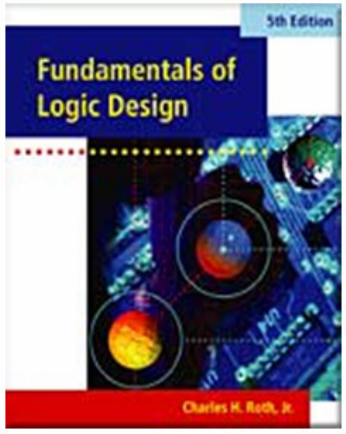
FIGURES FOR

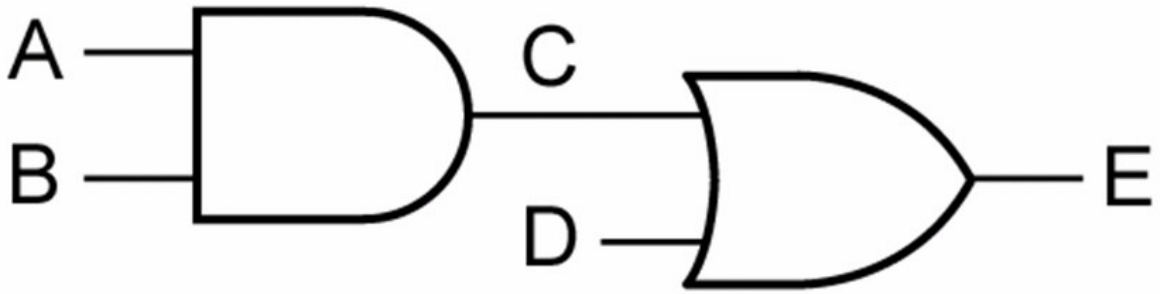
CHAPTER 10

INTRODUCTION TO VHDL



This chapter in the book includes:

Click the mouse to move to the next page.
Use the ESC key to exit this chapter.




```
C <= A and B after 5 ns;
E <= C or D after 5 ns;
```

