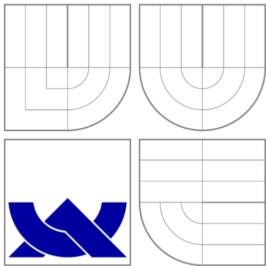
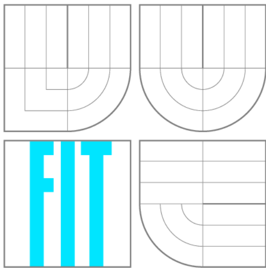


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

AKCELERACE EVOLUČNÍHO NÁVRHU OBVODŮ NA ÚROVNI TRANZISTORŮ NA PLATFORMĚ ZYNQ

ACCELERATION OF TRANSISTOR-LEVEL EVOLUTIONARY DESIGN OF DIGITAL CIRCUITS USING ZYNQ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. VOJTĚCH MRÁZEK
AUTHOR

VEDOUCÍ PRÁCE — Ing. ZDENĚK VAŠÍČEK, Ph.D.
SUPERVISOR

BRNO 2014

Abstrakt

Cílem této práce je návrh a realizace hardwarové jednotky umožňující automatickou syntézu integrovaných obvodů na úrovni tranzistorů. Práce je rozdělena na dvě části. První, teoretická část, se věnuje metodám návrhu obvodů s MOSFET tranzistory a problematice evolučních algoritmů. Dále rozebírá aktuální výsledky výzkumu v této oblasti a navazuje popisem nového přístupu evolučního návrhu a optimalizace číslicových obvodů na úrovni tranzistorů. Následující část se zabývá popisem hardwarové jednotky, která tuto novou metodu akceleruje na obvodu Zynq integrující procesor ARM a programovatelnou logikou. Funkčnost metody je prezentována na optimalizaci vícevstupých obvodů. Hardwarová jednotka byla využita v evolučním návrhu dvou a třívstupých hradel.

Abstract

The goal of this thesis is to design a hardware unit that is designed to accelerate evolutionary design of digital circuits on transistor level. The project is divided to two parts. The first one describes design methods of the MOSFET circuits and issues of evolutionary algorithms. It also analyses current results in this domain and provides a new method for the design and optimization. The second part describes proposed unit that accelerates the new method on the circuit Zynq which integrates ARM processor and programmable logic. The new method functionality has been empirically analysed in the task of optimization of few circuits with more inputs. The hardware unit has been tested for designing of gates on transistor level.

Klíčová slova

evoluční algoritmy, kartézské genetické programování, MOSFET tranzistory, transistorová úroveň, akcelerace, Zynq, ARM

Keywords

evolutionary algorithms, cartesian genetic programming, MOSFET tranzistors, transistor level, acceleration, Zynq, ARM

Citace

Vojtěch Mrázek: Akcelerace evolučního návrhu obvodů na úrovni tranzistorů na platformě Zynq, diplomová práce, Brno, FIT VUT v Brně, 2014

Akcelerace evolučního návrhu obvodů na úrovni tranzistorů na platformě Zynq

Prohlášení

Prohlašuji, že jsem tuto dipomovou práci vypracoval samostatně pod vedením pana Ing. Zdeňka Vašíčka, Ph.D.

....
Vojtěch Mrázek
18. května 2014

Poděkování

Tímto děkuji svému vedoucímu Ing. Zdeňku Vašíčkovi, Ph.D. za jeho příkladné vedení práce, rady s technickou realizací práce a pomoc s publikací. Dále chci poděkovat rodině a hlavně mé ženě za podporu.

© Vojtěch Mrázek, 2014.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod **3**

2 Evoluční algoritmy v oblasti návrhu analogových a číslicových obvodů **5**
2.1 Princip evolučního algoritmu . . . 5
2.2 Kartézské genetické programování . . . 7
2.3 Hodnocení úspěšnosti evolučních algoritmů . . . 10

3 Rekonfigurovatelné číslicové obvody **12**
3.1 Architektura PLA a PAL . . . 12
3.2 Architektura CPLD . . . 12
3.3 Architektura FPGA . . . 13

4 Implementace obvodů na úrovni tranzistorů **17**
4.1 Struktura MOSFET tranzistorů . . . 17
4.2 Popis obvodu . . . 18
4.3 Princip činnosti MOSFET tranzistorů . . . 20
4.4 Simulace obvodů s tranzistory . . . 23

5 Využití evolučních technik v návrhu obvodů s tranzistory **29**
5.1 Evoluce ve specializovaných obvodech . . . 29
5.2 Evoluce za použití simulátoru . . . 31
5.3 Analogová simulace . . . 31
5.4 Navržená metoda . . . 33

6 Platforma Zynq **38**
6.1 Procesorový systém . . . 38
6.2 Programovatelná logika . . . 39
6.3 Postup tvorby HW . . . 40
6.4 Postup tvorby SW . . . 41

7 Architektura akcelerátoru evolučního návrhu tranzistorových obvodů **42**
7.1 Řízení evaluace . . . 43
7.2 Virtuální rekonfigurovatelný obvod . . . 44
7.3 Výpočet fitness hodnoty a další výpočty ve VRC . . . 49
7.4 Implementace v procesorové jednotce s využitím HW akcelerace . . . 50

8 Softwarové nástroje **52**
8.1 Referenční implementace . . . 52
8.2 Pomocné nástroje . . . 53

9 Výsledky a experimenty **56**
9.1 Výsledky syntézy navrženého řešení 56
9.2 Rychlost obvodu 57
9.3 Vliv nastavení parametrů na úspěšnost evoluce 59
9.4 Evoluční návrh s využitím navržené metody 63

10 Závěr **67**

A Obsah CD **75**

B Příklady nalezených řešení **76**

Kapitola 1

Úvod

Tato diplomová práce se zabývá evolučním návrhem číslicových obvodů na úrovni tranzistorů a možnostmi jeho hardwarové akcelerace s využitím nedávno zveřejněné platformy Xilinx Zynq kombinující výhody výkonného procesoru s programovatelnou logikou. Motivací pro tuto práci je skutečnost, že v oblasti návrhu číslicových obvodů byla publikována řada prací, které využívají evolučních technik a demonstrují jejich výhody oproti konvenčně používaným návrhům. Typicky se však jedná o návrh obvodů na úrovni hradel. Například Walker a Miller navrhli systém pro návrh složitých obvodů, jako jsou násobičky [49]. Dále například Sekanina a Vašíček ukázali, že evoluční optimalizace je schopna produkovat řešení nedosažitelná komerčními algoritmy [46]. Zavedení abstrakce u návrhu na úrovni hradel má kromě výhod i daň v podobě suboptimální implementace na úrovni tranzistorů.

Typickým příkladem je čtyřvstupé hradlo AND-NOR. Při implementaci pomocí hradel využijeme 18 tranzistorů. Navrhneme-li však tranzistorovou strukturu přímo, jsme schopni zoptimalizovat počet tranzistorů na 8. Proces optimalizace je však náročný. Můžeme použít klasických metod, jako je například rozložení na DAG primitiva [17], což je však NP těžký problém, což nás nutí využít některou z heuristik. V této práci však použijeme nekonvenční metodu evolučního návrhu.

Ačkoliv využití evolučních algoritmů v oblasti návrhu přináší řadu výhod, obsahuje i řadu omezení. Typicky se potýkáme s problémem škálovatelnosti evoluce. Škálovatelnost představuje zásadní problém, který nám znemožňuje navrhovat složitější struktury. V oblasti evolučního návrhu rozlišujeme dva druhy škálovatelnosti — škálovatelnost reprezentace a škálovatelnost evaluace. Škálovatelnost reprezentace vyjadřuje schopnost rozšíření použitého kódování pro reprezentaci složitějších struktur. Škálovatelnost evaluace znamená, že pro složitější obvody je náročnější provést ohodnocení kandidátního řešení. Při syntéze číslicových obvodů nás zásadně limituje počet vstupů, protože je potřeba ohodnotit všechny vstupní kombinace, jejichž počet roste exponenciálně s počtem vstupů. Přechodem na nižší úroveň abstrakce tento problém nabývá na významu.

Při použití evolučních algoritmů při návrhu je bývá většinou největším problémem čas simulace jednotlivých kandidátních řešení. Abychom co nejvíce potlačili problém škálovatelnosti evaluace, je v této práci navržen nový přístup k optimalizaci těchto obvodů. Celá simulace je převedena do odpovídajícího diskrétního tvaru a je implementována jednak na CPU, tak i akcelerována na hradlovém poli. Cílem práce je tedy navrhnout rychlejší přístup k návrhu a optimalizaci obvodů s tranzistory s využitím jiné reprezentace.

Práce je členěna následovně. Kapitola 2 se věnuje popisu evolučních algoritmů, zejména kartézskému genetickému programování. Kapitola 3 ukazuje možnosti hardwarových obvodů využitelné pro zrychlení evolučního návrhu. Struktura MOSFET tranzistorů, jejich

chování, klasické způsoby návrhu a možnosti simulace jsou popsány v kapitole 4. V kapitole 5 jsou rozebrány aktuální výsledky v oblasti evolučního návrhu obvodů na úrovni tranzistorů a je v ní navržen nový přístup. Cílová architektura systému Xilinx Zynq, který je využit pro implementaci akcelerační jednotky, je popsaná v kapitole 6. Navržená struktura akceleračního obvodu a evolučního návrhu je popsána v kapitole 7. Pro porovnání byla vytvořena i softwarová implementace, které se věnuje kapitola 8. Mimo to obsahuje i popis pomocných nástrojů, které slouží k urychlení vývoje a testování navrženého řešení. V kapitole 9 je ověřena korektní funkce navrženého evolučního návrhu a jsou v ní vyhodnoceny parametry akcelerátoru. Tato kapitola obsahuje i některé příklady navržených obvodů. Poslední částí je kapitola 10 ukazující možnosti dalšího postupu a zhodnocení návrhu.

Kapitola 2

Evoluční algoritmy v oblasti návrhu analogových a číslicových obvodů

Evoluční algoritmy jsou stochastické, většinou populačně orientované, aplikace, které převádí řešení daného problému na úlohu prohledávání stavového prostoru. K tomuto úkolu využívají operací, které známe z genetiky – křížení a mutace, dále řeší výběr následného jedince. Následující text shrnuje princip využití evolučních algoritmů při návrhu analogových a číslicových obvodů tak, aby se čtenáři přiblížily metody použité v práci. Pro detailní informace lze nahlédnout do [35].

2.1 Princip evolučního algoritmu

Jako první krok při použití EA se musíme rozhodnout, jaký způsob reprezentace dat budeme využívat. Možností je mnoho, proto si představíme nejčastěji používané metody.

Jednou z možností kódování je přenesení problému do grafu. Tento způsob použil například J. Koza při svém návrhu analogových obvodů [19]. Reprezentace vycházela z jím používaného jazyka LISP, kdy z počátečního embryonálního obvodu přidáváním uzlů, rezistorů, kondenzátorů a cívek vytvářel vlastní kandidátní řešení.

Další způsob reprezentace je permutační kódování, které se používá pro speciální problémy, jako je problém obchodního cestujícího. Toto kódování vkládá za sebe průchod grafem, ve kterém problém řešíme.

Často také využíváme binární kódování použité například u CGP[1], kterým konfigurujeme propojení jednotlivých elementů v acyklickém grafu.

Dalších možností kódování je celá řada. U genetických algoritmů se většinou jedná o stromy, u evoluční strategie využíváme reálných vektorů.

Abychom pro řešení daného problému mohli použít evoluční algoritmus, musíme splnit tyto základní podmínky: (a) musíme být schopni jednoznačně zakódovat řešení a (b) musíme být schopní toto řešení deterministicky ohodnotit tak, aby kvalitnější řešení měla lepší ohodnocení a všechna ohodnocení musí být porovnavatelé.

Základní evoluční algoritmus můžeme popsat následujícími kroky:

1. Náhodné vytvoření počáteční populace jedinců

[1] Kartézské genetické programování

2. Ohodnocení jednotlivých jedinců
3. Výběr jedinců pro křížení (rekombinaci) a mutaci
4. Křížení jedinců a následná mutace
5. Aktualizace populace a pokračování krokem 2

2.1.1 Ohodnocení jedinců

K ohodnocení můžeme využít dvou přístupů. Nejčastěji používanou metodou je tzv. *nepravá evoluce* (anglicky *extrinsic evolution*), kdy pro ohodnocení kandidátního řešení předáme každé řešení simulátoru. Nevýhodou tohoto řešení je fakt, že nejsme schopni vytvořit simulátor, který by zahrnoval všechny aspekty reálného světa. Výhodou je však to, že můžeme počítat i s tolerancemi a jsme schopni vytvořit obecné řešení, které nevyužívá konkrétních fyzických vlastností cílové platformy.

Oproti tomu druhou metodou je tzv. *pravá evoluce* (anglicky *intrinsic evolution*), při které konfigurujeme přímo výsledné zařízení. Výhodou tohoto řešení je zahrnutí i vyvíjeného prostředí. Díky tomu je možné tento způsob využít například v radioaktivním nebo tepelně extrémním prostředí, kde se pomocí evoluce upraví parametry obvodu tak, aby fungoval správně [18]. Nevýhodou je možná nereprodukovatelnost řešení, protože výsledná konfigurace může využívat dalších fyzikálních zákonů a spojitostí nebo drobných chyb například v integrovaném obvodu. Příkladem může být experiment A. Thompsona z roku 1996, kdy používal neomezenou pravou evoluci v FPGA obvodu pro vytvoření tónového diskriminátoru. Výsledek byl plně funkční, ale nepřenositelný do jiného zařízení [42].

Pro každého jedince musíme převést výsledek simulace na tzv. fitness, která určuje správnost řešení. Je nutné, aby se jednalo o číslo a díky tomu byla jednotlivá řešení porovnatelná. V případě multikriteriální optimalizace může být těchto výsledků víc, pro vlastní určení lepších řešení využijeme Pareto fronty, kde nalezneme dominující řešení v jednotlivých parametrech. Vlastní výpočet se pak liší podle požadovaného výsledku – může se jednat například o počet správných výsledků u návrhu kombinačních obvodů, průměrnou odchylku od referenčního obrázku při návrhu obrazových filtrů a podobně.

2.1.2 Výběr jedinců

Po ohodnocení populace je nutné vybrat jedince, kteří budou základem další populace. Na výběr máme z více algoritmů. První kategorií jsou nedeterministické, při kterých využíváme náhody. Jako zástupce si můžeme představit algoritmus *ruleta*, kdy jedince hledáme náhodně s pravděpodobností $\frac{f}{\sum f}$, kde f je fitness.

Někdy však tento způsob výběru nemusí být vhodný. Jedná se o případy s jedincem, který má mnohokrát větší hodnotu fitness než ostatní jedinci. Pak můžeme využít *výběru podle pořadí.* V tomto případě seřadíme jedince podle fitness a rodiče vybereme s pravděpodobností odpovídající pořadí.

Dalším ze zástupců nedeterministických algoritmů výběrů je *turnaj.* Dokud potřebujeme najít rodiče, vybereme náhodně K jedinců (kde K je obvykle 2) a z těchto vybraných jedinců použijeme toho, kdo má lepší fitness.

Oproti tomu deterministickým algoritmem může být výběr n nejlepších jedinců.

Při vytvoření nové generace můžeme použít Steady-State nahrazení, kde P prvků nové populace je složeno z N nových a z M předcházejících, obvykle nejlepších. Dalším nahrazováním je generační, v němž je nová generace složena pouze z nových prvků (získaných

použitím genetických operátorů). Občas se také vyskytuje tzv. elitismus, kterým rozumíme to, že nejlepší jedinec z předchozí generace je vždy obsažen v generaci nové.

2.1.3 Genetické operátory

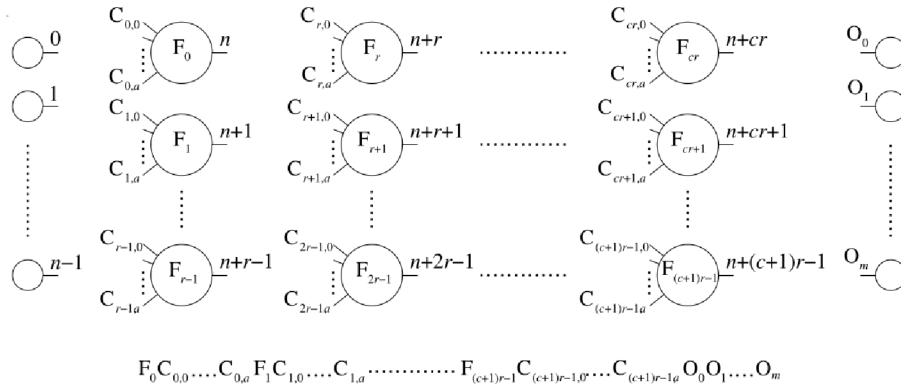
Rozlišujeme dva základní genetické operátory. Prvním operátorem je křížení. Konkrétní implementace závisí zejména na zvoleném kódování. U genetických algoritmů využívajících stromové kódování se jedná o výměnu určitého podstromu mezi dvěma jedinci (může se jednat i o stejné jedince). Při binární reprezentaci se některé bity z jednoho vektoru přesunou do druhého a naopak.

Druhým operátorem je mutace. Mutace typicky bývá méně pravděpodobná než křížení, ale konkrétní nastavení závisí až na aplikac. V některých aplikacích bývá výhodné intenzitu mutace měnit dynamicky. Při použití stromové reprezentace dat dochází k výměně části stromu s náhodně vygenerovaným uzlem. U reprezentace reálnými čísly nemůžeme prohodit jednotlivé bity, ale musíme mírně posunout číslo (nejlépe s normální pravděpodobností).

2.2 Kartézské genetické programování

Kartézské genetické programování je pravděpodobně nejpoužívanější algoritmus v oblasti evolučního návrhu digitálních obvodů. Tento algoritmus byl představen poprvé v roce 1997 J. Millerem [28]. Jak již název napovídá, jedná se o variantu genetického programování. Odstraňuje však problém komplexních operací nad stromy tím, že řešení nejsou kódována ve formě stromové struktury, ale pomocí acyklického grafu. Tento graf bývá často reprezentován ve formě dvoudimenzionální mřížky složené z výpočetních elementů. Jako samostatná metoda genetického programování bývá považovaná od roku 2000 [27].

V základní verzi CGP definované Millerem [29] se jedná o acyklický orientovaný graf. Jednotlivé geny reprezentující genotyp jsou celá čísla uspořádaná v n–ticích. Každý programovatelný element je realizovaný look–up funkcí, která zadaným vstupům přiřadí konkrétní výstup.



Obrázek 2.1: Základní reprezentace CGP. Mřížka má c sloupců, r řádků, n vstupů a m výstupů. Každý element realizuje funkci s maximálním počtem a operandů. Je zobrazeno i kódování chromozomu. [29]

2.2.1 Kódování chromozomu

Vzhledem k pevné mřížce má i chromozom pevnou délku. Je složen z celých čísel, jejichž význam se liší podle pozice. Předpokládejme, že každý element má a vstupů (operandů). První číslo definuje funkci elementu a nabývá hodnot podle počtu možných funkcí. Dalších a čísel určuje vstupní hodnotu pro každý z operandů elementu. Nabývá hodnot jednoznačně identifikující jeden z primárních vstupů a z výstupů elementů z předcházejícího sloupce. Tyto $(a + 1)$-tice se opakují pro každý element z mřížky. Za těmito čísly následuje definice výstupu. Výstup je definován jednoznačným identifikátorem elementu, jehož výstup odpovídá požadovanému výstupu. Počet těchto čísel odpovídá celkovému počtu výstupů.

Chromozom má tedy délku odpovídající rovnici

$$(r \cdot c)(a + 1) + o \tag{2.1}$$

kde r je počet řádků a c je počet sloupců. Element může mít pro některou funkci i méně operandů, pak jsou další propojení ignorována. Číslo o určuje počet výstupů.

Ačkoliv je k dispozici $r \times c$ elementů, neznamená to, že se obvod musí nutně skládat z přesně $r \times c$ komponent. Všechny elementy totiž nemusí být aktivní. Aktivitou rozumíme to, že od nich exituje cesta grafem k elementu, který je připojen k výstupu. Ve výsledku bývá aktivních pouze pár elementů, ty neaktivní jsou však důležité z důvodu neutrální mutace popsané v kapitole 2.2.3.

2.2.2 Výpočet fitness

Výpočet fitness hodnoty neboli hodnoty určující kvalitu řešení, kde vyšší hodnota určuje kvalitnější řešení, závisí na konkrétní aplikaci. Typickou úlohou je evoluční návrh číslicových obvodů na úrovni hradel, zaměříme se na výpočet fitness u těchto kombinačních obvodů. Při výpočtu spustíme simulaci obvodu pro všechny kombinace vstupu. Tudíž musíme provést 2^i běhů simulace, kde i odpovídá počtu výstupu. V programu máme definovanou tabulku mapování vstupů na výstup $I \rightarrow O$. Výsledná fitness poté odpovídá celkovému počtu správných výstupů. Pokud se nám podaří dosáhnout maximálního počtu správných výstupů $o \cdot 2^i$, začneme obvod optimalizovat z pohledu počtu použitých prvků. V tomto případě k fitness budeme připočítávat i celkový počet nepoužitých elementů. Není to však jediná možnost.

V některých případech nás tolik nezajímá přesný výsledek, ale spíše energetická náročnost nebo počet použitých prvků i za cenu menší přesnosti. Těmto obvodům říkáme aproximační. V této oblasti právě probíhá výzkum a na určení fitness je více způsobů [34, 30]. Dalším z případů, kdy nejsme schopni zaručit naprosto přesný výstup, je návrh obrazových filtrů. V těchto aplikacích často používáme trénovací obrázek, který máme v poškozené a nepoškozené verzi. Při výpočtu celkové fitness počítáme s celkovou odchylkou opraveného obrázku $M \times N$ w oproti referenčnímu v [33]

$$\text{fitness} = \sum_{i=1}^{M-2} \sum_{j=1}^{N-2} |v(i,j) - w(i,j)| \tag{2.2}$$

Vzhledem k tomu, že pro vyšší počet vstupů nám u přesného vyhodnocení kombinačních obvodů doba vyhodnocení roste exponenciálně, jedná se o nejslabší místo algoritmu. Tento problém je možné řešit více způsoby. Prvním způsobem je použití hrubé síly se zapojením velkého clusteru počítačů [47], nebo s využitím speciální platformy. Můžeme využít například grafické karty GPU [11], nebo použít FPGA, ve kterém vytvoříme virtuální

rekonfigurovatelný obvod [45]. Druhou možností je využití znalostí z formálních metod verifikace. Za použití SAT solveru jsme schopni verifikovat, zda jsou dva obvody (referenční a testovaný) shodné či nikoliv [46].

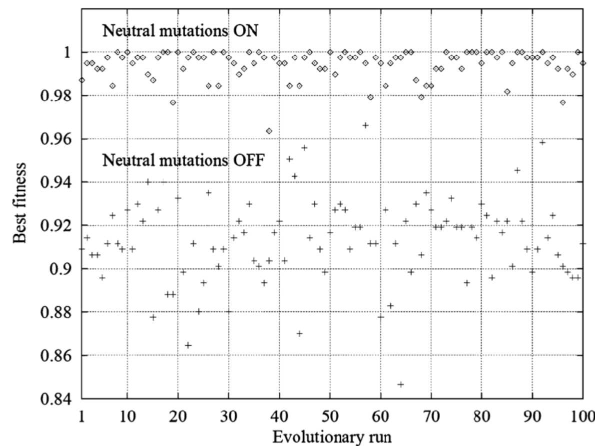
2.2.3 Prohledávací algoritmus

Vlastní algoritmus vychází z evoluční strategie $1 + \lambda$. V populaci je vždy nejlepší z předcházejících generací a dalších λ jedinců získaných genetickými operátory. Obvykle se λ pohybuje mezi 5 až 20. [33]. V CGP se využívají většinou jen mutace v rozsahu přibližně 5 % [29], zatím nebyl nalezen vhodný operátor křížení.

Algoritmus pracuje v těchto krocích:

1. Náhodně se vygeneruje λ jedinců, vyhodnotí se a nejlepší se označí za rodiče
2. Z rodiče vytvoříme λ jedinců pomocí mutace
3. Ohodnotíme všechny jedince
4. Vybereme jedince, který má vyšší nebo stejnou fitness jako rodič, a označíme jej za rodiče další generace. Pokud takový jedinec neexistuje, rodič zůstává původní
5. Dokud neproběhl daný počet iterací, pokračujeme bodem 2

Algoritmus zachovává nejlepšího jedince z celého běhu, dále z něj za pomoci mutace tvoří další jedince. Tuto vlastnost nazýváme *elitismus*. V algoritmu je při porovnávání fitness vůči rodiči použita nerovnost $\geq$. Označit prvek, který má stejnou fitness jako rodič, je důležité. Tyto mutace jsou nazývány jako tzv. neutrální mutace. Bez těchto mutací dochází k nalezení požadovaného řešení daleko později nebo se dokonce snižuje pravděpodobnost nalezení řešení vůbec. [29, str. 32] Tento vliv můžeme vidět na obrázku 2.2.



Obrázek 2.2: Vliv neutrálních mutací v úloze návrhu tříbitové paralelní sčítačky. Ukazuje normalizovanou fitness hodnotu ve dvou sadách běhů o 100 bězích s 1 milionem generací. v sadě běhu s deaktivovanou neutrální mutací se nepodařilo najít správné řešení. [29, str. 32]

2.2.4 Vliv parametrů na úspěšnost algoritmu

Vzhledem k tomu, že se jedná o genetický algoritmus, k nalezení výsledku je nutná určitá míra náhody. Pravděpodobnost nalezení správného výsledku můžeme ovlivnit volbou více parametrů. Prvním parametrem je počet řádků a sloupců mřížky elementů. To určuje, v jak velkém stavovém prostoru bude problém řešen. Příliš malá mřížka způsobuje to, že nebude možné řešení ve vymezeném prostoru implementovat tak, aby bylo zcela funkční — například z důvodu nedostatečného počtu uzlů nebo z důvodu nedostatečného počtu sloupců, který vzhledem ke způsobu propojení určuje maximální zpoždění výsledného obvodu. Naopak příliš velká mřížka může zapříčinit to, že prohledávaný prostor bude velký a díky pravděpodobnost nalezení řešení bude menší. S rozložením mřížky souvisí i tzv. *LBACK* parametr, který určuje, o kolik sloupců nazpět se může připojit vstup elementu. Pohybuje se v rozmezí 1 až $c - 1$. Při použití virtuálního rekonfigurovatelného obvodu v FPGA, například v obrazovém filtru [45], bývá obvykle hodnota tohoto parametru 1. Díky tomu můžeme většinou použít řetězené zpracování a tím výrazně zkrátit dobu evaluace. Někdy se provádí i to, že *LBACK* je maximální a $r = 1$. Potom prvky nejsou uspořádány v mřížce, ale v řadě s plnou konektivitou. Dalším možností ovlivnění úspěšnosti algoritmu je volba funkcí elementů. Pro klasické kombinační obvody volíme funkce NAND, OR a podobně, pro obrazové filtry se často objevuje operace sčítání, maxima a podobně.

2.3 Hodnocení úspěšnosti evolučních algoritmů

Vzhledem k tomu, že evoluční algoritmus je stochastický proces, který negarantuje nalezení jednoho konkrétního řešení, je nutné pro objektivní vyhodnocení jeho úspěšnosti využít typicky výsledků z více nezávislých běhů. Získaná data je dále nutné zpracovat a určit míru vlivu jednotlivých parametrů.

Mezi sledované vlastnosti může patřit fitness funkce, velikost výsledného řešení nebo například pouhá informace o tom, jestli byl návrh úspěšný. Většina sledovaných vlastností odpovídá průběhu náhodné veličiny, kterou jsme schopni matematicky popsat typicky pomocí normálního neboli Gaussova rozložení. Mezi základní charakteristiky náhodné veličiny patří její průměr μ.

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2.3}$$

Průměr nám však nic neříká o tvaru distribuční funkce, určuje pouze její střed. Důležitá je pro nás i její šířka. Tato šířka je definována směrodatnou odchylkou σ.

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (x_i - \mu)^2} \tag{2.4}$$

Výběrový průměr má normální rozdělení s rozptylem určeným ze směrodatné odchylky.

$$\overline{X} \sim N\left(\mu, \frac{\sigma^2}{n}\right) \tag{2.5}$$

Významným parametrem, který vyjadřuje velikost, je interval důvěry (*confidence interval*). Tento interval určuje rozsah, ve kterém se veličina bude vždy pohybovat s pravděpodobností p. Výpočet tohoto intervalu probíhá pro normální rozložení následovně [14]

$$\left(\overline{X} - u_{\frac{1+p}{2}} \cdot \frac{\sigma}{\sqrt{n}}, \overline{X} + u_{\frac{1+p}{2}} \cdot \frac{\sigma}{\sqrt{n}}\right) \tag{2.6}$$

Násobením střední chybou průměru normalizujeme směrodatnou odchylku. Díky tomu můžeme využít konstantu $1,96$ pro určení $u_{\frac{1+p}{2}}$ pro pravděpodobnost 95 %.

Významnou podmínkou pro předchozí ukazatele je to, že náhodná veličina musí mít normální rozložení. Pokud však veličina má neurčité rozložení, je pro popis rozložení lépe vypovídající medián. Medián je definován jako hodnota T, která větší než polovina vzorků a menší než polovina vzorků [7]. Je možné jej definovat také jako hodnotu, pro kterou je pravděpodobnost $F(X < T) = \frac{1}{2}$

$$\int_{-\infty}^{T} p(x)dx = \frac{1}{2} \tag{2.7}$$

Kromě statistických parametrů můžeme v literatuře [51] najít následující metriky vhodné pro porovnání jednotlivých algoritmů či jejich nastavení.

Intenzita úspěchu neboli *success effort* [52] vychází z definice *hit effort* podle J. Kozy [20]. Určuje generaci (včetně intervalu důvěry 95 %), při které dojde k nalezení řešení. Výpočet probíhá ze dvou množin g_s, která určuje množinu všech generací, při kterých došlo k nalezení úspěšného řešení, a množiny g_f, která určuje množinu všech generací, při kterých došlo k selhání algoritmu. Problémem aplikace je to, že v některých případech nejsme schopni určit množinu g_f, pokud nedojde k selhání algoritmu, ale pouze k jeho uváznutí. A pravděpodobnost obou jevů musí být dohromady 1.

Průměrná fitness představená v práci [2] ukazuje aritmetický průměr fitness hodnoty. Z této hodnoty můžeme pomocí definice normálního rozložení určit interval důvěry. Často bývá používáno i T-rozložení s n stupni volnosti. Při použití této metriky musíme dbát na to, aby sledovaná fitness měla normální rozložení, jinak aritmetický průměr není vypovídající hodnotou.

Podíl úspěchu neboli *success proportion* určuje kumulativní pravděpodobnost $P(i)$ nalezení správného řešení. Je definována pro generaci i jako podíl počtu běhů, které v dané generaci už nalezly správné řešení, a celkového počtu běhů.

Kapitola 3

Rekonfigurovatelné číslicové obvody

Pro implementaci vyvíjecího se hardwaru (evolvable hardware) často využíváme rekonfigurovatelných obvodů. Tyto obvody mění své chování podle konfiguračního řetězce, který můžeme buď měnit, nebo bývá nahrán pouze jednou na začátku.

3.1 Architektura PLA a PAL

Jedním z prvních platforem pro návrh adaptivních obvodů byla architektura PLA [16]. Jedná se o maticové uspořádání hradel AND a OR (nebo jiných). Při programování dochází k odstranění některých propojů, které nepotřebujeme. Toto programování může být buď destruktivní, kdy nebudeme už nikdy moci spoj obnovit, nebo na bázi EEPROM, kde je možnost vyšším napětím konfiguraci vrátit do původního stavu. Obvody umožňují realizovat libovolnou logickou funkci. Ne všechny programovatelné obvody umožňují plnou konfiguraci. Například obvody s monolitickou pamětí PAL mají programovatelnou AND matici, ovšem matice OR je spojena pevně od výroby. Počet termů pro vstup do OR bývá tedy omezen na počet 2, 4, 8 nebo 16 [16]. Klíčový rozdíl mezi PLA a PAL je to, že první z nich může využívat sdílených termů a druhý nikoliv.

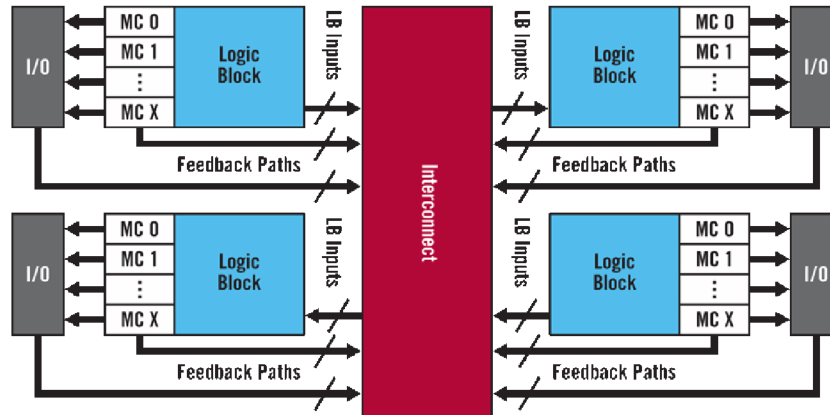
V evolučních algoritmech byly tyto architektury použity například pro hledání FIR filtrů s možností detekce a opravy chyby [12]. Dalším příkladem využití může být platforma PLAGA určená pro optimalizaci testů těchto obvodů [4].

