UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2016 Jan Lokvenc

Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Využití Raspberry Pi pro optimalizaci audio výstupu

Jan Lokvenc

Bakalářská práce

2016

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2015/2016**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jan Lokvenc**
Osobní číslo:	**I13169**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Využití Raspberry Pi pro optimalizaci audio výstupu**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem práce je navrhnout a realizovat komplexní zařízení využívajícího systému Raspberry Pi pro optimalizaci audio výstupu, V teoretické části autor představí principy optimalizace pro audio výstup, technologii Raspberry Pi a provede logický návrh řešení. V praktické části autor realizuje navržené řešení, otestuje a navrhne případné postupy pro optimalizaci. Součástí práce bude komplexní programová dokumentace a vyhodnocení pilotního nasazení.

Rozsah grafických prací:

Rozsah pracovní zprávy: **50**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

*** JESSALYN, Blossom Meghan. Raspberry pi. S.l.: Betascript Publishing, 2012. ISBN 9786136272641.**

*** DONAT, Wolfram. Learn Raspberry Pi programming with Python. New York: Apress, 2014, xxi, 231 pages. Technology in action series. ISBN 14302642410.**

Vedoucí bakalářské práce: **Mgr. Josef Horálek, Ph.D.**
Katedra informačních technologií

Datum zadání bakalářské práce: **31. října 2015**
Termín odevzdání bakalářské práce: **13. května 2016**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Mgr. Josef Horálek, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2016

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 12. 5. 2016

Jan Lokvenc

PODĚKOVÁNÍ

Chtěl bych poděkovat svému vedoucímu Mgr. Josefu Janu Horálkovi, Ph.D. za cenné rady při zpracování této práce a mamince a tatínkovi za podporu během studia.

ANOTACE

Tato práce pojednává o implementaci aplikace pro práci se zvukem na cílové platformě Raspberry Pi. V první části se zabývá seznámením s Raspberry Pi, principy digitálního zpracování zvukového signálu a barvami šumů včetně jejich možností využití. Druhá část se poté věnuje především vývoji multiplatformních aplikací s použitím frameworku JUCE.

KLÍČOVÁ SLOVA

Raspberry Pi, ARM, Fourierova analýza, šum, JUCE

TITLE

Using Raspberry Pi for optimalization of audio output

ANNOTATION

This research discusses the implementation of audio processing application on the target platform Raspberry Pi. The first part deals with introduction with the Raspberry Pi, the principles of digital audio signal processing and colors of noises, including their uses. The second part then focuses especially on the development of multi-platform applications using the framework JUCE.

KEYWORDS

Raspberry Pi, ARM, Fourier analysis, noise, JUCE

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZKRATEK A ZNAČEK

USB	Universal Serial Bus
RAM	Random Access Memory
GPIO	General-purpose input/output
CPU	Central Proccesing Unit
HDMI	High-Definition Multi-media Interface
SoC	System on chip
GPU	Graphic processing unit
OS	Operating systém
APT	Advanced Packaging Tool
RISC	Reduced instruction set computing
CISC	Complex instruction set computing
MIPS	Millions of instructions per second
ALU	Arithmetic logic unit
FFT	Fast Fourier Transformation

ÚVOD

Zvukem a jeho mnoha podobami jsme provázeni téměř neustále. Často se však setkáváme se špatně optimalizovanou formou, což může mít za následek nedostatečný požitek například z poslechu hudby, ale také fatální následky na zdraví lidského sluchu.

V dnešní době je možné využít mnoho komerčních řešení profesionální úrovně. Jedná se zvukové ekvalizéry s možností vizuálního zobrazení frekvenčního spektra upravovaného signálu, rozšiřující možnosti moderních digitálních mixážních pultů nebo softwarové aplikace pro úpravu zvukového záznamu. Většina těchto možností však vyžadují vysokých investic.

Cílem bakalářské práce je sestrojit malé přenosné zařízení, které bude poskytovat vestavěný modul pro řešení kalibrace zvukových systémů, který bude cenově dostupnější, než většina stávajících zařízení. Jako platforma pro vytvoření tohoto zařízení bylo vybráno Raspberry Pi, které se stává stále častější součástí moderní domácnosti.

