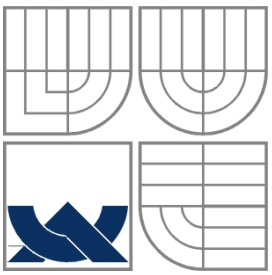
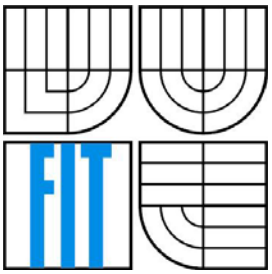


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

EVOLUČNÍ NÁVRH OBVODŮ NA ÚROVNI TRANZISTORŮ

EVOLUTIONARY DESIGN OF CIRCUITS ON TRANSISTOR LEVEL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. LUDĚK ŽALOUDEK

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. LUKÁŠ SEKANINA, Ph.D.

BRNO 2007

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2006/2007

Zadání diplomové práce

Řešitel: **Žaloudek Luděk, Bc.**
Obor: Počítačové systémy a sítě
Téma: **Evoluční návrh obvodů na úrovni tranzistorů**
Kategorie: Návrh číslicových systémů

Pokyny:

1. Seznamte se s problematikou návrhu elektronických obvodů pomocí programů typu PSpice.
2. Seznamte se s problematikou evolučního návrhu číslicových obvodů na úrovni tranzistorů.
3. Zpracujte studii na téma uvedené v bodech 1 a 2.
4. Navrhněte a implementujte vývojový systém pro evoluční návrh číslicových obvodů na úrovni tranzistorů.
5. Na zvolených úlohách demonstrujte funkčnost navrženého systému. Zaměřte se zejména na návrh základních logických členů.
6. Shrňte dosažené výsledky.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části diplomového projektu je požadováno:

- Splnění prvních tří bodů zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Sekanina Lukáš, doc. Ing., Ph.D.**, UPSY FIT VUT
Datum zadání: 28. února 2006
Datum odevzdání: 22. května 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. **Pan**

Jméno a příjmení: **Bc. Luděk Žaloudek**
Id studenta: 49340
Bytem: K lávce 14, 281 23 Starý Kolín
Narozen: 20. 12. 1982, Pardubice
(dále jen "autor")

a

2. **Vysoké učení technické v Brně**

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
diplomová práce

Název VŠKP: Evoluční návrh obvodů na úrovni tranzistorů
Vedoucí/školitel VŠKP: Sekanina Lukáš, doc. Ing., Ph.D.
Ústav: Ústav počítačových systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:

tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Abstrakt

Tato práce se zabývá evolučním návrhem elektronických obvodů na úrovni tranzistorů se zaměřením na číslicové obvody. Popisuje teoretické základy pro evoluční návrh obvodů na výpočetních systémech včetně vysvětlení evolučních algoritmů genetického programování a evolučních strategií, možných úrovní návrhu elektronických obvodů, přehledu technologie CMOS a nejdůležitějších evolučních metod pro návrh obvodů, jako jsou development a kartézské genetické programování (CGP). Dále je uvedena nová metoda návrhu číslicových obvodů s tranzistory založená na CGP a je představen vývojový systém, který tuto metodu využívá. Na závěr jsou popsány a vyhodnoceny experimenty provedené se systémem.

Klíčová slova

evoluční algoritmus, evoluční návrh, číslicové obvody, tranzistory, CMOS, development, kartézské genetické programování

Abstract

This project deals with evolutionary design of electronic circuits with an emphasis on digital circuits. It describes the theoretical basics for the evolutionary design of circuits on computer systems, including the explanation of Genetic Programming and Evolutionary Strategies, possible design levels of electronic circuits, CMOS technology overview, also the overview of the most important evolutionary circuits design methods like development and Cartesian Genetic Programming. Next introduced is a new method of digital circuits design on the transistor level, which is based on CGP. Also a design system using this new method is introduced. Finally, the experiments performed with this system are described and evaluated.

Keywords

evolutionary algorithm, evolutionary design, digital circuits, transistors, CMOS, development, Cartesian Genetic Programming

Citace

Žaloudek Luděk: Evoluční návrh obvodů na úrovni tranzistorů. Brno, 2007, diplomová práce, FIT VUT v Brně.

Evoluční návrh obvodů na úrovni tranzistorů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením doc. Ing. Lukáše Sekaniny, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Luděk Žaloudek
22.5.2007

Poděkování

Rád bych poděkoval doc. Ing. Lukáši Sekaninovi, Ph.D. za jeho odbornou pomoc při zpracování této práce a poskytnutou literaturu.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Cílem této práce je seznámit čtenáře s problematikou evolučního návrhu číslicových elektronických obvodů na úrovni tranzistorů a předvést návrhový systém, který je právě k tomuto účelu určen.

S novými technologiemi a stále dokonalejší miniaturizací se ve vývoji složitých číslicových systémů prosazují vyšší abstrakce návrhu. Zároveň ustupuje užívání nižších úrovní abstrakce při návrhu. Důvodů pro tento trend je několik, ale mezi nejdůležitější patří existence výkonných CAD systémů, poměrně dostupná realizace obvodu pomocí nejrůznějších programovatelných integrovaných obvodů obsahujících předpřipravené stavební bloky a také spolehlivost takových řešení. S dnešními stavebními bloky, mezi nimiž jsou logická hradla, multiplexory či rozličné typy pamětí, jsme schopni realizovat téměř jakýkoli číslicový obvod. Návrhář má rovněž k dispozici komponenty realizované uvnitř jediného čipu či dokonce několik takových systémů v jediném integrovaném obvodu. Problém nastane, chceme-li se zabývat nějakým specializovanějším oborem, jako jsou například polymorfní obvody [19], kde není možné použít technik klasického návrhu, či pokud se zabýváme například optimalizací již existujících číslicových obvodů, protože chceme zvýšit jejich výkon či snížit cenu.

Chceme-li dosáhnout malých rozměrů a velkého výkonu u složitějších obvodů, musíme se oprostit od stavebních bloků, na něž jsme zvyklí z číslicového návrhu a vrátit se zpět na analogovou úroveň, protože i to nejjednodušší logické hradlo je uvnitř složeno z tranzistorů a dalších součástek. S tímto řešením ale vyvstává původní problém analogových obvodů: Jejich návrh není triviální a je třeba mnoha zkušeností, znalostí a času, abychom dosáhli požadovaného výsledku, přičemž schopných analogových inženýrů je nedostatek. V tuto chvíli přichází na řadu evoluční návrh elektronických obvodů, což je metoda, která nám s malým množstvím znalostí či s nepatrným úsilím programátora/návrháře dokáže syntetizovat originální návrhy v přijatelném čase [10]. Abychom však pochopili problematiku automatizovaného návrhu, je třeba nejprve porozumět jejím jednotlivým částem.

Proto v druhé kapitole zmiňuji návrhové systémy typu SPICE, což jsou softwarové simulační nástroje pro návrh analogových obvodů, které se samozřejmě dají použít i pro návrh číslicových obvodů, které nás v této práci zajímají. Třetí kapitola obsahuje stručný popis evolučních algoritmů a speciálně podskupin zvaných genetické programování a evoluční strategie. O tom, na jakých úrovních abstrakce můžeme navrhovat hardware, je pojednáno v kapitole čtyři. V kapitole pět jsou popsány hlavní vlastnosti technologie CMOS, která je současným trendem v návrhu číslicových obvodů na úrovni tranzistorů. Šestá kapitola je soustředěna na některé z nejdůležitějších evolučních metod návrhu elektronických obvodů.

Praktický popis návrhu a implementace evolučního systému pro návrh číslicových obvodů na úrovni tranzistorů je obsažen v sedmé kapitole. Osmá kapitola představuje experimenty provedené na realizovaném návrhovém systému a jejich výsledky. V deváté kapitole jsou výsledky vyhodnoceny a jsou shrnuty vlastnosti návrhového systému. Desátou kapitolou je závěr, kde jsou shrnuty výsledky celé práce, její přínos a potenciální možnosti pokračování.

2 Systémy typu SPICE

2.1 Obecné vlastnosti systémů typu SPICE

Systémy rodiny SPICE jsou softwarové nástroje pro simulaci elektronických obvodů. Následující podkapitola je věnována tomu, co dnes takové systémy mohou nabídnout.
Nástroje pro simulaci elektronických obvodů mají několik možných funkcí, ale ta hlavní spočívá jistě v ověřování elektronických obvodů. Nejen že ušetří práci inženýrům, kteří nejsou nuceni každý navržený obvod sestavovat ze skutečných součástek, ale velmi důležité je to, že ušetří spoustu peněz při návrhu integrovaných obvodů. To se projevuje větší měrou u obvodů vyráběných pokročilejšími technologiemi, protože s vyšší hustotou součástek na čipu roste i cena, za kterou se vyrábí masky pro výrobu obvodů. U složitějších čipů může jít až o miliony amerických dolarů. Bohužel, simulace není nikdy dokonalá, protože nemusí brát v úvahu všechny fyzikální vlastnosti použitých materiálů, ale použití kvalitního softwaru může rizika minimalizovat. Drtivá většina výzkumníků pracuje právě s nástroji rodiny SPICE, což podporuje tvrzení, že patří k těm nejlepším.

2.1.1 Původ a rozšíření SPICE

SPICE vzniknul jako projekt na University of California at Berkeley v USA počátkem sedmdesátých let. Postupně se rozvíjel a objevilo se několik verzí psaných nejprve ve Fortranu a poté v C [30]. Simulace probíhala tak, že se různé charakteristiky obvodu postupně počítaly v jednotlivých uzlech tak, jak docházelo k zjednodušování problému. Algoritmy zahrnovaly systémy lineárních rovnic, pro nelineární části obvodu byly linearizovány Newton-Raphsonovým algoritmem a přechodová analýza byla řešena Gearovým integračním algoritmem. Je pravděpodobné, že na podobných principech funguje SPICE dodnes. Později byly přidány další algoritmy a vznikla odnož XSPICE, která dokáže simulovat abstraktní modely a smíšené analogově-digitální obvody.
Dnes existují otevřené i komerční verze na většinu hlavních výpočetních systémů. Mezi nejdůležitější otevřené představitele patří ngspice, tclspice či XSpice, zástupci komerčních systémů jsou PSpice (t.č. užíván na FIT v rámci balíku OrCAD), HSpice či MultiSIM [30].

2.1.2 Možnosti a omezení SPICE

Dnešní systémy podporující SPICE se soustředí hlavně na komfort uživatele a dodatečné funkce. Samotné simulační programy tedy většinou náleží k velkým softwarovým balíkům (zejména v komerční podobě), které umožňují mnohé další funkce, jako návrh desek plošných spojů, vizuální editaci schémat, smíšenou simulaci A/D atd. Je obvyklé, že většina nástrojů dostává simulovaný obvod v podobě tzv. SPICE Netlistu, což je textový soubor obsahující všechny potřebné informace o obvodu. Důležitou položkou jsou knihovny a modely používaných součástek, zahrnující různé výrobní technologie integrace. Většina otevřených systémů používá podobnou či stejnou syntaxi, problém omezující přenositelnost může nastat u komerčních systémů, kde jazyk Netlistu často obsahuje proprietární rozšíření. Lepší systémy obsahují funkce na import a export jiných formátů.
Následující seznam shrnuje možnosti běžně dostupné v SPICE systémech [8, 16, 17]:

- Simulace analogových obvodů,

- simulace číslicových obvodů,
- simulace smíšených obvodů,
- export/import cizích formátů,
- intuitivní grafická editace schémat,
- textový vstup a výstup,
- grafický výstup,
- znouvupoužitelnost kódu a schémat,
- automatická integrace FPGA a jiných PLD (programovatelná logická zařízení) a
- internetová či telefonní technická podpora včetně diskusních skupin.

Omezení SPICE systémů obvykle vyplývají z časové prodlevy mezi uvedením nové technologie na pole elektroniky a vytvořením příslušné knihovny obsahující specifikace nových součástek, případně z licenčních práv na příslušný software. Další, poněkud vážnější limitace jsou způsobeny nedokonalostí simulátorů. Mezi tyto problémy patří linearizace nelineárních problémů nebo nedokonalá reprezentace operačních zesilovačů. Mezi omezení je nutné zahrnout i skutečnost, že pro uživatele je velmi obtížné vložit do simulátoru parametry reálného obvodu už jen z toho důvodu, že obvykle všechny nezná nebo že jich je příliš velké množství. Rovněž je možné, že simulátor nemusí umět všechny parametry simulovat. V úvahu je nutné vzít i další problémy, které vznikají při evolučním intrinsickém a extrinsickém návrhu [31]. O nich se zmiňuji v další kapitole.

2.2 Stručný úvod do simulátoru ngspice

V této podkapitole bych se chtěl věnovat jednomu konkrétnímu příkladu simulátoru, který je založen na SPICE. Jde o systém vyvíjený skupinou gEDA (skupina zabývající se automatizací návrhu elektroniky) a jeho základem jsou tři otevřené softwarové balíčky: Spice3f5, Cider1b1 a XSpice [16]. Výhodou ngspice je zpětná kompatibilita s předchozími verzemi jazyka Spice [7].

2.2.1 Přehled syntaxe ngpice netlistů

Samotný vstupní soubor pro ngspice má textovou podobu a má v zásadě dvě části: Netlist, tedy popis simulovaného obvodu, a dále příkazovou část, kde určujeme typ analýzy.

První řádek vstupního souboru je **nadpis**, poslední řádek označuje **konec** a musí obsahovat `.END` případně následovaný komentářem.

Prvky obvodu, či jinak **součástky,** se obvykle užívají tak, že uvedeme první písmeno součástky (R pro rezistor, C pro kondenzátor, L pro cívku, V pro zdroj napětí, I pro zdroj proudu, SIN pro generátor sinusoidy atd.) následované dalšími znaky určenými pro pojmenování součástky. Následuje seznam parametrů, který podle nejrůznějších typů součástek a modelů může nabývat rozličných podob. Pro to zde bohužel není místo, více podrobností naleznete v [25].

V případě, že zadáváme číselné parametry prvků, je možné je nahradit znakovými **zkratkami**. Následující tabulka obsahuje výčet takových zkratek:

Tabulka 2.1: Seznam zkratek pro mocniny deseti

T = 10^{12}	G = 10^{9}	Meg = 10^{6}	K = 10^{3}	mil = 25,4x10^{-6}
m = 10^{-3}	u = 10^{-6}	n = 10^{-9}	p = 10^{-12}	f = 10^{-15}

Písmena ihned následující za číselným výrazem jsou ignorována, takže je třeba vložit mezeru, chceme-li se tomu vyhnout.

Mezi jednotlivými prvky obvodu se nachází referenční body označované jako **uzly**. Názvy uzlů jsou (na rozdíl od Spice2) řetězce, nikoliv čísla. Řetězce „01“ a „1“ představují tedy rozdílné uzly. **Referenční uzel** musí být přítomen a musí být nazván „0“. Obvod nesmí obsahovat smyčky zdrojů napětí či cívek a nesmít obsahovat zkratované proudové zdroje nebo kondenzátory. Každý uzel musí být uzemněn, což lze zajistit připojením paralelního rezistoru s velkým odporem (např. 1 TΩ) [7].

Definice modelů je u systémů SPICE důležitá a užitečná věc, protože nám umožňuje definovat parametry skupin součástek a pak je snadno používat v netlistu. **Model** je předpis pro simulátor popisující např. výrobní řadu jednoho druhu součástek se všemi společnými parametry, případně celé skupiny polovodičových součástek vyráběné jednou křemíkovou technologií. Modely si můžeme nadefinovat sami, či použít dostupné knihovny. Obecně se model definuje takto:

```
.MODEL MNAME TYPE(PNAME1=PVAL1 PNAME2=PVAL2 ... )
```

MNAME je náš název modelu, zatímco TYPE je jeden z 15 možných typů, které zahrnují mimo jiné rezistory, kondenzátory, diody, či mnoho typů tranzistorů. Typická definice modelu NPN tranzistoru může vypadat následovně:

```
.MODEL MOD1 NPN (BF=50 IS=1E-13 VBF=50)
```

NPN je označení typu a hodnoty v závorce definují různé parametry zesílení tranzistoru.

Příklad použití v netlistu u rezistoru:

```
R2 3 4 Rmod 2k
```

Tento řádek z netlistu reprezentuje rezistor o velikosti 2kΩ zapojený mezi uzly 3 a 4 s charakteristikami dle modelu Rmod.

Makra v Spice3 slouží k definici obvodů, které potom můžeme zapojit kdekoli v netlistu, jako by se jednalo o součástky. Definice makra vypadá tak, že začneme řádkem

```
.SUBCKT subnam N1 <N2 N3 ...>,
```

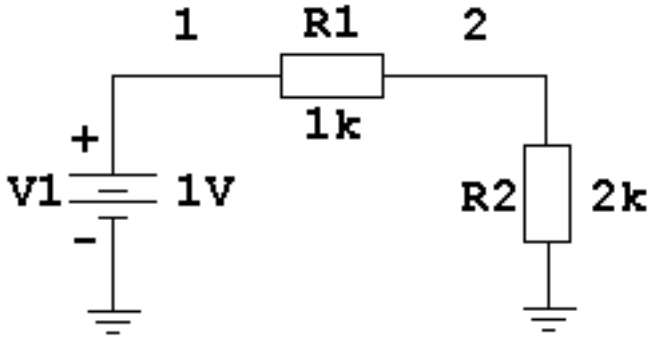
následovaným řádky obsahujícími popis obvodu a ukončenými „.ENDS“. „subnam“ je jméno našeho makra a N1 atd. jsou jména vnějších uzlů, které nesmí obsahovat „0“. Vyvolání makra se provádí jeho pojmenováním následovaným seznamem uzlů tak, jak mají být propojeny a poté jménem makra.

Komentáře jsou buď celořádkové začínající znakem „*“ a nebo na zbytek řádku, oddělené znakem „;“. Pokud se text nevejde na jeden řádek, můžeme navázat dalším znakem „+“. Interpret jazyka nerozlišuje velká a malá písmena. Názvy **prvků** vždy začínají písmenem, názvy **příkazů** (definice maker a modelů, direktivy určující analýzy a způsob výstupu atd.) vždy tečkou.

Na závěr popisu syntaxe je přiložena ukázka kompletního souboru včetně obrázku schématu výsledného obvodu [7].

```
Odporový dělič                          název obvodu
* netlist                               komentář
R1 1 2 1k ; rezistor mezi uzly 1 a 2    komentář jinak
```

```
R2 2 0 2k
V1 1 0 DC 1                    ss zdroj 1V
                               prázdný řádek
* definice analýz              další komentář
.OP                            výpočet pracovního bodu
.END                           soubor musí končit příkazem .END
```



Obrázek 2.1: Jednoduchý obvod [7]

2.2.2 Typy analýzy v ngspice

Systém ngspice umožňuje provádět 9 základních typů analýzy: Citlivostní (zkratka ve vstupním souboru .SENS), stejnosměrnou (.DC, .OP, .TF), střídavou (.AC), šumovou (.NOISE), přechodovou (v časové oblasti, .TRAN), spektrální, teplotní, nelineární stejnosměrnou a statistickou a toleranční [7]. Pro názornost jsou v následujících podkapitolách více popsány některé z nich.

2.2.2.1 Stejnosměrná analýza

Stejnosměrná (ss) analýza nejprve určí stejnosměrný operační bod zkratováním všech cívek a rozpojením všech kondenzátorů. Parametry se určují v příslušných řádcích (.DC, .TF, .OP), více o nich v [25]. Výsledek stejnosměrné analýzy je použit pro přechodovou a střídavou analýzu malých signálů. Pokud je to vyžadováno, jsou k ss analýze vypočítány přechodové funkce (vstup ku výstupu), vstupní a výstupní odpor.

2.2.2.2 Střídavá analýza malých signálů

Tato analýza počítá výstupní proměnné jako funkce kmitočtu. Nejprve je spočítán ss operační bod a poté jsou vytvořeny lineární modely pro všechny nelineární prvky v obvodu. Výsledný lineární obvod je poté analyzován na kmitočtech určených uživatelem. Výsledkem je obvykle nějaká přechodová funkce (napěťové zesílení, přechodový odpor atd.).

2.2.2.3 Přechodová analýza v časové oblasti

Přechodová analýza zobrazuje výsledné proměnné jako funkci času ve specifikovaném časovém intervalu. Počáteční podmínky jsou získány z ss analýzy. Všechny časově nezávislé zdroje jsou nastaveny na svou ss hodnotu. Časový interval je nastaven na řádku `.TRAN`.

Obecný zápis příkazu:

```
.TRAN TSTEP TSTOP [ [ TSTART ] TMAX ]
```

TSTEP znamená krok a TSTOP znamená konec intervalu. Pokud je vynechán TSTART, předpokládá se, že začátek je nula. TMAX je spíše berlička, která má zaručit, že je vybrán správný krok výpočtu, protože SPICE si vybírá minimum z hodnot TSTEP a (TSTOP-TSTART)/50.

2.2.2.4 Další typy analýzy a formát výstupů

Pro detailní popis těchto a dalších typů analýzy doporučuji nahlédnout do manuálu [25]. Na stejném místě je možné najít množství dalších příkazů, doporučení a příkladů.

