

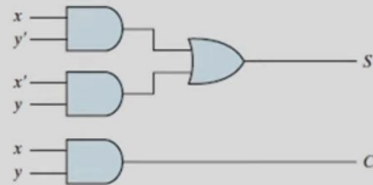
Adder Design

Half Adder, Full Adder, Ripple Carry, Carry Look Ahead,

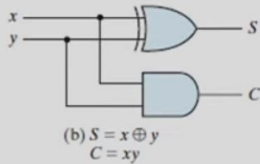
Adder for more bits, BCD adder, adder circuit reused for subtraction, overflow

Half Adder

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



$$(a) S = xy' + x'y$$
$$C = xy$$



$$(b) S = x \oplus y$$
$$C = xy$$

Full adder

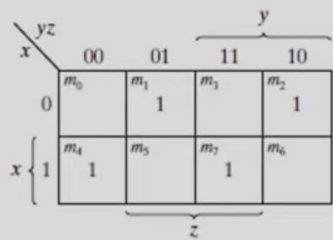
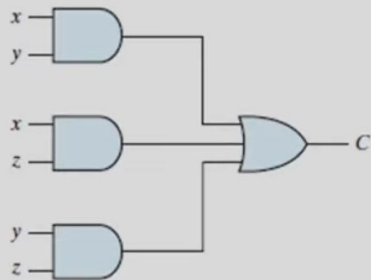
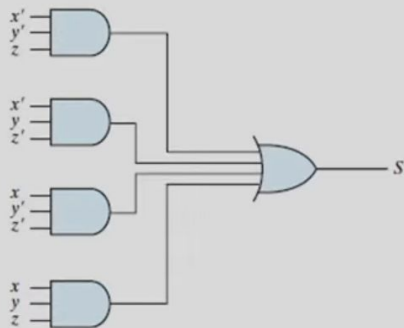
- For example, addition of the binary numbers 1011 and 0011

$$\begin{array}{r} 0\ 1\ 1\ \text{carry-in} \\ 1\ 0\ 1\ 1\ \text{augend} \\ 0\ 0\ 1\ 1\ \text{addend} \\ \hline 1\ 1\ 1\ 0\ \text{sum} \end{array}$$

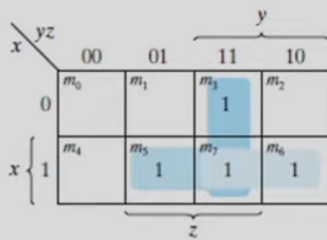
- The carry-out produced in the addition of the i th significant digits must be incorporated, as a carry-in, in the addition process for the $(i + 1)$ th significant digit.

Full Adder

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



(a) $S = x'y'z + x'yz' + xy'z' + xyz$



(b) $C = xy + xz + yz$

Full Adder – area optimized design

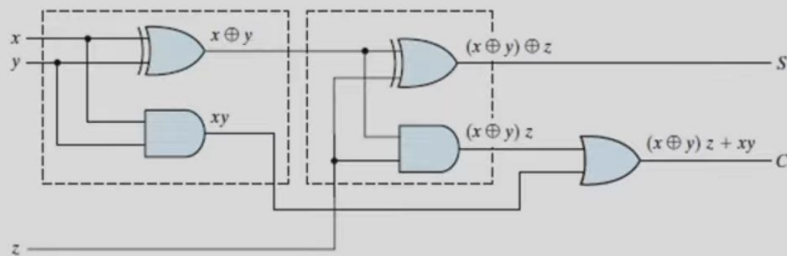


Table 4.4
Full Adder

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

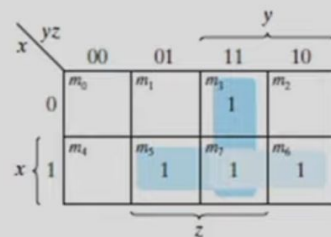
$$= xy'z' + x'yz' + xyz + x'y'z$$

$$= z'(xy' + x'y) + z(xy + x'y')$$

$$= z'(xy' + x'y) + z(xy' + x'y)'$$

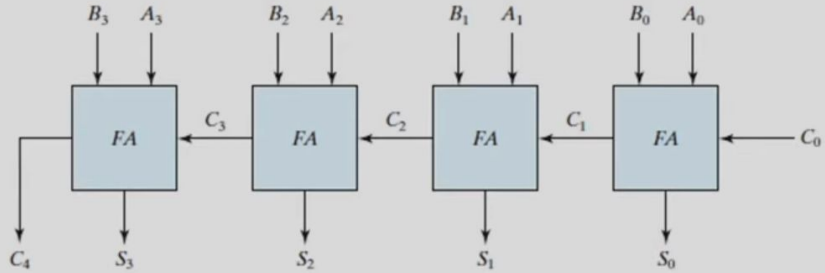
$$S = z \oplus (x \oplus y)$$

$$C = z(xy' + x'y) + xy = xy'z + x'yz + xy$$



(b) $C = xy + xz + yz$

Four bit binary adder



- Also known as **ripple carry adder**.

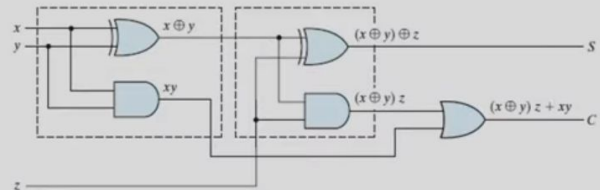
Verilog Implementation

```
//Gate-level hierarchical description of 4-bit adder
// Description of half adder
module halfadder (S, C, x, y);
    input x, y;
    output S, C;
    //Instantiate primitive gates
    xor (S, x, y);
    and (C, x, y);
endmodule

//Description of full adder
module fulladder (S, C, x, y, z);
    input x, y, z;
    output S, C;
    wire S1, D1, D2; //Outputs of first XOR and two AND gates
    //Instantiate the halfadder
    halfadder HA1 (S1, D1, x, y);
    halfadder HA2 (S, D2, S1, z);
    or g1 (C, D2, D1);
endmodule
```

```
module four_bit_adder (S, C4, A, B, C0);
    input [3:0] A, B;
    input C0;
    output [3:0] S;
    output C4;
    wire C1,C2,C3; //Intermediate carries

    //Instantiate the full adder cells
    fulladder FA0 (S[0],C1,A[0],B[0],C0),
               FA1 (S[1],C2,A[1],B[1],C1),
               FA2 (S[2],C3,A[2],B[2],C2),
               FA3 (S[3],C4,A[3],B[3],C3);
endmodule
```



```
module testbench ( );
```

```
// Inputs
```

```
reg [3:0] a;
```

```
reg [3:0] b;
```

```
reg cin;
```

```
// Outputs
```

```
wire [3:0] sum;
```

```
wire cout;
```

```
four_bit_adder DUT(
```

```
.A(a),
```

```
.B(b),
```

```
.C0(cin),
```

```
.S(sum),
```

```
.C4(cout)
```

```
);
```

```
reg clk;
```

```
reg rst;
```

```
// Initial values
```

```
initial begin
```

```
clk = 0;
```

```
rst = 1;
```

```
#20;
```

```
rst = 0;
```

```
end
```

