Lösningsförslag övningsuppgifter 2025-03-07

1. Förklara följande nyckelord i VHDL:

a) *entity*
b) *architecture*
c) *std_logic*
d) *process*
e) *signal*

Lösning

a) *entity* utgör utsidan av en modul, där portarna deklareras. Som exempel kan en OR-grind med inportar a och b samt utport x deklareras via en entitet döpt *or_gate* såsom visas nedan:

```
entity or_gate is
    port(a, b: in std_logic;
         x   : out std_logic);
end entity;
```

b) *architecture* utgör insidan av en modul, där modulens beteende/funktionalitet beskrivs. Som exempel, arkitekturen för entiteten *or_gate* ovan kan definieras såsom visas nedan för att realisera funktionaliteten av en OR-grind:

```
architecture behaviour of or_gate is
begin
    x <= a or b;
end architecture;
```

c) *std_logic* utgör en datatyp för signaler som ska kunna anta logiska värden ’0’ och ’1’ samt övriga värden som behövs för att realisera digitala signaler i praktiken, såsom högohmig/tri-state (’Z’), don’t care (’-’) med mera. Utmärkt för signaler och variabler som ska tilldelas en eller flera bitar (för fler bitar används datatypen *std_logic_vector,* dvs. en vektor med bitar). I VHDL måste datatypen *std_logic* inkluderas från ett package döpt *std_logic_1164* i biblioteket *IEEE,* vilket åstadkommes via följande instruktioner:

```
library ieee;
use ieee.std_logic_1164.all;
```

d) En *process* utgör ett sekventiellt block, vilket innebär att innehållet exekverar sekventiellt (uppifrån och ned) en instruktion i taget, så som sker vid mjukvaruprogrammering i C, C++, Python eller andra språk. Genom att använda flera processer kan saker ske parallellt för ökad prestanda, likt flertrådade mjukvaruprogram.

Funktionaliteten för OR-grinden ovan hade kunnat realiseras via en process döpt OR_PROCESS såsom visas nedan. Processen i fråga exekverar vid förändring av någon av insignaler a och b, vilket implementeras genom att deklarera dessa portar i den så kallade känslighetslistan (innehållet i parentesen efter nyckelordet *or_gate).* Via en if-else sats sätts utsignal x till hög om antingen a eller b är höga, annars sätts x till 0:

```
architecture behaviour of or_gate is
begin
    OR_PROCESS: process(a, b) is
    begin
        if (a = '1' or b = '1') then
            x <= '1';
        else
            x <= '0';
        end if;
    end process;
end architecture;
```

e) Nyckelordet *signal* används för interna signaler inom en arkitektur, tänk ledningar mellan logiska grindar. Signaler kan tilldelas ett värde kontinuerligt eller via en process. Signalens värde kan läsas i hela arkitekturen, men tilldelning kan bara ske från en källa. Signaler kan därmed användas likt filglobala variabler i ett programspråk; i detta fall sträcker sig dock synligheten inte till hela filen, utan till den aktuella arkitekturen. Alla signaler deklareras i den deklarativa delen av arkitekturen, alltså direkt ovanför nyckelordet *begin.*

Som exempel på användning av en signal i ett system med insignaler a, b, c och d och utsignal x som ska uppfylla den logiska funktionen x = a + b'cd kan en signal döpt y = b'cd implementeras enligt nedan, i detta fall primärt för att göra koden mer läsbar:

```
library ieee;
use ieee.std_logic_1164.all;

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

architecture behaviour of signal_example is
signal y: std_logic;
begin
    y <= (not b) and c and d;
    x <= a or y;
end architecture;
```

