第五章作业

2022211363 谢牧航

13

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity decoder3to8 is
    Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
           Y : out STD_LOGIC_VECTOR (7 downto 0));
end decoder3to8;

architecture Behavioral of decoder3to8 is
begin
    process(A)
    begin
        case A is
            when "000" => Y <= "00000001";
            when "001" => Y <= "00000010";
            when "010" => Y <= "00000100";
            when "011" => Y <= "00001000";
            when "100" => Y <= "00010000";
            when "101" => Y <= "00100000";
            when "110" => Y <= "01000000";
            when "111" => Y <= "10000000";
            when others => Y <= "00000000";
        end case;
    end process;
end Behavioral;
```

15

```
library IEEE
use IEEE.std_logic_1164.all

entity priorityEncoder8to3 is
    port(
        input: in std_logic_vector(7 downto 0);
        output: out std_logic_vector(2 downto 0)
    );
end priorityEncoder8to3;

architecture priorityEncoder8to3_architecture of priorityEncoder8to3 is
    begin
        process(input)
        begin
            if input(7) = '1' then
                output <= "111";
            elsif input(6) = '1' then
                output <= "110";
            elsif input(5) = '1' then
                output <= "101";
            elsif input(4) = '1' then
                output <= "100";
            elsif input(3) = '1' then
                output <= "011";
            elsif input(2) = '1' then
                output <= "010";
            elsif input(1) = '1' then
                output <= "001";
            else output <= "000";
            end if;
        end process;
    end priorityEncoder8to3_architecture;
```

17

D 触发器

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

entity dff is
    port (
        d   : in  std_logic;
        clk : in  std_logic;
        q   : out std_logic
    );
end dff;

architecture rtl of dff is
begin
    process (clk)
    begin
        if (clk'event and clk = '1') then
            q <= d;
        end if;
    end process;
end rtl;
```

寄存器

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

entity shift_reg is
    port (
        d1  : in  std_logic;
        cp  : in  std_logic;
        d0  : out std_logic
    );
end shift_reg;

architecture structure of shift_reg is
    component dff
        port (
            d   : in  std_logic;
            clk : in  std_logic;
            q   : out std_logic
        );
    end component;

    signal q: std_logic_vector(4 downto 0);

begin
    q(0) <= d1;

    g1: for i in 0 to 3 generate
        dffx: dff port map (q(i), cp, q(i+1));
    end generate g1;

    d0 <= q(4);
end structure;
```

19

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

entity bcd_counter is
    port(
        clk   : in  std_logic;
        reset : in  std_logic;
        count : out std_logic_vector(3 downto 0)
    );
end bcd_counter;

architecture behavior of bcd_counter is
    signal temp_count: std_logic_vector(3 downto 0) := "0000";
begin
    process(clk, reset)
    begin
        if reset = '1' then
            temp_count <= "0000";
        elsif rising_edge(clk) then
            if temp_count = "1001" then
                temp_count <= "0000";
            else
                temp_count <= std_logic_vector(unsigned(temp_count) + 1);
            end if;
        end if;
    end process;

    count <= temp_count;  -- Output the count value
end behavior;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- 定义实体
entity gray_counter is
    Port ( clk : in STD_LOGIC;
           y   : in STD_LOGIC;
           gray_out : out STD_LOGIC_VECTOR (2 downto 0));
end gray_counter;

architecture Behavioral of gray_counter is
begin
    process(clk)
    begin
        if rising_edge(clk) then
            if y = '1' then
                case gray_out is
                    when "000" => gray_out <= "001";
                    when "001" => gray_out <= "011";
                    when "011" => gray_out <= "010";
                    when "010" => gray_out <= "110";
                    when "110" => gray_out <= "111";
                    when "111" => gray_out <= "101";
                    when "101" => gray_out <= "100";
                    when "100" => gray_out <= "000";
                    when others => gray_out <= "000";
                end case;
            elsif y = '0' then
                case gray_out is
                    when "000" => gray_out <= "100";
                    when "100" => gray_out <= "101";
                    when "101" => gray_out <= "111";
                    when "111" => gray_out <= "110";
                    when "110" => gray_out <= "010";
                    when "010" => gray_out <= "011";
                    when "011" => gray_out <= "001";
                    when "001" => gray_out <= "000";
                    when others => gray_out <= "000";
                end case;
            end if;
```

```
        end if;
    end process;
end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity state_machine is
    Port ( clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           k : in STD_LOGIC;
           output_signal : out STD_LOGIC_VECTOR (1 downto 0));
end state_machine;

architecture Behavioral of state_machine is
    type state_type is (S0, S1, S2, S3);
    signal current_state, next_state : state_type;

begin
    process(clk, reset)
    begin
        if reset = '1' then
            current_state <= S0;
        elsif rising_edge(clk) then
            current_state <= next_state;
        end if;
    end process;

    process(current_state, k)
    begin
        case current_state is
            when S0 =>
                if k = '1' then
                    next_state <= S0;
                else
                    next_state <= S1;
                end if;

            when S1 =>
                if k = '1' then
                    next_state <= S2;
                else
                    next_state <= S1;
                end if;
```

```
            when S2 =>
                if k = '1' then
                    next_state <= S2;
                else
                    next_state <= S3;
                end if;

            when S3 =>
                if k = '1' then
                    next_state <= S0;
                else
                    next_state <= S3;
                end if;
        end case;
    end process;

    output_signal <= "00" when current_state = S0 else
                     "01" when current_state = S1 else
                     "10" when current_state = S2 else
                     "11";
end Behavioral;
```