Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava 4

Tímový projekt II
Prostredie pre návrh digitálnych systémov
(Digital System Designer)

Dokumentácia pre letný semester

Číslo tímu: 10
Členovia tímu: Bc. Peter Jurík, Bc. Ján Janičkovič, Bc. Jozef Vicen, Bc. Pavol Briš, Bc. Marián Bednár
Vedúci tímového projektu: Ing. Matej Jurikovič
Ročník, typ štúdia: 1, inžinierske štúdium
Akademický. rok: 2010/2011

Obsah

1 Úvod

1.1 Zloženie tímu

Pedagogický vedúci projektu	Ing. Matej Jurikovič
Členovia tímu	Bc. Marián Bednár
	Bc. Pavol Briš
	Bc. Ján Janičkovič
	Bc. Peter Jurík
	Bc. Jozef Vicen

1.2 Zadanie

Každý zo študentov odboru PKSS sa počas svojho štúdia stretol s viacerými prostrediami pre návrh digitálnych systémov. Jedná sa o rôzne úrovne návrhu od klasických kombinačných logických obvodov , cez použitie Petriho sietí na opis správania systému až po návrh procesorov alebo ASIC obvodov. Základným problémom je nízka podpora programových systémov pri testovaní niektorých spôsobov návrhu a následnej simulácie, poprípade syntézy. Z tohto dôvodu je cieľom vytvoriť prostredie, s ktorým by študenti mohli pracovať na čo najväčšom počte predmetov zameraných na návrh digitálnych systémov a zastrešovalo by všetky metodiky návrhu na rôznych úrovniach. Významné z pohľadu výskumu na našej fakulte je rovnako aj vytvorenie prostredia pre testovanie jednotlivých skúmaných metód (v rámci ústavu, fakulty, SR,..).

Niektoré z hlavných cieľov projektu:

- Základ pre modulárny aplikačný systém umožňujúci prácu s čo najväčším množstvom metodík návrhu.
- Umožniť dodatočné pridávanie nových metodík.
- Podporovať súborové štandardy - BLIF, KISS, SLIF, atd.
- Zahrnutý grafický editor pre klasické hradlové obvody, Petriho siete, Konečné stavové automaty, prípadne iné grafické modely používané pri návrhu.
- Podpora simulácie daných grafických modelov (príkladom je PIPE pre Petriho siete alebo LOG pre hradlové obvody)

Cieľom Tímového projektu je v prvom kroku vytvorenie základu pre daný modulárny systém. Je nutné podrobne zanalyzovať súvisiacu problematiku a implementovať prostredie spolu s grafickým editorom do ktorého by bolo možné pridávať jednotlivé metodiky návrhu a podporované modely.

1.3 Účel a rozsah dokumentu

Tento dokument vznikol ako výsledok práce na predmete Tímový projekt I počas zimného semestra. Dokument obsahuje správu k projektu s názvom „Prostredie pre návrh digitálnych systémov“ a je určený vedúcemu projektu, posudzujúcemu tímu, ako aj pre všetkých, ktorí sa o danú problematiku zaujímajú.

Účelom dokumentu je priblížiť čitateľovi analýzu problémovej oblasti a opísať možnosti ako realizovať vytvorenie prostredia na podporu návrhu a testovania digitálnych systémov.

1.4 Prehľad dokumentu

Dokument má nasledovnú štruktúru:

Kapitola 0: Úvod do dokumentu, a prehľad jeho súčastí

Kapitola 1: Analýza problémovej oblasti

Kapitola 2: Špecifikácia požiadaviek

Kapitola 3: Návrh systému

Kapitola 4: Zoznam zdrojov, použitých pri tvorbe dokumentu

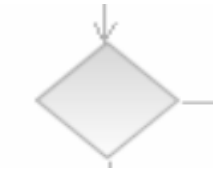
1.5 Použité pojmy a skratky

Skratka / pojem	Vysvetlenie
BDD	Binary Decision Diagram (binárny rozhodovací diagram)
BDS	BDD based logic optimization System (systém na logickú dekompozíciu BDD)
BLIF	Berkley Logic Interchange Format (súborový systém pre zápis logickýc obvodov)
EQN	Equation Format (rovnicový súborový formát)
FPGA	Field Programable Gate Array
FSM	Finish State Machine (konečný stavový automat)
KISS	Keep It Simple Stupid (tabuľkový formát pre zápis logických obvodov)
LUT	Look-Up Table (tabuľka pravdivostných hodnôt)
PLA	Programmable Logic Array (programovateľné zariadenie na kombinačné obvody)
SLIF	Specification-Level Intermediate Format (špecifikačný formát komponentov)
VHDL	VHSIC Hardware Description Language (hardvérový opisný jazyk)
VHSIC	Very High Speed Integrated Circuit (vysokorýchlostný integrovaný obvod)

1.6 Použitá notácia

V dokumente sú použité nasledovné komponenty diagramov:

Štart procesu



Rozhodovací blok

Proces

Koniec procesu

2 Analýza

V tejto časti dokumentácie sú opísané vybrané súborové štandardy používané na opis logických obvodov, vybrané grafické simulátory Petriho sietí, vybrané textové editory slúžiace na zápis kódu, vybrané existujúce programové systémy určené na návrh digitálnych systémov a celkové zhodnotenie aj čo sa týka nami vybraných použiteľných riešení.

2.1 Súborové štandardy

Pri analýze najznámejších súborových štandardov používaných na opis sekvenčných a kombinačných logických obvodov, ktoré boli doporučené v zadaní projektu (tj. BLIF, SLIF, KISS) sme objavili veľa daľších používaných štandardov, z ktorých sme ďalej analyzovali nasledujúce – PLA a EQN. Analýza jednotlivých štandardov pozostáva z formátu zápisu a opisu ich konkrétneho využitia.

2.1.1 PLA

Pomocou tohto formátu môžeme zapísať dvojúrovňový kombinačný logický obvod v textovej forme. Obvod je reprezentovaný pomocou logického výrazu v disjunktnej forme DNF (súčty súčinov). (napr. x0.x1.x2!+x2!.x3+x1!.x2)

Neúplne definované funkcie

K pochopeniu PLA je nutné poznať definíciu funkcií pomocou množín ON-set, OFF-set, DC-set, ktoré spoločne definujú úplnú funkciu.
ON-set – množina termov, ktorá generuje na výstup logickú 1
OFF-set – množina termov, ktorá generuje na výstup logickú 0
DC-set – neurčené stavy

$$F=(\text{ON-set}, \text{OFF-set}, \text{DC-set}): B^n \rightarrow \{1,0,x\}$$

K úplnej definícií funkcie je potrebné poznať dve množiny a tretia vznikne ako doplnok súčtu známych množín k množine Bn. [1]

Hlavička

V hlavičke sa nachádzajú kľúčové údaje, ktoré definujú vlastnosti obvodu.

.i – počet vstupov
.o – počet výstupov
.ilb – názvy vstupov, ich počet musí zodpovedať počtu vstupov
.ob – názvy výstupov, ich počet musí zodpovedať počtu výstupov
.p – nepovinný parameter, ktorý určuje počet termov v pravdivostnej tabuľke
.type – dôležitý parameter, ktorý určuje spôsob reprezentácie tabuľky termov

Typ f určuje množinu ON-set. OFF-set je doplnok ON-setu a DC-set je prázdna množina.
Typ r určuje OFF-set, ON-set je doplnok OFF-setu a DC-set je prázdna množina.
Typ fd definuje ON-set a DC-set a OFF-set je doplnkom súčtu množín ON-set a DC-set.
Typ fr definuje ON-set a OFF-set. DC-set sa získa ako doplnok k ich súčtu.
Typ dr definuje DC-set a OFF-set. ON-set sa získa ako doplnok k ich súčtu.
Typ fdr je úplne definovaný.

Telo

Telo PLA je pravdivostná tabuľka premenných, ktorá určuje vzťahy medzi kombináciou vstupov a tomu náležitou kombináciov výstupov. Počet znakov vstupnej a výstupnej časti musí byť zhodný s hodnotami zadanými v hlavičke súboru.

Príklad

Počet vstupov je 4. (a,b,c,d) {“.i”}

Počet výstupov je 2. (f, f1) {“.o”}

Následne podľa obvodu vytvoríme pravdivostnú tabuľku: (napr. pre 3 termy môže vyzerať takto)

A	B	C	D	F	F1
1	1	-	-	1	0
-	-	1	1	1	0
1	1	1	1	1	1

Tab.1 Pravdivostná tabuľka (PLA)

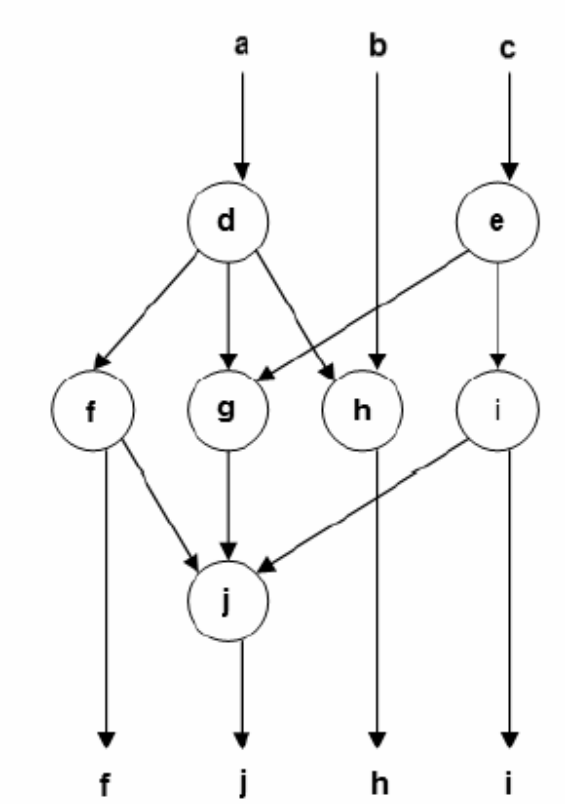
A výsledný súbor PLA bude vyzerať nasledovne:

```
.i 4
.o 2
.ilb a b c d
.ob f f1
.p 3
11-- 10
--11 10
1111 11
.end
```

Súbor je vždy zakončený .e alebo .end.

2.1.2 BLIF

Hlavným cieľom formátu BLIF (Berkeley Logic Interchange Format) je popísať logické obvody v textovej forme. Obvod je ľubovolná kombinačná alebo sekvenčná sieť logických funkcií a môže byť zobrazený ako orientovaný graf kombinačných uzlov a logických elementov. Tento formát vychádza z formátu PLA, avšak oproti nemu umožňuje popísať už spomenuté viacúrovňové kombinačné aj sekvenčné obvody. BLIF súbor sa skladá z viacerých tabuliek (môže byť aj jedna) tzv. K-LUTov (Look Up Table). Každý tento K-LUT je jednovýstupový PLA s K vstupmi. Vzťahy medzi jednotlivými LUTami (tabuľkami) sú jasne definované pomocou signálov. [2]



Obr.1 Štruktúra súboru BLIF (vstupy: a, b, c ;výstupy: f, j, h, i)

Hlavička

Každý BLIF súbor obsahuje jeden alebo viac tzv. modelov alebo odkazov na modely popísané v iných BLIF súboroch. Model slúži na popísanie určitej časti logického obvodu. Model má svoj názov, vstupy a taktiež výstupy.

.model [name] – reťazec name určuje názov modelu
.inputs [s1] . . . [sn] – názvy vstupov (určujú aj počet vstupov)
.outputs [s1] . . . [sn] – názvy výstupov (určujú aj počet výstupov)
.clock – hodinový signál, ak je potrebný

Telo

Telo súboru sa skladá z niekoľkých tzv. LUTov (tabuliek), ktoré obsahuju vlastnú hlavičku.

Hlavička LUTu obsahuje .names [s1] . . . [sn] [sx] , kde s1 až sn sú názvy vstupných signálov a sx je názov výstupného signálu (môže ich byť viac). Vstupné signály môžu nadobúdať hodnoty 1,0 (negovaný signál), - (nepoužitý signál). Výstupné hodnoty signálov môžu byť len 1 alebo 0, pričom sú v celej tabuľke rovnaké.

Telo LUTu sa skladá z riadkov definujúcich súčinové termy podobne ako pri formáte PLA.

Príklad

x0 x1 x2 x3 x4 x5 y

1 1 0 1 0 1 1

---

x0.x1. x2!.x3.x4.x5 = y

Pri viacerých riadkov v tabuľke tvoríme súčty súčinov. Každý súbor BLIF je zakončený .end. Príklad súboru BLIF môže vyzerať napríklad nasledovne:

```
.model traffic_c1
.inputs a b c d e
.outputs f
.name h f
0 1
.names d a b c e h
000-- 1
00-0- 1
0-00- 1
00--0 1
0-0-0 1
0--00 1
.end
```

Logické členy

Ako sme už hovorili BLIF popisuje kombinačné aj sekvenčné obvody. Okrem toho, že si môžeme logické členy sami zadefinovať, tak tento formát umožňuje používať aj určité predpripravené logické členy, ktoré sa načítavaju z rôznych knižníc. [2]

.gate <meno log. člena> <vnútorné mapovanie signálov> - kombinačné log. prvky
.mlatch <meno log. člena> <vnútorné mapovanie signálov> <control> [<init-val>] – pamäťové log. prvky
control – riadenie hodinovým signálom

2.1.3 KISS

KISS (keep it simple stupid) je široko používaný textový formát určený pre popis konečných stavových automatov (FSM), ktorý bol vytvorený na Berkeley University.

- priama reprezentácia stavových prechodových tabuliek sekvenčných obvodov
- používaný na minimalizáciu stavového diagramu
- tabuľkový formát, kde má každý riadok 4 vstupy: vstupné pole, aktuálne stavové pole, nasledujúce stavové pole a výstupné pole
- vždy má toľko riadkov koľko je prechodov v stavovom grafe FSM

Popis FSM v textovom KISS formáte obsahuje rovnako 2 časti: hlavičku a telo tvorené stavovovou prechodovou tabuľkou. [3]

Hlavička

Opisuje všeobecné vlastnosti FSM a je umiestnená pred opisom stavovej prechodovej tabuľky. Obsahuje atribúty riadkov, ktoré začínajú znakom '.'. Zoznam možných atribútov:

.i – špecifikuje počet vstupov v FSM
.o – špecifikuje počet výstupov v FSM (môže byť aj 0)
.p – špecifikuje počet riadkov stavovej tabuľky (môže byť vynechané)
.s – špecifikuje počet stavov v FSM (môže byť vynechané)
.r – špecifikuje stav reset

Telo

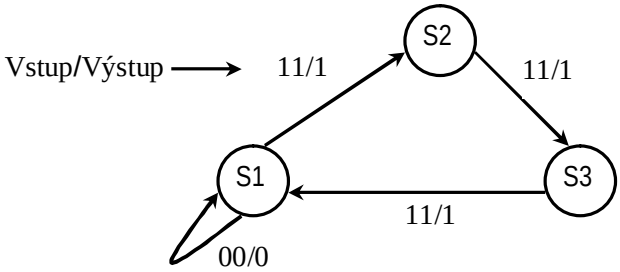
Opis stavovej prechodovej tabuľky obsahuje riadky nasledujúceho formátu:
<vstup> <aktuálny_stav> <nasledujúci_stav> <výstup>

Mená stavov sú špecifikované ľubovoľnými reťazcami znakov bez medzier. Ak nezáleží na mene stavu používame znak '*' (don't care). Vstupy a výstupy sú špecifikované ako trojstavové vektory. Ak je počet výstupov v hlavičke 0 výstupy by nemali byť špecifikované. Trojstavový vektor je reťazec nasledujúcich znakov: '0', '1' a '-' (pričom posledný znak znamená, že nezáleží či je to 0 alebo 1, tzv. don't care). Dĺžka vstupno/výstupných vektorov (je určený počtom znakov v reťazci) by mala zodpovedať počtu vstupy/výstupy v hlavičke. Koniec popisu FSM býva ukončený špeciálnym riadkom '.e' , avšak býva chápaný tiež ako koniec súboru. Popis FSM obsahujúci stav '-' je spracovaný tak, že všetky prechody z tohto stavu budú pridané do všetkých ostatných stavov FSM. Následne nižšie uvádzame príklad zápisu KISS súboru. [3]

```
.i 6
.o 7
.p 22
.s 9
.r s1
10----- s1 s1 11-----
00----- s1 s3 -1-----
01----- s1 s5 ----1--
11----- s1 s6 1------
-1----- s2 s2 -1---1-
-0----- s2 s7 ----1--
---1--- s3 s1 --1----
---00-- s3 s4 --11---
---01-- s3 s7 ---1---
--1---- s4 s3 -1--11-
--0---- s4 s5 ----1--
-----1- s5 s5 ------1
-----0- s5 s9 ------1
1-1---- s6 s7 ----11-
0-1---- s6 s8 ----1--
--0---- s6 s9 -----1-
-1----- s7 s8 -1-----
-0----- s7 s9 1----1-
----1-- s8 s3 --11---
----0-- s8 s8 ---1---
-----1- s9 s1 ------1
-----0- s9 s7 --11—1
.e
```

Príklad

Máme nasledujúci stavový diagram:



Obr.2 Stavový diagram FSM

Počet vstupov je 2 {“.i”}

Počet výstupov je 1 {“.o”}

Počet stavov je 3 {“.s”}

Počet riadkov stavovej tabuľky je 4 {“.p”}

Po špecifikovaní vstupov a výstupov špecifikujeme stavovú tabuľku sekvenčného obvodu.

Vstup	AS	NS	Výstup
00	S1	S1	0
11	S1	S2	1
11	S2	S3	1
11	S3	S1	1

AS-aktuálny stav
NS-nasledujúci stav

Tab.2 Stavová tabuľka (KISS)

Výsledný súbor KISS bude vyzerať nasledovne:

.i 2
.o 1
.s 3
.p 4
00 s1 s1 0
11 s1 s2 1
11 s2 s3 1
11 s3 s1 1
.e

2.1.4 EQN

Takzvaný rovnicový štandard (Equation format) je jeden z najjednoduchších súborových formátov. Je to priame vyjadrenie logickej reprezentácie rôznych logických členov logického obvodu. Obsahuje 2 časti: hlavičku na definovanie vstupov a výstupov a telo kde definujeme rovnice pre vnútorné signály obvodu a výstupy. [3]

Hlavička

INORDER – špecifikuje počet vstupov
OUTORDER – špecifikuje počet výstupov

Telo

Špecifikácia vnútorných signálov: [meno signálu] = logické vyjadrenie;
Špecifikácia výstupov: [meno signálu] = logické vyjadrenie;

Výstupy obvodu môžu byť vyjadrené ako logická funkcia vnútorných a vstupných signálov logického obvodu.

Príklad
Počet vstupov (a,b,c,d) {“INORDER”}
Počet výstupov (f, f1) {“OUTORDER”}

Máme vnútorný signál p, ktorý môže byť reprezentovaný napríklad ako [p] = a * b; a vnútorný signál q, ktorý môže byť napríklad [q] = c * d;. Možný výsledný formát EQN je nasledujúci:

```
INORDER = a b c d;
OUTORDER = f  f1;
[p] = a * b;
[q] = c* d;
f= [p] + [q]
f1= [p]*[q];
```

2.1.5 SLIF

SLIF reprezentuje štrukturálny systém komponentov a priradení ich funkčných objektov. Štrukturálne komponenty môžu byť napr. štandardný alebo špeciálny procesor, pamäť, zbernica, ktoré implementujú premenné. Pri návrhu čipu sa používa vyššia úroveň abstrakcie. SLIF formát sa na rozdiel od grafu riadenia toku dát (data flow control) používa pre systémovú úroveň návrhu. Návrh systému prechádza k systémovej úrovni a pri tejto úrovni návrhu, už nie sú vhodné control dataflow diagramy. Preto je vhodné použiť formát SLIF. Tento formát používa SpecSyn. [4]

Výhody

- Vie reprezentovať funkciu aj štruktúru.
- Dokáže riešiť problém s vysokou mierou abstrakcie. [12]

Nevýhody

- Nereprezentuje operácie s jemnou granularitou.
- Nie je simulovateľný. VHDL štandard je veľmi populárny, a nie je potreba simulátora pre tento formát zápisu.
- Odhad hardvéru pre nejakú množinu procedúr, nemôže byť odhadnutý presne tým, že sa sčíta veľkosť jednotlivých procedúr, preto sa vyvinul sofistikovaný zápis, ktorý zvažuje hardvérové zdieľanie medzi procedúrami.
- SLIF asociuje iba jeden priemerný čas ku každej procedúre. Tento čas je často závislý, na mieste, kde sa táto procedúra vykonáva. [12]

2.1.6 Vzájomné porovnanie

Analyzované súborové štandardy PLA, BLIF, KISS, EQN majú textovú formu a sú riadkovo-založené. Formát týchto súborov vyžaduje, aby riadky dát boli ukončené jedným z nasledujúcich znakov: '\r', '\n' alebo '\r\n'. Časť riadka, ktorá nasleduje za znakom '#' je považovaná za komentár a je ignorovaná. Súbor PLA je vhodný skôr na opis jednoduchého kombinačného obvodu lebo neposkytuje možnosť opisovať zložitejšie sekvenčné obvody. Súbor BLIF je nadstavba PLA a je vhodný na opis správania sa kombinačných aj sekvenčných obvodov jednoduchou formou. Taktiež dokážeme pomocou tohto formátu zapísať konečný stavový automat (FSM). Súbor EQN slúži na opis správania sa logických členov obvodu a nie je vhodný na opis zložitejších kombinačných a sekvenčných obvodov. Súbor KISS je možnosťami opisu sekvenčných a kombinačných obvodov a stavových automatov veľmi podobný súboru BLIF, ale má iný spôsob zápisu.

Štandard SLIF slúži na opis a abstrakciu napr. procesora, pamäte alebo zbernice. Reprezentuje štrukturálny systém takýchto komponentov. Vie reprezentovať funkciu aj štruktúru. Používa SpecSyn čo je systémové návrhové prostredie, ktoré môže byt rozšírené na zvládnutie mnohých problémov.

