České vysoké učení technické v Praze
Fakulta elektrotechnická

Diplomová práce

Převod víceúrovňové logiky na dvouúrovňovou

Jaroslav Bartoník

Vedoucí práce: Ing. Petr Fišer

Studijní program: Elektrotechnika a informatika

Obor: Informatika a výpočetní technika

leden 2008

Poděkování

Chtěl bych poděkovat slečně MUDr. Alexandře Breuerové za trpělivost, kterou se mnou musela po celou dobu psaní této diplomové práce mít, a také mým rodičům za podporu po celou dobu studia. Dále bych chtěl poděkovat Ing. Petru Fišerovi za vedení a připomínky k této diplomové práci.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu \S 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 18.1.2008 ……..

Abstract

This work engages in a transformation of a multi-level logic network into a two-level logic network in sum of products form. Two methods are here implemented. They are based on recursive steps through circuit from output to input and their efectivity is compared. As the input format describing circuit is used BENCH format, as the output format of circuit transformed in two-level logic network is PLA format. A comparison is made with a method that uses structures called Binary Decision Diagram (BDD), also with another solution based on identical principle as my solution, and also with programs SIS and MVSIS.

Abstrakt

Práce se zabývá převodem víceúrovňové logické sítě na dvouúrovňovou ve tvaru součtu součinů (SOP). Jsou zde implementovány dvě metody převodu založené na rekurzivním procházení obvodu od výstupu směrem ke vstupům a porovnána jejich efektivita. Vstupním formátem popisujícím obvod je formát BENCH, výstupním formátem obvodu převedeného na dvouúrovňovou logickou síť je formát PLA. Srovnání je provedeno jednak s přístupem založeným na binárních rozhodovacích diagramech (BDD), tak i s dalším řešením založeným na stejném principu jako moje řešení, a také s programy SIS a MVSIS.

Obsah

Seznam obrázků **xi**

Seznam tabulek **xiii**

1 Úvod **1**
1.1 Cíl práce ... 1
1.2 Základní pojmy ... 1

2 Popis řešení **3**
2.1 Metoda přímého převodu ... 3
2.1.1 Jednoduchý příklad průchodu a převodu ... 3
2.2 Metoda and-or ... 4
2.2.1 Jednoduchý příklad průchodu a převodu ... 4
2.3 Popis metody založené na BDD ... 6

3 Analýza a návrh řešení
3.1 Formáty vstupního a výstupního souboru ... 9
3.1.1 Formát BENCH ... 9
3.1.2 Formát PLA ... 11
3.2 Načtení obvodu ze vstupního souboru do datové struktury ... 12
3.2.1 Metoda přímého převodu ... 12
3.2.2 Metoda and-or ... 13
3.3 Procházení obvodem a tvorba jednotlivých hradel ... 14
3.3.1 Datová struktura pro uložení součtu součinů ... 14
3.3.2 Tvorba hradel pomocí metody přímého převodu ... 15
3.3.3 Tvorba hradel pomocí metody and-or ... 16
3.4 Minimalizace ... 17
3.4.1 Absorpce ... 17
3.4.2 Absorpce negace ... 18
3.4.3 Idempotence ... 18
3.4.4 Vyloučení třetího ... 18
3.4.5 Minimalizace při vytváření součinu ... 18
3.4.6 Minimalizace externím programem ... 20

4 Realizace
4.1 Metoda přímého převodu ... 21
4.1.1 Načtení vstupního souboru ... 21
4.1.2 Implementace datových struktur ... 22
4.1.3 Převod víceúrovňové logiky na dvouúrovňovou ... 23
4.2 Metoda and-or ... 25
4.2.1 Načtení vstupního souboru ... 25
4.2.2 Implementace datových struktur ... 25
4.2.3 Převod víceúrovňové logiky na dvouúrovňovou ... 26

5 Testování
5.1 Metoda přímého převodu 30
5.1.1 Minimalizace vytvořených součinů 30
5.1.2 Minimalizace vytvářených součinů 31
5.1.3 Minimalizace při procházení obvodem 32
5.1.4 Ukládání opakujících se hradel 33
5.2 Metoda and-or 34
5.2.1 Porovnání metody and-or a metody přímého převodu 34
5.2.2 Nové řešení minimalizace vytvořených součinů 36
5.2.3 Nové řešení převodu hradla typu „and“ 37
5.2.4 Vylepšená metoda and-or 38
5.2.5 Vliv minimalizace vytvořených součinů na metodu and-or 39
5.2.6 Vliv minimalizace vytvářených součinů na metodu and-or 40
5.2.7 Vliv ukládání opakujících se hradel na metodu and-or 41
5.2.8 Minimalizace pomocí externího programu Espresso 42
5.3 Porovnání s jinými existujícími řešeními 43

6 Závěr **45**

7 Seznam literatury **47**

A Uživatelská / instalační příručka **49**

B Obsah přiloženého CD **51**

Seznam obrázků

2.1 Příklad obvodu pro převod pomocí metody přímého převodu 3
2.2 Příklad obvodu pro převod pomocí metody and-or 4
2.3 Obvod po převedení hradla A 4
2.4 Obvod po převedení hradla B 5
2.5 Obvod po převedení hradla C 5
2.6 Obvod po převedení hradla D 5
2.7 Obvod po převedení hradla E 6
2.8 Příklad binárního rozhodovacího diagramu 7
3.1 Datová struktura pro metodu přímého převodu 12
3.2 Datová struktura pro metodu and-or 13
3.3 Příklad uložení obvodu do datové struktury pro metodu and-or 14
3.4 Datová struktura pro uložení součtu součinů 14
3.5 Příklad uložení součtu součinů do datové struktury 15
5.1 Vliv vynechání absorpce u metody přímého převodu 31
5.2 Vliv vynechání minimalizace při procházení obvodem u metody přímého převodu 33
5.3 Vliv vynechání ukládání opakujících se hradel u metody přímého převodu 34
5.4 Porovnání metody and-or oproti metodě přímého převodu 35
5.5 Vliv nového řešení minimalizace vytvořených součinů u metody and-or 36
5.6 Vliv nového řešení převodu hradla typu „and“ u metody and-or 37
5.7 Rozdíl mezi původní a vylepšenou metodou and-or 38
5.8 Vliv minimalizace vytvořených součinů na metodu and-or 39
5.9 Vliv minimalizace vytvářených součinů na metodu and-or 40
5.10 Vliv ukládání opakujících se hradel na metodu and-or 41

Seznam tabulek

5.1 Přehled testovacích benchmarků 29
5.2 Metoda přímého převodu s použitím všech minimalizačních metod 30
5.3 Metoda přímého převodu bez minimalizace vytvořených součinů 31
5.4 Metoda přímého převodu bez minimalizace při vytváření součinu 32
5.5 Metoda přímého převodu bez minimalizace při procházení obvodem 32
5.6 Metoda přímého převodu bez opakování hradel 33
5.7 Metoda and-or 34
5.8 Výsledky měření metody přímého převodu a metody and-or 35
5.9 Metoda and.or s novým řešením minimalizace vytvořených součinů 36
5.10 Metoda and-or s novým řešením převodu hradla typu „and“ 37
5.11 Vylepšená metoda and-or 38
5.12 Metoda and-or bez minimalizace vytvořených součinů 39
5.13 Metoda and-or bez minimalizace vytvářených součinů 40
5.14 Metoda and-or bez ukládání opakujících se hradel 41
5.15 Vliv minimalizace pomocí externího programu Espresso 42
5.16 Porovnání výsledného počtu termů pro různá řešení 43
5.17 Porovnání výsledného času pro různá řešení 44

1 Úvod

1.1 Cíl práce

Úkolem této diplomové práce bylo navrhnout a realizovat program pro převod víceúrovňové logické sítě na dvouúrovňovou, neboli collapsingem, a vyzkoušet různé alternativy přístupu k danému problému. Collapsing má vlastně za úkol zredukovat velké množství logických operací z původního obvodu na jednoduché zobrazení ohodnocení vstupních proměnných na hodnotu výstupních proměnných. Toto zobrazení je ve tvaru součtu součinů, tzn. že vstupní proměnné jsou propojeny pouze s hradly typu „and“, jejichž výstupy jsou propojeny na hradlo typu „or“ a výstup tohoto hradla je již výstupem celého obvodu. Jelikož u rozsáhlejších logických obvodů vzrůstá obtížnost převodu, je třeba řešení navrhnout co nejefektivněji.
Vyzkoušel jsem dvě metody, založené na principu rekurzivního procházení stromem hradel směrem od výstupu až ke vstupům. Při zpětném průchodu se pak provádí převod a minimalizace jednotlivých hradel na tvar součtu součinů, vytvořený pouze ze vstupních proměnných.
Řešení bude porovnáno s přístupem založeným na binárních rozhodovacích diagramech (BDD) a s dalším řešením založeným na podobném principu jako moje řešení, a také s programy SIS a MVSIS.

1.2 Základní pojmy

Zde vysvětlím pár základních pojmů, které budu v dalších kapitolách používat:

term – složen z logických proměnných, které jsou spolu vázány jedním logickým operátorem
součin – term obsahující operátor součinu (např. $x \cdot y \cdot z$)
součet součinů – součiny vázané operátorem součtu (např. $(a \cdot b \cdot c) + (x \cdot y \cdot z)$)
negované hradlo – hradlo typu „nand“, „nor“, „not“, nebo „xnor“
binární rozhodovací diagramy – grafové struktury určené pro efektivní manipulaci s logickými funkcemi

2 Popis řešení

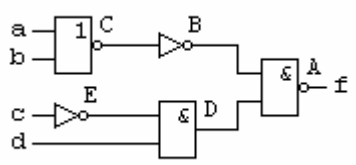
K řešení problému collapsingu jsem navrhnul dvě metody řešení. Obě metody jsou založeny na rekurzivním průchodu obvodu směrem od výstupů ke vstupům a převodu jednotlivých hradel na tvar součtu součinů. První metodu jsem nazval jako metodu přímého převodu, druhou jako metodu and-or.
Dané metody si obecně popíšeme v kapitolách 2.1 a 2.2, v kapitole 2.3 pak metodu založenou na binárních rozhodovacích diagramech, se kterou budu moje metody porovnávat.
Analýzu řešení provedu v kapitole 3, programovou realizaci v kapitole 4.
Nejdůležitějším kritériem pro porovnání kvality metod bude čas, za který zvládne metoda zadání vyřešit. Druhým, též důležitým kritériem, bude počet termů výsledného obvodu. Výsledky testování kvality metod budou popsány v kapitole 5.

2.1 Metoda přímého převodu

Obvodem se rekurzivně projde od výstupu až ke vstupům obvodu, přičemž při zpětném průchodu směrem k výstupu obvodu se pro každé hradlo provádí převod na tvar součtu součinů, sestávající se pouze ze vstupních proměnných.
Pro názornost uvedu jednoduchý příklad.

2.1.1 Jednoduchý příklad průchodu a převodu

Mějme následující obvod:



Obrázek 2.1: Příklad obvodu pro převod pomocí metody přímého převodu

Obvod projdeme rekurzivně od výstupu ke vstupům, tj. od hradla A, přes hradla B a C až ke vstupům *a* a *b*. Nyní lze hradlo C převést na součet součinů:

$$C = \overline{a + b} = \bar{a}\bar{b}$$

Jelikož hradlo B má jen jeden vstup, lze ho již převést:

$$B = \bar{C} = a + b$$

Nyní se vrátíme k hradlu A, které je ovšem dvouvstupé, a tudíž je třeba znát i tvar součtu součinů pro druhý vstup. Přes hradlo D a E se dostaneme až ke vstupu *c*. Nyní hradlo E:

$$E = \bar{c}$$

Druhý vstup hradla D je vstupem obvodu, tudíž můžeme převést hradlo D:

$$D = E \cdot d = \bar{c}d$$

Oba vstupy hradla A již máme převedeny, lze tedy převést hradlo A a získat výsledný tvar převedeného obvodu ve tvaru součtu součinů:

$$f = A = \overline{B \cdot D} = \overline{(a+b) \cdot (\bar{c}d)} = \bar{a}\bar{b} + c + \bar{d}$$

2.2 Metoda and-or

Jako v předchozí metodě se obvodem rekurzivně projde od výstupu až ke vstupům obvodu a při zpětném průchodu směrem k výstupu obvodu se pro každé hradlo provede převod na tvar součtu součinů, sestávající se pouze ze vstupních proměnných.
Rozdíl ovšem spočívá v tom, že se obvod již při průchodu od výstupu ke vstupům převede pomocí de Morganových zákonů [2] na obvod tvořený pouze hradly typu „and“ a „or“:

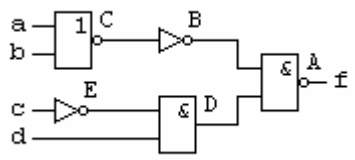
$$\overline{x.y} = \bar{x} + \bar{y}$$

$$\overline{x + y} = \bar{x}.\bar{y}$$

S negacemi vstupů hradel převedených de Morganovými zákony se bude počítat při převodu všech dalších hradel, jejichž výstup vede na vstup převedeného hradla.