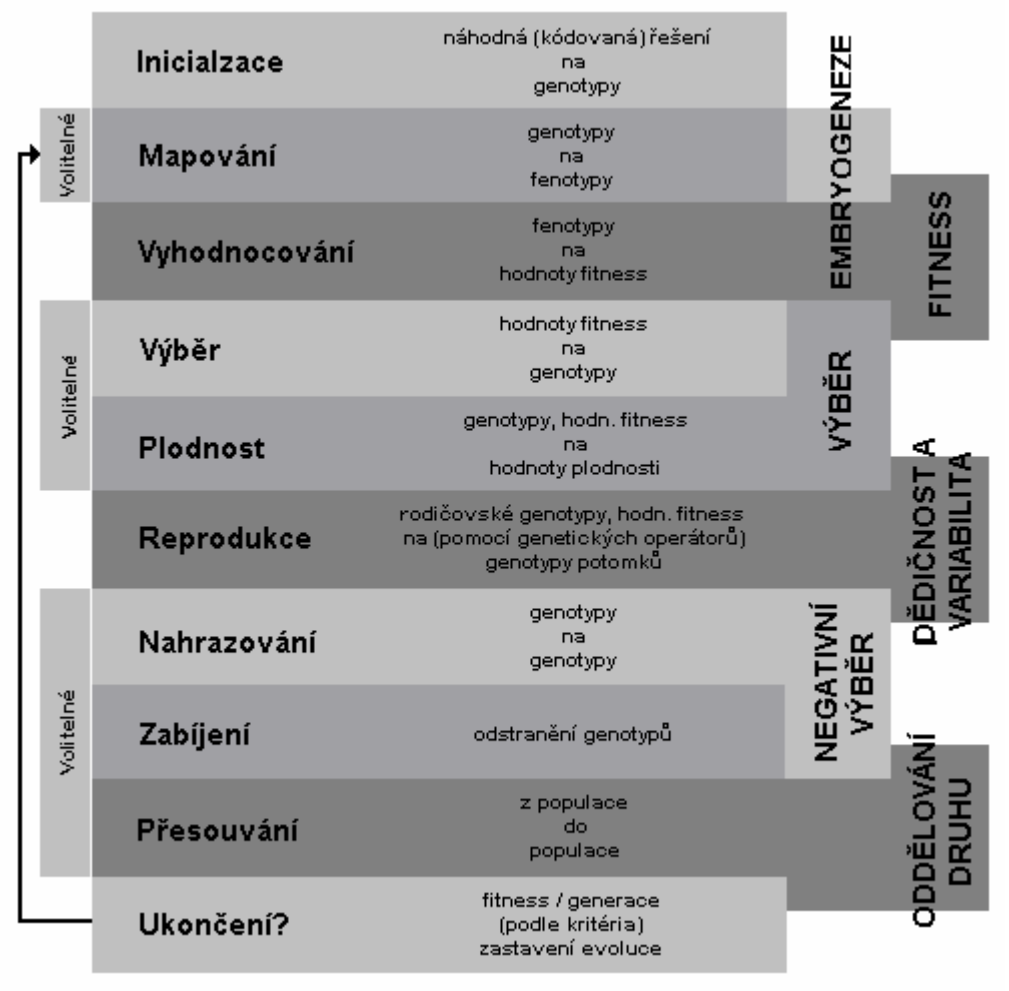
Výstupy analýz ze zapisují přímo do výstupního souboru, je ovšem možné je zobrazit jinak speciálními příkazy .PRINT a .PLOT. Příkaz .PRINT slouží pro tabulkové zobrazení jedné až osmi proměnných, ale jen pro analýzy .DC, .AC, .TRANS a .NOISE. Na stejné analýzy platí i příkaz .PLOT, který zobrazí graf složený z ASCII znaků. V příkazu je možné nastavit parametry grafu.

3 Evoluční návrh

3.1 Evoluční algoritmy

Evoluční algoritmy (EA) jsou prostředky, které se používají v oblasti známé jako evoluční návrh. V literatuře obvykle najdeme klasifikaci na čtyři základní typy těchto algoritmů: Genetické algoritmy (GA), evoluční strategie (ES), evoluční programování (EP) a genetické programování (GP). Všechny tyto algoritmy jsou inspirovány Darwinovou teorií evoluce [4] a jejími následovníky. Práce s nimi spočívá v tom, že z populace kandidátních řešení vyhledáváme ta nejlepší z nich a s jejich pomocí vytváříme další generaci řešení. Cílem je po určitém počtu generací dospět k řešení, které vyhovuje specifikaci. Jinak řečeno, pomocí biologií inspirovaných postupů se snažíme najít dostatečně kvalitní řešení v prostoru možných řešení.

Prostor prohledávání obsahuje kódovaná řešení zvané *genotypy* (či jinak *chromozomy*). Řešení vytvořená podle genotypů jsou nazývána *fenotypy*. Hodnota genu v chromozomu se nazývá *alela*. Vhodnost kandidátního řešení označujeme termínem *fitness hodnota*. Následující obrázek podrobněji ilustruje průběh EA. Obrázek platí pro obecný evoluční algoritmus, ale některé fáze jsou volitelné a jejich průběh závisí na typu algoritmu.



Obrázek 3.1: Obecná struktura evolučního algoritmu

Obrázek naznačuje, že existuje více variant, než jsou ony základní čtyři, ale v tomto případě se nadále budeme věnovat jen genetickému programování a evolučním strategiím. Více informací o EA lze nalézt v přiloženém seznamu literatury [2, 9, 10, 11, 14, 22, 32].

3.1.1 Klamné problémy

Klamné (deceptivní) problémy jsou zvláštní skupinou problémů, se kterými se EA obvykle jen obtížně potýkají. Kamenem úrazu je zde skutečnost, že u klamných problémů se vyskytují řešení, která zavádějí algoritmus do slepé uličky lokálního optima tím, že mají vyšší fitness hodnotu než řešení, která umožňují vývoj správným směrem. Po několika proběhnutých generacích se může stát, že tato špatná řešení kvůli své vyšší fitness zaplaví populaci a poté je jen velmi obtížné pomocí mutace nebo křížení opustit oblast takového lokálního optima a evoluce se prakticky zastaví.

Klamné problémy můžeme ilustrovat na tzv. minimálním klamném problému (MDP). Předpokládejme, že máme množinu čtyř schémat (teorie schémat v EA viz [6]) 2. řádu ve dvou pevných pozicích. Postačí nám informace, že schéma je stavební blok v rámci chromozomu s pevně danými hodnotami některých genů. Množina vypadá následovně:

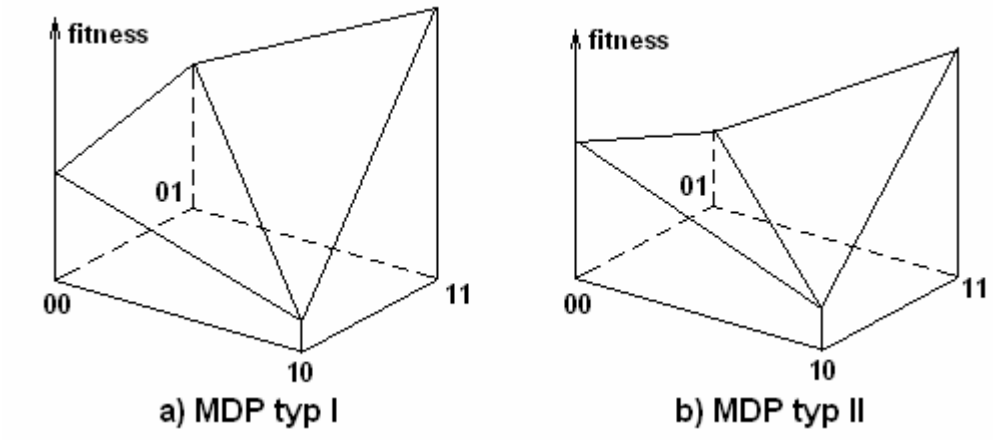
0**0*	f_{00}
0**1*	f_{01}
1**0*	f_{10}
1**1*	f_{11}

Funkce vhodnosti jsou označeny f a předpokládáme, že f11 je globální optimum a tedy platí $f_{11} > f_{00}$, $f_{11} > f_{10}$ a $f_{11} > f_{01}$. Rovněž předpokládejme, že jedno ze suboptimálních schémat 1. řádu má vyšší vhodnost než optimální schéma prvního řádu, např. f(0*) > f(1*). Podmínky z tohoto odstavce definují klamný problém 2. řádu. Pokud tyto podmínky rozebereme (podrobněji opět v [6]), vyjdou nám nakonec následující podmínky: $f_{10} < f_{00}$ a $f_{10} < f_{01}$. Zbývá jen rozlišit MDF na dva možné typy:

Typ I: $f_{01} > f_{00}$

Typ II: $f_{00} >= f_{01}$

Tyto typy mohou být znázorněny následujícím obrázkem 3.2 [6]:



Obrázek 3.2: Dva typy minimálního klamného problému [6]

Po důkladné analýze, ale i z obrázku, je patrné, že první typ MDP nebude pro EA překážkou. Bohužel, typ II MDP může představovat závažný problém, protože jednoduchou mutací či křížením z pozice 00 je velmi obtížné dostat se k pozici 11.

3.2 Genetické programování

Genetické programování vzniklo na začátku devadesátých let na Stanfordské univerzitě okolo profesora Johna Kozy, který je i nadále hlavním průkopníkem v této oblasti. Hlavním cílem GP bylo navrhovat pomocí evoluce počítačové programy. Tohoto cíle prozatím nebylo na složitější úrovni (ve smyslu rozsáhlosti programů) dosaženo, ale jak bude zmíněno dále, GP se dá využít pro návrh jiných objektů, které ne tak zcela souvisejí s počítači, ale přesto se dají reprezentovat programem.
Byla navržena velice jednoduchá metoda reprezentace programů v jazyce LISP, která využívá stromové struktury [9]. Není ovšem nutností vyvíjet programy právě v LISPu. Základní GA, který je velice podobný GP, využívá binární kódování pro reprezentaci kandidátních řešení. GP na rozdíl od toho kóduje řešení do stromů s dvěma hlavními typy uzlů [10]. **Funkční uzly** obsahují obvykle textové řetězce reprezentující funkce v programu, který se snažíme navrhnout. Běžně jde o aritmetické a logické operace nebo matematické funkce typu sinus, kosinus atd. **Listové uzly** zase obsahují proměnné vztahující se k problému či číselné nebo jiné vhodné konstanty. Alternativní názvosloví pro tyto uzly je **množina funkcí** (funkční uzly) a **množina terminálů** (listové uzly). Definice těchto množin patří k základním krokům, které musíme vykonat před vlastním započetím algoritmu.

3.2.1 Přípravné kroky GP

Ještě než popíši algoritmus, dám prostor přípravné fázi. Těchto tzv. přípravných kroků GP je v zásadě pět [10]:

Určení množiny terminálů – Tento krok je vlastně jakýmsi určením rozsahu proměnných a konstant, které budeme v našem výsledném programu očekávat. Navrhujeme-li například analogový obvod, bude množina terminálů obsahovat součástky jako tranzistory, rezistory, kondenzátory a vodiče. V případě návrhu matematické funkce zase půjde o proměnné jako x, a, b či konstanty např. v rozsahu od 0 do nekonečna.

Určení množiny funkcí – Množina funkcí opět úzce závisí na tom, jaký problém chceme pomocí GP řešit. Jde v podstatě o určení klíčových slov v našem programu. Typická množina funkcí může obsahovat hodnoty jako +, -, *, /, %, AND, OR, NOT, XOR, IF..THEN..ELSE, WHILE…DO, apod. Například, programujeme-li robota pohybujícího se mezi překážkami, mohou funkce vypadat takto: Zaboč vlevo, zaboč vpravo, dopředu, dozadu, otáčej se atd.

Určení fitness funkce – Stejně jako u ostatních EA, fitness je skalární hodnota reprezentující kvalitu kandidátního řešení. Vzhledem k povaze úkolů, které jsou předkládány GP, může být určení způsobu hodnocení fitness poněkud problematické. Například u návrhu obvodů nás zajímá, jak je výsledné řešení blízké specifikaci. Jedním z možných řešení může být vzorkování hodnot napětí po určenou dobu a spočítání vzdáleností hodnot napětí od požadovaných hodnot. V případě hledání matematické funkce můžeme postupovat podobně, ale navrhujeme-li anténu, je třeba přijít s nějakým jiným hodnotícím kritériem. Toto jsou však pouze ilustrační příklady, o využití GP se zmíním dále. Stručně řečeno, tato fáze je jednou z nejdůležitějších a nejobtížnějších na celém procesu GP, stejně tak jako u ostatních EA.

Nastavení řídicích parametrů algoritmu – Tento důležitý krok slouží k určení základních parametrů algoritmu, jako je například velikost populace, počet generací, pravděpodobnost použití genetických operátorů, maximální velikost programu atd. V GP se obvykle nastavuje velikost populace v řádech stovek a tisíců (až statisíců jedinců - dle složitosti problému) a počet generací je spíše menší, maximálně několik stovek [10].

Ukončovací kritérium a určení výsledku GP – Tento krok souvisí s předchozím, pokud jde o dosažení daného počtu generací, protože to bývá jedno z nejčastějších ukončovacích kritérií. Další možností může být dosažení určité hodnoty fitness. Výsledné řešení běhu GP bývá zpravidla nejlepší dosažené řešení ze všech generací, protože se snadno může stát, že poslední generace obsahuje řešení horší, než některé předchozí.

3.2.2 Algoritmus GP

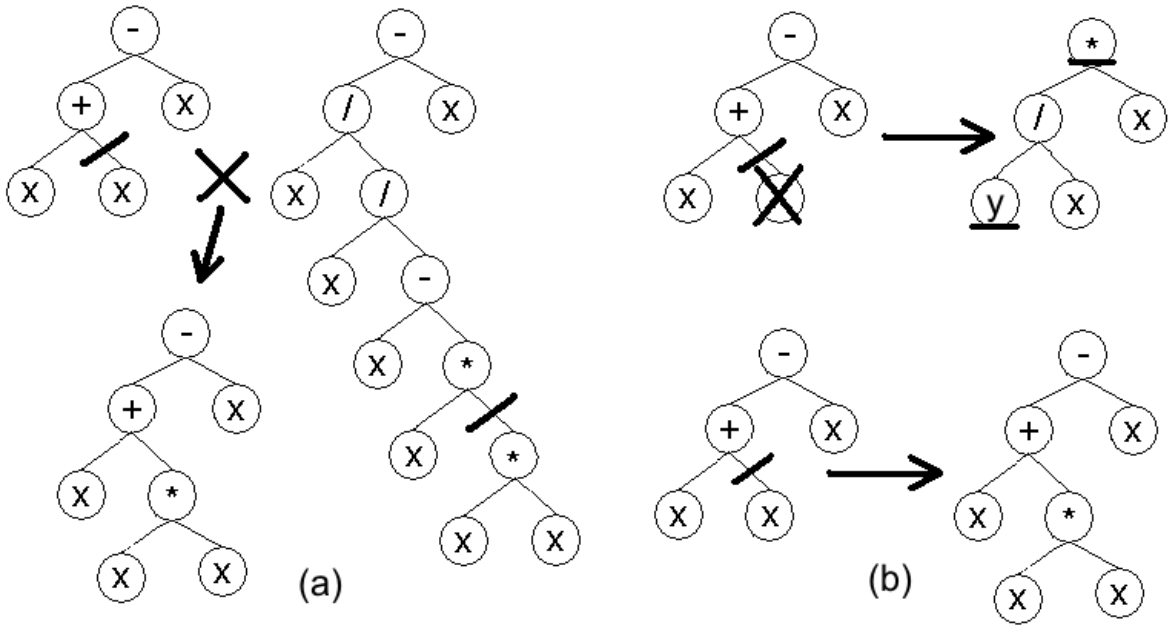
Po tom, co jsme provedli přípravné kroky, můžeme přistoupit k vlastnímu algoritmu, který se sestává z předem definovaných, na problému nezávislých kroků [9].

1. Náhodně vytvoř počáteční populaci kandidátních programů s využitím množin funkcí a terminálů
2. Opakovaně prováděj následují kroky (souhrnně nazývané generace), dokud není uspokojeno ukončovací kritérium:
    a) Proveď každý program v populaci a urči jeho fitness
    b) Vyber jeden nebo dva programy z populace s pravděpodobností závislou na jejich fitness pro další krok
    c) Vytvoř nové individuální programy aplikací následujících operací s definovanou pravděpodobností:
        i) *Reprodukce* – kopíruj vybraný program zpět do populace
        ii) *Křížení* – vytvoř dva nové programy rekombinací náhodně vybraných částí dvou zvolených programů
        iii) *Mutace* – vytvoř nový program zmutováním náhodně vybrané části jednoho zvoleného programu
        iv) *Operace úpravy architektury* – vyber jednu operaci úpravy architektury ze seznamu těchto operací a vytvoř její aplikací jeden nový program
3. Po tom, co je uspokojeno ukončovací kritérium, je vybráno nejlepší řešení z tohoto běhu GP a určeno jako výsledek. V případě úspěchu může jít o řešení (či přibližné řešení) daného problému.

Je třeba zmínit, že operace křížení je velice důležitou součástí GP, která vlastně přináší do populace nový genetický materiál a umožňuje se vyhnout příliš rychlé konvergenci do lokálního optima. U GP je nejčastěji chromozom n-árním stromem. Zatímco u GA při křížení náhodně vybereme bod, ve kterém prohodíme zbývající části chromozomu rodičů, u GP musíme zvolit takové body dva, což může být kdekoli mezi dvěma uzly. Poté vyměníme podstromy nacházející se pod bodem křížení. Pokud tedy dojde ke křížení dvou identických rodičů u GA, potomci budou také identičtí. Pokud ale zkřížíme identické rodiče v GP, je malá pravděpodobnost (menší, čím složitější jsou stromy), že

dosáhneme stejného výsledku, tudíž v populaci vznikne větší diverzita [31]. Obrázek 3.3 (a) demonstruje operaci křížení.

Operace mutace může mít různé podoby, ale nejčastější je výměna uzlu za nový náhodný podstrom, odstranění podstromu nebo zrušení či změna jednoho uzlu. Obrázek 3.3 (b) demonstruje některé možnosti mutace.



Obrázek 3.3: (a) Operace křížení a (b) některé metody mutace v GP – mazání podstromu (vlevo nahoře), náhodná změna uzlů (nahoře) a generování náhodného podstromu namísto uzlu (dole)

3.2.3 Rozšíření možností GP

3.2.3.1 Automaticky definované funkce (ADF)

Protože GP se zabývá vytvářením počítačových programů, ukazuje se, že myšlenka použití běžných programovacích postupů tak, jak je dělají lidé, může být výhodná z pohledu zkvalitnění výsledků evoluce [10]. Součástí takových postupů je používání procedur a funkcí, obecně podprogramů, které jsou v programu vyvolávány na provádění opakovaných výpočtů. Proto byl do GP zaveden pojem ADF. Místo toho, aby se ztratilo velké množství strojového času při vykonávání algoritmu GP tím, že se neustále vyvíjejí ve velkém množství stejné či podobné části programu, použijí se místo toho ADF.

ADF fungují tak, že se vyvíjený program rozdělí na více větví, z nichž několik jsou „hlavní programy“ produkující výsledek a několik z nich jsou ADF, které umožňují znovuvyužívání a hierarchickou strukturalizaci kódu. ADF obsahují vlastní proměnné a patří jen k jednomu kandidátnímu programu. V hlavní větvi je potom možné tyto funkce vyvolávat a je povolena i rekurze.

Jako dodatek k ADF se používají automaticky definované iterace (ADI), automaticky definované smyčky (ADL) a automaticky definované rekurze (ADR). Představují další prostředky v zvovuvyužití kódu. Dalším podobným prvkem jsou automaticky definované proměnné (ADS – Stores), ve kterých se mohou uchovat výsledky předchozích operací pro další použití.

3.2.3.2 Syntakticky omezené struktury

U složitějších problémů se mnohdy snažíme zabránit zbytečným operacím, které mohou nastávat při běhu GP, a ušetřit jejich vyřazením drahocenný strojový čas. Syntakticky omezené struktury jsou často využívaným prostředkem k dosažení tohoto cíle. Jde v podstatě o gramatiky, které zaručují, že funkcím nebudou předávány nesmyslné parametry. Takové opatření si samozřejmě žádá režii, protože je nutné zaručit syntaktickou správnost při generování počáteční populace i při aplikaci genetických operátorů. Dle výsledků v [10], které syntaktické omezení všechny používají, se to ale vyplatí.

3.2.3.3 Architektura programu a operace na její úpravy

S přihlédnutím ke kapitole 3.2.3.1 můžeme při návrhu s GP přidat jeden přípravný krok navíc a to určení architektury programu. Programátor tedy explicitně stanoví počet větví kandidátních programů a ovládá tak konečnou strukturu výsledku. Dále je možné takovou architekturu evolučně vyvíjet za běhu programu a nebo ji ovlivnit přidáním operací na úpravu architektury. U posledních dvou možností zůstává počet přípravných kroků na pěti.

3.2.3.4 Developmnent v GP

Development v GP je metoda, která je vhodná u složitějších problémů, jako je např. návrh elektronických obvodů. Její princip spočívá v použití tzv. embrya a jeho vývoje na základě programu. Vzhledem k celkovému záměru této práce je tento postup podrobněji vysvětlen v kapitole 6.

3.2.4 Využití GP

Jak jsem již naznačil na předchozích stránkách, GP se využívá v různých oblastech, avšak vytváření skutečných počítačových programů v jazycích typu C, C++ či Java není tou hlavní doménou, zejména kvůli složitosti takových programů. Výjimkou je v tomto směru jazyk LISP, pro který byla metoda GP vyvinuta a pro který se hodí. Nicméně LISP není příliš populární a jeho využití není široké.

Jednou z hlavních oblastí pro GP je návrh elektronických obvodů, právě kvůli využití developmentálního přístupu, což dokazují výsledky publikované v [10]. Špičkou v tomto oboru je právě prof. Koza, který označuje GP jako rutinní nástroj umělé inteligence, který je za určitých okolností schopen srovnatelných či někdy ještě lepších výsledků než lidský inženýr. Některé výsledky těchto experimentů jsou patentovatelné. Více o nich lze nalézt v [10].

Zajímavou aplikací GP může být návrh klasifikátorů a prediktorů. Jde o elektronické obvody, jejichž možný počet vstupních kombinací může být příliš velký na to, aby je bylo možné navrhovat konvenčními metodami. Typickou aplikací jsou obrazové filtry. Tímto tématem se zabývá práce v [21].

Dalšími oblastmi mohou být regulační technika, chemie a genetika, umisťování a propojování elektronických obvodů na deskách plošných spojů či návrh antén (využito NASA) [22]. Možnosti GP jsou velké a porostou spolu s výkonem výpočetních systémů.

3.2.5 Možná implementace GP

Obecně je možné provést implementaci GP v mnoha různých jazycích, např. [31] uvádí C, C++, Javu a LISP. Jednou z možností v C++ může být GAlib. Jde o soubor knihoven pro C++ pracující na mnoha platformách. GAlib byl vyvinut na Massachussets Institute of Technology (MIT) a používá se již mnoho let. S GAlibem jsem již pracoval, takže s ním mám zkušenosti a i když je knihovna určena

zejména pro GA, vzhledem k příbuznosti obou metod a k tomu, že GAlib obsahuje struktury pro stromy, jeví se mi jako vhodný prostředek. Podrobnější informace o GAlibu obsahuje literatura [26, 32, 33].