2.2 Simulátory Petriho sietí

Na vytváranie modelov systémov pomocou Petriho sietí nám slúžia grafické simulátory. Obsahujú grafický editor pre vkladanie a spájanie stavebných prvkov siete s možnosťou odsimulovania ich činnosti a analýzu vlastností správania sa a štruktúry takto vytvoreného modelu. Väčšinou bývajú určené pre špecifický typ Petriho siete. Umožňujú čo najlepšie pochopenie problematiky vytvárania modelov Petriho sietí. Uvádzame úzky výber niektorých z najpoužívanejších simulátorov. [7]

2.2.1 Platform Independent Petri Net Editor 2 (Pipe 2)

Simulátor je nezávislý od platformy, čo znamená že môže pracovať v každom operačnom systéme. Je určený na vytváranie, editovanie a simuláciu P/T Petriho sietí s následnou možnosťou ich analýzy.

2.2.2 Petri.NET Simulator

Simulátor je vytvorený pre systém MS Windows. Je určený taktiež na vytváranie, editovanie a simuláciu P/T Petriho sietí s možnosťou pridania časových závislostí a umožňuje ich čiastočnú analýzu.

2.2.3 HPSim

Simulátor je vytvorený pre systém MS Windows. Má rovnaké schopnosti, ako predchádzajúci simulátor, ale navyše vie modelovať aj stochastické časové závislosti.

2.2.4 CPN Tools

Simulátor je vytvorený pre systém Linux a MS Windows. Je určený na vytváranie, editáciu a simuláciu vysoko-úrovňových farebných Petriho sietí a časovaných Petriho sietí s možnosťou ich analýzy.

2.2.5 Vzájomné porovnanie

Simulátor	Prostredie	Simulácia		Analýza vlastností		Výpočet invariantov		Export do súboru
		Kroková	Rýchla	Jednoduchá	Štrukturálna	P	T	
Petri .NET simulator [11]	MS Windows	X	X	X				
Pipe[12]	Java	X	X	X	X	X	X	X
HPSim [13]	MS Windows	X	X	X				
JpetriNet [14]	Java	X	X		X			

Obr.3 Porovnanie simulátorov Petriho sietí [7]

Programových prostriedkov na simuláciu Petriho sietí existuje veľmi veľké množstvo preto sme vybrali len tie najpoužívanejšie, ktorými sú CPN Tools, HPSim, Pipe2 a Petri.NET Simulator. Každý z týchto simulátorov má trochu iné využitie, čo sa týka možností analýzy a návrhu ako aj špecifikácie len pre určitý typ Petriho siete.

2.3 Textové editory

Nasledovné textové editory slúžia na pokročilú prácu s textom a tvorbu kódu. Slúžia na jeho editovanie alebo inými slovami, dovoľujú text vytvárať a upravovať.
Sú jednoduché, intuitívne a podporujú editovanie veľkého množstva súborových formátov. Uvádzame výber z niektorých najznámejších používaných editorov.

2.3.1 Kate

Kate je pokročilý textový editor, ktorý je súčasťou grafického prostredia KDE, jeho názov je akronym pre KDE advanced text editor. Vďaka technológii KParts z KDE je možné použiť rozhranie Kate v akejkoľvek inej aplikácii. Tieto možnosti využívajú napr. vývojové prostredia KDevelop, jednoduchý textový editor KWrite, prostredie pre vývoj webových stránok Quanta Plus alebo editor Kile pro LaTeX. Zároveň využíva KParts i Kate napr. pre zobrazenie emulátora v konsole. Ďalšou výhodou integrácie v KDE je, že Kate dokáže pracovať so súbormi na lokálnych diskoch ale aj so súbormi dostupnými cez všetky protokoly podporovane KIO Slaves (HTTP, FTP, SMB, SSH a WebDAV). [9]

2.3.2 Gedit

Gedit je jednoduchý textový editor. Drží sa príslovia „Keep it simple“. Gedit umožňuje prácu so súbormi na lokálnych diskoch, ale umožňuje prácu aj s vzdialenými súbormi. Medzi jeho základné funkcie patrí napr. zvýrazňovanie syntaxe kódu, automatické odsadzovanie, podpora vyhľadávania a nahradzovania textu, zobrazovania čísiel riadkov, pokročilé zvýrazňovanie, podpora kontroly pravopisu, práca so záložkami, automatické ukladanie otvoreného dokumentu v určitých časových intervaloch atd. Gedit umožňuje zobraziť štatistiku aktuálneho dokumentu, ktorá zobrazuje počet riadkov, slov alebo znakov v dokumente. [10]

2.3.3 PSPad

PSPad patrí k obľúbeným nástrojom mnohých programátorov, lebo sa drží príslovia v jednoduchosti je krása. Je pomerne rýchly, čo je v porovnaní s množstvom funkcií, ktoré ponúka celkom prekvapujúce. Je určený pre každého, kto pracuje s textovými súbormi.
Či už teda tvoríte internetové stránky, programujete bežné aplikácie alebo len potrebujete pohodlne upravovať konfiguračné súbory. Jeho funkcie sa Vám určite zídu. Medzi funkcie, ktoré používam najčastejšie patria [8]:

- **Zvýrazňovanie syntaxu** Ide pre nás prakticky o najdôležitejšiu funkciu. Programátorom napomáha obrovským spôsobom pri orientácii v kóde. PSPad vie zvýrazňovať v celej rade programovacích jazykov.
- **Vyhľadaj a nahraď** – Už z názvu je jasné čo funkcia robí, samozrejme aj tu nájdete nejaké tie vylepšenia.
- **Kódovanie** – Dokáže vytvárať dokumenty v kódovaniach Win1250, Kamenických, Latin II, ISO 8859-2, UTF8 a UNICODE. Určite sa mnoho z Vás stretlo s tým, že dokument sa zobrazoval nejako čudne, pritom stačilo zmeniť kódovanie a už to zrazu bolo správne.
- **Automatické dokončovanie** – Pracuje s pojmami, ktoré už dokument obsahuje. Výborné pre opakujúce sa slová ako napríklad tagy.
- **Kontrola pravopisu** – Podobne, ako vo Worde pomocou klávesy F7.
- **Prieskumník kódu** – Zobrazuje akýsi "obsah" práve upravovaného súboru – fintou je však to, že používa šikovné rozbaľovanie oblasti, ktoré robia obsah veľmi prehľadným. Prieskumníka môžeme teda využiť v navigácii súborom, ale aj na skúmanie jeho obsahu.

2.4 Existujúce programové systémy

V tejto časti sú opísané niektoré známe programové systémy slúžiace na návrh digitálnych systémov. Žiadny z týchto systémov však nie je univerzálny ale slúži často len pre jeden účel a podporuje len málo súborových štandardov. Nachádza sa tu opis rôznych používaných systémov, ktoré by nám mali pomôcť pri riešení daného projektu.

2.4.1 Blif2vhdl

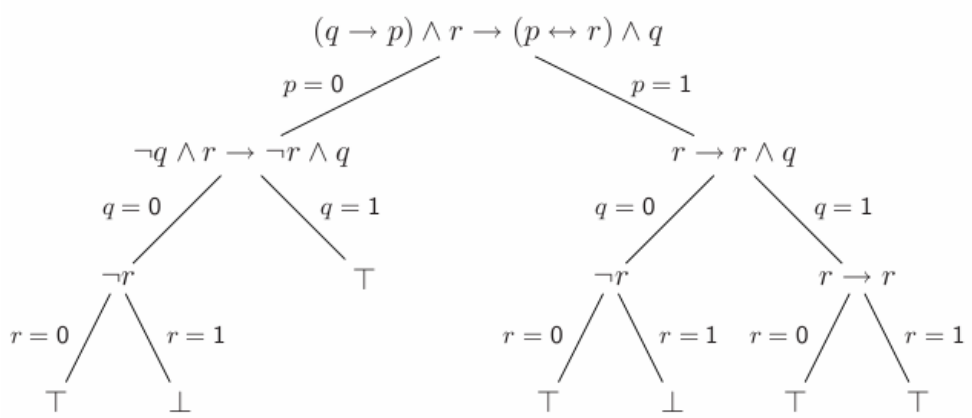
Blif2vhdl je opensource program vytvorený v jazyku Perl, ktorý vznikol na University of California at Berkeley. Neobsahuje grafické rozhranie, čiže pracuje len v termináli. Blif2vhdl prijíma na vstupe jeden vstupný súbor vo formáte BLIF. Následne tento súbor .BLIF transformuje do vhdl jazyka a uloží ho do aktuálneho adresára s koncovkou vhdl.

2.4.2 BDS

BDS systém je logický optimalizačný systém, ktorý je založený na dekompozičných technikách binárnych rozhodovacích diagramov (ďalej BDD), ktoré podporujú dekompozičné štruktúry AND, OR, XOR a MUX. Takáto metóda dekompozície pomocou BDD je veľmi efektívna pri syntéze AND/OR a XOR logických funkcií. [5]
Pri analýze BDS systému najskôr popíšeme BDD, ich tvorbu a vlastnosti. Priblížime princíp dekompozície logickej funkcie na základe BDD a spôsob akým BDS túto dekompozíciu realizuje.

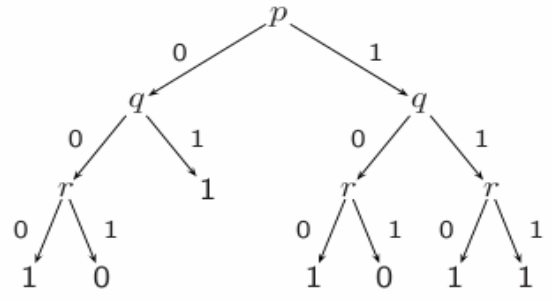
2.4.2.1 Binárne rozhodovacie diagramy

BDD vzniká spôsobom, ktorý si ukážeme na konkrétnej logickej funkcii. Majme funkciu:
$f = (q \rightarrow p) \wedge r \rightarrow (p \leftrightarrow r) \wedge q$
Spôsob, akým z danej funkcie vytvoríme BDD je nasledovný [6]:
Vytvoríme binárny strom pre výpočet pravdivostných hodnôt funkcie (obr.4).



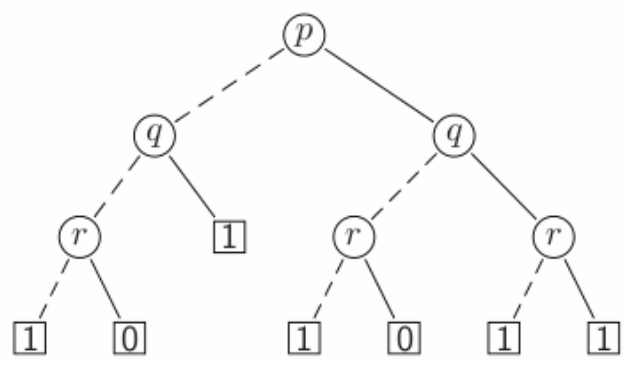
Obr.4 Binárny strom pre výpočet pravdivostných hodnôt [6]

Výmenou symbolov $\bot$ a $\top$ za 0 a 1, výmenou funkcií v interných uzloch za premennú použítú pri rozdelovaní v tomto uzle a označením šípiek symbolmi 0 a 1 dostávame binárny rozhodovací strom (obr.5).



Obr.5 Binárny rozhodovací strom [6]

Takýto binárny rozhodovací strom môžeme reprezentovať ešte v kompaktnejšej forme výmenou šípiek s označením 0 za prerušovanú čiaru a šípiek s označením 1 za plnú čiaru. Listy tohto stromu budú vždy označené 0 alebo 1 a vnútorné uzly budú predstavovať test premennej na jej hodnotu (Obr.6).

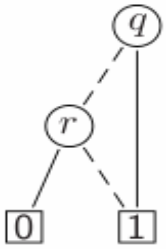


Obr.6 Kompaktná reprezentácia binárneho stromu [6]

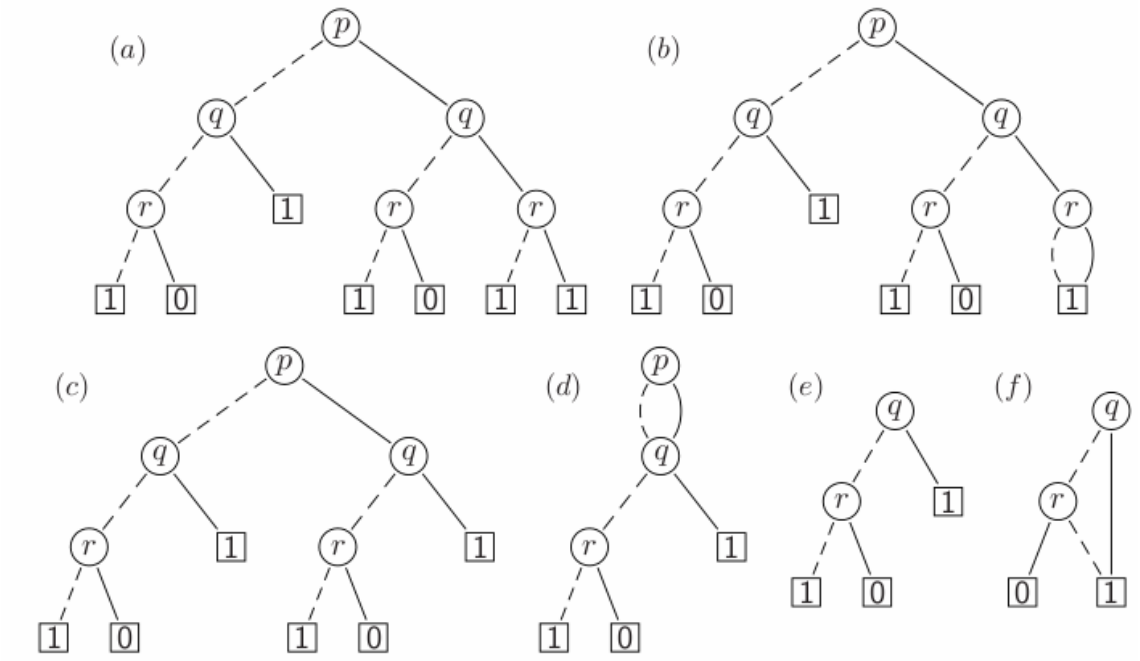
BDD z binárneho rozhodovacieho stromu vznikne nasledujúcimi transformáciami:
Eliminácia redundantných testov: Ak existuje uzol, ktorého pravý a ľavý podstrom je ten istý podstrom, takýto uzol môžeme odstrániť. Spojenie izomorfných podstromov: Ak dva podstromy, ktoré majú korene každý v inom uzle sú izomorfné, môžeme ich spojiť do jedného.

Z uvedenej transformácie vyplývajú nasledovné vlastnosti BDD:
Pravý a ľavý podstrom každého uzla sú rôzne.
Každý pár podstromov, ktoré majú korene každý v inom uzle nie sú izomorfné.

Aplikáciou transformácií na binárny rozhodovací strom z bodu 3 dostávame binárny rozhodovací diagram (obr.7). Znázornenie použitých transformácií je na obr.8.



Obr.7 Binárny rozhodovací diagram [6]



Obr.8 Použité transformácie [6]

Detailný opis transformácií z obr.8:

Z bodu (a) do (b): Spojenie izomorfných podstromov v uzle r.

Z bodu (b) do (c): Eliminácia redundantného testu v uzle r.

Z bodu (c) do (d): Spojenie izomorfných podstromov v uzle p.

Z bodu (d) do (e): Eliminácia redundantného testu v uzle p.

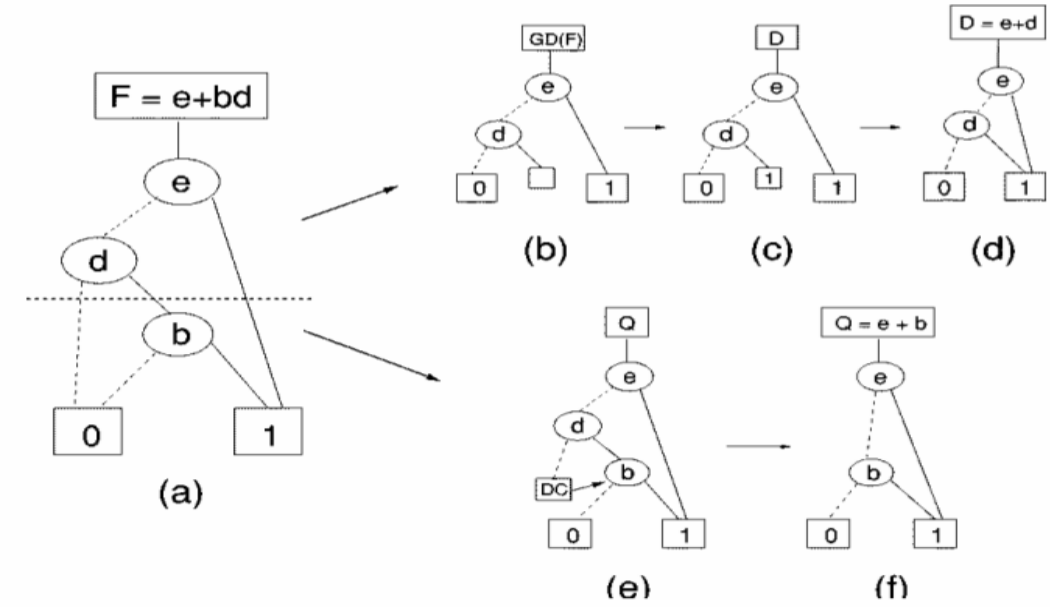
Z bodu (e) do (f): Spojenie izomorfných podstromov v uzle 1.

Dekompozícia BDD

V nasledujúcej časti sa bližšie pozrieme na konjuktívnu a disjunktívnu dekompozíciu BDD. Princíp týchto dekompozícií si ukážeme na príklade. Pre pochopenie dekompozície je nutné definovať rez cez BDD. Rez je množina hrán v BDD, ktoré rozdeľujú ich uzly do dvoch disjunktných množín D a (V – D) tak, že koreň BDD patrí do množiny D a terminály 0 a 1 patria do množiny (V – D). V nasledujúcej časti bližšie opíšeme konjuktívnu a disjunktívnu dekompozíciu.

Konjuktívna dekompozícia

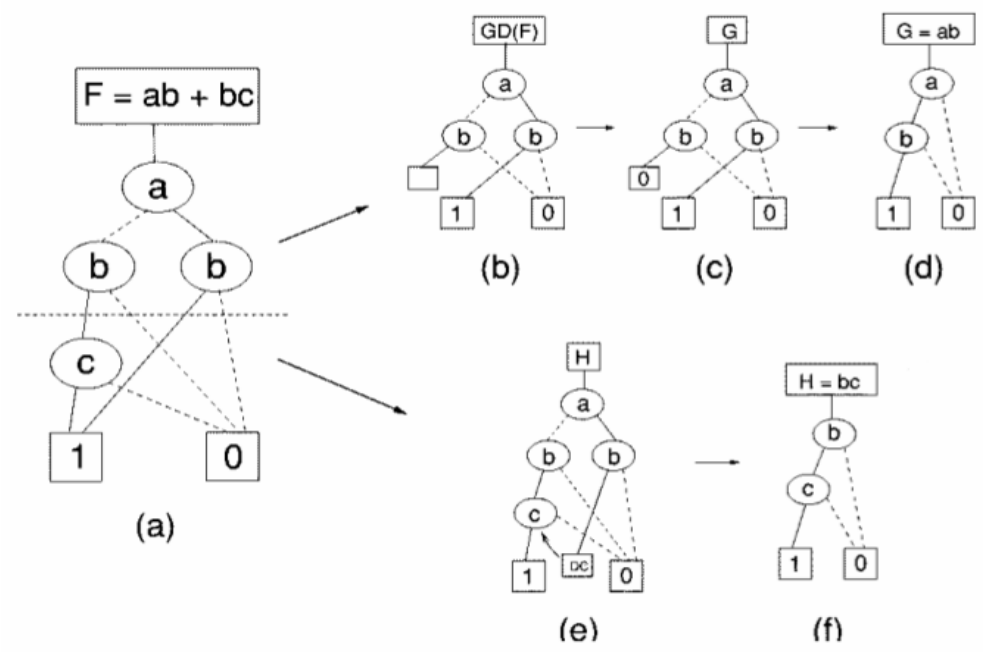
Z definície vyplýva, že boolovská funkcia F má konjuktívnu dekompozíciu, ak môže byť reprezentovaná pomocou F = D . Q kde D je boolovský deliteľ a Q je kvocient F v takejto dekompozícii. Majme rez, ktorý rozdelí BDD na množiny D a (V – VD). Časť grafu patriacej do množiny VD bude tvoriť nový graf, kde hrana e je spojená s 0 resp. 1 ak bola v pôvodnom grafe spojená s 0 resp. s 1. Všetky interné hrany ostanú voľné. Takýto nový graf sa nazýva „Generalized Dominator“ funkcie F (GD(F)). Boolovský deliteľ D je reprezentovaný grafom, ktorý vznikne z GD(F) presmerovaním voľných hrán do 1. Kvocient Q funkcie F je reprezentovaný grafom, ktorý vznikne presmerovaním všetkých hrán v F spojených s 0 a patriacich zároveň do D do nových uzlov, ktoré reprezentujú fakt, že hodnota funkcie F v týchto uzloch nie je definovaná (DC alebo „Don't Care“ uzly). Pomocou transformácií, ktoré sme si uviedli pri opise BDD dostávame dva BDD (D a Q), ktoré reprezentujú dekomponovanú funkciu F [5]. Príklad konjuktívnej dekompozície funkcie F = e + bd je znázornený na obr.9.



Obr.9 Príklad konjuktívnej dekompozície [5]

Disjunktívna dekompozícia

Z definície vyplýva, že boolovská funkcia F má disjunktívnu dekompozíciu, ak môže byť reprezentovaná pomocou F = G + H. Majme rez s rovnakými vlastnosťami ako pri konjuktívnej dekompozícii a vytvorme GD(F) postupom, uvedeným pri konjuktívnej dekompozícii.
Boolovský term G je reprezentovaný grafom, ktorý vznikne z GD(F) presmerovaním voľných hrán do 0. Term H funkcie F je reprezentovaný grafom, ktorý vznikne presmerovaním všetkých hrán v F spojených s 1 a patriacich zároveň do G do DC uzlov. Pomocou transformácií, ktoré sme si uviedli pri opise BDD dostávame dva BDD (G a H), ktoré reprezentujú dekomponovanú funkciu F [5]. Príklad konjuktívnej dekompozície funkcie F = ab + bc je znázornený na obr.10.



