

5 VHDL ČASŤ I.

5.1 Úvod

Jazyk VHDL používame ako prostriedok na návrh, modelovanie, verifikáciu a simuláciu obvodov, ale aj na návrh vložených diagnostických prostriedkov a testov.

Hlavnou úlohou pri návrhu číslicových obvodov s vysokou integráciou je zvládnutie postupu návrhu tak, aby tento postup nemal nepriaznivý vplyv na vlastnosti celkového riešenia. Zapojiť stovky tisíc tranzistorov do obvodu sa s dostatočnou kvalitou a rýchlosťou môže podariť len vtedy, keď je popísaná štruktúrou zložitého zapojenia dostatočne jasne a jednoducho.

Elektronické číslicové systémy sú v súčasnosti konštruované pomocou obvodov VLSI (**V**ery **L**arge **S**cale **I**ntegration). Implementované sú ako obvody ASIC (**A**pplication **S**pecific **I**ntegrated **C**ircuits). Vo fáze prototypu sa uplatňujú okrem programovateľných logických obvodov PLD (**P**rogrammable **L**ogic **D**evelopments) hlavne programovateľné hradlové polia FPGA (**F**ield **P**rogrammable **G**ate **A**rrays).

Hlavným dôvodom vzniku jazyka pre popis obvodov HDL (**H**ardware **D**escription **L**anguage) bolo to, že jazyk pomáha návrhárovi zachovať prostriedky pre popis elektronických funkcií na všetkých úrovniach abstrakcie. Pri použití HDL sa tiež urýchli a zjednoduší návrhový proces.

Zo začiatku existovalo väčšie množstvo jazykov, ale len niekoľko najpoužívanejších sa stalo de facto priemyslovými štandardmi (napr. VHDL a Verilog). Uvedenie VHDL (**V**ery **H**igh **S**peed **I**ntegrated **C**ircuits- **V**HSIC HDL) ako doporučenie

IEEE spôsobilo, že v súčasnej ponuke je na trhu VHDL ponúkaný ako základná časť súborov programov pre návrh.

Vývoj jazyka VHDL bol daný v roku 1983 firmám IBM, Texas Instruments a Intermetrics a bol riešený v rámci úloh pre vývoj veľmi rýchlych integrovaných obvodov (VHSIC). V roku 1987 bol jazyk prijatý ako štandard IEEE pod číslom 1079-1987. Tento štandard bol a aj je naďalej vyvíjaný. Najdôležitejším dôvodom pre použitie VHDL je to, že šetrí čas a peniaze. Medzi ďalšie výhody patrí zvýšenie produktivity práce pracovníkov.

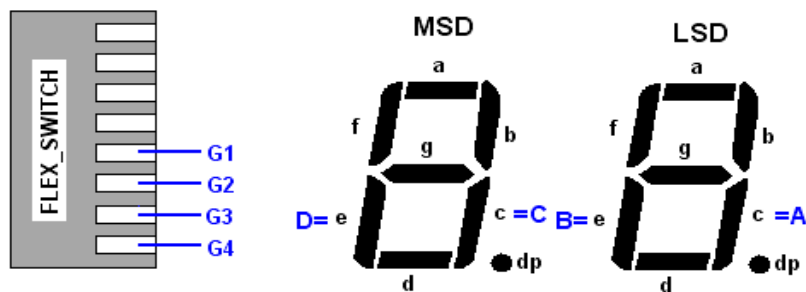
5.2 ZADANIE PRÍKLADU Č.1

Navrhnite prevodník z Grayovho kódu do kódu +3 ku BCD (0-9). Návrh realizujte pomocou jazyka VHDL, vo VHDL editore vývojového prostredia Quartus II.

5.3 RIEŠENIE

5.3.1 ROZBOR

Príklad budeme realizovať v textovom editore programu Quartus II pomocou VHDL kódu. Zostavíme pravdivostnú tabuľku prevodníka. Máme dve možnosti ďalšej realizácie príkladu. Jednou je vyhotoviť Karnaughove mapy, z nich napísať algebraické rovnice a na základe týchto rovníc realizovať VHDL kód. Týmto spôsobom dostaneme realizáciu príkladu totožnú s grafickým riešením. Druhou možnosťou je zostavenie VHDL kódu priamo z pravdivostnej tabuľky. Tento spôsob je jednoduchší, rýchlejší a kratší. Pre ilustráciu ukážeme oba prípady. Ako vstupy použijeme **FLEX_SWITCH** prepínače (obr.1). V hornej polohe reprezentujú logickú úroveň „1“ a v dolnej polohe reprezentujú logickú úroveň „0“. Výstupy budeme zobrazovať na sedem segmentovej LED, pričom použijeme segmenty **e**, **c** z MSD sedem segmentovky a segmenty **e**, **c** z LSD sedem segmentovky (obr.1). Pri návrhu ešte musíme brať do úvahy tú skutočnosť, že náš sedem segmentový displej reaguje na logickú úroveň 0.



