6 SEKVENČNÉ OBVODY VO VHDL

Opis hardvéru pomocou jazyka VHDL je pri práci na reálnych FPGA projektoch podstatne výhodnejší ako opis pomocou grafického editora. Grafický editor sa typicky využíva pri opise na vyšších úrovniach návrhu, napr. pri prepojení jednotlivých implementovaných blokov. V rámci cvičenia bude demonštrovaná resp. precvičovaná nasledujúca problematika:

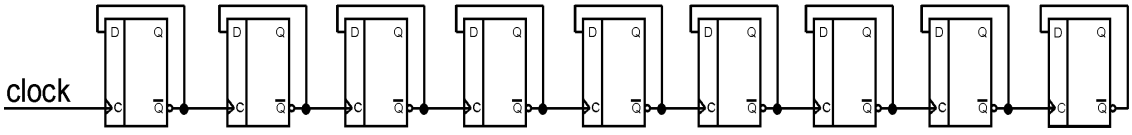
- výhoda jazyka VHDL oproti grafickému editoru,
- vytvorenie grafického symbolu pre návrh opísaný v jazyku VHDL,
- automatické generovanie VHDL kódu pre návrh v grafickom editore,
- časová simulácia v prostredí **Quartus II**, kritická cesta a maximálna taktovacia frekvencia,
- vplyv typu obvodov CPLD a FPGA na rýchlosť realizácie,
- využitie dvoch zdrojových VHDL kódov v jednom projekte.

6.1 POROVNANIE NÁVRHU VO VHDL A V GRAFICKOM EDITORE

Príklad 1
Navrhnite 9-bitový binárny čítač (asynchrónny alebo synchrónny) vpred s krokom +1. Realizujte návrh pomocou grafického editora a tiež pomocou VHDL kódu. Analyzujte ktorá z uvedených metód (grafický editor resp. VHDL) je z pohľadu užívateľa vhodnejšia (jednoduchšia) pre realizáciu 9-bitového binárny čítača vpred s krokom +3 a realizujte ho s využitím jednoduchšej metódy.

6.1.1 NÁVRH ČÍTAČA +1 S VYUŽITÍM GRAFICKÉHO EDITORA

Takýto čítač je možné realizovať vo forme asynchrónneho čítača s využitím D klopných obvodov zobrazených na obr.1. Realizácia synchrónneho čítača by bola podstatne komplikovanejšia, čo je zrejmé z predchádzajúcich cvičení. Na základe postupov preberaných v predchádzajúcich cvičeniach je možné realizovať kompletný návrh uvedeného čítača vrátane jeho simulácie.



Obr.1: Koncepčná schéma asynchrónneho čítača s krokom +1

6.1.2 NÁVRH ČÍTAČA +1 S VYUŽITÍM VHDL

Pri vytváraní čítača +1 v jazyku VHDL budeme vytvárať synchrónny čítač. Snaha je využiť čo najjednoduchší VHDL kód, čo v prípade VHDL vedie k synchrónnemu čítaču. Avšak ani vytvorenie asynchrónneho čítača vo VHDL nie je žiadnym problémom. V nasledujúcich častiach je uvedený detailný postup vytvorenia kompletného projektu s využitím jazyka VHDL. Uvedený postup bude možné využívať aj v ďalších cvičeniach s programom Quartus II.

6.1.2.1 OTVORENIE NOVÉHO PROJEKTU

Nový projekt otvoríme voľbou **New Project Wizard** z **File menu**. Ako prvé sa objaví úvodné okno, kde klikneme na tlačidlo **Next**. Objaví sa prvé okno, kde definujeme miesto uloženia, meno projektu a meno entity. Kliknutím na tlačidlo **...** sa objaví štruktúra adresárov, z ktorých vyberieme ten, do ktorého uložíme vytváraný projekt. Pre projekt si vytvoríme napr. adresár: *citac_D_up*. V ďalších riadkoch definujeme meno projektu a názov hlavnej úrovne **entity**. V našom prípade tam napíšeme *citac_D_up*. Tento údaj je dôležitý preto, že vo VHDL kóde sa tento názov musí zhodovať z názvom entity. Voľbu potvrdíme tlačidlom **Next**.

V prípade, že to je potrebné alebo výhodné, môžeme využiť určité bloky, resp. komponenty z iných projektov, môžeme ich pridať do vytváraného projektu v nasledujúcom okne.

Po stlačení **Next s**a objaví tretie okno voľby **New project Wizard**, v ktorom definujeme rodinu obvodov pomocou ktorých chceme projekt realizovať. V prípade funkčnej simulácie nezáleží na type obvodu, takže je jedno ktorú rodinu FPGA resp. CPLD obvodov zvolíme. Jediným obmedzením je skutočnosť, že zvolený obvod musí mať **dostatočnú kapacitu**, aby v ňom bolo možné návrh realizovať. Návrh pomocou VHDL je značne univerzálny a dá sa použiť pre rôzne typy cieľových obvodov. V tomto okne zaškrtneme možnosť *Auto device selected by the Fitter from the ´Available devices´ list*.