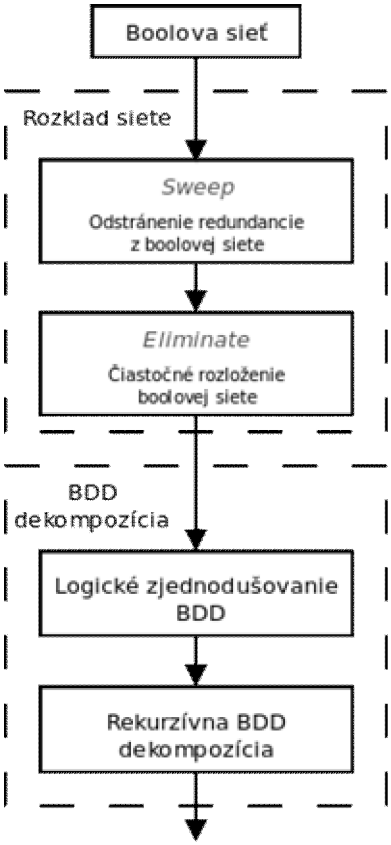
Obr.10 Príklad disjunktívnej dekompozície [5]

Realizácia dekompozície pomocou BDS

Pri dekompozícií vychádzame z danej boolovskej siete. Prvým krokom je odstránenie redundancií v boolovskej sieti. Pri tomto kroku sa nerealizuje žiadna logická dekompozícia, ale je dôležitý v príprave boolovskej siete na následnú dekompozíciu. Spolu s odstraňovaním konštánt sú v tomto kroku identifikované funkcionálne ekvivalentné uzly, ktoré sú následne z boolovskej siete odstránené. Táto procedúra výrazne znižuje zložitosť dekompozície.

Aplikovať logickú dekompozíciu na celú boolovskú sieť pomocou globálnej BDD reprezentácie nemusí byť praktické pri rozsiahlych systémoch. Preto v ďalšom kroku sa boolovská sieť čiastočne rozloží na množinu super-uzlov. Každý super-uzol potom môže byť reprezentovaný ako lokálny BDD a následne dekomponovaný.

Pri samotnej dekompozícii sa BDD najskôr zjednoduší pomocou preusporiadania premenných. Takto vznikne usporiadaný BDD, ktorý je následne rekurzívne dekomponovaný na menšie celky. Proces dekompozície je znázomený na obr.11.



Obr.11 Proces dekompozície pomocou BDS [5]

2.4.2.2 Implementácia systému BDS

Program BDS bol implementovaný v programovacom jazyku C. Vstupom do programu je kombinačný logický obvod zapísaný vo formáte BLIF. BDS z tohto kombinačného obvodu vytvorí boolovskú sieť, ktorá sa následne podľa opísaného procesu rozloží, zjednoduší a dekomponuje. Program môže byť spustený s rôznymi prepínačmi pre nastavenie parametrov pri dekompozícii. [11]

2.4.3 BDS-PGA

BDS-PGA je open source program vytvorený v jazyku C. Je to vylepšená verzia programu BDS, ktorý vznikol na University of Massachusetts Amherst. BDS-PGA pre správnu funkcionalitu potrebuje funkcie balíka Cuddy, ktorý vznikol na University of Colorado Boulder. Tento program robí syntézu a optimalizáciu pre FPGA, ktoré sú založené na LUT.
BDS-PGA prijíma na vstupe súbor vo formáte BLIF, ktorý následne transformuje, do niekoľkých výstupných súborov ako sú: cktmeno.final.eqn, cktmeno.final.dot a cktmeno.final.blif. Všetky štatistické informácie ukladá do súboru BDS.run.

2.4.4 MVSIS 2.0

Je to open source, ktorý vznikol na University of California at Berkeley. Neobsahuje grafické rozhranie, čiže pracuje len v termináli. Aplikácia MVSIS 2.0 nahradzuje staršiu verziu SIS a pridáva nové možnosti viacúrovňovej manipulácie. Podporuje údajové štruktúry a procedúry potrebné na technologicky nezávislú MV (Multi-valued) logickú syntézu. Špecializuje sa na optimalizačné algoritmy, ktoré zdokonaľujú kvalitu logických obvodov, ktoré sú vytvárané automatickými syntéznymi nástrojmi. Hlavnými cieľmi MVSIS bolo podstatné zrýchlenie spracovania logických úloh a generovanie čistejšieho kódu. Tento cieľ MVSIS splnil, lebo je podstatne rýchlejší ako jeho predchodca SIS, a pri vykonávaní spotrebúva menšie množstvo pamäte. [13]

MVSIS môže zahŕňať niekoľko aplikácii:

- Logickú syntézu pre multihodnotové hardvérové zariadenia
- Multihodnotový spôsob zápisu je prirodzenejší spôsob opísania procedúr na vyššej úrovni.
- Asynchrónnu syntézu

Je to logický synteticky program ktorý podporuje dátové štruktúry a procedúry potrebné pre technológie nezávislé na binárnych a multihodnotových logických syntézach. Súčasná implementácia je doplnená o nové funkcie, ale zároveň jej používanie je podobne používaniu systémov SIS. Aplikácia podporuje vstupné formáty BLIF a PLA, umožňuje kvalitnú a veľmi rýchlu dekompozíciu. [13] Po spustení MVSIS máme na výber niekoľko typov príkazov, ktoré môžeme rozdeliť do nasledovných podskupín: [14]

Základne príkazy

alias echo help history quit set snatch source time unalias undo unset usage

Príkazy na úpravu a zmenu názvu

chng_name rename reset_name

Príkazy na zobrazenie daného súboru

print

print_domi print_factor print_io print_level print_nd print_range print_spec print_stats print_value

Príkazy na načítanie súboru

read_blif *-načíta BLIF súbor*

read_blif_mv

read_blif_mvs

read_pla *-načíta PLA súbor*

Príkazy pre syntézu

club collapse decomp eliminate encode fullsimp fxu merge
mfs pair_decode reset_default resub simplify strash sweep

Iné príkazy

default dize free reorder window

Príkazy pre verifikáciu

verify

Príkazy pre zápis

write_blif *-vytvorí súbor BLIF*
write_blif_mv
write_blif_mvs
write_pla *-vytvorí súbor PLA*

2.4.5 Vzájomné porovnanie

Analyzované softvérové produkty následne porovnávame z hľadiska ich funkčnosti a vlastností. Systém Blif2vhdl slúži na transformáciu súborov BLIF na formát VHDL. BDS systém je logický systém používajúci metódu dekompozície logického obvodu pomocou BDD a je veľmi efektívny pri syntéze AND/OR a XOR logických funkcií. Vstupom do programu je kombinačný logický obvod zapísaný vo formáte BLIF. BDS z tohto kombinačného obvodu vytvorí boolovskú sieť, ktorá sa následne podľa opísaného procesu rozloží, zjednoduší a dekomponuje. Výstup z programu po logickej dekompozícií je zapísaný do BLIF formátu, DOT a rovnicovej reprezentácie. Systém BDS-PGA je vlastne len rozšírená verzia už spomínaného systému BDS. Najlepší systém pre návrh digitálnych systémov je existujúci MVSIS 2.0, ktorý je rozšírením staršieho SIS. Ide o interaktívny nástroj pre syntézu a optimalizáciu sekvenčných a kombinačných logických obvodov.

Podporuje rôzne súborové štandardy ako PLA, KISS, BLIF a EQN a umožnuje ich vzájomné konverzie. Taktiež umožňuje kvalitnú a veľmi rýchlu dekompozíciu obvodov a veľa ďalších možností spracovania logických obvodov.

2.5 Existujúce návrhové systémy

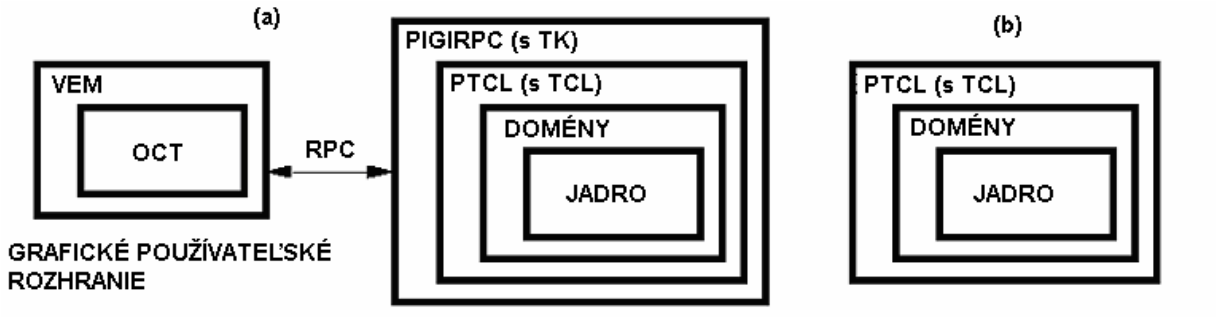
Najznámejšie návrhové systémy sú určené pre modelovanie, simuláciu a návrh rozličných vnorených systémov. Analyzovali sme nasledujúce: Ptolemy II, Bluespec, Metropolis, ForSyDe a SML-sys.

2.5.1 Ptolemy II

Ptolemy II je softvérový systém, ktorý sa zaoberá heterogenným modelovaním, simuláciou a dizajnom súbežných systémov. Tento systém je zameraný hlavne na modelovanie vnorených systémov, obzvlášť na tie, ktoré zmiešavajú rozličné technológie, vrátanie analógovej a digitálnej elektroniky, hardvéru a softvéru či elektrických a mechanických zariadení. Je to otvorený systém zameraný na tzv. aktor-orientované modelovanie (actor-orientated modeling). Aktory sú softvérové komponenty, ktoré sa vykonávajú súbežne a komunikujú navzájom pomocou správ cez prepojené porty. Pričom hierarchické prepojenie týchto aktorov, môžeme nazývať model. V Ptolemy II nie je semtantika modelov určovaná samotným jadrom systému, ale tzv. riadiacimi softvérovými komponentmi (director) v modeli. Tieto riadiace komponenty implementujú samotný výpočtový model. [15]

Približná organizácia Ptolemy II systému

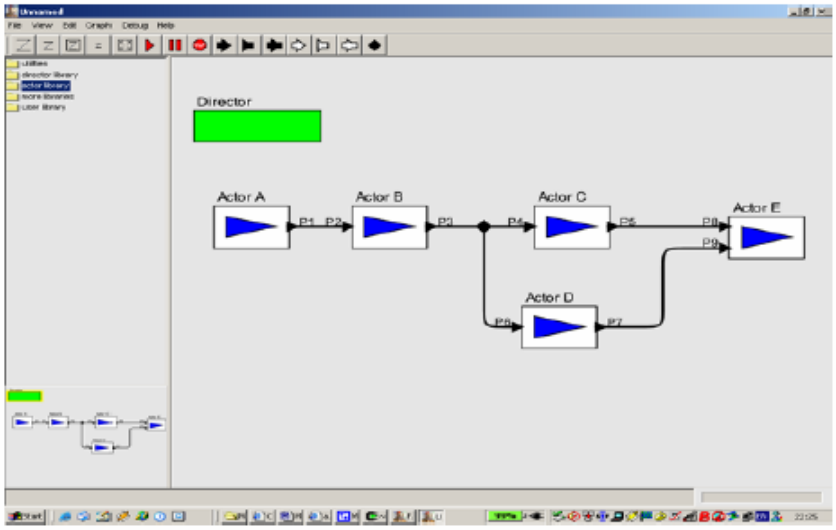
Na Obr.12 je možné vidieť približnú organizáciu Ptolemy II systému. Typické používanie Ptolemy II spúšťa dva procesy (Obr.12 a). Pri tomto spôsobe spúšťania je jeden proces určený na grafické rozhranie a druhý proces vykonáva samotný systém. Druhá alternatíva je spustiť Ptolemy II bez grafického rozhrania (Obr.12 b) ako jediný proces. V tomto prípade sa spustí Ptolemy ako jeden proces s textovým iterpreterom TCL (Tool Command Language).



Obr.12 a)organizácia Ptolemy II s grafickým rozhraním, b)bez grafického rozhrania [16]

Vergil

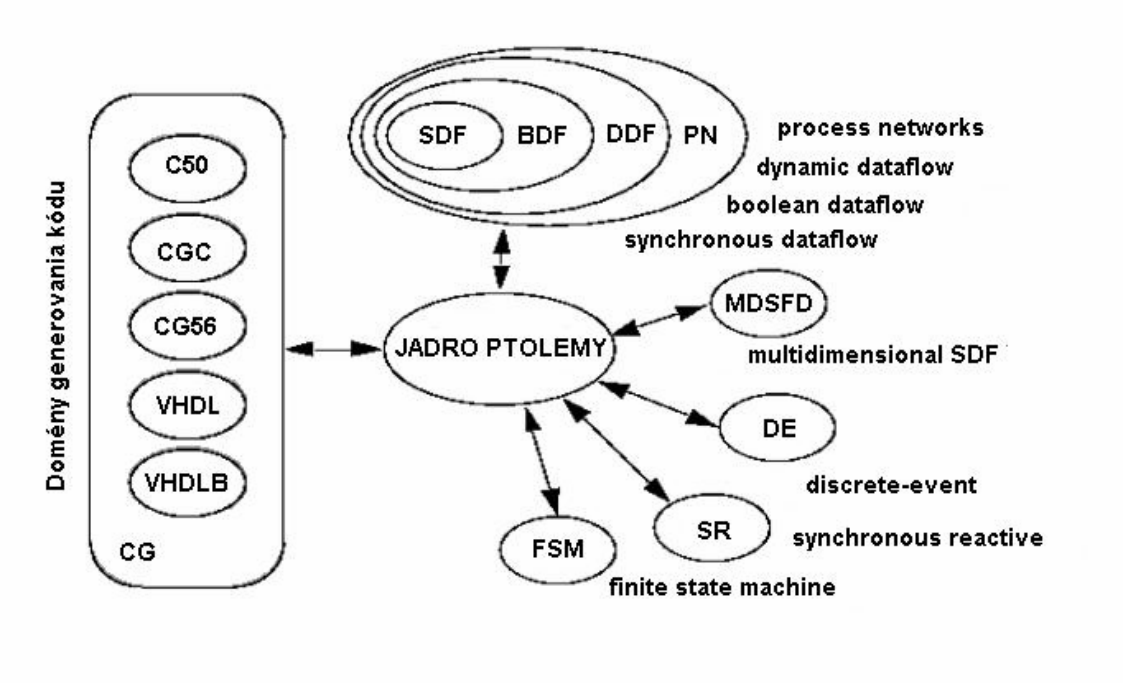
Modely v Ptolemy II môžu byť modelované v grafickom používateľskom rozhraní nazývanom Vergil (Obr.13). Aktory sú reprezentované blokmi, tieto poskytujú funkciu alebo množinu funkcií, ktoré mapujú stav a vstupy na výstup. Aktor pri spustení dostane dáta na svoje vstupné porty a produkuje dáta na svoje výstupne porty. Aktory, ktoré majú iba výstupné porty nazývame zdrojové a na druhej strane aktory, ktoré majú iba vstupné porty, nazývame cieľové. Prepojením portov jednotlivých aktorov vznikne komunikačný kanál. Na Obr.13 je napríklad zdrojový aktor A, ktorý produkuje dáta na porte P1 a posiela ich na vstupný port aktoru B. Ten ich spracuje, vykoná výpočet alebo svoju funkcionalitu a pošle dáta výstupným portom P3 na ďalšie spracovanie. [17]



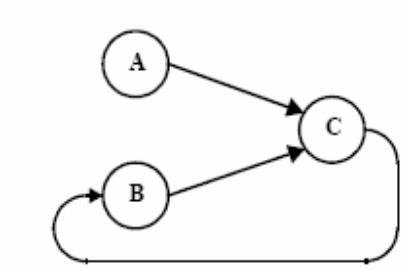
Obr.13 Vergil grafické rozhranie PtolemyII

Výpočtové modely

Výpočtový model poskytuje množinu pravidiel, ktoré špecifikujú spúšťanie a komunikáciu jednotlivých aktorov. Definuje poradie ich spúšťania, typ komunikácie medzi jednotlivými portami v čase. Hlavný princíp Ptolemy II systému je to, že výber výpočtového modelu silno vplýva na kvalitu navrhovaného systému. Zámerom tohto systému je podporiť konštrukciu a súčinnosť spustiteľných modelov, ktoré sú modelované rôznymi výpočtovými modelmi. V Ptolemy II sú výpočtové modely implementované ako domény. Vykonanie aktoru v rámci domény je riadené tzv. riadiacimi softvérovými komponentmi (director). V Ptolemy II je komplexný systém špecifikovaný ako hierarchická kompozícia jednoduchších podsystémov. Každý systém môže byť modelovaný rôznou doménou. Domény rozdeľujeme do dvoch skupín a to simulácie alebo generovanie kódu. Simulačné domény sú prekladače, ktoré spúšťajú špecifikáciu systému na počítači. Domény generovania kódu prekladajú špecifikáciu systému do nejakého jazyka ako napr. C++ alebo VHDL. Ptolemy II modely sú grafy (Obr.15), v ktorých uzly vyjadrujú entity a hrany vzťahy medzi entitami. Vo väčšine doménach sú entity aktory a vzťahy medzi nimi vyjadrujú ich vzájomnú komunikáciu. Ptolemy II výpočtové modely sú realizované nasledujúcimi doménami (Obr.14) (len niektoré z nich). [15]



Obr.14 Domény v Ptolemy II



Obr.15 Ptolemy II modely

Grafy v Ptolemy II

Ptolemy II kernel poskytuje veľkú infraštruktúru na tvorbu a manipuláciu s grafmi. Balík pre grafy poskytuje základnú infraštruktúru pre orientované, neorientované a acyklické grafy. Nad týmito grafmi sú definované rôzne algoritmy, ktoré operujú s nimi. Dátová štruktúra pre uchovávanie grafov a algoritmov operujúcich s nimi je popísaná v niekoľkých triedach balíku pre grafy (Obr.16). [16]

Najvýznamnejšie triedy:

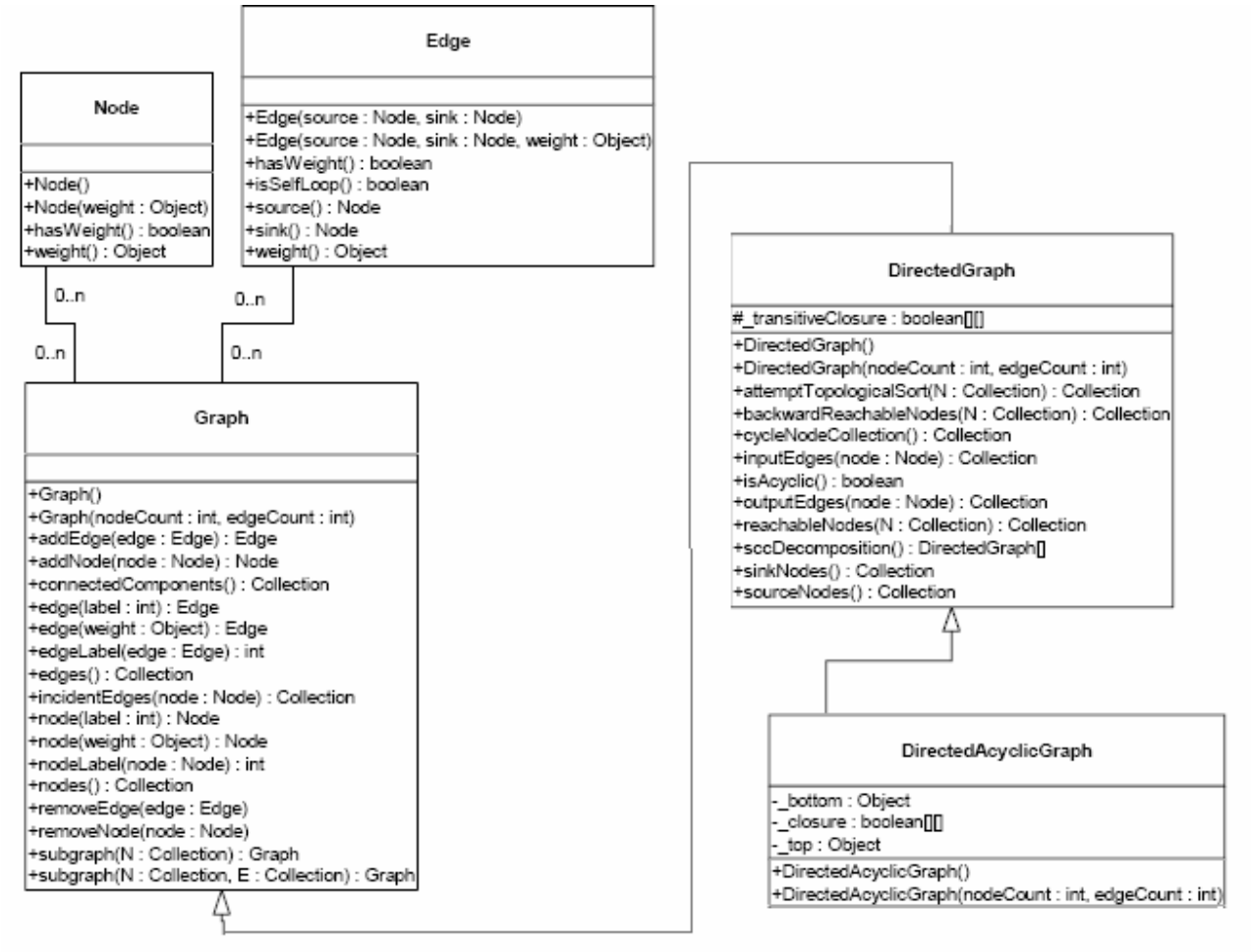
Egde: hrana

Node: uzol

Graph: samotný graf

DirectedGraph: rozšírenie triedy graph o orientované grafy

DirectedAcyclicGraph: rozšírenie triedy DirectedGraph o acyklické orientované grafy



