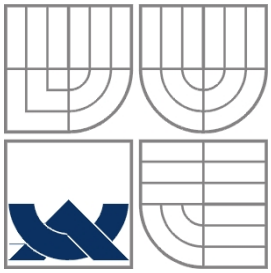
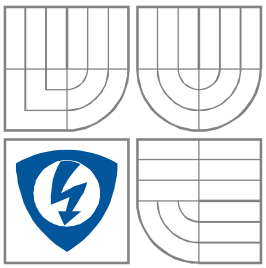


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV MIKROELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF MICROELECTRONICS

GENERÁTOR HARMONICKÝCH SIGNÁLŮ PRO SIMULACI MODELŮ ANALOGOVÝCH OBVODŮ

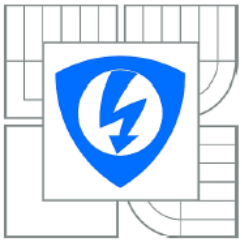
HARMONIC GENERATOR FOR SIMULATION OF ANALOG CIRCUITS MODELS

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE Jakub Tománek
AUTHOR

VEDOUCÍ PRÁCE Ing. Vojtěch Dvořák
SUPERVISOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav mikroelektroniky

Bakalářská práce

bakalářský studijní obor
Mikroelektronika a technologie

Student:	Jakub Tománek	***ID:***	158252
Ročník:	3	***Akademický rok:***	2014/2015

NÁZEV TÉMATU:

Generátor harmonických signálů pro simulaci modelů analogových obvodů

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s vlastnostmi čísel v plovoucí řádové čárce a jejich implementací v jazyce C++ a SystemC. Vyberte vhodné algoritmy pro generování harmonických signálů v plovoucí řádové čárce a popište je v jazyce C++ a SystemC. Proveďte implementaci zvolených algoritmů pomocí nástroje pro High-level syntézu do obvodů FPGA a ASIC. Na základě výsledků syntézy porovnejte jednotlivé algoritmy z hlediska plochy, maximální frekvence a celkové doby výpočtu jednoho vzorku.

DOPORUČENÁ LITERATURA:

Podle pokynů vedoucího práce

Termín zadání:	10.2.2015	***Termín odevzdání:***	4.6.2015

Vedoucí práce: Ing. Vojtěch Dvořák
Konzultanti bakalářské práce:

doc. Ing. Jiří Háze, Ph.D.
Předseda oborové rady

Prohlášení

Prohlašuji, že svou bakalářskou práci na téma **„Generátor harmonických signálů pro simulaci modelů analogových obvodů“** jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne 4. června 2015 .....

podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Vojtěchu Dvořákovi za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

V Brně dne 4. června 2015 .....

podpis autora

Abstrakt

Cílem této práce je prozkoumat možnosti generování diskrétního harmonického signálu na obvodu ASIC a FPGA za účelem simulace analogových obvodů. V teoretické části jsou rozebrány vybrané algoritmy, které se nejvíce hodí pro tuto aplikaci. V praktické části jsou jednotlivé algoritmy realizovány a je určen vhodný algoritmus pro danou aplikaci s ohledem na požadované vlastnosti signálu.

Abstract

The purpose of this paper is to explore the posibilities of genarating harmonic signal on a ASIC and FPGA chip with emphasis on simulation of analog circuits. In theoretical section the algoritms suited best for this purpose are expained. In practical section these algorithms are realized and a conclusion is drawn based on required properties of signal.

Klíčová slova:

BKM, CORDIC, Číslo s plovoucí desetinnou čárkou, FPGA, Behaviorální syntéza, Náhledová tabulka, SystemC

Keywords:

BKM, CORDIC, Floating point number, FPGA, High-Level Synthesis, Look-Up Table, SystemC

Bibliografická citace díla

TOMÁNEK, J. Generátor harmonických signálů pro simulaci modelů analogových modelů: bakalářská práce. Brno: FEKT VUT v Brně, 2014. 32 s.

OBSAH

Seznam obrázků

Úvod

Simulace velkých analogových obvodů je velmi drahá a časové náročná, a proto je snaha nalézt rychlejší a efektivnější řešení. Typickým příkladem takových obvodů jsou moderní obvody ASIC pracující ve smíšeném módu. Přístupy k problematice jejich simulace se různí podle úrovně abstrakce modelu. Čím vyšší je úroveň abstrakce, tím je simulace rychlejší a má nižší přesnost.

Tato práce je součástí řešení využívající obvodu FPGA a high-level syntézy. Podstata spočívá v modelování jednotlivých komponent obvodu v programovacím jazyce SystemC. Z jednotlivých částí mohou být následně sestaveny požadované obvody. Poté je automaticky provedena optimalizace a převod kódu do jazyka Verilog. Díky tomu by nebylo potřeba navrhovat pokaždé jiný speciální obvod pro simulaci a zároveň by čas simulace po implementaci do obvodu FPGA měl být výrazně nižší při zachování dobré přesnosti.

Cílem práce je najít vhodný způsob generování diskrétního harmonického signálu za účelem simulace analogových obvodů na obvodu FPGA. Výsledný signál musí být reprezentován čísly v plovoucí desetinné čárce.

Z podstaty úkolu vyplývá, že simulaci není potřeba provádět v reálném čase. Proto není možné uvažovat o frekvenci požadovaného harmonického signálu, ale pouze o počtu vzorků na periodu. Dalším podstatným bodem zadání je, že není potřeba počítat jednu konkrétní hodnotu funkce sinus, ale je potřeba generovat celý průběh. Toto je hlavní kritérium, na základě kterého byly algoritmy vybrány. Jsou mezi nimi i některé určené především pro výpočet konkrétních hodnot, které jsou upraveny pro generování průběhu. Algoritmus CORDIC je realizován jak v upravené, tak originální verzi z důvodu možného využití v jiných částech obvodu pro výpočet jedné hodnoty.

V závěrečné části práce bude provedeno srovnání implementace a efektivnosti jednotlivých přístupů a vybrán vhodný algoritmus pro dané aplikace.

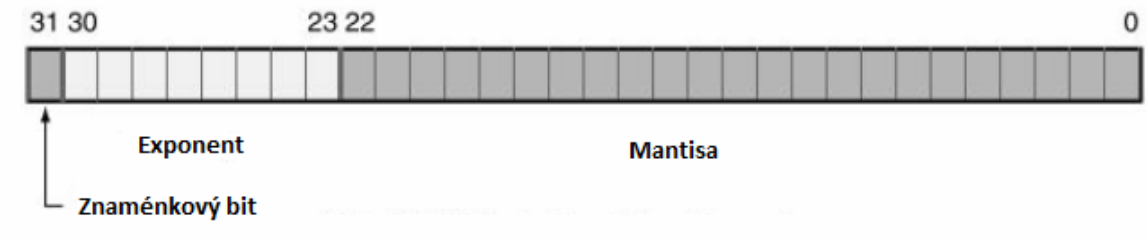
1 Čísla s plovoucí desetinnou čárkou

1.1 Reprezentace čísel s plovoucí desetinnou čárkou

Každé racionální číslo je možné vyjádřit jako číslo desetinné s nejvyšší cifrou na pozici jednotek vynásobené příslušnou mocninou počtu znaků číselné soustavy, v které je toto číslo reprezentováno. Například číslo 15/16 je možné zapsat jako: 9,375. 10^{-1}. Když se tedy omezíme na fixní počet platných cifer a bereme-li v úvahu pouze čísla ve dvojkové soustavě, je možné všechna reálná čísla vyjádřit ve tvaru:

$$A.1,C.2^{D} \qquad (1)$$

kde A je znaménkový bit, C je mantisa a D je exponent. Standard IEEE 754 definuje 32-bitové a 64-bitové čísla s plovoucí desetinou čárkou (single precision a double precision). Pro první platí bitové šířky C=23, D=8 a pro druhý C=52, D=11. Aby nebylo potřeba v exponentu využívat záporná čísla, je jako nulový exponent považována hodnota 2^{D-1}-1, ke které se přičítá nebo odečítá hodnota exponentu čísla. Na obrázku 1 je znázorněno bitové rozložení jednotlivých částí 32-bitového čísla dle standardu IEEE 754.



Obrázek 1: Bitové rozmístění jednotlivých částí čísla [1]

1.2 Speciální hodnoty

Ve standardu IEEE 754 existuje kladná a záporná nula. Je to proto, aby nedocházelo k chybám v některých aplikacích. Obě nuly se liší pouze ve znaménkovém bitu. Zbylé bity jsou všechny nulové.

V případě, že všechny bity v exponentu jsou rovny jedné a v mantise nule, je hodnota čísla ± nekonečno v závislosti na znaménkovém bitu.

Pokud jsou všechny bity v exponentu rovny jedné a v mantise jsou nenulové, jedná se o neurčitý výraz, který vzniká neplatnou matematickou operací.

1.3 Sčítání

Sčítání probíhá v následujících krocích:

1. Přepsání menšího čísla tak, aby exponent odpovídal číslu většímu.

2. Sečtení mantisy obou čísel.

3. Úprava výsledku do tvaru (1).

