Slovenská technická univerzita v Bratislave

FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Študijný program: POČÍTAČOVÉ A KOMUNIKAČNÉ SYSTÉMY A SIETE

Vizualizácia modelov digitálnych systémov

Tímový projekt

Vedúci projektu: Ing. Dominik Macko

Tím č. 2: Bc. Michal Ilko, Bc. Zuzana Krnáčová, Bc. Pavol Mnich, Bc. Martin Štepanovič,

Bc. Katarína Ťažká

November, 2011

Obsah

1 Úvod

Táto dokumentácia obsahuje analýzu problematiky, ktorú bude riešiť náš tím počas dvoch semestrov z predmetu Tímový projekt I a II.

Úvod do problematiky zahŕňa analýzu dostupných riešení na vizualizáciu modelov digitálnych systémov. Rozoberá problematiku zobrazovania návrhov digitálnych systémov v programovacích jazykoch Verilog, VHDL a System C. Programy, ktoré boli vytvorené na Fakulte informatiky a informačných technológií ako bakalárske alebo diplomové práce, budú použité počas riešenia tohto projektu.

Našim hlavným cieľom je vytvorenie univerzálneho programu, ktorý bude zobrazovať nie len návrhy jedného programovacieho jazyka, ale všetkých troch.

Hlavnou motiváciou k tvorbe takého programu bola analýza dostupných riešení na internete. Existuje veľa programov, ktoré vizualizujú digitálne systémy, no podpora viacerých vstupných údajov je to, čo mnohým chýba. Možno práve takýto program eliminuje vytváranie ďalších programov, ale vývojári sa budú skôr zaoberať rozširovaním funkcionalít už existujúcich riešení.

Analýza problematiky, požiadavky systému a hrubý návrh riešenia sú opísané v tomto dokumente.

2 Verilog

Verilog je špecifikačný a opisný jazyk využívaný pri navrhovaní a dokumentácií elektronických systémov. Verilog HDL umožňuje návrhárom ponoriť sa pri návrhu do rôznych úrovní abstrakcie. Spolu s VHDL, System C patri medzi najvýznamnejšie opisné jazyky. V týchto jazykoch je možné opísať štruktúru a správanie sa digitálnych systémov do takých detailov, že na základe tohto opisu je možné simulovať správanie systému.

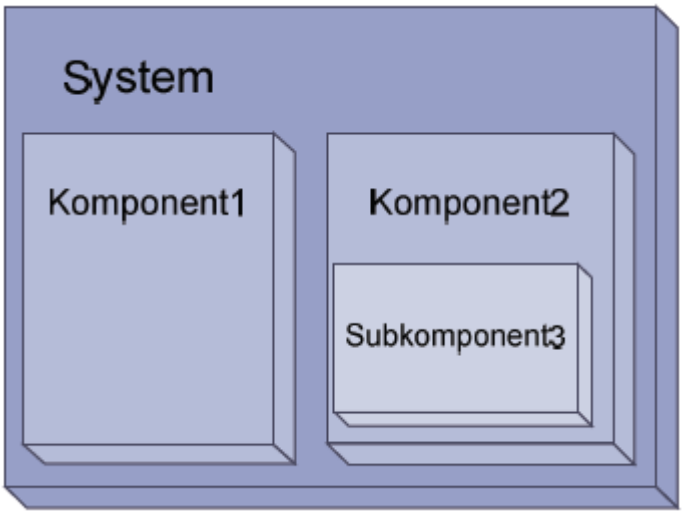
2.1 História

Verilog HDL bol vytvorený spoločnosťou Automated Integrated Design Systems v roku 1985. Spoločnosť bola v tom čase súkromným vlastníctvom Dr. Prabhu Goela, autora PODEM algoritmu. Tvorcom Verilog HDL bol Phil Moorby, ktorý bol neskôr hlavným vývojárom Verilog-XL. Spoločnosť, neskôr premenovaná na Gateway Design Automation, rýchlo rástla vďaka jej úspechu s Verilog-XL a v roku 1989 sa stala súčasťou spoločnosti Cadence Design Systems. Verilog bol navrhnutý ako jazyk určený na simuláciu. Myšlienka využiť Verilog pri syntéze prišla oveľa neskôr. (1)

Koncom osemdesiatych rokov začal byť uprednostňovaný štandard HDL pred komerčnými jazykmi akým bol aj Verilog. Pravdepodobne preto sa spoločnosť Cadence Design Systems rozhodla v roku 1990 otvoriť jazyk Verilog pre voľné používanie, čo predstavuje zrod OVI (Open Verilog International). Dovtedy bol Verilog HDL licencovaným komerčným jazykom, vlastníctvom spoločnosti Cadence Design Systems. Po vzniku OVI v roku 1991 niekoľko malých spoločnosti začalo vyvíjať Verilog simulátory. (1)

2.2 Základy jazyka Verilog

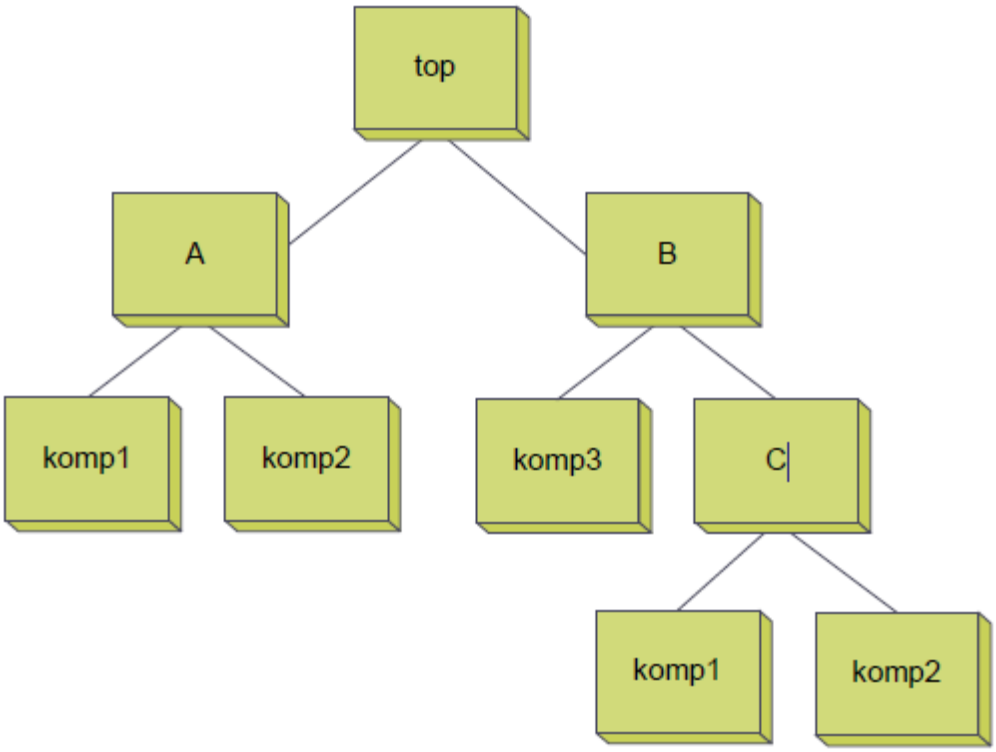
Každý Verilog model je kompozíciou modulov. Modul je základnou jednotkou celého modelu. Modul, ktorý pozostáva z iných modulov sa označuje ako rodičovský modul (parent) a modul, ktorý je časťou rodičovského modulu sa označuje ako potomok (child). Na obrázku č.1 je model vytvorený zo štyroch modulov a to: System, Komponent1, Komponent2, Subkomponent3. Modul System je rodičovským modulom modulov Komponent1 a Komponent2. Modul Komponent2 je rodičom modulu Subkomponent3. Modul Subkomponent3 je potomkom modulu Komponent2 a moduly Komponent1 a Komponent2 s (2)ú potomkami modulu System. (1)



Obrázok 1 Model systému

2.3 Moduly

V jazyku Verilog sú moduly presne definovane menom a typom. Pritom je veľmi dôležité dodržať podmienku, aby moduly na jednej hierarchickej úrovni mali jednoznačné pomenovanie.



Obrázok 2. Príklad definície modulov

Na obrázku č.2 je znázornený jeden modul typu top s menom top, dva moduly typu typ2 pomenované A a C, jeden modul B typu typ2 a päť modulov typu typ3 pomenované komp1, komp2, komp3 a komp1, komp2. Keďže každá inštancia typu modulu je identifikovaná nielen typom a menom, ale tiež celou cestou v hierarchii modulov, mená rôznych modulov môžu byť identické, pokiaľ sa tieto moduly nenachádzajú na rovnakej úrovni v hierarchii. (1)

2.4 Porty

Ide o rozhrania využívané na komunikáciu a prenos dát. V jazyku Verilog poznáme 3 typy portov. input, output a inout.

Input – rozhranie, ktoré slúži na načítavanie dát do modulu

Output – rozhranie, ktoré slúži na výstup dát z modulu

Inout – tento typ portu umožňuje vstup aj výstup dát z modulu

Pri vytváraní portov musí byť každý z portov pomenovaný.

2.5 Modelovacie štruktúry (1)

V jazyku Verilog je k dispozícií sedem typov preddefinovaných modulov, alebo komponentov:

- Parametre (Parameters)
- Siete (Nets)
- Registre (Registers)

- Primitívy a inštancie (Primitives and Instances)
- Procedurálne bloky (Procedural Blocks)
- Úlohy, funkcie (Task/Function definitions)

Parametre slúžia na konštanty, ktorých hodnoty sú známe v čase kompilácie. Parameter je viditeľný v rámci modulu, funkcie alebo bloku, v ktorom je deklarovaný.

Siete slúžia na prepájanie jednotlivých komponentov modulu s možnosťou zadefinovať oneskorenie.

Registre sú úložné elementy štyroch druhov:

- Reg – bitový dátový typ registra umožňujúci uložiť bitové vektory šírky 1 až 1000000
- Integer – celočíselný 32-bitový dátový typ registra
- Time – celočíselný nezáporný 64-bitový dátový typ registra
- Real (a Realtime) – registre 64-bitových dát v pohyblivej desatinnej čiarke

Primitívy sú nasledujúce preddefinované moduly:

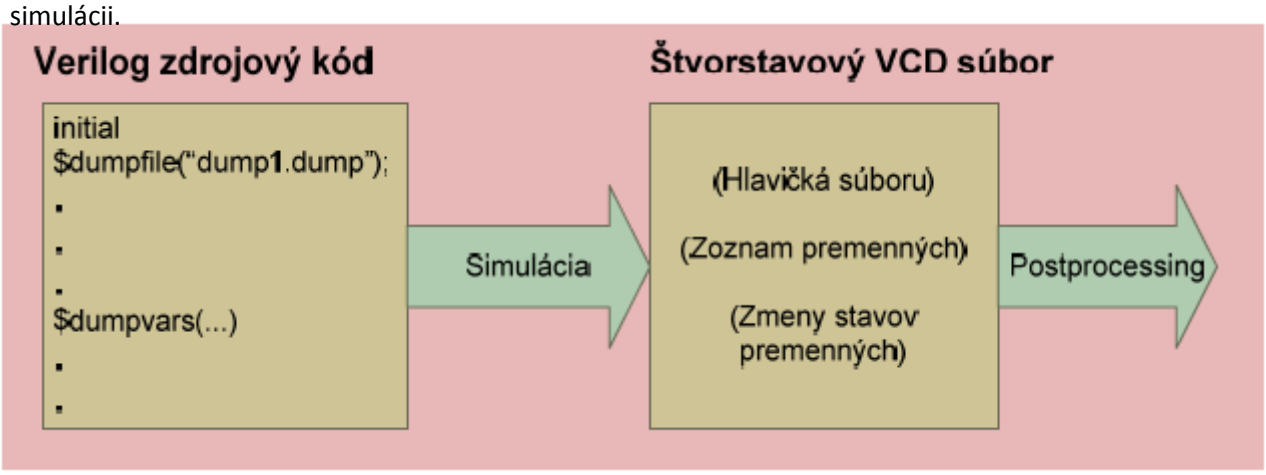
- 1-vstupové: buf, not
- 1-výstupové: and, nand, or, nor, xor, xnor
- Trojstavové: bufif0, notif0, bufif1, notif1
- Čítacie: pullup, pulldown
- MOS prepínače: cmos, rcmos, nmos, rnmos, pmos, rpmos
- Obojsmerné prepínače: tran, rtran, tranif0, rtranif0, tranif1, rtranif1

Procedurálne bloky sú časť jazyka Verilog umožňujúca modelovať sekvenčné správanie. V blokoch sa jednotlivé operácie vykonávajú sekvenčne, ale jednotlivé bloky sú vykonávané paralelne.

Úlohy (Tasks) a funkcie (functions) môžu byť deklarované vo vnútri modulov, ale nesmú byť deklarované v procedurálnych blokoch. Volanie úlohy alebo funkcie nesmie predchádzať ich deklaráciu v kóde. Úlohy môžu byť volané len v procedurálnych blokoch, pričom samotná úloha je príkaz, nemôže vystupovať ako operand vo výraze. Funkcie sú využívane ako operandy vo výrazoch, tiež môžu byť volané z procedurálnych blokov, iných funkcií alebo úloh.

2.6 VCD súbory

Value Change Dump súbor obsahuje informácie o hodnotách premenných a ich zmenách počas simulácii.



Obrázok 3. VCD súbor

VCD súbory sú dobre čitateľné aj pre človeka. Jeho obsah je možné rozdeliť do troch častí, prvá časť je hlavička, druhá pozostáva z definícií a tretia zo samotných zmien stavov premenných. V hlavičke a v časti definícii sa nachádzajú informácie, ako sú verzia simulátora, dátum vykonania simulácie, zoznam sledovaných premenných, časová mierka a oblasť, v ktorej sa sledujú premenné (module, task, function, fork a begin). Zvyšok obsahu VCD súboru predstavujú zmeny stavov premenných, ktoré sú zoradené podľa času vo formáte: #čas_zmeny, nový_stav a identifikátor premennej. (1)

2.7 Analýza diplomovej práce Vizualizácia Verilog modelov digitálnych systémov

V tejto kapitole je rozobraná diplomová práca Michala Nosala, v ktorej opísal svoj prístup k problematike vizualizácie Verilog HDL modelov digitálnych systémov.

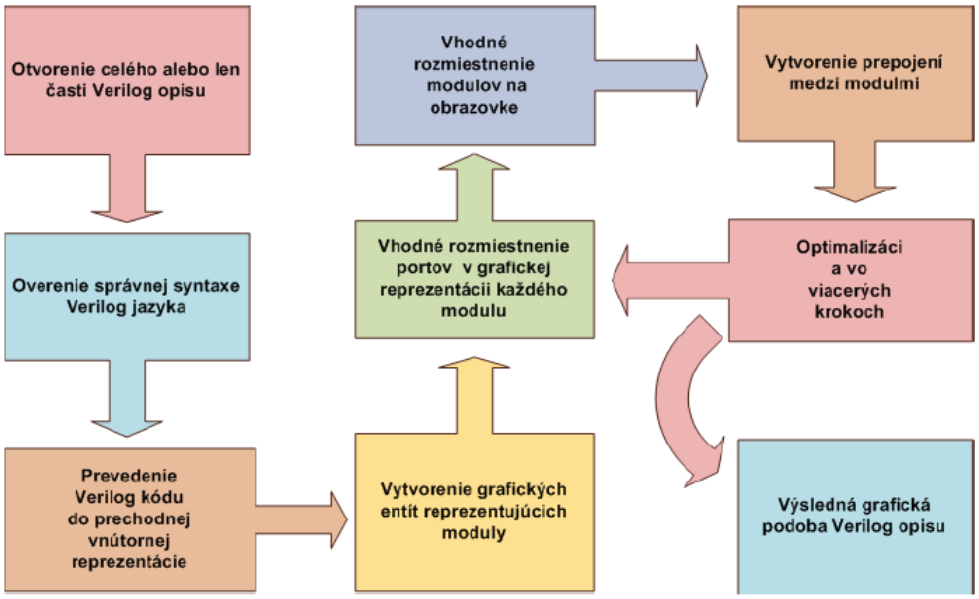
2.7.1 Vizualizácia HDL opisu štruktúry

Pri vizualizácii HDL opisov dochádza k prevedeniu zápisu v podobe HDL kódu do grafickej podoby, ktorá je viac zrozumiteľná pre laikov. Takto vizualizovaná štruktúra je tiež vhodná na verifikáciu samotného návrhu.

Na to, aby sme mohli vizualizovať model systému v jazyku Verilog je potrebné vyriešiť nasledovné problémy:

1. Prevedenie Verilog kódu do vnútornej reprezentácie opisu. V tomto kroku je potrebné vyriešiť dva problémy. Jedným problémom je syntaktická analýza Verilog kódu, ktorá musí byť vykonaná pred transformáciou do vnútornej reprezentácie a druhým problémom je voľba vhodnej vnútornej reprezentácie Verilog opisu.
2. Prevedenie štruktúry z vnútornej reprezentácie do grafickej podoby
3. Samotne zobrazenie štruktúry a to nájsť správne rozloženie objektov a ich prepojenie tak, aby výsledok vizualizácie bol čo najjednoduchšie čitateľný.

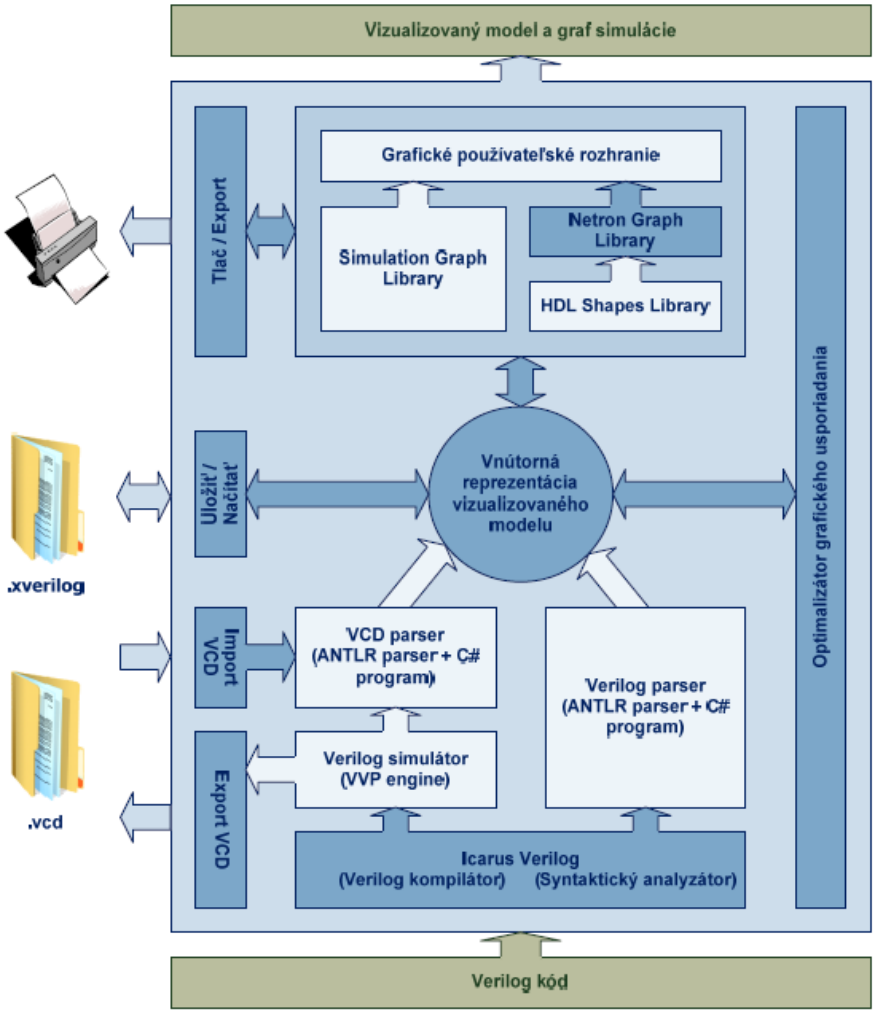
Na obrázku č.4 vidíme detailnejšie popísaný proces vizualizácie Verilog HDL modelu.



Obrázok 4. Proces vizualizácie Verilog HDL modelu

2.7.2 Architektúra systému

Na nasledujúcom obrázku je zobrazený diagram s architektúrou aplikácie slúžiacej na vizualizovanie Verilog HDL modelov digitálnych systémov.



Obrázok 5. Architektúra systému

V nasledujúcich kapitolách sú opísané jednotlivé časti architektúry systému.

2.7.2.1 Icarus Verilog (Syntaktický analyzátor, kompilátor)

Ide o syntaktický analyzátor, ktorý bude kontrolovať syntax vizualizovaného Verilog modelu.

2.7.2.2 Verilog parser

Pomocou parsera je skontrolovaný Verilog kod pretransformovaný do vnútornej reprezentácie s ohľadom na štruktúru a nie správanie, ktoré pri vizualizacii nie je podstatné. Na túto úlohu si autor aplikacie vytvoril vlastný parser využitím generátora parsera.

2.7.2.3 VCD parser

Výsledky simulácií sú uchovávané vo VCD textovom súbore. Syntax VCD súboru nieje explicitne kontrolovaná, keďže samotný súbor nie je vytváraný človekom a teda nepredpokladá sa v ňom chyba.

2.7.2.4 Vnútorná reprezentácia Verilogmodelu

Vnutorná reprezentácia Verilog modelu je realizovaná ako XML subor. Keďže vo vnútornej reprezentácii modelu nie sú potrebné informácie o funkcionalite respektíve správaní modulov, ale o štruktúre. Na čo postačujú možnosti ktoré nám ponúka XML.

