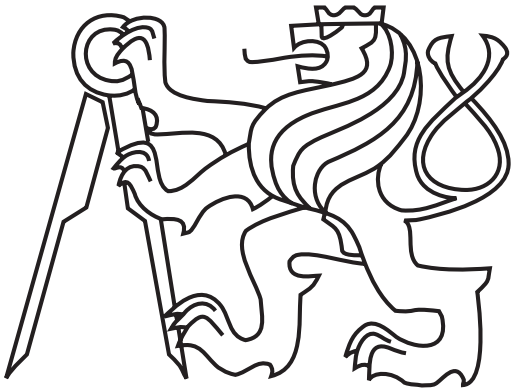
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

Dekompozice logických obvodů se zaměřením na jejich diagnostiku

Pavel Směšný

Vedoucí práce: Ing. Petr Fišer, Ph.D.

Studijní program: Elektrotechnika a informatika, strukturovaný

Obor: Výpočetní technika

Červen 2009

Poděkování

Chtěl bych poděkovat své rodině za podporu a především svému vedoucímu bakalářské práce Ing. Petru Fišerovi za odbornou pomoc a trpělivost.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 9.7.2009 .....

Abstract

The goal of this bachelor project is design and implementation of an application partitioning a combinatorial circuit formatted as netlist into more parts in a purpose of testing. So the most important thing is to choose a suitable algorithm for decomposing of a circuit represented by a graph with regard to minimum cut and good testability.

Abstrakt

Cílem této bakalářská práce je návrh a implementace aplikace, jež rozdělí kombinační obvod zadaný netlistem na více částí s ohledem na testování. Zabývá se tedy volbou vhodného algoritmu pro dekompozici obvodu jako grafu s ohledem na co nejmenší řez a tedy i dobrou testovatelnost následného obvodu.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

V současné době s rostoucími požadavky na výkon a větší funkčnost výpočetní techniky se zvětšuje počet tranzistorů a hradel v logických obvodech (VLSI). Na druhou stranu je snaha včlenit tyto celky na stále a stále menší celky ať už kvůli vyšší frekvenci či menší spotřebě. Výroba je tak hnána na hranici dané technologie, zvětšuje se složitost obvodů a i počet vstupů, přičemž roste i jejich důležitost - řídí jaderné elektrárny, kontrolují letadla apod. Všechny tyto faktory směřují k jedinému - obvody je nutné testovat. Při jejich velikosti a počtu vstupů však vzniká velký problém z hlediska obrovského množství testovaných kombinací a následné velké časové i výpočetní náročnosti. Proto vznikají různé algoritmy, jež se snaží otestovat jen některé kombinace vstupů, které mají největší pravděpodobnost odhalení chyby ve výrobě či návrhu. Ovšem to je jen částečné řešení. Stále zůstává velký problém, který je nutno dekomponovat na menší tím, že celý obvod se rozdělí na menší celky a ty jsou kontrolovány zvlášť. Zmenší se tak značně celá náročnost. V praxi se používá např. BIST (Built-in Self-test) čili vestavěná diagnostika, kdy jsou kontrolní obvody součástí obvodu testovaného. A právě dekomponováním obvodů na menší se zabývá tato práce.

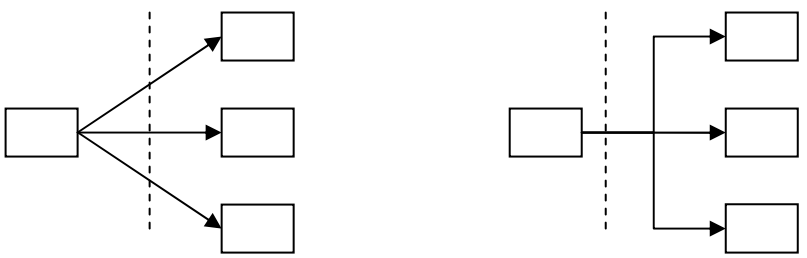
Kapitola 2

Analýza dekompozice

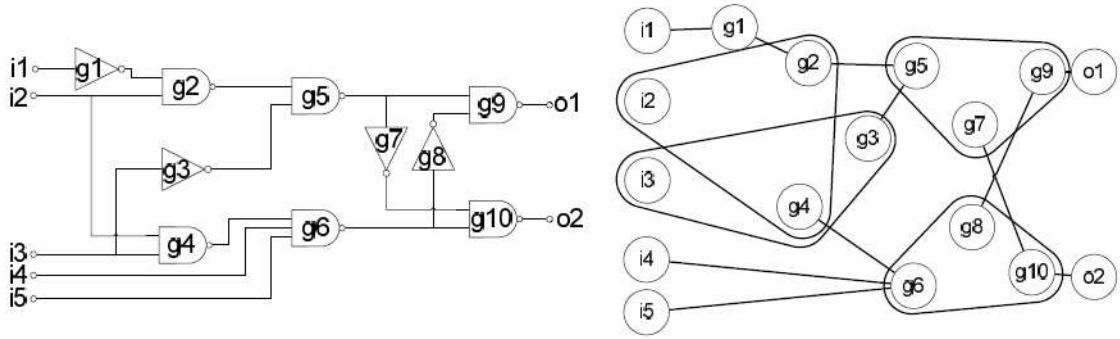
Cílem dekompozice je rozdělit vstupní obvod na menší, které mají co nejméně vstupů, protože vysoký počet vstupů je z hlediska testování náročný. Vstupní hodnoty obvodu mohou nabírat dvou logických hodnot. Pokud bychom měly testovat všechny možné kombinace a x je počet vstupů, získáme 2^x testovacích vektorů. Při $x = 100$ získáváme už $1{,}3\times10^{30}$ vektorů.

Logický obvod lze chápat z hlediska teorie algoritmů jako orientovaný graf G, přičemž jednotlivá hradla jsou v grafové terminologii uzly V a dráty jsou hrany E. Celý obvod lze tak definovat jako $G = (V, E)$. Pokud počet uzlů $V = |n|$, pak k-cestná dekompozice jest rozdělení V do podskupin V_1, V_2, ... , V_k, přičemž $V_i \cup V_j = \theta$ for $i \neq j$, $|V_i| = n/k$ a $\bigcup_i V_i = V$ a hranový řez je minimalizován. Hranovým řezem myslíme všechny hrany, které koincidují s uzly různých podskupin.

Zajímavou vlastností logického obvodu je fakt, že výstup z jednoho hradla může jít do vstupů více hradel. Vzniká tak hyperhrana, tj. hrana spojující více uzlů. Pro dekompozici je to významný fakt, protože tato vlastnost snižuje cenu řezu (obr. 1). Takovéto grafy nazýváme hypergrafy (obr. 2).



obr. 1: řez hranami (vlevo) vs. řez hyperhranou (vpravo)



obr. 2: logický obvod a jeho znázornění jako hypergrafu (zakroužkování uzlů znamená, že jsou spojeny jednou hyperhranou místo několika hranami)

Samotná dekompozice s minimálním řezem je však NP-úplný problém [6]. Tyto úlohy jsou nedeterministicky řešitelné v polynomiálně omezeném čase [1]. Existuje sice pro ně algoritmus řešení, např. probíráním všech možností, ale ten dovoluje dosáhnout výsledku pouze pro omezený rozsah vstupních dat. Máme-li graf G s n uzly rozdělit do podmnožin o velikosti p, kde $kp = n$, pak existuje $\binom{n}{p}$ způsobů pro vybrání první podmnožiny, $\binom{n-p}{p}$ pro vybrání druhé podmnožiny atd. Takže počet možných řešení je [2]

$$\frac{1}{k!}\binom{n}{p}\binom{n-p}{p}\cdots\binom{2p}{p}\binom{p}{p}$$

Pro většinu hodnot vzniká obrovský počet možností. Např. už při $n = 40$, $p = 10$ ($k = 4$) vychází větší než 10^{20}. Protože je ale tato dekompozice nutná v mnoha oborech, byly vyvinuty heuristické metody, které jsou rychlé a poskytují dobré výsledky, tj. snaží se optimálnímu řešení pouze přiblížit, na exaktní řešení v současné době neexistuje přiměřený výpočetní výkon.

