Lösningsförslag övningsuppgifter 2025-03-14

1. Vi ska realisera en 2-bitars prioritersavkodare, som under lektion ska användas i en AD-omvandlare konstruerad i LTspice. Prioritetsavkodaren ska ha fyra insignaler ABCD samt två utsignaler XY. Prioritetsavkodaren bryr sig bara om den mest signifikanta insignalen ABCD. Därmed gäller följande, där X betyder don't care, alltså att värdet inte spelar någon roll:

$$ABCD = \begin{cases} 1XXX => XY = 11 \\ 01XX => XY = 10 \\ 001X => XY = 01 \\ 000X => XY = 00 \end{cases}$$

Sanningstabellen för prioritetsavkodaren visas nedan:

ABCD	XY
0000	00
0001	00
0010	01
0011	01
0100	10
0101	10
0110	10
0111	10
1000	11
1001	11
1010	11
1011	11
1100	11
1101	11
1110	11
1111	11

Tabell 1: Sanningstabell för 2-bitars prioritetsavkodare

Tips: Eftersom prioritetsavkodaren enbart bryr sig om den mest signifikanta höga insignalen kan sanningstabellen ovan förenklas via don't care-värden (symboliserade via X), såsom visas nedan:

ABCD	XY
0000	00
001X	01
01XX	10
1XXX	11

Tabell 2: Förenklad sanningstabell för 2-bitars prioritetsavkodare.

a) Ta fram minimerade ekvationer för utsignaler X och Y, antingen matematiskt eller via Karnaugh-diagram. Använd någon av sanningstabellerna ovan.

b) Realisera motsvarande grindnät för hand.

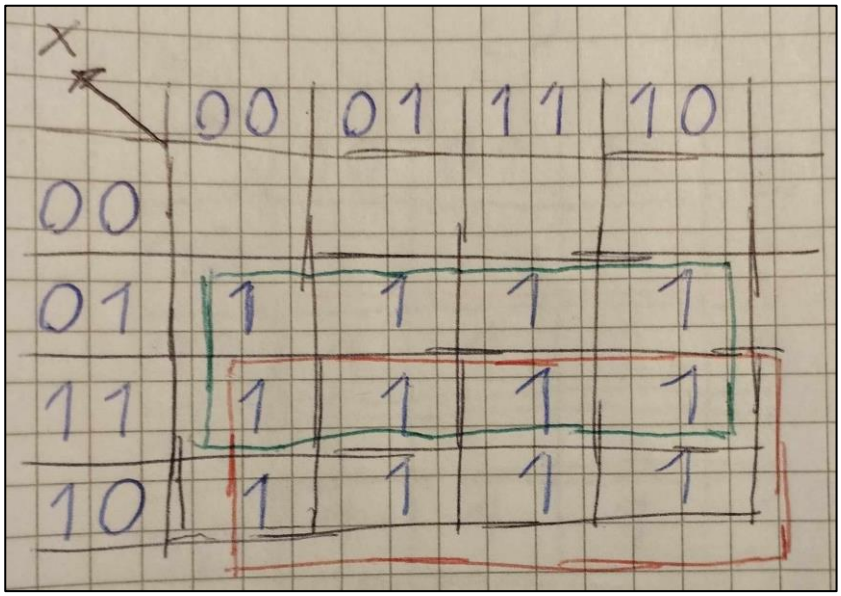
c) Simulera grindnätet via CircuitVerse, validera att konstruktionen fungerar som tänkt.

d) Skapa ett nytt projekt döpt *priority_encoder* i Quartus och realisera konstruktionen genom att implementera de framtagna ekvationerna direkt. Välj FPGA-kort Terasic DE0 (enhet 5CEBA4F23C7). Toppmodulen ska ha samma namn som projektet. Kompilera och inspektera den syntetiserade kretsen. Matchar den motsvarande krets som realiserades för hand?