3.3 Evoluční strategie

Evoluční strategie (ES) patří k hlavním směrům v oblasti evolučních algoritmů. Narozdíl od GP, které je poměrně novým směrem v EA, vznikly ES už počátkem 60. let v Německu jako výsledek práce Rechenberga a Schwefela [11]. Hlavním cílem ES nebyl návrh, ale optimalizace. Autoři ES se snažili upravit tvar křídel tak, aby kladla co nejmenší odpor vzduchu. Při tom se ukázalo, že konvenční inženýrský přístup není úspěšný, zatímco náhodné změny parametrů definujících tvar křídel úspěšné být mohou. K účelu optimalizace se tyto algoritmy ve víceméně nezměněné podobě používají dodnes. Další využití zahrnuje mimo jiné kartézské GP, o kterém se zmiňuji podrobněji v průběhu dalších kapitol.

3.3.1 Základní koncepce ES

Nejčastější verze ES pracují s reálnými čísly v chromozomu, narozdíl od GA, jejichž chromozomy obvykle obsahují binární či celá čísla. Základním kamenem ES je následující rovnice definující předpis, kterým získáme nové řešení x' z předchozího x:

$$x' = x + N(0,\sigma), \qquad (3.1)$$

Kde $N(0,\sigma)$ je vektor nezávislých náhodných čísel s normální distribucí, nulovou střední hodnotou a standardní odchylkou σ [11].

Nové řešení se akceptuje jedině tehdy, platí li, že fitness hodnota nového řešení je vyšší nebo stejná jako fitness hodnota předchůdce (za předpokladu, že vyšší fitness je lepší). Dalo by se říci, že jde o selektivní mutaci. V jednodušších ES se genetický operátor křížení nepoužívá. Algoritmus bude popsán dále v příslušné podkapitole o variantách ES níže, která je relevantní k tématu této práce.

3.3.1.1 Pravidlo 1/5

Pravidlo 1/5 je předpis pro změnu standardní odchylky σ podle úspěšnosti mutací. Jeho smyslem je vylepšení výsledků ES, tedy zrychlení konvergence k optimu.

Pravidlo funguje následovně: Předpokládejme, že $\phi(k)$ je poměr počtu úspěšných mutací v posledních k iteracích k počtu iterací. Potom nová hodnota σ', kterou počítáme po každé další iteraci se získá následovně:

$$\sigma' = \begin{cases} c_d \cdot \sigma & (\varphi(k) < 1/5) \\ c_i \cdot \sigma & (\varphi(k) > 1/5), \\ \sigma & (\varphi(k) = 1/5) \end{cases} \qquad (3.2)$$

kde $c_i>1$ a $c_d<1$, přičemž obvyklé hodnoty bývají $c_d=0{,}82$ a $c_i=1/c_d=1{,}22$ [11].

ES, které využívají pravidlo 1/5 nebo některé jiné podobné, se označují přívlastkem adaptivní. Tato modifikace se nepoužívá vždy, neboť v některých zvláštních případech nemusí vyhovovat.

Algoritmus může být modifikován i jiným způsobem, jako např. gaussovským rozložením namísto normálního, ale podrobnější vysvětlení dalších možných variant od standardního algoritmu by přesahovalo smysl tohoto textu.

3.3.2 Hlavní typy ES

To, co rozlišuje jednotlivé druhy mezi sebou, je způsob, jakým v průběhu algoritmu dochází k vytváření nové populace. Následuje výčet nejdůležitějších zástupců ES.

3.3.2.1 Strategie (1 + 1)

Základním typem ES je (1 + 1), což znamená, že každá generace obsahuje jen jedno kandidátní řešení a mutací (viz rovnice 3.1) se vytváří jeden potomek, který se v případě lepší fitness hodnoty stává kandidátním řešením pro následující generaci.

3.3.2.2 Strategie (1 + λ)

Tato strategie je pro téma této práce nejdůležitější, protože se používá jako součást evoluční návrhové metody kartézského GP, které bude popsáno v dalších kapitolách.

Název strategie znamená, že každá generace obsahuje pouze jednoho jedince, ze kterého se mutací vytváří λ potomků, přičemž se z výsledné množiny kandidátních řešení čítající λ + 1 jedinců vybírá nejlepší jedinec do příští generace. Algoritmus v pseudokódu vypadá následovně:

```
Náhodně vygeneruj rodiče;
Generace := 0;
(alternativně - Vezmi jednice, který má být optimalizován;)
while (není splněno ukončovací kritérium) begin
      Vygeneruj λ potomků z aktuální populace, každého pomocí
      mutace N náhodných genů;
      For i=1 to λ do
            if (fitness(rodič) < fitness(potomek[i])) then
                  rodič = potomek[i];
      inc(generace);
end;
Print(„Výsledek je“ + rodič + „ v “ + generace + „. generaci“);
```

3.3.2.3 Strategie (μ + λ)

Tato strategie je obecnějším vyjádřením obou předchozích případů. Plus v názvu znamená, že do následující generace se vybírají jedinci z množiny, která je sjednocením jak rodičů, tak potomků. Písmeno μ znamená, že každá generace obsahuje právě μ kandidátních řešení. λ znamená, že mutací je z rodičů vygenerováno λ potomků.

3.3.2.4 Strategie (μ, λ)

Poslední ze strategií, které zde jsou popsány je strategie s čárkou. Písmena μ a λ znamenají totéž, co v předchozím případě, ale čárka má takový význam, že do následující generace jsou vybráni jedinci jen z vygenerovaných potomků. Rodiče jsou zahozeni. λ musí být v tomto případě větší než μ, aby algoritmus měl smysl, tj. aby se vylepšovala populace.

Tato a předchozí generace využívají křížení. To funguje např. tak, že u dvou vybraných jedinců dojde ke zprůměrování odpovídajících genů (tedy obvykle prvků vektoru reálných čísel).

4 Návrh hardwaru s využitím EA

Pro lepší pochopení problematiky návrhu elektronických obvodů a hardwaru obecně je zařazena tato kapitola, která se zabývá rozdělením úrovní návrhu hardware a dále shrnutím některých důležitých návrhových prostředků v kontextu evolučních algoritmů.
Pro označení elektroniky vyvíjené pomocí EA se běžně používá termín **Evolvable Hardware** (zkratka **EWH**). EWH spočívá zejména ve spojení dvou hlavních principů: EA a rekonfigurovatelných zařízení, případně jejich simulátorů.

4.1 Úrovně návrhu hardwaru

4.1.1 Molekulární úroveň

Návrh elektronických obvodů na molekulární úrovni je oblast, která se teprve rozvíjí. Hlavní myšlenkou je používat jednotlivé molekuly a jejich řetězce jako součástky elektronického obvodu, tedy tranzistory, spoje a další. Molekulární úroveň se abstrakcí blíží tranzistorové úrovni, nicméně je třeba vzít v úvahu mnoho problémů, které sahají až do oblasti kvantové fyziky.

Molekulární úrovní se v dnešní době zabývají zejména univerzity a to hlavně v USA, v Evropě je tato oblast výzkumu v začátcích [3]. Hlavním problémem je, jak již bylo zmíněno, odlišné chování materiálů a aplikace jiných fyzikálních zákonů u výrobních technologií menších než 0.1 mikronu. Dalším problémem je nedostatek spolehlivých teoretických podkladů, se kterými by vědci mohli pracovat. To by teoreticky nemuselo vadit v případě evolučního návrhu, protože EA jsou v drtivé většině problémově nezávislé, takže vystačí s neúplnou znalostí problému. Bohužel, toto naráží na další překážku, kterou je nutnost vyvinuté obvody vyzkoušet, aby bylo možné určit hodnotu fitness. Vzhledem k tomu, že nejsou k dispozici vhodné teoretické základy, nemůžeme vlastnosti takového obvodu kvalitně simulovat. Nezbývá tedy, než obvod emulovat, nicméně to s sebou přináší řadu dalších technických problémů.

I přes všechny tyto překážky se objevují pokusy o evoluční vývoj na molekulární úrovni, i když se nedá přesně mluvit o použití spojů, tranzistorů či jiných součástek. Jeden takový úspěšný pokus využívá fyzikálních vlastností materiálu, který známe jako tekuté krystaly. Více informací je v [15].

4.1.2 Úroveň tranzistorů

Na této úrovni se dá mluvit o praktičtějším použití než v předchozím případě. Je zde možné navrhovat jak analogové obvody, tak i číslicové a smíšené obvody. Avšak číslicové a smíšené obvody uvnitř fungují právě díky tranzistorům, tedy v zásadě analogovým součástkám. Vlastnosti tranzistorů, diod, rezistorů a dalších elektronických součástek dokážeme velmi podrobně popsat a tudíž jsme schopni navrhovat obvody na základě teoretických znalostí.

Bohužel, i tady se vyskytuje problém. Navrhnout složitější obvod na úrovni tranzistorů není triviální a je rozšířeným názorem, že na návrh potřebujeme velmi schopného a zkušeného inženýra, protože v analogovém světě je velmi obtížné, či přímo nemožné řešit problémy pomocí automatizovaných návrhových nástrojů [10]. Mnoho návrhářů se proto raději uchyluje k vyšším

úrovním abstrakce, kde je možné pro návrh číslicových obvodů využít dostupné CAD (Computer Aided Design) systémy.

Právě z tohoto důvodu je tranzistorová úroveň vhodná oblast pro využití evolučních postupů. Evoluce nemusí „vědět“, jak obvod uvnitř funguje, dostačuje seznam použitelných součástek a případné omezení způsobů jejich propojení. Dále postačuje znát jeho reakci na vstupy a je možné určit, zda je obvod kvalitní. K tomuto účelu slouží dvě hlavní metody evolučního návrhu: **Extrinsická**, tedy využívající softwarových simulátorů (Kapitola 2), či **intrinsická**, využívající rekonfigurovatelných součástek, kdy máme možnost ověřit si chování obvodu ve fyzické realizaci [31]. Intrinsická metoda je vhodná tam, kde chceme dosáhnout velké spolehlivosti, komplexnosti a přesnosti navrhovaných obvodů, zatímco extrinsická metoda nám může pomoci ve vývoji robustních, na okolním prostředí nezávislých obvodů, protože experimenty přímo na hardwaru mohou být závislé na testovacích podmínkách v laboratoři.

4.1.3 Úroveň logických hradel

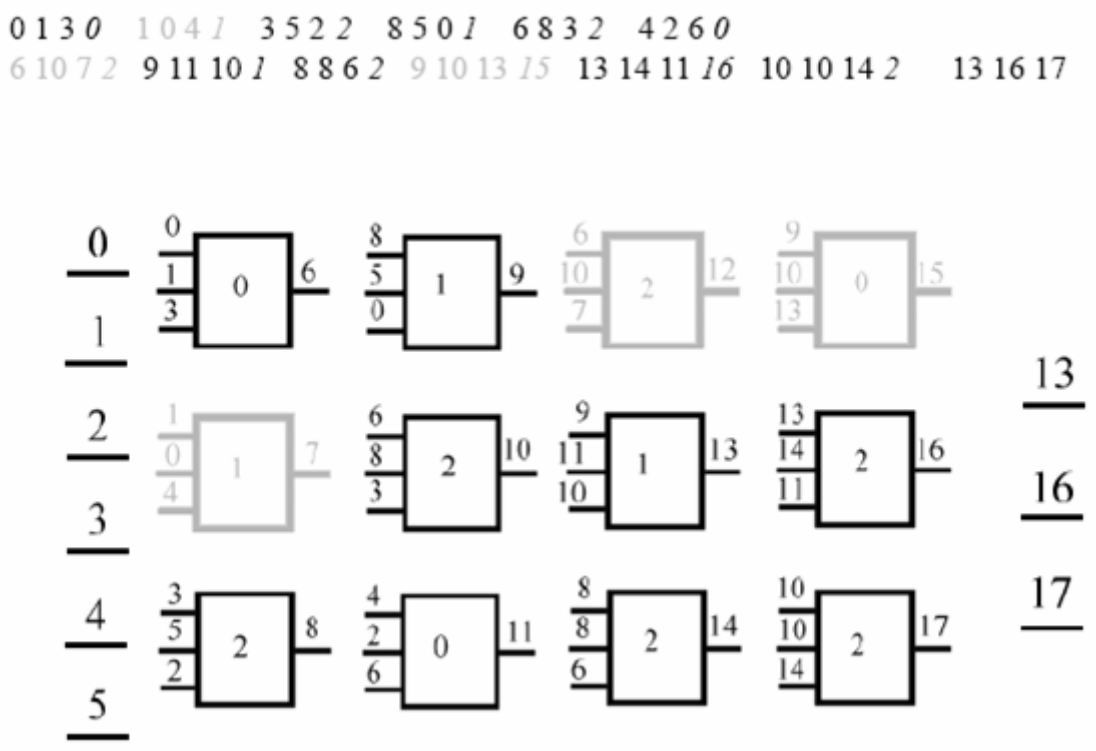
Touto úrovní se návrhář oprošťuje od analogového chování navrhovaného obvodu a může se soustředit jen na číslicovou logiku. Základními prvky takového číslicového obvodu jsou logická hradla (anglicky gate). Jde o komponenty s jedním až n 1-bitovými vstupy a jedním 1-bitovým výstupem v závislosti na funkci hradla. Hradla se označují podle své logické funkce, nejčastěji AND, OR, NOT, XOR atd. Tyto názvy odpovídají anglickým názvům operací v logické Boolově algebře (logický součin, logický součet, logická negace, nonekvivalence atd.), která se používá v číslicových obvodech. Tabulka 4.1 [1] shrnuje pravdivostní hodnoty těchto typických operací.

Tabulka 4.1: Pravdivostní hodnoty základních logických operací

A	B	AND	OR	NOT A	XOR
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

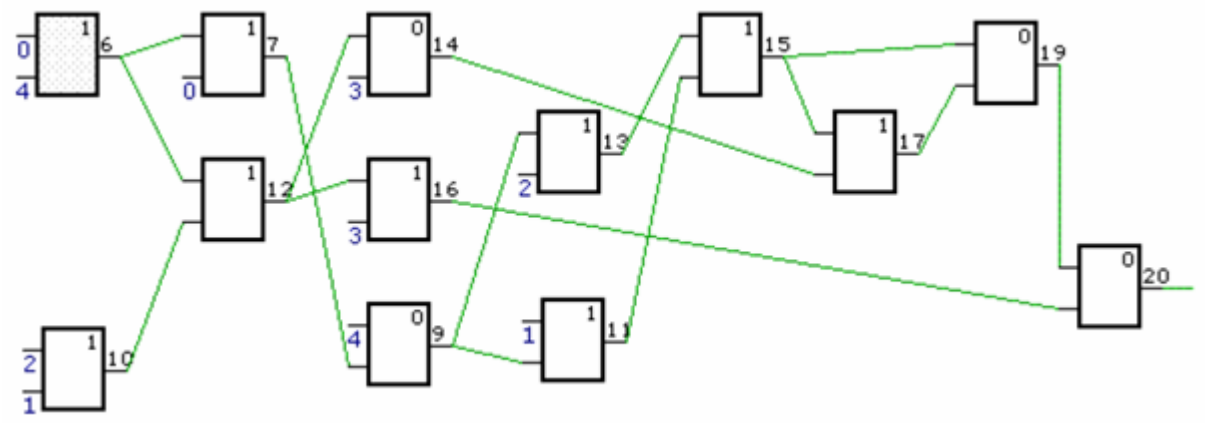
Návrh číslicových obvodů složených z logických hradel není triviální, ale existuje množství zjednodušujících postupů, které umožňují automatizaci takového návrhu např. pomocí rozličných CAD nástrojů.

Jednou z rozšířených metod v evolučním návrhu je kartézské GP (CGP) [14]. CGP je podobné GA, kde máme dvourozměrné pole či spíše graf logických hradel, která mají programovatelnou funkci a propojení vstupů. Pouze výstupy hradel a vstupní a výstupní vodiče celého obvodu jsou očíslovány. Máme-li např. pole 3x4 třívstupových hradel s šesti vstupy a třemi výstupy, může možné zakódování kandidátského řešení nějakého vypadat jako to na obrázku 4.1 [14]. Povšimněte si, že první tři číslice v každé čtveřici označují vodiče přivedené na vstupy hradla a čtvrtá číslice označuje typ hradla. Šedá hradla na obrázku nemají na funkci obvodu vliv.



Obrázek 4.1: Zakódování řešení v CGP [14]

Speciálním druhem logických hradel mohou být např. polymorfní hradla. Jde o logické členy, které mění svou funkci v závislosti na vnějších podmínkách, ať jde o teplotu okolí, hodnotu napájecího napětí či nějakou jinou přepínací veličinu. Na FITu v současné době probíhá výzkum takových hradel a obvodů z nich složených [19]. Obrázek 4.2 [19] zobrazuje ukázku polymorfního obvodu s funkcí medián/parita vyvinutého metodou CGP. Metoda CGP bude podrobněji popsána v následující kapitole.



Obrázek 4.2: Polymorfní obvod medián/parita (vstupy: 0-4, hradla: 0 – NAND/XOR, 1 XOR/NOR) [19]

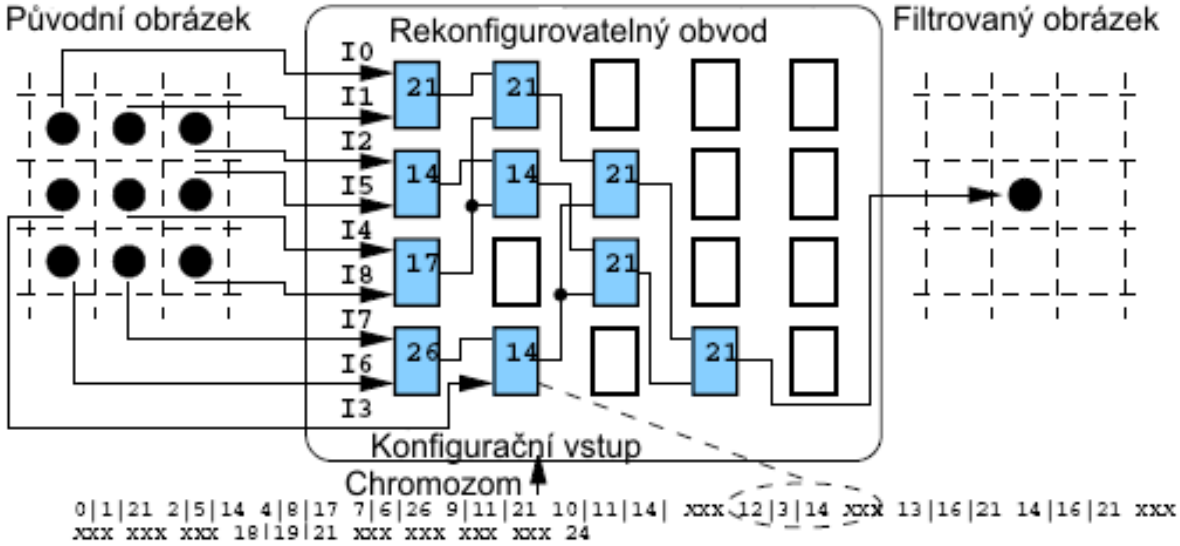
4.1.4 Úroveň funkčních bloků

Začneme-li z logických hradel vytvářet složitější celky, dospějeme do další úrovně abstrakce. Funkční bloky mohou být kombinační obvody, které reagují pouze na kombinaci vstupních proměnných, a nebo sekvenční obvody, které reagují nejen na kombinaci vstupů, ale také na aktuální vnitřní stav.

Podoba funkčních bloků může být různá, od logických funkcí s více vstupy a výstupy přes sčítačky, násobičky, multiplexory až např. po paměti.

K úrovni funkčních bloků přistupujeme v tom případě, potřebujeme-li navrhovat složitější obvody, které obsahují velké množství hradel v řádech stovek až tisíců kusů. To je něco, co by na předchozích úrovních bylo velice obtížné, ne-li přímo nemožné. Bohužel, využití EA má své limity a na této úrovni návrhu je možné řešit spíše jednodušší problémy skládající se z menší množiny typů použitých funkčních bloků. Na obrázku 4.3 [21] je ukázka návrhu obrazového filtru na úrovni funkčních bloků (binární operace a jednoduché sčítačky) za pomoci CGP, které i na této úrovni návrhu nachází uplatnění, přičemž se namísto logických funkcí používají složitější funkce, jako např. násobení, minimum a maximum a další [18]. Pokud bychom ovšem chtěli navrhovat s EA něco takového jako procesory, zjistili bychom, že to není prakticky možné kvůli velkému množství různých stavebních bloků, jež se dají propojit v obrovském množství různých kombinací. Dalším důvodem je, že evaluace složitých číslicových obvodů je velice časově náročná.

Jako poznámku na okraj (protože hlavním objektem zájmu v této práci jsou číslicové obvody) je nutno dodat, že podobné úrovně abstrakce můžeme dosáhnou v analogových obvodech, kdy nám jako stavební bloky poslouží např. zesilovače či zdroje napětí.



Obrázek 4.3: Ukázka rekonfigurovatelného obvodu a jeho konfigurace použitého u návrhu obrazového filtru za pomoci CGP. Devět obrazových bodů je použito k výpočtu hodnoty filtrovaného obrazového bodu [21]

4.1.5 Úroveň systémů

Poslední úrovní návrhu (hlavně číslicového) hardwaru, kterou hodlám zmínit, je úroveň systémů. Hranice mezi systémem a funkčním blokem je velmi tenká, ale vezmeme-li jako předpoklad jistou komplexnost, kdy za systém považujeme např. procesor či jinou, složitější výpočetní jednotku (a může to být i konečný automat), dostáváme se zejména k návrhu způsobu komunikace mezi těmito systémy. Vzhledem k tomu, že toto téma je více vzdálené smyslu této práce, nebudu ho zde dále rozvádět.

