9. KNIŽNICA PARAMETRIZOVATEĽNÝCH MODULOV (LPM) A ALTERA MEGAFUNKCIE

Veľký nárast zložitosti FPGA obvodov vyžaduje pri návrhu využívanie efektívnejších návrhových postupov. Do tejto oblasti patria aj knižnice parametrizovateľných modulov (**LPM** – **L**ibrary of **P**arameterized **M**odules) a ďalších parametrizovateľných prvkov, pre ktoré firma Altera používa názov **Megafunkcie**. Začiatkom 90-tych rokov bola definovaná množina štandardných funkcií – LPM, ktoré LPM poskytuje knižnicu parametrizovateľých logických funkcii a modulov, ktoré sú nezávislé na architektúre cieľového (FPGA, ASIC) obvodu. Je tak umožnený prenos návrhov medzi obvodmi rôznych výrobcov obvodov pri zachovaní efektívnosti implementácie. Efektívnu implementáciu pre cieľový obvod zabezpečujú výrobcovia návrhových prostriedkov.

Existujú aj ďalšie parametrizovateľné moduly, u ktorých neexistuje možnosť priamej migrácie medzi rôznymi typmi obvodmi. Ich použitie však umožňuje výrazné zvýšenie efektivity návrhu a v prípade, že návrh je realizovaný len v rámci obvodov jedného výrobcu (v našom prípade Altery, ktorá pre tieto bloky používa termín Megafunkcie) nie je medzi LPM a Megafunkciami rozdiel. Prehľad vybraných LPM a Altera Megafunkcií je uvedený v **Prílohe I**.

V rámci cvičenia bude demonštrovaná resp. precvičovaná nasledujúca problematika:
- využitie LPM prvkov (*lpm_counter, lpm_rom*),
- konfigurácia pomocou MegaWizard Plug-In Managera,
- ladenie v prostredí Quartus II a Modelsim,
- využitie Altera Megafunkcie pre konfiguráciu obvodov PLL.

9.1 PARAMETRIZOVATEĽNÉ MODULY (LPM A MEGAFUNKCIE)

Tieto parametrizovateľné moduly poskytujú návrhárovi zdroj predpripravených základných častí digitálnych systémov, ako sú napr. hradlá, vstupno/výstupné komponenty, rôzne druhy pamätí a komponenty pre základné algebraické operácie (sčítačky, odčítačky, násobičky, deličky). Dané bloky sú parametrizovateľné, čo znamená, že návrhár môže voliť šírku vstupných, resp. výstupných signálov, šírku (veľkosť) operandov, s ktorými parametrizovateľná jednotka pracuje. Taktiež umožňujú

meniť ďalšie parametre modulu, ako je napr. v prípade PLL (Phase Locked Loop) veľkosť vstupnej, resp. výstupnej frekvencie. Moduly sú plne dokumentované a testované, čo umožňuje ich okamžité použitie[1]. Moduly sú optimalizované z hľadiska rýchlosti, ako aj z hľadiska veľkosti na najnižšej úrovni (podobný postup sa používa pri programovaní softvéru, keď niektoré časti programu sú optimalizované v asembleri a dostupné programátorovi zvyčajne dostupné programátorovi vo forme knižničných funkcií. Týmto opis digitálneho systému sa stáva špecifickým pre určitý typ cieľovej súčiastky resp. rodiny súčiastok, a teda nie je ho možné použiť ako univerzálny modul[2]. Na druhej strane však s minimálnym úsilím je možné získať efektívnu implementáciu.

Samotný modul je potrebné deklarovať ako komponent – postup je podobný ako pri využívaní vlastných komponentov pri modulárnom návrhu. V tele architektúry sú následne priradené (mapované) jednotlivé komponenty do cieľového návrhu. Pri návrhu je možné využiť **MegaWizard Plug-In Manager** v prostredí Quartus II na automatické vygenerovanie VHDL kódu. Tento postup umožňuje pohodlné definovanie parametrov a automatické generovanie korektného VHDL kódu. Uvedený postup je preto veľmi výhodný a bude využitý v rámci celého cvičenia. V praxi je samozrejme možné písať VHDL kód aj bez využitia **MegaWizard Plug-In Managera**.

9.1.1 VYUŽITIE MODULU *LPM_COUNTER*

Využitie modulu *lpm_counter* bude demonštrované na príklade jednoduchého čítača. Bude realizovaný kompletný návrhový cyklus od parametrizácie až po časovú simuláciu v Modelsime.

Príklad 1
S využitím LPM funkcie navrhnite čítač. Návrh odsimulujte v prostredí Quartus II a Modelsim.

9.1.1.1 PARAMETRIZÁCIA V PROSTREDÍ MEGAWIZARD PLUG-IN MANAGERA

Detailný postup parametrizácie v prostredí Quartus II:

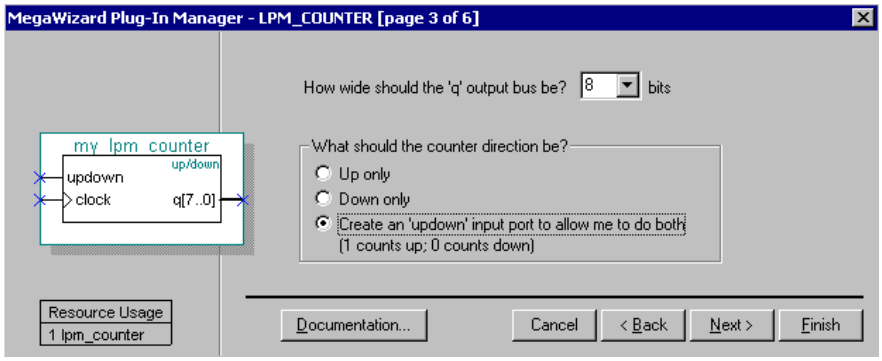
- Najskôr vytvoríme adresár v ktorom sa uloží vytváraný projekt (je výhodné nepoužívať medzery v ceste k projektu, resp. v názve projektu).

[1] Pre začiatočníkov však môže byť použitie niektorých zložitejších modulov neprehľadné a vyžaduje určitú prípravu.
[2] Platí to pre Altera Megafunkcie. LPM funkcie je možné využívať nezávisle na type cieľového obvodu (samozrejme len v prípade, že návrhové prostriedky pre cieľovú technológiu uvedený LPM modul podporujú).

- Štandardným postupom v prostredí Quartus II vytvoríme nový projekt a zadáme jeho umiestnenie do práve vytvoreného adresára, špecifikujeme cieľovú súčiastku (napr.) EPF10K20RC240-4, a keďže je potrebné vykonať aj časovú simuláciu projektu v ModelSime, zaškrtneme CheckBox označujúci *EDA simulation tool* a zvolíme *ModelSim (VHDL)*.

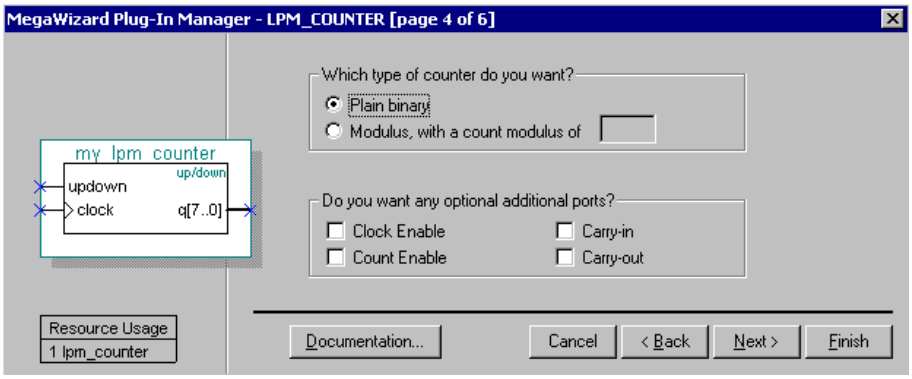
- Vyberieme požadovaný LPM modul (*lpm_counter*) a to tak, že z **Tools** menu Quartusu vyberieme **MegWizard Plug-In Manager**. Následne vyberieme **Create a new custom megafunction variation**. Spomedzi aritmetických megafunkcií vyberieme *LPM_COUNTER* a zadáme meno vytváranej LPM funkcie napr. *my_lpm_counter* (v okne **What name do you want for the output file?** za spätným lomítkom) a zvolíme VHDL ako popisný jazyk.

- V okne 3/6 definujeme, že vytvárame obojsmerný čítač (dokáže čítať vpred aj vzad, v závislosti na hodnote signálu *updown* – ak je definovaný ako log1, potom počíta vpred, inak počíta vzad (log0)) – veľkosť čítača ponecháme 8 bitov.



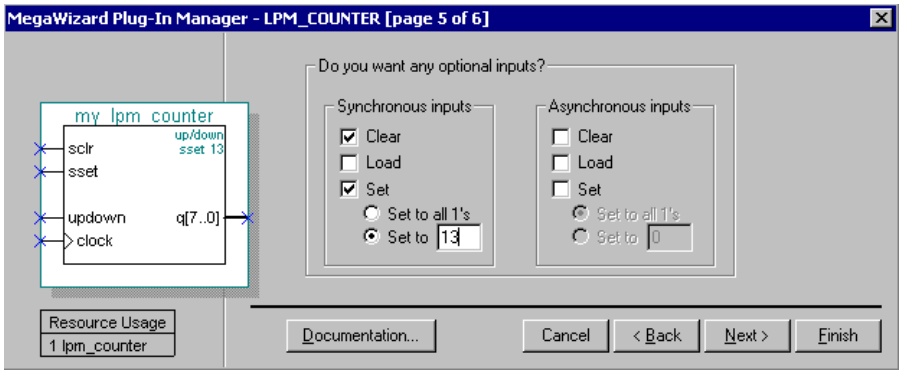
Obr.1 Okno 3/6 parametrizácie modulu lpm_counter

- V nasledujúcom okne je možné špecifikovať aký typ čítača zvolíme (binárny, modulo) a taktiež je možné určiť, či hodinový signál bude riadený signálom enable a pod ... (viď obr. 2).



Obr.2 Okno 4/6 parametrizácie modulu lpm_counter

- V okne 5/6 využijeme voľby **Clear** a **Set**, ktoré nám umožňujú vynulovať, resp. nastaviť určitý stav čítača (hodnotu na ktorú sa má čítač nastaviť definujeme v okne **Set to**, zadáme ju v decimálnom tvare), viď. obr. 3. V prípade ak zvolíme tieto signály z bloku **Synchronous inputs**, akcia nulovania, resp. nastavenia sa vykoná až v dobe nábežnej hrany hodinového signálu, ktorými sú tieto signály synchronizované. Ak sa rozhodneme pre asynchrónne vstupy, nulovanie, resp. nastavenie čítača sa vykoná okamžite pri prechode signálu z log0 do log1 (samozrejme že okamžite neznamená v čase t=0, ale s určitým oneskorením, ktoré závisí na danej cieľovej súčiastke).



