VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

NÁVRH LOGARITMICKÝCH NÁSOBIČEK

DESIGN OF LOGARITHMIC MULTIPLIERS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ALENA DRLÍČKOVÁ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. VOJTĚCH MRÁZEK, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

142678

Ústav:	Ústav počítačových systémů (UPSY)
Studentka:	**Drlíčková Alena**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Návrh logaritmických násobiček**
Kategorie:	Návrh číslicových systémů
Akademický rok:	2022/23

Zadání:

1. Seznamte se s přibližným počítáním, možnostmi konstrukce aritmetických obvodů a přibližných logaritmických násobiček.
2. Zpracujte studii na výše uvedená témata.
3. Vybrané logaritmické násobičky z literatury implementujte a porovnejte s dalšími přibližnými násobičkami.
4. Identifikujte možnosti dalšího zlepšení poměru příkonu a chybovosti logaritmických násobiček a navrhněte algoritmus pro systematické zlepšení tohoto poměru.
5. Navržený algoritmus implementujte.
6. Vyhodnoťte parametry navržených obvodů a diskutujte možnosti pokračování projektu.

Literatura:

- Dle pokynů vedoucího.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Mrázek Vojtěch, Ing., Ph.D.**
Vedoucí ústavu:	Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání:	1.11.2022
Termín pro odevzdání:	10.5.2023
Datum schválení:	31.10.2022

Abstrakt

Tato práce se zabývá možnostmi vylepšení logaritmických násobiček pomocí aproximačních metod. Cílem bylo naimplementovat logaritmické násobičky podle konstrukcí popsaných v literatuře a identifikovat možnosti jejich modifikací. V rámci této práce je popsán způsob, jakým proběhla implementace obvodů násobiček a jejich částí. Jsou zde navrženy způsoby vylepšení těchto obvodů založené na výměně jejich komponent a celkové modifikaci pomocí evoluční metody. Parametry vytvořených logaritmických násobiček jsou porovnány s hodnotami dostupných aproximačních násobiček.

Abstract

This thesis deals with the possibilities of improving logarithmic multipliers using approximation methods. The goal was to implement logarithmic multipliers according to the constructions described in the literature and to identify the possibilities of their modifications. This work describes the way in which the implementation of multiplier circuits and their parts took place. Ways to improve these circuits based on the replacement of their components and overall modification using evolutionary methods are proposed here. The parameters of the created logarithmic multipliers are compared with the values of the available approximation multipliers.

Klíčová slova

logaritmická násobička, aproximační obvody, kartézské genetické programování, přibližné počítání

Keywords

logarithmic multiplier, approximation circuits, Cartesian genetic programming, approximate computation

Citace

DRLÍČKOVÁ, Alena. *Návrh logaritmických násobiček*. Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vojtěch Mrázek, Ph.D.

Návrh logaritmických násobiček

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením pana Ing. Vojtěcha Mrázka, Ph.D. Uvedla jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpala.

.....
Alena Drlíčková
9. května 2023

Poděkování

Ráda bych poděkovala panu Ing. Vojtěchu Mrázkovi, Ph.D. za vstřícnost, trpělivost, lidský přístup, odbornou pomoc a cenné rady, které mi poskytl při vedení mé práce. Dále chci poděkovat své skvělé rodině za podporu a laskavost. Poděkování patří také panu příteli Ing. Miloši Minaříkovi, Ph.D. za oporu, kterou jsem v něm měla po celou dobu studia.

Obsah

1 Úvod **3**

2 Aproximační aritmetické obvody **5**
2.1 Parametry aritmetických obvodů 5
2.1.1 Fyzické parametry 5
2.1.2 Chybové metriky 6
2.2 Vztah mezi parametry 7
2.3 Konstrukce aproximačních obvodů 8
2.3.1 Výchozí přesné obvody 8
2.3.2 Ruční návrh aproximačních obvodů 8
2.3.3 Automatický návrh aproximačních obvodů 10
2.4 Dostupné obvody 11

3 Logaritmické násobičky **12**
3.1 Princip logaritmického násobení 12
3.1.1 Nalezení přibližného binárního logaritmu 12
3.1.2 Logaritmická reprezentace operandu 13
3.1.3 Samotný výpočet 13
3.2 Přibližná logaritmická násobička 13
3.2.1 Využití aproximačních sčítaček 14
3.3 Vylepšená logaritmická násobička 15
3.3.1 Využití aproximačních sčítaček v ILM 16
3.4 Násobení nulou 16

4 Implementace logaritmických násobiček **17**
4.1 Nástroj ArithsGen 17
4.2 Subkomponenty násobiček 17
4.2.1 Detektor nejvýznamnější jedničky 18
4.2.2 Detektor nejbližší jedničky 18
4.2.3 Prioritní kodér 19
4.2.4 Dekodér 19
4.2.5 Posouvač 20
4.2.6 Odčítačka 21
4.3 Implementace násobiček 21
4.3.1 ALM 21
4.3.2 ILM 23
4.4 Testování parametrů 24
4.5 Srovnání 24

5 Návrh vylepšení logaritmických násobiček **26**
5.1 Modifikace subkomponenty 26
5.2 Evoluční modifikace celého obvodu 29
5.3 Korekce při násobení nulou 29

6 Vylepšení logaritmických násobiček **31**
6.1 Modifikace pomocí nepřesných sčítaček 31
6.1.1 Přibližné úplné sčítačky 31
6.1.2 Evolučně navržené sčítačky 32
6.2 Evoluční úprava 32
6.3 Generování a testování v Pythonu 33

7 Vyhodnocení **34**
7.1 ALM využívající evoluční sčítačky 34
7.2 ILM využívající evoluční sčítačky 35
7.3 Evoluční aproximace logaritmických násobiček 37
7.4 Pareto optimální násobičky 39

8 Závěr **40**

Literatura **41**

Kapitola 1

Úvod

V posledních letech roste zájem o energeticky úsporná zařízení a u řady aplikací začíná být kladen větší důraz na rychlost a nízký příkon. To se projevuje i při návrhu číslicových obvodů. Vznikají obvody, které za cenu snížení přesnosti výsledku dosáhnou aproximací výpočtu menší spotřeby, plochy nebo zpoždění. Tento způsob optimalizace je vhodný především pro aplikace, které jsou inherentně odolné vůči chybám, například neuronové sítě. V neuronových sítích je nejnáročnější operací násobení [20]. Použitím aproximačních násobiček dochází ke ztrátě přesnosti, ale určitá chyba je v tomto případě tolerována vlastní chybovou tolerancí neuronových sítí [2]. Další oblastí, kde lze tento přístup využít, je například zpracování signálu. Stejně jako u neuronových sítí i zde dochází často k násobení, což je časově i energeticky náročné. Aproximační násobení poskytuje i v tomto případě uspokojivé výsledky, zejména při kompresi zvuku nebo videozáznamu [4].

Přístupů k aproximaci násobení existuje celá řada. Jedním z nich je například logaritmická násobička, která realizuje součin tak, že ze vstupních operandů spočítá logaritmy, ty sečte a výsledek odlogaritmuje. Narozdíl od běžných násobiček není její struktura ve všech částech uniformní, ale skládá se z několika různých komponent. Aproximace logaritmem zavádí do výpočtu nevratnou chybu, která snižuje přesnost výsledku.

Cílem této práce je prozkoumat možnosti dalšího zvýšení efektivity logaritmických násobiček. Navrhnout nové úpravy těchto obvodů a otestovat jejich vliv na chybovost a fyzické parametry.

Druhá kapitola se zabývá aproximačním počítáním. Jsou v ní popsány přístupy tvorby aproximačních sčítaček a násobiček. Dále jsou představeny knihovny dostupných aproximačních obvodů

Ve třetí kapitole jsou podrobněji probrány logaritmické násobičky. Je zde uveden princip logaritmického násobení. Dále je popsána konstrukce původní Mitchellovy násobičky a vylepšené logaritmické násobičky.

Ve čtvrté kapitole se nachází implementace násobiček popsaných ve druhé kapitole. Je zde uvedeno jakým způsobem byly implementovány jejich komponenty s použitím nástroje ArithsGen. Poté jsou vyhodnoceny parametry vytvořených násobiček a porovnány s jinými aproximačními násobičkami.

V rámci páté kapitoly jsou navržena možná zlepšení efektivity logaritmických násobiček. Jsou zde srovnány podíly jednotlivých částí násobiček na celkové velikosti a vybrány komponenty určené k další úpravě.

Šestá kapitola se zabývá realizací návrhu vylepšení. Jsou v ní popsány modifikace původních logaritmických násobiček změnou vnitřních komponent. Dále je uveden způsob, jakým byly upraveny celé struktury obvodů násobiček.

V sedmé kapitole jsou demonstrovány dosažené výsledky. Jsou zde shrnuta dosažená vylepšení v porovnání s původními obvody logaritmických násobiček. Poté jsou parametry nových násobiček porovnány s jinými aproximačními násobičkami.

Kapitola 2

Aproximační aritmetické obvody

Tato kapitola vysvětluje princip počítání aproximačními obvody a způsob jakým jsou tyto obvody implementovány. Dále je zde představena knihovna EvoApproxLib obsahující evolučně generované obvody.

S rostoucím využitím počítačových technologií v různých oblastech, rostou i nároky na výpočetní zdroje. Výroba se tomuto trendu přizpůsobuje a počítače se zmenšují, zatímco jejich výpočetní síla se zvyšuje. Jak tvrdí Moorův zákon, počet tranzistorů na čipu se zdvojnásobí zhruba každé dva roky. Avšak s vývojem technologií výroby čipů se blíží doba, kdy přestane Moorův zákon platit [22]. Vzhledem k fyzikálním vlastnostem tranzistorů nebude časem možné obvody neustále zmenšovat. Se zvětšující se poptávkou po výpočetních zdrojích bude nutné je využívat efektivněji. Jedním ze způsobů zefektivnění výpočtů je využití nepřesnosti v aritmetických operacích. V některých aplikacích je kladen větší důraz na příkon a plochu obvodu nebo rychlost výpočtu než na přesnost výsledku. Zvláště při interakci zařízení s lidskými smysly, které jsou omezené v rozpoznávání drobných rozdílů, je použití nepřesného počítání výhodné.

2.1 Parametry aritmetických obvodů

Pro návrh nových obvodů je stěžejní vyhodnocování jejich parametrů. U přesných se zkoumají hlavně fyzické vlastnosti jako je příkon, plocha a zpoždění. Nepřesné obvody jsou navíc charakterizovány i jejich chybovými metrikami.

2.1.1 Fyzické parametry

Fyzické parametry se určují stejně jak u aproximačních, tak i přesných obvodů. Jejich hodnoty se získávají pomocí syntézy. Přesnost těchto parametrů závisí na tom, v jakém kroku implementace byly získány.

Příkon

Příkon označuje množství energie spotřebované za jednotku času. Velikost celkového příkonu obvodů ovlivňuje zejména počet prvků obvodu. Skládá se ze dvou složek, statického a dynamického příkonu. Statický příkon tvoří proud protékající obvodem mimo jeho aktivitu. Velikost statického příkonu je ovlivněna především plochou. Dynamický příkon je závislý na přepínací aktivitě a zkratovém příkonu. Jeho hodnotu je složitější určit, protože přepínací aktivita vychází z použité aplikace. Často se používá uniformní rozdělení četnosti spínání vstupů.

Plocha
Plocha je stejně jako příkon závislá na celkovém počtu a velikosti prvků obvodu, zejména ale na technologické knihovně.
Zpoždění
Zpoždění kombinačního obvodu je parametr, který udává kolik času uplyne od okamžiku, kdy se na vstupu daného obvodu změní vstupní hodnota, do okamžiku, kdy dojde na výstupu obvodu k nastavení odpovídající výstupní hodnoty. Celkové zpoždění je dáno součtem zpoždění jednotlivých hradel, která tvoří kritickou cestu. Kritickou cestou v obvodu se rozumí propojení od vstupu k výstupu s největším počtem hradel zapojených za sebou. V sekvenčních obvodech se jedná o cestu mezi dvěma vstupy registrů, či registrem a vstupem nebo výstupem.

2.1.2 Chybové metriky

Přesnost, s jakou obvody počítají lze určit díky simulacím, kdy jsou na vstup obvodu přivedeny hodnoty, pro které je znám výstup přesných obvodů. Porovnáním takto získaného výstupu a očekávaného přesného výstupu lze spočítat hodnoty chybových metrik [17].
Pravděpodobnost chyby (Error probability, zkr. EP) udává podíl případů, kde nebyl výsledek spočítaný aproximačním obvodem přesný.

$$EP = \frac{\sum_{\forall i} O^{(i)}_{approx} \neq O^{(i)}_{orig}}{2^{n_i}} \tag{2.1}$$

Průměrná aritmetická chyba (Mean absolute error, zkr. MAE) je jednou z nejpoužívanějších chybových metrik. Vyjadřuje průměrný rozdíl mezi aproximovanými a přesnými výsledky.

$$MAE = \frac{\sum_{\forall i} \mid O^{(i)}_{approx} - O^{(i)}_{orig} \mid}{2^{n_i}} \tag{2.2}$$

Průměrný kvadrát chyby (Mean squared error, zkr. MSE) počítá průměr z druhých mocnin rozdílů mezi přesnými a očekávanými výsledky. Oproti MAE, metrika MSE zvýrazňuje větší chyby.

$$MSE = \frac{\sum_{\forall i} \mid O^{(i)}_{approx} - O^{(i)}_{orig} \mid^2}{2^{n_i}} \tag{2.3}$$

Průměrná relativní chyba (Mean relative error, zkr. MRE) zahrnuje do výpočtu i velikost očekávané přesné hodnoty. Větší absolutní chyby u výsledků s velkou očekávanou hodnotou tolik neovlivňují MRE jako by tomu bylo u MAE. V případě nepřesnosti u násobení nulou, by se ve výpočtu MRE mělo nulou dělit. Zde se používají dva přístupy - buď se tyto případy vynechají, nebo se dělí 1 (jak je v níže uvedeném vzorci).

$$MRE = \frac{\sum_{\forall i} \frac{|O^{(i)}_{approx} - O^{(i)}_{orig}|}{max(1, O^{(i)}_{orig})}}{2^{n_i}} \tag{2.4}$$

Maximální absolutní chyba (Worst-case error, zkr. WCE) je největší hodnota o jakou se aproximovaný výsledek lišil od přesného výsledku. Pomocí tohoto parametru lze vybírat obvody jejichž chyba není větší než stanovená hodnota.

$$WCE = max_{\forall i} \mid O^{(i)}_{approx} - O^{(i)}_{orig} \mid \tag{2.5}$$

Hammingova vzdálenost (Hamming distance, zkr. HD) uvádí, na kolika pozicích se v binární reprezentaci přibližný výsledek liší od přesného výsledku. Tato metrika je pro aritmetické obvody méně vhodná. Například pokud je vypočítaný přibližný výsledek $128_{10} = 10000000_2$ a očekávaný přesný výsledek je 127_{10} (01111111_2), jejich Hammingova vzdálenost je 8, zatímco relativní chyba je 0.78%.

