Lecture 14

ECE 0201: Digital Circuits and Systems

Carry-Lookahead Adders (4.5)

Signed Number Representation (1.6)

Announcements

- Assignments due Saturday Oct. 10
  - Homework 6
  - Quiz 6
  - Lab 6 report
- Start Lab 7 tomorrow (Friday Oct. 9)
  - In-Person Lab appointments on the Canvas calendar

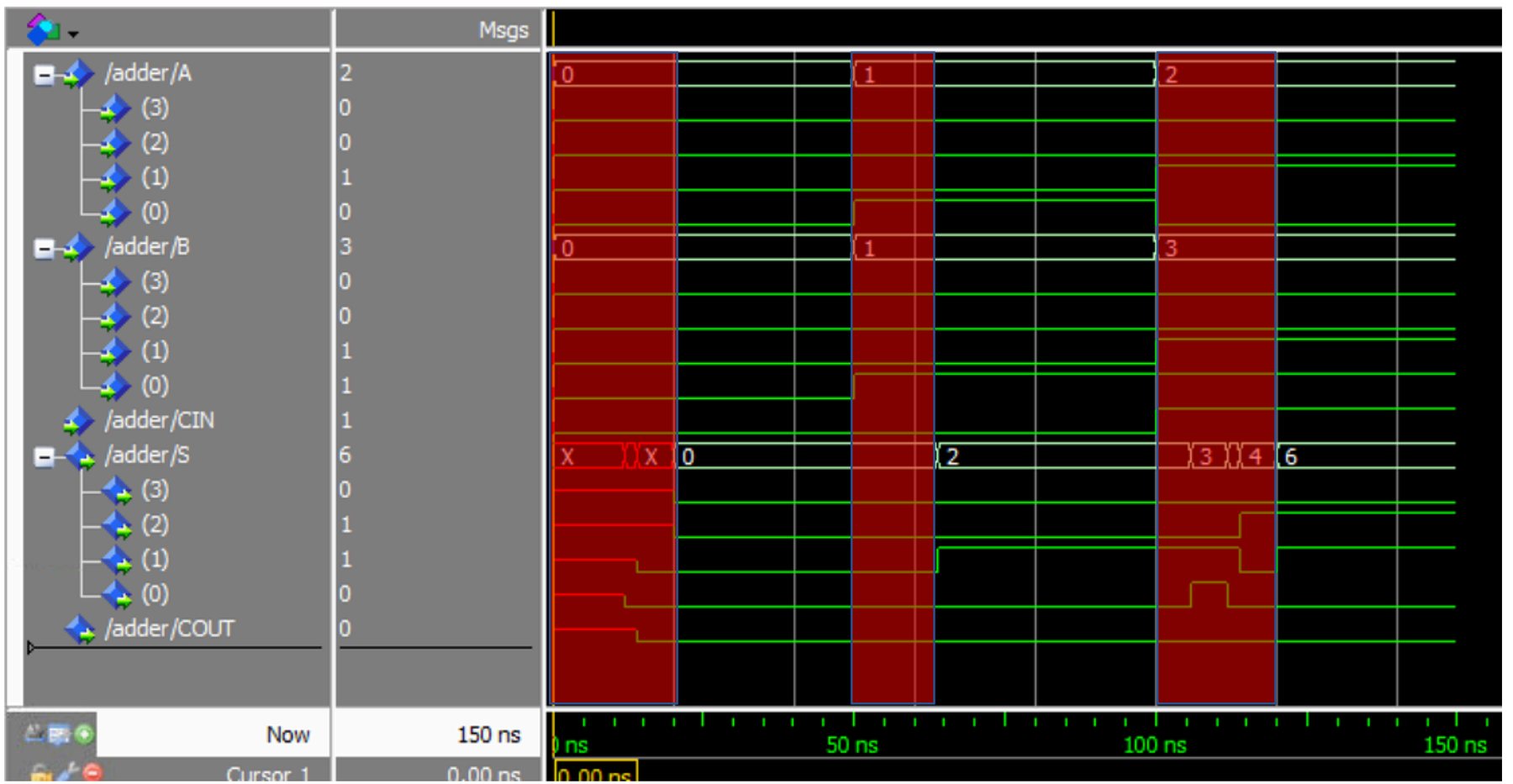
Questions for Today

How do we design digital systems?

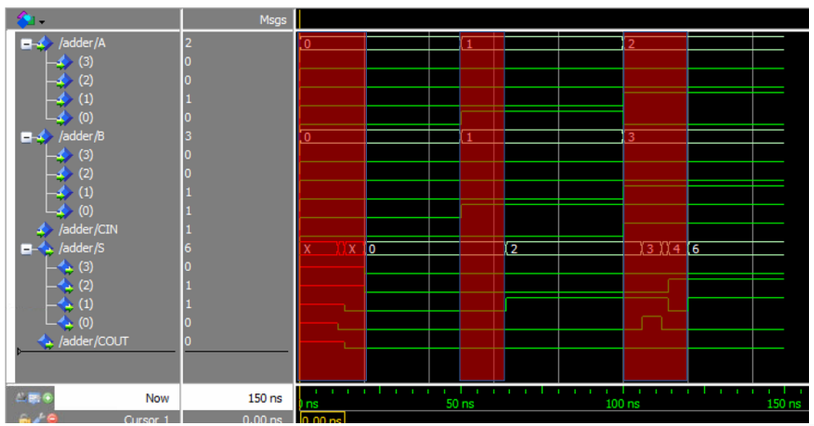
How do we use combinational logic to perform arithmetic?

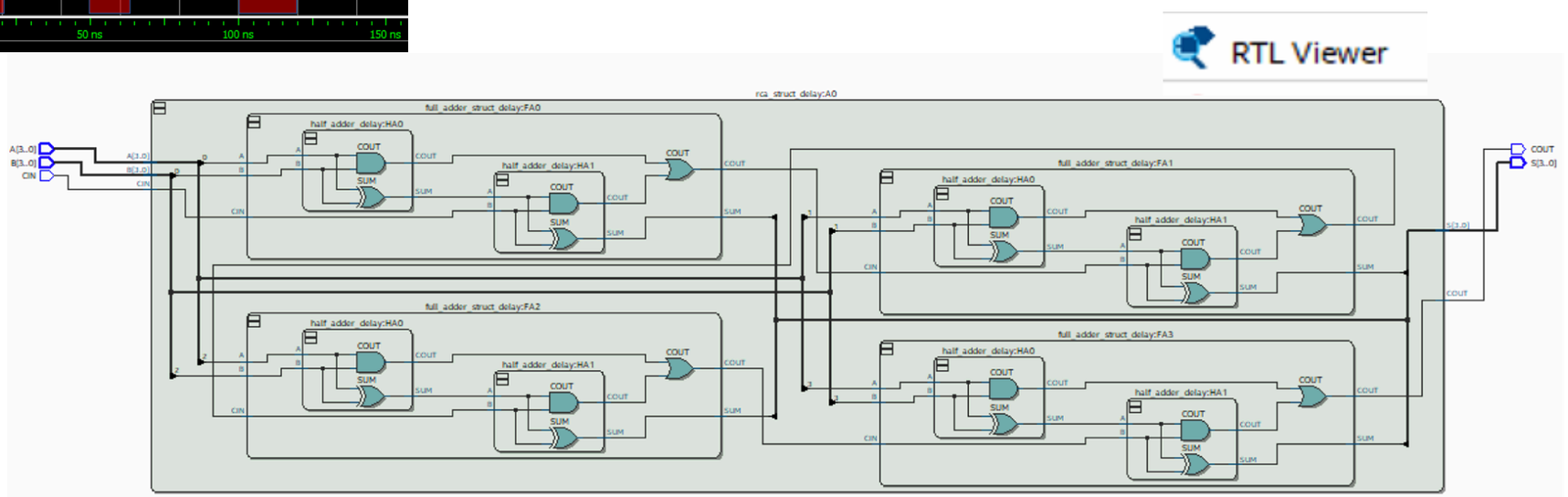
How do we measure and manage delay in digital systems?

Review: Ripple Carry Adder with Gate Delays



Review: Ripple Carry Adder with Gate Delays



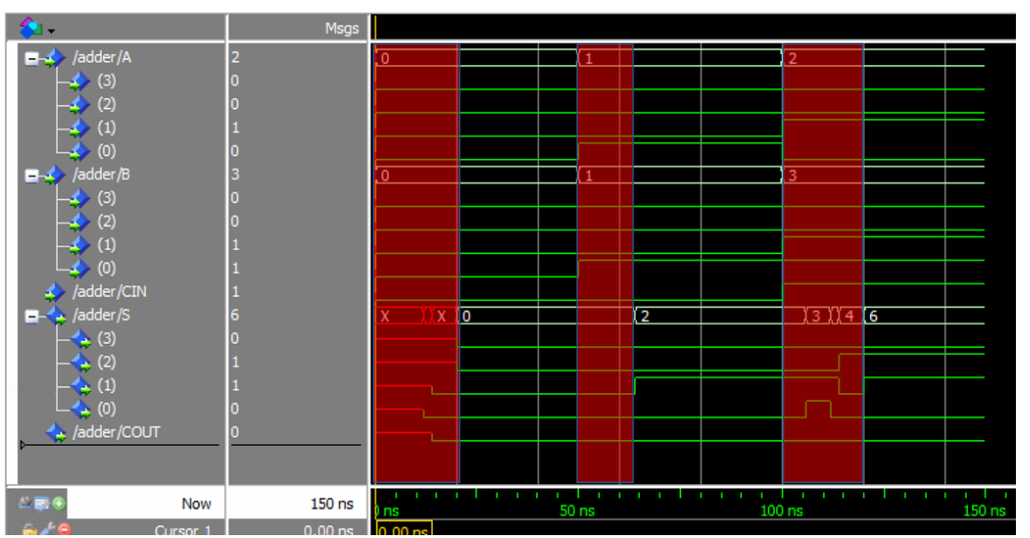


Propagation Delay

When the value of the input(s) to a circuit change, the **propagation delay** is the amount of time before the corresponding change occurs at the output.