2.7.2.5 Optimalizátor grafického usporiadania

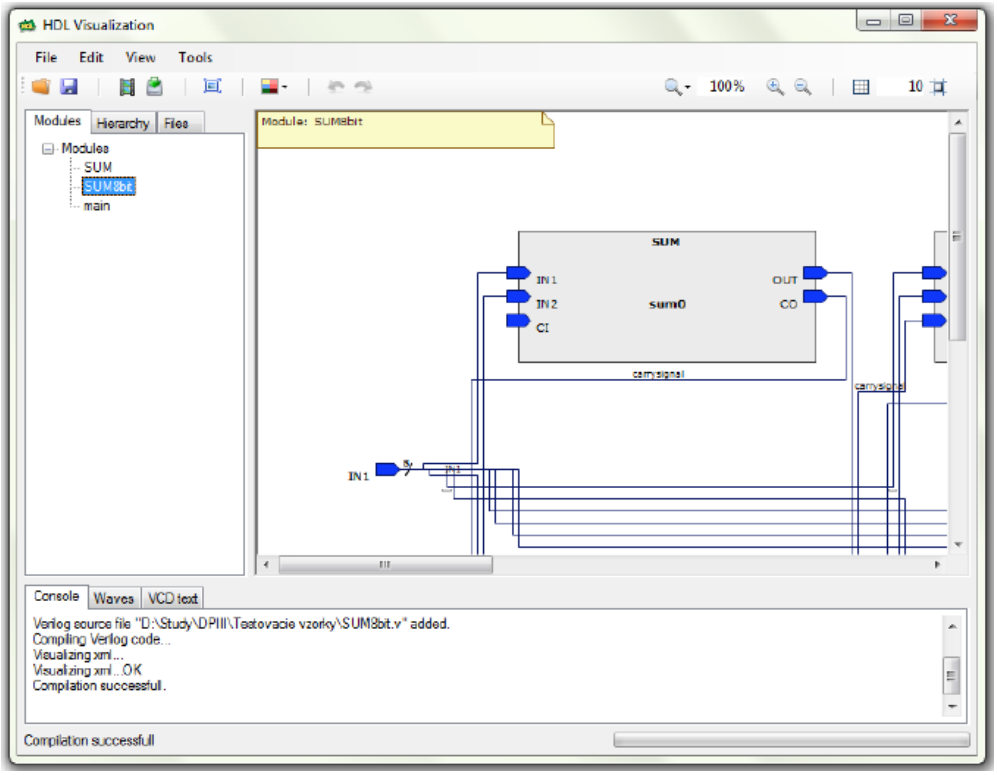
Táto časť programu realizuje optimalizačný proces usporiadania vizualizovaných objektov na obrazovke. Hlavné parametre sledované pri vykresľovaní objektov sú: počet prekrížení hrán, minimálna plocha, minimálna, suma dĺžok hrán…

2.7.2.6 Simulation Graph Library

Tento koponent slúži na vizualizáciu výsledkov simulácie. Jeho vstupom je vnútorná rerezentácia štruktúry vo formate XML.

2.7.3 Grafické používateľské rozhranie

Grafické používateľské rozhranie predstavuje dôležitú časť programu, ktorá zabezpečuje ovládanie ostatných komponentov programu a poskytuje pouzívateľovi požadovanú funkcionalitu. Pomocou Grafického používateľského rozhrania sú do aplikácie vkladané vstupy pre samotný výstup vizualizácie. Na nasledujúcom obrázku je ukážka grafického rozhrania aplikácie.



Obrázok 6. Ukážla grafického rozhrania

2.7.4 HDL Shapes Library

Pri implementácii navrhovaného systému bolo dalším krokom úprava prevzatého projektu VHDLShapesLibrary ktorý obsahoval definíciu tried VHDLEntity odvodenej od triedy Shape z Netron Graph Library knižnice, triedy VHDLPort odvodenej od triedy ShapeSinker a triedy HDLConnection odvodenej od triedy Connection tiež z knižnice Netron Graph Library. Ďalšou triedou je VHDLProject, ktorej členské metódy slúžia na spracovanie opisu štruktúry Verilog modelu v XML forme. (1)

Pre zachovanie spatnej kompatibility s programom VHDLVisualiser boli implementovane nove triedy VerilogModule, VerilogPort, VerilogConnection a VerilogProject. Okrem týchto nových tried tento projekt obsahuje triedy ConnectionLayout a DominoExtensionMethods, ktorých autormi sú Juraj Petráš a Dominik Macko. Tieto triedy bolo tiež potrebné jemne upraviť tak, aby ich bolo možné využiť na grafickú optimalizáciu nielen pôvodného VHDL opisu, ale aj nového Verilog opisu štruktúry v XML formáte. (1)

3 VHDL

Jazyk VHDL je formálny programovací jazyk, vyvinutý v roku 1987 a uznaný ako štandard IEEE. Používa sa na popisovanie hardvérových súčastí rôznych systémov. Patrí medzi vyššie opisné jazyky a má medzi nimi stabilné zastúpenie. Vznik VHDL bol podmienený potrebou popísať navrhované integrované obvody, napríklad pre výrobcov týchto súčiastok, pre ich simulovanie, či dokumentovanie. (3)

Označenie VHDL vzniklo z anglického VHSIC Hardware Description Language (jazyk pre opis hardvéru), kde VHSIC znamená Very-High-Speed Integrated Circuit (veľmi rýchle integrované obvody). Vývojári vychádzali z programovacieho jazyka Ada, a v roku 1993 prešiel tento štandard väčšími úpravami.

Jazyk VHDL umožňuje opísať štruktúru komponentov, ako aj ich správanie (funkcionalitu). Veľkou výhodou je možnosť simulácie, čo ušetrí časté prototypovanie, a tým sa šetria prostrediky pri vývoji. Tiež predchádza chybám v návrhu a zjednodušuje optimalizovanie systémov.

3.1 Syntax

Syntax jazyka obsahuje dve základné časti: deklaráciu entity a telo architektúry. Entity predstavujú akoby štruktúru systému a obsahujú porty – vstupy a výstupy. Architektúra popisuje správanie sa entít a viaže sa na konkrétnu entitu. Preto sa definuje až za deklaráciou entity.

Základná definícia entity vyzerá nasledovne:

```
entity meno_entity is
    port (názvy_signálov: mode typ;
            názvy_signálov : mode typ;
            :
            názvy_signálov : mode typ
            );
    end [meno_entity] ;
```

Syntax architektúry:

```
architecture meno_architektúry of meno_entity is
    -- Deklarácie komponentov, signálov, konštánt, funkcií, procedúr
    begin
    -- príkazy
    :
end meno_architektúry;
```

3.2 Porty

Vstupy a výstupy entít sa nazývajú porty. Patria sem:

- IN – vstupný port;
- OUT – výstupný port;
- BUFFER – výstup so spätnou väzbou;

- INOUT – vstupno výstupný port;
- LINKAGE – neznámy smer toku.

3.3 Základy syntaxe

Syntax jazyka používa určite slová alebo značky, ktoré majú svoj význam. V nasledujúcej časti je popísanie týchto špeciálnych konštrukcií.

3.3.1 Identifikátory

Sú to jednoznačné názvy objektov. Môžu obsahovať písmená, číslice a podčiarkovník, ktorý nemôže byť na poslednom mieste. Pričom prvý znak musí byť písmeno.

3.3.2 Rezervované slová

Jazyk ich používa na špeciálne operácie. Sú to napríklad IN, OUT, CONSTANT, MAP, PORT, AND, OR, atď.

3.3.3 Znaky a reťazce znakov

Znaky sa nchádzajú v apostrofoch a reťazce v úvodzovkách. Znak pre priradenie do premennej je <=, a pre inicializáciu premennej **:=.**

3.3.4 Komentáre

Tak ako v každom inom programovacom jazyku je aj tu možnosť vkladania komentárov do zdrojového kódu. V tomto prípade ide o kombináciu znakov **-- *Komentár***.

3.4 Prístupy opisu

V jazyku VHDL existujú dva prístupy opisu architektúry systému. Prvý je opis správania. Vtedy sú opísané výstupy systému na základe ich vstupov. Nie je dôležité ako sa to vykoná, ale čo sa vykoná. Medzi základné typy opisov správania patria:

- process – deklarovaný v architektúre, príkazy sú vykonávané sekvenčne
- if – konštrukcia môže byť použitá len v procese, pracuje rovnako ako pri iných programovacích jazykoch, podmienky sa kontrolujú postupne, kým aspoň jedna nie je pravdivá;
- case – postupnosť príkazov na základe hodnoty určitého výrazu;
- loop – opakované vykonávanie príkazov. Príkaz **next** ukončí vykonávanie a pokračuje ďalšou iteráciou slučky, príkaz **exit** skončí vykonávanie slučky, a pokračuje príkazmi za ňou;
- wait – zastavenie vykonávania, pokiaľ nenastane určitá udalosť.

Druhý prístup je opis pomocou štruktúry architektúry. Ten popisuje, aké komponenty treba použiť a ako ich navzájom poprepájať.

3.5 Dátové typy

Opisný jazyk VHDL definuje dátové objekty, ktoré majú svoju deklaráciou, typ a hodnotu. (4) Medzi najznámejšie dátové objekty patria:

- konštanta – definovaná nemenná hodnota, deklarácia sa nachádza v deklaračnej časti, prípadne v procese (použiteľná iba vnútri procesu)

constant *meno_konštanty: typ [:= inicializačná hodnota];*

- premenná – hodnota, ktorá môže byť menená, jej deklarácia a použitie je možné len v procese

variable *meno_premennej: typ [:= inicializačná hodnota];*

- signál – definovaný v deklaračnej časti architektúry, jeho hodnota je zmenená až po určitom oneskorení

Preddefinované dátové typy:

Typ	Rozsah	Príklad
boolean	FALSE, TRUE	variable TEST : boolean := FALSE;
integer	Celé čísla	constant CONST1 : integer := 123;
natural	Celé kladné čísla plus nula	variable VAR1 : natural := 2;
time	Celé číslo závislé na implementácii	variable DELAY : time := 5 ns;

3.6 Analýza diplomovej práce Vizualizácia VHDL modelov digitálnych systémov

3.6.1 Opis riešenia

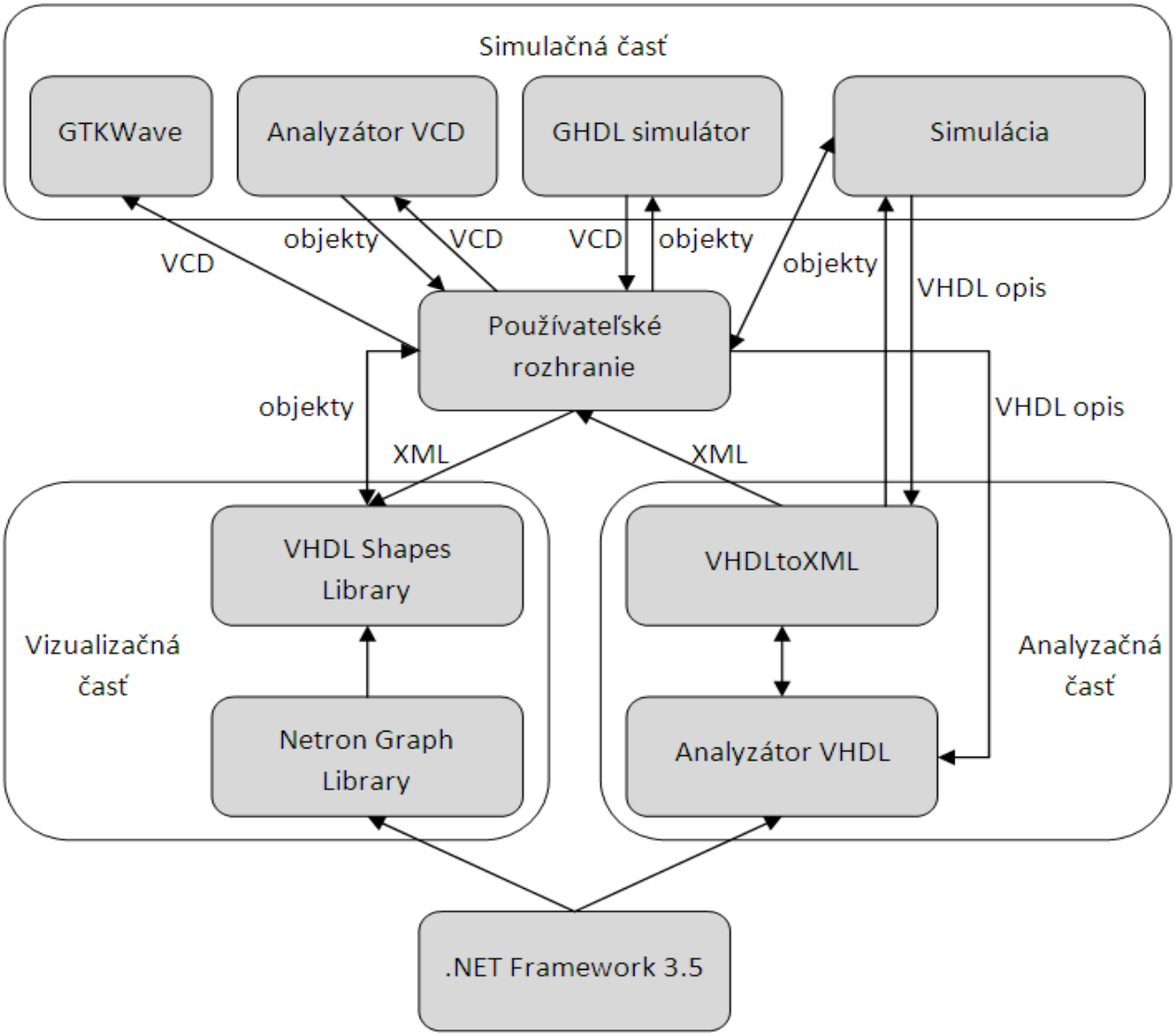
Autor tejto práce Ing. Dominik Macko opísal vizualizačný nástroj pre VHDL model. (5) Analyzoval niekoľko dostupných riešení, z ktorých potom ďalej vychádzal pri návrhu a spôsobu implementácie vizualizačného riešenia. Väčšina požiadaviek vychádzala priamo zo zadania, no niektoré bolo nutné doplniť. Špecifikované požiadavky na navrhované vizualizačné prostredie:

Požiadavky systému

Funkcionálne	**Nefunkcionálne**
Načítanie VHDL modelu zo súboru.	Jednoduchosť ovládania.
Transformácia VHDL modelu štruktúry na zodpovedajúci prehľadný schematický zápis.	Kvalitná výsledná vizualizácia.
Zmena usporiadania objektov danej hierarchickej úrovne.	Zachovanie hierarchie modelu.
Export schematického zobrazenia modelu do formátu obrázka.	Prenositeľnosť a jednoduchosť inštalácie.
Uloženie vizualizovaného návrhu pre neskoršie úpravy.	
Simulácia VHDL modelu.	
Vizualizácie simulácie VHDL modelu vo sfére štruktúry, prípadne správania sa.	

3.6.2 Architektúra systému

Návrh systému pozostáva z troch základných častí: Analyzačná, vizualizačná a simulačná časť.



Obrázok 7. Architektúra systému

Pre správnu vizualizáciu návrhu je potrebné, aby sa po otvorení VHDL súboru previedol zdrojový kód do schematického zápisu, čiže na určitú hierarchickú úroveň komponentov, v ktorej budú zadefinované všetky potrebné detaily komponentov. Na túto operáciu je potrebný parser, ktorý prevedie zdrojový kód do tohto zápisu. Autor to vyriešil tak, že najprv si vygeneroval parser pomocou dostupných riešení, a až potom ho použil. O to všetko sa stará analyzačná časť, ktorá má na vstupe VHDL zdrojový kód a výstup z nej tvorí XML súbor s opisom prechodovej reprezentácie. Je tu nutné riešiť aj identifikáciu objektov pre ich správne zobrazovanie.

Za formát prechodovej reprezentácie bol zvolený XML, lebo je to najrozšírenejší formát, a existuje k nemu veľa dostupných nástrojov pre jeho spracovanie. Ďalšou jeho výhodou je hierarchické zobrazenie komponentov a ich vlastností.

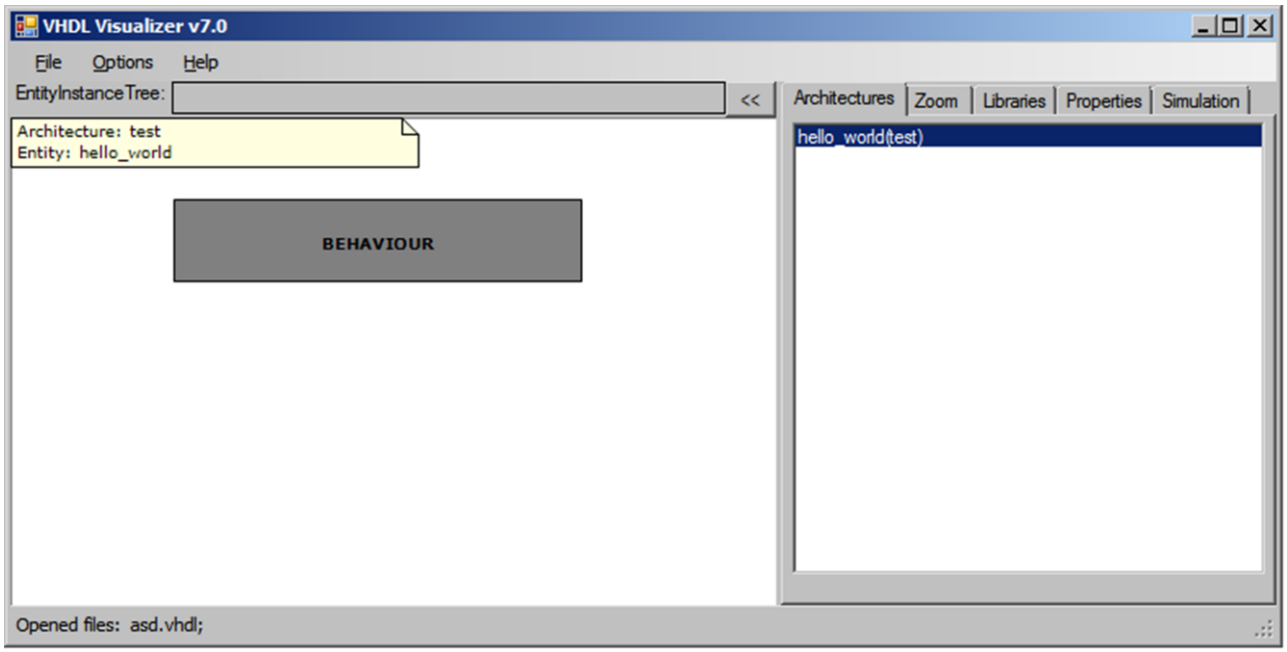
Vizualizačná časť transponuje objekty z prechodovej reprezentácie do grafického zobrazenie

systému. Obsahuje algoritmus na optimalizáciu umiestnenia na zobrazovanej ploche. Tieto informácie, pomocou neustálej komunikácie, zapíše späť do XML súboru. Ide hlavne o umiestnenie a iné zobrazovacie vlastnosti komponentov.

Na obrázku sú popísané aj niektoré knižnice pre prácu s objektami, ich spracovanie a zobrazenie. V tejto časti práce nie je nutné detailne popisovať tieto knižnice. Ak sa niektoré z nich budú ďalej používať, ich popis sa uvedie v návrhu riešenia.

3.6.3 Používateľské rozhranie

Používateľské rozhranie budú tvoriť okna, ktoré sú bežne známe z prostredia operačných systémov. Používateľ bude mať na dosah všetky prvky potrebné pre svoju prácu. Aplikácia zabezpečí prepojenie modulov a ich následnosť. Napríklad, ak používateľ vyberie akciu otvorenia súboru, súbor sa otvorí a posunie analyzačnej časti. Tam sa o jeho ďalšie spracovanie postará ďalší modul.



Obrázok 8. Používateľské rozhranie aplikácie

3.6.4 Prechodová reprezentácia

Ako už bolo spomenuté, pre ukladanie prechodovej reprezentácie modelu je najvýhodnejšie použiť formát XML. V ňom sa budú rozlišovať 3 základné kategórie:

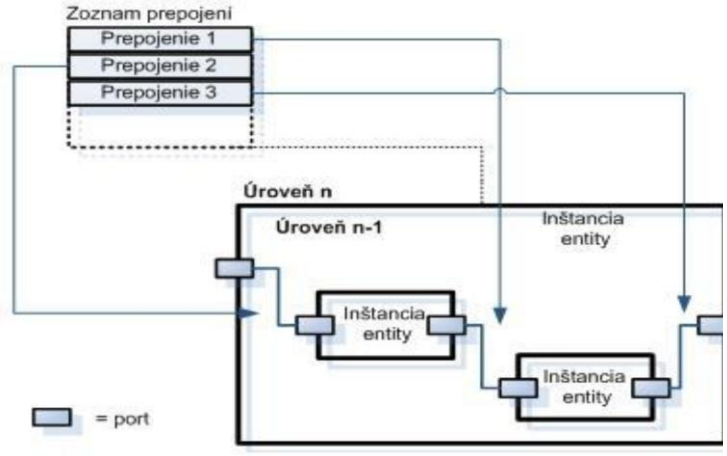
- inštancia entity: Obsahuje názov inštancie entity (jednoznačný identifikátor), názov deklarácie entity, názov architektúry, zoznam portov, zoznam prepojení. Názvy deklarácie a architektúry majú len informačný charakter;
- porty inštancie entity: Každá inštancia entity má svoj zoznam portov. Porty majú tieto parametre:
  - názov portu – jednoznačný identifikátor;
  - režim portu – z hľadiska vizualizácie sa podľa tohto režimu vykreslí typ portu, a z hľadiska simulácie je jeho význam v tom, že používateľ nemôže meniť výstupné porty (OUT);

- typ portu – totožný s typom portu vo VHDL opise. V simulácii určuje, aké hodnoty môže používateľ portu nastaviť;
- počet pinov portu – informatívny charakter, odstraňuje viacnásobné zisťovanie počtu portov počas simulácie;
- hodnota portu.

- prepojenie portov – signály

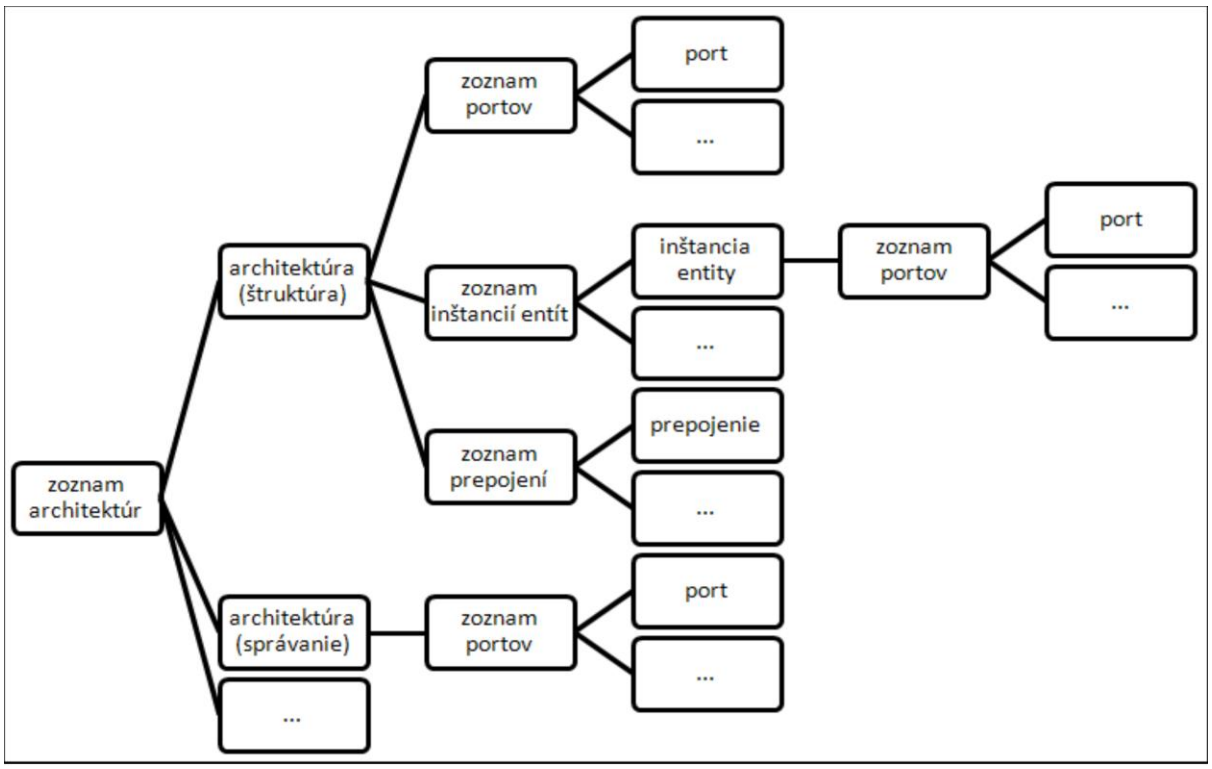
Signály vyjadrujú spojenie dvoch portov a určujú vzťahy medzi inštanciami entít nimi určenými. Každé prepojenie má tieto parametre:

- názov prepojenia (tak ako v pôvodnom VHDL opise);
- názov zdrojového portu a inštancie entity;
- názov cieľového portu a inštancie entity;
- intervalový názov zdrojového a cieľového portu – špecifikuje, na ktorý pin portov je prepojenie pripojené.



