Lösningsförslag övningsuppgifter 2025-03-21

1. Du ska realisera en 4-bitars komparator med insignaler ABCD samt utsignaler X, Y och Z.

Komparatorn ska fungera enligt nedan:

- Om AB > CD => XYZ = 100 => X = 1 indikerar att AB är större än CD.
- Om AB = CD => XYZ = 010 => Y = 1 indikerar att AB och CD är samma.
- Om AB < CD => XYZ = 001 => Z = 1 indikerar att AB är mindre än CD.

a) Ta fram minimerade ekvationer för utsignaler X, Y och Z, antingen matematiskt eller via Karnaugh-diagram.

b) Realisera motsvarande grindnät för hand.

c) Implementera konstruktionen i VHDL via en modul döpt *comparator_4bit.* Välj FPGA-kort Terasic DE0 (enhet 5CEBA4F23C7). Toppmodulen ska ha samma namn som projektet.

d) Skapa en testbänk döpt *comparator_4bit_tb* och simulera samtliga 16 kombinationer 0000 – 1111 av insignaler ABCD under 10 ns var.

e) Validera konstruktionen i ModelSim. Inspektera att in- och utsignalerna matchar ovanstående sanningstabell.

f) Verifiera konstruktionen på ett FPGA-kort. Anslut insignaler ABCD till var sin slide-switch och utsignaler XYZ till var sin lysdiod, se databladet för PIN-nummer (finns här).

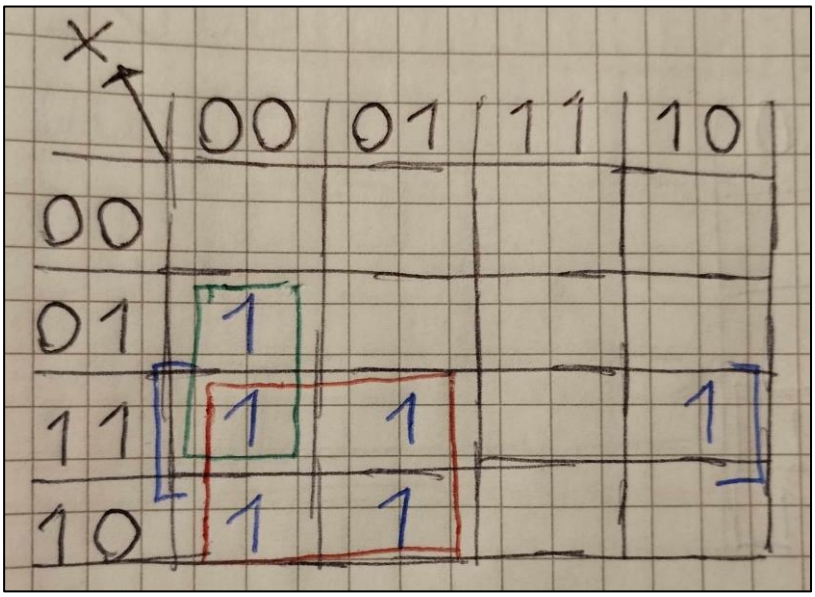
Lösning

Vi tar fram en sanningstabell för komparatorn i enlighet med beskrivningen ovan:
Sanningstabellen för grindnätet visas nedan:

ABCD	XYZ
0000	010
0001	001
0010	001
0011	001
0100	100
0101	010
0110	001
0111	001
1000	100
1001	100
1010	010
1011	001
1100	100
1101	100
1110	100
1111	010

Tabell 1: Sanningstabell för komparatorn.

Vi ritar om sanningstabellen ovan till nedanstående Karnaugh-diagram för respektive utsignal X, Y och Z.
Vi börjar med att rita Karnaugh-diagram för utsignal X:



Figur 1: Karnaugh-diagram för komparatorns utsignal X.

Vi placerar insignaler AB i y-led samt insignaler CD i x-led. Vi placerar AB samt CD i 2-bitars Grey-kod, alltså i ordningsföljden 00, 01, 11, 10, så att samtliga celler har en bit gemensam med samtliga intilliggande celler, inklusive ytterkanterna.

