VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

GENERÁTOR HUDBY A ZVUKOVÉ EFEKTY

GENERATOR OF MUSIC AND SOUND EFFECTS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **NIKITA VAŇKŮ**
AUTHOR

VEDOUCÍ PRÁCE — **Prof. Dr. Ing. PAVEL ZEMČÍK,**
SUPERVISOR

BRNO 2018

Abstrakt

Cílem práce je zpracovat návrh digitalního syntetizátoru a modulátoru na vestavěném systému. Práce nejdříve existující digitální syntetizéry a modulátory ve vestavěných systémech a následně se věnuje návrhu možného syntetizéru a modulátoru na programovatelných hradlových polích.

Abstract

The aim of this work is to design digital synthesizer and modulator on embedded systems. Work is exploring existing digital synthesizer and modulators in embedded systems and PC and with that gained knowledge is presenting possible solution of design on Field Programmable Gate Array.

Klíčová slova

Syntéza zvuku, Syntetizátor, MIDI, Simulátor, FPGA, Zvukové efekty, Wavetable, Sampler.

Keywords

Sound synthesis, Synthesizer, MIDI, Simulator, FPGA, Sound effects, Wavetable, Sampler.

Citace

VAŇKŮ, Nikita. *Generátor hudby a zvukové efekty.* Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Prof. Dr. Ing. Pavel Zemčík,

Generátor hudby a zvukové efekty

Prohlášení

Prohlašuji, že jsem tuto Semestrální práci vypracovala samostatně pod vedením pana profesora Pavla Zemčíka. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

.....
Nikita Vaňků
23. května 2018

Poděkování

Velmi děkuji mému vedoucímu práce, panu profesoru Zemčíkovi za jeho podporu a nekonečnou trpělivost. Také chci poděkovat pánům inženýrům Martinu Musilovi a Svetozáru Noskovi za jejich expertízu v oblasti FPGA.

Obsah

Kapitola 1

Úvod

Běžné hudební nástroje jsou přímočaré ve svém používání, uživatel používáním nástroje živě reprodukuje zvuk. V elektronickém světě se situace značně komplikuje, zařízení určená pro produkci elektronické hudby, tedy její syntézy mají dnes mnoho podob. Může se jednat o robustní nástroje pro práci ve studiu, či kompaktní zařízení vhodná pro živou reprodukci, ale i o profesionální zařízení určená k zpracování zvuku na koncertech. Z tohoto důvodu se zařízení ve svých návrzích od sebe velmi liší, i uživatelé pochází z různých odvětví a mají na tyto zařízení různé požadavky.

Nejznatelnější rozdíl je vidět na první pohled. Zařízení může být software součástí počítačového systému či se může jednat o vestavěné zařízení specializované pouze pro hudební produkci. Ačkoliv jsou dnes počítačové systémy na rozmachu, vestavěné zařízení se i dnes těší velké oblibě. Tím ale rozdíly nekončí, zkoumáním vestavěných zařízení je zřejmé, že neexistuje jednotný koncept jak takové zařízení má vypadat a jakým způsobem má zvuk pracovat. I dnes v moderní digitální se době se například můžeme setkat jak se zařízeními analogovými tak digitálními.

Tato práce si klade za cíl studiem prozkoumat tyto zařízení a následně zařízení realizující syntézu a modulaci zvuku navrhnout s využitím technologie programovatelných hradlových polích.

V kapitole 2 budu rozebírat existující typy zařízení volně dostupných na trhu, zařízení jak analogová tak digitální. V kapitole 3 se zaměřím na možný návrh takového zařízení. V této práci se nevěnuji návrhu jednoho konkrétního nástroje, ale generickému návrhu těchto zařízení obecně, pokusím se tedy pokrýt základní metody syntézy a modulace, které jsou využity v dnešních zařízeních a které také mohu využít dále ve své práci. Velmi často jsou také digitální zařízení realizovaná na mikroprocesorech. V tomto ohledu mě zajímala otázka zda-li by nebylo možné využít jiné postupy návrhu digitálních vestavěných obvodů, v kapitole 4 zaměřím se hlavně na technologii programovatelných hradlových polí FPGA. Budu zjišťovat jestli je tento postup uskutečnitelný a pokud je tak v čem je výhodný či naopak jaké jsou jeho úskalí. Toho chci dosáhnout pomocí simulace a hardwarové implementace. Veškeré zjištěné poznatky nakonec rozeberu a zhodnotím v závěru 5.

Kapitola 2

Existující zvukové syntetizéry a modulátory

Syntéza zvuku se dá popsat jako vytváření audio signálu na základě programových vstupů. Programové vstupy mohou přicházet přímo z uživatelských vstupů např. elektronické piáno kde uživatel stiskem kláves na klaviatuře vygeneruje požadavek na syntézu noty. Programový vstup může být ale řešen vstupem externím přes sběrnici, dnes je v tomto směru standard MIDI. I interní sekvencer, tedy zařízení vysílající programové vstupy v opakované sekvenci může sloužit jako programové vstup.

Ať už je vstup jakýkoliv na výstupu syntetizéru lze očekávat audio výstup se syntetizovaným zvukem.

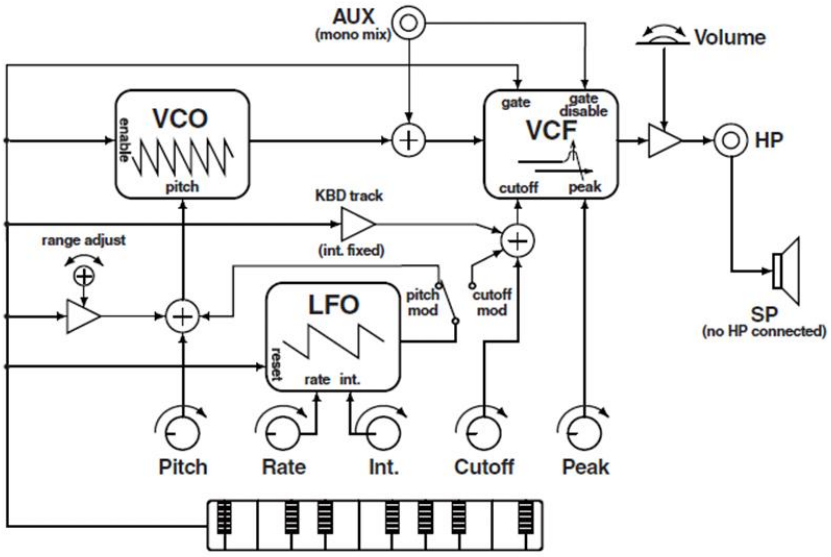
Modulace zvukového signálu se dá pochopit jako změnu již existujícího zvukového signálu, například její filtraci, časovou modulací, fázovou modulací apod. Častokrát je vhodné tyto modulace kombinovat k dosažení cíleného efektu. Jako zařízení, které provádí pouze modulaci zvuku si můžeme představit kytarové pedálové efekty s audio vstupem, audio výstupem a ovládacím prvkem v podobě posuvných či otočných rezistorů a spínačů či tlačítek. Na výstupu očekáváme modulovaný audio výstup.

Digitální zvukové zařízení na trhu velmi často kombinují možnosti syntézy a modulaci zvuku tak jak je to pro zařízení výhodné. Např. u digitálních sampleru může být vhodnější využít modulace s dolno propustním filtrem a časovým zpožděním než implementace fázového posunu. Jak také uvidíme v dalším rozboru existujících zařízení, tyto dogmata se také mohou časem měnit.

V následujících odstavcích je uvedeno a přiblíženo několik populárních audio syntetizérů. Vybrané jsou produkty značky Korg, k těmto produktům jsem se osobně dostala mohla se s nimi blíže seznámit a také jsem měla možnost jednat s dlouhodobými uživateli těchto produktů. Vybrané zařízení samozřejmě ani z malé části nepokryjí typy běžných zařízení volně dostupné. Tyto zařízení byla zvoleny především z důvodu jejich dostupnosti a zároveň propracované uživatelské i technické dokumentace, častokrát včetně diagramu zapojení a dalšími technickými detaily.

2.1 Analogové zařízení

Pro bližší studium zařízení v minimálním provedení je vhodné se seznámit se zařízeními analogovými. Z důvodu své architektury jsou tyto zařízení ve svých vlastnostech velmi minimalistická oproti zařízením digitálním. Krajnějším případem je například Korg Monotron.



Obrázek 2.1: Blokové schéma Korg Monotron

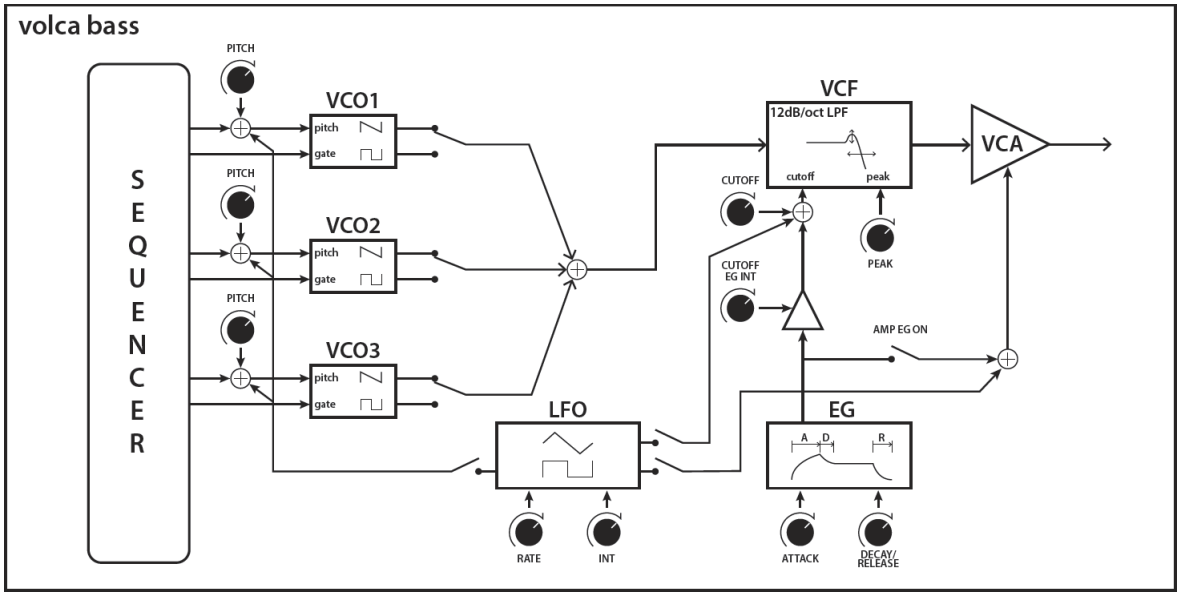
Na obrázku 2.1 je zobrazeno blokové schema zařízení Korg Monotron. Lze vidět, že celé zařízení je sestaveno ze dvou oscilátorů. Vysokofrekvenční oscilátor generuje tón na základě ribbon potenciometru a nízkofrekvenční oscilátoru (dále jen LFO) modulující parametry filtru.

Toto zařízení jsem neměla možnost vyzkoušet a nemohla jsem změřit jaká je maximální frekvence obou oscilátorů, ale pro představu lze předpokládat, že u hlavního oscilátorů je rozsah dán 0. až 7. oktávou, tedy 16 Hz až 4.1 kHz. U LFO se naopak častokrát setkáváme s oscilátory s rozsahem 0 Hz až 12 Hz.

