Cvičenie 4

Prevodník BCD na 7-segmentový displej. Funkčná a časová simulácia v Quartuse II a ModelSime. Vytváranie stimulov vo VHDL (Test benche). Praktická demonštrácia funkčnosti projektu na vývojovej doske CPLD_KIT.

Zadanie:

Realizujte prevodník BCD na 7-segmentový displej na vývojovej doske CPLD_KIT. Jeho štruktúru popíšte VHDL jazykom. Dizajn odsimulujte v simulátore Quartusu II pomocou grafického zadania vstupných signálov a v ModelSime pomocou VHLD test benchu.

Poznámka:

Toto je posledný kompletný postup, ako zostaviť a odsimulovať projekt. Ďalšie cvičenia budú zamerané na analýzu VHDL kódov a realizáciu zložitejších dizajnov. Preto vysoko odporúčam naštudovať celý postup, naučiť sa ho naspamäť, aby ste nestrácali na ďalších cvičeniach čas tým, že neviete kam kliknúť. Postup vyzerá byť dlhý a komplikovaný, avšak ak ho pochopíte, budete ho vedieť naklikať za 5 minút. Ak by boli niektoré časti vyslovene nezrozumiteľné, kontaktovať ma na miso.varchola@tuke.sk.

Plán cvičenia:

1) Vytvorenie projektu v Quartuse II
2) Napísanie VHDL kódu pre prevodník BCD na 7-segmentový displej
3) Kompilácia projektu a priradenie pinov
4) Funkčná a časová simulácia v simulátore Quartusu II
5) Napísanie VHDL test benchu
6) Konfigurácia prostredia Quartus II pre spúšťanie simulácie v ModelSime
7) Funkčná a časová simulácia v ModelSime
8) Konfigurácia dosky

Teoretický rozbor

BCD -> 7-seg. prevodník

Prevodník BCD (Binary Coded Decimal) na 7-segmentový displej sa používa na dekódovanie informácie v BCD kóde na informáciu vo forme číslice na 7-segmentovom displeji, čo je jednoducho interpretovateľné človekom. BCD kód sa používa v elektronických logických systémoch vďaka jeho jednoduchému spracovaniu. Takého prevodníky existujú aj v integrovanom prevedení, ktoré sa často používalo v minulosti, keď programovateľná logika nebola alebo bola nedostupná. Príkladom je napr. integrovaný obvod D147C vyrábaný bývalým DDR, resp. novší CD4511 (CMOS technológia) a veľa ďalších...

Pravdivostná tabuľka prevodníku (pre dosku CPLD_KIT, tabuľka je zrejmá zo schémy):

	DIP_SWICH (vstup)					Displej DG2 (výstup)						
	A	B	C	D		a	b	c	d	e	f	g
0	1	1	1	1		0	0	0	0	0	0	1
1	1	1	1	0		1	0	0	1	1	1	1
2	1	1	0	1		0	0	1	0	0	1	0
3	1	1	0	0		0	0	0	0	1	1	0
4	1	0	1	1		1	0	0	1	1	0	0
5	1	0	1	0		0	1	0	0	1	0	0
6	1	0	0	1		0	1	0	0	0	0	0
7	1	0	0	0		0	0	0	1	1	1	0

8	0	1	1	1		0	0	0	0	0	0	0
9	0	1	1	0		0	0	0	0	1	0	0
10 (a)	0	1	0	1		0	0	0	1	0	0	0
11 (b)	0	1	0	0		1	1	0	0	0	0	0
12 (c)	0	0	1	1		1	1	1	0	0	1	0
13 (d)	0	0	1	0		1	0	0	0	0	1	0
14 (e)	0	0	0	1		0	1	1	0	0	0	0
15 (f)	0	0	0	0		0	1	1	1	0	0	0
pin	31	33	34	35		2	3	5	6	42	43	44

Funkčná a časová simulácia

Robiť simulácie počas vývoja hardvéru v CPLD alebo FPGA je veľmi dobrým zvykom, nakoľko chyby je veľmi zložité hľadať v čipe na doske. Simulácie sú takmer nevyhnutné pre úspešnú realizáciu projektu. To platí aj pre jednoduché projekty. Je veľmi dôležité rozlišovať medzi funkčnou a časovou simuláciou. Pri funkčnej simulácii je VHDL kód (prípadne schéma preložená do VHDL kódu) simulovaný z matematického hľadiska. Simulujú sa rovnice. Takáto simulácia je rýchla, ale nemusí za všetkých podmienok odzrkadľovať správanie sa hardvéru. Pri dodržaní určitých pravidiel je možné považovať výsledok funkčnej simulácie za veľmi blízky hardvérovej realizácii projektu. Minimálne je možné výsledok použiť na posúdenie, či sa kód VHDL alebo schéma chová z funkčného hľadiska tak, ako chceme my. Časová simulácia uvažuje vo výpočtoch aj oneskorenie hradiel a oneskorenia plynúce zo šírenia sa signálu medzi hradlami. Táto simulácia je bližšie k správaniu sa reálneho hardvéru ale trvá dlhšie. Vo veľkých projektoch sa časová simulácia takmer nepoužíva, pretože môže trvať aj niekoľko hodín až dní, zatiaľ čo funkčná rádovo minúty.

Simulácia v Quartuse a ModelSime

Quartus je EDA nástroj vyvíjaný firmou Altera, ktorá sa zaoberá najmä vývojom a výrobou CPLD a FPGA čipov. Quartus obsahuje všetky náležitosti potrebné pre vývoj CPLD a FPGA. Okrem iného aj simulátor, ktorý ako-tak zvyšuje pridanú hodnotu vývojového prostredia Quartus II. Existujú však aj firmy, ktoré sa špecializujú na vývoj EDA (Electronic Design Automation) nástrojov, napr. firma MentorGraphics. Táto firma má vo svojom portfóliu celú sadu nástrojov potrebných pre vývoj elektroniky. Pre simuláciu logických obvodov vyvíjajú ModelSim. Dá sa povedať že práca s ModelSimom je podstatne rýchlejšia a efektívnejšia ako s Quartus-simulátorom. Donedávna bolo nutné za ModelSim platiť, dnes však existuje tzv. Altera Edition (ModelSim pre Alteru), ktorú Altera distribuuje zdarma (pretože jadro biznisu Altery tvoria čipy, ktoré chce predávať) spolu s Web Edíciou Quartusu. Takáto kombinácia postačuje aj na väčšie dizajny. Preto je veľmi výhodné naučiť sa používať aj ModelSim. Pred každou simuláciou odporúčam projektové súbory uložiť a projekt skompilovať.

Zadávanie stimulov (vstupných signálov) graficky a VHDL kódom

Jednoduchosť zadávania vstupných signálov graficky zvádza k jeho použitiu. Je to však nevýhodné a neprehľadné v rozsiahlych dizajnoch. Preto sa budeme učiť vytvoriť stimul (testovacie vstupné signály) pomocou VHDL kódu, ktorý je viac jednoznačný.

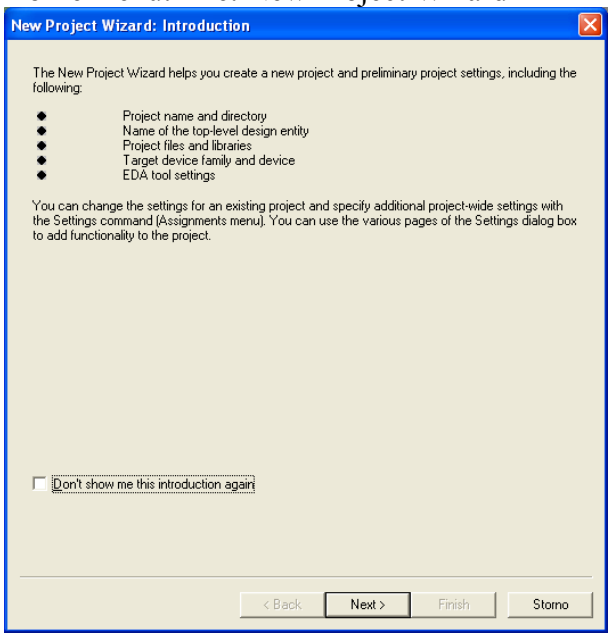
Poznámky ku VHDL

Všetky identifikátory (meno entity, meno signálu, meno architektúry...) začínajte vždy s veľkým alebo malým písmenom. Nikdy nie číslom alebo akýmkoľvek iným znakom. Ako oddeľovač v identifikátore používajte podtržník _ . V identifikátoroch používajte len písmená anglickej abecedy a čísla. Dodržujte formátovanie. Komentáre na jeden riadok sa píšu za dvojicu znakov mínus (`-- toto je komentar`). V komentároch sa odporúča nepoužívať

interpunkciu, opäť len písmena anglickej abecedy (alebo ešte lepšie – používať angličtinu). Syntax bude vysvetlená na príkladoch. Quartus Editor zvýrazňuje kľúčové slová modrou farbou a komentáre zelenou. Pri vysvetľovaní syntaxe dávajte pozor, kde sa píšu dvojbodky, bodkočiarky a čiarky. V ich písaní je zmysel, avšak na prvý pohľad ich použitie môže vyzerať komplikovane. VHDL jazyk nie je „case sensitive“, to znamená že AbCdE je ekvivalentné ku aBcDe.