Vi lägger till ettor i de celler där X = 1. I sanningstabellen ser vi att X = 1 för kombinationer ABCD = 0100, 1000, 1001, 1100, 1101 samt 1110. Vi struntar i att skriva ut nollor i övriga celler, då vi enbart är intresserade av ettorna.

Vi noterar i Karnaugh-diagrammet ovan att vi får tre block innehållande ettor. Vi ringar in dessa block med röd, grön respektive blå färg. De fyra ettor som är inringade i rött har gemensamt att A = 1 samt C = 0. De två ettor som är inringade i grönt har gemensamt att B = 1, C= 0 samt D = 0. De ettor som är inringade i blått har gemensamt att A = 1, B = 1 samt D = 0.

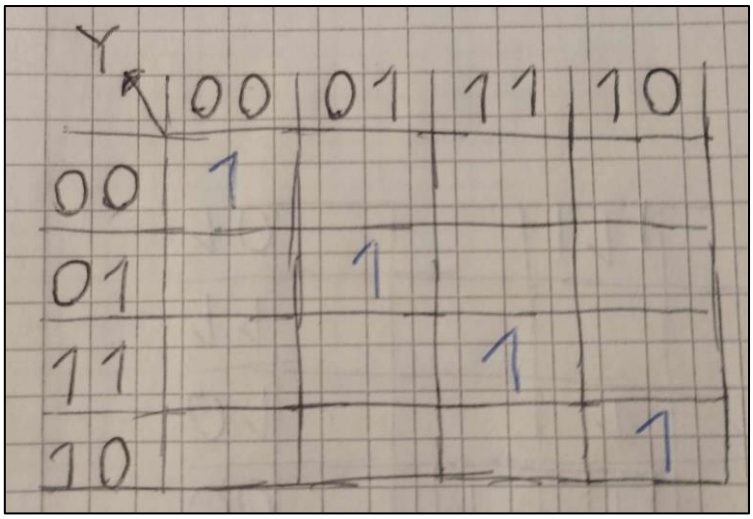
Därmed gället att X = 1 om AC = 10, BCD = 100 eller ABD = 110, vilket på boolesk algebra skrivs enligt nedan:

$$X = AC' + BC'D' + ABD'$$

Genom att bryta ut BD’ ut de två sista faktorerna kan ovanstående ekvation skrivas om till följande:

$$X = AC' + BD'(A + C')$$

Vi ritar sedan Karnaugh-diagram för utsignal Y:



Figur 2: Karnaugh-diagram för komparatorns utsignal Y.

Vi placerar insignaler AB i y-led samt insignaler CD i x-led. Vi placerar AB samt CD i 2-bitars Grey-kod, alltså i ordningsföljden 00, 01, 11, 10, så att samtliga celler har en bit gemensam med samtliga intilliggande celler, inklusive ytterkanterna.

Vi lägger till ettor i de celler där Y = 1. I sanningstabellen ser vi att Y = 1 för kombinationer ABCD = 0000, 0101, 1010 samt 1111. Återigen struntar vi att skriva ut nollor i övriga celler, då vi enbart är intresserade av ettorna.

Vi noterar i Karnaugh-diagrammet ovan att vi inte har några ettor som är gemensamma. Vi löser därmed ekvationen algebraiskt.

I sanningstabellen ser vi att Y = 1 för kombinationer ABCD = 0000, 0101, 1010 samt 1111, vilket på boolesk algebra skrivs enligt nedan:

$$Y = A'B'C'D' + A'BC'D + AB'CD' + ABCD$$

Ovanstående ekvation kan förenklas till

$$Y = A'C'(B'D' + BD) + AC(B'D' + BD),$$

där

$$B'D' + BD = (B\hat{}D)'$$

Därmed gäller att

$$Y = A'C'(B\hat{}D)' + AC(B\hat{}D)'$$

Genom att bryta ut (B^D)’ får vi att

$$Y = (B\hat{}D)'(A'C' + AC),$$

där

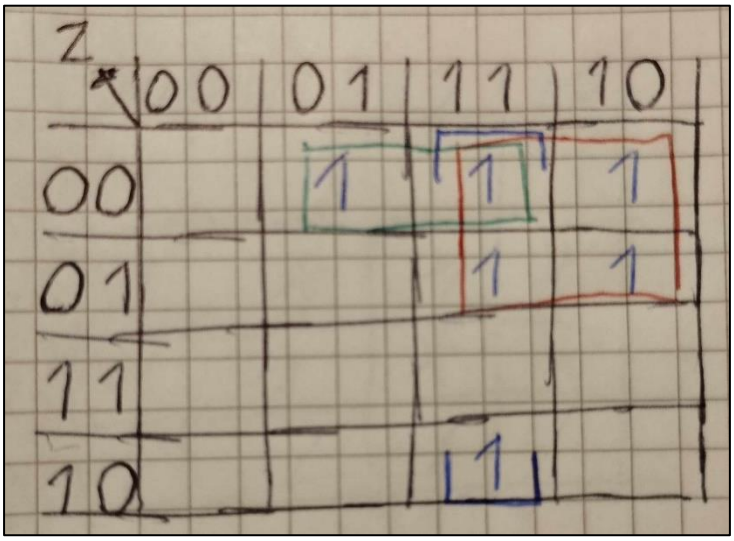
$$A'C' + AC = (A\hat{}C)'$$

Därmed gäller att

$$Y = (A\hat{}C)' * (B\hat{}D)',$$

vilket indikerar att Y = 1 om A = C samtidigt som B = D.

Vi ritar sedan Karnaugh-diagram för utsignal Z:



Figur 3: Karnaugh-diagram för komparatorns utsignal Z.

Vi placerar insignaler AB i y-led samt insignaler CD i x-led. Vi placerar AB samt CD i 2-bitars Grey-kod, alltså i ordningsföljden 00, 01, 11, 10, så att samtliga celler har en bit gemensam med samtliga intilliggande celler, inklusive ytterkanterna.

Vi lägger till ettor i de celler där Z = 1. I sanningstabellen ser vi att Z = 1 för kombinationer ABCD = 0001, 0010, 0011, 0110, 0111 samt 1011. Som tidigare skriver vi inte ut nollor i övriga celler, då vi enbart är intresserade av ettorna.

Vi noterar i Karnaugh-diagrammet ovan att vi får tre block innehållande ettor. Vi ringar in dessa block med röd, grön respektive blå färg. De fyra ettor som är inringade i rött har gemensamt att A = 0 samt C = 1. De två ettor som är inringade i grönt har gemensamt att A = 0, B = 0 och D = 1. De ettor som är inringade i blått har gemensamt att B = 0, C = 1 samt D = 1.

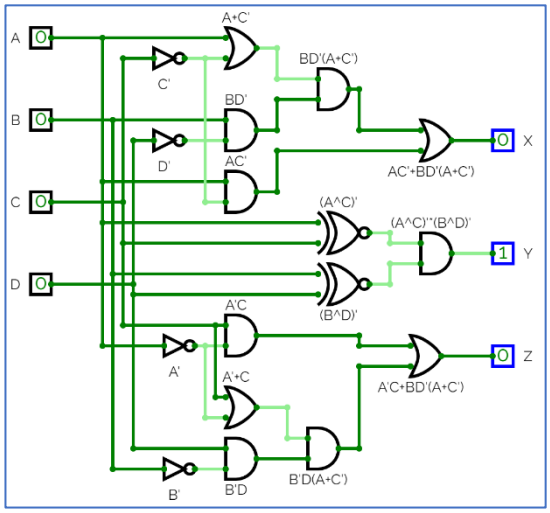
Därmed gället att Z = 1 om AC = 01, ABD = 001 eller BCD = 011, vilket på boolesk algebra skrivs enligt nedan:

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

Genom att bryta ut B'D ut de två sista faktorerna kan ovanstående ekvation skrivas om till följande:

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

Grindnätet kan därmed realiseras såsom visas nedan:



Figur 4: Grindnät för realisering av komparatorn.

