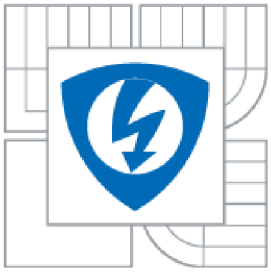
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV MIKROELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF MICROELECTRONICS

NÁVRH ČÍSLICOVÉHO FILTRU TYPU PÁSMOVÁ PROPUST

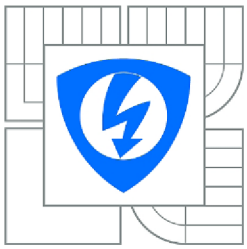
DESIGN OF BANDPASS DIGITAL FILTER

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — Vojtěch Dvořák
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. Lukáš Fujcik Ph.D.
SUPERVISOR

BRNO 2011



VYSOKÉ UČENÍ
TECHNICKÉ V BRNĚ

**Fakulta elektrotechniky
a komunikačních technologií**

Ústav mikroelektroniky

Bakalářská práce

bakalářský studijní obor
Mikroelektronika a technologie

Student: Vojtěch Dvořák ***ID:*** 118345
Ročník: 3 ***Akademický rok:*** 2010/2011

NÁZEV TÉMATU:

Návrh číslicového filtru typu pásmová propust.

POKYNY PRO VYPRACOVÁNÍ:

V prostředí Matlab navrhněte číslicový filtr typu pásmová propust. Vytvořte ideální model, který bude považován jako reference pro následnou verifikaci s číslicovým filtrem popsaným v jazyce VHDL. Prozkoumejte možnosti návrhu digitálního filtru pro obvody FPGA s využitím IP core generátoru v prostředí Xilinx ISE WebPack.

DOPORUČENÁ LITERATURA:

Podle pokynů vedoucího práce.

Termín zadání: 7. 2. 2011 ***Termín odevzdání:*** 2. 6. 2011

Vedoucí práce: doc. Ing. Lukáš Fujcik, Ph.D.

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

Abstrakt

Cílem práce je vysvětlit problematiku číslicových filtrů IIR, ukázat postup návrhu číslicového filtru v prostředí Matlab a navrhnout model ideálního filtru typu pásmová propust s konkrétními parametry v prostředí Matlab. Tento filtr bude následně sloužit jako referenční model pro verifikaci s filtrem popsaným v jazyce VHDL.

Abstract

The aim of this work is to explain the problems of digital IIR filters, demostrate process of designing digital filters in Matlab and design a model of ideal band-pass filter with concrete parameters in Matlab. This filter will then serve as a reference model for verification with the filter described in VHDL.

Klíčová slova

Digitální zpracování signálu, digitální filtr, IIR filtr, Matlab, FDA Tool, VHDL, programovatelné logické obvody

Keywords

Digital signal processing, digital filter, IIR filter, Matlab, FDA Tool, VHDL, programable logic circuits

Bibliografická citace

DVOŘÁK, V. *Návrh číslicového filtru typu pásmová propust.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2011. 41 s. Vedoucí bakalářské práce doc. Ing. Lukáš Fujcik, Ph.D.

Prohlášení

Prohlašuji, že tuto bakalářskou práci na téma **Návrh číslicového filtru typu pásmová propust** 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 tohoto projektu 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 1. června 2011 ….....

podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce doc. Ing. Lukáši Fujcikovi Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování práce.

V Brně dne 1. června 2011 ….....

podpis autora

Obsah

Seznam obrázků

Úvod

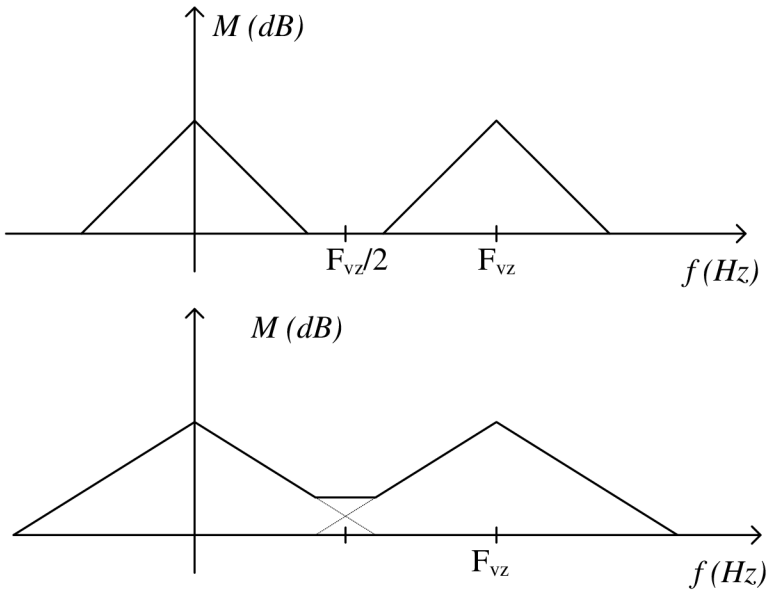
Digitální filtry jsou základem zpracování digitálního signálu podobně, jako jsou klasické analogové filtry určeny pro zpracování analogového signálu. Na rozdíl od analogových filtrů se v digitálních filtrech nevyužívá závislosti impedance na frekvenci, jaká je u pasivních součástek, cívek a kondenzátorů, ale jedná se o matematické operace násobení a sčítání řešených pomocí digitálních obvodů, mikroprocesorů nebo signálových procesorů.

Analogový signál se před zpracováním pomocí digitálního filtru musí vzorkovat, kvantovat a kódovat – tedy převést na digitální signál. Vzorkování signálu podléhá vzorkovacímu teorému – vzorkovací frekvence musí být minimálně dvakrát větší než nejvyšší frekvence obsažená ve spektru signálu, tedy

$$F_{max} = \frac{F_{vz}}{2}. \quad (1)$$

V případě nedodržení této podmínky dochází k aliasingu. Aliasing je jev, který je způsoben překrytím spekter signálu. Při vzorkování dochází k periodizaci spektra – spektrum původního analogového signálu se opakuje, přičemž středem každého opakování je vzorkovací frekvence F_{vz} a následně také její celočíselné násobky. Z Fourierovy transformace víme, že signál má kladnou i zápornou část spektra, přičemž záporná část je stejná, jako část kladná, avšak zrcadlově obrácená. Pokud je tedy nejvyšší frekvence ve spektru vyšší než $F_{vz}/2$, dojde k překrytí části dvou „pulsů“ ve spektru a k nenávratnému zkreslení signálu, jak je vidět na **Obrázku 1**. První spektrum odpovídá správně vzorkovanému signálu, nejvyšší kmitočet ve spektru je menší než $F_{vz}/2$, oproti tomu u spektra druhého signálu nebyl dodržen vzorkovací teorém a signál byl znehodnocen – nejvyšší frekvence byly zesíleny.

V praxi se aliasingu předchází použitím analogového filtru typu dolní propust nebo vzorkováním dostatečně vysokou vzorkovací frekvencí. Při použití analogového antialiasingového filtru dojde k odstranění části spektra, kterou by aliasing postihl. Signál je i tak znehodnocen, ale obvykle je takové zkreslení signálu menší než zkreslení způsobené aliasingem.



Obrázek 1: Aliasing

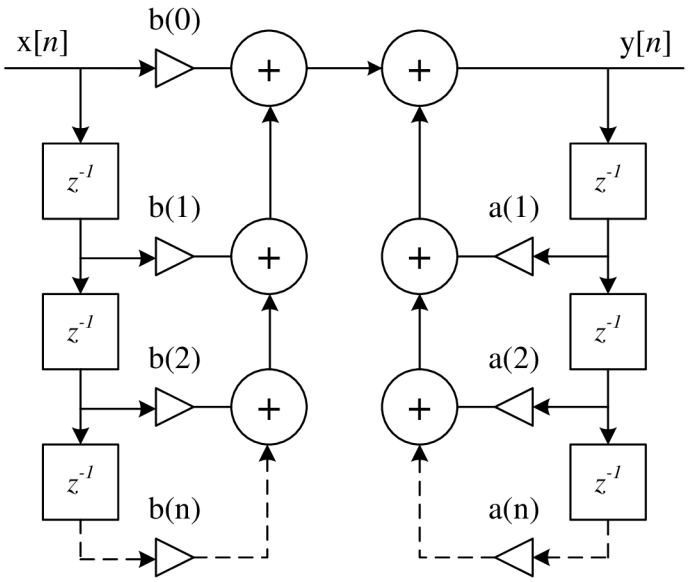
Digitální filtry lze z hlediska impulsní odezvy rozdělit na dva typy, filtr s konečnou impulzní odezvou (Finite Impulse Response – FIR) a filtr s nekonečnou impulzní odezvou (Infinite Impulse Response – IIR). Rozdíl mezi oběma filtry je, jak už název napovídá, v odezvě na jednotkový impuls. Jednotkový impuls je u digitálního signálu reprezentován jako posloupnost nul, pouze na pozici nula nabývá hodnoty jedna. Pokud takový impuls přivedeme na vstup FIR filtru, výstup se po nějaké době ustálí na hodnotu '0', pokud však jednotkový impuls bude vstupem pro IIR filtr, hodnota se nikdy neustálí na nulovou hodnotu, ačkoliv se k této hodnotě bude stále přibližovat.

Hlavní rozdíl mezi filtry FIR a IIR je v jejich konstrukci, zatímco FIR filtry jsou realizovány jako nerekurzivní struktura, u IIR filtrů se objevuje zpětná vazba. Každý ze jmenovaných filtrů má pak své výhody i nevýhody a z nich vyplívající využití. Filtr typu FIR je vždy stabilní, ale má vyšší řád, tedy množství zpožďovacích prvků ve struktuře; oproti tomu řád IIR filtru bývá zpravidla menší, ale je třeba dát pozor na stabilitu, aby nedošlo k nekontrolovatelnému rozkmitání. Filtr typu FIR pak nalezne uplatnění především při realizaci adaptivních filtrů, ve kterých se často mění koeficienty a není tak možné vždy kontrolovat stabilitu filtru, jako by bylo potřeba v případě IIR filtru. Oproti tomu, filtry IIR jsou díky nižšímu řádu rychlejší, a umožňují rychlejší zpracování signálu, proto se používají tam, kde je kritická doba zpoždění signálu.

1. IIR filtr

1.1. Přímá struktura IIR filtru

Filtr typu IIR vždy ve své struktuře obsahuje zpětnou vazbu – jeho výstup závisí nejen na vstupních hodnotách, současné i minulých, ale i na jeho předchozích výstupních hodnotách.



Obrázek 2: Přímá struktura IIR filtru

Na **Obrázku 2** je znázorněna přímá struktura filtru IIR pomocí grafu signálových toků. Ve schématu jsou použity různé bloky pro jednotlivé operace. Násobení konstantou znázorňuje trojúhelník s konstantou a nebo b, operaci sčítání znázorňuje plus v kroužku a zpožďovací blok je zakreslen jako čtverec s prvkem z^{-1}. Když na vstup přivedeme signál $x[0]$, jeho hodnota bude vynásobena konstantou b(0), následně bude k výsledku přičtena předchozí hodnota vstupu násobena konstantou b(1) (v případě první hodnoty signálu bude samozřejmě hodnota předchozího vzorku nulová) a přičtena hodnota minulého výstupu (taktéž bude ještě nulová). Výstupem bude první hodnota výstupního signálu $y[0]$. Při druhém vstupu $x[1]$ bude tato hodnota opět násobena konstantou b(0), bude k ní přičtena hodnota minulého vstupu násobena konstantou b(1) a hodnota minulého výstupu násobena konstantou $a(1)$. Výsledkem je výstupní hodnota $y[1]$. Tento popis lze přehledně zapsat pomocí diferenčních rovnic:

$$y[0] = b(0) * x[0]$$

$$y[1] = b(0) * x[1] + b(1) * x[0] + a(1) * y[0]$$

$$y[2] = b(0) * x[2] + b(1) * x[1] + a(1) * y[1] + b(2) * x[0] + a(2) * y[0]$$

$$\vdots$$

$$y[n] = b(0) * x[n] + \sum_{i=1}^{\infty} b(i) * x[n-i] + a(i) * y[n-i] \qquad (2)$$

Tímto způsobem by se soustava rovnic mohla rozšiřovat až do velikosti, která by odpovídala řádu filtru – v případě IIR filtru řád nebývá zpravidla zvlášť vysoký, proto je množství operací násobení a sčítání v přiměřeném počtu.

Právě zpětná vazba, na **Obrázku 2** zobrazena na pravé straně, způsobuje nekonečnou impulsní odezvu. Pokud vstupní signál skončí, na vstup budou přicházet už jen samé nuly, na výstupu se bude stále objevovat nenulová hodnota – a s touto hodnotou bude IIR filtr díky zpětné vazbě stále počítat. Zpětná vazba však způsobuje velkou citlivost filtru na kvantování koeficientů – při implementaci do hardwaru jsou koeficienty vždy zaokrouhleny a stejně jsou zaokrouhleny i výsledky násobení vzorků těmito koeficienty. Výstupní hodnota je mírně zaokrouhlena – to by nebyla závažná chyba, kdyby se s ní znovu nepočítalo, neustálé násobení zaokrouhlené výstupní hodnoty a zaokrouhleného koeficientu povede k postupnému znehodnocení údaje výstupní hodnoty. Jak se lze této degradaci výsledku alespoň částečně vyhnout bude popsáno později u kapitoly zabývající se kvantováním koeficientů.