Kliknutím na tlačidlo **Next** sa otvorí štvrté okno voľby **New Project Wizard,** v ktorom je možné nastaviť ďalšie externé návrhové (EDA) nástroje od iných výrobcov. V ďalších cvičeniach napr. využijeme externý simulátor **Modelsim** od firmy Mentor Graphics. V rámci dnešného cvičenia však nebudeme využívať žiadne externé nástroje a teda voľby necháme prázdne. Stlačíme **Next.**

V poslednom okne, sú zobrazené všetky vybrané nastavenia. Ak s nimi súhlasíme potvrdíme ich tlačidlom **Finish**. Ak chceme niektoré údaje zmeniť, je možné vrátiť sa späť pomocou tlačidla **Back**.

6.1.2.2 VYTVORENIE A ZAČLENENIE VHDL NÁVRHU DO PROJEKTU

Postupujeme nasledovne:

- Vyberieme z **File Menu** položku **New.**
- V záložke **Design Files** si zvolíme **VHDL File.**
- Klikneme na tlačidlo **OK** a otvorí sa nám okno textového editora.

- Z **File Menu** vyberieme položku **Save As.**
- Vyberieme adresár *citac_D_up*, do ktorého uložíme náš súbor z názvom *citac_D_up.vhd.*
- Pod riadkom, kde sa definuje názov vkladaného súboru zaškrtneme políčko **Add file to curent project** (pridať súbor do aktuálneho projektu). Táto položka by mala byť implicitne zaškrtnutá.
- Kliknutím na tlačidlo **Save** sa vytvorený súbor uloží a zároveň je pridaný do projektu.

6.1.2.3 VYTVORENIE FUNKCIE ČÍTAČA V JAZYKU VHDL

Pri zostavovaní VHDL kódu čítača je potrebné najskôr zadefinovať knižnice štandardov ktoré budú použité. Potom nasleduje definovanie entity. Názov entity musí byť zhodný s názvom, ktorý sme zadefinovali pri otváraní nového projektu. Po definovaní entity nasleduje vytvorenie samotnej architektúry projektu. Ak si nepamätáme syntax jednotlivých príkazov jazyka VHDL, je možné použiť preddefinované predlohy (tzv. templates). Po výbere voľby **Insert Template** z **Edit menu** je možné v zobrazenom okne vybrať potrebnú štruktúru a potvrdiť jej výber kliknutím na tlačidlo **OK**. Príklady syntaxe vložených pomocou voľby **Insert Template**:

Deklarácia entity

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

Príkaz IF

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

Príkaz CASE

```
CASE __expression IS                             --prikaz Case
    WHEN __constant_value =>
            __statement;
            __statement;
    WHEN __constant_value =>
```

```
            __statement;
            __statement;
    WHEN OTHERS =>
            __statement;
            __statement;
END CASE;
```

Vybraná štruktúra bude automatický umiestnená do textového súboru na aktuálnu pozíciu kurzora. Vložením (prepísaním) požadovaných parametrov je možné výsledný kód prispôsobiť požadovanej funkcii.

Výsledný VHDL kód čítača +1:

```
--zadefinovanie 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 citac_D_up is
generic
    (
            N : integer:= 9
    );
port
    (
    clk     : in    std_logic;                          --hodiny
    vystup  : out   std_logic_vector (N downto 1)       --vystupny vektor
    );
end entity citac_D_up;


architecture citac of citac_D_up is
    signal vystup_sig   : std_logic_vector (N downto 1);        --pomocny signal
            begin
            process (clk)
                    begin
                            if (clk'event and clk='1') then
                            vystup_sig <= vystup_sig + 1;
                            end if;
            end process;
    vystup <= vystup_sig;
end architecture citac;
```

6.1.2.4 KOMPILÁCIA VHDL KÓDU

Po zostavení kompletného VHDL kódu je možné realizovať kompiláciu kódu pomocou voľby **Start Compilation** z **Processing menu**, pomocou ikony na paneli nástrojov alebo prostredníctvom voľby **Compiler Tool** z **Tools** menu. Počas kompilácie je možné zistiť prípadné chyby o VHDL kóde a po ich korekcii v editore úspešne zrealizovať kompiláciu.

6.1.3 POROVNANIE ZLOŽITOSTI POUŽÍTÝCH METÓD, NÁVRH ČÍTAČA +3

Druhá časť príkladu vyžaduje realizáciu čítača +3 s využitím jednoduchšej návrhovej metódy. Návrh čítača +3 v grafickom editore Quartus II vyžaduje vykonanie kompletnej syntézy čítača, určenie vhodných stavebných blokov (typov klopných

obvodov) a zostavenie výsledného zapojenia. Tieto kroky sú v prípade zložitejších čítačov pomerne pracné. Návrh čítača +3 vo VHDL kóde jasne demonštruje základnú výhodu opisu s využitím jazyka pre opis hardvéru. V kóde čítača +1 postačuje zmeniť jediný riadok kódu:

```
vystup_sig <= vystup_sig + 3;
```

Zvyšná časť VHDL kódu zostáva nezmenená. Plne sa prejavuje základná výhoda opisu vo VHDL. V opise definujeme **čo potrebujeme** realizovať a otázku **ako to realizovať** rieši kompilátor. Naviac, zmenu veľkosti čítača je vo VHDL kóde je možné realizovať zmenou jediného parametra

