Obvody FPGA

Podklady na cvičenia

Publikácia neprešla redakčnou úpravou
Posledná úprava: september 2011 LaTeX

Obsah

Kapitola 1

Quartus II

1.1 Technické požiadavky

Quartus II ver. 9.1, minimálne hardvérové požiadavky:

- Procesor rady Pentium III alebo novší
- Farebný monitor s rozlíšením aspoň 1024 x 768 px
- Jeden alebo viac následujúcich vstupno-výstupných portov:
  - USB port pre USB-Blaster™ alebo MasterBlaster™ komunikačné káble.
  - Paralelný port pre ByteBlasterMV™ alebo ByteBlaster™ II zapisovacie káble.
- Operačná pamäť podľa [1] pre MAX® 3000A aspoň 512 MB, všeobecne odporúčaná aspoň 1GB
- Voľne miesto na disku aspoň 3,6 GB.

Quartus II má podporu pre následujúce operačné systémy:

- MS Windows Vista, Vista 64-bit, XP, XP 64-bit,
- Red Hat Enterprise Linux 4, 5 Suse Enterprise Linux 9, CentOS 4/5 (32/64 bit)

1.2 Získanie Quartus II

Vývojové prostredie Quartus II ver 9.1 je možné získať aj vo forme tzv. **web** verzie, ktorá je **zadarmo**. Táto verzia poskytuje všetky potrebné funkcie pre tvorbu projektu, kompiláciu, simulácie a programovanie a je voľne stiahnuteľná z domovskej stránky spoločnosti **Altera** (www.altera.com).
Na tejto stránke si vyberieme v hornej časti stranky položku **Suport** a v rozbalenom menu klikneme na **Downloads**. V pravej časti dalšej stranky klikneme **All Design Software** v sekcii **Archives**. Otvorí sa nám tabuľka so všetkými verziami softvérov od Altery. V riadku **November, 2009** s verziou Quartusu **9.1** klikneme na tlačidlo **SP2** v stĺpci **Quartus II Web Edition** (4. odkaz z ľava). Na ďalšej stranke v klikneme na **Download** v riadku prislúchajucemu **Quartus® II Web Edition Software Service Pack 2** podľa OS, ktorý používame. Otvorí sa nám stránka s formulárom na zadanie používateľského mena a hesla.
Ak konto ešte nemáme vytvorené, máme na výber dve možnosti. Buď si konto vytvoríme v spodnej časti stránky (**Create Your myAltera Account**), kde vyplníme emailovú adresu. Zobrazí sa nám tabuľka, v ktorej vyplníme povinne polia označene červenou hviezdičkou a pokračujeme kliknutím na tlačidlo **Create Account**. Zobrazi sa stránka s informáciou, že na zadaný mail boli poslané informácie o registrácií a kliknutím na **Continue** sa začne sťahovanie inštalačného programu.

Inou možnosťou je pokračovať bez registrácie a to zvolením voľby **Get One-Time Access**, kde tiež vyplníme povinne údaje (company, e-mail) a po úspešnom odoslaní údajov sa začne preberanie inštalačného súboru.

1.3 Inštalácia softweru Quartus II

Inštalácia spravidla nie je komplikovaná. Patrí medzi štandardné inštalácie a nevyžaduje si nejaké veľké technické znalosti. Po spustení inštalačného súboru sa nám zobrazí úvodné okno inštalácie s názvom: **Welcome to the Quartus II Install Shield Wizard**. Klikneme na Next. V ďalšom okne sa nám zobrazia licenčné podmienky. Ak súhlasíme a chceme pokračovať v inštalácii, musíme zvoliť možnosť **I accept the termsof the license agreement**. V nasledujúcom okne musíme vyplniť položky ako **User name** (užívateľské meno) a **Company Name** (meno spoločnosti). V ďalších dvoch oknách musíme zvoliť kam sa program inštalovať. Nasleduje zvolenie či má inštalovať všetko (**Complete**) alebo voliteľná inštalácia (**Custom**). Vhodné je nechať nainštalovať plnú verziu. V poslednom kroku zvolíme už len programovú skupinu a v celkom poslednom okne sa nám zobrazia nami nastavené údaje. Ak s nimi súhlasíme, klikneme na **Next** a spustíme inštaláciu. Ak nie, tak tlačidlom **Back** sa vrátime a zmeníme nastavenia. Po skopírovaní všetkých súborov sa nám zobrazí okno, či bol priebeh inštalácie úspešný, alebo nie. Potom nám program položí otázku, či má vytvoriť odkaz na plochu operačného systému. Nakoniec sa nám zobrazí už len záverečné okno inštalácie.

Kapitola 2

Základné logické hradla NOR, OR, AND, NAND

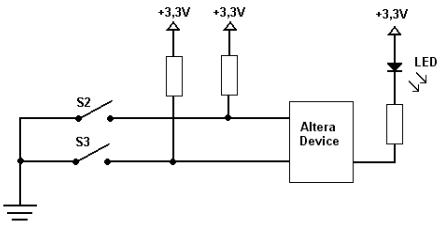
Projekt ukazuje návrh jednoduchého návrhu „krok za krokom“ vo vývojom systéme Quartus II. Ukazuje postup vytvorenia nového projektu, nových súborov návrhu, kompiláciu, simuláciu a programovanie. Projekt je vytvorený vo vývojom systéme Quartus II ver. 9.0, ale vo všeobecnosti platí aj pre staršie verzie a z časti aj pre novšie.

2.1 Zadanie 1.

V prostredí softvéru Quartus II realizujte rozsvietenie bodky pravej sedemsegmentovky na vývojovej doske CPLD_KIT stlačením jedného z tlačidiel (S2 alebo S3). Návrh realizujte v grafickom editore. Čísla a funkcie potrebných pinov sú uvedené v tabuľke č. 2.1.

Tabuľka 2.1: *Čísla pinov*

Meno pinu	Typ pinu	Pin	Funkcia pinu
vstup1	input	27	Tlačidlo S2 (0 = stlačené tlačidlo)
vstup2	input	28	Tlačidlo S3 (0 = stlačené tlačidlo)
vystup	output	10	Desatinná bodka 7-segmentu (0 = LED ON, 1 = LED OFF)



Obr. 2.1: *Schéma zapojenia S2, S3 a bodky sedemsegmentovky na doske CPLD_KIT*

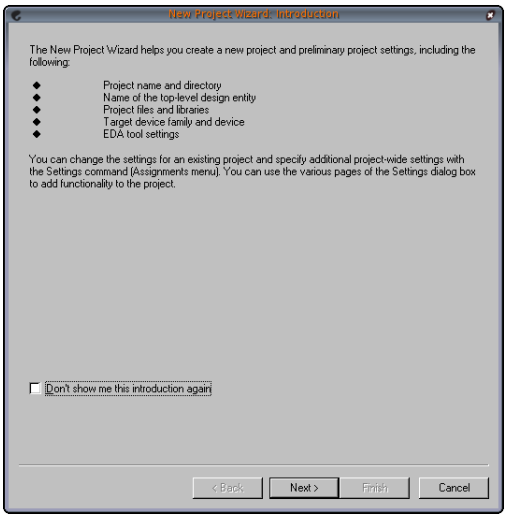
2.2 Riešenie

Úlohu môžeme riešiť návrhom obvodu, ktorý zmení logickú hodnotu na desatinnej bodke LED ak je stlačené jedno, alebo druhé tlačidlo. Takýto obvod môžeme zrealizovať použitím obvodu s logikou NOR,

ktorý má dva vstupy (S2 a S3) a jeden výstup (dp(9)). Tlačidlá S2, S3 a segment Led dp(9) na vývojovej doske CPLD_KIT sú zapojené podľa obr. 2.1.

2.2.1 Vytvorenie nového projektu

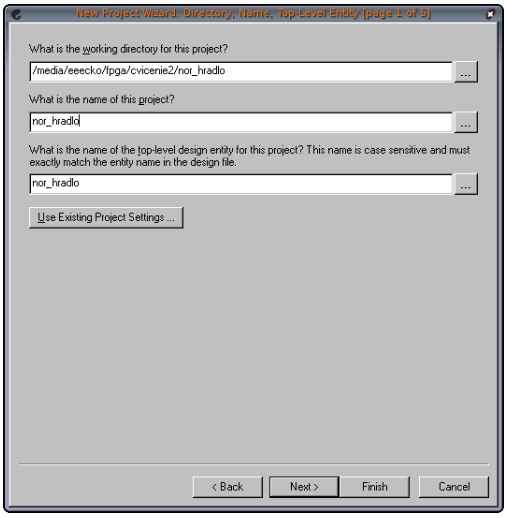
Nový projekt vytvoríme pomocou príkazu New Project Wizard z File menu. Ako prvé sa objaví úvodné okno (obr.2.2.). Kliknutím na tlačidlo Next sa objaví prvé okno (...[page 1 of 5], obr.2.3.), kde sa definuje miesto uloženia a meno projektu a entity.



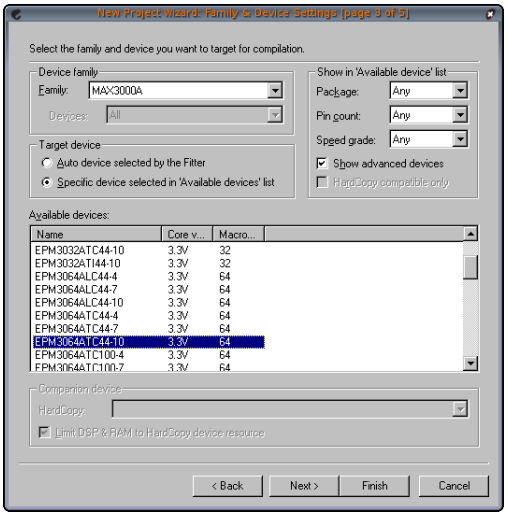
Obr. 2.2: *Úvodné okno New Project Wizard*

Kliknutím na tlačidlo **...** prvého riadku, sa objaví okno, kde si môžeme vybrať adresár, do ktorého uložíme projekt a do ktorého budú ukladané všetky vytvárané súbory, alebo si môžeme vytvoriť nový adresár. Pre náš projekt si vytvoríme adresár nor_hradlo. V druhom riadku definujeme meno projektu a v treťom meno najvyššej úrovne entity. Meno projektu a entity môže, ale nemusí byť také isté (väčšinou sa pre jednoduchosť a prehľadnosť volia rovnaké). Meno entity a projektu musí byť jeden výraz bez medzier a nesmie byť kľučovým slovom (nesmie to byť napríklad "nor hradloälebo AND, NOR, BEGIN a podobne). Pre náš projekt zadefinujeme rovnaké meno projektu aj meno entity aké je meno adresára – nor_hradlo. Ak sme v tomto okne všetko zadefinovali klikneme na tlačidlo **Next**.

Nasledujúce okno, ktoré sa otvorí slúži na pridanie súborov do projektu. Napríklad, ak v nejakom inom projekte je vytvorený VHDL kód, ktorý budeme používať aj v novom projekte, tak si ho tu môžeme pridať. Pridanie súborov do projektu je samozrejme možné aj neskôr. V tomto projekte nebudeme pridávať žiadny súbor – klikneme na Next.
Objaví sa tretie okno príkazu New project Wizard (obr.2.4.), v ktorom si definujeme rodinu obvodov, v našom prípade je to rodina MAX3000A. V spodnej časti tohoto okna vyberieme presne ten typ obvodu, s ktorým chceme pracovať. V našom prípade je to súčiastka EPM3064ATC44-10.
Kliknutím na tlačidlo Next sme otvorili okno, v ktorom si môžeme nastaviť nástroje EDA. Tu tentokrát nebudeme nič nastavovať.
V poslednom okne - piaté okno príkazu New Project Wizard (obr. 2.5.) sú zobrazené všetky voľby a nastavenia, ktoré sme v predchádzajúcich krokoch vykonali. Ak je všetko v súlade s našimi požiadavkami, klikneme na tlačidlo **Finish**. Ak zistíme, že niektoré nastavenie nevyhovuje, vrátime sa tlačidlom Back a zmeníme nastavenia podľa požiadaviek.