Obr.3 Okno 5/6 parametrizácie modulu lpm_counter

- V predposlednom okne je možné špecifikovať, ktoré typy súborov sa majú vygenerovať – tieto voľby ponecháme bez zmeny. V nasledujúcom okne

vytvoríme LPM modul obojsmerného čítača s možnosťou nulovania a nastavenia hodnoty (v našom prípade hodnoty 13)

9.1.1.2 POUŽITIE VYTVORENÉHO LPM MODULU V PROJEKTE

Vytvorenú LPM funkciu je možné vo výslednom VHDL projekte využiť vo forme štandardného VHDL komponentu, teda kľúčovým slovom *COMPONENT*. Nasledujúci VHDL kód dokumentuje využitie LPM modulu čítača.

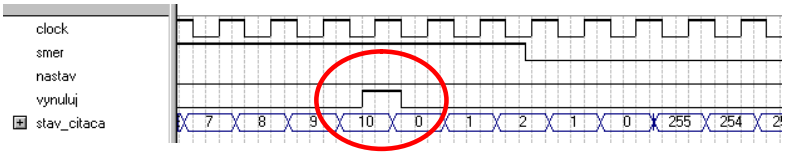
```
library ieee;
use ieee.std_logic_1164.all;

entity citac_clear_set is
        port
          (
          clock             : in      std_logic;
          smer              : in      std_logic;
          vynuluj           : in      std_logic;
          nastav            : in      std_logic;
          stav_citaca       : out     std_logic_vector (7 downto 0)
          );
end entity;

architecture arch of citac_clear_set is

        -- deklaracia lpm funkcie (modulu)
COMPONENT my_lpm_counter IS
                PORT
        (
                    clock     : IN STD_LOGIC ;
                    updown    : IN STD_LOGIC ;
                    sclr      : IN STD_LOGIC ;
                    sset      : IN STD_LOGIC ;
                    q         : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
        );
END COMPONENT;

        begin

        -- vytvorenie instancie komponentu
my_lpm_counter_inst: my_lpm_counter
                PORT MAP
        (
                    clock     => clock,
                    updown    => smer,
                    sclr      => vynuluj,
                    sset      => nastav,
                    q         => stav_citaca
        );

end architecture;
```

Daná *top-level* entita deklaruje LPM funkciu čítača v deklaračnej časti architektúry a v tele architektúry, začínajúcom kľúčovým slovom *BEGIN* vytvára inštanciu

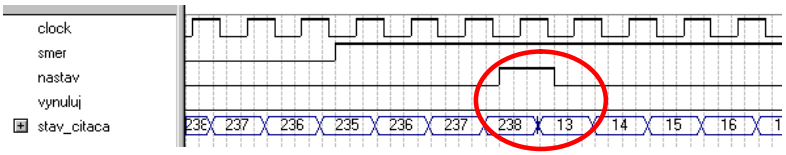
vytvorenej LPM funkcie. Jednotlivé signály vytvorenej LPM funkcie čítača sú mapované na vstupné, resp. výstupné signály entity.

9.1.1.3 SIMULÁCIA V QUARTUS II

Vytvorený čítač je samozrejme možné odsimulovať v prostredí Quartus II štandardným postupom. Vektor vstupných signálov je možné vytvoriť alebo je možné použiť súbor *citac_clear_set.vwf* v priloženom archíve k cvičeniu. Výsledok simulácie je zobrazený na obrázku 4 (pre lepšiu názornosť je rozdelený na dve časti):



nulovanie čítača



nastavenie čítača

Obr.4 Časová simulácia LPM funkcie čítača v prostredí Quartus II

9.1.1.4 ČASOVÁ SIMULÁCIA V MODELSIME

Postup časovej a funkčnej simulácie v ModelSime bol vysvetlený na jednom z predchádzajúcich cvičení. Výsledok časovej simulácie je zobrazený na Obr.5. Simulácia bola realizovaná s využitím nasledujúcich **.do* súborov (pre úplnosť je v **Prílohe II** uvedený detailný opis simulácie s využitím GUI prostredia Modelsimu).

 # do_first.do
vcom -work work -2002 -explicit -novitalcheck -no1164 -novital citac_clear_set.vho
vsim -sdftyp citac_clear_set_vhd.sdo work.citac_clear_set
add wave clock smer nastav vynuluj stav_citaca
write format wave wave.do
quit –sim

Dávkový súbor *do_first.do* vykonáme len raz. Jeho úlohou je vytvoriť vektorový súbor *wave.do*. Tento súbor by bolo možné vytvoriť aj prostredníctvom GUI rozhrania ModelSimu, ale na zautomatizovanie procesu bol vytvorený tento dávkový súbor obsahujúci potrebné príkazy.

Po vytvorení vektorového súboru *wave.do* (pomocou dávkového súboru *do_first.do*), je možné prejsť k časovej[3] simulácii návrhu. Tá je realizovaná ďalším dávkovým súborom *tsim.do*, ktorého obsah je následovný:

```
# tsim.do
quit -sim
vcom -work work -2002 -explicit -novitalcheck -no1164 -novital citac_clear_set.vho
vsim -sdftyp citac_clear_set_vhd.sdo work.citac_clear_set
view wave
do wave.do
do stimuls.do
```

Súbor *wave.do* bol automaticky vygenerovaný ModelSimom a jeho výpis je možné nájsť v doplnkových materiáloch k cvičeniu. Súbor *stimuls.do*, ktorý obsahuje definovanie vstupných signálov, teda ich tvary (je ich možné ľubovoľne meniť a pozorovať ako daný návrh reaguje na zmenu týchto signálov).

```
# stimuls.do
restart -f
force -freeze sim:/citac_clear_set/clock 1 0, 0 {40 ns} -r 80
force -freeze sim:/citac_clear_set/smer 1 0, 0 520, 1 1800
force -freeze sim:/citac_clear_set/nastav 0 0, 1 300, 0 340
force -freeze sim:/citac_clear_set/vynuluj 0 0, 1 780, 0 820
run 2000
```

[3] Samozrejme v Modelsime je možné realizovať aj funkčnú simuláciu.

Obr.5 Časová simulácia LPM funkcie čítača v prostredí Modelsim I

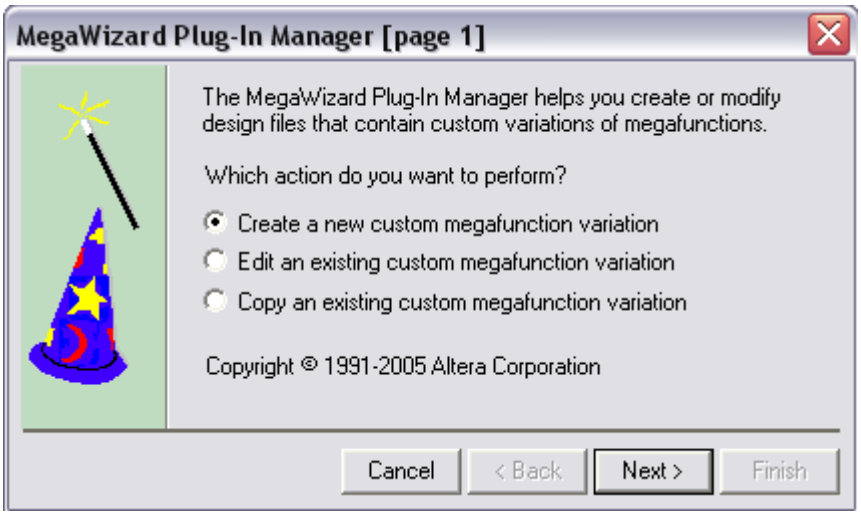
9.1.2 VYUŽITIE MODULU *LPM_ROM*

Príklad 2
S využitím LPM funkcie ROM pamäte vytvorte generátor znakov. Využite kombináciu návrhu vo VHDL a grafickom editore a realizujte simuláciu v prostredí Quartus II.

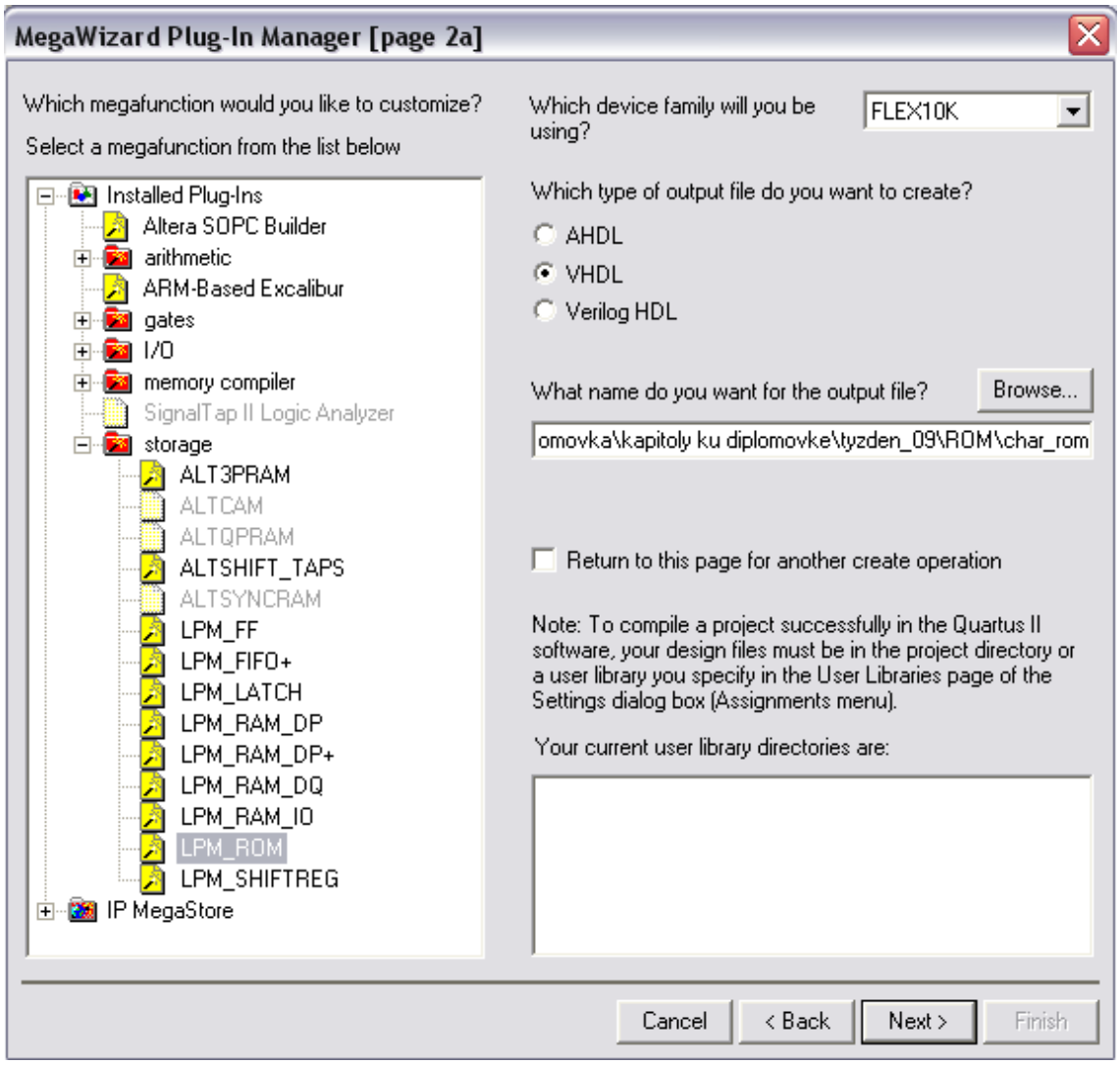
9.1.2.1 PARAMETRIZÁCIA V PROSTREDÍ MEGAWIZARD PLUG-IN MANAGERA

Stručný postup parametrizácie v prostredí Quartus II:

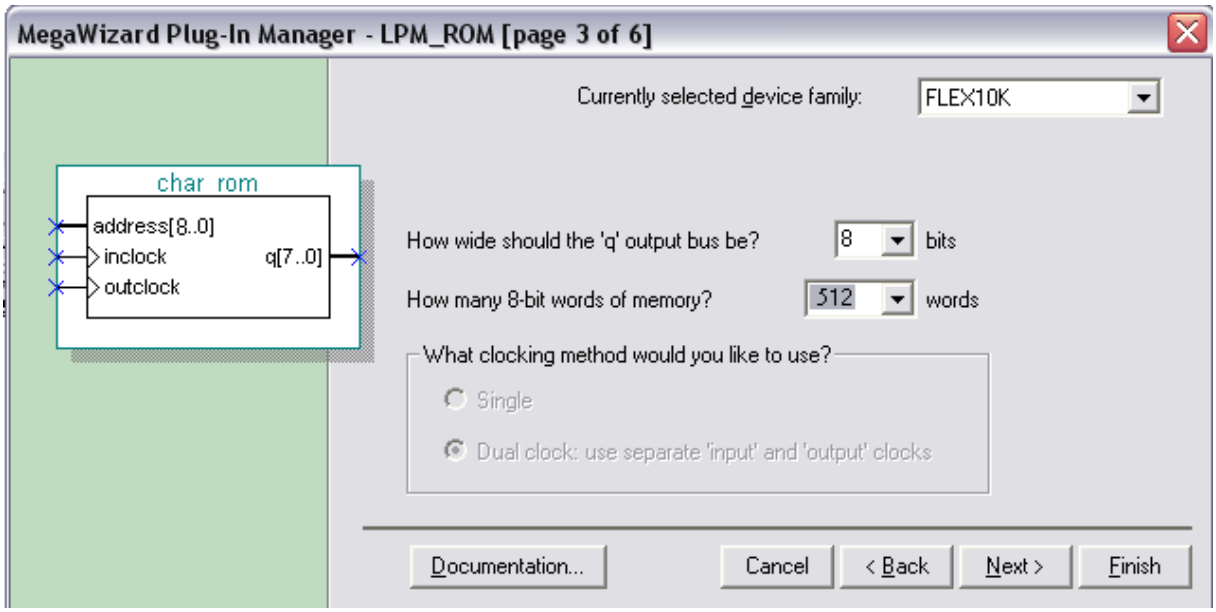
- Otvoríme nový projekt v Quartus II (zvolíme rodinu obvodov FLEX10K, meno entity napr. *rom_mem*).
- Otvoríme block diagram file (*.bdf*) a uložíme ho napr. pod menom *rom_mem*.
- Vyberieme voľbu z **Tools menu → MegaWizard Plug - In Manager**.
- V nasledujúcom okne špecifikujeme možnosť „**Create a new custom megafunction variation**“, klikneme na NEXT.



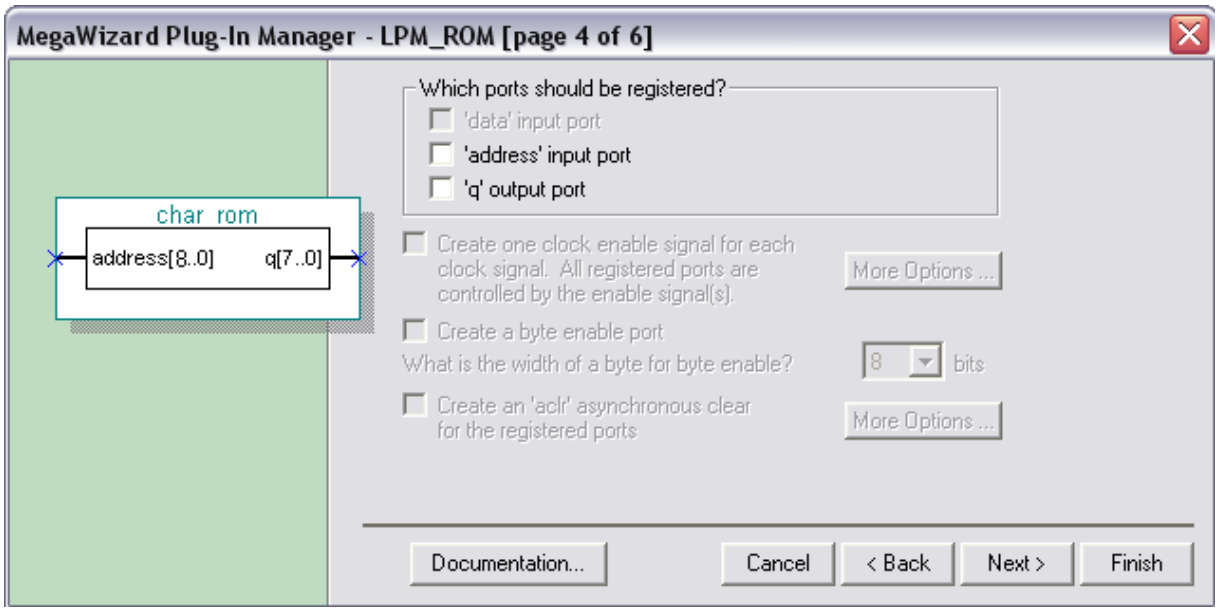
- V ďalšom okne zvolíme v riadku „**Which device family will you be using?**“ rodinu obvodov FLEX10K. Musí byť rovnaká s tou, čo sme zvolili pri otváraní nového projektu. V riadku „**Which type of output file do you want to create?**“ vyberieme možnosť VHDL.
- V riadku „**What name do you want for the output file?**“ dopíšeme meno výstupného súboru, ktoré sme si zvolili pre LPM funkciu. V knižnici **Storage** vyberieme možnosť **LPM_ROM**. Klikneme na **NEXT**.



- V nasledujúcom okne zvolíme veľkosť výstupnej zbernice (v našom prípade 8 bitov) a počet 8-bitových slov pamäte (v našom prípade 512). Klikneme na **NEXT**.

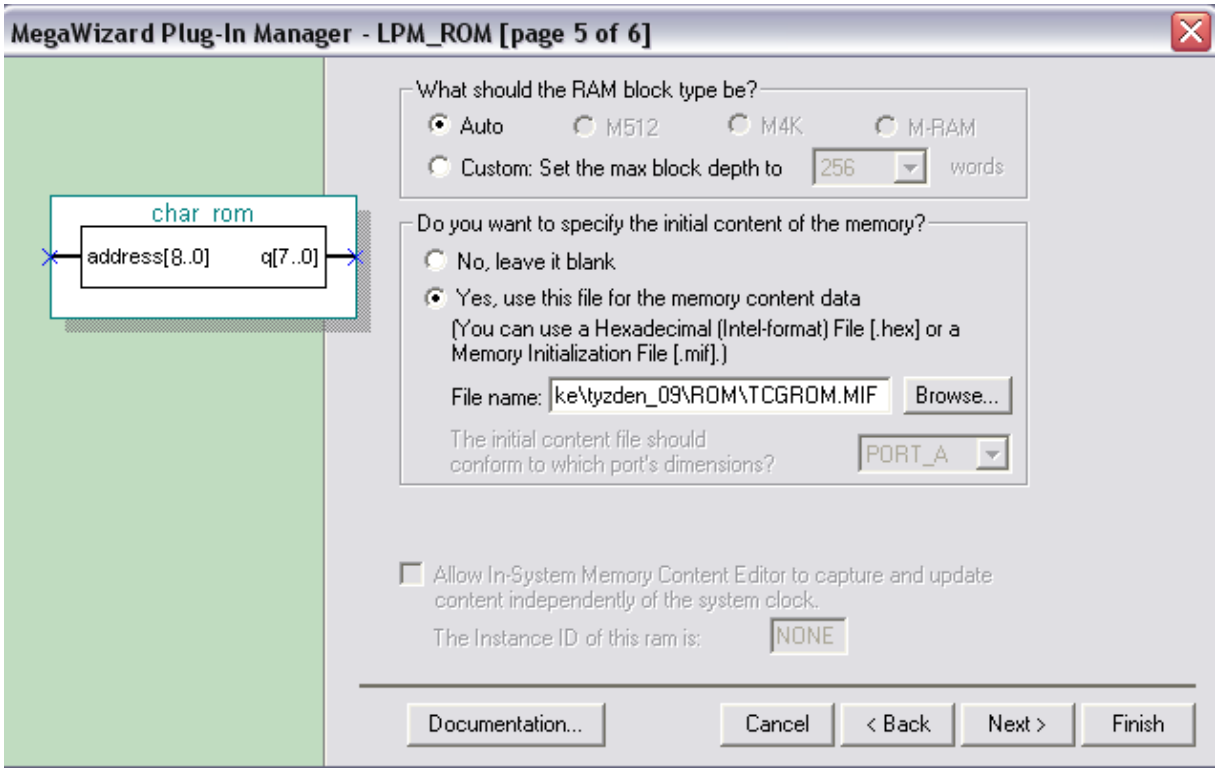


- Vo štvrtom okne zaškrtneme možnosť **„´address´ input port**“ a **„´q´ output port“**, klikneme na NEXT.