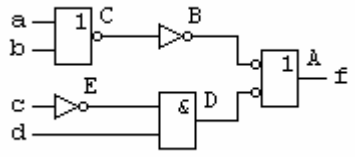
2.2.1 Jednoduchý příklad průchodu a převodu

Mějme stejný obvod jako u předchozí metody:



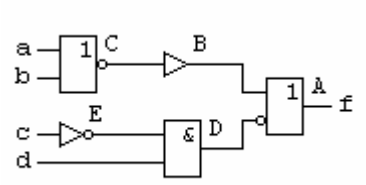
Obrázek 2.2: Příklad obvodu pro převod pomocí metody and-or

Prvním hradlem od výstupu směrem ke vstupům je hradlo A typu „nand“, které lze de Morganovým zákonem převést na hradlo typu „or“ se znegovanými vstupy:



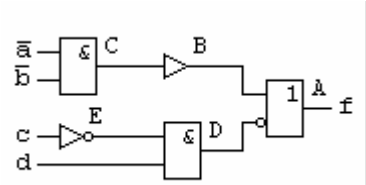
Obrázek 2.3: Obvod po převedení hradla A

Hradlo B je typu „not“ a jelikož hradlo A má negované vstupy, stane se z hradla B hradlo typu „buf“:



Obrázek 2.4: Obvod po převedení hradla B

Hradlo C typu „nor“ se de Morganovým zákonem převede na hradlo typu „and“ se znegovanými vstupy:



Obrázek 2.5: Obvod po převedení hradla C

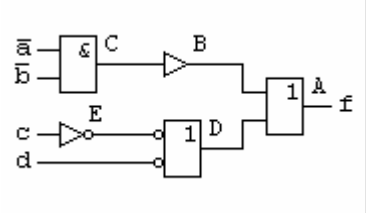
Jelikož vstupy hradla C jsou vstupy celého obvodu, lze ho již převést:

$$C = \bar{a}\bar{b}$$

Hradlo B je typu „buf“, tudíž není třeba nic převádět:

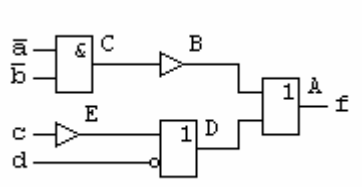
$$B = C = \bar{a}\bar{b}$$

Vrátíme se k hradlu A, to ale nemá ještě převedené všechny vstupy, proto pokračujeme převodem hradla D. To je typu „and“, ale díky znegovanému vstupu hradla A se z něj stane hradlo typu „nand“, které se převede de Morganovým zákonem na hradlo typu „or“ se znegovanými vstupy:



Obrázek 2.6: Obvod po převedení hradla D

Z hradla E typu „not“ se díky znegovanému vstupu hradla D stane hradlo typu „buf“:



Obrázek 2.7: Obvod po převedení hradla E

Vstup hradla E tvoří vstup celého obvodu, tudíž:

$$E = c$$

Hradlo D má nyní na jednom vstupu převedené hradlo E a na druhém vstupu vstup celého obvodu, tudíž lze převést:

$$D = E + \overline{d} = c + \overline{d}$$

Vstupy hradla A jsou již všechny převedeny, a proto lze převést hradlo A a získat výsledný obvod ve tvaru součtu součinů:

$$f = A = B + D = \overline{a}\overline{b} + c + \overline{d}$$

2.3 Popis metody založené na BDD

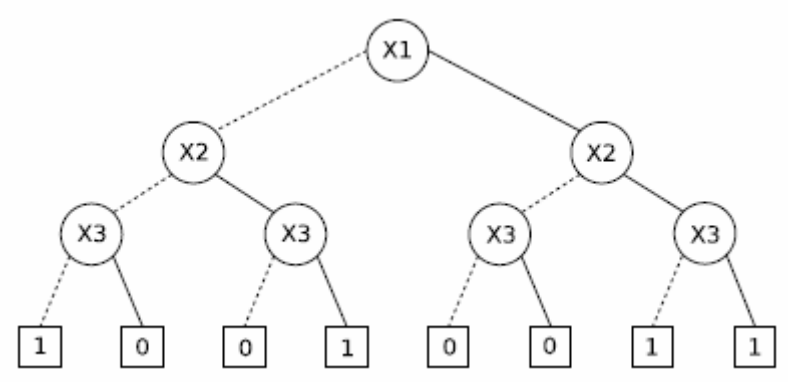
V této kapitole popíši řešení pomocí metody založené na binárních rozhodovacích diagramech (Binary Decision Diagram), které bylo zrealizováno v diplomové práci Přemysla Ruckého [3] a se kterým budu mé řešení porovnávat. Převzato z [3].

BDD je grafová struktura navržená k rychlému vyhodnocování logických funkcí. Má tvar orientovaného acyklického kořenového grafu. Jednotlivé uzly odpovídají iteracím Shannonova teorému o rozkladu, který říká, že každou logickou funkci lze rozložit podle následujícího vzorce:

$$f = (a, \overline{a}, b, \overline{b}, ...) = a \cdot f(1,0,b,\overline{b},...) + \overline{a} \cdot f(0,1,b,\overline{b},...)$$

Převedeno do lidské řeči to znamená, že každou funkci můžeme rozdělit na dvě. Před první funkci vytkneme její pozitivní reprezentaci a ve funkci její výskyty nahradíme jedničkou. Negované potom nahradíme nulou. To samé, pouze s negovanou formou, uděláme s druhou funkcí.
Do struktury samotného BDD se to promítne následovně. Každý vnitřní uzel (neterminál) grafu má 2 následníky (tj. rozklad funkce). Jeden odpovídá funkci, kde proměnná, kterou daný uzel reprezentuje, nabývá hodnoty ”1” a druhý odpovídá funkci, kde proměnná nabývá hodnoty ”0”. Pro přehlednost budu označovat vnitřní uzly jmény proměnných, na které se v dané iteraci aplikoval Shannonův teorém. Obrázek 2.8 ukazuje příklad BDD pro funkci
$f = \overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3} + x_1 \cdot x_2 + x_2 \cdot x_3$, kde čárkovaná čára značí hodnotu proměnné ”0” a plná čára hodnotu proměnné ”1”. Listy uzlu (terminály) mohou nabývat hodnot ”1” nebo ”0” a určují hodnotu funkce pro takové ohodnocení proměnných, které vzniklo průchodem grafu od jeho

kořene až do listu. Jako příklad je možno uvést první list zleva na obrázku 2.8, který má hodnotu ”1”. Je vidět, že odpovídá hodnotě funkce pro ohodnocení proměnných $x_1 = x_2 = x_3 = 0$. Z kořene grafu jsme totiž postupovali vždy po hraně označující nulové ohodnocení dané proměnné. Pohledem na zadanou funkci zjistíme, že tento výsledek je správný, protože při nulovém ohodnocení všech tří proměnných bude mít první term logickou hodnotu ”1”, tudíž je hodnota celé funkce také ”1”.



Obrázek 2.8: Příklad binárního rozhodovacího diagramu

3 Analýza a návrh řešení

Jako programovací jazyk pro vývoj řešení byl zvolen jazyk C++ [9],[10], neboť práce je založena na výukovém systému EDA vyvíjeném na katedře počítačů [1].
Problém byl rozdělen na následující podproblémy, které se mohly lišit v závislosti na metodě řešení collapsingu:

- zvolení formátu vstupního a výstupního souboru
- načtení obvodu ze vstupního souboru do datové struktury
- procházení obvodem a tvorba jednotlivých hradel
- minimalizace vytvářených termů
- výpis hotového obvodu do výstupního souboru

3.1 Formáty vstupního a výstupního souboru

Pro popis logických obvodů můžeme využít spoustu formátů, jako např. *bench*, *blif*, *pla*, *dbg*, *edif*, *kiss*, nebo *vhdl*. Jako vstupní formát byl zvolen formát *bench*, neboť je velmi jednoduchý a jak uvidíme dále, jeho formát je ve formě potřebné pro průchod obvodu od výstupu směrem ke vstupům.Výstupním formátem bude formát *pla*, který je téměř totožný s datovou strukturou reprezentující výsledný obvod.

3.1.1 Formát BENCH

Soubor ve formátu *bench* může obsahovat následující typy řádků:

- # komentář – za znakem # může být umístěn komentář
- INPUT(*x*) – definuje vstupní proměnnou celého obvodu se jménem *x*
- OUTPUT(*y*) – definuje výstupní proměnnou celého obvodu se jménem *y*
- *f* = AND(*a,b*) – definuje hradlo v logickém obvodu; *a*,*b* jsou vstupy hradla, které označují buď vstupní proměnnou celého obvodu nebo výstup jiného hradla; *f* je výstup hradla, pokud je hradlo propojeno s výstupem celého obvodu, tak se označuje jako výstupní proměnná celého obvodu

Z formátu *bench* je patrné, že je velmi jednoduchý na pochopení logického obvodu. V definicích hradel se mohou vyskytnout následující typy hradel:

- AND – logický součin; může mít více vstupních proměnných
- OR – logický součet; může mít více vstupních proměnných
- NAND – negovaný logický součin; může mít více vstupních proměnných
- NOR - negovaný logický součet; může mít více vstupních proměnných

- BUF – identita; jen jedna vstupní proměnná
- NOT – negace; jen jedna vstupní proměnná
- XOR – nonekvivalence; může mít více vstupních proměnných
- XNOR – ekvivalence; může mít více vstupních proměnných

Zde je příklad jednoduchého obvodu c17:

```
# c17 iscas example (to test conversion program only)
# -----------------------------------------------
#
#
#   total number of lines in the netlist ..............    17
#   simplistically reduced equivalent fault set size =     22
#         lines from primary input  gates .......     5
#         lines from primary output gates .......     2
#         lines from interior gate outputs ......     4
#         lines from **     3 ** fanout stems ...     6
#
#         avg_fanin  =  2.00,     max_fanin  =  2
#         avg_fanout =  2.00,     max_fanout =  2
#
#
#
#
#
INPUT(G1gat)
INPUT(G2gat)
INPUT(G3gat)
INPUT(G6gat)
INPUT(G7gat)
OUTPUT(G22gat)
OUTPUT(G23gat)

G10gat = nand(G1gat, G3gat)
G11gat = nand(G3gat, G6gat)
G16gat = nand(G2gat, G11gat)
G19gat = nand(G11gat, G7gat)
G22gat = nand(G10gat, G16gat)
G23gat = nand(G16gat, G19gat)
```

Z definice obvodu c17 vidíme, že má pět vstupů a dva výstupy a obsahuje šest dvouvstupých hradel typu „nand“.

3.1.2 Formát PLA

Formát *pla* obsahuje klíčová slova, popisující daný obvod. Zde jsou vyjmenovány ty, které jsou nejdůležitější a pro nás dostačující:

- *.model jméno* – jméno popisovaného obvodu
- *.i počet* – počet vstupních proměnných
- *.o počet* – počet výstupních proměnných
- *.ilb* i_1 i_2 i_3 … i_n – seznam jmen vstupních proměnných
- *.ob* o_1 o_2 o_3 … o_n – seznam jmen výstupních proměnných
- *.p počet* – celkový počet termů
- *.e* – konec souboru

Kromě výše uvedených klíčových slov obsahuje formát *pla* popis jednotlivých termů v součinovém tvaru. Každý řádek reprezentuje jeden součin. Nejprve je na řádku popsán vektor ohodnocení jednotlivých proměnných v termu, kde proměnné mohou nabývat hodnot ”0”,”1” a ”-”. Ohodnocení ”1” znamená, že se daná proměnná v daném termu vyskytuje v přímé formě, ohodnocení ”0” znamená, že se vyskytuje v negované formě a ohodnocení ”-” znamená, že se daná proměnná v daném termu nevyskytuje. Poté následuje, po oddělení oddělovačem, vektor ohodnocení výstupních funkcí. Zde ohodnocení ”1” značí hodnotu logické funkce ”1” a ohodnocení ”0” znamená, že daná výstupní proměnná nemá pro danou výstupní funkci význam.

Příklad souboru ve formátu *pla* si opět ukážeme na obvodu c17, převedeném do *pla*:

```
.model c17
.i 5
.o 2
.ilb G1gat G2gat G3gat G6gat G7gat
.ob G22gat G23gat
.p 7
1-1-- 10
-10-- 10
-1-0- 10
-10-- 01
-1-0- 01
--0-1 01
---01 01
.e
```

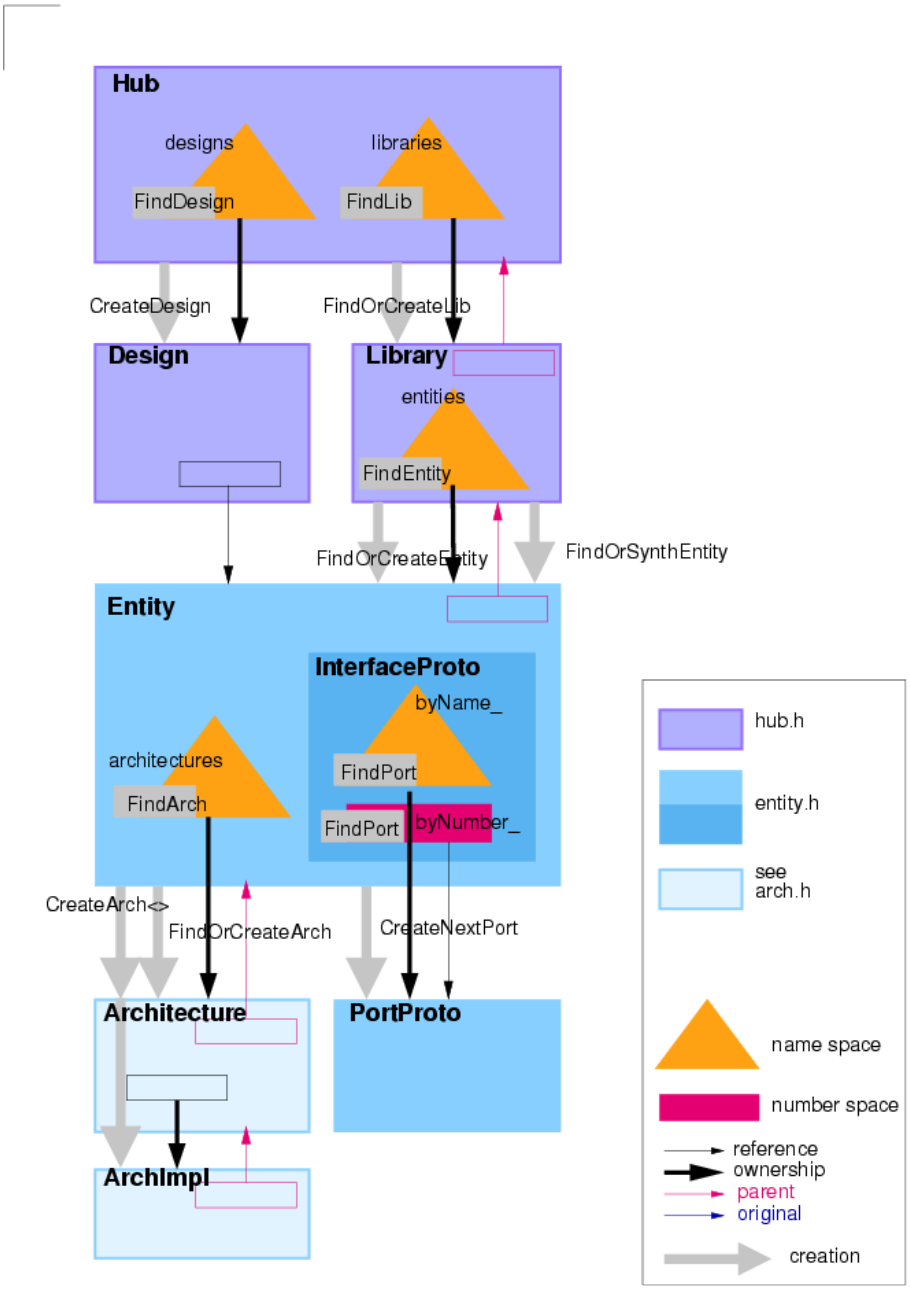
Z výše uvedeného přikladu vidíme, že obvod má 5 vstupů, dva výstupy a obsahuje 7 termů: první tři definují výstup G22gat, další čtyři pak výstup G23gat.