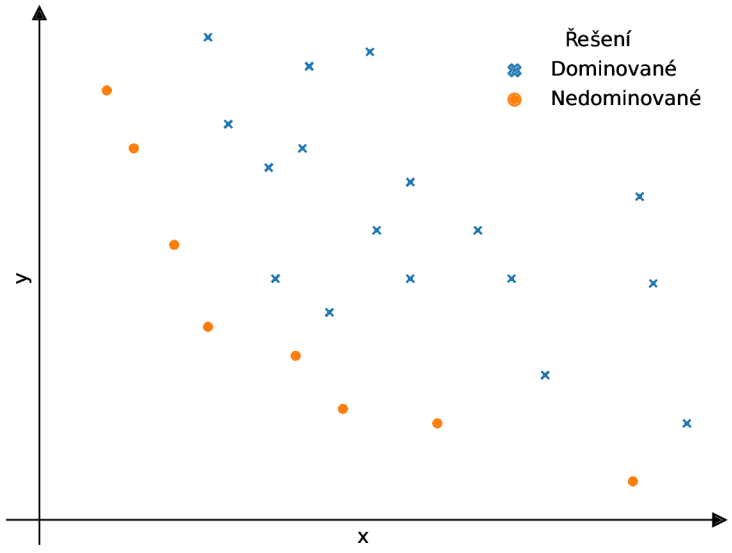
$$HD = \sum_{\forall i} C(O^{(i)}_{approx} \oplus O^{(i)}_{orig}) \tag{2.6}$$

Ve všech výše uvedených vzorcích je n celkový počet konfigurací, se kterými probíhá testování kvality obvodu. Aproximovaný výsledek i-té konfigurace je označen $O^{(i)}_{approx}$ a přesný výsledek pro stejnou konfiguraci $O^{(i)}_{orig}$. Hodnota C u výpočtu HD udává počet jedničkových bitů v binární reprezentaci čísla.

Při návrhu aproximačního aritmetického obvodu je použití chybových metrik důležité pro správné vyhodnocení jeho kvality. Výběr použité chybové metriky závisí na typu obvodu. Například u obvodu sčítačky je více vypovídající průměrná aritmetická chyba než Hammingova vzdálenost.

2.2 Vztah mezi parametry

Při konstrukci aproximačních obvodů je snaha snižovat plochu, příkon nebo zpoždění za cenu minimálního zvýšení chybovosti. V grafu 2.1 jsou znázorněny parametry obvodů. Osa x zde představuje například velikost chybové metriky a osa y příkon zkoumaných obvodů. Všichni jedinci, kteří mají všechny zkoumané vlastnosti horší než jiný jedinec, jsou tímto jedincem dominováni. Řešení, pro které neexistuje jiné řešení s oběma lepšími vlastnostmi, se nazývá dominující. Soubor všech dominujících řešení tvoří Pareto frontu.



Obrázek 2.1: Ukázka Pareto fronty

2.3 Konstrukce aproximačních obvodů

Aproximační návrh spočívá v úpravě funkce realizované přesným obvodem na funkci s podobnými vlastnostmi, kterou počítá jednodušší obvod. Aritmetické aproximační obvody lze vytvářet dvěma způsoby: ručně a automatizovaně. Ruční návrh provádí návrhář specifickým způsobem pro konkrétní obvod, zatímco při automatickém návrhu jsou počítačově generovány aproximace obvodů univerzálními technikami.

2.3.1 Výchozí přesné obvody

Sčítačky
Sčítačka provádí součet dvou binárních čísel. Dva základní typy sčítaček jsou: sčítačka s přenosem a sčítačka s předpovědí přenosu [9].
Sčítačka s přenosem (Ripple carry adder, zkr. RCA) o délce n bitů se skládá z n - 1 úplných sčítaček a jedné poloviční sčítačky. Každá sčítačka realizuje operaci součtu mezi dvěma příslušnými bity vstupních sběrnic a případně přenosu C_{in} z nižšího řádu. Výsledek Sum předává na výstup a přenos C_{out} předává další úplné sčítačce. Nevýhodou RCA je velké zpoždění způsobené předáváním přenosu z nižšího do vyššího řádu.
Sčítačka s předpovědí přenosu (Carry lookahead adder, zkr. CLA) je složena z bloků paralelně počítajících součty bitů a carry lookahead jednotky. Ta určuje, který pár bitů generuje nebo propaguje přenos do vyššího řádu. Díky tomu lze předem určit, kde nastane přenos, a provádět tak součet vstupních sběrnic bez vzniku zpoždění způsobeného čekáním na přenos z předchozích sčítaček. CLA pracuje výrazně rychleji než RCA za cenu větší plochy a vyššího příkonu.

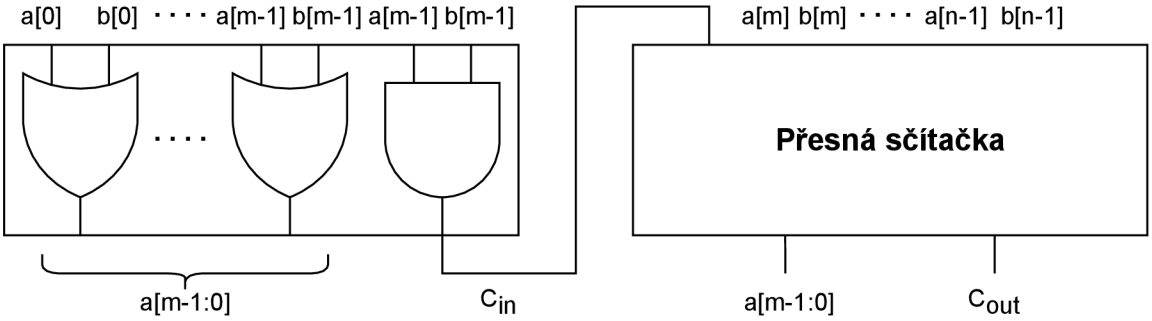
Kombinováním vlastností těchto sčítaček lze vytvářet sčítačky s různým poměrem zpoždění vůči ploše a příkonu. Zpoždění zde závisí na délce kritické cesty, která označuje nejdelší cestu obvodu. U RCA je délka kritické cesty ovlivněna předáváním přenosů mezi úplnými sčítačkami.

2.3.2 Ruční návrh aproximačních obvodů

Sčítačky
Pro dva základní typy sčítaček popsané v části 2.3.1 bylo navrženo několik ručních aproximačních metod, které modifikují jejich strukturu [9]. Zde jsou popsány některé z nich.

- **Spekulativní sčítačky** jsou založeny na predikci přenosu pro každý bit součtu. Příkladem aproximační spekulativní sčítačky je *téměř přesná sčítačka* (almost correct adder) [25], která určuje přenos pomocí méně významných bitů operandů.
- **Segmentované sčítačky** jsou tvořeny kratšími nezávislými úseky přesných sčítaček, mezi kterými nedochází k přenosu. Kritická cesta je tímto způsobem rozdělena na více kratších částí počítajících paralelně. Jako přenos z předchozích bloků sčítaček se v některých typech používá konstantní hodnota a v některých typech sčítaček se přenos predikuje. To ovlivňuje výslednou velikost a chybovost [9].
- **Přibližné úplné sčítačky** o délce n bitů se skládají z aproximační k-bitové části a z přesné sčítačky s délkou $n-k$ bitů. Významnější bity jsou přivedeny do přesné části a méně významné bity do aproximační části sčítačky. Aproximační část neprovádí standardní součet ale některou z funkcí aproximujících součet. Protože v aproximační

části se nepřenáší *propagate* signál, kritická cesta je zkrácena o délku nepřesného bloku sčítačky. Implementováno je několik druhů těchto sčítaček, které se liší provedením nepřesné části. Na obrázku 2.2 je znázorněna struktura sčítačky používající pouze OR hradla namísto celých sčítaček pro výpočet součtu nejméně významných bitů [15]. Jiné typy sčítaček na výstup připojují pouze jeden ze vstupů, nebo konstantní hodnotu. Použití takových sčítaček může být velmi efektivní v některých typech násobiček, jak je popsáno v části 3.2.1.



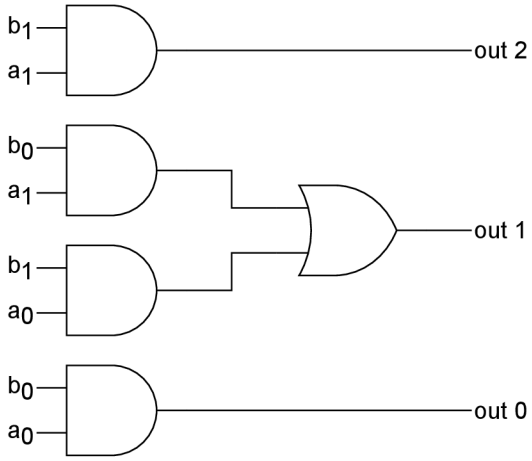
Obrázek 2.2: Struktura přibližné sčítačky Lower part or adder [14]

Tato práce se zabývá především využitím přibližných úplných sčítaček, které jsou použity v aproximačních logaritmických násobičkách.

Násobičky
N-bitové násobičky provádí součin dvou n-bitových operandů součtem částečných součinů mezi dvojicemi vstupních bitů operandů. Částečný součin dvou bitů produkuje hradlo AND a k součtům slouží úplné N-bitové sčítačky. Násobičky jsou poměrně složité aritmetické obvody s vysokou spotřebou a velkým zpožděním způsobeným délkou kritické cesty. Tento problém řeší využití aproximačních násobiček [5]. Příkladem ručně navržených obvodů jsou následující typy.

- **Aproximační násobičky odolné vůči chybě** jsou násobičky používající při násobení dvou dvoubitových operandů zjednodušený obvod násobičky s tříbitovým výstupem namísto čtyřbitového, jak by tomu bylo u klasické násobičky pro dvoubitové vstupy. Násobička zavádí chybu do výsledku pouze v případě součinu 3×3, kdy výsledek na výstupu není 9 ale 7. Z obrázku 2.3 je patrné, jak pomocí pěti hradel je realizován součin dvou 2bitových čísel. Pro vytvoření vícebitových násobiček lze použít více bloků dvoubitových násobiček a výsledky sečíst pomocí sčítaček [13].
- **Násobičky využívající oříznutí** redukují velikost použitého hardwaru vynecháním některých prvků. Například násobičky s pevnou šířkou počítají pouze n nejvýznamnějších bitů 2n-bitového součinu dvou n-bitových čísel a používají další korekční obvody ke snížení chyb způsobených zkrácením výpočtu [10].
- **Aproximační logaritmické násobičky** jsou podrobně popsány v kapitole 3

Manuální návrh obvodů poskytuje pouze omezené množství různých variant obvodů a navíc spotřebuje velké množství času návrháře. Z těchto důvodů návrháře mnohdy nahrazují rychlejší a efektivnější počítače, provádějící automatický návrh.



Obrázek 2.3: Struktura aproximační 2bitové násobičky [13].

2.3.3 Automatický návrh aproximačních obvodů

Velkou výhodou automatického návrhu obvodů je rychlost, s jakou lze generovat nové obvody. Při automatickém návrhu roste důležitost testování kvality. Protože vzniká velké množství různých aproximačních obvodů, objevují se i obvody s nevhodnými parametry, vykazující například velkou chybu a plochu návrhu. Pro automatický návrh existuje řada metod, například SALSA [24] SASIMI [8] nebo ABACUS [19]. Tato práce se však zabývá především automatickým evolučním návrhem obvodů.

Kartézské genetické programování Kartézské genetické programování (CGP) je metoda využívající evoluční algoritmy k návrhu výpočetních struktur. V CGP je program reprezentován jako acyklický orientovaný graf, jehož uzly jsou součástí pole o velikosti $n_c \times n_r$.

Každý uzel obsahuje předem pevně daný počet genů $n_n + 1$ reprezentovaných celými čísly, kde n_n je počet jeho vstupů. Geny uzlu nesou informaci o zdroji vstupních dat a operaci již uzel implementuje. Jako vstupy uzlů mohou být použity výstupy jiných uzlů nebo primární vstupy struktury.

Konkrétní řešení je zakódováno v chromozomu. Všechny chromozomy mají pevnou délku $n_c \times n_r \times (n_n+1)+n_o$ celých čísel, kde n_o je počet výstupů řešení. Přestože délka chromozomu je konstantní, výsledný obvod může mít různou velikost. Uzly, u kterých není výstup nikam připojen, neovlivňují chování zakódované struktury a v reálném obvodu jsou tak vynechány. Výsledný obvod se nazývá fenotyp.

Hlavní operací evolučního návrhu v CGP je mutace. Mutace probíhá tak, že vybere náhodně gen a změní jeho hodnotu. Uzly tak mohou měnit funkci nebo propojení s ostatními. Mutací se mohou měnit i primární výstupy. V CGP se využívá evoluční strategie označené jako $ES(1+\lambda)$, kdy ze staré populace je do nové vybrán nejlepší jedinec a jeho λ mutantů. Jedinec se vybírá podle hodnoty fitness funkce, která slouží k ohodnocení kvality řešení. Pro různé typy obvodů je třeba stanovit specifickou formu fitness funkce.
Průběh evoluce [21]

1. Vytvoření počáteční populace z $1 + \lambda$ jedinců.
2. Výpočet fitness pro každého jedince populace.
3. Výběr nejlepšího jedince z populace podle jeho fitness.

4. Generování λ mutantů nejlepšího jedince.
5. Vytvoření nové generace z nejlepšího jedince a jeho mutantů.
6. Kontrola ukončující podmínky. Pokud není podmínka splněna pokračuje se krokem 2.

Jedinci počáteční populace mohou být náhodně vytvoření nebo lze do první populace vložit existující návrh, který bude evolučně zlepšován.

2.4 Dostupné obvody

Při návrhu komplexnějších řešení je možné využít již existujících aproximačních obvodů. Open-Source knihovny poskytují celou škálu nízkopříkonových realizací sčítaček a násobiček.

EvoApproxLib je knihovna implementací evolučně navržených aproximačních aritmetických obvodů. Knihovna obsahuje znaménkové a neznaménkové obvody sčítaček a násobiček. Celkem knihovna implementuje 430 8bitových sčítaček a 471 8bitových násobiček, které jsou volně dostupné k využití při návrhu aproximačního hardwaru. Široká škála navržených obvodů umožňuje výběr optimální varianty s požadovaným poměrem chybovosti a fyzických parametrů. Modely obvodů jsou dostupné ve formátech C a Verilog. V této práci jsou jako součást obvodů logaritmických násobiček využity neznaménkové sčítačky z EvoApproxLib. Další knihovnou využitelnou při návrhu je lpACLib [6]. Knihovna lpACLib obsahuje VHDL popis přesných a přibližných verzí násobiček a sčítaček s různými bitovými šířkami a odpovídající softwarové implementace v jazycích C a MatLab usnadňující ohodnocení kvality. Dále je možné použití obvodů sčítaček QuAd [7]. Tyto sčítačky se vyznačují především nízkým zpožděním.

Kapitola 3

Logaritmické násobičky

Logaritmické násobičky jsou aritmetické obvody využívající k získání součinu dvou čísel logaritmy operandů a bitový posun čísel. V této kapitole je popsáno, jak probíhá násobení využívající logaritmy operandů. Dále je popsán způsob realizace logaritmického násobení v hardwaru. Na závěr jsou představeny druhy logaritmických násobiček a jejich možné modifikace.

3.1 Princip logaritmického násobení

Konvenční násobičky využívají k provedení výpočtu velké množství aritmetických sčítaček. Na jednu stranu tak dosahují přesných výsledků, ale na druhou výpočet tímto způsobem trvá déle a spotřebuje velké množství energie.

Tento problém lze vyřešit využitím binárních logaritmů činitelů. V matematice se v některých případech používají logaritmy pro zjednodušení aritmetických operací. Násobení a dělení je možné nahradit za sčítání a odčítání logaritmů operandů díky tomu, že platí vztah:

$$\log_2(a * b) = \log_2(a) + \log_2(b) \tag{3.1}$$

Uložení tabulky obsahující logaritmy čísel v počítači by bylo značně paměťově náročné, proto se hledání logaritmů čísel zjednodušuje na určení přibližného logaritmu z konkrétního operandu.
Díky tomu je možné zredukovat problém násobení dvou čísel na operace nalezení logaritmů, sečtení a odlogaritmování, což jsou operace, které lze v hardwaru provádět poměrně rychle a s menší spotřebou než cyklické sčítání. Při tomto způsobu násobení dvojkovou mocninou je výsledek přesný. Pokud však ani jeden z operandů není binární mocninou, je třeba jej rozdělit na mocninnou část a zbytek čísla. Různé typy logaritmických násobiček, které jsou představeny v následujících kapitolách pracují se zbytky čísel jiným způsobem. Základní principy původního algoritmu, který navrhl John N. Mitchell [16], jsou popsány níže.