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

Lösning

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



Figur 1: Karnaugh-diagram för utsignal X i uppgift 1.

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, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110 samt 1111. 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 rader innehållande fyra ettor vardera. Vi kan tänka att vi har ett block bestående av en rad (AB = 01) samt ett block bestående av två rader (AB = 11 & AB = 10). För att förenkla så mycket som möjligt vill vi dock göra blocken så stora som möjligt, förutsatt att blocket innehåller 2, 4, 8 eller 16 ettor. Därmed tänker vi i stället att vi har två block bestående av två rader var; det första blocket sträcker sig över AB = 01 – 11, samtidigt som det andra blocket sträcker sig över AB = 11 – 10. Den mittersta raden AB = 11 är därmed gemensam och får lite överlapp.

Vi ringar in dessa block med röd respektive grön färg. De åtta ettor som är inringade i rött har gemensamt att A = 1. De åtta ettor som är inringade i grönt har gemensamt att B = 1.

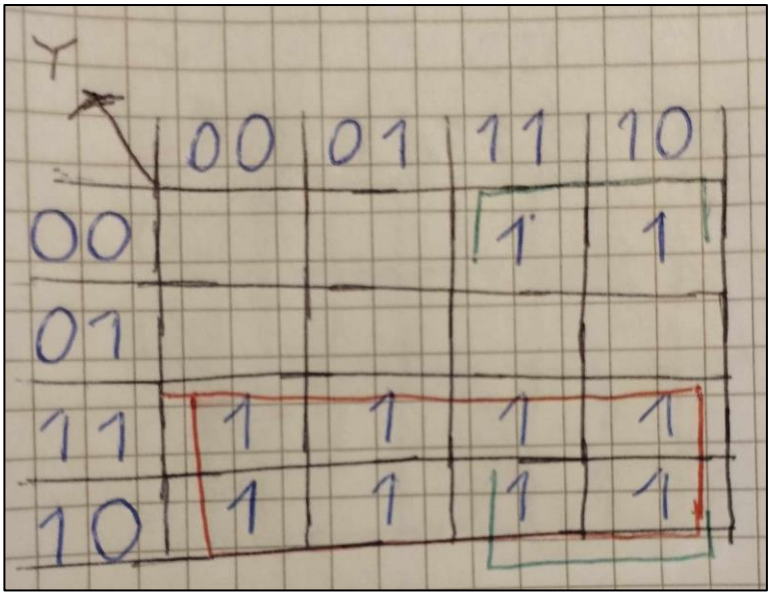
Därmed gället att X = 1 om A = 1 eller B = 1, vilket på boolesk algebra skrivs enligt nedan:

$$X = A + B$$

Grindnätet för utsignal X kan därmed realiseras via en OR-grind med A och B som insignaler.

OBS! Vänd blad!

Vi ritar sedan Karnaugh-diagram för utsignal Y:



Figur 2: Karnaugh-diagram för utsignal Y i uppgift 1.

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 = 0010, 0011, 1000, 1001, 1010, 1111, 1100, 1101, 1110 samt 1111. Återigen struntar vi att skriva ut nollor i övriga celler, då vi enbart är intresserade av ettorna.

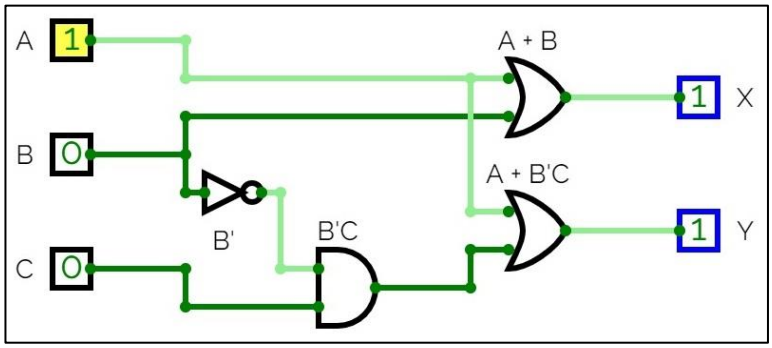
Vi noterar i Karnaugh-diagrammet ovan att vi ett block bestående av åtta ettor då AB = 10 samt 11. Vi får också ett block bestående av fyra ettor i den högra ytterkanten (även ytterkanterna har gemensamheter). Vi ringar in dessa block med röd respektive grön färg. De åtta ettor som är inringade i rött har gemensamt att A = 1. De fyra ettor som är inringade i grönt har gemensamt att B = 0 och C = 1.

