CEG3155: DGD 7

Chapter 7

Question 1 (Chu 7.1)

Consider an arithmetic circuit that can perform four operations: (a+b), (a-b), (a+l) and (a-1), where a and b are 16-bit unsigned numbers and the desired operation is specified by a 2-bit control signal, ctrl.

a) Design the circuit using two adders, one incrementor and one decrementor. Derive the VHDL code.
b) Design the circuit using only one adder. Derive the VHDL code.

Solution

a)

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity prob71 is
6     port (
7        a,b: in std_logic_vector(15 downto 0);
8        ctrl: in std_logic_vector(1 downto 0);
9        y: out  std_logic_vector(15 downto 0)
10       );
11 end prob71;
12
13 architecture direct_arch of prob71 is
14    signal au, bu, yu: unsigned(15 downto 0);
15 begin
16    au <= unsigned(a);
17    bu <= unsigned(b);
18    with ctrl select
19       yu <= au+bu when "00",
20             au-bu when "01",
21             au+1  when "10",
22             au-1  when others;
23    y <= std_logic_vector(yu);
24 end direct_arch;
25
```

b)

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity prob71 is
6     port (
7        a,b: in std_logic_vector(15 downto 0);
8        ctrl: in std_logic_vector(1 downto 0);
9        y: out  std_logic_vector(15 downto 0)
10       );
11 end prob71;
12
13 architecture effi_arch of prob71 is
14    signal src0, src1: unsigned(15 downto 0);
15    signal cin: unsigned(0 downto 0);
16 begin
17    src0 <= unsigned(a);
18    with ctrl select
19       src1 <= unsigned(b) when "00",
20               unsigned(not b) when "01",
21               "0000000000000001" when "10",
22               "1111111111111111" when others;
23    cin <= "1" when ctrl="01" else "0";
24    y <= std_logic_vector(src0 + src1 + cin);
25 end effi_arch;
```

Question 2 (Chu 7.2)

Design a circuit that converts an 8-bit signed input to 8-bit sign-magnitude output (where the MSB is the sign bit and the remaining 7 bits are magnitude). Use a minimal number of relational and arithmetic operators in your design. Draw the top-level diagram and derive the VHDL code.

Solution

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity prob72 is
6     port (
7        a: in std_logic_vector(7 downto 0);
8        y: out  std_logic_vector(7 downto 0)
9        );
10 end prob72;
11
12 architecture arch of prob72 is
13    signal a_neg: std_logic_vector(7 downto 0);
14 begin
15    -- no change if a is positive]
16    -- negative a to obtain magnitude if a is negative
17    a_neg <= std_logic_vector(0 - signed(a));
18    y <= (a(7) & a_neg(6 downto 0)) when a(7)='1' else
19         a;
20 end arch;
21
```

Question 3 (Chu 7.4)

Consider a 16-bit shifting circuit that can perform rotating right or rotating left. Use selected signal assignment statements to implement the shifting function.

a) Design the circuit using one rotate-right circuit, one rotate-left circuit and one 2-to-1 multiplexer to select the desired result. Derive the VHDL code.

b) Design the circuit using one rotate-right circuit with a pre- and post-processing reversing circuit. The reversing circuit either passes the original input or reverses the input bit-wise (e.g., if a 4-bit input $a_3a_2a_1a_0$ is used, the reversed output becomes $(a_0a_1a_2a_3)$). Derive the VHDL code.

Solution

a)

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

entity prob74 is
   port (
      a: in std_logic_vector(15 downto 0);
      RR: in std_logic;
      y: out std_logic_vector(15 downto 0)
      );
end prob74;

architecture arch of prob74 is
    aR, aL: std_logic_vector(15 downto 0);

    component rotateRight is
        port(
            a: in std_logic_vector(15 downto 0);
            y: out std_logic_vector(15 downto 0)
        );
    end component;

    component rotateLeft is
        port(
            a: in std_logic_vector(15 downto 0);
            y: out std_logic_vector(15 downto 0)
        );
    end component;

    component mux2to1 is
        port(
            a, b: in std_logic_vector(15 downto 0);
            s: in std_logic;
            y: out std_logic_vector(15 downto 0)
        );
    end component;
begin
    L1: rotateRight port map(a, aR);
    L2: rotateLeft port map(a, aL);
    L3: mux2to1 port map(aR, aL, RR, y);
end arch;
```

b)

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

entity prob74 is
   port (
      a: in std_logic_vector(15 downto 0);
      RR: in std_logic;
      y: out std_logic_vector(15 downto 0)
      );
end prob74;

architecture arch of prob74 is
    aR, aRR: std_logic_vector(15 downto 0);

    component rotateRight is
        port(
            a: in std_logic_vector(15 downto 0);
            y: out std_logic_vector(15 downto 0)
        );
    end component;

