EE1410: Digitale Systemen

BSc. EE, 1e jaar, 2011-2012, 2e werkcollege

Arjan van Genderen, Stephan Wong, Computer Engineering

15 t/m 22-3-2012

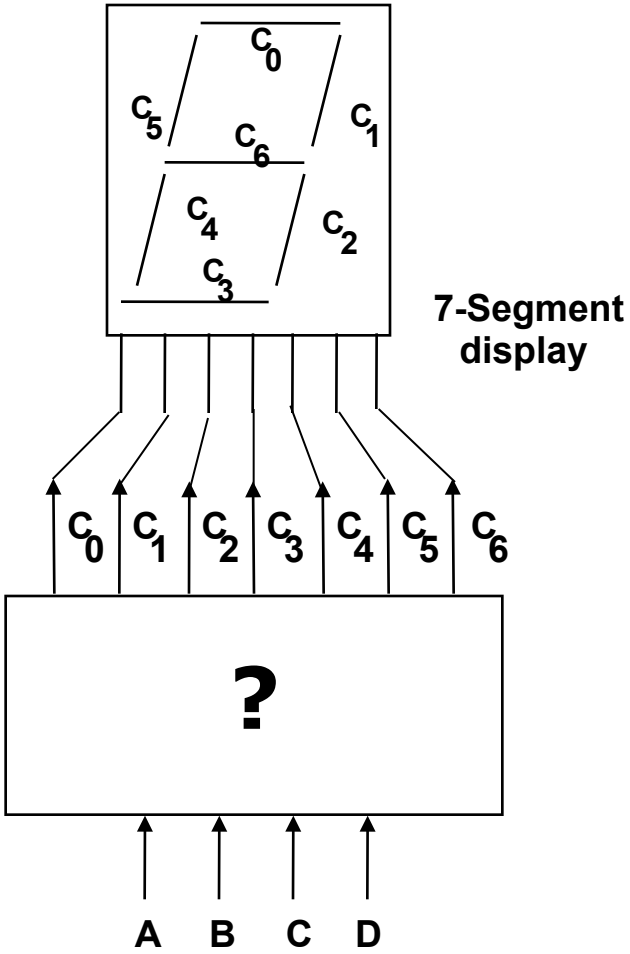
Voor je begint

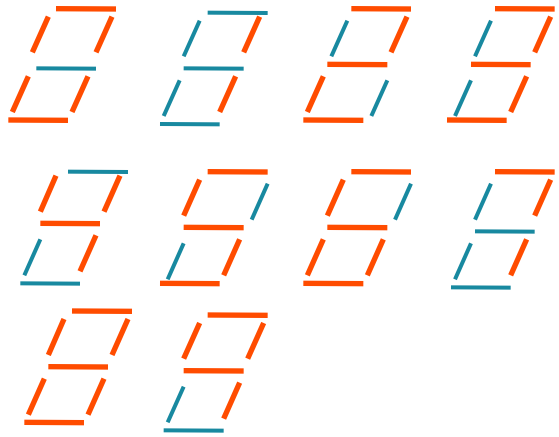
1. Download het bestand werkcolllege2.zip van Blackboard -> EE1410 -> Lectures -> Werkcollege 2, en pak het uit op een plaats waar je schrijfrechten hebt (bijv. H:). Er wordt nu een map werkcollege2 aangemaakt.
2. Start ModelSim met Programs->Engineering->Modelsim SE 6.2a -> ModelSim en maak een nieuw project (File -> New -> Project) in de map werkcollege2.
3. Voeg de VHDL files seg7dec.vhd en seg7dec_tb.vhd alvast toe aan het project.
4. In de map werkcollege2 staat een copie van deze slides (zonder antwoorden).

Werkcollege 2

- Ontwerp combinatorische schakeling: 7 segments decoder
  - VHDL
  - K-maps
  - Synthese voor FPGA
- Ontwerp sequentieele schakeling: 3-bit counter/pulsgenerator
  - FSMs
  - VHDL

Excess-3 to 7 Segment Decoder





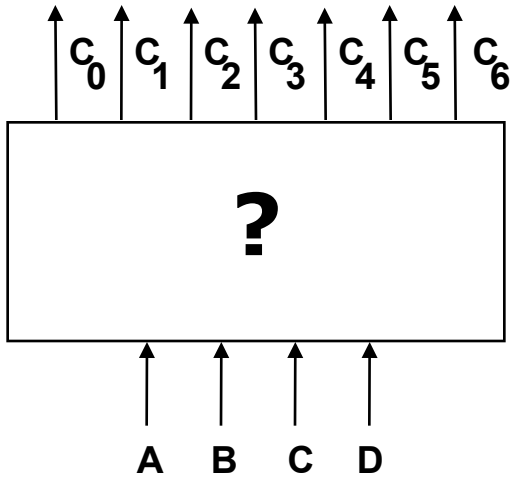
excess-3 codering

A	B	C	D	display
0	0	1	1	0
0	1	0	0	1
0	1	0	1	2
0	1	1	0	3
0	1	1	1	4
1	0	0	0	5
1	0	0	1	6
1	0	1	0	7
1	0	1	1	8
1	1	0	0	9

Analyse Excess-3 to 7 Segment Decoder

Doorgronden van het probleem

- input: 4 bit excess-3 digit
- output: control signalen voor het display
- 4 inputs A, B, C, D
- 7 outputs C_0, C_1, ..., C_6



Formuleren van het probleem mbv. een waarheidstabel

A	B	C	D	C_0	C_1	C_2	C_3	C_4	C_5	C_6
0	0	1	1	1	1	1	1	1	1	0
0	1	0	0	0	1	1	0	0	0	0
0	1	0	1	1	1	0	1	1	0	1
0	1	1	0	1	1	1	1	0	0	1
0	1	1	1	0	1	1	0	0	1	1
1	0	0	0	1	0	1	1	0	1	1
1	0	0	1	1	0	1	1	1	1	1
1	0	1	0	1	1	1	0	0	0	0
1	0	1	1	1	1	1	1	1	1	1
1	1	0	0	1	1	1	0	0	1	1
1	1	0	1	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X
0	0	0	0	X	X	X	X	X	X	X
0	0	0	1	X	X	X	X	X	X	X
0	0	1	0	X	X	X	X	X	X	X