Poslední prvek na tomto zařízení je dolno propustní filtr se zpětnou vazbou. Zařízení umožňuje zapojit výstup LFO na modulaci hlavního oscilátoru či pracovního bodu filtru. Pro představu celé zařízení je velké asi jako sluneční brýle, napájí se z baterie, má audio vstup, výstup a vestavěný reproduktor. Nelze nepřehlédnout, že celá realizace syntézy zvuku je mono. [10]

Obrázek 2.2: Korg Volca Bass

Na obrázku 2.2 je vybrán pro srovnání robustnější syntetizér Korg Volca Bass (dále jen Volca Bass), který je také analogový, ale obsahuje řadu prvků navíc včetně interního sekvenceru a externího vstupu MIDI. Oba tyto prvky u Korg Monotron chyběly. Interní sekvencer umožňuje naprogramovat sekvenci not v 16 stepovém patternu. Sekvencer je řízen interním hodinovým signálem, který lze uživatelem měnit. Zařízení lze také připojit jako MIDI periférii a interní sekvencer zcela eliminovat. Zapojením MIDI lze tyto hudební zařízení škálovat, lze je připojit k dalším zařízením či například k počítači. Sběrnici MIDI její využití a implementaci budu dále rozebírat v kapitole 3.8



Obrázek 2.3: Korg Volca Bass

Na obrázku 2.3 lze vidět blokové schéma realizace zvukové syntézy Volca Bass. Volca Bass má hlavní oscilátory tři, díky tomu lze například hrát například celý akord. Oscilátory lze nastavit na generování pily či obdélníků. LFO je zde zapojen pouze na pracovní bod dolno-propustního filtru. Ovšem filtr má také zapojen Attack Decay Release (dále jen

ADR) obálku. To znamená, že lze modulovat postupné otevírání a uzavírání filtru, tímto lze dosáhnout jemnějších přechodů mezi otevřeným a uzavřeným filtrem. [11]

Byly uvedeny dvě realizace zvukového syntetizéru. Je mezi nimi vidět podobnosti, další realizace jsou častokrát velmi podobné jako realizace Volca Bass. Občas lze nalézt jiný počet oscilátorů, u digitálních syntetizérů se lze setkat také generátoru trojúhelníku, sinusoidy či třeba šumu.

2.2 Digitální zařízení

Digitální zvukové syntetizéry a modulátory lze nalézt častokrát komplexnější a úplnější ve svém návrhu. Příkladem je existence tzv. GrooveBoxu což je elektronické hudební zařízení, které obsahuje sampler/syntetizér, sequencer a banku zvukových efektů v jednom. Je zřejmé že škála možnosti využití těchto nástroji je značně širší. Jako příklad jednoho z GrooveBoxu uvedu zde stále oblíbené zařízení Korg Electribe MX 1 (dále jen EMX) které vyšlo v roce 2010.

Na zařízení EMX kromě hraní několika stop syntetizéru a samplerů najednou je možno využít i řadu zvukových efektů, jejich implementace na EMX je provedena na třech DSP procesorech, tedy v jedné chvíli je možné na celém zařízení využít pouze tři efekty. [9]

Takové omezení je vidět i v dnešních iteracích těchto zařízení, například u modelu Korg electribe, který vyšel v roce 2014 jako další iterace řady Electribe. Na tomto zařízení je možné využít zvukových efektů dohromady 16, ovšem pouze maximálně jeden na každý ze stop tohoto groveboxu. [12]

Je zřejmé, že digitální syntetizéry a modulátory oproti svých analogových protějškům nabízí větší robustnost ve svých možnostech, ale současné implementace se stále potýkají s viditelnými omezeními.

Nesmím zapomenout, že se lze setkat i se zařízeními hybridními, takové zařízení je například Pioneer DJ TORAIZ AS-1, kde syntetizér je implementován v analogovém obchodu, ale banka zvukových efektu je implementována na DSP procesorech pro jejich možnou variabilitu. [4]

2.3 Software

Doposud jsem mluvila pouze o vestavěných zařízení, je ale také se vhodné ohlédnout na dostupné softwarové vybavení k digitální syntéze zvuku na osobních počítačích a jak se jejich realizace syntézy a modulace liší od zařízení vestavěných. Jako příklad jsem zde uvedla populární Digital Audio Workstation (dále jen DAW) Ableton Live od společnosti Ableton.

Obrázek 2.4: Ableton Live

Ableton v mnoha přistupuje k práci s hudebními nástroji stejně jako vestavěné zařízení. Skladbu rozdělí na jednotlivé stopy. Každá stopa se skládá z jednoho zdroje audio signálu, tedy syntetizátoru, sampleru či externího vstupu a sérii zvukových efektů realizující modulaci signálu. Jedna stopa je tedy pipeline, Ableton umožňuje ve stopě mít libovolné množství zvukových efektů. [1] Ve světě vestavěných zařízení si lze představit jeden hudební nástroj jako byla uvedena například Volca Bass zapojenou do řady kytarových efektů.

Díky pluginovým standartům jako je například VST od společnosti Steinberg je možno vytvořit hudební syntetizér či modulátor a využít ho ve všech DAW programech, které standard VST podporují. [17]

Z důvodu výkonu dnešních běžných osobních počítačů lze v Abletonu realizovat zapojení, které by zabraly plochu většího studia pokud by zapojení bylo sestaveno ze zařízení vestavěných. Z těchto důvodů si dovolím považovat stav Abletonu v digitální syntézi a modulaci jako ideální. Ačkoliv je jednoduchá syntéza jako byla popsána např u Volca Bass dostačující, implementace následné zvukové modulace a její volnosti je problém u vestavěných zařízení a to nejenom analogových, ale také současných digitálních zařízení.

Mikroprocesory vestavěných zařízení nezvládnou realizovat větší množství zvukových efektů tak jako osobní počítače. Možné řešení tohoto problémů uvedu v kapitole 4.

Kapitola 3

Metody zpracování audio signálu

V této kapitole se nachází rozepsané prvky a metody zvukových syntetizéru a modulátorů které byly doposud uvedeny. Ačkoliv metod na zpracování signálu existuje velké množství, tento text se zaměřuje obzvlášť na ty jednodušší prvky a metody, které budou dále použity v návrhu a jsou pro něj vhodné.

3.1 Syntéza zvuku

Zatím byly zmíněny jednoduché generátory signálu ve slyšitelném rozsahu. Jak již bylo zmíněno, tyto jednoduché generátory lze vidět převážně u analogových zařízení. Ve světě zařízeních digitálních máme větší možnost výběru, pro tuto diplomovou práci je vybrána wavetable či look-up table syntéza.

Wavetable syntéza spočívá ve vyhledávání vzorků z tabulky samplu, ve které je uložena jedna perioda požadovaného signálu. Vyhledaný vzorek se zapíše na výstup a čtecí ukazatel v tabulce se posune. [8] Rychlost posouvání pochopitelně ovlivňuje periodu, pokud v tabulce je uložena perioda jednoduchého signálu jako obdélník, sinus, pila, poté se přímou mírou ovlivňuje produkovaný ton na výstupu. Zvýší se li rychlost posunu, zvýší se tón produkce a naopak. Tímto jednoduché signály dokáží pokrýt velkou část hudebního rozsahu, záleží ovšem na zvolené kvalitě záznamu.

Byla by chyba opomenout samplery, jelikož v tabulce může být uložen i jiný záznam než záznam jednoduchého signálu, může wavetable syntéza sloužit jako sampler. V následujících odstavcích proto chápejme význam slov wavetable syntéza, look-up table syntéza a sampler jako stejný. Bude-li tabulek více a bude-li algoritmus mezi nimi spolehlivě přepínat, je možné pak vytvořit knihovní nástroje jako jsou například bicí.

Je možné taky wavetable algoritmus skládat dohromady a umožnit tak stavbu replik komplexnějších hudebních nástrojů jako klavír. [18]

3.2 Modulace zvuku

Modulaci zvuku rozumíme, který změní charakteristiku existující zvukové stopy v závislosti na řídících vstupních parametrech.

Zvukové modulátory neboli zvukové efekty můžeme rozdělit na několik typů v závislosti o jaký typ modulace se jedná. Následuje přehled nejčastějších základních prvků.

- Frekvenční filtrace
  - Dolní propusť neboli Lowpass
  - Pásmová propusť neboli Bandpass
  - Horní propusť neboli Highpass
  - Ekvalizér neboli EQ. Jedná se o skupinu paralelně zapojených pásmových propustí.
- Časová modulace
  - Zpoždění neboli Delay
  - Flanger. Jedná se o zdvojení signálu a zpoždění jedné z jejich stop.
  - Chorus
  - Reverbace neboli Reverb. Jedná se o simulaci ozvěny ve halových prostorech, kde se zvuk o uzavřený prostor láme a je částečně pohlcen.
- Fázová modulace
  - Phaser. Jedná se o zdvojení signálu a fázový posun jedné z jejich stop.
- Waveshaping
  - Zkreslení neboli Distortion.

3.3 Frekvenční filtrace

Frekvenční filtrace je stavební blok přítomný v jisté míře téměř ve všech zařízeních realizující zvukovou syntézi a modulaci. Pomocí filtrace dokážeme signály vyhladit a zbavit se jejich artefaktů, dokážeme potlačit zbytečné či překážející frekvenční pásma.

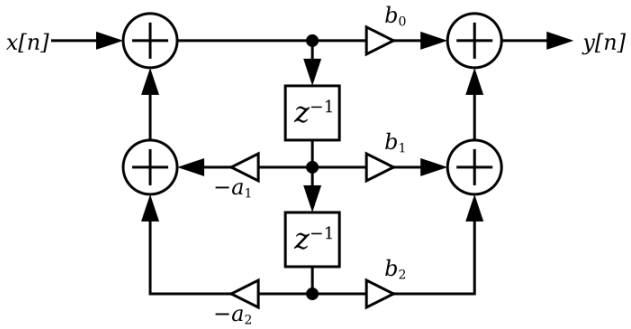
Vlastnosti filtru jsou velmi zajímavé a dokonce vlastnost postupného útlumu je pro nás v mnoha případech přímo žádaná, nižší útlum působí na poslech mnohem přirozeněji než útlum silný. Další zajímavou vlastností je jejich zpětná odezva, tedy "zvlnění" frekvenční charakteristiky v jejich pracovním bodu, zpětná odezva na poslech pomáhá vytvářet velmi charakteristický zvuk, který může být pro umělce žádaný.

Frekvenční filtrace je ale potřebná v mnoha dalších případech zpracování audio signálů. Například při převodu digitálního signálu na analogový výstup [16], nebo také potlačení zkreslení signálu při zesílení během jejich reprodukce.[15]

Pro naše potřeby budeme potřebovat typ filtru, který lze spolehlivě využít jako dolní propusť, pásmovou propusť i horní propusť. Takové frekvenční filtrace lze realizovat filtrama s konečnou impulzní odezvou (dále jen FIR) či nekonečnou (dále jen IIR). Filtry s konečnou impulzní odezvou jsou pro návrh jednodušší, ovšem výpočetně to již nemusí být pravda. Klasicky se lze setkat, že pro stejný výsledek je potřeba FIR filtr mnohem většího řádu než IIR filtr. Tento problém se objevuje hlavně v neparalelizovaných procesorech, například u FPGA se dá celý filtr paralelizovat a výpočet provést mnohem snadněji. Protože se tato práce věnuje jak audio modulace v software tak hardware, rozhodla jsem se zaměřit na IIR filtr, který je možno spolehlivě uskutečnit v jakémkoliv digitálním systému.

Takový populární filtr vhodný pro zpracování audio signálu je například digitální Biquad filtr, což je IIR filtr druhého řádu. [7]

Na obrázku 3.1 je zobrazeno zapojení filtru Biquad v jeho druhé přímé formě. Přenosová funkce je definována jako $H(z) = \frac{a_0+a_1z^{-1}+a_2z^{-2}}{b_0+b_1z^{-1}+b_2z^{-2}}$, koeficienty jsou normalizovány tak aby $b_0 = 1$.



Obrázek 3.1: Biquad filtr

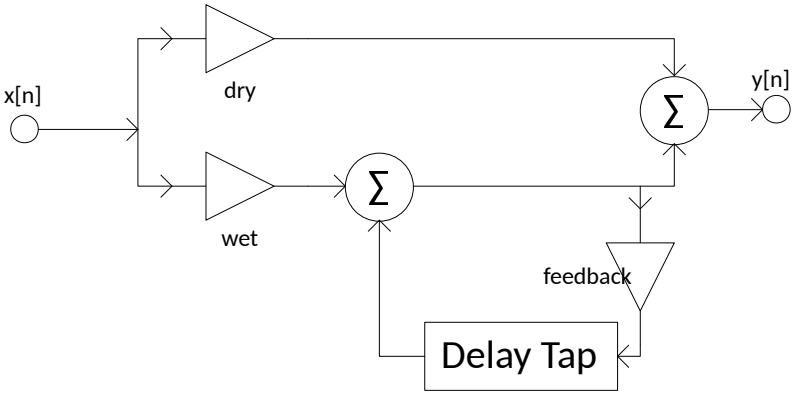
Útlum signálu filtru Biquad je 6 dB na oktávu, v praxi se běžně používají filtry s tlumením 12 dB či 24 dB na oktávu, ovšem můžeme se setkat i s jinými variantami jako například 18 dB na oktávu. Tento výsledek lze dosáhnout jednoduchým sériovým zapojením několika filtru Biquad do sebe. Dle zapojení na obrázku 3.1 vidíme, že výpočet výstupního signálu se jeví jako záležitost triviální. Naopak problémem je výpočet nových koeficientů pokud je potřeba změnit pracovní bod filtru či velikost jeho zpětné vazby.