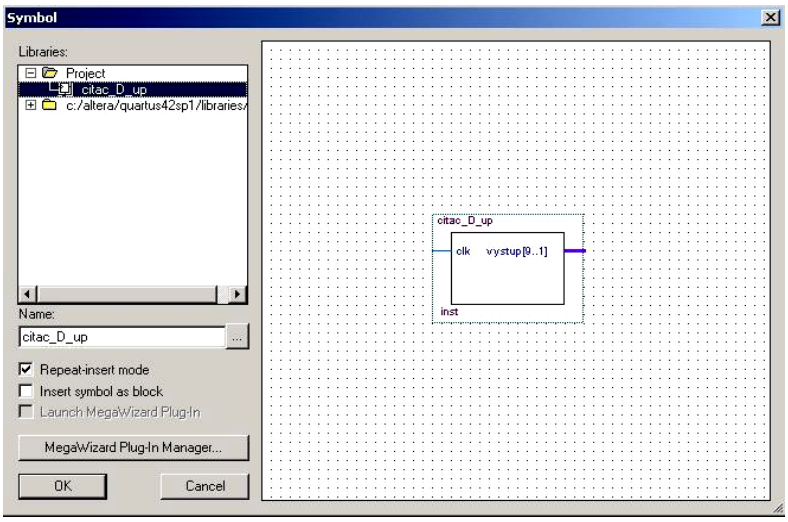
```
generic
    (
            N : integer:= 9
    );
port
```

čo je oproti grafickému návrhu obrovská výhoda. Zmeny veľkosti synchrónneho čítača v grafickom editore by bolo omnoho náročnejšia.

6.2 VYTVORENIE GRAFICKÉHO SYMBOLU PRE VHDL NÁVRH

Aj keď je návrh vo VHDL zvyčajne výhodnejší, na vyšších (abstraktnejších) úrovniach je často z dôvodu prehľadnosti využívaný grafický návrh. Je preto potrebné mať možnosť transformovať VHDL návrh do grafickej podoby. V tomto prípade však nie je potrebné vytvoriť „ekvivalentnú schému", ale len zodpovedajúcu blokovú schému resp. symbol.

V prostredí Quartus II je možné z VHDL kódu vygenerovať symbol tak, že súbor s príslušným VHDL kódom otvoríme a vyberieme možnosť **Create/Update→Create Symbol Files from Current File** z **File menu**. Po vytvorení je možné výsledný symbol zobraziť v **Symbol Tool**, položke **Project**. Symbol čítača navrhnutého pomocou VHDL je na obr.2. Pre jeho zobrazenie je potrebné pomocou voľby **New** z **File menu** otvoriť okno blokového návrhu.



Obr.2: Okno Symbol Tool

6.3 AUTOMATICKÉ GENEROVANIE VHDL KÓDU PRE NÁVRH V GRAFICKOM EDITORE

V prostredí Quartus II je možné realizovať aj opačný proces, t.j. z blokového návrhu vygenerovať VHDL kód. Ako vstup je potrebné použiť grafický návrh. Zostavíme schému pozostávajúcu z 9-tich D klopných obvodov, vstupných a výstupných pinov. Súbor uložíme pod menom napr. *citac_D_up_syntez*, vykonáme kompiláciu a potom vyberieme voľbu **Create/Update→Create HDL Design File from Current File** z **File menu**. Po výbere tejto voľby je potrebné zvoliť či má byť vygenerovaný kód v jazyku VHDL alebo Verilog[1]. Po voľbe jazyka VHDL potvrdíme OK. Po vygenerovaní si môžeme otvoriť VHDL súbor výberom voľby **Open** z **File menu**. Tento súbor bude mať názov *citac_D_up_syntez* a bude obsahovať generované VHDL príkazy v tvare:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY work;

ENTITY citac_D_up_syntez IS
    port
    (
            CLK :  IN  STD_LOGIC;
            vystup1 :  OUT  STD_LOGIC;
            vystup2 :  OUT  STD_LOGIC;
            vystup3 :  OUT  STD_LOGIC;
            vystup4 :  OUT  STD_LOGIC;
```

[1] Jazyk Verilog je druhým v praxi často využívaným jazykom pre opis hardvéru. Je širšie využívaný predovšetkým v USA. V Európe je obľúbenejší jazyk VHDL. Uvedené rozdelenie nie je samozrejme možné chápať doslovne.