Obr.16 Balík pre grafy

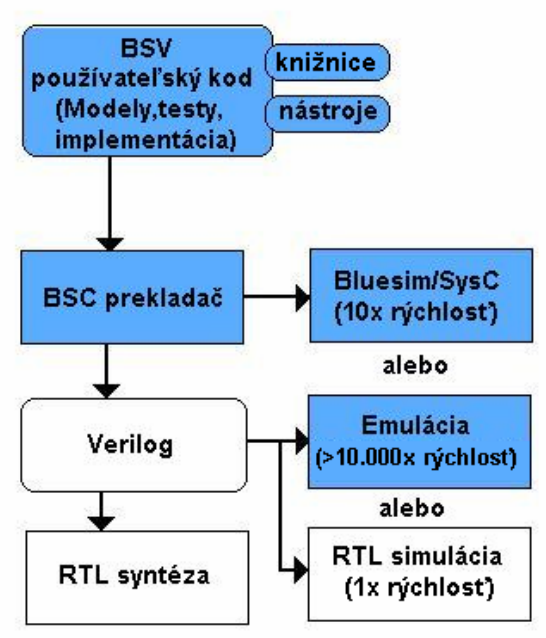
2.5.2 Bluespec

Bluespec je spoločnosť, ktorá sa zaoberá návrhom a implementáciou hardvérových a softvérových komponentov. Pre tento účel vyvinula prostredie, ktoré napomáha pri návrhu a implementácií hardvérových a softvérových komponentov. Bluespec ako jediný poskytuje rozhranie medzi návrhom modelu a jeho RTL implementáciou. Bluespec modely majú možnosť postupného odlaďovania chýb, môžu byť plne implementovatelné a umožňujú vysokorýchlostnú emuláciu vo všetkých štádiách vývoja. Vývoj pomocou Bluespec ponúka vyšší stupeň abstrakcie ako prostredie SystemC1, od ktorého je toto prostredie odvodené. [18]

Bluespec prosredie v sebe zahŕňa:

- Bluespec kompilátor (BSC) – kompiluje vysoko úrovňový model, hardvérovo-softvérové rozhranie, testy a implementáciu do VHDL alebo RTL.
- Bluespec simulátor (Bluesim) – simuluje Bluespec návrhy 5 až 20-krát rýchlejšie ako RTL.
- Bluespec Developement Workstation (BDW) – vysoko úrovňové grafické prostredie pre analýzu, modelovanie a návrh Bleuespec schém.
- AzureIP Foundation Library
- Emulačnú infraštruktúru

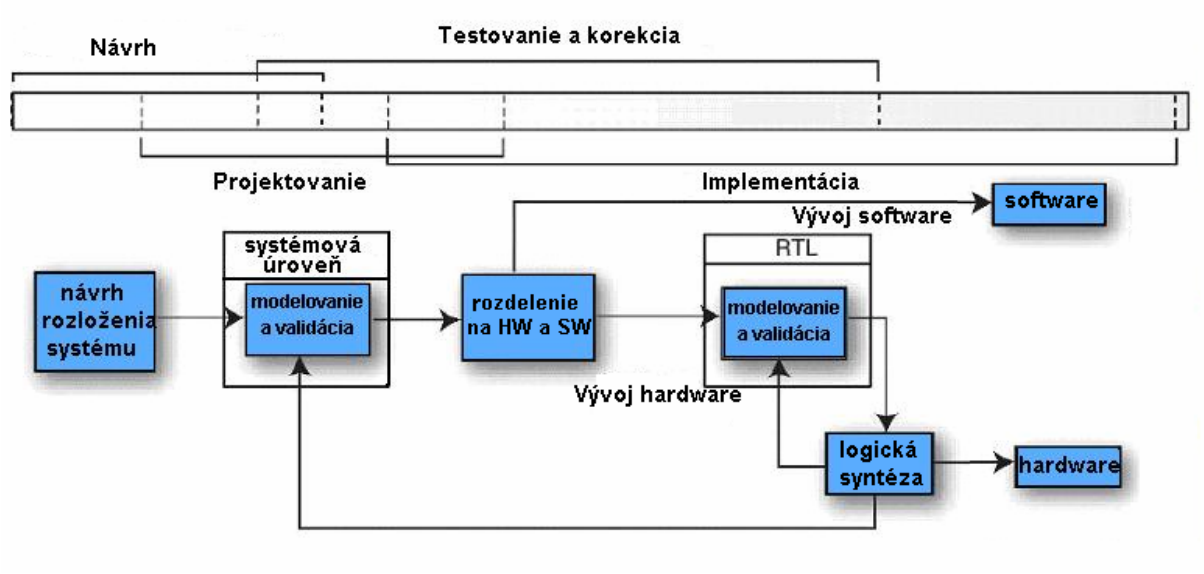
[1] Viac o jazylu SystemC na stránke http://www.doulos.com/knowhow/systemc/tutorial/.



Obr.17 Schéma prostredia Bluespec [18]

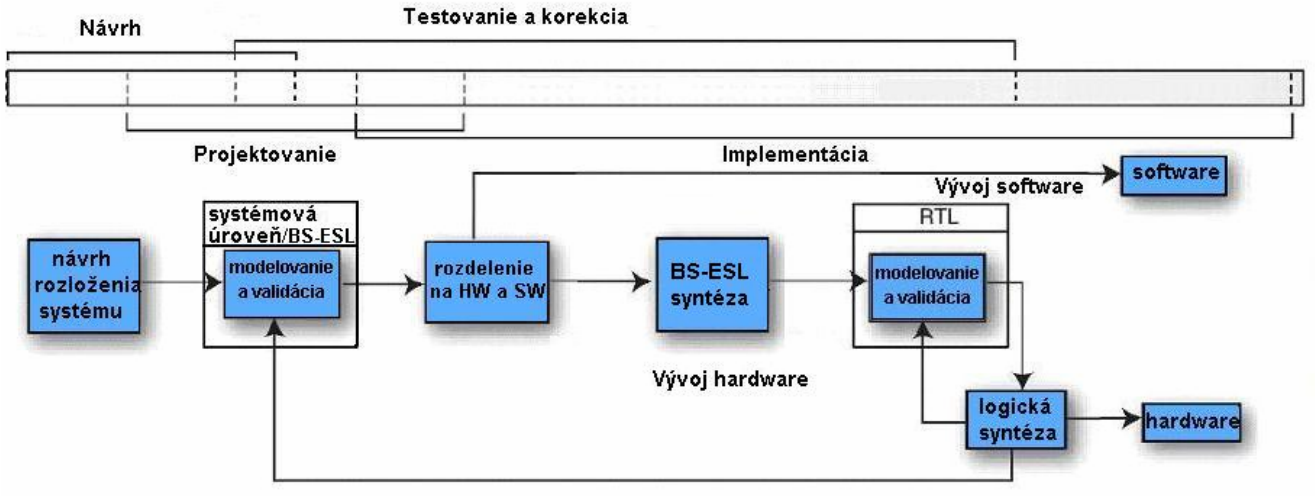
Bluespec – BS-ESL

BS-ESL rozšírenie zahŕňa syntaktické konštrukcie pre moduly, rozhrania, pravidlá a metódy. V jazykoch C++ a SystemC sú tieto rozšírenia definované ako makrá v hlavičkovom súbore bsystemc.h. Tieto makrá boli expandované do volaní funkcií, ktoré sú výkonávaná jadrom podľa zadefinovaných kritérií.



Obr.18 Proces návrhu a implemenácie obvodov pomocou jazyka SystemC

Obrázok Obr.18 znázorňuje proces od návrhu cez rozdelenie komponentov na hardvér a softvér až po ich implementáciu. Do tohoto procesu je vsunutý BS-ESL syntéza pre modelovanie a validáciu hardvérových komponentov. Rozšírený proces je znázornený na Obr.19.



Obr.19 Proces návrhu a implemenácie rozšírený o BS-ESL

BS-ESL moduly

Tieto moduly sú oddedené od modulov definovaných v jazyku SystemC, z čoho vyplýva, že koncept modulov v BS-ESL je rovnaký ako v jazyku SystemC. Každý modul musí obsahovať konštruktor a môže tiež obsahovať iné moduly, implementáciu rozhraní alebo metód, deklaráciu premenných alebo rôzne pravidlá. Základná koncepcia modulu je znázornená nižsie.

```
ESL_MODULE (name, prov_if)
ESL_CTOR(name, args ...)
ESL_END_CTOR
```

Tento modul obsahuje dva argumenty, a to name argument a prove_if argument. Argument name obsahuje názov modulu a argument prove_if zodpovedá definovanému rozhraniu. Každý modul musí obsahovať konštruktor, ktorý je definovaný na obrázku. Konštruktor je definovaný dvoma makrami, a to ESL_CTOR(), čo označuje začiatok konštruktora a ESL_END_CTOR(), čo označuje jeho koniec. Makro ESL_CTOR() môže mať rôzny počet argumentov, avšak prvý argument musí byť meno totožné s menom modulu.

BS-ESL pravidlá

Ďalším so základných stavebných prvkov BS-ESL sú tzv. pravildá (rules). Pravidlá sú deklarované ako privátne funkcie tried, do ktorých daný modul spadá. Každé pravidlo obsahuje v rámci modulu svoje jedinečné meno a tzv. Strážcu (guard). Strážcovia sú vyhodnocovaní ako booleovské funkcie a na základe ich hodnoty je (alebo) spúšťaná príslušná akcia2. Príklad zadefinovania strážcu je znázornený nižsie.

```
/*  Rule definition */
ESL_RULE (rule_name, rule_guard){
   // rule_action
};
```

[2] Akcia je spustená len v prípade, že je hodnota strážcu vyhodnotená ako TRUE.

BS-ESL metódy

Bluespec používa tri rôzne druhy metód. Sú nimi:

- Akčné metódy – action methods
- Hodnotové metódy – value methods
- Hybridné metódy3 – actionvalue methods

Ich rozdiel spočíva v tom, že hodnotové metódy len vracajú návratovú hodnotu bez akjkoľvek informácie o stave modulu, akčné metódy menia stav modulu a hybridné metódy menia stav modulu a zároveň vracajú hodnotu.

V C++ všetky metódy môžu meniť svoj stav aj návratovú hodnotu, ale v BS-ESL dodržiavanie vzťahov medzi rôznými typmi metód zabezpečuje BS-ESL preprocesor. Všetky tri makrá definujúce metódy sú makrá s premenlivým počtom parametrov. Všetky typy však musia obsahovať ako prvý parameter meno metódy, za ním nasleduje strážca, ktorého BS-ESL používa na plánovanie účely, pri hodnotových metódach musí nasledovať typ návratovej hodnoty a potom nasleduje ľubovolný počet argumentov. Definícia metód je znázornená nižšie.

```
ESL_METHOD_ACTION (name, guard, args ...)
{
 //...
}
ESL_METHOD_ACTION (name, guard, args ...)
{
 //...
}
ESL_METHOD_VALUE (name, guard, return_type, args
...)
{
 //...
}
```

[3] Označenie „hybridné metódy“ nie je štandardný názov pre tento typ metód, ale je najcharakteristickejší.

Rozhrania

Koncept rozhraní je v Bluespec odlišný od konceptu v C++ alebo SystemC. V tomto prípade musí každý modul implementovať nejaké rozhranie. Existujú aj prípady, v ktorých implementácia rozhrania nie je potrebná, v takom prípade sa implementuje špeciálne rozhranie typu ESL_EMPTY4. BS-ESL rozhranie môže obsahovať všetky tri typy metód. Deklarácia takéhoto rozhrania je znázornená nižsie.

```
ESL_INTERFACE ( if_name){
  ESL_METHOD_ACTION_INTERFACE ( name [, agrument list...]);
  ESL_METHOD_VALUE_INTERFACE (return_type, name [, argument
             list]);
  ESL_METHOD_ACTIONVALUE_INTERFACE (return_type, name [,
             argument list]);
};
```

Makro ESL_INTERFACE obsahuje len jeden argument, a to meno rozhrania. Metódy, ktoré dané rozhranie implementuje sú definované menom, v prípade hodnotových metód alebo hybridných metód aj návratovým typom a následne počtom argumentov.

Vykonávanie BS-ESL kódu

Vykonávanie programu je obdobné spúšťaniu programu SystemC. Vo funkcií sc_main() sú zadeklarované požiadavky, z ktorých sa vyrvorí hierarchia modulov. Akonáhle je táto hierarchia vytvorená, konštruktory vykonajú určené zápisy pomovou BS-ESL runtime systému. Následne funkcia sc_start() začne vykonávať program. BS-ESL rutime systém je knižnica, ktorá je nalinkovaná na štandardnú SystemC knižnicu. Knižnica pozostáva z plánovačov pravidiel (rule scheduler) a vykonávača týchto pravidiel (rule execution kernel). Keď sa začne vykonávať program, plánovač spustí analýzu všetkých dostupných pravidiel a na založených metódach rozhraní zistí, ktoré z nich môžu byť vykonávané súčasne v rámci jedného hodinového cyklu.

[4] Rozhranie neobsahujúce žiadne metódy.

Podľa tohoto naplánovania sa následne periodicky vykonávajú všetky pravidlá na základe hodinového signálu. [19]

2.5.3 Metropolis

Systém Metropolis bol vyvinutý za účelom poskytnutia infraštruktúry založenej na modeli s precíznou sémantikou, ktorá je dostatočne všeobecná kvôli podpore existujúcich a nových výpočtových modelov. Takýto metamodel podporuje nielen zachytenie funkcionality navrhovaného systému ale aj analýzu, opis architektúry a mapovanie medzi funkcionálnymi a architekturálnymi elementami.

Jedna z úloh systému metropolis pri navrhovaní systému sa zameriava na interakciu medzi ľuďmi, pracujúcimi na rozdielnych úrovniach abstrakcie modelu a medzi ľuďmi, ktorí pracujú na rovnakej úrovni abstrakcie. Metamodel obsahuje obmedzenia, ktoré hovoria o zatiaľ neimplementovaných požiadavkach. Ďalšou úlohou pri návrhu je analýza a formálna verifikácia na zistenie ako implementácia spĺňa požiadavky. Metropolis podporuje aj syntézu prostredníctvom úrovní abstrakcie použitých pri návrhu.

Výber techník a algoritmov na analýzu a syntézu určitého návrhu závisí na aplikačnej doméne a na fáze návrhu. Preto metropolis neposkytuje algoritmy a nástroje pre všetky možné návrhy ale poskytuje mechanizmus ako uložiť relevantné údaje o návrhu tak, aby vývojári mohli pridávať požadované algoritmy pre danú aplikačnú doménu alebo návrh.

Metropolis zahŕňa parser, ktorý načíta metamodel návrhu a štandardné API rozhranie, pomocou ktorého môžu vývojári zobrazovať, analyzovať a modifikovať dodatočné informácie v rámci daného návrhu. Každý nástroj, ktorý je integrovaný do systému si vytvorí vstupy pomocou tohto rozhrania. Tento unifikovaný mechanizmus umožňuje vkladať nové nástroje, vyvinuté mimo systému metropolis [20].

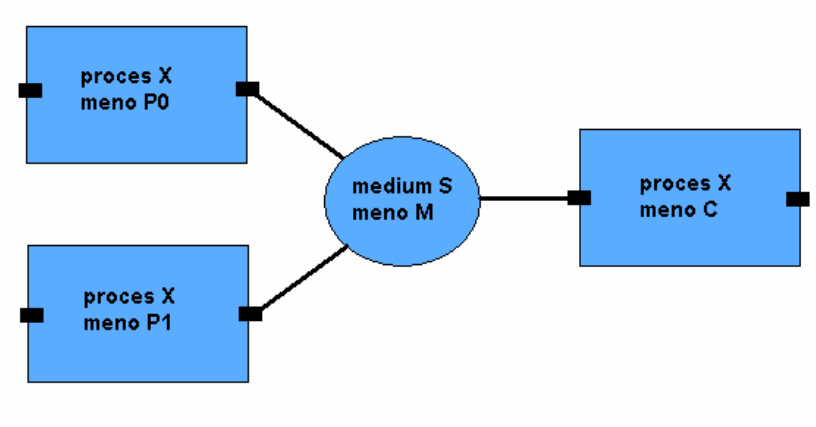
Metamodel

Jazyk metamodelu je použitý na špecifikáciu funkcionality a architektúry systému. Základnými prvkami metamodelu sú procesy, médiá, manažéry kvantity a netlisty. Každý proces obsahuje vlastné vlákno riadenia a vykonáva sa paralelne s ostatnými procesmi v systéme. Vykonávanie procesu je reprezentované sekvenciou udalostí, kde tieto udalosti sú akcie vykonávané procesmi.

Médium je pasívny objekt, ktorý je použitý na komunikáciu medzi procesmi. Každé médium implementuje množinu rozhraní. Médiá sú spojené s procesmi a s inými médiami prostredníctvom portov, ktoré sú kompatibilné s ich rozhraniami.Manažér kvantity kontroluje prístup k zdieľaným médiám alebo prideľuje fyzické kvantity ako čas udalostiam [21].

Funkcionálne modelovanie

Príklad funkcionálneho modelu, zobrazujúci sieť dvoch procesov (producentov) a jedného procesu (konzumenta), ktoré komunikukú prostredníctvom média je na Obr.20.



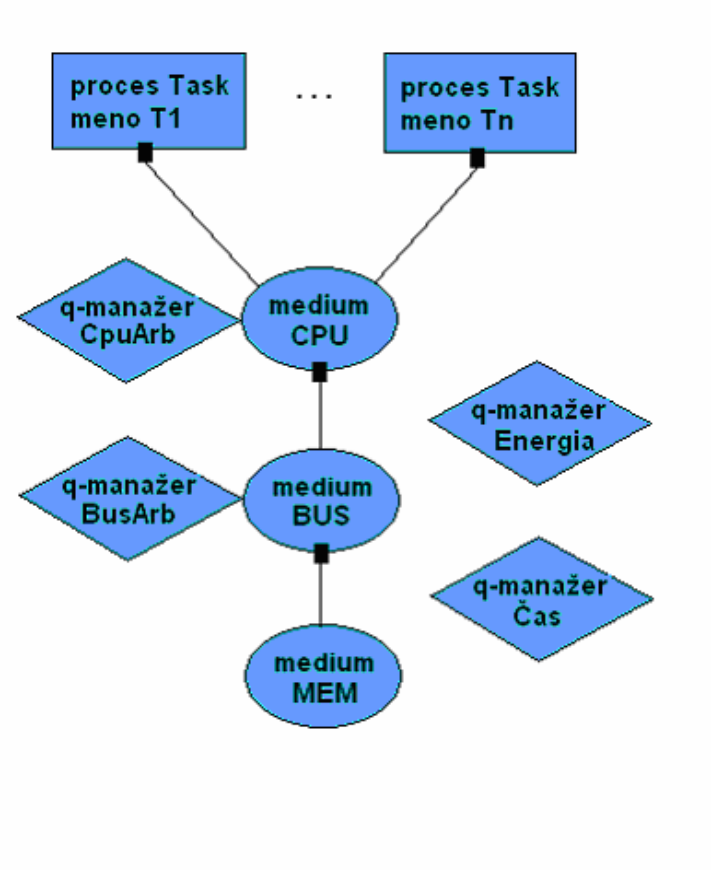
Obr.20 Príklad metamodelu [20]

Po vytvorení siete je potrebné precízne definovať správanie tejto siete pomocou množiny vykonávaní procesov ako sekvenciu udalostí.

Architekturálne modelovanie

Architektúru špecifikujú dva aspekty. Jedným z nich je funkcionalita, ktorú dokážu implementovať a druhým je efektivita danej implementácie. Funkcionalitu modelujeme ako množinu služieb, ktorú architektúra ponúkafunkcionálnemu modelu. Na reprezentovanie efektivity musíme modelovať cenu každej služby. Toto je dosiahnutie dekomponovaním služby na sekvenciu udalostí s hodnotou reprezentujúcou cenu tejto udalosti. Na dekomponovanie služieb používame sieť s procesmi a médiami. Tieto siete často korešpondujú s fyzickou štruktúrou implementačnej platformy [20].

Na Obr.21 je znázornená architektúra pozostávajúca z n procesov a troch médií - procesor, zbernica a pamäť.



Obr.21 Architektúra systému [20]

V tejto architektúre procesy modelujú softvérové úlohy vykonávané na procesore. Médiá modelujú procesor, zbernicu a pamäť. Služby, ktoré táto architektúra poskytuje sú opísané v metódach procesov. Procesy poskytujú tieto metódy funkcionálnej časti. Metamodel zahŕňa kvantitu na označenie jednotlivých udalostí a používa hodnoty na meranie ceny. V príklade je použitá energia a čas potrebný na vykonanie určitej udalosti [20].

Mapovanie

Vyhodnotenie určitej implementácie vyžaduje mapovanie funkcionálneho modelu na architektúrny model. Toto môže metamodel uskutočniť bez modifikácie funkčných a architektúrnych sietí. Definuje novú sieť, ktorá obsahuje funkčnú a architektúrnu sieť ktorých udalosti sú synchronizované. Táto mapovacia sieť predstavuje najvyššiu vrstvu, ktorá špecifikuje mapovanie medzi funkciami a architektúrov [20].

Podpora nástrojov

Systém metropolis na vstupe načítava zdrojový jazyk metamodelu a vytvára z neho abstraktný syntaktický strom. Následne tento strom môže predstavovať vstup do roznych nástrojov na jeho analýzu. Jedným z hlavných nástrojov je simulátor, ktrorý zachováva sémantiku metamodelu pri preklade špecifikácie metamodelu do vykonateľného jazyka SystemC. Pri simulácií sa uplatňujú definované obmedzenia. Nástroje na verifikáciu poskytujú kontrolu obmedzení. Metropolis poskytuje rozhranie pre systém xPilot na syntézu niektorých častí metamodelu [21].

Syntax jazyka metamodelu je podobná ako syntax programovacieho jazyka Java. Sú tu však zahrnuté niektoré obmedzenia a taktiež je tento jazyk rozšírený o niektoré kľúčové slová. Vykonávacia sémantika je taktiež rozdielna [22].

Limitácie

Všeobecnosť metamodelu predstavuje problémy pre používateľov a aj vývojárov. Používatelia sú prinútený učiť sa nový jazyk. Tento nový jazyk vyžaduje podporu pre jeho kompiláciu, simuláciu a debuggovanie.

Interakcie s kvantitami musia byť explicitne špecifikované a zjednodušenie predpokladov v špecifickej problémovej doméne nie je v metamodeli možné. Aj keď existujú konceptuálne a implementačné rozdiely medzi modelovanou cenou a plánovacími pravidlami s manažérmi kvantít, vo vykonávacej sémantike toto nie je jasné. Dôsledkom je náročná úloha pri špecifikácii manažérov kvantít a interakcií medzi nimi [21].

2.5.4 ForSyDe

SoC (System-on-chip) dokáže integrovať viaceré rozdielne výpočtové zdroje. Tieto zdroje môžeme rozdeliť na aplikačné špecifické integrované obvody (ASICs), programovateľné (procesor respektíve DSP), konfigurovateľné FPGAs alebo pasívne (pamäť) a ich vzájomné kombinácie.

