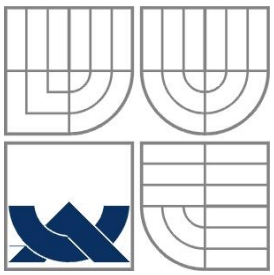
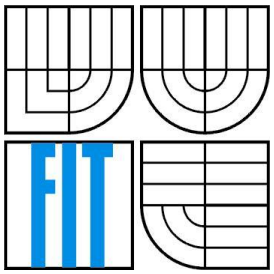


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

GENERÁTOR ARITMETICKÝCH OBVODŮ

ARITHMETIC CIRCUIT GENERATOR

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MICHAL BOLJEŠIK
AUTHOR

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

BRNO 2015

Abstrakt

Cílem této práce je návrh a implementace nástroje, který umožňuje generovat popis různých variant aritmetických obvodů (jako jsou sčítačky a násobičky), které tvoří součást složitějších systémů (filtrů, transformací apod.).
První část práce se věnuje rozboru různých variant sčítaček a násobiček jak z pohledu teoretického, tak praktického. V druhé části lze nalézt popis návrhu a implementaci parametrizovatelného nástroje vytvořeného v jazyce Python, který dovoluje generovat hierarchický i tzv. flattened popis různých obvodů ve formátech určených pro vizualizaci, simulaci i ověření korektní funkce. V závěru je nástroj využit k porovnání různých implementací sčítaček a násobiček.

Abstract

The goal of this thesis is to design and implement a tool that would be able to generate a description of various types of arithmetic circuits, such as adders and multipliers, that are involved in more complex systems (filters, transformations, etc.).
The first part of the thesis deals with analysis of different types of adders and multipliers on either theoretical or practical level. In the second part there is a description of the design and implementation of the tool created in Python language. On base of parameters, the tool is able to generate hierarchical or flattened description of various circuits in formats aimed for visualization, simulation and validation. In the end, the tool is used to compare different designs of adders and multipliers.

Klíčová slova

hradlo, aritmetický obvod, sčítačka, násobička, generátor, Python, C, SpiceVision

Keywords

gate, arithmetic circuit, adder, multiplier, generator, Python, C, SpiceVision

Citace

Bolješik Michal: Generátor aritmetických obvodů, bakalářská práce, Brno, FIT VUT v Brně, 2015

Generátor aritmetických obvodů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Zdeňka Vašíčka, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Michal Bolješik
20. května 2015

Poděkování

Rád bych se poděkoval vedoucímu práce, panu Ing. Zdeňkovi Vašíčkovi, Ph.D., za jeho odbornou pomoc, užitečné rady a vedení při tvorbě této bakalářské práce.

Obsah

1 Úvod

Aritmetické obvody predstavujú základný stavebný prvok zložitejších zapojení, akými sú napríklad filter s konečnou impulzovou odozvou (Finite Impulse Response filter) alebo rýchla Fourierova transformácia (FFT – Fast Fourier Transformation). V dnešnej dobe sme obklopení takmer všadeprítomnými vstavanými systémami, ktorých súčasťou môžu byť aj spomenuté zapojenia. Podstatnými faktormi pri takýchto systémoch sú, okrem iných, aj spotreba, výkon, rýchlosť či potrebná plocha. A to najmä v prípade, ak sú napájané v batérie. Vzťah medzi pomermi uvedených vlastností môžeme hľadať už na úrovni aritmetických obvodov. Práve zapojenie je tým faktorom, ktorý ich ovplyvňuje. V literatúre možno nájsť mnoho typov zapojení pre sčítačky a násobičky. Niektoré sa vyznačujú rýchlosťou výpočtu na úkor potrebnej plochy, iné zasa uprednostňujú úsporu plochy pred časom potrebným na dosiahnutie výsledku a napokon sú aj také, ktoré reprezentujú kompromis medzi týmito faktormi. Pre konkrétnu reprezentáciu obvodu sa však pomer týchto vlastností môže pri rôznych šírkach vstupných signálov aj odlišovať. Pri tvorbe výsledného produktu teda máme prostredníctvom vhodne zvolených stavebných blokov možnosť určiť, ktorým smerom sa bude tento produkt z pohľadu parametrov uberať.

Cieľom tejto práce je návrh a implementácia nástroja, ktorý bude generovať schémy zapojenia aritmetických obvodov v rôznej forme, pričom základným stavebným prvkom týchto obvodov majú byť hradlá. Prostredníctvom takto vygenerovaných obvodov bude potom možné zistiť, porovnať a vyhodnotiť parametre jednotlivých variantov zapojení. Pre tvorbu takéhoto nástroja je potrebné preskúmať jednotlivé druhy aritmetických obvodov a ich vlastnosti. Vo veľkej miere sa od nich bude odvíjať samotný návrh. Implementácia bude potom realizovaná v jazyku Python. Po nej by bolo vhodné overiť najmä to, či naozaj pracuje podľa očakávaných predpokladov, teda či ním generované obvody vykonávajú funkciu, na ktorú sú určené.

Text práce pozostáva z nasledujúcich logických celkov. Kapitola 2 ponúka prehľad vybraných typov sčítačiek a násobičiek. Prvá časť kapitoly 3 popisuje návrh generátora aritmetických obvodov, druhá časť zasa jeho implementáciu. V kapitole 4 je spomenutý spôsob jeho testovania a vyhodnotenie jeho časových vlastností a v kapitole 5 sa nachádzajú porovnania jednotlivých vygenerovaných aritmetických obvodov na základe ich parametrov.

2 Základné aritmetické obvody a ich vlastnosti

2.1 Elementárne stavebné bloky a ich parametre

Na úvod kapitoly si uveďme oneskorenie Δ_T konkrétnych hradiel a rozsah plochy A_T (podľa anglického *area* – „plocha“), ktorú zaberajú. Tieto údaje využijeme pri porovnaní jednotlivých, nižšie uvedených, obvodov. Referenčnými pre nás budú hradlá NAND a NOR, ktoré majú zhodné nielen oneskorenie vplyvom šírenia Δ_g (čo bude jednotka pre veličinu oneskorenia Δ_T), ale aj plochu potrebnú na čipe A_g (jednotka pre rozsah plochy A_T). Vyššie spomenuté hodnoty sú uvedené v Tabuľke 2.1.

Hradlo	$\mathbf{\Delta_T}$	$\mathbf{A_T}$
NAND	Δ_g	A_g
NOR	Δ_g	A_g
NOT	Δ_g	A_g
AND	$2\Delta_g$	$2A_g$
OR	$2\Delta_g$	$2A_g$
XOR	$2\Delta_g$	$3A_g$
XNOR	$2\Delta_g$	$3A_g$
Buffer	Δ_g	A_g
Multiplexor 2-1	$5\Delta_g$	$7A_g$

Tabuľka 2.1: Oneskorenie stavebných prvkov obvodu a plocha, ktorú zaberajú na čipe (údaje prevzaté z [1])

2.2 Sčítačky

Sčítačky uvediem v tejto kapitole ako prvé. Je to z toho dôvodu, že násobenie je založené na sčítaní. Sčítačky sú teda základným prvkom násobičiek.

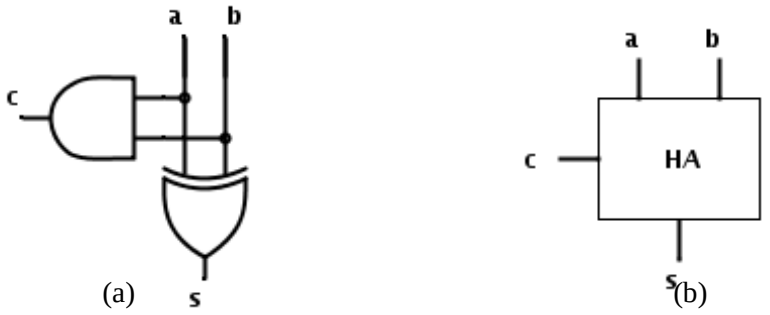
2.2.1 Jednobitová polovičná sčítačka (Single-Bit Half-Adder)

Jednobitová polovičná sčítačka má rovnako dva vstupy ako aj výstupy. Vstupy tvoria dva jednobitové operandy a výstupný signál možno rozčleniť na tzv. súčet *s* a prenos *c*. Vzťahy pre ich výpočet uvádzajú Rovnica (2.1) a Rovnica (2.2).

$$s = a \oplus b \tag{2.1}$$

$$c = ab \tag{2.2}$$

Ide o základný komponent realizovaný priamo na úrovni hradiel, a to použitím jedného hradla XOR a jedného hradla AND. To znamená, že oneskorenie komponentu Δ_T je $2\,\Delta_g$ a plocha A_T, ktorú jednobitová polovičná sčítačka zaberá je $5\,A_g$. Schéma sčítačky je znázornená na Obrázku 2.1 (a).



Obrázok 2.1: Jednobitová polovičná sčítačka reprezentovaná: (a) na úrovni hradiel a (b) blokovým diagramom

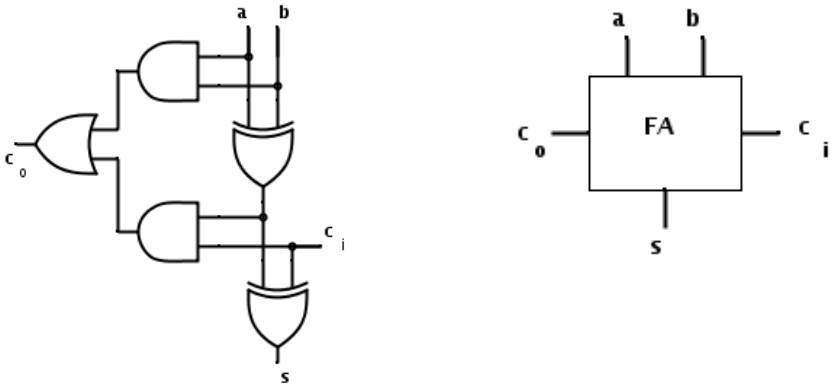
2.2.2 Jednobitová úplná sčítačka (Single-Bit Full-Adder)

V porovnaní s jednobitovou polovičnou sčítačkou má jednobitová úplná sčítačka o jeden vstupný operand navyše. Ide o prenos z nižšieho rádu a označíme ho ako c_i (podľa anglického *carry-in* – „vstupný prenos“). Počtom výstupných bitov sa už tento komponent od jednobitovej polovičnej sčítačky neodlišuje, no v tomto prípade označíme výstupný bit prenosu ako c_o (podľa anglického *carry-out* – „výstupný prenos“). Ten je výsledkom funkcie uvedenej v Rovnici (2.3). Výstupný bit *s* je, podobne ako pri jednobitovej polovičnej sčítačke, výsledkom exkluzívneho súčtu vstupných bitov, viď Rovnica (2.4).

$$c_o = ab + bc + ac \tag{2.3}$$

$$s = a \oplus b \oplus c_i \tag{2.4}$$

Jednobitovú úplnú sčítačku tvoria dve hradlá XOR, dve hradlá AND a jedno hradlo OR. Oneskorenie Δ_T tejto sčítačky je teda $6\,\Delta_g$ a plocha A_T, ktorú komponent potrebuje na čipe je $12\,A_g$. Jej schéma je na Obrázku 2.2 (a) a označenie, ktoré pre ňu budeme používať od tohto miesta na Obrázku 2.2 (b).

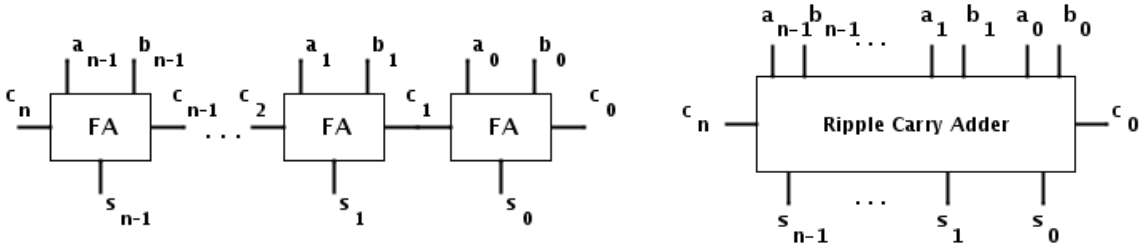


(a) (b)

Obrázok 2.2: Jednobitová úplná sčítačka reprezentovaná: (a) na úrovni hradiel a (b) blokovým diagramom

2.2.3 Paralelná sčítačka s postupným prenosom (Riple Carry Adder)

Paralelnou sčítačkou s postupným prenosom nazývame obvod tvorený jednobitovými úplnými sčítačkami. Vo všeobecnosti možno tvrdiť, že *n*-bitová paralelná sčítačka s postupným prenosom sa skladá z presne *n* jednobitových úplných sčítačiek a sčítava dve *n*-bitové čísla. Z jej schémy na Obrázku 2.3 je pozorovateľný fakt, že vstupný prenos jednobitovej úplnej sčítačky, ktorá sčítava bity na *i*-tej pozícii, je výstupným prenosom rovnakej sčítačky sčítavajúcej bity na pozícii *i-1*, pre *i = {1, 2, ..., n-1}*. Hodnoty na výstupoch jednobitovej úplnej sčítačky na pozícii *i* sú teda platné až v dobe, keď je dokončený výpočet sčítačky na pozícii *i-1* [2]. Inak povedané, bit prenosu „postupuje“ (vo voľnom preklade anglického *ripple carry*) od jednobitovej úplnej ščítačky na pozícii najmenej významných vstupných bitov (LSB) až k tej na pozícií najvýznamnejších bitov (MSB). To je dôvod, prečo má paralelná sčitačka s postupným prenosom relatívne vysoké oneskorenie (v porovnaní s nižšie uvedenými obvodmi realizujúcimi sčítanie).



(a) (b)

Obrázok 2.3: Paralelná *n*-bitová sčítačka s postupným prenosom reprezentovaná: (a) prostredníctvom jednobitových úplných sčítačiek a (b) blokovým diagramom

Odvodené od parametrov jednobitovej úplnej sčítačky, oneskorenie Δ_T *n*-bitovej paralelnej sčítačky je rovné hodnote $(4n+2)\,\Delta_g$ a plocha A_T , ktorú takáto sčítačka zaberá je $12n\,A_g$, kde *n* je už vyššie spomenutý počet bitov sčítancov.

2.2.4 Sčítačka s predikciou prenosu (Carry-Lookahead Adder)

N-bitovou sčítačkou s predikciou prenosu rozumieme také zapojenie hradiel, v ktorom sa výstupné prenosy jednotlivých pozícií súčtu generujú paralelne, nie postupne od najmenej významného bitu ako u sčítačky s postupným prenosom. Minimalizuje sa teda oneskorenie, ktoré by bolo spôsobené práve šírením prenosu.

Paralelné generovanie prenosu je realizované prostredníctvom výpočtu dvoch zásadných bitov vo vnútornom zapojení obvodu. Ide o bit g_i a p_i, kde

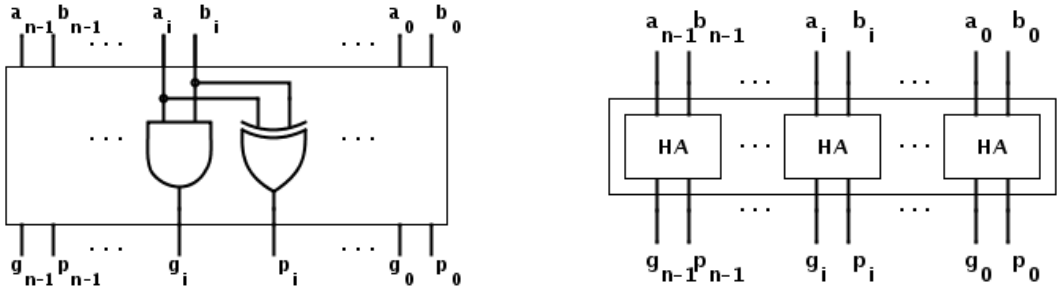
$$g_i = a_i b_i \,, \tag{2.5}$$

$$p_i = a_i \oplus b_i \,. \tag{2.6}$$

Bit g_i (podľa anglického *carry generation* – „generovanie prenosu“) vyjadruje skutočnosť, či na základe hodnôt bitov sčítancov *a* a *b* na pozícii *i* vznikne prenos do vyššieho rádu. Takáto situácia nastane v prípade, ak sú obe tieto hodnoty na úrovni logickej „1“, z čoho vyplýva, že funkcia je výsledkom logického súčinu jej vstupov.

