Laborversuch 1

Einführung Laborsystem und Entwurfssoftware / RTL

Vorbereitungsaufgaben :

V3)

a) Wie reagiert der Zähler counter4 auf das Taktsignal? Reagiert der Zähler auf den Pegel oder auf eine Flanke des Taktsignals?

⇨ Im Normalfall würde der *counter4* auf eine steigende Flanke und *q* um 2 inkrementiert.
Jedoch sind syntaktische /semantische Fehler enthalten, wodurch die Funktion nicht gegeben ist.

```
21      reg_proc:process (clk)   -- process for register function
22      begin
23          if reset='1' then
24              q <= (others => '0');
25          elsif ck'event and clk = '1' then   -- rising clock edge
26              q <= q;
27          end if;
28      end process;
29
30      nsd_proc:process (q)     -- process for next state decoder
31      begin
32          q_ns <= q + 2;
33      end process;
```

Zeile 25 : Stellt sicher dass der *counter4* auf eine steigende Flanke reagiert.
Zeile 32 : das Signal *q_ns* wird um 2 inkrementiert.

b) Wie reagiert der Zähler counter4 auf das Resetsignal? Reagiert der Zähler auf den Pegel oder auf eine Flanke des Resetsignals?

⇨ Sobald Reset auf 1 gesetzt wird (Zeile 23) , wird *q* (4 Bits) auf 0 gesetzt (Ausgänge). Wenn Reset auf 0 ist wird eine andere Bedingung geprüft.

```
21      reg_proc:process (clk)   -- process for register function
22      begin
23          if reset='1' then
24              q <= (others => '0');
25          elsif ck'event and clk = '1' then   -- rising clock edge
26              q <= q;
27          end if;
28      end process;
29
```

c) Wie heißen die Signale in der Testbench (nicht die Ports der Komponente counter4!), die an den Takt- und Reseteingang des Zählers angeschlossen sind?

⇨ *tb_clk* und *tb_reset* heißen die Signale. (Zeile 33 und 34)

```
30
31  begin
32      uut: counter4 port map (
33          clk               => tb_clk,
34          reset             => tb_reset,
35          q_out(3 downto 0) => tb_q_out(3 downto 0)
36      );
37
38
39      clk_process: process
```

d) In welchem der Testbench-Prozesse wird das Takt-, in welchem das Resetsignal generiert? Beschreiben Sie die Arbeitsweise der beiden Prozesse!

⇨ Der Takt wird im *clk_process* generiert: Dort wird zu Beginn der Takt auf 0 gesetzt nach 12.5ns (halbe Periode) wird er auf 1 gesetzt und schließlich wieder 12.5ns gewartet. Danach beginnt der Prozess erneut.

```
39      clk_process: process
40      begin
41          tb_clk <= '0';
42          wait for clk_period / 2;
43          tb_clk <= '1';
44          wait for clk_period / 2;
45      end process;
46
```

⇨ Reset wird im *stimulus_process* generiert: Dort wird zu Beginn der Reset auf 0 gesetzt nach 50ns (Periode * 2) wird er auf den invertierten Wert vom Anfang (1) gesetzt. Danach wartet der Prozess, solange bis die Simulation neu startet.

```
46
47      stimulus_process: process
48      begin
49          tb_reset <= reset_active_level;
50          wait for reset_active_time;
51          tb_reset <= not reset_active_level;
52          -- ...
53          wait;
54      end process;
55
```

e) Das FPGA auf dem Laborsystem kann extern mit einer Frequenz von 100 MHz getaktet werden (siehe Dokumentation zum BASYS3-Board). Welche Periodendauer des Taktes ergibt sich daraus? Wo könnten Sie diese Periodendauer im Quellcode der Testbench einstellen?

⇨ $\boldsymbol{T = 1 / f = 1 / 100\ MHz = 10\ ns}$
Einstellungen: *constant clk_period* : time= xxx ns.
Quellcode-Abschnitt: Zeile 25

```
24
25      constant clk_period : time := 25 ns; --40 Mhz
26      constant reset_active_time : time := clk_period * 2;
27      constant reset_active_level : std_logic := '1';
28
29
```

f) Welchen Signalpegel hat das von der Testbench generierte Reset-Signal in seiner aktiven Phase? Passt der Pegel des Reset-Impulses zu den Anforderungen der VHDL Komponente, die Sie testen möchten? Wo können Sie ggf. den Signalpegel des generierten Reset-Signals ändern?