3.4 Časová modulace a zpoždění

Základní stavební prvek pro časovou modulaci signálu je zpožďovací linka. Jednoduchou implementaci takové zpožďovací linky si můžeme představit jako kruhový buffer. Velikost takového kruhového bufferu vypočteme jako $n = \frac{l}{1000} * sampleRate$ kde l je požadovaná délka zpoždění v milisekundách a $sampleRate$ je vzorkovací frekvence se kterou zařízení pracuje.

Zpožďovací linka pracuje následovně. Příchozí vzorek ke zpracování přečte a sečte se vzorkem ze zpožďovací linky a následně zapíše na výstup. Pokud by tento výsledek byl zapsán také do kruhového bufferu, pak dojde k nekonečně cyklickému zvuku. Tento problém je vyřešen zesilovačem, který signál před zápisem do kruhového bufferu utlumí. Nakonec se posune čtecí a zapisovací ukazatel do kruhového bufferu.

Necháme-li zpožďovací linku zpracovat impulz, zjistíme že zpožďovací linka impulz bude periodicky opakovat s čím dál menší amplitudou až dojde k úplnému utlumení signálu. [7]



Obrázek 3.2: Delay

3.5 Funkce Wet / Dry zvukových efektů

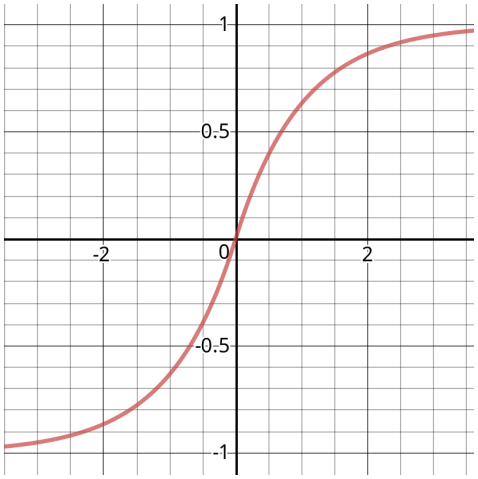
Často přítomná vlastnost implementace řady zvukových efektů je taky možnost změnit "přítomnost" zvukového efektu do výstupního signálu. Tato vlastnost je nevhodná u modulací jako je frekvenční filtrace, ale velmi užitečná například v případě časové modulace. Jestliže chceme např. simulovat efekt ozvěny v údolí pak je vhodné vstupní signál rozdělit na půl, polovinu signálu zapojit do zpožďovací linky a druhou polovinu přímo bez další modulace. Nakonec tyto dva signály sečteme a získáme tak zvuk, který je přítomný okamžitě tak i zpožděně, podobně jako skutečná ozvěna. Generickou realizace Wet/Dry vlastnosti můžeme vidět u blokového schématu zpožďovací linky na obrázku 3.2.

3.6 Waveshaping, digitální realizace distorze signálu

Filtry realizujicí waveshaping jsou nelineární filtry tvarující vstupní signál dle předpise zadané funkce funkce. [20] Waveshaping lze použít pro implementaci zkreslení vstupního signálu za účelem uvedení vyšších harmonických frekvencí. Oblíbená analogová varianta tohoto filtru je například jednoduché diodové zkreslení. Předpis takové funkce si můžeme představit jako.

$$f(x) = \begin{cases} 1 - e^{-x} : x > 0 \\ -1 + e^{x} : x < 0 \end{cases}$$

Na obrázku 3.3 lze vidět průběh této funkce.



Obrázek 3.3: Diodové zkreslení

3.7 Sequencer

Sequencer v audio syntetizéru slouží k naprogramování automatického spouštění not, efektů a jejich automatizaci. Jako příklad uvedu sekvencer v nástroji Volca Bass. Tento sekvencer implementuje běžný hudební čas $\frac{4}{4}$ na tempech od 56 úderů za minutu až 240 úderů za minutu, neboli Beats Per Minute (dále jen BPM). Volca Bass může sekvencer nastavit v modu 8 či 16 dob, na každou dobu lze naprogramovat notu a její modifikace. Ty mohou být pro každý vestavěný sekvencer rozdílné, Volca Bass například implementuje funkci note slide. Pokud nás zajímá standard, který je přítomen ve všech zařízení je dobré se podívat na sběrnici MIDI, která je rozepsána v podkapitole 3.8.

Sekvencer Volcy Bass má velmi omezenou paměť, umožňuje uložit pouze 8 patternu. Se srovnatelnou velikostí paměti se lze setkat u většiny sekvencerů v analogových zařízeních. Za druhý extrém lze považovat sekvencery v počítačích, které mají paměť téměř neomezenou. [11]

Sekvencery velmi často umožňuje využít rozhraní MIDI, které je dále rozvedeno v sekci 3.8, a ovládat tak řadu dalších zařízení. [19]

Sekvencerům se nadále v této práci již prostor věnovat nebude. Důvody jsou uvedeny dále v kapitole 4.

3.8 Specifikace MIDI

Midi je seriová asynchronní sběrnice navržena na přenosu zpráv mezi audio zařízeními. Dlouholetým používáním se osvědčila, v roce 1983 byla standardizovaná a je dnes udržována MIDI Manufacturers Association. Proto se s ní v různé podobě setkáme u většiny elektronicky hudebních zařízeních. Zprávy jsou velké 8 bitů a jsou přenášeny přenosovou rychlostí

$31.25\,\mathrm{kbit\,s^{-1}}$. Jedna sběrnice je rozdělena na 16 kanálů. tedy na jedné sběrnici spolu může komunikovat až 16 zařízení. Následuje krátký výtah typu zpráv tohoto standardu. [6]

- Channel Voice - Obsahuje hlavně zprávy ohledně not a důrazu, změny parametru např. hlasitost, atributy filtru, obálek aj.
- Channel Mode - Obsahuje řídící zprávy pro všechny nástroje poslouchající na daném kanálu např. Vypnutí zvuku, vyresetování parametru.
- System Common - Mimo jiné obsahuje velmi důležitou zprávu System Exclusive. System Exclusive zpráva začíná hlavičkou identifikátoru zařízení výrobce, kterému je přiřazeno podle mezinárodního registru, nutno podotknout, že se jedná o placenou službu. Výhoda spočívá v možnosti zajištění odklonění zpráv pro své vlastní zařízení od jakékoliv jiné zařízení, jelikož tyto zařízení nebudou takové zprávy ani číst a počkají na zprávu oznamující ukončení system exclusive sekce.
- System Real-Time - Obsahuje důležité zprávy oznamující tempo, signál startu, pozastavení, ukončení hudební produkce.

Zprávy mají většinou dva byty, první byte signalizuje typ zprávy a midi kanál, druhý byte obsah zprávy. Některé typy zpráv povolují i jiné velikosti, například System Exclusive může mít až 4 byty + délka exkluzivní zprávy + 1 ukončující byte.

Midi zařízení mohou pracovat buď v modu Master či v modu Slave. Master zařízení je v zapojení pouze jedno, Master zařízení posílá signál Midi Clock všem svým Slave zařízením. Master zařízení je většinou povětšinou sekvencer. Sekvencer určuje tempo, tedy BPM a na této závislosti generuje signál Midi Clock. Sekvencer může být řešen dedikovaným HW zařízením či Počítačem, Laptopem, Tabletem. Pro použíti spojení počítač - vestavěné zařízení se mnohdy využívá USB to Midi převodník.

Kapitola 4

Implementace řešení

Tato práce doposud řešila existující nástroje, rozebrala jejich vlastnosti a metody zpracování signálu, uvedla jejich slabé a silné články. V této kapitole se všechny tyto získané znalosti použijí a za účelem navrhnutí možné realizace na programovatelných hradlových polích.

Před návrhem možného řešení je potřeba zmínit, že doposud je v této práci zanedbána velmi důležitá část všech těchto zařízení a to jsou ovládací a zobrazovací prvky. Ačkoliv je tato problematika téměř nepostradatelná pro běžné používání, táto práce si neklade za cíl navrhnout ideální ovládací prvky. Proto tahle problematiku budu z velké části opomenuta a návrh zařízení lze chápat jako zařízení čistě simulační. Části návrhu, které by fungovaly jako ovládací prvky budou popsány pouze bez větších detailů se zmíněním možného rozšíření.

Jedna z možností je silně se inspirovat analogovými zařízeními. Implementovat digitální repliku analogového zařízení je možná cesta. Některé vlastnosti syntézy a modulace zvuku se sice ztratí, například se mohou očekávat problémy s aliasingem, ale je to možná jak FPGA využít. Hlavní problém, který v tomto přístupu vidím je, že tato cesta by obsahovala i všechny nevýhody analogových zařízení a to hlavně v ohledu na jejich statický charakter. Tam kde digitální syntetizéry a modulátory alokovaly DSP procesory podle uživatelského nastavení a dokázaly tak adaptivně využít své zdroje, analogové zařízení pokulhávají.

Jeden z návrhu řešení bylo implementovat prvky syntézy a modulace nezávisle na sobě a pak umožnit propojovací sítí jejich libovolné zapojení. Tento přístup realizován nebyl a to hlavně z následujícího důvodu. Každý prvek by byl implementován pouze ve stanovený počet, například implementace efektu filtru, zpoždění a zkreslení by byly v zapojeni přítomny v jediné své instanci. Uživatel ale nemůže některou z komponent využít několikrát a zároveň v jeho zařízení se nachází nevyužité komponenty. Jelikož s jistotou nelze předpovídat jaké prvky požaduje uživatel je velmi těžké najít takové rozložení komponent v síti, které by uspokojilo všechny uživatele a zároveň by nespotřebovaly příliš velké množství zdrojů.

Nelze si nepovšimnout, že přístup procesorových digitálních syntetizérů a modulátorů je v mnoha ohledech velmi vhodný. Mikroprocesory počítají jenom to co je potřeba, naráží se spíše na problémy škálovatelnosti. Mikroprocesor zvládne toho vypočítat v realném čase vypočítat pouze omezené množství. Procesory s vice jádrovou technologií či podporou Single Instruction Multiple Data (SIMD) instrukční architekturou jsou možným řešením tohoto problémů. [5]

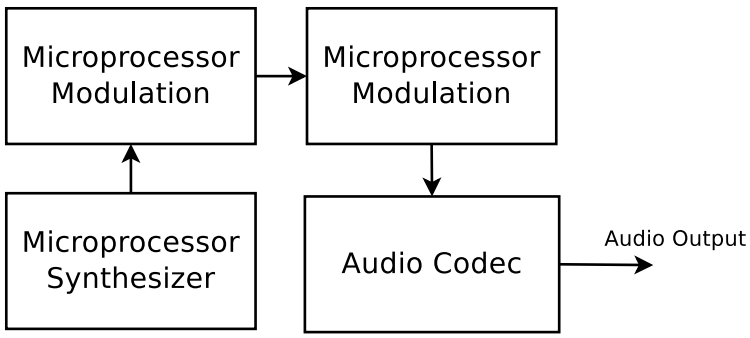
Přitom pro digitální zpracování signálu není potřeba plnohodnotný silných mikroprocesorů, ve své podstatě se bavíme o sériovou pipeline jednoduchých výpočetních jednotek, hlavní problémem je jejich dostatečné množství a adekvátní výkon.

Uvážíme-li implementaci malých, omezených sériově zapojených mikroprocesoru na FPGA. Takový čip by pro práci s audio signálem měl umět základní i pokročilé matematické operace jako například goniometrické funkce, měl by umět pracovat s pamětí v registrech a přístup do RAM paměti a také by měl umět základní řídící operace.

Myšlenka syntézy podobně omezených mikroprocesorů k využití jako Digital Signal Processing (DSP) již existuje. Dobrá vlastnost takových mikroprocesorů může být při adekvátním návrhu nižší spotřeba než běžné DSP procesory. [14]

Mikroprocesor by fungoval následovně. Na základně externího podnětu by přijal data a vygeneroval na svém výstupu jeden sample. Pokud mikroprocesor generuje samply s nenulovými hodnoty při přijmutí nulových vzorků jedná se o syntetizér, v opačném případě se jedná o modulátor. Nyní mikroprocesory uspořádejme do sériového zapojení. První mikroprocesor přijímá externí vstupy, tedy požadavky o syntézy noty/samplu, ty ostatní čekají na svém vstupu signál ke zpracování. Na výstupu posledního mikroprocesoru se zákonitě objeví digitalizovaný audio signál připravený k převedení do analogového audio signálu.