4.2 Rekonfigurovatelná zařízení v EWH

4.2.1 FPGA

Field programmable gate arrays, tedy programovatelná hradlová pole, jsou dnes jedním z nejvyužívanějších a nejdůležitějších prostředků pro intrinsický evoluční návrh. Jde o čipy, které uvnitř obsahují tisíce hradel a také desítky až stovky některých jednodušších funkčních bloků a je možné je jednoduše za chodu programovat. Kvůli stručnosti je tento popis trochu zjednodušený, více k vnitřnímu uspořádání FPGA, např. v [13, 31]. Programování znamená nastavení konfigurace propojení hradel a funkčních bloků v FPGA pomocí programovacího řetězce, sestávajícího se obvykle ze stovek bytů až několika MB, v závislosti na složitosti programovaného čipu.. To znamená, že je prakticky možné provádět EA, kandidátní řešení programovat do FPGA a vyhodnocovat fitness hodnotu hardwarového řešení přímo v místě jeho určení, to všechno velkou rychlostí. Dnešní FPGA čipy obsahují rovněž jednoduché procesory, které umožňují za chodu zařízení evolučním způsobem vylepšovat jeho funkci, případně provádět opravy v případě poruchy, protože je možné programovat jen část čipu, zatímco zbytek provádí evoluční výpočty. Popularita FPGA čipů navíc stále roste, takže to nutí výrobce k jejich dalšímu vývoji, což znamená vyšší výkon pro jakékoli aplikace na nich fungující.

4.2.2 FPAA

Tato zkratka znamená Field programmable analog arrays – tedy programovatelná analogová pole. Tato zařízení se dále dělí na obvody s hrubou a jemnou granularitou, přičemž první z nich se skládají z pole operačních zesilovačů a ty druhé z pole tranzistorů. FPAA s hrubou granularitou mají prakticky stejný význam pro analogovou oblast jako FPGA pro číslicovou. Bohužel FPAA nemají takové rozšíření jako FPGA a jejich vývojem se zabývají hlavně univerzity ve spolupráci s výrobci elektroniky (Motorola, Zetex, Palmo, Lattice) [31]. Na širší praktické uplatnění tato větev programovatelných polí stále čeká.

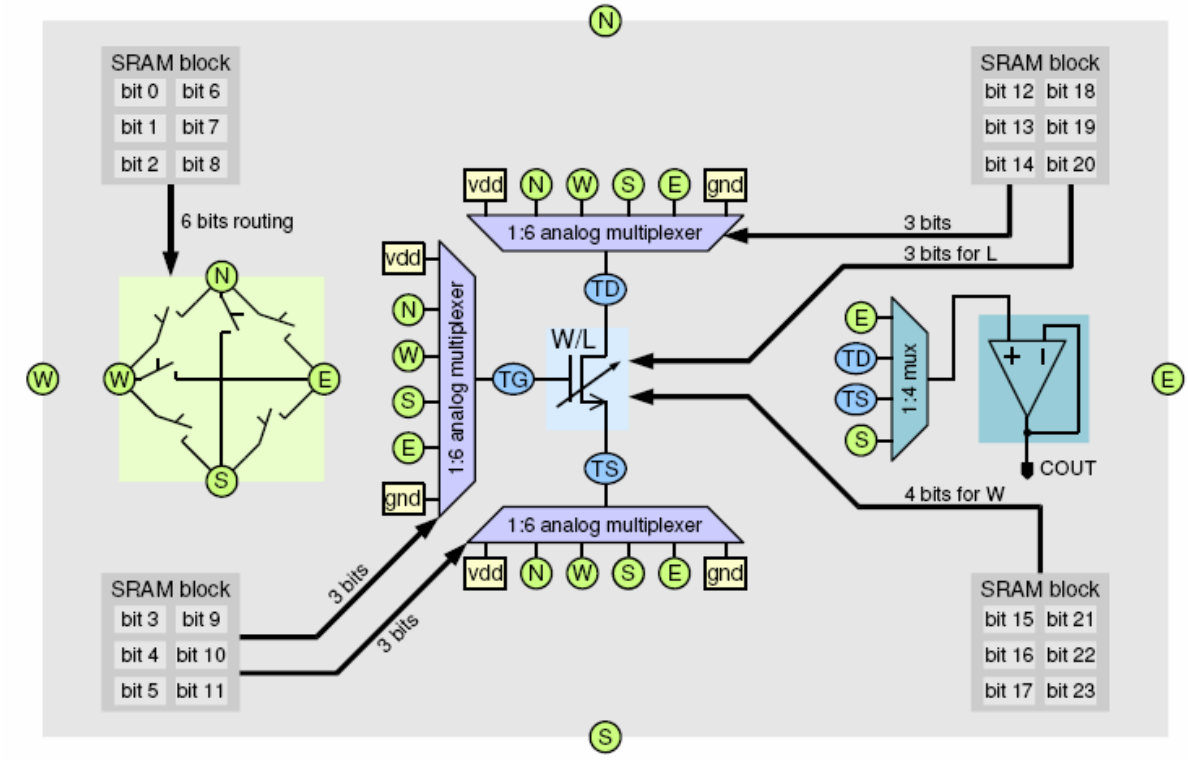
4.2.3 FPTA

Field programmable transistor arrays obvody by se daly považovat za větev FPAA, ale já je zde uvádím zvlášť, protože mají speciální pozici v oblasti evoluční elektroniky. Jejich princip je podobný jako u předchozích příkladů, nicméně skutečnost, že obsahují pole tranzistorů jim umožňuje použití jak ve vývoji číslicových obvodů na úrovni tranzistorů, tak na vývoji analogových obvodů.

Mezi důležité vývojáře těchto čipů patří Dr. Adrian Stoica z Jet Propulsion Laboratory [24], který s FPTA navrhoval jednoduché číslicové a analogové obvody [31].

Dalším zajímavým příkladem kvalitního a zejména transparentního FPTA je zařízení, které vytvořil Dr. Joerg Langeheine z univerzity v Heidelbergu [12]. Důležitou vlastností jmenovaného zařízení je to, že je velmi podrobně zdokumentováno a tudíž si potenciální uživatel může udělat přesnou představu o tom, co uvnitř probíhá. Na následujícím obrázku 4.4 je ukázka jedné buňky s tranzistorem NMOS včetně přehledové informace o zakódování konfigurace této buňky. Pole obsahuje polovinu buněk s tranzistory PMOS a druhou polovinu s tranzistory NMOS. Každá tato buňka se dá použít buďto jako cesta vedení nebo jako tranzistor s proměnnými vlastnostmi (závisí na

konfiguraci). FPTA z Heidelbergu by byl ideální pro realizaci cíle této práce, bohužel podobné vybavení v současné době není na FITu k dispozici.



Obrázek 4.4: Zjednodušená architektura buňky FPTA z Heidelbergu s NMOS tranzistorem [12]

5 Technologie CMOS

Vzhledem k tomu, že cílem této práce je navrhnout evoluční systém pro návrh obvodů na úrovni tranzistorů, obsahuje tato kapitola stručný přehled tranzistorové technologie, na kterou je návrh zaměřen. V první části kapitoly jsou přiblíženy základní charakteristiky CMOS, zatímco druhá část je zaměřena na vlastnosti technologie relevantní k návrhu obvodů.

5.1 Základní charakteristiky

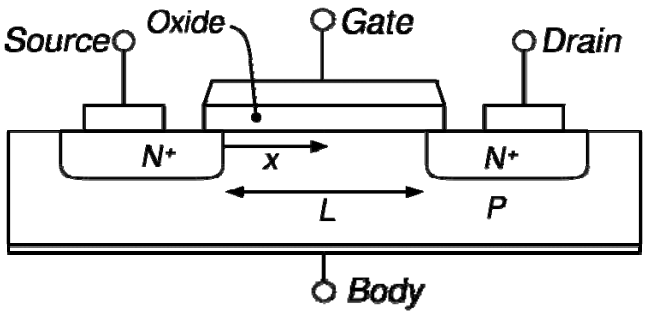
Moderní technologie návrhu číslicových elektronických obvodů má základ v tranzistorech MOS-FET, což je zkratka pro Metal Oxide Semiconductor Field Effect Transistor. V překladu to znamená „polem řízený tranzistor se strukturou kov, oxid, polovodič“. Tyto unipolární tranzistory se začaly používat namísto bipolárních proto, že v klidovém stavu neprotéká tranzistorem téměř žádný proud, což umožnilo integrovat mnohem větší množství tranzistorů na jednu křemíkovou destičku [27]. MOSFETy se v dnešní době používají v drtivé většině elektronických zařízení a proto je logickým krokem koncipovat vývojový software právě na ně.

MOSFETy mají dva základní typy a sice pMOS a nMOS, přičemž první písmenko zkratky symbolizuje způsob dotování. Právě proto, že jsou dva opačné typy MOS tranzistorů, se v elektronických obvodech vžila zkratka CMOS, kde C znamená „komplementární“. Použití těchto typů tranzistorů probíhalo určitým vývojem dokud se nezjistilo, že nejlepší je využití obou typů najednou. Odlišné vlastnosti obou typů umožnily jejich použití v různých částech obvodu a zajistily tak menší spotřebu proudu v klidovém stavu. Komplementárnost CMOS spočívá v tom, že v závislosti na vstupech obvodu je jedna část tranzistorů sepnutá a druhá rozepnutá, což umožňuje svádět na výstup buď napájecí napětí nebo zem.

Z pohledu velikosti jednotlivých tranzistorů na čipu došlo také k určitému vývoji, který samozřejmě nadále pokračuje. Zatímco v roce 1971 byly v procesoru Intel 4004 tranzistory s minimální velikostí 10μm, dnes, v roce 2007, se běžně setkáváme s velikostmi 90 a 65nm v procesorech Intel Pentium D. To vše koresponduje s tzv. Moorovým zákonem, který už od roku 1965 předpokládá zdvojnásobení počtu tranzistorů na čipu každých 18 měsíců a k tomu i odpovídající zdokonalení vlastností tranzistorů a technologie integrace [27]. Postup technologie ale nemůže pokračovat donekonečna vzhledem k fyzikálním limitům materiálů. Už v současnosti se Moorův zákon daří dodržovat jen kvůli paralelizaci procesorů. Další informace o budoucím vývoji technologie lze nalézt např. v [18].

5.1.1 Struktura a chování MOS tranzistorů

Struktura MOS tranzistorů vypadá následovně. Uvažujeme, že každý tranzistor má tři svorky označované jako S (source), D (drain) a G (gate), které odpovídají rozložení materálu v tranzistoru. Další součástí MOSFET je tělo (body), které je obvykle uzemněno pro případ nMOS a připojeno k napájení v případě pMOS. Obrázek 5.1 zobrazuje řez nMOS tranzistorem [29]. Tranzistor pMOS se liší tím, že jsou oproti nMOS prohozeny materiály v dopovaných oblastech.

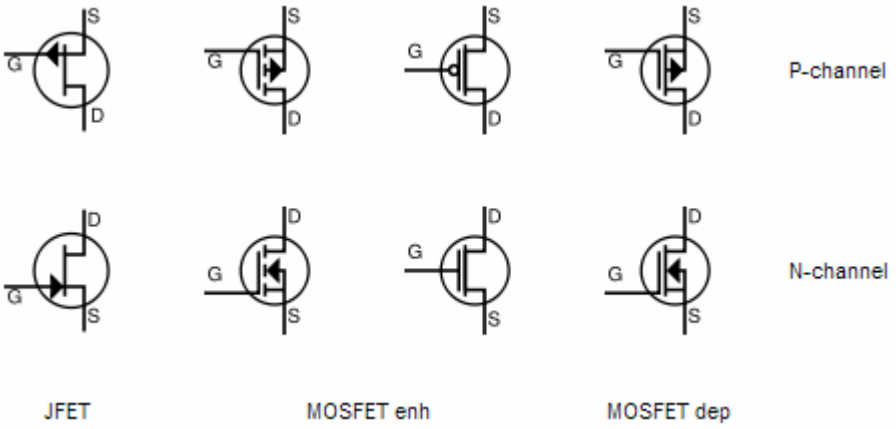


Obrázek 5.1: Řez nMOS tranzistorem [29]

Gate slouží jako ovládací svorka. Vezmeme-li v úvahu nMOS, pak stačí připojit k G napájecí napětí a tranzistor začne vést el. proud mezi svorkami S a D, protože rozdílné potenciály mezi gate a body vytváří elektrické pole v oblasti přilehlé k vrstvě oxidu, které přitahuje volné elektrony. Je-li napětí dostatečně vysoké, vytvoří se vodivý kanál a tranzistor považujeme za sepnutý. V případě, že je G připojena k zemi, nic se neděje a tranzistor je považován za vypnutý. V případě pMOS tranzistoru to funguje opačně. Logická nula (uvažujeme zde číslicové obvody) na G způsobí sepnutí, zatímco jednička zaručuje stav vypnuto.

Pokud jsou obě komplementární skupiny tranzistorových sítí v obvodu CMOS vypnuty, je na výstupu stav vysoké impedance označovaný Z. Pokud jsou obě sítě zapnuty, dochází k soupeření obou signálů na výstupu a takový stav se označuje jako neurčitá hodnota X.

Vzhledem k opačnému chování se tranzistory pMOS označují bublinkou u G, podobně jak je tomu u logických hradel s inverzní funkcí. Následující obrázek 5.2 [29] obsahuje běžné formy značení některých typů MOS tranzistorů. V této práci se bude nadále používat značení z třetího sloupce na obrázku s tím, že kroužek kolem značky bude z praktických důvodů vynechán.



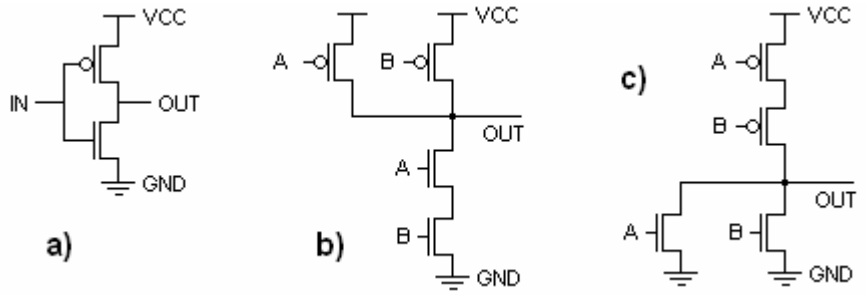
Obrázek 5.2: Formy značení FET tranzistorů [29]

Následující obrázek 5.3 demonstruje technologie CMOS aplikovanou na nejjednodušší logická hradla NOT, NAND a NOR. Hradlo NOT (a) funguje tak, že při signálu log. nuly se sepne horní pMOS tranzistor a propojí výstup s log. jedničkou na svorce napájecího napětí, zatímco spodní nMOS zůstává uzavřený. Při opačném vstupním signálu zůstává uzavřený horní pMOS a spodní nMOS je uzavřen, přičemž propojuje signál log. nuly na uzemňovací svorce s výstupem.

Podobně funguje 2-vstupové hradlo NAND (b). Pokud jsou oba vstupy v log. jedničce, jedině tehdy se otevřou oba spodní sériově spojené nMOS tranzistory. V jiném případě je cesta na zem uzavřená a jeden či oba horní paralelně zapojené pMOS tranzistory spojí výstup s napájením.

Funkce hradla NOR (c) je opět analogická. Pouze pokud jsou oba vstupy v log. nule se výstup propojí přes horní sériově propojené pMOS tranzistory k napájecímu napětí, jinak je cesta uzavřena a některý z dolních paralelně propojených nMOS tranzistorů propojí výstup s log. nulou.

Hradla s funkcemi AND a OR se vytváří jednoduše přidáním hradla NOT na výstup hradel NAND, resp. NOR.



Obrázek 5.3: Logická hradla a) NOT, b) NAND, c) NOR realizovaná CMOS

Důležitou poznámkou je úroveň napájecího napětí V_{CC} (také V_{DD} či v české terminologii U_{CC}), se kterou tranzistory pracují, tedy logická jednička. Nejstarší logika pracovala s hodnotou 5V, zatímco dnešní tranzistory akceptují jako log. jedničku hodnoty 3,3V, 2,5V, 1,8V, 1,5V atd., kdy napěťová úroveň závisí na výrobní řadě tranzistoru. Nízký potenciál reprezentující log. nulu je označován jako GND (také V_{SS} či česky „zem“) a nejčastěji jde o hodnotu 0V [27].

5.2 Další vlastnosti technologie CMOS

Obrázek 5.3 naznačuje některé další důležité vlastnosti CMOS technologie, které je třeba brát v úvahu při návrhu číslicových obvodů na úrovni tranzistorů.

První z těchto vlastností je zhoršení signálu log. jedničky resp. log. nuly, pokud napájecí napětí prochází tranzistorem pMOS resp. nMOS. Proto by např. hradlo neinvertujícího bufferu nemohlo být vytvořeno jako hradlo NOT s otočenými typy tranzistorů, protože by u něj docházelo k degradaci signálu a to by vadilo při zapojení většího počtu hradel za sebe. Neinvertující buffer se proto řeší pomocí dvou za sebou zapojených hradel NOT [27]. Návrhový systém by měl na tuto vlastnost pamatovat, aby nedocházelo k zbytečnému vyvíjení zmetkových obvodů.

Další vlastností je to, že prakticky neexistuje fyzikální důvod k tomu, rozlišovat svorky S a D, protože obě odpovídající části tranzistoru jsou shodné a nezáleží tedy na tom, jestli proud protéká od S k D či naopak. To může být výhodou při evolučním návrhu, kdy nemusíme svorky S a D rozlišovat při náhodném propojování.

Ostatní vedlejší vlastnosti MOS tranzistorů mohou způsobit další problémy při návrhu obvodů. Není smyslem této práce vyčíslovat různé přechodové charakteristiky či jiné složité děje, které probíhají uvnitř tranzistorů. Pro podrobnější informace je lepší se obrátit na vhodnou literaturu jako např. [27], případně na další odkazy na [28][29]. Chování tranzistorů v některých složitějších obvodech se těžko předvídá a proto je vhodné takové obvody simulovat, např. v simulátorech SPICE, čímž se dostáváme k další podkapitole.

5.2.1 Ověřování CMOS obvodů

Chceme-li si být jisti tím, že se navržený obvod chová správně, je třeba jeho funkci ověřit. Výhradní používání k tomu určených softwarových simulátorů typu SPICE aj. může být časově náročné. Proto se v mnoha případech přistupuje ke kompromisním řešením, jako jsou simulace na logické úrovni prováděné buď v jazycích typu C nebo ve speciálních HDL jazycích, které mají obvykle vlastní nástroje pro logické simulace obvodů, často s grafickým rozhraním. K simulaci pomocí SPICE se pak přistupuje jen jako k poslednímu ověřovacímu kroku nebo tehdy, chceme-li nějak ověřit či optimalizovat elektrické vlastnosti obvodu.

6 Důležité evoluční metody návrhu číslicových obvodů

Po úvodních kapitolách obsahujících teoretické základy pro simulaci elektronických obvodů, genetické programování, evoluční strategie a evoluční návrh hardwaru na různých úrovních nyní přistoupíme k možnostem, které nám skýtají kombinace předchozích témat.

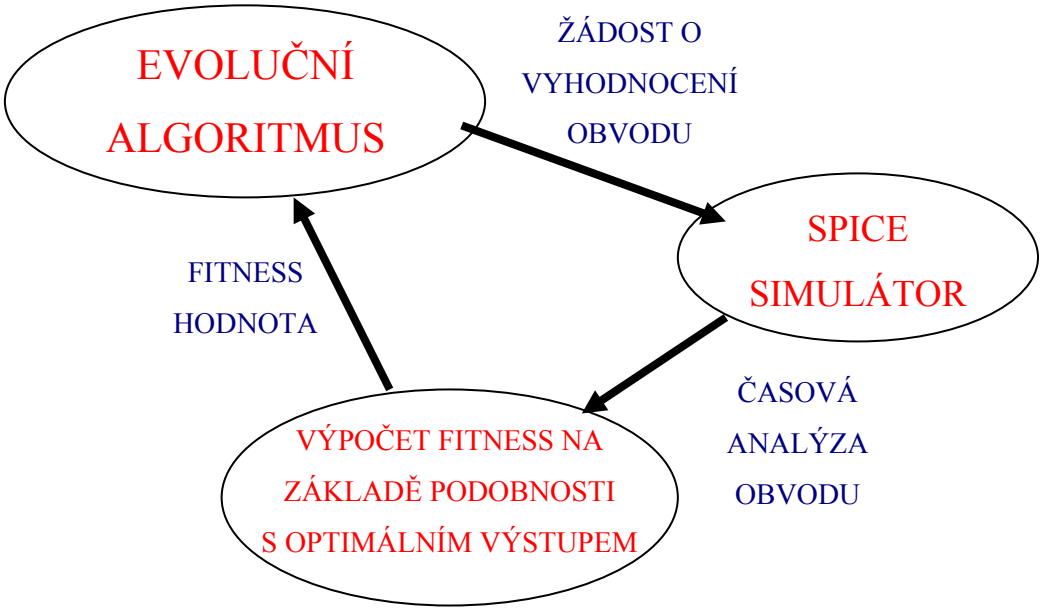
6.1 Simulace elektronických obvodů za běhu EA

V druhé kapitole jsme poznali, že je možné simulovat elektronické obvody v časové doméně pomocí výpočetních systémů, potažmo simulačního softwaru typu SPICE. Přesně tento typ simulace potřebujeme, chceme-li vyhodnocovat číslicové obvody.

V dřívějších dobách nebylo možné dostatečně implementovat do EA simulace v časové oblasti, protože jejich výpočet je podstatně složitější, než výpočet u simulací ve frekvenční oblasti [10]. Zejména v případě GP toto mohlo znamenat velký problém, neboť typická populace u běhu GP má až desítky tisíc kandidátních řešení. Výpočet fitness hodnot tedy znamená simulovat a analyzovat každého jedince v každé populaci.

V dnešní době jsou naštěstí k dispozici výkonnější výpočetní systémy, které se vejdou do stolního počítače či do serverové skříně a přesto dokáží pracovat rychleji než velké clustery používané před několika lety. Tato skutečnost znamená, že s přibývajícím časem (s přihlédnutím k Moorovu zákonu) budeme moci evolučně vyvíjet čím dál složitější číslicové obvody na úrovni tranzistorů.