Figure 10-1: **Gate Circuit**

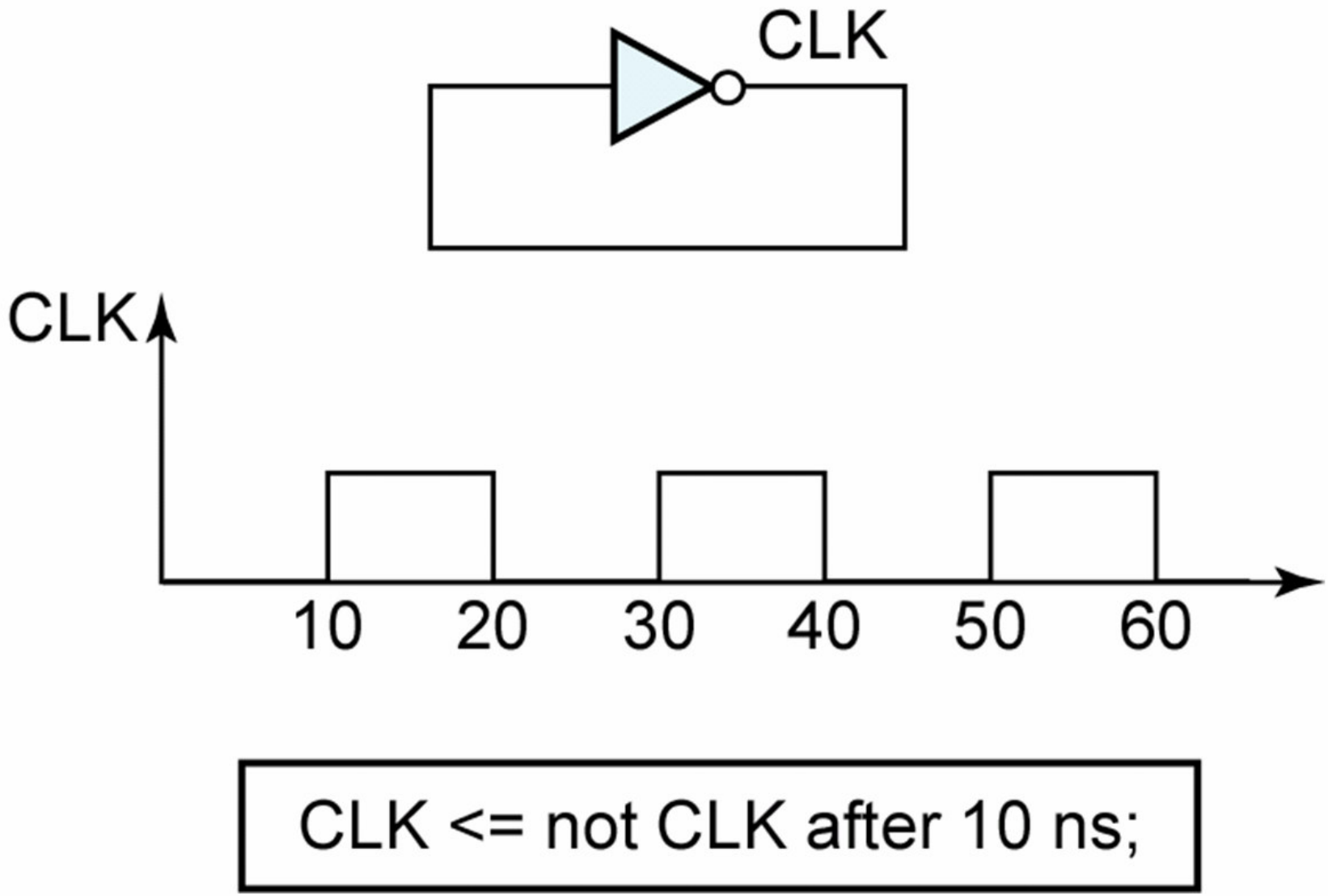
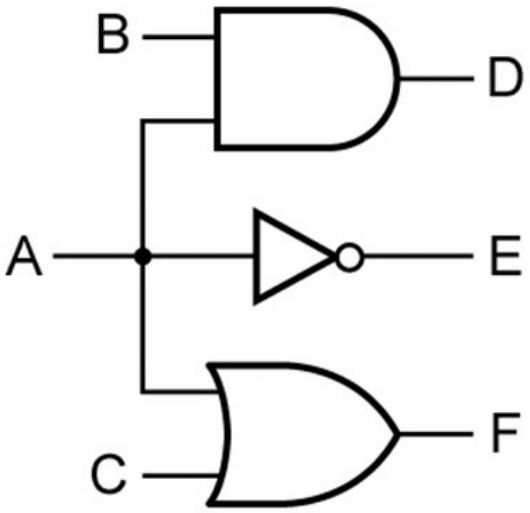
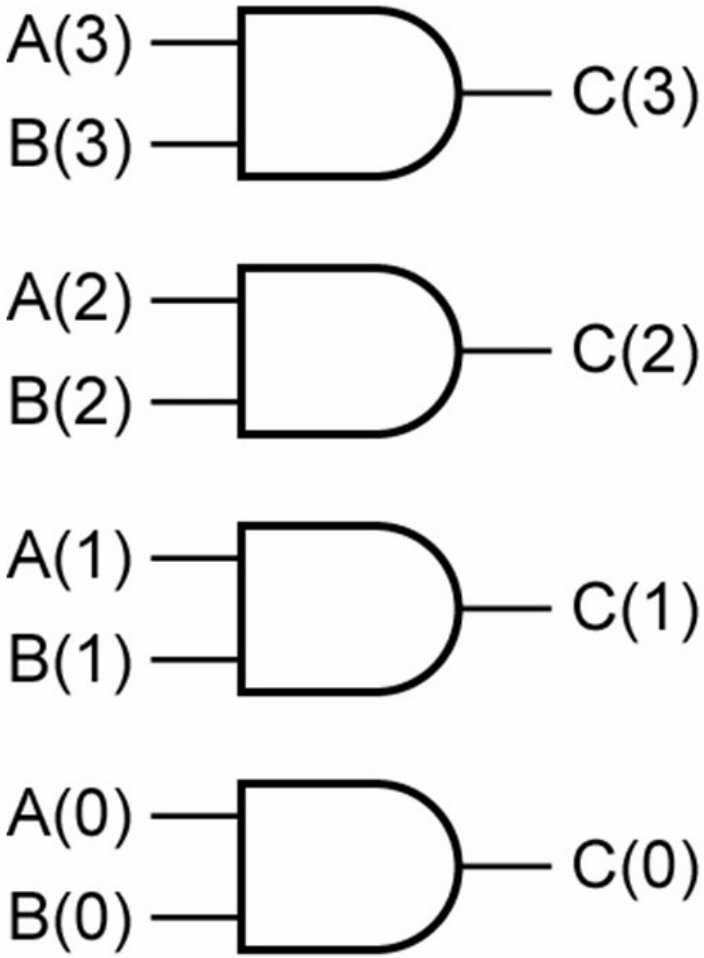


Figure 10-2: **Inverter with Feedback**




```
-- when A changes, these concurrent
-- statements all execute at the same time
D <= A and B after 2 ns;
E <= not A after 1 ns;
F <= A or C after 3 ns;
```

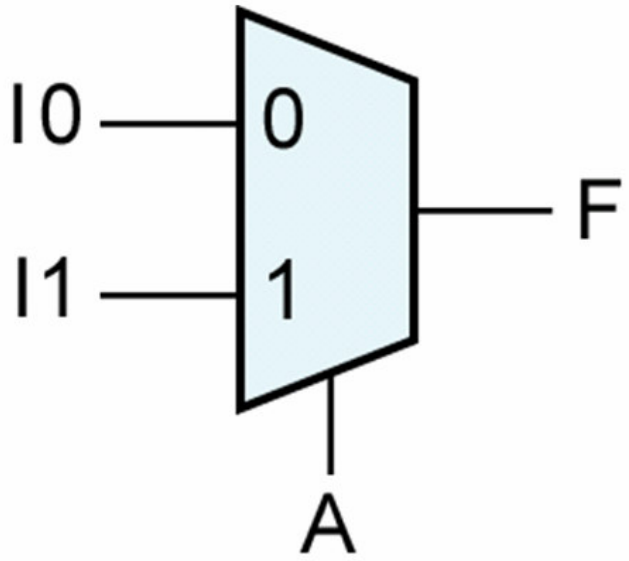
Figure 10-3: **Three Gates with a Common Input and Different Delays**




```
-- the hard way
C(3) <= A(3) and B(3);
C(2) <= A(2) and B(2);
C(1) <= A(1) and B(1);
C(0) <= A(0) and B(0);
```