Díky možnosti tvorby serio-paralelních zapojení na FPGA by taková pipeline mikroprocesoru pracovala paralelně a celkový čas ke zpracování vstupního pulzu až k prvnímu sample reflektující tento pulz je sečtením zpoždění všech mikroprocesorů.

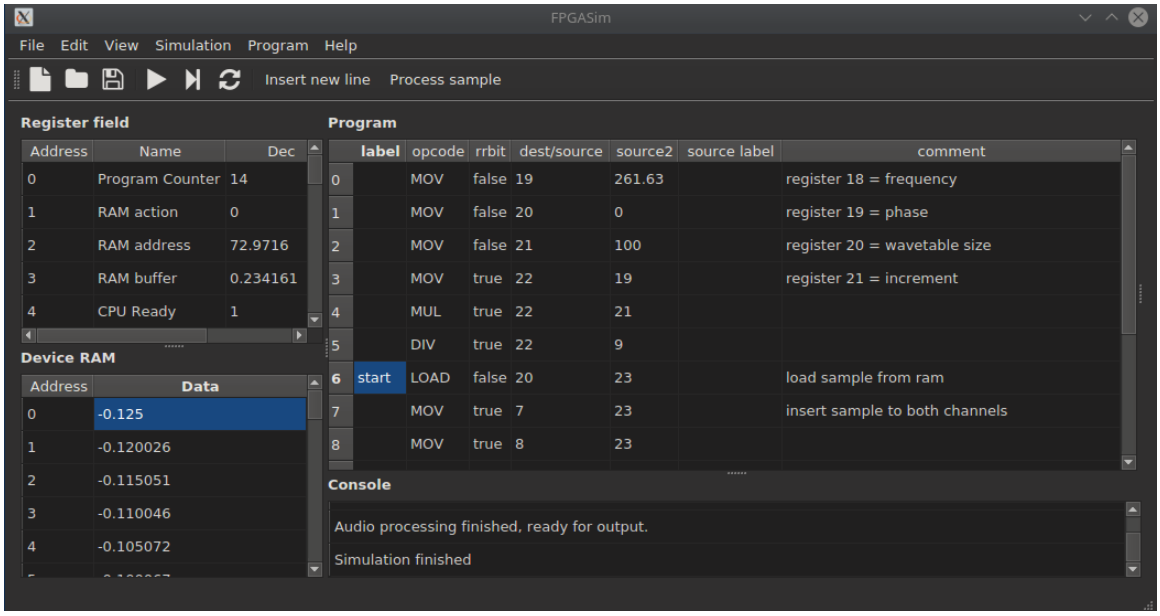


Obrázek 4.1: Blokové schema návrhu zařízení

V sekci 4.1 je rozebrán simulátor zvýše zmíněného mikroprocesoru společně s experimenty provedenými na simulátoru. Sekce 4.2 popisuje vývojové prostředí na kterém je implementace realizována a nakonec sekce 4.3 až 4.7 popisuje implementaci mikroprocesoru samotného a audio pipeline.

4.1 Simulátor mikroprocesoru

Před samotným návrhem mikroprocesoru je potřeba vědět zda-li výše popsané metody zpracování audio signálu jdou vůbec na jednoduchých mikroprocesorech implementovat a pokud ano, tak jaké vlastnosti by takové mikroprocesory měly mít. Pro řešení tohoto problému jsem navrhla desktopovou aplikaci, která abstrahuje chování tohoto mikroprocesoru.



Obrázek 4.2: Simulátor

Na obrázku 4.2 je zobrazeno uživatelské grafické rozhraní simulátoru. Uživatel zapisuje instrukce do tabulky, v panelu na levé straně lze vidět obsahy registrů a přiděleného obsahu RAM paměti, v panelu vpravo, který není na obrázku zobrazen ovládá uživatel simulaci.

Simulátor implementuje instrukční sadu, která by byla podporována navrženým procesorem. Pro zjednodušení datový formát dat je 32 bit float, na rozdíl od FPGA, kde je využita fixed point 16.16 aritmetika. V simulátoru si jsou všechny registry ve svých vlastnostech rovny a lze k nim libovolně přistupovat. Simulátor taky obsahuje paměťový řadič a velmi jednoduše řešen zápis a čtení ze simulované RAM paměti a to bez jakéhokoliv postihu.

Simulátor je schopen načíst WAV soubor a při správné implementaci programu v instrukčním jazyce je možno soubor zpracovat a zapsat na výstup. Při WAV souboru s nenulovým vstupem simulátor tedy může realizovat audio modulaci, v případě kdy vstupní WAV soubor bude obsahovat ticho, simulátor může pracovat jako syntetizér. Simulátor umožňuje přehrání vstupního a výstupního WAV souboru, tedy okamžitou odezvu jak transformace signálu proběhla.

Je zřejmé, že simulátor plně nereflektuje FPGA soft mikroprocesor, ale pro potřeby dokázání uskutečnitelnosti implementace a hrubé stanovení designu je tento simulátor dostačující.

Pomocí simulátoru bylo realizována řada experimentů. Implementace metod zpracování audio signálu zmíněné v kapitole 3. Během implementace metod byla upravována instrukční sada a také modifikován návrh mikroprocesoru. Dosavadní instrukční sadu lze vidět v příloze A. V následujících sekcích je také rozepsáno jak se implementace jednotlivých instrukcí liší na straně HW od simulátoru.

Simulátor také obsahuje překladač na bytecode kterým je možné naprogramovat FPGA mikroprocesor. Bytecode lze vložit do zdrojového kódu řídícího programu, kterým se pak naprogramuje instrukční paměť soft mikroprocesoru.

Z řadu experimentů jsou rozebrány tři podrobněji, které se dají považovat za stěžejní a to je simulace wavetable syntézy, simulace zpoždění a simulace digitálního biquad filtru.

Zároveň ať už v textu simulace či v příloze je uveden i instrukční kód experimentu. Tento instrukční kód je velmi podobný jazyku symbolických instrukcí NASM, instrukce jsou uvedeny v odsazeném zápise v následujícím formátu:

Instrukční kod 4.1: Ukázka zápisu

```
<navesti>: <instrukce>  <cil> <rrbit> <zdroj> <skok> //komentar

```

Návěští funguje obvykle jako v jiných jazycích, pokud u nějaké instrukce je uvedeno návěští na místě skoku, pak tato instrukce indikuje skok na dané návěští za splnění podmínky.

Instrukce prezentuje operaci k vykonání v příloze A lze nalézt kompletní seznám instrukcí s jejich podrobným popisem. Cíl označuje adresu registru, kam se uloží výsledek operace. Rrbit je povinný bit ke každé instrukce, který v hodnotě 1 označuje instrukci, která načítá zdrojovou hodnotu z registru a v hodnotě 0 označuje instrukci jejichž zdrojový operand je konstanta. Proto tedy zdroj může být adresa zdrojového registru či zdrojová konstanta.

4.1.1 Simulace wavetable syntézy

Simulace wavetable syntézy ověřuje schopnosti mikroprocesoru syntetizovat celou škálu zvuků pokud vezmeme na vědomí, že v tabulce základního vzorku může být uloženo téměř vše od jedné periody jednoduchého signálu až po nahrávky úderu bicího nástroje.

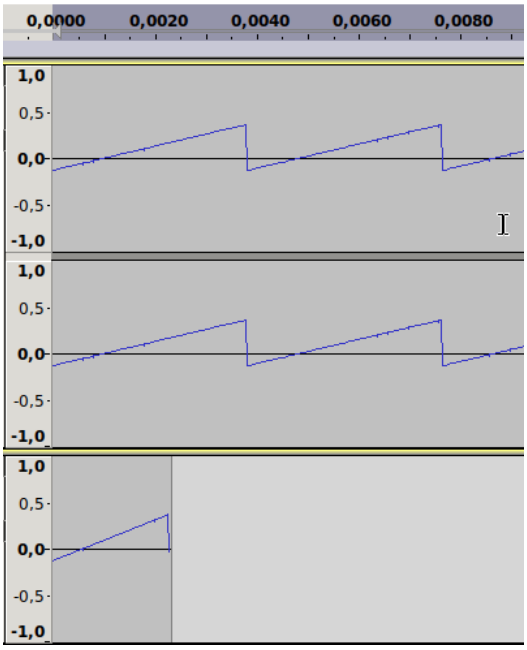
Tento experiment vyústil v krátký a jednoduchý kód. Program předpokládá, že v RAM paměti na adrese 0 až *wavetablesize* je uložena tabulka vzorků. Algoritmus podé podle požadované frekvence na výstup zapisuje vzorky z tabulky a posouvá svou pozici v tabulce o inkrement, pro který platí $inkrement = f * \frac{ws}{s}$, kde f značí požadovanou frekvenci tonu, ws velikost wavetable a s vzorkovací frekvenci. Nutno podotknout že se jedná o zapojení jediného generátoru, tedy monosynth.

Instrukční kod 4.2: Wavetable syntéza

```
        MOV     19      0       261.63  //register 19 = frequency
        MOV     20      0       0       //register 20 = phase
        MOV     21      0       100     //register 21 = wavetable size
        MOV     22      1       19      //register 22 = increment
        MUL     22      1       21
        DIV     22      1       9
start:  LOAD    23      1       20      //load sample from ram
        MOV     7       1       23      //insert sample to both channels
        MOV     8       1       23
        ADD     20      1       22
        CMP     20      1       21
        JL      0       0       0       nOver
        MOV     20      0       0
nOver:  FIN     4       0       1       //finish sample
        JMP     0       0       0       start
```

Tento program má několik limitací, jedna z nich je potřeba dostatečně velké tabulky a správně zformované tabulky, které musí obsahovat jednu periodu nedisharmonického

signálu. Program generuje uspokojivé výsledky syntézy jednoduchých tonu v rozmezí třetí až pátek oktávy na tabulkách klasických základních signálu jako je sinus, pila, obdélník či trojúhelník. Tabulky musí mít velikost alespoň 100 vzorků, čím větší a jemnější tabulka tím je možné generovat kvalitnější výsledky.



Obrázek 4.3: Ukázka syntézy pilového tonu frekvence 440 Hz (nahoře) Původní perioda o délce 100 vzorků (dole)

Na obrázku 4.3 je vidět wavetable tabulka, která byla použita k syntézi tonu pily o 440 Hz.

Pokud je potřeba využít syntetizér spíše jako sampler potřebujeme pro to pouze upravit inkrement na konstantu 1. Tedy za předpokladu, že sample má stejnou vzorkovací frekvenci jako je pracovní vzorkovací frekvence celého zařízení. V opačném případě by bylo nutno implementovat algoritmus který by tabulku převzorkoval, například s využitím interpolace.

Inkrement, ale může sloužit také jako modifikátor časové dimenze. Pokud se ve wavetable tabulce nachází záznam, je možné inkrement upravit v poměru tempa záznamu vůči aktuálnímu tempu a záznam tak zrychlit či naopak zpomalit, aby jak záznam tak syntéza probíhaly se stejnou periodou.

4.1.2 Simulace zpožďovací linky

Zpožďovací linka byla popsána v 3.4, cílem tohoto experimentu je zpožďovací linku implementovat v simulátoru.

Simulace zpožďovací linky je implementována jako cyklická fronta. Z důvodu potřeby větší velikosti cyklické fronty je nutné ji umístit do paměti RAM. To ale taky znamená, že cela implementace je velmi náročná k přístupu do paměti, jelikož každý takt potřebuje provést jedno čtení a zápis.