3.1.1 Nalezení přibližného binárního logaritmu

Pro nalezení logaritmu operandu o základu dva se využívá binární reprezentace operandu, kde první jednička na nejvyšší pozici signalizuje jeho nebližší nižší mocninu dvou. Podle Mitchellova postupu se nejprve podle velikosti sběrnice stanoví nejvyšší možný exponent dvojkové mocniny, kterou může nést, ta se uloží do čítače. Bity operandu se posouvají doleva a zároveň s tím se dekrementuje čítač. Při posunutí první jedničky na nejvyšší bit je v čítači uložen hledaný binární logaritmus.

Například při hledání logaritmu čísla 42 na osmibitové sběrnici, kde je jeho binární podoba 00101010_2, je nejvyšší možnou reprezentovanou dvojkovou mocninou 2^7. Do čítače je uložena hodnota 7. Při bitovém posunu binárního čísla 42 o dvě pozice doleva, tak že na nejvýznamnějším bitu sběrnice je jednička (10101000_2), se v čítači provede dekrementace o dva. V čítači se nyní nachází číslo 5, čímž je získána přibližná hodnota logaritmu čísla 42. V tomto případě je tak 42 aproximováno hodnotou 32.

3.1.2 Logaritmická reprezentace operandu

Jak je vidět na předchozím příkladu, nalezení logaritmu je nepřesné. Aby se zabránilo větší nepřesnosti ve výsledků, je důležité zahrnout do výpočtu i zbytek čísla. V Mitchellově algoritmu je logaritmus čísla reprezentován jako desetinné číslo, kde celá část představuje nepřesný logaritmus získaný způsobem popsaným v předchozí podkapitole a desetinnou část tvoří zbytek čísla na sběrnici za jedničkou posunutou na nejvyšší pozici. Číslo 42 z předchozího příkladu je tedy reprezentováno jako: $101,0101000_2 = 5,3125_{10}$. Tato aproximace je již přesnější, protože 42 je zde zastoupeno hodnotou cca $39,74$.

3.1.3 Samotný výpočet

Díky vlastnostem binárního logaritmu lze výpočet provést sečtením logaritmů získaných z operandů a dekódováním výsledku.Po vypočítání součtu se na pozicích sběrnice reprezentujících celou část čísla nachází logaritmus nejbližší nižší dvojkové mocniny výsledného čísla. Před zbývající desetinnou část logaritmického operandu se přidá jednička a takto získané číslo se posune o odpovídající počet bitů logaritmu výsledku.

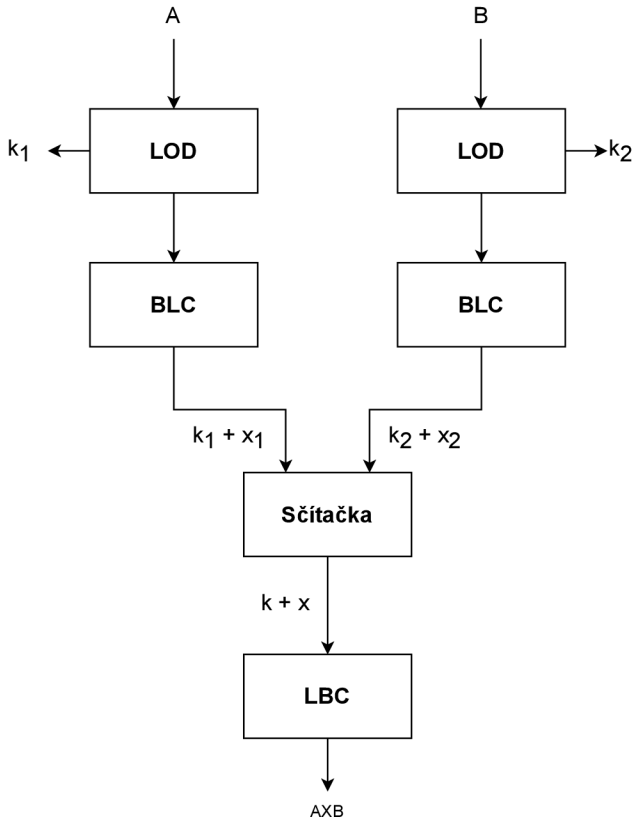
3.2 Přibližná logaritmická násobička

Přibližná logaritmická násobička (Approximate Logarithmic Multiplier, zkr. ALM) popsaná v článku [14] pracuje na základě Mitchellova algoritmu. K získání binárního logaritmu čísla využívá detektor nejvýznamnější jedničky (Leading One Detector, zkr. LOD) a konvertor z binární do logaritmické reprezentace (Binary to Logarithm Converter, zkr. BLC). Zapojení jednotlivých komponent je znázorněno v diagramu 3.1.

LOD je kombinační obvod schopný ze vstupní posloupnosti bitů detekovat pozici nejlevější, a tedy i nejvýznamnější jedničky. Délka výstupní sběrnice je stejná jako délka vstupní.

Po určení umístění nejlevější jedničky a tím pádem i nejvyšší nižší dvojkové mocniny čísla dochází k převedení operandu do logaritmického tvaru. Převod zajišťuje BLC, který se skládá z kodéru a posouvače. Díky kombinačnímu obvodu kodéru není při převodu do logaritmické podoby potřeba čítač, jak tomu je v obecném Mitchellově algoritmu popsaném v části 3.1. Kodér ze vstupní sběrnice, kde je jedničkou označena pozice nejvyšší nižší dvojkové mocniny operandu, převádí data na výstupní sběrnici, která obsahuje binární reprezentaci čísla určujícího pozici jedničky ze vstupu. Jinými slovy kodér převádí dvojkovou mocninu čísla na její logaritmus. Výstupní sběrnice kodéru má velikost $\log_2 n$, kde n je délka jeho vstupu.

Posouvač v této části zajišťuje správnou pozici desetinné části logaritmu, jak bylo pospáno v Mitchellově algoritmu. K tomuto účelu lze využít Válcový posouvač (Barrel shifter), jak popisuje článek [11]. Na výstup BLC je vyskládána celá část logaritmu získaná v LOD následovaná jeho desetinnou částí z posouvače. Na obrázku 3.1 je celá část označena jako k a desetinná jako x.



Obrázek 3.1: Diagram 16b logaritmické násobičky [14]

Součet logaritmů provádí klasická sčítačka. Po sečtení se z logaritmické podoby převádí výsledek do binárního čísla. Převod zajišťuje konvertor z logaritmické do binární reprezentace (LBC). Konvertor se skládá z posouvače a provádí odlogaritmování. Jako signál posunu se použije celá část logaritmu nacházející se na horních bitech vstupní sběrnice. Před desetinnou část logaritmu se doplní jednička a celé desetinné číslo je použito jako vstup posouvače. Po provedení posunu je na výstupu hodnota, která představuje přibližný součin operandů vstupujících do logaritmické násobičky.

3.2.1 Využití aproximačních sčítaček

Parametry logaritmické násobičky lze měnit využitím různých typů nepřesných sčítaček. Díky sčítačkám, u nichž lze nastavit počet nepřesných méně významných bitů, může být celý obvod menší, rychlejší nebo mít menší spotřebu. V [14] jsou popsány tři druhy nepřesných sčítaček použitých v logaritmických násobičkách.

Lower-part-or adder (LOA) je n-bitová sčítačka, která se skládá ze dvou částí, z m-bitové nepřesné sčítačky a $n - m$ bitů dlouhé přesné sčítačky. Přesná část počítá významnější bity. V nepřesné části nejsou vstupy sečteny standardně, ale je mezi nimi pouze provedena operace OR, kterou provádí pro každý bit výsledku pouze jedno hradlo typu OR. Pouze mezi nejvýznamnějšími bity nepřesné části sčítačky se provádí operace AND, jejíž výsledek se použije jako přenos do přesné části sčítačky. Oproti velikosti běžné sčítačky dochází k redukci počtu použitých hradel.

Mirror adder-A3 (MAA3) se stejně jako předchozí sčítačka skládá z přesné a nepřesné části. Jako výsledek v nepřesné části se používají odpovídající bity jednoho ze vstupů sčítačky. Jako přenos do přesné části se použije vstupní hodnota druhého ze vstupů na odpovídající pozici.

Set-one adder (SOA) je dalším typem přibližné sčítačky používané v Logaritmických násobičkách. Tato sčítačka všechny výsledné bity nepřesné části nastavuje jako jedničkové. Mezi nejvýznamnějšími bity nepřesné části se provede operace AND a stejně jako u LOA se výsledek použije jako přenos do přesné sčítačky.

3.3 Vylepšená logaritmická násobička

Podstatnou nevýhodou ALM násobiček je podhodnocování výsledku. Protože se část dat během výpočtu aproximuje, přibližné výsledky jsou menší, než je přesná hodnota. To představuje problém hlavně u výpočtů, kde se data násobí opakovaně a dochází tak ke kumulaci chyby. Nová vylepšená logaritmická násobička (Improved logarithmic multiplier, zkr. ILM) navržená v roce 2019 [1] pracuje, tak, že přibližné výsledky nabývají hodnot jak nižších, tak i vyšších, než je přesný výsledek. Díky tomu se při iterativním násobení nezmenšují výsledky.

Navržená násobička rozdělí každý činitel na jeho nejbližší dvojkovou mocninu a rozdíl mezi jeho hodnotou a nejbližší dvojkovou mocninou. Přibližné vynásobení lze zapsat následujícím způsobem.

$$\alpha = m_1 + q_1, \text{ kde } m_1 = 2^{k_1} \tag{3.2}$$

$$\beta = m_2 + q_2, \text{ kde } m_2 = 2^{k_2} \tag{3.3}$$

$$\alpha \times \beta \approx (2^{k_1+k_2} + q_2 2^{k_1} + q_1 2^{k_2}) \tag{3.4}$$

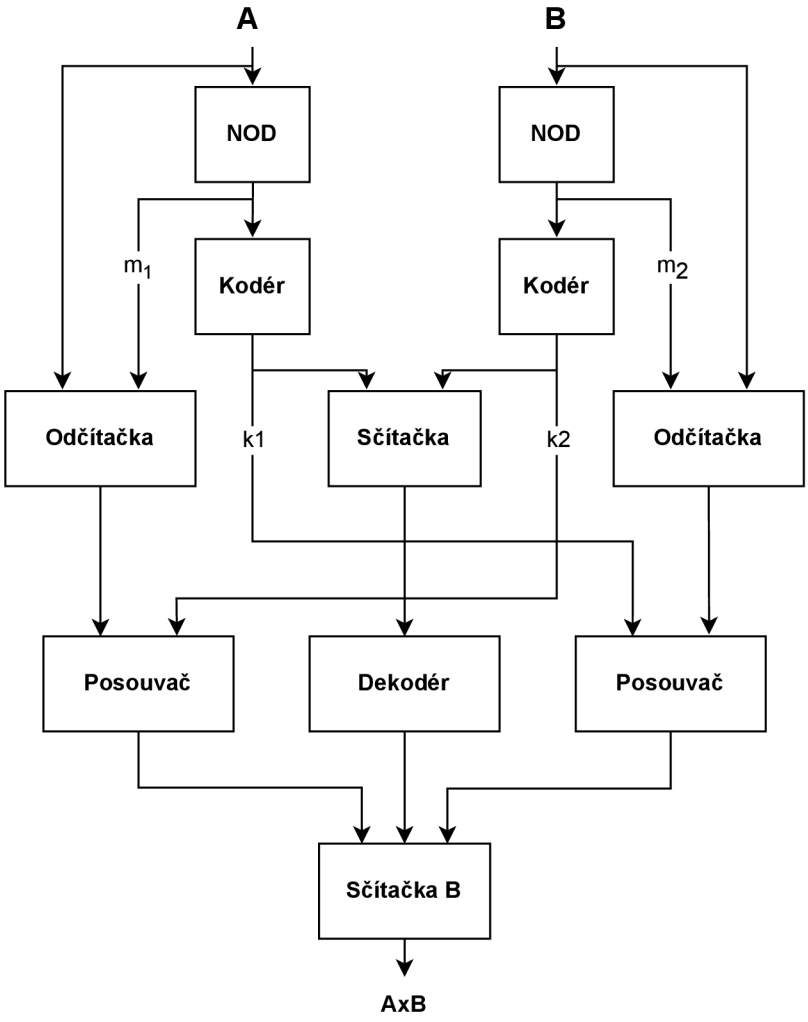
Jak je vidět v rovnici, všechny tři operandy jsou součiny dvojkových mocnin, které se v hardwaru zajišťují posunem doleva.

Narozdíl od Mitchellovy násobičky se u operandů hledá nejbližší dvojková mocnina, která je v některých případech vyšší než samotný operand. K tomuto účelu se v násobičce používá detektor nejbližší jedničky (NOD).

Tento obvod podobně jako LOD hledá nejlevější výskyt jedničky v binární reprezentaci operandu k určení nejbližší dvojkové mocniny, ale v případě, že za nejvýznamnější jedničkou na pozici n následuje jednička i na pozici $n-1$, na výstupu bude označena pozice $n+1$ jako nejbližší dvojková mocnina m_1 a m_2.

Například pokud je na vstupní sběrnici číslo 55, které je v binární reprezentaci zapsané posloupností bitů jako 00110111_2, na výstup detektoru se zapíše hodnota $001000000_2 = 64$. Hodnota byla zaokrouhlena na vyšší dvojkovou mocninu.

Do tvaru logaritmů k_1 a k_2 dvojkových mocnin jsou data převedena pomocí prioritních kodérů. Jak je patrné z obrázku 3.2 sběrnice z kodéru jsou přivedeny na vstup sčítačky. Přesná sčítačka provádí součet logaritmů, který je následně odlogaritmován pomocí dekodéru. Tím je získán člen $2^{k_1+k_2}$ z rovnice 3.4. Dva zbytkové členy q_1 a q_2 jsou vypočteny odčítačkou z originálních hodnot a dvojkových mocnin operandů a posunuty podle hodnot k_2 a k_1 tak, že vzniknou hodnoty $q_1 2^{k_2}$ a $q_2 2^{k_1}$. Nakonec se sečtou všechny tři výsledné členy a získá se přibližný součin.



Obrázek 3.2: Násobička typu ILM [1]

3.3.1 Využití aproximačních sčítaček v ILM

Za účelem zlepšení parametrů byla v obvodu násobičky použita aproximační sčítačka stejně jako v předchozím případě. Mitchellův typ násobičky podhodnocuje výsledek, a proto bylo vhodné použít sčítačku, která výsledek nadhodnocuje. V této násobičce se používá přibližná úplná sčítačka 2.3.2, s aproximační částí takovou, kde se nenastavují všechny nepřesné bity na hodnotu 1 jako v SOA, ale jedničkové a nulové bity se v pořadí střídají.

3.4 Násobení nulou

Problémem logaritmických násobiček je násobení nulou. Protože logaritmus nuly není definován, násobička se při počítání s nulovým operandem chová, jako by operand měl hodnotu jedna. Tím se zvyšuje chybovost násobičky, zvláště pak WCE. V článku [11] byl popsán způsob detekce a odstranění tohoto problému. K rozpoznání nulového vstupu se využívá detektor, který v případě nalezení operandu s hodnotou nula, vytvoří signál k vynulování výstupu násobičky.

Kapitola 4

Implementace logaritmických násobiček

Tato kapitola popisuje, jakým způsobem byly násobičky podle článků implementovány. Představuje konstrukci jednotlivých komponent. Dále je zde uveden způsob testování chybovosti a generování násobiček v jiných reprezentacích určených k určení fyzických parametrů.

