ADDERS

Circuits for Binary Addition

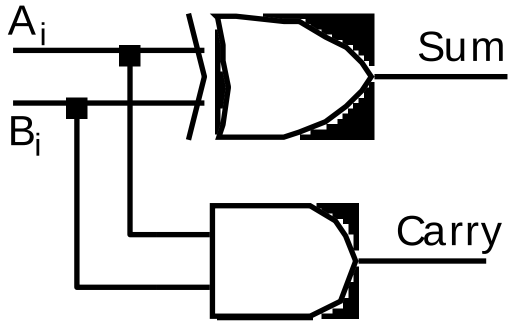
Half Adder

Ai	Bi	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$\begin{array}{c|cc} {}_{Bi}\backslash^{Ai} & 0 & 1 \\ \hline 0 & 0 & 1 \\ 1 & 1 & 0 \end{array} \qquad \begin{array}{c|cc} {}_{Bi}\backslash^{Ai} & 0 & 1 \\ \hline 0 & 0 & 0 \\ 1 & 0 & 1 \end{array}$$

$$\text{Sum} = \overline{Ai}\,Bi + Ai\,\overline{Bi}$$
$$= Ai \oplus Bi$$

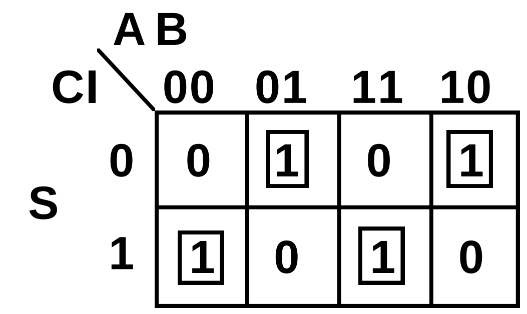
$$\text{Carry} = Ai\,Bi$$



Half-adder Schematic

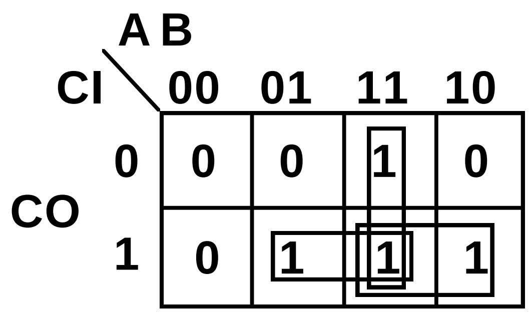
Full Adder

A	B	CI	S	CO
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

CI \ A B	00	01	11	10
0	0	1	0	1
1	1	0	1	0



CO

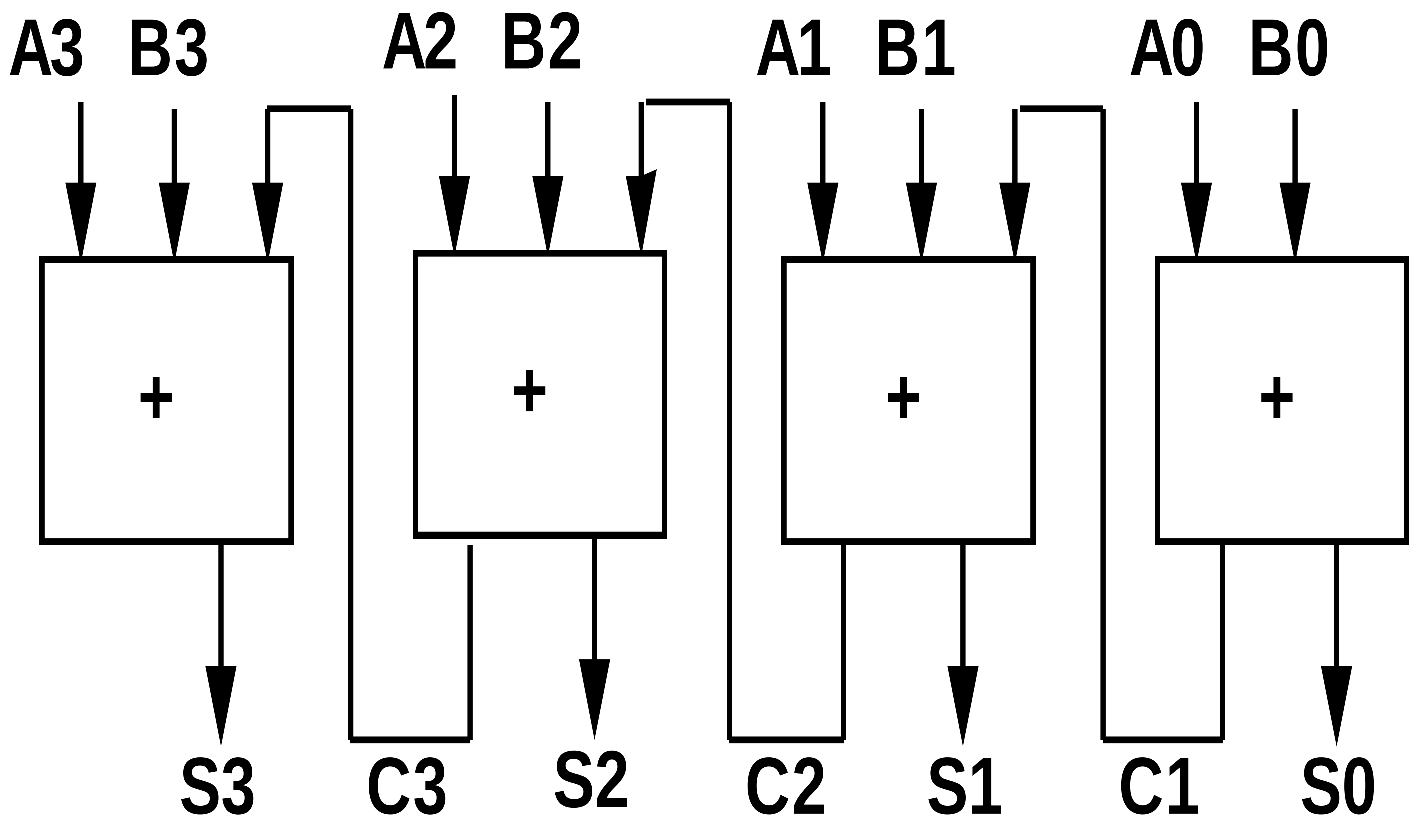
CI \ A B	00	01	11	10
0	0	0	1	0
1	0	1	1	1

S = CI xor A xor B

CO = B CI + A CI + A B = CI (A + B) + A B

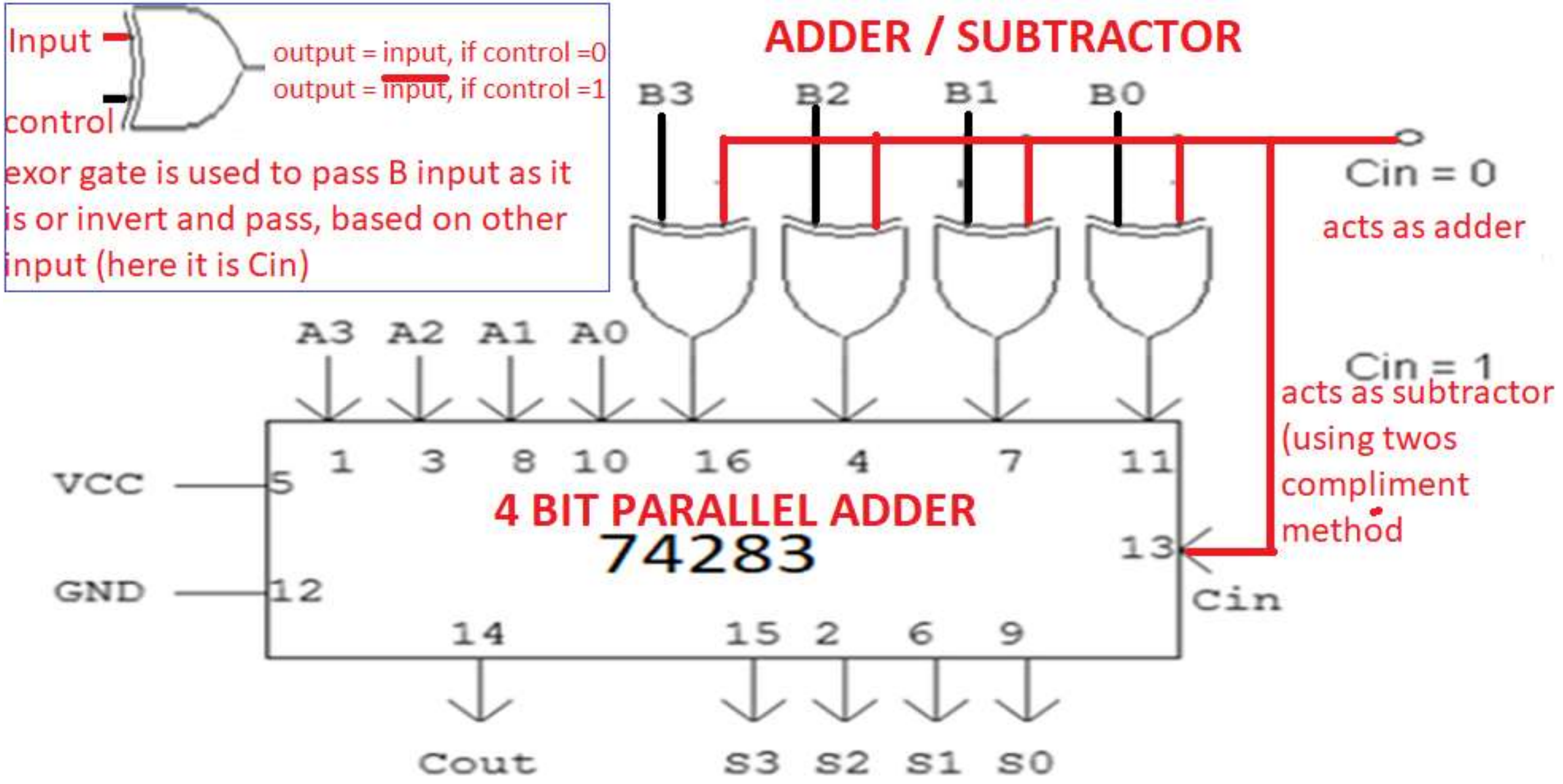
Cascaded 4 BIT
Parallel Adders / Ripple Adder

Parallel adders are digital circuits that compute the addition of variable binary strings of equivalent or different size in parallel. The schematic diagram of a parallel adder is shown, to add two 4 bit binary numbers. The circuit is laid out from right to left, similar to the way we add binary numbers. Therefore, the least significant column is on the right, and the most significant column is on the left. The boxes labeled FA/+ are full adders. The Carry out from each full adder is the Carry-in to the next higher full adder.