Obr. 2.3: *Okno New Project Wizard, definovanie pracovného adresára, názov projektu a hlavnej entity*



Obr. 2.4: *Okno New Project Wizard, nastavenie rodiny a konkrétny typ súčiastky*

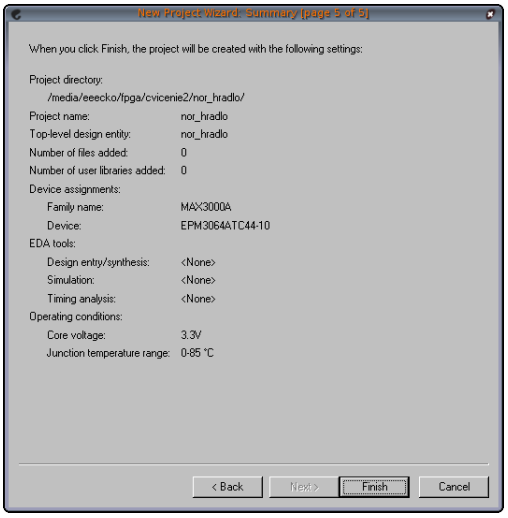
2.2.2 Vytvorenie návrhu prostredníctvom grafického editora (súbor - .bdf)

Tento súbor vytvoríme vykonaním nasledujúcich krokov: Vyberieme položku New z File menu a v sekcií Design Files si zvolíme Block DiagramSchematic File s klikneme na **OK**(obr. 2.6.). Otvorí sa okno blokového editora. Vyberieme položku Save As z File menu a vyberieme adresár nor_hradlo a uložíme tam náš súbor, ktorý nazveme nor_hradlo.bdf. Pod riadkom, kde sa definuje meno projektu zaškrtneme **Add file to curent project** (pridať súbor do vlastného projektu). Potvrdíme tlačidlom Save(obr. 2.7.). Takto sme uložili a aj vložili súbor do projektu.

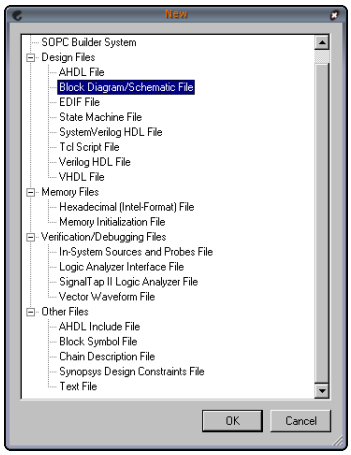
2.2.3 Vytvorenie (nakreslenie) schémy

Schému vytvoríme podľa nasledujúcich krokov:

- Na panely nástrojov klikneme na **Symbol Tool**
- V okne, ktoré sa objaví vyberieme $Quartus_ROOT => primitives => logic (obr. 2.8.)

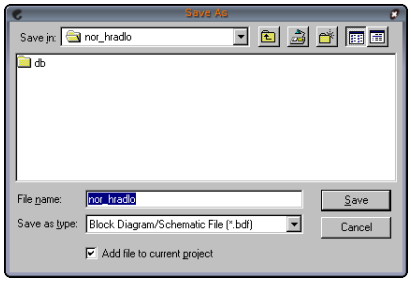


Obr. 2.5: *Okno New Project Wizard, zobrazenie volieb a nastavení projektu*



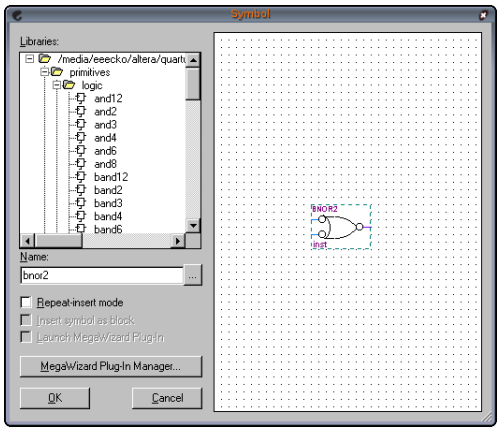
Obr. 2.6: *Otvorenie nového súboru pre kreslenie blokovej schémy*

- Zvolíme súčiastku **BNOR2** a potvrdíme stlačením na **OK**
- Symbol na ploche umiestníme na požadované miesto pohybom myšou a vložíme ho kliknutím ľavého tlačidla.
- Znovu klikneme na **Symbol tool**
- V okne vyberieme $Quartus_ROOT => primitives => pin
- Vyberieme pin **output** (výstupný pin), **OK**, umiestníme na požadované miesto
- Podobne postupujeme pri výbere pin-ov **input** (vstupné piny), tieto potrebujeme **dva**
- Spojíme symboly vodičmi, tak že sa s myšou priblížime k vstupu alebo výstupu symbolu a ak sa kurzor myši zmení na krížik, tak klikneme a držíme ľavé tlačidlo myši, približime sa k tomu výstupu alebo vstupu, ktorý chceme spojiť a ľavé tlačidlo myši pustíme. Tým požadované vstupy alebo výstupy spojíme.
- Názvy a hodnotu k pinom priradíme tak, že klikneme na pin pravým tlačidlom myši, zobrazí sa tabuľka, z ktorej vyberieme položku Properties a kliknutím na ňu sa zobrazí okno. To isté dosiahneme dvojitým kliknutím na symbol pinu.

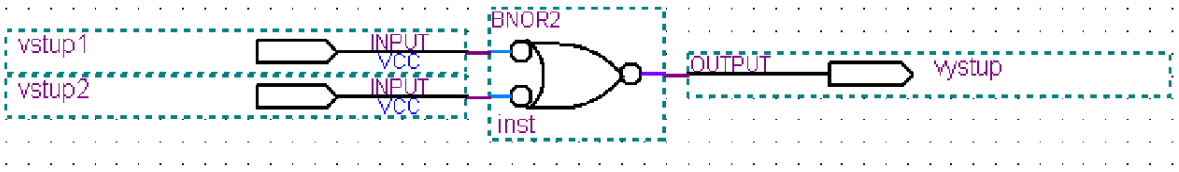


Obr. 2.7: *Uloženie súboru pre kreslenie blokovej schémy a vloženie do projektu*

- V riadku **Pin name(s):** zmeníme názvy pinov napríklad na **vstup1**, **vstup2** a **vystup**. Riadok **Default Value** ponecháme na hodnote VCC.
  POZOR!!! Mená pinov musia byť **jeden** výraz bez medzery, a nesmú byť kľúčovým slovom (teda napríklad nesmú to byť "druhy vstup", alebo AND, NAND, BEGIN...atď.)
- Výsledná schéma by mala vyzerať podobne, ako na obrázku 2.9 .
- Nakoniec vytvorenú schému uložíme (**File** menu => **Save**, alebo klavesovou skratkou **CTRL+S**)



Obr. 2.8: *Vloženie symbolu hradla nor do schémy*



Obr. 2.9: *Výsledná schéma riešenia pomocou hradla NOR v blokovom editore*

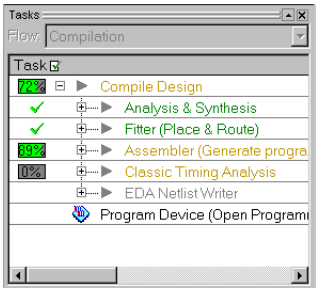
2.2.4 Kompilácia a priradenie pinov

V softvéry Quartus II obsahuje kompilátor niekoľko modulov, ktoré spracovávajú projekt. Sú to:

- Analysis Syntesis (Analýza a syntéza)
- Fitter (prispôsobenie)

- Assembler (Asemblér)
- Timing Analyzer (Časová analýza)

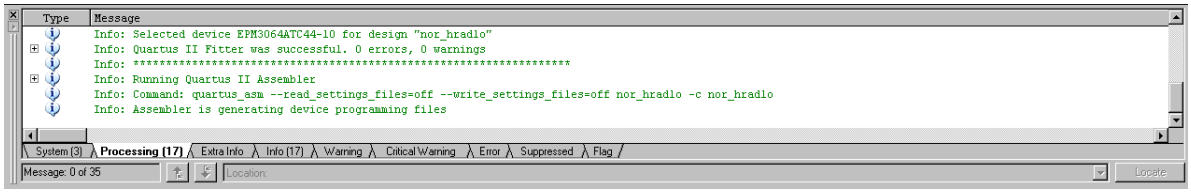
V Quartus II sa daju príkazom **Processing** menu => **Start** spúšťať jednotlivé moduly aj samostatne. (napr.: Procesing menu – Start – Start Fitting). Vo verzii WEB to nieje možné, je možné spustiť len všetky moduly naraz.



Obr. 2.10: *Status Window*

Flow Status	Successful - Wed Sep 15 16:38:16 2010
Quartus II Version	9.0 Build 132 02/25/2009 SJ Full Version
Revision Name	nor_hradlo
Top-level Entity Name	nor_hradlo
Family	MAX3000A
Device	EPM3064ATC44-10
Timing Models	Final
Met timing requirements	Yes
Total macrocells	1 / 64 (2 %)
Total pins	7 / 34 (21 %)

Obr. 2.11: *Compilation Report*



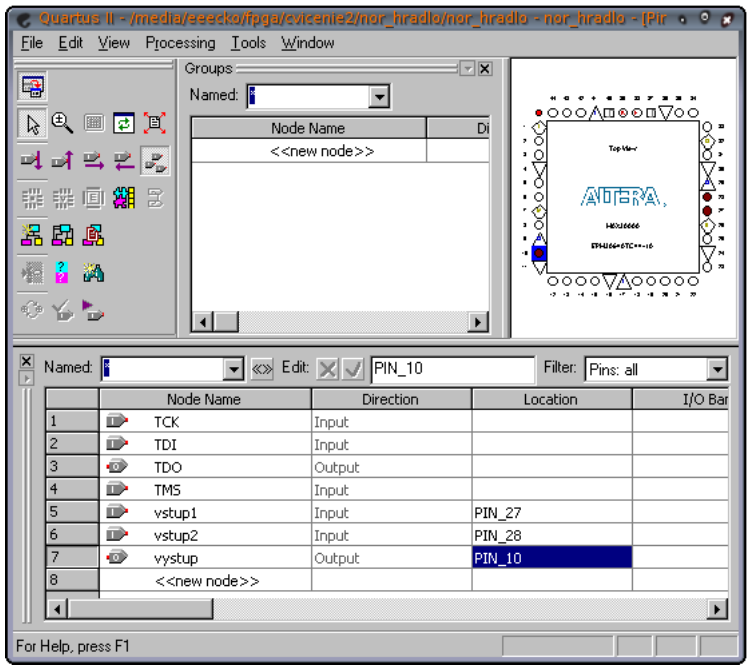
Obr. 2.12: *Message Windov*

Kompletná kompilácia sa spúšťa cez **Processing** menu => **Start Compilation** alebo pomocou tlačidla na hornej lište v tvare trojuholníka fialovej farby s názvom **Start Compilation**
Po spustení kompilácie sa automaticky zobrazia tieto okná:

- **Status Windov:** v tomto okne môžeme sledovať priebeh kompilácie. Môžeme tu pozorovať čas, za ktorý jednotlivé moduly spracovali projekt, aj celkový čas kompilácie. Okrem toho je tu znázornená aj miera úspešnosti celkovej kompilácie a jednotlivých modulov v percentách.(obr. 2.10.)
- **Compilation Report:** Toto okno je rozdelené na dve časti: ľavé okno a pravé okno. V ľavom okne sú uvedené výsledky kompilácie, usporiadané v stromovej štruktúre do adresárov, ktoré obsahujú súbory podľa toho, akým modulom boli generované. V pravom okne sa zobrazujú najdôležitejšie výsledky kompilácie a obsah jednotlivých súborov (obr. 2.11.).
- **Message Windov:** V tomto okne sú zobrazované všetky správy, ktoré sú počas kompilácie generované. Zobrazené správy môžu byť: informačné, varovné a chybové. Ak sa objavia chybové správy,

Quartus II umožňuje lokalizovať túto chybu priamo v súbore návrhu (dvojklik ľavým tlačidlom myši na správu). Okrem toho ponúka aj pomoc pri odstránení chyby - kliknutím na správu pravým tlačidlom sa zobrazí tabuľka, v ktorej vyberieme položku Help(obr. 2.12.).

