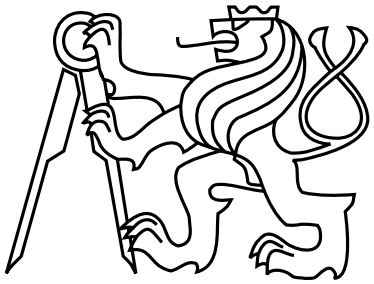
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Diplomová práce

Randomizace algoritmů v systému ABC

Bc. Tomáš Hruška

Vedoucí práce: Ing. Petr Fišer, Ph.D.

Studijní program: Elektrotechnika a informatika, strukturovaný, Navazující magisterský

Obor: Výpočetní technika

29. prosince 2011

Poděkování

Především bych chtěl poděkovat svému vedoucímu práce Ing. Petru Fišerovi za jeho pomoc při tvorbě mé diplomové práce.

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 29. 12. 2011 .....

Abstract

The work describes the general features of ABC system with great emphasis on the description of several commands that are used for combinational synthesis. Theory of operation and programming solutions of selected commands in the ABC system is discussed.

Several possible methods of randomization are suggested. The proposed methods are either implemented as an extension of the existing commands with new switches, or just use the existing commands properly for randomization.

These methods are also tested on standard benchmark circuits and test results are presented in the form of tables and graphs.

Abstrakt

V práci jsou popsány obecné vlastnosti systému ABC s velkým důrazem na popis několika příkazů, které se používají pro kombinační syntézu. U vybraných příkazů je probrána teorie jejich fungování a také programové řešení těchto příkazů v systému ABC.

Dále je navrženo několik možných způsobů randomizace těchto příkazů v systému ABC. Navržené způsoby jsou buď implementovány jako rozšíření stávajících funkcí systému ABC pomocí nově přidaných přepínačů ke stávajícím příkazům nebo pouze využívají stávajících příkazů způsobem vhodným pro randomizaci.

Tyto způsoby jsou v rámci práce také testovány a výsledky testování jsou předloženy ve formě tabulek a grafů.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Systém ABC je velmi komplexním a rychle se rozvíjejícím nástrojem, který má své uplatnění hlavně pro syntézu a verifikaci binárních sekvenčních logických obvodů. Jedná se o nástroj, který je ve své podstatě přísně deterministický, což znamená, že při každém provádění té samé sekvence příkazů dává pokaždé stejné výsledky.

V následujícím textu práce jsou rozebrány způsoby, jakými může být zahrnuta určitá náhodnost do tohoto deterministického chování. Výsledkem těchto způsobů jsou úpravy, které vedou k tomu, že se příkazy nechovají přísně deterministicky, ale pro opakovaná spouštění jsou schopny podávat různé výsledky, a to takovým způsobem, že je zachována korektnost výstupu. Z toho plyne, že při zachování stejné výsledné funkce se můžeme dostat k lepším, ale také horším výsledkům.

Smyslem nebylo navrhnout metody, které by vedly pouze k jednoznačně lepším výsledkům, ale k jakýmkoliv, které nejsou přísně deterministické. To znamená, že předkládané metody nemusí být vždy lepším řešením než stávající deterministická verze, ale mohou lepších výsledků dosáhnout. Samozřejmě můžeme říci, že snaha o dosažení lepších výsledků je očekávána a jde tedy o hledání globálního minima.

Všechny tyto kroky vedou k zavedení náhodnosti, což reprezentuje **randomizaci** algoritmů.

Postup samotné randomizace je možno rozdělit na dva způsoby, které můžeme pojmenovat takto:

- vnější randomizace
- vnitřní randomizace

Vnější randomizace je postup, kdy nedochází k zásahům do algoritmů jednotlivých příkazů systému ABC. Randomizace je tedy zajištěna zvenčí nějakým jiným postupem.

Vnitřní randomizace je postup, kdy dochází k zásahům do algoritmů jednotlivých příkazů systému ABC. Prvkem, který zajišťuje randomizaci, je zásah přímo v zdrojovém kódu daného příkazu. Tento zásah tedy mění logiku fungování příkazů.

Navržené randomizační postupy jsou dále experimentálně ověřeny a pomocí měření jsou získány výsledky, které odrážejí efektivnost těchto metod.

Jako výsledek vznikne náhled na to, jak tyto metody mohou buď pozitivně nebo i negativně ovlivnit výsledky, které bychom dostali pomocí stávající implementace příkazů v systému ABC.

Další rozvržení této práce vypadá následovně:

- V první části práce jsou shrnuty obecné specifikace systému ABC a jeho základní rysy. Jsou zde popsány některé možnosti a funkce, které systém ABC obsahuje.
- Druhá část práce se zabývá popisem příkazů systému ABC. V rámci této části jsou rozebírány algoritmy, které příkazy využívají a podrobně je rozepisován princip jejich fungování.
- Třetí část popisuje možné použití příkazů systému ABC pro randomizaci a podrobněji rozebírá, jak jsou tyto metody implementovány.
- Ve čtvrté části jsou uvedeny výsledky experimentů, ve kterých jsou tyto randomizační postupy využívány.

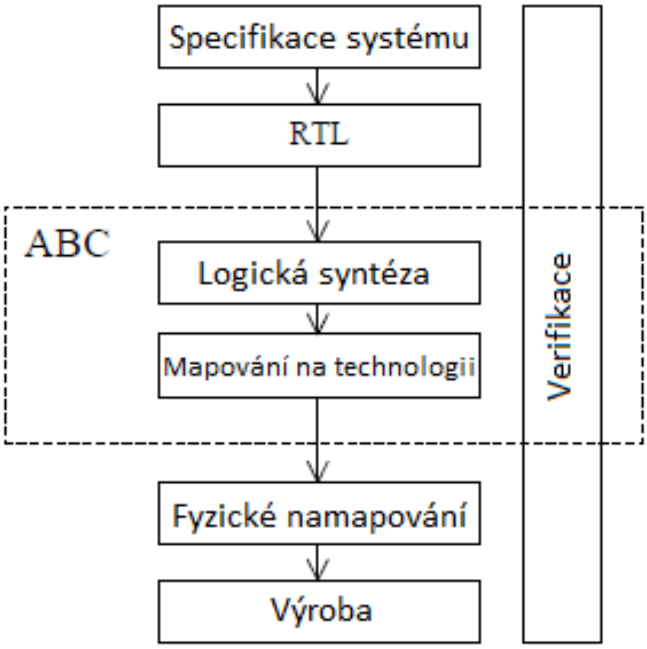
Kapitola 2

Systém ABC

Systém ABC je robustní nástroj pro syntézu a verifikaci binárních sekvenčních logických obvodů, který nachází uplatnění v synchronních hardwarových návrzích. Systém je vyvíjen na kalifornské univerzitě Berkeley pod záštitou Berkeley Verification and Synthesis Research Center[2]. Hlavním autorem, který stojí za vývojem tohoto systému, je Alan Mishchenko, který se vývoji a zdokonalování systému ABC stále věnuje.

Systém ABC je vyvíjen jako volně dostupný a zdrojové kódy jsou volně ke stažení[4]. Tvorba systému jako takového započala už velmi dávno a dodnes je kontinuálně vyvíjen a vylepšován.

Pole působnosti, ve kterém se systém ABC pohybuje, lze ukázat následovně:



Obrázek 2.1: Oblast působnosti systému ABC

2.1 Základní charakteristiky ABC

Systém ABC[1] v sobě kombinuje tři prvky:

- škálovatelnou logickou optimalizaci založenou na AIG grafech
- mapování na technologii[19] s ohledem na optimální zpoždění, které je založené na DAG grafech (orientované, acyklické grafy) pro „look-up tables“ neboli LUT a standardní buňky
- algoritmy pro sekvenční syntézu a verifikaci

Systém ABC jako takový poskytuje implementaci a programové nástroje pro používání všech těchto tří prvků. Cílem tvorby systému ABC je vytvořit veřejně dostupný nástroj, který využívá sofistikovaných algoritmů pro kombinační a sekvenční syntézu, které budou schopny tuto činnost vykonávat na moderních výpočetních prostředcích ve velmi dobrém čase. Základními rysy systému ABC jsou:

- Pro reprezentaci a manipulaci s kombinačními a sekvenčními obvody používá mnoho různých druhů datových struktur. Může je reprezentovat například jako „netlist“, logický obvod s uzly reprezentovanými pomocí SOP[8] („sum of products“ neboli úplná normální disjunktivní forma) nebo BDD[9] (binární rozhodovací diagram), namapovaný obvod, AIG graf[20].
- Vstupní soubory pro ABC mohou být v mnoha formátech, jelikož součástí ABC jsou exportéry, které umí tyto soubory zpracovat a převést do vnitřního formátu ABC. Příkladem takovýchto souborů mohou být soubory typu BLIF, BENCH nebo EDIF.
- Pro výstup ABC opět implementuje několik exportérů, které převádí vnitřní implementaci obvodu na nějaký výstupní formát. Například BLIF, PLA, BENCH nebo také na dva formáty vhodné pro grafové zobrazení (DOT, GML).
- Používání formátu BAF (binární formát pro AIG grafy), který zajišťuje čtení a zapisování velkých AIG grafů pomocí binárních souborů. Díky tomuto formátu je možno číst a zapisovat AIG grafy s více jak miliony uzlů ve velmi rychlých časech při zachování malé paměťové náročnosti.
- Datové struktury, které reprezentují logické obvody jsou v základech velmi podobné jako pro systém SIS[24] a je nad nimi také možno provádět podobné operace. Například „technology-independent sweeping“[18], „disjoint-support decomposition“[15], „structural hashing and balancing“[17] AIG grafů, kontrolu ekvivalence[10], zpětné krokování sekvenčních obvodů po časových úsecích a mnoho dalších.
- Různé operace nad AIG grafy - příkazy systému ABC balance, rewrite, refactor a mnoho dalších.
- Zavádí způsoby pro detekci a průběžné ukládání různých strukturních reprezentací booleovských funkcí.

- Umožňuje uložení několika různých stavů daného obvodu v průběhu zpracovávání systémem, čímž zajišťuje bezeztrátovou syntézu. Tato syntéza obsahuje strukturálně různé, avšak funkčně ekvivalentní verze, které jsou sesbírány v průběhu zpracovávání obvodu systémem ABC.
- Nástroje pro FPGA mapování[21] s proměnnou velikostí look-up tables neboli LUT[21] a mapování standardních buněk.
- Pro tvorbu BDD využívá CUDD balík[3] od Fabia Somenziho.
- Ověřování ekvivalence kombinačních obvodů pomocí SAT solveru[25]. Využívá řešení MiniSat[14] od Niklase Eéna a Niklase Sörenssona.
- Datové struktury pro sekvenční syntézu a experimentální implementaci integrace tří postupů do jednoho celku:
  1. logické syntézy
  2. mapování
  3. „re-timing“

Z výčtu těchto vlastností plyne, že pomocí systému ABC lze provádět několik základních úkonů:

- syntézu kombinačních obvodů
- syntézu sekvenčních obvodů
- FPGA mapovaní
- mapování standardních buněk
- verifikaci
- načítání a ukládání obvodů v různých formátech

Pro potřeby této práce je nejdůležitější částí syntéza kombinačních obvodů, kde se vyskytují příkazy, které jsou v rámci této práce randomizovány. Tyto příkazy pracují v systému ABC nad AIG grafy. Abychom tedy mohli popsat fungování těchto příkazů, musíme vědět jakým způsobem jsou realizovány AIG grafy.

2.2 AIG grafy

2.2.1 Vývoj AIG grafů

Samotný název AND-invertorové grafy napovídá, že AIG grafy jsou složené z dvouvstupových AND hradel a invertorů. Konverze obvodů do AIG grafů je rychlá a dobře škálovatelná. Tato konverze vlastně vyžaduje pouze to, aby každé hradlo bylo vyjádřeno pomocí AND hradel a invertoru. Postup konverze nevede k žádným markantním nárokům na paměť ani na dobu běhu. Díky tomu jsou AIG grafy dobrou reprezentací obvodů i ve srovnání například s BDD nebo SOP reprezentacemi.

Vývoj podobných reprezentací obvodů (reprezentace pomocí jednoduchých hradel) započal už velmi dávno. Výzkumy ohledně AIG grafů byly prováděny už někdy ke konci padesátých let 20. století a dále pokračovaly až do let sedmdesátých. V této době byly také vytvořeny různé způsoby pro lokální transformace (převody hradel na hradla AND a invertory). Tyto transformace jsou základem i pro systém ABC.

V dnešní době význam AIG grafů vzrostl, jelikož se začaly používat pro funkční reprezentaci při různých úlohách v rámci logické syntézy a verifikace obvodů. Dobrým příkladem tohoto trendu je sám systém ABC, kde jsou AIG grafy využívány přesně k těmto účelům. Důvodem pro tento vývoj je to, že v devadesátých letech 20. století došlo k tomu, že reprezentace které se běžně používaly (BDD) dosáhly svých limitů ve škálovatelnosti použití. Dalším dobrým důvodem pro využití AIG grafů je markantní rozvoj v oblasti SAT řešičů u kterých došlo v poslední době k obrovskému výkonnostnímu nárůstu. Díky tomu se kombinace AIG grafů a SAT řešičů stala zdrojem velkého zrychlení při řešení velké škály booleovských problémů.

Opět typickým příkladem tohoto spojení je systém ABC, který využívá AIG grafů ve spolupráci s SAT řešičem.

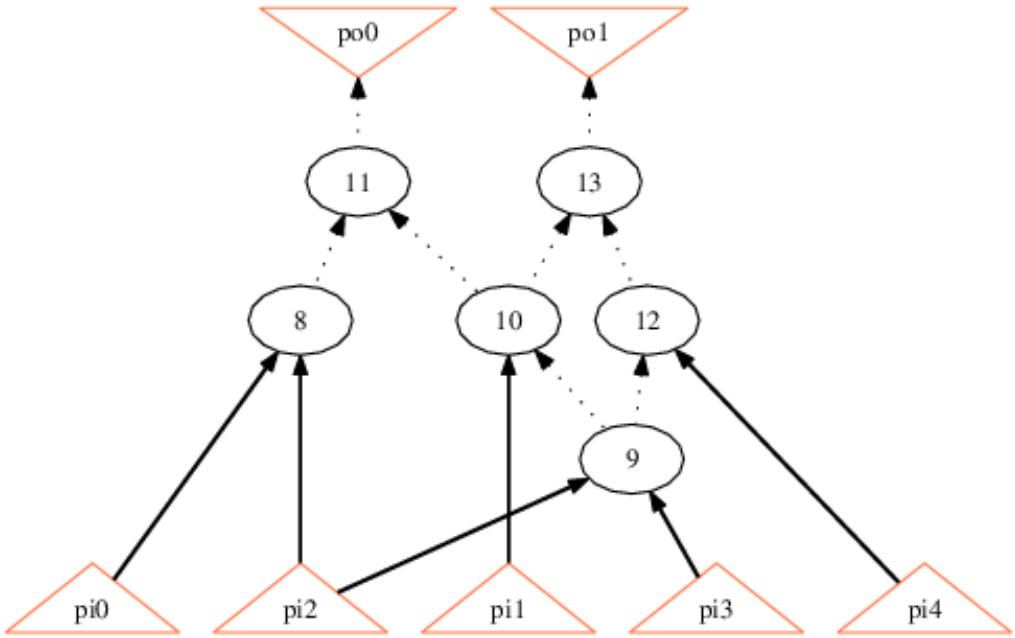
2.2.2 Teorie AIG grafů

AIG grafy jsou definovány takto:

Nechť (X, A), kde $A \subseteq X \times X$ je orientovaný, acyklický graf, kde každý uzel $n \in X$ má buď žádnou vstupní hranu a nebo právě dvě vstupní hrany. Uzly, které nemají vstupní hrany, jsou nazývány **vstupy**. Uzly, které mají dvě vstupní hrany, jsou nazývány **And**. Nechť **inv** je funkcí A nabývající hodnot z množiny $\{0, 1\}$.

Poté se n-tice $G = (X, A, inv)$ nazývá And-invertorový graf neboli AIG graf.

Grafická reprezentace AIG grafu v systému ABC vypadá například takto:



Obrázek 2.2: Ukázka obvodu c17 zobrazeného jako AIG graf

Hrana je invertovaná, pokud platí $inv(a) = 1$. Invertované hrany se zobrazují přerušovanou čárou, neinvertované hrany čarou nepřerušovanou.

Každý uzel a hrana v AIG grafu reprezentují booleovskou funkci. Tato booleovská funkce se nazývá funkcí daného uzlu nebo hrany. Každému ze vstupních uzlů n je přiřazena formální booleovská proměnná $\aleph_n$. Všechny funkce jednotlivých uzlů a hran jsou definovány za použití těchto proměnných.

Jedna booleovská funkce může být reprezentována různými AIG grafy s jedním výstupem, z čehož plyne, že AIG grafy nejsou kanonickou reprezentací booleovských funkcí. Důsledkem této vlastnosti je to, že v AIG grafech může docházet k redundancím, kdy dva různé uzly mohou počítat stejnou booleovskou funkci. Při konstrukci AIG grafu by se tedy mělo vycházet z několika podmínek, kterým se souhrnně říká „structural hashing“[17]. Splněním těchto podmínek u AIG grafu docílíme toho, že u něj takovéto redundance nevzniknou. Všechny AIG grafy se kterými, se v rámci této práce setkáváme, považujeme za takové, že splňují tyto podmínky.

Kapitola 3

Příkazy zvolené pro randomizaci a jejich implementace

V rámci této práce jsou vybrány příkazy, které jsou určené pro kombinační syntézu. Tím pádem se omezujeme pouze na velmi malou část schopností systému ABC, avšak pro rozsah této práce to postačuje.

V rámci používaných obvodů se tedy setkáme pouze s kombinačními obvody. To jsou obvody, ve kterých stavy na výstupech závisí pouze na okamžitých kombinacích vstupních proměnných a nezávisí na jejich předchozích hodnotách. Kombinační obvody nemají žádnou paměť předchozích stavů, z čehož plyne, že jedna kombinace vstupních proměnných odpovídá pouze jedné kombinaci výstupních proměnných.

Systém ABC poskytuje mnoho různých příkazů pro kombinační syntézu, pro randomizaci v rámci této práce jsou významné tyto:

- balance
- rewrite
- refactor
- resub
- rr
- collapse

Důvodem pro výběr této množiny příkazů je to, že mají předpoklad pro úspěšné použití pro randomizaci, což tedy znamená, že se v jejich algoritmech vyskytují postupy, které lze randomizovat, nebo tyto příkazy lze využít i pro vnější randomizaci.

3.1 Příkaz BALANCE

Příkaz BALANCE provádí vyvažování AIG grafu v systému ABC. Tento příkaz provede nad předem načteným AIG grafem algoritmus, který se nazývá AND-vyvažování[22].

AND-vyvažování je postup, který slouží především ke snížení počtu úrovní v AIG grafu. Cílem algoritmu je snaha o redukci výšky daného grafu změnou umístění jeho uzlů reprezentovaných AND hradly.

3.1.1 Teorie příkazu BALANCE

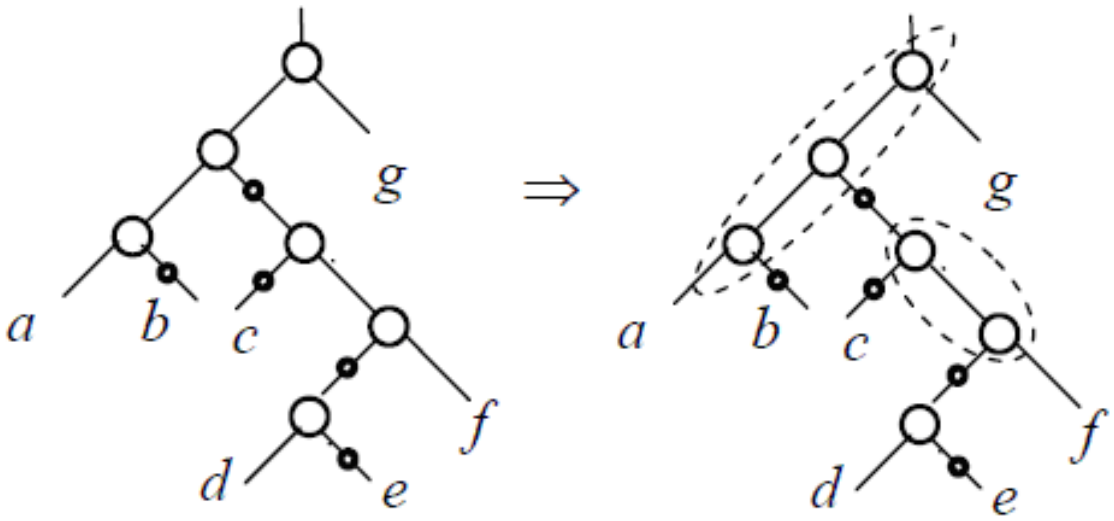
Samotný algoritmus AND-vyvažování se skládá ze dvou kroků:

1. hledání pokrytí
2. vyvažování

3.1.1.1 Fáze hledání pokrytí

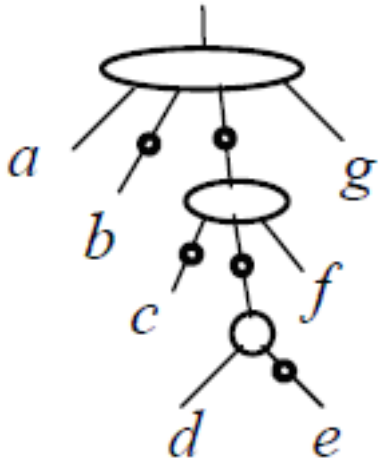
Při hledání pokrytí se AIG graf prohledává shora dolů a hledají se v něm supergate. Supergate je velké hradlo AND s mnoha vstupy, které se skládá z několika běžných AND uzlů AIG grafu. Tyto AND uzly musí splňovat dvě podmínky:

1. na hranách mezi nimi nesmí být invertor
2. z žádného z těchto uzlů nevede cesta na kombinační výstup celého obvodu



Obrázek 3.1: Hledání pokrytí v algoritmu AND-vyvažování

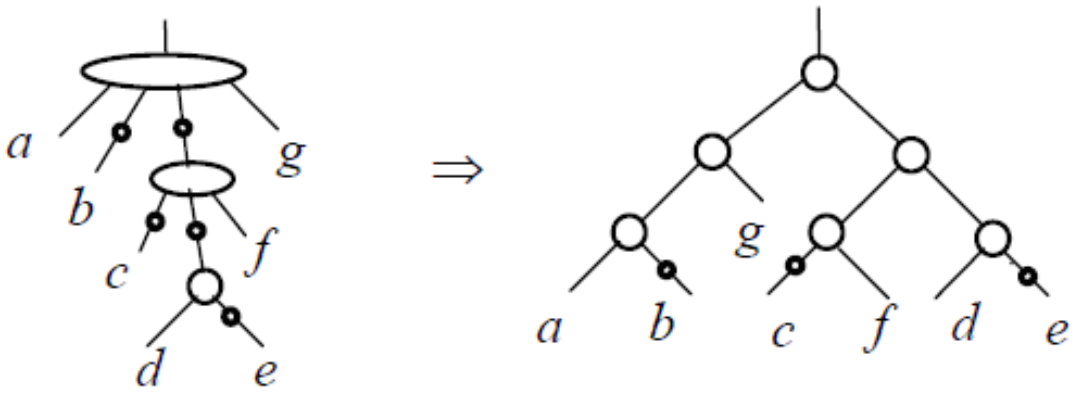
Algoritmus nalezne dvě supergate, které jsou znázorněny pomocí přerušované čáry. Výsledkem je tedy stav, který vypadá takto:



Obrázek 3.2: Výsledek hledání v algoritmu při AND-vyvažování

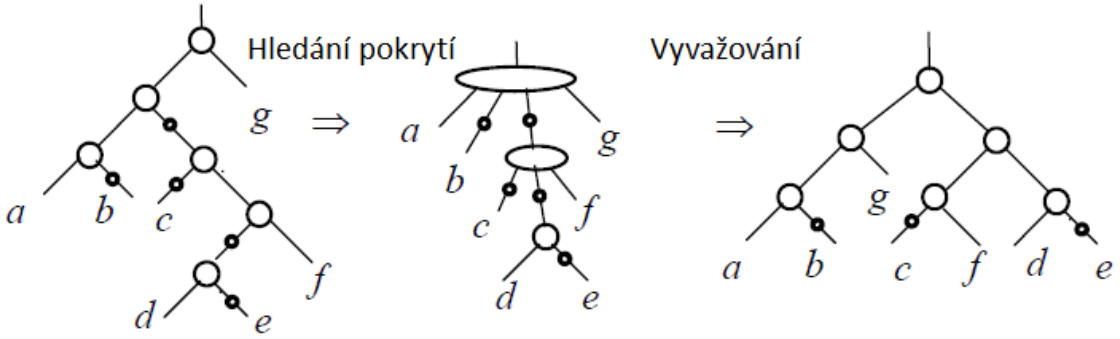
3.1.1.2 Fáze vyvažování

Po dokončení hledání pokrytí, což značí nalezení všech supergate, začíná druhá část algoritmu. Tato část provádí vyvažování všech uzlů kromě kombinačních výstupů obvodu a těch, které jsou součástí některé z nalezených supergate. Algoritmus rozloží zpět velká vícevstupová hradla AND na běžná dvouvstupová hradla AND a zbytek uzlů vyváží takovým způsobem, aby došlo k redukci výšky AIG stromu.



Obrázek 3.3: Fáze vyvažování v algoritmu AND-vyvažování

Při spojení obou částí algoritmu dostaneme výsledný postup práce příkazu BALANCE.



Obrázek 3.4: příkaz BALANCE

Obrázky uvedené v této kapitole jsou převzaty z článku[22] a upraveny.

3.1.2 Programové řešení příkazu BALANCE

Implementace ve zdrojovém souboru abcBalance.c v balíku base/abci

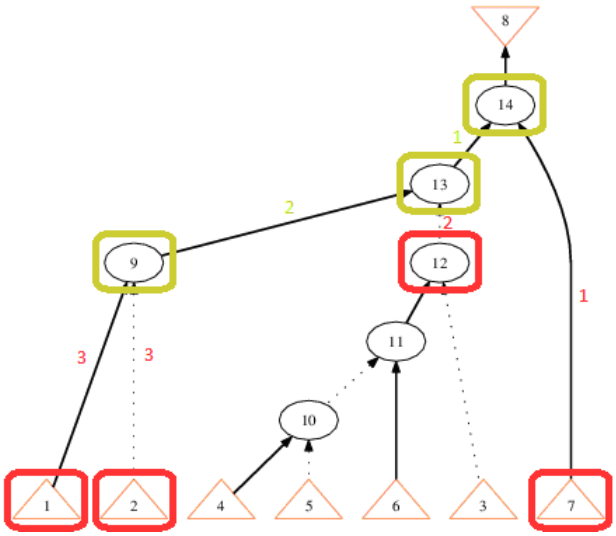
Příkaz BALANCE je v systému ABC spouštěn voláním funkce `Abc_NtkBalance`. Na začátku provádění příkazu je AIG graf reprezentován pouze primárními vstupy a primárními výstupy. Tohoto stavu je docíleno voláním funkce `Abc_NtkStartFrom`. Algoritmus AND-vyvažování je spuštěn voláním funkce `Abc_NtkBalancePerform`.

Abc_ NtkBalancePerform

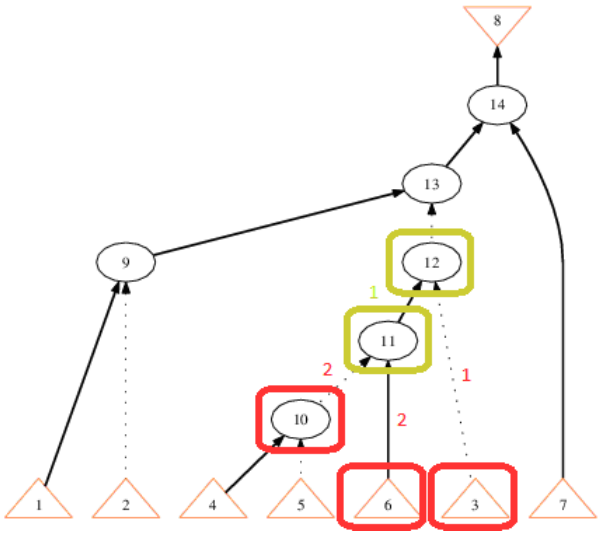
V této funkci je zahrnut hlavní cyklus `Abc_NtkForEachCo`, který zajišťuje průchod všemi kombinačními výstupy obvodu. To znamená, že algoritmus prochází AIG graf shora dolů, tedy od výstupů k vstupům. Nad každým uzlem je spuštěna funkce `Abc_NodeBalance_rec`, která zajistí obě části algoritmu AIG-vyvažování. Jedná se o rekurzivní funkci. Nejprve v ní musí proběhnout hledání pokrytí. To je zajištěno hned na počátku této funkce pomocí volání funkce `Abc_NodeBalanceCone`. Tato funkce rekurzivně volá další funkci starající se o hledání supergate `Abc_NodeBalanceCone_rec`.

Abc_NodeBalanceCone_rec

Funkce tedy rekurzivně volá samu sebe a jejím postup ilustrují následující obrázky.



Obrázek 3.5: Hledání první supergate



Obrázek 3.6: Hledání druhé supergate

Po nalezení supergate máme tedy uzly, které postupují do druhé fáze algoritmu. Následně dojde k návratu do funkce `Abc_NodeBalance_rec`, kde pokračuje druhá fáze algoritmu nad těmito uzly.

Programové řešení fáze vyvažování se dá popsat následujícím pseudokódem respektujícím skutečné názvy funkcí v systému ABC:

```
ForEachNodesSetAfterFindingSuperGates {
   Vec_PtrSort(Nodes, Abc_NodeCompareLevelsDecrease)
   Abc_NodeBalanceFindLeft(Nodes)
   Abc_VecObjPushUniqueOrderByLevel(SelectedNodes)
}
```

ForEachNodesSetAfterFindingSuperGates

Tento cyklus zajistí, že se budou procházet množiny uzlů, které vznikly po nalezení supergate. Jsou to tedy všechny uzly, které nejsou součástí supergate. Do cyklu se dostávají jako množiny uzlů za každou z nalezených supergate.

Vec_PtrSort(Nodes, Abc_NodeCompareLevelsDecrease)

Zajistí seřazení uzlů v sestupném pořadí dle jejich úrovně v AIG grafu, tento krok je nezbytný pro správnou funkci dalších částí algoritmu.

Abc_NodeBalanceFindLeft(Nodes)

Ze vstupní množiny uzlů, která vzešla z jedné nalezené supergate, se vezme nejpravější uzel, který má díky předešlému seřazení nejnižší úroveň v grafu. Následně se k němu hledá pozice nejlevějšího uzlu v množině, který má ještě stejnou úroveň jako uzel nejpravější.

Abc_VecObjPushUniqueOrderByLevel(SelectedNodes)

Do této části vstupují tedy dvě pozice. Pozice nejpravějšího uzlu s nejmenší úrovní v grafu a také pozice nejlevějšího uzlu s ještě stejnou úrovní jako zkoumaného nejpravějšího uzlu.

Poté se začne množina prohledávat od této nejlevější pozice takovým způsobem, že vždy vznikne dvojice nejpravějšího uzlu a uzlu na nejlevější pozici minus jedna za každé opakování hledání.

Pro každou takovou dvojici se provede kontrola, zda už v grafu neexistuje uzel, jenž by byl jejich rodičem. Pokud ano, musí se tato již existující trojice ponechat. Pokud při procházení množiny není tímto způsobem žádný rodič nalezen, dojde k situaci, kdy se pro dvojici nejpravějšího a aktuálního nejlevějšího uzlu takový rodič vytvoří.

Tímto způsobem se tedy AIG graf znovu sestavuje. Příkaz BALANCE na počátku vychází pouze z grafu složeného z primárních vstupů a primárních výstupů obvodu a voláním funkce `Abc_VecObjPushUniqueOrderByLevel` postupně do tohoto holého grafu přidává nové uzly. Z principu způsobu, jakým jsou uzly do grafu přidávány, dochází k redukci maximální úrovně AIG grafu.

3.2 Příkaz REWRITE

Opět se jedná o jeden ze základních příkazů pro kombinační syntézu. Z jeho názvu vyplývá, že je prováděn pomocí algoritmu, který přepisuje určité části AIG grafu na jiné.

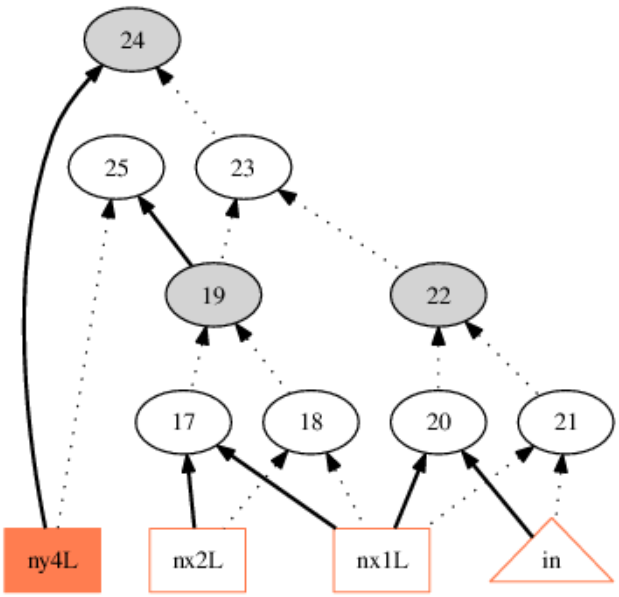
3.2.1 Teorie příkazu REWRITE

Tento algoritmus[20] slouží pro minimalizaci velikosti AIG grafu pomocí iterativního provádění, kde dochází k nahrazování AIG podgrafů pomocí předpřipravených podgrafů, při zachování funkce kořenového uzlu původního podgrafu.

Při provádění takovéhoto nahrazování je třeba nejprve určit jak velké podgrafy budou pro toto nahrazení zkoušeny. Algoritmus pracuje s pojmem „K-feasible cut“.

Cut C neboli řez C uzlu n je množina uzlů sítě, které se nazývají listy, taková, že každá cesta z primárních vstupů prochází aspoň jedním listem.

K-feasible cut je takový řez, že počet uzlů v něm nepřekročí hodnotu K. Jedná se tedy o velikost řezu, kde velikost specifikuje počet uzlů v daném řezu.



Obrázek 3.7: řez o velikosti 3

Oba tyto pojmy lze dobře vysledovat na výše uvedeném obrázku. Pokud bychom brali v potaz obyčejný řez bez specifikování jeho velikosti, dojdeme k tomu, že řezem C by byl opět kořenový uzel 24 a všechny čtyři primární vstupy.

Při tvorbě řezu o velikosti 3 bude výsledkem situace uvedená na obrázku. Hledáme tedy tři uzly takové, že jimi projdou všechny cesty z primárních vstupů do kořenového uzlu. Při podrobnějším pohledu na uzly 19 a 22, které jsou součástí řezu, zjistíme, že jsou to jediné

dva uzly v grafu, přes které vedou všechny cesty z primárních vstupů do kořenového uzlu za předpokladu, že uzel ny4L už byl vybrán.

Při hledání například řezu o velikosti 4 by uzel ny4L byl vždy součástí řezu, avšak pro zbylé tři hledané uzly by mohly nastat dvě situace:

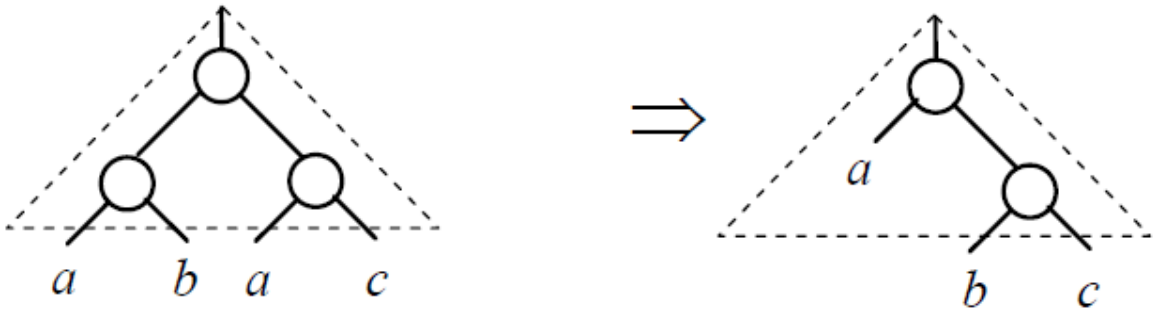
1. uzly 22, 17, 18
2. uzly 19, 20, 21

Vrátíme-li se k myšlence algoritmu, zjistíme, že algoritmus využívá všechny možné řezy o velikost 4. Nejprve se určí tyto řezy pro všechny uzly v grafu. Poté se uzly procházejí v topologickém uspořádání a pro každý uzel se porovnávají všechny jeho řezy s předpřipravenými podgrafy.

Pro dosažení efektivního porovnání těchto řezů a předpřipravených uzlů se musí pro všechny nalezené řezy o velikosti 4 spočítat jejich booleovská funkce a určit jejich NPN třídy ekvivalence.

Definice *NPN třída ekvivalence* říká, že dvě booleovské funkce F a G jsou NPN ekvivalentní, pokud funkce F může být odvozena z funkce G pomocí negací a permutací svých vstupů a negací výstupu.

