'$



Comparators

x	mod(x,2^4)	bits
0	0	0 0 0 0
1	1	0 0 0 1
2	2	0 0 1 0
3	3	0 0 1 1
4	4	0 1 0 0
5	5	0 1 0 1
6	6	0 1 1 0
7	7	0 1 1 1
8	-8	1 0 0 0
9	-7	1 0 0 1
10	-6	1 0 1 0
11	-5	1 0 1 1
12	-4	1 1 0 0
13	-3	1 1 0 1
14	-2	1 1 1 0
15	-1	1 1 1 1

in the 2's complement representation, the mod-operation maps all negative integers to their unsigned versions with the same bit pattern, thus,

if, $a_3 = b_3$, (a, b are both positive or both negative), then, the sense of the inequality is preserved ($L_{a<b} = L$), e.g.,

$4 > 3$

$-4 < -3$, with unsigned version, $12 < 13$

if, $a_3 \neq b_3$, then the inequality direction is reversed ($L_{a<b} = L'$), e.g.,

$-4 < 3$, with unsigned version, $12 > 3$

$5 > -3$, with unsigned version, $5 < 13$

$$x = -b_3 2^3 + b_2 2^2 + b_1 2^1 + b_0$$

Comparators

x	mod(x,2^4)	bits
7	7	0 1 1 1
6	6	0 1 1 0
5	5	0 1 0 1
4	4	0 1 0 0
3	3	0 0 1 1
2	2	0 0 1 0
1	1	0 0 0 1
0	0	0 0 0 0
-1	15	1 1 1 1
-2	14	1 1 1 0
-3	13	1 1 0 1
-4	12	1 1 0 0
-5	11	1 0 1 1
-6	10	1 0 1 0
-7	9	1 0 0 1
-8	8	1 0 0 0

in the 2's complement representation, the mod-operation maps all negative integers to their unsigned versions with the same bit pattern, thus,

if, $a_3 = b_3$, (a, b are both positive or both negative), then, the sense of the inequality is preserved ($L_{a<b} = L$), e.g.,

$4 > 3$

$-4 < -3$, with unsigned version, $12 < 13$

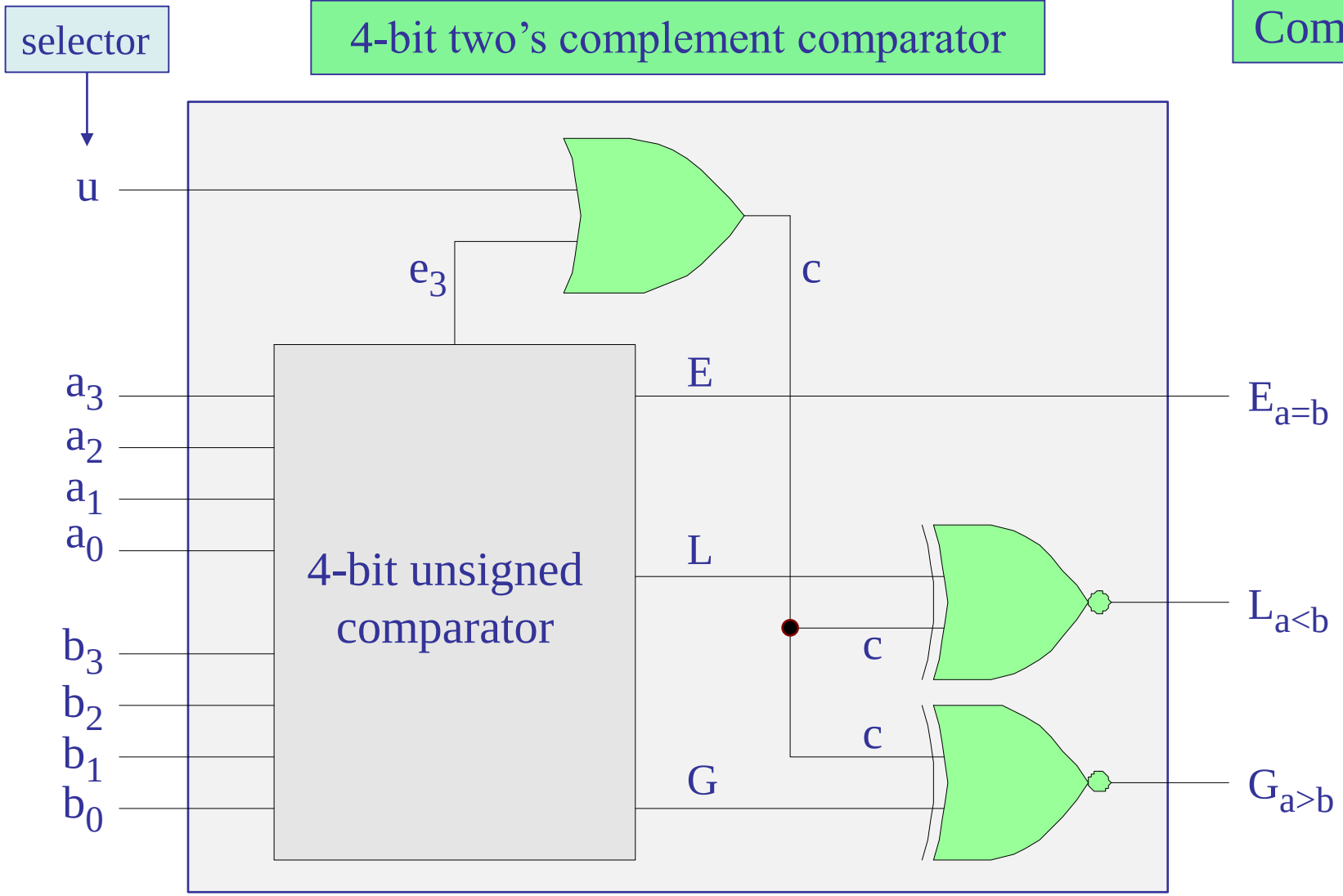
if, $a_3 \neq b_3$, then the inequality direction is reversed ($L_{a<b} = L'$), e.g.,

$-4 < 3$, with unsigned version, $12 > 3$

$5 > -3$, with unsigned version, $5 < 13$

$$x = -b_3 2^3 + b_2 2^2 + b_1 2^1 + b_0$$

4-bit two's complement comparator



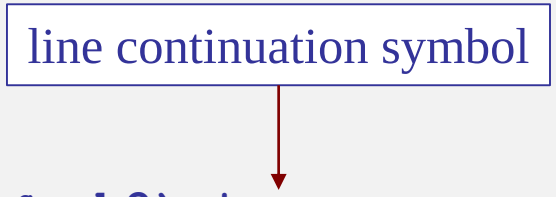
$c = u + e_3$

$L_{a<b} = (c \oplus L)'$

$G_{a>b} = (c \oplus G)'$

still valid: $E + G + L = 1$, $E\,G = 0$, $E\,L = 0$, $G\,L = 0$

```
% typical MATLAB code, given a=[a3,a2,a1,a0], b=[b3,b2,b1,b0]

a3 = a(1); a2 = a(2); a1 = a(3); a0 = a(4);
b3 = b(1); b2 = b(2); b1 = b(3); b0 = b(4);

e3 = ~xor(a3,b3);
e2 = ~xor(a2,b2);
e1 = ~xor(a1,b1);
e0 = ~xor(a0,b0);

c = u | e3;

E = e3 & e2 & e1 & e0;

G = (a3 & ~b3) | (e3 & a2 & ~b2) | ...
    (e3 & e2 & a1 & ~b1) | (e3 & e2 & e1 & a0 & ~b0);

L = ~(E | G);

G = ~xor(c,G);
L = ~xor(c,L);

% incorporated into the function, comp4.m, on Canvas Files
```



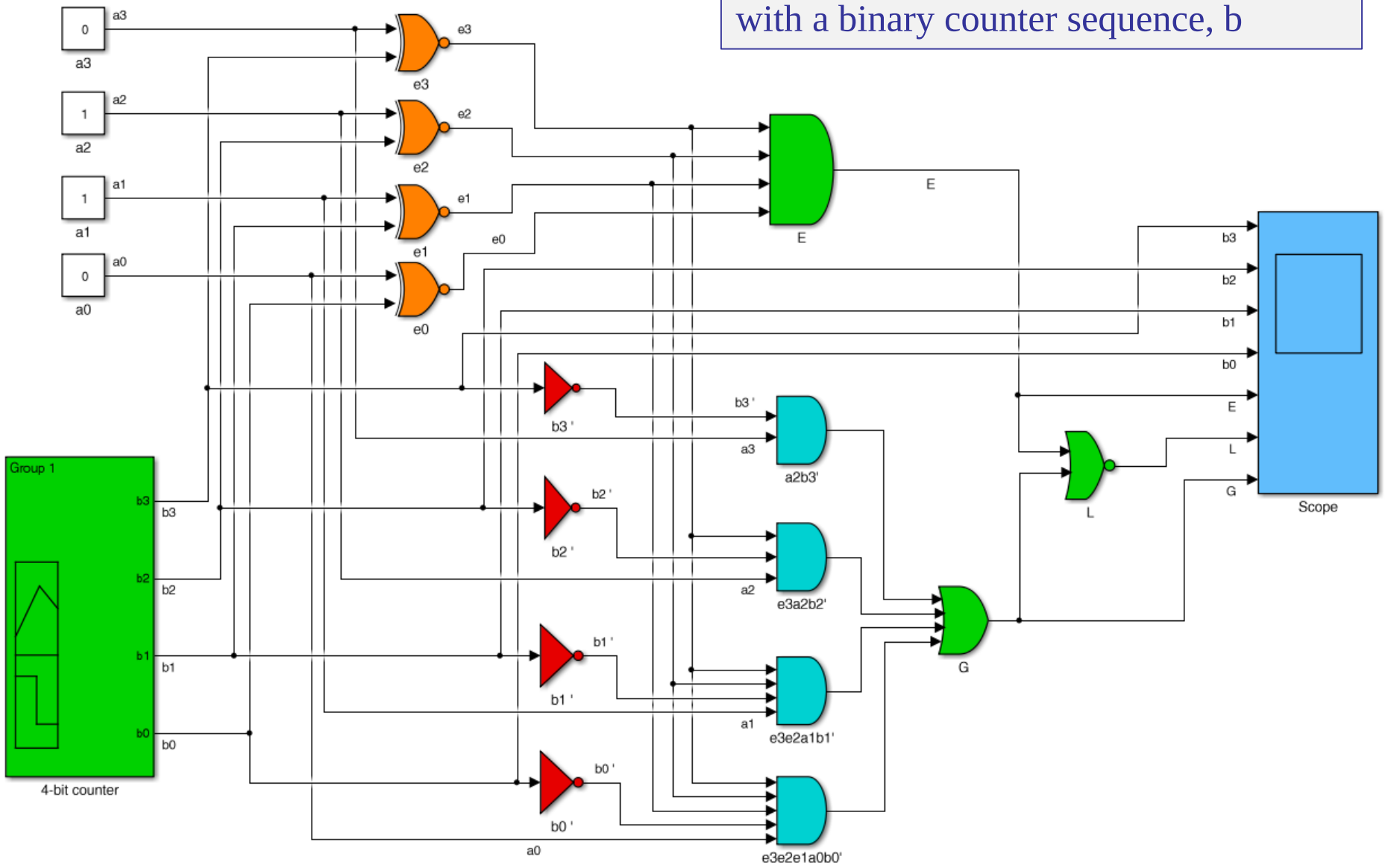
```
% comp4.m - 4-bit unsigned and 2's complement signed comparator
%
% Usage: [E,G,L] = comp4(a,b,u);
%
% a = 4-bit integer, a = [a3,a2,a1,a0]
%
% b = Lx4 matrix of bits, , each row representing an integer
%
% u = 1, 0, for unsigned or signed case, default u=1
%
% E,G,L = Lx1 vectors representing equal, greater than, less than
%
% Example: A = -3;
%          a = a2d(A,4);                              % a = [1 1 0 1]
%
%          [b3,b2,b1,b0] = a2d(-8:7, 4);              % 15x4 matrix
%
%          [E, G, L] = comp4(a,[b3,b2,b1,b0],0);      % comparisons
%
%          plotted below (p.28)
```

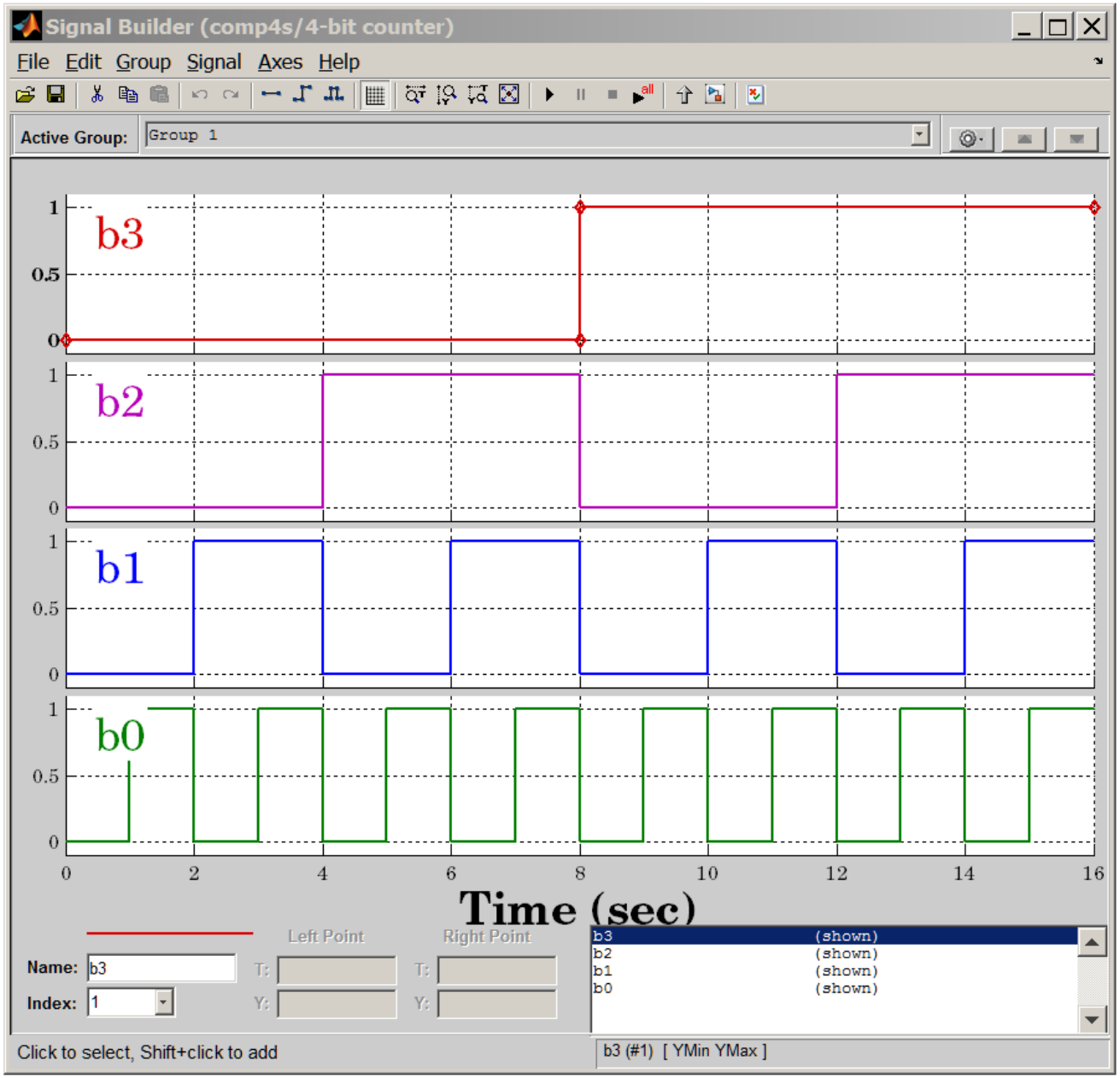
```
function [E,G,L] = comp4(a,b,u)

a3 = a(1); a2 = a(2); a1 = a(3); a0 = a(4);

for i=1:size(b,1)
   b3 = b(i,1); b2 = b(i,2); b1 = b(i,3); b0 = b(i,4);

   e3 = ~xor(a3,b3);  e2 = ~xor(a2,b2);
   e1 = ~xor(a1,b1);  e0 = ~xor(a0,b0);

   c = u | e3;

   E(i) = e3 & e2 & e1 & e0;

   G(i) = (a3 & ~b3) | (e3 & a2 & ~b2) | ...
          (e3 & e2 & a1 & ~b1) | (e3 & e2 & e1 & a0 & ~b0);

   L(i) = ~(E(i) | G(i));

   G(i) = ~xor(c,G(i));
   L(i) = ~xor(c,L(i));
end

E = E(:); G = G(:); L = L(:);
```