Därmed gället att Y = 1 om A = 1 eller BC = 01, vilket på boolesk algebra skrivs enligt nedan:

$$Y = A + B'C$$

Grindnätet för utsignal Y kan därmed realiseras via en OR-grind, en AND-grind samt en NOT-grind.

Grindnätet för prioritetsavkodaren kan därmed realiseras såsom visas nedan:



Figur 3: Grindnät för realisering av prioritetsavkodaren.

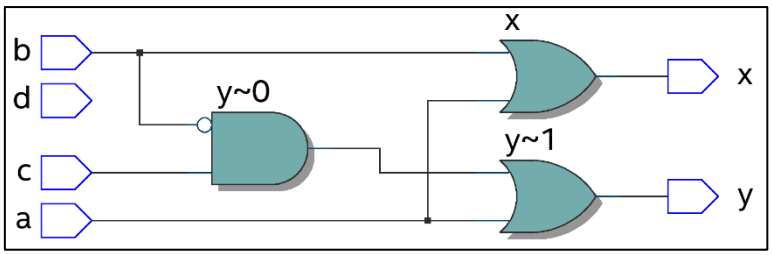
Motsvarande grindnät kan realiseras i VHDL via följande modul döpt *priority_encoder:*

```
--------------------------------------------------------------------------------
-- @brief Implementation of a 2-bit priority encoder consisting of four inputs
--        abcd and two outputs xy. The 4-bit input is encoded to yield a 2-bit
--        output based entirely on the most significant input bit, i.e. the
--        most the most significant input bit abcd is prioritized.
--
--        The truth table of the priority encoder is shown below.
--        Note that X means don't care, i.e. the value doesn't matter:
--
--        | abcd | xy |
--        | 1XXX | 11 |
--        | 01xx | 10 |
--        | 001X | 01 |
--        | 000X | 00 |
--
-- @note x = 1 when abcd = 1XXX and abcd = 01XX, hence
--       X = A + A'B = X = A + B (valid rule for boolean algebra).
-- @note y = 1 when abcd = 1XXX and abcd = 001X, hence
--       Y = A + A'B'C = A + A'(B'C) = A + B'C (same rule used as above).
--
-- @param a[in]  Most significant input bit.
-- @param b[in]  Second most significant input bit.
-- @param c[in]  Third most signifiant input bit.
-- @param d[in]  Least significant input bit.
-- @param x[out] Most significant output bit.
-- @param y[out] Least significant output bit.
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity priority_encoder is
    port(a, b, c, d: in std_logic;
         x, y      : out std_logic);
end entity;

architecture behaviour of priority_encoder is
begin
    x <= a or b;
    y <= a or ((not b) and c);
end architecture;
```

Efter kompilering av ovanstående VHDL-kod har följande krets syntetiserats för prioritetsavkodaren:



Figur 4: Syntetiserad krets för prioritetsavkodaren.

Notera att den syntetiserade kretsen är identisk med kretsen som konstruerades i CircuitVerse tidigare. Detta beror på att vi mata in ekvationerna för utsignaler x och y direkt. Ifall vi hade genomfört implementationen via if- och else-satser i stället (eller case-satsen, som motsvarar switch-satsen i C) hade kompilatorn möjligtvis realiserat konstruktionen annorlunda, exempelvis via så kallade multiplexers.