Poté se prohledají předpřipravené podgrafy a zjistí se, které z nich mají stejnou NPN třídu ekvivalence. Tím dostaneme výslednou množinu NPN ekvivalentních podgrafů. Množina se prochází a každým ze podgrafů se zkusí nahradit řez o velikosti 4, pro který se tento postup aktuálně provádí. Po vyzkoušení všech podgrafů se vybere ten, který poskytuje největší užitek, což je počet uzlů AIG grafu, které tímto nahrazením můžeme vypustit. Toto vypuštění ilustruje následující obrázek, který je převzat z článku[20].



Obrázek 3.8: Vypuštění uzlu z AIG grafu pomocí nahrazení příkazem REWRITE

3.2.2 Programové řešení příkazu REWRITE

Implementace ve zdrojovém souboru abcRewrite.c v balíku base/abci

Postup algoritmu se dá vyjádřit tímto pseudokódem:

```
Rewriting( network AIG, hashable PrecomputedStructures, bool UseZeroCost )
{
   for each node N in the AIG in the topological order {
    for each 4-input cut C of node N computed using cut enumeration  {
       F = Boolean function of N in terms of the leaves of C
         PossibleStructures = HashTableLookup( PrecomputedStructures s,F );
         //find the best logic structure for rewriting
         BestS = NULL; BestGain = -1;
         for each structure S in PossibleStructures  {
            NodesSaved = DereferenceNode( AIG, N );
            NodesAdded = ReferenceNode( AIG, S );
            Gain = NodesSaved - NodesAdded;
            Dereference( AIG, S );
            Reference( AIG, N );
            if ( Gain > 0  || (Gain = 0 && UseZeroCost )
               if ( BestS = NULL || BestGain < Gain )
                  BestS = S;  BestGain = Gain;
         }
         // use the best logic structure to update the netist
         if ( BestS  !=  NULL ) {
            NodesSaved = DereferenceNode( AIG, N );
            NodesAdded = ReferenceNode( AIG, S );
            assert( BestGain = NodesSaved - NodesAdded );
         }
      }
   }
}
```

Z implementace tohoto pseudokódu přímo v systému ABC zjistíme, že příkaz REWRITE je realizován voláním funkce `Abc_NtkRewrite`. Průběh příkazu REWRITE je řízen jedním velkým cyklem `Abc_NtkForEachNode` ve kterém jsou postupně procházeny všechny uzly v topologickém pořadí a pro každý uzel je zavolána funkce `Rwr_NodeRewrite`.

Rwr_ NodeRewrite

V těle této funkce je nejprve proveden výpočet všech řezů pro daný uzel pomocí volání `Abc_NodeGetCutsRecursive`. Dalším krokem je procházení všech těchto řezů v cyklu. Na začátku každého průchodu pro daný nalezený řez je zjištěno, zda splňuje podmínku velikosti 4. Při jejím nesplnění dojde k tomu, že se dál v cyklu nepokračuje a přejde se na další řez v pořadí.

Při splnění podmínky se pokračuje voláním funkce `Rwr_CutEvaluate`. V této funkci už dochází k hledání vhodných podgrafů stejné NPN ekvivalenční třídy pro nahrazení a určování toho, zda je to výhodné nahrazení, či nikoliv.

Rwr_ CutEvaluate

Při volání funkce `Rwr_CutEvaluate` je jejím prvním krokem to, že musí být vybrány podgrafy, které mají stejnou NPN třídu ekvivalence jako právě testovaný řez. Výběr těchto podgrafů je proveden následovně:

```
vSubgraphs = Vec_VecEntry( p->vClasses, p->pMap[uTruth] );
```

Následně jsou tyto podgrafy v cyklu prohledávány a testovány a je zjišťováno, k jak velkému zlepšení by při jejich použitím došlo.

3.3 Příkaz REFACTOR

Příkaz REFACTOR je v principu velmi podobný příkazu REWRITE, avšak pro svou práci využívá faktorizované formy booleovských funkcí.

3.3.1 Teorie příkazu REFACTOR

Hlavní úlohou v tomto algoritmu je určit jeden specifický řez pro každý uzel AIG grafu a poté se pokusit nahradit tuto nalezenou část AIG grafu pomocí faktorizované formy boolovské funkce této části.

Algoritmus prochází jednotlivé uzly AIG grafu v topologickém pořadí a pro každý uzel hledá tento specifický řez, který se nazývá reconvergence-driven cut.

3.3.1.1 Reconvergence-driven cut

Hledání tohoto řezu[17] vychází z principu hledání řezů, který se nazývá CollectNodesTFI neboli hledání tranzitivního fan-in uzlu (metoda CollectNodesTFO provádí hledání tranzitivního fan-out uzlu). Metoda CollectNodesTFI postupuje takto:

1. Určí se uzel N, pro který se bude CollectNodesTFI provádět.
2. Číslo m, které určuje v jaké vzdálenosti se uzly budou od uzlu N nacházet, bývá obvykle menší než 10.
3. Graf se začne procházet u uzlů ve vzdálenosti 0, což je uzel N sám.
4. Postupně se vzdálenost zvětšuje o 1 a do výsledného řezu se zařazují všechny uzly, které jsou potomky všech rodičů v právě procházené vzdálenosti od uzlu N
5. Tento postup se opakuje, dokud se nedosáhne vzdálenosti m.

Avšak u hledání reconvergence-driven cut se jako limit pro procházení uzlů nepoužívá vzdálenost, ale celková velikost tohoto řezu. Při hledání pomocí CollectNodesTFI můžeme dostat řez o obrovské velikosti, zatímco u reconvergence-driven cut je maximální velikost předem dána. Hledání reconvergence-driven cut v systému ABC realizuje funkce `Abc_NodeFindCut`.

Abc_NodeFindCut

Vstupním parametrem této funkce je uzel, ze kterého hledání reconvergence-driven cut započne. Samo hledání je poté rekurzivní proces, ale oproti postupu v CollectNodesTFI dochází k mnoha změnám. Jeden průchod hledání začíná u předaného kořenového uzlu. Před započetím se inicializují dvě proměnné `CostBest` na 100 a ukazatel `pFaninBest`, který je na počátku prázdný. Následujícím krokem je cyklus `Vec_PtrForEachEntry`.

*Vec_PtrForEachEntry(Abc_Obj_t *, vLeaves, pNode, i)*

Tento cyklus tedy dostane pole `vLeaves`, které reprezentuje všechny listy, které byly v dosavadním průběhu nalezeny. V cyklu se tedy do proměnné `pNode` zapisují postupně všechny uzly z pole `vLeaves` a nad těmito uzly je volána funkce `Abc_NodeGetLeafCostOne`, která určí cenu daného uzlu.

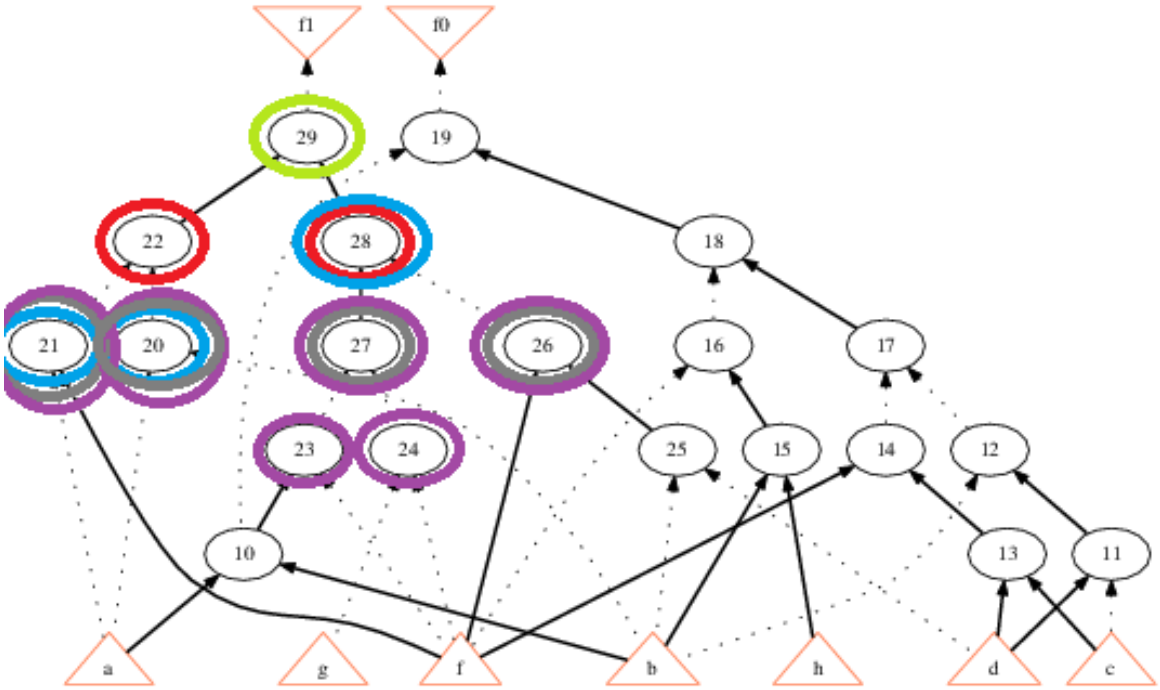
Cena pro uzel n se vypočítá jako $c = pocetPotomkuUzluN$. Po vypočtení ceny uzlu je provedeno porovnání cen:

```
if ( CostBest > CostCur ||
(CostBest == CostCur && pNode->Level > pFaninBest->Level) )
{
   CostBest   = CostCur;
   pFaninBest = pNode;
}
```

Tato podmínka specifikuje, že nejlepší cena se odvíjí od uzlu s největším počtem potomků. V běžném případě, kdy v AIG grafu mají uzly velmi často oba potomky, dochází k tomu, že jako nejlepší uzel s nejlepší cenou bude vybrán topologicky první z těch uzlů, které mají oba potomky. Tento postup zaručí to, že při hledání uzlů do reconvergence-driven cut se bude postupovat vždy přes ty uzly, které mají nejvyšší cenu.

Druhá část podmínky však toto tvrzení dále upravuje, a to tím způsobem, že při shodě cen musí mít přednost uzel, který má vyšší úroveň v grafu. Tato podmínka tedy zajistí, že při prohledávání AIG grafu shora dolů nebudeme přeskakovat úrovně.

Po nalezení uzlu s nejlepší cenou je provádění cyklu ukončeno a vracíme se zpět do funkce `Abc_NodeFindCut`, kde dojde k tomu, že potomci uzlu s nejlepší cenou jsou zařazeni do pole vLeaves, a také k vyřazení uzlu s nejlepší cenou z tohoto pole. Tím je zajištěno, že v poli `vLeaves` budou vždy jen listy výsledného reconvergence-driven cut. Toto zařazení je největší rozdíl oproti CollectNodesTFI, jelikož tam se vždy postupovalo rekurzivně směrem dolů v grafu po všech uzlech, zatímco při hledání reconvergence-driven cut se postupuje po potomcích, jejichž cena vyjde jako nejlepší. Nejlépe tuto situaci vystihuje následující obrázek.



Obrázek 3.9: Několik kroků tvorby reconvergence-driven cut

Barvy vyjadřují postupující kroky rekurze při hledání reconvergence-driven cut:

Krok 1 (zelená a červená barva) je reprezentován zelenou a červenou barvou. Zelená barva je kořenový uzel a červená jeho potomci.

Krok 2 (modrá barva) značí, že při průchodu cyklem pro potomky mají oba cenu rovnu dvě a pokud bychom brali jako topologické uspořádání pohled zleva doprava, musí dojít k tomu, že budou vybráni a přidáni potomci levého uzlu z dvou možných. Třetím uzlem v poli `vLeaves` zůstává jeden z potomků z prvního kroku.

Krok 3 (šedivá barva) ukazuje situaci, kdy dva ze tři uzlů mají cenu pouze jedna, jelikož mají pouze jednoho potomka, a proto jako nově přidaní potomci jsou vybráni potomci od uzlu z kroku 1.

Krok 4 (fialová barva) už pouze ukazuje další přidání potomků a naznačuje, jak by vypadal další průběh. Takovýmto způsobem jsou tedy uzly postupně přidávány do množiny reprezentující reconvergence-driven cut. Postupné přidávání je omezeno velikostí hledaného reconvergence-driven cut pomocí podmínky kontrolující, jestli už tato velikost nebyla překročena. Pokud k tomu dojde, je hledání ukončeno a reconvergence-driven cut je definován dvojicí kořenový uzel a nalezené pole potomků, kteří jsou v poli `vLeaves`.

Poslední krok algoritmu příkazu REFACTOR po nalezení reconvergence-driven cut je dán tím, že pro tento řez se vytvoří jeho faktorizovaná forma. Z této faktorizované formy se vytvoří AIG graf, který tuto formu reprezentuje. Pokud by po nahrazení části AIG grafu, která je reprezentovaná reconvergence-driven cut, AIG grafem vzniklým z faktorizované formy došlo ke zlepšení, je toto nahrazení akceptováno a AIG graf je patřičně upraven. Zlepšením může být snížení počtu AND uzlů nebo snížení počtu úrovní v AIG grafu.

3.3.2 Programové řešení příkazu REFACTOR

Implementace ve zdrojovém souboru abcFactor.c v balíku base/abci

Příkaz REFACTOR v systému ABC je spuštěn voláním funkce `Abc_Refactor` a opět je realizován cyklem `Abc_NtkForEachNode`, kde dochází k procházení všech uzlů v topologickém pořadí. V tomto cyklu je pro každý uzel nalezen jeho reconvergence-driven cut rekurzivním voláním funkce `Abc_NodeBuildCutLevelOne_int`, která jako svůj výsledek navrátí dvojici kořenový uzel a pole listů, které ohraničují reconvergence-driven cut. Po nalezení tohoto řezu se volá funkce `Abc_NodeRefactor`, v níž se provádí tvorba faktorizované formy.

Abc_ NodeRefactor

Při tvorbě faktorizované formy z reconvergence-driven cut potřebujeme tento řez mít reprezentován SOP formou. Konverze probíhá tím způsobem, že reconvergence-driven cut se převede na BDD reprezentaci (funkce `Abc_NodeConeBdd`) a následně se BDD reprezentace převede na SOP reprezentaci (funkce `Abc_ConvertBddToSop`).

Nad SOP reprezentací se už poté může začít tvorba faktorizované formy za pomocí postupu, zmíněného v části práce věnující se tvorbě AIG grafu, který využívá principů Booleovského dělení voláním funkce `Dec_Factor`.

Posledním krokem musí být zjišťování, jestli za pomocí této faktorizované formy dojde ke zlepšení. Pro provedení takového rozhodnutí potřebujeme vědět, o kolik uzlů AIG graf přijde, pokud by byl reconvergence-driven cut nahrazen. Tento výpočet se provádí zjištěním velikosti maximum fanout free cone daného kořenového uzlu, dále jen MFFC. MFFC můžeme definovat jako veškerou logiku uzlu, kterou můžeme vypustit, pokud by byl uzel z grafu odstraněn. Programově se to provádí voláním funkce `Abc_NodeMffcLabelAig`. Hodnota MFFC se uloží do proměnné `nNodesSaved`. Pro rozhodnutí, jestli je faktorizovaná forma výhodnější, se MFFC určí i pro ni a uloží do proměnné `nNodesAdded`.

Následně je proveden výpočet proměnné `nLastGain = nNodesSaved - nNodesAdded`. Pokud je hodnota `nLastGain` kladné číslo, dojde k tomu, že faktorizovaná forma nahradí původní reconvergence-driven cut a v proměnné `nLastGain` bude uložen počet uzlů, které tímto nahrazením ušetříme.

Pseudokód příkazu REFACTOR:

```
Refactor( network AIG )
{
   for each node N in the AIG in the topological order {
      r-dC = reconvergence-driven cut
      fForm = factorized form of r-dC
      nNodesSaved = |MFFC(r-dC)|
      nNodesAdded = |MFFC(fForm)|
      if ( nNodesSaved - nNodesAdded > 0 )
         UpdateAIG(r-dC, fForm)
   }
}
```

3.4 Příkaz RESUB

Příkaz RESUB[17] je úplně odlišný od příkazů REWRITE a REFACTOR. Jeho základním principem je substituce.

3.4.1 Teorie příkazu RESUB

Při práci algoritmu příkazu RESUB dochází k tomu, že funkce jednoho uzlu je nahrazena pomocí jiných uzlů, které jsou už v AIG grafu obsaženy. Těmto uzlům se říká divisory. Toto nahrazení se poté projeví v samotném grafu, pokud je v nějakém ohledu lepší než původní, kritériem je opět redukce počtu uzlů nebo úrovní v AIG grafu.

Nejlepším případem, který může při použití tohoto postupu v AIG grafu nastat, je to, když můžeme daný uzel a všechny další uzly, které patří do jeho MFFC, vypustit a nahradit je pouze jedním uzlem, který se už v grafu vyskytuje. Tento případ nese označení 0-resubstituce.

Dalšími případy od 0-resubstituce odvozenými jsou 1-resubstituce, kde dojde k nahrazení jednoho uzlu dvěma již existujícími uzly z AIG grafu a jedním navíc uměle vytvořeným uzlem. Tato varianta má význam jen pokud velikost MFCC nahrazovaného uzlu je více jak jeden uzel AIG grafu, což zaručí, že minimálně nedojde ke zhoršení.

Z toho plyne obecná formulace k-substituce, což je substituce při které dojde k přidání přesně k nových uměle vytvořených uzlů a ke zmenšení velikosti AIG grafu, pokud MFCC uzlu má alespoň velikost $k + 1$.

Pseudokód algoritmu je definován takto:

```
Resubstitution(AIG, int CutSizeLimit, int DivisorLimit, bool UseZeroCost) {
   for each node n in the AIG in the topological order {
      int MffcSize = | MFFC( n )|; assert( MffcSize >= 1 );
      nodeset C = ReconvergenceDrivenCut( n, CutSizeLimit );
      nodeset D = CollectNodesTFIChanged( C, Level(n), DivisorLimit );
      ComputeFunctions( D, n );
      nodeset R = Try0Resubstitution( n, D );
      if ( R == NULL && (MffcSize > 1 || (MffcSize == 1 && UseZeroCost)) )
         R = Try1Resubstitution( n, D );
      if ( R == NULL && (MffcSize > 2 || (MffcSize == 2 && UseZeroCost)) )
         R = Try2Resubstitution( n, D );
      if ( R == NULL && (MffcSize > 3 || (MffcSize == 3 && UseZeroCost)) )
         R = Try3Resubstitution( n, D );
      If ( R != NULL )
         UpdateNetwork( AIG, n, R );
   }
}
```

Při postupu pseudokódu se nejprve pro zkoumaný uzel vypočte velikost jeho MFFC a také nalezne jeho reconvergence-driven cut. Poté se spustí metoda CollectNodesTFOChanged, které jsou předány tři parametry:

- reconvergence-driven cut
- úroveň uzlu n
- limit pro počet divisorů - jako limit se používá 150 divisorů

Metoda CollectNodesTFIChanged je pozměněná verze metody CollectNodesTFI pro hledání řezu. Rozdíly mezi nimi jsou následující:

- Uzly jsou přidávány s ohledem na to, aby nepřekročily úroveň uzlu n.
- Listy v reconvergence-driven cut jsou brány v potaz, ale uzly z MFFC se přeskakují, protože budou odstraněny.
- Uzly se přidávají jen do limitu počtu divisorů stejně jako u tvorby reconvergence-driven cut - limit je opět 150.

Dalším krokem je výpočet booleovských funkcí nalezených uzlů v CollectNodesTFOChanged za pomocí literálů daných listy nalezeného řezu. Tento výpočet se provádí pomocí úplné simulace.

Následně se provádí pokusy o samotnou resubstituci, nejprve 0-resubstituce pokud booleovská funkce uzlu n je stejná jako některého z divisorů. Při nenalezení takového divisora se pokračuje 1-resubstitucí, kde dochází k hledání dvou divisorů, jejichž booleovské funkce jsou při provedení AND operace nad nimi rovny funkci uzlu n. Analogicky to funguje i pro 2-/3-resubstituce.

3.4.2 Programové řešení příkazu RESUB

Implementace ve zdrojovém souboru abcResub.c v balíku base/abci

Příkaz RESUB je spuštěn voláním funkce `Abc_NtkResubstitute`. Programové řešení tohoto příkazu se dá rozdělit na tři kroky:

- nalezení části AIG grafu, která by byla případnou resubstitucí nahrazena
- nalezení divisorů
- provedení úplné simulace a samotné resubstituce

Krok 1 - hledání vypustitelné logiky

Pro nalezení vypustitelné logiky pro právě zkoumaný uzel se použije postup velmi podobný hledání části AIG grafu pro převod na faktorizovanou formu u příkazu REFACTOR. Opět je voláno hledání reconvergence-driven cut, což zajišťuje funkce `Abc_NodeFindCut`. Zajímavým

rozdílem mezi příkazem REFACTOR, kde je reconvergence-driven cut omezen na velikost 10 uzlů, je to, že u příkazu RESUB je jeho velikost omezena na 8 uzlů.

Po nalezení reconvergence-driven cut je následně třeba nalézt jeho vnitřní uzly, které tvoří vypustitelnou logiku při resubstituci. To je zajištěno nalezením MFFC z tohoto řezu. Hledání MFFC je realizováno funkcí `Abc_NodeMffcInside`. Toto hledání už je prováděno ve funkci `Abc_ManResubEval`, která je cyklicky volána ve velkém cyklu `Abc_NtkForEachNode`, který prochází všechny uzly AIG grafu v topologickém pořadí.

Krok 2 - hledání divisorů

Toto hledání je voláno funkcí `Abc_ManResubCollectDivs`, která realizuje funkci CollectTFINodesChanged z pseudokódu. Postup při hledání se skládá ze tří kroků:

- Do pole divisorů jsou přidány listy reconvergence-driven cut
- Všechny uzly nalezené při určování MFFC jsou označeny takovým způsobem, aby při následujícím kroku nebyly přidány mezi divisory.
- Od uzlu n, který je zkoumán, se začne hledat jeho kužel s omezením na již navštívené uzly

Kužel pro uzel n

Kužel uzlu n jsou všechny uzly na cestách, které vedou z uzlu n do primárních vstupů, dalo by se tedy říct, že jsou to všechny uzly, na které narazíme při procházení všech cest vedoucích z uzlu n. Jedná se o obdobu metody CollectTFINodes, avšak je zde jeden podstatný rozdíl, a to takový, že je brán ohled na již navštívené uzly.

Pokud tedy při rekurzivním vykonávání tohoto hledání narazíme na již jednou navštívený uzel, což znamená uzel navštívený při příležitosti hledání divisorů pro uzel jiný než n nebo při hledání reconvergence-driven cut, dochází k tomu, že se tento uzel mezi divisory nezařadí.

Posledním krokem při hledání divisorů je, že se cyklicky projde pole dosud nalezených divisorů a pro každý z nich se zkontroluje, jestli jeho rodič nemá oba své potomky mezi už nalezenými divisory. Při splnění této podmínky je následně mezi divisory také zařazen. S takto naplněnou množinou divisorů se postupuje k dalšímu kroku.

Krok 3 - úplná simulace a resubstituce

Úplná simulace je spuštěna voláním funkce `Abc_ManResubSimulate`. Jejím cílem je určit booleovské funkce pro všechny divisory, takže je v cyklu prochází a na každý z nich spustí simulaci, kdy pro daný uzel zjišťuje jeho chování pro 2^k různých kombinací vstupních proměnných, kde k reprezentuje počet listů reconvergence-driven cut. Dále už jsou prováděny pouze samotné pokusy o resubstituci pomocí nalezených divisorů. Ta se, jak už bylo zmíněno v pseudokódu, může dělit na několik typů, které jsou realizovány následujícími funkcemi:

- 0-resubstituce pomocí `Abc_ManResubDivs0`
- 1-resubstituce pomocí `Abc_ManResubDivs1`
- 2-resubstituce pomocí `Abc_ManResubDivs2`
- 3-resubstituce pomocí `Abc_ManResubDivs3`

3.5 Příkaz RR

Příkaz RR[17] neboli odstranění redundance je zaměřený na odstraňování redundantních hran v AIG grafu.

3.5.1 Teorie příkazu RR

Principem příkazu RR a jeho algoritmu je odstranění redundantních hran v AIG grafu. To jsou takové hrany, které se žádným způsobem nepodílejí na celkový funkci obvodu realizovaného AIG grafem a jejich odstraněním tedy nedojde ke změně jeho funkce. Pokud se opět podíváme na pseudokód algoritmu vidíme, že toto odstranění se provádí následovně:

```
RedundancyRemoval( network AIG, int WindowSize, int Timeout )
{
   edgeset E = candidate AIG edges not disproved by random simulation;
   for each candidate edge in E  {
      W = CreateWindowForRR( E, WindowSize, WindowSize );
      if ( containing windows W of this size does not exist )
         continue;
      network Miter = ConstructMiter( W, E );
      if ( RandomSimulation(Miter r) // edge is not disproved by simulation
         if ( CheckRRusingSat( Miter r, Timeout ) // proved redundant by SAT
            UpdateNetwork( AIG, E );
   }
}
```

Nejprve je tedy spuštěna simulace, která jako svůj výsledek navrací uzly AIG, které by mohly být zdrojem redundance. Tomu odpovídají uzly, které mají více jak jednoho rodiče. Rodič je uzel, který je s podezřelým uzlem spojen hranou a nachází se ve vyšší úrovni grafu než sám podezřelý uzel. Uzel se stává podezřelým z redundance, pokud má více než jeden takový uzel. Po nalezení takto podezřelého uzlu je spuštěn algoritmus tvorby okénka.

3.5.1.1 Algoritmus tvorby okénka

Jedná se o algoritmus, který slouží k rozdělení AIG grafu na menší podgrafy, tedy jakási okénka, která se navzájem nepřekrývají. Jeho cílem je to, aby při nějakém postupu nad velkým AIG grafem se tento graf nemusel procházet uzel po uzlu, ale tento postup byl rozdělen do několika navzájem nezávislých částí, které jsou reprezentovány okénky.

Představme si modelovou situaci, ve které máme obrovský AIG graf a prakticky každá hrana v něm je podezřelá z redundance. Při nezavedení okének v takovéto situaci bychom pro ověření každé podezřelé hrany museli projít vždy celý AIG graf, což by bylo časově zbytečně náročné.

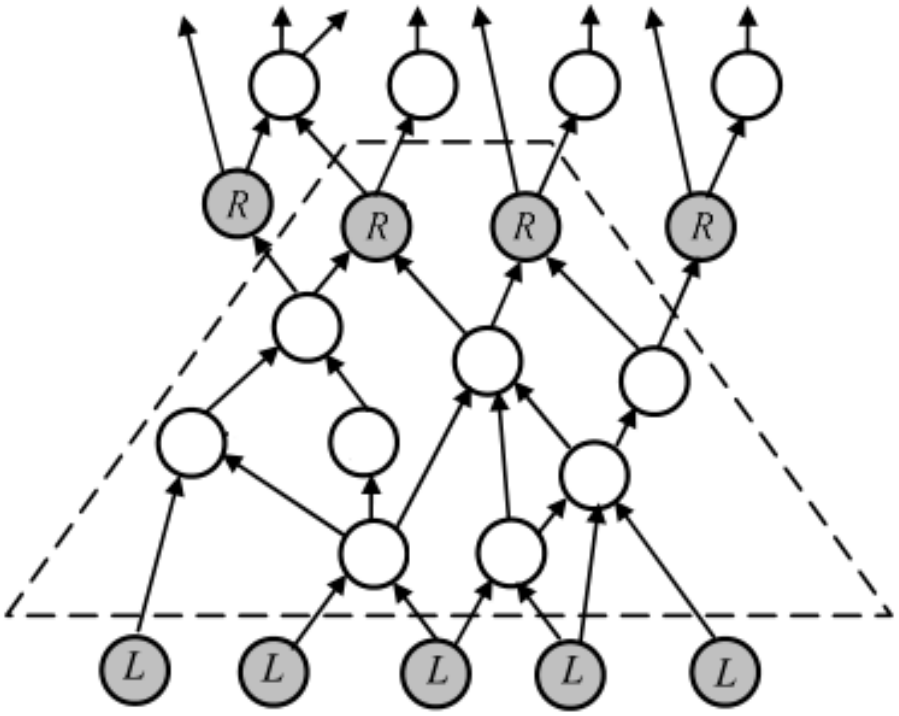
Okénko je definováno jako dvojice navzájem se nepřekrývajících množin. Tyto množiny nazýváme množina listů a množina kořenových uzlů. Vztah mezi nimi je definován tím, že každá cesta z primárních vstupů do kteréhokoliv z uzlů v kořenové množině projde přes nějaký uzel z množiny listů. Pro úplnou definici okénka je třeba ještě třetí množina uzlů, která obsahuje všechny uzly na cestách mezi množinami listů a kořenových uzlů.

Tvorba okénka jako takového je zajištěna voláním funkce `Abc_NtkRRWindow`. Před započetím tvorby okénka musí být stanoveno, od jaké hrany se má začít. To je definováno v dvojici `pEdgeFanout` a `pEdgeFanin`, která reprezentuje dva uzly nacházející se na koncích hrany podezřelé z redundance.

Dalším předpokladem pro úspěšné započetí tvorby okénka je nutnost znát jeho rozměr, ten je specifikován výškou okénka. Z výšky okénka můžeme určit hodnoty `LevelMin` a `LevelMax`, které reprezentují minimální a maximální úroveň v AIG grafu, kam až okénko může zasáhnout. Tyto dva údaje jsou dány takto:

```
LevelMin = ABC_MAX( 0, ((int)p->pFanin->Level) - p->nFaninLevels )
LevelMax = (int)pEdgeFanout->Level + p->nFanoutLevels
```

Pro lepší vysvětlení je dobré vyjít z obrázku reprezentujícího vzhled obvyklého okénka, který je převzat z článku[20] a upraven.



Obrázek 3.10: Ukázka okénka

Pro spodní hranici okénka, tedy `LevelMin`, bychom se měli dostat o jednu úroveň grafu níž, jak vyplývá z obrázku, a to z důvodu, že spodní hranice okna se určuje podle množiny listů, která se nachází v úrovni o jedna menší než je opravdová spodní hranice nalezeného okna. Minimální úroveň v AIG grafu se tedy odvíjí od uzlu `pFanin`, což je potomek uzlu n.

Po výpočtech těchto nezbytností může začít tvorba okénka, ta se skládá z několika kroků:

- Krok 1 - realizován funkcí `Abc_NtkRRTfi_int`
- Krok 2 - realizován funkcí `Abc_NtkRRTfo_int`
- Krok 3 - realizován funkcí `Abc_NtkRRTfo_rec`
- Krok 4 - realizován funkcí `Abc_NtkRRTfi_rec`

Krok 1 - Abc_NtkRRTfi_int

Tento krok spočívá v mírně upravené variantě metody CollectNodesTFI, což znamená že se rekurzivně prochází AIG graf směrem dolů od uzlu n až po uzly na úrovni `LevelMin`. Toto procházení je realizováno rekurzí sestupující po potomcích uzlů, avšak se dvěma rozdíly:

1. Po přidání všech potomků nějakého uzlu do množiny je tento uzel z množiny odebrán. Výsledkem rekurze je tedy to, že v množině zůstanou pouze naposledy přidaní potomci, což jsou listy grafu odpovídající listům L na ukázkovém obrázku okénka.
2. Při každém přidání je prováděna kontrola, jestli daný uzel nebyl už součástí nějakého jiného okénka.

Na konci tohoto kroku tedy existuje množinu listů okénka.

Krok 2 - Abc_NtkRRTfo_int

Jedná se o upravenou variantu metody CollectNodesTFO s tím rozdílem, že tentokrát směřujeme v AIG grafu směrem nahoru po rodičích jednotlivých uzlů. Výchozími uzly, ze kterých se začne postupovat směrem nahoru, je množina uzlů vzniklá v prvním kroku. Postup nahoru je zastaven ve chvíli, kdy je dosažena úroveň `LevelMax`. Při každém přidání je opět prováděna kontrola tří věcí:

1. uzel už byl procházen v kroku 1
2. uzel je součástí hrany podezřelé z redundance
3. uzel už je nad povolenou maximální úrovní `LevelMax`

Výstupem tohoto kroku je ale jiná množina, než v případě kroku 1. Množinou, která vznikne, je množina kořenových uzlů, které se nacházejí na horní hranici okénka, jak lze vidět na předešlém obrázku. Zde ovšem dochází k situaci, kdy tímto postupem nalezneme několik kořenových uzlů, ale z principu fungování kroku 1 a 2 může nastat stav, kdy některý z nalezených kořenových uzlů není dosažitelný z výchozího uzlu n.

Krok 3 - Abc_NtkRRTfo_rec

Třetí krok testuje dostupnost všech nalezených kořenových uzlů z výchozího uzlu n. V cyklu pro všechny nalezené kořenové uzly se zavolá rekurzivní funkce `Abc_NtkRRTfo_rec` implementující pozměněnou metodu CollectTFONodes. Změnou v této metodě je tentokrát to, že pro ověření dostupnosti kořenového uzlu z uzlu n je nutná podmínka, která stanoví, že do vzdálenosti o jedna větší než je rozsah okna, musíme být schopni nalézt cestu z uzlu n do právě v cyklu zkoumaného kořenového uzlu. Takto jsou tedy vyřazeny všechny kořenové uzly, které nesplňují podmínku, že patří do okénka o dané velikosti.

Krok 4 - Abc_NtkRRTfi_rec

Posledním krokem tvorby okénka je velký cyklus, který vezme zbylé kořenové uzly z kroku 3 a jeden po druhém je prochází. Nad každým uzlem je zavolána funkce `Abc_NtkRRTfi_rec`, která se stará o tvorbu finální podoby okénka.

Tato funkce je opět obdobou metody CollectTFINodes, avšak se třemi omezeními:

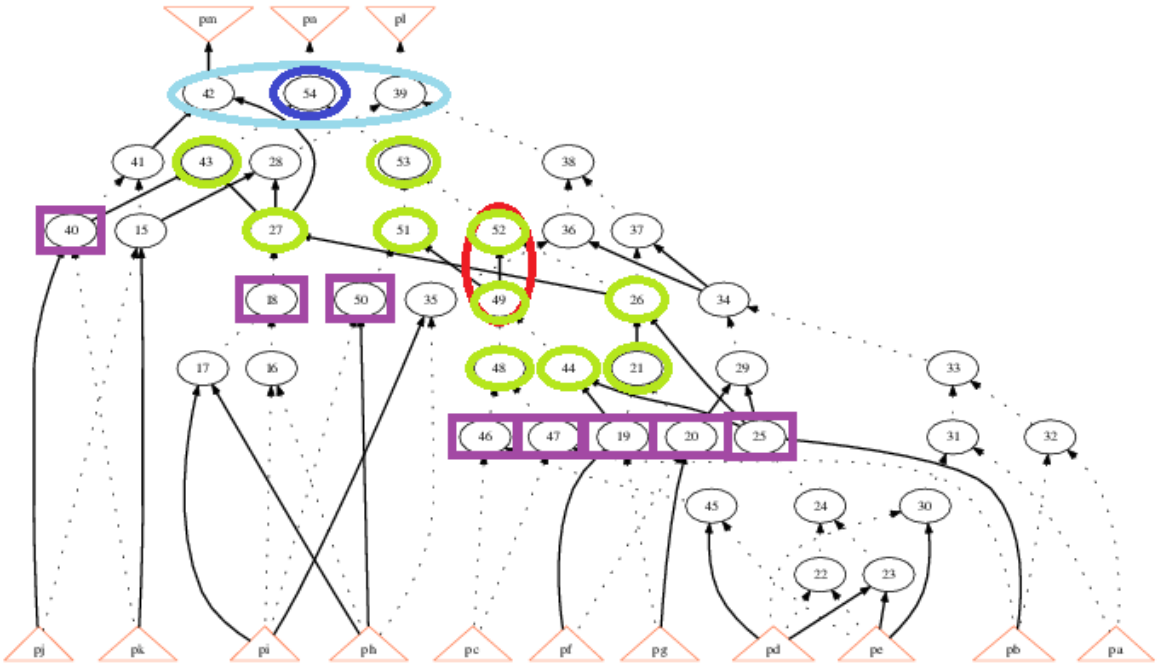
1. Do okénka se nebudou zařazovat uzly, které byly zařazeny v rámci tvorby nějakého jiného okénka.
2. Nebudou se podruhé zařazovat uzly, které už v okénku jsou, což jsou například uzly reprezentující hranu podezřelou z redundance nebo uzly, které už sama funkce při rekurzi navštívila.
3. Postup směrem dolů v AIG grafu při shromažďování uzlů je ukončen na úrovni dané `LevelMin`.

Při postupu této funkce vznikají tři množiny uzlů:

- `vRoot` - reprezentující všechny kořenové uzly okénka
- `vCone` - reprezentující vnitřní uzly okénka
- `vLeaves` - reprezentující listy okénka

Tyto tři množiny dohromady tedy tvoří okénko stejně jako je ukázáno na ilustrativním obrázku.

