

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, synchronne 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)
- Synchronne (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

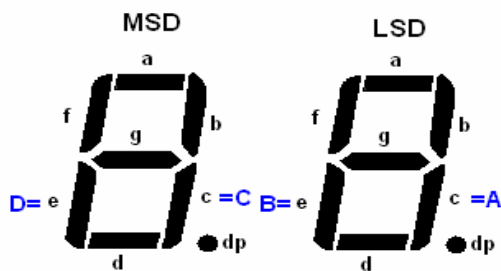
4.2 ZADANIE PRÍKLADU Č.1

Navrhните 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 (obr.1). Displej reaguje na logickú úroveň „0“, musíme teda na každý výstup pripojiť hradlo NOT.



Obr.1: Zobrazenie sedem segmentoviek MSD a LSD

Pri riešení tohto príkladu využijeme generátor hodinových impulzov. Keďže oscilátor na vývojovej doske UP1 CPLD má frekvenciu 25.175 MHz, musíme použiť 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 počítat s frekvenciou 1 s.

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

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

Tab.1: Tabuľka pinov

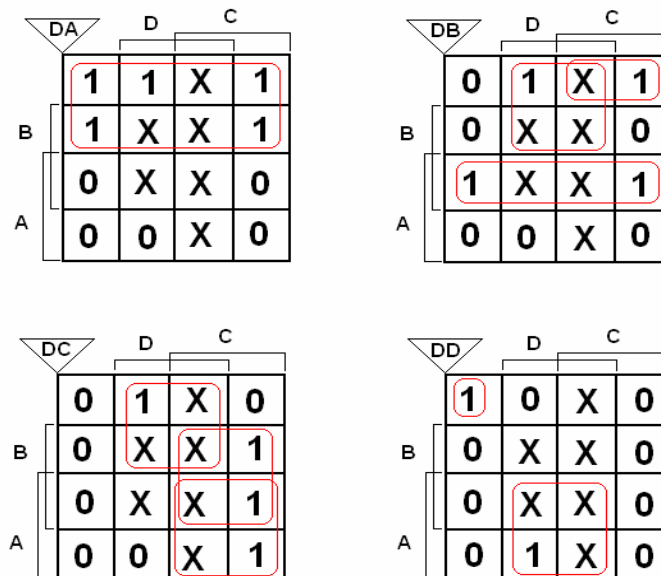
4.3.2 SYNTÉZA

Na začiatku syntézy, na základe činnosti čítača, zhotovíme pravdivostnú tabuľku (tab.2), v ktorej určíme logické hodnoty výstupov (A, B, C a D) pre možné stavy čítača.

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

Tab.2: Pravdivostná tabuľka čítača

Z pravdivostnej tabuľky zostavíme Karnaughove mapy (obr.2), z ktorých určíme algebraické funkcie pre jednotlivé vstupy D KO.



Obr.2: Karnaughove mapy

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:


$$DA = \bar{A} \quad (4.1)$$

$$DB = AB + \bar{A}D + \bar{A}\bar{B}C \quad (4.2)$$

$$DC = \bar{A}D + AC + BC \quad (4.3)$$

$$DD = AD + \bar{A}\bar{B}\bar{C}\bar{D} \quad (4.4)$$

4.3.3 OTVORENIE NOVÉHO PROJEKTU

Nový projekt otvoríme voľbou **New Project Wizard** z **File menu**. Ako prvé sa objaví úvodné okno, kde kliknutím na tlačidlo **Next** sa otvorí prvé okno, v ktorom definujeme miesto uloženia, názov projektu a názov entity. Kliknutím na tlačidlo  sa objaví štruktúra adresárov, z ktorých si vyberieme ten, do ktorého chceme náš projekt ukladať. Pre projekt vytvoríme napr. adresár: *4_D_down*. V ďalších riadkoch definujeme názov projektu a názov najvyššej úrovne entity. V tomto prípade zvolíme rovnaký názov projektu a názov najvyššej úrovne entity ako názov adresára, kde ukladáme súbory projektu - *4_D_down*. Voľbu potvrdíme tlačidlom **Next**.

V prípade, ak potrebujeme, resp. môžeme použiť určité bloky, resp. komponenty z iných projektov, môžeme tak urobiť a pridať ich do tohto projektu v nasledujúcom okne. Tým si v určitých prípadoch môžeme výrazne ušetriť čas. V tomto projekte pridáme súbor, ktorý je už vytvorený (súbor napísaný vo VHDL kóde) a realizuje funkciu deličky hodinových impulzov. V riadku **File name** klikneme na tlačidlo s tromi 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. *delicka.vhd*. Po výbere súboru ho pridáme do projektu kliknutím na tlačidlo **Add**.