4.1 Nástroj ArithsGen

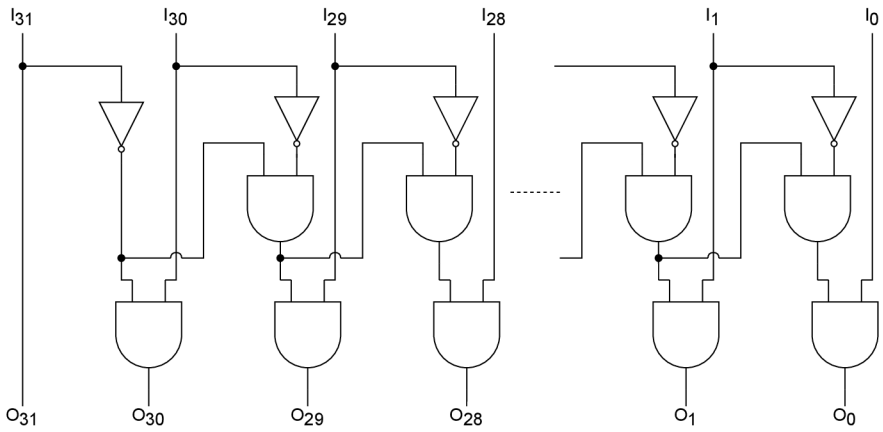
Obvody logaritmických násobiček jsou popsány v jazyce Python s využitím nástroje ArithsGen, který slouží ke generování aritmetických obvodů do různých reprezentací [12]. Díky tomuto nástroji je možné vytvořit strukturu jednoho obvodu v Pythonu a převést ji do C, Verilog, Blif a CGP formátu. Tento nástroj klade důraz na modularitu a hierarchické konstruování, což umožňuje skládání a obměnu komponent násobiček. Nejprve jsou popsány dílčí subkomponenty a následně spojeny do větších celků. Součástí nástroje ArithsGen je i knihovna obsahující kódy již implementovaných obvodů. Část z nich byla použita při návrhu logaritmických násobiček. Přestože ArithsGen obsahuje značné množství již popsaných aritmetických obvodů, bylo třeba implementovat některé specifické komponenty jako posuvníky, dekodéry, invertor a další.

4.2 Subkomponenty násobiček

Aby byly subkomponenty využitelné v obvodech různých velikostí, všechny jsou popsány s nastavitelnou šířkou vstupní sběrnice. Každá komponenta je implementována v příslušné třídě. Podle svého typu dědí třídy komponent z nadřazené třídy `ArithmeticCircuit` nebo `GeneralCircuit`. Společným atributem všech obvodů je seznam vstupů. Vstupy tvoří sběrnice s danou bitovou šířkou. Velikost vstupních sběrnic určuje, jak velký bude vytvořený obvod a jakou šířku budou mít jeho výstupní sběrnice. Díky hierarchickému zanoření komponent násobiček lze určit pouze velikost vstupních sběrnic celé násobičky. Velikost dílčích obvodů se určí podle nadřazené komponenty.

4.2.1 Detektor nejvýznamnější jedničky

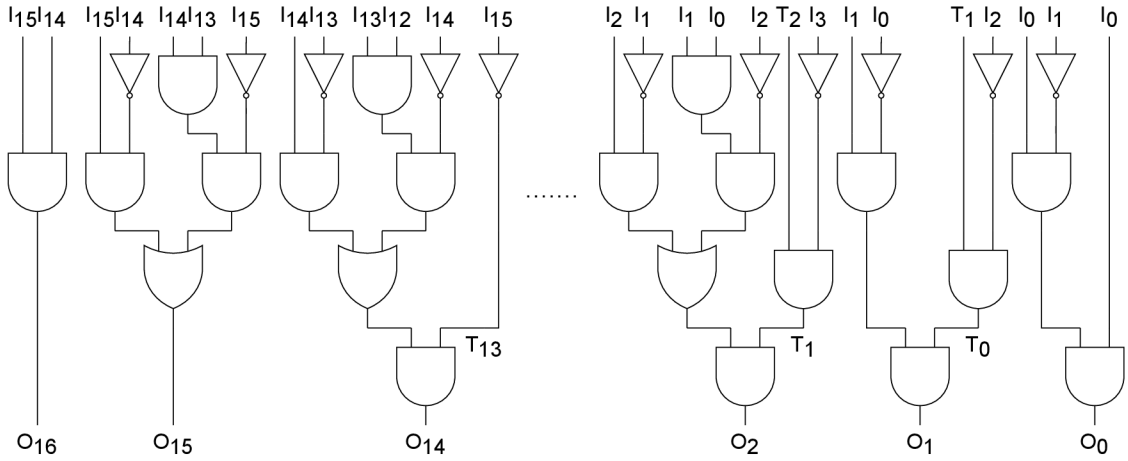
Detektor nejvýznamnější jedničky (Leading One Detector, zkr. LOD), jehož role ve výpočtu binárního logaritmu je popsána v 3.2, je implementován ve třídě `LeadingOneDetector`. Detektor přijímá pouze jednu vstupní sběrnici o délce n. Skládá se celkem z $3 * (n - 2) + 2$ hradel. V porovnání s ostatními komponentami násobičky je jeho velikost téměř zanedbatelná. Kritická cesta má délku n to je způsobeno přenášením informace z levé strany, zda již byl nalezen jedničkový bit. Struktura obvodu se opakuje pro každý bit vstupu, mimo prvních dvou, jak je patrné z obrázku 4.1. Při implementaci byla analyzována struktura a opakující se vzor v obvodu, aby bylo možné implementovat obvod univerzální velikosti.



Obrázek 4.1: Leading one detector 32b [18]

4.2.2 Detektor nejbližší jedničky

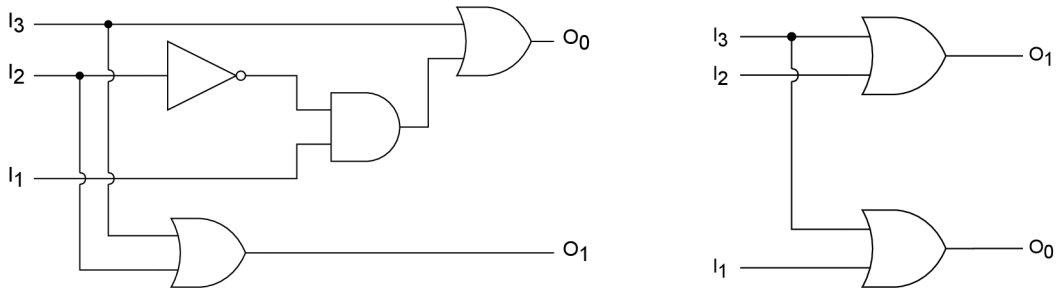
Funkce detektoru nejbližší jedničky (Nearest one detector, zkr. NOD) byla popsána v části 3.3. Tento obvod má komplikovanější strukturu než LOD. K výpočtu hodnoty konkrétního bitu na pozici k je třeba přenos informace z vyšších řádů, zda již nebyla nalezena jednička. Zároveň je pro určení k-tého bitu třeba hodnot $k - 1$ a $k - 2$. Pokud se první jednička nachází na pozici $k - 1$, ale zároveň je druhý výskyt jedničky hned za ní na pozici $k - 2$, na výstupní sběrnici bude jednička na pozici k. Tento obvod pro vstupní sběrnici délky n má stejně jako LOD délku kritické cesty n, ale na rozdíl od něj je složen z trojnásobného množství hradel. Jak je patrné z obrázku 4.2, jeho konstrukce je komplikovanější, protože vzor se opakuje mimo první 3 a poslední 2 bity. Při vytváření univerzálně velkého obvodu, bylo nutné popsat zvlášť případy, kdy velikost vstupní sběrnice dosahuje méně než 5 bitů. Implementace NOD je obsažena v třídě `NearestOneDetector`.



Obrázek 4.2: Nearest one detector [1]

4.2.3 Prioritní kodér

Prioritní kodér v logaritmických násobičkách slouží spolu s NOD nebo LOD k nalezení dvojkového logaritmu operandu. Z čísla v binární reprezentaci na vstupu určuje pozici bitu s nejvýznamnější, a tedy nejlevější jedničkou a tuto pozici ve formě binárního čísla dává na výstup. Velikost výstupní sběrnice je $\lceil \log_2 n \rceil$, kde n je velikost vstupní sběrnice. Schéma 4.3a znázorňuje vnitřní strukturu prioritního kodéru. Narozdíl od klasického kodéru může prioritní na vstupu přijímat různé vstupní hodnoty. U klasického kodéru musí být zajištěn pouze jeden výskyt jedničkového bitu na vstupní sběrnici. V násobičkách v této práci je vstup kodéru vždy připojen na výstup NOD nebo LOD, čímž je zajištěna validní hodnota vstupu. Tudíž zde lze použít klasický kodér, který má jednodušší strukturu než prioritní kodér, jak je patrné z obrázku 4.3.



(a) Prioritní kodér 4 na 2

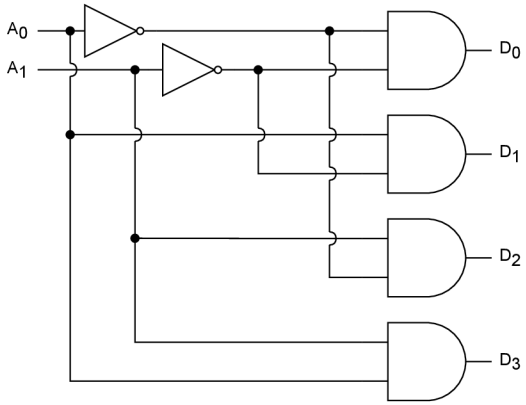
(b) kodér 4 na 2

Obrázek 4.3: Rozdíl mezi prioritním a běžným kodérem

4.2.4 Dekodér

Dekodér zastává opačnou funkci než Kodér. Ze vstupu o n bitech převádí data na výstup s délkou 2^n bitů. Na výstupu je na hodnotu 1 nastaven pouze jeden bit sběrnice, a to na pozici dané binární hodnotou na vstupu. Dekodér se v logaritmických násobičkách používá k odlogaritmování výsledku. Délka výstupní sběrnice narůstá exponenciálně vzhledem ke vstupu. U některých velikostí logaritmických násobiček může dojít k vytvoření výstupní sběrnice dekodéru delší, než má být výstupní sběrnice celé násobičky. Přebytečné bity nesou

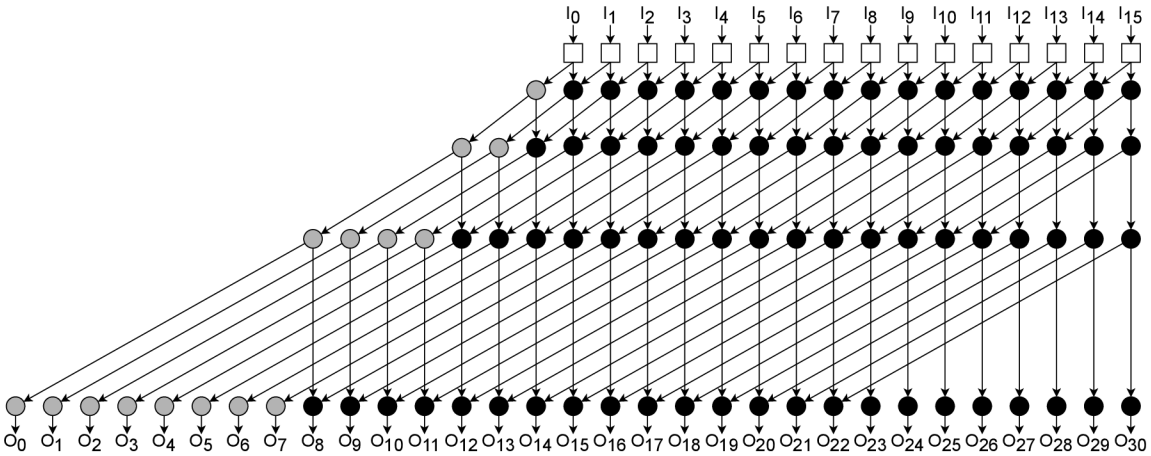
vždy nulovou hodnotu a proto je lze zanedbat a nepřipojit do dalších částí. Při syntéze se prvky nenapojené na výstup nevytvoří.



Obrázek 4.4: Dekodér

4.2.5 Posouvač

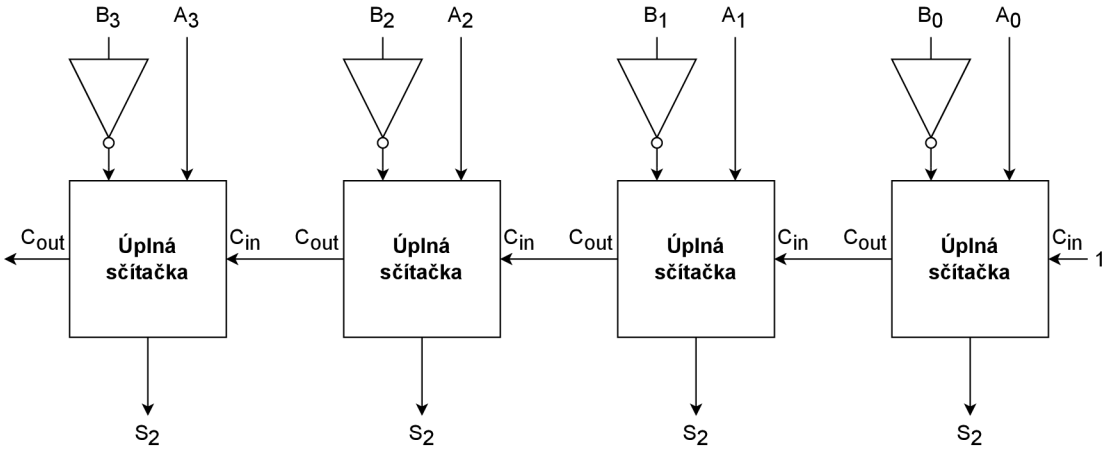
Posouvač umožňuje bitový posun čísel doprava (na nižší bitové pozice) nebo doleva (na pozice vyšší). V logaritmických násobičkách popsaných v části 3.3 zastupuje násobení dvojkovou mocninou. Má dvě vstupní sběrnice. Jedna nese hodnotu čísla n, které bude posouváno. Na druhé je nastavena hodnota k určující, o kolik bitových pozic bude číslo posunuto. Operace, kterou posouvač provádí, lze vyjádřit jako $n \times 2^k$. Schéma 4.5 zobrazuje zapojení posouvače. Každý uzel znázorňuje multiplexor. Implementovaný posouvač je znaménkový. Zachování znaménka čísla n je zajištěno připojením šedých uzlů na nejvýznamnější bit vstupní sběrnice místo na konstantní nulu, jako je tomu u neznaménkové varianty posouvače. Násobičky Mitchellova typu používají pravou variantu posouvače k získání zlomkové části logaritmu operandu a levou variantu k odlogaritmování. Stejně jako dekodér, může posouvač nabýt příliš velkých rozměrů. Tento problém se řeší ořezáním nepotřebné části obvodu podobně jako případě dekodéru. Posouvače jsou implementovány ve třídách `LeftShifter` a `RightShifter` podle popisu v článku [23].



Obrázek 4.5: Posouvač [23]

4.2.6 Odčítačka

Neznaménková odčítačka je využita v ILM násobičce k určení rozdílu mezi nejbližší dvojkovou mocninou operandu a jeho samotnou velikostí. Její konstrukce vychází z klasické vícebitové sčítačky. V odčítačce není první sčítačka poloviční, ale celá, a její bit přenosu je permanentně nastaven na logickou jedničku. Vstupní bity jednoho z operandů jsou znegovány pomocí hradel NOT.