3.2 Načtení obvodu ze vstupního souboru do datové struktury

V obou případech jsem pro načtení vstupního souboru ve formátu bench využil lexikální analyzátor z projektu HUB [1].

3.2.1 Metoda přímého převodu

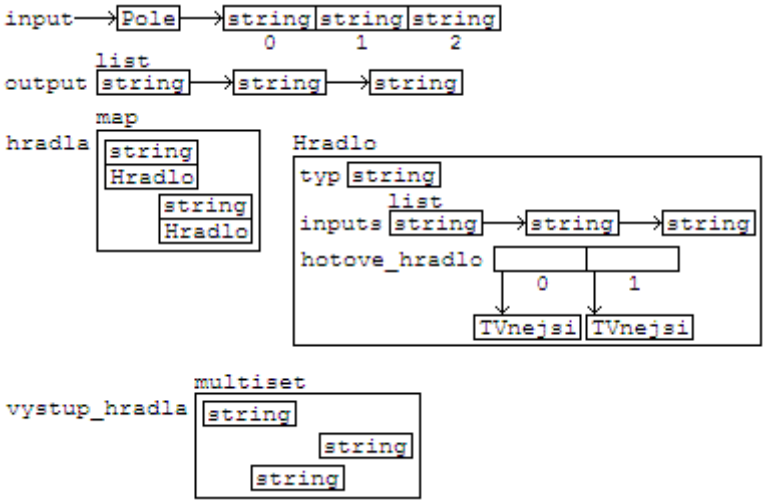
Při použití této metody byla použita třída BenchParser z projektu HUB [1], pomocí které byly data ze vstupního souboru načítány do datové struktury Hub, viz. obr. 3.1.



Obrázek 3.1: Datová struktura pro metodu přímého převodu, převzato z [1]

3.2.2 Metoda and-or

V této metodě jsem vstupní data načítal do mnou definované struktury, která je co do paměťové tak hlavně i časové náročnosti průchodu strukturou mnohem výkonnější. To si ukážeme v kapitole 5.2.1.
Vstupní proměnné se postupně v pořadí jak jsou v souboru zapsané za sebou zapisují do třídy reprezentující pole, které samo dokáže měnit svou velikost v závislosti na počtu proměnných. Vstupní proměnné pak budou reprezentovány pouze indexem v tomto poli, což umožní jejich mnohem rychlejší porovnávání a přístup.
Výstupní proměnné lze uložit do klasického seznamu ze standardní knihovny, neboť stačí pouze jeden průchod celým seznamem ke zjištění všech výstupů a jejich dvouúrovňové minimalizaci.
Pro uložení definic hradel jsem zvolil mapu ze standardní knihovny, jejíž klíčem je název hradla a datovým typem struktura popisující dané hradlo. Ta obsahuje typ hradla („and“, „nand“, „or“, „nor“, „buf“, „not“, „xor“, „xnor“), seznam vstupů jdoucích do hradla a pole dvou ukazatelů na typ TVnejsi, do něhož se ukládá výsledná podoba převedeného hradla v případě, že výstup z hradla vede do více hradel. Tím můžeme ušetřit čas, neboť dané hradlo nemusíme znovu převádět. Pole obsahuje dva prvky, protože záleží na tom, či počet negací směrem k výstupní proměnné je lichý nebo sudý. Datový typ TVnejsi si popíšeme v kapitole 3.3.1.
Poslední položkou je multimnožina obsahující názvy vstupů na které je propojen výstup hradla, podle které zjistíme zda výstup z jednoho hradla vede na více vstupů jiných hradel.
Výsledná struktura bude vypadat takto:

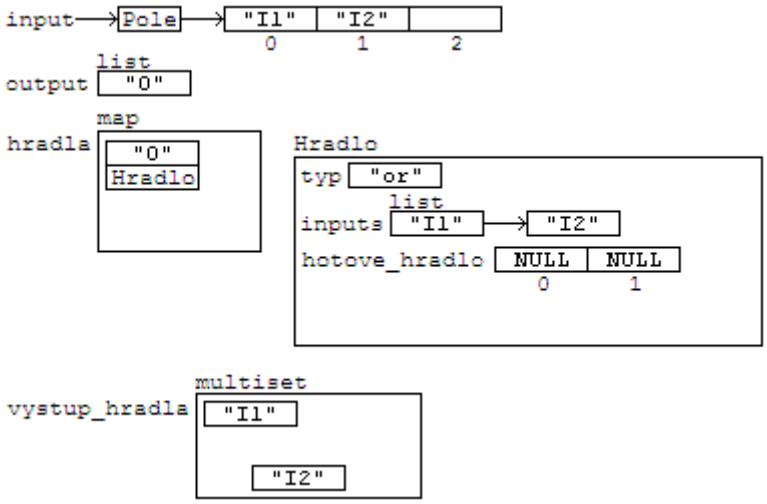


Obrázek 3.2: Datová struktura pro metodu and-or

Mějme následující jednoduchý obvod:

```
INPUT(I1)
INPUT(I2)
OUTPUT(O)
O = or(I1,I2)
```

Datová struktura popisující tento jednoduchý obvod bude vypadat takto:



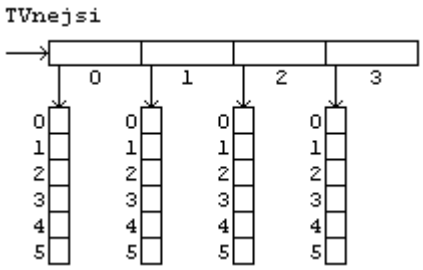
Obrázek 3.3: Příklad uložení obvodu do datové struktury pro metodu and-or

3.3 Procházení obvodem a tvorba jednotlivých hradel

Obě metody jsou založeny na rekurzivním průchodu obvodu směrem od výstupu ke vstupům. Pokud jde na vstup hradla výstup jiného hradla, provede se další rekurzivní vnoření. V případě, že vstupem hradla je vstup celého obvodu, další vnoření se neprovede a vrátí se hodnota vstupu ve tvaru součtu součinů.
Jakmile jsou všechny vstupy hradla převedeny na součet součinů, je třeba podle typu hradla převést výstupní funkci hradla též na součet součinů.

3.3.1 Datová struktura pro uložení součtu součinů

Jako datovou strukturu pro uložení výsledku jednotlivých dvouúrovňově minimalizovaných hradel jsem zvolil ukazatel na pole ukazatelů na pole typů char:



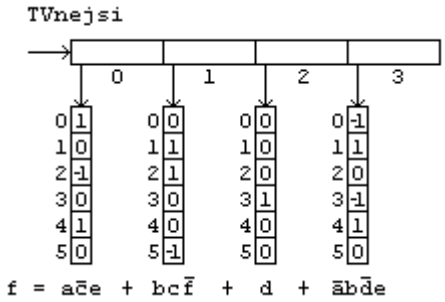
Obrázek 3.4: Datová struktura pro uložení součtu součinů

Velikost pole ukazatelů je závislá na počtu součinů a velikost pole typů char na počtu vstupních proměnných celého obvodu. Index v poli typů char odpovídá indexu v poli vstupních proměnných z obrázku 3.2.

Jednotlivé prvky součinu mohou nabývat těchto hodnot:

- -1 pokud je v součinu proměnná negovaná
- 0 pokud proměnná v součinu není
- 1 pokud proměnná v součinu je

Následující obrázek ukazuje, jak by vypadal zápis součtu součinů do dané struktury v případě, že vstupní proměnné budou načteny podle abecedy:



Obrázek 3.5: Příklad uložení součtu součinů do datové struktury

Součiny by šly realizovat též jako seznam nebo množina. Seznam by měl výhodu v tom, že když chci zjistit všechny proměnné v daném součinu, tak by prošel pouze ty dané proměnné a ne všechny proměnné jako v případě pole. Naopak nevýhoda seznamu je při hledání dané proměnné v součinu. Co se množiny týče, tak ta má podobné vlastnosti jako seznam, jen při vkládání je množina pomalejší a při hledání rychlejší než seznam, neboť prvky třídí a ukládá do setříděného stromu. Nevýhodou seznamu i množiny ze standardní knihovny je jejich větší časová složitost.

3.3.2 Tvorba hradel pomocí metody přímého převodu

U této metody se pomocí zákonů Booleovy algebry[2] převede výstup každého hradla na odpovídající součet součinů. V následujících kapitolách budu vždy uvažovat, že chci převést hradlo o dvou vstupech, které jsou již převedeny na tvar součtu součinů. Jedním vstupem bude *(ab+cd)*, druhým *(ef+gh)*. Nyní si ukážeme, jak bude vypadat převod pro jednotlivé typy hradel.

Pro převod hradla typu „and“, majícího vstupy ve tvaru součtu součinů využijeme distributivní zákon, podle kterého:

$$(ab+cd).(ef+gh) = abef + abgh + cdef + cdgh$$

Z výše uvedeného vyplývá, že vždy roznásobíme každý součin jednoho vstupu s každým součinem dalšího vstupu.

V případě hradla typu „or“ stačí pouze sloučit všechny součiny ze všech vstupů do jednoho součinu:

$$(ab+cd)+(ef+gh) = ab + cd + ef + gh$$

K převodu hradla typu „nand“ použijeme jednak de Morganovy zákony a také již zmiňovaný distributivní zákon:

$$\overline{(ab+cd).(ef+gh)} = \overline{ab+cd} + \overline{ef+gh} = \overline{ab}.\overline{cd} + \overline{ef}.\overline{gh} = (\bar{a}+\bar{b}).(\bar{c}+\bar{d}) + (\bar{e}+\bar{f}).(\bar{g}+\bar{h}) =$$
$$= \bar{a}\bar{c} + \bar{a}\bar{d} + \bar{b}\bar{c} + \bar{b}\bar{d} + \bar{e}\bar{g} + \bar{e}\bar{h} + \bar{f}\bar{g} + \bar{f}\bar{h}$$

Z výsledku je vidět, že pro převod hradla typu „nand“ musíme mezi sebou vynásobit jednotlivé proměnné z každého součinu pro každý vstup hradla a pak každou proměnnou znegovat.

U hradla typu „nor“ podobně jako u hradla typu „and“ použijeme de Morganovy zákony a distributivní zákon:

$$\overline{(ab+cd)+(ef+gh)} = \overline{ab}.\overline{cd}.\overline{ef}.\overline{gh} = (\bar{a}+\bar{b}).(\bar{c}+\bar{d}).(\bar{e}+\bar{f}).(\bar{g}+\bar{h}) =$$
$$= \bar{a}\bar{c}\bar{e}\bar{g} + \bar{a}\bar{c}\bar{e}\bar{h} + \bar{a}\bar{c}\bar{f}\bar{g} + \bar{a}\bar{c}\bar{f}\bar{h} + \bar{a}\bar{d}\bar{e}\bar{g} + \bar{a}\bar{d}\bar{e}\bar{h} + \bar{a}\bar{d}\bar{f}\bar{g} + \bar{a}\bar{d}\bar{f}\bar{h} +$$
$$+ \bar{b}\bar{c}\bar{e}\bar{g} + \bar{b}\bar{c}\bar{e}\bar{h} + \bar{b}\bar{c}\bar{f}\bar{g} + \bar{b}\bar{c}\bar{f}\bar{h} + \bar{b}\bar{d}\bar{e}\bar{g} + \bar{b}\bar{d}\bar{e}\bar{h} + \bar{b}\bar{d}\bar{f}\bar{g} + \bar{b}\bar{d}\bar{f}\bar{h}$$

V tomto případě je třeba mezi sebou roznásobit každou proměnnou z každého součinu ze všech vstupů hradla typu „nor“ a proměnnou znegovat.

U hradla typu „buf“ nemusíme žádný převod provádět, neboť jen předává vstup na výstup.

Hradlo typu „not“ se převede stejně jako hradlo typu „nor“ s jedním vstupem.

Funkce hradla typu „xor“ je taková, že se na výstupu objeví logická jednička v případě, že si nejsou všechny hodnoty na vstupech rovny. To lze zapsat jako:

$$x \oplus y \oplus z = (x + y + z).(\bar{x} + \bar{y} + \bar{z})$$

Tudíž pro převod hradla „xor“ využijeme předchozích převodů pro hradla „not“, „or“ a „and“.

Hradlo typu „xnor“ má přesně opačnou funkci jako hradlo „xor“, tudíž lze zapsat takto:

$$\overline{x \oplus y \oplus z} = x.y.z + \bar{x}.\bar{y}.\bar{z}$$

Opět pro převod využijeme předchozích převodů pro hradla „not“, „or“ a „and“.

3.3.3 Tvorba hradel pomocí metody and-or

U předchozí metody je nejsložitější převod hradel typu „nand“, „nor“ a „not“, a tudíž i hradel

typu „xor“ a „xnor“, které pro převod využívají hradlo typu „not“. Abychom dosáhli lepšího výsledku, musíme co nejvíc eliminovat negované hradla. Toho dosáhneme postupnou aplikací de Morganových zákonů, kdy se bude negace hradla postupně propadávat od výstupu blíže ke vstupům obvodu.

Hradla typu „and“, „or“, „buf“, „xor“, „xnor“ se převedou se stejně jako v kapitole 3.3.2.

Hradlo typu „nand“ se převede pomocí de Morganova zákona:

$$\overline{x.y} = \bar{x} + \bar{y}$$

To znamená, že z hradla „nand“ se stane hradlo „or“, které se převede jako v kapitole 3.3.2. Hradla, jejichž výstup byl vstupem do tohoto hradla, se znegují:

- z hradla typu „and“ se stane hradlo typu „nand“ a naopak
- z hradla typu „or“ se stane hradlo typu „nor“ a naopak
- z hradla typu „buf“ se stane hradlo typu „not“ a naopak
- z hradla typu „xor“ se stane hradlo typu „xnor“ a naopak

Také hradlo typu „nor“ se převede pomocí de Morganova zákona:

$$\overline{x + y} = \bar{x}.\bar{y}$$

Z hradla typu „nor“ se stane hradlo typu „and“, které se převede jako v kapitole 3.3.2. Hradla, jejichž výstup byl vstupem do tohoto hradla, se opět znegují.

U hradla typu „not“ se pouze provede výše zmiňované znegování vstupu.

3.4 Minimalizace

Pokud převedu hradla podle výše zmíněných metod, může výsledné převedené hradlo obsahovat velký počet součinů, které se dají zrušit, neboť jsou např. ve výsledku obsaženy vícekrát, nebo se dají určitými logickými úpravami zrušit. Proto budu už při převádění hradel používat následující minimalizace[2].

3.4.1 Absorpce

Zákon absorpce zní:

$$x + xy = x$$

To znamená, že pokud při vytváření součinu zjistím, že jeho proměnné jsou nadmnožinou resp. podmnožinou již vytvořeného součinu, tak vytvářený resp. již vytvořený součin zruším.