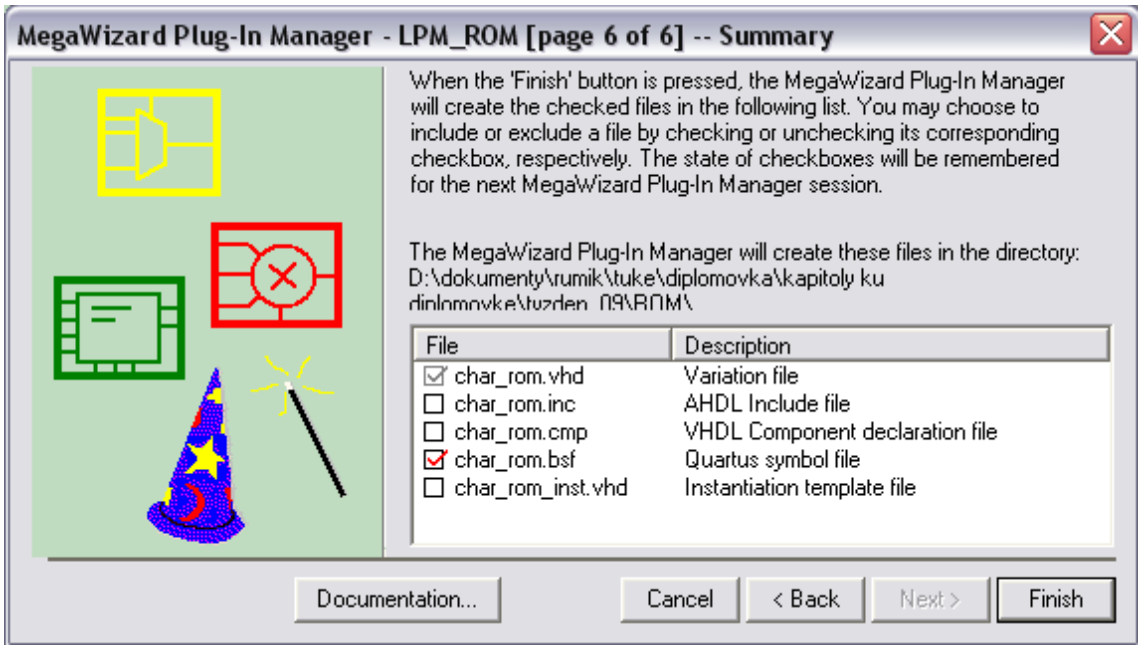
- V piatom okne špecifikujeme súbor ktorého dáta budú uložené v pamäti. V našom prípade do nášho projektu skopírujeme súbor z názvom

TCGROM.MIF[4]. V tomto okne **MegaWizard Plug-In Manager** nastavíme cestu k tomuto súboru. Klikneme na **NEXT**.



- V poslednom okne zaškrtneme ktoré typy súborov majú byť vygenerované pre zvolenú LPM funkciu. Implicitne je zaškrtnutá možnosť vytvorenia VHDL súboru. Keďže chceme realizovať aj grafický návrh, zaškrtneme aj možnosť *char_rom.bsf*. Klikneme na **FINISH**.

[4] Obsah súbory TCGROM.MIF je možné vytvoriť napr. ručne v ľubovoľnom editore, najlepšie však priamo v programe Quartus zvolením príslušného typu súboru v položke New. Pomocou editora je možné zistiť obsah jednotlivých adries ROM pamäte.



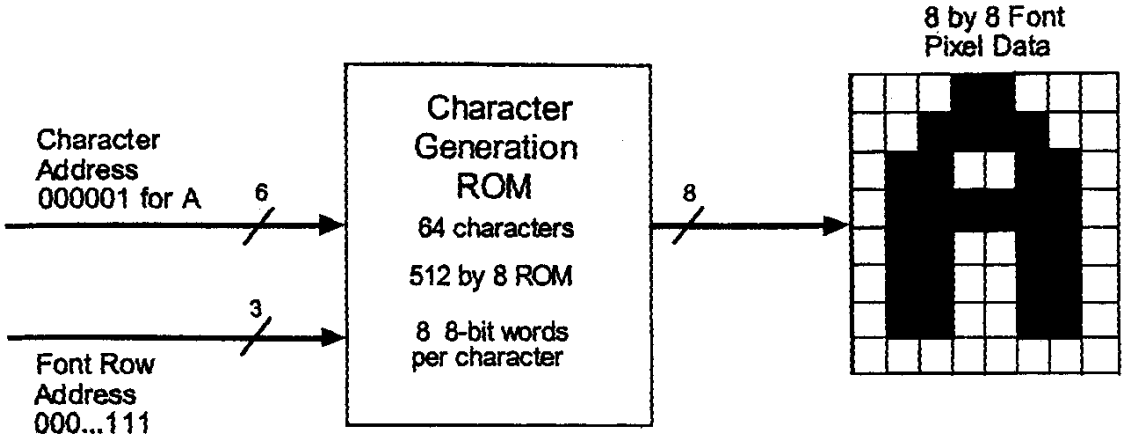
9.1.2.2 POUŽITIE VYTVORENÉHO LPM ROM MODULU V QUARTUS PROJEKTE

Vytvorený blok je možné v prostredí Quartus II použiť podobne ako vlastné bloky vytvorené na predchádzajúcich cvičeniach:

Stručný postup využitia LPM_ROM:

- s pomocou **Symbol Tool** vyberieme z knižnice **Project** vytvorenú pamäť ROM a vložíme ju na plochu,
- pripojíme vstupné a výstupné piny. Dvojitým kliknutím na príslušný pin zadefinujeme ich názvy (napr. IN[8..0], OUT[7..0]) a potvrdíme,
- celý projekt skompilujeme,
- simuláciu realizujeme štandardným spôsobom.

Pri vstupných kombináciách prvých 6-tich bitov reprezentuje adresu znaku (Tab.1 obsahuje informáciu o adresách všetkých uložených znakov) a posledné tri adresu riadku znaku. Na Obr.6 je znázornený obsah ROM pamäte pre znak „A“.



Obr.6 Grafická reprezentácia generátora znakov s LPM ROM modulom

Tabuľka 1: Tabuľka znakov a ich adresy (adresy sú zadané v OCT tvare)

Znak	Adresa (OCT)	Znak	Adresa (OCT)	Znak	Adresa (OCT)	Znak	Adresa (OCT)
@	00	P	20	SPACE	40	0	60
A	01	Q	21	!	41	1	61
B	02	R	22	‘‘	42	2	62
C	03	S	23	#	43	3	63
D	04	T	24	$	44	4	64
E	05	U	25	%	45	5	65
F	06	V	26	&	46	6	66
G	07	W	27	‘	47	7	67
H	10	X	30	(	50	8	70
I	11	Y	31	)	51	9	71
J	12	Z	32	*	52	A	72
K	13	[	33	+	53	B	73
L	14	šípka dole	34	,	54	C	74
M	15	]	35	-	55	D	75
N	16	šípka hore	36	.	56	E	76
O	17	šípka vľavo	37	/	57	F	77

Informácie o adrese znaku a adrese riadku znaku sú uložené v súbore z príponou *.mif* (v našom prípade *TCGROM.MIF*). Dáta v ROM pamäti sú pre znak „A“ zapísané podľa nasledujúceho priradenia:

Adresa	Dáta
000001000:	000**11**000;
000001001:	00**111**00;
000001010:	0**1**00**11**0;
000001011:	0**111111**0;
000001100:	0**1**00**11**0;
000001101:	0**1**00**11**0;
000001110:	0**1**00**11**0;
000001111:	00000000;

9.1.3 VYUŽITIE *LPM_COUNTER, LPM_ROM A PLL MEGAFUNKCIE*

Na nasledujúcom príklade demonštrujeme spôsob využitia viacerých LPM modulov a Altera Megafunkcie v jednom projekte.

Príklad 3
Vytvorte generátor číslicového sínusového signálu s parametrizovateľnou rýchlosťou výstupných vzoriek. Realizujte simuláciu v prostredí Modelsim.

Tento návrh využíva tri LPM funkcie (PLL, čítač a ROM) a Altera PLL (**P**hase **L**ocked **L**oop) Megafunkciu. Fázový záves vytvára z referenčného (externého) signálu o určitej frekvencii (napr. 100 MHz) hodinový signál požadovanej frekvencie (v našom prípade napr. 33.3 MHz). Týmto signálom bude riadený čítač, vytvorený ako LPM modul. Čítač generuje adresu pre ROM pamäť, v ktorej budú vzorky generovaného sínusového signálu uložené v *****.*mif* súbore. Tieto údaje[5] budú do ROM pamäte uložené v procese kompilácie.

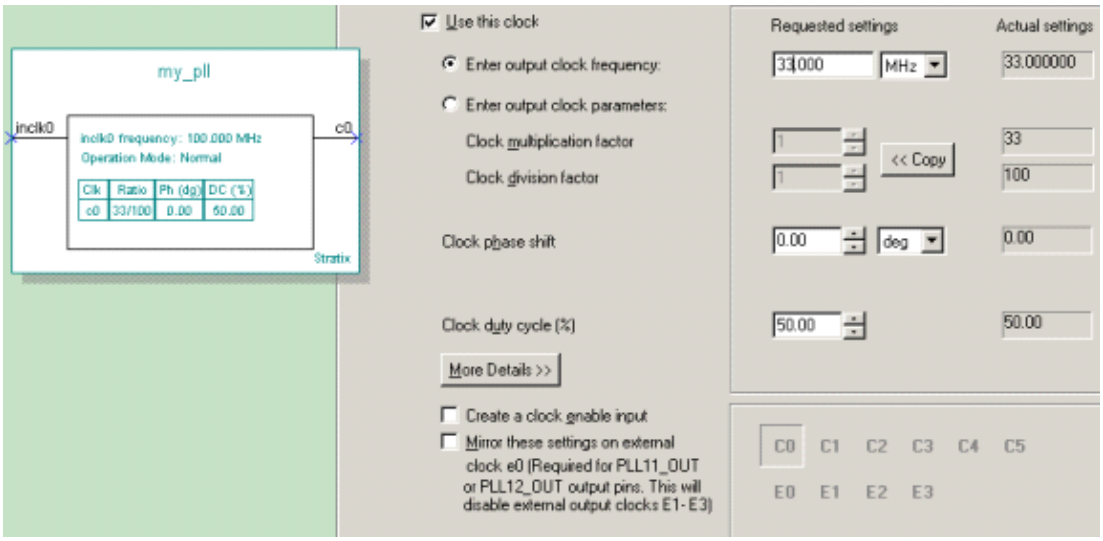
9.1.3.1 PARAMETRIZÁCIA ALTERA MEGAFUNKCIE PLL

Postup parametrizácie Megafunkcie PLL v prostredí Quartus II:

[5] Zmenou obsahu MIF súboru je možné meniť tvar generovaného signálu. Formát MIF súboru a zdrojový kód jednoduchého programu v jazyku C, ktorý generuje MIF súbor so vzorkami sínusového signálu je v **PríÍohe III**.