line continuation symbol

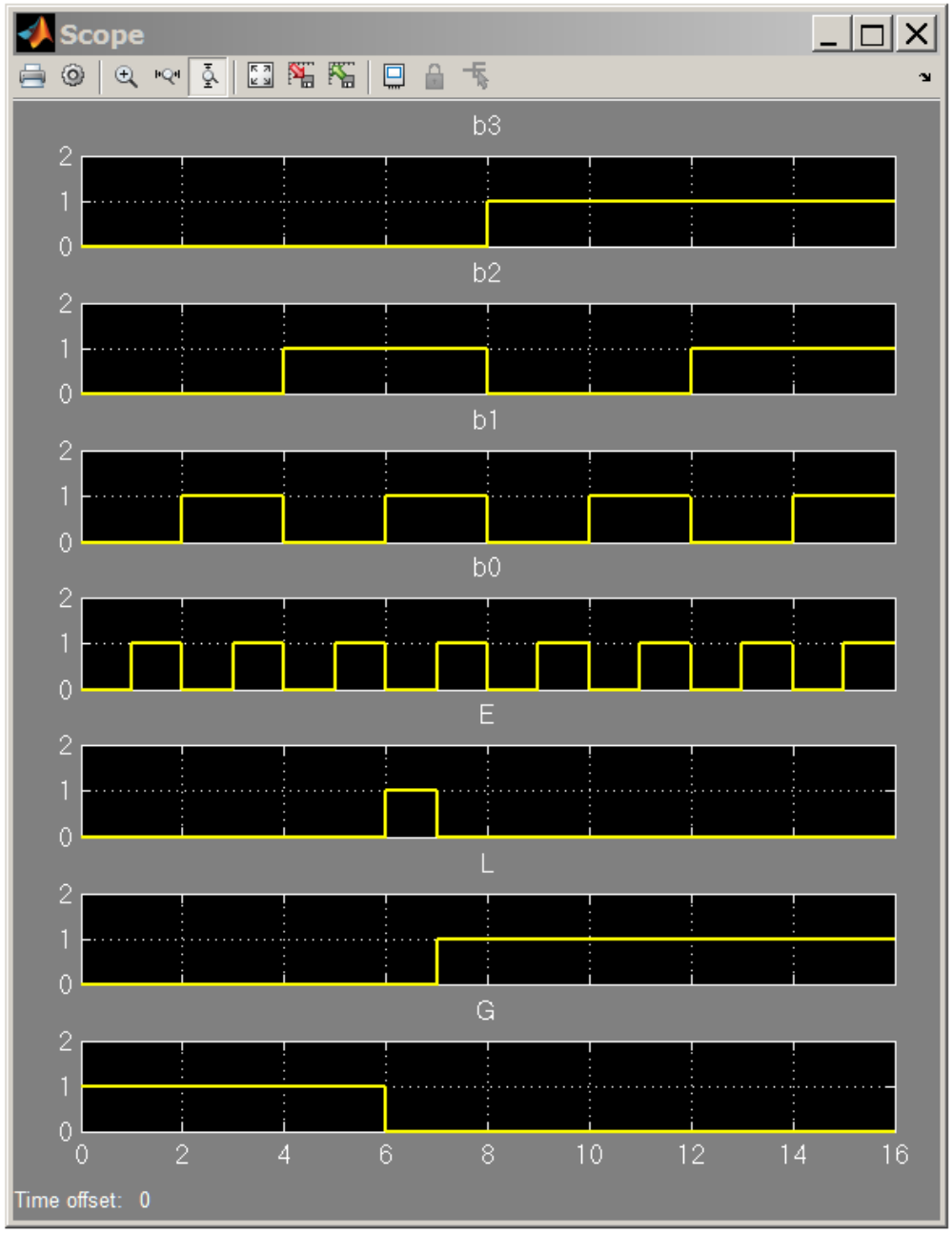
file: comp4s.slx on Canvas

Simulink implementation
successively compares the fixed number
a = 6 = (0 1 1 0)
with a binary counter sequence, b





binary-counter



E = 1, only when
b = a = 6 = (0 1 1 0)

L = 1, while
b > a = 6 = (0 1 1 0)

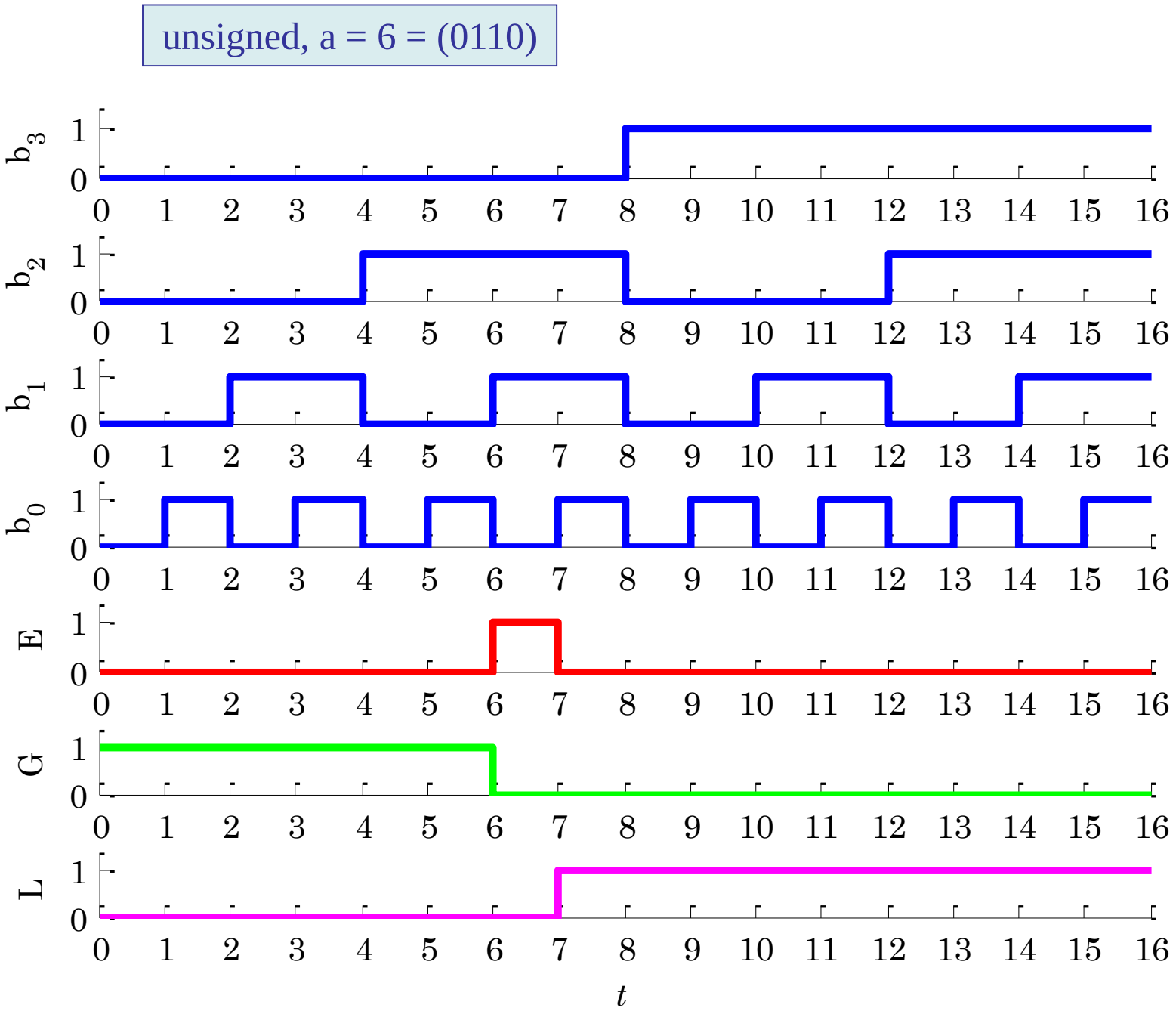
G = 1, while
b < a = 6 = (0 1 1 0)

unsigned

x	bits
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
10	1 0 1 0
11	1 0 1 1
12	1 1 0 0
13	1 1 0 1
14	1 1 1 0
15	1 1 1 1

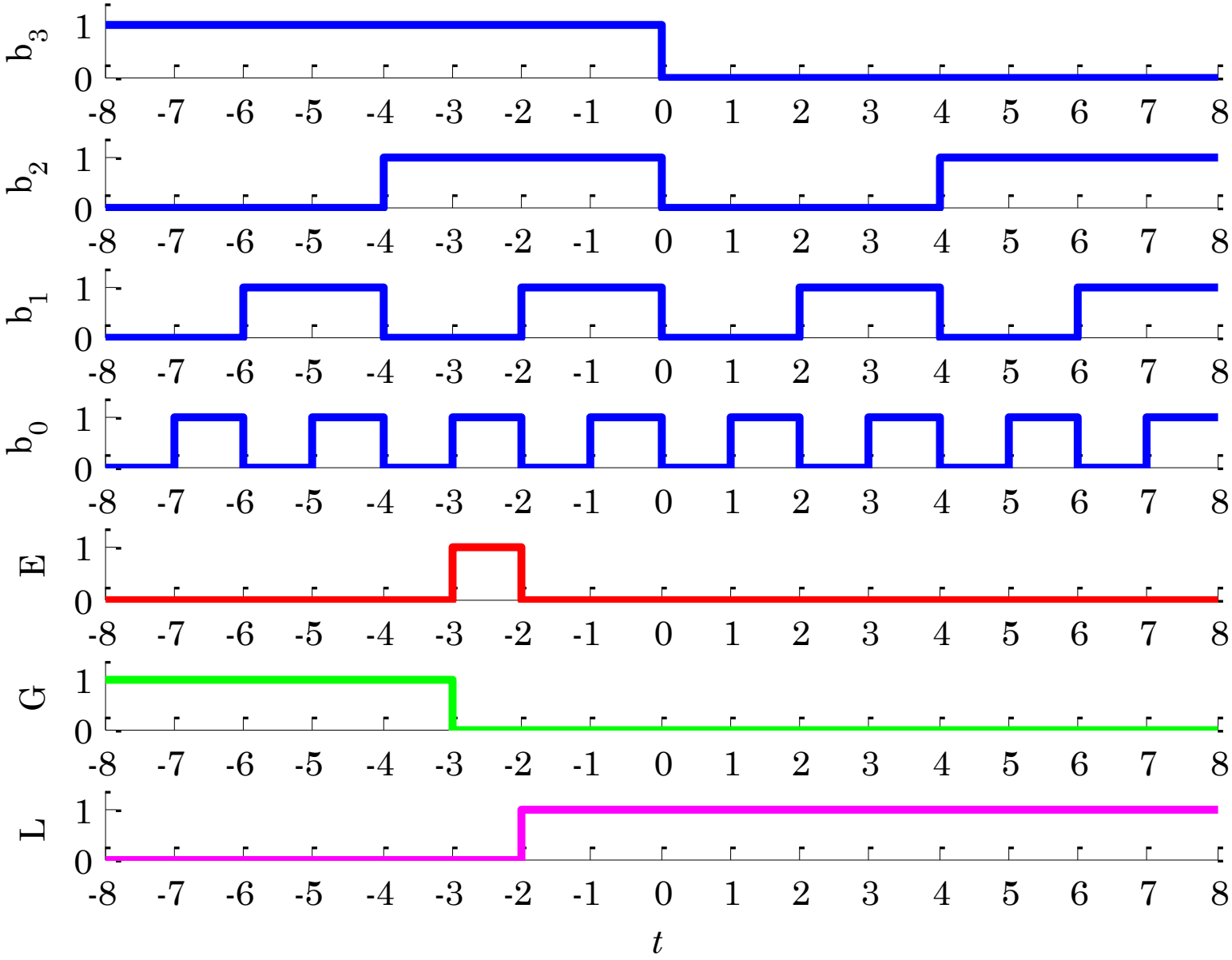
2's complement signed

x	mod(x,16)	bits
-8	8	1 0 0 0
-7	9	1 0 0 1
-6	10	1 0 1 0
-5	11	1 0 1 1
-4	12	1 1 0 0
-3	13	1 1 0 1
-2	14	1 1 1 0
-1	15	1 1 1 1
0	0	0 0 0 0
1	1	0 0 0 1
2	2	0 0 1 0
3	3	0 0 1 1
4	4	0 1 0 0
5	5	0 1 0 1
6	6	0 1 1 0
7	7	0 1 1 1


unsigned, a = 6 = (0110)
b3
b2
b1
b0
E
G
L
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
t

```
A = 6; a = a2d(A,4);                % a = [0 1 1 0]

[b3,b2,b1,b0] = a2d(0:15, 4);

[E,G,L] = comp4(a,[b3,b2,b1,b0],1);    % note, u = 1

E = [E; E(end)];        % extend duration of last bit
G = [G; G(end)];
L = [L; L(end)];
b3 = [b3; b3(end)];
b2 = [b2; b2(end)];
b1 = [b1; b1(end)];
b0 = [b0; b0(end)];

t = 0:16;               % last bit lasts from t=15 to t=16

figure;
subplot(7,1,1); stairs(t,b3,'b-'); ylabel('b_3');
subplot(7,1,2); stairs(t,b2,'b-'); ylabel('b_2');
subplot(7,1,3); stairs(t,b1,'b-'); ylabel('b_1');
subplot(7,1,4); stairs(t,b0,'b-'); ylabel('b_0');
subplot(7,1,5); stairs(t,E, 'r-'); ylabel('E');
subplot(7,1,6); stairs(t,G, 'g-'); ylabel('G');
subplot(7,1,7); stairs(t,L, 'm-'); ylabel('L');
xlabel('\itt');
```

2's complement, a = –3 = (1101)



```
A = -3; a = a2d(A,4);                % a = [1 1 0 1]

[b3,b2,b1,b0] = a2d(-8:7, 4);

[E,G,L] = comp4(a,[b3,b2,b1,b0],0);    % note, u = 0

E = [E; E(end)];        % extend duration of last bit
G = [G; G(end)];
L = [L; L(end)];
b3 = [b3; b3(end)];
b2 = [b2; b2(end)];
b1 = [b1; b1(end)];
b0 = [b0; b0(end)];

t = -8:8;               % last bit lasts from t=7 to t=8

figure;
subplot(7,1,1); stairs(t,b3,'b-'); ylabel('b_3');
subplot(7,1,2); stairs(t,b2,'b-'); ylabel('b_2');
subplot(7,1,3); stairs(t,b1,'b-'); ylabel('b_1');
subplot(7,1,4); stairs(t,b0,'b-'); ylabel('b_0');
subplot(7,1,5); stairs(t,E, 'r-'); ylabel('E');
subplot(7,1,6); stairs(t,G, 'g-'); ylabel('G');
subplot(7,1,7); stairs(t,L, 'm-'); ylabel('L');
xlabel('\itt');
```

MOTOROLA

SN54/74LS85

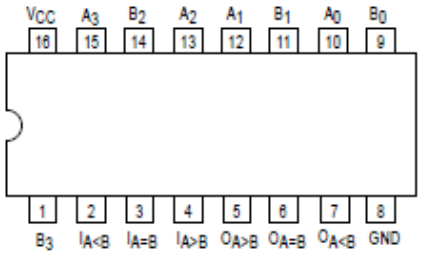
4-BIT MAGNITUDE COMPARATOR

The SN54/74LS85 is a 4-Bit Magnitude Camparator which compares two 4-bit words (A, B), each word having four Parallel Inputs (A_0-A_3, B_0-B_3); A_3, B_3 being the most significant inputs. Operation is not restricted to binary codes, the device will work with any monotonic code. Three Outputs are provided: "A greater than B" ($O_{A>B}$), "A less than B" ($O_{A<B}$), "A equal to B" ($O_{A=B}$). Three Expander Inputs, $I_{A>B}$, $I_{A<B}$, $I_{A=B}$, allow cascading without external gates. For proper compare operation, the Expander Inputs to the least significant position must be connected as follows: $I_{A<B}=I_{A>B}=L$, $I_{A=B}=H$. For serial (ripple) expansion, the $O_{A>B}$, $O_{A<B}$ and $O_{A=B}$ Outputs are connected respectively to the $I_{A>B}$, $I_{A<B}$, and $I_{A=B}$ Inputs of the next most significant comparator, as shown in Figure 1. Refer to Applications section of data sheet for high speed method of comparing large words.

The Truth Table on the following page describes the operation of the SN54/74LS85 under all possible logic conditions. The upper 11 lines describe the normal operation under all conditions that will occur in a single device or in a series expansion scheme. The lower five lines describe the operation under abnormal conditions on the cascading inputs. These conditions occur when the parallel expansion technique is used.

- Easily Expandable
- Binary or BCD Comparison
- $O_{A>B}$, $O_{A<B}$, and $O_{A=B}$ Outputs Available

CONNECTION DIAGRAM DIP (TOP VIEW)



NOTE:
The Flatpak version has the same pinouts (Connection Diagram) as the Dual In-Line Package.

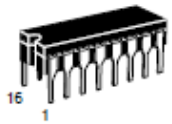
PIN NAMES

		LOADING (Note a) HIGH	LOW
A_0-A_3, B_0-B_3	Parallel Inputs	1.5 U.L.	0.75 U.L.
$I_{A=B}$	A = B Expander Inputs	1.5 U.L.	0.75 U.L.
$I_{A<B}$, $I_{A>B}$	A < B, A > B, Expander Inputs	0.5 U.L.	0.25 U.L.
$O_{A>B}$	A Greater Than B Output (Note b)	10 U.L.	5 (2.5) U.L.
$O_{A<B}$	B Greater Than A Output (Note b)	10 U.L.	5 (2.5) U.L.
$O_{A=B}$	A Equal to B Output (Note b)	10 U.L.	5 (2.5) U.L.