Kapitola 3

Dekompoziční (partitioning) algoritmy

Tato kapitola obsahuje seznam nejčastěji používaných algoritmů, jejich popis a posouzení vhodnosti k dekompozici.

3.1 Triviální algoritmy

V souvislosti s těmito algoritmy se v žádném případě nedá mluvit o kvalitě dekompozice, nicméně jsou zde zmíněny právě, protože se používají jako vstup složitějších algoritmů, které je potřebují jako počáteční dekompozici. Tu se pak snaží následně vylepšit a zmenšit tak cenu řezu.

3.1.1 Random algorithm.

Nejjednodušší a nejhorší cesta k rozdělení grafu. Při dobré náhodnosti by byly tak uzly každé podmnožiny rovnoměrně rozmístěné po celém grafu a cena řezu by byla obrovská. Další nevýhodou je i to, že vylepšení následným algoritmem by bylo daleko výpočetně náročnější než u dalších algoritmů. Jedinou jeho výhodou je nezávislost na volbě počátečního uzlu - vůbec tu není.

3.1.2 Graph growing partitioning algorithm (GGP) [4].

Další jednoduchá forma rozdělení grafu začínající na zvoleném uzlu, od kterého se podmnožina postupně rozrůstá bez ohledu na cenu řezu, dokud nedoroste požadované velikosti a to se stále opakuje, dokud není celý graf rozdělen. Tato metoda je silně závislá na volbě počátečního uzlu. Vylepšením by mohlo být opakování při různé volbě počátečního uzlu, ale ani tak neposkytuje uspokojivé výsledky.

3.1.3 Greedy graph growing partitioning algorithm (GGGP) [4].

Tato metoda vylepšuje předchozí GGP o to, že pro každý uzel je definována cena řezu, jež je spjata s vložením uzlu do rostoucí podmnožiny.

Sousední uzly jsou tedy seřazeny podle této ceny a je vložen uzel s nejmenším či největším vzrůstem ceny řezu. Metoda je tak daleko méně závislá na volbě počátečního uzlu a vykazuje lepší výsledky než předchozí metody.

3.2 Netriviální algoritmy

3.2.1 Spectral bisection (SB).

Tento algoritmus je založen na výpočtu charakteristického vektoru y, tzv. Fiedlerova vektoru, korespondující s druhou největší charakteristickou hodnotou Laplacianova matrixu [4]. Jelikož ale bisekce je pouze 2-cestná dekompozice, je zbytečné se jí dále zabývat. I když je pouze příkladem spektrálních algoritmů, které však obecně nevykazují dobré výsledky [6].

3.2.2 Simmulated annealing (SA) [7].

Simulované žíhání je alternativou k tzv. hladovým algoritmům, jež jsou založeny na přesunech, které daný řez pouze vylepšují. SA pracuje s celým okolím. Vezme náhodný sousední uzel a pokud představuje jeho přesun vylepšení, tak ho provede, pokud ne, tak narozdíl od ostatních algoritmů také přesune, ale pouze s jistou pravděpodobností $e^{\frac{-\Delta}{T}}$, nazývanou Boltzmanovou akceptancí. Δ je cena nového řešení a T je současná hodnota teploty. Ta slouží pro kontrolu konvergence, optimalizace a konečnosti řešení. SA tak konverguje k optimálnímu minimu daného nekonečným počtem přesunů, přičemž teplota T se ochlazuje značně pomalu. Toto řešení je zajímavé z teoretického hlediska, dává totiž lepší řešení, ale z praktického hlediska je příliš výpočetně náročný, pomalý.

3.2.3 Tabu search (TS) [7].

Tabu search je algoritmus obdobný k SA, navíc je vylepšen o tzv. tabu list obsahující seznam nedávných přesunů zrychlující konvergenci k lokálnímu minimu. Proklamovanou výhodou je tedy neztrácení času nad špatnými náhodnými přesuny a nevracení se již k vyřešeným oblastem. I přesto není však příliš rychlý.

3.2.4 Genetic algorithms (GA) [7].

Genetické algoritmy jsou založené na Darwinově teorii přirozené selekce v evoluci, kde vyvolení členové mají větší potomstvo než ti slabí. GA začíná s počáteční populací řešení. V implementaci se objevují různé kombinační a mutační operátory determinující další generaci s ohledem na lepší řešení. Z

praktického hlediska je tento algoritmus nezajímavý, nejde však jen o časovou náročnost, ale i složitost implementace operátorů tak, aby zaručovaly slušné výsledky.

3.2.5 Kernighan-Lin algorithm (KL) [2], Fiduccia-Mattheyses algorithm (FM) [3].

Tyto dva algoritmy patří k iterativně vylepšujícím metodám. Jsou také nejčastěji implementovány. Důvodem je jejich velmi krátký čas běhu, relativně výborné výsledky dekompozice. Jejich jednoduchý princip je nejen vhodný k implementaci, ale je také snadno adaptabilní k různým požadavkům na řešení.

Oba algoritmy potřebují jako vstup již nějakým způsobem dekomponovaný obvod, tj. počáteční dekompozici. Právě toto počáteční rozdělení je kritickým článkem. Čím je lepší, tím méně práce je pak potřeba k vylepšení. Dalším aspektem je, že špatná počáteční dekompozice už se nemusí podařit vylepšit. Kvůli tomuto je nutné celou dekompozici párkrát zopakovat, aby bylo nalezeno co nejoptimálnější řešení.

Principem KL je nalézt takovou dvojici nezamčených uzlů z dvou různých podmnožin tak, aby jejich prohozením v rámci podmnožin se zmenšila cena hranového řezu. Zisk *g* přesunutí jednoho uzlu z podmnožiny jedné do druhé je definován jako

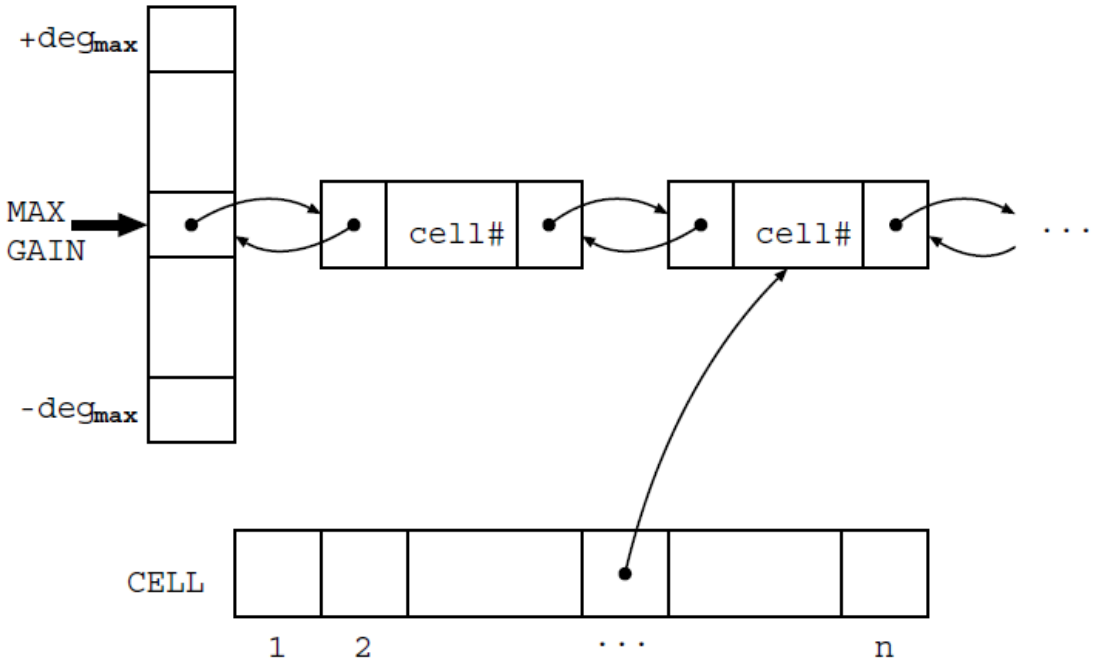
$$g_v = \sum_{(v,u)\in E \wedge P[v]\neq P[u]} w(v,u) - \sum_{(v,u)\in E \wedge P[v]=P[u]} w(v,u)$$