5. Kontrola přetečení exponentu.

4. Zaokrouhlení výsledku.

1.4 Násobení

Násobení také probíhá v několika krocích [2]:

1. Sečtení exponentů a odečtení konstanty 2^{D-1}-1.

2. Vynásobení mantis obu čísel.

3. Úprava výsledku do tvaru (1).

4. Zaokrouhlení výsledku.

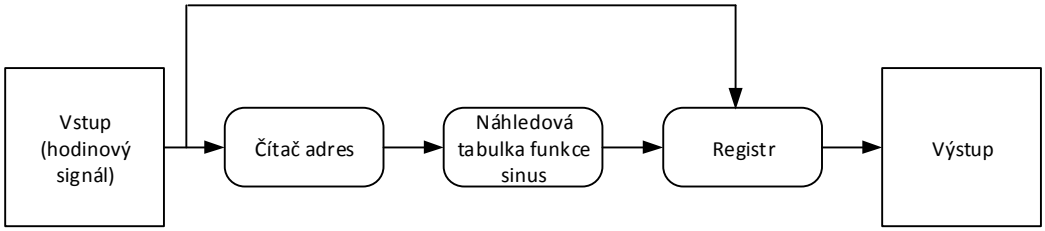
1.5 Knihovna sc_float

Praktická část této práce je provedena v programovacím jazyce SystemC za použití knihovny sc_float, která do tohoto jazyka přidává podporu čísel s plovoucí desetinnou čárkou. Při definici proměnné typu sc_float se zároveň definuje bitová šířka exponentu a mantisy této proměnné. Díky tomu je možné změnit bitovou šířku celého návrhu pouze přepsáním dvou čísel. Toho bylo využito v praktické části při měření vlastností jednotlivých algoritmů různých bitových šířek. Tato knihovna podporuje základní operace s čísly v plovoucí desetinné čárce.

2 Algoritmy pro generování harmonického signálu

2.1 Přímá číslicová syntéza

Přímá číslicová syntéza(Direct Digital Synthesis, dále jen DDS) je nejjednodušší algoritmus. V nejjednodušší formě může být implementován za pomocí referenčního hodinového signálu, čítače adres a náhledových tabulek (obrázek 2).



Obrázek 2: Základní schéma algoritmu DDS

V náhledových tabulkách jsou uloženy funkční hodnoty jedné čtvrtiny periody funkce sinus. Je to proto, aby byla paměťová náročnost tohoto algoritmu co nejnižší a čtvrtina periody je minimum potřebné k vygenerování celého průběhu. Z toho vyplývá omezení, že je nutné, aby požadovaný průběh měl celý počet vzorků na čtvrtinu periody. Blok čítač adres je obyčejný čítač od 0 do počtu uložených funkčních hodnot a zpět. Dále je potřeba přidat příznakový bit, který je negován vždy, když čítač nabude hodnoty 0. Podle toho se určí, jestli je výstupní hodnota signálu kladná či záporná. Přítomnost výstupního registru závisí na kompilátoru [3].

Výhodou tohoto algoritmu je, že není potřeba téměř žádné logiky (v podstatě pouze čítač), výstupní signál nemá žádné harmonické zkreslení a jednoduchost. Nevýhodou je, že při vyšších vzorkovacích frekvencích algoritmus zabere hodně paměťového místa.

Realizace tohoto algoritmu jsou dostupné v příloze 1 a 6.

2.2 2D rotace

Uvažujeme-li dvourozměrnou rotační matici:

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \tag{2}$$

vynásobíme-li tuto matici libovolným bodem v 2D prostoru, dojde k rotaci tohoto bodu okolo počátku souřadnic s úhlem Θ:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{3}$$

Po roznásobení matic získáme vztah pro souřadnice transformovaného bodu:

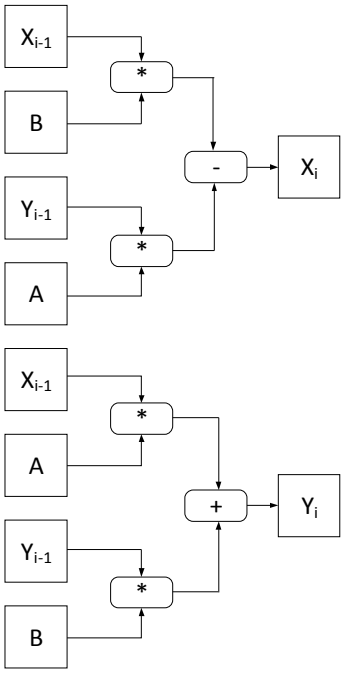
$$x' = x.\cos\theta - y.\sin\theta$$
$$y' = x.\sin\theta + y.\cos\theta \tag{4}$$

Pokud budou počáteční hodnoty x=1 a y=0, budou se při každé iteraci generovat v proměnné y funkce sinus a v proměnné x funkce cosinus. Jelikož úhel rotace je vždy konstantní, hodnoty sinΘ a cosΘ je možné přesně vypočítat ještě před syntézou obvodu. Tyto hodnoty jsou následně uloženy do proměnných A a B a celý výpočet se zjednoduší na:

$$x' = x.B - y.A$$
$$y' = x.A + y.B \tag{5}$$

což je obecný zápis výsledného algoritmu.

Výhodou algoritmu je opět vysoká přesnost a jednoduchost. Nevýhodou je, že je potřeba čtyřikrát násobení v plovoucí desetinné čárce. Na obrázku 1 je znázorněno blokové schéma. Realizace tohoto algoritmu jsou dostupné v příloze 2 a 7.



Obrázek 3: Algoritmus 2D rotace

2.3 Algoritmus CORDIC

Vytkneme-li v obou rovnicích soustavy (4) cosΘ, dostaneme výraz:

$$\begin{aligned} x' &= \cos\theta(x - y.tg\theta) \\ y' &= \sin\theta(x + y.tg\theta) \end{aligned} \tag{6}$$

Následně omezíme úhel Θ, aby splňoval:

$$tg\theta = \pm 2^{-i} \tag{7}$$

Nyní už předpokládáme, že se jedná o iterativní algoritmus, kde výslednou hodnotu funkce získáme pomocí rotací v kladném i záporném směru. Výraz cosΘ upravíme na:

$$\cos\theta = \frac{1}{\sqrt{1 + tg^2\theta}} = \frac{1}{\sqrt{1 + (\pm 2^{-i})^2}} = K_i \tag{8}$$

Jelikož tgΘ nabývá kladných nebo záporných hodnot, můžeme soustavu (6) upravit na:

$$\begin{aligned} \mathrm{x}' &= K_\mathrm{i}(x - y.d_i.2^{-i}) \\ y' &= K_i(y + x.d_i.2^{-i}) \end{aligned} \tag{9}$$

kde d_i nabývá hodnot ± 1. Nyní zavedeme třetí rovnici:

$$Z_{i+1} = Z_i - d_i.tg^{-1}(2^{-i}) \tag{10}$$

kde Z_0 je požadovaný úhel rotace. d_i je kladné v případě, že Z_i je větší než tento úhel. V opačném případě je záporné.

Z výrazu (9) je zjevné, že v případě, že i bude větší než bitová šířka x a y, všechny další iterace jsou zbytečné. Z toho důvodu je potřeba provádět stejný počet iterací, jako je bitová šířka x a y. Dále si můžeme všimnout, že multiplikační konstantu K_i je možné vytknout a násobení provést při první iteraci. V tom případě je možné tuto konstantu předen vypočítat:

$$K_n = \prod_{i=0}^{n-1} \frac{1}{\sqrt{1 + 2^{-2i}}} \tag{11}$$

$$K = \lim_{n\to\infty} K_n \approx 0{,}607252935 \tag{12}$$

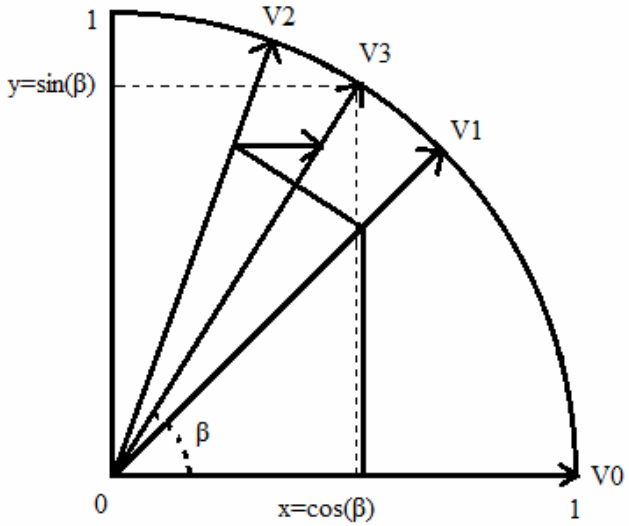
Výsledný vztah je:

$$x_{i+1} = x_i - y_i.d_i.2^{-i}$$

$$y_{i+1} = y_i + x_i.d_i.2^{-i} \qquad (13)$$

$$Z_{i+1} = Z_i - d_i.tg^{-1}(2^{-i})$$

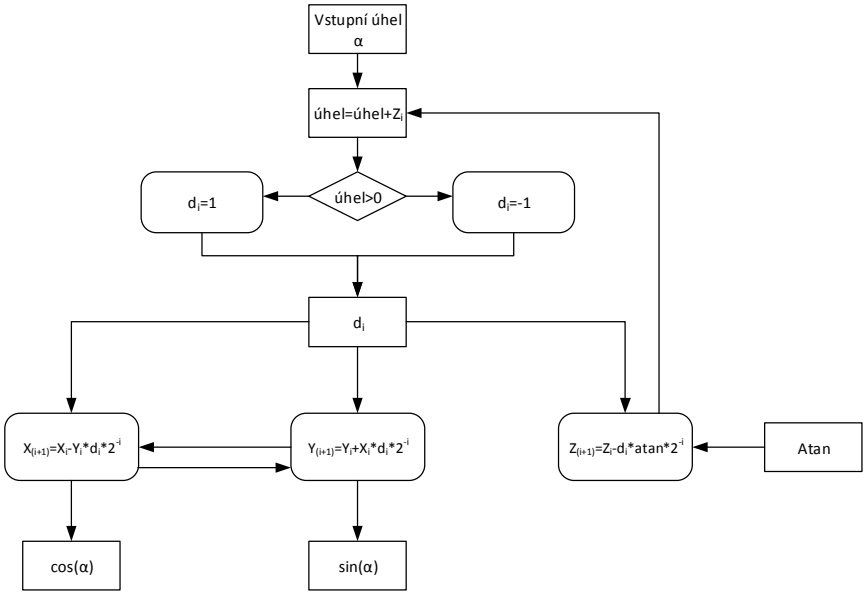
Pokud y_0=0 a x_0=K tak y_n= cosΘ a x_n= sinΘ. Graficky je znázorněn postup na obrázku 4.



Obrázek 4: Průběh algoritmu CORDIC

Při implementaci je nutné využít náhledové tabulku hodnot výrazu $tg^{-1}(2^{-i})$ pro všechna *i* od 0 do bitové šířky x a y. Díky tomu je při výpočtu funkce sinus zapotřebí pouze sčítání a bitový posun (shift-and-add algoritmus).

Výhodou algoritmu je, že není potřeba násobičky. Nevýhodou je vysoká výpočetní náročnost daná bitovým rozsahem čísla a nutnost náhledových tabulek pro funkci tangens [4]. Na obrázku 5 je znázorněno blokové schéma tohoto algoritmu. Realizace tohoto algoritmu jsou dostupné v příloze 3 a 8.



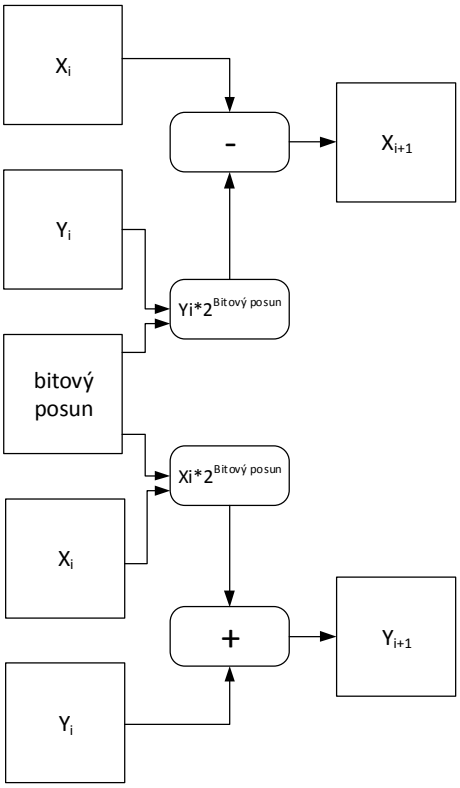
Obrázek 5: Algoritmus CORDIC

2.4 Upravený algoritmus CORDIC

Pokud úhel rotace omezíme přesně podle vztahu 7, algoritmus již přestává být iterativní, protože již v první iteraci je dosaženo přesné hodnoty. Proto třetí rovnice ve vztahu 13 zcela vypadne a taktéž členy d_i ve zbývajících dvou rovnicích, protože jsou vždy kladné. Nyní je možné při výpočtu každé další hodnoty funkce použít předchozí hodnoty. Je ovšem pořád nutné při každém (popřípadě při každém několikátém) výpočtu provést korekční násobení konstantou KU. Tato konstanta je rovna cosinu rotačního úhlu. Pro malé úhly rotace je možné tuto hodnotu aproximovat číslem jedna a tudíž vynechat násobení touto konstantou. Výsledný vztah vypadá takto:

$$\begin{aligned} x_{i+1} &= x_i - y_i.2^{-n} \\ y_{i+1} &= y_i + x_i.2^{-n} \end{aligned} \tag{14}$$

Nevýhodou je omezení rotace pouze na některé úhly (i když ve velkém rozsahu). Na obrázku 6 je znázorněno blokové schéma tohoto algoritmu. Realizace tohoto algoritmu jsou dostupné v příloze 4 a 9.



Obrázek 6: Upravený algoritmus CORDIC

2.5 Algoritmus BKM

Jedná se o komplexní shift-and-add algoritmus, který zobecňuje algoritmus CORDIC. Obecně se jedná o hledání posloupnosti $d_k \in \{-1,0,1\}$ takové, aby $x.\prod_{k=1}^{n}(1+d_k.2^{-k}) \approx 1$. Tento výraz můžeme upravit na:

$$\ln x \approx -\sum_{k=1}^{n} \ln(1+d_k.2^{-k}) \qquad (15)$$

což je v podstatě důkaz algoritmu BKM. Problém algoritmu CORDIC je, že pokud povolíme nulové hodnoty v posloupnosti d_i ze vztahu 13, tak multiplikační faktor K již není konstantní. U BKM algoritmu ovšem žádná multiplikační konstanta není a tím celý tento problém odpadá. Uvažujeme-li vztah 13 a nadefinujeme komplexní číslo L_n takové, že $L_n=x_n+iy_n$, dostaneme vztah $L_{n+1}=L_n(1+id_n.2^{-n})$. Pokud nyní nadefinujeme $d_n \in \{-1,0,1,-i,i,1-i,1+i,-1-i,-1+i\}$, kde $dn=dx+idy$ a uvažujeme vztah 15, získáme základní krok pro BKM algoritmus:

$$\begin{aligned} L_{n+1} &= L_n(1+d_n.2^{-n}) \\ E_{n+1} &= E_n - \ln(1+d_n.2^{-n}) \end{aligned} \qquad (16)$$

kde $E_n=Ex_n+iEy_n$.

Pokud najdeme posloupnost d_n takovou, že L_n konverguje k číslu 1, potom $E_n \rightarrow E_1+ln(L_1)$. Této iteraci se říká L-mód BKM algoritmu.

Pokud najdeme posloupnost d_n takovou, že E_n konverguje k číslu 0, potom $L_n \rightarrow L_1 e^{E_1}$. Této iteraci se říká E-mód BKM algoritmu.

V této práci se budu zabývat pouze E-módem, jelikož se pomocí něj počítají reálné funkce sinus, cosinus a 2D rotace. Jak již bylo zmíněno, je potřeba najít posloupnost d_n takovou, že E_n konverguje k číslu 0, což zahrnuje výpočet komplexní exponenciální funkce (E-mód). Když rozepíšeme vztah 16 na reálné a imaginární složky dostaneme:

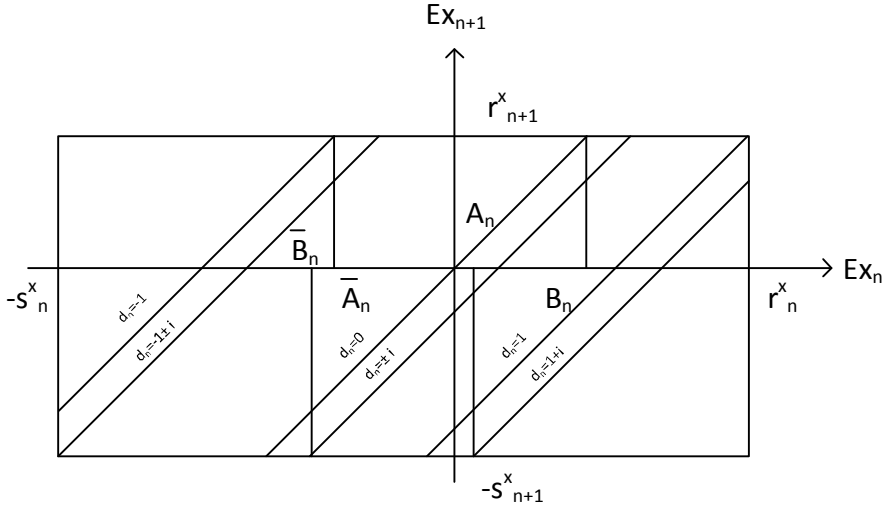
$$Lx_{n+1} = Lx_n + dx_n.Lx_n.2^{-n} - dy_n.Ly_n.2^{-n}$$

$$Ly_{n+1} = Ly_n + dy_n.Lx_n.2^{-n} + dx_n.Ly_n.2^{-n}$$

$$Ex_{n+1} = Ex_n - 0{,}5\ln(1 + dx_n.2^{-n+1} + (dx_n^2 + dy_n^2).2^{-2n}) \tag{17}$$

$$Ey_{n+1} = Ey_n - dy_n.arctg\left(\frac{2^{-n}}{1 + dx_n.2^{-n}}\right) \tag{18}$$

