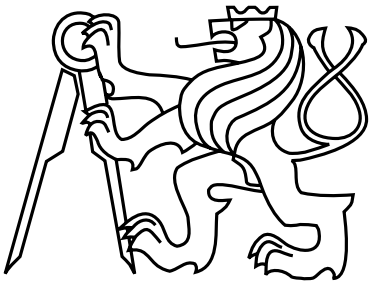
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalárská práca

Kreslení schémat logických obvodů

Tomáš Macik

Veducí práce: Ing. Petr Fišer, Ph. D.

Študijny program: Elektrotechnika a informatika, štruktúrovaný, Bakalársky

Odbor: Výpočetná technika

22. januára 2009

Poďakovanie

V prvej rade by som chcel poďakovať vedúcemu tejto práce, pánovi Ing. Petrovi Fišerovi, Ph. D., za jeho rady a odbornú kritiku mojej práce. Taktiež mojej rodine, ktorá ma podporovala a v neposlednej rade aj slečne Michaele Klímovej, za jej rady z pohľadu laika a morálnu podporu.

Prehlásenie

Prehlasujem, že som prácu vypracoval samostatne a použil som len podklady uvedené v priloženom zozname.
Nemam závažný dôvod proti použitiu tohoto školského diela v zmysle §60 Zákona č.121/2000 Zb., o autorskom práve, o právach súvisiacich s autorkým právom a o zmene niektorých zákonov (autorský zákon).

V Prahe dňa 23. 1. 2009 .....

Abstract

The task of this thesis is to create a tool for drawing logical circuit, described in a *bench* file, to a *xml* file defined by the program INKS and to a postscript file. The graphical representation of this circuit should be as readable and illustrative as it could be. The thesis outlines the conditions of readability and illustrability. It forms algorithms for placing gates, wires and other components of the scheme on their basis. The thesis describes at the end the implementations of the algorithms and values the achieved results.

Abstrakt

Účelom tejto práce je vytvoriť nástroj na vykreslenie schémy logického obvodu popísaného súborom vo formáte *bench* do formátu *xml* pre nástroj INKS a pre postscript. Grafická podoba tejto schémy by mala byť čo najčitateľnejšia a čo najnázornejšia. Načrtáva požiadavky pre čitateľnosť a názornosť a na ich základe vytvára algoritmus pre uloženie hradiel, spojov a iných prvkov schémy. Na záver popisuje implementáciu algoritmov a zhodnocuje dosiahnuté výsledky.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Kapitola 1

Úvod

Táto práca mala za úlohu nakresliť schému logického obvodu zadaného vo formáte bench a exportovať ju do formátu xml pre program INKS a do postscriptového formátu pdf. Program IKNS je nástroj pre grafický návrh schémat logických obvodov, ktorého autorom je Bc. Rostislav Pastor a vo svojej bakalárskej práci ho vylepšil Bc. Robert Škorpil.

Ani jedna z verzií tohoto programu však neobsahuje import z formátu bench. Môj nástroj Betopox by mal túto funkcionalitu dopĺňať ako samostatný program. Doplnenie tohoto nástroja vytvorí plnohodnotnú sadu na vytváranie schém strojovo alebo manuálne pomocou textového popisu a interaktívneho kreslenia. Betopox bude vytvárať súbory typu xml pre INKS a s pomocou INKS ich bude možné exportovať do formátov pdf a ps.

Zameriam sa hlavne na rozmiestnenie komponent a ich optimalizáciu.

Kapitola 2

Popis problému, špecifikácie cieľa

2.1 Popis problému

Na rozdiel od systémov typu INKS, kde sa jeho autori zaoberajú hlavne vytváraním a routovaním spojov medzi jednotlivými hradlami alebo blokmi, je tento problém rozšírený ešte aj o počiatočnú fázu rozmiestnenia samotných blokov alebo hradiel. Popis hradiel by sme mohli chápať ako štruktúru grafu, kde každé logické hradlo alebo celý blok je chápaný ako uzol a ich spoje ako hrany grafu. Pri zobarzovaní grafu existuje mnoho algoritmov. Ich upravením, by sme však mohli dostať ten správny teoretický nástroj. Do týchto algoritmov bude treba začleniť požiadavky na veľkosť grafikých prvkov. Ich umiestnenie nie je závislé len na ich rozmeroch, ale aj na ich vzájomnej pozícií. Budeme musieť brať ohľad aj na vodiče, ktoré dostanú reálnu grafickú podobu a tým aj rozmery a priestorové požiadavky.