Obr.1: Zobrazenie sedem segmentoviek MSD a LSD

Nepoužitú segmenty na displeji necháme v stave vysokej impedancie, alebo pripojíme na VCC, aby nám nesvietili. Všetky použité piny a ich popisy sú uvedené v tab.1.

Názov pinu	Typ pinu	Číslo pinu	Funkcia pinu
G1	Vstup	38	Prepínač (0 = poloha dole, 1 = poloha hore)
G2	Vstup	39	Prepínač (0 = poloha dole, 1 = poloha hore)
G3	Vstup	40	Prepínač (0 = poloha dole, 1 = poloha hore)
G4	Vstup	41	Prepínač (0 = poloha dole, 1 = poloha hore)
A	Výstup	19	Segment c LSD (0=LED ON-svieti, 1=LED OFF-nesvieti)
B	Výstup	21	Segment e LSD (0=LED ON-svieti, 1=LED OFF-nesvieti)
C	Výstup	8	Segment c MSD (0=LED ON-svieti, 1=LED OFF-nesvieti)
D	Výstup	11	Segment e MSD (0=LED ON-svieti, 1=LED OFF-nesvieti)
Other0	Výstup	17	Segment a LSD (0=LED ON-svieti, 1=LED OFF-nesvieti)
Other1	Výstup	18	Segment b LSD (0=LED ON-svieti, 1=LED OFF-nesvieti)
Other2	Výstup	20	Segment d LSD (0=LED ON-svieti, 1=LED OFF-nesvieti)
Other3	Výstup	23	Segment f LSD (0=LED ON-svieti, 1=LED OFF-nesvieti)
Other4	Výstup	24	Segment g LSD (0=LED ON-svieti, 1=LED OFF-nesvieti)
Other5	Výstup	25	Segment dp LSD (0=LED ON-svieti, 1=LED OFF-nesvieti)
Other6	Výstup	6	Segment a MSD (0=LED ON-svieti, 1=LED OFF-nesvieti)
Other7	Výstup	7	Segment b MSD (0=LED ON-svieti, 1=LED OFF-nesvieti)
Other8	Výstup	9	Segment d MSD (0=LED ON-svieti, 1=LED OFF-nesvieti)
Other9	Výstup	12	Segment f MSD (0=LED ON-svieti, 1=LED OFF-nesvieti)
Other10	Výstup	13	Segment g MSD (0=LED ON-svieti, 1=LED OFF-nesvieti)
Other11	Výstup	14	Segment dp MSD (0=LED ON-svieti, 1=LED OFF-nesvieti)

Tab.1: Tabuľka pinov

5.3.2 SYNTÉZA

Na začiatku návrhu si zostavíme pravdivostnú tabuľku (tab.2) nášho prevodníka, v ktorej si označíme vstupy a výstupy.

dekadicky	vstupy				výstupy			
	Grayov kód				Kód +3 ku BCD			
	G4	G3	G2	G1	D	C	B	A
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	1	0	1	0	1
3	0	0	1	0	0	1	1	0
4	0	1	1	0	0	1	1	1
5	0	1	1	1	1	0	0	0
6	0	1	0	1	1	0	0	1
7	0	1	0	0	1	0	1	0
8	1	1	0	0	1	0	1	1
9	1	1	0	1	1	1	0	0

Tab.2. Pravdivostná tabuľka

Ak budeme realizovať príklad tak, že bude totožný s grafickým riešením budeme postupovať nasledovne. Zostavíme si Karnaughove mapy a napíšeme si algebraické rovnice (kapitola.3). Na základe týchto rovníc realizujeme VHDL kód (pozri prílohu). Najprv zostavíme VHDL kód z pravdivostnej tabuľky (tab.2) nie z algebraických rovníc. Platí tabuľka pinov (tab.1) s tým rozdielom, že vstupné piny majú názov: *gray_code* s číslom od 1 po 4. V tabuľke č.1 a 2 sú označené ako *G1* až *G4*. Výstupné piny majú názov: *plus3_code* s číslami 1 až 4 a v tabuľkách č.1 a 2 sú označené ako *A*, *B*, *C* a *D*. Nepoužité segmenty majú názov: *ostatne_segmenty* s číslami od 0 po 11 a v tabuľke č.1 sú označené ako *Other* s číslami 0 až 11.