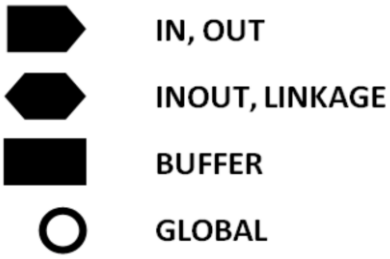
Obrázok 9. Zoznam prepojení entity na úrovni n

Pre uchovanie hierarchie VHDL opisu nie je nutné udržiavať hierarchiu inštancií entít, ale stačí ak uchováme informácie o architektúre s jej portami, prepojeniami a inštanciami entít. Na nasledujúcom obrázku je znázornená štruktúra XML súboru. Sú v nej definované 4 typy uzlov: architektúra, port, inštancia entity a prepojenie.



Obrázok 10. Štruktúra XML súboru

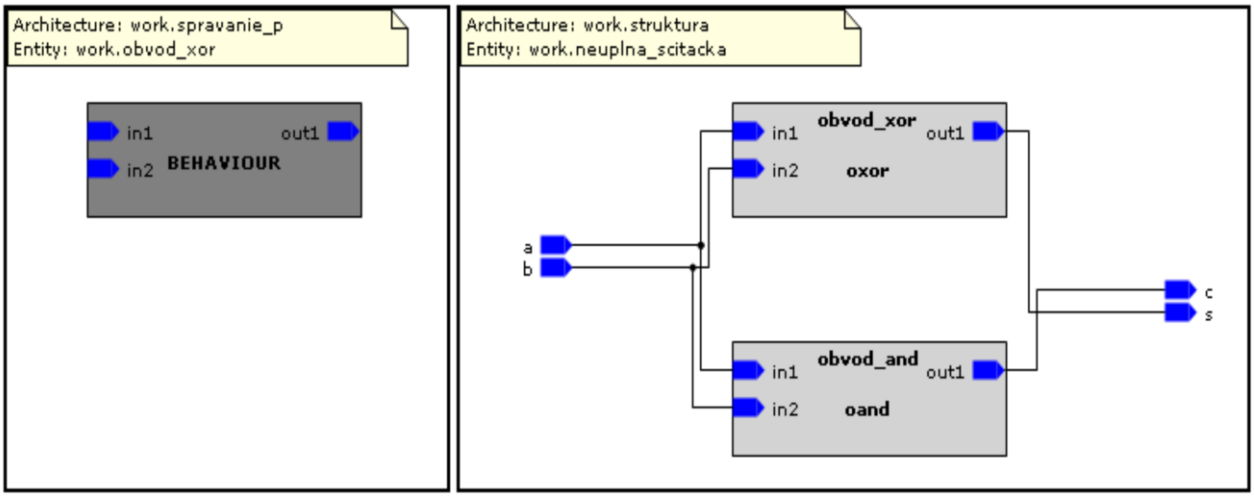
3.6.5 Zobrazenie a simulácia modelu



Obrázok 11. Zobrazenie

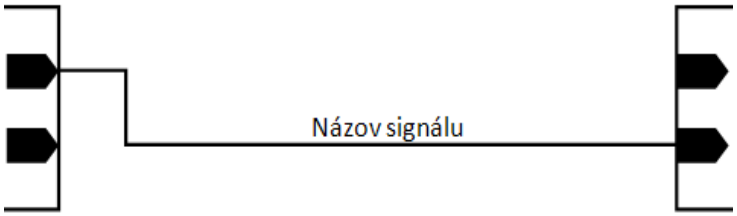
V prechodovej reprezentácii sa nachádzajú všetky potrebné informácie pre zobrazenie objektov. Každá Architektúra predstavuje jednu hierarchickú úroveň. Grafické zobrazenie obsahuje jednotlivé objekty týchto architektúr.

Na nasledujúcich obrázkoch je znázornené, ako vyzerajú jednotlivé časti modelu.



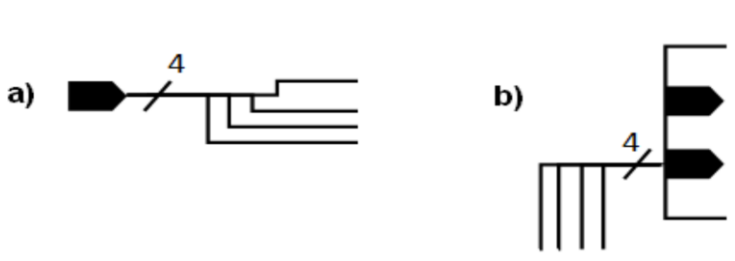
Obrázok 12. Vizualizácia architektúry správania (vľavo) a štruktúry (vpravo)

Na tomto obrázku vidieť inštanciu entity ako tzv. „čiernu skrinku“. To znamená, že nevieme nič o jej vnútornej štruktúre, ale poznáme iba jej vstupy a výstupy. Architektúra opísaná
správaním je jednoznačne odlíšená tmavšou farbou. Neobsahuje názov, ale iba text BEHAVIOUR.



Obrázok 13. Prepojenie medzi inštanciami entít

Prepojenie je čiara, ktorá spája dva porty. Je pri nej napísaný aj názov prepojenia. Signály sa môžu aj vetviť a spájať viacero inštancií entít. Ak je signál viacbitový, a každý bit ide do inej inštancie, vtedy sa zobrazí jeho delenie, prípadne spájanie.



Obrázok 14. Delenie a spájanie 4-bitového signálu

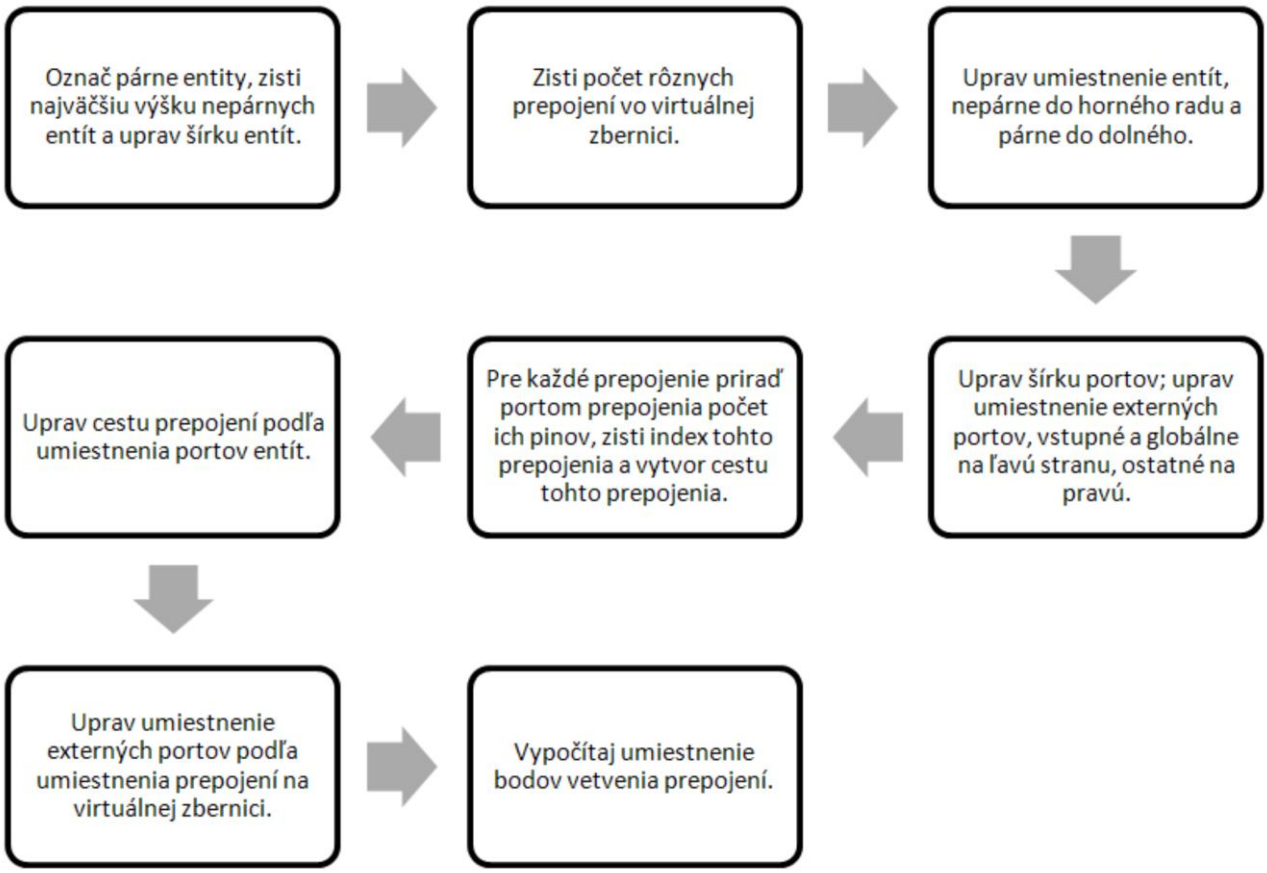
3.6.6 Usporiadanie objektov

Keďže zobrazenie objektov bez prekrývania, či iných chýb je pomerne zložité, použil sa na to

algoritmus, ktorý ich udržiava na správnych pozíciach. Entity sa nachádzajú v dvoch radoch, medzi ktorými sú cesty prepojení v tzv. zbernici. Entity sú usporiadané podľa poradia tak, že párne sú v spodnom a nepárne v hornom rade. Medzi jednotlivými entitami je pevne zadaná šírka, v ktorom sa nachádzajú prepojenia.

Porty, ktoré sa nenachádzajú v inštanciách, sú zobrazené po stranách plochy. Globálne a vstupné vľavo, ostatné vpravo. Každé prepojenie bude mať rezervovanú inú časť zbernice, čo zabezpečí, aby sa navzájom neprekrývali. Prekrývať sa môžu iba vtedy, ak vstupujú do rovnakého portu.

Na nasledujúcom obrázku je zobrazená postupnosť operácií, ktoré algoritmus vykonáva.



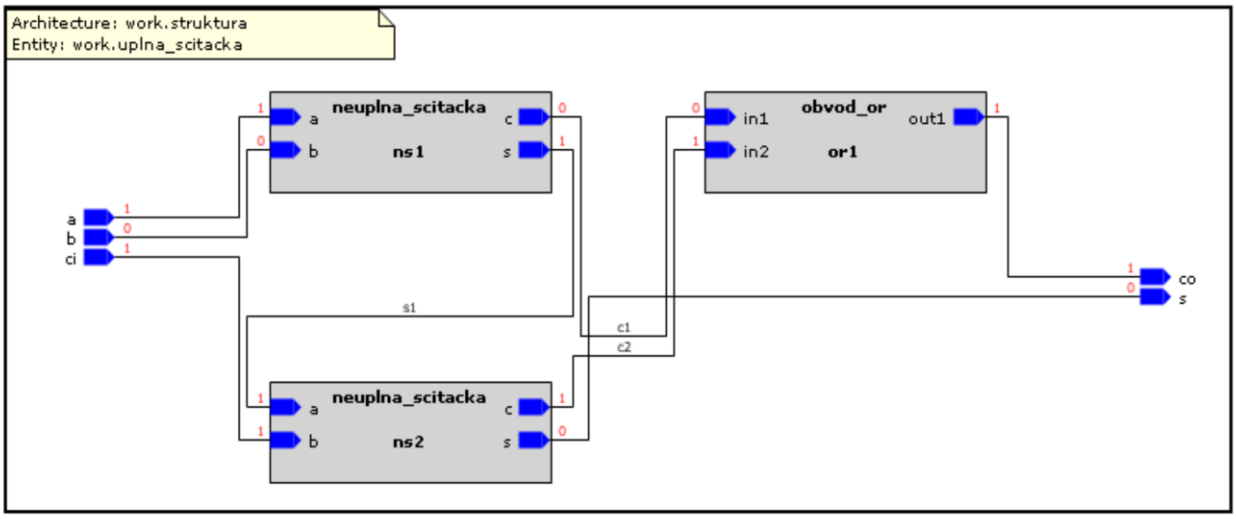
Obrázok 15. Postupnosť akcií algoritmu

3.6.6.1 Priebeh simulácie

Informácie pre simuláciu sú pripravené už pri analýze VHDL modelu. Každému portu sa priradí názov a ak ide o interný port inštancie, tak sa pred neho pridá reťazec s názvom inštancie oddelený bodkou. Vstupné porty architektúry nemajú priradenú hodnotu a všetky ostatné porty sú vyjadrené iba pomocou týchto vstupných portov. Pri interaktívnej simuláciu sa pri zadaní hodnôt vstupným portom prepočítajú hodnoty všetkých ostatných.

Nasledujúci obrázok predstavuje jednu z možností simulácie, kde každý port má zobrazenú

svoju hodnotu.



Obrázok 16. Priebeh simulácie

4 SystemC

SystemC je knižnica štandardného jazyka C++ umožňujúca modelovanie technických prostriedkov, ako aj vnorených programových prostriedkov súčasne na rôznych úrovniach abstrakcie. Je to schopnosť, ktorá tradičným špecifikačným a opisným jazykom chýba.

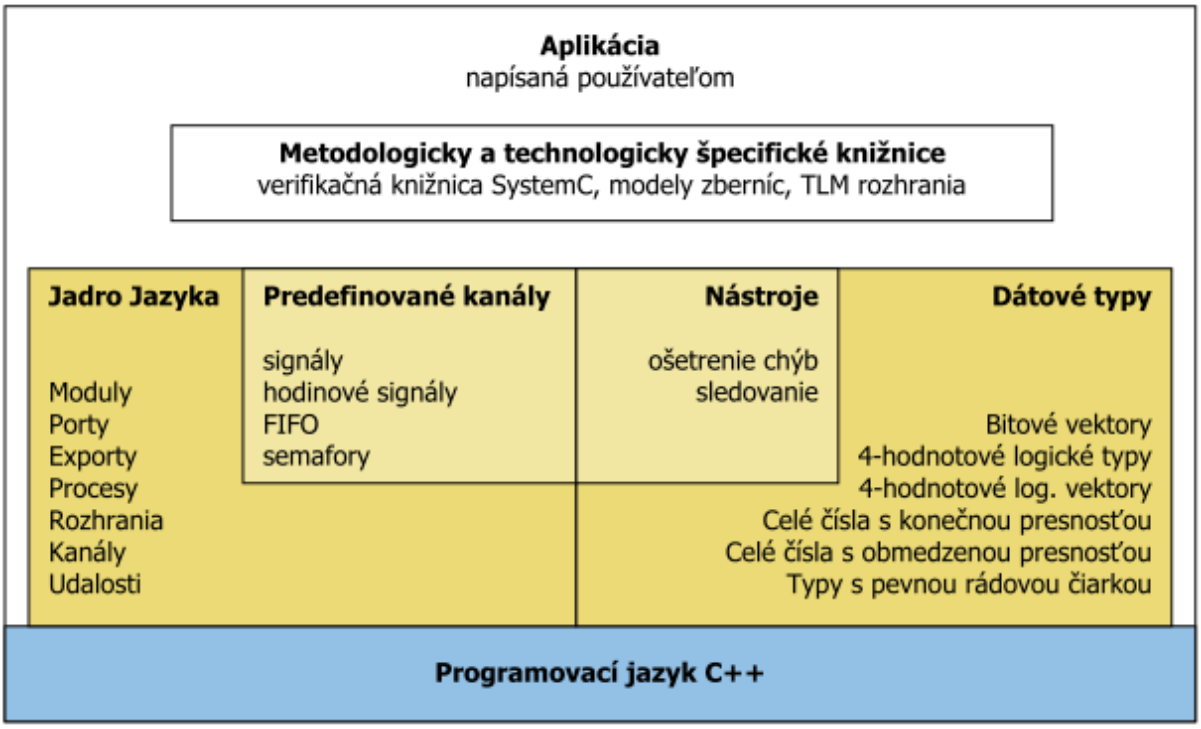
SystemC je definovaný ako IEEE 1666™ - 2005 štandard (6), ktorý považujeme za smerodajný pre akúkoľvek ďalšiu prácu. Na základe tohto štandardu uvádzame nasledujúcu charakteristiku:

SystemC pomocou tried jazyka C++ zabezpečuje:

- hierarchickú dekompozíciu systému na modely
- štrukturálnu konektivitu medzi modulmi pomocou portov a exportov
- plánovanie a synchronizáciu procesov pomocou udalostí a citlivosti procesu
- oddelenie výpočtov (procesov) od komunikácie (kanálov)
- nezávislé zjemňovanie výpočtov a komunikácie
- hardvérovo orientované dátové typy

SystemC umožňuje používateľovi napísať sadu vzájomne komunikujúcich funkcií (procesov), ktorých vykonávanie je riadené plánovačom tak, aby vznikol model systému. Procesy sú súčasťou hierarchie modelov, ktorá vyjadruje štruktúru a prepojenie v systéme. Modelom systému je teda aplikácia simulujúca správanie navrhovaného systému.

4.1 Architektúra SystemC aplikácie



Obrázok 17. Architektúra SystemC

Na obrázku je zobrazená architektúra SystemC aplikácie. Žltou farbou je vyznačená časť reprezentujúca knižnicu SystemC. A tá je postavená na štandardných knižniciach C++. Triedy knižnice SystemC zaraďujeme do štyroch kategórií:

- jadro jazyka
- dátové typy SystemC
- preddefinované kanály
- nástroje

Pričom jadro obsahuje prvky potrebné pre modelovanie digitálnych systémov. Zvyčajne sa používa spolu z dátovými typmi. Ďalšou dôležitou časťou SystemC je simulačný mechanizmus, ktorý obsahuje plánovač procesov, ktoré sa vykonávajú v reakcií na udalosti.

4.2 Modelovacie schopnosti

Základným štrukturálnym prvkom SystemC je modul. Modul môže obsahovať:

- porty
- exporty
- kanály
- procesy
- udalosti
- inštancie ďalších modulov
- ďalšie údaje a funkcie

Moduly, porty, exporty, kanály, rozhrania, udalosti a časovanie sú implementované ako triedy C++ odvodené od spoločnej základnej triedy *sc_object*. Inštancie týchto C++ tried sú teda SystemC objekty.

Procesy vykonávajú logické a aritmetické výpočty a vnášajú do modelu funkcionalitu systému. Poznáme tri typy procesov:

- **metóda** (*sc_method*) – sú určené na opísanie jednoduchých operácií, ktoré sa celé vykonajú po spustení
- **vlákno** (*sc_thread*) a **časové vlákno** (*sc_cthread*) – sú určené pre zložitejšie algoritmy a na rozdiel od metódy, vykonávanie vlákna môžeme zastaviť a znovu spustiť.

Citlivosť procesov označuje skupina udalostí, ktoré spôsobia vykonanie procesu plánovača. Vykonávanie paralelných procesov plánovačom je realizované sekvenčne, takže paralelizmus je simulovaný.

Kanály zabezpečujú komunikáciu medzi modulmi navzájom alebo medzi procesmi vnútri modulu. Rozhrania slúžia na sprístupnenie kanálov. Moduly medzi sebou komunikujú pomocou kanálov, ku ktorým pristupujú pomocou rozhraní. Jeden kanál je možne sprístupniť pomocou viacerých rozhraní a naopak. Implementáciou rozhrania je buď port alebo export.

Porty a **exporty** zabezpečujú prepojenie medzi modulmi na tej istej úrovni ako aj prepojenie medzi susednými úrovňami. Pri spracovaní sa vykoná mapovanie kanálov na príslušné porty.

Kanály delíme na hierarchické a primitívne. Hierarchický kanál je modulom a môže obsahovať procesy, inštancie iných modulov, porty alebo ďalšie kanály. Pričom primitívny kanál odvodený od špecifických tried *sc_prim_channel* nie je modulom a nemôže obsahovať žiadne ďalšie komponenty.

Od triedy *sc_prim_channel* sú odvodené ďalšie špecifické typy kanálov, ktoré sa niekedy označujú aj ako signály. Zástupcami kanálov sú:

- *sc_signal* – abstrakcia vodiča elektrického impulzu
- *sc_clock* – prenáša hodinový signál
- *sc_signal_resolved* – umožňuje pripojenie viacerých vstupov, v prípade konfliktu sa hodnota signálu určí podľa predpisu

4.3 Dátové typy

SystemC definuje skupinu dátových typov, ktoré uľahčujú modelovanie digitálnych systémov na rôznych úrovniach. Medzi významné patria:

- *sc_logic* – štvorstavová logika (logická 0 a logická 1, je stav vysokej impedancie a neznáma hodnota)
- *sc_lv<>* - vektor hodnôt v štvorstavovej logike – napríklad na modelovanie záchytného registra na zbernici
- *sc_bv<>* - vektor binárnych hodnôt – iba logická 0 a logická 1

Z uvedených komponentov dokážeme zostaviť kompletný model, ktorý vieme simulovať.

4.4 Simulácia

Zodpovedá vykonaniu aplikácie SystemC. Začína spracovaním opisu (*elaboration*), počas ktorého je zostavený model systému. Nasleduje vlastná simulácia (*simulation*) tohto modelu riadená plánovačom. V rámci celej postupnosti simulácie vieme identifikovať tieto po sebe idúce fázy:

- zostavenie hierarchie modulov
- vykonanie operácií pred koncom spracovania (*before_end_of_elaboration*)
- vykonanie operácií po spracovaní opisu (*end_of_elaboration*)
- odštartovanie simulácie (*start_of_simulation*)
- inicializácia
- simulácia
- ukončenie simulácie
- odstránenie simulovaného modelu

Prvé tri fázy sú obsiahnuté v **spracovaní opisu**. Základným účelom je zostavenie hierarchie modulov, portov, kanálov a ostatných štrukturálnych prvkov. Ďalej sú vytvorené inštancie týchto objektov spolu s prislúchajúcimi procesmi. Zároveň prebieha pripájanie portov ku kanálom. Poradie jednotlivých operácií sa môže meniť v rámci spracovania, musí sa však vykonať pred volaním *before_end_of_elaboration.* Pred ukončením spracovania sa nastaví časové rozlíšenie (veľkosť

časovej jednotky). Takto vytvorená štruktúra sa po ukončení spracovania opisu nemení a nie je možné ju dodatočne upravovať ani meniť veľkosť časovej jednotky.