P je počáteční podmnožina grafu *G = (V,E)*, *v* uzel a *w(v,u)* váha hrany *(v,u)*. Pokud je negativní, hranový řez by se zvětšil, pokud pozitivní, tak výměnu provést, uzly zamknout a tuto celou iteraci opakovat do té doby, než už není nalezena žádná taková odpovídající dvojice. V této chvíli je algoritmus ukončen, již je v lokálním minimu a není možné žádné další vylepšení.

Výzkumy ukazují [4], že celý tento běh je nutné opakovat 5x až 10x, aby bylo dosaženo dobré dekompozice, pokud počáteční je „slušná“. Tím se sníží výrazně runtime aplikace, protože v dalších bězích už dochází jen k zanedbatelným vylepšením.

Základní složitost KL je $O(n^3)$ [7], právě proto existují další nadstavby tohoto algoritmu, které ji vylepšují. Z nejlepších je právě Fiduccia-Mattheyses algorithm (FM), který pomocí vhodných datových struktur tuto složitost linearizuje na $O(|E|)$, na hypergrafech je $|E|$ nahrazeno sumou všech vstupů a

výstupů $|i+o|$. Klíčovým pro toto zrychlení je struktura „gain bucket“, jež dovoluje přístup k uzlům s největším ziskem v konstantním čase. Její schéma znázorňuje obrázek 3. Efektivnost této struktury je dána celočíselným vyjádřením zisku a jeho ohraničením shora $+deg_{max}$ a zdola $-deg_{max}$. Na začátku je pro každý uzel vypočítán zisk se složitosti $O(|i+o|)$ a je uložen do struktury. Uzly s nejvyšším ziskem jsou uloženy v položce s hodnotou MAXGAIN. Během iterace je uzel vyjmut z této položky ze spojového seznamu přesunut do druhé podmnožiny. Následně jsou změněny zisky sousedních odemčených uzlů a přesunuty ve struktuře. Pokud zisk nějakého uzlu převýší zisk uzlů v položce MAXGAIN, je toto označení přesunuto na položku uzlu s nejvyšším ziskem. V opačném případě, je-li MAXGAIN prázdná, je dekrementována na nejvyšší neprázdnou položku.



obr. 3: struktura „gain bucket“ pro přístup k nejziskovějším uzlům [3]

Nevýhodou této efektivní struktury je, že nepočítá s vícecestnou dekompozicí [3]. Každému uzlu totiž odpovídá jen jeden zisk, tedy přesun do druhé podmnožiny.

3.3 Závěr analýzy (volba algoritmu)

Z předchozí analýzy je patrné, že nejlepším algoritmem pro implementaci vícecestné dekopmozice hypergrafu je KL právě díky své univerzálnosti, rychlosti, relativní kvalitě výstupu a přizpůsobitelnosti. Možností je také inspirace strukturami FM a zefektivnění běhu aplikace. KL je též přizpůsobitelný požadovaným vlastnostem vyplývajícím z vlastností obvodů přenesených na graf, tj. že každý signálový drát má jeden zdroj a více cílů, tzv. hyperhrana. Pro jeho vstup bude použit GGGP kvůli jeho největší kvalitě z triviálních algoritmů, protože při iniciační dekompozici je už nutné zahrnout počet vstupů, který KL pouze vylepšuje.

Kapitola 4

Implementace

Kapitola obsahuje základní programový nástin aplikace - prostředí, použité datové struktury a klíčové použité algoritmy.

4.1 Volba prostředí

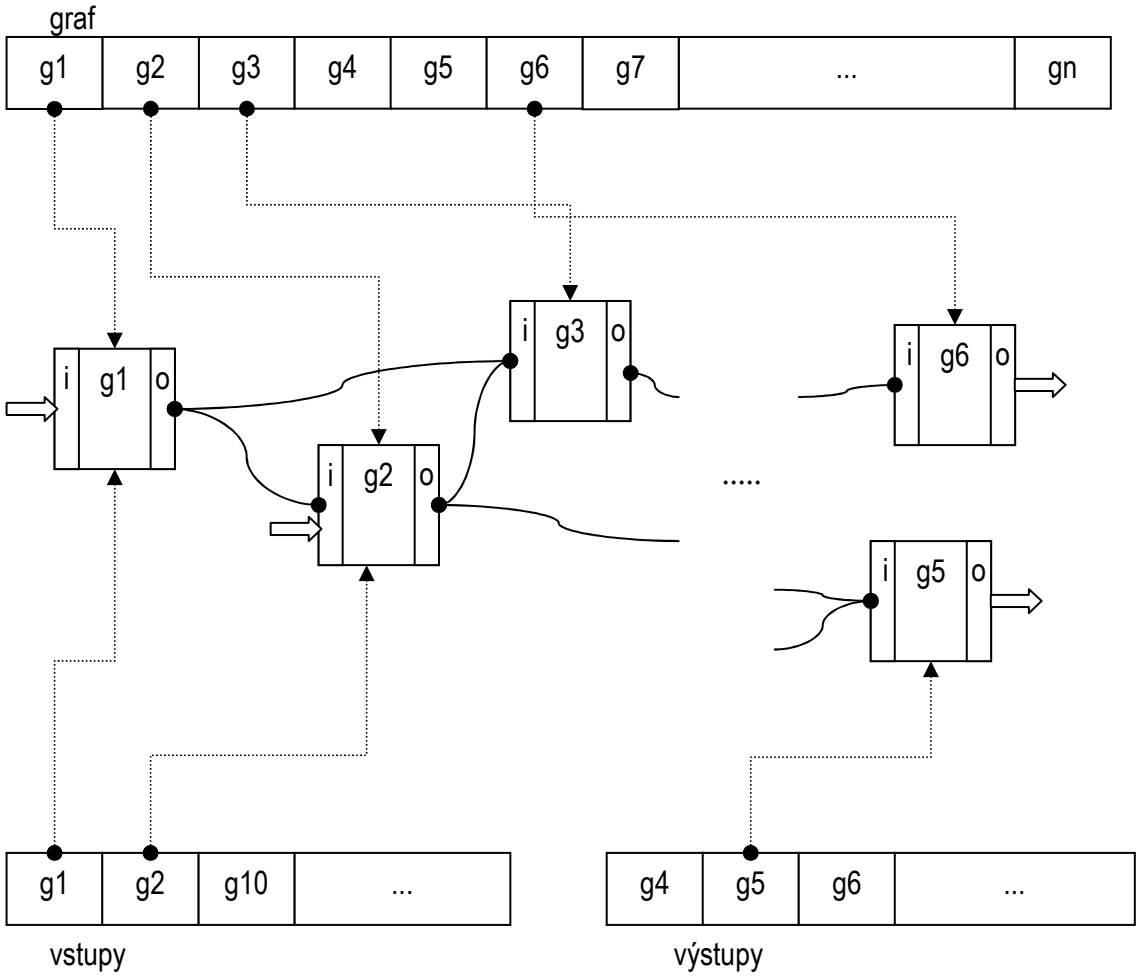
Za vývojové prostřed byl vybrán open-source projekt Bloodshed Dev-C++ (4.9.9.2), tedy plné IDE pro C/C++ obsahující Mingw port pro GCC (GNU Compiler Collection). C++ bylo zvoleno záměrně kvůli efektivitě - umožňuje totiž práci s ukazateli a díky svému rozšíření STL (standard template library) obsahuje i efektivní kontejnery. Další výhodou je přenositelnost - C++ není omezeno operačním systémem.