- V okne, ktoré sa zobrazí po spustení **MegaWizard Plug-In Manegeru** vyberieme z dostupných I/O megafunkcií megafunkciu **ALTPLL**[6]. Vytvoríme projekt napr. pre obvod z rodiny Stratix (tento obvod je použitý v tomto návrhu). Špecifikujeme názov vytváranej LPM funkcie (*my_pll*).
- V ďalšom okne zadáme frekvenciu vstupného hodinového signálu (100 MHz) a vyberieme typ PLL obvodu (*Fast PLL, Enhanced PLL*). Rozdiel medzi jednotlivými typmi PLL obvodov je vo veľkosti násobiteľa, resp. deliteľa frekvencie, a tým aj v tom, aké spektrum možných frekvencií môžeme v danom PLL implementovať. Zvoľme si *Enhanced PLL*, ktorý dokáže implementovať väčšie spektrum možných frekvencii.
- V okne 4/17 zaškrtneme nasledujúce CheckBoxy: **Create an 'pllena' input to selectively enable the PLL, Create an 'areset' input to asynchronously reset the PLL** a **Create 'locked' output**. Touto voľbou špecifikujeme, aby sa tieto signály vytvárali (samozrejme že ich vytváranie, resp. nevytváranie záleží na požiadavkách návrhára, resp. na type návrhu). Preklikáme sa až k oknu, v ktorom môžeme špecifikovať hodnotu frekvencie výstupného signálu. Existujú dve možnosti ako túto hodnotu zadať. Je možné zadať násobiteľ, resp. deliteľ pomocou ktorých sa výstupný signál požadovanej frekvencie dá vytvoriť, alebo priamo zadáme hodnotu výstupnej frekvencie (*Enter output clock frequency*).
- Jeden obvod PLL dokáže generovať viacero signálov rôznych frekvencií. V našom prípade v spodnej časti okna môžeme vidieť signály *C0 až C5* a *E0 až E3*. Signály označené ako *"C"* sú určené pre vnútorné použitie, teda vnútri súčiastky (C - **C**ore), zatiaľ čo signály označené ako *"E"* sú určené pre vonkajšie použitie, priradí sa im jeden z pinov súčiastky (E - **E**xternal). Pre návrh stačí len jeden signál, napr. *C0*, pre ktorý si nastavíme výstupnú frekvenciu 33 MHz.

[6] Obvody PLL sa nachádzajú len v novších FPGA obvodoch Altera. Preto je potrebné zvoliť niektorú súčiastku z rodín Cyclone, Stratix, Cyclone II alebo Stratix II.



9.1.3.2 POUŽITIE VYTVORENÝCH MODULU V PROJEKTE

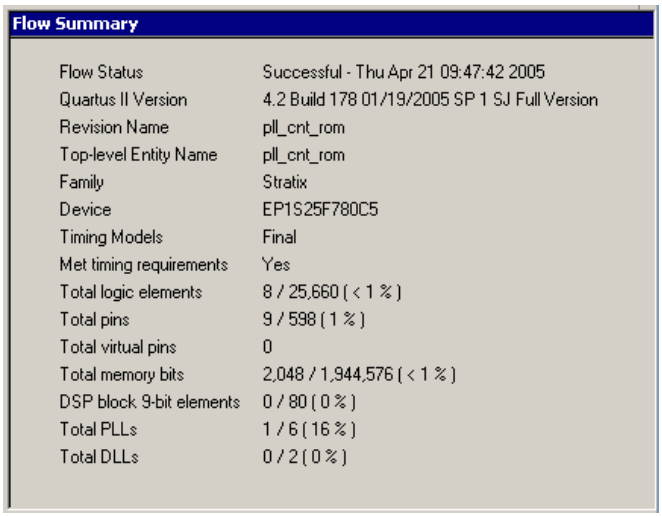
Nasledujúci VHDL kód[7] reprezentuje výpis *top-level* entity, ktorá realizuje pospájanie vytvorených LPM funkcií lpm_counter, lpm_rom a pll megafunkcie.

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

entity pll_cnt_rom is
        port
          (
          in_clock  : in        std_logic;
          vystup    : out       std_logic_vector (7 downto 0)
          );
end entity pll_cnt_rom;

architecture arch of pll_cnt_rom is

        -- deklaracia LPM funkcii
COMPONENT my_pll IS
                PORT
        (
                inclk0      : IN STD_LOGIC  := '0';
                c0          : OUT STD_LOGIC
        );
END COMPONENT;

COMPONENT my_counter IS
```

[7] Samozrejme bolo by možné využiť grafickú reprezentáciu projektu. Pri väčších projektoch je grafická reprezentácia na najvyššej úrovni často využívaná pre jej prehľadnosť. V prípade, že potrebujeme realizovať simuláciu projektu napr. v Modelsime, je nutné použiť VHDL kód.

```
                PORT
        (
                clock     : IN STD_LOGIC ;
                q         : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
        );
END COMPONENT;

COMPONENT my_rom IS
                PORT
        (
                address   : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                clock     : IN STD_LOGIC ;
                q         : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
        );
END COMPONENT;

        -- vnutorne signaly

signal clock        : std_logic;
signal address      : std_logic_vector (7 downto 0);

        begin

        -- vytvorenie instancii jednotlivych LPM funkcii
my_pll_inst: my_pll
                PORT MAP
        (
                inclk0    => in_clock,
                c0        => clock
        );

my_counter_inst: my_counter
                PORT MAP
        (
                clock     => clock,
                q         => address
        );

my_rom_inst: my_rom
                PORT MAP
        (
                address   => address,
                clock     => clock,
                q         => vystup
        );

end architecture;
```

Po skompilovaní celého návrhu je možné v okne **Flow Summary** zobraziť protokol o výsledku kompilácie (obr.7). Z výsledku kompilácie môžeme vyčítať, že celý návrh zaberá 8 LE (Logických Elementov), 2048 bitov pamäte a jeden fázový záves PLL.



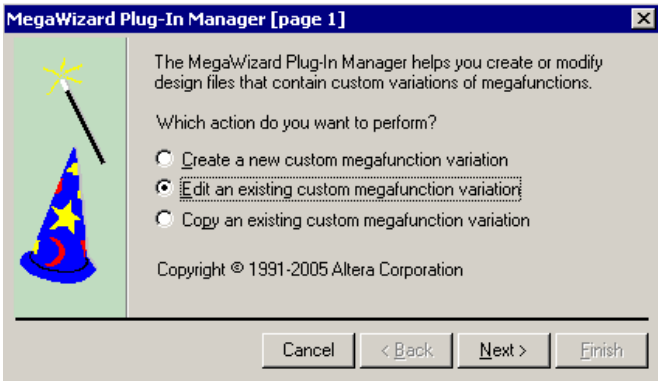
Obr.7 Protokol o preklade projektu v prostredí Quartus II

9.1.3.3 FUNKČNÁ SIMULÁCIA PROJEKTU V MODELSIME

Simuláciu návrhu v ModelSime je možné vykonať štandardným postupom. ModelSim však na rozdiel od Quartusu (ktorý umožňuje používať formáty MIF aj HEX) používa ako inicializačný súbor pre ROM pamäť súbor len vo formáte HEX[8] a nie MIF. Preto je potrebné súbor vo formáte HEX buď priamo vytvoriť, alebo ak už je vytvorený MIF súbor, je ho možné jednoducho prekonvertovať. Konverziu je možné vykonať aj v prostredí Quartus II otvorením MIF súboru (**File->Open**) a opätovným uložením vo formáte HEX (**File->Save as...**). Následne je potrebné LPM_ROM funkcii upraviť tak, ako inicializačný súbor použila HEX súbor, a nie MIF súbor. To je možné realizovať dvojakým spôsobom. Buď editáciou už vytvorenej LPM_ROM funkcie (*my_rom.vhd*) a to tak, že spustíme **MegaWizard Plug-In Manager** (**Tools->MegaWizard Plug-In Manager**) a vyberieme voľbu editácie už existujúcej megafunkciu zobrazenej na Obr.8.

Vyberieme megafunkciu, ktorú je potrebné editovať, v našom prípade je to *my_rom.vhd* a vo vhodnom okne špecifikujeme inicializačný súbor zmenou z *rom.mif* na súbor *rom.hex*.

[8] Formát Intel HEX je rozšírený formát využívaný predovšetkým v mikroporcesorovej technike. Na vyjadrenie binárneho obsahu súboru používa vybrané ASCII znaky.



Obr.8 Výsledok časovej simulácie v ModelSime

Druhým spôsobom (zvyčajne ľahším) je manuálna zmena v ľubovoľnom editore (je možné použiť aj editor v Quartuse alebo ModelSime). Po nájdení riadku, kde je špecifikovaný *rom.mif* súbor zmeníme súbor na *rom.hex*.

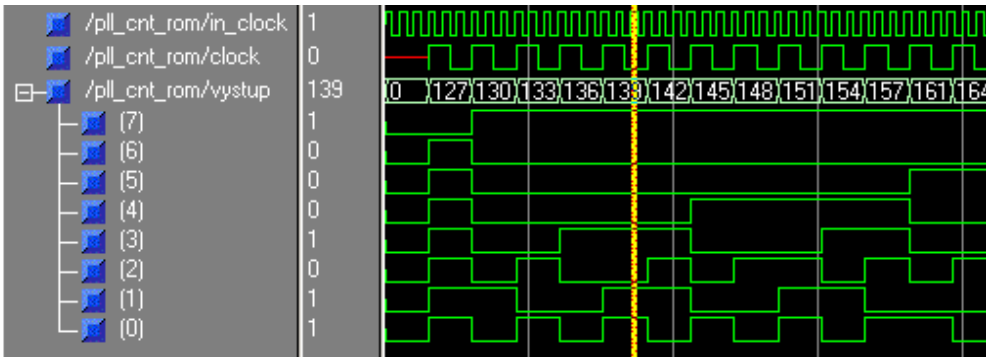
:
init_file => "rom.mif",
riadkom:
init_file => "rom.hex",

Po predefinovanú inicializačného súboru je možné funkčnú simuláciu vykonať pomocou nasledujúcich dávkových súborov.