3.4.2 Absorpce negace

Zákon absorpce negace zní:

$$x + \bar{x}y = x + y$$

Tudíž pokud naleznu dva součiny, které mají tu vlastnost, že právě jedna proměnná je v jednom součinu negovaná a v druhém nenegovaná a zároveň ostatní proměnné jednoho součinu jsou podmnožinou ostatních proměnných druhého součinu, tak odeberu proměnnou, která byla v obou součinech různá, ze součinu, který byl nadmnožinou druhého součinu.

3.4.3 Idempotence

Zákon idempotence:

$$x + x = x$$

V případě, ze budou oba součiny stejné, jeden z nich zruším.

3.4.4 Vyloučení třetího

$$xy + x\bar{y} = x$$

Jestliže se budou dva součiny lišit pouze v jedné proměnné, lze z jednoho součinu tuto proměnnou zrušit a druhý součin zrušit celý.

3.4.5 Minimalizace při vytváření součinu

Použití předchozích metod bylo vždy až po vytvoření nového součinu, tj. nově vytvořený součin se porovnával s již vytvořenými součiny.
Následující metody mají za úkol již při vytváření součinů minimalizovat jejich počet, tj. aby se nadbytečné součiny vůbec nevytvářely.

Stejný součin:

Mějme např. následující hradlo typu „and“:

$$(ab + cd) \cdot (ab + ce) \cdot (b + de)$$

Po jeho úplném roznásobení dostaneme 8 součinů:

$$(ab + abde + abce + abcde + abcd + abcde + bcde + cde)$$

Což po úplné minimalizaci dá 2 součiny:

$$(ab + cde)$$

Už při roznásobování lze zjistit, že se určité součiny budou dát minimalizovat. V prvním kroku mezi sebou násobím součiny *ab-ab-b*. Když zjistím, že součin, který mám násobit je podmnožinou právě vytvářeného součinu, můžu podle zákonu absorpce vyloučit kroky, při kterých násobím součiny ze stejného vstupu hradla.
Př.: V prvním kroku násobím součiny *ab-ab-*.. a zjistím, že součin *ab* je podmnožinou součinu *ab*, a tudíž můžu vyloučit kroky *ab-ce-*... V prvním kroku dále násobím *ab-ab-b*, ale součin *b* je podmnožinou součinu *ab*, tudíž můžu vyloučit krok *ab-ab-de*. V druhém kroku tedy násobím *cd-ab-b*, součin *b* je podmnožinou součinu *ab*, krok *cd-ab-de* lze vyloučit. V třetím kroku násobím *cd-ce-b* a ve čtvrtém *cd-ce-de*, součin *de* je podmnožinou součinů *cd* a *ce*, tudíž všechny další součiny *cd-ce-*.. by se daly vyloučit.
Po roznásobení tedy místo 8 součinů dostanu součiny 4:

$$(ab + abcd + bcde + cde)$$

Stejná proměnná:

Máme toto hradlo typu „nor“:

$$\overline{(ab + ac) + (cd)}$$

Po jeho převedení dostaneme 8 součinů:

$$(\bar{a}\bar{c} + \bar{a}\bar{d} + \bar{a}\bar{c} + \bar{a}\bar{c}\bar{d} + \bar{a}\bar{b}\bar{c} + \bar{a}\bar{b}\bar{d} + \bar{b}\bar{c} + \bar{b}\bar{c}\bar{d})$$

Což po úplné minimalizaci dá 3 součiny:

$$(\bar{a}\bar{c} + \bar{a}\bar{d} + \bar{b}\bar{c})$$

Obdobně jako u metody stejného součinu lze i zde již v průběhu vytváření součinů určité kroky vynechat. Postup bude stejný jako v předchozí kapitole, jen místo násobení součinů budu mezi sebou násobit jednotlivé proměnné součinů.
Př.: V prvním kroku násobím proměnné *a-a-*.. a zjistím, že proměnná *a* je podmnožinou proměnné *a*, tudíž krok *a-c-*.. můžu vyloučit. V prvním kroku tedy roznásobím proměnné *a-a-c* a v druhém *a-a-d*. Ve třetím, resp. čtvrtém kroku roznásobím proměnné *b-a-c* resp. *b-a-d*. V pátém kroku roznásobím *b-c-c*, ale jelikož *c* je podmnožinou *b-c*, lze další krok *b-c-d* vyloučit.
Po převedení tedy místo 8 součinů dostanu součinů 5:

$$(\bar{a}\bar{c} + \bar{a}\bar{d} + \bar{a}\bar{b}\bar{c} + \bar{a}\bar{b}\bar{d} + \bar{b}\bar{c})$$

3.4.6 Minimalizace externím programem

Minimalizace za použití všech předchozích metod nepřevede vždy hradlo na minimální formu. Proto pro úplnou minimalizaci použiji externí program Espreso[5]. Ten použiji až po převedení pomocí mých metod.

4 Realizace

V této kapitole se zaměříme na samotnou implementaci řešení v programovacím jazyku C++. Nejprve si ukážeme jednotlivé kroky implementace pomocí metody přímého převodu a poté pomocí metody and-or.

4.1 Metoda přímého převodu

U této metody jsem pro uložení víceúrovňového obvodu použil datovou strukturu `Hub` z projektu HUB [1].

4.1.1 Načtení vstupního souboru

Nejprve je třeba načíst vstupní soubor do datové struktury. K tomu jsem využil již hotovou třídu `BenchImport` z projektu HUB [1], která obsahuje instanci třídy `BenchParser`.
Třída `BenchParser` obsahuje instanci třídy `LexanBench`, která provádí lexikální analýzu. Hlavní metodou je metoda `run()`, která načte soubor a vytvoří datovou strukturu. Postupně volá metody `inputs()` pro načtení a tvorbu vstupů obvodu, `outputs()` pro načtení a tvorbu výstupů obvodu a `assigns()` pro načtení a tvorbu hradel, ze kterých se obvod skládá.

```
class BenchParser {
    BenchParserClient* client_;

    void inputs();
    void outputs();
    void assigns();
    void hradlo(list<std::string> *);
    void idents(list<std::string> *);
    void r_o_idents(list<std::string> *);

public:
    LexanBench l;
    int token;

    void run (const std::string&);
    BenchParser(BenchParserClient* client): client_(client) {}
    ~BenchParser() {}
};
```

4.1.2 Implementace datových struktur

Pro uložení hradel převedených na součet součinů slouží třída `THradlo`. Počet takovýchto hradel je uložen v proměnné `poc_vst`. Jednotlivá hradla jsou uloženy v proměnné `hradlo`, což je pole ukazatelů na `TVnejsi`.

```
class THradlo {
   public:
     TVnejsi **hradlo;
     int poc_vst;
     THradlo(int pocet_vstupu);
     ~THradlo();
};
```

Třída `TVnejsi` v sobě uchovává součet jednotlivých součinů, jejich počet je uložen v proměnné `clenu`. Součiny jsou uloženy v poli ukazatelů na `TVnitrni`. Třída má dva konstruktory, jeden když je a druhý když není známo kolik součinů bude uloženo. Jelikož při vytváření nové instance nemusí být známo, kolik součinů bude součet obsahovat, obsahuje třída `TVnejsi` metodu `Zvetsi()`, pomocí které lze pole pro uložení součinů zdvojnásobit.

```
class TVnejsi {
   public:
     TVnitrni **vnejsi;
     int clenu;
     TVnejsi();
     TVnejsi(int clen);
     ~TVnejsi();
     void Zvetsi();
};
```

Třída `TVnitrni`, která obsahuje jeden součin vstupních proměnných, je implementována jako pole typu `char`. Pole má velikost počtu vstupních proměnných celého obvodu, kde každý `char` reprezentuje jednu vstupní proměnnou a může nabývat následujících hodnot:

- 0 – vstupní proměnná se v daném součinu nevyskytuje
- 1 – vstupní proměnná se v daném součinu vyskytuje
- -1 – vstupní proměnná se v daném součinu vyskytuje negovaná

```
class TVnitrni {
   public:
char *vnitrni2;
     TVnitrni(int input);
     ~TVnitrni();
};
```

Následující třída `THotove_hradlo` slouží k uložení hradel převedených na součet součinů, jejichž výstup je propojen na více než jeden vstup jiných hradel. To ušetří čas, neboť se hradlo při opětovném požadavku na převod nebude muset znovu převádět. Proměnná `pocet` vyjadřuje, kolikrát je dané hradlo propojeno na vstup jiného hradla.

```
class THotove_hradlo {
  public:
     int pocet;
     TVnejsi *hradlo;
};
```

4.1.3 Převod víceúrovňové logiky na dvouúrovňovou

Převod víceúrovňové logiky na dvouúrovňovou realizuje třída `Collapsing`. Ta obsahuje proměnnou `INPUT`, určující počet vstupů celého obvodu, proměnnou `vstupy`, což je pole názvů jednotlivých vstupů, a mapu `seznam_hradel`, sloužící k uložení opakujících se hradel.

Hlavní metodou třídy je metoda `minimalize()`, která nejprve projde všechny vstupy obvodu a jejich názvy zapíše do proměnné `vstupy` a poté postupně na každý výstup obvodu volá rekurzivní funkci `minim_hradlo()` pro převod hradla z víceúrovňového na dvouúrovňové. Výsledný obvod uloží do souboru ve formátu `pla`. Další funkce `my_and()`, `my_or()`, `my_nor()`, `my_nand()`, `my_not()`, `absorpce()` a `minim()` jsou volány při převodu hradla z funkce `minim_hradlo()`.

```
class Collapsing {
     int INPUT;
     char *arg[5];  // argumenty prikazove radky
     string *vstupy;
     map<std::string,THotove_hradlo> seznam_hradel;
     TVnejsi* my_and(THradlo* hr);
     TVnejsi* my_or(THradlo* hr);
     TVnejsi* my_nor(THradlo* hr);
     TVnejsi* my_nand(THradlo* hr);
     TVnejsi* my_not(TVnejsi* vn);
     TVnejsi* minim_hradlo(std::string hradlo,
                    hub::StructArch* sa, bool negace = false);
     void absorpce(TVnejsi* nove, int* novy_soucin);
     TVnejsi* minim(TVnejsi* nove);
  public:
     Collapsing(int ac, char *a[]);
     void minimalize(hub::Hub theHub);
};
```

Funkce `minim_hradlo()` rekurzivně prochází obvodem od výstupu ke vstupům a postupně převádí každé hradlo na součet součinů. Převod hradla probíhá tak, že se nejprve

zjistí, či vstupy hradla jsou vstupy celého obvodu. Pokud ano, zapíší se jako vstupy hradla, pokud ne, zavolá se funkce `minim_hradlo()` na hradlo, jehož výstup je vstupem daného hradla.

V případě, že výstup převáděného hradla je propojen s více než jedním vstupem jiných hradel, tak se hradlo uloží resp. načte ze seznamu hotových hradel.

Pokud natrefím na hradlo typu „not“, zjistím, či hradlo, jehož výstup je propojen se vstupem hradla typu „not“, není typu „nand“, „nor“, „not“ či „xnor“. Pakliže je, tak podle zákona dvojí negace, vznikne z těchto dvou hradel jedno hradlo typu „and“, „or“, „buf“ či „xor“.

Nyní následuje nejdůležitější část, a to převod jednotlivých hradel na tvar součtu součinů. Ten je realizován funkcemi `my_and()`, `my_or()`, `my_nor()`, `my_nand()` a `my_not()`. Podle kapitoly 3.3.2 se jednotlivá hradla převedou následovně:

- „and“ – vynásobí se mezi sebou každý součin ze vstupu hradla s každým součinem z dalšího vstupu hradla
- „or“ – sloučí se všechny součiny ze všech vstupů hradla do jednoho součinu
- „nand“ – vynásobí se mezi sebou každá proměnná z každého součinu pro každý vstup hradla a pak se každá proměnná zneguje
- „nor“ – vynásobí se mezi sebou každá proměnná z každého součinu pro všechny vstupy hradla a proměnná se zneguje
- „not“ - vynásobí se mezi sebou každá proměnná z každého součinu pro jeden vstup hradla a proměnná se zneguje

Při roznásobování se podle typu hradla aplikuje pravidlo o stejném součinu nebo o stejné proměnné z kapitoly 3.4.5:

- „and“ – stejný součin
- „nand“ – stejná proměnná
- „nor“ – stejná proměnná
- „not“ – stejná proměnná

Po vytvoření každého nového součinu se zavolá metoda `absorpce()`, která zjistí, či nově vytvořený součin nelze minimalizovat s některým z již vytvořených součinů. Metoda testuje tyto minimalizační pravidla z kapitoly 3.4: absorpce, absorpce negace, idempotence.

Hradla typu „xor“ resp. „ xnor“ se podle kapitoly 3.3.2 převedou pomocí výše uvedených metod pro převod hradel „and“, „or“ a „not“.

Posledním krokem je minimalizace pomocí externího programu Espreso funkcí `minim()`.

4.2 Metoda and-or

V této metodě jsem pro uložení víceúrovňového obvodu použil svou vlastní datovou strukturu.

4.2.1 Načtení vstupního souboru

K načtení vstupního souboru do datové struktury jsem využil třídu `BenchParser` z projektu HUB [1], viz. kapitola 4.1.1. Upravil jsem ale její metody `inputs()`, `outpus()` a `assigns()` tak, aby se data ukládala do mé vlastní struktury.

4.2.2 Implementace datových struktur

Vlastní datovou strukturu pro uložení načteného souboru jsem deklaroval ve třídě `BenchParserClient`.

```
class BenchParserClient {
public:
    Pole *input;
    list<std::string> output;
    map<std::string,Hradlo> hradla;
    multiset<std::string> vystup_hradla;
    BenchParserClient();
    virtual ~BenchParserClient ();
};
```

Proměnná `output` obsahuje seznam názvů výstupů obvodu. Proměnná `vystup_hradla` je multimnožina a udává, na kolik hradel je propojen výstup daného hradla. V proměnné `hradla` jsou pomocí mapy uloženy definice hradel. K tomu slouží struktura `Hradlo`. Ta obsahuje proměnnou `typ`, která uchovává typ hradla („and“, „or“, „nand“, „nor“, „buf“, „not“, „xor“, „xnor“). Další položkou je `inputs`, což je seznam názvů hradel, které jsou propojeny na vstupy daného hradla. Poslední položkou je proměnná `hotove_hradlo`, sloužící k uložení již převedeného hradla ve tvaru součtu součinů. Ta musí mít dvě hodnoty, neboť při převodu obvodu jen na hradla typu „and“ a „or“ záleží, či je počet negací mezi výstupem hradla a výstupem celého obvodu lichý nebo sudý. Tudíž pokud se hradlo v obvodu vyskytuje na více místech, může mít převedený tvar hradla dva různé výsledky.

```
struct Hradlo {
     std::string typ;
     list<std::string> inputs;
     TVnejsi *hotove_hradlo[2];
};
```

Třída `Pole` slouží pro uchování názvů vstupních proměnných. Má metodu `Vloz()` pro vložení názvu vstupu, pokud nedostačuje velikost pole, automaticky se velikost zdvojnásobí

pomocí metody `Zvetsi()`.

```
class Pole {
     int velikost;
     void Zvetsi();
     int index;
     std::string *p;
public:
     Pole();
     ~Pole();
     void Vloz(std::string co);
};
```

Pro uložení hradel převedených na součet součinů jsem použil stejnou reprezentaci dat jako v metodě přímého převodu, a to třídy `THradlo`, `TVnejsi` a `TVnitrni`, viz. kapitola 4.1.2. Pouze třída `TVnejsi` má navíc proměnné `pocet_norm` a `pocet_neg` pro uchování počtu normálních a negovaných proměnných v součinu.