Předpokládáme, že Ex_n patří do intervalu $[-s_n^x, r_n^x]$. Pokud budeme rovnici 17 chápat jako lineární rovnici, kde průsečík s osou y je dán polovinou přirozeného logaritmu, který může nabývat pouze některých hodnot v závislosti na dx_n a dy_n, dostáváme šest rovnoběžných přímek. dx_n musí být zvoleno tak, aby $\forall\, Ex_{n+1} \in [-s_{n+1}^x, r_{n+1}^x] \wedge \lim_{n\to\infty} \left| s_n^x + r_n^x \right| = 0$ nezávisle na volbě dy_n. Tímto způsobem je sestaven Robertsonův diagram pro výběr dx_n (obrázek 7)



Obrázek 7: Robertsonův diagram pro výběr dx_n

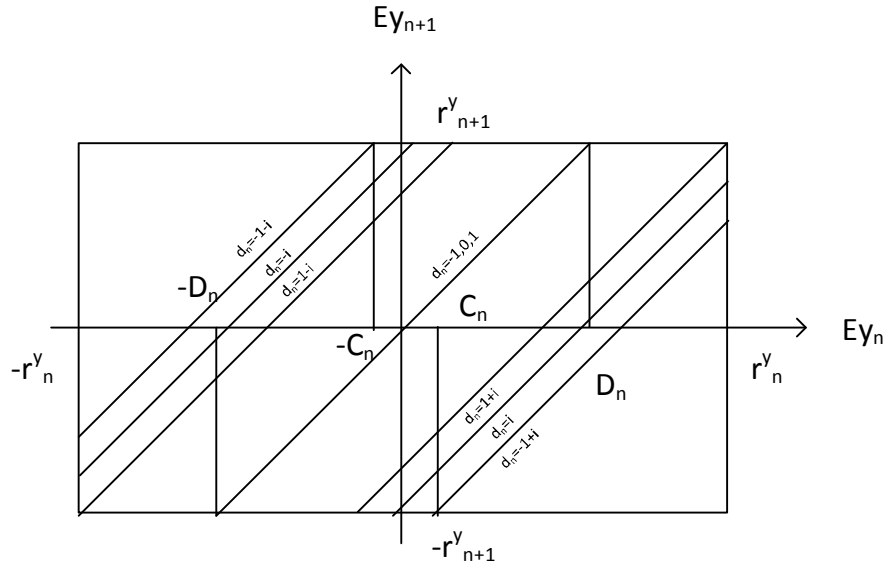
r_n^x musí splňovat podmínku: $r_{n+1}^x = r_n^x - \ln(1+2^{-n})$ a r_{n+1}^x musí být největší hodnota Ex_{n+1} odpovídající přímce d_n=1. Jelikož Ex musí konvergovat k nule, platí: $r_n^x = \sum_{k=n}^{\infty} \ln(1+2^{-k})$. Stejným způsobem se odvodí $s_n^x = -\frac{1}{2}\sum_{k=n}^{\infty} \ln(1-2^{-k+1}+2^{-2k+1})$. Konstanty v diagramu jsou proto rovny:

$$\begin{aligned} \bar{A}_n &= r_{n+1}^x + \ln(1-2^{-n}) \\ \bar{B}_n &= -s_{n+1}^x + \frac{1}{2}\ln(1+2^{-2n}) \\ A_n &= -s_{n+1}^x + \frac{1}{2}\ln(1+2^{-n+1}+2^{-2n+1}) \\ B_n &= r_{n+1}^x \end{aligned} \tag{19}$$

z toho jsou odvozeny následující volby dx_n:

$$\begin{aligned} &\text{pokud } Ex_n < -\bar{B}_n \text{ tak } dx_n\text{=-1} \\ &\text{pokud } -\bar{B} \le Ex_n < \bar{A}_n \text{ tak } dx_n\text{=-1 nebo 0} \\ &\text{pokud } \bar{A}_n \le Ex_n < A_n \text{ tak } dx_n\text{=0} \\ &\text{pokud } A_n \le Ex_n \le B_n \text{ tak } dx_n\text{=1 nebo 0} \\ &\text{pokud } B_n < Ex_n \text{ tak } dx_n\text{=1} \end{aligned} \tag{20}$$

Použijeme-li nyní vztah $Ey_{n+1}=Ey_n-dy_n\text{arctg}\left(\frac{2^{-n}}{1+dx_n.2^{-n}}\right)$, můžeme obdobným způsobem odvodit Robertsonův diagram pro volbu dy_n (obrázek 8).



Obrázek 8: Robertsonův diagram pro volbu dy_n

Volba dy_n musí být opět nezávislá na dx_n. Z toho odvodíme: $r_n^y = \sum_{k=n}^{\infty} arctg\left(\frac{2^{-k}}{1+2^{-k}}\right)$.

Pro konstanty v diagramu platí: $C_n = -r_{n+1}^y + arctg\left(\frac{2^{-n}}{1-2^{-n}}\right)$ a $D_n = r_{n+1}^y$. Z toho jsou odvozeny následující volby dy_n:

$$\begin{gathered}
\text{pokud } Ey_n < -D_n \text{ tak } dy_n = -1 \\
\text{pokud } -D_n \le Ey_n < -C_n \text{ tak } dy_n = -1 \text{ nebo } 0 \\
\text{pokud } -C_n \le Ey_n < C_n \text{ tak } dy_n = 0 \\
\text{pokud } C_n \le Ey_n \le D_n \text{ tak } dy_n = 1 \text{ nebo } 0 \\
\text{pokud } D_n < Ey_n \text{ tak } dy_n = 1
\end{gathered} \tag{21}$$

Algoritmus konverguje pro hodnoty:

$$\begin{gathered}
\text{-0,8298023738...} = -s_1^x \le Ex_1 \le r_1^x = \text{0,8688766517...} \\
\text{-0,749780302...} = -r_1^y \le Ey_1 \le r_1^y = \text{0,749780302...}
\end{gathered} \tag{22}$$

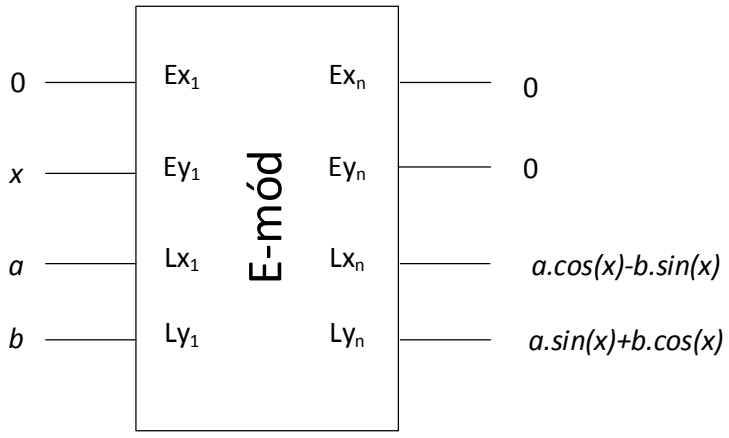
Je zjevné, že pro parametry $\bar{A}$=-1/2, A=1/4, C=3/4, $p_1 = 3$ a $p_2 = 4$ platí pro všechna *n*:

$$\begin{aligned} 2^n \bar{B}_n \le \bar{A} - 2^{-p_1} < \bar{A} \le 2^n . \bar{A}_n \\ 2^n A_n \le A < A + 2^{-p_1} \le 2^n . B_n \\ 2^n C_n \le C < C + 2^{-p_2} \le 2^n . D_n \end{aligned} \tag{23}$$

Proto pokud nadefinujeme $\bar{E}_n$ jako $2^n.E_n$ tak platí:

$$\begin{aligned} &\text{pokud } \bar{Ex}_n \le -\frac{5}{8} \text{ tak } dx_n=-1 \\ &\text{pokud } -\frac{5}{8} < \bar{Ex}_n < \frac{3}{8} \text{ tak } dx_n=0 \\ &\text{pokud } \frac{3}{8} \le \bar{Ex}_n \text{ tak } dx_n=1 \\ &\text{pokud } \bar{Ey}_n \le -\frac{13}{16} \text{ tak } dy_n=-1 \\ &\text{pokud } -\frac{13}{16} < \bar{Ey}_n < \frac{13}{16} \text{ tak } dy_n=0 \\ &\text{pokud } \frac{13}{16} \le \bar{Ey}_n \text{ tak } dy_n=1 \end{aligned} \tag{24}$$

Pokud vstupní hodnota do E-módu je ryze komplexní číslo, je provedeno násobení komplexní exponenciální funkcí, což znamená rotaci v komplexní rovině (obrázek 9).