```
        vystup5 :  OUT  STD_LOGIC;
        vystup6 :  OUT  STD_LOGIC;
        vystup7 :  OUT  STD_LOGIC;
        vystup8 :  OUT  STD_LOGIC;
        vystup9 :  OUT  STD_LOGIC
    );
END citac_D_up_syntez;

ARCHITECTURE bdf_type OF citac_D_up_syntez IS

signal      SYNTHESIZED_WIRE_26 :  STD_LOGIC;
signal      SYNTHESIZED_WIRE_27 :  STD_LOGIC;
signal      SYNTHESIZED_WIRE_3 :  STD_LOGIC;
signal      SYNTHESIZED_WIRE_4 :  STD_LOGIC;
signal      SYNTHESIZED_WIRE_5 :  STD_LOGIC;
signal      SYNTHESIZED_WIRE_6 :  STD_LOGIC;
signal      SYNTHESIZED_WIRE_7 :  STD_LOGIC;
signal      SYNTHESIZED_WIRE_8 :  STD_LOGIC;
signal      SYNTHESIZED_WIRE_9 :  STD_LOGIC;
signal      SYNTHESIZED_WIRE_10 :  STD_LOGIC;
signal      SYNTHESIZED_WIRE_28 :  STD_LOGIC;
signal      SYNTHESIZED_WIRE_29 :  STD_LOGIC;
signal      SYNTHESIZED_WIRE_30 :  STD_LOGIC;
signal      SYNTHESIZED_WIRE_31 :  STD_LOGIC;
signal      SYNTHESIZED_WIRE_32 :  STD_LOGIC;
signal      SYNTHESIZED_WIRE_33 :  STD_LOGIC;
signal      SYNTHESIZED_WIRE_24 :  STD_LOGIC;
signal      SYNTHESIZED_WIRE_25 :  STD_LOGIC;

BEGIN
vystup1 <= SYNTHESIZED_WIRE_10;
vystup2 <= SYNTHESIZED_WIRE_9;
vystup3 <= SYNTHESIZED_WIRE_7;
vystup4 <= SYNTHESIZED_WIRE_8;
vystup5 <= SYNTHESIZED_WIRE_6;
vystup6 <= SYNTHESIZED_WIRE_5;
vystup7 <= SYNTHESIZED_WIRE_4;
vystup8 <= SYNTHESIZED_WIRE_3;
vystup9 <= SYNTHESIZED_WIRE_25;

process(CLK)
begin
if (rising_edge(CLK)) then
    SYNTHESIZED_WIRE_25 <= SYNTHESIZED_WIRE_26;
end if;
end process;

process(SYNTHESIZED_WIRE_26)
begin
if (rising_edge(SYNTHESIZED_WIRE_26)) then
    SYNTHESIZED_WIRE_3 <= SYNTHESIZED_WIRE_27;
end if;
end process;

SYNTHESIZED_WIRE_27 <= NOT(SYNTHESIZED_WIRE_3);
SYNTHESIZED_WIRE_28 <= NOT(SYNTHESIZED_WIRE_4);
SYNTHESIZED_WIRE_29 <= NOT(SYNTHESIZED_WIRE_5);
SYNTHESIZED_WIRE_30 <= NOT(SYNTHESIZED_WIRE_6);
SYNTHESIZED_WIRE_32 <= NOT(SYNTHESIZED_WIRE_7);
SYNTHESIZED_WIRE_31 <= NOT(SYNTHESIZED_WIRE_8);
SYNTHESIZED_WIRE_33 <= NOT(SYNTHESIZED_WIRE_9);
SYNTHESIZED_WIRE_24 <= NOT(SYNTHESIZED_WIRE_10);

process(SYNTHESIZED_WIRE_27)
begin
if (rising_edge(SYNTHESIZED_WIRE_27)) then
    SYNTHESIZED_WIRE_4 <= SYNTHESIZED_WIRE_28;
end if;
```

```
end process;

process(SYNTHESIZED_WIRE_28)
begin
if (rising_edge(SYNTHESIZED_WIRE_28)) then
    SYNTHESIZED_WIRE_5 <= SYNTHESIZED_WIRE_29;
end if;
end process;


process(SYNTHESIZED_WIRE_29)
begin
if (rising_edge(SYNTHESIZED_WIRE_29)) then
    SYNTHESIZED_WIRE_6 <= SYNTHESIZED_WIRE_30;
end if;
end process;

process(SYNTHESIZED_WIRE_30)
begin
if (rising_edge(SYNTHESIZED_WIRE_30)) then
    SYNTHESIZED_WIRE_8 <= SYNTHESIZED_WIRE_31;
end if;
end process;

process(SYNTHESIZED_WIRE_32)
begin
if (rising_edge(SYNTHESIZED_WIRE_32)) then
    SYNTHESIZED_WIRE_9 <= SYNTHESIZED_WIRE_33;
end if;
end process;

process(SYNTHESIZED_WIRE_31)
begin
if (rising_edge(SYNTHESIZED_WIRE_31)) then
    SYNTHESIZED_WIRE_7 <= SYNTHESIZED_WIRE_32;
end if;
end process;

process(SYNTHESIZED_WIRE_33)
begin
if (rising_edge(SYNTHESIZED_WIRE_33)) then
    SYNTHESIZED_WIRE_10 <= SYNTHESIZED_WIRE_24;
end if;
end process;

SYNTHESIZED_WIRE_26 <= NOT(SYNTHESIZED_WIRE_25);

