VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

POLYMORFNÍ SAMOČINNĚ TESTOVATELNÉ OBVODY

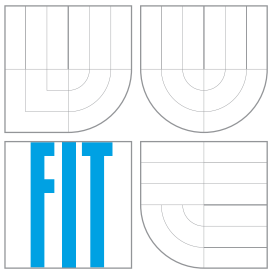
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN MAZUCH

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

POLYMORFNÍ SAMOČINNĚ TESTOVATELNÉ OBVODY

SELF-CHECKING POLYMORPHIC CIRCUITS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN MAZUCH

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. LUKÁŠ SEKANINA, Ph.D.

BRNO 2008

Abstrakt

Táto diplomová práca sa zaoberá problematikou návrhu samočinne testovateľných polymorfných obvodov. Pojednáva o konvenčnom návrhu spoľahlivých a samočinne kontrolovaných obvodov, predstavujúc základné techniky a metódy ich návrhu a konštrukcie. Ďalej vysvetľuje metódu kartézskeho genetického programovania pre návrh kombinačných logických obvodov, ktorá je použitá v praktickej časti práce. Takisto uvádza koncepciu polymorfných hradiel a obvodov a ich praktické využitie. Predstavené sú aj existujúce samočinne kontrolovateľné polymorfné obvody a nad konkrétnymi príkladmi je vykonaná analýza ich činnosti. Ďalej je uvedený návrh realizácie návrhového systému pre samočinne kontrolovateľné polymorfné obvody. Podľa uvedeného návrhu bola vytvorená aplikácia pre návrh obvodov a tiež aj aplikácia umožňujúca simulovanie a analýzu navrhnutých obvodov. Nad vytvoreným systémom bola vykonaná rada experimentov a získaných niekoľko zaujímavých riešení. V závere sa nachádza zhrnutie dosiahnutých výsledkov a prínos práce.

Klíčová slova

kartézske genetické programovanie, CGP, kontrolór, dvoj-drátový kontrolór, polymorfné hradlo, polymorfný obvod, samočinne kontrolovaný polymorfný obvod, komplexná váhovaná fitness funkcia

Abstract

This Master's thesis deals with question of the development of self-checking polymorphic circuits. It deals with a traditional way of creating reliable and self-checking circuits, presenting basic principles and methods. Also a method of Cartesian Genetic Programming for development of combinational circuits is explained. This thesis describes concepts of polymorphic gates and circuits and their benefits in practical use. Some existing self-checking polymorphic circuits are presented and their self-checking capabilities are analyzed. A proposal of realization of a design system for self-checking polymorphic circuits is given. A design system has been built based on presented specification and an application allowing simulations and analysis of system-proposed solutions has been created. Variety of experiments have been performed at created system and several interesting solutions have been acquired. At the end, conclusion is given and benefits of MSc. project are discussed.

Keywords

Cartesian Genetic Programming, CGP, self-checking, checker, two-rail checker, polymorphic gate, polymorphic circuit, self-checking polymorphic circuit, complex weighted fitness function

Citace

Martin Mazuch: Polymorfní samočinně testovatelné
obvody, diplomová práce, Brno, FIT VUT v Brně, 2008

Polymorfní samočinně testovatelné obvody

Prehlásenie

Prehlasujem, že som túto prácu vypracoval samostatne pod vedením doc. Ing. Lukáša Sekaniny, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Martin Mazuch
19. mája 2008

Poďakovanie

Touto cestou by som chcel poďakovať doc. Ing. Lukášovi Sekaninovi Ph.D. za odborné vedenie práce a cenné poskytnuté rady a Ing. Zdeňkovi Vašíčkovi za poskytnutie kostry implementácie CGP pre návrh kombinačných logických obvodov.

Obsah

Kapitola 1

Úvod

Táto diplomová práca spadá do oblasti návrhu číslicových obvodov, ktoré využívajú nekonvenčné hradlá, tzv. polymorfné hradlá [16]. Cieľom tejto práce bolo vytvoriť aplikáciu umožňujúcu navrhovať polymorfné obvody s rovnakou funkciou v obidvoch módou, ktoré budú dosahovať vysokej úrovne v schopnostiach samočinnej kontroly. Pre návrh boli vybrané evolučné techniky, konkrétne kartézske genetické programovanie s viacúrovňovým hodnotením evolvovaného obvodu podľa kritérií funkčnosti, miery samočinnej kontroly, či ceny. Bol skúmaný jednak vplyv nastavení parametrov evolúcie, genetických operátorov, a tiež spôsob ohodnocovania kandidátnych riešení pomocou fitness funkcie a ich dopad na úspešnosť evolúcie. Pre overenie získaných výsledkov bola vytvorená aplikácia, ktorá umožňuje simulovať a komplexne hodnotiť navrhnuté obvody z pohľadu schopností samočinného testovania.

V druhej kapitole sú zhrnuté základné princípy spoľahlivosti systémov. Taktiež sa tu nachádza aj popis princípu samočinného testovania. Kapitola uvádza konvenčné spôsoby návrhu samočinne kontrolovaných číslicových systémov.

Tretia kapitola detailne popisuje metódu kartézskeho genetického programovania pre návrh číslicových obvodov. Jedná sa o variantu genetického programovania vhodnú predovšetkým na návrh kombinačnej logiky.

V štvrtej kapitole je popísaná problematika návrhu a použitia polymorfných obvodov. Špeciálne sa venuje praktickému použitiu takýchto obvodov v prípadoch, kde je vyžadovaná vysoká spoľahlivosť.

V ďalšej kapitole je detailne rozobratých niekoľko polymorfných obvodov so schopnosťou samočinného testovania. Analyzované obvody sú alternatívnou k obvodom, ktoré je schopné aplikácia vytvorená vytvorená v tejto diplomovej práci sama navrhovať metódou integrovania komplexnej fitness funkcie do návrhu polymorfných obvodov s jednou funkciou.

V šiestej kapitole je prezentovaná myšlienka návrhu samočinne kontrolovateľných polymorfných obvodov, ktorá bola aplikovaná v programe pre návrh polymorfných samočinne kontrolovateľných obvodov. V kapitole možno nájsť podrobnejší popis funkcie aplikácie navrhujúcej riešenia a aplikácie simulujúcej navrhnuté obvody. Nachádzajú sa tu aj špecifikácie formátov vstupných dát.

V siedmej kapitole sú popísané vykonané testy a experimentálne získané výsledky. Testy boli postavené tak, aby sa pokúsili znovuobjaviť a prípadne aj vylepšiť najlepšie známe polymorfné samočinne testovateľné obvody. Výsledky sú tvorené predovšetkým porovnávaním navrhnutých riešení s kvalitnými existujúcimi alternatívami.

V predposlednej kapitole je uvedený popis požiadaviek na systémové prostriedky a prostredie, potrebných pre spustenie vytvorených aplikácií. Možno tu nájsť postup ako spustiť

a vyskúšať vytvorené aplikácie. Taktiež sa tu nachádza stručný popis schopností a limitov vytvorených aplikácií.

V závere je zhodnotený prínos diplomovej práce a zhrnuté dosiahnuté výsledky.

Kapitola 2

Princípy návrhu samočinne testovateľných obvodov

S problémom spoľahlivosti najrôznejších zariadení sa každodenne dostáva do styku takmer každý z nás. Konštruktéri a výrobcovia sú nútení sa zaoberať spoľahlivosťou veľmi dôkladne, pretože na ich úsilí závisí, či bude daný produkt spoľahlivý. Ak chceme mať možnosť hodnotiť a porovnávať spoľahlivosť systémov, musíme definovať ukazovatele spoľahlivosti, pretože spoľahlivosť ako taká nie je kvantifikovateľná. Teória spoľahlivosti je veľmi rozsiahla oblasť, preto budú zmienené iba základné prípady a princípy.

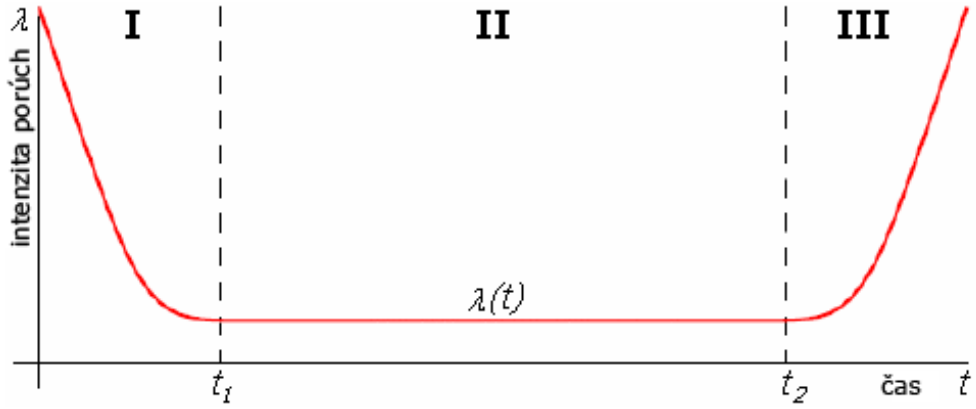
Podľa ČSN 01 0102 bolo názvoslovie spoľahlivosti v technike definované ako obecná vlastnosť objektu spočívajúca v schopnosti plniť funkcie pri zachovaní hodnôt stanovených prevádzkových ukazovateľov v danom rozmedzí a čase podľa stanovených technických podmienok.

Technickými podmienkami sa rozumie súhrn špecifikácií technických vlastností predpísaných pre požadovanú objektu zariadenia, ďalej spôsob činnosti, údržby a opravy. V oblasti číslicových systémov je pojem objekt chápaný ako súčiastka, obvod, funkčný blok, systém a podobne. Pri študovaní spoľahlivosti je nutné rozlišovať dva typy objektov – obnovované a neobnovované. Obnova je pritom chápaná ako prechod z poruchového do bezporuchového stavu činnosťou označovanou ako oprava. V literatúre [1, 4] sú uvedené rôzne ukazovatele spoľahlivosti.

Medzi najpoužívanejšie patria pravdepodobnosť bezporuchového stavu, stredná doba bezporuchovej činnosti, intenzita porúch, stredná doba od jedného výskytu poruchy do ďalšieho výskytu poruchy. Každý ukazovateľ spoľahlivosti mení svoje hodnoty v závislosti na množstve externých faktorov. Mnoho týchto vplyvov bolo empiricky zistených, avšak ich analytické vyjadrenie nie je jednoduché. Na obrázku 2.1 je uvedená takzvaná vaňová krivka, ktorá udáva závislosť intenzity porúch na čase.

Interval $< t1, t2 >$ označujeme ako obdobie normálnej činnosti a platí pre neho, že intenzita porúch má približne konštantnú hodnotu. Intenzita porúch v prvej fáze (Burn-in Period) býva vysoká z dôvodu zavedenia systému do prevádzky. Prvotné poruchy môžu byť spôsobené napríklad nedostatočnou výstupnou kontrolou [5]. V treťom období (Wear-out Period) vznikajú poruchy najmä z dôvodu obmedzenej životnosti jednotlivých súčiastok a komponent systému. Každý systém časom starne a raz bude musieť byť vyradený alebo nahradený.

V poslednom období dochádza k výraznému zvyšovaniu zložitosti digitálnych systémov. Aj keď sa výrobcovia snažia zaistiť, aby boli ich produkty spoľahlivé, v prevažnej väčšine



Obrázek 2.1: Priebeh intenzity porúch λ v závislosti na čase t

prípadov je systém bez akejkoľvek chyby vyskytujúcej sa v danom časovom období prakticky nemožné zostrojiť. S neustále rastúcou hustotou integrácie sú číslicové obvody viac náchylné na hazardy vzniknuté pôsobením rôznych externých faktorov (teplotou, radiáciou, atď.). Z toho pramení, že spoľahlivosť je jedným z kľúčových aspektov množstva navrhovaných systémov.

2.1 Návrh spoľahlivých systémov

Podľa [4] existujú štyri hlavné fázy návrhu systému odolného proti poruchám:

1.) stanovenie cieľov,

2.) voľba metód detekcie porúch,

3.) návrh algoritmov zotavenia po poruche a

4.) vyhodnotenie odolnosti proti poruchám.

V prvom rade je potrebné vytvoriť jasne formulované zadanie projektu. Vzhľadom k tomu, že žiadny systém nemôže byť odolný proti všetkým poruchám, musia byť presne špecifikované všetky situácie, v ktorých si má systém zachovať schopnosť funkcie podľa požiadaviek. Zoznam porúch by mal byť roztriedený podľa pravdepodobnosti ich výskytu. Pre zaistenie odolnosti proti poruchám má kľúčový význam detekcia porúch. Zisťovanie výskytu poruchy býva realizované niekoľkými spôsobmi. Typicky je pri štarte systému zahájená spúšťacia diagnostika (z angl. Power-On-Self-Test, skrátene POST). Pri *periodickej diagnostike* sa zisťovanie chýb vykonáva v čase prestávok medzi aplikačnými programami. Takýmto testom je zaručené odhalenie chyby v okamihu testu, ale nie je zaručené, že daný stav sa nezmení do nasledujúceho testu. Preto je dobré voliť periodicitu testov tak, aby bola pravdepodobnosť vzniku poruchy medzi dvomi testami malá. *Priebežná diagnostika* (angl. online testing) je veľmi obľúbeným prostriedkom kontroly správneho fungovania číslicových systémov. Hlavnou výhodou je jej časová nenáročnosť, pretože pracovné hodnoty sú zároveň chápané aj ako testovacie vektory. Určitou nevýhodou pribežnej diagnostiky je závislosť rozsahu získanej informácie na riešenom probléme. Diagnostika signalizuje len také poruchy, ktoré sa prejavili chybou pri výpočte. Zotavenie po poruche zahŕňa všetky

úkony, ktoré sú potrebné od okamihu zistenia prítomnosti poruchy až po obnovenie funkcie systému. Vyhodnotenie odolnosti proti poruchám je kontrolou do akej miery sa podarilo splniť zadanie. Viac podrobností o jednotlivých krokoch možno nájsť v [4].

2.2 Zvyšovanie spoľahlivosti

Spoľahlivosť je teoreticky možné zvýšiť použitím metód najhoršieho možného prípadu návrhu (worst case design), konštrukciou z vysoko kvalitných komponent, prísnou kontrolou kvality počas zostavovania systému z komponent. Avšak tieto postupy výrazne zvyšujú celkovú cenu, čím sa daný systém predražuje. Možným prístupom dosiahnutia zvýšenej spoľahlivosti je použitím redundancie, čím sa vzniknuté chyby podarí zamaskovať. Týmto postupom vyhovujú aj komponenty v štandardnej kvalite a aj celkové náklady sú v porovnaní s prvým spôsobom nižšie. Nevýhodou je spotreba pomerne veľkej plochy na čipe. Veľmi často sa používa statická *trojmodulová redundancia*.

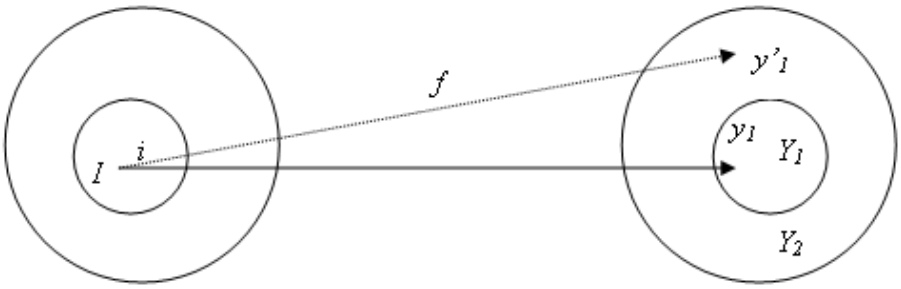
Ako sa veľkosť a zložitosť systémov zvyšuje, je kvôli jednoduchej diagnostike a pohodlnej údržbe a opravám čoraz viac žiadaná schopnosť samočinného kontrolovania systému. Táto schopnosť, označovaná aj ako *samočinná kontrolovateľnosť*, angl. *self-checking* je definovaná ako automatické zisťovanie výskytu chýb vo vnútornej logike (čipy, dosky, správne zostavenie systému), bez potreby externého prikladania testovacích stimulov. Samočinne kontrolovateľné systémy dovoľujú priebežnú detekciu chýb, ktorá môže byť vykonávaná počas normálnej činnosti obvodu. Tento prístup je alternatívou k zvýšeniu spoľahlivosti bez použitia exaktnej kópie časti systému.

Všeobecne môžu byť samočinne kontrolovateľné obvody navrhované iba pre určenú množinu chýb. Množina typicky zahŕňa chyby uviaznutia v trvalej logickej úrovni (single stuck-at faults) a jednosmerné viacnásobné chyby (unidirectional multiple faults). Prvá skupina predpokladá také poškodenie logiky obvodu, pri ktorom úroveň výstupného signálu z logiky je stále na rovnakej úrovni. Konkrétne sa môže jednať buď o uviaznutie v „trvalej logickej 0“ (stuck-at-0) alebo v „trvalej logickej 1“ (stuck-at-1).

Z hľadiska kvality diagnostiky je účelné, aby bol obvod schopný sám rozpoznať a signalizovať všetky poruchy, ktoré by mohli ovplyvniť jeho funkciu [4]. Aby bol daný obvod označený ako úplne samočinne kontrolovaný, musí spĺňať obidve nasledujúce podmienky:

a) Musí byť samočinne testovateľný (angl. Self testing, viď obrázok 2.2)

b) Musí byť bezpečný proti poruchám (angl. Fault-secure, viď obrázok 2.3)



Obrázek 2.2: Znázornenie podmienky samočinného testovania

Zamerajme sa na obrázok 2.2.

Nech
F je množina chýb,
I je množina kódových vstupných slov,
Y_1 je množina výstupných kódových slov[1],
Y_2 je množina nekódových (výstupných) slov,
množina Z je celkový výstupný priestor, pričom $Z = Y_1 \cup Y_2$

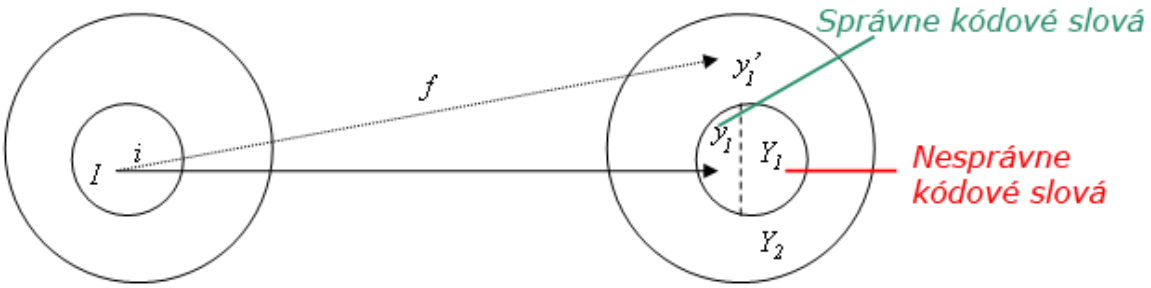
Nech
$i \in I$
y_i je správny výstup obvodu ($y_i \in Y_1$) pre vstup i
a pre rovnaký vstupný vektor pri výskyte chyby $f \in F$ nech výstup obvodu je $y_i' \in Y_2$.

Podmienka samočinného testovania zaručuje, že pre každú chybu z predpokladanej množiny chýb existuje aspoň jedno vstupné kódové slovo, z ktorého bude na výstupe nekódové slovo. Inak povedané, pre výskyt hocijakej chyby z predpokladanej množiny bude zistený aspoň jedným vstupným kódovým slovom. To platí jedine v prípade, ak vektory privedené na vstup obvodu tvoria úplný diagnostický test.

Na obrázku 2.3 si vysvetlíme podmienky pre obvod bezpečný proti poruchám.

Obvod je bezpečný proti poruchám z množiny F, ak pre každú poruchu $f \in F$ a pre každý vstupný vektor, je výstup buď rovný bezporuchovému výstupu, alebo nie je z množiny platných výstupov (nie je to kódové slovo). To znamená, že výstup pri výskyte poruchy je buď platné kódové slovo, alebo nekódové slovo.

Ak sa jedná o obvod bezpečný proti poruchám pre predpokladanú množinu chýb, potom pre ľubovoľnú chybu $f \in F$ na výstupe obvodu bude buď nekódové slovo (z množiny Y_2) alebo iba správne kódové slovo (zelená podmnožina Y_1). To znamená, že obvod nikdy nevyprodukuje nesprávne výstupné kódové slovo pre dané (správne) vstupné kódové slovo.



Obrázek 2.3: Podmienka pre obvod bezpečný proti poruchám

Obvod nazývame *úplne samočinne kontrolovaný* (angl. *totally self-checking*), ak spĺňa podmienku samočinného testovania a zároveň podmienku bezpečnosti proti poruchám.

[1]Kódové slovo vzniká pripojením kontrolných bitov za normálne výstupné bity z obvodu. To vedie na použitie systematických kódov ako napr. parity alebo Bose-Linových a Bergerových kódov, alebo použitie nesystematických kódov ako je kód m-z-n.