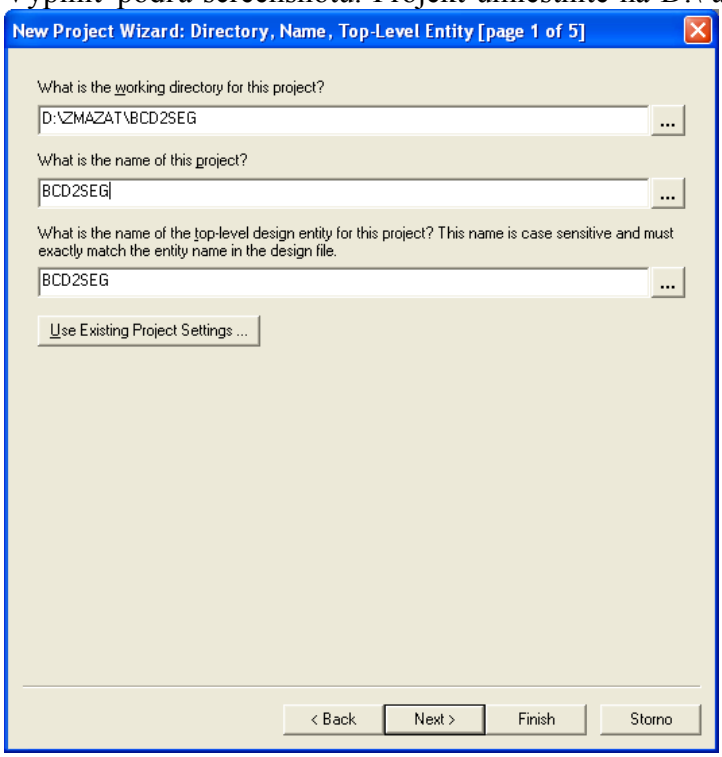
Postup:

1. **Vytvorenie projektu v Quartuse II**
   a. Otvoriť Quartus II
   b. Horné menu: File: New Project Wizard



   c. Next>

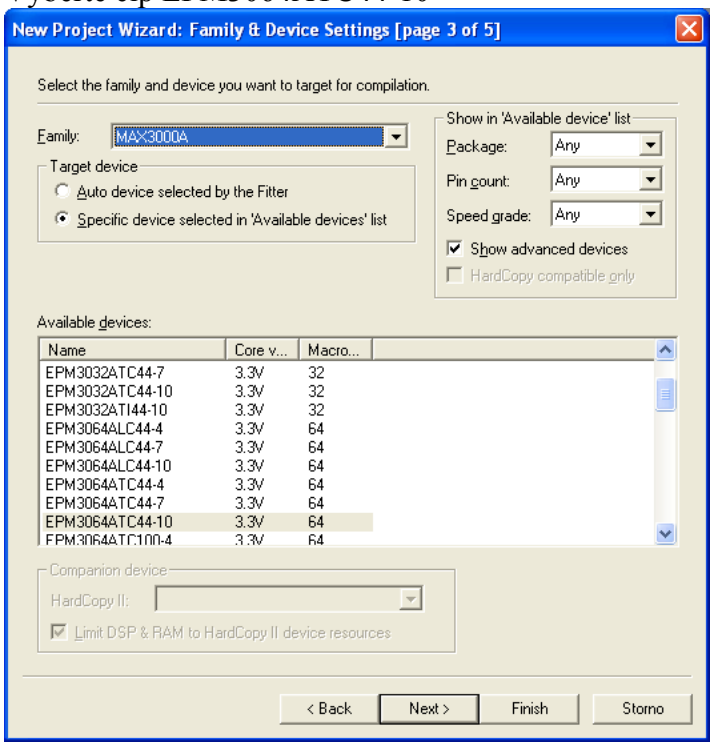
d. Vyplniť podľa screenshotu. Projekt umiestnite na D:\ a nazvite BCD2SEG !!!



e. Next>

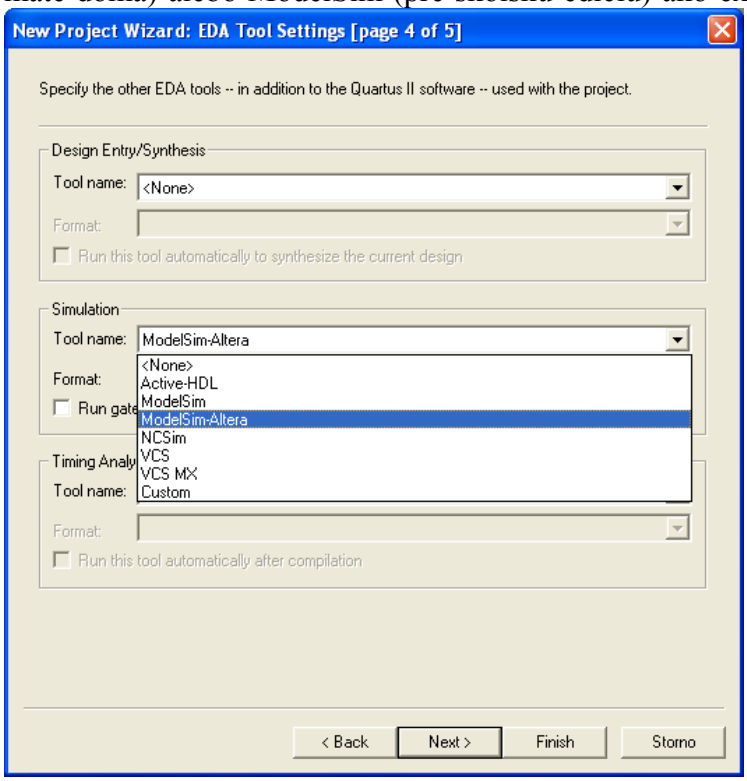
f. Next>

g. Vyberte čip EPM3064ATC44-10



h. Next>

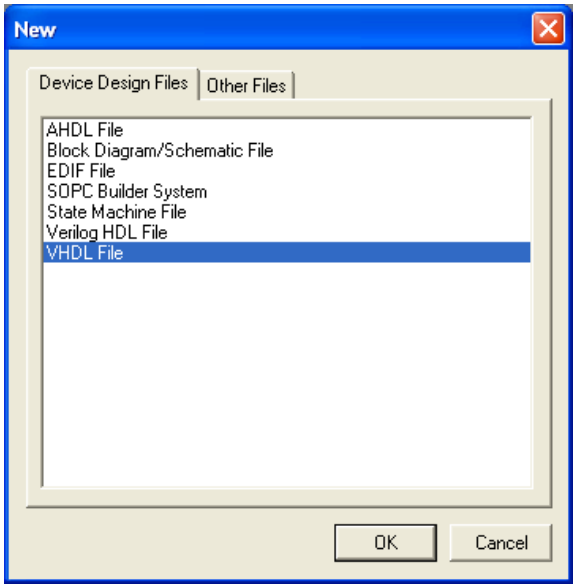
i. Vyberte ModelSim-Altera (pre Web Edíciu Quartusu a ModelSimu – to čo máte doma) alebo ModelSim (pre školskú edíciu) ako externý EDA simulátor



j. Next>
k. Finish

2. Napísanie VHDL kódu pre prevodník BCD na 7-segmentový displej

a. Horné menu: New: File: VHDL File



b. OK
c. Opište kód:

```
--prevodnik BCD na 7-segmentovy displej

-- zoznam pouzivanych kniznic
library ieee;
use ieee.std_logic_1164.all;
```

```
--definicia entity
entity BCD2SEG is                      --BCD2SEG je nazov entity
port(
  dip_sw  : in std_logic_vector  (3 downto 0); -- vstupny port
  display : out std_logic_vector (6 downto 0)  -- vystupny port
);
end BCD2SEG;

architecture arch of BCD2SEG is     -- arch je nazov architektury
begin
  with dip_sw select                -- syntax prepinaca
    display <=
      "0000001" when "1111", -- 0
      "1001111" when "1110", -- 1
      "0010010" when "1101", -- 2
      "0000110" when "1100", -- 3
      "1001100" when "1011", -- 4
      "0100100" when "1010", -- 5
      "0100000" when "1001", -- 6
      "0001111" when "1000", -- 7
      "0000000" when "0111", -- 8
      "0000100" when "0110", -- 9
      "0001000" when "0101", -- A
      "1100000" when "0100", -- b
      "1110010" when "0011", -- c
      "1000010" when "0010", -- d
      "0110000" when "0001", -- E
      "0111000" when "0000", -- F
      "1111111" when others; -- pre ine moznosti
end arch;
```

d. Súbor uložíme: Horné menu: File: Save As: BCD2SEG.vhd

e. **Analýza kódu: knižnice**. Na začiatku kódu je nutné uviesť knižnice, v ktorých sú definované napr. typy signálov, ktoré nie sú štandardne zahrnuté vo VHDL:

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

Budeme používať knižnicu IEEE (Institute of Electrical and Electronics Engineers). Z nej použijeme definíciu štandardnej logiky „ieee.std_logic_1164“ (viac na prednáške). Túto knižnicu budeme používať vždy.

f. **Analýza kódu: entita**. Entitu si môžeme predstaviť ako elektronickú logickú súčiastku (alebo integrovaný obvod), ktorá má svoje meno, vstupy a výstupy. Tento popis musí obsahovať každý VHDL kód, ktorým popisujeme hardvér, pre syntézu v CPLD alebo FPGA.

```
entity BCD2SEG is       --BCD2SEG je nazov entity
port(
   dip_sw  : in std_logic_vector  (3 downto 0); --vstup
   display : out std_logic_vector (6 downto 0)  --vytup
);
end BCD2SEG;
```

g. **Analýza kódu: `std_logic_vector`.** Označuje typ - zoskupenie bitov štandardnej logiky. Typ `std_logic_vector` sa používa pre syntézu logiky. Iný typ je napr. `integer`.

h. **Analýza kódu: `(3 downto 0).`** znamená že sa jedná o 4 bitový vektor s príslušným radením bitov.

i. **Analýza kódu: architektúra.** Vo vnútri architektúry bude popis logiky, ktorá bude syntetizovaná. Aj keď VHDL umožňuje použitie viacerých architektúr, v rámci jedného VHDL súboru, vždy budeme používať len jednu. Riadok „end arch“ je posledným riadkom v súbore a zvyšný syntetizovateľný kód budeme písať medzi „begin“ a „end arch“. Nad begin sa píšu deklaracie komponentov a signálov, ale o tom neskôr...

```
architecture arch of BCD2SEG is--arch je nazov architekt.
 -- deklaracie komponentov a signalov
begin

 -- popis logiky

end arch;
```

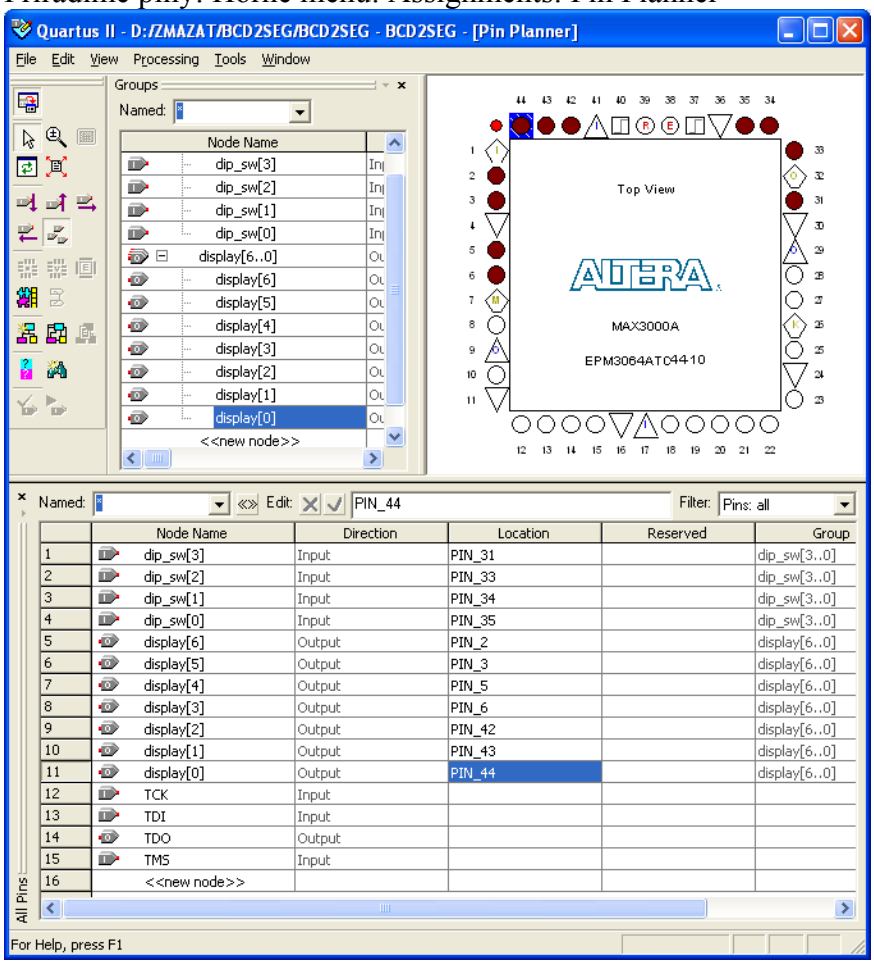
j. **Analýza kódu: kombinačný obvod.** Jedná sa o prepínač (kóder). V tomto prípade prepínač použijeme na implementáciu pravdivostnej tabuľky. Prepínač je možné popísať viacerými spôsobmi použitím syntaxe VHDL. Jeden z nich je:

```
with dip_sw select       -- syntax prepinaca
  display <=
    "0000001" when "1111", -- 0
    "1001111" when "1110", -- 1
    "0010010" when "1101", -- 2
    "0000110" when "1100", -- 3
    "1001100" when "1011", -- 4
    "0100100" when "1010", -- 5
    "0100000" when "1001", -- 6
    "0001111" when "1000", -- 7
    "0000000" when "0111", -- 8
    "0000100" when "0110", -- 9
    "0001000" when "0101", -- A
    "1100000" when "0100", -- b
    "1110010" when "0011", -- c
    "1000010" when "0010", -- d
    "0110000" when "0001", -- E
    "0111000" when "0000", -- F
    "1111111" when others; -- pre ine moznosti
```

3. Kompilácia a priradenie pinov

a. Projekt skompilujeme: Horné menu: Processing: Start Compilation

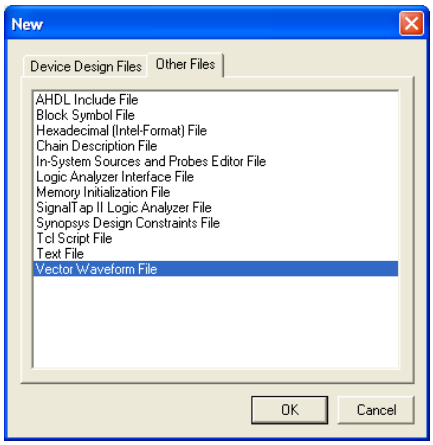
b. Priradíme piny: Horné menu: Assignments: Pin Planner



c. Projekt skompilujeme: Horné menu: Processing: Start Compilation

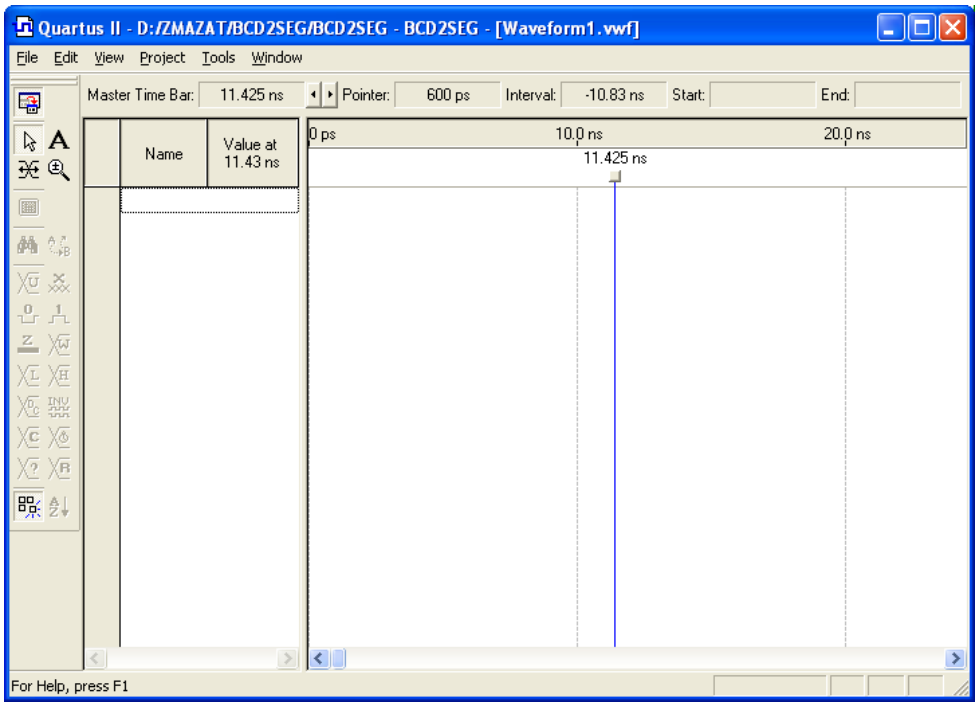
4. Funkčná a časová simulácia v simulátore Quartusu II

a. Vytvorenie stimulu graficky: Horné menu: File: New: Other Files: Vector Waveform File



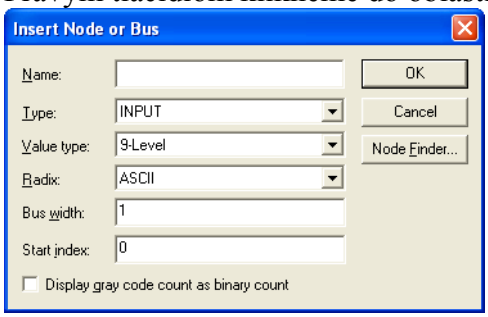
b. OK

c.