4 BIT Adder / Subtractor

The numbers being processed are A4 A3 A2 A1and B4 B3 B2 B1, while the answer is S4 S3 S2 S1. 4 bit adder is built as described in the previous slide, using 4 numbers of single bit full adders. Here ExOR gate is used, control the circuit to work as an 4 bit adder or 4 bit subtractor.


Input
control
output = input, if control =0
output = input, if control =1
exor gate is used to pass B input as it is or invert and pass, based on other input (here it is Cin)
ADDER / SUBTRACTOR
B3
B2
B1
B0
Cin = 0
acts as adder
Cin = 1
acts as subtractor (using twos compliment method
A3
A2
A1
A0
1
3
8
10
16
4
7
11
VCC
5
GND
12
4 BIT PARALLEL ADDER
74283
13
Cin
14
15
2
6
9
Cout
S3
S2
S1
S0

4 BIT Adder / Subtractor Examples:

Perform addition of following numbers, given in binary form

```
 1000
 1100
-------
10100
```

Perform addition of following numbers, given in decimal number system.

```
15   -  1111
+9   -   1001
----    -----------
24    1  1000
```

Perform addition of following numbers given in hexadecimal number system

```
A         1010
9        1001
--      ---------
1 3      1 0011
```

Note: for subtraction, take the 2s compliment of second number and add to the first number. Note, the carry will be the opposite of the actual value

Application of Adder: used for Excess-3 to BCD Converter, vice versa

Binary Coded Decimal: Binary Coded Decimal is a method of using binary digits to represent the decimal digits 0 through 9. It is possible to assign weights to the binary bits according to their positions. The weights in the BCD code are 8, 4, 2 and 1. Ex: $(137)_{10}$ - BCD equivalent $(0001\ 0011\ 0111)_{2.}$

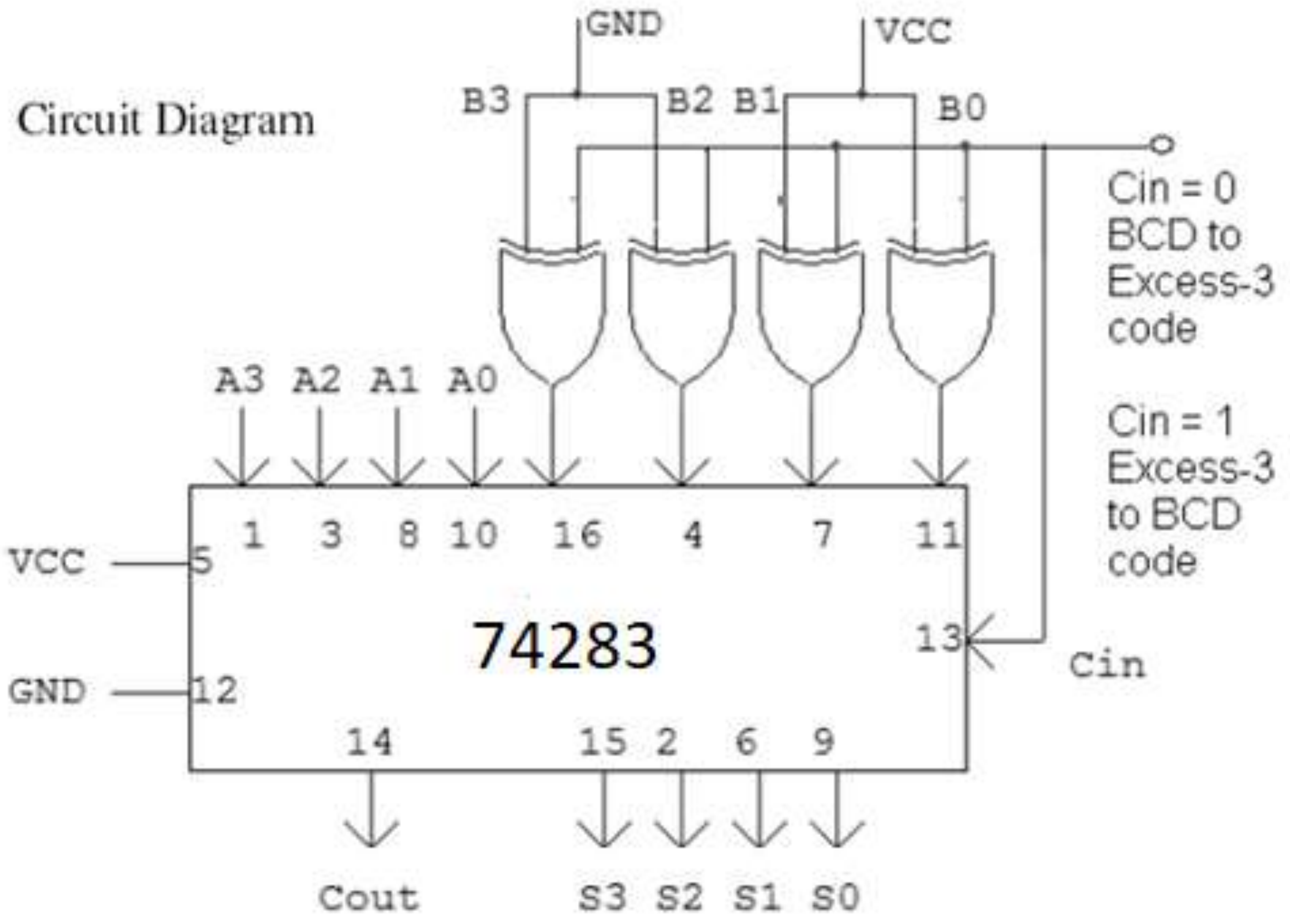
Excess-3 Code: This is an un-weighted code. Its code assignment is obtained from the corresponding value of BCD after the addition of $(0011)_2$.

BCD to Excess-3 (or) Excess-3 to BCD: Since each code uses four bits to represent a decimal digit, there must be four inputs and four output variables. The input variable are designated as B3, B2, B1, B0 and the output variables are designated as E3 , E2, E1, E0 in the truth table.

BCD				EXCESS-3			
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

EXCESS-3				BCD			
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	1
0	1	0	1	0	0	1	0
0	1	1	0	0	0	1	1
0	1	1	1	0	1	0	0
1	0	0	0	0	1	0	1
1	0	0	1	0	1	1	0
1	0	1	0	0	1	1	1
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	1

Application of Adder: used for Excess-3 to BCD Converter, vice versa



During the BCD to Excess 3 conversion: 0011 is applied to B3 B2 B1 B0 INPUTS of 4 bit adder (GND-0,VCC-1).

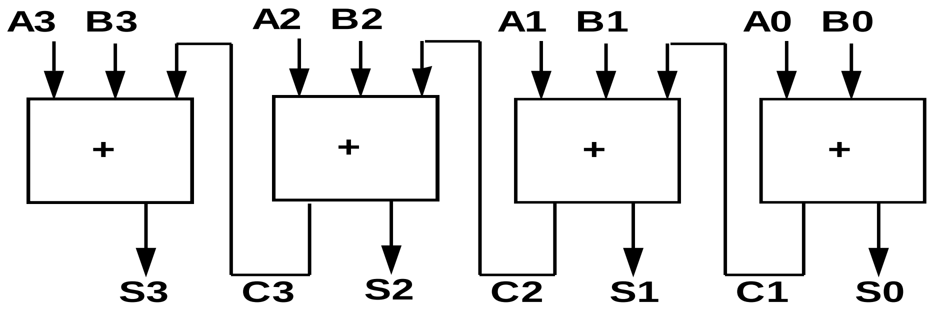
During Excess 3 to BCD conversion, 3 (0011) is subtracted from the number, using twos compliment method.

Delay Analysis of Ripple Adder

- Carry out of a single stage can be implemented in 2 gate delays (Units of Time)
- For a 16 bit adder, the 16th bit carry is generated after 16 * 2 = 32 gate delays.

