⇒ Modeling Styles

There are three modeling techniques which can be used for describing a circuit:

1. Gate level modeling / structural modeling.
2. Dataflow modeling
3. Behavioral modeling.

A circuit can be described in any one of the above techniques or by taking combination of them.

⇒ Structure of Dataflow Description

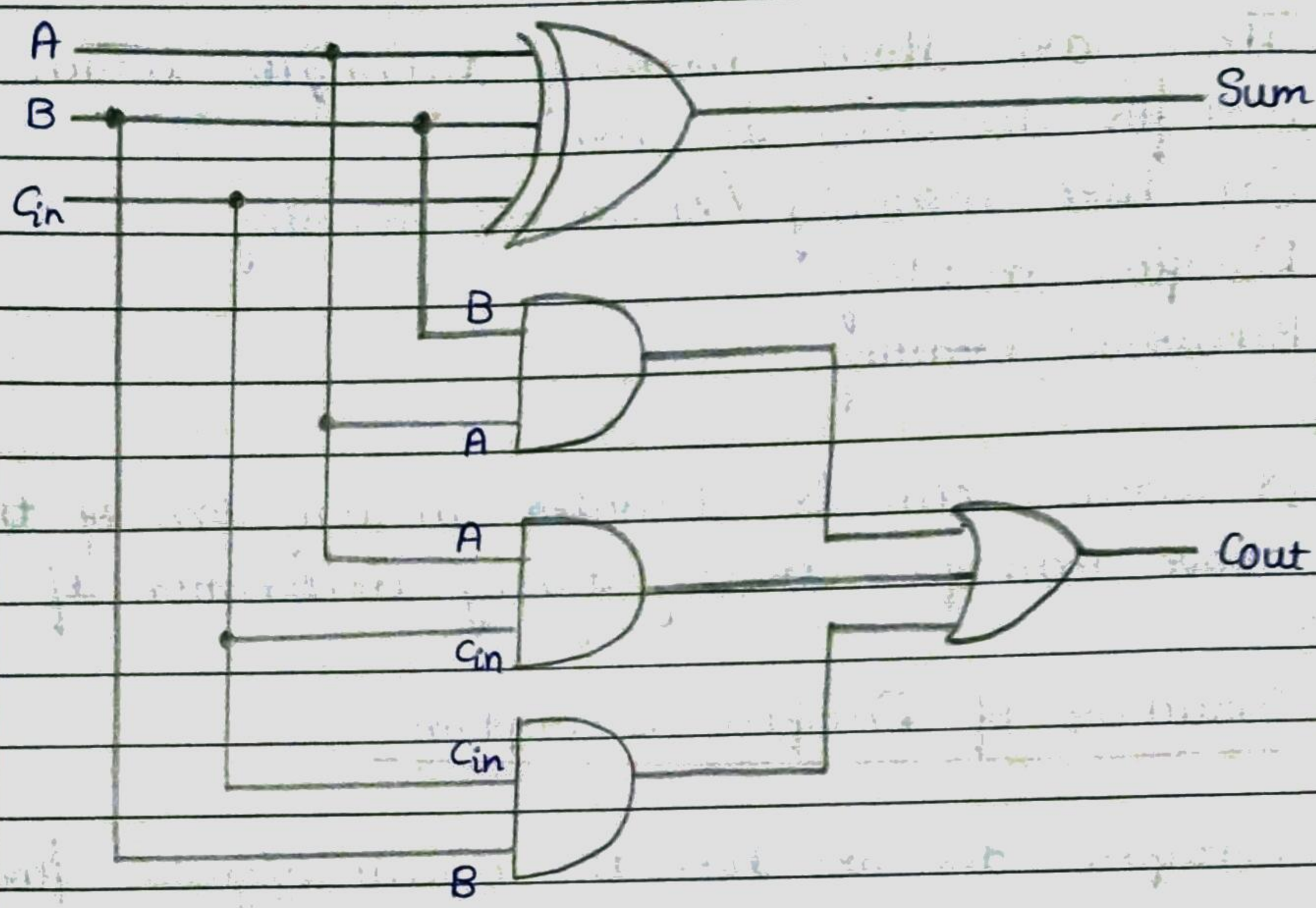
→ Dataflow describes how the circuit signals flow from the inputs to the outputs.