Ďalej nasleduje **implementácia**, počas ktorej sa model pripraví na simuláciu. Určia sa všetky spustiteľné procesy a tie sa pridajú do množiny spustiteľných procesov. Spomedzi týchto vybraných sa odstránia tie ktorých inicializácia bola zakázaná volaním funkcie *dont_initialize*.

Simulácia pozostáva z troch krokov, ktoré sa v réžii plánovača opakujú až do jej ukončenia:

1. **Výpočtová fáza** – postupne sa vykonávajú všetky procesy z množiny vykonateľných. Proces beží až do jeho ukončenia alebo po volanie funkcie čakania – *wait*. Pri ďalšom spustení procesu sa pokračuje od tohto miesta.
2. **Aktualizácia** – výsledky vypočítané v procesoch sa propagujú do štruktúry. Nastavujú sa premenné, zapisujú sa hodnoty do kanálov.
3. **Notifikácia** – porovnaním citlivosti procesov a aktuálneho stavu modelu sa určí, či majú byť jednotlivé procesy vykonávané v ďalšom cykle. Tieto sa buď pridajú alebo odstránia z množiny vykonateľných procesov. Okrem toho sa odstránia čakajúce procesy a pridajú sa tie, ktorým doba čakania nastavená funkciou *wait* uplynula.

Po vykonaní používateľom zadaného počtu cyklov, prípadne prerušení nekonečného cyklu sa formálne **ukončí simulácia** a **zlikviduje model**. Odstránia sa všetky údaje používané simulátorom a zavolajú deštruktory všetkých objektov, vytvorených pri spracovaní opisu.

4.5 Analýza diplomovej práce Vizualizácia simulácie SystemC modelu

4.5.1 Riešenie projektu

V riešení projektu sú riešené jednotlivé časti projektu. Prvá kapitola je špecifikácia požiadaviek, nasleduje návrh riešenia, jeho implementácia a overenie. V riešení vychádzame z poznatkov získaných analýzou.

4.5.2 Špecifikácia požiadaviek

V špecifikácií požiadaviek si určíme charakteristické vlastnosti budúceho softvérového systému. Pri ich definovaní vychádzame zo zadania. Obsahujú vhodné vlastnosti existujúcich riešení. Jednotlivé požiadavky sú zaradené medzi funkcionálne požiadavky, požiadavky na výsledok a požiadavky na rozhranie.

Tabuľka 1. Požiadavky na softvérové systémy

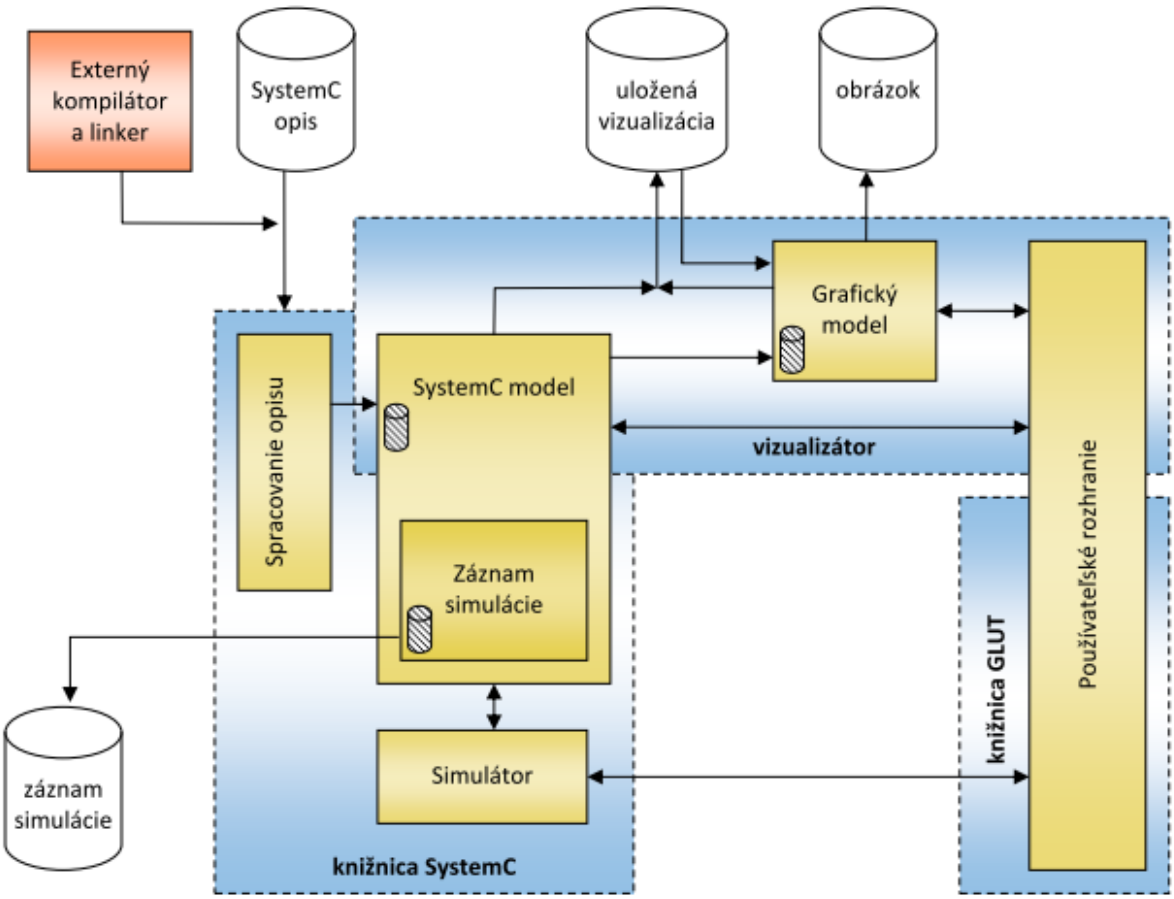
Funkcionálne požiadavky	Požiadavky na výsledok	Požiadavky na rozhranie
Zostavenie modelu správania	Založený na existujúcom produkte pri zachovaní funkcionality	Tradičné používateľské rozhranie
Zobrazenie výsledkov pre celú štruktúru aj jej komponenty	Spolupráca s existujúcimi vývojovými nástrojmi	Intuitívna obsluha
Simulácia kombinačných a sekvenčných obvodov	Použitie v prostredí Microsoft Windows	Aktívna časová os
Simulácia v súlade so štandardom	Bezplatné šírenie vykonateľnej verzie pre nekomerčné účely	Prehľadné zobrazenie hodnôt
Riadenie simulácie		
Zobrazenie hodnôt pre zvolený diskrétny čas		
Export výsledkov		

4.5.3 Návrh riešenia

Návrh riešenie sa usilujeme navrhnúť tak, aby čo najpresnejšie splnil požiadavky a zároveň bol produkt dokončený v stanovenom čase. Používame už existujúci produkt SystemC+Visualizer, ktorý navrhujeme upraviť a doplniť o štruktúru pre záznam výsledkov simulácie. Ďalej navrhujeme spôsob zobrazenia výsledkov simulácie , niektoré ďalšie funkcie a úpravu používateľského rozhrania.

4.5.3.1 Koncepcia a architektúra

Keďže existujú overené a voľne dostupné simulátory neriešime vytváranie a tvorenie nového simulátora. Použijeme vstavaný simulátor SystemC knižnice, ktorý spĺňa požiadavky simulácie kombinačných aj sekvenčných obvodov a to v súlade so štandardom, nakoľko simulátor OSCI (7) je praktický jeho implementáciou. Ako základ pre nový produkt si zvolíme SystemC+Visualizer, ktorý je vytvorený na základe knižnice SystemC.



Obrázok 18. Architektúra nového softvérového produktu

Pri rozširovaní SystemC+Visualizer vychádzame z pôvodnej koncepcie statickej knižnice, ktorá sa s používateľským SystemC opisom prepojí pomocou externého kompilátora a linkera. Vytvorená aplikácia pozostáva s pôvodnej knižnice SystemC s OSCI simulátorom, vizualizátora a používateľského rozhrania. Pôvodnú aplikáciu zásadne nemeníme, časti programu optimalizujeme a dopĺňame niektoré v minulosti neimplementované funkcie.

Keďže v predchádzajúcej verzii sa simulátor nepoužíval, musíme ho aktivovať a čiastočne modifikovať, aby dokázal spolupracovať s vizualizátorom. Výstupy simulácie nebudeme prezentovať v reálnom čase, zaznamenáme ich a zobrazíme až po ukončení simulácie. Na tento účel navrhujeme dátovú štruktúru pre záznam simulácie vrátane mechanizmov, ktoré zabezpečia jej naplnenie, čítanie podľa požiadavky používateľa a prezentáciu. V neposlednom rade potrebujeme vedieť riadiť simuláciu z používateľského rozhrania. Tiež navrhujeme niekoľko doplnkových funkcií.

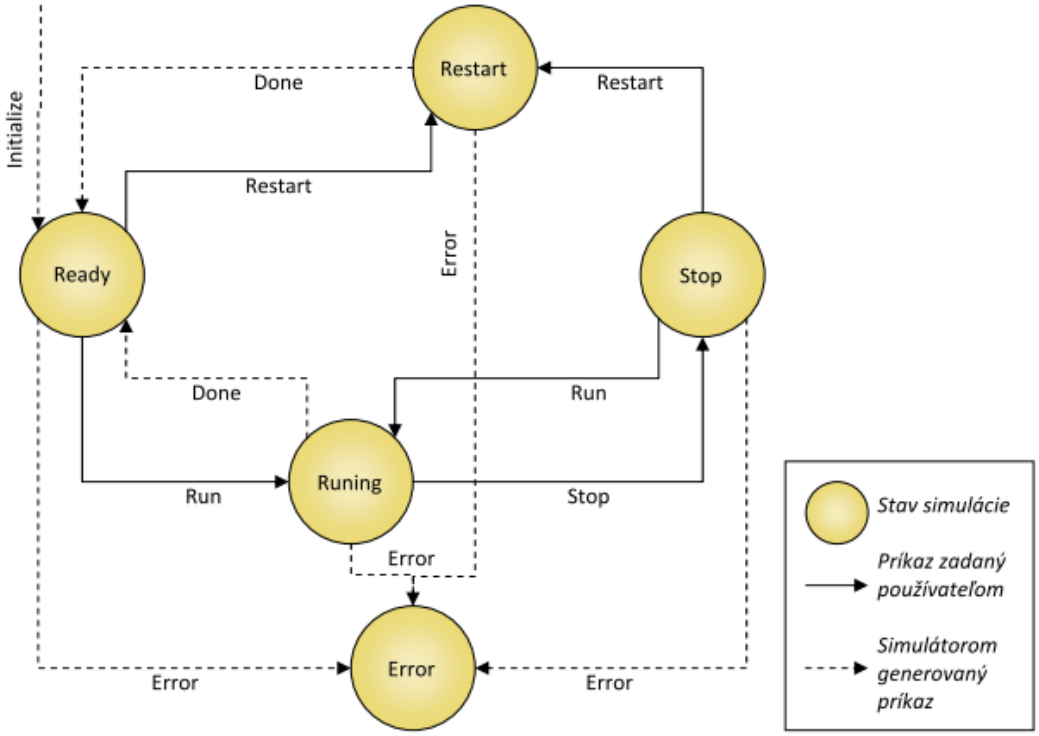
4.5.3.2 Riadená simulácia

Pôvodný SystemC+Visualizer umožňuje vykonať buď simulátor alebo vizualizátor, bez možnosti prepínať medzi nimi počas behu programu. Aby sme dokázali model simulovať a súčasne v okne vizualizátora prezentovať výsledky, vytvoríme dve vetvy programu, ktoré spustíme paralelne.

Prvá vetva simulátor ako rodičovský proces, ktorý je pri nečinnosti uspatý a druhá vetva vizualizátor spustený vo vlákne, ktoré vzniká pri spustení aplikácie a zaniká až pri ukončení programu.

Simulátor je potrebné upraviť aby dokázal spolupracovať s vizualizátorom. Zmeny sú nasledujúce:

1. Pri chybe simulácie sa ukončí iba táto a nie celý program.
2. Chybové hlásenia sa okrem príkazového riadku zobrazia aj v grafickom rozhraní.
3. Na konci každého kroku simulácie sa vynúti prekreslenie používateľského rozhrania. Predpokladáme, že každý krok môže zmeniť hodnoty na portoch či signáloch a túto zmenu je potrebné propagovať používateľovi.
4. Na konci každého kroku simulácie sa prečítajú údaje zo všetkých objektov a tieto sa uložia do štruktúry záznam simulácie, ktorá je definovaná nižšie. Tieto údaje sú neskôr podľa požiadaviek používateľa čítané a zobrazené.
5. Simulátor je riadený príkazmi používateľa. Postačujúce sú „Spustiť/Pokračovať," „Zastaviť" a „Reštartovať." Tieto priamo nespôsobia spustenie či zastavenie simulátora, ale iba zmenia príznak „stav simulácie". Simulátor tento príznak opakovane kontroluje a na základe jeho hodnoty rozhoduje o svojej ďalšej činnosti.



Obrázok 19. Riadenie simulácie prostredníctvom príkazov

Po spustení programu je simulátor pripravený, príznak stavu sa inicializuje na hodnotu *Ready*. Simulátor sa spustí príkazom *Run*, začne sa vykonávať vlastná simulácia. Používateľ ju môže kedykoľvek prerušiť príkazom Stop a následne v nej pokračovať ďalším príkazom *Run*. Po uplynutí simulačného času je automaticky vygenerovaný príkaz *Done*. Pokiaľ simulácia neprebieha môžeme ju reštartovať, čím simulátor vrátime do východiskového stavu pričom musíme vyprázdniť záznam simulácie. Pri vzniku chybovej udalosti sa nastavuje stav *Error*, ktorý nebráni používaniu aplikácie avšak znemožňuje ďalšiu simuláciu.

4.5.3.3 Zobrazenie výsledkov simulácie

Hodnoty na kanáloch reprezentujeme pomocou rôznych farieb v doplnení o alfanumerický výpis hodnoty. Kanály, ktoré nesú iba trojstavovú logiku sú rozlíšené farebne, doplnené o alfanumerické vyjadrenie hodnoty. Priradenie farieb hodnotám udáva tabuľka 2. Hodnoty na ostatných sú reprezentované alfanumericky, uvedené ako súčasť menovky. Ekvivalente sú reprezentované hodnoty na portoch, pričom farby sa aplikujú na symbol portu. Ďalej navrhujeme implementovať funkcie pre vypnutie alebo zapnutie farebného zvýraznenia, vypnutie alebo zapnutie alfanumerického výpisu hodnôt a skrytie menoviek objektov.

Tabuľka 2. Priradenie farieb simulácie

Hodnota	Farba
Bez hodnoty (simulácia vypnutá)	Čierna
Stav vysokej impedancie	Modrá
Logická 0	Zelená
Logická 1	Svetlá oranžová
Korektná alfanumerická hodnota	Tmavá zelená
Konflikt	Červená

4.5.3.4 Ďalšie funkcie

Uloženie a načítane modelu

Pre úspešné načítanie vzhľadu modelu potrebujeme model vlastný (používateľský SystemC opis) a súčasne súbor so vzhľadom modelu, do ktorého ukladáme:

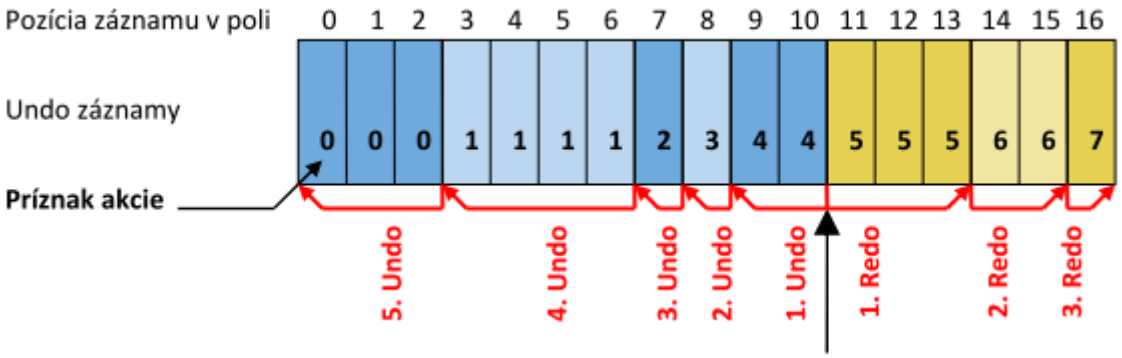
- **Identifikátor verzie súboru** – v budúcnosti umožní odlíšiť rôzne verzie súborov so vzhľadom modelu.
- **Označenie objektu** – umožňuje priradiť uložené údaje k objektom modelu. Používame úplný názov objektu, ktorý je v rámci modelu vždy unikátny.
- **Údaje o vzhľade objektu** – sú všetky parametre týkajúce sa vzhľadu a umiestnenia objektu tak, ako boli definované v predchádzajúcej verzii SystemC+Visualizer. (8)
- **Riadiace údaje** – sú potrebné pre správne načítanie súboru. Jedná sa o veľkosti polí, dĺžky spájaných zoznamov a príznaky prázdnych reťazcov.

Viacnásobný výber

Pôvodný program dovoľuje pracovať iba s jedným objektom, čo je nepraktické. Preto všetkým objektom dopĺňame príznak, ktorý určuje či je objekt vybraný alebo nie. Operácie ako rotácia, presun a iné sa vykonajú na všetkých objektoch, ktoré majú tento príznak nastavený.

Vrátenie a opakovanie akcie

Vrátenie akcie je ako „Undo“ v pôvodnom programe implementované poľom záznamov kde každý záznam obsahuje informácie o modifikovanom objekte a vykonanej operácii. V tejto funkcií musíme zabezpečiť, aby sa pri kroku späť spracovali záznamy naraz a nie jednotlivo. Vyriešiť sa to dá pridaním funkcie akcie, ktorý pre všetky záznamy jednej akcie rovnaký. Pri kroku spať sa spracujú všetky záznamy s rovnakým príznakom, tak ako vidíme na obrázku. Opakovanie akcie „Redo“ je inverzné pole k operácií Undo.



Obrázok 20. Čítanie Undo záznamov pri vrátení a opakovaní akcie

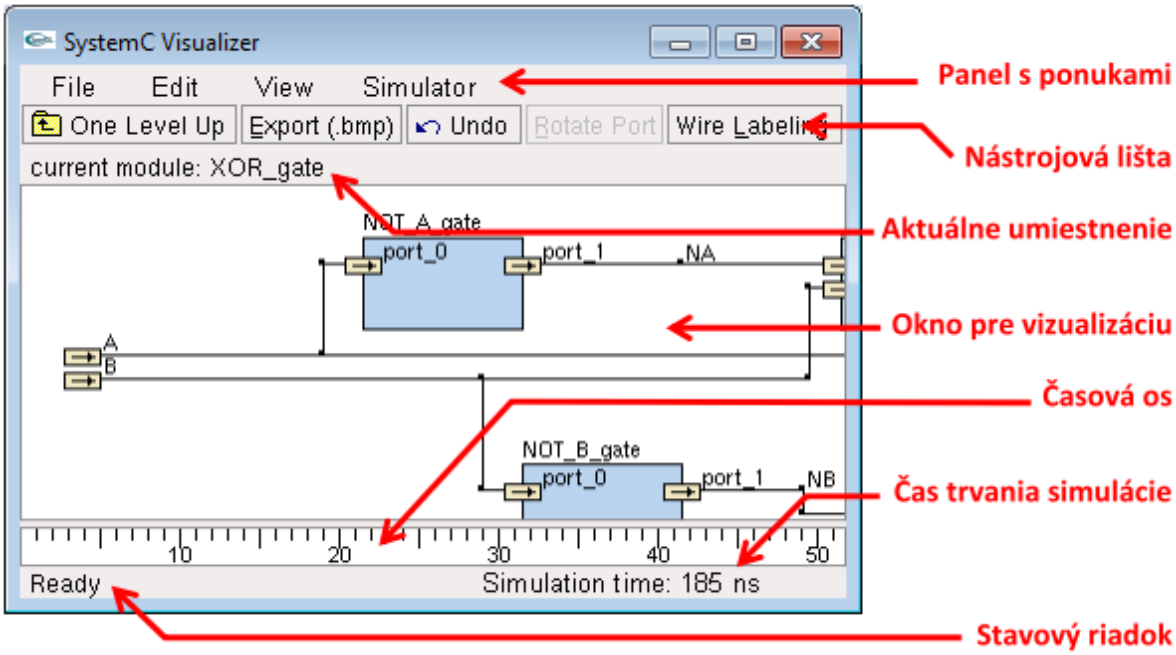
Export výsledkov simulácie

SystemC už obsahuje algoritmus pre export výsledkov simulácie do VCD súboru, ktorý je určení pre uloženie výsledkov simulácie. Jeho vhodnou modifikáciou možno vytvoriť vlastný algoritmus pre export. Budú sa doňho ukladať iba zmeny hodnôt.