Výstižným pre bit p_i je jeho anglický popis, teda *carry propagation*. Voľne ho preložme ako „šírenie vstupného prenosu“. Hodnota výstupného prenosu pozície *i* bude zhodná s hodnotou toho vstupného iba za podmienky, že bity a_i a b_i budú rozdielne (čo odpovedá logickej funkcii XOR). V takejto situácii nadobúda p_i hodnotu logickej „1“ (pre ilustráciu – ak by mali a_i a b_i zhodne hodnotu „0“, výstupný prenos by bol, bez ohľadu na vstupný, taktiež v hodnote logickej „0“; ak by boli a_i a b_i obe rovné „1“, výstupný prenos by bol vždy „1“).

Zhrnutím dvoch predošlých odsekov dospievame k poznatku, že hodnoty bitov g_i a p_i na príslušnej pozícii *i* sú vypočítané v momente, keď sú známe sčítance *a* a *b*. Sú tiež nezávislé na výstupnom prenose predchádzajúceho rádu, ako možno postrehnúť z Rovnice (2.5), resp. Rovnice (2.6). Táto časť sčítačky je zobrazená na Obrázku 2.4.



(a) (b)

Obrázok 2.4: Časť *n*-bitovej sčítačky s predikciou prenosu generujúca a šíriaca prenos: (a) reprezentácia na úrovni hradiel a (b) reprezentácia pomocou jednobitovej polovičnej sčítačky

Výstupný prenos rádu *i* možno všeobecne vyjadriť vzorcom

$$c_{i+1} = a_i b_i + (a_i \oplus b_i) c_i = g_i + p_i c_i \, . \tag{2.7}$$

Z neho si vyjadrime konkrétne výstupné prenosy postupne pre $i = \{1, 2, 3\}$:

$$c_1 = g_0 + c_0 p_0 \, , \tag{2.8}$$

$$c_2 = g_1 + c_1 p_1 \, , \tag{2.9}$$

$$c_3 = g_2 + c_2 p_2 \, . \tag{2.10}$$

Tieto tri predchádzajúce rovnice možno ešte rekurzívne rozviesť, a to tak, že do Rovnice (2.9) dosadíme Rovnicu (2.8). Obdobný postup zvolíme aj pri Rovnici (2.10). Výsledok bude nasledovný:

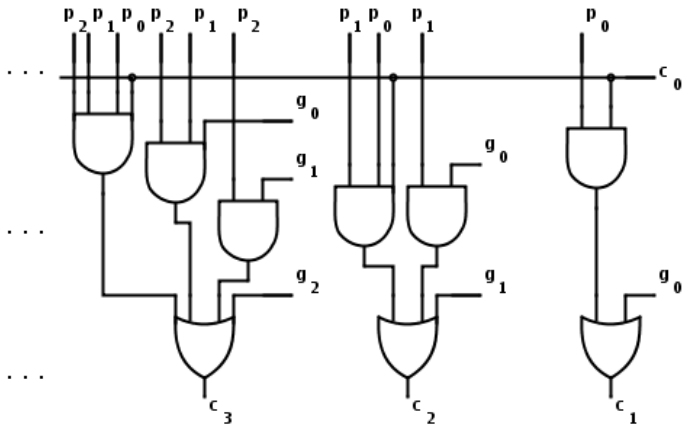
$$c_2 = g_1 + g_0 p_1 + c_0 p_0 \, p_1 \, , \tag{2.11}$$

$$c_3 = g_2 + g_1 p_2 + g_0 p_1 p_2 + c_0 p_0 \, p_1 p_2 \, . \tag{2.12}$$

Všeobecný rekurzívny tvar rovnice pre prenos c_{i+1} , odvodený z Rovnice (2.11) a Rovnice (2.12), môže mať podobu

$$c_{i+1} = g_i + g_{i-1} p_i + g_{i-2} p_{i-1} p_i + \cdots + g_0 p_1 p_2 \ldots p_i + c_0 p_0 \, p_1 \ldots p_i \, , \tag{2.13}$$

z ktorej je vyvoditeľný fakt, že hodnota prenosu c_{i+1} je plne nezávislá od prenosu z nižšieho rádu, teda od hodnoty bitu c_i. Generovaná je prostredníctvom bitov g_i, p_i a c_0 a popisuje časť sčítačky, ktorú (s použitím *n*-vstupových hradiel) znázorňuje Obrázok 2.5.



Obrázok 2.5: Časť *n*-bitovej sčítačky s predikciou prenosu paralelne generujúca prenosy

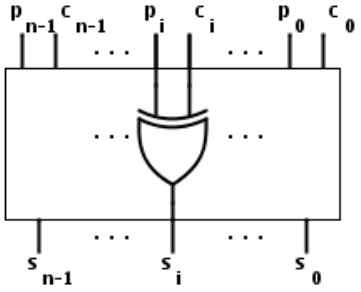
Pre konkrétny bit súčtu s_i tejto sčítačky platí funkcia:

$$s_i = a_i \oplus b_i \oplus c_i \,, \tag{2.14}$$

z ktorej po dosadení Rovnice (2.6) dostaneme, že

$$s_i = p_i \oplus c_i \,. \tag{2.15}$$

Výsledok súčtu *s* na pozícii *i* je teda známy v momente, keď je vypočítaný vstupný prenos c_i tejto pozície. Generuje ho sumačná časť sčítačky práve podľa Rovnice (2.15). Popisuje ju Obrázok 2.6.



Obrázok 2.6: Sumačná časť *n*-bitovej sčítačky s predikciou prenosu

Z troch vyššie popísaných častí sčítačky s predikciou prenosu môžeme dospieť k hodnote jej oneskorenia Δ_T:

$$\Delta_T = \Delta_{generation/propagation\ unit} + \Delta_{parallel\ carries\ unit} + \Delta_{summation\ unit} = \\ = 2\Delta_g + [2\log_2 n]\,2\Delta_g + 2\Delta_g \,. \tag{2.16}$$

Čo sa týka plochy A_T, ktorú zaberá *n*-bitová sčítačka s predikciou prenosu, taktiež ju odvodíme na základe plôch jednotlivých jej častí:

$$\mathrm{A}_T = \mathrm{A}_{generation/propagation\ unit} + \mathrm{A}_{parallel\ carries\ unit} + \mathrm{A}_{summation\ unit} = \\ = 5n\mathrm{A}_g + \left(\sum_{i=1}^{n}[\frac{1}{2}i(i+1)] + \frac{1}{2}n(n+1)\right)2\mathrm{A}_g + 3n\mathrm{A}_g \qquad (2.17)$$

2.2.5 Sčítačka s výberom prenosu (Carry-Select Adder)

V sčítačke s výberom prenosu dochádza, podobne ako v sčítačke s predikciou prenosu, k eliminácii oneskorenia zapríčineného šírením prenosu. V tomto prípade je ale postup docielenia tohto kroku odlišný.

Ako popisuje [3], sčítačka s výberom prenosu je rozdelená na viac pomyselných sektorov. V každom, okrem toho, ktorý obsahuje najmenej významný bit, sa generujú paralelne dva súčty a príslušné prenosy do vyššieho rádu. Dva z dôvodu toho, že prenos z nižšieho rádu sa môže i nemusí vyskytnúť. Alternatívy čiastočných súčtov a prenosov sú už teda nezávisle vypočítané v čase, keď signál prenosu z nižšieho rádu nadobúda platnú hodnotu. Na jej základe vystavia multiplexory na svoje výstupy zodpovedajúci súčet príslušnej časti sčítačky a taktiež prenos do vyššieho rádu.

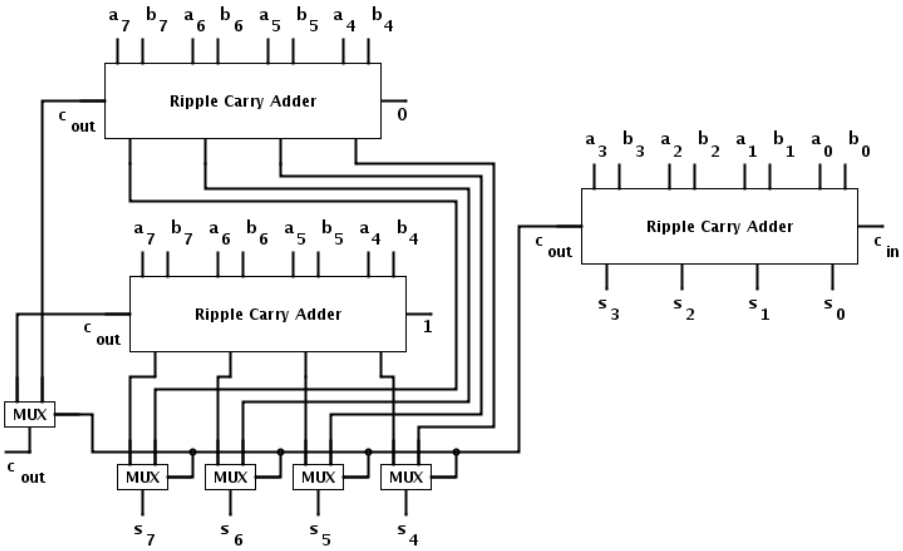
Ako komponenty vykonávajúce súčty sú v tejto logike použité paralelné sčítačky s postupným prenosom, pričom hodnota ich šírky je zhodná s hodnotou šírky vyššie spomenutých sektorov. V tomto prípade zvoľme šírku 4 bity. V obvode sa ešte vyskytujú už zmienené multiplexory. Schéma sčítačky s výberom prenosu je na Obrázku 2.7.

Sčítačka s výberom prenosu bude mať oneskorenie Δ_T vypočítané podľa nasledujúceho vzťahu:

$$\Delta_T = \Delta_{Ripple\ Carry\ Adder} + \Delta_{multiplexers} = \\ = (4m+2)\Delta_g + \left(\frac{n}{m} - 1\right)5\Delta_g\ , \qquad (2.18)$$

kde *n* udáva šírku celej sčítačky a *m* šírku paralelných sčítačiek s postupným prenosom. Jej plocha A_T potrebná na čipe bude činiť:

$$\mathrm{A}_T = \mathrm{A}_{Ripple\ Carry\ Adder} + \mathrm{A}_{multiplexers} = \\ = \left(1 + 2\frac{n-m}{m}\right)12m\mathrm{A}_g + \left(\left(\frac{n}{m} - 1\right)(m+1)\right)7\mathrm{A}_g\ . \qquad (2.19)$$



Obrázok 2.7: 8-bitová sčítačka s výberom prenosu

2.2.6 Sčítačka s uchovaním prenosu (Carry-Save Adder)

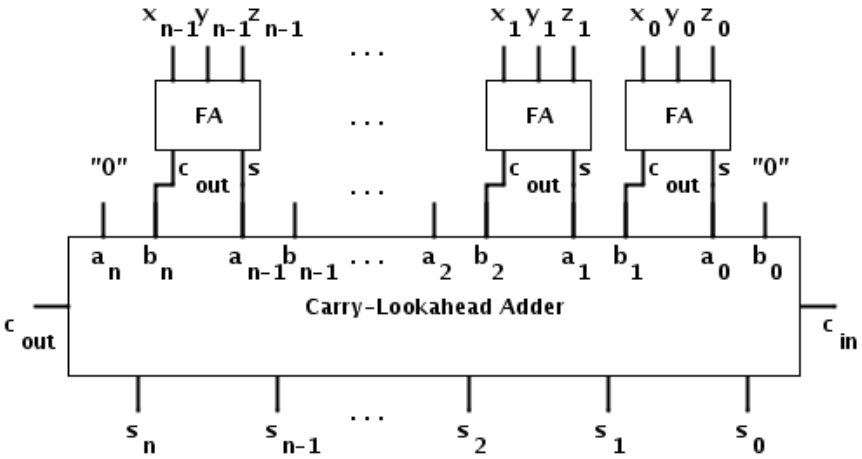
Doposiaľ boli predstavené sčítačky pracujúce s dvomi operandmi. Sčítačka s uchovaním prenosu sa od týchto obvodov odlišuje skutočnosťou, že počíta súčet až troch svojich vstupov. Má teda tendenciu byť využitá pri náročnejších aritmetických operáciach akými sú násobenie, delenie či práca s maticami.

Základom *n*-bitovej sčítačky s uchovaním prenosu je *n* jednobitových úplných sčítačiek. Štandardne sa pri jednobitových úplných sčítačkách uvažujú vstupy *a*, *b* a c_i, kde *a* a *b* sú dva vstupné operandy a c_i je prenos z nižšieho rádu. V tomto prípade ale vstupný bit c_i nadobúda význam ďalšieho vstupného operandu, nie vstupného prenosu [4]. Pre účely použitia jednobitovej úplnej sčítačky ako stavebného bloku sčítačky s uchovaním prenosu si jej vstupy označme ako *x*, *y* a *z* (teda tri vstupné operandy, ako bolo avizované). Výstupné signály tohto obvodu zostávajú interpretované i označené rovnakým spôsobom ako v iných prípadoch. Okrem jednobitových úplných sčítačiek je ešte potrebné do obvodu zakomponovať akúkoľvek logiku pre súčet dvoch operandov (napr. paralelnú sčítačku s postupným prenosom, či sčítačku s predikciou prenosu). Od jej voľby závisia aj parametre výslednej sčítačky.

Pre neskoršie osvetlenie princípu, na ktorom pracuje sčítačka s uchovaním prenosu, si uveďme ako príklad binárny súčet čísel 5, 6 a 10. Čísla ale nesčítame priamo. Najskôr vypočítame dva čiastočné výsledky - súčet *S* bez uvažovania prenosov do nasledujúceho rádu a prenosy z nižšieho rádu na jednotlivých pozíciách (označíme ako *C*). *S* a *C* napokon sčítame.

(5)		0101
(6)	+	0110
(10)	+	1010
S		1001
C	+	01100
(21)		10101

K filozofii výpočtu sčítačky s uchovaním prenosu teda povedzme, že jednobitové úplné sčítačky generujú súčet aj prenos paralelne, nakoľko v zapojení nie sú vstupné prenosy z nižších rádov uvažované. Ihneď po vystavení troch vstupných vektorov *x, y* a *z* sa generuje vektor súčtu *S* a vektor prenosu *C*. Tieto dva vektory sú napokon sčítané vyššie spomenutou sčítačkou pracujúcou s dvomi vstupnými operandmi s tým, že vo vektore prenosov *C* dochádza ešte pred samotným sčítaním k logickému posunu vľavo. K tomuto úkonu nie je potrebná žiadna špeciálna logika, ide o správne prepojenie výstupov na vstupy. Príklad zapojenia sčítačky s uchovaním prenosu uvádza Obrázok 2.8 (ako logika pre súčet dvoch operandov použitá sčítačka s predikciou prenosu).



Obrázok 2.8: Sčítačka s uchovaním prenosu

Uvažujúc v zapojení sčítačku s predikciou prenosu, pre oneskorenie Δ_T sčítačky s uchovaním prenosu bude platiť, že

$$\Delta_T = \Delta_{Full\ Adder} + \Delta_{Carry-Lookahead\ Adder} = \\ = 6\Delta_g + 4\Delta_g + [2\log_2(n+1)]\,2\Delta_g\ . \qquad (2.20)$$

Plochu A_T, ktorú potrebuje sčítačka s uchovaním prenosu na čipe, vypočítame nasledovne:

$$\mathrm{A}_T = \mathrm{A}_{Full\ Adders} + \mathrm{A}_{Carry-Lookahead\ Adder} =$$

$$= 12n\mathrm{A}_g + 8(n+1)\mathrm{A}_g + \left(\sum_{i=1}^{(n+1)} \left[\frac{1}{2} i(i+1) \right] + \frac{1}{2}(n+1)(n+2) \right) 2\mathrm{A}_g \qquad (2.21)$$

2.2.7 Porovnanie parametrov uvedených sčítačiek

V Tabuľke 2.2 je uvedený prehľad sčítačiek popísaných v tejto podkapitole. Pri každej sa nachádza hodnota jej šírky, počtu operandov, oneskorenia Δ_T a plochy A_T.