Po úspešnej kompilácii (= žiadná chybova hláška, možno niekoľko warningov) pristupíme k priradeniu vstupných a výstupných pinov k jednotlivým vývodom obvodu. Otvoríme **Pins** z **Assignments** menu, kde po úspešnej kompilácii vidíme naše vstupné a výstupné piny (obr. 2.13.). V stĺpci **Node Name** sú uvedené názvy pinov, v stĺpci **Direction** je uvedený typ pinu, či sa jedná o vstupný, výstupný, vstupno-výstupný pin. My sa sústredíme na stĺpec s názvom **Location**, kde ku jednotlivým pinom priradíme čísla vývodov, tak ako je to na obrázku a potvrdíme tlačidlom **Enter**. Po priradení pinov zavrieme Pin Planner Ďalším krokom



Obr. 2.13: *Nastavenie pinov v Pin Planner-y*

je nastavenie nevyužitých pinov, ktoré niesu definované v projekte. Toto nastavenie prevedieme tak, že klikneme na **Device** v menu **Assignments** => **Device and Pin Options...** => **Unused Pins** a voľbu **Reserve all unused pins** nastavíme na **As input tri-stated**. Potvrdíme dvakrát **OK**. Znova spustíme kompiláciu, aby sa zmeny zakomponovali do projektu. Po úspešnej kompilácii môžeme projekt simulovať aj konfigurovať.

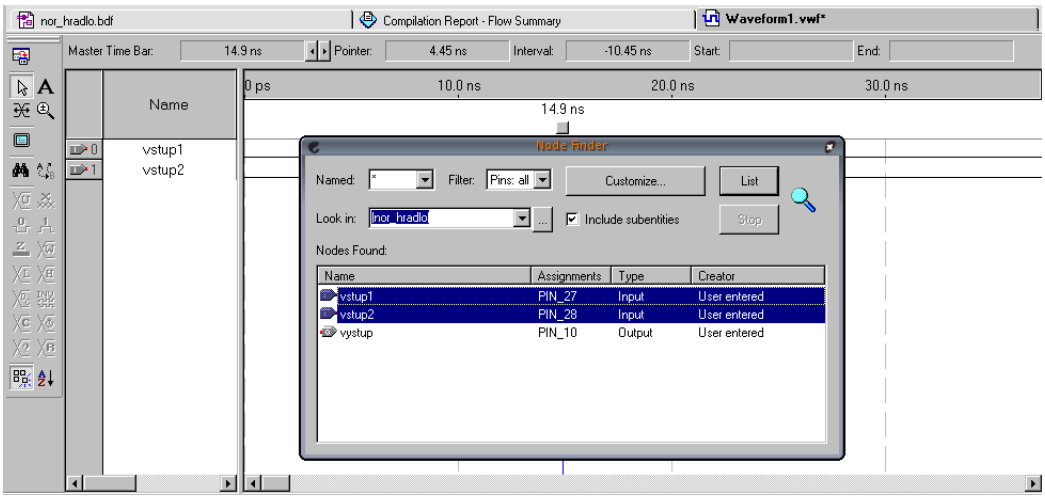
2.2.5 Simulácia projektu

Po úspešnej kompilácii sa môže projekt simulovať. Simuláciu vykonáme podľa nasledujúcich krokov:

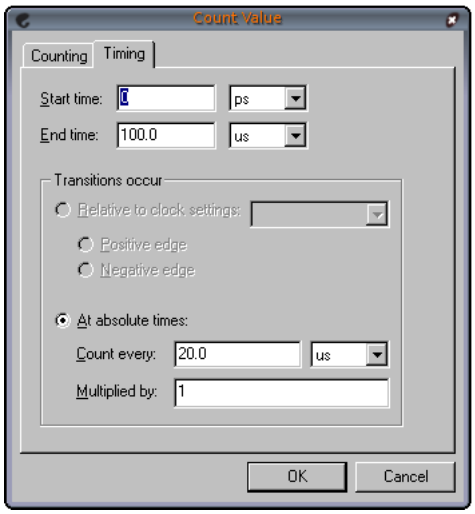
- Vytvoríme vektorový súbor priebehu signálu – **File** menu => **New** =>**Vector Waveform file**(.vwf),
- Pomocou príkazu **Save As** z **File** menu uložíme tento súbor ako nor_hradlo.vwf
- Pomocou nástroja **Node Finder** (**View** => **Utility Windows** => **Node Finder**) vložíme do tohto súboru všetký vstupné signáli - Stlačíme **List**, označíme všetký vstupné signáli, v našom prípade **vstup1** a **vstup2**, kliknutím a pretiahnutím do okna s editorom vektorového súboru (obr. 2.14.).
- Nastavíme si koncový čas simulácie: **Edit** => **End Time**, tam nastavíme hodnotu **time** napr. na **100**μ**s**
- Klikneme pravým tlačidlom myši na pin **vstup1**, potom sa objaví tabuľka, kde vyberieme: **Value** => **Count Value**. Zobrazí sa okno (obr. 2.15.), kde zadefinujeme v záložke **Timing: End Time:** 100μs a

Cout Every: 20μs. Pre pin **vstup2** zadefinujeme: **End Time:** 100μs a **Count Every:** 10μs. (Môžeme zadefinovať aj iné hodnoty).

- Uložíme pomocou **Save** z **File** menu alebo prostredníctvom ikony na panely nástrojov a spustíme simuláciu príkazom **Start Simulation** z **Proccesing** menu, alebo prostredníctvom ikony na panely nástrojov (modry trojuholník s).



Obr. 2.14: *Okno Node Finder-u a s označenými vstupnými pinmi*

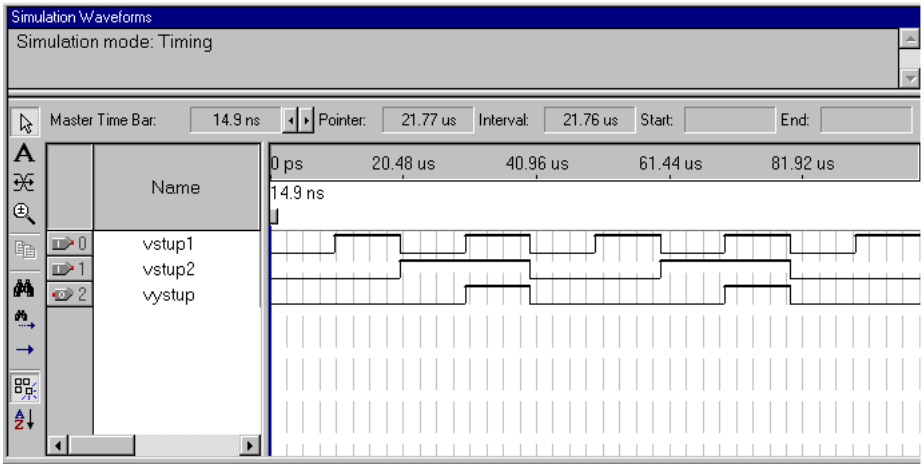


Obr. 2.15: *Nastavenie vstupného simulačného vektoru*

Z výsledkov simulácie (obr. 2.16.) môžeme vidieť, kedy má byť bodka 7-segmentovky rozsvietená a kedy zhasnutá. Tlačidlá vstup1 (S2) a vstup2 (S3), keď sú stlačené majú na výstupe logickú úroveň 0. Tiež bodka 7-segmentovky je zasvietená vtedy, keď je na ňu privedená logická úroveň 0. Preto pri vyhodnocovaní priebehov vo waveform editore musíme uvažovať s touto skutočnosťou.

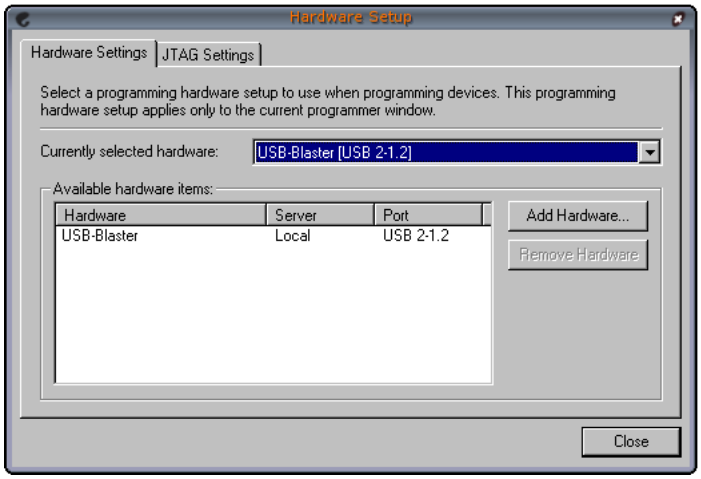
2.2.6 Konfigurácia

Po úspešnej kompilácii môžeme hneď programovať obvod a to pomocou **Tools** menu => **Programmer**, alebo **Programmer** ikonka na hornej liste nástrojov. USB-Blaster musí byť pripojený k PC aj k doske CPLD_KIT



Obr. 2.16: *Výsledne okno simulácie*

a zároveň musí mať CPLD_KIT zapnuté napájanie.
Najprv otvoríme **Hardware Setup** (obr. 2.17.) kde nastavíme **Currently selected hardware** a vyberieme **USB-Blaster**. Zavrieme toto okno - **Close**. V tomto okne zaškrtneme políčko **Program/Configure** a tlačidlom **Start** nahrajeme konfiguráciu do obvodu (obr. 2.18.).

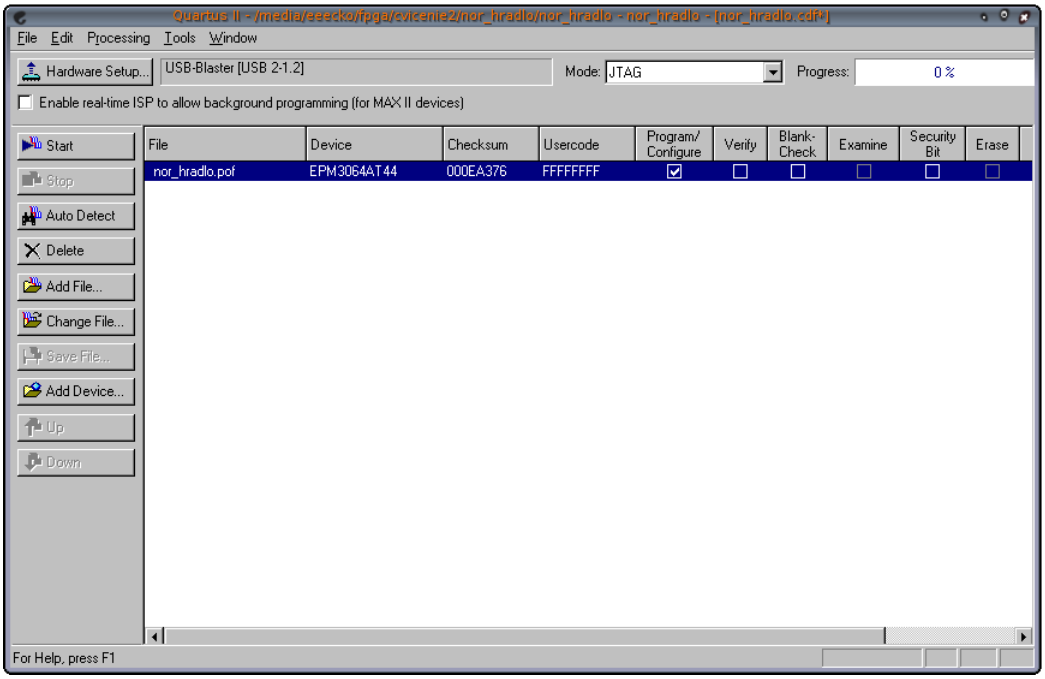


Obr. 2.17: *Nastavenie komunikácie s USB-Blastrom*

2.2.7 Archivácia

V procese tvorby, kompilácie, simulácie a konfigurovania projektu do súčiastky, návrhový softvér Quartus II vytvára veľké množstvo súborov a z toho dôvodu to môže robiť problémy pri prenose projektu na inu pracovnú stanicu (iný počítač). Preto Quartus II obsahuje takzvanú archiváciu projektu, ktorá vloží do jedného súboru len tie najpotrebnejšie informácie k obnoveniu projektu po prenose. Po obnovení projektu a vykonaní kompilácie, simulácií dostaneme pôvodné súbory.