5.3.3 OTVORENIE NOVÉHO PROJEKTU

Nový projekt otvoríme voľbou **New Project Wizard** z **File menu**. Ako prvé sa objaví úvodné okno, kde kliknutím na tlačidlo **Next** sa otvorí prvé okno, v ktorom definujeme miesto uloženia, názov projektu a názov entity. Kliknutím na tlačidlo sa objaví štruktúra adresárov, z ktorých si vyberieme ten, do ktorého chceme náš projekt ukladať. Pre projekt vytvoríme napr. adresár: *prevodnik_Gray_vhdl_sof*. V ďalších riadkoch definujeme názov projektu a názov najvyššej úrovne entity, v tomto prípade zvolíme napr. *gray*. Tento údaj je dôležitý a to preto lebo vo VHDL kóde sa tento názov musí zhodovať z názvom entity. Našu voľbu potvrdíme tlačidlom **Next**.

V ďalšom okne môžeme priradiť súbory z iného projektu do tohto projektu, ak sú zhodné s tými, ktoré môžeme využiť pri návrhu. Nemusíme tú istú vec robiť dvakrát. Ak žiadne súbory nechceme priradiť, klikneme na tlačidlo **Next**. Pre náš projekt nepotrebujeme priradiť iné súbory.

Stlačením **Next** sa objaví okno – tretie okno voľby **New project Wizard**, v ktorom definujeme rodinu obvodov, v našom prípade rodinu FLEX10K.

V časti **Target device** zaškrtneme možnosť *Specific devices selected in 'Available devices' list*.

V spodnom okne vyberieme presný typ obvodu, s ktorým chceme pracovať. V našom prípade súčiastku EPF10K20RC240-4. Vo web verzii vývojového prostredia Quartus II ver.4.2 sa súčiastka EPF10K20RC240-4 nemusí nachádzať, preto zvolíme súčiastku EPF10K20RC240-3. Rozdiel je len v rýchlosti logiky súčiastok.

Kliknutím na tlačidlo **Next** sa otvorí štvrté okno voľby **New Project Wizard**, v ktorom môžeme nastaviť nástroje EDA. V tomto okne nebudeme nič nastavovať. Stlačíme **Next**.

V poslednom piatom okne, sú zobrazené všetky naše nastavenia. Ak s nimi súhlasíme potvrdíme ich tlačidlom **Finish**. Ak chceme niektoré údaje zmeniť, vrátime sa späť na nastavenia tlačidlom **Back**.

5.3.4 VYTVORENIE TEXTOVÉHO NÁVRHU PROJEKTU

Postupujeme nasledovne:

- Z **File Menu** vyberieme položku **New**
- V záložke **Design Files** zvolíme **VHDL File**
- Kliknutím na tlačidlo **OK** sa otvorí okno textového editora
- Z **File Menu** vyberieme položku **Save As**
- Vyberieme adresár *prevodnik_Gray_vhdl_sof*, do ktorého uložíme náš súbor s názvom *gray.vhd*. Pod riadkom, kde sa definuje názov ukladaného súboru, zaškrtneme voľbu **Add file to current project** (pridať súbor do aktuálneho projektu). Táto položka by mala byť implicitne zaškrtnutá.
- Kliknutím na tlačidlo **Save**, uložíme a zároveň vložíme súbor do projektu.

5.3.5 VYTVORENIE VHDL KÓDU

Ako prvé zdefinujeme knižnice štandardov ktoré budeme používať. Po zdefinovaní knižníc nasleduje deklarácia entity, v ktorej definujeme vstupné a výstupné porty. Ak si nepamätáme syntax jednotlivých funkcií a príkazov môžeme použiť preddefinované predlohy. Vyberieme ich voľbou **Insert Template** z **Edit menu**. V okne ktoré sa nám zobrazí si vyberieme potrebnú štruktúru a potvrdíme kliknutím na tlačidlo **OK**. Vybraná štruktúra nám bude automaticky umiestnená do textového súboru na pozíciu kurzora. Potom do štruktúry vložíme parametre, pre náš príklad.