Obrázek 9: 2D rotace pomocí BKM algoritmu

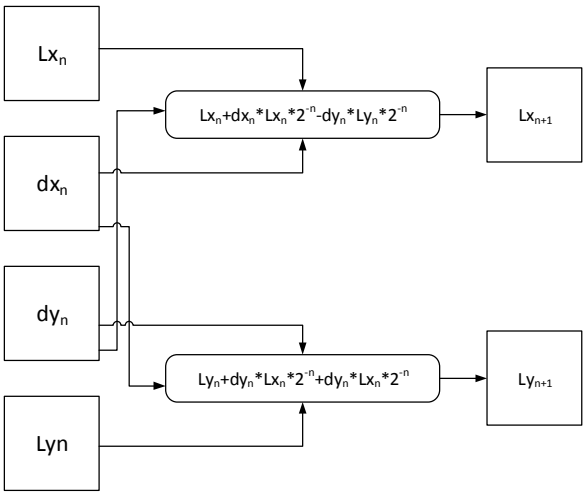
Algoritmus BKM konverguje stejně rychle jako algoritmus CORDIC a vyžaduje, aby v paměti byly uloženy hodnoty funkcí $\ln(1+dx_n.2^{-n+1}+(dx_n^2+ dy_n^2).2^{-2n})$ a $\operatorname{arctg}\left(\frac{2^{-n}}{1+dx_n.2^{-n}}\right)$ pro všechny možné kombinace *dx* a *dy*. Jelikož se jedná o velké množství hodnot, je v této podobě tento algoritmus nevhodný pro generování harmonického průběhu [5].

2.6 Upravený algoritmus BKM

Pokud bude rotace prováděna o konstantní úhel, je vstupní hodnota Ey_1 konstantní. Z toho plyne, že i posloupnost dn je vždy stejná. Tudíž můžeme celý algoritmus zjednodušit na iterace:

$$\begin{aligned} Lx_{n+1} &= Lx_n + dx_n.Lx_n.2^{-n} - dy_n.Ly_n.2^{-n} \\ Ly_{n+1} &= Ly_n + dy_n.Lx_n.2^{-n} + dx_n.Ly_n.2^{-n} \end{aligned} \tag{25}$$

kde dn je posloupnost předem vypočítaných dvoubitových čísel uložených v paměti. Díky tomu se sníží výpočetní náročnost, protože není potřeba kontrolovat podmínky a razantně se sníží paměťová náročnost. S touto úpravou se tento algoritmus již stává zajímavý pro účely této práce. Na obrázku 10 je znázorněno blokové schéma jedné iterace tohoto algoritmu. Realizace tohoto algoritmu jsou dostupné v příloze 5 a 10.



Obrázek 10: Iterace algoritmu BKM

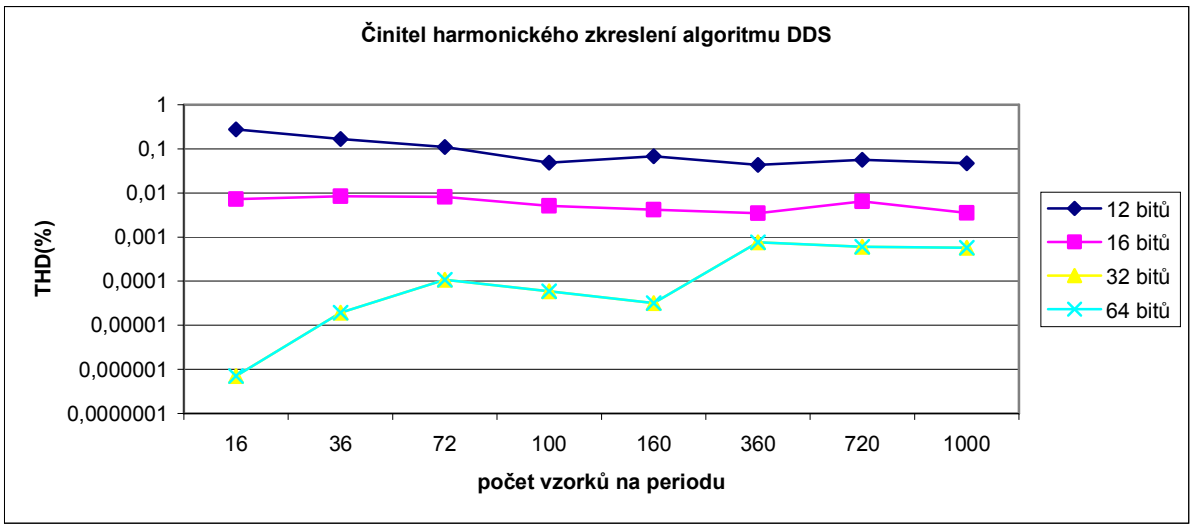
2.7 Přesnosti generovaných funkcí jednotlivých algoritmů

Přesnost generovaných funkcí je vyjádřena pomocí činitele harmonického zkreslení v procentech. Počet vzorků na periodu pro měření byl zvolen, aby byl pokryt relativně široký rozsah a zároveň mohly být tyto hodnoty použity i u algoritmu DDS (ten vyžaduje celý počet vzorků na čtvrt periodu). U upraveného algoritmu CORDIC z principu není možné použít tyto hodnoty. Proto bylo nutné u tohoto algoritmu zvolit takový počet který odpovídá bitovému

posunu. Všechna měření byla také provedena pro bitové šířky šedesát čtyři, třicet dva a šestnáct bitů formátu IEEE 754. Dále byla experimentálně zjištěna nejmenší bitová šířka, pro kterou všechny algoritmy ještě fungovaly alespoň pro jednu hodnotu počtu vzorků na periodu. Bylo zjištěno, že je to dvanáct bitů (pět bitů exponent, šest bitů mantisa).

2.7.1 Algoritmus DDS

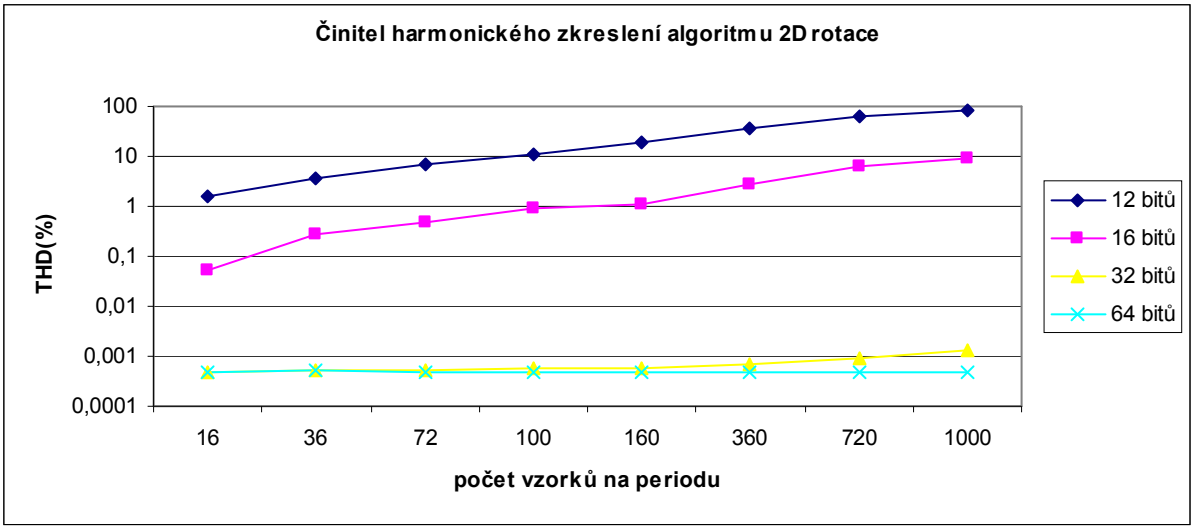
U tohoto algoritmu není možné, aby docházelo k významnému činiteli harmonického zkreslení, protože se jedná pouze o postupné načítání předen vypočítaných hodnot. K chybě dochází při zaokrouhlování vypočítaných hodnot na posledním bitu. Na grafu 1 je vidět, že činitel harmonického zkreslení závisí převážně na bitové šířce vzorků. Vzorky jsou vypočítány s bitovou přesností šedesát čtyři bitů ale při zápisu do proměnné typu sc_float je redukována na požadovanou bitovou šířku, čímž se ztrácí přesnost.