- Delays occur due to gates and wires – we will focus on gates
- Delays can be shown in timing diagrams

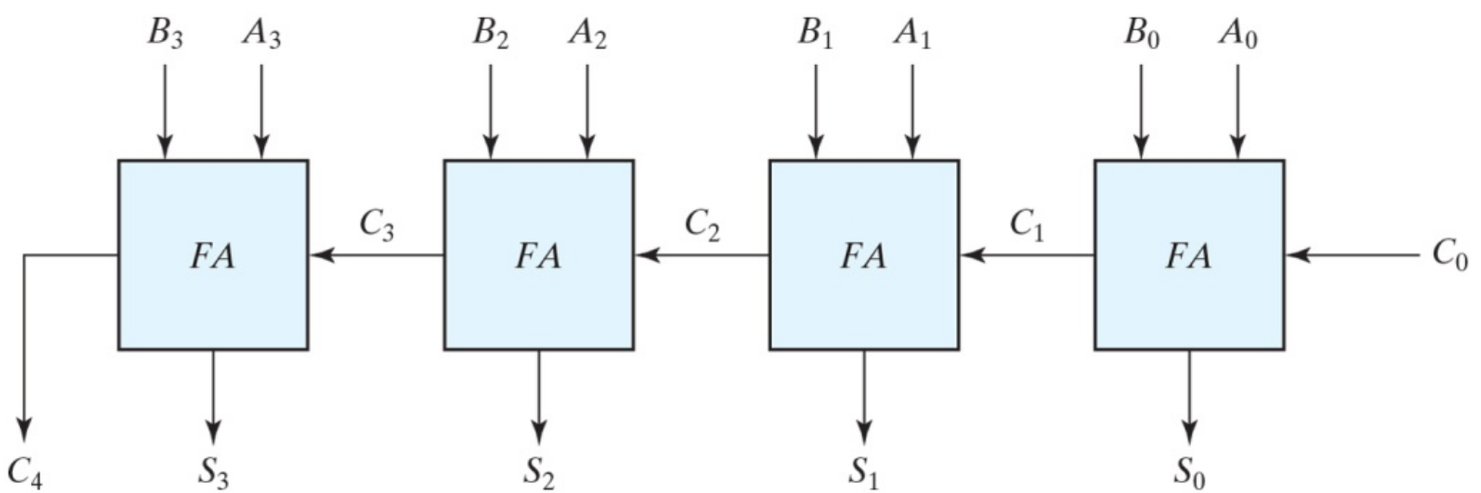
Multilevel circuits have longer propagation delays



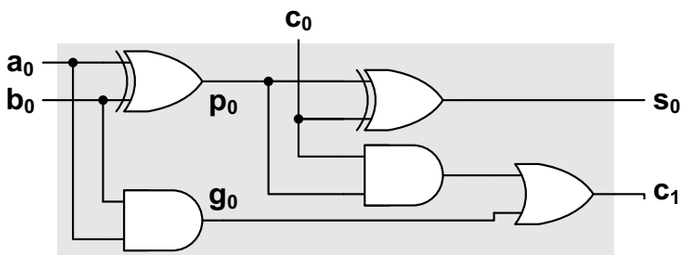
Propagation Delay

Critical path delay: the longest delay path through a circuit

The delay for a Ripple-Carry Adder increases as more bits are added.



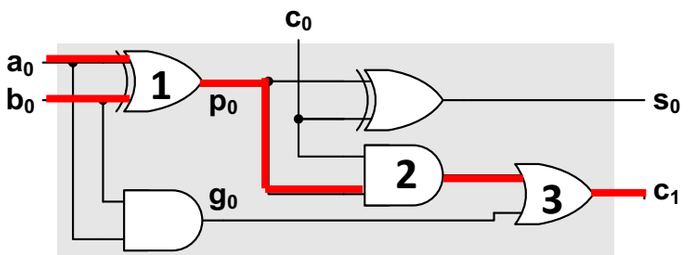
Propagation Delay



In a ripple-carry adder, the delay increases with more bits.

1 bit

Propagation Delay



In a ripple-carry adder, the delay increases with more bits.

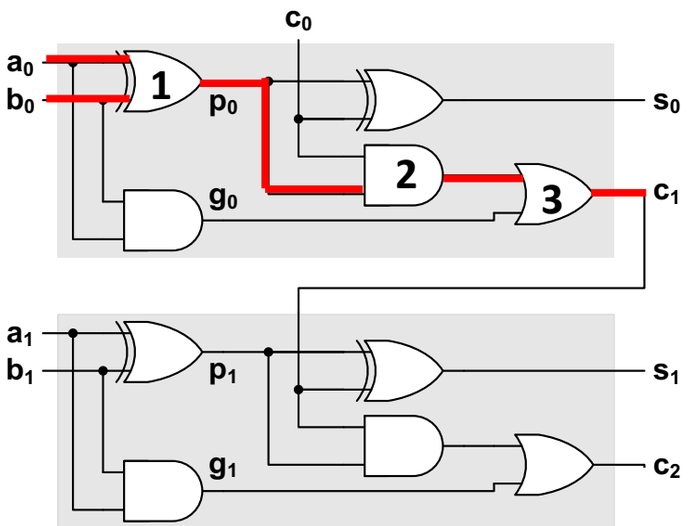
1 bit

Critical Path= 3 gates

Propagation Delay

In a ripple-carry adder, the delay increases with more bits.

2 bits

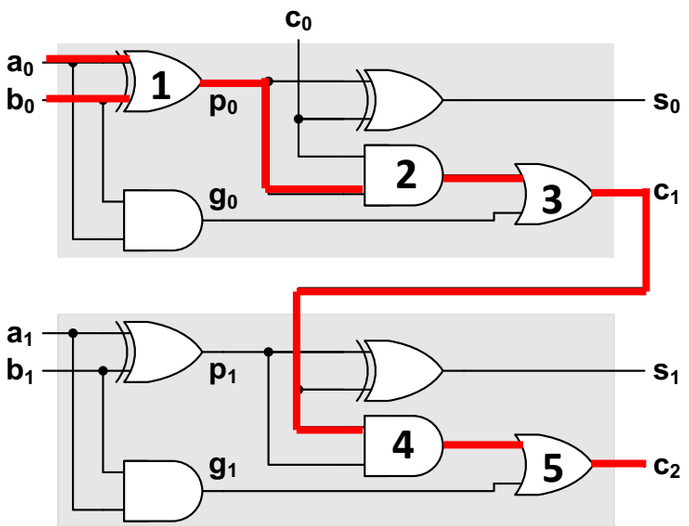


Propagation Delay

In a ripple-carry adder, the delay increases with more bits.

2 bits

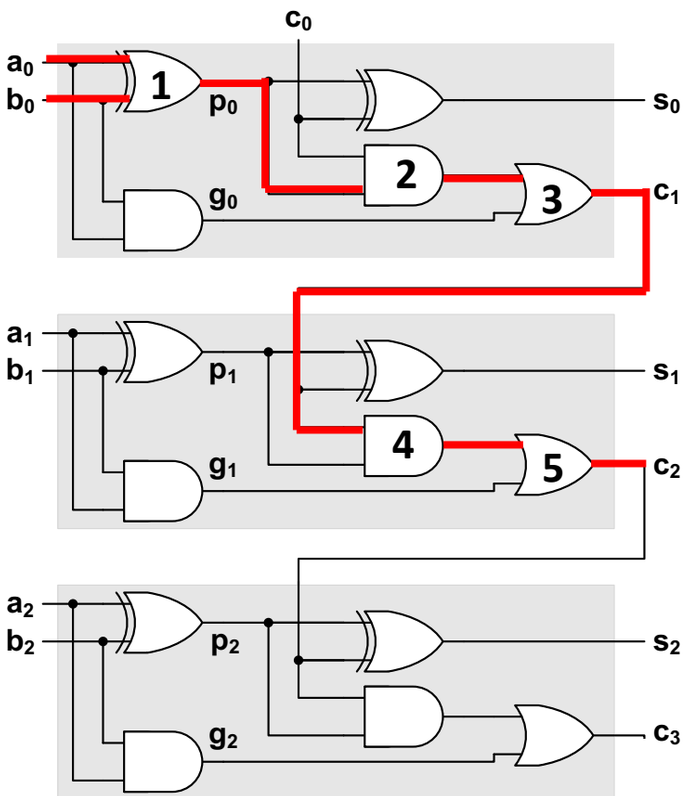
Critical Path= 5 gates



Propagation Delay

In a ripple-carry adder, the delay increases with more bits.

3 bits

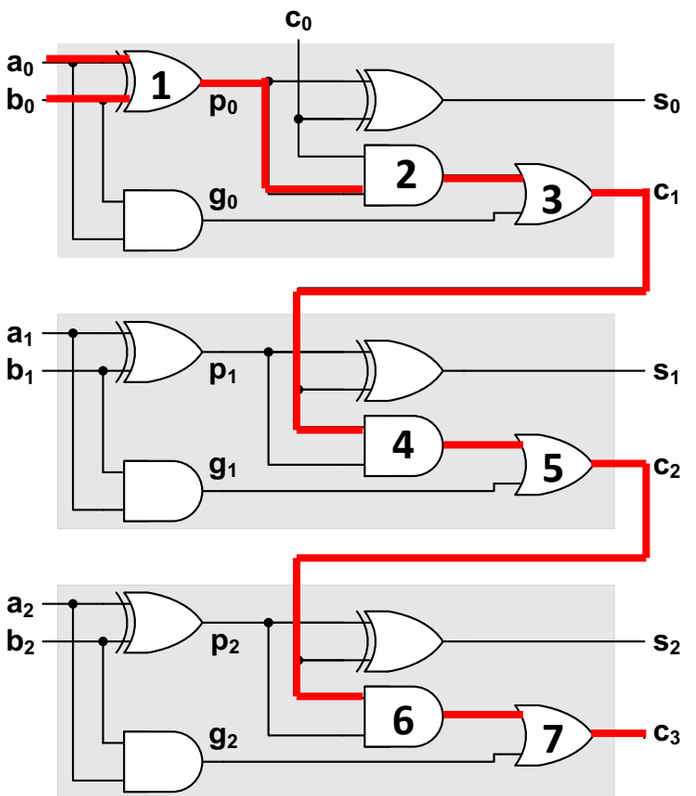


Propagation Delay

In a ripple-carry adder, the delay increases with more bits.