Sčítačka	**Šírka [bit]**	**Počet operandov**	**Oneskorenie Δ_T [Δ_g]**	**Plocha A_T [A_g]**
Jednobitová polovičná sčítačka	1	2	2	5
Jednobitová úplná sčítačka	1	2	6	12
Paralelná sčítačka s postupným prenosom	n	2	$4n$+2	$12n$
Sčítačka s predikciou prenosu	n	2	$2(2\log_2 n + 2)$	$2[4n + \left(\sum_{i=1}^{n} \left[\frac{1}{2} i(i+1) \right] + \frac{1}{2} n(n+1) \right)]$
Sčítačka s výberom prenosu	n	2	$4m + 5\frac{n}{m} - 3$	$5m + 24(n-m) + 7(n + \frac{n}{m} - 1)$
Sčítačka s uchovaním prenosu	n	3	$10 + 4\log_2(n+1)$	$20n + 8 + 2\left(\sum_{i=1}^{(n+1)} \left[\frac{1}{2} i(i+1) \right] + \frac{1}{2}(n+1)(n+2) \right)$

Tabuľka 2.2: Sčítačky a ich parametre

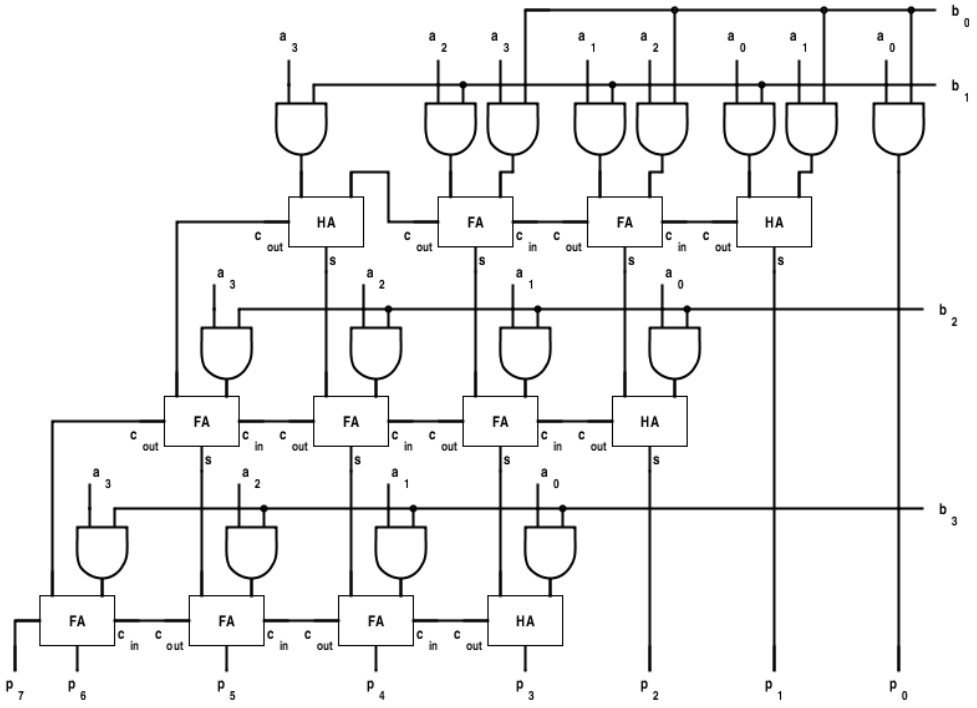
2.3 Násobičky

V modernej digitálnej elektronike zohráva proces násobenia dôležitú úlohu. Ten spotrebúva v porovnaní s procesom sčítania viac času a hardvérových zdrojov. Násobičky majú teda zásadný

vplyv na výslednú plochu i rýchlosť výsledného obvodu. Nasledujúce podkapitoly predstavia rôzne techniky násobenia, ktorých účelom je dosiahnutie optimalizácie vyššie spomenutých vlastností.

2.3.1 Kombinačná násobička (Ripple Carry Array Multiplier)

Táto násobička využíva klasický algoritmus násobenia. S násobencom je vykonaná operácia logického súčinu postupne s každým bitom násobiteľa. Výsledkom tohto sú bitové produkty, ktoré sú nakoniec krok po kroku sčítané s tým, že každý je posunutý o jednu pozíciu doľava v porovnaní s tým predchádzajúcim. Tomuto zodpovedá štruktúra sčítačiek a ich prepojenie znázornené na Obrázku 2.9. To sa skladá z hradiel AND a jednobitových polovičných i úplných sčítačiek.



Obrázok 2.9: 4-bitová kombinačná násobička

Za oneskorenie možno považovať cestu od najmenej významného vstupného bitu po najviac významný výstupný bit. Môže byť vyjadrená vzorcom

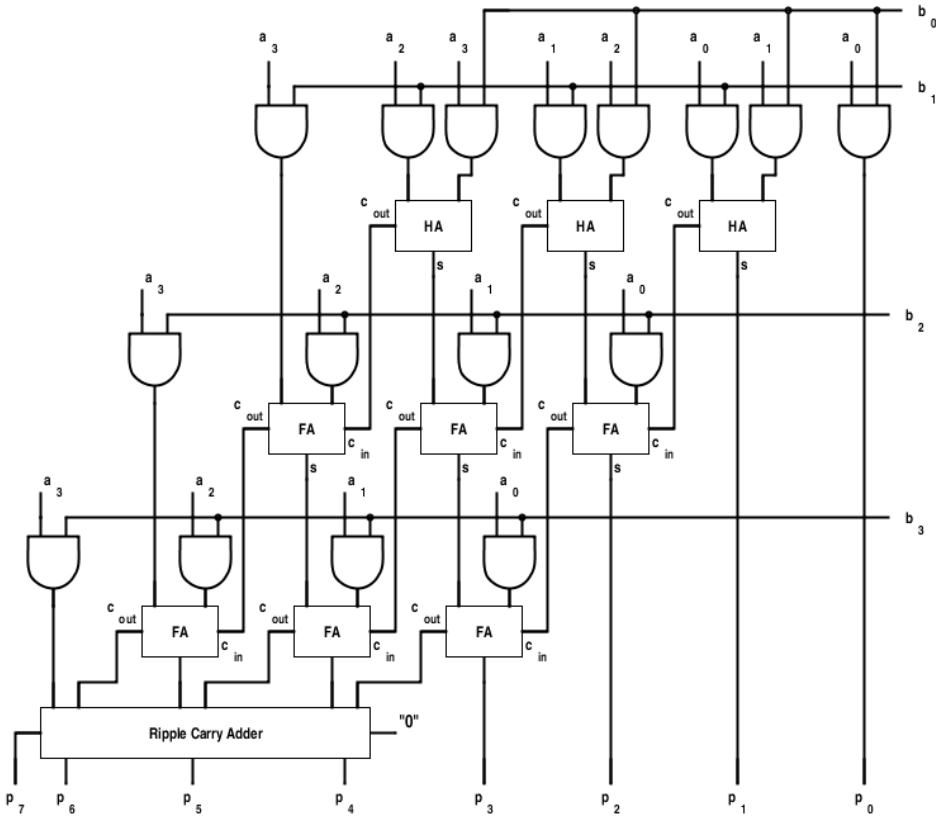
$$\Delta_T = \Delta_{AND} + (n-1)\left(\Delta_{Half\ Adder} + \Delta_{Full\ Adder}\right) + (n-2)\Delta_{Full\ Adder} = \\ = 2\Delta_g + (n-1)\left(2\Delta_g + 6\Delta_g\right) + (n-2)6\Delta_g \ . \qquad (2.22)$$

Pre plochu tejto násobičky pri šírke operandov *n* platí podľa [5] vzorec

$$\mathrm{A}_T = n^2\mathrm{A}_{AND} + n\mathrm{A}_{Half\ Adder} + n(n-2)\mathrm{A}_{Full\ Adder} = \\ = 2n^2\mathrm{A}_g + 5n\mathrm{A}_g + 12n(n-2)\mathrm{A}_g \qquad (2.23)$$

2.3.2 Násobička s uchovaním prenosu (Carry Save Array Multiplier)

Filozofiou násobičky s uchovaním prenosu je nezávislosť ktorejkoľvek sčítačky na výstupe sčítačky nachádzajúcej sa napravo od nej na rovnakom riadku pomyselnej matice. Tento fakt je zároveň aj zásadným rozdielom v porovnaní s kombinačnou násobičkou. Ostatné operácie zostávajú zhodné – pomocou operácie logického súčinu všetkých kombinácií bitov násobenca a násobiteľa sú tvorené čiastočné produkty, ktoré sú potom podľa [6] „zredukované“ prostredníctvom jednobitových polovičných sčítačiek, resp. jednobitových úplných sčítačiek. Výsledkom pravidelného zapojenia spomenutých sčítačiek sú bity reprezentujúce samotný výsledok násobenia. Vo vrchnej časti tohto bitového poľa ale pripadajú dva bity na jednu pozíciu výsledku, na základe čoho je nutné k dosiahnutiu samotného súčinu tieto dva bity na každej pozícii ešte sčítať zvlášť sčítačkou (viď Obrázok 2.10, kde je v tomto prípade použitá paralelná sčítačka s postupným prenosom).



Obrázok 2.10: 4-bitová násobička s uchovaním prenosu

Napriek tomu je vo väčšine prípadov, ako sa tvrdí v [7], násobička s uchovaním prenosu stále výhodnejšia z pohľadu oneskorenia ako kombinačná násobička. To má hodnotu

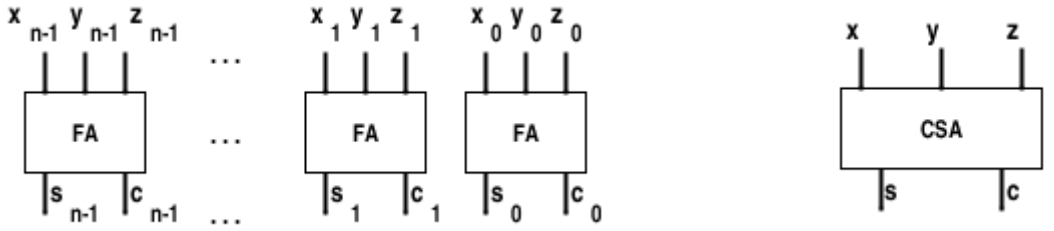
$$\Delta_T = \Delta_{AND} + \Delta_{Half\ Adder} + (n-2)\Delta_{Full\ Adder} + \Delta_{Ripple\ Carry\ Adder} =$$
$$= 2\Delta_g + 2\Delta_g + (n-2)6\Delta_g + (4[n-1]+2)\Delta_g \ . \qquad (2.24)$$

Plochu potrebnú pre implementáciu násobičky s uchovaním prenosu možno vypočítať vzťahom

$$\mathrm{A}_T = n^2\mathrm{A}_{AND} + (n-1)\mathrm{A}_{Half\ Adder} + (n-1)(n-2)\mathrm{A}_{Full\ Adder} +$$
$$+\mathrm{A}_{Ripple\ Carry\ Adder} =$$
$$= 2n^2\mathrm{A}_g + 5(n-1)\mathrm{A}_g + 12(n-1)(n-2)\mathrm{A}_g + 12(n-1)\mathrm{A}_g \qquad (2.25)$$

2.3.3 Násobička s využitím Wallaceovho stromu (Wallace Tree Multiplier)

Proces činnosti násobičky s využitím Wallaceovho stromu pozostáva, podobne ako v predchádzajúcich prípadoch, z troch krokov. V tom prvom je pomocou AND hradiel tvorená matica čiastočných produktov. V druhom kroku sa z riadkov matice zoskupia trojice (ak to rozmery matice umožnia). Tie sú pomocou jednobitových úplných sčítačiek (tvoriacich sčítačky s uchovaním prenosu, viď Obrázok 2.11) redukované na riadky dva. Podľa [8] sa tento proces opakuje pre každú takúto trojicu a následne aj pre výsledné dvojice riadkov dovtedy, pokiaľ sa týmto spôsobom nezredukuje matica na dva riadky. Tieto dva riadky sú potom v poslednom, treťom, kroku sčítané sčítačkou. Schému štvorbitovej násobičky s využitím Wallaceovho stromu (s použitím paralelnej sčítačky s postupným prenosom) znázorňuje Obrázok 2.12.



(a) (b)

Obrázok 2.11: Reprezentácia sčítačky s uchovaním prenosu pomocou: (a) jednobitových úplných sčítačiek a (b) blokovej schémy

Ako uvádza [9], hĺbka (alebo počet úrovní) samotného Wallaceovho stromu je pre *n*-bitové operandy hodnota funkcie

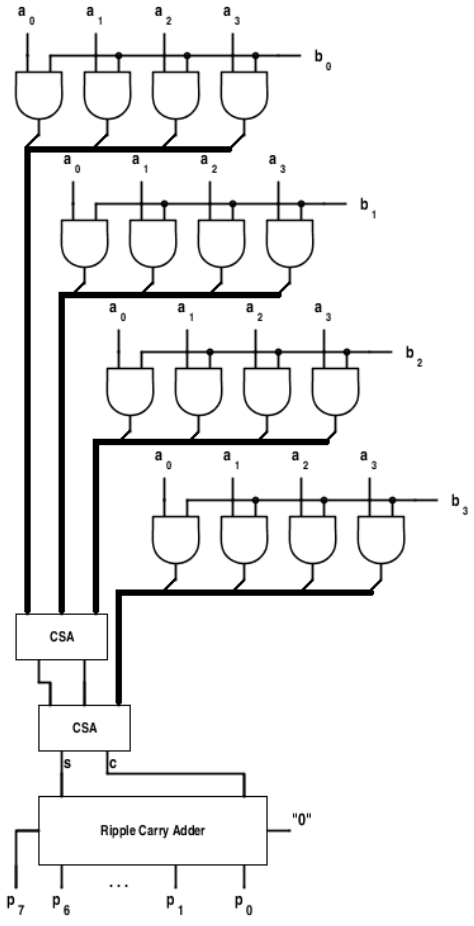
$$D(n) = \begin{cases} 0\,, & n \leq 2 \\ 1, & n = 3 \\ D\left(\left\lceil \frac{2n}{3} \right\rceil\right) + 1, & n \geq 4 \end{cases}. \tag{2.26}$$

Tá je potrebná pre výpočet oneskorenia, ktorého hodnotu vyjadruje tento vzorec:

$$\begin{aligned} \Delta_T &= \Delta_{AND} + D(n)\Delta_{Full\ Adder} + \Delta_{Ripple\ Carry\ Adder} = \\ &= 2\Delta_g + D(n)6\Delta_g + (4[2n-1]+2)\Delta_g\,. \end{aligned} \tag{2.27}$$

Pre plochu tejto násobičky na čipe zasa platí rovnica

$$\begin{aligned} \mathrm{A}_T &= n^2\mathrm{A}_{AND} + (n-2)(2n-1)\mathrm{A}_{Full\ Adder} + \mathrm{A}_{Ripple\ Carry\ Adder} = \\ &= 2n^2\mathrm{A}_g + 12(n-2)(2n-1)\mathrm{A}_g + 12(2n-1)\mathrm{A}_g \end{aligned} \tag{2.28}$$



Obrázok 2.12: 4-bitová násobička s využitím Wallaceovho stromu

2.3.4 Porovnanie parametrov uvedených násobičiek

Na konci tejto podkapitoly, podobne ako v podkapitole o sčítačkách, si taktiež uvedieme prehľad násobičiek. Ten ponúka Tabuľka 2.3. V porovnaní so spomínanou Tabuľkou 2.2 v nej však pri jednotlivých komponentoch chýbajú hodnoty šírok operandov a ich počtu. Zmieňovať ich by bolo zbytočné, nakoľko sú ich hodnoty pre všetky násobičky zhodné (*n* bitov, resp. 2 operandy).