```
# fsim.do
quit -sim
vcom -work work -2002 -explicit -novitalcheck -no1164 -novital my_pll.vhd
my_counter.vhd my_rom.vhd pll_cnt_rom.vhd
vsim -t ps work.pll_cnt_rom
add wave in_clock -radix unsigned vystup
do stimuls.do
```

```
# stimuls.do
restart -f
force -freeze sim:/pll_cnt_rom/in_clock 1 0, 0 {5 ns} -r 10
run 10000
```

Dôležité je všimnúť si, že pre simuláciu PLL megafunkcie je potrebné zvoliť rozlíšenie simulátora minimálne na pikosekundy. Výsledok funkčnej simulácie je zobrazený na Obr.9.



Obr.9 Výsledok funkčnej simulácie projektu v ModelSime

9.1.3.4 ČASOVÁ SIMULÁCIA PROJEKTU V MODELSIME

Pri skompilovaní projektu v prostredí Quartus II vytvoríme SDO a VHO súbory pre vykonanie časovej simulácie v ModelSime. Generovanie SDO a VHO súborov zabezpečíme nastavíme pri vytváraní projektu resp. je možné dodatočne nastastaviť cez **Assignments->Settings** výberom položky *EDA* **Tool Settings -> Simulation** a určíme **Tool name** ako **ModelSim***(VHDL)*.

Časovú simulácie v ModelSime je možné realizovať pomocou nasledujúcich dávkových súborov:

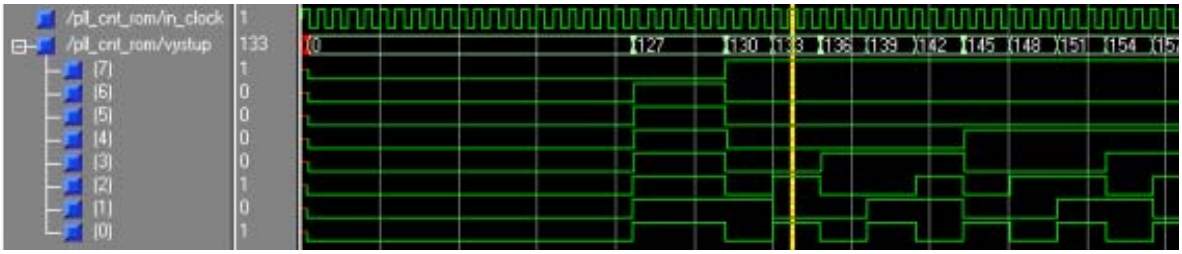
```
# tsim.do
quit -sim
vcom -work work -2002 -explicit -novitalcheck -no1164 -novital pll_cnt_rom.vho
vsim -t ps -sdftyp pll_cnt_rom_vhd.sdo work.pll_cnt_rom
add wave in_clock -radix unsigned vystup
do stimuls.do
```

V dávkovom súbore je využitý prídavný parameter príkazu **vsim**, tj. parameter **–t ps**, ktorý zaručí väčšie rozlíšenie simulácie, ktoré je nevyhnutné pre simuláciu PLL

obvodu. Tiež príkaz **add** využíva naviac parameter **–radix unsigned**, ktorý zabezpečí, že výsledok simulácie, teda vektor *vystup* nebude zobrazený v binárnej podobe, ale ako neznamienkové celé číslo.

 # stimuls.do
restart -f
force -freeze sim:/pll_cnt_rom/in_clock 1 0, 0 {5 ns} -r 10
run 10000

Výsledok časovej simulácie je zobrazené na Obr.10.



Obr.10 Výsledok časovej simulácie projektu v ModelSime

PRÍLOHA I

Prehľad vybraných LPM a Altera Megafunkcií, ktoré sú dostupné pri návrhu v rámci väčšiny obvodov Altera. Ich podrobnejší opis je možné nájsť v helpe Quartusu resp. v príslušných manuáloch (www.altera.com).

Aritmetické komponenty
altaccumulate (Accumulator) Megafunction – megafunkcia poskytujúca parametrizovateľný akumulátor. Dizajnér môže pre túto megafunkciu generovať *clear-box* model, ktorý sa dá následne syntetizovať v prostredí *third-pary* EDA. *Clear-box* je

metóda, ktorá generuje plne syntetizovateľnú megafunkciu, resp. LPM funkciu pre použitie s EDA syntetizačným nástrojom. S vlastnosťou *clear-box* môže EDA plne syntetizovať a optimalizovať dizajny, v ktorých sú použité Altera megafunkcie, resp. LPM funkcie. Aby sme mohli túto vlastnosť využiť, musíme zapnúť *Generate ClearBox body (for EDA tools only)* v okne **MegaWizard Plug-In Manager** z **Tools menu**.

altfp_mult (Floating Point Multiplier) Megafunction – násobička pracujúca s operandami vyjadrenými v pohyblivej rádovej čiarke. Táto megafunkica podporuje následné formáty pohyblivej rádovej čiarky: jednoduchá presnosť, dvojitá presnosť a jednoduchá rozšírená presnosť.

altmemmult (Memory Multiplier) Megafunction – pamäťová parametrizovateľná násobička. Táto megafunkica umožňuje RAM-based znamienkové, alebo neznamienkové násobenie.
- stále sa generuje *clear-box* model.

altmult_accum (Multiply-Accumulate) Megafunciton – parametrizovateľná megafunkcia násobenie-akumulovanie. Táto megafunkcia pozostáva z jednej násobičky, ktorej výstup napĺňa akumulátor.

altmult_add (Multiply-Add) Megafunction – parametrizovateľná megafunkcia násobenie-sčítanie. Táto megafunkcia pozostáva z jednej alebo viacerých násobičiek, ktorých výstupy sa sčítavajú v paralelnej sčítačke.
- môžeme generovať *clear-box* model

altsqrt (Integer Square Root) Megafunction – parametrizovateľná megafunkcia na výpočet druhej odmocniny z celých čísel

altsquare (Squarer) Megafunction – parametrizovateľná megafunkcia vykonávajúca druhú mocninu vstupných dát

lpm_abs (Absolute Value) Megafunction – výpočet absolútnej hodnoty vstupných dát

lpm_add_sub (Adder/Subtract) Megafunction – megafunkcia umožňujúca súčet/rozdiel vstupných operandov

lpm_compare (Comparator) Megafunction – porovnanie vstupných operandov

lpm_counter (Counter) Megafunction – parametrizovateľný čítač. Táto megafunkcia je binárnym čítačom, ktora dovoľuje realizovať čítač vpred, vzad alebo vpred/vzad, s možnosťou využitia synchrónnych, resp. asynchrónnych clear, set a load portov.

lpm_divide (Divider) Megafunction – parametrizovateľná delička počítajúca celú časť a zvyšok po delení, za podmienky že na vstupe sú násobiteľ a deliteľ.

alt_mult (Multiplier) Megafunction – parametrizovateľná násobička. Ak násobime M-bitové číslo s N-bitovým číslom (M*N), táto megafunkcia generuje výsledok široký (M+N) bitov. V prípade ak M=1 (resp. N=1), potom len N (resp. M) nižších bitov je podstatných.

parallel_add (Parallel Adder) Megafunction – megafunkcia pozostávajúca zo sčítačky a určitého počtu vstupov (vstupných údajov - sčítancov), ktorá na výstupe dáva ich súčet

Hradlá

lpm_mux, mux & busmux (Multiplexers) Megafunction – parametrizovateľný multiplexer.

Mux a *busmux* megafunkcie sú odvodené od *lpm_mux* a sú zavedené z dôvodu uľahčenia použitia *lpm_mux* v grafickom režime návrhu dizajnu (.bdf).

- *busmux* je ekvivalentné *lpm_mux* s *LPM_SIZE=2*
- *mux* je ekvivalentné *lpm_mux* s *LPM_WIDTH=1*

- dá sa generovať *clear-box*

lpm_and (AND Gate) Megafunction – Altera odporúča používať *AND* primitívnu funkciu pred použitím *lpm_and* megafunkcie. Avšak, v prípade ak potrebujeme parametrizovateľné vstupy, môžeme využiť túto megafunkciu.

lpm_gustri (Buffer) Megafunction – parametrizovateľný troj-stavový buffer. Je vhodné ju použiť v prípade ak potrebujeme riadiť jednosmerné a obojsmerné V/V zbernice.

lpm_clshift (Shifter) Megafunction – parametrizovateľný posúvač (shifter)

lpm_constant (Constant Generator) Megafunction – parametrizovateľný generátor konštánt. Vhodné v prípade, ak potrebujeme konvertovať parameter na konštantu.

lpm_decode (Decoder) Megafunction – parametrizovateľný dekóder. Altera odporúča použiť túto megafunkciu v prípade, ak šírka vstupných dát je menšia, resp. rovná 8. Dá

sa taktiež s výhodou použiť *lpm_compare* s jedným vstupom nastaveným na fixnú hodnotu.
- dá sa generovať *clear-box*

lpm_inv (NOT Gate) Megafunction – parametrizovateľný invertor. Altera odporúča používať *NOT* primitívnu funkciu pred použitím *lpm_inv*. *Lpm_inv* funkcia je užitočná len v prípade ak compilačné súbory sú importované z iných než priemyselne-štandardných návrhových prostriedkoch.

lpm_or (OR Gate) Megafunction – parametrizovateľné hrádlo OR. Táto megafunkcia má uplatnenie v prípade ak potrebujeme parametrizovateľné vstupy.

lpm_xor (XOR Gate) Megafunction – parametrizovateľné hrádlo XOR. Táto megafunkcia má uplatnenie v prípade ak potrebujeme parametrizovateľné vstupy.

IO komponenty
altcdr_rx (CDR) Megafunction – Clock Data Recovery (CDR) prijímač. *Altcdr_rx* megafunkcia implementuje *CDR* prijímač. Táto megafunkcia využíva CDR prijímač pre deserializáciu, a obsahuje digitálnu fázovú slučku (PLL – Phase-Locked Loop) pre kompenzáciu channel-to-lock a channel-to-channel asymetriu a taktiež FIFO na zachovanie časovej základne signálu. Dostupná len pre *Mercury* FPGA obvody.

altcdr_tx (CDR) Megafunction - Clock Data Recovery (CDR) vysielač. *Altcdr_tx* megafunkcia implementuje *CDR* vysielač. Využíva CDR vysielač pre serializáciu, taktiež ako aj FIFO na zachovanie časových základní hodinových signálov. Dostupná len pre *Mercury* FPGA obvody.

altclkctrl(Clock Control Block) Megafunction – megafunkcia výberu hodinového signálu – dokáže vybrať spomedzi globálnych hodín, regionálnych hodín, duálny regionálnych hodín alebo externých ciest hodinového signálu. Je to tzv. clock-selector.

altclklock (Phase-Locked Loop) Megafunction – parametrizovateľný fázový záves (PLL). Táto megafunkcia sprístupňuje použitie PLL obvodu umiestneného v konkrétnej cieľovej súčiastke. PLL sa používa k syntéze hodinového signálu odvodeného od referenčného signálu (obyčajne kryštál umiestnený na doske). *Altclklock* megafunkcia redukuje časové oneskorenie a rozptyl, a môže byť využitá ku generovaniu vnútorného hodinového signálu pracujúceho na frekvenciách ktoré sú násobkami frekvencie systémových hodín.

altddio_bidir (DDIO Bidirectional) Megafunction – Double Data Rate (DDR) – zdvojnásobená prenosová rýchlosť. *Altddio_bidir* vysiela a prijíma údaje na obidve hrany hodinového signálu (nábežná, spádová)

altddio_in (DDIO input) Megafunction – prijíma údaje na obidve hrany referenčného hodinového signálu

altddio_out (DDIO output) Megafunction - vysiela údaje na obidve hrany referenčného hodinového signálu

altdq (data strobe) Megafunction - prijíma a vysiela údaje na obidve hrany referenčného hodinového signálu

altdqs (Bidirectional Data Strobe) Megafunction – generuje skupinu DQS signálov potrebných pre snímanie read/write údajov na externom DDR/FCRAM pamäťovom rozhraní. *Altdqs* megafunkcia implementuje jeden DLL a určitý počet užívateľom špecifikovaných DQS signálov . Maximálny počet DQS signálov podporovaných konkrétnym DLL závisí na konkrétnom obvode.
- táto megafunkcia stále generuje *clear-box*

altgxb (Gigabit Transceiver Block) Megafunction – umožňuje použitie špeciálneho obvodu pre Stratix-GX obvody.

