UNIVERSITEIT
GENT

Benjamin Fraeyman (+Matisse Bonvin)
Ingebedde Systemen – Groep 3
ICT (groep 34)

LABRESULTS

2017-2018

Contents

Inleiding

Het doel van dit labo is inzicht te verwerven over de werking van VHDL. Doorheen het semester voeren we verscheidene labos uit om onze kennis te verbreden. We leren werken met de Artix-7 FPGA. Het doel van deze labresults is om onze bevindingen en opmerking neer te pennen.

Materiaal en Software

Digilent Basys 3 Artix-7 FPGA: het bordje gebruikt om te testen.

Vivado: simulatie tool, gebruikt voor het effectieve programmeren.

Sigasi: handige tool voor editten van code.

Lab 1: 4-bit comparator

Deel 1: Klassieke comparator

Opgave

Ontwerp een 4-bit comparator met 2 woorden A en B.

Deze wordt opgemaakt in VHDL en er wordt een behavioral- en PIT-simulatie uitgevoerd om de correcte werking te testen. Daarna wordt de maximale frequentie bepaald en wordt er gekeken hoeveel resources er gebruikt worden.

VHDL beschrijving

```
entity Comparator is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           AGTB : out STD_LOGIC;
           AEQB : out STD_LOGIC;
           ALTB : out STD_LOGIC;
           EPEQ : out STD_LOGIC);
end Comparator;

architecture Behavioral of Comparator is
begin
    AGTB <= '1' when A>B else '0';
    AEQB <= '1' when A=B else '0';
    ALTB <= '1' when A<B else '0';
    EPEQ <= '1' when (A(0) XOR A(1) XOR A(2) XOR A(3)) = (B(0) XOR B(1) XOR B(2)
XOR B(3)) else '0';
end Behavioral;
```

Redenering

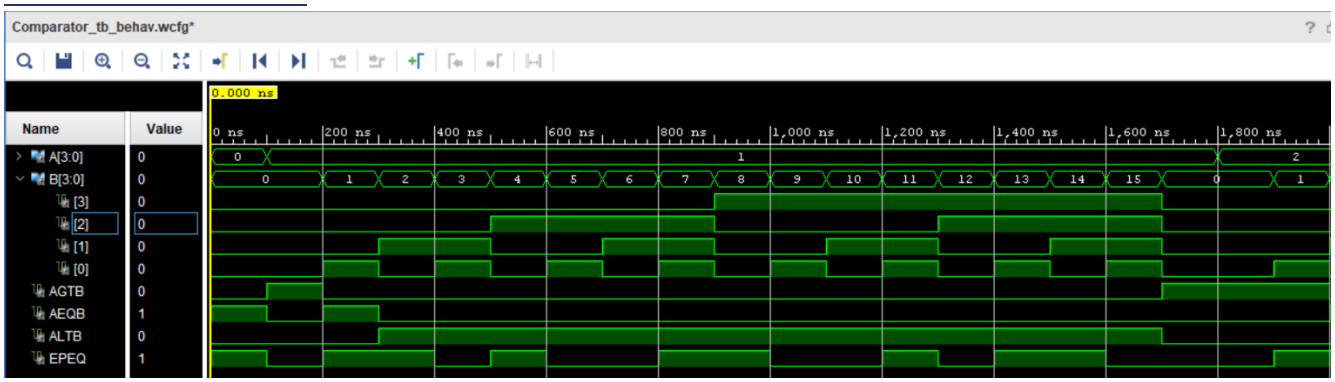
Een comparator is een vergelijker. Een digitale comparator is een schakeling die twee n-bit woorden met elkaar vergelijkt.
De uitgang van AGTB wordt hoog als de bits van woord A groter zijn dan de bits van woord B.
De uitgang van AEQB wordt hoog als de bits van woord A gelijk zijn aan de bits van woord B.
De uitgang van ALTB wordt hoog als de bits van woord A kleiner zijn dan de bits van woord B.
De uitgang van EPEQ wordt hoog als de pariteit van de bits van woord A gelijk zijn aan de pariteit van de bits van woord B.
Er wordt onmiddellijk een waarde toegekent aan de uitgangen. Elke uitgang staat los van de andere.

TestBench

```
architecture Behavioral of Comparator_tb is
    COMPONENT Comparator
    PORT(
            A : IN std_logic_vector(3 downto 0);
            B : IN std_logic_vector(3 downto 0);
            AGTB : OUT std_logic;
            AEQB : OUT std_logic;
            ALTB : OUT std_logic;
            EPEQ : OUT std_logic
        );
    END COMPONENT;

signal A : std_logic_vector(3 downto 0) := "0000";
```

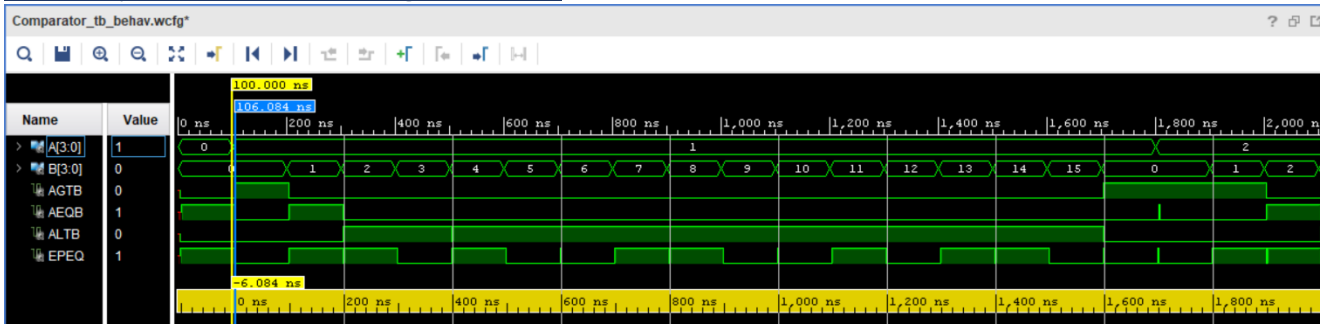
```
signal B : std_logic_vector(3 downto 0) := "0000";
signal AGTB :  std_logic:='0';
signal AEQB :  std_logic:='0';
signal ALTB :  std_logic:='0';
signal EPEQ :  std_logic:='0';

begin
    uut: Comparator PORT MAP (
        A => A,
        B => B,
        AGTB => AGTB,
        AEQB => AEQB,
        ALTB => ALTB,
        EPEQ => EPEQ
        );

    stim_proc: process
    begin
        for i in 0 to 15 loop
                WAIT FOR 100ns;
                A <= STD_LOGIC_VECTOR (A + '1');
                for j in 0 to 15 loop
                        WAIT FOR 100ns;
                        B <= STD_LOGIC_VECTOR (B + '1');
                END LOOP;
        END LOOP;
        wait;
    end process;
end Behavioral;
```

Behavioral simulatie



Figuur 1: Behavioral labo1 deel1

Post implementation timing simulatie



Figuur 2: PIT labo1 deel1

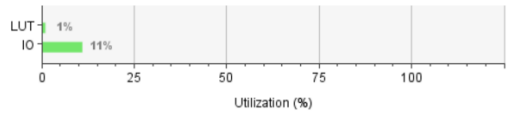
Maximale werkfrequentie

We zien op figuur 2 dat er een vertraging is van 6.084ns -> 1/(6.084) = 164.366MHz.

Resources

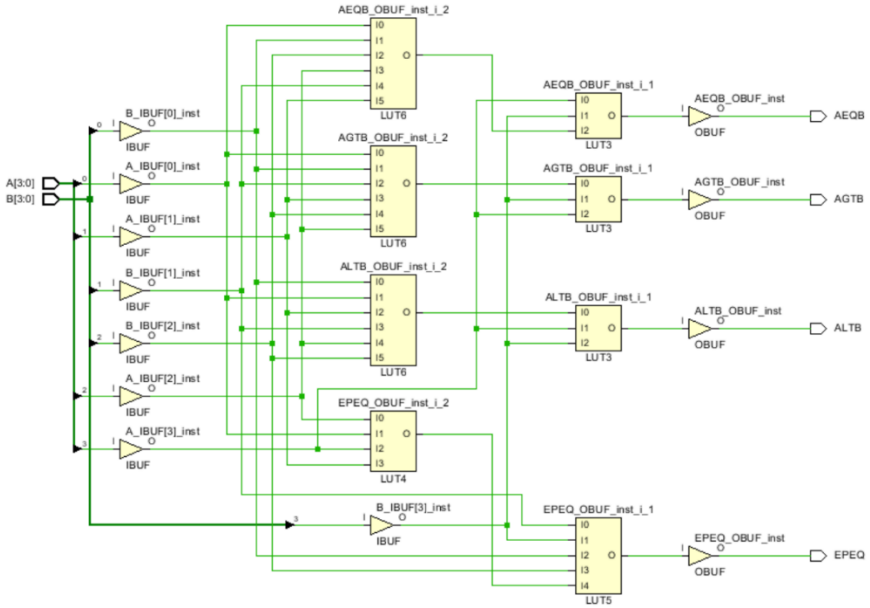
Figuur 3 toont het aantal resources dat de code benut.

Resource	Utilization	Available	Utilization %
LUT	7	20800	0.03
IO	12	106	11.32



Figuur 3: Resources labo1 deel1

Schematic



Figuur 4: Schematic labo1 deel1

Commentaar

Deze methode heeft een structuur die makkelijk te volgen is maar splits alles op.

Alles verloopt asynchroon, er is dus geen geheugenwerking.

Deel 2: Comparator met 1 uitgangsvector

Opgave

Ontwerp een 4-bit comparator met 2 woorden A en B maar de vier uitgangen plaats je in één vector van 4 bit : uit_{3-0}

Deze wordt opgemaakt in VHDL en er wordt een behavioral en PIT simulatie uitgevoerd om de correcte werking te testen. Daarna wordt de maximale frequentie bepaald en wordt er gekeken hoeveel resources er gebruikt worden.

VHDL beschrijving

```
entity Comparator is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           Uit : out STD_LOGIC_VECTOR (3 downto 0));
end Comparator;

architecture Behavioral of Comparator is
begin
    Uit(0) <= '1' when A>B else '0';
    Uit(1) <= '1' when A=B else '0';
    Uit(2) <= '1' when A<B else '0';
    Uit(3) <= '1' when (A(0) XOR A(1) XOR A(2) XOR A(3))
                    = (B(0) XOR B(1) XOR B(2) XOR B(3)) else '0';
end Behavioral;
```

Redenering

Idem als deel 1.