Násobička	**Oneskorenie Δ_T [Δ_g]**	**Plocha A_T [A_g]**
Kombinačná násobička	$14n - 18$	$14n^2 - 19n$
Násobička s uchovaním prenosu	$10n - 10$	$14n^2 - 19n + 7$
Násobička s využitím Wallaceovho stromu	$8n + 6D(n)$	$26n^2 A_g - 36n A_g + 12 A_g$

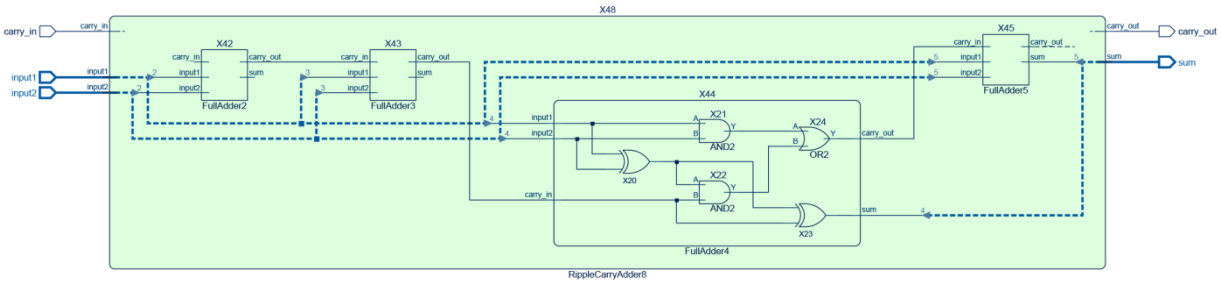
Tabuľka 2.3: Násobičky a ich parametre

3 Generátor aritmetických obvodov

V tejto kapitole si popíšeme vytvorený nástroj. Najprv budú spomenuté všetky detaily a okolnosti, ktoré treba pri jeho realizácii brať do úvahy, no a potom ich samotná implementácia v rámci vzniknutého balíka.

3.1 Návrh

Úlohou nástroja je generovanie obvodov na úrovni hradiel. V praxi to znamená produkovanie výstupu nielen vo forme samotných hradiel (ich zoznamu, tzv. netlistu), ale napr. i vo formáte, ktorý bude môcť byť následne vizualizovaný (viď Obrázok 3.1).



Obrázok 3.1: Schéma sčítačky s postupným prenosom vygenerovaná programom SpiceVision [10]

Z tohto dôvodu je nutné uvažovať i hierarchický popis, pretože uvažujúc iba zoznam hradiel, neboli by sme schopní obvod takto zobraziť. Pre isté účely (napr. pri demonštrácií funkčnosti obvodu prostredníctvom zdrojového kódu programovacieho jazyka) ale postačuje použitie samotného zoznamu hradiel, čo sa zdroja informácií o obvode týka. Hierarchický popis by jednotlivé hradlá triedil do menších či väčších skupín. Tieto skupiny by potom reprezentovali logické stavebné prvky celého obvodu (alebo väčšieho stavebného prvku obvodu).

Poďme sa pre demonštráciu detailnejšie pozrieť na Obrázok 3.1. Ten zachytáva schému sčítačky s postupným prenosom (len jej časť, ostatné stavebné bloky sú skryté). Ak popíšeme tento komponent začínajúc od hradiel, vidíme, že dvojice hradiel AND a XOR a jedno hradlo OR tvoria jednobitovú úplnú sčítačku, teda pomyselnú skupinu, či stavebný blok. Viac takýchto vhodne prepojených stavebných blokov tvorí samotnú sčítačku s postupným prenosom. Takúto informáciu by sme nemali ako zistiť, ak by sme mali k dispozícií iba samotný zoznam hradiel.

Informácie o hierarchii budú v generátore aritmetických obvodov uchované prostredníctvom dvoch prostriedkov - dátovej štruktúry (triedy) reprezentujúcej komponent a dátovej štruktúry reprezentujúcej zoznam komponentov.

3.1.1 Zoznam hradiel

Poďme sa bližšie pozrieť na štruktúru, ktorá bude slúžiť k uchovaniu netlistu (teda množiny hradiel a ich prepojení). V našom prípade budeme uvažovať kombinačné obvody, ktoré sa skladajú zo základných hradiel, kde každé hradlo môžem mať najviac dva vstupy. Uvažujúc takéto hradlá, poznať potrebujeme dva vstupné signály (v niektorých prípadoch iba 1, napr. pri invertovaní signálu), logickú funkciu, ktorú hradlo reprezentuje a výsledok tejto funkcie, teda výstupný signál hradla. Spolu sú to štyri informácie o jednom hradle.

Riešiť uchovanie informácie o výstupnom signáli hradla by však bolo zbytočné. K tejto hodnote sa vieme dopracovať z údajov o logickej funkcii hradla a jej vstupných operandov. Hradlo teda bude z dôvodu efektívneho spracovania definované trojicou, ktorej položky budú určovať vstupné signály a funkciu hradla. Výstupný signál bude definovaný práve touto trojicou.

Čo sa týka vstupných signálov hradla, pri tých budeme musieť rozlišovať medzi vstupnými signálmi celého obvodu, signálmi už vygenerovanými obvodom, napätím a uzemnením. Tento problém vyriešime použitím rozdielnych dátových typov pre jednotlivé signály. Keďže konkrétne hradlo má v zozname všetkých hradiel svoje poradové číslo, práve celé číslo ako vstupný signál bude indikovať, že ide o signál vytvorený v obvode. Jeho prostredníctvom potom možno rekurzívne zistiť hodnotu signálu (z vyššie spomenutej trojice). Pri ostatných typoch vstupných signálov je vhodné použiť ľubovoľný iný dátový typ. Ako najlepší variant sa javí reťazec znakov s tým, že napätie a uzemnenie by reprezentovali vopred dané postupnosti znakov.

Pre lepšie pochopenie si uveďme príklad. Majme dvojbitovú sčítačku bez uvažovania vstupného prenosu (z dôvodu jednoduchosti). Pre jej tvorbu použijeme jednobitovú polovičnú a jednobitovú úplnú sčítačku, keďže treba počítať s prípadným prenosom medzi rádmi. Pri názvoch vstupných signálov `op1`, `op2` a indexovaní od jednotky by v zozname trojíc vyzeral takýto obvod nasledovne:

(„`op1[1]`“,„`op2[1]`“,XOR), („`op1[1]`“,„`op2[1]`“,AND), („`op1[2]`“,„`op2[2]`“,XOR),
(2,3,XOR), („`op1[2]`“,„`op2[2]`“,AND), (2,3,AND), (5,6,OR).

Počet trojíc v zozname (sedem) sa zhoduje s predpokladaným počtom hradiel potrebných pre vytvorenie opísanej sčítačky, pretože podľa informácií v podkapitole 2.2.1 tvoria jednobitovú polovičnú sčítačku dve hradlá a podľa podkapitoly 2.2.2 tvorí jednobitovú úplnú sčítačku zasa hradiel päť. Môžeme si všimnúť, že v niektorých trojiciach sú prvými dvom položkami reťazce, v iných zasa celé čísla. V tom druhom prípade ide o poradové číslo hradla (trojice), ktorého výstupný signál je jedným zo vstupných signálov aktuálneho hradla (trojice, v ktorej sa toto poradové číslo nachádza). Doplňme ešte, že výstup prvej trojice je zároveň nižší bit výsledného súčtu, výstup štvrtej trojice je jeho vyšší bitom a výstup siedmej, teda poslednej trojice, je výstupný prenos sčítačky.

3.1.2 Štruktúra uchovávajúca zoznam komponentov

V tejto štruktúre budeme, ako už bolo spomenuté, niesť informáciu o celkovej hierarchii obvodu. Nadpis podkapitoly hovorí o zozname komponentov, no komponenty ako také nám k opisu hierarchie nestačia. Dôvody sú dva.

Tým prvým je fakt, že zo samotného zoznamu komponentov nie je jasné, ako tieto komponenty spolu súvisia, resp. ako sú medzi sebou prepojené. Problém vyriešime pripojením zoznamu stavebných prvkov (ostatných komponentov alebo hradiel) ku každému komponentu. Pre uľahčenie práce i pamäti počítača to nemusí byť zoznam konkrétnych inštancií (komponentov) alebo zoznamov (hradiel), ale ich poradových čísel v príslušných zoznamoch (v prípade zoznamu komponentov je reč o štruktúre, ktorú popisuje práve táto podkapitola). Tieto čísla sa ale musia týkať vždy iba jedného zo zoznamov. Momentálne teda položku štruktúry reprezentujúcej zoznam komponentov tvorí dvojica – inštancia komponentu a zoznam prvkov, z ktorých sa skladá.

Poďme k druhému dôvodu. Ten nadväzuje na riešenie toho prvého a bol už naznačený v jednej z predchádzajúcich viet. Pri zozname poradových čísel stavebných prvkov daného komponentu by sme pri aktuálnom stave nevedeli určiť, či ide o poradové čísla hradiel alebo položiek v zozname komponentov. Do položky v popisovanej štruktúre teda pridáme ďalšiu, tretiu, informáciu – celé číslo popisujúce úroveň (v rámci hierarchie komponentov), na ktorej sa daný komponent nachádza. Ak je hodnota tohto čísla rovná nule (teoreticky možno zvoliť akúkoľvek inú hodnotu), zoznam indexov stavebných prvkov komponentu sa odkazuje do zoznamu hradiel, v ostatných prípadoch sú položky tohto zoznamu poradové čísla položiek v tej istej štruktúre. Pridanie takejto informácie rieši nielen nastolený problém, no zároveň i, ako bolo spomenuté vyššie, udáva úroveň komponentu, čo neskôr uľahčí prácu, napr. pri generovaní kódu pre program SpiceVision, ktorý popisuje daný obvod pomocou schémy. Dohromady teda pre popis stavebného prvku z pohľadu hierarchie potrebujeme trojicu informácií o ňom.

Poďme si predchádzajúce demonštrovať na príklade. Pokračovať budeme vo vyriešenom príklade z podkapitoly 3.1.1, ktorý sa zaoberá dvojbitovou sčítačkou (v nasledujúcom príklade označenou ako 2BitA) tvorenou z jednobitovej polovičnej (HA) a jednobitovej úplnej sčítačky (FA). Poznáme zoznam hradiel (trojíc), ktoré tento aritmetický obvod tvoria, no zatiaľ nemáme vytvorenú žiadnu informáciu o jeho hierarchií. Tú ponúka nasledovný zoznam trojíc (popisovaná štruktúra):

(HA,0,(1,2)), (FA,0,(3,4,5,6,7)), (2BitA,1,(1,2)).

Prvá trojica tohto zoznamu podáva informácie o HA – hodnota „0“ na druhom mieste znamená, že ide o prvok na úrovni hradiel a na základe toho potom vieme, že ju tvorí prvé a druhé hradlo (trojica) zo zoznamu v predchádzajúcej kapitole. Analogické to bude i pri druhej trojici – FA je takisto tvorená hradlami (hodnota „0“ na druhej pozícií trojice), konkrétne tretím až siedmym zo spomínaného zoznamu. Zmena nastáva pri poslednej trojici. Jej druhou položkou je hodnota „1“, čo znamená, že indexy v zozname, ktorý je treťou položkou, ukazujú do toho istého zoznamu trojíc, v ktorom sa

nachádza daná trojica (teda už nie do zoznamu trojíc - hradiel). Z toho vyplýva, že 2BitA pozostáva z HA a FA.

3.1.3 Reprezentácia komponentu

V náväznosti na predchádzajúce podkapitoly prechádzame k charakteristike triedy reprezentujúcej samotnú komponentu. Jednými z najpodstatnejších informácií, ktoré komponentu popisujú, sú práve zoznam hradiel a jej prípadná vnútorná hierarchia zastúpená zoznamom komponent. Existujú ale aj iné nepostrádateľné údaje a táto trieda má za úlohu uchovať všetky podstatné fakty o obvode.

Takými sú nepochybne i vstupné a výstupné signály komponentu, či už ide o komponent na najvyššej úrovni alebo jeden zo stavebných prvkov. V prvom spomínanom prípade budú tieto signály zaujímať najmä užívateľa, keďže zámerom aritmetického obvodu je vo všeobecnosti výpočet nejakej hodnoty (a tieto signály poskytujú výsledok výpočtu). V druhom prípade bude zasa informácia o vstupných a výstupných signáloch nezištná pre správne prepojenie jednotlivých prvkov obvodu (ako konkrétny príklad si uveďme prepojenie signálu výstupného prenosu jednej jednobitovej úplnej sčítačky na signál vstupného prenosu ďalšej jednobitovej úplnej sčítačky na Obrázku 3.1).

Každý komponent pri svojom vytvorení zabezpečuje, aby boli informácie o ňom uložené v štruktúre popísanej v podkapitole 3.1.2. Následne si uchová poradové číslo trojice tohto zoznamu, ktorá mu zodpovedá. Toto je opodstatnené faktom, že teoreticky na každý komponent môže byť odkázané pri tvorbe iného ako na jeden zo stavebných prvkov. Na takúto „referenciu“ nám stačí poznať iba poradové číslo trojice reprezentujúcej daný komponent v zozname týchto trojíc. Všetky ostatné potrebné informácie (inštanciu komponentu, jeho úroveň a zoznam jeho stavebných prvkov) nám už poskytne samotná položka.

3.1.4 Problém rozdielnej úrovne stavebných prvkov v rámci hierarchie obvodu

Pri tvorbe obvodu môže nastať situácia, v ktorej sa generovaný komponent (či už je to komponent na najvyššej úrovni alebo nie) skladá z hradiel a komponentu (prípadne viacerých komponentov) súčasne (takýto jav môžeme pozorovať napr. pri kombinačnej násobičke – ak sa pozrieme na Obrázok 2.9, postrehneme, že ako celok ju tvoria AND hradlá, jednobitové úplne a jednobitové polovičné sčítačky). V štruktúre reprezentujúcej zoznam komponentov je v každej trojici uvedený i zoznam jeho stavebných prvkov. Akonáhle by sa v zozname stavebných prvkov vyskytovali indexy hradiel i indexy komponentov súčasne, neboli by sme schopní určiť, ktorého zoznamu (zoznam trojíc - hradiel, zoznam trojíc - komponentov) je číslo indexom.

Problém vyriešime tak, že z hradiel vytvoríme ďalší komponent (napriek tomu, že teoreticky spolu nemusia nijak súvisieť). Od tohto momentu budú v spomínaných prípadoch v zozname stavebných prvkov už len indexy odkazujúce sa na zoznam komponentov (pre kombinačnú sčítačku

to teda budú indexy jednobitových úplných sčítačiek, jednobitových polovičných sčítačiek a index komponentu „obaľujúceho“ AND hradlá).

Podobná situácia, i keď z pohľadu funkcionality oveľa menej závažná, nastane, ak sa komponent (opäť nezáleží na jeho úrovni) skladá z iných komponentov a tie nie sú na rovnakej úrovni z pohľadu hierarchie obvodu (ako príklad si vezmime sčítačku s výberom prenosu na Obrázku 2.7 – skladá sa zo sčítačiek s postupným prenosom a z multiplexorov 2-1; sčítačku s postupným prenosom tvoria jednobitové úplne sčítačky, zatiaľ čo multiplexor 2-1 je komponent vytvorený z hradiel). Tú sprehľadníme tak, že budeme dbať na to, aby komponent pozostával iba z komponentov na zhodnej úrovní. V prípade, že sú ich úrovne rozdielne, budeme z tých s nižšou úrovňou vytvárať nové komponenty (bez akéhokoľvek logického významu či funkcie ako celku) dovtedy, pokiaľ sa úroveň tohto novovytvoreného komponentu nebude rovnať úrovni komponentu, ktorý bol pôvodne na vyššej úrovni (uvažujúc sčítačku s postupným prenosom, práve multiplexory 2-1 sú vhodnými kandidátmi na stavebné prvky takéhoto nového komponentu).

3.1.5 Parametrizácia generovaného obvodu

Obvod ako celok môže mať určité parametre. Jedným z nich je aritmetická funkcia, ktorú realizuje. S týmto parametrom naložíme tak, že všetky implementované aritmetické obvody (rôzne typy sčítačiek a násobičiek) budú reprezentované vlastnou triedou.

Parametrom je nepochybne i šírka vstupných (a z nej odvodená šírka tých výstupných) signálov. Toto číslo nemusí byť nikde uvedené. Stačí, ak sa pri volaní konštruktoru triedy na miesto očakávaného viacbitového vstupného signálu dosadí zoznam reťazcov, celých čísel alebo ich kombinácií (pre vysvetlenie viď podkapitola 3.1.1) a šírka sa odvodí automaticky z dĺžky tohto zoznamu. Na druhej strane možno šírku uviesť aj explicitne, a to vložením zoznamu s dvomi položkami, kde prvá udáva názov signálu a tá druhá jeho šírku (napr. `["input1",8]`), na rovnaké miesto v konštruktore triedy. V tomto prípade zareaguje konštruktor na túto sekvenciu v zozname vytvorením a uložením zoznamu číslovaných reťazcov požadovanej dĺžky (teda napr. `["input1[0]", "input1[1]", …, "input1[7]"]`), s ktorým bude potom ďalej pracovať. V prípade, ak majú zoznamy reprezentujúce vstupné signály rozdielnu dĺžku, za oficiálnu šírku sa bude považovať väčšia z týchto dĺžok. Kratší zo zoznamov sa potom doplní signálmi uzemnenia tak, aby boli dĺžky signálov zhodné (ušetrenie réžie kvôli testovaní dĺžky pre následné indexovanie) a jeho pôvodná hodnota zostala nezmenená.