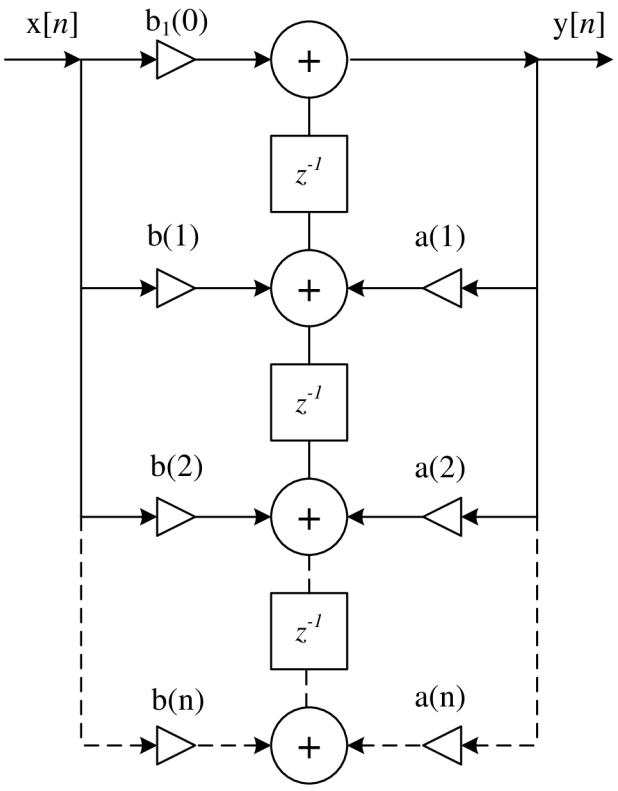
1.2. Transponovaná struktura IIR filtru

Schéma IIR filtru na **Obrázku 2** není jediné možné zakreslení filtru, ve skutečnosti obsahuje dvakrát více zpožďovacích bloků než je potřeba. To je nevýhodné při realizaci na úrovni hardwaru, více zpožďovacích bloků znamená větší požadavky na paměť. Proto se častěji používá popis pomocí stavových veličin. Stavovou veličinu $v[n]$ definujeme jako součet vstupní hodnoty a výstupní hodnoty, každá násobena příslušným koeficientem. S takto definovanou stavovou veličinou lze přepsat diferenční rovnici následujícím způsobem,

$$v[n] = b(i) * x[n] + a(i) * y[n]$$

$$y[n] = b(0) * x[n] + \sum_{i=1}^{\infty} v[n-i]. \qquad (3)$$

Výhodou takovéto úpravy je existence právě tolika zpožďovacích bloků, kolikátý je řád filtru, a tím dosažena právě ona požadovaná úspora paměti. Touto úpravou se samozřejmě změní i graf signálových toků. Vstupní hodnota je postupně násobena jednotlivými koeficienty a každý součin je ukládán do paměti, podobně se pracuje i s výstupní hodnotou a následně jsou jednotlivé sečteny a je k nim dále připočten minulý stav filtru – proto pojem stavová veličina. Taková struktura filtru je pak nazývána transponovaná struktura IIR filtru, ukázka je na **Obrázku 3**.



Obrázek 3: Přímá forma IIR filtru – transponová struktura

1.3. Přenosová funkce IIR filtru

Jak už bylo zmíněno, digitální filtr typu IIR se využívá pro zpracování signálu tam, kde je kritická rychlost zpracování nebo by obdobný filtr typu FIR dosahoval vysokého řádu a zabíral by velkou část paměti, jak by tomu bylo v případě zde navrhovaného filtru. IIR filtr je popsán jeho přenosovou funkcí,

$$H(z) = \frac{\sum_{i=0}^{M} a_i z^i}{\sum_{i=0}^{N} b_i z^i}. \tag{4}$$

Přenosovou funkci IIR filtru lze chápat jako podíl dvou polynomů, přičemž kořeny polynomu v čitateli jsou nazývány nulové body a kořeny polynomu ve jmenovateli jsou nazývány póly přenosové funkce. Pokud jsou nulové body přenosové funkce rovny nule, je i přenosová funkce rovna nule. Frekvence odpovídající nulovým bodům jsou tedy potlačovány. Oproti tomu, pokud by póly funkce byly rovny nule, přenosová funkce by rostla nade všechny meze, to prakticky znamená, že frekvence blížící se pólům jsou přenášeny bez potlačení.

Z přenosové funkce lze také odvodit některé podmínky nutné pro realizaci filtru. První podmínkou je kauzalita filtru, ta je splněna, když je mocnina ve jmenovateli větší než v čitateli, tedy $N \geq M$. Pokud by nebyla tato podmínka dodržena, filtr by byl nekauzální – výstupní hodnoty by byly závislé na vstupních hodnotách, které by do filtru měly vstupovat

později, jinými slovy, filtr by předbíhal čas. Něco takového samozřejmě nelze realizovat, proto každý reálný filtr je vždy kauzální a v jeho popisu pomocí přenosové funkce má čitatel vždy nižší řád než jmenovatel.

Druhou podmínkou je stabilita filtru. Aby byl filtr stabilní, musí všechny póly přenosové funkce daného filtru ležet uvnitř jednotkové kružnice v komplexní rovině z. Oproti první podmínce, která je splněna u každého reálného filtru, stabilitu filtru je třeba u každého návrhu kontrolovat. Pokud by totiž nebyla tato podmínka splněna, filtr by mohl začít nekontrolovatelně oscilovat.

2. Návrh IIR filtru typu pásmová propust v Matlabu

Navrhovaný filtr je určen k potlačení stejnosměrné složky a odstranění možného vlivu aliasingu potlačením frekvencí vyšších než $F_{vz}/2$, podle toho jsou voleny i parametry filtru. Propustné pásmo je mezi frekvencemi *10 Hz* a *5 kHz*, přičemž potlačení frekvencí nižších než *0,01 Hz* a vyšších než *7,5 kHz* musí být alespoň *40 dB*, v propustném pásmu je povoleno zvlnění *0,1 dB*. Použitá vzorkovací frekvence je *15,625 kHz*.

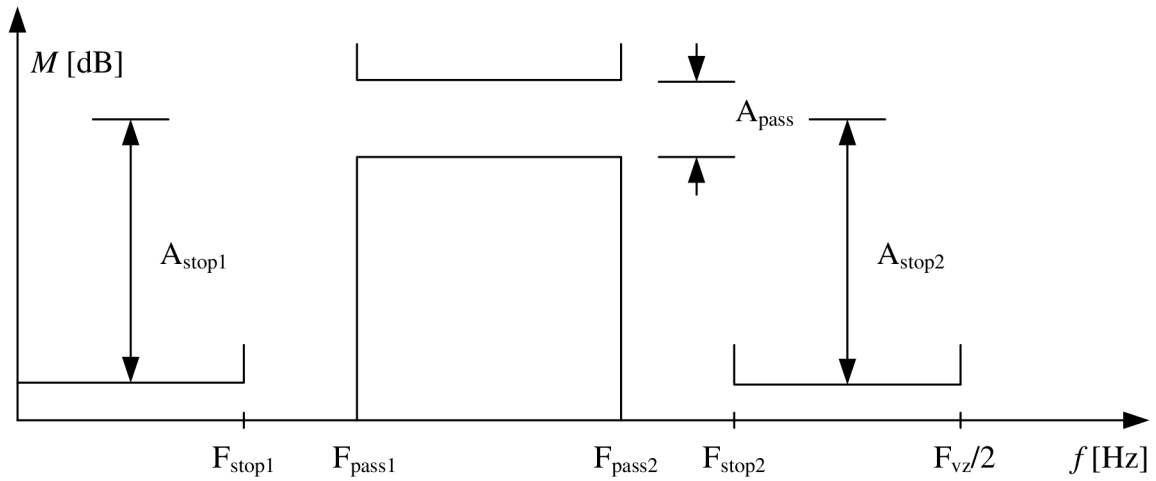
2.1. Návrh filtru v nástroji FDA Tool

K návrhu digitálního filtru nejdříve použijeme toolbox implementovaný přímo v Matlabu. Jedná se o toolbox Filter Design & Analysis Tool určený pro počítačový návrh a analýzu číslicových filtrů. Kromě samotného návrhu podle zvolených kritérií tak umožňuje i analýzu zde navrženého filtru. K tomu slouží mnoho funkcí, z těch základních jsou to například odezva na jednotkový impuls, odezva na jednotkový skok nebo zobrazení nulových bodů a pólů v komplexní rovině *z* pro kontrolu stability filtru.

Při návrhu filtru v nástroji FDA Tool je třeba nejdříve zvolit typ filtru podle přenášené frekvence, v tomto případě se jedná o filtr typu pásmová propust. Další volbou je výběr filtru podle impulsní odezvy. Vzhledem k zadaným kritériím filtru byl zvolen filtr IIR, protože přenosová funkce obdobného filtru FIR by byla velmi vysokého řádu. Jako analogový prototyp byla zvolena Butterworthova aproximace ideální pásmové propusti o zadaných mezních kmitočtech.

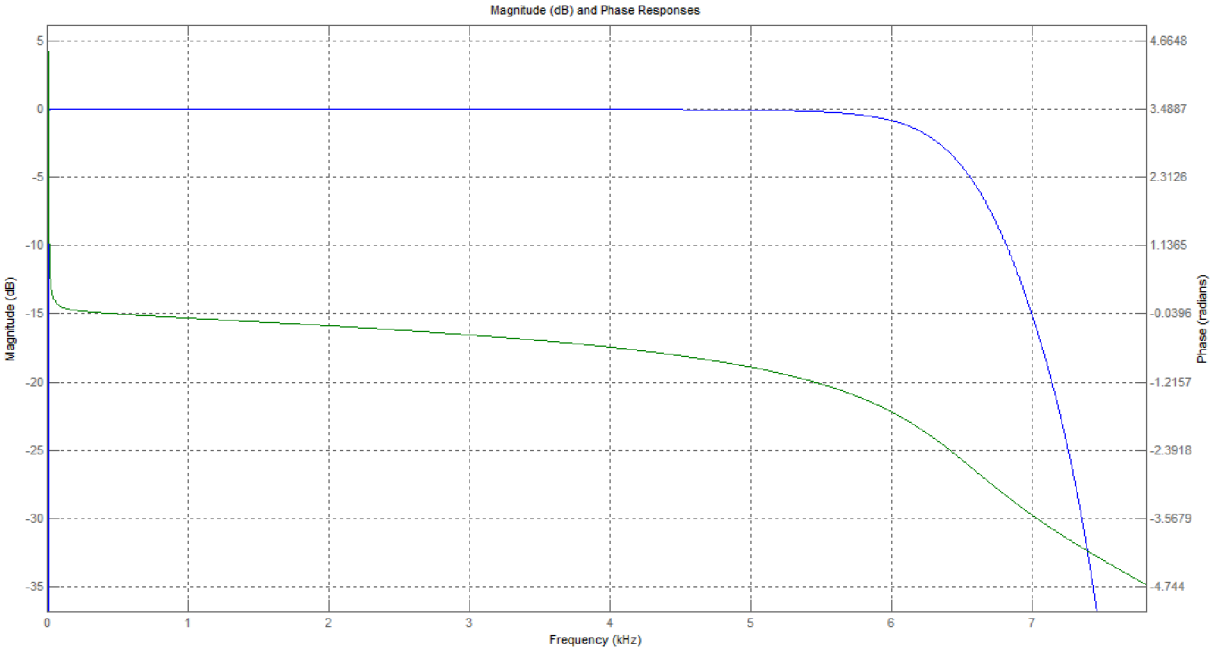
2.1.1. Přenosová charakteristika IIR filtru

Na **Obrázku 4** je toleranční schéma IIR filtru použité v nástroji FDA Tool. Na frekvenční ose je pět význačných frekvencí, F_{stop1} a F_{stop2} jsou frekvence, při kterých dochází k potlačení části spektra signálu o zadanou hodnotu, F_{pass1} a F_{pass2} jsou hodnoty, které vytyčují přenášenou část spektra, poslední vyznačenou frekvencí je $F_{vz}/2$ – polovina vzorkovacího kmitočtu, za kterým musí být potlačení tak výrazné, aby nedošlo ke zkreslování signálu vlivem aliasingu. Na ose modulu spektra jsou vyznačeny tři intervaly, hodnoty A_{stop1} a A_{stop2} znamenají míru potlačení spektra signálu mezi frekvencemi F_{stop1} a F_{pass1} , resp. F_{pass2} a F_{stop2}. Hodnota A_{pass} reprezentuje povolené zvlnění charakteristiky v propustném pásmu.



Obrázek 4: Toleranční schéma IIR filtru

Zadaným parametrům odpovídá filtr, jehož přenosová funkce je vykreslena na **Obrázku 5**. Modrá křivka je amplituda přenosové funkce, zelená křivka zobrazuje její fázi. Na horizontální ose je zobrazena frekvence, na vertikálních osách pak hodnota amplitudy v decibelech a fáze v radiánech.