Majú efektívne návrhové metódy adresovania komplexných systémov, ktoré začínajú vysokou úrovňou abstrakcie. Používajú formálne modely a transformácie v systémovom návrhu ako verifikácia a syntéza, ktoré môžu byť výhodami pri samotnom návrhu. Sú to dôležité body funkcionálnej špecifikácie a je to základný model matematického modelu výpočtu. Všetky tieto atribúty podporuje ForSyDe. Prevádza modelovacie techniky tak, že výsledky sú v abstraktnom a formálnom systémovom modeli. Tento systém je založený na synchronizovanom výpočtovom modeli. Metóda poskytuje knižnicu procesových konštruktorov, ktorá sa používa na konštrukciu procesov. Sú implementované v synchrónnom výpočtovom modeli a majú štruktúrovanú interpretáciu v hardvéri a softvéri. [23]

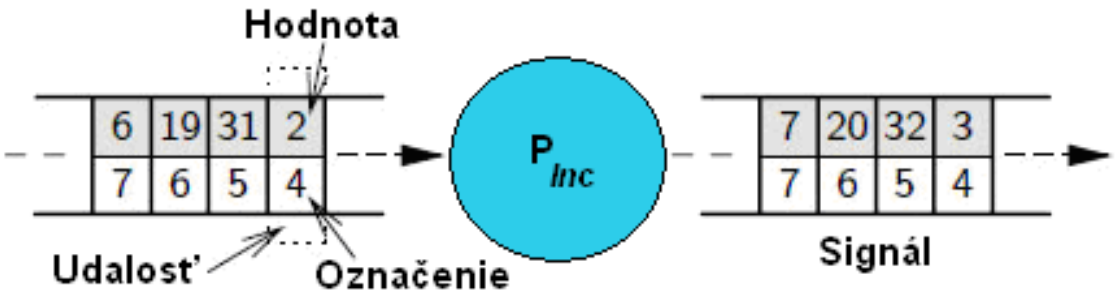
ForSyDe Metodológie

V ForSyDe systémový návrh začína s vývojom formálneho systémového modelu, ktorý je čisto funkcionálny a založený na synchronizácii hypotéz. Formálny charakter povoľuje postupne navrhovať vylepšenia cez formálne definovaný návrh transformácií. Ktorý je buď sémantický zachovaný, alebo obsahuje návrhové rozhodnutie, ktoré je obmedzujúce veľkosťou ideálneho nekonečného bufferu. Procesy sú vytvorené z formálnych metód konštruktorov. Tieto konštruktory sú hardvérové a softvérové sémantiky, ktorým povoľujeme preklad z vylepšeného systémového modelu do kombinovanej hardvérovej/softvérovej implementácie. [24]
Systémový model odráža návrhové princípy z ForSyDe metodológie.

Za účelom povolenia formálneho návrhu vysokej úrovne abstrakcie, obsahuje tieto základné charakteristiky:

- je založený na synchrónnom výpočtovom modeli, ktorý čisto oddeľuje výpočet od komunikácie
- je čisto funkcionálny a deterministický
- používa ideálne dátové typy ako zoznamy s nekonečnou veľkosťou
- používa koncept dobre definovaných procesových konštruktorov, ktoré sú implementované v synchrónnom výpočtovom modeli
- je založený na formálnych sémantikách a môže byť vytvorený použitým funkcionálneho jazyka Haskell

Za účelom formálneho opisu ForSyDe výpočtového modelu, používame denotational framework od Lee a Sangiovanni-Vincentelli. Definuje signály ako množinu udalosti , kde každá udalosť e má označenie t a hodnotu v. $e = (t; v) \in T \times V$
Ak je náš systém synchrónny, T je z množiny prirodzených čísiel, a všetky signály sú z rovnakej množiny označení. Na obrázku je znázornené modelovanie signálov a správanie procesov. Počas n-krát opakujúcej sa udalosti, sa každý proces spracovania udalosti z každého vstupného signálu s označením n, transformuje na výsledok s rovnakým označením n. Model procesu môžeme použiť ako koncept procesových konštruktorov. [24]



Obr.22 Modelovanie signálov a procesov

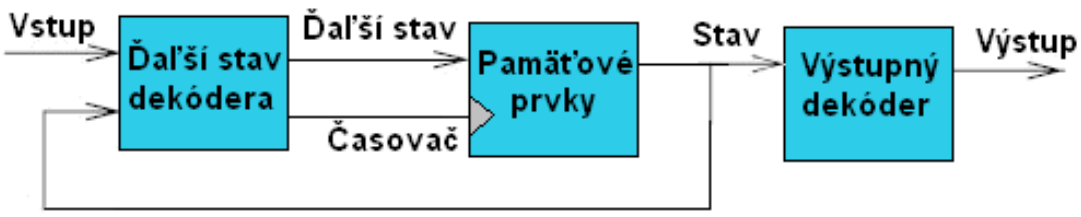
Procesov konštruktor môžeme nazvať ako mapSY. MapSY dostáva ako argument kombinačnú funkciu a vytvára proces. Proces Pinc bol vytvorený procesovým konštruktorom mapSY spolu inkrementom funkcie ako:

- Pinc = mapSY inc

Procesový konštruktor mooreSY je príklad konštruktora s lokálnym stavom. Tento model je konečný automat Moorovho typu. Ako prvý argument dostáva funkciu ns, ktorou prepočítava ďalší stav, druhý argument je funkcia out, ktorá sa používa na prepočítanie výstupu a ako posledný argument dostáva hodnotu s0, číže začiatočný stav.

- PMoore = mooreSY ns out init

Týmto spôsobom ho implementuje ako konečný automat. Hardvérový proces Pmoore môže byt implementovaný ako FSM, kde ďalší stav dekodéra obsahuje funkciu ns. Výstupný dekodér reprezentuje funkciu out a pamäťové časti reprezentujú dátový typ s0.



Obr.23 Hardvérová interpretácia

Nové procesy môžeme vytvoriť podľa sieti procesov, ktoré sú zapísané ako množina rovníc. Takú sieť voláme blok. Blok s1 = s5 kde (s2,s3) = P1s1s5 = P2(s2,s4)s4 = P3s3.

Návrhový proces

ForSyDe návrhový proces začína so vývojom formálneho abstraktného funkcionálneho systémového modelu, vytvoreného v funkcionálnom jazyku Haskell. Tento model je vylepšený o vnútornú funkcionálnu doménu, ktorá postupne aplikáciu pretransformuje od dobre definovaného návrhu transformácií do efektívneho implementačného modelu. [25]

2.5.5 SML-sys

V dnešnej dobe rýchlo narastá zložitosť návrhu vnorených systémov čo je veľmi obmedzujúce pri návrhu tzv. System-on-Chip systémov (SoC). Označenie System-on-Chip patrí procesorom, ktoré integrujú všetku potrebnú konektivitu a moduly pre koncové zariadenia. Nie je potrebný samostatný procesor, grafická karta alebo čipset. Snahou je zariadenie čo najviac integrovať do jedného alebo čo najmenej obvodov. Pre zvládnutie tohto nárastu zložitosti bola snaha sústredená na vývoj potrebných nástrojov a ich integráciu s vhodnou technológiou pre poskytnutie vyššieho stupňa automatizácie návrhu.
SoC a iné komplexné distribuované systémy obsahujú rozličné súčasti ako sú digitálne signálové procesory, mikro-regulátory, aplikačnú špecifickú logiku a pod., ktoré vyžadujú viaceré samostatné framework-y vyjadrujúce rozličné výpočtové modely pre modelovanie ich funkčnosti. Každá časť systému má väčšinou iné správanie sa, čo sa najlepšie modeluje v samostatných a rôznych výpočtových modeloch. Samotná povaha výpočtov v každej časti systému sa výrazne líši od ostatných. Formálna verifikácia komplexného modelu s viacerými interaktívnymi modelmi výpočtu je dôležitá, pretože interakcia medzi rôznymi modelmi s rozdielnou opisovou schopnosťou a rozdielnymi vlastnosťami by mohla viesť k chybám. Framework-y modelujúce systémovú úroveň ako napr. Ptolemy II alebo SystemC-H zjednodušujú modelovanie rôznych výpočtových modelov ale sú založené na programovacom jazyku ako Java, C++, atď..

V týchto framework-och sú spolu výpočtové a komunikačné aspekty medzi jednotlivými modelmi častí systému zväčša prepletené, čo nie je dobré pre formálnu analýzu. Pre zjednodušenie problému a umožneniu funkčnej verifikácie existuje Axel Jantsch-ov funkcionálny vzor založený na sémantickej definícii výpočtových modelov nazývaný SML-sys. SML-sys je framework implementovaný v štandardnom SML (Standard ML) modulárnom programovacom jazyku. Dôležitá vlastnosť je, že každý pod-systém môže byť modelovaný na základe vlastného výpočtového charakteru. SML-sys popisuje výpočtové modely významovo na rozdiel od Ptolemy II, ktorý predstavuje pohľad na ich funkcionalitu. Významový popis sa skladá z rekurzívneho formalizmu sémantiky modelov s použitím matematických objektov. [26] SML-sys ponúka odpovede na tieto otázky:

- aký výpočet bol vykonaný?
- ako prebieha komunikácia medzi rôznymi procesmi modelu?
- aké je časové správanie sa modelu v rôznych úrovniach?

Výpočtové modely (Models of Computation, MoC)

Výpočtový model je vybraný na základe vhodnosti (kompaktnosť popisu, presnosť návrhového štýlu, schopnosť syntetizácie a optimalizácie správania sa vhodnej implementácie) a popisuje správanie sa návrhu. Skladá sa z procesov (processes), udalostí (events) a signálov (signals). Udalosti sú základné jednotky vymieňané medzi procesmi.
Procesy príjmajú alebo spracovávajú udalosti a odosielajú alebo vysielajú udalosti. Signály sú konečné alebo nekonečné postupnosti udalostí. Aktivita procesov je delená na vyhodnocovacie cykly. Proces rozdelujeme jeho vstupnými a výstupmnými signálmi do sub-sekvencií zodpovedajúcich ich vyhodnocovacím cyklom. Počas každého vyhodnocovacieho cyklu proces vysiela presne jednu sub-sekvenciu každého vstupného signálu.

SML-sys podporuje nasledujúce výpočtové modely:

- nečasovaný (untimed) – Ide o komunikáciu a synchronizáciu procesov s ostatnými procesmi bez ohľadu na čas. Býva použitý na vyjadrenie dátového toku modelov, automatov, atď..
- synchrónny (synchronous) – Každý výpočet v intervale nastáva v rovnaký čas a intervaly sú rozdelené pozdĺž časovej osi. Používaný je na modelovanie výpočtov, u ktorých predpokladáme synchrónne správanie.
    - dokonalý synchrónny (perfect) – Výstupná udalosť procesu nastáva v rovnaký čas ako vstupná udalosť.
    - hodinový synchrónny (clocked) – Každý proces obsahuje oneskorenie zo vstupnej udalosti na výstupnú. Oneskorenie je ekvivalentné trvaniu vyhodnocovacieho cyklu. Najlepšie vyjadruje digitálny hardware.
- časovaný (timed) – Ide o zovšeobecnenie synchrónneho modelu kde môže proces spracovať a vyslať akýkoľvek počet udalostí vo vyhodnocovacom cykle a časová štrukúra je tu podrobnejšia. Býva použitý na modelovanie požiadaviek v reálnom čase ako sú napr. časové analýzy na výpočet šírky hodinového cyklu. [26]

Formulácia výpočtových modelov v SML-sys

Procesy komunikujú s inými procesmi prostredníctvom zápisu a čítania signálov. Množina hodnôt V reprezentuje dáta prenášané signálom a množina E predstavuje hodnoty, ktoré obsahujú udalosti. Postupnosť udalostí predstavuje signál. [27]

Udalosti

Rozlišujú sa tri typy udalostí. Prvým je množina nečasovaných udalostí E. = V, druhým je množina synchrónnych udalostí E- = V ∪ {|_|}, kde |_| je absencia udalosti, a posledným je množina časovaných udalostí E^ = E-, ktorá je identická s druhou množinou ale s iným označením.

Signály

Sú to usporiadané postupnosti udalostí označovaných ako ei kde hodnota i predstavuje poradové číslo signálu. Používame S., S-, S^ na označenie množiny nečasovaných, synchrónnych a časovaných signálov. <> predstavuje prázdny signál a ⊕ spája dva signály.

Možné manipulácie zo signálmi sú nasledovné:

- take(s,n) – získanie prvého n elementu signálu s
- drop(s.n) – vymazanie prvého n elementu signálu s
- head(s) – získanie prvého elementu signálu s
- length(s) – dĺžka signálu s

Procesy

Procesy delíme na dva typy v závislosti od toho či ide o vnútorný stav alebo nie. Procesy kde nejde o vnútorný stav definujeme ako funkcie nad signálmi mapované medzi signálovými množinami napr. (p: S.→S.). Procesy vnútorného stavu reagujú inak - v rozdielnom čase a okamžite pre rovnaké vstupné udalosti. Generujú výstup, ktorý je závislý na konkrétnom vstupe rovnako ako predchádzajúci stav procesu.

2.6 Zhodnotenie analýzy

Jednotlivé časti analýzy nám pomohli vytvoriť si lepšiu predstavu o funkcionalite a modularite vytváraného prostredia pre návrh digitálnych systémov. Na základe analyzovaných súborových štandardov a podporných systémov môžeme povedať, že nami navrhovaný systém by mohol podporovať najdôležitejšie a najrozšírenejšie súborové štandardy BLIF, KISS, PLA a EQN. Analyzované systémy predstavujú dobrý základ pre náš systém pretože vedia pracovať so spomínaními súborovými štandardmi a umoňujú vykonávať rôzne transformačné a iné operácie nad logickými obvodmi zapísanými pomocou týchto štandardov. Systémy ako napríklad BDS sa dajú priamo využiť, keďže sú jednoducho skompilovateľné a spustiteľné s požadovanými parametrami. Preto navrhovaný systém môže využiť funkcionalitu takýchto systémov, pridaním príslušného vykonateľného programu (napríklad prostredníctvom modulu).

Systém môže obsahovať aj grafický simulátor Petriho sietí (napríklad Pipe2), ktorý je vhodnou voľbou pre jeho dobrú funkcionalitu a nezávislosť od platformy. Systém by mal tiež podporovať zápis rôznych druhov kódu (napríklad opisný jazyk VHDL). Preto by mal obsahovať aj textový editor podporujúci danú syntax a možnosti editovania textu. Z analyzovaných textových editorov by bol najvhodnejší PSPad pre jeho najväčšiu podporu požadovaných vlastností.

Analýzou existujúcich frameworkov sme si ujasnili, aké vlastnosti by mal spĺňať nami navrhovaný systém. Systém by mal mať presne špecifikované rozhrania, ktoré poskytuje pre pridávanie nových súčastí.

3 Návrh riešenia

Kapitola návrh riešenia spočíva v špecifikovaní hlavných požiadaviek na vytváraný systém čo sa týka funkcionality, robustnosti a hrubého návrhu systému, kde je navrhnuté jadro systému, grafické rozhranie, jednotlivé vstupy a výstupy, pridávanie, modifikovanie a integrácia rôznych pluginov (programy pre rozšírenie funkcionality), spustenie a riadenie navrhnutej aplikácie.

3.1 Špecifikácia požiadaviek

Pri šecifikovaní jednotlivých požiadaviek sme brali do úvahy rôzne kritériá s používateľského hľadiska ako aj z pohľadu správnej a dostatočnej funkcionality vzhľadom na zadanie projektu. Špecifikovali sme funkcionálne aj nefunkcionálne požiadavky uvedené v nasledujúcich podkapitolách.

3.1.1 Funkcionálne požiadavky

Funkcionálne požiadavky pre navrhovaný systém vychádzajú aj zo samotného zadania projektu. Aby systém dosahoval cieľe projektu, ktoré boli v tomto zadaní stanovené musí spĺňať nasledujúce funkcionálne požiadavky, ktoré budú v ďalšej časti podrobne vysvetlené:

- Modularita
- Rozšíriteľnosť
- Škálovateľnosť
- Univerzálnosť
- Prezentovateľnosť

3.1.1.1 Modularita

Na základe existencie rôznych nástrojov, ktoré pracujú s digitálnymi systémami, a ktoré sme bližšie analyzovali v predchádzajúcej kapitole, implementovaný systém musí byť v čo najväčšej miere modulárny tak, aby bolo možné tieto nástroje prostredníctvom systému používať. To znamená, že musí byť vytvorený postup, ktorý bude umožňovať špecifickým spôsobom zakomponovať požadovaný nástroj do systému. Pri tomto zakomponovaní musia byť zohľadnené špecifické vstupy, výstupy a interné vlastnosti požadovaného nástroja. Tieto vstupy, výstupy a vlastnosti nástroja musia byť pri procese zakomponovania opísané používateľom, ktorý tieto nástroje do systému pridáva. Aj napriek skutočnosti, že rôzne nástroje majú rôzne vlastnosti, môžu byť do systému na základe takéhoto opisu tieto nástroje vložené.
Modularita systému musí byť zabezpečená naznačeným spôsobom. Rozšírenia programu alebo moduly teda budú obsahovať okrem samotného nástroja na prácu s digitálnymi systémami aj opis jeho vlastností spôsobom špecifikovaným v návrhu systému.

3.1.1.2 Rozšíriteľnosť

Požiadavka na rozšíriteľnosť je priamo závislá na modularite. Implementovaný systém vo svojej podstate nemusí vykonávať žiadne operácie nad digitálnymi systémami. Musí však poskytovať možnosť ako rozšíriť funkcionalitu tak aby plnil požadovanú úlohu. Napríklad ak bude potrebné aby systém poskytoval možnosť vykreslenia kombinačného logického obvodu, musí existovať postup ako tento systém o danú funkcionalitu rozšíriť. V tomto konkrétnom príklade pôjde o nástroj, ktorý dokáže takýto obvod vykresliť a o jeho zakomponovanie do systému prostredníctvom modulu. V konečnom dôsledku musí byť používateľovi umožnené zvoliť si, rozšíriť alebo modifikovať funkcionalitu systému prostredníctvom predpripravených modulov alebo vytvorenia nových modulov.

3.1.1.3 Škálovateľnosť

V analýze sme opísali rôzne systémy na zapisovanie logických systémov do súboru, nástroje na modifikovanie logických obvodov a nástroje na transformáciu medzi rôznymi zápismi logických obvodov. Podstatou systému je podporovať čo najväčšiu škálu nástrojov, metodík a postupov pri práci s logickými obvodmi. V prípade súborových systémov, ktoré opisujú logické obvody musí systém vedieť nielen zobraziť a upraviť tento zápis ale musí vedieť aj zvýrazňovať určité hlavné črty syntaxe, pomocou ktorej je daný súborový systém charakterizovaný. Pri rôznych transformačných nástrojoch musí byť možné zobrazovať vstupy do týchto nástrojov a výstupy z týchto nástrojov. Tak isto musí systém umožňovať modifikovať tieto vstupy pre experimentovanie s príslušnou transformáciou. Systém musí podporovať editor na vykresľovanie a prípadnú úpravu rôznych reprezentácií logických obvodov.

3.1.1.4 Univerzálnosť

Výsledný systém musí byť v čo možno v najväčšej miere implementovaný nezávisle na platforme. Môžu však nastať obmedzenia pri pridávaní modulov a konkrétne jednotlivých vykonateľných programov v týchto moduloch.
Je to z toho dôvodu, že niektoré programy, ktoré pracujú s logickými obvodmi môžu byť dostupné len pre platformu windows a niektoré len pre platformu linux. Systém teda musí podporovať tieto programy podľa toho, na ktorú platformu bol nainštalovaný a musí kontrolovať kompatibilitu príslušných modulov tak aby zistil, či je modul kompatibilný s danou platformou.

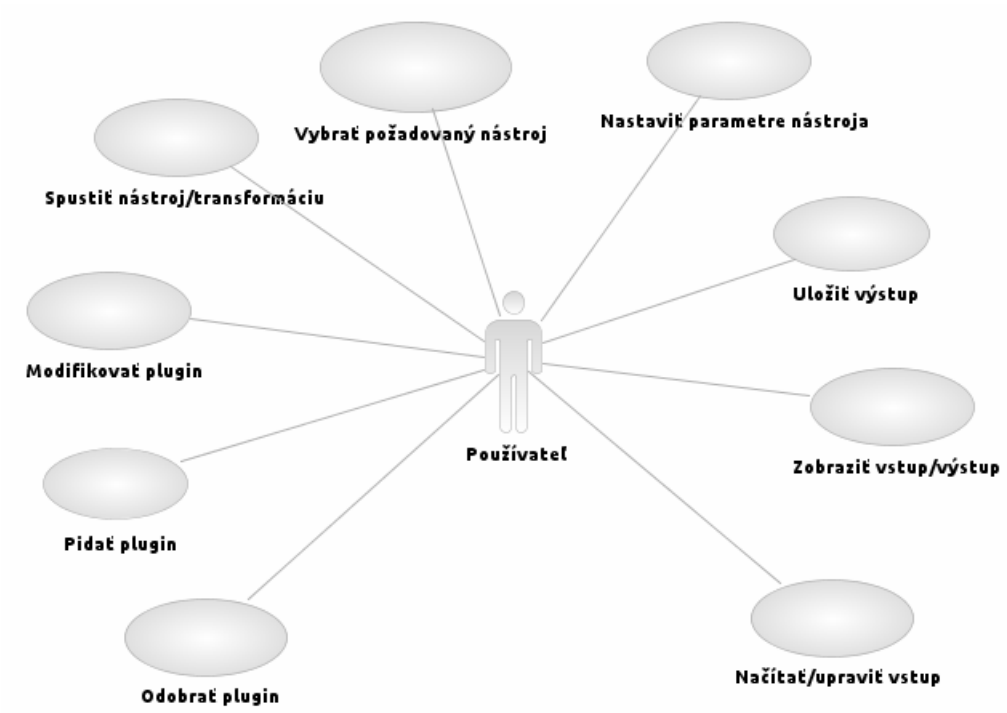
3.1.1.5 Prezentovateľnosť

Systém musí implementovať grafické používateľské rozhranie, ktoré bude poskytovať okrem bežnej možnosti práce so systémom aj možnosť spravovania systému. Používateľ bude primárne používať toto grafické rozhranie.