Jak je taková evoluce vůbec proveditelná? Následující obrázek ilustruje, jakým způsobem dochází k vyhodnocování fitness hodnot jednotlivých obvodů:



Obrázek 6.1: Simulace obvodů v rámci evolučního algoritmu

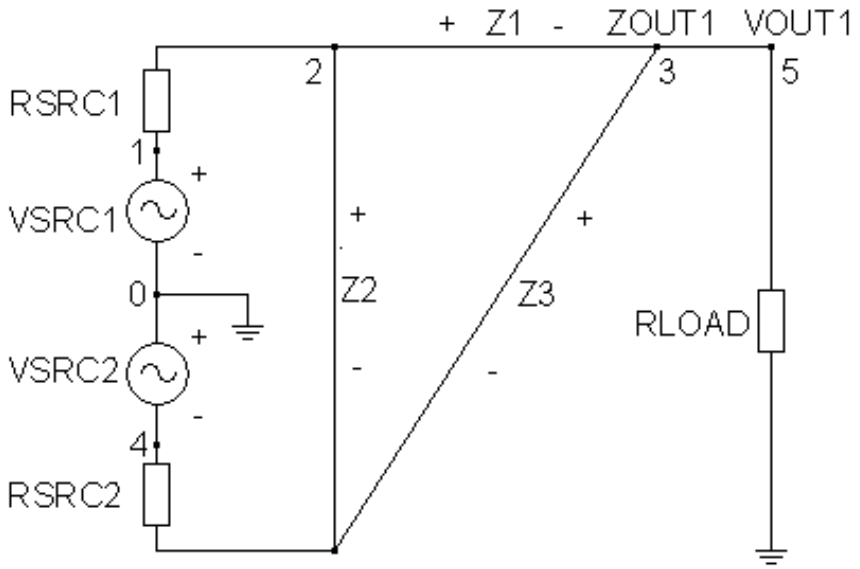
6.2 Development elektronických obvodů s GP

Nyní následuje popis metody, kterou byla již dříve použita [9, 10] a která by mohla být vhodným řešením problému zadaného v tomto projektu.

6.2.1 Reprezentace chromozomu

Uvědomíme-li si, že vlastně předem neznáme podobu a tedy ani velikost elektronických obvodů, které chceme nalézt (jinak bychom to nemuseli dělat), je celkem jasné, že je třeba najít takovou reprezentaci, která bude schopna vypořádat se s proměnnou délkou kandidátských obvodů. Stromová struktura GP je tedy z tohoto hlediska vhodným kandidátem.

Bohužel, kromě proměnné velikosti nastává další, podstatně závažnější problém. Je třeba namapovat schéma elektronického obvodu, které je cyklickým grafem, na stromovou strukturu chromozomu GP, která už z principu cykly nepodporuje. Odpovědí na tento problém je právě metoda, kterou nazýváme developmentem. Její princip spočívá v tom, že program, který je reprezentován stromem GP, je předpisem pro sestavení elektronického obvodu podle předem definovaných pravidel. Mapování z programu probíhá tak, že si před touto procedurou připravíme obvod, který se skládá ze dvou základních částí, a sice pevné části a embryonální, proměnlivé části. Obrázek 6.2 demonstruje takový počáteční obvod připravený pro syntézu dvouvstupového logického hradla [10]:

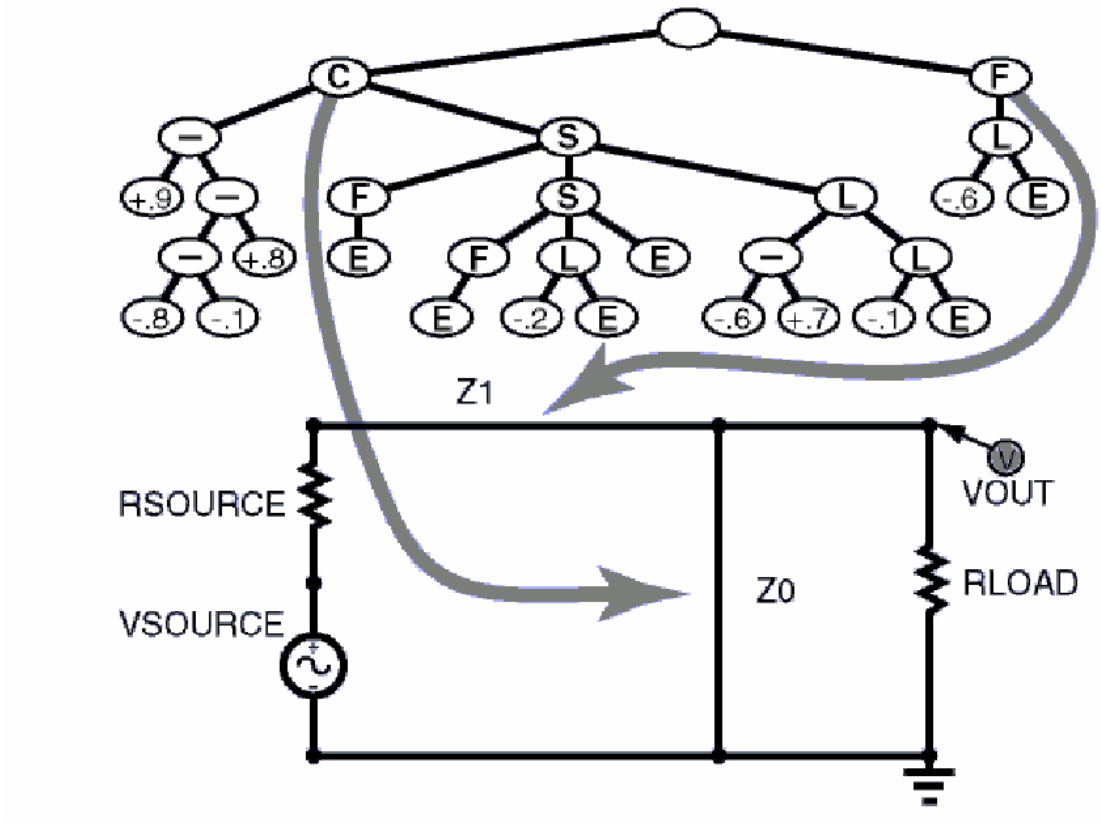


Obrázek 6.2: Embryonální obvod se dvěma vstupy a jedním výstupem [10]

Prvky Z1, Z2 a Z3 na obrázku jsou modifikovatelné vodiče, které dohromady tvoří embryonální část obvodu. Všechny ostatní prvky jsou pevné.

6.2.1.1 Příklad developmentu jednoduchého elektronického obvodu

Jiným příkladem pro ilustraci postupu při developmentu může být elektronický obvod, který má být vytvořen dle předpisu obsaženém v chromozomu na obrázku 6.3 [20].

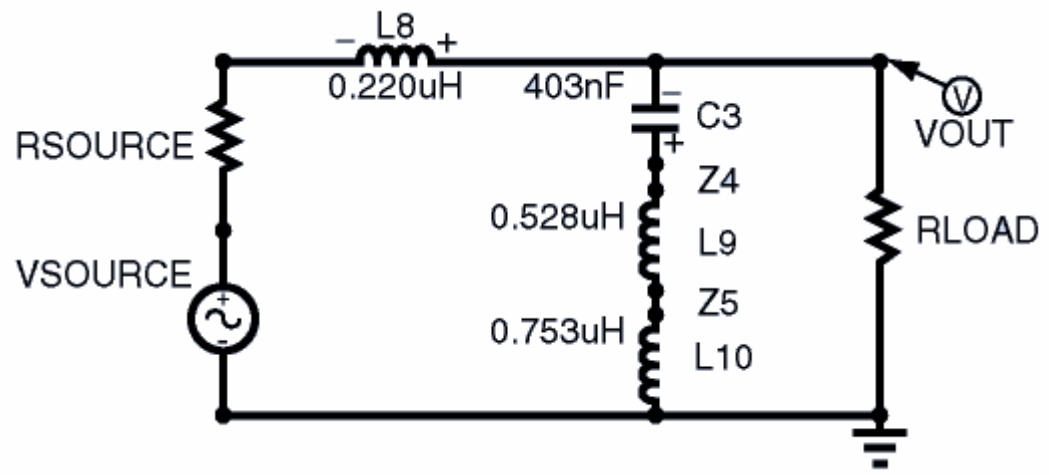


Obrázek 6.3: Ilustrace developmentu na embryonálním elektronickém obvodu [20]

Jak je vidět z obrázku, embryonální obvod se skládá ze dvou modifikovatelných vodičů Z0 a Z1, přičemž ke každému z nich náleží jedna vývojová větev stromu.

Budeme se nejprve zabývat větví stromu příslušející vodiči Z0: Nejprve namísto vodiče vytvoříme nový kondenzátor, kterému přiřadíme hodnotu 403 nF podle aritmetického podstromu na levé straně. Další uzel (S) naznačuje, že je potřeba rozdělit větev obvodu na 3 sériové části (S má 3 parametry – podstromy) První z nich bude kopie původního kondenzátoru s převrácenou polaritou (F), přičemž list s označením (E) znamená konec větve. Druhá bude nový modifikovatelný vodič, protože parametrem je další (S). Tento vodič se dále rozdělí na tři modifikovatelné vodiče. První z nich obrátí svou polaritu (F) a skončí (E). Druhý vodič se změní na cívku (L) s hodnotou 0,528 μH a skončí. Třetí modifikovaný vodič ihned skončí. Vrátíme-li se o úroveň výše, zbývá popsat třetí sériovou větev , která se rozdělila nejvyšším příkazem (S). Větev je nadepsána L, takže se původní druhý vygenerovaný sériový kondenzátor změní na cívku (L), jejíž hodnota je nejprve nastavena jejím levým podstromem na 0,041 μH a poté je pravým podstromem obsahujícím zbytečný příkaz na vytvoření cívky změněna na 0,753 μH. Poté tato větev končí svůj vývoj.

Zbývá dokončit větev přidruženou embryu Z1: Nejprve se přehodí polarita modifikovatelného vodiče (F), ten je poté změněn na cívku (L), nastaven se na hodnotu 0,22 μH a větev je ukončena. Výsledný obvod je na obrázku 6.4 [20].



Obrázek 6.4: Výsledný obvod získaný developmentem [20]

K tomu, aby bylo možné s embryem pracovat tak jako v předcházejícím příkladu, musíme před započetím algoritmu provést přípravné kroky, tak jak jsou popsány v kapitole 3, abychom určili metodu zakódování obvodu do chromozomu.

6.2.2 Určení množiny funkcí a terminálů

Množina funkcí se bude skládat ze dvou typů funkcí [10]:

- Aritmetických funkcí pro reprezentaci číselných hodnot – parametrů součástek v obvodu a
- funkcí pro konstrukci obvodu.

Parametry součástek se určují ve vlastním aritmetickém podstromu, který je tvořen reálnými čísly a matematickými funkcemi. Pro to nám stačí funkce + a -.

Určení funkcí pro konstrukci obvodu je jen o málo složitější. Nejprve do množiny funkcí přidáme takové, které vytvářejí součástky. Budeme-li vytvářet logické hradlo, použijeme několik funkcí na vkládání PNP a NPN tranzistorů, stejně jako diod a rezistorů. Mezi elektroinženýry je

běžnou znalostí, že logická hradla se skládají právě z těchto součástek a proto není odborná znalost vkládaná do algoritmu příliš velká. Dalšími vhodnými funkcemi může být propojení se zemí a propojení s jedním ze dvou vstupních zdrojů. Dále je nutné vložit do množiny funkce, které budou ovlivňovat větvení obvodu a směrování vodičů, tedy příkazy na vytvoření sériového a paralelního vodiče a převrácení jeho polarity. Autor [10] ve své práci uvádí ještě další funkce, jako je vložení tranzistoru s uzemněným emitorem a prázdnou operaci.

Zbývá připravit množinu terminálů. Ty budou dva: První, který změní modifikovatelnou komponentu, se kterou je propojen na nemodifikovatelnou a tak ukončí vývoj větve obvodu. Druhý bude sloužit k tomu, aby smazal komponentu z obvodu a zároveň udržel jeho správnost.

6.2.3 Vyhodnocení fitness

S vytvořenými množinami funkcí a terminálů je nyní poměrně snadné si představit fázi mapování chromozomu na obvod. Začne se s embryem a postupně se provádějí funkce v programovém stromu, dokud se nedojde na konec. Výsledkem by měl být typický netlist (samozřejmě je třeba ještě přepsat každou funkci do správné SPICE syntaxe), který můžeme dodat simulátoru a tak získat výsledný průběh v časové doméně. Vyhodnocení je závislé na daném problému, v případě logického hradla budou oba vstupy postupně nabývat všech 16 možných přechodů mezi čtyřmi možnými kombinacemi dvou vstupů.

Výsledek simulace je porovnán s referenčním průběhem a je vypočítána fitness hodnota, která znamená součet absolutních hodnot odchylek napětí od ideálního případu [10]. Absolutní hodnoty by měly být váhovány podle přijatelnosti úrovní napětí v tolerančních hranicích logického modelu.

Fitness se poté chápe tím způsobem, že čím nižší hodnota, tím lepší.

6.2.4 Ostatní přípravné kroky

Zbývá uvážit parametry GP algoritmu a ukončovací kritérium. Parametry jako jsou velikost populace a maximální velikost stromu je třeba uvážit podle problému a experimentovat, dokud se nedosáhne ideálních výsledků. Rovněž je vhodné do algoritmu zařadit syntaktická omezení, jež jsou vysvětlena v kapitole 3.

Ukončovacím kritériem by mělo být nalezení obvodu splňujícího veškeré toleranční limity cílového návrhu.

6.2.5 Výsledky metody

Prof. Koza ve své práci [10] jasně uvádí, že jeho metoda zaznamenala úspěch a do data vydání knihy se podařilo znovu vynaleznout 19 před tím patentovaných elektronických obvodů a to bez větší znalosti dané problematiky, pouze se zadáním součástek a žádaného výsledku. To rovněž podle prof. Kozy znamená, že dosažené výsledky jsou výsledkem umělé inteligence schopné soupeřit s lidskou invencí. Mezi výsledky jsou i takové obvody jako logické hradlo NAND, zesilovač se zpětnou vazbou, ALU jednotka atd.

Existují však námitky k tomuto řešení [31]. První z nich je ta, že zakódování je nepřímé a tedy se zvyšuje náročnost výpočtu. Další je v tom, že proces je víceméně nezávislý na problému a tak vytváří příliš velké obvody tam, kde by specializovanější nástroje mohly ušetřit počet součástek. Dle mého názoru je tato námitka přeceněná, protože v prvé řadě chceme objevit nová inovátorská řešení a optimalizace bývá zpravidla až dalším krokem. Poslední námitka hovoří o tom, že navrhovaná

metoda vytváří obvody se součástkami, jejichž hodnoty se průmyslově nevyrábí. To by se ale dalo napravit malým zásahem do postupu – reprezentací hodnot např. celými čísly, která by mapovala parametry součástek do tabulek s existujícími hodnotami.
Na FIT VUT v Brně byl v nedávné době postup úspěšně zopakován [7], ovšem s tím, že šlo o analogové obvody bez tranzistorů.

6.3 Kartézské genetické programování

Další důležitou metodou návrhu číslicových obvodů je Kartézské genetické programování (CGP). Tato metoda je dílem pánů Millera a Thompsona z Univerzity v Birminghamu [14] a byla krátce zmíněna v souvislosti s evolučním návrhem elektronických obvodů na úrovni hradel a funkčních bloků v kapitole 4. V této kapitole bych rád vysvětlil CGP podrobněji, protože má důležitý vztah k této práci, který bude objasněn v dalších kapitolách o implementaci návrhového systému.

6.3.1 Základní vlastnosti CGP

CGP je evoluční metoda založená na GP. Z malé ukázky v kapitole 4 se to může zdát podivné, nicméně podobnost začne být zřejmá, podíváme-li se na obrázky struktur, které obě metody kódují. Zatímco CGP představuje problém ve formě acyklického orientovaného grafu, GP kóduje stromy, což jsou vlastně jisté formy grafů. CGP slouží pro evoluci na úrovni hradel či funkcí, které jsou vlastně seřazeny za sebou podle propojení prvků a jsou v tomto pořadí aplikovány na vstupní proměnné. V tom lze nalézt analogii s programy, které jsou kódovány ve chromozomech GP.

Chromozom CGP obsahuje řadu celých čísel, která mají význam buď jako propojení uzlů v obvodu nebo jako funkce, kterou každý uzel provádí. Každý uzel je reprezentován jako n-tice celých čísel, přičemž n-1 prvních hodnot představuje propojení vstupů uzlu k výstupům předchozích uzlů nebo k vstupům celého obvodu. Poslední hodnota n-tice je pak zakódování funkce uzlu. Za n-ticemi následuje m dalších hodnot, které reprezentují propojení výstupů obvodu k výstupům jednotlivých buněk. Každému výstupu uzlu je totiž přiřazeno číslo v posloupnosti následující za čísly vstupů obvodu. Máme-li tedy obvod skládající se z k n-vstupových buněk a m výstupů, můžeme spočítat délku L chromozomu následovně:

$$L = k \cdot (n+1) + m \qquad (6.1)$$

Parametry CGP, které uvažujeme při zakódování problému a návrhu algoritmu včetně těch, které byly zmíněny výše, jsou [18]:

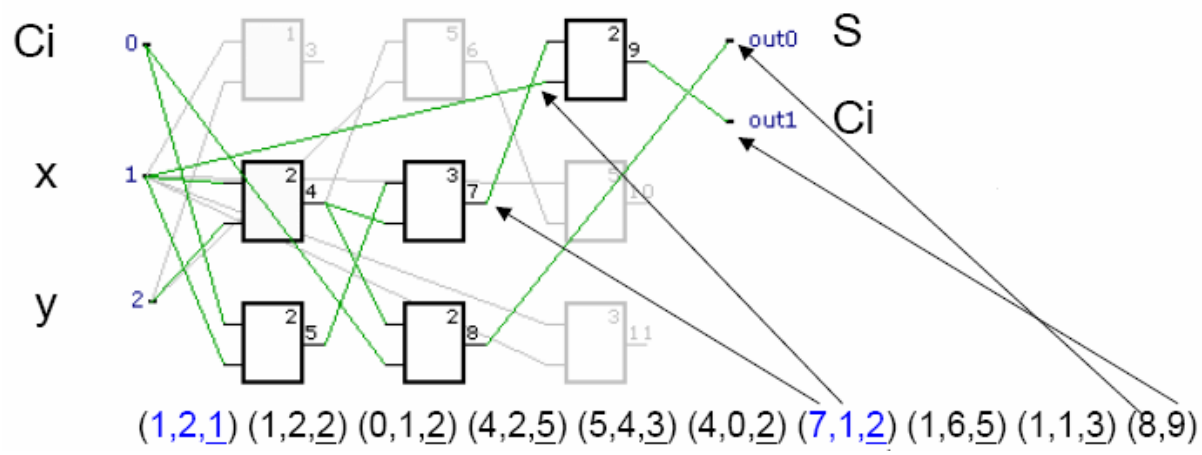
- Počet vstupů obvodu,
- počet výstupů obvodu,
- počet vstupů jednoho uzlu,
- množina funkcí použitelných v uzlech,
- počet řádků v mřížce uzlů,
- počet sloupců v mřížce uzlů.

Proč jsou uzly organizovány do mřížky bude vysvětleno dále. Pro lepší pochopení parametrů CGP je vložen obrázek 6.5 [18]. Obvod na obrázku odpovídá chromozomu (1, 2, 1) (1, 2, 2) (0, 1, 2) (4, 2, 5) (5, 4, 3) (4, 0, 2) (7, 1, 2) (1, 6, 5) (1, 1, 3) (8, 9). Jde o úplnou sčítačku a parametry CGP jsou následující:

- Počet vstupů: 3
- Počet výstupů: 2

- Počet vstupů jednoho uzlu: 2
- Množina funkcí použitelných v uzlech: NAND(0), NOR(1), XOR(2), AND(3), OR(4), NOT(5)
- Počet řádků: 3
- Počet sloupců: 3

Šedě jsou na obrázku vyznačeny nepoužité uzly/hradla.



Obrázek 6.5: Ukázka zakódováni úplné sčítačky v CGP [18]

Důležitým parametrem CGP je i tzv. L-back parametr. Jeho hodnota vyznačuje, zda je při mutaci povoleno připojovat vstupy uzlu na výstupy libovolných uzlů před ním či ne. Běžné omezení v CGP je to, že není možné připojit uzel na stejný sloupec nebo sloupce za ním. Hodnota L-back = 0 zanamená, že není omezeno připojování se na sloupce před uzlem. Jakékoli jiné kladné znamená počet sloupců, ke kterým je možné se připojit dopředu směrem od každého uzlu. Některé verze CGP rozlišují mezi tím, zda L-back platí pro vstupy obvodu či ne. Příklad na obrázku 6.5 má nastaveno L-back buď 2 nebo 3 v závislosti na tomto rozlišení, jak dokazuje pravý horní uzel, který je propojen přímo ke vstupu obvodu. Pokud je L-back nastaven na 1, mění se tak na efektivní prostředek pro zajištění podpory pro HW zřetězení obvodu. Pokud je nastaven na 0 a počet řádků v obvodu je 1, znamená to maximální možné propojení ke všem předchozím uzlům [18].