  Total Time Required for N bit adder using Ripple parallel adder = N * 2
- Takes too long - need to investigate FASTER adders!
- Carry Look Ahead Adders, CLA adders, overcomes the disadvantage of taking more units of time. The principle behind design of CLA adders, can be understood by looking at the equation for carry generated at the ith stage, Ci+1, for full adder

Ci+1 = Xi Yi + Xi Ci + Yi Ci



Let us write S and Cout expressions for full adder, in terms of inputs X and Y and Cin, for 'i' th bit

Si = Xi .xor. Yi .xor. Ci

Ci+1 = Xi Yi + Xi Ci + Yi Ci

= Xi Yi + Ci (Xi + Yi)

= Xi Yi + Ci (Xi .or. Yi)

Let us represent, XiYi by Gi (also called as Carry generate function) &
Xi.or.Yi by Pi (also called carry propagate function)

= Gi + Ci Pi

Now, lets write the these expressions, for C1,C2,C3 and C4

C1 = G0 + P0 C0

C2 = G1 + P1 C1 = G1 + P1 G0 + P1 P0 C0

C3 = G2 + P2 C2 = G2 + P2 G1 + P2 P1 G0 + P2 P1 P0 C0

C4 = G3 + P3 C3 = G3 + P3 G2 + P3 P2 G1 + P3 P2 P1 G0 + P3 P2 P1 P0 C0

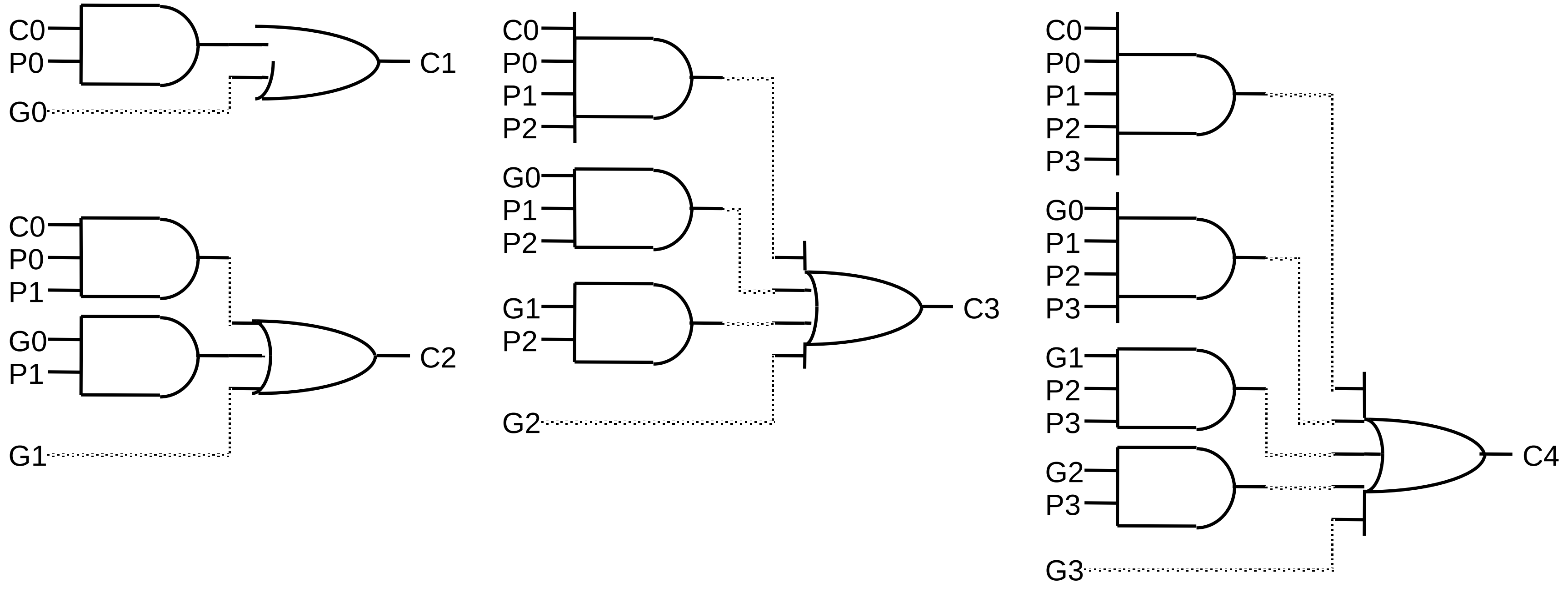
Important: Carry input at each stage is direct function of operand bits. Thus the idea of having SUM and CARRY outputs of the ith stage be a function of x_i,y_i and c_i is replaced by having these outputs be a function of x_0,x_1,…,x_i, y_0,y_1,…,y_i and c_0

C1 = G0 + P0 C0

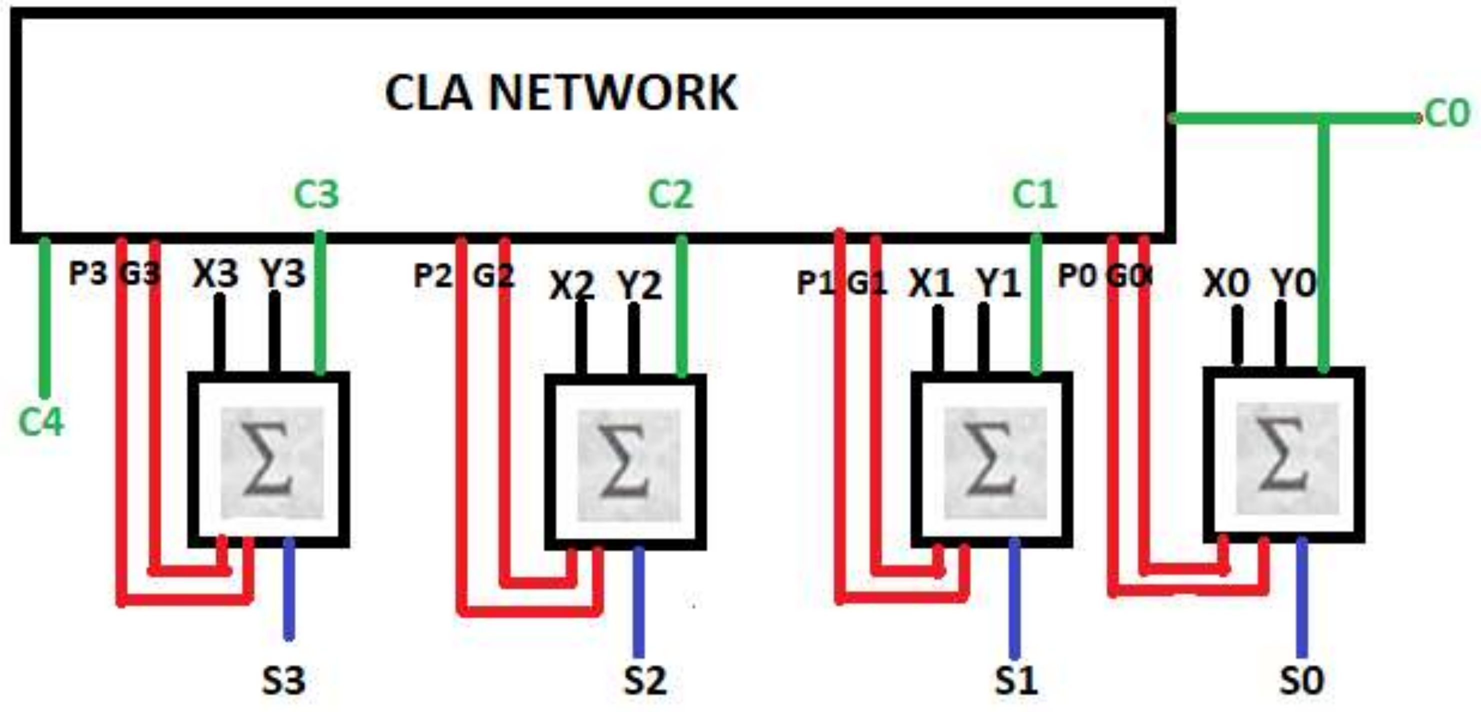
C2 = G1 + P1 C1 = G1 + P1 G0 + P1 P0 C0

C3 = G2 + P2 C2 = G2 + P2 G1 + P2 P1 G0 + P2 P1 P0 C0

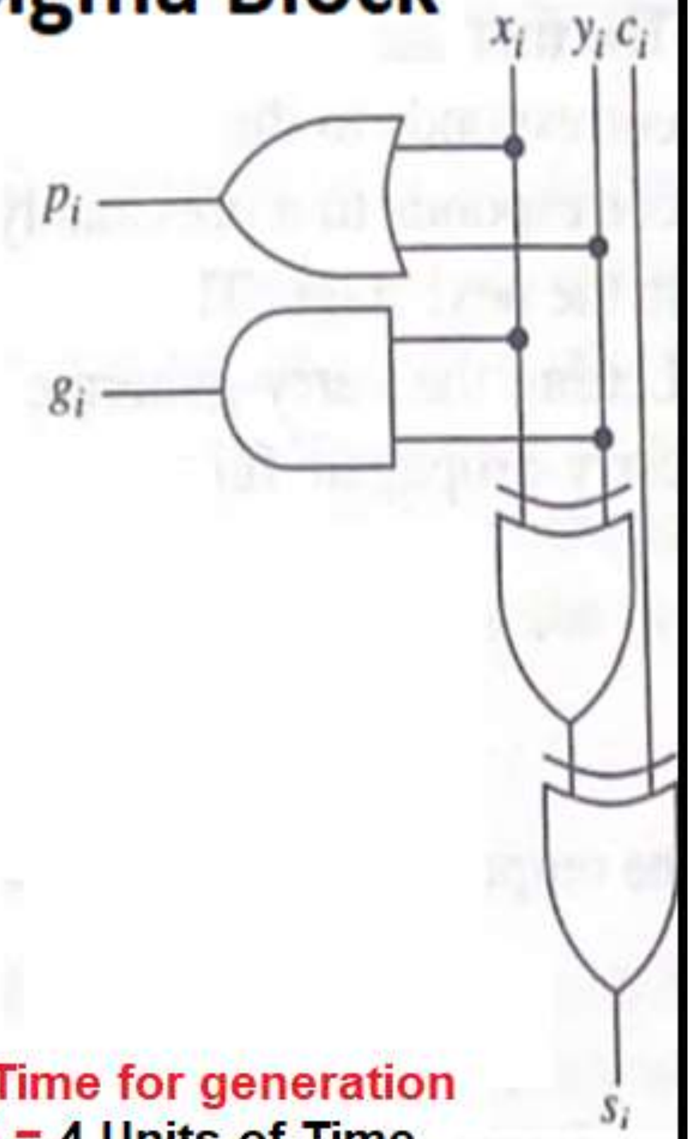
C4 = G3 + P3 C3 = G3 + P3 G2 + P3 P2 G1 + P3 P2 P1 G0 + P3 P2 P1 P0 C0



Design of 4 bit Carry Look Ahead Adder



Sigma Block



CLA Network, consists of Circuits, which implements following expressions.