NOTES:
a) 1 TTL Unit Load (U.L.) = 40 µA HIGH/1.6 mA LOW.
b) The Output LOW drive factor is 2.5 U.L. for Military (54) and 5 U.L. for Commercial (74) Temperature Ranges.

4-BIT MAGNITUDE COMPARATOR

LOW POWER SCHOTTKY



J SUFFIX
CERAMIC
CASE 620-09

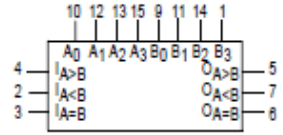
N SUFFIX
PLASTIC
CASE 648-08

D SUFFIX
SOIC
CASE 751B-03

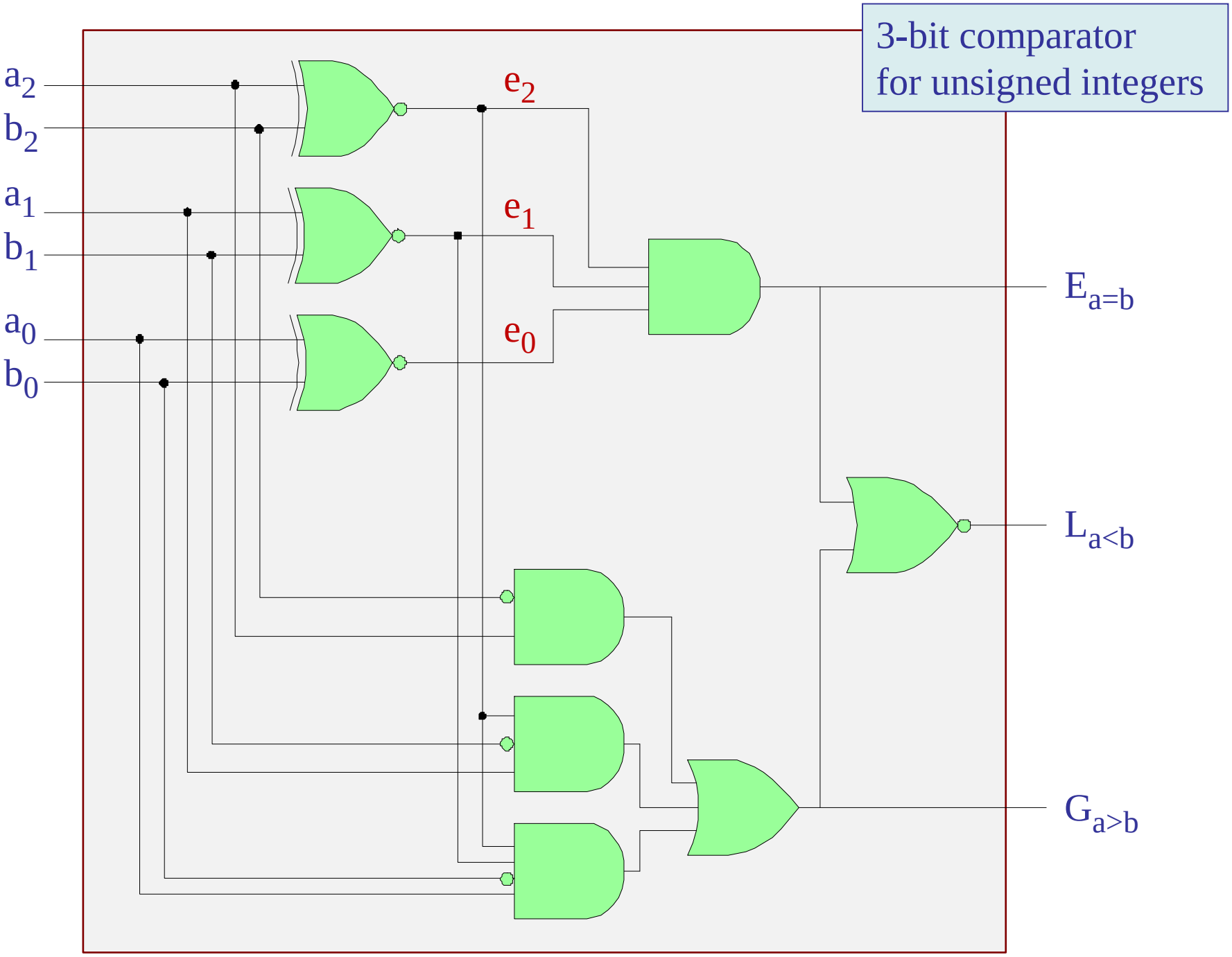
ORDERING INFORMATION

SN54LSXXJ	Ceramic
SN74LSXXN	Plastic
SN74LSXXD	SOIC

LOGIC SYMBOL



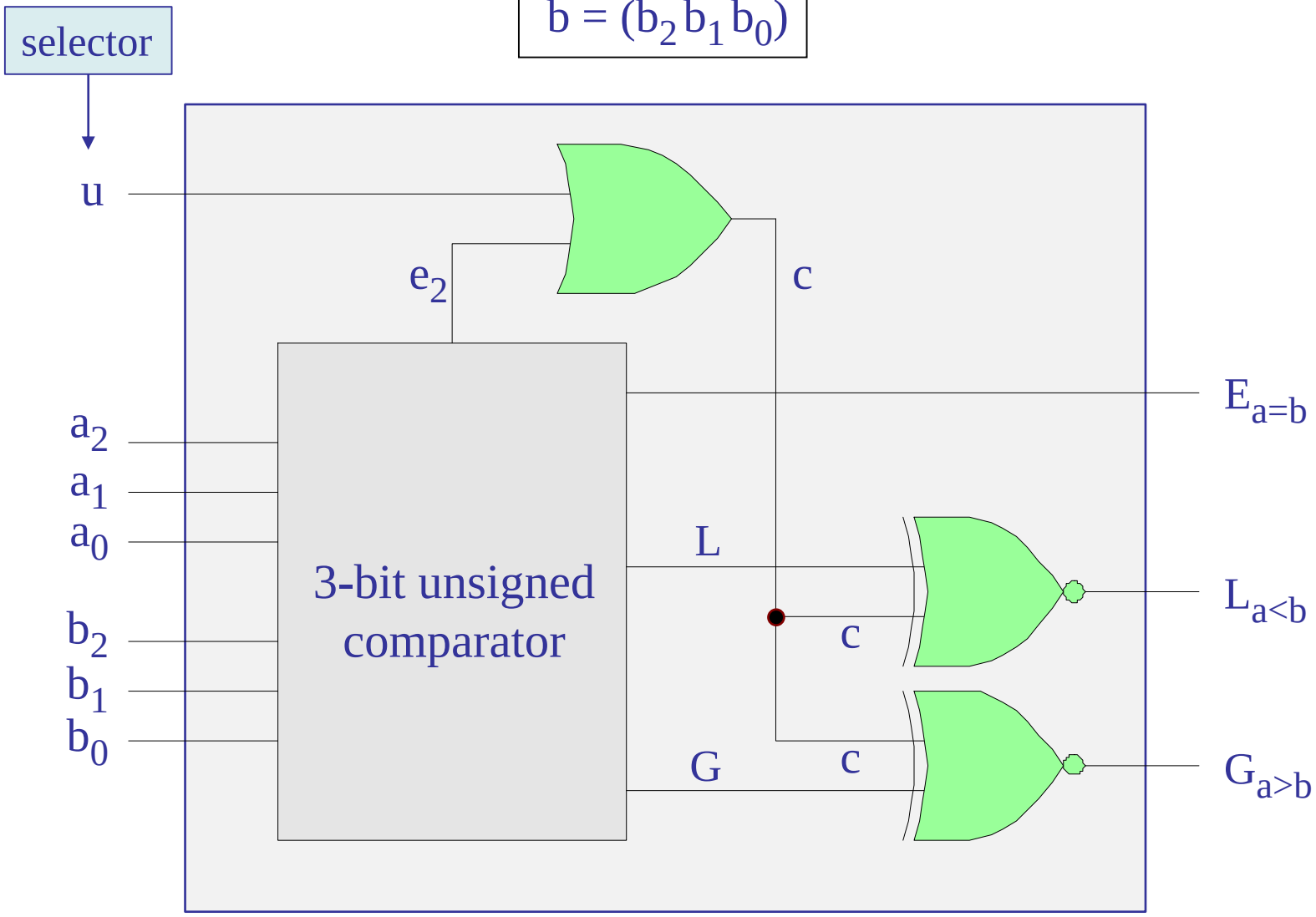
V_{CC} = PIN 16
GND = PIN 8


3-bit comparator
for unsigned integers
a2
b2
a1
b1
a0
b0
e2
e1
e0
Ea=b
La<b
Ga>b

3-bit comparator for 2's complement signed integers

$a = (a_2\, a_1\, a_0)$

$b = (b_2\, b_1\, b_0)$

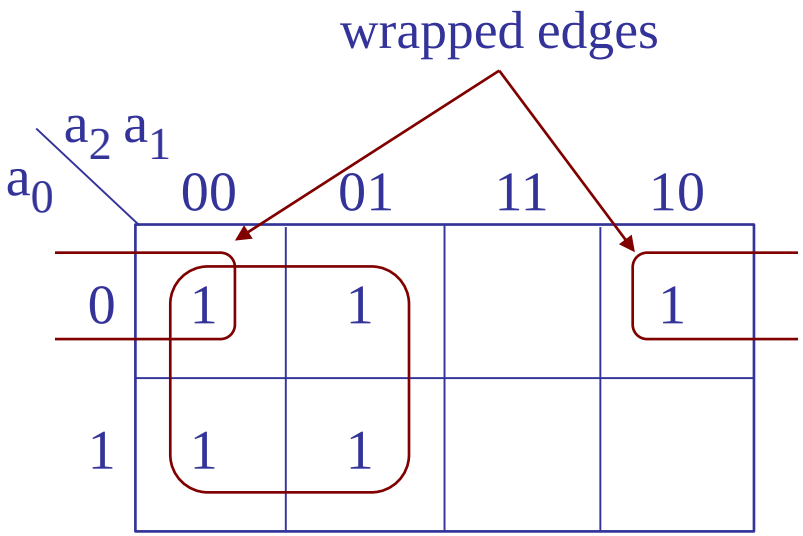


Additional Examples - Problem 1

Design a combinational circuit that detects whether a 3-bit number, $a = [a_2, a_1, a_0]$, is less than 5, that is, $a < 5$, or, $a \leq 4$.

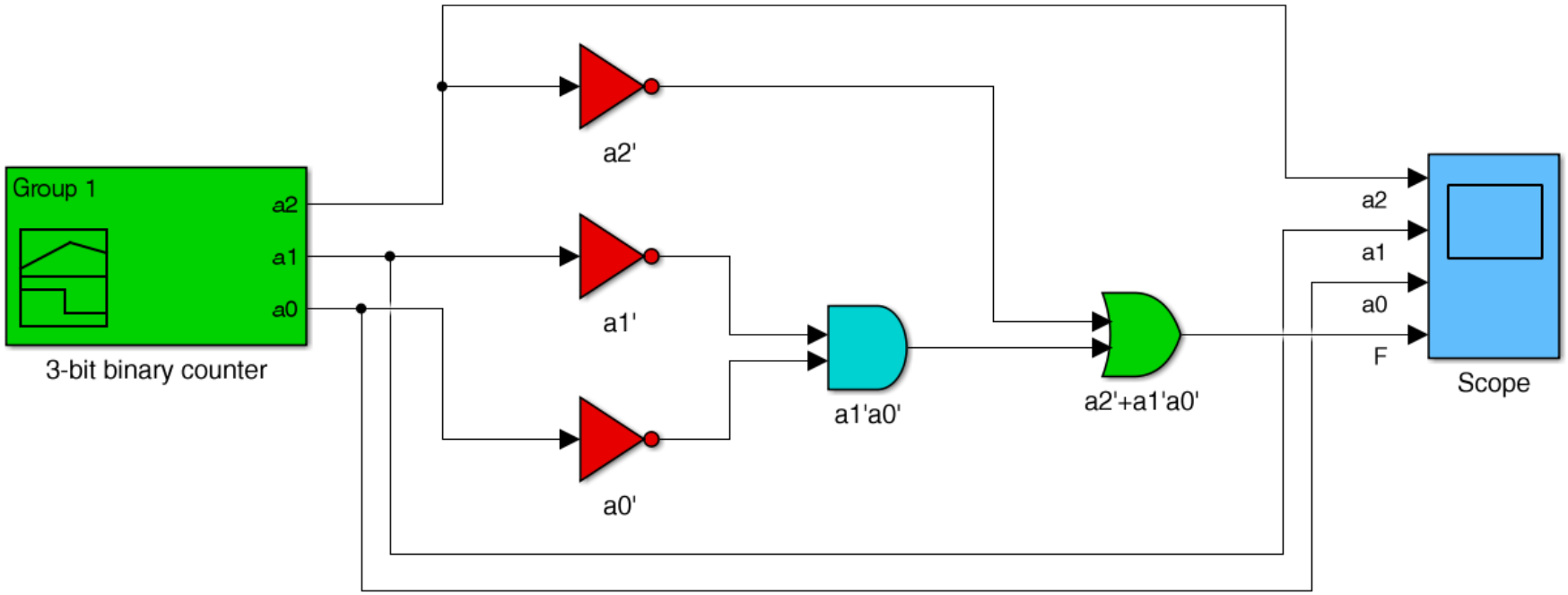
Solution: Start with a truth-table and a K-map

a	a2	a1	a0	F
0	0	0	0	1
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	0

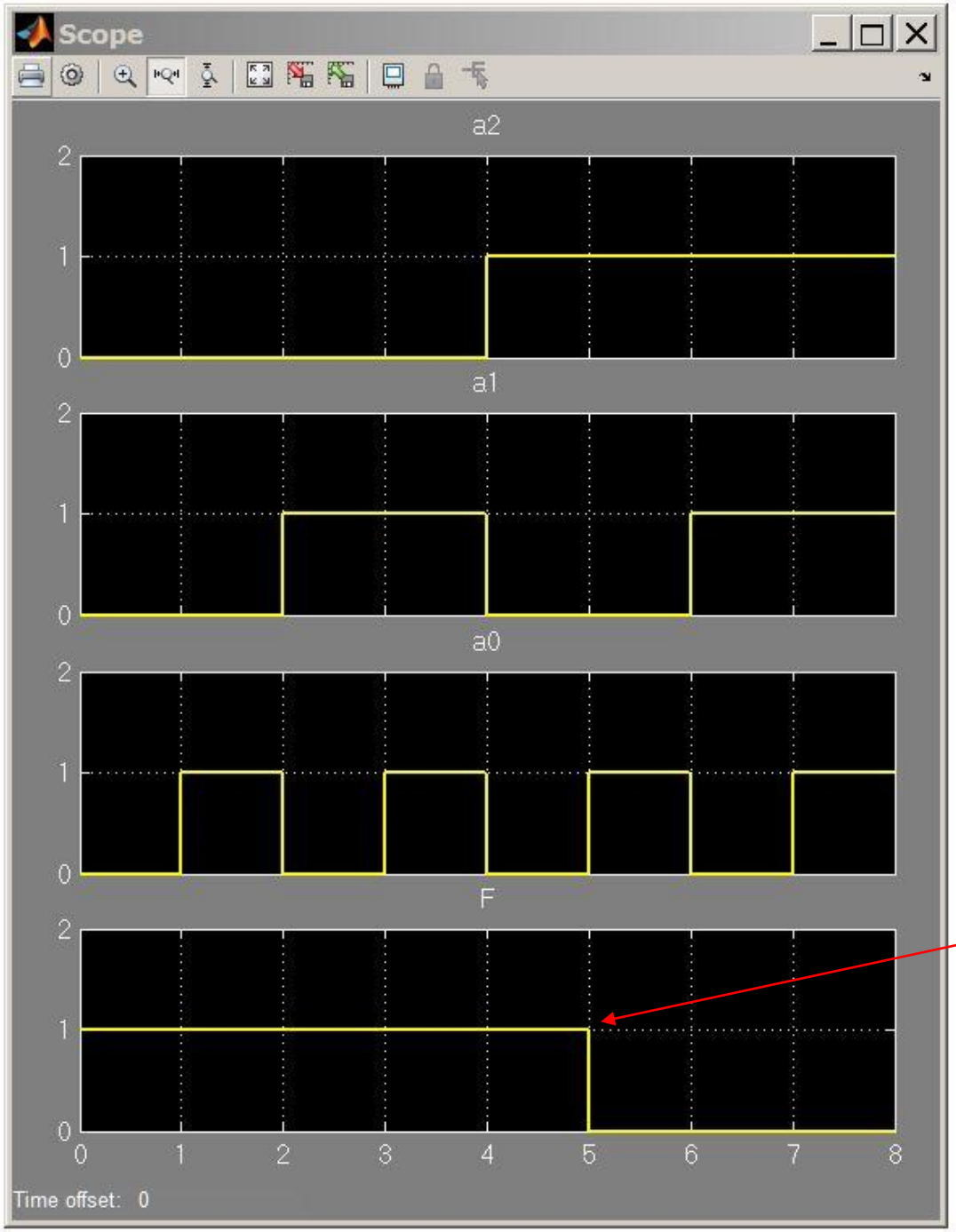


$F = a_2' + a_1' a_0'$

Implement with Simulink



$$F = a_2' + a_1' a_0'$$



$F = a_2' + a_1' a_0'$

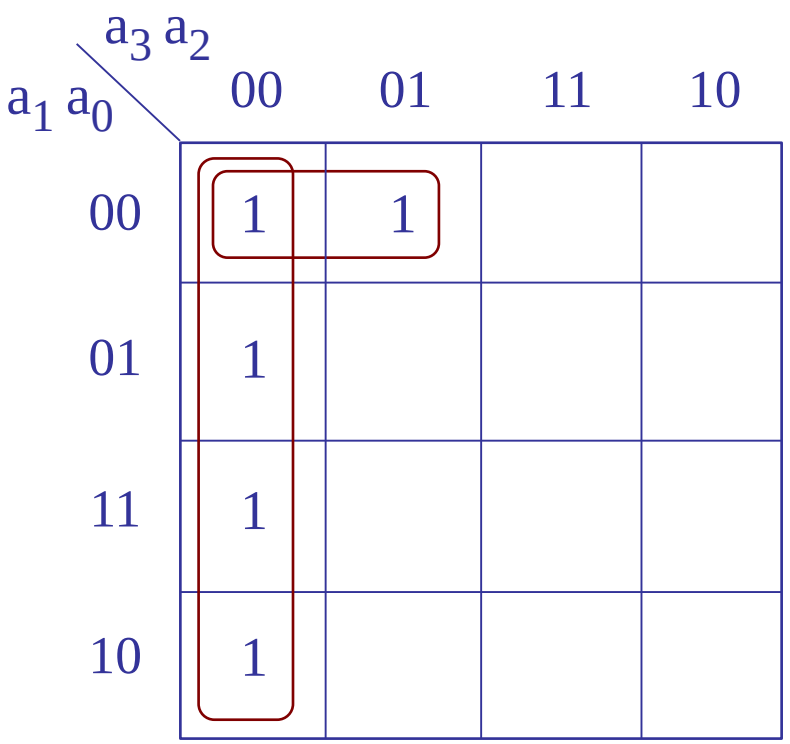
t=4 extends to t=5

Additional Examples - Problem 2

Design a combinational circuit that detects whether a 4-bit number, $a = [a_3, a_2, a_1, a_0]$, is less than 5, that is, $a < 5$, or, $a \leq 4$.