⇨ In seiner aktiven Phase hat das Resetsignal eine 0 (vor Änderung). Nein es passt nicht, da der Reset **High-aktiv** ist, somit soll er in seiner aktiven Phase den Pegel 1 haben und in seiner nicht aktiven Phase 0.

```
1   reg_proc:process (clk, reset)    -- process for register functi
2   begin
3       if reset='1' then
4           q <= (others => '0');
5       elsif clk'event and clk = '1' then  -- rising clock edge |
6           q <= q_ns;
7       end if;
8   end process;
```

Zeile 3+4 (in counter4.vhd) => beweist, dass der Reset High-Aktiv sein muss.

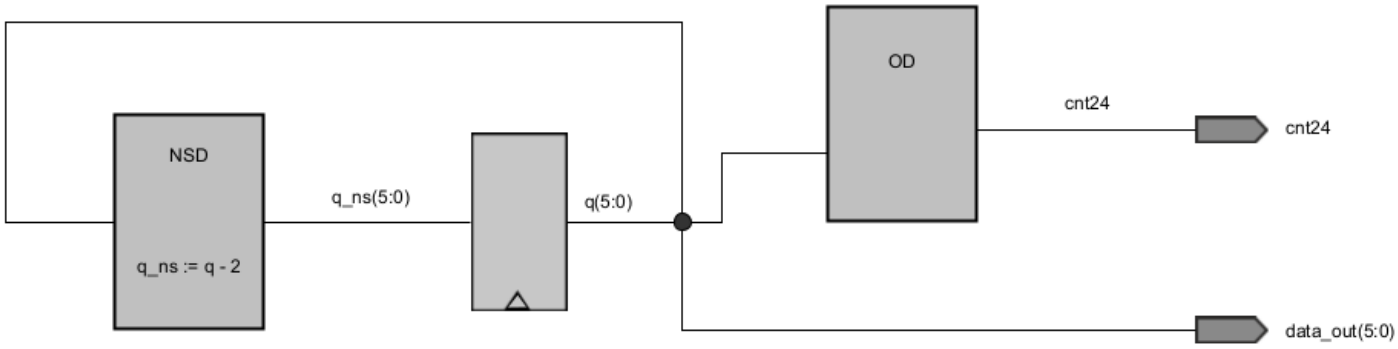
⇨ Man kann in Zeile 27 (tb_counter4.vhd) der Konstante:
constant reset_active_level einen gewünschten Wert zuweisen.

```
24
25      constant clk_period : time := 25 ns; --40 Mhz
26      constant reset_active_time : time := clk_period * 2;
27      constant reset_active_level : std_logic := '1';
28
29
```

g) Wie können Sie den Taktgeber des Peripherieboards mit einem FPGAAnschlusspin verbinden?

⇨ Um den Taktgeber des Peripherieboards mit einem FPGAAnschlusspin zu verbinden, kann man die sogenannte *Constraints (Randbedingungen)* festlegen.

V4)



Reset auf 0 , asynchron , 0-Aktiv

OD:

if q = 24 then
cnt24 := 1
else
cnt24 := 0
end if

• Schriftliche Beantwortung der Fragen zum Vivado-Einführungskurs

Kapitel 4 (Seite 8):

Siehe Vorbereitungsaufgabe V3) a) bis f)

Kapitel 5 (Seite 11):

o In welchem Prozess ist das Zählerregister modelliert, in welchem das Überführungsschaltnetz?

⇨ Das Zählerregister ist in *reg_proc:process* modelliert.

Das Überführungsschaltnetz ist in *nsd_proc:process* modelliert.

o Wo wird im Registerprozess der vom Überführungsschaltnetz berechnete Wert übernommen?

⇨ auf Zeile 26

```
21  reg_proc:process (clk, reset)   -- process for register functi
22  begin
23      if reset='1' then
24          q <= (others => '0');
25      elsif clk'event and clk = '1' then  -- rising clock edge |
26          q <= q_ns;
27      end if;
28  end process;
29
```

o Wo wird der nächste Zählerstand berechnet?

⇨ Auf Zeile 32

```
29
30  nsd_proc:process (q)    -- process for next state decoder
31  begin
32      q_ns <= q + 1;
33  end process;
34
```

Kapitel 5 (Seite 12):

o Wie viele Bits hat der Zähler? Was ist demnach der größtmögliche Zählerstand?

- ⇨ Der Zähler hat 4 Bits. Das heißt, der größtmögliche Zählerstand ist 2^4= 16

o Zu welchem Zeitpunkt wird also ein Überlauf des Zählers erfolgen, wenn Sie die Dauer des Resetimpulses mit einkalkulieren?

Mit jedem Takt (steigende Taktflanke) wird der Zählerstand um Eins erhöht. Das heißt, um den größtmögliche Zählerstand zu erreichen muss die Simulation mindestens 16 Takte enthalten.