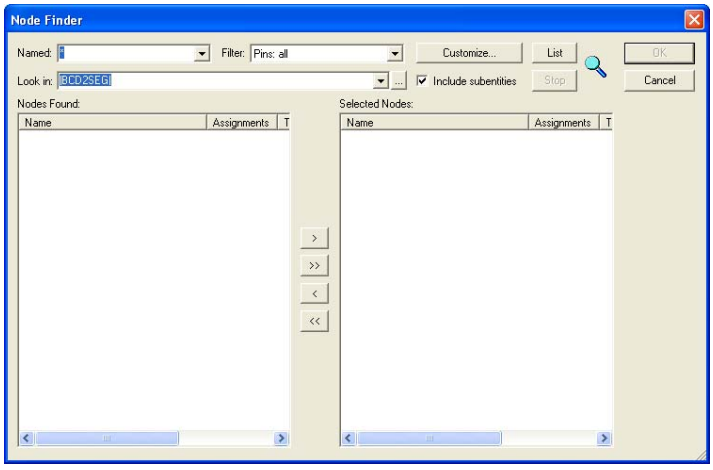
d. Pravým tlačidlom klikneme do oblasti pod Name: Insert: Insert Node or Bus...

e.

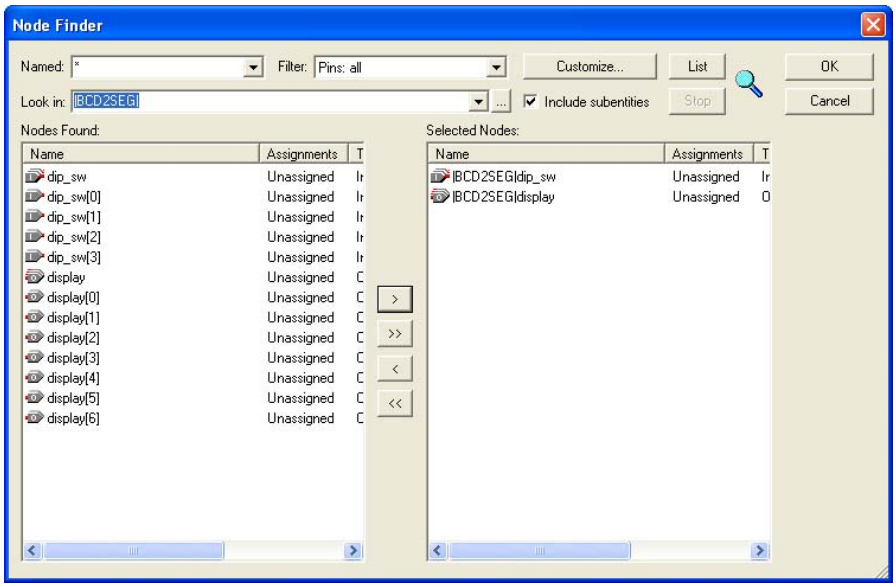


f. Kliknúť na Node Finder

g.



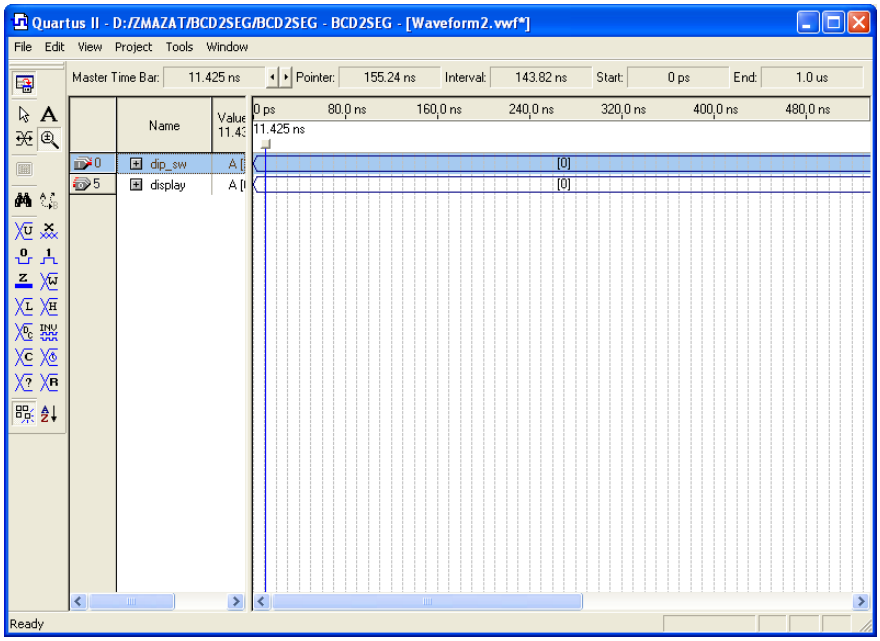
h. Kliknúť na List

i. 

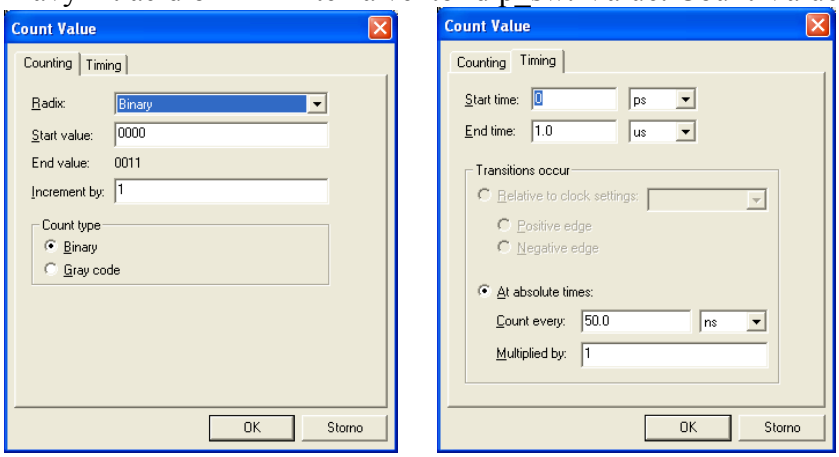
j. Vyberieme piny podľa obrázku hore

k. OK

l. OK

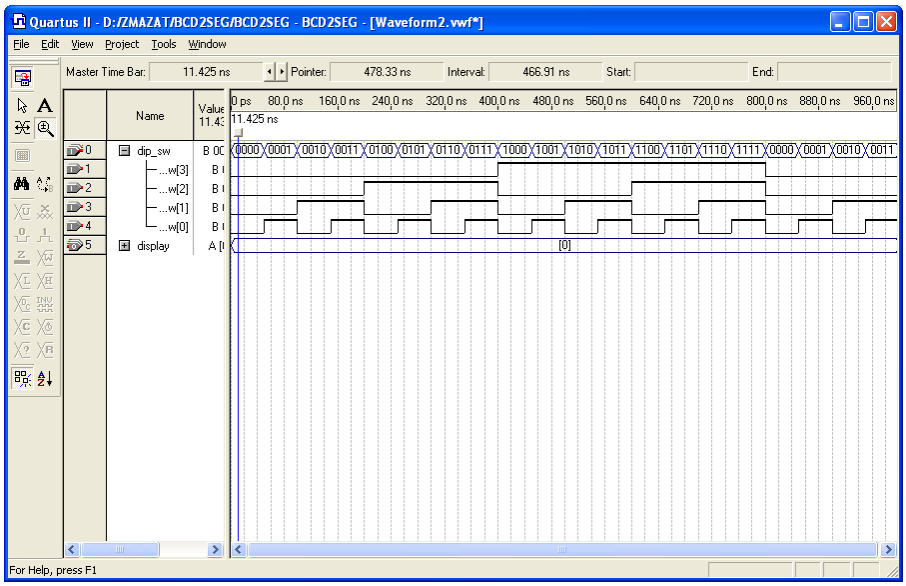
m. 

n. Pravým tlačidlom kliknite na vektor dip_sw: Value: Count Value

o. 