Pri archivácií postupujeme tak, že v menu **Project** vyberieme **Archive Project**.Do riadku Archive File Name zadávame meno archivačného súboru (nor_hradlo), poprípade pomocou tlačidla **...** vyberieme aj cestu kde má byť súbor uložený. Po kliknutí na tlačidlo **Advanced** možme definovať ktoré súčasti projektu chceme archivovať. V našom prípade stačí, ak budú zaškrtnuté voľby **Project source and settings**



Obr. 2.18: *Okno konfigurácie obvodu*

files a **Automatically detected source files**. Potvrdíme dvakrát **OK** Archivačný súbor má príponu . qar. Po stlačení OK bude vygenerovaný archivačný súbor nor_hradlo.qar.

Obnovenie z archívu spočíva v otvorení tohoto súboru v Quartuse II (**File** => **Open**, nastaviť **Files of type** na **Quartus II archive files (*.qar)** => vybrať súbor **nor_hradlo.qar** => **OK**) a definovanie cesty, kde sa má tento projekt rozbaliť.

2.3 Zadanie 2.

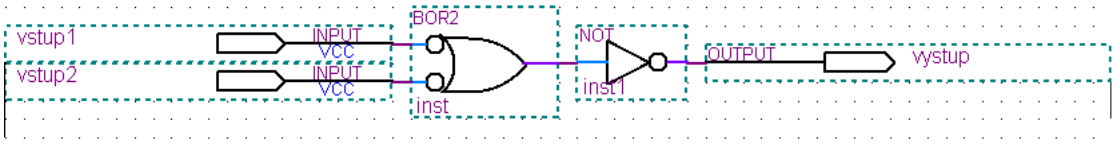
V prostredí softvéru Quartus II realizujte rozsvietenie bodky na sedem segmentovej LED na vývojovej doske CPLD_KIT stlačením jedného z tlačidiel, použitím hradiel OR, AND, NAND. Návrh realizujte v grafickom editore.

2.4 Riešenie

Postup riešenia bude taký istý ako pre hradlo NOR. Tiež platí tá istá tabuľka pinov ako pri hradle NOR. Opäť si otvoríme nový projekt, podľa postupu uvedeného pre hradlo NOR. Rozdiel je iba v tom, že pri výbere prvku si nevyberieme hradlo NOR, ale hradlo OR, AND alebo NAND. Priradenie pinov, kompiláciu, simuláciu a konfiguráciu, taktiež vykonáme podľa predchádzajúceho postupu.

2.4.1 OR

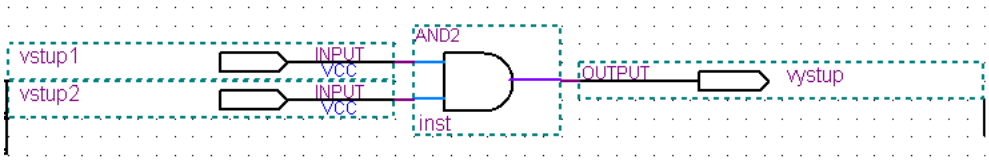
Výsledná schéma s použitím hradla OR je na obrázku 2.19. V tomto prípade sme museli dať na výstup ešte invertor, lebo ak by sme ho tam nedali, dostali by sme presne opačný efekt, Led by bola stále rozsvietená a zhasla by pri stlačení tlačidla S2 alebo S3.



Obr. 2.19: *Schéma riešenia s hradlom OR*

2.4.2 AND

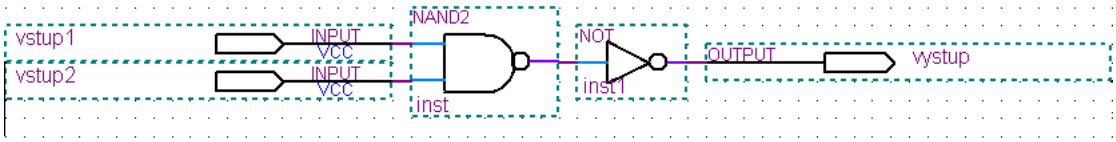
Riešenie zadania tvorené dvojvstupovým hradlom AND, dvoma vstupmi (S2, S3) a jedným výstupom - bodka 7-segmentovky (obr. 2.20.)



Obr. 2.20: *Schéma riešenia s hradlom AND*

2.4.3 NAND

Výsledné zapojenie je tvorené jedným dvojvstupovým hradlom NAND, jedným invertorom, dvoma vstupmi (S2, S3) a jedným výstupom - bodka 7-segmentovky (obr. 2.21.)



Obr. 2.21: *Schéma riešenia s hradlom NAND*

Kapitola 3

Kombinačné obvody

3.1 Teória

Kombinačné obvody - sú logické obvody, ktorých výstup závisí len od kombinácie vstupov v danom časovom okamihu (obvody ktoré nemajú pamäť). Medzi takéto obvody môžeme zaradiť prevodníky kódov, kódery a dekódery, multiplexory a demultiplexory, kompáratory binárnych čísel, generátory parity, sčítačky.
BCD kód(**B**inary **C**oded **D**ecimal) - je to štvorbitový, váhový, binárny kód, kde každej desiatkovej číslici od 0 – 9 zodpovedajú štvorbitové binárne kódové slová. Zo 16 možných stavov sa využívajú 10. 6 nežiadúcich sa vylúci pomocou korelacných faktorov.
Gray – ov kód - je to n- bitový lineárny kód, u ktorého sa každá susedná kódová kombinácia líši len v 1 bite. Tým sa odstráni nebezpečenstvo viacnásobných zmien, čo je výhoda voči BCD kódu. Využíva sa pri číslicovom spracovaní obrazov. Dosahuje vyššiu kompresiu.
Kód +3 ku BCD - je 4 bitový, binárny, doplnkový kód. Vytvorí sa pripocítaním čísla 3 k desiatkovému číslu a vyjadrením v BCD kóde.
Syntézu kombinacných obvodov môžeme popísat pomocou nasledovných bodov:

- Podla činnosti kombinačného obvodu zostavíme pravdivostnú tabuľku, v ktorej určíme vstupy a výstupy
- Z pravdivostnej tabulky, alebo karnaughovej mapy určíme algebraické vyjadrenie výstupnej funkcie
- Na základe algebraického vyjadrenia výstupnej funkcie realizujeme navrhnutý obvod

3.2 Zadanie 1.

Navrhnite prevodník z Grayovho kódu do kódu +3 ku BCD (0-9). Návrh realizujte pomocou hradiel NAND v grafickom editore vývojového prostredia Quartus II.

3.3 Riešenie

3.3.1 Rozbor

Po syntéze zadaného príkladu, teda po zostavení pravdivostnej tabuľky, karnaughových máp, určení algebraických funkcií nakreslime schému logického obvodu. Tú potom realizujeme v grafickom editore Quartus II. Ako vstupy použijeme prepínače S4. 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 MSD sedem segmentovky a segmenty e, c LSD sedem segmentovky. Displej reaguje na logickú úroveň „0“, preto musíme na každý výstup zaradiť jedno hradlo NOT.

Tabuľka 3.1: *Čísla pinov*

Meno pinu	Typ pinu	Pin	Funkcia pinu
G1	Vstup	31	Prepínač 4 (1 = poloha dole, 0 = poloha hore)
G2	Vstup	33	Prepínač 3 (1 = poloha dole, 0 = poloha hore)
G3	Vstup	34	Prepínač 2 (1 = poloha dole, 0 = poloha hore)
G4	Vstup	35	Prepínač 1 (1 = poloha dole, 0 = poloha hore)
A	Výstup	5	Segment c LSD (0=LED ON, 1=LED OFF)
B	Výstup	42	Segment e LSD (0=LED ON, 1=LED OFF)
C	Výstup	20	Segment c MSD (0=LED ON, 1=LED OFF)
D	Výstup	22	Segment e MSD (0=LED ON, 1=LED OFF)

3.3.2 Syntéza

Na začiatku návrhu podla činnosti prevodníka zostavíme pravdivostnú tabuľku, v ktorej určíme vstupy a výstupy.

Tabuľka 3.2: *Pravdivostná tabuľka*

dekadický	**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

Z pravdivostnej tabuľky zostavíme pre jednotlivé výstupné funkcie Karnaughove mapy(obrázok 3.1). Z Karnaughových máp určíme algebraické funkcie a zapíšeme ich pomocou Boolovej algebry do UDNF (úplná disjunktná normálna forma) tvaru.

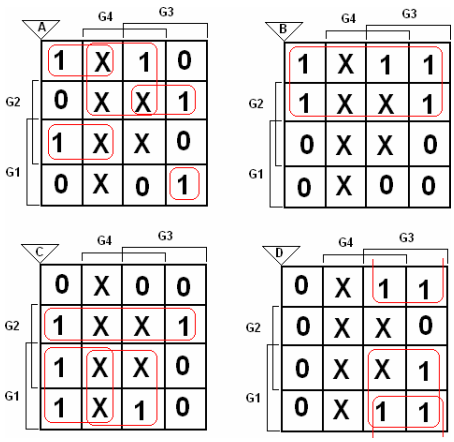
Z karnaughových máp budú mať výstupné funkcie tvar:

$A = \overline{G1}G4 + \overline{G1}G2\overline{G3} + \overline{G1}G2G3 + G1G2\overline{G3} + G1\overline{G2}G3\overline{G4}$
$B = \overline{G1}$
$C = \overline{G1}G2 + G1\overline{G3} + G1G4$
$D = G1G3 + \overline{G2}G3$

Tieto funkcie prepíšeme pomocou dvojitej negácie a Boolovej algebry na súčin súčinov za účelom realizácie uvedených funkcií na základe zadania pomocou hradiel NAND.