Další možností by bylo použití javy, jež je však pro tyto účely příliš robustní a neobsahuje paměťové ukazatele. Celý běh aplikace by byl tak o dost pomalejší.

4.2 Reprezentace grafu

Existují dvě základní reprezentace. První je maticová. Může jít např. o matici sousednosti nebo incidence. Tato struktura není ale dynamická a za další při velkém počtu uzlů by vznikaly rozsáhlé matice. Jejich procházení a vyhledávání v nich by bylo časově náročné nehledě na paměťové nároky.

Druhým způsobem jsou moderní objektové dynamické datové struktury pomocí kontejnerů a ukazatelů. Každý uzel má tak vlastní metody a obsahuje všechny informace, které o sobě potřebuje vědět, tj. své jméno, typ hradla, vstupy a výstupy a další podružné informace pro dekompozici. Celou strukturu znázorňuje obrázek 4.

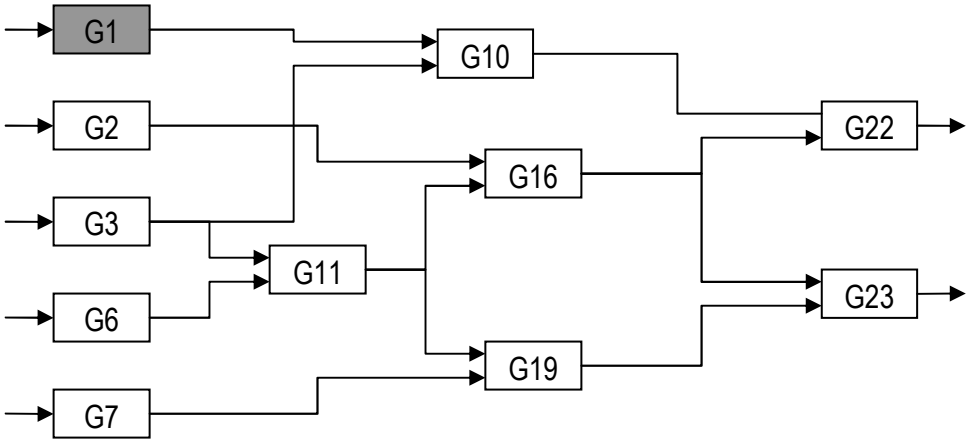


obr. 4: objektová reprezentace grafu (z hlediska datových struktur graf, vstupy a výstupy jsou mapy a i,o jednotlivých uzlů jsou vektory)

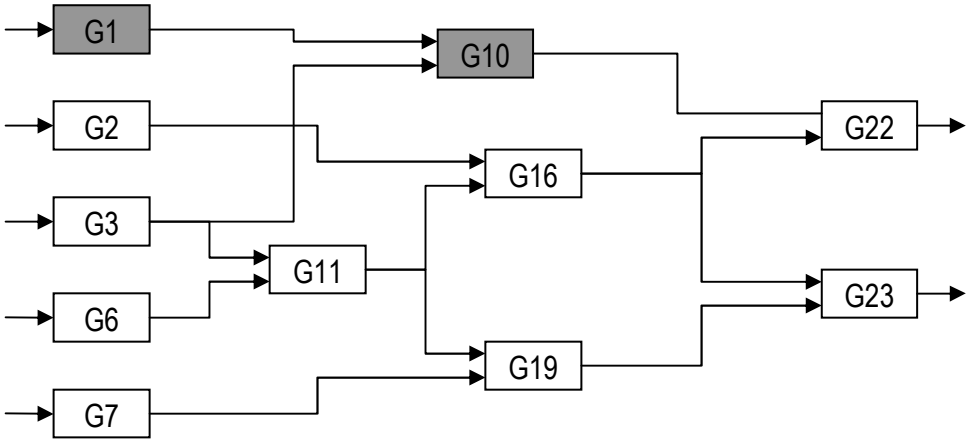
Vzhledem k dynamičnosti této struktury je možné její vytváření již při načítání vstupního souboru. Postupně se tak uzly vkládají do mapy všech uzlů, případně i do seznamu vstupů a výstupů. Vytvoření obousměrných ukazatelů mezi uzly se provede až následně, což zaručuje větší odolnost vůči chybám v souboru - nelze odkazovat na nedefinované uzly.

4.3 Iniciační fáze

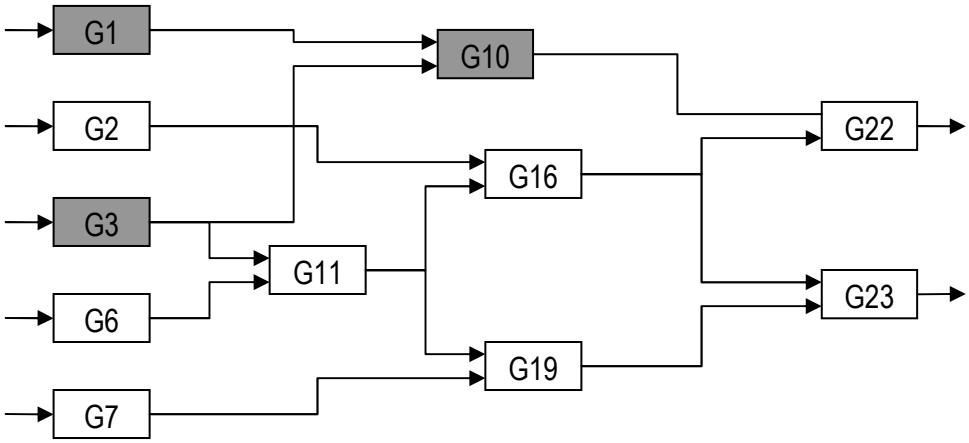
V této fázi probíhá iniciační dekompozice. Je náhodně vybrán počáteční uzel, přiřadí se do první podmnožiny a od něho se začíná podmnožina rozrůstat pomocí GGGP, tj. začne je vytvořena prioritní fronta sousedních uzlů a je přidán ten, která zaručuje nejmenší zvýšení počtu vstupů. Postup GGGP znázorňuje obr. 5. To se provádí tak dlouho, dokud počet vstupů podmnožiny nepřesáhne zadané maximum či počet uzlů v podmnožině není větší než zadané maximum, pak se vytvoří další podskupina a vše se opakuje, dokud nejsou všechny uzly rozděleny do podskupin.