Obrázek 11: Závislost činitele harmonického zkreslení na počtu vzorků na periodu algoritmu DDS

2.7.2 2D rotace

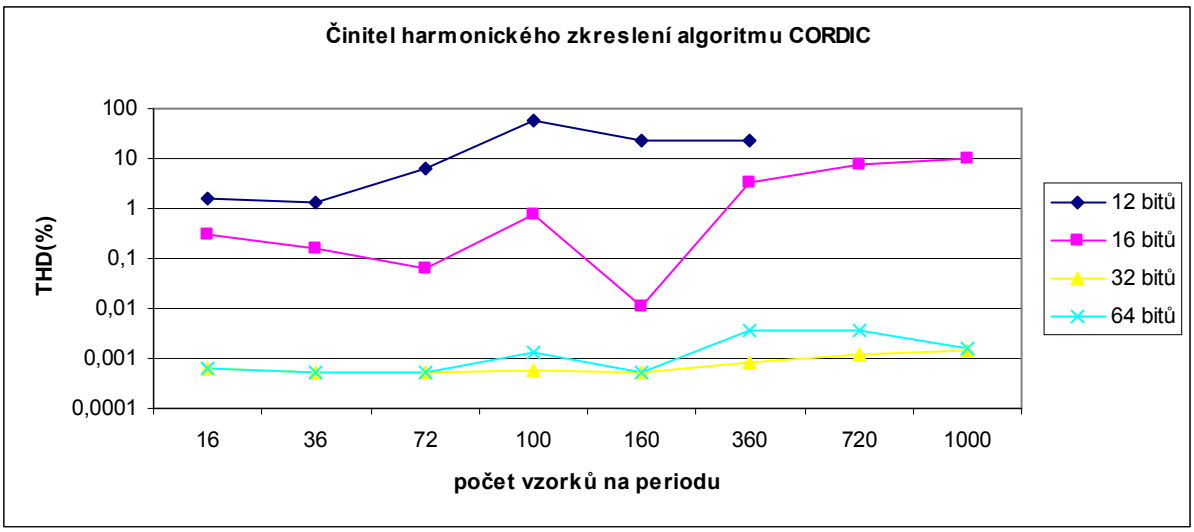
Tento algoritmus využívá k výpočtu vzorku hodnotu předcházejícího vzorku. Jelikož při každé operaci s plovoucí desetinnou čárkou nevyhnutelně dochází k nepřesnosti dané bitovým rozsahem čísla, tak dochází k postupné akumulaci této chyby. Aby byl generátor spolehlivý i pro velmi vysoký počet vygenerovaných period, bylo přidat čítač, pomocí něhož se hodnoty signálu resetují do počátečních hodnot po každé periodě. Na grafu 2 je vidět, že přesnost generovaných funkcí výrazně klesá s rostoucím počtem vzorků na periodu pro malé bitové šířky. Nutno podotknout, že pro hodnoty činitele harmonického zkreslení více než zhruba 10% už se jednalo o selhání algoritmu a vygenerovaný průběh se dá označit na nepoužitelný. U větších bitových šířek je pokles přesnosti s rostoucím počtem vzorků na periodu mnohem mírnější, protože je omezena velikost akumulované chyby.



Obrázek 12: Závislost činitele harmonického zkreslení na počtu vzorků na periodu algoritmu 2D rotace

2.7.3 Algoritmus CORDIC

Jedná o univerzální algoritmus, který pouze vypočítává funkční hodnoty funkce sinus vstupní hodnoty, které se postupně zvětšují. Z naměřených hodnot je vidět, že při bitové šířce dvanáct bitu algoritmus selhal už při sto vzorcích na periodu. Je to způsobeno tím, že čítač úhlů nebyl schopen správně načítat úhly při této bitové šířce.

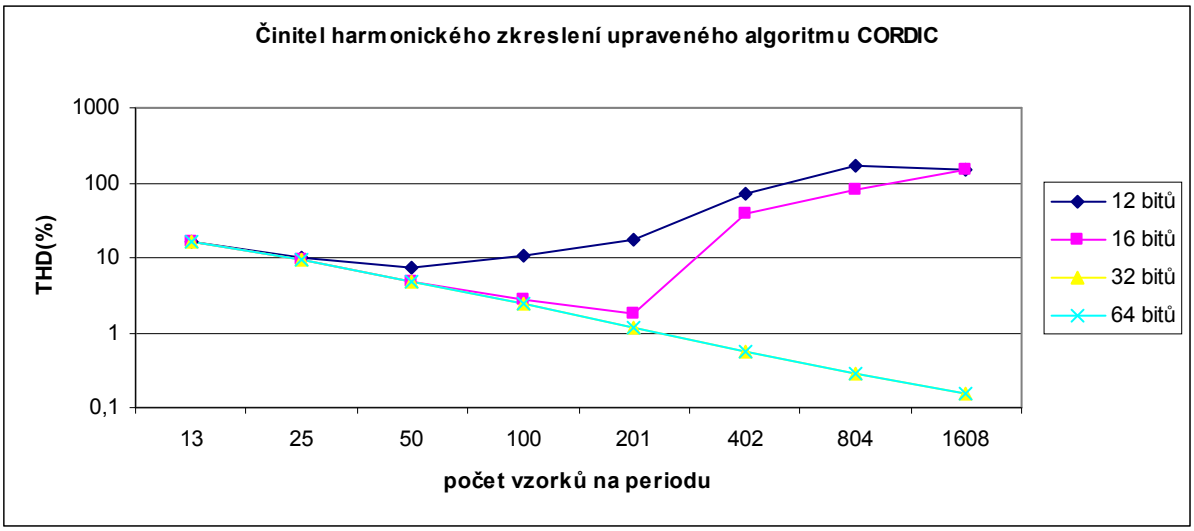


Obrázek 13: Závislost činitele harmonického zkreslení na počtu vzorků na periodu algoritmu CORDIC

Nepřesnost u bitové šířky šestnáct bitů je způsobena tím, že při jemném vzorkování se dva za sebou jdoucí vzorky mohou opakovat kvůli příliš nízkému rozlišení čísla. U vyšších bitových šířek je nepřesnost převážně způsobena nepřesností v měření činitele harmonického zkreslení.

2.7.4 Upravený algoritmus CORDIC

Tento algoritmus opět využívá k výpočtu vzorku hodnotu předcházejícího vzorku a tudíž také dochází k postupné akumulaci chyb. Bohužel u tohoto algoritmu není možné nastavit úhel rotace tak, aby vycházel celý počet vzorků na periodu a proto vzniká nepřesnost resetování po jedné periodě.

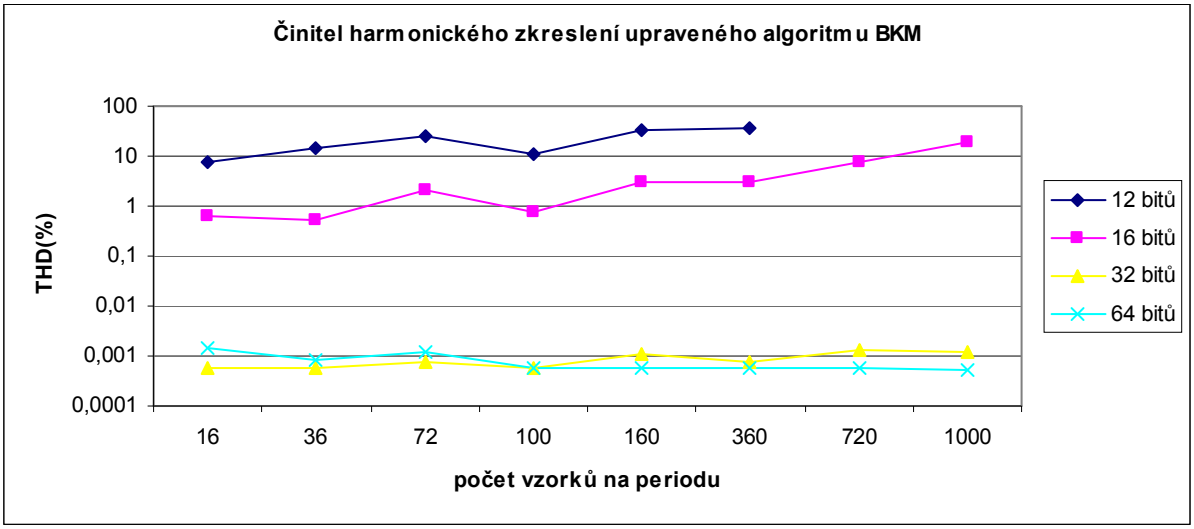


Obrázek 14: Závislost činitele harmonického zkreslení na počtu vzorků na periodu upraveného algoritmu CORDIC

Hlavní faktor ovlivňující činitel harmonického zkreslení je ovšem aproximace $\cos(\alpha)=1$ pro úhel rotace α. Tato aproximace je tím přesnější, čím více je vzorků na periodu. To vysvětluje průběh na grafu 4. U malých bitových šířek v kombinaci velkým počtem vzorků na periodu dochází k podtékání exponentu čísla v okamžiku, kdy se hodnota x nebo y na jednotkové kružnici blíží nule. Z toho důvodu v těchto případech tento algoritmus zcela selhává.

2.7.5 *Upravený algoritmus BKM*

Tento algoritmus opět využívá k výpočtu vzorku hodnotu předcházejícího vzorku a tudíž také dochází k postupné akumulaci chyb. Jak je vidět na grafu 5, tak tento faktor se u tohoto algoritmu příliš neprojevuje. Je to tím, že dochází k částečné kompenzaci střídavým zaokrouhlováním na posledním desetinném místě. Přesnost tohoto algoritmu velmi rychle stoupá s bitovou šířkou. Všechny případy, kde je činitel harmonického zkreslení vyšší než 10% lze opět považovat za selhání algoritmu.