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

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

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

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

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

4.3.4 VYTvorenie grafického návrhu projektu

Postupujeme podľa nasledovného postupu:

- Z **File Menu** vyberieme položku **New**
- V záložke **Design Files** zvolíme **Block Diagram/Schematic File**
- Kliknutím na tlačidlo **OK** sa otvorí okno grafického editora
- Z **File Menu** vyberieme položku **Save As**
- Vyberieme adresár *4_D_down*, do ktorého uložíme náš súbor s názvom *4_D_down.bdf*. Pod riadkom, kde sa definuje názov ukladaného súboru, zaškrtneme voľbu **Add file to current project** (pridať súbor do aktuálneho projektu). Táto položka by mala byť implicitne zaškrtnutá.
- Kliknutím na tlačidlo **Save**, uložíme a zároveň vložíme súbor do projektu.

4.3.5 Vytvorenie schémy zapojenia

Tak ako to bolo uvedené v predchádzajúcej kapitole (kapitola 3), klikneme na voľbu **Symbol Tool**. V okne, ktoré sa zobrazí si v knižniciach nájdeme D klopný obvod (*primitives*→*storage*→*dff*) a umiestnime ho na pracovnú plochu blokového editora, kde realizujeme náš návrh. Pre náš návrh potrebujeme celkovo štyri D - KO pre náš návrh. Dvojitým kliknutím na D klopný obvod si zmeníme názov na DA, ďalšie D – KO na DB, DC a DD aby sme ich vedeli rozlíšiť. Na realizáciu funkcií (4.1, 4.2, 4.3, 4.4), ktoré sme dostali z Karnaughových máp si vyberieme potrebné logické hradlá NAND

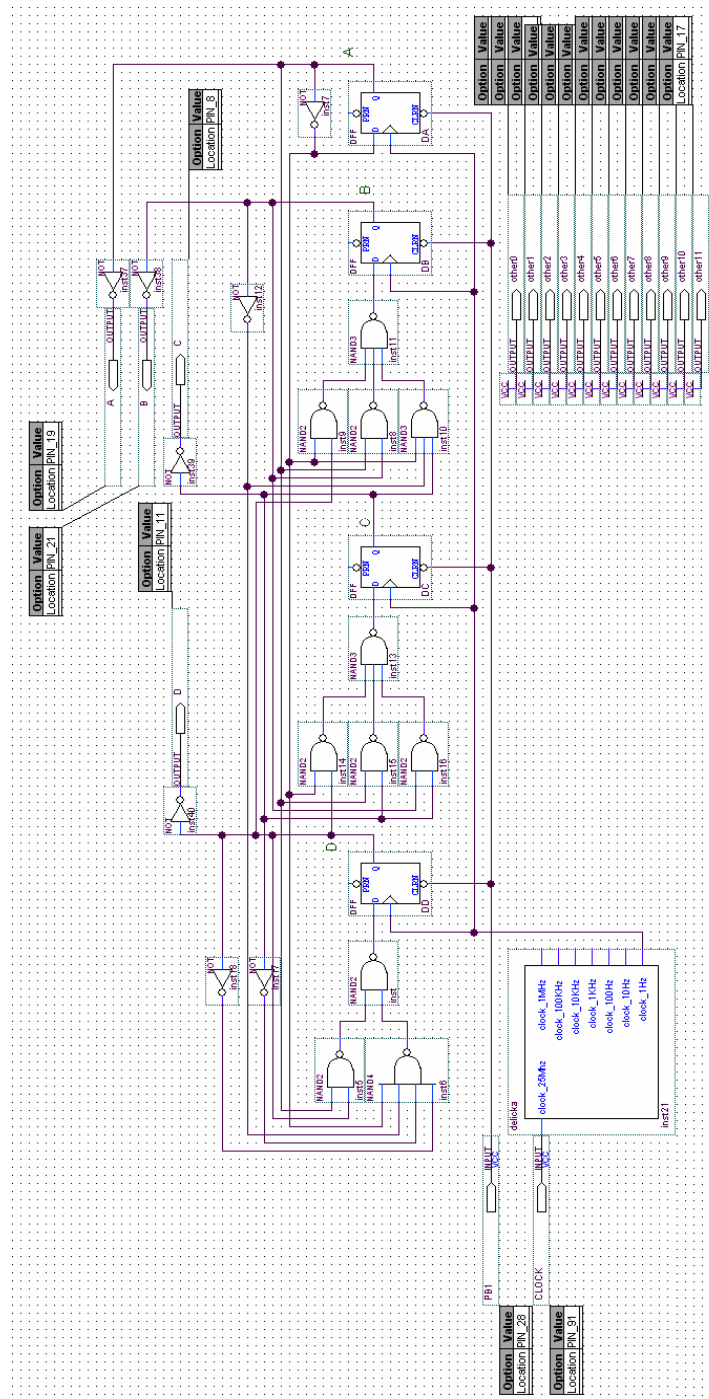
a inventory NOT a umiestnime ich na pracovnú plochu. Podľa funkcií (4.1, 4.2, 4.3, 4.4) zapojíme logický obvod, ktorého výstupy pripojíme na požadované vstupy D – KO. Výstupné piny D – KO (A, B, C a D) pripojíme na jednotlivé segmenty LED sedem segmentoviek. Na hodinové vstupy na D klopných obvodov privedieme signál zo systémových hodín. Do trasy hodín pripojíme ešte blok deličky, ktorý vygenerujeme z VHD kódu (pozri prílohu).

