In The Name of God

COMPUTER ENGINEERING DEPARTMENT OF AMIRKABIR UNIVERSITY OF TECHNOLOGY

Design Automation Homework - 3

Iman Tabrizian (9331032)

April 10, 2017

1 PROBLEM 5

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

entity counter is
    port(
            direction: in std_logic;
            reset: in std_logic;
            clk: in std_logic;
            number: out std_logic_vector(4 downto 0)
        );
end entity;

architecture behavorial of counter is
    type STATE_TYPE is (C0, C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12, C13, C14, C
    signal state: STATE_TYPE := C0;
begin
    process(clk)
        variable counting: integer := 0;
    begin
        if(clk'event and clk = '1') then
            if(reset = '1') then
```

```
        counting := 0;
        number <= std_logic_vector(to_unsigned(counting, 5));
    else
        number <= std_logic_vector(to_unsigned(counting, 5));
        case state is
            when C0 =>
                if(direction = '1') then
                    counting := counting + 1;
                    state <= C1;
                else
                    counting := 18;
                    state <= C18;
                end if;
            when C1 =>
                if(direction = '1') then
                    counting := counting + 1;
                    state <= C2;
                else
                    report integer'image(counting);
                    counting := counting - 1;
                    state <= C0;
                end if;
            when C2 =>
                if(direction = '1') then
                    counting := counting + 1;
                    state <= C3;
                else
                    counting := counting - 1;
                    state <= C1;
                end if;
            when C3 =>
                if(direction = '1') then
                    counting := counting + 1;
                    state <= C4;
                else
                    counting := counting - 1;
                    state <= C2;
                end if;
            when C4 =>
                if(direction = '1') then
                    counting := counting + 1;
                    state <= C5;
                else
                    counting := counting - 1;
```

```
            state <= C3;
        end if;
    when C5 =>
        if(direction = '1') then
            counting := counting + 1;
            state <= C6;
        else
            counting := counting - 1;
            state <= C4;
        end if;
    when C6 =>
        if(direction = '1') then
            counting := counting + 1;
            state <= C7;
        else
            counting := counting - 1;
            state <= C5;
        end if;
    when C7 =>
        if(direction = '1') then
            counting := counting + 1;
            state <= C8;
        else
            counting := counting - 1;
            state <= C6;
        end if;
    when C8 =>
        if(direction = '1') then
            counting := counting + 1;
            state <= C9;
        else
            counting := counting - 1;
            state <= C7;
        end if;
    when C9 =>
        if(direction = '1') then
            counting := counting + 1;
            state <= C10;
        else
            counting := counting - 1;
            state <= C8;
        end if;
    when C10 =>
        if(direction = '1') then
```

```
            counting := counting + 1;
            state <= C11;
        else
            counting := counting - 1;
            state <= C9;
        end if;
    when C11 =>
        if(direction = '1') then
            counting := counting + 1;
            state <= C12;
        else
            counting := counting - 1;
            state <= C10;
        end if;
    when C12 =>
        if(direction = '1') then
            counting := counting + 1;
            state <= C13;
        else
            counting := counting - 1;
            state <= C11;
        end if;
    when C13 =>
        if(direction = '1') then
            counting := counting + 1;
            state <= C14;
        else
            counting := counting - 1;
            state <= C12;
        end if;
    when C14 =>
        if(direction = '1') then
            counting := counting + 1;
            state <= C15;
        else
            counting := counting - 1;
            state <= C13;
        end if;
    when C15 =>
        if(direction = '1') then
            counting := counting + 1;
            state <= C16;
        else
            counting := counting - 1;
```