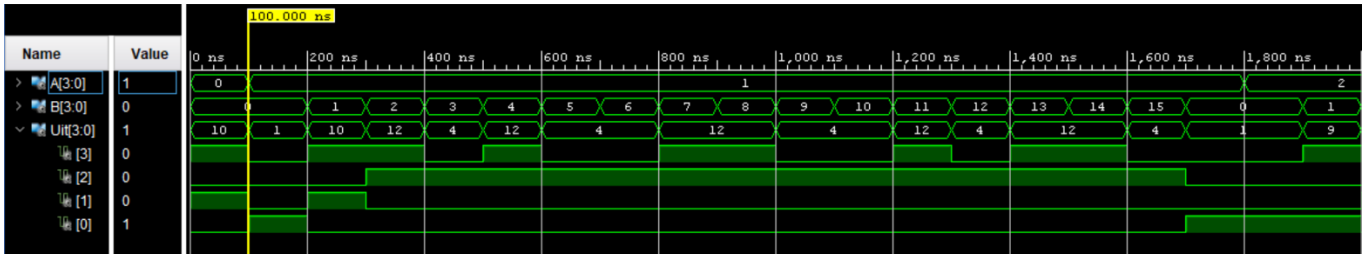
In plaats van de uitgangen los van elkaar te definiëren, zijn deze samen in 1 vector geplaatst.

Testbench

```
entity Comparator_tb is
end Comparator_tb;
architecture Behavioral of Comparator_tb is
    COMPONENT Comparator
    PORT(
            A : IN std_logic_vector(3 downto 0);
            B : IN std_logic_vector(3 downto 0);
            Uit : OUT std_logic_vector(3 downto 0)
        );
    END COMPONENT;

signal A : std_logic_vector(3 downto 0) := "0000";
signal B : std_logic_vector(3 downto 0) := "0000";
signal Uit : std_logic_vector(3 downto 0) := "0000";

begin
    uut: Comparator PORT MAP (
        A => A,
        B => B,
        Uit => Uit
        );

stim_proc: process
    begin
        for i in 0 to 15 loop
```

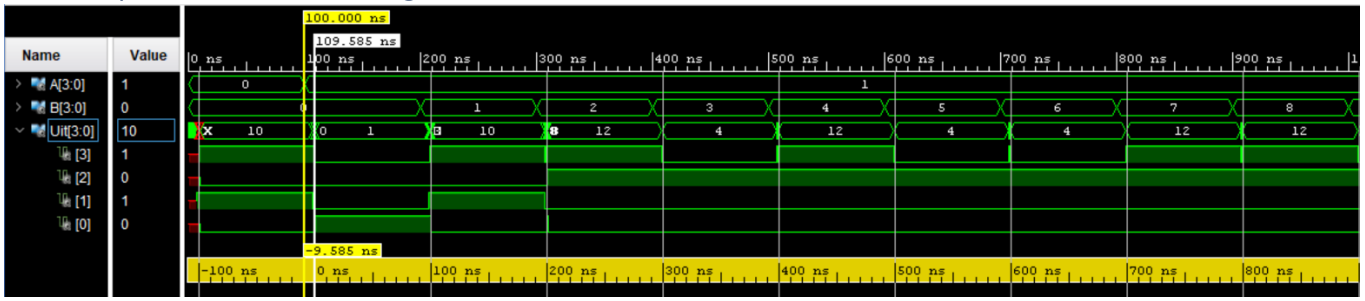
```
                WAIT FOR 100ns;
                A <= STD_LOGIC_VECTOR (A + '1');
                for j in 0 to 15 loop
                        WAIT FOR 100ns;
                        B <= STD_LOGIC_VECTOR (B + '1');
                END LOOP;
        END LOOP;
        wait;
    end process;
end Behavioral;
```

Behavioral simulatie



Figuur 5: Behavioral labo1 deel2

Post implementation timing simulatie



Figuur 6: PIT labo1 deel2

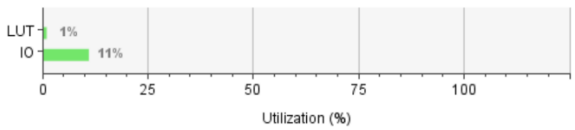
Maximale werkfrequentie

We zien op figuur 6 dat er een vertraging is van 9.585ns -> 1/(9.585ns) = 104.33MHz.

Resources

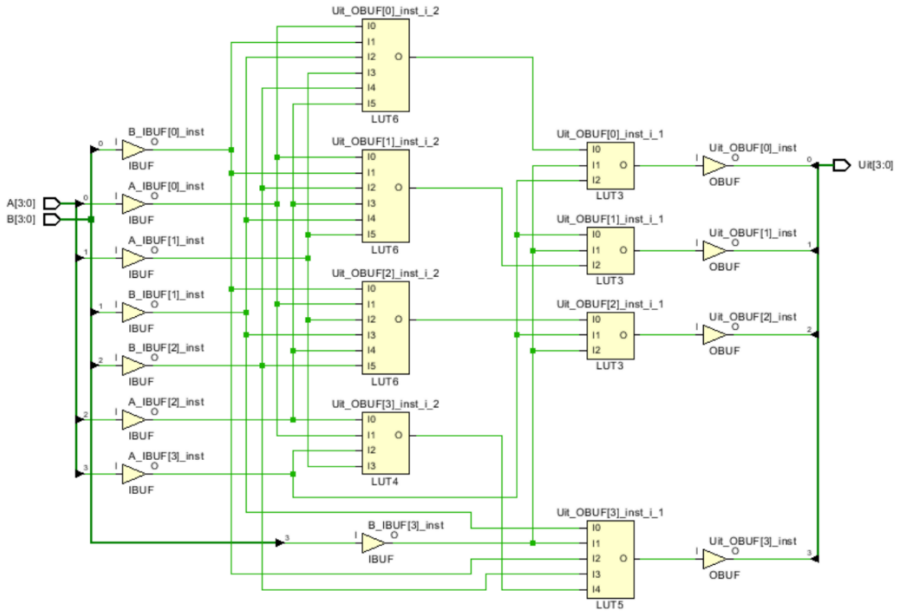
Het aantal resources blijft gelijk.

Resource	Utilization	Available	Utilization %
LUT	7	20800	0.03
IO	12	106	11.32



Figuur 7: Resources labo1 deel2

Schematic



Figuur 8: Schematic labo1 deel2

Commentaar

Het aantal resources blijft gelijk maar de maximale frequentie ligt veel lager (van 164 naar 104 MHz).

De schematic toont dat het enige verschil is dat de uitgangen aan elkaar gekoppeld zijn.

Ook verloopt hier alles asynchroon, er is dus geen geheugenwerking.

Deel 3: Comparator met geklokte uitgang

Opgave

Ontwerp een 4-bit comparator met 2 woorden A en B maar de uitgang wordt geklokt

Deze wordt opgemaakt in VHDL en er wordt een behavioral- en PIT-simulatie uitgevoerd om de correcte werking te testen. Daarna wordt de maximale frequentie bepaald en wordt er gekeken hoeveel resources er gebruikt worden.

VHDL beschrijving

```
entity Comparator is
    Port ( clk : in STD_LOGIC;
           A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           AGTB : out STD_LOGIC;
           AEQB : out STD_LOGIC;
           ALTB : out STD_LOGIC;
           EPEQ : out STD_LOGIC);
end Comparator;

architecture Behavioral of Comparator is

begin
process (clk)
begin
   if rising_edge(clk) then
        if ( A>B ) then
            AGTB <= '1';
            ALTB <= '0';
            AEQB <= '0';
        elsif( A<B ) then
            AGTB <= '0';
            ALTB <= '1';
            AEQB <= '0';
        else
            AEQB <= '1';
            ALTB <= '0';
            AGTB <= '0';
        end if;
        if (A(0) XOR A(1) XOR A(2) XOR A(3))
             = (B(0) XOR B(1) XOR B(2) XOR B(3)) then
            EPEQ <= '1';
        else
            EPEQ <= '0';
        end if;
   end if;
end process;
end Behavioral;
```

Redenering

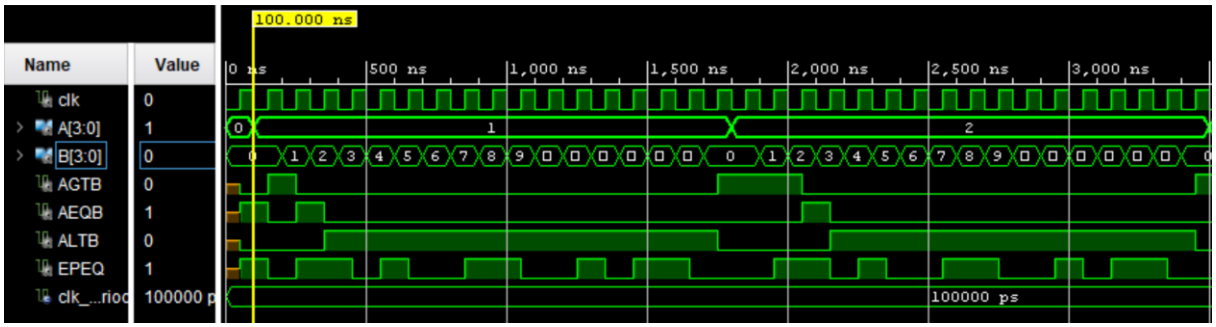
Idem als deel 1.

Enkel bij een stijgende flank van de klok worden er waarden aan de uitgangen toegekend.

Testbench

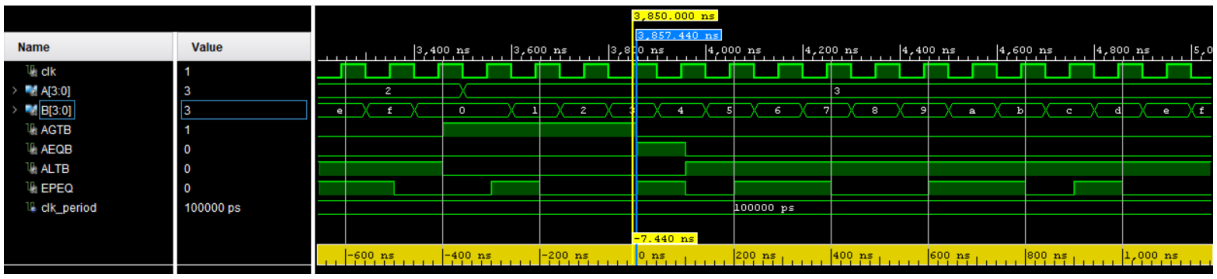
```
architecture Behavioral of Comparator_tb is
    COMPONENT Comparator
    PORT(
            clk : IN std_logic;
            A : IN std_logic_vector(3 downto 0);
            B : IN std_logic_vector(3 downto 0);
            AGTB : OUT std_logic;
            AEQB : OUT std_logic;
            ALTB : OUT std_logic;
            EPEQ : OUT std_logic
        );
    END COMPONENT;

signal clk : std_logic := '0';
signal A : std_logic_vector(3 downto 0) := "0000";
signal B : std_logic_vector(3 downto 0) := "0000";
signal AGTB :  std_logic:='0';
signal AEQB :  std_logic:='0';
signal ALTB :  std_logic:='0';
signal EPEQ :  std_logic:='0';

constant clk_period : time := 100 ns;

begin
    uut: Comparator PORT MAP (
        clk => clk,
        A => A,
        B => B,
        AGTB => AGTB,
        AEQB => AEQB,
        ALTB => ALTB,
        EPEQ => EPEQ
        );

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

    stim_proc: process
    begin
        for i in 0 to 15 loop
                WAIT FOR 100ns;
                A <= STD_LOGIC_VECTOR (A + '1');
                for j in 0 to 15 loop
                        WAIT FOR 100ns;
                        B <= STD_LOGIC_VECTOR (B + '1');
                END LOOP;
        END LOOP;
        wait;
    end process;
end Behavioral;
```

Behavioral simulatie



Figuur 9: Behavioral labo1 deel3

Post implementation timing simulatie



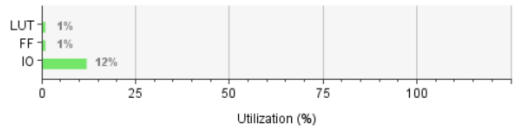
Figuur 10: PIT labo1 deel3

Maximale werkfrequentie

We zien op figuur 10 dat er een vertraging is van 7.440ns -> 1/(7.440ns) = 134.409MHz.