Príklady syntaxov vložených pomocou voľby **Insert Template**:

Deklarácia entity

```
ENTITY __entity_name IS                                --deklaracia entity
  GENERIC
  (
    __parameter_name      : string := __default_value;
    __parameter_name      : integer:= __default_value
  );
  PORT
  (
    __input_name, __input_name      : IN   STD_LOGIC;
    __input_vector_name             : IN   STD_LOGIC_VECTOR(__high DOWNTO __low);
    __bidir_name, __bidir_name      : INOUT STD_LOGIC;
    __output_name, __output_name     : OUT  STD_LOGIC
  );
END __entity_name;
```

Príkaz IF

```
IF __expression THEN                                  --príkaz if
  __statement;
  __statement;
ELSIF __expression THEN
  __statement;
  __statement;
ELSE
  __statement;
  __statement;
END IF;
```

Príkaz CASE

```
CASE __expression IS                                 --príkaz Case
  WHEN __constant_value =>
    __statement;
    __statement;
  WHEN __constant_value =>
    __statement;
    __statement;
  WHEN OTHERS =>
    __statement;
    __statement;
END CASE;
```

V samotnom tele architektúry zadefinujeme príkazmi nami požadovaný proces podľa algebraických rovníc ktoré sme dostali z Karnaughových máp (rovnice 5.1 až 5.4), alebo výhodnejšie na základe pravdivostnej tabuľky.

Výstupné funkcie:

$$A = \overline{G1}G4 + \overline{G1}G2\overline{G3} + \overline{G1}G2G3 + G1G2\overline{G3} + G1\overline{G2}G3\overline{G4} \quad (5.1)$$

$$B = \overline{G1} \quad (5.2)$$

$$C = \overline{G1}G2 + G1\overline{G3} + G1G4 \quad (5.3)$$

$$D = G1G3 + \overline{G2}G3 \quad (5.4)$$

Zostavíme VHDL kód podľa pravdivostnej tabuľky:

```
--definovanie kniznic standardov
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

--definovanie vstupov a vystupov
entity gray is
port
(
    gray_code      : IN  std_logic_vector (4 downto 1);  --vektor vstupov
    plus3_code     : OUT std_logic_vector (4 downto 1);  --vektor vystupov
    ostatne_segmeny : OUT std_logic_vector (11 downto 0)  --vektor nepouzitych segmentov
);
end entity gray;

-- samotne telo programu
architecture gray_tabulka of gray is
begin
    ostatne_segmeny <= "111111111111"; --pripojenie nepouzitych segmentov na jednotku

    process (gray_code)                --prevod gray kodu na plus3 kod
    begin
        case gray_code is
            when "0000" => plus3_code <= "1100";
            when "0001" => plus3_code <= "1011";
            when "0011" => plus3_code <= "1010";
            when "0010" => plus3_code <= "1001";
            when "0110" => plus3_code <= "1000";
            when "0111" => plus3_code <= "0111";
            when "0101" => plus3_code <= "0110";
            when "0100" => plus3_code <= "0101";
            when "1100" => plus3_code <= "0100";
            when "1101" => plus3_code <= "0011";
            when others => plus3_code <= "1111";      -- ak nieco ine...
        end case;
    end process;
end architecture gray_tabulka;
```

5.3.6 KOMPILÁCIA

Kompiláciu spustíme voľbou **Start Compilation** z **Processing menu**, ikonou na panely nástrojov alebo voľbou **Compiler Tool** z **Tools menu**.

Po kompilácii musíme ešte výstupným signálom priradiť zodpovedajúce čísla pinov obvodu, voľbou **Pins** v **Assignments menu** (obr.2). V stĺpci **To** napíšeme názvy výstupných signálov a v stĺpci **Location** im priradíme príslušné čísla pinov podľa tab.1.

	To	Location	General Function	Special Function	Reserved
1	gray_code[1]	PIN_38	Row I/O		
2	gray_code[2]	PIN_39	Row I/O		
3	gray_code[3]	PIN_40	Row I/O		
4	gray_code[4]	PIN_41	Row I/O		
5	plus3_code[1]	PIN_19	Row I/O		
6	plus3_code[2]	PIN_21	Row I/O		
7	plus3_code[3]	PIN_8	Row I/O		
8	plus3_code[4]	PIN_11	Row I/O	CLKUSR	
9	ostatne_segmeny[0]	PIN_17	Row I/O		
10	ostatne_segmeny[1]	PIN_18	Row I/O		
11	ostatne_segmeny[2]	PIN_20	Row I/O		
12	ostatne_segmeny[3]	PIN_23	Row I/O	RDYnBSY	
13	ostatne_segmeny[4]	PIN_24	Row I/O		
14	ostatne_segmeny[5]	PIN_25	Row I/O		
15	ostatne_segmeny[6]	PIN_6	Row I/O		
16	ostatne_segmeny[7]	PIN_7	Row I/O		
17	ostatne_segmeny[8]	PIN_9	Row I/O		
18	ostatne_segmeny[9]	PIN_12	Row I/O		
19	ostatne_segmeny[10]	PIN_13	Row I/O		
20	ostatne_segmeny[11]	PIN_14	Row I/O		
21	<<new>>	<<new>>			