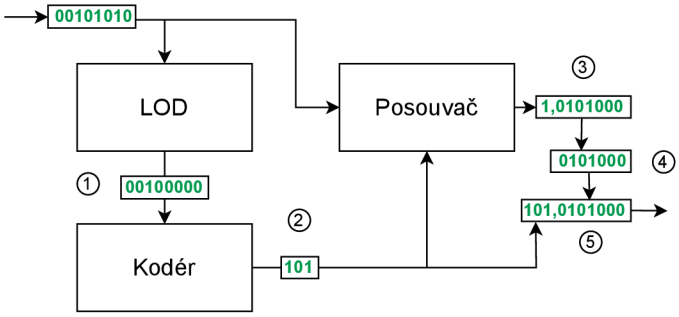
Obrázek 4.6: Odčítačka tvořená celými sčítačkami

4.3 Implementace násobiček

Logaritmické násobičky jsou složeny z několika číslicových obvodů vzájemně propojených sběrnicemi o různých šířkách. Všechny komponenty byly vytvořeny s nastavitelnou délkou. To umožňuje vytvářet libovolně velké násobičky a délky jejich jednotlivých částí určovat podle nadřazené komponenty. Tím je usnadněno propojení částí násobičky.

4.3.1 ALM

Násobička uvedená v části 3.2, byla implementována ve třídě `MitchellBaseLogMultiplier`, která dědí z třídy `ArithmeticCircuit`. Vstupními parametry jsou objekty sběrnic A a B. Pokud nemají sběrnice stejnou délku, je kratší z nich rozšířena, tak aby byly stejně dlouhé. Pro oba vstupy jsou v násobičce implementovány převodníky z binární do logaritmické podoby, skládající se z detektoru nejvýznamnější jedničky, kodéru a pravého posouvače [23]. Zapojení prvků v převodníku je znázorněno na obrázku 4.7. Ze vstupního operandu je určena jeho nejvyšší celá dvojková mocnina tak, že je detekována nejvýznamnější jednička pomocí LOD (krok 1). Tato hodnota je kodérem převedena na její dvojkový logaritmus (krok 2). Zlomková část je vytvořena v posouvači tak, že se hodnota vstupního operandu bitově posune doprava o počet míst určených hodnotou celé části logaritmu. Tím se výstup posouvače rozšíří o desetinnou část a všechny bity za první jedničkou jsou do ní odsunuty (krok 3). Desetinná část sběrnice se ořízne od zbytku (krok 4). Pro reprezentaci desetinného logaritmu je vytvořena nová sběrnice, ve které se spojuje celá a desetinná část do jedné posloupnosti (krok 5). Příklad postupu převodu je znázorněn v obrázku 4.7 se zeleně značenými konkrétními daty pro vstupní hodnotou 42 (00101010_2).



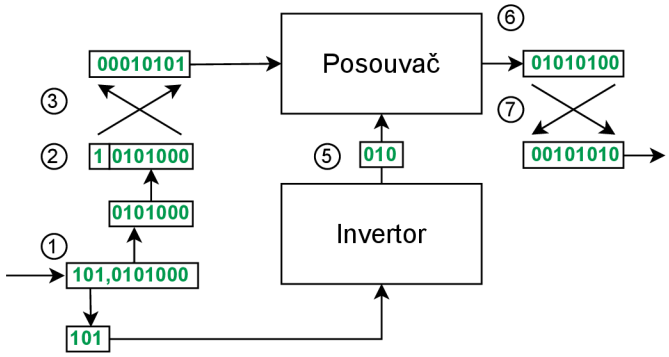
Obrázek 4.7: Převodník z binární do logaritmické reprezentace

Pro provedení součtu logaritmů je v násobičce zapojena sčítačka, jejíž velikost je určena podle velikosti vstupních sběrnic. Tím, že je zajištěna stejná velikost obou převodníků, jsou bity celé části a desetinné části logaritmu v obou operandech na stejných pozicích.

Převodník z logaritmické do binární reprezentace byl implementován pouze pomocí posouvače a invertoru. Za účelem vytvoření univerzální násobičky s nastavitelnou délkou byla provedena analýza způsobu výpočtu. V článcích, z nichž se vycházelo, byl převod z logaritmické do binární podoby popsán teoreticky, nebo pouze pro standardní velikosti násobiček. V této práci je navržena obecná konstrukce převodníku pro volitelnou bitovou velikost.

Vstupem převodníku je sběrnice obsahující na nejvýznamějších pozicích exponent a na méně významných mantisu. Podle Mitchellova algoritmu 3.1 se odlogaritmování provádí tak, že se oddělí mantisa doplní se před ni jednička a posune se tak, aby tato jednička byla na pozici určené exponentem.

Nejprve je ze vstupu odříznuta část sběrnice nesoucí desetinnou část logaritmu (krok 1). Před nejvýznamnější bit je přidán bit s konstantní jedničkou (krok 2). Tato nová sběrnice je rozšířena na velikost mocniny dvojky v případě, že nemá standardní šířku. Posunutí čísla na sběrnici na správnou pozici je vyřešeno pomocí posouvače. Před vstupem do posouvače je sběrnice otočena tak, aby připojená jednička byla na pozici 0 (krok 3). Velikost bitového posunu je určena invertováním bitů celé části původního logaritmu (krok 5). Pomocí levého posouvače je číslo posunuto o tuto hodnotu doprava (krok 6). Výsledná sběrnice je znovu otočena a ořízuta na požadovanou délku (krok 7). Příklad postupu převodu je znázorněn v obrázku 4.8 se zeleně značenými konkrétními daty pro vstupní hodnotou $5,0625$ $(101,0101000_2)$.



Obrázek 4.8: Převodník z logaritmické do binární reprezentace

Třída `MitchellBaseLogMultiplier` je obecná pro Mitchellovu násobičku a její modifikace. Volitelným parametrem je typ sčítačky, která bude použita, případně s kolika nepřesnými bity počítá. Pokud není zvolen typ sčítačky, použije se její přesná verze. Tím je vytvořen popis klasické Mitchellovy násobičky. Třída `MitchellLogMultiplier` dědící z `MitchellBaseLogMultiplier` implementuje přímo Mitchellovu násobičku.

4.3.2 ILM

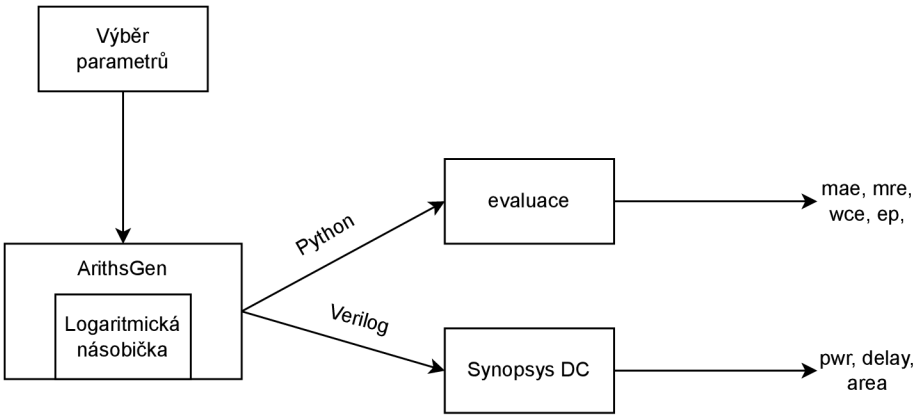
Vstupními parametry jsou objekty sběrnic A a B. Pokud nemají sběrnice stejnou délku, je kratší z nich rozšířena tak, aby byly stejně dlouhé. Pro oba vstupy jsou v násobičce implementovány detektory nejbližší jedničky, jejichž výstupy jsou připojeny do kodérů a odčítačky. Protože detektory o bit rozšiřují sběrnici, vstupy enkodérů v násobičkách standardních velikostí nemají zarovnané šířky na dvojkovou mocninu. Funkce kodéru nemusí být plně využita. Jejich výstupní sběrnice reprezentující přibližný dvojkový logaritmus se připojují na vstup přesné neznaménkové sčítačky. Provedením součtu je výstupní sběrnice rozšířena o další bit. Poté vede do dekodéru, který odlogaritmuje součet. Získaný výsledek je nepřesný z důvodu aproximace NOD. Reálný součin může být jak vyšší i nižší, protože NOD hledá nejbližší dvojkovou mocninu, se kterou se následně počítá. Ke korekci této chyby jsou v násobičce implementovány odčítačky. Vstupy jsou původní operandy a jejich aproximace nalezené pomocí NOD. Od hodnoty operandu se odečítá jeho aproximace. Odčítačka počítá rozdíl dvou neznaménkových vstupů, ale výsledek je znaménkový. Velikost chyby reprezentovaná binárním číslem je sběrnicí přivedena na vstup posouvače. Zde je bitově posunuta o hodnotu z kodéru druhého operandu, to znamená že je vynásobena přibližnou hodnotou druhého z operandů. Tím je zjištěna přibližná hodnota chyby ve výsledku způsobená nepřesným určením logaritmu. Velikost výstupu posouvače je určena velikostmi jeho vstupů.

$$n_{out} = n_{in} + \sum_{i=1}^{n_{sel}} 2^{i-1} \tag{4.1}$$

V rovnici 4.1 je n_{out} velikost výstupní sběrnice, n_{in} velikost sběrnice posouvaného vstupu a n_{sel} je velikost sběrnice kontrolního signálu. Výstup posouvače může dosahovat větší šířky než samotný výstup celé násobičky, proto je sběrnice oříznuta na potřebnou velikost. Korekce výsledku je provedena sečtením hodnot chyb aproximace obou operandů a přičtením k získanému výsledku z dekodéru. Tuto operaci provádí dvě znaménkové sčítačky.

4.4 Testování parametrů

Parametry implementovaných násobiček jsou získány pomocí testů napsaných v jazyce Python a syntézy v nástroji Synopsys Design Compiler. Diagram 4.9 znázorňuje průběh vytvoření a ohodnocení násobiček.

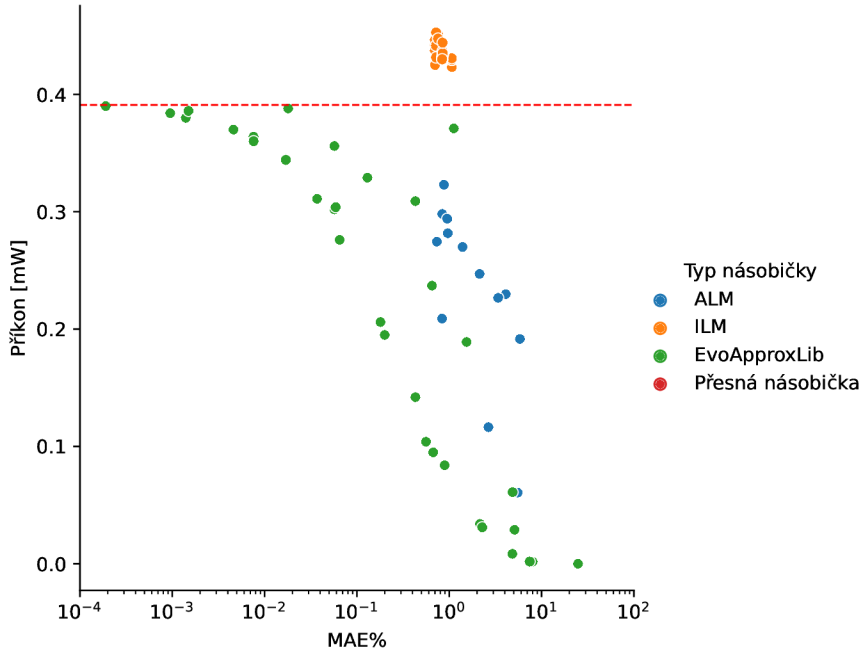


Obrázek 4.9: Generování a testování nových obvodů logaritmických násobiček.

Chybovost obvodů byla počítána pomocí NumPy. Pro každý druh logaritmických násobiček byly sepsány testy hodnotící jejich kvalitu. Pokud není násobička příliš velká, spočítají se výsledky pro všechny možné kombinace vstupních hodnot. V opačném případě se vybere menší podmnožina náhodných vstupů. Na počátku se vytvoří pole 2^n různých hodnot, kterých může nabývat vstup na sběrnici o délce n. Druhé pole se získá transpozicí prvního. Do vytvořeného objektu třídy testované násobičky se jako parametry zadají tato dvě pole. Získaný výstup je ve formě dvourozměrného pole s 2^{2n} hodnotami výsledků. Přesné hodnoty výsledků se získají jednoduchým vynásobením zmíněných polí. Díky dalším funkcím knihovny NumPy lze efektivně počítat rozdíly mezi získanými hodnotami. Fyzické parametry byly zjištěny pomocí syntézy v nástroji Synopsis DC, který pracuje s Verilog reprezentacemi. K převodu z jazyka Python byl využit nástroj ArithsGen.

4.5 Srovnání

Z implementovaných obvodů byly pomocí nástroje ArithsGen vygenerovány jejich C a Verilog soubory. Násobičky ve Verilog formátu prošly syntézou s použitím nástroje Synopsys Design Compiler. Dále byla otestována jejich chybovost.



Obrázek 4.10: Parametry existujících osmibitových logaritmických násobiček ILM a ALM z článků v porovnání s obvody z EvoApproxLib

Parametry obou typů obvodů s různými obměnami sčítaček jsou znázorněny v grafu 4.10 společně s násobičkami z knihovny EvoApproxLib. Jak je patrné, výměnou sčítaček bylo dosaženo větších změn u typu ALM. Celkově mají logaritmické násobičky horší poměr mezi příkonem a průměrnou aritmetickou chybou v porovnání s evolučně navrženými násobičkami. Implementované násobičky se vyznačují poměrně velkou chybovostí. MAE se pohybuje kolem jednoho procenta. Architektury ILM navíc spotřebují více energie než přesné sčítačky. Tento problém způsobuje použití neefektivních posouvačů.

V provedeném srovnání ILM a ALM v článku [4] dosahují ILM lepších výsledků. To může být způsobeno jinou konstrukcí údajné Mitchellovy násobičky, která je tomto článku strukturně bližší ILM.

Kapitola 5

Návrh vylepšení logaritmických násobiček

Tato kapitola se zabývá nalezením způsobu zlepšení poměru příkonu a chybovosti logaritmických násobiček a popisuje metodiku pro systematické zlepšení tohoto poměru.

Cílem práce je identifikovat možné způsoby vylepšení logaritmických násobiček a aplikovat je. Součástí výstupu bude knihovna implementací modifikovaných násobiček, které rozšíří stávající knihovnu ArithsGen. V části 2.3.2 jsou uvedeny některé metody aproximace násobiček. Tato práce se zaměřuje na logaritmické násobičky, které mají specifickou strukturu sestávající z několika různých subkomponent. Jedním ze způsobů modifikace by mohlo být nahrazení nebo úprava jednotlivých subkomponent.