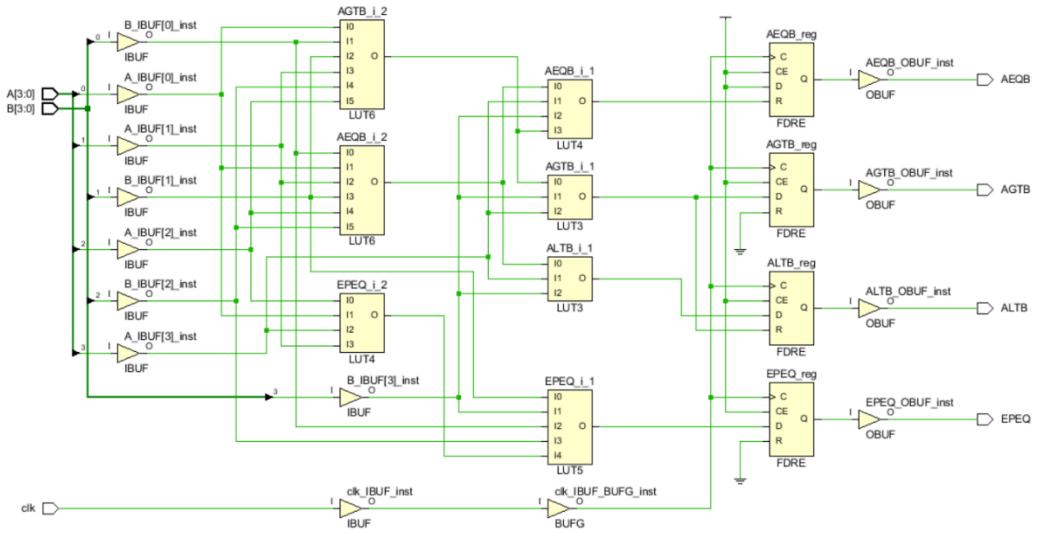
Resources

Resource	Utilization	Available	Utilization %
LUT	7	20800	0.03
FF	4	41600	0.01
IO	13	106	12.26



Figuur 11: Resources labo1 deel3

Schematic



Figuur 12: Schematic labo1 deel3

Commentaar

De maximale frequentie ligt tussen de andere 2 methoden.

Bij het vergelijken van de schematic is er geheugenwerking nodig bij de geklokte uitgang, dit is zichtbaar door de flipflops. Deze methode benut dus de meeste resources.

Er is hier wel geen reset voorzien, de counter kan dus niet op zijn beginwaarde gezet worden.

Lab 2: Schuifregister

Deel 1: Simuleren

Opgave

Ontwerp een VHDL-model voor de component 74LS166

Deze wordt opgemaakt in VHDL en er wordt een behavioral en PIT simulatie uitgevoerd om de correcte werking te testen. Daarna wordt de maximale frequentie bepaald en wordt er gekeken hoeveel resources er gebruikt worden.

VHDL beschrijving

```
entity shifter is
    Port ( clear  : in  STD_LOGIC;
           serial_input  : in  STD_LOGIC; -- 1 om te shiften
           sl : in  STD_LOGIC; -- shift load
           data_load : IN std_logic_vector(7 downto 0); --a tot h
           clk : in  STD_LOGIC;
           clk_inh   : in  STD_LOGIC;
           Q : out std_logic_vector(7 downto 0);
           Qh : out STD_LOGIC);
end shifter;

architecture Behavioral of shifter is
begin
      process(clk)
      VARIABLE datac : std_logic_vector(7 downto 0);
      begin
            if clear = '0' then
                  datac := "00000000";
            elsif rising_edge(clk) then
                  if clk_inh = '0'then -- inverted
                        if sl = '0' then -- shift or load
                              datac := data_load;
                        else
                              datac := datac(6 downto 0) & serial_input;
                        end if;
            end if;
      end if;
      Q <= datac;
      Qh <= datac(7);
      end process;
end Behavioral;
```

Redenering

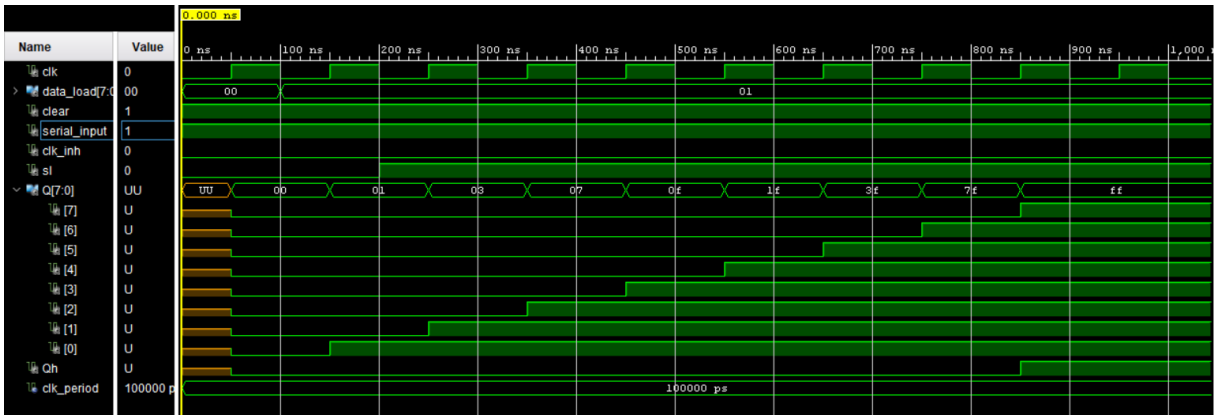
Wanneer clear laag is dan zetten we alle ingangen op 0.

Wanneer clear hoog is en er een stijgende flank is van de klok en clk_inh laag is dan wordt er gekeken of er geschoven of ingeladen moet worden. Wanneer er ingeladen moet worden wordt de data die op data_load staat naar datac geschreven. Indien er geschoven moet worden wordt er met een reeks Dff´s een 1 toegevoegd en worden de uitgangen opgeschoven.

Testbench

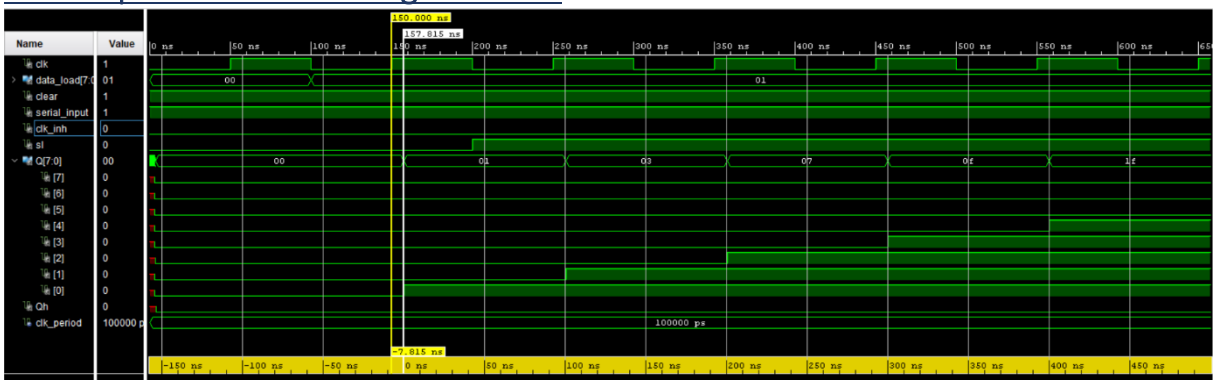
```
architecture Behavioral of shifter_tb is
COMPONENT shifter
    PORT(
           clear  : in  STD_LOGIC;
           serial_input  : in  STD_LOGIC; -- 1 om te shiften    (ser)
           sl : in  STD_LOGIC; -- shift load
           data_load : IN std_logic_vector(7 downto 0); --a tot h
           clk : in  STD_LOGIC;
           clk_inh   : in  STD_LOGIC;
           Q : out std_logic_vector(7 downto 0);
           Qh : out STD_LOGIC
        );
    END COMPONENT;

-- in
signal clk : std_logic := '0';
signal data_load : std_logic_vector(7 downto 0) := "00000000";
signal clear :  std_logic:='1';
signal serial_input :  std_logic:='1';
signal clk_inh :  std_logic:='0';
signal sl :  std_logic:='0';
-- out
signal Q : std_logic_vector(7 downto 0) := "00000000";
signal Qh :  std_logic:='0';

constant clk_period : time := 100 ns;

begin
    uut: shifter PORT MAP (
        clk => clk,
        clear => clear,
        data_load => data_load,
        serial_input => serial_input,
        clk_inh => clk_inh,
        sl => sl,
        Qh => Qh,
        Q => Q
        );

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

    stim_proc: process
    begin
       wait for 100ns;
        data_load <= "00000001";
        wait for 100ns;
        sl <= '1';

        wait;
    end process;
end Behavioral;
```

Behavioral simulatie



Figuur 13: Behavioral labo2 deel1

Post implementation timing simulatie



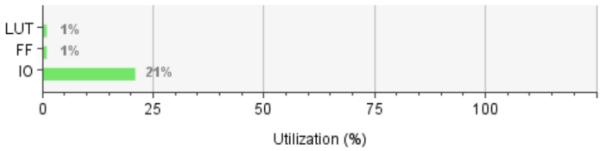
Figuur 14: PIT labo2 deel1

Maximale werkfrequentie

We zien op figuur 14 dat er een vertraging is van 7.815ns -> 1/(7.815ns) = 127.960MHz.

Resources

Resource	Utilization	Available	Utilization %
LUT	6	20800	0.03
FF	8	41600	0.02
IO	22	106	20.75



Figuur 15: Resources labo2 deel1

Commentaar

De reset is asynchroon en heeft dus voorrang op de geklokte werking.