Gi = XiYi, Pi = Xi + Yi

C1 = G0 + P0 C0

C2 = G1 + P1 C1 = G1 + P1 G0 + P1 P0 C0

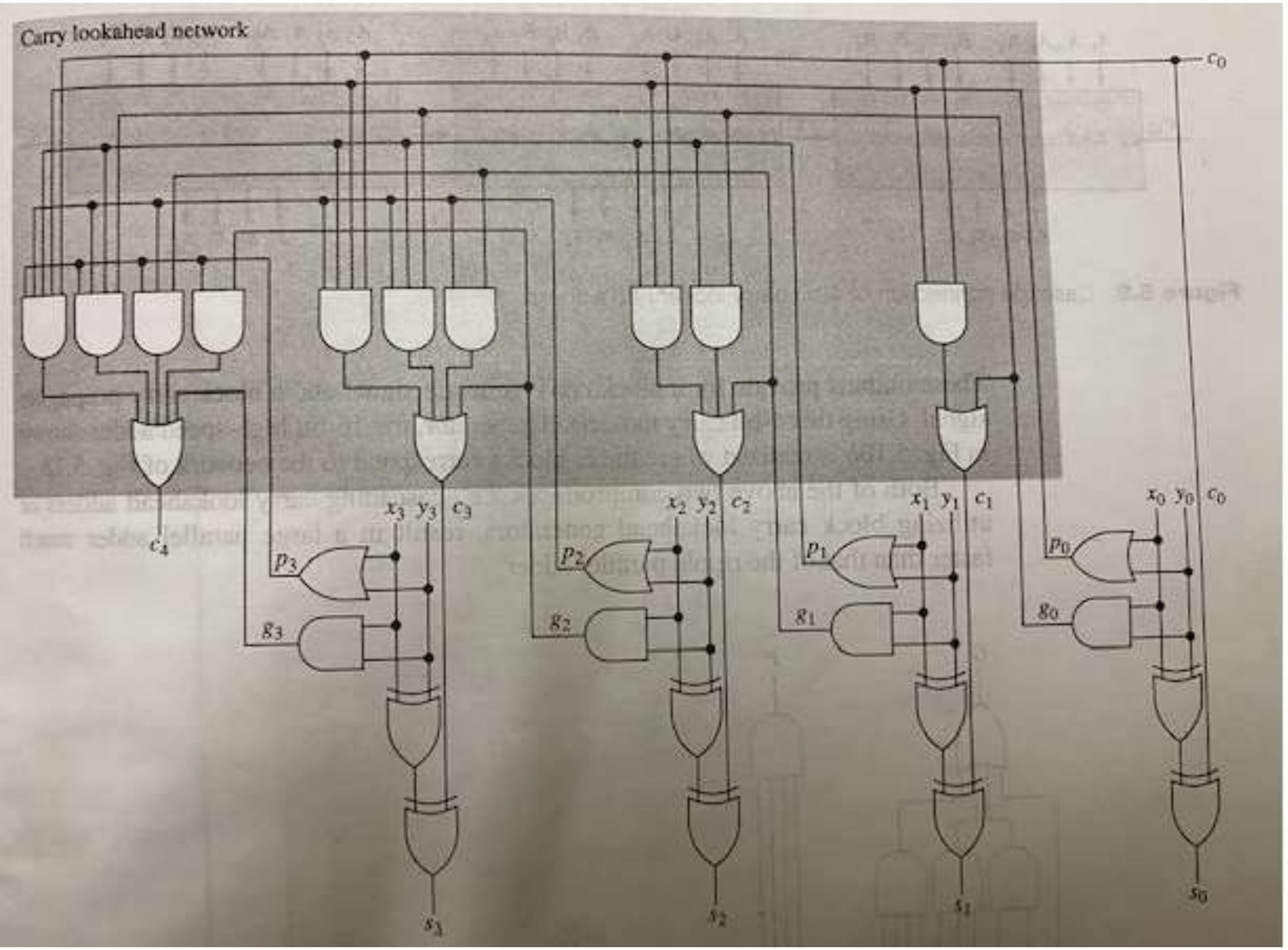
C3 = G2 + P2 C2 = G2 + P2 G1 + P2 P1 G0 + P2 P1 P0 C0

C4 = G3 + P3 C3 = G3 + P3 G2 + P3 P2 G1 + P3 P2 P1 G0 + P3 P2 P1 P0 C0

Total Time = 1Unit of Time for generation of Pi/Gi(AND/OR gate), 2 Units of Time for generation of Ci (AND-OR gates), 1Unit of Time for generation of Sum bit,Si (XOR gate) = 4 Units of Time

Note: 1Unit of Time, is the propogation delay of one gate.

Carry Look Ahead Adder (following diagram for understanding)



carry look ahead adder for 4 bit opernads is shown in the figure. Generalizing from the figure, The path length from the generation of carry to its apperance as an input at any higher order stage, i.e the path length through any stage of the carry look ahead network, is two levels of logic.
Thus with one level of logic to from Gi, two levels of logic for the carry to propogate between any two stages, and one level of logic to have the carry effect a sum output, the maximum propogation delay for CLA is 4 Units of time, assuming each gate introduces a unit of time of propogation delay.

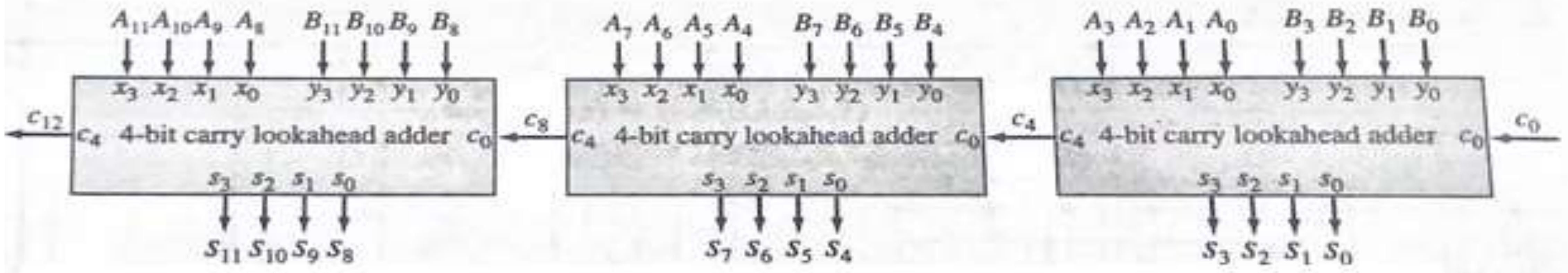
<- 4Bit CLA Desgin

Even though CLA performs high speed addition based on the carry look ahead principle, it presents a limitation in the realization of large high speed adders. The carry lookahead network can get quite large in terms of gates and gate inputs as the number of bits in the operands increases.

One approach to circumvent this problem is to divide the bits of the operands into blocks. Then, by using carry look ahead adders/carry look ahead generators for each block, their cascade connection results in a large high speed adders.

1. Cascading CLA adders
2. Using block CLA generators

Cascade Connection of 4-bit carry lookahead adders



Let 'n' be the no. of bits in an operand,is divisible by 4.

A carry generated in the least significant adder stage requires one level of logic, two levels of logic are needed to propagate the carry to the end of the first stage, two levels of logic are needed to propagate the carry through each of the remaining (n/4)-1 stages, and, finally,one level of logic is needed to effect a sum output in the final stage.