    component reverse is
        port(
            a: in std_logic_vector(15 downto 0);
            r: in std_logic;            -- reverse if r = 1
            y: out std_logic_vector(15 downto 0)
        );
    end component;
begin
    L1: reverse port map(a, RR, aR);
    L2: rotateRight port map(aR, aRR);
    L3: reverse port map(aRR, RR, y);
end arch;
```

Question 4 (Chu 7.8)

Design a 16-bit rotate-left shifting circuit using the multilevel structure discussed in Section 7.4.4.

Example (Section 7.4.4):

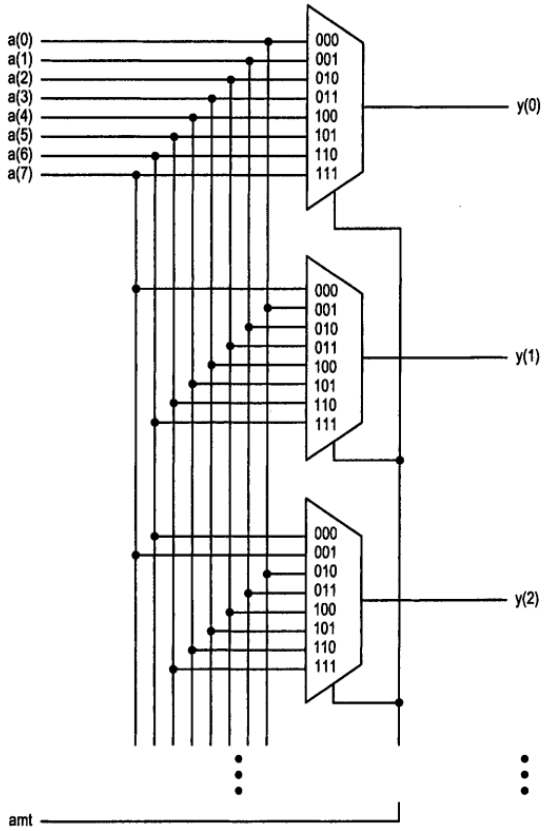
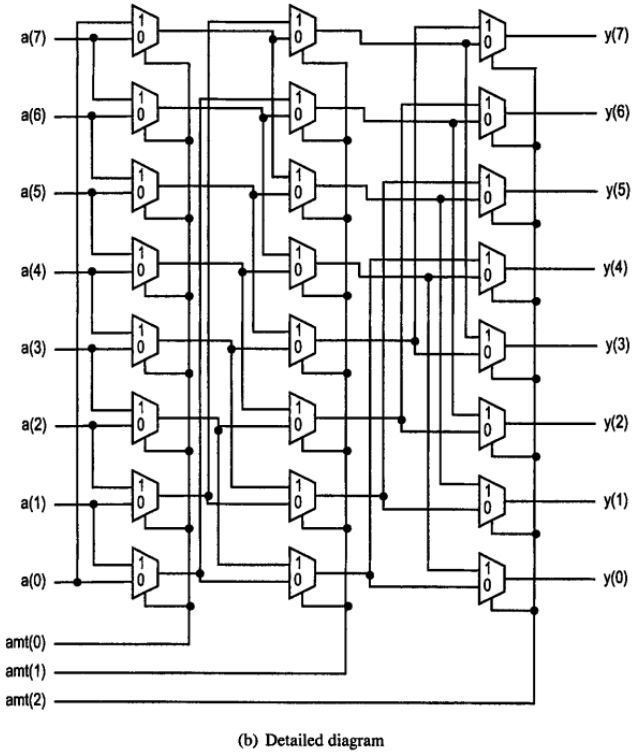
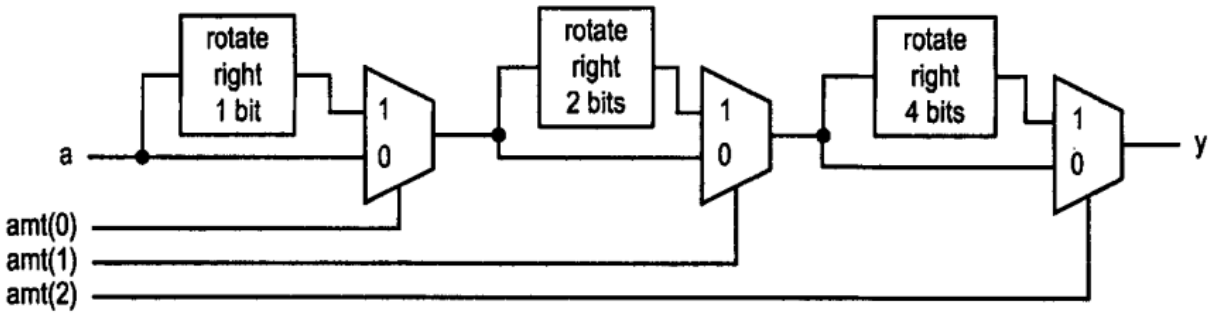


Figure 7.11 Barrel shifter using a single level of 8-to-1 multiplexers.



(b) Detailed diagram

Figure 7.12 Barrel shifter using three levels of 2-to-1 multiplexers.



(a) Block diagram

Solution

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity prob78 is
5     port(
6        a: in std_logic_vector(15 downto 0);
7        amt: in std_logic_vector(3 downto 0);
8        y: out std_logic_vector(15 downto 0)
9     );
10 end prob78;
11
12 architecture multi_level_arch of prob78 is
13    signal le0_out, le1_out, le2_out, le3_out: std_logic_vector(15 downto 0);
14 begin
15    -- level 0, shift 0 or 1 bit
16    le0_out <= a(14 downto 0) & a(15) when amt(0)='1' else
17               a;
18    -- level 1, shift 0 or 2 bits
19    le1_out <=
20       le0_out(13 downto 0) & le0_out(15 downto 14) when amt(1)='1' else
21       le0_out;
22    -- level 2, shift 0 or 4 bits
23    le2_out <=
24       le1_out(11 downto 0) & le1_out(15 downto 12) when amt(2)='1' else
25       le1_out;
26        -- level 3, shift 0 or 8 bits
27    le3_out <=
28       le2_out(7 downto 0) & le2_out(15 downto 8) when amt(3)='1' else
29       le2_out;
30    y <= le2_out;
31 end multi_level_arch;
32
```