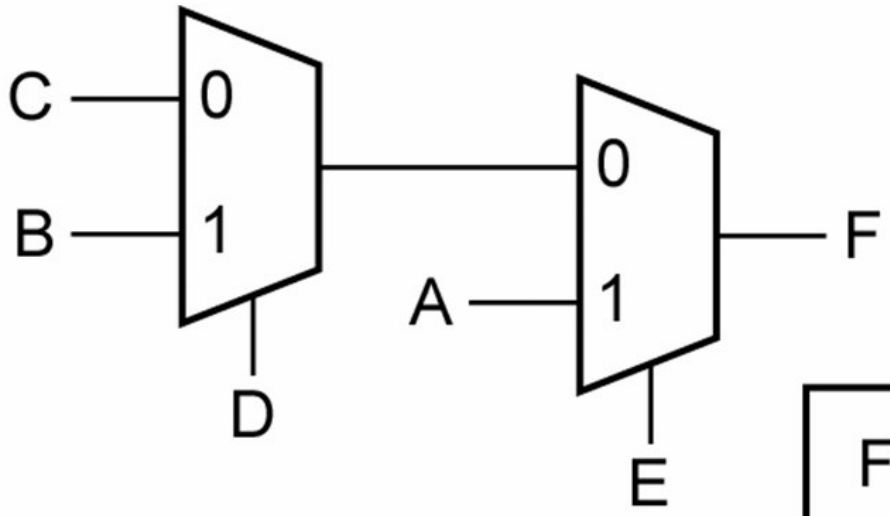
```
-- the easy way
C <= A and B;
```

Figure 10-4: **Array of AND Gates**



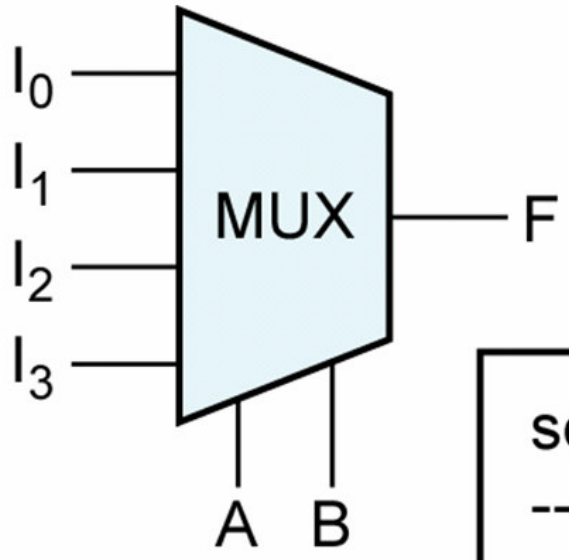

```
-- conditional signal assignment statement
F <= I0 when A = '0' else I1;
```

Figure 10-5: **2-to-1 Multiplexer**



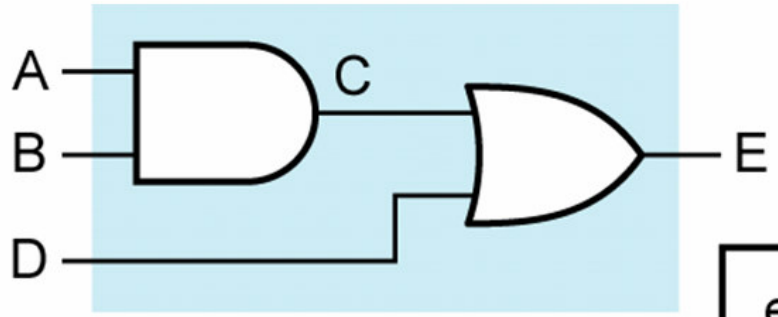

```
F <= A when E = '1'
     else B when D = '1'
     else C;
```

Figure 10-6: **Cascaded 2-to-1 MUXes**




```
sel <= A&B;
-- selected signal assignment statement
with sel select
   F <= I0 when "00",
        I1 when "01",
        I2 when "10",
        I3 when "11";
```

Figure 10-7: **4-to-1 Multiplexer**




```
entity two_gates is
    port (A,B,D: in bit; E: out bit);
end two_gates;
architecture gates of two_gates is
    signal C: bit;
begin
    C <= A and B; -- concurrent
    E <= C or D; -- statements
end gates;
```