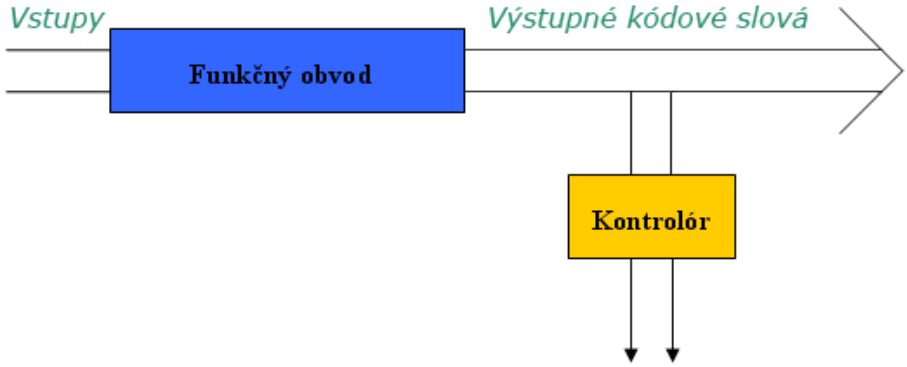
Obvody s takýmito vlastnosťami sú veľmi žiadané, keď potrebujme vysokú spoľahlivosť systému. Výhodné je ich nasadenie najmä kvôli:

a) schopnosti zistiť nielen permanentné, ale aj dočasné chyby (spôsobené napr. rušením),

b) chyby sú zistené ihneď po výskyte, čo zabraňuje práci s neplatnými dátami a

c) softvérové diagnostické programy už nie sú potrebné, alebo môžu byť značne zjednodušené.

Model úplne samočinne kontrolovaného obvodu (viď obrázok 2.4) sa skladá z funkčného obvodu a kontrolnej jednotky, nazývanej aj kontrolór (angl. checker), pričom obidve tieto hlavné časti sú úplne samočinne kontrolovateľné. Úlohou kontrolóra je kontrolovať platnosť výstupných kódových slov. Teda pozorovaním výstupov z tejto jednotky je možné zistiť prítomnosť chyby v obvode, ale aj v kontrolnej jednotke samotnej. Avšak nie je možné presne určiť, či sa chyba vyskytuje v obvode alebo v kontrolóri, len zo samotných informácií poskytovaných kontrolnou jednotkou.



Obrázek 2.4: Model úplne samočinne kontrolovaného obvodu

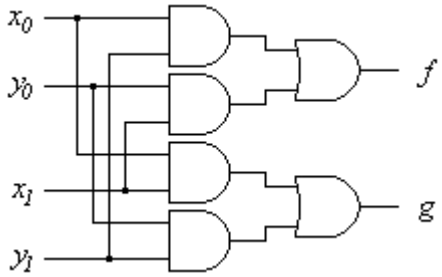
2.3 Samočinne kontrolovateľný kontrolór

Úplne samočinne kontrolovateľný kontrolór na obrázku 2.5 má typicky dva výstupy, preto existujú spolu štyri výstupné kombinácie. Dve z nich sú označené ako platné, napr. 0 1 , 1 0. Neplatný výstup 0 0 alebo 1 1 naznačuje, že sa jedná buď o nekódové slovo vstupujúce do kontrolóra alebo poruchu tejto jednotky. Kontrolór potrebuje dva výstupy aj preto, že ak by existoval iba jeden výstup a normálna hodnota na výstupe by bola napr. 1, tak potom chyba typu „trvalá 1“ by na výstupe nemohla byť nijako detekovaná. Podobne výstupné kombinácie 0 0 a 1 1 nie sú vybrané ako platné, pretože jednosmerná viacbitová chyba môže zmeniť 0 0 na 1 1. Podrobné informácie o návrhu a konštrukcii kontrolórov možno nájsť v [1, 5].

Obrázek 2.5: Úplne samočinne kontrolovateľný kontrolór

2.4 Kontrolór s dvomi výstupmi

Dvoj-drátový kontrolór[2] má dve skupiny vstupov $(x_1, x_2, \cdots, x_n)$ a $(y_1, y_2, \cdots, y_n)$ a dva výstupy f a g. Výstupné signály by mali byť vždy vzájomne komplementárne, čo zodpovedá kódu 1-z-2 práve vtedy, ak každá dvojica x_j, y_j je tiež komplementárna pre všetky j, pričom $(1 \leq j \leq n)$. Táto technika je ilustrovaná na obrázku 2.6 uvádzaného v [1].



Obrázek 2.6: Úplne samočinne kontrolovateľný kontrolór s dvomi výstupmi

Pre vstupy z obrázku 2.6 platí, že $y_i = \overline{x_i}$. V bezchybovom prípade, keď $x_0x_1 = 1\ 1$ a $y_0y_1 = 0\ 0$, sú na výstupoch výsledky $f = 0, g = 1$. V prípade výskytu chyby, napr. $y_0y_1 = 0\ 1$, sa na výstupe objaví nekódové slovo $f = g = 1$, čo naznačuje prítomnosť chyby. V skutočnosti je obvod z obrázku 2.6 úplne samočinne kontrolovaný pre všetky jednonásobné a viacnásobné jednosmerné chyby.

Hoci je možné navrhnúť kontrolór s dvomi výstupmi pre ľubovoľný počet vstupných dvojíc, v praxi sa používa zapojenie, ktoré je realizované ako strom prepojených modulov kontrolórov s dvoma vstupnými pármi.

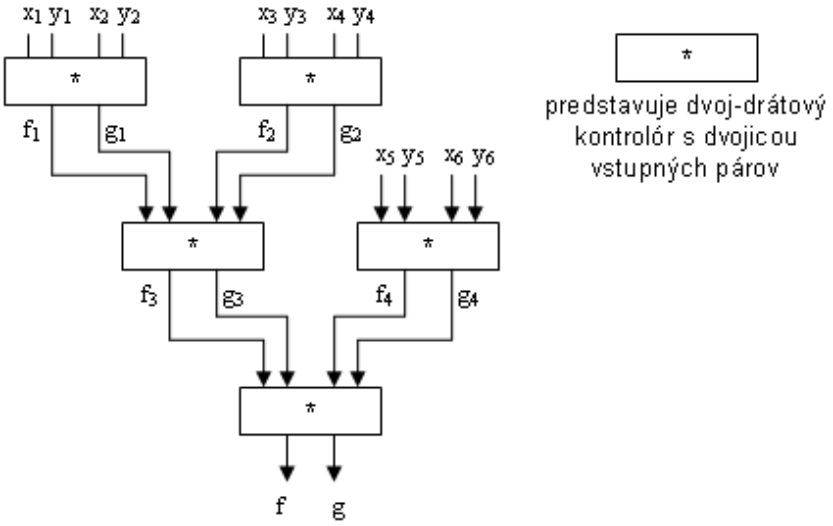
Všeobecne je viacúrovňová stromová realizácia pre kontrolór s m vstupnými pármi tvorená prepojením modulov s x vstupnými pármi. Vyžaduje sa $\lceil (m-1)/(x-1) \rceil$ modulov a $\lceil \log_2 m \rceil$ úrovní. Príklad takého zapojenia s $m = 6$ a $x = 2$ je na obrázku 2.7.

Ďalej je známe, že je možné takto vyskladať ľubovoľný n-párový kontrolór s dvomi výstupmi. Viac o tejto problematike je možné nájsť v [1].

2.5 Zhrnutie metód konvenčného návrhu

V niektorých prípadoch, keď je požadovaná vysoká spoľahlivosť obvodov, je nutná prítomnosť vstavaného mechanizmu zabezpečujúceho kontrolu funkčnosti obvodu. Typicky býva použitý generátor testovac9ch vektorov generujúci vstupné signály pre časť kombinačného

[2] Ako preklad termínu Two-rail checker zo zahraničnej literatúry sa uvádza dvoj-drátový kontrolór alebo kontrolór s dvomi výstupmi.



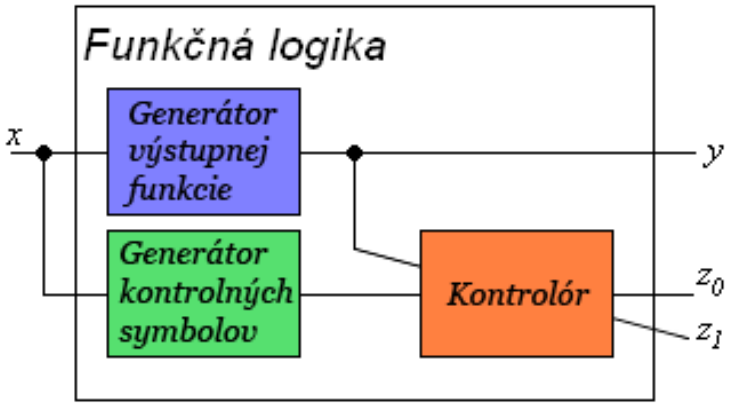
Obrázek 2.7: Úplne samočinne kontrolovaný dvoj-drátový kontrolór so 6 vstupnými pármi

obvodu a analyzátor signatúr skúma výstupné signály a vytvára syndróm chyby. Tento syndróm chyby je porovnaný so správnym syndrómom a na základe výsledkov sa určí, či logika funguje podľa špecifikácie.

Testovacie techniky môžeme klasifikovať na súbežné (CED) a nesúbežné. V kategórii súbežných sú použité vstavané kontrolné jednotky, ktoré zabezpečujú neustále testovanie aj počas normálnej činnosti obvodu. Druhá kategória autonómne vykonáva testovanie vtedy, keď je zariadené odstavené od normálnej činnosti (napríklad pred nasadením do prevádzky). Prvá kategória je schopná zistiť ako prítomnosť trvalých, tak aj dočasných chýb. Skoro všetky techniky tejto kategórie rozkladajú obvod na 2 moduly (viď obrázok 2.8):

a) funkčnú logiku (zloženú z generátorov výstupnej funkcie a kontrolných symbolov)

b) kontrolór (najčastejšie dvoj-drátový)



Obrázek 2.8: Príklad CED obvodu

Funkčná logika poskytuje výstupy zakódované chybovým detekčným kódom a kontrolór následne určí, či sa jedná o kódové slovo. Kontrolór tradične poskytuje dva výstupné signály, ktoré ak nadobúdajú rovnakú hodnotu, oznamujú tým prítomnosť chyby. Klasickou technikou je duplikácia, pri ktorej generátor výstupnej funkcie a generátor kontrolného syndrómu chyby sú identické jednotky.

2.6 Alternatívny prístup pomocou evolučného návrhu

Väčšina konvenčných techník syntézy sa stretáva s problémom štrukturálnych obmedzení pôvodného generátoru výstupných funkcií. Existujú aj techniky pre návrh obvodov testovaných počas svojej činnosti, ktoré nie sú zaťažené štrukturálnymi obmedzeniami [2, 3], ale majú buď viac než dvojnásobné nároky na plochu alebo sa výsledná latencia kontroly pohybuje za hranicou 1 hodinového taktu. Zdroj [6] tvrdí, že neexistuje konvenčná technika automatickej syntézy, ktorá by bola schopná pridávať logiku okolo nemodifikované obvodu generujúceho výstup tak, aby bola zachovaná vlastnosť úplnej samočinnej kontroly a zároveň spotrebovaný menej než dvojnásobok pôvodnej plochy.

M. Garvie v [6] navrhol metódu automatickej syntézy úplne samočinne kontrolovateľných obvodov. Táto technika používajúca genetické algoritmy bola schopná generovať kombinačné logické obvody s pomerne malým nárastom spotrebovanej plochy. Bolo ukázané, že vo vytvorených obvodoch nie je potrebné mať kontrolór a logika je použitá pre generovanie funkčných výstupov a kontrolu šírenia chýb. Ďalej bolo preukázané, že generovanie ad-hoc kontrolných stratégií pre každý obvod je vysoko prispôsobivé pre rôzne typy pôvodných obvodov. A napokon, že je možné uskutočniť jednoduché pridanie logiky k nemodifikovanému pôvodnému obvodu tak, že potom výsledok spĺňa stanovené podmienky samočinného testovania. Z vytvorených návrhov boli vybrané princípy návrhu blokov šíriacich chybu a boli riešené možnosti vylepšenia danej syntéznej techniky. Práca položila základ vedúci k rýchlej syntéze samočinne testovateľných obvodov s nízkym nárastom spotreby ďalšej plochy pre veľké testovacie obvody (benchmarky) a akceleráciu prehľadávania priestoru pomocou GA.

Kapitola 3

Kartézske genetické programovanie

Kartézske genetické programovanie (skr. CGP z angl. Cartesian Genetic Programming) bolo prvý krát popísané v práci Juliana Millera a Petera Thomsona [7]. Jeho zámerom bolo pomocou evolúcie navrhnúť číslicové obvody. CGP je možné chápať ako variantu genetického programovania, ktorá sa od genetického programovania líši v reprezentácii problému. V kartézskom genetickom programovaní sú programy všeobecne reprezentované orientovanými acyklickými grafmi (namiesto stromov). Reprezentácia pomocou grafov prináša oproti iným možnostiam výhodu v podobe možnosti implicitného znovupoužitia vytvorených podgrafov. Kartézske genetické programovanie používa obdĺžnikovú mriežku výpočetných uzlov (computational nodes), kde uzly musia splňovať pravidlá prepojenia, ktoré budú popísané v ďalšom texte. Z pohľadu návrhu číslicových obvodov sa takto dajú navrhovať jedine kombinačné logické obvody, pretože je zakázaná spätná väzba. CGP rieši niektoré problémy genetického programovania. Viac o tejto problematiky je možné nájsť v závere kapitoly alebo v [7, 8].

3.1 Reprezentácia číslicového obvodu pomocou CGP

Definujeme pevne rozmery mriežky (výška a šírka). Obdĺžnikový tvar je výhodou, kvôli tomu, že aj na čipe je najvhodnejšie realizovať štruktúry ktoré majú obdĺžnikový tvar, aby sa čo optimálne využila malá plocha. Ďalej špecifikujeme počet primárnych vstupov a výstupov. Definujeme konečnú množinu logických funkcií pre výpočetné uzly (bločky).

Evolučný algoritmus pracuje s chromozómom konštantnej dĺžky, ktorý je tvorený celými číslami. Pre danú reprezentáciu kde majú všetky výpočetné uzly rovnaký počet vstupov a výstupov (typicky 1), môžeme dĺžku chromozómu určiť podľa vzťahu 3.1:

$$DLZKA = m * n * (i + 1) + o \tag{3.1}$$

kde:
m = počet riadkov
n = počet stĺpcov
i = počet vstupov výpočetného uzlu
o = počet primárnych výstupov obvodu

Pre návrh kombinačných logických obvodov sa typicky používajú bločky s 2 vstupmi a jedným výstupom. Celý obvod máva obvykle do 10 primárnych vstupov a výstupov.

3.2 Pravidlá zapojenia

Vstup uzlu môže byť pripojený ku primárnemu vstupu alebo k výstupu iného uzlu, ktorý sa nachádza v predchádzajúcich stĺpcoch. V rámci rovnakého stĺpca je toto prepojovanie zakázané. Z pohľadu číslicových obvodov tak môžu vzniknúť jedine kombinačné logické obvody.

V CGP bol zavedený špeciálny L-back parameter, ktorý určuje konektivitu uzlov vo vytváranom grafe. L-back parameter určuje počet stĺpcov predchádzajúcich i-ty stĺpec, z ktorých môže byť vybraný vstup pre uzol nachádzajúci sa v danom stĺpci. Primárne vstupy sú obvykle chápané ako výstupy „nultého“ stĺpca[1]. Tento parameter nadobúda rozsah hodnôt z celočíselného intervalu $< 1, n >$.

Tabulka 3.1: Vplyv parametru L-back na mriežku s konštantným počtom uzlov

L-back	**Ďalšie podmienky**	**Efekt**
1	———	môžu byť prepojené iba susedné stĺpce, takéto zapojenie je veľmi vhodné pre zreťazenú HW implementáciu navrhovaného obvodu
n	———	neexistuje obmedzenie na prepojovanie medzi stĺpcami
n	Iba 1 riadok	je povolená maximálna možná konektivita, najväčší stavový priestor

Z tabuľky 3.1 je zrejmé, že parameter L-back výrazne ovplyvňuje množinu vytvoriteľných grafov a tým pádom aj veľkosť prehľadávaného stavového priestoru.

Primárne výstupy nie sú nijako obmedzené, môžu byť vyvedené z ľubovoľného uzlu v grafe. Počet primárnych vstupov a výstupov v mriežke by mal korešpondovať so vstupmi a výstupmi hľadaného obvodu.

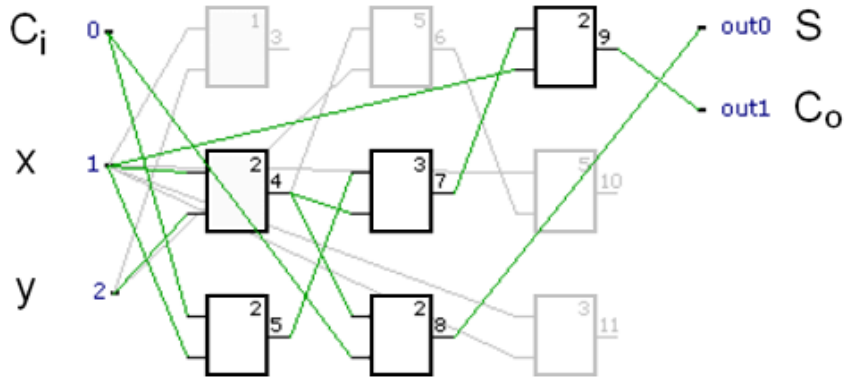
Každý uzol v mriežke má priradené špecifické číslo, ktoré určuje ako môže byť obvode zapojený. Číslovanie bločkov prebieha sekvenčne po stĺpcoch, pričom sa začína od prvého primárneho vstupu, ktorému je priradené číslo 0.

3.3 Chromozóm používaný v CGP

Príklad z obrázku 3.1 prevzatého z [9] má 3 primárne vstupy označené ako C_i, x, y a očíslované 0, 1, 2. Veľkosť matice programovateľných bločkov je 3 riadky $\times$ 3 stĺpce. Sú použité bločky s dvomi vstupmi a jedným výstupom. Obrázok názorne ukazuje ako prebieha číslovanie po stĺpcoch. Bločkom bola priradená konkrétna logická funkcia z nasledovnej množiny funkcií:

$$\mathbf{F} = \{NAND\ (0),\ NOR\ (1),\ XOR\ (2),\ AND\ (3),\ OR\ (4),\ NOT\ (5)\} \tag{3.2}$$

[1] V niektorých implementáciách CGP je povolené priviesť primárny vstup na vstup ľubovoľného výstupu uzlu, aj keď to nastavenie parametru L-back nedovoľuje.



Obrázek 3.1: Fenotypová reprezentácia úplnej 1-bitovej sčítačky v CGP v mriežke 3×3

Chromozóm pre obvod z obrázku 3.1 vyzerá nasledovne:

$$Chr: \ (1,2,1)\ (1,2,2)\ (0,1,2)\ (4,2,5)\ (5,4,3)\ (4,0,2)\ (7,1,2)\ (1,6,5)\ (1,1,3)\ (8,9)$$

Genotyp je tvorený nasledovne:
Na konci (v poslednej zátvorke) sa nachádza údaj o tom, z ktorých bločkov sú vyvedené jednotlivé primárne výstupy. V danom prípade sú to bločky 8 a 9. Počet číselných hodnôt v poslednej zátvorke je zhodný s počtom primárnych výstupov.

Vo všetkých ostaných predchádzajúcich zátvorkách sú zakódované prepojenia a logické funkcie bločkov zoradených vzostupne podľa priradeného čísla. Celé čísla v každej zátvorke reprezentujú vstupy a výstupy uzlu. Vidíme nasledovnú situáciu:

Začína sa od uzlu číslo 3, pretože obvod má 3 primárne vstupy očíslované 0, 1 a 2.

Uzol č.3 - (1,2,1) má privedený na svoj 1. vstup výstup z uzlu číslo 1 (primárny vstup), má privedený na svoj 2. vstup výstup z uzlu číslo 2 (primárny vstup) a realizuje logickú funkciu číslo 1 (NOT).

Uzol č.4 - (1,2,2) má privedený na svoj 1. vstup výstup z uzlu číslo 1 (primárny vstup), má privedený na svoj 2. vstup výstup z uzlu číslo 2 (primárny vstup) a realizuje logickú funkciu číslo 2 (XOR).

Uzol č.5 - (0,1,2) má privedený na svoj 1. vstup výstup z uzlu číslo 0 (primárny vstup), má privedený na svoj 2. vstup výstup z uzlu číslo 2 (primárny vstup) a realizuje logickú funkciu číslo 2 (XOR).

Uzol č.6 - (4,2,5) má privedený na svoj 1. vstup výstup z uzlu číslo 4, má privedený na svoj 2. vstup výstup z uzlu číslo 2 (primárny vstup) a realizuje logickú funkciu číslo 5 (NOT).

A obdobne sa pokračuje ďalej až do vyplnenia celej matice 3 riadkov × 3 stĺpcov.