→ There are some concurrent statements which allow to describe the circuit in terms of operations on signals and flow of signals in the circuit. When such concurrent statements are used in a program, the style is called a 'dataflow design'.

→ Concurrent signal assignment statements are used in this type of modeling style.

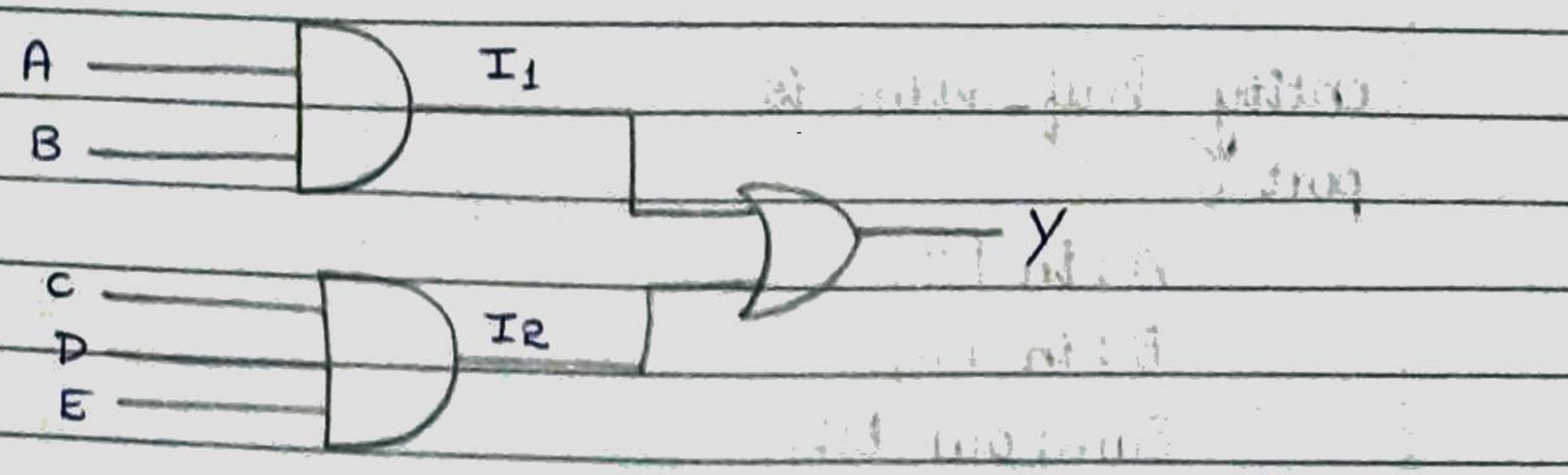
→ Example of VHDL dataflow description:

→ (1) Implementation of full-adder




```
entity full_add is
port (A, B, Cin : in bit;
      Sum, Cout : out bit);
end full_add;
architecture adder of full_add is
begin
    Sum <= A xor B xor Cin;
    Cout <= (A and B) or (Cin and A) or (Cin and B);
end adder;
```