3.2 Architektura CPLD

Dalším stupněm vývoje jsou CPLD[1] obvody. Tyto obvody obsahují takzvané makrobuňky, které mohou realizovat nejen kombinační funkci, ale také sekvenční obvody a další specializované funkce. V dnešní době se do těchto obvodů přidávají také analogové prvky (DA a AD převodníky), což umožňuje další možnosti zpracování signálů s poměrně malou energetickou náročností. Například obvod Xilinx CoolRunner-II má udávanou spotřebu 4,5 – 100 mW při frekvenci 50 MHz (podle složitosti obvodu) [54].

Na obrázku 3.1 můžeme vidět základní strukturu bloku CPLD obvodu. Jedná se o spojení makrobuněk přes konfigurovatelnou propojovací síť mezi sebou. Spojení mezi vstupy

[1]Complex programmable logic device



Obrázek 3.1: Struktura CPLD [59]

a výstupy bývá řešeno pomocí PLA částí v obvodu. U nejmenšího obvodu ve výše uvedené sérii XC2C32A můžeme využít až 32 makrobuněk a 33 pinů při frekvenci 323 MHz. Nejsložitější XC2C512A obsahuje 512 makrobuněk propojitelný s 270 piny. Tento obvod dosahuje frekvence až 179 MHz [54]. Tyto architektury postupně nahrazují obyčejné PLA a PAL čipy.

3.3 Architektura FPGA

Na nejvyšší úrovni z hlediska konfigurovatelnosti jsou dnes obvody FPGA[2]. Tyto obvody vyrábí více firem, jako je například Xilinx nebo Altera. V tomto přehledu se však více zaměříme na součástky firmy Xilinx, protože na těchto obvodech bude realizovaná praktická část práce.

Základní jednotkou obvodů FPGA je tzv. CLB[3]. Jedná se o buňku obsahující několik slice bloků, které se skládají například z look–up tabulek, úplné sčítačky a flip-flop obvodu typu D. Dále na čipu můžeme najít paměti Block RAM, obvody zajišťující distribuci hodin a další pevná jádra, jako může být třeba procesor PowerPC u architektury Virtex. Všechny části jsou propojeny pomocí propojovací matice, která je velmi složitá (zabírá cca 80 % plochy čipu).

Programování takového obvodu znamená nastavení propojovací matice a také konfiguraci jednotlivých CLB obvodů. Tato data jsou zakódována v tzv. bitstreamu, který přenášíme do zařízení nejčastěji přes JTAG rozhraní.

3.3.1 Obvod FPGA XC6200

Jedním z prvních obvodů použitých pro evoluční přístup s tzv. intrinsic evolucí byl obvod XC6200. Jeho architektura byla významná tím, že u ní byl přesně známý konfigurační řetězec. V dnešní době je však komerčně zastaralá a nevyrábí se.

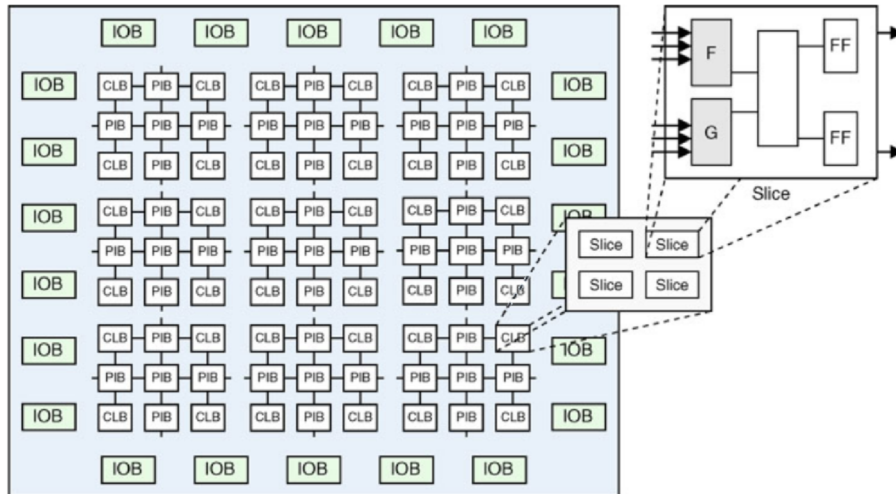
Výše uvedené obvody využíval A. Thopson ke svým experimentům [41, 42]. V těchto pracích představil evoluční metodu pro hledání správné konfigurace obvodu. Výhodné na konfiguraci bylo to, že se jednalo o multiplexorově založenou architekturu, kde nebylo možné jakýmkoliv způsobem poškodit obvod nevhodnou konfigurací [43]. Bylo tedy možné spustit

[2]Field Programmable Gate Array
[3]Configurable logic block

neomezenou evoluci pro nalezení optimálního řešení. Vzhledem k tomu, že se už tento obvod nevyrábí, byly vytvořeny konfigurace stávajících obvodů firmy Xilinx a Altera, které emulovaly obvod XC6200 [36].

3.3.2 Moderní obvody FPGA

Dnešní FPGA obvody se mírně odlišují od předchozího představeného. Vzhledem k jejich složitosti je konfigurační řetězec mnohem delší a hlavně jeho obsah je nedokumentovaný. Není možné tedy provádět evoluci přímo, ale pomocí dalších metod uvedených v kapitole 3.3.4.



Obrázek 3.2: Typická architektura FPGA firmy Xilinx [33]

Obrázek 3.2 ukazuje typickou architekturu FPGA firmy Xilinx. Zde se jedná o řadu Virtex. Skládá se z dvoudimenzionální mřížky CLB bloků, programovatelných mezispojů PIB a rekonfigurovatelných vstupně výstupních IOB. CLB blok obsahuje 3-, 4- nebo 6-vstupou look-up tabulku, přídavnou logiku a dva flip-flop obvody.

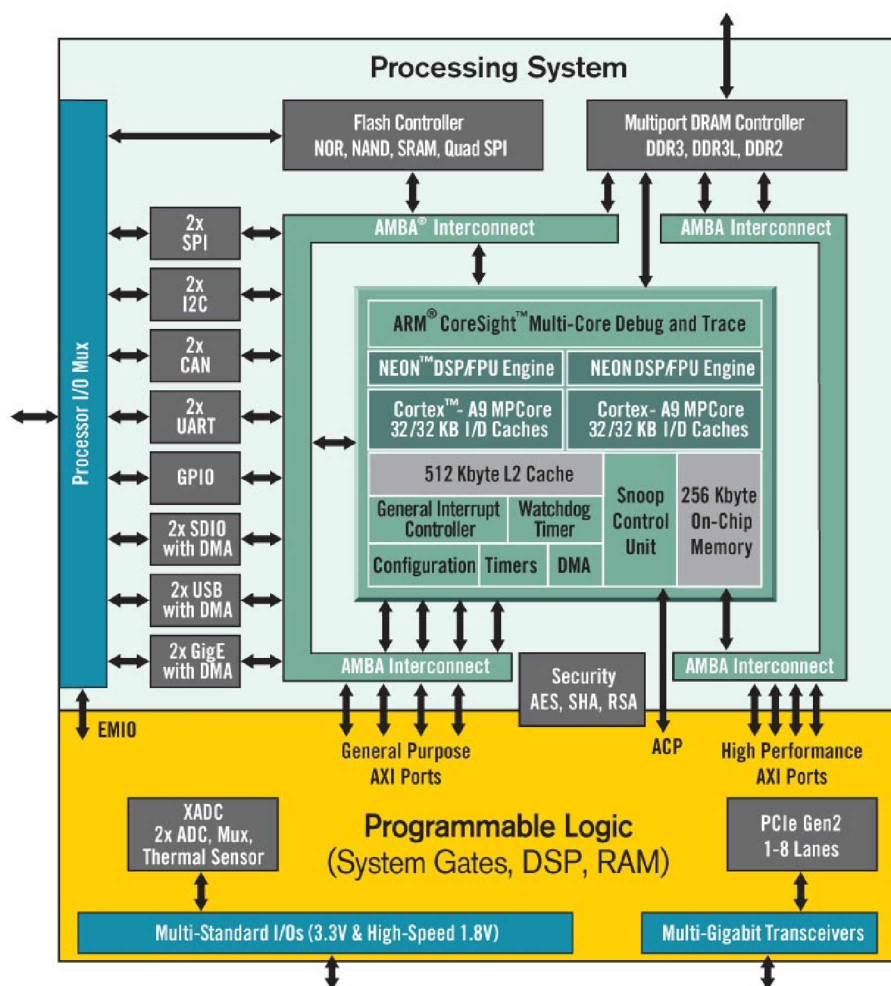
Celý bitstream je uložen v paměti SRAM, do které je tedy nutné po připojení data nahrát. V dnešní době dostupné čipy se liší hlavně v tom, jaké pevné součásti obsahují. Nejjednodušší řada **Spartan-6**, vyráběná 45 nm technologií, obsahuje kolem 150 tisíc logických jednotek. Výrobce upouští od *soft procesorů* distribuovaných ve formě bitstreamu nebo VHDL kódu a dnešním trendem je i do těchto jednoduchých čipů integrovat procesory MicroBlaze nebo PicoBlaze ve formě *hard jádra* [55].

U vyšších řad, jako je **Artix-7, Kintex-7** nebo **Virtex-7**, výrobce integruje mnohem více součástí. Můžeme zde najít PCI express rozhraní, dále procesor MicroBlaze nebo PowerPC. Vzhledem k většímu stupni integrace výrobní proces vyžaduje lepší technologii. V dnešní době se používá 28 nm až 16 nm u **Virtex-7 UltraScale** rodiny. Počet logických bloků dosahuje až 2 milionů [62].

3.3.3 Zynq

Architektura Zynq představuje jednu z nejnovějších architektur rekonfigurovatelných obvodů. Výrobce ji již neoznačuje jako FPGA, ale jako SoC[4]. Skládá se z procesorového

[4]System on chip



Obrázek 3.3: Architektura SoC Zynq firmy Xilinx [60]

systému a programovatelné logiky. Od ostatních architektur Xilinx se odlišuje integrovanými součástmi a také celkovým přístupem. Srdcem celého systému je dvoujádrový procesor rodiny ARM Cortex A9. Nejen, že integrovaný procesor je výkonnější než u jiných rodin FPGA, ale také se liší spouštěním. Architektura Virtex je označovaná jako *PL-centric*, což znamená, že se nejprve inicializuje logická část (programmable logic). Oproti tomu u SoC Zynq se jedná o architekturu *PS-centric*, kde jako první startuje procesor (processign system), který se stará o běh programovatelné logiky.

Vzhledem k tomu, že se jedná o poměrně novou architekturu, jejímu použití v oblasti evolučních algoritmů se věnuje pouze několik prací. Má ovšem velký potenciál, protože díky vysoké taktovací frekvenci PS (667 MHz - 1 GHz) a rychlému spojení s logickou částí jsme schopni dosáhnout velkého zrychlení. Použití této architektury se věnoval R. Dobai [6]. Při jeho výzkumu se mu podařilo zrychlit evoluci obrazového filtru až 200x.

3.3.4 Rekonfigurace

Pro správnou funkci evolučního algoritmu budeme potřebovat dynamicky měnit konfiguraci obvodu. Tuto rekonfiguraci můžeme provádět více způsoby, které představil Upegui a

Sanchez [43] pro Virtex-II. Princip se však promítá i do jiných rodin rekonfigurovatelných FPGA a SoC.

Modulárně orientovaná rekonfigurace je založená na základních blocích, modulech. Tyto bloky komunikují se svými sousedy. Jejich konfiguraci můžeme měnit, čímž změníme funkci modulu, nebo také výběr souseda, se kterým modul komunikuje. Využití této rekonfigurace je na úrovni HDL. Jedná se o případ, kdy jednotlivé moduly jsou genericky uspořádané a modifikací nejvyšších vrstev spojení modulů pomocí evolučních algoritmů dochází k vytváření dalších kandidátních řešení. Poté musí dojít k syntéze nebo se využije přeuspořádání.

Rekonfigurace využivající pevná makra představuje nižší úroveň modifikace. Využívá parciální rekonfigurace, kterou dnešní obvody umožňují. Jedná se o speciální modul FPGA obvodu (např. ICAP), který umožňuje interně zaměnit část obvodu za jinou část. S použitím těchto maker definujeme omezení pro umístění makra a dále pak přesně určíme LUT tabulku. Pomocí znalosti umístění konfigurace této tabulky můžeme potom měnit dynamicky propojení. Nevýhodou tohoto řešení je to, že musíme přesně znát umístění konfiguračních součástí.

Zmiňovaný přístup byl například použit v koevoluci fuzzy systému [26]. Byla definována dvě pevná makra – parametrické a fuzzy pravidlo. Parametrické pouze zpracovávalo konstantní parametry. Po specifikování omezení tohoto makra bylo možné přistupovat a modifikovat jeho obsah automaticky pomocí FPGA editoru. Podobným způsobem byl upravován fuzzy modul — mohl být automaticky konfigurován pro implementaci fuzzy-OR nebo fuzzy-AND funkce [43].

Manipulace s bitstreamem je nejnižší úrovní dynamické rekonfigurace obvodu. Tato rekonfigurace je velmi závislá na nástrojích firmy Xilinx pro dynamickou rekonfiguraci obvodu. Dostupnost nástrojů pro editaci bitstreamu je však velmi malá. Vývojáři Xilinx oznámili nástroj XPART, ovšem ten nebyl nikdy vydán [43]. Proto mezi nejrozšířenější nástroje pro manipulaci s bitstreamem patří SDK pro Javu JBits [10, 9]. Jedná se o nástroj určený pro obvody řady XC4000, později i pro Virtex-II. Problematickou částí je to, že není možné měnit bitstream přímo binárně pomocí neomezené evoluce, protože by mohlo dojít k poškození obvodu. Tento problém se však netýkal obvodu XC6200 zmíněného v kapitole 3.3.1.

Virtuální rekonfigurovatelný obvod VRC [32] se od předchozích způsobů liší, protože se nejedná o přímou rekonfiguraci na úrovni spojovací matice PL, ale jedná se o konfiguraci s využitím logických bloků FPGA. Využívá spojení kartézské mřížky modulů, kde můžeme pomocí multiplexorů měnit zapojení vstupů a tím vybírat sousedy, kteří ovlivňují výstup. Dále můžeme případně měnit funkci modulu, například velmi často se jedná o hradla nebo jiné prvky filtrů. Celkově struktura tohoto obvodu koresponduje s reprezentací CGP, proto je často využívána ve spojení s touto metodou. Využití našla například na platformě Zynq v hledání obrazového filtru [5]. Zde byly použity dvouvstupé funkce (s dvěma multiplexory předchozích modulů a vstupů) s binárními funkcemi, sčítáním, sčítáním se saturací, posunem a dalšími operacemi. Ze všech představených metod se jedná o nejrychlejší způsob pro evoluci obvodů [6] oproti dynamické parciální rekonfiguraci. Nevýhodou tohoto zapojení je však větší náročnost na použité zdroje.

Kapitola 4

Implementace obvodů na úrovni tranzistorů

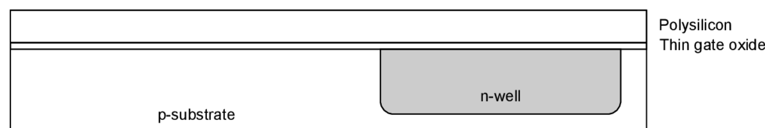
Technologie CMOS založená na využití tranzistorů s indukovaným kanálem označovaných souhrnně jako MOSFET tranzistory, představuje v dnešní době pravděpodobně nejčastěji používanou technologii pro implementaci VLSI systémů[1]. Vzhledem k tomu, že se práce věnuje návrhu číslicových obvodů na úrovni tranzistorů, představuje pochopení principu činnosti tranzistorů MOSFET jeden ze základních předpokladů pro vytvoření správné metody. Proto v této kapitole bude čtenář seznámen s funkcí MOSFET tranzistorů a s možnostmi jejich využití v obvodech. Tyto tranzistory patří do kategorie unipolárních tranzistorů, což znamená, že nosičem energie je vždy jen jeden typ náboje (buď elektrony, nebo „díry“).

4.1 Struktura MOSFET tranzistorů

Tyto tranzistory patří do kategorie tranzistorů s indukovaným kanálem. Princip jejich fungování spočívá v tom, že po připojení napájení na vstup označovaný jako *gate* (v bipolárních tranzistorech má podobnou funkci báze) dojde k přesunutí majoritních prvků k izolovanému hradlu. Tím se vytvoří vodivý kanál, přes který mohou procházet elektrony vedoucí proud mezi piny *source* (v bipolárních tranzistorech kolektor) a *drain* (u bipolárních emitor).

Pro správnou představu o funkci těchto prvků je nutné znát strukturu výroby VLSI systémů [53]. Výrobní proces bývá označen podle výrobního parametru λ. Tento parametr, který popularizovali Mead a Conway [25, 53], charakterizuje rozlišení výrobního procesu. Určuje polovinu minima délky kanálu tranzistoru. Tato délka je vzdálenost mezi elektrodami *source* a *drain* a je dána minimální šířkou vodiče na polysilikonu.

Popíšeme si základní postup výroby VLSI obvodu, který se nazývá litografie [15]. Prvotní materiál nutný k výrobě můžeme vidět na obrázku 4.1. V základním substrátu je přidáno

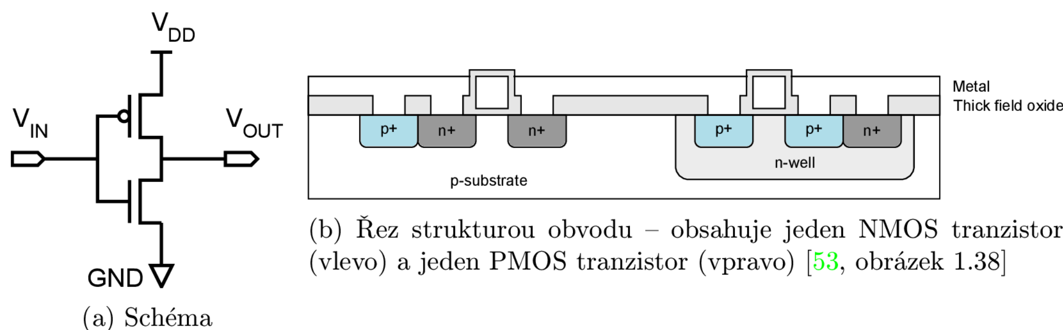


Obrázek 4.1: Prvotní materiál k vytvoření VLSI systému [53, obrázek 1.38]

více P prvků. Musí už být vytvořená oblast nazvaná *n-well*, kde je větší množství N prvků.

[1] Very large scale integration — systémů s vysokým stupněm integrace

V dalším kroku dojde k vyleptání vodičů. Dále se nanese maska z oxidu a v této masce se vyleptají díry k substrátu. Do substrátu v místech, kde není maska, se přivedou N prvky, čímž vzniknou oblasti N+. Oxidová maska se odstraní a stejný proces se opakuje pro oblasti P+. Na konci se nanese oxidová maska pro odizolování jednotlivých polí a rozlije se kovová vodící vrstva. Výsledný řez obvodem můžeme vidět na obrázku 4.2b. Obvod vykonává funkci jednoho ze základních číslicových hradel — hradla NOT, které invertuje vstupní signál a jehož schéma je znázorněno obrázku 4.2a.



(a) Schéma

(b) Řez strukturou obvodu – obsahuje jeden NMOS tranzistor (vlevo) a jeden PMOS tranzistor (vpravo) [53, obrázek 1.38]

Obrázek 4.2: Obvod realizující funkci NOT

Vzhledem k tomu, že je vhodné mít připojený substrát k napájení, jinak dochází k tzv. body efektu [3, str. 148], který bude popsán níže, připojujeme substrát i *n-well* část k napájení. Připojení provádíme přes oblasti s větší dotací stejných prvků, jako je v substrátu.

4.2 Popis obvodu

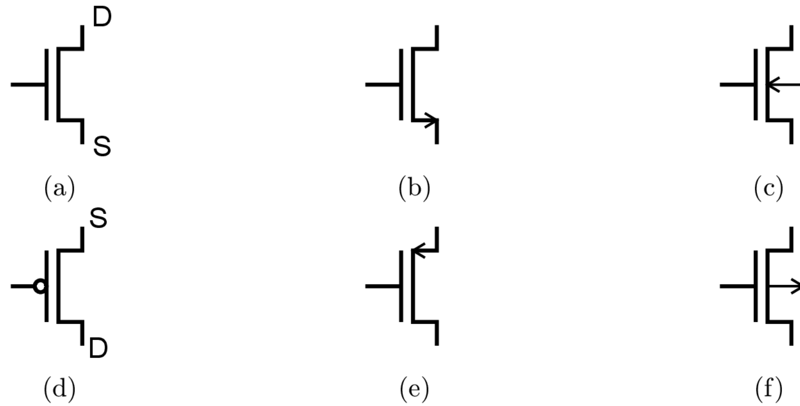
Při modelování VLSI systémů se typicky používá různá úroveň abstrakce. V dnešní době rozlišujeme tři úrovně abstrakce. Nejvyšší úrovní je tzv. architekturní či funkcionální popis. V této úrovni kombinujeme algoritmy, systémy, HW moduly a jednotlivé čipy. Nižší úrovní abstrakce je RTL[2] nebo logická abstrakce. Pro tento popis používáme programy, hradla, ALU jednotky, registry, moduly nebo některé fyzické buňky. Nejnižší úrovní, na kterou se zaměříme při našem návrhu, je obvodová abstrakce. Na straně behaviorálního (funkcionálního) popisu se často využívá booleovských výrazů a funkcí.

Nejvíce nás bude zajímat strukturální popis. Při použití této abstrakce budeme využívat jednotlivé tranzistory. Typicky používané schématické značky jsou znázorněny na obrázku 4.3. Rozdíl mezi NMOS a PMOS tranzistorem je to, jaký kanál se vytváří. Tento kanál může být buď pozitivní (u PMOS), který je vytvořen připojením nulového napětí k vstupu *gate*. U NMOS tranzistorů je tento kanál negativní, takže je vodivý po připojení kladného řídícího napětí. Zapojení jednotlivých prvků se provádí pomocí elektrotechnických schémat.

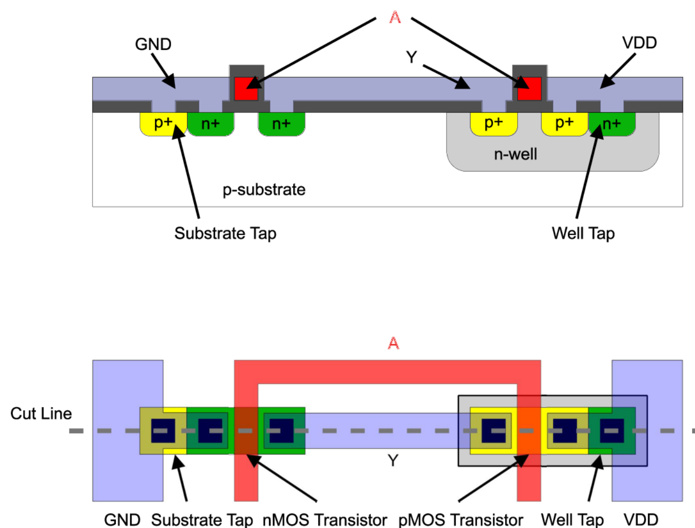
Na úrovni geometrické reprezentace (někdy nazývané jako fyzické), budeme vycházet ze způsobu výroby obvodu. Postup výrobu bude znázorněn na invertoru zobrazeném na obrázku 4.4.

Tento způsob popisu struktury není příliš vhodný, vzhledem k tomu, jak je náročný. Pro jednodušší aplikace, jako jsou jednotlivé buňky a hradla, je jednoduché použít některý z přímočarých způsobů nákresu. Tato rozvržení mohou být generována automaticky nebo ručně. Často se pro jednoduché návrhy používá v automatizovaných systémech způsob návrhu založený na „přímce rozptylu“. Tento styl návrhu se skládá ze čtyř horizontálních pruhů:

[2] Register Transfer Level



Obrázek 4.3: Schématické značky používané pro označení tranzistorů NMOS (a)–(c) a PMOS (d)–(e). Varianty (a) a (d) znázorňují typický MOSFET tranzistor s připojeným substrátem k zemi pro NMOS a k napájení pro PMOS, varianty (b) a (e) jejich odvozenou bipolární variantu s obdobným připojením substrátu. Varianty (c) a (f) znázorňují MOSFET tranzistor jako zařízení se čtyřmi elektrodami, a to *source, gate, drain* a *body* [3]

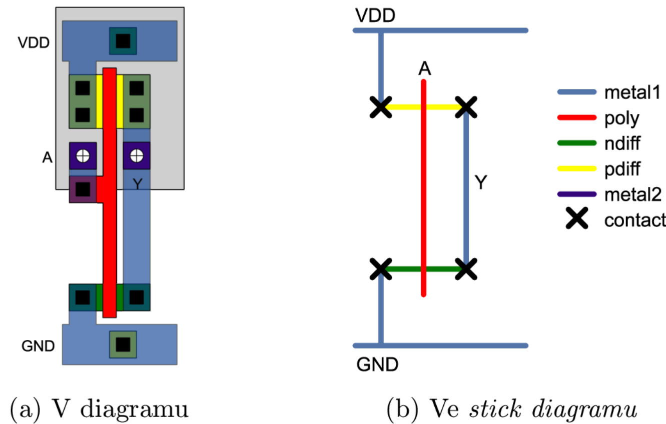


Obrázek 4.4: Struktura invertoru v řezu a v pohledu z vrchu [53]

kovové země ve spodní části, n-difuzní blok, p-difuzní blok a napájení v horní části. Spoje z polysilikonu jsou vedené vertikálně k určení *gate* tranzistorů. Na obrázku 4.5a můžeme vidět tento způsob návrhu tranzistorů. Fialově zvýrazněné spoje jsou tzv. bondy, které přivádí vstupy a výstupy prvků. Návrh je časově náročný, proto je důležité mít prostředky pro rychlé plánování. Samotné překreslení se provádí až po vlastním návrhu. Využíváme tzv. *stick diagramů* zobrazených na obrázku 4.5b.

Pomocí evolučního návrhu budeme řešit přechod mezi booleovskými výrazy jednotlivých prvků a tranzistorovým zapojením. Při návrhu můžeme vycházet z těchto základních požadavků

- Výkonnost — rychlost, napájení, funkčnost, flexibilita
- Velikost matrice



(a) V diagramu (b) Ve *stick diagramu*

Obrázek 4.5: Struktura invertoru [53]

- Doba nutná k návrhu
- Jednoduchost verifikace a testovatelnost

Všechny tyto požadavky nejsou splnitelné současně, proto se musíme rozhodnout, který z parametrů je pro nás nejdůležitější a s ohledem na něj provést optimalizaci.

Při klasickém návrhu vycházíme většinou z návrhové metodologie „divide and conquer“. V tomto návrhu rozdělíme komplexní systémy na jednodušší moduly. Můžeme používat tzv. *virtuální komponenty*, které se dále dělí na menší části. Často v tomto návrhu využíváme i IP jádra[3]. Důležitou vlastností hierarchického návrhu je modularita, kdy se části opakují a díky tomu vzniká menší počet submodulů. To zapříčiňuje jednodušší verifikaci výsledného hardwaru, protože počet testovaných částí je menší. Další důležitou vlastností je regularita, díky které popisujeme bloky na různých úrovních, a tyto bloky se opakují. Zjednodušuje to vlastní verifikaci, kdy je kontrola komplexnějších bloků jednodušší. Další požadovanou vlastností je lokalita. Díky té nedochází k vedení vodičů přes celý systém a jednotlivé bloky jsou soustředěny v jednom místě.

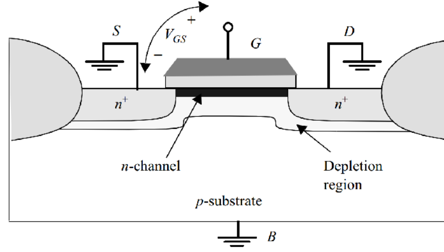
4.3 Princip činnosti MOSFET tranzistorů

Vzhledem k tomu, že chování tranzistorů je komplementární, omezíme se při popisu funkce tranzistorů pouze na tranzistory NMOS. V této části textu bude čtenář seznámen s funkcí tranzistorů a základními parametry obvodu. Text čerpá převážně z [31], kde lze nalézt další podrobnosti týkající se přesných matematických modelů tranzistorů.

Název MOSFET, zkráceně také jenom MOS, plyne z anglického názvu *metal–oxide–semiconductor field–effect–transistor*. První polovina názvu vyplývá ze struktury tranzistoru tvořené napařenými vrstvami oxidu železitého, jak bylo popsáno výše. Druhá polovina určuje, že se jedná o tranzistor řízený elektrickým polem (tj. napětím).

Schematicky znázorněnou situaci otevřeného tranzistoru NMOS můžeme vidět na obrázku 4.6. Připojením napájení na elektrodu *gate* došlo k přesunu elektronů blíž k elektrodě, čímž vznikl indukovaný kanál typu N. Přes tento kanál mohou procházet elektrony mezi elektrodami *gate* (G) a *source* (S). Vzhledem k tomu, že se „díry“ (jak označujeme pozitivní náboje v mřížce polovodičů, kde chybí elektrony) přesunuly směrem k připojení substrátu,

[3]Intellectual property

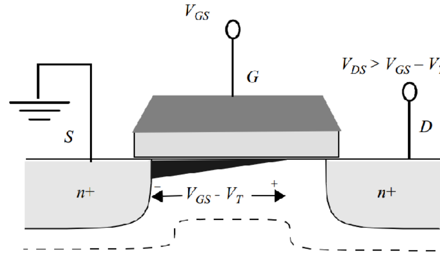


Obrázek 4.6: Funkce NMOS tranzistoru s pozitivním napětím V_{GS}[31]

vytvořila se okolo *n vrstev* tranzistoru vyprázdněná oblast *depletion region.* Tato oblast izoluje zbytek substrátu od indukovaného kanálu.

Napětí nutné k vytvoření plného indukovaného kanálu nazýváme **prahové napětí** a označujeme jej V_{tn} pro NMOS a V_{tp} pro PMOS tranzistory. Při tomto napětí přestává tranzistor lineárně zesilovat proud řídící elektrodou, ale otevře se úplně s minimálním odporem pro proud mezi elektrodami *source* (S) a *drain* (D). Tento stav nazýváme **režimem saturace** a při návrhu digitálních obvodů tranzistory většinou pracují v tomto režimu. Oba stavy jsou znázorněny na obrázcích 4.6, kde je kanál plně otevřen, oproti 4.7, kde je splněna podmínka nenasycení tranzistoru

$$V_{GS} - V_{DS} \leq V_T \tag{4.1}$$



Obrázek 4.7: Funkce NMOS tranzistoru, který není v režimu saturace [31]

V režimu saturace potom proud elektrodou *drain* můžeme vypočítat jako

$$I_D = \frac{k'_n}{2}\frac{W}{L}(V_{GS} - V_{tp})^2 \tag{4.2}$$

kde k'_n je konstanta daná technologií výroby, W je úměrná efektivní šířce kanálu a L jeho délce. Přestože to vypadá, že v režimu saturace se tranzistor chová jako ideální proudový zdroj, u kterého je proud mezi elektrodami *drain* a *source* konstantní bez ohledu na připojené napětí. Této abstrakce často využíváme, ovšem není to přesné. Efektivní délka indukovaného kanálu je modulována připojeným napětím V_{DS}, kdy zvyšující napětí způsobuje rozšíření vyprázdněné oblasti, čímž se zmenší indukovaný kanál. Potom celkový proud elektrodou *drain* můžeme vypočítat jako

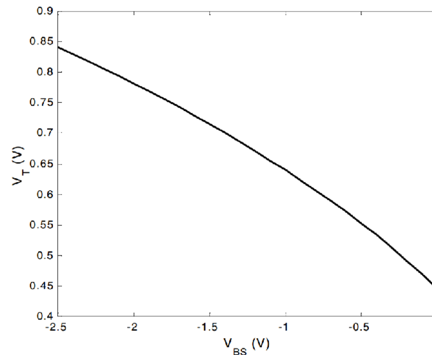
$$I_D = I'_D(1 + \lambda V_{DS}) \tag{4.3}$$

kde I'_D odpovídá ideálnímu výpočtu proudu ukázanému výše a λ je empirický parametr nazývaný *channel–length modulation.* Analytické vyjádření tohoto parametru je velmi složité

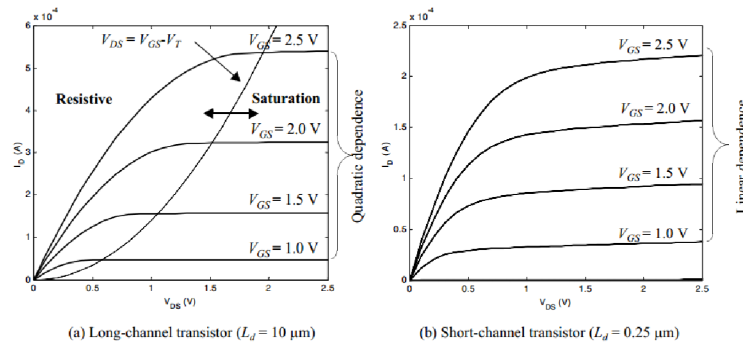
a navíc nepřesné. Mění se nepřímo se změnou délky kanálu. U tranzistorů s kratším kanálem reprezentuje vyprázdněná oblast u připojení *drain* větší část kanálu, a proto je kanálová modulace výraznější. Proto pokud je nutné vytvořit tranzistor, u kterého je výstupní proud stálejší, používají se tranzistory s větší délkou kanálu.