3.3.3 Otvorenie nového projektu

Postupujeme rovnako ako v podkapitole 2.2.1. Nový projekt vytvoríme pomocou príkazu New Project Wizard z File menu. Ako prvé sa objaví úvodné okno. Kliknutím na tlačidlo Next sa objaví prvé okno (...[page 1 of 5]), kde sa definuje miesto uloženia a meno projektu a entity. Kliknutím na tlačidlo **...** prvého riadku,



Obr. 3.1: *Karnaughove mapy pre jednotlive výstupy prevodníka z Grayovho kódu do kódu +3 ku BCD (0-9)*

sa objaví okno, kde si môžeme vybrať adresár, do ktorého uložíme projekt a do ktorého budú ukladané všetky vytvárané súbory, alebo si môžeme vytvoriť nový adresár. Pre náš projekt si vytvoríme adresár **prevodnik_gray**. V druhom riadku definujeme meno projektu a v treťom meno najvyššej úrovne entity. Pre náš projekt zadefinujeme rovnaké meno projektu aj meno entity aké je meno adresára – prevodnik_gray. Ak sme v tomto okne všetko zadefinovali klikneme na tlačidlo **Next**.Nasledujúce okno, ktoré sa otvorí slúži na pridanie súborov do projektu. V tomto projekte nebudeme pridávať žiadny súbor – klikneme na Next. Objaví sa tretie okno príkazu New project Wizard, v ktorom si definujeme rodinu obvodov, v našom prípade je to rodina MAX3000A. V spodnej časti tohoto okna vyberieme presne ten typ obvodu, s ktorým chceme pracovať. V našom prípade je to súčiastka EPM3064ATC44-10. Kliknutím na tlačidlo Next sme otvorili okno, v ktorom si môžeme nastaviť nástroje EDA. Tu tentokrát nebudeme nič nastavovať. V poslednom okne - piaté okno príkazu New Project Wizard sú zobrazené všetky voľby a nastavenia, ktoré sme v predchádzajúcich krokoch vykonali. Ak je všetko v súlade s našimi požiadavkami, klikneme na tlačidlo **Finish**.

3.3.4 Vytvorenie grafického návrhu projektu

Tento súbor vytvoríme vykonaním nasledujúcich krokov:

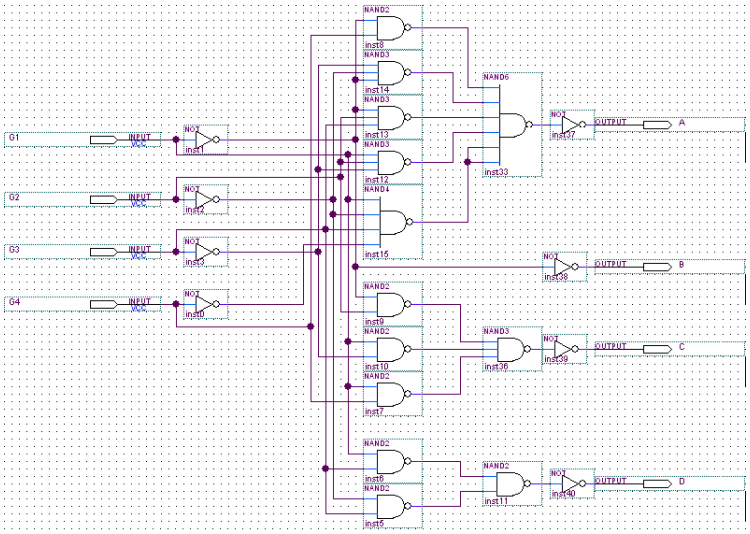
- Z **File** Menu vyberieme položku **New**
- V časti **Design Files** zvolíme **Block DiagramSchematic File**
- Kliknutím na tlacidlo **OK** sa otvorí okno grafického editora
- Z **File** Menu vyberieme položku **Save As**
- Vyberieme adresár **prevodnik_gray**, do ktorého uložíme náš súbor s názvom **prevodnik_gray.bdf**. Pod riadkom, kde sa definuje názov ukladaného súboru zaškrtneme voľbu **Add file to curent project** (pridať súbor do vlastného projektu).
- Kliknutím na tlačidlo **Save** uložíme a zároveň vložíme náš súbor do projektu.

3.3.5 Vytvorenie schémy

Schému vytvoríme podľa nasledujúcich krokov:

- Na panely nástrojov klikneme na **Symbol Tool**
- V okne, ktoré sa objaví vyberieme $Quartus_ROOT => primitives => logic

- Zvolíme súčiastku podľa schémy a potvrdíme stlačením na **OK**
- Symbol na ploche umiestníme na požadované miesto pohybom myšou a vložíme ho kliknutím ľavého tlačidla.
- Znovu klikneme na **Symbol tool**
- Pokračujeme vo vyberaní jednotlivých hradiel a pinov.
- Spojíme symboly hradiel a pinov s vodičmi, tak že sa s myšou priblížime k vstupu alebo výstupu symbolu a ak sa kurzor myši zmení na krížik, tak klikneme a držíme ľavé tlačidlo myši, približime sa k tomu výstupu alebo vstupu, ktorý chceme spojiť a ľavé tlačidlo myši pustíme.
- Názvy a hodnotu k pinom priradíme tak, že klikneme na pin pravým tlačidlom myši, zobrazí sa tabuľka, z ktorej vyberieme položku Properties a kliknutím na ňu sa zobrazí okno. To isté dosiahneme dvojitým kliknutím na symbol pinu.
- Výsledná schéma by mala vyzerať podobne, ako na obrázku 3.2. Na každý výstup pridáme hradlo **not**, aby nam logickú **"1"** reprezentoval rozsvietený segment sedemsegmentovky.
- Nakoniec vytvorenú schému uložíme (**File** menu => **Save**, alebo klavesovou skratkou **CTRL+S**)



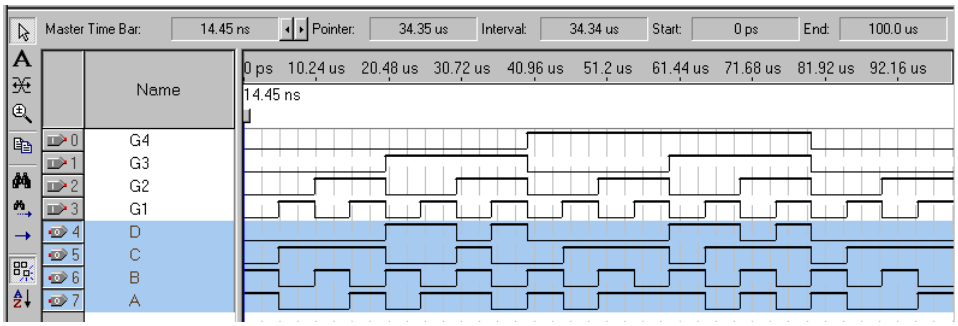
Obr. 3.2: *Výsledna schéma prevodníka z Grayovho kódu do kódu BCD+3 (0-9) v grafickom editore*

3.3.6 Kompilácia a priradenie pinov

Po uložení schémy pristupíme ku kompilácii projektu. Postupujeme podľa podkapitoly 2.2.4. Po úspešnej kompilácii pristupíme k priradeniu vstupných a výstupných pinov k jednotlivým vývodom obvodu podobne ako v podkapitole 2.2.4. Čísla jednotlivých pinov sú uvedené v tabuľke 3.1.

3.3.7 Simulácia

Po úspešnej kompilácii a priradení pinov môžeme prejsť na simuláciu projektu. Simuláciu urobíme podľa postupu uvedeného v podsekcií 2.2.5. **End time** nastavíme na 100μs, periódu jednotivých vstupných signálov nastavíme pomocou **Value** => **Count Value** na 40μs pre G4, 20μs s pre G3, 10μs s pre G2 a 5μs s pre G1. Uložíme a spustíme simuláciu. Podľa pravdivostnej tabuľky (tabuľka 3.2) skontrolujeme funkčnosť nášho zapojenia.



Obr. 3.3: *Časová simulácia prevodníka z Grayovho kódu do kódu BCD+3 (0-9) bez zapojených invertorov na výstupe*

3.3.8 Konfigurácia

Ak všetký simulované hodnoty súhlasia s hodnotami v pravdivostnej tabuľke, pokračujeme nahraním konfiguračného súboru do obvodu (postupujeme podľa podkapitoly 2.2.6).

3.3.9 Archivácia

Postup archivácie bol podrobne popísaný v podkapitole 2.2.7.

3.4 Zadanie 2.

Navrhnite prevodník z BCD kódu do kódu +3 ku BCD (0-9). Návrh realizujte pomocou hradiel NAND, v grafickom editore vývojového prostredia Quartus II.

3.5 Riešenie

3.5.1 Rozbor

V tomto príklade postupujeme analogicky ako v zadaní č. 1. (kapitola 3.2). Po syntéze zadaného príkladu, otvoríme nový projekt, zhotovíme v grafickom editore zapojenie, vykonáme kompiláciu a simuláciu. Použijeme tie isté prepínače a sedem-segmentovky ako v príklade č.1. Platí teda aj tá istá tabuľka pinov (tab. 3.1).

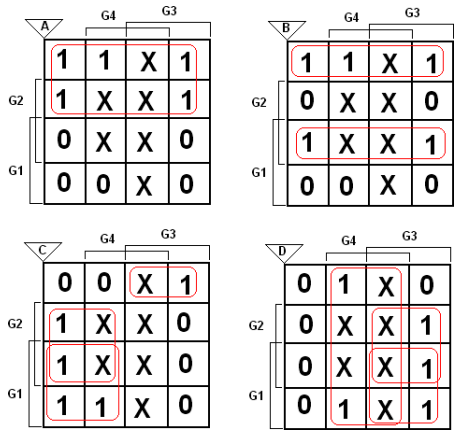
3.5.2 Syntéza

Ako prvé si zostavíme pravdivostnú tabuľku, v ktorej si označíme vstupy, výstupy (tab. 3.3)

Z pravdivostnej tabuľky zostavíme Karnaughove mapy (obr. 3.4) z ktorých určíme algebraické funkcie pre každý výstup. Funkcie potom upravíme podľa Boolovej algebry na tvar, ktorý môžeme realizovať pomocou hradiel NAND.

Tabuľka 3.3: *Pravdivostná tabuľka pre prevodník z BCD kódu do kódu +3 ku BCD*

dekadický	vstupy				výstupy			
	BCD 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	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



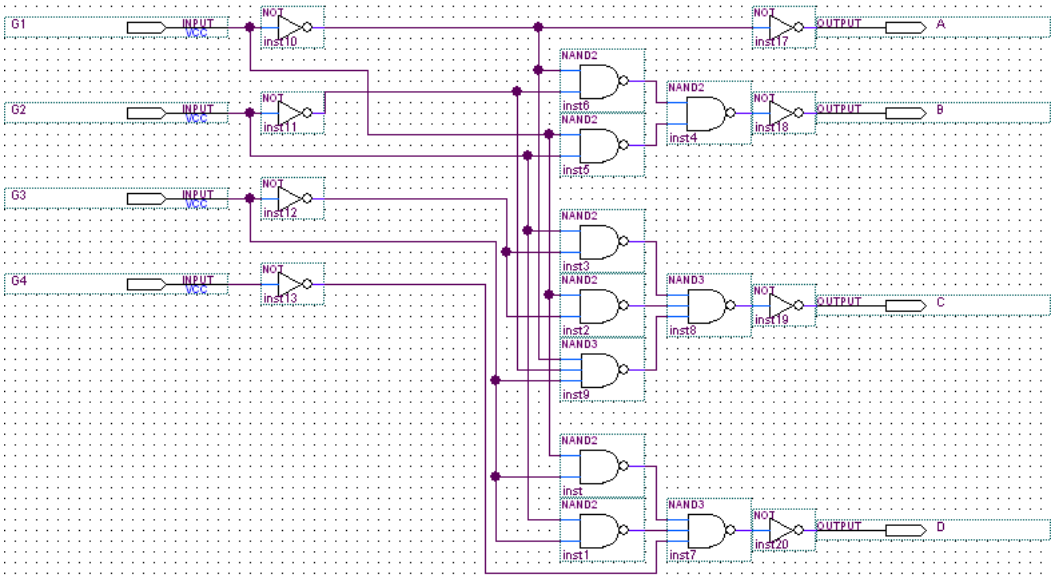
Obr. 3.4: *Karnaughove mapy pre jednotlive výstupy prevodníka z BCD kódu do kódu BCD+3 (0-9)*

Z Karnaughových máp dostávame výstupné funkcie v tvare:

$A = \overline{G1}$
$B = \overline{G1}\overline{G2} + G1G2$
$C = G2\overline{G3} + G1\overline{G3} + \overline{G1G2}G3$
$D = G4 + G1G3 + G2G3$

Tieto funkcie prepíšeme pomocou dvojitej negácie a Boolovej algebry na tvar súčin súčinov, aby sme mohli uvedené funkcie realizovať pomocou hradiel NAND. Na základe predchádzajúcej syntézy a použitím grafického editora (po otvorení nového projektu, v tomto prípade napríklad s názvom prevodnik_bcd) realizujeme zapojenie prevodníka BCD kódu na kód +3 ku BCD (obr. 3.5).