4.5.3.5 Používateľské rozhranie

Pôvodné rozhranie vyhovuje požiadavkám preto nebude zásadne menené. V dôsledku pridania novej funkcionality a s ohľadom na jednoduchosť používania navrhujeme:

- Pridať **časovú os** na spodný okraj okna aplikácie. Zobrazuje celé trvanie simulácie alebo jeho časť. Zároveň umožňuje jednoduchú navigáciu v simulačnom čase. Používateľ kliknutím na požadovaný okamih zvolí čas, pre ktorý chce poznať hodnoty na portoch a vodičoch. Aktuálne zvolený čas je označený kurzorom.
- Pridať **Panel s ponukami – menu:**
  - File (Súbor) – všetky súborové operácie
  - Edit (Úprava) nástroje na úpravu modulov, portov a vodičov
  - View (Zobrazenie) – príkazy týkajúce sa spôsobu zobrazenie výsledkov simulácie , modelu a časovej osi
  - Simulator (Simulátor) – riadenie simulácie a ďalšie súvisiace príkazy
- Upraviť nástrojovú lištu, tak aby obsahovala len najčastejšie používané funkcie.
- Do prvého dolného rohu umiestniť údaj o trvaní simulácie a jednotke času.
- Klávesy A, S, D, W pre posúvanie objektov a plátna zmeniť za kurzorové šípky.
- Pridať dialóg pre zadanie mena súboru pre export, uloženie alebo načítanie. (9)



Obrázok 21. Návrh používateľského rozhrania na základe pôvodného dizajnu

5 Dostupné riešenia vizualizácie modelov digitálnych systémov

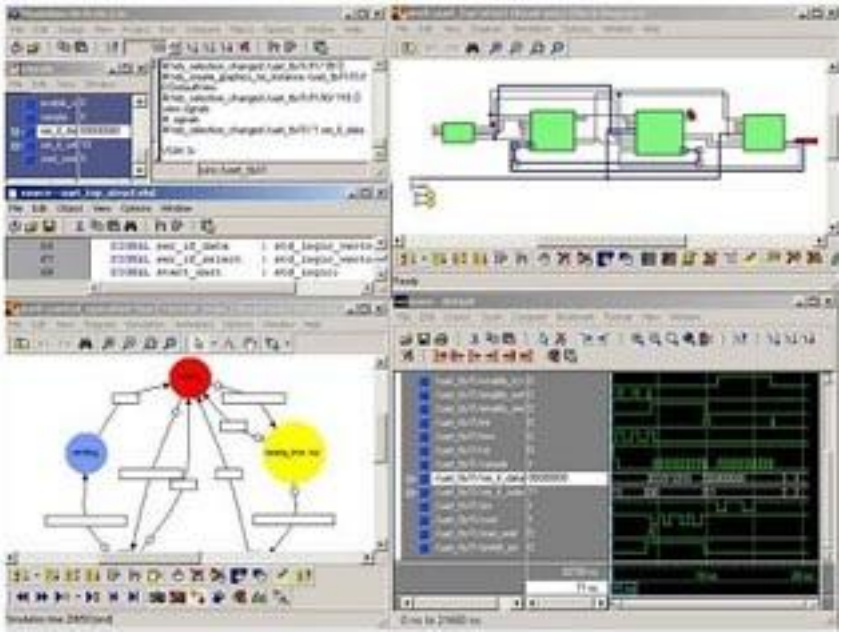
V našej práci sa venujeme trom momentálne najznámejším HDL jazykom a to VHDL, Verilog a SystemC. V tejto časti dokumentu spíšem dostupné aplikácie a riešenia, pokúsim sa ich objasniť a poukázať na ich pozitíva resp. nedostatky. Čo sa týka nedostatkov, môžeme vypichnúť hlavne neuspokojivo pokrytú oblasť vizualizácie simulácie modelov digitálnych systémov a teda ako daná konkrétna aplikácia túto oblasť pokrýva. Tento všeobecne známi nedostatok riešili viacerý študenti našej fakulty ich práce síce spomeniem, ale podrobný opis týchto nástrojov bude v inej časti dokumentu. Táto časť dokumentu je rozdelená na dve časti. Ako prvé prejdem komerčnými riešeniami vizualizácie modelov digitálnych systémov. V tejto časti zhromaždím všetky dôležité a zaujímavé informácie o dostupných riešeniach v komerčnej sfére. V nasledujúcej časti budú opísané riešenia, ktoré sú dostupnejšie a to hlavne nástroje ktoré sú vyvíjané pod licenciou GPL. Postupne prejdem štruktúrou týchto riešení, princípmi ich fungovania a vlastnosťami ktorými sa odlišujú od iných nástrojov.

5.1 Dostupné riešenia v komerčnej sfére

Silným hráčom na trhu, čo sa týka VHDL vizualizácie, je firma Mentor Grapihcs. Táto firma vyvíja viacero aplikácii, ktoré sa zaoberajú vizualizáciou simulácie resp. štruktúry modelov digitálnych systémov v jazyku VHDL. Za podstatnú nevýhodu týchto riešení možno považovať ich nedostupnosť, keďže sa jedná o komerčné riešenia a ich cena nie je pre niektoré skupiny ľudí ako napr. študentov prijateľná. Samozrejme že Mentor Graphics nie je jedinou spoločnosťou zaoberajúcou sa VHDL vizualizáciou. Existuje niekoľko ďalších firiem zaoberajúcich sa toto problematikou. V tejto časti práce sa pokúsim zhromaždiť čo najviac nástrojov zaoberajúcich sa problematikou vizualizácie modelov digitálnych systémov. Každý jeden bude popísaný a to hlavne jeho zaujímavé funkcie a odlišnosť od iných riešení. Taktiež budú niektoré vlastnosti resp. funkcie daných riešení demonštrované aj na obrázkoch. Keďže najviac nástrojov v tejto sfére pochádza od firmy Mentor Graphics začnem týmito riešeniami danej problematiky.

5.1.1 HDL Author

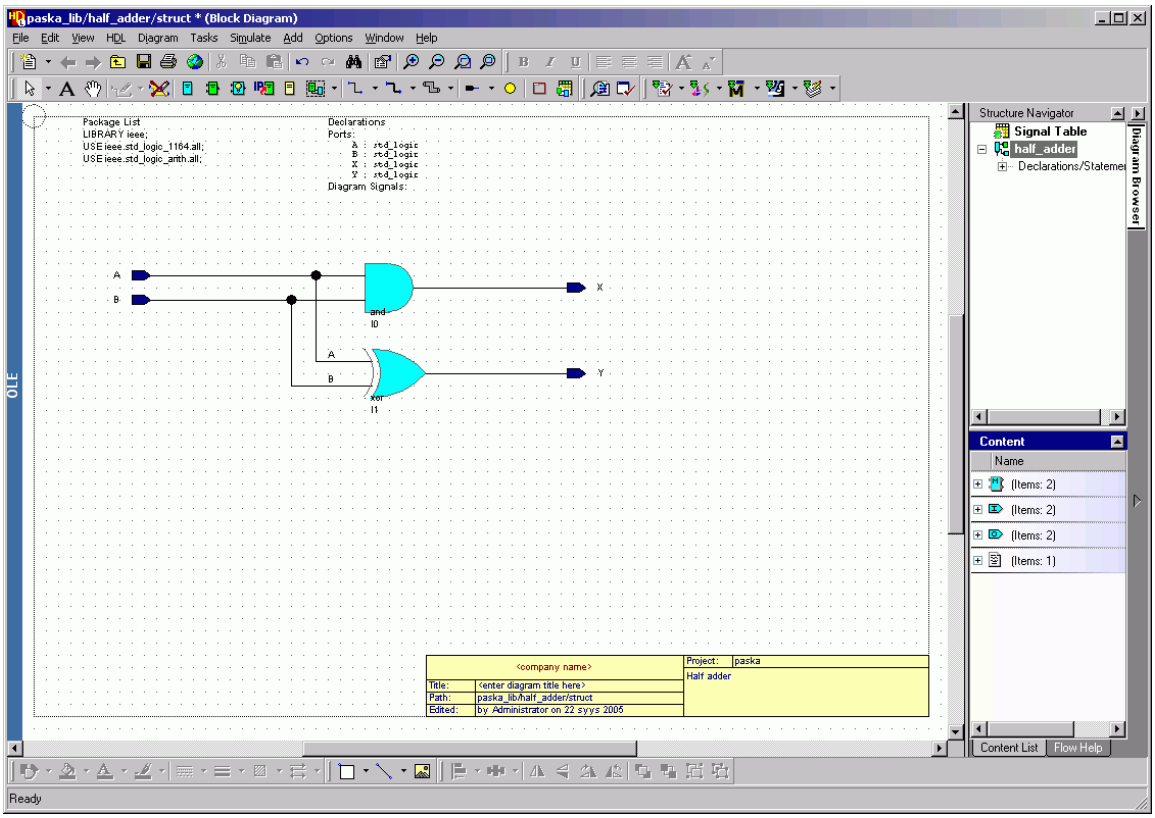
Jedným z prvých produktov od spoločnosti Mentor Graphics je HDL Author (2). Ako je uvedené na webovej stránke firmy Mentor Graphics, toto riešenie je vhodné na prácu v týme, pretože umožňuje správu návrhových dát a tokov návrhu. Touto vlastnosťou by sa mala zvýšiť produkcia každého jedného člena týmu a teda vo finále aj produkcia celého týmu. Ďalšia veľmi významná výhoda je podpora pokročilých editorov. Tiež je možná vizualizácia HDL kódu, či už v textovej, tabuľkovej alebo grafickej podobe. Ďalším plusom je integrovanie iných nástrojov, ktoré môžu byť ľahko spustené priamo z prostredia HDL Author. Ako veľkú nevýhodu však môžeme vnímať, tak ako u väčšiny produktov od firmy Mentor Graphics, cenu tejto aplikácie. Čo sa týka histórie je HDL Author nástupcom nástroja HDL Detective, ktorý v dnešnej dobe už nie je ďalej podporovaný a vyvíjaný. Ako už bolo uvedené HDL Author je nástroj na vizualizáciu HDL kódu a medzi podporované jazyky patria VHDL a Verilog.



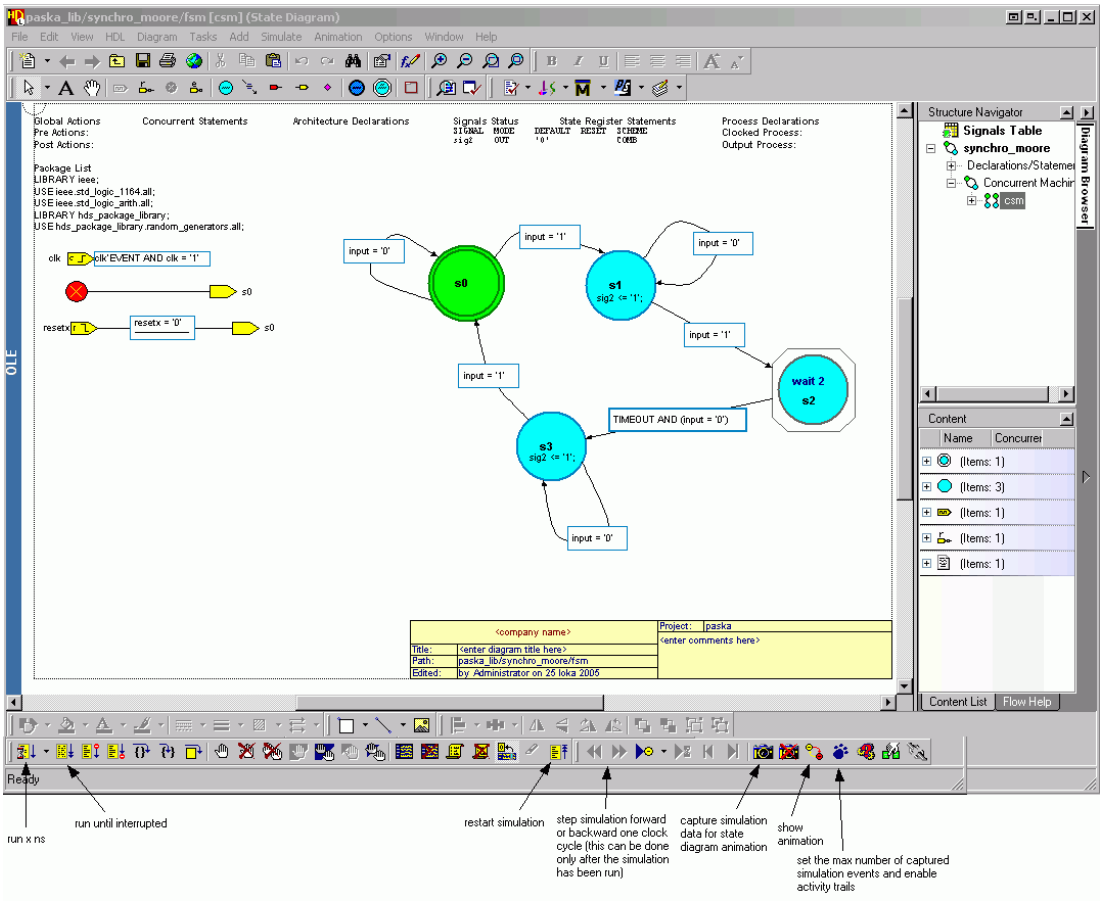
Obrázok 22. HDL Author workspace

5.1.2 HDL Designer

Ďalším nástrojom z dielne Mentor Graphics je HDL Designer (10). Tento nástroj ponúka širokú paletu podporovaných vlastností. Okrem podpory pokročilých editorov a správy návrhových dát a tokov návrhu, ktoré môžeme nájsť aj v HDL Author, tento nástroj ponúka navyše importovanie a konvertovanie opisu návrhu modelov digitálnych systémov a vytváranie rôznych náhľadov na návrh. Takisto nezabúda na prácu v týme. V tomto smere však neponúka nič viac ako jednoduchší HDL Author. Čo sa týka grafického zobrazenia, na výber sú dve možnosti. Možné je zobrazenie blokovými diagramami, ale tiež aj zobrazenie pomocou tabuliek. Výhodou je tiež, že v grafoch nie je umožnené robiť zmeny, ktoré by nejakým spôsobom mohli ovplyvniť logickú štruktúru obvodov. Užitočná je taktiež tvorba dokumentácie, ktorá je generovaná priamo v aplikácii. So zohľadnením všetkých funkcií, ktoré tento nástroj ponúka, môžeme považovať HDL Designer za jednu z najlepších aplikácii v odvetví vizualizácie modelov digitálnych systémov. Avšak jeho robustnosť sa často odráža na náročnosti práce s týmto nástrojom. Okrem toho tak ako HDL Author je táto aplikácia komerčná a teda finančne náročná. V riešené je taktiež zahrnutý aj simulátor, ktorý však nie je postačujúci a dostatočne robustný.



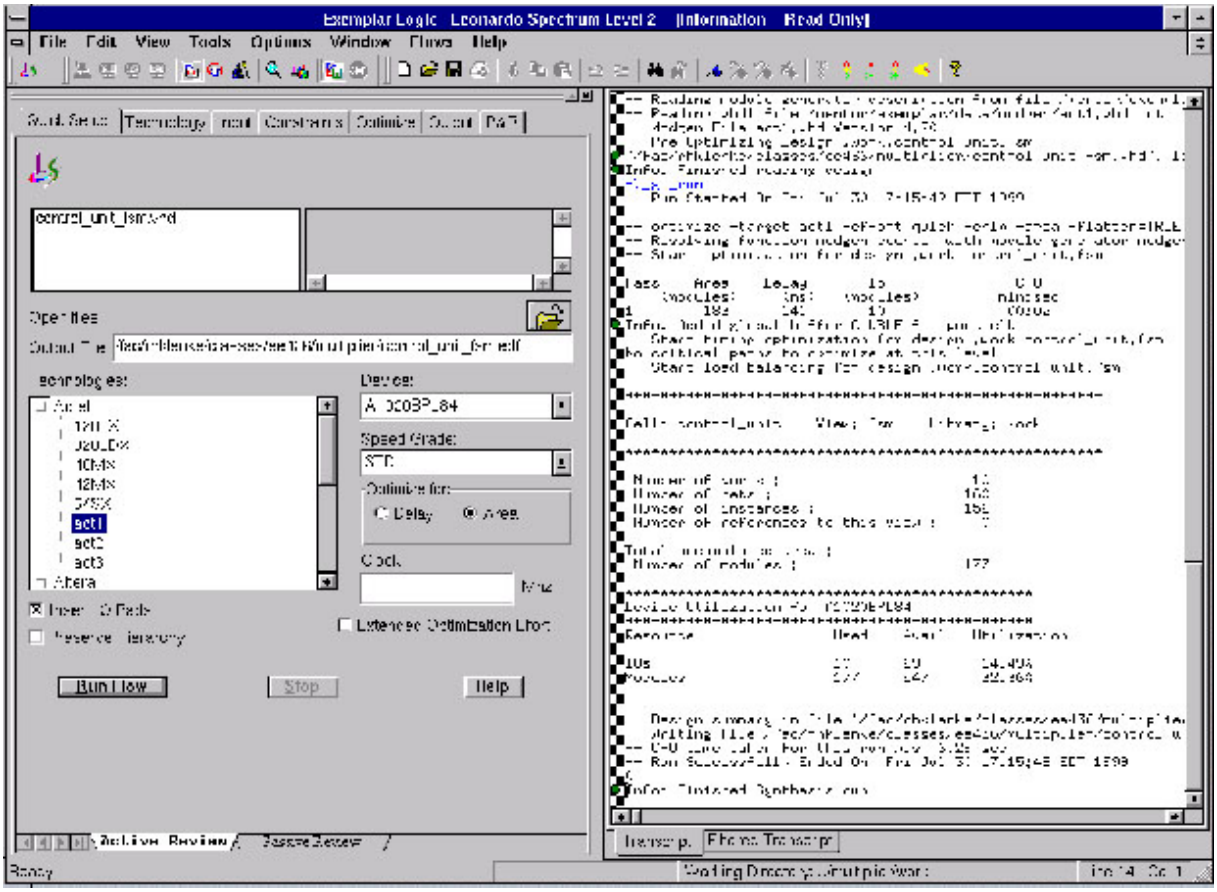
Obrázok 23. Blokový diagram v prostredí HDL Designer



Obrázok 24. Stavový diagram v prostredí HDL Designer

5.1.3 Leonardo Spectrum

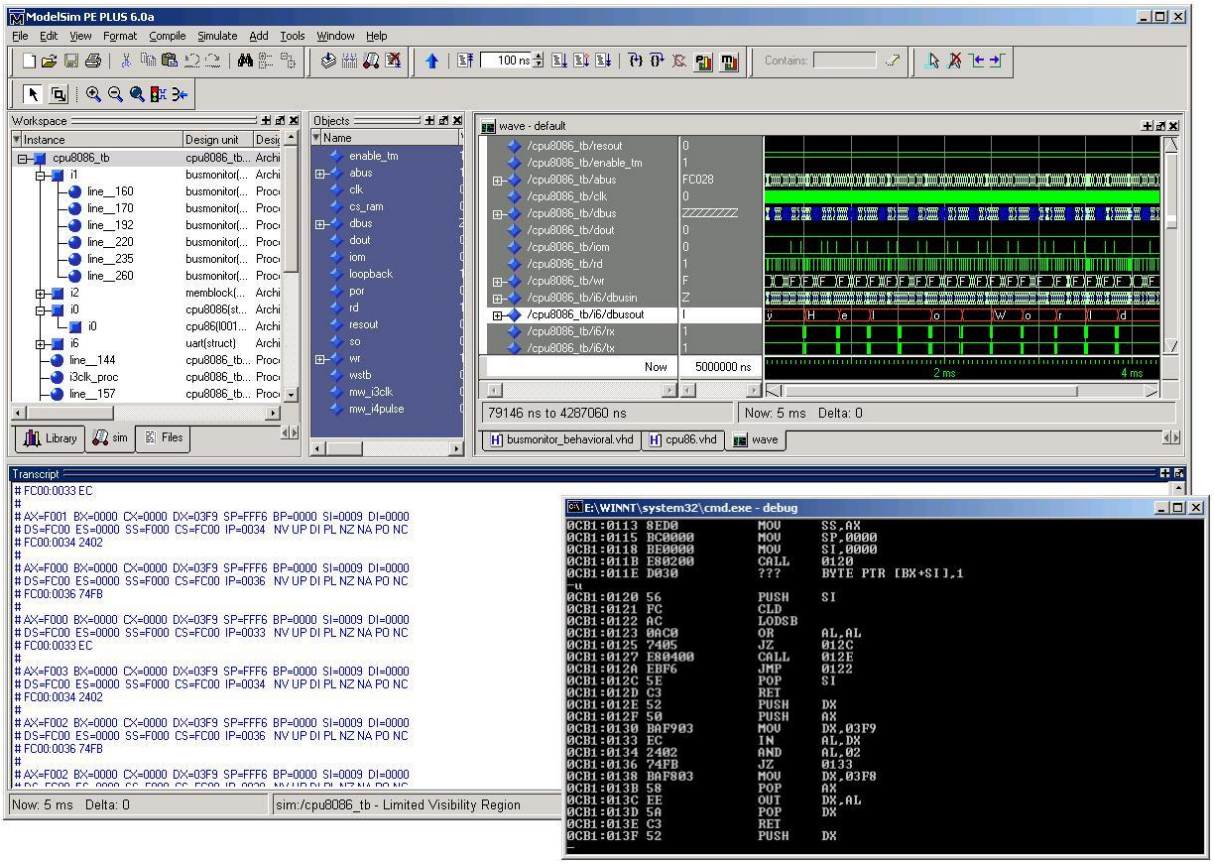
Leonardo Spectrum (11) (12) je ďalším nástrojom z dielne nami známej firmy Mentor Graphics. Svojou štruktúrou ho môžeme považovať za odľahčenú verziu nástroja HDL Designer avšak platformovo a strojovo nezávislú, čo je veľká výhoda. Oproti HDL Designeru je vďaka “odľahčeniu” užívateľsky prívetivejší, rýchlejší a cenovo trošku viac dostupnejší. Tak ako v HDL Designeri aj v tomto nástroji je možné navrhnúť HDL model či pre jazyk VHDL alebo Verilog. Tiež obsahuje niekoľko ďalších funkcií zdedených od svojho robustnejšieho “brata”. Medzi tieto funkcie môžeme zaradiť napr. Importovanie HDL modelov a ich zobrazenie v grafickej podobe alebo reprezentáciu modelov v hierarchických úrovniach.



Obrázok 25. Leonardo Spectrum workspace

5.1.4 Modelsim

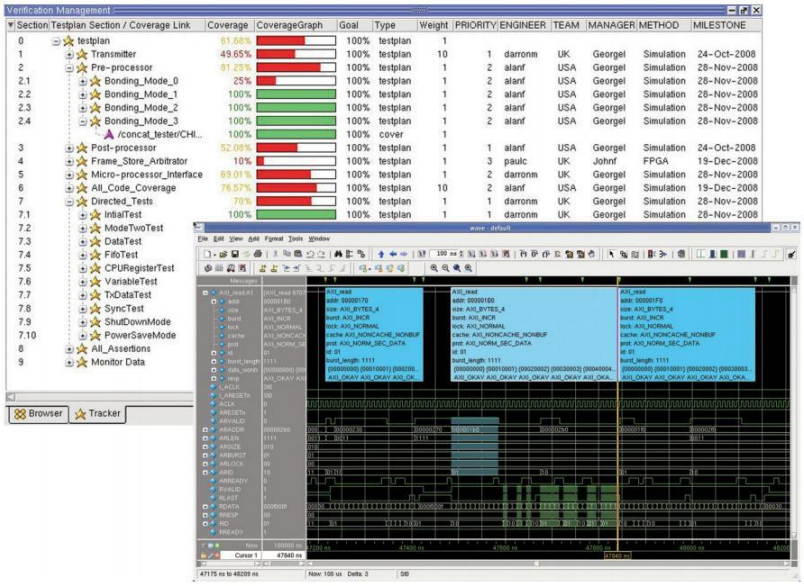
V nástroji Modelsim (12) je umožnené nielen pracovať s modelmi vytvorenými v jazykoch VHDL, Verilog alebo SystemC ale taktiež kombinovať medzi nástrojmi na verifikáciu modelov v týchto jazykoch. Ako predchádzajúce nástroje aj Modelsim je prácou spoločnosti Mentor Graphics. Tento verifikačný nástroj podporuje nielen reprezentáciu simulácie, ktorá zobrazuje časový priebeh signálov, ale taktiež je možné sledovať danú simuláciu v sche (13)matickom zápise.