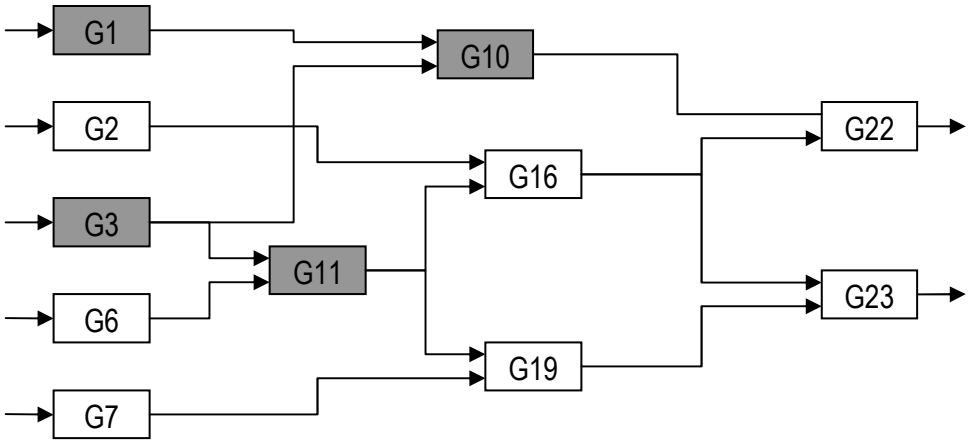
obr. 5.1: zvolen počáteční uzel



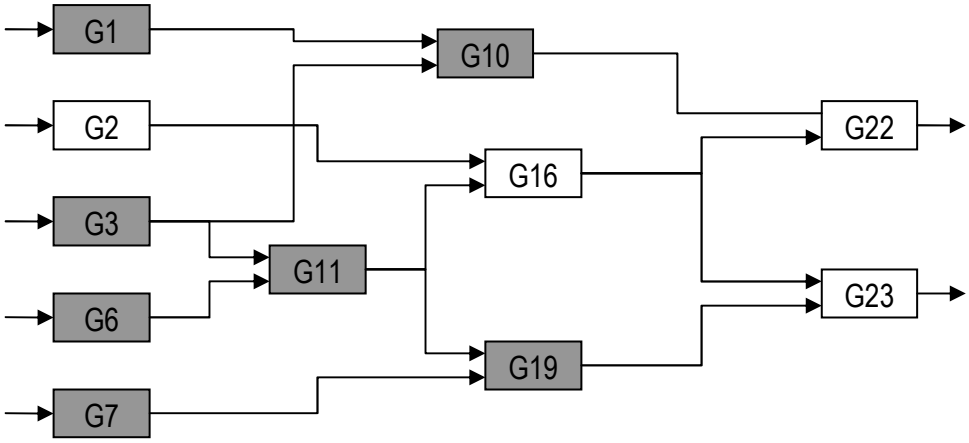
obr. 5.2: přidán G10 - uzel s nejmenším počtem přidaných vstupů



obr. 5.3: přidán G3



obr. 5.4: přidán G11



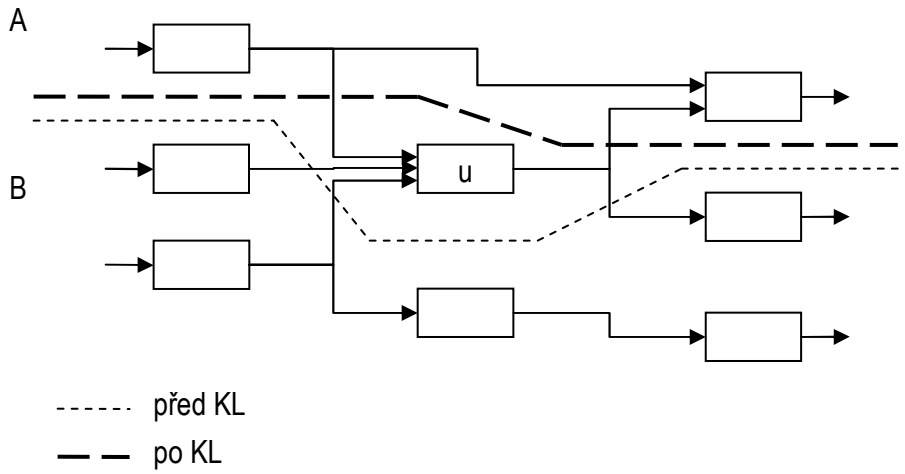
obr. 5.5: přidán G6, G19 a G7 a tím končí první podskupina, další přidání by už znamenalo překročení zadaného limitu vstupů, neoznačené uzly se stanou součástí druhé podskupiny

4.4 Kernighan-Lin

Vstupem KL je dekomponovaný graf z iniciační fáze. KL se jej pouze snaží vylepšit tím, že přehazuje uzly tak, aby se zmenšil hranový řez. Jeho implementovanou variantu ukazuje obr. 5. KL prochází všechny uzly a u každého si spočítá ceny pro podmnožinu, tj. sečte všechny sousedící hrany a hyperhrany vzhledem k podmnožinám. Následně prochází všechny ceny a zjišťuje, jestli uzel patří vzhledem k cenám k dané podmnožině či by se měl přesunout (příklad na obr. 6). Přitom kontroluje, zda-li by přesunutí neporušilo zadané limity pro dekompozici (max. počet hradel, max. počet vstupů). Pokud jsou nalezeny vhodné přesuny, vybere se ten nejlepší a uzel je přesunut. Následně jsou také změněny vlastnosti podskupin a uzlu související s přesunem. Časová složitost jednoho běhu tak je:

$$\mathrm{O}(n \times (i(n) + o(n) + ...)) = \mathrm{O}(n)$$

protože počet vstupů a výstupů uzlů je průměrně konstantní vzhledem k počtu vstupních dat, dají se tak zanedbat a zbyde počet uzlů *n*, tj. lineární časová závislost. Celý KL je však nutno několikrát opakovat, aby se dosáhlo opravdu lokálního minima.



obr. 6: příklad uplatnění KL algoritmu - přesunutím uzlu *u* z podmnožiny *A* do podmnožiny *B* se zmenší hranový řez o 1

```
1   for each node do
2
3     for each node.input do
4       cost[input.subset]+=input.subset.getCost();
5     for each node.output do
6       cost[output.subset]+=output.subset.getCost();
7     //vypočítáni ceny pro jednotlivé podmnožiny
8
9     for each cost do
10      gainMove=computeGain();
11      if ((gainMove>maxGain)
12        &&(countGates[B]+1<limitGates)
13        &&(countInputs[A]+changeGatA<limitInputs)
14        &&(countInputs[B]+changeGatB<limitInputs))
15          maxGain=gainMove;
16    }
17    //vybráni nejlepší možné varianty pro přesun
18
19    if (maxGain) swap(gateA, gateB);
20    //samotný přesun uzlu
21
22  }
```

obr. 7: pseudokód implementované varianty KL

4.5 Shrnutí celého běhu programu

Po zadání parametrů (názvu souboru netlistu, max. počtu vstupů a max. počtu uzlů) se obvod načte a uloží se do paměti jako graf. Potom se dekomponuje pomocí GGGP, následuje oprava dekompozice pomocí KL a nakonec se uloží statistiky a dekomponované obvody v podobě netlistu a VHDL formátu.

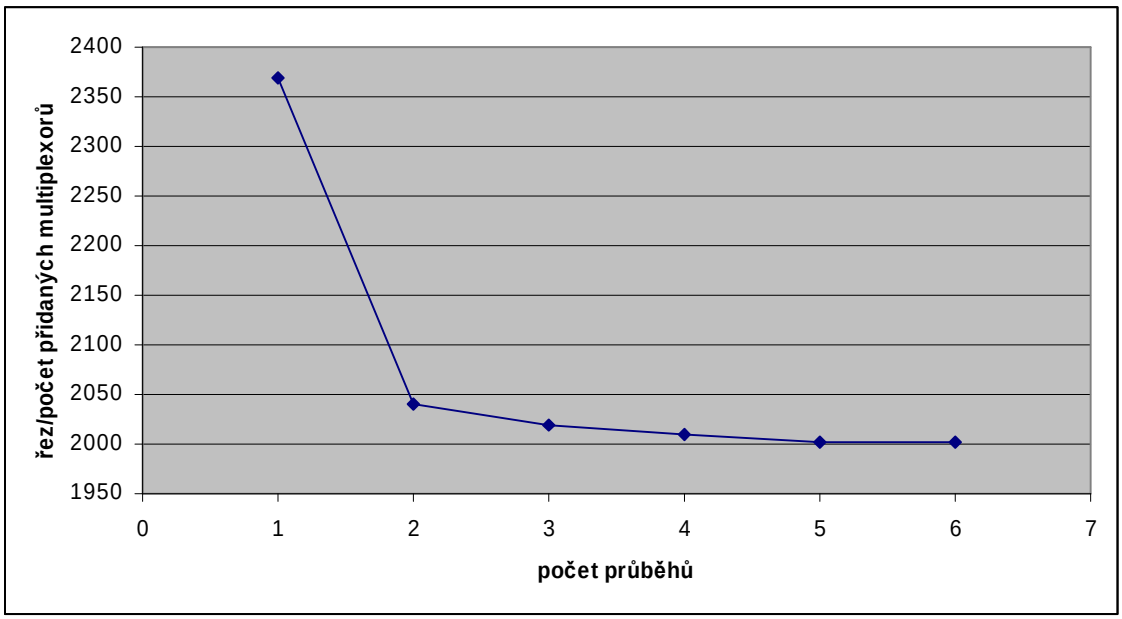
Kapitola 5

Testování

Kapitola obsahuje testování jednotlivých obvodových úloh, jejich statistiky, srovnání a závěry.