Po zhotovení schémy grafickom editore a jej uložení, projekt skompilujeme, priradíme jednotlivým výstupom piny obvodu, je tiež vhodné vykonať simuláciu. Po odkontrolovaní simulovaných výsledkov s pravdivostnou tabuľkou (tab. 3.3) nahrajeme konfiguráciu do obvodu a overíme funkčnosť návrhu. Projekt nakoniec archivujeme (kapitola 2.2.7).



Obr. 3.5: *Výsledna schéma prevodníka z BCD kódu do kódu +3 ku BCD (0-9) v grafickom editore*

Kapitola 4

Sekvenčné obvody

4.1 Teória

Sekvenčné obvody – sú logické obvody, ktorých výstup závisí od kombinácie vstupov a od vnútorných stavov obvodu z predchádzajúceho taktu. Medzi takéto obvody môžeme zaradiť klopné obvody (RS, D, T a JK), čítače (vpred, vzad, synchrónne a asynchrónne), posuvné registre a pamäte. **Čítače** – sú sekvenčné obvody, ktoré slúžia na registráciu počtu impulzov, ide o pamäťové obvody, ktoré možno použiť v rôznych aplikáciách napr. ako počítadlo impulzov, delenie frekvencie a na matematické operácie. Základnými prvkami sú klopné obvody. Čítače môžeme rozdeliť podľa viacerých hľadísk.
Podľa spôsobu preklápania:

- Asynchrónne (klopné obvody preklápajú postupne)
- Synchrónne (klopné obvody preklápajú súčasne so spoločným hodinovým impulzom)

Podľa smeru počítania:

- Vpred
- Vzad
- Vratný (reverzný - vpred, vzad)

Podľa použitého kódu:

- BCD (10) čítač
- Binárny (16) čítač
- Grayov čítač
- Johnsonov čítač

Syntézu sekvenčných obvodov môžeme opísať pomocou nasledovných bodov:

- Zostavenie pravdivostnej tabuľky na základe požadovanej činnosti sekvenčného obvodu
- Na základe prechodových matíc použitých klopných obvodov (D, T, JK a RS) sa zostavia Karnaughove mapy premenných
- Z Karnaughovej mapy sa určia vstupné funkcie klopných obvodov
- Koncová realizácia

4.2 Zadanie 1.

Navrhnite a zrealizujte desiatkový synchrónny čítač vzad v kóde BCD na báze D klopných obvodov (KO). Návrh realizujte v grafickom editore vývojového prostredia Quartus II.

4.3 Riešenie

4.3.1 Rozbor

Postup riešenia príkladu bude podobný ako v kapitole 3. Po syntéze zadaného príkladu zostavíme pravdivostnú tabuľku. Vykonáme minimalizáciu pomocou Karnaughových máp, napíšeme si algebraické rovnice, prepíšeme ich do UNDF tvaru a nakreslíme výslednú schému, ktorú potom zrealizujeme v grafickom editore softvéru Quartus II. Hodnoty výstupov zobrazíme na sedem segmentovej LED, z ktorých využijeme segmenty e, c z MSD sedem segmentovky a segmenty e, c z LSD sedem segmentovky. Displej reaguje na logickú úroven „0“, musíme teda na každý výstup pripojit hradlo NOT. Pri riešení tohto príkladu využijeme generátor hodinových impulzov. Kedže oscilátor na vývojovej doske CPLD-KIT má frekvenciu 25 MHz, musíme použit deličku frekvencie (VHDL kód deličky je uvedený v prílohe). V tomto príklade využijeme výstup z deličky, ktorého frekvencia je 1Hz, t.j. čítač nám bude počítať s frekvenciou 1 s. Všetky použité piny a ich popisy sú uvedené v tabuľke 4.1.

Tabuľka 4.1: *Čísla pinov*

Meno pinu	Typ pinu	Pin	Funkcia pinu
clk	Vstup	37	Vstup hodinového signálu z oscilátora 25MHz
reset	Vstup	27	Tlačidlo S2, slúžiace ako nulovacie (reset) tlačidlo
A	Výstup	5	Segment c LSD (0=LED ON, 1=LED OFF)
B	Výstup	42	Segment e LSD (0=LED ON, 1=LED OFF)
C	Výstup	20	Segment c MSD (0=LED ON, 1=LED OFF)
D	Výstup	22	Segment e MSD (0=LED ON, 1=LED OFF)

4.3.2 Syntéza

Na začiatku syntézy, na základe činnosti čítača, zhotovíme pravdivostnú tabuľku 4.2, v ktorej určíme logické hodnoty výstupov (A, B, C a D) pre možné stavy čítača.
Z pravdivostnej tabuľky zostavíme Karnaughove mapy (obr. 4.1), z ktorých určíme algebraické funkcie pre jednotlivé vstupy D KO.

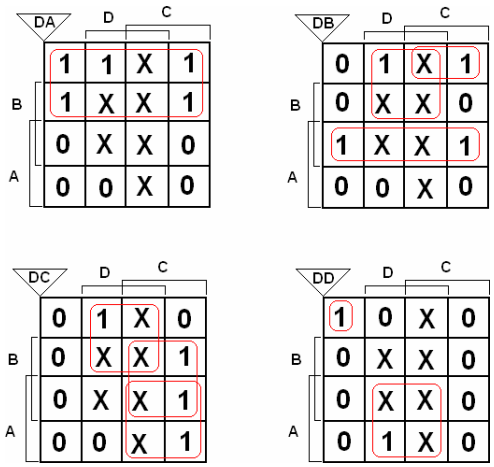
Algebraické funkcie zapíšeme v UDNF tvare, pomocou dvojitej negácie a Boolovej algebry ich upravíme do takého tvaru, aby sme ich mohli realizovať logikou NAND. Výsledné funkcie budú mať tvar:
$DA = \overline{A}$
$DB = AB + \overline{A}D + \overline{A\overline{B}}C$
$DC = \overline{A}D + AC + BC$
$DD = AD + \overline{ABCD}$

4.3.3 Otvorenie nového projektu

Postupujeme podobne ako v podkapitole 2.2.1. Nový projekt vytvoríme pomocou príkazu New Project Wizard z File menu. Ako prvé sa objaví úvodné okno. Kliknutím na tlačidlo Next sa objaví prvé okno (...[page 1 of 5]), kde sa definuje miesto uloženia a meno projektu a entity. Kliknutím na tlačidlo **...** prvého riadku, sa objaví okno, kde si môžeme vybrať adresár, do ktorého uložíme projekt a do ktorého budú ukladané všetky vytvárané súbory, alebo si môžeme vytvoriť nový adresár. Pre náš projekt si vytvoríme adresár

Tabuľka 4.2: *Pravdivostná tabuľka*

počet impulzov	Výstupy				stav čítača
	D	C	B	A	
0	0	0	0	0	0
1	1	0	0	1	9
2	1	0	0	0	8
3	0	1	1	1	7
4	0	1	1	0	6
5	0	1	0	1	5
6	0	1	0	0	4
7	0	0	1	1	3
8	0	0	1	0	2
9	0	0	0	1	1
10	0	0	0	0	0
11	1	0	0	1	9



Obr. 4.1: *Karnaughove mapy pre jednotlive výstupy čítača vzad v kode BCD*

counter_down. V druhom riadku definujeme meno projektu a v treťom meno najvyššej úrovne entity. Pre náš projekt zadefinujeme rovnaké meno projektu aj meno entity aké je meno adresára – **counter_down**. Ak sme v tomto okne všetko zadefinovali klikneme na tlačidlo **Next**.Nasledujúce okno, ktoré sa otvorí slúži na pridanie súborov do projektu. **V tomto projekte prídáme súbor**, ktorý realizuje funkciu deličky hodinových impulzov a je napísany v kóde VHDL. V riadku **File name** klikneme na tlačidlo s troma bodkami (**...**) a nájdeme cestu alebo miesto, kde sa uvedený súbor nachádza. V našom prípade si potrebný súbor skopírujeme do adresára nášho projektu a nazveme ho napr. delička.vhd. Po výbere súboru ho pridáme do projektu kliknutím na tlacidlo **Add**. Nakoniec klikneme na **Next**. Objaví sa tretie okno príkazu New project Wizard, v ktorom si definujeme rodinu obvodov, v našom prípade je to rodina **MAX3000A**. V spodnej časti tohoto okna vyberieme presne ten typ obvodu, s ktorým chceme pracovať. V našom prípade je to súčiastka **EPM3064ATC44-10**. Kliknutím na tlačidlo **Next** sme otvorili okno, v ktorom si môžeme nastaviť nástroje EDA. Tu tentokrát nebudeme nič nastavovať. V poslednom okne - piaté okno príkazu New Project Wizard sú zobrazené všetky voľby a nastavenia, ktoré sme v predchádzajúcich krokoch vykonali. Ak je všetko v súlade s našimi požiadavkami, klikneme na tlačidlo **Finish**.

4.3.4 Vytvorenie grafického návrhu projektu

Tento súbor vytvoríme vykonaním nasledujúcich krokov:

- Z **File** Menu vyberieme položku **New**
- V časti **Design Files** zvolíme **Block DiagramSchematic File**
- Kliknutím na tlacidlo **OK** sa otvorí okno grafického editora
- Z **File** Menu vyberieme položku **Save As**
- Vyberieme adresár **counter_down**, do ktorého uložíme náš súbor s názvom **counter_down.bdf**. Pod riadkom, kde sa definuje názov ukladaného súboru zaškrtneme voľbu **Add file to curent project** (pridať súbor do vlastného projektu).
- Kliknutím na tlačidlo **Save** uložíme a zároveň vložíme náš súbor do projektu.