Deel 2: Testen op FPGA-bord

Opgave

Ontwerp een synthetiseerbaar VHDL-model voor test op FPGA-board

Om de werking van het schuifregister te kunnen testen op het oefenbord moeten een aantal wijzigingen gemaakt worden in de VHDL-beschrijving:

a. Om de werking van het schuifregister vlot te kunnen testen, gebruiken we de drukknop om klokpulsen aan het schuifregister te geven.
b. Voor clock_inhibit, serial_in, clear, pushbutton en shift_load gebruik je de schuifschakelaars
c. Serial_out verbind je met één van de LEDs
d. Datac zijn de 8 uitgangen van de DFF's die we naar buiten brengen voor het testen. Verbind ze met de 8 LED's op het oefenbord (LD0 – LD7)
e. data_load verbind je met 8 schakelaars op het oefenbord

VHDL beschrijving

```
entity shifter is
    Port ( clear  : in  STD_LOGIC;
           serial_input  : in  STD_LOGIC; -- 1 om te shiften    (ser)
           sl : in  STD_LOGIC; -- shift load
           pushbutton : IN std_logic;
           data_load : IN std_logic_vector(7 downto 0); --a tot h
           clk : in  STD_LOGIC;
           clk_inh   : in  STD_LOGIC;
           Q : out std_logic_vector(7 downto 0);
           Qh : out STD_LOGIC);
end shifter;

architecture Behavioral of shifter is
      signal shift_clock,denderclk : std_logic;
begin
      drukclock: PROCESS(clk)
             variable i : integer range 0 to 1000000;
             BEGIN
                    IF rising_edge(clk) then
                    i:=i+1;
                           IF i=1000000 THEN
                                  i:=0;
                                  denderclk <= NOT denderclk;
                           END IF;
                    END IF;
      END PROCESS drukclock;

      leesdruk: PROCESS(denderclk)
      BEGIN
            IF rising_edge(denderclk) then
                  shift_clock <= pushbutton;
            END IF;
      END PROCESS leesdruk;

      schuif : PROCESS( clear, shift_clock)
      VARIABLE datac : std_logic_vector(7 downto 0);
      BEGIN
            if clear = '0' then
                  datac := "00000000";
```

```
            elsif rising_edge(shift_clock) then
                  if clk_inh = '0'then -- inverted
                        if sl = '0' then -- shift or load
                              datac := data_load;
                        else
                              datac := datac(6 downto 0) & serial_input;
                        end if;
                  end if;
            end if;
            Q <= datac;
            Qh <= datac(7);
      END PROCESS schuif;
end Behavioral;
```

Redenering

Idem als vorige deel.

Commentaar

Bij het testen op het bord is het duidelijk dat de vhdl-beschrijving doet wat er verwacht wordt.

Er werdt hier een extra proces aangemaakt: “leesdruk”. Deze zordt ervoor dat de drukknop dendervrij is. Wanneer een schakelaar van positie verandert dan zit hier door de bewegingswetten een bepaalde tijd op waardoor het signaal niet stabiel is (dender), dit proces zorgt ervoor dat er maar 1 signaal gegenereerdt wordt in plaats van duizenden.

Lab 3: Up/Downcounter

Deel 1: Simuleren

Opgave

Ontwerp een VHDL-beschrijving voor een populaire up/down counter : 74LS190

Deze wordt opgemaakt in VHDL en er wordt een behavioral- en PIT-simulatie uitgevoerd om de correcte werking te testen. Daarna wordt de maximale frequentie bepaald en wordt er gekeken hoeveel resources er gebruikt worden.

VHDL beschrijving

```
entity counter is
    Port ( enable : IN std_logic;
           load :  IN std_logic;
           clk : in  STD_LOGIC;
           downUp : IN std_logic;
           Q : OUT std_logic_vector(3 downto 0);
           segmentr : OUT std_logic_vector(7 downto 0);
           rippleClock : out std_logic;
           maxMin : out std_logic;
           data : IN std_logic_vector(3 downto 0));
end counter;

architecture Behavioral of counter is
begin
      process(clk)
      VARIABLE datac : std_logic_vector(3 downto 0);
      begin
            if load = '0' then
                  datac := data;

            elsif rising_edge(clk) then
                  if enable = '0'then
                        if downUp = '0' then --count down
                              if datac = "0000" then
                                    datac := "1001";
                              else
                                    datac := datac - 1;
                              end if;
                        elsif downUp = '1' then -- count up
                              if datac = "1001" then
                                    datac := "0000";
                              else
                                    datac := datac + 1;
                              end if;
                        end if;
                  end if;
            end if;

            if (datac = "1001" AND downUp = '1')
                  OR (datac = "0000" AND downUp = '0') then
                  maxMin <= '1';
                  if clk = '0' then
                        rippleClock <= '0';
                  else
                        rippleClock <= '1';
                  end if;
```

```
            else
                  maxMin <= '0';
                  rippleClock <= '1';
            end if;
            Q <= datac;
      end process;
end Behavioral;
```

Redenering

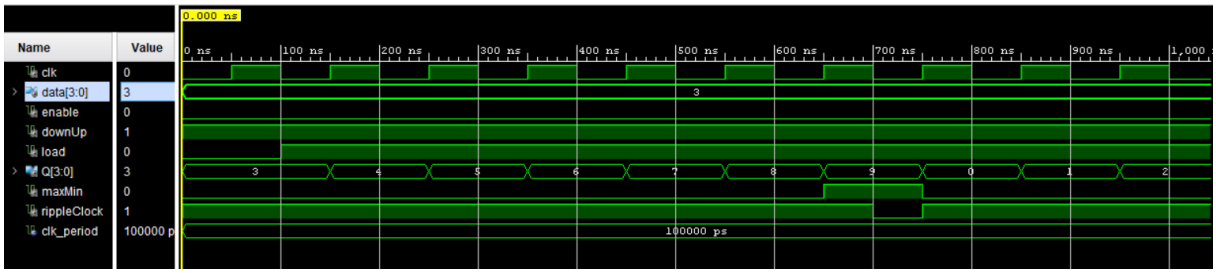
Indien load laag is wordt een getal ingeladen, anders wordt er op- of afgeteld. Tellen gebeurt enkel op de stijgende flank van de klok en als enable laag is. Wanneer de minimale of maximale waarde bereikt wordt, wordt er gereset. Anders is het een simpele + of – operatie.

Testbench

```
architecture Behavioral of counter_tb is
COMPONENT counter
    PORT(
          clk : in  STD_LOGIC;
          load : IN std_logic;
          enable : in  STD_LOGIC;
          Q : out std_logic_vector(3 downto 0);
          data : IN std_logic_vector(3 downto 0);
          downUp : IN std_logic;
          segmentr : OUT std_logic_vector(7 downto 0);
          rippleClock : out std_logic;
          maxMin : out std_logic
        );
    END COMPONENT;
-- in
signal clk : std_logic := '0';
signal data : std_logic_vector(3 downto 0) := "0100";
signal enable :  std_logic:='0';
signal downUp :  std_logic:='1';
signal load :  std_logic:='0';
-- out
signal Q : std_logic_vector(3 downto 0) := "0000";
signal maxMin :  std_logic:='0';
signal rippleClock :  std_logic:= '1';

constant clk_period : time := 100 ns;

begin
    uut: counter PORT MAP (
        clk => clk,
        enable => enable,
        data => data,
        downUp => downUp,
        load => load,
        maxMin => maxMin,
        rippleClock => rippleClock,
        Q => Q
        );
    clk_process :process
    begin
      clk <= '0';
      wait for clk_period/2;
      clk <= '1';
```

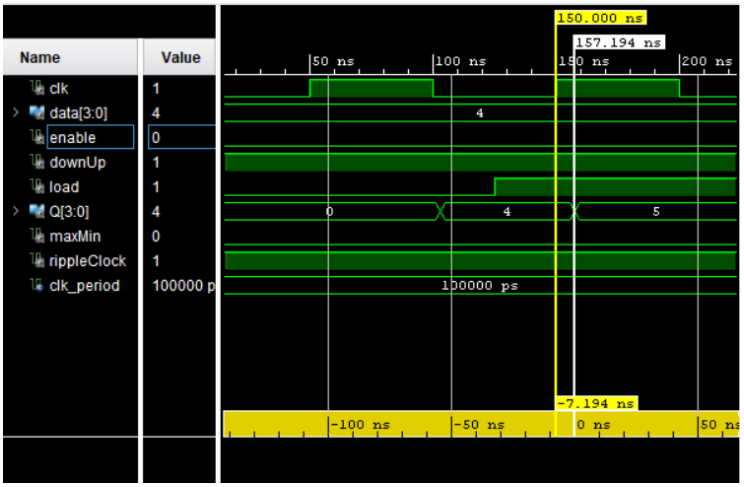
```
      wait for clk_period/2;
   end process;

    stim_proc: process
    begin
        wait for 125ns;
        load <= '1';
    end process;
end Behavioral;
```

Behavioral simulatie



Figuur 16: Behavioral labo3 deel1

Post implementation timing simulatie



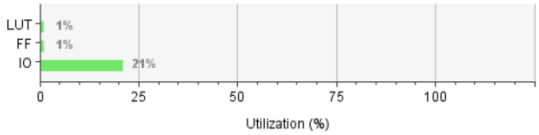
Figuur 17: PIT labo3 deel1

Maximale werkfrequentie

We zien op figuur 17 dat er een vertraging is van 7.194ns -> 1/(7.194ns) = 139.005MHz.

Resources

Resource	Utilization	Available	Utilization %
LUT	20	20800	0.10
FF	12	41600	0.03
IO	22	106	20.75



Figuur 18: Resources

Commentaar

Uit de testbench is af te leiden dat de werking van de vhdl-beschrijving correct is.

Deel 2: Testen op FPGA-bord

Opgave

Aangepaste versie om de counter te laten werken op het FPGA-board.

VHDL beschrijving

```
entity counter is
    Port ( enable : IN std_logic;
           load :  IN std_logic;
           clk : in  STD_LOGIC;
           downUp : IN std_logic;
           Q : OUT std_logic_vector(3 downto 0);
           segmentr : OUT std_logic_vector(7 downto 0);
           rippleClock : out std_logic;
           maxMin : out std_logic;
           data : IN std_logic_vector(3 downto 0));
end counter;

architecture Behavioral of counter is
      signal workClk : std_logic := '0';
begin
      makeClk: process(clk)
            variable a : integer range 0 to 100000000 := 0;
      begin
             if rising_edge(clk) then
                   a:=a+1;
                   if a = 100000000/2 then
                         workClk <= NOT workClk; -- 1 sec
                         a:=0;
                   end if;
             end if;
      end process makeClk;

      count: process(workClk,load)
      VARIABLE datac : std_logic_vector(3 downto 0);
      begin
            if load = '0' then
                  datac := data;

            elsif rising_edge(workClk) then
                  if enable = '0'then -- inverted
                        if downUp = '0' then --count down
                              if datac = "0000" then
                                    datac := "1001";
                              else
                                    datac := datac - 1;
                              end if;
                        elsif downUp = '1' then -- count up
                              if datac = "1001" then
                                    datac := "0000";
                              else
                                    datac := datac + 1;
                              end if;
                        end if;
                  end if;
            end if;
```

```
            if (datac = "1001" AND downUp = '1')
                  OR (datac = "0000" AND downUp = '0') then
                  maxMin <= '1';
                  if clk = '0' then
                        rippleClock <= '0';
                  else
                        rippleClock <= '1';
                  end if;
            else
                  maxMin <= '0';
                  rippleClock <= '1';
            end if;
            Q <= datac;

            case datac is

                  when "0000" => segmentr <= "00000011"; -- abcdefgp
                   when "0001" => segmentr <= "10011111";
                   when "0010" => segmentr <= "00100101"; -- |-a-|
                   when "0011" => segmentr <= "00001101"; -- f b
                   when "0100" => segmentr <= "10011001"; -- |-g-|
                   when "0101" => segmentr <= "01001001"; -- e c
                   when "0110" => segmentr <= "11000001"; -- |-d-|
                   when "0111" => segmentr <= "00011111";
                   when "1000" => segmentr <= "00000001";
                   when "1001" => segmentr <= "00011001";
                   when others => segmentr <= "11111111";

             end case;
      end process count;
end Behavioral;
```

Redenering

Idem als deel 1.