```
                        state <= C14;
                    end if;
                when C16 =>
                    if(direction = '1') then
                        counting := counting + 1;
                        state <= C17;
                    else
                        counting := counting - 1;
                        state <= C15;
                    end if;
                when C17 =>
                    if(direction = '1') then
                        counting := counting + 1;
                        state <= C18;
                    else
                        counting := counting - 1;
                        state <= C16;
                    end if;
                when C18 =>
                    if(direction = '1') then
                        counting := 0;
                        state <= C0;
                    else
                        counting := counting - 1;
                        state <= C17;
                    end if;
            end case;
        end if;
    end if;
  end process;
end behavorial;
```

No of states, in state machine is equal to 18. For each number we have a seperate state

2 Problem 6

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

entity fsm is
    port(
            x: in std_logic;
            res: out std_logic;
            clk: in std_logic;
```

```
            rst: in std_logic
        );
end entity;

architecture process_3 of fsm is
    type STATE_TYPE is (A, B, C, D, RST_ST);
    signal current_state, next_state: STATE_TYPE;
begin
    process(clk)
    begin
        if(rising_edge(clk)) then
            if(rst = '1') then
                current_state <= RST_ST;
            else
                current_state <= next_state;
            end if;
        end if;
    end process;

    process(current_state, x)
    begin
        case current_state is
            when RST_ST =>
                next_state <= A;
            when A =>
                if(x = '1') then
                    next_state <= B;
                else
                    next_state <= A;
                end if;
            when B =>
                if(x = '1') then
                    next_state <= C;
                else
                    next_state <= A;
                end if;
            when C =>
                if(x = '1') then
                    next_state <= D;
                else
                    next_state <= A;
                end if;
            when D =>
                if(x = '1') then
```

```
                    next_state <= D;
                else
                    next_state <= A;
                end if;
            end case;
        end process;

        process(current_state)
        begin
            case current_state is
                when RST_ST =>
                    res <= '0';
                when A =>
                    res <= '0';
                when B =>
                    res <= '0';
                when C =>
                    res <= '0';
                when D =>
                    res <= '1';
            end case;
        end process;
    end process_3;

    architecture process_2 of fsm is
        type STATE_TYPE is (A, B, C, D, RST_ST);
        signal current_state: STATE_TYPE;
    begin

        process(clk)
        begin
            if(rising_edge(clk)) then
                if(rst = '1') then
                    current_state <= RST_ST;
                else
                    case current_state is
                        when RST_ST =>
                            current_state <= A;
                        when A =>
                            if(x = '1') then
                                current_state <= B;
                            else
                                current_state <= A;
                            end if;
```

```
                    when B =>
                        if(x = '1') then
                            current_state <= C;
                        else
                            current_state <= A;
                        end if;
                    when C =>
                        if(x = '1') then
                            current_state <= D;
                        else
                            current_state <= A;
                        end if;
                    when D =>
                        if(x = '1') then
                            current_state <= D;
                        else
                            current_state <= A;
                        end if;
                end case;
            end if;
        end if;
    end process;

    process(current_state)
    begin
        case current_state is
            when RST_ST =>
                res <= '0';
            when A =>
                res <= '0';
            when B =>
                res <= '0';
            when C =>
                res <= '0';
            when D =>
                res <= '1';
        end case;
    end process;
end process_2;
```

3 Problem 7

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

entity remainder is
    port(
            input: in std_logic;
            clk: in std_logic;
            output: out integer
        );
end entity;

architecture behavorial of remainder is
    type STATE_TYPE is (START, s0, s1, s2, s3, s4, s5, FINAL);
    signal state: STATE_TYPE := START;
begin
    process(clk)
        variable length: integer := 0;
        variable remain: integer := 0;
    begin
        case state is
            when START =>
                if(input = '1') then
                    state <= s0;
                else
                    state <= START;
                end if;
                remain := 0;
            when s0 =>
                length := length + 1;
                if(length = 8) then
                    state <= FINAL;
                else
                    if(input = '1') then
                        state <= s3;
                    else
                        state <= s1;
                    end if;
                end if;
                remain := 0;
            when s1 =>
                length := length + 1;
```

```
        if(length = 8) then
            state <= FINAL;
        else
            if(input = '1') then
                state <= s4;
            else
                state <= s0;
            end if;
        end if;
        remain := 0;
    when s2 =>
        length := length + 1;
        if(length = 8) then
            state <= FINAL;
        else
            if(input = '1') then
                state <= s5;
            else
                state <= s3;
            end if;
        end if;
        remain := 1;
    when s3 =>
        if(length = 8) then
            state <= FINAL;
        else
            if(input = '1') then
                state <= s0;
            else
                state <= s3;
            end if;
        end if;
        remain := 1;
    when s4 =>
        length := length + 1;
        if(length = 8) then
            state <= FINAL;
        else
            if(input = '1') then
                state <= s1;
            else
                state <= s5;
            end if;
        end if;
```