Při bližším prozkoumání tohoto průchodu se dá výsledek na příkladu pro okénko o velikosti 2 znázornit následovně:



Obrázek 3.11: Nalezené okénko velikosti 2

- Červená barva - dvojice uzlů tvořící hranu podezřelou z redundance
- Světle modrá barva - trojice kořenových uzlů, před ověřením jestli patří do okénka
- Tmavě modrá barva - `vRoot` - kořenový uzel splňující podmínku náležitosti do okénka
- Zelená barva - `vCone` - vnitřní uzly okénka
- Fialová barva - `vLeaves` - listy okénka

Po vytvoření okénka je dalším krokem příkazu RR konstrukce „miter“. Důvodem této konstrukce je zjištění, jestli je hrana podezřelá z redundance opravdu redundantní, nebo není. Konstrukce „miter“ se skládá ze tří částí:

1. Okénko obsahující hranu podezřelou z redundance.
2. Okénko neobsahující hranu podezřelou z redundance.
3. Přídavná logika, která zajistí operaci XOR dvojic výstupů z těchto okének stylem první s prvním, druhý s druhým a tak dále až po poslední dvojici výstupů a následné provedení operace OR nad všemi výstupy z operací XOR.

Po zkonstruování „miter“ dochází k otestování jeho splnitelnosti pomocí SAT řešiče. Hrana je redundantní a můžeme ji vypustit z AIG grafu, pokud je výsledkem nesplnitelnost. Celý tento princip konstrukce „miter“ tedy slouží k zjišťování funkční ekvivalence dvou vzniklých okének.

V případě potvrzeni redundantnosti hrany je posledním krokem algoritmu úprava AIG grafu, která spočívá v odebrání této hrany.

3.5.2 Programové řešení příkazu RR

Implementace ve zdrojovém souboru abcRR.c v balíku base/abci

Příkaz RR je reprezentován funkcí `Abc_NtkRR`. Nejprve je třeba spustit simulátor, který bude dodávat uzly, jež mají alespoň dva potomky, to jest mohou z nich vést hrany podezřelé z redundance. To je zajištěno voláním funkce `Abc_NtkRRSimulateStart`.

Poté následuje obvyklý velký cyklus `Abc_NtkForEachNode`, který zajišťuje průchod těmito uzly v topologickém uspořádání. V tomto cyklu je nejprve zavolána funkce `Abc_NtkRRWindow`, která pro daný uzel a jeho hranu podezřelou z redundance vytvoří okénko postupem, který byl podrobně rozepsán v části práce věnující se algoritmu tvorby okénka.

Následně je třeba zkonstruovat „miter“ a tím ověřit, jestli je hrana skutečně redundantní, což je provedeno voláním funkce `Abc_NtkRRProve`.

Závěrečným krokem při úspěšném potvrzení redundantnosti hrany je provedení náležitých úprav v AIG grafu funkcí `Abc_NtkRRUpdate`.

3.6 Příkaz COLLAPSE

Příkaz COLLAPSE převádí načtený obvod do realizace pomocí dvouúrovňové logiky.

3.6.1 Teorie příkazu COLLAPSE

Příkaz COLLAPSE provádí tento převod tím, že rekurzivně skládá vstupní uzly do výstupních uzlů, což vede k síti, ve které každý kombinační obvod je tvořen uzlem, jehož vstupním uzlem jsou kombinační vstupy.

V systému ABC je tento postup realizován tvorbou globálních funkcí reprezentovaných pomocí binárních rozhodovacích diagramů neboli BDD. Po provedení příkazu COLLAPSE jsou tedy uzly obvodu v systému ABC reprezentovány jako BDD uzly.

3.6.2 Programové řešení příkazu COLLAPSE

Implementace ve zdrojovém souboru abcCollapse.c v balíku base/abci

Příkaz COLLAPSE je reprezentován funkcí `Abc_NtkCollapse`. Při volání této funkce dochází k tvorbě globálních BDD pomocí funkcí `Abc_NtkBuildGlobalBdds` a `Abc_NtkMinimumBase`. Tyto funkce vytváří globální BDD za pomoci volání funkcí balíku CUDD, který je integrován do prostředí ABC.

Kapitola 4

Postupy randomizace příkazů systému ABC a jejich realizace

V systému ABC se všechny příkazy zmíněné v předešlé kapitole chovají deterministicky, což znamená, že při opakovaném spouštění nad stejným obvodem budeme dostávat stále stejné výsledky. Za opakované spuštění je považován postup, který se skládá ze dvou kroků:

1. načtení obvodu
2. spuštění příkazu

Tyto dva kroky se při opakovaném spuštění provádějí v celku, to znamená, že nemůžeme načíst obvod a pak nad ním spouštět opakovaně příkaz. Takovýto postup by nevedl ke korektním výsledkům, jelikož při opakovaném spuštění příkazu kontinuálně nad jedním obvodem dosáhneme pouze výsledků, které reprezentují citlivost daného obvodu na příkaz.

Pro případ randomizace algoritmů v systému ABC je tedy třeba zvolit postup, kterým chceme dosáhnou toho, aby se výsledky pro obvod nějakým způsobem lišily pro jednotlivá opakování. Randomizace tedy zajistí, že výsledky nebudou vždy jen deterministické, ale mohou a nemusí se pro jednotlivá opakování měnit. Pro dosažení tohoto chování můžeme postupy, které k němu vedou, rozdělit na dva typy:

- vnější randomizace
- vnitřní randomizace

Vnější randomizace, jak již bylo zmíněno v úvodu této práce, je postup, kdy nedochází k zásahům do algoritmů jednotlivých příkazů systému ABC. Randomizace je tedy zajištěna zvenčí nějakým jiným postupem. Pro lepší představu je možné upravit původní dvoukrokový postup provádění příkazů na následující:

1. provedení změn v obvodu
2. načtení obvodu
3. spuštění příkazu

Vnitřní randomizace je postup, kdy dochází k zásahům do algoritmů jednotlivých příkazů systému ABC. Prvkem, který zajišťuje randomizaci, je zásah přímo ve zdrojovém kódu daného příkazu. Tento zásah má tedy za cíl měnit logiku fungování příkazu. Pro lepší představu opět můžeme upravit původní dvoukrokový postup provádění příkazů na následující:

1. načtení obvodu
2. spuštění upraveného příkazu

Pokud tedy aplikujeme alespoň jednu z výše zmíněných variant randomizace, očekávaným výsledkem budou změny, které povedou k odlišným výsledkům oproti běžnému provádění příkazů v rámci systému ABC s deterministickým chováním.

Tyto rozdílné výsledky následně můžeme porovnat s výsledky získanými běžným postupem a zjistit tímto účinnost randomizovaných algoritmů.

Pro tyto potřeby byly tedy v rámci této práce implementovány následující postupy:

- permutace PI (primární vstup) a PO (primární výstup) pro příkazy BALANCE, REFACTOR, REWRITE, RESUB
- randomizace příkazů REFACTOR, REWRITE, RESUB
- randomizace příkazu RR
- randomizace příkazu COLLAPSE
- kombinace permutací a randomizovaných příkazů
- použití randomizace v reálném syntézním procesu

4.1 Různé permutace pro příkazy BALANCE, REFACTOR, REWRITE, RESUB

Základním prvkem tohoto přístupu k randomizaci je náhodná permutace. Zjednodušeně řečeno je permutace nějaký způsob setřídění prvků v množině takovým způsobem, že toto setřídění je náhodné. Po každé permutaci tedy dostáváme náhodným způsobem setříděné prvky v množině.

Tohoto postupu se dá velmi efektivně využít pro zavedení vnější randomizace v rámci systému ABC, a to takovým způsobem, že před načtením obvodu do systému ABC provedeme náhodnou permutaci nad nějakými prvky obvodu. Pokud vezmeme v úvahu, že do systému ABC načítáme obvody v BLIF formátu, tedy v SOP reprezentaci, vyplyne z toho, že těmito permutovanými prvky mohou být:

- primární vstupy obvodu
- primární výstupy obvodu
- uzly obvodu
- termy uzlu obvodu

Permutace všech čtyř prvků může u nějakého jednoduchého obvodu vypadat například takto:

```
.model before_permute        .model after_permute
.inputs a b c d e f          .inputs d b e a c f
.outputs g h i               .outputs h i g
.names a b g                 .names e f i
11  1                        00 0
.names b c d h               .names a b g
11-  1                       11 1
1-1  1                       .names b c d h
-11  1                       1-1 1
.names e f i                 11- 1
00  0                        -11 1
.end                         .end
```

Obrázek 4.1: Příklad permutace obvodu

Z obrázku vyplývá, že díky permutaci bylo změněno pořadí všech čtyř možných prvků. Pro další spuštění permutace bychom zase dostali jinak náhodně uspořádané prvky.

K provedení permutace obvodu můžeme použít dva způsoby:

- externí program
- interní realizace permutace v systému ABC

Externí program

Jako externí program pro provádění permutací nad obvody byl použit program od Ing. Petra Fišera. Jedná se o program pracující v příkazové řádce, který jako svůj vstup vyžaduje soubor s obvodem ve formátu BLIF a produkuje permutovaný obvod do souboru opět ve formátu BLIF. Program umí permutovat všechny čtyři prvky obvodu. Toto chování je ukázáno na obrázku ukázky permutace a volba se provádí pomocí přepínačů, které značí, který z prvků se má permutovat.

Interní realizace permutace v systému ABC

V rámci systému ABC je také implementována možnost permutace různých části obvodu. Tato implementace je realizována funkcí `Abc_NtkPermute`. Na rozdíl od použitého externího

programu pro permutace umí interní implementace permutovat primární vstupy, primární výstupy a klopné obvody.

Dalším podstatným rozdílem oproti permutacím za použití externího programu je to, že pro její činnost je nezbytný generátor náhodných čísel a ten je v systému ABC inicializován konstantou. Z toho plyne, že pro každé spuštění systému ABC bude náhodný generátor použitý pro permutace generovat pokaždé ta samá čísla.

Tyto dvě varianty mohou tedy vést ke dvěma odlišným způsobům, jak permutace provádět. Pokud vezmeme v potaz provádění permutací externím programem, výsledný postup bude následující:

1. spustit externí program a provést permutace
2. spustit systém ABC a načíst permutovaný obvod
3. provést příkazy
4. vypnout systém ABC

Velkou nevýhodou tohoto způsobu je to, že systém ABC musí být po dokončení tohoto cyklu a spuštění dalšího například pro jinou permutaci vypnut a znova zapnut. Tato nevýhoda se projevuje velkou časovou náročností řešení, kdy pro desítky tisíc opakování je toto zdržení už značné.

Při provádění permutací interně v rámci systému ABC je postup následující:

1. spustit systém ABC - krok který se už dále neopakuje
2. načíst obvod v původním stavu
3. provést permutaci
4. provést příkazy

Z tohoto postupu vyplývá velká výhoda oproti provádění permutací pomocí externího programu spočívající v tom, že nemusíme pro každé opakování znovu spouštět systém ABC. Systém ABC je spuštěn pouze na začátku a dokud se nedokončí potřebné opakování, zůstává stále zapnut.

Určitou nevýhodou tohoto řešení je, jak už bylo výše zmíněno, inicializace generátoru náhodných čísel na konstantu. Pokud si ale představíme situaci, kdy systém ABC jednou zapneme a pak provádíme desítky tisíc opakování permutací, je tento problém svým způsobem eliminován a dostaneme srovnatelné řešení s použitím externího programu pro permutace.

Díky možnosti permutovat obvod jedním z těchto dvou způsobů můžeme říct, že jsou splněny všechny nezbytné předpoklady pro testování této varianty randomizace.

4.1.1 Realizace experimentů s externím programem pro permutace

Při tomto způsobu realizace se provádění dá popsat pomocí pseudokódu:

```
for (i in obvody[]) {
   for (j in permutace[]) {
      for (k in prikazy[]) {
         for (pocetIteraci) {
            skriptProABC += nactiPermutovanyObvod[i, j];
            skriptProABC += prikaz[k];
            skriptProABC += zapisVysledek[i, j, k];
            provedPermutaci(i, j);
            spustABC(skriptProABC);
         }
      }
   }
}
```

Pro to, aby mohly být experimenty prováděny, je třeba znát čtyři věci:

1. pole `obvody[]` reprezentující obvody, pro které se budou experimenty provádět
2. pole `permutace[]` ve kterém je dáno, která ze čtyř věcí v obvodu se bude permutovat
3. pole `prikazy[]` značící jaké příkazy se budou nad obvodem provádět
4. hodnotu `pocetIteraci`, která určuje, kolik opakování se bude nad daným obvodem provádět

Máme-li tedy definované tyto čtyři údaje, je pak zřejmé, že průběh pro dva obvody s permutacemi primárních vstupů(PI) a výstupů(PO) a pro příkazy refactor, rewrite při dvou opakováních by byl například následující:

Pro obvod 1, permutaci PI, příkaz 1, iterace 1, vytvořím skript pro ABC, permutuji obvod, spouštím systém ABC se skriptem nad tímto permutovaným obvodem, iterace 2, příkaz 2, iterace 1, iterace 2, permutace PO, příkaz 1, iterace 1, iterace 2, příkaz 2, iterace 1, iterace 2 a stejným způsobem pro obvod 2.

Důležitým krokem v tomto postupu je tvorba skriptu a jeho spouštění v systému ABC. Skript pro systém ABC je textový soubor, který obsahuje libovolný počet příkazů systému ABC, které se zapisují samostatně na nové řádky. Jednoduchý skript vypadá tedy takto:

```
read test.blif
strash
refactor
print_stats
```

Spuštění takového skriptu v systému ABC je umožněno zavoláním programu `abc.exe` s přepínačem `-f` a souborem skriptu odděleným mezerou.

4.1.2 Realizace experimentů s využitím interní permutace systému ABC

Opět můžeme tuto realizace definovat pomocí pseudokódu:

```
for (i in obvody[]) {
   for (j in permutace[]) {
      for (k in prikazy[]) {
         for (pocetIteraci) {
            skriptProABC += nactiOriginalniObvod[i];
            skriptProABC += provedInterniPermutaci[j];
            skriptProABC += prikaz[k];
            skriptProABC += zapisVysledek[i, j, k];
         }
      }
   }
}
spustABC(skriptProABC);
```

Oproti předchozímu způsobu jsou zde dva rozdíly. Prvním je to, že tedy není využívána permutace pomocí externího programu a její provedení je tedy začleněno do tvorby skriptu pro systém ABC, jelikož je permutace volána jako příkaz systému ABC.

Druhým podstatnějším rozdílem je to, že celý skript pro provádění je vygenerován najednou a až po tomto vygenerování dochází ke spuštění systému systému ABC. Tím je tedy zajištěno to, že se systém ABC spustí pouze jednou a ne opakovaně.

4.2 Randomizace příkazů REFACTOR, REWRITE, RESUB

Tento způsob randomizace už spadá do kategorie vnitřní randomizace, jelikož pro jeho realizaci je nutno zasáhnout přímo do algoritmů v systému ABC.

Společným znakem příkazů REFACTOR, REWRITE, RESUB je přítomnost jednoho velkého cyklu `Abc_NtkForEachNode`, který řídí celou jejich činnost. V původní implementaci tento cyklus zajišťuje průchod přes všechny uzly v grafu v topologickém uspořádání. Uzlem je v tomto případě myšlen uzel takový, který není primárním či kombinačním vstupem ani výstupem.

Topologické uspořádání uzlů je dáno už při konstrukci AIG grafu reprezentujícího daný obvod. Pro představu to znamená, že každý uzel má přiřazené své `Id`, které jej jednoznačně identifikuje, a tato `Id` jsou uzlům přidělována v topologickém pořadí, které je dáno úrovní v AIG grafu. Pole uzlů AIG grafu potom tedy začíná od uzlu s `Id` 1 až po uzel s `Id` n, kde n je počet uzlů v grafu.

Ve skutečnosti je cyklus `Abc_NtkForEachNode` reprezentován tímto kódem:

```
#define Abc_NtkForEachNode( pNtk, pNode, i )
   for ( i = 0; (i < Vec_PtrSize((pNtk)->vObjs)) &&
   (((pNode) = Abc_NtkObj(pNtk, i)), 1); i++ )
      if ( (pNode) == NULL || !Abc_ObjIsNode(pNode) ) {} //nedělej nic
      else //prováděj kód
```

Při podrobném pohledu na cyklus je vidět, že prochází pole uzlů zleva doprava podle `Id`. Uzly jsou tedy v poli seřazeny v topologickém uspořádání, díky čemuž cyklus takovýto průchod zajišťuje.

Velmi důležitým prvkem tohoto cyklu je splnění podmínky `Abc_ObjIsNode`, která slouží ke kontrole, zda právě zkoumaný uzel `pNode` je vůbec uzlem vyskytujícím se v aktuálním AIG grafu. Tato podmínka je velmi důležitým prvkem, jehož význam bude v dalším textu vysvětlen.

4.2.1 Provedení randomizace příkazů REFACTOR, REWRITE, RESUB

Pro dosažení randomizace algoritmů těchto příkazů se dá využít cyklu `Abc_NtkForEachNode`, a to takovým způsobem, že cyklus nebude uzly procházet v topologickém pořadí zleva doprava, ale bude je procházet v náhodném pořadí za podmínky zachování **topologického pořadí**.

Topologické pořadí v AIG grafu je vyjádřeno pomocí úrovní AIG grafu. To znamená, že uzly s nejnižším topologickým pořadím jsou ty uzly, které se nacházejí v grafu úplně dole, tedy na úrovni 1 a zároveň splňují podmínku, že se nejedná o primární vstup nebo výstup. Dále se v poli uzlů nacházejí takové, které mají úroveň v AIG grafu 2 a tak dále až po nejvyšší úroveň v AIG grafu.

Můžeme si tedy pole uzlů AIG grafu představit jako n menších polí, kde n je počet úrovní AIG grafu, která jsou naplněna uzly s odpovídající úrovní. Pro zachování podmínky průchodu v topologickém pořadí je při náhodném průchodu přes uzly nutné nejprve náhodně zvolit všechny uzly v poli s uzly o úrovni 1 a ve chvíli, kdy už budou uzly o úrovni 1 vyčerpány, dochází k přechodu na uzly o úrovni 2 a tak dále až po pole s uzly nejvyšší úrovně. Takovýto průchod je realizován tím, že dojde k nahrazení cyklu `Abc_NtkForEachNode` jiným kódem, který vypadá následovně:

```
ntkLevel = Abc_AigLevel(pNtk);
for (i = 1; i <= ntkLevel; i++) {
   Abc_GetNodesByLevel( pNtk, levelNodes, i);
   for (j = 0; j < levelNodes->count; levelNodes->count--) {
      k = rand() \% levelNodes->count;
      pNode = (Abc_Obj_t *)Vec_PtrEntry(levelNodes->vNodes, k);
      if ((pNode) == NULL || Abc_ObjIsNone(pNode)) {
         Vec_PtrRemovePosition( levelNodes->vNodes, k );
         continue;
      }
//zbylý kód příkazu
}
```

Pro funkci tohoto kódu jsou zapotřebí dvě věci:

Abc_ GetNodesByLevel

```
Abc_LevelNodes_t * Abc_GetNodesByLevel ( Abc_Ntk_t * pNtk,
Abc_LevelNodes_t * levelNodes, int Level ) {
   int i, nodeLevel;
   Abc_Obj_t * pNode;

   levelNodes->count = 0;
   for ( i = 0; (i < Vec_PtrSize((pNtk)->vObjs)) &&
   (((pNode) = Abc_NtkObj(pNtk, i)), 1); i++ )
      if ( (pNode) == NULL || !Abc_ObjIsNode(pNode) ) {
         Vec_PtrRemovePosition( (pNtk)->vObjs, i );
         i--;
         continue;
      } else
         nodeLevel = pNode->Level;
         if (Level == nodeLevel) {
            Vec_PtrPush( levelNodes->vNodes, pNode );
            levelNodes->count++;
         }
      }
   return levelNodes;
}
```

A také struktura `levelNodes`, která reprezentuje právě ty pole AIG uzlů obsahující pouze uzly dané úrovně. Druhou její proměnnou je proměnná `count` reprezentující počet uzlů v dané úrovni.

Z pseudokódu je zřejmé, že prvním krokem je zjištění maximální úrovně grafu pomocí volání funkce `Abc_AigLevel`. Potom v cyklu procházíme úroveň za úrovní, kde pro každou úroveň na začátku sesbíráme všechny uzly AIG grafu na této úrovni zavoláním funkce `Abc_GetNodesByLevel`. Té jako parametr předáme číslo úrovně, pro kterou hledáme uzly, a ona poté projde všechny uzly AIG grafu a do struktury `levelNodes` je uloží i s jejich počtem.

Uzly se následně v cyklu procházejí a voláním `k = rand() \% levelNodes->count` se vždy jeden z nich náhodně vybere. Po tomto výběru dochází ke kroku, v němž kontrolujeme, zda-li je zvolený uzel součástí aktuálního AIG grafu. Tento test je naprosto zásadní pro správnou funkci randomizace.

Test existence uzlu - Abc_ObjIsNode(pNode)

Důležitost tohoto testu spočívá v tom, že při jakékoliv změně, kterou některý ze tří randomizovaných příkazů v AIG grafu provede, může dojít k tomu, že nějaký uzel bude z AIG grafu vypuštěn. Toto vypuštění je symbolizováno tím, že vypuštěnému uzlu je změněno jeho `Id` na 0.

K takovéto změně může dojít při kterémkoliv průchodu cyklem, který zpracovává jednotlivé uzly z úrovně AIG grafu. Například bychom měli uzly 1, 2, 3, 4, 5, 6 v jedné úrovni. Při provádění příkazu nad uzlem 3 může dojít k tomu, že z grafu bude vypuštěn nejenom uzel 3, ale například i uzel 4. Po dokončení tohoto průchodu započne další pro uzel 4. Ten již není uzlem AIG grafu a má `Id` 0. Proto je nutné nejprve provést tento test a při zjištění, že daný uzel už uzlem AIG grafu není, dojde k tomu, že je automaticky vypuštěn z pole.

Poslední změnou nutnou pro správně fungování tohoto způsobu randomizace jsou zásahy do míst, kde při provádění původního cyklu `Abc_NtkForEachNode` dochází k volání příkazu `continue`, což značí přerušení provádění cyklu pro daný uzel a pokračování s dalším uzlem. Na těchto místech je třeba zajistit, aby byl uzel ze struktury `levelNodes` také vyňat.

To je provedeno doplněním příkazu `Vec_PtrRemovePosition(levelNodes->vNodes, k)` všude tam, kde je volán příkaz `continue`.

Stejný příkaz musí být také vložen na závěr celého cyklu pro každý uzel, kdy už je jasné, že příkaz REFACTOR, REWRITE nebo RESUB byl celý pro daný uzel úspěšně vykonán. Opět to proběhne přidáním řádku `Vec_PtrRemovePosition(levelNodes->vNodes, k)`.

Celý postup randomizace je do systému ABC zapracován jako volitelný přepínač `-r` pro všechny tři příkazy REFACTOR, REWRITE a RESUB, takže při jejich voláni je možno zvolit způsob, jakým bude příkaz prováděn.

4.3 Randomizace příkazu RR

Způsob randomizace příkazu RR je velmi podobný postupu uvedenému v předchozí části. Rozdílem je zde to, že příkaz RR nepracuje při svém průchodu pouze nad jednotlivými uzly AIG grafu, ale bere v potaz hranu podezřelou z redundance, která je tvořena ne jedním ale dvěma uzly.

Dalším rozdíl vyplývá z toho, že v příkazu RR je nejprve prováděna simulace pro zjištění hran podezřelých z redundance. Díky tomuto je nutné nejprve nechat doběhnout tuto simulaci a teprve až nad uzly AIG grafu, které tato simulace označí, provádět postup podobný tomu z předešlé části.

Opět se tedy bude jednat o úpravu cyklu `Abc_NtkForEachNode` se zachováním průchodu v topologickém uspořádání.

4.3.1 Provedení randomizace příkazu RR

Randomizace z postupu pro příkazy REWRITE, REFACTOR, RESUB musí být upravena pro práci se dvěma uzly. Další nutnou úpravou je to, aby pracovala nad polem uzlů vzniklých při simulaci hran podezřelých z redundance.

Toto pole je definováno strukturou `pivots`, do které se nejprve umístí všechny uzly, které jsou podezřelé z redundance, a až teprve nad touto strukturou se provádí operace, který jsou dány tímto kódem:

```
ntkLevel = Abc_AigLevel(pNtk);
for (l = 1; l <= ntkLevel; l++) {
   Abc_GetNodesByLevel( pivots, levelNodes, l);
   for (j = 0; j < levelNodes->count; levelNodes->count--) {
      k = rand() \% levelNodes->count;
      pPivot = (Abc_Obj_t *)Vec_PtrEntry(levelNodes->vNodes, k);
      pFanin = (Abc_Obj_t *)Vec_PtrEntry(levelNodes->vFanins, k);
      if ( ((pPivot) == NULL || Abc_ObjIsNone(pPivot)) ||
      ((pFanin) == NULL ) || Abc_ObjIsNone(pFanin)) {
         Vec_PtrRemovePosition( levelNodes->vNodes, k );
         Vec_PtrRemovePosition( levelNodes->vFanins, k );
         continue;
      }
//zbylý kód příkazu
}
```

Tento kód se vykonává až po vytvoření struktury `pivots` a díky tomu je samotná funkce reprezentující příkaz RR rozdělena na dvě funkce. Původní funkce `Abc_NtkRR` vykoná všechny kroky nutné pro vytvoření pole `pivots` pomocí simulace, až poté je zavolána nově implementovaná funkce `Abc_NtkRRPivots`, která zajišťuje randomizaci dle výše uvedeného kódu a také provádí samotný algoritmus příkazu RR.

Hrana podezřelá z redundance je tedy reprezentována dvěma uzly `pPivot` a `pFanin`, kde `pPivot` je uzel procházený stejně jako v rámci cyklu `Abc_NtkForEachNode` a uzel `pFanin` značí spodní uzel této hrany. Pro tuto funkčnost musí být také náležitě upravena struktura `levelNodes` shromažďující uzly se stejnou úrovní v AIG grafu, a to tím způsobem, že je do ní navíc přidáván i druhý uzel reprezentující hranu podezřelou z redundance.

Náležitě upravena je také funkce `Abc_GetNodesByLevel` sloužící k shromažďování uzlů AIG grafu dle jejich úrovně v něm.

```
Abc_LevelNodes_t * Abc_GetNodesByLevel ( Abc_RRPivots_t * pivots,
Abc_LevelNodes_t * levelNodes, int Level ) {
   int i, nodeLevel;
   Abc_Obj_t * pPivot, * pFanin;

   levelNodes->count = 0;
   for (i = 0; i < pivots->count; i++) {
      pPivot = (Abc_Obj_t *)Vec_PtrEntry(pivots->vPivots, i);
      pFanin = (Abc_Obj_t *)Vec_PtrEntry(pivots->vFanins, i);
      if ( ((pPivot) == NULL || Abc_ObjIsNone(pPivot)) ||
      ((pFanin) == NULL ) || Abc_ObjIsNone(pFanin)) {
         Vec_PtrRemovePosition( pivots->vPivots, i );
         Vec_PtrRemovePosition( pivots->vFanins, i );
         pivots->count--;
         i--;
         continue;
      }
```

```
      nodeLevel = pPivot->Level;
      if (Level == nodeLevel) {
         Vec_PtrPush( levelNodes->vNodes, pPivot );
         Vec_PtrPush( levelNodes->vFanins, pFanin );
         levelNodes->count++;
      }
   }
   return levelNodes;
}
```

Nezbytným krokem stejně jako u randomizace příkazů REFACTOR, REWRIE, RESUB je přidání příkazů pro odstranění uzlů ze struktury `levelNodes` na místa, kde je to nutné. Tentokrát se však jedná o dva řádky:

```
Vec_PtrRemovePosition( levelNodes->vNodes, k )
Vec_PtrRemovePosition( levelNodes->vFanins, k )
```

Celý postup randomizace je do systému ABC opět zapracován jako volitelný přepínač `-r`, takže existuje možnost volby, zda tento postup využít nebo ne.

4.4 Randomizace příkazu COLLAPSE

V případě případu COLLAPSE nejde o žádný zásah do algoritmu příkazu, ale o stejný postup jako u prvně zmíněného způsobu randomizace pro příkazy BALANCE, REFACTOR, REWRITE, RESUB pomocí permutací. Permutace v případě příkazu COLLAPSE má z principu tvorby globálních BDD smysl pouze pro permutace primárních vstupů. Tyto primární vstupy se v tomto případě reprezentují jako vstupní proměnné a při tvorbě globálních BDD dosti závisí na jejich pořadí.

Pro různá pořadí vstupních proměnných může příkaz BALANCE produkovat výsledky značně odlišné, takže je nutnost tento efekt nějakým způsobem redukovat, jelikož v opravdu špatném případě se může příkaz COLLAPSE i pro malý obvod stát neproveditelným. V případě běžného použití příkazu COLLAPSE jsou vstupní proměnné před započetím tvorby globálních BDD seřazeny pomocí algoritmu prosévání[23], což by mělo zajistit to, že provádění příkazu nebude produkovat obrovské výsledky. Existují ale i takové případy, kdy ani použití algoritmu prosévání nemusí zajistit proveditelnost COLLAPSE příkazu.

V případě vypnutí používání algoritmu prosévání je možným způsobem randomizace využití permutace primárních vstupů pomocí externího programu nebo interně implementované permutace v systému ABC. Využití interně implementované permutace je stejné jako při permutacích pro příkazy BALANCE, REFACTOR, REWRITE, RESUB za předpokladu permutace pouze primárních vstupů.

Balík CUDD, pomocí něhož systém ABC realizuje tvorbu globálních BDD, ale umožňuje algoritmus prosévání pro řazení vstupních proměnných nejenom vypnout, ale i zvolit jiný. Jednou z možností je i zapnutí náhodného seřazení vstupních proměnných, což je řešení srovnatelné s použitím externího programu i interní permutace.

Při zapnutí náhodného řazení vstupních proměnných nebo při použití permutací primárních vstupů však může dojít k obrovskému nárůstu počtu BDD uzlů. Tomuto chování je nutno předejít omezením jejich počtu, což příkaz COLLAPSE umožňuje volbou přepínače `-B` následovaným maximálním počtem vytvořených BDD uzlů odděleným mezerou.

4.4.1 Provedení randomizace příkazu COLLAPSE

V případě volby externího programu nebo interní permutace je postup popsán v kapitole 4.1 a zůstane stejný až na to, že bude volena pouze permutace primárních vstupů. Implementace druhé možnosti je realizována přidáním následujícího kódu do implementace funkce `Abc_NtkBuildGlobalBdds`, která zajišťuje tvorbu globálních BDD pro příkaz COLLAPSE:

```
if ( fRandomVariables ) {
   Cudd_AutodynEnable( dd, CUDD_REORDER_RANDOM );
} else {
   if ( fReorder )
      Cudd_AutodynEnable( dd, CUDD_REORDER_SYMM_SIFT );
}
```

Proměnná `fReorder` reprezentuje přepínač `-r` náležící příkazu COLLAPSE, který je automaticky při volání příkazu zapnut. Proto byl přidán nový přepínač `-R`, který je reprezentován proměnnou `fRandomVariables`.

Pokud je zvolen při spuštění příkazu přepínač `-R`, dochází k tomu, že je funkce přepínače `-r` ignorována a balíku CUDD je nastaveno, aby při řazeni vstupních proměnných použil náhodné řazení.

4.5 Kombinace permutací a randomizovaných příkazů

Dalším možným randomizačním postupem, který vychází z už popsaných postupů, je kombinace vnější randomizace pomocí permutací různých částí obvodů a vnitřní randomizace úpravami v algoritmech příkazů systému ABC. Pro tento postup je tedy využíváno permutací pomocí externího programu permute a příkazů, jejichž způsob randomizace je už popsán v této práci.

Pro tuto kombinaci byly jako randomizované příkazy zvoleny příkazy REWRITE, REFACTOR, RESUB a RR.

4.5.1 Provedení randomizace s kombinací permutací a příkazů

Způsob provedení v tomto případě vychází z postupů uvedených v kapitole 4.1, ve které je popsáno, jakým způsobem se provede randomizace za pomocí permutací nad obvody pro neupravené příkazy BALANCE, REWRITE, REFACTOR, RESUB, a v kapitole 4.2 a 4.3, kde jsou uvedeny postupy pro randomizaci příkazů REWRITE, REFACTOR, RESUB a RR.

Využije se tedy stejný postup pro provádění testování jako v kapitole 4.1, avšak s použitím randomizovaných variant příkazů REWRITE, REFACTOR, RESUB a RR. Použije se i stejný pseudokód pro testovaní, avšak se dvěma úpravami:

1. provedou se permutace primárních vstupů a primárních výstupů obvodů najednou
2. v poli `prikazy[]` se použijí randomizované varianty příkazů

4.6 Použití randomizace v reálném syntézním procesu

Možným využitím randomizace příkazů v systému ABC, která je popsána v kapitolách 4.2 a 4.3, je její zařazení do reálných syntézních procesů. Tyto procesy jsou v systému ABC reprezentovány mnoha různými postupy, ve kterých je možno nahradit běžně používané varianty příkazů REWRITE, REFACTOR, RESUB A RR za jejich randomizované varianty.

4.6.1 Provedení randomizace v reálném syntézním procesu

Pro účely této práce byly randomizované varianty příkazů zařazeny do dvou postupů reálných syntézních procesů.

Mapování na standardní buňky

Tento syntézní proces je realizován příkazy CHOICE a MAP. Příkaz CHOICE reprezentuje v systému ABC syntézní skript, který je kombinací dvou syntézních skriptů, a to skriptů RESYN a RESYN2. Tyto dva skripty v sobě zahrnují posloupnosti provádění příkazů systému ABC, ve kterých jsou využívány příkazy BALANCE, REWRITE a REFACTOR. Bylo tedy nutno vytvořit nový skript, ve kterém došlo k nahrazení původních příkazů systému ABC jejich randomizovanými variantami.

Příkaz MAP poté slouží k provedení samotného mapování a není u něj nutno provádět jakoukoliv změnu.

Mapování na „look-up tables“ neboli LUT

Realizace tohoto syntézního procesu se skládá z příkazů CHOICE, IF a LUTPACK. Příkaz CHOICE je nutno upravit stejně jako v případě mapování na standardní buňky. Příkazy IF a LUTPACK opět zůstanou beze změny.

Postup randomizace se v případě mapování na standardní buňky dá popsat tímto pseudokódem:

```
skriptProABCOrig += read_library;
for (i in obvody[]) {
   skriptProABCOrig += nactiObvod[i];
   for (pocetIteraci) {
      skriptProABCOrig += choice;
      skriptProABCOrig += map;
      skriptProABCOrig += zapisVysledek[i];
   }
}
spustABC(skriptProABCOrig);
skriptProABCR += read_library;
for (i in obvody[]) {
   skriptProABCR += nactiObvod[i];
   for (pocetIteraci) {
      skriptProABCR += choicer;
      skriptProABCR += map;
      skriptProABCR += zapisVysledek[i];
   }
}
spustABC(skriptProABCR);
```

Z pseudokódu je vidět, že testování se skládá ze dvou částí. V první části se provede celý postup s původními příkazy systému ABC, což reprezentuje příkaz CHOICE. Druhá část provede testování s randomizovanými variantami příkazů, tedy příkaz CHOICER, který je upravenou variantou příkazu CHOICE.

Důvodem pro takovéto dvojité provádění je nutnost získání výsledků pro možnost porovnání mezi randomizovanou a nerandomizovanou variantou. Oba skripty, které jsou následně spuštěny v systému ABC, začínají příkazem `read_library`, který zajišťuje načtení knihovny standardních buněk, na kterou bude mapování probíhat.

Pro případ mapování na „look-up tables“ neboli LUT se využije podobný postup jako v případě mapování na standardní buňky. Opět se bude jednat o dvě části, které zajistí získání výsledků pro porovnání.

Dojde zde pouze ke dvěma rozdílům. Prvním z nich je to, že už není nutno načítat knihovnu standardních buněk, takže řádky pseudokódu `skriptProABCOrig += read_library` a `skriptProABCR += read_library` se vypustí.

Druhým bude použití jiných příkazů při tvorbě skriptu pro systém ABC, takže tělo cyklu bude vypadat následovně:

```
skriptProABCR += choice/choicer;
skriptProABCR += if;
skriptProABCR += lutpack;
skriptProABCR += zapisVysledek[i];
```

Kapitola 5

Testování randomizačních postupů

5.1 Způsoby a nástroje pro testování

Testování bylo prováděno za použití dvou způsobů:

V rámci této práce implementovaná aplikace **abcScriptGen** umožňuje vytvářet skripty pro systému ABC a také provádění těchto skriptů rovnou spouštět. Pomocí aplikace je také možnost spouštět program permute pro provádění permutací nad obvody.

Na **výpočetním svazku MAGNUM** se prováděly experimenty čítající tisíce opakování. Jedná se o počítač s se dvěma čtyřjádrovými procesory, který je provozován pod operačním systémem Linux. Pro spuštění experimentů se používají skripty generované aplikaci abcScriptGen.

Pro testování bylo použito 450 různých obvodů, které jsou součástí sbírek ISCAS'85[11], ISCAS'89[12], IWLS'93[16], ITC'99[13] a OpenCores.