Pro tranzistory typu PMOS platí uvedené zákonitosti také s tím, že polarita napětí je opačná. Mimo to má však tranzistor typu PMOS oproti tranzistorům typu NMOS jednu negativní vlastnost, a sice to, že mobilita nábojů je nižší, což je způsobeno jejich opačnou polaritou. Z toho plyne to, že maximální proud tranzistorem je pouze 42 % maximálního proudu ekvivalentního NMOS tranzistoru vyrobeného stejnou technologií a se stejnými rozměry. Proto bývá ve stejném obvodě šířka kanálů W PMOS tranzistorů dvojnásobná, než šířka NMOS tranzistorů.

Jak bylo řečeno výše, napětí připojené k substrátu vytváří tzv. *body–effect*. Je to vlastnost tranzistoru, která způsobuje posun prahového napětí v závislosti na napětí mezi elektrodou *source* a substrátem. Tuto závislost pro jeden konkrétní typ tranzistorů je ukázána na obrázku 4.8.



Obrázek 4.8: Vliv připojení substrátu na prahové napětí (body–effect)[31]

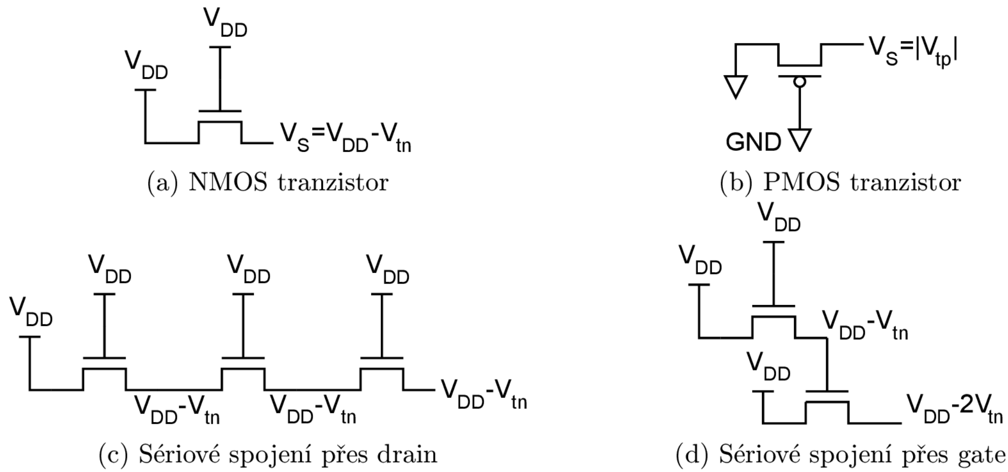


Obrázek 4.9: VA charakteristika NMOS tranzistoru pro (a) tranzistor s dlouhým kanálem (b) tranzistor s krátkým kanálem [31]

Výsledné elektrické chování tranzistoru je znázorněno na voltampérové charakteristice v otevřeném režimu na obrázku 4.9. Můžeme si všimnout jednak oblasti saturace, tak i dlouhého kanálu roste proud tranzistorem téměř kvadraticky, na rozdíl od tranzistoru s krátkým kanálem, kde roste pouze lineárně.

Prahové napětí nemá vliv pouze na režim saturace, dalším významným efektem je tzv. prahový pokles (*threshold drop*) [53]. Jedná se o situaci, kdy máme víc spojených tranzistorů

za sebe. NMOS tranzistory prochází logické nuly dobře, jedničky však degradují. Vezměme tranzistor s elektrodou *source* nastavenou na $V_s = 0$ a s *gate* nastavenou na $V_g = V_{DD}$. Vzhledem k tomu, že pro napětí mezi těmito dvěma elektrodami platí $V_{gs} > V_{tn}$, kde V_{tn} je prahové napětí tranzistoru NMOS, tak je tranzistor otevřený. Pokud však *source* V_s vzroste k hodnotě $V_i = V_{DD} - V_{tn}$, klesne napětí V_{gs} na hodnotu V_{tn} a tranzistor se přepne do zavřeného stavu. Díky tomu při průchodu logické jedničky nikdy nedojde k překročení hodnoty na výstupu $V_{DD} - V_{tn}$ (obr. 4.10a). Analogicky i u PMOS tranzistoru při průchodu logické nuly dostaneme na výstupu jeho prahové napětí $|V_{tp}|$ (obr. 4.10b).



(a) NMOS tranzistor

(b) PMOS tranzistor

(c) Sériové spojení přes drain

(d) Sériové spojení přes gate

Obrázek 4.10: Znázornění prahového poklesu [53]

Při spojování tranzistorů do série můžeme odvodit to, že při propojení *source* a *drain* elektrod je degradace stejná, jako u samotného tranzistoru (obr. 4.10c). Ovšem pokud použijeme degradovaný výstup tranzistoru pro řízení *gate*, snížení výstupního napětí bude dvojnásobné $V_s = V_{DD} - 2V_{tn}$, jak je znázorněno na obrázku 4.10d.

4.4 Simulace obvodů s tranzistory

Způsob simulace kombinačních obvodů se liší podle úrovně popisu. Na vyšších úrovních často používáme HDL jazyky, jakými je například VHDL nebo Verilog a diskrétní simulací vycházející z použité abstrakce (nejčastěji hradla). Na úrovni obvodů s tranzistory je však tento způsob popisu komplikovaný.

4.4.1 Analogová simulace

Pro získání přesného analogového výstupu využíváme typicky simulačního nástroje SPICE[4]. Takto nazýváme celou skupinu programů. Jejich vstupem bývá zpravidla popis propojení, tzv. *netlist*. Tento popis je textový a je psán ve formátu

```
{znak součástky}{identifikátor} {terminál 1} ...{terminál N} {parametry}
```

kde znak součástky může být například `V` pro zdroj, `C` pro kondenzátor, `L` pro cívku, `R` pro rezistor nebo `M` pro specifický model. Terminály představují jednotlivé elektrody součástky

[4]Simulation Program with Integrated Circuit Emphasis

a na jejich místo se napíše identifikátor vodiče, který chceme na danou elektrodu připojit. Vodiče není nutné definovat.

Popis tvorby netlistů je ukázán na hradle NOT zapsaném v CMOS logice na obrázku 4.11.

```
v1 vdd 0 3V
v2 vss 0 0V
v3 in vss pulse(0 3 0 100p 100p 1.9n 4n)

Mp1 vdd in out vdd pch l=0.35u w=20.0u
Mn1 vss in out vss nch l=0.35u w=10.0u

Cload out vss 200f

.MODEL nch NMOS
.MODEL pch PMOS

.TRAN 10p 12n
.end
```

Obrázek 4.11: Netlist hradla NOT pro SPICE

Jedná se o spojení dvou tranzistorů MOSFET s definováním parametrů délky kanálu a jeho šířky. Můžeme si všimnout toho, že v souladu s předchozí definicí má PMOS tranzistor dvojnásobnou šířku kanálu. Výstup obvodu je tvořen kondenzátorem.

V některých nástrojích, jako například *LTSpice*, máme kromě textového popisu propojení možnost použít i grafický editor. Velmi často používaným simulátorem je `NGSpice`[5] s otevřeným zdrojovým kódem.

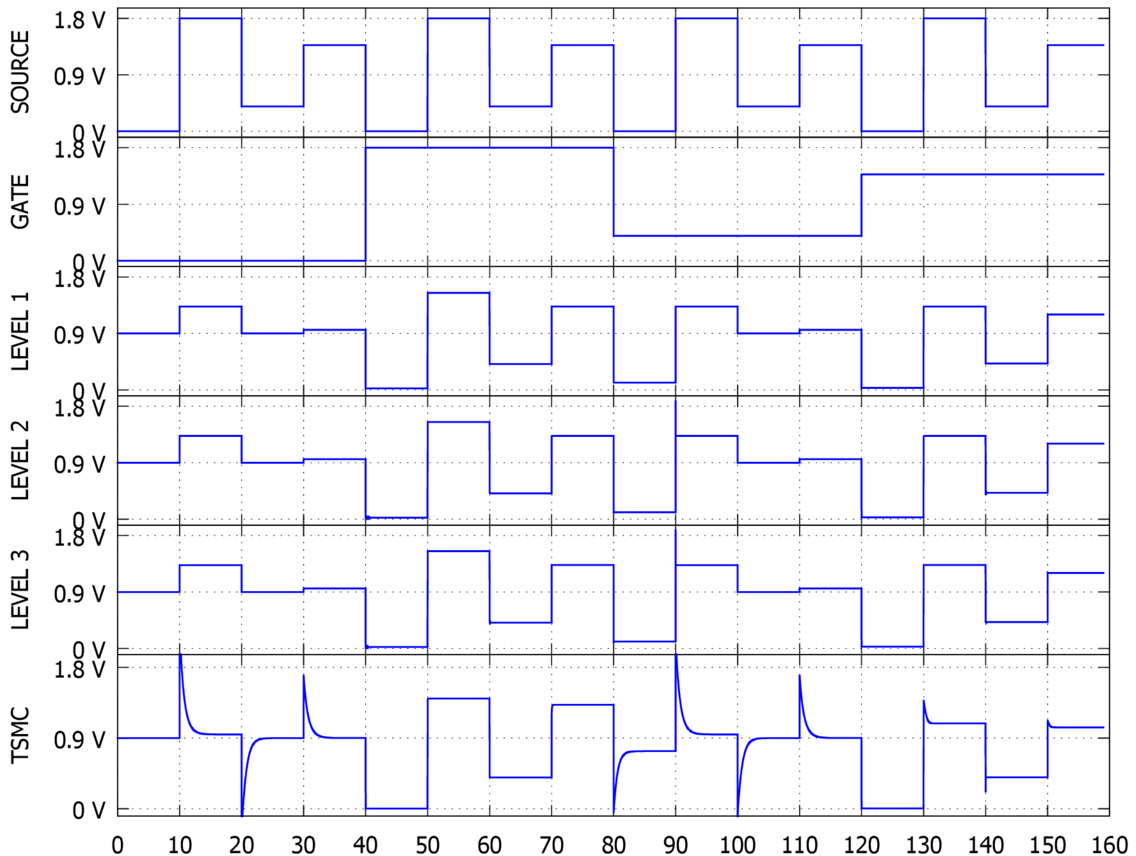
SPICE má implicitně vestavěné tři úrovně modelů vybrané parametrem `LEVEL` v definici modelu. Přesnější informace o jednotlivých modelech je možné najít v [31]. Všechny tyto modely však nereflektují současný vývoj tranzistorů s krátkým kanálem. Proto jsou doporučeny pouze po prvotní analýzu. Úroveň 1 se typicky používá tehdy, když je důležitější rychlost než přesnost simulace. U digitálních obvodů dosahujeme polovičního času simulace, než pro vyšší úroveň. Tranzistory jsou implementovány pomocí *Shichman-Hodges* modelu. Úroveň 2 je založena na geometrickém modelu, který využívá detailní fyzikální vlastnosti. Nevýhodou tohoto modelu je to, že model založený na fyzikálních vlastnostech je velmi komplexní a nepřesný. Úroveň 3 je poloempirický model. Kombinuje analytické a empirické výpočty a využívá měřená data z konkrétních tranzistorů k určení chování.

Jméno parametru	**Symbol**	**SPICE**	**Jednotky**	**Výchozí**
Délka kanálu	L	`L`	m	-
Efektivní šířka kanálu	W	`W`	m	-
Plocha *source*	$AREA$	`AS`	m^2	0
Plocha *drain*	$AREA$	`AD`	m^2	0
Obvod *source*	$PERIM$	`PS`	m	0
Obvod *drain*	$PERIM$	`PD`	m	0
Koeficient difůze *source*		`NRS`	-	1
Koeficient difůze *drain*		`NRD`	-	1

Tabulka 4.1: Nejčastěji nastavované parametry SPICE MOSFET tranzistorů [31]

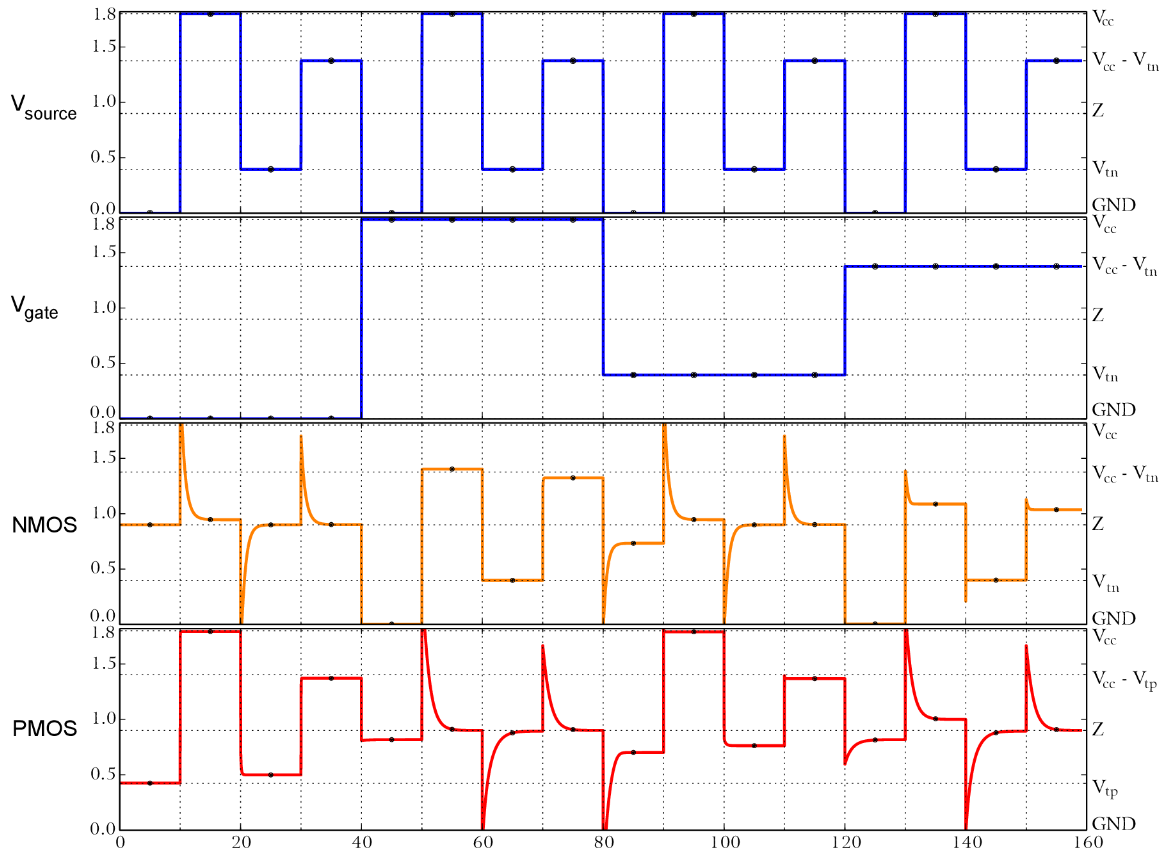
[5] http://ngspice.sourceforge.net/

Pro přesnější simulaci se využívá model tranzistorů, který byl do nástroje SPICE převzat později, a to Berkeley Short-Channel IGFET Model označovaný jako BSIM3V3. Tato úroveň popisu je označena jako `LEVEL` 49 a obsahuje přes 200 parametrů, jejichž vliv je podrobně popsán v literatuře [3]. V praxi se často postupuje tak, že se při uvedení nové technologie vytvoří prototyp, podle něj se změří parametry pro simulátor a tak vznikne přesný model jednotlivých elementů. Návrhář zpravidla definuje pro použité tranzistory parametry uvedené v tabulce 4.1. Parametry `L`, `W`, `AS`, `AD`, `PS` a `PD` vyplývají z geometrického uspořádání a jejich základní vliv popsán v kapitole 4.3. Hodnotami `NRS` a `NRD` násobí rezistivitu vrstev popsanou v modelu tranzistoru. Tyto koeficienty se využívají pro přesné doladění parazitního sériového odporu jednotlivých elektrod každého tranzistoru.



Obrázek 4.12: Chování simulátoru SPICE pro různé úrovně modelů tranzistorů NMOS

Výsledné chování tranzistorů NMOS se stejnými geometrickými parametry, avšak s rozdílnou úrovní modelu, je znázorněno na obrázku 4.12. V charakteristice je znázorněno napětí *drain*, když na elektrody *source* a *gate* bylo připojováno různé napětí. Výstupní napětí je spojeno přes napěťový dělič, který je tvořen dvěma rezitory o odporu 1 MΩ. Proto stav vysoké impedance odpovídá polovině napětí, 0,9 V. Je vidět, že pro první úroveň pouze degraduje hodnoty o prahové napětí. Úrovně 2 a 3 se chovají podobně, ovšem pro tranzistor zavřený špatnou napěťovou úrovní na elektrodě *gate* se objevuje na výstupu napěťový skok. Simulace v technologii TSMC se od předcházejících liší zohledněním kapacity a dalších vnitřních zpoždění, čímž dochází k degradaci hran. Dále v případě degradované úrovně logické 0 na elektrodě *gate* a silné logické úrovně logické 1 je tranzistor simulovaný v tomto modelu zavřený, přičemž v úrovních 1 – 3 je mírně otevřen.



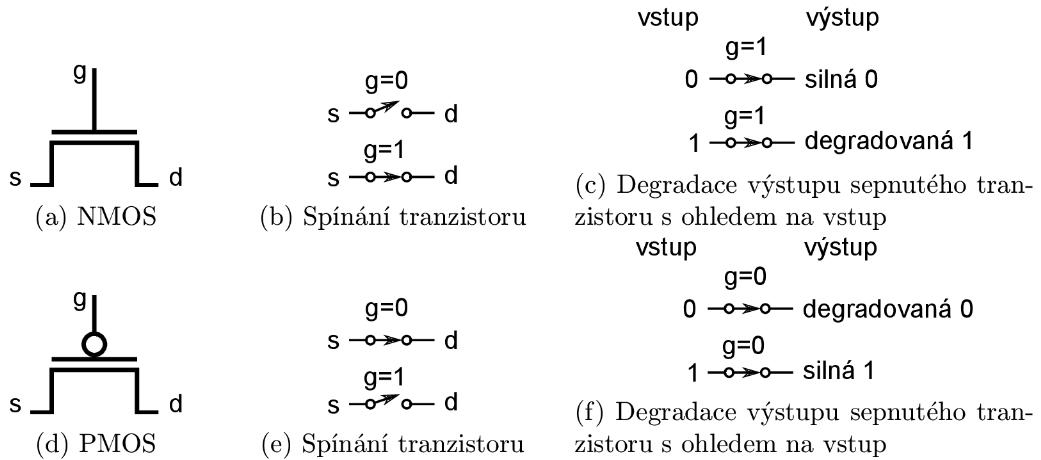
Obrázek 4.13: Technologie TSMC 0.25 μm v simulátoru SPICE

Pro odzkoušení vlastností tranzistorů byl v simulátoru SPICE simulován obvod s jedním tranzistorem v technologii TSMC 0.25 μm. Průběh této simulace je znázorněn na obrázku 4.13. Můžeme vidět, že se jednotlivé napěťové úrovně i zpoždění odlišují. Elektrody *source* a *gate* byly stimulovány pomocí napěťových zdrojů, *drain* byla stejně jako v předchozím případě připojena k napěťovému děliči z rezistorů v poměru 1 MΩ : 1 MΩ. Díky tomuto děliči odpovídá stav vysoké impedance Z hodnotě $\frac{1}{2}V_{DD} = 0,9\ V$. V čase od 0 do 40 ns je napětí přivedené na elektrodu *gate* nulové, což znamená, že tranzistor NMOS je zavřený a PMOS je otevřený s tím, že obě hodnoty logické 1 přenáší beze změny, ovšem silnou i degradovanou 0 degraduje na napětí $|V_{tp}|$. V čase od 40 do 80 ns je situace analogická. V dalším kroku, čase 80 až 120 ns, lze sledovat, že tranzistor NMOS není úplně uzavřen a objevuje se zde drobná odchylka od stavu vysoké impedance. Tranzistor typu PMOS je otevřen a hodnoty logické 1 jsou zase přenášeny beze změny. Hodnota logické 0 je však dvojitě degradovaná. V následujícím kroku je situace taktéž analogická. Můžeme také sledovat, že dvojitě degradovaná hodnota 0 odpovídá hodnotě $|2V_{tp}|$, která je téměř nerozlišitelná od hodnoty dvojitě degradované logické 1, která odpovídá $V_{CC} - 2V_{tn}$.

4.4.2 Diskrétní simulace

Můžeme-li si dovolit nižší přesnost simulace, je možné chování tranzistoru zjednodušit a simulovat obvod s využitím diskrétní simulace. Vyjdeme z chování tranzistorů odvozené z obrázku 4.13. Při této abstrakci používáme všechny tranzistory jako spínače. Jak bylo ukázáno, nebude však stačit pouze tříhodnotová logika. Vzhledem ke struktuře tranzistoru

docházÍ k tzv. oslabené hodnotě, jak můžeme vidět na obrázku 4.14. V nesepnutém stavu

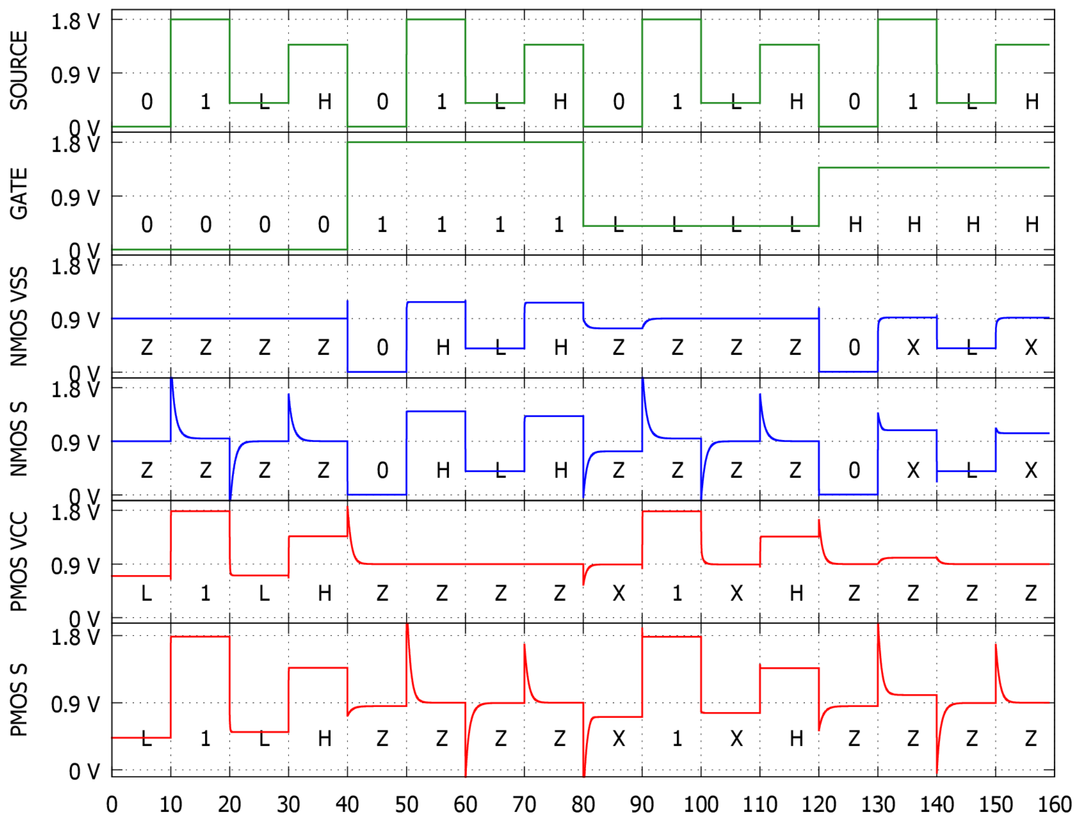


(a) NMOS (b) Spínání tranzistoru (c) Degradace výstupu sepnutého tranzistoru s ohledem na vstup

(d) PMOS (e) Spínání tranzistoru (f) Degradace výstupu sepnutého tranzistoru s ohledem na vstup

Obrázek 4.14: Použití tranzistorů jako spínačů [53]

na obrázcích 4.14b a 4.14e nedochází přenosu žádné energie, proto tento výstup označujeme jako stav vysoké impedance *Z*. Pokud připojíme na řídící vstup *gate* napětí, které sepne, výstupní hodnota bude závislá na vstupní hodnotě, jak vidíme na obrázcích 4.14c a 4.14f. Je nutné vyřešit to, co se stane po připojení vodiče ve stavu vysoké impedance *Z* na *gate*. V tomto případě výstup bude nedefinovaný *X*, protože bude záviset na konkrétních vlastnostech použitého MOSFET tranzistoru.



Obrázek 4.15: Technologie TSMC v simulátoru SPICE s rozdílným připojením substrátu

Chování tranzistorů je ovlivněno i připojením substrátu, které bývá typicky řešeno dvěma způsoby — připojením k elektrodě *source*, anebo k jednomu z napájení. Vliv tohoto připojení na modelu tranzistoru v analogovém simulátoru je vidět na obrázku 4.15. Testování bylo prováděno pro substrát připojený k *source* elektrodě (v grafu označeno písmenem S) a se substrátem připojeným k zemi V_{SS} pro NMOS, respektive napájení V_{CC} pro PMOS. Můžeme si všimnout, že pro oba druhy připojení je chování identické. Pro substrát připojený k elektrodě však při změně napěťové úrovně na elektrodě *source* dochází ke krátkému sepnutí tranzistoru. Dále je vidět, že v tomto případě je úroveň X, označující dvojitě degradovanou – v druhém případě nerozlišitelnou úroveň, odlišná.

Aby byly omezeny nežádoucí špičky signálů při přepínání napětí na elektrodě *source*, předpokládá se při použití výše uvedených zákonitostí následující připojení čtvrté elektrody — u NMOS připojíme *p-substrát* k zemi *GND* a u PMOS připojíme *n-well* ke kladnému napájecímu napětí. Tento postup je doporučen i v literatuře [53].

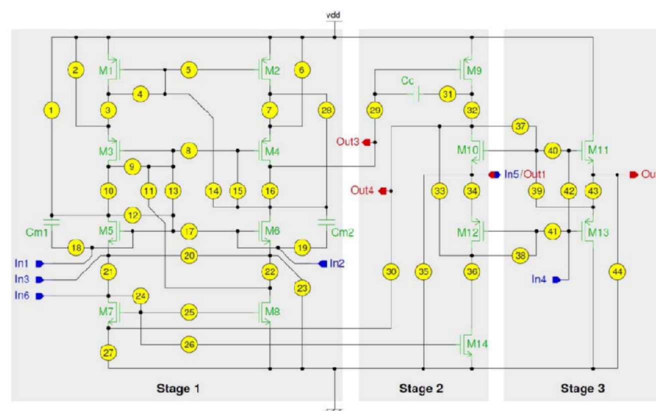
Kapitola 5

Využití evolučních technik v návrhu obvodů s tranzistory

Obvody s MOSFET tranzistory jsou velmi specifické a jejich návrh pomocí CGP je odlišný od návrhu kombinačních obvodů za použití hradel, sčítaček a dalších částí. V této kapitole si představíme několik stávajících metod návrhu digitálních obvodů. Dále zde bude podrobně popsán návrh metody nové.

5.1 Evoluce ve specializovaných obvodech

Návrh digitálních obvodů na úrovni tranzistorů je možné provádět pomocí pravé (*intrinsic*) evoluce. Pro tento návrh byly vytvořeny čipy FPTA[1], které integrují buňky tranzistorů, u kterých dynamicky podle konfigurace měníme propojení [50]. Evoluční algoritmy jsou využity k vytváření obvodů pro tyto čipy. Nejčastěji jsou používané dvě architektury těchto čipů. První je FPTA0, FPTA1 a FPTA2 z laboratoře NASA's Jet Propusion Laboratory (JPL). Druhou architekturou je FPTA a FPTA–2 z Univerzity Heidelberg.



Obrázek 5.1: Architektura buňky čipu JPL FPTA2, kolečka znázorňují propojky [50, 40]

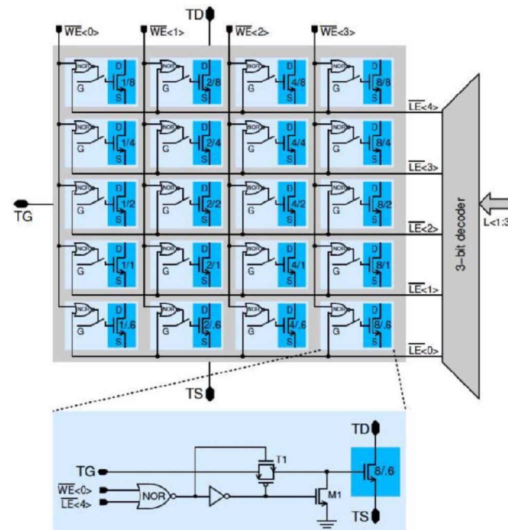
Čipy FPTA0 a FPTA1 z laboratoře JPL [39] jsou předchůdci moderní architektury FPTA2 [40]. FPTA0 se skládá ze samostatných buněk, které integrují 8 tranzistorů propojených pomocí spojek. Další dva čipy FPTA1 a FPTA2 využívají stejnou architekturu

[1]Field Programmable Transistor Array

zobrazenou na obrázku 5.1 založenou na návrhu operačních zesilovačů s dvouúrovňovým výstupem. Architektura FPTA1 využívá 12 buněk a byla vytvořena jako prototyp k architektuře FPTA2, která je složena z mřížky 8×8 buněk. Buňky čipu FPTA2 jsou připojeny ke čtyřem svým sousedům.

Aplikace JPL FPTA čipů je zaměřena na obnovení funkcionality v extrémním prostředí (například teploty [37] nebo radiace ve vesmíru [38]).

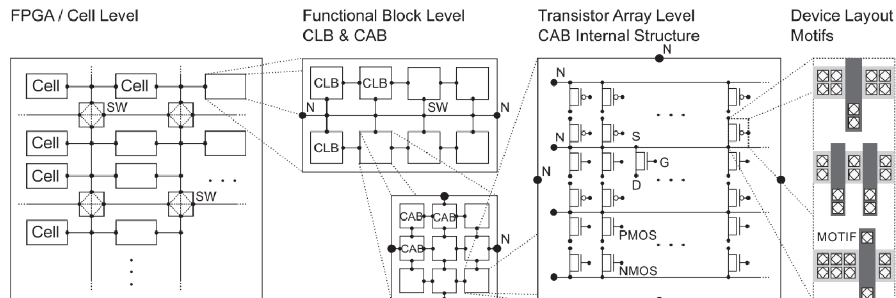
Oproti tomu se FPTA Heidlbergu skládá z pole 16×16 programovatelných MOS tranzistorů [21]. Programovatelná tranzistorová (PT) buňka (obrázek 5.2) obsahuje matici 5×4 MOS tranzistorů s různou délkou a šířkou kanálů, které sdílí stejné připojení elektrod. Spínáním rozdílných podmnožin matic měníme tedy vnitřní parametry tranzistorů a tím i jejich vlastnosti. Jednotlivé PT jsou připojeny ke čtyřem nejbližším sousedům.



Obrázek 5.2: Architektura tranzistorové buňky čipu FPTA–2 [50, 21]

Tato architektura představuje tedy jednu z architektur s největšími možnostmi nastavení [50]. Na této platformě byla realizována řada aplikací obsahující logická hradla, analogové filtry, komparátory, DA a AD převodníky a operační zesilovače [22, 24, 23].

Jednou z nejnovějších architektur je architektura PAnDA (Programmable Analog and Digital Array) [50]. Jedná se o nedávno představenou architekturu, tudíž ještě nebylo publikováno mnoho prací využívající tuto platformu.



Obrázek 5.3: Architektura čipu PAnDA [50]

Na obrázku 5.3 můžeme vidět konceptuální návrh této architektury. Úroveň buněk *cell*

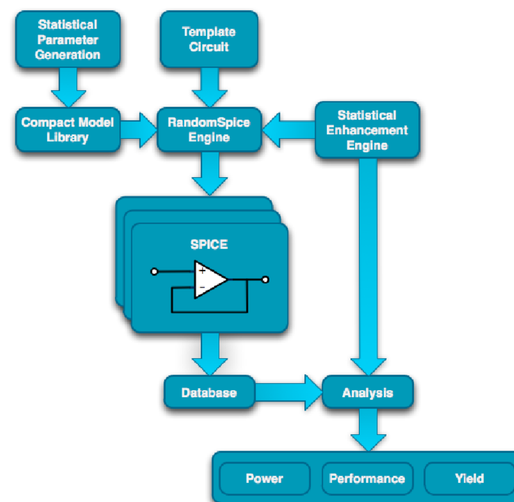
a jejich bloků *CLB* reprezentují shlukování a využívání komponent, jaké vidíme v dnešních komerčních FPGA čipech. Tranzistorové pole *Transistor Array* odpovídají konceptu PT v FPTA čipech Univerzity Heildberg. *CAB* úroveň ukazuje komunikační vrstvu mezi FPGA a FPTA návrhem a umožňuje sdružování bloků tranzistorových polí tak, aby mohly být konfigurovány ve formě logických funkcí. Kombinování těchto úrovní je unikátní v této platformě a nemůžeme je najít v žádném z dnešních FPGA. Na rozdíl od nich PAnDA umožňuje konfiguraci nejen na digitální úrovni, tak i na úrovni tranzistorové, a tím představuje jednu z nejmodernějších platforem pro rekonfigurovatelné obvody.