```
                remain := 2;
            when s5 =>
                if(length = 8) then
                    state <= FINAL;
                else
                    if(input = '1') then
                        state <= s2;
                    else
                        state <= s4;
                    end if;
                end if;
                remain := 2;
            when FINAL =>
                if(input = '0') then
                    output <= remain;
                else
                    output <= 3;
                end if;

        end case;

    end process;
end behavorial;
```

From the benefits of this encoding is that the number of FFs are minimum and disadvantage of this method is that if a noise occurs it's more difficult to detect it.

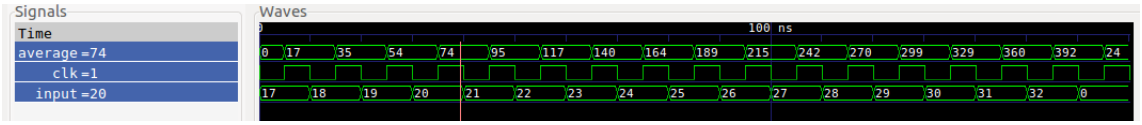
4 Problem 8

You can find the state machine in the attachments

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

entity averager is
    port(
            input: in natural range 15 to 40;
            average: inout integer := 0;
            clk: in std_logic
        );
end averager;

architecture behavorial of averager is
    function avg (a, b: in natural) return integer is
```

```
    begin
        if(a = 0) then
            return b / 16;
        else
            return a + b;
        end if;

    end avg;
begin
    process(clk)
    begin
        if(rising_edge(clk)) then
            average <= avg(input, average);
        end if;
    end process;

end behavorial;
```

Below is the figure of simulation result:



5 Problem 9

You can find the state machine in the attachments


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

entity sequence_detector is
    port(
            input: in std_logic_vector(1 downto 0);
            clk: in std_logic;
            output: out std_logic_vector(1 downto 0)
        );
end sequence_detector;
```

```
architecture behavorial of sequence_detector is
    type STATE_TYPE is (s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12);
    signal state: STATE_TYPE := s0;
    constant A: std_logic_vector(1 downto 0) := "00";
    constant T: std_logic_vector(1 downto 0) := "01";
    constant C: std_logic_vector(1 downto 0) := "10";
    constant G: std_logic_vector(1 downto 0) := "11";
begin
    process(clk)
    begin
        if (rising_edge(clk)) then
            case state is
                when s0 =>
                    if(input = A) then
                        state <= s12;
                    elsif(input = T) then
                        state <= s0;
                    elsif(input = C) then
                        state <= s0;
                    elsif(input = G) then
                        state <= s1;
                    end if;
                    output <= "00";
                    report integer'image(0);
                when s1 =>
                    if(input = A) then
                        state <= s2;
                    elsif(input = T) then
                        state <= s0;
                    elsif(input = C) then
                        state <= s0;
                    elsif(input = G) then
                        state <= s1;
                    end if;
                    output <= "00";
                    report integer'image(1);
                when s2 =>
                    if(input = A) then
                        state <= s12;
                    elsif(input = T) then
                        state <= s3;
                    elsif(input = C) then
                        state <= s11;
```

```
        elsif(input = G) then
            state <= s1;
        end if;
        output <= "00";
        report integer'image(2);
    when s3 =>
        if(input = A) then
            state <= s4;
        elsif(input = T) then
            state <= s0;
        elsif(input = C) then
            state <= s0;
        elsif(input = G) then
            state <= s1;
        end if;
        output <= "00";
        report integer'image(3);
    when s4 =>
        if(input = A) then
            state <= s12;
        elsif(input = T) then
            state <= s5;
        elsif(input = C) then
            state <= s11;
        elsif(input = G) then
            state <= s1;
        end if;
        output <= "00";
        report integer'image(4);
    when s5 =>
        if(input = A) then
            state <= s6;
        elsif(input = T) then
            state <= s0;
        elsif(input = C) then
            state <= s0;
        elsif(input = G) then
            state <= s1;
        end if;
        output <= "00";
        report integer'image(5);
    when s6 =>
        if(input = A) then
            state <= s12;
```