2.2 Ciele práce

Dokončená práca by mala splňovať nasledujúce požiadavky:

- rozmiestniť prvky logického obvodu popísané v súbore typu bench,
- zoptimalizovať ich rozmiestnenie,
- prepojiť ich príslušnými spojeniami (*routing*),
- vyexportovať výslednú grafickú podobu logického obvodu do súboru typu xml pre program INKS. odôvodním

2.3 Štruktúra práce

Ako prvú časť mojej práce rozoberiem niektoré existujúce algoritmy z grafového chápania obvodu. Z nich vytvorím algoritmus pre rozmiestnenie a optimalizáciu jeho prvkov a odôvodním jeho správnosť, prípadne prediskutujem jeho nedostatky a navrhnem vylepšenia, s ktorými bude možno dosiahnuť ešte lepších výsledkov.

Kapitola 3

Analýza a návrh riešenia

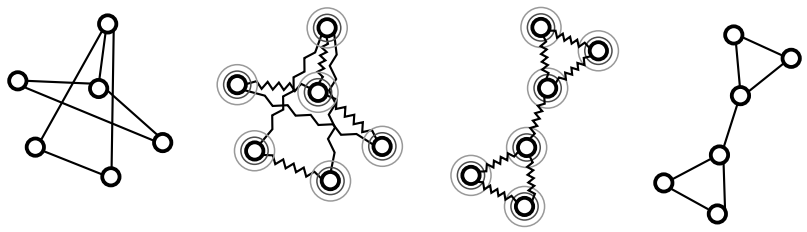
V popise obvodu budeme mať zoznam pravidiel, ktoré tento obvod charakterizujú. Tieto pravidlá reprezentujú hradlá a bloky (zoskupenia hradiel chápané ako jeden prvok). Potrebujeme teda nájsť spôsob, ako uložiť do priestoru prvky so vzájomnými vzťahmi. Prvky a vzťahy sa dajú jednoducho charakterizovať pomocou grafu, kde prvky budú uzly a ich vzájomné vzťahy budú reprezentovať hrany. Toto pojatie logického obvodu, vedie k riešeniu pomocou teórie grafov, konkrétne vykresľovania grafov do roviny.

3.1 Grafové algoritmy - uzly

3.1.1 Silovo-založené algoritmy

Graf je možno chápať ako fyzikálny systém. Ak by sme nahradili hrany za pružiny a uzly za elektricky nabité častice, mohli by sme na usporiadanie grafu aplikovať fyzikálne zákony(Hookov zákon pre pružiny a Coulombov zákon pre elektricky nabité častice), a nechať graf, nech sa usporiadá sám. Toto usporiadanie zaručí, že hrany budú viac-menej rovnakej dĺžky a budú sa minimálne krížiť 3.1.1. [4]

Pri zobrazovaní logického obvodu ale tento algoritmus nemožno použiť. Od zobrazenia očakávame čiatateľnosť a pokiaľ možno aj rovnomerné logické rozloženie. Nezaručí nám teda úhľadné rozloženie, aké sa očakáva od logického obvodu.



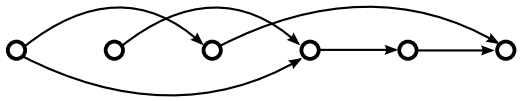
Obr. 3.1: Vizuálna ukážka fungovania silovo-založeného algoritmu

3.1.2 Topologické usporiadanie uzlov

Topologické usporiadanie uzlov 3.1.2 je možné na acyklickom grafe. Je to usporiadanie, kde sú všetky prvky v jednej horizontálnej úrovni a orientované hrany tohoto grafu smerujú rovnakým smerom. Z tejto definície vyplýva nutnosť acyklyčnosti [1]. Topologické usporiadanie svojou charakteristikou velmi dobre charakterizuje logický obvod bez spätných väzieb, kde vstupy sú na jednej horizontálnej strane obvodu a výstupy na druhej. Je to vlastne postupné uloženie prvkov od vstupov až k výstupom do jednej horizontálnej roviny.