Jedná se o to složitější situaci při uvědomění, že mikroprocesor musí zpracovat dva kanály současně a také díky faktu, že komponenta zpožďovací linky je velmi populární

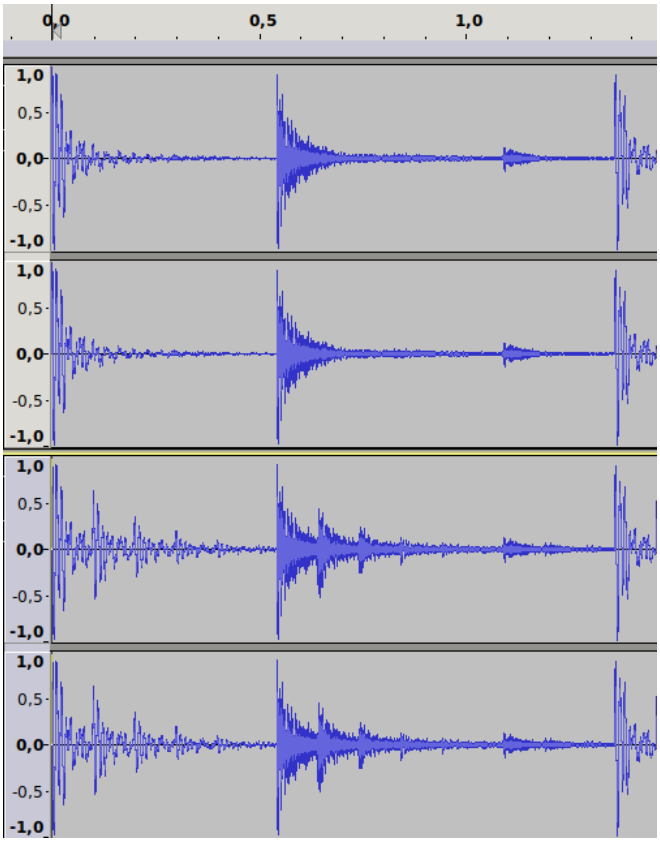
v mnoha audio zapojeních. Protože navíc je každá cyklická fronta uložena na jiném místě v paměti, jeví se pak přístup do paměti jako náhodný. Také je potřeba myslet na limitace adresovatelné paměti v mikroprocesoru ta se podrobněji rozebírá v sekci 4.5.

Ve instrukčním kódu 4.3 je uveden celý kód této simulace. Simulace vytvoří cyklickou frontu s fixním zpoždění 1000 ms pro každý kanál. Je zřejmé, že implementace na rozdíl od systémových požadavků je vesměs triviální, program pro jednu zpožďovací linku má včetně inicializace kolem 40 instrukcí.

Instrukční kod 4.3: Zpožďovací linka

```
1         MOV     19      0       100     //register 19 = Length of delay in ms
2         DIV     19      0       1000
3         MUL     19      1       9       //Convert ms to sample count
4         MOV     10      0       0.5     //Initial feedback (Control change 0)
    range (0,1)
5         MOV     20      0       0       //register20 = Read Pointer L = 0
6         MOV     21      1       19      //register21 = Write pointer L = sample
    count - 1
7         SUB     21      0       1
8         MOV     22      1       21      //register 22 count - 1
9         MOV     23      1       20      //register 23 = Read Pointer R
10        ADD     23      1       19
11        MOV     24      1       21      //register 24 = Write Pointer R
12        ADD     24      1       19
13  start:  LOAD    25      1       20      //Read L
14        MOV     26      1       5
15        ADD     26      1       25
16        MOV     7       1       26
17        MUL     26      1       10
18        STORE   23      1       26      //Write L
19        LOAD    27      1       23      //Read R
20        MOV     28      1       6
21        ADD     28      1       27
22        MOV     8       1       28
23        MUL     28      1       10
24        STORE   24      1       28      //Write R
25        ADD     20      0       1       //increment read pointer L
26        ADD     23      0       1       //increment read pointer R
27        CMP     20      1       22
28        JL      0       0       0       readNotOverL
29        MOV     20      0       0
30        MOV     23      1       19
31  readNotOverL:   ADD     21      0       1       //increment write pointer
32        ADD     24      0       1
33        CMP     21      1       22
34        JL      0       0       0       writeNotOverL
35        MOV     24      1       19
36  writeNotOverL:  FIN     0       0       0
37        JMP     0       0       0       start
```

Na obrázku 4.4 je zobrazeno srovnání vstupního signálu s výstupním po zpracování v tomto experimentu. V dolní části je patrné vidět jednotlivé zpožděné a tlumené impulsy.



Obrázek 4.4: Ukázka fixního zpoždění o délce 1000 ms s 50 % zpětnou vazbou. Nahoře původní signál, dole signál po zpracování.

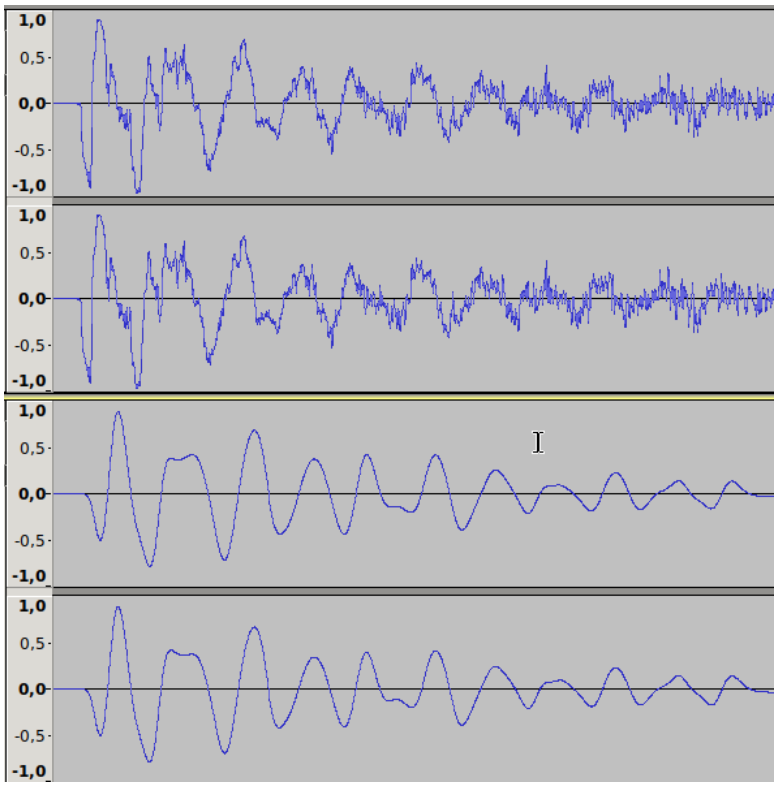
V praxi často uvažujeme délku odezvy proměnlivou a s ní potřebu přepočítat velikost cyklické fronty. To ale taky znamená, že tento algoritmus je velmi náchylný na artefakty. Tento problém by se dál vyřešit jednoduchým filtrem, který by si pamatoval několik posledních výstupních vzorků v cyklické fronty bez závislosti na její aktuální pozici. Tyto vzorky by se váhově sečetly s pravidlem, že nejstarší vzorek by měl nejmenší váhu. Tento interpolační filtr by fungoval jako dolní propusť, ovšem jistě by zamezil případným artefaktům. Mohl by být realizován jak přímo v instrukčním programu s využitím registrů, tak také jako dedikována jednotka součástí soft mikroprocesoru.

4.1.3 Simulace digitálního filtru

Jako další důležitý experiment je uvedena implementace filtrace zvuku. Byly využity IIR filtry s nekonečnou impulzní odezvou, které jsem popsala v 3.3.

Během implementace Biquad filtru se ukázala nutná potřeba využití matematických goniometrických funkcí. Tyto funkce jsou v simulátoru implementovány nativně, v případě soft mikroprocesoru je funkce vypočtena se znatelným zpožděním přeposláním na ARM mikroprocesor. Funkce jsou v implementace přítomny z důvodu výpočtu koeficientu filtru, které jsou potřeba přepočítat při každé změně jednoho z řídících parametrů. Modifikace řídících parametrů filtru je často používána vlastnost, dokonce mnohdy automatizována [1]. Z tohoto důvodu musíme počítat s nejhorší možnou variantou a to nutnost přepočítat koeficienty při zpracování každého samplu.

V experimentu byla implementována druhá přímá forma, která je vhodná pro čísla s plovoucí desetinnou čárku, pro implementaci na soft mikroprocesor je lepší využít první přímou formu.

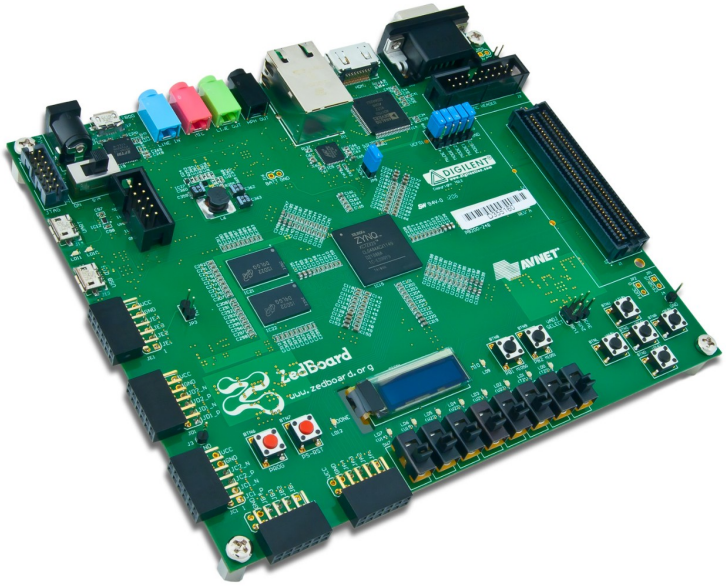


Obrázek 4.5: Ukázka filtrace dolní propusť filtrem biquad s tlumením 12 dB na oktávu. Nahoře původní signál, dole signál po filtraci.

Zdrojový kód pro tento experiment obsahuje přes 90 instrukcí, je příliš dlouhý a proto jej lze nalézt až v příloze B. Zde je již jasné, že implementace krátkých jednoduchých programů je sice možná, jejich následná kombinace může způsobit nemalé potíže. Tento problém by šel například vyřešit implementaci překladače z omezeného programovacího jazyka. Takový překladač ale již nespadá do oblasti této diplomové práce a jeho existence nebyla realizována.

4.2 Zedboard vývojový kit

Implementaci je realizována na vývojovém kitu Zedboard, který je postaven na architektuře Zynq 702 System on Chip. K dispozici má ARM A9 Dual core mikroprocesor, programovatelné pole Xilinx Artix-7, 512 MB DDR3 paměti a na kitu se nachází i stereo audio 24 bitový kodek ADAU1761. [3]



Obrázek 4.6: Zedboard

Na FPGA části vývojového kitu, tedy v programovací logice se nachází implementace soft mikroprocesor a audio pipeline. Na straně ARM mikroprocesoru běží řídící program, který při startu systému inicializuje instrukce FPGA mikroprocesorů a během běhu systému funguje jako matematický koprocesor realizující pokročilé matematické funkce, které by byly příliš drahé implementovat na straně FPGA. Stranu mikroprocesoru ARM může obsluhovat distribuce operační systému Linux či "bare metal" aplikace. Pro tuto aplikaci je mikroprocesor utilizován jako bare metal, v sekci 4.7 jsou uvedeny detaily implementace na mikroprocesoru ARM.

4.3 Soft mikroprocesor

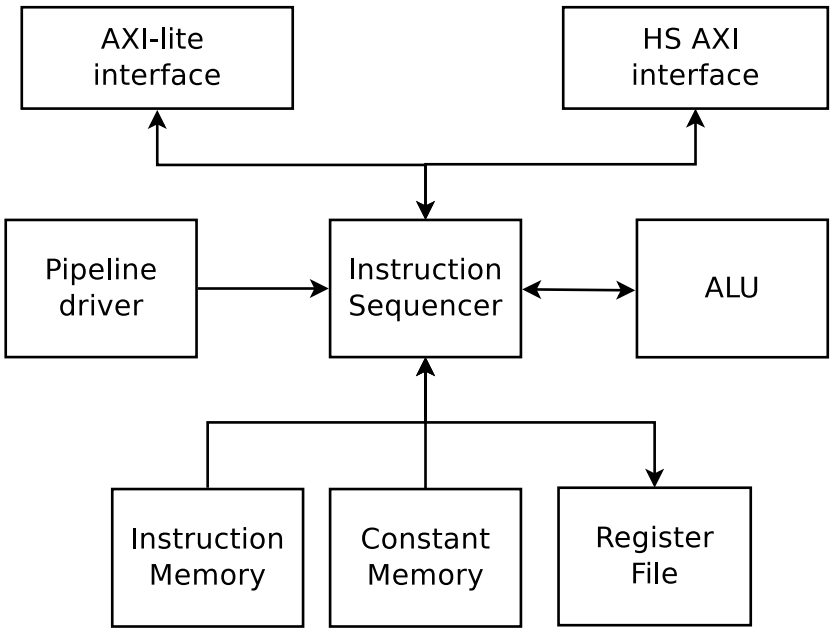
Soft mikroprocesor je tedy subskalární mikroprocesor implementující úzkou množinu operací v programovací logice. Soft mikroprocesor komunikuje na audio pipeline, zpracovává vstupní vzorky a přeposílá je dále. Na audio pipeline může být zapojen či více soft mikroprocesorů. Jednotlivé mikroprocesory jsou od sebe izolovány, nemohou si navzájem přistupovat do paměti, jediný způsob komunikace je předávání zvukových vzorků po směru pipeline. Mikroprocesory pracují v modu stereo, vždy přijmou a zpracují a odešlou vzorky levého a pravého kanálu společně. Mikroprocesor zpracovává vždy pouze jeden vzorek od levého a pravého kanálu. Mezi různými vzorky se neresetují registry mikroprocesoru ani se nijak jinak neupravují, po odeslání zpracovaných vzorků je mikroprocesor pouze uspán do doby, než příjme další vzorky ke zpracování.