Ďalšími príkladmi parametrov, i keď už nepravidelnými, ktoré sa líšia obvod od obvodu, sú napr. šírka sčítačky s postupným prenosom použitej v sčítačke s výberom prenosu či použitý typ sčítačky v niektorých typoch násobičiek. Takéto parametre budú obsahovať konštruktory tried, ktorých sa to týka.

3.1.6 Optimalizácia obvodu

Trieda reprezentujúca komponent zabezpečuje, okrem uchovania podstatných informácií, aj isté operácie. Jednou z nich je optimalizácia hradiel. Ide o zmenu dvojvstupového hradla na jednovstupový buffer, prípadne invertor, za účelom zníženia hodnoty oneskorenia a rozsahu plochy obvodu. Takáto operácia je ale možná iba v prípadoch, ak je jedným zo vstupných signálov pôvodného hradla napätie alebo uzemnenie (čo znamená, že nie vždy je takáto optimalizácia realizovateľná). Od tejto hodnoty sa potom odvíja hodnota vstupného signálu bufferu, resp. invertoru. Príklady takýchto redukcií pre logické operácie (hradlá) použité pri generovaní aritmetických obvodov uvádza Tabuľka 3.1.

Logická funkcia	AND		OR		XOR	
1. operand	*x*					
2. operand	0	1	0	1	0	1
Výsledok	0	*x*	*x*	1	*x*	$\neg x$

Tabuľka 3.1: Výsledky vybraných logických funkcií s konštantným operandom

3.1.7 Generovanie kódov reprezentujúcich daný obvod

Ďalšou z funkcionalít, ktoré ponúka trieda zapúzdrujúca obvod je jeho reprezentácia prostredníctvom zdrojového kódu programovacích jazykov alebo netlistu, ktorý môže byť použitý pre simuláciu, syntézu i vizualizáciu.

V prvom prípade je kód generovaný najmä za účelom overenia funkčnosti obvodu, keďže zo samotného zoznamu hradiel je to len ťažko zistiteľné. Pri generovaní tohto kódu budeme taktiež optimalizovať, podobne, ako pri zozname hradiel. Zatiaľ čo pri zozname hradiel ide o ušetrenie plochy a zmenšenie oneskorenia obvodu, v tomto prípade sa optimalizácia týka ušetrenia (i keď možno nebadateľného) výpočtových zdrojov. Keďže optimalizácia kódu sa týka redukcie dvojvstupových hradiel na jednovstupové buffery, v generovanom kóde by sa potom objavili zbytočné priradenia typu

```
signal10 = 0
```

(uvažujúc pseudokód, nie konkrétny programovací jazyk, príkaz by reprezentoval uzemnenie ako vstup bufferu). Po odstránení priradení hodnoty do premennej bez operácie by sa nám v niektorých prípadoch zasa objavili priradenia ako

```
signal20 = (1 | signal10)
```

(operácia logického bitového súčtu), čo by tiež nebolo príliš efektívne (viď predchádzajúca podkapitola). Celkovým cieľom tejto optimalizácie by teda bolo z kódu úplne odstrániť okrem priradení bez operácie i výskyt konštánt.

V prípade druhom sme schopní zobraziť hierarchiu a prepojenie jednotlivých stavebných prvkov v rámci danej komponenty. Okrem zoznamu hradiel je pri generovaní tohto kódu využitá aj štruktúra reprezentujúca zoznam komponent. V podstate ide o vizualizáciu práve tejto štruktúry.

3.1.8 Konštantný koeficient ako vstupný signál obvodu

V praxi sa môžu vyskytnúť prípady, keď sa jeden zo vstupných signálov komponentu nemení (napr. pri násobičke vo filtri s konečnou impulzovou odpoveďou). Z pohľadu obvodu teda môžeme konštatovať, že sa znížil počet jeho vstupných signálov. Vstupné signály komponentu, ktoré predstavujú spomínanú konštantnú hodnotu, napojíme na napätie alebo uzemnenie tak, aby ich logické hodnoty predstavovali práve tento koeficient. S určitosťou (výskyt signálov napätia, resp. uzemnenia v obvode) tak získame priestor pre časovú i priestorovú optimalizáciu obvodu pri zachovaní pôvodnej aritmetickej operácie, na ktorú bol určený.

3.2 Implementácia

V nasledujúcom texte uvedieme, ako sa faktory podstatné pre obvod, ktoré sú popísané v podkapitole o návrhu, premietli do implementácie balíku `arithmetic_circuit_generator` v jazyku Python (vo verzii 2.7.3). Na úvod zmieňme fakt, že implementácia každej triedy sa nachádza vždy v samostatnom a rovnomennom module a moduly sú na základe logickej príbuznosti (napr. sčítačky, násobičky) väčšinou zoskupené do balíkov (zložiek obsahujúcich súbor `__init__.py`).

3.2.1 Trieda GateArray

Táto trieda s bezparametrickým konštruktorom reprezentuje samotný obvod. Nesie informáciu zo zozname hradiel (prostredníctvom svojho atribútu `gates`). Hradlo je v tomto zozname zastúpené zoznamom s tromi položkami (atribút `gates` je teda zoznam zoznamov), pričom prvé dve sú vstupnými signálmi hradla a tou treťou je vopred určený reťazec priraďujúci hradlu jeho logickú funkciu. Ak má hradlo iba jeden vstupný signál (napr. invertor alebo buffer), dĺžka signálu sa napriek tomu nemení, no na miesto druhej položky v zozname (pomyselného druhého vstupného signálu) sa vloží hodnota `None`. V opačnom prípade by v prípade práce s týmito zoznamami došlo k navýšeniu réžie spojenej s testovaním dĺžky zoznamov a na základe toho aj s ich indexovaním. Metóda `appendGate` je zodpovedná za ukladanie informácii o hradle do atribútu `gates`. Jej návratovou hodnotou je index novo pridanej položky (kvôli prípadnému neskoršiemu uloženiu).

Druhým (a zároveň posledným) atribútom triedy `GateArray` je atribút `components`. Ten je zodpovedný za uchovanie hierarchie obvodu a sú v ňom uložené logické stavebné prvky výsledného komponentu. Takýto prvok je, rovnako ako pri hradle, zastúpený zoznamom s tromi položkami. V rámci tej prvej je uložená inštancia triedy konkrétneho komponentu, druhá udáva úroveň tohto

komponentu v celkovej hierarchii výsledného komponentu a tá tretia je zoznamom indexov komponentov (zoznamu `components`) alebo hradiel (zoznamu `gates`), z ktorých komponent popísaných prvou položkou pozostáva. Pre pridávanie položiek do tohto zoznamu sú určené metódy `appendGateComponent` a `appendComponent`. Odlišujú sa tým, že prvá spomínaná metóda pridáva do zoznamu komponent, ktorej úroveň v rámci celej hierarchie je najnižšia (teda komponent zložený z hradiel), čo bude reprezentované hodnotou „0“ v druhej položke zoznamu popisujúceho komponent. Metóda `appendComponent` zasa pridáva do zoznamu komponent skladajúci sa z iných komponentov a hodnota jeho úrovne sa vypočíta z hodnôt úrovní práve týchto komponentov (nájdením ich maximálnej hodnoty a jej zvýšením o hodnotu „1“).

Trieda tiež prostredníctvom metódy `optimalizeArrayOfGates` optimalizuje príslušné hradlá v atribúte `gates`, ako bolo popísané v podkapitole 3.1.6.

3.2.2 Trieda BaseComponent

Ako je spomenuté v podkapitole 3.1.5, pre každý typ sčítačky alebo násobičky budeme mať vyhradenú zvlášť triedu. To, čo majú všetky tieto komponenty spoločné, či už sú nimi sčítačky, násobičky a ich rôzne varianty, je fakt, že pozostávajú z hradiel a prípadných komponentov. Práve vlastnosti, ktoré robia tieto komponenty v istom slova zmysle príbuznými, sú vhodnými kandidátmi na atribúty rodičovskej triedy. Tou bude teda trieda `BaseComponent`.

Jedným z jej atribútov bude atribút `gateArray`, ktorý bude uchovávať informácie o obvode (rovnaké ako uchováva trieda `GateArray` – ak by bol Python staticky typový jazyk, tak by bolo možné povedať, že atribút `gateArray` je typu `GateArray`).

Konštruktor triedy `BaseComponent` má voliteľný parameter `createdGateArray`. V prípade použitia sa na jeho mieste očakáva inštancia triedy `GateArray`, ktorá by sa priradila atribútu `gateArray`. V opačnom prípade sa tomuto atribútu priradí novo vytvorená inštancia triedy `GateArray`. Vysvetlenie je jednoduché. Ak sa komponent na najvyššej úrovni skladá z iných komponentov, musia zdieľať spoločný zoznam hradiel i komponentov, do ktorého budú generovať hradlá, resp. ukladať komponenty (teda samých seba), keďže vo výsledku tvoria jeden obvod. Toto sa dosiahne práve spomínaným voliteľným parametrom.

Trieda má ešte dva atribúty – `gates` a `components`. Sú to vlastne kópie zhodne pomenovaných atribútov atribútu `gateArray`. Špeciálny význam nemajú, dôvod ich použitia je čisto praktický a skracuje prístup k jednotlivým zoznamom (napr. `instance.gates` miesto zdĺhavejšieho `instance.gateArray.gates`).

Čo sa týka metód tejto triedy, všetky sú použité pre generovanie zdrojových kódov reprezentujúcich vytvorený obvod. Metóda `generatePyOrCCode` vracia na základe logickej hodnoty parametru `generatingPythonCode` reťazec so zdrojovým kódom funkcie v jazyku

Python alebo C, `generateSpiceCode` zasa vracia reťazec so zdrojovým kódom pre vizualizačný program SpiceVision.

3.2.3 Trieda GateLevelPartOfComponent a ComponentLevelPartOfComponent

Tieto dve triedy si sú svojim zámerom veľmi podobné. Riešia problematiku rozobranú v podkapitole 3.1.4. Trieda `GateLevelPartOfComponent` zabezpečuje vytvorenie komponentu z prijatého zoznamu hradiel (`gates`) v jej konštruktore. Okrem tohto zoznamu hradiel sú parametrami konštruktoru `name` (názov komponentu), `inputs` (jeho vstupné signály), `outputs` (jeho výstupné signály) a voliteľný parameter `createdGateArray`. Obsahy všetkých parametrov sú ulozené do rovnako pomenovaných atribútov. Prítomnosť posledného spomenutého parametru v konštruktore a fakt, že trieda je potomkom triedy `BaseComponent`, naznačujú že inštancia tejto triedy je považovaná za plnohodnotný komponent. Od tých ostatných sa odlišuje iba tým, že, okrem konštruktoru neimplementuje žiadne iné metódy (keďže všetky hradlá, ktoré ju tvoria, sú v čase vytvorenia inštancie tejto triedy už vygenerované).

Popis triedy obsiahnutý v predchádzajúcom odseku možno kľudne považovať i za popis triedy `ComponentLevelPartOfComponent`. Majú len malú rozdielnosť – parameter konštruktoru a atribút `gates` (`GateLevelPartOfComponent`) je nahradený parametrom, resp. atribútom `components`., keďže táto trieda vytvára komponent zo zoznamu komponent a nie hradiel.

3.2.4 Konkrétny aritmetický obvod

Opisovať všetky aritmetické obvody triedu po triede by bolo zdĺhavé. Taktiež by bola väčšina informácií veľmi podobných. Namiesto toho si tieto údaje zhrnieme na jednom mieste.

Najdôležitejšou časťou každého komponentu (triedy) je jeho konštruktor. V ňom sa deje všetko podstatné. Poďme si to priblížiť.

Konštruktor každej takejto triedy vždy prijíma prostredníctvom parametrov vstupné signály komponentu (prípadné ostatné parametre závisia na type komponentu). Tie sú v prípade potreby (a uvedenia správneho formátu, viď podkapitola 3.1.5) vygenerované funkciou `generateInputSignalsIfNeeded`, neskôr sú kontrolované na dátový typ funkciou `checkIfStringOrInt` (akceptovateľnými hodnotami sú reťazec a celé číslo, pre vysvetlenie viď podkapitola 3.1.1) a do tretice je vďaka funkcii `editInputsIfNeeded` zaobstaraná zhodná dĺžka zoznamov reprezentujúcich jednotlivé vstupné signály (všetky spomenuté funkcie sú z modulu `arithmetic_circuit_generator.library`). Nasleduje volanie konštruktoru rodičovskej triedy (`BaseComponent`), uloženie potrebných informácií (názov komponentu, vstupných signálov) a volanie metódy triedy, ktorá má na starosti vygenerovanie daného obvodu. Po týchto úkonoch sa

vytvorený komponent (po prípadnom „obalení“ jeho hradiel či komponentov, viď podkapitola 3.1.4) pridá do zoznamu `components` (prostredníctvom atribútu). Na úplnom konci sa pomocou atribútu `gateArray` zavolá metóda `optimalizeArrayOfGates`, ktorá zoptimalizuje obvod, ak to je možné.

Ďalšou z metód, ktorú by sme mali spomenúť, je metóda `getGeneratedCode` s parametrom `language` a voliteľným parametrom `showBrackets`, ktorej návratovou hodnotou je reťazec obsahujúci zdrojový kód popisujúci obvod. Parameter `language` predstavuje práve voľbu jazyka zdrojového kódu (jeho hodnotou môže byť jedna z trojice konštánt `PYTHON_LANGUAGE`, `C_LANGUAGE`, `SPICE_LANGUAGE` nachádzajúcich sa v module `arithmetic_circuit_generator.library`). Použitie parametru `showBrackets` má význam iba pri generovaní zdrojového kódu pre program SpiceVision. Užívateľ jeho použitím rozhoduje, či budú názvy signálov číslované s hranatými zátvorkami (napr. „`input[0]`“) alebo bez nich (napr. „`input0`“). Tento krok má opodstatnenie pri vizualizácií signálov.

3.2.5 Balík subcomponents

Ako už samotný názov balíku napovedá, moduly reprezentujúce komponenty sa v ňom síce nachádzajú, no ako samostatné logické celky tieto moduly žiaden význam, čo sa aritmetických výpočtov týka, nemajú. Do balíku patria moduly `BlackCell`, `DoubleInputBuffer`, `GreyCell`, `Multiplexer`, `ValencyBlackCell` a `ValencyGreyCell`. Okrem modulu `Multiplexer`, ktorý implementuje multiplexor 2-1 (ten využíva sčítačka s výberom prenosu), sú všetky ostatné komponenty použité v topológiách sčítačiek typu Tree Adder (modul `adders.TreeAdder`) a Higher Valency Tree Adder (`adders.HigherValencyTreeAdder`) popísaných v [11].

3.2.6 Balík adders

Tento balík obsahuje moduly implementujúce rôzne druhy sčítačiek. Tými sú jednobitové dvojvstupové sčítačky (moduly `FullAdder` a `HalfAdder`) či viacbitové dvojvstupové sčítačky (`CarryLookaheadAdder`, `CarrySelectAdder` a `RippleCarryAdder`), medzi ktoré patria i v podkapitole 2.2 nespomenuté sčítačky Tree Adder (modul `TreeAdder`) a Higher Valency Tree Adder (modul `HigherValencyTreeAdder`). Pri oboch je možnosť výberu topológie generujúcej prenosy jednotlivých rádov (topológie Brent-Kung, Sklansky, Kogge-Stone a Han-Carlson sú s malým rozdielom schopné implementovať oba spomenuté moduly, no modul `TreeAdder` navyše implementuje i topológie Knowles, Ladner-Fischer). Nachádza sa tu ešte modul pre viacbitovú trojvstupovú sčítačku (`CarrySaveAdder`), ako aj modul `ConstantCoefficientAdder`, ktorý generuje viacbitovú dvojvstupovú sčítačku (niektorú z modulov `CarryLookaheadAdder`,

CarrySelectAdder a RippleCarryAdder), ktorej jedným zo vstupných operandov je konštantná hodnota.