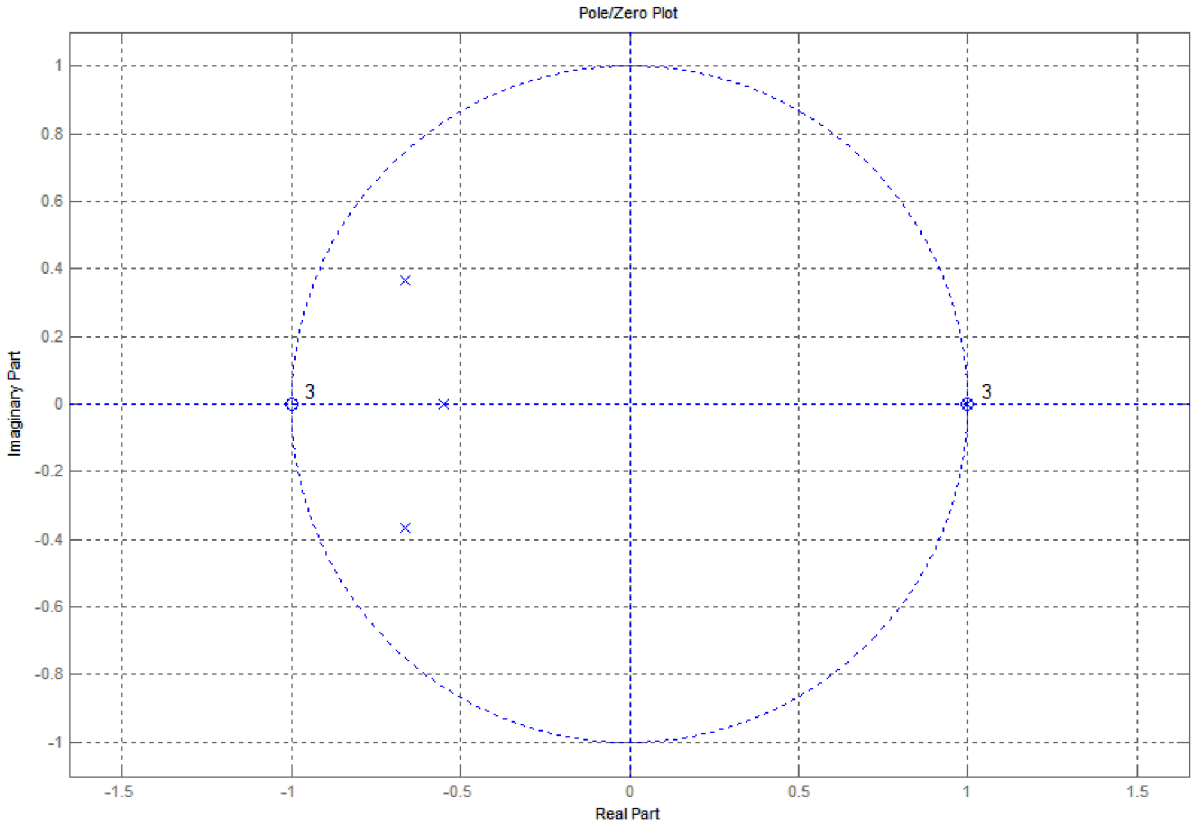
Obrázek 5: Přenosová charakteristika filtru

Z přenosové funkce lze vyčíst, že navržený filtr odpovídá požadavkům, dochází k potlačení stejnosměrné složky signálu a frekvencí nad $F_{vz}/2$. Frekvence mezi 10Hz a 5 kHz jsou přenášeny bez znatelného útlumu a odpovídá požadovanému malému zvlnění přenosové charakteristiky. K potlačení signálu o 3 dB dochází při frekvenci 5,5Hz v případě dolní

hranice propustného pásma a 6,4 kHz v případě horní hranice propustného pásma a tím definováno propustné pásmo filtru.

2.1.2. Nulové body a póly přenosové funkce filtru

Jak už bylo zmíněno v **kapitole 1.3**, při návrhu IIR filtru je třeba kontrolovat stabilitu pomocí zakreslení pólů přenosové funkce do roviny *z*. Přestože FDA Tool kontroluje stabilitu filtru automaticky, je dostupná i funkce, která umožní provést kontrolu ručně. Jak je vidět na **Obrázku 6**, všechny póly, značeny křížkem, jsou uvnitř jednotkové kružnice. Zároveň platí, že každý reálný filtr musí všechno póly i nulové body na reálné ose nebo musí být podél reálné osy komplexně sdruženy. Tři póly přenosové funkce jsou vykresleny v levé části kružnice, jeden z nich leží přímo na reálné ose, dva další jsou komplexně sdruženy. Tři zbývající póly jsou umístěni v pravé straně kružnice, blízko hodnoty jedna. Podobně jako tři póly na levé straně kružnice, i z těchto tří zbývajících jsou dva komplexně sdruženy a jeden leží na reálné ose. Nulové body přenosové pak leží v hodnotě -1 a +1, v každé této hodnotě jsou nulové body trojité.

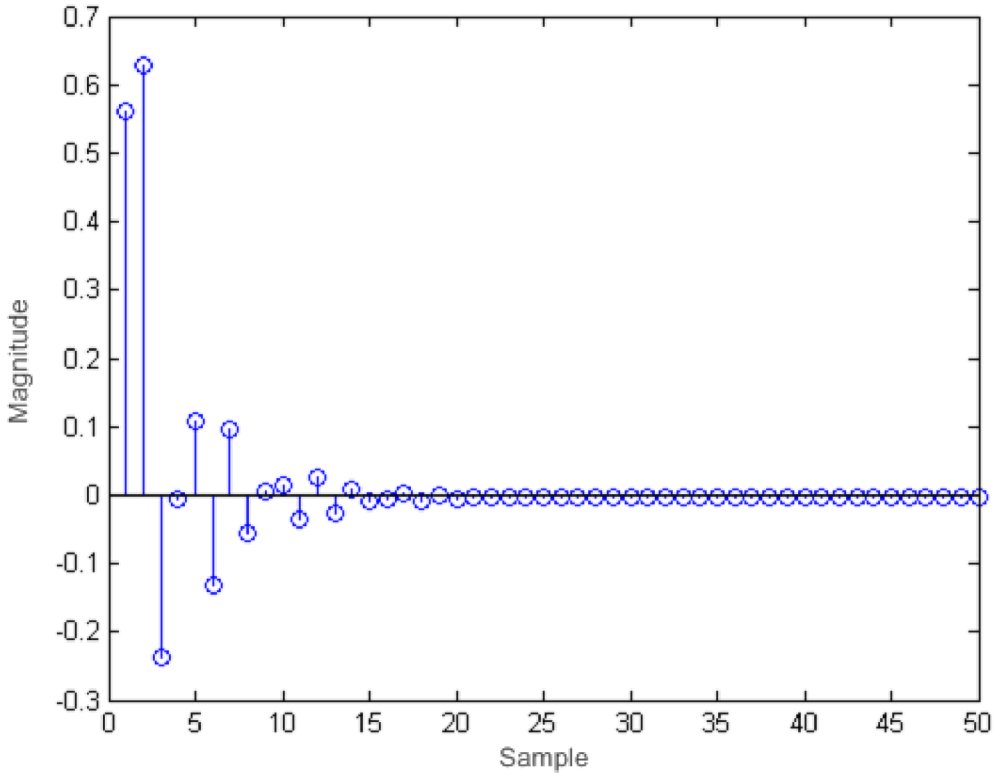


Obrázek 6: Nulové body a póly v rovině z

2.1.3. Impulsní odezva a odezva na jednotkový skok

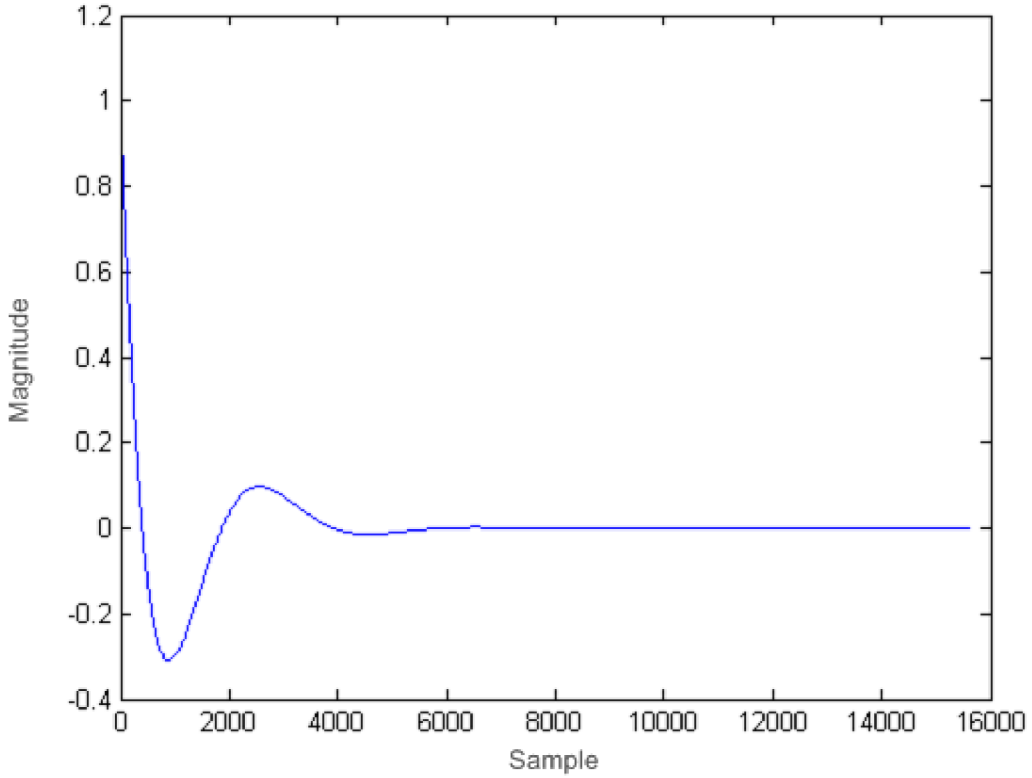
Mezi základní analýzy filtru patří také odezva na jednotkový impuls a odezva na jednotkový skok. Odezva na jednotkový impuls vykresluje výstup filtru, když je vstupním

signálem pouze jeden vzorek, $x[0] = 1$. Impulsní odezva pro zde navrhovaný filtr je na **Obrázku 7**. Impulsní odezva filtru umožňuje odvodit odezvu filtru na jakýkoli libovolný signál. Diskrétní signál lze chápat jako posloupnost jednotkových impulsů násobených nějakou konstantou a posunutých na příslušnou pozici. Pokud je systém lineární, což u tohoto filtrů platí, odezvu na libovolný signál lze získat jako součet jednotlivých odezev na jednotkový impuls.



Obrázek 7: Odezva na jednotkový impuls

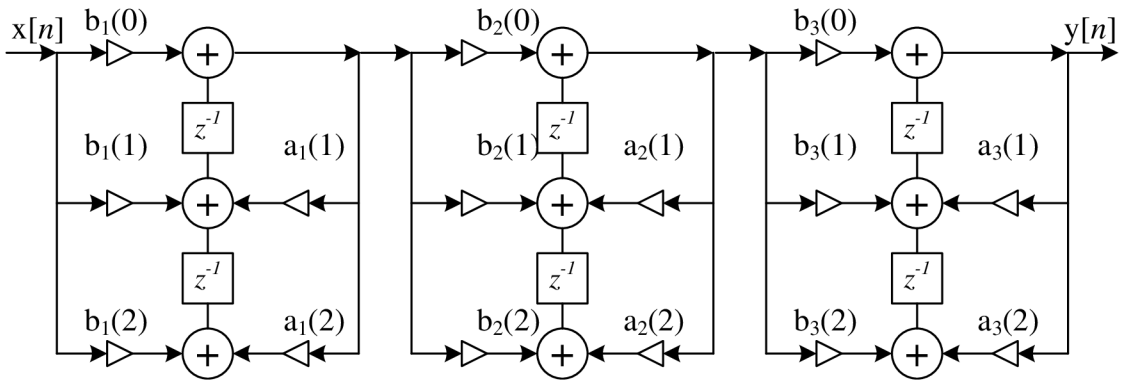
Odezva na jednotkový skok je vykreslena na **Obrázku 8**. Jednotkový skok je změna vstupních hodnot ze samých nul na samé jedničky. Taková změna je ve spektrální oblasti rovna vysokým frekvencím, naopak po odeznění vlivu přechodu z nuly na jedničku je signál v podstatě jen stejnosměrná složka, kterou chceme potlačit. Z grafu odezvy na jednotkový skok je snadné vyčíst, že vstupní vysoká frekvence způsobí překmit přeneseného signálu do záporných hodnot – dojde k otočení fáze. Po odeznění těchto vlivů dojde k přiblížení výstupního signálu k hodnotě nula, tedy k potlačení stejnosměrné složky. Vzhledem k nekonečné impulsní odezvě tohoto filtru nebude hodnota výstupu nikdy ustálena na nulu, přestože se jí bude blížit.



Obrázek 8: Odezva na jednotkový skok

2.1.4. Kvantování koeficientů

Po návrhu a simulaci filtru je hlavním požadavkem samozřejmě jeho hardwarová realizace. Pro tuto realizaci jsou nutné koeficienty filtru, na **Obrázku 2** jsou to hodnoty $a(1)$ až $a(3)$ a $b(0)$ až $b(3)$ kterými jsou jednotlivé vzorky signálu násobeny. FDA Tool vypočte koeficienty pro dva základní typy struktur IIR filtru – Single Section a Second-Order Section. Význam těchto dvou typů struktur je v požadované přesnosti koeficientů při kvantování. V příloze je uveden výpis koeficientů pro obě struktury.



