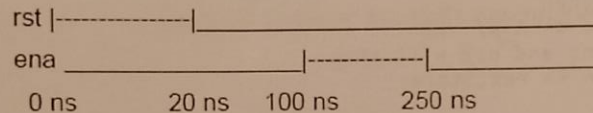


HÖHERE TECHNISCHE BUNDESLEHRANSTALT HOLLABRUNN

ABTEILUNG ELEKTRONIK UND TECHNISCHE INFORMATIK

Datum: 9. Jan 2015	Art der Leistungsfeststellung: Test 1		Pos.:	Wertung	Bewert.	Pos.:	Wertung	Bewert.
Klasse: 4BHEL	Gegenstand: HWE	Gruppe: A	1	10	9	9		
Zeit: 20 min.	Lehrer: Dipl.-Ing. Wolfgang Bodner		2	40	20	10		
			3	30	30	11		
			4	20	20	12		
Note: 3	Name des Schülers: Lukas Graussam	Kat.Nr.:	5			13		
			6			14		
	0 - 49 Nicht genügend 50 - 62 Genügend 63 - 75 Befriedigend 76 - 89 Gut 90 - 100 Sehr Gut		7			15		
			8			16		
		Summe 100 Punkte			69			

1. (10 Punkte) Beschreiben Sie den „Stimulus Process“ folgende Signale für eine Testbench. (rst ist mit '1' und ena mit '0' initialisiert.)



```

STIM_PROC: process
begin
    wait for 10ns; ✓
    wait for 10ns; ✓
    rst <= '0'; ✓
    wait for 80ns; ✓
    ena <= '1'; ✓
    wait for 150ns; ✓
    ena <= '0'; ✓
    
```

wait;

9

```
end process STIM_PROC;
```

2. Skizzieren Sie das Zustandsdiagramm entsprechend der folgenden Beschreibung eines Automaten.

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity automat1 is
    Port ( E0 : in STD_LOGIC;
          E1 : in STD_LOGIC;
          clk : in STD_LOGIC;
          E2 : in STD_LOGIC;
          TRIG : out STD_LOGIC;
          Q1 : out STD_LOGIC;
          Q2 : out STD_LOGIC;
          rst : in STD_LOGIC);
end automat1;

architecture Behavioral of automat1 is
    type state_type is (A_s, B_s, C_s, D_s);
    signal curState, nextState: state_type;

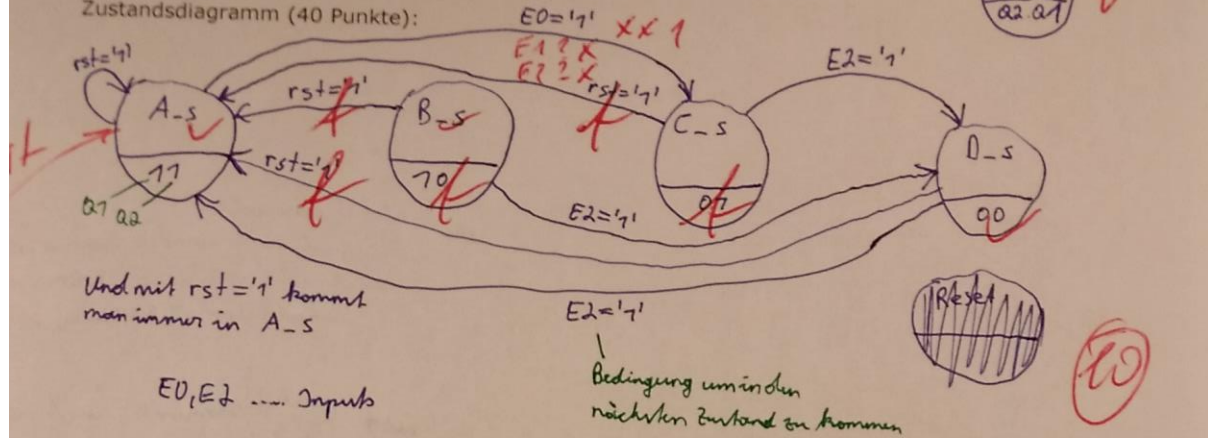
begin

    process (clk, rst)
    begin
        if rst = '1' then
            curState <= A_s;
        elsif clk'event and clk = '1' then
            curState <= nextState;
        end if;
    end process;