5.2 Výsledky testování jednotlivých metod randomizace

5.2.1 Příkazy BALANCE, REWRITE, REFACTOR, RESUB s použitím permutací

Měření je zpracováno pomocí tabulek, jejichž sloupce vyjadřují následující:

- oA - počet AND uzlů v obvodu při použití neupraveného příkazu
- oL - nejvyšší úroveň v obvodu při použití neupraveného příkazu
- minA - minimální počet AND uzlů dosažený ze všech opakování
- maxA - maximální počet AND uzlů dosažený ze všech opakování
- min% - procentuální poměr hodnot sloupců minA k oA
- max% - procentuální poměr hodnot sloupců maxA k oA
- d - rozdíl mezi minimálním a maximálním počtem AND uzlů ze všech opakování
- % - rozdíl v procentech mezi nejlepšími dosaženými a původními hodnotami
- minR% - relativní poměr - $(oA - minA) \div oA$
- maxR% - relativní poměr - $(oA - maxA) \div oA$
- prum+ - počet obvodů, které kladně reagují, v procentech ze všech opakování
- prum- - počet obvodů, které záporně reagují, v procentech ze všech opakování
- prumC - průměrná hodnota ze zlepšení a zhoršení ve výsledcích obvodu ze všech opakování v procentech

Pro každý obvod bylo prováděno 10000 opakování. Zelená barva značí výsledek lepší než výsledek dosažený bez randomizace, červená opak.

Tabulka 5.1: Příkaz BALANCE s permutacemi primárních vstupů - část 1.

Obvod	oA	oL	minA	maxA	min%	max%	d	%
Altera_mux32_16bit	2440	19	2209	2417	90,53	99,06	208	9,47
i5	329	8	319	332	96,96	100,91	13	3,04
vg2	526	9	514	538	97,72	102,28	24	2,28
e64	736	7	722	747	98,1	101,49	25	1,9
x6dn	476	19	470	480	98,74	100,84	10	1,26
duke2	504	9	498	508	98,81	100,79	10	1,19
m2	304	13	301	304	99,01	100	3	0,99
soar	612	14	606	615	99,02	100,49	9	0,98
c880	327	22	324	327	99,08	100	3	0,92
i7	557	6	552	569	99,1	102,15	17	0,9
b10	584	22	579	588	99,14	100,68	9	0,86
exep	603	18	598	603	99,17	100	5	0,83
x4	719	9	713	726	99,17	100,97	13	0,83
in3	489	20	485	498	99,18	101,84	13	0,82
m3	407	13	404	407	99,26	100	3	0,74
term1	696	17	691	704	99,28	101,15	13	0,72
b3	446	20	443	453	99,33	101,57	10	0,67
t481	1279	14	1271	1281	99,37	100,16	10	0,63
apex1	1488	12	1479	1496	99,4	100,54	17	0,6
cps	1553	10	1545	1558	99,48	100,32	13	0,52
apex5	3133	10	3117	3152	99,49	100,61	35	0,51
in0	648	22	645	652	99,54	100,62	7	0,46
Altera_oc_miniuart	897	16	893	902	99,55	100,56	9	0,45
mlp4	466	15	464	467	99,57	100,21	3	0,43
Altera_nut_004	779	25	776	779	99,61	100	3	0,39
Altera_oc_ata_ocidec1	2125	17	2117	2127	99,62	100,09	10	0,38
example2	314	9	313	315	99,68	100,32	2	0,32
apex3	1575	12	1570	1627	99,68	103,3	57	0,32
Altera_oc_hdlc	2940	19	2931	2953	99,69	100,44	22	0,31
s820	322	10	321	325	99,69	100,93	4	0,31
s832	327	10	326	333	99,69	101,83	7	0,31
s5378	1327	15	1323	1334	99,7	100,53	11	0,3
Altera_radar12	38790	158	38679	38787	99,71	99,99	108	0,29
ITC_b07	351	26	350	351	99,72	100	1	0,28
s1269	462	26	461	462	99,78	100	1	0,22
alu2	477	24	476	482	99,79	101,05	6	0,21
s1196	476	18	475	477	99,79	100,21	2	0,21
Altera_oc_gpio	963	17	961	963	99,79	100	2	0,21
idxlat00n_m12	3292	13	3286	3296	99,82	100,12	10	0,18
dsip	2520	12	2520	2521	100	100,04	1	0
Altera_oc_fcmp	705	101	705	705	100	100	0	0
Altera_oc_v_c_s_jpeg	63523	81	63521	63525	100	100	4	0

Tabulka 5.2: Příkaz BALANCE s permutacemi primárních vstupů - část 2.

Obvod	oA	oL	minR%	maxR%	prum+	prum-	prumC
Altera_mux32_16bit	2440	19	9,47	0,94	100	0	4,98
i5	329	8	3,04	0,91	86,15	5,5	1,05
vg2	526	9	2,28	2,28	39,72	51	0,11
e64	736	7	1,9	1,49	51,71	36,83	0,1
x6dn	476	19	1,26	0,84	50,89	29,16	0,12
duke2	504	9	1,19	0,79	37,89	41,74	0,01
m2	304	13	0,99	0	90,53	0	0,48
soar	612	14	0,98	0,49	72,8	9,29	0,23
c880	327	22	0,92	0	84,5	0	0,47
i7	557	6	0,9	2,15	28,14	66,36	0,63
b10	584	22	0,86	0,68	45,89	30,37	0,07
exep	603	18	0,83	0	91,69	0	0,4
x4	719	9	0,83	0,97	26,79	55,81	0,1
in3	489	20	0,82	1,84	22,27	59,44	0,24
m3	407	13	0,74	0	73,63	0	0,37
term1	696	17	0,72	1,15	22,85	62,92	0,19
b3	446	20	0,67	1,57	13,76	68,69	0,31
t481	1279	14	0,63	0,16	86,65	6,42	0,23
apex1	1488	12	0,6	0,54	54,81	31,13	0,05
cps	1553	10	0,52	0,32	58,97	25,15	0,06
apex5	3133	10	0,51	0,61	30,47	62,07	0,06
in0	648	22	0,46	0,62	21,85	48,99	0,06
Altera_oc_miniuart	897	16	0,45	0,56	34,45	44,38	0,03
mlp4	466	15	0,43	0,21	50,02	20,05	0,11
Altera_nut_004	779	25	0,39	0	86,57	0	0,19
Altera_oc_ata_ocidec1	2125	17	0,38	0,09	84,99	6,5	0,14
example2	314	9	0,32	0,32	24,49	25,53	0
apex3	1575	12	0,32	3,3	6,42	91,94	1,43
Altera_oc_hdlc	2940	19	0,31	0,44	18,93	71,99	0,08
s820	322	10	0,31	0,93	14,36	51,81	0,18
s832	327	10	0,31	1,83	7,1	80,19	0,62
s5378	1327	15	0,3	0,53	24	63,45	0,11
Altera_radar12	38790	158	0,29	0,01	100	0	0,15
ITC_b07	351	26	0,28	0	48,87	0	0,14
s1269	462	26	0,22	0	49,95	0	0,11
alu2	477	24	0,21	1,05	7,46	73,53	0,39
s1196	476	18	0,21	0,21	32,69	33,79	0
Altera_oc_gpio	963	17	0,21	0	49,36	0	0,1
idxlat00n_m12	3292	13	0,18	0,12	58,14	24,83	0,03
dsip	2520	12	0	0,04	0	50,25	0,02
Altera_oc_fcmp	705	101	0	0	0	0	0
Altera_oc_v_c_s_jpeg	63523	81	0	0	27,69	29,75	0

Tabulka 5.3: Příkaz BALANCE s permutacemi primárních výstupů - část 1.

Obvod	oA	oL	minA	maxA	min%	max%	d	%
exep	603	18	582	616	96,52	102,16	34	3,48
s820	322	10	312	324	96,89	100,62	12	3,11
s832	327	10	319	330	97,55	100,92	11	2,45
e64	736	7	720	749	97,83	101,77	29	2,17
b3	446	20	437	455	97,98	102,02	18	2,02
i5	329	8	323	333	98,18	101,22	10	1,82
apex1	1488	12	1461	1505	98,19	101,14	44	1,81
duke2	504	9	495	513	98,21	101,79	18	1,79
apex3	1575	12	1548	1599	98,29	101,52	51	1,71
c1908	415	27	408	416	98,31	100,24	8	1,69
in3	489	20	482	501	98,57	102,45	19	1,43
newxcpla1	141	11	139	146	98,58	103,55	7	1,42
vg2	526	9	519	528	98,67	100,38	9	1,33
idxlat10n_m12	3038	13	3004	3054	98,88	100,53	50	1,12
idxlat00n_m12	3292	13	3259	3325	99	101	66	1
m3	407	13	403	409	99,02	100,49	6	0,98
soar	612	14	606	622	99,02	101,63	16	0,98
c880	327	22	324	327	99,08	100	3	0,92
Altera_barrel16	663	19	657	665	99,1	100,3	8	0,9
alu4	3401	13	3373	3425	99,18	100,71	52	0,82
x4	719	9	714	728	99,3	101,25	14	0,7
s1512	437	17	434	437	99,31	100	3	0,69
m2	304	13	302	305	99,34	100,33	3	0,66
example2	314	9	312	316	99,36	100,64	4	0,64
x6dn	476	19	473	478	99,37	100,42	5	0,63
s5378	1327	15	1319	1340	99,4	100,98	21	0,6
opa	680	11	676	688	99,41	101,18	12	0,59
cps	1553	10	1544	1566	99,42	100,84	22	0,58
apex5	3133	10	3116	3152	99,46	100,61	36	0,54
x3	1266	14	1260	1277	99,53	100,87	17	0,47
term1	696	17	693	698	99,57	100,29	5	0,43
Altera_usb_phy	461	9	459	463	99,57	100,43	4	0,43
alu2	477	24	475	478	99,58	100,21	3	0,42
s6669	2208	80	2199	2215	99,59	100,32	16	0,41
ITC_b13	255	12	254	256	99,61	100,39	2	0,39
ex5	774	10	771	785	99,61	101,42	14	0,39
clip	523	9	521	523	99,62	100	2	0,38
ttt2	542	10	540	546	99,63	100,74	6	0,37
i7	557	6	555	561	99,64	100,72	6	0,36
9sym	322	10	322	322	100	100	0	0
Altera_mem	16726	33	16726	16726	100	100	0	0
ITC_b21	12676	65	12681	12699	100,04	100,18	18	0,04

Tabulka 5.4: Příkaz BALANCE s permutacemi primárních výstupů - část 2.

Obvod	oA	oL	minR%	maxR%	prum+	prum-	prumC
exep	603	18	3,48	2,16	85,58	10,44	0,87
s820	322	10	3,11	0,62	99,32	0,11	1,67
s832	327	10	2,45	0,92	94,89	1,66	1,07
e64	736	7	2,17	1,77	60,45	30,32	0,21
b3	446	20	2,02	2,02	60,65	26,6	0,26
i5	329	8	1,82	1,22	73,63	11,56	0,51
apex1	1488	12	1,81	1,14	68,05	26,5	0,24
duke2	504	9	1,79	1,79	31,05	54,23	0,14
apex3	1575	12	1,71	1,52	34,29	59,29	0,11
c1908	415	27	1,69	0,24	34,13	0,23	0,22
in3	489	20	1,43	2,45	15,86	74,03	0,48
newxcpla1	141	11	1,42	3,55	33,5	32,52	0,05
vg2	526	9	1,33	0,38	66,57	13,89	0,32
idxlat10n_m12	3038	13	1,12	0,53	88,66	8,41	0,27
idxlat00n_m12	3292	13	1	1	26,62	69,31	0,14
m3	407	13	0,98	0,49	91,52	1,13	0,57
soar	612	14	0,98	1,63	19,59	66,28	0,25
c880	327	22	0,92	0	98,51	0	0,78
Altera_barrel16	663	19	0,9	0,3	72,74	6,61	0,21
alu4	3401	13	0,82	0,71	50,83	45,35	0,02
x4	719	9	0,7	1,25	8,88	80,72	0,37
s1512	437	17	0,69	0	98,14	0	0,4
m2	304	13	0,66	0,33	66,85	5,62	0,27
example2	314	9	0,64	0,64	74,59	9,02	0,33
x6dn	476	19	0,63	0,42	57,54	29,95	0,15
s5378	1327	15	0,6	0,98	7,99	86,64	0,3
opa	680	11	0,59	1,18	12,13	70,43	0,2
cps	1553	10	0,58	0,84	21,49	67,66	0,12
apex5	3133	10	0,54	0,61	73,4	21,71	0,12
x3	1266	14	0,47	0,87	32,38	50,87	0,06
term1	696	17	0,43	0,29	58,53	17,93	0,13
Altera_usb_phy	461	9	0,43	0,43	11,66	41,85	0,08
alu2	477	24	0,42	0,21	27,74	35,72	0,04
s6669	2208	80	0,41	0,32	50,55	42,88	0,03
ITC_b13	255	12	0,39	0,39	25,94	24,1	0,01
ex5	774	10	0,39	1,42	28,14	49,2	0,08
clip	523	9	0,38	0	54,85	0	0,15
ttt2	542	10	0,37	0,74	9,44	66,22	0,19
i7	557	6	0,36	0,72	86,83	1,22	0,29
9sym	322	10	0	0	0	0	0
Altera_mem	16726	33	0	0	0	0	0
ITC_b21	12676	65	0,04	0,18	0	100	0,11

Tabulka 5.5: Příkaz REWRITE s permutacemi primárních vstupů - část 1.

Obvod	oA	oL	minA	maxA	min%	max%	d	%
opa	654	12	641	655	98,01	100,15	14	1,99
term1	344	16	338	358	98,26	104,07	20	1,74
Altera_oc_miniuart	592	13	585	597	98,82	100,84	12	1,18
m2	267	13	264	269	98,88	100,75	5	1,12
Altera_nut_004	517	21	512	517	99,03	100	5	0,97
soar	590	16	585	591	99,15	100,17	6	0,85
b3	391	23	388	391	99,23	100	3	0,77
ex5	637	11	633	647	99,37	101,57	14	0,63
rd73	350	12	348	351	99,43	100,29	3	0,57
m3	365	14	363	367	99,45	100,55	4	0,55
in3	421	20	419	422	99,52	100,24	3	0,48
mlp4	423	15	421	424	99,53	100,24	3	0,47
Altera_oc_ata_ocidec1	1556	14	1549	1559	99,55	100,19	10	0,45
vg2	514	11	512	514	99,61	100	2	0,39
b10	512	22	510	513	99,61	100,2	3	0,39
apex5	1790	12	1783	1823	99,61	101,84	40	0,39
pope	561	11	559	570	99,64	101,6	11	0,36
alu4	3203	16	3192	3215	99,66	100,37	23	0,34
cps	1394	12	1390	1396	99,71	100,14	6	0,29
ttt2	344	13	343	345	99,71	100,29	2	0,29
s953	341	13	340	341	99,71	100	1	0,29
Altera_radar12	26707	87	26644	26711	99,76	100,01	67	0,24
prom2	2948	21	2941	2964	99,76	100,54	23	0,24
clip	440	10	439	442	99,77	100,45	3	0,23
s15850	3061	40	3054	3065	99,77	100,13	11	0,23
s1196	466	19	465	466	99,79	100	1	0,21
s1238	513	22	512	513	99,81	100	1	0,19
Altera_oc_hdlc	2082	15	2078	2104	99,81	101,06	26	0,19
in0	569	22	568	570	99,82	100,18	2	0,18
dalu	1188	35	1186	1188	99,83	100	2	0,17
Altera_sasc	637	8	636	637	99,84	100	1	0,16
exep	653	19	652	661	99,85	101,23	9	0,15
pair	1379	23	1377	1381	99,85	100,15	4	0,15
apex3	1544	13	1542	1544	99,87	100	2	0,13
s5378	1221	17	1220	1221	99,92	100	1	0,08
idxlat10n_m12	2879	18	2877	2881	99,93	100,07	4	0,07
Altera_oc_v_c_s_dct	33954	65	33952	33959	99,99	100,01	7	0,01
Mentor_1_02	398	63	398	398	100	100	0	0
Mentor_1_03	397	63	397	397	100	100	0	0
idxlat00n_m12	3420	20	3420	3420	100	100	0	0
ITC_b22_1	13471	71	13471	13473	100	100,01	2	0
ITC_b13	239	13	239	239	100	100	0	0

Tabulka 5.6: Příkaz REWRITE s permutacemi primárních vstupů - část 2.

Obvod	oA	oL	minR%	maxR%	prum+	prum-	prumC
opa	654	12	1,99	0,15	84,28	2,09	0,67
term1	344	16	1,74	4,07	65,04	26,33	0,38
Altera_oc_miniuart	592	13	1,18	0,84	43,45	38,96	0,03
m2	267	13	1,12	0,75	59,92	14,74	0,32
Altera_nut_004	517	21	0,97	0	69,77	0	0,32
soar	590	16	0,85	0,17	76,87	7,14	0,32
b3	391	23	0,77	0	52,5	0	0,15
ex5	637	11	0,63	1,57	6,98	88,64	0,69
rd73	350	12	0,57	0,29	50,44	15,73	0,24
m3	365	14	0,55	0,55	51,8	14,43	0,17
in3	421	20	0,48	0,24	50,84	17,35	0,12
mlp4	423	15	0,47	0,24	24,67	17,4	0,02
Altera_oc_ata_ocidec1	1556	14	0,45	0,19	78,78	10,15	0,14
vg2	514	11	0,39	0	47,93	0	0,19
b10	512	22	0,39	0,2	57,32	8,34	0,13
apex5	1790	12	0,39	1,84	9,37	88,44	0,75
pope	561	11	0,36	1,6	2,5	95,3	0,65
alu4	3203	16	0,34	0,37	33,86	58,15	0,04
cps	1394	12	0,29	0,14	58,49	19,78	0,07
ttt2	344	13	0,29	0,29	25,71	13,49	0,04
s953	341	13	0,29	0	50,29	0	0,15
Altera_radar12	26707	87	0,24	0,01	99,95	0,05	0,17
prom2	2948	21	0,24	0,54	6,66	88,92	0,17
clip	440	10	0,23	0,45	9,74	52,41	0,12
s15850	3061	40	0,23	0,13	53,19	29,22	0,02
s1196	466	19	0,21	0	49,39	0	0,1
s1238	513	22	0,19	0	51,17	0	0,1
Altera_oc_hdlc	2082	15	0,19	1,06	2,75	92,86	0,22
in0	569	22	0,18	0,18	24,27	25,72	0
dalu	1188	35	0,17	0	67,2	0	0,08
Altera_sasc	637	8	0,16	0	32,59	0	0,05
exep	653	19	0,15	1,23	28,45	42,26	0,23
pair	1379	23	0,15	0,15	25,75	24,1	0
apex3	1544	13	0,13	0	47,86	0	0,06
s5378	1221	17	0,08	0	24,58	0	0,02
idxlat10n_m12	2879	18	0,07	0,07	32,77	34,2	0
Altera_oc_v_c_s_dct	33954	65	0,01	0,01	36,34	51,73	0
Mentor_1_02	398	63	0	0	0	0	0
Mentor_1_03	397	63	0	0	0	0	0
idxlat00n_m12	3420	20	0	0	0	0	0
ITC_b22_1	13471	71	0	0,01	0	49,89	0
ITC_b13	239	13	0	0	0	0	0

Tabulka 5.7: Příkaz REWRITE s permutacemi primárních výstupů - část 1.

Obvod	oA	oL	minA	maxA	min%	max%	d	%
i5	223	12	202	238	90,58	106,73	36	9,42
ex5	637	11	605	646	94,98	101,41	41	5,02
ttt2	344	13	327	351	95,06	102,03	24	4,94
m2	267	13	255	269	95,51	100,75	14	4,49
Altera_radar12	26707	87	25547	25769	95,66	96,49	222	4,34
mult32a	344	96	330	346	95,93	100,58	16	4,07
Altera_usb_phy	400	9	386	402	96,5	100,5	16	3,5
in3	421	20	408	429	96,91	101,9	21	3,09
g25	247	15	240	249	97,17	100,81	9	2,83
b10	512	22	498	526	97,27	102,73	28	2,73
Altera_oc_miniuart	592	13	576	613	97,3	103,55	37	2,7
example2	276	9	269	276	97,46	100	7	2,54
m3	365	14	356	367	97,53	100,55	11	2,47
Altera_oc_v_c_s_dct	33954	65	33147	33531	97,62	98,75	384	2,38
mlp4	423	15	413	427	97,64	100,95	14	2,36
newxcpla1	129	10	126	137	97,67	106,2	11	2,33
b3	391	23	382	396	97,7	101,28	14	2,3
x3	744	16	727	759	97,72	102,02	32	2,28
x4	514	10	503	520	97,86	101,17	17	2,14
opa	654	12	641	663	98,01	101,38	22	1,99
Altera_pci_spoci_ctrl	1109	17	1088	1150	98,11	103,7	62	1,89
soar	590	16	579	599	98,14	101,53	20	1,86
s832	288	14	283	295	98,26	102,43	12	1,74
Altera_barrel16	416	14	409	420	98,32	100,96	11	1,68
duke2	493	11	485	498	98,38	101,01	13	1,62
exep	653	19	643	680	98,47	104,13	37	1,53
s820	281	14	277	288	98,58	102,49	11	1,42
in0	569	22	561	584	98,59	102,64	23	1,41
dalu	1188	35	1173	1196	98,74	100,67	23	1,26
cps	1394	12	1378	1407	98,85	100,93	29	1,15
clip	440	10	435	442	98,86	100,45	7	1,14
idxlat10n_m12	2879	18	2847	2911	98,89	101,11	64	1,11
Altera_sasc	637	8	630	639	98,9	100,31	9	1,1
apex1	1477	14	1462	1486	98,98	100,61	24	1,02
s9234,1	1583	29	1567	1586	98,99	100,19	19	1,01
e64	690	8	683	699	98,99	101,3	16	1,01
pope	561	11	556	577	99,11	102,85	21	0,89
mm9b	363	72	360	363	99,17	100	3	0,83
apex5	1790	12	1776	1813	99,22	101,28	37	0,78
s953	341	13	341	341	100	100	0	0
Altera_mux32_16bit	1490	12	1490	1490	100	100	0	0
Altera_oc_v_c_s_jpeg	41229	63	41273	41656	100,11	101,04	383	0,11

Tabulka 5.8: Příkaz REWRITE s permutacemi primárních výstupů - část 2.

Obvod	oA	oL	minR%	maxR%	prum+	prum-	prumC
i5	223	12	9,42	6,73	75,06	19,44	1,74
ex5	637	11	5,02	1,41	94,16	3,99	1,54
ttt2	344	13	4,94	2,03	81,58	11,32	1,14
m2	267	13	4,49	0,75	93,08	1,91	1,34
Altera_radar12	26707	87	4,34	3,51	100	0	3,79
mult32a	344	96	4,07	0,58	70,12	7,17	0,71
Altera_usb_phy	400	9	3,5	0,5	96,35	0,82	1,29
in3	421	20	3,09	1,9	69,86	17,79	0,45
g25	247	15	2,83	0,81	71,02	10,34	0,73
b10	512	22	2,73	2,73	42,89	47,69	0,06
Altera_oc_miniuart	592	13	2,7	3,55	24,42	70,81	0,61
example2	276	9	2,54	0	92,05	0	1,09
m3	365	14	2,47	0,55	90,33	2,47	0,8
Altera_oc_v_c_s_dct	33954	65	2,38	1,25	100	0	1,9
mlp4	423	15	2,36	0,95	75,28	12,38	0,61
newxcpla1	129	10	2,33	6,2	11,24	78,56	1,54
b3	391	23	2,3	1,28	85,94	5,51	0,72
x3	744	16	2,28	2,02	26,04	68,15	0,34
x4	514	10	2,14	1,17	32,15	63,69	0,17
opa	654	12	1,99	1,38	34,4	51,25	0,03
Altera_pci_spoci_ctrl	1109	17	1,89	3,7	70,54	28,55	0,11
soar	590	16	1,86	1,53	78,57	12,87	0,45
s832	288	14	1,74	2,43	45,04	34,71	0,07
Altera_barrel16	416	14	1,68	0,96	52,97	23,28	0,19
duke2	493	11	1,62	1,01	42,47	31,32	0,09
exep	653	19	1,53	4,13	13,85	80,05	0,66
s820	281	14	1,42	2,49	31,5	46,97	0,15
in0	569	22	1,41	2,64	26,27	65,72	0,45
dalu	1188	35	1,26	0,67	61,03	29,29	0,17
cps	1394	12	1,15	0,93	35,35	56,3	0,07
clip	440	10	1,14	0,45	69,68	21,24	0,37
idxlat10n_m12	2879	18	1,11	1,11	47,63	47,7	0
Altera_sasc	637	8	1,1	0,31	89,83	2,71	0,39
apex1	1477	14	1,02	0,61	70,2	22,15	0,17
s9234,1	1583	29	1,01	0,19	90,27	2,69	0,25
e64	690	8	1,01	1,3	18,15	68,36	0,23
pope	561	11	0,89	2,85	1,36	96,95	1,13
mm9b	363	72	0,83	0	37,7	0	0,17
apex5	1790	12	0,78	1,28	19,41	75,23	0,24
s953	341	13	0	0	0	0	0
Altera_mux32_16bit	1490	12	0	0	0	0	0
Altera_oc_v_c_s_jpeg	41229	63	0,11	1,04	0	100	0,5

Tabulka 5.9: Příkaz REFACTOR s permutacemi primárních vstupů - část 1

Obvod	oA	oL	minA	maxA	min%	max%	d	%
x4	466	9	459	467	98,5	100,21	8	1,5
m2	270	13	268	270	99,26	100	2	0,74
t481	817	16	811	817	99,27	100	6	0,73
in0	546	20	543	548	99,45	100,37	5	0,55
b10	488	20	486	490	99,59	100,41	4	0,41
dalu	1226	34	1221	1226	99,59	100	5	0,41
s820	314	14	313	314	99,68	100	1	0,32
clip	419	10	418	420	99,76	100,24	2	0,24
x6dn	437	20	436	437	99,77	100	1	0,23
s1269	453	29	452	453	99,78	100	1	0,22
s3271	1064	21	1062	1064	99,81	100	2	0,19
Altera_oc_hdlc	2110	16	2106	2114	99,81	100,19	8	0,19
cordic	1164	18	1162	1164	99,83	100	2	0,17
s5378	1210	17	1208	1210	99,83	100	2	0,17
s15850	3100	40	3095	3100	99,84	100	5	0,16
soar	656	18	655	658	99,85	100,3	3	0,15
apex1	1572	14	1570	1574	99,87	100,13	4	0,13
alu4	3036	16	3033	3038	99,9	100,07	5	0,1
cps	1472	12	1471	1472	99,93	100	1	0,07
s9234,1	1585	28	1584	1590	99,94	100,32	6	0,06
Altera_mem	13849	31	13848	13849	99,99	100	1	0,01
Altera_radar12	28121	121	28119	28121	99,99	100	2	0,01
Altera_barrel16a	438	14	438	438	100	100	0	0
Altera_oc_ata_ocidec1	1569	14	1569	1569	100	100	0	0
c1908	397	28	397	397	100	100	0	0
mm9b	429	71	429	429	100	100	0	0
idxlat00n_m12	3793	20	3793	3793	100	100	0	0
mm9a	367	53	367	367	100	100	0	0
Altera_ss_pcm	400	7	400	400	100	100	0	0
ITC_b22_1	13766	71	13766	13766	100	100	0	0
Altera_oc_v_c_s_dct	36745	67	36745	36745	100	100	0	0
rd73	322	12	322	322	100	100	0	0
Altera_xbar_16x16	736	10	736	736	100	100	0	0
s953	345	13	345	345	100	100	0	0
in3	467	20	467	469	100	100,43	2	0
Altera_oc_miniuart	620	13	620	620	100	100	0	0
pope	629	12	629	629	100	100	0	0
b3	433	20	433	434	100	100,23	1	0
s1196	473	19	473	473	100	100	0	0
opa	712	14	712	713	100	100,14	1	0
alu2	469	32	469	469	100	100	0	0
prom2	3235	22	3235	3235	100	100	0	0

Tabulka 5.10: Příkaz REFACTOR s permutacemi primárních vstupů - část 2

Obvod	oA	oL	minR%	maxR%	prum+	prum-	prumC
x4	466	9	1,5	0,21	68,72	12,09	0,53
m2	270	13	0,74	0	49,85	0	0,37
t481	817	16	0,73	0	49,34	0	0,36
in0	546	20	0,55	0,37	51,99	18,65	0,1
b10	488	20	0,41	0,41	35,27	31,77	0,01
dalu	1226	34	0,41	0	50,43	0	0,21
s820	314	14	0,32	0	49,65	0	0,16
clip	419	10	0,24	0,24	35,35	48,92	0,03
x6dn	437	20	0,23	0	47,73	0	0,11
s1269	453	29	0,22	0	48,34	0	0,11
s3271	1064	21	0,19	0	50,48	0	0,07
Altera_oc_hdlc	2110	16	0,19	0,19	38,79	45,73	0,01
cordic	1164	18	0,17	0	48,54	0	0,08
s5378	1210	17	0,17	0	66,35	0	0,08
s15850	3100	40	0,16	0	94,16	0	0,08
soar	656	18	0,15	0,3	24,81	50,57	0,08
apex1	1572	14	0,13	0,13	38,4	36,32	0
alu4	3036	16	0,1	0,07	44,89	27,1	0,01
cps	1472	12	0,07	0	49,35	0	0,03
s9234,1	1585	28	0,06	0,32	9,4	81,15	0,14
Altera_mem	13849	31	0,01	0	21,44	0	0
Altera_radar12	28121	121	0,01	0	74,86	0	0
Altera_barrel16a	438	14	0	0	0	0	0
Altera_oc_ata_ocidec1	1569	14	0	0	0	0	0
c1908	397	28	0	0	0	0	0
mm9b	429	71	0	0	0	0	0
idxlat00n_m12	3793	20	0	0	0	0	0
mm9a	367	53	0	0	0	0	0
Altera_ss_pcm	400	7	0	0	0	0	0
ITC_b22_1	13766	71	0	0	0	0	0
Altera_oc_v_c_s_dct	36745	67	0	0	0	0	0
rd73	322	12	0	0	0	0	0
Altera_xbar_16x16	736	10	0	0	0	0	0
s953	345	13	0	0	0	0	0
in3	467	20	0	0,43	0	73,36	0,21
Altera_oc_miniuart	620	13	0	0	0	0	0
pope	629	12	0	0	0	0	0
b3	433	20	0	0,23	0	50,38	0,12
s1196	473	19	0	0	0	0	0
opa	712	14	0	0,14	0	48,62	0,07
alu2	469	32	0	0	0	0	0
prom2	3235	22	0	0	0	0	0

Tabulka 5.11: Příkaz REFACTOR s permutacemi primárních výstupů - část 1.

Obvod	oA	oL	minA	maxA	min%	max%	d	%
m2	270	13	250	279	92,59	103,33	29	7,41
ex5	669	13	622	689	92,97	102,99	67	7,03
ttt2	233	9	218	249	93,56	106,87	31	6,44
Altera_barrel16	461	17	433	472	93,93	102,39	39	6,07
newxcpla1	124	8	117	126	94,35	101,61	9	5,65
apex5	1540	12	1455	1593	94,48	103,44	138	5,52
Mentor_1_03	405	43	387	405	95,56	100	18	4,44
term1	296	15	283	309	95,61	104,39	26	4,39
Mentor_1_02	409	43	392	409	95,84	100	17	4,16
Altera_barrel16a	438	14	421	457	96,12	104,34	36	3,88
mult32a	430	96	414	434	96,28	100,93	20	3,72
pope	629	12	607	636	96,5	101,11	29	3,5
b3	433	20	419	454	96,77	104,85	35	3,23
m3	353	14	342	368	96,88	104,25	26	3,12
x4	466	9	452	480	97	103	28	3
exep	620	21	605	678	97,58	109,35	73	2,42
clip	419	10	409	424	97,61	101,19	15	2,39
mm9b	429	71	419	441	97,67	102,8	22	2,33
mm9a	367	53	359	382	97,82	104,09	23	2,18
x3	790	14	773	798	97,85	101,01	25	2,15
Altera_oc_miniuart	620	13	607	631	97,9	101,77	24	2,1
Altera_pci_spoci_ctrl	1074	17	1052	1090	97,95	101,49	38	2,05
in0	546	20	535	553	97,99	101,28	18	2,01
Altera_oc_v_c_s_dct	36745	67	36019	36304	98,02	98,8	285	1,98
in3	467	20	459	476	98,29	101,93	17	1,71
vg2	517	11	509	526	98,45	101,74	17	1,55
Altera_sasc	666	8	657	669	98,65	100,45	12	1,35
Altera_radar12	28121	121	27754	27854	98,69	99,05	100	1,31
Altera_usb_phy	403	10	398	419	98,76	103,97	21	1,24
b10	488	20	482	495	98,77	101,43	13	1,23
cps	1472	12	1455	1475	98,85	100,2	20	1,15
duke2	527	11	521	527	98,86	100	6	1,14
soar	656	18	649	657	98,93	100,15	8	1,07
example2	288	10	285	288	98,96	100	3	1,04
opa	712	14	705	722	99,02	101,4	17	0,98
i5	207	11	205	212	99,03	102,42	7	0,97
Altera_oc_ata_ocidec1	1569	14	1554	1579	99,04	100,64	25	0,96
x6dn	437	20	433	444	99,08	101,6	11	0,92
Altera_oc_hdlc	2110	16	2091	2133	99,1	101,09	42	0,9
s1512	421	19	421	421	100	100	0	0
Altera_oc_v_c_s_jpeg	46372	63	46520	46770	100,32	100,86	250	0,32
c1908	397	28	400	411	100,76	103,53	11	0,76

Tabulka 5.12: Příkaz REFACTOR s permutacemi primárních výstupů - část 2.

Obvod	oA	oL	minR%	maxR%	prum+	prum-	prumC
m2	270	13	7,41	3,33	48,74	43,9	0,31
ex5	669	13	7,03	2,99	77,24	19,4	1,17
ttt2	233	9	6,44	6,87	81,37	15,32	1,95
Altera_barrel16	461	17	6,07	2,39	75,84	18,28	1,03
newxcpla1	124	8	5,65	1,61	74,94	2,78	1,49
apex5	1540	12	5,52	3,44	59,17	38,31	0,39
Mentor_1_03	405	43	4,44	0	99,99	0	2,13
term1	296	15	4,39	4,39	35,81	61,11	0,64
Mentor_1_02	409	43	4,16	0	99,99	0	2,11
Altera_barrel16a	438	14	3,88	4,34	35,41	57,67	0,37
mult32a	430	96	3,72	0,93	61,51	15,14	0,51
pope	629	12	3,5	1,11	85,69	9,02	0,94
b3	433	20	3,23	4,85	33,66	62,05	0,74
m3	353	14	3,12	4,25	54,4	38,17	0,08
x4	466	9	3	3	12,47	80,68	0,67
exep	620	21	2,42	9,35	2,21	96,84	2,19
clip	419	10	2,39	1,19	60,19	18,16	0,36
mm9b	429	71	2,33	2,8	38,48	53,13	0,09
mm9a	367	53	2,18	4,09	32,46	60,57	0,53
x3	790	14	2,15	1,01	93,95	3,75	0,87
Altera_oc_miniuart	620	13	2,1	1,77	27,39	60,89	0,19
Altera_pci_spoci_ctrl	1074	17	2,05	1,49	64,6	29,96	0,26
in0	546	20	2,01	1,28	73,59	16,49	0,44
Altera_oc_v_c_s_dct	36745	67	1,98	1,2	100	0	1,61
in3	467	20	1,71	1,93	21,97	63,16	0,26
vg2	517	11	1,55	1,74	52,76	28,62	0,19
Altera_sasc	666	8	1,35	0,45	97,22	0,69	0,61
Altera_radar12	28121	121	1,31	0,95	100	0	1,15
Altera_usb_phy	403	10	1,24	3,97	48,42	43,4	0,46
b10	488	20	1,23	1,43	39,46	39,65	0,01
cps	1472	12	1,15	0,2	98,1	0,57	0,42
duke2	527	11	1,14	0	97,57	0	0,55
soar	656	18	1,07	0,15	78,3	2,52	0,25
example2	288	10	1,04	0	88,9	0	0,6
opa	712	14	0,98	1,4	9,05	84,46	0,39
i5	207	11	0,97	2,42	11,86	46,82	1
Altera_oc_ata_ocidec1	1569	14	0,96	0,64	96,04	2,13	0,38
x6dn	437	20	0,92	1,6	29,77	53,47	0,24
Altera_oc_hdlc	2110	16	0,9	1,09	28,09	65,93	0,16
s1512	421	19	0	0	0	0	0
Altera_oc_v_c_s_jpeg	46372	63	0,32	0,86	0	100	0,57
c1908	397	28	0,76	3,53	0	100	2,91

Tabulka 5.13: Příkaz RESUB s permutacemi primárních vstupů - část 1.

