7. SIMULÁCIA VHDL V MODELSIME

Opis hardvéru v jazyku VHDL prípadne Verilog umožňuje veľkú časť návrhu a overenia cieľového systému nezávisle od cieľovej ASIC alebo FPGA platformy. V rámci predchádzajúcich cvičení bol využívaný návrhový systém Quartus II od firmy Altera, v ktorom bolo možné realizovať všetky etapy návrhu. Existujú však aj simulačné nástroje, ktoré sú nezávislé od výrobcov obvodov (avšak poskytujú podporu pre všetkých významných výrobcov ASIC a FPGA obvodov). Svetovú špičku v tejto oblasti predstavuje program **ModelSim** firmy Mentor Graphics. Podobne samotní výrobcovia (samozrejme vrátane firiem Altera a Xilinx) poskytujú resp. priamo využívajú možnosti programu ModelSim. Hlavnou motiváciou výrobcov pre využívanie ModelSimu (za čo musia firme Mentor Graphics platiť!) je možnosť realizácie aj veľmi zložitých simulácii a testov, čo firemné simulačné nástroje často neumožňujú. Pre užívateľov je tiež často výhodné používať nástroj, ktorý umožňuje overiť veľkú časť návrhu bez nutnosti voľby cieľovej technológie. Naviac, je možné cieľovú technológiu zmeniť a stále využívať rovnaké simulačné prostredie. Užívateľ však za tieto možnosti musí zaplatiť pomerne veľké sumy. Keďže na FEI TU v Košiciach je v rámci **Univerzitného programu firmy Mentor Graphics** (www.kemt.fei.tuke.sk/fpga) k dispozícii aj simulačný program ModelSim, budeme v rámci cvičení využívať **plne funkčnú** verziu ModelSimu. Na domácich počítačoch je možné inštalovať **obmedzenú verziu** programu ModelSim firmy Xilinx, ktorý je možné využívať po bezplatnej registrácii (podrobnosti sú v **prílohe II**).

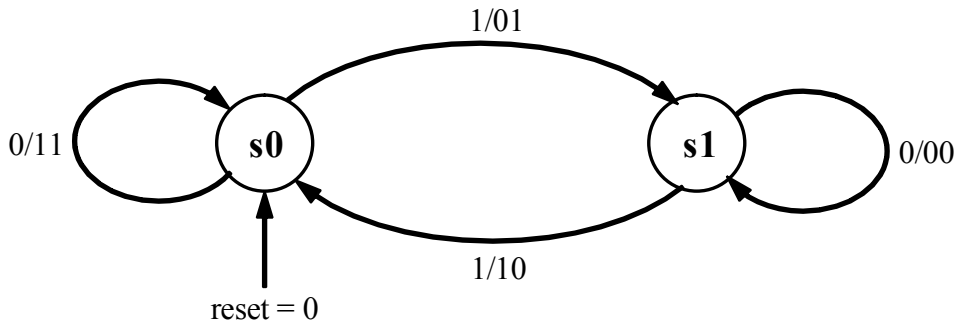
V rámci cvičenia bude demonštrovaná resp. precvičovaná nasledujúca problematika:

- vytvorenie Mealyho automatu v jazyku VHDL,
- základné príkazy ModelSimu (**Príloha I**),
- kompilácia a **funkčná** simulácia v ModelSime,
- dávkové spracovanie v ModelSime,
- využitie komponentov vo VHDL.

7.1 MEALYHO AUTOMAT (FINITE STATE MACHINE)

Príklad 1

Navrhnite konečný automat Mealyho typu, ktorý bude mať dva stavy (s0 a s1). Prechody medzi stavmi sú znázornené na obr.1. Do počiatočného stavu (s0) sa automat dostane privedením log0 na resetovací vstup, resp. prechodom zo stavu s1. Návrh realizujte vo VHDL.



Obr.1: Mapa prechodov konečného automatu Mealyho typu z Príkladu 1

7.1.1 ROZBOR MEALYHO AUTOMATU

Konečné automaty sú v praktických hardverových realizáciách veľmi často využívané. Typickým príkladom je opis TAP radiča v štandarde JTAG, ktorý bol analyzovaný počas prednášky. Ďalšie príklady realizácii 4-stavových konečných automatov Mealyho a Moorovho typu vo VHDL je možné nájsť napr. na adrese http://www.icaen.uiowa.edu/~vlsi1/web_131_dir/vhdl_help.html, kde je možné získať aj testbenche k jednotlivým realizáciám (pojem testbench a jeho realizácia bude vysvetlená na budúcom cvičení).

Mealyho konečný automat je automat, ktorého výstupy závisia na aktuálnom vstupe(och), ako aj na stave, v ktorom sa automat aktuálne nachádza. Vstupný signál je v príklade realizovaný jedným bitom, signálom *input*, zobrazeným na obr.1 ako 1/ resp. 0/. V závislosti od tohto vstupu a aktuálneho stavu automatu bude dochádzať k jednotlivým prechodom, ako to naznačuje mapa prechodov na obr.1. Výstup je realizovaný pomocou dvoch bitov (2-bitového vektora), signálu *output*, zobrazených ako /00, /01, /10, /11. Pre demonštračné účely nám takáto špecifikácia konečného automatu v Príklade 1 úplne postačuje (ak by mal automat viac stavov, širší vstupný vektor (viac bitov) a nakoniec aj viac prechodových stavov, návrh by bol zložitejší a v konečnom dôsledku aj neprehľadnejší).

7.1.2 REALIZÁCIA MEALYHO AUTOMATU VO VHDL

VHDL kód podobne ako VHDL kódy preberané v predchádzajúcich cvičeniach obsahuje na začiatku definície použitých knižníc:

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

Použité knižnice patria medzi základné a najčastejšie používané VHDL knižnice.

ieee.std_logic_1164 – definuje dátové typy pre prepojenia medzi komponentmi
ieee.std_logic_arith – obsahuje základné aritmetické operácie, konverzie jednotlivých podporovaných údajových typov medzi sebou
ieee.std_logic_unsigned – obsahuje bezznamienkovú aritmetiku, konverzie a funkcie porovnania pre dátový typ *std_logic_vector*

Nasleduje deklarácia entity, ktorá definuje vstupy a výstupy entity, t.j. ich počet, prípadne rozmery jednotlivých vektorov. Pri deklarácii entity sa abstrahujeme od jej realizácie, t.j. napr. opisu konkrétneho zapojenia logických členov, ktoré tvoria entitu. Deklarácia entity sa začína kľúčovým slovom *entity* a končí kľúčovým slovom *end*, ako je vidno z výpisu časti zdrojového kódu nižšie. Kľúčové slovo *in* resp. *out* určuje, či sa jedná o vstupný, resp. výstupný signál.

```
entity mealy_2states is
port
        (
        clock   : in    std_logic;                      -- vstupne hodiny (hodinovy signal)
        input   : in    std_logic;                      -- vstupny signal (urcuje prechod do ineho stavu)
        reset   : in    std_logic;
        output  : out   std_logic_vector (1 downto 0) -- vystup (informacny vystup)
        );
end entity mealy_2states;
```

