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

Tímový projekt 1

Prostredie pre návrh digitálnych systémov
(Digital System Designer)

Dokumentácia pre zimný 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 1. počas zimného semestra k 11.11.2010. Dokumet 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

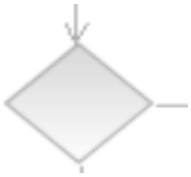
1.5 Použité pojmy a skratky

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)
SLIF	Specification-Level Intermediate Format (špecifikačný formát komponentov)
FPGA	Field Programable Gate Array
LUT	Look-Up Table (tabuľka pravdivostných hodnôt)
PLA	Programmable Logic Array (programovateľné zariadenie na kombinačné obvody)
KISS	Keep It Simple Stupid (tabuľkový formát pre zápis logických obvodov)
VHSIC	Very High Speed Integrated Circuit (vysokorýchlostný integrovaný obvod)
VHDL	VHSIC Hardware Description Language (hardvérový opisný jazyk)
FSM	Finish State Machine (konečný stavový automat)
EQN	Equation Format (rovnicový súborový formát)

1.6 Použitá notácia

V dokumente sú použité nasledovné komponenty diagramov:

Štart procesu



Rozhodovací blok

Proces

Koniec procesu

2 Analýza

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, BDD) 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

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

K úplnej definícií funkcie je potreba 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

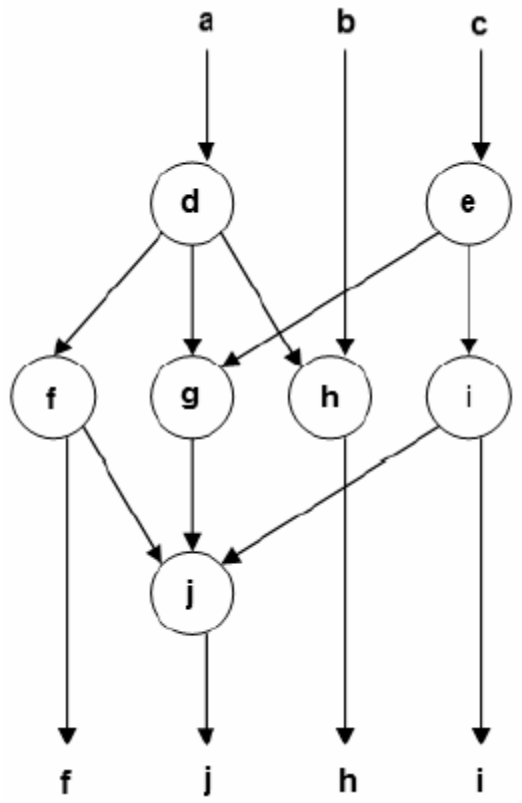
A výsledný súbor PLA bude vypadať nasledovne:

```
.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 IP 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. x5 = y

Pri viacerých riadkov v tabuľke tvoríme súčty súčinov. Každý súbor BLIF je zakončený .end.

```
.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
```

Obr.2 Ukážka BLIF súboru

Logické členy

Ako sme už hovorili BLIF popisuje kombinačné aj sekvenčné obvody. Okrem toho, že si môžeme logické členy samy zadefinovať, tak tento formát umožňuje používať aj nejaké 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>

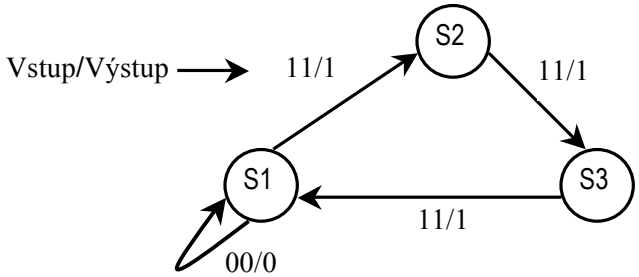
Mená stavov sú špecifikované ľubovoľnými reťazcami znakov bez medzier. Ak nezáleží na mene stavu používame znak '*' (don't care). Vstupy a výstupy sú špecifikované ako tri-stavové vektory. Ak je počet výstupov v hlavičke 0 výstupy by nemali byť špecifikované. Tri-stavový vektor je reťazec nasledujúcich 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. Na nasledujúcom obrázku je príklad 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
```