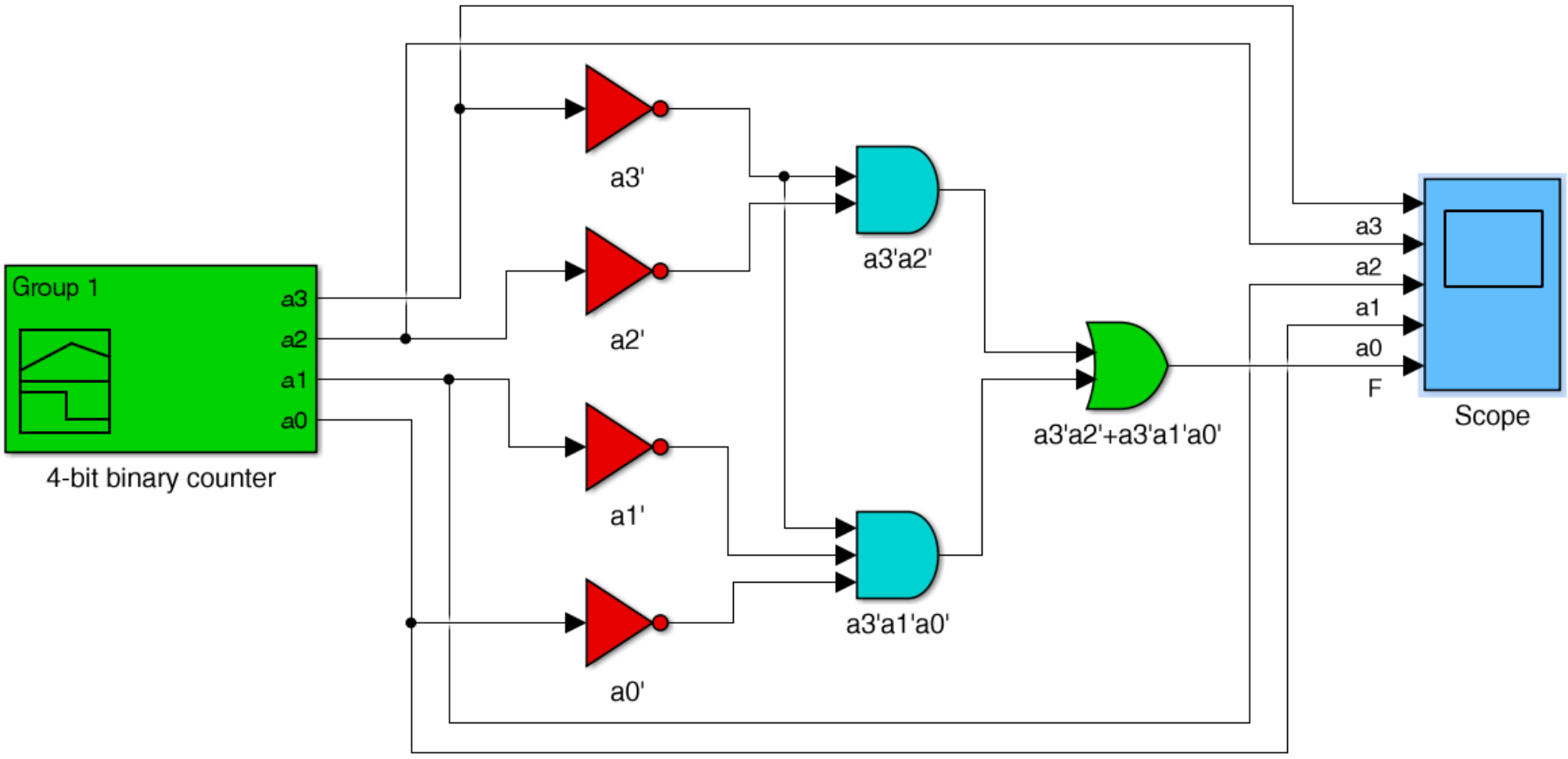
a	a3	a2	a1	a0	F
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

Solution

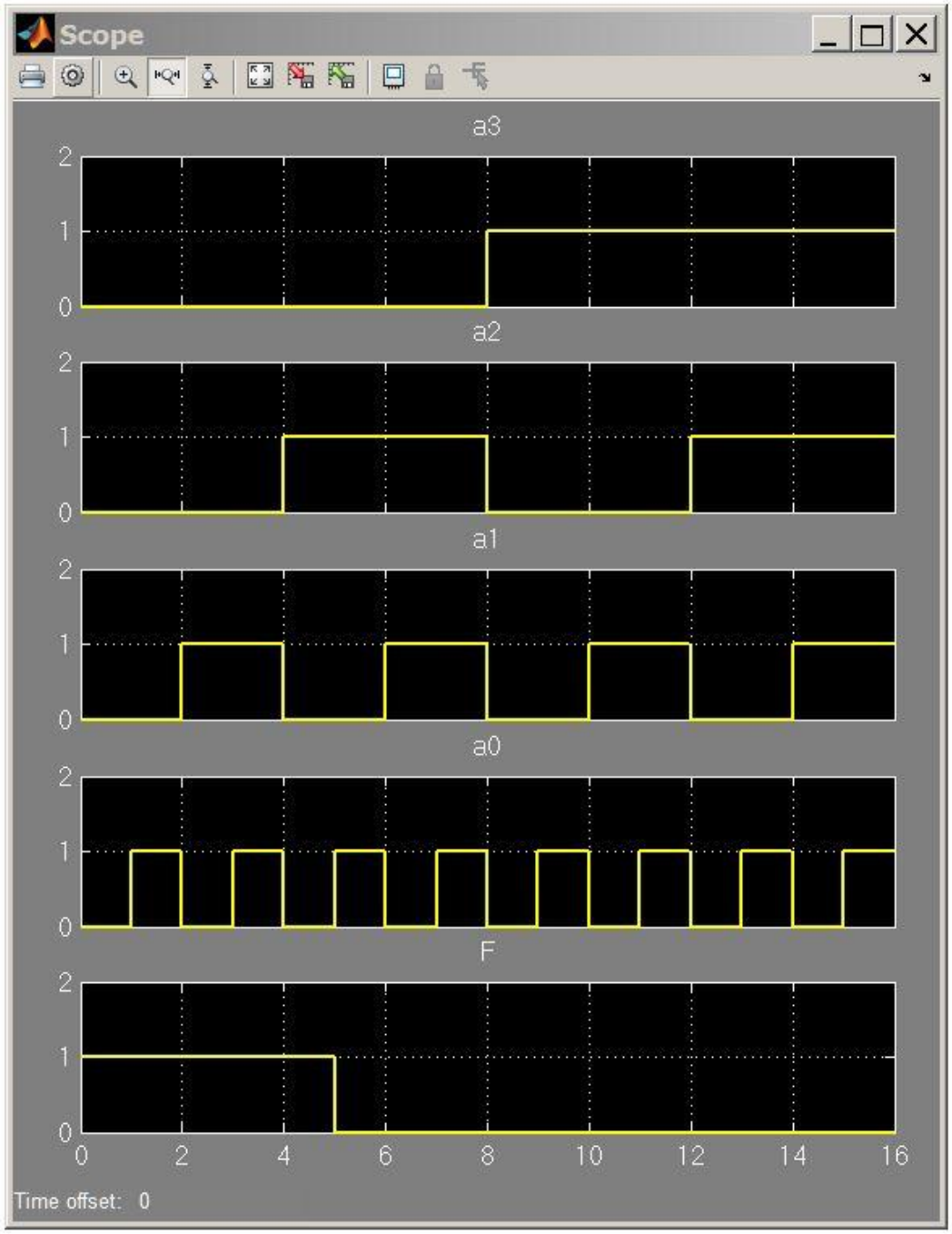


$F = a_3' a_2' + a_3' a_1' a_0'$

Implement with Simulink



$$F = a_3' a_2' + a_3' a_1' a_0'$$



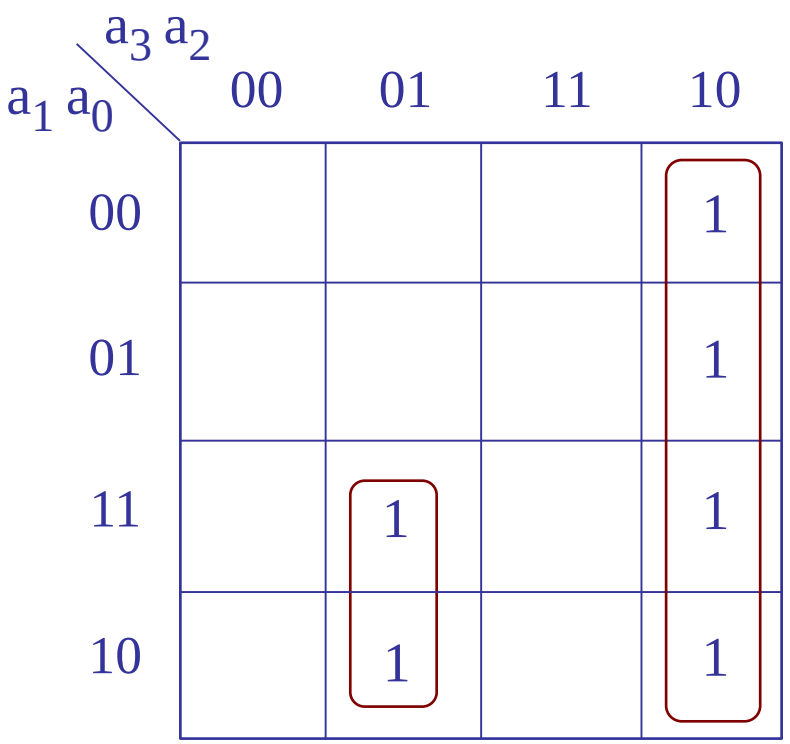
$$F = a_3' a_2' + a_3' a_1' a_0'$$

Additional Examples - Problem 3

Design a combinational circuit that detects whether a 4-bit number, $a = [a_3, a_2, a_1, a_0]$, is in the range, $5 < a < 12$, or, $6 \leq a \leq 11$.

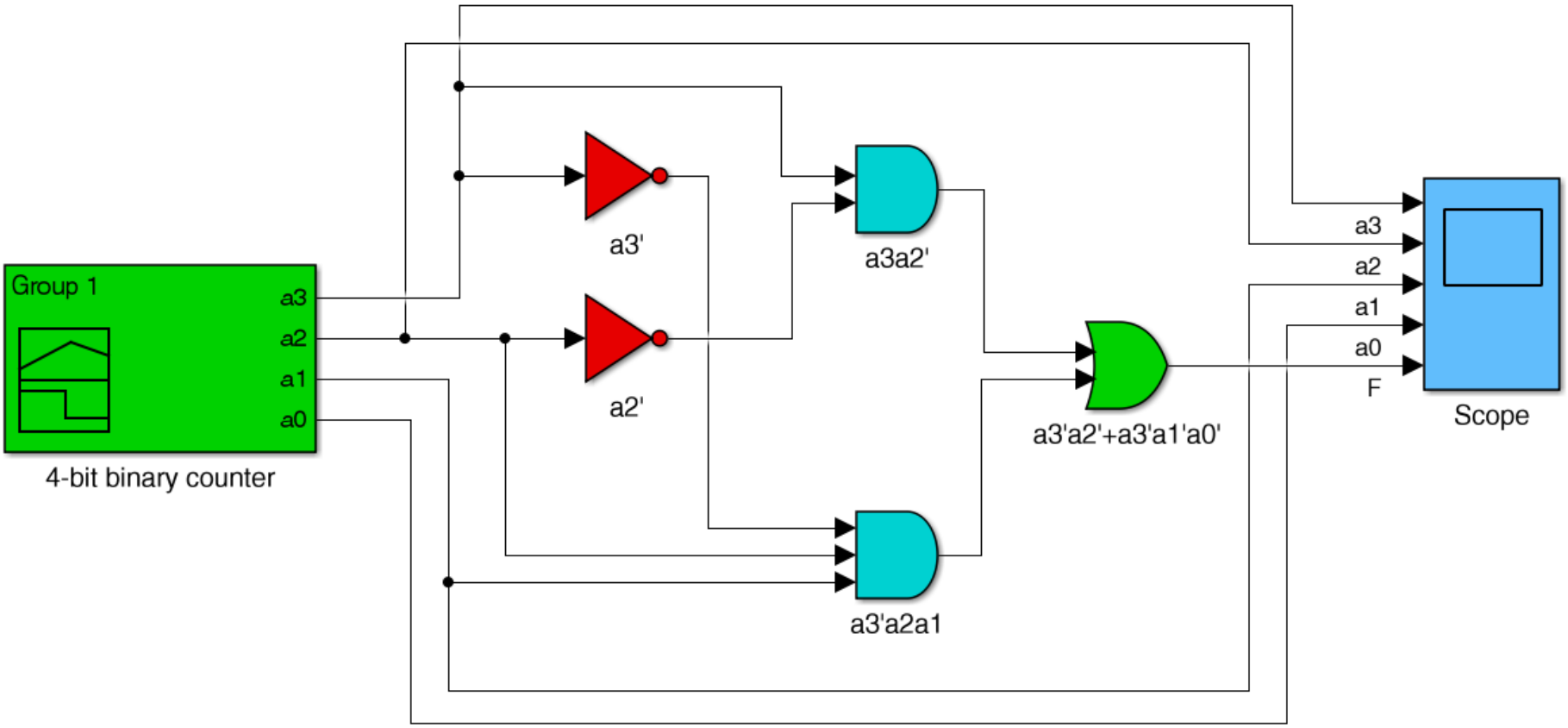
a	a3	a2	a1	a0	F
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