Obrázek 15: Závislost činitele harmonického zkreslení na počtu vzorků na periodu algoritmu BKM

3 Behaviorální syntéza

Hlavním smyslem je zvýšit úroveň abstrakce tak, aby bylo možné danou aplikaci popsat ve vyšším programovacím jazyku a behaviorální syntéza (High-Level Sythesis, HLS) se následně postará o interpretaci algoritmického popisu a vytvoří RTL kód, který odpovídá tomuto popisu. Díky tomu je možné přímo využít obvodu FPGA pro danou aplikaci namísto procesoru, což přináší výrazné zrychlení [6].

3.1 provedení programu na procesoru

všechny procesory provádí program jako sekvenci instrukcí. Tyto instrukce jsou generovány kompilátorem, který převede algoritmus napsaný v C/C++ do asembleru. Ze zápisu ve formě:

$$x = a + b; \tag{26}$$

do formy:

$$\text{ADD } \$R1, \$R2, \$R3 \tag{27}$$

Kód (27) představuje operaci sčítání z vnitřních registrů procesoru. Tento kód sečte hodnoty v registrech R1 a R2 . Před provedením této instrukce je potřeba nahrát požadované hodnoty do těchto registrů. Tato operace trvá několik desítek hodinových cyklů v případě, že jsou data uložena v interní paměti procesoru. V případě, že jsou uložena v některé externí paměti, tak to může trvat až desítky tisíc hodinových cyklů. Interní cache paměť procesoru je ovšem velmi malá, a proto programátoři musí věnovat hodně času optimalizaci svého algoritmu.

3.2 Provedení programu na obvodu FPGA

Do obvodu FPGA je možné implementovat kteroukoliv funkci, která může běžet na procesoru. Hlavní rozdíl je v tom, že HLS kompilátor, který se používá k překladu kódu do RTL, nemá tak velká omezení paměti. HLS kompilátor umí zkompilovat kód takovým způsobem, že dané operaci jsou exkluzivně přiřazeny LUT. Na rozdíl od procesoru, kde jsou všechny výpočty prováděny na stejné ALU, na obvodu FPGA jsou přiřazeny nezávislé LUT každému výpočtu.

Další rozdíl spočívá v architektuře paměti a obtížnosti přístupu k paměti. V případě FPGA implementace HLS kompilátor rozmístí paměť co nejblíže příslušným operacím. Díky tomu je umožněn téměř okamžitý přístup k paměti a zároveň je mnohonásobně zvýšena paměťová propustnost, což je největší omezení výkonu procesorů.

3.2.1 Graf datového toku

Jedná se o další návrhovou techniku zaměřenou na paralelizaci programu. Tato metoda slouží k paralelnímu provedení celých funkcí v rámci jednoho programu. HLS tohoto dosahuje pomocí porovnávání vstupů a výstupů různých funkcí v programu. V nejjednodušším případě, když funkce pracují s jinými daty a nekomunikují se sebou, HLS alokuje prostředky pro každou funkci a vytvořené bloky běží nezávisle na sobě. U softwarových programů typicky nastává druhý případ, kdy jedna funkce poskytuje výsledky pro druhou (consumer-producer). V tomto případě HLS podporuje dva modely.

V prvním případě producer dokončí veškerá data před tím, než consumer může začít operace. Paralelizace je dosáhnuta vytvořením paměťových částí ping a pong. Každé funkci je přístupná pouze jedna část během volání funkce. Při začátku volání funkce si consumer i producer prohodí paměťové části. Tento přístup zaručuje správné fungování, ale omezuje úroveň paralelizace.

V druhém případě consumer může začít pracovat s částečnými výsledky a úroveň paralelizace je rozšířena o operace během volání funkce. Moduly obou funkcí jsou propojeny pomocí FIFO pamětí, které se chovají jako programová fronta a poskytují datovou synchronizaci mezi moduly. Během celého volání funkce na obou modulech běží operace s výjimkou začátku výpočtu, kdy consumer čeká na první data.[7]

3.2.2 Výběr zdrojů

Jedná se o výběr vhodných komponent pro realizaci potřebných a operátorů z technologické knihovny. Při výběru zdrojů se bere v úvahu zpoždění a plocha komponenty a také zda daná komponenta není již k dispozici.

3.2.3 Scheduling

V tradičním návrhu pro FPGA se jedná o manuální proces paralelizace algoritmu pro hardwarovou implementaci. HLS analyzuje jednotlivé operace a na základě toho dává jednotlivé operace do skupin, které proběhnou během jednoho hodinového taktu a umožňuje nastavit hardware tak, aby bylo možné překrývání volání funkcí. To odstraňuje nevýhodu procesoru, který potřebuje, aby byla daná funkce plně ukončena dříve, než může být volána další funkce se stejnými operacemi.

4 Implementace do obvodů FPGA a ASIC

Všechny algoritmy se porovnávaly z hlediska plochy, maximální frekvence a doby výpočtu jednoho vzorku. Měření se prováděla pro stejné bitové šířky a počty vzorků jako v kapitole 2.7 s výjimkou nejnižších hodnot, jelikož v této kapitole není cílem zkoušet stabilitu algoritmů. Byla použita behaviorální syntéza CtoS 12.20, syntetizátor Cadence RTL compiler 14.10. Pro obvod ASIC byla použita technologie AMIS I3T25 0.35µm. Upravené algoritmy CORDIC a BKM jsou v tabulkách pro zjednodušení nazvány pouze CORDICV2 a BKM

4.1 Plocha na čipu

Z naměřených hodnot je vidět, že pro malý počet vzorků na periodu je algoritmus DDS nejefektivnější. Také jde vidět, že všechny ostatní algoritmy mají prakticky konstantní plochu pro všechny počty vzorků na periodu. Je to dáno tím, že je u nich vždy stejná obvodová struktura. Z tabulky 1 je možné vyčíst, pro který počet vzorků na periodu jsou menší než algoritmus DDS. Plocha narůstá s rostoucí bitovou šířkou přibližně stejně u všech algoritmů. V tabulce 1 jsou uvedeny hodnoty pro obvod ASIC. Pro obvod FPGA budou hodnoty velmi podobné, ovšem s tím rozdílem, že algoritmus DDS bude zabírat větší plochu. Je to tím, že v obvodu ASIC jsou konstanty řešené tím způsobem, že se jednotlivé bity napojí na napájení a zem tak aby poskládaly požadovanou hodnotu. V obvodu FPGA dojde k uložení hodnot do paměti, což zabere víc plochy.

Tab. 1: Plocha na čipu jednotlivých algoritmů (μm^2)

DDS	počet vzorků	36	72	100	160	360	720	1000
bitová šířka								
16		418	584	730	1033	1954	3645	4907
32		691	1011	1298	1891	3691	7065	9583
64		1238	1871	2438	3613	7190	13913	18963
SINUS	počet vzorků	36	72	100	160	360	720	1000
bitová šířka								
16		3155	3155	3155	3155	3155	3155	3155
32		8317	8317	8317	8317	8317	8317	8317
64		23270	23270	23270	23270	23270	23270	23270
CORDIC	počet vzorků	36	72	100	160	360	720	1000
bitová šířka								
16		3639	3639	3639	3639	3639	3639	3639
32		8935	8937	8935	8937	8937	8937	8935
64		19566	21547	21550	21547	21546	21561	21552

CORDICV2	počet vzorků	25	50	100	201	402	804	1608
bitová šířka								
16		1983	1994	1983	1957	1994	1977	1994
32		4625	4619	4625	4584	4619	4624	4619
64		10138	10129	10138	10079	10129	10132	10130
BKM	počet vzorků	36	72	100	160	360	720	1000
bitová šířka								
16		2510	2509	2510	2510	2511	2508	2509
32		5636	5641	5649	5650	5647	5648	5646
64		12170	12160	12158	12172	12170	12172	12170

4.2 Maximální frekvence

Nejvyšší maximální frekvence dosahuje algoritmus DDS, protože se v podstatě jedná pouze o čítač adres. Všechny ostatní algoritmy mají přibližně desetkrát nižší maximální frekvenci a jsou řádově srovnatelné. Maximální dosažitelná frekvence klesá s rostoucí bitovou šířkou u všech algoritmů přibližně stejně s výjimkou algoritmu DDS pro nízký počet vzorků na periodu, kde od bitové šířky třicet dva bitů je pokles malý. V tabulce 2 jsou opět uvedeny pouze hodnoty pro obvod ASIC. U obvodu FPGA bude opět algoritmus DDS horší z důvodu odlišného řešení konstant, protože načítání z paměti trvá déle než pouhé přepínání vstupních hodnot.