Obr.3 Príklad formátu KISS súboru

Príklad

Máme nasledujúci stavový diagram:



Obr.4 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

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 control dataflow grafu sa 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. Je to systémové návrhové prostredie, ktoré môže byt rozšírené na zvládnutie mnohých problémov. [4]

Výhody

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

Nevýhody

- Nereprezentuje operácie s jemnou granularitou.
- Nie je simulovateľný. Preto lebo 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.

2.1.6 BDS systém

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]

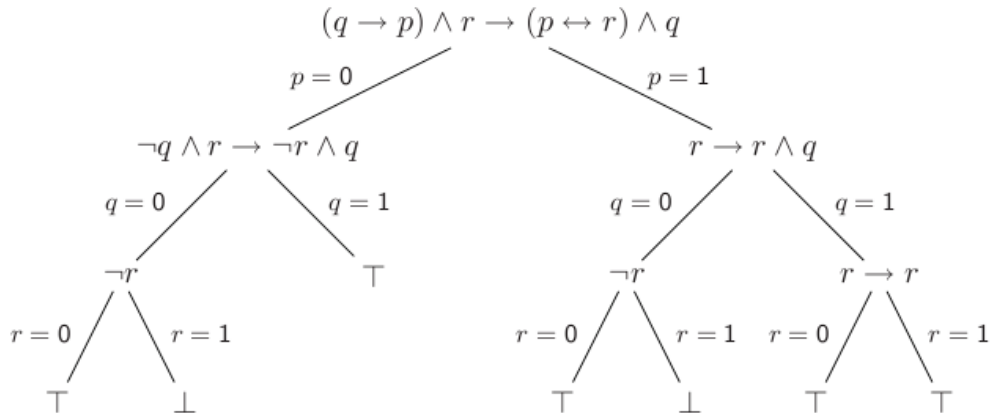
Pri analýze BDS systému najskôr popíšeme BDD, ich tvorbu a vlastnosti. Priblížime princíp dekompozície logickej funkcie na základe BDD a spôsob akým BDS túto dekompozíciu realizuje.

Binárne rozhodovacie diagramy

BDD vzniká spôsobom, ktorý si ukážeme na konkrétnej logickej funkcii. Majme funkciu:

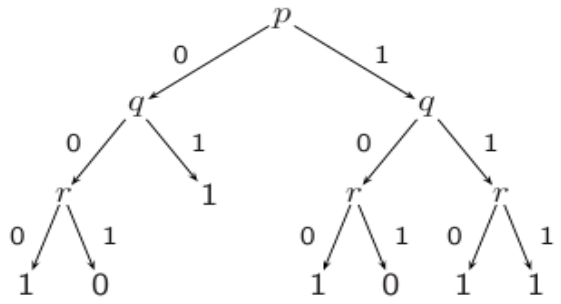
$$f = (q \rightarrow p) \wedge r \rightarrow (p \leftrightarrow r) \wedge q$$

Spôsob, akým z danej funkcie vytvoríme BDD je nasledovný:
Vytvoríme binárny strom pre výpočet pravdivostných hodnôt funkcie (obr.5).