Solution

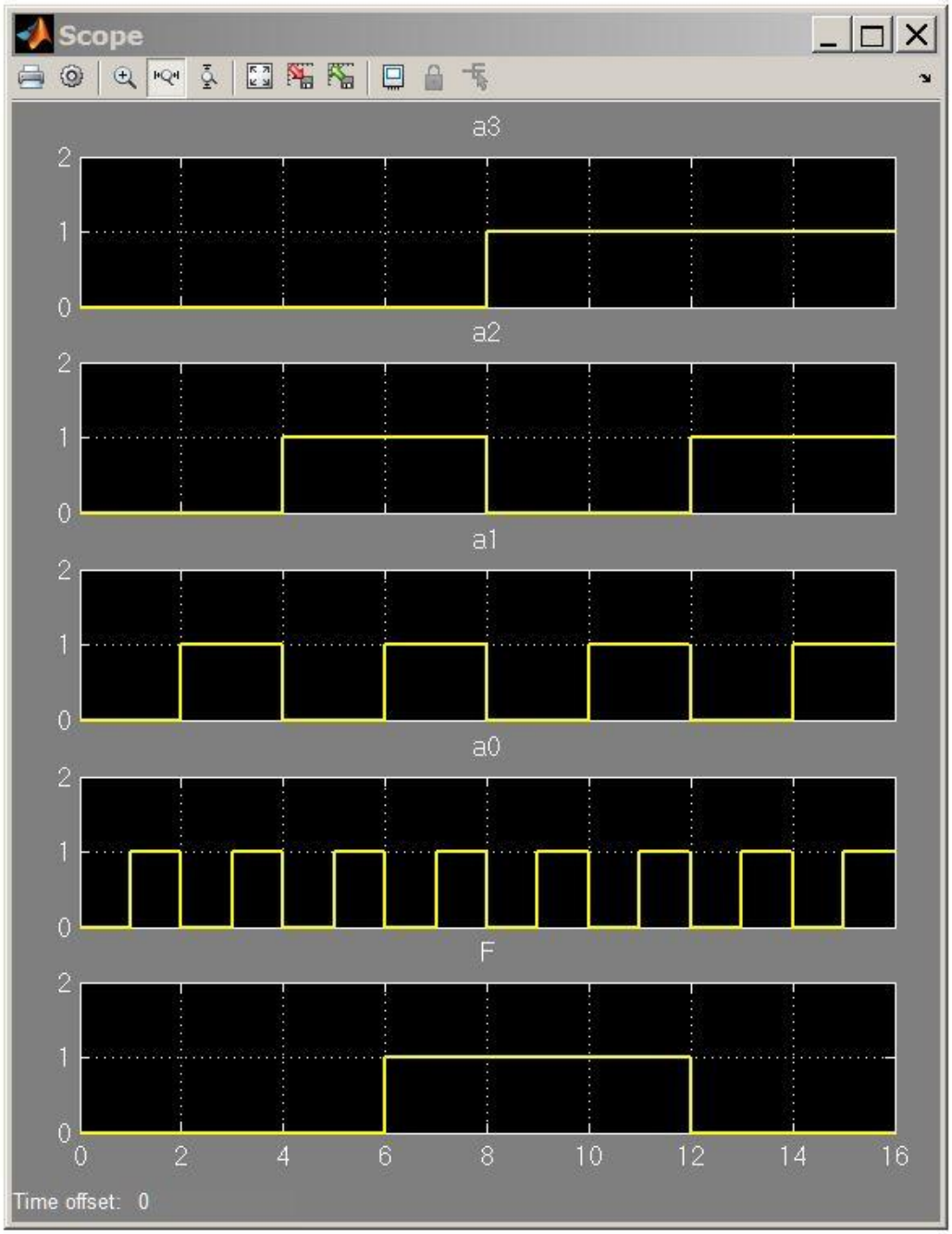


$F = a_3 a_2' + a_3' a_2 a_1$

Implement with Simulink



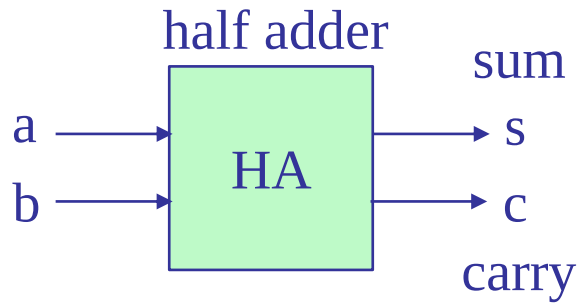
$$F = a_3\, a_2' + a_3'\, a_2\, a_1$$



$$F = a_3\, a_2' + a_3'\, a_2\, a_1$$

Addition and Subtraction – Half-Adders

Wakerly, Sect. 8.1



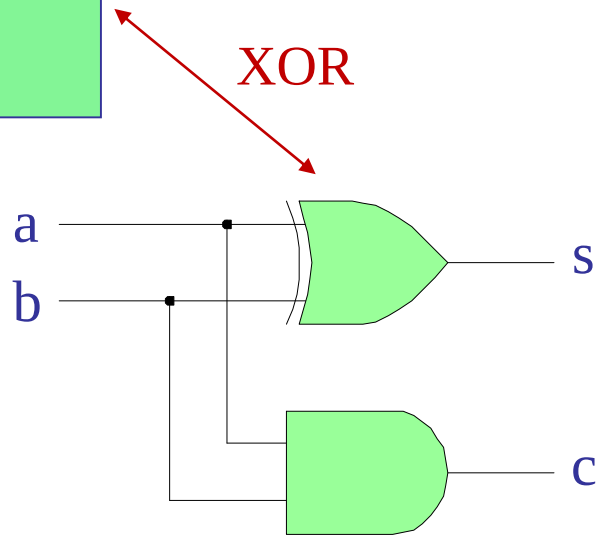
a	0	1	0	1
+ b	+ 0	+ 0	+ 1	+ 1
c s	0 0	0 1	0 1	1 0

truth table

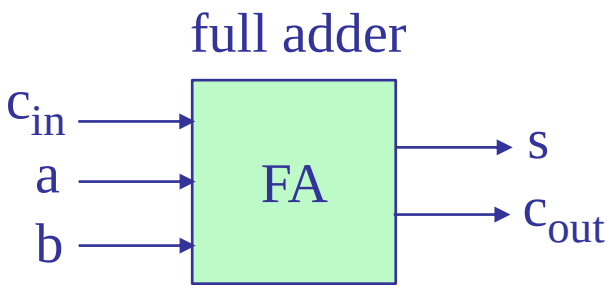
a b	sum s	carry c
0 0	0	0
0 1	1	0
1 0	1	0
1 1	0	1

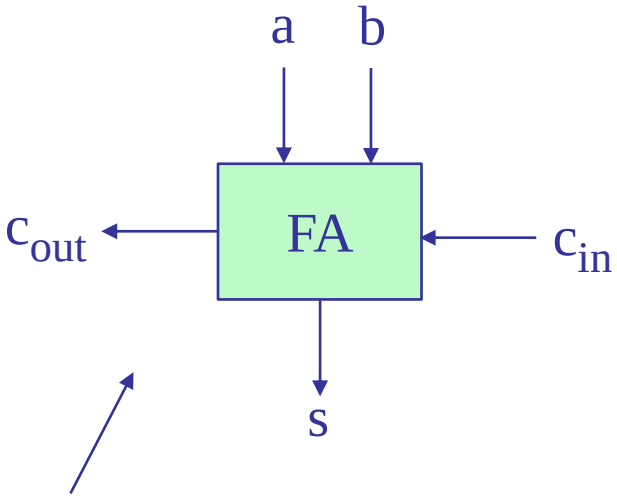
$s = a \cdot b' + a' \cdot b = a \oplus b$

$c = a \cdot b$



Addition and Subtraction – Full-Adders





equivalent symbol, suitable for cascading full adders for multibit addition

half-adder case

truth table

c_{in}	a	b	s	c_{out}
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

$$s = a \oplus b \oplus c_{in}$$

$$c_{out} = a \cdot b + (a + b) \cdot c_{in}$$

full-adder

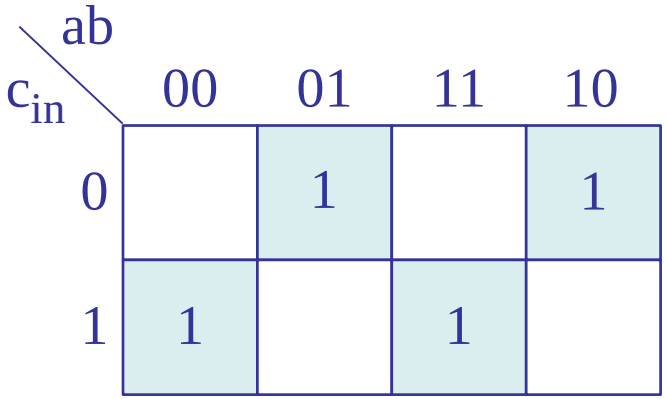
truth table

c_{in}	a	b	s	c_{out}
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

XOR properties

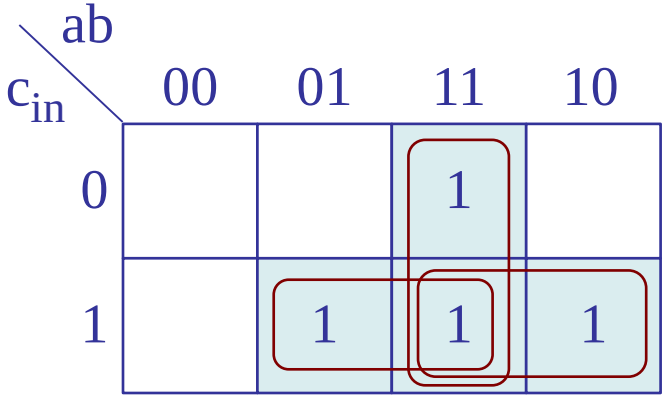
$0 \oplus X = X$

$1 \oplus X = X'$



K-map for s

$s = a \oplus b \oplus c_{in}$



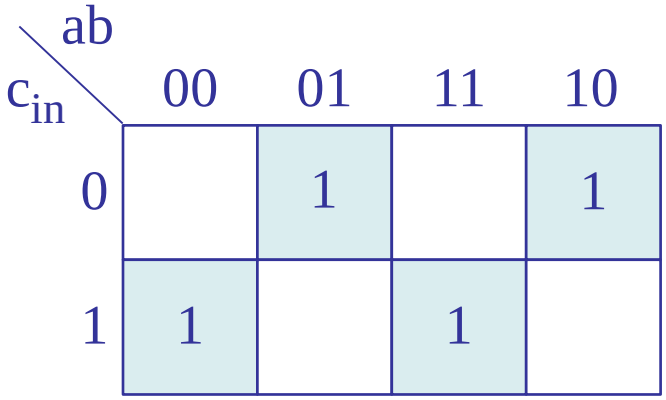
K-map for c_{out}

$c_{out} = a \cdot b + (a + b) \cdot c_{in}$

full-adder

truth table

c_{in}	a	b	s	c_{out}
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



K-map for s

$s = a \oplus b \oplus c_{in}$

$s = a' b' c_{in} + a b c_{in} + a' b c'_{in} + a b' c'_{in}$

$= (a' b' + a b)\, c_{in} + (a' b + a b')\, c'_{in}$

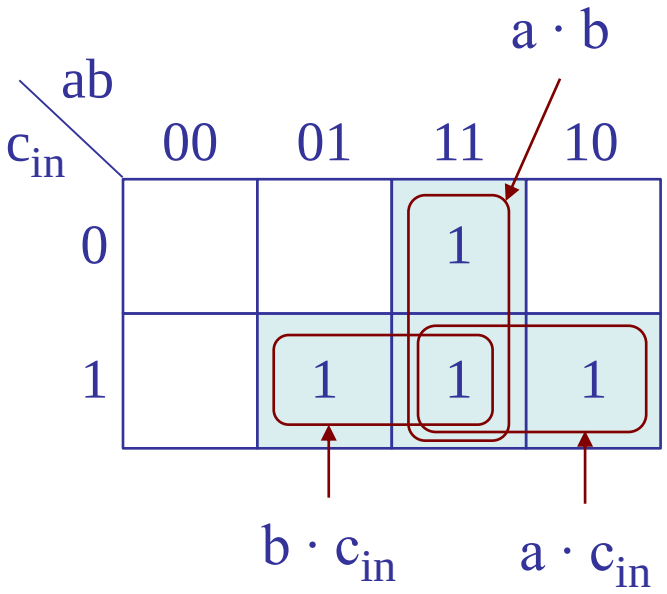
$= (a \oplus b)'\, c_{in} + (a \oplus b)\, c'_{in}$

$= (a \oplus b) \oplus c_{in}$

full-adder

truth table

c_{in}	a	b	s	c_{out}
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



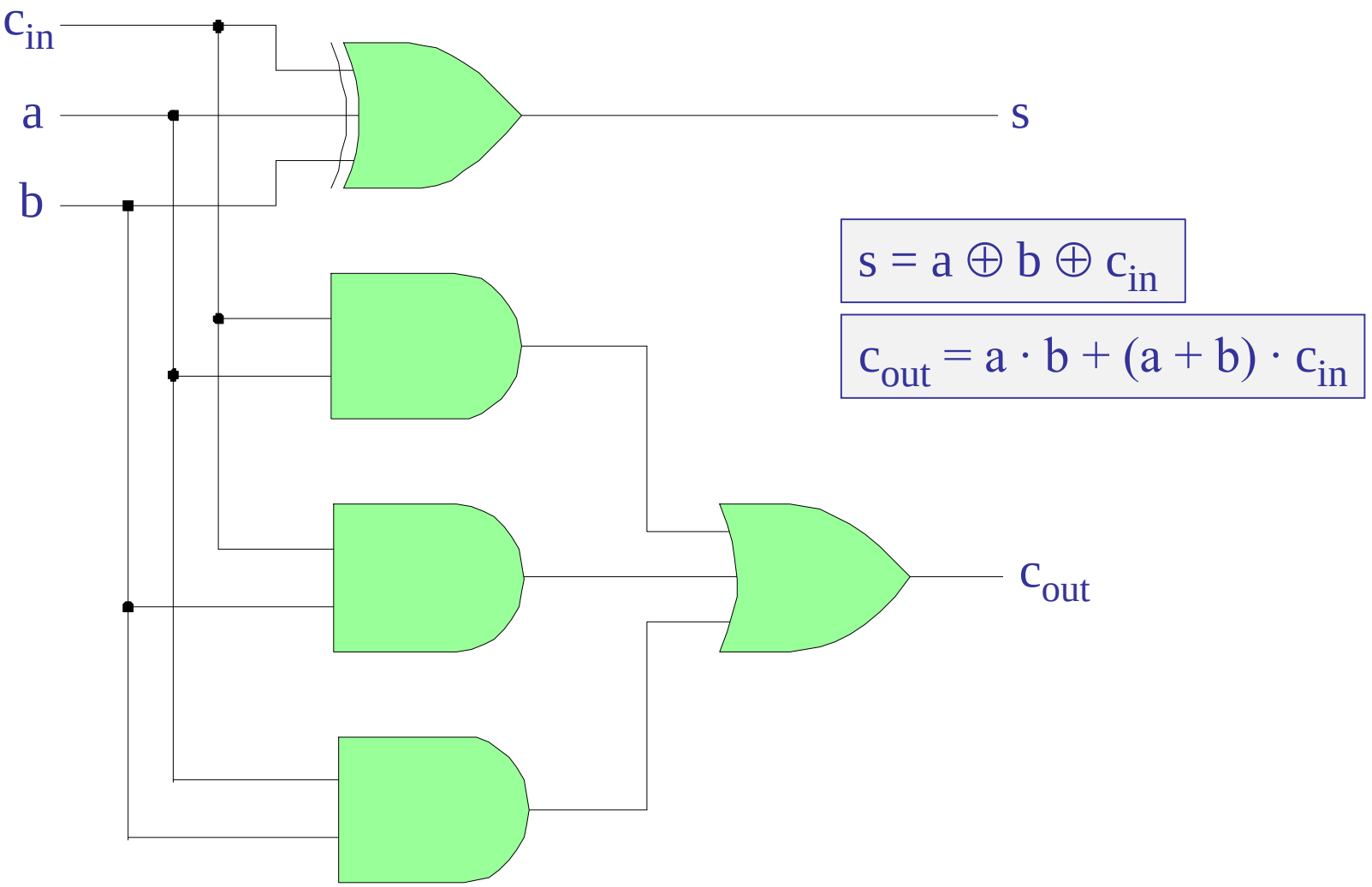
K-map for c_{out}

$$c_{out} = a \cdot b + (a + b) \cdot c_{in}$$

note also (looking at the K-map 0s,

$$c_{out}' = a' \cdot b' + (a' + b') \cdot c_{in}'$$

full-adder



$$s = a \oplus b \oplus c_{in}$$

$$c_{out} = a \cdot b + (a + b) \cdot c_{in}$$

full-adder implementation by cascading two half-adders

$$s = a \oplus b \oplus c_{in} = (a \oplus b) \oplus c_{in}$$

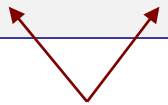
$$c_{out} = a \cdot b + a \cdot c_{in} + b \cdot c_{in} =$$

$$= a \cdot b + a \cdot (b + b') \cdot c_{in} + b \cdot (a + a') \cdot c_{in}$$

$$= a \cdot b + a \cdot b \cdot c_{in} + a \cdot b' \cdot c_{in} + a \cdot b \cdot c_{in} + a' \cdot b \cdot c_{in}$$