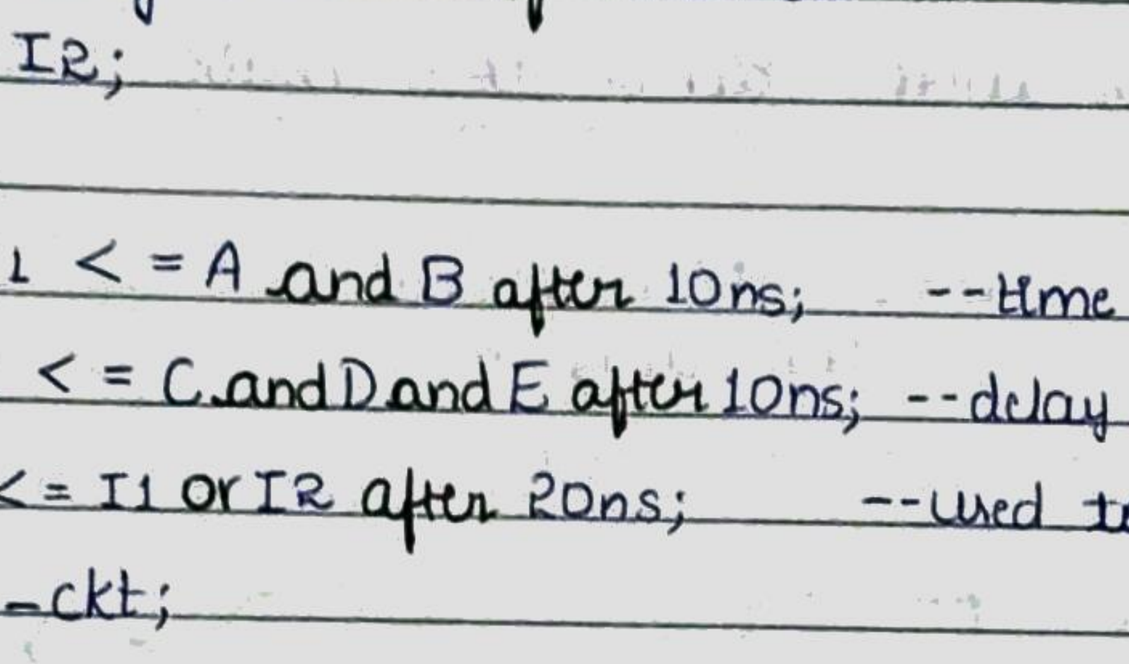
(2) AND-OR circuit



VHDL code for AND-OR circuit.

```
entity AND_OR is
port (A, B, C, D, E : in bit;
      Y : out bit);
end;
architecture digital_ckt of AND_OR is
Signal I1, I2;
begin
   St1 : I1 <= A and B after 10ns;    --time 10ns indicates the propagation
   St2 : I2 <= C and D and E after 10ns;  --delay of gate and word after is
   St3 : Y <= I1 or I2 after 20ns;    --used to specify propagation delay
end digital_ckt;
```

(3) → Dataflow description of a half adder



Logic diagram for half adder.

VHDL half adder description

```
entity half_adder is
port (
        A : in bit;
        B : in bit;
        Sum: out bit;
        Cout : out bit);
end half_adder;
architecture adder of half_adder is
begin
   Sum <= A xor B;        -- signal assignment statement
   Cout <= A and B;       -- signal assignment statement.
end adder;
```

(4) → Multiplexer with active low enable.

2 x 1 Multiplexer.

(a) Logic diagram.

(b) Logic Symbol

Truth table for a 2x1 multiplexer.

Input		Output
S	Enbar	y
X	1	0
0	0	D_0
1	0	D_1

VHDL code of a 2x1 multiplexer

```
library ieee;
use ieee.std_logic_1164.all;
entity muxc2x1 is
port (D0, D1, S, Enbar: in std_logic;
      y : out std_logic);
end muxc2x1;
architecture MUX of muxc2x1 is
signal I1, I2, I3, I4 : std_logic;
begin
                --Assume 10 nanoseconds propagation delay
                --for all and, or, and not
I1 <= not S after 10ns;
I2 <= not Enbar after 10ns;
```

