1. A counter is used to generate a sequence of outputs that are control signals, thus implementing a controller. The control signals are directly encoded as the counter's outputs. The control signals are labeled A, B, C, and D. Their sequence of values is 0000, 0001, 0110, 1100, 1000 and 0000 (sequence repeats).

(a) Write a complete VHDL design description of the controller as a FSM using either two or three processes as is appropriate. The outputs are to be declared as scalars. The counter has a synchronous reset that is active low.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity controller is
   Port ( clk : in  std_logic;
          reset : in  std_logic;
          A : out  std_logic;
          B : out  std_logic;
          C : out  std_logic;
          D : out  std_logic);
end controller;

architecture Behavioral of controller is

type state_type is (S0, S1, S2, S3, S4, S5);
signal current_state, next_state : state_type;

begin

process (clk, reset)
begin
   if reset = '0' then
      current_state <= S0;
   elsif rising_edge(clk) then
      current_state <= next_state;
   end if;
```

```
end process;

process (current_state)
begin
    case current_state is
        when S0 =>
            A <= '0';
            B <= '0';
            C <= '0';
            D <= '0';
            next_state <= S1;
        when S1 =>
            A <= '0';
            B <= '0';
            C <= '0';
            D <= '1';
            next_state <= S2;
        when S2 =>
            A <= '0';
            B <= '1';
            C <= '1';
            D <= '0';
            next_state <= S3;
        when S3 =>
            A <= '1';
            B <= '1';
            C <= '0';
            D <= '0';
            next_state <= S4;
        when S4 =>
            A <= '1';
            B <= '0';
            C <= '0';
            D <= '0';
            next_state <= S5;
```

```
        when S5 =>
            A <= '0';
            B <= '0';
            C <= '0';
            D <= '0';
            next_state <= S0;
        when others =>
            next_state <= S0;
    end case;
end process;

end Behavioral;
```

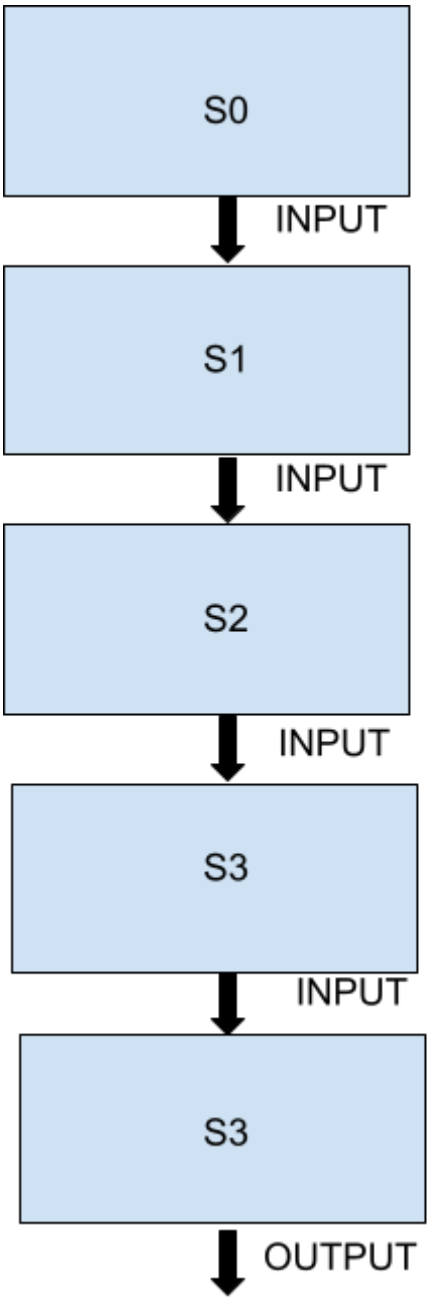
(b) Is this controller a Mealy or Moore FSM? Explain your answer.

A Mealy FSM is a type of finite state machine where the output depends on both the current state and the input. In contrast, a Moore FSM is a type of finite state machine where the output depends only on the current state.

The VHDL design above uses two processes to implement the FSM. One process is responsible for the state transition logic, and the other process is responsible for the output logic. The output logic assigns the values of the control signals A, B, C, and D based on the current state, but the input is not used in the output logic. This means that the output is only dependent on the current state, and not on any external input. Thus, this controller is a Moore FSM.

2. A sequence recognizer has a single input and a single output. The output of the sequence recognizer is a 1 if and only if the last four input values were 0101. The sequence recognizer has a synchronous active low reset input.

(a) Draw a state diagram for the sequence recognizer.



(b) Write a complete VHDL design description of the sequence recognizer as a two- or three- (whichever is most appropriate) process FSM.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity seq_recog is
    Port ( clk : in  std_logic;
           reset : in  std_logic;
           input : in std_logic;
           output : out  std_logic);
end seq_recog;

architecture Behavioral of seq_recog is

type state_type is (S0, S1, S2, S3);
signal current_state, next_state : state_type;
signal seq : std_logic_vector(3 downto 0);

begin

process (clk, reset)
begin
    if reset = '0' then
        current_state <= S0;
        seq <= (others => '0');
    elsif rising_edge(clk) then
        current_state <= next_state;
    end if;
end process;

process (current_state, input)
begin
    case current_state is
        when S0 =>
            seq <= input & seq(2 downto 0);
            next_state <= S1;
        when S1 =>
            seq <= input & seq(2 downto 0);
```

```
            next_state <= S2;
        when S2 =>
            seq <= input & seq(2 downto 0);
            next_state <= S3;
        when S3 =>
            seq <= input & seq(2 downto 0);
            if seq = "0101" then
                output <= '1';
            else
                output <= '0';
            end if;
            next_state <= S0;
        when others =>
            next_state <= S0;
    end case;
end process;

end Behavioral;
```