Obrázok 26. Simulácia vizualizácie pomocou nástroja Modelsim

5.1.5 Visual Elite HDL

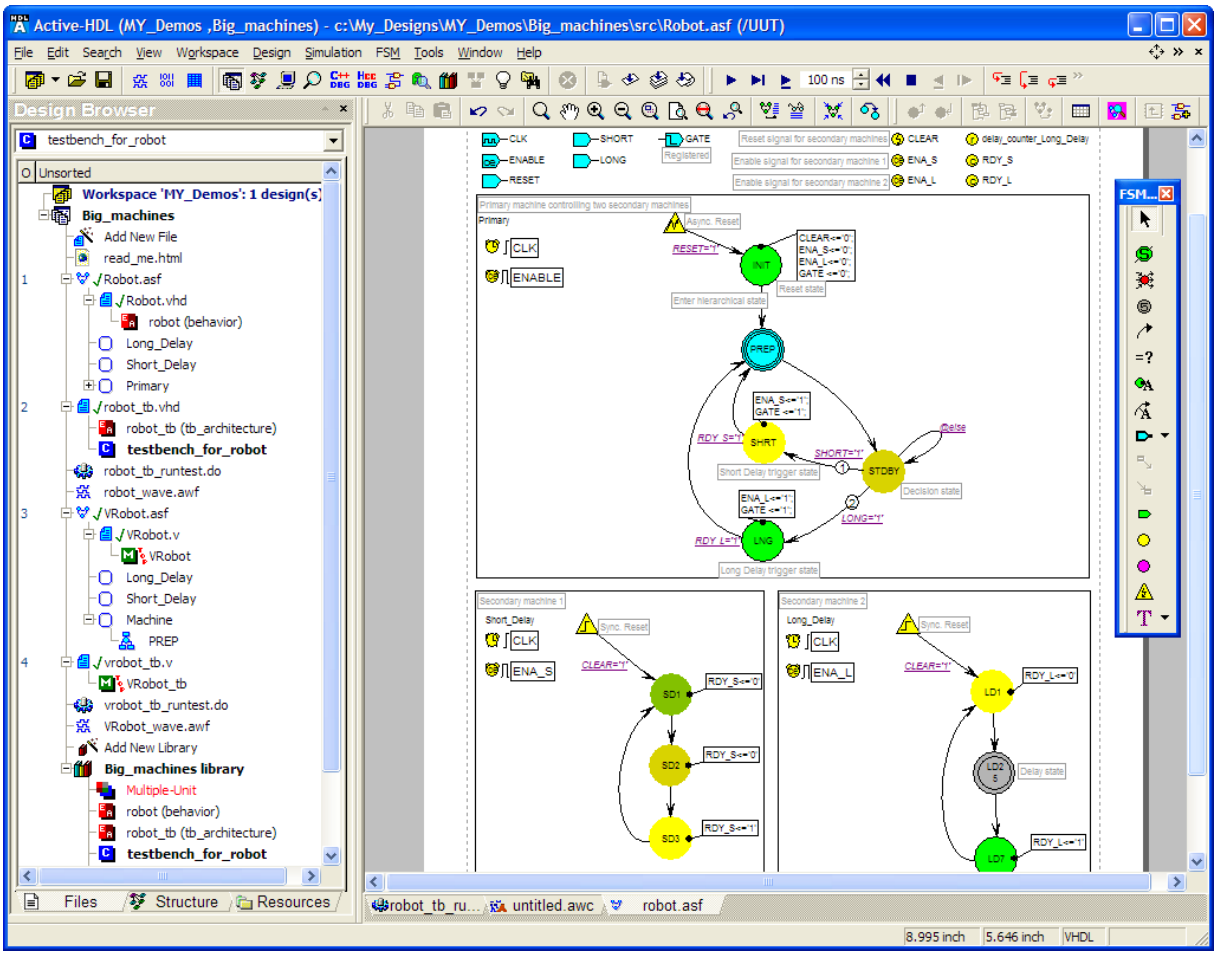
Ako posledný nástroj od Mentor Graphics si spomenieme Visual Elite HDL (13), ktorý síce nepochádza priamo z dielne tejto spoločnosti, ale potom ako ho vytvorila firma Summit Design, bol odkúpený práve firmou Mentor Graphics. Visual Elite HDL je robustný nástroj, ktorý v sebe zahŕňa množstvo funkcií. Takisto ako HDL Designer má v sebe integrované nástroje pre textový, tabuľkový resp. grafický návrh. Zaujímavosťou je, že textový opis vie previesť na grafický opis. Tento návrh môže byť následne verifikovaný pomocou simulácie. Dokáže tiež kombinovať modely vytvorené v rôznych HDL jazykoch a to v VHDL, Verilog a SystemC. Návrh možno vytvoriť v hocijakom z týchto jazykov či už zdrojovým kódom alebo taktiež podporuje možnosť vytvoriť návrh v grafickom rozhraní. Vytváranie v grafickom modely prebieha pomocou modulov, ktoré si môžeme v blokovom diagrame predstaviť ako jednotlivé bloky, a ich interakciou. Jednotlivé moduly majú na výstupe resp. vstupe porty, ktorých správanie možno definovať. Taktiež je možné zadefinovať vnútorné správanie modulu. Tento nástroj aj napriek tomu, že sa robustnosťou vyrovná HDL Designeru, ponúka veľmi prívetivé používateľské prostredie. Ako jeho najväčšiu výhodu však možno považovať možnosť kombinovania jednotlivých HDL jazykov. Ako nevýhoda je tak ako u každého nástroja od Mentor Graphics jeho cena a teda dostupnosť. (14)



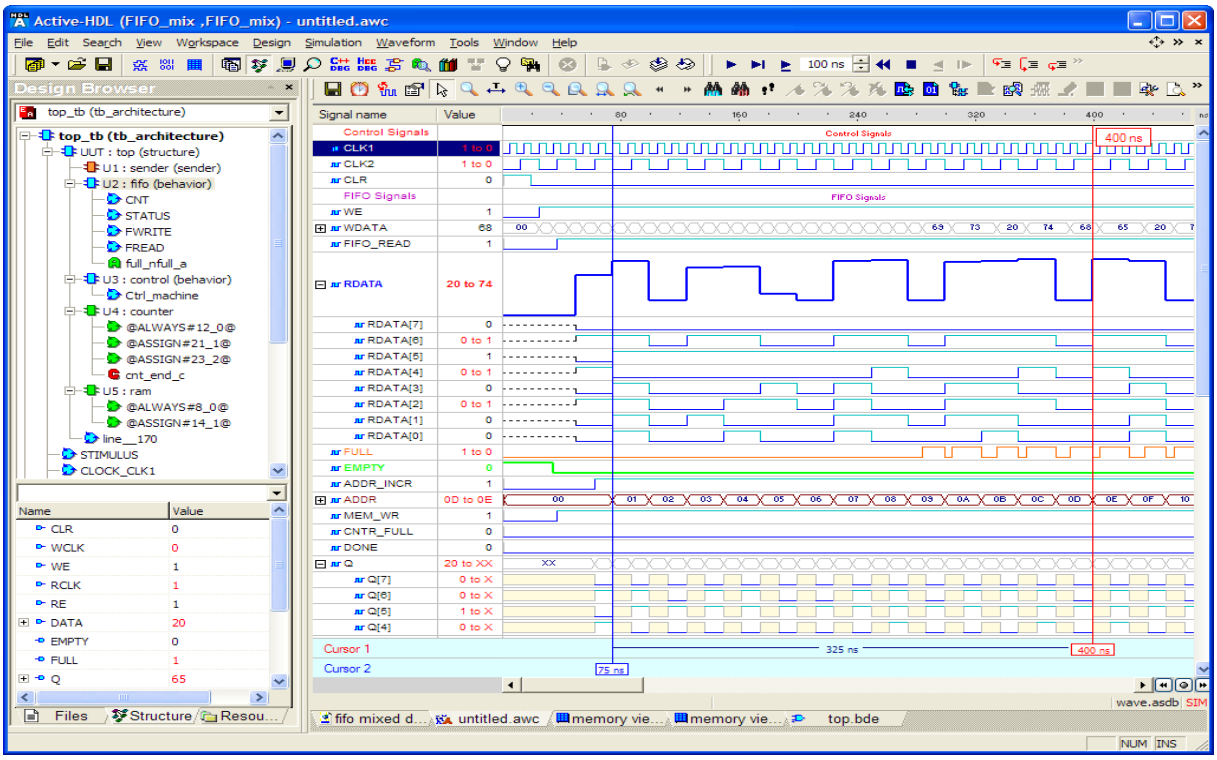
Obrázok 27. Visual Elite HDL workspace

5.1.6 Active-HDL

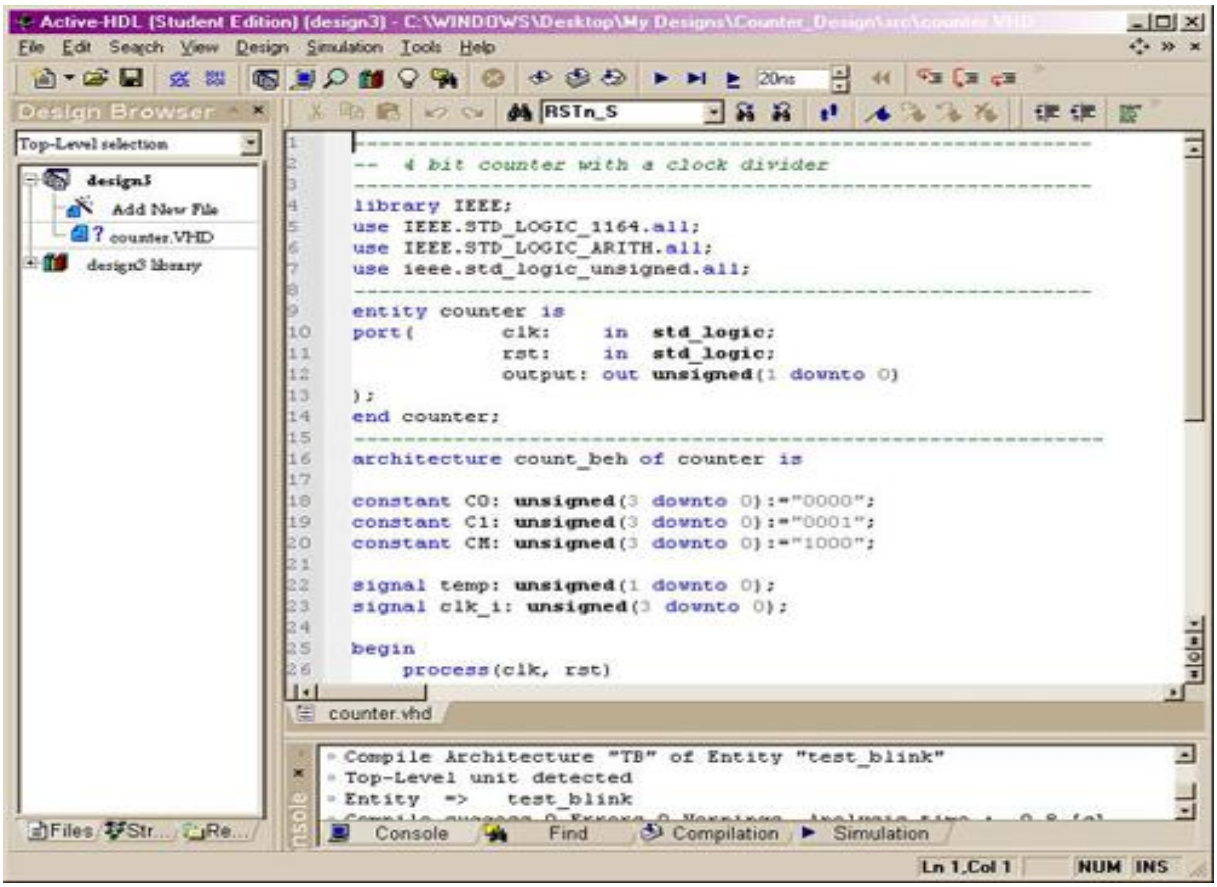
Active-HDL (14) je v našom zozname prvým zástupcom pochádzajúcim od inej spoločnosti ako Mentor Graphics a to od firmy Aldec Inc. Bohužiaľ, keďže sa jedná opäť o komerčné riešenie, opäť nie je tento nástroj voľne prístupný. Tak isto ako predchádzajúce opisované aplikácie aj Active-HDL má v sebe okrem textového editor integrovaný aj grafický editor. Ďalšou už známou funkciou je možnosť prevedenia opisu modelu na opis grafický, ale navyše podporuje aj spätnú možnosť, čiže previesť grafický opis na opis textový. Keďže podporuje viacero HDL jazykov je možnosť napr. Previesť textový opis v jazyku VHDL na grafický opis a následne previesť tento grafický opis na textový opis v jazyku Verilog. Tak isto ako Modelsim aj v Active-HDL je integrovaný simulátor na verifikovanie návrhu, ktorý takisto zobrazuje priebeh signálov v časovom interval. Taktiež tento nástroj podporuje automatický generovanú dokumentáciu, tak ako sme sa s tým stretli u nástroja HDL Designer. Tento modul je však v Active-HDL prepracovanejší a podporuje nielen viac výstupných formátov ale taktiež štruktúra dokumentu je prehľadnejšia a užívateľsky lepšie spracovaná. Vizualizácia modelov je v tomto nástroji podobná ako v riešení od Mentor Graphics Visual Elite HDL a teda je založený na reprezentácii modulov ako blokov, ktoré sú potom bližšie opísané, či už portami na vstupe resp. výstupe alebo vnútornou štruktúrou.



Obrázok 28. Stavový diagram v prostredí Active-HDL



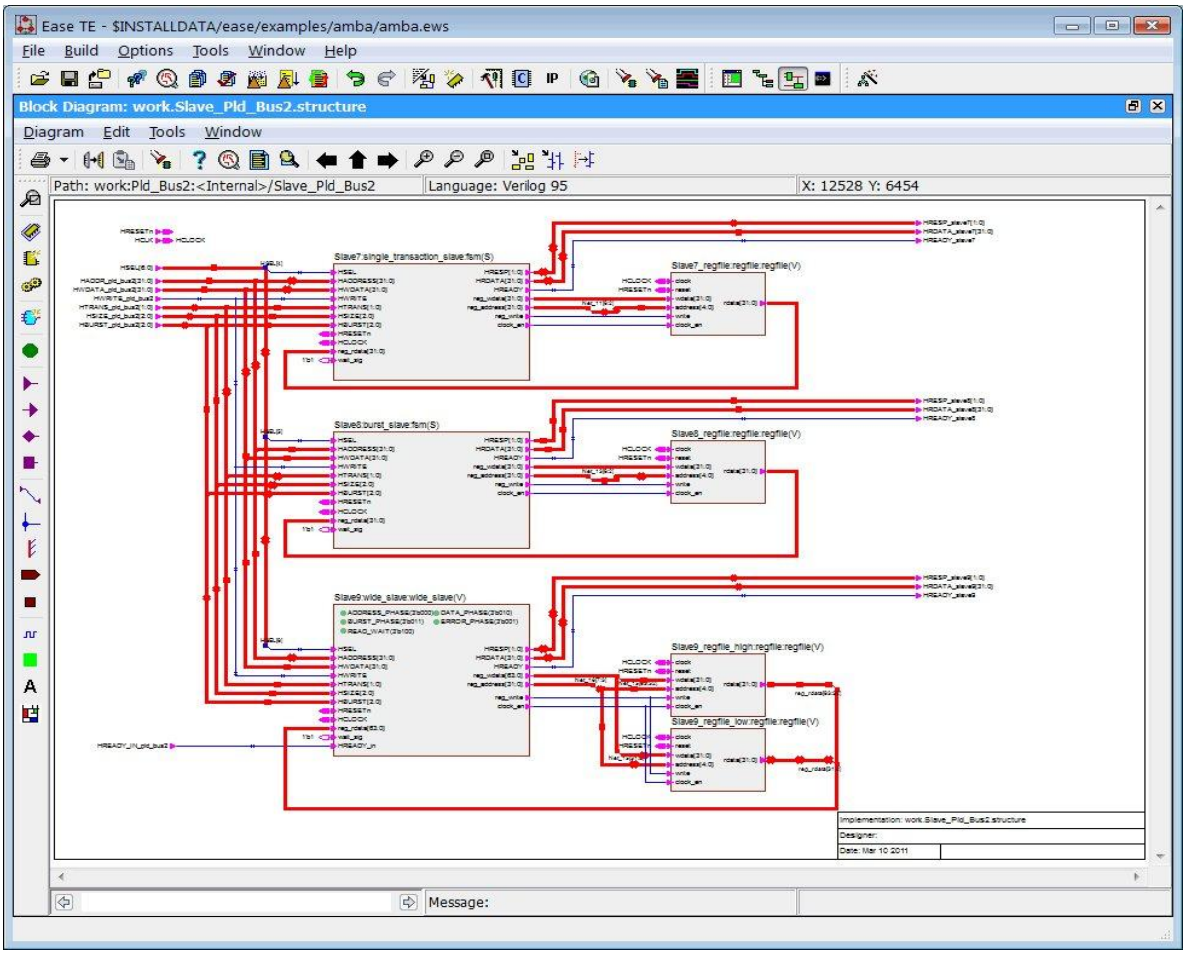
Obrázok 29. Simulácia vizualizácie v prostredí Active-HDL



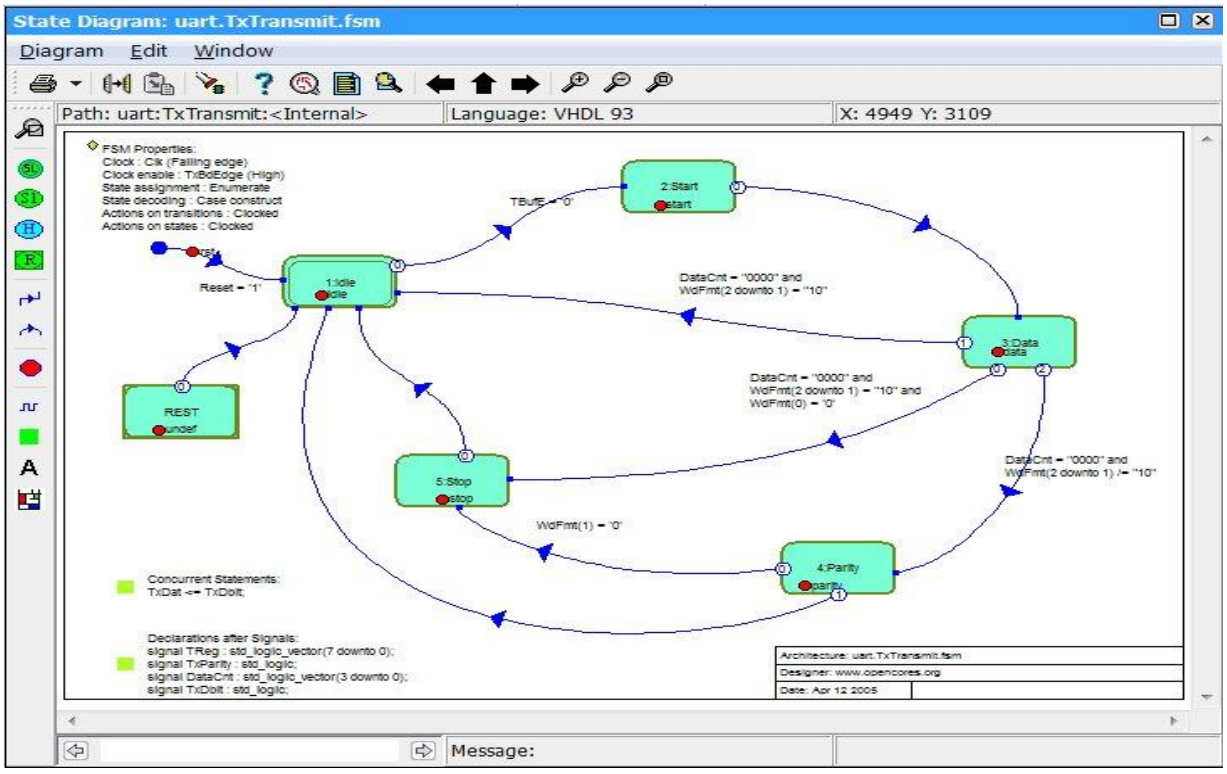
Obrázok 30. Študentská verzia programu Active-HDL

5.1.7 EASE

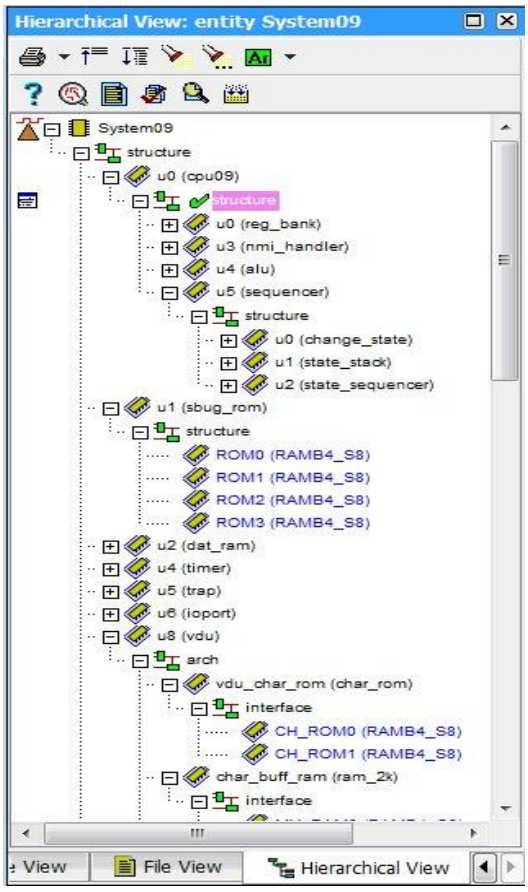
Ďalším nástrojom, ktorý si popíšeme, bude riešenie od firmy HDL Works EASE (15). Tento nástroj, tak ako predošlé, ponúka hierarchicky štruktúrovaný pohľad na projekt. Má v sebe taktiež integrované pokročilé editory blokových a stavových diagramov a pravdivostnej tabuľky a tiež textový editor. Ako veľká výhoda je možnosť integrovania externého HDL kódu, ktorý môže byť vytvorený v inom vývojovom prostredí. Tak ako HDL Author aj tento nástroj sa snaží svojim poňatím zľahčiť a zefektívniť prácu vývojárov v týme. EASE podporuje HDL jazyky Verilog a VHDL. Podporovanou funkciou je aj prevedenie grafického modelu na zdrojový kód a tiež spätne, tak isto ako sme to mohli vidieť u predošlom opisovanom nástroji Active-HDL. Napriek tomu že je aj tento nástroj komerčný, môžeme za výhodu považovať možnosť vyskúšanie trial verzie.