3.2.7 Balík multipliers

V tomto balíku sa nachádzajú moduly pre násobičky. Každá z nich využíva aspoň jeden z modulov vyššie spomenutého balíku adders. Moduly adders.HalfAdder a adders.FullAdder sú importované v moduloch RippleCarryArrayMultiplier a CarrySaveArrayMultiplier. Moduly CarrySaveArrayMultiplier a WallaceTreeMultiplier síce v poslednom kroku k dosiahnutiu výsledného súčinu používajú jednu z tried v moduloch adders.CarryLookaheadAdder, adders.CarrySelectAdder alebo adders.RippleCarryAdder, no tie nie sú importované priamo v nich, ale v module arithmetic_circuit_generator.library, keďže ten implementuje príslušnú funkciu (getResultOfSummation), ktorá toto vykonáva. Posledným modulom tohto balíku je modul ConstantCoefficientMultiplier, ktorý je schopný vygenerovať každú zo zmieňovaných násobičiek s jedným vstupom konštantným.

3.2.8 Balík absolute_difference_circuit

V tomto balíku sa nachádzajú moduly generujúce obvod počítajúci absolútny rozdiel dvoch operandov. V princípe ide o obdobu sčítačky Tree Adder využívajúcu Kogge-Stone topológiu. Tá je mierne pozmenená, rovnako ako posledná, sumačná časť pôvodnej sčítačky. Opísaný obvod je implementovaný v module AbsoluteDifferenceByFlaggedPrefixAdder. Okrem neho sa v balíku nachádza ešte druhý modul, a to ConstantCoefficientAbsolutDifferenceByFlaggedPrefixAdder. Rovnako ako v balíku adders alebo multipliers, i v tomto prípade ide o implementáciu varianty obvodu s konštantným koeficientom.

4 Experimentálne vyhodnotenie generátoru aritmetických obvodov

Po návrhu a následnej implementácií generátoru aritmetických obvodov sa prešlo k testovaniu tohto nástroja. Tiež boli zistené jeho vlastnosti. Čo sa testovania týka, overovala sa funkčnosť vytvorených obvodov. Pri nesplnení tohto predpokladu by sa stal nástroj prakticky nepoužiteľným. Ako vlastnosť, ktorá by balík `arithmetic_circuit_generator` a jeho moduly výstižne charakterizovala, sa javí počet časových jednotiek potrebných na vykonanie vybraných operácií inštancií tried.

4.1 Testovanie funkčnosti implementovaných obvodov

Na validáciu komponentov generovaných vytvoreným balíkom bol použitý testovací framework jazyka Python `unittest`. Zdrojové kódy testov sa nachádzajú v balíku `tests_lib`. Tie sú potom spúšťané modulom `testsOfComponents`, v ktorom je spomínaný balík importovaný.

Pre testy sčítačiek, násobičiek a obvodu počítajúceho absolútny rozdiel boli vytvorené samostatné triedy. Tieto triedy sú potomkami triedy `unittest.TestCase` (to znamená, že ich metódy sa pomocou inštancie triedy `TestSuite` združia do skupín a dosadením takejto inštancie ako parametru do metódy `run` triedy `TextTestRunner` sa spustia samotné testy s výstupom na štandardný chybový výstup). V každej ich metóde sa testuje jeden komponent. Takáto metóda pozostáva zo slučky `for`, ktorá iteruje hodnotami zoznamu (premennej `WIDTHS_TO_TEST` definovanej v module `library` tohto balíku), ktorý obsahuje požadované bitové šírky pre testovanie (8, 16, 32, 64). V každom cykle sa potom vytvorí inštancia triedy (komponent) s požadovanou šírkou vstupných operandov. Jej prostredníctvom sa získa funkcia v kóde Python (reťazec) popisujúca daný obvod a následne sa zavolá funkcia z modulu `library`, ktorej sa tento kód odovzdá ako parameter. Voľba funkcie závisí od vlastností konkrétneho komponentu (ak ide vo všeobecnosti napr. o násobičku, použije sa funkcia `compileCodeAndTestMultiplier`, v ostatným prípadoch je to analogické). V spomínanej funkcii dochádza ku skompilovaniu prijatého kódu (Python funkcie). Nasleduje slučka `for`, prostredníctvom ktorej sa užívateľom definovaný počet krát (hodnota za prepínačom `--count` pri spustení modulu `testsOfComponents`) náhodne vygeneruje potrebný počet vstupných operandov (v požadovanom rozsahu), ktoré sú potom dosadené ako parametre

kompilovanej funkcie. Jej výsledok sa porovná s očakávaným výsledkom pomocou metódy `assertEqual` triedy `TestCase` z `unittest` frameworku.

Pri sčítačkách prebehne vykonanie kompilovanej funkcie a následné otestovanie jej výstupu dvakrát – raz bez vstupného prenosu a následne aj s ním. Pri komponentoch s konštantným koeficientom sa zasa pre každú šírku vstupných operandov vytvára inštancia triedy niekoľkokrát – vždy s iným konštantným vstupom (aby sa netestovalo iba s jednou konštantnou hodnotou pre danú šírku – takýto test by príliš dôveryhodný nebol).

Na záver dodajme, že všetky triedy testy splnili. Každá vygenerovaná funkcia bola otestovaná tisíckrát (modul `testsOfComponents` bol teda spustený s argumentmi `--count 1000`)

4.2 Rýchlosti generovania

Pre zmeranie počtu časových jednotiek potrebných ku konkrétnym operáciám generátoru aritmetických obvodov bol použitý modul `timeit` zo štandardnej knižnice jazyka Python. Zdrojové kódy popísané v tejto podkapitole sa nachádzajú v balíku `code_features_lib` (okrem spustiteľného modulu `codeFeatures`, ktorý obsahuje import tohto balíka).

Samotnými meranými operáciami sú vytvorenie inštancie triedy a vygenerovanie zdrojových kódov (v jazyku Python, C a vo vstupnom formáte vizualizéru SpiceVision) popisujúcich vytvorený obvod už existujúcimi inštanciami (pre každý z typov zdrojových kódov je určené samostatné meranie). Podobne ako pri validácii komponentov prostredníctvom vytvorených zdrojových kódov, aj v tomto prípade budú merania prebiehať pre rôzne šírky (8, 16, 32 a 64 bitov) vstupných operandov každého komponentu. Pri spustení modulu `codeFeatures` možno zvoliť počet meraní každej operácie prostredníctvom čísla za prepínačom `--count`. Čím je toto číslo väčšie, tým budú namerané hodnoty presnejšie, keďže z nameraných hodnôt sa potom spraví aritmetický priemer.

Poďme sa pozrieť na výsledky meraní. Tie boli vykonané na počítači HP ProBook 4530s s procesorom Intel(R) Core(TM) i5-2450M CPU @ 2.50GHz × 4, kapacitou RAM pamäte 4,0 GB a 64-bitovým operačným systémom ubuntu 12.04 LTS. Čas každej z operácií bol odmeraný desaťkrát (teda modul `codeFeatures` bol spustený s argumentmi `--count 10`). V tejto podkapitole sú uvedené len vybrané namerané údaje. Úplne všetky hodnoty je v prípade záujmu možné nájsť v tabuľkách v Prílohe B.

4.2.1 Sčítačky

V Tabuľke 4.1 sa nachádzajú namerané hodnoty vybraných tried, ktoré implementujú sčítačky so šírkou vstupov 64 bitov. Pri vytváraní inštancií sčítačiek tejto šírky treba počítať s časmi od niekoľkých tisícin až po niekoľko desatín sekundy Pri generovaní kódov reprezentujúcich obvody sú tieto hodnoty o už niečo lepšie – tu ide maximálne o stotiny sekundy.

Vo všetkých meraných vlastnostiach je najefektívnejšia trieda generujúca obvod sčítačky s postupným prenosom. Naopak najpomalšou je vo väčšine ukazovateľoch (okrem rýchlosti generovania kódu pre vizualizér SpiceVision) sčítačka s uchovaním prenosu využívajúca sčítačku s predikciou prenosu ako výslednú sčítaciu logiku. Treba si ale všimnúť, že tento fakt je spôsobený práve použitím sčítačky s predikciou prenosu, keďže pri použití inej sčítacej logiky sa jej výsledky v niektorých prípadoch rádovo líšia. Domnievam sa, že dôvodom je použitie optimalizačného algoritmu, ktorý predchádza duplikácií hradiel s rovnakými vstupmi, v konštruktore triedy `CarryLookaheadAdder` a tiež fakt, že 64-bitový variant tejto sčítačky je celkom náročný na plochu.

Typ sčítačky	**Čas vytvorenia inštancie triedy [s]**	**Čas generovania Python kódu [s]**	**Čas generovania C kódu [s]**	**Čas generovania kódu pre SpiceVision[s]**
Sčítačka s postupným prenosom	5.5181×10^{-3}	1.1379×10^{-3}	1.1899×10^{-3}	5.8491×10^{-3}
Sčítačka s predikciou prenosu	4.4239×10^{-1}	2.1788×10^{-2}	2.1801×10^{-2}	2.5245×10^{-2}
Sčítačka s výberom prenosu	5.2429×10^{-2}	3.0109×10^{-3}	3.0009×10^{-3}	5.4966×10^{-2}
Sčítačka s uchovaním prenosu (používajúc sč. s post. pren.)	2.1039×10^{-2}	2.2201×10^{-3}	2.2211×10^{-3}	2.0423×10^{-2}
Sčítačka s uchovaním prenosu (používajúc sč. s pred. pren.)	4.9501×10^{-1}	2.3645×10^{-2}	2.3519×10^{-2}	4.0659×10^{-2}
Sčítačka s uchovaním prenosu (používajúc sč. s výb. pren.)	9.8017×10^{-2}	4.1029×10^{-3}	4.1048×10^{-3}	9.2925×10^{-2}

Tabuľka 4.1: Časy operácií vybraných tried reprezentujúcich 64-bitové sčítačky

4.2.2 Násobičky

Pri vytváraní tried reprezentujúcich 64-bitové násobičky musíme v najhoršom prípade hovoriť až o čase väčšom ako minúta. V prípade najlepšom je tento čas o niečo menší ako 26 sekúnd. Ak porovnáme rýchlosť generovania kódu pre program SpiceVision a rýchlosti generovania funkcií v jazyku Python či C, v prvom prípade hovoríme o desiatkach sekúnd, v prípade druhom o stotinách a desatinách sekundy. Tieto hodnoty sa vo veľkej miere líšia od hodnôt sčítačiek. Uvedené sú v Tabuľke 4.2. Do úvahy ale treba brať aj fakt, že výsledok 64-bitových sčítačiek má šírku 64 bitov, zatiaľ čo výsledok násobičiek s rovnakou šírkou vstupov (64 bitov) má šírku výstupu dvojnásobnú (128 bitov).

Z Tabuľky 4.2 vidíme, že medzi násobičkami možno za najrýchlejšie vyhlásiť dve triedy súčasne. Je to trieda implementujúca kombinačnú násobičku a trieda implementujúca násobičku s uchovaním prenosu (ale iba v prípade použitia sčítačky s postupným prenosom). V jednotlivých

časových vlastnostiach sa odlišujú iba nebadane (najväčší rozdiel dosahujú pri čase vytvorenia inštancie – niečo viac ako pol sekundy). Z vyššie uvedeného vyplýva (keďže násobičky sú implementované iba tri), že najhoršie časové vlastnosti má trieda generujúca násobičku s využitím Wallaceovho stromu. O pomyselnú poslednú priečku sa delia jej dva varianty (z dôvodu podobnosti časových vlastností) – variant s využitím sčítačky s predikciou prenosu a variant s využitím sčítačky s výberom prenosu.

Typ násobičky	**Čas vytvorenia inštancie triedy [s]**	**Čas generovania Python kódu [s]**	**Čas generovania C kódu [s]**	**Čas generovania kódu pre SpiceVision [s]**
Kombinačná násobička	25.8646	7.9859×10^{-2}	7.9343×10^{-2}	16.2918
Násobička s uchovaním prenosu (používajúc sč. s post. pren.)	26.3659	7.9854×10^{-2}	8.0587×10^{-2}	16.7085
Násobička s uchovaním prenosu (používajúc sč. s pred. pren.)	27.1575	9.9154×10^{-2}	1.0038×10^{-1}	16.7494
Násobička s uchovaním prenosu (používajúc sč. s výb. pren.)	28.9131	8.1296×10^{-2}	8.1959×10^{-2}	18.0325
Násobička s využitím Wall. stromu (používajúc sč. s post. pren.)	76.4162	1.1893×10^{-1}	1.1850×10^{-1}	44.3930
Násobička s využitím Wall. stromu (používajúc sč. s pred. pren.)	84.2637	2.1775×10^{-1}	2.1651×10^{-1}	43.8575
Násobička s využitím Wall. stromu (používajúc sč. s výb. pren.)	83.6459	1.2293×10^{-1}	1.2425×10^{-1}	49.4625

Tabuľka 4.2: Časy operácií tried reprezentujúcich 64-bitové násobičky

5 Využitie generátoru na porovnanie parametrov aritmetických obvodov

Okrem oneskorenia a plochy na čipe (spomínaných v kapitole 2), ktoré sa viažu ku konkrétnym použitým hradlám, možno pri komplexnejších zapojeniach týchto hradiel (teda pri obvodoch) brať do úvahy aj parametre ako *počet úrovní komponentu (logic depth)*, *rozvetvenie (fanout)* a *počet prepojení (wiring tracks)* [12]. Pod pojmom *počet úrovní komponentu* rozumieme číslo vyjadrujúce, koľko stupňov logických členov komponent obsahuje. Hodnota *rozvetvenia* udáva najväčší počet vstupov v obvode, na ktoré je privedený jeden konkrétny signál a *počet prepojení* je najväčší počet drôtov, ktoré sú vedené súčasne, medzi dvomi po sebe nasledujúcimi stupňami logiky.

V tejto kapitole si prostredníctvom implementovaného generátoru aritmetických obvodov (z ním vygenerovaných zoznamov hradiel) odmeriame a porovnáme vyššie uvedené hodnoty charakteristické pre konkrétne komponenty. Toto zabezpečujú moduly v balíku `component_features_lib`. Jednotlivé funkcie z tohto balíku sú potom volané v spustiteľnom module `component_features`. Čo sa porovnania výsledkov týka, namerané hodnoty sú uvedené prostredníctvom radarových grafov. Pre ich vytvorenie bola použitá knižnica `matplotlib` [13] a ich zdrojové kódy sa nachádzajú v zložke `charts`. V nasledujúcich podkapitolách si teda uveďme tieto grafy pre šírky vstupných signálov 8 bitov, 16 bitov, 32 bitov a 64 bitov jednotlivých obvodov.

Vopred poznamenajme, keďže táto záležitosť sa týka všetkých grafov, že hodnota oneskorenia a hodnota udávajúca počet úrovní majú takmer vždy rovnaké pomery bez ohľadu na to, aké komponenty sa práve porovnávajú. Je to celkom logické, pretože oneskorenie je časové ohodnotenie väčšinou najdlhšej cesty v obvode a tá s počtom úrovní rozhodne súvisí. V grafoch to teda znamená to, že plochy, ktoré tieto vlastnosti reprezentujú, sú s výnimkou malých odchýlok stále prekryté.

5.1 Porovnanie sčítačiek

Pri sčítačkách urobíme tri porovnania. V prvom prípade budú porovnávané sčítačky uvedené v podkapitole 2.2, v prípade druhom to budú topológie sčítačiek Tree Adders a nakoniec topológie sčítačiek Higher Valency Tree Adders.