Simulation

```
// Clock generator
```

```
always begin
```

```
#10 clk = ~clk;
```

```
end
```

```
// Test sequence
```

```
reg [3:0] i;
```

```
always @(posedge clk, posedge rst) begin
```

```
if (rst) begin
```

```
i = 0;
```

```
end else begin
```

```
// Apply all possible val values (0-9)
```

```
a <= $random%16;
```

```
b <= $random%16;
```

```
cin <= $random%2;
```

```
#20;
```

```
i = i + 1;
```

```
if (i == 10) $finish;
```

```
end
```

```
$monitor("%d: a = %b, b = %b, cin = %b, sum = %b, cout = %b", $time, a, b, cin, sum, cout);
```

```
end
```

```
endmodule
```

```
10: a = xxxx, b = xxxx, cin = x, sum = xxxx, cout = x
```

```
30: a = 0100, b = 0001, cin = 1, sum = 0110, cout = 0
```

```
50: a = 0011, b = 1101, cin = 1, sum = 0001, cout = 1
```

```
70: a = 0101, b = 0010, cin = 1, sum = 1000, cout = 0
```

```
90: a = 1101, b = 0110, cin = 1, sum = 0100, cout = 1
```

```
110: a = 1101, b = 1100, cin = 1, sum = 1010, cout = 1
```

```
130: a = 0110, b = 0101, cin = 0, sum = 1011, cout = 0
```

```
150: a = 0101, b = 0111, cin = 0, sum = 1100, cout = 0
```

```
170: a = 1111, b = 0010, cin = 0, sum = 0001, cout = 1
```

```
190: a = 1000, b = 0101, cin = 0, sum = 1101, cout = 0
```

```
210: a = 1101, b = 1101, cin = 1, sum = 1011, cout = 1
```

RIPPLE-CARRY ADDER – Area Overhead

- Time required to perform addition is the time required for the propagation (or ripple) of the carries in the stages.
- Large combinational/propagation delay
 - $2n$ for normal design for n bits addition
 - $3n$ for area optimized design for n bits addition
- The time allotted must be at least equal to the longest carry-propagation time (plus the addition time in the last full adder).
- In order to increase the speed of the adder, it is necessary to minimize the fixed time required for carry propagation.

Carry Look ahead adder

- G_i is called a *carry generate*, and it produces a carry of 1 when both A_i and B_i are 1, regardless of the input carry C_i .

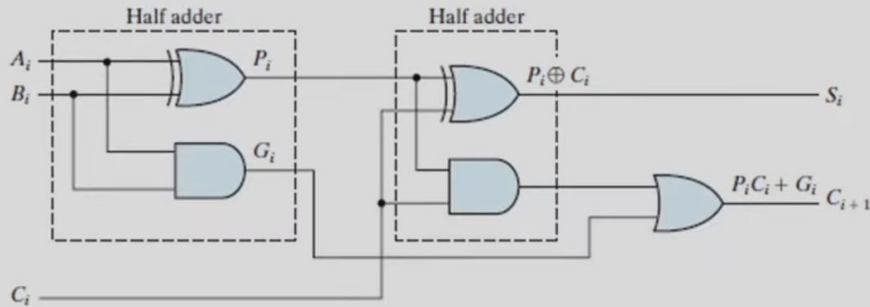


Table 4.4
Full Adder

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Carry Look ahead adder

$$P_i = A_i \oplus B_i$$

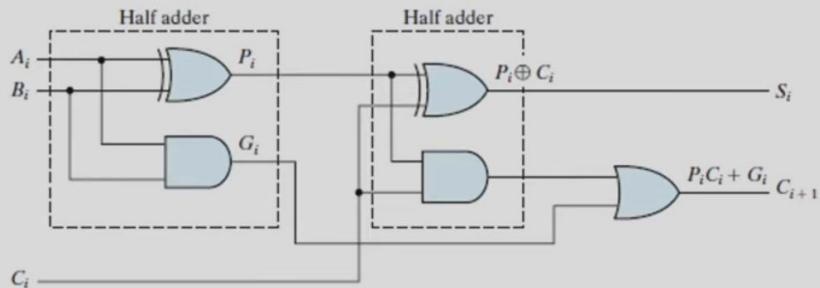
$$G_i = A_i B_i$$

$$S_i = P_i \oplus C_i$$

$$C_{i+1} = G_i + P_i C_i$$

C_0 = input carry

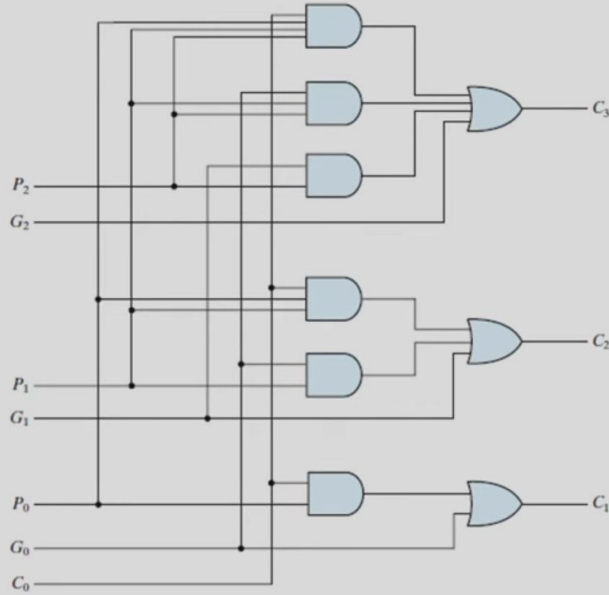
$$C_1 = G_0 + P_0 C_0$$



$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 (G_0 + P_0 C_0) = G_1 + P_1 G_0 + P_1 P_0 C_0$$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$

Carry Look ahead adder



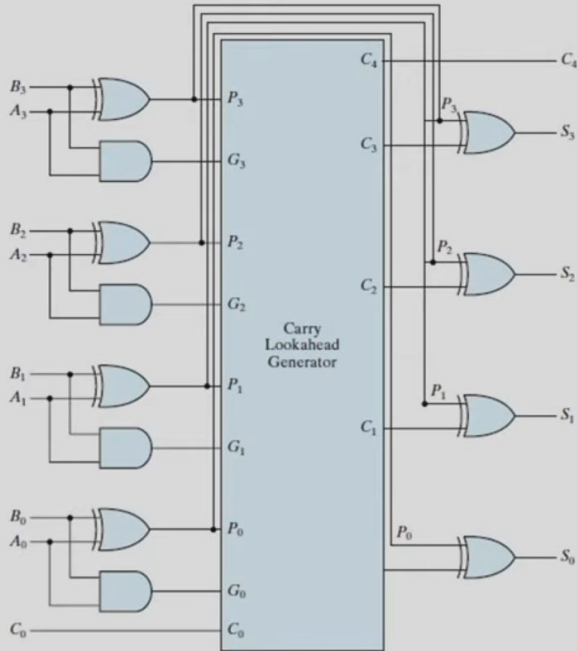
C_0 = input carry

$$C_1 = G_0 + P_0C_0$$

$$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0) = G_1 + P_1G_0 + P_1P_0C_0$$

$$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$$

Four bit adder with carry Look ahead adder



What is the delay of a n-bit adder?