END;
```

Z uvedeného VHDL kódu je jasne vidno, že kód je generovaný počítačom. Je tiež na prvý pohľad pomerne komplikovaný. Dokumentuje to fakt, že vo všeobecnosti nie je možné očakávať, že jednoduchému návrhu v grafickom editore musí po automatickom vygenerovaní počítačom zodpovedať jednoduchý VHDL kód. To isté samozrejme platí aj opačne, čo je vidno na príklade čítača +3 vo VHDL kóde, kde kód VHDL je veľmi jednoduchý, grafická schéma 9-bitového synchrónneho čítača by bola značne zložitejšia.

6.4 ČASOVÁ SIMULÁCIA V PROSTREDÍ QUARTUS II, MAXIMÁLNA TAKTOVACIA FREKVENCIA A KRITICKÁ CESTA

Časová simulácia zvyčajne nasleduje po funkčnej simulácii. Keďže časová simulácia je výrazne **časovo náročnejšia**, pri praktickom návrhu je výhodné overiť návrh najskôr pomocou funkčnej simulácie. Niektoré simulačné nástroje (napr. Modelsim) umožňujú realizovať funkčnú simuláciu bez voľby cieľovej súčiastky resp. rodiny cieľových obvodov.

6.4.1 FUNKČNÁ SIMULÁCIA

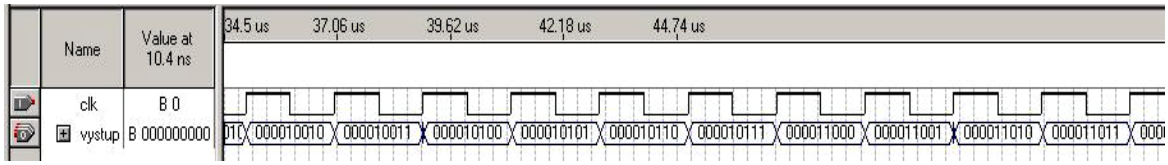
Výberom voľby **New** z **File menu** otvoríme nové vektorové okno (**.vwf - Vector Waveform File**) a príslušný .vwf súbor uložíme. Vo .vwf súbore zadefinujeme stimuly pomocou **Node Finder** z **View→ UtilityWindows**. Do .vwf súboru vložíme clk vstup a výstupný vektor. Po uzavretí voľby **Node Finder** zadefinujeme v položke **Edit→End Time** koncový čas simulácie. Nastavíme ho na napr. na hodnotu 100 μs. Kliknutím pravým tlačidlom myši na hodiny clk vyberieme voľbu **Count→Count Value**. Tam nastavíme v záložke **Timing** položku **End time** na 100 μs a položku **Count Event** na 1 μs (je samozrejme možné nastaviť aj iné časy). Všetky tieto nastavenia uložíme. Po otvorení okna **Simulator Tool** z **Tools menu** je v ňom možné meniť časovú resp. funkčnú simuláciu (**timing→functional**). Zvolíme **funkčnú simuláciu**. V ďalšom kroku je potrebné najskôr vygenerovať **Netlist**. Po jeho vygenerovaní je možné spustiť samotnú funkčnú simuláciu a to kliknutím na tlačidlo **Start**. Výsledkom bude okno s funkčnou simuláciou nášho projektu, ktoré je možné otvoriť kliknutím na tlačidlo **Open**.

6.4.2 ČASOVÁ SIMULÁCIA

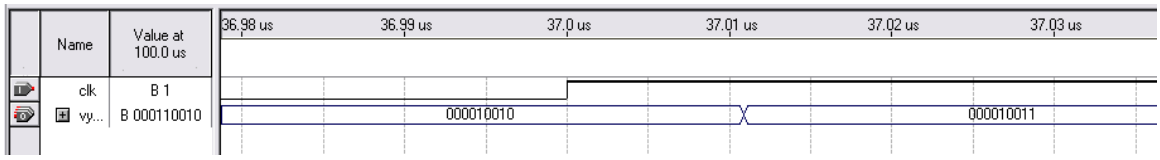
Časovú simuláciu je možné z pohľadu užívateľa realizovať podobným spôsobom ako funkčnú. Je možné použiť tie isté nastavenia a priamo tak nadviazať na výsledky funkčnej simulácie. Po otvorení okna **Simulator Tool** z **Tools menu** je možné zvoliť (zmeniť) funkčnú simuláciu (**functional→timing**). Spustenie časovej simulácie je možné vykonať kliknutím na tlačidlo **Start,** stlačením ikony na paneli nástrojov alebo výberom **Start Simulation** z **Proccesing menu**. Výsledkom je grafická reprezentácia časovej simulácie čítača zobrazená na obr.3 a obr.4.

Z obr.3 je možné overiť korektnú funkciu navrhnutého čítača +1. Na obr.4 je možné pozorovať oneskorenie medzi hranou vstupného hodinového signálu a výstupom čítača, ktoré je väčšie ako 10 ns. Tieto oneskorenia závisia na zložitosti realizovanej funkcie, spôsobe ich realizácie a samozrejme tiež na rodine a type použitých CPLD resp. FPGA obvodov. V rámci konkrétneho typu obvodu je možné vybrať niekoľko súčiastok s rovnakým typovým označením a rôznymi rýchlosťami[2] (speed grade).