3 bits

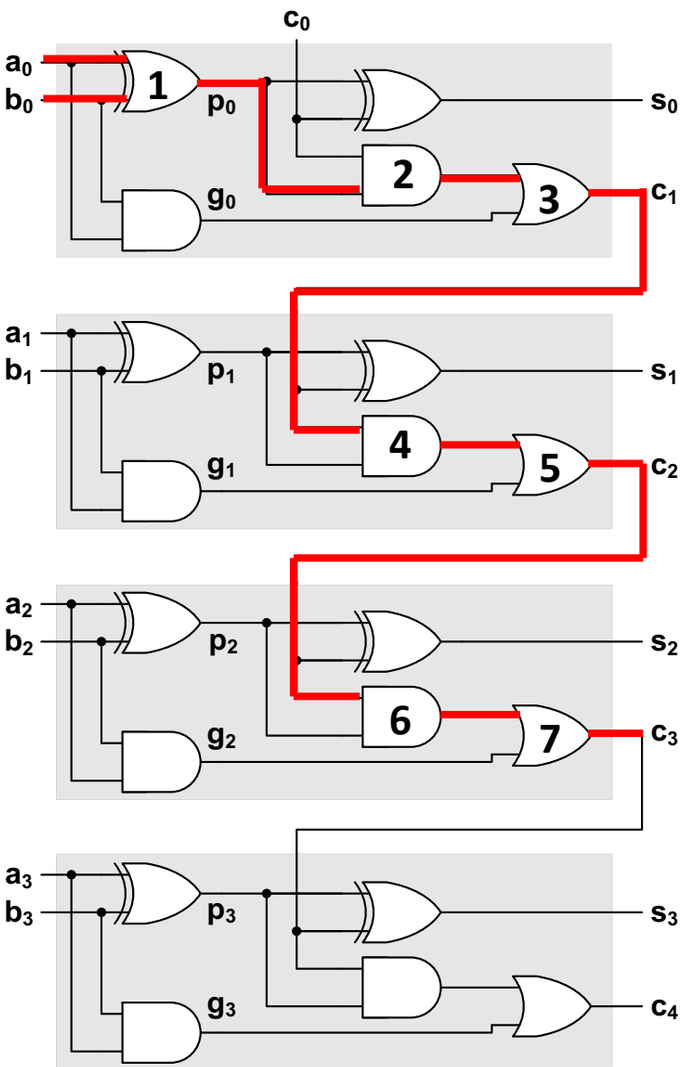
Critical Path= 7 gates



Propagation Delay

In a ripple-carry adder, the delay increases with more bits.

4 bits



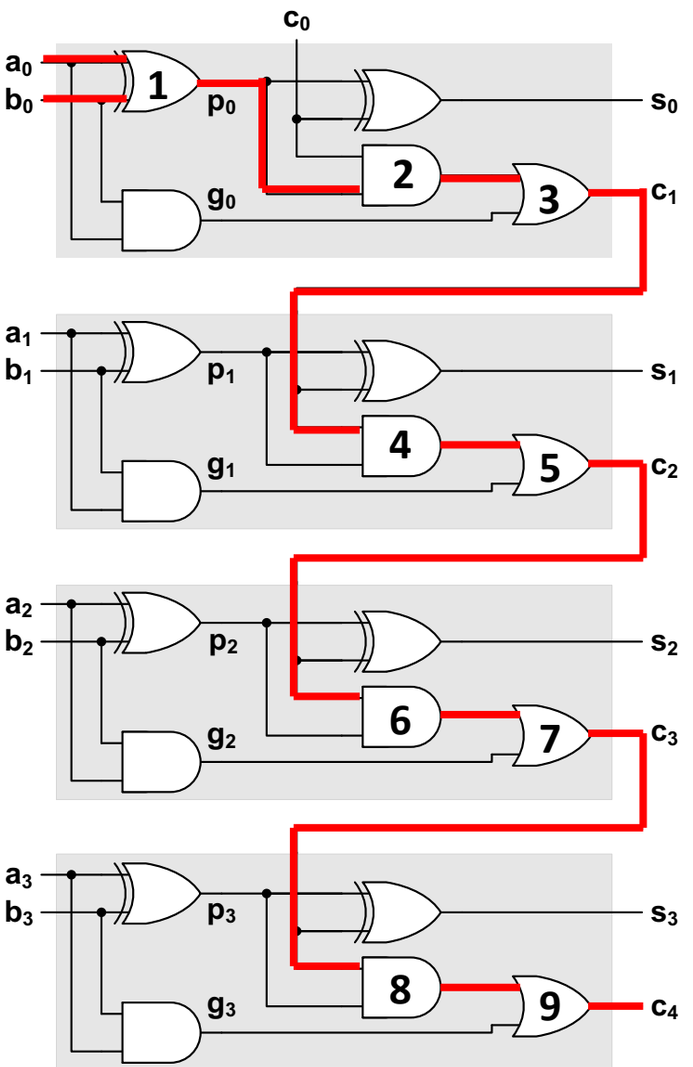
4-bit Ripple-Carry Adder

Propagation Delay

In a ripple-carry adder, the delay increases with more bits.

4 bits

Critical Path= 9 gates



4-bit Ripple-Carry Adder

Propagation Delay

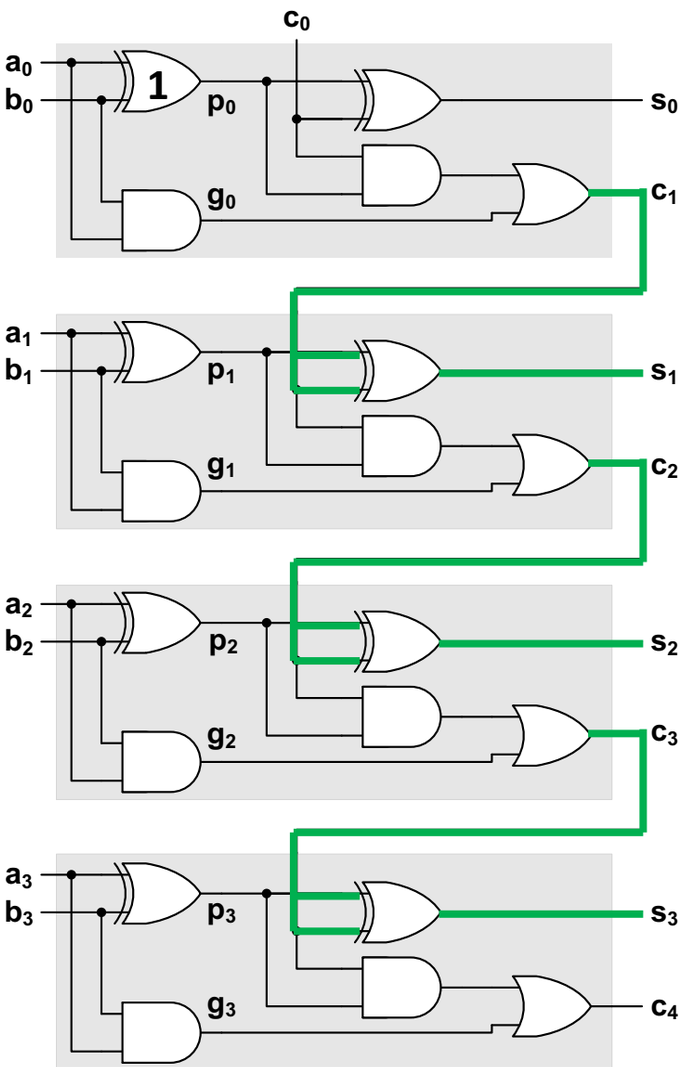
In a ripple-carry adder, the delay increases with more bits.

4 bits

Critical Path= 9 gates

Note that delays to s_1, s_2, s_3 are each one gate more than delays to c_1, c_2, c_3.

→ If we can reduce delays to carry bits, delays to sum bits will also be reduced.



4-bit Ripple-Carry Adder

Fast Adders

The delays to the carry bits can be reduced by calculating them directly from the inputs, rather than the outputs of the previous stage.

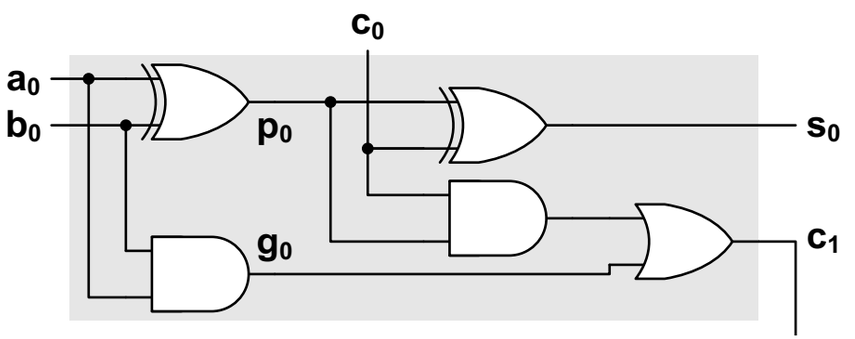
Fast adders that calculate carries in advance are commonly referred to as **Carry-Lookahead Adders.**

There are two ways to produce a carry-out bit from each adder stage: carry **generate** and carry **propagate**.

Carry Lookahead Adder

Carry Generate (g_n): If both bits of a stage are one, a carry will be **generated**

Carry Propagate (p_n): If a carry comes in to a given stage, the carry will be **propagated** to the next stage if either bit is 1



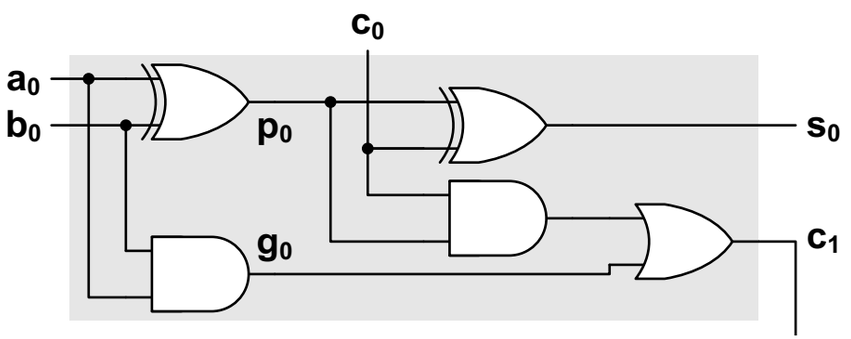
Carry Lookahead Adder

Carry Generate (g_n): If both bits of a stage are one, a carry will be **generated**

$$g_0 = a_0 b_0$$

Carry Propagate (p_n): If a carry comes in to a given stage, the carry will be **propagated** to the next stage if either bit is 1

$$p_0 = a_0 \oplus b_0$$



Carry Lookahead Adder

Carry-Generate at stage *n*:

$$g_n = a_n b_n$$

Carry-Propagate at stage *n:*

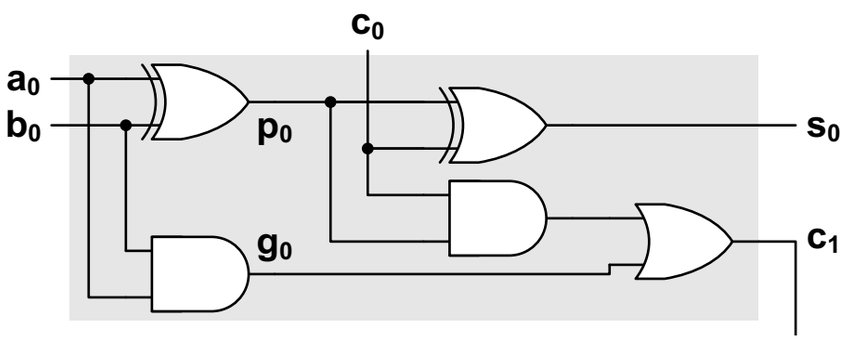
$$p_n = a_n \oplus b_n$$

Carry-Out at stage *n:*

$$c_{n+1} = g_n + p_n c_n$$

Sum at stage *n*:

$$s_n = p_n \oplus c_n$$

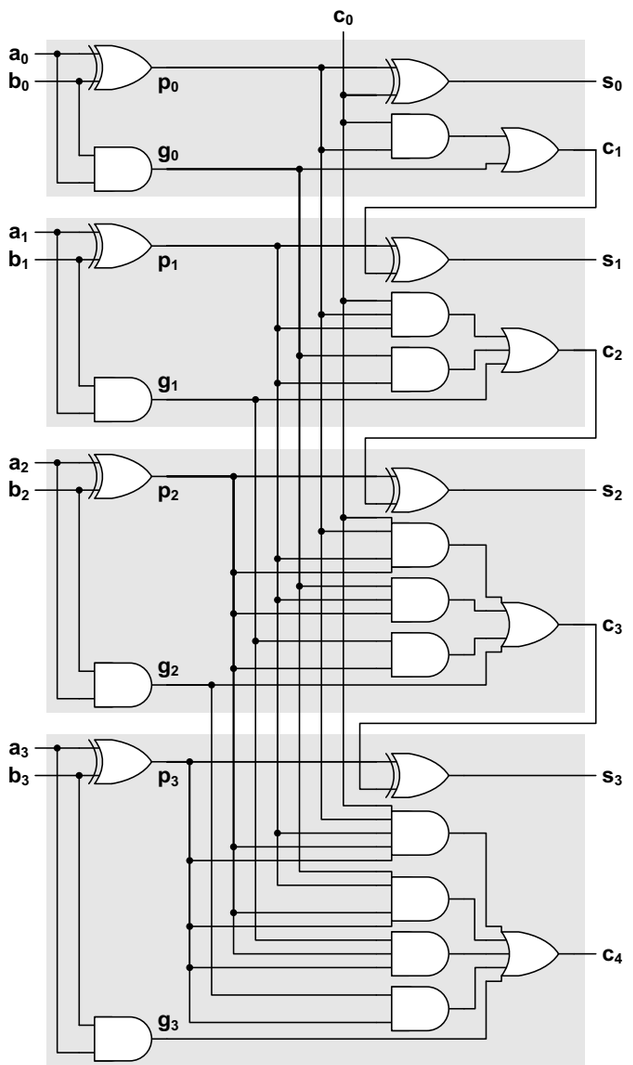


Carry Lookahead Adder

4-bit CLA carry chain:

$c_1 = g_0 + p_0 c_0$

$c_2 = g_1 + p_1 c_1$

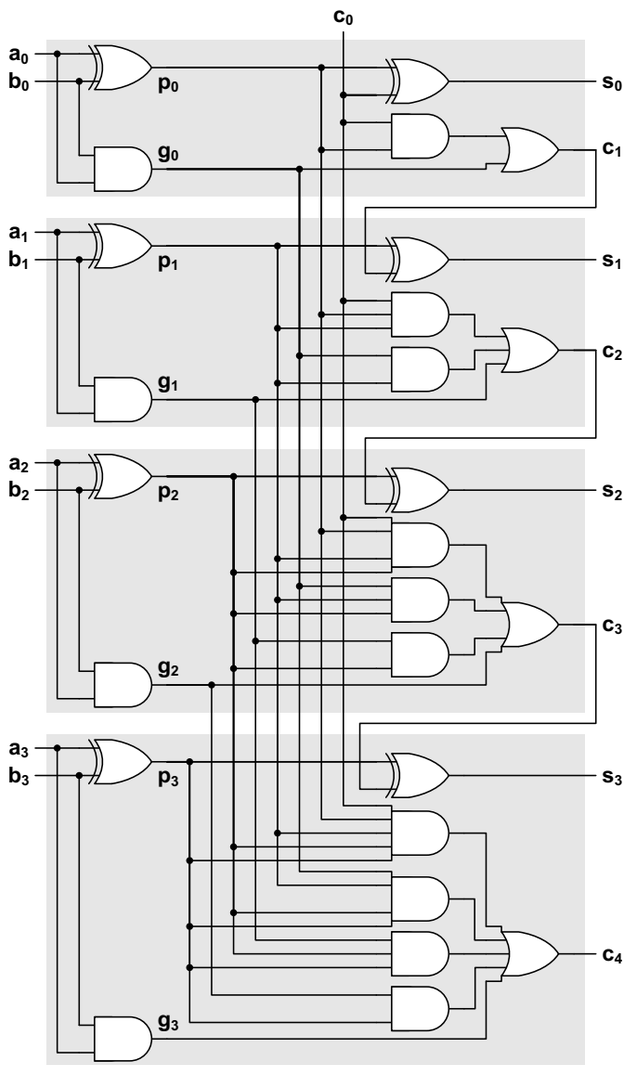


Carry Lookahead Adder

4-bit CLA carry chain:

$$c_1 = g_0 + p_0 c_0$$

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



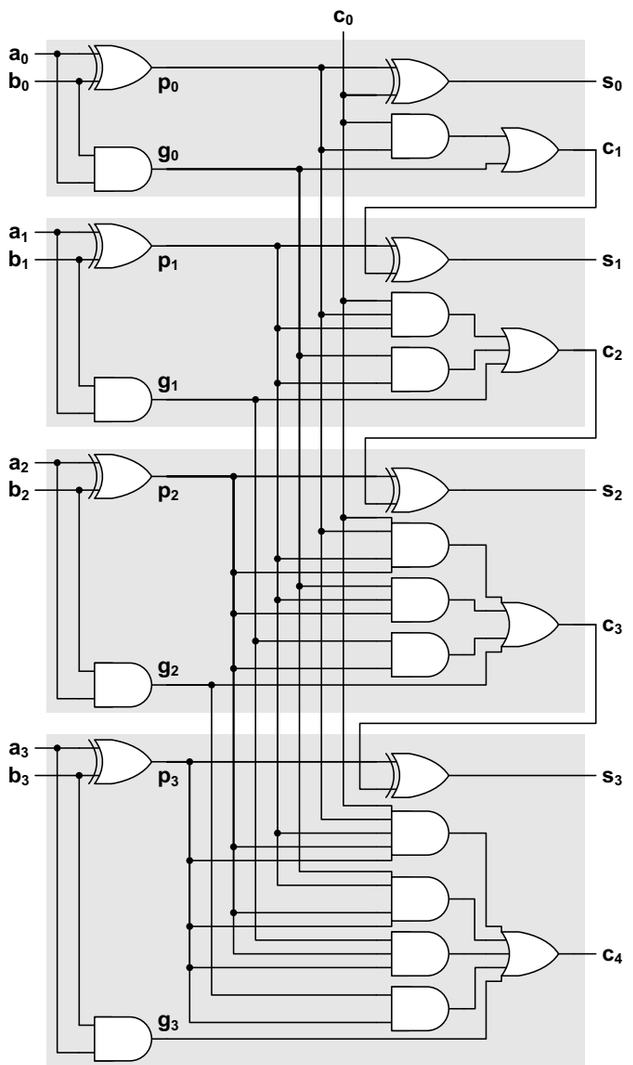
Carry Lookahead Adder

4-bit CLA carry chain:

$c_1 = g_0 + p_0 c_0$

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

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



Carry Lookahead Adder

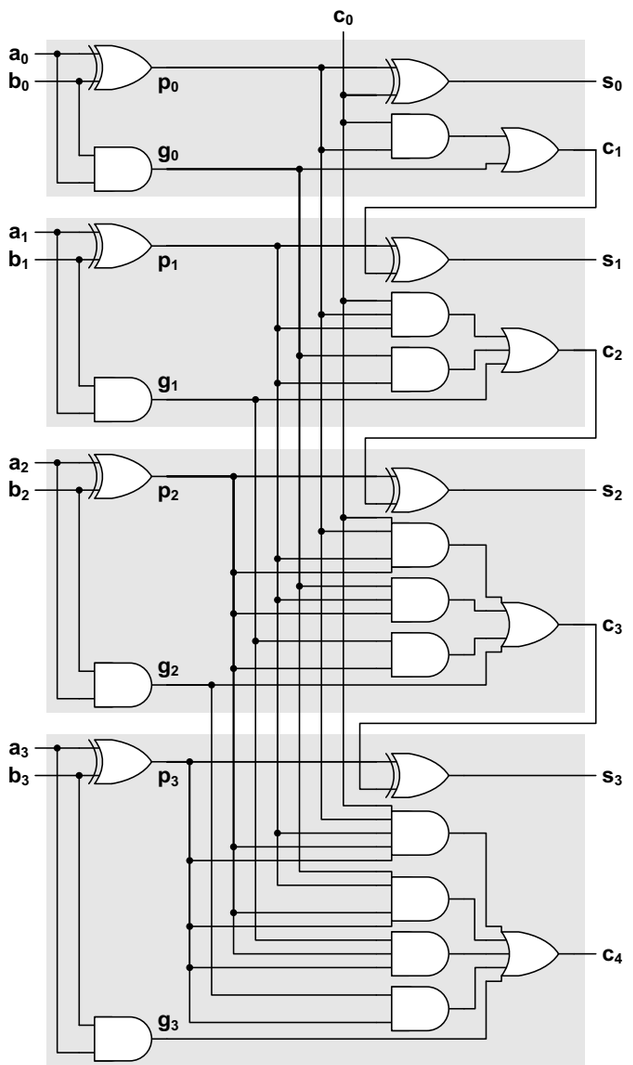
4-bit CLA carry chain:

$$c_1 = g_0 + p_0 c_0$$

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

$$\begin{aligned} c_3 &= g_2 + p_2 c_2 \\ &= g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0 \end{aligned}$$

$$\begin{aligned} c_4 &= g_3 + p_3 c_3 \\ &= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0 \end{aligned}$$