5.1 Testování konvergence KL algoritmu

Na obrázku 8 je vidět, jak probíhá postupná optimalizace dekompozice v závislosti na počtu průběhů KL algoritmu. Po prvním průběhu je znát velké vylepšení řezu, které se po dalších průbězích už vylepšuje jen velmi málo a zastaví se většinou ve 4-5 průběhu. Charakteristikou je tedy rychlá konvergence k lokálnímu minimu. Všechny průběhy obsahuje tabulka 1.



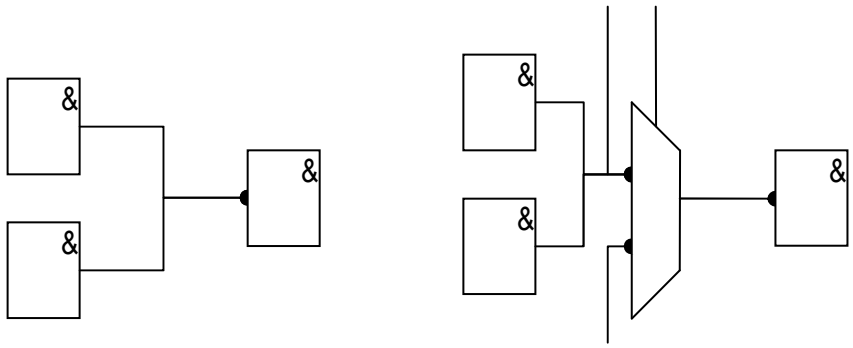
obr. 8: rychlost konvergence KL

informace o obvodu					rychlost konvergence KL vůči počtu průběhů [cena řezu]						
název	vstupů	výstupů	uzlů	podskupin	1	2	3	4	5	6	7
b14s	96	118	4625	86	2370	2040	2020	2010	2001	2001	/
b15s	132	166	8691	179	4971	4509	4468	4458	4458	/	/
b17s	152	212	23945	471	13222	12027	11741	11706	11700	11695	11695
b20s	96	86	9301	162	4285	3776	3711	3692	3690	3690	/
b21s	522	512	9259	177	4406	3917	3877	3873	3873	/	/
b22s	373	363	14676	259	7112	6245	6159	6121	6116	6116	/
c5315	178	123	2307	20	456	411	399	399	/	/	/
c7552	207	108	3512	25	653	542	542	/	/	/	/
s3271	102	90	1612	12	255	237	237	/	/	/	/
s3384	136	119	1775	9	160	139	138	138	/	/	/
s4863	153	78	2342	12	266	206	204	204	/	/	/
s9234.1	247	250	5597	25	723	682	675	675	/	/	/
s13207	90	180	8561	18	537	500	499	499	/	/	/
s15850	79	152	10304	35	1173	1124	1122	1122	/	/	/
s35932	195	480	17633	126	2956	2836	2825	/	/	/	/
s38417	150	228	23693	114	3202	3026	3012	3010	3008	3008	/

tab. 1: rychlost konvergence KL

5.2 Testování dekompozice

Vezmeme-li v úvahu, že v místě přechodu z jednoho podobvodu do druhého bude umístěn multiplexer (obr. 9), tak tab. 2, 3 a 4 ukazuje nárůst plochy z hlediska počtu hradel.



obr. 9: znázornění možného přechodu podobvodů (přibyl multiplexer, drát s jeho přepínáním a dráty pro kontrolování výstupu a pro generovaný vstup)

původní obvod			dekomponovaný obvod									
název	vstupů	hradel	cena řezu / počet podobvodů								min. řez	nárůst hradel [%]
			1	2	3	4	5	6	7	8		
b14s	96	4625	2069	2041	2107	1978	2044	2159	2013	2048	1978	42,8
			145	143	144	143	141	436	144	143		
b15s	132	8691	4700	4717	4713	4730	4778	4772	4827	4704	4700	54,1
			311	309	312	310	314	313	314	312		
b17s	152	23945	12738	12493	12491	12800	12608	12594	12669	12477	12477	52,1
			851	849	847	855	844	846	848	842		
b20s	96	9301	4014	3832	4027	3999	3901	3970	3922	4090	3832	41,2
			284	281	279	281	277	281	280	291		
b21s	522	9259	4024	4052	4016	4039	4032	3983	4079	3989	3983	43,0
			298	301	299	294	299	297	299	299		
b22s	373	14676	6542	6479	6388	6337	6312	6418	6384	6362	6312	43,0
			454	453	448	448	449	451	453	452		
c5315	178	2307	619	610	563	567	575	602	533	588	533	23,1
			47	47	45	45	45	47	45	46		
c7552	207	3512	692	649	681	684	681	707	692	694	649	18,5
			53	50	51	53	51	52	53	53		
s3271	102	1612	356	347	335	340	351	322	285	324	285	17,7
			27	26	24	26	27	25	22	24		
s3384	136	1775	170	234	229	222	191	231	177	212	170	9,6
			18	21	21	21	19	22	19	20		
s4863	153	2342	419	424	488	438	404	457	283	504	283	12,1
			32	34	34	34	32	35	22	35		
s9234.1	247	5597	872	780	926	853	848	882	871	880	780	13,9
			47	48	50	50	46	51	50	50		
s13207	90	8561	943	870	997	952	651	935	946	890	651	7,6
			63	56	63	60	37	63	57	60		
s15850	79	10304	1197	1181	1194	1239	1099	1191	1270	1244	1099	10,7
			66	67	63	66	60	66	65	65		
s35932	195	17633	3549	3386	3447	3504	3399	3606	3580	3531	3386	19,2
			255	244	246	250	240	264	268	262		
s38417	150	23693	3606	3191	3521	3091	3377	3252	3191	3264	3091	13,0
			211	206	207	202	204	212	200	208		
průměrný nárůst hradel:												26,3