Důležitý aspekt mikroprocesoru je formát dat se kterým pracuje. Audio codec pracuje ve fixed point aritmetice 24 bit ve formátu 1.23 tedy 1 bit signalizuje znaménko a dalších 23 část za desetinnou čárkou, audio codec tedy na vstupu očekává číslo v intervalu $< -1, 1 >$. Potom pokud soft mikroprocesor má pouze realizovat zpracování audio a chceme zamezit ztráty dat měl by být interní formát na straně soft mikroprocesoru taky 1.23. Bohužel tento formát se jeví více než problematický, jelikož při výpočtu v audio algoritmech pracujeme s čísly v mnohem větším rozsahu, čísla která vyjadřují počty milisekund či frekvence. Další

problém je předávání a práce s těmito čísly na strašně ARM mikroprocesoru. Po delším uvážení jsem se rozhodla jako datový formát využít 32 bit fixed point čísla ve formátu 16.16. Před zavedením na audio codec je zkrácena celočíselná strana na pouze znaménkový bit a bity 17-23 na straně desetinné čárky jsou nulovány.

Na obrázku 4.7 lze vidět prozatímní blokové schéma mikroprocesoru. Pro účely experimentů je počítáno pouze se statickou instrukční paměti o maximální velikosti programu několika stovek instrukcí. Mikroprocesor je implementován na omezení adresovatelné instrukční paměti 16 bitů.

Mikroprocesor je dimenzován tak aby na každý stupeň v pipeline tedy zvukovou syntézu či implementace zvukového efektu stačil jeden mikroprocesor. V případě experimentů je to dostačující jelikož se jedná o velmi jednoduché programy, které většinou již nejde dále rozdělit. Jistě tedy nejde o ideální rozdělení problémů tak aby jej mikroprocesory zpracovaly vyváženě.

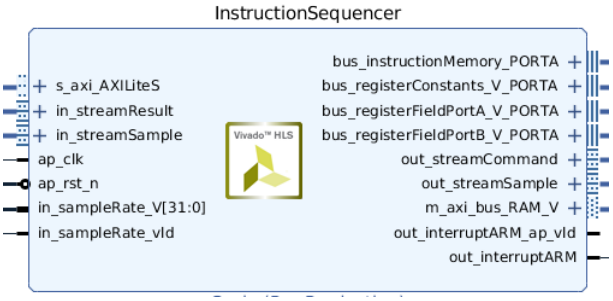


Obrázek 4.7: Blokové schéma mikroprocesoru

4.4 Instrukční sekvencer

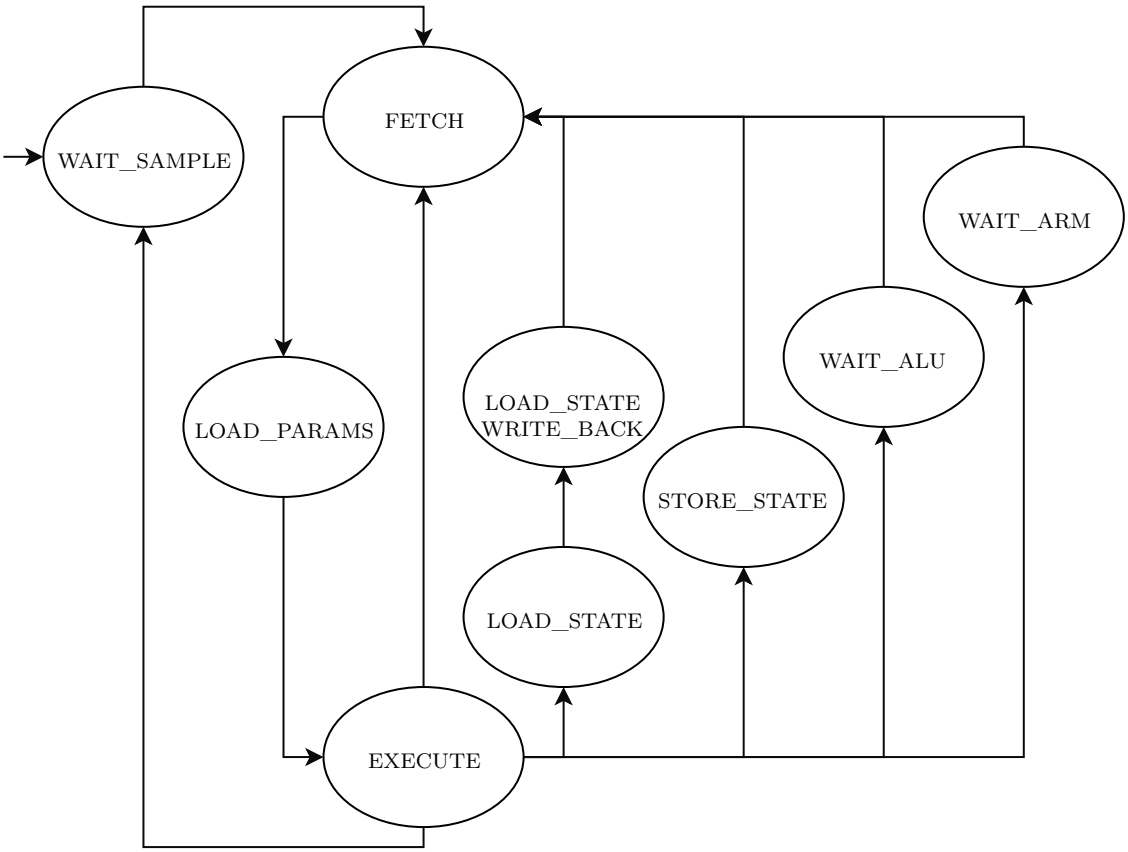
Instrukční sekvencer je hlavní řídící jednotka soft mikroprocesoru. Na obrázku 4.8 lze vudět vstupy a výstupy této komponenty. Instrukční sekvencer má přístup k BRAM instrukční paměti, poli BRAM registrů, komunikačních AXI-lite rozhraní s ARM mikroprocesorem své Aritmeticko Logické Jednotce a konečně napojení na audio pipeline.

Instrukční paměť je rozepsána v sekci 4.7. BRAM registry jsou podrobněji zmíněny v sekci 4.5. AXI-late registry jsou adresovatelné z mikroprocesoru ARM, jejich přehled lze najít v sekci 4.5, jejich chování spolu s výstupem přerušení pak v sekci 4.7.



Obrázek 4.8: Instrukční sekvencer

Přístup do RAM paměti je řešen na rozhraní AXI-master pomocí High Performance (dále jen HP) portů, které umožňují čtení i zápis přes FIFO rozhraní. Instrukční sekvencer je na rozhranní AXI-master v modu master. Zynq architektura umožňuje využít 4 separátní HP porty [13], přístup k RAM paměti je tedy umožněno 4 soft mikroprocesorům. Instrukční sekvencer je implementován jako konečný automat.



Obrázek 4.9: Stavový diagram instrukčního sekvenceru

Instrukční sekvencer začíná svou práci ve stavu `WAIT_SAMPLE`, v tomto stavu provádí blokové čtení na vstupní AXI4 Stream sběrnici na které očekává packet se vzorkem ke zpracování. Po přijetí tohoto packetu nahraje jeho obsah do vnitřních registrů a přechází do stavu `FETCH`.

Stav FETCH signalizuje nahrání instrukce z BRAM do vnitřního registru. Sekvencer vždy nahrává také konstantní pamět instrukce. po nahrání sekvencer vždy přechází do stavu LOAD_PARAMS.

Ve stavu LOAD_PARAMS instrukční sekvencer naplní vnitřní registry sekvenceru obsahem BRAM registrů z adres instrukce. Toto nahrávání probíhá vždy i když má instrukce hodnotu rr bitu nastavenou v 0. Poté sekvencer přechází do stavu EXECUTE.

Stav EXECUTE v závislosti na operačním kódu instrukce provádí potřebné činnosti k vykonání instrukce. Instrukce, které nepracují s BRAM, RAM pamětí ani nevyžadují vyvolání přerušení na mikroprocesoru ARM jsou v tomto stavu také dokončeny a sekvencer přechází zpět do stavu FETCH. Výjimka je instrukce FIN, která pozastavuje činnost sekvenceru přechodem zpět do stavu WAIT_SAMPLE. Tento stav je také jediný stav ve kterém se modifikuje obsah registru Program Counter.

Operandy a operační kód instrukcí vyžadující výpočet v Aritmeticko Logické Jednotce (dále jen ALU) se vystaví na výstupu sekvenceru a sekvencer přechází do stavu WAIT_ALU ve kterém čeká na výsledek z ALU, jakmile je výsledek k dispozici, zapisuje se jeho obsah do BRAM registrů a sekvencer přechází do stavu FETCH.

Náročnější matematické instrukce se na ALU neposílají a namísto toho je potřeba využít obsluhy mikroprocesoru ARM. U těchto instrukcí sekvencer nastaví AXI-lite registry určené k předání dat do mikroprocesoru ARM, vyvolá přerušení přepnutím signálu out_interruptARM na hodnotu 1 a přechází do stavu WAIT_ARM. Stav WAIT_ARM očekává na signalizaci mikroprocesoru ARM a po dokončení je výsledek zapsán do BRAM registru, následně sekvencer přechází do stavu FETCH.

Instrukce LOAD nahrává obsah RAM paměti do BRAM registru, proto jsou potřeba dva další čekací stavy. Ve stavu LOAD_STATE sekvencer přistupuje do RAM paměti a ve stavu WRITE_BACK zapisuje nahrané hodnoty do BRAM registru. Oproti tomu instrukce STORE již má hodnoty z BRAM registru k dispozici a pouze přistupuje do RAM paměti pro zápis. Sekvencer u obou instrukcí po jejich dokončení přechází do stavu FETCH.

4.5 Adresový prostor soft mikroprocesoru

V tabulce 4.1 lze vidět speciální adresy v poli registrů soft mikroprocesoru. K těmto registrům se převážně přistupuje přes operaci MOV. Je nutno podotknout, že pouze Axi-lite registry jsou adresovatelné z ARM mikroprocesoru. Adresy, které v tabulce nejsou uvedeny jsou adresy s běžným registrem bez žádných nadstandardních vlastností. Registry u kterých není uvedena ani jedna z možností čtení či zápis jsou registry určené pouze pro vnitřní použití a instrukce MOV na ně nemůže přistoupit.

Adresa	Popis	Čtení	Zápis	Axi-lite
0	Rezervováno			
1	Předání hodnoty ARM mikroprocesoru			✓
2	Indikace dokončení operace ARM mikroprocesoru			✓
3	Operace ARM mikroprocesoru			✓
4	Rezervováno			
5	Vstupní vzorek levého kanálu	✓		
6	Vstupní vzorek pravého kanálu	✓		
7	Výstupní vzorek levého kanálu		✓	
8	Výstupní vzorek pravého kanálu		✓	
9	Vzorkovací frekvence	✓		
10 - 17	Rezervováno pro řídící parametry	✓		
18	Výsledek porovnání	✓	✓	

Tabulka 4.1: Nadstandardní registry soft mikroprocesoru.

V tabulce 4.2 je vidět globalní adresový prostor na vývojové desce Zedboard. Připomeňme si, že Zedboard má adresovatelných 512 MB RAM paměti, AXI-lite registry, vyhrazené prostory pro soft mikroprocesory jsou mapovány v tomto prostoru.

Adresy u kterých je vlastník označen jako Soft Mikroprocesor jsou adresy prvního instrukčního sekvenceru v audio pipeline. Těchto mikroprocesorů v aplikace může být větší počet a všechny budou potřebovat adresový prostor stejné velikosti.