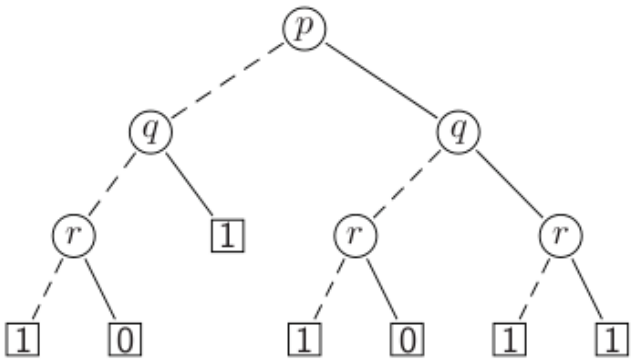
Obr.5 Binárny strom pre výpočet pravdivostných hodnôt

Výmenou symbolov $\bot$ a $\top$ za 0 a 1, výmenou funkcií v interných uzloch za premennú použitú pri rozdelovaní v tomto uzle a označením šípiek symbolmi 0 a 1 dostávame binárny rozhodovací strom (obr.6).



Obr.6 Binárny rozhodovací strom

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.7).

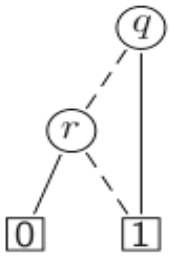


Obr.7 Kompaktná reprezentácia binárneho stromu

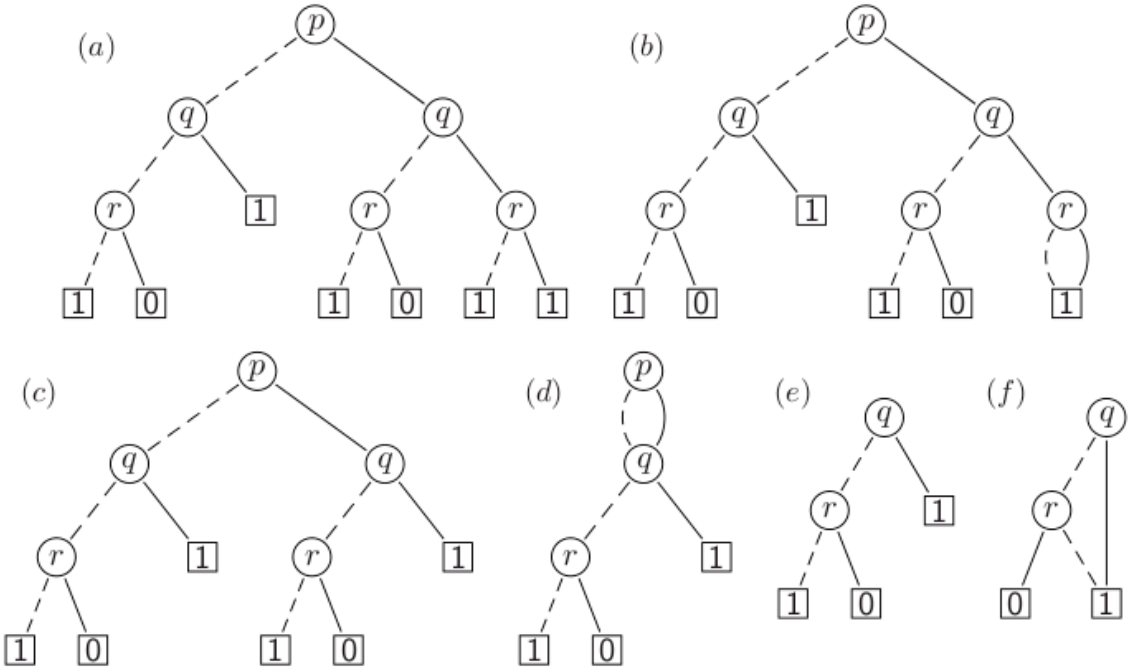
BDD z binárneho rozhodovacieho stromu vznikne nasledujúcimi transformáciami:
Eliminácia redundantných testov: Ak existuje uzol, ktorého pravý a ľavý podstrom je ten istý podstrom, takýto uzol môžeme odstrániť. Spojenie izomorfných podstromov: Ak dva podstromy, ktoré majú korene každý v inom uzle sú izomorfné, 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.8). Znázornenie použitých transformácií je na obr.9.



Obr.8 Binárny rozhodovací diagram



Obr.9 Použité transformácie

Detailný opis transformácií z obr.9:

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.