Constant 4

Verilog Implementation

```
module halfadder (S, C, x, y);
    input x, y;
    output S, C;
    //Instantiate primitive gates
    xor (S, x, y);
    and (C, x, y);
endmodule

module carrygenerator(cin, p0, p1, p2, p3, g0, g1, g2, g3, c0, c1, c2, c3, c4);
    input cin, p0, p1, p2, p3, g0, g1, g2, g3;
    output c0, c1, c2, c3, c4;

    assign c0=cin;
    assign c1=g0|(p0&cin);
    assign c2=g1|(p1&g0)|(p1&p0&cin);
    assign c3=g2|(p2&g1)|(p2&p1&g0)|(p1&p1&p0&cin);
    assign c4=g3|(p3&g2)|(p3&p2&g1)|(p3&p2&p1&g0)|(p3&p2&p1&p0&cin);
endmodule
```

```
module CLA_Adder(a,b,cin,sum,cout);
    input[3:0] a,b;
    input cin;
    output [3:0] sum;
    output cout;
    wire p0,p1,p2,p3,g0,g1,g2,g3,c1,c2,c3,c4;

    halfadder HA0(p0, g0, a[0], b[0]);
    halfadder HA1(p1, g1, a[1], b[1]);
    halfadder HA2(p2, g2, a[2], b[2]);
    halfadder HA3(p3, g3, a[3], b[3]);

    carrygenerator CG(cin, p0, p1, p2, p3, g0, g1,
g2, g3, c0, c1, c2, c3, c4);

    assign sum[0]=p0^c0;
    assign sum[1]=p1^c1;
    assign sum[2]=p2^c2;
    assign sum[3]=p3^c3;
    assign cout=c4;
endmodule
```

CLA_Adder dut (

.a(a),

.b(b),

.cin(cin),

.sum(sum),

.cout(cout)

);

// Clock and reset signals

reg clk;

reg rst;

// Initial values

initial begin

clk = 0;

rst = 1;

#20;

rst = 0;

end

// Clock generator

always begin

#10 clk = ~clk;

end

Simulation

// Test sequence

reg [3:0] i;

always @(posedge clk, posedge rst) begin

if (rst) begin

i = 0;

end else begin

// Apply all possible val values (0-9)

a <= \$random%16;

b <= \$random%16;

cin <= \$random%2;

#20;

i = i + 1;

if (i == 10) \$finish;

end

\$monitor("%d: a = %b, b = %b, cin = %b, sum =

%b, cout = %b", \$time, a, b, cin, sum, cout);

end

endmodule

10: a = xxxx, b = xxxx, cin = x, sum = xxxx, cout = x

30: a = 0100, b = 0001, cin = 1, sum = 0110, cout = 0

50: a = 0011, b = 1101, cin = 1, sum = 0001, cout = 1

70: a = 0101, b = 0010, cin = 1, sum = 1000, cout = 0

90: a = 1101, b = 0110, cin = 1, sum = 0100, cout = 1

110: a = 1101, b = 1100, cin = 1, sum = 1010, cout = 1

130: a = 0110, b = 0101, cin = 0, sum = 1011, cout = 0

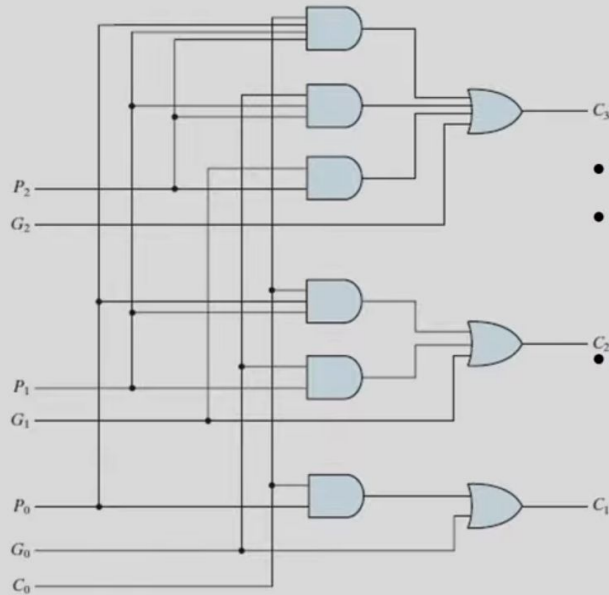
150: a = 0101, b = 0111, cin = 0, sum = 1100, cout = 0

170: a = 1111, b = 0010, cin = 0, sum = 0001, cout = 1

190: a = 1000, b = 0101, cin = 0, sum = 1101, cout = 0

210: a = 1101, b = 1101, cin = 1, sum = 1011, cout = 1

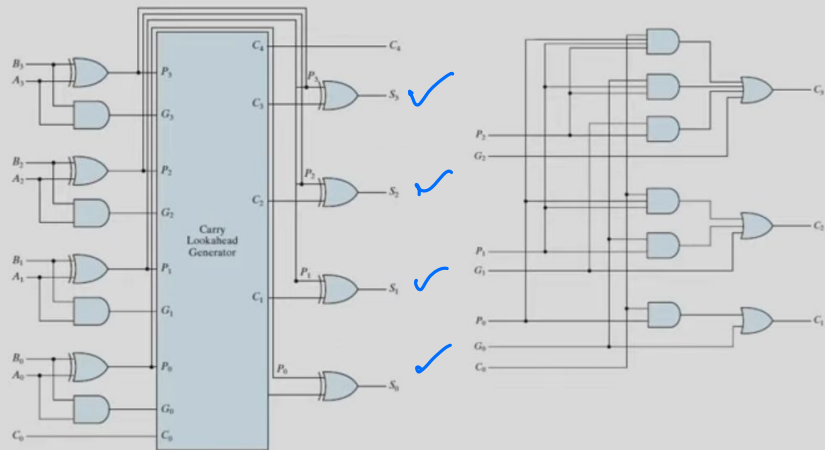
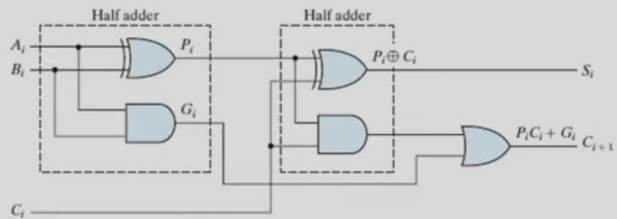
Area overhead of Carry look ahead adder



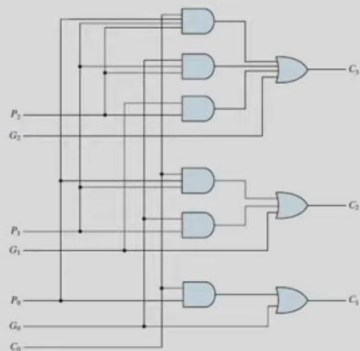
$$C_2 = G_1 + P_1C_1 = G_1 + P_1(G_0 + P_0C_0) = G_1 + P_1G_0 + P_1P_0C_0$$

$$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$$

- It requires a very large number of gates.
- For each stage of the adder it is necessary to have an OR gate with n inputs and n AND gates with 1 through n inputs
- Modern day computer is 64-bit word



CLA



Comparison

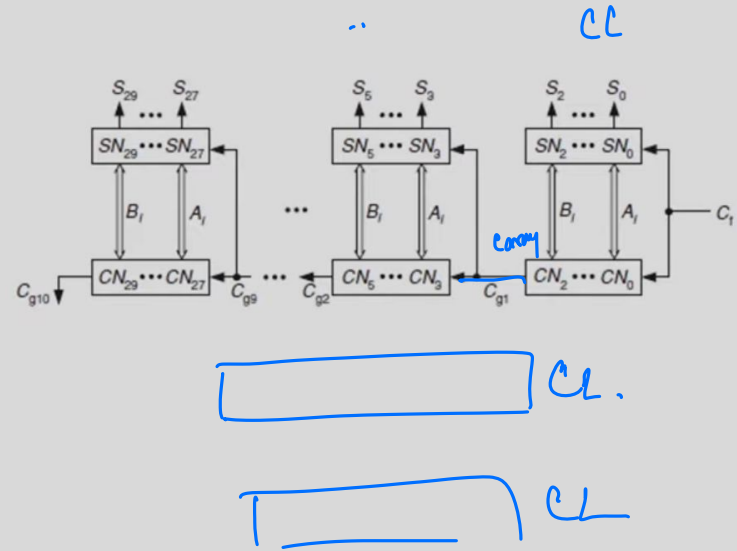
	Area (2 inputs gates)	Delay (unit)
32 bit RCA	160	96
32 bit CLA	~6500	4 - 32
(8 4-bit CLA in RCA node)	224	32 ← Ripple carry across CLAs
Comparison with CLA	29x less Area	8x slow
Comparison with RCA	1.4x more Area	3x faster