Postup generovania grafického symbolu z VHD kódu je nasledovný:

- Voľbou **Open** z **File menu** si otvoríme súbor *delicka.vhd*
- Z **File menu** si vyberieme položku **Create/Update** a tam si vyberieme **Create Symbol Files for Current File**
- Ku grafickej reprezentácii bloku deličky pristúpime cez **Symbol Tool** (project→delicka)
- Vložíme blok do okna grafického editora

Z vygenerovaného bloku deličky využijeme len výstup s frekvenciou 1 Hz. Ak by sme chceli, aby naše počítadlo počítalo rýchlejšie, použijeme výstup s vyššou frekvenciou. Pomocou programového prostriedku **Orthogonal Node Tool** pospájame všetky prvky podľa pravidiel, ktoré sme dostali na základe syntézy. Dostaneme výslednú schému projektu (obr.3). Názvy pinom priradíme tak, že klikneme pravým tlačidlo na príslušný pin a vyberieme položku **Properties**.

Pred kompiláciou musíme ešte výstupným signálom priradiť zodpovedajúce čísla pinov obvodu, voľbou **Pins** v **Assignments menu**. V stĺpci **To** napíšeme názvy výstupných signálov a v stĺpci **Location** im priradíme príslušné čísla pinov podľa tab.1 (obr.4).



Obr.3: Výsledné zapojenie

	To	Location	General Function	Special Function	Reserved
1	CLOCK	PIN_91	Dedicated Clock	Global_CLK,CKLK	
2	PB1	PIN_28	Row I/O		
3	A	PIN_19	Row I/O		
4	B	PIN_21	Row I/O		
5	C	PIN_8	Row I/O		
6	D	PIN_11	Row I/O	CLKUSR	
7	other0	PIN_17	Row I/O		
8	other1	PIN_18	Row I/O		
9	other2	PIN_20	Row I/O		
10	other3	PIN_23	Row I/O	RDYnBSY	
11	other4	PIN_24	Row I/O		
12	other5	PIN_25	Row I/O		
13	other6	PIN_6	Row I/O		
14	other7	PIN_7	Row I/O		
15	other8	PIN_9	Row I/O		
16	other9	PIN_12	Row I/O		
17	other10	PIN_13	Row I/O		
18	other11	PIN_14	Row I/O		
19	<<new>>	<<new>>			

Obr.4: Okno priradenia pinov

Zadefinovaním všetkých výstupných signálov okno **Pins** uložíme a zavrieme. Týmto máme celý základný návrh hotový.