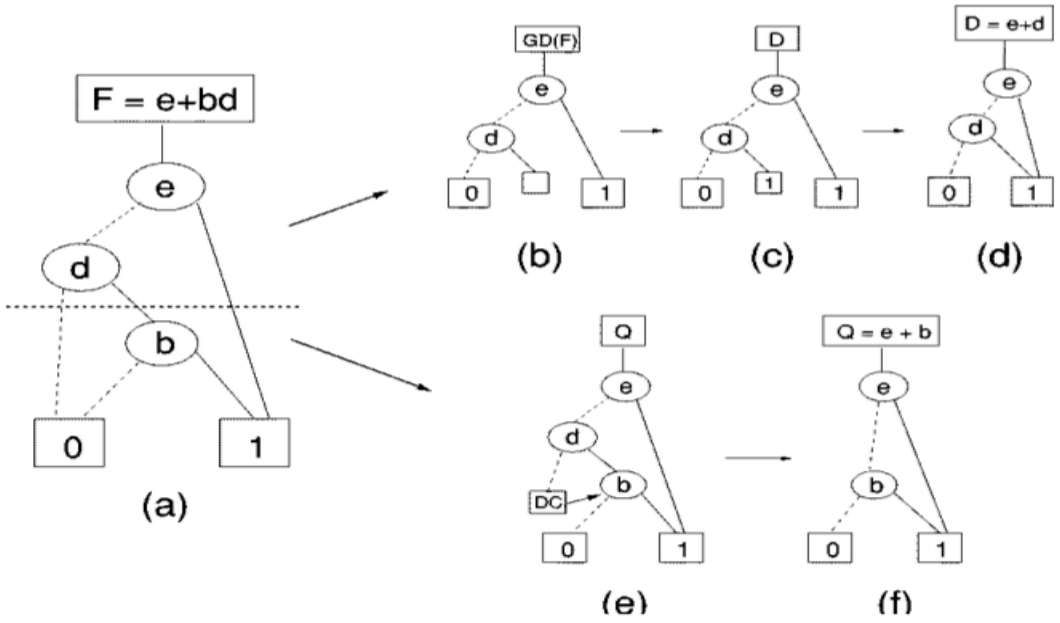
Z bodu (e) do (f): Spojenie izomorfných podstromov v uzle 1.

Dekompozícia BDD

V nasledujúcej časti sa bližšie pozrieme na konjuktívnu a disjunktívnu dekompozíciu BDD. Pricíp 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

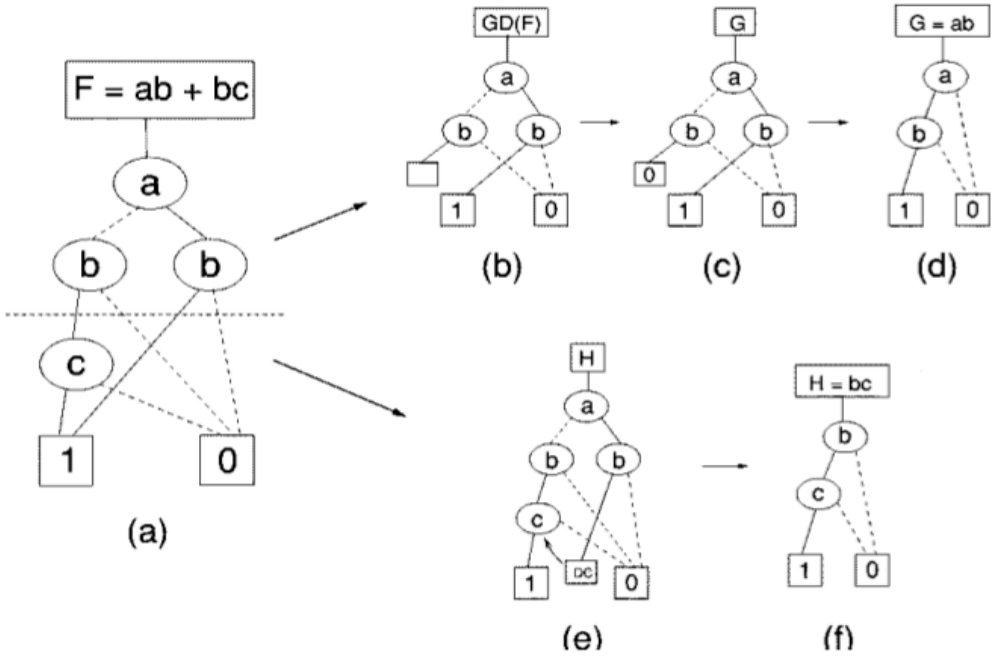
Z definície vyplýva, že boolova funkcia F má konjuktívnu dekompozíciu, ak môže byť reprezentovaná pomocou F = D . Q kde D je boolov deliteľ a Q je kvocient F v takejto dekompozícii. Majme rez, ktorý rozdelí BDD na množiny D a (V – VD). Časť grafu patriacej do množiny VD bude tvoriť nový graf, kde hrana 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)). Boolov 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.10.