32bit
adder



Carry look ahead adder

- The limitation can be overcome, though at the expense of computation speed.
- By dividing the n stages of the adder into groups such that within each group a full carry look ahead is achieved while a ripple carry is maintained between groups.
- For group of size k -stages, delay: $2n/k$
- Area overhead:



Using Propagate and Generate Values

- **Sum** and **Carry_out** can be derived from **P_i** and **G_i** values
- **1 logic level** to generate **P_i** and **G_i**
 - treating AND and XOR as 1 gate level
- **1 logic level** to generate **Sum**

$$s_i = x_i \oplus y_i \oplus c_i$$
$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

$$G_i = x_i y_i$$
$$P_i = x_i \oplus y_i$$



$$s_i = P_i \oplus c_i$$
$$c_{i+1} = G_i + P_i c_i \quad (\text{verify})$$

Carry Lookahead Logic

$$c_{i+1} = G_i + P_i c_i$$

$$c_1 = G_0 + P_0 c_0$$

$$c_2 = G_1 + P_1 c_1 = G_1 + P_1(G_0 + P_0 c_0) = G_1 + P_1 G_0 + P_1 P_0 c_0$$

$$c_3 = G_2 + P_2 c_2 = G_2 + P_2(G_1 + P_1 G_0 + P_1 P_0 c_0) = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 c_0$$

$$c_4 = G_3 + P_3 c_3 = G_3 + P_3(G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 c_0)$$

$$= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 c_0 \rightarrow \text{area} \sim 6500 \quad \underline{32\text{bit}}$$

- **2 logic levels to generate c_4 from c_0**
- Approx: 5 i/p gate has same delay as 2 i/p gate



4-bit Carry Lookahead Adder (CLA)

$$\begin{aligned} G_i &= x_i y_i & s_i &= P_i \oplus c_i \\ P_i &= x_i \oplus y_i & c_{i+1} &= G_i + P_i c_i \end{aligned}$$

$$c_1 = G_0 + P_0 c_0$$

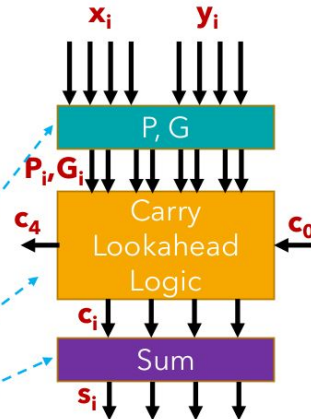
$$c_2 = G_1 + P_1 G_0 + P_1 P_0 c_0$$

$$c_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 c_0$$

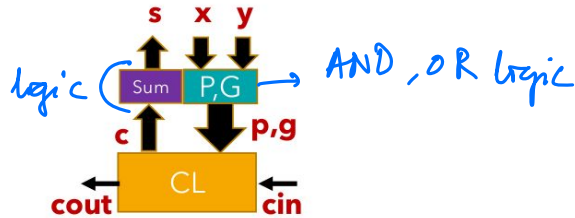
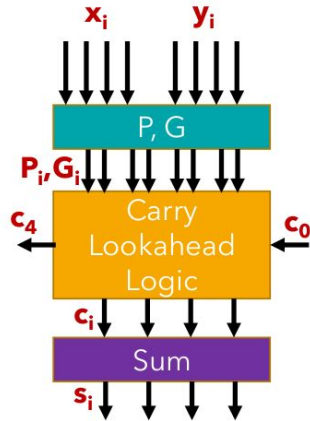
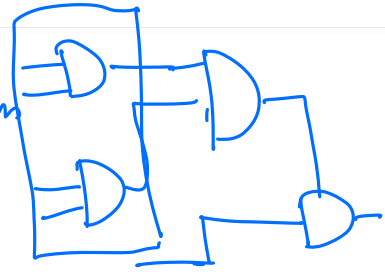
$$c_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 c_0$$

- 1 logic level to generate all P_i and G_i
- 2 logic levels to generate c_4 from c_0
 - Approx: 5 i/p gate has same delay as 2 i/p gate
- 1 logic level to generate all sums s_i
- 4-bit Adder delay: **1+2+1 = 4 levels**

(C) P. R. Panda, IIT Delhi, 2025



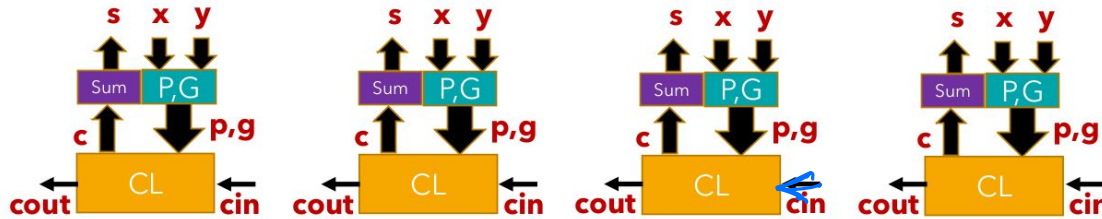
4-bit CLA: Simplified Diagram
 2 i/p AND 6 transistors
 15 i/p



~~16~~ input and gate

16-bit Adder from 4-bit CLA

4 bit-



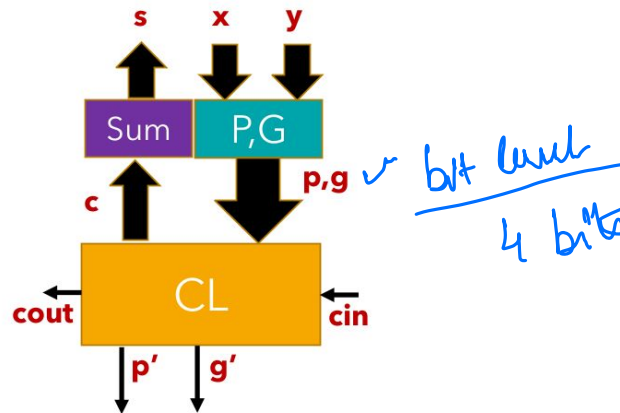
$$g_i = x_i y_i$$
$$p_i = x_i \oplus y_i$$

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

5/1P. Ripple carry.

How do we extend the structure?