Obr.2: Okno Assignments pins

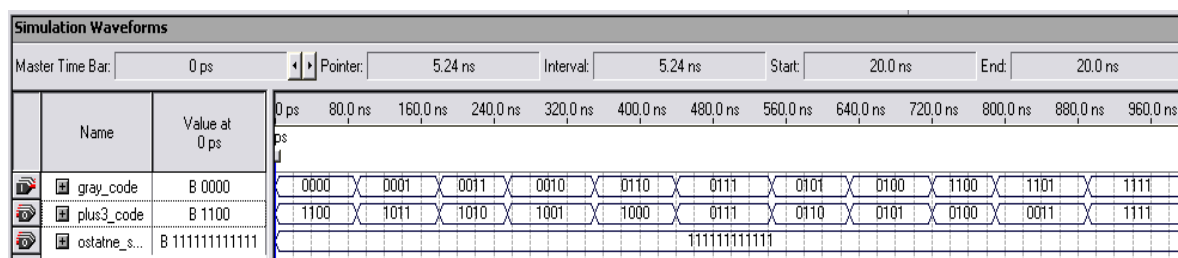
5.3.7 SIMULÁCIA

5.3.7.1 FUNKČNÁ SIMULÁCIA

Po odladení všetkých chýb pomocou kompilácie môžeme prejsť k simulácii projektu. Simuláciu vykonáme podľa nasledujúceho postupu:

- Vytvoríme vektorový súbor, v ktorom si zadefinujeme tzv. stimuly (stimuly sú vstupné signály, ktoré spôsobia požadované prechodové zmeny v realizácii) – *Vector Waveform file (.vwf)*, voľbou **New** z **File menu**
- Voľbou **Save As** z **File menu** uložíme tento vektorový súbor, v tomto prípade s názvom *gray.vwf*

- Pomocou **Node Finder** vložíme do tohto súboru všetky vstupy a výstupy, ktoré chceme simulovať. **Node Finder** otvoríme nasledujúcim postupom: **View**→**UtilityWindows**→**Node Finder**. Označíme požadované piny (Shift + ľavé tlačidlo myši), skopírujeme ich (Ctrl + C) a vložíme do vektorového súboru (Ctrl+V). Druhou možnosťou je chytiť myšou požadovaný pin a jednoducho ho presunúť do vektorového súboru (.vwf).
- Nastavíme koncový čas simulácie. V položke **Edit**→**End Time** nastavíme hodnotu time na 100µs.
- Na vstupných pinoch musíme zadefinovať vstupné kombinácie. Vykonáme to kliknutím na ikonu **Waveform Editing Tool** () . V riadku určenom pre vstupné piny vyberieme úsek pre danú vstupnú kombináciu. Po výbere sa zobrazí okno do ktorého túto kombináciu zadefinujeme. Kombinácie vstupov vyberieme tak, aby sme použili všetky kombinácie z pravdivostnej tabuľky.
- Otvoríme okno **Simulator Tool** z **Tools menu**. Je v ňom možné meniť časovú resp. funkčnú simuláciu (**timing**→**functional**). Zvolíme funkčnú simuláciu. V ďalšom kroku je potrebné najskôr vygenerovať **Netlist**. Po vygenerovaní **Netlistu** môžeme spustiť funkčnú simuláciu kliknutím na tlačidlo **Start**. Výsledkom bude okno **Simulation Report** (obr.3) s funkčnou simuláciou nášho projektu, ktoré otvoríme kliknutím na tlačidlo **Report**.



Obr.3: Výsledok funkčnej simulácie

Z výsledkov simulácie môžeme vidieť či prevodník pracuje správne, alebo nie. Musíme uvažovať s tou skutočnosťou, že na výstup sme museli pripojiť hradlá NOT. Z dôvodu, že LED svieti pri logickej hodnote „0“. Preto logická úroveň „0“ predstavuje logickú „1“ a logická úroveň „1“ predstavuje logickú „0“.

5.3.8 PROGRAMOVANIE/KONFIGURÁCIA

Po úspešnej kompilácii a preverení projektu pomocou simulácie môžeme prejsť k programovaniu obvodu. Programovanie vykonáme výberom položky **Programmer** z **Tools menu**. V programovacom okne musíme zaškrtnúť políčko **Program/Configure** a kliknutím na tlačidlo **Start** spustíme proces programovania. Od tohto okamihu je projekt naprogramovaný do súčiasťky na doske UP1 CPLD. Teraz môžeme preveriť činnosť nášho projektu priamo na doske.