Darüber hinaus, am Anfang der Simulation wird der Reset auf ‚1' gesetzt, welche eine Wartezeit von 2 Perioden verursacht, der 2 Takte entspricht.

- ⇨ Nach 18 (16 + 2) Takten wird ein Überlauf des Zählers stattfinden.
- ⇨ Und weil ein Takt 25ns dauert, daraus ergibt sich **450ns** (18 Take * 25ns)

Kapitel 5 (Seite 13):

☐ Ändert sich der Zählerstand mit jeder steigenden Flanke des Eingangstakts?

- ⇨ Ja

☐ Arbeitet der Zähler korrekt (aufsteigend, Einerschritte)?

- ⇨ Ja

☐ Welcher Zählerstand wird einen Takt nach dem Erreichen des höchsten Zählerstands angezeigt (Verhalten bei Überlauf)?

Nach dem Erreichen des höchsten Zählerstands hat der Zählerstand einen Wert von ‚0' und mit jedem weiteren Takt wird der Zähler um Eins erhöht.

- ⇨ Ein Überlauf agiert als ein Reset.

☐ Auf welchen Wert wird der Zähler durch das Reset-Signal zurückgesetzt?

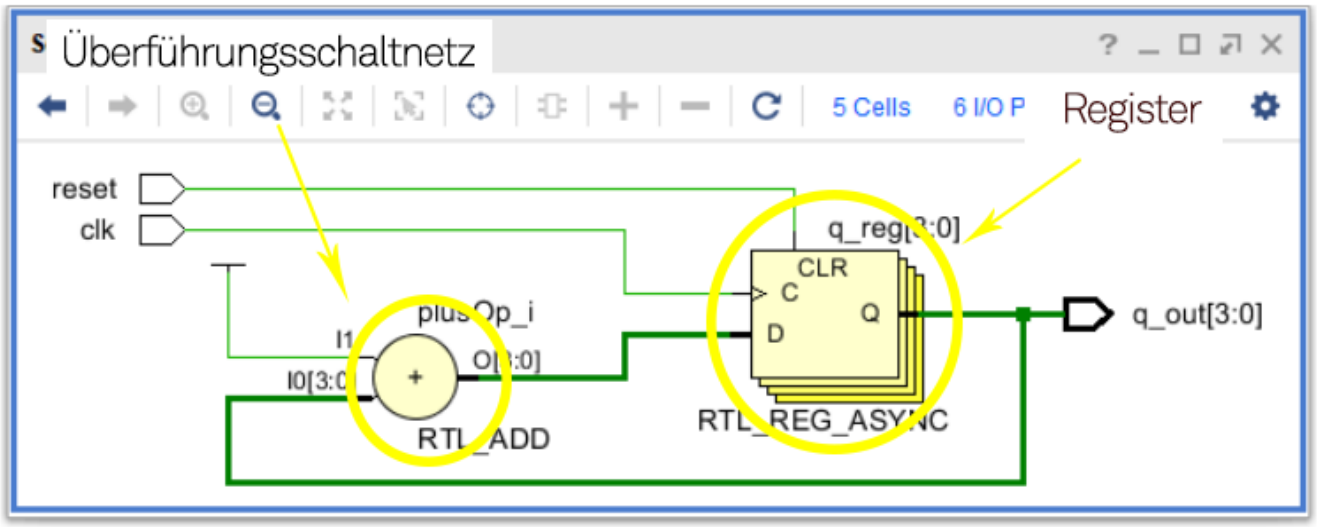
- ⇨ 0000_2

☐ Ist der Reset-Eingang des Zählers 0- oder 1-aktiv (oder anders formuliert: Durch welchen Pegel des Reset-Signals wird der Zähler auf 0 zurückgesetzt)?

- ⇨ der Reset-Eingang des Zählers ist 1-Aktiv.

Kapitel 7 (Seite 16):

☐ Welcher der dargestellten Blöcke ist das Register, welcher das Überführungsschaltnetz?



☐ Wie viele Bits hat das Register?

⇨ *q_ref[3:0]* => 4 Bits

☐ Ist im Überführungsschaltnetz die richtige Funktion implementiert?

⇨ Ja. (Addition)

☐ Sind die Ein- und Ausgangssignale vollständig und besitzen Sie die richtige Polarität?

⇨ Ja

Kapitel 8 (Seite 17):

☐ Welcher Prozess des Zählers muss sensitiv auf das Reset-Signal sein?

⇨ Der Prozess *reg_proc:process* muss auf das Reset-Signal sein.