Topologické usporiadanie je možné vytvoriť dvomi spôsobmi :

- použitím a upravením DFS algoritmu (prehľadávanie grafu do hĺbky),
- elimináciou koreňov.



Obr. 3.2: Topologické usporiadanie uzlov grafu

3.1.2.1 Upravenie DFS algoritmu

Pri prehľadávaní grafu si každý uzatvorený uzol u uložíme do zoznamu S. Po ukončení prehľadávania budeme mať v S topologicky usporiadané uzly grafu.

Upravený DFS algoritmus potom bude vyzerať následovne:

- $S = \emptyset$,
- v DFS pri uzavretí uzlu u, pridaj tento uzol na začiatok zoznamu $S = S \cup \{u\}$,
- v zozname S sa nachádzajú uzly zoradené podľa požiadaviek topologického usporiadania.

3.1.2.2 Eliminácia koreňov

Budeme prechádzať postupne všetky uzly grafu $G < U, H >$. Pri výskyte uzlu u, pre ktorý platí $\delta^+[u] = 0$ (uzol u je koreňom grafu), uložíme tento uzol do fronty F a upravíme stupne jeho nasledovníkov.

Tento postup aplikujeme v algoritme:

- $F = \emptyset$,

- prehľadávaj všetky uzly grafu
- ak pre uzol u platí $\delta^+[u] = 0$, tak $F.push(u),\ \forall u' \in Adj(u) : \delta^+[u'] - -$,
- ak platí $U \sim \emptyset$, tak F obsahuje uzly zoradené podľa požiadaviek topologického usporiadania.

Pre oba algoritmy je dôležitá acykličnosť grafu. V logických obvodoch sa však môžu nachádzať aj cykly v podobe spätných väzieb. Ak by sa nám podarilo odstrániť spätné väzby, dostali by sme usporiadanie prvkov také, aby sme pri jeho upravení (vertikálnom poziciovaní) dostali požadovanú postupnosť prvkov.

3.2 Grafové algoritmy - hrany

3.2.1 Routovanie spojov

Zatiaľ čo v predošlej podkapitole sme sa zaoberali algoritmami na uloženie prvkov, v tejto podkapitole si ukážeme algoritmus, ktorý nám zaručí optimálne rozmiestnenie vodičov. Zvolil som si algoritmus použitý v bakalárskej práci p. Škorpila [2] . Je to algoritmus A*. Jeho aplikovaním dokázal p. Škorpil vygenerovať efektívne a optimálne grafické reprezentácie vodičov.

3.2.1.1 A* algoritmus

Algoritmus A* je tzv. informovaný algoritmus. Znamená to, že pri vyhľadávaní cesty hľadá najprv tam, kde sa cesta zdá byť najbližšie cieľovému uzlu. Toto prioritné vyhľadávanie je určené funkciou $f(p) = g(p) + h(p)$, kde funkcia $g(p)$ udáva cenu cesty a odhadová funkcia $h(p)$ udáva približnú vzdialenosť posledného uzlu cesty od riešenia.

Samotný algorimus A* má nasledovný priebeh:

1. na počiatnu si inicializujeme množinu uzlov $CLOSED = \emptyset$ a množinu ciest $Q = \{< s >\}$,
2. pokiaľ $Q \sim \emptyset$ skončíme, pretože riešenie nebolo nájdené,
3. z Q vyberieme cestu $p < \ldots, u >$ s najmenšou hodnotou funkcie $f(p)$,
4. $CLOSED = CLOSED \cup u$, pokiaľ u je cieľový uzol skončíme, a riešenie sa nachádza v p,
5. pre $\forall v : v \in Adj(u), v \notin CLOSED$ pridaj do Q cestu $p < \ldots, u, v >$ a pokračuj bodom 2).

Q nám reprezentuje zoznem ciest, z ktorých jedna je tá najlacnejšia a zároveň najbližšie k cielu. Z Q postupne vyberáme cesty z najmenšou hodnotou funkcie $f(p)$, ktorá znamená vzdialenosť a cenové ohodnotenie cesty. Postupným testovaním okolia týchto ciest vytvárame nové cesty, z nich vyberáme opäť tie s najmenšou hodnotou funlcie $f(p)$. Algoritmus ukončíme, ak v Q nezostane ani jedna cesta. To znamená, že ani jedna z možných ciest nedokázala prepojiť počiatočný uzol s a koncový uzol t. Algoritmus tiež končí ak posledný uzol cesty $p \in Q$ je t. Potom sme našli vhodnú cestu pre uzly s, t.