4.2.3 Převod víceúrovňové logiky na dvouúrovňovou

Převod víceúrovňové logiky na dvouúrovňovou realizuje třída `Collapsing`. Ta obsahuje proměnnou `INPUT`, určující počet vstupů celého obvodu, proměnnou `vstupy`, což je pole názvů jednotlivých vstupů, a proměnné `b` a `data` pro načtení a uložení vstupního souboru. Pokud obvod obsahuje nějaké hradlo typu „xor“ či „xnor“, tak se již při načítání souboru převedou do tvaru využívajícího pouze hradel typu „and“, „or“ a „not“, viz. kapitola 3.3.3.
Hlavní metodou třídy je metoda `minimalize()`, která nejprve projde všechny vstupy obvodu a jejich názvy zapíše do proměnné `vstupy` a poté postupně na každý výstup obvodu volá rekurzivní funkci `minim_hradlo()` pro převod hradla z víceúrovňového na dvouúrovňové. Výsledný obvod uloží do souboru ve formátu `pla`. Funkce `absorpce()` a `minim()` jsou volány při převodu hradla z funkce `minim_hradlo()`.

```
class Collapsing {
     int INPUT;
     char *arg[5];  // argumenty prikazove radky
     BenchParserClient *data;
     BenchParser *b;
     string *vstupy;
     TVnejsi* minim_hradlo(std::string hradlo,
bool negace = false);
     void absorpce(TVnejsi* nove, int* novy_soucin);
     TVnejsi* minim(TVnejsi* nove);
  public:
     Collapsing(int ac, char *a[]);
     void minimalize();
};
```

Funkce `minim_hradlo()` rekurzivně prochází obvodem od výstupu ke vstupům a postupně převádí každé hradlo na součet součinů. Převod hradla probíhá tak, že se nejprve zjistí, či vstupy hradla jsou vstupy celého obvodu. Pokud ano, zapíší se jako vstupy hradla, pokud ne, zavolá se funkce `minim_hradlo()` na hradlo, jehož výstup je vstupem daného hradla.
V případě, že výstup převáděného hradla je propojen s více než jedním vstupem jiných hradel, tak se hradlo uloží resp. načte ze seznamu hotových hradel.
V proměnné `negace` si ukládám stav, zda je počet negací od výstupu obvodu směrem k danému hradlu lichý či sudý. V závislosti na počtu negací a původnímu typu hradla se změní typ hradla:

Sudý počet negaci:

- „nand“ – změní se na „and“
- „nor“ – změní se na „or“

Lichý počet negaci:

- „and“ – změní se na „or“
- „or“ – změní se na „and“
- „nand“ – změní se na „or“
- „nor“ – změní se na „and“

Nyní následuje nejdůležitější část, a to převod jednotlivých hradel na tvar součtu součinů. Nyní máme pouze hradla typu „and“ a „or“, neboť hradla typu „buf“ a „not“ se nemusí převádět. Podle kapitoly 3.3.3 se jednotlivá hradla převedou následovně:

- „and“ – vynásobí se mezi sebou každý součin ze vstupu hradla s každým součinem z dalšího vstupu hradla
- „or“ – sloučí se všechny součiny ze všech vstupů hradla do jednoho součinu

Při roznásobování se u hradla typu „and“ aplikuje pravidlo o stejném součinu z kapitoly 3.4.5. V metodě přímého převodu se u více než dvouvstupého hradla typu „and“ roznásobují mezi sebou naráz součiny ze všech vstupů hradla. Zde bylo roznásobování upraveno tak, že se vícevstupé hradlo roznásobuje v podstatě jako více dvouvstupých hradel.

Po vytvoření každého nového součinu se zavolá metoda `absorpce()`, která zjistí, či nově vytvořený součin nelze minimalizovat s některým z již vytvořených součinů. Metoda testuje tyto minimalizační pravidla z kapitoly 3.4: absorpce, absorpce negace, idempotence, vyloučení třetího. Oproti metodě přímého převodu jsem se pokusil metodu `absorpce()` pozměnit tak, aby dosahovala lepších výsledků za kratší dobu výpočtu.

Zda nové řešení převodu hradla typu „and“ a metody `absorpce()` vede ke kvalitnějšímu řešení si ukážeme v kapitole 5.2.

Posledním krokem je minimalizace pomocí externího programu Espreso, kterou provádí funkce `minim()`.

5 Testování

V této kapitole se budu věnovat důkladnému otestování implementovaných řešení. Nejprve otestuji metodu přímého převodu a vliv jednotlivých minimalizačních metod na kvalitu řešení. V další kapitole se budu věnovat metodě and-or a testování jiných přístupů oproti metodě přímého převodu. V poslední kapitole porovnám moje řešení s jinými již existujícími řešeními.

Testování provedu na následujících testovacích obvodech z [8]:

benchmark	počet vstupů	počet výstupů	počet hradel
s208	19	10	61
s298	17	20	75
s526	24	27	141
s382	24	27	99
s400	24	27	106
s820	23	24	256
s444	24	27	119
s1196	32	32	388
s1238	32	32	428
s349	24	26	104
s731	54	42	139
s838	67	34	241
s641	54	42	107
s1423	91	79	490

Tabulka 5.1: Přehled testovacích benchmarků

Výsledky testování budu zaznamenávat do tabulek, v nichž budou pro jednotlivé obvody zobrazeny tyto údaje:

- termy – počet termů výsledného převedeného obvodu
- čas – celkový čas nutný pro převod obvodu
- absorpce – celkový čas provádění metody absorpce() / počet provedených volání metody absorpce(), neboli počet vytvořených součinů - po každém vytvoření nového součinu je totiž volána metoda absorpce()
- čas převodu / počet jednotlivých hradel – doba potřebná k převedení všech hradel daného typu a počet takovýchto převodů
- ostatní – kolik času se spotřebuje např. na procházení obvodem, ukládání hradel atd.

Pokud se v tabulce objeví hodnota času rovna nule, znamená to, že čas výpočtu byl tak rychlý, že se jej nepodařilo změřit.

Pro porovnání testovaného řešení s řešením referenčním využiji grafy. Referenční řešení bude v grafu reprezentováno hodnotou 1 na ypsilonové ose. Pokud nebude uvedeno jinak, bude se na x-ové ose porovnávat čas dané veličiny.
Graf bude tedy zobrazovat, kolikrát se testované řešení zhoršilo, resp. zlepšilo a tomu budou na ypsilonové ose odpovídat hodnoty větší, resp. menší než hodnota 1.
Jestliže nebude v grafu některá z testovaných veličin pro daný obvod zobrazena, znamená to, že výsledná hodnota testované veličina byla rovna nule, pokud ovšem testovaná i referenční veličina budou rovny nule, je výsledkem hodnota 1. V případě, že testované řešení do určitého času nedospěje k výsledku, nabude měřená veličina hodnoty 1000.

Testovací sestavou bude notebook HP Compaq nx6310 s procesorem Intel Core 2 Duo 1,66 GHz a pamětí 512 MB RAM.

5.1 Metoda přímého převodu

Nejprve otestuji metodu přímého převodu s použitím všech mých minimalizačních metod:

	termy	čas [ms]	absorpce [ms]/počet	čas převodu [ms] / počet jednotlivých hradel					ostatní [ms]
				„and“	„or“	„nand“	„nor“	„not“	
s208	35	15	0/622	0/27	0/12	0/14	0/13	0/20	15
s298	92	31	0/954	0/33	0/17	0/9	0/19	0/25	31
s526	166	94	0/2172	0/58	0/29	0/22	0/35	0/32	94
s382	176	250	47/26850	0/14	0/26	0/29	188/36	0/31	62
s400	176	343	47/42172	0/14	0/27	0/35	250/36	0/28	93
s820	173	766	156/78923	0/76	0/60	453/54	47/66	0/25	266
s444	176	1141	139/100803	0/26	0/16	0/47	1094/36	0/29	47
s1196	3548	4938	1451/578548	46/171	31/126	1673/123	2563/51	15/35	610
s1238	3488	5156	1376/594120	47/186	48/141	1656/131	2780/56	15/33	610
s349	307	16015	3751/770269	0/58	0/34	15906/14	0/16	0/15	109

Tabulka 5.2: Metoda přímého převodu s použitím všech minimalizačních metod

Z výsledků vyplývá, že u jednodušších obvodů zaberou všechen čas pouze ostatní činnosti, kdežto u složitějších obvodů už to je hlavně převod hradel typu „nand“ a „nor“, ale také metoda absorpce().

V dalších testech se budu věnovat tomu, jaký vliv mají použitá minimalizační vylepšení na kvalitu výsledného řešení. To provedu tak, že se při testování dané minimalizační vylepšení nebude využívat a výsledky porovnám s tabulkou 5.2

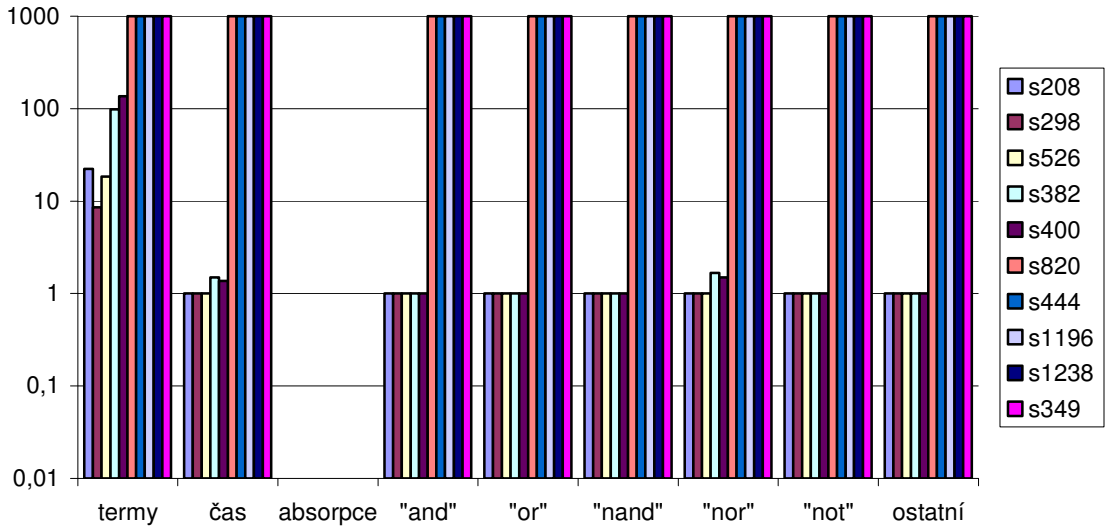
5.1.1 Minimalizace vytvořených součinů

K minimalizaci vytvořených součinů slouží metoda absorpce(), tudíž se v následujícím testu nebude využívat.

	termy	čas [ms]	absorpce [ms]/počet	čas převodu [ms] / počet jednotlivých hradel					ostatní [ms]
				„and“	„or“	„nand“	„nor“	„not“	
s208	782	15	0/0	0/27	0/12	0/14	0/13	0/20	15
s298	787	31	0/0	0/33	0/17	0/9	0/19	0/25	31
s526	3059	94	0/0	0/58	0/29	0/22	0/35	0/32	94
s382	17245	375	0/0	0/14	0/26	0/29	313/36	0/31	62
s400	24031	468	0/0	0/14	0/27	0/35	375/36	0/28	93
s820	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
s444	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
s1196	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
s1238	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
s349	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Tabulka 5.3: Metoda přímého převodu bez minimalizace vytvořených součinů

Z grafu 5.1 je patrné, že se počet termů bez použití minimalizace po vytvoření součinu u všech testovaných obvodů zvýšil mnohonásobně. Čas výpočtu se u jednodušších obvodů nezměnil, u obvodů s382 a s400 se zvýšil jen asi o polovinu, neboť se zvýšila doba výpočtu hradel typu „nor“. U složitějších obvodů se díky mnohonásobnému zvyšování počtu termů nepodařilo k řešení dospět.



Graf 5.1: Vliv vynechání absorpce u metody přímého převodu

5.1.2 Minimalizace vytvářených součinů

Nyní otestuji vliv minimalizace, která se provádí již při vytváření součinů:

	termy	čas [ms]	absorpce [ms]/počet	čas převodu [ms] / počet jednotlivých hradel					ostatní [ms]
				„and“	„or“	„nand“	„nor“	„not“	
s208	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
s298	92	18422	2441/2502163	0/33	0/17	0/9	18391/19	0/25	31
s526	166	15904	2452/2587745	0/58	0/29	16/22	15797/35	0/32	94
s382	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
s400	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
s820	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
s444	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
s1196	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
s1238	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
s349	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Tabulka 5.4: Metoda přímého převodu bez minimalizace při vytváření součinu

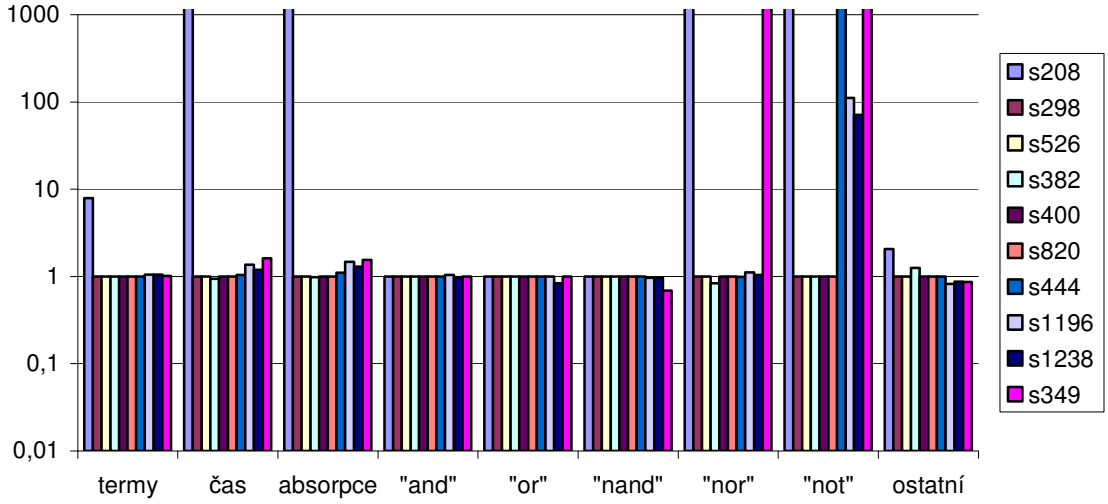
Obrovský vliv minimalizace při vytváření součinu je patrný z tabulky 5.4. Bez minimalizace se podařilo převést jen dva obvody, u kterých je patrné, že se počet vytvářených součinů zvýšil několika-řádově, což nám udává počet volání metody absorpce(). To je důvod, proč se u ostatních řešení nepodařilo k výsledku dospět.