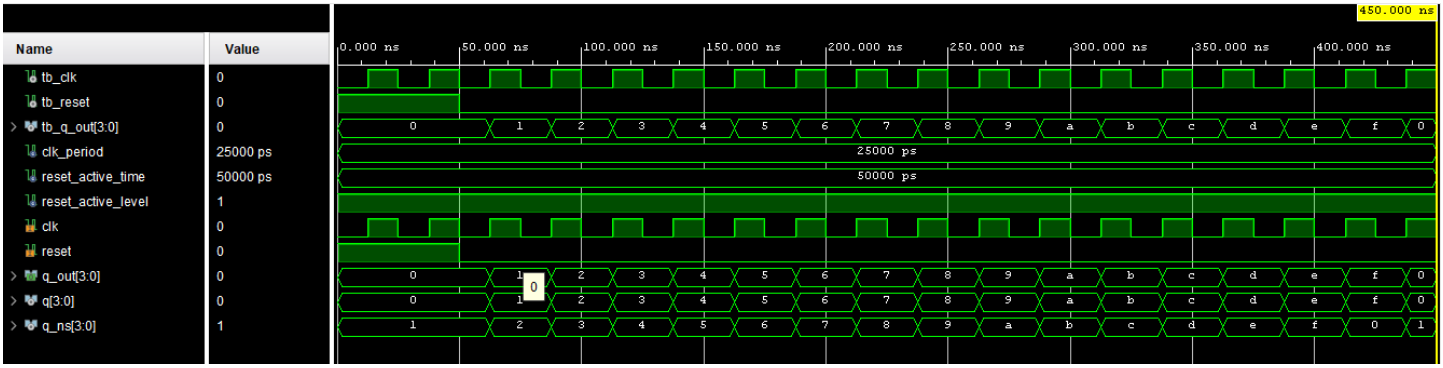
- **Korrigierter VHDL-Code des 4-Bit-Zählers aus dem Vivado-Einführungskurs1**

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4
5
6  entity counter4 is
7      port (
8          clk   : in  std_logic;
9          reset : in  std_logic;
10         q_out : out std_logic_vector(3 downto 0)
11     );
12 end counter4;
13
14
15 architecture beh of counter4 is
16     signal q, q_ns: unsigned(3 downto 0);
17
18 begin   -- beh
19     q_out <= std_logic_vector(q);   -- connect internal signals to output ports
20
21     reg_proc:process (clk, reset)   -- process for register function
22     begin
23         if reset='1' then
24             q <= (others => '0');
25         elsif clk'event and clk = '1' then  -- rising clock edge | what is clk'event ??
26             q <= q_ns;
27         end if;
28     end process;
29
30     nsd_proc:process (q)     -- process for next state decoder
31     begin
32         q_ns <= q + 1;
33     end process;
34
35 end beh;
```

- **VHDL-Code der Testbench zum 4-Bit-Zähler aus dem Vivado-Einführungskurs1**

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


entity tb is
end entity tb;

architecture struct of tb is

    component counter4 is -- this is should have the same name as the Entity in the main file.
        port (
            clk   : in std_logic;
            reset : in std_logic;
            q_out : out std_logic_vector(3 downto 0)
        );
    end component;

    signal tb_clk   : std_logic;
    signal tb_reset : std_logic;
    signal tb_q_out : std_logic_vector(3 downto 0);

    constant clk_period : time := 25 ns; --40 Mhz
    constant reset_active_time : time := clk_period * 2;
    constant reset_active_level : std_logic := '1';

begin
    uut: counter4 port map (
        clk              => tb_clk,
        reset            => tb_reset,
        q_out(3 downto 0) => tb_q_out(3 downto 0)
    );


    clk_process: process
    begin
        tb_clk <= '0';
        wait for clk_period / 2;
        tb_clk <= '1';
        wait for clk_period / 2;
    end process;

    stimulus_process: process
    begin
        tb_reset <= reset_active_level;
        wait for reset_active_time;
        tb_reset <= not reset_active_level;
        -- ...
        wait;
    end process;

end struct;
```

• Screenshot der Signalverläufe in der Simulation2



• Constraints-Datei constr.xdc

```
set_property PACKAGE_PIN V19 [get_ports {q_out[3]}]
set_property PACKAGE_PIN U19 [get_ports {q_out[2]}]
set_property PACKAGE_PIN E19 [get_ports {q_out[1]}]
set_property PACKAGE_PIN U16 [get_ports {q_out[0]}]
set_property PACKAGE_PIN A16 [get_ports clk]
set_property PACKAGE_PIN U18 [get_ports reset]
set_property IOSTANDARD LVCMOS33 [get_ports {q_out[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {q_out[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {q_out[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {q_out[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports reset]

create_clock -period 10.000 -name clk -waveform {0.000 5.000} [get_ports clk]

```

• Screenshot des Fensters Project Summary