[2] Rýchlosť obvodu priamo ovplyvňuje jeho cenu. V praxi je preto výhodné realizovať finálny návrh s využitím pomalších verzií obvodov.



Obr.3: Časť výsledkov časovej simulácie čítača +1



Obr.4: Časová simulácia (zoom obr. 3)

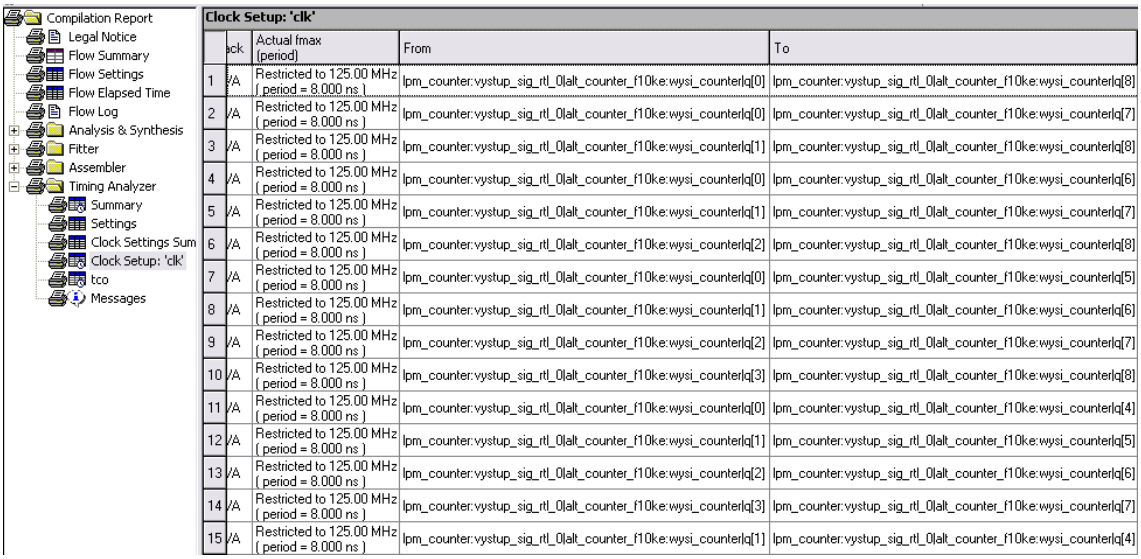
Cvičenie 1:
Vyhodnoťte časové oneskorenie výstupných signálov od vstupného hodinového signálu pre rôzne rodiny a rýchlostné varianty obvodov z rodín: MAX 7000, MAX II, Cyclone, Cyclone II, Stratix, Stratix II, ...

6.4.3 KRITICKÁ CESTA A MAXIMÁLNA TAKTOVACIA FREKVENCIA

Dôležitými parametrami návrhu je ich **kritická cesta a maximálna taktovacia frekvencia**. Kritická cesta umožňuje zistiť, ktorá časť návrhu je z časového hľadiska kritická. Vhodnou optimalizáciou návrhu je možné kritickú cestu „skrátiť", prípadne presunúť ju do úplne inej časti návrhu. Dĺžka kritickej cesty zároveň určuje maximálnu taktovaciu frekvenciu návrhu v analyzovanom (použitom) obvode.

Tieto parametre je samozrejme možné určiť aj v programe Quartus II. Po výbere rodiny obvodov CPLD alebo FPGA a výbere konkrétneho cieľového obvodu vo voľbe **Device** z **Assigments menu** spustíme kompiláciu. V okne **Compilation report** vyberieme voľbu **Clok Setup** v záložke **Timing Analyzer**. V pravej časti okna sa zobrazia kritické cesty a maximálne frekvencie, ktoré sú zobrazené na obr.5 pre vybraný obvod FLEX 10K.

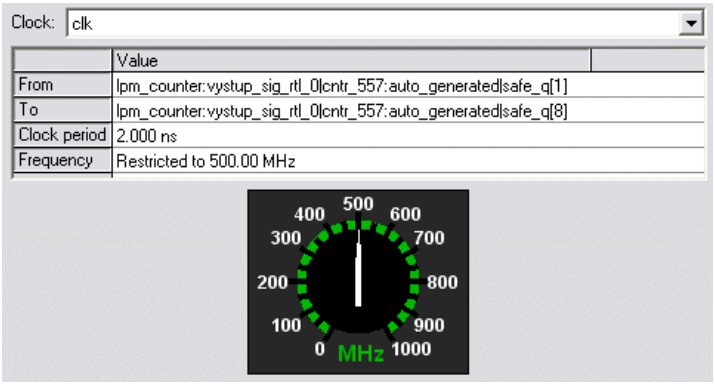
Maximálnu frekvenciu je možné zobraziť aj po výbere voľby **Timing Analyzer Tool** z **Tools menu**, ktorá je zobrazená na obr.6 pre vybrané obvody FLEX10K a Stratix II. Maximálna taktovacia frekvencia pre testovaný návrh čítača je pri rodine súčiastok FLEX10K 125 MHz a pri rodine súčiastok Stratix II až 500 MHz. Tieto výrazné rozdiely sú dané jednak odlišnou architektúrou oboch rodín a tiež použitou VLSI technológiou. Z maximálnych taktovacích frekvencií je jasne vidno, že výber rodiny cieľových obvodov má výrazný vplyv na dosiahnuteľné parametre.

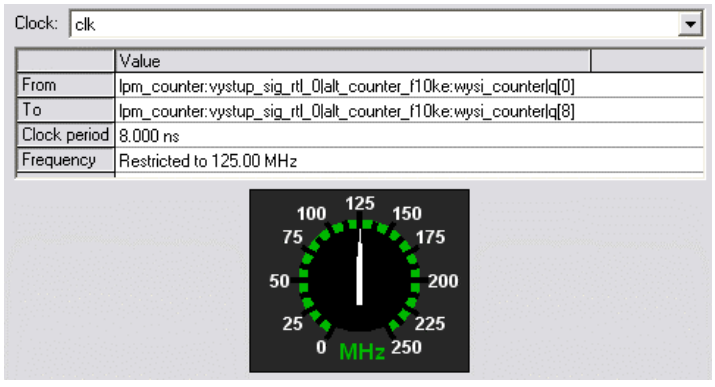


Clock Setup: 'clk'

	ock	Actual fmax (period)	From	To
1	N/A	Restricted to 125.00 MHz (period = 8.000 ns)	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[0]	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[8]
2	N/A	Restricted to 125.00 MHz (period = 8.000 ns)	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[0]	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[7]
3	N/A	Restricted to 125.00 MHz (period = 8.000 ns)	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[1]	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[8]
4	N/A	Restricted to 125.00 MHz (period = 8.000 ns)	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[0]	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[6]
5	N/A	Restricted to 125.00 MHz (period = 8.000 ns)	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[1]	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[7]
6	N/A	Restricted to 125.00 MHz (period = 8.000 ns)	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[2]	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[8]
7	N/A	Restricted to 125.00 MHz (period = 8.000 ns)	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[0]	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[5]
8	N/A	Restricted to 125.00 MHz (period = 8.000 ns)	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[1]	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[6]