5.4 ZADANIE PRÍKLADU Č.2

Navrhňte prevodník z Grayovho kódu do kódu +3 ku BCD (0-9). Návrh realizujte vo VHDL kóde na základe pravdivostnej tabuľky. Realizáciu vykonajte v textovom editore prostredia Quartus II.

5.5 RIEŠENIE

5.5.1 ROZBOR

Postup riešenia tohto príkladu bude ten istý ako v príklade č.1. Zostavíme si pravdivostnú tabuľku prevodníka. Na základe pravdivostnej tabuľky (tab.3) zostavíme VHDL kód, vykonáme kompiláciu, priradenie pinov, simuláciu a konečnú konfiguráciu projektu.

dekadický	vstupy BCD kód				výstupy Kód +3 ku BCD			
	G4	G3	G2	G1	D	C	B	A
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Tab.3: Pravdivostná tabuľka prevodníka

Ako vstupy využijeme prepínače **FLEX_SWITCH** a zobrazíme ich na sedem segmentovom displeji. Platí tabuľka pinov (tab.1) s tým rozdielom, že vstupné piny majú názov: *bcd_code* s číslami od 1 po 4. V tabuľke č.1 a 3 sú označené ako *G1* až *G4*. Výstupné piny majú názov: *plus3_code* s číslami 1 až 4 a v tabuľkách č.1 a 3 sú označené ako *A*, *B*, *C* a *D*. Nepoužité segmenty majú názov: *ostatne_segmeny* s číslami od 0 po 11 a v tabuľke č.1 sú označené ako *Other* s číslami 0 až 11. Nepoužité segmenty opäť privedieme na logickú úroveň „1“, aby zbytočne nesvietili.

5.5.2 OTVORENIE NOVÉHO PROJEKTU

Otvorenie nového projektu vykonáme podľa postupu uvedeného v kap. 5.3.3 s tým rozdielom, že projektu a najvyššej úrovne entity dáme názov *BCD*.

Otvoríme nové textové okno, ktoré uložíme pod názvom *bcd.vhd*.

Zostavíme VHDL kód podľa pravdivostnej tabuľky:

```
--definovanie kniznic
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

--definovanie vstupov a vystupov
entity BCD is
port
(
  bcd_code      : IN   std_logic_vector (4 downto 1);      --vstupny vektor
  plus3_code    : OUT  std_logic_vector (4 downto 1);      --vystupny vektor
  ostatne_segmeny : OUT std_logic_vector (11 downto 0)      --vektor nepouzitych segmentov
);
end entity BCD;

--samotna architektura
architecture bcd_tabulka of bcd is
begin
  ostatne_segmeny <= "11111111111"; --pripojenie nepouzitych segmentov na log.1

  process (bcd_code)
    --pravdivostna tabulka
    begin
      case bcd_code is
        when "0000" => plus3_code <= "1100";
        when "0001" => plus3_code <= "1011";
        when "0010" => plus3_code <= "1010";
        when "0011" => plus3_code <= "1001";
        when "0100" => plus3_code <= "1000";
        when "0101" => plus3_code <= "0111";
        when "0110" => plus3_code <= "0110";
        when "0111" => plus3_code <= "0101";
        when "1000" => plus3_code <= "0100";
        when "1001" => plus3_code <= "0011";
        when others => plus3_code <= "1111"; -- ak nieco ine...
      end case;
    end process;
end architecture bcd_tabulka;
```

5.5.3 KOMPILÁCIA

Kompiláciu spustíme voľbou **Start Compilation** z **Processing menu**, ikonou na panely nástrojov alebo voľbou **Compiler Tool** z **Tools menu**.

Po kompilácii musíme ešte výstupným signálom priradiť zodpovedajúce čísla pinov obvodu, voľbou **Pins** v **Assigments menu** (obr.4).