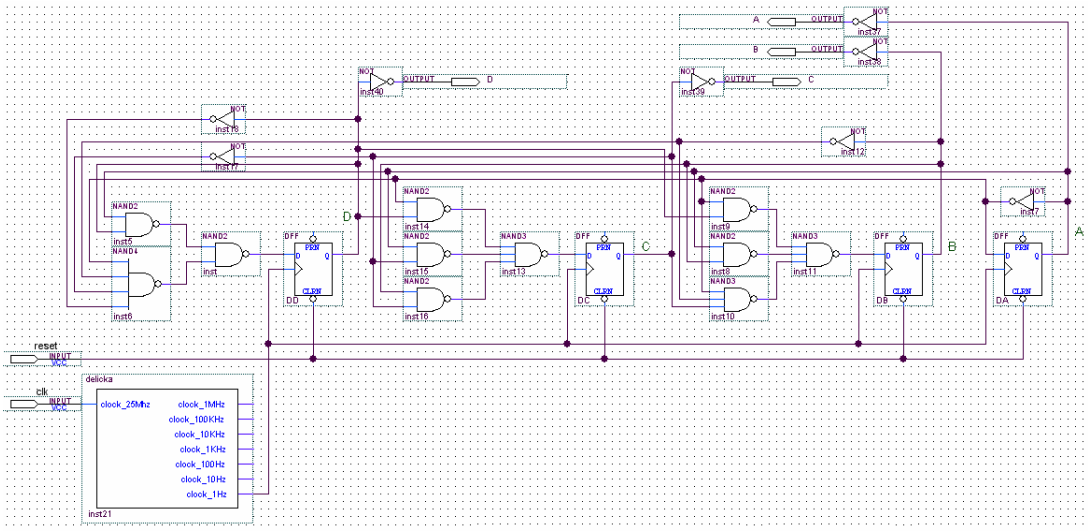
4.3.5 Vytvorenie schémy

Schému vytvoríme podľa nasledujúcich krokov:

- Na panely nástrojov klikneme na **Symbol Tool**
- V okne, ktoré sa objaví vyberieme D klopný obvod ($Quartus_ROOT => primitives => storage => dff) a potvrdíme stlačením na **OK**
- Symbol na ploche umiestníme na požadované miesto pohybom myšou a vložíme ho kliknutím ľavého tlačidla.
- Potrebujeme 4 kusy D klopných obvodov, vložíme teda ešte ďalšie tri kusy.
- Premenujeme D klopné obvody na DA, DB, DC a DD (dvakrát klikneme na symbol KO a zmeníme meno).
- Na realizáciu funkcií, ktoré sme dostali z Karnaughových máp si vyberieme potrebné logické hradlá NAND a NOT a umiestníme ich na pracovnú plochu.
- Podla týchto funkcií zapojíme logický obvod, ktorého výstupy pripojíme na požadované vstupy D – KO.
- Výstupné piny D – KO pripojíme na výstupne piny A, B, C a D, ktoré budú pripojené na jednotlivé segmenty LED sedem-segmentoviek.
- Na hodinové vstupy na D klopných obvodov privedieme signál clock_1Hz z bloku deličky, ktorý vygenerujeme z VHDL kódu.
- V ľavej časti v okne **Project navigator** v záložke **Files** klikneme pravým tlačidlom myši na meno súboru **delicka.vhd** a vyberieme **Create Symbol Files for Current File**.
- Pomocou nástroja **Symbol tool** vyberieme vygenerovaný symbol deličky (Project => delicka)
- Vstup deličky zapojíme na vstupný pin **clk**, ktorý sa pripojí k oscilátoru 25MHz na doske (pin 37).
- Vstup **reset**, teda tlačidlo S2 pripojíme na všetky vstupy reset (CLRN) D KO.
- Výsledná schéma by mala vyzerať podobne, ako na obrázku 4.2. Na každý výstup ešte pridáme hradlo **not**, aby nám logickú **"1"** reprezentoval rozsvietený segment sedemsegmentovky.
- Nakoniec vytvorenú schému uložíme (**File** menu => **Save**, alebo klavesovou skratkou **CTRL+S**)

4.3.6 Kompilácia a priradenie pinov

Po uložení schémy pristupíme ku kompilácii projektu. Postupujeme podľa podkapitoly 2.2.4. Po úspešnej kompilácii pristupíme k priradeniu vstupných a výstupných pinov k jednotlivým vývodom obvodu podobne ako v podkapitole 2.2.4. Čísla jednotlivých pinov sú uvedené v tabuľke 4.1.



Obr. 4.2: *Výsledna schéma desiatkového synchrónného čítača vzad v kóde BCD v grafickom editore*

4.3.7 Časová simulácia

Po úspešnej kompilácii a priradení pinov môžeme prejsť na simuláciu projektu. Simuláciu musíme robiť bez zapojenej deličky hodinového signálu, pretože by sme museli nastaviť dlhý čas simulácie a na relevantné výsledky by sme čakali veľmi dlho. Preto pred simuláciou deličku odpojime a priamo privedieme vstupný hodinový signál z pinu clk na hodinové vstupy D klopných obvodov. Simuláciu urobíme podľa postupu uvedeného v podsekcií 2.2.5. **End time** nastavíme na napríklad na 100μs, periódu vstupného hodinového signálu **clk** nastavíme pomocou **Value => Count Value** napríklad na **5μs**. Vstup **reset** nastavíme trvalo na logicku jednotku (napríklad pomocou **Value => Forcing High (1)**). Uložíme a spustíme simuláciu. Podľa pravdivostnej tabuľky (tabuľka 4.2) skontrolujeme funkčnosť nášho zapojenia.

4.3.8 Funkčná simulácia

Pre funkcnú simuláciu použijeme tie isté nastavenia ako pre časovú simuláciu. Otvoríme okno Simulator Tool výberom z Tools menu. Ako prvé zmeníme **Simulation mode** (mód simulácie) na **Functional**. Potom klikneme na tlačidlo **Generate Functional Simulation Netlist**. Ak máme vygenerovaný Netlist, začneme simuláciu kliknutím na tlacidlo Start. Po skončení simulácie si môžeme výsledky pozrieť po kliknutí na tlačidlo **Report**.

4.3.9 Konfigurácia

Ak všetký simulované hodnoty súhlasia s hodnotami v pravdivostnej tabuľke, pokračujeme nahraním konfiguračného súboru do obvodu (postupujeme podľa podkapitoly 2.2.6).

4.3.10 Archivácia

Postup archivácie bol podrobne popísaný v podkapitole 2.2.7.

4.4 Zadanie 2.

Navrhnite a zrealizujte desiatkový synchrónny čítač vpred v kóde BCD na báze JK KO. Návrh realizujte v grafickom editore vývojového prostredia Quartus II.

4.5 Riešenie

4.5.1 Rozbor

V tomto príklade postupujeme analogicky ako v zadaní číslo 1 (kapitola 4.2). Vykonáme syntézu obvodu, otvoríme nový projekt, zhotovíme zapojenie v grafickom editore, vykonáme kompiláciu, simuláciu a konečnú konfiguráciu. Výstupy čítača zobrazíme na segmentoch sedem-segmetových displejoch (tabuľka 4.1).

4.5.2 Syntéza

Na začiatku syntézy, na základe činnosti čítača, zostavíme pravdivostnú tabuľku, v ktorej určíme logické hodnoty výstupov (A, B, C a D) pre možné stavy čítača (tabuľka č. 4.4. Z pravdivostnej tabuľky a na zá-

Tabuľka 4.3: *Matica prechodov*

Q_n	$\longrightarrow$	$Q_n + 1$	**J**	**K**
0	$\longrightarrow$	0	0	X
0	$\longrightarrow$	1	1	X
1	$\longrightarrow$	0	X	1
1	$\longrightarrow$	1	X	0

Tabuľka 4.4: *Pravdivostná tabuľka*

počet impulzov	**Výstupy**				**stav čítača**
	D	C	B	A	
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	2
3	0	0	1	1	3
4	0	1	0	0	4
5	0	1	0	1	5
6	0	1	1	0	6
7	0	1	1	1	7
8	1	0	0	0	8
9	1	0	0	1	9
10	0	0	0	0	0
11	0	0	0	1	1

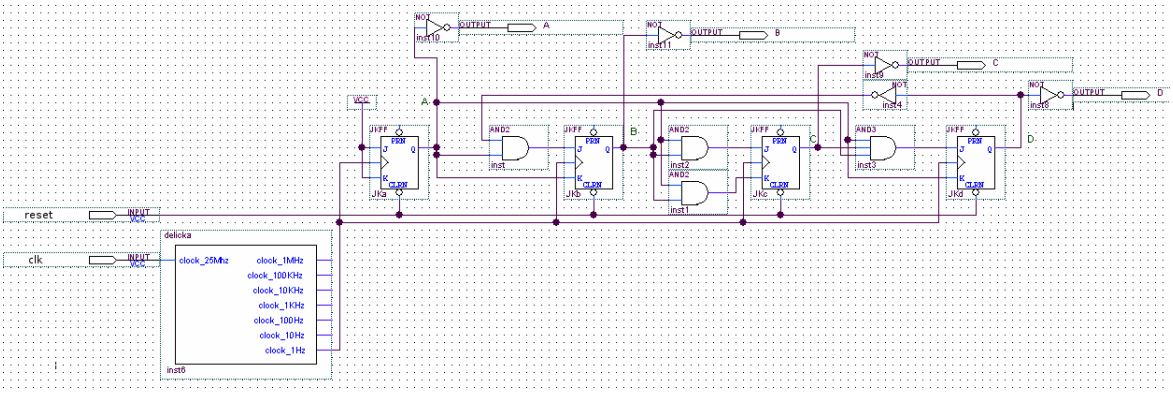
klade matice prechodov (tabuľka 4.3) zostavíme Karnaughove mapy pre jednotlivé vstupy J a K KO. Z nich dostaneme výsledné logické funkcie pre jednotlivé vstupy J a K KO. Pomocou týchto funkcií zhotovíme výslednú schému.

Z Karnaughových máp dostaneme výsledné logické funkcie v tvare:

$J_A = 1$ $\quad K_A = 1$
$J_B = A\overline{D}$ $\quad K_B = A$
$J_C = AB$ $\quad K_C = AB$
$J_D = ABC$ $\quad K_D = A$

Pomocou týchto rovníc realizujeme hradlami AND vo výslednom zapojení logiku, ktorá slúži na získanie potrebných stavov čítača. V Quartus II otvoríme nový projekt s názvom counter_up. V grafickom okne realizujeme naše zapojenie zastavené zo štyroch JK klopných obvodov, hradiel AND, invertorov NOT, vstupných pinov a podobne ako v prvom zadaní pridáme aj obvod deličky hodinových impulzov(kapitola

4.3.5). Vzájomným prepojením týchto prvkov na základe vykonanej syntézy dostaneme výsledné zapojenie 4 bitového čítača vpred na báze JK klopných obvodov(obrázok 4.3).



Obr. 4.3: *Výsledna schéma desiatkového synchrónného čítača vpred v kóde BCD v grafickom editore*

4.5.3 Kompilácia a priradenie pinov

Po uložení schémy pristupíme ku kompilácii projektu. Postupujeme podľa podkapitoly 2.2.4. Po úspešnej kompilácii pristupíme k priradeniu vstupných a výstupných pinov k jednotlivým vývodom obvodu podobne ako v podkapitole 2.2.4. Čísla jednotlivých pinov sú rovnaké ako v predchadzajúcom zadaní a sú uvedené v tabuľke 4.1.

4.5.4 Časová a funkčná simulácia

Postupujeme rovnako, ako v podkapitolách 4.3.7 a 4.3.8.

4.5.5 Konfigurácia

Ak všetký simulované hodnoty súhlasia s hodnotami v pravdivostnej tabuľke (tabuľka číslo 4.4), pokračujeme nahraním konfiguračného súboru do obvodu (postupujeme podľa podkapitoly 2.2.6).

4.5.6 Archivácia

Postup archivácie bol podrobne popísaný v podkapitole 2.2.7.