Popis	Velikost	Vlastník
Předání hodnoty ARM mikroprocesoru	32 bit	Soft mikroprocesor
Indikace dokončení operace ARM mikroprocesoru	32 bit	Soft mikroprocesor
Operace ARM mikroprocesoru	32 bit	Soft mikroprocesor
RAM	16 kB	Soft mikroprocesor
Povolení audio pipeline	32 bit	Pipeline řadič
Vzorkovací frekvence	32 bit	Pipeline řadič

Tabulka 4.2: Globální adresový prostor

4.6 Aritmeticko logická jednotka

Implementaci jednoduchých matematických výpočtů realizuje ALU. ALU komunikuje s instrukčním sekvencerem na rozhraní AXI4 Stream, jakmile ALU přijme paket s daty, provede operaci na základě operačního kodu a následně pošle zprávu z výsledkem zpět instrukčnímu sekvenceru. ALU je implementována jako stavový automat. Formát vstupu a výstupu je stejný jako v ostatních částech soft mikroprocesoru, tedy fixed point 16.16 formát. Ve vstupním paketu jsou operandy a operační kod serializovány.

Obrázek 4.10: Aritmeticko logická jednotka

AlU implementuje následující operace:

- sčítání
- odečítání
- násobení
- dělení
- logická rotace
- porovnání

Operace porovnání je identická s operací odečítání, chování ALU se nijak neliší, instrukční sekvencer výsledek porovnání ukládá do svého porovnávacího registru.

Během simulace většina operací trvala několik málo taktů, výjimka je operace dělení, která byla odhadnuta na několik desítek taktů. Naštěstí operace dělení není často nutná a dělení konstantním číslem lze optimalizovat na násobení.

ALU tedy zvládne základní aritmetické a logické operace, v případě potřeby využít nadstandardní matematické funkce je nutné využít mikroprocesor ARM.

4.7 Řídicí mikroprocesor ARM

Program mikroprocesoru je složen ze dvou fází, inicializace systému a následně obsluha soft mikroprocesoru.

V první fázi mikroprocesor ARM inicializuje celý systém, oživí periferie, naplní instrukční paměť mikroprocesoru, řídící registry a následně spustí audio pipeline. Instrukční program je uložen v paměti kódu mikroprocesoru jako pole unsigned integer, které se přesune na adresu BRAM instrukční paměti soft mikroprocesoru. V tabulce 4.3 je vidět v jakém formátu se instrukce ukládají do dvou BRAM paměti na strašně FPGA. Konstantní pamět je pamět pro zdrojové konstanty těch instrukcí jejichž rr bit je v hodnotě 1.

Instrukční pamět je instrukčnímu sekvenceru přístupná pouze pro čtení, jakékoliv přeprogramování je proto nutné provést z mikroprocesoru ARM a to pozastavením pipeline, přeprogramování BRAM paměti a opětovné spuštění pipeline.

32 bitů			
operační kod	rr bit	cílová adresa	zdrojová adresa
5 bitů	1 bit	7 bitů	7 bitů

(a) Řídící a adresní pamět

32 bitů
zdrojová konstanta
32 bitů

(b) Konstatní pamět

Tabulka 4.3: Instrukční pamět rozdělena na dvě 32 bitové části pro BRAM pamět.

V druhé fázi čeká ARM na přerušení vyvolané soft mikroprocesorem, toto přerušení signalizuje nutnost spolupráce mikroprocesor ARM na matematické operaci, obsluha přerušení přečte patřičné registry operačního kódu operace a hodnoty operandu, následně převede formát s pevnou desetinnou čárkou dat na formát s plovoucí desetinnou čárkou, vykoná patřičnou operaci a konečně výsledek naformátuje zpět do formátu s pevnou desetinnou čárkou a zapíše zpět do registru soft mikroprocesoru společně se signalizací dokončení operace.

Soft mikroprocesor po zápisu výsledku ukončí požadavek o přerušení a pokračuje dále ve zpracování následujících instrukcí.

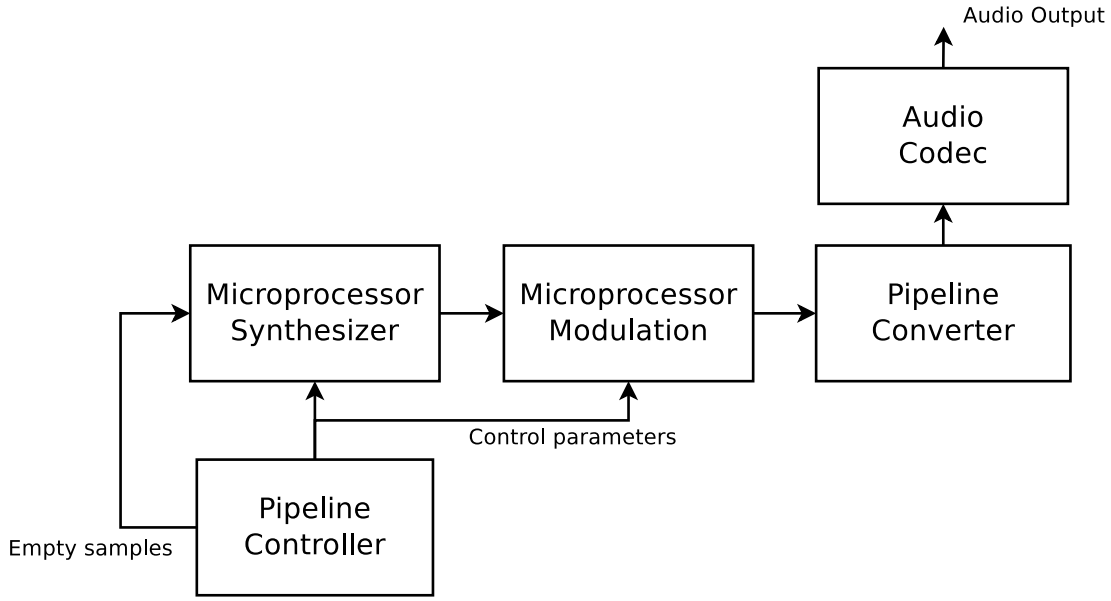
4.8 Audio pipeline

Vzorky mezi mikroprocesory jsou předávány v packetech na AXI4 - Stream rozhraní v blokovém režimu. Tím je vyřešen problém hromadění se vzorku mezi mikroprocesory. Packet je velký $64 + 3$ bity řídící signály a obsahuje dva 32 bitové vzorky levého a pravého kanálu.

Jako první zařízení na pipeline je řadič, který rozvádí do mikroprocesoru řídící parametry jako jsou například rychlost vzorkování, hlasitost aj. ale také neustále posílá "tichý" vzorek levého a pravého kanálu do první mikroprocesoru v pipeline. První mikroprocesor realizuje funkce syntetizéru a na tento tichý vzorek přičítá syntetizovaný zvuk.

Řadič je navržen tak aby šel bez velkých problémů rozšířit pro metody použití v aplikacích, které v této diplomové práci nebyly realizovány a to hlavně v MIDI a pouze Audio modulace. V případě MIDI je možnost řadič upravit tak aby dekódoval vstupní MIDI signál a převáděl je na parametry, které se rozesílají mikroprocesorům. V případě audio modulace lze řadič upravit tak byl na jeho vstupu zapojen výstupní signál z audio codecu Line - In, poté už samozřejmě není potřeba aby první soft mikroprocesor v pipeline realizoval funkci syntetizéru.

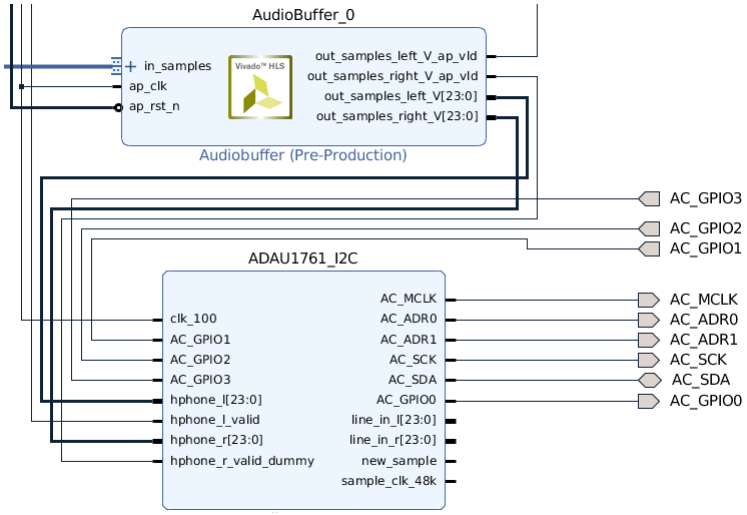
Na obrázku 4.11 lze vidět blokové schéma zapojení audio pipeline tak jak byla popsána výše. Blok pipeline converter realizuje funkci agregace dat z pipeline a změnu formátu vzorků z 16.16 fixed point na 1.23 fixed point a to zachováním pouze znamenkového bitu a celé desetinné části.



Obrázek 4.11: Blokové schema audio pipeline

Řadič na vývojové desce Zedboard je primárně určen pro aplikace s operačním systémem Linux, který obsahuje ovladače řadiče tohoto kodeku. Jelikož jsem se operačnímu systéme chtěla vyhnout a neviděla jsem žádnou zvláštní výhodu přeposílání dat z programové logiky do procesoru a zpět, rozhodla jsem se komunikovat s kodekem přímo v programové logice na konci audio pipeline.

Jelikož komunikace s řadičem ADAU1761, který se nachází na vývojové desce Zedboard probíhá na I2C rozhranní je potřeba tak poslat i vzorky z audio pipeline. Na obrázku 4.12 je zobrazen konec audio pipeline a výše zmíněný blok pipeline converter, který se skládá z převodníku fomátu a I2C řadiče. Řadič přivedenou frekvenci 100 MHz dále dělí a na frekvenci 48 kHz přeposílá vstupní vzorky na vstupech `hphone_l` a `hphone_r` po I2C sběrnici přímo do řadiče ADAU1761.



Obrázek 4.12: Propojení audio pipeline s codecem ADAU1761

Řadič ADAU1761 na vstupu přijímá rozsah hodnot $< -1, 1 >$ ve formátu 1.23, který ale dále rozšíří na rozsah $< -16, 16 >$ ve formátu 5.23. Rozšířený obsah dovolí vnitřní zesílení signálu až na zesílení 24 dB, tento rozsah je dále pak ořezán zpět na formát 1.23 na konečné výstupní zesílení 0 dB. [2]

4.9 Zhodnocení práce

Největší překážka je pro navrhovaný mikroprocesor implementace pokročilých matematických funkcí, tato problematika lze vyřešit různými způsoby, tato práce se zaměřila na možnost využití mikroprocesoru ARM jako matematický koprocesor.

Další z problému je efektivní psaní programu na takto navržený mikroprocesor, problém je velmi patrný u obzvlášť složitých instrukčních programů. Problém by mohl být vyřešen navržením překladače omezeného programovacího jazyka pro architekturu tohoto mikroprocesoru.

Důležitý aspekt je také velmi nákladný přístup do RAM paměti. Nákladný natolik, že v současně podobě je realizace většího množství soft mikroprocesoru na architektuře Zynq není možná právě z důvodu nutného přístupu do RAM paměti. Přístup do RAM paměti je nutný z důvodu nedostatky paměti v registrech. Ovšem během simulací v paměti vždy byly uloženy pouze vzorky audio signálu ať už ve formě tabulek či front. Je možné, že existuje lepší způsob uložení těchto formátu a to takovým způsobem aby nebylo nutné využit vlastnosti náhodného přístupu. Tato myšlenka potřebuje další zkoumání.

Jak již bylo zmíněno, architektura soft mikroprocesoru se dá klasifikovat jako procesor subskalární, to je obzvlášť patrné ve svém stavovém diagramu. Mikroprocesor často čeká ať už na výpočet v ALU tak na výsledek z mikroprocesoru ARM nebo při přístupu do RAM

paměti. Možné zlepšení by spočívalo v implementování další řídící logiky, která by detekovala závislosti instrukcí a v případě nenalezeného žádného konfliktu by mohla instrukce vykonávat paralelně. Je ovšem nutné se zamyslet nad tím, zda-li je skutečně potřeba vyšší výpočetní výkon na úkor více spotřebovaných zdrojů na FPGA čipu.

Oblast, která si zaslouží další zkoumání je hledání možného algoritmu rozdělení programu na soft mikroprocesory. Pro uživatele není důležité vědět či mít kontrolu nad alokováním soft mikroprocesorů v pipeline, mnohem důležitější pro něj je aby systém responsně reagoval a zvládal zpracovat signál bez velkých zpoždění. Tento problém je tedy vhodný k abstrakci za algoritmus který by dokázal předpovědět ideální rozložení problému.