Algoritmus s odhadovaním a hodnotením cesty je dostatočne rýchly, pretože "intuitívne" dokáže vyhľadávať vždy v okolí cesty s najnižším cenovým ohodnotením.

Kapitola 4

Realizácia

4.1 Načítanie súboru

Načítavanie popisu obvodu zo súboru bench je vytvorené pomocou automatu. Jeho pravidlá sú stanovené gramatikou formátu bench.

počiatočný neterminál	S	$\rightarrow$	IS|OS| RS|ε
vstup	I	$\rightarrow$	"INPUT "n
výstup	O	$\rightarrow$	"OUTPUT "n
pravidlo	R	$\rightarrow$	n"="n"("n{ ","n} ")"
názov (terminál)	n	=	[a-zA-Z0-9]{ [a-zA-Z0-9]}

Tabuľka 4.1: Pravidlá gramatiky bench

Pri načítavaní súboru sa vytvára objektová štruktúra reprezentujúca obvod. Vytvorí sa množina I, ktorej prvky reprezentujú vstupy, O s reprezentáciou výstupov a R, ktorej prvky sú $r =< t, I_r >$, kde r je pravidlo, t je jeho typ a I_R je množina pravidiel, ktoré sú jeho argumentmi.

Ďalej si vo vnútornej štruktúre zavedieme pojem prvok $p < r, x, y, w, h, I_y, O_y >$ ako repezentáciu pravidla s jeho rozmerovými (výška h a šírka w), polohovými (súradnice ľavého horného rohu x a y) a grafickými vlastnosťami (zoznam vertikálnych súradníc vstupov I_y a výstupov O_y) a jeho množinu P.

Priradenie prvku k pravidlu vyjadríme ako zobrazenie $\pi : R \rightarrow P$. Pre zobrazenie π si zavedieme nasledujúce pravidlá:

- horizontálna poloha prvku $p < r, \ldots >, r < t, I >$ musí byť väčšia ako najväčšia horizontálna poloha akéhokoľvek z prvkov $p' < r', \ldots >$, pre ktoré platí $r' \in I$,
- vertikálna poloha prvku $p < r, x, \ldots >$ musí byť väčšia ako najväčšia vertikálna poloha akéhokoľvek z prvkov $p' < r', x', \ldots >$, pre ktoré platí $x = x'$,

- rozmery prvku w, h a množiny vertikálnych polôch vstupov I_y a výstupov O_y sú definované osobitne pre každý typ t pravidla $r < r, I_r >$. Tieto definície sú uložené v súboroch typu des v adresári pieces. Sú zhodné s charakteristikami prvkov pre program INKS, kde je popísaná aj ich vizuálna podoba.

4.2 Prvotné umiestnenie

Druhým krokom pri kreslení obvodu je jeho prvotné umiestnenie. Použitie silovozaložený algoritmus nie je vhodné. Jeho počiatočné usporiadanie je náhodné a tak nám nezaručí požadovaný výsledok. Ako podklad pre svoj algoritmus som si zoberal topologické usporiadanie uzlov. Ako uzly budeme chápať pravidlá $r < t, I_r >$resp. prvky a ako hrany vzťah $r' \in I_r$. Ďalšou úpravou bude zmena horizontálneho 1–dimenzionálneho usporiadania na 2–dimenzionálne podľa pravidiel zobrazenia π.

Algoritmus prvotného umiestnenia zobrazí pravidlo r do prvku p ak je splnené pravidlo konektivity, že pravidlá, reprezentujúce jeho parametre, už majú svoj obrav v množine P. Inými slovami, ak obvod reprezentujeme grafom, budeme z tohoto grafu postupne odoberať koreňe (pravidlá zo svojím obrazom v P) a dostaneme tak topologiské usporiadanie uzlov.