CL block-level carry propagate/generate



$$g_i = x_i y_i$$

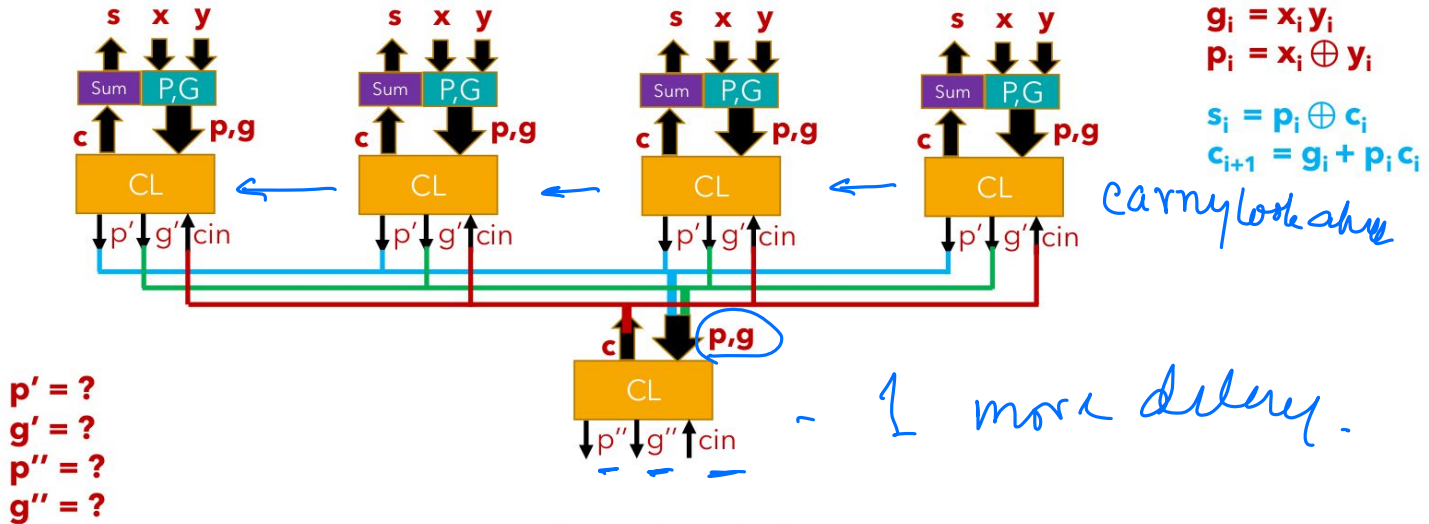
$$p_i = x_i \oplus y_i$$

$$s_i = p_i \oplus c_i$$

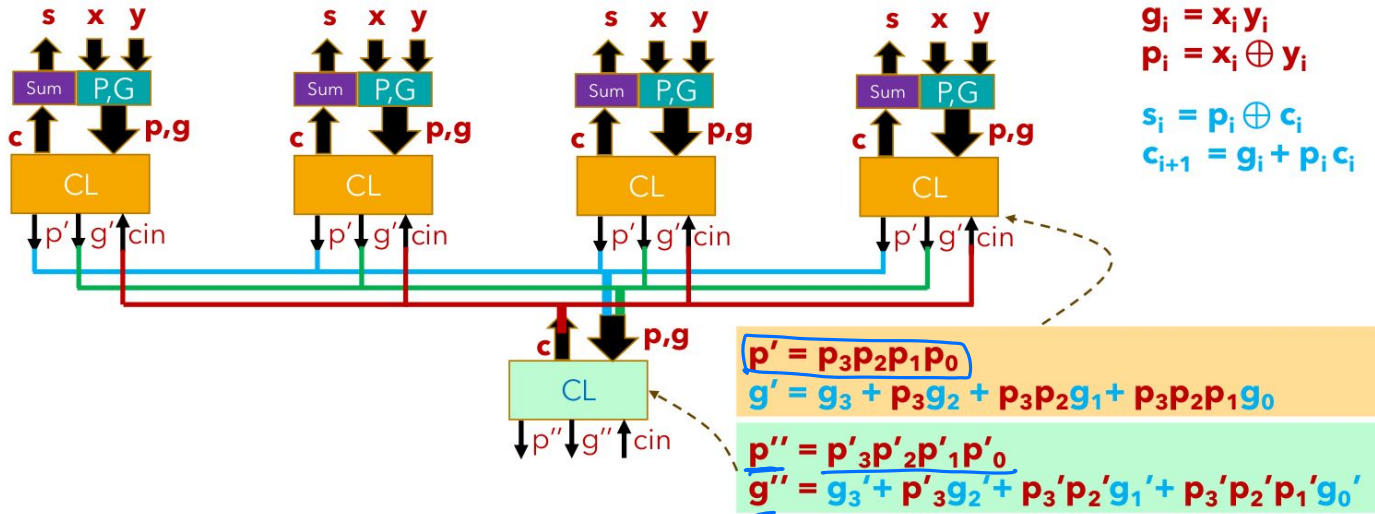
$$c_{i+1} = g_i + p_i c_i$$

$$p' = ?$$
$$g' = ?$$

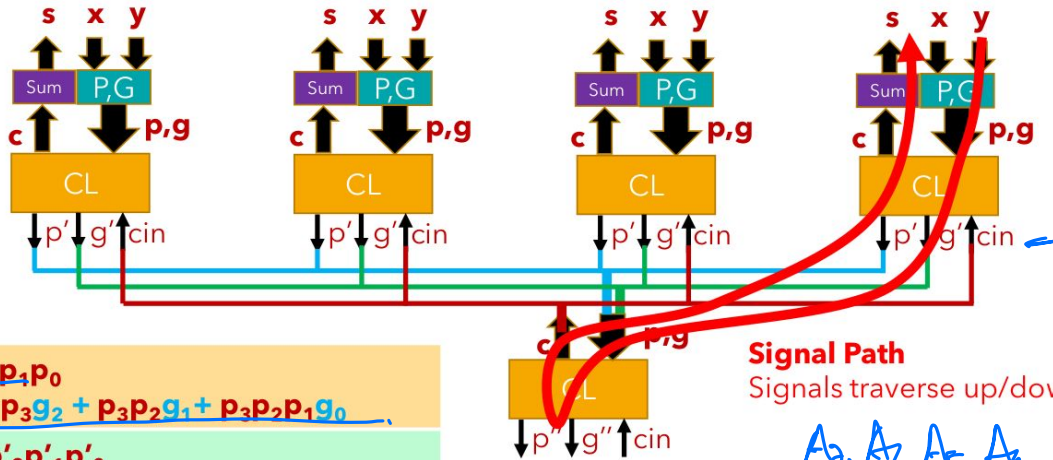
16-bit Adder from 4-bit CLA



16-bit Adder from 4-bit CLA



16-bit Adder from 4-bit CLA: Delay Analysis



$$g_i = x_i y_i$$

$$p_i = x_i \oplus y_i$$

$$s_i = p_i \oplus c_i$$

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

$$p' = p_3 p_2 p_1 p_0$$

$$g' = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0$$

$$p'' = p'_3 p'_2 p'_1 p'_0$$

$$g'' = g'_3 + p'_3 g'_2 + p'_3 p'_2 g'_1 + p'_3 p'_2 p'_1 g'_0$$

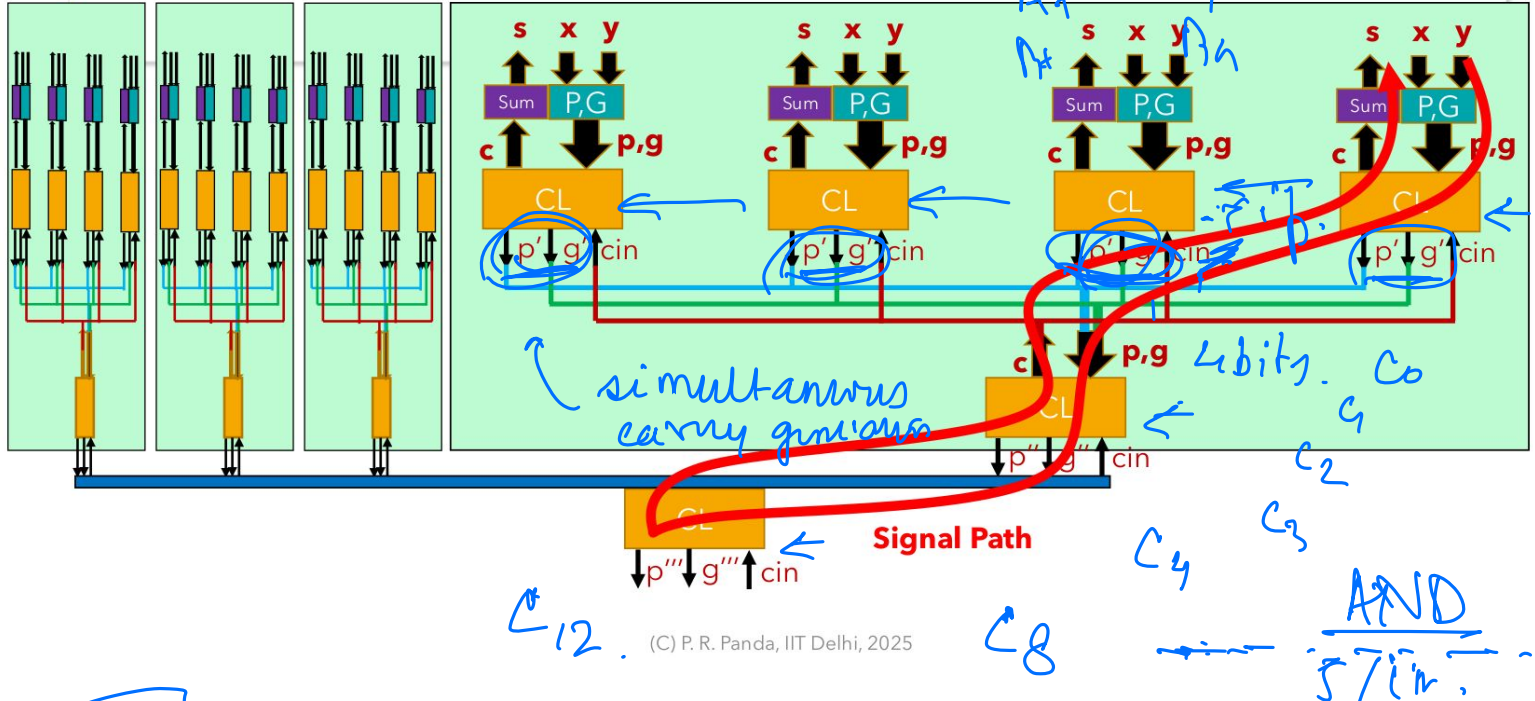
Signal Path

Signals traverse up/down, not left/right

$$A_3, A_2, A_1, A_0 \quad | \quad A_3, A_2, A_1, A_0$$

$$(i \dots i) \quad | \quad B_3, B_2, B_1, B_0$$

64-bit Adder from 16-bit CLAs



CLA

Ripple carry

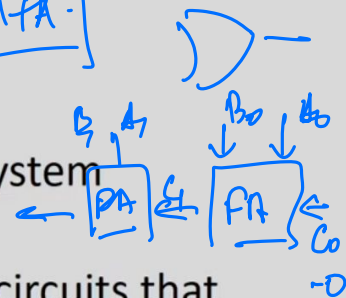
Decimal adder

FA

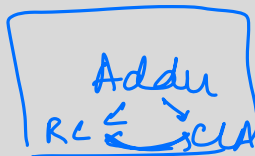
HA

1FA

C_{in} A_i B_i



- Arithmetic operations directly in the decimal number system represent decimal numbers in binary coded form