Figure 10-8: **VHDL Module with Two Gates**

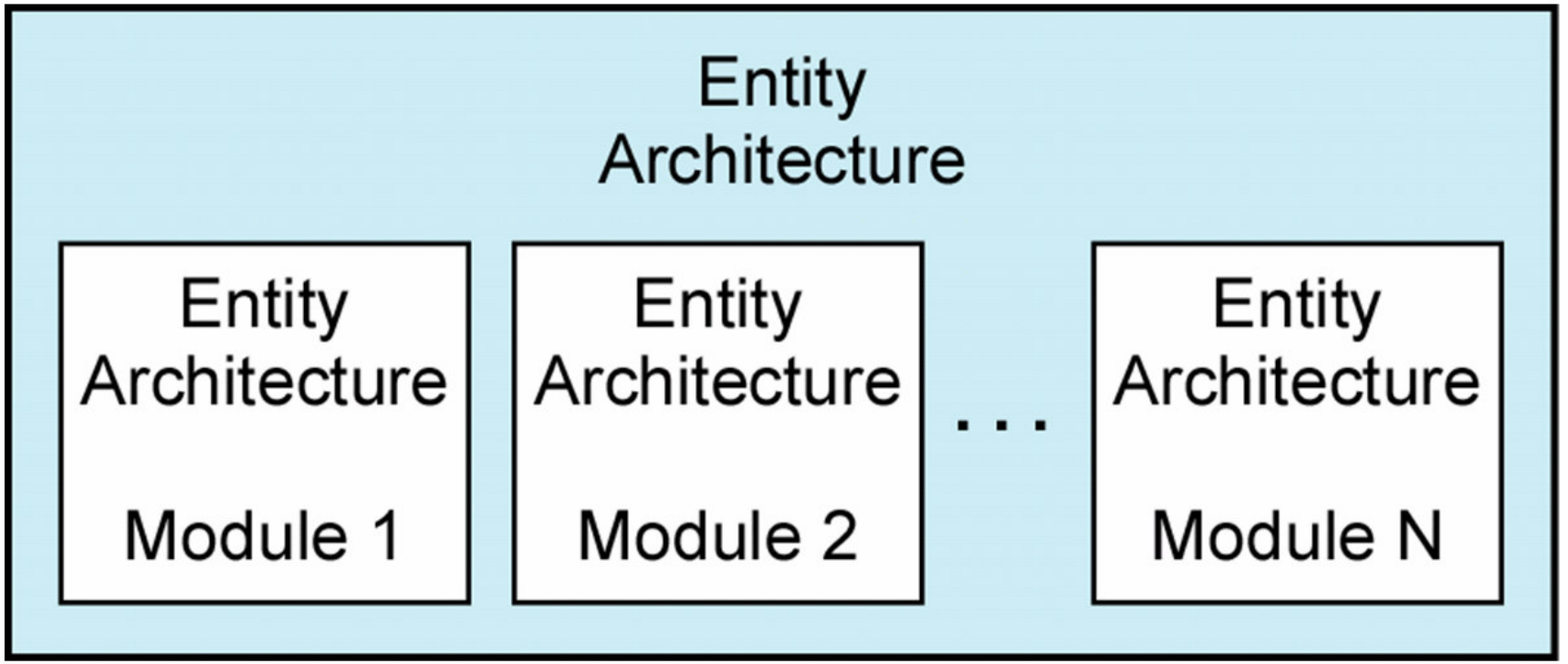
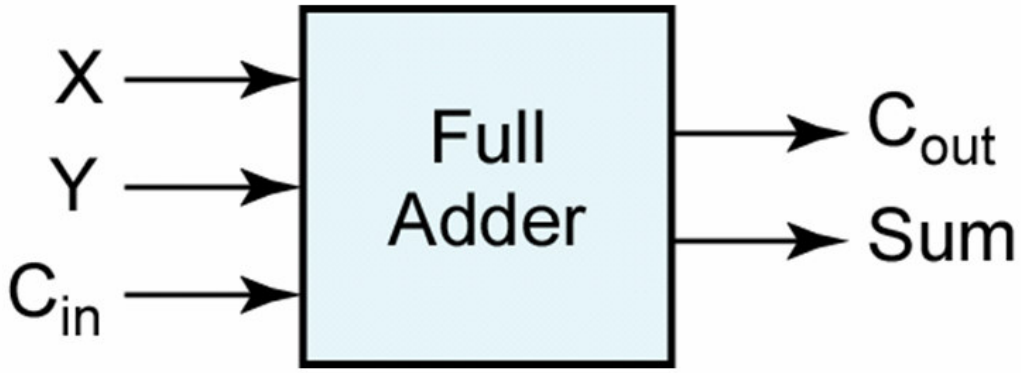


Figure 10-9: **VHDL Program Structure**




```
entity  FullAdder is
   port (X,Y,Cin: in bit;              --Inputs
         Cout, Sum: out bit);          --Outputs
end FullAdder;

architecture Equations of FullAdder is
begin                                  -- concurrent assignment statements
          Sum  <= X xor Y xor Cin after 10 ns;
          Cout <= (X and Y) or (X and Cin) or (Y and Cin) after 10 ns;
end Equations ;
```