Commentaar

Idem als deel 1.

Lab 4: LCD aansturen

Deel 1: RGB-blok ontwerpen en simuleren

Opgave

Ontwerp het blok RGB om een volledig scherm met één kleur te vullen. Simuleer eerst dit blok en pas als de werking volledig correct is, implementeer je dit in de LCDcontroller.

Volgende gegevens zijn gegeven (Figuur 19):

Horizontale timing (line):

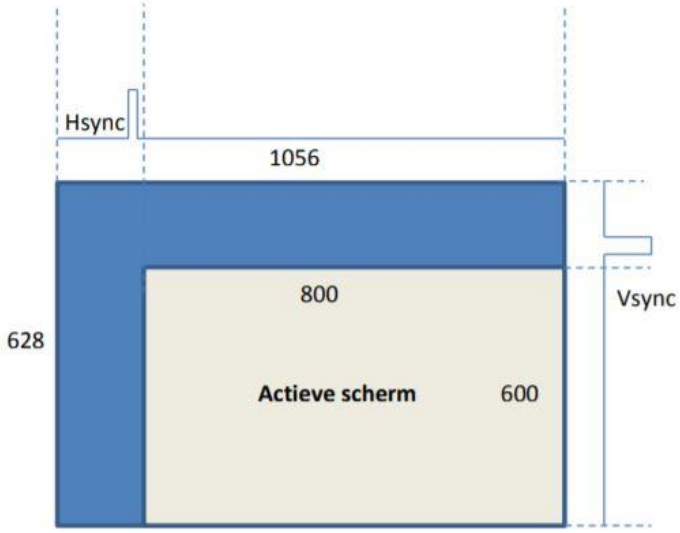
Scanline part	Pixels	Time (µs)
Visible area	800	20
Front porch	40	1
Sync pulse	128	3.2
Back porch	88	2.2
Whole line	1056	26.4

Polarity of horizontal pulse is positive

Verical timing (frame):

Frame part	Lines	Time (ms)
Visible area	600	15.84
Front porch	1	0.0264
Sync pulse	4	0.1056
Back porch	23	0.6072
Whole line	628	16.579

Polarity of vertical pulse is positive



Figuur 19: Schema LCD

VHDL-beschrijving

LCD:

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

ENTITY LCD IS
      PORT(clk100mhz          : IN  std_logic;
           vgaRed       : OUT std_logic_vector(3 DOWNTO 0);
           vgaGreen     : OUT std_logic_vector(3 DOWNTO 0);
           vgaBlue      : OUT std_logic_vector(3 DOWNTO 0);
           reset        : in  std_logic;
           locked                : out std_logic;
           Hsync, Vsync : OUT std_logic);
END LCD;

architecture Behavioral of LCD is

component CLKgen is
    Port ( CLK100mhz : in STD_LOGIC;
           reset : in STD_LOGIC;
           locked : out STD_LOGIC;
           CLK40mhz : out STD_LOGIC);
end component;
```

```
component RGB is
      PORT(pixelclk     : IN  std_logic;
           vgaRed       : OUT std_logic_vector(3 DOWNTO 0);
           vgaGreen     : OUT std_logic_vector(3 DOWNTO 0);
           vgaBlue      : OUT std_logic_vector(3 DOWNTO 0);
           reset        : in  std_logic;
           Hsync, Vsync : OUT std_logic);
end component;

signal pixelclk: std_logic;

begin

u1: CLKgen port map(CLK100mhz,reset,locked,pixelclk);

U2: RGB port map(pixelclk,vgaRed,vgaGreen,vgaBlue,reset,Hsync, Vsync);

end Behavioral;
```

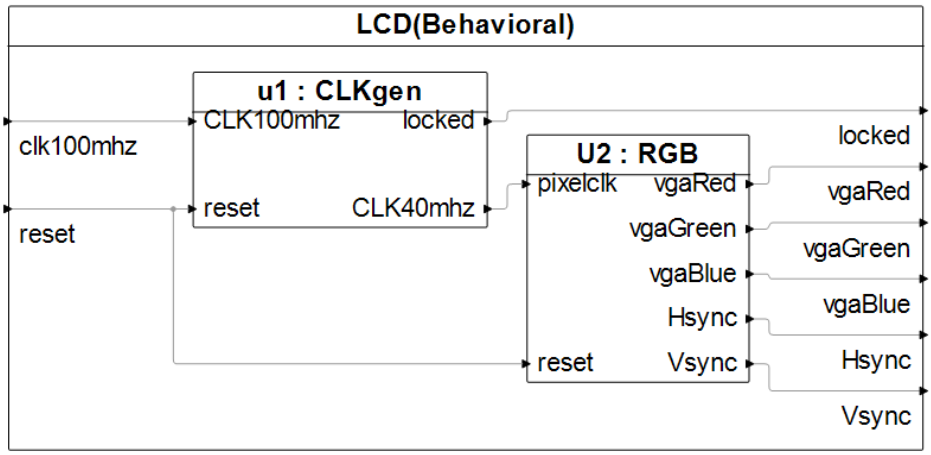
RGB:

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

entity RGB is
    Port ( reset : in  STD_LOGIC;       -- system Reset
           pixelclk     : IN  std_logic;
           vgaRed       : OUT std_logic_vector(3 DOWNTO 0);
           vgaGreen     : OUT std_logic_vector(3 DOWNTO 0);
           vgaBlue      : OUT std_logic_vector(3 DOWNTO 0);
           Hsync, Vsync : OUT std_logic);
end RGB;

architecture Behavioral of RGB is
constant innerCol : integer := 800;
constant innerRow : integer := 600;
constant horizontalBlanking : integer := 256;
constant verticalBlanking : integer := 28;
constant totalRow : integer := innerRow + verticalBlanking;
constant totalCol : integer := innerCol + horizontalBlanking;

begin
process(reset, pixelclk)
      variable col : integer range 0 to totalCol := 0;
      variable row : integer range 0 to totalRow := 0;
      begin
      if reset = '1' then
                  col := 0;
                  row := 0;
      elsif rising_edge(pixelclk) then
            col := col + 1;
            if col = totalCol then
                  col := 0;
                  row := row + 1;
                  if row = totalRow then
                        row := 0;
                  end if;
```

```
            end if;
        end if;

        if col > 39 and col < 128 + 40 then
            Hsync <= '1';
        else
            Hsync <= '0';
        end if;

        if row > 0 and row < 4 + 1 then
            Vsync <= '1';
        else
            Vsync <= '0';
        end if;

        if col > horizontalBlanking and row > verticalBlanking then
            vgaRed <= (others => '1');
            vgaGreen <= (others => '0');
            vgaBlue <= (others => '0');
        else
            vgaRed <= (others => '0');
            vgaGreen <= (others => '0');
            vgaBlue <= (others => '0');
        end if;

end process;
end Behavioral;
```

Redenering + blokschema

In het synchrone deel gaan we over het hele scherm heen. Als we het hele scherm hebben overlopen beginnen we weer opnieuw. Daarnaast moeten de Hsync en Vsync hoog of laag worden bij de juiste kolommen en rijen. Uiteindelijk moet het scherm rood worden en dit gebeurt enkel in het zichtbare gebied, buiten het zichtbare gebied is het zwart.



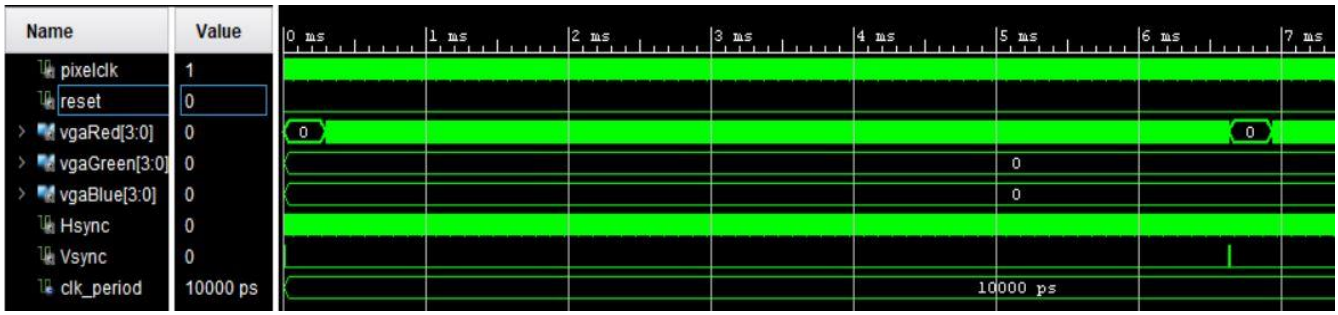
Figuur 20: Blokschema

Testbench

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

entity LCD1_tb is
```

```
end LCD1_tb;

architecture Behavioral of LCD1_tb is
COMPONENT LCD1
    PORT(
            reset : in  STD_LOGIC;
           pixelclk     : IN  std_logic;
           vgaRed       : OUT std_logic_vector(3 DOWNTO 0);
           vgaGreen     : OUT std_logic_vector(3 DOWNTO 0);
           vgaBlue      : OUT std_logic_vector(3 DOWNTO 0);
           Hsync, Vsync : OUT std_logic
        );
END COMPONENT;


-- in
signal pixelclk : std_logic := '0';
signal reset :  std_logic:='0';

-- out
signal vgaRed : std_logic_vector(3 downto 0) := "0000";
signal vgaGreen : std_logic_vector(3 downto 0) := "0000";
signal vgaBlue : std_logic_vector(3 downto 0) := "0000";
signal Hsync :  std_logic:='0';
signal Vsync :  std_logic:='0';


constant clk_period : time := 10 ns;

begin
      uut: LCD1 PORT MAP (
        reset => reset,
        pixelclk => pixelclk,
             vgaRed => vgaRed,
             vgaGreen => vgaGreen,
             vgaBlue => vgaBlue,
             Hsync => Hsync,
             Vsync => Vsync
        );

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

Behavioral simulation



Figuur 21: Behavioral Simulation

Commentaar

De bewerkingen moeten nauwkeurig opgevolgd worden zodat de juiste pixel aangesproken wordt.