```
        elsif(input = T) then
            state <= s7;
        elsif(input = C) then
            state <= s11;
        elsif(input = G) then
            state <= s1;
        end if;
        output <= "00";
        report integer'image(6);
    when s7 =>
        if(input = A) then
            state <= s12;
        elsif(input = T) then
            state <= s0;
        elsif(input = C) then
            state <= s0;
        elsif(input = G) then
            state <= s8;
        end if;
        output <= "00";
        report integer'image(7);
    when s8 =>
        if(input = A) then
            state <= s12;
        elsif(input = T) then
            state <= s0;
        elsif(input = C) then
            state <= s9;
        elsif(input = G) then
            state <= s1;
        end if;
        output <= "00";
        report integer'image(8);
    when s9 =>
        if(input = A) then
            state <= s12;
        elsif(input = T) then
            state <= s0;
        elsif(input = C) then
            state <= s0;
        elsif(input = G) then
            state <= s1;
        end if;
        output <= "01";
```

```
                report integer'image(9);
            when s10 =>
                if(input = A) then
                    state <= s2;
                elsif(input = T) then
                    state <= s0;
                elsif(input = C) then
                    state <= s11;
                elsif(input = G) then
                    state <= s1;
                end if;
                output <= "10";
                report integer'image(10);
            when s11 =>
                if(input = A) then
                    state <= s12;
                elsif(input = T) then
                    state <= s0;
                elsif(input = C) then
                    state <= s0;
                elsif(input = G) then
                    state <= s10;
                end if;
                output <= "00";
                report integer'image(11);
            when s12 =>
                if(input = A) then
                    state <= s12;
                elsif(input = T) then
                    state <= s0;
                elsif(input = C) then
                    state <= s11;
                elsif(input = G) then
                    state <= s1;
                end if;
                output <= "00";
                report integer'image(12);
        end case;
    end if;

end process;
end behavorial;
```

6 Problem 10

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

entity remainder is
    port(
            input: in std_logic;
            clk: in std_logic;
            output: out integer
        );
end entity;

architecture behavorial of remainder is
    type STATE_TYPE is (START, s0, s1, s2, s3, s4, s5, FINAL);
    signal state: STATE_TYPE := START;
begin
    process(clk)
        variable length: integer := 0;
        variable remain: integer := 0;
    begin
        case state is
            when START =>
                if(input = '1') then
                    state <= s0;
                else
                    state <= START;
                end if;
                remain := 0;
            when s0 =>
                length := length + 1;
                if(length = 8) then
                    state <= FINAL;
                else
                    if(input = '1') then
                        state <= s3;
                    else
                        state <= s1;
                    end if;
                end if;
                remain := 0;
            when s1 =>
                length := length + 1;
```

```
        if(length = 8) then
            state <= FINAL;
        else
            if(input = '1') then
                state <= s4;
            else
                state <= s0;
            end if;
        end if;
        remain := 0;
    when s2 =>
        length := length + 1;
        if(length = 8) then
            state <= FINAL;
        else
            if(input = '1') then
                state <= s5;
            else
                state <= s3;
            end if;
        end if;
        remain := 1;
    when s3 =>
        if(length = 8) then
            state <= FINAL;
        else
            if(input = '1') then
                state <= s0;
            else
                state <= s3;
            end if;
        end if;
        remain := 1;
    when s4 =>
        length := length + 1;
        if(length = 8) then
            state <= FINAL;
        else
            if(input = '1') then
                state <= s1;
            else
                state <= s5;
            end if;
        end if;
```

```
                remain := 2;
            when s5 =>
                if(length = 8) then
                    state <= FINAL;
                else
                    if(input = '1') then
                        state <= s2;
                    else
                        state <= s4;
                    end if;
                end if;
                remain := 2;
            when FINAL =>
                if(input = '0') then
                    output <= remain;
                else
                    output <= 3;
                end if;

        end case;

    end process;
end behavorial;
```

We used only one process.