Figure 10-10: **Full Adder Module**

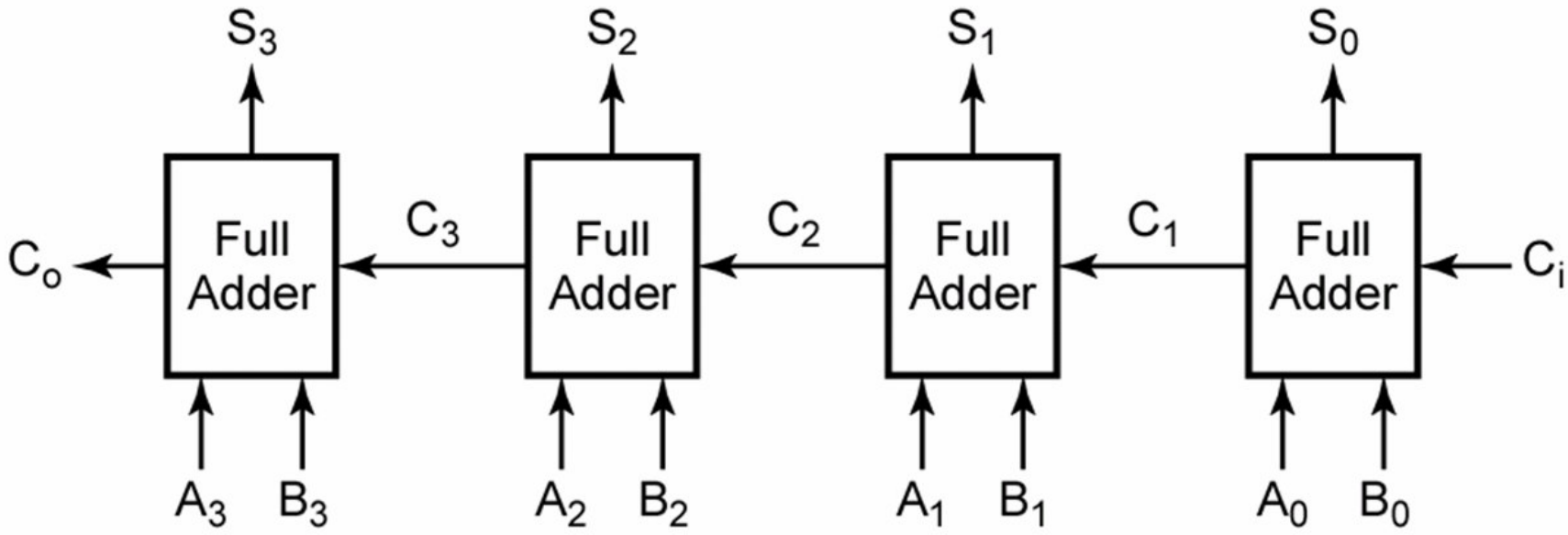
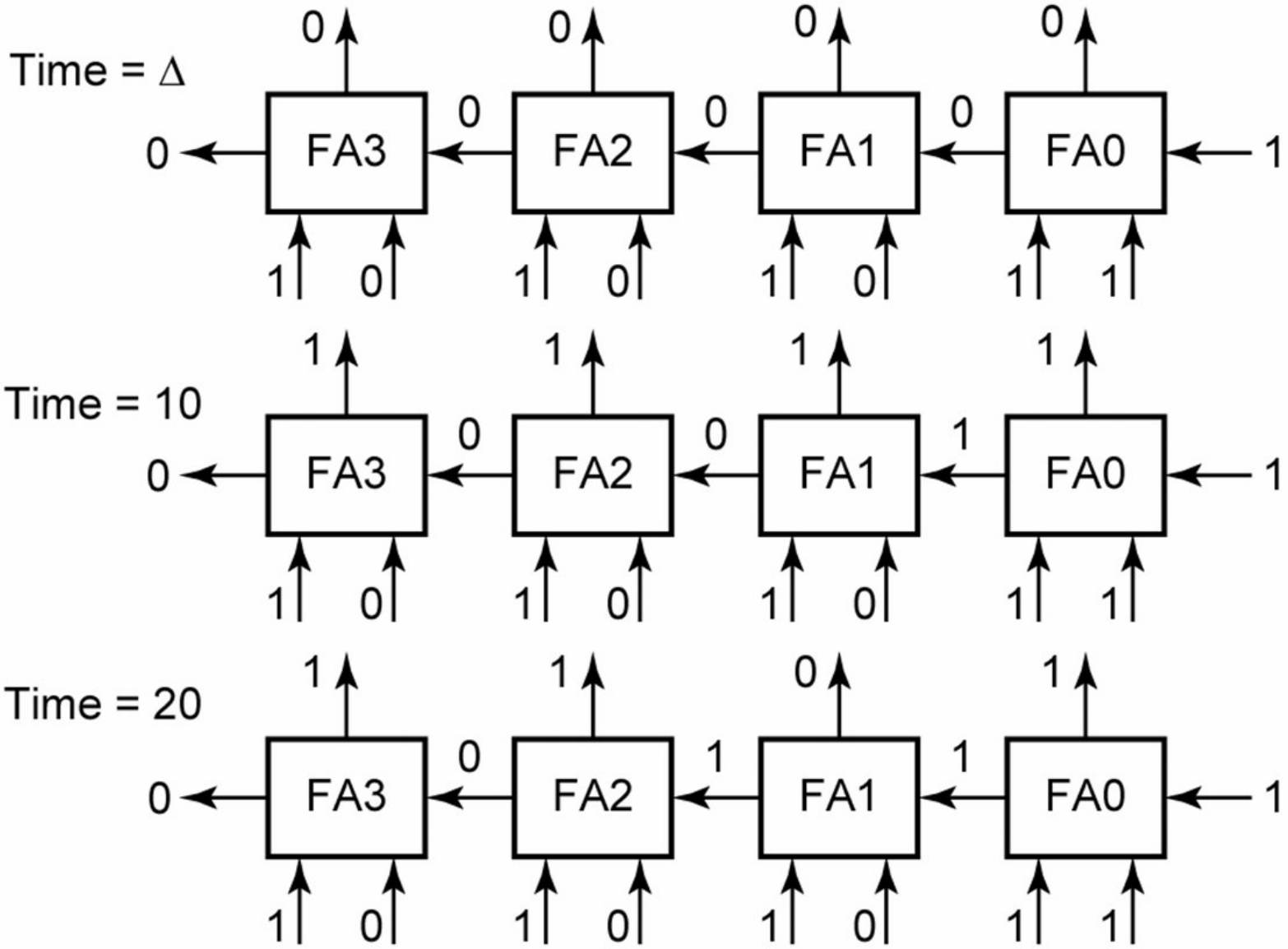


