Lösningsförslag övningsuppgifter 2023-03-29

1. Förklara följande nyckelord i VHDL:

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

Svar

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).* Vid 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;
```

2. Realisera grindnätet för nedanstående VHDL-modul, där en lysdiod tänds vid udda antal nedtryckta tryckknappar. Sätt *button_n[2:0]* = ABC samt *led* = X i grindnätet.

```
--------------------------------------------------------------------------------
-- gate_example.vhd: Module consisting of three push buttons and one LED.
--                   The LED is enabled at odd number of pressed push buttons,
--                   else it's disabled.
--
--                   Inputs:
--                      - button_n[2:0]: Inverting push buttons (active low).
--                   Outputs:
--                      - led          : LED enabled at odd number of pressed
--                                       push buttons.
--
--                   Hardware implemented for FPGA card Terasic DE0.
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity gate_example is
   port(button_n: in std_logic_vector(2 downto 0);
        led     : out std_logic);
end entity;
```

```
architecture behaviour of gate_example is
begin

   ---------------------------------------------------------------------------
   -- LED_PROCESS: Counts the number of pressed push buttons and enables the LED
   --              when an odd number of buttons are pressed, else the LED is
   --              disabled.
   ---------------------------------------------------------------------------
   LED_PROCESS: process(button_n) is
   variable num: natural range 0 to 2;
   begin
      num := 0;
      for i in 0 to 2 loop
         if (button_n(i) = '0') then
            num := num + 1;
         end if;
      end loop;
      if (num = 1 or num = 3) then
         led <= '1';
      else
         led <= '0';
      end if;
   end process;

end architecture;
```

Lösning

Först tar vi fram en sanningstabell för systemet, där ABC = button_n[2:0] och X = led. Notera att 0 innebär nedtryckt knapp, så vid udda antal nedtryckta tryckknappar förekommer udda antal nollor i insignaler ABC och då ska utsignal X ettställas för att tända lysdioden:

ABC	X
000	1
001	0
010	0
011	1
100	0
101	1
110	1
111	0

Tabell 1 – Sanningstabell för uppgift 2.

Ur ovanstående sanningstabell ser vi att utsignal X ska ettställas i följande fyra fall:

- A = 0 samtidigt som B = 0 och C = 0
- A = 0 samtidigt som B = 1 och C = 1
- A = 1 samtidigt som B = 0 och C = 1
- A = 1 samtidigt som B = 1 och C = 0

Ovanstående fyra fall kan skrivas om via boolesk algebra till följande logiska funktion:

$$X = A'B'C' + A'BC + AB'C + ABC'$$

Ovanstående ekvation kan förenklas genom att bryta ut A' respektive A:

$$X = A'(B'C' + BC) + A(B'C + BC')$$

Eftersom

$$B \wedge C = B'C + BC'$$

samt

$$(B \wedge C)' = B'C + BC'$$

kan ovanstående ekvation förenklas till följande:

$$X = A'(B \wedge C)' + A(B \wedge C)$$

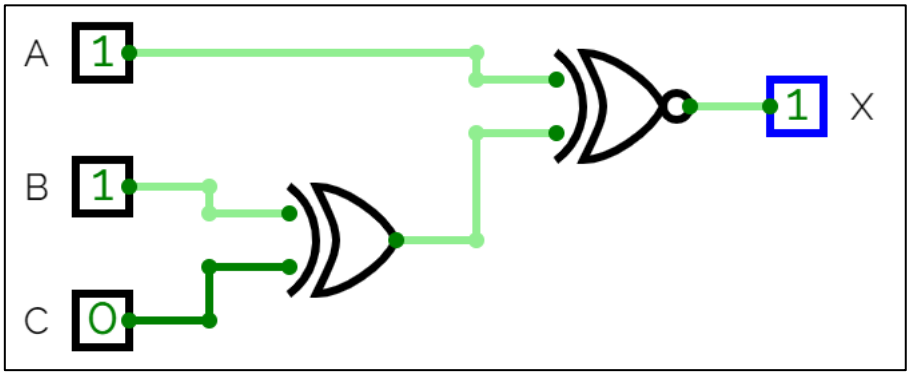
Eftersom

$$\big(A \wedge (B \wedge C)\big)' = A'(B \wedge C)' + A(B \wedge C),$$

gäller att

$$X = \big(A \wedge (B \wedge C)\big)'$$

Därmed kan grindnätet realiseras via en XOR-grind med B och C som insignaler samt en XNOR-grind med A samt utsignalen från föregående XOR-grind (B^C) som insignaler, såsom visas i figuren nedan (simulerat i CircuitVerse):



Figur 1: Grindnät för realisering av grindnätet i uppgift 2.

Modulen *gate_example* hade därmed kunnat förenklas till följande:

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

entity gate_example is
   port(button_n: in std_logic_vector(2 downto 0);
        led     : out std_logic);
end entity;

architecture behaviour of gate_example is
begin
   led <= button_n(2) xnor (button_n(1) xor button_n(0));
end architecture;
```