4.6 VHDL kód deličky

```
library IEEE;
use  IEEE.STD_LOGIC_1164.all;
use  IEEE.STD_LOGIC_ARITH.all;
use  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY delicka IS

        PORT
        (
                clock_25Mhz                             : IN    STD_LOGIC;
                clock_1MHz                              : OUT   STD_LOGIC;
                clock_100KHz                    : OUT   STD_LOGIC;
                clock_10KHz                             : OUT   STD_LOGIC;
                clock_1KHz                              : OUT   STD_LOGIC;
                clock_100Hz                             : OUT   STD_LOGIC;
                clock_10Hz                              : OUT   STD_LOGIC;
                clock_1Hz                               : OUT   STD_LOGIC);

END delicka;

ARCHITECTURE a OF delicka IS

        SIGNAL  count_1Mhz: STD_LOGIC_VECTOR(4 DOWNTO 0);
        SIGNAL  count_100Khz, count_10Khz, count_1Khz : STD_LOGIC_VECTOR(2 DOWNTO 0);
        SIGNAL  count_100hz, count_10hz, count_1hz : STD_LOGIC_VECTOR(2 DOWNTO 0);
        SIGNAL  clock_1Mhz_int, clock_100Khz_int, clock_10Khz_int, clock_1Khz_int: STD_LOGIC;
        SIGNAL  clock_100hz_int, clock_10Hz_int, clock_1Hz_int : STD_LOGIC;
BEGIN
        PROCESS
        BEGIN
-- Divide by 25
                WAIT UNTIL clock_25Mhz'EVENT and clock_25Mhz = '1';
                        IF count_1Mhz < 24 THEN
                                count_1Mhz <= count_1Mhz + 1;
                        ELSE
                                count_1Mhz <= "00000";
                        END IF;
                        IF count_1Mhz < 12 THEN
                                clock_1Mhz_int <= '0';
                        ELSE
                                clock_1Mhz_int <= '1';
                        END IF;

-- Ripple clocks are used in this code to save prescalar hardware
-- Sync all clock prescalar outputs back to master clock signal
                        clock_1Mhz <= clock_1Mhz_int;
                        clock_100Khz <= clock_100Khz_int;
                        clock_10Khz <= clock_10Khz_int;
                        clock_1Khz <= clock_1Khz_int;
                        clock_100hz <= clock_100hz_int;
                        clock_10hz <= clock_10hz_int;
                        clock_1hz <= clock_1hz_int;
        END PROCESS;

-- Divide by 10
        PROCESS
```

```
        BEGIN
                WAIT UNTIL clock_1Mhz_int'EVENT and clock_1Mhz_int = '1';
                        IF count_100Khz /= 4 THEN
                                count_100Khz <= count_100Khz + 1;
                        ELSE
                                count_100khz <= "000";
                                clock_100Khz_int <= NOT clock_100Khz_int;
                        END IF;
        END PROCESS;

-- Divide by 10
        PROCESS
        BEGIN
                WAIT UNTIL clock_100Khz_int'EVENT and clock_100Khz_int = '1';
                        IF count_10Khz /= 4 THEN
                                count_10Khz <= count_10Khz + 1;
                        ELSE
                                count_10khz <= "000";
                                clock_10Khz_int <= NOT clock_10Khz_int;
                        END IF;
        END PROCESS;

-- Divide by 10
        PROCESS
        BEGIN
                WAIT UNTIL clock_10Khz_int'EVENT and clock_10Khz_int = '1';
                        IF count_1Khz /= 4 THEN
                                count_1Khz <= count_1Khz + 1;
                        ELSE
                                count_1khz <= "000";
                                clock_1Khz_int <= NOT clock_1Khz_int;
                        END IF;
        END PROCESS;

-- Divide by 10
        PROCESS
        BEGIN
                WAIT UNTIL clock_1Khz_int'EVENT and clock_1Khz_int = '1';
                        IF count_100hz /= 4 THEN
                                count_100hz <= count_100hz + 1;
                        ELSE
                                count_100hz <= "000";
                                clock_100hz_int <= NOT clock_100hz_int;
                        END IF;
        END PROCESS;

-- Divide by 10
        PROCESS
        BEGIN
                WAIT UNTIL clock_100hz_int'EVENT and clock_100hz_int = '1';
                        IF count_10hz /= 4 THEN
                                count_10hz <= count_10hz + 1;
                        ELSE
                                count_10hz <= "000";
                                clock_10hz_int <= NOT clock_10hz_int;
                        END IF;
        END PROCESS;
```

```
-- Divide by 10
        PROCESS
        BEGIN
                WAIT UNTIL clock_10hz_int'EVENT and clock_10hz_int = '1';
                        IF count_1hz /= 4 THEN
                                count_1hz <= count_1hz + 1;
                        ELSE
                                count_1hz <= "000";
                                clock_1hz_int <= NOT clock_1hz_int;
                        END IF;
        END PROCESS;

END a;
```

Kapitola 5

VHDL

5.1 Teória

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 rýchlostou môže podariť len vtedy, ked 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**evices) 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 niekolko najpoužívanejších sa stalo de facto priemyslovými štandardmi (napr. VHDL a Verilog). Uvedenie **VHDL** (**V**ery **H**igh **S**peed **I**ntegrated **C**ircuits- VHSIC 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 velmi rýchlych integrovaných obvodov (VHSIC).V roku 1987 bol jazyk prijatý ako štandard IEEE pod číslom 1079-1987. Tento štandard bol a aj je nad'alej vyvíjaný. Najdôležitejším dôvodom pre použitie VHDL je to, že šetrí čas a peniaze.

5.2 Zadanie 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 d'alšej 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 z kapitoli 4. 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 prepínače S4. V hornej polohe reprezentujú logickú úroven „0“ a v dolnej polohe reprezentujú logickú úroven „1“. Výstupy budeme zobrazovať na sedem segmentovej LED, pričom použijeme segmenty e, c z MSD sedem segmentovky a segmenty e, c z LSD sedem segmentovky. Pri návrhu ešte musíme brať do úvahy tú skutocnosť, že náš sedem segmentový displej reaguje na logickú úroven 0 (hradla **not** na výstup).

Tabuľka 5.1: *Čísla pinov*

Meno pinu	Typ pinu	Pin	Funkcia pinu
G1	Vstup	31	Prepínač 4 (1 = poloha dole, 0 = poloha hore)
G2	Vstup	33	Prepínač 3 (1 = poloha dole, 0 = poloha hore)
G3	Vstup	34	Prepínač 2 (1 = poloha dole, 0 = poloha hore)
G4	Vstup	35	Prepínač 1 (1 = poloha dole, 0 = poloha hore)
A	Výstup	5	Segment c LSD (0=LED ON, 1=LED OFF)
B	Výstup	42	Segment e LSD (0=LED ON, 1=LED OFF)
C	Výstup	20	Segment c MSD (0=LED ON, 1=LED OFF)
D	Výstup	22	Segment e MSD (0=LED ON, 1=LED OFF)

5.3.2 Syntéza

Na začiatku návrhu si zostavíme pravdivostnú tabuľku (tabuľka 5.2) nášho prevodníka, v ktorej si označíme vstupy a výstupy.

Tabuľka 5.2: *Pravdivostná tabuľka*

dekadický	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

Ak budeme realizovať príklad tak, že bude totožný s grafickým riešením budeme postupovať podobne ako v kapitole 3.3.2. Zostavíme si Karnaughove mapy a napíšeme si algebraické rovnice a na základe týchto rovníc realizujeme VHDL kód (príloha).

My si najprv zostavíme VHDL kód priamo z pravdivostnej tabuľky (tabuľka 5.2). Platí tabuľka pinov 5.1 s tým rozdielom, že vstupné piny majú názov gray_code s číslom od 1 po 4. V tabuľke č.5.1 a 5.2 sú označené ako G1 až G4. Výstupné piny majú názov: plus3_code s číslami 1 až 4 a v tabuľkách č.5.1 a 5.2 sú oznacené ako A, B, C a D.

5.3.3 Otvorenie nového projektu

Postupujeme ako v minulých kapitolach (napríklad kapitola č. 2.2.1). Meno hlavnej entity zvolíme **gray**. Je to preto, lebo ako hlavnú entitu použijeme entitu s presne týmto názvom (viď nižšie). Po vytvorení projektu pokračujeme vytvorením textového návrhu použitím jazyka VHDL.

5.3.4 Vytvorenie textového návrhu pomocou jazyka VHDL

Postupujeme následovne:

- Z File Menu vyberieme položku **New**
- V sekcií Design Files zvolíme **VHDL File**
- Kliknutím na tlacidlo **OK** sa otvorí okno textového editora
- Z **File** Menu vyberieme položku **Save As**
- Vyberieme adresár nášho projektu (automaticky by mal byť vybraný, ak sme postupovali podľa postupu uvedeného vyššie), do ktorého uložíme náš súbor (**Save**) s názvom gray.vhd. Pod riadkom, kde sa definuje názov ukladaného súboru zaškrtneme voľbu **Add file to curent project** (pridat súbor do aktuálneho projektu). Táto položka by mala byt implicitne zaškrtnutá.
- Kliknutím na tlacidlo **Save** uložíme a zároveň vložíme súbor do projektu.

5.3.5 Vytvorenie VHDL kódu

Ako prvé zadefinujeme knižnice štandardov, ktoré budeme používat. Po zadefinovaní 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 volbou **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 automatický umiestnená do textového súboru na pozíciu kurzora. Potom do štruktúry vložíme parametre, pre náš príklad. V samotnom tele architektúry zadefinujeme príkazmi nami požadovaný proces podľa algebraických rovníc ktoré sme dostali z Karnaughových máp (pozri kapitolu 3.3.2) alebo výhodnejšie na základe pravdivostnej tabuľky.

Výsledný kód vytvorený na základe pravdivostnej tabuľky vložíme do otvoreného textového editora:

```
--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 --nazov entity, v nasom pripade aj hlavnej entity
port
(
gray_code : IN std_logic_vector (4 downto 1); --vektor vstupov
plus3_code : OUT std_logic_vector (4 downto 1); --vektor vystupov
);
end entity gray;
-- samotne telo programu
architecture gray_tabulka of gray is
begin
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 a priradenie pinov

Po uložení schémy pristupíme ku kompilácii projektu. Postupujeme podľa podkapitoly 2.2.4. Po úspešnej kompilácii pristupíme k priradeniu vstupných a výstupných pinov k jednotlivým vývodom obvodu podobne ako v podkapitole 2.2.4. Čísla jednotlivých pinov sú uvedené v tabuľke 5.1.

5.3.7 Funkčná a časová simulácia

Po úspešnej kompilácii a priradení pinov môžeme prejsť na simuláciu projektu. Simuláciu urobíme rovnako ako v kapitole 3.3.7. **End time** nastavíme na 100μs, periódu jednotivých vstupných signálov nastavíme pomocou **Value** => **Count Value** na 40μs pre gray_code(4), 20μs s pre gray_code(3), 10μs s pre gray_code(2) a 5μs pre gray_code(1). Uložíme. V okne **Simulator Tool** z **Tool Menu** vyberieme typ simulácie na **Functional** a vygenerujeme netlist. Po vygenerovaní spustíme simuláciu tlačidlom **Start**. Podľa pravdivostnej tabuľky (tabuľka 5.2) skontrolujeme funkčnosť nášho zapojenia.

5.3.8 Konfigurácia

Ak všetký simulované hodnoty súhlasia s hodnotami v pravdivostnej tabuľke, pokračujeme nahraním konfiguračného súboru do obvodu (postupujeme podľa podkapitoly 2.2.6).

5.3.9 Archivácia

Postup archivácie bol podrobne popísaný v podkapitole 2.2.7.

5.4 Zadanie 2.

Navrhnite prevodník z BCD 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 zadaní č. 1.(kapitola 5.2.) Zostavíme si pravdivostnú tabuľku prevodníka. Na základe pravdivostnej tabuľky (tab. 5.3.) zostavíme VHDL kód, vykonáme kompiláciu, priradenie pinov, simuláciu a konečnú konfiguráciu projektu.

Tabuľka 5.3: *Pravdivostná tabuľka pre prevodník z BCD kódu do kódu +3 ku BCD*

dekadický	**vstupy**				**výstupy**			
	BCD 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	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

Výsledný kód vytvorený na základe pravdivostnej tabuľky vložíme do otvoreného textového editora:

```
--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
(
--vstupny vektor
bcd_code : IN std_logic_vector (4 downto 1);
--vystupny vektor
plus3_code : OUT std_logic_vector (4 downto 1);
);
end entity BCD;
--samotna architektura
architecture bcd_tabulka of bcd is
begin
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;
```

Ďalej postupujeme rovnako ako v prvom zadaní (kompilácia, simulácia, konfigurácia, archivácia).

5.6 Príklad VHDL kódu využitím algebraických rovníc získaných pomocou Karnaughových máp pre prevodník Grayovho kódu do BCD+3

```
--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;
);
END prevodnik_Gray_vhdl;
--samotne telo navrhu
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

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