9	N/A	Restricted to 125.00 MHz (period = 8.000 ns)	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[2]	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[7]
10	N/A	Restricted to 125.00 MHz (period = 8.000 ns)	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[3]	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[8]
11	N/A	Restricted to 125.00 MHz (period = 8.000 ns)	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[0]	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[4]
12	N/A	Restricted to 125.00 MHz (period = 8.000 ns)	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[1]	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[5]
13	N/A	Restricted to 125.00 MHz (period = 8.000 ns)	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[2]	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[6]
14	N/A	Restricted to 125.00 MHz (period = 8.000 ns)	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[3]	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[7]
15	N/A	Restricted to 125.00 MHz (period = 8.000 ns)	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[1]	lpm_counter:vystup_sig_rtl_0|alt_counter_f10ke:wysi_counter|q[4]

Obr.5: Maximálne frekvencie a kritické cesty pre FLEX10K





Obr.6: Maximálna taktovacia frekvencia pre FLEX10K (hore) pre STRATIX II (dole)

Cvičenie 2:
Určite kritické cesty a maximálne taktovacie frekvencie pre ďalšie rodiny súčiastok napr:
APEX20KC, MAX II, Cyclone, Cyclone II, ...

PRÍLOHY

Príklad 2
Navrhnite desiatkový asynchrónny čítač vzad/vpred v kóde BCD na báze D klopných obvodov a zrealizujte ho vo VHDL kóde.

Riešenie

Pravdivostná tabuľka čítača je zobrazená v Tab.1. Na základe pravdivostnej tabuľky zostavíme zodpovedajúci VHDL kód. Uvedený príklad je zložený z dvoch zdrojových VHDL kódov. Logicky oddelené bloky (samotný čítač a vstupná delička, ktorá umožňuje prípadné testovanie čítača pomocou vývojovej dosky UP1) sú uložené v rôznych VHDL súboroch. Umožňuje to výrazné zvýšenie prehľadnosti a modularity návrhu. VHDL kód deličky hodinových impulzov delí vstupný 25,175 MHz signál až na frekvenciu 1 Hz. Uvedený kód deličky bol použitý aj v predchádzajúcich cvičeniach a pre úplnosť je uvedený na konci príkladu 2.

Počet impulzov	Výstupy				Stav čítača
	vystup[4]	vystup[3]	vystup[2]	vystup[1]	
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.1: Pravdivostná tabuľka 4-bitového BCD čítača

Otvoríme nový projekt. Pre projekt vytvoríme adresár napr.: *4_D_down_vhdl*. V ďalších riadkoch definujeme meno projektu a názov hlavnej úrovne entity, napr. *stvorbit_D_down*. Pridáme súbor ktorý má názov *delicka.vhd*. Tento súbor skopírujeme do pracovného adresára.

Po otvorení projektu otvoríme nový textový súbor. Vyberieme adresár *4_D_down_vhdl*, do ktorého uložíme vytvorený súbor z názvom *stvorbit_D_down.vhd.*

Pri zostavovaní VHDL kódu postupujeme podobne ako v predchádzajúcom príklade. Na rozdiel od predchádzajúceho návrhu čítača +1 resp. +3, vytváraný čítač bude využívať „component" deličky, čo je vo výslednom VHDL kóde zabezpečené príkazom **component**. Kompletný opis navrhnutého čítača vo VHDL kóde obsahuje nasledujúce príkazy:

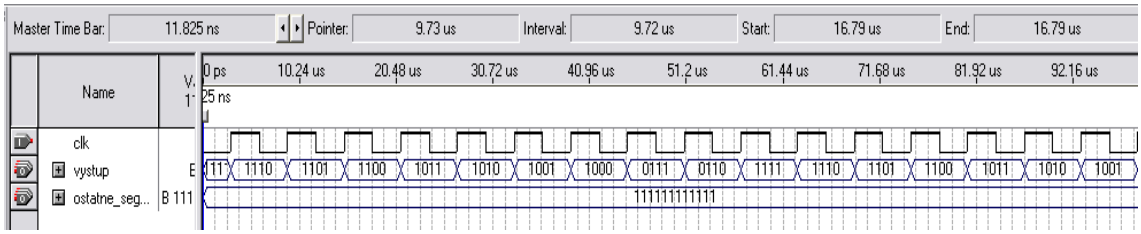
```
--zadefinovanie kniznic
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