5.2 Evoluce za použití simulátoru

Další možností pro návrh digitálních obvodů s tranzistory je využití simulátoru, který pomocí přesného modelu vyhodnocuje chování obvodu. Výhodou tohoto přístupu je rychlost vyhodnocení oproti pravé evoluci a také to, že jsme schopni využívat obecných zákonitostí a nenalezneme řešení specifické pouze pro daný čip. Simulaci můžeme provádět pomocí analogového simulátoru, nejčastěji typu SPICE, anebo s využitím diskrétního simulátoru.

5.3 Analogová simulace

Jak již bylo řečeno, simulace pomocí nástroje *SPICE* je časově náročná. Někdy však bývá nutné přesně vyhodnotit funkci obvodu. Aby byl proces evoluce zkrácen, provádí se výpočet paralelně na počítačovém clusteru. Při tomto přístupu bývají často využívány nástroje typu *Randomspice* [8]. Jedná se o nástroj, který provede mutaci obvodu a předá výsledek nástroji *SPICE*, jak je znázorněno na obrázku 5.4.

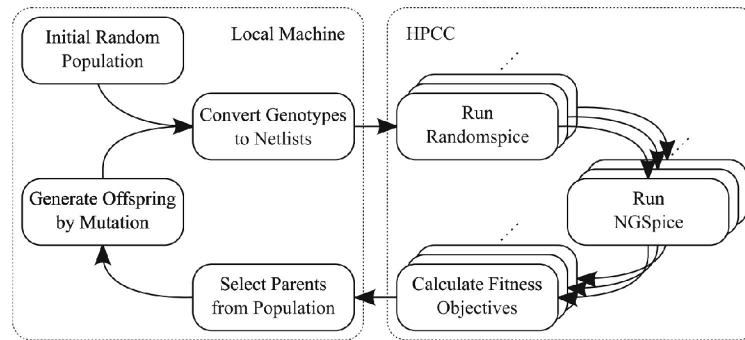


Obrázek 5.4: Ukázka fungování nástroje *Randomspice* v evolučních algoritmech [8]

Evoluční systém potom výsledky zpracuje, spočítá fitness a vybere rodiče pro následující generaci. Vzhledem k velké výpočetní náročnosti tohoto přístupu simulace se často využívá HPCC[2]. Výše popsaný proces byl použit například při hledání modulů nových CMOS obvodů s ohledem na příkon a zpoždění [47] nebo při hledání CMOS obvodů s ohledem

[2]High-performance computer cluster — vysoce výkonný počítačový cluster

na variabilitu [48]. V druhém jmenovaném nebylo využito CGP, ale jednalo se o genetický algoritmus. Princip vyhodnocení je vidět na obrázku 5.5.



Obrázek 5.5: Princip evolučního algoritmu s využitím HPCC [47]

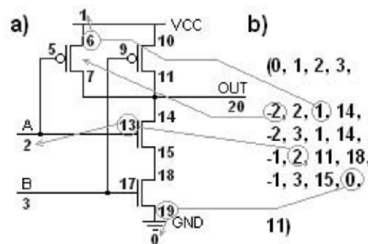
Na začátku vygenerujeme počáteční generaci. Tu potom předáme clusteru počítačů, které provedou mutaci a vyhodnotí výsledky pomocí *SPICE* simulace. Nástroj *Randomspice* předá řídicímu počítači analýzu průběhu funkce, energie a počtu použitých elementů [8]. Z těchto výsledků se spočítá fitness jedince s ohledem na optimalizační parametr. Ve výše uvedené práci se jedná o multikriteriální optimalizaci s ohledem na zpoždění a spotřebu elektrické energie. Na řídícím PC se nakonec vybere rodič pro následující generaci a proces se opakuje.

Výsledkem běhu evolučního algoritmu s ohledem na variabilitu byly jednotlivé moduly, které vykazovaly správnou funkci napříč různými technologiemi výroby.

5.3.1 Diskrétní simulace

Simulace obvodu pomocí SPICE simulátoru je velmi přesná, ale časově velmi náročná. Proto byl v práci L. Žaloudka [1] výpočet fitness hodnoty pomocí přesného vyhodnocení nahrazen vlastním simulátorem.

Pro reprezentaci dat byla využita upravená CGP reprezentace definovaná v [29]. Jednotlivé bloky představují tranzistory, jejich funkce se přepíná mezi NMOS a PMOS. Pro každou elektrodu tranzistoru se volí vodič, se kterým je spojen. Převod zapojení je znázorněn na obrázku 5.6



Obrázek 5.6: (a) klasické zapojení dvouvstupého hradla NAND (b) chromozom představující zapojení tohoto obvodu [1]

Pro propojení existují další omezení

- Minimálně jeden tranzistor musí být připojen k V_{CC} a minimálně jeden k GND

- *Source* a *drain* elektrody mohou být připojeny pouze k napájecímu napětí nebo k jiným *source/drain* elektrodám
- Z důvodu degradace signálu není možné připojit V_{CC} k NMOS tranzistoru a GND k PMOS tranzistoru
- Není možné připojit výstup přímo k napájení
- Elektrody *gate* mohou být připojeny k primárním vstupům. Ve standardních CMOS obvodech jsou tranzistory s řídicími vstupy připojenými jinak použity pouze jako invertory vstupů a výstupů
- Předpokládá se, že čtvrtá elektroda tranzistoru představující substrát tranzistoru je připojena k V_{CC} u PMOS a k GND u NMOS tranzistoru

Stav vodičů při simulaci je definován pomocí následujících šesti hodnot: silná a slabá 1, silná a slabá 0, nedefinovaná hodnota a stav vysoké impedance. Při simulaci se prvně nastaví všechny terminály tranzistorů do stavu vysoké impedance. Dále se rozdistribuuje signál silné 1 pro V_{CC} a silné 0 pro V_{GND}. Podle vstupního vektoru dojde k nastavení vstupních napětí a k jednotlivému nastavování tranzistorů. Simulátor nerozlišuje mezi vstupy *source* a *drain*, protože z hlediska elektrického se jedná o identicky chovající se vstupy. Pokud dojde ke změně stavu tranzistoru, je tato změna propagována dál. Tranzistory degradují některé hodnoty podle fyzikálního modelu ukázaného v kapitole 4.4. Zkraty jsou identifikovány už na tranzistorech.

V práci je využita evoluční strategie $1 + \lambda$, kde se λ pohybuje kolem 8 – 40. Cílem práce bylo vyhodnotit navrženou metodu v úloze evolučního návrhu základních logických hradel. Přestože evoluce vykazovala poměrně vysokou úspěšnost, chování nalezených řešení neodpovídala skutečnosti. Výsledně v 33 % běhích algoritmu pro NAND/NOR obvody dával simulátor stejné výsledky jako *SPICE* Chyba v jedné kombinaci vstupního signálu byla v 55 % běhů, chybu ve dvou kombinacích bylo možné najít v 11 % běhů a čtyři chyby v 1 % běhů. Tato chybovost je způsobena degradací tvaru výstupního signálu.

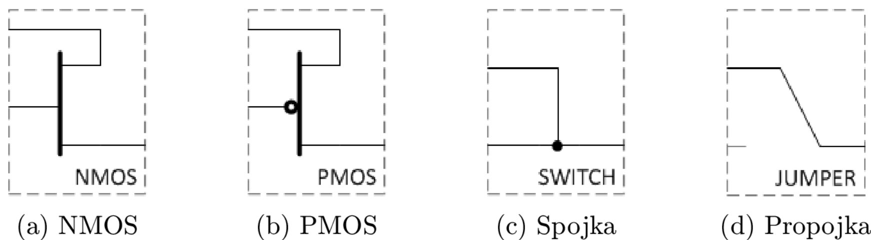
5.4 Navržená metoda

Reprezentace chromozomu použitá v práci s diskrétním simulátorem, jejíž činnost byla demonstrována v kapitole 5.3.1, není vhodná pro simulaci v hardwaru. Navíc oproti jiným reprezentacím neumožňuje připojit signál na elektrodu *gate* tranzistoru. Další reprezentaci použitou v [29], která opět využívá pevný počet tranzistorů, u nichž není omezená jejich konektivita a které se dynamicky navzájem propojují, není také možné použít. Základním problémem těchto reprezentací při přenosu do HW je nutnost implementace úplného propojení, tj. každého elementu s každým. Proto musíme vytvořit jiný způsob zobrazení reálného obvodu.

5.4.1 Reprezentace obvodu

V našem případě budeme obvod na úrovni tranzistorů reprezentovat pomocí množiny dvouvstupých elementů ukázaných na obrázku 5.7, které mohou vykonávat jednu z následujících funkcí: NMOS (ve schématech označený jako N) nebo PMOS tranzistor (ve schématech označený jako P), spoj více signálů (označený SWITCH nebo S) nebo funkci propojky (označenou JUMPER nebo J). Kódování bude vycházet z CGP, které používá celá čísla.

Výhodou tohoto přístupu bude fakt, že tento způsob uspořádání a zakódování bude možné využít pro efektivní HW realizaci akcelerační jednotky. Propojovací element má význam



(a) NMOS (b) PMOS (c) Spojka (d) Propojka

Obrázek 5.7: Použité elementy v kartézské mřížce

při přenosu do hardwarové verze kvůli omezení parametru LBACK.

Abychom zbytečně negenerovali nevalidní zapojení, která nemohou v praxi pracovat korektně, je vhodné zavést strukturální omezení. Jednotlivé podmínky jsou definovány v tabulce 5.1. Nejdůležitější omezení vyplývá z toho, že nemůžeme propojit vstupní signál k libovolnému výstupu, což může nastat s využitím spojky nebo propojky. Pokud bychom tato zapojení tolerovali, museli bychom kontrolovat, zda nedošlo ke zkratu na úrovni vstupního signálu. Navíc takovéto zapojení nemá v praxi žádný význam, protože by došlo buď ke zkratu, nebo by všechny signály byly vždy stejné a jejich spojení by bylo zbytečné.

(a) Omezení tranzistorů

Pin	**Možné připojení**
Drain (vstup A)	V_{CC} GND Výstup elementu V_{INx}[i]
Gate (vstup B)	V_{INx} Výstup elementu
Source (výstup)	Bez omezení

(b) Omezení spojů a propojek

Pin	**Možné připojení**
Vstup A	Výstup elementu
Vstup B	Výstup elementu
Výstup	Bez omezení

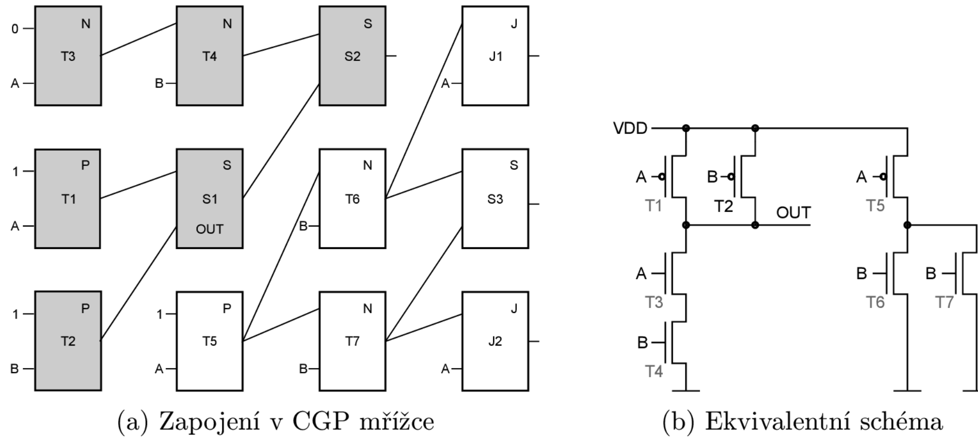
Tabulka 5.1: Omezení možností propojení jednotlivých typů elementů CGP

[i] Může způsobovat větší napěťovou zátěž vstupů, záleží na povolení vývojáře.

5.4.2 Simulace obvodu

Celá simulace se skládá ze dvou kroků, a to z určení aktivních elementů a poté plnění primárních vstupů jednotlivými vektory.

Prvním krokem je určení aktivních elementů. Pokud bychom tuto detekci neprovedli, některé obvody, ač validní, by kvůli nepoužívané části obvodu produkovali zkrat. Tento jev je demonstrován na obrázku 5.8, který ukazuje zapojení hradla NAND–2 v mřížce CGP. Můžeme si všimnout, že při vstupním vektoru $A = 0$ a $B = 1$ dochází ke zkratu na tranzistorech T_5, T_6 a T_7, ačkoliv výstup je správně a část, ve které dochází ke zkratu, se nikdy nebude podílet na výstupu. V CGP mřížce můžeme vidět šedě vyznačené elementy, které by měly být aktivní, když výstup je připojen k elementu S_1. Proces detekce aktivních prvků probíhá rekurzivně a jako první se označí za aktivní elementy připojené k výstupům obvodu, v našem případě pouze S_1. Vzhledem k tomu, že se jedná o spojku (stejný stav by nastal i u propojky), aktivují se elementy připojené nejen svým výstupem k používanému



(a) Zapojení v CGP mřížce (b) Ekvivalentní schéma

Obrázek 5.8: Proces detekce aktivních prvků na obvodu NAND–2

vstupu elementu, ale i ty, jejichž vstup je připojen k výstupu tohoto elementu. U elementů vykonávajících funkci tranzistoru se označí za aktivní pouze elementy, které mají připojený svůj výstup ke vstupu analyzovaného tranzistoru. Proto se v následujícím kroku označí za aktivní elementy T_1, T_2 a S_2. V dalším kroku se postupuje obdobně a v našem případě je aktivován pouze T_4. Jako poslední je označen za aktivní tranzistor T_3.

Při vyhodnocení se předpokládá, že všechny elementy pracují synchronně, tzn. všechny současně spočítají svůj výstup a pak najednou dojde k přenosu informací mezi jednotlivými prvky. Tento průběh odpovídá reálnému chování obvodů, kdy všechny tranzistory pracují současně. Podle svých vstupů (včetně zpětné vazby) vypočítají novou hodnotu, kterou uloží do vnitřní paměti. Po výpočtu všech elementů dojde k propagaci vypočtených hodnot na vstupy spojených prvků. Simulace probíhá přesný počet taktů, který odpovídá součtu počtu sloupců a počtu prvků typu spoj.

Jakmile se vyhodnotí výstup pro jeden vstupní vektor, dojde k vynulování všech elementů (kromě stavu aktivace). Celkem se tento proces opakuje 2^{vstupy}-krát. Díky synchronnímu přístupu výpočtu hodnot elementů je tento proces ideální pro přenos do hardwaru. Časová složitost problému však bude pořád $t = O(2^n)$.

Vstup source	**Vstup gate**	**Výstup drain**
Silná 0	Silná nebo slabá 1	Silná 0
Slabá 0	Silná nebo slabá 1	Slabá 0
Silná nebo slabá 1	Silná 1	Slabá 1
Silná nebo slabá 1	Slabá 1	Nedefinovaná hodnota
Stav vysoké impedance	Slabá nebo silná 1	Stav vysoké impedance[i]
Cokoliv	Slabá nebo silná 0	Stav vysoké impedance
Ostatní	Ostatní	Nedefinovaná hodnota

Tabulka 5.2: Výpočet výstupu NMOS tabulky

[i] Podle literatury [53] je toto validní stav, jen může nekdy vést k neoptimálnímu řešení u obvodů, kde vyžadujeme na výstupu stav vysoké impedance – například při vývoji obvodu s třístavovým výstupem. Obvody s takovýmto výstupem však nebudeme navrhovat, a proto toto omezení není nutné řešit.

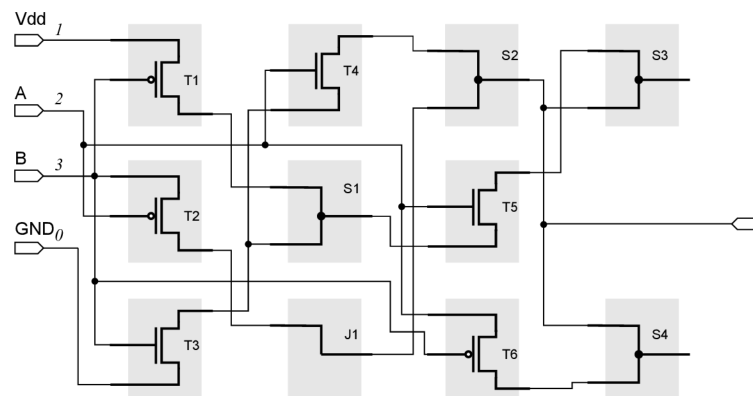
Simulace pracuje s vícehodnotovou logikou. Mezi základní signály patří silná hodnota a slabá (degradovaná) hodnota signálu logické 1 a logické 0. Dvojitě degradovaná hodnota

není implementována, protože v dnešních technologiích výroby VLSI obvodů bývá těžko rozlišitelná. Kromě těchto hodnot se využívá hodnoty reprezentující neznámou hodnotu U, která je v obvodech do té doby, než je možné vypočítat podle vstupů výstupní hodnotu. Další hodnotou je stav vysoké impedance Z a nedefinovaná hodnota X.

Element s funkcí spoje a propojky pouze propojují signály, případně při spojení slabé a silné hodnoty dojde k zesílení. Chování bloků NMOS tranzistorů je popsáno v tabulce 5.2. Tranzistor může vést proud i směrem od *drain* k *source*. V tomto případě dojde jen k záměně sloupců. Pro PMOS tranzistor je chování analogické se záměnou logické 0 za logickou 1 a naopak.

5.4.3 Příklad simulace

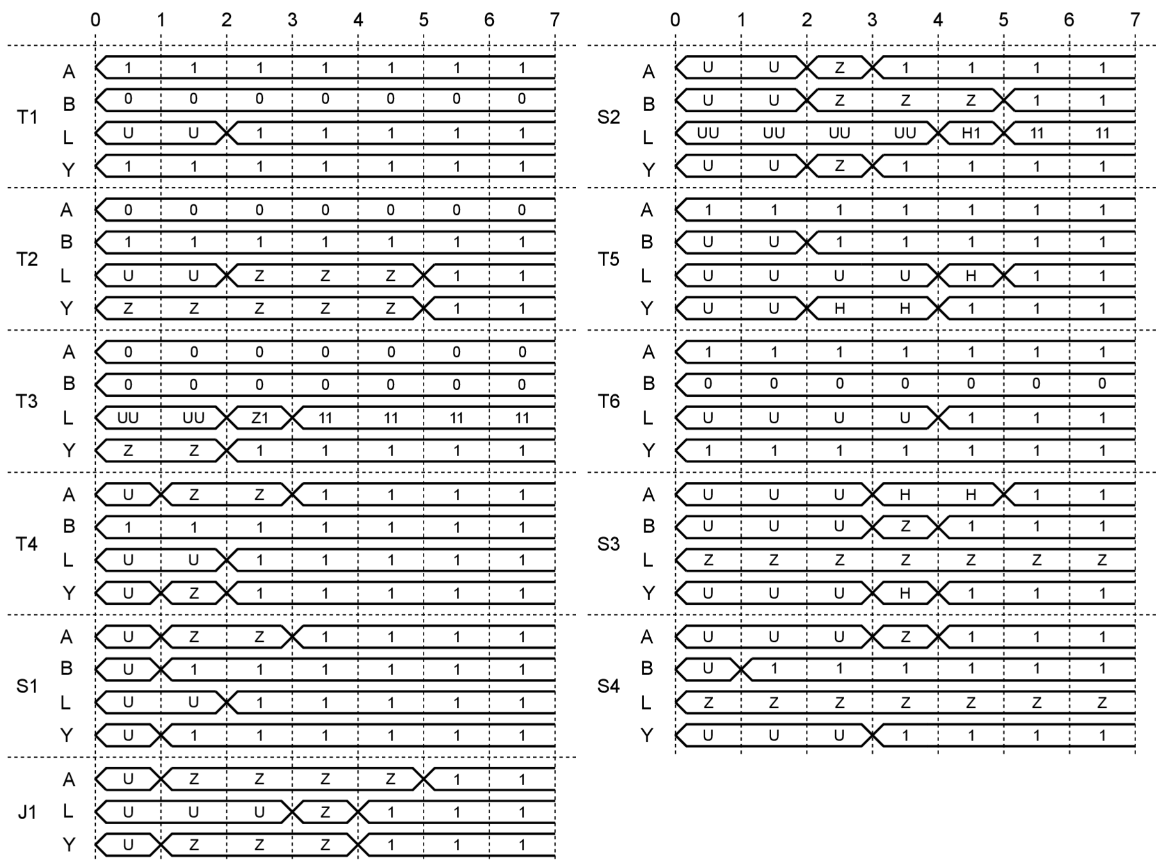
Proces synchronní simulace a komunikace mezi jednotlivými elementy bude znázorněn na příkladu XOR hradla uvedeném na obrázku 5.9



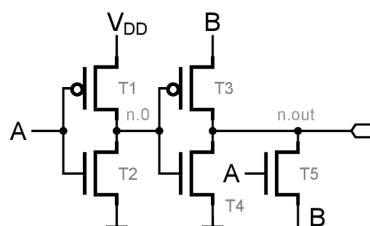
Obrázek 5.9: XOR v mřížce CGP

Průběh testování obvodu pro vstupy $A = 1$ a $B = 0$ je znázorněn na obrázku 5.10. Na obrázku se vyskytují hodnoty 0 pro signál GND, 1 pro signál V_{DD}, L pro oslabenou 0 ($|V_{tp}|$), H pro oslabenou 1 ($V_{DD} - V_{tn}$) a úroveň Z pro stav vysoké impedance. Vstupy tranzistorů nejsou označeny klasicky *source, gate, drain*, ale budeme vycházet z reprezentace v CGP (A,B,Y). Vstup L je daný zpětnou vazbou a v případě, že dochází k výběru z více prvků, jsou všechny hodnoty vypsány. K výběru z více hodnot (a tím pádem k redukci) dochází pouze u tranzistorů T_3, který je připojen k S_1 a k T_4 a dále u spojky S_2 připojené současně k S_3 a S_4. Výstup obvodu odpovídá hodnotě výstupu elementu S_2, což je v našem testovaném případě 1.

Na dalším příklade dvouvstupého hradla NAND2 znázorněném na obrázku 5.11a bude ukázán stav vodičů v ustáleném stavu. Pro zjednodušení nebudeme ukazovat princip řešení propojek a spojů, jak bylo demonstrováno u jedné kombinace hradla XOR, ale pouze fungování tranzistorů. Stav vodičů pro jednotlivé vstupy po ustálení simulace je ukázán na obrázku 5.11b.



Obrázek 5.10: Diskrétní simulace XOR obvodu na CGP úrovni pro $A = 1$ a $B = 0$



(a) Schéma

Vstupy		Vodič n.0			Vodič n.out			
A	B	T_1	T_2	$n.0$	T_3	T_4	T_5	$n.out$
0	0	1	Z	1	Z	0	Z	0
0	1	1	Z	1	Z	0	Z	0
1	0	Z	0	0	L	Z	0	0
1	1	Z	0	0	1	Z	H	1

(b) Ustálený stav tranzistorů

Obrázek 5.11: Dvouvstupé hradlo AND s povolením většího odběru ze vstupních vodičů

Kapitola 6

Platforma Zynq

V kapitole 3 jsme si představili technologie určené k akceleraci evolučních algoritmů a také byly ukázány jednotlivé aplikace. Vlastní akcelerační jednotka bude vytvořena na čipu Zynq, proto si ji v následující části podrobně představíme. Platforma Xilinx Zynq představuje moderní technologii integrace více typů obvodů do jednoho pouzdra označovanou jako SoC. Jedná se o spojení procesorového systému (PS) s dvoujádrovým procesorem ARM Cortex A9 a programovatelné logiky (PL) Artix-7 [57]. Logika Artix-7 je použita v čipech nižší řady, která je mimo jiné použita v čipu XC7020, na kterém bude práce vyvíjena. Na složitějších čipech je použita architektura logiky odvozená z řady Kintex-7.

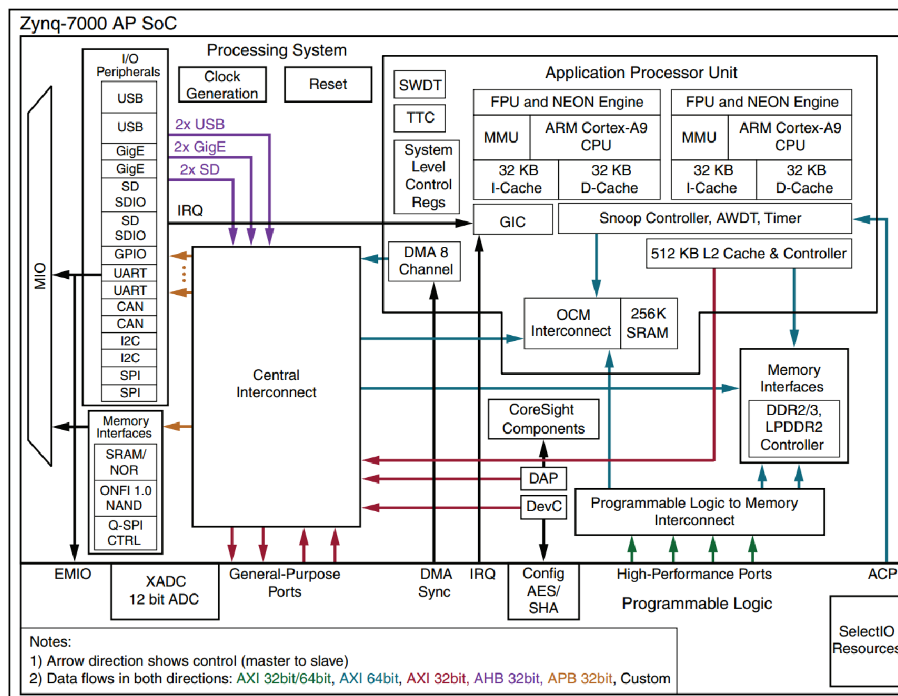
6.1 Procesorový systém

Strukturu systému můžeme vidět na obrázku 6.1. Jak již bylo řečeno, procesor se skládá z programovatelné logiky a procesorové jednotky. Na schématu je zejména znázorněna vnitřní struktura procesoru, který je složen z jádra procesoru *Application Processor Unit*, propojovacího mostu *Central Interconnect*, který provádí propojení a alokaci adresových prostorů jednotlivých periferií a vlastních periferií.

Procesorový systém kromě jádra ARM obsahuje i jednotku NEON, která umožňuje SIMD zpracování, a také jednotku FPU s dvojitou přesností. Každá výpočetní jednotka obsahuje vlastní intrukční i datovou cache L1 a 512 KB sdílenou paměť L2. Všechny další periferie jsou mapovány do společné paměti.

Systém obsahuje řadu vestavěných periferií. Jednou z nejdůležitějších je jednotka umožňující připojení SD karty, která je zásadní kvůli tomu, že celý systém může při startu načítat instrukce i konfiguraci programovatelné logiky z této karty. Tato funkce je využívána při použití operačního systému. Další z periferií je řadič USB, který může fungovat v režimu USB Host i USB Device. Velmi často je používaná funkce virtuálního COM portu, který přes emulovanou sběrnici RS232 může komunikovat s počítačem a vytvářet například interaktivní terminál. Procesor obsahuje také 2 RGMII[1] rozhraní, které po doplnění čipu pro fyzickou vrstvu umožňuje zpracovávat data ze sítě o rychlosti až 1 Gbps. Díky použití 2 rozhraní může systém pracovat například jako směrovač. Výhodou systému je i integrování rozšiřujícího se protokolu CAN. Tento třívrstvý protokol určený pro automobilový průmysl je v dnešní době hodně používaný. Na procesoru jsou implementovány dvě nejvyšší vrstvy – objektová a transportní. Nejnižší vrstva, fyzická, může být propojena pomocí vnějšího integrovaného obvodu nebo s využitím IP jádra realizující tuto funkci v logické části. Dále

[1]Reduced Gigabit Media Independend Iterface



Obrázek 6.1: Blokové schéma systému Zynq [57]

obsahuje standardní periferie pro synchronní komunikaci SPI a I^{2}C i pro asynchronní přes jednotky UART. Další jednotkou rozšiřující možnosti využití procesoru jsou ADC převodníky.

Operační paměť procesoru může být buď mapována do interních BRAM pamětí v logické části, nebo může být využito vnější paměti. Pro její připojení je možné využít standardů DDR2, DDR3 nebo LPDDR2, jejichž řadiče jsou přímo integrované v procesoru.

6.2 Programovatelná logika

Implementace vychází z architektury FPGA řady Artix-7 nebo Kintex-7 dle typu čipu. Je inicializována až po nakonfigurování procesorovou jednotkou pomocí bitstreamu, který bývá často uložen na SD kartě, na které je uložen i obraz operačního systému.

Pro komunikaci mezi jednotkami je využita sběrnice AXI4 [56]. Firma Xilinx se snaží v nových řadách nahradit více typů sběrnic určených pro komunikaci mezi IP jádry právě tímto protokolem. Obecná verze této sběrnice s velikostí 32b dokáže na tomto SoC pracovat na frekvenci 150 MHz a dosáhnout přenosové rychlosti pro zápis i čtení 600 MB/s oběma směry současně. Existuje i vysokorychlostí 64b verze sběrnice, která dosahuje dvojnásobné přenosové rychlosti, tedy 1200 MB/s oběma směry.

Sběrnice může pracovat v jednom ze tří režimů. Prvním režimem je režim označovaný jako *AXI4*. Tento režim podporuje registry a dávkové přenosy. Dávkovými přenosy rozumíme přenosy více registrů současně, kdy nečekáme na dokončení předcházejícího přenosu. Rozhraní je připojen adresový prostor, který odpovídá minimálně počtu registrů. Při manipulaci s daty je jako adresa registru dekódována spodní část adresy z paměti. Druhým režimem je režim *AXI4–Lite*. Tento režim, podobně jako předcházející, přenáší registry včetně adresy. Jediným rozdílem je to, že se čeká na doručení registru, díky čemu nemůže

dojít k časové kolizi mezi zápisem a čtením. Posledním režimem je *AXI4–Stream*, který přenáší velké bloky dat. S tímto protokolem jsme schopni dosáhnout nejvyšší přenosové rychlosti.

Často se používá kombinace více AXI portů, kdy jeden pracuje například s protokolem *AXI4–Lite* a slouží k přenosu řídících bitů. Vedle toho má IP ještě port s protokolem *AXI4–Stream*, přes který potom přenáší bloky zpracovaných dat. Pokud chceme použít více zařízení, použijeme IP jádro *AXI central interconnect*, které tvoří přepínač s více porty pro tuto sběrnici a také všechny režimy komunikace převádí na režim *AXI4*.

PL jednotka má hodinový signál oddělený od časování procesoru. Při vývoji je tedy nutné nastavit správně výstupní frekvence, přičemž jednotlivé IP jádra mohou pracovat na různých frekvencích.

6.3 Postup tvorby HW

Firma Xilinx dlouhou dobu používala návrhové prostředí ISE. S nástupem nové řady Virtex 7 UltraScale FPGA však ISE přestávalo stačit a bylo nutné navrhnout nový systém. Nástupcem této řady se stalo prostředí Vivado, které podporuje většinu trendů moderního vývoje obvodů, včetně syntézy na vyšší úrovni z jazyka C[2]. SoC Zynq však dlouho podporován nebyl, proto bylo nutné využívat starší prostředí ISE. Teprve na konci roku 2013 přišla podpora v nástroji Vivado.

6.3.1 Návrh v prostředí ISE

Návrh v prostředí ISE vyžaduje využívat více nástrojů [58]. Základní projekt je vytvořen v programu PlanAhead. V tomto projektu je nutné založit definici IP jader pomocí nástroje Xilinx Platform Studio. Nástroj využijeme k vložení IP jádra *processing_system_7*, který představuje procesorovou jednotku. V této aplikaci také vytvoříme mapování periferií do paměti procesoru a nastavíme frekvenci časování.

Vlastní IP jádro vytvoříme stejně jako všechna ostatní jádra pomocí generátoru v XPS. Zdrojový kód jádra ve VHDL můžeme otevřít jako projekt ISE a simulovat jej samostatně.

Pro sestavení bitstreamu je nutné syntetizovat samotné IP jádro, poté vytvořit výstupní soubory v XPS. Po provedení těchto operací lze znovu provést syntézu a mapování v nástroji PlanAhead, čímž získáme výsledný konfigurační kód obvodu.

6.3.2 Návrh v prostředí Vivado

Prostředí Vivado integruje všechny nástroje v jeden celek a díky tomu je možné v jednom projektu provést všechny potřebné kroky [61]. Pokud využijeme tento nástroj, není nutné vytvářet samostatný IP modul, ale kódy v jazyce popisující HW (VHDL, Verilog nebo C/C++ pro HLS) je možné mít vedle souborů popisujících architekturu. Tyto soubory popisující architekturu nahrazují původní nastavení z programu XPS. Návrh IP jader musí být složen z procesorového bloku a propojovacího bloku sběrnice AXI4. Na výstupním portu je nutné nastavit správný typ sběrnice AXI4. Dále je nutné vyvést časování *FCLKn* a tomuto signálu nastavit správnou frekvenci. Oba tyto výstupní porty označíme jako externí.