p. Nastavte hodnoty ako na obrázkoch hore

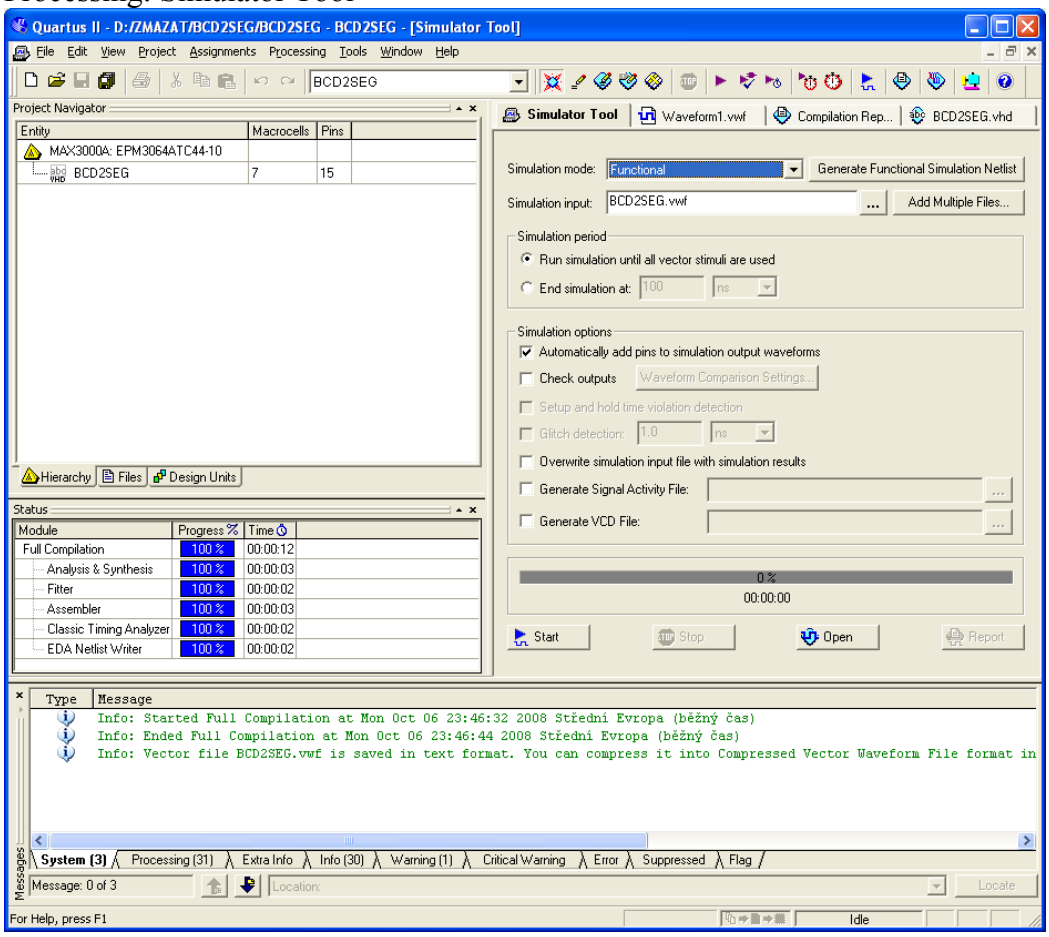
q. OK



r.

s. Súbor uložiť ako BCD2SEG.vwf

t. Processing: Simulator Tool



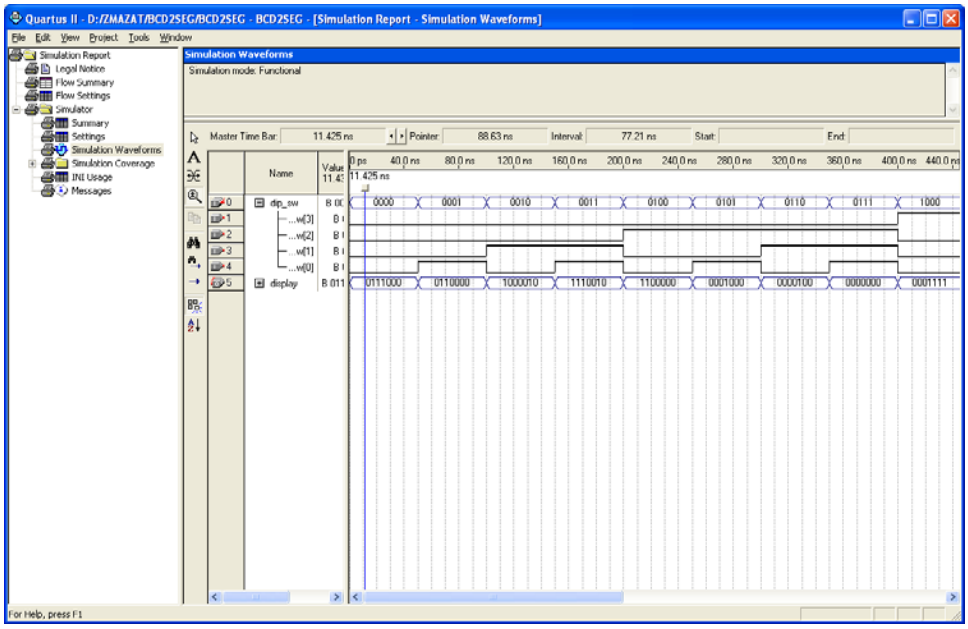
u.

v. Zvoliť: Simulation Mode: Functional

w. Klikúť: Generate Functional Simulation Netlist

x. Start

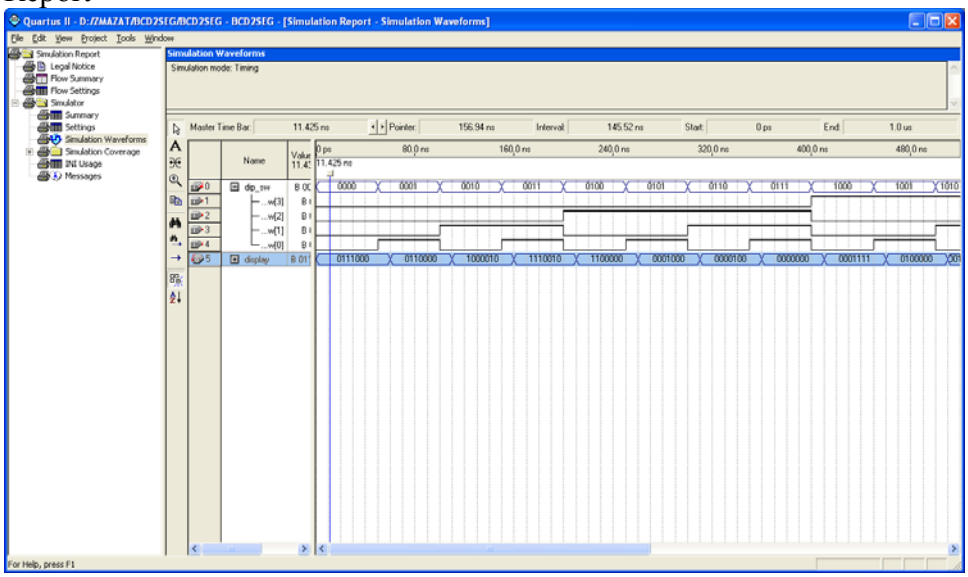
y. Report

z. 

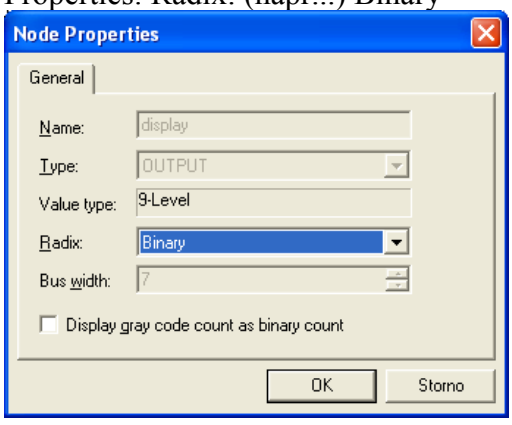
aa. Zvoliť: Simulation Mode: Timing

bb. Start

cc. Report

dd. 

ee. Radix zobrazenia signálov môžeme zmeniť pravým kliknutím na signál: Properties: Radix: (napr...) Binary

ff. 