Figure 10-11: **4-Bit Binary Adder**

```
entity Adder4 is
    port (A, B: in bit_vector(3 downto 0); Ci: in bit;    -- Inputs
          S: out bit_vector(3 downto 0); Co: out bit);    -- Outputs
end 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;
```

Figure 10-12: **Structural Description of a 4-Bit Adder**



Section 10.3, p. 271

```
1    entity ROM9_17 is
2      port (A, B, C: in bit;  F: out bit_vector(0 to 3));
3    end entity;
4    architecture ROM of ROM9_17 is
5    type ROM8X4 is array (0 to 7) of bit_vector(0 to 3);
6    constant ROM1: ROM8X4 := ("1010", "1010",
       "0111","0101", "1100", "0001", "1111", "0101");
7    signal index: Integer range 0 to 7;
8    begin
9      Index <= vec2int(A&B&C);   -- A&B&C Is a 3-bit vector
10       -- vec2int is a function that converts this vector to an
             integer
11     F <= ROM1 (index);
12       -- this statement reads the output from the ROM
13   end ROM;
```

Figure 10-13: **VHDL Description of a ROM**

```
signal A,B: integer range 0 to 15;
signal C, D, E: Boolean;
--------------------------------
C <= A <= B;
D <= A = B;
E <= A > B;
```

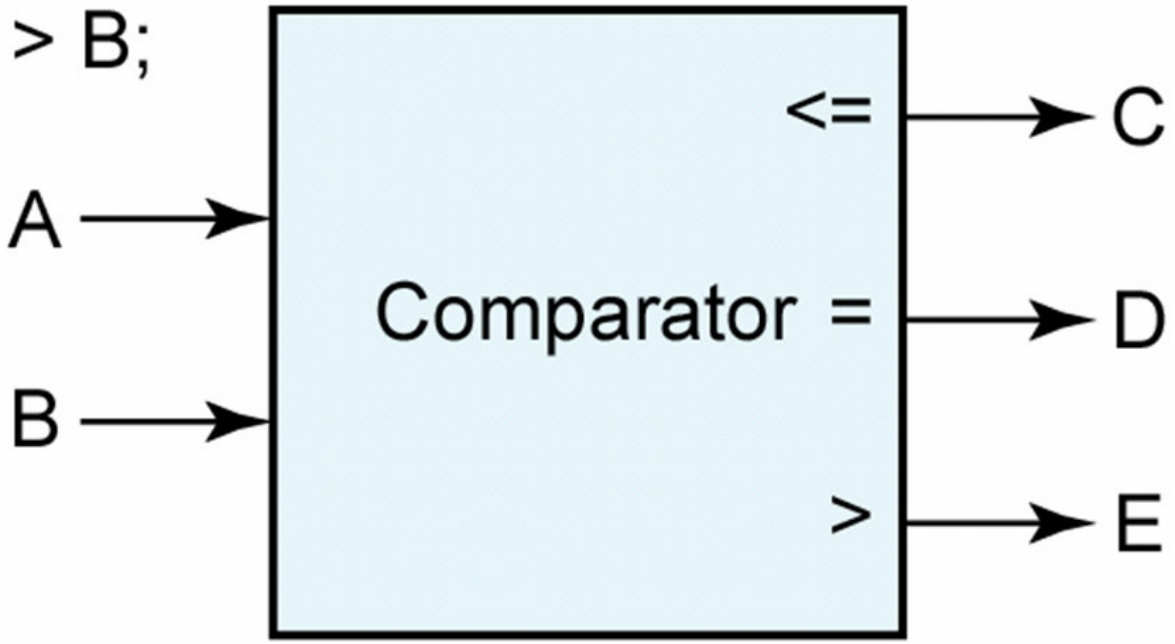
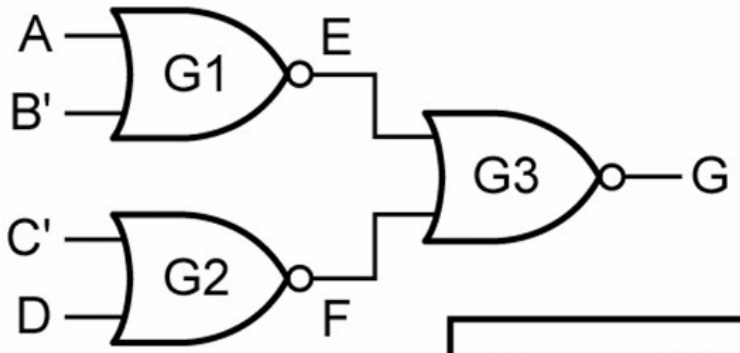
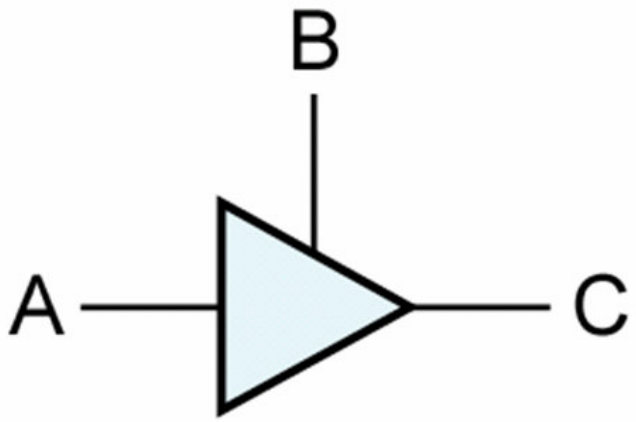