Po ukončení návrhu propojení bloků vygenerujeme VHDL wrapper. Ten však bude nutné upravit, protože námi vytvořené externí porty nebudou připojeny na výstupy čipu,

[2]HLS – High Level Synthesis

ale budou interně propojeny s dalším blokem našeho modulu. Přemapováním těchto portů docílíme připojení vnitřního bloku.

6.4 Postup tvorby SW

Pro vývoj softwaru pro procesorovou jednotku je určen nástroj XSDK neboli Xilinx Software Development Kit. Jedná se o integrované vývojové prostředí založené na open-source projektu Eclipse. Založení projektu probíhá z vývojového nástroje pro logickou část, z Vivada nebo z XPS podle použité návrhové platformy, pomocí nástroje *Export HW*. Při vyváření softwaru pro SoC máme možnost využít operačního systému.

Pokud využíváme operační systém (nejčastěji Linux, můžeme však využít i Android nebo Microsoft Windows Embedded), slouží nástroj XSDK pouze k vývoji uživatelských aplikací a k sestavení obrazu systému s požadovanou konfigurací HW. Pro následující přístup k jednotkám v logické části používáme klasických paměťových operací na adresách určených při návrhu bloků. U operačních systémů však dochází k problémům s oprávněním aplikací k přístupu do této chráněné paměti. Proto musíme napřed přemapovat tento prostor do uživatelské paměti [58].

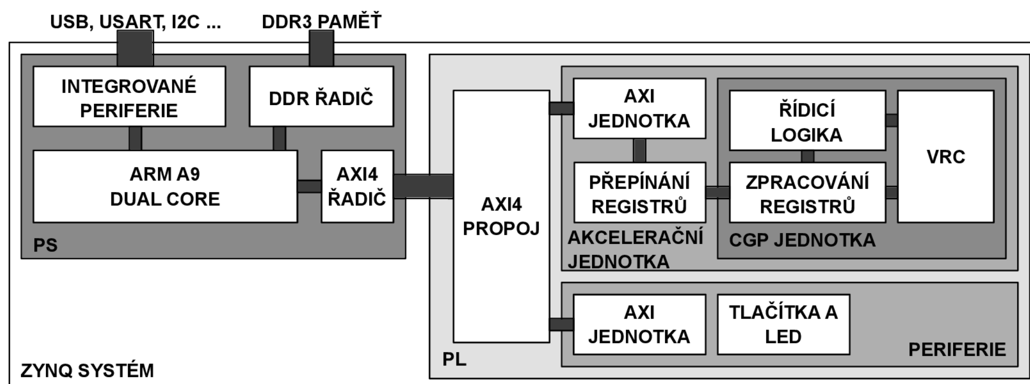
Jednodušší přístup k HW jednotkám je bez použití klasických operačních systémů nebo s využitím real-time operačních systémů (FreeRTOS, QNX, μC/OS a podobně), které jsou distribuovány jako knihovny pro jazyk C/C++. Vlastní aplikace je kompletně navržena v nástroji XSDK a je složena minimálně ze 3 projektů. Prvním projektem je *hw_platform*, která je vygenerována při exportu HW. Obsahuje kompletní nastavení konfigurovatelných bloků, jako jsou například taktovací frekvence nebo mapování jednotlivých součástí v paměti. Obsahuje funkce pro konfiguraci zařízení po startu pomocí inicializačních funkcí ze strany procesoru a také inicializaci pro případ ladění přes skripty TCL. Z definice platformy je vygenerován balíček ovladačů jednotlivých periferií, tzv. *board supporting package*. Kromě ovladačů povolených periferií obsahuje také definici procesoru a odkazuje se na HW platformu. Na této úrovni volíme, se kterým jádrem procesorové jednotky budeme pracovat. Můžeme tedy vytvořit 2 *BSP* projekty. Tento balík obsahuje i definici cílového operačního systému, můžeme tedy zvolit například i FreeRTOS, který podporuje vícejádrové procesory. Na nejvyšší úrovni se nachází aplikační projekt. Tento projekt představuje kód vlastní aplikace, která je přímo svázána s BSP balíčkem a kódem použitého operačního systému.

Nástroj umožňuje i programování a ladění aplikace pomocí rozhraní JTAG. Prvním krokem je naprogramování FPGA konfiguračním souborem vygenerovaný syntézou a mapováním z návrhového nástroje. Vlastní ladění je řešeno pomocí GDB serveru, díky tomu můžeme při ladění kódu využívat všech možností, jako jsou například body přerušení nebo pohledy na registry. Aplikace implicitně mapuje standardní výstup na virtuální COM port, díky tomu můžeme v kódu využívat klasických funkcí jako je `printf` nebo `scanf` a vytvořit i bez podpory OS uživatelsky příjemnou aplikaci s konzolovým CLI rozraním.

Kapitola 7

Architektura akcelerátoru evolučního návrhu tranzistorových obvodů

Blokové schéma navrženého akcelerátoru je znázorněno na obrázku 7.1. Akcelerátor se skládá ze dvou částí – evolučního algoritmu běžícího na procesorovém systému ARM (PS) a akcelerační jednotky implementované v programovatelné logice (PL). Mimo toho PL obsahuje i řízení externích periferií, jako jsou tlačítka a LED diody, a také propojovací most sběrnice AXI realizovaný pomocí IP jádra *AXI Central Interconnect*. Evoluční algoritmus bude zpracováván výpočetní jednotkou a každé kandidátní řešení bude předáno HW akcelerační jednotce pro ohodnocení.



Obrázek 7.1: Architektura Zynq

Pro účely komunikace PL a PS je k dispozici rozhraní, které výrobce označuje jako AXI4. Toto rozhraní bude využito i v našem případě, kdy přes něj budeme přenášet chromozom, který kóduje kandidátní řešení obvodu, směrem z procesoru do akcelerátoru a dále hodnoty pro výpočet fitness opačným směrem. Dále procesor přes sdílený paměťový prostor komunikuje s integrovanými periferiemi a DDR3 pamětí o kapacitě 1 GB, která slouží jako operační paměť.

Akcelerační část od procesoru konfigurační řetězec (chromozom). Po potvrzení příjmu spustí simulaci řešení. Celá komunikace prochází přes rozhraní AXI4 – Lite a využívá celkem pět 32b registrů, 3 vstupní a 2 výstupní. Rozkódování registrů podle délky řeší jednotka *Přepínání registrů*. Ta předá konkrétní řetězce *Zpracování registrů*, protože je nutné je dále

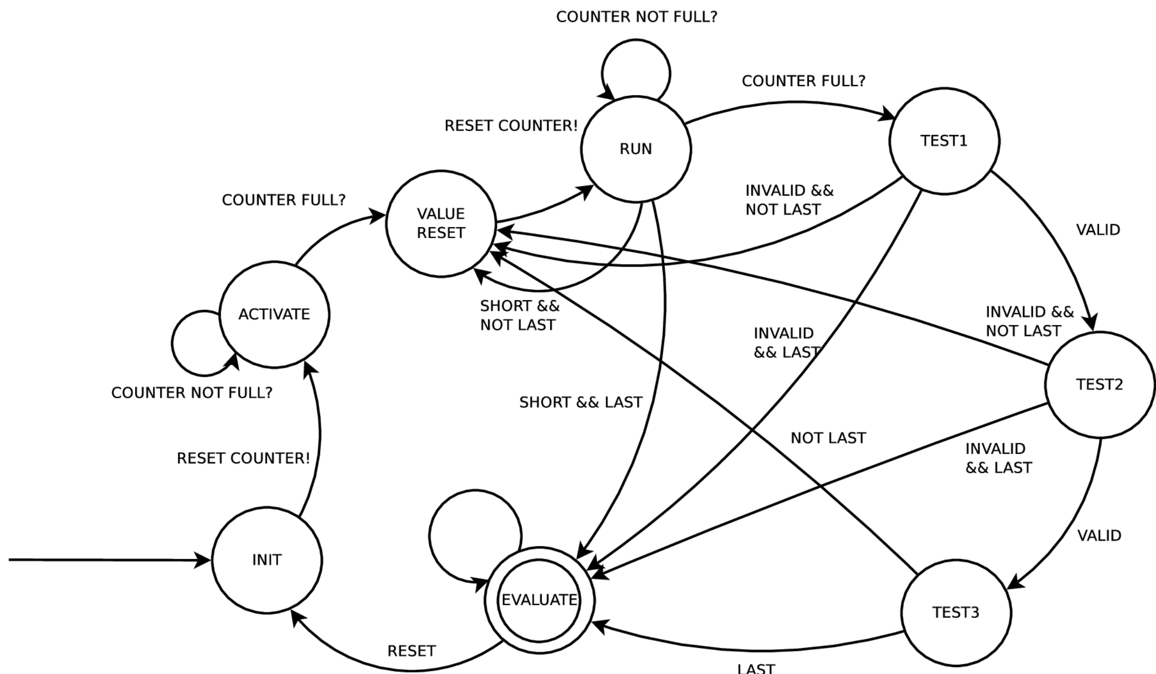
zpracovat pro vlastní simulaci. Chromozom se totiž přenáší po částech, kdy spodních 27 bitů je vyhrazeno pro data a horních 5 bitů určuje adresu. Bylo by možné posílat pouze data s tím, že by se adresa postupně inkrementovala po blocích, což by mohlo mírně snížit počet přenášených registrů. Přístup s vestavěnou adresou má však tu výhodu, je možné posílat data pouze diferenciálně a bloky, které se nezměnily, neposílat. Navíc není nutné předem znát výslednou délku chromozomu (a tím i dynamicky měnit počet registrů), což by bylo potřeba při využití adresování AXI sběrnice. Jednotka mimo jiné řeší i přepínání signálů pro spouštění a ukončení běhu simulace.

Evaluace kandidátního řešení se skládá z detekce aktivních prvků, generování a vložení jednotlivých vstupních vektorů a vyhodnocení výsledků. Zajišťuje jej *Řídicí logika*, která je podrobně popsána v následující kapitole.

Virtuální rekonfigurovatelný obvod složený z matice jednotlivých elementů je implementován v jednotce označené *VRC*. Detailní struktura jednotlivých elementů je popsána v kapitole 7.2.

7.1 Řízení evaluace

Evaluace kandidátního řešení je řízena stavovým automatem uvedeným na obrázku 7.2 a skládá se z detekce aktivních prvků, běhu simulace a testování výstupu.



Obrázek 7.2: Řízení CGP v CGP jednotkce

Simulace probíhá podle následujícího algoritmu, ve kterém figurují proměnné COL, což je generická proměnná určující počet sloupců VRC obvodu. Algoritmus využívá i generické proměnné $INPUTS$ označující počet variabilních vstupů VRC.

1. Spuštění detekce aktivních prvků

2. Čekání $\frac{1}{2} \cdot COL \cdot ROW$ hodinových taktů (viz kapitola 7.2.2)

3. Vynulování vnitřních proměnných a výstupů bloků a nastavení primární vstupů podle čítače běhů

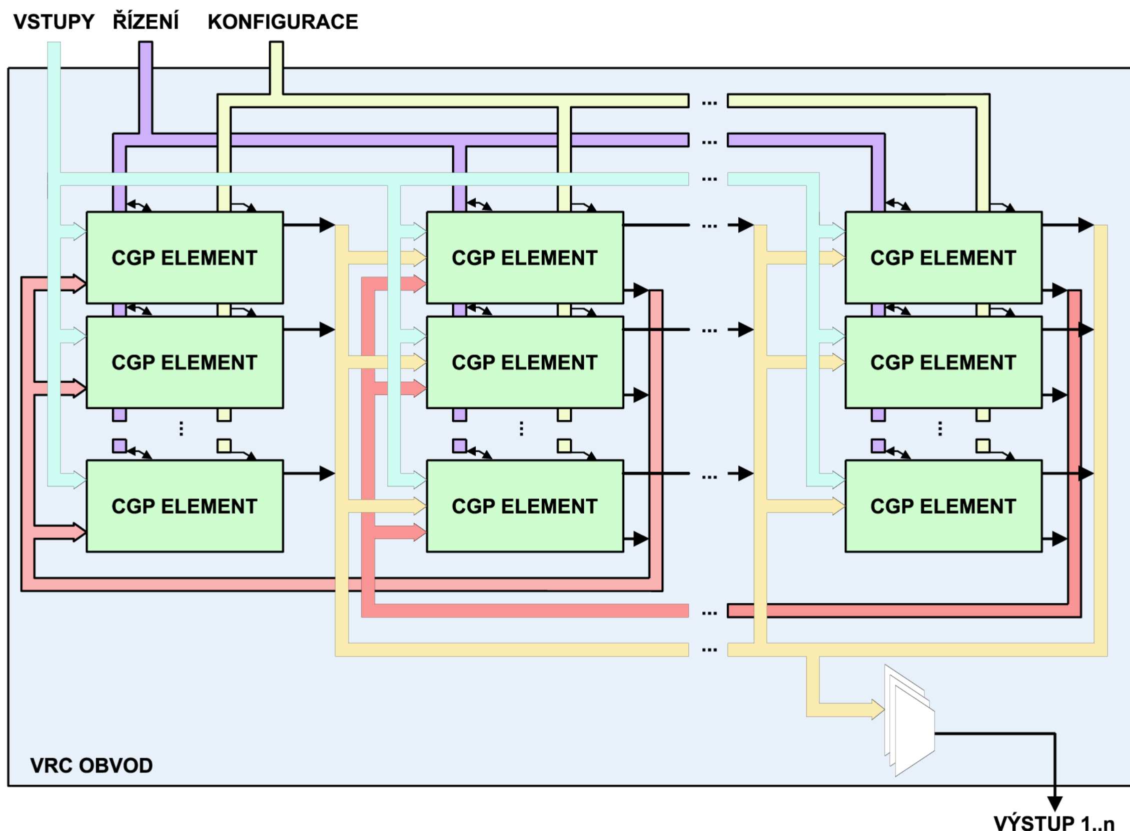
4. Spuštění procesu simulace po dobu n hodinových taktů, kde n je součet počtu aktivních sloupců a počtu spojů (v případě chyby skok na krok 3 nebo na 6 podle toho, jestli je to poslední běh)

5. Ve třech po sobě jdoucích taktech otestování výstupní hodnoty s referenční hodnotou a započítání správných / chybných hodnot. Pokud počet běhů byl menší než 2^{INPUTS} pokračuj krokem 3, jinak pokračuj dále

6. Naplnění výstupních registrů a zaslání informace do procesoru o dokončení operace

7.2 Virtuální rekonfigurovatelný obvod

Pro realizaci rekonfigurovatelné logiky, která na základě chromozomu dynamicky propojí bloky mřížky CGP, se používá virtuální rekonfigurovatelný obvod. Základní bloky tvoří elementy realizující funkce definované v navržené metodě simulace obvodu. Struktura znázorněná na obrázku 7.3 odpovídá matici CGP elementů, které jsou navzájem pospojované. Jednotka má řadu generických parametrů: *ROWS* a *COLS* určující velikost mřížky, *INPUTS* určující počet vstupů a *OUTPUTS* vyjadřující počet výstupů.



Obrázek 7.3: Architektura VRC

Na rozdíl od VRC publikovaných v pracích [44, 45, 5], je nutné kromě dopředných vazeb zavedení i zpětných vazbu, protože u tranzistorů na rozdíl od hradel není možné předem určit, která elektroda je vstupní a která výstupní. To přináší úskalí v podobě rostoucího počtu spojů mezi elementy. Protože je CGP definováno tak, že více vstupů může být spojeno se stejným výstupem, tak pro zpětnou vazbu může být spojeno více signálů do jednoho. Tato redukce je výpočetně náročná, a proto je zpětná vazba pouze mezi sousedícími dvojicemi sloupců, což určuje i LBACK parametr, který je z tohoto důvodu omezen na 1. Bez tohoto omezení by obvod VRC byl velmi náročný na syntézu a už i pro malé velikosti matic by mohlo dojít k vyčerpání všech zdrojů čipu.

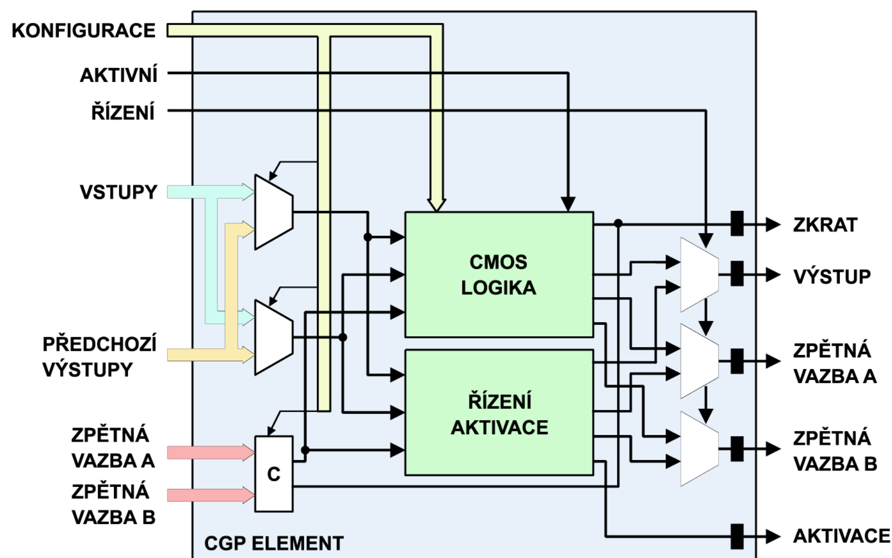
Celkové schéma návrhu VRC jednotky je znázorněno na obrázku 7.3. Každý vnitřní element dostává z konfiguračního řetězce své bity, podle kterých mění svou funkci. Toto nastavení je přenášeno přímo z procesoru a až v jednotce VRC dochází k rozdělení konfigurace pro jednotlivé elementy.

Dalším vstupem jednotlivých elementů jsou řídicí signály z řídicí jednotky. V této skupině signálů je šířen resetovací vodič, dále obsahuje přepínání režimu detekce aktivních prvků a vlastní simulace. Obsahuje také signály sloužící k detekci. Komunikace v této skupině je obousměrná, jednotka může řídicí jednotce oznámit zkrat nebo nevalidní konfiguraci.

Elementy také od nadřazené jednotky dostávají i stimulující vstupní vektory, pro které pak probíhá simulace. První prvek vektoru je vždy nízká logická úroveň, druhý prvek je vysoká logická úroveň (GND a V_{cc}). Další prvky už jsou složeny z primárních vstupů. Vstupní prvky vždy obsahují silné hodnoty, protože řídicí jednotka obsahuje čítač pro určení jednotlivých vektorů. Z tohoto čítače jsou odvozeny i vstupní vektory, kdy 0 odpovídá nízké hodnotě a 1 vysoké.

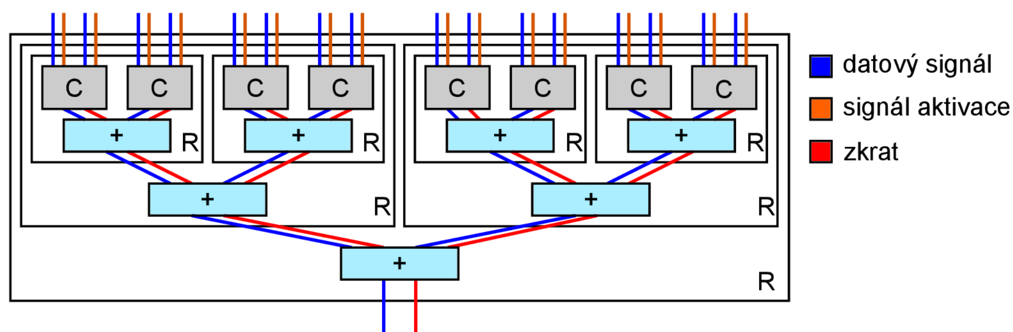
7.2.1 Struktura elementu

Struktura každého elementu je zobrazena na obrázku 7.4. Element zpracovává vstupy, z nichž vybírá konkrétní hodnoty podle konfigurace. Dále obsahuje dva obvody pro jednotlivé režimy funkce elementu.



Obrázek 7.4: Architektura CGP elementu

Element má na vstupu celý stimulující vektor složený ze silné nízké hodnoty reprezentující signál GND a silné vysoké hodnoty signálu V_{CC} a primárních vstupů. Na vstupu má také výstupy všech elementů z předchozího sloupce. U elementů v prvním sloupci je na vstupu pouze vstupní vektor. Ze vstupů pomocí dvou multiplexorů řízených konfiguračním registrem vybírá dvě vstupní hodnoty. Řídicí signál multiplexoru má šířku n bitů, která odpovídá hodnotě $\lceil log_2(n) \rceil$ odvozené dle generického parametru $INPUTS$. Je vidět, že obsazených vstupů multiplexoru může být méně a některé vstupy multiplexoru pak nemusí být definované. V konfiguračním řetězci se však může objevit jiná hodnota, a proto je řídicí registr multiplexoru omezen funkcí minima tak, aby nedošlo k nevalidní konfiguraci. Tento problém by bylo možné řešit i pomocí obsazení nevyužitých vstupů poslední hodnotou. Bylo však ověřeno, že v rychlosti syntetizovaného řešení nehraje tato skutečnost žádný vliv, protože se tyto omezující prvky neobjevují v kritické cestě časování, a proto bylo zvoleno omezení minimem.



Obrázek 7.5: Architektura entity provádějící stromovou redukci signálu pro 16 vstupů

Ve zpětné vazbě jsou do elementu přenášeny hodnoty z následujícího sloupce. Jelikož však výstup elementu může být připojen na vstupy elementů následujícího sloupce, je nutné spojit více úrovní zpětné vazby do jednoho. Toto spojení však může vyvolat zkrat, který je nutné zachytit. Signál zkratu je následně zaveden zpět na řízení VRC jednotky. Pro účel redukce více signálů do jednoho byla vytvořena komponenta, která na základě úrovně a signálu určující aktivitu odpovídajícího elementu (obrázek 7.5) vypočte redukovanou hodnotu a signalizuje případný zkrat. Signály určující aktivitu jsou nastaveny podle konfiguračního řetězce a jsou aktivní pouze pro elementy, které mají svůj vstup propojený s výstupem aktivního elementu. Výstupy pro zpětnou vazbu jsou definovány pro každý ze vstupů (zpětná vazba A pro první vstup a zpětná vazba B pro druhý).

Proces spojování zpětnovazebních signálů je poměrně náročný na zdroje a vytváří dlouhou cestu, která často bývá kritická. Vzhledem k optimalizaci pro rychlost je z důvodu malého zpoždění vytvářeno stromovou strukturou spojovacích elementů, které je docíleno za pomoci rekurze na úrovni VHDL. Díky tomu se podařilo redukovat časovou složitost na straně zpětné vazby závislou na počtu řádků z $O(n)$ na $O(log\ n)$. Na obrázku je ukázaná struktura rekurzivní redukce signálu pro 16 vstupů, která se využívá pro 8 řádkový VRC. Obvod označený jako C kombinuje dva CMOS signály s příznakem aktivace, kdy může spojit dva stejné signály. Pokud je jeden signál degradovaný a druhý je silný, na výstupu se objeví signál silný. Dále může zkombinovat libovolný signál se stavem vysoké impedance Z, který druhý signál nemění, nebo s nedefinovaným signálem U, který se agresivně šíří dál. V případě, že vstupní signál není aktivní, bere se vstup jako stav vysoké impedance. V ostatních případech je vyvolán zkrat a na výstupu se objeví nedefinovaný stav. Podobně funguje i prvek označený jako $+$, který také kombinuje dva signály. Nevyužívá však signálu

aktivace, zato ale propaguje předchozí detekci zkratu dále. Z tohoto prvku je dále vytvářena stomová struktura.

Element pracuje ve dvou módech. Na začátku simulace kandidátního řešení je provedena detekce aktivních prvků. Poté je simulována funkce obvodu s tranzistory pro každou možnou kombinaci vstupního vektoru, tj. $2^n\times$.

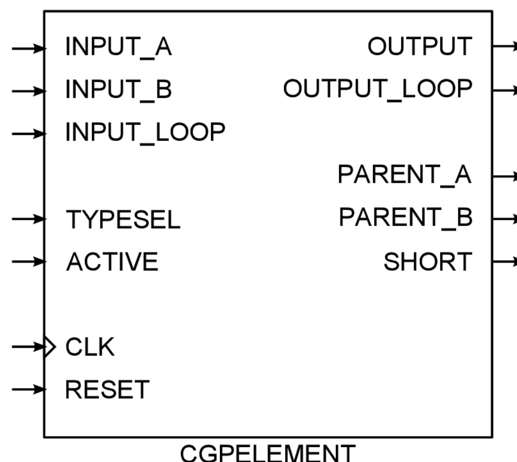
7.2.2 Mód detekce aktivních prvků

Tento mód označuje proces, ve kterém je nutné projít všechny elementy, které jakkoliv ovlivňují výstup, a zároveň označit elementy, které na výstup nemají vliv kvůli nežádoucím zkratům, jak bylo ukázáno v kapitole 5.4. Vzhledem k výskytu zpětných vazeb není možné určit aktivní prvky jedním průchodem. Ty, které jsou aktivní, přiřadí na výstup zpětné vazby logickou 1, pokud se jedná o element realizující propojku (jumper), tak se hodnota objeví i na výstupu, na výstupu zpětné vazby B však ne. Podobně i u elementů realizující funkci spoje (switch) se silná 1 objeví jak u zpětné vazby, tak i u výstupu. V dalším taktu se nastaví jako aktivní prvky, které mají na některém ze vstupů jedničku (kromě propojky, která ignoruje vstup B).

Musíme počítat s tím, že každá propojka může prodloužit proces detekce aktivních prvků o jeden takt, protože může aktivovat elementy v již detekovaném sloupci. Experimentálně bylo určeno, že maximálně polovina elementů může být propojkami. Celá aktivace tedy trvá $\frac{1}{2}R \cdot C$ taktů, kde R označuje počet řádků VRC a C počet jeho sloupců. Pokud by náhodou bylo v obvodu více propojek, neznamenalo by to nutně to, že by se všechny nezaktivovaly. Pokud však by se nepodařilo aktivovat všechny, simulace by fungovala korektně, pouze s menším počtem elementů. Během testování však tento stav nikdy nenastal.

7.2.3 Mód simulace činnosti tranzistoru

Tento režim elementu simuluje vlastní fungování tranzistorů. Tento mód následuje vždy až po detekci aktivních prvků, jinak není zaručeno bezchybné chování. Na začátku testování každého vstupního vektoru je provedeno vynulování hodnot (odlišujeme dva druhy resetu – tento a celkový, který je proveden před evaluací dalšího kandidátního řešení). Jak bylo popsáno, nadřazená entita vybírá hodnotu jednotlivých vstupů.



Obrázek 7.6: Rozhraní entity realizující logiku CGP elementu

Rozhraní entity realizující funkci CGP elementu je znázorněna na obrázku 7.6. Každá z entit má kromě konfiguračních vstupů tři datové vstupy, kde vstupy INPUT_A a INPUT_B vzniknou výběrem ze vstupních signálů a předchozího sloupce, a vstup INPUT_LOOP je výsledkem redukce zpětných vazeb. Dále využívá konfiguračních vstupů určující aktivitu prvku ACTIVE a také funkci elementu definovanou vstupem TYPESEL. Kromě základního výstupu využitého při určení výstupu celého VRC OUTPUT obsahuje i signál pro výstup vazbyna další prvky OUTPUT_LOOP, který je důležitý pro prvek spojky a propojky. U tranzistorů tento signál odpovídá klasickému výstupu. Propojení mezi entitami jsou obousměrné, proto obsahuje zpětnou vazbu pro vstup A i pro vstup B označenou jako PARENT_A a PARENT_B. Posledním výstupem elementu je signál pro detekci zkratu na prvku SHORT.

Entita dle stavu signálu TYPESEL počítá jednu ze čtyř funkcí uvedených v kapitole 5.4. Jednou z funkcí, kterou lze realizovat, je **propojka**, kdy výstupní hodnota OUTPUT_LOOP odpovídá jedinému vstupu INPUT_A a naopak zpětná vazba PARENT_A odpovídá vstupu z následujícího sloupce INPUT_LOOP. Výstup obvodu OUTPUT odpovídá vyšší hodnotě z kombinace INPUT_A a INPUT_LOOP. K tomu využívá pomocnou funkci, která ze dvou signálů vybere silnější. Vychází z toho, že při kombinaci oslabeného signálu a silného signálu je výsledkem silný signál. Může však i identifikovat zkrat při kombinaci dvou nekompatibilních signálů.

Další funkcí, kterou entita realizuje, je **spoj**, která slouží k simulaci elektrického spojení vstupů A a B a výstupu. Kromě toho musí spojovat i vstup ze zpětné vazby. Podobně jako při realizaci funkce propojky, využívá funkci kombinující dva signály s výsledkem silnějšího signálu. Každý výstup je dán zpracováním ostatních dvou vstupů. To znamená, že PARENT_A odpovídá kombinaci INPUT_B a INPUT_LOOP, PARENT_B se vypočte z INPUT_A a INPUT_LOOP a poslední vazba OUTPUT_LOOP odpovídá kombinaci signálů INPUT_A a INPUT_B. V tomto případě nemusí výstupní hodnota OUTPUT odpovídat výstupní hodnotě pro vazbu na další prvky OUTPUT_LOOP, protože je dána kombinací všech třech vstupů. Bez rozdělení těchto dvou výstupů by totiž mohlo docházet k jevu, kdy na vstupu INPUT_LOOP prvku bude silná hodnota, všechny ostatní vstupy budou mít hodnotu stejnou, ale degradovanou. Tato silná hodnota se bude propagovat dále. Pokud by se však vstup přepnul ze silné hodnoty na degradovanou, předchozí elementy udrží hodnotu silnou a spojka se pak zachová jako paměťový prvek.

Třetí funkcí realizovanou touto entitou je simulace chování tranzistoru typu **NMOS**. Vstup INPUT_A odpovídá elektrodě *source*, INPUT_B odpovídá gate a INPUT_LOOP odpovídá *drain*. Abychom mohli korektně simulovat chování tranzistoru, využíváme další tři vnitřní paměťové prvky. Jedná se o indikaci stavu reverzace tranzistoru, čímž rozumíme změnu směru proudu elektrodou *drain* a vstupy *gate* a *drain* použité pro otočení funkce. Ve výchozím stavu tranzistor pracuje jako spínač, který propojuje v závislosti na vstupu *gate* signál ze *source* do *drain*. Po vynulování prováděném na začátku simulace každého vstupního vektoru jsou všechny výstupy v neznámém stavu U. Jakmile jsou vstupy A a B definované, vypočte se podle tabulky 5.2 v kapitole 5.4 výstup.

Po výpočtu nové hodnoty dojde ke kombinaci nové hodnoty se zpětnou vazbou připojenou na vstup INPUT_LOOP. Pokud je kombinace validní, dostává se na výstup vyšší hodnota z obou hodnot. Pokud je tedy výstupem například oslabená hodnota a některý z následujících připojených elementů má na své zpětné vazbě pro daný prvek (PARENT_A nebo PARENT_B) hodnotu vyšší, tak nedojde ke změně, pouze tato vyšší hodnota se propaguje i do dalších připojených elementů. Každý tranzistor tedy může fungovat i jako propojka.

Pokud však vstup *source* je ve stavu vysoké impedance a na vstupu *drain* se objeví definovaná hodnota, tranzistor začne pracovat v opačném režimu. Tento jev plyne ze struktury

tranzistoru, neboť podle definice tranzistoru v kapitole 4 zjistíme, že ve fyzikálním popisu není významný rozdíl. Při přechodu tranzistoru do tohoto módu se výstup počítá úplně stejně jako v předešlém případě, pouze dojde k záměně vstupů *source* a *drain*. Hodnota *gate* i *drain* se však uloží do interní paměti. Může dojít ke stavu, kdy zpětná vazba ovlivní vstupní hodnotu a pak se musí tranzistor přepočítat.

Poslední funkcí entity je simulace chování tranzistoru typu **PMOS**. V tomto módu pracuje entita obdobně, jako bylo popsáno v předchozím případě u tranzistoru typu NMOS. Využívá se toho, že chování je symetrické a postačí tedy pouze zaměnit logické 1 za 0 a naopak. V kódu aplikace je však každá funkce realizována samostatně, protože záměna vzhledem k vícehodnotové logice je náročnější na čas zpracování a touto operací bychom sice ušetřili plochu čipu, mohli bychom však zpomalit taktovací frekvenci obvodu.

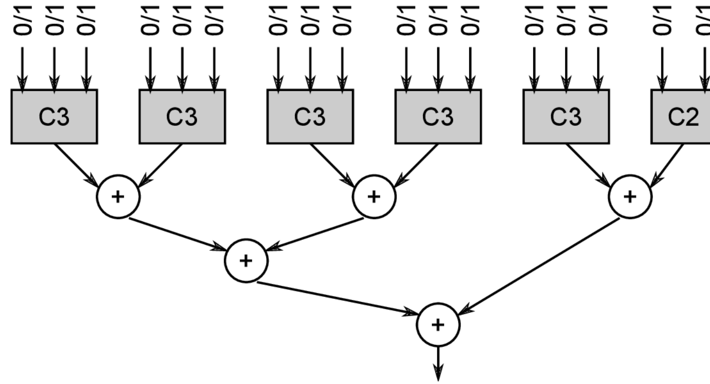
7.3 Výpočet fitness hodnoty a další výpočty ve VRC

Výpočet fitness hodnoty se provádí až v procesorové jednotce. Tento způsob vyhodnocení je zvolen z důvodu snadné modifikace koeficientů fitness pro jednotlivé typy hodnot (silnou a slabou hodnotu) a také z důvodu různých nastavení požadavků evolučního algoritmu. Aby bylo možné fitness hodnotu spočítat, je nutné znát vlastnosti konkrétního kandidátního řešení. Tyto hodnoty jsou vypočteny v jednotce VRC a v jednotce řízení běhu simulace. Základem jsou hodnoty jednotlivých výstupů z běhu simulace. Abychom snížili režii přenosů, jsou do procesoru přenášeny pouze počty výstupů, které byly validní, dále těch, které byly validní, ale degradované. Informování o zkratech a nestabilním výstupu je řešeno stejným způsobem. Tyto hodnoty jsou vypočteny v jednotce řízení.