Obrázek 9: Struktura Second-Order Section IIR filtru

Matlab standardně počítá v plovoucí řádové čárce s přesností na 64 bitů (double-precision floating-point), takovou přesnost počítání není výhodné používat při implementaci v hardwaru – bylo by třeba velká paměť a 64 bitová násobička a sčítačka by zabíraly velké místo v obvodu a způsobovaly významné zpoždění. Proto je mnohdy výhodnější snížit požadavky na přesnost výměnou za menší hardwarové požadavky. FDA Tool umožňuje kvantování koeficientů a převod do formátu pevné řádové čárky. Při kvantování je však třeba dát pozor na stabilitu filtru – pokud není číslo dostatečně přesné, můžou se některé póly zaokrouhlené přenosové funkce dostat mimo vnitřek jednotkové kružnice a způsobit nestabilitu filtru. Kvantováním se také zhorší strmost přenosové funkce na dolní mezi přenosu filtru. Při návrhu filtru jako Single Section bylo třeba kvantovat koeficienty s přesností na 25 bitů. Proto je výhodné použít strukturu filtru Second-Order Section, kdy je filtr i při přesnosti čísla na 18 bitů stabilní. Tato struktura je vytvořena kaskádním zapojením filtrů druhého řádu, její schéma je na **Obrázku 9**. Zmíněná výhoda snížení náročnosti na kvantování je však vykoupena větším zpožděním při výpočtu.

2.2. Návrh filtru v textovém režimu

K návrhu filtru v prostředí Matlabu není nutné použít FDA Tool, v knihovně Signal Processing Toolbox je mnoho funkcí, které lze použít přímo k zapsání zdrojového kódu pro návrh filtru. Výhodou zápisu v textovém formátu je přenositelnost mezi počítači a různými verzemi Matlabu. Celý zdrojový kód je uveden v **Příloze 1**.

V první části zdrojového kódu je specifikace parametrů filtru. Vektor Fp vyjadřuje propustné pásmo, hodnoty uvedené v závorkách jsou krajní hodnoty přenášeného spektra. Vektor Fs obsahuje hodnoty frekvencí, na kterých je definováno potlačení. Vzorkovací frekvence je uložena v proměnné Fvz. Proměnná Ap vyjadřuje povolené zvlnění v propustném pásmu, v proměnné As je uložena hodnota potlačení mezi frekvencemi ve vektoru Fp a vektoru Fs. Poslední dvě hodnoty jsou udávány v decibelech.

Samotný návrh filtru probíhá pomocí funkce *fdesign.bandpass* a funkce *design*. Funkce *fdesign.bandpass* vrací hodnoty odpovídající parametrům filtru ve formátu, s kterým může funkce *design* pracovat. Návratovou hodnotou funkce *design* jsou koeficienty navrženého filtru. V parametrech funkce můžeme specifikovat použitou aproximaci pro návrh filtru, v tomto případě byla použita Butterworthova aproximace stejně jako při návrhu v nástroji FDA Tool, druhým vstupním parametrem jsou parametry filtru vypočtené funkce *fdesign.bandpass*.

S vypočtenými koeficienty filtru už můžeme provést stejné simulace jako v nástroji FDA Tool. Funkce *freqz* vykreslí modul a fázi kmitočtové charakteristiky s použitím toolboxu Filter Visualization Tool, vstupními parametry jsou koeficienty filtru, Fvz je použita k nastavení frekvenční osy v grafu – maximální zobrazená frekvence na frekvenční ose je Fvz/2. Funkce *zplane* slouží k vykreslení nulových bodů a pólů na jednotkovou kružnici v rovině *z*, vstupním parametrem jsou koeficienty filtru. Pokud výstupní hodnoty funkce přiřadíme do proměnné, místo vykreslení nulových bodů a pólů do roviny *z* budou tyto hodnoty uloženy do použité proměnné.

Pro simulaci impulsní odezvy, odezvy na jednotkový skok nebo odezvy filtru na jakýkoliv libovolný signál je stěžejní funkce *filter*. Vstupními hodnotami funkce jsou parametry filtru a signál, který chceme filtrovat. K vykreslení odezvy je využita funkce *stem*, resp. *plot*. Funkce *plot* u odezvy na jednotkový skok byla zvolena proto, že odezva filtru na jednotkový skok trvala podstatně déle než odezva na jednotkový impuls a jednotlivé vzorky výstupní signálu by nebyly rozeznatelné.

3. Návrh filtru v jazyce VHDL

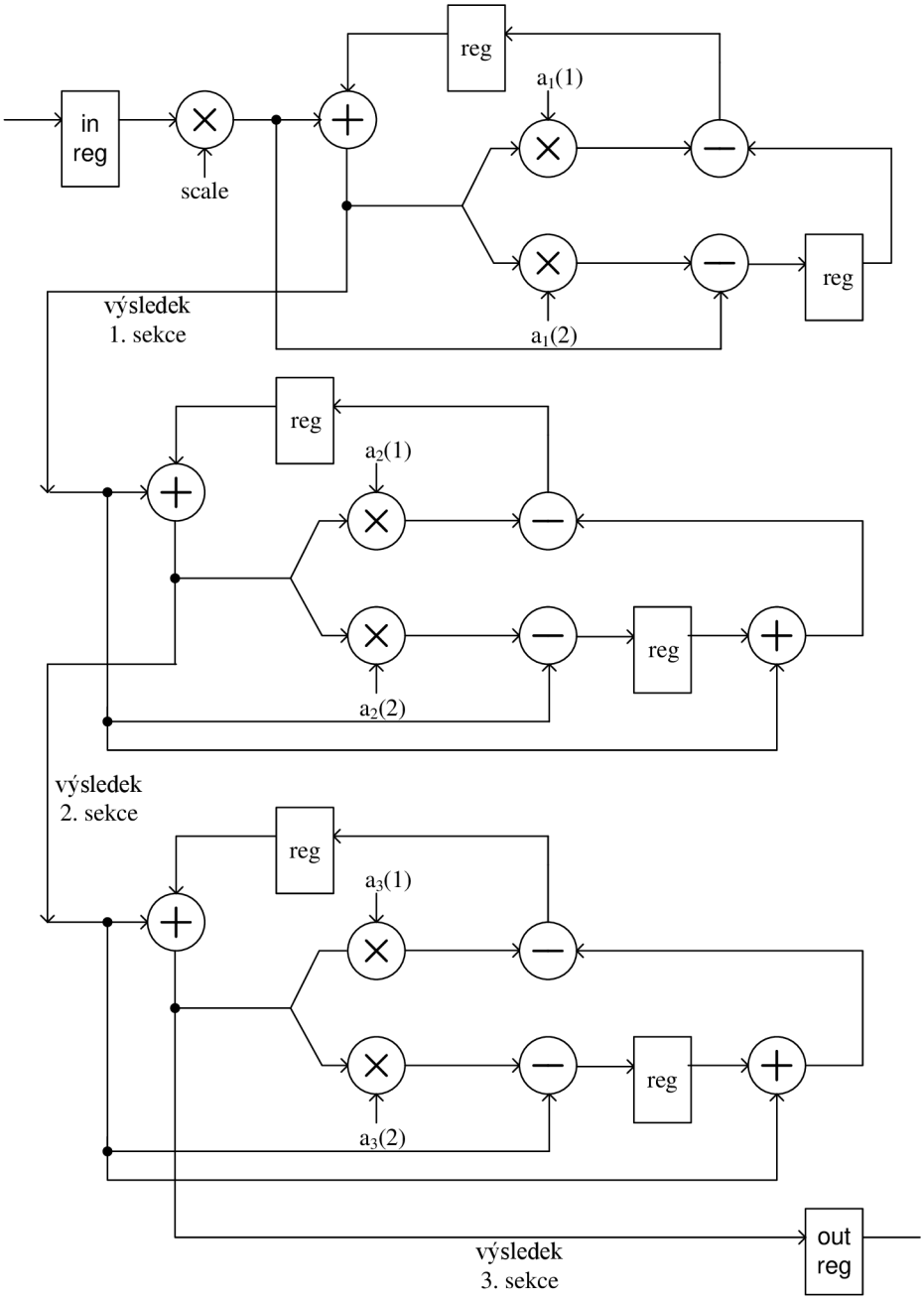
Digitální filtr lze na hardwarové úrovni rozdělit na dvě základní části, aritmetickou část a logickou část, jejíž součástí může být i paměťový blok.

Aritmetická část je složena z bloků provádějících matematické operace, sčítaček a násobiček. Počet těchto sčítaček a násobiček záleží na použité struktuře filtru v závislosti na požadavku rychlosti zpracování signálu a potřebné plochy na čipu, přičemž tyto dva požadavky jdou vždy proti sobě. Rychlé filtry obvykle obsahují větší aritmetickou část, která zabírá velkou plochu čipu, ale zpracovávají signál s velmi malým zpožděním. Pokud ovšem u zpracovávaného signálu není zpoždění kritické, nebo je jeho vzorkovací frekvence v porovnání s pracovním kmitočtem obvodu malá, je výhodné použít minimum obvodů provádějících matematické operace i za cenu nutnosti použít paměťové bloky a složitější řídící logiku.

Logická část rozhoduje o postupu prováděných operacích, odesílá signály do aritmetické části a výsledky pak zapisuje do paměti nebo s nimi dále pracuje. V závislosti na struktuře filtru pak může obsahovat různé další bloky. Při použití více sčítaček a násobiček bývá řídící logika obvykle jednodušší, naopak při použití minimálního množství těchto komponent musí rozhodovat, který signál a kdy je do těchto bloků přiřazen a je tedy výrazně složitější.

3.1. Referenční model

Jako referenční model pro navrhovaný filtr bude použit filtr navržený programem Matlab. V předchozí kapitole byly ukázány možnosti návrhu filtru v prostředí FDA Tool, včetně generování a kvantování koeficientů. Ke generování samotného VHDL popisu filtru slouží další nástroj implementovaný přímo do prostředí FDA tool, Filter Design HDL Coder™ 2. Tento nástroj umožňuje u navrženého filtru ještě několik specifikací před samotným generováním VHDL popisu. Lze specifikovat, kde budou uloženy koeficienty filtru a jak s nimi bude následně počítáno. V obou případech pak byla vybrána ta jednodušší možnost, koeficienty uložené přímo v obvodu a klasické násobení bez úprav koeficientů. Dalšími variantami by bylo použít pro koeficienty další vstup a nahrávat je z externí paměti nebo s nimi počítat v režimu CSD (canonical signed digit). Použití režimu CSD by sice urychlilo násobení, ale znamenalo by to složitější a tedy i rozsáhlejší obvod a v tomto případě by to bylo zbytečné. Kromě nastavení parametrů je možné nechat vygenerovat i testovací soubor, tato možnost ale nebyla využita.



Obrázek 10: Struktura filtru generovaného programem Matlab

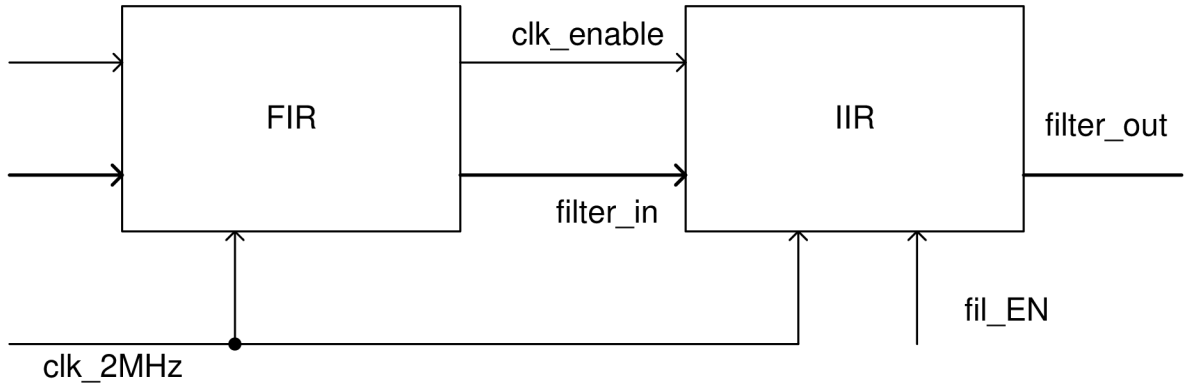
Struktura referenčního filtru je rozkreslena na **Obrázku 10**. K filtraci signálu je použita téměř výhradně kombinační logika, registry jsou použity pouze pro zpožďovací bloky a případné vstupní a výstupní registry. To znamená, že všechny operace probíhají v jednom hodinovém taktu. Nevýhodou filtrace během jednoho hodinového taktu je skutečnost, že pro každé sčítání nebo násobení je potřeba samostatná sčítačka či násobička, výsledkem je struktura obsahující sedm násobiček a jedenáct sčítaček zabírající velkou plochu na čipu.

Z **Obrázku 10** je také možné vyčíst tok signálu ve filtru. Jednotlivé operace jsou zařazeny za sebe, ale protože nejsou použity žádné registry, v kterých by signál čekal na hodinový takt, celý filtr si lze představit jako rozsáhlou kombinační síť, kterou je vstupní signál určitým způsobem transformován.

Takový popis filtru je neefektivní, každé násobení i sčítání je prováděno se stejnou přesností, a proto i všechny použité sčítačky a násobičky jsou stejné a při použití vhodného algoritmu by bylo možné pro všechny operace použít jednu sčítačku a jednu násobičku.

