Lund university

Exercise 3 in VHDL

EITF65

1 Sequential logic with VHDL

Task 1: A state machine

Describe in VHDL the state machine in example 2.13 from the course book, which calculates the parity of an incomming sequence. Divide the state machine into two processes. If the provided simulation file is used, the entity and ports should be named as in the listing below. The name of the design file should be the same as the entity name.

```
entity parity is
    port(
        clk : in  std_logic;
        i   : in  std_logic;
        p   : out std_logic
    );
end parity;
```

Task 2: The lion cage

Describe in VHDL the lion cage from example 2.4 of the course book. This time there is no provided simulation file. Create your own. You can use the one from the previous exercise as a reference.

Task 3: Help your friend Miranda

A girl named Miranda wants to create a synchronous sequential circuit that outputs a 1 if and only if the last 5 input signals have been 00100. Describe the circuit in VHDL. As before, if the provided simulation file is used the entity and ports should be named as in the listing below. The name of the design file should be the same as the entity name.

```
entity mirandas_state_machine is
    port(
        clk     : in  std_logic;
        n_rst   : in  std_logic;
        i       : in  std_logic;
        o       : out std_logic
    );
end mirandas_state_machine;
```

Please note that the reset is active low (that is what the prefix n_ denotes). This means that the state machine should go to state s_0 when the reset signal is 0.

2 Solutions

Task 1: A state machine

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


entity parity is
    Port ( clk : in STD_LOGIC;
           i : in STD_LOGIC;
           p : out STD_LOGIC);
end parity;

architecture Behavioral of parity is
    type state_type is (even, odd);
    signal current_state, next_state: state_type;
begin
    clocking: process (clk)
    begin
        if rising_edge(clk) then
            current_state <= next_state;
        end if;
    end process;

    process (i, current_state)
    begin
        next_state <= current_state;
        case current_state is
            when even =>
                if i = '1' then
                    p <= '1';
                    next_state <= odd;
                else
                    p <= '0';
                end if;
            when odd =>
                if i = '1' then
                    p <= '0';
                    next_state <= even;
                else
                    p <= '1';
                end if;
        end case;
    end process;
end Behavioral;
```

Task 2: The lion cage

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


entity lioncage is
    Port ( clk : in STD_LOGIC;
           detector : in STD_LOGIC_VECTOR (1 downto 0);
           danger : out STD_LOGIC);
end lioncage;

architecture Behavioral of lioncage is
    type state_type is (s0, s1, s1hat, s2);
    signal current_state, next_state: state_type;
begin
    process (clk)
    begin
        if rising_edge(clk) then
            current_state <= next_state;
        end if;
    end process;

    process (detector, current_state)
    begin
        next_state <= current_state;
        case current_state is
            when s0 =>
                if detector = "01" then
                    next_state <= s1;
                end if;
            when s1 =>
                if detector = "10" then
                    next_state <= s1hat;
                elsif detector = "11" then
                    next_state <= s0;
                end if;
            when s1hat =>
                if detector = "01" then
                    next_state <= s2;
                elsif detector = "00" then
                    next_state <= s1;
                end if;
            when s2 =>
                if detector = "11" then
                    next_state <= s1hat;
                end if;
        end case;
    end process;

    output: process (detector, current_state)
    begin
        danger <= '1';
        if current_state = s0 and detector = "00" then
            danger <= '0';
        end if;
    end process;
end Behavioral;
```

Task 3: Miranda's state machine

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


entity mirandas_state_machine is
    Port ( clk      : in  STD_LOGIC;
           n_rst    : in  STD_LOGIC;
           i        : in  STD_LOGIC;
           o        : out STD_LOGIC);
end mirandas_state_machine;

architecture Behavioral of mirandas_state_machine is
    type state_type is (s0, s1, s2, s3, s4);
    signal current_state, next_state: state_type;
begin
    clocking: process (clk)
    begin
        if rising_edge(clk) then
            if n_rst = '0' then
                current_state <= s0;
            else
                current_state <= next_state;
            end if;
        end if;
    end process;

    process (current_state, i)
    begin
        next_state <= current_state;
        case current_state is
            when s0 =>
                if i = '0' then
                    next_state <= s1;
                end if;
            when s1 =>
                if i = '0' then
                    next_state <= s2;
                else
                    next_state <= s0;
                end if;
            when s2 =>
                if i = '1' then
                    next_state <= s3;
                end if;
            when s3 =>
                if i = '0' then
                    next_state <= s4;
                else
                    next_state <= s0;
                end if;
            when s4 =>
                if i = '0' then
                    next_state <= s2;
                else
                    next_state <= s0;
                end if;
        end case;
    end process;

    process (current_state, i)
    begin
        o <= '0';
        if current_state = s4 and i = '0' then
            o <= '1';
        end if;
    end process;
end Behavioral;

end Behavioral;
```