Schrijven van VHDL code

zie volgende slide

Excess-3 to 7 Segment Decoder in VHDL

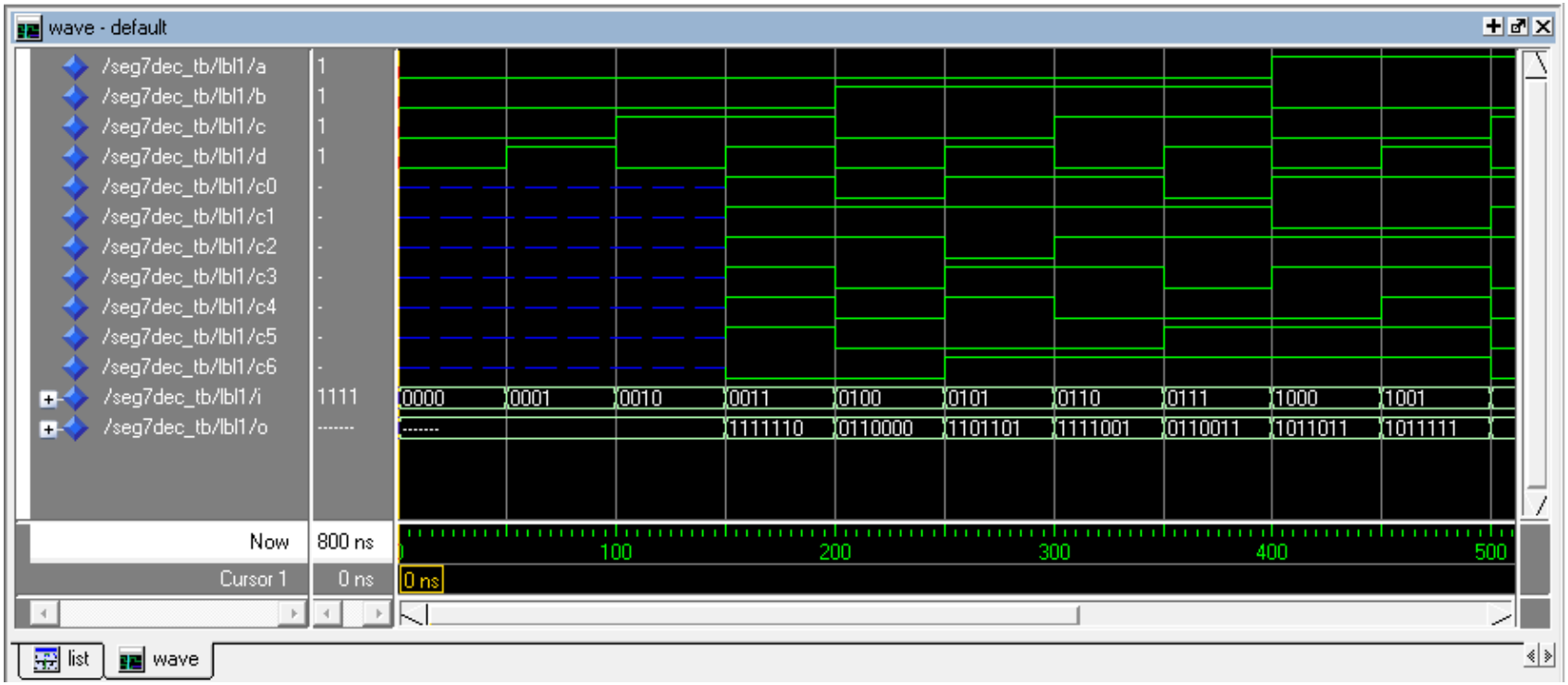
```
library IEEE;
use IEEE.std_logic_1164.ALL;

entity seg7dec is
    port (A, B, C, D : in std_logic;
          C0, C1, C2, C3,
          C4, C5, C6 : out std_logic
    );
end entity seg7dec;
```

```
architecture behaviour of seg7dec is
    signal I : std_logic_vector(0 to 3);
    signal O : std_logic_vector(0 to 6);
begin
    I <= (A, B, C, D);

    comb: process (I)
    begin
        case I is
            when "0011" => O <= "1111110";  -- 0
            when "0100" => O <= "0110000";  -- 1
            when "0101" => O <= "1101101";  -- 2
            when "0110" => O <= "1111001";  -- 3
            when "0111" => O <= "0110011";  -- 4
            when "1000" => O <= "1011011";  -- 5
            when "1001" => O <= "1011111";  -- 6
            when "1010" => O <= "1110000";  -- 7
            when "1011" => O <= "1111111";  -- 8
            when "1100" => O <= "1110011";  -- 9
            when others => O <= "-------";
        end case;
    end process;

    (C0, C1, C2, C3, C4, C5, C6) <= O;
end behaviour;
```

Testbench 7 Segment Decoder in VHDL

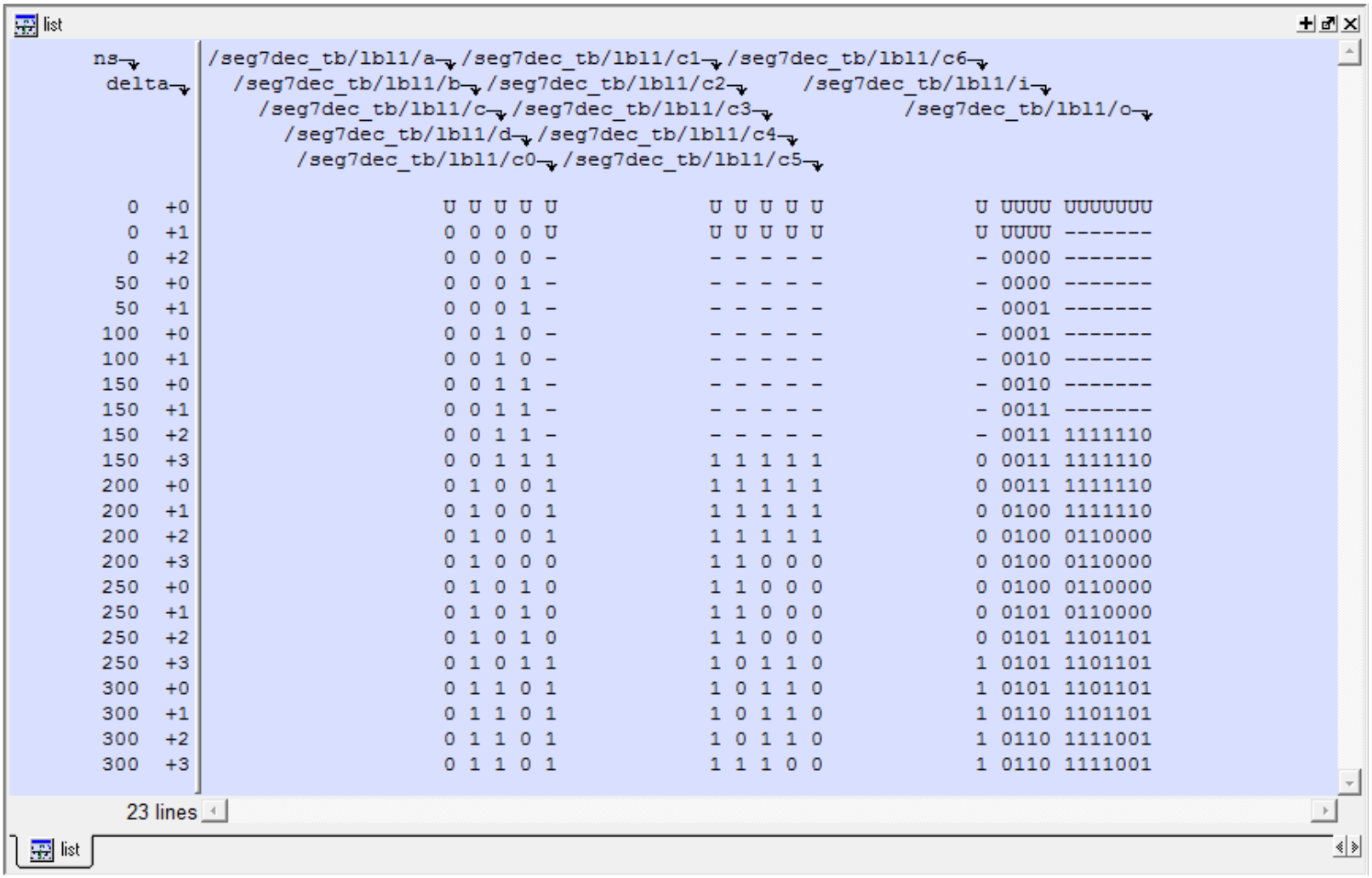
```
library IEEE;
use IEEE.std_logic_1164.ALL;

entity seg7dec_tb is
end entity seg7dec_tb;

architecture behaviour of seg7dec_tb is
    component seg7dec
        port (A, B, C, D : in std_logic;
            C0, C1, C2, C3,
            C4, C5, C6 : out std_logic
        );
    end component;

    signal A, B, C, D : std_logic;
    signal C0, C1, C2, C3,
           C4, C5, C6 : std_logic;
begin

    lbl1: seg7dec port map (A, B, C, D,
             C0, C1, C2, C3, C4, C5, C6);

    A <= '0' AFTER   0 NS,
         '1' AFTER 400 NS;

    B <= '0' AFTER   0 NS,
         '1' AFTER 200 NS,
         '0' AFTER 400 NS,
         '1' AFTER 600 NS;

    C <= '0' AFTER   0 NS,
         '1' AFTER 100 NS,
         '0' AFTER 200 NS,
         '1' AFTER 300 NS,
         '0' AFTER 400 NS,
         '1' AFTER 500 NS,
         '0' AFTER 600 NS,
         '1' AFTER 700 NS;

    D <= '0' AFTER   0 NS,
         '1' AFTER  50 NS,
         '0' AFTER 100 NS,
         '1' AFTER 150 NS,
         '0' AFTER 200 NS,
         '1' AFTER 250 ns,
         '0' AFTER 300 NS,
         '1' AFTER 350 NS,
         '0' AFTER 400 NS,
         '1' AFTER 450 NS,
         '0' AFTER 500 NS,
         '1' AFTER 550 NS,
         '0' AFTER 600 NS,
         '1' AFTER 650 ns,
         '0' AFTER 700 NS,
         '1' AFTER 750 NS;
end behaviour;
```