    process (curState, E0, E1, E2, rst)
    begin
        nextState <= curState;
        Q1 <= '1';
        Q2 <= '1';
        case curState is
            when A_s =>
                if E0 = '1' then
                    nextState <= C_s;
                end if;
            when B_s =>
                if E2 = '1' then
                    nextState <= D_s;
                end if;
                Q2 <= '0';
            when C_s =>
                if E2 = '1' then
                    nextState <= D_s;
                end if;
                Q1 <= '0';
            when D_s =>
                if E2 = '1' then
                    nextState <= A_s;
                end if;
                Q1 <= '0';
                Q2 <= '0';
        end case;
    end process;
end Behavioral;
```

rst async. !!

Zustandsdiagramm (40 Punkte):



Und mit $rst = '1'$ kommt man immer in A-s

$E0, E2$ Inputs

$E2 = '1'$

Bedingung uminden nächsten Zustand zu kommen

In anderen Fällen (z.B.: wenn $E0 = '1'$ wird während man in C-s ist) bleibt der Automat im selben Zustand

3. Ein kombinatorischer Prozess wird durch folgenden Code beschrieben (Syntax ist richtig!).

```
p_sel: process ( A, B, C, D) - 1. Fehlerzeile
begin
case disp_cycle_reg is
when "0111" => disp_mx <= A;
when "1011" => disp_mx <= B;
when "1101" => disp_mx <= C;
when "1110" => disp_mx <= D;
```

----- 2. Fehlerzeile (Anweisung fehlt)

```
end case;
end process;
```

a. (10 Punkte) Korrigieren Sie die beiden Fehler.

1) In der Sensitivity list muss $disp_cycle_reg$ stehen. ✓

2) Es muss ein "when others" geben. ✓

b. (10 Punkte) Welche Auswirkung hat es, wenn der erste Fehler nicht korrigiert wird?

Beim Prozess wird nicht ausgeführt wenn sich $disp_cycle_reg$ ändert

c. (10 Punkte) Welche Auswirkung hat es, wenn der zweite Fehler nicht korrigiert wird?

Bei nicht angeführten Zuständen entsteht ein Fehler

4 nicht angeführte Fälle

4. (20 Punkte) Mit einem Prozess soll ein mit positiver Taktflanke gesteuertes D-FF mit Reset beschrieben werden. Vergleichen Sie die 4 angegebenen Versionen. Geben Sie an, welche sinnvoll sind und begründen Sie warum Sie eine Version für nicht sinnvoll halten.

```
dff_p1: process (clk, r)
begin
    if r='1' then q_old <= '0'; end if;
    if clk'event and clk='1' then
        q_old <= q_new;
    end if;
end process dff_p1;
```

~~also~~ - nicht sinnvoll
 elsif sollte verwendet werden, da der reset sonst nicht richtig funktioniert (überschrieben wird bei pos. Taktflanke) ✓

```
dff_p2: process (clk)
begin
    if clk'event and clk='1' then
        q_old <= q_new;
        if r='1' then q_old <= '0'; end if;
    end if;
end process dff_p2;
```

~~reset~~ - nicht sinnvoll, reset nicht im sensitivitätslist
 reset wird nur möglich wenn bei einer pos. Taktflanke ✓

```
dff_p3: process (clk, r)
begin
    if r='1' then
        q_old <= '0';
    elsif clk'event and clk='1' then
        q_old <= q_new;
    end if;
end process dff_p3;
```

- sinnvoll. ✓

```
dff_p4: process (clk, r)
begin
    if clk'event and clk='1' then
        q_old <= q_new;
    end if;
    if r='1' then
        q_old <= '0';
    end if;
end process dff_p4;
```

- um eine Zeile länger als die dritte Version aber auch sinnvoll. ✓

(20)