5. **Napísanie VHDL test benchu**
    a. Pomocou VHDL test benchu vieme vnútiť (okrem iného) signály na vstup testovaného projektu. V tomto projekte sa zameriame len na generovanie testovacieho signálu priamo v kóde, avšak možnosti sú podstatne širšie. Testovaný projekt budeme označovať DUT (Design Under Test).
    b. Výpis kódu:

```
-- situmulus
library ieee;
use ieee.std_logic_1164.all; --kniznice

entity stim_BCD2SEG is -- v test benchoch entita NIKDY nema porty!!!
end stim_BCD2SEG;

architecture stim of stim_BCD2SEG is
-- Deklaracia komponentu, v tomto pripade to je DUT, syntax podobna
-- s entitou daneho komponentu. POROVNAT!!!
component BCD2SEG is                              --BCD2SEG je nazov entity
  port(
    dip_sw  : in std_logic_vector  (3 downto 0); -- vstupny port
    display : out std_logic_vector (6 downto 0)  -- vystupny port
  );
end component;

-- Deklaracia internych signalov
-- (tieto signaly budu implicitne zobrazene v grafickom vystupe ModelSimu)
  signal stim_data : std_logic_vector (3 downto 0);
  signal out_data  : std_logic_vector (6 downto 0);

begin
-- Instanciacia (pripojenie) deklarovaneho komponentu do dizajnu
  the_BCD2SEG: BCD2SEG
    port map (
      dip_sw => stim_data,
      display => out_data
    );
-- Generovanie vstupneho signalu pre DUT
-- process sa zvacsa pouziva pre popis sekvencnych obvodov alebo algoritmov
  stimulus: process -- stimulus je nazov processu,
    begin           -- zaciatok processu
      stim_data <= "1111"; wait for 50ns;
      stim_data <= "1110"; wait for 50ns;
      stim_data <= "1101"; wait for 50ns;
      stim_data <= "1100"; wait for 50ns;
      stim_data <= "1011"; wait for 50ns;
      stim_data <= "1010"; wait for 50ns;
      stim_data <= "1001"; wait for 50ns;
      stim_data <= "1000"; wait for 50ns;
      stim_data <= "0111"; wait for 50ns;
      stim_data <= "0110"; wait for 50ns;
      stim_data <= "0101"; wait for 50ns;
      stim_data <= "0100"; wait for 50ns;
      stim_data <= "0011"; wait for 50ns;
      stim_data <= "0010"; wait for 50ns;
      stim_data <= "0001"; wait for 50ns;
      stim_data <= "0000"; wait;
    end process stimulus;  -- koniec processu
end stim;
```

c. **Analýza kódu: entita.** Entita v test bechnoch nikdy neobsahuje vstupné a výstupné porty nakoľko všetky signály sú generované a vyhodnotené v rámci tejto entity.

```
entity stim_BCD2SEG is
end stim_BCD2SEG;
```

d. **Analýza kódu: component.** Toto je VHDL syntax pre deklaráciu ďalej používaného komponentu (kódu uloženého v inom súbore, ktorý reprezentuje určitý funkčný blok s príslušným vstupom a výstupom). Porovnajte s bodom 2.f – je to presne taká istá syntax, ako v popise entity funkčného bloku (BCD2SEG), len s takým rozdielom, že namiesto `entity` je použité kľúčové slovo `component`.

```
component BCD2SEG is        --BCD2SEG je nazov entity
  port(
    dip_sw  : in std_logic_vector  (3 downto 0); --vstup
    display : out std_logic_vector (6 downto 0)  --vystup
  );
end component;
```

e. **Analýza kódu: signály.** Ďalšia štandardná súčasť jazyka VHDL sú signály. Deklarujú sa kľúčovým slovom `signal`. Signál je možné predstaviť si ako drôt, ktorým sú prepojené dva funkčné bloky, alebo ako register. Syntax: za `signal` sa píše meno signálu, za dvojbodku typ signálu.

```
signal stim_data : std_logic_vector (3 downto 0);
signal out_data  : std_logic_vector (6 downto 0);
```

Ak používame signály v test benchi, budú implicitne zobrazené v grafickom výstupe ModelSimu.

f. **Analýza kódu: Inštanciácia.** Po slovensky „inštanciácia“ znamená pripojenie deklarovaného komponentu do dizajnu. Jeden komponent môžeme inštanciovať aj viackrát, avšak, s iným názvom. Takto pripojíme DUT do test benchu:

```
the_BCD2SEG: BCD2SEG
  port map (
    dip_sw => stim_data,
    display => out_data
  );
```

Syntax:

```
Názov_Inštanciácie: meno_komponentu
  port map (
    port_komponentu1 => deklarovaný_signál1,
    port_komponentu2 => deklarovaný_signál2,
    port_komponentu3 => deklarovaný_signál3
  );
```

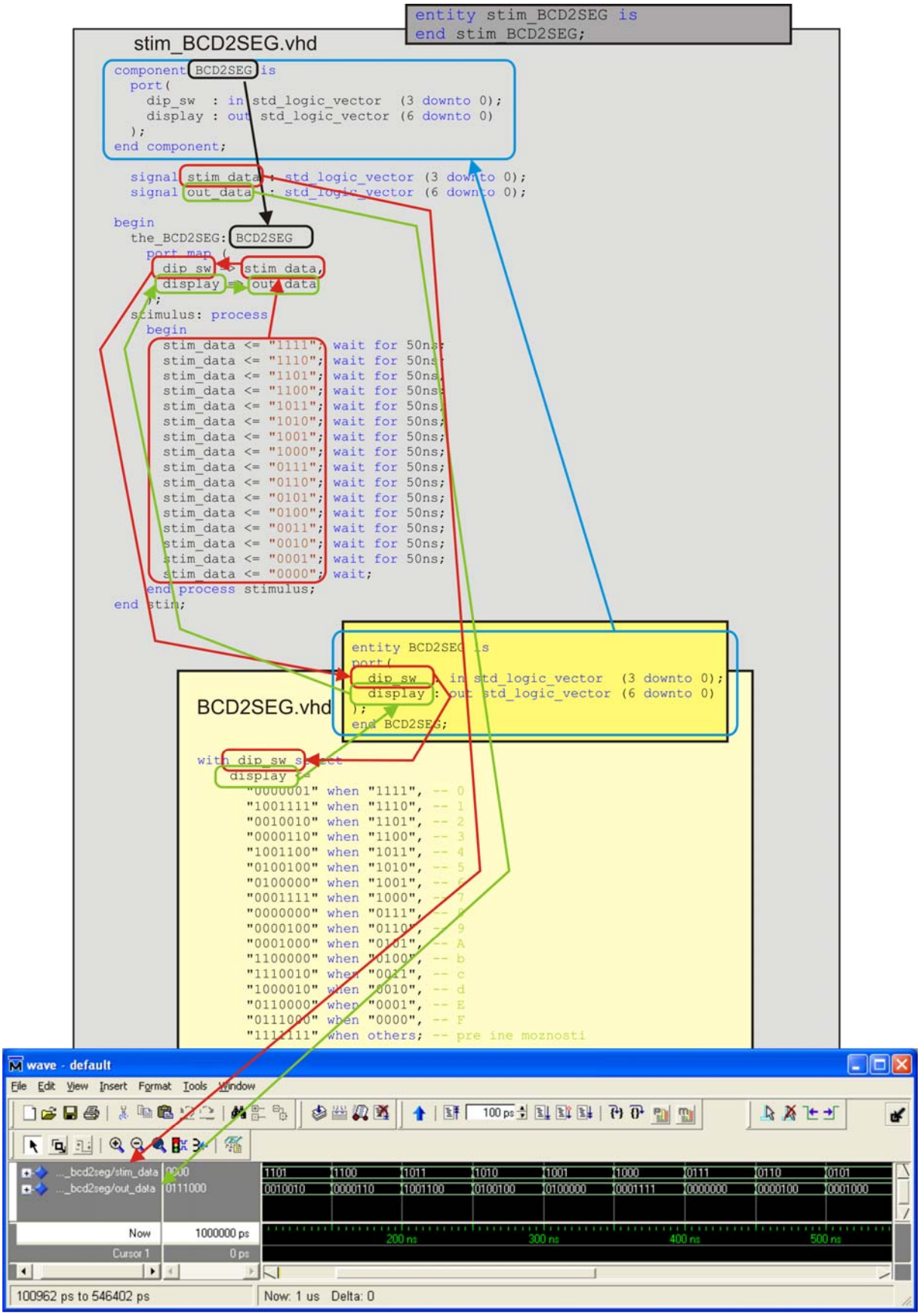
g. **Analýza kódu: Generovanie testovacieho signálu.** Testovací signál bude generovaný sekvenčne. Každých 50 ns sa pošle na vstup BCD2SEG určitá postupnosť bitových štvoríc. Výstup BCD2SEG bude zobrazený graficky v ModelSime. Celý algoritmus generovania testovacieho signálu je „obalený“ processom. `Process` je ďalší dôležitý prvok jazyka VHDL. Prevažne má spojitosť so sekvenčnou logikou v popise syntetizovateľného hardvéru alebo s generovaním signálov v test benchoch. Processy budú detailnejšie analyzované na cvičeniach, ktoré budú zamerané na popis sekvenčnej logiky.

```
stimulus: process   -- stimulus je nazov processu,
    begin           -- zaciatok processu
      stim_data <= "1111"; wait for 50ns;
      stim_data <= "1110"; wait for 50ns;
      stim_data <= "1101"; wait for 50ns;
      stim_data <= "1100"; wait for 50ns;
      stim_data <= "1011"; wait for 50ns;
      stim_data <= "1010"; wait for 50ns;
      stim_data <= "1001"; wait for 50ns;
      stim_data <= "1000"; wait for 50ns;
      stim_data <= "0111"; wait for 50ns;
      stim_data <= "0110"; wait for 50ns;
      stim_data <= "0101"; wait for 50ns;
      stim_data <= "0100"; wait for 50ns;
      stim_data <= "0011"; wait for 50ns;
      stim_data <= "0010"; wait for 50ns;
      stim_data <= "0001"; wait for 50ns;
      stim_data <= "0000"; wait;
    end process stimulus;  -- koniec processu
end stim;
```

h. Vytvorte nový súbor v Quartuse II, a hore popisovaný kód do neho skopírujte. Súbor pomenujte stim_BCD2SEG.vhd