	To	Location	General Function	Special Function	Reserved
1	bcd_code[1]	PIN_38	Row I/O		
2	bcd_code[2]	PIN_39	Row I/O		
3	bcd_code[3]	PIN_40	Row I/O		
4	bcd_code[4]	PIN_41	Row I/O		
5	plus3_code[1]	PIN_19	Row I/O		
6	plus3_code[2]	PIN_21	Row I/O		
7	plus3_code[3]	PIN_8	Row I/O		
8	plus3_code[4]	PIN_11	Row I/O	CLKUSR	
9	ostatne_segmenty[0]	PIN_17	Row I/O		
10	ostatne_segmenty[1]	PIN_18	Row I/O		
11	ostatne_segmenty[2]	PIN_20	Row I/O		
12	ostatne_segmenty[3]	PIN_23	Row I/O	RDYnBSY	
13	ostatne_segmenty[4]	PIN_24	Row I/O		
14	ostatne_segmenty[5]	PIN_25	Row I/O		
15	ostatne_segmenty[6]	PIN_6	Row I/O		
16	ostatne_segmenty[7]	PIN_7	Row I/O		
17	ostatne_segmenty[8]	PIN_9	Row I/O		
18	ostatne_segmenty[9]	PIN_12	Row I/O		
19	ostatne_segmenty[10]	PIN_13	Row I/O		
20	ostatne_segmenty[11]	PIN_14	Row I/O		
21	<<new>>	<<new>>			

Obr.4: Okno Assignments Pins

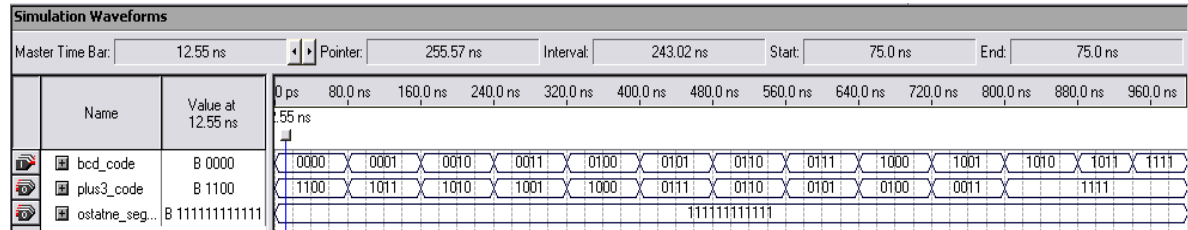
5.5.4 FUNKČNÁ SIMULÁCIA

Vytvoríme vektorový súbor, v ktorom si zadefinujeme tzv. stimuly – *Vector Waveform file* (.vwf), príkazom **New** z **File menu** a uložíme pod názvom *bcd.vwf*. Do vektorového súboru vložíme všetky vstupy a výstupy, ktoré chceme simulovať pomocou voľby **Node Finder**.

Na vstupných pinoch zadefinujeme vstupné kombinácie. Vyberieme kombinácie na základe pravdivostnej tabuľky tak, aby reprezentovali všetky možnosti.

Otvoríme okno **Simulator Tool** z **Tools menu** a zmeníme časovú simuláciu na funkčnú simuláciu (**timing**→**functional**). Kliknutím na tlačidlo s nápisom **Netlist** necháme vygenerovať **Netlist**. Po jeho vygenerovaní spustíme funkčnú simuláciu

kliknutím na tlačidlo **Start**. Výsledkom je okno **Simulation Report** (obr.5) s funkčnou simuláciou nášho projektu, ktoré je možné otvoriť kliknutím na tlačidlo **Report**.



Obr.5: Výsledky funkčnej simulácie

Na základe funkčnej simulácie zistíme či náš prevodník pracuje správne, alebo nie. Po vyhodnotení výsledkov simulácie prejdeme k programovaniu/konfigurácii projektu. Programovanie spustíme výberom voľby **Programmer** z **Tools menu**.