5.1 Modifikace subkomponenty

Výběr subkomponenty pro aproximaci závisí na několika parametrech. Důležitými aspekty při výběru subkomponenty k aproximaci jsou:

- velikost
- zpoždění
- umístění v násobičce
- náchylnost k chybě
- existence vhodné aproximační metody

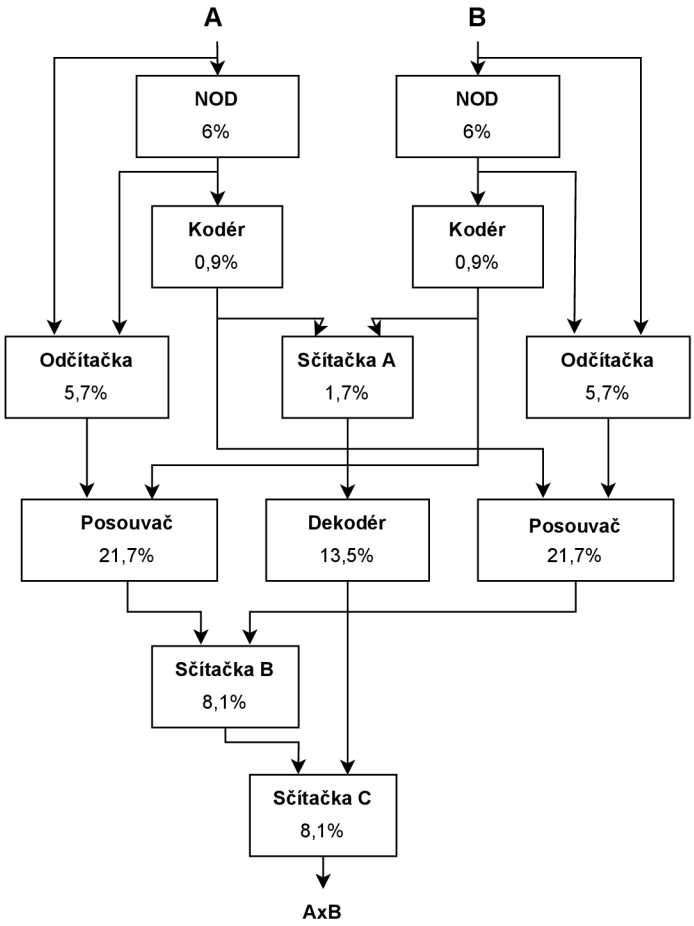
U větších obvodů lze aproximací dosáhnout výraznějšího snížení plochy a příkonu výsledné násobičky. Při výběru bude vhodné zaměřit se na obvody, které svou velikostí výrazně ovlivňují fyzické parametry násobičky. Přibližný podíl jednotlivých částí násobičky na velikosti byl určen sečtením jim náležících hradel podle vygenerovaného kódu v jazyce C. Při syntéze se tento podíl může mírně změnit. Diagramy 5.1 a 5.2 znázorňují procentuální velikost jednotlivých částí násobiček podle počtu obsažených hradel. Komponenty umístěné na začátku obvodu, do kterých vstupují data nejdříve, mohou mít větší vliv na výsledek, než pokud by byly na konci. Problémem je, že při změně dat na počátku výpočtu bude chyba distribuována celým obvodem a její velikost by mohla narůstat. Například při vzniku chyby ve sčítačce A v rámci ILM násobičky se po provedení dekódování v dekodéru velikost chyby

řádově zvýší. Naproti tomu chyba sčítačky C umístěné na konci násobičky se propíše do výsledku se stejnou absolutní velikostí.

Aproximace NOD by mohla být problematická, protože komponenta se nachází v násobičce před dekodérem a posouvačem. Chyba způsobená aproximací NOD se průchodem dat těmito obvody zvýší. K získání validního výstupu je třeba předávat informaci skrz celou komponentu z vyšších pozic o již nalezeném výskytu jedničky. Pokud by se změnou přerušila tato cesta, mohlo by dojít k zápisu více jedničkových bitů na výstupní sběrnici. Zároveň je velikost chyby způsobená posunutím pozice detekovaného bitu v řádovém rozpětí. Ani z hlediska ušetřené velikosti se aproximace NOD nejeví slibně, protože oproti ostatním částem je jeho velikost zanedbatelná.

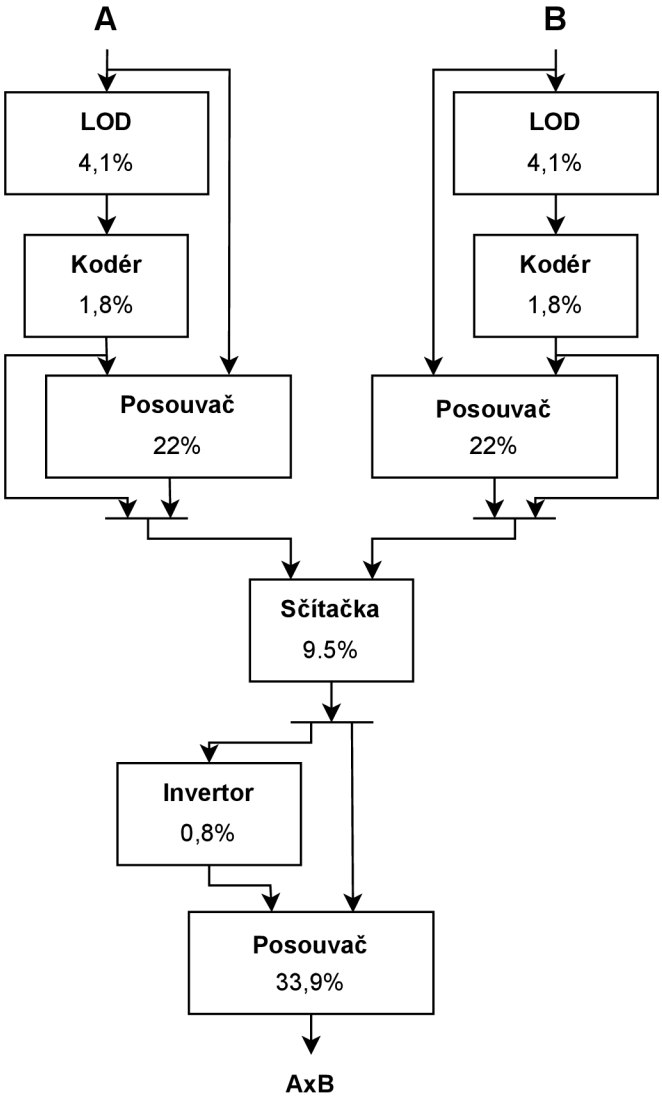
Jako nejvhodnější způsob modifikace ILM se jeví obměna sčítaček B a C. Jedná se o poměrně velké obvody umístěné před výstupem násobičky. Oproti posouvačům by mohly mít i větší zpoždění [3]. Jejich úpravou se nezvyšuje chyba výsledku násobení tak výrazně jako při aproximaci sčítačky A. Důvodem je, že sčítačka A provádí součet logaritmů. Poměrně malá chyba v součtu logaritmů se po odlogaritmování projeví, jako mnohonásobně větší. Navíc tato sčítačka je zpravidla menší než zbývající dvě. Například v osmibitové násobičce je sčítačka logaritmů čtyřbitová a obě dvě sčítačky výsledku šestnáctibitové.

Při úpravě lze využít již vytvořené obvody aproximačních sčítaček z knihovny EvoApproxLib a navázat tak na výzkum evolučních aproximačních struktur. Do implementovaných násobiček z kapitoly 3 budou namísto dvou přesných sčítaček zapojeny evolucí modifikované aproximační obvody provádějící přibližný součet.



Obrázek 5.1: Poměr počtu hradel jednotlivých komponent osmibitové ILM

V případě ALM byly již popsány změny typů sčítaček a jejich vliv na přesnost součinu. Ale jednalo se pouze o ručně navržené sčítačky typu SOA, LOA a MAA3. Bude tedy vhodné vyzkoušet použití evolučně modifikovaných verzí sčítaček. V ALM se nachází pouze jedna sčítačka. V osmibitových násobičkách jsou použity desetibitové sčítačky. Ty ale nejsou součástí EvoApproxLib, jsou však k dispozici v neveřejné části knihovny v rámci výzkumné skupiny EHW@FIT.



Obrázek 5.2: Poměr počtu hradel jednotlivých komponent osmibitové ALM

5.2 Evoluční modifikace celého obvodu

Dalšího zlepšení parametrů a jejich poměru by mohlo být dosaženo modifikací celého obvodu logaritmické násobičky. Z obvodů realizovaných v části 4 v jazyce Python budou vybrány nejvhodnější verze. Poté budou pomocí nástroje ArithsGen převedeny do CGP reprezentace a následně evolučně optimalizovány pomocí CGP. Bude nutné zvolit vhodnou fitness pro řízení evolučního výběru. Z nově vytvořených obvodů budou na základě jejich parametrů vybrána nejlepší řešení z hlediska poměru fyzických a chybových parametrů.

5.3 Korekce při násobení nulou

Problémem logaritmických násobiček, který snižuje jejich přesnost je násobení nulou. Pokud je operand nulový násobička jeho hodnotu aproximuje na jedna. To ovlivňuje především

WCE. Na MAE se tato chyba neprojevuje tak výrazně protože při testování tvoří nulové vstupy jen zlomek všech vyzkoušených vstupních hodnot. Pokud by byly logaritmické násobičky použity v aplikaci, kde se častěji vyskytují součiny nul, mohl by se tento problém projevit výrazněji. Řešením je použití detektoru nulového operandu a korekce výsledku na základě jeho detekce. Detektor bude začleněn do násobičky a ověří se jeho vliv na výslednou chybovost a další parametry obvodu.

Kapitola 6

Vylepšení logaritmických násobiček

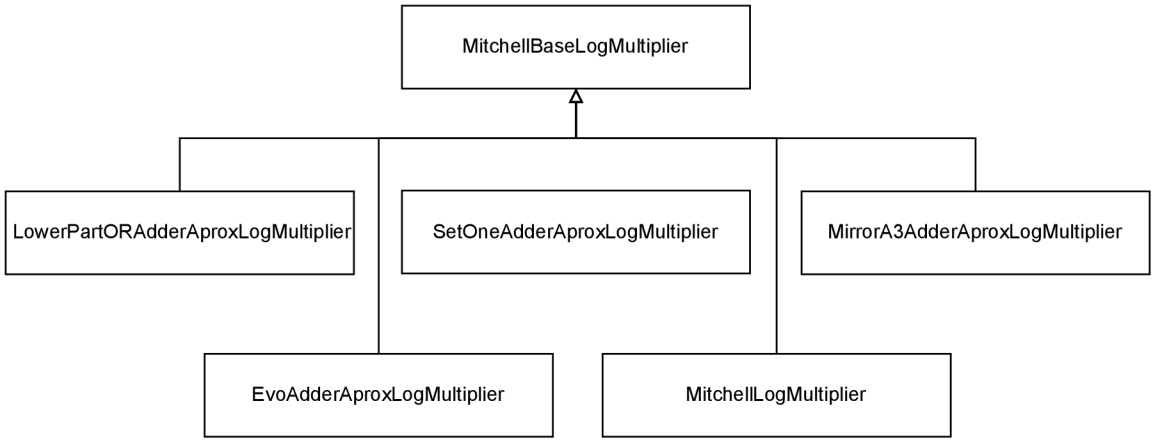
V rámci této práce jsou zkoumány dva hlavní přístupy vedoucí k vylepšení logaritmických násobiček. Jedním je výměna části násobičky při zachování původní konstrukce. Druhý přístup je zaměřen na obvod násobičky jako na jeden celek, který je upravován pomocí evoluční optimalizace. V této kapitole je popsán způsob implementace vylepšení logaritmických násobiček a jejich následné testování.

6.1 Modifikace pomocí nepřesných sčítaček

Část 5.1 popisovala výběr subkomponent k následné úpravě. V ALM se nachází pouze jedna sčítačka zatímco z ILM byly vybrány dvě nejvhodnější. V této části je popsáno, jakým způsobem probíhala výměna vybraných subkomponent za přibližné úplné sčítačky nebo evolučně navržené sčítačky.

6.1.1 Přibližné úplné sčítačky

Aproximační sčítačky uvedené v článku [14] byly naimplementovány s variabilní velikostí. Protože tyto sčítačky mají stejný tvar a liší se pouze v realizaci výpočtu nepřesných bitů, je pro ně vytvořena třída `AproximateAdder`. Celkovou šířku sčítačky určuje šířka sběrnice, která se jako objekt vkládá do jejího konstruktoru. Konkrétní typ se vytvoří podle zadaného parametru `type`. Pokud nejsou parametry zadány vytvoří se přesná sčítačka. Počet nižších nepřesných bitů určuje `inexactBits`. Pro výslednou sčítačku je nejprve zkonstruována nepřesná část o zadané délce a typu. Poté se vytvoří přesná sčítačka z úplných sčítaček a přenos z této části nastaví jako C_{in} v nejnižší úplné sčítačce přesné části. Velikost přesné části je vypočtena z celkové velikosti a počtu nepřesných bitů sčítačky. Násobičky s nepřesnými sčítačkami byly implementovány z několika důvodů. Slouží pro porovnání hodnot metrik nově navržených aritmetických obvodů v této práci. Byly použity jako počáteční obvody pro CGP modifikaci. Bude možné je využívat jako součást knihovny ArithsGen. Pro úplnost vznikly samostatné třídy násobiček `LowerPartORAdderAproxLogMultiplier`, `SetOneAdderAproxLogMultiplier` a `MirrorA3AdderAproxLogMultiplier`, které dědí z třídy `MitchellBaseLogMultiplier`. Parametrem `inexBits` se určuje počet nepřesných bitů použité sčítačky. Diagram dědičnosti těchto tříd je znázorněn na obrázku 6.1.



Obrázek 6.1: Diagram tříd logaritmických násobiček s aproximačními sčítačkami

6.1.2 Evolučně navržené sčítačky

Tyto sčítačky tvoří obvody s pevnou délkou, proto je nelze vkládat do libovolně velkých násobiček. Z knihovny EvoApproxLib byly použity implementace osmibitových neznaménkových sčítaček pro čtyřbitové násobičky a šestnáctibitových pro osmibitové násobičky Mitchellova typu.

Převod do ArithsGen

V knihovně EvoApproxLib jsou vytvořené obvody dostupné ve formátech C a Verilog, které jsou vhodné pro syntézní vyhodnocení, ale pro snadnější práci s nimi a pro budoucí využití, bylo nutné je převést do formátu, s jakým pracuje nástroj ArithsGen. K tomuto účelu byl vytvořen skript, který ze vstupního souboru obsahujícího evolučně vygenerovaný obvod v jazyce C, vysází do výstupního souboru kód odpovídajícího obvodu v požadovaném formátu v jazyce Python. Tento skript je využitelný i při dalším překladu obvodů v jazyce C z knihovny EvoApproxLib. Jelikož hlavním cílem bylo získat implementace aproximačních sčítaček pro tuto práci ve formátu nástroje ArithsGen, skript byl vytvořen podle konkrétních zápisů implementací. Z toho důvodu nemusí být použitelný pro převod jakýchkoliv popisů obvodů v jazyce C.

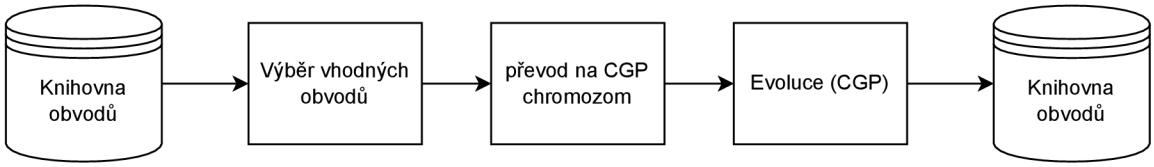
Pomocí skriptu byly přeloženy soubory s kódy sčítaček z jazyka C a uloženy ve tvaru použitelném pro další práci. Každý obvod tvoří nová třída s původním názvem sčítačky. Objekty této třídy lze přímo zapojit v obvodu násobičky. Pro snadnější generování násobiček se všemi sčítačkami byla vytvořena obecná třída `EVOAproximateAdder`, které se jako parametr předává název požadované sčítačky. Název se použije jako klíč ve slovníku obsahujícím objekty všech dostupných evolučních sčítaček. Díky zavedení všech sčítaček ve slovníku lze při generování obvodů násobiček využít iterace nad jeho položkami. Generování všech obvodů typu ILM je implementováno v `test-ILM-EVO`. Při spuštění dojde k vytvoření vstupních sběrnic se zadanou šířkou a tím i k určení velikosti násobičky. Následně se ve dvou vnořených cyklech prochází slovník sčítaček určený pro konkrétní velikost obvodu. Do každé násobičky jsou vloženy dvě evoluční sčítačky.