Vnútorné zapojenie obvodu je opísané v bloku *architecture*, kde je opísané **správanie**[1] sa navrhovaného obvodu. K vytvoreniu nového údajového typu, ktorý bude obsahovať všetky možné stavy, do ktorých sa môže automat dostať (v našom prípade sú to stavy s0 a s1) je použité kľúčové slovo *type*. Použitie je zrejmé z výpisu kódu. Následne je definovaný *signál* vytvoreného údajového typu, ktorý bude obsahovať aktuálny stav automatu.

```
architecture rtl of mealy_2states is
        type mealy_states is (s0, s1);          -- dva stavy, do ktorych sa FSM moze dostat
        signal state    : mealy_states;
begin
```

Nasleduje process, ktorý je aktívny na hodinový signál *clock* a vstupný signál *reset*. V prípade ak je *reset* signál v stave log0, automat sa dostane do počiatočného stavu, čo je v našom prípade stav s0 (`if reset='0' then state <= s0;`). V opačnom prípade, ak je signál *reset* v stave log1, testuje sa hodinový signál na nábežnú hranu (`if (clock'event and clock='1') then`) a v prípade ak hodinový signál prechádza zo stavu log0 do stavu log1 (nábežná hrana), testuje sa najskôr, v ktorom z možných stavov sa momentálne automat nachádza a v závislosti od vstupného signálu buď dôjde k prechodu do ďalšieho stavu, alebo automat ostane v pôvodnom stave. Pre lepšiu názornosť uvažujme, že automat je momentálne v stave *s0*, vstupný signál *input* je v stave log1, v tomto prípade pri nábežnej hrane hodinového signálu dôjde k prechodu do stavu *s1*, a výstup sa nastaví na

[1] V prípade použitého opisu automatu sa skutočne jedná o opis správania, kde je úlohou kompilátora vytvoriť finálnu realizáciu. Pre úplnosť je však potrebné dodať, že jazyk VHDL umožňuje aj prípadný opis konkrétnej realizácie (napr. pomocou klopných obvodov, logických členov a pod.), čo by však z pohľadu užívateľa bolo v tomto prípade menej výhodné.

hodnotu "01" (when s0 => if input='1' then state <= s1; output <= "01";). Týmto spôsobom sa dá určiť správanie sa automatu za ľubovoľných podmienok – viď VHDL realizácia.

```
process (clock, reset)
begin
        if reset='0' then
                state <= s0;
        else
                if (clock'event and clock='1') then
                        case (state) is
                                when s0 =>
                                        if input='1' then
                                                state <= s1;
                                                output <= "01";
                                        else
                                                state <= state;
                                                output <= "11";
                                        end if;
                                when s1 =>
                                        if input='1' then
                                                state <= s0;
                                                output <= "10";
                                        else
                                                state <= state;
                                                output <= "00";
                                        end if;
                        end case;
                end if;
        end if;
end process;
end architecture rtl;
```

7.2 PRÁCA V MODELSIME

Na jednom z predchádzajúcich cvičení bol preberaný prevodník *gray-ovho* kódu do *kódu+3*. Na tomto príklade bude ukázaný postup pri návrhu aplikácie v ModelSime, ako aj jej simulácia. Budú vysvetlené výhody **.do* súborov (určitá forma dávkového súboru), v porovnaní s využívaním GUI (**G**raphical **U**ser **I**nterface) prostredia ModelSimu.

Pre úplnosť je uvedený opäť zdrojový VHDL kód[2] prevodníka:

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

--definovanie vstupov a vystupov
entity gray_to_plus3 is
port
    (
    gray_code       : IN    std_logic_vector (4 downto 1);   --vektor vstupov
    plus3_code      : OUT   std_logic_vector (4 downto 1);   --vektor vystupov
```

[2] Uvedený VHDL kód nie je kompatibilný s VHDL verziou z rku 1987, ktorý je preddefinovanóu verziou starších verzií ModelSimu (napr. v.5.7). Definovať inú verziu VHDL jazyka je možné buď globálne pre všetky spustenia v premennej ..\Modelstech\modelsim.ini, alebo pre konkrétny súbor po kliknutí pravým tlačidlom myši a zadaním verzie VHDL v položke vlastnosti. Najnovšie verzie ModelSimu už majú preddefinovanú verziu VHDL 2002.

```
    ostatne_segmenty  :  OUT  std_logic_vector (11 downto 0)    --vektor nepouzitych segmentov
    );
end entity gray_to_plus3;


-- samotne telo programu
architecture gray_tabulka of gray_to_plus3 is
begin
    ostatne_segmenty  <= "111111111111";                        --pripojenie nepouzitych segmentov na jednotku