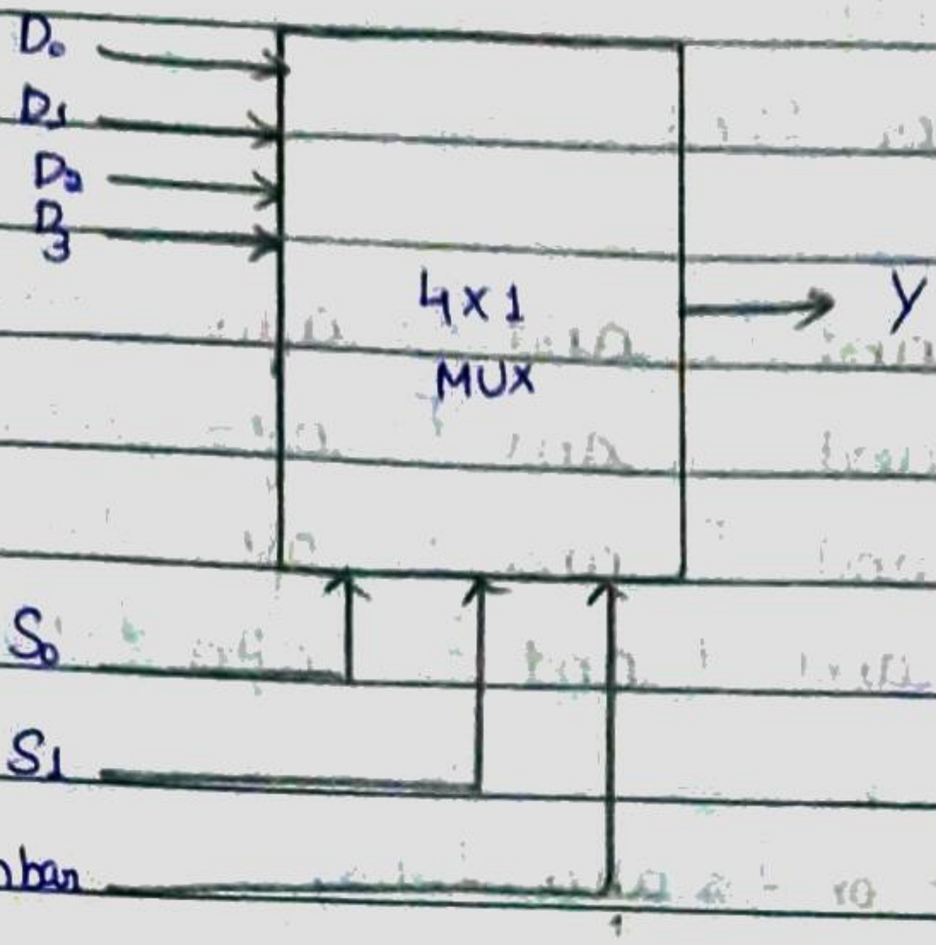
Date
Page

$I_3 <= D_0$ and I_1 and I_2 after 10ns;
$I_4 <= D_1$ and S and I_2 after 10ns;
$Y <= I_3$ or I_4 after 10ns;
end MUX;

5 → 4x1 multiplexer.

(a) Logic diagram

(b) Logic symbol



Truth Table

Input			Output
S_1	S_0	Enbar	Y
X	X	1	0
0	0	0	D_0
0	1	0	D_1
1	0	0	D_2
1	1	0	D_3

VHDL code of a 4x1 multiplexer

```
library ieee;
use ieee.std_logic_1164.all;
entity mux4x1 is
port ( D: in std_logic_vector(3 downto 0);
       S, Enbar: in std_logic;
       Y: out std_logic);
end mux4x1;
architecture MUX of mux4x1 is
signal I1, I2, I3, I4, I5, I6, I7: std_logic;
begin
                -- Assume 10 nanoseconds propagation delay
                -- for all and, or, and not.
```

I_1 <= not S_0 after 10ns;
I_2 <= not S_1 after 10 ns;
I_3 <= not Enbar after 10ns;

I_4 <= D_0 and I_1 and I_2 and I_3 after 10ns;
I_5 <= D_1 and S_0 and I_2 and I_3 after 10ns;
I_6 <= D_2 and S_1 and I_1 and I_3 after 10ns;
I_7 <= D_3 and S_0 and S_1 and I_3 after 10ns;