Obvod	oA	oL	minA	maxA	min%	max%	d	%
i5	284	11	280	284	98,59	100	4	1,41
ex5	698	12	694	699	99,43	100,14	5	0,57
alu4	3216	16	3210	3221	99,81	100,16	11	0,19
soar	624	19	623	624	99,84	100	1	0,16
intb	1316	27	1314	1316	99,85	100	2	0,15
cps	1442	12	1440	1442	99,86	100	2	0,14
apex1	1490	14	1489	1490	99,93	100	1	0,07
Altera_barrel16a	623	18	623	623	100	100	0	0
s820	311	14	311	311	100	100	0	0
pair	1645	24	1645	1645	100	100	0	0
s1196	458	19	458	458	100	100	0	0
x3	1211	20	1211	1211	100	100	0	0
x4	557	9	557	557	100	100	0	0
dsip	2518	14	2518	2518	100	100	0	0
s838	272	41	272	272	100	100	0	0
prom2	3403	22	3403	3403	100	100	0	0
s832	319	14	319	319	100	100	0	0
Altera_oc_v_c_s_jpeg	49548	72	49548	49548	100	100	0	0
exep	761	23	761	761	100	100	0	0
dalu	1416	36	1416	1416	100	100	0	0
s9234,1	1818	32	1818	1818	100	100	0	0
m2	274	14	274	274	100	100	0	0
m3	376	14	376	376	100	100	0	0
s1423	458	54	458	458	100	100	0	0
example2	292	10	292	292	100	100	0	0
s15850	3389	43	3389	3389	100	100	0	0
Altera_oc_hdlc	2456	19	2456	2456	100	100	0	0
Altera_ss_pcm	405	7	405	405	100	100	0	0
apex3	1604	13	1604	1605	100	100,06	1	0
apex5	3146	12	3146	3146	100	100	0	0
alu2	473	32	473	473	100	100	0	0
Altera_oc_gpio	866	20	866	866	100	100	0	0
mlp4	436	15	436	436	100	100	0	0
s5378	1251	17	1251	1251	100	100	0	0
Altera_pci_spoci_ctrl	1194	17	1194	1194	100	100	0	0
mult32a	502	96	502	502	100	100	0	0
mult32b	366	4	366	366	100	100	0	0
i7	535	6	535	535	100	100	0	0
i6	388	5	388	388	100	100	0	0
Altera_xbar_16x16	1168	18	1168	1168	100	100	0	0
duke2	496	11	496	496	100	100	0	0
newxcpla1	121	11	121	121	100	100	0	0

Tabulka 5.14: Příkaz RESUB s permutacemi primárních vstupů - část 2.

Obvod	oA	oL	minR%	maxR%	prum+	prum-	prumC
i5	284	11	1,41	0	93,49	0	0,7
ex5	698	12	0,57	0,14	77,43	9,49	0,24
alu4	3216	16	0,19	0,16	34,46	49,6	0,01
soar	624	19	0,16	0	51,35	0	0,08
intb	1316	27	0,15	0	73,87	0	0,08
cps	1442	12	0,14	0	51,68	0	0,04
apex1	1490	14	0,07	0	50,24	0	0,04
Altera_barrel16a	623	18	0	0	0	0	0
s820	311	14	0	0	0	0	0
pair	1645	24	0	0	0	0	0
s1196	458	19	0	0	0	0	0
x3	1211	20	0	0	0	0	0
x4	557	9	0	0	0	0	0
dsip	2518	14	0	0	0	0	0
s838	272	41	0	0	0	0	0
prom2	3403	22	0	0	0	0	0
s832	319	14	0	0	0	0	0
Altera_oc_v_c_s_jpeg	49548	72	0	0	0	0	0
exep	761	23	0	0	0	0	0
dalu	1416	36	0	0	0	0	0
s9234,1	1818	32	0	0	0	0	0
m2	274	14	0	0	0	0	0
m3	376	14	0	0	0	0	0
s1423	458	54	0	0	0	0	0
example2	292	10	0	0	0	0	0
s15850	3389	43	0	0	0	0	0
Altera_oc_hdlc	2456	19	0	0	0	0	0
Altera_ss_pcm	405	7	0	0	0	0	0
apex3	1604	13	0	0,06	0	48	0,03
apex5	3146	12	0	0	0	0	0
alu2	473	32	0	0	0	0	0
Altera_oc_gpio	866	20	0	0	0	0	0
mlp4	436	15	0	0	0	0	0
s5378	1251	17	0	0	0	0	0
Altera_pci_spoci_ctrl	1194	17	0	0	0	0	0
mult32a	502	96	0	0	0	0	0
mult32b	366	4	0	0	0	0	0
i7	535	6	0	0	0	0	0
i6	388	5	0	0	0	0	0
Altera_xbar_16x16	1168	18	0	0	0	0	0
duke2	496	11	0	0	0	0	0
newxcpla1	121	11	0	0	0	0	0

Tabulka 5.15: Příkaz RESUB s permutacemi primárních výstupů - část 1.

Obvod	oA	oL	minA	maxA	min%	max%	d	%
ttt2	408	13	384	416	94,12	101,96	32	5,88
opa	693	18	670	700	96,68	101,01	30	3,32
Altera_oc_miniuart	697	15	675	707	96,84	101,43	32	3,16
m3	376	14	366	379	97,34	100,8	13	2,66
ex5	698	12	680	719	97,42	103,01	39	2,58
m2	274	14	267	280	97,45	102,19	13	2,55
newxcpla1	121	11	118	134	97,52	110,74	16	2,48
pope	569	12	555	582	97,54	102,28	27	2,46
term1	511	22	499	533	97,65	104,31	34	2,35
i5	284	11	278	288	97,89	101,41	10	2,11
b3	441	25	432	460	97,96	104,31	28	2,04
Altera_oc_v_c_s_dct	39355	84	38579	39023	98,03	99,16	444	1,97
Altera_nut_004	574	18	563	577	98,08	100,52	14	1,92
in0	618	25	608	631	98,38	102,1	23	1,62
Altera_barrel16	559	19	550	565	98,39	101,07	15	1,61
vg2	499	11	491	503	98,4	100,8	12	1,6
s832	319	14	314	319	98,43	100	5	1,57
e64	747	8	737	749	98,66	100,27	12	1,34
s820	311	14	307	311	98,71	100	4	1,29
soar	624	19	617	641	98,88	102,72	24	1,12
Altera_barrel16a	623	18	616	625	98,88	100,32	9	1,12
b10	552	25	546	568	98,91	102,9	22	1,09
exep	761	23	753	763	98,95	100,26	10	1,05
dalu	1416	36	1402	1431	99,01	101,06	29	0,99
mlp4	436	15	432	441	99,08	101,15	9	0,92
cps	1442	12	1430	1453	99,17	100,76	23	0,83
x3	1211	20	1201	1212	99,17	100,08	11	0,83
in3	512	25	508	519	99,22	101,37	11	0,78
x4	557	9	553	567	99,28	101,8	14	0,72
apex3	1604	13	1595	1611	99,44	100,44	16	0,56
s3384	1064	54	1058	1064	99,44	100	6	0,56
idxlat10n_m12	2881	18	2866	2902	99,48	100,73	36	0,52
c7552	1880	29	1871	1897	99,52	100,9	26	0,48
idxlat00n_m12	3435	20	3419	3455	99,53	100,58	36	0,47
intb	1316	27	1310	1323	99,54	100,53	13	0,46
ITC_b20	11419	70	11383	11496	99,68	100,67	113	0,32
s1512	439	19	439	439	100	100	0	0
Altera_ss_pcm	405	7	405	405	100	100	0	0
s838	272	41	272	272	100	100	0	0
Altera_oc_v_c_s_jpeg	49548	72	49630	49914	100,17	100,74	284	0,17
c1908	407	27	408	418	100,25	102,7	10	0,25
Altera_radar12	33079	158	33338	33431	100,78	101,06	93	0,78

Tabulka 5.16: Příkaz RESUB s permutacemi primárních výstupů - část 2.

Obvod	oA	oL	minR%	maxR%	prum+	prum-	prumC
ttt2	408	13	5,88	1,96	97,4	1,66	2,58
opa	693	18	3,32	1,01	94,87	2,86	1,17
Altera_oc_miniuart	697	15	3,16	1,43	83,58	12,36	0,77
m3	376	14	2,66	0,8	79,74	9,44	0,73
ex5	698	12	2,58	3,01	36,93	55,89	0,18
m2	274	14	2,55	2,19	46,15	37,79	0,07
newxcpla1	121	11	2,48	10,74	11,96	76,84	3,99
pope	569	12	2,46	2,28	35,27	55,74	0,12
term1	511	22	2,35	4,31	29,46	67,04	0,87
i5	284	11	2,11	1,41	59,68	21,15	0,34
b3	441	25	2,04	4,31	11,26	87,37	1,39
Altera_oc_v_c_s_dct	39355	84	1,97	0,84	100	0	1,46
Altera_nut_004	574	18	1,92	0,52	86,76	4,92	1,01
in0	618	25	1,62	2,1	42,4	53,24	0,12
Altera_barrel16	559	19	1,61	1,07	15,82	75,98	0,31
vg2	499	11	1,6	0,8	60,79	31,04	0,3
s832	319	14	1,57	0	73,83	0	0,52
e64	747	8	1,34	0,27	73,29	12,58	0,31
s820	311	14	1,29	0	72,28	0	0,41
soar	624	19	1,12	2,72	6,17	89,44	0,8
Altera_barrel16a	623	18	1,12	0,32	81,65	8,31	0,36
b10	552	25	1,09	2,9	23,44	69,77	0,77
exep	761	23	1,05	0,26	96,15	0,96	0,48
dalu	1416	36	0,99	1,06	37,85	53,29	0,05
mlp4	436	15	0,92	1,15	36,88	42,57	0,03
cps	1442	12	0,83	0,76	46,3	41,93	0,01
x3	1211	20	0,83	0,08	95,97	0,43	0,33
in3	512	25	0,78	1,37	28	55,74	0,27
x4	557	9	0,72	1,8	1,91	94,87	1
apex3	1604	13	0,56	0,44	51,7	30,2	0,04
s3384	1064	54	0,56	0	97,15	0	0,47
idxlat10n_m12	2881	18	0,52	0,73	22,56	71,3	0,11
c7552	1880	29	0,48	0,9	1,03	95,05	0,28
idxlat00n_m12	3435	20	0,47	0,58	24,39	68,83	0,08
intb	1316	27	0,46	0,53	36,79	41,38	0
ITC_b20	11419	70	0,32	0,67	10,37	88,98	0,28
s1512	439	19	0	0	0	0	0
Altera_ss_pcm	405	7	0	0	0	0	0
s838	272	41	0	0	0	0	0
Altera_oc_v_c_s_jpeg	49548	72	0,17	0,74	0	100	0,41
c1908	407	27	0,25	2,7	0	100	2,16
Altera_radar12	33079	158	0,78	1,06	0	100	0,92

Postup testování vychází z principu, který je uveden v kapitole 4.1. Ve zkratce jde tedy o opakované spuštění dvojice:

- permutace některé části obvodu
- provedení příkazu

Z tabulek vyplývá, že permutovanými částmi obvodu byly pouze:

1. primární vstupy
2. primární výstupy

Z testování zbylých dvou variant (permutace uzlů a termů) vyplynulo, že nemají žádný vliv na výsledky zkoumaných příkazů systému ABC, takže z tohoto důvody byly vynechány a v testování se neobjevují.

Příkaz BALANCE permutace primárních vstupů

Příkaz BALANCE je na permutaci primárních vstupů poměrně citlivý a z první části tabulky můžeme vidět, že na tuto variantu randomizace reagovala většina obvodů.

Lze také vypozorovat, že se díky permutaci primárních vstupů dá docílit i zlepšení v minimálním počtu dosažených AND uzlů.

Z druhé tabulky vyplývá, že použitím permutací primárních vstupů se u nadpoloviční většiny obvodu dá dosáhnout i průměrného zlepšení ze všech opakování prováděných nad daným obvodem.

Příkaz BALANCE permutace primárních výstupů

Chování příkazu BALANCE při použití permutací primárních výstupů je velmi podobné jako v případě použití permutací primárních vstupů.

Opět je většina obvodů na tyto permutaci citlivá a můžeme dosáhnout zlepšení nejenom v minimálním počtu dosažených AND uzlů, ale i v průměrných výsledcích za všechna opakování.

Lepších výsledků v počtu AND uzlů v jednotlivých opakování dosáhneme přibližně ve 45% případů stejně jako u varianty s permutacemi primárních vstupů pro příkaz BALANCE.

Příkaz REWRITE permutace primárních vstupů

V tomto případě jsou výsledky velmi podobné jako pro příkaz BALANCE a permutaci primárních vstupů nebo výstupů. Citlivost je potvrzená s tím rozdílem, že se projevuje u menšího počtu obvodů.

Dosažení lepšího počtu AND uzlů při jednotlivých opakováních je v o něco menší míře, což znamená přibližně 38% případů.

Příkaz REWRITE permutace primárních výstupů

Výsledky pro tuto kombinaci příkazu a permutací jsou velmi podobné jako pro stejný příkaz při použití permutací primárních vstupů. Jen s tím rozdílem, že zde obvody ve větším množství opakování reagují dosažením lepšího počtu AND uzlů (54%).

Příkaz REFACTOR permutace primárních vstupů

Citlivost příkazu REFACTOR na permutaci primárních vstupů je velmi malá. Tato necitlivost se projevuje nejenom v počtu obvodů, které na tuto kombinaci vůbec nereagují, ale i v dosažených výsledcích. Důvodem pro tuto necitlivost je to, že příkaz REFACTOR při své práci využívá reconvergence-driven cut, z jehož konstrukčního principu tato necitlivost plyne.

Pouze u 24% všech prováděných opakování dosáhneme průměrně lepšího počtu AND uzlů.

Příkaz REFACTOR permutace primárních výstupů

Na rozdíl od permutace primárních vstupů pro příkaz REFACTOR je permutace primárních výstupů jednou z kombinací, která podává velmi podobné výsledky jako například permutace primárních výstupů pro příkazy BALANCE a REFACTOR.

Příkaz RESUB permutace primárních vstupů

Opakuje se situace jako u příkazu REFACTOR při permutacích primárních vstupů. Citlivost je velmi malá, avšak v případě příkazu RESUB je ještě méně obvodů, které na tuto kombinaci vůbec reagovaly. Důvodem necitlivosti je stejně jako u příkazu REFACTOR využívání reconvergence-driven cut.

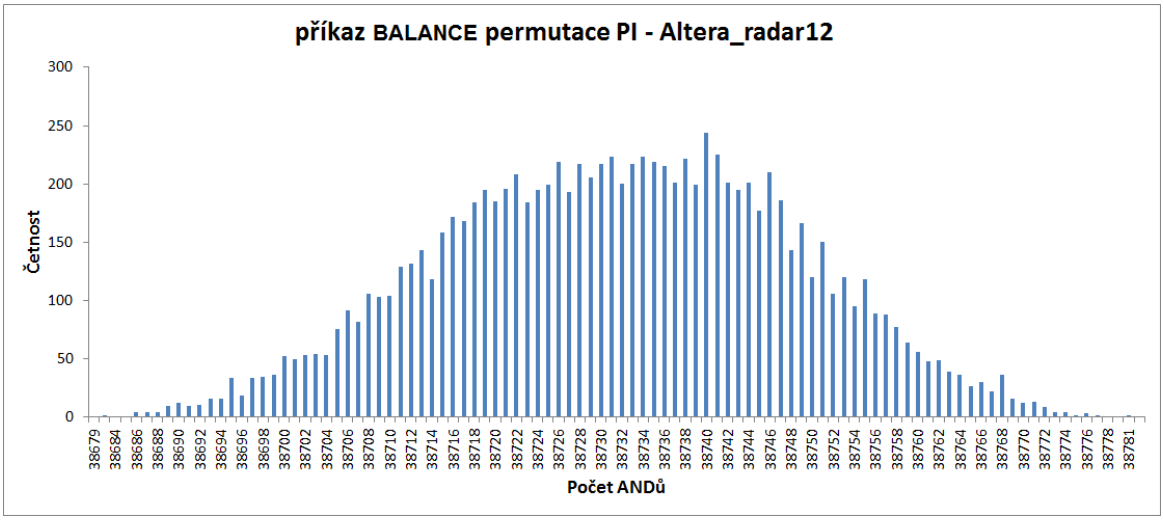
Průměrná hodnota výskytu dosažení lepšího počtu AND uzlů ze všech opakování zde klesá až k 10%.

Příkaz RESUB permutace primárních výstupů

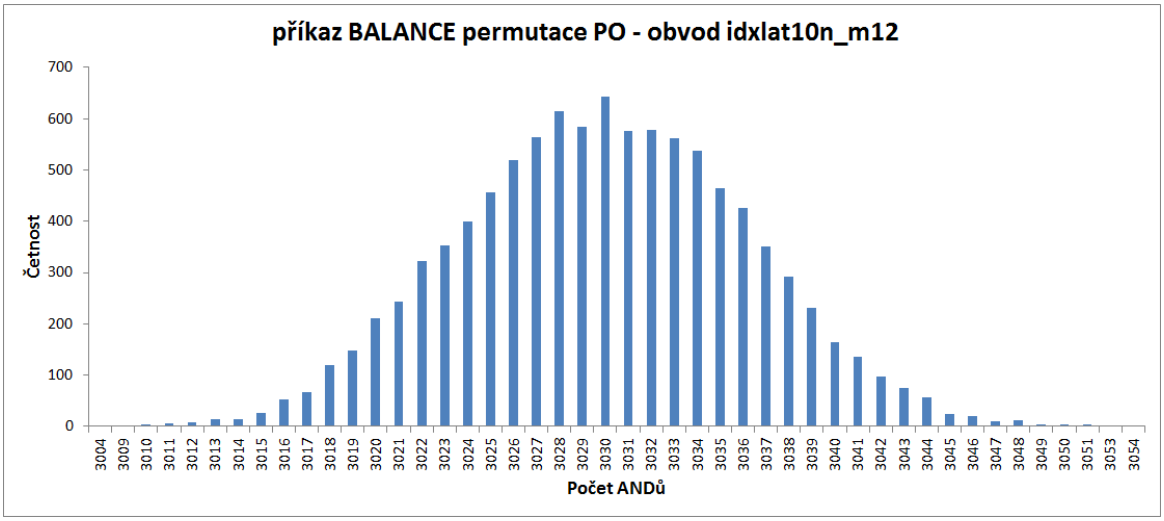
Touto kombinací je také dosaženo dobrých výsledků, které odpovídají výsledkům pro permutace primárních výstupů u příkazů BALANCE nebo REWRITE.

Tabulky podávají přehled testování za mnoho obvodů najednou. Při podrobnějším pohledu na testování pro jeden specifický obvod se dají výsledky zobrazit jako histogramy. Histogramy reprezentují četnost výskytů počtu dosažených uzlů AND pro jednotlivé opakování při testování.

Histogramy pro permutace primárních vstupů u příkazů REFACTOR a RESUB nejsou uváděny z důvodu malé citlivosti obvodů na tyto varianty randomizace.

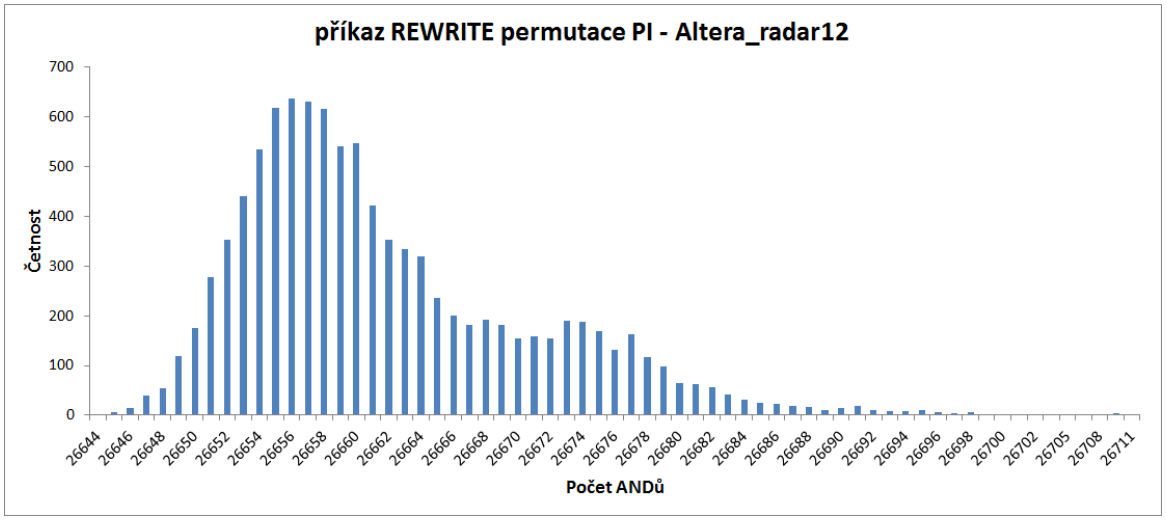


Obrázek 5.1: Histogram pro příkaz BALANCE při permutaci primárních vstupů

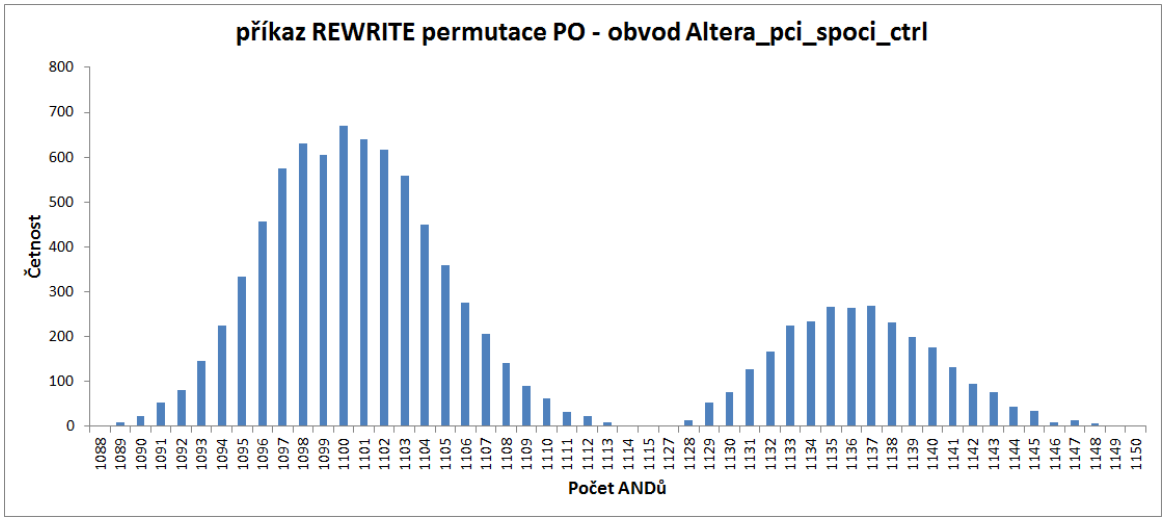


Obrázek 5.2: Histogram pro příkaz BALANCE při permutaci primárních výstupů

Z grafů pro příkaz BALANCE je vidět, že výskyty počtu dosažených AND uzlů tvoří Gaussovu křivku.

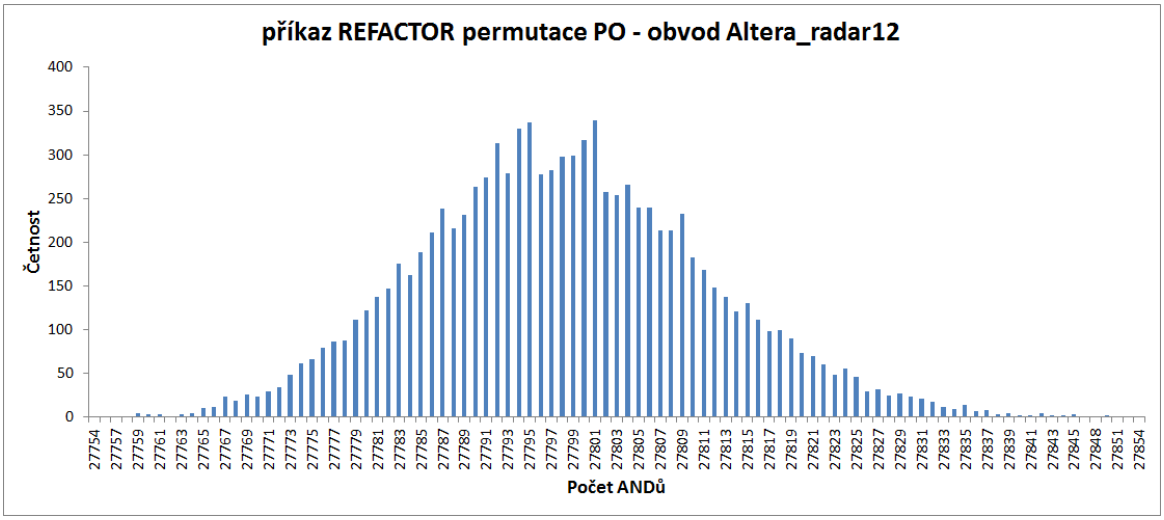


Obrázek 5.3: Histogram pro příkaz REWRITE při permutaci primárních vstupů

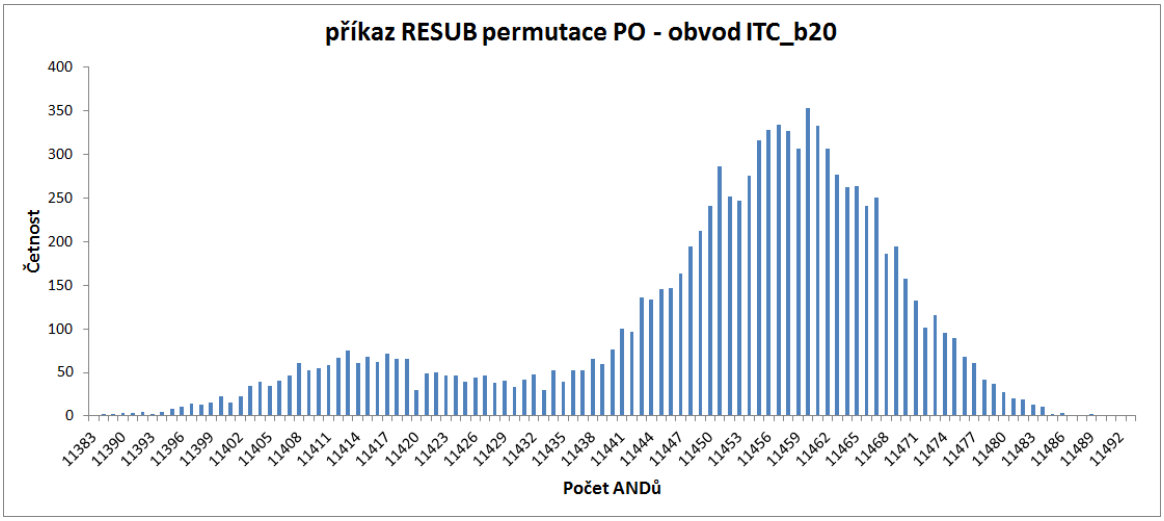


Obrázek 5.4: Histogram pro příkaz REWRITE při permutaci primárních výstupů

Zde už je situace oproti příkazu BALANCE trošku jiná, jelikož histogramy jsou reprezentovány ne úplně přesnými Gaussovými křivkami. Zvlášť histogram pro permutaci primárních výstupů u příkazu REWRITE je zajímavý, a proto byl vybrán. Pro většinu ostatních obvodů dopadlo testování standardně, a to tak, že výsledné histogramy reprezentovaly Gaussovu křivku.



Obrázek 5.5: Histogram pro příkaz REFACTOR při permutaci primárních výstupů



Obrázek 5.6: Histogram pro příkaz RESUB při permutaci primárních výstupů

Opět se dá říct, že hlavním tvarem histogramů je Gaussova křivka. V případě permutace primárních výstupu je vidět stejně jako v případě příkazu REWRITE a permutace primárních vstupů protažení křivky na jednu stranu.

5.2.2 Randomizované příkazy REWRITE, REFACTOR, RESUB

Měření je stejně jako v předchozím případě reprezentováno tabulkami a histogramy, kde formáty obou zůstávají zachovány a stejně tak i počet opakovaní, který je 10000.

Tabulka 5.17: Randomizovaný příkaz REWRITE - část 1.

Obvod	oA	oL	minA	maxA	min%	max%	d	%
Altera_mux32_16bit	1490	12	1334	1862	89,53	124,97	528	10,47
Altera_mux8_64bit	1345	7	1219	2049	90,63	152,34	830	9,37
g125	1854	23	1700	2190	91,69	118,12	490	8,31
Altera_barrel16	416	14	399	667	95,91	160,34	268	4,09
dalu	1188	35	1144	1636	96,3	137,71	492	3,7
Mentor_1_09	1308	63	1263	1601	96,56	122,4	338	3,44
Mentor_1_10	1308	63	1265	1601	96,71	122,4	336	3,29
c5315	1446	35	1403	1602	97,03	110,79	199	2,97
ex4p	1799	13	1772	2088	98,5	116,06	316	1,5
s3271	1043	21	1028	1096	98,56	105,08	68	1,44
t481	1022	16	1009	1416	98,73	138,55	407	1,27
s5378	1221	17	1207	1349	98,85	110,48	142	1,15
in0	569	22	563	719	98,95	126,36	156	1,05
apex1	1477	14	1466	1592	99,26	107,79	126	0,74
prolog	882	20	876	923	99,32	104,65	47	0,68
apex4	2272	13	2257	2448	99,34	107,75	191	0,66
ex5p	2119	15	2112	2392	99,67	112,88	280	0,33
mm30a	1009	157	1006	1008	99,7	99,9	2	0,3
ti	1036	19	1035	1238	99,9	119,5	203	0,1
s3384	1003	54	1002	1067	99,9	106,38	65	0,1
vda	826	16	826	927	100	112,23	101	0
opa	654	12	654	792	100	121,1	138	0
c6288	2305	117	2305	2334	100	101,26	29	0
dsip	2514	10	2514	2518	100	100,16	4	0
s4863	1410	45	1411	1733	100,07	122,91	322	0,07
prom2	2948	21	2951	3508	100,1	119	557	0,1
apex2	4032	17	4042	4487	100,25	111,28	445	0,25
Altera_pci_spoci_ctrl	1109	17	1114	1391	100,45	125,43	277	0,45
bc0	1294	29	1301	1608	100,54	124,27	307	0,54
k2	1489	23	1504	1724	101,01	115,78	220	1,01
Altera_oc_ata_ocidec1	1556	14	1682	2153	108,1	138,37	471	8,1
Altera_oc_minirisc	1888	43	2053	2981	108,74	157,89	928	8,74
Altera_oc_s_fm_r	2785	68	3074	4753	110,38	170,66	1679	10,38
i8	1781	20	1967	2993	110,44	168,05	1026	10,44
Altera_nut_003	2154	56	2383	3361	110,63	156,04	978	10,63
Altera_oc_des_area_opt	3048	32	3453	4677	113,29	153,44	1224	13,29
Altera_cord1	8291	55	9428	11832	113,71	142,71	2404	13,71
Altera_barrel64	2534	19	2895	4088	114,25	161,33	1193	14,25
x1	899	17	1042	1466	115,91	163,07	424	15,91
Altera_oc_v_c_s_h_dec	1811	26	2100	2929	115,96	161,73	829	15,96
clma	504	28	588	1117	116,67	221,63	529	16,67
Altera_oc_v_c_s_h_enc	2200	23	2644	3487	120,18	158,5	843	20,18

Tabulka 5.18: Randomizovaný příkaz REWRITE - část 2.

Obvod	oA	oL	minR%	maxR%	prum+	prum-	prumC
Altera_mux32_16bit	1490	12	10,47	24,97	96,73	3,27	9,07
Altera_mux8_64bit	1345	7	9,37	52,34	67,96	32,04	1,19
g125	1854	23	8,31	18,12	14,45	85,41	4,71
Altera_barrel16	416	14	4,09	60,34	9,24	90,33	29,36
dalu	1188	35	3,7	37,71	0,76	99,24	21,31
Mentor_1_09	1308	63	3,44	22,4	0,01	99,99	19,19
Mentor_1_10	1308	63	3,29	22,4	0,04	99,96	19,19
c5315	1446	35	2,97	10,79	1,37	98,54	7,12
ex4p	1799	13	1,5	16,06	0,08	99,91	15,24
s3271	1043	21	1,44	5,08	0,35	99,63	4,46
t481	1022	16	1,27	38,55	0,04	99,96	34,78
s5378	1221	17	1,15	10,48	0,02	99,98	10,24
in0	569	22	1,05	26,36	0,09	99,9	17,08
apex1	1477	14	0,74	7,79	0,18	99,78	6,13
prolog	882	20	0,68	4,65	0,47	99,34	3,1
apex4	2272	13	0,66	7,75	0,24	99,77	7,28
ex5p	2119	15	0,33	12,88	0,03	99,97	12,33
mm30a	1009	157	0,3	0,1	100	0	0,23
ti	1036	19	0,1	19,5	0,1	99,9	14,51
s3384	1003	54	0,1	6,38	13,3	86,7	0,1
vda	826	16	0	12,23	0	99,99	8,5
opa	654	12	0	21,1	0	99,99	15,15
c6288	2305	117	0	1,26	0	90,24	1,1
dsip	2514	10	0	0,16	0	12,84	0,02
s4863	1410	45	0,07	22,91	0	100	9,74
prom2	2948	21	0,1	19	0	100	17,95
apex2	4032	17	0,25	11,28	0	100	9,91
Altera_pci_spoci_ctrl	1109	17	0,45	25,43	0	100	22,05
bc0	1294	29	0,54	24,27	0	100	15,57
k2	1489	23	1,01	15,78	0	100	12,61
Altera_oc_ata_ocidec1	1556	14	8,1	38,37	0	100	34,97
Altera_oc_minirisc	1888	43	8,74	57,89	0	100	55,33
Altera_oc_s_fm_r	2785	68	10,38	70,66	0	100	36,14
i8	1781	20	10,44	68,05	0	100	60,98
Altera_nut_003	2154	56	10,63	56,04	0	100	45,85
Altera_oc_des_area_opt	3048	32	13,29	53,44	0	100	48,18
Altera_cord1	8291	55	13,71	42,71	0	100	36,99
Altera_barrel64	2534	19	14,25	61,33	0	100	40,65
x1	899	17	15,91	63,07	0	100	61,09
Altera_oc_v_c_s_h_dec	1811	26	15,96	61,73	0	100	59,78
clma	504	28	16,67	121,63	0	100	84,22
Altera_oc_v_c_s_h_enc	2200	23	20,18	58,5	0	100	52,94

Tabulka 5.19: Randomizovaný příkaz REFACTOR - část 1.

Obvod	oA	oL	minA	maxA	min%	max%	d	%
i8	1733	20	1274	3084	73,51	177,96	1810	26,49
clma	462	26	390	1128	84,42	244,16	738	15,58
Altera_barrel16	461	17	399	670	86,55	145,34	271	13,45
mm30a	1242	158	1103	1103	88,81	88,81	0	11,19
s5378	1210	17	1121	1345	92,64	111,16	224	7,36
in0	546	20	508	696	93,04	127,47	188	6,96
Mentor_1_09	1331	43	1263	1580	94,89	118,71	317	5,11
Altera_oc_s_fm_r	3151	74	3000	4670	95,21	148,21	1670	4,79
Altera_nut_003	2102	60	2005	3111	95,39	148	1106	4,61
Altera_mux8_64bit	1408	8	1344	1472	95,45	104,55	128	4,55
Mentor_1_10	1331	43	1275	1584	95,79	119,01	309	4,21
frg2	1158	16	1111	1945	95,94	167,96	834	4,06
k2	1696	23	1641	1737	96,76	102,42	96	3,24
s4863	1595	50	1547	1706	96,99	106,96	159	3,01
x3	790	14	770	1264	97,47	160	494	2,53
c5315	1516	35	1480	1663	97,63	109,7	183	2,37
Altera_mux32_16bit	1520	12	1488	1488	97,89	97,89	0	2,11
dalu	1226	34	1201	1610	97,96	131,32	409	2,04
bc0	1231	26	1209	1596	98,21	129,65	387	1,79
prolog	907	20	895	948	98,68	104,52	53	1,32
pcont2	2527	61	2504	2912	99,09	115,24	408	0,91
s3384	932	54	924	1016	99,14	109,01	92	0,86
s3271	1064	21	1057	1097	99,34	103,1	40	0,66
g125	1875	24	1864	2136	99,41	113,92	272	0,59
Altera_oc_hdlc	2110	16	2100	2800	99,53	132,7	700	0,47
des	3971	16	3953	4796	99,55	120,78	843	0,45
apex1	1572	14	1565	1617	99,55	102,86	52	0,45
i10	2257	46	2247	2506	99,56	111,03	259	0,44
opa	712	14	710	799	99,72	112,22	89	0,28
ti	1102	18	1099	1234	99,73	111,98	135	0,27
pair	1409	22	1407	1726	99,86	122,5	319	0,14
x1	837	18	836	1457	99,88	174,07	621	0,12
dsip	2522	14	2522	2522	100	100	0	0
c6288	2336	120	2336	2336	100	100	0	0
ex5p	2343	15	2343	2392	100	102,09	49	0
Altera_oc_ata_ocidec1	1569	14	1578	2087	100,57	133,01	509	0,57
Altera_cord1	8483	54	8943	11695	105,42	137,86	2752	5,42
ex4p	1561	13	1650	2097	105,7	134,34	447	5,7
Altera_oc_des_perf_o	26107	21	27769	34133	106,37	130,74	6364	6,37
Altera_oc_v_c_s_h_e	2274	24	2434	3385	107,04	148,86	951	7,04
Altera_oc_v_c_s_h_d	1887	25	2028	2853	107,47	151,19	825	7,47
t481	817	16	912	1402	111,63	171,6	490	11,63

Tabulka 5.20: Randomizovaný příkaz REFACTOR - část 2.