altlvds_rx (LVDS) Megafunction – deserializačný prijímač. LVDS je jedným z vysoko-rýchlostných I/O rozhraní využívajúci diferenčné signály bez potreby referenčného napätia. K prenosu využíva dva vodiče, z ktorých napokon vytvorí jeden kanál. K deserializácii sa využíva PLL obvod.

altlvds_rx (LVDS) Megafunction – serializačný vysielač. Na serializáciu je použitý PLL obvod.

altpll (Phase-Locked Loop) Megafunction – megafunkcia parametrizovateľného PLL. Sprístupňuje použitie PLL obvodu umiestneného na čipe súčiastky. PLL sa používa k syntéze hodinového signálu odvodeného od referenčného signálu.

altpll_reconfig (Phase-Locked Loop Reconfiguration) Megafunction – umožňuje kontrolovať PLL počas behu aplikácie (real-time control). *MegaWizard Plug-In Manager* generuje Verilog HDL, resp. VHDL použiteľné v EDA simulátore, avšak negeneruje samostatný funkčný simulačný model tejto megafunkcie.
- vždy sa generuje *clear-box*

altremote_update (Remote Update) Megafunction – umožňuje rekonfiguráciu súčiastky za behu aplikácie (real-time reconfiguration).
-vždy sa generuje *clear-box*

altufm_osc (I/O MAX II Oscilator) Megafunction – sprístupňuje použitie vnútorného UFM (User Flash Memory) oscilátora v súčiastkach MAX II. Oscilátor sa dá použiť ako zdroj všeobecných hodín, bez použitia FLASH pamäte. V prípade ak použijeme v našom návrhu túto megafunkciu, potom nie je možné použiť následovné megafunkcie: *altufm_none, altufm_parallel, altufm_spi* a *altufm_i2c.*

Kompilátor pamäte
Altcsmem (FIFO partitioner) Megafunction – FIFO partitioner je nástroj na mapovanie viacerých FIFO do jedného. Možnosť rozdeliť M-RAM bunky do viacerých FIFO výrazne vylepšuje využitie pamäťovej architektúry v Stratix obvodoch. FIFO partitioner automaticky generuje logiku potrebnú pre časový multiplex Stratix M-RAM buniek medzi viacerými užívateľom špecifikovanými FIFO.

SignalTap II Logic analyzátor
sld_signaltap (SignalTap II Logic Analyzer) Megafunction – megafunkcia umožňuje vytvorenie inštancie *SignalTap II Logic Analyzer* v skorších cykloch návrhu a bez potreby *SignalTap II File (.stp)*. Návrhy obsahujúce túto megafunkciu sa dajú kompilovať bez STP súboru. V prípade potreby sa tento STP súbor dá vytvoriť z tejto megafunkcie prostredníctvom *File Menu* (*Create/Update->Create SignalTap II File from Design Instance(s)*).
Altera neodporúča vytvárať inštancie priamo v návrhovom súbore, ale radšej použitie *MegaWizard Plug-In Manager* na vytvorenie inštancie *sld_signaltap* megafunkcie.

zálohovacie komponenty (pamäte, FIFO, posuvné registre ...)
alt3pram (Tripple-Port Ram) Megafunction – parametrizovateľný troj-portový RAM pamäťový modul. Táto megafunkcia je poskytovaná len z dôvodu spätnej kompatibility v Cyclone, Cyclone II, HardCopy Stratix, Stratix a Stratix GX návrhoch. Namiesto tejto megafunkcie Altera odporúča použitie *altsyncram* megafunkcie.

altcam (Content-Addressable Memory) Megafunction – Content Addressable Memory (CAM) – Údaje uložené v CAM sú množinou vzoriek (charakteristík) ktoré môžu byť prehľadávane v jednom hodinovom cykle. Ak vstupná vzorka zadaná do CAM súhlasí s jednou vzorkou uloženou v CAM , generuje sa jej adresa. *Altcam* megafunkcia dovoľuje každý bit uloženej vzorky špecifikovať ako log1, log0, alebo „don't care".

Porovnávanie bitu uloženej vzorky špecifikovaného ako „don't care" s jeho korešpondujúcim bitom vstupnej vzorky stále vedie ku zhode.

altqpram (Quad-Port Ram) Megafunction - parametrizovateľný štvor-portový RAM pamäťový modul.

altshift_taps (Shift Register with Taps) Megafunction – parametrizovateľný SHIFT register s TAP megafunkciou. Implementuje RAM-based shift register pre efektívne vytvorenie veľkých posúvnych registrov.
- dá sa generovať *clear-box*

altsyncram Megafunction – parametrizovateľný RAM modul s duálnym portom.
- môžeme generovať *clear-box*

altufm_i2c (User Flash Memory with Intr-Integrated Circuit Protocol) Megafunction – implementuje I^2C protokol ako rozhranie medzi MAX II UFM blokom a externými zariadeniami. Dostupná len pre MAX II obvody.

altufm_none (User Flash Memory) Megafunction – implementuje užívateľskú pamäť nezávislú na napätí pre MAX II obvody bez použitia protokolu rozhrania. Dostupná len pre MAX II obvody.

altufm_parallel (User Flash Memory wtih Parallel Protocol) Megafunction – implementuje užívateľskú pamäť nezávislú na napätí pre MAX II obvody s využitím paralelného protokolu rozhrania. Dostupná len pre MAX II obvody.

altufm_spi (User Flash Memory with SPI Protocol) Megafunction - implementuje užívateľskú pamäť nezávislú na napätí pre MAX II obvody s využitím SPI protokolu rozhrania. Dostupná len pre MAX II obvody.

lpm_ff (D- or T-type FlipFlop) Megafunction – parametrizovateľné klopné obvody. *Lpm_ff* funkcia obsahuje vlastnosti, ktoré nie sú dostupné v DFF, DFFE, DFFEA, DFFEAS, TFF a TFFE primitívach, ako je napríklad asynchrónny vstup pre nastavenie, mazanie načítanie.

lpm_fifo (Single-Clock FIFO) Megafunction – parametrizovateľné single-clock FIFO. *Lpm_fifo* využíva M-RAM, M4K, M512, ESB(Embedded System Block) alebo EAB (Embedded Array Block) pamäťové bloky, v závislosti od cieľovej súčiastky. V prípade ak potrebujeme špeciálne vlastnosti, dá sa s výhodou použiť *scfifo* megafunkcia. Altera odporúča používať synchrónne, a nie nesynchrónne RAM funkcie.

lpm_fifo_dc (Dual-Clock FIFO) Megafunction - parametrizovateľné dual-clock FIFO.

lpm_latch (Latch) Megafunction – parametrizovateľný latch (zásobník). Altera odporúča používať túto megafunkciu len v prípade ak je potrebné využiť *aset* port, ináč je výhodnejšie použiť LATCH primitívnu funkciu.

lpm_ram_dp (Dual-Port RAM) Megafunction – parametrizovateľná dual-port pamäť (dva vstupy a dva výstupy, ktoré môžu byť taktované spoločným hodinovým signálom, resp. samostatným hodinovým signálom (jeden hodinový signál pre zápis, druhý pre čítanie)). Možnosť použitia tejto megafunkcie k realizácii kruhového registra.

lpm_ram_dq (RAM) Megafunction – parametrizovateľná RAM s oddelenými vstupnými a výstupnými portami. Altera odporúča použiť k realizácii asynchrónnej pamäte, alebo pamäte so synchrónnymi vstupmi/výstupmi túto megafunkciu.

lpm_ram_io (RAM) Megafunction – parametrizovateľná RAM s jedným I/O portom. Táto megafunkcia je uvedená len z dôvodu spätnej kompatibility v Cyclone, Cyclone II, HardCopy Stratix, Stratix a Stratix GX návrhoch. – Altera odporúča použitie *altsyncram* megafunkcie.

lpm_rom (ROM) Megafunction – parametrizovateľná ROM. Táto megafunkcia je uvedená len z dôvodu spätnej kompatibility v Cyclone, Cyclone II, HardCopy Stratix, Stratix a Stratix GX návrhoch. – Altera odporúča použitie *altsyncram* megafunkcie.

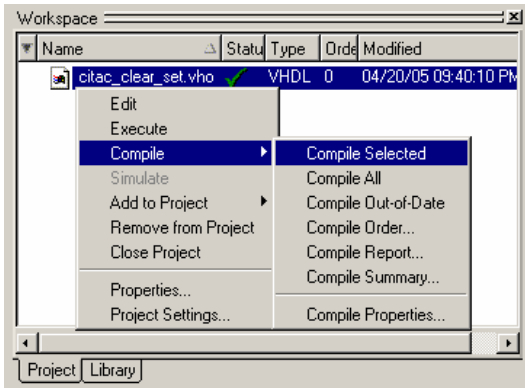
lpm_shiftreg (Universal Shift Register) Megafunction – parametrizovateľný posuvný register

PRÍLOHA II

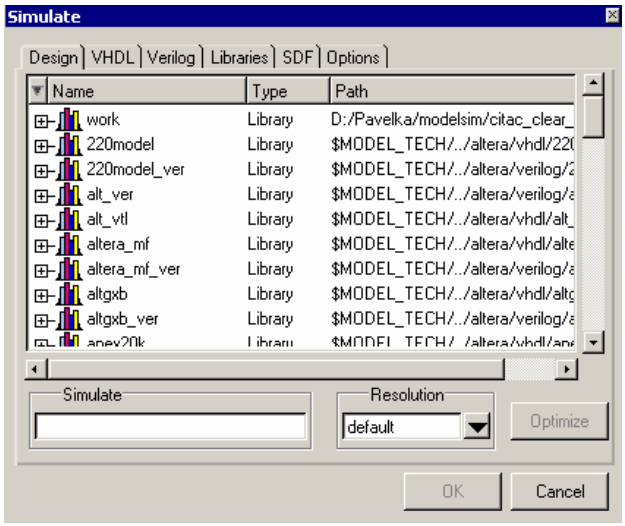
Postup kompilácie VHDL súborov a simulácie v Modelsime bez použitia dávkových súborov

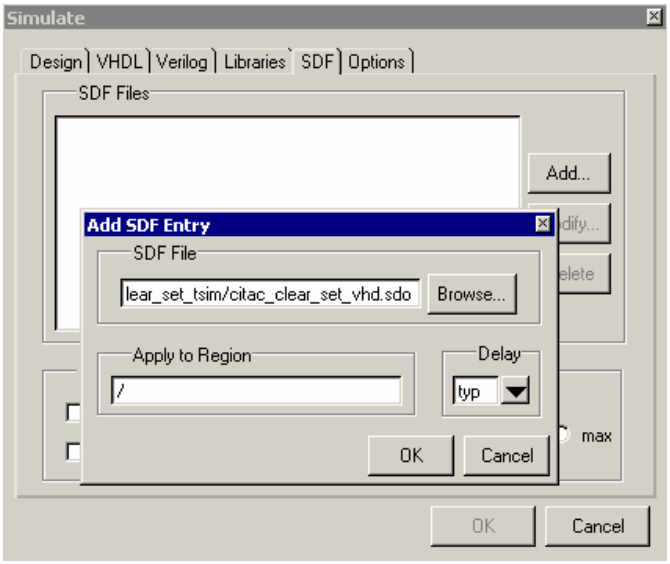
Tento postup je uvedený pre úplnosť. Jednotlivé etapy je možné pomocou GUI realizovať podľa nasledujúceho postupu:

- kompilácia
Vo *Workspace* okne otvoríme záložku **Project** a pravým tlačidlom myši klikneme na súbor, ktorý chceme skompilovať. Z **Compile** položky vyberieme **Compile Selected**. V prípade ak by kompilujeme všetky súbory nachádzajúce sa v aktuálnom **Worksace**, zvolíme **Compile All**.

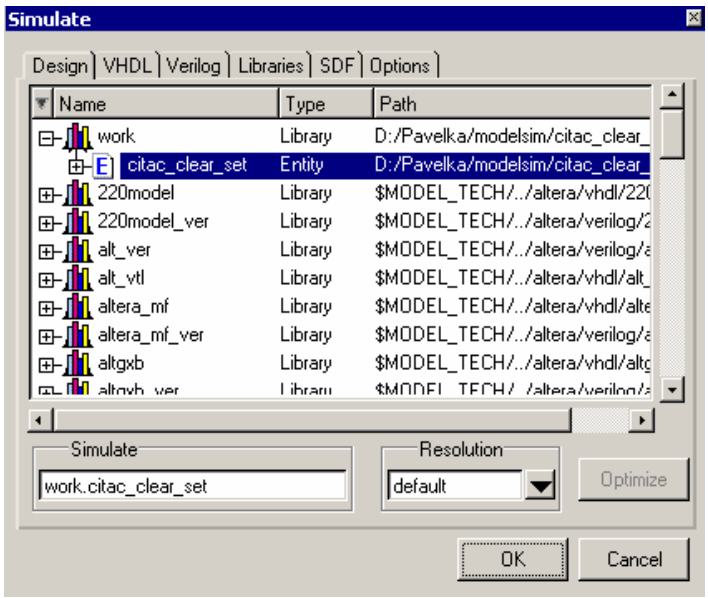


- simulácia
Vyberieme položku **Simulate->Simulate**, čím sa nám otvorí **Simulate** okno



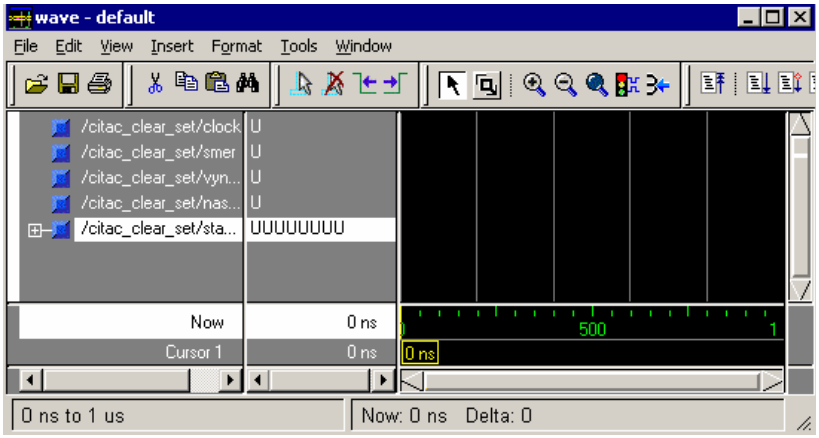


V zložke *SDF* definujeme umiestnenie **.sdf* súboru, ktorý obsahuje informácie o časových oneskoreniach obvodu pre ktorý danú simuláciu vykonávame Prejdeme späť do zložky **Design** a vyberieme entitu, pre ktorú chceme vykonať simuláciu a potvrdíme

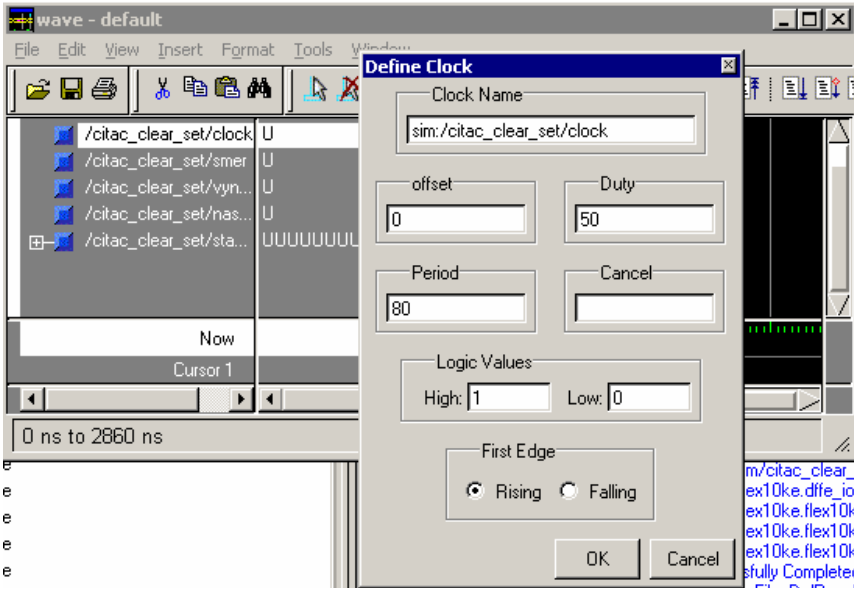


OK (viď. obr.)

Ďalej potrebujeme vytvoriť vstupné stimuly. Otvoríme okná **Signals** a **Wave** (**View->Signals, View->Wave**). Jednotlivé signály do okna **Wave** presunieme z okna **Signals** myškou (**Drag&Drop**)



Nastavíme vstupné stimuly (pravým tlačidlom myši klikneme na signál, ktorý chceme editovať a nastavíme potrebnú hodnotu) – na obrázku je zobrazené nastavenie



hodinového signálu o perióde 80 ns.

Ak sme postupovali správne, dostaneme rovnaký výsledok, ako keby sme používali dávkové súbory.

PRÍLOHA III

Naplnenie ROM pamäte údajmi, ktoré zodpovedajú jednej perióde signálu, je definované MIF súborom. Nasledujúci program v jazyku C generuje MIF súbor obsahujúci vzorky sínusového signálu:

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

void zaciatok (FILE *subor)
{
        fprintf(subor, "WIDTH=8;\n");
        fprintf(subor, "DEPTH=256;\n\n");
        fprintf(subor, "ADDRESS_RADIX=DEC;\n");
        fprintf(subor, "DATA_RADIX=DEC;\n\n");
        fprintf(subor, "CONTENT BEGIN\n");
}

void koniec (FILE *subor)
{
        fprintf(subor, "END;");
}

int main ()
{
        FILE *subor;
        double PI = 3.1415927;
        double x;
        double vysledok;
        int i;

        if ((subor = fopen("sin.mif", "wt")) == NULL)
        {
        fprintf(stderr, "Nemozem vytvorit subor.\n");
        return 1;
        }

        zaciatok(subor);

        for (i=0; i<256; i++)
        {
                x = ((double)i/255)*2*PI;
                vysledok = (255/2) + sin(x)*(255/2);
                fprintf(subor, "%d\t:\t%d;\n", i, (int)vysledok);
   }

        koniec(subor);

        fclose(subor);
        return 0;
}
```