    process (gray_code)                                         --prevod gray kodu na plus3 kod
    begin
            case gray_code is
                    when "0000" => plus3_code <= "1100";
                    when "0001" => plus3_code <= "1011";
                    when "0011" => plus3_code <= "1010";
                    when "0010" => plus3_code <= "1001";
                    when "0110" => plus3_code <= "1000";
                    when "0111" => plus3_code <= "0111";
                    when "0101" => plus3_code <= "0110";
                    when "0100" => plus3_code <= "0101";
                    when "1100" => plus3_code <= "0100";
                    when "1101" => plus3_code <= "0011";
                    when others => plus3_code <= "1111";      -- ak nieco ine...
            end case;
    end process;
end architecture gray_tabulka;
```

V ďalšej časti ukážeme ako je možné postupovať s využitím GUI a jednotlivých položiek **menu**, pri návrhu (vytvorení nového projektu), kompilácii a simulácii projektu. Paralelne tiež bude ukázané ako je možné postupovať s využitím príkazov ModelSimu, zadávaných z príkazového riadku ModelSimu.

7.2.1 VYTVORENIE NOVÉHO PROJEKTU

Je výhodné, ak je pre každý nový projekt vytvorený samostatný adresár (**„File->New->Project“**). Nedochádza tak k prepisovaniu dôležitých súborov inými projektmi (ďalšou výhodou je celková prehľadnosť celej hierarchie).

Pre vytváraný projektu prevodníka *gray-ovho* kódu do *kódu+3* zvolíme vhodný názov, napr. *gray_to_plus3*. Následne vytvoríme nový VHDL súbor pomocou „**Create New File**“, resp. vzhľadom na to, že už sme raz tento prevodník vytvárali, stačí ak už vytvorený VHDL súbor pridáme do projektu pomocou „**Add Existing File**“, a pomenujeme ho *gray_to_plus3.vhd.*

Vytvorenie projektu, ako aj pridanie existujúceho súboru do vytvoreného projektu je možné realizovať aj pomocou príkazového riadku ModelSimu, a to nasledujúcimi príkazmi[3]:

project new c:/modeltech_xe_starter/projects/gray_to_plus3 gray_to_plus3
project addfile gray_to_plus3.vhd

[3] Príkaz predpokladá umiestnenie projektu v adresári c:/modeltech_xe_starter/projects/.

kde prvý príkaz vytvorí nový projekt s názvom *gray_to_plus3*, ktorý uloží do adresára (domovský adresár pre daný projekt) zadaného ako argument príkazu. Druhý príkaz pridá už existujúci VHDL súbor, ktorý sa nachádza v domovskom adresári projektu do vytváraného projektu.

7.2.2 KOMPILÁCIA

Po vytvorení projektu je možné prejsť k samotnej kompilácii projektu. V okne „**Workspace**“ pravým tlačidlom myši klikneme na súbor *gray_to_plus3.vhd* a vyberieme „**Compile->Compile Selected***”*.

Aj proces kompilácie je možné realizovať z príkazového riadku a to príkazom **vcom** ktorého parametrom je názov súboru, ktorý chceme kompilovať, v našom prípade je to súbor *gray_to_plus3.vhd.* Konkrétny príkaz vyzerá následovne:

vcom gray_to_plus3.vhd

Samozrejme ako ostatné príkazy ModelSimu, aj tento príkaz môže obsahovať ďalšie voľby, ktoré bližšie špecifikujú správanie (napr. podpora pre určité špecifikácie) kompilátora, teda špecifikujú spôsob, akým sa bude daný vstupný súbor kompilovať. Stručný prehľad tých najzákladnejších volieb tohto príkazu je uvedený v Prílohe I..

7.2.3 ČASOVÁ SIMULÁCIA A DÁVKOVÉ SPRACOVANIE

Po kompilácii je možné realizovať simuláciu obvodu. V menu položke „**Simulate**“ vyberieme „**Start Simulation**“, označíme čo chceme simulovať (v našom prípade je to *work.gray_to_plus3* jednotka, ktorú nájdeme vo *work* knižnici) a potvrdíme tlačidlom **OK**.

Z príkazového riadku ModelSim sa tento postup dá vyvolať príkazom **vsim**, a to následovne:

vsim work.gray_to_plus3

Ďalej je potrebné definovať vstupné stimuly. Toto je možné realizovať v GUI jednoduchým presunutím požadovaných signálov z okna „**Objects**“ (Signals) do okna „**Wave**“. Jednotlivé okná sa dajú zobraziť cez menu ploţku „**View->Debug Windows-><okno_ktoré_chceme_zobraziť>**“. Myšou presunieme požadované signály (*gray code* a *plus_3_code*). Uložíme okno *„***Wave***“* do súboru *wave.do*.(„**File->Save As...***“*).

Opäť, celý postup je možné realizovať jedným príkazom **add wave** z príkazového riadku.

add wave gray_code plus3_code

resp. uloženie okna „**Wave***“* príkazom

write format wave wave.do

Definovanie vstupných stimulov prebieha tak, že vo *„***Wave***“* okne označíme signál, ktorému chceme priradiť vstupný stimul (napr. nech je to 4-tý bit signálu *gray_code*). Pravým tlačidlom myši klikneme na príslušný pin a vyberieme položku[4] *„***Clock***“*, kde definujeme napr. periódu 10ns[5], pre ďalší bit by sme mohli definovať signál s dvojnásobnou periódou predošlého signálu a pod. a tým zaistíme, že 4-bitový vektor signálov *gray_code* nadobudne všetky možné stavy. Následne z menu položky *„***Simulate***“* vyberieme *„***Run->Run 100ns***“* a po prebehnutí simulácie sa v okne *„***Wave***“* zobrazí výsledok simulácie.

Celý postup je samozrejme možné realizovať aj pomocou príkazov z príkazového riadka. Uvedené príkazy definujú vstupné stimuly pre jednotlivé signály.

force -freeze sim:/gray_to_plus3/gray_code(4) 1 0, 0 {5 ns} -r 10
force -freeze sim:/gray_to_plus3/gray_code(3) 1 0, 0 {10 ns} -r 20
force -freeze sim:/gray_to_plus3/gray_code(2) 1 0, 0 {20 ns} -r 40
force -freeze sim:/gray_to_plus3/gray_code(1) 1 0, 0 {40 ns} -r 80

prípadne len ich skrátenú formu v tvare[6]

force gray_code(4) 1 0, 0 {5 ns} -r 10
force gray_code(3) 1 0, 0 {10 ns} -r 20
force gray_code(2) 1 0, 0 {20 ns} -r 40
force gray_code(1) 1 0, 0 {40 ns} -r 80

Časový úsek simulácie je možné určiť príkazom **run <time_units>**, kde **<time_units>** určuje časový úsek simulácie, napr. **run 2500** (simulácia sa vykoná pre časový úsek 2500ns).

Definície jednotlivých stimulov je možné uložiť do *do* súboru a následne vykonať príkazom **do**, ktorého vstupným parametrom je názov *do* súboru. Uložíme ich napríklad do súboru *stimulus.do*, ktorý potom budeme volať príkazom

do stimulus.do

Pre úplnosť je obsah súboru uvedený ešte raz:

restart -force
force -freeze sim:/gray_to_plus3/gray_code(4) 1 0, 0 {5 ns} -r 10

[4] V prípade starších verzií ModelSimu sa uvedený postup mierne líši. Je potrebné použiť položku „Edit” v okne „Signal”.

[5] Príkazovom okne Modelsimu je možné sledovať ekvivaletné príkazy generované Modelsimom.

[6] Uvedené príkazy sú skrátenou verziou automaticky generovaných príkazov. V ďalšom texte budú pre jednoznačnosť uvádzané len automaticky generované príkazy.

force -freeze sim:/gray_to_plus3/gray_code(3) 1 0, 0 {10 ns} -r 20
force -freeze sim:/gray_to_plus3/gray_code(2) 1 0, 0 {20 ns} -r 40
force -freeze sim:/gray_to_plus3/gray_code(1) 1 0, 0 {40 ns} -r 80
run 2500

Príkaz **restart -force** okrem iného nastaví čas simulácie na nulu.

Stimuly sa dajú definovať aj priamo celému bitovému vektoru, a to nasledovne:

force –freeze sim:/gray_to_plus3/gray_code 0000 0, 0001 5, 0010 10, 0011 15, 0100 20, 0101 25, 0110 30, 0111 35, 1000 40, 1001 45, 1010 50, 1011 55, 1100 60, 1101 65, 1110 70, 1111 75 -r 80

Týmto spôsobom (hoci na prvý pohľad zložitejším) je možné definovať ľubovoľný priebeh vstupných stimulov s ľubovoľnou periódou. Pri použití funkcií GUI by bol spôsob definovania stimulov podstatne zložitejší.

Pre úplnosť je uvedený príklad *do* súboru, ktorý zrealizuje kompiláciu, simuláciu nastaví vstupné stimuly a uloží potrebné súbory pod zodpovedajúcimi názvami.

project new c:/modeltech_xe_starter/projects/gray_to_plus3 gray_to_plus3
project addfile gray_to_plus3.vhd
vcom gray_to_plus3.vhd
vsim work.gray_to_plus3
add wave gray_code plus3_code
write format wave wave.do
do stimulus.do

Tento *do* súbor by sme mohli nazvať *project.do* a následne ho z príkazového riadku ModelSimu vykonať, a to príkazom:

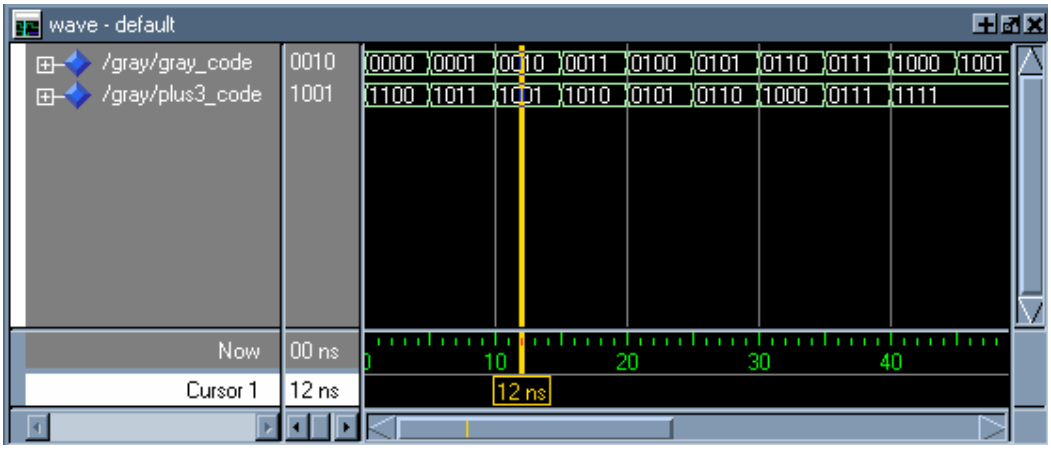
do project.do

Ak v návrhu niečo zmeníme a chceme opätovne spustiť proces kompilácie a simulácie, nepotrebujeme vytvárať znovu nový projekt a pod. *do* súbor upravíme na požadovaný tvar a uložíme ho pod názvom napr. *do_all.do*.

vcom gray_to_plus3.vhd
vsim work.gray_to_plus3
view signals, wave
do wave.do
do stimulus.do

kde súbor *wave.do* je súbor generovaný ModelSimom a preto nemá zmysel ho tu vypisovať. Súbor *stimulus.do* je súbor, do ktorého sme uložili naše vstupné stimuly

(viď. vyššie). Po vykonaní simulácie dostaneme v okne „Wave“ výsledky zobrazené na obr.2.



Obr.2: Výsledok funkčnej simulácie prevodníka Gray na BCD

Rovnakým spôsobom by sme mohli postupovať aj pri simulácii sekvenčného obvodu[7]. Vstupné stimuly budú definované len pre signál *clk*, ktorý je hodinovým signálom. Definovanie vstupného stimulu pre daný signál je možné realizovať nasledovne:

force -freeze sim:/counter_up/clk 1 0, 0 {10 ns} -r 20

Tento príkaz určuje periódu vstupného sitnálu *clk* na hodnotu 20ns. Zdrojový VHDL kód čítača vpred s krokom +1 je uvedený nižšie.

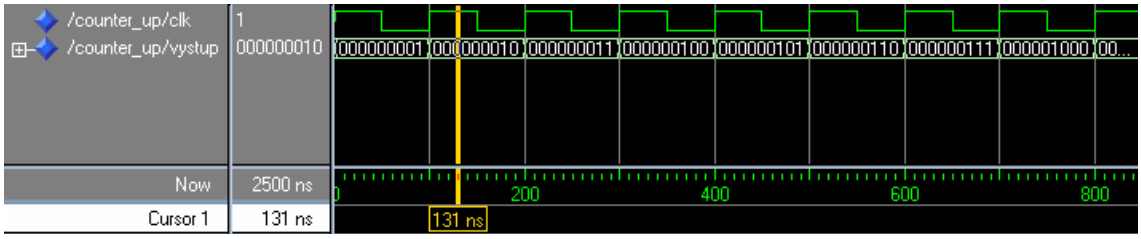
```
library ieee;                                                   --zadefinovanie kniznic
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