4.3.6 KOMPILÁCIA

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

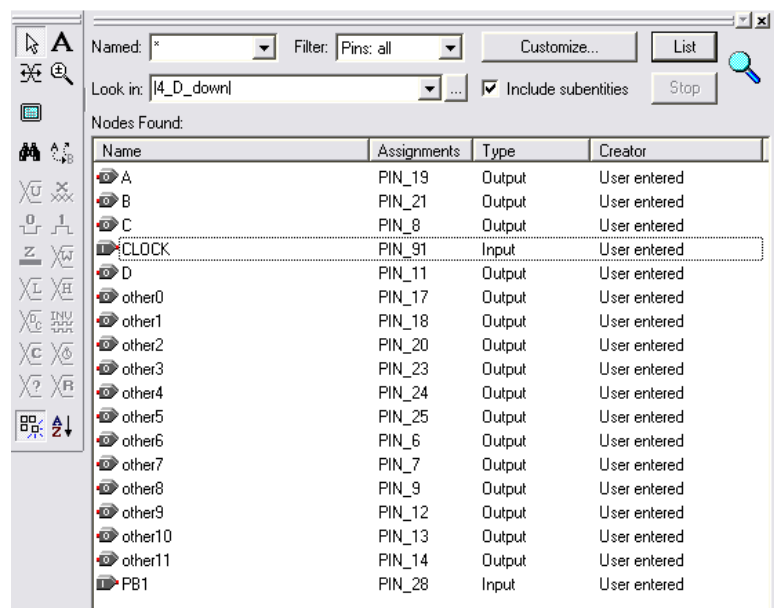
4.3.7 SIMULÁCIA

4.3.7.1 ČASOVÁ SIMULÁCIA

Po bezchybnej kompilácii môžeme prejsť k simulácii projektu. Simuláciu vykonáme podľa nasledujúceho postupu:

- Vytvoríme vektorový súbor, v ktorom si zadefinujeme tzv. stimuly (stimuly sú vstupné priebehy signálov, ktoré spôsobia požadované prechodové zmeny v realizácii) – *Vector Waveform file (.vwf)*, voľbou **New** z **File menu**
- Voľbou **Save As** z **File menu** uložíme tento vektorový súbor, v tomto prípade s názvom *4_D_down.vwf*.

- Pomocou **Node Finder** (obr.5) vložíme do tohto súboru všetky vstupy a výstupy, ktoré chceme simulovať. **Node Finder** otvoríme nasledujúcim postupom: **View**→**UtilityWindows**→**Node Finder**. Označíme požadované piny (Shift + ľavé tlačidlo myši), skopírujeme ich (Ctrl + C) a vložíme do vektorového súboru (Ctrl+V). Druhou možnosťou je chytiť myšou požadovaný pin a jednoducho ho presunúť do vektorového súboru (.vwf). V našom prípade presunieme piny: Clock, PB1, A, B, C a D.
- Nestavíme koncový čas simulácie. V položke **Edit**→**End Time** nastavíme hodnotu time na 100 μ s.



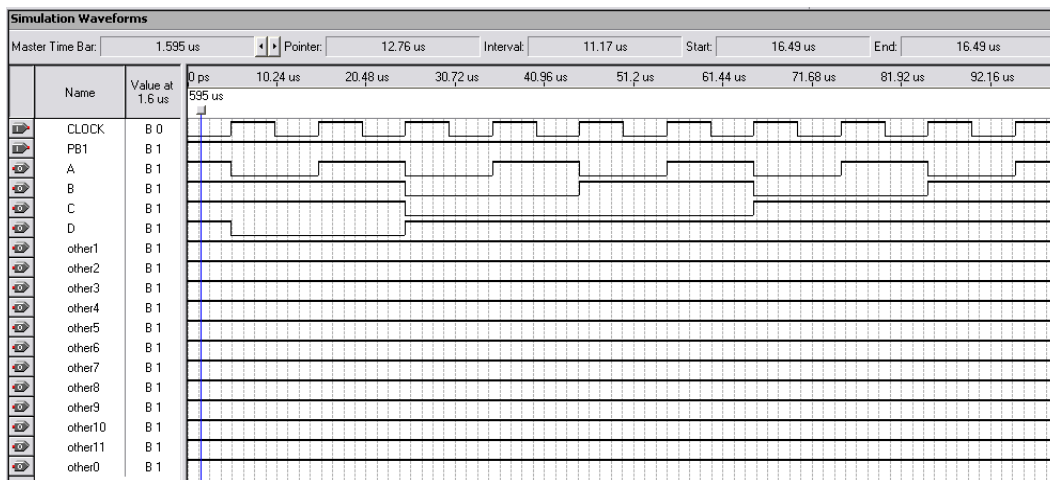
Obr.5: Node Finder

Časovú simuláciu projektu vykonáme bez bloku deličky hodinových impulzov. Keby sme ju nechali zapojenú museli by sme nastaviť veľmi dlhý čas simulácie aby sme mali odsimulované všetky stavy čítača. Keby sme simulovali s frekvenciou 1Hz potrebovali by sme nastaviť dlhý čas simulácie na to, aby sme simulovali každý možný stav. Preto namiesto signálu zo systémových hodín použijeme stimul – vstupný signál definovaný pri simulácii a pripojený na hodinové vstupy D klopných obvodov.