V práci bude seznámeno s mikropočítačem Raspberry Pi a jeho možnostmi. Další kapitoly se poté budou zabývat digitalizací zvukového signálu a jeho spektrální analýzou. Kromě představení obecných principů těchto metod, také bude představen princip implementace zrychlené metody. V závěru bude nahlédnuto do vlastnosti různých barev šumů a jejich využití. Představeny budou nejznámější varianty a způsoby využití šumu v oblasti zvukového inženýrství.

Část praktická se poté zabývá především implementací aplikace na využitou platformu, která poskytuje funkce jednoduchého hudebního přehrávače, spektrálního analyzátoru a zabudované nástroje, které jsou potřebné k úpravě frekvenční charakteristiky zvukového výstupu.

1 REŠERŠE

V této závěrečné práci bude vyvíjeno přenosné zařízení, které bude fungovat jako spektrální analyzátor s možností kalibrace zvukového výstupu, realizované na platformě Raspberry Pi.

Cílem práce je tedy implementovat aplikaci běžící na platformě Raspberry Pi pro generování šumu a následnou vizualizaci spektrografu vstupního signálu v reálném čase. Pro optimalizaci bude k dispozici grafický ekvalizér pro možnost změny frekvenčního spektra zvukového výstupu. Vývojem podobné aplikace se v devadesátých letech zabýval Konstantin Zeldovich. Jeho program se jmenuje Winscope 2.51, který lze použit jako osciloskop nebo spektrální analyzátor pro zobrazení frekvenčního spektra. Tento program je volně šiřitelný (Collinson, 2016). Stejně tak se analýzou hudebního signálu na digitálních strojích zabýval James Andel'son Moorer ve své disertační práci (Moorer, 1975). Cílem jeho práce bylo dekódovat signál na jednotlivé nástroje a jejich melodické linky a jejich následný zápis do notového záznamu.

Proč Raspberry Pi? Jedná se o plnohodnotný levný počítač velikosti peněženky, který nachází uplatnění ve vědě i v domácnostech, od kterého můžeme očekávat i další rozšíření v souvislost s automatizací a Internetem věcí (Richardson, 2015). Použitím Raspberry Pi pro levnou modernizaci, konkrétně systému videoreklamy, se zabývali také na konferenci v Jasi (2013 RoEduNet International Conference 12th Edition: Networking in Education and Research, 2014). Přímo pro vývoj zvukového přenosného zařízení využil Raspberry Pi Florian Meier se svými kolegy při vývoji zařízení s názvem The JamBerry. Toto zařízení slouží muzikantům pro společnou hru bez fyzického kontaktu s využitím streamování hudebního signálu přes internet (Meier, 2014).

2 RASPBERRY PI

Raspberry Pi je nízkonákladový, jednodeskový počítač velikosti kreditní karty vyvinutý v Anglii firmou Raspberry Pi Foundation s cílem podpory počítačové vědy na školách a rozvojových zemí (Cellan-Jones, 2011). První verze přišla na trh v roce 2012 (Graziano, 2014).

2.1 Generace Raspberry Pi

Tato kapitola seznamuje s generacemi Raspberry Pi. V této kapitole je odkazováno na oficiální web (Raspberry Pi foundation, 2014).

Raspberry Pi se neustále vyvíjí a zdokonaluje. Neustále přichází na trh nové verze toho malého počítače. Aktuální novinkou uveřejněnou na trh je Raspberry Pi 3. Samozřejmě můžeme dále vybírat i z nabízených nižších řad Raspberry Pi 2 model B a Raspberry Pi 1 model B+ a A+. Další variantou je ještě menší verze nazývaná Raspberry Pi Zero.

Nejlevnější variantou je Raspberry Pi 1 Model A+. Jedná se zařízení poskytující 256MB RAM, jeden USB port, 40 GPIO pinů, ale už zde chybí ethernetový port. Poté je tu Model B+, který je konečnou verzí původního Raspberry Pi. Disponuje dvakrát větší pamětí než Model A+, čtyřmi USB porty, počtem 40 GPIO pinů a ethernevovým portem, který v nižších modelech chybí.