--definovanie vstupov a vystupov
entity counter_up is
generic
    (
            N : integer:= 9
    );
port
    (
    clk     : in    std_logic;                                  --hodiny
    vystup  : out   std_logic_vector (N downto 1)               --vystupny vektor
    );
```

[7] Pri simuláciách sekvenčných obvodov je však potrebné definovať hodnoty počiatočných hodnôt v pamätiach sekvenčných obvodov (napr. v našom príklade v klopných obvodoch čítača). ModelSim je v tomto veľmi dôsledný. Pokiaľ počiatočné hodnoty čítača nedefinujeme, je hodnota čítača v ďalších taktoch nedefinovaná a teda v simulácii neuvidíme očakávaný výsledok! V praxi je samozrejme potrebné čítač najskôr resetovať a tým určiť definovanú hodnotu. Inicializáciu je možné realizovať buď s využitím skutočnosti, že po FPGA obvod po inicializácii zvyčajne resetuje všetky klopné obvody. Iným riešením je možnosť využitia dodatočného resetovacieho vstupu.

```
end entity counter_up;

architecture counter_up_arch of counter_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 counter_up_arch;
```



Obr.3: Výsledok funkčnej simulácie čítača vpred o +1

Vnútiť počiatočné hodnoty pre simuláciu v ModelSime je možné napr. využitím inicializácie signálu priamo vo VHDL kóde

```
signal vystup_sig       : std_logic_vector (N downto 1) := “000000000”;   --inicializacia pomocneho signalu
```

Po uvedenej úprave, by ste mali dostať v okne „**Wave**“ výsledok podobný tomu na obr.3 (samozrejme svoju úlohu tu zohráva určenie periódy hodinového signálu *clk* a definovanie počiatočného stavu čítača). Zodpovedajúcich **.do* súborov:

Výpis súboru *all.do*:

```
quit -sim
vcom -2002 -explicit -novitalcheck -no1164 –novital counter_up.vhd
vsim work.counter_up
view signals wave
do wave.do
do stimuls.do
```

Výpis súboru *stimuls.do*:

```
restart -f
force -freeze sim:/counter_up/clk 1 0, 0 {50 ns} -r 100
run 2500
```

Výpis súboru *wave.do*:

```
onerror {resume}
```

```
quietly WaveActivateNextPane {} 0
add wave -noupdate -format Logic /counter_up/clk
add wave -noupdate -format Literal /counter_up/vystup
TreeUpdate [SetDefaultTree]
WaveRestoreCursors {{Cursor 1} {0 ns} 0}
configure wave -namecolwidth 150
configure wave -valuecolwidth 100
configure wave -justifyvalue left
configure wave -signalnamewidth 0
configure wave -snapdistance 10
configure wave -datasetprefix 0
configure wave -rowmargin 4
configure wave -childrowmargin 2
configure wave -gridoffset 0
configure wave -gridperiod 1
configure wave -griddelta 40
configure wave -timeline 0
update
WaveRestoreZoom {0 ns} {1 us}
```

Iným riešením je možnosť využitia explicitného vstupu reset. Aktivácia vstupu reset je možné vnútiť asynchrónnu inicializáciu klopných obvodov čítača, čo dokumentuje nasledujúci VHDL kód:

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

--definovanie vstupov a vystupov
entity counter_up is
generic
    (
            N : integer:= 9
    );
port
    (
    clk     : in    std_logic;                                  --hodiny
    reset   : in    std_logic;                                  --inicializácia citaca
    vystup  : out   std_logic_vector (N downto 1)               --vystupny vektor
    );
end entity counter_up;

architecture counter_up_arch of counter_up is
    signal vystup_sig   : std_logic_vector (N downto 1);        --pomocny signal
            begin
            process (clk, reset)                                --reaguje aj na reset!!!
                    begin
                            if reset = '1' then                 --asynchronny reset
                                vystup_sig <= (OTHERS=> "0");
                            elsif (clk'event and clk='1') then
                                vystup_sig <= vystup_sig + 1;
                            end if;
            end process;
    vystup <= vystup_sig;
end architecture counter_up_arch;
```

Uvedená iniciálizácia explicitným signálom do definovaného stavu je zvyčajne najvhodnejším riešením pokiaľ je inicializácia do definovaného stavu nevyhnutná. Pokiaľ na počiatočnej hodnote nezáleží, je možné realizovať návrh aj bez inicializácie. Takýto postup bol využívaný v predchádzajúcich cvičeniach s využitím návrhového systému Quartus II. Funkčnosť návrhu v cieľovej FPGA súčiastke bola zabezpečená tým, že klopné obvody súčiastky sú po štarte v nejakom definovanom stave (zvyčajne vynulované).

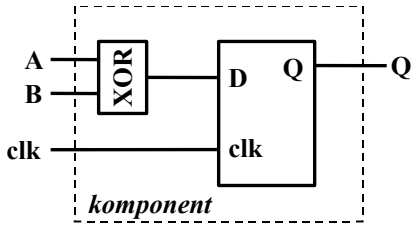
Príklad 2
Realizujte simuláciu čítača +1 v ModelSime s využitím explicitného vstupu Resest, ktorý je využitý v predchádzajúcom VHDL kóde.

Príklad 3
Overte, že pri syntéze čítača +1 s inicializáciou signálu vystup_sig v prostredí Quartus II je inicializácia ignorovaná.

7.3 MODULÁRNY NÁVRH V JAZYKU VHDL

Veľkou výhodou jazyka VHDL je, že podporuje tzv. modulárnosť, t.j. celý návrh je možné rozdeliť do menších častí a následne jednotlivé časti „namapovať“ do výsledného návrhu. Výsledný návrh na najvyššej úrovni tak nebude obsahovať konkrétnu funkciu obvodu, ale len mapovanie jednotlivých častí. Funkcia obvodu (návrhu) je pritom vhodným spôsobom rozdelená do jednotlivých menších častí. Vo VHDL terminológii sa takéto *menšie* časti nazývajú komponenty. Výhodou rozdelenia celého návrhu do komponentov je väčšia prehľadnosť a v konečnom štádiu aj lepšia laditeľnosť návrhu (ak nastane chyba v návrhu, je lokalizovaná v konkrétnom komponente). Využitie komponentov má význam aj v prípade, ak sme už raz realizovali určitú funkciu, ktorú je možné využiť v novom obvode (návrhu). Je ju možné jednoducho namapovať do vytváraného projektu a tým ušetriť množstvo času s jej prepisovaním. V predchádzajúcich cvičeniach sme napr. takto využívali deličku hodinového signálu.

V nasledujúcej časti je uvedená realizácia komponentu (obvodu), ktorý je následne viacnásobne využitý v ďalšom VHDL návrhu a sprístupnený ako komponent. Nad hlbším významom, resp. funkciou obvodu sa netreba zamýšľať, momentálnym cieľom je naznačenie princípu začlenenia (mapovania) časti obvodu do výslednej realizácie. Schéma komponentu je zobrazená na obr.4.



Obr.4: Schéma VHDL komponentu

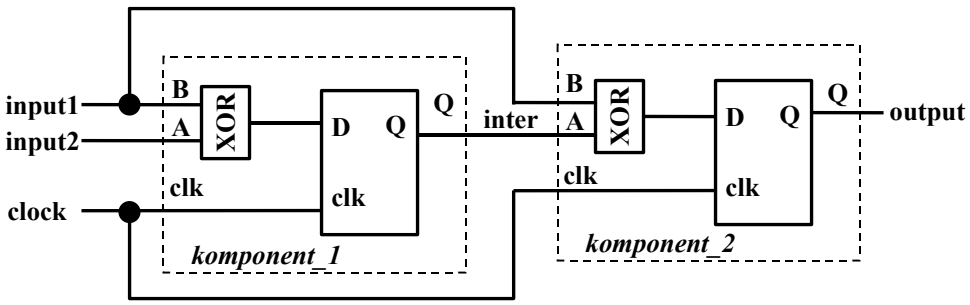
Komponent je realizovaný nasledujúcim VHDL kódom

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

entity komponent is
port
        (
        A       : in    std_logic;
        B       : in    std_logic;
        clk     : in    std_logic;
        Q       : out   std_logic
        );
end entity komponent;

architecture komponent_arch of komponent is
signal D        : std_logic;
begin
        dko: process (clk)
begin
                if (clk'event and clk='1') then
                        Q <= D;
                end if;
        end process dko;
        D <= A xor B;
end architecture komponent_arch;
```

Po vytvorení komponentu je možné vytvoriť návrh, v ktorom je tento komponent využitý. Na obr.5 je grafická reprezentácia obvodu, ktorý je navrhnutý s využitím vytvoreného komponentu. Obvod je vytvorený kaskádnym zapojením dvoch identických komponentov. Podobne je možné prepojiť aj odlišné komponenty.



Obr.5 Princíp návrhu s využitím komponentov

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

library work;
use work.all;

entity use_component is
port
        (
        input1  : in    std_logic;
        input2  : in    std_logic;
        clock   : in    std_logic;
        output  : out   std_logic
        );
end entity use_component;

architecture use_component_arch of use_component is

        component komponent is
        port
                (
                A       : in    std_logic;
                B       : in    std_logic;
                clk     : in    std_logic;
                Q       : out   std_logic
                );
        end component;

        signal inter    : std_logic;

begin

        komponent_1   : komponent port map (A=>input1, B=>input2, clk=>clock, Q=>inter);
        komponent_2   : komponent port map (A=>inter, B=>input2, clk=>clock, Q=>output);

end architecture use_component_arch;
```

Komponent sa deklaruje v časti architektúry, v bloku deklarácii. V podstate deklarácia komponenty má podobný tvar ako deklarácia entity, s tým rozdielom, že kľúčové slovo *entity* nahradíme kľúčovým slovom *component*.