- Tlačidlo PB1 nastavíme trvalo na logickú úroveň „1“. Klikneme pravým tlačidlom na PB1 a vyberieme **Value**→**Forcing High(1)**.
- Vstup Clock nastavíme podľa nasledujúceho postupu:

Kliknutím pravého tlačidla myši na pin *CLOCK* vyberieme v tabuľke ktorá sa objaví **Value**→**Count Value**. V zobrazenom okne v záložke **Timing** definujeme hodnoty **End Time** 100µs a **Count every** 5µs

- Všetky nastavenia uložíme voľbou **Save** z **File menu**, alebo kliknutím na ikonu na panely nástrojov.
- Po uložení nastavení spustíme simuláciu voľbou **Start Simulation** z **Processing menu**, alebo kliknutím na ikonu na panely nástrojov.
- Po bezchybnom prebehnutí simulácie dostaneme okno **Simulation Report** (obr.6), v ktorom sa zobrazia výsledky časovej analýzy.



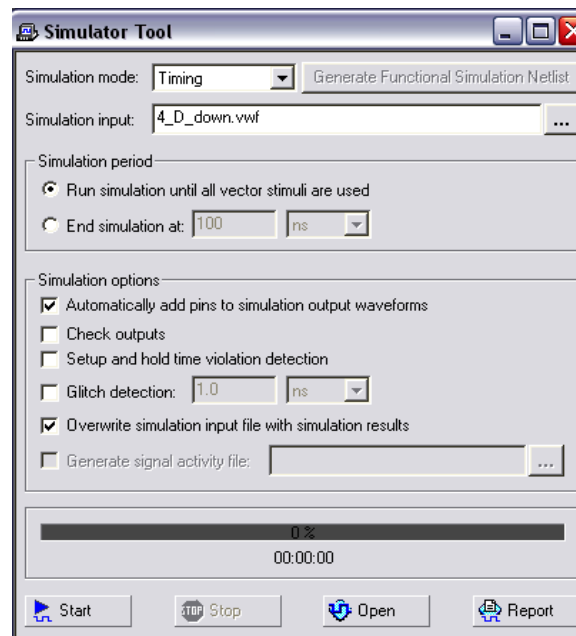
Obr.6: Výsledky časovej simulácie

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

4.3.7.2 FUNKČNÁ SIMULÁCIA

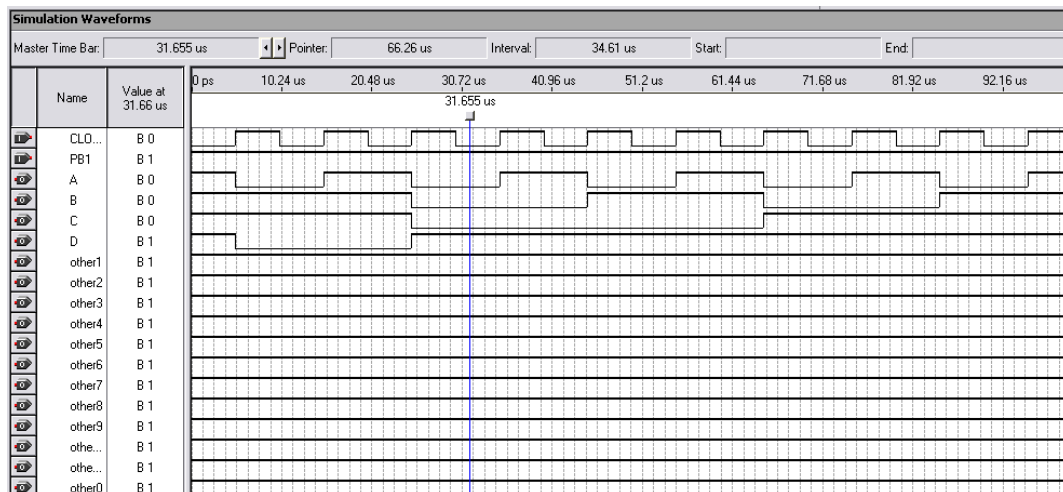
Pre funkčnú simuláciu použijeme tie isté nastavenia ako pre časovú simuláciu.