3.4 Vytvorenie fenotypu z genotypu

Príklad budeme ilustrovať na mriežke tvorenej 3 riadkami a 3 stĺpcami v obvode s 3 vstupmi a 2 výstupmi. Množina funkcií bude zhodná s 3.2. Niektoré uzly nemusia využívať všetky vstupy, napríklad 2-vstupový bloček realizujúci NOT.

$$\textit{Genotyp}: \quad (1,2,1)(1,2,2)(0,1,2)(4,2,5)(5,4,3)(4,0,2)(7,1,2)(1,6,5)(1,1,3)(8,9)$$

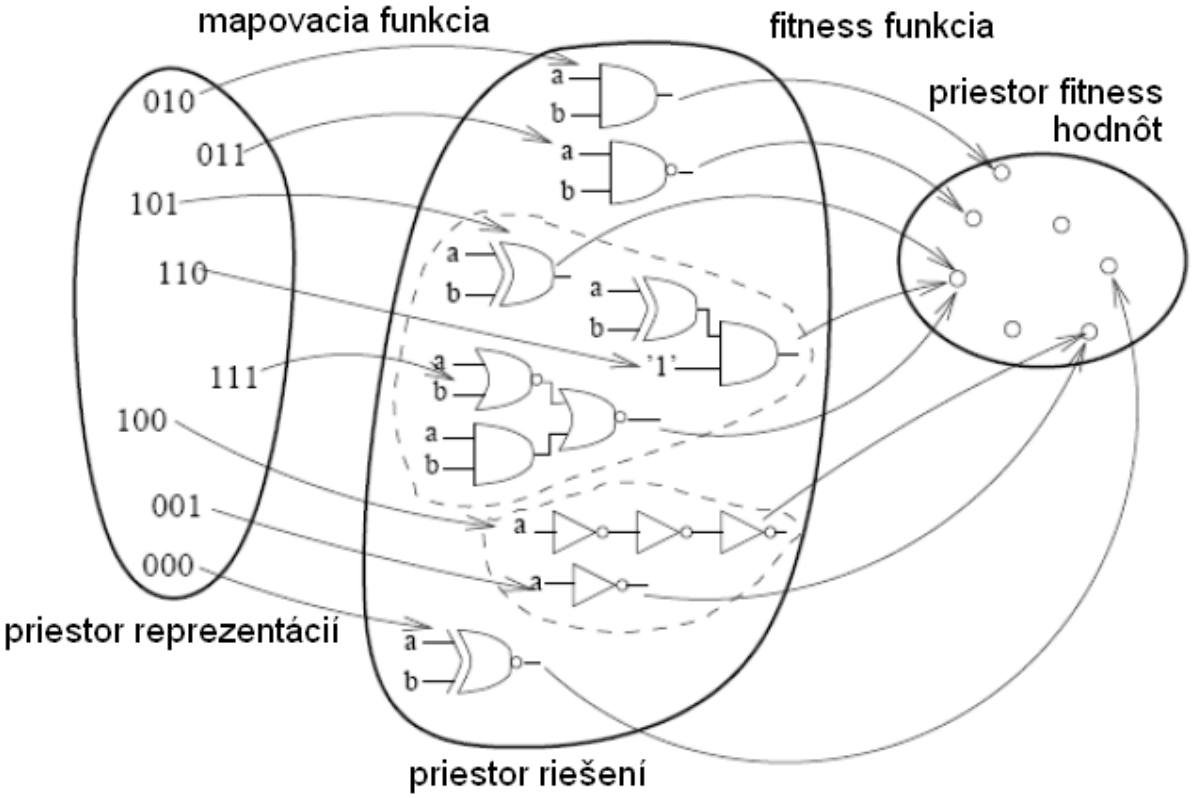
Postupujeme zo strany výstupov obvodu (napravejšia časť genotypu[2]) a potom podľa indexov uzlov (zostupne). Najprv zakreslíme do schémy bloček číslo 9, potom bloček 8 a vyvedieme ich na primárne výstupy. Bloček číslo 8 bude hradlo realizujúce funkciu XOR, bloček 9 takisto. Prvý vstup bločku 8 pripojíme na výstup bločku 4, a na druhý vstup privedieme primárny vstup 0. Prvý vstup bločku 9 pripojíme na výstup bločku 7, a na druhý vstup privedieme primárny vstup 1. Pokračujeme rozpracovaním bločku číslo 7, pretože je to najvyššie platné číslo v rade. Na jeho prvý vstup pripojíme výstup z bločku číslo 5, na druhý vstup privedieme výstup z bločku číslo 4. Bloček číslo 7 bude realizovaný logickým hradlom AND. Ďalej pokračujeme bločkom 5, ktorý bude realizovaný hradlom XOR a na jeho vstupy budú privedený primárne vstupy 0 a 1. Potom pokračujeme bločkom 4, ktorý bude realizovaný hradlom XOR a na jeho vstupy budú privedené primárne vstupy 1 a 2. Žiadne ďalšie bločky už nenasledujú, preto je konštrukcia fenotypu skončená (viď tučne zvýraznené bloky na obrázku 3.1).

Bločky s číslami 3, 6, 10 a 11 označené šedou farbou vo výslednom obvode nie sú potrebné. Chromozóm teda môže obsahovať aj redundantné informácie z hľadiska vytvorenia fenotypu. Redundancia je však z pohľadu evolučného návrhu výhodná, pretože dáva vzniknúť tzv. neutralite. Neutralita nastáva ak kandidátni jedinci dosiahnu rovnaké ohodnotenie pomocou fitness funkcie. Tento fakt pomáha evolúcii pri hľadaní riešení [12]. Daný obvod realizuje funkciu úplnej 1-bitovej sčítačky a jedná sa o obvod zobrazený na obrázku 3.1.

[2]V reálnych implementáciách CGP sa chromozóm realizuje poľom čísel typu integer. V tomto prípade má oddelenie čísel pomocou čiarok a zátvoriek len ilustratívny charakter.

3.5 Kódovanie CGP spôsobuje neutralitu

Dva jedince (fenotypy) sú vzájomne neutrálne, ak im je pomocou fitness funkcie priradená rovnaká fitness hodnota. Neutralita sa v CGP vyskytuje vo zvýšenej miere oproti klasickému genetickému programovaniu.



Obrázek 3.2: Zobrazenie genotyp – fenotyp a fenotyp – fitness hodnota

Obrázok 3.2, prevzatý z [9], ukazuje bijektívne zobrazenie genotyp – fenotyp. Nech fitness funkcia ohodnocuje kandidátne obvody len na základe ich funkčnosti (neuvažujeme plochu, príkon a podobne). Zobrazenie fenotyp – fitness hodnota už bijektívne nie je, pretože je všeobecne porušená injektívnosť zobrazenia. Viacerým obvodom možno prisúdiť rovnaké funkčné ohodnotenie.

Teória neutrálnej evolúcie [10] vysvetľuje, že väčšina účelových znakov u organizmov vznikla náhodou, a väčšina genetických zmien mala neutrálne vplyvy na organizmy (na ich zdatnosť). Ale dôležitý vplyv neutrality spočíva v tom, že chráni organizmy pred škodlivými mutáciami.

3.6 Algoritmus používaný v CGP

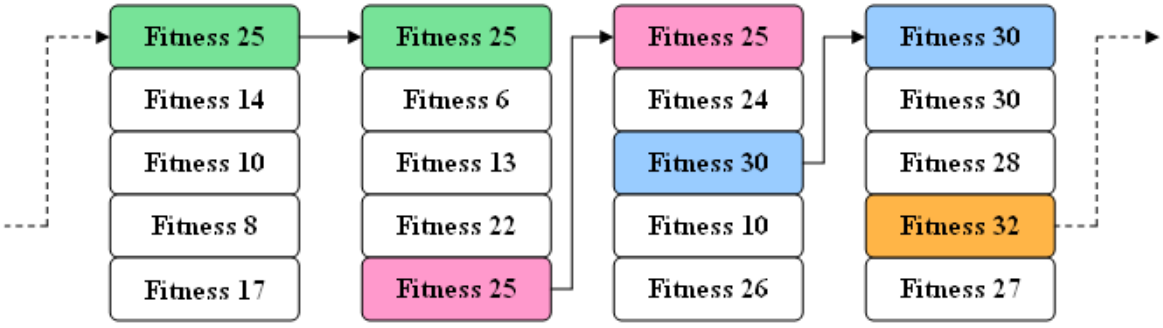
Parametre ako počet primárnych vstupov a výstupov, L-back, počet riadkov a stĺpcov sú počas evolúcie nemenné. Mení sa prepojenie jednotlivých bločkov a ich logické funkcie.

Algoritmus používaný v CGP odzrkadľuje evolučnú stratégiu $(1 + \lambda)$, kde hodnota λ sa rádovo pohybuje v jednotkách. Používa sa výhradne genetický operátor mutácie. Celý algoritmus sa dá popísať nasledujúcimi krokmi:

1. Vygeneruje sa $1 + \lambda$ náhodných jedincov (chromozómov) pre inicializáciu populácie
2. Ohodnotenie všetkých jedincov v populácii pomocou fitness funkcie
3. Vyberie sa najschopnejší kandidát (jedinec s najvyššou fitness)
   Ak najlepšie fitness ohodnotenie dosiahne opäť rodič z predchádzajúcej generácie, tak sa hľadá sa rovnako kvalitný potomok; v prípade ak existuje, stane sa práve tento potomok novým rodičom
4. Vygeneruje sa λ potomkov operáciou mutácie najlepšieho jedinca
5. Najlepší nájdený jedinec spolu s λ potomkami vytvorí novú populáciu
6. Ak nie sú splnené ukončujúce kritériá, tak sa pokračuje znovu od kroku 2

Počiatočná populácia sa môže generovať náhodne, alebo sa môžu použiť zakódovanie známych kvalitných riešení, ktoré sa bude evolúcia snažiť ešte nejako vylepšiť. Populáciu jedincov tvorí $1 + \lambda$ jedincov a nová populácia sa tvorí tak, že sa zo starej populácie vyberie „najlepší“ jedinec a spolu s ním sa do novej populácie dostane aj λ jedincov vzniknutých mutáciou vybraného jedinca.

Keď v dvoch po sebe nasledujúcich generáciách by sa mal ako rodič použiť ten istý obvod, tak sa testuje, či náhodou neexistuje rovnako kvalitný potomok, ktorý by sa stal novým rodičom pre nasledujúcu generáciu. Je to z toho dôvodu, aby zostala čo najvyššia diverzita populácie. Mutovaní potomkovia z rovnakého rodiča v niekoľkých generáciách po sebe sa považujú za degenerovanú populáciu. Ukončovacím kritériom môžu byť napríklad dosiahnutie požadovanej fitness hodnoty alebo vyčerpanie počtu generácií pre evolúciu (rádovo milióny).



Obrázek 3.3: Výber najkvalitnejšieho jedinca z populácie vo vybraných 4 generáciách

Na obrázku 3.3 je ilustrovaný prípad výberu rodiča z kandidátnych jedincov. Šípkou je naznačený kandidát zvolený rodičom, pričom pozícia rodiča je vždy na vrchu každého stĺpca, ktorý zas predstavuje populáciu v danej generácii. Najprv sa mutáciou rodiča nepodarilo nájsť nového kvalitnejšieho jedinca. Preto sa (zelený) kandidát s fitness 25 stal rodičom aj pre nasledujúcu generáciu. Potom je ilustrovaný prípad, keď sa za rodiča zvolí potomok (ružový) s fitness 25, ktorý mal rovnakú fitness funkciu ako rodič z poslednej generácie. Na obrázku pekne vidno, že v ďalšom priebehu sa podarilo vytvoriť aj kvalitnejších potomkov, ktorí sa stali rodičmi nových generácií. Rodičom je vždy najvrchnejší prvok v každom stĺpci, pričom celý stĺpec predstavuje populáciu z danej generácie.

Pre kartézske genetické programovanie je typické použitie vysokého počtu generácií a malého počtu jedincov v populácii.

3.7 Použitie operátoru kríženia a mutácie v CGP

Kríženie sa v kartézskom genetickom programovaní vôbec nepoužíva. Nepoužíva sa preto, že nie známe ako ho aplikovať tak, aby to uspokojivo fungovalo. Existujú štúdie [11], ktoré ukazujú rôzne typy krížení v CGP, ale nebolo preukázané, že by bolo výhodné to či ono kríženie používať. Pretože doposiaľ publikované výsledky nepreukázali uspokojivý prínos kríženia pre CGP, tak nie je používané.

Mutácia funguje tak, že sa náhodne vyberie gén v danom rodičovskom chromozóme a vygeneruje sa jeho nová hodnota. Je potrebné zaistiť, aby táto hodnota nebola úplne náhodná, ale splňovala určité kritériá. Musia byť splnené kritériá ako napríklad L-back parameter, číslo kódujúce logickú funkciu nesmie byť vyššie než počet logických funkcií v množine funkcií a podobne. To sú už ale skôr implementačné špecifiká. Mutáciou, ktorá je v CGP chápaná ako silný operátor, sa toho môže veľa zmeniť. Napríklad topológia celého obvodu, zmena primárnych výstupov z obvodu, zmena neaktívnych uzlov na aktívne.

Mutácia sa nazýva neutrálnou, ak sa jej pomocou dosiahne rovnaké fitness ohodnotenie. Neutrálne mutácie sú v princípe jedným z dvoch druhov:

a) z pôvodného fenotypu sa stane nový fenotyp, ale má rovnaké ohodnotenie pomocou fitness, alebo

b) k mutácii dôjde v časti fenotypu, ktorá nie je vo fenotype znázornená (v neaktívnej vetve)

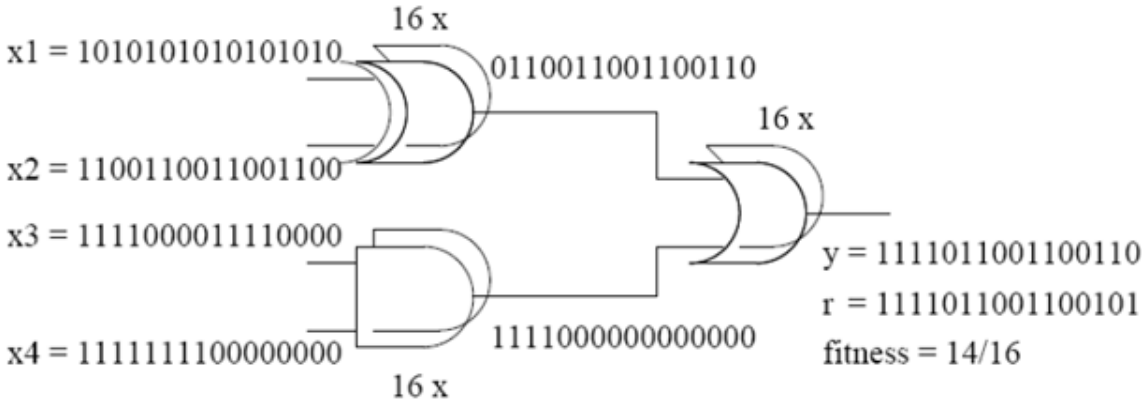
Ale aj neutrálne mutácie môžu mať priaznivý vplyv niekedy v budúcnosti z pohľadu priebehu evolúcie.

Dôležitým faktorom je pravdepodobnosť mutácie, ktorá určuje ako často k mutácii dochádza.

3.8 Paralelná simulácia

Nech pomocou kartézskeho genetického programovania navrhujeme kombinačné logické obvody. Fitness funkcia nech určuje počet správnych výstupov kandidátneho obvodu oproti požadovaným výstupom. Najzdĺhavejšou a najfrekventovanejšou časťou evolúcie je vyhodnotenie výstupov kandidátneho obvodu. Na vyhodnotenie kvality použijeme simulátor, ktorý generuje všetky vstupné kombinácie. Postupovalo by sa nasledovne. Vezme sa vstupná kombinácia, prejde sa postupnosť hradiel a poznačí sa výsledok, ktorý sa porovnáva s požadovaným výstupom. Potom prebieha to celé odznova s ďalšou vstupnom kombináciou, až kým nevyčerpáme všetky možnosti. Ak má obvod $\boldsymbol{x}$ vstupov, muselo by vyhodnotenie prebehnúť $\mathbf{2}^{\boldsymbol{x}}$ krát. To je samozrejme pomerne mnoho, preto sa používa na simuláciu špeciálny postup.

Urýchlenie simulácie spočíva v tom, že sa vhodne zakóduje vstupná pravdivostná tabuľka. Ak procesor dovoľuje použitie binárnych logických operácií (AND, OR, XOR, ...) na číslach typu integer, tak sa využije tejto vlastnosti. Potom je v jedinom priechode vypočítaných až toľko vstupných variant koľko bitov je použitých pre reprezentáciu integeru. Napríklad 64–bitová aritmetika dovoľuje až 64-násobné urýchlenie (pre obvody až do 6 vstupov) oproti naivnému prístupu. Na záver sa porovná výsledok s číslom reprezentujúcim požadované výstupy.



Obrázek 3.4: Paralelná simulácia kandidátneho obvodu

Obrázok 3.4 ukazuje 4-vstupový kandidátny obvod. Celá pravdivostná tabuľka je zakódovaná do štyroch 16-bitových čísel typu integer. Pomocou 16-bitovej aritmetiky sa vyhodnotia výstupy len v jedinom priechode. Získa sa tak 16-násobné urýchlenie oproti naivnému prístupu. Záverečným porovnaním s požadovaným výstupom r = 1111011001100101 bolo kandidátnemu obvodu priradené fitness ohodnotenie 14 pretože dosiahol zhodu v 14 bitoch s požadovaným výstupom. Simuláciu obvodu možno k-bitovou aritmetikou urýchliť až k-krát, ak má daný obvod najviac x vstupov, pričom $k = 2^x$.

3.9 Problémy kartézskeho genetického programovania

So zvyšujúcou sa zložitosťou požadovaných obvodov narastá aj čas potrebný na to, aby evolúcia bola schopná nájsť uspokojivé riešenie. Takisto rastie aj počet generácií, ktoré sú na to potrebné. Počet všetkých možných kombinácií na vstupoch kombinačného logického obvodu rastie exponenciálne s každým pridaným vstupom. V ohodnotení v SW sa hranica nájdenia uspokojivého riešenia v akceptovateľnom čase pohybuje okolo 10 vstupov. Určité zlepšenie prináša vyhodnocovanie v HW, keď sa použitím FPGA dá pre určitú triedu obvodov nájsť obvod až s 25 vstupmi [9]. S rastúcou zložitosťou cieľového obvodu klesá počet behov, ktoré skončia úspešne. Tento problém sa nazýva problém škálovateľnosti evolučného návrhu [15].

V určitých prípadoch sa ešte zavádza takzvaná trénovacia množina, ktorá je tvorená iba percentuálnou časťou všetkých vstupných kombinácií. Ohodnocovanie prebieha oproti tejto množine, avšak nájdené riešenia zaručene fungujú len na určitú podmnožinu všetkých vstupov. To sa dá využiť ak sa nám to hodí v konkrétnych úlohách (napr. filtrácia obrazu). Ak sa nám to nehodí (je požadovaná úplná zhoda výstupov), potom evolúciou nájdené riešenia ešte porovnávame oproti kompletnej požadovanej výstupnej kombinácii.

Výhodou kartézskeho genetického programovania oproti štandarnému genetickému programovaniu je, že genotyp má fixnú dĺžku a preto väčšinou behom evolúcie nevzniká nežiadúci efekt, tzv. BLOAT. Gény sú reprezentované pomocou celých čísel, ktoré kódujú výpočetnú funkciu daného uzlu a prepojenie jednotlivých uzlov. V rámci mriežky je vytvorený orientovaný acyklický graf, ktorý má pevnú veľkosť. Na základe reprezentácie je zostrojený fenotyp. Môže nastať situácia (a často k nej dochádza), keď sa určité gény neprejavia na výslednej podobe nájdeného riešenia. Časť obvodu z pohľadu cesty z primárnych

vstupov k výstupom je zbytočná. To vedie na ohraničený fenotyp premenlivej dĺžky. Charakteristika tohto typu genotypovej redundancie bola predmetom rôznych detailných výskumov [7, 8, 12, 13, 14] a bolo zistené, že je veľmi prospešné mať tento takzvaný efekt neutrality. Keď to zhrnieme, tak genotyp má pevnú veľkosť a fenotyp má premenlivú avšak ohraničenú veľkosť.

3.10 Aplikácie CGP

Kartézske genetické programovanie bolo úspešne použité v nasledovných oblastiach[3]

- návrh programov – symbolická regresia ...
- návrh číslicových obvodov
- návrh číslicových filtrov
- návrh kontrolérov pre roboty
- biológiou inšpirovaný development

Existujú aj určité varianty kartézskeho genetického programovania ako napríklad:

- CGP, v ktorom môžu vznikať moduly a kde je možné opätovné použitie týchto modulov,
- CGP s viacerými chromozómami,
- CGP s možnosťou spätnej väzby pre experimentálny návrh sekvenčných logických obvodov,
- a iné.

[3]Prehľad projektov s využitím metódy CGP je možné nájsť na http://www.cartesiangp.co.uk.

Kapitola 4

Polymorfné obvody

Polymorfné obvody sú také obvody, ktoré sú zložené z konvenčných hradiel (AND, OR, atď.) a zároveň z takzvaných polymorfných hradiel. Polymorfné hradlá sú špeciálne hradlá umožňujúce vykonávať pri rôznych pracovných podmienkach rôzne logické funkcie. Podmienkami môžu byť napríklad rôzne teploty, hladiny napájacieho napätia a podobne.

Predpokladá sa, že polymorfné obvody bude výhodné vstavať do inteligentných zariadení, ktorých funkcia má byť premenlivá v závislosti na okolitom prostredí. Takéto inteligentné zariadenia môžu disponovať vlastnosťami ako napríklad automatické riadenie spotreby energie pri poklese napájacieho napätia (polymorfný obvod by realizoval inú funkciu pri nízkom napätí, jeho štruktúra by sa však vôbec nezmenila). Ďalej také zariadenie môže implementovať funkciu, ktorá je pre bežného používateľa skrytá, ale pre výrobcu už nie je. Ešte inou aplikáciou môžu byť lacné adaptívne systémy, ktoré umožňujú meniť správanie v závislosti na určitých premenných prostredia.

Adrian Stoica predstavil koncept polymorfných hradiel pre implementáciu polymorfných logických sietí [16]. Pretože funkcia polymorfných hradiel závisí na vonkajších faktoroch ako je napr. napájacie napätie, teplota, intenzita svetla, dá sa povedať, že sú v istom zmysle konfigurovateľné. Takéto hradlá integrujú logickú funkciu so schopnosťou snímania. Takže polymorfné hradlá môžu byť veľmi užitočné pri vytváraní inteligentných zariadení, ktoré reagujú na okolité prostredie.