Vzhľadom na to, že v grafickom rozhraní sa integrujú všetky vlastnosti systému, teda aj vlastnosti vyplývajúce z uvedených požiadaviek, pomocou tohto rozhrania musí mať používateľ možnosť vykonávať tieto akcie:

- Pridať nový modul
- Odobrať modul
- Modifikovať modul
- Modifikovať typy vstupov, výstupov a parametre modulu
- Vybrať si požadovaný nástroj
- Zvoliť požadované parametre vybraného nástroja
- Uložiť a načítať vstup
- Zobraziť a upraviť požadovaný vstup do programu
- Zobraziť, upraviť a uložiť výstup z programu

Na obr. 24 je znázornený model prípadov použitia systému.



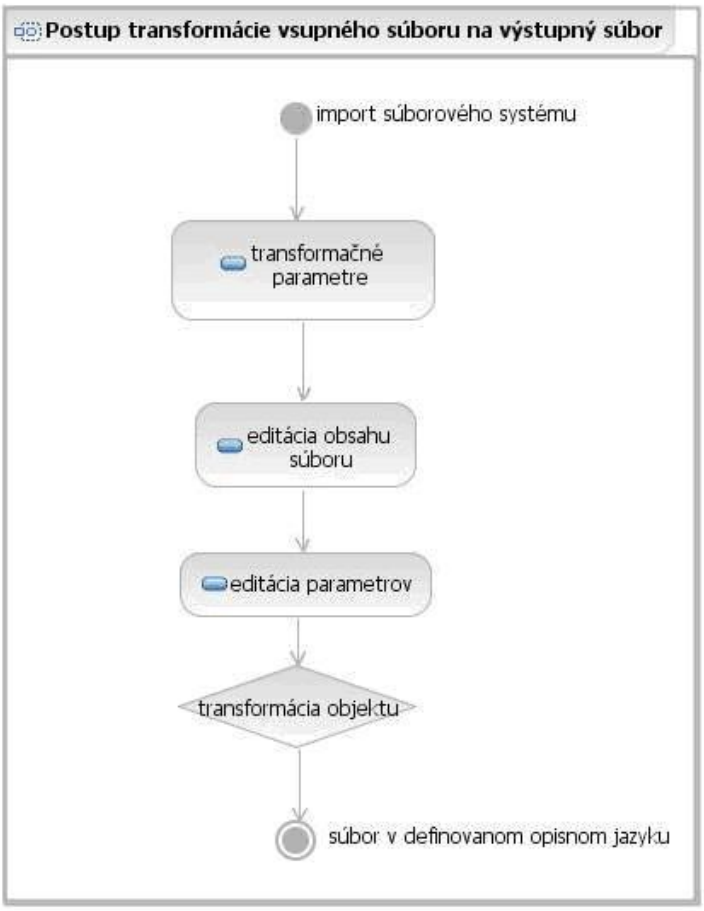
Obr.24 Model prípadov použitia.

3.1.2 Nefunkcionálne požiadavky

Z dôvodu vstupovania rôznych programov a s nimi súvisiacimi vstupnými údajmi a parametrami do systému je potrebné zabezpečiť aby bol systém robustný a teda odolával prípadnému nesprávnemu formátu vstupných dát alebo parametrov a reagoval na tieto situácie vhodným upozornením používateľa o vzniknutom probléme.
Ak by sa vyskytol problém pri vykonávaní niektorého z programov a tento program by prestal reagovať, systém musí tento stav detegovať a tento program ukončiť aby nedošlo k obmedzeniu činnosti celého systému.

3.2 Návrh

Aplikácia bude tvorená pomocou dvoch hlavných stavebných prvkov. Jedným je samotné jadro aplikácie, kde sa vykonáva celá logika systému, tou druhou je grafická časť. Toto rozdelenie zabezpečí, že program bude spustitelný aj pod operačnými systémami bez grafického rozhrania. Aplikácia bude schopná príjmať používateľom zadané vstupné údaje, ktorými budú rôzne typy súborových systémov. Tieto súbory môžu byť v aplikácií editované pomocou grafického rozhrania. Následne sa vykoná transformácia do opisného jazyka, ktorý si sám používateľ zvolí. Celá základná procedúra, ktorú bude aplikácia vykonávať, je znázornená na Obr. 25.



Obr.25 Riadenie logiky aplikácie.

3.2.1 Jadro aplikácie

Jadro tvorí základ celého programu, ktorý bude z veľkej časti implementovaný na základe špecifikácie požiadaviek. Jadro aplikácie bude tvorené všetkou logikou, ktorá bude zahŕňať:

- Importovanie a editáciu obsahu súborov
- Transformáciu súborových systémov na definované opisné jazyky
- Integráciu externých programov
- Integráciu grafického nástroja pre vykresľovanie grafov

Architektúra jadra bude navrhnutá tak, aby sa dali ľubovolne podľa uváženia pridávať, modifikovať alebo odoberať jednotlivé funkčné celky.

3.2.2 Grafické rozhranie

Spustenie samotnej aplikácie bude riešené cez intuitívne grafické rozhranie, ktoré bude slúžiť používateľovi na zjednodušenie práce s programom. Bude generovať používateľom zvolené parametre a následne volať funkcie z jadra aplikácie. Možnosti, ktoré bude rozhranie poskytovať sú:

- Import a editácia súborov
- Nastavenia parametrov pre generovanie výstupu
- Importovanie, modifikácia a mazanie pluginov, resp. externých programov

3.2.3 Návrh vstupov

Vstupmi budú jednotlivé súborové systémy, ktoré budú definované aplikáciou. Vstupy musia byť regulérne súborové systémy obsahujúce kombinačné alebo sekvenčné logické obvody, konečné automaty alebo Petriho siete. Program bude následne tieto vstupy spracovávať podľa uvdených požiadaviek.

3.2.4 Návrh výstupov

Výstupom bude formátovaný súbor, ktorého typ si používateľ sám vyberie. Súbor bude vygenerovaný na základe vstupného súboru a hodnôt príslušných parametrov.
Pred samotným generovaním výstupu program overí, či je možné definovaný typ transformácie vykonať. Ak nie, upozorní používateľa na nekorektnú definíciu, a pokúsi sa určiť, kde nastala chyba. Ak sú všetky parametre aj vstupný súbor v poriadku, vykoná sa transformácia.

3.2.5 Integrácia pluginov

Systém bude ľubovolne rozšíritelný o žiadanú funkcionalitu. Pridávanie funkcionality bude riešene prostredníctvom tzv. pluginov, ktoré si bude môcť používateľ nahrať do systému. Plugin v našom prípade bude znamenať nejaký predpripravený spustiteľný súbor, ktorý bude mať nejaké vstupy, parametre programu a následne môže dávať nejaký výstup. Systém bude udržiavať zoznam programov (pluginov) vo viacrozmernom poli. V tomto poli okrem programu, budú udržiavané formát vstupu programu, formát výstupu programu a prípadne aj parametre programu. Pri práci so systémom používateľ už iba zadá aké vstupy bude používať , a aké výstupy bude požadovať. Na základe toho ako používateľ navolí vstupy a výstupy, systém zvolí jeden zo zoznamu programov.

3.2.6 Pridávanie pluginov

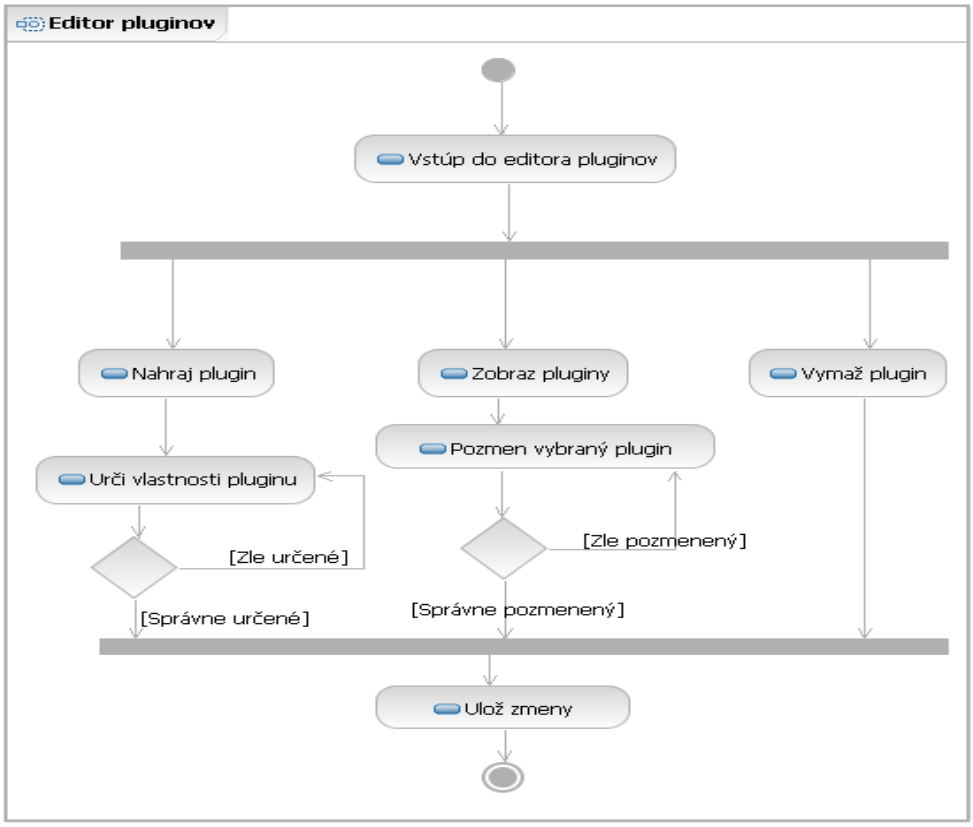
Systém bude obsahovať používateľské rozhranie určené na rozširovanie a modifikovanie pluginov. Rozhranie umožní vyhľadať daný plugin a pripojiť resp. nahrať ho do systému. Pred tým ako sa nahrá plugin do systému, používateľ vyplní požadované vlastnosti pluginu ako vstupný súbor, výstupný súbor a parametre, aby systém neskoršie vedel určiť, ktorý program (plugin) má použiť.

3.2.7 Modifikovanie pluginov

Systém bude udržiavať zoznam pluginov v akomsi viacrozmernom poli alebo databáze, ktorú nám systém umožní prezerať v rozhraní pre to určenom. Systém zobrazí zoznam a používateľ kliknutím na plugin zobrazí všetky informácie (vstupy, výstupy a parametre) o tomto plugine. Tieto následne bude môcť podľa uváženia meniť.

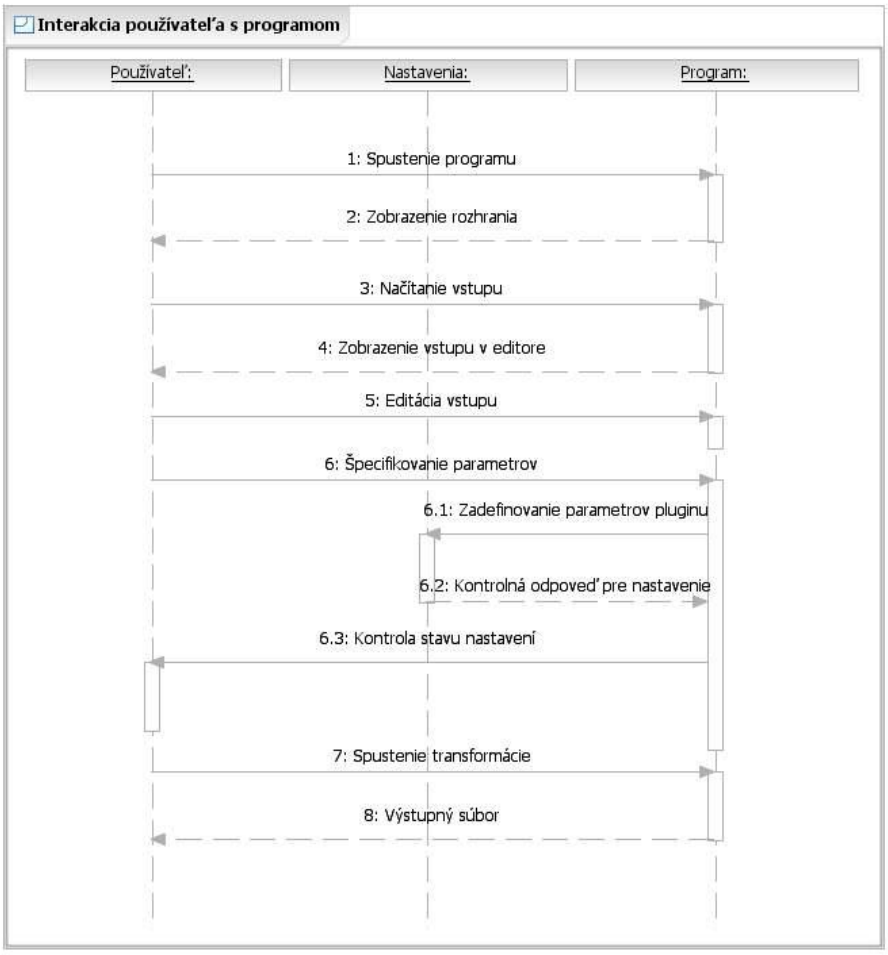
3.2.8 Riadenie aplikácie

Riadenie aplikácie bude pozostávať z interakcie používateľského rozhrania a používateľa. V ňom si používateľ bude môcť postupne definovať rôzne postupy, ktoré ho intuitívne navedú k požadovanému výsledku. Tieto kroky sú rozdelené podľa toho, či sa bude jednať o konfiguráciu programu, to znamená pridávanie, úprava alebo odoberanie programových prostriedkov alebo samotná konverzia vstupov na požadované výstupy. Tieto kroky sú znázornené pomocou diagramov na obrázku Obr. 26.



Obr.26 Integrácia pluginov

Diagram toku údajov (Obr. 27) je znázornený na systéme s grafickým používateľským rozhraním, no v prípade terminálovej aplikácie je postup rovnaký, no kroky už nie sú také intuitívne a postup taký zrejmý[5]. Ďalším rozdielom programu s použitím grafického rozhrania je možnosť editácie vstupu. Tento krok sa pri terminálovej plaikácií vynecháva. Kroky 1 až 5 sú zrejmé už podľa názvov z diagramu, preto k nim nie je potrebné ďalšie vysvetlenie. V kroku 6 sa definuje, s akými parametrami má program pracovať. Medzi mini patria aj definície nových alebo modifikácia existujúcich externých programov a ich možností spúšťania. Celý proces sa v krokoch 6.1 až 6.3 overuje a oznámi používateľovi odpoveď obsahujúcu stav kofigurácie. Následne môže v krokoch 7 a 8 prebehnúť transformácia vstupu na očakávaný výstup.



Obr. 27 Diagram toku informácií medzi používateľom a systémom.

[5] Všetky kroky sa vykonajú pomocou jedného príkazu.

3.2.9 Návrh grafického rozhrania

Pomocou grafického rozhrania bude používateľ schopný vykonávať všetky funkcie systému. Funkcie bude môcť vykonávať v krokoch, takže bude mať plnú kontrolu nad tým, kedy a ako sa čo vykoná. Grafiké rozhranie bude obsahovať možnosť editácie vstupného súboru pomocou textového editora, ktorý by mal mať možnosť zvýrazňovať syntax na základe vstupu, čo spehľadní kód a tým napomôže používateľovi k lepšej orientácií v súbore.
Prototyp rozhrania je znázornený na obrázku Obr. 28. Tento prototyp sa môže ďalej modifikovať podľa potrieb aplikácie.

Obr. 28 Prototyp grafického rozhrania

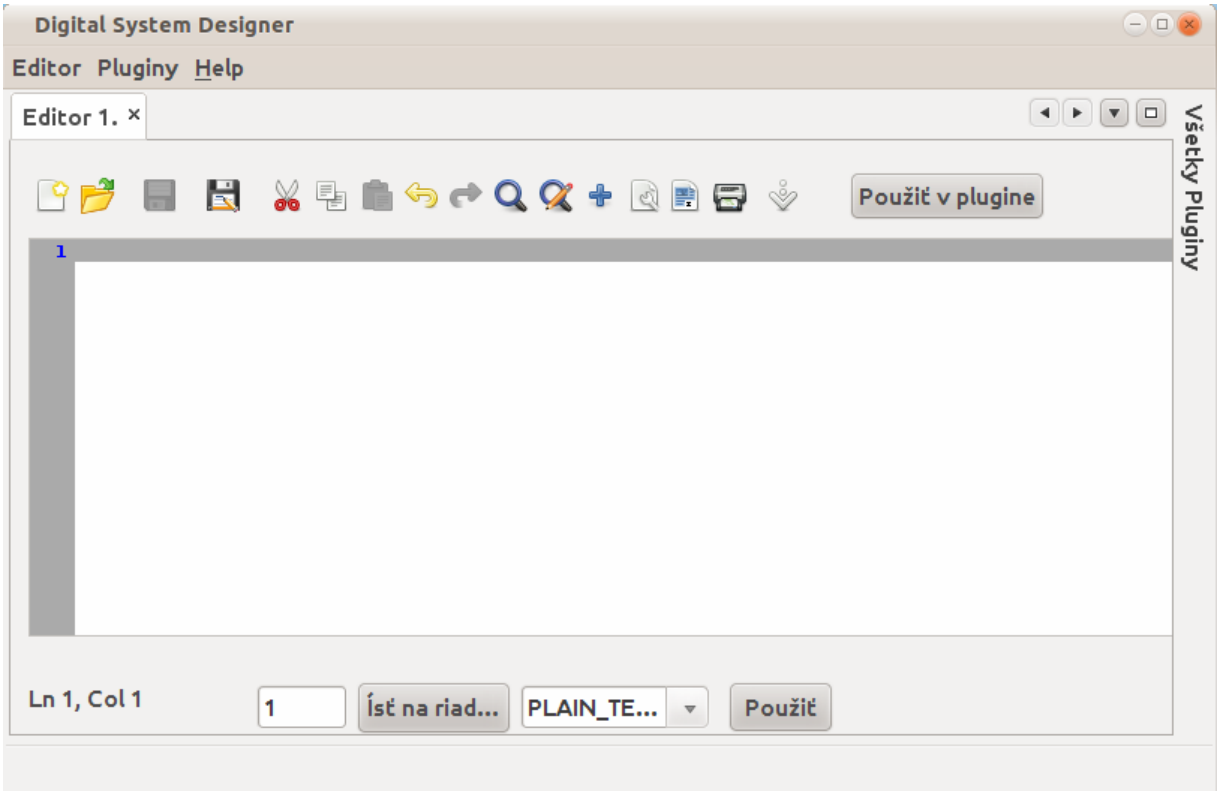
Grafické rozhranie je rozdelené na dve hlavné časti. Jednou je zobrazovanie a editácia súborov, ktorá má názov „Editor“. Táto časť môže obsahovať viac záložiek, takže bude možné pracovať

s viacerými súbormi naraz. Druhou časťou je zobrazovanie a editácia jednotlivých pluginov, v ktorej budú zobrazované dostupné pluginy a má názov „Pluginy“.

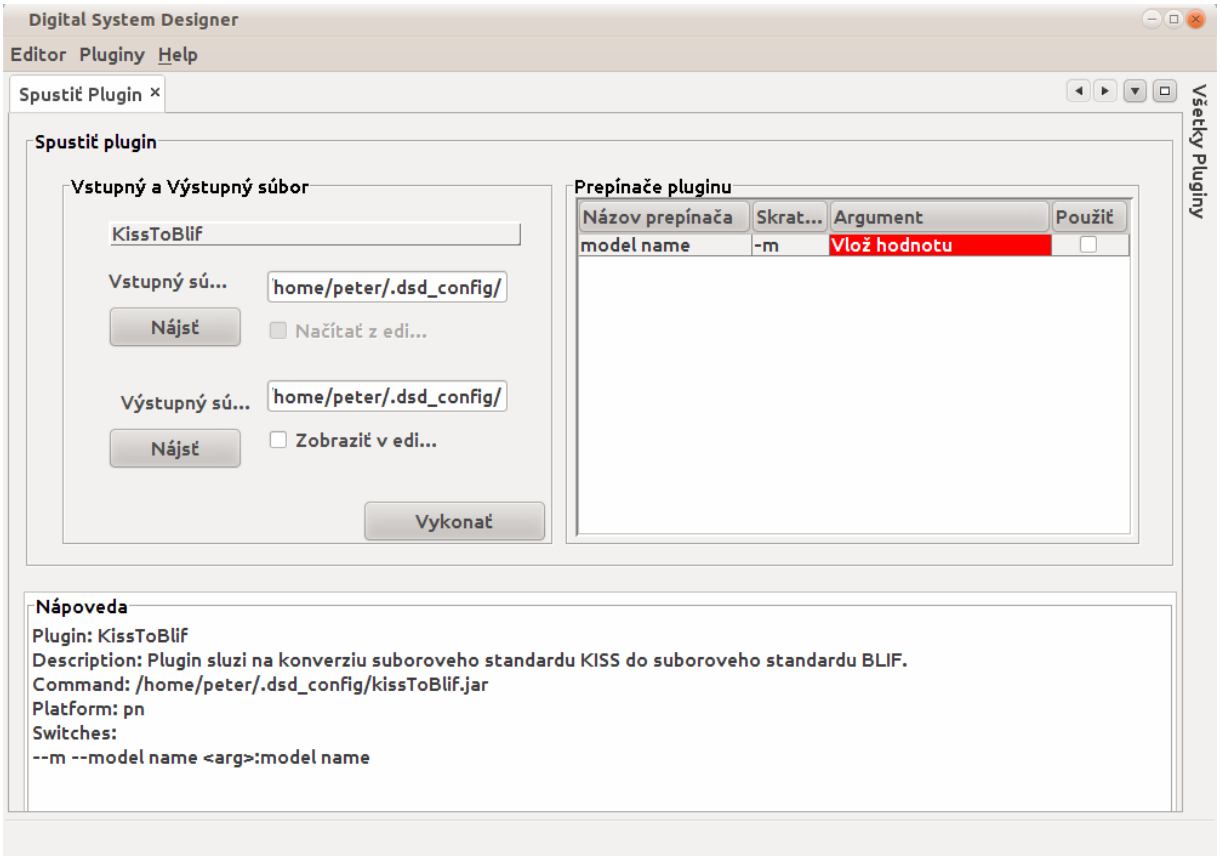
Rozhranie tiež obsahuje menu, v ktorom sú definované nasledovné možnosti:

- Menu Editor
  - Editor – zobrazenie okna textového editora pre úpravu a prehliadanie rôznych typov súborov s možnosťou použiť ich v konkrétnom plugine.
  - Exit – ukončenie aplikácie.
- Menu Pluginy
  - Zobraz pluginy – zobrazenie zoznamu dostupných pluginov s danými možnosťami:
    - Novy plugin – vytvorenie (pridanie) nového pluginu.
    - Pouzit plugin – použitie pluginu s vybranými parametrami na konkrétny súbor.
    - Uprav plugin – zmena existujúceho pluginu.
    - Delete plugin – odstránenie pluginu.
- Menu Help
  - About – vypísanie verzie aplikácie, typu platformy, typu operačného systému a adresárovej cesty.