Ako však vyplíva z definície topologického usporiadania uzlov, graf musí byť acyklický. Acykličnosť bude musieť zaručiť užívateľ. Do obvodu bude možné pridať spätné väzby až v programe INKS.

Algoritmus 1 primalPlace

```
Proc.push(∀r : r ∈ R)
repeat
  connectOk=false
  while Proc ≁ ∅ do
    if !connect(r) then
      Wait.push(r)
    else
      connectOk=true
      place(r)
    end if
  end while
  Proc.push(∀r : r ∈ Wait)
  Wait = ∅
until Proc ≁ ∅ ∧ connectOk
x_max = 0
for all p < r, x, ... >∈ P do
  if r ∼ o ∈ O ∧ x_max < x then
    x_max = x
  end if
end for
for all p < r, x, ... >∈ P do
  if r ∼ o ∈ O then
    x = x_max
  end if
end for
```

Algoritmus 2 connect $(r < t, I_r >)$

```
for all r_i ∈ I_r do
  if ∀p < r', ... >∈ P : r' ≠ r_i then
    return false
  end if
end for
return true
```

Algoritmus 3 place$(r < t, I_r >)$

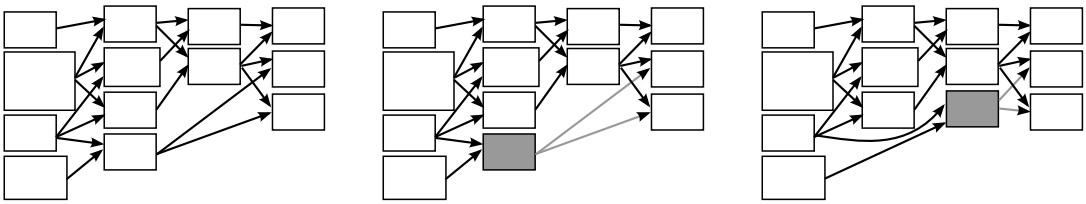
```
p = π(r)
P = P ∪ p
```

4.3 Optimalizácie umiestnenia prvkov

Rozmiestnenie prvkov pomocou prvotného umiestnenia väčšinou nie je optimálne. Výnimkou sú hlavne jednoduché logické obvody do cca. 10 hradiel a blokov. Z tohoto dôvodu by sme mali usporiadanie hradiel pozmeniť, aby bolo úhladnejšie. Najprv skontrolujeme ich horizontálnu a potom vertikálnu polohu. Moje algoritmy pre optimalizáciu len zohladňujú vzájomnú pozíciu hradiel a pokúšajú sa odstrániť veľké skupiny a prehodené prvky vzhľadom na pozíciu ich vstupných prvkov.

4.3.1 Horizontálna optimalizácia prvkov

V tejto optimalizácii sa snažíme o vyvážiť počet prvkov vo vertikálnych vrstvách navzájom. Vyváženie medzi vrstvou l_1 a l_2 robíme tak, že nájdeme vhodný prvok (kandidát) vo vrstve l_1 a premiestnime ho do vrstvy l_2. Kandidát je taký prvok, ktorý nie je obrazom ani jedného pravidla z množiny parametrov pravidla každého prvku z vrstvy l_2.



Obr. 4.1: Ukážka horizontálnej optimalizácie prvkov

Rozmiestnené prvky si musíme najprv zmapovať, aby sme zistili, ktoré prvky sú v spoločnej hladine. Zavedieme si preto pojem mapa hladiny l_i $M_i \subset P$. Pre mapu hladiny platí

$$M_i = \{\forall p_1 < r_1, x_1, \ldots >, p_2 < r_2, x_2, \ldots > \in P : x_1 = x_2\}.$$

Funckia `mapLayerHoriz` vytvorí množiny M_i.Pre vyváženie vrstiev budeme potrebovať vedieť počet prvkov v jednotlivýh vrstvách. Táto hodnota sa bude nachádzať v mape hustoty

$$D = \{d_1, d_2, \ldots : d_i = |M_i|\}.$$

Funkcia `mapDensity` vytvorí množinu D.

Po ukončení vyváženia budú prvky podľa možnosí rovnomerne rozložené v horizontálnych vrstvách.