Polymorfné obvody sú také obvody, ktoré obsahujú nejaké polymorfné hradlá. Bolo ukázané, že pre návrh zložitejších polymorfných obvodov je možné použiť polymorfné hradlá v kombinácii spolu s klasickými hradlami. Tieto obvody podľa očakávania dosahujú zaujímavé známky správania sa, ktoré u klasických logických obvodov nie sú bežné [17].

Navrhnúť polymorfný obvod podľa požadovanej špecifikácie nie je triviálny problém. Doposiaľ neboli publikované pre návrh polymorfných obvodov konvenčné (matematické) metódy, ktoré sa využívajú pri návrhu klasických obvodov zložených z tradičných hradiel (napr. metóda Quine-McCluskey, Espresso). Tento nedostatok sa dá čiastočne obísť pomocou evolučného návrhu. V tomto projekte bude použitý návrh s využitím techniky kartézskeho genetického programovania.

4.1 Polymorfné hradlá

Teoreticky je možné navrhnúť polymorfné hradlo, ktoré implementuje $\boldsymbol{k}$ rôznych logických funkcií pre $\boldsymbol{k}$ rôznych stavov prostredí (pre hradlá s $\boldsymbol{n}$ vstupmi je hodnota $\boldsymbol{k}$ rovná max. $\mathbf{2^{2^n}}$).

V súčasnosti sa hodnota $\boldsymbol{k}$ pohybuje medzi 2 až 4. Ak bude možné polymorfné hradlo použiť ako stavebný blok, potom bude možné navrhovať „polymorfnú elektroniku“. Veľkým problémom je však návrh samotných polymorfných hradiel. Takmer všetky doposiaľ publikované polymorfné hradlá boli navrhnuté metódami evolučného návrhu. Táto situácia naznačuje, že ani skúsený návrhár nie je schopný takéto hradlá navrhovať konvenčným spôsobom.

V tabuľke 4.1 sú uvedené príklady niektorých doposiaľ publikovaných polymorfných hradiel. Najpopulárnejším príkladom polymorfného hradla je NAND/NOR [18]. Toto hradlo je zložené zo 6 tranzistorov a bolo vyrobené 0,5 mikrónovou CMOS technológiou. Hradlo je stabilné s maximálnou odchýlkou 10% napájacieho napätia a pre teploty v rozmedzí 20°C až 200°C. Príklad polymorfného hradla navrhnutého konvenčne je uvedený v [23].

Tabulka 4.1: Príklad doposiaľ publikovaných polymorfných hradiel

Hradlo	**Prahové hodnoty**	**Riadenie**	**Tranz.**	**Zdroj**
AND/OR	27/125°C	Teplota	6	[16]
AND/OR/XOR	3.3/0.0/1.5 V	Externé napätie	10	[16]
AND/OR	3.3/0.0 V	Externé napätie	6	[16]
AND/OR	1.2/3.3 V	Udd	8	[19]
NAND/NOR	3.3/1.8 V	Udd	6	[18]
NAND/NOR/NXOR/AND	0/0.9/1.1/1.8 V	Externé napätie	11	[20]
NAND/NOR	5/3.3 V	Externé napätie	8	[23]

4.2 Klasifikácia polymorfných obvodov

Polymorfné obvody môžeme klasifikovať do niekoľkých skupín:

Polymorfné obvody s viacerými funkciami sú také obvody, ktoré menia svoju funkciu v závislosti na móde polymorfných hradiel. Napríklad ak určitý obvod obsahuje polymorfné hradlo NAND/NOR riadené napájacím napätím Udd, potom sa pre Udd=3.3 V obvod môže správať ako sčítačka a pre Udd=1.8 V sa obvod môže správať ako násobička.

4.2.1 Charakteristika polymorfných obvodov s jednou funkciou

Problematiku budeme ilustrovať na polymorfných obvodoch s dvomi funkciami. Takéto obvody je buď možné navrhovať pomocou konvenčného návrhu s využitím binárnych rozhodovacích stromov [15] alebo pomocou nekonvenčných techník ako je evolučný návrh. Ani jeden z prístupov však nie je triviálny, naopak problematika návrhu je pomerne zložitá.

Pre evolučný návrh bolo úspešne použité kartézske genetické programovanie. Pre výpočet fitness hodnoty sa vygeneruje pravdivostná tabuľka kandidátneho obvodu a porovná sa s požadovanou pravdivostnou tabuľkou. Potom je fitness hodnota typicky definovaná takto:

$$\textbf{fitness} = \boldsymbol{B_0} + \boldsymbol{B_1} + (\boldsymbol{u} - \boldsymbol{g})$$

kde $\boldsymbol{B_0}$, resp. $\boldsymbol{B_1}$, je počet správnych bitov na výstupoch obvodu vzhľadom k hodnotám požadovanej logickej funkcie $\boldsymbol{f_0}$, resp. $\boldsymbol{f_1}$, $\boldsymbol{g}$ je počet využitých hradiel v obvode a $\boldsymbol{u}$ je

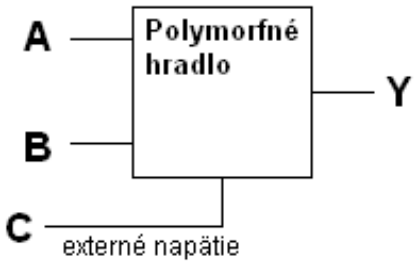
celkový počet programovateľných elementov. Výraz $(\boldsymbol{u} - \boldsymbol{g})$ je vyhodnotený iba vtedy, keď obvod funguje korektne v obidvoch módoch, inak platí $\boldsymbol{u} - \boldsymbol{g} = \mathbf{0}$. Spozdenie obvodu a ďalšie vlastnosti tu nie sú optimalizované.

Pri evolučnom návrhu s využitím kartézskeho genetického programovania však nastáva nepriaznivý jav a síce, že s rastúcim počtom vstupov/výstupov obvodu rastie časová aj pamäťová zložitosť algoritmu a to exponenciálne. Od určitého počtu vstupov potom nie je možné nájsť obvod požadovaných parametrov počas rozumnej doby. Kvôli problému škálovateľnosti evolučného návrhu musíme pre zložité polymorfné obvody hľadať a použiť inú metódu návrhu než CGP.

Polymorfné obvody s jednou funkciou realizujú iba jednu funkciu. Obsahujú polymorfné hradlá (riadené napr. externým napätím), ktoré umožňujú efektívnu implementáciu určitých funkcií [15]. Výstup ľubovoľného hradla tu môže pracovať ako riadiaci signál pre polymorfné hradlá.

4.2.2 Charakteristika polymorfných obvodov s jednou funkciou

Polymorfné hradlá v obvode sú riadené typicky externým napätím, ktoré môžeme v určitých prípadoch chápať ako korektnú logickú úroveň. Výstup ľubovoľného hradla môže pracovať ako riadiaci signál pre polymorfné hradlo. Takéto polymorfné hradlo je potom možné modelovať ako logický člen s 3 vstupmi tak, ako je znázornené na obrázku 4.1. Motiváciou pre použitie polymorfných hradiel riadených pomocou externého napätia je zlepšenie vlastností kombinačného obvodu oproti obvodom zložených iba z konvenčných hradiel [15]. Medzi takéto vlastnosti patrí cena/plocha obvodu, spotreba energie, spozdenie obvodu (delay) apod.



Obrázek 4.1: Znázornenie 2-vstupového polymorfného hradla riadeného ext. napätím

Obrázok 4.1 ukazuje schematické zobrazenie polymorfného hradla riadeného pomocou externého napätia. A a B sú vstupy pre logickú funkciu vykonávanú hradlom. Môže ňou byť napríklad NAND pre C v „logickej 0“, NOR pre C v „logickej 1“. Vstup externého napätia C riadi logickú funkciu hradla, napr. pre C v „logickej 0“ je externé napätie 0,0 V, pre C v „logickej 1“ je externé napätie 3,3 V. Výstup Y je výsledkom logickej funkcie: NAND pre C v „logickej 0“ alebo NOR pre C v „logickej 1“.

Metoda kartézskeho genetického programovania polymorfných obvodov s jednou funkciou vychádza z metódy CGP uvedenej v kapitole 3. Najvýznamnejšie rozdiely sú pre-

dovšetkým v tom, že programovateľný element obsahuje 3 vstupy, ďalej je použitá komplexnejšia mutácia a v neposlednom rade je vylepšená fitness funkcia. Viac o polymorfných obvodoch, ich návrhu a použití možno nájsť v [15].

Polymorfné obvody s jednou funkciou s diagnostickými vlastnosťami sú špecifické obvody typu polymorfný obvod s jednou funkciou.

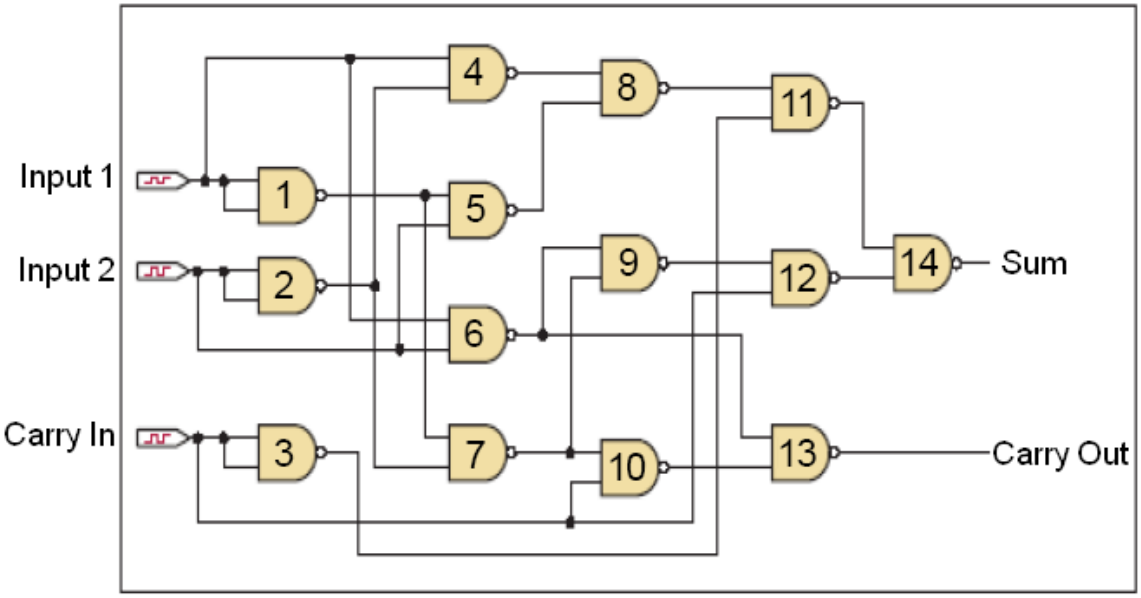
4.2.3 Charakteristika polymorfných obvodov s diagnostickými vlastnosťami

U týchto obvodov je možné zmenou prostredia diagnostikovať poruchu obvodu, ktorá sa môže prejaviť na výstupe pre rôzne prostredia odlišne - odlišnými výstupnými hodnotami. Napríklad obvod obsahujúci polymorfné hradlo NAND/NOR (z tabuľky 4.1) sa môže správať za bežných podmienok ako 1-bitová sčítačka pre napájacie napätie Udd=1,8 V, keď funguje ako hradlo NAND, tak aj pre napätie Udd=3,3 V, keď funguje ako NOR. Ak však vo vnútri obvodu nastane porucha, môže byť diagnostikovaná pomocou zmeny Udd (počas činnosti obvodu).

Kapitola 5

Analýza vybraných samočinne testovateľných polymorfných obvodov

Diplomová práca sa zaoberá návrhom polymorfných samočinne testovateľných obvodov. V tejto kapitole bude naznačené ako sa dá spojením dvoch prístupov navrhnúť samočinne testovateľný obvod s veľmi priaznivými parametrami. Jeden z prvých publikovaných obvodov [21] je znázornený na obrázku 5.1.



Obrázek 5.1: Sčítačka zložená zo 14 polymorfných hradiel typu NAND/NOR

Obvod sa skladá zo 14 polymorfných hradiel NAND/NOR. Bolo zistené, že tento obvod funguje ako sčítačka v obidvoch režimoch a ak je režim polymorfných hradiel menený s určitou frekvenciou, tak potom disponuje zaujímavými vlastnosťami 5.1.

- Ak nie je prítomná porucha, tak obvod pracuje ako sčítačka nezávisle na móde polymorfných hradiel.

- Obvod má určitú schopnosť samočinného testovania voči chybám trvalej úrovne.
- Obvod zisťuje prítomnosť chýb trvalá 0 a trvalá 1 na niekoľkých hradlách.
- Chyba je detekovaná pomocou oscilácie jedného alebo oboch výstupov medzi 0 a 1 s rovnakou frekvenciou akou je prepínaná funkcia polymorfných hradiel NAND/NOR.
- Chyby na hradlách 1, 2, 3, 10, 13, 14 sú nezistiteľné osciláciami na výstupe $\boldsymbol{Sum}$.
- Chyby na hradlách 3, 4, 5, 8, 9, 11, 12 sú nezistiteľné osciláciami na výstupe $\boldsymbol{C_{out}}$.
- Z toho vyplýva, že obvod nie je schopný zistiť chyby uviaznutia v nejakej logickej úrovni iba na hradlách číslo 3, 13 a 14.
- Na zisťovane prítomnosti chýb nie je potrebný žiadny špeciálny diagnostický signál.

5.1 Nové prístupy v samočinne testovateľných obvodoch

Ako bolo ukázané na poslednom príklade, je možné navrhovať také obvody, ktoré prítomnosť chyby naznačujú pomocou oscilácie na výstupe, prípadne viacerých výstupoch.

5.1.1 Zisťovanie chýb pomocou oscilácií

Polymorfné obvody s rovnakou funkciou v oboch módoch vykazujú schopnosť samočinného testovania. Ak je funkcia všetkých hradiel v obvode správna, tak prepínaním medzi režimami sa na výstupe polymorfných hradiel nemení logická úroveň signálu. Pri poruche typu trvalá úroveň na polymorfnom hradle sa môže výstup meniť s takou frekvenciou, akou sú prepínané jednotlivé režimy. Tým vznikajú oscilácie, ktoré sa šíria smerom k primárnym výstupom. Poruchy uviaznutia v trvalej logickej rovni na hradlách tvoriacich primárne výstupy sa nedajú pomocou oscilácií zistiť.

Oscilácie by sa mali šíriť obvodom na patričné výstupy a mali by byť pokryté všetky chyby uviaznutia v logickej úrovni. Ďalšou požiadavkou je, aby nebola spotrebovaná prílišná plocha na čipe.

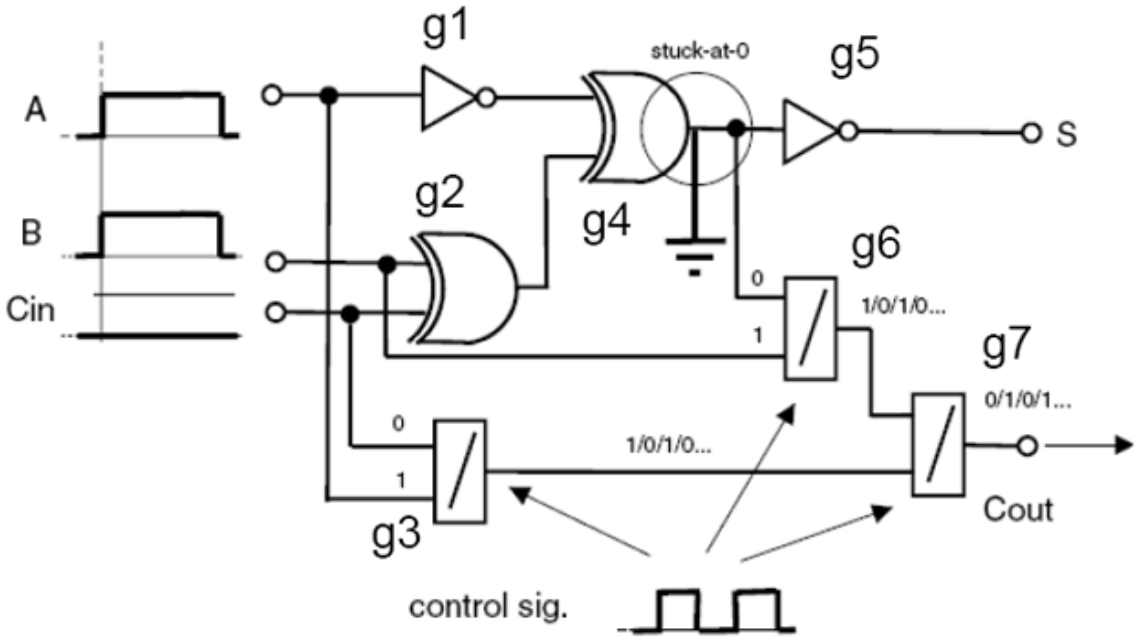
Pre návrh bola vyvinutá 2-stupňová metóda [17, 22]:

1. Pomocou evolúcie vytvoriť polymorfný obvod s jednou (požadovanou) funkciou s ohľadom na spotrebovanú plochu
2. Skontrolovať pokrytie chýb uviaznutia v logickej úrovni

5.1.2 Analýza konkrétneho obvodu

Ako príklad uveďme návrh 1-bitovej sčítačky metódou kartézskeho genetického programovania. Experimentálne bolo zistené, že najvhodnejšie parametre sú 3 riadky mriežky a 3 stĺpce, množina funkcií F = { NAND (0), NOR (1), XOR (2), AND (3), OR (4), NAND/NOR (5) }, veľkosť populácie až 15 jedincov a evolúcia prebiehajúca v 100000 generáciách. Mutácia mení jeden náhodný gén z najlepšieho kandidáta. Fitness funkcia pracuje tak, ako bolo popísané v kapitole 3.

Pokrytie chýb napríklad pre polymorfný obvod na obrázku 5.2 sa určí nasledovne. Vezme sa prípad uviaznutia v „logickej 0“ a pre všetky vstupné kombinácie sa pre každé hradlo určí, či je možné osciláciou zistiť chybu. To isté sa spraví aj pre prípad uviaznutia v „logickej 1“ pre každé hradlo a určí sa či je možné osciláciou zistiť chybu. Výsledky pre obvod z obrázku 5.2 možno nájsť v tabuľke na obrázku 5.3. Tento obvod bol publikovaný v [22].



Obrázek 5.2: Úplná 1-bitová sčítačka zložená z polymorfných a konvenčných hradiel

Symbolom × je v tabuľke naznačené, že sa osciláciami podarilo zistiť chybu uviaznutia v danej logickej úrovni na danom hradle. Z tabuľky sa ďalej určuje minimálna množina vstupných vektorov, ktoré sú potrebné na odhalenie všetkých uvažovaných chýb. V tomto prípade by sa jednalo o množinu $\boldsymbol{M = \{1, 2, 3, 5\}}$.[1]

Na zistenie jednonásobnej chyby uviaznutia v logickej úrovni pomocou oscilácií sú potrebné 4 testovacie vektory. Pravdepodobnosť odhalenia chyby iba jediným vektorom je 0,325 %. Implementačná cena takéhoto riešenia je 36 tranzistorov.

Podľa obrázku 5.2 sa pokúsime overiť správnu funkciu sčítačky (podľa [22]). Výstup S je nezávislý na móde polymorfných hradiel a je počítaný ako

$$\boldsymbol{S = \overline{A \oplus B \oplus C_{in}}} \tag{5.1}$$

Rovnako je pri správnej funkcii všetkých hradiel aj výstup $\boldsymbol{C_{out}}$ nemenný. Označme $\boldsymbol{C^{NAND}}$ výstup prenosu v režime NAND a $\boldsymbol{C^{NOR}}$ v režime NOR. Z obvodu získame pre výstup prenosu nasledovné vzťahy.

[1] Prípadne by mohli byť požité aj množiny M’ = { 1, 2, 5, 6 } alebo M” = { 2, 4, 5, 6 }.