--definovanie vstupov a vystupov
entity stvorbit_D_down is
port
    (
    clk                 : in    std_logic;                          --hodiny
    vystup              : out   std_logic_vector (4 downto 1);      --vystupny vektor
    ostatne_segmenty    : out   std_logic_vector (11 downto 0)      --nepouzite segmenty
    );
end entity stvorbit_D_down;

architecture tabulka of stvorbit_D_down is
    signal clk_1Hz      : std_logic;
    signal vystup_sig   : std_logic_vector (4 downto 1);

component delicka IS                            --vstupy a vystupy delicky
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 component;

begin
ostatne_segmenty  <= "111111111111";            --zhasnutie nepouzitych segmentov
delicka_inst : delicka
port map
            (
            clock_25Mhz => clk,
            clock_1Hz => clk_1Hz
            );

--samotne telo citaca
    process (clk_1Hz)
    begin
            if (clk_1Hz'event and clk_1Hz='1') then
                    if vystup_sig = "0000" then
                            vystup_sig <= "1001";
else
                            vystup_sig <= vystup_sig - '1';
                    end if;
            end if;
    end process;
```

```
vystup <=not vystup_sig;
end architecture tabulka;
```

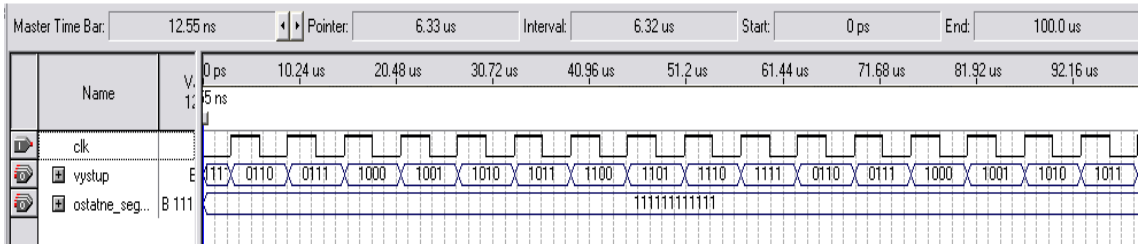
Ak je potrebné realizovať čítač vpred tak vo VHDL kóde stačí zmeniť dekrementáciu výstupu na inkrementáciu a ošetriť počiatočnú hodnotu počítania a nulovanie.

```
if (clk_1Hz'event and clk_1Hz='1') then
        if vystup_sig = "1001" then
                vystup_sig <= "0000";
        else
                vystup_sig <= vystup_sig + '1';
        end if;
end if;
```

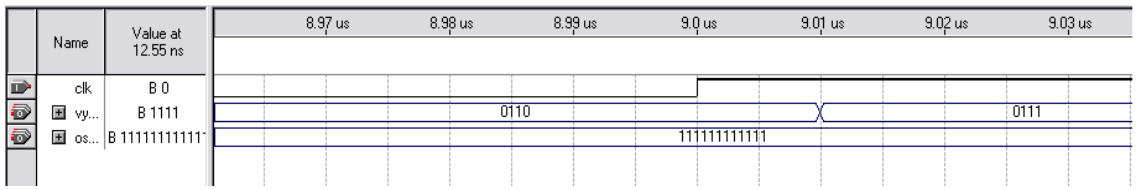
Po zostavení VHDL kódu a jeho kompilácii je možné vykonať funkčnú funkčnú (obr.7) a časovú simuláciu (obr.8) projektu podobne ako v predchádzajúcich príkladoch.



Obr.7: Výsledky funkčnej simulácie čítača vzad



Obr.8: Výsledky časovej simulácie čítača vpred



Obr.9: Výsledky časovej simulácie čítača vpred (zoom obr.8)

Výsledky simulácie potvrdzujú, že realizovaný čítač pracuje správne.

Cvičenie 3:
Pri simulácii VHDL kódu zmeňte použitý výstup deličky a otestujte vplyv jeho voľby na rýchlosť časovej a funkčnej simulácie.

VHDL kód deličky hodinového signálu:

```
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;
```