Na prepojenie dvoch komponent použitých v tomto návrhu je vytvorený signál *inter*, taktiež deklarovaný v bloku deklarácii architektúry.

Komponent sa mapuje do návrhu následovne:

<identifikator> : <nazov_komponentu> port map (<co> => <na_co>, ...);

kde

<identifikator>

- určuje jedinečný názov pre konkrétnu inštanciu komponenty – povinné

<nazov_komponenty>

- určuje názov komponentu, ktorý mapujeme

<co>

- určuje port (in/out) komponentu, ktorý mapujeme na port (signál) <na_co> v našom návrhu

Príklad 4
Vytvorte vo VHDL ľubovoľný návrh s využitím aspoň 2 komponentov a odsimulujte celý projekt v Modelsime[8].

PRÍLOHA I

Základné príkazy Modelsimu

ModelSim poskytuje množstvo príkazov, ktoré sa dajú použiť z príkazového riadku, pričom veľká časť sa dá použiť aj priamo pomocou grafického ovládania (GUI). Uvedené príkazy je možné zoskupovať aj do tzv. „*do*“ (dávkových súborov) a týmto spôsobom automatizovať proces návrhu. Aj keď na prvý pohľad dávkové súbory pôsobia pomerne archaicky, treba si uvedomiť, že pri väčších projektoch je ich použitie výrazne efektívnejšie ako grafické ovládanie. Preto budeme tomuto ovládaniu venovať počas cvičení náležitú pozornosť.

Cieľom tejto prílohy nie je opísať všetky príkazy Modelsimu, ale uviesť len tie najpoužívanejšie príkazy, ktoré budú postačovať k vykonaniu základnej (funkčnej) simulácie. Podrobný zoznam všetkých príkazov je možné nájsť napr. na adrese[9]

http://www.actel.com/documents/oem_cmds.pdf,

resp. na internete (ak si napríklad v ***google*** necháte vyhľadávať *modelsim command reference*).

quit [-f | -force | -sim]

-sim

Uvoľní aktuálny návrh v simulátore, bez toho aby ukončil ModelSim. Všetky súbory otvorené v simulátore sa zatvoria, vrátane WLF súboru (*vsim.wlf*).

[8] Pri preklade projektov, ktoré sú zložené z viacerých zdrojových VHDL súborov je v Modelsime potrebné realizovať preklad VHDL súborov viac ako raz. Pri prvom preklade nie je možné vyriešiť všetky referencie. Kompletnosť prekladu je jasne viditeľná v okne „Workspace“.

[9] Uvedený manuál je pre obmedzenú verziu Modelsimu firmy Actel. Podobné obmedzené verzie Modelsimu poskytujú aj iní výrobcovia, napr. Altera a Xilinx. Tieto obmedzené verzie podporujú napr. len návrh (napr. **časovú simuláciu**) v obvodoch konkrétneho výrobcu. Filozofia podpory jednotlivých výrobcov sa však výrazne líši. Napr. firma Altera poskytuje možnosť využitia **bezplatnej verzie** Modelsim-Altera pre všetkých užívateľov **plnej** verzie Quartus II. Uvedená verzia je dostupná aj v laboratóriu PC FEI resp. na KEMT. Xilinx poskytuje obmedzenú verziu Modelsim-Xilinx pre všetkých záujemcov po bezplatnej registrácii. Je však potrebné zdôrazniť, že Modelsim-Xilinx má podstatne **výraznejšie obmedzenia** ako Modelsim-Altera, čo je pri väčších projektoch výrazné obmedzenie. Pre potreby výučby je však pre študentov Modelsim-Xilinx optimálnou voľbou.

vcom [-work | -2002 | -explicit | -novitalcheck | -no1164 | -novital ...] <filename>
Príkaz **vcom** skompiluje zdrojový VHDL kód do určenej pracovnej knižnice (default je work knižnica). Tento príkaz môže byť volaný v rámci ModelSimu, ako aj z príkazového riadku Windows. Taktiež môže byť volaný počas simulácie.
Kompilované knižnice sú závislé na verzii (nedá sa použiť knižnica kompilovaná s verziou 5.7 v simulácii využívajúcej *vsim* z verzie 5.8, a pod.). V tomto prípade je potrebné aktualizovať knižnice argumentom **-refresh** príkazu **vcom**.

-93
Vypína podporu pre VHDL-1987 a 2002. - voliteľné
-87
Vypína podporu pre VHDL-1993 a 2002. - voliteľné
-2002
Určuje, že kompilátor podporuje VHDL-2002 špecifikáciu. - default hodnota
-explicit
Táto voľba robí ModelSim kompatibilným so všeobecne zaužívaným postupom.
-work <meno_knižnice>
Určuje logické meno, resp. cestu ku knižnici, ktorá má byť mapovaná do logickej knižnice work. Voliteľné. Implicitne sú navrhované jednotky pridané do work knižnice.
-novital
Táto voľba spôsobí, že **vcom** použije VHDL kód pre VITAL procedúry, a nie časovo optimalizované a primitívne balíky zabudované v jádre simulátora. Máme možnosť zadať vo VITAL procese miesta prerušenia, a taktiež je možné vykonať krokovanie v rámci VITAL procedúr, na odladenie návrhu. Všetky VITAL údaje sa dajú sledovať v okne *Premenných*, resp. *Signálov*.
-novitalcheck
Vypína dodržiavanie VITAL 2000 kontroly, v prípade ak používame VITAL 2.2b. - voliteľné
-no1164
Spôsobí, že zdrojové súbory budú skompilované bez vylepšení vstavanej verzie IEEE std_logic_1164 balíka. To vedie obyčajne k dlhším simulačným časom VHDL kódov, ktoré využívajú signály typu std_logic.

vsim [<library_name>.<design_unit>]
Príkaz **vsim** sa využíva k vyvolaniu VSIM simulátora, resp. k zobrazeniu výsledkov predošlej simulácie (v prípade zadania *-view* voľby). Pre simuláciu je možné zadať VHDL konfiguráciu, alebo pár entita/architektúra, resp. Verilog modul, alebo konfiguráciu. V prípade ak je zadaná VHDL konfigurácia, nie je dovolené zadávať aj architektúru.
Počas spracovania **vsim** určí, či došlo k modifikácii zdrojových súborov od poslednej kompilácie. Tento príkaz sa dá vyvolať okrem príkazového riadku ModelSimu taktiež z príkazového riadku Windows. V prípade ak chceme manuálne prerušiť proces spracovania, použijeme klávesu Break.

view [*] [-height <n>] [-width <n>] [-title {New Window Title}] <window_type>
Príkaz **view** otvorí ModelSim okno, a umiestní ho do popredia. Ak chceme odstrániť okno, použijeme príkaz **noview**. Samostatný príkaz **view** bez parametrov zobrazí zoznam otvorených okien.