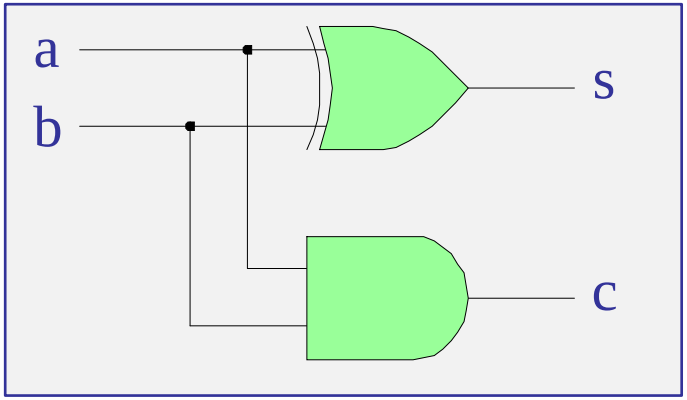
$$= a \cdot b \cdot (1 + c_{in} + c_{in}) + (a \cdot b' + a' \cdot b) \cdot c_{in}$$

$$= a \cdot b + (a \oplus b) \cdot c_{in}$$

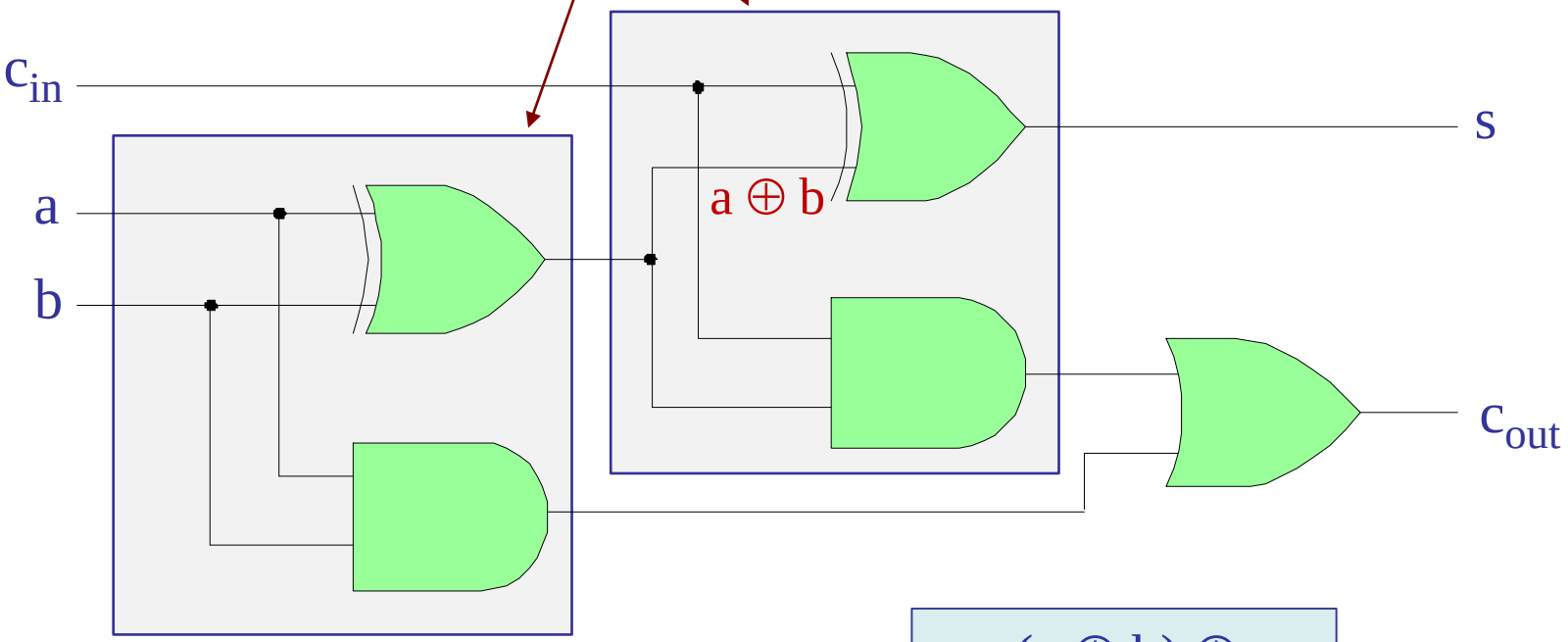


half-adder outputs

full-adder



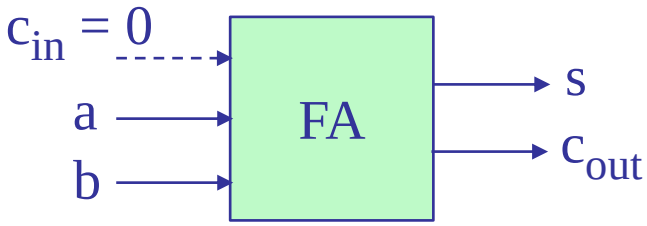
half-adders

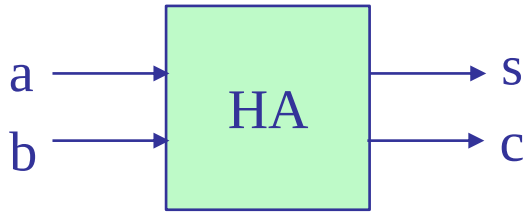


$$s = (a \oplus b) \oplus c_{in}$$

$$c_{out} = a \cdot b + (a \oplus b) \cdot c_{in}$$

half-adder as special case of full-adder





$c_{in} = 0$

$s = a \oplus b \oplus c_{in} = a \oplus b$

$c_{out} = a \cdot b + a \cdot c_{in} + b \cdot c_{in} = a \cdot b$

half-adder

truth table

c_{in}	a	b	s	c_{out}
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

XOR properties

$0 \oplus X = X$

$1 \oplus X = X'$

note also,

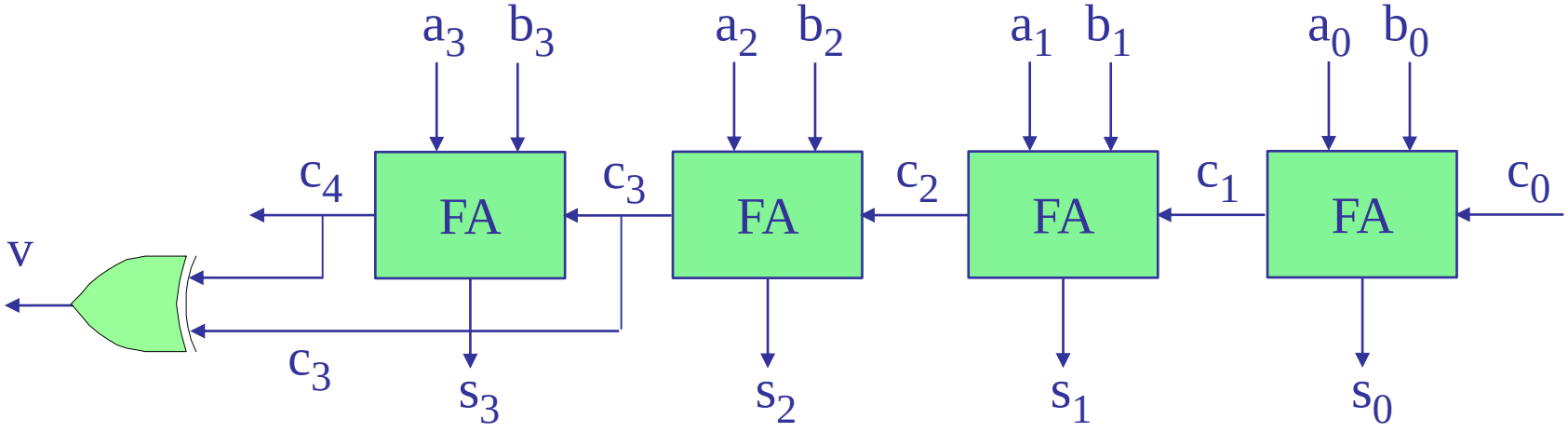
if, $c_{in} = 1$, then,

$s = a \oplus b \oplus c_{in} = (a \oplus b)'$

$c_{out} = a \cdot b + a + b = a + b$

ripple-carry full-adder for 2's complement integers

addition, a+b



$c_4\ c_3\ c_2\ c_1\ c_0$	carries
$a_3\ a_2\ a_1\ a_0$ $b_3\ b_2\ b_1\ b_0$	addend addend
$s_3\ s_2\ s_1\ s_0$	sum
$v = c_4 \oplus c_3$	overflow bit, $v = c_N \oplus c_{N-1}$, for N-bit case

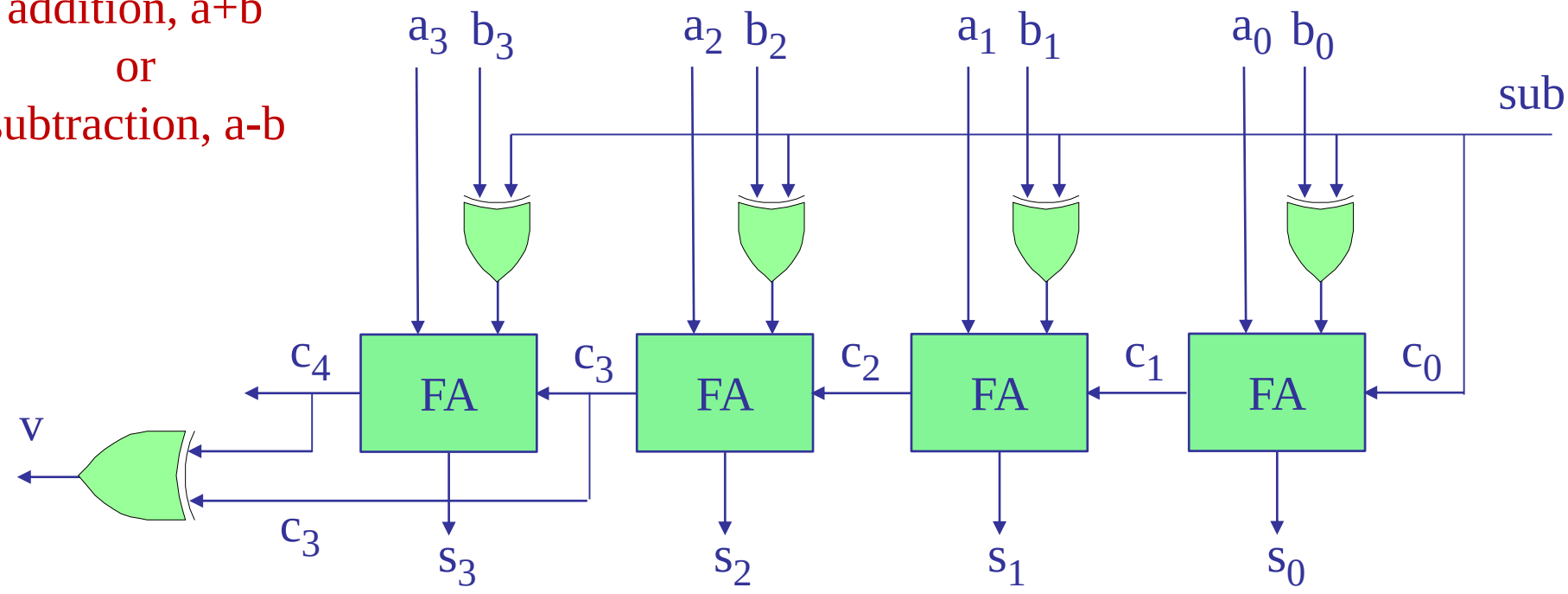
for i = 0,1,2,3, do,

$s_i = a_i \oplus b_i \oplus c_i$

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

ripple-carry full-adder/subtractor for 2's complement integers

addition, a+b
or
subtraction, a-b



recall the properties,

$X \oplus 0 = X$

$X \oplus 1 = X'$

$a - b = a + (-b) = a + (b)_{2c} = a + (b)' + 1$

sub = 0, addition
sub = 1, subtraction

overflow rules for 2's complement addition/subtraction:

1. Range: $[-2^{N-1},\ 2^{N-1}-1]$, for N-bit 2's complement integers.

2. Overflow bit is computed by, $v = c_N \oplus c_{N-1}$

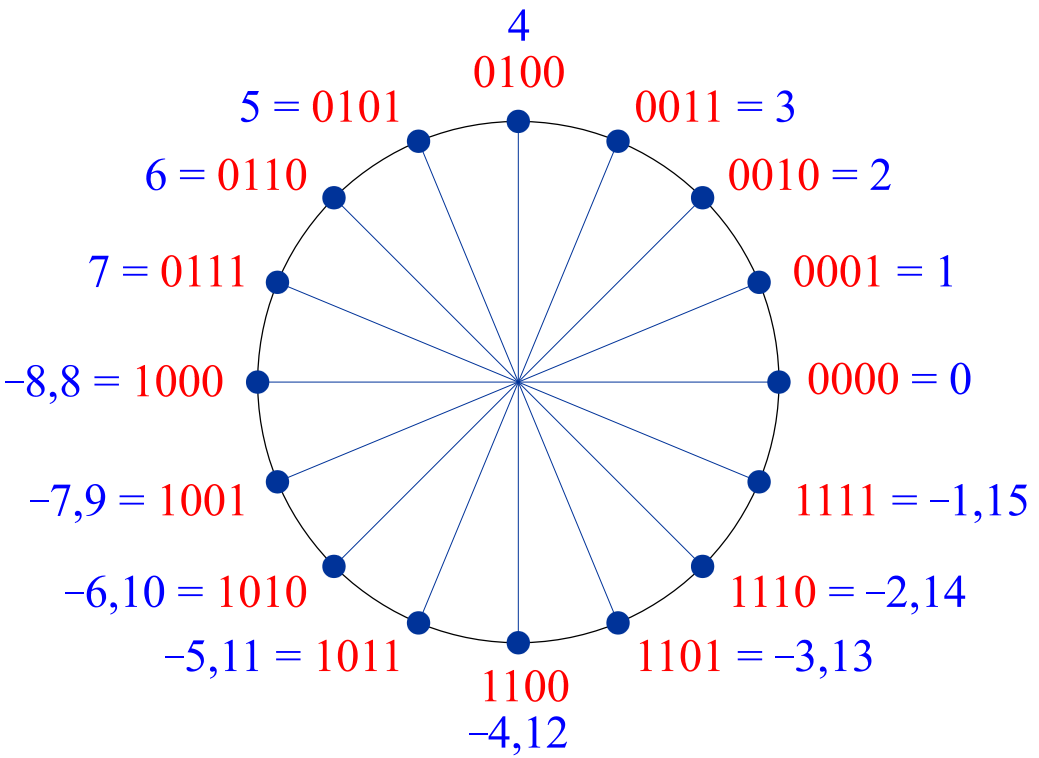
3. If the numbers a, b are both positive or both negative, then, there is no overflow (v=0) for the difference, a - b, and the output carry c_N can be ignored (this also the case if one of the numbers a, b is positive and the other negative and they are added.)

4. If the numbers a, b are both positive or both negative, then there will be overflow in the sum, s = a + b, if s lies beyond the range, $[-2^{N-1},\ 2^{N-1}-1]$, and in this case the overflow bit is v=1, and the output carry c_N (whether it is 0 or 1) must be used to extend the result to N+1 bits, i.e., c_N mut be prepended to become the MSB of the resulting sum s.

see some examples in next few pages

x	mod(x,2^4)	bits
7	7	0 1 1 1
6	6	0 1 1 0
5	5	0 1 0 1
4	4	0 1 0 0
3	3	0 0 1 1
2	2	0 0 1 0
1	1	0 0 0 1
0	0	0 0 0 0
-1	15	1 1 1 1
-2	14	1 1 1 0
-3	13	1 1 0 1
-4	12	1 1 0 0
-5	11	1 0 1 1
-6	10	1 0 1 0
-7	9	1 0 0 1
-8	8	1 0 0 0

we recall that 2's complement negative 4-bit integers have the same bit pattern as their mod-16 positive counterparts



$$x = -b_3 2^3 + b_2 2^2 + b_1 2^1 + b_0$$

for N=4 case, range: $[-2^3, 2^3-1] = [-8, 7]$

if both numbers a,b are positive or both negative, then, there is no overflow for the difference, a-b, but there may be for their sum, a+b.

both a,b positive or zero,

$0 \le a \le 7$ $\quad\Rightarrow\quad$ $0 \le a \le 7$ $\quad\Rightarrow\quad$ $-7 \le a - b \le 7$

$0 \le b \le 7$ $\qquad$ $-7 \le -b \le 0$

but,

$0 \le a + b \le 14$, and sum requires possibly 5 bits

both a,b negative,

$-8 \le a \le -1$ $\quad\Rightarrow\quad$ $-8 \le a \le -1$ $\quad\Rightarrow\quad$ $-7 \le a - b \le 7$

$-8 \le b \le -1$ $\qquad$ $1 \le -b \le 8$

but,

$-16 \le a + b \le -2$, and sum requires possibly 5 bits

5-bit range:
$[-2^4, 2^4-1] = [-16,15]$

for N=4 case, range: $[-2^3, 2^3-1] = [-8, 7]$

if one of the numbers a,b is positive and the other negative, then, there is no overflow for the sum, a+b, but there may be in their difference, a-b.

for example,

$$0 \le a \le 7 \quad \Rightarrow \quad 0 \le a \le 7 \quad \Rightarrow \quad 1 \le a - b \le 15, \text{ and can overflow}$$
$$-8 \le b \le -1 \qquad 1 \le -b \le 8$$

but,

$-8 \le a + b \le 6$, and remains within the 4-bit range

$v = c_4 \oplus c_3$

4-bit range [-8,7]

$0 \oplus X = X$

$1 \oplus X = X'$

$c_4\ c_3\ c_2\ c_1\ c_0$

```
carries:   0  0  0  0  0

     5  =    0  1  0  1
     2  =    0  0  1  0
   ------------------------
     7  =    0  1  1  1
```

v = 0 ⊕ 0 = 0

c_4 = 0, ignored

```
carries:   1  1  1  0  0

    -5  =    1  0  1  1
    -2  =    1  1  1  0
   ------------------------
    -7  =    1  0  0  1
```

v = 1 ⊕ 1 = 0

c_4 = 1, ignored

```
carries:   1  1  0  0  0

     5  =    0  1  0  1
    -2  =    1  1  1  0
   ------------------------
     3  =    0  0  1  1
```

v = 1 ⊕ 1 = 0

c_4 = 1, ignored

```
carries:   0  0  1  0  0

    -5  =    1  0  1  1
     2  =    0  0  1  0
   ------------------------
    -3  =    1  1  0  1
```

v = 0 ⊕ 0 = 0

c_4 = 0, ignored

```
carries:  0 1 1 0 0

      6  =   0 1 1 0
      3  =   0 0 1 1
    ---------------
      9  = 0 1 0 0 1
   (5-bit)
```