3.2. Struktura navrhovaného filtru

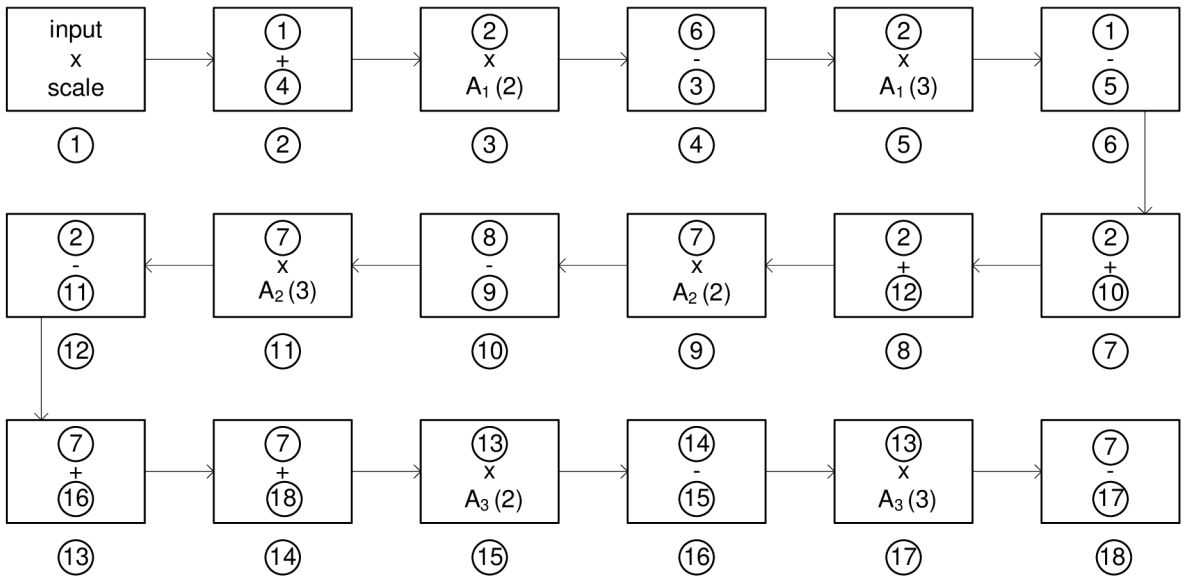
Navrhovaný filtr bude realizován na čipu společně s dalšími bloky pro zpracování signálu. Všechny bloky budou pracovat se společným hodinovým signálem o frekvenci 2MHz. Pro návrh tohoto IIR filtru je nejdůležitější předchozí blok, kterým je decimační filtr FIR, a který decimuje signál čtyřikrát ze vstupní frekvence vzorků 62,5 kHz na výstupní frekvenci 15,625kHz, s kterou tedy přicházejí vzorky do IIR filtru.



Obrázek 11: Zapojení filtru na čipu

Na **Obrázku 11** je zapojení IIR filtru za výše zmiňovaným decimačním FIR filtrem. Do IIR filtru vstupuje signál přenášející data *filter_in* s frekvencí 15,625 kHz, signál *clk_enable* funguje jako povolení načítání vstupu ve filtru IIR a je generován v předchozím bloku, ve FIR filtru. Tento signál přichází s frekvencí 15,625 kHz, stejně jako filtrovaná data, a jeho délka odpovídá jednomu taktu použitého hodinového signálu, tzn. puls trvá 0,5 μs a zbývajících 63,5 μs je jeho hodnota nulová. Do IIR flitru také vstupuje signál *fil_EN*, který umožňuje vypnutí filtru nastavením signálu do nuly, pokud by to bylo potřeba například při vedení signálu do jiného bloku umístěného mimo čip. Výstupem IIR filtru je signál *filter_out* nesoucí filtrovaný signál, jeho frekvence je stejná jako frekvence vstupního signálu, tedy 15,625 kHz.

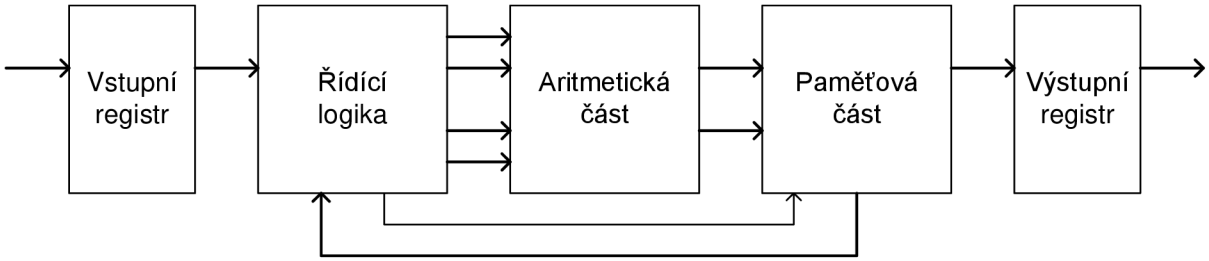
Při zadávání tohoto filtru byl jako jeden z hlavních požadavků použití minimálního množství aritmetických bloků, ideálně jedné násobičky a jedné sčítačky. Tomuto požadavku také odpovídá navržený algoritmus. Filtrace signálu probíhá ve 20 krocích, v prvním kroku dochází k načtení vstupního signálu, v dalších osmnácti krocích se počítají jednotlivé mezivýsledky a poslední krok slouží pouze k zápisu do registru poslední vypočtené hodnoty. Podrobně jsou jednotlivé kroky počítání mezivýsledků rozkresleny na **Obrázku 12**, jedná se o stejný postup výpočtů jako u referenčního filtru, ovšem s tím rozdílem, že každý krok probíhá v jednom hodinovém taktu a mezivýsledky jsou ukládány do posuvného registru. Tento postup umožňuje použít jedné sčítačky a jedné násobičky.



Obrázek 12: Algoritmus filtrace

Na **Obrázku 12** je výpočetní algoritmus zakreslen pomocí jednotlivých kroků, které se postupně provádějí, přičemž číslo kroku je použito jako identifikátor výsledku, tzn. je pomocí něj odkazováno na výsledek konkrétní operace. Zpožďovací bloky jsou pak snadno realizovány pomocí odkazů na kroky, které v současném cyklu ještě nebyly vypočteny, a na jejich místě v paměti je uložena hodnota vypočtená v minulém cyklu. Pro správnou činnost filtru je tedy nezbytné před každou spuštěním vymazat obsah paměti pomocí signálu *reset*.

Prvním početním krokem je násobení vstupního signálu normovacím koeficientem. Dalších pět kroků pak odpovídá výpočtům v prvním filtru kaskádní struktury. Oproti dalším dvěma sekcím, pracujících každá v šesti krocích, v první sekci chybí jeden krok – koeficient $B_1(2)$ je roven nule a proto je jedno sčítání zbytečné, k číslu by se vždy přičítaly jen nuly.



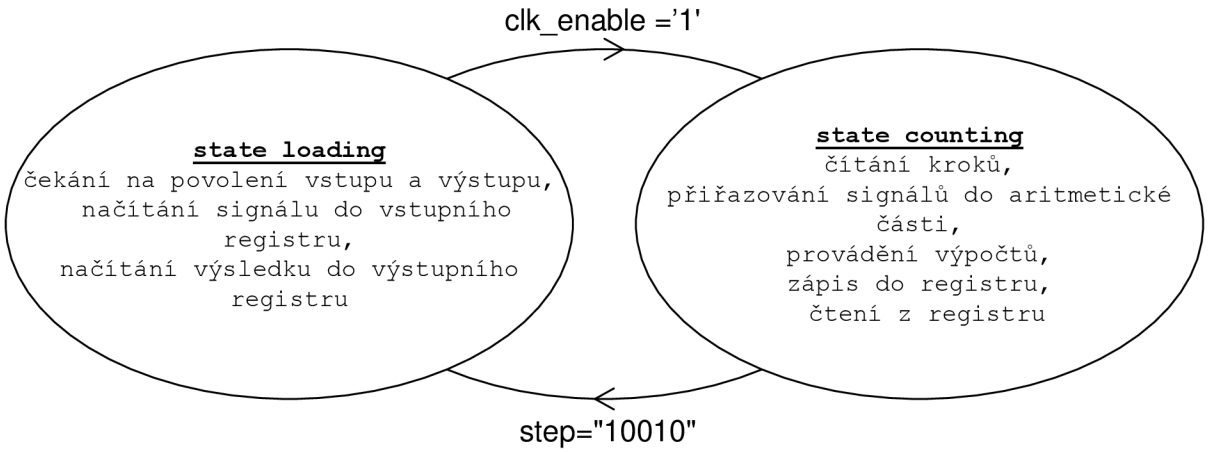
Obrázek 13: Bloková struktura filtru

Na **Obrázku 13** jsou tři hlavní části filtru, paměťová část, aritmetická část a řídící logika, a také vstupní a výstupní registry, které lze považovat za část řídící logiky. Zpracovávaný signál je nejprve nahrán do vstupního registru, odkud ho načítá řídící logika a dle potřeby s ním dále pracuje. Signály z řídící logiky jsou vedeny do aritmetické části a z ní následně do paměťové části. Z paměti čte řídící logika dříve uložené hodnoty a opět odesílá data do aritmetické části. Tento proces probíhá dokola, dokud nejsou provedeny všechny potřebné výpočty. Po dokončení výpočtů je do výstupního registru nahrán výsledek a filtr čeká na další vstupní hodnotu. Jednotlivé části filtru budou popsány dále.

Samotný filtr je navržen jako jednoduchý stavový automat pracující ve dvou stavech. Důvodem použití stavového automatu je nezávislost funkce filtru na použité frekvenci na čipu. Minimální frekvence na čipu je dána počtem kroků výpočtu a vzorkovací frekvencí filtrovaného signálu, v tomto případě je počet kroků 20 a vzorkovací frekvence 15625 Hz. Jednoduchým výpočtem tedy určíme minimální frekvenci, která je vyjádřena jako

$$f_{min} = n * f_{vz} = 20 * 15\,625 = 312\,500\,Hz. \quad (5)$$

Při použití vyšší frekvence, jak tomu v tomto případě bude, dochází k rychlejšímu výpočtu a je nutno čekat na další vstupní hodnotu. Na obrázku je tento stavový automat zakreslen i s jednotlivými činnostmi, které filtr v daném stavu provádí.

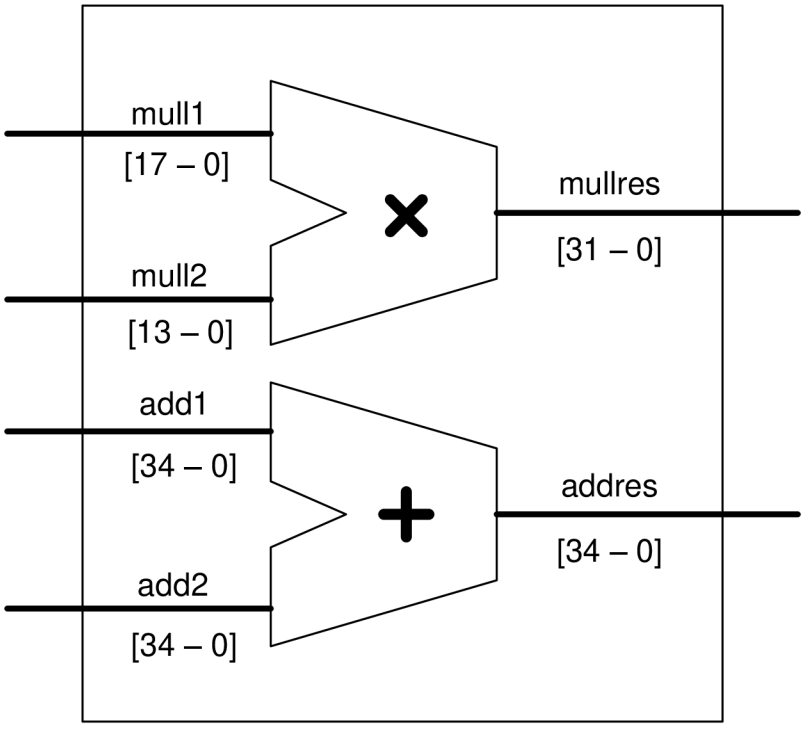


Obrázek 14: Stavový automat

Stavový automat může přecházet mezi dvěma stavy, *loading* a *counting*. V prvním jmenovaném stavu neprobíhá žádné počítání, filtr jen čeká na povolení zápisu vstupního signálu do vstupního registru a povolení načtení výsledku filtrace do výstupního registru, který je řízen signálem *clk_enable*. Pokud je tento signál roven '1', přechází stavový automat do stavu *counting*. V něm pak dochází k samotné filtraci signálu. Pokud jsou provedeny všechny potřebné operace a čítač kroků se dostane k hodnotě 10010, dojde k překlopení do stavu *loading* a k čekání na další vstupní signál.

3.2.1. Aritmetická část

Aritmetická část filtru se stará o potřebné matematické operace při filtraci signálu. Schéma tohoto bloku je na **Obrázku 15**. Je zde použita jedna sčítačka pracující s 35 bitovou šířkou čísel a jedna násobička s osmnáctibitovým vstupem pro koeficienty filtru a čtrnáctibitovým vstupem pro vnitřní signály filtru, výsledek násobení je pak v šířce 32 bitů. Ačkoliv při sčítání dvou 35bitových čísel může vyjít výsledek jako 36bitové číslo, respektive dojde k přetečení prvního bitu, při dalším počítání je tento bit oříznut, a proto není nutné ho ve sčítačce počítat a výsledek sčítání je také 35bitový.