Tab. 2: Maximální frekvence jednotlivých algoritmů (MHz)

DDS	počet vzorků	36	72	100	160	360	720	1000
bitová šířka								
16		290,10734	277,9322	268,0247	380,6624	264,2706	242,3655	255,037
32		188,00526	160,5136	170,097	148,9869	136,11	117,9523	120,8167
64		183,75597	172,1763	161,0565	140,5877	134,1922	112,8541	98,8533
SINUS	počet vzorků	36	72	100	160	360	720	1000
bitová šířka								
16		30,07157	30,07157	30,07157	30,07157	30,07157	30,07157	30,07157
32		16,106431	16,10643	16,10643	16,10643	16,10643	16,10643	16,10643
64		9,1224229	9,122423	9,122423	9,122423	9,122423	9,122423	9,122423
CORDIC	počet vzorků	36	72	100	160	360	720	1000
bitová šířka								
16		24,829299	24,76228	24,78868	24,78868	24,76228	24,8293	24,78868
32		13,763299	13,75894	13,7633	13,75894	13,75894	13,75894	13,7633
64		8,5323504	8,5753	8,605408	8,5753	8,510349	8,558347	8,539418
CORDICV2	počet	25	50	100	201	402	804	1608

	vzorků							
bitová šířka								
16		44,450371	45,14673	44,45037	47,03448	45,14673	45,47108	45,14673
32		26,504811	27,08632	26,50481	27,83499	27,08632	27,01316	27,08632
64		16,54205	16,32253	16,54205	16,74369	16,32253	16,54205	16,32253
BKM	počet vzorků	36	72	100	160	360	720	1000
bitová šířka								
16		41,440471	41,44047	41,3736	41,40958	41,44391	41,39416	41,15226
32		25,334414	25,02941	24,84287	24,82683	24,78868	25,30556	24,89358
64		15,683568	15,68357	15,80078	15,68357	15,6976	15,6848	15,68357

4.3 Minimální doba výpočtu jednoho vzorku

Minimální doba výpočtu jednoho vzorku je odvozena z maximální frekvence a počtu hodinových taktů potřebných pro výpočet. Pro algoritmus DDS a upravený algoritmus CORDIC stačí jeden hodinový takt na výpočet. Algoritmus 2D rotace potřebuje čtyři. Algoritmy CORDIC a BKM se řídí vzorcem 30

$$\text{Počet hodinových taktů} = \text{bitová šířka mantisy} * A \tag{30}$$

kde A=4 pro CORDIC a A=3 pro BKM. Je to kvůli rychlosti konvergence a počtu potřebných taktů na jednu iteraci. Nejkratší doby výpočtu dosahuje algoritmus DDS. Hodnoty v tabulce 3 jsou opět naměřeny pro obvod ASIC. Rozdíly pro FPGA budou stejné jako u maximální frekvence jelikož jsou tyto hodnoty na sobě přímo závislé.

Tab. 3: Doba výpočtu jednoho vzorku u jednotlivých algoritmů (ns)

DDS	počet vzorků	36	72	100	160	360	720	1000
bitová šířka								
16		3,45	3,60	3,73	2,63	3,78	4,13	3,92
32		5,32	6,23	5,88	6,71	7,35	8,48	8,28
64		5,44	5,81	6,21	7,11	7,45	8,86	10,12
SINUS	počet vzorků	36,00	72,00	100,00	160,00	360,00	720,00	1000,00
bitová šířka								
16		133,02	133,02	133,02	133,02	133,02	133,02	133,02
32		248,35	248,35	248,35	248,35	248,35	248,35	248,35
64		438,48	438,48	438,48	438,48	438,48	438,48	438,48
CORDIC	počet vzorků	36,00	72,00	100,00	160,00	360,00	720,00	1000,00
bitová šířka								
16		1611,00	1615,36	1613,64	1613,64	1615,36	1611,00	1613,64

32		6684,44	6686,56	6684,44	6686,56	6686,56	6686,56	6684,44
64		24377,81	24255,71	24170,85	24255,71	24440,83	24303,76	24357,63
CORDICV2	počet vzorků	25,00	50,00	100,00	201,00	402,00	804,00	1608,00
bitová šířka								
16		22,50	22,15	22,50	21,26	22,15	21,99	22,15
32		37,73	36,92	37,73	35,93	36,92	37,02	36,92
64		60,45	61,27	60,45	59,72	61,27	60,45	61,27
BKM	počet vzorků	36,00	72,00	100,00	160,00	360,00	720,00	1000,00
bitová šířka								
16		723,93	723,93	725,10	724,47	723,87	724,74	729,00
32		2723,57	2756,76	2777,46	2779,25	2783,53	2726,67	2771,80
64		9946,72	9946,72	9872,93	9946,72	9937,82	9945,94	9946,72

5 Závěr

V této práci byly uvedeny základní vlastnosti reprezentace čísel s plovoucí desetinnou čárkou a přiblížen princip behaviorální syntézy z vyššího programovacího jazyka. Hlavní část práce se zabývala algoritmy pro generování harmonického průběhu.

Algoritmus DDS (direct digital synthesis) je nejméně náročný na výpočet (stačí jediný čítač). Paměťová náročnost je závislá na požadovaném počtu vzorků na periodu. Proto bude tento algoritmus nejefektivnější v případě, že není potřeba velkého množství vzorků. V opačném případě bude paměťová náročnost značná a bude vhodnější použít jiný algoritmus.

2-D rotace má téměř nulovou paměťovou náročnost, ale vyžaduje čtyřikrát násobení čísel s plovoucí desetinnou čárkou a dvakrát sčítání. Tento algoritmus bude vhodný v případě, že kompilátor bude schopný využít násobičky a sčítačky i jinde v návrhu a zároveň je potřeba určitý přesně daný krok mezi vzorky.

Algoritmus CORDIC je výpočetně náročný a navíc vyžaduje, aby v paměti bylo uloženo tolik konstant, kolik je bitová šířka mantisy generovaného čísla. Výhodou je, že není potřeba násobení. Tento algoritmus bude vhodné využít v případě, že bude potřeba počítat jednu konkrétní hodnotu funkce sinus nebo cosinus a kompilátor bude schopný využít stejné prostředky.

Algoritmus CORDIC pro generování harmonických průběhů je podobný jako 2-D rotace ale s tím, že úhel rotace je striktně omezen na hodnoty $tg\theta = \pm 2^{-i}$. Tento algoritmus se ukázal jako velmi efektivní, a to především pro velký počet vzorků na periodu, kdy zabírá nejmenší plochu na čipu ze všech testovaných algoritmů a má krátkou dobu výpočtu jednoho vzorku. Činitel harmonického zkreslení je v tomto případě taktéž velmi nízký. Tento algoritmus bude vhodné použít v případě, že si v návrhu vystačíme s výše definovanými úhly.

Algoritmus BKM je kvůli své vysoké paměťové náročnosti nevhodný pro generování harmonického signálu, avšak díky své podstatě je narozdíl od algoritmu CORDIC možné při výpočtu následujícího vzorku využít vzorku předešlého a není potřeba žádné multiplikační konstanty. Jedná se o násobení komplexní exponenciální funkcí realizované za pomocí sčítání a bitových posunů. Díky tomu je možné tento algoritmus implementovat tak, aby jednotlivé operace byly vždy stejné. Takto implementovaný algoritmus BKM již nevyžaduje téměř žádnou paměť a zároveň se snižuje výpočetní náročnost. Takto upravený algoritmus bude vhodné použít v případě, že ve zbytku návrhu není potřeba počítat konkrétní hodnotu funkce sinus nebo cosinus a zároveň není k dispozici násobička.

Všechny algoritmy se podařilo realizovat úspěšně realizovat pomocí behaviorální syntézy.

Z hlediska přesnosti generovaných funkcí nejlépe dopadl algoritmus DDS, který z principu ani nemůže být příliš nepřesný. Algoritmy 2D rotace, CORDIC a BKM mají činitel harmonického zkreslení pod 0,01% při použití bitové šířky třicet dva a více bitů. Přesnost upraveného algoritmu CORDIC stoupá s rostoucím počtem vzorků na periodu. Při dvě stě jedna vzorcích na periodu a bitové šířce třicet dva bitů byl činitel harmonického zkreslení 1%.

6 Seznam literatury

[1] Sites.google.com [online]. 2001 [cit. 2014-12-17]. Dostupné z: https://sites.google.com/site/dtcsinformation/data-representation/number-systems/floating-point-numbers

[2] GOLDBERG, David. What Every Computer Scientist Should Know About Floating-Point Arithmetic [online]. 1991 [cit. 2015-06-03]. Dostupné z: http://www.validlab.com/goldberg/paper.pdf

[3] A Technical Tutorial on Digital Signal Synthesis. Http://www.analog.com/ [online]. 1999 [cit. 2014-12-17]. Dostupné z: http://www.analog.com/static/imported-files/tutorials/450968421DDS_Tutorial_rev12-2-99.pdf

[4] ANDRAKA, Ray. A survey of CORDIC algorithms for FPGA based computers [online]. 1998 [cit. 2015-06-03]. Dostupné z: http://www.andraka.com/files/crdcsrvy.pdf

[5] BAJARD, Jean-Claude, Sylvanus KLA a Jean-Michel MULLER. BKM: A New Hardware Algorithm for Complex Elementary Functions [online]. 1994 [cit. 2015-06-03]. Dostupné z: http://perso.ens-lyon.fr/jean-michel.muller/BKM94.pdf

[6] A look inside behavioral synthesis. Http://www.eetimes.com/ [online]. 2004 [cit. 2014-12-17]. Dostupné z: http://www.eetimes.com/document.asp?doc_id=1217645

[7] Introduction to FPGA Design with Vivado High-Level Synthesis. Http://www.xilinx.com/ [online]. 2013 [cit. 2014-12-17]. Dostupné z: http://www.xilinx.com/support/documentation/sw_manuals/ug998-vivado-intro-fpga-design-hls.pdf