M_g / vector	0	1	2	3	4	5	6	7	stuck-at
M_{g1}			x		x		x		0
M_{g2}			x	x	x	x			0
M_{g3}		x			x				0
M_{g4}				x		x	x	x	0
M_{g5}									0
M_{g6}	x		x						0
M_{g7}									0
M_{g1}		x		x		x			1
M_{g2}		x					x		1
M_{g3}				x			x		1
M_{g4}		x	x		x				1
M_{g5}									1
M_{g6}						x		x	1
M_{g7}									1

Obrázek 5.3: Pokrytie chýb uviaznutia v logickej úrovni pre obvod z obrázku 5.2

$$C^{NAND} = \overline{\overline{\overline{S}B} \cdot \overline{CA}} \tag{5.2}$$

$$C^{NOR} = \overline{\overline{S} + B + \overline{C + A}} \tag{5.3}$$

Pomocou DeMorganových zákonov môžeme C^{NAND} a C^{NOR} previesť na tvar:

$$C^{NAND} = \overline{S}B + CA \tag{5.4}$$

$$C^{NOR} = (\overline{S} + B) \cdot (C + A) = \overline{S}C + \overline{S}A + BC + AB \tag{5.5}$$

Pričom pre negáciu S platí:

$$\overline{S} = \overline{A}.\overline{B}.\overline{C} + A.B.\overline{C} + A.\overline{B}.C + \overline{A}.B.C \tag{5.6}$$

Dosadaním predchádzajúceho vzťahu (5.6) do predpisov pre prenosy v jednotlivých režimoch (5.4 , 5.5) získame vzťahy:

$$C^{NAND} = AB\overline{C} + \overline{A}BC + CA = AB + BC + AC \tag{5.7}$$

$$C^{NOR} = A\overline{B}C + \overline{A}BC + AB\overline{C} + BC + AB = AB + BC + AC \tag{5.8}$$

Pretože pre konvenčnú 1-bitovú sčítačku platí $C_{out} = AB + BC + AC$ potom

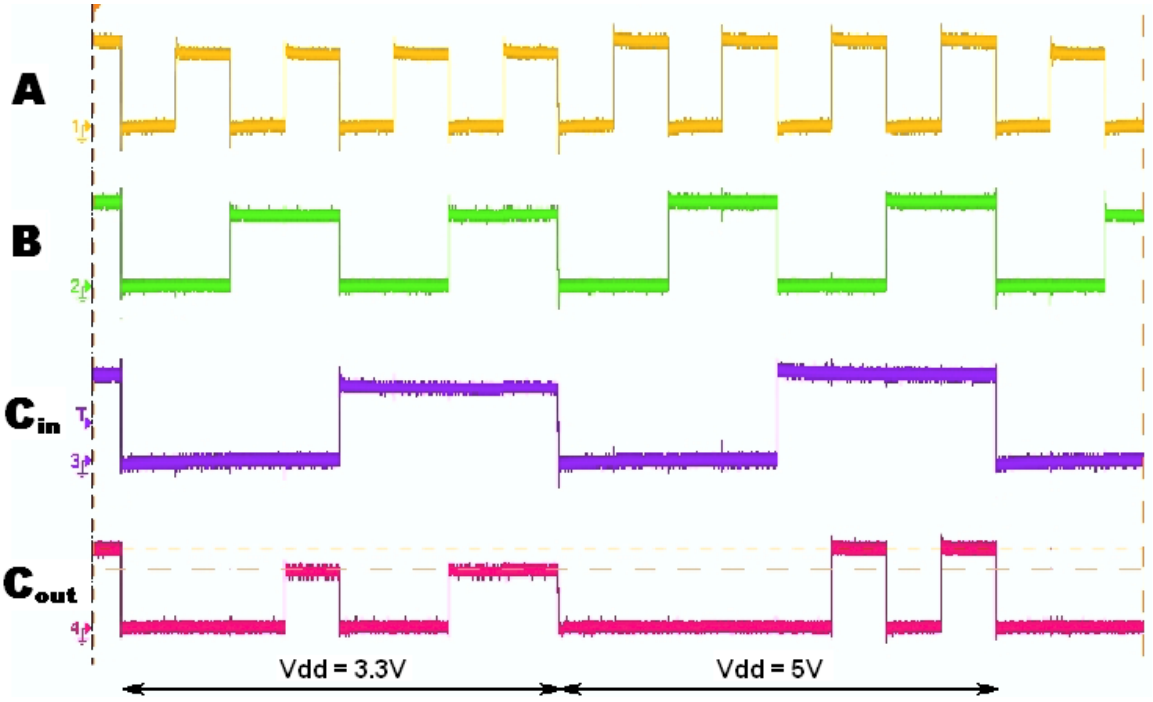
$$C^{NAND} = C^{NOR} = C_{out}$$

Z jednobitových sčítačiek možno pomocou kaskádneho zapojenia vyskladať n-bitovú sčítačku. Prítomnosť chýb sa osciláciami šíri smerom k výstupu C_{out}, pričom ale nastáva

pokles pravdepodobnosti detekcie chyby v častiach nachádzajúcich sa ďalej od primárneho výstupu C_{out}. Pokles je dvojnásobný s každou 1-bitovou časťou smerom k najmenej významnému bitu.

Takúto sčítačku by bolo principiálne možno použiť v pracovnom (on-line testing) alebo diagnostickom režime (off-line testing). V pracovnom režime sa nachádza ak sú prepínané módy polymorfných hradiel medzi NAND a NOR, pričom všetky bežné vstupy sú potom zároveň chápané aj ako testovacie vektory. V diagnostickom režime by na zistenie chyby stačili 4 testovacie vektory a testy by sa vykonávali periodicky v naplánovanej dobe. Nevýhodou však je, že zachytiť chyby uviaznutia v logickej úrovni na každom primárnom výstupe obvodu je principiálne nemožné. Všetky chyby sa prejavia osciláciami na výstupe C_{out}.

Navrhovaná sčítačka bola nedávno fyzicky vyrobená v technológii AMIS CMOS 0,7 μm. Jedná sa o jeden z prvých prípadov fyzickej realizácie samočinne kontrolovaného polymorfného obvodu. Obvod bol podrobený dôkladnému testovaniu, ktoré potvrdilo správanie odhadované podľa návrhovej schémy [23]. Sčítačka je zložená z 42 tranzistorov a je schopná pracovať na takej frekvencii, ako jej najpomalšie hradlo. Najpomalším hradlom je polymorfné hradlo NAND/NOR, ktoré dokáže pracovať až na 38,6 MHz v polymorfnom móde NAND, 56,2 MHz v polymorfnom móde NOR a je riadené externým napätím. Celá sčítačka je teda schopná pracovať na frekvencii 38,6 MHz v teplotnom rozsahu 0°C až 70°C. Obrázok 5.4 ilustruje odozvu sčítačky na chybu trvalej 0, ktorá je prítomná v hradle g4 (viď schéma na obrázku 5.2).



Obrázek 5.4: Zmeraná odozva sčítačky podľa schémy z obrázku 5.2 na chybu trvalá 0 v g4

Doposiaľ bolo publikovaných iba niekoľko polymorfných obvodov schopných samočinného testovania. Konkrétne návrhy možno nájsť napríklad v [22].

Kapitola 6

Koncepcia návrhu samočinne kontrolovateľných polymorfných obvodov

Ako je uvedené v predchádzajúcej kapitole, v minulosti už boli publikované príklady polymorfných obvodov schopných samočinného testovania [20, 22]. Metódy použité pre návrh takýchto systémov spočívajú v zostavení polymorfného obvodu s výstupnými funkciami totožnými vo všetkých (typicky dvoch) polymorfných módoch. Proces pokračuje analýzou získaného návrhu na schopnosť samočinnej kontroly. Tento dvoj-fázový prístup je však pomerne časovo náročný, pretože prechod medzi fázami nie je plynulý pretože si väčšinou vyžaduje zmenu prostredia. V dôsledku neexistencie vývojových nástrojov, je aj pre skúseného návrhára komplikované navrhovať polymorfné obvody podľa požadovanej funkcie, s ohľadom na schopnosti samočinného testovania navyše. Tam, kde konvenčné metódy zlyhávajú alebo neexistujú, sa vynára použitie evolučných techník. Nepríjemným aspektom evolučného návrhu týchto obvodov je, že ak chceme do riešenia aj nejaké konvenčné hradlá, tak evolúcia vyberá v prevažnej miere práve tieto hradlá a navrhované riešenia majú nulovú alebo iba nízku schopnosť samočinnej kontroly. Zatiaľ však nebol publikovaný materiál o návrhu takýchto obvodov pomocou evolúcie s využitím viacstupňovej komplexnej fitness funkcie. Navrhovať obvody práve týmto prístupom je cieľom v praktickej časti diplomovej práce. Návrhová časť aplikácie je založená na kostre kartézskeho genetického programovania, ktorú vytvoril Ing. Zdeněk Vašíček v roku 2007.

6.1 Návrh fitness funkcie

V evolučných technikách má použitie spôsobu ohodnocovania jedincov pomocou fitness funkcie zásadný vplyv na úspešnosť evolúcie. Voľba typu fitness funkcie má tiež zásadný dopad na dobu evolúcie, pretože výpočet fitness ohodnotenia kandidátneho jedinca pomocou kriteriálnej funkcie je obecne považované za najnáročnejší proces v evolučnej etape. Obzvlášť pri použití techniky kartézskeho genetického programovania, ktoré je typické vysokým počtom generácií pri riešení zadaného problému, je voľba ohodnocovania kandidátnych riešení fitness funkciou veľmi citlivou otázkou. Pri riešení diplomovej práce som vychádzal z dvoch základných možností:

1. Pri návrhu evolvovať kandidátne riešenia, až kým nie je dosiahnutá plná požadovaná funkčnosť. Potom nastaviť bonusovú zložku fitness funkcie v závislosti od schopnosti samočinnej kontroly jedinca a prípadne aj spotrebovanej plochy na čipe.

2. Použiť váhovanie s koeficientami pre funkčnosť, mieru samočinnej kontroly a spotrebovanú plochu, zároveň aplikované na každého kandidátneho jedinca.

6.1.1 Fitness funkcia s pomerným váhovaním a prioritizáciou funkčnosti

Tento spôsob ohodnotenia kandidátneho riešenia vychádza z odstupňovania jednotlivých vlastností kandidátneho riešenia podľa priority. Najvyššiu prioritu má úplná funkčnosť riešenia (totožná v oboch polymorfných módoch). Pokiaľ kandidátne riešenie nedosahuje plnú funkčnosť, potom nie je považované za akceptovateľné. V prípade dosiahnutia požadovanej funkčnosti, je k dosiahnutej fitness hodnote pripočítaná ešte istá bonusová zložka fitness, ktorá je odvodená od schopností samočinného testovania kandidátneho obvodu. Táto zložka je váhovaná pomocou koeficientov tak, aby bola pokiaľ možno rádovo odstupňovaná od fitness vyjadrujúcej funkčnosť. Ďalej je tiež ešte možné uvažovať minimalizáciu obvodu z pohľadu zabranej plochy na čipe. Nápodobne sa môže použiť princíp s odstupňovaním pomocou váhového koeficientu tak, aby bolo možné odlíšiť schopnosť samočinného testovania od zaberanej plochy. V prípade použitia v programe sa zameriame na prioritizáciu schopností samočinnej kontroly pred spotrebou plochy (pretože cieľom je získať obvody s čo najlepšími schopnosťami samočinnej kontroly). Avšak obecne v použití v iných prípadoch úplné splnenie jedného z kritérií nepodmieňuje druhé. Pri výpočte spotrebovanej plochy je vhodné využiť detekciu nevyužitých hradiel, určiteľnú už vo fáze výpočtu miery samočinnej kontroly. Fitness hodnotu kandidátneho obvodu určíme nasledovným spôsobom:

1. $\boldsymbol{fitnessV1 = fitness_{mod1} + fitness_{mod2}}$

2. Ak($\boldsymbol{fitnessV1}$ je rovná $\boldsymbol{fitness_{MAX\_funkcnost}}$) pokračuj krokom 3., inak ukonči ohodnocovanie

3. $\boldsymbol{fitnessV1 = fitnessV1 + k_2 * self\text{–}checking}$

4. $\boldsymbol{fitnessV1 = fitnessV1 + k_3 * usetrena\_plocha}$

Hodnoty koeficientov $\boldsymbol{k_1}$, $\boldsymbol{k_2}$ a $\boldsymbol{k_3}$ by mali byť volené podľa charakteru riešeného problému. Do praktickej časti som zvolil prístup nastaviť koeficienty podľa odhadu a potom upravovať podľa úspešnosti evolúcie. Nastavenie týchto koeficientov bude jedným z experimentov vykonaných na implementovanom návrhovom systéme.

6.1.2 Fitness funkcia s pomerným váhovaním funkčnosti a miery samočinnej kontroly

Ohodnotenie schopnosti samočinnej kontroly jedinca si vyžaduje náročné výpočetné operácie. Komplexnosť tohto procesu možno znížiť určením nevyužitých blokov. Ich vynechaním v procese evaluácie miery samočinnej kontroly môžeme ušetriť časť výpočetného času. Avšak detekcia nevyužitých blokov sama o sebe zaberie istú nezanedbateľnú dobu. Je na

zváženie, či pri danej veľkosti obvodov použiť techniku s vynechávaním nevyužitých blokov alebo nie. Druhý uvažovaný spôsob určenia fitness hodnoty kandidátneho obvodu je nasledovný:

1. $\boldsymbol{fitnessV2 = k_1 * (fitness_{mod1} + fitness_{mod2})}$
2. $\boldsymbol{fitnessV2 = fitnessV2 + k_2 * self\text{–}checking}$
3. $\boldsymbol{fitnessV2 = fitnessV2 + k_3 * usetrena\_plocha}$

Súčet váhových koeficientov položíme rovný jednej. Je nanajvýš vhodné, aby boli koeficienty $\boldsymbol{k_1}$ a $\boldsymbol{k_2}$ nenulové, pretože v opačnom prípade by sa úplne potlačila funkčnosť alebo miera schopností samočinného testovania hľadaného riešenia. Koeficient $\boldsymbol{k_3}$ by mal byť nenulový jedine v prípade, že chceme riešenie optimalizovať aj na minimalizáciu spotreby plochy na čipe.

6.2 Využitie paralelnej simulácie

Táto technika slúži na isté urýchlenie vyhodnotenia výstupov obvodu oproti naivnému prístupu. Podrobnejšie je popísaná v časti 3.8. Ako programovací jazyk bolo zvolené C/C++ a v programe použijeme paralelnú simuláciu na bitových logických operáciách celých čísel typu integer (32 bitov).

6.3 Formát pre vstupy a výstupy

Kľúčovou požiadavkou na návrhový systém bolo, aby bol optimalizovaný na rýchlosť a spustiteľný na počítačoch s operačným systémom Microsoft Windows XP, Linux alebo prípadne FreeBSD. Ako programovací jazyk bolo zvolené C/C++. Pre dosiahnutie maximálnej efektivity bol navrhnutý návrh výpočetného systému bez grafického užívateľského rozhrania. Vstupy pre návrhovú aplikáciu sa zadávajú pomocou pravdivostnej tabuľky s kompletným popisom funkcie obvodu.

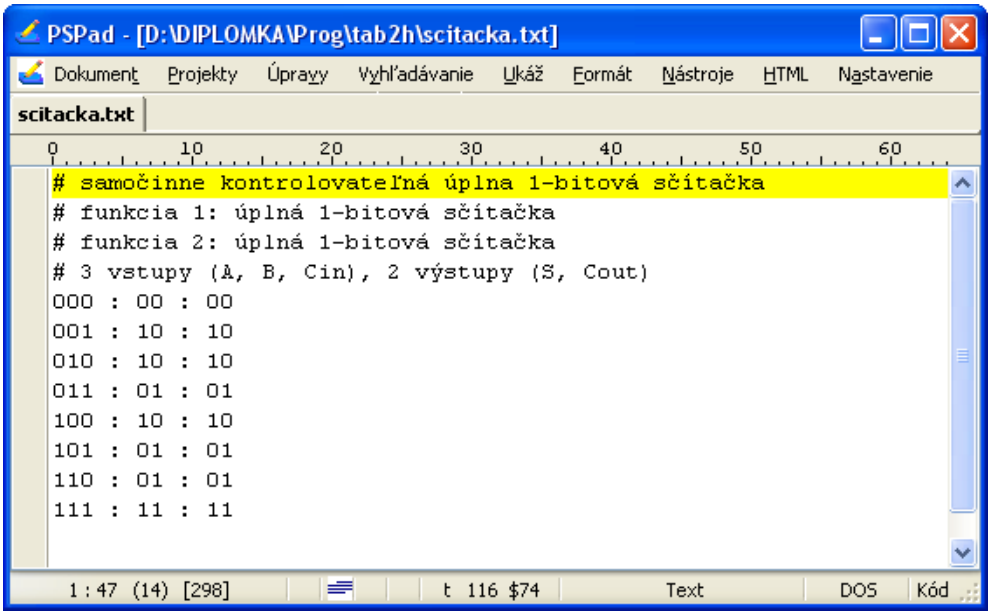
6.3.1 Vstupný formát pre návrhový systém

Pravdivostná tabuľka musí byť vytvorená podľa nasledovného modelu:

 # komentár - riadok začína maltským krížom
 # funkcia: špecifikácia výstupnej funkcie 1 - napr. úplná sčítačka
 # očakávaný počet primárnych vstupov I, očakávaný počet primárnych výstupov O
binárna vstupná kombinácia 1 : požadované výstupy v móde1 : požadované výstupy v móde2
binárna vstupná kombinácia 2 : požadované výstupy v móde1 : požadované výstupy v móde2
. . .
binárna vstupná kombinácia n : požadované výstupy v móde1 : požadované výstupy v móde2

Binárna kombinácia musí svojou dĺžkou presne odpovedať požadovanému bitovému rozsahu. Nie je možné do tabuľky vkladať neurčené hodnoty[1]. Príklad pravdivostnej tabuľky pre úplnú 1-bitovú sčítačku sa nachádza na obrázku 6.1:

[1] Z anglického jazyka sú tieto hodnoty známe ako DON'T CARE. Bývajú označované symbolom 'X' a symbolizujú, že zadaný výstup môže nadobúdať ľubovoľných hodnôt pre danú vstupnú kombináciu.



Obrázek 6.1: Pravdivostná tabuľka pre úplnú 1-bitovú sčítačku

Pre načítanie vstupných dát do aplikácie je vhodné, aby bola pravdivostná tabuľka transformovaná do ortogonálne preklopenej podoby, čím dosiahneme efektívnejšie načítanie hodnôt zo vstupného súboru do poľa čísel integer, nad ktorým potom budú vykonávané výpočty ohodnotenia obvodu. Pre účely tohoto prevodu bude vytvorená jednoduchá aplikácia prevádzajúca pravdivostnú tabuľku charakterizujúcu riešený problém do hlavičkového súboru, ktorý bude slúžiť ako vstup pre hlavnú časť výpočetného systému. V hlavičkovom súbore budú zadané informácie o šírke dátových vstupov a výstupov a o ich požadovaných hodnotách.

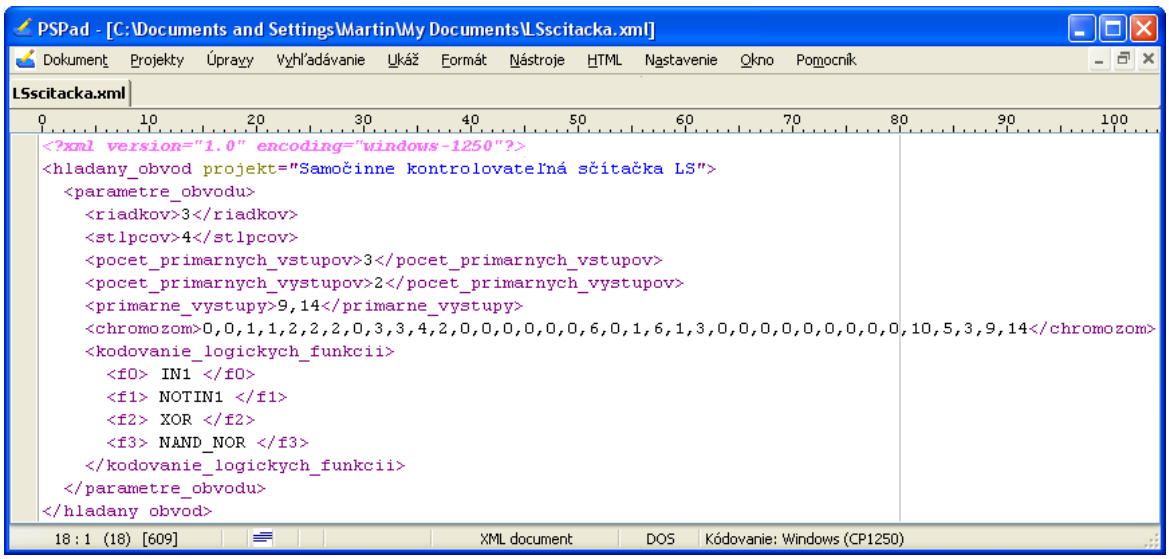
6.3.2 Výstupný formát pre navrhnuté obvody

Pre predávanie výstupov z výpočetnej aplikácie do aplikácie analyzujúcej a simulujúcej navrhované obvody bolo navrhnuté použitie XML. Súbor používa nasledovnú štruktúru:

<?xml version="1.0" encoding="typ kódovania"?>
<hladany_obvod projekt="Nepovinný názov projektu">
  <parametre_obvodu>
    <riadkov> počet riadkov v mriežke CGP </riadkov>
    <stlpcov> počet stĺpcov v mriežke CGP </stlpcov>
    <pocet_primarnych_vstupov> ich počet </pocet_primarnych_vstupov>
    <pocet_primarnych_vystupov> ich počet </pocet_primarnych_vystupov>
    <primarne_vystupy> čísla log. blokov primárnych výstupov </primarne_vystupy>
    <chromozom> hodnota chromozómu CGP pre navrhnuté riešenie </chromozom>
    <kodovanie_logickych_funkcii>
      <f0> názov logickej funkcie kódovanej v aplikácii pod 0 </f0>
      <f1> názov logickej funkcie kódovanej v aplikácii pod 1 </f1>
      <f2> pre polymorfné hradlá sa v názve použije znak '_' </f2>

...
<fn> názov logickej funkcie kódovanej v aplikácii pod n </fn>
</kodovanie_logickych_funkcii>
</parametre_obvodu>
</hladany_obvod>

Daný typ výstupu bude generovať výpočetná aplikácia. V značke <primarne_vystupy> sa musí nachádzať presne toľko celých čísel oddelených čiarkou, koľko je primárnych výstupov z obvodu. Obsah značky by sa mal zhodovať so záverečnou časťou chromozómu. Kódovanie funkcií je dôležité kvôli tomu, aby simulačný nástroj dokázal správne interpretovať navrhnuté riešenie zakódované v chromozóme. Príklad zakódovania navrhnutého obvodu - sčítačky z [22], je na obrázku 6.2.