Algoritmus 4 horizontalPlaceOptm

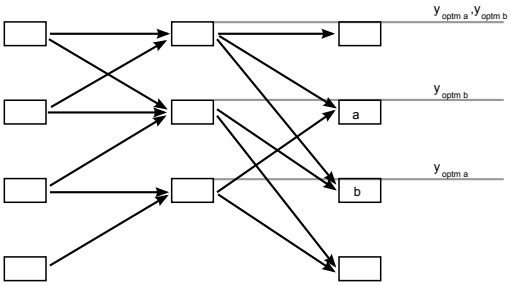
```
change=true
mapLayerHoriz
while change do
  mapDensity
  change=false
  for all d_i, d_{i+1} ∈ D do
    if d_i > d_{i+1} ∧ d_i > d_{i+1} + 1 then
      change = change ∨ moveCandidate(i)
    end if
  end for
end while
```

Algoritmus 5 moveCandidate(i)

```
for all p_cnd < r_cnd, x_cnd, ... >∈ M_i do
  cndOk=true
  for all p < r_p < t, I >, ... >∈ M_{i+1} do
    for all r ∈ I do
      if π(r) = p_cnd then
        cndOk=false
      end if
    end for
  end for
  if cndOk then
    M_i = M_i − p_cnd
    x_cnd = x; p < r, x >∈ M_{i+1}
    M_{i+1} = M_{i+1} ∪ p_cnd
  end if
end for
```

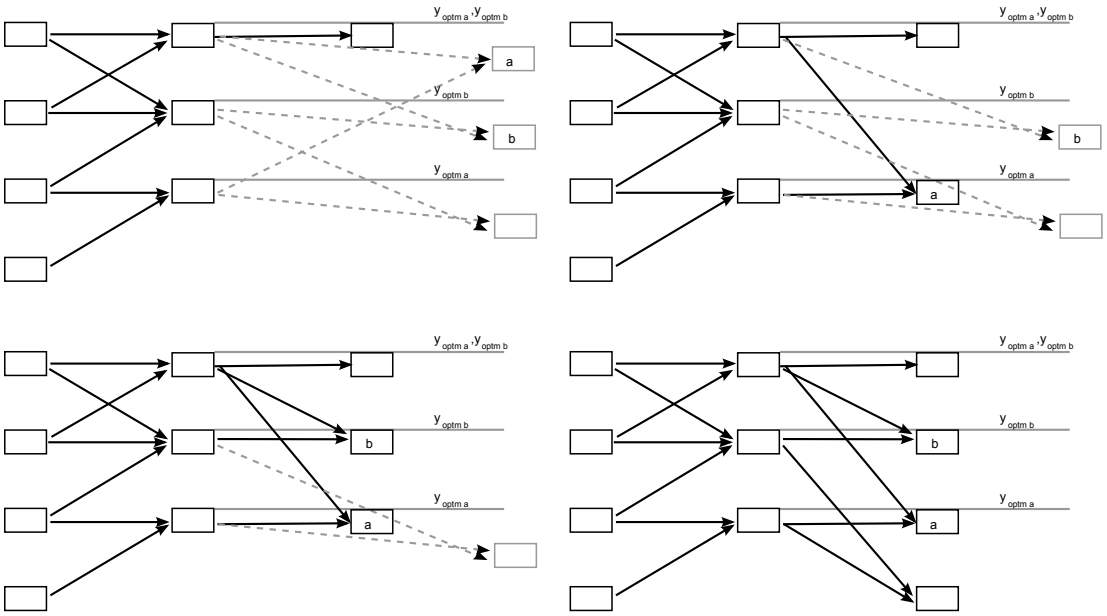
4.3.2 Vertikálna optimalizácia prvkov

Aby sa predišlo zbytočnému kríženiu prvkov, ktorého príčina je ich neprávne vertikálne uloženie, zoptimalizujeme ich pozície aj vertikálne. Filozofia vertikálneho uloženia je umiestniť prvok čo najbližšie k pozícii, ktorá mu vyhovuje najviac z hladiska vstupov (prvkov reprezentujúcich parametre pravidla prvku). Jeho optimálna pozícia y_{optm} je vždy na úrovni s jedným jeho vstupným prvkom.