5.1.3 Minimalizace při procházení obvodem

Pokud se v obvodu vyskytne hradlo typu „not“, jehož vstup je propojen s výstupem jiného negovaného hradla, provede se minimalizace zákonem dvojí negace, čímž se ze dvou negovaných hradel stane jedno hradlo nenegované. To by mělo ušetřit čas, neboť nejvíce časově náročné jsou právě převody negovaných hradel.

	termy	čas [ms]	absorpce [ms]/počet	čas převodu [ms] / počet jednotlivých hradel					ost. [ms]
				„and“	„or“	„nand“	„nor“	„not“	
s208	277	86203	28835/5064055	0/17	0/4	0/19	125/21	86047/35	31
s298	92	31	0/930	0/31	0/16	0/9	0/19	0/44	31
s526	166	94	0/2089	0/56	0/28	0/22	0/35	0/52	94
s382	176	235	46/27010	0/11	0/24	0/30	157/34	0/59	78
s400	176	343	47/42322	0/11	0/25	0/36	250/34	0/28	93
s820	173	766	156/78931	0/76	0/60	453/54	47/66	0/33	266
s444	176	1188	153/114729	0/13	0/14	0/58	1079/34	62/62	47
s1196	3737	6719	2137/806481	48/118	31/101	1608/119	2859/50	1672/141	501
s1238	3658	6166	1786/735828	46/134	40/112	1594/125	2889/57	1065/80	532
s349	311	25889	5805/1237455	0/44	0/10	10904/19	9376/31	5515/57	94

Tabulka 5.5: Metoda přímého převodu bez minimalizace při procházení obvodem



Graf 5.2: Vliv vynechání minimalizace při procházení obvodem u metody přímého převodu

V grafu 5.2 vidíme, že vliv minimalizace při procházení obvodem nejvíce ovlivňuje čas převodu negovaných hradel. Může mít i za následek zvýšený počet výsledných termů.
Vliv minimalizace při procházení obvodem se projeví u obvodů, majících více negovaných hradel, jejichž výstup je propojen na vstup hradla typu „not“. U mnou testovaných obvodů je vliv patrný hlavně u složitějších obvodů, i když se může projevit i u jednoduššího obvodu, jako např. obvod s208, u kterého se vliv minimalizace projevil nejvíce.

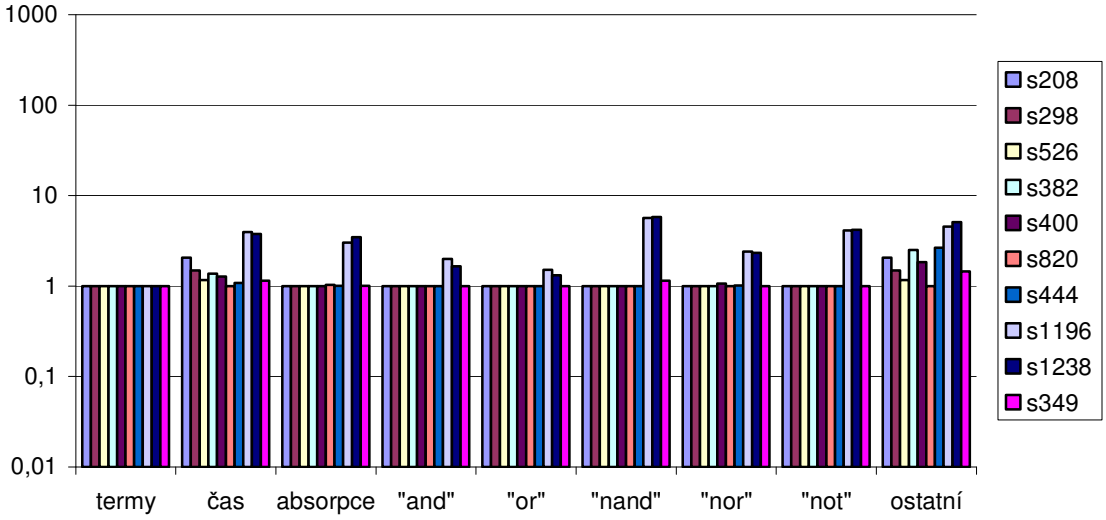
5.1.4 Ukládání opakujících se hradel

U hradel, jejichž výstup je propojen na více vstupů jiných hradel, si můžeme převedené hradlo uložit pro pozdější použití. Nyní otestuji metodu přímého převodu bez opakování hradel. Výsledky jsou zobrazeny v tabulce 5.6.

	termy	čas [ms]	absorpce [ms]/počet	čas převodu [ms] / počet jednotlivých hradel					ost. [ms]
				„and“	„or“	„nand“	„nor“	„not“	
s208	35	31	0/785	0/38	0/18	0/36	0/34	0/51	31
s298	92	46	0/1232	0/47	0/22	0/18	0/40	0/127	46
s526	166	110	0/2595	0/76	0/35	0/42	0/59	0/222	110
s382	176	344	47/27549	0/15	0/42	0/45	188/141	0/143	156
s400	176	437	47/42969	0/15	0/48	0/56	266/153	0/162	171
s820	173	766	161/79133	0/76	0/60	453/54	47/66	0/235	266
s444	176	1234	140/102992	0/27	0/41	0/79	1109/189	0/192	125
s1196	3548	19539	4385/2127086	92/935	47/483	9438/867	6171/348	62/1012	2779
s1238	3488	19384	4795/2179240	78/1032	63/582	9582/1044	6453/386	63/1178	3109
s349	307	18330	3782/770679	0/152	0/61	18172/21	0/44	0/155	158

Tabulka 5.6: Metoda přímého převodu bez opakování hradel

Z grafu 5.3 je patrné, že opakování hradel má vždy vliv na dobu průchodu obvodem a tudíž i na celkovou dobu převodu obvodu. Dále se výhoda opakování hradel projeví u složitějších obvodů, u kterých doba převodu opakujících se hradel nemá zanedbatelnou hodnotu.



Graf 5.3: Vliv vynechání ukládání opakujících se hradel u metody přímého převodu

5.2 Metoda and-or

Při testování této metody nejprve otestuji všechny její vylepšení oproti metodě přímého převodu a poté, opět jako u předchozí metody, vliv minimalizačních metod na výsledné řešení.

5.2.1 Porovnání metody and-or a metody přímého převodu

Tento test ukáže rozdíl mezi jiným přístupem k řešení problému pomocí metody and-or oproti metodě přímého převodu. Změny oproti metodě přímého převodu jsou jednak v jiné metodě, ale též ve změně datové struktury pro uložení vstupního obvodu. Výsledek měření metody and-or je v tabulce 5.7:

	termy	čas [ms]	absorpce [ms]/počet	čas převodu [ms] / počet jednotlivých hradel		ostatní [ms]
				„and“	„or“	
s208	30	0	0/108	0/49	0/16	0
s298	95	0	0/546	0/31	0/49	0
s526	169	0	0/933	0/66	0/82	0
s382	176	0	0/744	0/58	0/58	0
s400	173	0	0/853	0/61	0/65	0
s820	142	0	0/967	0/156	0/100	0
s444	173	0	0/932	0/73	0/66	0
s1196	3348	187	93/26357	126/286	46/244	15
s1238	3117	202	94/26900	141/324	46/255	15
s349	295	0	0/1471	0/57	0/70	0
s641	1659	375	265/48766	375/73	0/68	0
s713	1659	453	281/52041	437/116	16/97	0

Tabulka 5.7: Metoda and-or

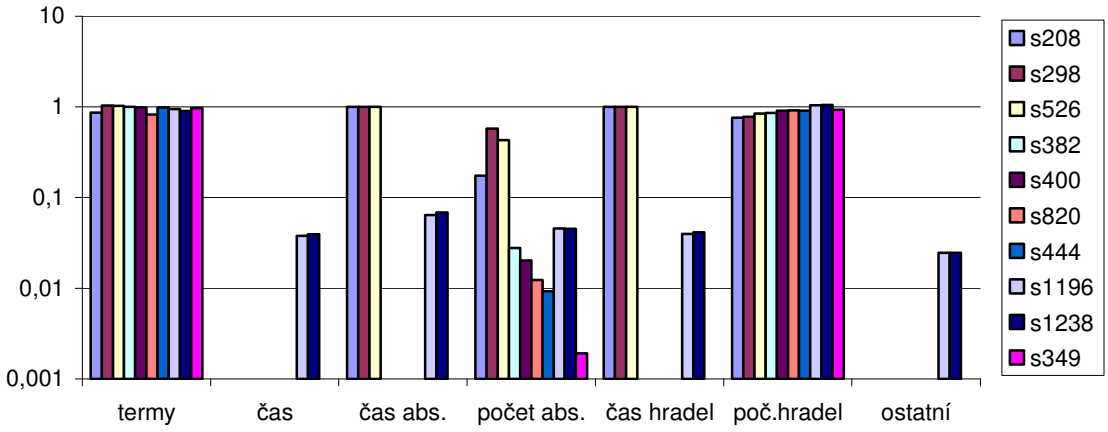
V tabulce 5.7 jsou otestovány navíc obvody s641 a s713, které sice nyní pro porovnání nevyužiji, ale jejich výsledky budu potřebovat v dalších testech.

Výsledné počty termů a časy obou metod uvádím v tabulce 5.8:

	metoda přímého převodu		metoda and-or	
	termy	čas [ms]	termy	čas [ms]
s208	35	15	30	0
s298	92	31	95	0
s526	166	94	169	0
s382	176	250	176	0
s400	176	343	173	0
s820	173	766	142	0
s444	176	1141	173	0
s1196	3548	4938	3348	187
s1238	3488	5156	3117	202
s349	307	16015	295	0

Tabulka 5.8: Výsledky měření metody přímého převodu a metody and-or

V grafu 5.4 porovnám výsledky měření metody přímého převodu z tabulky 5.2 s výsledky měření metody and-or z tabulky 5.7.
Pro určení veličin čas hradel a počet hradel v grafu 5.4 jsem sečetl čas a počet všech typů hradel pro danou metodu a testovací obvod.



Graf 5.4: Porovnání metody and-or oproti metodě přímého převodu

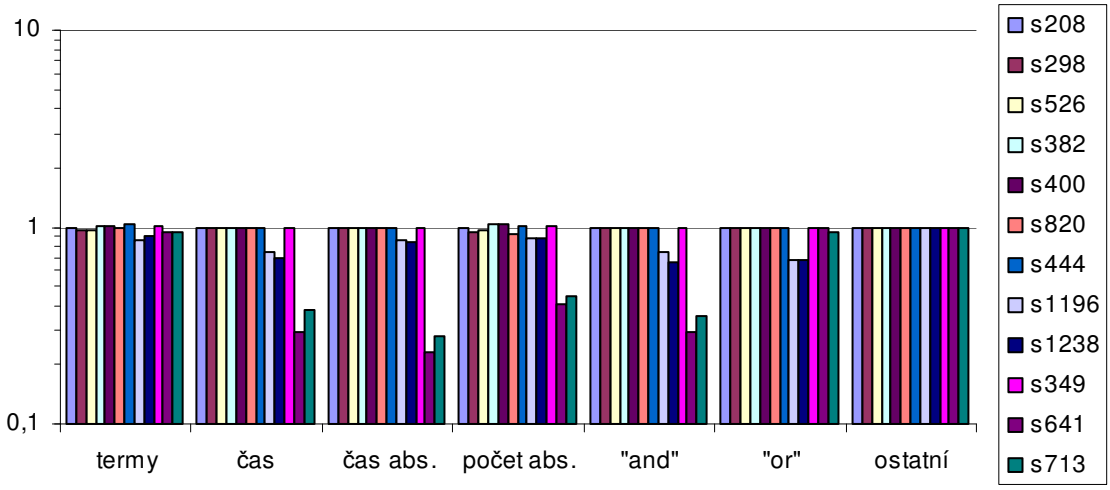
Z tabulky 5.7 a grafu 5.4 vidíme, že metoda and-or oproti metodě přímého převodu nemá zásadní vliv na počet výsledných termů ani na celkový počet převáděných hradel. Naproti tomu má velmi vysoký vliv na počet a dobu volání metody absorpce() a hlavně na čas převodu hradel a průchodu obvodem.