- An adder for such a computer must employ arithmetic circuits that accept coded decimal numbers and present results in the same code.
- A decimal adder requires a minimum of nine inputs and five outputs, since four bits are required to code each decimal digit and the circuit must have an input and output carry
- Depends on the code used.



BCD adder

- Numbers range 0-9.
- Adder results: 0-19 (including input carry)

$$K + 2s_8 + 2s_4 + 2s_2 + 2s_1$$

adder →

Derivation of BCD Adder

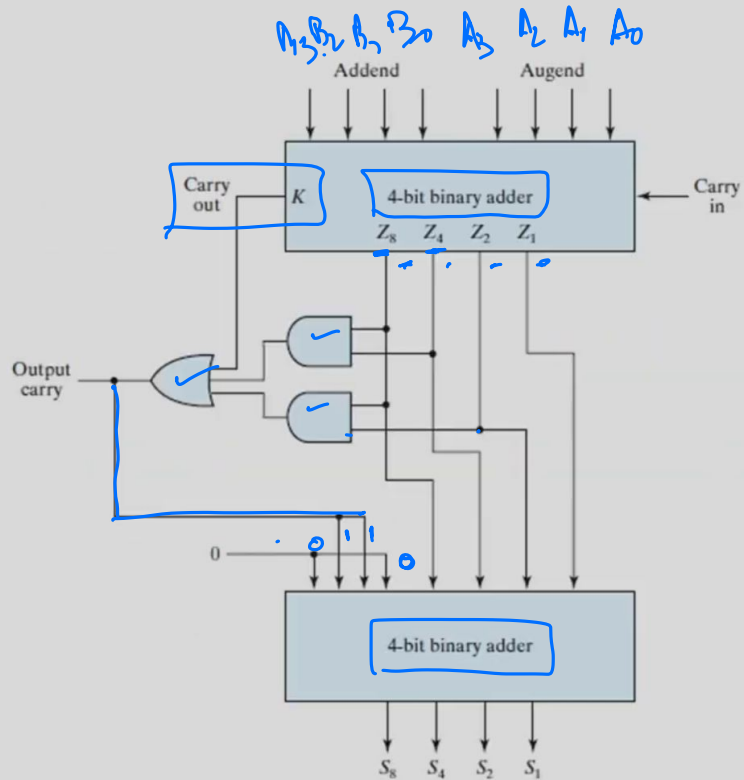
K	Binary Sum				C	BCD Sum				Decimal
	<u>Z₈</u>	Z ₄	Z ₂	Z ₁		S ₈	S ₄	S ₂	S ₁	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

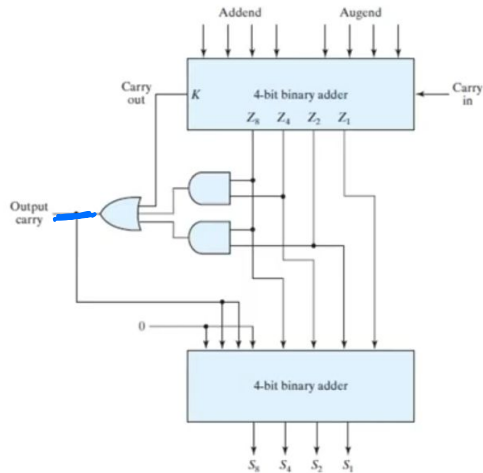
binary →
code

when to add 6 to
the normal
sum

BCD Adder

$$C = K + Z_8Z_4 + Z_8Z_2$$





```
module bcd_adder (A, B, Cin, S, Cout);
  input [3:0] A, B;
  input Cin;
  output [3:0] S;
  output Cout;
```

```
  wire carry, carry1;
  wire [3:0] sum, b;
  wire z;
```

```
  four_bit_adder Add1 (.A(A), .B(B), .C0(Cin), .S(sum), .C4(carry));
  four_bit_adder Add2 (.A(sum), .B(b), .C0(1'b0), .S(S), .C4(carry1));
  assign z = (sum[3] & sum[2]) | (sum[3] & sum[1]) | carry;
  assign b = {1'b0, z, z, 1'b0};
  assign Cout = z;
```

```
endmodule
```