Carry Lookahead Adder

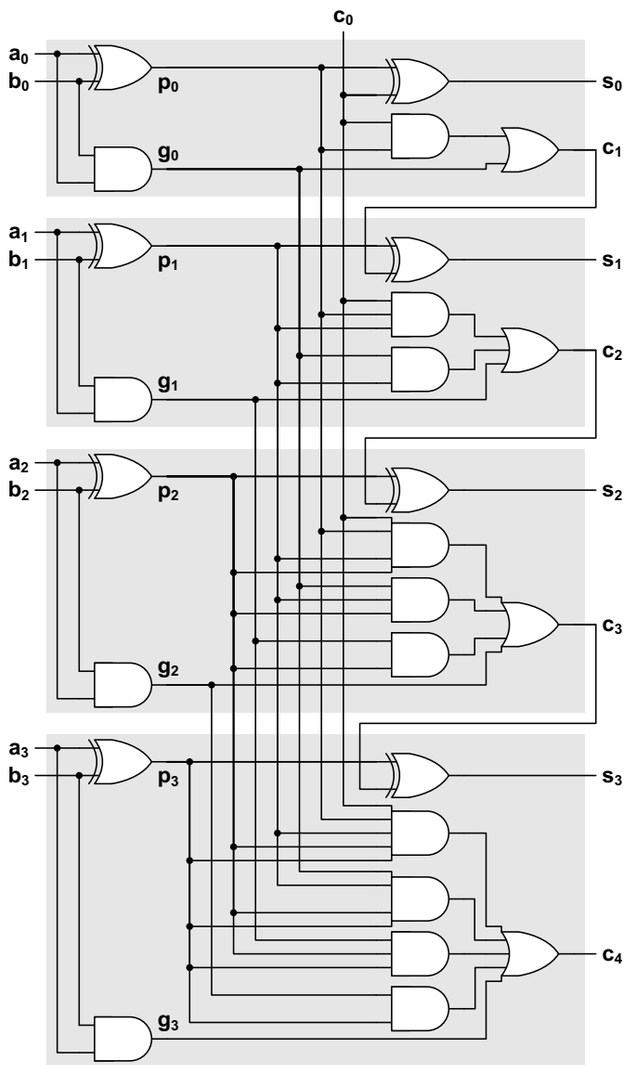
4-bit CLA carry chain:

$c_1 = g_0 + p_0 c_0$

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

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

$c_4 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0$



Carry Lookahead Adder

4-bit CLA carry chain:

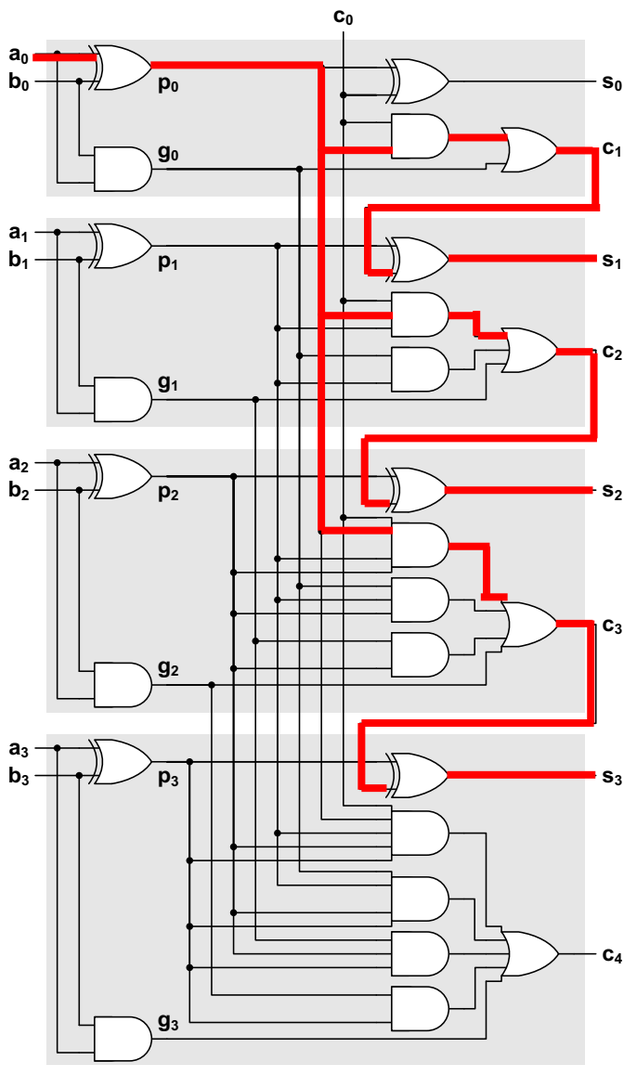
$c_1 = g_0 + p_0 c_0$

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

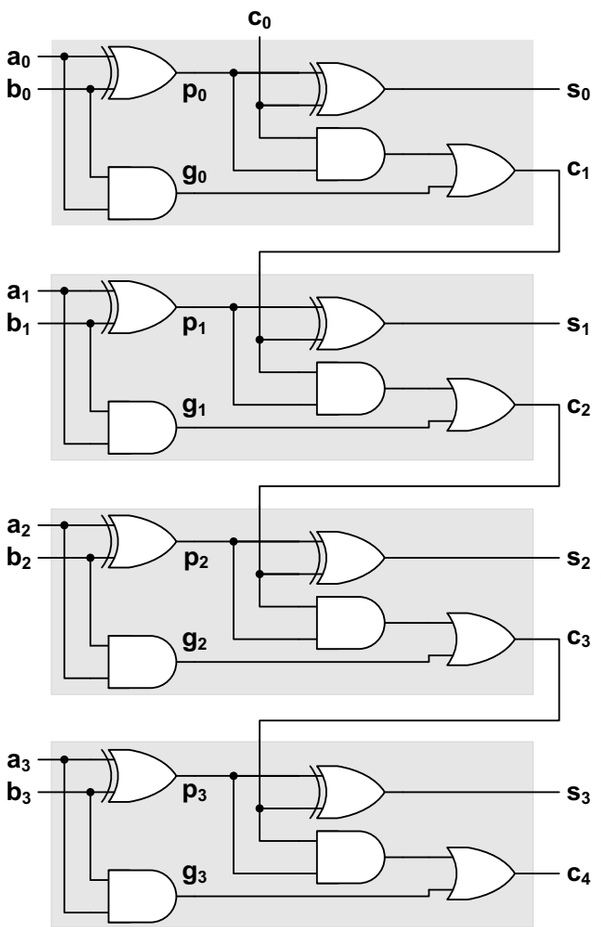
$c_3 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$

$c_4 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0$

Critical Path Delay: 4 gates, regardless of *n*



4-bit Ripple Carry Adder

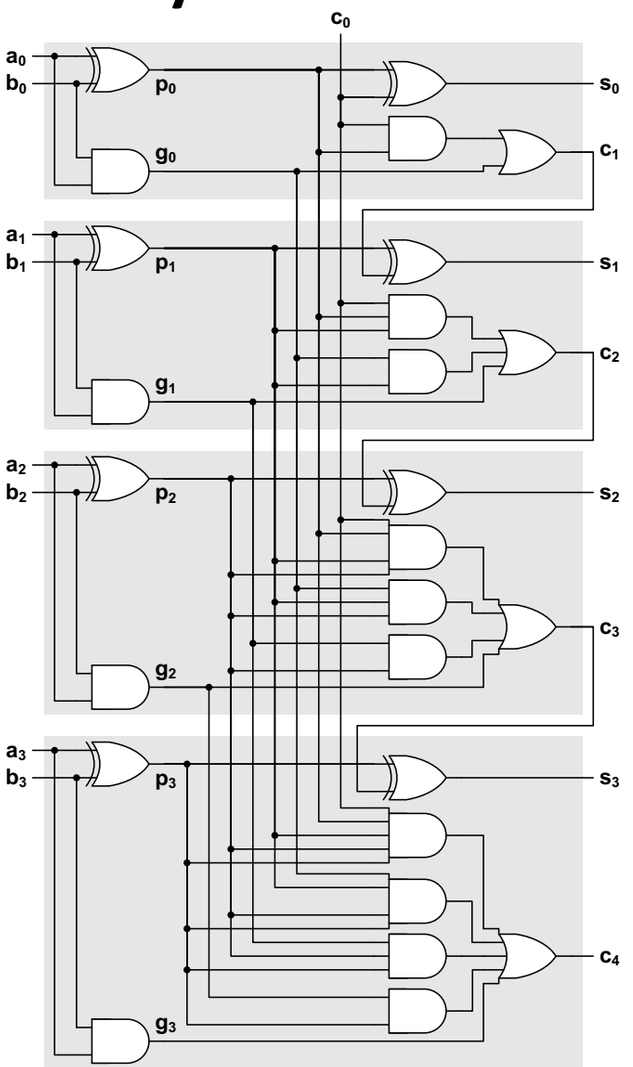


VS

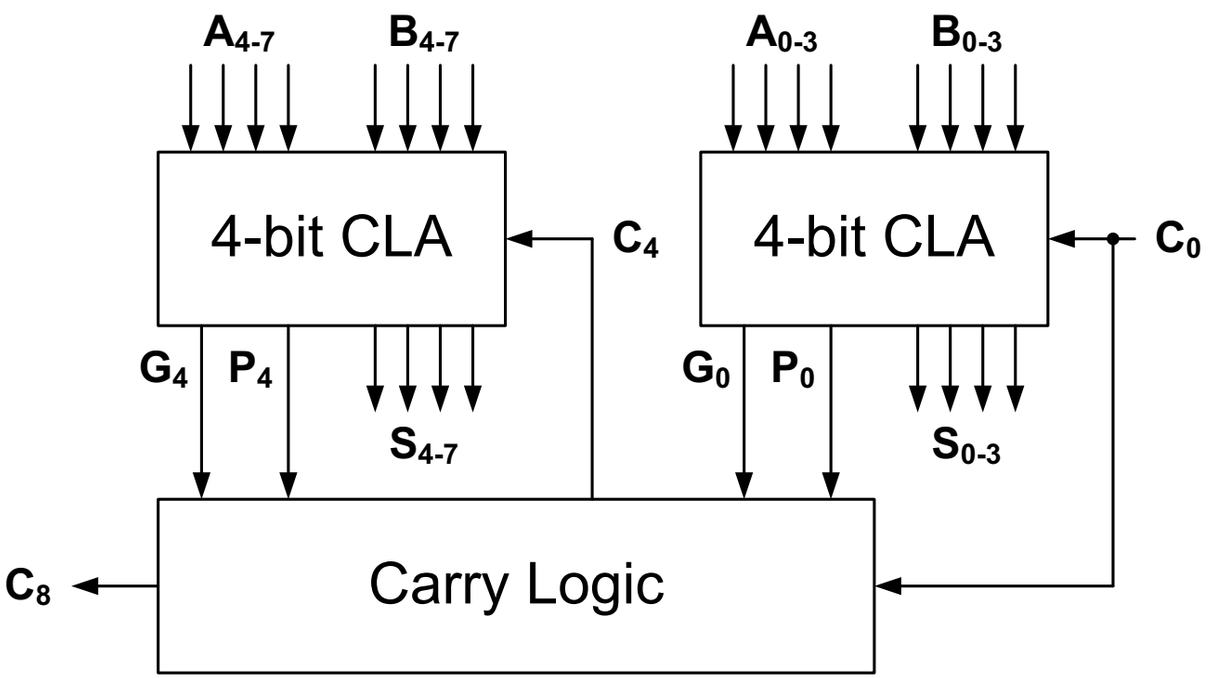
Propagation Delay

Circuit Complexity
(gates, fan-in, connections)