V únoru 2015 byla zveřejněna druhá generace Raspberry Pi 2 konkrétně Model B. Toto tehdy nové zařízení sdílí většinu hardwarové specifikace s Pi 1 B+. Oproti svému předchůdci používá rychlejší 900MHz quad-core ARM Cortex-A7 CPU procesor a disponuje vyšší kapacitou operační paměti RAM, která je rovna 1 GB. Pi 2 je plně kompatibilní s první generací desek, což umožnuje používat periferie vyvíjené pro první řadu.

Novinkou roku 2016 je Raspberry Pi 3 Model B, které má nový 1.2GHz 64bitovým quad-core ARM Cortex-A53 CPU procesor, 1GB RAM a dále disponuje integrovaným 802.11n pro připojení k bezdrátové síti a také podporou Bluetooth 4.1.

Poslední zmíněné variantou je Raspberry Pi Zero, které je počítačem poloviční velikosti vzoru A+. Toto opravdu miniaturní zařízení v sobě má zabudovaný 1GHz procesor, 512MB RAM, mini-HDMI a USB On-The-Go port.

2.2 Procesor

Raspberry Pi využívá tzv. Systém na čipu (SoC), což představuje konstrukci, se na jedné desce nachází procesor zároveň s grafickou kartou a operační pamětí. Systém na čipu, který je použitý

u první Raspberry Pi je téměř stejný jako ty, které se používaly pro chytré telefony. Raspberry Pi je založeno na Broadcom BCM2835 SoC, který se skládá z 700MHz ARM1176JZF-S procesoru, VideoCore IV grafické karty (GPU) a operační paměti (Brose, ©2010-2012). Novější generace používají dokonalejší čipy. Druhá generace je postavená na Broadcom BCM2836 SoC a třetí Broadcom BCM2837 SoC (Upton, 2016).

2.3 Operační paměť

V této sekci bude seznámeno s problémy a možnostmi práce s operační pamětí. Tato část práce se odkazuje na příspěvek ze Stack Exchange (Adamski, © 2016).

Vzhledem k tomu, že množství použitelné paměti na Raspberry Pi není nijak vysoké, je nutné s ní dobře pracovat. Jelikož nejčastěji Raspberry Pi běží pod operačním systémem Linux, používá se operační paměť jako paměť vyrovnávací. Také vzhledem k množství operační paměti může docházet k nedostatku tohoto zdroje. Je nutné doplnit, že se paměť dělí mezi procesor a grafický čip.

Vyrovnávací paměť slouží pro překlenutí rozdílných rychlostí dvou zařízení za účelem urychlení dané operace. Pod Linuxem platí, že pokud nějaká paměť je nevyužitá aplikacemi, potom ji operační systém využije jako vyrovnávací paměť pro rychlejší chod systému. Pokud aplikace potřebují více paměti, tak bude využita paměť, která sloužila jako vyrovnávací. Nedochází k zabraňování přidělení více paměti pro aplikace. Přesto je dobré s operační pamětí šetřit v rámci zvýšení výkonu systému.

Nedostatek paměti se projeví využíváním disku jako odkládacího prostoru, což má za příčinu velké zpomalení systému. Aplikace, speciálně desktopové, v dnešní době vyžadují mnoho paměti. Některé dokonce více, než může Raspberry Pi poskytnout. Není to však jediný problém. Aplikací totiž může být spuštěno více a poté se jejich paměťové nároky sčítají a může opět docházet k vyčerpání paměti a stejnému efektu na výkon systému.

Dělení paměti je nutné z důvodu sdílené paměti mezi procesorem a grafickým čipem. Při nastavování správného poměru je nutné mít představu o konečném využívání daného zařízení a dle toho se rozhodovat. Existují různé možné varianty k jednotlivým velikostem operační paměti. Starší verze Raspberry Pi disponovalo 256MB RAM a bylo možné ji dělit čtyřmi způsoby.

240/16 – Jedná se o ideální nastavení pro práci na počítači, při které není třeba 3D grafiky nebo jiných grafických nároků.

224/32 – Toto nastavení se moc nepoužívá. Nepřináší téměř žádnou výhodu oproti předcházejícímu nastavení. Jediný důvod by mohlo být používání 32 bpp framebufferu.

192/64 – Je dobré nastavení, pro aplikace, kde budeme potřebovat pracovat s 3D grafikou. Bohužel to pořád nemusí stačit pro přehrávání filmů ve vysokém rozlišení. Také pokud potřebujete pracovat s věcmi, které spotřebují hodně paměti (textury), tak mohou nastat problémy.