De waarden r g en b toekennen gebeurdt asynchroon.

Hsync wordt gebruikt om de kolommen bij te houden en Vsync is om terug te keren naar het begin om een nieuwe frame te maken.

De blanking is er omdat de resolutie van het scherm lager ligt dan de beschikbare resolutie. Dit is handig omdat zo alle kleinere resoluties makkelijk kunnen gebruikt worden.

Deel 2: Kleur instellen met schakelaars

Opgave

Idem als opgave 1 maar het kleur stellen we nu in met 4 schakelaars. Hierdoor zijn er 16 kleurmogelijkheden.

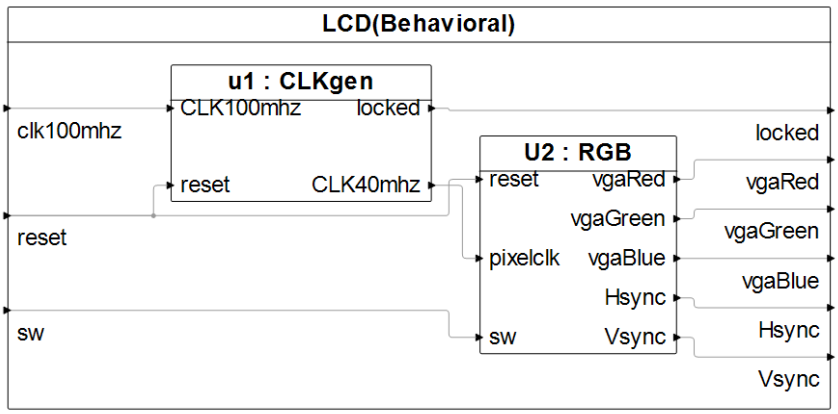
VHDL-beschrijving

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

entity RGB is
    Port ( reset : in  STD_LOGIC;       -- system Reset
           pixelclk     : IN  std_logic;
           sw       : IN std_logic_vector(3 DOWNTO 0);
           vgaRed       : OUT std_logic_vector(3 DOWNTO 0);
           vgaGreen     : OUT std_logic_vector(3 DOWNTO 0);
           vgaBlue      : OUT std_logic_vector(3 DOWNTO 0);
           Hsync, Vsync : OUT std_logic);
end RGB;

architecture Behavioral of RGB is
constant innerCol : integer := 800;
constant innerRow : integer := 600;
constant horizontalBlanking : integer := 256;
constant verticalBlanking : integer := 28;
constant totalRow : integer := innerRow + verticalBlanking;
constant totalCol : integer := innerCol + horizontalBlanking;

begin
process(reset, pixelclk)
      variable col : integer range 0 to totalCol := 0;
      variable row : integer range 0 to totalRow := 0;
      begin
      if reset = '1' then
                  col := 0;
                  row := 0;
      elsif rising_edge(pixelclk) then
            col := col + 1;
            if col = totalCol then
                  col := 0;
                  row := row + 1;
                  if row = totalRow then
                        row := 0;
                  end if;
            end if;
      end if;

      if col > 39 and col < 128 + 40 then
            Hsync <= '1';
      else
            Hsync <= '0';
      end if;

      if row > 0 and row < 4 + 1 then
            Vsync <= '1';
      else
            Vsync <= '0';
      end if;
```

```
        if col > horizontalBlanking and row > verticalBlanking then
                vgaRed <= sw;
                vgaGreen <= sw;
                vgaBlue <= sw;
        else
                vgaRed <= (others => '0');
                vgaGreen <= (others => '0');
                vgaBlue <= (others => '0');
        end if;

end process;
end Behavioral;
```

Redenering + blokschema

Met de vector sw stellen we de waardes in van de kleuren. We kunnen hier met 4 schakelaars 16 mogelijke grijswaardes instellen. Dit loopt van zwart tot wit (0000 tot 1111).



Figuur 22: Blokschema

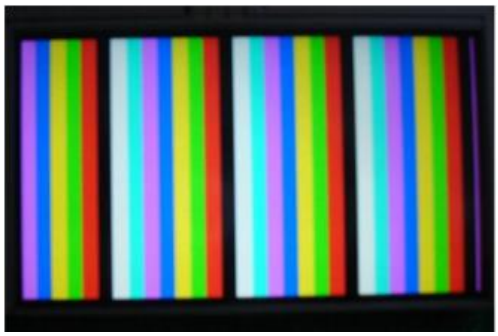
Commentaar

Dit is een zeer kleine aanpassing daar de switches standaard al een toestand zullen hebben.

Deel 3: Kleurbalken

Opgave

Genereer kleurbalken volgens onderstaand voorbeeld. Ontwerp jouw VHDL-model zodanig dat met een minimum aan code-aanpassing de breedte van de kleurbalken kan ingesteld worden.



Figuur 23: Te bekomen op LCD

VHDL-beschrijving

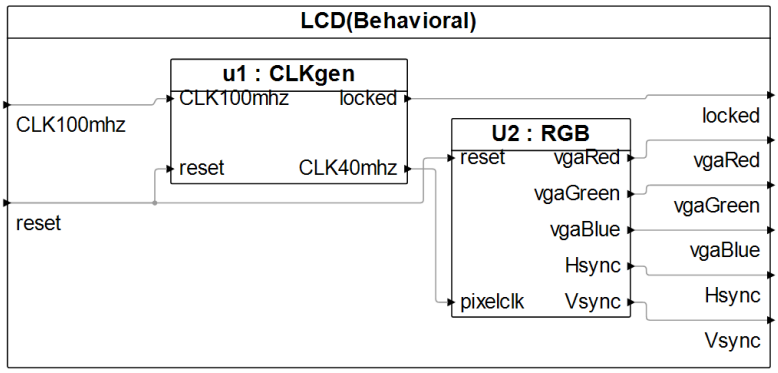
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity RGB is
    Port ( reset : in  STD_LOGIC;       -- system Reset
           pixelclk     : IN  std_logic;
           vgaRed       : OUT std_logic_vector(3 DOWNTO 0);
           vgaGreen     : OUT std_logic_vector(3 DOWNTO 0);
           vgaBlue      : OUT std_logic_vector(3 DOWNTO 0);
           Hsync, Vsync : OUT std_logic);
end RGB;

architecture Behavioral of RGB is
constant innerCol : integer := 800;
constant innerRow : integer := 600;
constant horizontalBlanking : integer := 256;
constant verticalBlanking : integer := 28;
constant totalRow : integer := innerRow + verticalBlanking;
constant totalCol : integer := innerCol + horizontalBlanking;
constant stripes : integer := 20;
constant width : integer := innerCol / stripes;
constant colors : integer := 8;

begin
process(reset, pixelclk)
      variable col : integer range 0 to totalCol := 0;
      variable row : integer range 0 to totalRow := 0;
      variable currentStripe : integer range 0 to colors := 0;
      variable currentWidth : integer range 0 to width + 1 := 0;

      begin
      if reset = '1' then
                  col := 0;
                  row := 0;
                  currentStripe := 0;
                  currentWidth := 0;
```

```
elsif rising_edge(pixelclk) then
	col := col + 1;
	if col = totalCol then
		col := 0;
		row := row + 1;
		currentWidth := 0;
		currentStripe := 0;
		if row = totalRow then
			row := 0;
		end if;
	end if;

	if col > horizontalBlanking and row > verticalBlanking then
		currentWidth := currentWidth + 1;
		if currentWidth = width + 1 then
			currentWidth := 1;
			currentStripe := currentStripe + 1;
			if currentStripe = colors then
				currentStripe := 0;
			end if;
		end if;
	end if;
end if;

if col > 39 and col < 128 + 40 then
	Hsync <= '1';
else
	Hsync <= '0';
end if;
if row > 0 and row < 4 + 1 then
	Vsync <= '1';
else
	Vsync <= '0';
end if;

if col > horizontalBlanking and row > verticalBlanking then
	case currentStripe is
		when 0 =>   vgaRed <= (others => '1');
				vgaGreen <= (others => '0');
				vgaBlue <= (others => '1');
		when 1 => vgaRed <= (others => '0');
				vgaGreen <= (others => '0');
				vgaBlue <= (others => '1');
		when 2 => vgaRed <= (others => '1');
				vgaGreen <= (others => '1');
				vgaBlue <= (others => '0');
		when 3 => vgaRed <= (others => '0');
				vgaGreen <= (others => '1');
				vgaBlue <= (others => '0');
		when 4 => vgaRed <= (others => '1');
				vgaGreen <= (others => '0');
				vgaBlue <= (others => '0');
		when 5 => vgaRed <= (others => '0');
				vgaGreen <= (others => '0');
				vgaBlue <= (others => '0');
		when 6 => vgaRed <= (others => '1');
				vgaGreen <= (others => '1');
				vgaBlue <= (others => '1');
		when 7 => vgaRed <= (others => '0');
```

```
                                        vgaGreen <= (others => '1');
                                        vgaBlue <= (others => '1');
                        when others => vgaRed <= (others => '0');
                                        vgaGreen <= (others => '0');
                                        vgaBlue <= (others => '0');
                end case;

        else
                vgaRed <= (others => '0');
                vgaGreen <= (others => '0');
                vgaBlue <= (others => '0');
        end if;
end process;
end Behavioral;
```

Redenering + blokschema

Aan de hand van currentStripe wordt de kleur bepaald. Aan de hand van currentWidth weten we wanneer het volgnde kleur moet worden weergegeven. Als currentWidth aantoont dat er een nieuwe kleur moet verschijnen (als deze gelijk is aan width) zal currentWidth herbeginnen bij 1 en zal currentStripe zorgen voor de volgende kleur. De rest is gelijkaardig als vorige delen.



Figuur 24: Blokschema

Commentaar

We hebben hier redelijk veel problemen gehad. we kwamen tot de conclusie dat het synchrone en asynchrone deel niet los stonden van elkaar en dat de connector niet goed bevestigt was. Na dit thuis in C# te coderen en de juiste werking te bevestigen werd deze in vhdl omgezet en werkte de implementatie.

We besluiten dat de waardes correct opgegeven moeten zijn en dat synchroon en asynchrone splitsing nauwkeurig moet gebeuren.

Deel 4: Animatie

Opgave

Ontwerp een unieke animatie. Je mag hier vrij kiezen maar de animatie moet verschillend zijn van de andere groepen en verschillend van de animaties van vorig jaar.