5.2.2 Nové řešení minimalizace vytvořených součinů

Metoda and-or obsahuje oproti metodě přímého převodu novou verzi metody absorpce().
V kapitole 5.2.1 proběhlo testování se starou verzí, nyní provedu otestování nové verze a porovnám s výsledky z tabulky 5.7.

	termy	čas [ms]	absorpce [ms]/počet	čas převodu [ms] / počet jednotlivých hradel		ostatní [ms]
				„and“	„or“	
s208	30	0	0/108	0/49	0/16	0
s298	91	0	0/510	0/31	0/49	0
s526	165	0	0/897	0/66	0/82	0
s382	179	0	0/767	0/58	0/58	0
s400	176	0	0/874	0/61	0/65	0
s820	142	0	0/897	0/156	0/100	0
s444	179	0	0/943	0/73	0/66	0
s1196	2853	140	79/23308	94/286	31/244	15
s1238	2786	140	79/23740	94/324	31/255	15
s349	301	0	0/1475	0/57	0/70	0
s641	1574	110	62/19995	110/73	0/68	0
s713	1574	171	79/23223	156/116	15/97	0

Tabulka 5.9: Metoda and-or s novým řešením minimalizace vytvořených součinů



Graf 5.5: Vliv nového řešení minimalizace vytvořených součinů u metody and-or

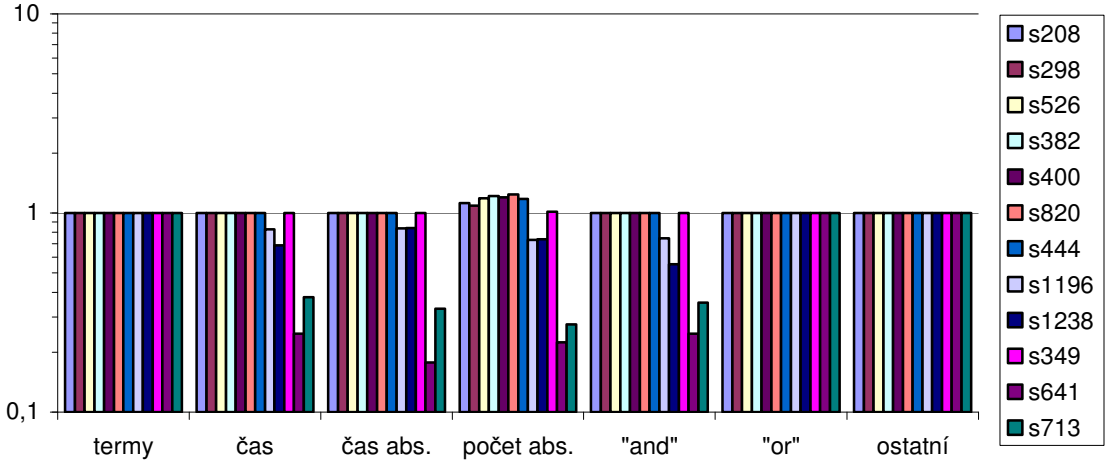
Z grafu 5.5 je patrné, že u jednoduchých obvodů má nové řešení minimalizace vytvořených součinů pouze mírný vliv na zvýšený počet výsledných termů, kdežto u složitějších obvodů má velký vliv na snížení počtu termů a absorpcí a na snížení dob výpočtů všech měřených veličin.

5.2.3 Nové řešení převodu hradla typu „and“

Metoda and-or obsahuje také novou verzi převodu hradla typu „and“. V kapitole 5.2.1 proběhlo testování se starou verzí, nyní provedu otestování nové verze a porovnám s výsledky z tabulky 5.7.

	termy	čas [ms]	absorpce [ms]/počet	čas převodu [ms] / počet jednotlivých hradel		ostatní [ms]
				„and“	„or“	
s208	30	0	0/121	0/49	0/16	0
s298	95	0	0/594	0/31	0/49	0
s526	169	0	0/1106	0/66	0/82	0
s382	176	0	0/906	0/58	0/58	0
s400	173	0	0/1023	0/61	0/65	0
s820	142	0	0/1198	0/156	0/100	0
s444	173	0	0/1096	0/73	0/66	0
s1196	3348	155	78/19312	94/286	46/244	15
s1238	3117	139	79/19894	78/324	46/255	15
s349	295	0	0/1494	0/57	0/70	0
s641	1659	93	47/10941	93/73	0/68	0
s713	1659	171	93/14344	155/116	16/97	0

Tabulka 5.10: Metoda and-or s novým řešením převodu hradla typu „and“



Graf 5.6: Vliv nového řešení převodu hradla typu „and“ u metody and-or

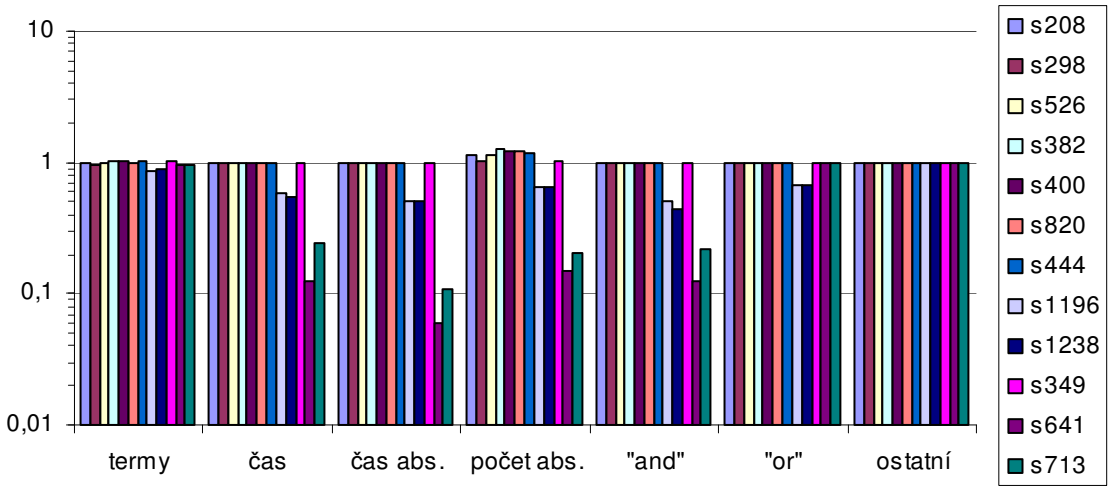
Vliv nového řešení převodu hradla typu „and“ se sice u jednoduchých obvodů projeví zvýšeným počtem absorpcí, ale výsledný počet termů a celkový čas to neovlivní. Naopak u složitějších obvodů počet absorpcí výrazně klesá, čímž klesá čas výpočtu absorpce a celkový čas. Nové řešení má samozřejmě též velký vliv na čas samotného výpočtu hradla typu „and“.

5.2.4 Vylepšená metoda and-or

V této kapitole otestuji metodu and-or jak s novým řešením minimalizace vytvořených součinů tak s novým řešením převodu hradla typu „and“ a porovnám s výsledky z tabulky 5.7.

	termy	čas [ms]	absorpce [ms]/počet	čas převodu [ms] / počet jednotlivých hradel		ostatní [ms]
				„and“	„or“	
s208	30	0	0/121	0/49	0/16	0
s298	91	0	0/558	0/31	0/49	0
s526	165	0	0/1070	0/66	0/82	0
s382	179	0	0/929	0/58	0/58	0
s400	176	0	0/1044	0/61	0/65	0
s820	142	0	0/1198	0/156	0/100	0
s444	179	0	0/1107	0/73	0/66	0
s1196	2857	109	47/16830	63/286	31/244	15
s1238	2785	109	47/17225	63/324	31/255	15
s349	301	0	0/1498	0/57	0/70	0
s641	1574	47	16/7343	47/73	0/68	0
s713	1574	110	31/10769	94/116	16/97	0

Tabulka 5.11: Vylepšená metoda and-or



Graf 5.7: Rozdíl mezi původní a vylepšenou metodou and-or

Vliv vylepšení metody and-or novým řešením minimalizace vytvořených součinů a novým řešením převodu hradla typu „and“ zobrazuje graf 5.7. Je na něm vidět, že u jednodušších obvodů se může mírně navýšit počet termů, ale naopak u složitějších obvodů se počet termů může výrazněji snížit. Dále také u složitějších obvodů velmi klesá doba výpočtu, u obvodu s641 dokonce až o jeden řád.

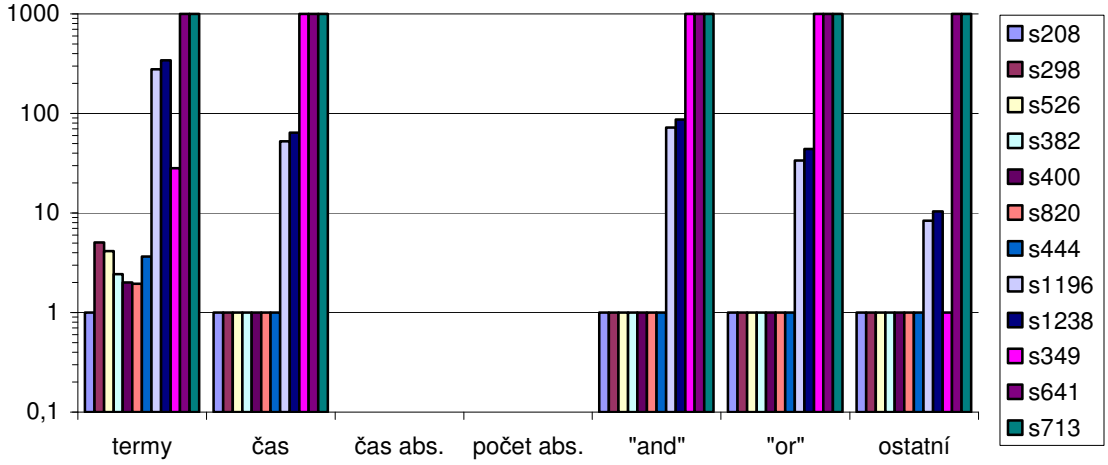
V následujících testech otestuji vliv minimalizačních metod na vylepšenou metodu and-or.

5.2.5 Vliv minimalizace vytvořených součinů na metodu and-or

Minimalizace vytvořených součinů se provádí pomocí metody absorpce(), která se v tomto testu nebude využívat. Výsledky porovnám s výsledky vylepšené metody and-or z tabulky 5.11.

	termy	čas [ms]	Absorpce [ms]/počet	čas převodu [ms] / počet jednotlivých hradel		ostatní [ms]
				„and“	„or“	
s208	30	0	0/0	0/49	0/16	0
s298	462	0	0/0	0/31	0/49	0
s526	683	0	0/0	0/66	0/82	0
s382	434	0	0/0	0/58	0/58	0
s400	352	0	0/0	0/61	0/65	0
s820	277	0	0/0	0/156	0/100	0
s444	652	0	0/0	0/73	0/66	0
s1196	793043	5735	0/0	4563/286	1046/244	126
s1238	952807	6984	0/0	5468/324	1360/255	156
s349	8497	31	0/0	15/57	16/70	0
s641	N/A	N/A	N/A	N/A	N/A	N/A
s713	N/A	N/A	N/A	N/A	N/A	N/A

Tabulka 5.12: Metoda and-or bez minimalizace vytvořených součinů



Graf 5.8: Vliv minimalizace vytvořených součinů na metodu and-or

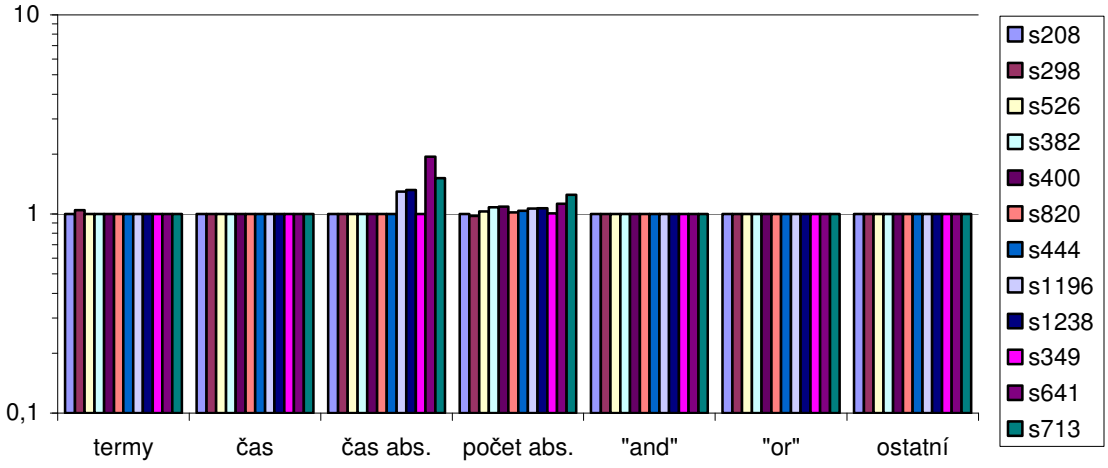
Minimalizace vytvořených součinů má velký vliv na zvýšený počet termů, u složitějších obvodů až několika-řádový. Na dobu výpočtů vliv u jednoduchých obvodů nemá, ale u složitějších obvodů opět stoupá doba výpočtů až několika-řádově. U obvodů s641 a s713 se kvůli rychle se zvyšujícímu počtu termů nepodařilo k řešení dospět, neboť se po chvíli zahltila paměť.

5.2.6 Vliv minimalizace vytvářených součinů na metodu and-or

Nyní otestuji vliv minimalizace, která se provádí při vytváření součinů. Výsledky bez této minimalizace zahrnuje tabulka 5.13:

	termy	čas [ms]	absorpce [ms]/počet	čas převodu [ms] / počet jednotlivých hradel		ostatní [ms]
				„and“	„or“	
s208	30	0	0/121	0/49	0/16	0
s298	95	0	0/546	0/31	0/49	0
s526	165	0	0/1101	0/66	0/82	0
s382	179	0	0/1004	0/58	0/58	0
s400	176	0	0/1135	0/61	0/65	0
s820	142	0	0/1219	0/156	0/100	0
s444	179	0	0/1147	0/73	0/66	0
s1196	2857	109	61/17940	63/286	31/244	15
s1238	2785	109	62/18442	63/324	31/255	15
s349	301	0	0/1511	0/57	0/70	0
s641	1574	47	31/8268	47/73	0/68	0
s713	1574	110	47/13432	94/116	16/97	0

Tabulka 5.13: Metoda and-or bez minimalizace vytvářených součinů



Graf 5.9: Vliv minimalizace vytvářených součinů na metodu and-or

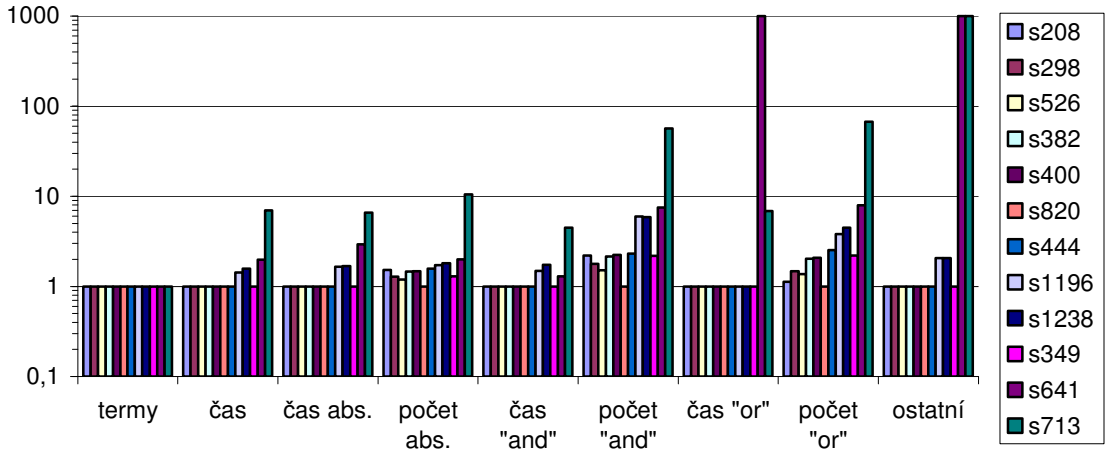
Na grafu 5.9 vidíme, že minimalizace vytvářených součinů má vliv pouze na zvýšený počet volání metody absorpce() a tudíž i na dobu jejího provádění, celkový čas výpočtu ovšem neovlivní.