5.1.1 Porovnanie rôznych typov

V tejto podkapitole si porovnáme parametre sčítačky s postupným prenosom (ďalej ako RCA), sčítačky s predikciou prenosu (CLA), sčítačky s výberom prenosu (CSelA) a sčítačky s uchovaním

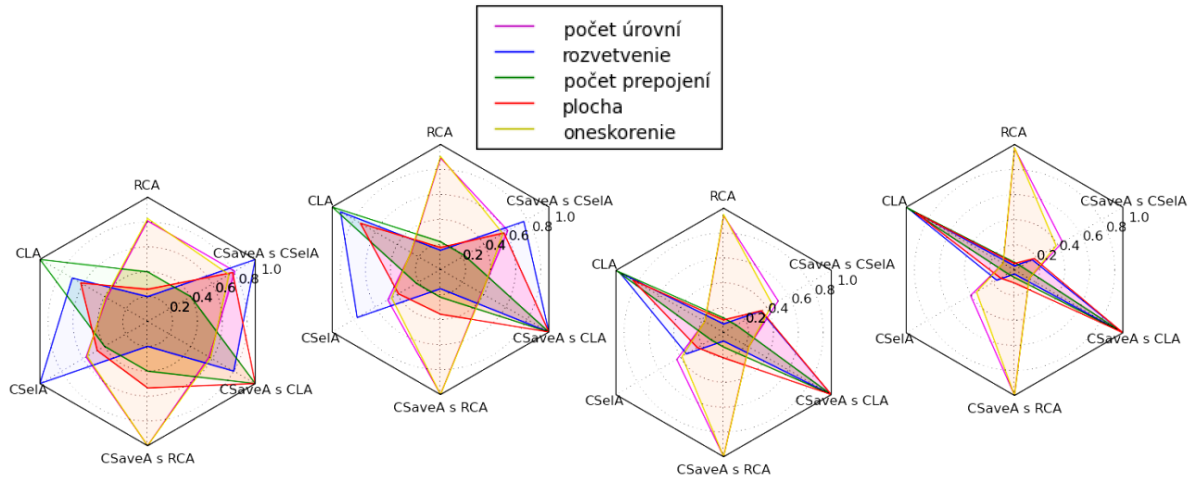
prenosu (CSaveA) s rôznymi variantmi použitej sčítacej logiky (ako sčítacia logika budú použité práve RCA, CLA a CSelA).

Z Obrázku 5.1 možno badať, že vo všetkých variantoch šírok vstupných signálov je z pohľadu plochy najvýhodnejšia RCA. Najviac miesta zasa zaberá CSaveA s použitím CLA, ale tento fakt je spôsobený práve použitím CLA. Hodnota jej plochy je totižto tiež vysoká v porovnaní s ostatnými sčítačkami (najmä pri 32-bitových a 64-bitových variantoch).

V oneskorení naopak CLA víťazí (tým pádom aj CSaveA s použitím CLA). Najhoršou je v tomto meradle RCA (čo sa premieta aj do oneskorenia CSaveA s jej použitím). Ako už bolo spomenuté, oneskorenie a počet úrovní majú pomery takmer rovnaké, takže vyhodnotenie oneskorenia sa vzťahuje aj na túto vlastnosť.

Najmenšie hodnoty rozvetvenia má RCA. Pri 8-bitových variantoch má hodnotu rozvetvenia najhoršiu CSelA, pri väčších šírkach ale túto štafetu preberá CLA (analogicky teda aj CSaveA s použitím CLA).

Pri hodnotách počtov prepojení sú najlepšími hneď dva typy – RCA a CSelA. Z toho vyplýva, že najhoršie bude na tom v tomto ukazovateli CLA, resp. CSaveA práve s použitím CLA.



Obrázok 5.1: Parametre rôznych typov sčítačiek pre šírky vstupov: (a) 8 bitov, (b) 16 bitov, (c) 32 bitov a (d) 64 bitov

Ak by sme teda mali zhodnotiť predchádzajúce, s najhoršími vlastnosťami sa vo všetkých faktoroch spájala CSaveA. Je to ale spôsobené tým, že na rozdiel od ostatných spomenutých typov sčítačiek je táto trojoperandová (zatiaľ čo ostatné majú operandy iba dva) a zároveň v každom svojom variante používa práve jednu z dvojoperandových sčítačiek. Z týchto má pri šírkach 8 bitov a 16 bitov každá svoje klady a zápory, no pri 32-bitových a 64-bitových šírkach vstupov je najlepším kompromisom z pohľadu všetkých spomenutých faktorov CSelA (viď Obrázok 5.1 (c) a (d)).

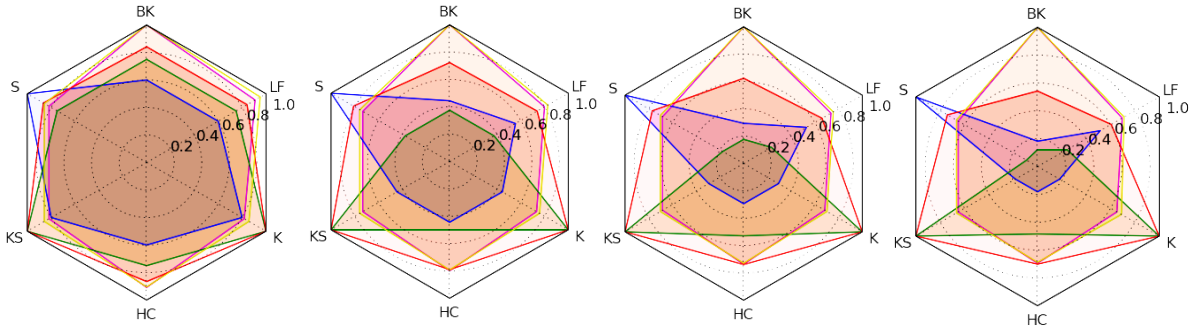
5.1.2 Porovnanie topológií sčítačky Tree Adder

Na nasledujúcich riadkoch sa pozrieme na vlastnosti jednotlivých topológií, ktoré používa sčítačka Tree Adder. Ide o topológie Brent-Kung (ďalej iba BK), Sklansky (S), Kogge-Stone (KS), Han-Carlson (HC), Knowles (K) a Ladner-Fischer (LF). Vychádzať pritom budeme z grafov na Obrázku 5.2.

Topológia BK je najvýhodnejšia z pohľadu plochy vo všetkých prípadoch šírok vstupov. Najviac plochy zaberajú topológie dve – KS a K. Čo sa oneskorenia a počtu úrovní týka, najlepšími sú zhodne topológie S, KS a K, najhoršou zasa BK.

Pri 8-bitovej šírke vstupných signálov majú z pohľadu rozvetvenia až tri topológie najlepšie vlastnosti – HC, LF a BK. Pri ostatných šírkach vstupov sú rovnako výhodné všetky topológie s výnimkou topológií LF a S (tá je celkovo najhoršou v tomto meradle).

V počte prepojení dominujú (čo sa efektivity týka) topológie BK a S (pri 8-bitových a 16-bitových variantoch sa k nim ešte pridáva topológia LF). Najhoršími sú topológie KS a K.



(a) (b) (c) (d)

Obrázok 5.2: Parametre topológií sčítačky Tree Adder pre šírky vstupov: (a) 8 bitov, (b) 16 bitov, (c) 32 bitov a (d) 64 bitov

Zo spomenutých topológií reprezentuje takmer každá maximum niektorej konkrétnej vlastnosti. Sú tu ale dve výnimky - topológia HC a topológia LF. Práve preto ich môžeme považovať za určité kompromisy, čo sa rozoberaných faktorov týka.

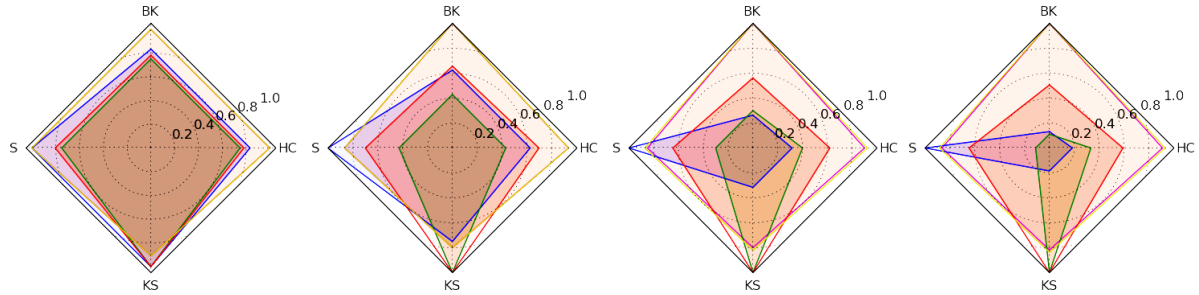
5.1.3 Porovnanie topológií sčítačky Higher Valency Tree Adder

Podobne ako v predchádzajúcej podkapitole, aj v tejto budeme porovnávať topológie, no tento krát nimi budú topológie sčítačky Higher Valency Tree Adder. Sú to topológia Brent-Kung (ďalej ako BK), Sklansky (S), Kogge-Stone (KS) a Han-Carlson. Ich vlastnosti sú načrtnuté na Obrázku 5.3.

Rozsah plochy má pre všetky šírky najmenší topológia BK. Pri šírke vstupných signálov 8 bitov sa o pomyselné prvenstvo delí s topológiami S a HC. Najhoršou je pre všetky šírky topológia KS. Pri oneskorení a počte prepojení sú výsledky presne opačné – najvýhodnejšou je topológia KS

a najmenej výhodnou naopak topológia BK. Najmenšiu hodnotu rozvetvenia dosahuje topológia BK (pri šírkach vstupov 8 bitov a 16 bitov spolu s ňou aj topológia HC), hodnotu najväčšiu zasa topológia S.

Topológie BK a S majú celkovo najmenšie hodnoty počtov prepojení. Pri 8-bitovej a 16-bitovej sčítačke sa k nim radí aj topológia HC. Z toho vyplýva (keďže topológie sú celkom štyri), že najväčšie čísla v rámci počtov prepojení dosahuje topológia KS.



(a) (b) (c) (d)

Obrázok 5.3: Parametre topológií sčítačky Higher Valency Tree Adder pre šírky vstupov: (a) 8 bitov, (b) 16 bitov, (c) 32 bitov a (d) 64 bitov

Za kompromis spomedzi uvedených topológií možno zvoliť topológiu HC. Pri nej sa totiž nenachádza žiadna z uvažovaných vlastností v jej maxime.

5.2 Porovnanie násobičiek

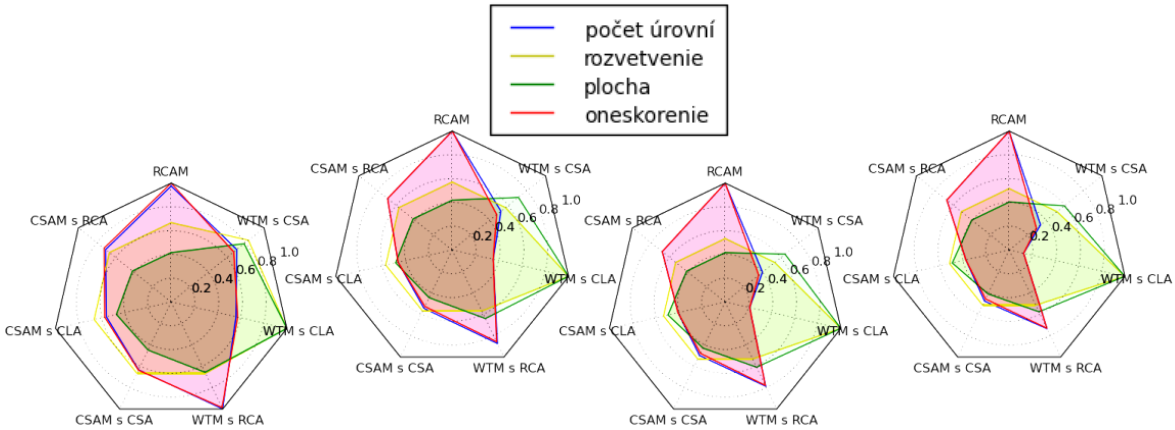
Pri násobičkách vykonáme, na rozdiel od sčítačiek, porovnanie iba jedno. V rámci tohto porovnania budeme uvažovať násobičky spomínané v podkapitole 2.3. Sú nimi kombinačná násobička (ďalej už iba RCAM), násobička s uchovaním prenosu (CSAM) a násobička s využitím Wallaceovho stromu (WTM). Posledné dve menované používajú pre výpočet súčinu i dvojbitovú sčítaciu logiku. Tou bude sčítačka s postupným prenosom (RCA), sčítačka s predikciou prenosu (CLA) alebo sčítačka s výberom prenosu (CSA). Porovnanie jednotlivých variantov násobičiek demonštrujú grafy na Obrázku 5.4. V týchto grafoch sa nenachádza plocha reprezentujúca hodnoty počtu prepojení. Je to z toho dôvodu, že pre všetky uvedené varianty násobičiek je pri každej šírke vstupných signálov táto hodnota rovnaká.

Z pohľadu plochy je najmenej zaberajúca RCAM a CSAM s použitím RCA. Pri 64-bitovom variante je to ešte aj CSAM využívajúca CSA. Najviac miesta zasa zaberie WTM s CLA ako sčítacou logikou.

Pri 8-bitových vstupných operandoch je, čo sa oneskorenia a počtu úrovní týka, najvýhodnejšou WTM s použitím CLA spolu s CSAM taktiež s použitím CLA. Pri tejto šírke sú najhoršie varianty dve – RCAM a WTM s využitím RCA. Pri ostatných šírkach vstupných signálov

možno badať už len dominanciu WTM s CLA (pozoruhodný pomer je najmä pri porovnaní 64-bitových variantov). Najhoršou je v týchto prípadoch RCAM.

RCAM a CSAM (bez ohľadu na jej sčítaciu logiku) dosahujú najefektívnejšie hodnoty rozvetvenia. Vo väčšine (okrem toho 8-bitového) variantov sa k nim pridáva i WTM, a to s RCA a CSA ako sčítacími logikami. Z toho vyplýva, že najhoršou v tomto ukazovateli je WTM s použitím CLA.



(a) (b) (c) (d)

Obrázok 5.4: Parametre rôznych typov násobičiek pre šírky vstupov: (a) 8 bitov, (b) 16 bitov, (c) 32 bitov a (d) 64 bitov

Pri pohľade na grafy na Obrázku 5.4 môžeme usúdiť, že najideálnejšími kompromismi, berúc do úvahy faktory, ktoré sme porovnávali, sú CSAM so sčítacou logikou CLA alebo CSA. Obe majú hodnoty spomenutých vlastností takmer rovnaké, nehovoriac o ich nízkej hodnote v porovnaní s ostatnými typmi sčítačiek.

6 Záver

Cieľom práce bola implementácia nástroja pre generovanie aritmetických obvodov. Ten sa podarilo úspešne naplniť, navrhnutý a implementovaný bol plne funkčný generátor.

Základným kameňom úspechu bol rozbor konkrétnych aritmetických komponentov (sčítačiek a násobičiek). Najmä z neho sa čerpali informácie a fakty, ktoré boli podstatné pri návrhu. Podstatným míľnikom práce tiež bolo vytvorenie testov pre nástroj a ich následné splnenie z pohľadu generátora. Bez nich by jeho funkcionalita nemohla byť potvrdená. Prínosom je porovnanie parametrov jednotlivých typov sčítačiek a násobičiek prostredníctvom grafov. Údaje pre ne boli získané práve z nástrojom vygenerovaných obvodov.

Existuje určite mnoho rozšírení, ktoré sa pre generátor núkajú. Ako príklady uveďme doplnenie nových tried reprezentujúcich doposiaľ neimplementované aritmetické obvody, reprezentáciu vytvoreného obvodu v jazykoch popisujúcich hardvér (napr. VHDL či Verilog) alebo jednoduché užívateľské rozhranie generátoru.

Použitá literatúra

[1] LU, Mi. Arithmetic and Logic in Computer Systems. Hoboken, New Jersey: John Wiley & Sons, Inc., 2004. ISBN 0-471-46945-9.

[2] Ripple carry adder, 4 bit ripple carry adder circuit , propagation delay. *Electronic Circuits and Diagram-Electronics Projects and Design* [online]. 2014 [cit. 2014-10-11]. Dostupné z: http://www.circuitstoday.com/ripple-carry-adder

[3] VIJAYAN, Vidya, M. MOHANAPRIYA a Sharon PAUL. Area, Delay and Power Comparison of Adder Topologies. In: *International Journal of VLSI design & Communication Systems* [online]. 2012 [cit. 2014-11-03]. Dostupné z: http://airccse.org/journal/vlsi/papers/3112vlsics13.pdf

[4] BHOWMICK, Sabyasachi a P. Mohan KUMAR. Efficient implementation of Carry Save Adder. In: *Academia.edu - Share research* [online]. 2014 [cit. 2014-10-18]. Dostupné z: http://www.academia.edu/7371421/Efficient_implementation_of_Carry_Save_Adder

[5] DAMLE, M. B., Dr. S. LIMAYE a M. G. SONWANI. Comparative Analysis of Array Multiplier Using Different Logic Styles. In: *IOSR Journal of Engineering* [online]. 2013 [cit. 2014-04-02]. Dostupné z: http://www.iosrjen.org/Papers/vol3_issue5%20%28part-2%29/D03521622.pdf

[6] STINE, James E. *Digital computer arithmetic datapath design using Verilog HDL*. Boston: Kluwer Academic Publishers, 2004. ISBN 1-4020-7710-6.