Obr. 4.2: Obvod pred aplikáciou vertikálnej optimalizácie

Na obrázku 4.2 je obvod pred aplikáciou vertikálnej optimalizácie. Vidíme, že prvky a a b nie su ani na jednej zo svojich optimálnych pozícií. Môžeme preto tieto prvku medzi sebou vymeniť a dostaneme optimálnejšie usporiadanie s menším počtom krížiacich sa spojov. Inými slovami, pokúsime sa prvky znovu uložiť tak, aby sme rešpektovali ich optimálne pozície 4.3. Ak sa nedá dostať na optimálnu pozíciu, umiestnime ho pod všetky už uložené prvky.



Obr. 4.3: Aplikácia vertikálnej optimalizácie

4.4 Routovanie

Pre routovanie som použil úpravu algoritmu A* popísaného p. Škorpilom v jeho bakalárskej práci [2]. Pre tento postup je potrebné si vytvoriť spojové uzly na vstupoch a výstupoch hradiel (blokov). Potom je už vedenie spoja len vyhľadanie cesty medzi uzlami s a t podľa štandartného algoritmu s ohľadom na prekážky. Odhadovú funkciu ceny cesty $g(p)$ som stanovil ako súčet cien buniek, cien ohybov a cien krížení ciest.

4.4.1 Vytváranie spojových uzlov

Spojové uzly sa vytvárajú ako prepojenia hradlo–spoj alebo ako kríženia spoj–spoj. Pri type hradlo–spoj sa uzly vytvoria pre routovaním ciest a to na vstupoch a výstupoch všetkých hradiel a vstupov. Takto vytvorené uzly slúžia ako počiatočné a koncové uzly ciest medzi hradlami. Spojový uzol môže na seba viazať najviac 4 spoje. Pri uzloch typu hradlo–spoj je to len 1 spoj, pretože z hradla je vždy preferovaný smer spoja od hradla. Preto pri prepájaní ďalšieho spoja k hradlu je pri nedostatku uzlov vytvorený ďalší uzol. Takto vytvorené uzly reprezentujú typ spoj–spoj.

4.4.2 Vytvorenie spojení

Spojenie je cesta medzi spojovými uzlami s a t, ktorá sa skladá z bodov popísaných súradnicami a taktiež smeru, z ktorého sa do daného bodu dá po ceste prísť. Keďže vytvorené cesty by s použitím A* algoritmu obsahovali veľké množstvo bodov (každý bod prieniku mriežky a cesty), zavedieme funkciu za zlučovanie bodov na spoločnej priamke. Toto zlučovanie redukuje popis cesty len na zlomové body, v ktorých sa cesta ohýba. Toto spojenie vykonáme ihneď po nájdení cesty, aby sa zložitosť algoritmu na hľadanie križení ciest rádovo znížila.

4.5 Vykreslenie obvodu

Po umiestnení a spojení prvkov je obvod pripravený k vykresleniu. Pretože program by mal spolupracovať s programom INKS od p. Pastora [3] a upraveného p. Škorpilom [2], využijeme jeho programové schopnosti. Obvod, zatiaľ stále uložený len ako objektový model, vyexportujeme do xml formátu podľa špecifikácie prác zameraných na INKS. Tento export nám vytvorí projekt so schémou obvodu. Pomocou tohoto exportu a programu INKS sme schopný vytvoriť zo schémy vektorový súbor vo formáte pdf alebo ps.

Kapitola 5

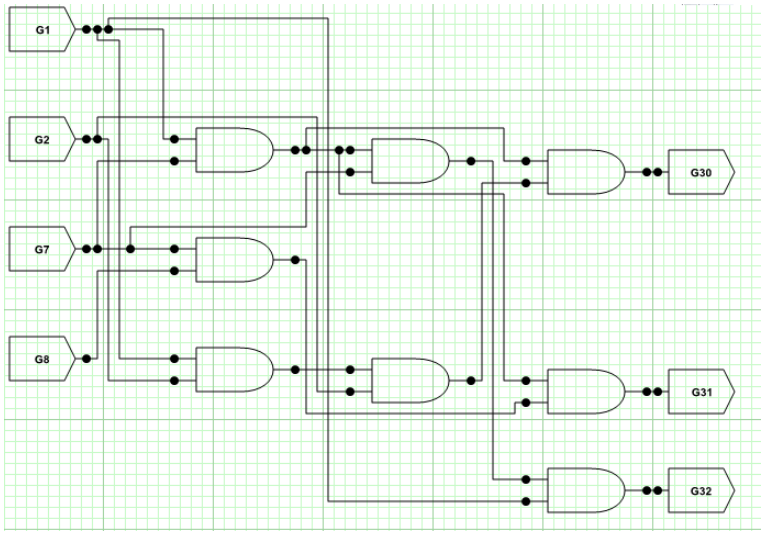
Testovanie

Testovanie som robil s testovacím súborom, na ktorom bolo možné vykonať všetky optimalizácie prvkov.