↓
BCD

BCD

*9

combinational
←

Ripple carry / CLA / min

0110

```
module testbench ( );
```

```
reg [3:0] a;
```

```
reg [3:0] b;
```

```
reg cin;
```

```
wire [3:0] sum;
```

```
wire cout;
```

```
bcd_adder DUT(
```

```
.A(a),
```

```
.B(b),
```

```
.Cin(cin),
```

```
.S(sum),
```

```
.Cout(cout)
```

```
);
```

```
reg clk;
```

```
reg rst;
```

```
// Initial values
```

```
initial begin
```

```
clk = 0;
```

```
rst = 1;
```

```
#20;
```

```
rst = 0;
```

```
end
```

```
// Clock generator
```

```
always begin
```

```
#10 clk = ~clk;
```

```
end
```

```
// Test sequence
```

```
reg [3:0] i;
```

```
always @(posedge clk, posedge rst) begin
```

```
if (rst) begin
```

```
i = 0;
```

```
end else begin
```

```
// Apply all possible val values (0-9)
```

```
a <= $urandom%10;
```

```
b <= $urandom%10;
```

```
cin <= $random%2;
```

```
#20;
```

```
i = i + 1;
```

```
if (i == 10) $finish;
```

```
end
```

```
$monitor("%d: a = %b, b = %b, cin = %b, sum = %b, cout = %b",
```

```
$time, a, b, cin, sum, cout);
```

```
end
```

```
endmodule
```

www.youtube.com – to exit full screen, press Esc

10: a = xxxx, b = xxxx, cin = x, sum = xxxx, cout = x

30: a = 0110, b = 1001, cin = 0, sum = 0101, cout = 1

50: a = 1001, b = 1001, cin = 1, sum = 1001, cout = 1

70: a = 0101, b = 0101, cin = 1, sum = 0001, cout = 1

90: a = 0111, b = 0100, cin = 1, sum = 0010, cout = 1

110: a = 1001, b = 0111, cin = 1, sum = 0111, cout = 1

130: a = 0110, b = 0101, cin = 1, sum = 0010, cout = 1

150: a = 0111, b = 0000, cin = 1, sum = 1000, cout = 0

170: a = 0111, b = 1000, cin = 0, sum = 0101, cout = 1

190: a = 1001, b = 0100, cin = 1, sum = 0100, cout = 1

210: a = 0011, b = 0111, cin = 1, sum = 0001, cout = 1

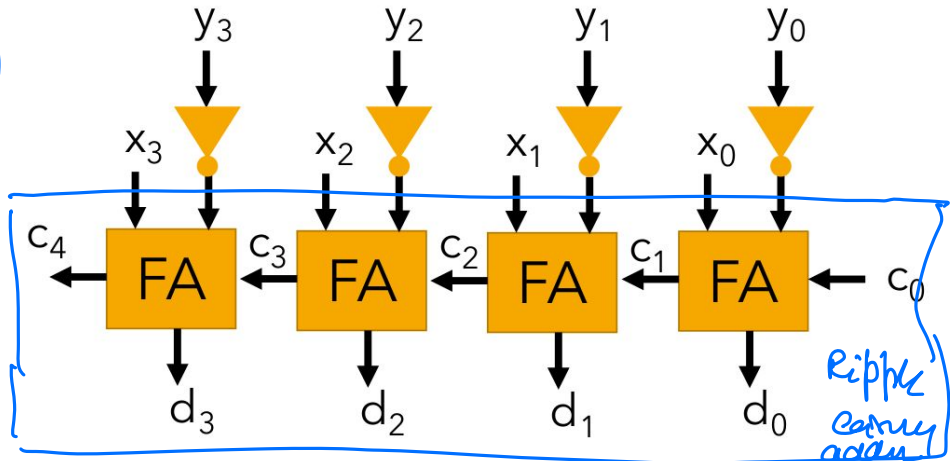
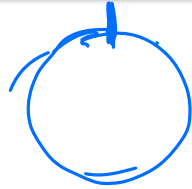
7 + 8 = 15
1111 0x

n-bit Subtraction

$$2^n - y$$

→ 1's complement bit flip + 1

- $d = x - y$
- $d = x + (-y)$
- $-y$: 2's complement of y
- $-y$: $y' + 1$
- y' : inverter
- How do we add 1?



smaller chip

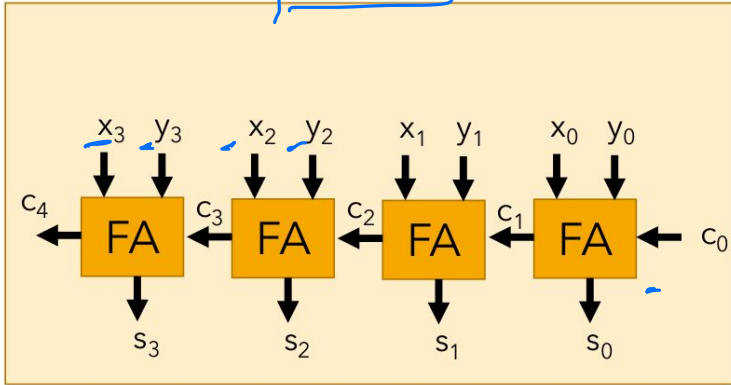
Prop.

Add

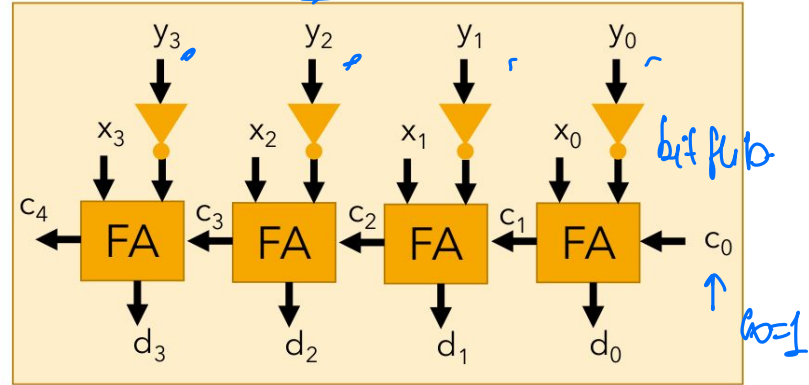
Subtr

Programmable Adder/Subtractor

Adder

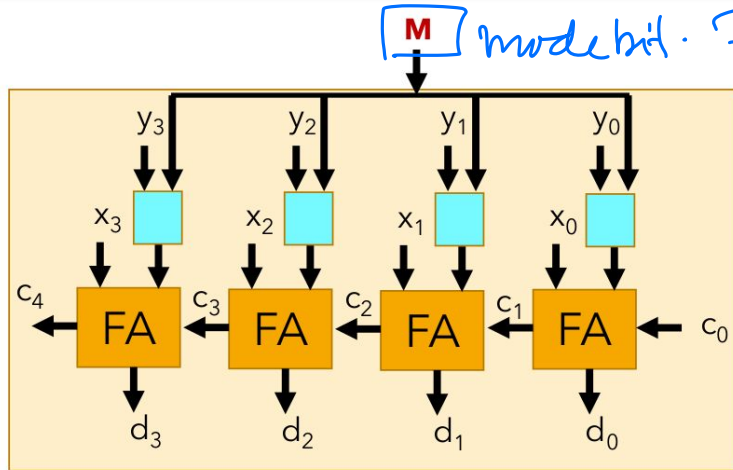


Subtractor



Very similar!
Can we combine into one structure?