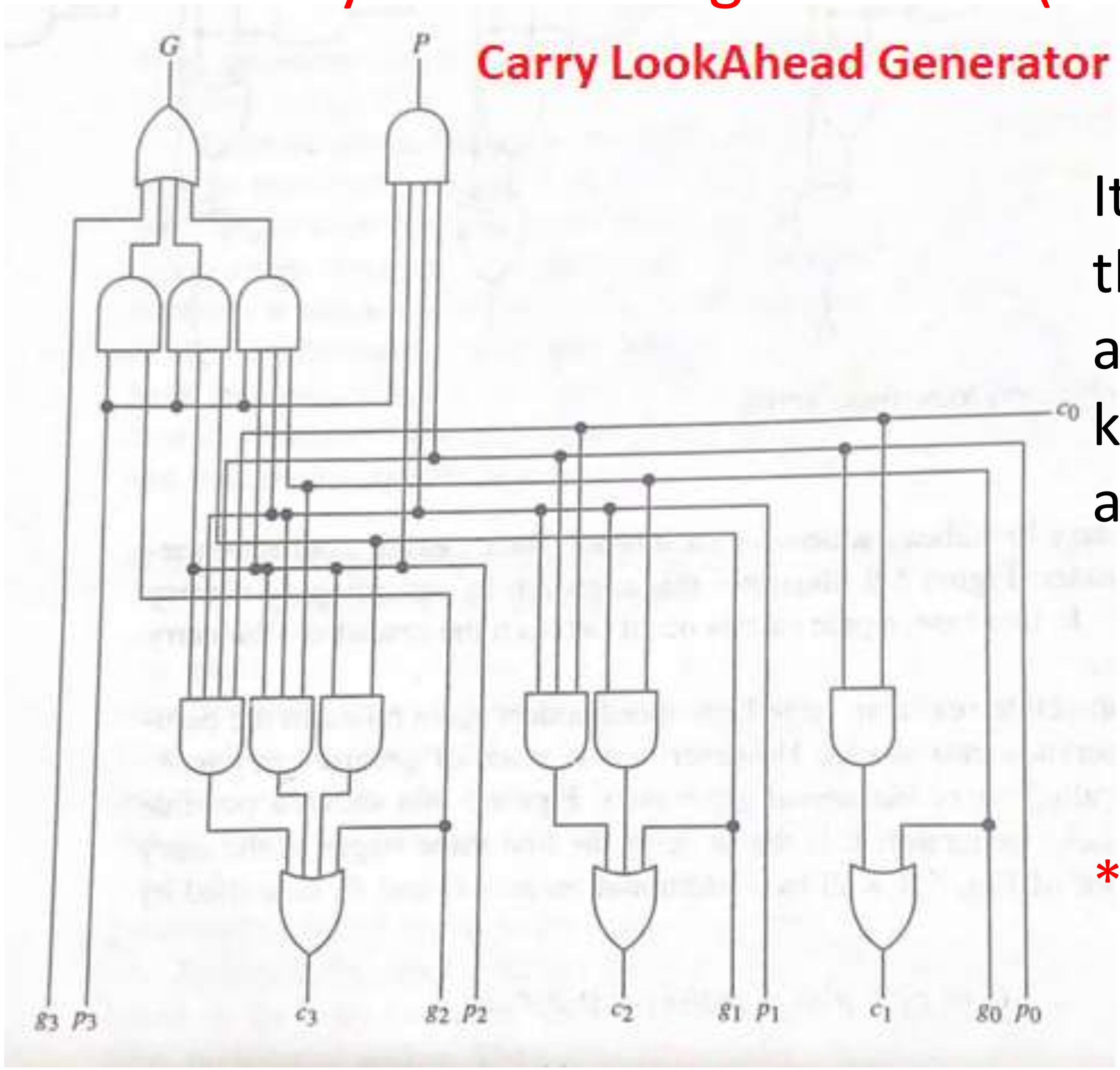
Therefore,

$$= (1+2)+2\,[(n/4)-1]\ +1\ =2+(n/2)$$

levels of logic corresponds to the longest path length

Note: in this case ripple carries occur between the cascaded 4 bit carry look ahead adders

Carry lookahead generator (Extra Info – Not in syllabus)



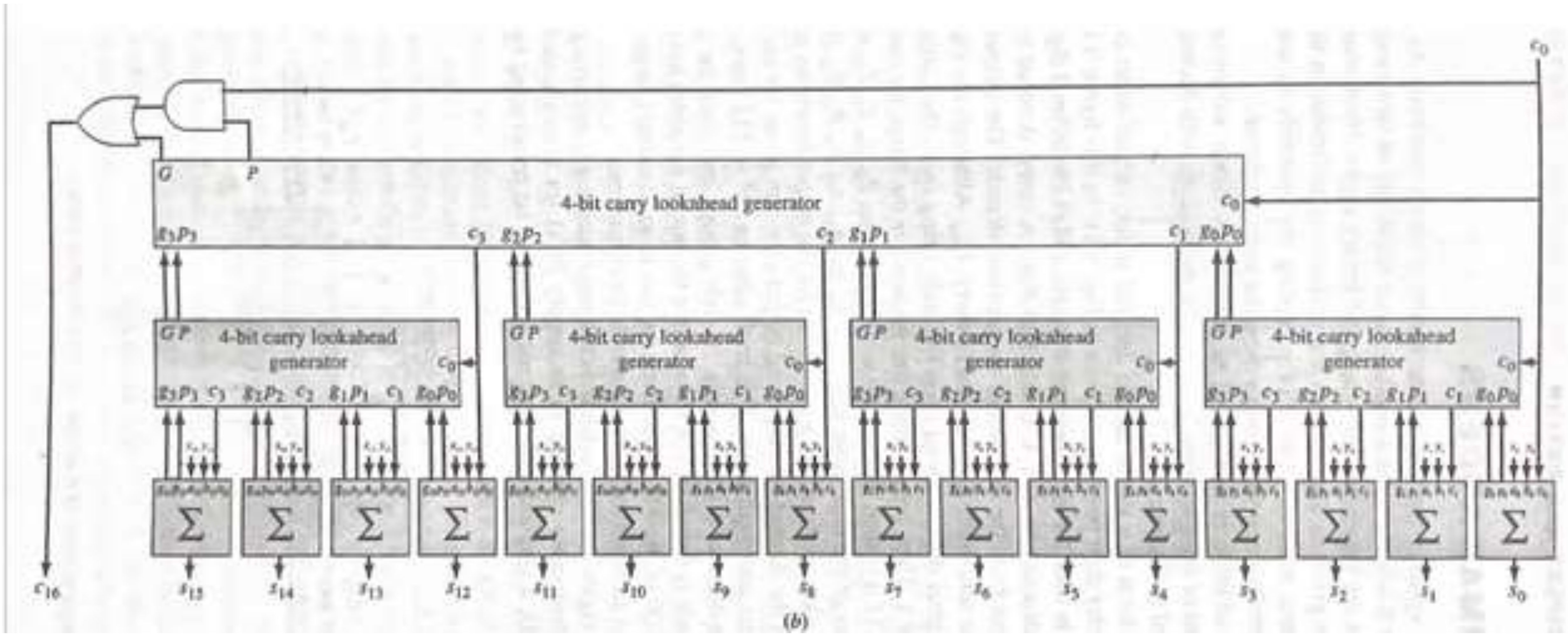
It is the same as the first three stages of the carry look ahead network (of CLA adder), with two additional G and P, known as "block carry generate signal" and "block carry propagate signal"

$G = g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0$

$P = p_3p_2p_1p_0$

Extra info: Not in syllabus

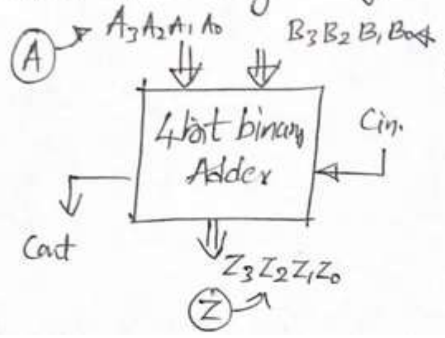
Using these 4 bit carry lookahead generator, the 16 bit high speed adders are realized (Extra info: not in syllabus)



(b)

Decimal Adder

let us use 4 bit binary adder to add, two decimal digits. Assume decimal digits are represented in binary using BCD (8421 code) format.



A – 0 to 9
B – 0 to 9

[BCD format

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

let us add few numbers, ①

```
  5  – 0101      5  – 0101     8 – 1000
+ 4  – 0100      6  – 0110     9 – 1001
 (9) – 1001    (11) – 1011   (17)   0001
```

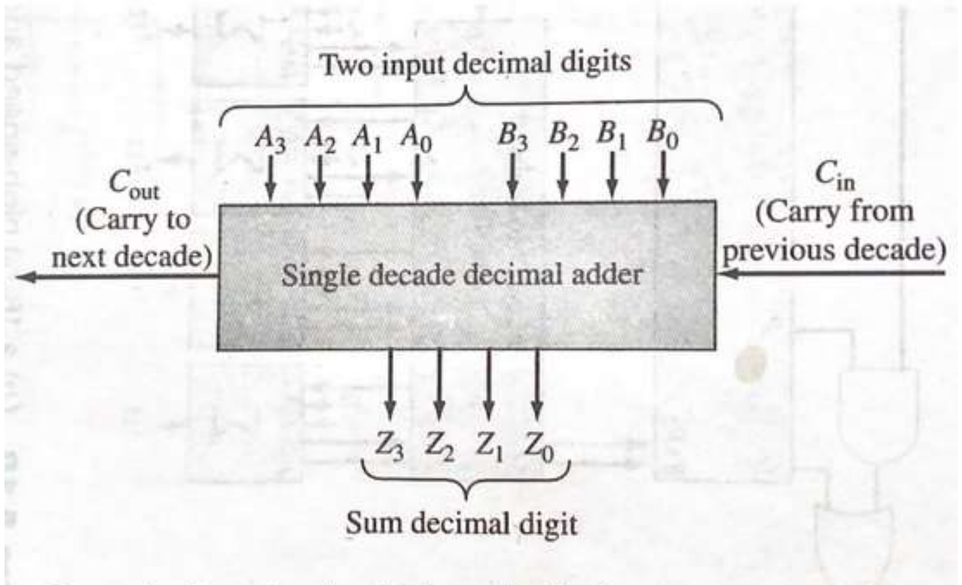
In first case, 5+4, answer generated by binary adder is correct. In second case, 5+6, expected decimal answer is 11, but we are getting '1011', which is not a valid BCD digit (not in the range 0000–1001). In third case, 8+9, answer is 17, but we get 1 0001, i.e 11, wrong answer.