Obrázek 15: Aritmetický blok

Všechny matematické operace jsou realizovány pomocí počítání v pevné řádové čárce (fixed-point), číslo je zakódováno do vektoru číslic, ve kterém první část číslic reprezentuje celá čísla a zbývající část číslic reprezentuje zlomkovou část. Skutečná hodnota tohoto

vektoru je dána váhovým koeficientem. Je to číslo vyjadřující poměr mezi skutečnou hodnotou reálného čísla a hodnotou čísla v pevné řádové čárce. Na následujícím příkladu je ukázáno, jak je koeficient filtru *scale* přepočítán z reálných čísel do pevné řádové čárky.

$$scale = 0.51205205484621141, k = 2^{17} = 131072$$
$$scale_k = scale * k = 67115{,}688 \doteq 67116 \quad (6)$$

Koeficient filtru *scale* je teoretická hodnota vypočtená v Matlabu s maximální přesností na sedmnáct desetinných míst, protože filtr je stabilní při zaokrouhlení koeficientů na 18 bitů, váhový koeficient *k* byl zvolen kvůli maximální možné přesnosti 2^{17}, to znamená, že v 18 bitovém slovu je řádová čárka hned za prvním bitem, který by byl v tomto případě vyhrazen znaménkovému bitu. Výpočet pak probíhá tak, že reálné číslo vynásobíme váhovým koeficientem a zaokrouhlíme na nejbližší celé číslo a toto číslo následně převedeme do dvojkové soustavy. Výpočet reálného čísla z vektoru hodnot v pevné řadové čárce probíhá naopak, číslo ve dvojkové soustavě převedeme na celé číslo v desítkové soustavě a následně vydělíme váhovým koeficientem.

Oproti počítání v plovoucí řádové čárce, používané především v náročnějších aplikacích, je realizace matematických operací v pevné řádové čárce hardwarově jednodušší, zabírá menší část čipu a výsledky jsou pro potřebu filtrace signálu počítány s dostatečnou přesností.

Při sčítání dvou čísel v pevné řádové čárce je vhodné používat stejné váhové koeficienty pro oba sčítance. Při použití odlišných koeficientů, je třeba provést bitový posun tak, aby se sčítali vždy čísla se stejnou váhou.

Při násobení dvou čísel v pevné řádové čárce se zároveň násobí i váhové koeficienty, to znamená, že abychom získali správné číslo v desítkové soustavě, je potřeba výsledné číslo v pevné řádové čárce vydělit součinem váhových koeficientů jednotlivých čísel.

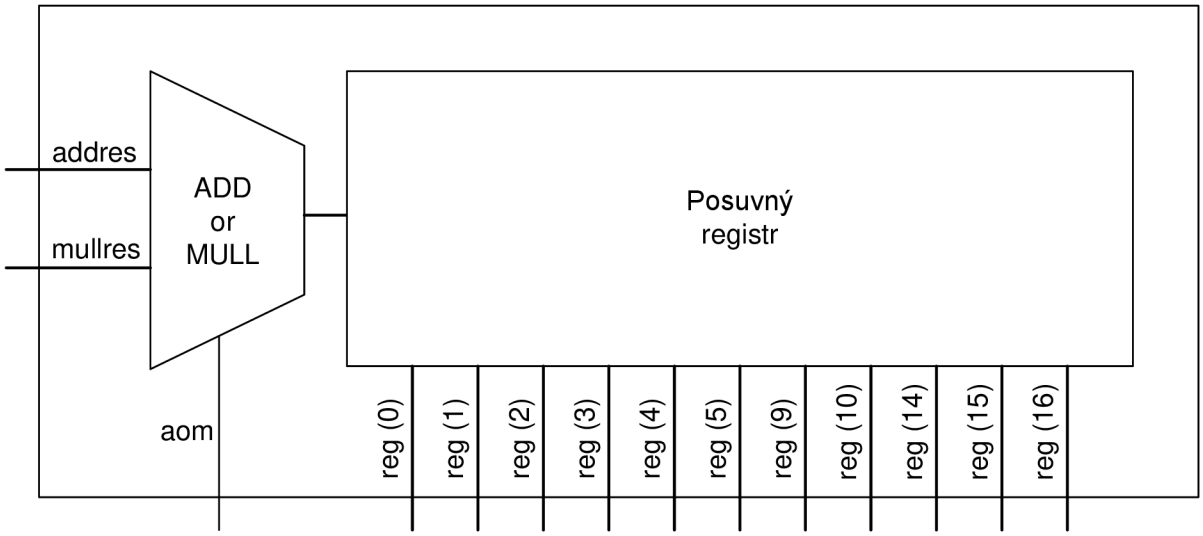
Záporná čísla jsou reprezentována pomocí dvojkového doplňku. V jazyce VHDL je dvojkový doplněk spojen s datovým typem *signed* a matematické operace s tímto datovým typem jsou popsány v knihovně *IEEE.numeric_std.ALL*. Jedná se především o algoritmus pro násobení, který pracuje tak, že nejprve kontroluje, jestli je číslo kladné nebo záporné pomocí znaménkového bitu, v případě kladného čísla provede rovnou násobení, v případě záporného čísla převede toto číslo na kladné, provede násobení s kladným číslem a výsledek převede zpět na záporné číslo ve dvojkovém doplňku. Násobení je realizováno pomocí kombinační logiky, která obstarává bitový posunu a sčítání vzájemně posunutých čísel v jednom hodinovém taktu.

3.2.2. Paměťová část

V paměťové části je jako hlavní komponenta použit posuvný registr, obsahující osmnáct 35 bitových registrů. Výhodou použití posuvného registru je snadný zápis a čtení bez nutnosti řadiče paměti. Výsledky z aritmetického bloku jsou přiváděny přímo na první registr, při hodinovém taktu dojde k posunutí dříve zapsaných hodnot a k zápisu do paměti. Čtení se provádí z jednotlivých paměťových bloků podle momentálního kroku výpočtu.

V jazyce VHDL je posuvný registr popsán následujícím způsobem. Nejdříve je definován typ *shift_reg* jako vektor devatenácti prvků, z nichž každý je složen z 35-bitového signálu typu *signed*. Samotný posun signálu mezi jednotlivými paměťovými bloky je zapsán pomocí smyčky *loop*, které lze rozumět tak, že do následujícího registru je zapsána hodnota toho předchozího, přičemž se využívá základní vlastnost jazyka VHDL, že všechny příkazy probíhají souběžně a nedojde tedy k vymazání hodnot z následujícího registru. Jedná se o doporučený zápis posuvného registru, který je návrhovým systémem vždy vyhodnocen jako posuvný registr. Do prvního registru (ve zdrojovém kódu se jedná o signál reg(0)) jsou zapisovány výsledky z aritmetického bloku, přičemž o přiřazení výsledku výpočtu z násobičky nebo sčítačky rozhoduje signál *aom*, který vystupuje z řídící logiky. Vzhledem k 35 bitové šířce registru je výsledek násobení, který je ve 32 bitové šířce, rozšířen o tři bity, aby vždy došlo ke korektnímu zápisu do registru. Všechny tyto operace jsou synchronní a jsou tedy prováděny při hodinovém taktu.

```
type shift_reg is array (18 downto 0)                -- deklarace registru
      of signed (34 downto 0);                          a typu signed
                          ⋮
if (clk'event AND clk = '1' and fil_en ='1') then
            for i in 0 to 17 loop                    -- smyčka loop
                  reg(i+1) <= reg(i);                -- posun dat
            end loop;                                   v registru
    if (aom ='1') then
            reg(0) <= '0' & '0' & '0' & mullres;     -- přiřazení výpočtu
    else                                                do registru
            reg(0) <= addres;
     end if;
end if;
```

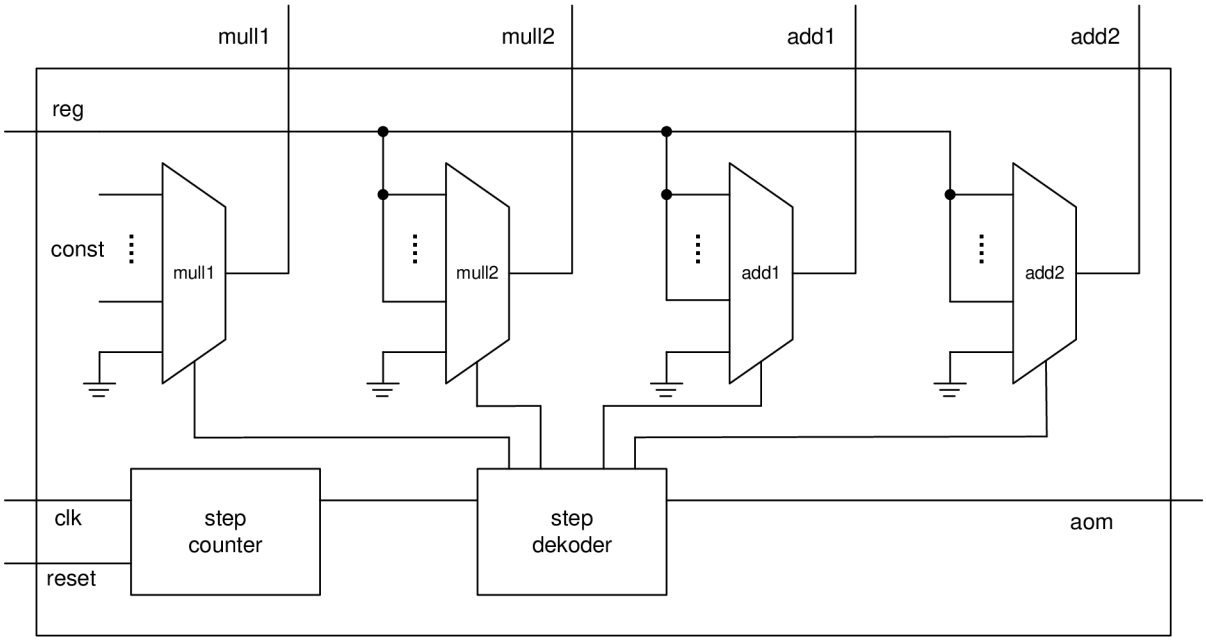


Obrázek 16: Struktura paměťové části

Na **Obrázku 16** je nakreslena struktura paměťové části. V ní jsou obsaženy dva hlavní bloky. První blok, označený jako *ADD or MULL*, rozhoduje, výsledek které matematické operace je zapisován do paměti. Jedná se o multiplexor, který je řízen z řídící logiky prostřednictvím signálu *aom*. V případě, že signál *aom* = '0', do registru je zapsán výsledek sčítání, v opačném případě je zapsán výsledek násobení. Tento blok je navržen jako synchronní, výsledek počítání je poslán do posuvného registru až při následujícím hodinovém taktu. Druhým blokem je dříve zmiňovaný posuvný registr. Řídící logika využívá současně nejvýše dva registry z celého posuvného registru, v každém kroku se však jedná o jejich rozdílnou kombinaci a z paměťové části vede jedenáct signálů do řídící logiky, která s nimi pracuje podle potřeby.

3.2.3. Řídící logika

Řídící logika slouží k ovládání dalších částí filtru. Jejím úkolem je číst data z registrů a následně zajišťovat jejich přísun do aritmetické části podle kroku, v kterém se právě výpočet nachází. Pro určení kroku je v řídící logice čítač, podle kterého řídí dekodér čtyři multiplexory, které přiřazují data z registrů do výstupních signálů. Vstupními signály do řídící logiky jsou hodinový signál, signál *reset* a data z registrů, výstupní signály jsou pak data pro aritmetický blok – *mull1, mull2, add1* a *add2* a do paměťového bloku vede signál *aom*, který řídí zápis výpočtu do registru. Vzájemné propojení jednotlivých bloků je na **Obrázku 17**.



Obrázek 17: Řídící logika

Blok *step counter* je jednoduchý pětibitový čítač, který čítá do hodnoty 19, což je počet kroků potřebných ke všem výpočtům i k zápisu poslední vypočtené hodnoty do registru. Vstupem do tohoto bloku je signál *reset* pro vynulování čítače a hodinový signál, při kterém dochází k navýšení hodnoty na výstupu čítače. V jazyce VHDL je čítač zapsán opět doporučeným zápisem, aby při překladu nedošlo k chybě, kombinační a paměťová část jsou odděleny.

```
if (step = "10011") the            -- omezení počtu kroků
      step<="00000";
elsif (clk'event AND clk = '1'     -- synchronní část, při hodinovém taktu
       and fil_en ='1') then       dojde k načtení hodnoty z kombinační
         step <= step_D;           logiky na výstup
end if;

step_D <= step + 1;                -- kombinační část, inkrementace kroku
```

Výstupní signál z bloku *step counter* je veden do dalšího bloku, kterým je *step dekoder*. Tento blok ovládá jednotlivé multiplexory přiřazující data do signálů pro aritmetický blok a tímto rozhoduje o tom, který výpočet bude prováděn v závislosti na momentálním kroku. Kromě toho také generuje již zmiňovaný signál *aom*.