Otvoríme okno **Simulator Tool** výberom z **Tools** menu (obr.7).



Obr.7: Okno Simulator Tool

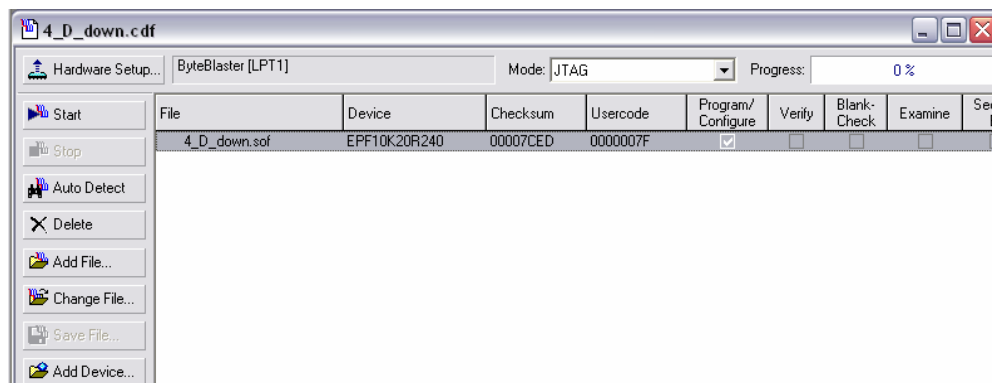
Ako prvé zmeníme **Simulation mode** (mód simulácie) na **Functional**. Potom klikneme na tlačidlo **Generate Functional Simulation Netlist**. Ak máme vygenerovaný **Netlist**, začneme simuláciu kliknutím na tlačidlo **Start**. Po skončení simulácie si môžeme výsledky pozrieť po kliknutí na tlačidlo **Report** (obr.8).



Obr.8: Výsledky funkčnej simulácie

4.3.8 PROGRAMOVANIE/KONFIGURÁCIA

Po úspešnej kompilácii a preverení projektu pomocou simulácie môžeme prejsť k programovaniu obvodu. Programovanie vykonáme výberom položky **Programmer** z **Tools menu** (obr.9). V tomto okne musíme zaškrtnúť políčko **Program/Configure** a kliknutím na tlačidlo **Start** spustíme proces programovania. Od tohto okamihu je projekt naprogramovaný do súčiastky na doske UP1 CPLD. Teraz môžeme preveriť činnosť nášho projektu priamo na doske.



Obr.9: Okno konfigurácie obvodu

4.4 ZADANIE PRÍKLADU Č.2

Navrhните 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 príklade číslo 1. 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 (analogicky ako v príklade č.1 - obr.1). Platí tá istá tabuľka pinov (tab.1). V zapojení tlačidlo PB1 využijeme na reset.

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 (tab.3).

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

Tab.3: Pravdivostná tabuľka čítača

Z pravdivostnej tabuľky a na základe matice prechodov (tab.4) zostavíme Karnaughove mapy pre jednotlivé vstupy J a K KO. Z nich dostaneme výsledné logické funkcie. Pomocou týchto funkcií zhotovíme výslednú schému.

$Q_n \rightarrow Q_{n+1}$	J	K
$0 \rightarrow 0$	0	X
$0 \rightarrow 1$	1	X
$1 \rightarrow 0$	X	1
$1 \rightarrow 1$	X	0

Tab.4: Matica prechodov

Z Karnaughových máp dostaneme výsledné logické funkcie v tvare:

$$J_A = 1 \quad (4.5)$$

$$J_B = A\bar{D} \quad (4.6)$$

$$J_C = AB \quad (4.7)$$

$$J_D = ABC \quad (4.8)$$

$$K_A = 1 \quad (4.9)$$

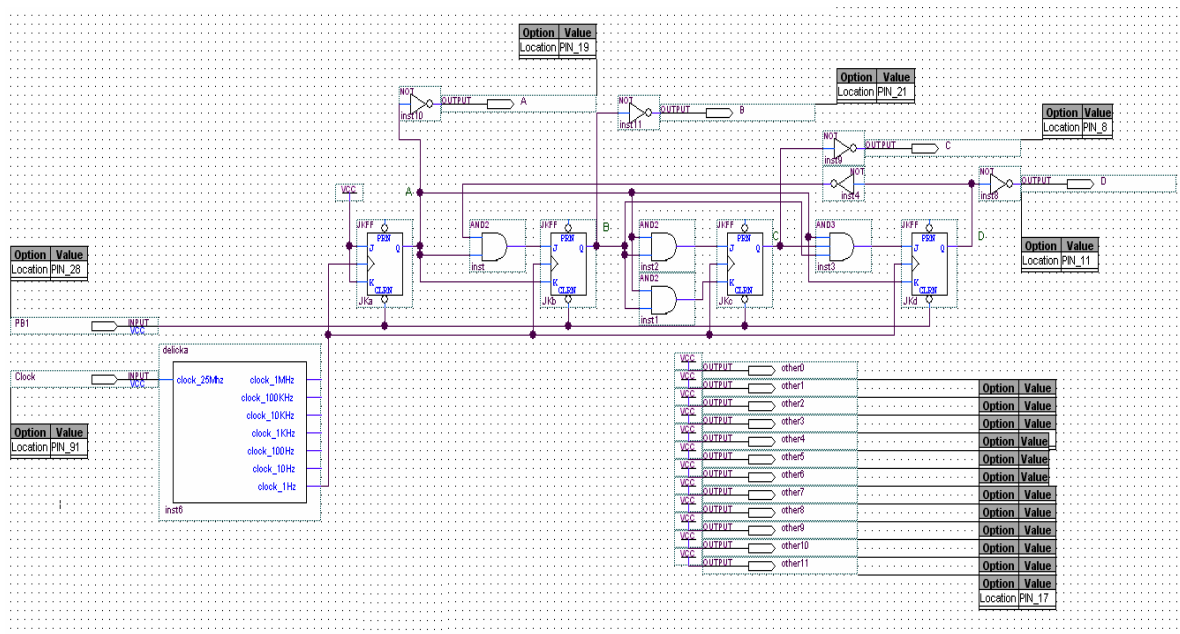
$$K_B = A \quad (4.10)$$

$$K_C = AB \quad (4.11)$$

$$K_D = A \quad (4.12)$$

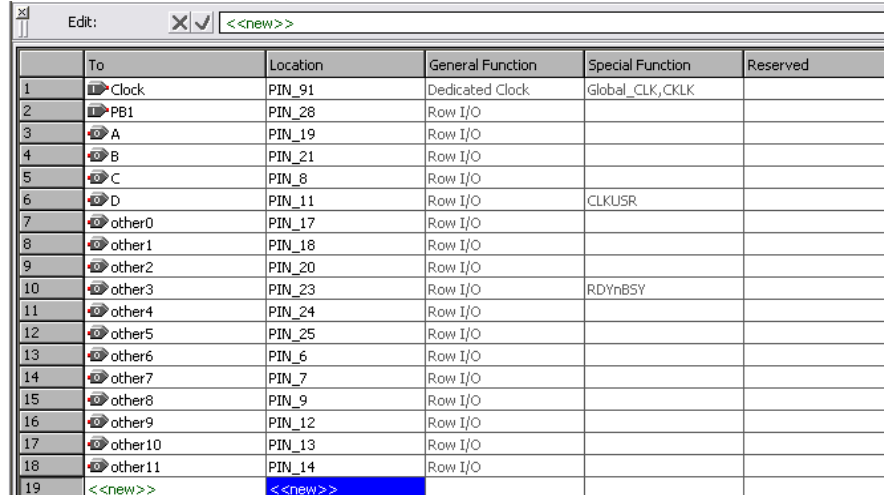
Pomocou rovníc (4.5) až (4.12) realizujeme hradlami AND vo výslednom zapojení logiku, ktorá slúži na získanie potrebných stavov počítadla.

V Quartus II otvoríme nový projekt s názvom *4_JK_up*. V grafickom okne realizujeme naše zapojenie zastavené zo štyroch JK klopných obvodov, hradiel AND, invertorov NOT, vstupných pinov a podobne ako v príklade číslo 1 pridáme aj obvod deličky hodinových impulzov. Ten vygenerujeme tým istým postupom ako v príklade č.1 z VHDL kódu (pozri Prílohu). Vzájomným prepojením týchto prvkov na základe vykonanej syntézy dostaneme výsledné zapojenie 4 bitového čítača vpred na báze JK klopných obvodov (obr.10). Vstupným a výstupným pinom pridáme názvy kliknutím pravého tlačidla na príslušnom pine a výberom položky **Properties**.