Figure 10-14: **Comparator for Integers**



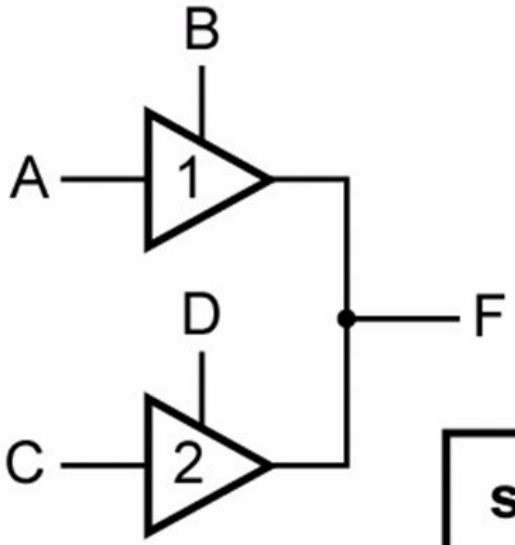

```
library BITLIB;
use BITLIB.bit_pack.all;
entity nor_nor is
    port (A,B,C,D: in bit; G: out bit);
end nor_nor;
architecture structural of nor_nor is
signal E,F,BN,CN: bit   -- internal signals
begin
    BN <= not B;  CN <= not C;
    G1: Nor2 port  map (A, BN, E);
    G2: Nor2 port  map (CN, D, F);
    G3: Nor2 port  map (E, F, G);
end structural;
```

Figure 10-15: **NOR-NOR Circuit and Structural VHDL Code Using Library Components**




```
signal A,B,C: std_logic;
--------------------------------
C <= A when B = '1' else 'Z';
```

Figure 10-16: **Tri-State Buffer**




```
signal A,C,F: std_logic_vector(3 downto 0);
signal B,D: std_logic;
---------------------------------------
-- concurrent statements
F <= A when B = '1' else "ZZZZ";
F <= C when D = '1' else "ZZZZ";
```

Figure 10-17: **Tri-State Buffers Driving a Bus**

	S2				
S1	U	X	0	1	Z
U	U	U	U	U	U
X	U	X	X	X	X
0	U	X	0	X	0
1	U	X	X	1	1
Z	U	X	0	1	Z

Figure 10-18: **Resolution Function for Two Signals**

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
-----------------------------------------
signal A,B,Sum: std_logic_vector(3 downto 0);
signal Addout: std_logic_vector(4 downto 0);
signal Cin,Cout: std_logic;
-----------------------------------
Addout <= '0'&A + B + Cin;
Sum <= Addout(3 downto 0);
Cout <= Addout(4);
```

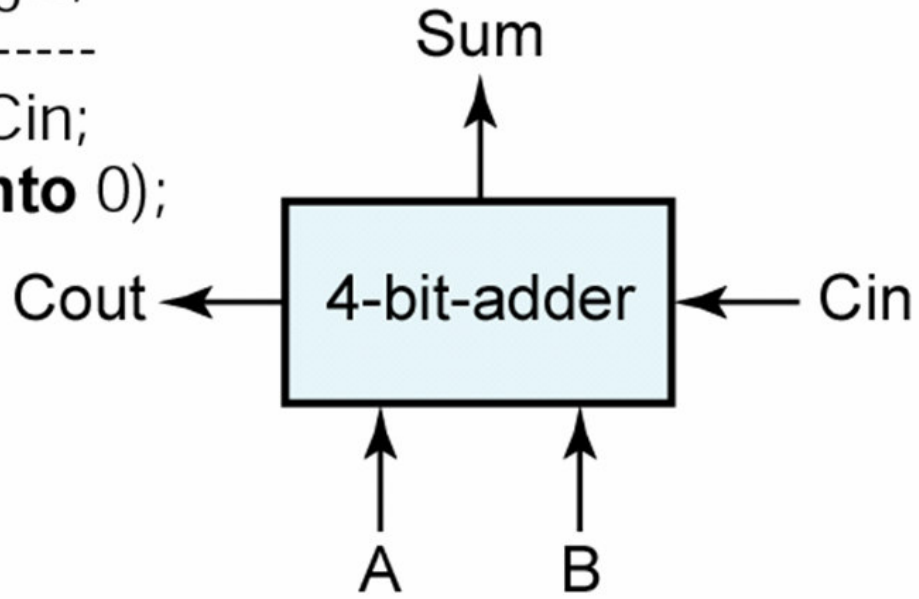


Figure 10-19: **VHDL Code for Binary Adder**