- Simuleer seg7dec.vhd met seg7dec_tb.vhd
- Klik op lbl1 en gebruik Add -> Add to Wave en Add -> Add to List.
- Probeer bij List -> List Preferences... : Expand Deltas of Collapse Deltas


wave - default
/seg7dec_tb/lbl1/a
/seg7dec_tb/lbl1/b
/seg7dec_tb/lbl1/c
/seg7dec_tb/lbl1/d
/seg7dec_tb/lbl1/c0
/seg7dec_tb/lbl1/c1
/seg7dec_tb/lbl1/c2
/seg7dec_tb/lbl1/c3
/seg7dec_tb/lbl1/c4
/seg7dec_tb/lbl1/c5
/seg7dec_tb/lbl1/c6
/seg7dec_tb/lbl1/i
/seg7dec_tb/lbl1/o
1
1
1
1
-
-
-
-
-
-
-
1111

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001

1111110
0110000
1101101
1111001
0110011
1011011
1011111
Now
800 ns
Cursor 1
0 ns
0 ns
100
200
300
400
500
list
wave



list

ns / delta	/seg7dec_tb/lbl1/a	/seg7dec_tb/lbl1/b	/seg7dec_tb/lbl1/c	/seg7dec_tb/lbl1/d	/seg7dec_tb/lbl1/c0	/seg7dec_tb/lbl1/c1	/seg7dec_tb/lbl1/c2	/seg7dec_tb/lbl1/c3	/seg7dec_tb/lbl1/c4	/seg7dec_tb/lbl1/c5	/seg7dec_tb/lbl1/c6	/seg7dec_tb/lbl1/i	/seg7dec_tb/lbl1/o
0 +0	U	U	U	U	U	U	U	U	U	U	U	UUUU	UUUUUUU
0 +1	0	0	0	0	U	U	U	U	U	U	U	UUUU	-------
0 +2	0	0	0	0	-	-	-	-	-	-	-	0000	-------
50 +0	0	0	0	1	-	-	-	-	-	-	-	0000	-------
50 +1	0	0	0	1	-	-	-	-	-	-	-	0001	-------
100 +0	0	0	1	0	-	-	-	-	-	-	-	0001	-------
100 +1	0	0	1	0	-	-	-	-	-	-	-	0010	-------
150 +0	0	0	1	1	-	-	-	-	-	-	-	0010	-------
150 +1	0	0	1	1	-	-	-	-	-	-	-	0011	-------
150 +2	0	0	1	1	-	-	-	-	-	-	-	0011	1111110
150 +3	0	0	1	1	1	1	1	1	1	1	0	0011	1111110
200 +0	0	1	0	0	1	1	1	1	1	1	0	0011	1111110
200 +1	0	1	0	0	1	1	1	1	1	1	0	0100	1111110
200 +2	0	1	0	0	1	1	1	1	1	1	0	0100	0110000
200 +3	0	1	0	0	0	1	1	0	0	0	0	0100	0110000
250 +0	0	1	0	1	0	1	1	0	0	0	0	0100	0110000
250 +1	0	1	0	1	0	1	1	0	0	0	0	0101	0110000
250 +2	0	1	0	1	0	1	1	0	0	0	0	0101	1101101
250 +3	0	1	0	1	1	1	0	1	1	0	1	0101	1101101
300 +0	0	1	1	0	1	1	0	1	1	0	1	0101	1101101
300 +1	0	1	1	0	1	1	0	1	1	0	1	0110	1101101
300 +2	0	1	1	0	1	1	0	1	1	0	1	0110	1111001
300 +3	0	1	1	0	1	1	1	1	0	0	1	0110	1111001

23 lines

list

Implementatie Excess-3 to 7 Segment Decoder

Kies de implementatie-hardware

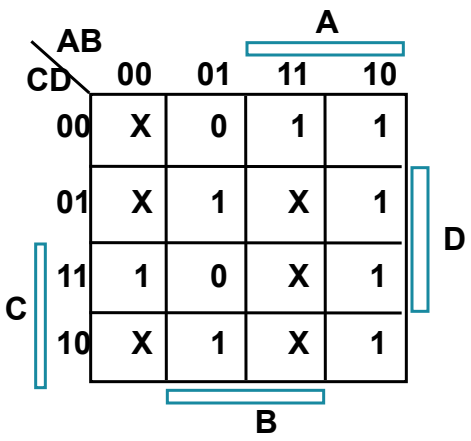
Bijv.