4-bit Carry Lookahead Adder



Cascading Carry-Lookahead Adders



Block Generate and Propagate

$c_1 = g_0 + p_0 c_0$

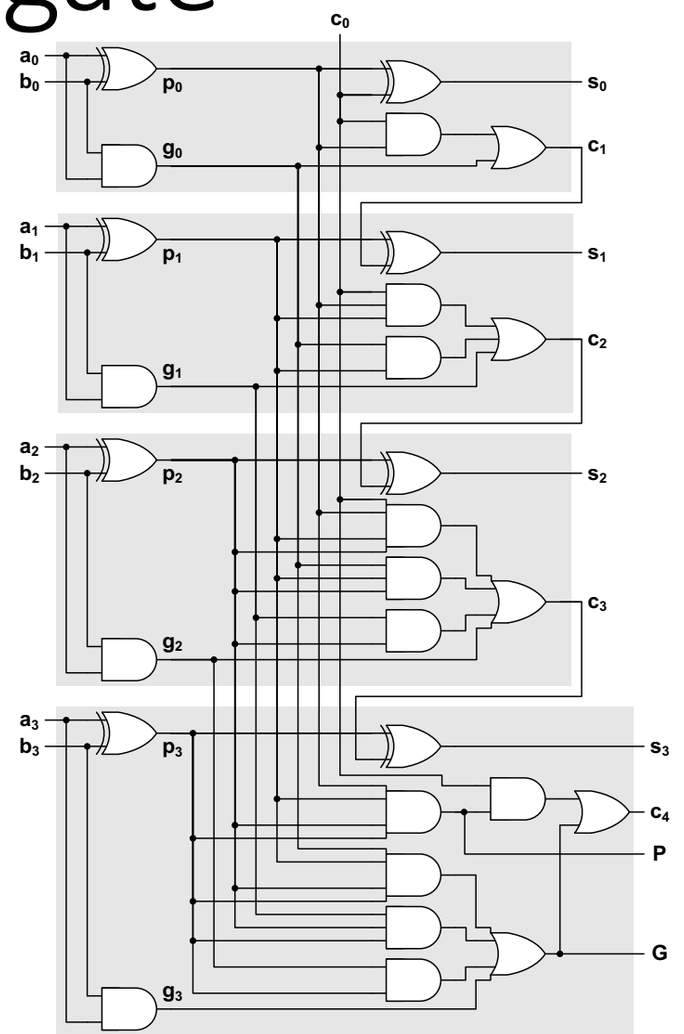
$c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0$

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

$c_4 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0$

$P = p_3 p_2 p_1 p_0$

$G = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0$



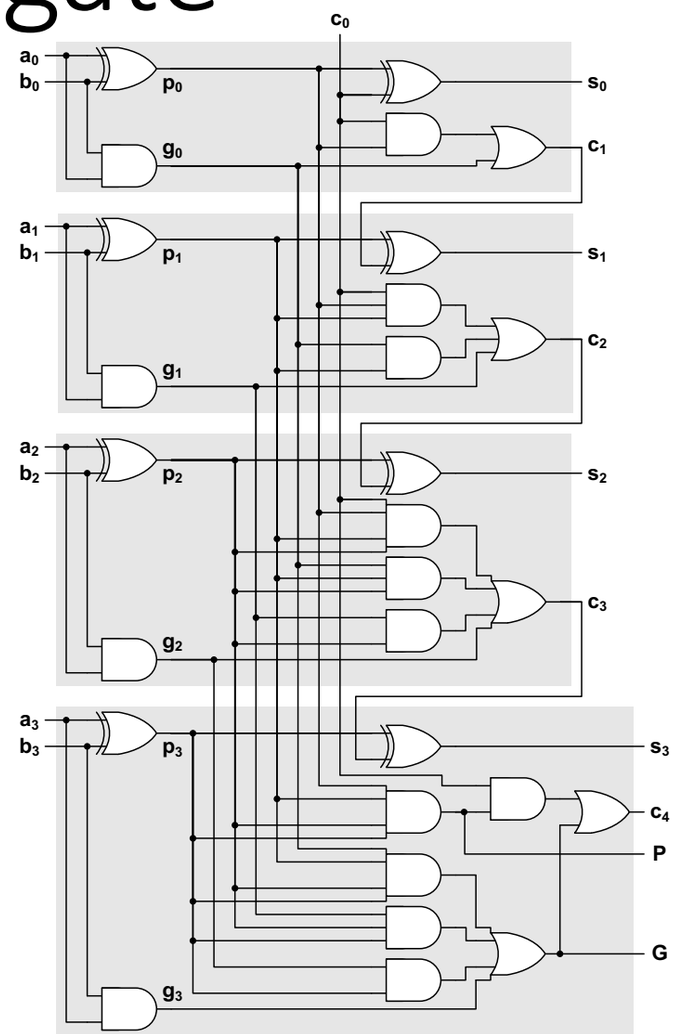
Block Generate and Propagate

$P = p_3 p_2 p_1 p_0$

→ *P* = 1 if and only if a carry input propagates through all four stages

$G = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0$

→ *G* = 1 if a carry is generated at some stage, and is then propagated through to the carry output.



Block Generate and Propagate

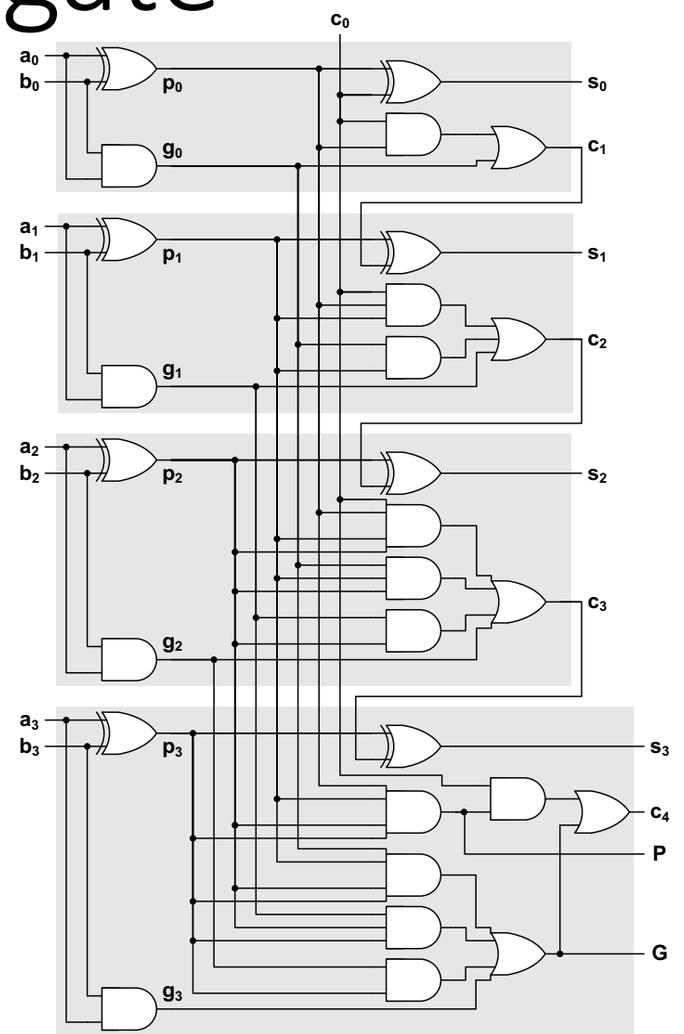
$P = p_3 p_2 p_1 p_0$

→ *P* = 1 if and only if a carry input propagates through all four stages

$G = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0$

→ *G* = 1 if a carry is generated at some stage, and is then propagated through to the carry output.

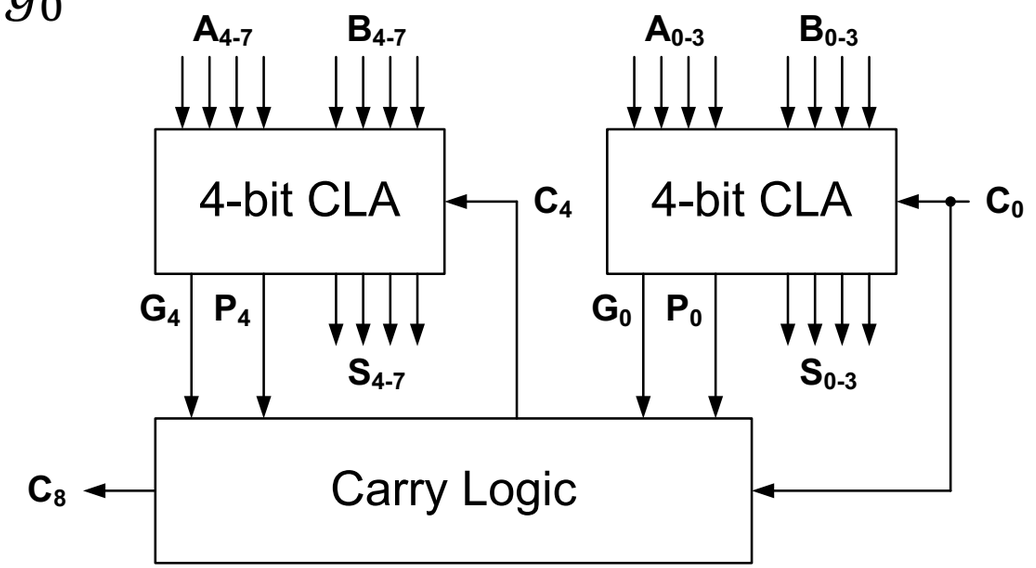
$c_4 = G + P c_0$



Cascading Carry-Lookahead Adders

$P_0 = p_3 p_2 p_1 p_0$
$G_0 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0$