These examples indicate 4 bit binary adder can't be used directly for adding decimal numbers.

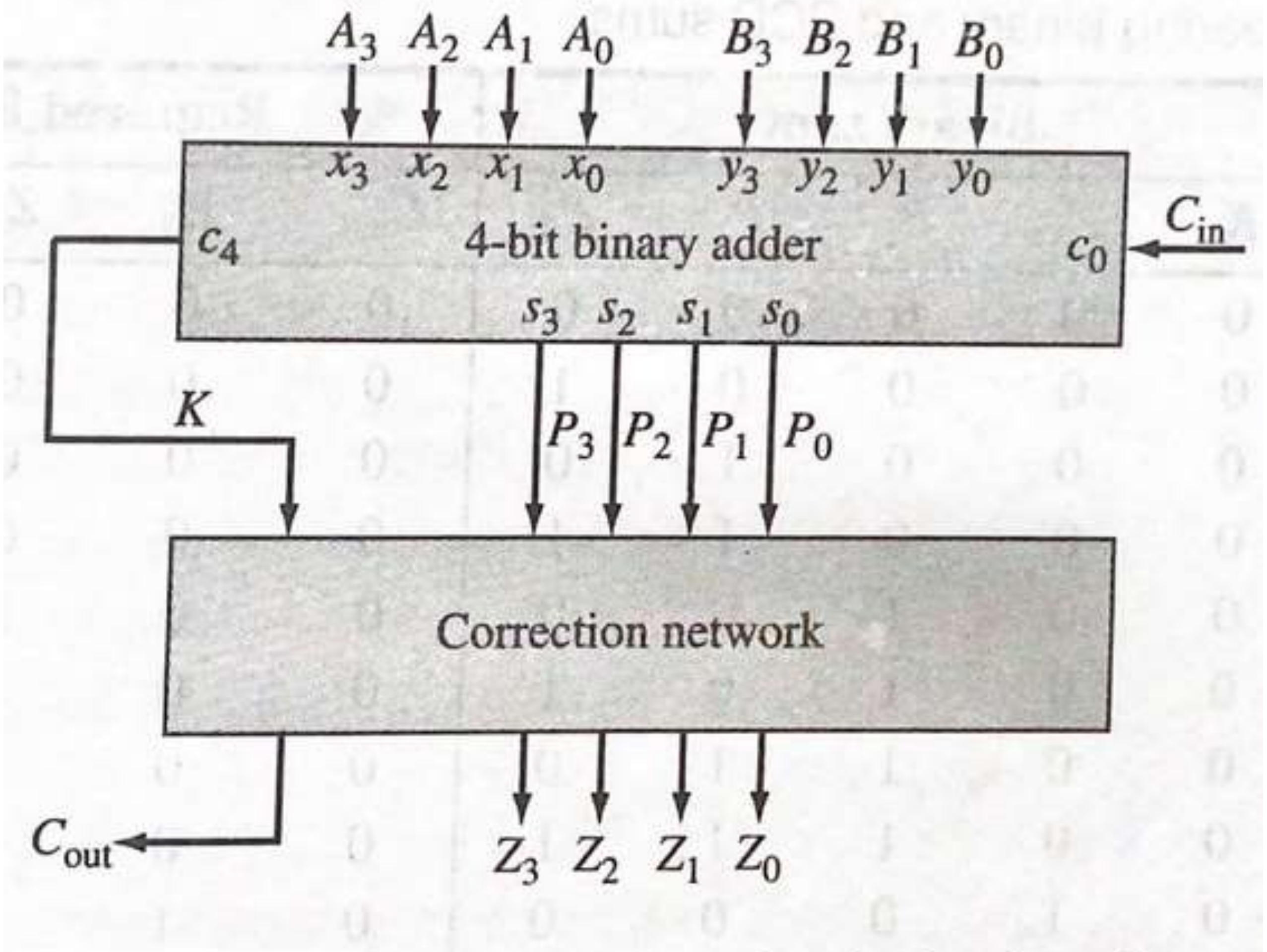
By adding '6', we can provide correction to the wrong answer generated in the previous examples.

```
(dec)  5        – 0101          (dec)  8      1 1000
       6        – 0110                 9        1001
      11          1011                17        0001
        add '6'   0110            add '6'       0110
                1 0001                        ① 0111
                ① ①  ✓                          ⑦  ✓
```

Decimal Adder

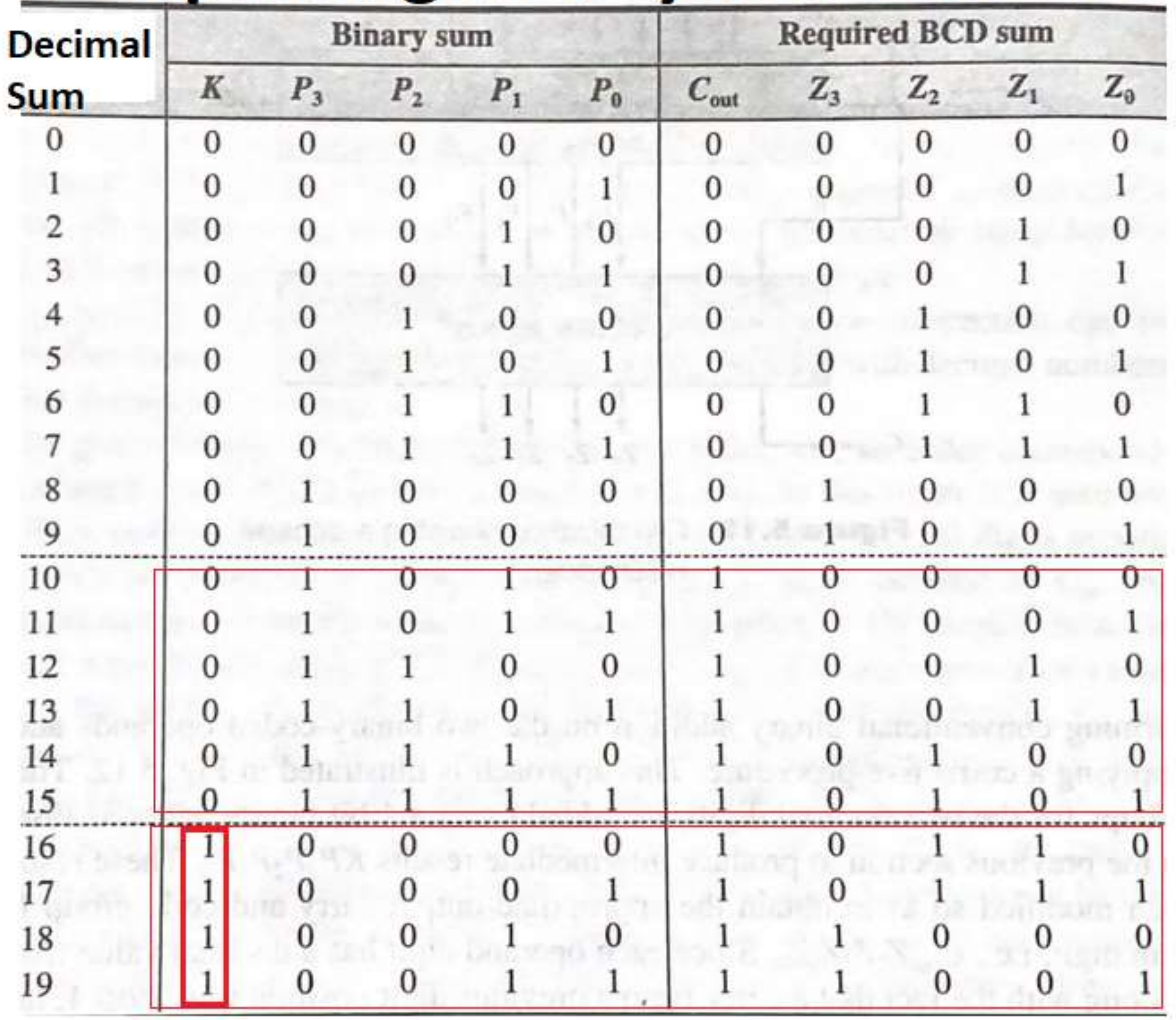

Two input decimal digits
A_3 A_2 A_1 A_0 B_3 B_2 B_1 B_0
C_{out} (Carry to next decade)
C_{in} (Carry from previous decade)
Single decade decimal adder
Z_3 Z_2 Z_1 Z_0
Sum decimal digit

Organization of a single-decade decimal adder.



Organization of a single-decade BCD adder.