Prototyp okna textového editora so základnými funkciami je na Obr. 29 a prototyp okna použitia konkrétneho pluginu je na Obr. 30. Pri použití pluginu bude potrebné zadať vstupný a výstupný súbor a parametre pluginu.



Obr. 29 Prototyp okna textového editora



Obr. 30 Prototyp okna použitia pluginu

4 Implementácia

4.1 Funkcionalita systému

Funkcionalita systému spočíva v korektnej kofigurácii a riadení rôznych externých častí systému a jeho spolupráce s externými časťami. Tu je potrebné zabezpečiť, aby bolo možné so systémom pracovať jednoducho, rýchlo a efektívne.

Práca so systémom je rozdelená do jednotlivých častí podľa toho, akú akciu chce používateľ vykonať:

1. integrácia nových častí systému, modifikácia alebo mazanie existujúcich častí systému
2. vykonanie požadovanej konverzie, syntézy a inej modifikácie digitálneho systému za pomoci externých častí – pluginov
3. vytvorenie alebo editácia vstupu (pripadá do úvahy len pri grafickom rozhraní)

Všetky postupy sú krokové, čo používateľovi sprehľadňuje vykonávanie jednotlivých akcií.

4.2 Špecifikácia požiadaviek na pluginy

V návrhu boli opísané požiadavky na pridávanie, odoberanie a modifikovanie pluginov. V tejto časti opíšeme, akým sposobom sú pluginy realizované a aké rozhrania musia byť pri tvorbe pluginov vytvorené, na zabezpečenie spr8vnej komunikácie s hlavným programom.

Plugin je na určitej platforme vykonateľný súbor, ktorý umožňuje spracovávať vstup a podľa špecifikovaných pravidiel zadaných prostredníctvom prepínačov, vygenerovať príslušný výstup. Syntax príkazu, ktorý spustí vykonateľný súbor je nasledovná:

```
program [ prepinac1 <argument1> … ] vstupny_subor vystupny_subor
```

Prepínače, ktoré program podporuje sú spolu s cestou k vykonateľnému súboru špecifikované v konfiguračnom súbore opísanom v nasledujúcej časti dokumentu.
Hlavný systém na základe týchto informácií, načítava z editora vstupné dáta, vyzve používatela na zadanie parametrov pri zvolenom plugine, vytvorí príkaz z uvedenou syntaxou a vykoná príslušný program. Výstupné dáta si používateľ môže pozrieť v editore.

4.3 Integrácia pluginov

Všetky pluginy a informácie o pluginoch sú uložené na jednom mieste, v xml súbore. Súbor obsahuje elemeny – pluginy, ktorého atribúty a vnorené elementy obsahujú konfiguračné parametre. Všetky tieto parametre sú konfigurované priamo používateľom pomocou aplikácie.

Xml súbor má nasledovnú štruktúru:

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- author, description, xml informations, etc. -->

<plugins>

      <plugin name="val" description="val" path="val" os=”val”>
            <argument shortName="a1" longName="val" description="val"
requireValue=”val” />
            <argument shortName="a2" longName="val" description="val"
requireValue=”val” />
      </plugin>

      <plugin name="val" description="val" path="val" os=”val”>
            ...
      </plugin>

      ...