tab. 2: testování dekompozice pro max. 30 vstupů pro dekomponované obvody

původní obvod			dekomponovaný obvod									
název	vstupů	hradel	cena řezu / počet podobvodů								min. řez	nárůst hradel [%]
			1	2	3	4	5	6	7	8		
b14s	96	4625	1991	1918	1892	1977	1995	1915	1980	1866	1866	40,3
			81	82	82	80	83	83	83	83		
b15s	132	8691	4416	4427	4414	4358	4480	4384	4352	4489	4352	50,1
			176	179	172	173	172	177	173	179		
b17s	152	23945	11722	11788	11787	11804	11728	11646	11759	11684	11646	48,6
			478	471	474	474	472	470	472	471		
b20s	96	9301	3837	3758	3785	3926	3584	3733	3772	3669	3584	38,5
			164	166	165	165	164	167	163	164		
b21s	522	9259	3815	3788	3884	3904	3851	3814	3856	3900	3788	40,9
			173	172	172	172	175	172	174	174		
b22s	373	14676	5957	6132	6017	6266	6184	6382	6114	6201	5957	40,6
			259	260	258	265	260	261	259	261		
c5315	178	2307	307	428	329	311	382	409	379	384	307	13,3
			15	21	16	16	19	20	20	20		
c7552	207	3512	516	514	467	501	516	459	496	554	459	13,1
			24	25	23	24	24	23	24	26		
s3271	102	1612	116	200	207	237	290	221	251	243	116	7,2
			8	11	11	12	15	11	13	12		
s3384	136	1775	152	150	174	167	170	167	176	150	150	8,5
			10	9	11	11	11	10	11	9		
s4863	153	2342	331	317	378	357	398	91	91	91	91	3,9
			16	16	18	16	18	6	6	6		
s9234.1	247	5597	684	775	791	746	715	688	780	678	678	12,1
			25	27	29	26	26	26	27	25		
s13207	90	8561	495	533	502	752	449	539	476	533	449	5,2
			19	19	18	31	16	19	17	19		
s15850	79	10304	1120	1103	981	951	1107	1179	1070	1111	951	9,2
			37	36	34	31	35	38	33	37		
s35932	195	17633	2760	2811	2926	2854	2878	2849	2805	2861	2760	15,7
			123	126	137	130	128	128	124	130		
s38417	150	23693	2922	2984	2905	2890	3253	2891	2910	3121	2890	12,2
			116	115	112	115	114	113	112	114		
průměrný nárůst hradel:												22,5

tab. 3: testování dekompozice pro max. 50 vstupů pro dekomponované obvody

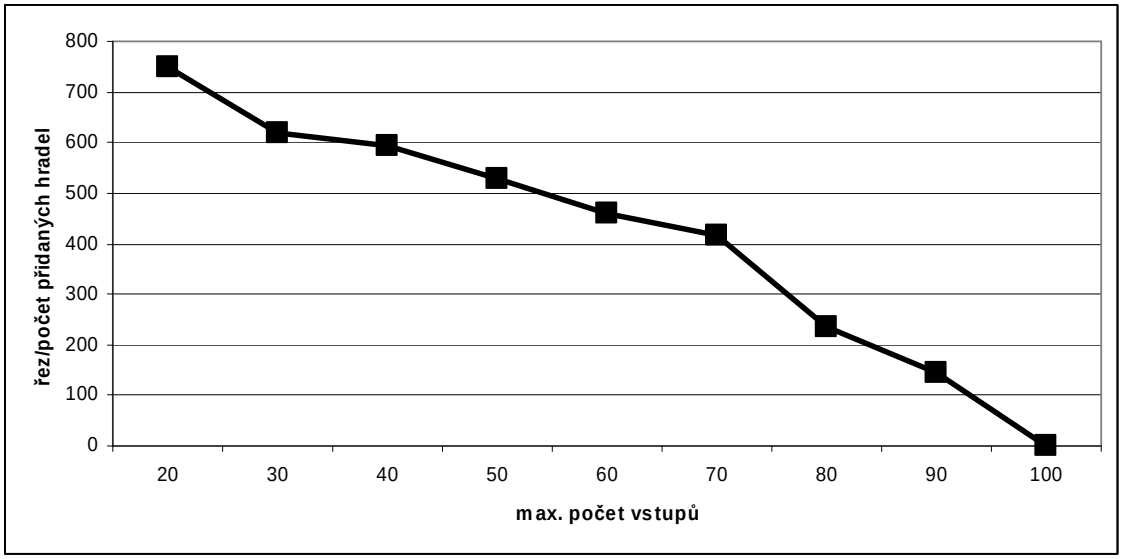
původní obvod			dekomponovaný obvod									
název	vstupů	hradel	cena řezu / počet podobvodů								min. řez	nárůst hradel [%]
			1	2	3	4	5	6	7	8		
b14s	96	4625	1873	1849	1785	1936	1896	1811	1765	1817	1765	38,2
			58	57	58	59	58	58	58	60		
b15s	132	8691	3915	4153	4053	4175	4188	4395	4046	4303	3915	45,0
			117	119	116	118	120	120	116	120		
b17s	152	23945	10991	10976	10856	11098	11449	11248	11258	11162	10856	45,3
			323	320	322	322	322	324	323	318		
b20s	96	9301	3608	3653	3699	3667	3560	3810	3708	3421	3421	36,8
			112	114	114	112	115	115	110	111		
b21s	522	9259	3711	3591	3770	3638	3702	3721	3900	3888	3591	38,8
			121	117	122	119	122	121	119	120		
b22s	373	14676	5768	5800	5775	5922	5979	5911	5804	5942	5768	39,3
			181	183	186	181	185	184	182	182		
c5315	178	2307	266	238	255	282	252	348	144	277	144	6,2
			11	11	10	11	10	18	8	11		
c7552	207	3512	391	459	454	416	454	430	391	386	386	11,0
			15	17	16	16	16	15	15	14		
s3271	102	1612	114	79	114	105	75	107	93	47	47	2,9
			5	4	5	5	4	5	5	4		
s3384	136	1775	125	150	71	118	138	126	124	66	66	3,7
			6	7	5	6	7	6	6	4		
s4863	153	2342	57	23	165	51	23	103	297	51	23	1,0
			4	3	8	4	3	5	11	4		
s9234.1	247	5597	590	643	639	603	682	619	623	713	590	10,5
			17	16	17	16	18	17	17	19		
s13207	90	8561	700	455	511	675	445	501	426	420	420	4,9
			21	12	13	19	11	14	11	11		
s15850	79	10304	992	1034	912	1067	928	941	1077	1034	912	8,9
			24	24	23	26	21	22	26	24		
s35932	195	17633	2622	2422	2196	2623	2288	2656	2108	2203	2108	12,0
			87	80	74	87	76	89	72	73		
s38417	150	23693	3048	2843	2756	2619	2612	2872	3032	2764	2612	11,0
			74	73	69	75	76	75	75	77		
průměrný nárůst hradel:												19,7

tab. 4: testování dekompozice pro max. 70 vstupů pro dekomponované obvody

původní obvod				dekomponovaný obvod					
název obvodu	vstupů	výstupů	hradel	nárůst počtu hradel	počet hradel v prvním podobvodu	počet hradel v druhém podobvodu	počet vstupů prvního podobvodu	počet vstupů druhého podobvodu	nárůst hradel [%]
b14s	96	118	4625	288	2448	2177	153	231	6,2
b15s	132	166	8691	438	4446	4245	62	508	5,0
b17s	152	212	23945	603	12591	11354	177	587	2,5
s9234.1	247	250	5597	212	2812	2745	123	334	3,8
s13207	90	180	8561	280	4363	4198	73	297	3,3

tab. 5: testování složitosti obvodu bisekcí (z 10 pokusů o 20 KL průbězích byla vybrána nejlepší bisekce)

Z tabulek 2-4 je vidět, že dekompozice je silně závislá na vlastnostech obvodu. Některé obvody dopadly velmi špatně, např. obvod *b15s* má při dekompozici o 50 vstupech 50% nárůst hradel. Oproti tomu *s4863* má pouze 3,9% nárůst. Ne každý obvod je tedy z plošného hlediska vhodný na dekompozici. Na obr. 10 je znázorněná křivka závislosti ceny řezu na max. počtu vstupů pro obvod *s13207*. V této oblasti je téměř lineární.



obr. 10: závislost ceny řezu na max. počtu vstupů obvodu *s13207*

Tabulka 5 ukazuje chování grafu při rozdělení na 2 části. Už při této operaci jde o znatelný nárůst plochy dekomponovaného obvodu a to se při této fázi počet vstupů zvyšuje. Je tak jasné, že kdyby byla například zvolená jiná varianta algoritmu, např. rekurzivní bisekce, nemůže se dosáhnout uspokojivých výsledků.