*****
Spôsobí otvorenie všetkých okien
-height <n>
Určuje výšku okna v pixeloch. - voliteľné
-width <n>
Určuje šírku okna v pixeloch. - voliteľné
-title {New Window Title}
Určuje názov konkretného okna. Zátvorky sú potrebné v prípade ak názov obsahuje medzery.
<window_type>
Určuje typ ModelSim okna, ktoré sa má zobraziť. - povinné
Možné typy sú:
assertion, dataflow, list, memory, process, signals, source, structure, variables, wave

do <filename> [<parameter_value>]
Príkaz *do* vykoná príkazy ktoré sa nachádzajú v makro súbore. Makro súbor môže mať ľubovoľný názov a príponu.

<filename>
Určuje názov makro súbora, ktorý sa má vykonať. Povinné. Názvom môže byť úplna cesta k súboru (vrátane), resp. relatívny názov súbora (relatívna cesta).
V prípade ak je **do** príkaz vyvolaný z príkazového riadka, cesta je relatívna k aktuálnemu pracovnému adresáru.
<parameter_value>
Určuje hodnoty, ktoré sú predané odpovedajúcim parametrom $1 až $9 v makro súbore. Jednotlivé hodnoty sú oddelené medzerami.
napr. **do macros/stimulus 100** - tento príkaz vykoná súbor *macros/stimulus* (vykoná prákazy, ktoré sa nachádzajú v tomto súbore) a odovzdá hodnotu 100 parametru $1 v makro súbore.

restart [-force]
Tento príkaz zabezpečí opakované načítanie častí návrhu a nastaví čas simulácie na nulu. Načítané sú len tie časti, ktoré boli zmenené. (SDF súbory sú pri každom reštarte načítavané)
- force
Určuje, že k reštartu simulácie dôjde bez vyžiadania potvrdenia v **pop-up** okne.

force [-freeze] [-repeat <time>] [-cancel <time>]

Umožňuje interaktívne zadávanie stimulov VHDL signálom. Vzhľadom k tomu, že ako všetky ostatné príkazy aj tento príkaz sa dá uložiť v *makro* súbore a vytvoriť týmto spôsobom zložité sekvencie stimulov.

-freeze

Táto voľba „zamrazí" signál na určenej hodnote, až kým nie je znova prenastavená na inú hodnotu, resp. pokiaľ nepoužijeme príkaz **noforce** a neuvolníme „zmrazenie".

-cancel <time>

Zruší **force** príkaz v špecifikovanom čase. Čas je relatívny voči aktuálnemu času, pokiaľ nie je špecifikovaný absolútny čas, ktorý sa vyznačuje prvým znakom @. Nulová hodnota zruší **force** na konci aktuálnej periódy. - voliteľné

-repeat <time>

Opakuje **force** príkaz, kde **<time>** je čas, v ktorom sa začína opakovať cyklus. Čas je relatívny k aktuálnemu času. - voliteľné

run [<time_steps> [<time_units>]]

Tento príkaz rozšíri simuláciu o špecifikovaný počet časových jednotiek.

<time_steps> [<time_units>]

Špecifikuje počet časových jednotiek pre dĺžku simulácie. Číslo môže byť zadané ako zlomok, alebo v absolútnom tvare ktorému predchádza znak @. – voliteľné. Naviac sú taktiež voliteľné aj časové jednotky: *fs, ps, ns, us, ms* alebo *sec.*

project [new <home_dir> <prj_name>] [addfile <filename>]

Príkaz **project** je príkaz používaný k vykonaniu operácii tykajúcich sa projektu (vytvorenie, otvorenie, zavretie, pridanie súboru do projektu ...). Tento príkaz musí byť volaný mimo procesu simulácie (ak máme spustenú simuláciu, musíme ju najskôr ukončiť **quit -sim**).

new <home_dir> <prj_name>

Vytvorí nový projekt s umiestnením v špecifikovanom adresári **<home_dir>** s požadovaným názvom **<prj_name>**

addfile <filename>

Pridá špecifikovaný súbor **<filename>** do aktuálneho otvoreného projektu – voliteľné.

add wave <item_name>

Príkaz **add wave** pridá nasledujúce položky do *„Wave"* okna: VHDL signály a premenné a tiež Verilog spoje a registre. Užívateľsky definované zbernice je možné taktiež pridať. **<item_name>** špecifikuje názov položky, ktorá sa má pridať do *„***Wave***"* okna. Voliteľné. Sú povolené tzv. *divoké znaky* ako napr. „*".

write format [list | wave <filename>]

Tento príkaz zaznamená názvy a vlastnosti VHDL položiek aktuálne zobrazených v *„***List***"* alebo *„***Wave***"* okne. Vytvorený súbor je v prevažnej miere tvorený **add list** alebo **add wave** príkazmi, môže obsahovať zopár iných príkazov. Tento súbor sa môže

opätovne vykonať *do* príkazom a tak znovu rekonštruovať formát „**List**“ resp. „**Wave**“ okna pri opätovne vykonávanej simulácii.

list | wave
Určuje, že buď to položky „**List**“ alebo „**Wave**“ okna sú zaznamenané. – nevyhnutné
<filename>
Určuje názov výstupného súboru, kde sa majú údaje uložiť. - nevyhnutné

PRÍLOHA II

Inštalácia Modelsim Xilinx

Kde získať inštalačný súbor

Spoločnosť Xilinx na svojej WEB stránke poskytuje zadarmo k stiahnutiu ModelSim Xilinx Edition III (MXE III), ktoré má určité limity v porovnaní s plnou verziou ModelSim-u. Na adrese http://www.xilinx.com/ise/mxe3/download.htm sa dá stiahnúť najnovšia verzia tohto softvéru. Tento odkaz je špecifický pre verziu III, a preto pri vydaní novšej verzie sa pravdepodobne tento odkaz stane neplatným, preto je lepšie priamo ísť na ich domovskú stránku http://www.xilinx.com a preklikať sa na aktuálnu verziu ModelSimu. Aktuálnou verziou je momentálne verzi 3.6.0a. Uvedený odkaz obsahuje inštalačný súbor (v **.zip* forme) pre *Plnú* ako aj *Štartovaciu* verziu MXE III. *Štartovacia* verzia je voľne šíriteľná „skúšobná“ verzia, na rozdiel od *Plnej* verzie, na ktorej činnosť potrebujeme mať zakúpenú platnú licenciu. Pre potreby cvičení postačuje voľná *Štartovacia* verzia. Jej obmedzenia v prípade časovej simulácie budú diskutované v ďalších cvičeniach.