$v = 0 \oplus 1 = 1$

$c_4 = 0$, included

```
carries:  1 0 0 0 0

     -6  =   1 0 1 0
     -3  =   1 1 0 1
    ---------------
     -9  = 1 0 1 1 1
   (5-bit)
```

$v = 1 \oplus 0 = 1$

$c_4 = 1$, included

9 = -7 + 16
7 = -9 + 16
23 = 32-9

$v = c_4 \oplus c_3$

5-bit range [-16,15]

4-bit range [-8,7]

```
carries:  1 1 0 0 0

      6  =   0 1 1 0
     -3  =   1 1 0 1
    ---------------
      3  =   0 0 1 1
```

$v = 1 \oplus 1 = 0$

$c_4 = 1$, ignored

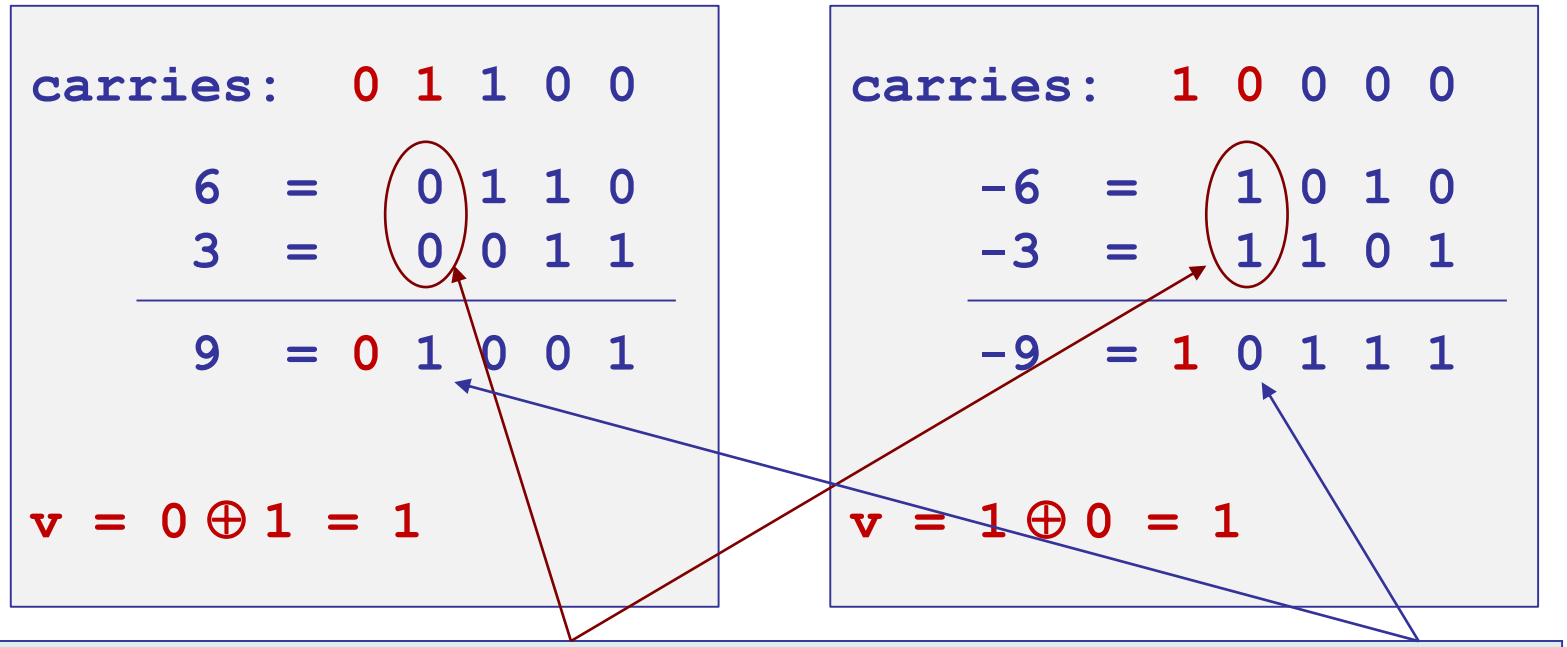
```
carries:  0 0 1 0 0

     -6  =   1 0 1 0
      3  =   0 0 1 1
    ---------------
     -3  =   1 1 0 1
```

$v = 0 \oplus 0 = 0$

$c_4 = 0$, ignored

$0 \oplus X = X$

$1 \oplus X = X'$



overflow occurs when, a_3,b_3, (or rather, a, b) have the same sign, but the sum, s_3, has the opposite sign, thus, we have alternatively,

$$\mathbf{v = a_3\, b_3\, s_3' + a_3'\, b_3'\, s_3 = a_3\, b_3\, c_3' + a_3'\, b_3'\, c_3 = c_4 \oplus c_3}$$

exercise exercise

or, more generally, $v = a_{N-1}\, b_{N-1}\, s_{N-1}' + a_{N-1}'\, b_{N-1}'\, s_{N-1}$

$= a_{N-1}\, b_{N-1}\, c_{N-1}' + a_{N-1}'\, b_{N-1}'\, c_{N-1} = c_N \oplus c_{N-1}$

overflow occurs when, a_3,b_3, (or rather, a, b) have the same sign, but the sum, s_3, has the opposite sign, thus, we have alternatively,

$$\mathbf{v = a_3\, b_3\, s_3' + a_3'\, b_3'\, s_3 = a_3\, b_3\, c_3' + a_3'\, b_3'\, c_3 = c_4 \oplus c_3}$$

exercise exercise

to prove these, use the relationships:

$$\mathbf{s_3 = a_3 \oplus b_3 \oplus c_3}$$

$$\mathbf{c_4 = a_3 \cdot b_3 + (a_3 + b_3) \cdot c_3}$$

```
 10 =     0 1 0 1 0
  8 =     0 1 0 0 0
 ---------------------
 18 =   0 1 0 0 1 0

 v,c5 = 1,0
```

extended

```
 -10 =     1 0 1 1 0
  -8 =     1 1 0 0 0
 ---------------------
 -18 =   1 0 1 1 1 0

v,c5 = 1,1
```

extended

```
 -10 =     1 0 1 1 0
   8 =     0 1 0 0 0
 ---------------------
  -2 =     1 1 1 1 0

v,c5 = 0,0
```

not extended

some additional 5-bit examples from
f22-e2-practice.pdf

in each case, please work out all the carry bits

$c_5\ c_4\ c_3\ c_2\ c_1\ c_0$

and the overflow bit

$v = c_5 \oplus c_4$

5-bit range	6-bit range
[-16,15]	[-32,31]

5-bit

$$x = -b_4 2^4 + b_3 2^3 + b_2 2^2 + b_1 2^1 + b_0$$

6-bit

$$x = -b_5 2^5 + b_4 2^4 + b_3 2^3 + b_2 2^2 + b_1 2^1 + b_0$$

more 5-bit practice examples

```
carries:  0 1 1 1 1 0
-------------------------
   11 =     0 1 0 1 1
   15 =     0 1 1 1 1
-------------------------
   26 =   0 1 1 0 1 0
          ↑
v = 1     extended to 6 bits
```

$v = c_5 \oplus c_4$

5-bit range [-16,15]

6-bit range [-32,31]

```
carries:  1 0 0 0 1 0
-------------------------
  -11 =     1 0 1 0 1
  -15 =     1 0 0 0 1
-------------------------
  -26 =   1 0 0 1 1 0
          ↑
v = 1     extended to 6 bits
```

```
carries:  1 1 1 1 1 0
-------------------------
  -11 =     1 0 1 0 1
   15 =     0 1 1 1 1
-------------------------
    4 =     0 0 1 0 0
            ↑
v = 0       not extended
```

$$x = -b_5 2^5 + b_4 2^4 + b_3 2^3 + b_2 2^2 + b_1 2^1 + b_0$$

x	5-bit
15	0 1 1 1 1
14	0 1 1 1 0
13	0 1 1 0 1
12	0 1 1 0 0
11	0 1 0 1 1
10	0 1 0 1 0
9	0 1 0 0 1
8	0 1 0 0 0
7	0 0 1 1 1
6	0 0 1 1 0
5	0 0 1 0 1
4	0 0 1 0 0
3	0 0 0 1 1
2	0 0 0 1 0
1	0 0 0 0 1
0	0 0 0 0 0
-1	1 1 1 1 1
-2	1 1 1 1 0
-3	1 1 1 0 1
-4	1 1 1 0 0
-5	1 1 0 1 1
-6	1 1 0 1 0
-7	1 1 0 0 1
-8	1 1 0 0 0
-9	1 0 1 1 1
-10	1 0 1 1 0
-11	1 0 1 0 1
-12	1 0 1 0 0
-13	1 0 0 1 1
-14	1 0 0 1 0
-15	1 0 0 0 1
-16	1 0 0 0 0

x	6- bit	x	6-bit
31	0 1 1 1 1 1	-1	1 1 1 1 1 1
30	0 1 1 1 1 0	-2	1 1 1 1 1 0
29	0 1 1 1 0 1	-3	1 1 1 1 0 1
28	0 1 1 1 0 0	-4	1 1 1 1 0 0
27	0 1 1 0 1 1	-5	1 1 1 0 1 1
26	0 1 1 0 1 0	-6	1 1 1 0 1 0
25	0 1 1 0 0 1	-7	1 1 1 0 0 1
24	0 1 1 0 0 0	-8	1 1 1 0 0 0
23	0 1 0 1 1 1	-9	1 1 0 1 1 1
22	0 1 0 1 1 0	-10	1 1 0 1 1 0
21	0 1 0 1 0 1	-11	1 1 0 1 0 1
20	0 1 0 1 0 0	-12	1 1 0 1 0 0
19	0 1 0 0 1 1	-13	1 1 0 0 1 1
18	0 1 0 0 1 0	-14	1 1 0 0 1 0
17	0 1 0 0 0 1	-15	1 1 0 0 0 1
16	0 1 0 0 0 0	-16	1 1 0 0 0 0
15	0 0 1 1 1 1	-17	1 0 1 1 1 1
14	0 0 1 1 1 0	-18	1 0 1 1 1 0
13	0 0 1 1 0 1	-19	1 0 1 1 0 1
12	0 0 1 1 0 0	-20	1 0 1 1 0 0
11	0 0 1 0 1 1	-21	1 0 1 0 1 1
10	0 0 1 0 1 0	-22	1 0 1 0 1 0
9	0 0 1 0 0 1	-23	1 0 1 0 0 1
8	0 0 1 0 0 0	-24	1 0 1 0 0 0
7	0 0 0 1 1 1	-25	1 0 0 1 1 1
6	0 0 0 1 1 0	-26	1 0 0 1 1 0
5	0 0 0 1 0 1	-27	1 0 0 1 0 1
4	0 0 0 1 0 0	-28	1 0 0 1 0 0
3	0 0 0 0 1 1	-29	1 0 0 0 1 1
2	0 0 0 0 1 0	-30	1 0 0 0 1 0
1	0 0 0 0 0 1	-31	1 0 0 0 0 1
0	0 0 0 0 0 0	-32	1 0 0 0 0 0

5-bit and 6-bit two's complement refererence tables

ranges:

5-bit: [–16, 15]

6-bit: [–32, 31]