- **losse poorten**
  - **NORs**
  - **NANDs**
- **FPGA**

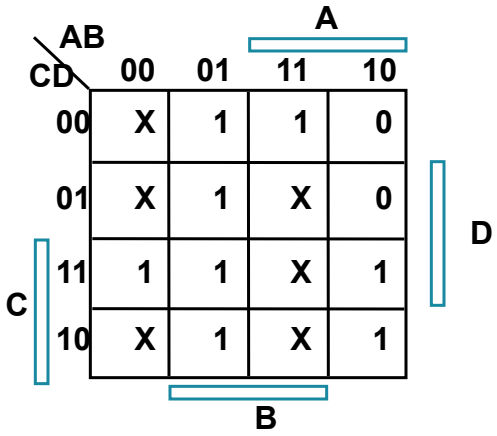
Volg de implementatie-procedure

- **K-maps**
- **Xilinx ISE / Synplify**

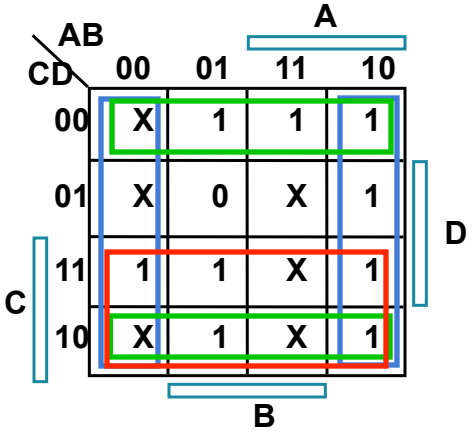
K-maps: Sum of Products



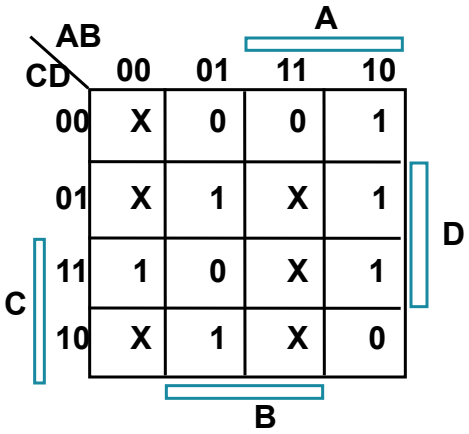
K-map for C_0



K-map for C_1



K-map for C_2



K-map for C_3

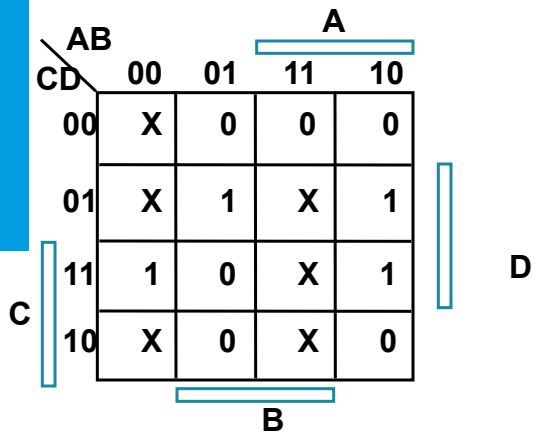
$C_0 = A + B' + CD' + C' D$

$C_1 = B + C$

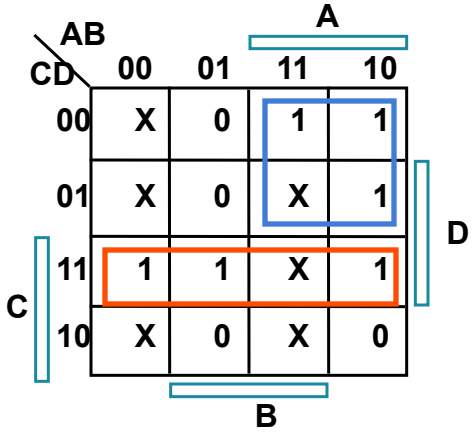
$C_2 = B' + C + D'$

$C_3 = B'C' + B'D + C'D + BCD'$

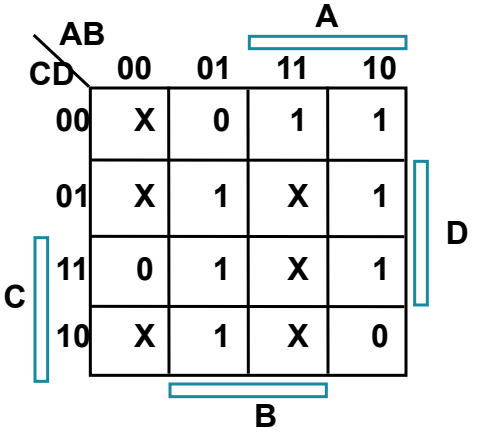
K-maps: Sum of Products: NANDs



K-map for C_4



K-map for C_5

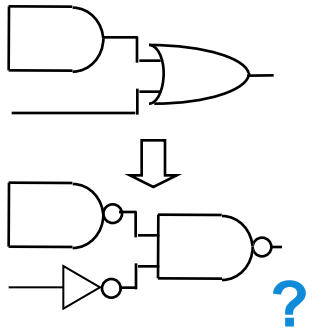


K-map for C_6

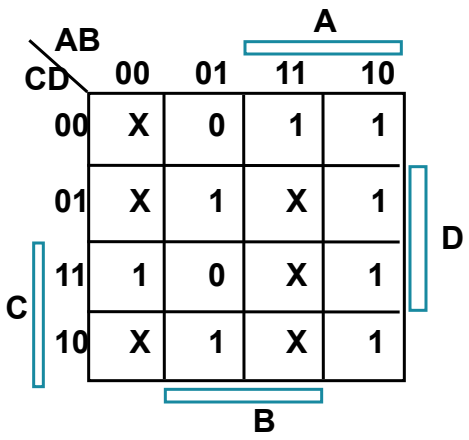
$C_0 = A + B' + CD' + C' D$
$C_1 = B + C$
$C_2 = B' + C + D'$
$C_3 = B'C' + B'D + C'D + BCD'$
$C_4 = B'D + C'D$
$C_5 = AC' + CD$
$C_6 = AC' + AD + BC + C'D$

14 unieke product-termen:
9 omvatten meer dan 1 variabele
3 geinverteerde ingangen nodig

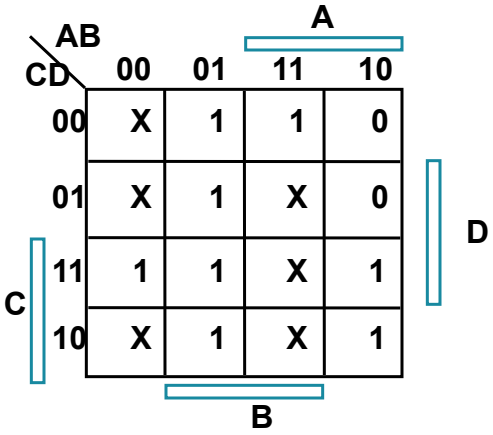
Benodigde losse poorten:
3 INV, 9 AND, 7 OR
Wanneer alleen INV en NAND:
4 INV, 16 NAND



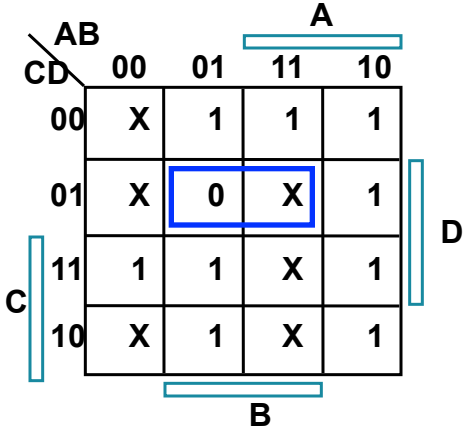
K-maps: Product of Sums



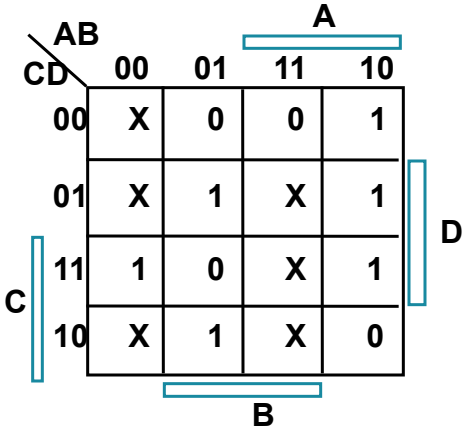
K-map for C_0



K-map for C_1



K-map for C_2



K-map for C_3

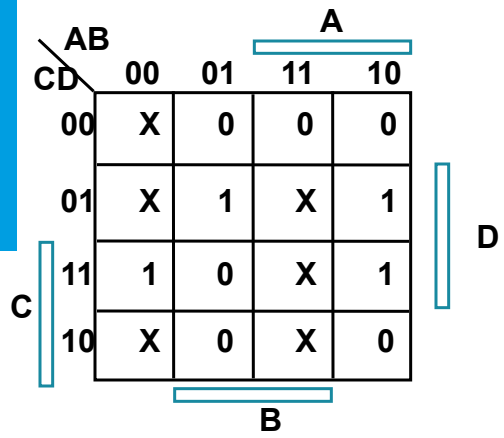
$C_0 = (A + C + D)(B' + C' + D')$

$C_1 = B + C$

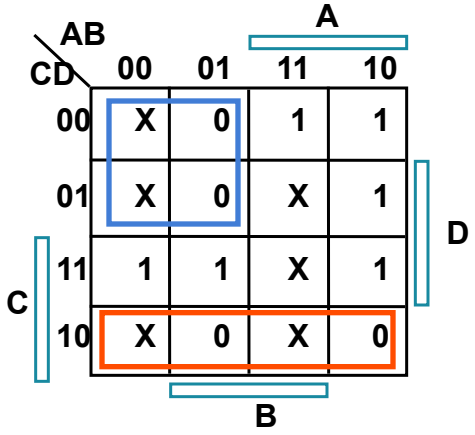
$C_2 = B' + C + D'$

$C_3 = (B' + C + D)\ (B' + C' + D')\ (B + C' + D)$

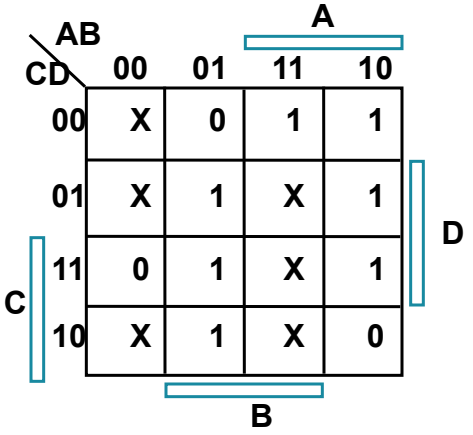
K-maps: Sum of Products: NORs



K-map for C_4



K-map for C_5

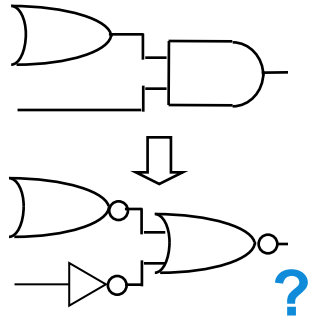


K-map for C_6

$C_0 = (A + C + D)(B' + C' + D')$
$C_1 = B + C$
$C_2 = B' + C + D'$
$C_3 = (B' + C + D)\ (B' + C' + D')$
$(B + C' + D)$
$C_4 = D\ (B' + C')$
$C_5 = (A + C)\ (C' + D)$
$C_6 = (A + C + D)\ (A + B)\ (B + C' + D)$