Obrázok 31. Blokový diagram v nástroji EASE



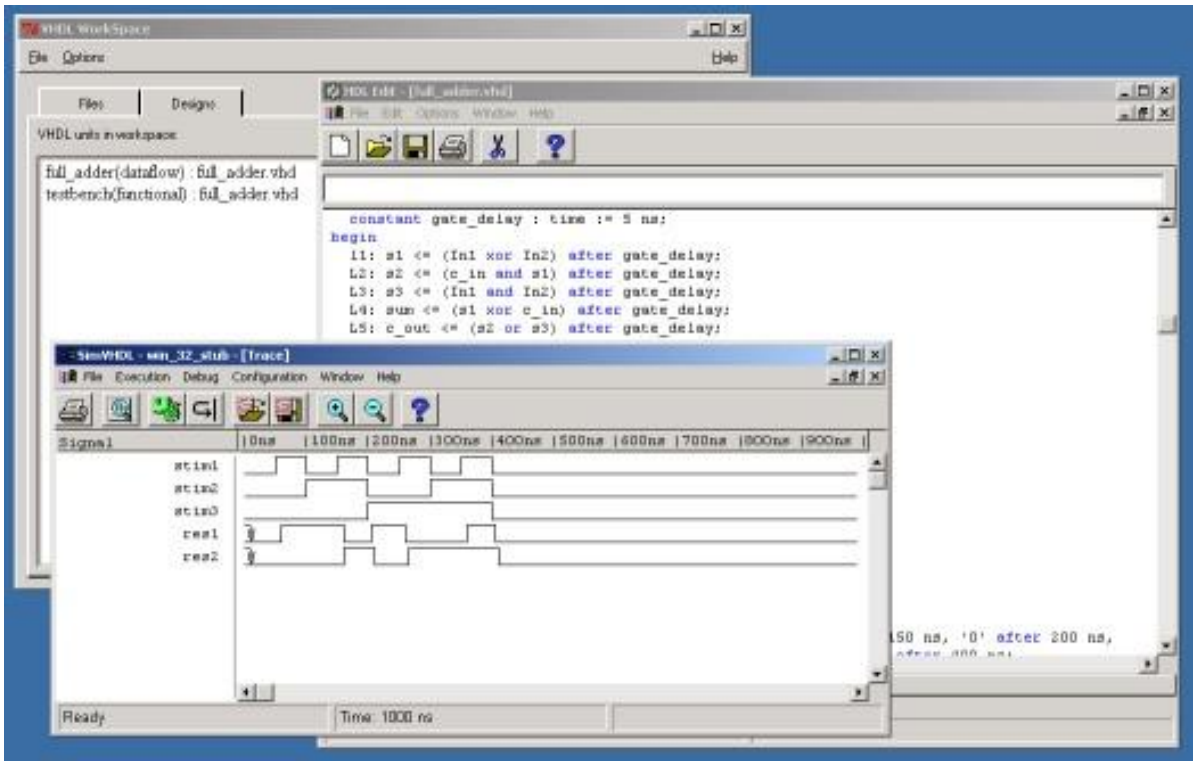
Obrázok 32. Stavový diagram v prostredí EASE



Obrázok 33. Hierarchická štruktúra v nástroji EASE

5.1.8 DirectVHDL

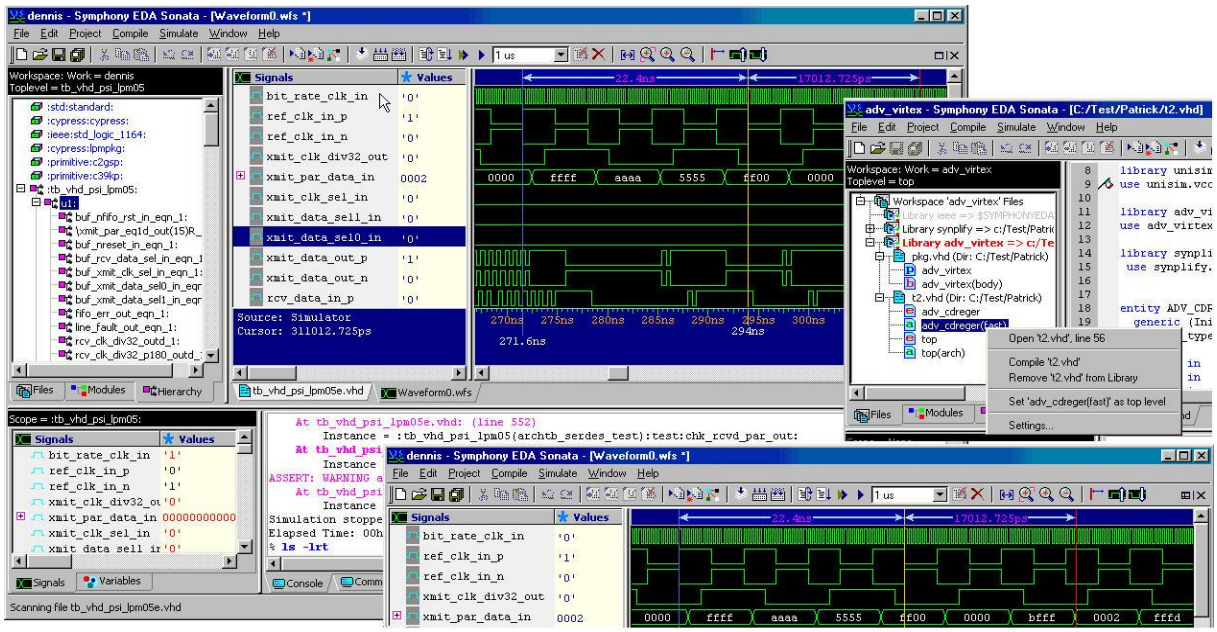
DirectVHDL (16) ponúka používateľsky prívetivé prostredie z ktorého možno spúšťať editor alebo simulátor. Tento nástroj je vytvorený hlavne pre ľudí bez skúsenosti s VHDL kódom a to sa odráža aj na editore, ktorý dokáže farebne zvýrazňovať VHDL syntax a taktiež dokáže odhaliť a poukázať na chyby v kóde. Simulácia návrhu prebieha opäť vo forme časového priebehu signálov, ako sme mohli vidieť napr. u nástroja Modelsim. Aplikácia v sebe obsahuje VHDL tutoriál, čo poukazuje na to, že je vytvorený hlavne pre začiatočníkov. Taktiež môžeme poukázať na veľmi jednoduché, používateľsky prívetivé a rýchle editovanie a simulovanie, keďže je to v tomto nástroji previesť v jednom kroku. Napriek jeho jednoduchosti a cieľovou skupinou začiatočníkov nie je tento nástroj zadarmo a cena nie je až taká prívetivá.



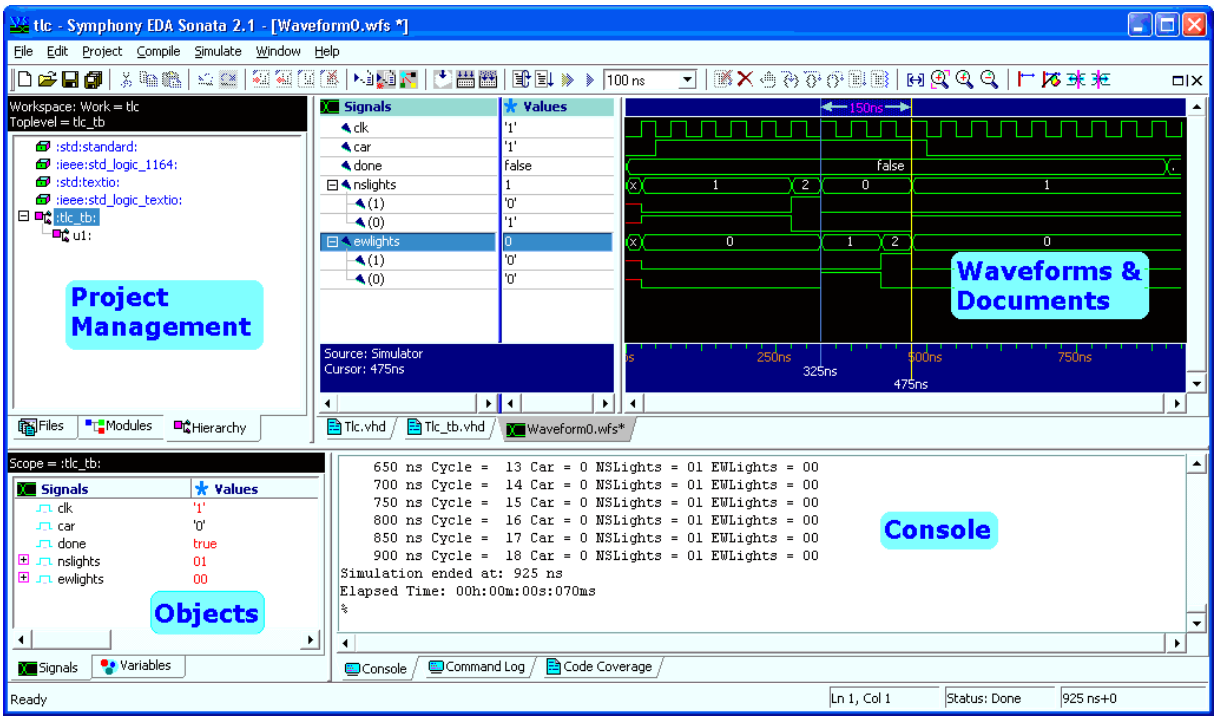
Obrázok 34. Projekt vytvorený v Direct VHDL

5.1.9 VHDL Simili

Ako tvrdí spoločnosť Symphony EDA nástroj VHDL Simili (17) je veľmi silný nástroj na vývoj digitálnych systémov, s množstvom užitočných funkcií, určený hlavne pre skúsenejších vývojárov. Kompilátor a simulátor, integrované v tomto nástroji, je možné spustiť nielen z grafického prostredia aplikácie ale aj pomocou batch skriptov. Taktiež možno kompilovanie a simulovanie považovať za rýchlejšie ako v robustnejších nástrojoch. Simulácia návrhu je však opäť možná vo forme časového priebehu signálov, čiže v tejto oblasti neponúka nič navyše. Čo sa týka licencovania a ceny je VHDL Simili k používateľom právetivejší a dokonca je dostupný študentom na vyskúšanie malých projektov zadarmo. Pre väčšie projekty môže byť taktiež vyskúšaný a to či už možnosťou demo verzie alebo trial verzie.



Obrázok 35. Vizualizácia simulácie vo VHDL Simili

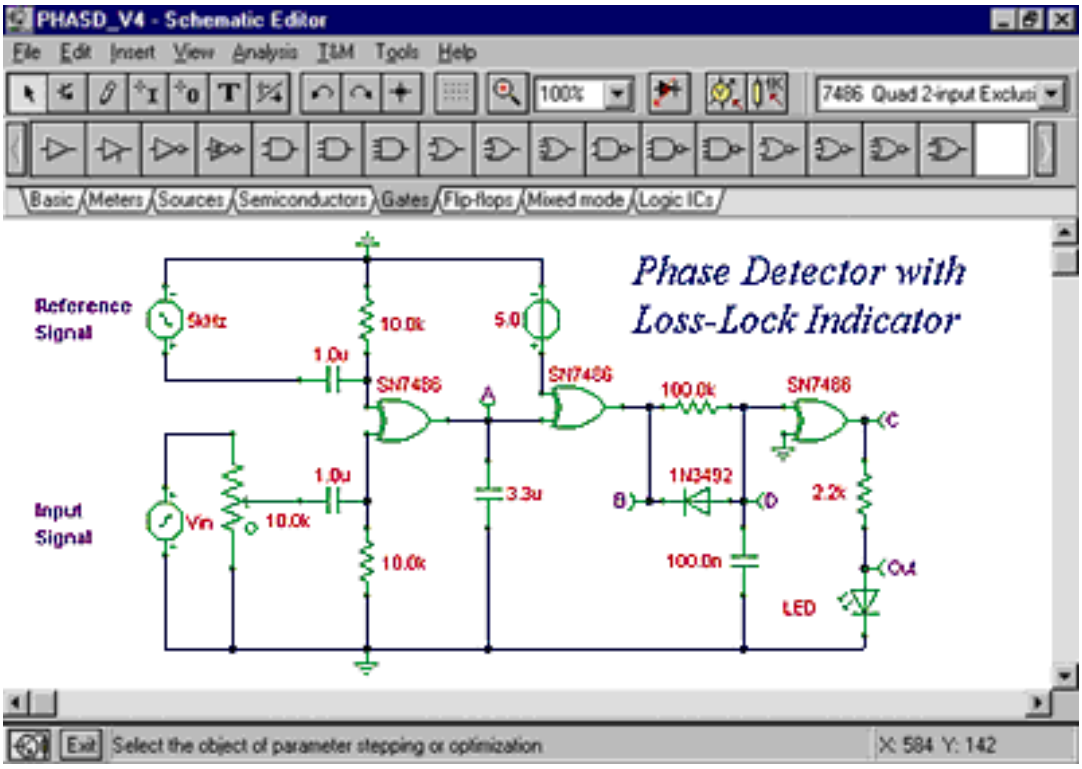


Obrázok 36. Popísané VHDL Simili workspace

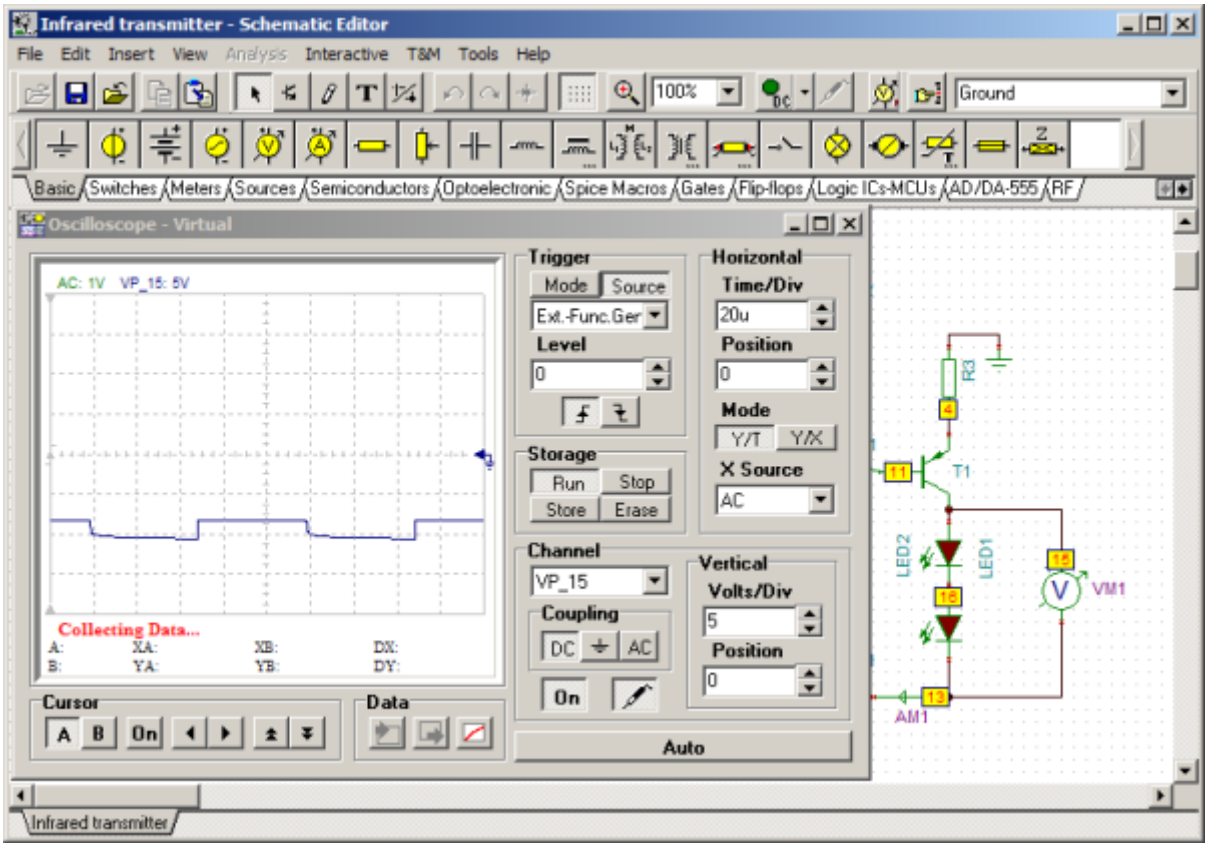
5.1.10 TINA Design Suite

TINA Design Suite (18) je bohatý, veľmi výkonný nástroj pochádzajúci z dielne spoločnosti DesignSoft, ponúkajúci to správne prostredie nielen pre vývojárov, svojou robustnosťou, ale aj začiatočníkom svojim prívetivým a jednoducho poňatým používateľským prostredím. Vo svojom grafickom editore ponúka širokú paletu prvkov od rôznych výrobcov, ktorá je zoskupená v integrovanej knižnici. Zaujímavou vlastnosťou tohto nástroja je možnosť modifikovania, ladenia a testovania návrhu počas behu simulácie. Práve svojimi možnosťami simulácie sa TINA Design Suite líši od vyššie uvedených nástrojov, keďže okrem podpory zobrazenia časového priebehu signálov,

podporuje niekoľko ďalších režimov. Napriek svojej robustnosti nezabúda tento nástroj ani na začiatočníkov či študentov a obsahuje niekoľko integrovaných funkcií, ktoré uľahčujú prácu s touto aplikáciou.



Obrázok 37. TINA Design Suite editor



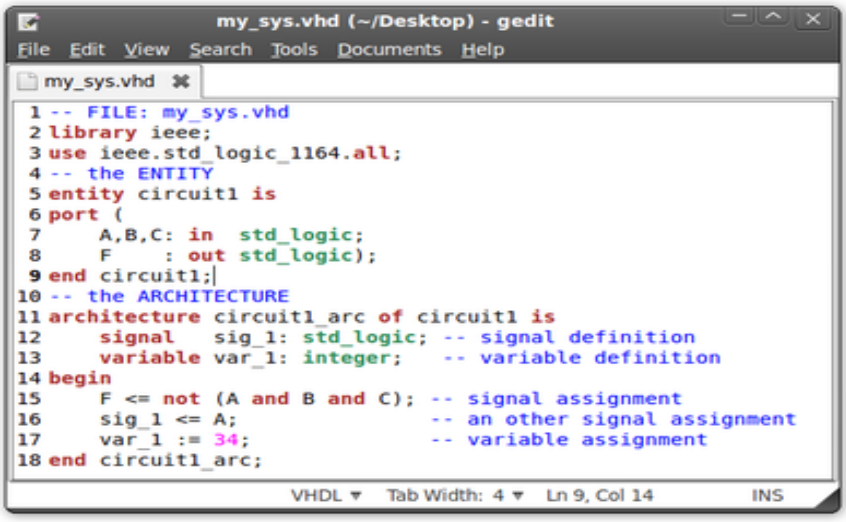
Obrázok 38. Vizualizácie simulácie v prostredí TINA Design Suite

5.2 Dostupné riešenia v „open source“ sfére

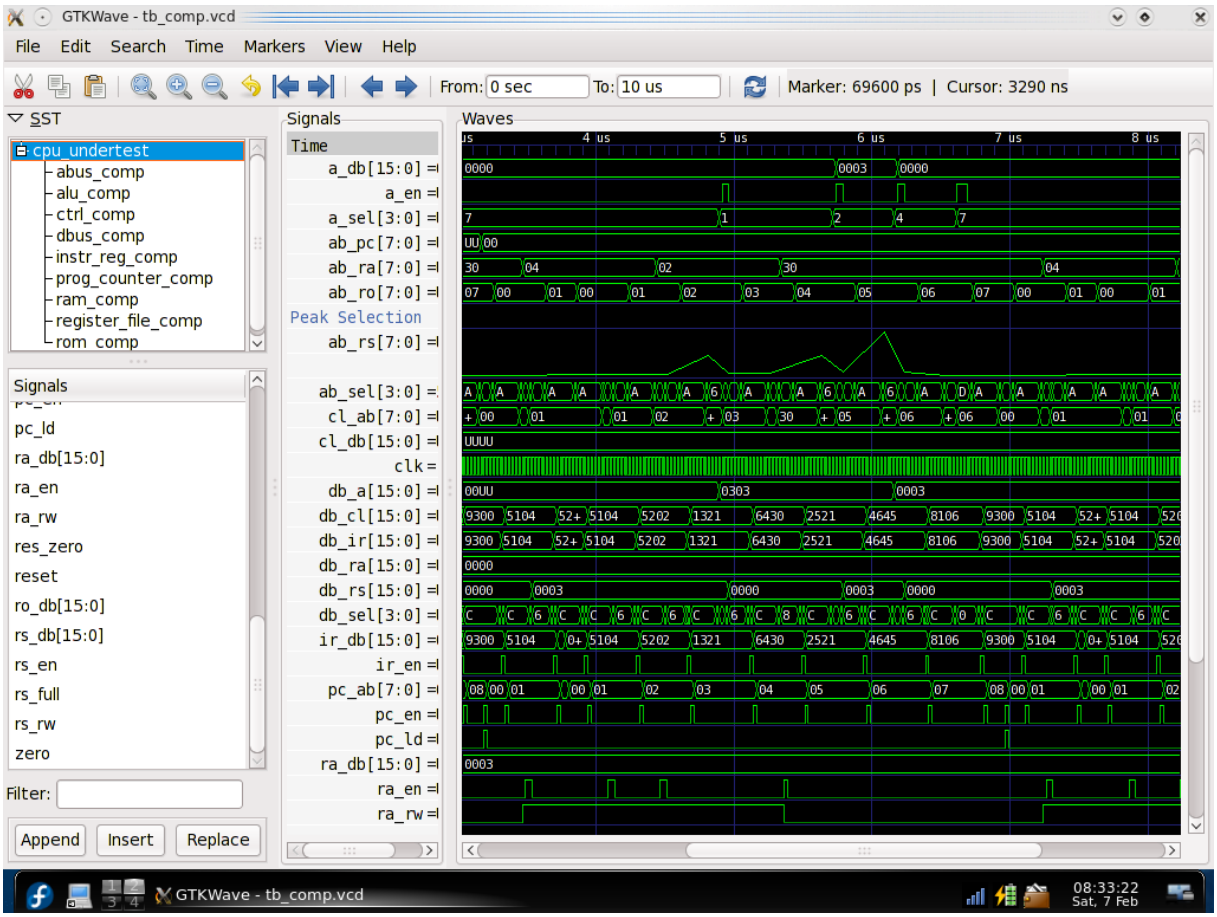
Okrem komerčných riešení existujú aj riešenia vytvorené filozofiou otvoreného zdrojového kódu, ktoré sú voľne dostupné bežným ľuďom. Bohužiaľ keďže vývojári týchto nástrojov nie sú ziskový, tieto nástroje nie sú až také robustné ako konkurencia v komerčnej sfére a ich množstvo je oproti nim taktiež veľmi skromné. V tomto dokumente opíšem dvoch zástupcov tejto sekcie ktorý stoja za zmienku a to FreeHDL a GHDL.