6.2 Evoluční úprava

Za účelem dalšího vylepšení logaritmických násobiček byla použita evoluční otimalizace celých struktur vybraných obvodů. U obvodů implementovaných podle popisu v literatuře

byly vyhodnoceny parametry a vytvořena Pareto fronta. Z té bylo vybráno několik jedinců ke genetickému vylepšení. K dalšímu vylepšení bylo vybráno i několik násobiček již upravených v části 6.1. Vybrané obvody byly převedeny do CGP formátu pomocí nástroje ArithsGen.

Tyto obvody byly následně evolučně vylepšovány s využitím existující implementace automatické aproximace pomocí CGP. Optimalizovanými parametry byl počet uzlů a průměrná aritmetická chyba. Evoluce se ukončovala po uplynutí jedné hodiny. Výběr jedinců do další generace se řídil evoluční strategií $1+\lambda$ přičemž velikost populace je 5. Chyba se vyhodnocovala na všech 65536 testovacích vektorech. Diagram 6.2 znázorňuje průběh úpravy pomocí evoluce. V rámci práce byly vytvořeny vhodné vstupní obvody a byly vybrány pro evoluci. Pro vlastní aproximaci obvodů a jejich charakterizaci byly využity existující nástroje.



Obrázek 6.2: Úprava obvodů s využitím evoluční metody

6.3 Generování a testování v Pythonu

Pro všechny typy násobiček byly napsány skripty, které vytváří několik jejich implementací s různými délkami nebo obměnami subkomponent. Každá takto vytvořená násobička se uloží ve formátu C, Verilog a CGP. Zároveň jsou všechny implementace otestovány a hodnoty jejich chybových metrik uloženy v textovém souboru.

Ověřování funkčnosti a ladění v prostředí Python je v tomto případě problematické. Implementace v nástroji ArithsGen neumožňuje krokovat v kódu násobičky za běhu výpočtu. Testování s konkrétními hodnotami na vstupech probíhalo nad obvody násobiček v C reprezentaci.

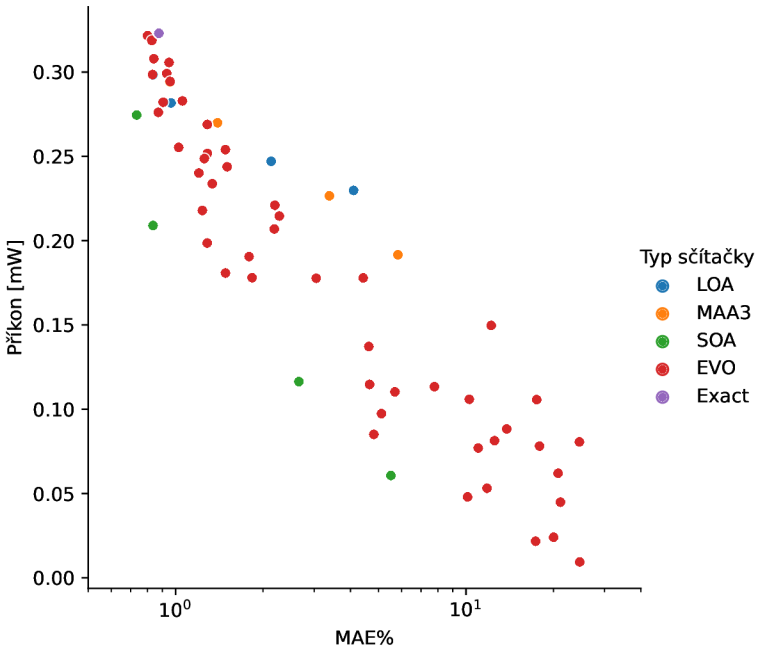
Kapitola 7

Vyhodnocení

Tato kapitola demonstruje dosažené výsledky. Jsou zde vyhodnoceny parametry získaných násobiček. Obvody byly ohodnoceny za pomocí existujících nástrojů používaných při analýze obvodů pro knihovnu EvoApproxLib: pro určení chybových metrik se využívá implementace C získána z obvodů navržených v rámci práce v nástroji AritshGen, fyzické parametry obvodů byly získány syntézou v nástroji Synopsys Design Compiler s 45nm technologickou knihovnou.

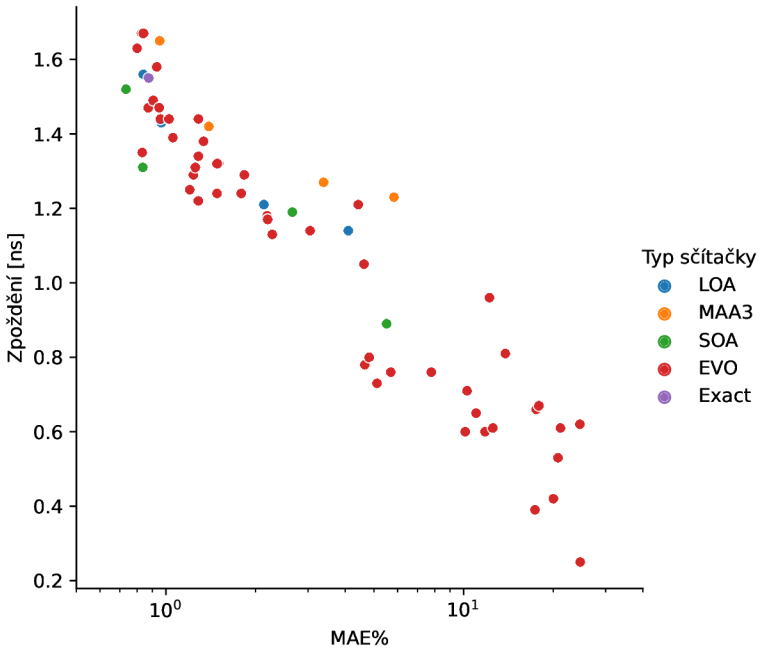
7.1 ALM využívající evoluční sčítačky

Jak bylo uvedeno v kapitole 5, v ALM násobičkách je možné nahradit použitou sčítačku. Tato část se věnuje porovnání kvality výsledků při výměně sčítaček.



Obrázek 7.1: Porovnání příkonu a průměrné aritmetické chyby ALM násobiček s přesnými a aproximačními sčítačkami z literatury a násobiček s evolučními sčítačkami.

Z grafu 7.1 je patrné, že násobičky s použitím SOA sčítaček dosahují nejlepších výsledků v poměru příkonu a průměrné aritmetické chyby. To je způsobeno vlastnostmi Mitchellovy logaritmické násobičky, která podhodnocuje výsledky. Díky nepřesným sčítačkám nadhodnocujícím součet se tyto chyby vyvažují. Použité evoluční sčítačky byly původně navrženy na konkrétní MAE. Výsledná MAE násobičky je proto větší, kvůli podhodnoceným výsledkům získaným s použitím evolučních sčítaček. Dalšího zlepšení by se zřejmě dalo dosáhnout evoluční úpravou tak, že by se očekávalo, že výsledky mají mít vyšší hodnoty. Bylo by tak možné vytvořit sčítačky určené právě pro tento typ násobiček.

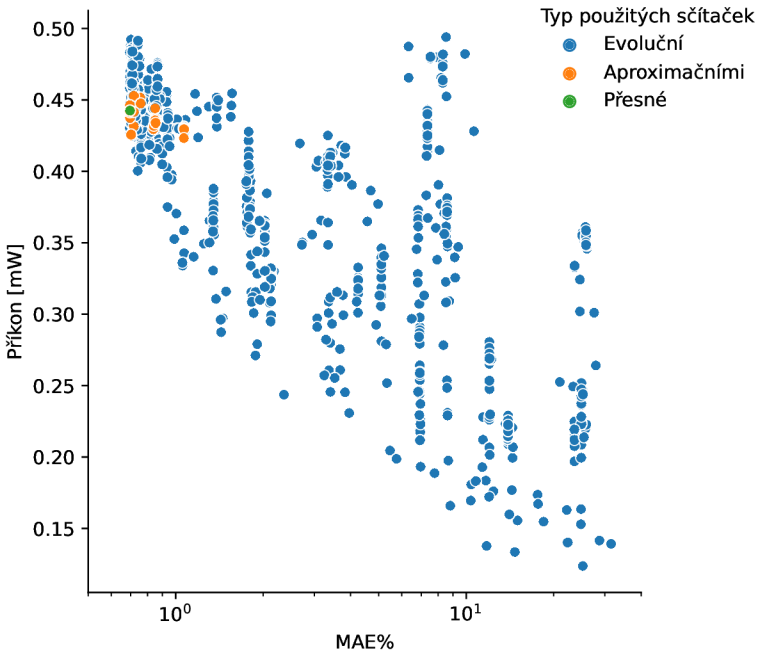


Obrázek 7.2: Porovnání zpoždění a průměrné aritmetické chyby ALM násobiček s přesnými a aproximačními sčítačkami z literatury a násobiček s evolučními sčítačkami.

V grafu 7.2 lze vidět, že v některých případech dosahovaly násobičky s geneticky upravenými sčítačkami lepších výsledků než s ručně navrženými aproximačními sčítačkami. Vyšší zpoždění způsobuje ta část aproximačních sčítaček LOA, MAA3 a SOA, která je realizována přesnou sčítačkou. Přesná sčítačka má dlouhou kritickou cestu a tím i větší zpoždění. Některé evolučně optimalizované sčítačky nepočítají s přenosem skrz celou komponentu a tím je jejich kritická cesta kratší.

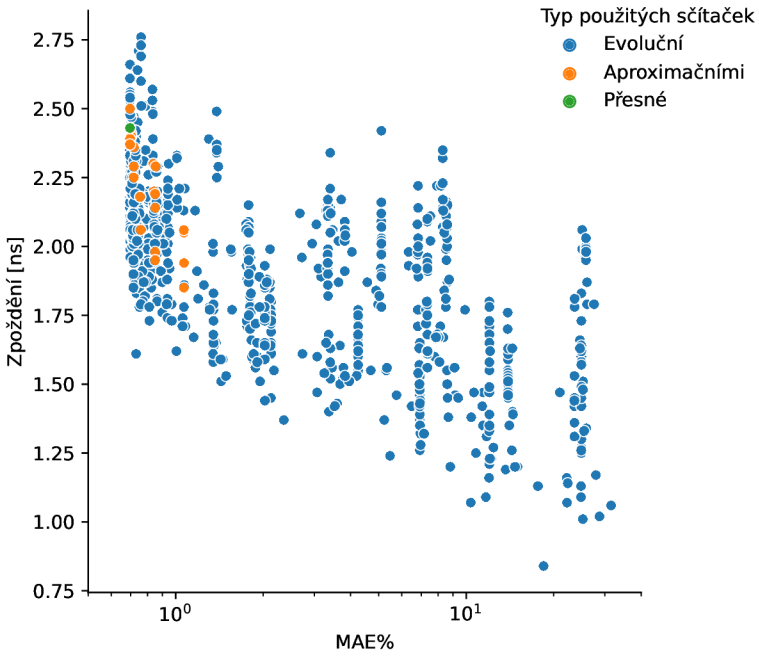
7.2 ILM využívající evoluční sčítačky

V násobičkách ILM byly nahrazeny dvě sčítačky evolučně navrženými aproximacemi. Z EvoApproxLib bylo vybráno 30 realizací osmibitových sčítaček, které byly v obvodech násobiček nakombinovány. Vzniklo tak 900 různých násobiček. Dále bylo vytvořeno 25 násobiček s kombinacemi různě přesných ručně vytvořených sčítaček. V následujících grafech jsou vykresleny společně s výsledky původního obvodu ILM.



Obrázek 7.3: Porovnání zpoždění a průměrné aritmetické chyby ILM násobiček s přesnými a aproximačními sčítačkami z literatury a násobiček s evolučními sčítačkami.

Graf 7.3 vykreslující závislost příkonu logaritmických násobiček na jejich průměrné aritmetické chybě ukazuje, že na rozdíl od ALM nebylo použitím aproximačních sčítaček dosaženo snížení chybovosti. Násobičky s nepřesnými sčítačkami popsanými v části 3.2.1 nemají v hodnotách tak velký rozptyl jako při použití evolučních sčítaček. Díky velkému množství vytvořených variant ILM obvodů byly získány násobičky s novými vlastnostmi. Zvláště v oblasti s průměrnou aritmetickou chybou větší než jedno procento došlo k výraznějšímu snížení příkonu. Zároveň byly objeveny konstrukce dominující násobičky z literatury, kde jsou využity přibližné sčítačky navržené ručně. Přestože změna byla provedena pouze u sčítaček, které tvoří jen část spotřeby energie, došlo k výraznému snížení celkového příkonu. To je způsobeno aproximací některých bitů sčítaček napojením na konstantní hodnoty. Nedochází tak k propojení některých předcházejících částí s výstupem. Protože tyto části tak nemají vliv na výsledek, jsou při syntéze ignorovány. Změnou ve sčítačce tak může dojít i k celkové změně obvodu.

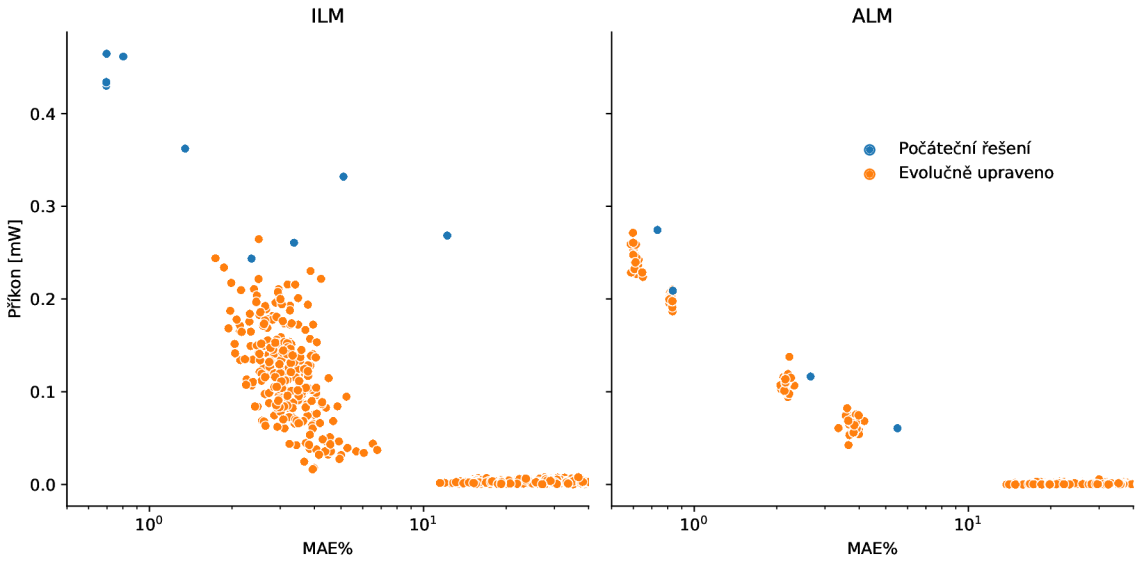


Obrázek 7.4: Porovnání zpoždění a průměrné aritmetické chyby ILM násobiček s přesnými a aproximačními sčítačkami z literatury a násobiček s evolučními sčítačkami.

Při srovnání hodnot MAE a Zpoždění z grafu 7.4 je patrné, že došlo k vytvoření velkého množství násobiček jejichž hodnota průměrné aritmetické chyby se pohybuje pod jedním procentem a zároveň dosáhly výrazně nižšího zpoždění.