Poslední částí řídící logiky jsou už zmiňované multiplexory sloužící k přiřazení dat do signálů vstupujících do aritmetického bloku. V případě multiplexoru *mull1* jsou jako vstupní

hodnoty použity koeficienty filtru, které jsou uloženy přímo na čipu pomocí připojení signálů do logické nuly nebo logické jedničky. Vstupní data do dalších tří multiplexorů jsou data přivedená z registrů. Na **Obrázku 17** jsou do každého multiplexoru přivedeny jen dva signály z důvodu větší přehlednosti obrázku, ve skutečnosti do multiplexoru *mull2* vstupují čtyři signály, do multiplexoru *add1* vstupuje sedm signálů z registrů a do multiplexoru *add2* tři signály. Kromě signálů z registrů, případně konstant u multiplexoru *mull1*, je u všech multiplexorů jeden vstupní signál nulový, na **Obrázku 17** je označen symbolem uzemnění. V případě posledních dvou jmenovaných částí řídící logiky, *step dekoderu* a multiplexorů *mull1* až *add2*, se prakticky jedná o implementaci dříve popsaného algoritmu do hardwaru, jednotlivé kroky odpovídají krokům na **Obrázku 12**, ale je v nich už uvažováno posunutí dat v registrech.

Ve zdrojovém kódu v jazyce VHDL jsou multiplexory i dekodér popsány společně pomocí jednoho příkazu *case*. Z tohoto zdrojového kódu bude popsána jen část, protože další kroky fungují obdobně. Celý zdrojový kód je pak k nalezení v příloze. Na začátku procesu jsou nejprve výstupní signály nastaveny do nuly, respektive u signálu *aom* do hodnoty jedna, přičemž se využívá jedné vlastnosti procesu, a to že k přiřazení do signálu dochází až na konci celého procesu. Pokud je v daném kroku potřeba násobit, jako je tomu v prvním kroku, data jsou přiřazeny do signálů *mull1* a *mull2*, do signálů *add1* a *add2* jsou přiřazeny nuly, které byly nastaveny na začátku procesu, což znamená, že sčítačka nic nedělá. Signál *aom* v tomto kroku není řešen a jeho hodnota je ‘1‘, jak byl definováno na začátku procesu. V druhém kroku je to naopak, data jsou odesílány do sčítačky a na vstup násobičky jsou přivedeny nuly definované na začátku procesu, signál *aom* je nastaven do hodnoty ‘0’, aby byl do registru zapisován výsledek sčítání. Kvůli syntéze je nutné, aby byly řešeny všechny hodnoty čítače, tedy i takové, které by nastat neměly. V případě, že by se na výstupu čítače objevily hodnoty vyšší než 18, což je počet všech kroků a také hodnota, při které dochází k resetování čítače, do jednotlivých signálů jdoucích do aritmetického bloku budou přiřazeny nuly.

```
step_mul: process (step,mull1,mull2,mullres,input_register,reg)
  begin
   aom <= '1';
   mull1 <= (others => '0');
   mull2 <= (others => '0');
   add1  <= (others => '0');
   add2  <= (others => '0');
```

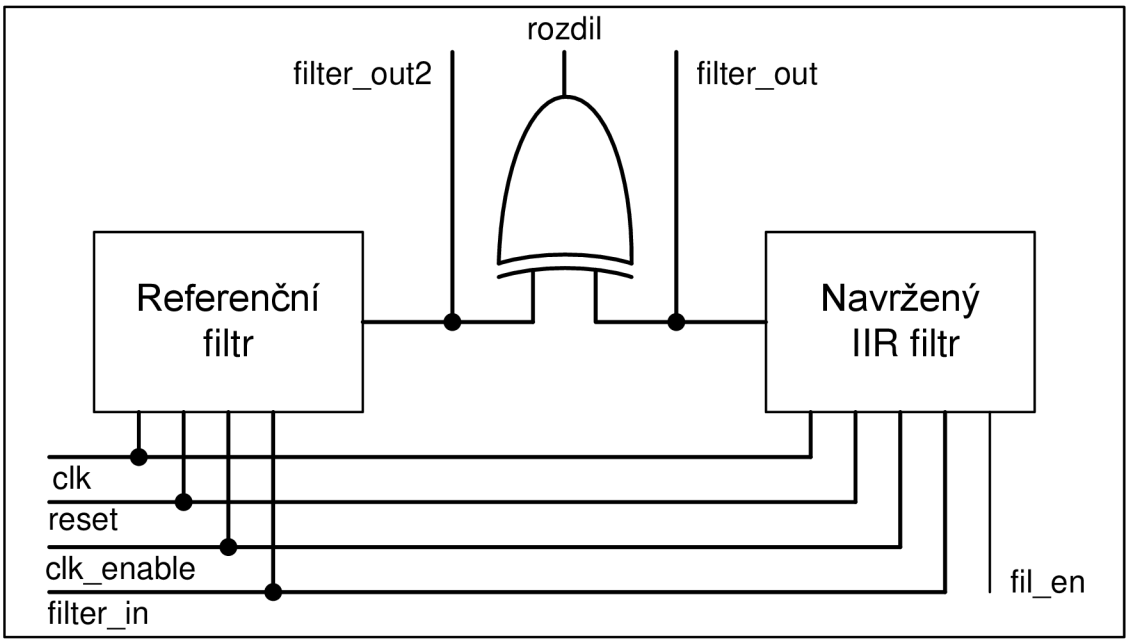
```
case step is
   when "00001" =>
      mull1 <= scale;
      mull2 <= input_register;
   when "00010" =>
      aom <='0';
      add1 <= resize(reg (0)(30 DOWNTO 17) & "0000000000000000", 35);
      add2 <= resize(reg(16) (33 downto 20) & "00000000000000000000",35);
       ⋮
   when others =>
      aom <='0';
      mull1 <= (others => '0');
      mull2 <= (others => '0');
      add1 <= (others => '0');
      add2 <= (others => '0');
 end case;
end proces step_mul;
```

4. Verifikace

Pod pojmem verifikace lze rozumět kontrolu správnosti nějakého systému vzhledem k požadavkům na jeho fungování. Metod verifikace systému existuje více druhů, v tomto případě bude kontrola správnosti provedena pomocí referenčního filtru, který byl popsán v **kapitole 3.1** a to takovým způsobem, že budou porovnávány výstupní hodnoty referenčního a navrženého filtru.

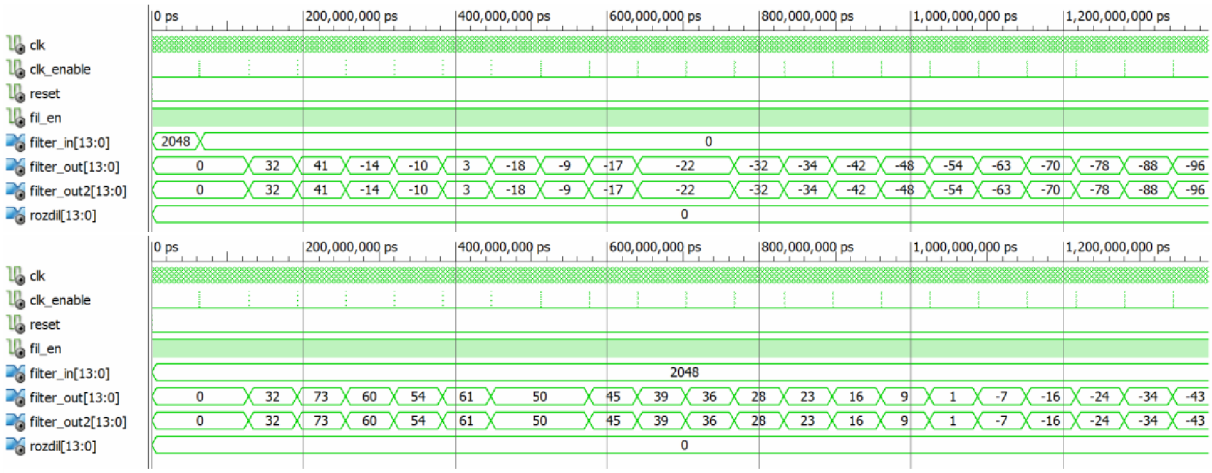
Pro kontrolu správnosti funkce obvodu se používají simulace. V jazyce VHDL je vstupní soubor pro simulace tzv. test bench, který lze vygenerovat přímo v návrhovém prostředí ISE Webpack. Testovací soubor má stejnou strukturu jako každý jiný soubor v jazyce VHDL, obsahuje dvě základní části, deklaraci entity a tělo architektury. Entita je však u testovacích souborů prázdná – do souboru nevstupují ani z něj nevystupují žádné signály, všechny data jsou generována uvnitř a testované soubory jsou vkládány jako komponenty do těla architektury.

Pro kontrolu správnosti zde navrženého filtru byl vytvořen testovací soubor, který lze nalézt v příloze. Oba filtry byly vloženy jako komponenty a byly do nich přivedeny stejné signály. Pro porovnání výsledků simulace byl použit exklusivní logický součet – operátor XOR. V případě, že je výstupní hodnota referenčního i navrhovaného filtru stejná, rozdílový signál popsaný operátorem XOR je nulový. V případě, že by se při návrhu filtru vyskytla chyba, signál *rozdil* by změnil hodnotu a na místě chybného bitu by se objevila jednička. Zapojení jednotlivých bloků v testovacím souboru je na **Obrázku 18.**



Obrázek 18: Zapojení filtrů v testovacím souboru

Simulace filtrů byla provedena v programu ISim, který je stejně jako návrhové prostředí ISE Webpack vytvořen firmou Xilinx. Výsledek simulace je na **Obrázku 19**.



Obrázek 19: Simulace

Simulace byla provedena pro dva vstupní signály, v horní části obrázku je vykreslena odezva filtru na jednotkový impuls a ve spodní části obrázku je odezva na jednotkový skok. Hodinový signál *clk* a signál *clk_enable* mají pro potřeby simulace stejné parametry, jako by měly mít při implementaci na čipu, v případě signálu *clk* je perioda 0,5 µs, signál *clk_enable* je v testovacím souboru popsán délkou pulzu 0,5 µs a délkou trvání nuly po dobu 63,5 µs. Pro správnou funkci filtru je třeba ho nejprve resetovat a do všech registrů nastavit nuly, pro simulaci je signál *reset* do hodnoty '1' nastavena jen po dobu 1 ns, která by samozřejmě při realizaci na čipu nebyla dostatečná. Signál *fil_en* je nastaven do hodnoty '1' po celou dobu simulací. Pro simulaci jednotkového impulsu i jednotkového skoku je použit téměř stejný zdrojový kód, v příloze je pak u příslušného řádku poznámka, že jeho zakomentováním lze vytvořit z jednotkového impulsu jednotkový skok.

5. Závěr

V práci bylo rozebráno téma IIR filtrů, jejich vlastností a způsobu jejich návrhu v prostředí Matlab. Pro konkrétní parametry požadovaného filtru bylo provedeno několik základních analýz, koeficienty filtru byly kvantovány pro implementaci do hardwaru a takto navržený filtr byl exportován do jazyka VHDL. Po exportu se projevila nevýhoda automatického generování filtrů, návrh obsahoval sedm násobiček a jedenáct sčítaček, pro implementaci do hardwaru to není příliš vhodné. Tento filtr však slouží jako referenční filtr pro kontrolu správnosti ručního návrhu filtru.

V druhé části práce byl ukázán návrh filtru v návrhovém prostředí ISE Webpack a jazyce VHDL. Byl popsán algoritmus filtrace, který umožnil použití jedné násobičky a jedné sčítačky a s tím spojenou úsporou místa na čipu. Zároveň byly rozebrány a popsány jednotlivé části filtru i s jejich zápisem v jazyce VHDL. V poslední části práce byla provedena verifikace návrhu a s pomocí simulací byla ukázána správná funkčnost navrhovaného filtru.

Seznam použité literatury

[1] LYONS, Richard G. *Understanding Digital Signal Processing*. 2nd ed. New Jersey : Bernard Goodwin, 2004. 665 s. ISBN 0-13-108989-7.

[2] *Mathworks* [online]. 2010 [cit. 2010-12-14]. Signal Processing Toolbox™ 6 User's Guide. Dostupné z WWW: <http://www.mathworks.com/help/pdf_doc/signal/signal_tb.pdf>.

[3] Digital filter. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 2002-02-25, last modified on 2010-12-12 [cit. 2010-12-14]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Digital_filter>.

[4] Infinite impulse response. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 204-04-01, last modified on 2010-11-29 [cit. 2010-12-14]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Infinite_impulse_response>.

[5] HAAG, Michael. *Connexions* [online]. 2010-06-22 [cit. 2010-12-14]. Understanding Pole/Zero Plots on the Z-Plane. Dostupné z WWW: <http://cnx.org/content/m10556/latest/>.

[6] SOVKA, Pavel; ČMEJLA, Roman; ŠMEJKAL, Ladislav. Číslicové filtry. *Automatizace* [online]. 07-2005, 7-8, [cit. 2010-12-14]. Dostupný z WWW: < www.automatizace.cz/download.php?d=QXRtX0FydGljbGUscGRmX2FydCw3OTI= číslicové filtry>.

[7] MEYER-BAESE, Uwe. *Digital Signal Processing with Field Programmable Gate Arrays*. Berlin : Springer, 2007. 774 s.

[8] *Mathworks* [online]. 2010 [cit. 2011-5-31]. Filter Design HDL Coder™ 2 User's Guide. Dostupné z WWW: <http://www.mathworks.com/help/pdf_doc/hdlfilter/hdlfilter.pdf>

[9] Formal verification. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 17. 7. 2003, last modified on 30. 4. 2011 [cit. 2011-05-31]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Formal_verification>.