$c_4 = G_0 + P_0 c_0$

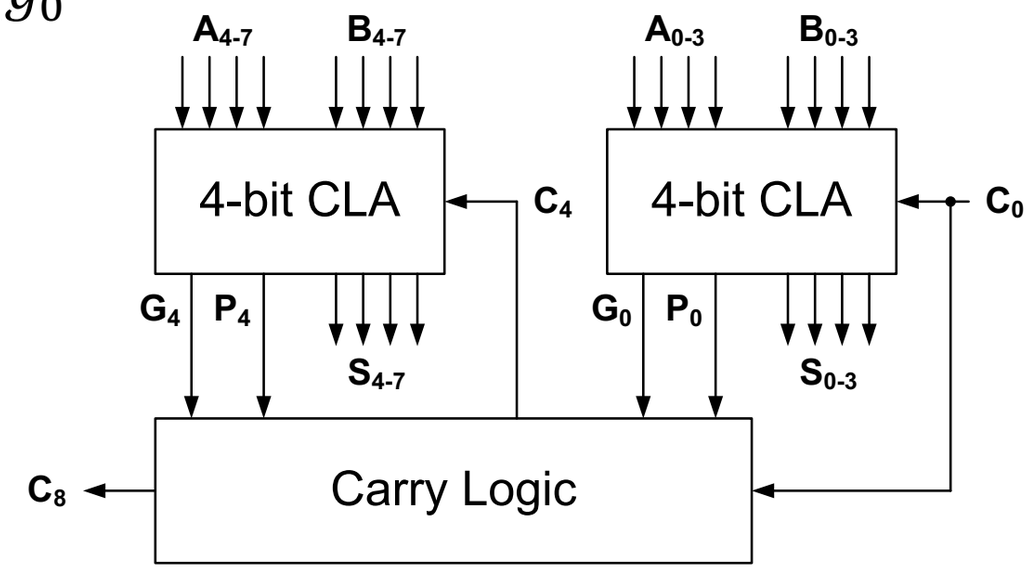


Cascading Carry-Lookahead Adders

$$P_0 = p_3 p_2 p_1 p_0$$
$$G_0 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0$$

$$c_4 = G_0 + P_0 c_0$$

$$c_8 = G_4 + P_4 c_4$$
$$= G_4 + P_4(G_0 + P_0 c_0)$$



Signed Number Representation

Signed-magnitude represents the sign of a number with the MSB; the remaining bits indicate the magnitude.

8	0000 1000
-8	1000 1000
-11	1000 1011
-25	1001 1001

Signed Number Representation

Signed-complement represents negative numbers with the complement. It is more convenient for arithmetic in computer systems.

Signed-1's-complement: complement all bits

Signed-2's-complement: complement all bits, add 1

Signed Number Representation

Signed-complement represents negative numbers with the complement. It is more convenient for arithmetic in computer systems.

Signed-1's-complement: complement all bits

Signed-2's-complement: complement all bits, add 1

-9

Signed-magnitude:
Signed-1's-complement:
Signed-2's-complement:

Signed Number Representation

Signed-complement represents negative numbers with the complement. It is more convenient for arithmetic in computer systems.

Signed-1's-complement: complement all bits

Signed-2's-complement: complement all bits, add 1

-9

Signed-magnitude:	1000 1001
Signed-1's-complement:	1111 0110
Signed-2's-complement:	1111 0111

Signed Number Representation

Signed-2's-complement is the most common representation because it enables easy addition of signed numbers

$$\begin{array}{r} 1111\ 0111 \\ +\ 0000\ 0010 \\ \hline \end{array}$$

Signed Number Representation

Signed-2's-complement is the most common representation because it enables easy addition of signed numbers

$$\begin{array}{r} 1111\ 0111 \\ +\ 0000\ 0010 \\ \hline 1111\ 1001 \end{array}$$

Signed Number Representation

Signed-2's-complement is the most common representation because it enables easy addition of signed numbers

-9	1111 0111
+ 2	+ 0000 0010
-7	1111 1001

Behavioral VHDL: 4:1 MUX

```
architecture mux4t1 of mux_4t1 is
    …
begin

end mux4t1;
```

Behavioral VHDL: 4:1 MUX

```
architecture mux4t1 of mux_4t1 is
    …
begin

end mux4t1;
```

Structural

```
M2A: mux_2t1 port map ( D0 => D0,
                D1 => D1,
                SEL => SEL(0),
                MUX_OUT => M2A_OUT );
M2B: mux_2t1 port map ( D0 => D2,
                D1 => D3,
                     SEL => SEL(0),
                MUX_OUT => M2B_OUT );
M2C: mux_2t1 port map ( D0 => M2A_OUT,
                D1 => M2B_OUT,
                SEL => SEL(1),
                MUX_OUT => MUX_OUT );
```

Data-Flow

```
with SEL select
   MUX_OUT <= A when "00",
              B when "01",
              C when "10",
              D when "11",
              (others => '0') when others;
--
```

Behavioral

```
process (SEL, A, B, C, D)
begin
   case SEL is
      when "00" => MUX_OUT <= A;
      when "01" => MUX_OUT <= B;
      when "10" => MUX_OUT <= C;
      when "11" => MUX_OUT <= D;
      when others => (others => '0');
   end case;
end process;
```

Behavioral VHDL: 4:1 MUX

```
architecture mux4t1 of mux_4t1 is
    …
begin

    

end mux4t1;
```

Structural

```
M2A: mux_2t1 port map ( D0 => D0,
                D1 => D1,
                SEL => SEL(0),
                MUX_OUT => M2A_OUT );
M2B: mux_2t1 port map ( D0 => D2,
                D1 => D3,
                     SEL => SEL(0),
                MUX_OUT => M2B_OUT );
M2C: mux_2t1 port map ( D0 => M2A_OUT,
                D1 => M2B_OUT,
                SEL => SEL(1),
                MUX_OUT => MUX_OUT );
```

Data-Flow

```
with SEL select
   MUX_OUT <= A when "00",
              B when "01",
              C when "10",
              D when "11",
              (others => '0') when others;
--
```

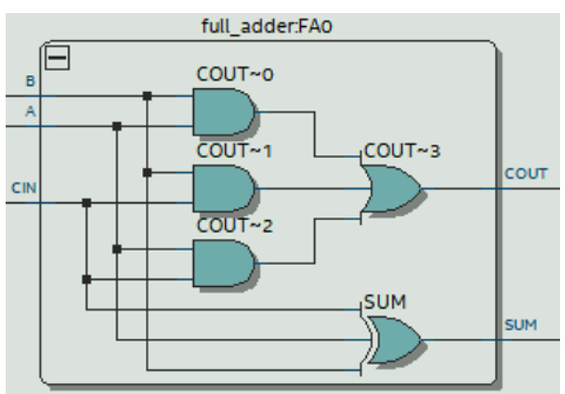
Behavioral

```
process (SEL, A, B, C, D)
begin
   case SEL is
      when "00" => MUX_OUT <= A;
      when "01" => MUX_OUT <= B;
      when "10" => MUX_OUT <= C;
      when "11" => MUX_OUT <= D;
      when others => (others => '0');
   end case;
end process;
```

Behavioral VHDL: Full Adder

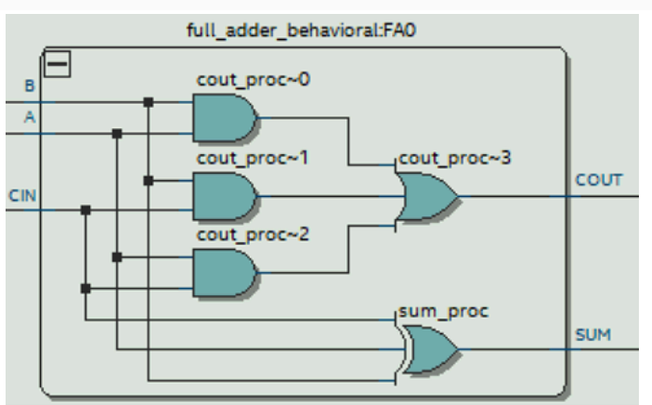
Data-Flow Model

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity full_adder is
5      port (A, B, CIN: in std_logic;
6            SUM, COUT: out std_logic);
7   end full_adder;
8
9
10  architecture full_adder of full_adder is
11  begin
12     SUM <= A xor B xor CIN;
13     COUT <= (A and B) or (B and CIN) or (CIN and A);
14  end full_adder;
15
```



Behavioral Model

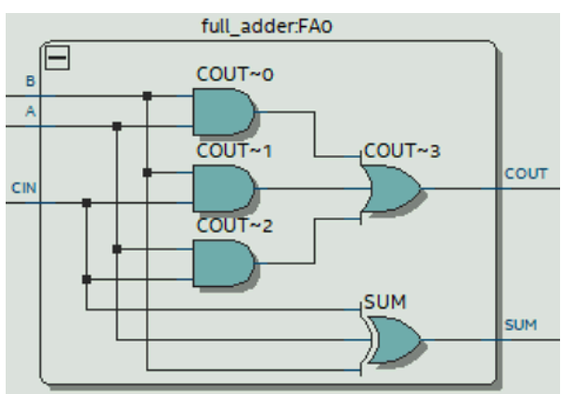
```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity full_adder_behavioral is
5      port (A, B, CIN: in std_logic;
6            SUM, COUT: out std_logic);
7   end full_adder_behavioral;
8
9
10  architecture behavioral of full_adder_behavioral is
11  begin
12     sum_proc: process (A,B,CIN) begin
13        if (A xor B xor CIN) = '1' then
14           SUM <= '1';
15        else
16           SUM <= '0';
17        end if;
18     end process sum_proc;
19
20     cout_proc: process (A, B, CIN) begin
21        if ((A and B) or (B and CIN) or (CIN and A)) = '1' then
22           COUT <= '1';
23        else
24           COUT <= '0';
25        end if;
26     end process cout_proc;
27
28  end behavioral;
```

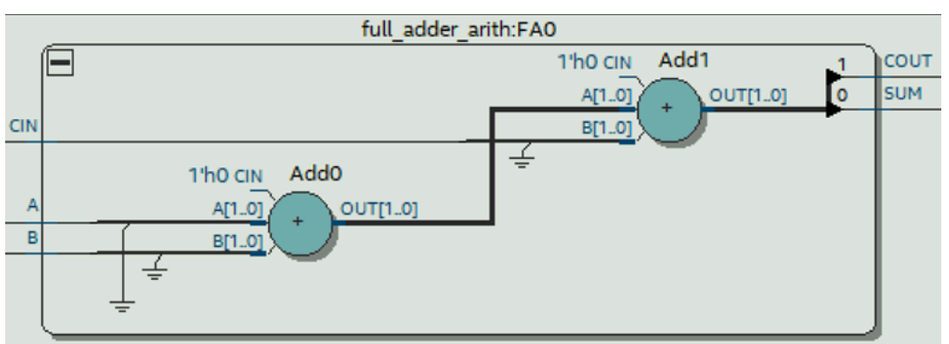


VHDL Arithmetic

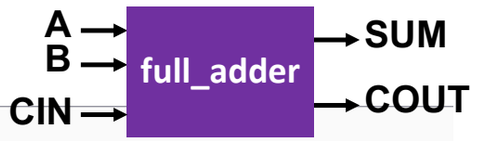

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity full_adder is
    port (A, B, CIN: in std_logic;
          SUM, COUT: out std_logic);
end full_adder;


architecture full_adder of full_adder is
begin
    SUM <= A xor B xor CIN;
    COUT <= (A and B) or (B and CIN) or (CIN and A);
end full_adder;