Y <= I_4 or I_5 or I_6 or I_7 after 10ns;
end MUX;

⇒ **Structure of VHDL Behavioral Description.**

→ The keyword in the behavioral description is process.
→ Every behavioral description has to include in the process body.
→ The process (A, B) is a concurrent statement; so its execution is initiated by the occurrence of an event.
→ The ports A and B included in the process (A, B) statement is called sensitivity list.
→ Any change in the state of any element of the sensitivity list is treated as an event.
→ The process is activated (initiated) only if an event occurs; otherwise process remains inactive.
→ If the process has no sensitivity list, the process is executed continuously.

Ex VHDL behavioral description of half adder

```
entity half_adder is
port (A : in bit;
      B : in bit;
      Sum : out bit;
      Cout : out bit);
end half_adder;
architecture adder of half_adder is
begin
process(A, B)
begin
  Sum <= A xor B after 10 ns;   -- signal-assignment statement 1
  Cout <= A and B after 10 ns;  -- signal-assignment statement 2
                                -- with 10 nanoseconds delays.
end process;
end adder;
```

⇒ **Gate Level / Structural Description.**

→ In this VHDL description, the entity part is same as that of behavioral description. However, architecture part has two components: declaration and instantiation

→ In declaration part all different components used in the system description are declared. For example, following discription declared AND gate component.

```
Component and2
port (I1, I2 : in std_logic;
      O1 : out std_logic);
end component;
```

The and2 components has two inputs: I_1 and I_2 and one output O_1.

→ Once the component is declared we can use the same component one or more times in the system description.

→ The instantiation part of the code maps the generic inputs/outputs to the actual inputs/outputs of the system.

→ For example, the statement and2 port map (A, B, Cout); maps A to input I_1 of and2, input B to input I_2 of and2, and output Cout to output O1 of and2. This mapping means that the logic relationship between A, B and Cout is the same as between I_1, I_2 and O1.

→ VHDL half-adder description.