VHDL-beschrijving

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

entity RGB is
    Port ( reset : in  STD_LOGIC;       -- system Reset
           pixelclk     : IN  std_logic;
           vgaRed       : OUT std_logic_vector(3 DOWNTO 0);
           vgaGreen     : OUT std_logic_vector(3 DOWNTO 0);
           vgaBlue      : OUT std_logic_vector(3 DOWNTO 0);
           Hsync, Vsync : OUT std_logic);
end RGB;

architecture Behavioral of RGB is
constant innerCol : integer := 800;
constant innerRow : integer := 600;
constant horizontalBlanking : integer := 256;
constant verticalBlanking : integer := 28;
constant totalRow : integer := innerRow + verticalBlanking;
constant totalCol : integer := innerCol + horizontalBlanking;
constant width : integer := innerCol/20;
constant height : integer := innerRow/8;

signal LR: boolean := false;

begin
process(reset, pixelclk)
      variable col : integer range 0 to totalCol := 0;
      variable row : integer range 0 to totalRow := 0;
      variable xShift : integer range 0 to innerCol := 0;
      variable yShift : integer range 0 to height := 0;

      variable draw : integer range 0 to 1:= 0;

      begin
      if reset = '1' then
                  col := 0;
                  row := 0;
                  xShift := 0;
                  yShift := 0;
                  LR <= false;
                  draw := 0;
      elsif rising_edge(pixelclk) then
            col := col + 1;
            if col = totalCol then
                  col := 0;
                  row := row + 1;
                  yShift := yShift + 1;

            end if;
            if row = totalRow then
                  row := 0;
```

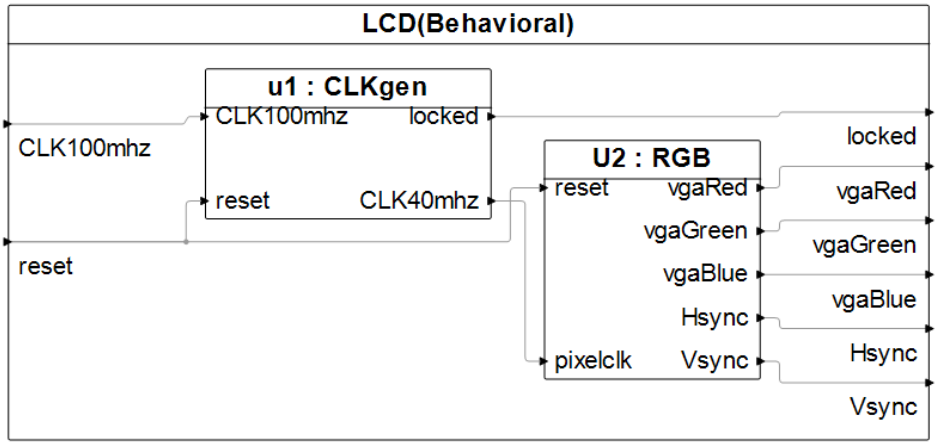
```
                xShift := xShift + 1;
            end if;
            if xShift = innerCol then
                xShift := 0;
            end if;
            if yShift = height then
                yShift := 0;
                LR <= not(LR);
            end if;
        end if;

        if col > 39 and col < 128 + 40 then
            Hsync <= '1';
        else
            Hsync <= '0';
        end if;

        if row > 0 and row < 4 + 1 then
            Vsync <= '1';
        else
            Vsync <= '0';
        end if;


        if col > xShift + horizontalBlanking
            and col < xShift + width + horizontalBlanking
            and LR = false then
            draw := 1;
        elsif col < totalCol - xShift
            and col > totalCol - xShift - width
            and LR = true then
            draw := 1;
        else draw := 0;
        end if;

        if col > horizontalBlanking and row > verticalBlanking then
            case draw is
                when 1 =>  vgaRed <= (others => '1');
                                    vgaGreen <= (others => '1');
                                    vgaBlue <= (others => '1');

                when others =>  vgaRed <= (others => '0');
                                    vgaGreen <= (others => '0');
                                    vgaBlue <= (others => '0');
            end case;
        else
            vgaRed <= (others => '0');
            vgaGreen <= (others => '0');
            vgaBlue <= (others => '0');
        end if;

end process;
end Behavioral;
```

Redenering + blokschema

Onze animatie bestaat uit een reeks blokjes die van links naar rechts bewegen en een reeks blokjes die van links naar rechts bewegen. Er wordt per blokje geswitched tussen de linker en rechterzijde van het scherm. De blokjes gaan ook van onder naar boven.

Hiervoor houden we een paar variabelen bij:

- xShift: hoeveel we het blokje opschuiven
- yShift: hoeveel rijen van het huidige blokje reeds getekend is
- LR: links of rechts van het scherm
- Draw: wordt de huidige pixel gekleurd of niet



Figuur 25: Blokschema

Commentaar

Animatie werkte op het bordje.

Deel 5: Sound-blok

Opgave

Ontwerp een extra VHDL-blok sound.vhd om geluid aan je animatie toe te voegen.

VHDL-beschrijving

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

entity RGB is
	Port(reset        : in  STD_LOGIC;  -- system Reset
	     pixelclk     : IN  std_logic;
	     vgaRed       : OUT std_logic_vector(3 DOWNTO 0);
	     vgaGreen     : OUT std_logic_vector(3 DOWNTO 0);
	     vgaBlue      : OUT std_logic_vector(3 DOWNTO 0);
	     Hsync, Vsync : OUT std_logic;
	     JB           : out std_logic_vector(5 DOWNTO 0));
end RGB;

architecture Behavioral of RGB is
	constant innerCol           : integer := 800;
	constant innerRow           : integer := 600;
	constant horizontalBlanking : integer := 256;
	constant verticalBlanking   : integer := 28;
	constant totalRow           : integer := innerRow + verticalBlanking;
	constant totalCol           : integer := innerCol + horizontalBlanking;
	constant width              : integer := innerCol/20;
	constant height             : integer := innerRow/8;

	signal LR   : boolean := false;
	signal horn : boolean := false;
	signal count : boolean := false;

begin
	sound : process(reset, pixelclk)
		variable counter : integer range 0 to 640000 := 0;
	begin
	if reset = '1' then
		counter := 0;
		horn    <= false;
	elsif rising_edge(pixelclk) then
		counter := counter + 1;
		if count then
			if counter = 160000 then
				horn <= not horn;
			else
				if counter >= 320000 then
					horn    <= not horn;
					counter := 0;
				end if;
			end if;

		else
			if counter = 320000 then
				horn <= not horn;
			else
				if counter >= 640000 then
					horn    <= not horn;
```

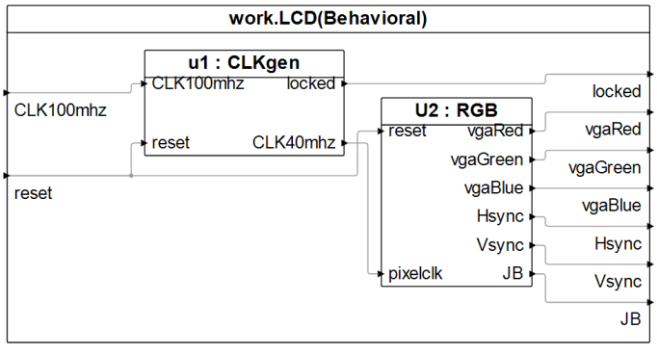
```
                        counter := 0;
                    end if;
                end if;
            end if;
            if horn then
                    JB <= "111111";
                else
                    JB <= "000000";
            end if;
    end if;
    end process sound;

    rgb : process(reset, pixelclk, LR)
            variable col    : integer range 0 to totalCol := 0;
            variable row    : integer range 0 to totalRow := 0;
            variable xShift : integer range 0 to innerCol := 0;
            variable yShift : integer range 0 to height   := 0;
            variable draw   : integer range 0 to 1        := 0;

    begin
            if reset = '1' then
                    count <= false;
                    col    := 0;
                    row    := 0;
                    xShift := 0;
                    yShift := 0;
                    LR     <= false;
                    draw   := 0;
            elsif rising_edge(pixelclk) then
                    col := col + 1;
                    if col = totalCol then
                            col    := 0;
                            row    := row + 1;
                            yShift := yShift + 1;

                    end if;
                    if row = totalRow then
                            row    := 0;
                            xShift := xShift + 1;
                    end if;
                    if xShift = innerCol then
                            xShift := 0;
                    end if;
                    if yShift = height then
                            yShift := 0;
                            LR     <= not (LR);
                    end if;
            end if;

            if col > 39 and col < 128 + 40 then
                    Hsync <= '1';
            else
                    Hsync <= '0';
            end if;

            if row > 0 and row < 4 + 1 then
                    Vsync <= '1';
            else
                    Vsync <= '0';
```

```
            end if;

            if xShift > innercol/2 then
                  count <= true;
            else count <= false;
            end if;

            if col > xShift + horizontalBlanking
                  and col < xShift + width + horizontalBlanking
                  and LR = false then
                  draw := 1;
            elsif col < totalCol - xShift
                  and col > totalCol - xShift - width
                  and LR = true then
                  draw := 1;
            else
                  draw := 0;
            end if;

            if col > horizontalBlanking and row > verticalBlanking then
                  case draw is
                        when 1 => vgaRed   <= (others => '1');
                              vgaGreen <= (others => '1');
                              vgaBlue  <= (others => '1');

                        when others => vgaRed   <= (others => '0');
                              vgaGreen <= (others => '0');
                              vgaBlue  <= (others => '0');
                  end case;
            else
                  vgaRed   <= (others => '0');
                  vgaGreen <= (others => '0');
                  vgaBlue  <= (others => '0');
            end if;
      end process rgb;
end Behavioral;
```

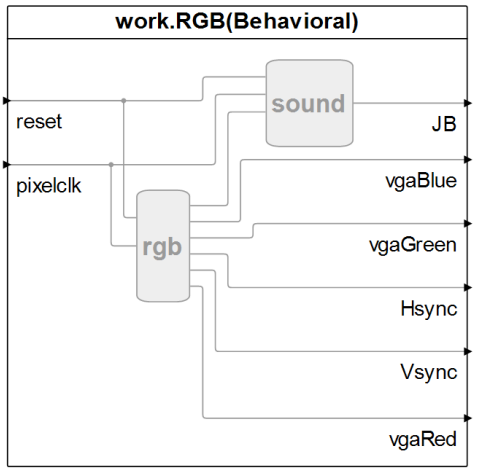
Redenering + blokschema

We switchen tussen pitch van geluid wanneer we overgaan van het linker gedeelte van het scherm naar het rechtergedeelte. Dit doen we door een proces “sound” te maken die aangeroepen wordt wanneer pixelklok en reset aangeroepen wordt.

Door tijdsgebrek hebben we dit niet zeer complex gemaakt. We doen een standaard overgang door een andere frequentie aan te nemen. Hier gebeurdt dit synchroon omdat we de horn synchroon aanmaken, deden we dit asynchroon kregen we niet de verwachte resultaten.



Figuur 26: Blokschema



Figuur 27: Blokschema 2

Commentaar

Op figuur 27 zien we duidelijk dat er een 2de blok sound is die los staat van het rgb-blok.