```
entity IC_pin is
  port(IO_pin: inout std_logic);
end entity;
architecture bi_dir of IC_pin is
  component IC
    port(input: in std_logic;  output: out std_logic);
  end component;
  signal input, output, en: std_logic;
begin        --  connections to bi-directional I/O pin
  IO_pin <= output when en = '1' else 'Z';
  input <= IO_pin;
  IC1: IC port map (input, output);
end bi_dir;
```

Figure 10-20: **VHDL Code for Bi-Directional I/O Pin**

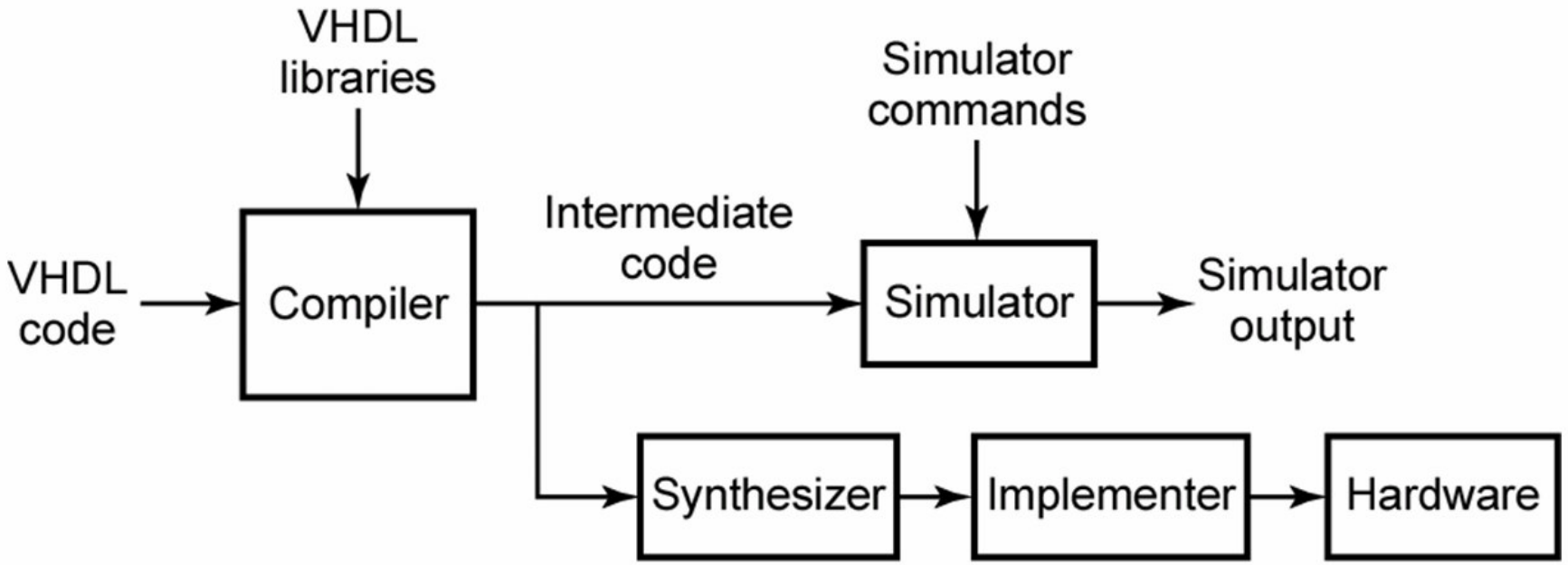
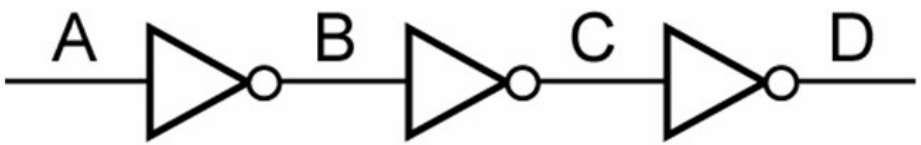


Figure 10-21: **Compilation, Simulation, and Synthesis of VHDL Code**




```
1    B <= not  A;
2    C <= not  B;
3    D <= not  C after 5 ns;
```

ns	delta	A	B	C	D
0	+0	0	1	0	1
3	+0	1	1	0	1
3	+1	1	0	0	1
3	+2	1	0	1	1
8	+0	1	0	1	0

Figure 10-22: **Simulation of VHDL Code**