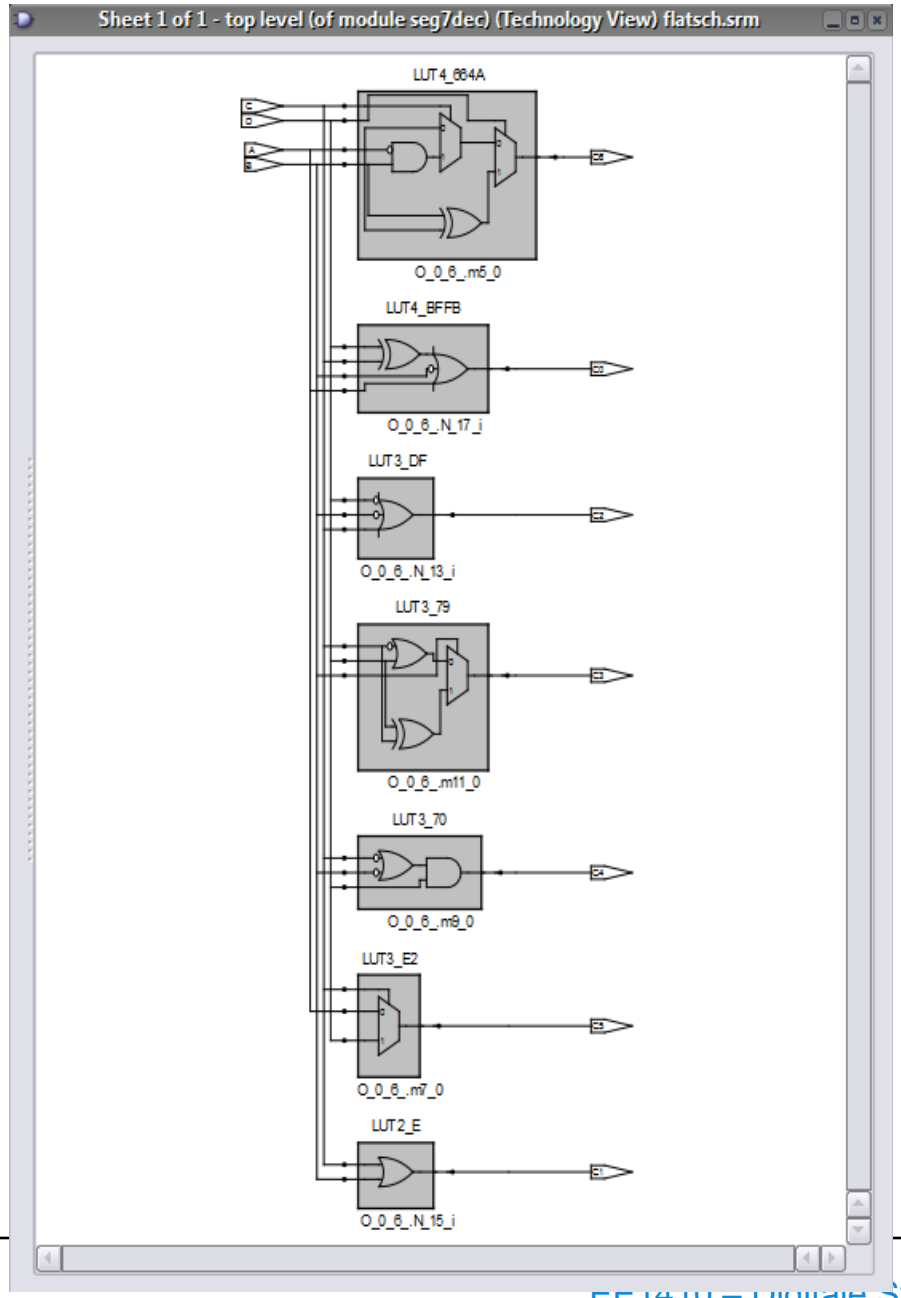
11 unieke som-termen:
10 omvatten meer dan 1 variabele
3 geinverteerde ingangen nodig

Benodigde losse poorten:
3 INV, 10 OR, 5 AND
Wanneer alleen INV en NOR:
3 INV, 15 NOR

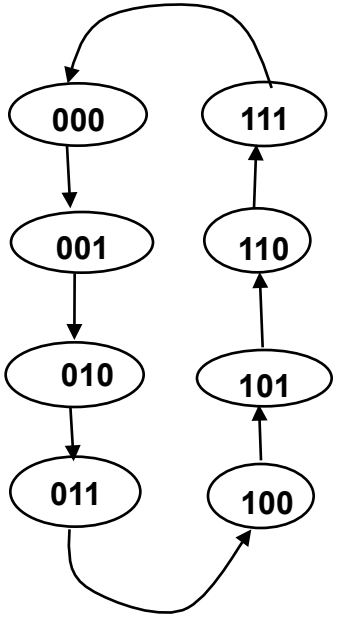


Implementatie met FPGA

1. Start het programma Synplify Pro:
   Start -> Programs -> Practica -> Synplicity -> Synplify Pro
2. Maak een nieuw project met File -> New:
   Selecteer Project File (Project) en zet File Location naar de map waar de file seg7dec.vhd staat.
3. Gebruik Add File om seg7dec.vhd toe te voegen.
4. Klik Implementation Options en selecteer in tab Device:
   Technology: Xilinx Spartan 3, Part XC3S200,
   Package: FT256, Speed -4,
   vink aan: Option: Disable I/O insertion
5. Klik Run
6. Bekijk schema met HDL-Analyst -> Technology -> Flattened View

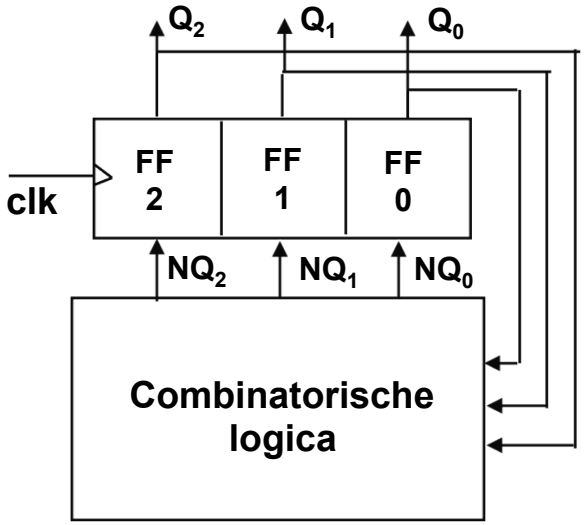

Sheet 1 of 1 - top level (of module seg7dec) (Technology View) flatsch.srm
LUT4_664A
O_0_6_.m5_0
LUT4_BFFB
O_0_6_.N_17_i
LUT3_DF
O_0_6_.N_13_i
LUT3_79
O_0_6_.m11_0
LUT3_70
O_0_6_.m9_0
LUT3_E2
O_0_6_.m7_0
LUT2_E
O_0_6_.N_15_i

3-bit Binary Counter



Toestandstabel

Huidige State			Volgende State		
Q_2	Q_1	Q_0	NQ_2	NQ_1	NQ_0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

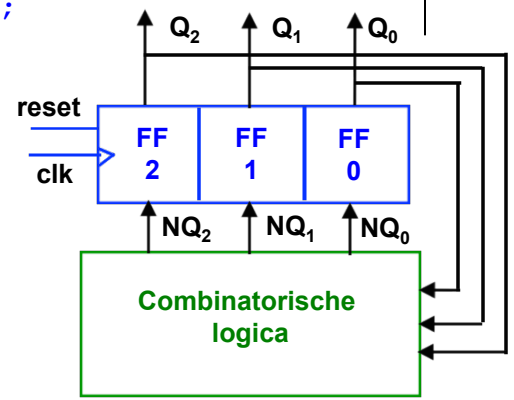


Bij counters kunnen de state bits ook gelijk als uitgangen gebruikt worden

De volgende state hangt niet af van de ingangen
(of in geringe mate via een reset, enable en/of load signaal)