```

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity full_adder_arith is
    port (A, B, CIN: in std_logic;
          SUM, COUT: out std_logic);
end full_adder_arith;


architecture arithmetic of full_adder_arith is
    signal AV, BV, CINV : std_logic_vector(1 downto 0);
    signal S : std_logic_vector(1 downto 0);
begin

    AV <= '0' & A;
    BV <= '0' & B;
    CINV <= '0' & CIN;

    S <= std_logic_vector(unsigned(AV) + unsigned(BV) + unsigned(CINV));

    SUM <= S(0);
    COUT <= S(1);

end arithmetic;

```



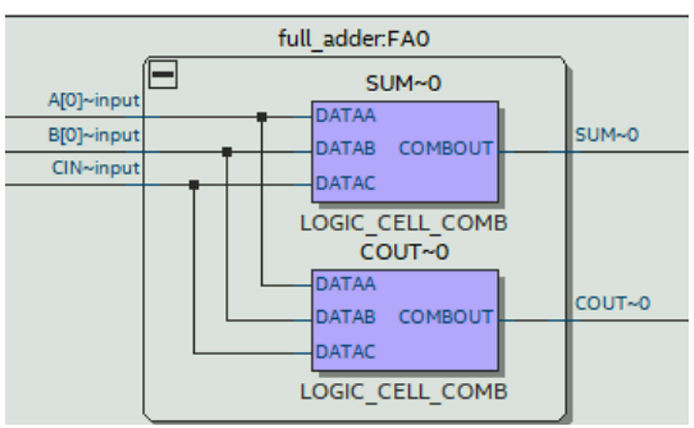


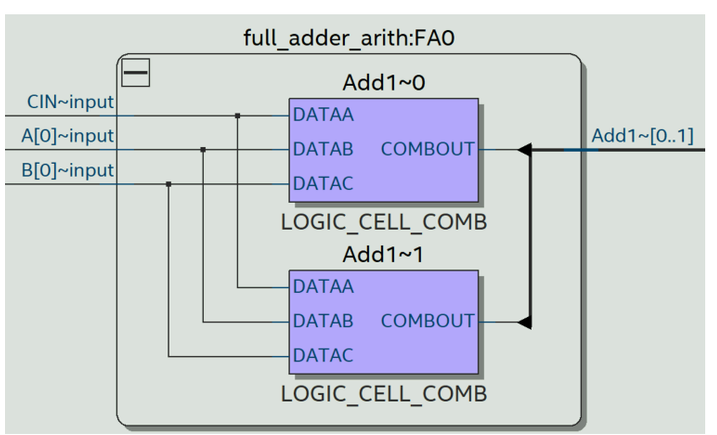
VHDL Arithmetic



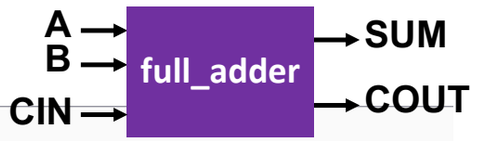

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity full_adder is
    port (A, B, CIN: in std_logic;
          SUM, COUT: out std_logic);
end full_adder;


architecture full_adder of full_adder is
begin
    SUM <= A xor B xor CIN;
    COUT <= (A and B) or (B and CIN) or (CIN and A);
end full_adder;

```

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity full_adder_arith is
    port (A, B, CIN: in std_logic;
          SUM, COUT: out std_logic);
end full_adder_arith;


architecture arithmetic of full_adder_arith is
    signal AV, BV, CINV : std_logic_vector(1 downto 0);
    signal S : std_logic_vector(1 downto 0);
begin

    AV <= '0' & A;
    BV <= '0' & B;
    CINV <= '0' & CIN;

    S <= std_logic_vector(unsigned(AV) + unsigned(BV) + unsigned(CINV));

    SUM <= S(0);
    COUT <= S(1);

end arithmetic;

```



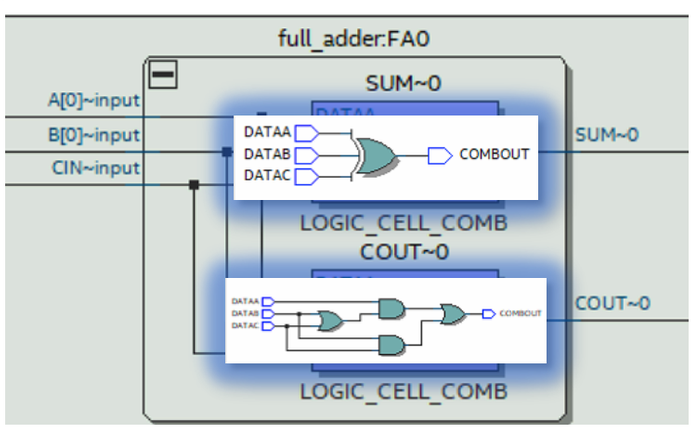


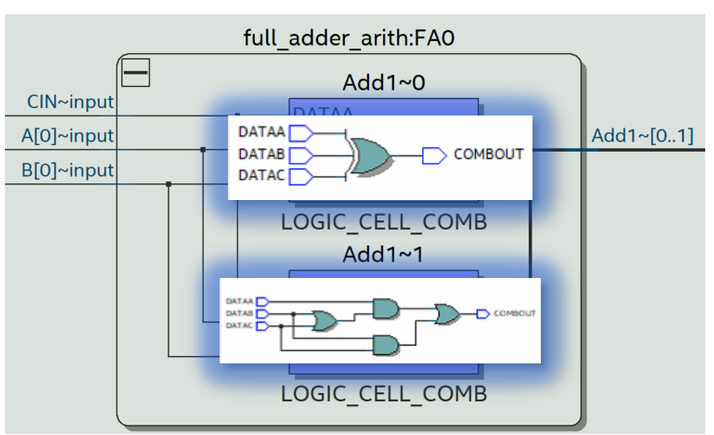
VHDL Arithmetic




```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity full_adder is
    port (A, B, CIN: in std_logic;
          SUM, COUT: out std_logic);
end full_adder;


architecture full_adder of full_adder is
begin
    SUM <= A xor B xor CIN;
    COUT <= (A and B) or (B and CIN) or (CIN and A);
end full_adder;

```

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity full_adder_arith is
    port (A, B, CIN: in std_logic;
          SUM, COUT: out std_logic);
end full_adder_arith;


architecture arithmetic of full_adder_arith is
    signal AV, BV, CINV : std_logic_vector(1 downto 0);
    signal S : std_logic_vector(1 downto 0);
begin

    AV <= '0' & A;
    BV <= '0' & B;
    CINV <= '0' & CIN;

    S <= std_logic_vector(unsigned(AV) + unsigned(BV) + unsigned(CINV));

    SUM <= S(0);
    COUT <= S(1);

end arithmetic;

```





VHDL Arithmetic


```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity full_adder_arith is
    port (A, B, CIN: in std_logic;
          SUM, COUT: out std_logic);
end full_adder_arith;


architecture arithmetic of full_adder_arith is
    signal AV, BV, CINV : std_logic_vector(1 downto 0);
    signal S : std_logic_vector(1 downto 0);
begin

    AV <= '0' & A;
    BV <= '0' & B;
    CINV <= '0' & CIN;

    S <= std_logic_vector(unsigned(AV) + unsigned(BV) + unsigned(CINV));

    SUM <= S(0);
    COUT <= S(1);

end arithmetic;

```

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity full_adder_arith_u is
    port (A, B, CIN: in std_logic;
          SUM, COUT: out std_logic);
end full_adder_arith_u;


architecture arithmetic_u of full_adder_arith_u is
    signal AV, BV, CINV : unsigned(1 downto 0);
    signal S : unsigned(1 downto 0);
begin

    AV <= '0' & A;
    BV <= '0' & B;
    CINV <= '0' & CIN;

    S <= AV + BV + CINV;

    SUM <= S(0);
    COUT <= S(1);

end arithmetic_u;

```

VHDL Arithmetic


```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity adder_arith is
    port (A, B : in std_logic_vector(3 downto 0);
          CIN : in std_logic;
          S : out std_logic_vector(3 downto 0);
          COUT : out std_logic);
end adder_arith;

architecture arith of adder_arith is
    signal Au, Bu, CINu, result : unsigned(4 downto 0);
begin

    Au(3 downto 0) <= unsigned(A);
    Au(4) <= '0';
    Bu(3 downto 0) <= unsigned(B);
    Bu(4) <= '0';
    CINu <= (0 => CIN, others => '0');

    result <= Au + Bu + CINu;

    S <= std_logic_vector(result(3 downto 0));
    COUT <= result(4);

end arith;
```

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity adder_arith_u is
    port (A, B : in unsigned(3 downto 0);
          CIN : in std_logic;
          S : out unsigned(3 downto 0);
          COUT : out std_logic);
end adder_arith_u;

architecture arith_u of adder_arith_u is
    signal Au, Bu, CINu, result : unsigned(4 downto 0);
begin

    Au(3 downto 0) <= A;
    Au(4) <= '0';
    Bu(3 downto 0) <= B;
    Bu(4) <= '0';
    CINu <= (0 => CIN, others => '0');

    result <= Au + Bu + CINu;

    S <= result(3 downto 0);
    COUT <= result(4);

end arith_u;
```

VHDL Number Representations

"000000000000"

B"000000000000"

B"0000_0000_0000"

X"000"

O"0000"