Comparting binary and BCDsums



Decimal	Binary sum					Required BCD sum				
Sum	K	P_3	P_2	P_1	P_0	C_{out}	Z_3	Z_2	Z_1	Z_0
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	1
2	0	0	0	1	0	0	0	0	1	0
3	0	0	0	1	1	0	0	0	1	1
4	0	0	1	0	0	0	0	1	0	0
5	0	0	1	0	1	0	0	1	0	1
6	0	0	1	1	0	0	0	1	1	0
7	0	0	1	1	1	0	0	1	1	1
8	0	1	0	0	0	0	1	0	0	0
9	0	1	0	0	1	0	1	0	0	1
10	0	1	0	1	0	1	0	0	0	0
11	0	1	0	1	1	1	0	0	0	1
12	0	1	1	0	0	1	0	0	1	0
13	0	1	1	0	1	1	0	0	1	1
14	0	1	1	1	0	1	0	1	0	0
15	0	1	1	1	1	1	0	1	0	1
16	1	0	0	0	0	1	0	1	1	0
17	1	0	0	0	1	1	0	1	1	1
18	1	0	0	1	0	1	1	0	0	0
19	1	0	0	1	1	1	1	0	0	1

If the sum of the two decimal digits and input carry is less than 10, the required BCD sum is same as Binary Sum, produced by the 4 bit binary adder,

$$K\ P_3P_2P_1P_0 = C_{out}\ Z_3Z_2Z_1Z_0$$

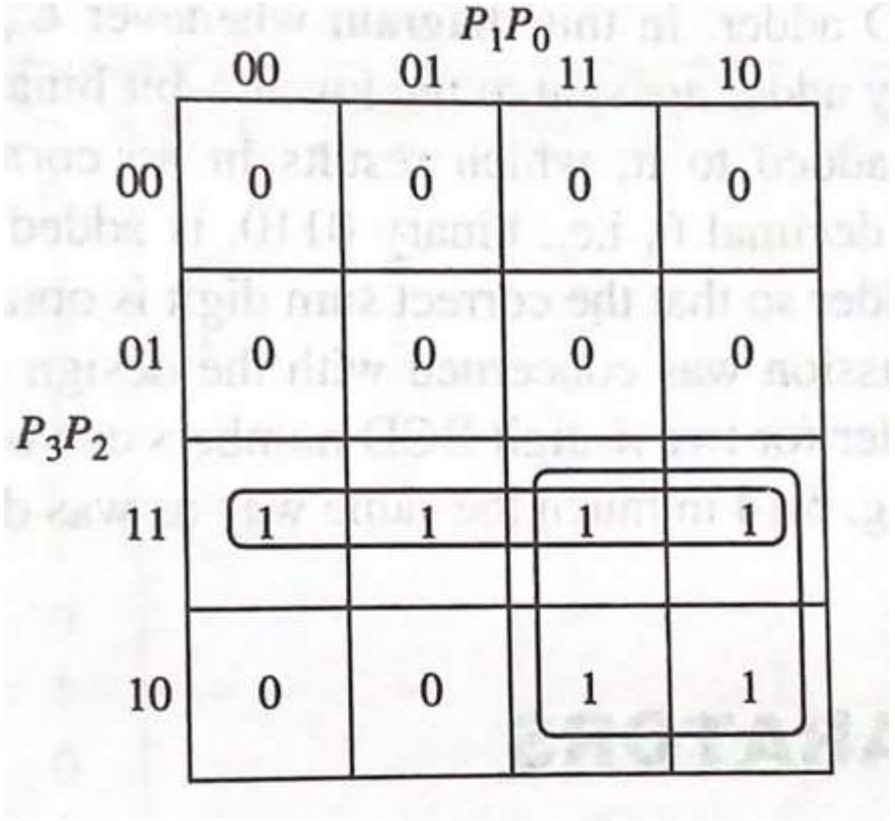
The correction, of adding '6' is required in two cases.

1. **Answer is invalid BCD codes, (A to F), decimal sums of 10 throguh 15**

$K\ P_3P_2P_1P_0$ –>0 1010, 0 1011, 0 1100, 0 1101, 0 1110, 0 1111

2. **Answer produces, the Carry, decimal sums 16 through 19**

$K\ P_3P_2P_1P_0$ -> 1 0000, 1 0001, 1 0010, 1 0011



P_3P_2 \ P_1P_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

KMap to detect the combinations

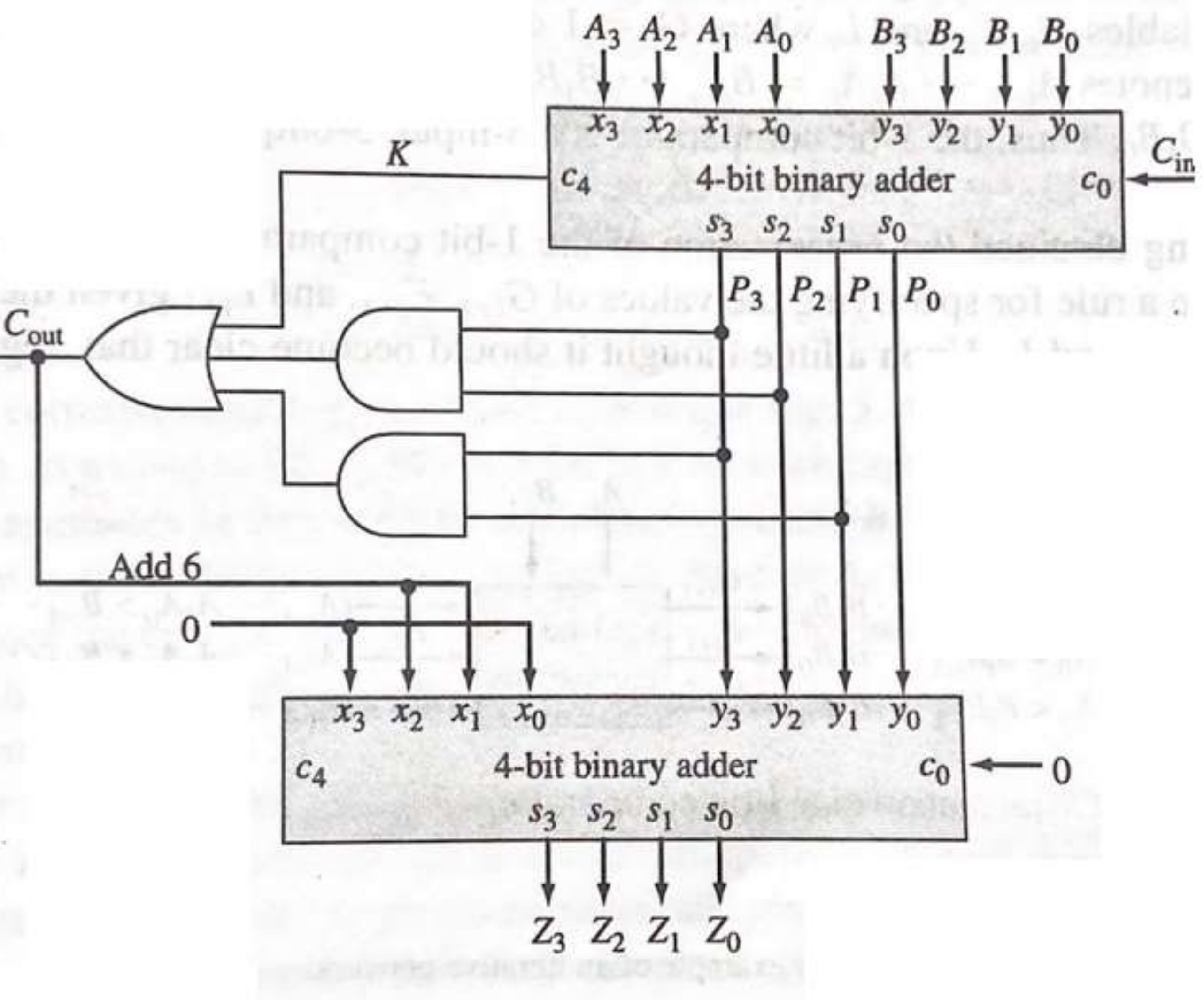
1010,1011,1100,1101,1110,1111

from the Kmap

$$ADD\_6 = P_3P_2 + P_3P_1$$

Including the Carry output (K), in the expression, the Boolean expression for adding the correction is

$$ADD\_6 = K + P_3P_2 + P_3P_1$$



A single-decade BCD adder.

Corrective network is realised using another 4 bit binary adder, whenever ADD_6 is true, 0110 (6) is applied as one of the inputs to second adder, to add 6 to the result produced by the first adder.
Final Cout of the BCD adder is same as ADD_6 output.

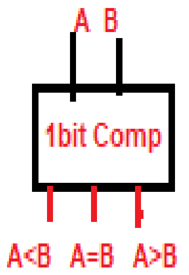
In this diagram, Whenever Cout (i.e ADD_6) Is 0, the output of the first adder is added with 0. Whenver Cout (ADD_6) is 1, the output of first adder is added with 6 (0110).

Note: Final Cout can also be generated using "OR" of Cout from first adder and Cout from second adder.
Cout = Cout_1 + Cout_2

Comparators

- Compare the magnitude of two binary numbers for the purpose of establishing whether one is greater than, equal to, or less than the other.
- A magnitude comparator is a combinational circuit that compares two numbers A and B and determines their relative magnitude. The outcome of the comparison is specified by 3 binary variables that indicate whether **A = B** or A**< B** or **A>B**.