7.3 Evoluční aproximace logaritmických násobiček

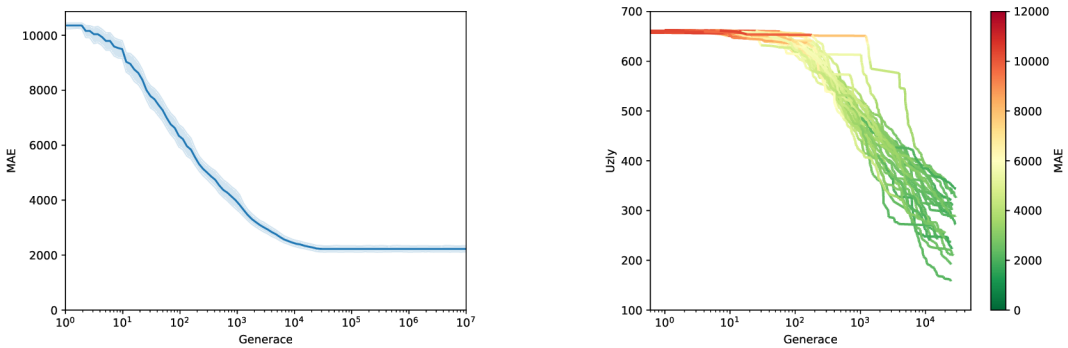
Za účelem dalšího vylepšení byla provedena úprava násobiček evolučně pomocí CGP. Ze všech vytvořených logaritmických násobiček s různými typy sčítaček byla vybrána nedominovaná řešení z pohledu příkonu versus MAE a zpoždění versus MAE. K další úpravě z nich bylo určeno 13 obvodů s různými hodnotami parametrů, aby se nevycházelo z více podobných konstrukcí. Pro ALM i ILM byl výběr proveden zvlášť, protože ALM byly ve většině případů dominující ILM.



Obrázek 7.5: Porovnání příkonu a průměrné aritmetické chyby vybraných násobiček před a po evoluční úpravě

Hodnoty evolučně upravených a ručně vytvořených násobiček jsou znázorněny v grafu 7.5. Hodnoty chybovosti a příkonu násobiček vytváří skupiny obvodů s podobnými vlastnostmi. Jak je z grafu patrné bylo dosaženo celkového zlepšení v poměru příkonu vůči chybovosti.

Pro každý z vybraných obvodů bylo spuštěno třicet běhů, při kterých docházelo k různě výrazným změnám mezi generacemi. Křivka v grafu 7.6a ukazuje jak se v průběhu měnil počet uzlů a chybovost jednoho obvodu. Tmavě vynesené hodnoty jsou průměry ze všech třiceti běhů a světlý pruh 95% interval spolehlivosti spočítaný z distribuce. Je patrné, že mezi prvními generacemi jsou skokové rozdíly. Hodnoty parametrů v čase konvergují a obvody už se tak výrazně nemění. V grafu 7.6b křivky znázorňují jednotlivé běhy. Hodnota MAE pro jednotlivé generace každého běhu je v tomto případě znázorněna barvou. Díky tomu je jasně vidět, že se během evoluce snižovalo množství uzlů, ale současně klesala i chybovost.

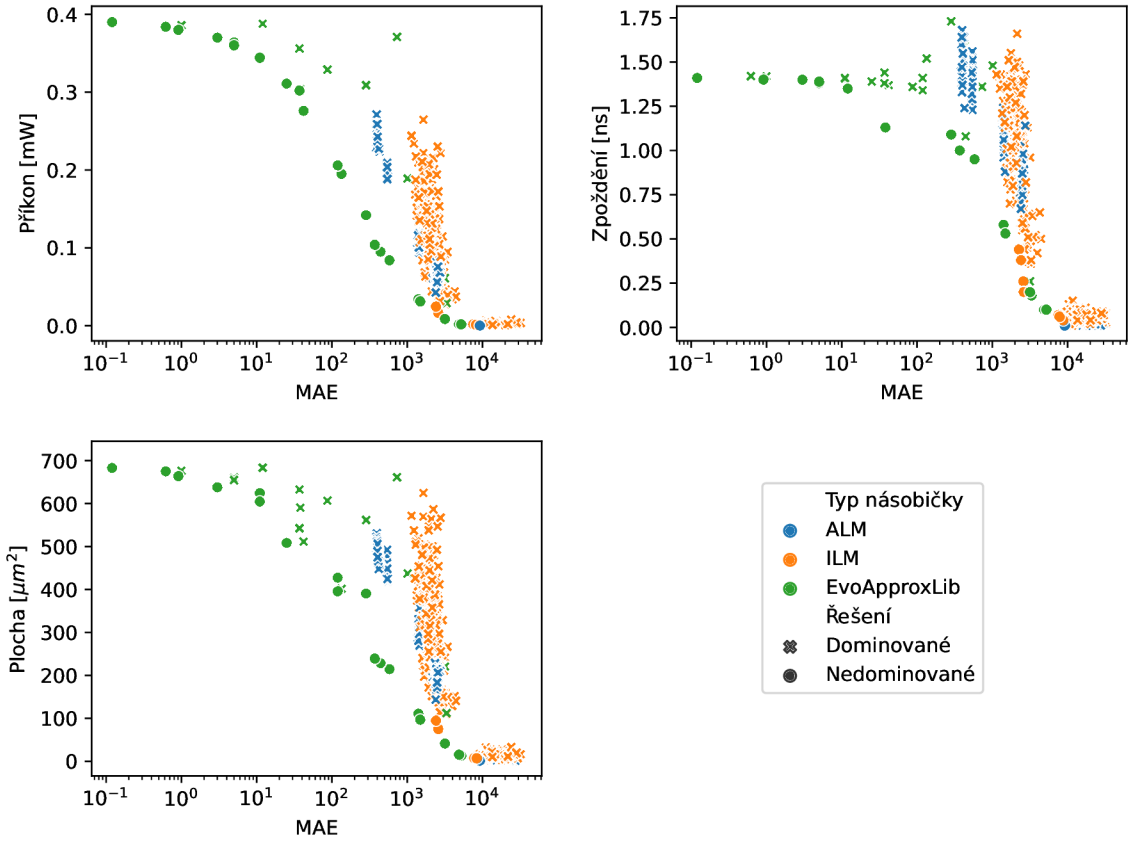


(a) MAE (b) Uzly

Obrázek 7.6: Konvergence evolučního návrhu

7.4 Pareto optimální násobičky

Z nově vytvořených násobiček byla vybrána Pareto optimální řešení. Podařilo se vytvořit celkem 11 násobiček které nejsou dominovány EvoApproxLib obvody v MAE vůči příkonu, zpoždění nebo ploše. Všechna tato řešení mají poměrně vysokou chybovost. To je způsobeno modifikací velmi nepřesných počátečních řešení.



Obrázek 7.7: Porovnání mae a hardwarových parametrů

Kapitola 8

Závěr

Cílem této práce bylo nastudovat možnosti konstrukce aritmetických obvodů, prozkoumat vylepšení logaritmických násobiček a navrhnout jejich vylepšení. V rámci této práce jsem v nástroji ArithsGen naimplementovala několik obvodů z literatury. Díky tomu jsou násobičky dostupné dalším výzkumníkům k novému vylepšení, nebo k využití v komplexnějších konstrukcích. Dále byly dostupné implementace aproximačních obvodů převedeny taktéž do ArithsGen formátu a použity jako části logaritmických násobiček.

Vyhodnocením parametrů nových obvodů byla objevena řešení s lepším poměrem chybovosti a příkonu, než mají některé stávají logaritmické násobičky využívající ručně navržené aproximační sčítačky.

Dalšího vylepšení bylo dosaženo úpravou celých konstrukcí násobiček evoluční metodou. Po vyhodnocení byly nalezeny obvody, které se svými parametry přibližují řešením z EvoApproxLib. Nicméně EvoApproxLib násobičky zůstávají ve většině případů efektivnější. Výsledkem je nalezení nových variant doplňujících Pareto frontu. Dalším přínosem této práce je rozšíření knihovny obvodů o implementace logaritmických násobiček, které lze dále využít v nástroji ArithsGen samostatně nebo jako součást komplexnějších konstrukcí.

Další prostor pro zlepšení vytváří posouvače. Svou velikostí představují podstatnou energetickou zátěž logaritmických násobiček, přičemž některé části jejich konstrukce ovlivňují výsledek minimálně. Zajímavým řešením by mohl být i automatický evoluční návrh sčítaček určených pro konkrétní typy násobiček. V rámci této práce bylo otestováno použití sčítaček navržených pro obecné použití. Ale vzhledem k vlastnostem logaritmických násobiček Mitchellova typu, je nejvhodnější použití nadhodnocujících sčítaček. Proto se jeví jako perspektivní v průběhu evoluce testovat vlastnosti obvodů zapojených přímo v násobičkách a zkoumat jejich dopad na výsledek celé násobičky. Tento přístup by se mohl využít i při návrhu optimalizovaných posouvačů.

Literatura

[1] Ansari, M. S., Cockburn, B. F. a Han, J. A Hardware-Efficient Logarithmic Multiplier with Improved Accuracy. In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2019, s. 928–931. DOI: 10.23919/DATE.2019.8714868. ISBN 978-3-9819263-2-3.

[2] Ansari, M. S., Cockburn, B. F. a Han, J. An Improved Logarithmic Multiplier for Energy-Efficient Neural Computing. *IEEE Transactions on Computers*. 2021, sv. 70, č. 4, s. 614–625. DOI: 10.1109/TC.2020.2992113.

[3] Arunachalam, K. Design and Implementation of A Reversible Logic based 8-Bit Arithmetic and Logic Unit. *International Journal of Computers and Applications*. Červenec 2014, sv. 36, s. 49–55.

[4] Babic, Z., Avramovic, A. a Bulić, P. An iterative logarithmic multiplier. *Microprocessors and Microsystems*. Únor 2011, sv. 35, s. 23–33. DOI: 10.1016/j.micpro.2010.07.001.

[5] Goswami, S. S. P., Paul, B., Dutt, S. a Trivedi, G. Comparative Review of Approximate Multipliers. In: *2020 30th International Conference Radioelektronika (RADIOELEKTRONIKA)*. 2020, s. 1–6. DOI: 10.1109/RADIOELEKTRONIKA49387.2020.9092370.

[6] Hafiz, R., Shafique, M. a El Harouni, W. *LpACLib: An Open-Source Library for Low-Power Approximate Computing Modules*. Oct 2016. Dostupné z: https://sourceforge.net/projects/lpaclib/.

[7] Hanif, M. A., Hafiz, R., Hasan, O. a Shafique, M. QuAd: Design and analysis of Quality-area optimal Low-Latency approximate Adders. In: *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 2017, s. 1–6. DOI: 10.1145/3061639.3062306.

[8] Jain, J., Cauley, S., Koh, C.-K. a Balakrishnan, V. SASIMI: sparsity-aware simulation of interconnect-dominated circuits with nonlinear devices. In: *Asia and South Pacific Conference on Design Automation, 2006.* 2006, s. 6 pp.–. DOI: 10.1109/ASPDAC.2006.1594719.

[9] Jiang, H., Liu, C., Liu, L., Lombardi, F. a Han, J. A Review, Classification, and Comparative Evaluation of Approximate Arithmetic Circuits. *ACM Journal on Emerging Technologies in Computing Systems*. Srpen 2017, sv. 13, s. 1–34. DOI: 10.1145/3094124.

[10] Jou, J. M., Kuang, S. R. a Chen, R. D. Design of low-error fixed-width multipliers for DSP applications. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*. 1999, sv. 46, č. 6, s. 836–842. DOI: 10.1109/82.769795.

[11] Kim, M. S., Barrio, A. A. D., Oliveira, L. T., Hermida, R. a Bagherzadeh, N. Efficient Mitchell's Approximate Log Multipliers for Convolutional Neural Networks. *IEEE Transactions on Computers*. 2019, sv. 68, č. 5, s. 660–675. DOI: 10.1109/TC.2018.2880742.

[12] Klhůfek, J. a Mrázek, V. ArithsGen: Arithmetic Circuit Generator for Hardware Accelerators. In: *2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS '22)* [online]. Institute of Electrical and Electronics Engineers: Institute of Electrical and Electronics Engineers, April 2022, kap. 176991, s. 44–47. DOI: 10.1109/DDECS54261.2022.9770152.

[13] Kulkarni, P., Gupta, P. a Ercegovac, M. Trading Accuracy for Power in a Multiplier Architecture. *J. Low Power Electronics*. Prosinec 2011, sv. 7, s. 490–501. DOI: 10.1166/jolpe.2011.1157.

[14] Liu, W., Xu, J., Wang, D. a Lombardi, F. Design of Approximate Logarithmic Multipliers. In: *Proceedings of the on Great Lakes Symposium on VLSI 2017*. New York, NY, USA: Association for Computing Machinery, 2017, s. 47–52. GLSVLSI '17. DOI: 10.1145/3060403.3060409. ISBN 9781450349727. Dostupné z: https://doi.org/10.1145/3060403.3060409.

[15] Mahdiani, H. R., Ahmadi, A., Fakhraie, S. M. a Lucas, C. Bio-Inspired Imprecise Computational Blocks for Efficient VLSI Implementation of Soft-Computing Applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*. 2010, sv. 57, č. 4, s. 850–862. DOI: 10.1109/TCSI.2009.2027626.

[16] Mitchell, J. N. Computer Multiplication and Division Using Binary Logarithms. *IRE Transactions on Electronic Computers*. 1962, EC-11, č. 4, s. 512–517. DOI: 10.1109/TEC.1962.5219391.

[17] Mrazek, V., Hrbacek, R., Vasicek, Z. a Sekanina, L. EvoApprox8b: Library of Approximate Adders and Multipliers for Circuit Design and Benchmarking of Approximation Methods. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. 2017, s. 258–261. DOI: 10.23919/DATE.2017.7926993.

[18] Nandan, D. AN EFFICIENT ARCHITECTURE OF LEADING ONE DETECTOR. *International Journal of Pure and Applied Mathematics*. Únor 2018, sv. 118.

[19] Nepal, K., Li, Y., Bahar, R. I. a Reda, S. ABACUS: A technique for automated behavioral synthesis of approximate computing circuits. In: *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2014, s. 1–6. DOI: 10.7873/DATE.2014.374.

[20] Sarwar, S. S., Venkataramani, S., Ankit, A., Raghunathan, A. a Roy, K. Energy-Efficient Neural Computing with Approximate Multipliers. *J. Emerg. Technol. Comput. Syst.* New York, NY, USA: Association for Computing Machinery.

jul 2018, sv. 14, č. 2. DOI: 10.1145/3097264. ISSN 1550-4832. Dostupné z: https://doi.org/10.1145/3097264.

[21] Sekanina, L., Růžička, R., Bidlo, M., Jaroš, J., Švenda, P. et al. *Evoluční hardware*. Gerstner. Praha: Academia, 2009. ISBN 978-80-200-1729-1.

[22] Shalf, J. The future of computing beyond Moore's Law. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*. Březen 2020, sv. 378, s. 20190061. DOI: 10.1098/rsta.2019.0061.

[23] Sullivan, M. B. a Swartzlander, E. E. Truncated Logarithmic Approximation. In: *2013 IEEE 21st Symposium on Computer Arithmetic*. 2013, s. 191–198. DOI: 10.1109/ARITH.2013.34.

[24] Venkataramani, S., Sabne, A., Kozhikkottu, V., Roy, K. a Raghunathan, A. SALSA: Systematic Logic Synthesis of Approximate Circuits. In: *Proceedings of the 49th Annual Design Automation Conference*. New York, NY, USA: Association for Computing Machinery, 2012, s. 796–801. DAC '12. DOI: 10.1145/2228360.2228504. ISBN 9781450311991. Dostupné z: https://doi.org/10.1145/2228360.2228504.

[25] Verma, A. K., Brisk, P. a Ienne, P. Variable Latency Speculative Addition: A New Paradigm for Arithmetic Circuit Design. In: *Proceedings of the Conference on Design, Automation and Test in Europe*. New York, NY, USA: Association for Computing Machinery, 2008, s. 1250–1255. DATE '08. DOI: 10.1145/1403375.1403679. ISBN 9783981080131. Dostupné z: https://doi.org/10.1145/1403375.1403679.