Obvod	oA	oL	minR%	maxR%	prum+	prum-	prumC
i8	1733	20	26,49	77,96	0,48	99,52	66,61
clma	462	26	15,58	144,16	0,22	99,78	69,89
Altera_barrel16	461	17	13,45	45,34	14,92	84,78	16,05
mm30a	1242	158	11,19	11,19	100	0	11,19
s5378	1210	17	7,36	11,16	1,74	98,17	10,27
in0	546	20	6,96	27,47	0,37	99,59	17,62
Mentor_1_09	1331	43	5,11	18,71	0,78	99,16	10,39
Altera_oc_s_fm_r	3151	74	4,79	48,21	25,74	74,26	23,39
Altera_nut_003	2102	60	4,61	48	0,02	99,98	28,63
Altera_mux8_64bit	1408	8	4,55	4,55	99,37	0,63	4,49
Mentor_1_10	1331	43	4,21	19,01	0,63	99,34	10,38
frg2	1158	16	4,06	67,96	0,02	99,98	64,18
k2	1696	23	3,24	2,42	5,67	93,63	1,29
s4863	1595	50	3,01	6,96	88,56	11,41	1,11
x3	790	14	2,53	60	0,02	99,98	38,23
c5315	1516	35	2,37	9,7	1,5	98,38	8,81
Altera_mux32_16bit	1520	12	2,11	2,11	100	0	2,11
dalu	1226	34	2,04	31,32	0,65	99,34	18,83
bc0	1231	26	1,79	29,65	0,06	99,94	20,62
prolog	907	20	1,32	4,52	1,2	98,34	3,27
pcont2	2527	61	0,91	15,24	1,01	98,6	10,87
s3384	932	54	0,86	9,01	50,25	49,75	2,15
s3271	1064	21	0,66	3,1	0,44	99,41	2,86
g125	1875	24	0,59	13,92	0,07	99,93	8,82
Altera_oc_hdlc	2110	16	0,47	32,7	0,01	99,99	23,62
des	3971	16	0,45	20,78	0,07	99,93	20,54
apex1	1572	14	0,45	2,86	0,02	99,97	2,65
i10	2257	46	0,44	11,03	0,02	99,98	7,82
opa	712	14	0,28	12,22	0,02	99,98	11,33
ti	1102	18	0,27	11,98	0,19	99,81	7,19
pair	1409	22	0,14	22,5	0,03	99,97	16,47
x1	837	18	0,12	74,07	0,02	99,98	60,36
dsip	2522	14	0	0	0	0	0
c6288	2336	120	0	0	0	0	0
ex5p	2343	15	0	2,09	0	99,99	2,07
Altera_oc_ata_ocidec1	1569	14	0,57	33,01	0	100	22,5
Altera_cord1	8483	54	5,42	37,86	0	100	34,11
ex4p	1561	13	5,7	34,34	0	100	33,52
Altera_oc_des_perf_o	26107	21	6,37	30,74	0	100	30,31
Altera_oc_v_c_s_h_e	2274	24	7,04	48,86	0	100	39,53
Altera_oc_v_c_s_h_d	1887	25	7,47	51,19	0	100	45,23
t481	817	16	11,63	71,6	0	100	66,34

Tabulka 5.21: Randomizovaný příkaz RESUB - část 1.

Obvod	oA	oL	minA	maxA	min%	max%	d	%
pcont2	2695	62	2620	2912	97,22	108,05	292	2,78
Altera_oc_s_fm_rec	3383	88	3293	4876	97,34	144,13	1583	2,66
opa	693	18	676	784	97,55	113,13	108	2,45
clma	885	34	864	1151	97,63	130,06	287	2,37
Altera_nut_003	2512	73	2467	3343	98,21	133,08	876	1,79
dalu	1416	36	1392	1661	98,31	117,3	269	1,69
in0	618	25	608	713	98,38	115,37	105	1,62
x1	1020	18	1007	1453	98,73	142,45	446	1,27
x3	1211	20	1200	1305	99,09	107,76	105	0,91
ex5p	2258	15	2242	2297	99,29	101,73	55	0,71
Altera_oc_minirisc	2402	62	2387	2974	99,38	123,81	587	0,62
frg2	1587	21	1578	1882	99,43	118,59	304	0,57
s3384	1064	54	1058	1066	99,44	100,19	8	0,56
c5315	1571	35	1563	1641	99,49	104,46	78	0,51
ex4p	1998	13	1989	2083	99,55	104,25	94	0,45
s3271	1057	21	1053	1095	99,62	103,6	42	0,38
vda	852	16	849	944	99,65	110,8	95	0,35
apex4	2259	13	2251	2452	99,65	108,54	201	0,35
i10	2210	47	2204	2504	99,73	113,3	300	0,27
Mentor_1_10	1496	43	1493	1593	99,8	106,48	100	0,2
Mentor_1_09	1496	43	1493	1593	99,8	106,48	100	0,2
pair	1645	24	1642	1716	99,82	104,32	74	0,18
Altera_pci_spoci_ctrl	1194	17	1192	1393	99,83	116,67	201	0,17
t481	1391	16	1389	1419	99,86	102,01	30	0,14
prolog	918	20	917	939	99,89	102,29	22	0,11
i8	2516	21	2514	3035	99,92	120,63	521	0,08
apex1	1490	14	1489	1603	99,93	107,58	114	0,07
apex2	4242	17	4239	4500	99,93	106,08	261	0,07
s4863	1718	51	1717	1747	99,94	101,69	30	0,06
prom2	3403	22	3401	3505	99,94	103	104	0,06
g125	2058	24	2057	2161	99,95	105	104	0,05
Altera_mux8_64bit	2113	12	2113	2177	100	103,03	64	0
Altera_barrel16	559	19	559	680	100	121,65	121	0
Altera_mux32_16bit	2370	22	2370	2370	100	100	0	0
mm30a	1395	159	1395	1395	100	100	0	0
Altera_oc_des_area_opt	4122	47	4146	4676	100,58	113,44	530	0,58
Altera_oc_ata_ocidec1	1895	18	1911	2147	100,84	113,3	236	0,84
Altera_oc_ata_ocidec2	2291	17	2316	2598	101,09	113,4	282	1,09
ti	1095	20	1114	1247	101,74	113,88	133	1,74
Altera_oc_v_c_s_h_d	2226	34	2285	2905	102,65	130,5	620	2,65
Altera_oc_des_perf_opt	28286	22	29709	34133	105,03	120,67	4424	5,03
Altera_cord1	9171	73	10068	11804	109,78	128,71	1736	9,78

Tabulka 5.22: Randomizovaný příkaz RESUB - část 2.

Obvod	oA	oL	minR%	maxR%	prum+	prum-	prumC
pcont2	2695	62	2,78	8,05	9,93	90,02	4,5
Altera_oc_s_fm_rec	3383	88	2,66	44,13	46,62	53,01	10,18
opa	693	18	2,45	13,13	0,59	99,31	9,24
clma	885	34	2,37	30,06	0,6	99,4	23,27
Altera_nut_003	2512	73	1,79	33,08	1,58	98,42	13,65
dalu	1416	36	1,69	17,3	0,78	99,17	15,08
in0	618	25	1,62	15,37	0,12	99,88	10,39
x1	1020	18	1,27	42,45	0,05	99,95	31,94
x3	1211	20	0,91	7,76	0,28	99,67	5,71
ex5p	2258	15	0,71	1,73	2,34	97,22	1,09
Altera_oc_minirisc	2402	62	0,62	23,81	0,37	99,57	13,3
frg2	1587	21	0,57	18,59	0,05	99,95	14,16
s3384	1064	54	0,56	0,19	11,33	69,5	0,06
c5315	1571	35	0,51	4,46	0,19	99,78	3,62
ex4p	1998	13	0,45	4,25	3,42	96,41	3,77
s3271	1057	21	0,38	3,6	0,04	99,92	3,24
vda	852	16	0,35	10,8	0,03	99,94	9,93
apex4	2259	13	0,35	8,54	0,24	99,73	8,23
i10	2210	47	0,27	13,3	0,02	99,96	9,19
Mentor_1_10	1496	43	0,2	6,48	2,29	97,3	2,88
Mentor_1_09	1496	43	0,2	6,48	2,05	97,32	2,88
pair	1645	24	0,18	4,32	0,09	99,9	2,93
Altera_pci_spoci_ctrl	1194	17	0,17	16,67	0,01	99,99	16,45
t481	1391	16	0,14	2,01	2,73	95,49	1,73
prolog	918	20	0,11	2,29	0,02	99,91	1,76
i8	2516	21	0,08	20,63	0,07	99,92	15,12
apex1	1490	14	0,07	7,58	0,01	99,99	6,6
apex2	4242	17	0,07	6,08	0,03	99,97	5,3
s4863	1718	51	0,06	1,69	2,89	89,44	0,89
prom2	3403	22	0,06	3	0,02	99,97	2,91
g125	2058	24	0,05	5	0,14	90,66	2,94
Altera_mux8_64bit	2113	12	0	3,03	0	1,23	0,04
Altera_barrel16	559	19	0	21,65	0	99,04	1,44
Altera_mux32_16bit	2370	22	0	0	0	0	0
mm30a	1395	159	0	0	0	0	0
Altera_oc_des_area_opt	4122	47	0,58	13,44	0	100	12,13
Altera_oc_ata_ocidec1	1895	18	0,84	13,3	0	100	11,5
Altera_oc_ata_ocidec2	2291	17	1,09	13,4	0	100	11,94
ti	1095	20	1,74	13,88	0	100	12,04
Altera_oc_v_c_s_h_d	2226	34	2,65	30,5	0	100	20,78
Altera_oc_des_perf_opt	28286	22	5,03	20,67	0	100	20,55
Altera_cord1	9171	73	9,78	28,71	0	100	23,11

Testování je provedeno pomocí úprav uvedených v kapitole 4.2. Jde tedy o programovou úpravu algoritmů v příkazech REWRITE, REFACTOR, RESUB.

Randomizované příkazy REWRITE, REFACTOR, RESUB

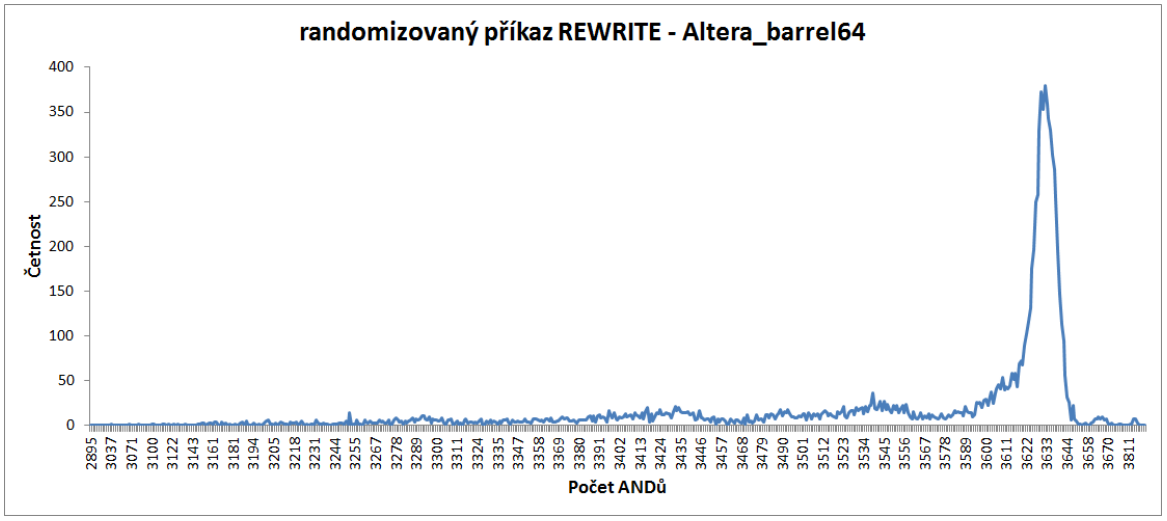
Pro všechny tři příkazy platí, že jejich randomizací dostáváme velmi odlišné výsledky pro každý obvod. Většina obvodů na tyto randomizace reaguje, avšak s dosti špatnými výsledky.

Pokud se podíváme na druhé části tabulek pro všechny tři příkazy, zjistíme, že průměrné hodnoty ze všech zlepšení a zhoršení dosažených ve všech opakováních pro jednotlivé obvody jsou velmi špatné a použití těchto randomizovaných příkazů nepodává dobré výsledky.

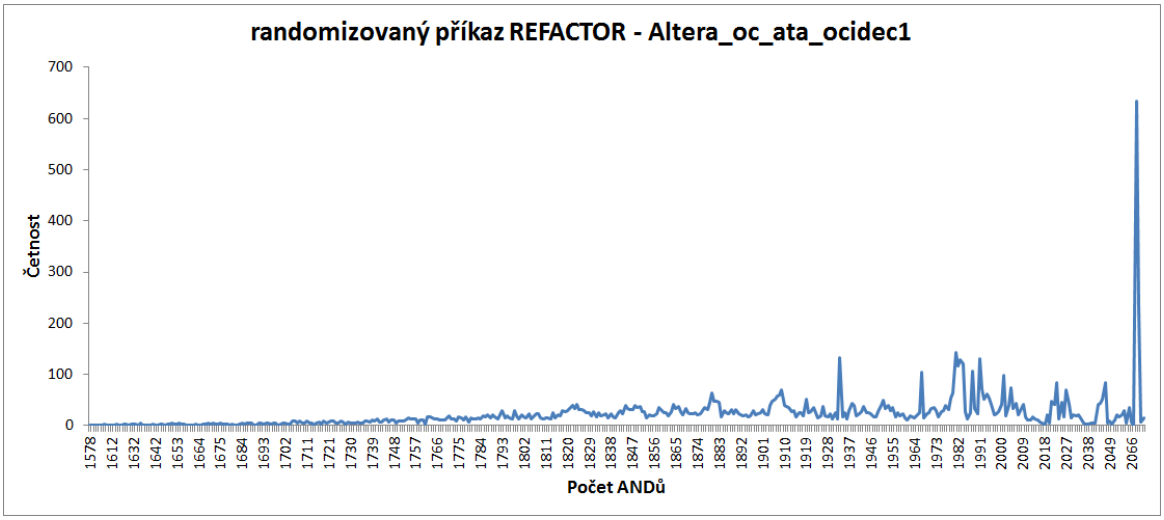
Ve výsledcích se u některých obvodů objevuje i zlepšení v minimálním dosaženém počtu AND uzlů, ale pravděpodobnost dosažení takových výsledků je s ohledem na průměrné hodnoty ze všech zlepšení a zhoršení velmi malá.

Pro všechny tři příkazy by se dalo říci, že průměrně dosáhneme lepšího počtu dosažených AND uzlů pouze v 5 až 10% případů. Oproti tomu horšího počtu až v 90%.

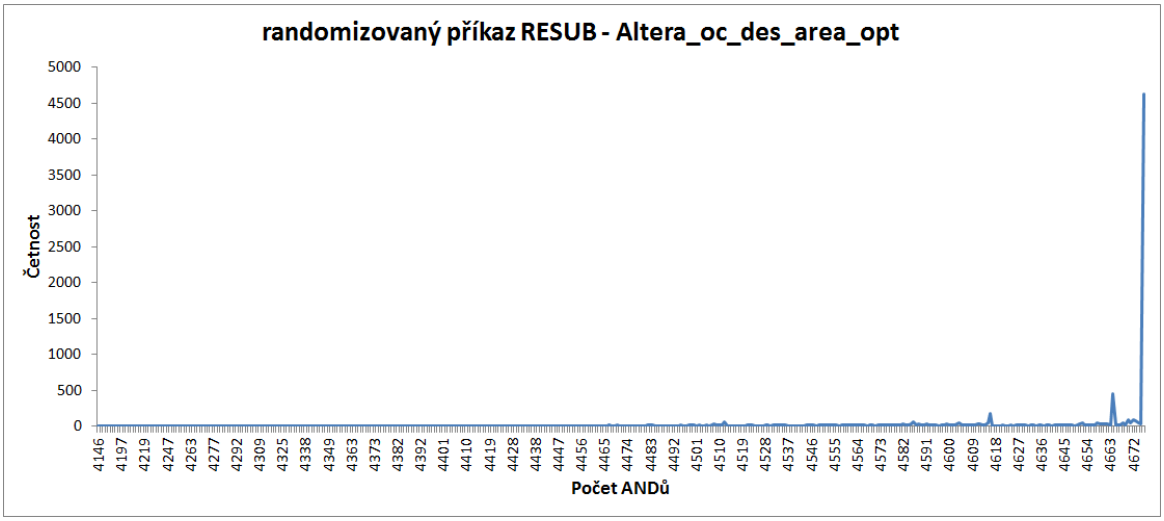
Ukázkové histogramy pro příkazy vypadají takto:



Obrázek 5.7: Histogram pro randomizovaný příkaz REFACTOR



Obrázek 5.8: Histogram pro randomizovaný příkaz REFACTOR



Obrázek 5.9: Histogram pro randomizovaný příkaz RESUB

Z histogramů lze vidět, že rozložení dosažených počtů uzlů AND v jednotlivých opakováních je dosti rovnoměrné, avšak s tím, že vždy existuje jedna hodnota, která je doslova dominantní.

Dochází tedy k tomu, že dosažené počty výskytů uzlů AND se ve většině opakování rovnají jedné hodnotě, což ještě více degraduje účinnost tohoto postupu randomizace, jelikož dosažení jiných výsledků než této hodnoty je značně nepravděpodobné.

5.2.3 Randomizovaný příkaz RR

Testování je jako v předchozím případě reprezentováno tabulkou, avšak už ne histogramy. Formát tabulky je zachován a stejně tak i počet opakovaní, který je 10000.

Tabulka 5.23: Randomizovaný příkaz RR - část 1.

Obvod	oA	oL	minA	maxA	min%	max%	d	%
Altera_oc_des_area_opt	4128	46	4072	4100	98,64	99,32	28	1,36
c2670	690	20	685	691	99,28	100,14	6	0,72
s9234,1	1902	32	1895	1897	99,63	99,74	2	0,37
s9234	1897	32	1890	1892	99,63	99,74	2	0,37
bc0	1572	30	1567	1569	99,68	99,81	2	0,32
ti	1217	19	1214	1217	99,75	100	3	0,25
s13207,1	2661	32	2656	2660	99,81	99,96	4	0,19
rot	1044	30	1042	1044	99,81	100	2	0,19
Altera_oc_v_c_s_h_e	2977	30	2972	3000	99,83	100,77	28	0,17
Altera_wb_dma	4044	24	4041	4044	99,93	100	3	0,07
s13207	2645	33	2644	2648	99,96	100,11	4	0,04
bca	3818	26	3817	3818	99,97	100	1	0,03
Altera_aes_core	20794	24	20795	20803	100	100,04	8	0
c7552	1966	28	1966	1969	100	100,15	3	0
ITC_b20	12130	70	12132	12139	100,02	100,07	7	0,02
ITC_b21	12665	72	12673	12680	100,06	100,12	7	0,06
Altera_i2c	1145	13	1146	1147	100,09	100,17	1	0,09
Altera_oc_correlator	2862	76	2865	2880	100,1	100,63	15	0,1
Altera_oc_s_fm_rec	4530	86	4535	4546	100,11	100,35	11	0,11
cps	1601	12	1603	1604	100,12	100,19	1	0,12
pcont2	2885	62	2889	2897	100,14	100,42	8	0,14
Altera_mem	16147	32	16177	16190	100,19	100,27	13	0,19
b12	913	13	915	925	100,22	101,31	10	0,22
Altera_radar12	35425	155	35511	35564	100,24	100,39	53	0,24
Mentor_1_09	1386	42	1393	1396	100,51	100,72	3	0,51
Mentor_1_10	1386	42	1393	1396	100,51	100,72	3	0,51
Altera_oc_minirisc	2587	58	2601	2633	100,54	101,78	32	0,54
i10	2343	47	2362	2366	100,81	100,98	4	0,81
Altera_oc_hdlc	2611	21	2634	2660	100,88	101,88	26	0,88
Altera_oc_ata_ocidec1	1910	17	1933	1948	101,2	101,99	15	1,2
Altera_oc_ata_vhd_3	4599	27	4657	4672	101,26	101,59	15	1,26
Altera_oc_ata_ocidec3	4648	27	4710	4726	101,33	101,68	16	1,33
Altera_nut_001	5336	125	5427	5463	101,71	102,38	36	1,71
Altera_fip_cordic_rca	2559	106	2605	2615	101,8	102,19	10	1,8
Altera_oc_rtc	1842	43	1879	1889	102,01	102,55	10	2,01
ITC_b12	942	17	961	962	102,02	102,12	1	2,02
Altera_nut_000	1511	99	1548	1563	102,45	103,44	15	2,45
Altera_os_sdram16	1692	25	1734	1750	102,48	103,43	16	2,48
Altera_oc_ata_ocidec2	2253	17	2312	2326	102,62	103,24	14	2,62
Altera_nut_003	2798	69	2872	2895	102,64	103,47	23	2,64
Altera_fip_cordic_cla	2620	99	2691	2703	102,71	103,17	12	2,71
Altera_oc_v_c_s_h_d	2248	32	2310	2331	102,76	103,69	21	2,76

Tabulka 5.24: Randomizovaný příkaz RR - část 2.

Obvod	oA	oL	minR%	maxR%	prum+	prum-	prumC
Altera_oc_des_area_opt	4128	46	1,36	0,68	100	0	1,01
c2670	690	20	0,72	0,14	64,3	4,7	0,28
s9234,1	1902	32	0,37	0,26	100	0	0,32
s9234	1897	32	0,37	0,26	100	0	0,32
bc0	1572	30	0,32	0,19	100	0	0,25
ti	1217	19	0,25	0	92,6	0	0,15
s13207,1	2661	32	0,19	0,04	100	0	0,11
rot	1044	30	0,19	0	49,6	0	0,09
Altera_oc_v_c_s_h_e	2977	30	0,17	0,77	0,8	98,7	0,32
Altera_wb_dma	4044	24	0,07	0	74,6	0	0,04
s13207	2645	33	0,04	0,11	23,4	76,6	0,04
bca	3818	26	0,03	0	47,68	0	0,01
Altera_aes_core	20794	24	0	0,04	0	100	0,02
c7552	1966	28	0	0,15	0	87,3	0,07
ITC_b20	12130	70	0,02	0,07	0	100	0,04
ITC_b21	12665	72	0,06	0,12	0	100	0,09
Altera_i2c	1145	13	0,09	0,17	0	100	0,13
Altera_oc_correlator	2862	76	0,1	0,63	0	100	0,38
Altera_oc_s_fm_rec	4530	86	0,11	0,35	0	100	0,23
cps	1601	12	0,12	0,19	0	100	0,16
pcont2	2885	62	0,14	0,42	0	100	0,24
Altera_mem	16147	32	0,19	0,27	0	100	0,23
b12	913	13	0,22	1,31	0	100	0,83
Altera_radar12	35425	155	0,24	0,39	0	100	0,32
Mentor_1_09	1386	42	0,51	0,72	0	100	0,61
Mentor_1_10	1386	42	0,51	0,72	0	100	0,61
Altera_oc_minirisc	2587	58	0,54	1,78	0	100	1,18
i10	2343	47	0,81	0,98	0	100	0,9
Altera_oc_hdlc	2611	21	0,88	1,88	0	100	1,43
Altera_oc_ata_ocidec1	1910	17	1,2	1,99	0	100	1,57
Altera_oc_ata_vhd_3	4599	27	1,26	1,59	0	100	1,43
Altera_oc_ata_ocidec3	4648	27	1,33	1,68	0	100	1,5
Altera_nut_001	5336	125	1,71	2,38	0	100	2,02
Altera_fip_cordic_rca	2559	106	1,8	2,19	0	100	1,99
Altera_oc_rtc	1842	43	2,01	2,55	0	100	2,32
ITC_b12	942	17	2,02	2,12	0	100	2,07
Altera_nut_000	1511	99	2,45	3,44	0	100	2,95
Altera_os_sdram16	1692	25	2,48	3,43	0	100	2,98
Altera_oc_ata_ocidec2	2253	17	2,62	3,24	0	100	2,92
Altera_nut_003	2798	69	2,64	3,47	0	100	3,06
Altera_fip_cordic_cla	2620	99	2,71	3,17	0	100	2,92
Altera_oc_v_c_s_h_d	2248	32	2,76	3,69	0	100	3,29

Testování vychází z kapitoly 4.3. Jde tedy opět o programovou úpravu algoritmu v příkazu RR.

Randomizovaný příkaz RR

V tabulce jsou zvoleny obvody, které jsou na tuto variantu randomizace citlivé. I přes tuto volbu dostáváme výsledek, že v 80% případů obvody dosahují horších průměrných výsledků v počtu dosažených AND uzlů ze všech opakování. Avšak na rozdíl od randomizovaných variant příkazů REWRITE, REFACTOR, RESUB průměrné hodnoty ze všech zlepšení a zhoršení pro jednotlivé obvody nejsou tak špatné.

V malém množství případů může tato metoda vést k lepšímu výsledku v minimálním dosaženém počtu AND uzlů, ale i při této situaci je dosažené zlepšení velmi malé.

Tvorba histogramů zde nemá smysl, jelikož dosažené výskyty počtu uzlů AND mají velmi malý rozptyl.

5.2.4 Příkaz COLAPSE s použitím permutací

Výsledkem testování je jako v předchozím případě tabulka. Formát tabulky vypadá takto:

- oB - počet BDD uzlů při použití algoritmu prosévání
- minB - minimální počet BDD uzlů dosažený ze všech opakování
- maxB - maximální počet BDD uzlů dosažený ze všech opakování
- min% - procentuální poměr hodnot sloupců minB k oB
- max% - procentuální poměr hodnot sloupců maxB k oB
- d - rozdíl mezi minimálním a maximálním počtem BBB uzlů ze všech opakování
- % - rozdíl v procentech mezi nejlepšími dosaženými a původními hodnotami
- minR% - relativní poměr - $(oB - minB) \div oB$
- maxR% - relativní poměr - $(oB - maxB) \div oB$
- prum+ - počet obvodů, které kladně reagují, v procentech ze všech opakování
- prum- - počet obvodů, které záporně reagují, v procentech ze všech opakování
- prumC - průměrná reakce obvodu ze všech opakování v procentech

Počet opakovaní je 1000.

Tabulka 5.25: Randomizovaný příkaz COLLAPSE - část 1.

Obvod	oB	minB	maxB	min%	max%	d	%
s635	1151	650	972	56,47	84,45	322	43,53
i6	613	349	617	56,93	100,65	268	43,07
s499	979	870	1142	88,87	116,65	272	11,13
mlp4	206	190	233	92,23	113,11	43	7,77
cht	183	170	253	92,9	138,25	83	7,1
exp	336	320	413	95,24	122,92	93	4,76
clip	141	136	299	96,45	212,06	163	3,55
dist	177	171	225	96,61	127,12	54	3,39
luc	342	334	511	97,66	149,42	177	2,34
mark1	529	525	580	99,24	109,64	55	0,76
Altera_p_c_c_a_d	146	146	236	100	161,64	90	0
apla	253	253	352	100	139,13	99	0
example2	746	751	1232	100,67	165,15	481	0,67
mp2d	145	147	258	101,38	177,93	111	1,38
al2	275	280	420	101,82	152,73	140	1,82
s298	114	119	177	104,39	155,26	58	4,39
exep	1058	1125	1528	106,33	144,42	403	6,33
c8	116	124	265	106,9	228,45	141	6,9
sct	124	133	266	107,26	214,52	133	7,26
ttt2	243	262	387	107,82	159,26	125	7,82
Altera_ss_pcm	698	755	1100	108,17	157,59	345	8,17
tms	218	237	377	108,72	172,94	140	8,72
ITC_b10	238	261	560	109,66	235,29	299	9,66
mult16b	137	151	177	110,22	129,2	26	10,22
mult32b	278	307	348	110,43	125,18	41	10,43
t3	97	108	220	111,34	226,8	112	11,34
ITC_b08	195	221	362	113,33	185,64	141	13,33
ITC_b03	134	153	247	114,18	184,33	94	14,18
ts10	400	460	2984	115	746	2524	15
jbp	727	837	1164	115,13	160,11	327	15,13
unreg	96	111	247	115,63	257,29	136	15,63
t2	192	225	306	117,19	159,38	81	17,19
s510	229	269	8989	117,47	3925,33	8720	17,47
count	200	237	465	118,5	232,5	228	18,5
gary	470	558	1395	118,72	296,81	837	18,72
in6	406	483	889	118,97	218,97	406	18,97
m3	204	246	377	120,59	184,8	131	20,59
in2	465	570	2542	122,58	546,67	1972	22,58
b9	228	284	443	124,56	194,3	159	24,56
s526	195	244	379	125,13	194,36	135	25,13
lal	146	183	334	125,34	228,77	151	25,34
s526n	195	245	385	125,64	197,44	140	25,64

Tabulka 5.26: Randomizovaný příkaz COLLAPSE - část 2.

Obvod	oB	minR%	maxR%	prum+	prum-	prumC
s635	1151	43,53	15,55	100	0	32,62
i6	613	43,07	0,65	98	2	23,13
s499	979	11,13	16,65	54	46	0,32
mlp4	206	7,77	13,11	45	54	0,94
cht	183	7,1	38,25	2	98	17,71
exp	336	4,76	22,92	3	97	11,34
clip	141	3,55	112,06	2	98	54,43
dist	177	3,39	27,12	3	97	16,16
luc	342	2,34	49,42	2	98	14
mark1	529	0,76	9,64	2	96	4,02
Altera_p_c_c_a_d	146	0	61,64	0	93	26,1
apla	253	0	39,13	0	99	19,35
example2	746	0,67	65,15	0	100	33,72
mp2d	145	1,38	77,93	0	100	38,74
al2	275	1,82	52,73	0	100	22,77
s298	114	4,39	55,26	0	100	29,03
exep	1058	6,33	44,42	0	100	15,67
c8	116	6,9	128,45	0	100	61,92
sct	124	7,26	114,52	0	100	52,98
ttt2	243	7,82	59,26	0	100	33,41
Altera_ss_pcm	698	8,17	57,59	0	100	32,06
tms	218	8,72	72,94	0	100	37,15
ITC_b10	238	9,66	135,29	0	100	65,09
mult16b	137	10,22	29,2	0	100	18,25
mult32b	278	10,43	25,18	0	100	17,86
t3	97	11,34	126,8	0	100	57,98
ITC_b08	195	13,33	85,64	0	100	47,82
ITC_b03	134	14,18	84,33	0	100	54,42
ts10	400	15	646	0	100	83,78
jbp	727	15,13	60,11	0	100	31,53
unreg	96	15,63	157,29	0	100	69,91
t2	192	17,19	59,38	0	100	37,33
s510	229	17,47	3825,33	0	100	213,72
count	200	18,5	132,5	0	100	69,95
gary	470	18,72	196,81	0	100	95,17
in6	406	18,97	118,97	0	100	65,84
m3	204	20,59	84,8	0	100	47,26
in2	465	22,58	446,67	0	100	155,48
b9	228	24,56	94,3	0	100	66,41
s526	195	25,13	94,36	0	100	62,67
lal	146	25,34	128,77	0	100	75,99
s526n	195	25,64	97,44	0	100	64,8

Tabulka 5.27: Randomizovaný příkaz COLLAPSE - část 3.

Obvod	oB	minB	maxB	min%	max%	d	%
b4	396	521	811	131,57	204,8	290	31,57
misex3c	567	749	1440	132,1	253,97	691	32,1
s641	1130	1493	3928	132,12	347,61	2435	32,12
x2dn	197	267	479	135,53	243,15	212	35,53
s444	205	281	578	137,07	281,95	297	37,07
s713	1130	1558	4838	137,88	428,14	3280	37,88
s382	213	294	620	138,03	291,08	326	38,03
s420,1	215	298	4734	138,6	2201,86	4436	38,6
duke2	642	930	1733	144,86	269,94	803	44,86
i5	706	1065	1676	150,85	237,39	611	50,85
s400	213	325	601	152,58	282,16	276	52,58
in3	569	870	3611	152,9	634,62	2741	52,9
in7	185	285	877	154,05	474,05	592	54,05
chkn	397	617	1447	155,42	364,48	830	55,42
ITC_b13	453	705	3623	155,63	799,78	2918	55,63
apex7	584	946	4246	161,99	727,05	3300	61,99
ex4	579	948	1403	163,73	242,31	455	63,73
ibm	476	792	1659	166,39	348,53	867	66,39
in5	435	734	2395	168,74	550,57	1661	68,74
s349	135	231	650	171,11	481,48	419	71,11
vtx1	249	428	3343	171,89	1342,57	2915	71,89
vg2	280	482	3177	172,14	1134,64	2695	72,14
mm4a	439	772	2223	175,85	506,38	1451	75,85
Altera_s	492	918	1699	186,59	345,33	781	86,59
x6dn	401	766	42993	191,02	10721,45	42227	91,02
s420	163	326	9483	200	5817,79	9157	100
x1dn	249	524	2248	210,44	902,81	1724	110,44
x9dn	271	612	2693	225,83	993,73	2081	125,83
s1196	1164	2648	11031	227,49	947,68	8383	127,49
ITC_b07	1503	4058	43554	269,99	2897,8	39496	169,99
signet	1683	5439	49692	323,17	2952,58	44253	223,17
c499	68651	287623	2624293	418,96	3822,66	2336670	318,96
s838	361	2754	266389	762,88	73791,97	263635	662,88
mult16a	453	3511	166099	775,06	36666,45	162588	675,06
my_adder	457	5252	171385	1149,23	37502,19	166133	1049,23
mm9a	3181	41243	924475	1296,54	29062,4	883232	1196,54
comp	165	2482	285554	1504,24	173063,03	283072	1404,24
g25	1305	23047	582242	1766,05	44616,25	559195	1666,05
c432	1394	46331	902370	3323,6	64732,42	856039	3223,6
i3	132	5202	138492	3940,91	104918,18	133290	3840,91
i4	210	38633	333333	18396,67	158730	294700	18296,67
mult32a	1708	485298	1001531	28413,23	58637,65	516233	28313,23

Tabulka 5.28: Randomizovaný příkaz COLLAPSE - část 4.

Obvod	oB	minR%	maxR%	prum+	prum-	prumC
b4	396	31,57	104,8	0	100	66,8
misex3c	567	32,1	153,97	0	100	76,85
s641	1130	32,12	247,61	0	100	133,79
x2dn	197	35,53	143,15	0	100	73,62
s444	205	37,07	181,95	0	100	113,38
s713	1130	37,88	328,14	0	100	135,31
s382	213	38,03	191,08	0	100	105,68
s420,1	215	38,6	2101,86	0	100	438,94
duke2	642	44,86	169,94	0	100	91,96
i5	706	50,85	137,39	0	100	87,04
s400	213	52,58	182,16	0	100	118,47
in3	569	52,9	534,62	0	100	154,78
in7	185	54,05	374,05	0	100	156,63
chkn	397	55,42	264,48	0	100	155,9
ITC_b13	453	55,63	699,78	0	100	141,06
apex7	584	61,99	627,05	0	100	140,95
ex4	579	63,73	142,31	0	100	95,65
ibm	476	66,39	248,53	0	100	150,94
in5	435	68,74	450,57	0	100	235,47
s349	135	71,11	381,48	0	100	187,79
vtx1	249	71,89	1242,57	0	100	304,89
vg2	280	72,14	1034,64	0	100	301,64
mm4a	439	75,85	406,38	0	100	210,43
Altera_s	492	86,59	245,33	0	100	165,65
x6dn	401	91,02	10621,45	0	100	803,15
s420	163	100	5717,79	0	100	743,16
x1dn	249	110,44	802,81	0	100	317,98
x9dn	271	125,83	893,73	0	100	356,16
s1196	1164	127,49	847,68	0	100	349,39
ITC_b07	1503	169,99	2797,8	0	100	688,95
signet	1683	223,17	2852,58	0	100	797,29
c499	68651	318,96	3722,66	0	100	948,24
s838	361	662,88	73691,97	0	100	11101,4
mult16a	453	675,06	36566,45	0	100	5178,88
my_adder	457	1049,23	37402,19	0	100	5965,77
mm9a	3181	1196,54	28962,4	0	100	10184,59
comp	165	1404,24	172963,03	0	100	17662,64
g25	1305	1666,05	44516,25	0	100	5412,07
c432	1394	3223,6	64632,42	0	100	28342,13
i3	132	3840,91	104818,18	0	100	16309,45
i4	210	18296,67	158630	0	100	58228,41
mult32a	1708	28313,23	58537,65	0	100	47208,16

Princip testování je popsán v kapitole 4.4 a spočívá ve dvou krocích jako v případě příkazu BALANCE:

- permutace primárních vstupů
- provedení příkazu

Permutovány jsou tedy pouze primární vstupy reprezentující pořadí vstupních proměnných.

Příkaz COLLAPSE permutace primárních vstupů

Z tabulek vyplývá, že příkaz COLLAPSE je na permutaci primárních vstupů, která mění pořadí vstupních proměnných, obrovským způsobem závislý. Pomocí permutace můžeme tedy dosahovat úplně odlišných výsledků a rozdíly mezi těmito výsledky můžou být až takové, že provedení příkazu COLLAPSE se stane nemožným.

Předpokladem zde bylo, že náhodným pořadím vstupních proměnných dosáhneme daleko horších výsledků, než za použití algoritmu prosévání pro jejich řazení. Toto tvrzení se vesměs i potvrdilo, avšak i zde došlo k několika výjimkám, které ukazují, že náhodné řazení může dosáhnout lepších výsledků.