128/128 – Nastavení pro graficky náročné operace. Ideální na přehrávání fullHD videa.

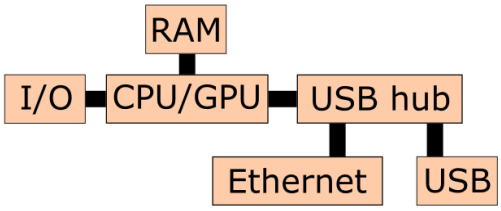
Dynamické dělení je způsob jak lze danou operační paměť rozdělit. Je dobré zmínit, že existují dvě verze daného firmwaru. V dnešní době, přesněji po roce 2012, se setkáme spíše s novou verzí. Celé nastavení lze definovat v konfiguračním souboru, který se nachází v souboru s cestou `/boot/config.txt`, v kterém je nutné přepsat následující řádek požadovaným množstvím paměti pro grafický čip například následujícím způsobem.

```
gpu_mem=16
```

Možné přípustné hodnoty pro nastavení množství grafické paměti jsou 16, 32, 64, 128, 256. Tyto hodnoty jsou uvedeny v MB.

2.4 Připojení do sítě

Zde je nutné si rozdělit Raspberry Pi dle jednotlivých generací. Modely A, A+ a Zero nemají zabudované ethernetové rozhraní. Jediná možnost jak se připojit k síti je pomocí externí síťové karty, která musí být připojená pomocí USB. Vyšší modely, konkrétně B, B+ a všechny modely řad 2 a 3 již mají zabudované ethernetové rozhraní a poskytované přes klasický konektor RJ-45. Je nutné zmínit, že se nejedná gigabitový ethernet, ale dochází zde ke zpomalení na rychlost USB verze 2. Dochází k tomu z důvodu připojení ethernetového rozhraní přes USB sběrnici. Poslední nejvyšší řada, tedy 3. generace, již má, kromě drátového připojení, zabudované Wi-Fi rozhraní a dále disponuje podporou Bluetooth (FAQS, 2016).



Obrázek 1 Schéma Raspberry Pi, zdroj: vlastní

2.5 Připojení externích zařízení

Raspberry Pi je minimalistické zařízení, které samo o sobě nejsme schopni ani nijak ovládat. Jako řešení tohoto problému Raspberry Pi poskytuje, dle verze, různé počty USB portů, na které jsme schopni připojit většinu USB 2.0 zařízení. Mezi tyto zařízení může patřit například myš, klávesnice, zvukové a síťové karty, pevné disky a jiné (FAQS, 2016).

Ještě stojí za zmínění, že pro připojení grafického výstupního zařízení jako je například monitor nebo televize, je zde k dispozici HDMI výstup.

2.6 Operační systém

Raspberry Pi je standardně spojováno s jedním operačním systémem a to systémem Raspbian. Je dobré si uvědomit, že se však nejedná o jediný použitelný OS pro toto zařízení. V této kapitole si představíme hlavní zástupce operačních systému a jejich zastoupení v používání.

Raspbian založený na distribuci Debian označován jako standartní OS pro Raspberry Pi. Je vyvíjen společností Raspberry Pi Foundation, stejně jako Raspberry Pi samotné. Tento operační systém přichází s APT pro instalaci softwaru přímo z Raspbian repozitářů a také přichází s tzv. raspi-config, který umožňuje intuitivní správu konfigurace systému. V roce 2014 byla implementována knihovna RPi.GPIO pro jednodušší programování komunikace v jazyku Python s GPIO rozhraním (Raspberry Pi 2: Which OS is best?, © 2009-2016).

Ubuntu MATE je operační systém optimalizovaný pro platformu Raspberry Pi. Tato distribuce zahrnuje mnoho předinstalovaných aplikací. Případné další lze instalovat ze Software Center nebo pomocí stažení z libovolného zdroje a následné instalace. Je doporučeno používat SD kartu třídy 6 nebo 10 pro lepší výkon systému (Raspberry Pi 2: Which OS is best?, © 2009-2016).