Counter als FSM beschreven in VHDL

```
library IEEE;
use IEEE.std_logic_1164.ALL;

entity counter is
    port (clk, reset : in std_logic;
        P : out std_logic_vector(2 downto 0)
    );
end entity counter;

architecture fsm of counter is
    signal Q, NQ :
              std_logic_vector(2 downto 0);
begin

    reg: process (clk) -- state register
    begin
        if (clk'event and clk = '1') then
            if (reset = '1') then
                Q <= "000";
            else
                Q <= NQ ;
            end if;
        end if;
    end process;
```

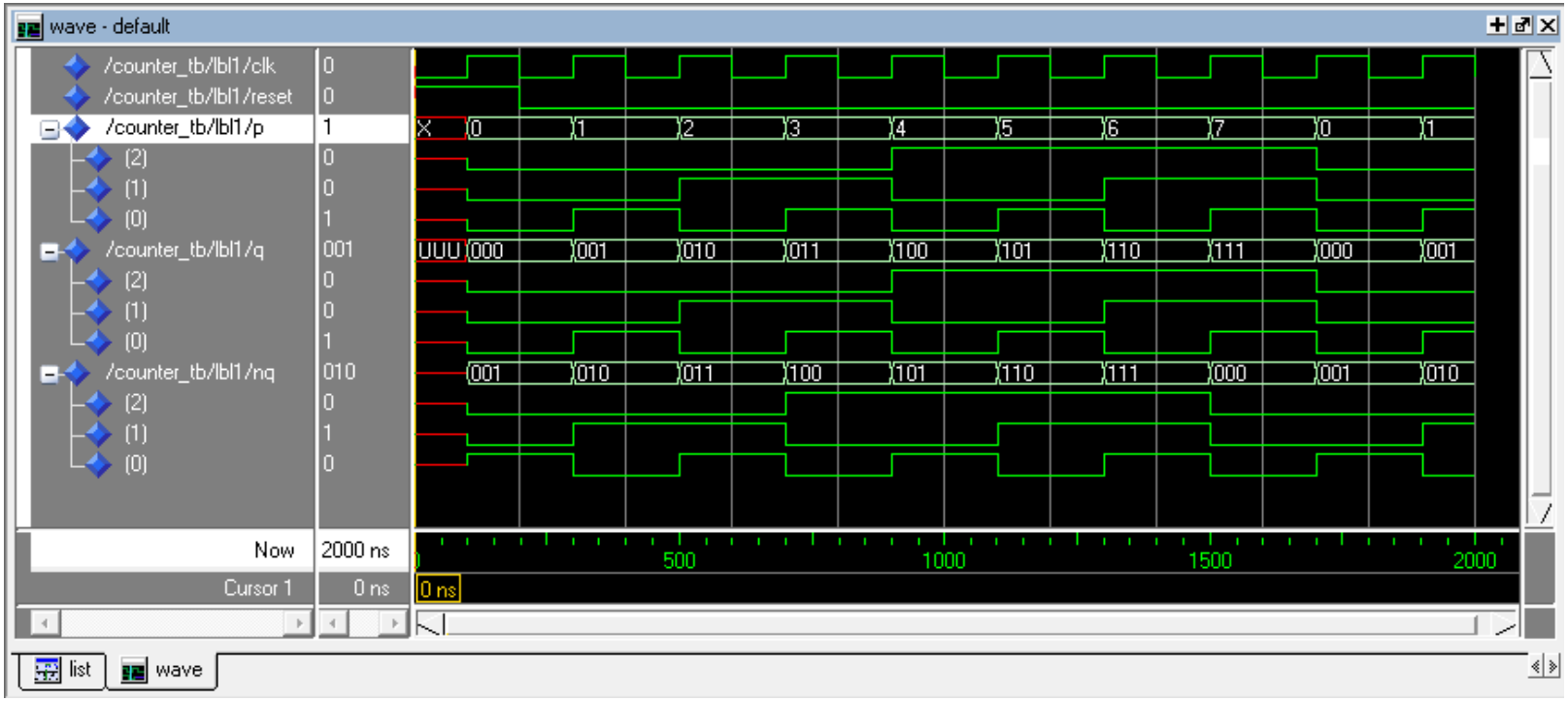



```
    comb: process (Q)
    begin
        case Q is
            when "000" =>  -- S0
                NQ <= "001";
            when "001" =>  -- S1
                NQ <= "010";
            when "010" =>  -- S2
                NQ <= "011";
            when "011" =>  -- S3
                NQ <= "100";
            when "100" =>  -- S4
                NQ <= "101";
            when "101" =>  -- S5
                NQ <= "110";
            when "110" =>  -- S6
                NQ <= "111";
            when "111" =>  -- S7
                NQ <= "000";
            when others =>
                -- to catch "UUU" etc.
                NQ <= "000";
        end case;
    end process;

    P <= Q;

end fsm;
```

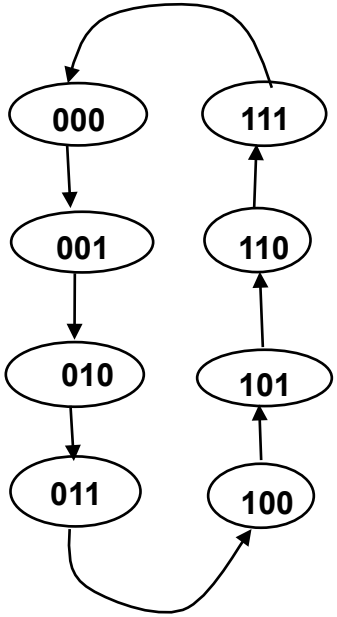
Testbench Counter

```
library IEEE;
use IEEE.std_logic_1164.ALL;

entity counter_tb is
end counter_tb;

architecture behaviour of counter_tb is
    component counter
        port (clk, reset : in std_logic;
            P : out std_logic_vector(2 downto 0)
        );
    end component;

    signal clk, reset : std_logic;
    signal P : std_logic_vector(2 downto 0);
begin

    lbl1: counter port map (clk, reset, P);

    clk <= '0' after 0 ns,
           '1' after 100 ns when clk /= '1' else
           '0' after 100 ns;

    reset <= '1' after 0 ns,
             '0' after 200 ns;

end behaviour;
```

- Simuleer counter-fsm.vhd met counter_tb.vhd
- Gebruik Add -> Add All Signals to Wave
- Klik met rechts op een bus signaal (P, Q, NQ) en probeer Radix -> Unsigned


wave - default
/counter_tb/lbl1/clk
/counter_tb/lbl1/reset
/counter_tb/lbl1/p
(2)
(1)
(0)
/counter_tb/lbl1/q
(2)
(1)
(0)
/counter_tb/lbl1/nq
(2)
(1)
(0)
0
0
1
0
0
1
001
0
0
1
010
0
1
0
X 0 1 2 3 4 5 6 7 0 1
UUU 000 001 010 011 100 101 110 111 000 001
001 010 011 100 101 110 111 000 001 010
Now
2000 ns
Cursor 1
0 ns
500
1000
1500
2000
0 ns
list
wave

Implementatie als FSM mbv. D Flip-Flops



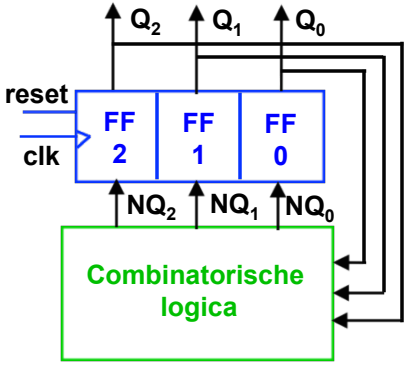
Toestandstabel

Huidige State			Volgende State		
Q_2	Q_1	Q_0	NQ_2	NQ_1	NQ_0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