Obvod zakódovaný v GCP má velkou redundanci, která může být několika typů [14]:

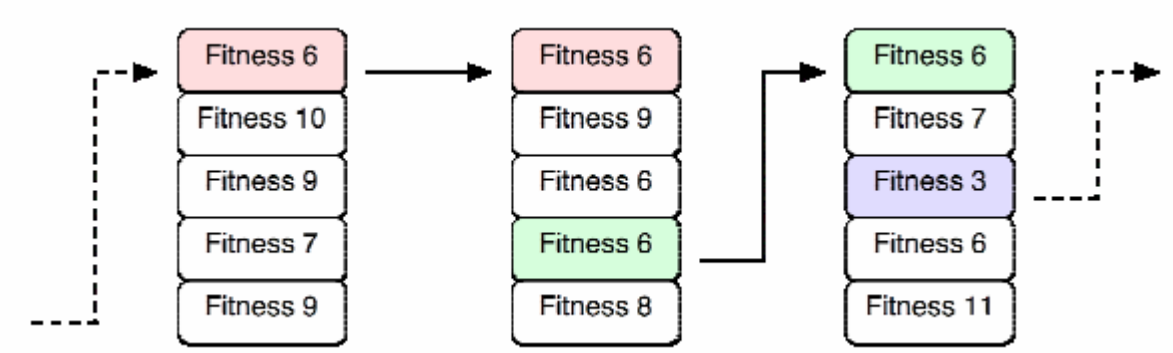
- Může obsahovat množství uzlů, které nejsou propojeny do výsledného obvodu (na obrázku 6.5 označeny šedě), protože nemají vliv na výsledek – nejsou propojeny s výstupy obvodu.
- Funkční redundance někdy označovaná jako *bloat*. Její význam je v tom, že množství uzlů je propojeno ve funkci, která by mohla být řešena menším počtem uzlů. Klasickým případem je funkce NOT(NOT(NOT(A))), která by mohla být nahrazena jedním NOT(A).
- Redundance vstupů, kdy uzel má více vstupů, než je využito některými funkcemi – typicky opět NOT, který využívá jen jeden vstup.

Redundance má v CGP velký význam, neboť obvody v sobě mohou obsahovat „spící“ vlastnosti, které mohou být kdykoli probuzeny mutací a třeba se významně podílet na zvýšení fitness hodnoty.

6.3.2 Algoritmus CGP

Algoritmus CGP se velmi shoduje s algoritmem evoluční strategie (1 + λ) z podkapitoly 3.3.2.2. ES použitá v CGP má pouze ten rozdíl, že dochází k evoluci chromozomů s celočíselnými hodnotami a tedy neuvažujeme odchylku v rozložení pravděpodobnosti. Namísto toho jsou generována náhodná čísla s tím, že jsou přijata jen čísla, která nejsou v rozporu s L-back parametrem či nějakým jiným omezením.

Dalším rozdílem může být metoda výběru nejlepšího jedince do další populace. Obvykle se totiž vybírá potomek, pokud má stejnou fitness jako rodič. Pokud má několik jedinců stejnou fitness, ale jiné zakódování, označujeme je jako neutrální. Výběr neutrálního jedince namísto původního se nazývá *neutrální prohledávání*. To by podle J.Millera [14] mělo zajistit lepší výsledky algoritmu. Obrázek 6.6 [18] ilustruje použití tohoto pravidla.



Obrázek 6.6: Průběh výběru nejlepšího jedince s přihlédnutím k pravidlu neutrálního výběru [18]. Lepší fitness zde má menší numerickou hodnotu.

Jedním z faktorů ovlivňujících úspěšnost může být i četnost mutací, jejíž optimální nastavení ale může záviset na řešeném problému. Četnost je možné nastavit na pevný počet mutací na jednoho jedince nebo na pravděpodobnost zmutování každého genu.

6.3.3 Výsledky metody

CGP má jako mnohé jiné evoluční metody několik problémů:

- Ve fitness funkci je nutné ohodnocovat všechny vstupní kombinace,
- na úrovni hradel je metoda schopná navrhovat obvody do cca. deseti vstupů a
- s rostoucí složitostí navrhovaného obvodu klesá počet úspěšných běhů evoluce.

V porovnání s klasickým GP se CGP jeví jako rychlejší, protože nepotřebuje složitě mapovat programy na obvody. V GP neprobíhá velké množství generací (stovky), ale populace má obrovské množství jedinců (stovky až tisíce). CGP má málo jedinců v populaci (jeden rodič a max. desítky potomků), ale může běžet obrovské množství generací (až miliony) – počet vyhodnocení fitness je tedy zhruba srovnatelný. Nižší rychlost GP v mapování tedy může být rozhodující. Otázkou ovšem zůstává, jestli by CGP bylo možné upravit tak, aby vyvíjelo obvody na úrovni tranzistorů.

Na úrovni hradel a funkcí je CGP nicméně úspěšné, na FITu [18] i jinde se podařilo objevit množství obvodů včetně mediánů, řadicích sítí či násobiček. Podrobnější výsledky a srovnání lze najít v literatuře.

7 Návrh a implementace vývojového systému

Tato kapitola popisuje návrh a implementaci vývojového systému pro evoluční návrh číslicových obvodů na úrovni tranzistorů.

Dále je v této kapitole popsáno, jakým způsobem a proč byly vybírány metody a algoritmy navrhovaného vývojového systému, přičemž je vysvětlena funkce doposud nezmíněných metod. Dále je analyzováno, jaké vstupy by takový systém měl mít a jaké byly nakonec implementovány. Rovněž je popsáno evoluční jádro systému na úrovni jednotlivých částí algoritmu.

Protože bylo předpokládáno použití knihovny GAlib pro C++ (ke kterému nakonec nedošlo), byla vyžadována vysoká rychlost provádění programu, bylo od začátku rozhodnuto o implementaci v jazyce C++.

7.1 Výběr metody

7.1.1 Použitý evoluční algoritmus a reprezentace problému

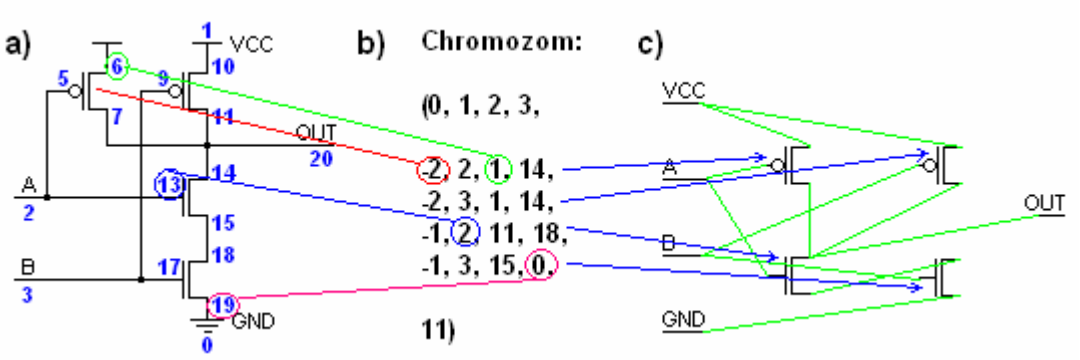
Prvním krokem při návrhu nějakého evolučního systému bývá rozbor toho, co chceme vlastně vyvíjet a jakým způsobem to budeme reprezentovat s přihlédnutím k tomu, jaké máme výpočetní prostředky. U návrhu obvodů na úrovni tranzistorů je ale situaci komplikovanější, protože nejde o triviální problém a ne všechny metody jsou tento problém vhodné.

V předchozím textu byla popsána jediná metoda, která byla použita pro návrh na úrovni tranzistorů. Zmíněnou metodou bylo GP a cílový obvod zde byl reprezentován nepřímo, tedy jako předpis či program na jeho sestavení technikou developmentu. Tato metoda vyvíjela hradla obsahující prvky jako bipolární tranzistory PNP a NPN, diody, rezistory a další součástky [10]. Další možností je použití technologie CMOS, která využívá jen pMOS a nMOS tranzistory. V této práci se zaměříme právě na technologii CMOS. Díky absenci ostatních součástek a „spínačovému“ chování MOS tranzistorů není nutné brát v prvotní fázi návrhu v úvahu parametry tranzistorů a to, co nás tedy zajímá, je pouze propojení součástek.

Návrh obvodu obsahující pouze dva typy tranzistorů je velice podobný návrhu na úrovni hradel s tím, že pravidla propojení jsou zde složitější a simulace probíhá jinak. To ovšem nijak neovlivňuje reprezentaci problému, která může být taková, že se bude podobat zakódování běžně používanému v CGP. Každý z t tranzistorů může být reprezentován čtveřicí celých čísel, přičemž jedno z čísel bude reprezentovat typ tranzistoru a další tři budou určovat, na které svorky ostatních tranzistorů, vstupy obvodu či napájení nebo zem budou připojeny svorky G, S a D tranzistoru. Na konci sekvence bude ještě n čísel reprezentujících připojení n výstupů obvodu. Aby byla jednodušší práce s reprezentací na úrovni programu, bude začátek sekvence obsahovat nulu a jedničku reprezentující GND a UCC. Za nimi bude následovat k-tice prázdných pozic reprezentujících k vstupů obvodu. Každé číslo svorky tak bude odpovídat pozici v řetězci čísel. Výpočet délky chromozomu každého jedince potom vypadá následovně:

$$L = 2 + k + t \cdot 4 + n \qquad (7.1)$$

Následující obrázek 7.1 pomáhá objasnit způsob reprezentace na příkladu obvodu dvouvstupového hradla NAND. Na obrázku je skutečný obvod (a), dále je přiložen příslušný chromozom (b) se zvýrazněním některých svorek a indexů v chromozomu jim odpovídajících. Délka chromozomu je dle rovnice 7.1 rovna 21, protože obvod je složen ze čtyř tranzistorů, má dva vstupy a jeden výstup. Rovněž je zobrazeno rozmístění a propojení tranzistorů v mřížce (c), protože jde o metodu inspirovanou CGP.



Obrázek 7.1: a) Skutečný obvod NAND-2 (modrá čísla zde indikují indexy svorek v chromozomu) s b) odpovídajícím chromozomem a c) reprezentací obvodu v mřížce CGP. První čtveřice čísel symbolizuje napájecí svorky a vstupy obvodu, kde hodnota znamená index sama sebe. Další čtveřice symbolizují tranzistory, kde první číslo je typ tr. (-2: pMOS, -1: nMOS) a další jsou připojení G, S a D k jiným svorkám, kde čísla znamenají jejich indexy v chromozomu. Poslední číslo řetězce udává připojení výstupu.

Navržená reprezentace byla shledána dostatečnou, protože postihuje všechny možnosti zapojení jako jsou uzly a neomezuje takové vlastnosti CMOS, jako je např. možnost prohození funkce svorek S a D. Problém reprezentovaný jako CGP má navíc tu výhodu, že odpadá časově náročné mapování genotypu na fenotyp (generování obvodu dle předpisu).

Vzhledem k zmíněnému zjednodušení problému, reprezentaci podobné CGP a nižší časové náročnosti jsem se tedy nakonec rozhodl implementovat vlastní systém založený na CGP, který si ovšem bude bere za inspiraci některé prvky použité v návrhu tranzistorových obvodů s GP, zejména použití syntakticky omezených struktur (podkapitola 3.2.3.2), které jsou při vysokém počtu možných kombinací propojení nutností.

Z pohledu evolučních metod se systém drží CGP, tedy používá ES (1 + λ). Protože chceme evolvovat funkční obvody, musí být hlavním ukončovacím kritériem evoluce dosažení maximální fitness hodnoty. Protože jde ale o diplomovou práci, jsou výpočetní možnosti omezeny na servery FITu, kde probíhá množství jiných experimentů a čas je omezen, navíc není zaručeno, že EA skončí úspěšně v konečném čase, bylo přidáno i omezení počtu generací. Hrubý návrh algoritmu vypadá následovně:

```
Náhodně vygeneruj počáteční populaci (dbej na omezení);
Generace := 0;
while ((Generace <= MAXGEN) and (fit(rodič) < MAXFIT)) begin
      Vygeneruj λ potomků z aktuální populace, každého pomocí
      mutace N náhodných genů (dbej na omezení);
      For i=1 to λ do
            if (fit(rodič) <= fit(potomek[i])) then
```

```
                    rodič = potomek[i];
            inc(Generace);
    end;
    Tiskni výsledky;
```

7.1.2 Způsob simulace obvodů

Po zvolení evoluční metody a reprezentace je ještě třeba rozhodnout, jakým způsobem budou simulovány vytvořené obvody, aby mohla být ohodnocena jejich fitness.

Po dohodě s vedoucím bylo rozhodnuto, že bude prováděna pouze simulace obvodů na logické úrovni, vzhledem k vysoké časové náročnosti simulace v systémech SPICE a už tak velké náročnosti projektu jako takového. SPICE má být využit jen pro ověření některých vygenerovaných obvodů, které nebudou odpovídat známým zapojením v dostupné literatuře (např. [27]).

Pro vyhodnocování fitness byl tedy vyvinut vlastní simulátor CMOS obvodů na logické úrovni, který bude popsán v níže. Právě protože jde pouze o logickou úroveň, je maximální fitness hodnota rovna množství správných výstupů pro všechny kombinace na vstupech obvodu dle rovnice 7.2:

$$MaxFit = 2^N \cdot V , \qquad (7.2)$$

kde N je počet vstupů obvodu a V je počet výstupů obvodu.

7.2 Vstupní parametry systému

Protože nejde jen o návrh EA, ale o celý vývojový systém, bylo nutné zvážit způsob předávání parametrů algoritmu tak, aby bylo pro uživatele pohodlné systém používat a zároveň bylo možné parametry snadno měnit. Ke zkoušení různých parametrů evoluce při experimentech dochází velmi často a je to přímo nutné, chceme-li z EA dostat kvalitní výsledky [5].

Pro vyšší komfort byl zvolen způsob předávání skrze vstupní soubor. Systém není navržen pro grafický výstup, protože drtivá většina testovacích experimentů byla spouštěna na školních serverech zapojených do Sun Grid Engine (SGE), což je systém, který se nejefektivněji ovládá dávkovými soubory z příkazové řádky.

Mezi nejdůležitější parametry předávané systému byly od začátku návrhu následující:

- Seznam trénovacích vektorů se správnými výstupními hodnotami,
- maximální počet generací,
- počet tranzistorů a
- velikost parametru λ u ES.

Počet tranzistorů byl později změněn na velikost mřížky tranzistorů, protože bylo dodáno omezení rozsahu připojení svorek v duchu parametru L-back u CGP, které ale narozdíl od L-back omezuje maximální vzdálenost připojení svorek tranzistoru v mřížce. Parametr byl nazván „radius“ a je možné ho neomezovat, stejně jako L-back.

Později, v průběhu návrhu a testování, byla dodána možnost nastavení následujících parametrů, včetně změny ve způsoby určení počtu tranzistorů:

- Počet řádků mřížky tranzistorů,
- počet sloupců mřížky tranzistorů,
- počet mutací na jednoho jedince,
- radius a
- invertované výstupy (bude vysvětleno později).

Dalším důležitým parametrem, který je ale předáván přímo jako parametr u spouštěcího souboru, je inicializátor náhodného generátoru čísel, protože jednotlivé evoluční běhy dávky úkolů často bývají v SGE spouštěny v jeden okamžik a inicializace náhodného generátoru jen pomocí času může způsobit množství stejných běhů, takže se časový údaj raději násobí zmiňovaným parametrem, který může odpovídat např. pořadí v dávce.

Výstupy systému probíhají do jiného výstupního souboru ve tvaru: Výpis chromozomu nejlepšího jedince, jeho fitness hodnota a generace, ve které byla fitness dosažena.

7.3 Popis funkcí navrženého algoritmu

Po tom, co byly specifikovány vstupy, výstupy a použitá metoda návrhu, je na řadě podrobnější popis implementace vývojového systému, konkrétně jeho evolučního jádra. Tato podkapitola se zabývá reprezentací, fitness funkcí, genetickým operátorem mutace a syntaktickými omezeními.

7.3.1 Reprezentace obvodu

Reprezentace je stejná, jak byla popsána v podkapitole 7.1.1, avšak je třeba doplnit některé detaily. Prvním takovým detailem je vysvětlení zakódování typu tranzistoru.

Na obrázku 7.1 o několik stránek výše je vidět použití obou typů tranzistorů. V chromozomu jsou tranzistory pMOS zakódovány hodnotou „-2“, zatímco nMOS jsou zakódovány hodnotou „-1“. Nabízí se otázka, proč nejsou zakódovány jako „0“ a „1“ podle toho, na který signál se otevírají. Odpověď není příliš složitá: Záporná čísla jsou použita, aby se při simulaci obvodu a kontrole syntaktických omezení nepletla čísla označující typy tranzistorů s indexy svorek GND a U_{CC}. Při zjišťování otevřenosti tranzistoru je pak k typu přičtena hodnota 2 a to je porovnáno se signálem na svorce G. Je to rychlejší způsob, než kontrolovat pozici indexu v obvodu při procházení chromozomu.

Hodnoty na pozicích reprezentujících svorky GND a U_{CC} a vstupy jsou stejné jako jejich index proto, aby se vyhnulo případům, kdy by se odkazovaly na nesmyslné indexy, kdyby nebyly inicializovány. Pokud chceme připojit nějakou z těchto svorek k tranzistoru, uděláme to jednoduše na indexech svorek tranzistoru.

7.3.2 Inicializace

Inicializace počátečního jedince je poněkud komplikovanější, něž běžně v EA bývá. Typické je inicializovat náhodně jeden po druhém všechny geny (s přihlédnutím k případným syntaktickým omezením), dokud se nepovede inicializovat celý chromozom.

V navrhovaném systému je to ale jinak, protože syntaktická omezení jsou poměrně přísná a inicializace by tedy nemuselo být dosaženo vůbec – program by cyklil a pokoušel se o možná vůbec neproveditelnou mutaci (rozuměj naplnění genu legální hodnotou) stále dokola. Kvůli tomu musí být při inicializaci dodrženo zvláštní pořadí, aby se nejdůležitější geny dostaly do chromozomu.

Následuje algoritmus inicializace v pseudokódu:

```
nuly := 0;  jedničky := 0;
Naplň první (2 + vstupů) geny hodnotou jejich indexu;
Ostatní geny naplň hodnotou –3;
Všechny geny typů tranzistorů nastav buď na -2 nebo -1;
for each (gen in chromozom)
```

```
    if (random() % 100 < 30) and (gen = S or D) then
            if (tr = -1) then begin
                gen := 0;
                inc(nuly)
            end
            else begin
                gen := 1;
                inc(jedničky);
            end;
for each (gen in chromozom)
    if (gen = -3)
            repeat mutuj(gen) until (mutaceLegalni(gen));
if (jedničky < 1) then
      repeat
            mutuj(random(gen));
            if not mutaceLegalni(gen) then
                nastavPuvodni(gen);
      until (jedničky > 0);
Poslední if zopakuj pro nuly;
```

Jak napovídá algoritmus, před náhodným naplněním všech genů se nejprve inicializují geny odpovídající typům tranzistorů, protože tam může vznikat nejvíce omezení. Dalším krokem je 30% pravděpodobnost připojení svorek S a D na GND či U_{CC}, dle typu tranzistoru. Důvodem je to, že při hustém propojení, které vzniká v chromozomu, je obtížné legálně připojit nějaké svorky tranzistoru k U_{CC} nebo GND, které jsou nutností pro úspěšnou funkci obvodu. Ze stejného důvodu se na konci inicializace kontroluje, zda jsou obsažena nějaká připojení na GND a U_{CC}. Pokud ne, zkouší se maximálně 100000krát náhodně změnit nějaký gen na příslušnou hodnotu. Pokud se to v oněch 100000 pokusech nepovede, je do výstupního souboru oznámen neúspěch a pokus je ukončen.

Počet úspěšných pokusů o inicializaci je omezen proto, že i přes posloupnost inicializačních operací se stále může stát, že obvod nesplňuje vhodné podmínky a není možné legálně inicializovat některé nezbytné vlastnosti, což by mohlo vést k cyklení nebo trvat neúnosně dlouho, dokud by se inicializace nepovedla. To si není možné dovolit, protože počet spuštěných procesů na SGE je omezen a byla by škoda plýtvat prostředky a časem. V rámci experimentu nevadí, když několik jednotlivých pokusů selže hned na začátku, protože se jich spouští větší počet najednou.

7.3.3 Fitness funkce

Fitness funkce je jednou z nejkomplikovanějších součástí systému. V podstatě bylo nutné navrhnout simulátor CMOS obvodů na logické úrovni.

Simulátor pracuje s celým chromozomem a také s dvourozměrným polem trénovacích vektorů, kde sloupce označují vektor a řádky vstupy. Pro každý trénovací vektor si simulátor vytvoří vektor výstupů, kam zaznamenává hodnoty podle toho, jak dopadla simulace obvodu se vstupy nastavenými podle trénovacího vektoru. Pro každý trénovací vektor porovná výsledné výstupy s polem referenčních výstupů, které jsou načteny už ze vstupního souboru a podle jejich rovnosti aktualizuje fitness.

Jakým způsobem probíhá určování signálů na výstupek pro jeden trénovací vektor popisuje následující algoritmus (zjednodušeno). Algoritmus potřebuje dvě fronty (jednu pro potenciální svorky s nulovým signálem a jednu pro jedničkový signál) a čtyři množiny (jednu pro svorky s nulovým signálem a jednu pro svorky s jedničkovým signálem, dále jednu množinu pro svorky s nerozhodnutým nulovým signálem a druhou pro totéž s jedničkovým signálem):

```
Vyčisti všechny fronty a množiny;
Do množiny 0 vlož „0“ a všechny ostatní svorky, které se
 odkazují na „0“;
Do množiny 1 vlož „1“ a všechny ostatní svorky, které se
 odkazují na „1“;
while(fronty 1 a 0 jsou neprázdné) do begin
      while(fronta 1 je neprázdná) do begin
            Zapamatuj si vršek fronty 1 a odstraň ho;
            If(vršek = 0) then zkrat := true;
            If(vršek není v množině 1) then vlož ho tam;
            If(vršek = G)
                  Pokud ještě nebyla zjištěna otevřenost
                  tranzistoru, přidej případné nerozhodnuté
                  svorky do příslušných front;
            Najdi svorky odkazující se na vršek a nasyp je do
              fronty 1, pokud tam už nejsou;
            Totéž proveď se svorkou odkazovanou vrškem;
            If(vršek = 0 or 1) then přeskoč na konec bloku;
            If(vršek = S or D) then
                  If(tranzistor je otevřen) then
                        Vlož opačnou svorku do fronty 1, pokud
                          tam už není;
      end;
      Zopakuj poslední blok pro frontu 0 (s příslušnými
        úpravami);
end;
if(zkrat) then výstup := -2;
else
      if(výstupní svorka je v mnozine 1 and v mnozine 0) then
            výstup := -2;
      else if(výstupní svorka je v množině 1) then
            výstup := 1;
      else if(výstupní svorka je v množině 0) then
            výstup := 0;
      else
            výstup := -1;
```

Hodnoty výstupů reprezentují buď log. signály („0“ a „1“), neurčitou hodnotu X („-2“) nebo vysokou impedanci („-1“). Bylo rozhodnuto, že vysoká impedance není daleko od správné hodnoty a proto jí je přidělena malá bonusová hodnota, zatímco neurčité hodnotě je přidělena malá penalizace.