PRÍLOHA

Výsledný VHDL kód príkladu číslo 1. zostavený na základe algebraických rovníc, ktoré dostaneme pomocou syntézy z Karnaughových máp:

```
--zadefinovanie kniznic standardov
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY work;

--definovanie vstupnych a vystupnych premennych
ENTITY prevodnik_Gray_vhdl IS
    port
    (
        G1 : IN STD_LOGIC;
        G2 : IN STD_LOGIC;
        G3 : IN STD_LOGIC;
        G4 : IN STD_LOGIC;
        B : OUT STD_LOGIC;
        A : OUT STD_LOGIC;
        C : OUT STD_LOGIC;
        D : OUT STD_LOGIC;
        other0 : OUT STD_LOGIC;
        other1 : OUT STD_LOGIC;
        other2 : OUT STD_LOGIC;
        other3 : OUT STD_LOGIC;
        other4 : OUT STD_LOGIC;
        other5 : OUT STD_LOGIC;
```

```
        other6 : OUT STD_LOGIC;
        other7 : OUT STD_LOGIC;
        other8 : OUT STD_LOGIC;
        other9 : OUT STD_LOGIC;
        other10 : OUT STD_LOGIC;
        other11 : OUT STD_LOGIC;
    );
END prevodnik_Gray_vhdl;

--samotne telo navrhnu
ARCHITECTURE bdf_type OF prevodnik_Gray_vhdl IS

--pomocne premenne
signal SYNTHESIZED_WIRE_26 : STD_LOGIC;
signal SYNTHESIZED_WIRE_1 : STD_LOGIC;
signal SYNTHESIZED_WIRE_2 : STD_LOGIC;
signal SYNTHESIZED_WIRE_27 : STD_LOGIC;
signal SYNTHESIZED_WIRE_28 : STD_LOGIC;
signal SYNTHESIZED_WIRE_8 : STD_LOGIC;
signal SYNTHESIZED_WIRE_10 : STD_LOGIC;
signal SYNTHESIZED_WIRE_11 : STD_LOGIC;
signal SYNTHESIZED_WIRE_12 : STD_LOGIC;
signal SYNTHESIZED_WIRE_29 : STD_LOGIC;
signal SYNTHESIZED_WIRE_14 : STD_LOGIC;
signal SYNTHESIZED_WIRE_16 : STD_LOGIC;
signal SYNTHESIZED_WIRE_17 : STD_LOGIC;
signal SYNTHESIZED_WIRE_18 : STD_LOGIC;
signal SYNTHESIZED_WIRE_19 : STD_LOGIC;
signal SYNTHESIZED_WIRE_21 : STD_LOGIC;
signal SYNTHESIZED_WIRE_22 : STD_LOGIC;

BEGIN

--nastavenie nepouzitych segmentov na UCC
other0 <= '1';
other1 <= '1';
other2 <= '1';
other3 <= '1';
other4 <= '1';
other5 <= '1';
other6 <= '1';
other7 <= '1';
other8 <= '1';
other9 <= '1';
other10 <= '1';
other11 <= '1';

--samotna realizacia logiky prevodnika pomocou hradiel NAND (1:1)
SYNTHESIZED_WIRE_8 <= NOT(G4);
SYNTHESIZED_WIRE_27 <= NOT(G1);
SYNTHESIZED_WIRE_17 <= NOT(SYNTHESIZED_WIRE_26 AND G1);
SYNTHESIZED_WIRE_22 <= NOT(SYNTHESIZED_WIRE_1 AND SYNTHESIZED_WIRE_2);
SYNTHESIZED_WIRE_14 <= NOT(G1 AND G2 AND SYNTHESIZED_WIRE_26);
SYNTHESIZED_WIRE_10 <= NOT(SYNTHESIZED_WIRE_27 AND G2 AND G3);
SYNTHESIZED_WIRE_11 <= NOT(SYNTHESIZED_WIRE_26 AND SYNTHESIZED_WIRE_28 AND
SYNTHESIZED_WIRE_27);
SYNTHESIZED_WIRE_29 <= NOT(G1 AND SYNTHESIZED_WIRE_8 AND SYNTHESIZED_WIRE_28 AND G3);
SYNTHESIZED_WIRE_28 <= NOT(G2);
SYNTHESIZED_WIRE_26 <= NOT(G3);
SYNTHESIZED_WIRE_19 <= NOT(SYNTHESIZED_WIRE_10 AND SYNTHESIZED_WIRE_11 AND
SYNTHESIZED_WIRE_12 AND SYNTHESIZED_WIRE_29 AND SYNTHESIZED_WIRE_14 AND
SYNTHESIZED_WIRE_29);
SYNTHESIZED_WIRE_21 <= NOT(SYNTHESIZED_WIRE_16 AND SYNTHESIZED_WIRE_17 AND
SYNTHESIZED_WIRE_18);

A <= NOT(SYNTHESIZED_WIRE_19);
B <= NOT(SYNTHESIZED_WIRE_27);
C <= NOT(SYNTHESIZED_WIRE_21);
D <= NOT(SYNTHESIZED_WIRE_22);

SYNTHESIZED_WIRE_1 <= NOT(G3 AND SYNTHESIZED_WIRE_28);
SYNTHESIZED_WIRE_2 <= NOT(G3 AND G1);
```

```
SYNTHESIZED_WIRE_18 <= NOT(G4 AND G1);  
SYNTHESIZED_WIRE_12 <= NOT(G4 AND SYNTHESIZED_WIRE_27);  
SYNTHESIZED_WIRE_16 <= NOT(G2 AND SYNTHESIZED_WIRE_27);  
END;
```