Všechny další hodnoty jsou vypočteny přímo v jednotce VRC. Důležité jsou však i další parametry odvozené z konfigurace bez nutnosti spouštění samotného procesu simulace. Prvním z těchto parametrů je validita zapojení, protože simulace nemůže fungovat správně, pokud nejsou dodržené podmínky připojování vstupních hodnot. Příkladem může být propojka, ke které nesmíme připojit vstupní hodnotu, ale pouze výstup jiného elementu. Validita zapojení a to, jestli byl připojen vstupní signál na source tranzistorů jsou detekovány jednobitovými signály.

Další důležitou hodnotou pro výpočet fitness je počet aktivních elementů. Pro výpočet fitness hodnoty nás zajímá počet elementů řešící dva typy funkcí – tranzistory a spoje. Propojky zohledněny nejsou, protože se jedná pouze o vodiče, které nás při výrobě nic nestojí, jelikož neřešíme návrh na úrovni geometrického uspořádání. Pro každý element existují dva výstupní signály, a to jestli se jedná o aktivní tranzistor, nebo jestli se jedná o aktivní spoj. Tyto hodnoty jsou složeny do řetězce a pomocí vlastní entity znázorněné na obrázku 7.7, která pomocí stromové struktury spočítá počet jedniček ve vstupním vektoru tím, že redukuje n bitový řetězec na $\lceil log_2(n) \rceil$ bitovou binární hodnotu. Je složená ze stromové sčítačky a z kombinačních prvků určujících počet jedniček v 3 bitovém řetězci (C3) a počet jedniček v 2 bitovém řetězci (C2).

VRC jednotka dále počítá hodnoty, které nejsou důležité pro fitness, ale pro určení délky běhu simulace. První z těchto hodnot je počet sloupců, ve kterých je aktivní aspoň jeden element. Vzhledem k tomu, že každý element má na výstupu registr, dostaneme hodnotu ze vstupu na výstup právě za počet taktů daným tímto registrem. Výpočet je prováděn pomocí n-vstupého OR hradla, kde n je počet řádků, čímž pro každý sloupec získáme hodnotu, zda je v něm aspoň jeden aktivní sloupec. Pomocí výše popsané entity redukujeme počet jedniček na binární číslo.



Obrázek 7.7: Architektura entity stromové sčítačky

Druhou hodnotou z VRC, která určuje délku simulace je počet spojek, kterou určujeme i pro výpočet fitness. Simulace jednoho vstupního vektoru trvá přesně $C + SW$ taktů, kde C je počet sloupců a SW je počet taktů. V nejhorším případě může tedy trvat $C + R \cdot (C - 1)$ taktů, ovšem toto je horní odhad a v tomto případě by všechny elementy musely být nastaveny na spoj.

7.4 Implementace v procesorové jednotce s využitím HW akcelerace

Procesorový systém je nakonfigurován pomocí kódu psaného v jazyce C a využívá HW akcelerační jednotku. Využívá pouze jednoho jádra procesorové části a z důvodu dosažení maximální rychlosti nepoužívá žádný operační systém. Akcelerační jednotka je připojena přes 6 registrů, z nichž 3 jsou vstupní a 3 výstupní — poslední slouží pouze pro ladící účely. Mapování registrů a délka jednotlivých částí (pro konstanty I – počet vstupů včetně 0 a 1, O je počet výstupů, C je počet sloupců VRC a R je počet řádků VRC mřížky) je uveden v tabulce 7.1:

Z použitého mapování plyne omezení maximálního počtu primárních vstupů. Vzhledem k tomu, že registry mají šířku 32 bitů, není možné, aby bylo použito více jak 7 vstupů (včetně 2 pro napájecí vodiče, tedy 5 funkčních), neboť by byla šířka registru 2 překročena.

Aplikace pracuje následovně: po spuštění dojde k inicializaci generátoru náhodných čísel. Inicializace je prováděna pomocí doby běhu procesoru, která, jak bylo experimentálně ověřeno, je pokaždé jiná. Pro zvýšení míry entropie náhodných dat by bylo možné využít některý z vestavěných AD převodníků. Dále se vygeneruje počáteční populace chromozomů k testování. Tyto chromozomy jsou vygenerovány na úrovni jednotlivých spojů a funkcí a následně jsou převedeny do binární formy kódování chromozomu. Všechna řešení jsou přenesena do HW jednotky a následně je podle počtu výstupů zejména v registru 0 vypočtena fitness podle koeficientů v konfiguračním souboru v programu. Pokud počet validních výstupů (případně v součtu s degradovanými, pokud to uživatel povolí) odpovídá maximálnímu počtu $2^{I-2+log_2(O)}$, tak je nalezené řešení označeno za úplné a je připočítán počet nevyužitých elementů podle registru 4. Následně se vybere nejlepší řešení a z něj se vytvoří první jedinec následující populace. Při dalším testování jedinců v generaci je mutace prováděna během testování předcházejícího jedince. Díky tomuto překrytí je efektivněji využit výpočetní čas a jednotky na sebe nemusí tolik čekat.

Provádění mutací je možno dělat dvěma způsoby — řízenou a neřízenou mutací. Neří-

Registr	Délka (bitů)	Popis
Registr 0 (čtení)	1	indikace ukončení běhu
	$I - 2 + log_2(O) + 1$	počet výstupů se zkratem
	$I - 2 + log_2(O) + 1$	počet výstupů s neustáleným výstupem
	$I - 2 + log_2(O) + 1$	počet výstupů s degradovanou hodnotou
	$I - 2 + log_2(O) + 1$	počet výstupů se správnou hodnotou
	1	validní konfigurace
	1	signalizace připojeného vstupu do *source*
Registr 1 (zápis)	27	data konfiguračního slova
	5	adresa zápisu do konfiguračního slova
Registr 2 (zápis)	$O \cdot 2^{I-2}$	referenční výstup, kde 1 odpovídá vysoké a 0 nízké hodnotě
Registr 3 (zápis)	1	zahájení zápisu konfiguračního slova
	1	zahájení běhu simulace ve VRC
	1	ukončení běhu simulace ve VRC
	1	ukončení zápisu konfiguračního slova
Registr 4 (čtení)	$log_2(R \cdot C) + 1$	počet aktivních tranzistorových elementů
	$log_2(R \cdot C) + 1$	počet aktivních spojovacích elementů

Tabulka 7.1: Mapování registrů

zená mutace v konfiguračním řetězci náhodně zamění některý z bitů. Tato operace je prováděna vícekrát podle konfigurace. Její výhodou je rychlost a možnost přenesení jen zmodifikované části chromozomu, nevýhodou je však velké množství nevalidních řešení. Pokud nejsou dodrženy podmínky spojení elementů uvedené v kapitole 5.4, nemá smysl ani spouštět simulaci, protože nebude validně fungovat. Testování validity konfigurace je prováděno v HW jednotce a je signalizováno bitovou informací v registru 0. Při testování vykazovala tato verze trochu vyšší rychlost, ale algoritmus velmi pomalu konvergoval ke správnému řešení.

Druhou možností je sémanticky řízená mutace. Chromozom se do zařízení odesílá ve formě binárního řetězce tak, aby neobsahoval žádný redundantní bit. Procesor však nedokáže přistupovat k paměti po krátkých řetězcích, které navíc nejsou zarovnané na požadovaných 32 bitů. Pro chromozom je proto v operační paměti uložen nejen výsledný konfigurační řetězec v binární formě, ale také reprezentace jednotlivých položek nastavení, kdy každá položka je tvořena jedním 32 bitovým číslem. Náhodně se vybere položka, která se bude měnit. Podle nastavení dalších dvou položek daného elementu se vygeneruje nová náhodná hodnota tak, aby nedošlo k porušení podmínek uvedených v tabulce 5.1 v kapitole 5.4. Současně se při změně elementu upraví odpovídající bity v binární reprezentaci. Tento způsob mutace je pomalejší díky složitějším výpočtům, takže pro zrychlení bylo zavedeno časové překrývání HW simulace a SW mutace následujícího jedince. Negeneruje však nevalidní řešení a tudíž by měl algoritmus konvergovat rychleji.

Kapitola 8

Softwarové nástroje

Abchom mohli ověřit korektní funkci evolučního algoritmu a také vyhodnotit výkonnost navrženého řešení, byla vytvořena také softwarová implementace. Dále byly pro zrychlení vývoje a testování implementovány další aplikace pro PC.

8.1 Referenční implementace

Tato metoda byla implementována na 64 bitovém CPU a pak byla přenesena na procesor ARM. Díky použití univerzálního překladače GCC můžeme porovnat výkonnost v evoluční úloze jak čipu ARM, tak i klasických 64 bitových CPU (Intel nebo AMD). Aplikace je napsána za pomoci objektového programovacího jazyka C++. Jako nejefektivnější úroveň abstrakce se jevilo mít každý element jako objekt. Pokud by byl každý typ elementu jako samostatná třída objektu, bylo by nutné pro každý chromozom znovu instanciovat objekty.

Na začátku jsou vygenerovány náhodně chromozomy podle omezení, aby nebylo zapojení nevalidní. Mřížka elementů je vyresetována a dojde k jejímu přenastavení podle chromozomu. Vazby jsou vytvořeny pomocí paměťových referencí. Tím se vyhneme postupnému testování konfiguračního vektoru a dvojitému přístupu do paměti. V procesu aktivace jednotlivých elementů se aktivují elementy připojené k výstupu. Ty potom rekurzivně aktivují i své rodiče (a v případě spoje i následovníky). Vzhledem k tomu, že jsme schopni jednoduše simulovat řešení s maximálním LBACK parametrem, není funkce propojky implementována.

Poté se přistupuje k otestování všech vstupních vektorů. Všechny elementy, které jsou připojené k některému ze vstupů nebo napájecímu napětí V_{CC} nebo GND, dostanou na vstup hodnotu. Dále všechny aktivní elementy synchronně spočítají svůj výstup. Teprve až jsou všechny výstupy spočítané, dojde k propagaci hodnot na další elementy. Při propagaci hodnot se vyhodnocuje i to, jestli nedošlo ke zkratu při zpětnovazebním přenosu. Vazba opačným směrem totiž může být z více elementů ($n : 1$) a může dojít ke spojení nekompatibilních hodnot. Pokud jsou všechny hodnoty kompatibilní, zůstává při spojení silnější hodnota. Tento proces se opakuje v n iteracích, kdy n odpovídá součtu počtů sloupců a aktivních elementů typu spoj. Není zde využito počtu aktivních sloupců jako v hardwarové verzi, protože bylo experimentálně ověřeno, že výpočet zabere většinou více času, než provedení přesného počtu iterací.

Pokud došlo v simulaci ke zkratu, který je indikován výjimkou, simulace daného vektoru se přeruší a započítá se do fitness hodnoty penalizační hodnota. Ke stejné penalizaci dojde i tehdy, kdy po dokončení simulace vektoru dochází ke změnám výstupní hodnoty. To se

stane tehdy, kdy v obvodě existuje zpětná vazba, která způsobuje oscilaci výstupu. Když se simulace dostane až do ustáleného stavu, je výstup porovnán s referenčním výstupem a podle toho je připočítán k fitness hodnotě správný koeficient. Pokud vývojář povolí i oslabenou hodnotu na výstupu místo silné, dojde k připočítání menšího koeficientu.

Jakmile se provede testování všech vstupních vektorů, může být k fitness připočítaný bonus za úplné splnění, pokud všechny výstupy odpovídaly referenčním hodnotám. Dále jsou zohledněny nevyužité tranzistory a nevyužité spoje, kdy za nevyužitý tranzistor je větší bonus, protože je v implementaci výsledného hardwaru náročnější, než spoj.

Po dokončení testování všech chromozomů je nejlepší chromozom zmutován. Mutace probíhá ve dvou krocích. První krok je výběr pozice vstupu nebo funkce, který bude mutován. Až podle dané pozice je podle nastavení ostatních dvou parametrů elementu vygenerována nová náhodná hodnota. Mutace totiž musí vždy vytvořit pouze validní řešení.

Jakmile skončí mutace všech chromozomů, tak se znovu všichni jedinci vyhodnotí, jak nám určuje CGP. Při správném výstupu dojde ke zvýšení fitness, pokud je výstup správný, ale hodnota je oslabena, bude přičteno méně bodů. Pokud v průběhu simulace dojde ke zkratu, tak se fitness mírně vylepší, protože bylo experimentálně ověřeno, že je to výhodnější, než penalizace.

Tato metoda umožňuje paralelní přístup na úrovni vláken, který však není z důvodu přenositelnosti na výpočetní jednotku ARM implementován. Každé vlákno by mělo vlastní instanci identické CGP mřížky a rozložení testovacích vektorů by se rozdělilo mezi jednotlivé výpočetní jednotky. Aplikace však není vhodná pro použití masivního paralelismu například s využitím GPU díky velké diverzitě jednotlivých elementů při rozdělení výpočetních jednotek pro jednotlivé elementy. Pokud bychom je dělili pro jednotlivé vstupy, tak by došlo k velké diverzitě jednotlivých vláken a klesal by výpočetní výkon.

Tato implementace pracuje identicky jako verze s využitím SW akcelerační jednotky s tím rozdílem, že dokonce umožňuje použít jiný parametr LBACK. Toho bude využito pro ověření vlivu tohoto parametru na úspěšnost evoluce.

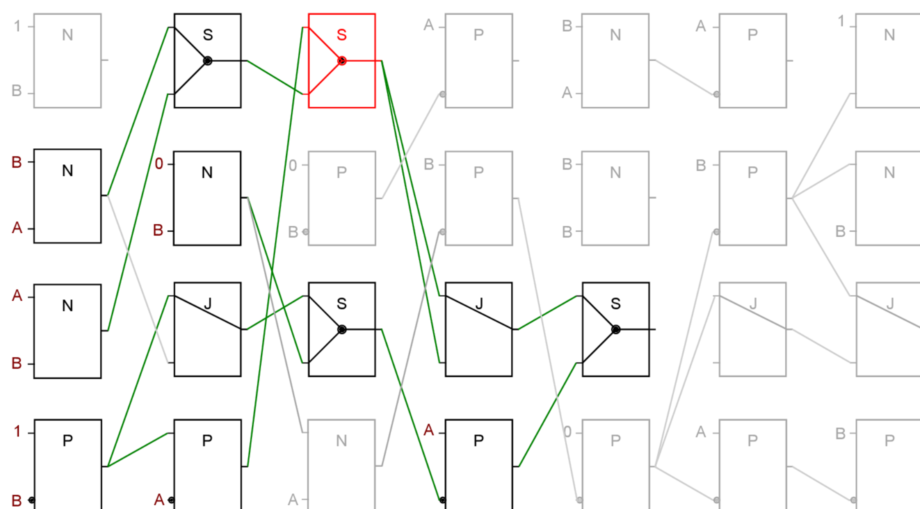
8.2 Pomocné nástroje

Pro rychlý vývoj byly vytvořeny další pomocné nástroje na počítač. Jedná se o vizualizér, který daný chromozom převede do skutečného zapojení a umožní i jeho simulaci v programu NGSpice. Druhým programem je generátor umožňující vytvořit chromozom pro testování. Posledním z nástrojů je nástroj pro výpočet parametrů pro komunikaci. Všechny nástroje jsou psané v jazyce C# a jsou spustitelné na platformě Microsoft .NET 4.

8.2.1 Vizualizér

Prvním z používaných nástrojů vytvořených pro účel této práce je vizualizér konfigurace chromozomů. Jedná se o program, který vstupní genotyp zakódovaný v binární podobě přenese do grafické podoby fenotypu. Kromě toho provede i proces aktivace popsaný výše a zvýrazní pouze prvky, které ovlivňují výstup. Na obrázku 8.1 je znázorněná vizualizace pro evolučně navržený dvouvstupý obvod XNOR. Červeně zvýrazněný spoj označuje výstup.

Mimo zobrazení nástroj umožňuje převod chromozomu do formátu konfigurace softwarové verze programu. Tato verze využívá z důvodu složitosti přístupu k jednotlivým bitům rozdílnou interpretaci. Aplikace však umožňuje jednostranně převést binární chromozom do konfigurace objektů C++.



Obrázek 8.1: Výstup vizualizéru pro obvod XNOR na mřížce 7x4

Poslední funkcí programu je export binárního chromozomu do zapojení *ngpice*. Jednotlivé elementy jsou interpretovány jako bloky, tudíž aplikace neprovádí žádnou optimalizaci, jako je například odstranění spojů a propojek. Tyto operace jsou přenechány simulačnímu nástroji. Simulátor však neumožňuje vytvořit spoj mezi jednotlivými vodiči, proto se využívají ideální rezistory s odporem 0 Ω. Využitá technologie tranzistorů je TMSC 0.25 μm. Program umožňuje po nastavení správné cesty spustit program *ngpice*. Po spuštění simulace můžeme zobrazit jednotlivé signály, kde vstupní jsou označeny jako $in.0$ až $in.X$ a výstupní jako $out.0$ až $out.Y$. Výstupní zapojení pro SPICE však můžeme nechat exportovat a provést vlastní úpravy.

8.2.2 Generátor

V průběhu procesu simulování HW jednotky v počítači bylo nutné vytvářet různé testovací případy. Proto byl vytvořen nástroj, který je nadstavbou vizualizéru, umožňující generovat konfigurační slova. Tento program byl použit pro generování všech testovacích případů, které jsou ve skriptech VHDL.

8.2.3 Nástroj pro výpočet parametrů

Dalším z nástrojů je utilita pro generování konstant pro kód procesorové jednotky podle parametrů VRC v akcelerační jednotce. Tyto konstanty slouží k výpočtu pozice jednotlivých genomů v binárně zakódovaném chromozomu. Pro výpočet se často používá binární logaritmus, který je možné vypočítat pomocí smyčky `while`, ale potom by všechny konstanty musely být uloženy jako proměnné. Konstanty jsou však definovány jako čísla přímo ve zdrojovém kódu, protože se nad nimi provádí řada výpočtů a při použití pevně definovaných konstant může kompilátor provádět větší množství optimalizací a provést jednotlivé výpočty dopředu. Aplikace tyto konstanty vypočítá a uživateli poskytuje výstup ve formě definice jazyka C. Vzhledem k tomu, že máme čtyři možné konfigurace funkce elementů, bude konstantní proměnná dále označená jako T vyjadřovat délku tohoto nastavení v chromozomu a bude 2. Parametry I, O, C a R označují počet vstupů, výstupů a velikost mřížky,

které jsou generickými parametry HW akcelerační jednotky, a jsou také uloženy v kódu jako definované konstanty.

Pro první sloupec chromozomu každý ze vstupů definuje řetězec délky odpovídající definici použitého vstupu.

$$IF = log_2(I) \tag{8.1}$$

V dalších sloupcích se vstup vybírá nejen podle vstupů VRC, ale také z předcházejícího sloupce.

$$IN = log_2(I + R) \tag{8.2}$$

Zbývá definovat délku řetězce na konci chromozomu definující element použitý jako výstup.

$$OS = log_2(R \cdot C) \tag{8.3}$$

Celková délka konfiguračního slova chromozomu, ze které se dělením 27 získá počet slov odesílaných do HW jednotky, bude vypočítána následovně.

$$CL = R(2 \cdot IF + T) + R(C - 1)(2 \cdot IN + T) + O \ \cdot OS \tag{8.4}$$

Další konstantou potřebnou v kódu je délka konfigurace očekávaných výstupů

$$OC = I \ + log_2(O) - 1 \tag{8.5}$$

Pro získání počtu využitých elementů potřebujeme znát délku této konfigurace

$$EC = log_2(R \cdot C) + 1 \tag{8.6}$$

Kapitola 9

Výsledky a experimenty

V této kapitole si představíme dosažené experimentální výsledky. V první části se budeme věnovat vyhodnocení parametrů navrženého akcelerátoru, jako jsou nároky na zdroje v FPGA, a dále dosaženému urychlení. V druhé části budou ukázány výsledky úspěšnosti návrhu a optimalizace na úrovni tranzistorů využívající navrženou evoluční techniku. Poslední část se věnuje dosaženým experimentálním výsledkům návrhu konkrétních obvodů.

9.1 Výsledky syntézy navrženého řešení

Konfigurace obvodu Zynq obsahující akcelerační jednotku, procesorovou jednotku a propojovací součásti byla syntetizována a mapována v prostředí Vivado. Akcelerátor byl implementován s využitím vývojového kitu Xilinx ZC702 obsahující čip Zynq XC7020, což je druhá nejmenší verze rodiny Zynq. Nároky na zdroje programovatelné logiky vyjádřené v počtu look–up tabulek (LUT) a flip–flop obvodů (FF) v závislosti na počtu primárních vstupů a velikosti VRC jsou uvedeny v tabulce 9.1. Můžeme vidět to, že s rostoucím počtem vstupů není nárůst nároků na zdroje velký, dokonce v některých případech dokonce mírně klesají, což je zřejmě způsobeno lepším využití multiplexorů a dalších obvodů. Největší vliv na využití zdrojů má počet řádků VRC, protože se zvyšujícím počtem řádků roste počet vstupů obvodů pro zpracování zpětné vazby. Tabulka vyjadřuje nároky na zdroje pro jeden primární výstup, s rostoucím počtem výstupů je nárůst pouze minimální způsobený pouze alokací dalšího multiplexoru pro výběr konkrétní hodnoty.

Velikost VRC		**2 vstupy**		**3 vstupy**		**4 vstupy**		**5 vstupů**	
7×4	LUT	8 279	16 %	7 848	15 %	7 818	15 %	8 205	15 %
	FF	1 913	2 %	1 974	2 %	1 979	2 %	1 984	2 %
8×6	LUT	15 461	29 %	16 436	31 %	15 103	28 %	15 884	30 %
	FF	2 711	3 %	2 728	3 %	2 733	3 %	2 738	3 %
10×8	LUT	28 525	54 %	29 519	55 %	28 116	53 %	29 007	55 %
	FF	3 925	4 %	3 946	4 %	3 951	4 %	3 956	4 %

Tabulka 9.1: Nároky na zdroje

Na základě využití LUT tabulek si můžeme všimnout možnosti dalšího urychlení, a to že pro mřížku 7×4 bychom mohli fitness jednotku až šestkrát replikovat, čímž bychom proces evaluace urychlili, podobně jako v jiných pracích, například při návrhu obrazových filtrů [13]. Podobně i pro mřížku o velikost 8×6 by bylo možné akcelerační jednotku umístit na

čip až třikrát. Pro největší syntetizovanou mřížku 10×8 přesahují nároky na zdroje 50 %, a proto takto velkou jednotku nejsme schopni replikovat.

Dalším významným parametrem obvodu je jeho příkon, který dělíme na statický a dynamický. Statický příkon vzniká přímo ve struktuře obvodu, je dán fyzikálními vlastnostmi tranzistorů a nemá na něj vliv frekvence. Oproti tomu dynamický příkon je přímo závislý na frekvenci obvodu. Celkově, s přesností napěťové analýzy nástroje Vivado, bude celková spotřeba obvodu 1,553 W. Nejvíce se na napájení podílí procesorový systém PS7, který spotřebuje energii 1,353 W. Při změně velikosti obvodu se napájení mění v řádech desítek mW, což je zanedbatelný narůst v poměru k energii spotřebovanou statickým unikajícím proudem a procesorovým systémem. Celkově se tedy statický příkon podílí na celkové spotřebě 0,149 W a dynamický včetně procesorového systému energií 1,404 W. Čip by se měl ohřívat na teplotu přibližně 43 °C.

9.2 Rychlost obvodu

Cílem akcelerátoru je urychlit dobu simulace kandidátního řešení a tím urychlit proces evolučního návrhu obvodů na úrovni tranzistorů. Nejdůležitějším parametrem je tedy celkové zrychlení. Vyhodnocení je ukázané v tabulce 9.2, jako referenční řešení je považována SW implementace vykonávající identickou funkci, jako řešení využívající HW akcelerační jednotku. Všechny testované případy evolučního návrhu obvodů na tranzistorové úrovni byly spuštěné s úlohou evolučního návrhu dvouvstupého hradla NAND na mřížce 10×8 s 20 populacemi a omezením na 500 000 generací a s osmi mutacemi. HW akcelerační jednotka byla konfigurovaná na taktovací frekvenci 40 MHz. Po syntéze této jednotky omezovala kritická cesta maximální frekvenci přibližně na 130 MHz, ale vzhledem k velké náročnosti na zdroje po mapování a přičtení transportního zpoždění klesla frekvence na 45 – 50 MHz. Proto byl v IP jádře procesorové jednotky nastaven takt externích hodin FCLK0 na 40 MHz.

Platforma	**2 vstupy**		**3 vstupy**		**4 vstupy**		**5 vstupů**	
	t	**s**	**t**	**s**	**t**	**s**	**t**	**s**
Xeon – ngspice	49 000	-	60 000	-	100 000	-	237 000	-
Xeon – diskrétní	34	1.00	73	1.00	180	1.00	253	1.00
Zynq – diskrétní	130	0.26	320	0.23	762	0.27	950	0.27
Zynq – SW/HW	41	0.83	44	1.65	49	3.67	54	4.68
Zynq – HW	2	17.00	3	24.34	6	30.00	11	23.00

Tabulka 9.2: Průměrná doba evaluace jednoho kandidátního řešení t v μs a zrychlení s

Verze, které nepotřebovaly k běhu HW akcelerační jednotku, byly testovány na serveru s procesorem Intel Xeon E5-2630 s taktovací frekvencí 2,30 GHz a s 16 GB RAM. Při testování nebyly na počítači spuštěné žádné další klientské aplikace a procesor byl tedy maximálně využit. Jako první případ byla spuštěna analogová simulace s využitím simulátoru *ngspice* na úrovni tranzistorů TSMC. Tato simulace byla spuštěna pouze pro jeden testovaný chromozom, nebyla spuštěna v úloze evolučního návrhu. Slouží pro porovnání doby trvání analogové simulace oproti diskrétní. Na stejném procesoru byla spuštěna i softwarová implementace navrženého algoritmu využívající synchronního přístupu k předávání dat mezi jednotlivými elementy. Ukázalo se, že využití diskrétní simulace umožní až $1000\times$ rychlejší vyhodnocení obvodu. Tato verze byla upravena pro běh na procesoru ARM v čipu Zynq taktovaném na frekvenci 667 MHz. Vyhodnocení každého řešení bylo v průměru $3,5\times$

pomalejší než u procesoru Intel Xeon, což koresponduje s nižší taktovací frekvencí tohoto procesoru.

Rychlost obvodu využívajícího hardwarové akcelerační jednotky je uvedena na řádku „Zynq – SW/HW“. Je vidět, že už od 2 vstupů dosahujeme zrychlení oproti verzi určené pro procesor Intel, přestože se jedná o energeticky úspornější řešení. Abychom vyhodnotili vliv režie komunikace, obsahuje tabulka i rychlost vyhodnocení testovaného obvodu pouze v akcelerační jednotce. V tomto případě dosahujeme nejvyšších rychlostí a je vidět, že snížením náročnosti komunikace jsme schopni dosáhnout ještě většího zrychlení. Komunikace vyžaduje čas pohybující se okolo 40 μs bez ohledu na počet vstupů.

Při evolučním návrhu či optimalizaci využíváme operátoru mutace. Jak bylo zmíněno v kapitole 7.4, mutace může probíhat sémanticky na úrovni jednotlivých genů, nebo také pouze binární mutací v chromozomu. Vliv typu mutace na rychlost vyhodnocení obvodu při evoluci na čipu Zynq s využitím akcelerační jednotky a se stejnou konfigurací jako v předchozích testovaných případech je znázorněn v tabulce 9.3. Můžeme si všimnout toho, že se typ mutace podílí na rychlosti méně než 10 %. Tímto malým nárůstem docílíme rychlejší konvergence z důvodu omezení testování nevalidních řešení.

Primárních vstupů	**Binární mutace**	**Mutace genů**	**Nárůst**
2	41 μs	43 μs	4,9 %
3	44 μs	46 μs	4,5 %
4	49 μs	52 μs	6,1 %
5	54 μs	59 μs	9,3 %

Tabulka 9.3: Vliv typu mutace na dobu ohodnocení jednoho kandidátního řešení

Další porovnávanou vlastností je poměr rychlosti a příkonu. Budeme sledovat počet ohodnocených kandidátních řešení na jeden joule (Ws). Jeho hodnotu získáme pomocí vzorce

$$m = \frac{1}{t \cdot P} \tag{9.1}$$

kde t odpovídá času evaluace jednoho řešení a P příkonu jednotky. Vzhledem k tomu, že není možné změřit přesně příkon samotného čipu Zynq, jelikož vývojový kit obsahuje řadu dalších nepoužívaných periferií, které však jsou připojeny ke zdroji napájení, budeme využívat analýzu příkonu z nástroje Vivado. Pro Zynq systém bez využití HW části budeme počítat příkon 1,343 W, při použití akcelerační jednotky budeme počítat s příkonem 1,553 W, který čip využívá při použití největší mřížky. Podobně i spotřebu počítače, na kterém běžela SW verze algoritmu, není možné přesně měřit, protože k němu není fyzický přístup, tak budeme vycházet z informací udávaných v dokumentaci procesoru. Procesor Intel Xeon má udávanou spotřebu 95 W[1]. Jako referenční proto vezmeme tuto hodnotu, přestože využíváme pouze jednoho jádra z šesti dostupných. Naopak však nebudeme zohledňovat spotřebu dalších součástí, jako je základní deska, RAM paměť nebo ventilátory, které mohou u běžné sestavy představovat dalších až 130 W. Celkovou výkonnost na energii 1 J vidíme v tabulce 9.4.

Z vypočtených hodnot plyne to, že implementace na klasickém procesoru je elektricky nejnáročnější. Přestože je diskrétní SW implementace na procesoru Zynq téměř čtyřikrát pomalejší, v energetické náročnosti procesor Intel předčila více než 12 ×. Jako nejvýhodnější

[1] http://ark.intel.com/products/64593/Intel-Xeon-Processor-E5-2630-15M-Cache-2_30-GHz-7_20-GTs-Intel-QPI

Platforma	2 vstupy	3 vstupy	4 vstupy	5 vstupů
Xeon – ngspice	0.21	0.18	0.11	0.04
Xeon – diskrétní	309.60	144.20	58.48	41.6
Zynq – diskrétní	5 727.70	2 326.88	977.17	783.79
Zynq – SW/HW	15 705.24	14 634.43	13 141.12	11 924.35

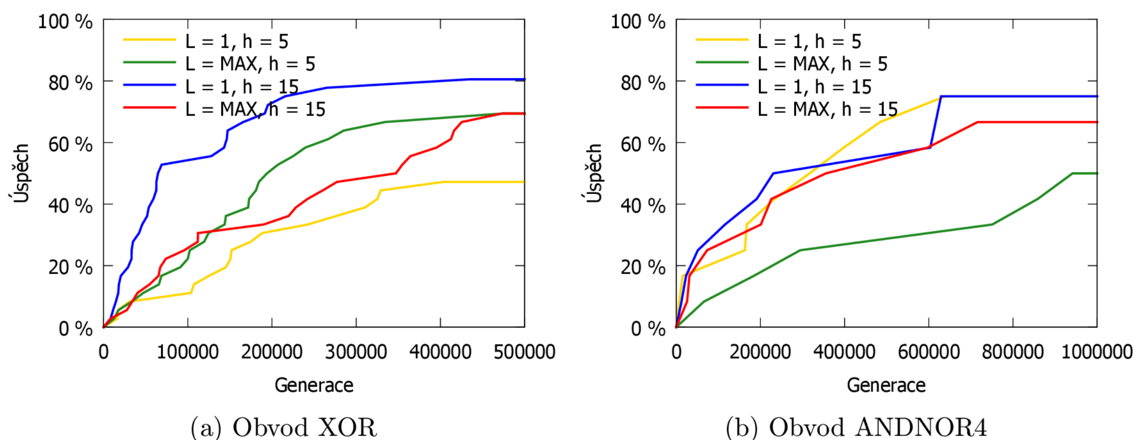
Tabulka 9.4: Výkonnost jednotlivých platforem vyjádřená jako počet ohodnocení kandidátních řešení na energii 1 J v závislosti na počtu primárních vstupů

varianta z pohledu elektrické spotřeby se jeví použít HW akcelerační jednotku, kde oproti implementaci na procesoru Intel dosahujeme zlepšení energetického využití pohybující se mezi 50 až 300.

9.3 Vliv nastavení parametrů na úspěšnost evoluce

Úspěšnost evolučního návrhu byla vyhodnocena pomocí metodiky *success proportion*. Porovnávány byly případy evolučního návrhu dvouvstupého hradla XOR a obvodu ANDNOR4. Aby byla data statisticky významná, bylo pro každé testované nastavení provedeno 35 běhů. Vzhledem k tomu, že jsem mimo jiné porovnával vliv parametru LBACK, úspěšnost evoluce musela být vyhodnocena v ekvivalentní softwarové implementaci bez využití HW akcelerační jednotky.