Obr.10: Výsledná schéma čítača

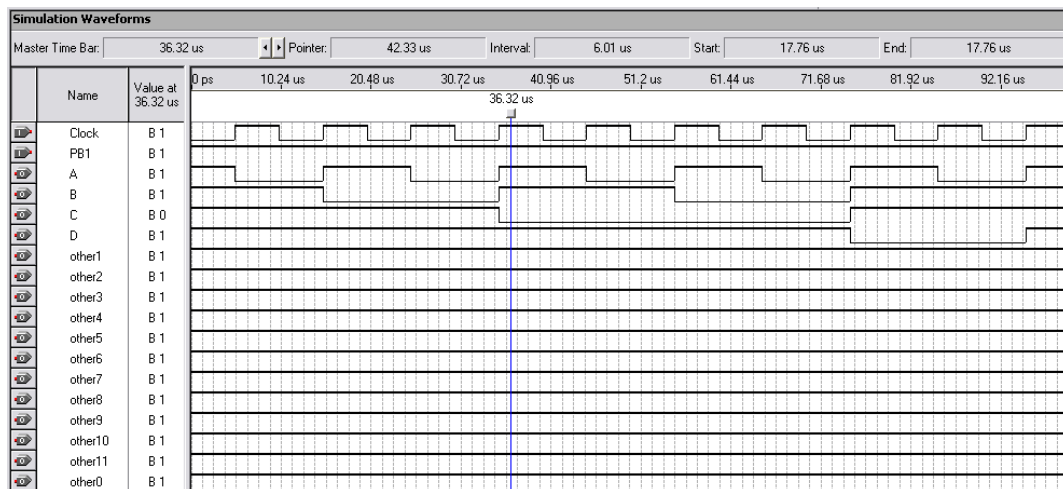
Pred samotnou kompiláciou nám ešte zostáva priradiť vstupným a výstupným signálom konkrétne čísla pinov obvodu podľa tab.1. Priradenie vykonáme v **Assigments menu**→**Pins** (obr.11). Výslednú schému uložíme.



	To	Location	General Function	Special Function	Reserved
1	Clock	PIN_91	Dedicated Clock	Global_CLK,CLK	
2	PB1	PIN_28	Row I/O		
3	A	PIN_19	Row I/O		
4	B	PIN_21	Row I/O		
5	C	PIN_8	Row I/O		
6	D	PIN_11	Row I/O	CLKUSR	
7	other0	PIN_17	Row I/O		
8	other1	PIN_18	Row I/O		
9	other2	PIN_20	Row I/O		
10	other3	PIN_23	Row I/O	RDYnBSY	
11	other4	PIN_24	Row I/O		
12	other5	PIN_25	Row I/O		
13	other6	PIN_6	Row I/O		
14	other7	PIN_7	Row I/O		
15	other8	PIN_9	Row I/O		
16	other9	PIN_12	Row I/O		
17	other10	PIN_13	Row I/O		
18	other11	PIN_14	Row I/O		
19	<<new>>	<<new>>			

Obr.11: Okno priradenia pinov

Po priradení pinov projekt skompilujeme. Ak prebehne kompletná kompilácia projektu bez chýb, môžeme prejsť k simulácii projektu. Časovú aj funkčnú simuláciu vykonáme podľa postupu uvedeného v príklade č.1. Vytvoríme si nový súbor, v ktorom definujeme stimuly - súbor s príponou *.vwf*. Uložíme ho pod názvom, v tomto prípade napr. *4_JK_up.vwf*. Použijeme ten istý postup simulácie, aj tie isté nastavenia. Výsledkom sú priebehy v okne Simulation Report (obr.12).



Obr.12: Výsledky časovej simulácie

Po úspešnej simulácii naprogramujeme projekt do súčiastky voľbou **Programmer** v **Tools menu**. Po naprogramovaní projektu do obvodu na vývojovej doske UP1 môžeme vykonať verifikáciu návrhu.

PRÍLOHA

V tejto prílohe je uvedený VHDL kód deličky použitej v predchádzajúcich príkladoch. Tento súbor je možné vytvoriť otvorením nového textového okna voľbou **New** z **File menu** a výberom **VHDL File**. Do otvoreného okna môžeme skopírovať tento text a súbor potom nahráť.

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
```

```
-- delenie 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;

-- Synchronizacia vystupov vsetkych hodin spatne k hlavnemu hodinovemu signalu
    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;

-- delenie 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;

-- delenie 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;

-- delenie 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;

-- delenie 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;
```

```
-- delenie 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;

-- delenie 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;
```