5.2.1 GHDL

Najvydarenejším nástrojom v tejto sekcii je jednoznačne GHDL (19), ktorý sa svojou robustnosťou vyrovná aspoň niektorým nástrojom v komerčnej sfére. Veľmi vhodnou vlastnosťou je využitie textového editoru gedit, ktorý je jednotkou hlavne čo sa týka Linux systémov. Ako druhý nástroj ktorý VHDL využíva je GTKWave pomocou ktorého je možné verifikovať a simulovať návrh. GHDL podporuje dva formáty. Prvým je VCD formát, ale keďže je orientovaný na Verilog, nepodporuje všetky VHDL typy a preto bol vyvinutý druhý formát priamo v GHDL. Ako výhodu tohto nástroja možno taktiež považovať jeho používateľské prostredie, možnosť prezerania tutoriálov a taktiež veľmi dobra používateľská príručka. Tieto vlastnosti s dostatočnou komunitou robia GHDL iným od komerčných riešení resp. ostatných dostupných riešení.



Obrázok 39. VHDL projekt v prostredí editora gedit



Obrázok 40. Simulácia v prostredí GTKWave

5.2.2 FreeHDL

Dá sa povedať, že tento nástroj je ešte v ranom štádiu vývoja. Jediná výhoda, ktorá stojí za zmienku je zatiaľ len dostupnosť a vývoj aplikácie pod GPL. Čo sa týka nevýhod je azda najhoršie, že je tento nástroj podporovaný iba pod operačným systémom Linux. Ako ďalšou nevýhodou je možné vnímať nedostatočné ciele projektu, keďže či už grafický simulátor zobrazujúci návrh formou časového priebehu signálov alebo možnosť ladenia návrhu počas simulácie, podporuje už niekoľko návrhov. Teda FreeHDL (20) nielen že neponúka nijakú funkcionalitu ktorou by sa mohlo líšiť od ostatných riešení v tejto sfére, ale ani plány projektu zatiaľ takúto funkcionalitu nepodporujú. Avšak možno je cieľom projektu vytvoriť úplne nový návrh a teda začať úplne “od podlahy”. To sa však dozvieme až počas vývoja tejto aplikácie.

5.3 Dostupné riešenia vytvorené na FIIT

Téma nášho tímového projektu, vizualizácia modelov digitálnych systémov, je v poslednom období veľmi živou témou a preto aj na našej fakulte sa v tejto oblasti pohybovalo viacero študentov resp. profesorov. Na našej fakulte vznikalo resp. vyvíjalo sa viacero aplikácii zaoberajúcich sa touto problematikou. Dva nástroje budú bližšie popísané v tomto dokumente a to či už HDL Visualization od autora Michala Nosáľa alebo aplikácia VHDL Visualizer od autora Dominika Macka. Prvá práca je zameraná na jazyk Verilog a druhá na jazyk VHDL.

6 Špecifikácia požiadaviek

Špecifikácia požiadaviek na systém vychádza zo zadania projektu. Požiadavky sú špecifikované na základe požiadaviek zadania a požiadaviek vyplývajúcich z analýzy jazykov Verilod, VHDL, SystemC a existujúcich riešení. Požiadavky sú rozdelené na funkcionálne a nefunkcionálne.

Požiadavky systému	
Funkcionálne	**Nefunkcionálne**
Načítanie modelu zo súboru	Jednoduchá inštalácia
Analyzovanie zdrojového kódu modelu	Intuitívne rozhranie
Rozoznanie Verilog, VHDL, SystemC modelu	Príjemné používateľské rozhranie
Vizualizácia štruktúry modelu z univerzálnej reprezentácie	Rýchlosť voľby
	Rýchlosť reakcie programu
	Platforma Microsoft Windows

Funkcionálne požiadavky:

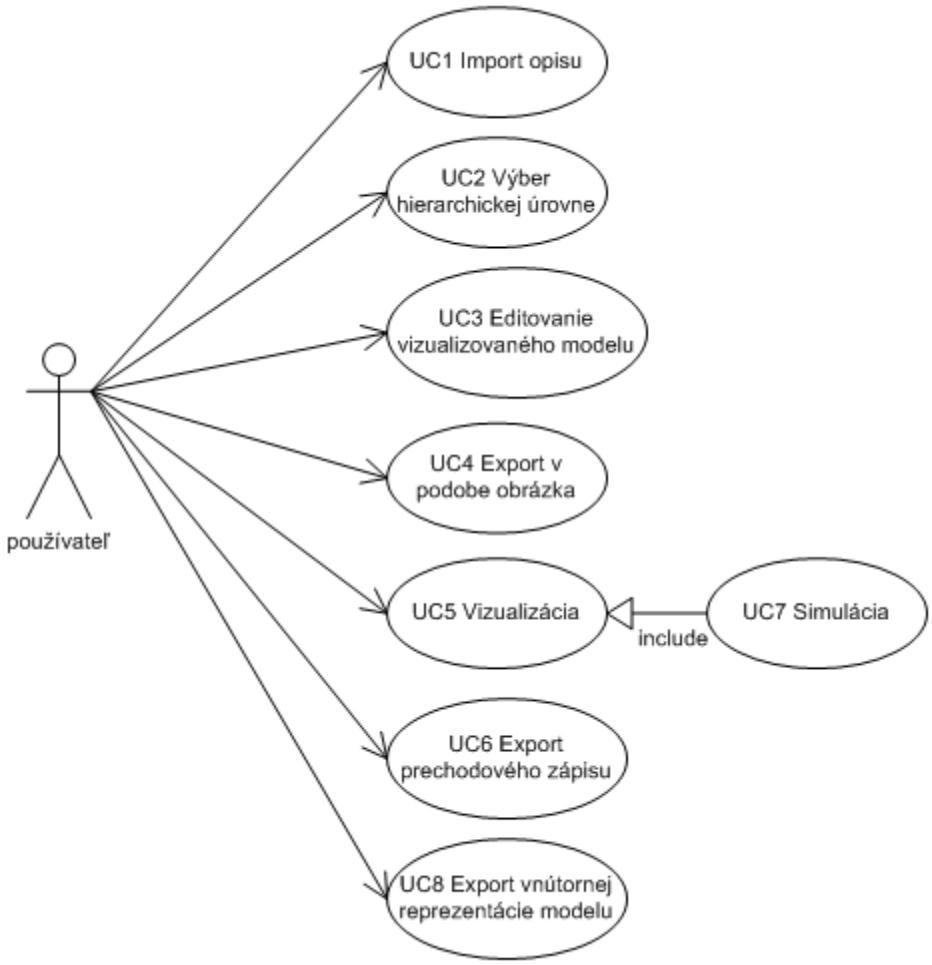
- načítanie modelu zo súboru – systém používateľovi umožní načítať model zo súboru
- analyzovanie zdrojového kódu modelu - systém zanalyzuje zdrojový kód načítaného modelu
- rozoznanie Verilog, VHDL, SystemC modelu – systém na základe prebehnutej analýzy rozozná model v jazyku Verilog, VHDL alebo SystemC
- vizualizácia štruktúry modelu z univerzálnej reprezentácie – systém následne vizualizuje štruktúru modelu z univerzálnej reprezentácie. Aby sme dostali univerzálnu reprezentáciu, systém musí rozanalyzovať zdrojový súbor.

Nefunkcionálne požiadavky:

- jednoduchá inštalácia – pre jednoduché a rýchle použite sa systém nebude priamo inštalovať, bude sa nachádzať v skomprimovanom archíve *.zip
- rýchlosť voľby – systém bude pre používateľov jednoducho ovládateľný, čo bude zabezpečovať prehľadné a upravené používateľské rozhranie.
- rýchlosť reakcie programu – program bude pohotovo reagovať na požiadavky vykonávané používateľom
- platforma Windows – medzi základné nefunkcionálne požiadavky patrí podpora pre platformu Windows, hlavne z dôvodu veľkého rozšírenia medzi používateľmi
- používateľské rozhranie príjemné pre bežného používateľa, ľahká navigácia v okne, jednoduché ovládacie prvky, intuitívne spracované funkcionálne požiadavky

7 Prípady použitia

V nasledujúcej kapitole sú na obrázku č. 41 niektoré prípady použitia vo forme diagramu. Bližší popis jednotlivých prípadov sa nachádza nižšie.



Obrázok 41. Prípady použitia

UC1 Import opisu - systém umožní používateľovi načítať existujúci zdrojový kód modelu.

UC2 Výber hierarchickej úrovne - systém umožní používateľovi zobraziť určitú hierarchickú úroveň. Vygenerovaný prechodový zápis musí zodpovedať pôvodnému HDL opisu.

UC3 Editovanie vizualizovaného modelu – ide o zmenu veľkosti objektov reprezentujúcich moduly, zmena polohy týchto objektov na obrazovke, zmena tvaru prepojení medzi týmito objektmi, pridávanie poznámok a podobne.

UC4 Export v podobe obrázka - systém umožní grafickú schému zobrazenej hierarchickej úrovne exportovať do formátu obrázka.

UC5 Vizualizácia – používateľ ma možnosť vizualizovať HDL opis.

UC6 Export prechodového zápisu - vytváraný systém musí používateľovi umožniť vytvorený a prechodový zápis uložiť pre prípadný import.

UC7 Simulácia – systém dokáže simulovať návrh modelu digitálneho systému. Teda pre každý vstup entity vráti správny výstup.

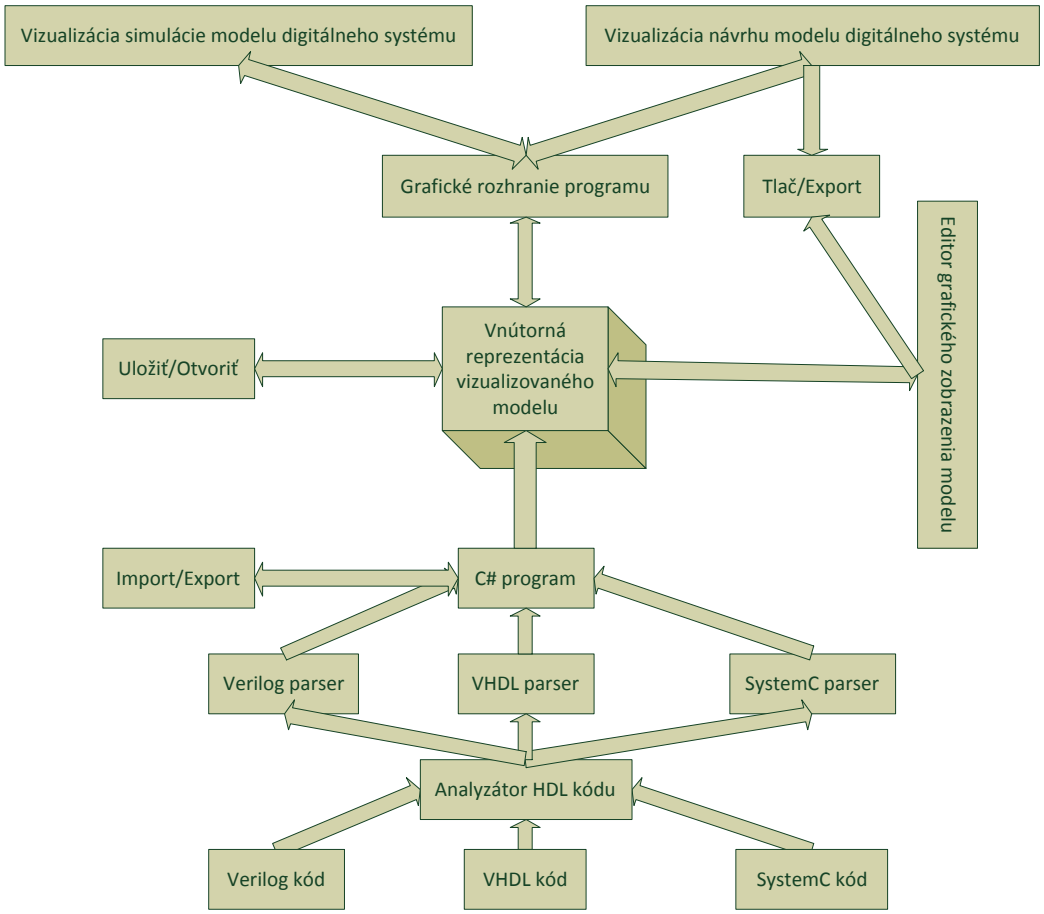
UC8 Export vnútornej reprezentácie opisu – systém umožní používateľovi uložiť vnútornú reprezentáciu opisu pre jej nasledovný import.

8 Hrubý návrh

Aplikáciu pre vizualizáciu modelov digitálnych systémov sme sa rozhodli implementovať v operačnom systéme Microsoft Windows. Medzi hlavné dôvody patrí to, že programovací jazyk, v ktorom sa bude vyvíjať daná aplikácia, je taktiež produkt firmy Microsoft a funguje iba v tomto operačnom systéme. Ďalším dôvodom bolo veľmi rozšírené používanie systému Windows, či už v bežných osobných počítačoch, alebo na akademickej pôde. Operačný systém Windows sa veľmi často používa pri vzdelávacích procesoch v skoro všetkých školách. Tento fakt môže pomôcť k väčšiemu rozšíreniu tejto aplikácie.

Programovací jazyk sme si určili rovnaký, v akom bola implementovaná aplikácia v diplomovej práci, z ktorej sme vychádzali. Tým je C# (Sharp), vyvíjaný firmou Microsoft. Tento jazyk je pomerne intuitívny, prehľadný a syntaxou sa podobá Jave, s ktorou má skúsenosti väčšina nášho tímu.

8.1 Architektúra systému



Obrázok 42. Architektúra systému

V hrubom návrhu si môžeme architektúru systému predstaviť tak ako je zobrazená na obr. 42. Keďže moduly ako Tlač/Export, Uložiť/Otvoriť a Import/Export sú veľmi jednoduché, čo sa týka zložitosti ich vykonávania, nebudem ich rozoberať. Tak isto moduly Verilog kód, VHDL kód a SystemC kód, keďže tieto moduly prestavujú iba kódy v HDL jazykoch ktoré budú načítané analyzátorom, s ktorým začneme predstavovanie jednotlivých modulov.

8.1.1 Analyzátor HDL kódu

Tento modul bude slúžiť na analýzu jednotlivých kódov, ktoré mu budú podsunuté. Teda zo vstupu, ktorý bude predstavovať kód nejakého HDL jazyka resp. viacero kódov vo viacerých HDL jazykov, zanalyzuje o aký HDL jazyk sa jedná a podľa toho podsunie tento kód nejakému ďalšiemu modulu.

8.1.2 Verilog parser

Ako možno usúdiť z názvu tento modul bude spracovávať nejaký kód vytvorený v jazyku Verilog. Na základe vopred definovanej gramatiky spracuje tento kód a na jeho výstupe sa bude nachádzať súbor, ktorý zatiaľ predpokladáme že bude vo formáte XML.

8.1.3 VHDL parser

Tento parser bude slúžiť na preklad VHDL kódu do výstupného súboru, opäť predpokladáme, že bude vo formáte XML. Taktiež na tento preklad bude používať zadefinovanú gramatiku z ktorej bude vychádzať a na základe ktorej daný výstupný súbor vytvorí.

8.1.4 SystemC parser

Ako posledný z modulov, ktoré budú parsovať, je SystemC parser. Ten takisto ako dva predchádzajúce na základe vytvorenej gramatiky vytvorí výstupný súbor. Pri dvoch predchádzajúcich moduloch môžeme vychádzať z už vytvorených parserov, či už pre Verilog resp. VHDL. Pri tomto treťom parseri zatiaľ nemáme presne určené z čoho budeme vychádzať teda je možné, že tento modul budeme vytvárať my.

8.1.5 C# program

Jadro aplikácie bude tvoriť program vytvorený v prostredí C# a teda je tento modul zatiaľ pomenovaný ako C# program. Tento modul dostane na vstupe súbor v univerzálnej reprezentácii, ktorý prijme od už opisovaných parserov. Na základe vopred definovaných funkcií a operácií bude tento modul schopný daný súbor na vstupe spracovať a preložiť na výstupný súbor, ktorý bude predstavovať vnútornú reprezentáciu modelu.

8.1.6 Vnútorná reprezentácia vizualizovaného modelu

Tento modul bude predstavovať vnútornú reprezentáciu modelu, ktorý budeme vizualizovať. Tento modul bude priamo spolupracovať s modulom grafického rozhrania a taktiež modulom editora grafického zobrazenia modelu, keďže každá zmena vykonaná v tomto module bude musieť byť priamo premietnutá do vnútornej reprezentácie modelu.

8.1.7 Editor grafického zobrazenia modelu

Modul editor grafického zobrazenia modelu bude slúžiť na priamu úpravu navrhnutého modelu v grafickom prostredí. V tejto časti bude poskytnuté používateľovi pridávanie, editovanie ale aj mazanie jednotlivých súčastí navrhnutého modelu. Každá zmena v tomto editore však bude musieť

byť vykonaná v module vnútornej reprezentácie vizualizovaného modelu, keďže každá operácia so sebou nesie aj nepriame zmeny, ktoré v editore grafického zobrazenia modelu nie sú viditeľné.

8.1.8 Grafické rozhranie programu

Modul grafického rozhrania, jeho vzhľad ale aj funkcionalita boli v tomto dokumente už popásané.

8.1.9 Vizualizácia návrhu modelu digitálneho systému

Tento modul bude nepriamo spolupracovať s modulom editorom grafického zobrazenia modelu. Táto spolupráca však bude prebiehať pomocou modulov vnútornej reprezentácie vizualizovaného modelu a modulu grafického rozhrania. Vizualizácia bude veľmi podobná modelu ktorý bude zobrazený v editore grafického rozhrania modelu, ale nebude v ňom možné vykonávať žiadne zmeny. To znamená, že tento modul nebude mať žiaden vplyv na vnútornú reprezentáciu vizualizovaného modelu.

8.1.10 Vizualizácia simulácie modelu digitálneho systému

Táto časť aplikácie bude veľmi podobná modulu vizualizácie návrhu modelu digitálneho systému. Avšak rozdiel bude, že tento modul bude vizualizovať simuláciu daného modulu. Keďže simulácia slúži hlavne na verifikáciu navrhnutého modelu, nebude možné v tomto module vykonávať žiadne zmeny, ktoré by ovplyvnili vnútornú reprezentáciu vizualizovaného modelu.

8.2 Používateľské rozhranie

Podmienkou pre jednoduché používanie programu je intuitívne rozhranie, kde bude používateľ pristupovať k ponúkaným funkcionalitám. Základné prvky, ktoré bude rozhranie obsahovať, sú:

- panel s ponukami funkcií – bude obsahovať všetky funkcie, ktoré program ponúka. Rozdelené budú do kategórií podľa funkčnosti
- nástrojová lišta – na nástrojovej lište budú vybraté niektoré funkcie, ku ktorým sa používateľ dostane priamo kliknutím a nemusí sa preklikávať panelom funkcií
- aktuálne umiestnenie otvoreného súboru – bude informovať používateľa o umiestnení aktuálne otvoreného súboru
- okno s vizualizáciou – hlavná časť používateľského rozhrania
- stavový riadok – oznamuje stav, v akom sa program nachádza

9 Bibliografia

1. **Nosáľ, Bc. Michal.** *Vizualizácia Verilog modelov digitalnych systémov.* Bratislava : Slovenská technická univerzita, Fakulta informatiky a informačných technológií, 2010.
2. Mentor Graphics. *HDL Author.* [Online] 2010. [Dátum: 1. 11 2011.] http://www.mentor.com/products/fpga/hdl_design/hdl_author/.
3. Doulos. *What is VHDL?* [Online] [Dátum: 3. 11 2011.] http://www.doulos.com/knowhow/vhdl_designers_guide/what_is_vhdl/.
4. VHDL. *Overview and Application Field .* [Online] [Dátum: 3. 11 2011.] http://www.vhdl-online.de/tutorial/englisch/inhalt.htm.
5. **Macko, Bc. Dominik.** *Vizualizácia VHDL modelov digitálnych systémov.* Bratislava : Slovenská technická univerzita, Fakulta informatiky a informačných technológií, 2011.
6. IEEE. *Standard SystemC Language Reference Manual.* [Online] [Dátum: 31. 10 2011.] http://standards.ieee.org/getieee/1666/download/1666-2005.pdf.
7. OSCI Standart:. *Open SystemC Initiative (OSCI).* [Online] [Dátum: 31. 10 2011.] http://www.systemc.org/downloads/standards/.
8. **Turoň, Ján.** *Vizualizácia opisu v jazyku SystemC.* Bratislava : Slovenská technická univerzita, Fakulta informatiky a informačných technológií, 2008.
9. **Turoň, Bc. Ján.** *Vizualizácia simulácie SystemC modelu.* Bratislava : Slovenská technická univerzita, Fakulta informatiky a informačných technológií, 2010.
10. Mentor Graphics. *HDL Designer.* [Online] 2010. [Dátum: 1. 11 2011.] http://www.mentor.com/products/fpga/hdl_design/hdl_designer_series/.
11. Mentor graphics. *Leonardo Spectrum.* [Online] 2010. [Dátum: 1. 11 2011.] http://www.mentor.com/products/fpga/simulation/modelsim.
12. Mentor Graphics. *ModelSim.* [Online] 2010. [Dátum: 1. 11 2011.] http://www.mentor.com/products/fpga/simulation/modelsim.
13. Mentor Graphics. *Visual Elite HDL.* [Online] 2010. [Dátum: 1. 11 2011.] http://www.mentor.com/products/fpga/hdl_design/visual-elite-hdl/.
14. Aldec. *Active-HDL.* [Online] 2010. [Dátum: 1. 11 2011.] http://www.aldec.com/products/active-hdl/.
15. HDL Works. *EASE.* [Online] [Dátum: 1. 11 2011.] http://www.hdlworks.com/products/ease/index.html.
16. Green Mountain Computing Systems. *DirectVHDL.* [Online] [Dátum: 1. 11 2011.] http://www.gmvhdl.com/directVHDL.html.
17. Symphony EDA. *VHDL Simili.* [Online] [Dátum: 1. 11 2011.] http://www.symphonyeda.com/products.htm.
18. DesignSoft. *TINA Design Suite.* [Online] [Dátum: 1. 11 2011.] http://www.tina.com/English/tina/start.php.
19. **Gingold, T.** *GHDL.* [Online] [Dátum: 1. 11 2011.] http://ghdl.free.fr/.
20. The FreeHDL Project. *FreeHDL.* [Online] [Dátum: 1. 11 2011.] http://www.freehdl.seul.org/.
21. Verilog Resources. *Verigol.com.* [Online] 2011. [Dátum: 1. 11 2011.] http://www.verilog.com.
22. Wikipedia. *Verilog.* [Online] 2011. [Dátum: 1. 11 2011.] http://en.wikipedia.org/wiki/Verilog.