Z hlediska testovatelnosti je na tom každý obvod lépe. A to i zmiňovaný b15s, který má 132 vstupů, což by teoreticky znamenalo otestovat $5{,}5.10^{39}$ kombinací. Oproti tomu jeho dekompozice pro max. 50 vstupů a 172 podobvodech stačí otestovat $172.1{,}1.10^{15}$ kombinací, tj. $2{,}9.10^{22}$ méně kombinací. V dnešní době se však používají různé generátory a důmyslnější algoritmy testování, které netestují všechny kombinace, ale snaží se testovat pouze ty nejrelevantnější, i tak jsou silně závislé na počtu vstupů obvodu.

Dalším aspektem je zpoždění, způsobené vložením multiplexorů do obvodu. Je jasné, že to sníží max. frekvenci obvodu, ale pro přesné vyčíslení by bylo nutné implementovat algoritmus pro hledání nejdelší cesty, což je stejně jako dekompozice NP-úplný problém.

Také je potvrzen fakt, že KL je závislý na počáteční dekompozici, většinou je odchylka od nejlepší vybrané varianty do 10%. Pro nalezení dobré nejlepší varianty je potřeba provést pár pokusů, proto je i možností zvolení pokusů aplikace vybavena.

5.3 Závěr testování

Dekompozice výrazným způsobem snižuje počet nutných testovacích kombinací, na druhou stranu se ale zvýší zpoždění obvodu a také plocha, u některých obvodů dramaticky. Záleží na složitosti obvodu,

Kapitola 6

Závěr

Po prostudování dekompozičních algoritmů jsem navrhl a implementoval aplikaci, která dekomponuje obvod s ohledem na testovatelnost, tj. se snaží o rozdělení výchozího obvodu na méně částí se zadaným max. počtem vstupů. Aplikaci jsem otestoval na zkušebních úlohách a porovnal původní a dekomponované obvody. Tímto jsem tedy splnil zadání.

Práce jednoznačně prokazuje, že dekompozice znamená v obecném obvodu velký nárůst plochy obvodu a zároveň jeho zpomalení. Z tohoto hlediska se tedy nevyplatí. Na druhé straně ale zásadně vylepšuje rychlost testování. Je tedy otázkou, které kritérium upřednostníme. Osobně se přikláním k názoru, že dekomponovat již optimalizované a hotové obvody je obecně nevhodné, s dekompozicí by se mělo počítat již při návrhu obvodu.

Literatura

[1] Kolář J.: Teoretická informatika. ČIS, ISBN 80-900853-8-5, Praha 2004.

[2] B. W. Kernighan, S. Lin: An Efficient Heuristic Procedure for Partitioning Graphs, The Bell system technical journal, Vol. 49, No. 1., pp. 291-307, 1970.

[3] C. M. Fiduccia, R. M. Mattheyses: A Linear-time Heuristic for Improving Network Partitions, Design Automation, 1982. 19th Conference on, ISBN: 0-89791-020-6, 1982.

[4] George Karypis, Vipin Kumar: A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM Journal on Scientific Computing, Vol. 20, No. 1., pp. 359-392, 1998.

[5] George Karypis, Rajat Aggarwal, Vipin Kumar, Shashi Shekhar: Multilevel Hypergraph Partitioning: Application in VLSI Domain, dac, pp. 526-529, University of Minnesota, Computer Science Department, Minneapolis, 1997.

[6] David A. Papa and Igor L. Markov, Hypergraph Partitioning and Clustering, University of Michigan, EECS Department, Ann Arbor, MI 48109-2121, 2007.

[7] Charles J. Alpert and Andrew B. Kahng: Recent Directions in Netlist Partitioning: A Survey, Integration, the VLSI Journal, Volume 19, No. 1., 1995.

Dodatek A

Seznam použitých zkratek a symbolů

BIST built-in self-test, vestavěná diagnostika
E edge, hrana
FM Fiduccia-Mattheyses algorithm
G graf
GA genetic algorithms
GGGP greedy graph growing partitioning algorithm
GGP graph growing partitioning algorithm
i input vstup
k počet cest
KL Kernighan-Lin algorithm
n počet uzlů
NP nondeterministic polynomial, nedeterministicky polynomiální
o output, výstup
p partition, podmnožina
SA simmulated annealing, simulované žíhání
SB spectral bisection
TS tabu search
V vertices, uzel
VHDL - VHSIC (Very-High-Speed Integrated Circuit) Hardware Description Language, jazyk pro popis hardwaru rychlých integrovaných obvodů
VLSI very large scale integration

Dodatek B

Uživatelská příručka

Tato aplikace je konzolová, běží tedy na příkazovém řádku. Při spuštění bez parametrů se ukáže nápověda:

```
	Napoveda - parametry: <jmeno souboru> <pocet vstupu> <pocet hradel> <pocet pokusu>

	<jmeno souboru> nazev souboru ve formatu netlist*
	<pocet vstupu> maximalni pocet vstupu pro podobvody*
	*povinne parametry
	<pocet hradel> maximalni pocet, ktery muze byt v podobvodu (0=vypnuto)
	<pocet pokusu> pocet vykonanych pokusu o dekompozici (1)
	(v zavorkach jsou uvedeny defaultni hodnoty pri nezadani)
```

První 2 parametry, jméno souboru ve formátu ISCAS89 a maximální počet vstupů pro podobvody, jsou povinné. Dalším parametrem může být max. počet hradel v podobvodu. Implicitně je nastaveno na 0, tj. nezáleží na něm. Posledním parametrem je počet vykonávaných pokusů, protože GGGP algoritmus začíná náhodně, může to ovlivnit celou dekompozici. Při správném zadání parametrů se na obrazovce postupně vypíše něco podobného:

```
	C:\Dev-Cpp\Projects>dekompozice c7552.bench 50 0 5
	-DEKOMPOZICE LOGICKYCH OBVODU S OHLEDEM NA DIAGNOSTIKU-

	multiplexoru pro test c. 0: 446
	multiplexoru pro test c. 1: 513
	multiplexoru pro test c. 2: 618
	multiplexoru pro test c. 3: 554
	multiplexoru pro test c. 4: 488

	Pocet vstupu: 207
	Pocet vystupu: 108
	Pocet hradel: 3512
```

```
CELKEM PODMNOZIN: 22

CELKEM PRIDANO MULTIPLEXORU: 446
Hotovo za 1.0 sekund.

Pokračujte stisknutím libovolné klávesy...
```

Celá operace může trvat i déle, záleží na velikosti a složitosti grafu. Po názvu programu se postupně vypíšou testy s informací, jak moc mají velký hranový řez. Následně je vybrán ten s nejmenším, jsou o něm vypsány informace a následně je uložen do nového adresáře s časovým razítkem a názvem obvodu ve svém názvu, např.:

```
2009-07-01-12-51-53-c7552
```

Adresář obsahuje soubor *a_bist_info.txt* s informacemi o obvodu, jako je název, počet vstupů, výstupů, hradel, podmnožin, cena řezu a nakonec informace o dekomponovaných obvodech ve formátu:

```
číslo_obvodu[g:počet_hradel/i:počet_vstupů/o:počet_výstupů]
```

Dále obsahuje dekomponované obvody ve formátu ISCAS89 (pro každý podobvod 1 soubor) a VHDL (2 soubory).

Dodatek C

Obsah přiloženého CD

- **/aplikace** obsahuje zkompilovanou aplikaci pro systém MS Windows (32bit)
- **/prostredi** obsahuje instalační soubor vývojového prostředí Bloodshed Dev-C++ 4.9.9.2. Nejnovější verzi lze stáhnout z http://www.bloodshed.net/dev/devcpp.html.
- **/text** obsahuje text vlastní bakalářské práce.
- **/zdrojove kody** obsahuje kódy aplikace v jazyce C++
- **/zkusebni ulohy** obsahuje všechny zkušební úlohy použité při testování