Komparatorn kan realiseras i VHDL via följande modul döpt *comparator_4bit,* där en case-sats används för att beskriva funktionen i stället för att vi använder de framtagna ekvationerna. Därmed blir det kompilatorns uppgift att hitta de logiska ekvationerna och realisera grindnätet. Det är inte säkert att grindnätet blir samma som vad vi gjorde för hand. Vid konstruktion av större logiska grindnät är det fördelaktigt att låta kompilatorn sköta realiseringen av grindnätet, så kan vi fokusera på konstruktionens beteende och funktion i stället.

```
--------------------------------------------------------------------------------
-- @brief Implementation of a 4-bit comparator consisting of inputs abcd and
--        outputs xyzv.
--
--        The truth table of the comparator is shown below:
--
--        | abcd | xyz |
--        | 0000 | 010 |
--        | 0001 | 001 |
--        | 0010 | 001 |
--        | 0011 | 001 |
--        | 0100 | 100 |
--        | 0101 | 010 |
--        | 0110 | 001 |
--        | 0111 | 001 |
--        | 1000 | 100 |
--        | 1001 | 100 |
--        | 1010 | 010 |
--        | 1011 | 001 |
--        | 1100 | 100 |
--        | 1101 | 100 |
--        | 1110 | 100 |
--        | 1111 | 010 |
--
-- @param abcd[in] Comparator input.
-- @param xyz[out] Comparator output.
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity comparator_4bit is
    port(abcd: in std_logic_vector(3 downto 0);
         xyz : out std_logic_vector(2 downto 0));
end entity;

architecture behaviour of comparator_4bit is
begin
    output_process: process(abcd) is
    begin
        case (abcd) is
            when "0000" => xyz <= "010";
            when "0001" => xyz <= "001";
            when "0010" => xyz <= "001";
            when "0011" => xyz <= "001";
            when "0100" => xyz <= "100";
            when "0101" => xyz <= "010";
            when "0110" => xyz <= "001";
            when "0111" => xyz <= "001";
            when "1000" => xyz <= "100";
            when "1001" => xyz <= "100";
            when "1010" => xyz <= "010";
            when "1011" => xyz <= "001";
            when "1100" => xyz <= "100";
            when "1101" => xyz <= "100";
            when "1110" => xyz <= "100";
            when "1111" => xyz <= "010";
            when others => xyz <= "000";
        end case;
    end process;
end architecture;
```

Konstruktionen kan valideras via simulering i ModelSim via nedanstående testbänk *comparator_4bit_tb.*
Varje kombination 0000 – 1111 av insignaler *abcd* testas under 10 ns var. Simuleringstiden uppgår därmed till 160 ns:

```
--------------------------------------------------------------------------------
-- @brief Test bench for top module comparator_4bit. All 16 combinations of
--        inputs abcd = 0000 - 1111 are tested during 10 ns each. Hence the
--        total simulation time is 160 ns.
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity comparator_4bit_tb is
end entity;

architecture behaviour of comparator_4bit_tb is

--------------------------------------------------------------------------------
-- @brief Signals used to simulate the top module.
--------------------------------------------------------------------------------
signal abcd: std_logic_vector(3 downto 0) := (others => '0');
signal xyz : std_logic_vector(2 downto 0) := (others => '0');
begin

    --------------------------------------------------------------------------------
    -- @brief Creates an instance of the top module and connects its ports to
    --        signals abcd and xyzv for simulation.
    --------------------------------------------------------------------------------
    sim_instance: entity work.comparator_4bit
    port map(abcd, xyz);

    --------------------------------------------------------------------------------
    -- @brief Tests the top module with all 16 combinations 0000 - 1111 of inputs
    --        abcd during 10 ns each. The process is halted after every single
    --        combination has been tested, else it would start all over.
    --------------------------------------------------------------------------------
    sim_process: process is
    begin
        for i in 0 to 15 loop
            abcd <= std_logic_vector(to_unsigned(i, 4));
            wait for 10 ns;
        end loop;
        wait;
    end process;

end architecture;
```