carry-lookahead adder

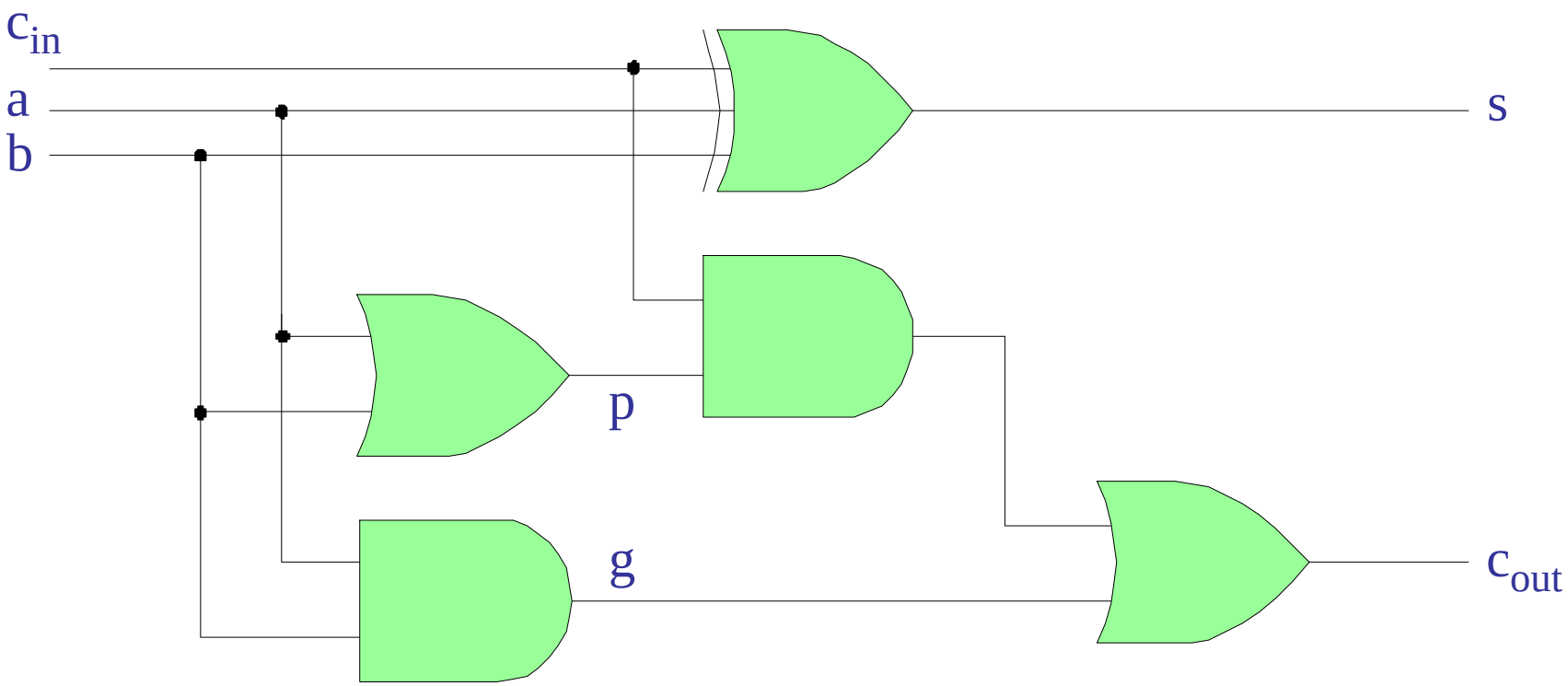
$s = a \oplus b \oplus c_{in}$

$g = a \cdot b$ = carry-generate

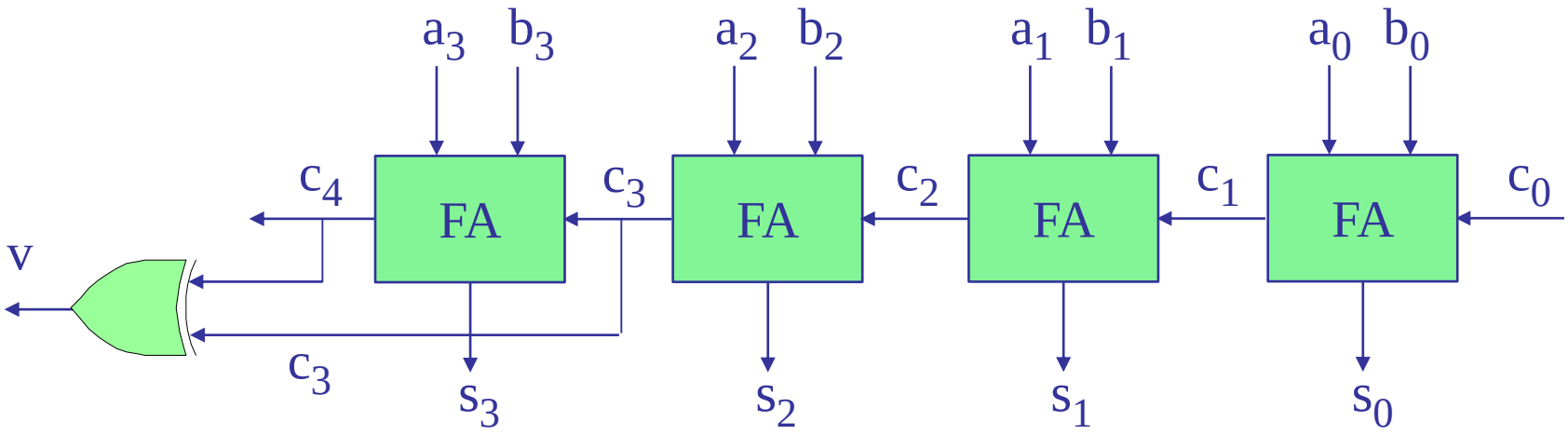
$p = a + b$ = carry-propagate

$c_{out} = a \cdot b + (a + b) \cdot c_{in}$

$c_{out} = g + p \cdot c_{in}$



carry-lookahead adder



$c_1 = g_0 + p_0 \cdot c_0$ $\quad (g_0 = a_0 \cdot b_0,\ \ p_0 = a_0 + b_0)$

$c_2 = g_1 + p_1 \cdot c_1$ $\quad (g_1 = a_1 \cdot b_1,\ \ p_1 = a_1 + b_1)$

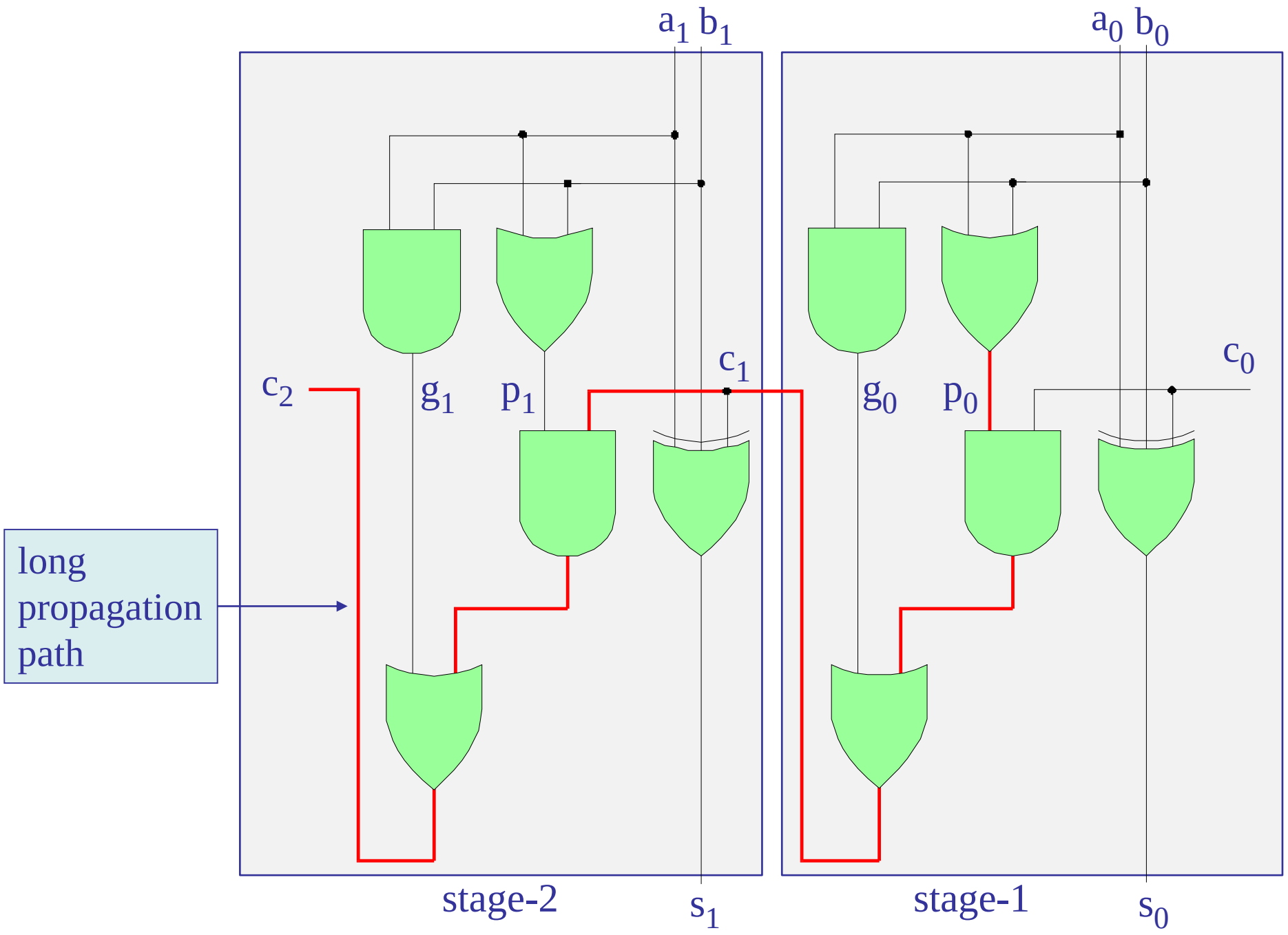
$= g_1 + p_1 \cdot g_0 + p_1 \cdot p_0 \cdot c_0$

$c_3 = g_2 + p_2 \cdot c_2$

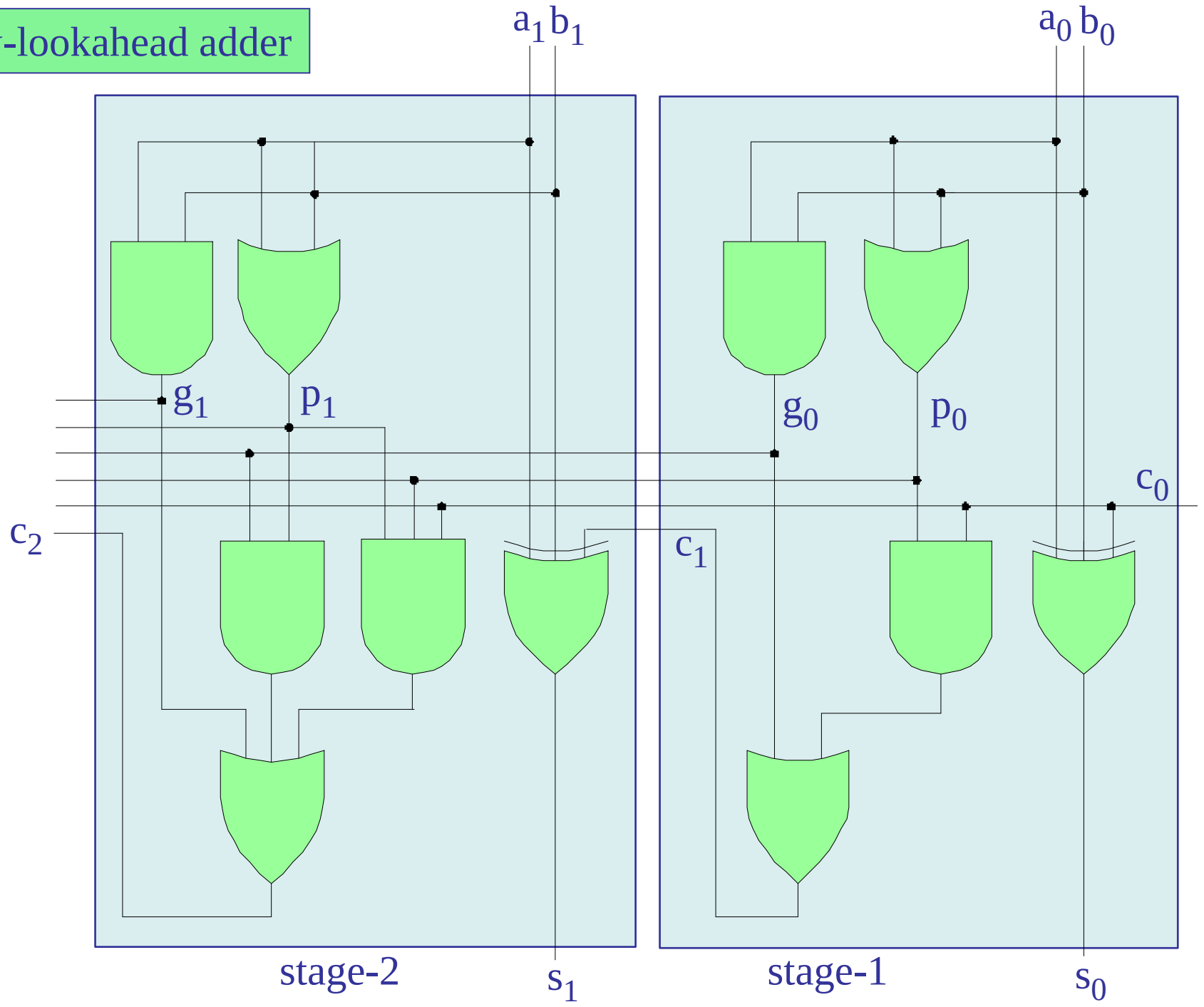
$= g_2 + p_2 \cdot g_1 + p_2 \cdot p_1 \cdot g_0 + p_2 \cdot p_1 \cdot p_0 \cdot c_0$

$c_4 = g_3 + p_3 \cdot c_3$

$= g_3 + p_3 \cdot g_2 + p_3 \cdot p_2 \cdot g_1 + p_3 \cdot p_2 \cdot p_1 \cdot g_0 + p_3 \cdot p_2 \cdot p_1 \cdot p_0 \cdot c_0$

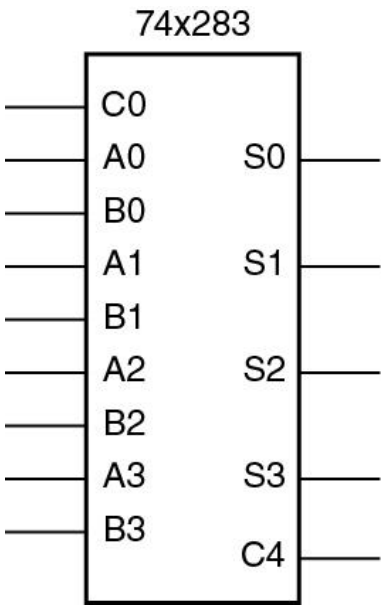

a1 b1
a0 b0
c2
g1
p1
c1
g0
p0
c0
long propagation path
stage-2
s1
stage-1
s0

carry-lookahead adder



carry-lookahead adder

Fig. 8-6. Logic Diagram for the 74×283 4-Bit Binary Adder with Internal Carry Lookahead



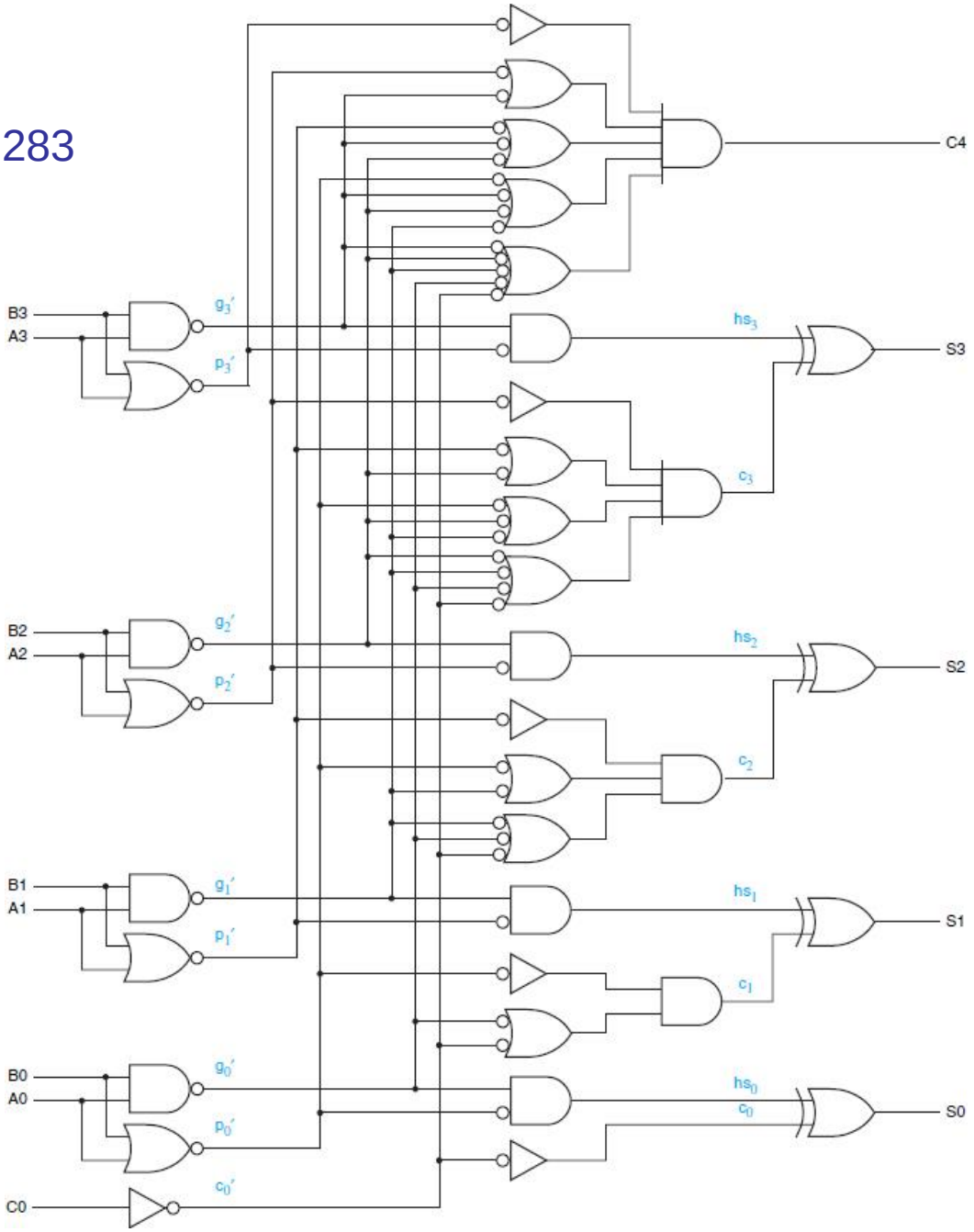


Fig. 8-22. Partial Products for an 8 × 8 Multiplier

$x_7\ x_6\ x_5\ x_4\ x_3\ x_2\ x_1\ x_0$
$y_7\ y_6\ y_5\ y_4\ y_3\ y_2\ y_1\ y_0$

partial products can be realized by AND gates

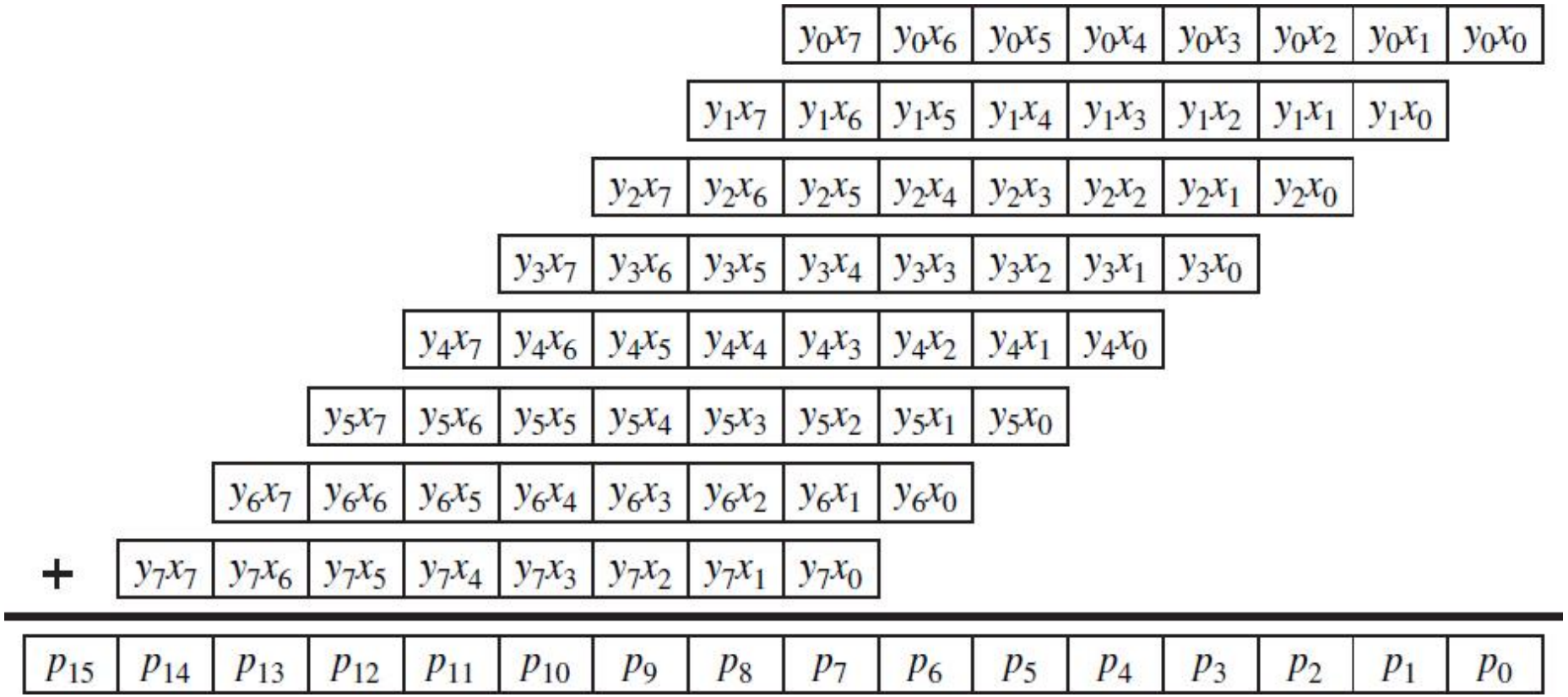


Fig. 8-23. Interconnections for an 8 × 8 Combinational Multiplier

+ signs indicate full-adders