Dosažení takových výsledků při reálném použití je ale značně nepravděpodobné.

5.2.5 Kombinace permutací a randomizovaných příkazů

Výstupem testování je tabulka, jejíž formát odpovídá testování randomizovaných příkazů nebo testování příkazů s permutacemi.

Počet opakovaní je 10000. Zelená barva opět značí dosažení lepšího výsledku, zatímco červená opak.

Tabulka 5.29: Kombinace permutací a randomizovaného příkazu REWRITE - část 1.

Obvod	oA	oL	minA	maxA	min%	max%	d	%
frg1	378	13	318	543	84,13	143,65	225	15,87
g125	1854	23	1700	2190	91,69	118,12	490	8,31
ITC_b11	545	26	511	620	93,76	113,76	109	6,24
div16	626	59	598	751	95,53	119,97	153	4,47
mult16	471	56	455	553	96,6	117,41	98	3,4
ex5	637	11	620	934	97,33	146,62	314	2,67
apex5	1790	12	1749	3010	97,71	168,16	1261	2,29
amd	500	16	489	659	97,8	131,8	170	2,2
s5378	1221	17	1197	1349	98,03	110,48	152	1,97
cps	1394	12	1371	1566	98,35	112,34	195	1,65
max512	613	19	605	740	98,69	120,72	135	1,31
alu4	3203	16	3164	3519	98,78	109,87	355	1,22
max1024	917	20	906	1013	98,8	110,47	107	1,2
in1	1464	20	1453	1773	99,25	121,11	320	0,75
Altera_pci_spoci_ctrl	1109	17	1102	1391	99,37	125,43	289	0,63
apex3	1544	13	1539	1704	99,68	110,36	165	0,32
s6669	1905	75	1902	1975	99,84	103,67	73	0,16
apex6	641	15	640	660	99,84	102,96	20	0,16
exps	1339	16	1337	1622	99,85	121,14	285	0,15
b2	1464	20	1462	1773	99,86	121,11	311	0,14
s3384	1003	54	1002	1067	99,9	106,38	65	0,1
c6288	2305	117	2305	2334	100	101,26	29	0
dsip	2514	10	2514	2518	100	100,16	4	0
prom2	2948	21	2962	3508	100,47	119	546	0,47
t481	1022	16	1027	1416	100,49	138,55	389	0,49
ex1010	2754	23	2772	3337	100,65	121,17	565	0,65
s9234,1	1583	29	1610	1965	101,71	124,13	355	1,71
s9234	1577	29	1605	1961	101,78	124,35	356	1,78
Altera_nut_002	782	25	796	1199	101,79	153,32	403	1,79
x3	744	16	768	1307	103,23	175,67	539	3,23
c7552	1708	27	1764	2002	103,28	117,21	238	3,28
bcd	1680	24	1740	2177	103,57	129,58	437	3,57
cordic	1017	18	1056	2735	103,83	268,93	1679	3,83
Altera_nut_003	2154	56	2260	3361	104,92	156,04	1101	4,92
Altera_oc_ata_ocidec2	1794	13	1892	2570	105,46	143,26	678	5,46
Altera_oc_ata_ocidec1	1556	14	1685	2153	108,29	138,37	468	8,29
Altera_oc_ata_vhd_3	3577	18	3997	5223	111,74	146,02	1226	11,74
Altera_oc_ata_ocidec3	3641	18	4108	5290	112,83	145,29	1182	12,83
Altera_oc_v_c_s_h_dec	1811	26	2084	2929	115,07	161,73	845	15,07
Altera_barrel64	2534	19	2932	4067	115,71	160,5	1135	15,71
Altera_nut_001	3860	104	4651	6224	120,49	161,24	1573	20,49
Altera_oc_v_c_s_h_enc	2200	23	2655	3487	120,68	158,5	832	20,68

Tabulka 5.30: Kombinace permutací a randomizovaného příkazu REWRITE - část 2.

Obvod	oA	oL	minR%	maxR%	prum+	prum-	prumC
frg1	378	13	15,87	43,65	1,88	98,01	25,04
g125	1854	23	8,31	18,12	14,85	84,99	4,69
ITC_b11	545	26	6,24	13,76	15,81	83,72	9,6
div16	626	59	4,47	19,97	63,03	34,95	3,7
mult16	471	56	3,4	17,41	65,62	30,81	0,27
ex5	637	11	2,67	46,62	0,07	99,93	34,25
apex5	1790	12	2,29	68,16	0,01	99,99	52,71
amd	500	16	2,2	31,8	0,08	99,9	25,25
s5378	1221	17	1,97	10,48	0,03	99,97	10,26
cps	1394	12	1,65	12,34	0,08	99,9	10,24
max512	613	19	1,31	20,72	0,03	99,97	19,04
alu4	3203	16	1,22	9,87	0,2	99,78	8,44
max1024	917	20	1,2	10,47	0,58	99,27	8,29
in1	1464	20	0,75	21,11	0,05	99,95	15,83
Altera_pci_spoci_ctrl	1109	17	0,63	25,43	0,01	99,99	21,96
apex3	1544	13	0,32	10,36	0,17	99,76	8
s6669	1905	75	0,16	3,67	0,18	99,64	0,99
apex6	641	15	0,16	2,96	0,02	99,97	2,8
exps	1339	16	0,15	21,14	0,03	99,94	18,92
b2	1464	20	0,14	21,11	0,02	99,97	15,71
s3384	1003	54	0,1	6,38	14,09	85,91	0,09
c6288	2305	117	0	1,26	0	90,41	1,11
dsip	2514	10	0	0,16	0	12,98	0,02
prom2	2948	21	0,47	19	0	100	17,96
t481	1022	16	0,49	38,55	0	100	34,74
ex1010	2754	23	0,65	21,17	0	100	20,5
s9234,1	1583	29	1,71	24,13	0	100	22,42
s9234	1577	29	1,78	24,35	0	100	23,26
Altera_nut_002	782	25	1,79	53,32	0	100	45,89
x3	744	16	3,23	75,67	0	100	58,02
c7552	1708	27	3,28	17,21	0	100	16,73
bcd	1680	24	3,57	29,58	0	100	23,12
cordic	1017	18	3,83	168,93	0	100	153,13
Altera_nut_003	2154	56	4,92	56,04	0	100	45,93
Altera_oc_ata_ocidec2	1794	13	5,46	43,26	0	100	37,41
Altera_oc_ata_ocidec1	1556	14	8,29	38,37	0	100	34,89
Altera_oc_ata_vhd_3	3577	18	11,74	46,02	0	100	38,87
Altera_oc_ata_ocidec3	3641	18	12,83	45,29	0	100	38,44
Altera_oc_v_c_s_h_dec	1811	26	15,07	61,73	0	100	59,83
Altera_barrel64	2534	19	15,71	60,5	0	100	40,58
Altera_nut_001	3860	104	20,49	61,24	0	100	52,93
Altera_oc_v_c_s_h_enc	2200	23	20,68	58,5	0	100	52,93

Tabulka 5.31: Kombinace permutací a randomizovaného příkazu REFACTOR - část 1.

Obvod	oA	oL	minA	maxA	min%	max%	d	%
frg1	344	17	252	615	73,26	178,78	363	26,74
cordic	1164	18	1039	2782	89,26	239	1743	10,74
Altera_nut_002	756	27	694	1076	91,8	142,33	382	8,2
div16	659	56	608	747	92,26	113,35	139	7,74
s5378	1210	17	1123	1345	92,81	111,16	222	7,19
amd	494	15	465	638	94,13	129,15	173	5,87
Altera_nut_001	3884	109	3692	5797	95,06	149,25	2105	4,94
c7552	1799	27	1713	1978	95,22	109,95	265	4,78
ex5	669	13	640	922	95,67	137,82	282	4,33
s6669	2011	80	1951	2161	97,02	107,46	210	2,98
Altera_pci_spoci_ctrl	1074	17	1049	1391	97,67	129,52	342	2,33
ITC_b11	552	28	542	602	98,19	109,06	60	1,81
max512	697	19	686	736	98,42	105,6	50	1,58
in1	1565	19	1543	1771	98,59	113,16	228	1,41
b2	1565	19	1543	1771	98,59	113,16	228	1,41
s9234,1	1585	28	1564	1938	98,68	122,27	374	1,32
s9234	1576	28	1558	1933	98,86	122,65	375	1,14
x3	790	14	782	1264	98,99	160	482	1,01
cps	1472	12	1459	1600	99,12	108,7	141	0,88
s3384	932	54	924	1016	99,14	109,01	92	0,86
g125	1875	24	1863	2138	99,36	114,03	275	0,64
mult16	487	45	484	554	99,38	113,76	70	0,62
exps	1469	17	1464	1598	99,66	108,78	134	0,34
apex3	1699	13	1698	1745	99,94	102,71	47	0,06
c6288	2336	120	2336	2336	100	100	0	0
apex6	650	15	650	657	100	101,08	7	0
max1024	1012	20	1012	1014	100	100,2	2	0
dsip	2522	14	2522	2522	100	100	0	0
ex1010	3097	22	3098	3303	100,03	106,65	205	0,03
Altera_oc_ata_ocidec1	1569	14	1571	2087	100,13	133,01	516	0,13
Altera_barrel64	2875	22	2880	4161	100,17	144,73	1281	0,17
prom2	3235	22	3246	3478	100,34	107,51	232	0,34
t481	817	16	824	1402	100,86	171,6	578	0,86
Altera_oc_ata_ocidec2	1776	14	1797	2535	101,18	142,74	738	1,18
bcd	1885	25	1910	2187	101,33	116,02	277	1,33
alu4	3036	16	3080	3594	101,45	118,38	514	1,45
Altera_nut_003	2102	60	2159	3098	102,71	147,38	939	2,71
apex5	1540	12	1592	3292	103,38	213,77	1700	3,38
Altera_oc_ata_ocidec3	3610	20	3734	5129	103,43	142,08	1395	3,43
Altera_oc_ata_vhd_3	3597	20	3803	5079	105,73	141,2	1276	5,73
Altera_oc_v_c_s_h_dec	1887	25	2026	2853	107,37	151,19	827	7,37
Altera_oc_v_c_s_h_enc	2274	24	2446	3387	107,56	148,94	941	7,56

Tabulka 5.32: Kombinace permutací a randomizovaného příkazu REFACTOR - část 2.

Obvod	oA	oL	minR%	maxR%	prum+	prum-	prumC
frg1	344	17	26,74	78,78	0,89	99,07	63,79
cordic	1164	18	10,74	139	0,05	99,95	133,43
Altera_nut_002	756	27	8,2	42,33	0,38	99,61	24,5
div16	659	56	7,74	13,35	91,67	6,47	1,02
s5378	1210	17	7,19	11,16	1,55	98,32	10,31
amd	494	15	5,87	29,15	0,34	99,64	19,41
Altera_nut_001	3884	109	4,94	49,25	0,08	99,92	32,97
c7552	1799	27	4,78	9,95	0,36	99,62	7,95
ex5	669	13	4,33	37,82	0,03	99,97	29,28
s6669	2011	80	2,98	7,46	99,96	0,04	1,8
Altera_pci_spoci_ctrl	1074	17	2,33	29,52	0,04	99,94	25,17
ITC_b11	552	28	1,81	9,06	1,77	97,76	5,7
max512	697	19	1,58	5,6	0,23	99,7	4,74
in1	1565	19	1,41	13,16	0,16	99,84	9,17
b2	1565	19	1,41	13,16	0,11	99,89	9,19
s9234,1	1585	28	1,32	22,27	0,12	99,88	16,42
s9234	1576	28	1,14	22,65	0,11	99,89	16,87
x3	790	14	1,01	60	0,01	99,99	38,26
cps	1472	12	0,88	8,7	0,04	99,96	8,34
s3384	932	54	0,86	9,01	50,15	49,84	2,14
g125	1875	24	0,64	14,03	0,07	99,91	8,73
mult16	487	45	0,62	13,76	27,07	17,81	0,29
exps	1469	17	0,34	8,78	0,04	99,94	6,61
apex3	1699	13	0,06	2,71	0,02	99,79	2,22
c6288	2336	120	0	0	0	0	0
apex6	650	15	0	1,08	0	99,6	0,89
max1024	1012	20	0	0,2	0	95,76	0,19
dsip	2522	14	0	0	0	0	0
ex1010	3097	22	0,03	6,65	0	100	5,73
Altera_oc_ata_ocidec1	1569	14	0,13	33,01	0	100	22,41
Altera_barrel64	2875	22	0,17	44,73	0	100	26,37
prom2	3235	22	0,34	7,51	0	100	6,01
t481	817	16	0,86	71,6	0	100	66,1
Altera_oc_ata_ocidec2	1776	14	1,18	42,74	0	100	29,28
bcd	1885	25	1,33	16,02	0	100	12,66
alu4	3036	16	1,45	18,38	0	100	17,9
Altera_nut_003	2102	60	2,71	47,38	0	100	28,86
apex5	1540	12	3,38	113,77	0	100	104,36
Altera_oc_ata_ocidec3	3610	20	3,43	42,08	0	100	32,48
Altera_oc_ata_vhd_3	3597	20	5,73	41,2	0	100	31,29
Altera_oc_v_c_s_h_dec	1887	25	7,37	51,19	0	100	45,36
Altera_oc_v_c_s_h_enc	2274	24	7,56	48,94	0	100	39,51

Tabulka 5.33: Kombinace permutací a randomizovaného příkazu RESUB - část 1.

Obvod	oA	oL	minA	maxA	min%	max%	d	%
frg1	443	18	419	612	94,58	138,15	193	5,42
Altera_nut_001	4505	122	4396	6195	97,58	137,51	1799	2,42
exps	1466	16	1437	1617	98,02	110,3	180	1,98
dalu	1416	36	1389	1661	98,09	117,3	272	1,91
max1024	855	20	840	1001	98,25	117,08	161	1,75
ex5	698	12	686	938	98,28	134,38	252	1,72
Altera_nut_003	2512	73	2479	3343	98,69	133,08	864	1,31
amd	550	17	543	648	98,73	117,82	105	1,27
x3	1211	20	1197	1305	98,84	107,76	108	1,16
Altera_wb_dma	3942	24	3905	4067	99,06	103,17	162	0,94
ITC_b11	556	27	551	599	99,1	107,73	48	0,9
in1	1548	21	1535	1786	99,16	115,37	251	0,84
Altera_nut_002	881	30	874	1192	99,21	135,3	318	0,79
apex3	1604	13	1592	1714	99,25	106,86	122	0,75
div16	701	61	696	752	99,29	107,28	56	0,71
mult16	506	46	503	556	99,41	109,88	53	0,59
s3384	1064	54	1058	1066	99,44	100,19	8	0,56
apex6	647	15	644	657	99,54	101,55	13	0,46
b2	1548	21	1542	1786	99,61	115,37	244	0,39
Altera_systemcdes	2723	26	2714	2965	99,67	108,89	251	0,33
max512	651	19	649	741	99,69	113,82	92	0,31
alu4	3216	16	3206	3600	99,69	111,94	394	0,31
Mentor_1_09	1496	43	1493	1593	99,8	106,48	100	0,2
Mentor_1_10	1496	43	1493	1593	99,8	106,48	100	0,2
pair	1645	24	1642	1716	99,82	104,32	74	0,18
t481	1391	16	1389	1419	99,86	102,01	30	0,14
seq	3927	14	3922	4169	99,87	106,16	247	0,13
prom1	7488	24	7483	7781	99,93	103,91	298	0,07
ex1010	3290	23	3288	3338	99,94	101,46	50	0,06
g125	2058	24	2057	2161	99,95	105	104	0,05
prom2	3403	22	3402	3505	99,97	103	103	0,03
cordic	2725	18	2725	2766	100	101,5	41	0
Altera_mux32_16bit	2370	22	2370	2370	100	100	0	0
apex5	3146	12	3146	3246	100	103,18	100	0
c6288	2334	120	2334	2334	100	100	0	0
dsip	2518	14	2518	2522	100	100,16	4	0
s9234	1817	32	1817	1958	100	107,76	141	0
Altera_oc_ata_ocidec3	4687	27	4745	5310	101,24	113,29	565	1,24
c7552	1880	29	1904	1999	101,28	106,33	95	1,28
Altera_oc_des_des3area	7198	72	7305	8010	101,49	111,28	705	1,49
Altera_oc_v_c_s_h_enc	3117	31	3167	3495	101,6	112,13	328	1,6
Altera_oc_v_c_s_h_dec	2226	34	2300	2905	103,32	130,5	605	3,32

Tabulka 5.34: Kombinace permutací a randomizovaného příkazu RESUB - část 2.

Obvod	oA	oL	minR%	maxR%	prum+	prum-	prumC
frg1	443	18	5,42	38,15	0,85	98,99	25,29
Altera_nut_001	4505	122	2,42	37,51	2,05	97,95	20,26
exps	1466	16	1,98	10,3	0,06	99,93	8,84
dalu	1416	36	1,91	17,3	0,83	99,11	15,19
max1024	855	20	1,75	17,08	1,46	98,48	15,33
ex5	698	12	1,72	34,38	0,05	99,93	25,41
Altera_nut_003	2512	73	1,31	33,08	1,19	98,81	13,7
amd	550	17	1,27	17,82	0,22	99,75	13,64
x3	1211	20	1,16	7,76	0,39	99,57	5,67
Altera_wb_dma	3942	24	0,94	3,17	0,05	99,94	2,94
ITC_b11	556	27	0,9	7,73	0,95	98,24	5,01
in1	1548	21	0,84	15,37	0,03	99,97	14,27
Altera_nut_002	881	30	0,79	35,3	0,06	99,92	16,87
apex3	1604	13	0,75	6,86	0,79	98,85	5,21
div16	701	61	0,71	7,28	33,34	66,47	3,06
mult16	506	46	0,59	9,88	78,24	21,76	1,54
s3384	1064	54	0,56	0,19	11,25	68,16	0,06
apex6	647	15	0,46	1,55	0,36	99,6	1,43
b2	1548	21	0,39	15,37	0,02	99,98	14,3
Altera_systemcdes	2723	26	0,33	8,89	0,01	99,99	8,46
max512	651	19	0,31	13,82	0,01	99,97	13,21
alu4	3216	16	0,31	11,94	0,05	99,94	11,81
Mentor_1_09	1496	43	0,2	6,48	2,34	97,25	2,86
Mentor_1_10	1496	43	0,2	6,48	2,36	97,04	2,86
pair	1645	24	0,18	4,32	0,06	99,92	2,92
t481	1391	16	0,14	2,01	2,63	95,69	1,74
seq	3927	14	0,13	6,16	0,02	99,98	5,2
prom1	7488	24	0,07	3,91	0,03	99,97	3,73
ex1010	3290	23	0,06	1,46	0,01	99,99	1,45
g125	2058	24	0,05	5	0,06	90,96	3
prom2	3403	22	0,03	3	0,03	99,96	2,91
cordic	2725	18	0	1,5	0	86,64	1,07
Altera_mux32_16bit	2370	22	0	0	0	0	0
apex5	3146	12	0	3,18	0	99,87	2,33
c6288	2334	120	0	0	0	0	0
dsip	2518	14	0	0,16	0	99,99	0,16
s9234	1817	32	0	7,76	0	99,99	7,26
Altera_oc_ata_ocidec3	4687	27	1,24	13,29	0	100	11,36
c7552	1880	29	1,28	6,33	0	100	5,98
Altera_oc_des_des3area	7198	72	1,49	11,28	0	100	9,31
Altera_oc_v_c_s_h_enc	3117	31	1,6	12,13	0	100	11,69
Altera_oc_v_c_s_h_dec	2226	34	3,32	30,5	0	100	20,83

Tabulka 5.35: Kombinace permutací a randomizovaného příkazu RR - část 1.

Obvod	oA	oL	minA	maxA	min%	max%	d	%
Altera_oc_des_area_o	4128	46	4070	4104	98,59	99,42	34	1,41
c2670	690	20	685	691	99,28	100,14	6	0,72
s9234	1897	32	1890	1892	99,63	99,74	2	0,37
s9234,1	1902	32	1895	1897	99,63	99,74	2	0,37
bc0	1572	30	1567	1569	99,68	99,81	2	0,32
ti	1217	19	1214	1217	99,75	100	3	0,25
rot	1044	30	1042	1044	99,81	100	2	0,19
s13207,1	2661	32	2656	2660	99,81	99,96	4	0,19
Altera_oc_v_c_s_h_e	2977	30	2974	2998	99,9	100,71	24	0,1
Altera_wb_dma	4044	24	4041	4044	99,93	100	3	0,07
s13207	2645	33	2644	2648	99,96	100,11	4	0,04
bca	3818	26	3817	3818	99,97	100	1	0,03
Altera_aes_core	20794	24	20795	20803	100	100,04	8	0
c7552	1966	28	1966	1969	100	100,15	3	0
Altera_i2c	1145	13	1146	1147	100,09	100,17	1	0,09
Altera_oc_correlator	2862	76	2865	2881	100,1	100,66	16	0,1
Altera_oc_s_fm_r	4530	86	4535	4546	100,11	100,35	11	0,11
cps	1601	12	1603	1604	100,12	100,19	1	0,12
pcont2	2885	62	2889	2897	100,14	100,42	8	0,14
Altera_mem	16147	32	16177	16190	100,19	100,27	13	0,19
b12	913	13	915	925	100,22	101,31	10	0,22
Altera_radar12	35425	155	35510	35568	100,24	100,4	58	0,24
Altera_oc_minirisc	2587	58	2599	2632	100,46	101,74	33	0,46
Mentor_1_10	1386	42	1393	1396	100,51	100,72	3	0,51
Mentor_1_09	1386	42	1393	1396	100,51	100,72	3	0,51
i10	2343	47	2362	2366	100,81	100,98	4	0,81
Altera_oc_hdlc	2611	21	2635	2661	100,92	101,91	26	0,92
Altera_oc_ata_ocidec1	1910	17	1932	1948	101,15	101,99	16	1,15
Altera_oc_ata_vhd_3	4599	27	4657	4672	101,26	101,59	15	1,26
Altera_oc_ata_ocidec3	4648	27	4708	4725	101,29	101,66	17	1,29
Altera_nut_001	5336	125	5426	5464	101,69	102,4	38	1,69
Altera_fip_cordic_rca	2559	106	2605	2615	101,8	102,19	10	1,8
Altera_oc_rtc	1842	43	1878	1890	101,95	102,61	12	1,95
ITC_b12	942	17	961	962	102,02	102,12	1	2,02
Altera_nut_000	1511	99	1545	1564	102,25	103,51	19	2,25
Altera_os_sdram16	1692	25	1733	1750	102,42	103,43	17	2,42
Altera_oc_ata_ocidec2	2253	17	2311	2326	102,57	103,24	15	2,57
Altera_nut_003	2798	69	2871	2897	102,61	103,54	26	2,61
Altera_fip_cordic_cla	2620	99	2690	2702	102,67	103,13	12	2,67
Altera_oc_v_c_s_h_d	2248	32	2311	2335	102,8	103,87	24	2,8
Altera_nut_004	623	21	641	661	102,89	106,1	20	2,89
Altera_barrel64	3121	25	3230	3267	103,49	104,68	37	3,49

Tabulka 5.36: Kombinace permutací a randomizovaného příkazu RR - část 2.

Obvod	oA	oL	minR%	maxR%	prum+	prum-	prumC
Altera_oc_des_area_o	4128	46	1,41	0,58	100	0	1,01
c2670	690	20	0,72	0,14	60,7	5,1	0,27
s9234	1897	32	0,37	0,26	100	0	0,32
s9234,1	1902	32	0,37	0,26	100	0	0,32
bc0	1572	30	0,32	0,19	100	0	0,25
ti	1217	19	0,25	0	91,6	0	0,15
rot	1044	30	0,19	0	52,6	0	0,1
s13207,1	2661	32	0,19	0,04	100	0	0,11
Altera_oc_v_c_s_h_e	2977	30	0,1	0,71	0,6	98,7	0,32
Altera_wb_dma	4044	24	0,07	0	74,9	0	0,04
s13207	2645	33	0,04	0,11	23,1	76,9	0,04
bca	3818	26	0,03	0	48,4	0	0,01
Altera_aes_core	20794	24	0	0,04	0	100	0,02
c7552	1966	28	0	0,15	0	85,9	0,07
Altera_i2c	1145	13	0,09	0,17	0	100	0,13
Altera_oc_correlator	2862	76	0,1	0,66	0	100	0,38
Altera_oc_s_fm_rec	4530	86	0,11	0,35	0	100	0,23
cps	1601	12	0,12	0,19	0	100	0,15
pcont2	2885	62	0,14	0,42	0	100	0,23
Altera_mem	16147	32	0,19	0,27	0	100	0,23
b12	913	13	0,22	1,31	0	100	0,83
Altera_radar12	35425	155	0,24	0,4	0	100	0,32
Altera_oc_minirisc	2587	58	0,46	1,74	0	100	1,17
Mentor_1_10	1386	42	0,51	0,72	0	100	0,62
Mentor_1_09	1386	42	0,51	0,72	0	100	0,61
i10	2343	47	0,81	0,98	0	100	0,89
Altera_oc_hdlc	2611	21	0,92	1,91	0	100	1,41
Altera_oc_ata_ocidec1	1910	17	1,15	1,99	0	100	1,57
Altera_oc_ata_vhd_3	4599	27	1,26	1,59	0	100	1,43
Altera_oc_ata_ocidec3	4648	27	1,29	1,66	0	100	1,5
Altera_nut_001	5336	125	1,69	2,4	0	100	2,02
Altera_fip_cordic_rca	2559	106	1,8	2,19	0	100	1,99
Altera_oc_rtc	1842	43	1,95	2,61	0	100	2,32
ITC_b12	942	17	2,02	2,12	0	100	2,07
Altera_nut_000	1511	99	2,25	3,51	0	100	2,96
Altera_os_sdram16	1692	25	2,42	3,43	0	100	2,97
Altera_oc_ata_ocidec2	2253	17	2,57	3,24	0	100	2,93
Altera_nut_003	2798	69	2,61	3,54	0	100	3,06
Altera_fip_cordic_cla	2620	99	2,67	3,13	0	100	2,92
Altera_oc_v_c_s_h_d	2248	32	2,8	3,87	0	100	3,28
Altera_nut_004	623	21	2,89	6,1	0	100	4,8
Altera_barrel64	3121	25	3,49	4,68	0	100	4,13

Testování se provádí dle postupu uvedeného v kapitole 4.5. Jde tedy o kombinaci použití permutací částí obvodů a programové úpravy algoritmů v příkazech REWRITE, REFACTOR, RESUB a RR.

Kombinace permutací a příkazu REWRITE

U této kombinace lze pozorovat, že přidání permutací primárních vstupů a výstupů vede k ještě o něco horším výsledkům, než při použití samostatného randomizovaného příkazu REWRITE.

Kombinace permutací a příkazu REFACTOR

V případě výsledků pro tuto kombinace došlo k velmi podobnému chování jako u kombinace permutací a příkazu REWRITE. Také se dá vypozorovat mírné zhoršení v dosažených výsledcích, než při použití samostatného randomizovaného příkazu REFACTOR.

Kombinace permutací a příkazu RESUB

Kombinace permutací a randomizovaného příkazu RESUB, na rozdíl od variant s příkazy REWRITE a REFACTOR, dosáhla malého zlepšení ve výsledcích oproti použití samostatného randomizovaného příkazu RESUB.

Kombinace permutací a příkazu RR

Pro tuto kombinaci můžeme říci, že použití permutací nemělo na randomizovaný příkaz RR prakticky žádný vliv a stejných výsledků se tedy dá dosáhnout i bez použití permutací.

5.2.6 Randomizace v reálných syntézních procesech

Výstup testování se liší podle toho, jaký typ syntézního procesu se použije pro testování. Pro případ mapování na standardní buňky je sledovaným parametrem hodnota **area**, kterou jako svůj výstup produkuje systém ABC. Hodnota area značí velikost plochy obvodu po namapování na standardní buňky.

- minO - minimální dosažená hodnota area bez použití randomizace
- maxO - maximální dosažená hodnota area bez použití randomizace
- minR% - procentuální poměr minimální dosažené hodnoty area za použití randomizace vůči minimální hodnotě bez jejího použití
- maxR% - procentuální poměr maximální dosažené hodnoty area za použití randomizace vůči maximální hodnotě bez jejího použití
- % - rozdíl v procentech mezi nejlepšími dosaženými a původními hodnotami

Počet opakovaní je tentokráte 100. Zelená barva značí dosažení lepšího výsledku, zatímco červená horšího.

Tabulka 5.37: Mapování na standardní buňky

Obvod	minO	maxO	minR%	maxR%	%
apex5	2016	2691	84,03	90,64	15,97
in1	2529	2658	94,82	99,66	5,18
x3	1443	1504	95,5	100	4,5
cps	2624	2932	95,85	99,18	4,15
b2	2529	2658	95,93	100,19	4,07
c7552	3128	3290	97,41	99,06	2,59
Altera_pci_spoci_ctrl	1681	1908	97,5	99,11	2,5
t481	2111	2428	98,44	94,56	1,56
Altera_nut_002	1496	1518	98,46	99,47	1,54
ex1010	5574	5640	98,51	98,74	1,49
Altera_barrel64	4676	4795	98,52	98,81	1,48
frg1	615	708	98,7	105,37	1,3
mult16	966	978	98,76	103,27	1,24
prom2	5784	5946	98,86	99,46	1,14
ITC_b11	1049	1057	98,95	99,43	1,05
Altera_oc_ata_ocidec3	7450	7551	98,97	98,13	1,03
Altera_nut_003	3596	3658	98,97	100,41	1,03
apex6	1347	1365	99,03	99,78	0,97
Altera_oc_ata_vhd_3	7424	7475	99,38	99,3	0,62
Altera_nut_001	6467	6793	99,63	99,32	0,37
s9234	2977	3045	99,83	99,05	0,17
Altera_oc_v_c_s_h_enc	3867	3962	99,87	101,74	0,13
dsip	6145	6146	99,95	99,97	0,05
s3384	2100	2131	100	100	0
div16	1298	1315	100,31	100,46	0,31
s5378	2190	2257	100,55	99,65	0,55
Altera_oc_ata_ocidec2	3702	3708	100,65	100,65	0,65
s9234,1	2979	3036	101,14	100,79	1,14
ex5	1036	1164	101,35	101,89	1,35
max1024	1926	2009	101,77	102,09	1,77
exps	2610	2714	102,18	100,26	2,18
c6288	4245	4389	102,26	100,55	2,26
Altera_oc_ata_ocidec1	3221	3239	102,27	101,91	2,27
max512	1218	1278	102,71	100,16	2,71
alu4	6168	7046	103	101,97	3
cordic	2773	3339	103,07	99,4	3,07
apex3	3162	3561	103,38	95,96	3,38
amd	749	897	104,27	89,86	4,27
bcd	2980	3232	104,33	101,36	4,33
Altera_oc_v_c_s_h_dec	3329	3508	105,08	104,33	5,08
x7dn	740	819	106,22	96,46	6,22

Testování je provedeno dle postupu uvedeného v první části kapitoly 4.6. Jedná se tedy o opakování syntézního procesu s příkazy CHOICE a MAP.

Mapování na standardní buňky

Z výsledky lze vypozorovat, že pro některé obvody může zařazení randomizovaných variant příkazů REWRITE, REFACTOR, RESUB a RR do sytézních procesů vést ke značnému zlepšení.

Díky prvku náhodnosti můžeme dostat i horší výsledky, ale dosažené zlepšení může být natolik výrazné, že tato metoda určitě má potenciál pro reálné využití.

Pro testování druhého syntézního procesu, a to mapování na „look-up tables“ neboli LUT, je sledovaným parametrem hodnota **nd**, kterou opět jako svůj výstup produkuje systém ABC. V tomto případě hodnota nd značí počet „look-up tables“ neboli LUT, na které bude testovaný obvod namapován. Druhým sledovaným parametrem je hodnota `lev`, která udává výsledné zpoždění v obvodu po jeho namapování. Výsledná tabulka pro toto testování bude tedy vypadat takto:

- minO - minimální dosažená hodnota nd bez použití randomizace
- levO - zpoždění v obvodu při minimální dosažené hodnotě nd
- minR% - procentuální poměr minimální dosažené hodnoty nd za použití randomizace vůči minimální hodnotě bez jejího použití
- levR - zpoždění v obvodu při minimální dosažené hodnotě nd za použití randomizace
- % - rozdíl v procentech mezi nejlepšími dosaženými a původními hodnotami

Počet opakovaní je 100. Zelená barva značí dosažení lepšího výsledku, zatímco červená horšího.

Tabulka 5.38: Mapování na look-up tables neboli LUT

Obvod	minO	levO	minR%	levR	%
amd	141	5	94,33	5	5,67
b2	522	6	94,64	6	5,36
x7dn	138	5	95,65	5	4,35
cordic	522	8	96,93	8	3,07
Altera_oc_ata_ocidec2	597	4	97,82	4	2,18
Altera_oc_v_c_s_h_enc	621	6	98,07	6	1,93
s9234	588	7	98,3	7	1,7
s3384	355	16	98,31	16	1,69
t481	362	7	98,34	7	1,66
c7552	490	7	98,37	7	1,63
in1	522	6	98,85	6	1,15
max1024	349	6	98,85	6	1,15
ex5	176	4	98,86	4	1,14
Altera_barrel64	1035	6	98,94	6	1,06
apex6	211	5	99,05	5	0,95
Altera_nut_001	1115	23	99,37	21	0,63
prom2	1063	6	99,53	6	0,47
s5378	464	5	99,57	6	0,43
Altera_nut_002	351	8	99,72	8	0,28
bcd	620	7	99,84	7	0,16
ex1010	1024	7	99,9	7	0,1
g125	604	6	100	6	0
Altera_oc_ata_ocidec1	513	5	100	4	0
Altera_nut_003	641	10	100	10	0
div16	234	20	100	20	0
dsip	1100	3	100,09	3	0
Altera_oc_v_c_s_h_dec	648	8	100,15	7	0,15
s9234.1	573	7	100,17	7	0,17
Altera_pci_spoci_ctrl	320	6	100,31	6	0,31
x3	201	5	100,5	5	0,5
ITC_b11	175	6	100,57	6	0,57
alu4	1135	7	100,79	7	0,79
apex3	623	6	101,44	6	1,44
max512	206	6	101,46	6	1,46
exps	469	5	101,71	5	1,71
cps	482	5	101,87	5	1,87
mult16	183	16	102,19	14	2,19
c6288	521	25	102,3	25	2,3
Altera_oc_ata_ocidec3	1186	5	102,36	5	2,36
Altera_oc_ata_vhd_3	1184	5	102,36	5	2,36
apex5	275	5	104,36	5	4,36
frg1	104	6	106,73	5	6,73

Testování je provedeno dle postupu uvedeného v druhé části kapitoly 4.6. To znamená, že je to opakování syntézního procesu s příkazy CHOICE, IF a LUTPACK.

Mapování na look-up tables neboli LUT

Výsledky jsou velmi podobné dosaženým výsledkům pro syntézní proces mapování na standardní buňky. Při použití randomizovaných příkazů v tomto syntézním procesu můžeme dosáhnout reálného zlepšení, díky čemuž má tato metoda opět smysl i pro běžné použití.

5.2.7 Shrnutí výsledků randomizačních přístupů

Sloupce tabulky značí následující:

- prum+ - průměrný počet kladně reagujících obvodů za všech 42 obvodů (respektive 84 pro příkaz COLLAPSE), které byly testovány v rámci uvedené metody, v procentech
- prum- - průměrný počet záporně reagujících obvodů za všech 42 obvodů (respektive 84 pro příkaz COLLAPSE), které byly testovány v rámci uvedené metody, v procentech
- prumC - průměrná hodnota ze zlepšení a zhoršení obvodů za všech 42 obvodů (respektive 84 pro příkaz COLLAPSE), které byly testovány v rámci uvedené metody, v procentech

Tabulka 5.39: Průměrné výsledky pro jednotlivé metody randomizace

Metoda	prum+	prum-	prumC
BALANCE - PI	45,65	32,35	0,12
BALANCE - PO	47,94	31,91	0,15
REWRITE - PI	37,94	24,47	0,03
REWRITE - PO	54	31,41	0,33
REFACTOR - PI	24,96	12,49	0,04
REFACTOR - PO	55,4	34,12	0,2
RESUB - PI	10,3	2,55	0,03
RESUB - PO	43,09	41,88	0,05
rand. REWRITE	7,27	90,4	22,51
rand. REFACTOR	11,76	83,42	19,95
rand. RESUB	2,12	89,78	8,73
rand. RR	20,31	75,41	0,86
COLLAPSE	3,7	96,17	2575,1
rand. REWRITE - PI, PO	4,21	93,33	26,54
rand. REFACTOR - PI, PO	6,55	87,19	22,43
rand. RESUB - PI, PO	3,33	90,78	8,05
rand. RR - PI, PO	20,28	75,4	1,07

Z tohoto závěrečného shrnutí tedy plyne, že použití permutací primárních vstupů a výstupů u nerandomizovaných variant příkazů poskytuje i v průměrných hodnotách ze zlepšení a zhoršení u všech obvodů použitelné výsledky a tato varianta randomizace se tedy osvědčila.