Po tom, co je předchozí algoritmus proveden pro všechny trénovací vektory, jsou získané výstupy porovnány s referenčními výstupy získanými ze vstupního souboru. Pravidla pro nastavování fitness hodnoty jsou následující (fitness je nejprve nastavena na nulu):

- Pokud se oba výstupy rovnají, fitness se zvýší o 1,
- pokud je získaný výstup „-2“, pak se fitness sníží o 0,5 a
- pokud je získaný výstup „-1“, pak se fitness zvýší o 0,5.

Maximální hodnota fitness tak odpovídá rovnici 7.2.

7.3.4 Syntaktická omezení

Velmi důležitým aspektem navrženého systému jsou syntaktická omezení, která byla dříve popsána v podkapitole 3.2.3.2 o GP. Nejenže pro algoritmus zužují velikost prohledávaného prostoru jen na taková řešení, která mají smysl a tím ho vlastně urychlují, ale zároveň zjednodušují implementaci dalších částí algoritmu jako jsou např. fitness funkce či mutace.

Je nutno poznamenat, že kromě běžných omezení daných omezeními technologií CMOS funkce pro syntaktická omezení obsahuje i přidané znalosti, jako např. to, že obvod musí vždy obsahovat alespoň po jednom připojení k GND a U_{CC}.

Následující výčet obsahuje všechna omezení, které byla do systému implementována:

- Dodržení radia propojení, pokud je povolen,
- odkazování svorky tranzistoru na sebe samotnou nebo na jinou vlastní svorku,
- připojený výstupu na jinou svorku než S nebo D,
- výstup nesmí být připojen přímo na uzel sousedící s GND nebo s U_{CC},
- tranzistory nMOS nesmějí být připojeny na U_{CC},
- tranzistory pMOS nesmějí být připojeny na GND,
- ochrana před odstraněním posledního připojení na GND a U_{CC} a
- připojení G jen na vstup, S nebo D.

Algoritmus pro dodržení omezení je velmi dlouhý a nepřehledný, proto ho zde neuvádím.

7.3.5 Mutace

Funkce mutace v navrhovaném systému funguje na základě syntaktických omezení a maximálního počtu pokusů o zmutování jednoho genu. Stejně jako při inicializaci je možné, že gen vybraný pro mutaci není kvůli syntaktickým omezením možné zmutovat a maximální počet pokusů zabraňuje nekonečnému cyklení. Hodnota maxima je napevno nastavena na 30.

Samozřejmostí je, že se mutovaný gen nastavuje na náhodnou hodnotu generováním náhodného čísla užitím bitů vyššího řádu. Pokud mutace nesplňuje omezení, je navrácen gen na původní hodnotu a pokus je opakován do té doby, kdy je zmutováno tolik genů, kolik odpovídá hodnotě počtu mutací na jednoho jedince, což je parametr EA, který se zadává do vstupního souboru.

8 Popis a výsledky experimentů

Tato kapitola obsahuje výsledky experimentů spouštěných na evolučním vývojovém systému rozdělené podle obvodů, které byly cílem cílem návrhu.

Jednodušší vyevolvované obvody byly ověřovány podle literatury [27] či ručně, složitější experimenty v systému ngspice, tam kde je to uvedeno.

Výběr experimentů a množství pokusů byly ovlivněny limitovaným výpočetním výkonem a časem. Snahou bylo poskytnout alespoň vzorek rozsahu obvodů, které bylo či nebylo možné systémem navrhnout. Množství z těchto pokusů běželo ještě nedlouho před odevzdáním této práce.

Každý experiment obsahuje výčet parametrů EA, se kterými byl spuštěn, verzi programu (poslední verze je nejdůležitější), ukázky vyvinutých chromozomů či schémat vzniklých interpretací vyvinutých chromozomů. Každý experiment je v samostatném adresáři na CD spolu s dosaženými výsledky, zdrojovým kódem použité verze programu, spouštěcím skriptem pro SGE a vstupním souborem.

Položka „Max. generací“ v tabulkách znamená, na kolik generací byly omezeny jednotlivé experimenty. Položka „Nedokončených běhů“ udává, kolik procent běhů experimentu bylo nuceně přerušeno před dokončením max. počtu generací. To je způsobeno tím, že spouštěné experimenty nejsou omezeny jen počtem generací, ale také časem běhu procesu v systému SGE. Pokud je tento čas překročen, jsou procesy násilně ukončeny. Tyto časy byly nastavovány různě s předpokládaným časem běhu experimentu, nicméně vzhledem k charakteru distribuce úkolů v SGE bylo obtížné udělat správný odhad. Sloupeček „Min. generace“ udává, v jaké nejnižší generaci se podařilo dosáhnout úspěšného řešení. Údaj není uveden u neúspěšných experimentů či u těch, kdy nebyla hodnota min. generace zaznamenána.

U experimentů byly nastavovány i další méně významné parametry, které lze dohledat v přiložených zdrojových souborech programu.

8.1 Základní logická hradla

Základní logická hradla jsou samozřejmým prvním krokem při návrhu číslicových elektronických obvodů, protože představují nejjednodušší problémy. Následující tabulky 8.1 a 8.2 podávají přehled funkcí jednotlivých hradel se dvěma nebo třemi vstupy.

Tabulka 8.1: NOT a hradla se dvěma vstupy

A	B	NOT	NAND	NOR	AND	OR	XOR
0	0	1	1	1	0	0	0
0	1	1	1	0	0	1	1
1	0	0	1	0	0	1	1
1	1	0	0	0	1	1	0

Tabulka 8.2: Logická hradla se třemi vstupy

A	B	C	NAND	NOR	AND	OR
0	0	0	1	1	0	0
0	0	1	1	0	0	1
0	1	0	1	0	0	1
0	1	1	1	0	0	1

1	0	0	1	0	0	1
1	0	1	1	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	1	1

8.1.1 Hradlo NOT

Experiment s evolucí hradla NOT byl proveden jen jeden, protože zapojení je triviální a není na něm co zkoušet. Pokud nebyly na začátku vygenerovány dva stejné tranzistory, experiment se většinou povedl. Nepodařené experimenty se dvěma stejnými tranzistory jsou dány konfliktem toho, že při inicializaci se algoritmus pokouší mutovat tak, aby obvod obsahoval propojení na obě svorky U_{CC} a GND a tím, že na jeden druh tranzistoru nejdou obě nutné svorky připojit.

Tabulka 8.3: Výsledky evoluce hradla NOT

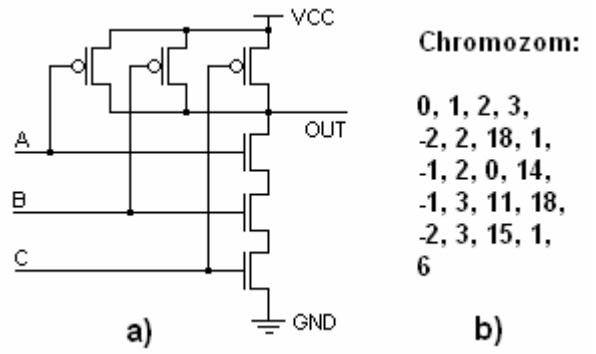
Označení exp.	Verze systému	Vstupů	Tranzistorů	Max. generací	Mutací na jedince	Populace	Běhů	Úspěšných běhů [%]	Nedokon-čených běhů [%]	Min. generace
NOT	R2Xb	1	1x2	10	2	10	100	17,0	18,0	0

8.1.2 Hradla NAND

Hradla NAND se dvěma či třemi vstupy se ukázala jako snadno řešitelné problémy, když většina běhů EA vrátila úspěšný výsledek. U 4-vstupových hradel nebyl žádný experiment dokončen v nastaveném čase (viz omezení času běhu procesů v SGE – v tomto případě cca 12 hodin). Obrázek 8.1 obsahuje ukázku nejlepšího vyvinutého 3-vstupového hradla s odpovídajícím chromozomem. Nejlepší 2-vstupové hradlo koresponduje s obrázkem 7.1 a). Výsledky byly ověřeny porovnáním s literaturou.

Tabulka 8.4: Výsledky evoluce hradel NAND

Označení exp.	Verze systému	Vstupů	Tranzistorů	Max. generací	Mutací na jedince	Populace	Běhů	Úspěšných běhů [%]	Nedokon-čených běhů [%]	Min. generace
2a1	3	2	3x3	5 mil.	2	8	6	100,0	0,0	
2b1	3	2	2x2	5 mil.	2	8	25	24,0	24,0	
2c1-2	R	2	2x2	5 mil.	2	8	29	0,0	34,5	
2d1-2	R2	2	2x2	8 mil.	2	8	33	51,5	33,3	20
2e1	R2	2	2x2	8 mil.	2	8	20	65,0	20,0	28
3b1	P	3	3x2	7 mil.	2	8	20	80,0	5,0	406
3c1	R2	3	3x2	7 mil.	2	8	20	95,0	0,0	356
4a1	P	4	4x3	7 mil.	2	8	20	0,0	95,0	



Obrázek 8.1: Nejlepší vyvinuté hradlo NAND: a) překreslený fenotyp a b) odpovídající chromozom

8.1.3 Hradla NOR

Výsledky experimentů s hradly NOR se svou úspěšností podobají experimentům s NAND. Vyvinuté obvody opět odpovídají vzorovým obvodům v literatuře. Nejlepší vyvinuté 2-vstupové hradlo odpovídá obrázku 5.3 c), 3-vstupové má stejnou strukturu, jen s příslušnými dvěma tranzistory navíc.

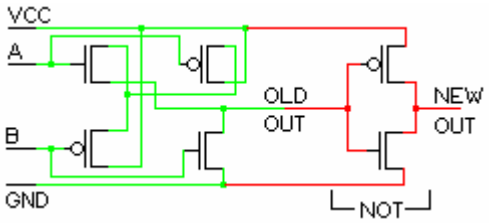
Tabulka 8.5: Výsledky evoluce hradel NOR

Označení exp.	Verze systému	Vstupů	Tranzistorů	Max. generací	Mutací na jedince	Populace	Běhů	Úspěšných běhů [%]	Nedokončených běhů [%]	Min. generace
2a1	P	2	2x2	5 mil.	2	8	10	40,0	0,0	
2b1	R2	2	2x2	5 mil.	2	8	20	65,0	10,0	9
3a1	3	3	3x3	7 mil.	2	8	20	80,0	5,0	29
3b1	P	3	3x2	7 mil.	2	8	20	80,0	0,0	65
3c1	R2	3	3x2	7 mil.	2	8	20	95,0	0,0	945

8.1.4 Hradla AND

Hradla AND se ukázala jako problematická. Tyto obvody jsou svým propojením složitější a v množství spuštěných experimentů se nepodařilo objevit žádný plně funkční obvod, nejlepší měly fitness hodnotu 3,5.

Později byla upravena verze systému, ve které lze explicitně dodat invertor na výstup obvodu, takže se úspěšnost nalezených obvodů vyrovnala hledání NAND a NOR. Postup ilustruje obrázek 8.2 na jednom z výsledných hradel AND včetně rozmístění v mřížce.



Obrázek 8.2: Vyvinuté hradlo AND - zelená označuje spojení a tranzistory vyevolvované systémem (funkce NAND), červená označuje explicitně dodané tranzistory tvořící funkci NOT

Tabulka 8.6: Výsledky evoluce hradel AND

Označení exp.	Verze systému	Vstupů	Tranzistorů	Max. generací	Mutací na jedince	Populace	Běhů	Úspěšných běhů [%]	Nedokon-čených běhů [%]	Min. generace
2f2	R2	2	2x3	7 mil.	2	8	20	0,0	10,0	
2g2	R2	2	2x3	20 mil.	2	8	20	0,0	30,0	
2h1	R2	2	2x3	1 mil.	2	8	20	0,0	10,0	
2i1	R2b	2	2x3	1 mil.	2	8	20	0,0	5,0	
2j1	R2b	2	2x3	1 mil.	2	8	20	0,0	5,0	
2k1	R2b	2	2x5	1 mil.	2	8	30	0,0	0,0	
2l1	5	2	2x2	2 mil.	2	8	30	80,0	0,0	12

8.1.5 Hradla OR

Stejně jako hradla AND jsou hradla OR problematická. Relativně velké množství spuštěných experimentů (některé trvaly celý den) vrátilo jediný obvod s maximální fitness hodnotou, což je spíše dílem náhody. Stejně jako u AND byly experimenty spouštěny s různými verzemi programu tak, jak byly průběžně opravovány chyby a také s množstvím variant nastavení parametrů algoritmu, nicméně výsledek to nepřineslo. Jen malé množství experimentů skončilo ve stanoveném čase.

I zde bylo později použito upravené verze systému s explicitním invertorem a objeveno množství správných obvodů. Objevené obvody odpovídají hradlu NOR s připojeným invertorem.

Tabulka 8.7: Výsledek evoluce hradel OR

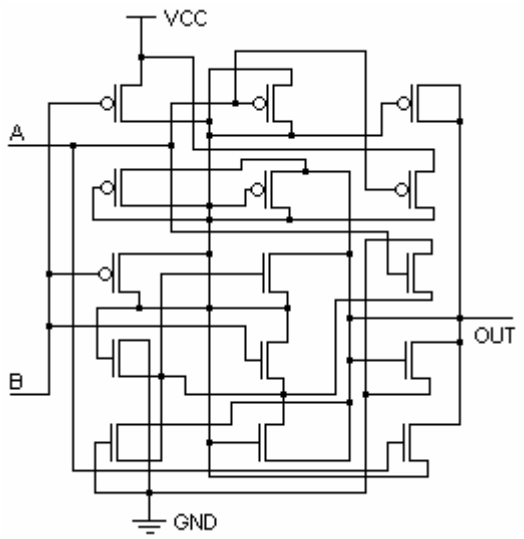
Označení exp.	Verze systému	Vstupů	Tranzistorů	Max. generací	Mutací na jedince	Populace	Běhů	Úspěšných běhů [%]	Nedokon-čených běhů [%]	Min. generace
2b1	R2b	2	3x4	20 mil.	2	8	19	0,0	100,0	
2d1	R2b	2	2x2	3 mil.	2	150	20	0,0	90,0	
2f1	R2X	2	3x4	20 mil.	2	10	20	0,0	0,0	
2g1	R2Xb	2	3x4	20 mil.	2	10	40	2,5	97,5	923207
2h1	5	2	2x2	2 mil.	2	8	30	70	10,0	12

8.1.6 Hradla XOR

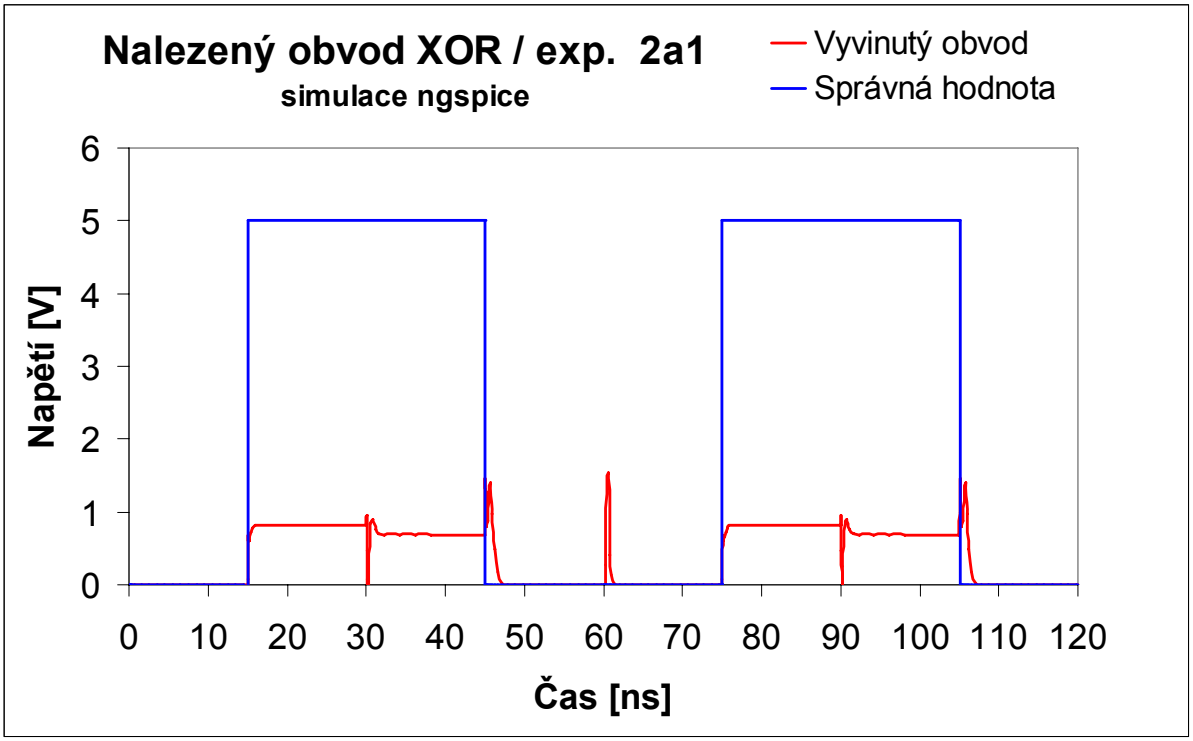
Během experimentování se podařilo objevit menší počet hradel XOR, které se ale při ověřování ngspice ukázala jako nevyhovující. Tvar signálu se značně podobal hledané funkci, avšak napěťová úroveň byla příliš nízká, oproti vzoru (model s 5V logikou). Vyevolvovaný nekvalitní obvod XOR je na obrázku 8.3.Výsledek ověření tohoto obvodu je na obrázku 8.4.

Tabulka 8.8: Výsledky evoluce hradel XOR

Označení exp.	Verze systému	Vstupů	Tranzistorů	Max. generací	Mutací na jedince	Populace	Běhů	Úspěšných běhů [%]	Nedokon-čených běhů [%]	Min. generace
2a1	R2b	2	5x3	5 mil.	1	50	10	40,0	60,0	56945



Obrázek 8.3: Nekvalitní nalezený obvod XOR



Obrázek 8.4: Nekvalitní nalezený obvod XOR. Vstupy simulovány dvěma obdélníkovými pulsními signály s periodami 30 a 60ns. (viz „glitche“ na obrázku při každé změně prvního signálu)

8.2 Složitější obvody

8.2.1 Kombinovaná hradla NAND-NOR

Se značným úspěchem byla navržena i kombinace hradel NAND a NOR na ukázku toho, jak systém pracuje s více výstupy. Fitness se zde počítá dle rovnice 7.2 z předchozí kapitoly.

Tabulka 8.9: Výsledky evoluce kombinovaných hradel NAND-NOR

Označení exp.	Verze systému	Vstupů	Tranzistorů	Max. generací	Mutací na jedince	Populace	Běhů	Úspěšných běhů [%]	Nedokon-čených běhů [%]	Min. generace
2a1	P	2	3x3	5 mil.	2	8	20	95,0	0,0	208
2b1	R2Xb	2	3x2	5 mil.	2	15	20	80,0	0,0	3259

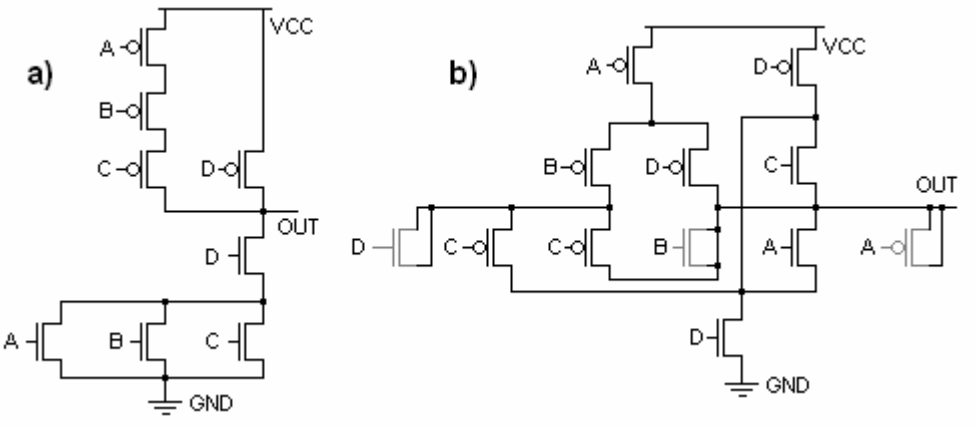
8.2.2 Složený AND-OR-Invertor

AND-OR-Invertor je 4-vstupový obvod, který počítá funkci $Y = \overline{(A + B + C) \cdot D}$. Kvůli omezenému času bylo provedeno zpočátku jen 17 experimentů s jednou sadou parametrů (4a1), přičemž jen jeden běh byl úspěšný. Později se ukázalo, že obvod má po ověření v ngspice špatné vlastnosti.

Při dalších pokusech s verzí systému povolující připojení hradel gate pouze na vstupy se podařilo najít správné obvody. Při ověření v ngspice se ukázalo, že se obvod zvolený pro ověření chová chybně. Protože byly svorky gate tranzistorů propojeny jen na vstupy, pokusil jsem se obvod ještě ověřit ručně, což se podařilo. Rozdíl v obou ověřovacích pokusech může být způsoben nevhodně zvoleným modelem v ngspice nebo tím, že ngspice bere v úvahu nejrůznější elektrické vlastnosti tranzistorů, zatímco ruční ověřování bylo provedeno na logické úrovni. Takovou elektrickou vlastností může být např. špatné vedení jistých logických signálů jistými tranzistory, jak je uvedeno v podkapitole 5.2. Nepředvídání některých elektrických vlastností může být nevýhodou logických simulátorů obvodů. Obrázek 8.5 (a) zobrazuje konvenční řešení obvodu [27], zatímco (b) zobrazuje evolucí nalezený obvod. Výsledek ověřování je možné nalézt v přílohách na CD.