[7] MATHIASSEN, Stian. *Power optimized multipliers* [online]. Trondheim, 2010 [cit. 2015-04-03]. Dostupné z: http://www.diva-portal.org/smash/get/diva2:349523/FULLTEXT01.pdf. Master thesis.

[8] BANSAL, Himanshu a K. G. SHARMA. Wallace Tree Multiplier Designs: A Performance Comp arison Review. In: *Innovative Systems Design and Engineering* [online]. 2014 [cit. 2015-04-05]. Dostupné z: http://www.iiste.org/Journals/index.php/ISDE/article/viewFile/12916/13257

[9] CORMEN, Thomas H. *Introduction to algorithms* [online]. Cambridge, Mass.: MIT Press, 2009 [cit. 2015-04-05]. ISBN 0262533057.

[10] SpiceVision PRO - High Capacity Transistor-Level Debugger and Viewer. *Concept Engineering - We make things visible* [online]. 2014 [cit. 2015-05-15]. Dostupné z: http://www.concept.de/SpiceVision.html

[11] WESTE, Neil H a David Money HARRIS. *CMOS VLSI design: a circuits and systems perspective*. 4th ed. Boston: Addison Wesley, 2011. ISBN 0321547748.

[12] PATIL, Dinesh, Omid AZIZI, Mark HOROWITZ, Ron HO a Rajesh ANANTHRAMAN. Robust Energy-Efficient Adder Topologies. In: *Computer*

Arithmetic, 2007. ARITH '07. 18th IEEE Symposium on [online]. 2007 [cit. 2014-10-08]. ISBN 0-7695-2854-6ISSN 1063-6889. DOI: 10.1109/ARITH.2007.31. Dostupné z: https://www.lirmm.fr/arith18/papers/patil-RobustEnergyEffcientAdder.pdf

[13] *Matplotlib: python plotting - Matplotlib 1.4.3 documentation* [online]. 2012 [cit. 2015-05-15]. Dostupné z: http://matplotlib.org/index.html

Príloha A

Obsah CD

Priložený CD nosič obsahuje:

- zložku `source_codes`, v ktorej sa nachádzajú zdrojové kódy generátoru aritmetických obvodov, ako aj zdrojové kódy všetkých ostatných skriptov použitých k zisku dát uvedených v tejto práci
- zložku `report_sources`, v ktorej sa nachádza elektronická verzia tejto práce spolu s jej zdrojovým súborom
- súbor `readme.txt` bližšie popisujúci obsah vyššie spomenutých zložiek

Príloha B

Tabuľky časových vlastností generátoru

Časové vlastnosti vybraných sčítačiek

Typ	Šírka [bit]	Čas vytvorenia inštancie triedy [s]	Čas generovania Python kódu [s]	Čas generovania C kódu [s]	Čas generovania kódu pre SpiceVision[s]
Sčítačka s postupným prenosom	8	2.4795×10^{-4}	1.5997×10^{-4}	1.6188×10^{-4}	3.6597×10^{-4}
	16	5.8007×10^{-4}	2.9993×10^{-4}	3.0994×10^{-4}	7.9202×10^{-4}
	32	1.6691×10^{-3}	5.8603×10^{-4}	5.9604×10^{-4}	2.0160×10^{-3}
	64	5.5181×10^{-3}	1.1379×10^{-3}	1.1899×10^{-3}	5.8491×10^{-3}
Sčítačka s predikciou prenosu	8	8.7690×10^{-4}	37097×10^{-4}	3.8695×10^{-4}	5.8007×10^{-4}
	16	3.8070×10^{-3}	1.2769×10^{-3}	1.3079×10^{-3}	1.7349×10^{-3}
	32	3.2001×10^{-2}	5.2709×10^{-3}	5.1639×10^{-3}	6.3970×10^{-3}
	64	4.4239×10^{-1}	2.1788×10^{-2}	2.1801×10^{-2}	2.5245×10^{-2}
Sčítačka s výberom prenosu	8	7.9298×10^{-4}	2.8800×10^{-4}	2.8681×10^{-4}	1.0590×10^{-3}
	16	3.2129×10^{-3}	6.9594×10^{-4}	7.0691×10^{-4}	3.8959×10^{-3}
	32	1.3100×10^{-2}	1.4469×10^{-3}	1.5020×10^{-3}	1.4334×10^{-2}
	64	5.2429×10^{-2}	3.0109×10^{-3}	3.0009×10^{-3}	5.4966×10^{-2}
Sčítačka s uchovaním prenosu (používajúc sč. s post. pren.)	8	7.5602×10^{-4}	3.1495×10^{-4}	3.1709×10^{-4}	1.0211×10^{-3}
	16	1.9578×10^{-3}	6.1297×10^{-4}	6.0510×10^{-4}	2.4161×10^{-3}
	32	6.0667×10^{-3}	1.1479×10^{-3}	1.1589×10^{-3}	6.4411×10^{-3}
	64	2.1039×10^{-2}	2.2201×10^{-3}	2.2211×10^{-3}	2.0423×10^{-2}
Sčítačka s uchovaním prenosu (používajúc sč. s pred. pren.)	8	1.5928×10^{-3}	6.0892×10^{-4}	6.1607×10^{-4}	1.3411×10^{-3}
	16	6.2489×10^{-3}	1.7678×10^{-3}	1.7750×10^{-3}	3.5820×10^{-3}
	32	4.3892×10^{-2}	5.9878×10^{-3}	5.9828×10^{-3}	1.1084×10^{-2}
	64	4.9501×10^{-1}	2.3645×10^{-2}	2.3519×10^{-2}	4.0659×10^{-2}
Sčítačka s uchovaním prenosu (používajúc sč. s výb. pren.)	8	2.1610×10^{-3}	4.6896×10^{-4}	4.8303×10^{-4}	2.5329×10^{-3}
	16	7.0841×10^{-3}	9.9992×10^{-4}	9.9992×10^{-4}	7.6830×10^{-3}
	32	2.5849×10^{-2}	2.0461×10^{-3}	2.0542×10^{-3}	2.5654×10^{-2}
	64	9.8017×10^{-2}	4.1029×10^{-3}	4.1048×10^{-3}	9.2925×10^{-2}

Časové vlastnosti topológií sčítačky Tree Adder

Topológia	Šírka [bit]	Čas vytvorenia inštancie triedy [s]	Čas generovania Python kódu [s]	Čas generovania C kódu [s]	Čas generovania kódu pre SpiceVision[s]
Brent-Kung	8	1.1408×10^{-3}	2.7394×10^{-4}	2.4485×10^{-4}	1.8200×10^{-3}
	16	2.9661×10^{-3}	4.9710×10^{-4}	5.0282×10^{-4}	5.7439×10^{-3}
	32	1.0778×10^{-2}	1.0330×10^{-3}	1.0218×10^{-3}	2.0265×10^{-2}
	64	4.1714×10^{-2}	2.0749×10^{-3}	2.0980×10^{-3}	7.6757×10^{-2}
Sklansky	8	9.7012×10^{-4}	2.3698×10^{-4}	2.5510×10^{-4}	1.8298×10^{-3}
	16	3.5591×10^{-3}	5.7721×10^{-4}	5.6600×10^{-4}	6.5820×10^{-3}
	32	1.6407×10^{-2}	1.2571×10^{-3}	1.2509×10^{-3}	2.8475×10^{-2}
	64	7.9241×10^{-2}	2.8338×10^{-3}	2.8479×10^{-3}	1.3343×10^{-1}
Kogge-Stone	8	1.0440×10^{-3}	2.7394×10^{-4}	2.8586×10^{-4}	1.8990×10^{-3}
	16	4.2991×10^{-3}	6.8712×10^{-4}	6.8688×10^{-4}	7.3680×10^{-3}
	32	2.1644×10^{-2}	1.6341×10^{-3}	1.6260×10^{-3}	3.3665×10^{-2}
	64	1.1268×10^{-1}	3.7400×10^{-3}	3.7519×10^{-3}	1.6767×10^{-1}
Han-Carlson	8	9.8395×10^{-4}	2.4199×10^{-4}	2.5010×10^{-4}	1.8079×10^{-3}
	16	3.2119×10^{-3}	5.4883×10^{-4}	5.5503×10^{-4}	6.0031×10^{-3}
	32	1.3472×10^{-2}	1.2009×10^{-3}	1.2469×10^{-3}	2.3390×10^{-2}
	64	6.0833×10^{-2}	2.6469×10^{-3}	2.6788×10^{-3}	1.0241×10^{-1}
Knowles	8	1.1100×10^{-3}	2.7203×10^{-4}	2.8586×10^{-4}	1.9059×10^{-3}
	16	4.3499×10^{-3}	6.6518×10^{-4}	6.6900×10^{-4}	7.3990×10^{-3}
	32	2.1497×10^{-2}	1.5909×10^{-3}	1.6109×10^{-3}	3.3707×10^{-2}
	64	1.1310×10^{-1}	3.7870×10^{-3}	3.7760×10^{-3}	1.6656×10^{-1}
Ladner-Fischer	8	9.6511×10^{-4}	2.3603×10^{-4}	2.7203×10^{-4}	1.7979×10^{-3}
	16	3.0710×10^{-3}	4.9686×10^{-4}	5.0592×10^{-4}	5.8197×10^{-3}
	32	1.1888×10^{-2}	1.0881×10^{-3}	1.0859×10^{-3}	2.1925×10^{-2}
	64	5.1720×10^{-2}	2.3021×10^{-3}	2.3241×10^{-3}	9.2801×10^{-2}

Časové vlastnosti topológií sčítačky Higher Valency Tree Adder

Topológia	Šírka [bit]	Čas vytvorenia inštancie triedy [s]	Čas generovania Python kódu [s]	Čas generovania C kódu [s]	Čas generovania kódu pre SpiceVision[s]
Brent-Kung	8	1.0049×10^{-3}	2.2983×10^{-4}	2.4104×10^{-4}	1.5900×10^{-3}
	16	2.7699×10^{-3}	4.9996×10^{-4}	5.0282×10^{-4}	4.9450×10^{-3}
	32	1.0139×10^{-2}	1.0690×10^{-3}	1.1169×10^{-3}	1.7327×10^{-2}
	64	4.0096×10^{-2}	2.2680×10^{-3}	2.2408×10^{-3}	6.5392×10^{-2}
Sklansky	8	8.2015×10^{-4}	2.3412×10^{-4}	2.4318×10^{-4}	1.4500×10^{-3}
	16	2.6621×10^{-3}	5.2595×10^{-4}	5.2809×10^{-4}	4.5871×10^{-3}
	32	1.0277×10^{-2}	1.2109×10^{-3}	1.2350×10^{-3}	1.6581×10^{-2}
	64	4.7904×10^{-2}	2.7639×10^{-3}	2.8231×10^{-3}	7.1357×10^{-2}
Kogge-Stone	8	9.8299×10^{-4}	2.9087×10^{-4}	2.8991×10^{-4}	1.6059×10^{-3}
	16	4.3349×10^{-3}	7.7605×10^{-4}	7.8797×10^{-4}	6.0038×10^{-3}
	32	2.2541×10^{-2}	1.9102×10^{-3}	1.9059×10^{-3}	2.6540×10^{-2}
	64	9.7339×10^{-2}	4.3258×10^{-3}	4.3599×10^{-3}	1.0254×10^{-1}
Han-Carlson	8	8.8596×10^{-4}	2.3007×10^{-4}	2.5892×10^{-4}	1.5769×10^{-3}
	16	2.9678×10^{-3}	5.2690×10^{-4}	5.3191×10^{-4}	5.0919×10^{-3}
	32	1.1427×10^{-2}	1.1770×10^{-3}	1.2080×10^{-3}	1.8276×10^{-2}
	64	4.6621×10^{-2}	2.6049×10^{-3}	2.6161×10^{-3}	6.8455×10^{-2}

Časové vlastnosti vybraných násobičiek

Typ	Šírka [bit]	Čas vytvorenia inštancie triedy [s]	Čas generovania Python kódu [s]	Čas generovania C kódu [s]	Čas generovania kódu pre SpiceVision[s]
Kombinačná násobička	8	5.2800×10^{-3}	1.0991×10^{-3}	1.1031×10^{-3}	4.7941×10^{-3}
	16	8.3698×10^{-2}	4.5769×10^{-3}	4.5320×10^{-3}	5.8769×10^{-2}
	32	1.3824	1.8794×10^{-2}	1.9159×10^{-2}	8.9704×10^{-1}
	64	25.8646	7.9859×10^{-2}	7.9343×10^{-2}	16.2918
Násobička s uchovaním prenosu (používajúc sč. s post. pren.)	8	5.5899×10^{-3}	1.2209×10^{-3}	1.1928×10^{-3}	5.1441×10^{-3}
	16	8.4113×10^{-2}	4.7760×10^{-3}	4.8158×10^{-3}	6.1269×10^{-2}
	32	1.3766	1.9844×10^{-2}	1.9787×10^{-2}	9.2557×10^{-1}
	64	26.3659	7.9854×10^{-2}	8.0587×10^{-2}	16.7085
Násobička s uchovaním prenosu (používajúc sč. s pred. pren.)	8	6.2899×10^{-3}	1.3780×10^{-3}	1.3751×10^{-3}	5.5191×10^{-3}
	16	8.8057×10^{-2}	5.6860×10^{-3}	5.7418×10^{-3}	6.2794×10^{-2}
	32	1.4950	2.4137×10^{-2}	2.4572×10^{-2}	9.3266×10^{-1}
	64	27.1575	9.9154×10^{-2}	1.0038×10^{-1}	16.7494
Násobička s uchovaním prenosu (používajúc sč. s výb. pren.)	8	7.7300×10^{-3}	1.3248×10^{-3}	1.3160×10^{-3}	6.9720×10^{-3}
	16	1.1011×10^{-1}	5.1600×10^{-3}	5.1090×10^{-3}	7.8440×10^{-2}
	32	1.5900	2.0366×10^{-2}	2.0881×10^{-2}	1.0681
	64	28.9131	8.1296×10^{-2}	8.1959×10^{-2}	18.0325
Násobička s využitím Wall. stromu (používajúc sč. s post. pren.)	8	1.8273×10^{-2}	1.8570×10^{-3}	1.8529×10^{-3}	1.4254×10^{-2}
	16	2.7283×10^{-1}	7.4379×10^{-3}	7.3800×10^{-3}	1.7681×10^{-1}
	32	4.1938	2.9277×10^{-2}	2.9630×10^{-2}	2.5767
	64	76.4162	1.1893×10^{-1}	1.1850×10^{-1}	44.3930
Násobička s využitím Wall. stromu (používajúc sč. s pred. pren.)	8	2.2257×10^{-2}	2.9489×10^{-3}	2.9439×10^{-3}	1.5600×10^{-2}
	16	3.0777×10^{-1}	1.2132×10^{-2}	1.2185×10^{-2}	1.8382×10^{-1}
	32	4.6558	5.2196×10^{-2}	5.2181×10^{-2}	2.5607
	64	84.2637	2.1775×10^{-1}	2.1651×10^{-1}	43.8575
Násobička s využitím Wall. stromu (používajúc sč. s výb. pren.)	8	2.9850×10^{-2}	2.3639×10^{-3}	2.3670×10^{-3}	2.3680×10^{-2}
	16	3.7158×10^{-1}	8.5320×10^{-3}	8.5289×10^{-3}	2.4821×10^{-1}
	32	5.3975	3.1687×10^{-2}	3.2814×10^{-2}	3.1399
	64	83.6459	1.2293×10^{-1}	1.2425×10^{-1}	49.4625

Časové vlastnosti obvodu počítajúceho absolútny rozdiel

Šírka [bit]	Čas vytvorenia inštancie triedy [s]	Čas generovania Python kódu [s]	Čas generovania C kódu [s]	Čas generovania kódu pre SpiceVision[s]
8	1.8301×10^{-3}	4.4202×10^{-4}	4.3702×10^{-4}	2.8409×10^{-3}
16	7.2231×10^{-3}	1.2359×10^{-3}	1.1248×10^{-3}	1.0579×10^{-2}
32	3.4796×10^{-2}	2.5091×10^{-3}	2.4819×10^{-3}	4.7255×10^{-2}
64	1.7411×10^{-1}	5.4340×10^{-3}	5.4278×10^{-3}	2.2335×10^{-1}