Tento súbor je možné využiť ako súbor obsahujúci inicializačné údaje pri vytváraní ROM pamäte.

Formát MIF (Memory Inicialization File) súboru:

Kľúčovým slovom **WIDTH** je špecifikovaná bitová šírka uložených údajov (8 bitov). **DEPTH** určuje počet slov, ktoré sa do pamäte uložia, teda určuje veľkosť pamäte (256 slov, čomu zodpovedá 8 bitová adresa). **ADDRESS_RADIX** určuje formát čísel, ktorými sú vyjadrené adresy (je to špecifikácia formátu v MIF súbore,

v hardvérovej realizácii bude adresa vyjadrená v binárnej podobe). Je možné zvoliť jednu z týchto možností:

BIN - binárna reprezentácia
DEC - desiatková sústava
HEX - hexidecimálna reprezentácia
OCT - osmičková sústava
UNS - unsigned (neznamienkové číslo)

DATA_RADIX určuje formát čísel, ktorými sú vyjadrené údaje. Jednotlivé možnosti, ktoré platia pre **ADDRES_RADIX** platia aj v tomto prípade.

Samotné údaje sú uzatvorené medzi kľúčovými slovami **CONTENT BEGIN** a **END**. Údaje je možné špecifikovať po jednom, alebo v určitom rozsahu. Napríklad je potrebné aby prvých 10 miest v pamäti obsahovalo hodnotu 13, zadáme ich ako:

[0..10] : 13;

Adresa je od údaja oddelená dvojbodkou.

Pre lepšiu predstavu je uvedený príklad takéhoto súboru:

```
WIDTH=8;
DEPTH=256;

ADDRESS_RADIX=DEC;
DATA_RADIX=DEC;

CONTENT BEGIN
        [0..100]        :   13;
        [101..200]      :   21;
        [201..255]      :   17;
END;
```