Výpis testovacieho súboru `test.bench`:

```
#The test .bench file to BETOPOX project

#inputs

INPUT G1
INPUT G2
INPUT G7
INPUT G8

#outputs

OUTPUT G30
OUTPUT G31
OUTPUT G32

#rules

G10=AND(G1,G2)
G11=AND(G1,G7)
G12=AND(G7,G8)
G20=AND(G1,G2)
G21=AND(G11,G8)
G30=AND(G10,G11)
G31=AND(G11,G12)
G32=AND(G20,G21)
```

Z tohoto súboru sa pomocou programu vygeneroval projekt `test` pre INKS. Tento projekt obsahuje schému `bench`, ktorá je grafickou reprezentáciou popísaného obvodu.



Obr. 5.1: Grafická podoba obvodu

Program Betopox teda dokázal vygenerovať schému pre INKS. Patrične rozmiestnil vstupy, hradlá a výstupy. Vytvoril ich prepojenia a vodivé kríženia spojov.

Z grafickej podoby môžeme vidieť, že algoritmus pre rozmiestnenie prvkov má vážne rezervy. Vo vertikálnom smere by sa prvky mali ukladať na volné miesta a nie až na koniec. Umiestňovanie dodatočných uzlov tiež nieje vhodné. Bolo by vhodné využiť každý uzol, pokiaľ má volný práve požadovaný smer.

Vytvorenie grafickej reprezentácie je úplné. Užívateľ však pre prehľadnejšie znázornenie sa bude musieť zapojiť a teto obvod si upraviť.

Kapitola 6

Záver

Práca bola navrhnutá ako doplnok k programu INKS pre zobrazovanie bench súborov. Jej zadanie bolo zobraziť logický obvod, čo sa aj podarilo. V spolupráci s INKS dokáže tento obvod exportovať aj do formátov pdf a ps. Poskytla pohľad do filozofie algoritmov pre ukladanie prvkov a vytváranie obvodov. Tento pohľad podložila teoretickými použiteľnými aj pri nasledujúcich verziách tejto práce.

6.1 Pokračovanie práce

Doplnenie a pokračovanie tejto práce by mohlo predstavovať zameranie sa na pokročilé algoritmy umiestňovania prvkov a ich následného prepojovania. Nasledujúce doplnky majú v mojej práci dostatočný základ pre ďalší rozvoj. Rezervy práce však spočívajú v jej neoptimalizovanom pojatí vykreslovania a v neschopnosti vykresliť spätné väzby. Zaujímavou možnosťou zmeny mojej práce by bolo začlenenie jej do INKS ako doplnku pre import schém z formátu bench.

Literatúra

[1] J. Kolář. *Teoretická informatika.* Česká informatická společnost, 2004.

[2] R. Škorpil. *Interaktivní nástroj pro kreslení schémat logických obvodů.* ČVUT v Praze, 2008.

[3] Pastor. *Interaktivní nástroj pro kreslení schémat logických obvodů.* ČVUT v Praze, 2006.

[4] Network workbench. `https://nwb.slis.indiana.edu/community/?n=VisualizeData.ForceDirected`.

Dodatok A

Obsah priloženého CD

bakalarskaPraca.pdf
: bakalárska práca v elektronickej podobe,

readme.txt
: užívateľská príručka,

bin/
: spustiteľná verzia programu Betopox a INKS,

src/
: zdrojové kódy programu Betopox.

Dodatok B

Užívateľská príručka

Program je spustiteľný pomocou príkazového riadku pomocou príkazu `betopox fileName`, kde `fileName` je meno súboru bench bez prípony.

Výstup programu je súbor `fileName.xml`.