</plugins>
```

Význam jednotlivých elementov, atribútov je nasledovný:

`<plugin` – root element xml súboru. Prítomný len kvôli dodržaniu štandardnej xml štruktúry. Element neobsahuje žiadne atribúty.

`<plugin>` – externá časť systému, obsahuje konfiguračné elementy a atribúty

`name` – meno pluginu

`description` – popis pluginu

`path` – absolútna cesta k externému programu

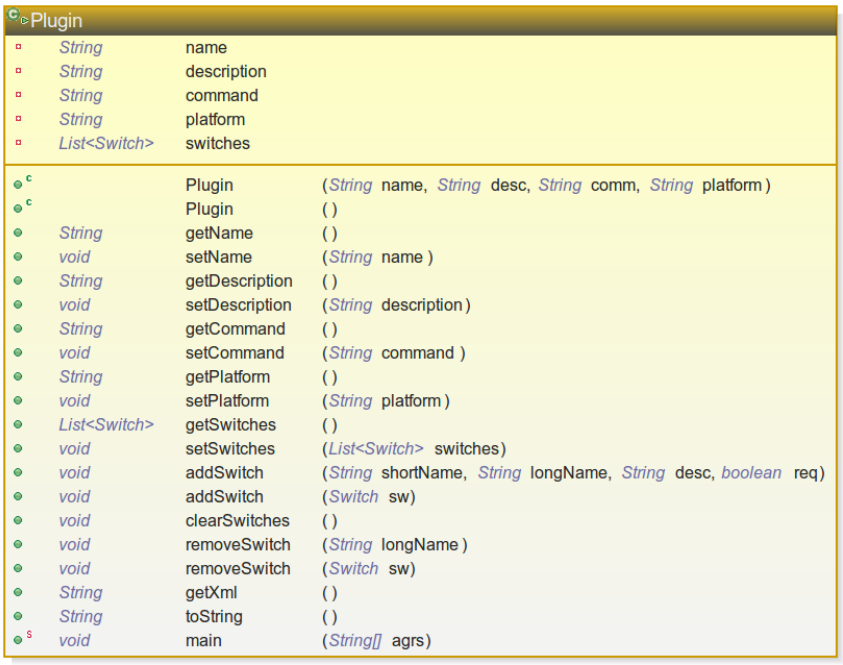
`os` – hodnota linux, windows alebo pn

`<argument>` – argument externého programu

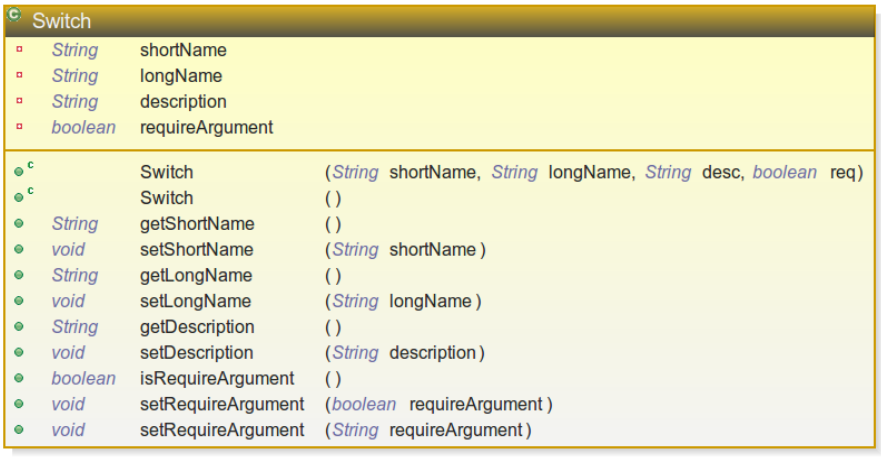
`requireValue` – hodnota true alebo false podla toho ci prepinac potrebuje argument

4.4 Implementácia pluginov

Pluginy sú implementované triedou Plugin (obr. 31), ktorá obsahuje informácie o názve, ceste k vykonateľnému súboru, opise a iné informácie o plugine a triedou Switch (obr. 32), pomocou ktorej sú reprezentované parametre pluginu.. Detaily príslušných tried sú znázornené na jednotlivých obrázkoch.

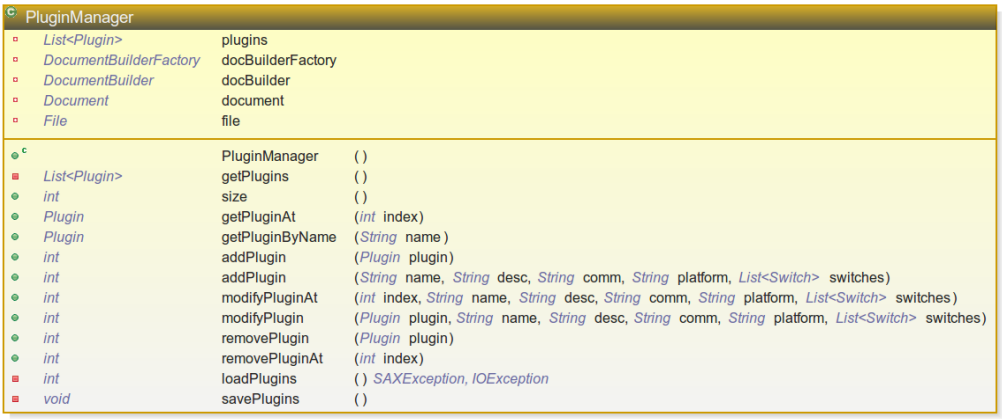


Obr. 31 Trieda Plugin



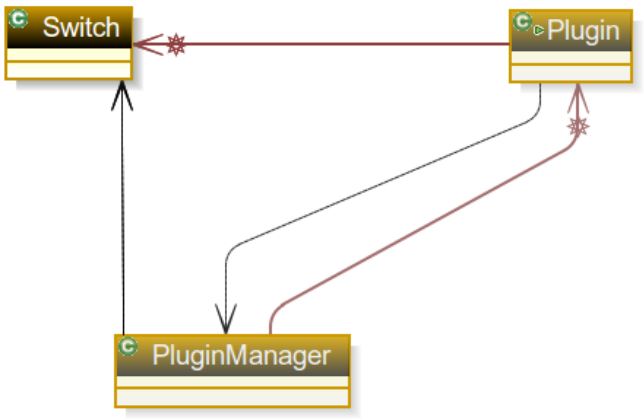
Obr. 32 Trieda Switch

Trieda PluginManager ukladá pluginy, s ktorými potom program pracuje. Na základe zoznamu týchto pluginov sa v grafickom rozhraní zobrazujú príslušné informácie o pluginoch. Trieda PluginManager je znázornená na obr. 33. Táto trieda má taktiež na starosti ukladanie a načítavanie informácií o pluginoch z konfiguračného súbora.



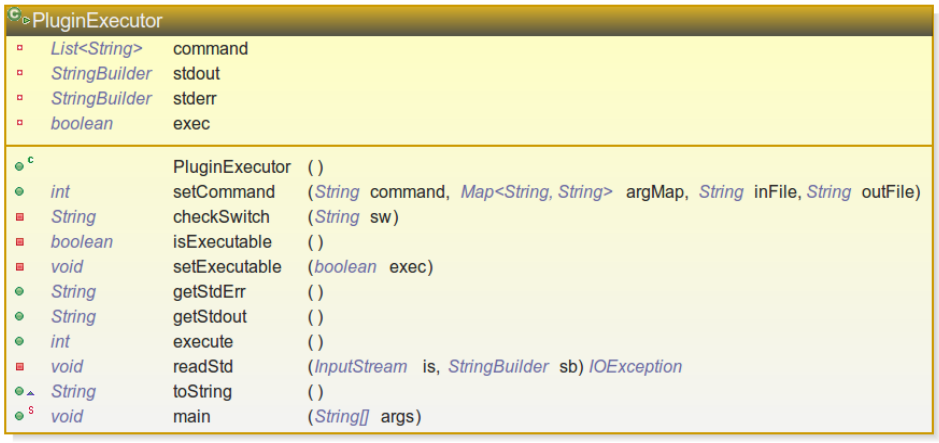
Obr. 33 Trieda PluginManager

Na obr. 34 sú znázornené relácie medzi jednotlivými triedami.



Obr. 34 Relácie medzi triedami Plugin, Switch a PluginManager

Ďalšou významnou triedou je PluginExecutor, ktorá má na starosti vykonávanie požadovaných príkazov, ktoré sú vytvorené za pomoci informácií o plugine a používateských vstupov. Táto trieda je znázornená na obr. 35.



Obr. 35 Trieda PluginExecutor

5 Testovanie

Vytvorený systém bol testovaný v operačnom systéme linux, windows a macintosh. Systém sa v každom prostredí správal bezchybne a bol plne funkčný. Jediné čo bolo potrebné je, aby bol na každom operačnom systéme nainštalovaný najnovší nástroj na podporu inštalácie java programov Java SE 6 Update 24, konkrétne Java Development Kit (JDK).

Funkcionalita systému bola testovaná pridávaním, odoberaním, modifikovaním a používaním rôznych pluginov, nastavovaním ich jednotlivých argumentov, zadávaním rôznych chybných vstupov a výstupov, používaním predvolenej a vlastnej syntaxe v editore, prácou so rôznymi typmi súborových štandardov a pod..

Systém testovali všetci členovia tímu pričom v každom prípade bola funkčnosť systému správna. Vytvorený systém je ošetrený voči všetkým známym chybám, ktoré môžu nastať.

6 Záver

Cieľom tímového projektu bolo vytvoriť prostredie pre návrh digitálnych systémov. Úlohou bolo vytvoriť čo najlepšie intuitívne prostredie s potrebnou funkcionalitou a možnosťou jej rozšíriteľnosti.

Najskôr sme spracovali podrobnú analýzu problematiky, ktorá zahŕňala všetky potrebné oblasti vrátane podobných existujúcich riešení. Ďalej sme vytvorili podrobný návrh riešenia a špecifikovali požiadavky na systém, ktoré sme implementovali.

Vytvorený systém je navrhnutý tak aby mohol byť dopĺňaný o ďalšiu funkcionalitu. Je ho možné modifikovať a zlepšovať. Pridávaním rôznych pluginov či už vlastných alebo dostupných sa zvyšuje jeho celková výpočtová schopnosť a stáva sa komplexnejším a univerzálnejším nástrojom v porovnaní s inými podobnými riešeniami. Systém je robustný a odolný voči chybám.

Počas práce na projekte si všetci členovia tímu osvojili základné postupy práce v tíme a získali hlbšie vedomosti z danej témy. Spoločne sa nám podarilo vytvoriť aplikáciu, ktorá má veľmi dobré predpoklady pre jej reálne nasadanie a používanie pre rôzne účely.

7 Použitá literatúra

[1] HOUŽVIČKA, K.: Rychlý simulátor kombinačních obvodů, Praha: FEL ČVUT, 2010. Bakalárska práca.

[2] University of California Berkeley. 2005. Berkeley Logic Interchange Format. http://www.cs.uic.edu/~jlillis/courses/cs594/spring05/blif.pdf

[3] SENTOVICH M., E. et. al.: SIS: A System for Sequential Circuit Synthesis. Electronics Research Laboratory. USA, Berkeley: Dept. of Electrical Engineering and Computer Science, University of California, 1992. 45 s.

[4] VAHID, F., GAJSKI, D.: SLIF: A Specification-Level Intermediate Format for System Design. In: IEEE, 1066-1409/95, 1995.

[5] Ciesielski, Maciej, et. al.. 2002. BDS: A BDD-Based Logic Optimization System. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.25.7158&rep=rep1&type=pdf

[6] Voronkov, Andrei. 2009. Binary Decision Diagrams. http://www.voronkov.com/lics_doc.cgi?what=chapter&n=3

[7] VICEN, J.: Podpora výučby problematiky Petriho sietí, Bratislava: FIIT STU, 2010. Bakalárska práca.

[8] Fiala, Jan. 2001. PSPad freeware editor. http://www.pspad.com/sk

[9] Crowd Favorite. 2000. Kate editor | Get an edge in editing. http://kate-editor.org

[10] The GNOME Project. 2007. Gedit text editor. http://projects.gnome.org/gedit

[11] Vemuri, Navin, et. al.. BDS-pga. Ver. 2.0. 2004.
http://www.ecs.umass.edu/ece/tessier/rcg/bds-pga-2.0

[12] VAHID, F., D. GAJSKI, D., et. al.: System-Level Exploration with SpecSyn. Design Automation Conference, 1998, pp. 812-817.

[13] Chai, D., et. al.. 2002. MVSIS 2.0 User's Manual.
http://embedded.eecs.berkeley.edu/mvsis/doc/mvsis_20_manual.pdf

[14] Chai, D., et. al.. 2002. MVSIS 2.0 Programmer's Manual.
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.132.7378

[15] University of California Berkeley. 1996. Ptolemy project.
http://ptolemy.eecs.berkeley.edu/ptolemyII/index.htm

[16] University of California Berkeley. 1998. Ptolemy design environment.
http://radio.feld.cvut.cz/Docs4Soft/ptolemy/

[17] University of California Berkeley. 2002. Ptolemy II heterogenerous concurrent modeling and design in Java. http://www.ece.umd.edu/DSPCAD/papers/bhat2002x3.pdf

[18] PATEL, H. D., SHUKLA, K. S.: Ingredients for Successful System Level Design Methodology, In: University of California Berkeley. USA: Springer, 2008, s. 92 – 107.

[19] Bluespec, Inc.. 2009. Bluespec. http://www.bluespec.com

[20] VINCENTELLI , S. A., et al.: Metropolis: An Integrated Electronic System Design Environment. In: Cadence Berkeley Labs. USA, 2003, s. 45 – 52.

[21] DAVARE, A., et al.: A Next-Generation Design Framework for Platform-Based Design, University of California Berkeley, USA: DVCon, 2007.

[22] Metropolis project team. The Metropolis Meta Model. Ver. 0.4. 2006. http://embedded.eecs.berkeley.edu/metropolis/forum/26/metropolis-1.1.2.src.tar.gz

[23] SANDER, Z. L., JANTSCH, I. A.: A Case Study of Hardware and Software Synthesis in ForSyDe, Royal Institute of Technology, Sweden: System synthesis, 2002, s. 86 – 91.

[24] SANDER, Z. L., JANTSCH, I. A.: The ForSyDe Semantics, Royal Institute of Technology, Sweden: System synthesis, 2002, s. 5.

[25] ACOSTA, A.: Hardware synthesis in ForSyDe, Sweden: KTH/ICT/ETS. 2007.

[26] JANTSCH, A., et. al.: SML-Sys: A Functional Framework with Multiple Models of Computation for Modeling Heterogeneous System, Forum of Specification and Design Languages, USA, Sweden, 2008, s. 33.

[27] VirginiaTech. 2010. SML Framework. http://www.fermat.ece.vt.edu/atool.php?uid=16

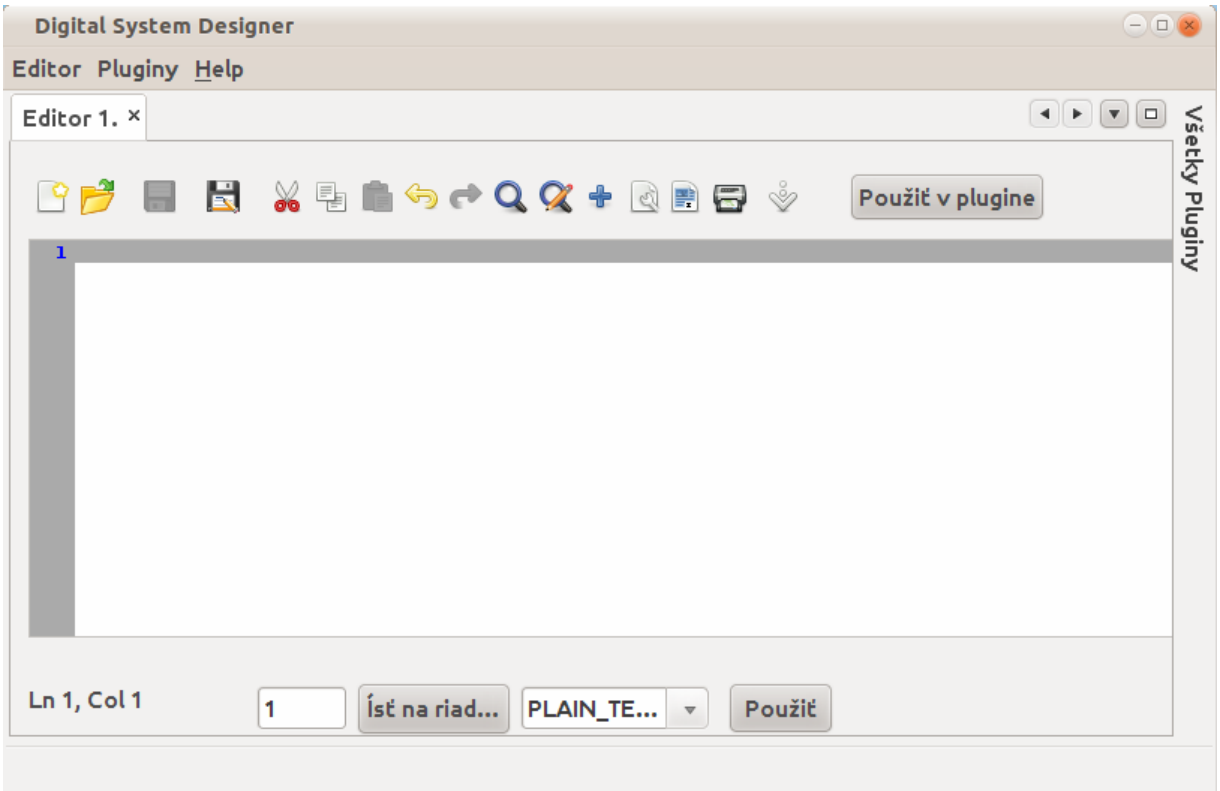
8 Príloha A: Používateľská príručka

8.1 Používanie

Systém môžeme rozdeliť na textový editor a na časť tzv. pluginov. Tieto sú umiestňované do vlastných okien. Systém umožňuje ľubovoľne premiestňovať a usporiadavať tieto okná.

8.1.1 Textový editor

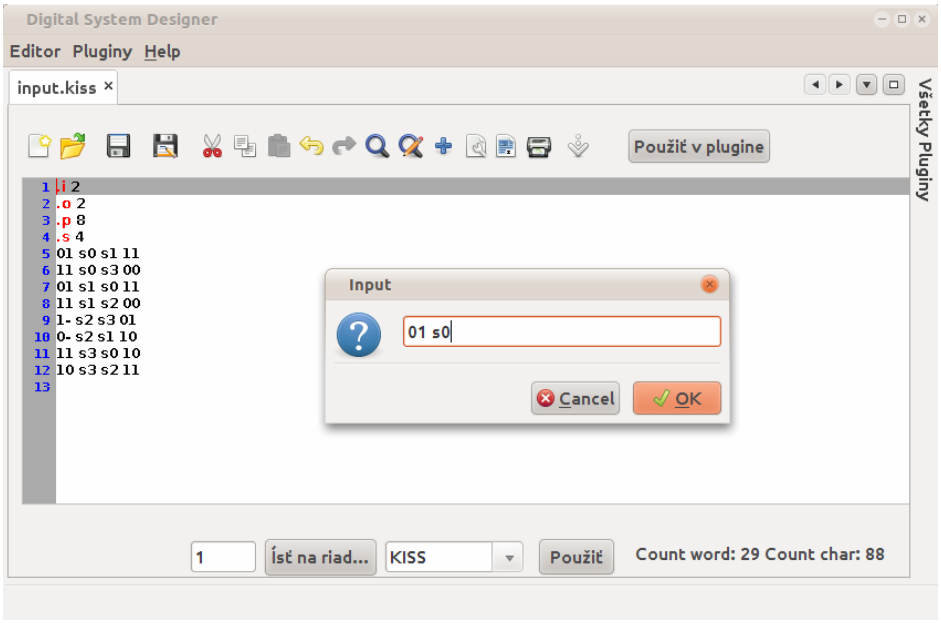
Textový editor spúšťame z úvodného okna, kliknutím na možnosť “editor” v hornej časti okna aplikácie alebo skratkou Alt+E (Obr. 36).



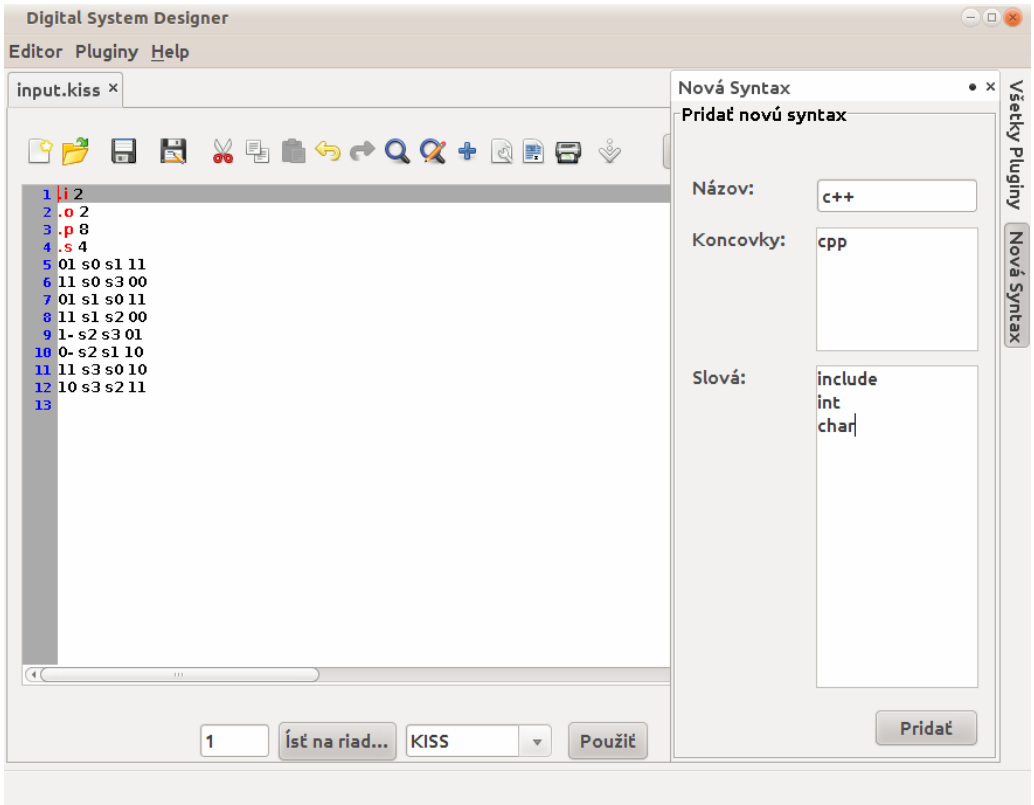
Obr. 36 Okno textového editora

Textový editor ponúka klasické funkcie, ktoré si teraz popíšeme. Tlačidlá pre funkcie editora sú v hornej časti okna v nasledujúcom poradí :

- vytvoriť nový súbor
- otvoriť súbor
- uložiť súbor
- uložiť súbor ako
- vystrihnúť (časť textu z editora)
- kopírovať (časť textu z editora)
- vložiť do editora (kopírovanú alebo vystrihnutú časť textu)
- tlačidlo späť (týka sa úpravy textu, späť na predchádzajúci stav)
- tlačidlo dopredu (týka sa úpravy textu)
- tlačidlo hľadať – umožňuje vyhľadať v texte požadovaný reťazec (Obr. 37)
- tlačidlo nahradiť – nahradí všetky zadané reťazce znakov iným zadaným reťazcom
- tlačidlo vytvoriť syntax – slúži na pridanie nového zvýrazňovania syntaxe, najprv musíme zadať názov tejto syntaxe, potom názov prípony súboru bez bodky napr. pri blif súbore „pokus.blif“, bude názov prípony iba „blif“, následne určíme, ktoré slova sa budú zvýrazňovať pri použitej syntaxe, neskoršie pri načítaní hocijakého súboru s príponou „blif“ do editora sa toto zvýrazňovanie syntaxe automaticky aplikuje na načítaný súbor (Obr. 38)
- tlačidlo správa – zobrazí informácie o texte v editori
- tlačidlo chod na riadok – je umiestnené v dolnej časti editora. Do kolónky napíšeme číslo riadku, na ktorý chceme, aby „skočil“ kurzor myši



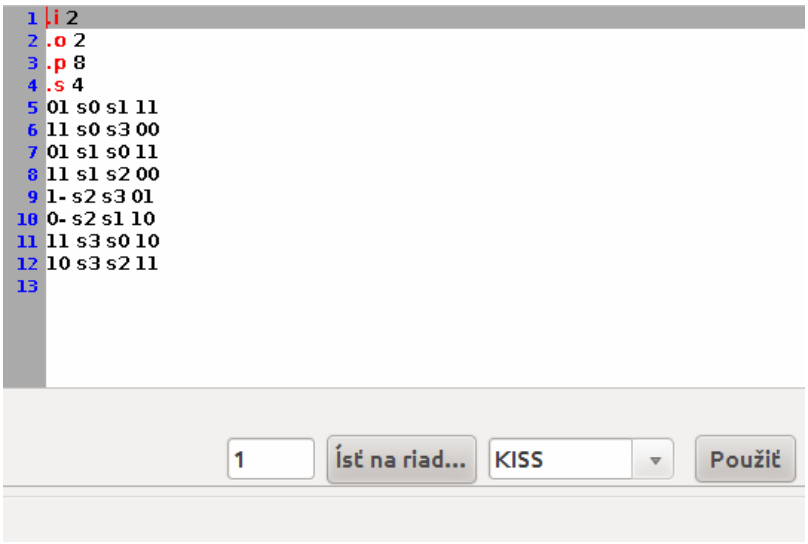
Obr. 37 Vyhľadávanie v texte



Obr. 38 Pridávanie zvýrazňovania syntaxe

Zvýrazňovanie syntaxe

Editor taktiež ponúka možnosť zvýrazňovania syntaxy pre rôzne súborové štandardy. Pri načítaní súboru do editora, systém automaticky rozpozná súborový štandard a prispôsobí mu aj zvýrazňovanie syntaxe.
Použite zvýrazňovanie syntaxy možno tiež, ale meniť. V spodnej časti okna je možnosť výberu zvýrazňovania syntaxe. Z ponuky si vyberieme požadovanú syntax a aplikujeme tlačidlom „Použiť“ (Obr. 39).



Obr. 39 Zvýrazňovanie syntaxe

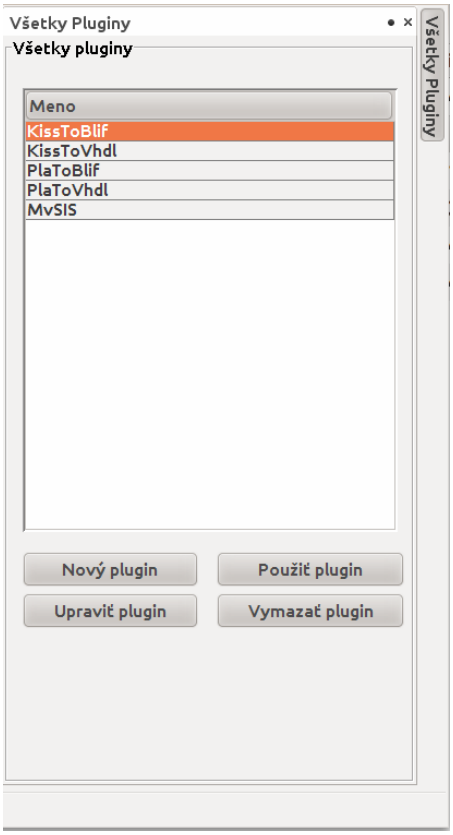
8.1.2 Časť pluginov

Táto časť systému je určená na prácu so vstupnými súbormi(resp. načítaného súboru do textového editora) pomocou tzv. pluginov. Taktiež nám systém umožňuje editovať existujúce pluginy, vytvárať nové alebo odstraňovať nežiadúce. Skratka na zobrazenie Alt+P.

Správa pluginov

V hornej časti okna máme možnosť „Zobraz pluginy“. Po vybratí a kliknutí na túto položku sa nám zobrazí zoznam existujúcich pluginov (Obr. 40). Nasledujúce okno nám okrem zobrazenia existujúcich pluginov ponúkne nasledujúce možnosti:

- vytvoriť nový plugin – tlačidlo „Nový plugin“
- editovať už existujúci plugin – tlačidlo „Uprav plugin“
- vymazať existujúci plugin – tlačidlo „Vymaž plugin“
- použiť plugin – tlačidlo „Použiť plugin“

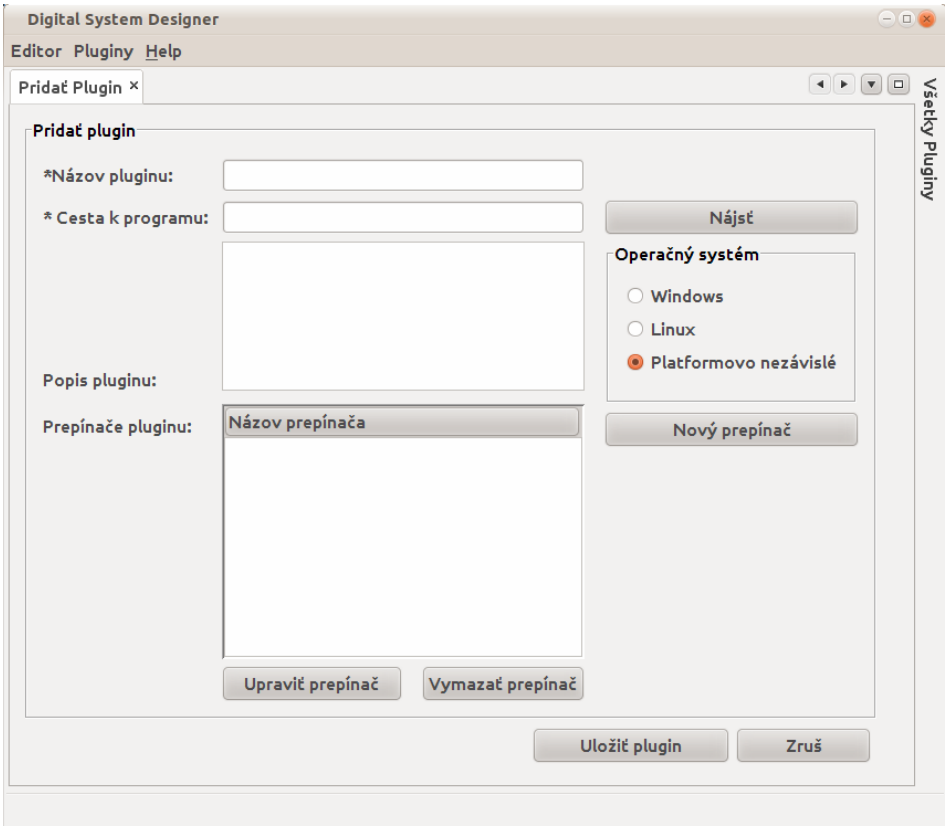


Obr. 40 Zoznam existujúcich pluginov

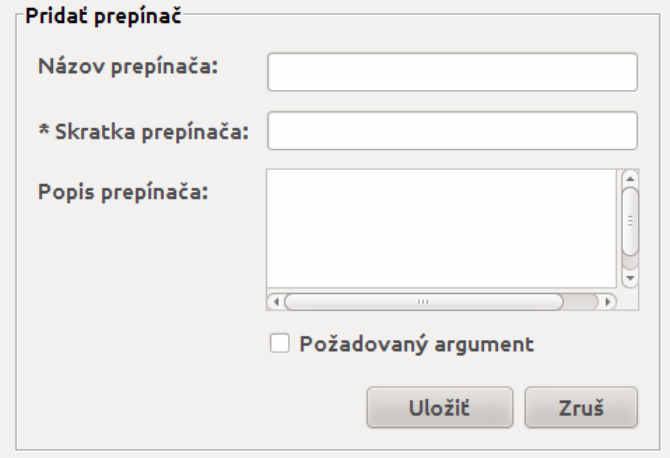
Vytvorenie nového pluginu

Stlačením tlačidla „Nový plugin“ sa nám objaví okno na pridanie nového pluginu (Obr. 41). Obsahuje nasledujúce položky:

- meno pluginu
- cesta k programu(pluginy sú tvorené vykonateľnými súbormi, program atď. – cesta k nim)
- popis – krátky popis pluginu
- Windows/Linux/Platformovo nezávislé – používateľ musí vedieť pod akým systémom je daný program na obsluhu(plugin) funkčný
- prepínače – používateľ pridá do pluginu prepínače programu, ktoré chce využívať v našom systéme (Obr. 42). Zadá celý názov prepínaču napr. „--program“, potom skratku „-p“ a nakoniec jeho popis(názov prepínača a názov skratky funguje aj bez pomlčiek napr. názov „program“ a skratka „p“). Taktiež môže určiť či daný prepínač bude požadovať argument(zaškrtnutím „Požadovaný argument“). Tento argument následne môžeme uložiť tlačidlom „Uložiť“ alebo zrušiť tlačidlom „Zrušiť“
- tlačidlá pre úpravu alebo odstránenie už existujúcich prepínačov
- tlačidlo „Uložiť plugin“ a tlačidlo „Zrušiť“(zrušenie vytvárania nového pluginu)



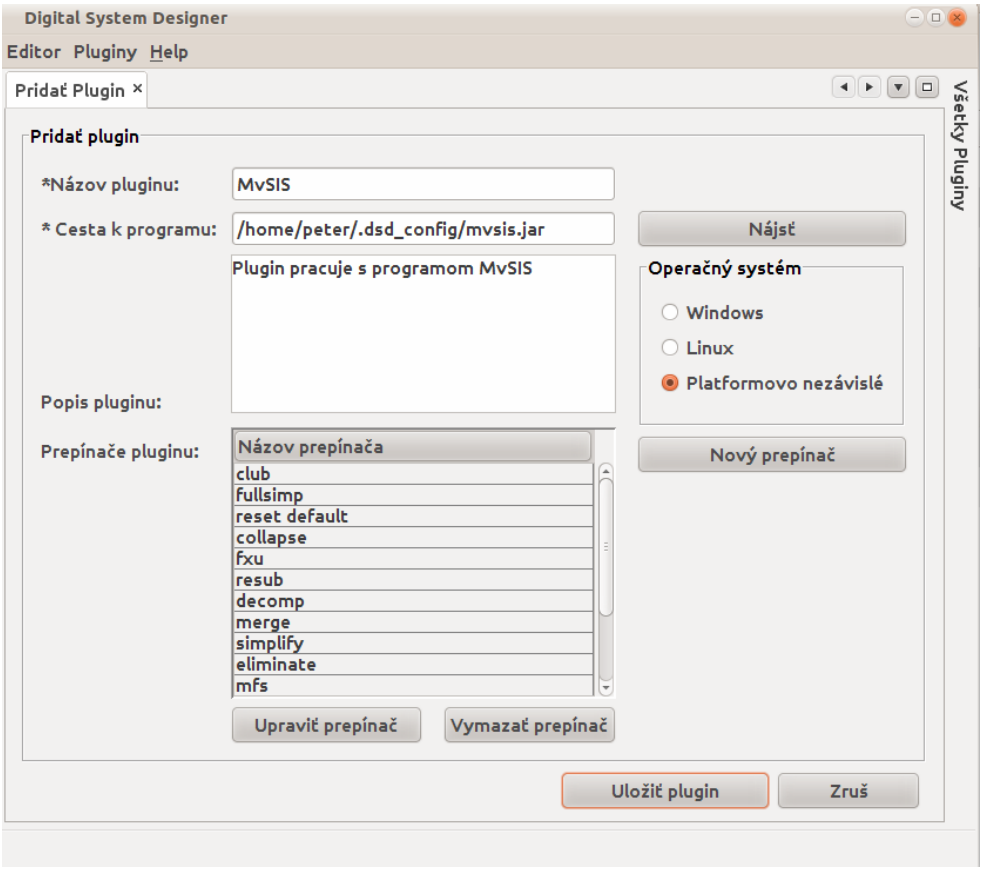
Obr. 41 Pridávanie pluginov



Obr. 42 Pridávanie prepínačov pluginu

Editovanie pluginu

Zo zoznamu pluginov vyberieme existujúci plugin a vyberieme možnosť „Uprav plugin“. Následne sa nám zobrazí rovnaké okno ako pri vytváraní nového pluginu s tým rozdielom, že jednotlivé položky sú už vyplnené, prípadne zaškrtnuté. Plugin môžeme zmeniť a následne uložiť (Obr. 43).



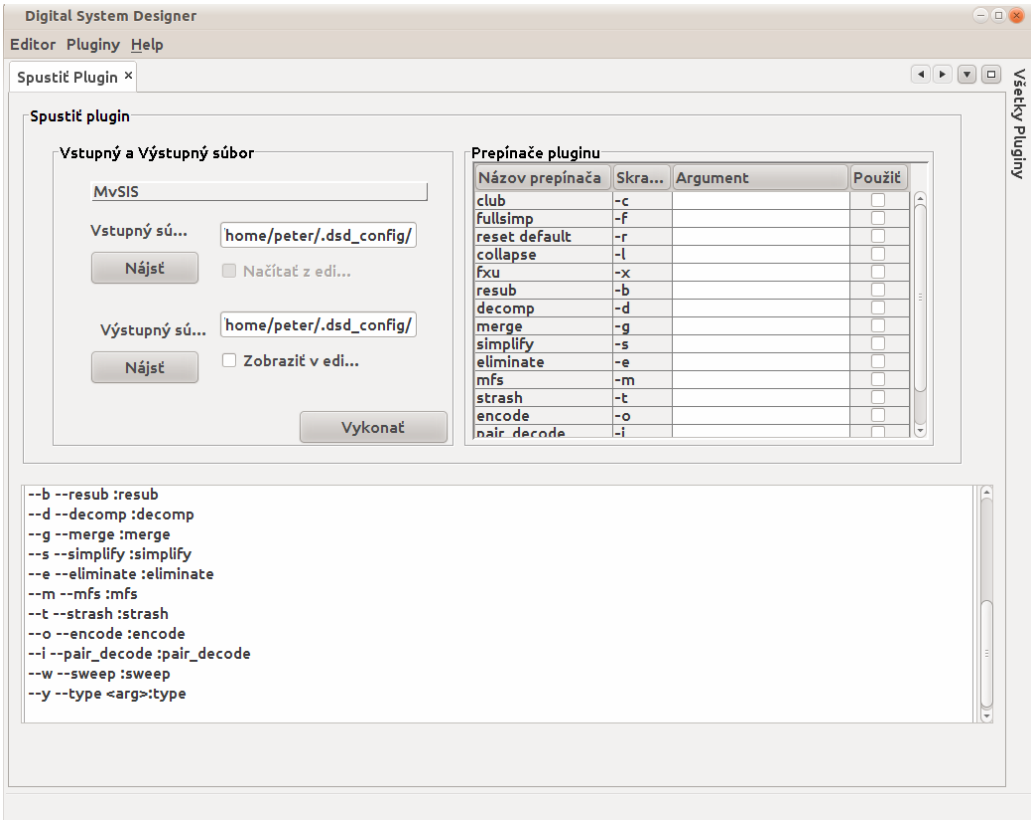
Obr. 43 Editovanie pluginu

Vymazanie pluginu

Tlačidlom „Vymaž plugin“ vymažeme vybratý plugin (Obr. 40).

Používanie pluginov

Tlačidlom „Použiť plugin“ (je prístupne aj z editora (Obr. 36) alebo zo zoznamu pluginov (Obr. 40)) aplikujeme plugin na súbor načítaný v editori alebo na súbor, ku ktorému zadáme cestu (Obr. 44). Pri používaní pluginov používateľ musí vedieť, čo dané pluginy robia, pri nesprávnych vstupoch pluginu systém ohlási, že používateľ používa nesprávne vstupy pluginu alebo nesprávny plugin.



Obr. 44 Používanie pluginu

Položky:

- Vstupný súbor – zadáme cestu k vstupnému súboru, nezadávame ak načítavame súbor z editora
- Výstupný súbor – zadáme cestu k súboru kam uložíme výsledok po použití pluginu
- Zobraziť v editori – zaškrtnutím zobrazíme výsledok do textového editora
- Prepínače – k danému pluginu môžeme použiť aj prepínače. V pravej časti si vyberieme požadované prepínače. Ak majú požadovaný argument, tak musíme vybrať aj tento.
- Popis – ku každému pluginu je určený krátky popis jeho vlastností a funkcionality(spodná časť okna)

Vykonať – tlačidlo,