Obrázek 6.2: Výstup zakódovania navrhovaného riešenia z [22] do navrhnutého formátu xml

6.4 Záznamy z evolúcie

Aby mohol užívateľ sledovať priebeh evolúcie, musia byť periodicky vykonávané záznamy jej aktuálneho stavu. Pre prehľadnosť postačuje upozorniť na zmenu pri každom zlepšení fitness hodnoty najlepšieho jedinca. Záznamy z priebehu evolúcie sú ukladané do formátu *.xml*. Každý chromozóm spĺňujúci požadovanú funkčnosť môže byť tiež zaznamenaný, aby bolo možné detailne analyzovať jeho schopnosti samočinnej kontroly.

Kapitola 7

Experimentálne výsledky

Hlavnou náplňou diplomovej práce bolo pokúsiť sa evolučným návrhom získať samočinne testovateľné polymorfné obvody s charakteristikami podobnými najlepším publikovaným riešeniam alebo prípadne s ešte lepšími vlastnosťami. S vytvoreným návrhovým prostredím bolo vykonané množstvo testov a analýz. Najskôr boli hľadané optimálne nastavenia algoritmu kartézskeho genetického programovania pre obecné samočinne testovateľné polymorfné obvody s rovnakou výstupnou funkciou v obidvoch módoch. Potom som tieto nastavenia použil ako východzí bod pre hľadanie zaujímavých riešení k zvoleným referenčným obvodom.

7.1 Experiment č. 1. Optimálny počet jedincov v populácii

Najprv som sa zameral na určenie optimálneho počtu jedincov v populácii. Sledoval som, aký dopad má zmena počtu jedincov na úspešnosť evolúcie. Vychádzajúc z teórie kartézskeho genetického programovania, kde je typické použitie veľmi malého počtu jedincov som stanovil hodnotu 5 jedincov ako východzie nastavenie pre prvý beh testov. Nastavenia ďalších parametrov možno nájsť v tabuľke 7.1.

Tabulka 7.1: Parametre experimentu určenia optimálneho počtu jedincov v populácii

Parameter	**Hodnota**
Počet riadkov v mriežke CGP	5
Počet stĺpcov v mriežke CGP	7
Počet evaluácií	3000000
L-back	3
Spôsob mutácie	štandardná CGP mutácia (tj. vždy maximálne 3 génov)
Množina logických funkcií	AND, OR, NOT, XOR, NAND, NOR, AND_OR, NAND_NOR, XOR_NOR, NAND_XOR

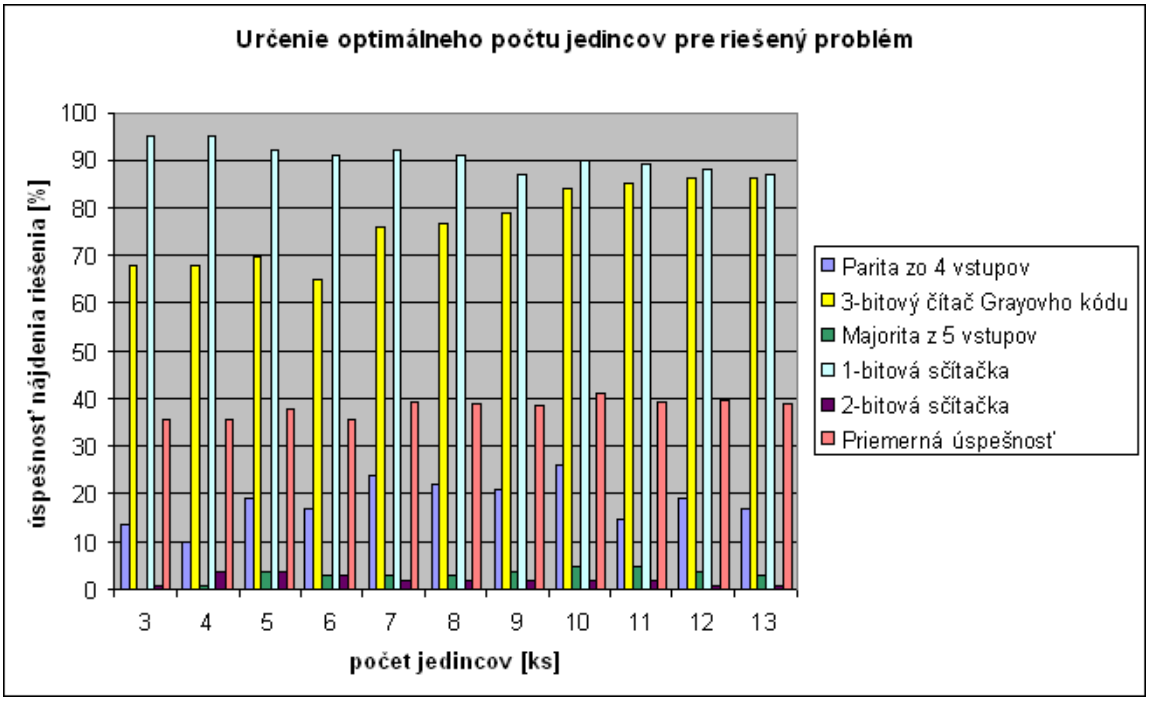
Bolo spustených 100 behov algoritmu kartézskeho genetického programovania s nastaveniami uvedenými v tabuľke 7.1. Riešené boli návrhy samočinne kontrolovateľných čítačov, obvodov parity, majority a sčítačiek, aby bolo určenie hodnôt parametrov čo najmenej ovplyvnené typom hľadaného obvodu. Počet potomkov bol postupne menený v rozsahu od

3 do 13 jedincov. Za úspešné riešenie bolo považované každé riešenie splňujúce funkčné požiadavky podľa zadanej pravdivostnej tabuľky, ktoré má schopnosti detekcie chýb „trvalej“ 0 a „trvalej 1“ na všetkých svojich hradlách, ktoré neslúžia ako primárne výstupy.

Tabulka 7.2: Výsledky určenia vhodného počtu jedincov pre zvolený typ problému

Typ problému	**Úspešnosť uvedeného počtu jedincov**										
	3	4	5	6	7	8	9	10	11	12	13
párna parita zo 4 vstupov	14	10	19	17	24	22	21	26	15	19	17
3-bitový čítač Grayovho kódu	68	68	70	65	76	77	79	84	85	86	86
majorita z 5 vstupov	0	1	4	3	3	3	4	5	5	4	3
1-bitová sčítačka	95	95	92	91	92	91	87	90	89	88	87
2-bitová sčítačka	1	4	4	3	2	2	2	2	2	1	1
priemerne	36	36	38	36	39	39	39	41	39	40	39

Záverom pre výber vhodného počtu jedincov v populácii bola označená hodnota 5. Ako z grafu na obrázku 7.1 vidno, má pomerne zaujímavú úspešnosť a celkom prijateľnú dobu výpočtu.



Obrázek 7.1: Graf s výsledkami experimentu určenia optimálneho počtu jedincov v populácii

Uvedené výsledky boli dosiahnuté pomocou ohodnocovania kandidátnych riešení fitness funkciou popisovanou v časti 6.1.1. Koeficienty pre tento experiment boli nastavené na $\boldsymbol{k_2 = 100}$ a $\boldsymbol{k_3 = 1}$. Fitness funkcia z časti 6.1.2 dosiahla výrazne horšie výsledky.

7.2 Experiment č. 2. Optimálny rozmer matice logických blokov CGP

Dôležitým faktorom vplývajúcim na úspešnosť nájdenia dostatočne kvalitného riešenia je aj istá pomoc evolúcii vložením informácií o rozmere riešeného problému. Návrhár číslicových obvodov môže na základe požadovanej charakteristiky obvodu pomerne dobre odhadnúť počet hradiel obvodu, ktorý vytvára. Ak je rozmer zadanej matice logických blokov príliš malý, potom evolúcia pre zložitejší problém nie je schopná so zadaným počtom logických blokov nájsť riešenie s požadovanou funkciou. Ak je však veľkosť prehľadávanej oblasti príliš veľká, potom zas evolúcia nemusí byť v zadanom počte generácií schopná nájsť požadované riešenie, kvôli nadmerne veľkému prehľadávanému stavovému priestoru. Vhodné obmedzenie veľkosti prehľadávanej oblasti pomôže algoritmu CGP k rýchlejšiemu nájdeniu prijateľného riešenia. Cieľom experimentu bolo rámcovo stanoviť vhodnú veľkosť matice logických blokov pre návrh jednoduchých samočinne kontrolovateľných obvodov (viď tabuľka 7.4). Zoznam parametrov experimentu možno nájsť v tabuľke 7.3.

Tabulka 7.3: Parametre experimentu určenia optimálnej veľkosti mriežky logických blokov

Parameter	**Hodnota**
Počet jedincov v populácii	5
Počet generácií	500000
L-back	3
Spôsob mutácie	štandardná CGP mutácia
Množina logických funkcií	AND, OR, NOT, XOR, NAND, NOR, AND_OR, NAND_NOR, XOR_NOR, NAND_XOR

Bolo spustených 100 behov algoritmu kartézskeho genetického programovania s nastaveniami uvedenými v tabuľke 7.3. Boli skúšané nasledovné rozmery matice logických blokov:

- Malinká matica 3 riadky × 4 stĺpce,
- stredne veľká matica 4 riadky × 8 stĺpcov,
- malá matica 4 riadkov × 5 stĺpcov,
- stredne veľká matica 5 riadkov × 7 stĺpcov,
- stredne veľká matica 5 riadkov × 10 stĺpcov,
- veľká matica 8 riadkov × 8 stĺpcov a
- veľká matica 8 riadkov × 12 stĺpcov.

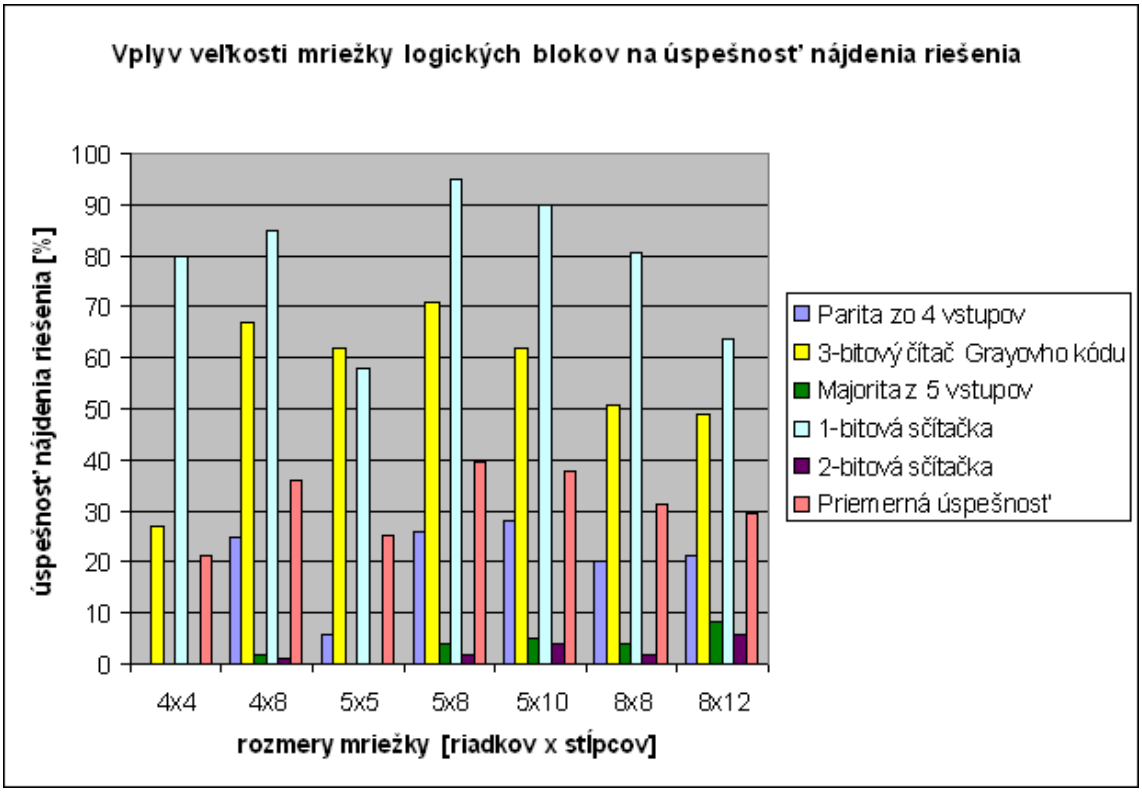
Výsledky experimentu sú zhrnuté v tabuľke 7.4 a zobrazené v grafe na obrázku 7.2.

Uvedené výsledky naznačujú, že pre každý problém je vhodné veľkosť matice prispôsobiť na mieru. Avšak všetky uvedené problémy spadajú do kategórie jednoduchých samočinne testovateľných polymorfných obvodov, preto keď vyslovíme všeobecný záver pre veľkosť mriežky CGP u tejto kategórie, ako univerzálny rozmer vychádza 5 riadkov × 8 stĺpcov.

Uvedené výsledky boli dosiahnuté pomocou ohodnocovania kandidátnych riešení fitness funkciou popisovanou v časti 6.1.1. Koeficienty pre tento experiment boli nastavené na $\boldsymbol{k_2 = 100}$ a $\boldsymbol{k_3 = 1}$. Fitness funkcia z časti 6.1.2 dosiahla opäť výrazne horšie výsledky.

Tabulka 7.4: Výsledky určenia vhodného rozmeru matice logických blokov

Typ problému	**Úspešnosť pre uvedenú veľkosť mriežky CGP**						
	4×4	**4×8**	**5×5**	**5×8**	**5×10**	**8×8**	**8×12**
párna parita zo 4 vstupov	0	25	6	26	28	20	21
3-bitový čítač Grayovho kódu	27	67	62	71	62	51	49
majorita z 5 vstupov	0	2	0	4	5	4	8
1-bitová sčítačka	80	85	58	95	90	81	64
2-bitová sčítačka	0	1	0	2	4	2	6
priemerne	21	36	25	40	38	32	30



Obrázek 7.2: Graf s výsledkami experimentu určenia optimálnej veľkosti mriežky CGP

7.3 Experiment č. 3. Určenie primeraného počtu generácií

Jedinou zastavovacou podmienkou pre algoritmus návrhu samočinne kontrolovateľných polymorfných obvodov je vyčerpanie stanoveného počtu generácií na evolúciu. Ak sa za túto dobu nepodarí nájsť riešenie disponujúce požadovanými vlastnosťami, potom sa daný beh evolúcie považuje za neúspešný. Určenie adekvátneho počtu generácií je jedným z kľúčových nastavení v algoritme CGP. Ak počet neodhadneme správne a je príliš malý, potom evolúcia dobehne rýchlo, ale má malý priestor na to, aby našla riešenie s kvalitnými parametrami. V prípade, že je počet generácií príliš vysoký, evolúcia môže uviaznuť v lokálnom extréme a zotrvať v tomto bode dlhý čas, čo nevedie k uspokojivým výsledkom a výrazne predlžuje dobu nájdenia riešenia s požadovanou kvalitou. Často používaným prístupom v takýchto prípadoch býva zastavenie evolúcie a jej opätovné spustenie odznova. Kartézske genetické programovanie je typické použitím veľmi vysokého počtu generácií pri evolúcii. Experimentálne bol skúmaný dopad zmeny počtu generácií na úspešnosť evolúcie. Nastavenia parametrov kartézskeho genetického programovania možno nájsť v tabuľke 7.5.

Tabulka 7.5: Parametre experimentu určenia optimálneho počtu generácií

Parameter	**Hodnota**
Počet riadkov v mriežke CGP	5
Počet stĺpcov v mriežke CGP	8
Počet jedincov v populácii	5
L-back	3
Spôsob mutácie	štandardná CGP mutácia
Množina logických funkcií	AND, OR, NOT, XOR, NAND, NOR, AND_OR, NAND_NOR, XOR_NOR, NAND_XOR

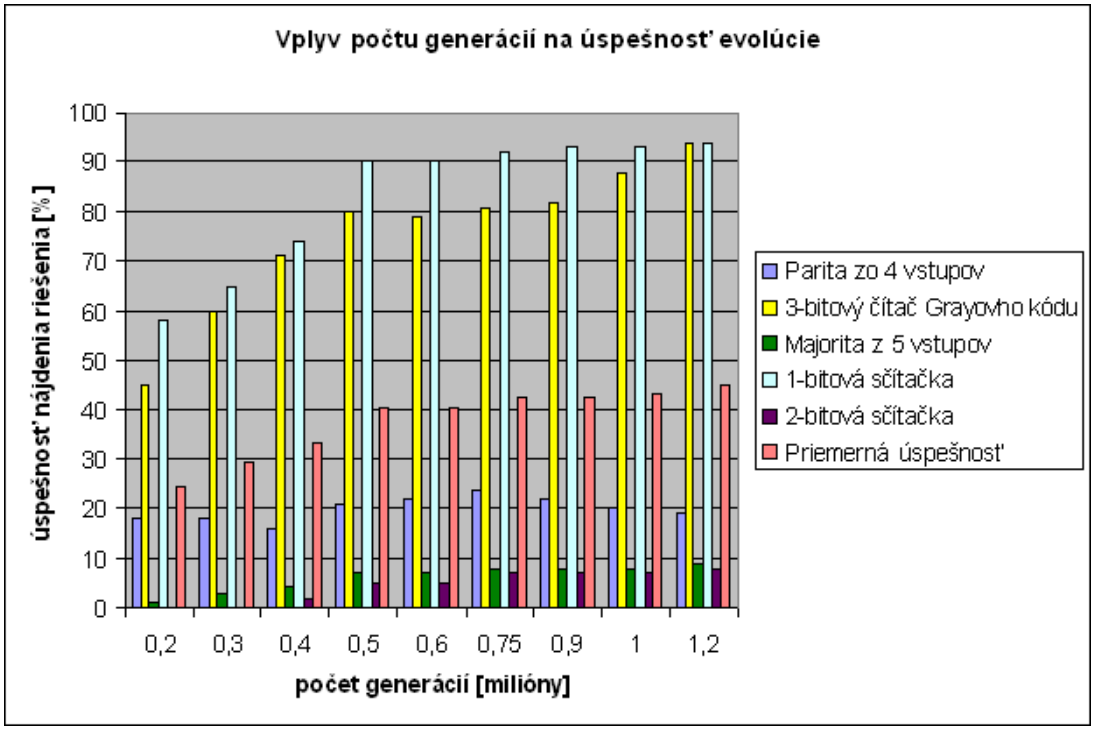
Bolo spustených 100 behov algoritmu kartézskeho genetického programovania s nastaveniami uvedenými v tabuľke 7.5. Riešené boli návrhy jednoduchých samočinne kontrolovateľných obvodov od 3 do 5 vstupov. Nastavenie počtu generácií je veľmi citlivá otázka a vyžaduje si individuálny prístup u každého hľadaného typu obvodu. Cieľom experimentu bolo univerzálne nastavenie tohto parametru CGP aspoň rádovo odhadnúť. Testované hodnoty a výsledky sú uvedené v tabuľke 7.6.

Tabulka 7.6: Výsledky určenia vhodného počtu generácií pre zvolený typ problému

Typ problému	**Úspešnosť pre počet generácií [milióny]**								
	0,2	0,3	0,4	0,5	0,6	7,5	0,9	1	1,2
párna parita zo 4 vstupov	18	18	16	21	22	24	22	20	19
3-bitový čítač Grayovho kódu	45	60	71	80	79	81	82	88	94
majorita z 5 vstupov	1	3	4	7	7	8	8	8	9
1-bitová sčítačka	58	65	74	90	90	92	93	93	94
2-bitová sčítačka	0	0	2	5	5	7	7	7	8
priemerne	24	29	33	41	41	42	42	43	45

Záverom pre výber vhodného počtu generácií pre zvolené problémy bola označená hod-

nota 750000. Ako z grafu na obrázku 7.3 vidno, má pomerne vysokú úspešnosť a akceptovateľnú dobu výpočtu. Hodnota 1200000 má síce vyššiu úspešnosť, ale rozdiel medzi zlepšením výsledkov a spotrebovanou dobou nie je vyhovujúci.



Obrázek 7.3: Graf s výsledkami experimentu určenia optimálneho počtu generácií

Uvedené výsledky boli dosiahnuté pomocou ohodnocovania kandidátnych riešení fitness funkciou popisovanou v časti 6.1.1. Koeficienty pre tento experiment boli nastavené na $k_2 = 100$ a $k_3 = 1$. Fitness funkcia z časti 6.1.2 dosiahla opäť výrazne horšie výsledky. Experimenty potvrdili trend zlepšovania úspešnosti evolúcie s rastúcim počtom generácií.

7.4 Experiment č. 4. Určenie nastavenia parametru L-back

Voľnosť prepojení v navrhovanom obvode limituje možnosti procesu evolúcie. Nastavenie parametru L-back sa volí v rozsahu od jednej do počtu stĺpcov matice logických blokov. Čím je jeho hodnota vyššia, tým má evolúcia viac možností na zostavenie prepojení v navrhovanom obvode a tým pádom väčšiu šancu na nájdenie nejakého zaujímavého inovatívneho riešenia. Nastavenie parametru L-back na hodnotu 1 má však tú výhodu, že v podstate umožňuje evolúcii konštruovať iba obvody s reťazeným spracovaním. Význam parametru L-back je podrobne popísaný v časti 3.2. V tomto experimente som sa pokúsil určiť vhodnú hodnotu L-back. Nastavenia ostatných parametrov kartézskeho genetického programovania možno nájsť v tabuľke 7.7.