Abychom zjistili, jestli spuštěním evolučního návrhu na úrovni tranzistorů na akcelerační jednotce, může pracovat s LBACK $= 1$, nedojde-li k zhoršení vlastností tohoto procesu, byl jako první zkoumán vliv tohoto parametru. Tento parametr určuje, o kolik sloupců nazpět je možné připojit vstup s výstupem. Na grafech kromě parametru L ještě uveden vliv s počtu mutací h, který vyjadřuje, kolik prvků v chromozomu bude maximálně modifikováno. Pro mřížku 10×8 mělo toto omezení pozitivní vliv, protože došlo k omezení prohledávaného prostoru. Výsledky můžeme vidět na obrázku 9.1.

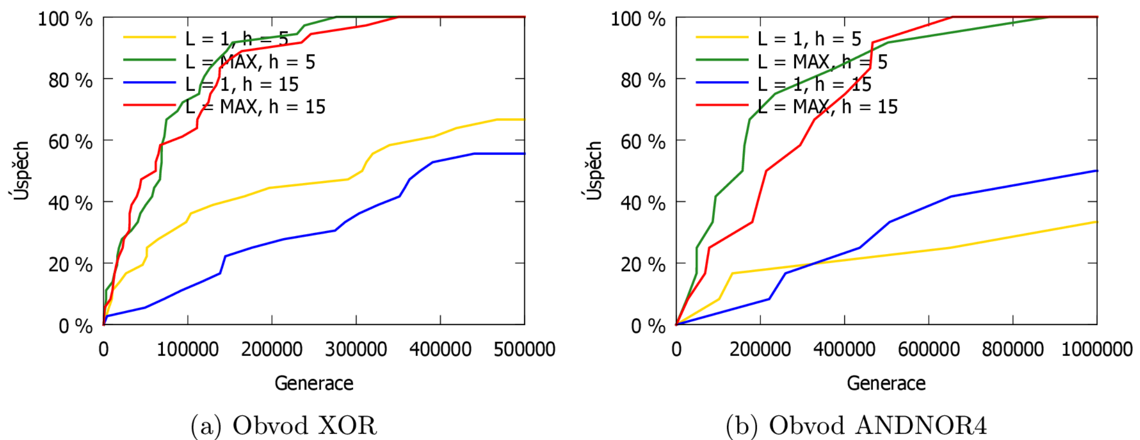


(a) Obvod XOR

(b) Obvod ANDNOR4

Obrázek 9.1: Vliv parametru LBACK na úspěšnost evoluce pro mřížku 10×8

Ovšem při testování toho samého jevu na mřížce 7×4 znázorněné na obrázku 9.2 už má tento parametr větší vliv. Je to zřejmě způsobeno zmenšením stavového prostoru, při kterém se nalezne řešení rychleji. Rozdíl v úspěšnosti však není tak velký a spíše tedy závisí na velikosti mřížky než na tomto parametru.

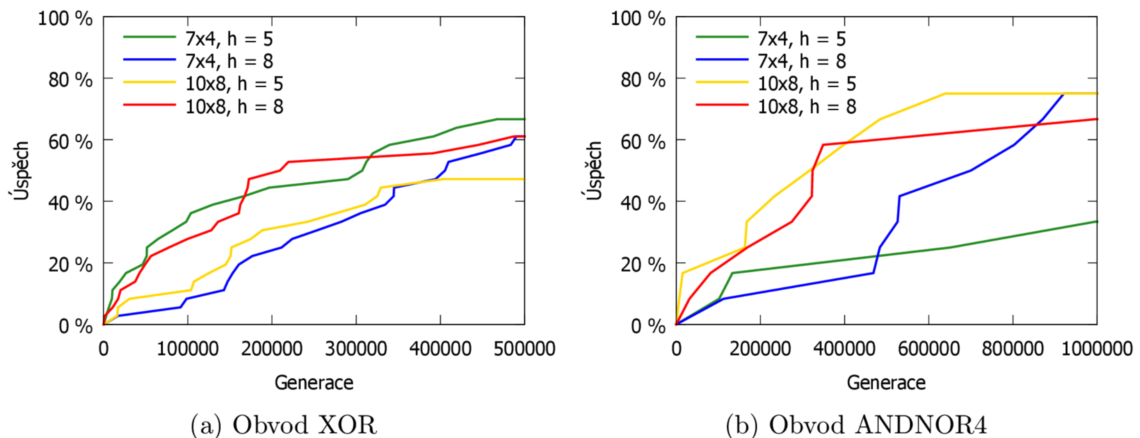


(a) Obvod XOR

(b) Obvod ANDNOR4

Obrázek 9.2: Vliv parametru LBACK na úspěšnost evoluce pro mřížku 7×4

Dalším zkoumaným faktorem je vliv velikosti mřížky. Jak bylo vidět na obrázku 9.2a, u maximálního LBACK má velikost mřížky velký vliv. Vzhledem k tomu, že nás spíše zajímá chování při LBACK $= 1$, který podporuje navržený akcelerátor, budeme zkoumat vliv velikosti pro toto nastavení. Jako vzorky byly vybrány běhy s mutací 5 a 8, protože většina běhů měla při tomto nastavení lepší výsledky, jak bude ukázáno dále. Na obrázku 9.3 můžeme vidět, že pro XOR obvod má menší mřížka v průměru lepší výsledky, ovšem jejich rozdíl není tak markantní. Oproti tomu při hledání implementace ANDNOR4 obvodu nalézáme řešení na větší mřížce rychleji, tj. stejné úspěšnosti lze dosáhnout s nižším počtem generací.



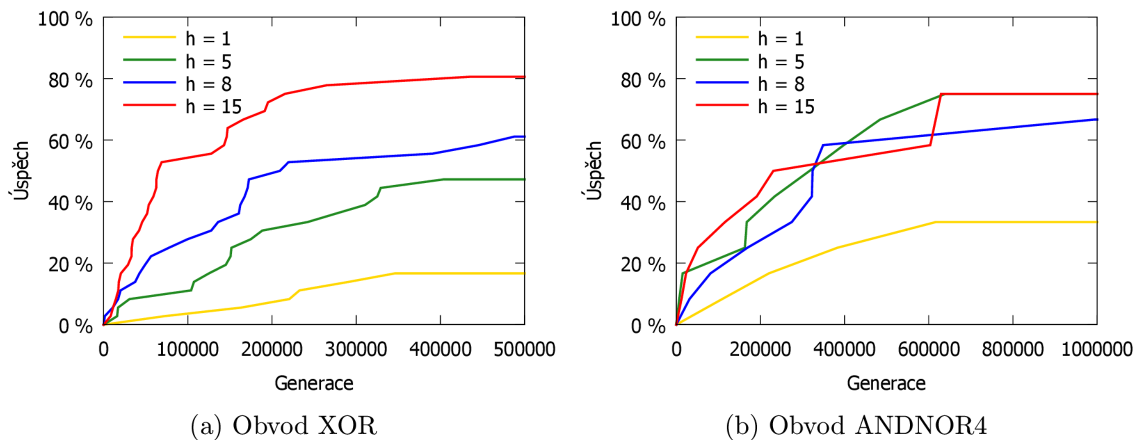
(a) Obvod XOR

(b) Obvod ANDNOR4

Obrázek 9.3: Vliv velikosti mřížky při mutacích $h = 5$ a $h = 8$

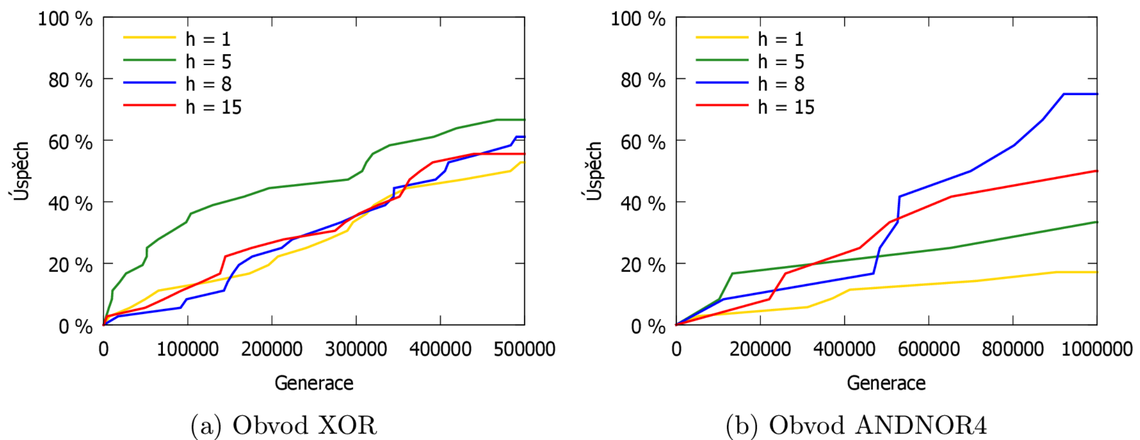
Posledním zkoumaným parametrem bude vliv velikosti mutace. Při testování na mřížce 10×8 (obrázek 9.4) se ukázalo, že nejlepších výsledků se dosahuje při vyšším počtu mutací. Z měření jednoznačně plyne, že pro malou mutaci nejsou výsledky tak dobré.

Druhým testovaným případem bude mřížka 7×4 zobrazená na obrázku 9.5. Při použití této mřížky se ukazuje, že je lepší použít menší počet mutací, než u větší mřížky. Nejlepších výsledků se dosahuje u řešení mutujících v 5, respektive v 8 genomech. Pokud přepočteme nejlepší výsledky na procenta z délky chromozomu, dostáváme se na 5,8 % pro mřížku 7×4 a na 6,3 % pro mřížku o velikosti 10×8, což zhruba odpovídá doporučení mutace 5 % [29].



(a) Obvod XOR (b) Obvod ANDNOR4

Obrázek 9.4: Vliv velikosti mutace na úspěšnost evoluce pro mřížku 10×8



(a) Obvod XOR (b) Obvod ANDNOR4

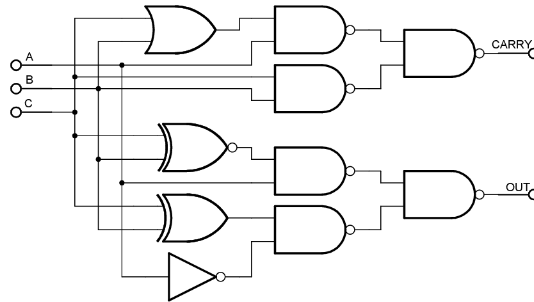
Obrázek 9.5: Vliv velikosti mutace na úspěšnost evoluce pro mřížku 7×4

Jednou z nevýhod evolučního návrhu je špatná škálovatelnost pro složitější vícevstupé obvody. Proto jsem zkusil spustit navržený algoritmus i v úloze evoluční optimalizace, která se od evolučního návrhu liší v tom, že počáteční populace není generována náhodně, ale je inicializována počátečním obvodem. Počáteční řešení bylo získáno tak, že jsme převedli obvod popsaný na úrovni hradel na úroveň tranzistorů, kdy jednotlivá hradla byla implementována pomocí následujícího počtu tranzistorů: INVERTOR 2, BUFFER 4, NAND/NOR 4, XOR/XNOR 8 a AND/OR 6.

Úloha optimalizace byla zkoušena na optimalizaci úplné jednobitové sčítačky. Jedná se o velmi často používaný modul na úrovni VLSI. Jako počáteční jedinec byl použit přepis z hradlové úrovně znázorněné na obrázku 9.6, který využívá 48 tranzistorů.

Pro tento běh bylo využito lineární uspořádání mřížky, neboli mřížky s maximálním LBACK parametrem a pouze s jedním řádkem o délce 72 elementů. Počáteční konfigurace odvozená z hradlové úrovně využívala 48 tranzistorů. Délka mřížky (počet sloupců) byla vypočtena jako $1,2$ násobek počáteční konfigurace. Při evoluční optimalizaci bylo povoleno připojovat primární vstupy k elektrodám *source*. Maximální počet generací byl nastaven na 3 000 000 a pro každé nastavení velikosti mutace bylo spuštěno 60 nezávislých běhů.

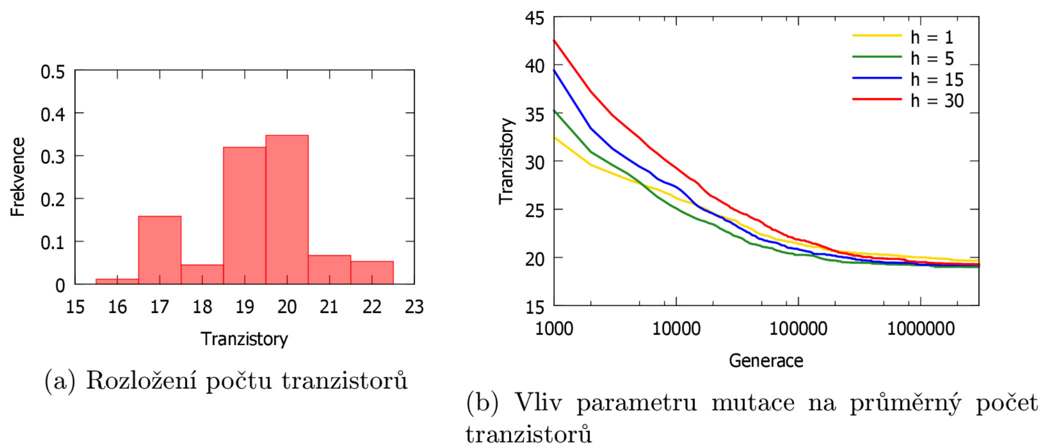
Výsledky evoluce vyjádřené ve formě histogramu udávajícího rozložení poměru počtu



Obrázek 9.6: Úplná sčítačka na hradlové úrovni použitá jako počáteční jedinec evoluční optimalizace

nalezených řešení v závislosti na počtu tranzistorů jsou uvedeny na obrázku 9.7a. Na základě získaných výsledků lze pozorovat, že nejčastěji bylo získáno řešení využívající 19 či 20 tranzistorů, což odpovídá více než 60% redukci oproti počátečnímu řešení. Nejlepší nalezené řešení bylo složeno z 16 hradel. Sledovaným průběhem bude průměrný počet tranzistorů nalezených řešení v odpovídající generaci. Rozložení počtu tranzistorů přibližně odpovídá normálnímu rozložení, a proto budeme moci pro jednotlivé konfigurace běhu určit interval důvěry. Pro přesný test normálnosti rozložení, jako je například χ^2 test, nemáme dostatečný vzorek dat, protože bylo nalezeno pouze 7 velikostí řešení.

Pro běh optimalizace jsem zkoumal vliv nastavení parametru mutace. Každé nastavení bylo testováno na 60 vzorcích a jeho vliv na optimálnost nalezených řešení můžeme vidět na obrázku 9.7b. Interval důvěry 95 % se pohyboval $\pm 0,2$ tranzistoru, což znamená, že optimalizace přinášela stabilní výsledky.



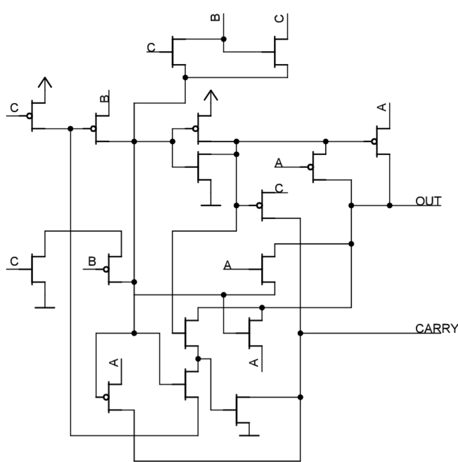
(a) Rozložení počtu tranzistorů

(b) Vliv parametru mutace na průměrný počet tranzistorů

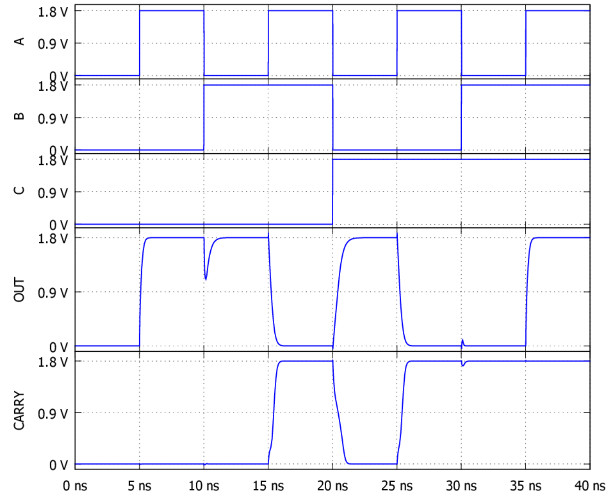
Obrázek 9.7: Statistické vyhodnocení evoluční optimalizace jednobitové sčítačky

Jeden z nalezených obvodů realizovaný za pomocí 17 tranzistorů je obvod znázorněný na obrázku 9.8a. Byl také otestován v analogovém simulátoru *ngspice*. Porovnáme-li výsledky simulace s požadovanými hodnotami, tak zjistíme, že řešení vykazuje korektní funkci při frekvenci 200 MHz a technologii TSMC. Můžeme si však všimnout, že vlivem zpoždění mezi tranzistory dochází k mírnému zpoždění na obou výstupech. Výstup této simulace je vidět na obrázku 9.8b.

Těmito výsledky bylo na dvou ukázkových případech evolučního návrhu hradel XOR a ANDNOR4 ukázáno, že parametr LBACK nemá vliv na úspěšnost evoluce a akcelerací



(a) Schéma obvodu



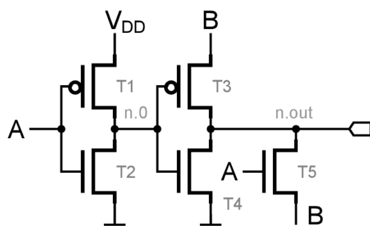
(b) Analogová simulace

Obrázek 9.8: Optimalizované řešení úplné jednobitové sčítačky využívající 17 tranzistorů

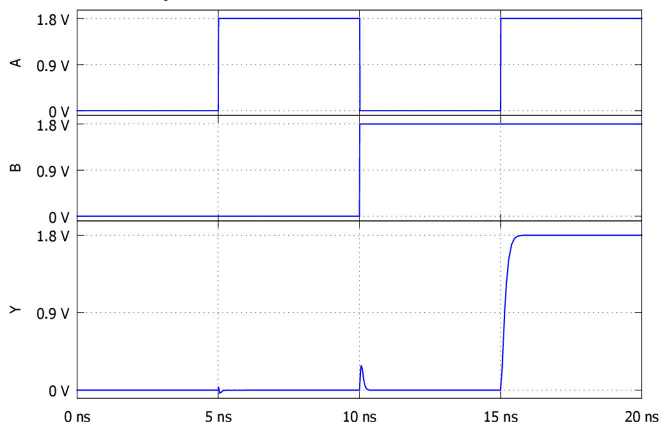
v HW akcelerační jednotce s tímto omezením nesnížíme schopnosti algoritmu evolučního návrhu. Dále jsme zjistili, že největší vliv na úspěšnost evoluce má velikost mřížky VRC. Ukázalo se, že jsme schopni úspěšně překonat problém škálovatelnosti pomocí evoluční optimalizace.

9.4 Evoluční návrh s využitím navržené metody

Korektní činnost navrženého algoritmu byla ověřena na úloze evolučního návrhu. Evoluční návrh byl spuštěn v čipu Zynq s využitím akcelerační jednotky a správnost výstupu byla poté validována analogovým simulátorem *ngspice* s technologií TSMC 0.250 μm a na frekvenci 200 MHz. Při návrhu dvouvstupých hradel byla použita konfigurace VRC o velikosti 7×4 a proces byl spuštěn s náhodnou počáteční populací. Z důvodu velikosti jsou zobrazení konfigurace CGP mřížky nalezených řešení uvedeny v příloze B. Pro hradla NAND a NOR byla nalezená klasická zapojení se čtyřmi tranzistory.



(a) Schéma



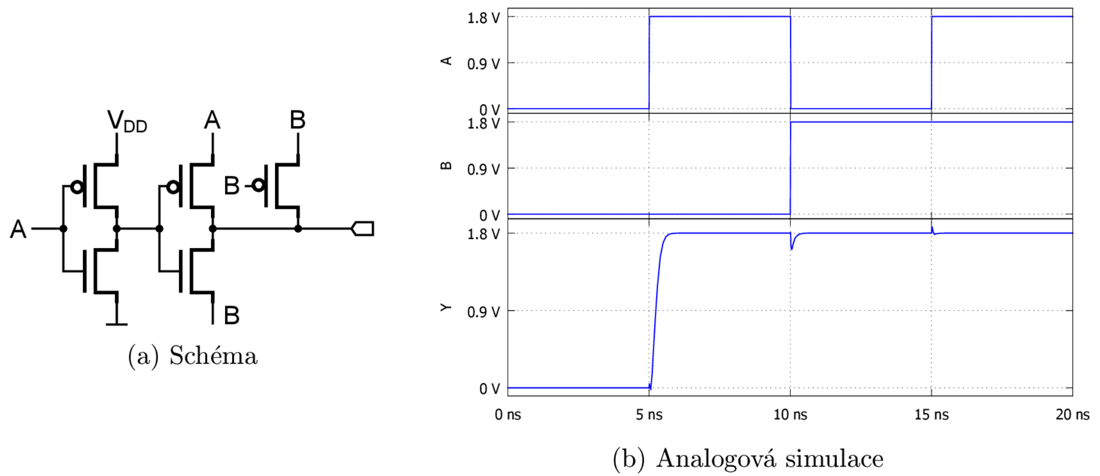
(b) Analogová simulace

Obrázek 9.9: Navržené dvouvstupé hradlo AND

Dále jsem zkoušel navrhnout hradlo AND. Pokud jsem nepovolil hledání obvodů, kde

vstupy jsou připojeny na jiné elektrody tranzistoru než je *gate*, tak jsme našli klasická řešení s kombinací NAND a implicitně přidaného invertoru. Ovšem při povolení většího proudového odběru ze vstupů jsme našli řešení znázorněné na obrázku 9.9a. Při omezení maximálního počtu generací na 50 000 a mutací 10 byl obvod navrhnut vždy.

Ze schématu lze odvodit, že tranzistory T_1 a T_2 dohromady tvoří invertor signálu A, a proto hodnota vodiče $n.0$ odpovídá $\overline{A}$. Tranzistor T_4 v případě, že je A je nulový, dává na výstup logickou 0. Tranzistory T_3 a T_5 se doplňují, protože se jedná o NMOS a PMOS se stejným vstupem, ale s komplementárním řídicím vstupem *gate*. Díky tomu má při sepnutí jeden z nich výstupů oslabený, ale druhý má silný v závislosti na úrovni signálu B. Výsledek analogové simulace je znázorněn na obrázku 9.9b.

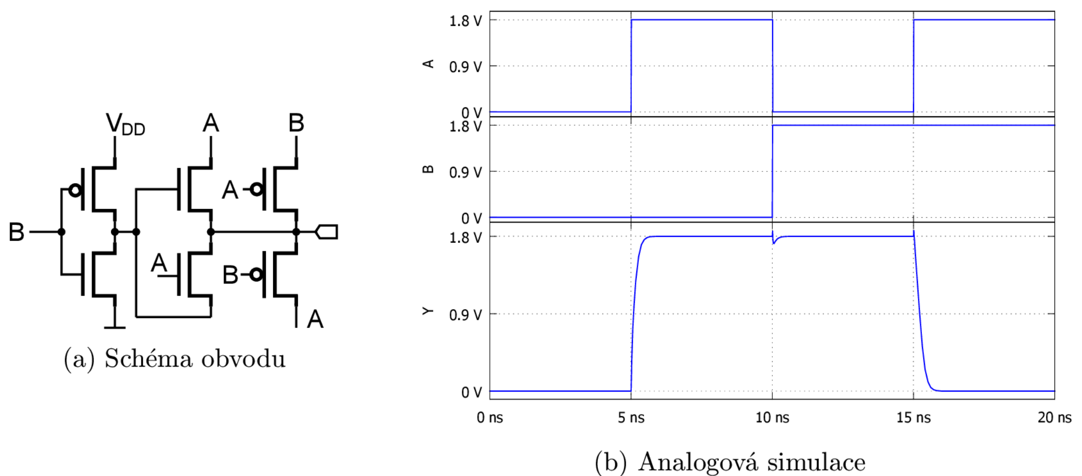


(a) Schéma

(b) Analogová simulace

Obrázek 9.10: Navržené dvouvstupé hradlo OR s povolením většího odběru ze vstupních vodičů

Dalším benchmarkovým problémem byl návrh dvouvstupého hradla OR. Jeden z výsledků je znázorněný na obrázku 9.10a. Jedná se taktéž o obvod s povolením větší zátěže vstupů využívající 5 tranzistorů a pracující na podobném principu, jako hradlo AND. Bez povolení tohoto odběru byly nalezené obvody totožné s konvenčním řešením využívající obvod NOR a invertor. Analogovou simulaci obvodu můžeme vidět na obrázku 9.10b.

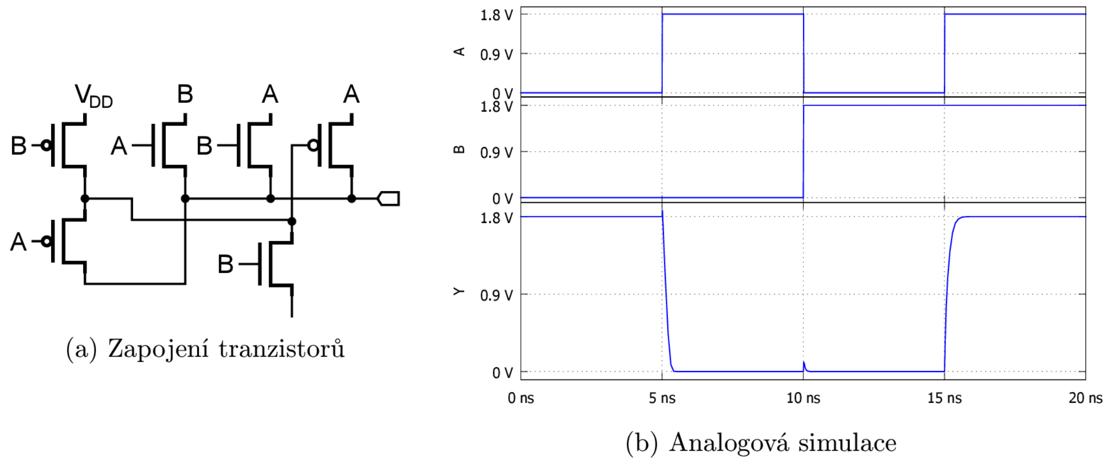


(a) Schéma obvodu

(b) Analogová simulace

Obrázek 9.11: Navržený obvod XOR

Také jsem se zaměřil na návrh složitějších dvouvstupých hradel, jako je návrh hradla XOR s povolením použití vstupních signálů, znázorněný schematicky i s analogovou simulací na obrázku 9.11. Jedná se o obvod, na němž byl demonstrován průběh diskrétní simulace v kapitole 5.4.3.

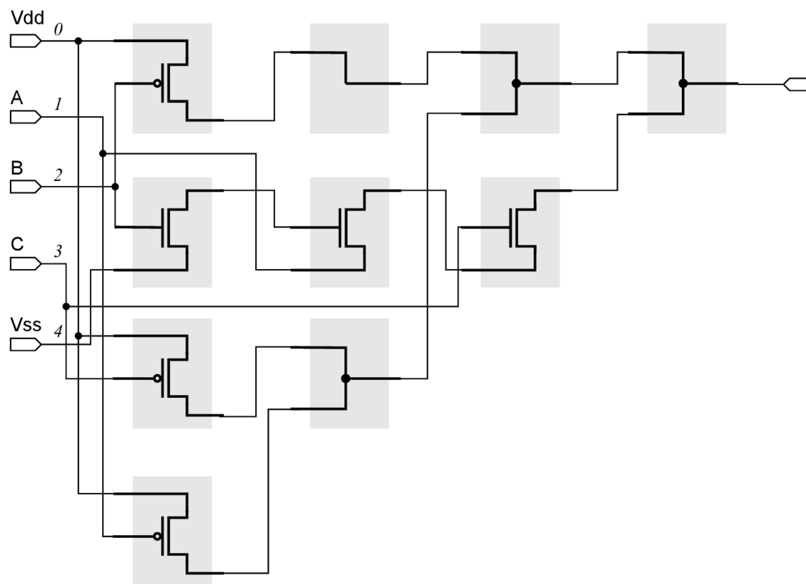


(a) Zapojení tranzistorů

(b) Analogová simulace

Obrázek 9.12: Navržený obvod XNOR s povolením většího odběru ze vstupů

Posledním testovaným dvouvstupým obvodem byl obvod XNOR znázorněný na obrázku 9.12a. Také tento obvod byl také simulován v technologii TSMC a výsledek tohoto procesu můžeme vidět na obrázku 9.12b. Stejně jako u ostatních navržených hradel byl povolen větší odběr z primárních vstupů způsobený připojením vstupu k *source* elektrodě.

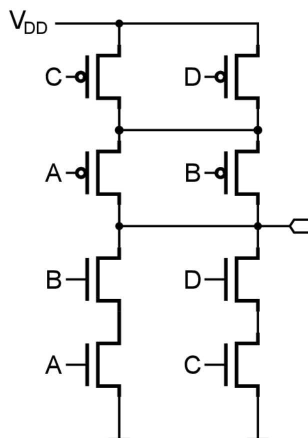


Obrázek 9.13: Nalezené řešení třívstupého obvodu NAND3 odpovídající klasickému zapojení

Kromě návrhu základních logických členů byl zkoumán i návrh vícevstupých obvodů. Jedním z testovaných obvodů bylo třívstupé hradlo NAND, jehož zapojení v CGP mřížce je zobrazeno na obrázku 9.13. V mřížce jsou znázorněné pouze aktivní elementy, kompletní znázornění mřížky je podobně jako u ostatních obvodů znázorněno v příloze B. Jeho imple-

mentace odpovídá standardnímu zapojení 3 tranzistorů NMOS a 3 tranzistorů typu PMOS, proto zde není uveden výstup analogové simulace.

Dalším z problémů byl návrh obvodu, který řeší čtyřvstupou funkci $\overline{A \cdot B + C \cdot D}$. Jedná se o případ, při kterém je nejvíce zřejmé, proč je vhodné použít návrh na úrovni tranzistorů namísto hradel. Řešení využívající hradla využívalo 16 tranzistorů, nalezené řešení znázorněné na obrázku 9.14 využívá pouze 8 tranzistorů. Jedná se o konvenční řešení, proto také není znázorněn výstup analogové simulace.



Obrázek 9.14: Příklad nalezeného řešení funkce ANDNOR4, které odpovídá optimálnímu řešení [53]

Kapitola 10

Závěr

Cílem této práce bylo navrhnout metodu evolučního návrhu obvodů na tranzistorové úrovni a tu potom akcelerovat na platformě Xilinx Zynq. V teoretickém úvodu byly představeny techniky evolučního návrhu se zaměřením na kartézské genetické programování. Dále byly ukázány možnosti akcelerace evolučního návrhu s využitím rekonfigurovatelných obvodů. Pro přiblížení chování obvodů popsaných na tranzistorové úrovni byly představeny vlastnosti MOSFET tranzistorů, jejich parametry a možnosti jejich počítačové simulace. Dále byly ukázány výsledky aktuálních výzkumů na téma návrhu obvodů na tranzistorové úrovni, a to s využitím pravé i nepravé evoluce.

V rámci diplomové práce byl navržen algoritmus umožňující evoluční návrh kombinačních obvodů na úrovni tranzistorů a hardwarový akcelerátor umožňující proces urychlit. Algoritmus vychází z chování MOSFET tranzistorů v saturovaném režimu a využívá diskretizace na šest logických úrovní. Je navržen tak, aby reflektoval všechny možnosti zapojení obvodů a s ohledem na jeho snadnou simulaci v HW. Z tohoto algoritmu vychází akcelerátor navržený s využitím moderní výkonné platformy Xilinx Zynq.

Navržená metoda evoluce a její úspěšnost byla ověřena metodikou *success proportion* v úloze evolučního návrhu obvodů hradel. Byl zkoumán vliv jednotlivých parametrů kartézského genetického programování. Při vyhodnocení výsledků bylo zjištěno, že omezení LBACK parametru v HW akcelerační jednotce má pouze minimální vliv na úspěšnost evoluce a že přesunem návrhu do této jednotky nezhoršíme schopnosti algoritmu. Při evolučním návrhu složitých obvodů se objevil problém škálovatelnosti s ohledem na počet vstupů, který byl vyřešen převodem problému návrhu na evoluční optimalizaci z vyšší, hradlové, úrovně. Tímto algoritmem se podařilo nalézt řešení složitějších obvodů, jako je například jednobitová úplná sčítačka.

Akcelerátor, který běží na frekvenci 40 MHz, dosahuje oproti referenční softwarové implementaci zrychlení až $4,86\times$ v závislosti na počtu primárních vstupů obvodu. Oproti analogovému simulátoru *ngspice* dosahuje navržený akcelerátor zrychlení až o tři řády. Také je energeticky mnohem méně náročný, kdy na jeden joule provede $50\times$ až $300\times$ více evaluací kandidátních řešení, než při implementaci na klasickém procesoru Intel Xeon.