Utan att använda signalen Y hade koden sett ut såsom visas nedan:

```
library ieee;
use ieee.std_logic_1164.all;

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

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

OBS! Vänd blad!

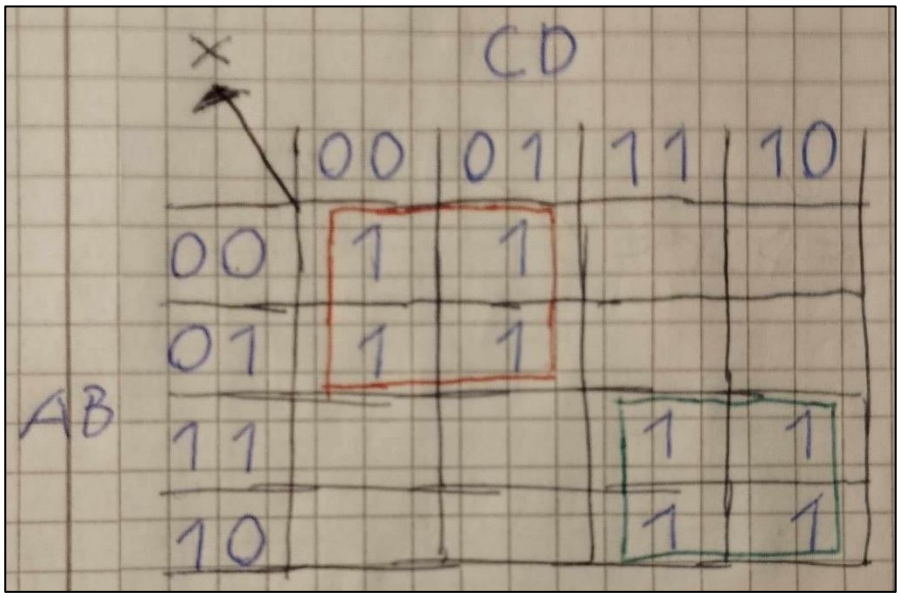
2. Härled en minimerad logisk ekvation ur sanningstabellen nedan via ett Karnaugh-diagram och realisera grindnätet.

ABCD	X
0000	1
0001	1
0010	0
0011	0
0100	1
0101	1
0110	0
0111	0
1000	0
1001	0
1010	1
1011	1
1100	0
1101	0
1110	1
1111	1

Sanningstabell 1: Sanningstabell för uppgift 2.

Lösning

Vi ritar om sanningstabellen ovan till nedanstående Karnaugh-diagram:



Figur 1: Karnaugh-diagram för uppgift 2.

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 = 0000, 0001, 0100, 0101, 1010, 1011, 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 två block innehållande fyra ettor vardera. Vi ringar in dessa med röd respektive grön färg. De fyra ettor som är inringade i rött har gemensamt att A = 0 samt C = 0. De fyra ettor som är inringade i grönt har gemensamt att A = 1 samt C = 1.

Därmed gället att X = 1 om AC = 00 eller AC = 11, vilket på boolesk algebra skrivs enligt nedan:

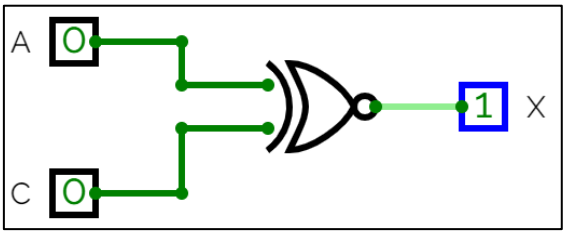
$$X = A'C' + AC$$

Notera att detta är ett XNOR-mönster; A och C måste ha samma värde (antingen 0 eller 1) för att utsignal X ska bli 1. Därmed gäller att

$$X = (A \hat{} C)'$$

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

Grindnätet kan därmed realiseras såsom visas nedan:



Figur 2: Realisering av grindnät för uppgift 2.

OBS! Vänd blad!

3. Realisera en implementering av ADAS-systemet i VHDL. Systemet ska kunna användas bromsa ett fordon i två fall:

- Föraren bromsar.
- ADAS-systemet indikerar att ett föremål framför fordonet närmar sig (och ADAS-systemet fungerar som det ska).

Systemet ska bestå utav insignaler *driver_break, camera, radar* och *adas_error* samt utsignal *vehicle_break:*

- ***driver_break*** blir ettställd när föraren trycker ned bromspedalen. Bilen ska då omedelbart bromsa.
- ***camera*** blir ettställd när ett annat fordon ligger mindre än 100 meter framför bilen.
- ***radar*** blir ettställd när ett annat fordon närmar sig bilen.
- ***adas_error*** är ettställd när ADAS-systemet inte fungerar som det ska. Om detta sker ska signalerna *camera* samt *radar* ignoreras.
- ***vecicle_break*** ska ettställas för att bromsa bilen.

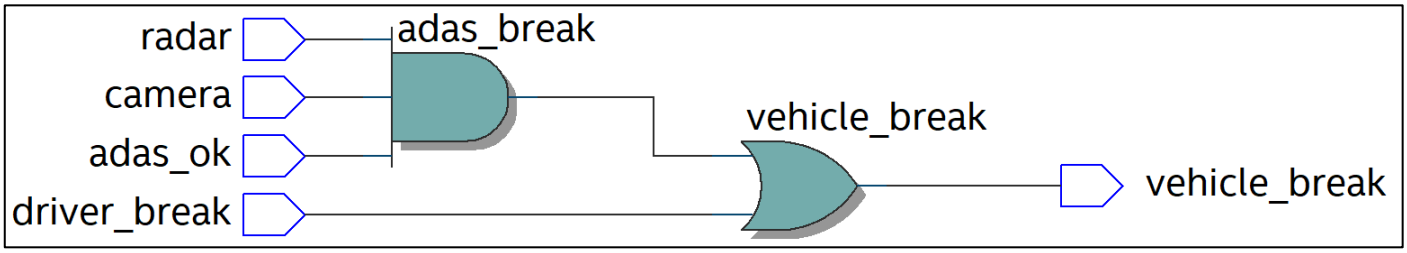
Lösning

```
--------------------------------------------------------------------------------
-- @brief Implementation of ADAS (Advanced Driver Assistance System) in VHDL.
--
-- @param[in]  driver_break  Signal from break pedal.
-- @param[in]  camera        Indicates object in front of the vehicle.
-- @param[in]  radar         Indicates object approaching vehicle.
-- @param[in]  adas_ok       Indicates that ADAS is working correctly.
-- @param[out] vehicle_break Signal to enable vehicle break for reducing speed.
--
-- @note The vehicle break signal is set if the driver pushes the break pedal
--       or id the camera senses an object in front of the car while the radar
--       indicates that the object is approaching and ADAS is working correctly.
--       If ADAS isn't working correctly, the camera and radar signals are
--       ignored.
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity adas is
    port(driver_break, camera, radar, adas_ok: in std_logic;
         vehicle_break                       : out std_logic);
end entity;

architecture behaviour of adas is
signal adas_break: std_logic := '0';
begin
    adas_break    <= adas_ok and camera and radar;
    vehicle_break <= driver_break or adas_break;
end architecture;
```

Det syntetiserade grindnätet visas nedan:



Figur 3: Det syntetiserade grindnätet.