Příloha

1. Koeficienty filtru Single Section:

```
Single Section:
Numerator:
 0.51205205484621141
 0.00000000000000037747582837255322
-1.536156164538633
-0.00000000000000064392935428259079
 1.5361561645386315
 0.00000000000000039968028886505635
-0.51205205484621086
Denominator:
 1
-1.2863700295825939
-0.99622840588861949
 0.98638828196013129
 0.89963921358078625
-0.34164144714773848
-0.2617875721124423
```

2. Koeficienty filtru Second-Order Section

```
Second-Order Sections
--------------------------
Section #1
--------------------------
Numerator:
 1
 0.000006729045122444443 25
-1.0000091349807454
Denominator:
 1
-0.50497989310934677
-0.49181014885042862
Gain:
0.51205205484621141
--------------------------
Section #2
--------------------------
Numerator:
1
1.9999920679918382
0.99999206805475482
Denominator:
1
1.216457403147571
0.53343970244754668
Gain:
1
--------------------------
Section #3
--------------------------
Numerator:
 1
-1.9999987970369566
 0.99999879703840411
Denominator:
 1
-1.9978475396208242
 0.99785216285321754
Gain:
1
--------------------------
Output Gain:         1
```

3. Zdrojový kód v Matlabu

```
% navrh filtru
Fp  = [10 5000];          % propustne pasmo
Fs  = [0.01 7500];        % frekvence utlumeni
Fvz = 15625;              % vzorkovaci frekvence
Ap = 0.1;                 % zvlneni v propustnem pasmu [dB]
As = 40;                  % potlaceni mezi Fp a Fs [dB]


d=fdesign.bandpass(Fs(1),Fp(1),Fp(2),Fs(2),As,Ap,As,Fvz);
                            % navrh filtru o danych vlastnostech
hd=design(d,'butter');

freqz(hd,Fvz) ;             % amplitudove a fazove spektrum prenosove fce
zplane(hd);                 % vykresleni nulovych bodu a polu v rovine z

% impulsni odezva
imp = [1; zeros(49,1)];     % jednotkovy impuls
h = filter(hd,imp);         % odezva filtru na jednotkovy impuls
figure;
stem(h);

%odezva na jednotkovy skok
one=ones(Fvz,1);            % hodnota Fvz odpovida 1s trvajici odezve
g = filter(hd,one);         % odezva filtru na jednotkovy skok
figure;
plot(g);
```

4. Zdrojový kód filtru v jazyce VHDL

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.ALL;

ENTITY filter_IIR IS
   PORT(clk                    :   IN    std_logic;
        clk_enable             :   IN    std_logic;
        reset                  :   IN    std_logic;
        fil_en                 :   IN    std_logic;
        filter_in              :   IN    std_logic_vector(13 DOWNTO 0);
        filter_out             :   OUT   std_logic_vector(13 DOWNTO 0) );

END filter_IIR;

ARCHITECTURE iir OF filter_IIR IS

  CONSTANT scale             : signed(17 DOWNTO 0) := to_signed(67116,18);
  CONSTANT coeff_a2_section1 : signed(17 DOWNTO 0) := to_signed(-32971,18);
  CONSTANT coeff_a3_section1 : signed(17 DOWNTO 0) := to_signed(-32171,18);
  CONSTANT coeff_a2_section2 : signed(17 DOWNTO 0) := to_signed(79722,18);
  CONSTANT coeff_a3_section2 : signed(17 DOWNTO 0) := to_signed(34959, 18);
  CONSTANT coeff_a2_section3 : signed(17 DOWNTO 0) := to_signed(-131054,18);
  CONSTANT coeff_a3_section3 : signed(17 DOWNTO 0) := to_signed(65519, 18);
  -- Signaly
  SIGNAL step                : signed (4 downto 0); -- pocet kroku 19
  SIGNAL step_D              : signed (4 downto 0); -- pocet kroku 19
  SIGNAL mull1               : signed(17 DOWNTO 0); -- nasobeni
  SIGNAL mull2               : signed(13 DOWNTO 0); -- nasobeni
  SIGNAL mullres             : signed(31 DOWNTO 0); -- vysledek nasobeni
  SIGNAL add1                : signed(34 DOWNTO 0); -- scitani
  SIGNAL add2                : signed(34 DOWNTO 0); -- scitani
  SIGNAL addres              : signed(34 DOWNTO 0); -- vysledek scitani
  SIGNAL input_register      : signed(13 DOWNTO 0); -- vstupni registr
  SIGNAL output_register     : signed(13 DOWNTO 0); -- vystupni registr
  SIGNAL aom                 : std_logic;           -- nasobeni nebo scitani
```

```
  type shift_reg is array (18 downto 0)  -- posuvny registr
                    of signed (34 downto 0);
  type state_type is (loading,counting); -- stav automat
             signal reg : shift_reg;
             signal next_state, present_state : state_type;

BEGIN

state: process (present_state,next_state,clk,fil_en) --preklapeni stavu
  begin
       if (clk'event and clk = '1' and fil_en ='1') then
             present_state <= next_state;
       end if;
  end process;

process (clk, reset,step_D,step,clk_enable,present_state,filter_in,fil_en)

  begin
   case present_state is                        -- stavovy automat

       when counting =>                         -- stav pocitani

             if step="10010" then               --zmena stavu po dokonceni vypoctu
                next_state <= loading;
             else
                next_state <= present_state;
             end if;

             if reset = '1' then                -- asynchronni reset pro stav counting
                    next_state <= loading;
                    step <= (others => '0');
                    for i in 0 to 18 loop
                    reg(i) <= (others=>'0');
                    end loop;
                    input_register <= (others => '0');
                    output_register <= (others => '0');

             elsif step = "10011" then          --citac kroku - omezeni
                      step<="00000";

             elsif (clk'event AND clk = '1' and fil_en ='1') then
                    for i in 0 to 17 loop       -- posuv registru
                       reg(i+1) <= reg(i);
                    end loop;
                       step <= step_D;          -- citac kroku - synchronni cast

                    if (aom ='1') then          -- prirazeni vysledku do registru
                       reg(0) <= '0' & '0' & '0' & mullres;
                    else
                       reg(0) <= addres (34 downto 0);
                    end if;

             end if;

       when others =>                           -- stav nahravani vstupu a vystupu

             if clk_enable = '1' then           -- povoleni vstupuv-> stav pocitani
                next_state <= counting;
             else
                next_state <= present_state;
             end if;

       if reset = '1' then                      -- asynchronni reset pro stav loading
             input_register <= (others => '0');
             output_register <= (others => '0');
             step <= (others => '0');
             for i in 0 to 18 loop
                reg(i) <= (others=>'0');
```

```
            end loop;
            next_state <= loading;

      elsif (clk'event and clk = '1' and clk_enable = '1' and fil_en = '1') then
            input_register <= signed(filter_in);
            output_register <= resize(reg(5) (29 downto 21),14);
                              -- nacteni do vstupniho a vystupniho registeru
      end if;
  end case;
end process;

   mullres <= mull1 * mull2;           -- nasobeni
   addres  <= add1 + add2;             -- scitani
   step_D <= step + 1;                 -- inkrementace kroku

 step_mul: process (step,mull1,mull2,mullres,input_register,reg)
  begin
   aom <= '1';                         -- nastaveni signalu do defaultni hodnoty
      mull1 <= (others => '0');
      mull2 <= (others => '0');
      add1  <= (others => '0');
      add2  <= (others => '0');
      case step is
            when "00001" =>           --nasobeni vsupu skalovaci konstantou
               mull1 <= scale;
               mull2 <= input_register;
            when "00010" =>           --vysledek prvni sekce
               aom <='0';
               add1 <= resize(reg(16) (33 downto 20) & "00000000000000000000",35);
               add2 <= resize((reg (0)(30 DOWNTO 17) & "0000000000000000"), 35);
            when "00011" =>
               mull1 <= coeff_a2_section1;
               mull2 <= reg (0)(29 DOWNTO 16);
            when "00100" =>
               aom <='0';
               add1 <= resize (reg(16)(33 downto 20) & "00000000000000000000",35);
               add2 <= -resize((reg(0)(31 downto 0)),35);
            when "00101" =>
               mull1 <= coeff_a3_section1;
               mull2 <= reg (2)(29 DOWNTO 16);
            when "00110" =>
               aom <='0';
               add1 <=resize(-(reg(4)(30 downto 17) )&"0000000000000000",35);
               add2 <=-resize(reg(0)(31 downto 0),35);
            when "00111" =>                       --vysledek druhe sekce
               aom <='0';
               add1 <= resize ((reg(4) (29 downto 16)& "0000000000000000"),35);
               add2 <= resize ((reg(15)(33 downto 20)&"00000000000000000000"),35);
            when "01000" =>
               aom <='0';
               add1 <= resize((reg(5)(29 downto 16)& "00000000000000000"),35);
               add2 <= resize (reg(14)(33 downto 20) & "00000000000000000000",35);
            when "01001" =>
               mull1 <= coeff_a2_section2;
               mull2 <= reg (1)(29 DOWNTO 16);
            when "01010" =>
               aom <='0';
               add1 <= reg(1);
               add2 <= - resize(reg(0) (31 downto 0),35);
            when "01011" =>
               mull1 <= coeff_a3_section2;
               mull2 <= reg (3)(29 DOWNTO 16);
            when "01100" =>
               aom <='0';
               add1 <= resize((reg(9)(29 downto 16) & "0000000000000000"), 35);
               add2 <= -resize(reg(0)(31 downto 0),35);
```

```
            when "01101" =>           --vysledek treti sekce a celeho filtru
               aom <='0';
               add1 <= resize (reg(5)(29 downto 16) & "0000000000000000",35);
               add2 <= resize (reg(15)(33 downto 20) & "00000000000000000000",35);
            when "01110" =>
               aom <='0';
               add1 <= -resize(shift_left(reg(6),1),35);
               add2 <= resize (reg(14)(33 downto 20) & "00000000000000000000",35);
            when "01111" =>
               mull1 <= coeff_a2_section3;
               mull2 <= reg (1)(29 DOWNTO 16);
            when "10000" =>
               aom <='0';
               add1 <= reg(1);
               add2 <= -resize(reg(0)(31 downto 0),35);
            when "10001" =>
               mull1 <= coeff_a3_section3;
               mull2 <= reg (3)(29 DOWNTO 16);
            when "10010" =>
               aom <='0';
               add1 <= resize((reg(10)(29 downto 16) &"0000000000000000"),35);
               add2 <= -resize(reg(0) (31 downto 0), 35);
            when others =>            --nestandardni stavy, cekani na vstup
               aom <='0';
               mull1 <= (others => '0');
               mull2 <= (others => '0');
               add1 <= (others => '0');
               add2 <= (others => '0');
            end case;
  end process step_mul;

  filter_out <= std_logic_vector(output_register);   --vystupni registr -> vystup
END iir;
```

5. Testovací soubor

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;

ENTITY test IS
END test;

ARCHITECTURE behavior OF test IS
    COMPONENT filter_IIR         -- vlozeni IIR filtru
    PORT(
         clk : IN  std_logic;
         clk_enable : IN  std_logic;
         reset : IN  std_logic;
         fil_en : IN  std_logic;
         filter_in : IN  std_logic_vector(13 downto 0);
         filter_out : OUT  std_logic_vector(13 downto 0)
        );
    END COMPONENT;

        COMPONENT filter_ref     -- vlozeni referencniho filtru
    PORT(
         clk : IN  std_logic;
         clk_enable : IN  std_logic;
         reset : IN  std_logic;
         filter_in : IN  std_logic_vector(13 downto 0);
         filter_out : OUT  std_logic_vector(13 downto 0)
        );
    END COMPONENT;
```

```
      --vstupni signaly
   signal clk : std_logic := '0';
   signal clk_enable : std_logic := '0';
   signal reset : std_logic := '0';
   signal fil_en : std_logic := '1';
   signal filter_in : std_logic_vector(13 downto 0) := (others => '0');

      --vystupni signaly
   signal filter_out : std_logic_vector(13 downto 0);      --IIR filtr
   signal filter_out2 : std_logic_vector(13 downto 0);     --referencni filtr

   signal rozdil : std_logic_vector(13 downto 0);
   constant clk_period : time := 500 ns;             -- frekvence = 2MHz

begin
   uut: filter_IIR port map (
          clk => clk,
          clk_enable => clk_enable,
          reset => reset,
          fil_en => fil_en,
          filter_in => filter_in,
          filter_out => filter_out
        );
       uut2: filter_ref port map (
          clk => clk,
          clk_enable => clk_enable,
          reset => reset,
          filter_in => filter_in,
          filter_out => filter_out2
        );

  clk_process : process      -- definice hodinoveho signalu
   begin
            clk <= '1';
            wait for clk_period/2;
            clk <= '0';
            wait for clk_period/2;
   end process;

  IO_process : process           -- definice clk_enable signalu
      begin
       clk_enable <='0';
            wait for 63.5 us;
       clk_enable <='1';
            wait for 0.5 us;
   end process;

  stim_proc: process             --reset a vstup dat
   begin
      reset<='1';
     wait for 1 ns;
      reset<='0';
     wait for clk_period;
      filter_in<="00100000000000";
     wait for 64 us;
      filter_in<="00000000000000";--zakomentovanim radku vznika jednotkovy skok
         wait;
   end process;
 rozdil <= filter_out2 xor filter_out;
END;
```