Design of 1bit Comparator

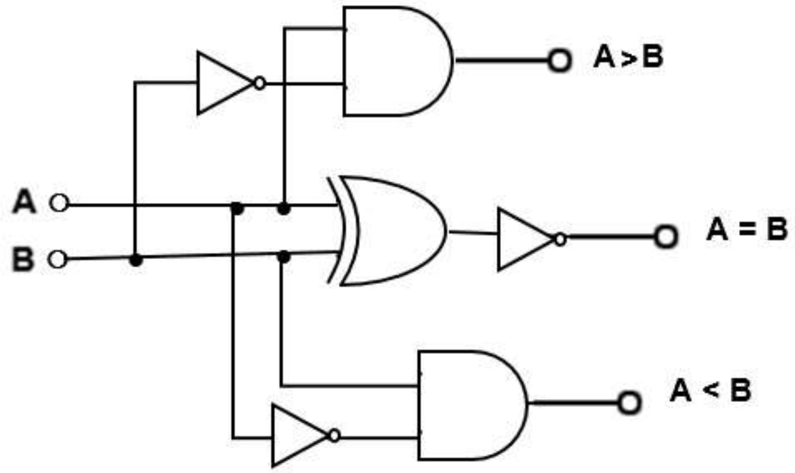


Boolean Exp (A<B) $= \bar{A}B$

Boolean Exp (A=B) $= \bar{A}\bar{B} + AB$

Boolean Exp (A>B) $= A\bar{B}$

Inputs		Outputs		
A	B	A<B	A=B	A>B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0



Designing of N bit Comparators

- An N bit comparator makes use of a cascade connection of identical subnetworks similar to the case of the parallel adder.
- Consider two n-bit binary numbers:

$$A = A_{n-1} \cdots A_i A_{i-1} \cdots A_1 A_0$$
$$B = B_{n-1} \cdots B_i B_{i-1} \cdots B_1 B_0$$

- Assume A_i, B_i are entering the subnetwork and that the binary numbers are analyzed from right to left, Subnetwork is called a 1-bit comparator.
- 3 conditions describing the relative magnitudes of $A_{i-1} \cdots A_1 A_0$, $B_{i-1} \cdots B_1 B_0$

$G_i = 1$ denotes $A_{i-1} \cdots A_1 A_0 > B_{i-1} \cdots B_1 B_0$

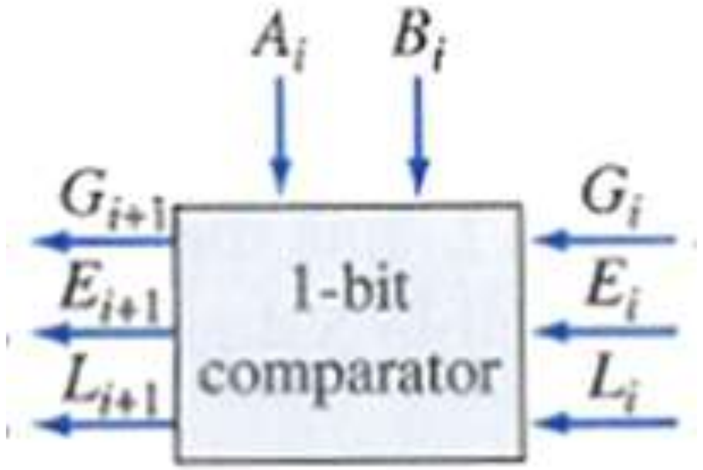
$E_i = 1$ denotes $A_{i-1} \cdots A_1 A_0 = B_{i-1} \cdots B_1 B_0$

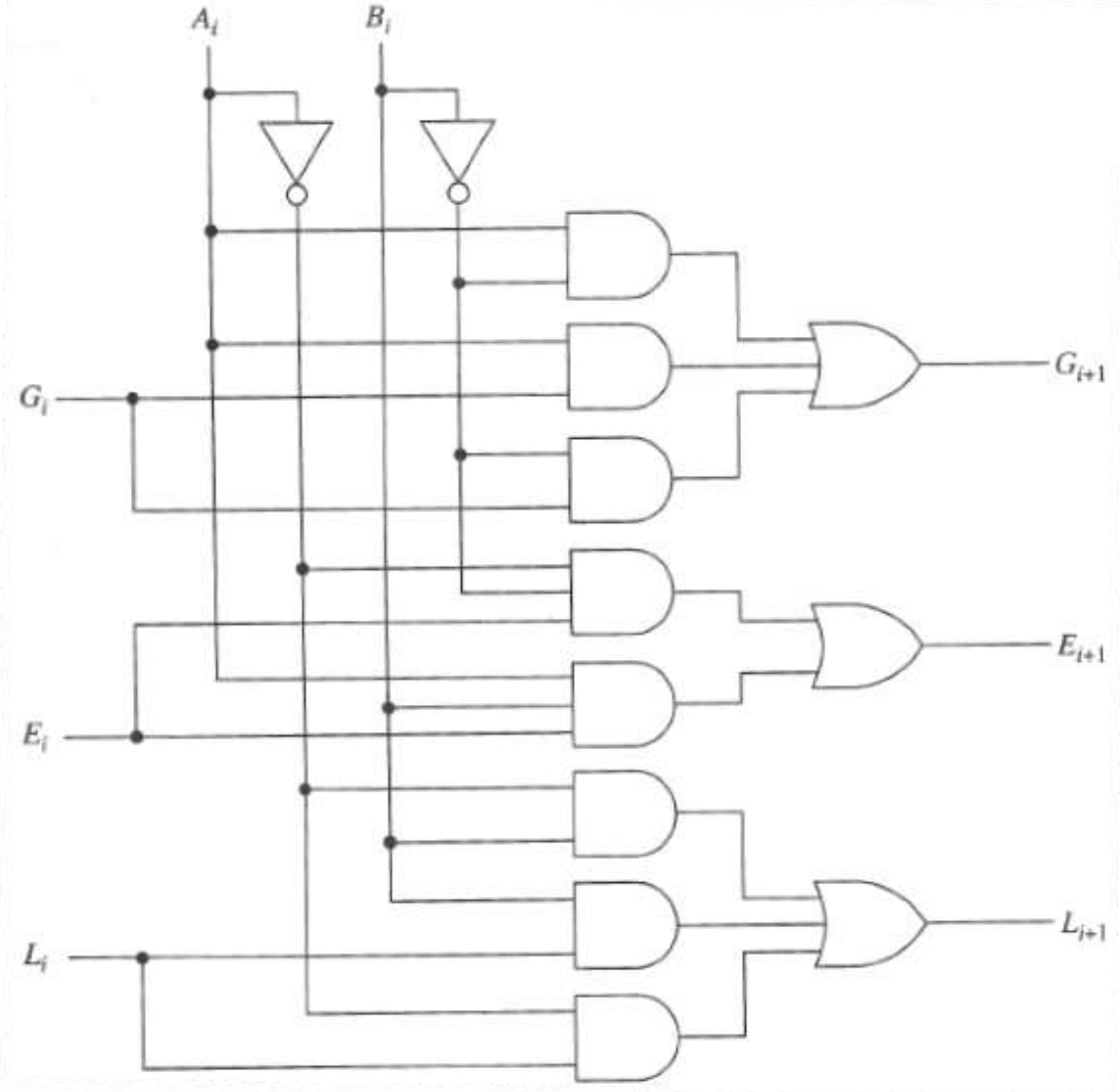
$L_i = 1$ denotes $A_{i-1} \cdots A_1 A_0 < B_{i-1} \cdots B_1 B_0$

- 1-bit comparator is a 5-input 3-output network

- Rules:
  - If $A_i = 0, B_i = 1$ then $L_{i+1} = 1$
  - If $A_i = 1, B_i = 0$ then $G_{i+1} = 1$

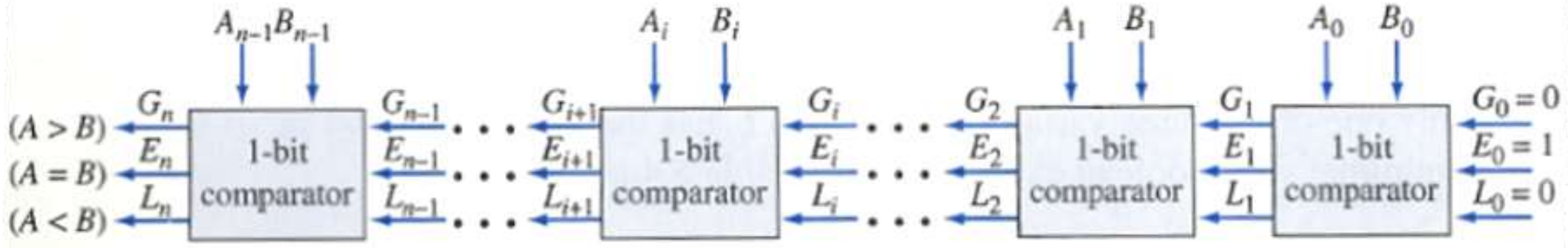
  - If $A_i = B_i$ and $L_i = 1$ then $L_{i+1} = 1$
  - If $A_i = B_i$ and $G_i = 1$ then $G_{i+1} = 1$
  - If $A_i = B_i$ and $E_i = 1$ then $E_{i+1} = 1$
- Can use this to construct a truth table and derive the Boolean expressions





Minimal Sum Boolean Expressions:

$$G_{i+1} = A_i\overline{B}_i + A_iG_i + \overline{B}_iG_i$$
$$E_{i+1} = \overline{A}_i\overline{B}_iE_i + A_iB_iE_i$$
$$L_{i+1} = \overline{A}_iB_i + B_iL_i + \overline{A}_iL_i$$



$A_{n-1}B_{n-1}$
A_i
B_i
A_1
B_1
A_0
B_0
$(A > B)$
$(A = B)$
$(A < B)$
G_n
E_n
L_n
1-bit comparator
G_{n-1}
E_{n-1}
L_{n-1}
G_{i+1}
E_{i+1}
L_{i+1}
1-bit comparator
G_i
E_i
L_i
G_2
E_2
L_2
1-bit comparator
G_1
E_1
L_1
1-bit comparator
$G_0 = 0$
$E_0 = 1$
$L_0 = 0$