NQ_1

Q_0 \ Q_2Q_1	00	01	11	10
0	0	1	1	0
1	1	0	0	1

$NQ_1 (= D_1) = Q_0' Q_1 + Q_0 Q_1'$



NQ_2

Q_0 \ Q_2Q_1	00	01	11	10
0	0	0	1	1
1	0	1	0	1

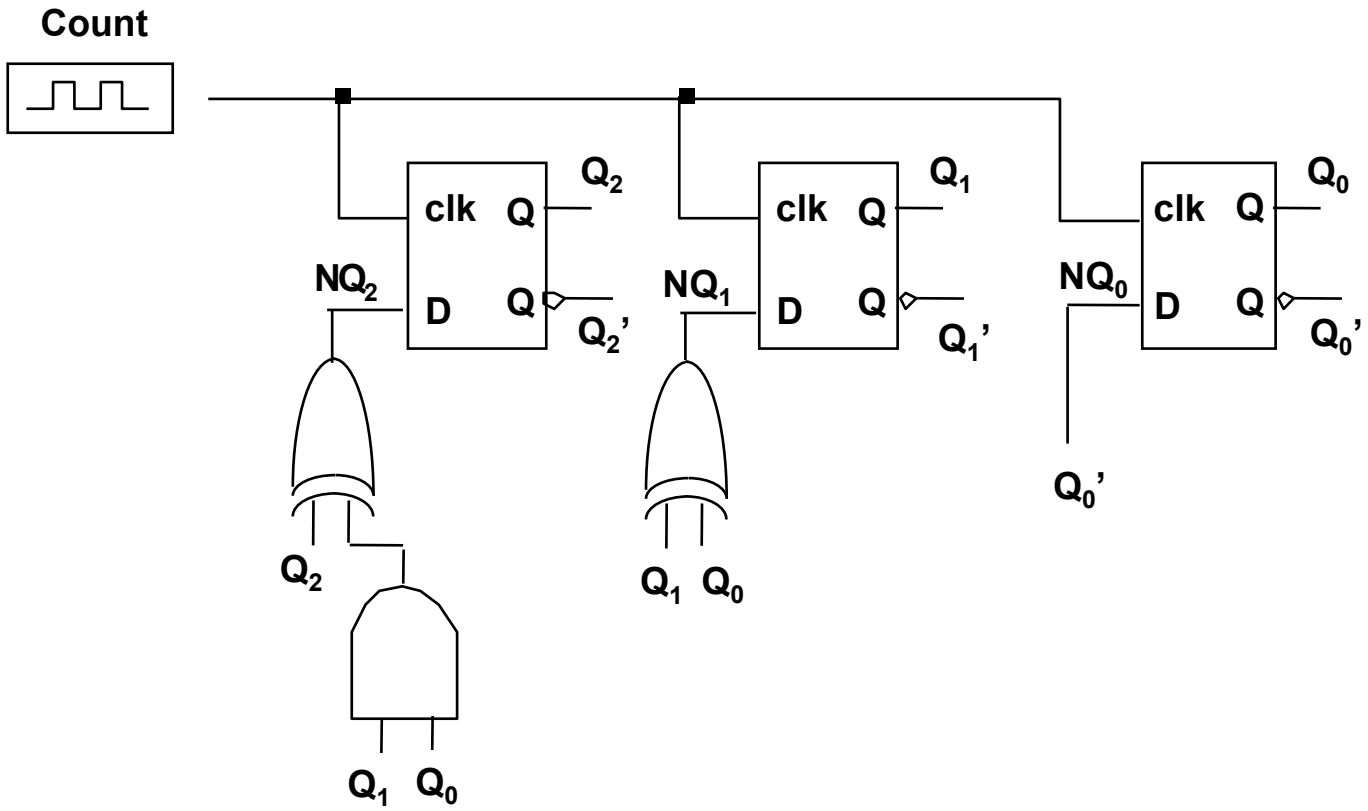
$$NQ_2 (= D_2) = Q_0' Q_2 + Q_1' Q_2 + Q_0 Q_1 Q_2'$$
$$= (Q_0' + Q_1') Q_2 + (Q_0 Q_1) Q_2'$$
$$= (Q_0 Q_1)' Q_2 + (Q_0 Q_1) Q_2'$$

NQ_0

Q_0 \ Q_2Q_1	00	01	11	10
0	1	1	1	1
1	0	0	0	0

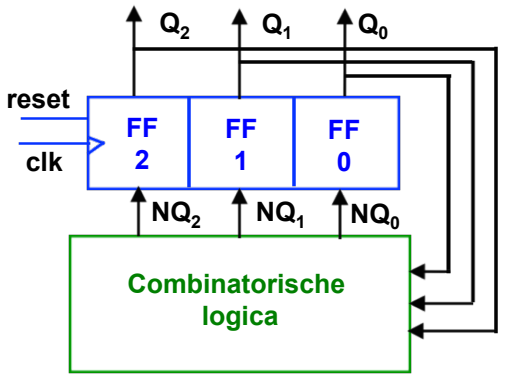
$NQ_0 (= D_0) = Q_0'$

Implementatie als FSM mbv. D Flip-Flops



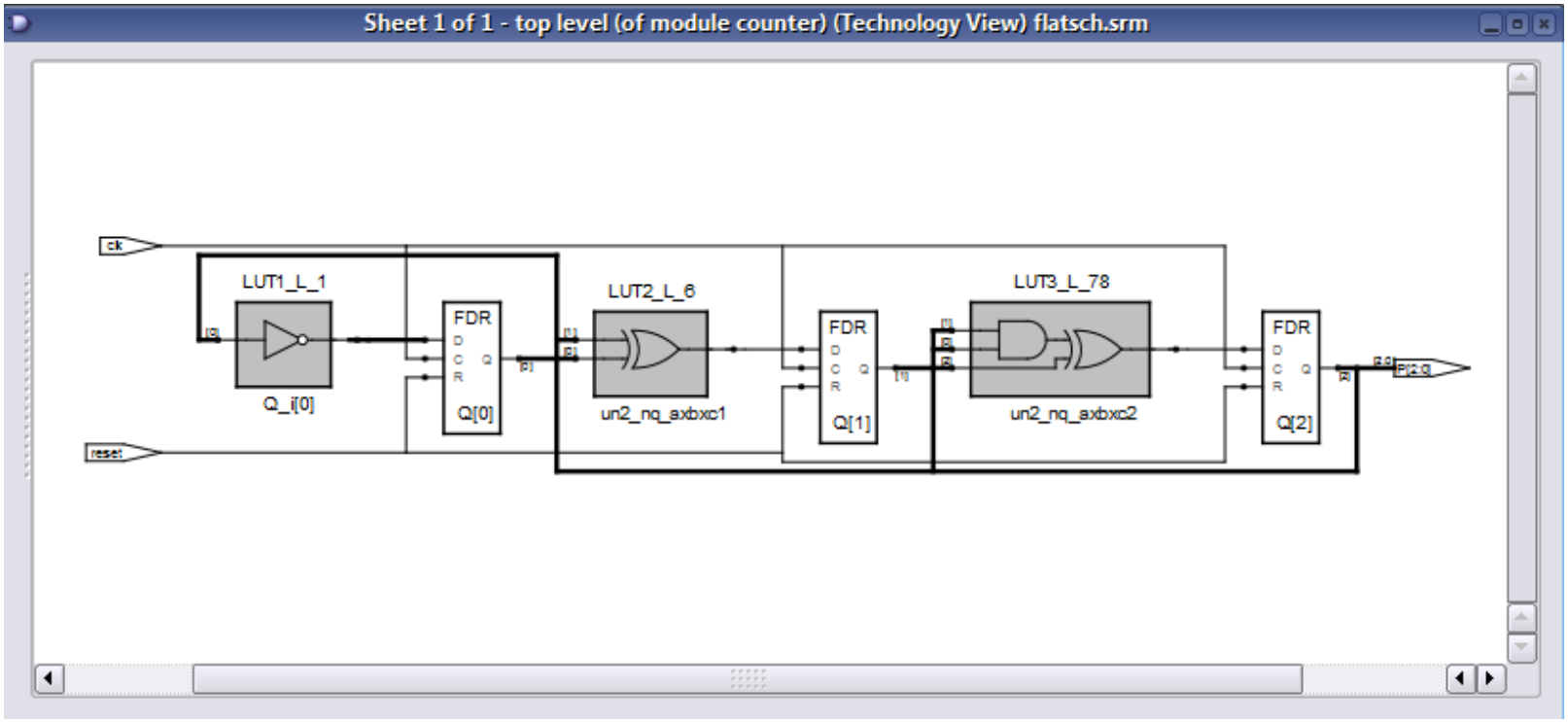
Counter arithmetisch beschreven in VHDL

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

entity counter is
    port (clk, reset : in std_logic;
          P : out std_logic_vector(2 downto 0)
    );
end entity counter;

architecture arithmetic of counter is
    signal Q, NQ : unsigned (2 downto 0);
begin
    reg: process (clk) -- state register
        begin
        if (clk'event and clk = '1') then
            if (reset = '1') then
                Q <= "000";
            else
                Q <= NQ;
            end if;
        end if;
    end process;

    comb: process (Q) -- compute NQ value
    begin
        NQ <= Q + 1;
    end process;