Deel 6: Kleurenfoto

Opgave

Neem een kleurenfoto naar keuze, reduceer de resolutie tot 300 op 200, laad de foto in BRAM van de FPGA en plaats die op het LCD-scherm.

VHDL-beschrijving

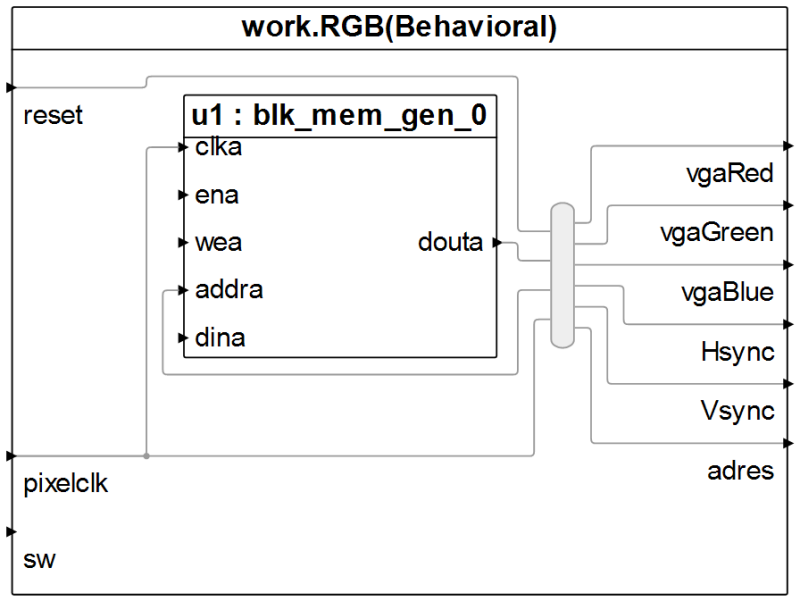
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity RGB is
    Port ( reset : in  STD_LOGIC;       -- system Reset
           pixelclk     : IN  std_logic;
           sw       : IN std_logic_vector(3 DOWNTO 0);
           vgaRed       : OUT std_logic_vector(3 DOWNTO 0);
           vgaGreen     : OUT std_logic_vector(3 DOWNTO 0);
           vgaBlue      : OUT std_logic_vector(3 DOWNTO 0);
           Hsync, Vsync : OUT std_logic;
           adres : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
              );
end RGB;

architecture Behavioral of RGB is
constant innerCol : integer := 800;
constant innerRow : integer := 600;
constant horizontalBlanking : integer := 256;
constant verticalBlanking : integer := 28;
constant totalRow : integer := innerRow + verticalBlanking;
constant totalCol : integer := innerCol + horizontalBlanking;
constant width : integer := 300; --breedte foto
constant height : integer := 200; --hoogte foto

component blk_mem_gen_0 is
port(
       clka : IN STD_LOGIC;
          ena : IN STD_LOGIC;
          wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
          addra : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
          dina : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
          douta : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
);
end component;
signal addra : std_logic_vector(15 downto 0);
signal douta : std_logic_vector(15 downto 0);
begin
u1: blk_mem_gen_0 port map
(
   clka => pixelclk,
   ena => '1',
   wea => "0",
   addra => addra,
   dina => (others => '0'),
   douta => douta
);
process(reset, pixelclk)
       variable col : integer range 0 to totalCol := 0;
       variable row : integer range 0 to totalRow := 0;
       VARIABLE red : std_logic_vector(3 downto 0);
```

```
	VARIABLE green : std_logic_vector(3 downto 0);
	VARIABLE blue : std_logic_vector(3 downto 0);
	VARIABLE adr : std_logic_vector(15 downto 0) := (others => '0');
	variable draw : integer range 0 to 1:= 0;
begin
	if reset = '1' then
			col := 0;
			row := 0;
			adr := (others => '0');
			draw := 0;
	elsif rising_edge(pixelclk) then
		col := col + 1;
		if col = totalCol then
			col := 0;
			row := row + 1;
			if row = totalRow then
				row := 0;
			end if;
		end if;
	end if;

	if col > 39 and col < 128 + 40 then
		Hsync <= '1';
	else
		Hsync <= '0';
	end if;

	if row > 0 and row < 4 + 1 then
		Vsync <= '1';
		adr := (others => '0');
	else
		Vsync <= '0';
	end if;


	if col > horizontalBlanking and col < width then
		if row > verticalBlanking and row < height then
			draw := 1;
		else
			draw := 0;
		end if;
	else
		draw := 0;
	end if;

if rising_edge(pixelclk) then
	if col > horizontalBlanking and row > verticalBlanking then
		case draw is
			when 1 =>  red := douta(11 downto 8);
				green := douta(7 downto 4);
				blue := douta(3 downto 0);
				adr := adr+1;
			when others =>  red := (others => '0');
				green := (others => '0');
				blue := (others => '0');
		end case;
	else
		vgaRed <= (others => '0');
		vgaGreen <= (others => '0');
```

```
            vgaBlue <= (others => '0');
        end if;
        vgaRed <= red;
        vgaGreen <= green;
        vgaBlue <= blue;
end if;

        vgaRed <= red;
        vgaGreen <= green;
        vgaBlue <= blue;
        addra <= adr;
        adres <= adr;
end process;
end Behavioral;
```

Redenering + blokschema

Er wordt een .coe-file ingeladen van een afbeelding met breedte 300 en hoogte 200 in een aparte BRAM-blok. Elke keer als er een pixel van de afbeelding moet verschijnen (i.e. linksboven), worden de kleuren van de .coe-file ingeladen (douta) en wordt het volgende adres aangewezen. Buiten de afbeeldingszone mag het adres niet verspringen en wordt zwart weergegeven.



Figuur 28: Blokschema

Commentaar

Dit deel hebben we niet meer kunnen testen.

Lab 5: Extra Oefeningen

Oef 1 Vermenigvuldiger

VHDL-beschrijving

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity vermenigvuldiger is
port
	(
		A, B   : IN STD_LOGIC_VECTOR(15 downto 0);
		res    : OUT STD_LOGIC_VECTOR(31 downto 0);
		clk : IN  STD_LOGIC
	);
end entity;

architecture Behavioral of vermenigvuldiger is
constant AB_Bits : integer := 15; --aantal bits in A of B
constant res_Bits : integer := 31; --aantal bits in res
begin
process(A, B, clk)
	variable A_temp: STD_LOGIC_VECTOR(res_Bits downto 0);
	variable res_temp: STD_LOGIC_VECTOR(res_Bits downto 0);
	begin
	if rising_edge(clk) then
		A_temp := (others => '0');
		A_temp := A_temp + A;
		res_temp := (others => '0');
		for i in 0 to AB_Bits loop
			if B(i) = '1' then --als huidig bit = 1 -> optellen, anders 0
				res_temp := res_temp + A_temp;
			end if;
			A_temp := A_temp(res_Bits-1 downto 0) & '0'; --opschuiven links
		end loop;
		res <= res_temp;
	end if;
end process;
end Behavioral;
```

Testbench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;


entity vermenigvuldiger_tb is
end vermenigvuldiger_tb;

architecture Behavioral of vermenigvuldiger_tb is
COMPONENT vermenigvuldiger
	Port ( clk : in  STD_LOGIC;
		A, B : IN std_logic_vector(15 downto 0);
		res: OUT std_logic_vector(31 downto 0));
END COMPONENT;
--Inputs
 signal clk : std_logic := '0';
```

```
 signal A : std_logic_vector(15 downto 0) := (others => '0');
 signal B : std_logic_vector(15 downto 0) := (others => '0');
--Outputs
 signal res : std_logic_vector(31 downto 0) := (others => '0');
 constant clk_period : time := 100 ns;
begin
 uut: vermenigvuldiger PORT MAP (
    clk => clk,
    A => A,
    B => B,
    res => res
   );
 -- Clock process definitions
clk_process :process
begin
   clk <= '0';
   wait for clk_period/2;
   clk <= '1';
   wait for clk_period/2;
   end process;
stim_proc: process
    begin
       wait for 100 ns;
       A <= "0100100001010100";
       B <= "0100000100000101";
       wait;
    end process;
END;
```

Commentaar

De maximale bewerkingen zijn gelinkt aan de maximale frequentie.

Oef 2 Botsing

VHDL-beschrijving

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

entity botsing is
    Port ( PC : in STD_LOGIC_VECTOR (31 downto 0);
           clk : in STD_LOGIC;
           bots : out STD_LOGIC);
end botsing;

architecture Behavioral of botsing is
begin
process(clk)
      variable n : integer range 0 to 31;
begin
      if rising_edge(clk) then
            n := 0;
            for i in PC'range loop
                if PC(i) = '1' then
                  n := n + 1;
                end if;
            end loop;
      end if;
      if(n <= 1) then
            bots <= '0';
      elsif (n > 1) then
            bots <= '1';
      end if;
end process;

end Behavioral;
```

Testbench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;


entity botsing_tb is
end botsing_tb;

architecture Behavioral of botsing_tb is
COMPONENT botsing
      Port ( PC : in STD_LOGIC_VECTOR (31 downto 0);
           clk : in STD_LOGIC;
           bots : out STD_LOGIC);
END COMPONENT;
--Inputs
 signal clk : std_logic := '0';
 signal PC : std_logic_vector(31 downto 0) := (others => '0');
--Outputs
  signal bots : std_logic := '0';
 constant clk_period : time := 100 ns;
begin
 uut: botsing PORT MAP (
    clk => clk,
    bots => bots,
```

```
    PC => PC
   );
 -- Clock process definitions
clk_process :process
begin
   clk <= '0';
   wait for clk_period/2;
   clk <= '1';
   wait for clk_period/2;
   end process;
stim_proc: process
    begin
       wait for 100 ns;
       PC <= "01001000010101000100100001010100";
       wait for 100 ns;
       PC <= "00001000000000000000000000000000";
       wait for 100 ns;
       PC <= "00000000000000000100000001000000";
       wait for 100 ns;
       PC <= (others => '0');
       wait;
    end process;
END;
```

Besluit

Hier volgen een paar besluiten die ik doorheen de labos genomen heb.

1) Behavioral/functionele simulatie = puur vhdl code, geen hardware, geen vertragingen.
2) Post-route simulatie = simulatie met vertraging die je krijgt door de fpga.
3) Synchroon versus asynchroon deel van de code is zeer belangrijk, niet alles synchroon zetten door geheugenwerking maar ook niet alles asynchroon doen.
4) Best een maximale waarde toekennen wanneer je variabelen aanmaakt, anders heb je een overschot aan resources die nooit gebruikt zullen worden.
5) De maximale frequentie ligt tussen 100 en 200MHz.
6) Een proces wordt opgeroepen wanneer 1 van de signalen uit de sensitivitylist verandert.

Besluiten na het maken van de labos:

Veel code die ik doorheen de labos gemaakt heb zou ik veel efficienter kunnen maken. Maar dit is de bedoeling van het labo, je leert bij en geleidelijk aan begrijp je hoe je het best vhdl opstelt.