Obr.10 Príklad konjuktívnej dekompozície

Disjunktívna dekompozícia

Z definície vyplýva, že boolova 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.
Boolov 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.11.



Obr.11 Príklad disjunktívnej dekompozície

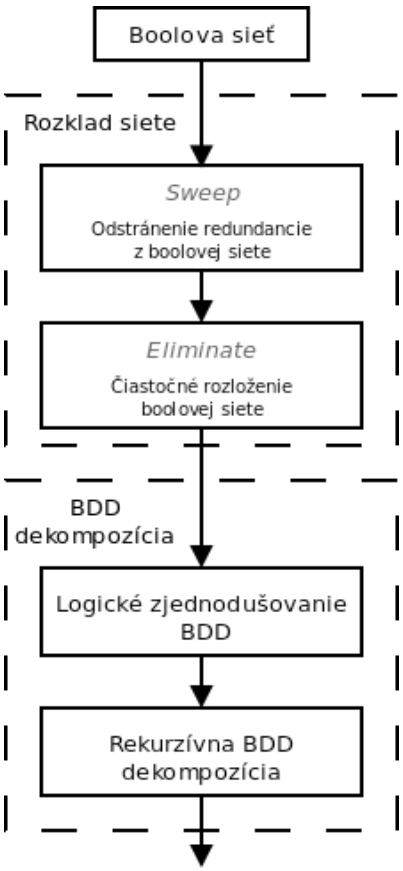
Realizácia dekompozície pomocou BDS

Pri dekompozícií vychádzame z danej boolovej siete. Prvým krokom je odstránenie redundancií v boolovej sieti. Pri tomto kroku sa nerealizuje žiadna logická dekompozícia ale je dôležitý v príprave boolovej siete na následnú dekompozíciu. Spolu s odstraňovaním konštánt sú v tomto

kroku identifikované funkcionálne ekvivalentné uzly, ktoré sú následne z boolovej siete odstránené. Táto procedúra výrazne znižuje zložitosť dekompozície.

Aplikovať logickú dekompozíciu na celú boolovu sieť pomocou globálnej BDD reprezentácie nemusí byť praktické pri rozsiahlych systémoch. Preto v ďalšom kroku sa boolova 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 znázornený na obr.12.



Obr.12 Proces dekompozície pomocou BDS

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 boolovu sieť, ktorá sa následne podľa opísaného procesu rozloží, zjednoduší a dekomponuje. Program môže byť spustený s rôznymi prepínačmi pre nastavenie parametrov pri dekompozícii.

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. Uvádzam úzky výber niektorých z najpoužívanejších simulátorov:

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.

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.13 Porovnanie simulátorov Petriho sietí

2.3 Textové editory

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).

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.

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:

- **Zvýrazňovanie syntaxu** Ide pre mňa 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

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-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 and cktmeno.final.blif. Všetky štatističke informácie ukladá do súboru BDS.run.