Bolo spustených 100 behov algoritmu kartézskeho genetického programovania s nastaveniami uvedenými v tabuľke 7.7. Riešené boli návrhy jednoduchých samočinne kontrolovateľných obvodov od 3 do 5 vstupov. Nastavovanie parametru L-back v experimentoch bolo vykonávané podľa hodnôt z tabuľky 7.8.

Tabulka 7.7: Parametre experimentu nastavovania hodnoty L-back

Parameter	**Hodnota**
Počet riadkov v mriežke CGP	5
Počet stĺpcov v mriežke CGP	8
Počet jedincov v populácii	5
Počet generácií	750000
Spôsob mutácie	štandardná CGP mutácia
Množina logických funkcií	AND, OR, NOT, XOR, NAND, NOR, AND_OR, NAND_NOR, XOR_NOR, NAND_XOR

Tabulka 7.8: Výsledky určenia vhodného nastavenia parametru L-back

Typ problému	**Úspešnosť hodnoty L-back**						
	1	2	3	4	5	6	8
párna parita zo 4 vstupov	13	14	16	18	12	10	7
3-bitový čítač Grayovho kódu	80	84	82	76	71	68	66
majorita z 5 vstupov	4	5	6	3	2	2	3
1-bitová sčítačka	85	81	80	71	64	60	49
2-bitová sčítačka	3	4	5	4	4	1	6
priemerne	37	38	38	34	31	28	26

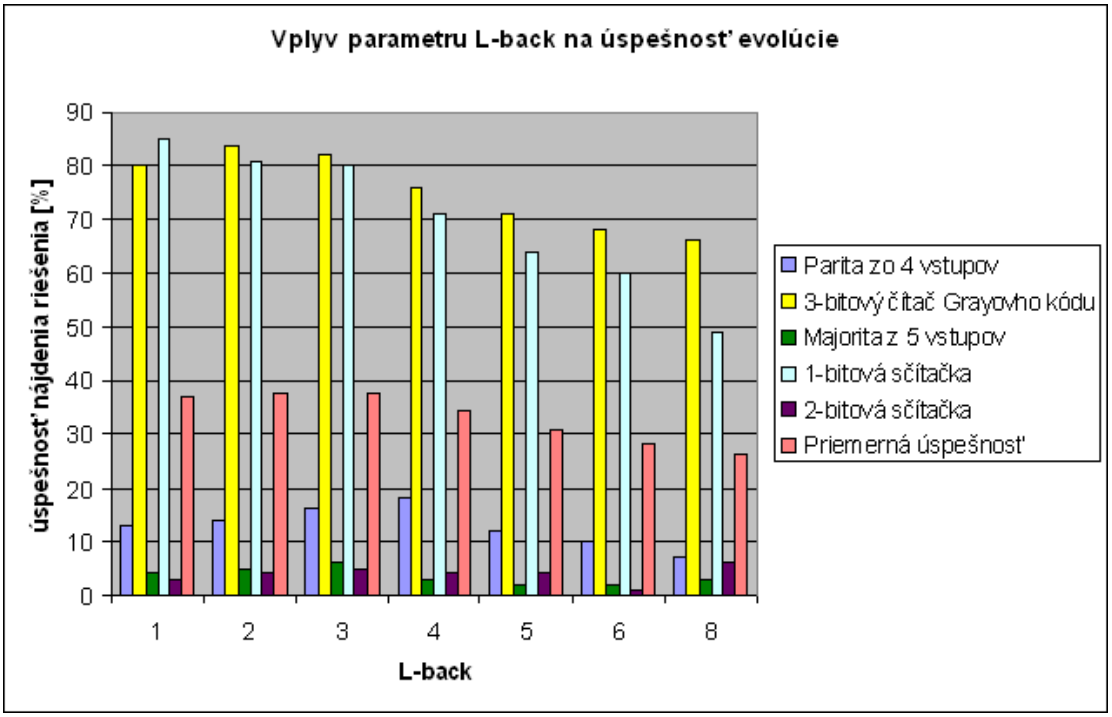
Z experimentálnych výsledkov som vyvodil záver, že najlepšie nastavenie parametru L-back je hodnotu 3. Ak striktne chceme aby, bol navrhnutý obvod reťazený potom musíme nastaviť hodnotu L-back parametru na 1 a evolúcia bude navrhovať iba obvody s reťazenou štruktúrou. Výsledky experimentu sú znázornené v grafe na obrázku 7.4.

Uvedené výsledky boli dosiahnuté pomocou ohodnocovania kandidátnych riešení fitness funkciou popisovanou v časti 6.1.1. Koeficienty pre tento experiment boli nastavené na $\boldsymbol{k_2 = 100}$ a $\boldsymbol{k_3 = 1}$. Fitness funkcia z časti 6.1.2 dosiahla horšie výsledky.

7.5 Experiment č. 5. Určenie množín polymorfných hradiel vhodných pre riešenie špecifických úloh

Polymorfné hradlá boli rozdelené do 8 skupín:

1. NAND/NOR, NAND/XOR, XOR/NOR.
2. AND/OR, NAND/XOR.
3. AND/OR, XOR/NOR.
4. NAND/NOR, AND/XOR.
5. AND/OR, NAND/XOR.
6. XOR/NOT, AND/IN1.



Obrázek 7.4: Graf s výsledkami experimentov s nastavením parametru L-back

7. AND/XOR, NAND/OR.

8. NOT/NAND, AND/XOR.

V aplikácii boli nasimulované niektoré už existujúce polymorfné hradlá, ale aj polymorfné hradlá zložené z náhodných logických funkcií. Jednotlivé skupiny hradiel boli kombinované s konvenčnými hradlami z množiny {AND, OR, XOR, NOT, NAND, NOR, NXOR}. Nastavenia parametrov kartézskeho genetického programovania možno nájsť v tabuľke 7.9.

Tabulka 7.9: Parametre experimentu určenia vhodných typov hradiel pre riešené problémy

Parameter	**Hodnota**
Počet riadkov v mriežke CGP	5
Počet stĺpcov v mriežke CGP	8
Počet jedincov v populácii	5
L-back	3
Počet generácií	300000
Spôsob mutácie	štandardná CGP mutácia
Množina konvenčných log. funkcií	AND, OR, XOR, NOT, NAND, NOR, NXOR

Bolo spustených 50 behov algoritmu kartézskeho genetického programovania s nastaveniami uvedenými v tabuľke 7.9. Riešené boli návrhy samočinne kontrolovateľných čítačov, obvodov parity, majority a sčítačiek. Výsledky experimentu sú zhrnuté v tabuľke 7.10.

Tabulka 7.10: Vhodnosť použitia polymorfných hradiel pre riešenie daného typu problému

Typ problému	**Výhodné použitie polymorfných hradiel**
párna parita zo 4 vstupov	NAND/NOR, XOR/NOR
3-bitový čítač Grayovho kódu	AND/OR, XOR/NOR
majorita z 5 vstupov	NAND/XOR,
1-bitová sčítačka	NAND/NOR
2-bitová sčítačka	NAND/NOR, NOT/NAND, AND/XOR

Pre tento typ experimentu bolo použité ohodnocovanie kandidátnych riešení jedine pomocou fitness funkcie popisovanej v časti 6.1.1. Koeficienty pre tento experiment boli nastavené na $\boldsymbol{k_2 = 100}$ a $\boldsymbol{k_3 = 1}$. Experimentálne zisteným záverom je, že použitie špecifického typu polymorfného hradla pre obvod konkrétneho určenia (napr. parity) nemalo veľký vplyv na úspešnosť riešenia, pretože nedostatok logickej funkcie polymorfných hradiel sa dorovnal funkciami množiny konvenčných logických hradiel. V tabuľke 7.10 sú polymorfné hradlá so štatisticky najväčším uplatnením, ale neznamená to, že iné typy polymorfných hradiel ako sú k danému problému uvedené, majú výrazne menšiu použiteľnosť.

7.6 Experiment č. 6. Určenie typu mutácie

Genetický operátor mutácie je u kartézskeho genetického programovania jediným prostriedkom tvorby nových jedincov z rodičovskej populácie (1 jedinec). Existujú rôzne spôsoby ako uskutočniť mutáciu chromozómu CGP tak, aby sa vytvoril nový jedinec. Štandardný postup je stanovenie maximálneho množstva mutovaných génov a mutovať náhodne vyberané gény z rozsahu nula až maximálna hranica. Ďalšou variantou je stanovenie pravdepodobnosti mutácie s ktorou sa pokúsime pozmeniť gény chromozómu na nové (prípustné) hodnoty. Taktiež často používaným spôsobom býva mutácia pevne zvoleného počtu náhodne vyberaných génov. Z uvedených možností mutácie som skúsil určiť najvhodnejší typ mutácie pre kategóriu jednoduchých polymorfných samočinne kontrolovaných obvodov. Nastavenia parametrov kartézskeho genetického programovania pre tento experiment možno nájsť v tabuľke 7.11.

Tabulka 7.11: Parametre experimentu určenia vhodného spôsobu mutácie

Parameter	**Hodnota**
Počet riadkov v mriežke CGP	5
Počet stĺpcov v mriežke CGP	8
Počet jedincov v populácii	5
L-back	3
Počet generácií	1000000
Množina logických funkcií	AND, OR, NOT, XOR, NAND, NOR, AND_OR, NAND_NOR, XOR_NOR, NAND_XOR

Bolo spustených 100 behov algoritmu kartézskeho genetického programovania s nasta-

veniami uvedenými v tabuľke 7.11. Riešené boli návrhy jednoduchých samočinne kontrolovateľných obvodov od 3 do 5 vstupov. Testované boli nasledovné spôsoby mutácie:

1.) Štandardná mutácia CGP s maximálnym počtom mutovaných génov 4,

2.) štandardná mutácia CGP s maximálnym počtom mutovaných génov 7,

3.) mutácia všetkých génov chromozómu s pravdepodobnosťou 15%,

4.) mutácia všetkých génov chromozómu s pravdepodobnosťou 25%,

5.) mutácia všetkých génov chromozómu s pravdepodobnosťou 35%,

6.) mutácia troch náhodne vybraných génov chromozómu,

7.) a mutácia piatich náhodne vybraných génov chromozómu.

Výsledky experimentu sú uvedené v tabuľke 7.12.

Tabulka 7.12: Výsledky určenia vhodného typu mutácie

Typ problému	**Úspešnosť evolúcie pre zvolený typ mutácie**						
	1. typ	2. typ	3. typ	4. typ	5. typ	6. typ	7. typ
párna parita zo 4 vstupov	27	19	23	9	3	18	9
3-bitový čítač Grayovho kódu	76	36	68	16	7	62	49
majorita z 5 vstupov	7	1	6	2	0	3	8
1-bitová sčítačka	89	39	72	24	4	70	55
2-bitová sčítačka	7	8	8	2	1	2	8
priemerne	41	21	35	11	3	31	26

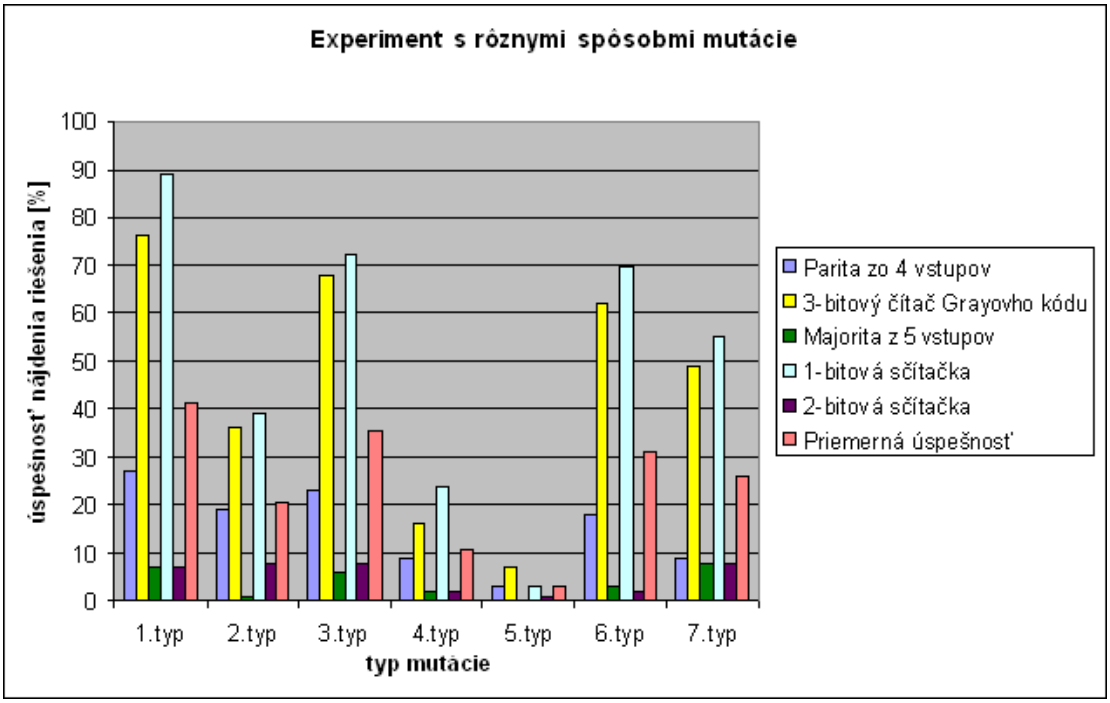
Uvedené výsledky boli zanesené to grafu (viď obrázok 7.5). Na vzorke jednoduchých polymorfných obvodov sa najviac osvedčili štandardná mutácia s hornou hranicou 4 jedincov a mutácia s pravdepodobnosťou 15%.

Uvedené výsledky boli dosiahnuté pomocou ohodnocovania kandidátnych riešení fitness funkciou popisovanou v časti 6.1.1. Koeficienty pre tento experiment boli nastavené na $\boldsymbol{k_2 = 100}$ a $\boldsymbol{k_3 = 1}$. Fitness funkcia z časti 6.1.2 dosiahla opäť výrazne horšie výsledky. Experimenty potvrdili, že voľba spôsobu mutácie môže značne ovplyvniť priebeh evolúcie.

7.7 Experiment č. 7. Štruktúra fitness funkcie

Všetky doposiaľ uvedené experimenty boli vykonávané nad dvomi typmi fitness funkcie. Prvý typ, váhovaná fitness funkcia s prioritizáciou funkčnosti riešenia, bola vo všetkých uskutočnených testoch lepšia než varianta s váhovaním rozdeleným medzi funkčnosť, schopnosti samočinnej kontroly a spotrebu plochy. Koeficienty boli počas experimentovania nastavené nasledovne:

- 1. typ: $\boldsymbol{k_2 = 100}$ a $\boldsymbol{k_3 = 1}$,
- 2. typ: $\boldsymbol{k_1 = 0,75}$, $\boldsymbol{k_2 = 0,2}$ a $\boldsymbol{k_3 = 0,05}$.



Obrázek 7.5: Graf s výsledkami experimentálneho určenia spôsobu mutácie

Koeficient k_1 určuje váhu splnenia požadovanej funkčnosti, koeficient $k_2 = 0,2$ určuje váhu schopností samočinnej kontroly a $k_3 = 0,05$ slúži ako váhový koeficient zabranej plochy na čipe. Keďže výsledky s 2. typom fitness funkcie nedopadli uspokojivo, rozhodol som sa zaexperimentovať s váhami koeficientov. Vyskúšané varianty sú uvedené v tabuľke 7.14. Nastavenia parametrov CGP pre tento experiment možno nájsť v tabuľke 7.13.

Tabulka 7.13: Parametre experimentu určenia vhodného spôsobu fitness funkcie

Parameter	**Hodnota**
Počet riadkov v mriežke CGP	5
Počet stĺpcov v mriežke CGP	8
Počet jedincov v populácii	5
Počet generácií	2000000
L-back	3
Spôsob mutácie	štandardná CGP mutácia
Množina logických funkcií	AND, OR, NOT, XOR, NAND, NOR, AND_OR, NAND_NOR, XOR_NOR, NAND_XOR

Z výsledkov jasne vyplýva, že na riešenie úloh návrhov polymorfných samočinne kontrolovaných obvodov sa použitím fitness funkcie s prioritizáciou funkčnosti obvodu, dosiahne oveľa kvalitnejších výsledkov ako aplikovaním druhého navrhovaného spôsobu. Ďalej je možné pozorovať istý nárast v úspešnosti evolúcie, ak pri fitness funkcii s rozdeleným váhovaním zvyšujeme váhu pre funkčnosť riešenia. Avšak tento spôsob hodnotenia kan-

Tabulka 7.14: Výsledky určenia vhodného typu fitness funkcie

Typ problému	**Úspešnosť evolúcie pre zvolený typ fitness**						
		1. typ	2. typ	2. typ	2. typ	2. typ	2. typ	2. typ
	k_1		0,8	0,8	0,85	0,9	0,95	0,99
	k_2	100	0,11	0,18	0,1	0,05	0,04	0,01
	k_3	1	0,09	0,02	0,05	0,05	0,01	0
párna parita zo 4 vstupov		31	0	0	1	1	2	2
3-bitový čítač Grayovho kódu		79	0	0	1	1	1	3
majorita z 5 vstupov		9	0	0	0	0	1	1
1-bitová sčítačka		85	0	1	1	1	1	2
2-bitová sčítačka		7	0	0	0	0	1	1
priemerne		42	0	0	1	1	1	2

didátnych riešení výrazne zaostáva za prioritne váhovanou fitness. Pri porovnávaní doby výpočtu pre dané dva typy fitness funkcie dosiahla fitness funkcia 2.typu horšie výsledky, pretože schopnosti samočinnej kontroly a veľkosť zaberanej plochy na čipe sa musia určovať pre každý kandidátny obvod a jedná sa o pomerne náročný výpočet. Výsledky experimentov názorne ilustruje graf na obrázku 7.6.

7.8 Zaujímavé nájdené riešenia

Po stanovení univerzálnych hodnôt parametrov som sa snažil ešte samostatne vyladiť algoritmus CGP pre konkrétny typ hľadaného obvodu. Skúšal som nájsť takú kombináciu parametrov kartézskeho genetického programovania, ktorá v ideálnom prípade dáva akceptovateľné riešenie v každom spustenom behu (úspešnosť 100%). Podarilo sa mi vytvoriť obrovské množstvo návrhov, ktoré spĺňajú kladené požiadavky. Najväčším problémom bolo z množstva návrhov vybrať tie, ktoré budú „najzaujímavejšie“.

7.8.1 Úplná jednobitová sčítačka

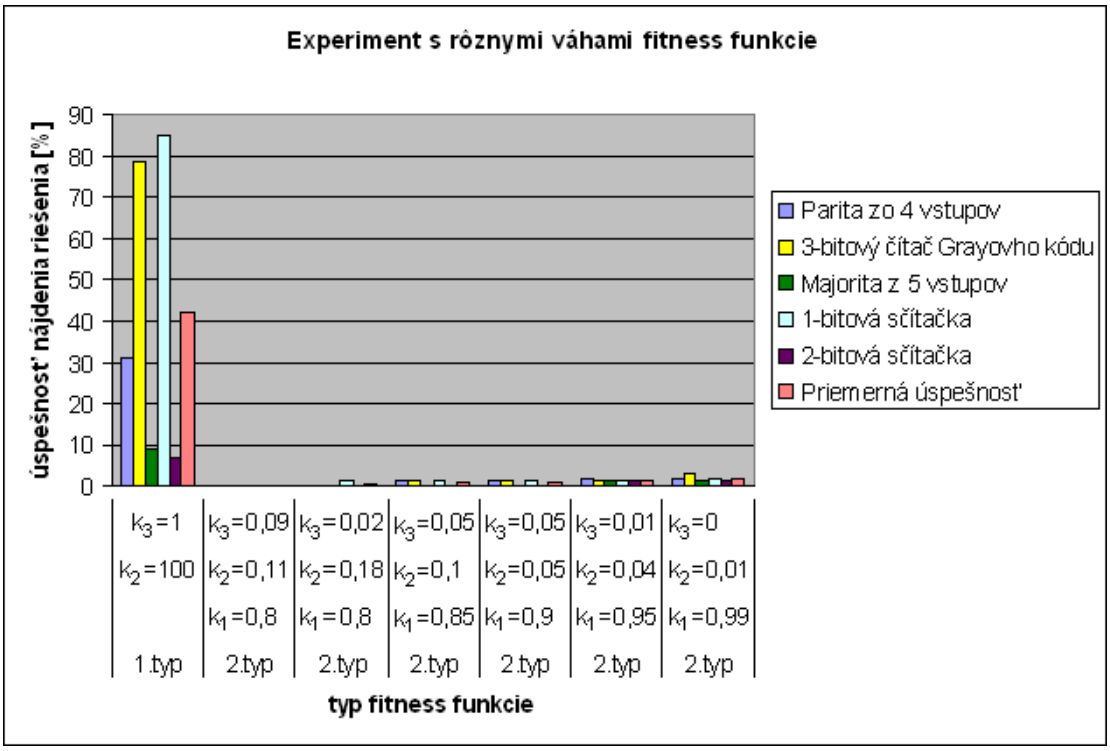
Úplná jednobitová sčítačka patrila pri experimentoch medzi najúspešnejšie riešené návrhy. Medzi množstvom riešení bola znovu nájdená obvodová realizácia sčítačky z [22] (viď sekcia 5.1.2). Ale tu si popíšeme iný dizajn.

Medzi zaujímavý príklad samočinne testovateľnej úplnej jednobitovej sčítačky možno zaradiť obvod z obrázku 7.7, ktorý sa skladá zo 7 hradiel.