Oproti tomu výsledky podávané randomizovanými variantami příkazů nejsou příliš použitelné, což lze vidět z průměrů dosahovaných těmito metodami. Z toho tedy můžeme usoudit, že nerandomizované varianty těchto příkazů jsou pro reálné použití lepšími variantami a při zavedení randomizace do těchto příkazů způsobíme zhoršení ve výsledcích, které tyto příkazy bez randomizace dosahují.

Kapitola 6

Závěr

Základním kamenem celé práce bylo pochopení algoritmů, které používají některé příkazy systému ABC. K tomu, aby mohlo být uvažováno o nějakém způsobu randomizace těchto příkazů v systému ABC, bylo pochopení principů jejich fungování naprosto klíčové.

Z této úvahy tedy také plyne celá rozsáhlá kapitola práce, která má za cíl popsat fungování zvolených příkazů. V dané kapitole se mi tedy doufám podařilo vnést náhled na tyto příkazy, na principy jejich fungování a na to, jak jsou v rámci systému ABC využívány. Kapitola se podrobně věnuje jednotlivým příkazům a rozepisuje to, jak přesně fungují nejenom z hlediska teoretického, ale i z hlediska programového řešení v systému ABC.

Materiály, které by prezentovaly takto podrobný popis uvedených příkazů, neexistují. Z tohoto důvodu by mohla být tato kapitola i dobrým výchozím bodem pro další studium různých úprav, které by se v těchto příkazech mohly realizovat.

Po nastudování nezbytných teoretických znalostí o zvolených příkazech byla dalším krokem snaha o jejich randomizaci. Během práce bylo prozkoumáno mnoho míst v kódech příkazů, o kterých by se dalo pro možnou randomizaci uvažovat.

Nakonec byla zvolena ta s největším předpokladem pro úspěšné provedení randomizace a hlavně také ta, která měla i předpoklad dosažení dobrých výsledků. Výsledkem je i zjištění, že některá místa vhodná k randomizaci nebyla využita a mohla by tedy sloužit jako odrazový můstek pro další práci, která by se věnovala tomuto tématu.

Takovými místy mohou být například:

- randomizace tvorby reconvergence-driven cut
- randomizace volby divisorů v příkazu RESUB

Jistě existuje daleko více míst, která by se dala randomizovat a následnými experimenty ověřit účinnost takových postupů. Další možností by také bylo vydat se mimo rámec příkazů kombinační syntézy a podívat se na další oblasti, které systém ABC podporuje.

Práce by tedy mohla sloužit nejen jako reference k některým příkazům systému ABC, ale také jako inspirace při hledání dalších randomizačních postupů v systému ABC.

V poslední části práce jsou shrnuty dosažené výsledky navrhovaných randomizačních postupů, které, jak se nakonec ukázalo, nebyly ve všech případech úspěšné. Avšak i při pohledu na méně úspěšné varianty vždy záleží na tom, jak s těmito randomizačními postupy bude naloženo a v jakém kontextu by byly použity.

Lze si třeba představit jejich využití v rámci pokročilejších syntézních skriptů, kde by randomizované varianty příkazů mohly vést k lepším dosaženým výsledkům. Jedna část práce se věnuje i tomuto tématu, avšak pouze pro jeden specifický případ, takže i zde by byla možnost pro další rozšíření a nové experimenty.

Závěrem tedy mohu konstatovat, že jsem provedl všechny nezbytné kroky vedoucí ke zjištění možností randomizace příkazů systému ABC. Cesta k tomu vedla přes studium jejich funkce, návrhy randomizací, jejich implementaci a testování. Výsledkem tohoto postupu je tato práce, která všechny kroky popisuje a vytváří určitý náhled na tuto problematiku.

Literatura

[1] ABC - A System for Sequential Synthesis and Verification, . `http://www.eecs.berkeley.edu/~alanmi/abc/`, stav z 26. 11. 2011.

[2] Berkeley Verification and Synthesis Research Center, . `http://www.bvsrc.org/`, stav z 26. 11. 2011.

[3] CUDD: CU Decision Diagram Package, . `http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html`, stav z 26. 11. 2011.

[4] Berkeley Logic Synthesis and Verification Group, ABC: A System for Sequential Synthesis and Verification, Release 20111126., . `https://bitbucket.org/alanmi/abc`, stav z 26. 11. 2011.

[5] Java SE Runtime Environment 7 Downloads, . `http://www.oracle.com/technetwork/java/index.html`, stav z 26. 11. 2011.

[6] Simple csv parser library for Java, . `http://opencsv.sourceforge.net/`, stav z 26. 11. 2011.

[7] Semináře z číslicového návrhu, . `https://edux.fit.cvut.cz/PI-SCN`, stav z 26. 11. 2011.

[8] Converting truth tables into Boolean expressions, . `http://www.allaboutcircuits.com/vol_4/chpt_7/9.html`, stav z 26. 11. 2011.

[9] ANDERSEN, H. R. An introduction to binary decision diagrams. Technical report, Course Notes on the WWW, 1997.

[10] BERTACCO, V. – DAMIANI, M. The disjunctive decomposition of logic functions. In *Proceedings of the 1997 IEEE/ACM international conference on Computer-aided design*, ICCAD '97, s. 78–82, Washington, DC, USA, 1997. IEEE Computer Society. Dostupné z: <http://dl.acm.org/citation.cfm?id=266388.266429>. ISBN 0-8186-8200-0.

[11] BRGLEZ, F. – FUJIWARA, H. A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran. In *ISCAS*, s. 663–698, June 1985. Special Session: "Recent Algorithms for Gate-Level ATPG with Fault Simulation and their Performance Assesment".

[12] BRGLEZ, F. – BRYAN, D. – KOZMINSKI, K. *Combinational profiles of sequential benchmark circuits*, 3, s. 1929–1934. 1989.

[13] CORNO, F. – REORDA, M. S. – SQUILLERO, G. RT-Level ITC'99 Benchmarks and First ATPG Results. *IEEE Des. Test*. July 2000, 17, s. 44–53. ISSN 0740-7475. doi: http://dx.doi.org/10.1109/54.867894. Dostupné z: <http://dx.doi.org/10.1109/54.867894>.

[14] EéN, N. – SöRENSSON, N. An Extensible SAT-solver. In *Theory and Applications of Satisfiability Testing*, 2919 / *Lecture Notes in Computer Science*. Sweden: Springer Berlin / Heidelberg, 2004. s. 333–336. Dostupné z: <http://dx.doi.org/10.1007/978-3-540-24605-3_37>. ISBN 978-3-540-20851-8.

[15] MARTINELLI, A. – KRENZ, R. – DUBROVA, E. Disjoint-support Boolean decomposition combining functional and structural methods. In *Proceedings of the 2004 Asia and South Pacific Design Automation Conference*, ASP-DAC '04, s. 597–599, Piscataway, NJ, USA, 2004. IEEE Press. Dostupné z: <http://dl.acm.org/citation.cfm?id=1015090.1015252>. ISBN 0-7803-8175-0.

[16] MCELVAIN, K. IWLS'93 Benchmark Set: Version 4.O distributed as part of the IWLS'93 benchmark distribution. 1993.

[17] MISHCHENKO, A. – BRAYTON, R. Scalable logic synthesis using a simple circuit structure. *IWLS '06*. 2006.

[18] MISHCHENKO, A. – CHATTERJEE, S. – BRAYTON, R. FRAIGs: A unifying representation for logic synthesis and verification. Technical report, 2005.

[19] MISHCHENKO, A. et al. An Integrated Technology Mapping Environment. In *PROC. INTERNATIONAL WORKSHOP ON LOGIC AND SYNTHESIS*, s. 383–390, 2005.

[20] MISHCHENKO, A. – CHATTERJEE, S. – BRAYTON, R. DAG-aware AIG rewriting: A fresh look at combinational logic synthesis. In *In DAC '06: Proceedings of the 43rd annual conference on Design automation*, s. 532–536. ACM Press, 2006.

[21] MISHCHENKO, A. – CHATTERJEE, S. – BRAYTON, R. Improvements to Technology Mapping for LUT-based FPGAs. In *IEEE TCAD*, s. 41–49, 2007.

[22] MISHCHENKO, A. et al. Delay optimization using SOP balancing. 2011.

[23] RUDELL, R. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design*, ICCAD '93, s. 42–47, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press. Dostupné z: <http://dl.acm.org/citation.cfm?id=259794.259802>. ISBN 0-8186-4490-7.

[24] SENTOVICH, E. et al. SIS: A System for Sequential Circuit Synthesis. Technical Report UCB/ERL M92/41, EECS Department, University of California, Berkeley, 1992. Dostupné z: <http://www.eecs.berkeley.edu/Pubs/TechRpts/1992/2010.html>.

[25] ZHANG, L. – MALIK, S. The Quest for Efficient Boolean Satisfiability Solvers. In *Proceedings of the 14th International Conference on Computer Aided Verification*, CAV '02, s. 17–36, London, UK, UK, 2002. Springer-Verlag. Dostupné z: <http://dl.acm.org/citation.cfm?id=647771.734434>. ISBN 3-540-43997-8.

Příloha A

Seznam použitých zkratek

ABC Systém ABC

AIG And-invertorový graf

BDD Binární rozhodovací diagram

SOP Sum-of-Products

BLIF Berkeley Logic Interchange Format

BAF Binary AIG format

FPGA Field-programmable gate array

LUT Lookup table

DAG Direct acyclic graph

CUDD CU Decision Diagram Package

SAT Boolean satisfiability problem

MFFC Maximum fanout free cone

JRE Java Runtime Enviroment

CSV Comma separated values

Příloha B

Tvorba AIG grafů v systému ABC

Systém ABC využívá AIG grafy jako svou základní reprezentaci obvodů. To znamená, že po načtení obvodu je nutné jej nejprve do formy AIG grafu převést. Ne všechny příkazy ABC automaticky pracují nad touto reprezentací, ale pro příkazy uvedené v této práci je forma AIG grafu naprosto stěžejní.

Pro tento převod je už nezbytně nutné začít popisovat jeho implementaci přímo v programovém kódu systému ABC. K výslednému AIG grafu vede tato sekvence příkazů:

1. příkaz READ
2. příkazy AIG a STRASH

B.1 Příkaz READ

Implementace ve zdrojovém souboru io.c v balíku base/io

Příkaz READ se stará o načtení obvodu z předem připraveného souboru. Systém ABC podporuje mnoho různých forem, jak tento soubor může vypadat, ale pro potřeby této práce je typickým představitelem formát BLIF.

Formát BLIF

Jedná se o způsob zápisu obvodů do souboru opět vyvinutý na kalifornské univerzitě v Berkeley. Zkratka BLIF znamená Berkeley Logic Interchange Format. Abychom mohli popsat, jak systém ABC nakládá s takovými soubory, je nutno stručně zmínit jejich charakteristiku. Velmi jednoduchý obvod zapsaný v BLIF formátu vypadá následovně:

```
.model example
.inputs a,b,c
.outputs d
.names a b w1
11 1
.names a b c w2
11- 1
0-1 1
.names w1 w2 d
00 0
.end
```

Značka .model vyjadřuje název obvodu, .inputs specifikuje počet vstupů a jejich názvy, .output značí počet výstupů a opět jejich názvy. Další struktury v obvodu jsou reprezentovány pomocí značky .names, kde za ní následuje seznam vstupů a výstupů do této struktury. Z ukázkového obvodu je vidět, že například řádek .names a b w1 je reprezentací dvouvstupového hradla AND se vstupy a, b a výstupem w1. Struktura .names a b c w2 obsahuje ve své specifikaci -, což značí nezávislost na daném vstupu. Celý zápis obvodu tedy značí, že je reprezentací jednoduchého multiplexeru.

Při podrobnějším pohledu řádek 11 1 u struktury .names a b w1 znamená, že vstup a je na jedničce a vstup b je také na jedničce. Při takových podmínkách bude tedy na výstupu jednička, což značí, že se jedná o hradlo AND a řádek 11 1 je vyjádřením jeho SOP formy. Formát BLIF je tedy založen na SOP zápisu reprezentujícím jednotlivá hradla.

Po zadání READ příkazu do systému ABC s parametrem v podobě BLIF souboru dojde k provedení následujícího kódu:

```
pNtk = Io_Read( fileName, Io_ReadFileType)
```

`pNtk` je vnitřní struktura systému ABC pro reprezentaci obvodu. Jakékoliv operace, které se v systému ABC budou nad obvodem provádět, se provádí nad tímto ukazatelem. Funkce `Io_ReadFileType` vrací typ souboru, který se bude načítat, například BLIF.

IO_ Read

V těle funkce `Io_Read` dochází k načítání BLIF souboru, a to pomocí exportéru, který dle známých klíčových slov BLIF soubor prochází a vytahuje z něj jednotlivé údaje jako .inputs, .outputs, .names a další možné, které vyplývají z BLIF specifikace.

Po dokončení READ příkazu je obvod v systému ABC reprezentován uzly, které odpovídají jednotlivým .names řádkům z BLIF souboru a jejich vnitřními reprezentacemi v SOP formě. Dalším krokem je vytvoření AIG grafu z této reprezentace. Tento krok se skládá ze dvou částí:

- převedení SOP uzlů na AIG uzly za pomocí příkazu AIG
- vytvoření AIG grafu, který splňuje podmínku „structural hashing“, za pomocí příkazu STRASH

B.2 Příkaz AIG

Implementace ve zdrojovém souboru abcFunc.c v balíku base/abc

Tento příkaz je uvnitř ABC reprezentován funkcí `Abc_NtkToAig`, kterou lze vyjádřit pomocí pseudokódu:

```
if ( Abc_NtkHasMapping(pNtk) )
{
  Abc_NtkMapToSop(pNtk);
    return Abc_NtkSopToAig(pNtk);
}
if ( Abc_NtkHasBdd(pNtk) )
{
  Abc_NtkBddToSop(pNtk)
    return Abc_NtkSopToAig(pNtk);
}
if ( Abc_NtkHasSop(pNtk) )
    return Abc_NtkSopToAig(pNtk);
```

Tento pseudokód vystihuje, jakým způsobem systém ABC provádí konverzi jakéhokoliv obvodu na AIG graf. **Při převodu vstupního obvodu kteréhokoliv typu na AIG graf je vždy provedena taková jeho konverze, aby uzly obvodu byly reprezentovány v SOP formě.** Z toho vyplývá, že BLIF formát je opravdu ideálním zástupcem možných reprezentací obvodů, jelikož jeho struktury už přímo používají SOP reprezentaci.

Po převedení obvodu do SOP formy dojde k zavolání funkce `Abc_NtkSopToAig`, která následně zajistí samotný převod na AIG graf.

Abc_NtkSopToAig

Při podrobnějším zkoumání této funkce narazíme na strukturu, která se dále vyskytuje v mnoha různých obdobách ve většině příkazů systému ABC. Proto tuto strukturu zmíním zde pouze jednou, jelikož její využití je v jiných příkazech velmi obdobné. Touto strukturou je manažer, v tomto případě manažer `hopMan` reprezentovaný strukturou `HopMan_t`.

AIG manažer - HopMan_t

AIG manažer je struktura, která se na pozadí stará o nově vznikající AIG graf. Funkci této struktury opravdu nejlépe vystihuje slovo manažer, jelikož zabezpečuje způsob ukládání vznikajícího grafu do paměti. Součástí této struktury jsou mnohé další ukazatele, které například uchovávají informaci o:

- počtu primárních vstupů
- počtu primárních výstupů

- počtu AND hradel
- počtu vytvořených uzlů
- počtu smazaných uzlů
- počtu úrovní grafu

Podmínkou pro to, aby AIG manažer mohl všechny tyto údaje poskytovat, je jeho schopnost starat se o jakékoliv paměťové operace spojené s tvorbou AIG grafu.

Před jeho použitím je nejprve zavolána funkce `Hop_ManStart`, která zajistí jeho inicializaci. To znamená, že se pro něj alokuje paměť a vytvoří se struktury pro ukládání všech informací nezbytných pro AIG graf. Například pole primárních vstupů a výstupů, tabulka pomocí které se poté budou jednotlivé uzly AIG grafu vyhledávat a mnohé další.

Jednotlivé uzly grafu jsou v manažeru reprezentovány jako objekt typu `Hop_Obj_t`. Vlastnosti tohoto objektu určují jeho umístění v grafu. Těmito vlastnostmi jsou:

- `pFanin0`, `pFanin1` – reprezentují uzly, které jsou vstupy daného uzlu
- `pNext` – reprezentuje uzel, pro nějž je daný uzel vstupem

Funkce tohoto konkrétního AIG manažeru se nedá úplně obecně vztáhnout na jiné manažery, ale v principu je jejich fungování dosti podobné. Hlavní úlohou jakéhokoliv manažeru v systému ABC je starat se o všechny paměťové operace a struktury, které bude jakákoliv operace využívající manažer potřebovat. Své vlastní manažery mají i všechny ostatní příkazy zmíněné v této práci.

Po inicializaci AIG manažeru je dalším krokem započetí konverze na AIG graf. To se provádí ve velkém cyklu `Abc_NtkForEachNode`, který zajišťuje průchod přes všechny uzly načtené z obvodu a konvertované do SOP formy. Pro každý uzel dochází k zavolání funkce `Abc_ConvertSopToAig`.

Abc_ ConvertSopToAig

Při konverzi jednotlivých uzlů mohou nastat dvě varianty:

1. konverze pomocí De Morganových zákonů
2. konverze za využití faktorizované formy

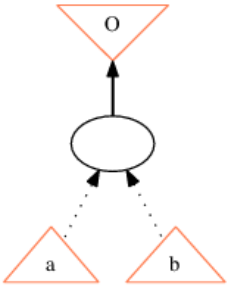
Rozhodnutí realizuje podmínka `if (fUseFactor && Abc_SopGetVarNum(pSop) > 2 && Abc_SopGetCubeNum(pSop) > 1 && !Abc_SopIsExorType(pSop))`. V této podmínce se zjišťuje, jestli má SOP forma daného uzlu více jak dva literály a alespoň dva termy.

Při splnění podmínky se spustí konverze pomocí faktorizované formy zavoláním funkce `Dec_GraphFactorSop`.

Při nesplnění podmínky se pokračuje konverzí za pomoci De Morganových zákonů voláním `Abc_ConvertSopToAigInternal`.

Abc_ ConvertSopToAigInternal

- konverze je prováděna podle De Morganových zákonů
- SOP reprezentace hradla OR 000 se převede dle De Morganova zákona $a + b = \overline{\overline{a} \cdot \overline{b}}$



Obrázek B.1: Ukázka převodu OR hradla na AIG graf

Dec_ GraphFactorSop

Při konverzi pomocí faktorizované formy je už situace daleko složitější. Tato varianta se používá, pokud má SOP reprezentace daného uzlu více jak dva literály a více jak jeden term. Z toho tedy plyne, že se jedná o postup pro všechny větší a složitěji definované uzly. Tento typ konverze se provádí pomocí techniky booleovského dělení[7].

Booleovské dělení

Booleovské dělení znamená vydělení funkce f funkcí p. Máme-li tedy funkce f a p, pak toto dělení znamená nalezení funkcí q a r takových, že $f = p \cdot q + r$, kde

- p je booleovský dělitel
- q je kvocient
- r je zbytek

Při r rovném nule se p nazývá booleovský faktor, z čehož plyne, že

- $f = p \cdot q$
- $f = p \subseteq q$
- $p = f/q$

Dále platí, že

- p existuje pokud $p \cdot f \neq \emptyset$
- q a r nejsou unikátní

Funkce `Dec_GraphFactorSop` využívá tohoto postupu k tomu, aby vytvořila faktorizovanou formu pro SOP reprezentaci uzlu, nad kterým byla zavolána.

Celkový postup při použití této metody konverze se dá shrnout takto:

- vytvoří se minimální pokrytí pro všechny termy konvertovaného uzlu
- provede se booleovské dělení, díky kterému vznikne faktorizovaná forma
- faktorizovaná forma se pomocí De Morganových zákonů upraví pro potřeby AIG grafu, což znamená převod na AND hradla a invertory

Příkladem může být uzel reprezentovaný třemi literály a dvěma termy:

```
10- 1
1-1 1
```

Tento uzel se dá vyjádřit jako booleovská funkce

$$f = a \cdot \overline{b} + a \cdot c$$

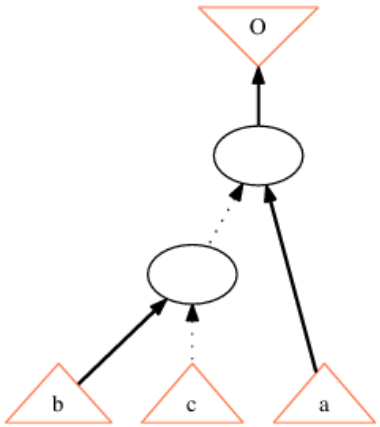
Faktorizovaná forma této funkce získaná booleovským dělením může vypadat takto

$$f = a \cdot (\overline{b} + c)$$

To lze za pomocí De Morganových zákonů upravit na

$$f = a \cdot \overline{b \cdot \overline{c}}$$

A výsledná funkce f už je lehce zobrazitelná pomocí AIG grafu



Obrázek B.2: Ukázka využití booleovského dělení při tvorbě AIG grafu

Shrnutí tvorby AIG grafu

Tvorba AIG grafu by se tedy dala jednoduše sepsat pomocí následujícího pseudokódu:

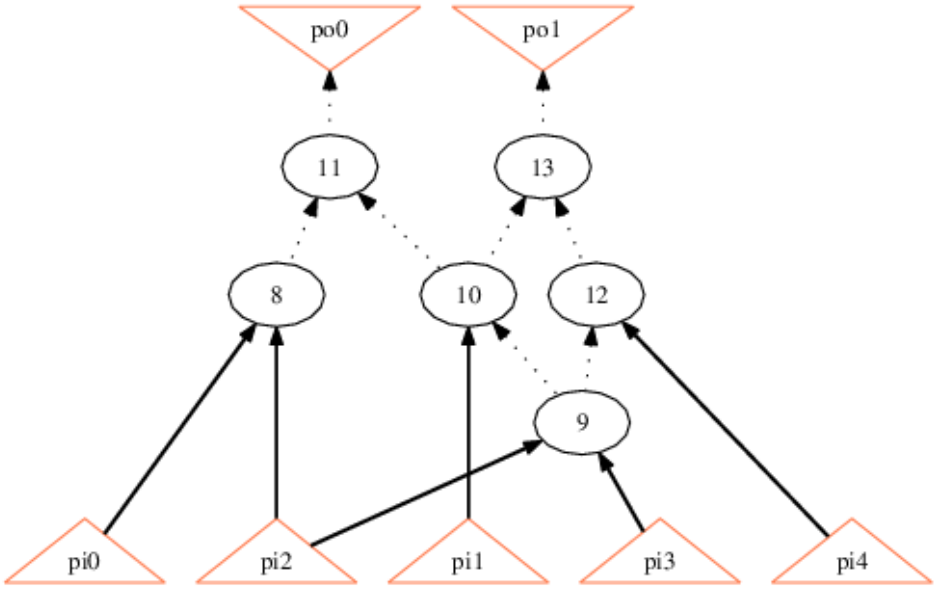
```
ConvertEverythingToSop(Ntk)
ForEachNodeIn(Ntk) {
   If (NodeIsSmall(Node))
      DeMorganTransformation(Node)
   If (NodeIsBig(Node))
      BooleanDivisionTransformation(Node)
   AddToAIGNetwork(Node)
}
```

Z pseudokódu lze vyčíst, že vznik AIG grafu probíhá vždy ze SOP formy. Poté se prochází uzel po uzlu a dle složitosti daného uzlu se zvolí jedna z metod konverze.

Závěrečným krokem při procházení uzlů je pseudokrok `AddToAIGNetwork(Node)`, který zajistí to, že nově vytvořený kus AIG grafu, který vznikl při konverzi daného SOP uzlu, se připojí do již hotového AIG grafu, který vznikl předešlými konverzemi uzlů. Tak vznikne jeden AIG graf reprezentující všechny původní SOP uzly.

Příkladem celého průchodu může být konverze následujícího obvodu

```
.model c17.blif
.inputs pi0 pi1 pi2 pi3 pi4
.outputs po0 po1
.names n7 n9 po0
0- 1
-0 1
.names n9 n10 po1
0- 1
-0 1
.names pi0 pi2 n7
0- 1
-0 1
.names pi2 pi3 n8
0- 1
-0 1
.names n8 pi1 n9
0- 1
-0 1
.names n8 pi4 n10
0- 1
-0 1
.end
```



Obrázek B.3: Vytvoření AIG grafu ze SOP reprezentace

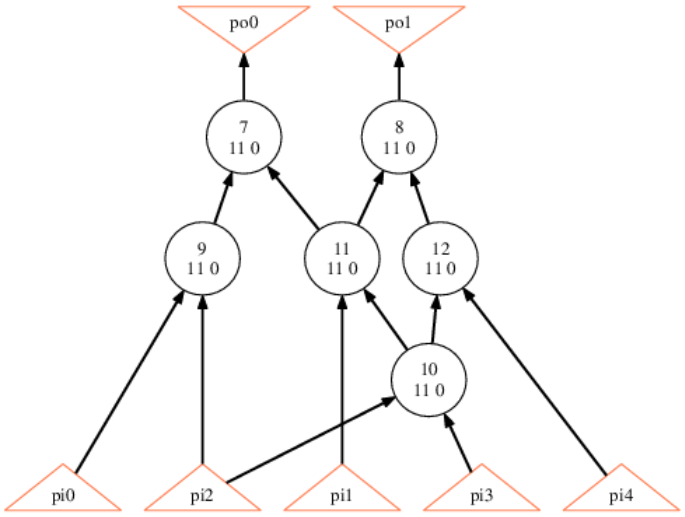
B.3 Příkaz STRASH

Implementace ve zdrojovém souboru abcStrash.c v balíku base/abci

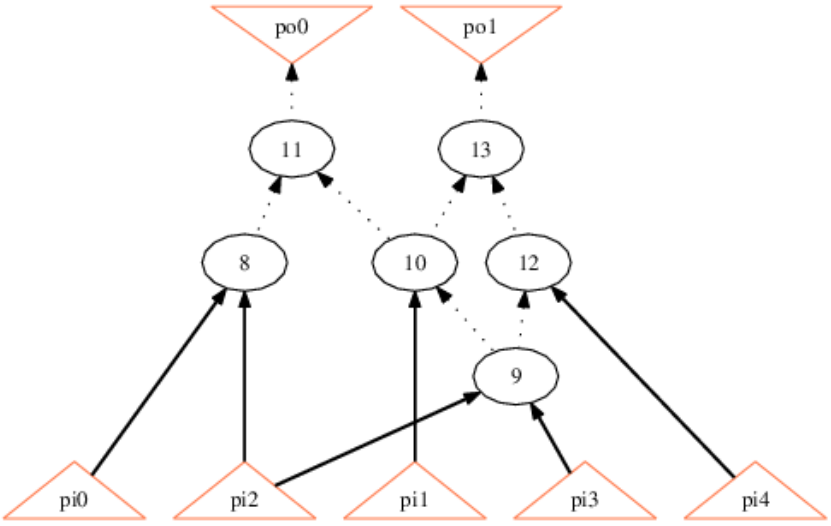
Příkaz STRASH zajišťuje:

1. propagaci všech konstant
2. splnění podmínky „structural hashing“ u AIG grafu, čímž je zajištěno, že žádné dva uzly v AIG grafu nejsou identické (staly by se také nadbytečnými)
3. převod na AIG graf s komplementárními atributy na hranách

Výsledek těchto kroků vidíme na následujícím srovnání:



Obrázek B.4: AIG graf před spuštěním příkazu STRASH



Obrázek B.5: AIG graf po spuštění příkazu STRASH

Z obrázků vyplývá, že dojde k převodu na AIG graf splňující podmínku „structural hashing“, ve kterém čárkovaně zobrazené hrany značí přítomnost komplementárního atributu na této hraně, což odpovídá přítomnosti invertoru.

Příloha C

Instalační a uživatelská příručka

Pro možnost reprodukce testování prováděné v rámci této práce je třeba nejprve zkompilovat upravené zdrojové kódy systému ABC a následně použít dvě vytvořené aplikace pro spuštění testování a tvorbu výsledných tabulek z výsledků testování.

C.1 Kompilace zdrojových souborů systému ABC

Na CD přiloženém k této práci jsou ve složce src_abc umístěny zdrojové soubory pro systém ABC, do kterých jsou zahrnuty i všechny úpravy provedené v rámci navržených metod pro randomizaci příkazů systému ABC.

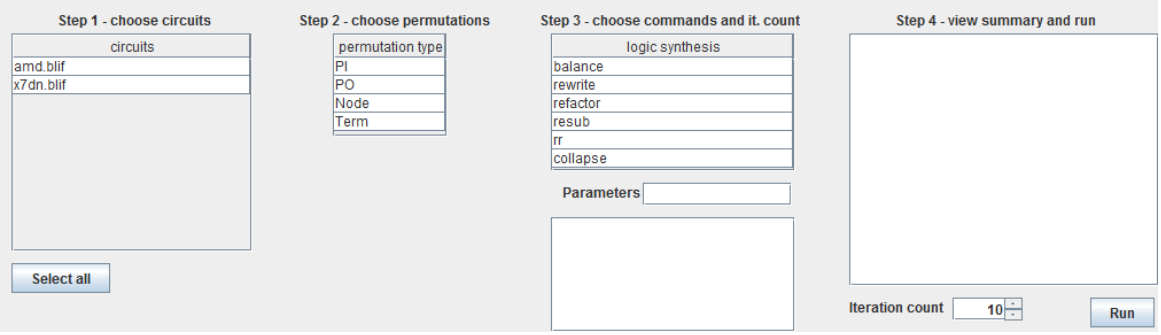
Tyto zdrojové soubory je třeba zkompilovat v prostředí Microsoft Visual Studio 2010. Postup je následující:

- v prostředí Microsoft Visual Studio 2010 otevřít soubor projektu abcspace.sln
- vybrat volbu Build Solution z menu Build

Tímto dojde k vytvoření spustitelné verze systém ABC v podobě souboru abcexe.exe, který vznikne ve složce DebugExe. Je také možné využít už zkompilované verze, která se nachází na přiloženém CD ve složce dist_abc.

C.2 Program abcScriptGen

Aplikace je vytvořena za účelem generování skriptů pro systém ABC a také umožňuje jejich spouštění. Umí také pro svou činnost využívat externího programu permute pro provádění permutací nad obvody. Aplikace je vytvořena v programovacím jazyce JAVA ve verzi 1.7 a pro svůj běh vyžaduje operační systém Microsoft Windows. Součástí aplikace je i uživatelské rozhraní.



Obrázek C.1: Aplikace abcScriptGen

Aplikace po svém spuštění načte všechny obvody, které se nacházejí ve složce `\circuits` a zobrazí je v prvním okně. V druhém okně je umožněn výběr permutací, které se budou nad vybranými obvody provádět pomocí externího programu permute. Třetí okno slouží k výběru příkazů, které se nad obvody provedou, s možností použití přepínačů a také k výběru počtu opakování všech kroků. Čtvrté okno slouží pouze jako přehled voleb před spuštěním provádění.

V případě volby použití permutací se postupuje stejně, jak je uvedeno pro případ realizace experimentů s externím programem pro permutace. Tento postup je uveden v kapitole 4.1.1. Případ bez permutací pak odpovídá kapitole 4.1.2 realizace experimentů s využitím interní permutace systému ABC.

Spuštění programu se provádí pomocí dávkového souboru run.bat. Před spuštěním je nutno mít korektně nainstalováno JRE ve verzi minimálně 1.7[5]. Dalším nezbytným krokem je manuální vytvoření tří složek ve složce, kde bude program spuštěn.

- abc - Adresář do kterého se nakopíruje zkompilovaná distribuce systému ABC v podobě souboru abc.exe a také soubor abc.rc, ve kterém jsou definovány podoby skriptů, které systém ABC využívá (nachází se na přiloženém CD ve složce dist_abc).
- circuits - Adresář, do kterého je třeba nakopírovat zdrojové soubory obvodů, se kterými bude systém ABC pracovat.
- permute - Adresář, do kterého se nakopíruje externí program pro provádění permutací permute.exe (nachází se na přiloženém CD ve složce dist_abcScripGen).

Při úspěšném spuštění aplikace jsou vytvořeny následující adresáře:

- circuitsorigstats - Adresář, do kterého budou ukládány statistiky obvodů před jejich zpracováním. Vzniká zde soubor ve formátu CSV s názvem origStats.
- errorlogs - Zde vznikají dva soubory reprezentující chyby, které mohou nastat při provádění příkazů v systému ABC nebo při provádění permutací pomocí externího programu permute. Jedná se o soubory abcError a permError.
- executionlog - Adresář, do kterého se zapisuje soubor, který obsahuje zprávu o průběhu provádění příkazů v systému ABC.

- permute - Adresář, do kterého se nakopíruje externí program pro provádění permutací permute.
- permuted - Adresář pro tvorbu dočasných permutací obvodů.
- results - Adresář pro ukládání výsledků provádění.
- scripts - Adresář, ve kterém jsou uloženy skripty, které řídí provádění příkazů v systému ABC.

Aplikace produkuje svůj výstup do souboru ve formátu CSV.

C.3 Program resultsParser

Tento program je stejně jako aplikace abcScriptGen napsán v programovacím jazyce JAVA a pro svůj běh vyžaduje operační systém Microsoft Windows. Jeho hlavním úkolem je zpracovávat tabulky ve formátu CSV, které produkuje program abcScriptGen. Výsledným produktem jsou tabulky odpovídající těm, které jsou uvedeny v této práci. Pro rychlé zpracovávání souborů ve formátu CSV je v rámci programu využívána volně dostupná knihovna OpenCSV[6].

Spuštění programu je realizováno pomocí dávkového souboru run.bat. Před spuštěním je opět nutno mít korektně nainstalováno JRE ve verzi minimálně 1.7[5]. Nutným krokem pro spuštění je manuální vytvoření tří složek ve složce, kde bude program spuštěn.

- data - Adresář, ve kterém jsou umístěny soubory s výsledky programu abcScriptGen ve formátu CSV.
- circuitsorigStats - Adresář, ve kterém je soubor origStats, který reprezentuje původní statistiky souborů před provedením příkazů v systému ABC. Jako zdroj tohoto souboru se používá soubor origStats z aplikace abcScriptGen, který vzniká při její činnosti.
- results - Adresář pro ukládání výsledných tabulek ve formátu CSV.

Tato aplikace nemá žádné GUI a její činnost spočívá v tom, že automaticky načte všechny soubory z adresáře data a ty následně zpracuje.

C.4 Celkový postup pro zprovoznění

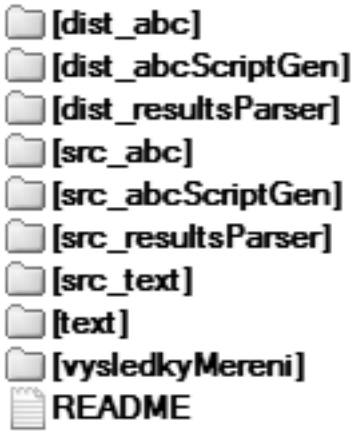
Postup pro provedení všech kroků vedoucích od přípravy pro testování až po vytvoření výsledných CSV souborů se dá shrnout takto:

1. Zkopírovat složky dist_abc, dist_abcScriptGen, dist_resultsParser na pevný disk počítače.
2. Ve složce dist_abcScriptGen vytvořit složky abc, circuits, permute.
3. Ve složce dist_resultsParser vytvořit složky data, circuitsorigStats, results.

4. Do složky abc ve složce dist_abcScriptGen nakopírovat soubory abc.exe a abc.rc ze složky dist_abc.
5. Do složky permute ve složce dist_abcScriptGen nakopírovat soubor permute.exe ze složky dist_abcScriptGen.
6. Do složky circuits ve složce dist_abcScriptGen nahrát soubory testovaných obvodů ve formátu BLIF.
7. Spustit program abcScriptGen pomocí dávkového souboru run.bat ze složky dist_abcScriptGen.
8. Nastavit prováděné testování v uživatelském rozhraní programu abcScriptGen a spustit testování tlačítkem Run.
9. Po dokončení testování zkopírovat soubory ze složky results ve složce dist_abcScriptGen do složky data ve složce dist_resultsParser.
10. Zkopírovat soubor origStats.csv ze složky circuitsOrigStats ve složce dist_abcScriptGen do složky circuitsOrigStats ve složce dist_resultsParser.
11. Spustit program resultsParser pomocí dávkového souboru run.bat ze složky dist_resultsParser.
12. Výsledné soubory ve formátu CSV se vytvoří ve složce results, která se nachází ve složce dist_resultsParser.

Příloha D

Obsah přiloženého CD



Obrázek D.1: Struktura přiloženého CD

- dist_abc - Adresář se zkompilovanou verzí systému ABC s úpravami provedenými v rámci této práce.
- dist_abcScriptGen - Adresář se zkompilovanou verzí aplikace abcScriptGen.
- src_resultsParser - Adresář se zkompilovanou verzí aplikace resultsParser.
- src_abc - Adresář se zdrojovými soubory systému ABC s úpravami provedenými v rámci této práce.
- src_abcScriptGen - Adresář se zdrojovými soubory aplikace abcScriptGen.
- src_resultsParser - Adresář se zdrojovými soubory aplikace resultsParser.
- src_text - Adresář se zdrojovými soubory textu této práce.
- text - Adresář s textem této práce ve formátu PDF.
- vysledkyMereni - Adresář s naměřenými výsledky ve formátu CSV.