2.4.3 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. Aplikácia podporuje vstupné formáty BLIF a PLA, umožňuje kvalitnú a veľmi rýchlu dekompozíciu. Po spustení MVSIS máme na výber niekoľko typov príkazov, ktoré môžeme rozdeliť do nasledovných podskupín:

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*

2.5 Zhodnotenie analýzy

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.

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 boolovu sieť, ktorá sa následne podľa opísaného procesu rozloží, zjednoduší a dekomponuje. Program môže byť spustený s rôznymi prepínačmi pre nastavenie parametrov pri dekompozícii. Výstup z programu po logickej dekompozícií je zapísaný do BLIF formátu, DOT a rovnicovej reprezentácie.

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.

Z textových editorov určených pre tvorbu rôznych druhov kódu sme rovnako vybrali len tie najpoužívanejšie. Sú jednoduché, intuitívne a podporujú editovanie veľkého množstva súborových formátov.

Podobných existujúcich programov určených na návrh digitálnych systémov sme pri štúdiu materiálov veľa nenašli. Systém Blif2vhdl slúži na transformáciu súborov BLIF na formát VHDL. Systém BDS-PGA je vlastne len rozšírená verzia už spomínaného systému BDS. Najlepší systém na návrh 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.

3 Návrh riešenia

3.1 Špecifikácia požiadaviek

3.1.1 Funkcionálne požiadavky

Funkcionnálne požiadavky na 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

3.1.2 Modularita

Na základe existencie rôznych nástrojov, ktoré pracujú s digitálnimi systémami, a ktoré sme bližšie analyzovali v predchádzajúcej kapitole, implementovaný systém musí byť v čo najvaäčšej miere modulárny tak, aby bolo možné tieto nástroje prostredníctvom systému používať. To znamená, že musí byť vytvorený postup, ktorý bude umožňovať špecifikovaným 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. Takto aj napriek skutočnosti, že rôzne nástroje majú rôzne vlastnosti, na základe takéhoto opisu môžu byť do systému 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.

3.1.3 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 potup 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.

3.1.4 Škálovateľnosť

V analýze sme opísali rozne 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 nie len zobraziť a upraviť tento zápis ale musí vedieť aj zvýraznňovať 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 systém musí umožňovať modifikovať tieto vstupy pre experimentovanie s príslušnou transformáciou. Systém musí podporovať editor na vykresľovanie a prípadnú úpravu rôznych reprezentácií logických obvodov.

3.1.5 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.

3.1.6 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
- Zobrazovať grafy logických obvodov
- Upravovať grafy logických obvodov

