**Projektovanje pomoću računara – Termin 5**

**Zadatak 1**. Napisati entitet i arhitekturu za D Flip-flop sa direktnim resetom i presetom kao sto je dato u tabeli:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| D | S | R | CLK | Q |
| X | 0 | 0 | 0 | Q' |
| X | 0 | 0 | 1 | Q' |
| D | 0 | 0 | \_↑¯ | D |
| X | 1 | 0 | X | 1 |
| X | 0 | 1 | X | 0 |

Rešenje:

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

**entity** D\_FlipFlop **is**

 **port**(

 D: **in** STD\_LOGIC;

 R: **in** STD\_LOGIC;

 S: **in** STD\_LOGIC;

 CLK: **in** STD\_LOGIC;

 Q: **out** STD\_LOGIC);

**end** D\_FlipFlop;

**architecture** D\_FlipFlop **of** D\_FlipFlop **is**

**begin**

DFF: **process** (CLK,R,S)

**begin**

 **if** R=’1’ **then** Q<=’0’; --direktno resetovanje (izlazu Q se dodeljuje 0)

**elsif** S=’1’ **then** Q<=’1’; --direktno setovanje ( izlazu Q se dodeljuje 1)

**elsif** (clk**’event** and clk=’1’**) then** Q<=D; -- ukoliko je nastupila rastuća ivica klok signala

--izlazu Q dodeli onu vrednost koja se trenutno nalazi na D liniji.

**end if**;

**end process** DFF;

**end** D\_FlipFlop;

NAPOMENA: Da ponovimo ‘event naredbu. Ispravna sintaksa je ime\_signala’event i ova naredba vraca true vrednost ukoliko je signal koji navodimo ispred ‘event naredbe promenio svoju vrednost. Zbog toga da bi ispitali rastucu ivicu signala neophodno da pored provere da li je signal promenio vrednost ispitamo i da li je naradno stanje signala jednaka HIGH(1). Ukoliko su oba uslova ispunjena nastala je rastuca ivica signala.

Realizacija potpunog sabirača

**Zadatak 2**. Napisati VHDL kod za jednobitni potpuni sabirač čija je tablica istinitosti data u tabeli 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| cin | a | b | sum | cout |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Tabela 1. Tablica istinitosti potpunog sabirača

REŠENJE:

Teorijski uvod: Potpuni sabirač ima tri ulaza (a – prvi operand, b – drugi operand i cin – ulazni bit prenosa) i dva izlaza (sum – zbir signala na ulazima, cout – izlazni bit prenosa). Pošto je rečeno da se radi o jednobitnom potpunom sabiraču svi signali će biti tipa STD\_LOGIC.

KOD:

**library** IEEE;

**use** IEEE.STD\_LOGIC\_1164.all;

**entity** PotpuniSabirac **is**

 **port** (

 cin: **in** STD\_LOGIC;

 a**: in** STD\_LOGIC;

 b: **in** STD\_LOGIC;

 sum: **out** STD\_LOGIC;

 cout: **out** STD\_LOGIC

 );

**end** PotpuniSabirač;

**architecture** PotpuniSabirac\_Arch **of** PotpuniSabirač **is**

**begin**

 sum<=((a **XOR** b) **XOR** cin);

 cout<=(a **AND** b) **OR** (a **AND** cin) **OR** (b **AND** cin);

end PotpuniSabirac\_Arch;

Zadatak 3. Realizovati četvorobitni potpuni sabirač pri čemu se koristi tabela 1.

REŠENJE:

Pošto se radi o četvorobitnom potpunom sabiraču, imena linija će biti ista, samo neće biti tipa Standard logic već će biti četvorobitne magistrale. Prattično imamo 4 paralelno povezana jednobitn potpuna sabirača, kao što je predstavjeno na slici 1.

 Slika 1. 4-bitni potpuni sabirač

**library** IEEE;

**use** IEEE.STD\_LOGIC\_1164.all;

**entity** Potpuni\_4bitni\_Sabirac **is**

 **port** (

 cin: **in** STD\_LOGIC;

 a**: in** STD\_LOGIC\_VECTOR (3 **downto** 0);

 b: **in** STD\_LOGIC\_VECTOR (3 **downto** 0);

 sum: **out** STD\_LOGIC\_VECTOR (3 **downto** 0);

 cout: **out** STD\_LOGIC

 );

**end** Potpuni\_4bitni\_Sabirac;

**architecture** Potpuni\_4bitni\_Sabirac\_Arch **of** Potpuni\_4bitni\_Sabirac **is**

**begin**

Zbir: **process** ( a , b , cin) --Proces se izvrsava kad nastupi promena nekog ulaznog signala

**variable** Prenos: STD\_LOGIC\_VECTOR(4 **downto** 0); --Pomoćni signal za čuvanje bita

 --prenosa

**variable** LokalniZbir: STD\_LOGIC\_VECTOR (3 **downto** 0); --Pomoćni signal za čuvanje ----------------------------------------------------------------------------------lokalnog zbira

**begin**

Prenos (0) := cin;

**for** i **in** 0 **to** **3 loop --**for petlja da izvrtimo sva četri bita počev od LSB

 LokalniZbir(i) := ( a ( i ) **XOR** b ( i ) ) **XOR** Prenos ( i );

 Prenos ( i + 1 ) := ( a ( i ) **AND** b ( i ) ) **OR** ( Prenos ( i ) **AND** ( a ( i ) **OR** b ( i ) ) );

**end loop**;

sum <= LokalniZbir; --Stanje lokalnog zbira je vrednost sume na izlazu

cout<= Prenos(4); --najviši bit prenosa je ustvari cout.

**end process** Zbir;

**end** Potpuni\_4bitni\_Sabirac\_Arch;