5.2.7 Vliv ukládání opakujících se hradel na metodu and-or

Ukládání opakujících se hradel by mělo zamezit opakovaným převodům hradel, která již jednou byla převedena. Výsledky testu bez opakování hradel jsou v tabulce 5.14:

	termy	Čas [ms]	absorpce [ms]/počet	čas převodu [ms] / počet jednotlivých hradel		ostatní [ms]
				„and“	„or“	
s208	30	0	0/184	0/108	0/18	0
s298	91	0	0/717	0/55	0/72	0
s526	165	0	0/1278	0/100	0/112	0
s382	179	0	0/1359	0/125	0/118	0
s400	176	0	0/1538	0/137	0/135	0
s820	142	0	0/1198	0/156	0/100	0
s444	179	0	0/1747	0/169	0/167	0
s1196	2857	156	78/28923	94/1705	31/928	31
s1238	2785	171	79/31077	109/1897	31/1147	31
s349	301	0	0/1941	0/124	0/154	0
s641	1574	93	47/14617	61/548	16/540	16
s713	1574	769	204/113458	422/6545	110/6550	237

Tabulka 5.14: Metoda and-or bez ukládání opakujících se hradel



Graf 5.10: Vliv ukládání opakujících se hradel na metodu and-or

Vypnutí funkce ukládání opakujících se hradel má vliv na zvýšený počet převáděných hradel a tudíž i na větší čas potřebný k jejich převodu, čímž vzrůstá i celkový čas potřebný k převodu obvodu. Jelikož některé hradla se opakují vícekrát, musí se též určitými částmi obvodu procházet vícekrát, což má také za následek zvýšenou dobu průchodu obvodem.

5.2.8 Minimalizace pomocí externího programu Espresso

K dosažení lepšího výsledku, co se počtu termů tyče, použiji pro minimalizaci převedených hradel ještě program Espresso. Ten můžu použít buď na všechny hradla, nebo jen na hradla od určitého počtu součinů, a nebo jen na výsledné výstupy.
V tabulce 5.15 jsou otestované různé kombinace použití Espressa a jejich vliv na celkový čas a celkový počet termů, také uvádím počet volání Espressa:

	bez espressa			na výsledek			na všechny		
	termy	čas	počet	termy	čas	počet	termy	čas	počet
s208	30	0	0	30	265	10	30	1718	65
s298	91	0	0	68	531	20	68	2125	80
s526	165	0	0	142	718	27	142	3968	148
s382	179	0	0	167	734	27	167	3093	116
s400	176	0	0	167	719	27	167	3359	126
s820	142	0	0	126	641	24	126	6813	256
s444	179	0	0	167	719	27	167	3703	139
s1196	2857	109	0	1048	1313	32	1049	14406	530
s1238	2785	109	0	1048	1313	32	1049	15703	579
s349	301	0	0	249	703	26	249	3406	127
s641	1574	47	0	912	1296	42	912	3968	141
s713	1574	110	0	912	1390	42	912	6156	210

	na víc než 10			na víc než 20 a na výsledek			na víc než 50 a na výsledek		
	termy	čas	počet	termy	čas	počet	termy	čas	počet
s208	30	31	1	30	265	10	30	265	10
s298	74	156	6	68	531	20	68	531	20
s526	148	515	19	142	719	27	142	718	27
s382	168	578	21	167	781	29	167	718	27
s400	168	593	22	167	781	29	167	719	27
s820	126	485	18	126	781	29	126	641	24
s444	168	640	24	167	812	30	167	719	27
s1196	1052	3328	114	1050	2859	94	1050	2078	65
s1238	1052	3406	117	1050	2828	93	1050	2062	64
s349	249	750	27	249	1140	41	249	906	32
s641	912	1828	61	912	2343	78	912	2078	68
s713	912	2765	92	912	2797	92	912	2390	76

Tabulka 5.15: Vliv minimalizace pomocí externího programu Espresso

Z tabulky 5.15 je patrné, že použitím Espressa dosáhneme vždy menšího nebo stejného počtu výsledných termů. Pokud Espresso nepoužijeme, dosáhneme řešení v kratším čase.
Zvýšení doby řešení při použití Espressa je zřejmě také hodně ovlivněno vstupně/výstupními operacemi při předávání termů mezi programy.
Nejlepších výsledků dosáhneme použitím Espressa jen pro minimalizaci výsledných výstupů obvodu, u ostatních kombinací se buď zvýší čas výpočtu nebo výsledný počet termů.

5.3 Porovnání s jinými existujícími řešeními

V této kapitole porovnám moje řešení pomocí metody and-or z kapitoly 5.2.4 a řešení s použitím programu Espreso z kapitoly 5.2.8 s jinými již existujícími řešeními, kterými jsou:

- DP – diplomová práce Přemysla Ruckého pomocí BDD (leden 2007) [3]
- BP – bakalářská práce Davida Tomana založená na podobném principu jako moje řešení (srpen 2007) [4]
- SIS 1.2 (verze z 21.5.2002) [6]
- MVSIS 1.1 (verze z 13.5.2002) [7]
- MVSIS 2.0 (verze z 27.5.2003) [7]

V tomto testu se zaměřím na čas potřebný k převedení obvodu a na výsledný počet termů.
Čas u DP a BP měřili programy samotné. U programů SIS a MVSIS jsem musel změřit dobu potřebnou pro načtení obvodu a dobu pro načtení a převod obvodu a výsledné časy od sebe odečíst.
DP jsem musel testovat pod operačním systémem Linux, kdežto všechny ostatní metody pod operačním systémem Windows XP, může zde dojít k nepřesnostem v porovnání kvality řešení.
V tabulce 5.16 je pro jednotlivá řešení zobrazen výsledný počet termů a v tabulce 5.16 čas potřebný k převedení obvodu.

	and-or	and-or + espresso	DP	BP	SIS 1.2	MVSIS 1.1	MVSIS 2.0
s208	30	30	37	31	30	30	30
s298	91	68	69	93	79	78	69
s526	165	142	174	185	152	152	143
s382	179	167	250	255	174	174	167
s400	176	167	250	264	175	175	167
s820	142	126	142	203	134	133	127
s444	179	167	226	220	175	175	167
s1196	2857	1048	2082	2421	1479	1462	1118
s1238	2785	1048	2083	2256	1461	1444	1105
s349	301	249	449	253	264	264	261
s641	1574	912	1532	1592	1021	1021	912
s713	1574	912	1532	1592	1021	1021	912
s838	114	114	N/A	471	114	114	114
s1423	111409	43533	464645	N/A	51737	55314	43885

Tabulka 5.16: Porovnání výsledného počtu termů pro různá řešení

	and-or	and-or + espresso	DP	BP	SIS 1.2	MVSIS 1.1	MVSIS 2.0
s208	0	265	20	31	16	14	0
s298	0	531	10	110	0	14	0
s526	0	718	30	282	14	16	0
s382	0	734	20	281	0	14	0
s400	0	719	30	313	15	14	0
s820	0	641	60	297	14	31	14
s444	0	719	30	500	0	16	14
s1196	109	1313	290	9812	31	63	31
s1238	109	1313	310	12687	31	79	31
s349	0	703	40	313	14	16	14
s641	47	1296	310	12500	47	62	31
s713	110	1390	330	78672	61	79	15
s838	0	906	N/A	3375	16	31	62
s1423	4780171	5312937	1284100	N/A	14359	21079	186

Tabulka 5.17: Porovnání výsledného času pro různá řešení

Z výsledků měření v tabulkách 5.16 a 5.17 lze vyvodit, že nejlepšího řešení dosahuje MVSIS 2.0, neboť dosahuje nejlepšího poměru počtu výsledných termů za čas. Rozdíl mezi programy SIS 1.2 a MVSIS 1.1 se výrazně neliší, SIS 1.2 dosahuje řešení v trochu kratším čase, kdežto MVSIS 1.1 dosahuje o trochu menšího počtu termů, což ovšem neplatí u nejsložitějšího obvodu s1423.

Menšího počtu termů než MVSIS 2.0 dosahuje pouze metoda and-or v kombinaci s Espressem, ovšem čas jejího výpočtu je mnohonásobně vyšší. Samotná metoda and-or může dosáhnout kratší doby výpočtu než MVSIS 2.0, ale výsledný počet termů je vyšší.

Při porovnání metody and-or s DP a BP je patrné, že metoda and-or dosahuje kratší doby výpočtu, oproti BP dokonce až řádově. Co se počtu termů tyče, tak and-or dosahuje menšího počtu termů u jednodušších obvodů, u složitějších naopak dosahují menšího počtu DP a BP.

Při testování nejsložitějšího obvodu s1423 se ukázalo, jak velmi časově náročné jsou metody and-or, DP a BP oproti SISu a MVSISu. U BP se výsledku ani nepodařilo dosáhnout, neboť se po pár minutách začala rychle zvyšovat paměťová náročnost programu a došlo k zahlcení paměti. DP sice dosáhla řešení asi ve čtyřikrát kratším čase než metoda and-or, ovšem výsledný počet termů byl u metody and-or zase čtyřikrát menší.

6 Závěr

Úkolem této práce bylo navrhnout a naimplementovat nástroj pro převod víceúrovňové logické sítě na dvouúrovňovou. Byly zrealizovány dvě metody založené na rekurzivním průchodu obvodu od výstupů směrem ke vstupům a postupném převodu hradel na součet součinů, které jsem nazval jako metodu přímého převodu a metodu and-or. Ty byly důkladně otestovány a porovnány, metoda and-or pak byla porovnána s dalšími již existujícími řešeními.

Testování metody přímého převodu ukázalo, jak velmi důležité je při převodu použití minimalizačních metod, pomocí kterých dosáhneme kvalitnějšího řešení v kratším čase. Nejvíce důležitá v tomhle směru je minimalizace vytvářených součinů a také minimalizace vytvořených součinů, které jsou důležité pro všechny typy obvodů. Pro obvody obsahující velké množství negovaných hradel je důležitá minimalizace při procházení obvodem, pro obvody obsahující velké množství opakujících se hradel zase ukládání opakujících se hradel. Nejvíce času u této metody zabral převod negovaných hradel, což se mi podařilo odstranit v metodě and-or.

U metody and-or se oproti metodě přímého převodu podařilo dosáhnout mnohem kvalitnějšího řešení, a to hlavně díky eliminaci převodu negovaných hradel. Dalším vylepšením bylo použití vlastní datové struktury, která byla vytvořena přímo pro účely této metody, kdežto u metody přímého převodu byla použita obecná struktura. Kromě toho byla ještě vylepšena minimalizace vytvořených součinů a také funkce pro převod hradel.

Pro vylepšení výsledného řešení poskytují obě metody též možnost minimalizace externím programem Espresso, a to nejen na výsledné výstupy, ale také v průběhu převodu obvodu.

Porovnání kvality řešení bylo provedeno nejen s přístupem založeným na binárních rozhodovacích diagramech z diplomové práce Přemysla Ruckého, ale též s bakalářskou prací Davida Tomana a programy SIS 1.2, MVSIS 1.1 a MVSIS 2.0. Bylo zjištěno, že metoda and-or se co do rychlosti může rovnat s programy SIS a MVSIS a u jednodušších obvodů neztrácí mnoho ani ve výsledném počtu termů. Oproti zbylým dvěma řešením poskytuje metoda and-or mnohem rychlejší řešení a u jednodušších obvodů dosahuje i menšího počtu termů. Navíc při použití externího programu Espresso lze dosáhnout nejmenšího počtu termů ze všech testovaných řešení.

Z výsledků při použití programu Espresso vyplývá, že je zde ještě možnost vylepšení stávajícího řešení, a to vylepšením minimalizace vytvořených součinů a možná i zdokonalením převodu hradla typu „and“ tak, aby se generovalo ještě méně součinů, které jsou následně minimalizovány

7 Seznam literatury

[1] J. Schmidt. *projekt HUB (nyní EduArd)*

[2] Internetové stránky předmětu logické systémy. `http://service.felk.cvut.cz/courses/36LS/`.

[3] P. Rucký. *Převod víceúrovňové logické sítě na dvouúrovňovou pomocí BDD*. DP ČVUT, FEL, 2007.

[4] D. Toman. *Nástroj pro manipulaci s logickými funkcemi popsanými algebraickým výrazem.* BP ČVUT, FEL, 2007.

[5] R. Rudell. *Espresso – Boolean Minimization*. University of California, 1988.

[6] D. Protheroe, P. Stallard. *SIS*. `ic.eecs.berkeley.edu`

[7] D. Chai, J.-H. Jiang, Y. Jiang, Y. Li, A. Mishchenko, R. Brayton. *MVSIS*. University of California. `http://www-cad.eecs.berkeley.edu/mvsis`.

[8] F. Brglez, D. Bryan, K. Koźmiński. *Combinational Profiles of Sequential Benchmark Circuits*. Microelectronics Center of North Carolina, 1989.

[9] P. Herout. *Učebnice jazyka C*. Kopp, České Budějovice, třetí upravené vydání, 1999.

[10] M. Virius. *Od C k C++*. Kopp, České Budějovice, druhé vydání, 2002.

[11] Pokyny pro psaní bakalářských a magisterských závěrečných prací na katedře počítačů. `https://info336.felk.cvut.cz`.

A Uživatelská / instalační příručka

Program není nutno instalovat. Spouští se z příkazové řádky s minimálně dvěma argumenty, které určují vstupní a výstupní soubor:

```
collapsing vst_soubor výst_soubor [esp_vysledek [esp_pocet]]
```

Poslední dva nepovinné argumenty určují použití externího programu Espresso při minimalizaci obvodu (defaultně se nepoužívá):

- `esp_vysledek` – či se má použít Espresso na minimalizaci výsledných výstupů obvodu, může nabývat hodnot `true` nebo `false`, defaultně `false`
- `esp_pocet` – od jakého počtu součinů se po převedení každého hradla má použít Espresso: – pokud se nemá použít, zadá se hodnota `-1` (defaultně)
  – pro minimalizaci všech hradel hodnota 1
  – pro minimalizaci hradel s více než 2,3,4… součiny hodnota `2`, `3`, `4`…

Pro použití minimalizace pomocí Espressa je třeba mít program Espresso nahraný ve stejném adresáři jako program collapsing.

B Obsah přiloženého CD

Přiložené CD obsahuje v kořenovém adresáři následující soubor a adresáře:

- index.html – výchozí stránka projektu
- text/ – adresář obsahující text DP
- exe/ – adresář se spustitelným programem a popisem spuštění programu
- data/ – adresář s testovanými benchmarky
- src/ – adresář se zdrojovými kódy programu