3.1.7 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

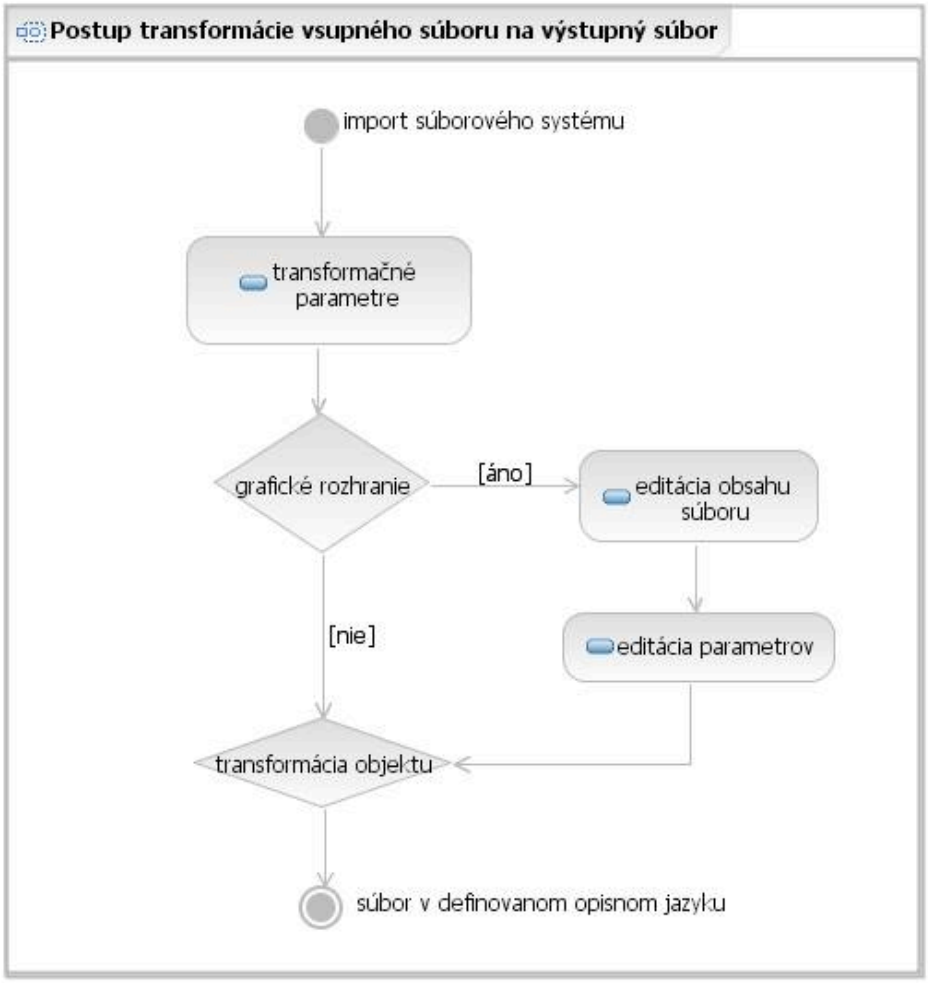
vykonávaní niektorého z programov a tento program by prestal reagovať, systém musí tento stav detegovať a tento program ukončiť aby nedošlo k obmedzeniu činnosti celého systému.

3.2 Hrubý 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 bubú rôzné typy súborových systémov. Tieto súbory môžu byť v aplikácií editované pomocou grafického rozhrania[1] podľa používateľových požiadaviek. Následne sa vykoná transformácia do opisného jazyka, ktorý si sám používateľ zvolí. Celá základná procedúra, ktorú bude aplikácia vykonávať, je znázornená na nasledujúcom obrázku.

[1] Ak je grafické rozhranie podporované operačným systémom.



Obr.14 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

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

3.2.2 Grafická nadstavba

Grafické rozhranie bude slúžiť používateľovi na zjednodušenie práce s programom. Táto nadstavba bude generovať používateľom zvolené parametre a následne volať funkcie z jadra aplikácie. Možností, ktoré bude rozhranie poskytovať sú:

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

3.2.3 Spustenie aplikácie

Spustenie samotnej aplikácie bude riešené hneď niekoľkými spôsobmi, v závislosti od prostredia a voľby operačného systému. Voľby môžu byť nasledovné:

- **Cez príkazový riadok**
    - jadro aplikácie – tento spôsob bude preferovaný pre používateľov, ktorí nebudú mať možnosť spustenia grafického užívateľského prostredia, alebo pre používateľov s rozsiahlejšími skúsenosťami o danej problematike. Pomocou parametrov definovaných pri spustení sa bude automaticky vykonávať celá logika programu v závislosti od nastavených parametrov.
    - grafické rozhranie – cez zadanie názvu aplikácie sa spustí grafické rozhranie aplikácie.
- **Kliknutím na ikonu**[2]

[2] Vykoná sa to isté ako pri spustení grafického rozhrania cez príkazový riadok.

- **Cez webové rozhranie**[3]

3.2.4 Návth vstupov

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

3.2.5 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 pokusi sa určiť, kde nastala chyba. Ak sú všetky parametre, aj vstupný súbor v poriadku, vykoná sa transformácia.

3.2.6 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 spustitelný 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.