Programmable Adder/Subtractor



$M = 0$: Add
 $M = 1$: Subtract



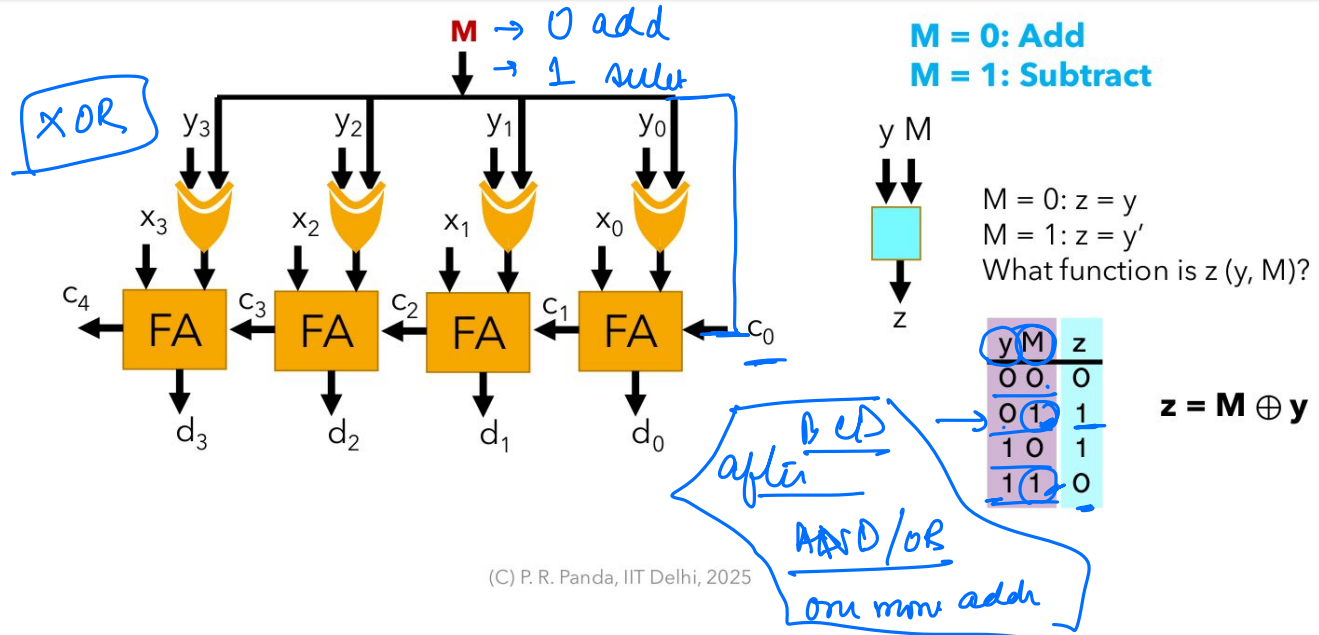
$M = 0$: $z = y$
 $M = 1$: $z = y'$
What function is $z(y, M)$?

XOR

$a \text{ XOR } 0 \rightarrow a$

$a \text{ XOR } 1 \rightarrow \sim a$

Programmable Adder/Subtractor

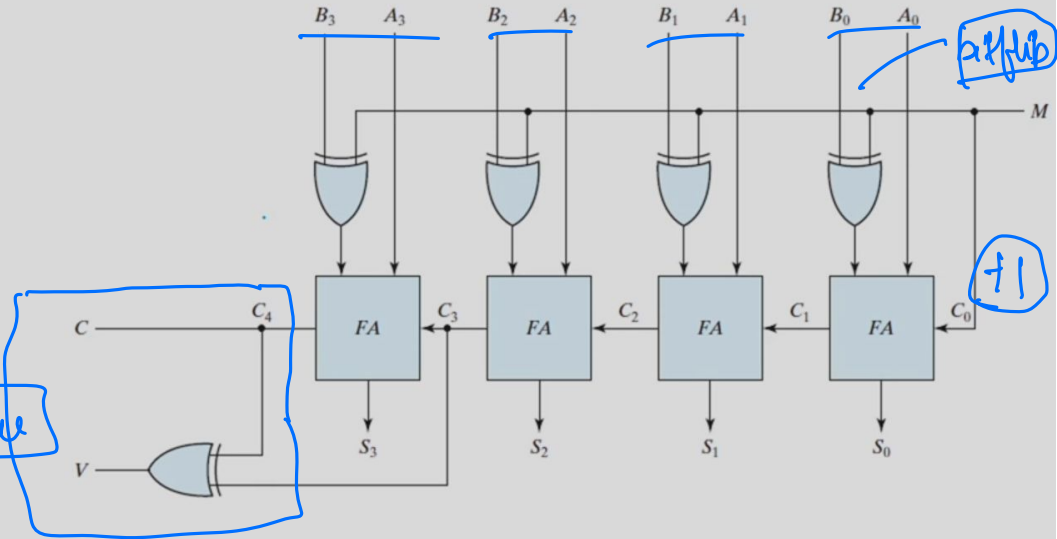


Full Adder/Subtractor

- $M=1 \rightarrow$ subtractor
- $M=0 \rightarrow$ adder

adder

overflow



Subtraction of Unsigned Numbers

- Add M to the r 's complement of N . Mathematically, $M + (r^n - N) = M - N + r^n$
- If $M \geq N$, the end carry r^n will be discarded.
- If $M < N$, there will no end carry. Result will be $r^n - (N - M)$. To obtain the answer in a familiar form, take the r 's complement of the result and place a negative sign in front.
- Inverters placed between each data input B and the input carry $C_0 = 1$
 - XOR with 1 for each bits of B

Full Adder/Subtractor

Unsigned number:

- $A \geq B$: $2^n + (A-B) \rightarrow 2^n$ will be discarded.
- $A < B$: $2^n - (B-A) \rightarrow 2$'s complement of $B-A$

Signed number:

$$(\pm A) - (+B) = (\pm A) + (-B);$$

$$(\pm A) - (-B) = (\pm A) + (+B).$$

- Binary numbers in the signed-complement system are added and subtracted by the same basic addition and subtraction rules as are unsigned numbers.
- Therefore, computers need only one common hardware circuit to handle both types of arithmetic.

unsigned $\rightarrow$ variable

signed 0

-2^{n-1} 0

$-2^{n-1}-1$

-8 0

7.

(1 bit reserved for sign)

5 bit represents
4 bits

sign

2's complement if negative
number if positive

Overflow

- Two numbers with n digits each are added and the sum is a number occupying $n + 1$ digits
 - a result that contains $n + 1$ bits cannot be accommodated by an n -bit word.
 - Overflow occurs
- For unsigned number:
 - two positive numbers are added.
 - reflected by the carry out bits

Overflow for signed numbers

- The leftmost bit always represents the sign
- negative numbers are in 2's-complement form.
- When two signed numbers are added, the sign bit is treated as part of the number.
- the end carry does not indicate an overflow.
- overflow cannot occur after an addition if one number is positive and the other is negative.

Examples

5 bits

1 bit for sign

Sign bit

5, 12

0

0101

17

0

1100

0001

overflow

→

4 bit adder
1 bit sign

1 bit
31 bit
32 bit

signed complement.

-5, -12

$\boxed{-5} \quad 1 \quad 10 \quad 11$

adder

Our flow

- 17

$$\begin{array}{r} 0101 \\ 1010 \\ 1100 \\ 0011 \\ \hline 0100 \end{array}$$

Examples

5,3

$$\begin{array}{r} 0 \quad 0101 \\ 0 \quad 0011 \\ \hline 0 \quad 1000 \end{array} \rightarrow \text{correct result.}$$

adder

Examples

no overflow

-5, -3

-8

2's com.

1
←
carry.

1 1 0 1 1
1 1 1 0 1

1

0 0 0 0

→ 2's
complement.

sign bit

0 0 1 1

1 1 0 0

1 0 0 0

0 1 1 1

1

Examples

5,-3

0	0	1	0	1
1	1	1	0	1

Full Adder/Subtractor

- $M=1 \rightarrow$ subtractor
- $M=0 \rightarrow$ adder