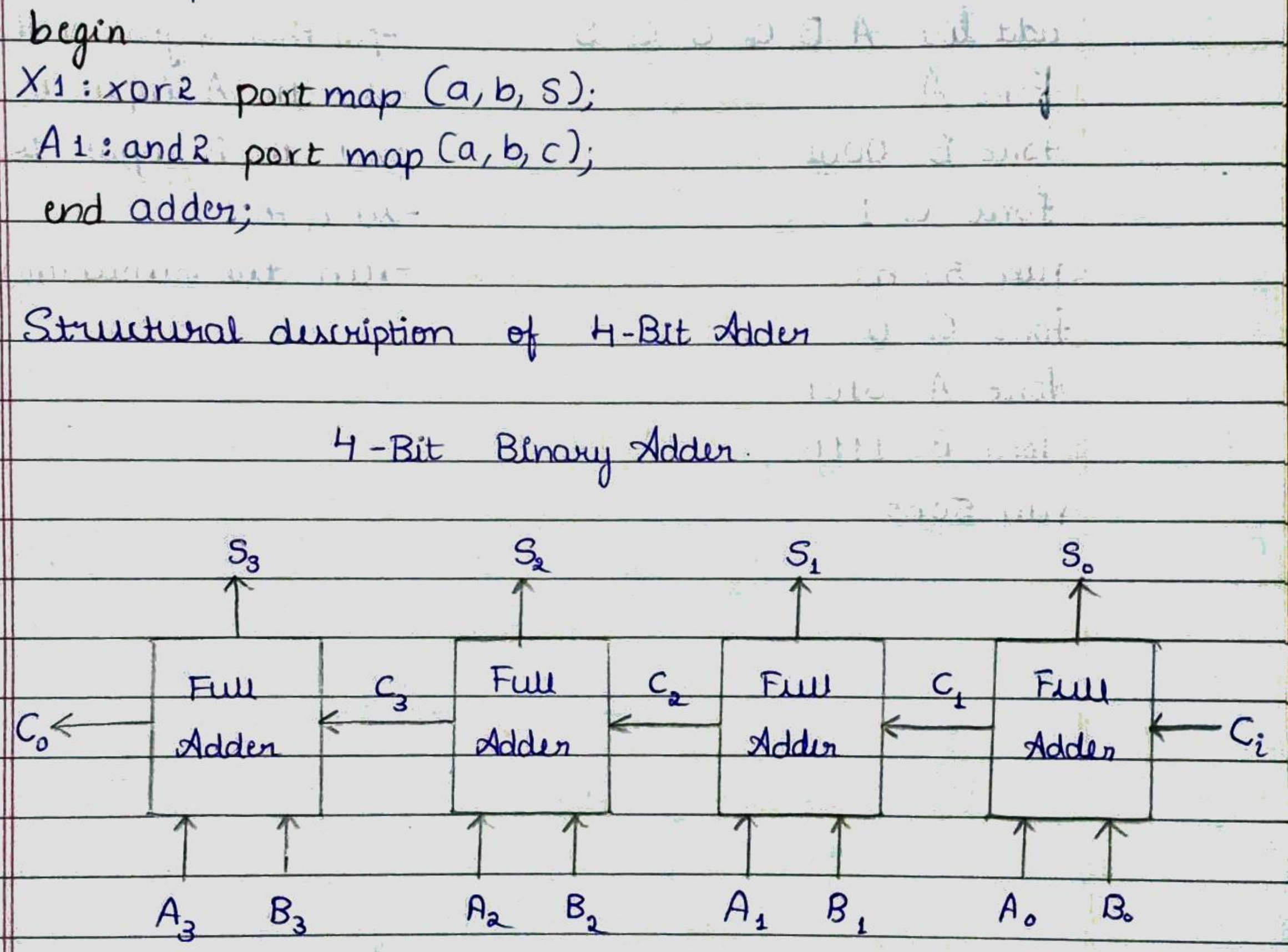
Ex

```
library ieee;
use ieee.std_logic_1164.all;
entity xor2 is
port (I1, I2: in std_logic;
      O1 : out std_logic);
end xor2;
architecture xor_gate of xor2 is
begin
    O1 <= I1 xor I2;
end xor_gate;
library ieee;
use ieee.std_logic_1164.all;
entity and2 is
port (I1, I2: in std_logic; O1: out std_logic);
end and2;
architecture and_gate of and2 is
begin
  O1 <= I1 and I2;
end and_gate;
```

```
library ieee;
use ieee.std_logic_1164.all;
entity half_adder is
port (A, B: in std_logic;
      Sum, Cout: out std_logic);
end half_adder;
architecture adder of half_adder is
component xor2
port (I1, I2: in std_logic;
      O1 : out std_logic);
end component;
component and2
port (I1, I2: in std_logic;
      O1 : out std_logic;
end component;
begin
X1 : xor2 port map (a, b, s);
A1 : and2 port map (a, b, c);
end adder;
```

→ Structural description of 4-Bit Adder

4-Bit Binary Adder.



```
entity Adder4;
architecture Structure of Adder4 is
component FullAdder
    port (X, Y, Cin: in bit; - Inputs
          Cout, Sum: out bit); - Outputs
end component;
signal C: bit_vector (3 downto 1);
begin - Instantiate four copies of the FullAdder
    FA0: FullAdder port map (A(0), B(0), Ci, C(1), S(0));
    FA1: FullAdder port map (A(1), B(1), C(1), C(2), S(1));
    FA2: FullAdder port map (A(2), B(2), C(2), C(3), S(2));
    FA3: FullAdder port map (A(3), B(3), C(3), Co, S(3));
end Structure;
```

```
add list A B Co C Ci S        -put these signals on the output list
force A                       -Set the A inputs to 1111
force B 0001                  -Set the B inputs to 0001
force Ci 1                    -Set Ci to 1
run 50 ns                     -run the simulation for 50ns.
force Ci 0
force A 0101
force B 1110
run 50ns
```