Pomocí akcelerátoru se podařilo navrhnout řadu dvouvstupých hradel a také některá vícevstupá. Korektnost nalezených řešení byla ověřena v simulátoru *ngspice* s tranzistory reflektující parametry komerčně využívané technologie výroby integrovaných obvodů. Také se ukázalo, že diskrétní simulace nalezených řešení funguje, ale u některých komplexnějších obvodů se začal objevovat problém rychlosti přepínání vstupních signálů a v méně než 5 % případů docházelo k tomu, že tranzistory použité technologie TSMC měnily svůj stav pomalu. Na nižších frekvencích však tyto obvody fungovaly korektně.

Ačkoliv bylo dosaženo oproti existujícím pracím lepších výsledků, zejména přesnější a rychlejší simulace, stále existuje prostor pro vylepšení navrženého algoritmu. Jednou z možností je použití více VRC v programovatelné logice akcelerátoru pro maximalizaci využití zdrojů a zvýšení průměrné rychlosti evaluace kandidátních řešení. Jinou cestou zvýšení výkonu je optimalizace komunikační linky mezi akcelerační jednotkou a procesorem, kde se nabízí možnost přesunu vytváření populace z rodiče do akcelerační jednotky.

Jako jedna z možných cest vylepšení algoritmu pro zvýšení přesnosti simulace je reflektování předchozího stavu tranzistorů při testování dalšího stimulujícího vektoru. Tímto přístupem by bylo zřejmě možné měřit i zpoždění změny stavu použitých tranzistorů a základním způsobem analyzovat spotřebu obvodu.

Literatura

[1] Žaloudek, L.; Sekanina, L.: Transistor-level Evolution of Digital Circuits Using a Special Circuit Simulator. In *Evolvable Systems: From Biology to Hardware*, LNCS 5216, Springer Verlag, 2008, ISBN 978-3-540-85856-0, s. 320–331.

[2] Angeline, P. J.: An Investigation into the Sensitivity of Genetic Programming to the Frequency of Leaf Selection During Subtree Crossover. In *Proceedings of the First Annual Conference on Genetic Programming*, GECCO '96, Cambridge, MA, USA: MIT Press, 1996, ISBN 0-262-61127-9, s. 21–29.

[3] Baker, R.; of Electrical, I.; Engineers, E.; aj.: *CMOS: Circuit Design, Layout, and Simulation*. číslo sv. 1 in IEEE Press Series on Microelectronic Systems, Wiley, 2008, ISBN 9780470229415.

[4] Cruz, A.; Mukherjee, S.: PLAGA: a highly parallelizable genetic algorithm for programmable logic arrays test pattern generation. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, ročník 2, 1999, s. 944–951.

[5] Dobai, R.; Sekanina, L.: Image Filter Evolution on the Xilinx Zynq Platform. In *Proceedings of the 2013 NASA/ESA Conference on Adaptive Hardware and Systems*, IEEE Circuits and Systems Society, 2013, ISBN 978-1-4673-6381-5, s. 164–171.

[6] Dobai, R.; Sekanina, L.: Towards Evolvable Systems Based on the Xilinx Zynq Platform. In *2013 IEEE International Conference on Evolvable Systems (ICES)*, Proceedings of the 2013 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE Computational Intelligence Society, 2013, ISBN 978-1-4673-5869-9, s. 89–95.

[7] Garner, W.: Probability, Mean and Median. URL http://math.ucsd.edu/~wgarner/reference/math10c_su10/lectures/probability_mean_and_median.pdf, 2010 [cit. 2014-02-11].

[8] Gold Standard Simulations Ltd (GSS): Randomspice [online]. URL http://www.goldstandardsimulations.com/services/circuit-simulation/random-spice/, 2010 [cit. 2013-09-11].

[9] Guccione, S.; Levi, D.: JBits: A Java-Based Interface to FPGA Hardware. Xilinx. URL http://www-inst.eecs.berkeley.edu/~cs294-59/fa10/resources/Xilinx-history/jbits.pdf

[10] Guccione, S.; Levi, D.; Sundararajan, P.: JBits: Java based interface for reconfigurable computing. Xilinx.

URL
http://users.utcluj.ro/~baruch/media/resources/JBits/JBitsMAPPLD.pdf

[11] Harding, S.; Banzhaf, W.: Hardware Acceleration for CGP: Graphics Processing Units. In *Cartesian Genetic Programming*, editace J. F. Miller, Natural Computing Series, Springer Berlin Heidelberg, 2011, ISBN 978-3-642-17309-7, s. 231–253.

[12] Hounsell, B.; Arslan, T.: Evolutionary design and adaptation of digital filters within an embedded fault tolerant hardware platform. In *Evolvable Hardware, 2001. Proceedings. The Third NASA/DoD Workshop on*, 2001, s. 127–135.

[13] Hrbáček, R.; Šikulová, M.: Coevolutionary Cartesian Genetic Programming in FPGA. In *Advances in Artificial Life, ECAL 2013, Proceedings of the Twelfth European Conference on the Synthesis and Simulation of Living Systems*, MIT Press, 2013, ISBN 978-0-262-31709-2, s. 431–438.

[14] Hurvich, C.: Confidence Intervals for the Mean; Known Variance. URL http://pages.stern.nyu.edu/~churvich/MBA/Handouts/11-CI(2).pdf, 2009-06-26 [cit. 2014-03-10].

[15] Kang, S.; Leblebici, Y.: *Cmos Digital Integrated Circuits,3/E*. Tata McGraw-Hill, 2003, ISBN 9780070530775, 655 s.

[16] Katz, R. H.; Borriello, G.: *Contemporary Logic Design*. Upper Saddle River, New Jersey, USA: Prentice Hall, druhé vydání, 2005, ISBN 978-0-2013-0857-0, 608 s.

[17] Keutzer, K.: DAGON: Technology Binding and Local Optimization by DAG Matching. In *Proceedings of the 24th ACM/IEEE Design Automation Conference*, DAC '87, New York, NY, USA: ACM, 1987, ISBN 0-8186-0781-5, s. 341–347.

[18] Keymeulen, D.; Stoica, A.; Zebulum, R.; aj.: Self-Reconfigurable Mixed-Signal Integrated Circuits Architecture Comprising a Field Programmable Analog Array and a General Purpose Genetic Algorithm IP Core. In *Evolvable Systems: From Biology to Hardware*, *Lecture Notes in Computer Science*, ročník 5216, editace G. Hornby; L. Sekanina; P. Haddow, Springer Berlin Heidelberg, 2008, ISBN 978-3-540-85856-0, s. 225–236.

[19] Koza, J.; aj.: *Genetic Programming III: Darwinian Invention and Problem Solving*. San Francisco CA: Morgan Kaufmann Publishers, 1999.

[20] Koza, J. R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992, ISBN 0-262-11170-5.

[21] Langeheine, J.; Becker, J.; Folling, S.; aj.: A CMOS FPTA chip for intrinsic hardware evolution of analog electronic circuits. In *Evolvable Hardware, 2001. Proceedings. The Third NASA/DoD Workshop on*, 2001, s. 172–175.

[22] Langeheine, J.; Becker, J.; Folling, S.; aj.: A CMOS FPTA chip for intrinsic hardware evolution of analog electronic circuits. In *Evolvable Hardware, 2001. Proceedings. The Third NASA/DoD Workshop on*, 2001, s. 172–175, doi:10.1109/EH.2001.937959.

[23] Langeheine*, J.; Trefzer, M.; Brüderle, D.; aj.: On the evolution of analog electronic circuits using building blocks on a CMOS FPTA. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004), Part I, LNCS 3102*, ročník 1, editace K. Deb; aj., Seattle, WA, USA: Springer- Verlag, June 2004, ISBN ISBN 3-540-22344-4, s. 1316–1327.

[24] Langeheine, J.; Trefzer, M.; Schemmel, J.; aj.: Intrinsic Evolution of Digital-To-Analog Converters Using a CMOS FPTA Chip. In *Proc. of the 2004 NASA/DoD Conference on Evolvable Hardware*, Seattle, WA, USA: IEEE Press, Červen 2004, s. 18–25, iSBN: 0-7695-2145-2.

[25] Mead, C.; Conway, L.: *Introduction to VLSI systems*. Addison-Wesley series in computer science, Addison-Wesley, 1980, ISBN 9780201043587.

[26] Mermoud, G.; Upegui, A.; Peña-Reyes, C. A.; aj.: A Dynamically-Reconfigurable FPGA Platform for Evolving Fuzzy Systems. In *Computational Intelligence and Bioinspired Systems, 8th International Work-Conference on Artificial Neural Networks IWANN 2005*, Springer, 2005.

[27] Miller, J.; Thomson, P.: Cartesian Genetic Programming. In *Proc. European Conference on Genetic Programming, LNCS, vol. 1802*, Springer, 2000, str. 121–132.

[28] Miller, J.; Thomson, P.; Fogarty, T.: Designing Electronic Circuits Using Evolutionary Algorithms: Arithmetic Circuits: A Case Study. In *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science: Recent Advancements and Industrial Applications*, editace D. Quagliarella; J. Periaux; C. Poloni; G. Winter, Wiley, 1998, str. 105–131.

[29] Miller, J. F. (editor): *Cartesian genetic programming*. Natural Computing Series, Berlin: Springer, 22 vydání, 2011, ISBN 978-3-642-17310-3, 344 s.

[30] Petrlík, J.; Sekanina, L.: Multiobjective evolution of approximate multiple constant multipliers. In *IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems 2013*, IEEE Computer Society, 2013, ISBN 978-1-4673-6133-0, s. 116–119.

[31] Rabaey, J. M.; Chandrakasan, A.; Nikolic, B.: *Digital integrated circuits- A design perspective*. Prentice Hall, druhé vydání, 2004, 513 s.

[32] Sekanina, L.: Virtual Reconfigurable Circuits for Real-World Applications of Evolvable Hardware. In *Evolvable Systems: From Biology to Hardware*, *Lecture Notes in Computer Science*, ročník 2606, editace A. Tyrrell; P. Haddow; J. Torresen, Springer Berlin Heidelberg, 2003, ISBN 978-3-540-00730-2, s. 186–197, doi:10.1007/3-540-36553-2˙17.

[33] Sekanina, L.: Evolvable Hardware. In *Handbook of Natural Computing*, editace G. Rozenberg; T. Bäck; J. Kok, Springer Berlin Heidelberg, 2012, ISBN 978-3-540-92909-3, s. 1657–1705.

[34] Sekanina, L.; Vašíček, Z.: Approximate Circuits by Means of Evolvable Hardware. In *2013 IEEE International Conference on Evolvable Systems (ICES)*, Proceedings of the 2013 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE Computer Society, 2013, ISBN 978-1-4673-5847-7, s. 21–28.

[35] Sekanina, L.; aj.: *Evoluční hardware. Od automatického generování patentovatelných invencí k sebemodifikujícím se strojům.* Academia Praha, 2010, ISBN 978-80-200-1729-1, 328 s.

[36] Slorach, C.; Sharman, K.: The Design and Implementation of Custom Architectures for Evolvable Hardware Using Off-the-Shelf Programmable Devices. In *Evolvable Systems: From Biology to Hardware*, *Lecture Notes in Computer Science*, ročník 1801, editace J. Miller; A. Thompson; P. Thomson; T. Fogarty, Springer Berlin Heidelberg, 2000, ISBN 978-3-540-67338-5, s. 197–207.

[37] Stoica, A.; Keymeulen, D.; Arslan, T.; aj.: Circuit Self-Recovery Experiments in Extreme Environments. In *Evolvable Hardware*, IEEE Computer Society, 2004, ISBN 0-7695-2145-2, s. 142–145.

[38] Stoica, A.; Keymeulen, D.; Duong, V.; aj.: Evolutionary recovery of electronic circuits from radiation induced faults. In *IEEE Congress on Evolutionary Computation*, IEEE, 2004, ISBN 0-7803-8515-2, s. 1786–1793.

[39] Stoica, A.; Keymeulen, D.; Zebulum, R.; aj.: Evolution of analog circuits on field programmable transistor arrays. In *Evolvable Hardware, 2000. Proceedings. The Second NASA/DoD Workshop on*, 2000, s. 99–108.

[40] Stoica, A.; Zebulum, R.; Keymeulen, D.: Progress And Challenges In Building Evolvable Devices. In *Proceedings of the The 3rd NASA/DoD Workshop on Evolvable Hardware*, EH '01, Washington, DC, USA: IEEE Computer Society, 2001, ISBN 0-7695-1180-5, str. 33.

[41] Thompson, A.: An evolved circuit, intrinsic in silicon, entwined with physics. In *Evolvable Systems: From Biology to Hardware*, *Lecture Notes in Computer Science*, ročník 1259, editace T. Higuchi; M. Iwata; W. Liu, Springer Berlin Heidelberg, 1997, ISBN 978-3-540-63173-6, s. 390–405.

[42] Thompson, A.: Hardware Evolution: Automatic design of electronic circuits in reconfigurable hardware by artificial evolution. *Distinguished dissertation series*, Springer-Verlag, 1998, ISSN 3-540-76253-1.

[43] Upegui, A.; Sanchez, E.: Evolving Hardware by Dynamically Reconfiguring Xilinx FPGAs. In *Evolvable Systems: From Biology to Hardware*, *Lecture Notes in Computer Science*, ročník 3637, editace J. Moreno; J. Madrenas; J. Cosp, Springer Berlin Heidelberg, 2005, ISBN 978-3-540-28736-0, s. 56–65.

[44] Vašíček, Z.; Sekanina, L.: Evaluation of a New Platform For Image Filter Evolution. In *Proc. of the 2007 NASA/ESA Conference on Adaptive Hardware and Systems*, IEEE Computer Society, 2007, ISBN 076952866X, s. 577–584.

[45] Vašíček, Z.; Sekanina, L.: An Evolvable Hardware System in Xilinx Virtex II Pro FPGA. *International Journal of Innovative Computing and Applications*, ročník 1, č. 1, Inderscience Publishers, 2007: s. 63–73, ISSN 1751-648X.

[46] Vašíček, Z.; Sekanina, L.: Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware. *Genetic Programming and Evolvable Machines*, ročník 12, č. 3, Kluwer Academic Publishers, 2011: s. 305–327, ISSN 1389-2576.

[47] Walker, J.; Hilder, J.; Reid, D.; aj.: The evolution of standard cell libraries for future technology nodes. *Genetic Programming and Evolvable Machines*, ročník 12, č. 3, Springer US, 2011: s. 235–256, ISSN 1389-2576.

[48] Walker, J.; Hilder, J.; Tyrrell, A.: Evolving Variability-Tolerant CMOS Designs. In *Evolvable Systems: From Biology to Hardware*, *Lecture Notes in Computer Science*, ročník 5216, editace G. Hornby; L. Sekanina; P. Haddow, Springer Berlin Heidelberg, 2008, ISBN 978-3-540-85856-0, s. 308–319.

[49] Walker, J. A.; Miller, J. F.: Improving the Evolvability of Digital Multipliers Using Embedded Cartesian Genetic Programming and Product Reduction. In *Evolvable Systems: From Biology to Hardware, 6th International Conference, ICES 2005, Proceedings*, *Lecture Notes in Computer Science*, ročník 3637, editace J. M. Moreno; J. Madrenas; J. Cosp, Sitges, Spain: Springer, September 12-14 2005, ISBN 3-540-28736-1, s. 131–142.

[50] Walker, J. A.; Trefzer, M. A.; Bale, S. J.; aj.: PAnDA: A Reconfigurable Architecture that Adapts to Physical Substrate Variations. *IEEE Transactions on Computers*, ročník 62, č. 8, Los Alamitos, CA, USA: IEEE Computer Society, 2013: s. 1584–1596, ISSN 0018-9340.

[51] Walker, M.; Edwards, H.; Messom, C. H.: Success effort and other statistics for performance comparisons in genetic programming. In *IEEE Congress on Evolutionary Computation*, IEEE, 2007, s. 4631–4638.

[52] Walker, M.; Edwards, H.; Messom, C. H.: "Success effort"for performance comparisons. In *GECCO*, editace H. Lipson, ACM, 2007, ISBN 978-1-59593-697-4, str. 1760.

[53] Weste, N. H.; Harris, D.: *CMOS VLSI design: a circuits and systemes perspective*. Boston, USA: Addison-Wesley, třetí vydání, 2005, ISBN 0-321-14901-7, 968 s.

[54] Xilinx: CoolRunner-II CPLD Family DS090 (v3.1). 2008-08-11.

[55] Xilinx: Extended Spartan-3A Family Overview DS706 (v1.1). 2011-02-02.

[56] Xilinx: AXI Reference Guide UG761 (v13.1). 2011-03-27.

[57] Xilinx: Zynq-7000 All Programmable SoC technical reference manual UG585 (v1.4). 2012-11-16.

[58] Xilinx: Zynq-7000 All Programmable SoC: Concepts, Tools, and Techniques (CTT) A Hands-On Guide to Effective Embedded System Design UG873 (v14.4). 2012-18-12.

[59] Xilinx: CPLD devices [online]. 2012 [cit. 2013-08-01]. URL http://www.xilinx.com/cpld/

[60] Xilinx: Zynq-7000 All Programmable SoC. 2013. URL http://www.xilinx.com/publications/prod_mktg/zynq-7000-generation-ahead-backgrounder.pdf

[61] Xilinx: AR #56609: 2013.2 Vivado IPI, Zynq-7000 – How do I connect custom AXI HDL outside of IPI to a Zynq AXI interface? [online]. URL http://www.xilinx.com/support/answers/56609.htm, 2013-02-12 [cit. 2014-02-10].

[62] Xilinx: 7 Series FPGAs Overview DS180 (v1.14). 2013-06-29.

Příloha A

Obsah CD

Přiložené CD obsahuje veškeré zdrojové kódy pro SW i HW v následující struktuře:

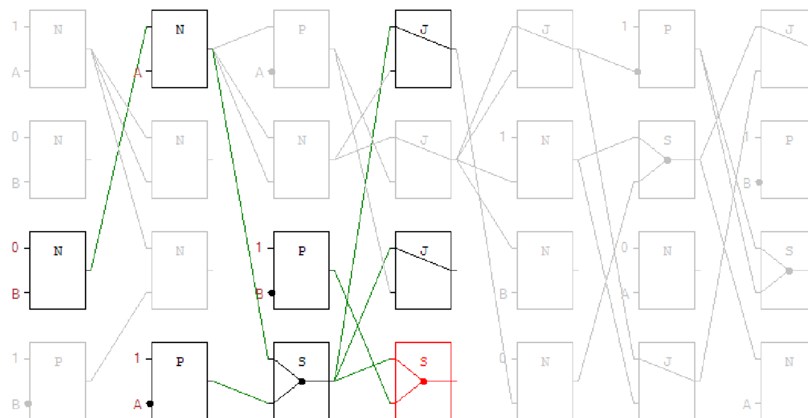
- `/bin` — syntetizované konfigurace pro programovatelnou logiku
- `/doc` — text práce a zdrojové kódy pro program LaTeX
- `/src` — zdrojové soubory:
  - `vivado` — projekt Vivado pro konfiguraci čipu Zynq včetně kódu pro procesor
  - `cpu` — referenční SW implementace
- `/tools` — pomocné softwarové nástroje

Příloha B

Příklady nalezených řešení

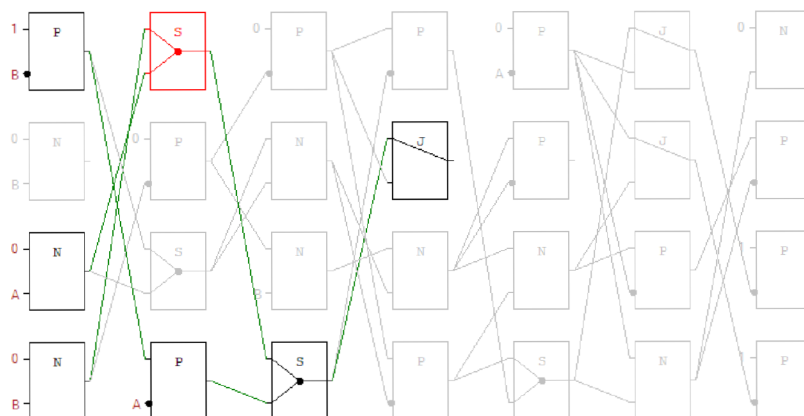
Tato příloha znázorňuje příklady nalezených řešení dvouvstupých hradel. Pro každé z řešení zobrazuje kompletní konfiguraci mřížky VRC znázorněnou pomocí vizualizéru. Mimo to je pro každý případ uvedena i konfigurace chromozomu ve formátu VHDL řetězců s vestavěnou pětibitovou adresou. Tento formát je možné použít ve vizualizéru pro vytvoření netlistu *ngspice*, nebo pro testbench VHDL souborů v nástroji Vivado.

`X"0674c309", X"0a278482", X"11649151", X"1bf2f7de", X"229edb7e", X"2ac2403a", X"377b042f", X"38ad22cf", X"4000000f"`



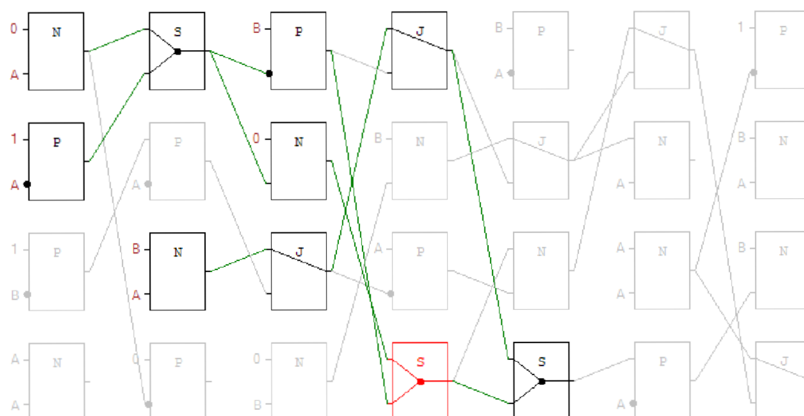
Obrázek B.1: Příklad konfigurace hradla NAND

X"0730831d", X"0a968f16", X"1074d9a1", X"1f73be5e", X"276506c2", X"29cf4e7c", X"360f19bd", X"3b4b0bf1", X"40000004"



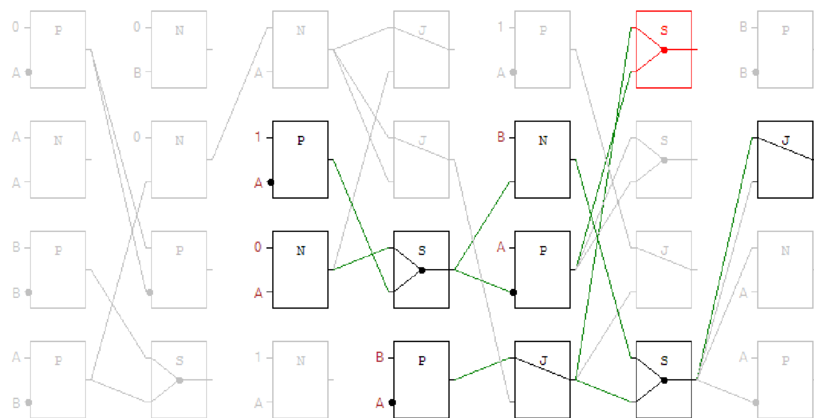
Obrázek B.2: Příklad konfigurace hradla NOR

X"0429d648", X"0c026ad5", X"13b8818d", X"191df30c", X"26553a57", X"2bdd786f", X"3455c485", X"3f31d89b", X"4000000f"



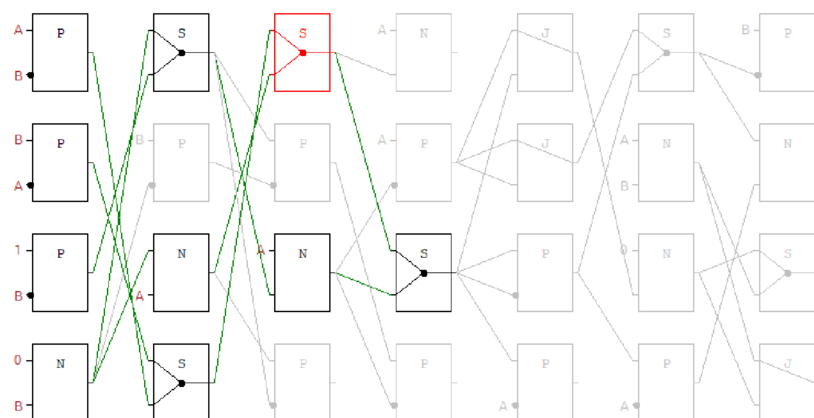
Obrázek B.3: Příklad konfigurace hradla AND

X"0079f298", X"0fcc8703", X"14414456", X"1f727a08", X"2335153a", X"2d6fdee4", X"36ef7f2d", X"3bd0bffa", X"40000014"



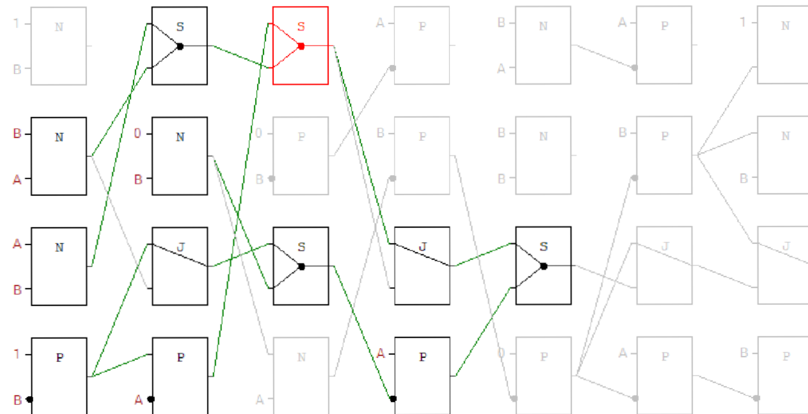
Obrázek B.4: Příklad konfigurace hradla OR

X"0731d6de", X"0ca2ef76", X"1089b2de", X"1a391133", X"26df575b", X"2d6aaced", X"30d58806", X"3fad71e3", X"40000008"



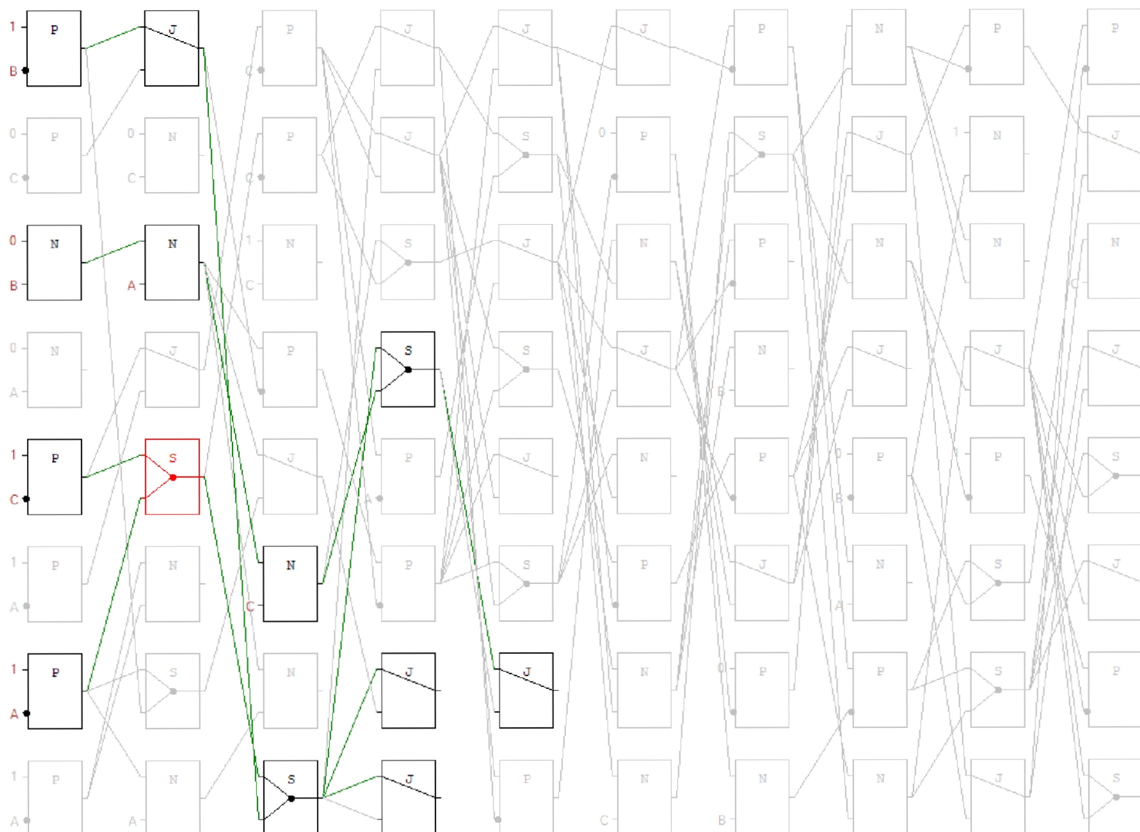
Obrázek B.5: Příklad konfigurace hradla XOR

X"0674e2cd", X"0afde315", X"16b9629d", X"1a3db50a", X"21b1372e", X"2ec4d17c", X"325ebdde", X"3bdfa8e9", X"40000008"



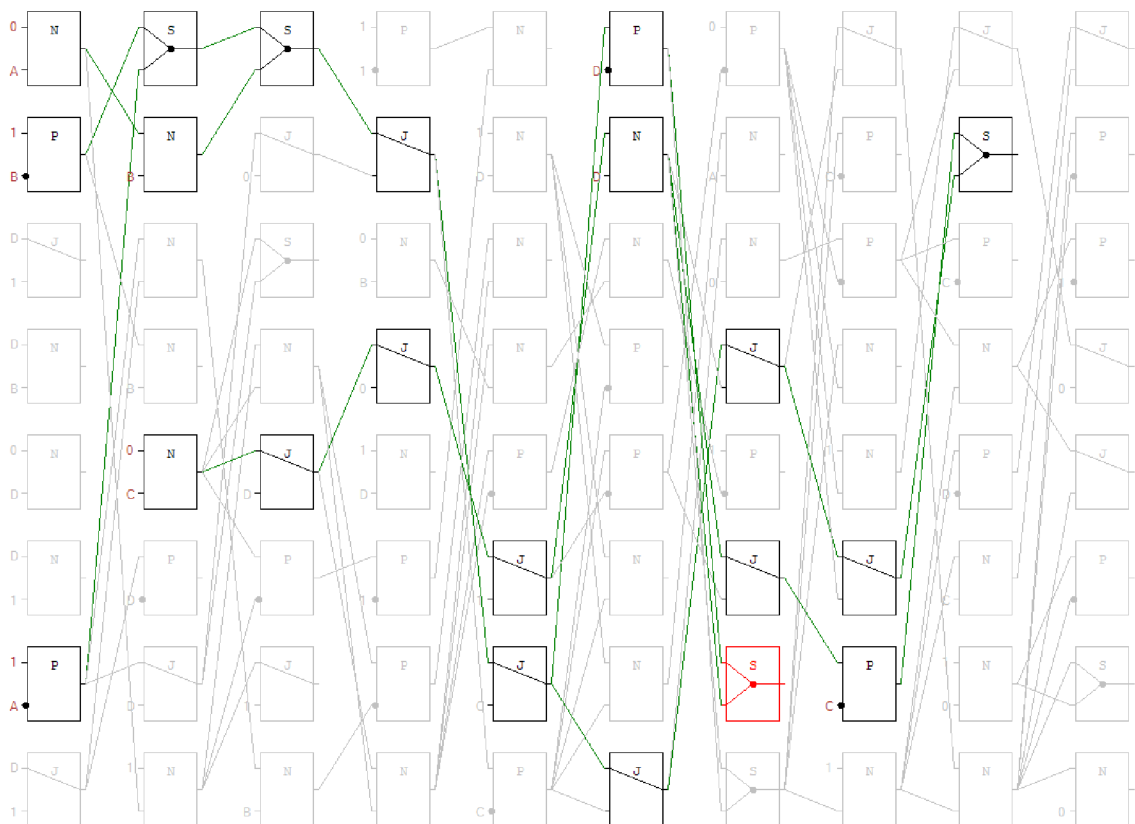
Obrázek B.6: Příklad konfigurace hradla XNOR

X"00186059", X"0a2a2c22", X"140d9545", X"1cf52138", X"265b332b", X"2a4a9015", X"35d57105", X"3a6388fd", X"4557c696", X"4cb564d5", X"54e71584", X"5d2bd379", X"66aa9ba7", X"6b711356", X"71a0d5d5", X"7bd74e36", X"81857560", X"8f5e2b02", X"96283b62", X"9ecb06ee", X"a1ea86b0", X"aa61d48d", X"b6971425", X"ba922ab0", X"c571df05", X"ccb5974d", X"d2fa5723", X"df45f962", X"e22189e2", X"e8000019"



Obrázek B.7: Příklad konfigurace hradla NAND–3

```
X"07ce5910", X"0a21c603", X"136b1f3d", X"1806e6e0", X"275c5744", X"2864ec30",
X"368adb2b", X"398f3b5a", X"45d9110e", X"4a385061", X"527a51b1", X"58a25b13",
X"66f23cdc", X"6be8f8cb", X"705b5714", X"7df3ae4d", X"830f73dd", X"8e16341f",
X"96077d03", X"9cf3b74e", X"a053b9e9", X"a8c2372d", X"b6152f69", X"bc579c51",
X"c15e3e14", X"cf300227", X"d65eeee8", X"d9ece0c8", X"e03ecd7b", X"ee65d26c"
```



Obrázek B.8: Příklad konfigurace hradla ANDNOR–4 v neoptimální verzi s povolením degradovaného výstupu