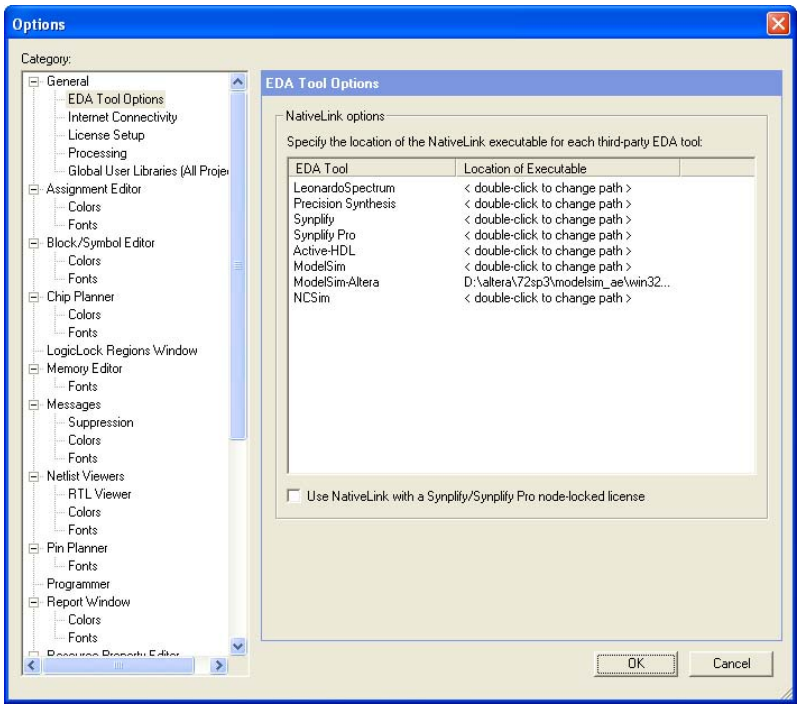
i. **Ako si predstaviť takéto prepojenie VHDL kódov?** Použijem analógiu s meraním VA charakteristiky žiarovky. Na to, aby sme odmerali VA charakteristiku žiarovky potrebujeme regulovateľný zdroj napätia, ampérmeter, voltmeter, káble a žiarovku. Pri rôznom napájacom napätí, bude tiecť žiarovkou rôzny prúd a žiarovka bude mať rôzny svit. Žiarovka = DUT = BCD2SEG.vhd; Zdroj napätia = stim_BCD2SEG.vhd; miera svitu žiarkovky a údaje z meracích prístrojov = grafický výstup z ModelSimu; voltmeter a ampérmeter = ModelSim; Meracie káble = `signal stim_data` a `signal out_data` (5.e); Svorky na pripojenie žiarovky (5.d) a (2.f); Gombíky pre nastavenie napätia na zdroji (+ obsluha) = `stimulus: process` (5.g); VA charakteristika žiarovky (resp. závislosť jej svitu od veľkosti prúdu) = pravdivostná tabuľka prevodníka.

j. **Ako si vizuálne interpretovať takéto prepojenie VHDL kódov?**

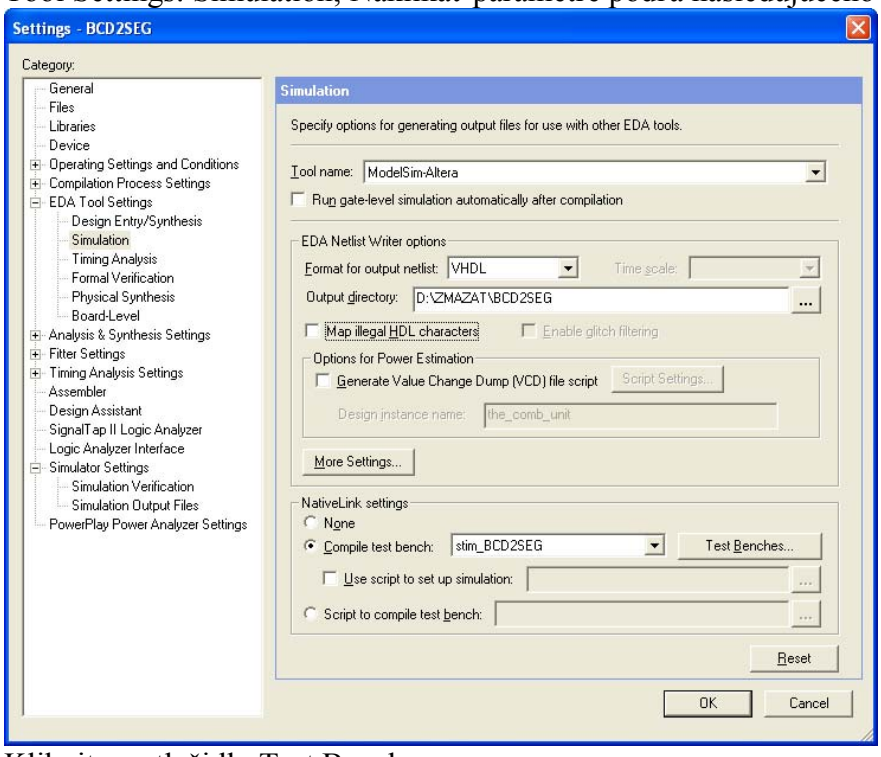

entity stim_BCD2SEG is
end stim_BCD2SEG;
stim_BCD2SEG.vhd
component BCD2SEG is
port(
dip_sw : in std_logic_vector (3 downto 0);
display : out std_logic_vector (6 downto 0)
);
end component;
signal stim_data : std_logic_vector (3 downto 0);
signal out_data : std_logic_vector (6 downto 0);
begin
the_BCD2SEG: BCD2SEG
port map (
dip_sw => stim_data,
display => out_data
);
stimulus: process
begin
stim_data <= "1111"; wait for 50ns;
stim_data <= "1110"; wait for 50ns;
stim_data <= "1101"; wait for 50ns;
stim_data <= "1100"; wait for 50ns;
stim_data <= "1011"; wait for 50ns;
stim_data <= "1010"; wait for 50ns;
stim_data <= "1001"; wait for 50ns;
stim_data <= "1000"; wait for 50ns;
stim_data <= "0111"; wait for 50ns;
stim_data <= "0110"; wait for 50ns;
stim_data <= "0101"; wait for 50ns;
stim_data <= "0100"; wait for 50ns;
stim_data <= "0011"; wait for 50ns;
stim_data <= "0010"; wait for 50ns;
stim_data <= "0001"; wait for 50ns;
stim_data <= "0000"; wait;
end process stimulus;
end stim;
entity BCD2SEG is
port(
dip_sw : in std_logic_vector (3 downto 0);
display : out std_logic_vector (6 downto 0)
);
end BCD2SEG;
BCD2SEG.vhd
with dip_sw select
display <=
"0000001" when "1111", -- 0
"1001111" when "1110", -- 1
"0010010" when "1101", -- 2
"0000110" when "1100", -- 3
"1001100" when "1011", -- 4
"0100100" when "1010", -- 5
"0100000" when "1001", -- 6
"0001111" when "1000", -- 7
"0000000" when "0111", -- 8
"0000100" when "0110", -- 9
"0001000" when "0101", -- A
"1100000" when "0100", -- b
"1110010" when "0011", -- c
"1000010" when "0010", -- d
"0110000" when "0001", -- E
"0111000" when "0000", -- F
"1111111" when others; -- pre ine moznosti
wave - default
File Edit View Insert Format Tools Window
100 ps
/bcd2seg/stim_data
/bcd2seg/out_data
0000
0111000
1101
1100
1011
1010
1001
1000
0111
0110
0101
0010010
0000110
1001100
0100100
0100000
0001111
0000000
0000100
0001000
Now
1000000 ps
Cursor 1
0 ps
200 ns
300 ns
400 ns
500 ns
100962 ps to 546402 ps
Now: 1 us Delta: 0

6. Konfigurácia prostredia Quartus II pre spúšťanie simulácie v ModelSime

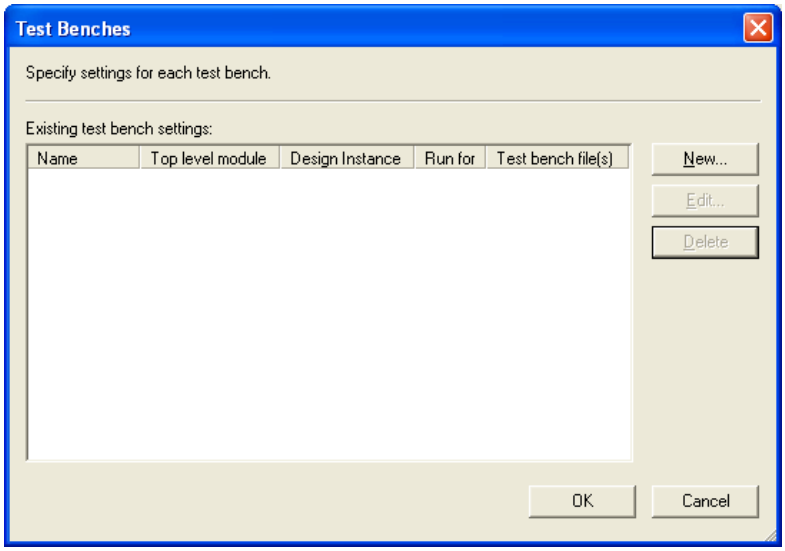
a. Zadáme cestu k modelsim.exe: Horné menu: Tool: Options: General: EDA Tool Options; Dvojklikom na ModelSim-Altera (pre domácu inštaláciu) alebo na ModelSim (pre školskú inštaláciu) a nasledným kliknutím na „...“ zadáme cestu k modelsim.exe

b. 