    P <= std_logic_vector(Q);
end arithmetic;
```



Implementatie met FPGA

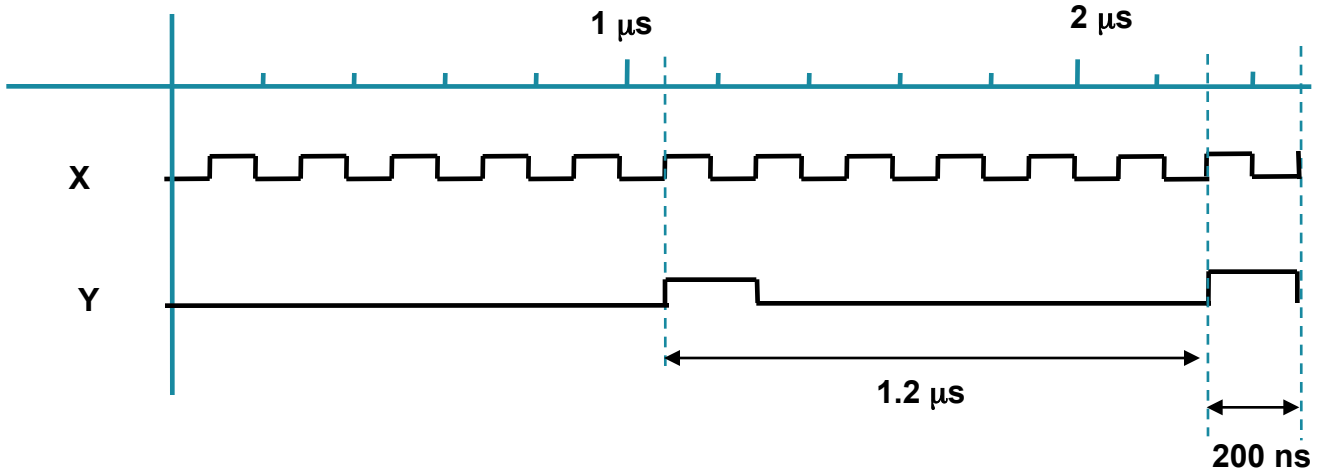
1. Gebruik Change File om seg7dec.vhd te vervangen door counter-arithmetic.vhd.
2. Klik Run
3. Bekijk schema met HDL-Analyst -> Technology -> Flattened View


Sheet 1 of 1 - top level (of module counter) (Technology View) flatsch.srm
clk
reset
LUT1_L_1
Q_i[0]
FDR
Q[0]
LUT2_L_6
un2_nq_axbxc1
FDR
Q[1]
LUT3_L_78
un2_nq_axbxc2
FDR
Q[2]
P[2:0]

Ontwerp pulsgenerator

Stel men heeft een digitaal signaal X dat elke 100 ns van waarde wisselt.

Ontwerp een circuit dat dit signaal als input heeft en iedere 1.2 μs een 200 ns puls genereert.



Hint:

X is periodiek met 200 ns periode dus met een frequentie f_x = 1/200 = 50 MHz

Y is periodiek met 1200 ns periode dus met een frequentie f_y = 1/1200 = f_x /6

Dus bedenk een manier om de frequentie van X door 6 te delen!

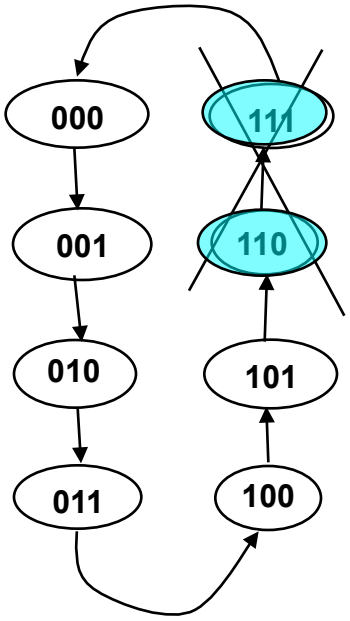
Ontwerp pulsgenerator: 6-deler

Aanpak: tel de pulsen van X en produceer een Y puls na 6 pulsen van X.

Hint: modificeer de 3-bit counter

Oplossing:

- gebruik X als clock signaal
- elimineer de laatste twee states van de counter, dwz. next_state(101) = 000
- het Y output signaal moet "1" zijn alleen in de state 101



Toestandstabel

Huidige State Q_2 Q_1 Q_0	Volgende State NQ_2 NQ_1 NQ_0	Y
0 0 0	0 0 1	0
0 0 1	0 1 0	0
0 1 0	0 1 1	0
0 1 1	1 0 0	0
1 0 0	1 0 1	0
1 0 1	0 0 0	1

6-deler VHDL Beschrijving en Testbench

- Copieer counter-fsm.vhd naar divider6.vhd en maak een entity divider6 met een extra uitgangsignaal Y en een bijbehorende achitecture.

- Simuleer met divider6_tb.vhd

```
library IEEE;
use IEEE.std_logic_1164.ALL;

entity divider6_tb is
end divider6_tb;

architecture behaviour of divider6_tb is
    component divider6
        port (clk, reset : in std_logic;
            P : out std_logic_vector
                                (2 downto 0);
            Y : out std_logic
        );
    end component;

    signal X, reset, Y : std_logic;
    signal P : std_logic_vector(2 downto 0);
begin
    lbl1: divider6 port map (X, reset, P, Y);

    X <= '0' after 0 ns,
         '1' after 100 ns when X /= '1' else
         '0' after 100 ns;

    reset <= '1' after 0 ns,
             '0' after 200 ns;
end behaviour;
```

6-deler VHDL Beschrijving

```
library IEEE;
use IEEE.std_logic_1164.ALL;

entity divider6 is
    port (clk, reset : in std_logic;
        P : out std_logic_vector(2 downto 0);
        Y : out std_logic);
end divider6;

architecture behaviour of divider6 is
    signal Q, NQ : std_logic_vector(2 downto 0);
begin
    reg: process (clk) -- state register
    begin
        if (clk'event and clk = '1') then
            if (reset = '1') then
                Q <= "000";
            else
                Q <= NQ ;
            end if;
        end if;
    end process;
```

```
    comb: process (Q)
    begin
        case Q is
            when "000" =>  -- S0
                NQ <= "001"; Y <= '0';
            when "001" =>  -- S1
                NQ <= "010"; Y <= '0';
            when "010" =>  -- S2
                NQ <= "011"; Y <= '0';
            when "011" =>  -- S3
                NQ <= "100"; Y <= '0';
            when "100" =>  -- S4
                NQ <= "101"; Y <= '0';
            when "101" =>  -- S5
                NQ <= "000"; Y <= '1';
            when others =>
                NQ <= "000"; Y <= '0';
        end case;
    end process;

    P <= Q;

end behaviour;
```

6-deler Implementatie mbv. D Flip-Flops

Huidige State $Q_2\ Q_1\ Q_0$	Volgende State $NQ_2\ NQ_1\ NQ_0$	Y
0 0 0	0 0 1	0
0 0 1	0 1 0	0
0 1 0	0 1 1	0
0 1 1	1 0 0	0
1 0 0	1 0 1	0
1 0 1	0 0 0	1

Q_0 \ Q_2Q_1	00	01	11	10
0	0	1	X	0
1	1	0	X	0

$D_1 = Q_0'Q_1 + Q_0Q_1'Q_2'$

Q_0 \ Q_2Q_1	00	01	11	10
0	1	1	X	1
1	0	0	X	0

$D_0 = Q_0'$

Q_0 \ Q_2Q_1	00	01	11	10
0	0	0	X	1
1	0	1	X	0

$D_2 = Q_0'Q_2 + Q_0Q_1$

Q_0 \ Q_2Q_1	00	01	11	10
0	0	0	X	0
1	0	0	X	1

$Y = Q_0Q_2$

6-deler Implementatie mbv. D Flip-Flops