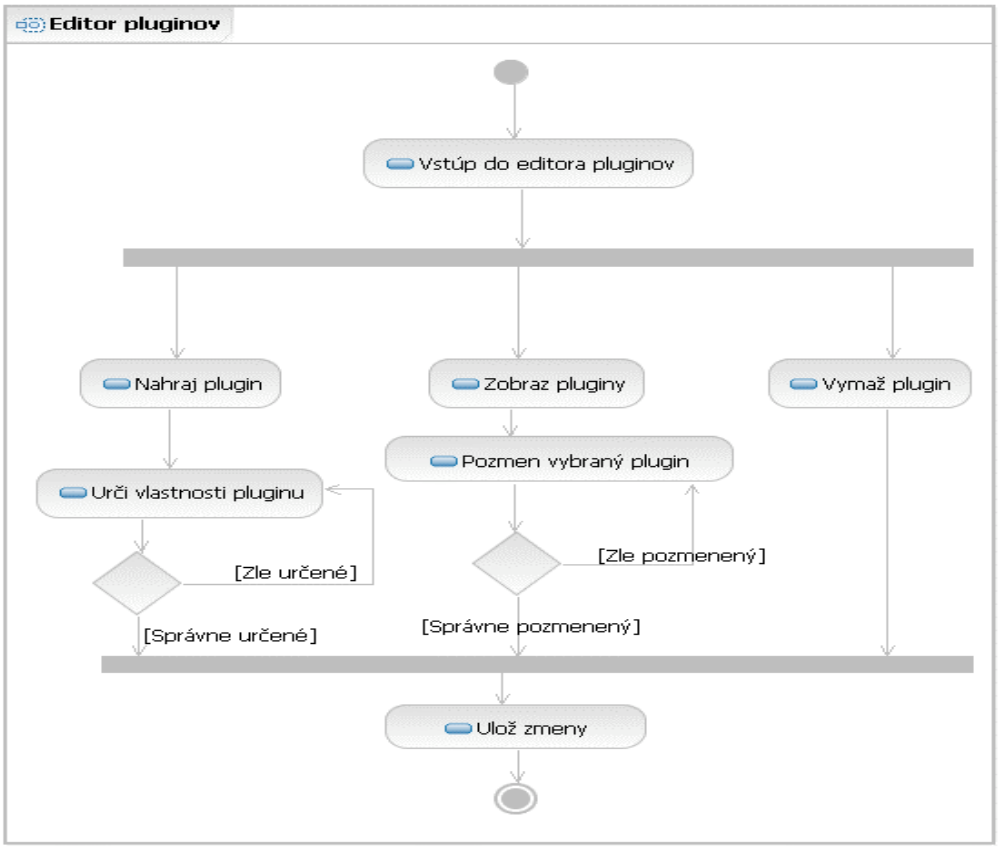
[3] Táto možnosť je alternatívna, závisi na konkrétnejších požiadavkách od zadávateľa projektu.

3.2.7 Pridávanie pluginov

Systém bude obsahovať používateľské rozhranie určené iba na rozširovanie, 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 vstup, výstup a parametre, aby systém neskoršie vedel určiť, ktorý program(plugin) má použiť.

3.2.8 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.



Obr.15 Integrácia pluginov

4 Použité zdroje

[1] Houžvička, K: Rychlý simulátor kombinačních obvodů, Bakalárska práca, 2010.

[2] University of California Berkeley:Berkeley Logic Interchange Format, dostupné na internete http://www.cs.uic.edu/~jlillis/courses/cs594/spring05/blif.pdf, (7.10.2010).

[3] E. M. Sentovich *et. al.*, "SIS: A System for Sequential Circuit Synthesis". Dept. of Electrical Engineering and Computer Science, University of California, Berkeley, USA, Technical Report CA 94720, 1992.

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

[5]Cieselsky, M. et al.: BDS: A BDD-Based Logic Optimization System, dostupné na internete http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.25.7158&rep=rep1&type=pdf, (10.10.2010).