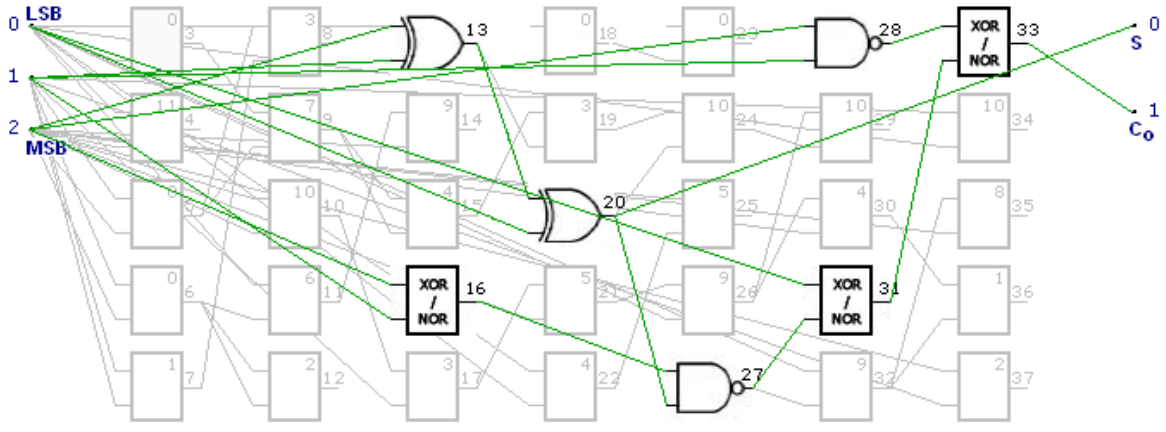
Skutočnosť, že daný obvod pracuje ako jednobitová úplná sčítačka možno najlepšie skontrolovať podľa tabuľky 7.15.

Ak si pozorne všimneme stĺpce v tabuľke 7.15, tak stĺpce g20 a g33 tvoria primárne výstupy pre sumu a prenos a ich hodnoty presne odpovedajú požadovaným výstupom úplnej jednobitovej sčítačky.

Zaujímavosťou navrhnutej sčítačky je to, že na svoju činnosť využíva iba logické funkcie XOR a NOR. Spomedzi polymorfných hradiel využíva iba typ XOR/NOR a je schopná zistiť chybu aj na svojom primárnom výstupe pre sumu (hradlo g20). Detekciu poruchy vykonáva na obidvoch svojich primárnych výstupoch.



Obrázek 7.6: Graf s výsledkami experimentálneho určenia vhodného typu fitness funkcie



Obrázek 7.7: Schéma navrhnutej sčítačky

Tabulka 7.15: Tabuľka vstupov a výstupov pre obvod z obrázku 7.7

Primárne vstupy	**Polymorfný mód**	**Výstupy hradla (matica 5 × 7)**						
		g13	g16	g20	g27	g28	g31	g33
(MSB) 000 (LSB)	XOR	0	0	0	1	1	1	0
	NOR	0	1	0	0	1	1	0
(MSB) 001 (LSB)	XOR	1	1	1	0	0	0	0
	NOR	1	0	1	0	0	1	0
(MSB) 010 (LSB)	XOR	1	1	1	0	0	0	0
	NOR	1	0	1	0	0	1	0
(MSB) 011 (LSB)	XOR	0	0	0	1	0	1	1
	NOR	0	0	0	1	0	0	1
(MSB) 100 (LSB)	XOR	0	0	1	0	1	1	0
	NOR	0	1	1	0	1	0	0
(MSB) 101 (LSB)	XOR	1	1	0	0	0	1	1
	NOR	1	0	0	1	0	0	1
(MSB) 110 (LSB)	XOR	1	1	0	0	0	1	1
	NOR	1	0	0	1	0	0	1
(MSB) 111 (LSB)	XOR	0	0	1	0	0	1	1
	NOR	0	0	1	0	0	0	1

Sčítačka bola analyzovaná vo vytvorenom simulátore a dosiahla výsledky zhrnuté v tabuľke na obrázku 7.8. Symbol 'X' znamená, že chyba zmieneného typu sa dá na danom hradle odhaliť horeuvedenou vstupnou kombináciou (vstupným vektorom).

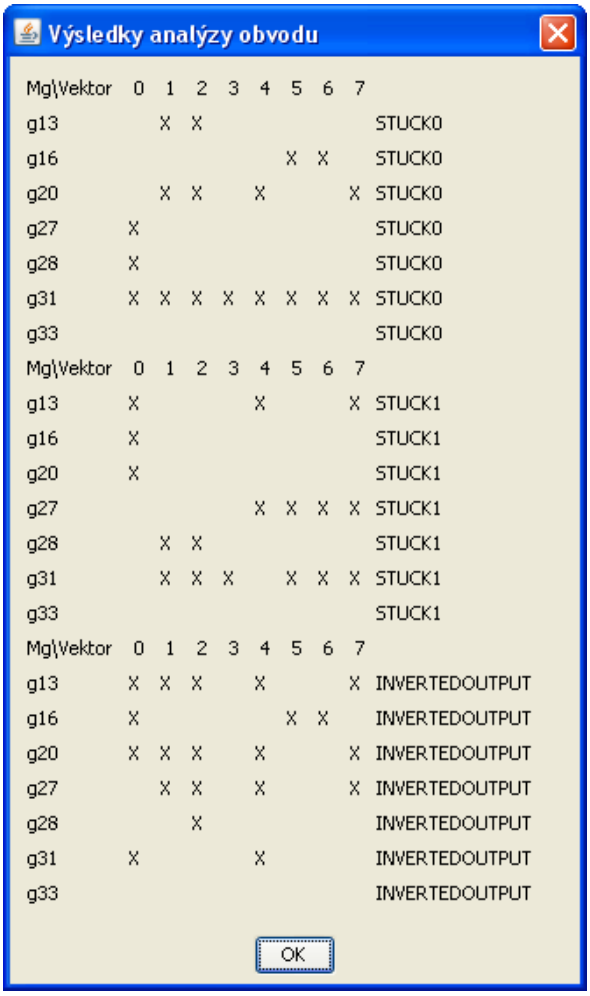
Pre návrh samočinne testovateľnej úplnej jednobitovej sčítačky sa mi podarilo nájsť takú kombináciu parametrov kartézskeho genetického programovania, ktorá dáva úspešnosť 99 % (viď tabuľka 7.16).

Tabulka 7.16: Najlepšia zistená zostava parametrov CGP pre návrh 1-bitovej úplnej sčítačky

Parameter	**Hodnota**
Počet riadkov v mriežke CGP	4
Počet stĺpcov v mriežke CGP	10
Počet jedincov v populácii	5
Počet generácií	600000
L-back	2
Spôsob mutácie	štandardná CGP mutácia
Množina logických funkcií	AND, OR, NOT, XOR, NAND, NOR, AND_OR, NAND_NOR, XOR_NOR, NAND_XOR

7.8.2 Majorita

Nájdenie obvodu samočinne kontrolovanej majority z 5 vstupov sa stalo najproblematickejšou úlohou pri experimentovaní. Nízka úspešnosť pri väčšine vykonaných testov bola



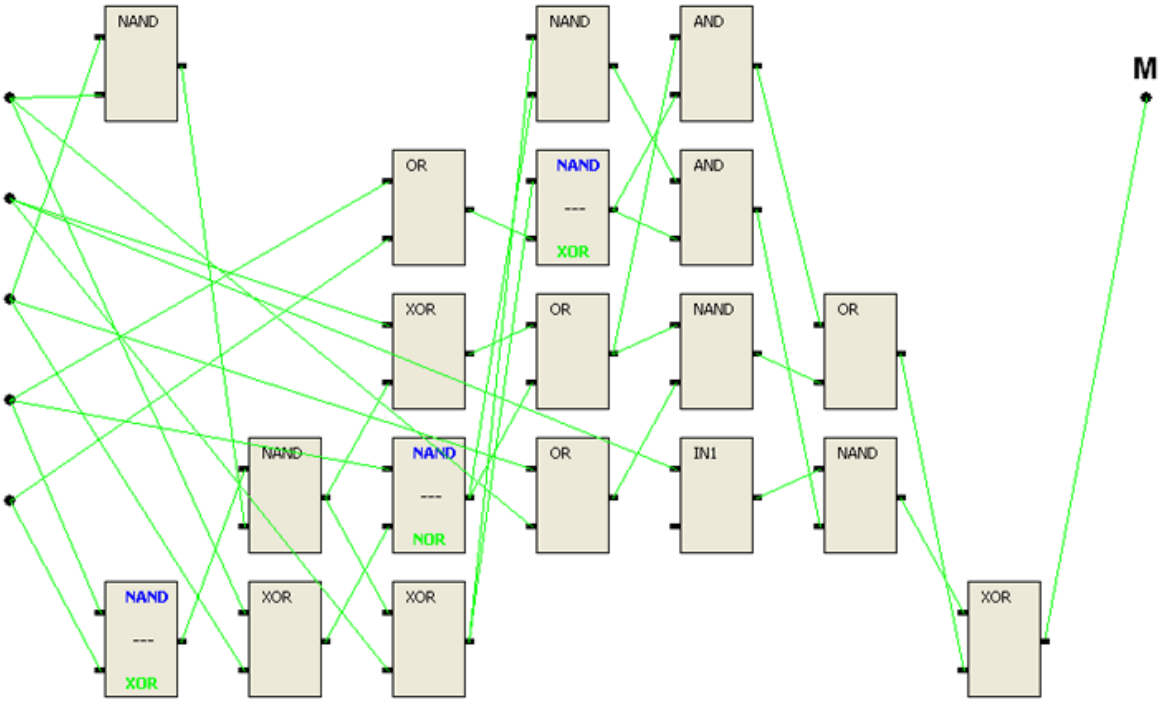
Obrázek 7.8: Výsledky analýzy z vytvoreného simulátoru pre navrhnutú sčítačku

spôsobená použitím malých rozmerov matice logických blokov. Po vyladení parametrov CGP pre tento špecifický problém (viď tabuľka 7.17) sa mi podarilo dosiahnuť úspešnosť 18 %.

Tabulka 7.17: Najlepšia zistená zostava parametrov CGP pre návrh majority z 5 vstupov

Parameter	**Hodnota**
Počet riadkov v mriežke CGP	10
Počet stĺpcov v mriežke CGP	14
Počet jedincov v populácii	11
Počet generácií	750000
L-back	3
Spôsob mutácie	4 náhodne zvolené gény
Množina logických funkcií	čo najviac rozmanitých logických funkcií

Príkladom samočinne kontrolovanej majority je obvod z obrázku 7.9. Testy preukázali, že takto navrhnutý obvod dokáže diagnostikovať chybu uviaznutia v trvalej 0 a v trvalej 1 na všetkých hradlách okrem primárneho výstupu. Chybu invertovaného výstupu obvod nedokáže detekovať len na jednom hradle NAND[1] a na primárnom výstupe. Na detekciu chýb obvod využíva dva typy polymorfných hradiel: NAND/NOR a NAND/XOR.



Obrázek 7.9: Schéma navrhnutej samočinne kontrolovanej majority z 5 vstupov

Obvod navrhovanej majority sa skladá z 19 hradiel, pričom hradlo IN1 môže byť realizované jediným vodičom a tak takýto obvod možno realizovať pomocou iba 18 fyzických

[1] Jedná sa o hradlo NAND ležiace v 1.riadku a 4.stĺpci (viď obrázok 7.9).

hradiel.

7.9 Časová náročnosť evolučného návrhu

Pre úplnosť uvediem ešte ako dlho trvá 100 behov evolúcie pre jednotlivé riešené problémy. Kvôli problémom s neexistenciou štandardných nastavení parametrov CGP bolo použité nastavenie podľa tabuľky 7.18.

Tabulka 7.18: Zostava parametrov CGP pre zistenie časovej náročnosti evolučného návrhu

Parameter	**Hodnota**
Počet riadkov v mriežke CGP	5
Počet stĺpcov v mriežke CGP	8
Počet jedincov v populácii	5
Počet generácií	750000
L-back	3
Fitness funkcia	1. typ fitness funkcie z časti 6.1.1
Spôsob mutácie	štandardná CGP mutácia
Množina logických funkcií	AND, OR, NOT, XOR, NAND, NOR, AND_OR, NAND_NOR, XOR_NOR, NAND_XOR

Výsledné časy z tabuľky 7.19 boli dosiahnuté na počítači s procesorom Intel Core 2 Duo E6750 (2,66 GHz) s operačnou pamäťou kapacity 2 GB a operačným systémom Microsoft Windows XP (SP2).

Tabulka 7.19: Časová náročnosť pre evolúciu zvolených typov obvodov

Typ problému	**Doba výpočtu 100 behov evolúcie**
párna parita zo 4 vstupov	1 h : 12 m : 02 s
3-bitový čítač Grayovho kódu	1 h : 48 m : 07 s
majorita z 5 vstupov	1 h : 15 m : 54 s
1-bitová sčítačka	1 h : 41 m : 08 s
2-bitová sčítačka	1 h : 16 m : 08 s

Výsledné časy sa môžu pre rovnakú konfiguráciu spustenú na tom istom počítači mierne líšiť v dôsledku nedeterministického nájdenia riešenia s požadovanou funkčnosťou. Pretože ohodnotenie riešenia s plnou funkčnosťou trvá dlhšie ako ohodnotenie riešenia s neúplnou funkčnosťou, môže pri 100 behoch odchýlka dosiahnuť okolo 1 minúty.

Kapitola 8

Nároky vytvoreného softvéru

V praktickej časti diplomovej práce bola vytvorená aplikácia slúžiaca pre návrh jednoduchých polymorfných samočinne kontrolovaných polymorfných obvodov. Jej výstupy sú vo formáte *.xml* prenositeľné do aplikácie umožňujúcej simuláciu a detailnú analýzu navrhnutých obvodov.

8.1 Návrhové prostredie

Pre implementáciu rýchleho a efektívneho návrhového prostredia bol použitý programovací jazyk C/C++. Pre preklad aplikácie je potrebný prekladač zdrojových kódov jazyka C, napríklad **gcc** pre Linux alebo **MinGW** pre MS Windows. K prekladu slúži príkaz **make** podľa priloženého súboru Makefile. Aplikácia vyžaduje zadanie riešeného problému vo forme pravdivostnej tabuľky. Po pretransformovaní pravdivostnej tabuľky na hlavičkový súbor pomocou príkazu **tab2h pravdivostnátabuľka.txt > pravdivostnátabuľka.h** vzniknutý hlavičkový súbor v súbore **cgp.h** prilinkujeme k aplikácii CGP. Evolúciu spustíme príkazom **cgp**. Minimálne požiadavky na spustenie evolúcie sú:

- Striktne 32-bitová architektúra,
- procesor Intel Celeron 750 MHz alebo lepší,
- aspoň 256 MB operačnej pamäte a
- aspoň 500 MB voľného miesta na disku, pre potreby tvorby záznamov.

8.2 Analyzátor samočinne testovateľných polymorfných obvodov

Pre implementáciu bol zvolený programovací jazyk JAVA, aby bola splnená požiadavka prenositeľnosti. Aplikácia bola vyvíjaná v prostredí NETBEANS IDE 6.0. Pre preklad a spustenie aplikácie je potrebný prekladač jazyka JAVA. Pre spustenie je potrebná JAVA verzie 5 alebo vyššia. Aplikáciu možno spustiť príkazom **java -jar ErrorCheckSimulatorV1.jar** alebo otvorením projektu ErrorCheckSimulatorV1 vo vývojovom prostredí NETBEANS IDE 6.0 (voľba RUN). Grafické užívateľské rozhranie dovoľuje načítať súbor s popisom obvodu, vykonať detekciu nevyužitých hradiel, analýzu schopností samočinnej kontroly, simulácie detekcie chýb a podobne. Pre demonštračné účely bola vytvorená užívateľská

príručka, ktorá detailne popisuje možnosti práce s analyzátorom. Táto príručka je súčasťou diplomovej práce a možno ju nájsť na priloženom CD v adresári **\prirucka**

Kapitola 9

Záver

V práci bol zhrnutý prehľad konvenčného spôsobu návrhu samočinne kontrolovateľných obvodov a načrtnuté trendy vývoja uberajúce sa cestou využívania evolučných techník pri návrhu. V praktickej časti bol zrealizovaný evolučný vývojový program schopný nájsť riešenia, ktoré sa zhodujú s doposiaľ najlepšími známymi riešeniami, prípadne riešenia porovnateľnej kvality. Bol vytvorený nástroj umožňujúci simuláciu a analýzu schopností samočinnej kontroly navrhnutých riešení. S vytvoreným systémom bolo vykonané rozsiahle množstvo experimentov, ktoré odhalili, že nastavenie parametrov kartézskeho genetického programovania má veľký vplyv na úspešnosť evolúcie. Výsledky experimentov potvrdzujú vysoký potenciál využitia polymorfných hradiel v oblasti návrhu spoľahlivých systémov. K práci s vytvoreným programom bola vytvorená užívateľská príručka, ktorá sa nachádza na priloženom CD. Ďalšie pokračovanie projektu by mohlo viesť k použitiu multikriteriálnej optimalizácie a hľadaniu pareto-optimálnych riešení.

Literatura

[1] Lala, P. K.: Self-Checking and Fault Tolerant Digital Design. San Francisco, Morgan Kaufmann, 2001.

[2] Leveugle, R. Saucier, G.: Optimized synthesis of concurrently checked controller. IEEE Transactions on Computers, Volume 39(4), 1990, s. 419–425.

[3] Parekhji, R., Venkatesh, G., Sherlekar, S.: Concurrent error detection using monitoring machines. IEEE Design & Test of Computers, Volume 12(3), 1995, s. 24–32.

[4] Hlavička, J., Racek, S., Golan, P., Blažek, T.: Číslicové systémy odolné proti poruchám. Praha, Vydavatelství ČVUT, 1992.

[5] Gramatová, E. : Diagnostika a spoľahlivosť. Podklady k prednáškam zo školského roku 2006/2007, FIIT STU Bratislava.

[6] Garvie, M.: Reliable Electronics through Artificial Evolution. [PhD thesis], University of Sussex (2005).

[7] Miller, J. F., Thomson, P.: Cartesian genetic programming. In Proceedings of the Third European Conference on Genetic Programming. LNCS Volume 1802, Edinburgh, 2000, s. 121-132.

[8] Miller, J. F., Vassilev, V. K., Job, D.: Principles in the Evolutionary Design of Digital Circuits-Part I. Genetic Programming and Evolvable Machines, Volume 1(1/2), 2000, s. 7–35.

[9] Sekanina, L.: Biologií inspirované počítače. Podklady k prednáškam zo školského roku 2006/2007, FIT VUT Brno.

[10] Excerpta Medica Foundation: Human Genetics. 1962, s. 121-132.

[11] Vassilev, V. K., Miller J. F., Fogarty T. C.: On the Nature of Two-Bit Multiplier Landscapes. Proceedings of the First NASA/DOD Workshop on Evolvable Hardware (EH'99). New Jersey, IEEE Computer Society, 1999, s. 36-45.

[12] Vassilev, V. K., Miller, J. F.: The advantages of landscape neutrality in digital circuit evolution. In Proceedings of the Third International Conference on Evolvable Systems: From Biology to Hardware. LNCS Volume 1801, Edinburgh, 2000, s. 252-263.

[13] Yu, T., Miller, J. F.: Neutrality and the evolvability of Boolean function landscape. In Proceedings of the Fourth European Conference on Genetic Programming. LNCS Volume 2038, London, 2001, s. 204–217.

[14] Yu, T., Miller, J. F.: Finding needles in haystacks is not hard with neutrality. In Proceedings of the Fifth European Conference on Genetic Programming. LNCS Volume 2278, Kinsale, 2002, s. 13–25.

[15] Gajda, Z.: Metody návrhu polymorfních obvodů. [Téza k štátnej doktorskej skúške], FIT VUT v Brne (2007).

[16] Stoica, A., Zebulum, R. S., Keymeulen, D.: Polymorphic electronics. In Proceedings of Evolvable Systems: From Biology to Hardware Conference. Volume 2210, London, 2001, s. 291–302.

[17] Sekanina, L.: Evolutionary design of gate-level polymorphic digital circuits. In Applications of Evolutionary Computing. LNCS Volume 3449, Lausanne 2005, s. 185–194.

[18] Stoica, A., Zebulum, R. S., Keymeulen, D., Guo, X., Ferguson, M. I., Duong, V.: Taking Evolutionary Circuit Design From Experimentation to Implementation: Some Useful Techniques and a Silicon Demonstration. IEE Proceedings Computers and Digital Techniques. Volume 151(4), 2004, s. 295–300.

[19] Stoica, A., Zebulum, R. S., Keymeulen, D., Lohn, J.: On polymorphic circuits and their design using evolutionary algorithms. In Proceedings of IASTED International Conference on Applied Informatics AI2002. Insbruck, 2002.

[20] Zebulum, R. S., Stoica, A.: Four-Function Logic Gate Controlled by Analog Voltage. NASA Tech Briefs, Volume 30(3), 2006, s. 8.

[21] Zebulum, R. S., Stoica, A.: Multifunctional Logic Gates for Built-In Self-Testing. NASA Tech Briefs. Volume 30(3), 2006, s. 10.

[22] Sekanina, L.: Design and Analysis of a New Self-Testing Adder Which Utilizes Polymorphic Gates. In Proceedings of the 10th IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop. Gliwice, 2007, s. 243-246.

[23] Růžička, R., Sekanina, L., Prokop, R.: Physical Demonstration of Polymorphic Self-checking Circuits. IEEE International On-Line Testing Symposium 2008. V tlači.