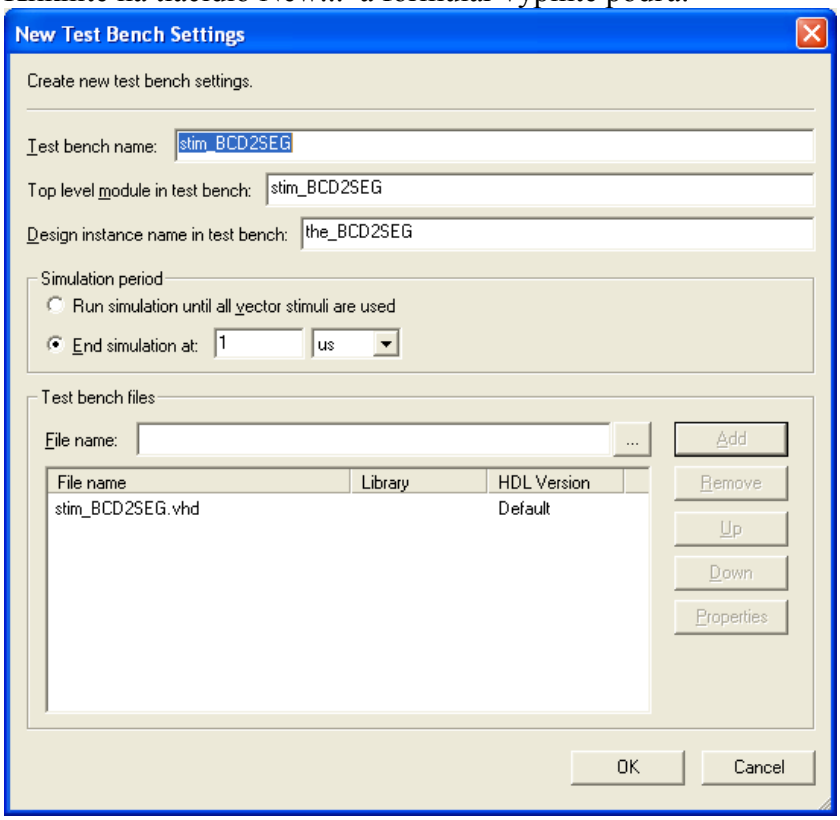
c. Nakonfigurujeme nastavenia pre simuláciu: Horné menu: Assignments: EDA Tool Settings: Simulation; Naklikať parametre podľa nasledujúceho obrázku:

d. 

e. Kliknite na tlačidlo Test Benches...

f. 

g. Kliknite na tlačidlo New... a formulár vyplňte podľa:

h. 

i. OK

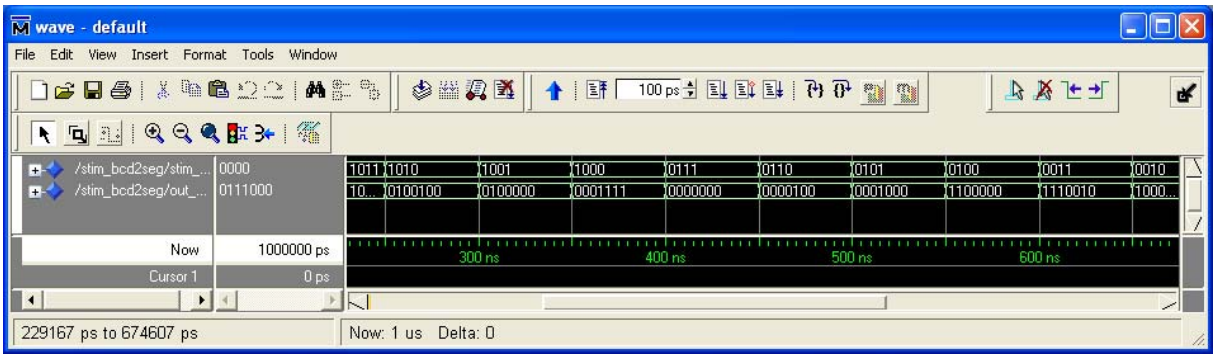
j. OK

k. OK

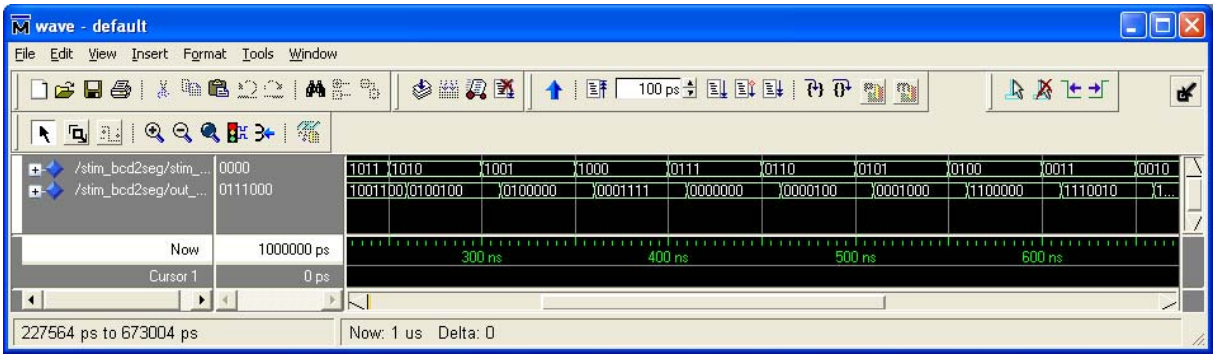
7. Funkčná a časová simulácia v ModelSime

a. Po každej zmene v projekte je nutné projekt uložiť prekompilovať pred simulovaním. Kompilácia: Horné menu: Processing: Start Compilation

b. Spustíme funkčnú simuláciu: Horné menu: Tools: EDA Simulation Tool: Run EDA RTL Simulation

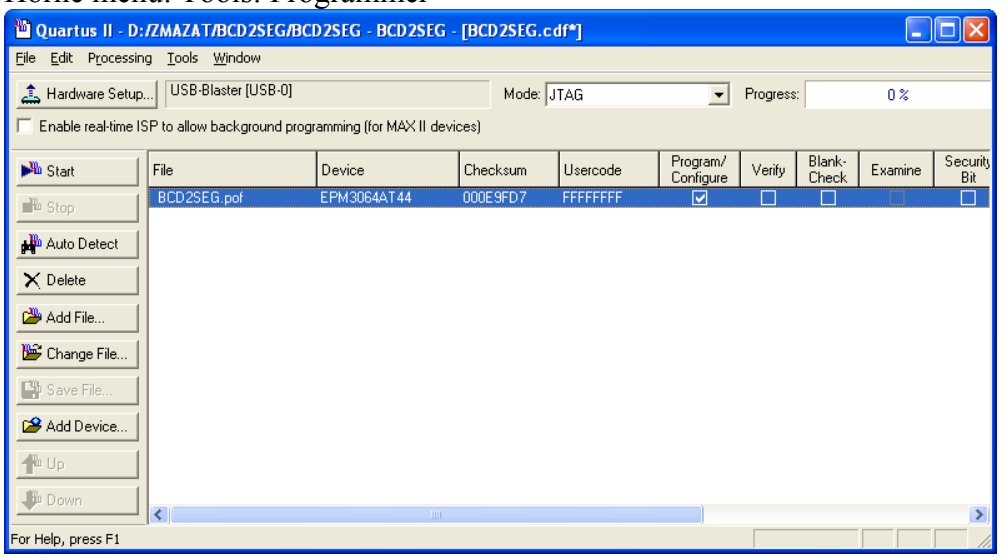


c. Spustíme časovú simuláciu: Horné menu: Tools: EDA Simulation Tool: Run EDA Gate Level Simulation



8. Konfigurácia dosky

a. Horné menu: Tools: Programmer



b.
c. Start
d. Odskúšať na doske, či dizajn funguje.