Tabulka 8.10: Evoluce obvodu AND-OR-INVERT

Označení exp.	Verze systému	Vstupů	Tranzistorů	Max. generací	Mutací na jedince	Populace	Běhů	Úspěšných běhů [%]	Nedokon-čených běhů [%]	Min. generace
4a1	R2b	4	4x3	10 mil.	2	8	17	5,9	94,1	506436
4c1	4	4	4x3	5 mil.	1	20	30	76,7	23,3	1637



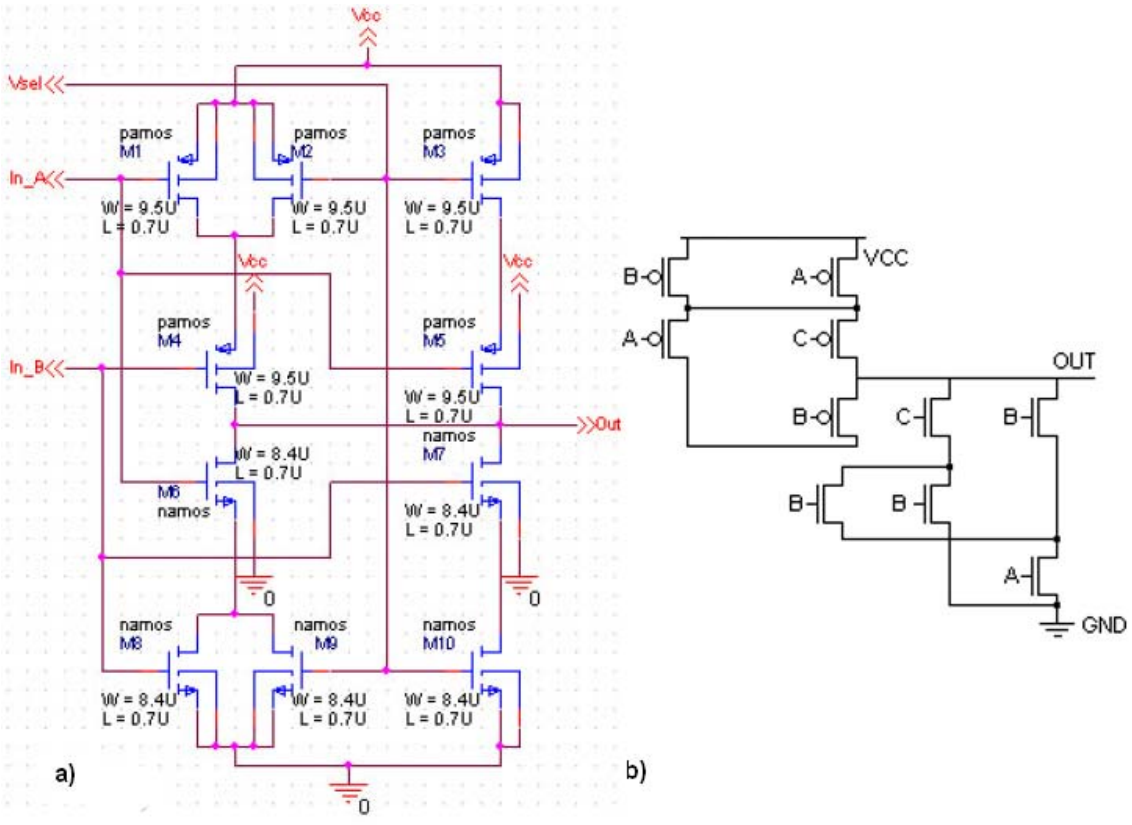
Obrázek 8.5: a) konvenční AND-OR-Invert [27] a b) evolucí navržený. Šedě jsou vyznačeny tranzistory, které pravděpodobně nemají v obvodu žádnou funkci

8.2.3 Polymorfní obvod NAND/NOR řízený ext. vstupem

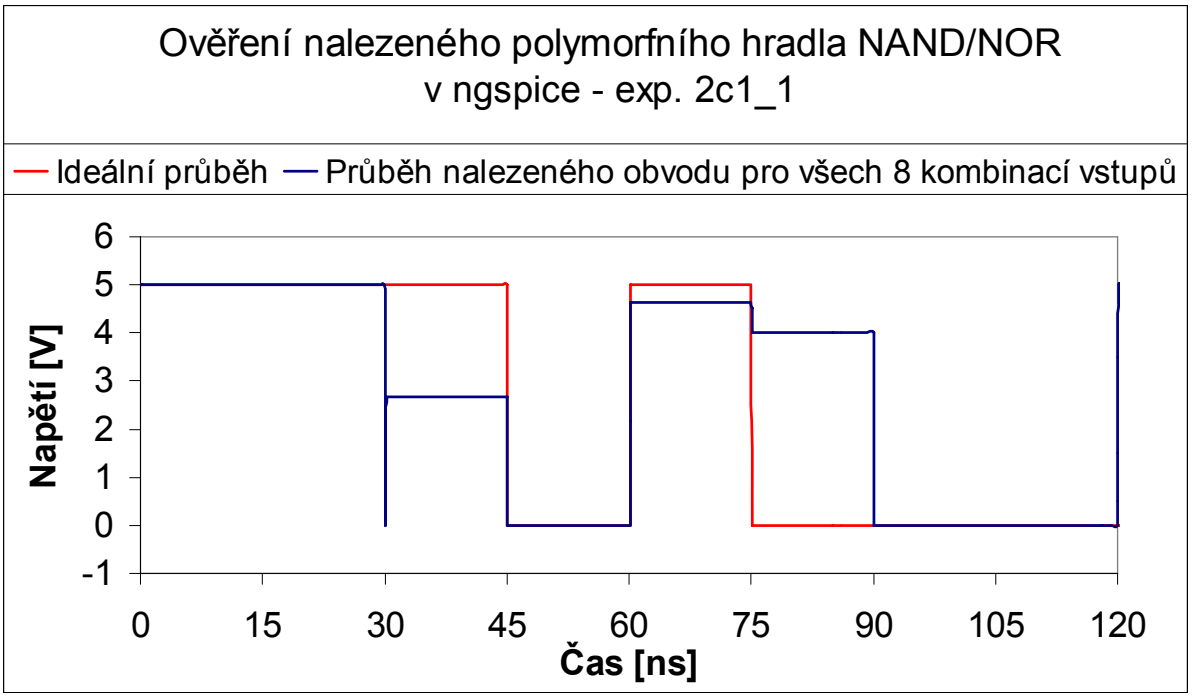
Polymorfní obvod NAND/NOR řízený externím vstupem funguje tak, že v případě log. nuly na řídícím vstupu se chová jako NAND, v případě log. jedničky se chová jako NOR. Obvody byly vyvíjeny pro různé velikosti mřížek tranzistorů a obvod s deseti tranzistory byl ověřen ručně a v ngspice, protože je možné ho srovnat se stejně velkým obvodem navrženým konvenčním způsobem [23] tak, jak je to na obrázku 8.6. Bohužel se stejně jako u AND-OR-Invertoru ukázalo, že ruční ověřování na logické úrovni se liší od ověřování v ngspice, pravděpodobně ze stejných důvodů jako v předchozím případě. Výsledek ngspice se však výrazně liší jen v jedné kombinaci vstupů (101). Výstup obvodu v ngspice je možné porovnat se správným průběhem na obrázku 8.7.

Tabulka 8.11: Výsledky evoluce polymorfních hradel NAND/NOR

Označení exp.	Verze systému	Vstupů	Tranzistorů	Max. generací	Mutací na jedince	Populace	Běhů	Úspěšných běhů [%]	Nedokon-čených běhů [%]	Min. generace
2a1	4	3	5x4	5 mil.	1	20	20	100,0	0,0	537
2b1	4	3	5x3	5 mil.	1	20	20	100,0	0,0	258
2c1	4	3	5x2	5 mil.	1	20	20	100,0	0,0	492
2d1	4	3	4x2	5 mil.	1	20	20	20,0	0,0	262



Obrázek 8.6: Porovnání a) konvečně navrženého polymorfního obvodu [23] a b) obvodu vyvinutého evolucí



Obrázek 8.7: Výsledky ověřování vyvinutého polymorfního hradla NAND/NOR v ngspice

9 Zhodnocení výsledků

9.1 Limitace systému

Z provedených experimentů se zdá, že navržený evoluční systém má problémy navrhnout určitou skupinu obvodů, mezi něž patří např. základní logická hradla AND a OR. Tyto skupiny obvodů lze označit za tzv. klamné problémy (podkapitola 3.1.1), jejichž řešení je obtížnější nalézt pomocí EA. Částečnou vinu na tomto neúspěchu by rovněž mohlo mít ohodnocování fitness jen na základě logických signálů. Granularita takové fitness je malá a v případě přibližování se optimu může dojít k tomu, že nelze dostatečně rozlišit kvalitu obvodů, což snižuje selekční tlak a vývoj se tak téměř zastavuje.

Dalším problémem je, že některé jiné obvody, jako např. XOR, se pomocí systému podařilo vyvinout, ale jejich ověřování v ngspice dopadlo špatně. Důvodem, který se podařilo zjistit až v pozdější fázi testování, je nejspíše nedokonalost v simulátoru při určování fitness hodnoty. Některé z navržených obvodů, které obdržely v simulátoru maximální hodnotu fitness se ukázaly jako nesprávné při ručním ověřování nebo při simulaci v programu ngspice, tak jak je to ilustrováno na obrázku 8.4 obvodu XOR. Podskupinou s tímto problémem jsou obvody se správnou logickou funkcí (G propojeny jen na vstupy), které vracejí špatné výsledky v simulátoru, pričemž vysvětlení lze nalézt v podkapitole 8.2.2.

Nedokonalost v simulaci spočívá v problematickém určování otevřenosti tranzistorů nepřipojených přímo na vstupy obvodu. Při propagaci signálů logické jedničky a logické nuly dochází k jejich míšení a záleží na tom, který ze signálů dorazí k svorce G tranzistoru dříve. Tomu by se pravděpodobně dalo zabránit implementací modelu časování, případně kontrolou propagace signálů na úrovni front. Bohužel, právě proto, že byla chyba objevena v pozdní fázi testování, nezbyl čas na její opravu.

Dřívější verze programu (do verze 3) však spolehlivě pracují na evolučním návrhu obvodů, ve kterých jsou svorky G připojeny jen na vstupy obvodu. Zvýšení funkčnosti těchto předchozích verzí systému by mohlo být dosaženo možností explicitně přidávat do obvodu hradla NOT.

Zmíněný problém byl nakonec vyřešen v jedné z verzí sytému (4) tak, že je možné ve vstupních parametrech specifikovat explicitní přidání invertoru na výstup obvodu, což by se dalo chápat jako vložení znalostí do algoritmu. Takto se podařilo velmi rychle objevit i obvody AND a OR. Nakonec byla přidána možnost vybrat, které z výstupů budou obsahovat invertor, takže jsou možné i kombinace (v5).

Omezená použitelnost verzí systému s možností připojení svorek G dovnitř obvodu by se dala dosáhnou propojením se simulátorem SPICE, který by případné obvody s maximální dosaženou fitness dodatečně ověřoval.

Dodání druhé fáze evoluce s použitím SPICE systému za účelem optimalizace elektrických vlastností obvodu bylo zpočátku mým sekundárním cílem, nicméně obtíže se zmíněnými problematickými obvody a dlouhé trvání mnoha experimentů mě donutily udělat závěr, že je to časově nerealizovatelné v rámci DP.

Pokud jde o maximální schopnosti systému, odvažuji se je pouze odhadovat podle informací o limitacích CGP [18] na méně než deset vstupů na obvod.

Pokud jde o rychlost hledání složitějších obvodů, tak by se pravděpodobně dala zvýšit dodáním dalších znalostí do syntaktických omezení. V systému totiž byla schválně opomenuta některá omezení, jako např. vzájemné spojení některých svorek tranzistoru, aby bylo možné zachovávat neutralitu existencí latentních částí obvodu, které mohou být ve vhodný okamžik správně zmutovány a přinést tak nové možnosti. Otázkou je, co by se poté stalo s problémem uváznutí v lokálních extrémech, když by tyto latentní části obvodu vymizely.

9.2 Porovnání s developmentem v GP

Pokud přehlédneme obtížnost vyvinutí některých skupin problematických obvodů, dokáže implementovaný systém za určitých předpokladů navrhovat jednoduché obvody vyšší rychlostí než developmnent v GP profesora Kozy [10].

Důvodem pro toto tvrzení je skutečnost, že v experimentech se mnohokrát podařilo navrhnou právě např. hradlo NAND, které je navrhováno i v [10], a to během malého počtu generací. Tyto generace měly malý počet jedinců (max. 20) a vynásobíme-li známé počty generací z developmentu GP s jejich obrovskou populací, vyjde větší hodnota než vynásobení několika desítek či stovek generací s malou populací v ES. Získané počty evaluací fitness funkce se pro získání časového údaje navíc musí vynásobit trváním doby evaluace fitness funkce, což může v případě SPICE používaného ve zmíněných experimentech s GP být podstatně delší (dle mých zkušeností desetiny sekundy na obvod), než logický simulátor ve zde popisovaném systému. Simulátor v tomto systému totiž pracuje na základě jednoduchých if-then-else pravidel, porovnávání a případnému procházení polí v paměti, zatímco SPICE počítá množství rovnic, které jsou výpočetně náročnější, rovněž komunikace s programem používajícím SPICE bývá vedena přes textové soubory, což je další výrazné zpomalení.

Je ovšem nutné uvést, že vyšší rychlost navrženého systému je na úkor kvality jeho výsledků, protože algoritmus nebere v úvahu časování a pracuje jen na logické úrovni, zatímco Kozův development v GP se snaží o návrh s dokonalým elektrickým chováním na analogové úrovni.

10 Závěr

10.1 Dosažené cíle práce

V první části této práce byla zpracována studie na téma evolučního návrhu obvodů včetně přehledu evolučních algoritmů, možných úrovní návrhu, metod simulace a ověřování obvodů systémy SPICE, používaných technologií a nejdůležitějších evolučních metod návrhu obvodů.

V druhé části byl představen systém sloužící pro evoluční návrh elektronických číslicových obvodů na úrovni tranzistorů. Pro systém byla navržena vlastní vývojová metoda založená zejména na Kartézském genetickém programování, které je původně určeno pro úroveň hradel. Bylo zdůvodněno, proč byla metoda vybrána a navržena.

Rovněž bylo popsáno, jakým způsobem byl systém realizován. Bylo prokázáno, že implementovaný systém dokáže navrhovat některé číslicové obvody na úrovni tranzistorů. Bohužel bylo také konstatováno, že systém má problémy s návrhem určité skupiny obvodů z důvodů obtížnosti jejich struktury a také kvůli nedokonalosti simulátoru obsaženém v systému. První z těchto problémů byl nakonec částečně vyřešen možností explicitního přidání invertoru na výstup vyvinutých obvodů.

Zdá se, že zapojení hradel jsou velmi unikátní a že neexistuje příliš mnoho variant pro implementaci. Proto se běh EA podobá „hledání jehly v kupce sena“.

10.2 Další vývoj projektu

Protože projekt nedosáhl v úplnosti všech svých cílů, bylo by logickým krokem jeho zdokonalení. Vývojová metoda teoreticky slibuje solidní výsledky při návrhu menších obvodů, takže by se jí dalo využít pro implementaci úplně nového systému nebo zdokonalení stávající metody.

První případ by mohl namísto logického simulátoru obsahovat vyhodnocování SPICE, čímž by se sice snížila rychlost návrhu, ale podstatně by se zvýšila jemnost evaluace a tím i snad kvalita obvodů. Druhou cestou by bylo využití rychlosti logického simulátoru za účelem návrhu obvodů jen na logické úrovni. K tomuto druhému případu by bylo možné dodatečně připojit druhou fázi, kdy by logicky navržené obvody byly optimalizovány vzhledem ke svým elektrickým vlastnostem pomocí simulace ve SPICE.

10.3 Přínosy práce

Pokud by byly následovány kroky zmíněné v předchozím odstavci, dal by se systém využívat v některých projektech probíhajících na FIT, jako je např. návrh polymorfních obvodů. Systém demonstroval, že je schopen vyvíjet polymorfní obvody (alespoň na logické úrovni), což je popsáno v podkapitole 8.2.3. Bohužel jde o systém využívající jen tzv. extrinsickou evoluci, tudíž jeho využití by bylo omezeno limitacemi tohoto postupu zmíněnými v jedné z předchozích kapitol. Na FIT v této

chvíli ale není k dispozici vhodný HW pro provádění intrinsické evoluce na úrovni tranzistorů (FPTA), takže než bude něco takového opatřeno, mohl by být vhodnou suplující pomůckou.

Z teoretického hlediska je přínos této práce v navržení a otestování (ač limitovaném) nové metody na návrh obvodů na úrovni tranzistorů. Další výzkum této metody, kterou bych si dovolil nazvat CGPT (T jako tranzistor), by mohl přinést zajímavé výsledky.

Z osobního hlediska mi tato práce pomohla nabýt nových znalostí v oblastech EA a technologií návrhu HW, které mohu využít pro další studium či budoucí zaměstnání.

Literatura

[1] Bartsch, H.J.: Matematické vzorce, Třetí, revidované vydání, Praha, Mladá fronta, 2000

[2] Bentley, P. (ed.): Evolutionary Design by Computers. San Francisco, Morgan Kaufmann Publishers, 1999, Chapter 1

[3] Brocks, G. a kol.: Working Group 8 - Electronics at the Molecular Level, University of Twente, Netherlands, http://psi-k.dl.ac.uk/data/wg8.html

[4] Darwin, C.: On The Origin of Species by Means of Natural Selection, or The Preservation of Favoured Races in the Struggle for Life, First Edition, 1859, http://www.talkorigins.org/faqs/origin.html

[5] Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing, Springer, 2003, chapter 14

[6] Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley, Reading, MA, 1989

[7] Havránek, V.: Evoluční návrh analogových obvodů [Diplomový projekt], Fakulta informačních technologií, Vysoké učení technické v Brně, 2006

[8] Chvastek, D.: Orcad Capture 9.1 – Simulace analogových obvodů, Fakulta Elektrotechniky a informatiky, Vysoké učení technické v Brně, 2001

[9] Koza, J.R. a kol.: Genetic Programming – On the Programming of Computers by Means of Natural Selection, Sixth printing, Cambridge, Massachusetts, The MIT Press, 1992

[10] Koza, J.R. a kol.: Genetic Programming IV – Routine Human-Competitive Machine Intelligence, First paperback printing, Springer, 2005

[11] Kvasnička, V., Pospíchal, J., Tiňo, P: Evolučné algoritmy, 1. vydanie, Vydavateľstvo STU v Bratislave, 2000

[12] Langeheine, J.: Intrinsic Hardware Evolution on the Transistor Level [Dissertation], Rupertus Carola University of Heidelberg, 2005

[13] Martínek, T.: Návrh mikroprocesoru v FPGA [Přednáška do předmětu Pokročilé číslicové systémy], Fakulta informačních technologií, Vysoké učení technické v Brně, 2006

[14] Miller, J. F., Thomson P.: Cartesian Genetic Programming, Proceedings of the European Conference on Genetic Programming, April 15-16, 2000, p.121-132

[15] Miller, J. F., Downing, K.: Evolution in materio: Looking beyond the G/A silicon box, NASA/DOD Conference on Evolvable Hardware. IEEE Comp. Soc. Press, 2002, pp. 167-176.

[16] ngspice – A SPICE simulator, http://www.geda.seul.org/tools/ngspice/index.html

[17] OrCAD Capture Datasheet, Cadence Design Systems, http://www.cadence.com/datasheets/orcad_capture_cis_ds.pdf

[18] Sekanina, L.: Biologií inspirované počítače [přednášky k předmětu], Fakulta informačních technologií, Vysoké učení technické v Brně, 2006

[19] Sekanina, L.: Evolutionary Design of Gate-Level Polymorphic Digital Circiuts, Lecture Notes in Computer Science, 2005, nr. 3449, s. 185-194

[20] Sekanina, L.: Genetické programování [Přednáška do předmětu Evoluční algoritmy], Fakulta informačních technologií, Vysoké učení technické v Brně, 2006

[21] Sekanina, L.: Image Filter Design with Evolvable Hardware, Lecture Notes in Computer Science, 2002, nr. 2279, s. 255-266

[22] Sekanina, L.: Problém implementace z pohledu evolučního návrhu [Habilitační přednáška], Fakulta informačních technologií, Vysoké učení technické v Brně, 2006

[23] Stareček, L., Sekanina, L., Gajda, Z., Kotásek, Z., Prokop, R., Musil, V.: On Properties and Utilization of Some Polymorphic Gates, Technicka zprava, UPSY, Fakulta informačních technologií, Vysoké učení technické v Brně, 2007, 8. str.

[24] Stoica, A. a kol.: Evolvable Hardware for Autonomous Systems, CEC-2004 Tutorial, Portland, Oregon, Jet Propulsion Laboratory, 2004

[25] Thomas, K.; Williams, C.: SPICE 3 User's Manual, School of Physics web, University of Exeter, 2003, http://newton.ex.ac.uk/teaching/CDHW/Electronics2/userguide/index.html#toc

[26] Wall, M.: GAlib: A C++ Library of Genetic Algorithm Components, version 2.4. Massachusetts Institute of Technology, 1996, http://lancet.mit.edu/ga/

[27] Weste, N.H.E., Harris, D.: CMOS VLSI Design: A Circuits and Systems Perspective, third edition, Pearson/Addison-Wesley, 2005

[28] Wikipedia, the free encyclopedia – CMOS, http://en.wikipedia.org/wiki/CMOS

[29] Wikipedia, the free encyclopedia – MOSFET, http://en.wikipedia.org/wiki/MOSFET

[30] Wikipedia, the free encyclopedia – SPICE, http://en.wikipedia.org/wiki/SPICE

[31] Zebulum, R.S. a kol.: Evolutionary Electronics – Automatic Design of Electronic Circuits and Systems by Genetic Algorithms, CRC Press, 2002

[32] Žaloudek, L.: Koevoluce řadicích sítí a trénovacích vektorů [Bakalářská práce], Fakulta informačních technologií, Vysoké učení technické v Brně, 2005

[33] Žaloudek, L.: Tutorial ke knihovně GAlib pro C++ [Projekt do předmětu Inteligentní systémy], Fakulta informačních technologií, Vysoké učení technické v Brně, 2005

Seznam příloh