Často kladené otázky

Q1: Čo je MXE III?
MXE III je HDL simulátor, určený na simuláciu návrhov len pre Xilinx technológiu. Simulačné jádro je založené na ModelSim PE (Professional Edition) od Mentor Graphics.

Q2: Aké sú hlavné rozdiely medzi MXE III a ModelSim PE?
MXE III predstavuje ModelSim PE, s nikoľkými limitmi (výkon, limit v počte príkazov (executable line limit)).

Q3: Čo je to MXE III Starter?
MXE III Starter je identické MXE III, avšak s väčsími limitmi, ako MXE III.

Q4: Do akej hustoty je MXE III Starter vhodným simulátorom?
MXE III Starter je vhodný pre väčšinu ISE WebPACK obvodov.

Q5: Aké sú hlavné rozdiely medzi MXE III Starter, MXE III, ModelSim PE a ModelSim SE?

Vlasnosť	Starter	MXE III	PE	SE
Výkon	20% PE	40% PE	PE	až do 300% PE[10]
Zmiešaný jazyk	nie	nie	áno	áno
Odlaďovacie prostredie	áno	áno	áno	áno
Waveform prehliadač	áno	áno	áno	áno
Verilog PLI	áno	áno	áno	áno
VHDL FLI	nie	nie	nie	áno
Analyzátor výkonu	nie	nie	nie	áno

Postup inštalácie

Po stiahnutí inštalačného súboru a jeho rozbalení spustíme *setup.exe*. Následne vyberieme "*MXE III Starter - Limited Version of MXE III (Free)*". Preklikáme sa cez úvodné okná, až sa dostaneme k zadaniu, kde sa má ModelSim nainštalovať. (Tu treba dať pozor, aby zadaná cesta neobsahovala v názve prázdne znaky, teda medzery – mohli by byť s nimi neskôr pri behu programu problémy). Najvhodnejšie je ponechať inštalátorom zadané umiestnenie.

V nasledujúcom okne "**Select Library Installation Option**" vyberieme "**Full VHDL**". Táto voľba nám nainštaluje všetky Xilinx VHDL knižnice a zdrojové súbory. Voľba "**VHDL Custom**" umožňuje špecifikáciu konkrétnych častí, ktoré sa majú nainštalovať. Táto voľba je však vhodná len pre skúsených užívateľov.

V ďalšom (poslednom) z konfiguračných okien je možné určiť umiestnenie vytvorených zástupcov v ponuke "**Štart**" – túto položku využijeme v procese registrácie.

Po ukončení inštalácie si od nás inštalátor vypýta ďalšie doplňujúce informácie, ktoré potvrdíme "**YES**".

Postup registrácie

Na to, aby sme mohli využívať práve nainštalovaný ModelSim, je potrebná bezplatná registrácia a získanie licenčný súboru, ktorý je zviazaný so sériovým číslom vášho disku. Ak ste v čase inštalácie pripojený na internet, po ukončení inštalácie sa vám vo vašom default prehliadači zobrazí stránka, kde je možné registrovanie.

Pravdepodobne ešte nemá nik z vás vytvorené svoje konto v Xilinx a práve preto sa vás týka odkaz "**Register**", kde si od vás vyžiadajú základné informácie. Licenčný súbor vám bude zaslaný na e-mailovú adresu, ktorú zadáte pri registrácii.

Ak momentálne nemáte možnosť pripojenia na internet, spravte nasledujúce:

[10] 300% je dosiahnutých použitím Gate-level VITAL modelov

Cez ponuku "**Štart**" otvorte textový súbor "*License Request Instructions*", ktorý nájdete v položke ModelSim, kde ste si nechali vytvoriť programových zástupcov. V tomto súbore nájdete alternatívnu URL adresu, ktorú môžete použiť z ktoréhokoľvek miesta, kde máte prístup na internet a vyžiadať si licenčný súbor týmto spôsobom.

V tejto alternatívnej adrese sú obsiahnuté určité informácie týkajúce sa vášho počítača ako je napr.:

ei - Eval číslo

ea - MAC adresa sieťovej karty (v prípade ak váš počítač sieťovou kartou disponuje)

ds - sériove číslo disku

in - IP adresa počítača (v prípade ak je počítač vybavený sieťovou kartou)

Táto linka odkazuje na stránku, na ktorú by ste sa dostali, keby ste mali v čase inštalácie pripojenie na internet.

Pre úplnosť je uvedený postup registrácie:

Vytvoríte si nový účet - "**Create an Account**"

V položke "**User ID**" zadáte prihlasovanie meno (login) do systému. Pri zadávaní hesla je potrebné zvoliť heslo, ktoré má minimálne 6 znakov a obsahuje aspoň jednu číslicu.

Pri zadávaní e-mailovej adresy zadajte adresu, ktorá je stále platná a máte k nej prístup, lebo licenčný súbor bude zaslaný práve na túto adresu. Je však potrebné sa odhlásiť a znova prihlásiť (zadajte celú alternatívnu URL adresu a následne potvrďte "**Continue**"). Systém si od Vás vyžiada ďalšie doplňujúce údaje, ktoré sú povinné (či zadáte pravdivé, alebo nepravdivé údaje záleží len a len na vás „☺).

Po týchto krokoch na vami zadanej e-mailovej adrese nájdete licenčný súbor, ktorý si skopírujte niekde na váš disk, najlepšie do adresára "*c:\Modeltech_xe_starter\win32xoem*" (v každom prípade si to "*Licensing Wizard*" skopíruje práve do tohto adresára).

Cez ponuku Štart spustite "**Licensing Wizard**", zadáte umiestnenie licenčného súbora "*license.dat*" a potvrdíte. "**Licensing Wizard**" bude žiadať, aby ste potvrdili pridanie cesty k licenčnému súboru do systémovej premennej, čo je pre funkčnosť ModelSimu nevyhnutné. Taktiež do systémovej premennej *PATH* sa pridá cesta k binárnym súborom ModelSimu.

Povzbudivá správa je, že ak ste sa dostali až tu, znamená to, že Vaša registrácia prebehla v poriadku a v plnej miere môžete využívať nainštalovaný ModelSim - Xilinx.