Poslední otázkou zůstává, zda-li by se tento přístup realizace audio syntetizérů vyplatil. Na tuto otázku je těžké odpovědět, jelikož se musíme rozhodnout, které aspekty zařízení či vývoje jsou pro nás důležité. Je zřejmé, že doba a náročnost vývoje zařízení na FPGA je náročnější než implementace na DSP procesorech. To už nemůžeme s jistotou tvrdit o možnosti jejich využití, zvládnutí požadované funkcionality, v tomto ohledu je další výzkum nutný.

Kapitola 5

Závěr

Cílem práce bylo studium existujících elektronických hudebních zařízení a následný vlastní návrh vestavěného zařízení realizující syntézu a modulaci zvuku se zaměřením na technologii programovatelných hradlových polí. Cíl byl splněn.

Zkoumáním elektronických hudebních zařízení se podařila získat skupina metod zpracování zvuku pro které se navrhlo vestavěné zařízení umožňující jejich realizaci. Byly popsány dva fundamentální odlišené přístupy jejich návrhu, analogový a digitální pomocí CPU. Analogové zařízení jsou ve svých možnostech silně omezeny, největší omezení spočívá v jejich statické nátuře a implementace pokročilých metod zpracování signálu. Digitální vestavěná zařízení pomocí CPU jsou omezena svým škálováním kvůli potřebě výpočtu v reálném čase.

Dále se práce věnovala metodám syntézy a modulace zvuku, podrobněji byly popsané vybrané digitální metody a připraveny k návrhu a implementaci.

Byl Uskutečněn návrh digitálního zařízení možného zpracování zvuku na architektuře FPGA a to pomocí jednoduchých digitálních signál mikroprocesorů. Mikroprocesory by plnily funkci zpracování zvukového signálu v pipeline jako syntetizéry i zvukové modulátory. Sestavila jsem simulátor takových omezených mikroprocesorů na kterém jsem simulovala výše zmíněné metody zpracování audio signálu.

Díky experimentům v simulátoru se ukázalo, že velmi jednoduché mikroprocesory opravdu mohou zaujmout roli digitálních audio syntetizérů a modulátorů. Simulátor navíc také pomohl definovat jaké vlastnosti by takový mikroprocesor musel mít a s jaké omezení se dají předpokládat.

Další zkoumání této problematiky se může zaměřit na optimalizaci návrhu mikroprocesoru a introdukcí nástrojů zefektivňujících jejich programování.

Literatura

[1] Ableton, Berlin, Germany: *Ableton Reference Manual Version 9*. 2016.

[2] Analog Devices, Norwood, USA: *ADAU1761, SigmaDSP Stereo, Low Power, 96 kHz, 24-Bit Audio Codec with Integrated PLL*. 2010.

[3] AVNET, Phoenix, USA: *ZedBoard Hardware User's Guide*. 2014.

[4] DJ Pioneer, Jokohama, Japan: *Analog Synthesizer TORAIZ AS-1*. 3 2017.

[5] Gabrielli, L.; Zambon, S.; Fontana, F.: Parallel digital signal processing for efficient piano synthesis. In *2015 23rd European Signal Processing Conference (EUSIPCO)*, 2015.

[6] Heckroth, J.: *Complete MIDI 1.0 Detailed Specification*. MIDI Manufacturers Association, 1996, ISBN 13: 9780972883108.

[7] III, J. O. S.: *Physical Audio Signal Processing*. W3K Publishing, 2010, ISBN 978-0-9745607-2-4.

[8] Jennifer Berg, E. S., Jason Romney: *Digital Sound & Music: Concepts, Applications, and Science*. Franklin, Beedle & Associates Inc, 2016, ISBN 1590282744.

[9] Korg, Tokyo, Japan: *Korg Electribe MX Owner's Manual*. 2003.

[10] Korg, Tokyo, Japan: *Korg Monotron Owner's Manual*. 2010.

[11] Korg, Tokyo, Japan: *Korg Volca Bass Owner's Manual*. 2013.

[12] Korg, Tokyo, Japan: *Korg electribe Music Production Station Owner's Manual*. 2014.

[13] Louise H. Crockett, M. A. E. R. W. S., Ross A. Elliot: *The Zynq Book*. Strathclyde Academic Media, 2014, ISBN 099297870X 9780992978709.

[14] Paker, Ö.; Sparsø, J.; Haandbæk, N.; aj.: A Low-Power Heterogeneous Multiprocessor Architecture for Audio Signal Processing. *Journal of VLSI signal processing systems for signal, image and video technology*, ročník 37, č. 1, 2004: s. 95–110, ISSN 0922-5773.

[15] Shavelis, R.; Ozols, K.; Greitans, M.: Optimization of filtering scheme for reduction of loudspeaker frequency distortions. In *2014 14th Biennial Baltic Electronic Conference (BEC)*, 2014, ISSN 1736-3705, s. 117–120.

[16] SongYu; Lihua: The filter in the DAC of audio processing. In *2010 International Conference on Computer, Mechatronics, Control and Electronic Engineering*, ročník 5, 2010, ISSN 2159-6026, s. 53–55.

[17] Tanev, G.; Božinovski, A.: Virtual Studio Technology inside Music Production. In *ICT Innovations 2013*, editace V. Trajkovik; M. Anastas, Heidelberg: Springer International Publishing, 2014, ISBN 978-3-319-01466-1, s. 231–241.

[18] Wright, B.; Sukittanon, S.: Integer-based wavetable synthesis for low-computational embedded systems. In *2013 Proceedings of IEEE Southeastcon*, April 2013, ISSN 1091-0050, s. 1–4.

[19] Ze-Nian li, J. L., Mark S. Drew: *Basics of Digital Audio*. Wiley, 2011, ISBN 978-3-319-05290-8, 139–181 s.

[20] Zölzer, U.: *DAFX: Digital Audio Effects*. Wiley, 2002, ISBN 0470665998, 116–127 s.

Příloha A

Makroinstrukční sada

V tabulce A.1 jsou uvedeny veškeré implementované instrukce. Je nutno připomenout, že tyto instrukce se implementačně liší v simulátoru a mikroprocesoru a mohou se dobrat k jiným výsledkům.

Instrukce	**Operandy A, B**	**Popis**
NOP		Prázdná instrukce
ADD	registr, registr	Přičte obsah B do registru A
SUB	registr, registr	Odečte obsah B do registru A
MUL	registr, registr	Vynásobí obsah A registrem B
DIV	registr, registr	Vydělí obsah A registrem B
SHL	registr, kontanta	Provede logický posun vlevo registru A obsahem B
SHR	registr, kontanta	Provede logický posun vpravo registru A obshaem B
AND	registr, registr	Provede logický součin registru A registrem B
OR	registr, registr	Provede logický součet registru A registrem B
CMP	registr, registr	Porovná registr A s registrem B
JMP	návěští [1]	Skočí na návěští
JE	návěští [1]	Skočí na návěští pokud A a B mají stejnou hodnotu
JNE	návěští [1]	Skočí na návěští pokud A a B mají rozdílné hodnoty
JG	návěští [1]	Skočí na návěští pokud A je větší než B
JGE	návěští [1]	Skočí na návěští pokud A je větší nebo stejná B
JL	návěští [1]	Skočí na návěští pokud A je menší než B
JLE	návěští [1]	Skočí na návěští pokud A je menší nebo stejná B
MOV	registr, registr	Zkopíruje registr B do registru A
LOAD	registr	Nahraje z RAM paměti hodnotu na adrese registru
STORE	registr, registr	Uloží do RAM paměti hodnotu z registru B na adrese z registru A
SIN	registr	Vypočítá sinus registru A
COS	registr	Vypočítá cosinus registru A
LOG	registr	Vypočítá přirozený logaritmus registru A
TAN	registr	Vypočítá tangens registru A

Tabulka A.1: Makroinstrukční sada

[1] Návěští je uloženo jako číslo instrukce v konstantní paměti.

Příloha B

Program digitalního filtru

V instrukčním kódu B.1 je uveden stereo Biquad filtr realizující dolní propusť s tlumením 12 dB. Program je rozdělen do dvou částí, první je výpočet koeficientů filtru, druhá část je smyčka simulující průchod filtrem pro levý a pravý kanál. Filtrem se vzorek prochází dvakrát.

Instrukční kod B.1: Biquad dolní propusť

```
1  init:   MOV     19      0       400     //register 19 = cutoff frequency
2          DIV     19      1       9
3          MOV     20      1       19      //register 20 K = tan(PI * cutoff)
4          MUL     20      0       3.14159
5          TAN     20      1       20
6          MOV     21      1       20      //register 21 K*K
7          MUL     21      1       20
8          MOV     23      1       20      //register 22 norm  = 1 / (1 + K / Q + K2)
9          MOV     24      0       1       //register 24 Q
10         DIV     23      1       24
11         ADD     23      0       1
12         ADD     23      1       21
13         NOP     0       0       0
14         MOV     22      0       1
15         DIV     22      1       23
16 lowpass:        MOV     26      1       21      //Lowpass 6dB register 26 = a0 =
       K2 * norm
17         MUL     26      1       22      //register 27 = a1 = 2 * a0
18         MOV     27      1       26
19         MUL     27      0       2
20         MOV     28      1       26      //register 28 = a2 = a0
21         MOV     29      1       21      //register 29 = b1 = 2 * (K2 - 1) * norm
22         SUB     29      0       1
23         MUL     29      1       22
24         MUL     29      0       2
25         MOV     31      1       20
26         DIV     31      1       24
27         MOV     30      0       1       //register 30 = b2 = (1 - K / Q + K2) *
       norm
28         SUB     30      1       31
29         ADD     30      1       21
30         MUL     30      1       22
```

```
process:        MOV     36      1       5       //define z1 & z2 twice for two same filters
                MUL     36      1       26      //so the slope will be 2x6 = 12db per octave
                ADD     36      1       32      //register 32 = z1_A_L
                MOV     32      1       5       //register 33 = z2_A_L
                MUL     32      1       27      //register 34 = z1_B_L
                ADD     32      1       33      //register 35 = z2_B_L
                MOV     31      1       29      //register 36 = out_A_L
                MUL     31      1       36      //register 37 = out_B_L
                SUB     32      1       31      //register 38 = z1_A_R
                MOV     33      1       5       //register 39 = z2_A_R
                MUL     33      1       28      //register 40 = z1_B_R
                MOV     31      1       30      //register 41 = z2_B_R
                MUL     31      1       36      //register 42 = out_A_R
                SUB     33      1       31      //register 43 = out_B_R
                NOP     0       0       0
                MOV     37      1       36      //Filter B
                MUL     37      1       26      //out = sourceSample * a0 + z1;
                ADD     37      1       34
                MOV     34      1       36      //sourceSample * a1 + z2 - b1 * out;
                MUL     34      1       27
                ADD     34      1       35
                MOV     31      1       29
                MUL     31      1       37
                SUB     34      1       31
                MOV     35      1       36      //sourceSample * a2 - b2 * out;
                MUL     35      1       28
                MOV     31      1       30
                MUL     31      1       37
                SUB     35      1       31
                MOV     7       1       37      //Filter B output to left channel
                NOP     0       0       0
rightChannel:   MOV     42      1       6       //Filter A
                MUL     42      1       26      //out = source * a0 + z1
                ADD     42      1       38
                MOV     38      1       6       //z1 = source * a1 + z2 - b1 * out
                MUL     38      1       27
                ADD     38      1       39
                MOV     31      1       29
                MUL     31      1       42
                SUB     38      1       31
                MOV     39      1       6       //z2 = source * a2 - b2 * out
                MUL     39      1       28
                MOV     31      1       30
                MUL     31      1       42
                SUB     39      1       31
                NOP     0       0       0
                MOV     43      1       42      //Filter B
                MUL     43      1       26      //out = source * a0 + z1
                ADD     43      1       40
                MOV     40      1       42      //z1 = source * a1 + z2 - b1 * out
                MUL     40      1       27
                ADD     40      1       41
```

```
83          MOV     31      1       29
84          MUL     31      1       43
85          SUB     40      1       31
86          MOV     41      1       42      //z2 = source * a2 - b2 * out
87          MUL     41      1       28
88          MOV     31      1       30
89          MUL     31      1       43
90          SUB     41      1       31
91          MOV     8       1       43      //Filter B output to right channel
92          NOP     0       0       0
93          FIN     4       0       1
94          JMP     0       0       0       process
```