Fedora je dalším z možných použitelných systémů. Je nutné zvolit méně náročného správce oken. Tento operační systém je možné nainstalovat pouze na SD karty velikosti 8 GB kvůli uzpůsobenému obrazu systému. Standardní instalace v sobě obsahuje několik jednoduchých aplikací pro úpravu textu nebo prohlížení internetu a podobně. Další aplikace je možné instalovat pomocí správce balíků yum. Také podporuje knihovnu RPi.GPIO pro podporu GPIO rozhraní (Raspberry Pi 2: Which OS is best?, © 2009-2016).

Jak by se mohlo zdát, není Raspberry určeno pouze pro odnože Linuxu. S příchodem Windows 10 je možné na Raspberry Pi instalovat i operační systém Windows (Heath, © 2016).

Pro začátky s Raspberry Pi je nejjednodušší cestou používaní operačního systému Raspbian pro intuitivnost nástrojů, které podporuje, a také pro vysokou podporu od výrobce. Jedná se o nejpoužívanější operační systém, který používá téměř 70 % uživatelů Raspberry Pi (Raspberry Pi 2: Which OS is best?, © 2009-2016).

2.7 Instalace OS

Pro instalaci operačního systému na platformu Raspberry Pi je možné použít dva možné způsoby instalace.

Prvním způsobem pro instalaci je použít instalační nástroj Noobs. Tento nástroj je možné stáhnout přes stránky výrobce. Po stažení je nutné ho nainstalovat na SD kartu. Po vložení předinstalované paměťové karty do Raspberry Pi a následného zapnutí se dostaneme do grafického průvodce instalací. Je důležité zmínit, že v offline režimu je možné nainstalovat pouze Raspbian. Při možnosti použít internetové připojení během instalace je možné vybrat z několika operačních systému, kterými jsou například Pidora, Arch nebo Windows 10. V mnoha prodejnách lze koupit předinstalovanou SD kartu, právě s připraveným instalačním nástrojem Noobs.

Druhou možností je stáhnout obraz požadovaného operačního systému s následně systém nainstalovat na SD kartu pomocí speciálního nástroje. Nástroj, který se používá pro tento úkon pod operačního systémem Windows je Win32DiskImager. Jedná se jednoduchý nástroj, který nainstaluje operační systém na SD kartu pro následné spuštění na Raspberry Pi (Installing operating system images using Windows, 2016).

3 ARM ARCHITEKTURA

Procesor je čip, který zajištuje veškeré provádění programových instrukcí, komunikaci periferií a provádění matematických operací. Jako hlavní parametry procesoru jsou výkon uváděný frekvencí, počet jader a velikosti primární a sekundární vyrovnávací paměti také nazývané jako paměť cache. Poté ještě můžeme rozdělit procesory na dva základní typy architektur a to RISC a CISC. Tyto dva typu se od sebe liší sadou instrukcí. V prvním případě se jedná o redukovanou instrukční sadu a v druhém o komplexní, která v sobě zapouzdřuje i tzv. makro instrukce. Jedná se o složitější instrukce, které mohou být složeny z více základních (Rouse, © 1999 - 2016).

ARM je jednou z architektur založených na RISC architektuře vyvinutá firmou Advanced RISC Machines (ARM). Tato firma vyrábí 32 a 64bitové vícejádrové procesory. Tyto procesory jsou navrženy tak, že pracují s minimálním množstvím instrukcí, díky čemuž jsou schopny pracovat rychleji a poskytovat více operací za sekundu (MIPS). Díky absenci složitých instrukcí a dalším optimalizacím zde dostaneme lepší výkon, než s procesory založenými na architektuře CISC s obsáhlejší instrukční sadou (Rouse, © 1999 - 2016).

Procesory ARM jsou díky menší instrukční sadě a tím i menším počtem potřebných tranzistorů realizovatelné na menších prostorových nárokách. Díky této vlastnosti jsou vyhledávanými procesory pro použití v miniaturních zařízeních. V dnešní době jsou široce používány ve velkém množství spotřebních elektronických zařízeních, jako jsou chytré telefony, tablety, multimediálních přehrávačů a především také v minipočítačích Raspberry Pi (Rouse, © 1999 - 2016).

ARM architektura disponuje základními vlastnostmi, které jsou typické pro procesory založené na této filozofii. Těmito charakteristickými vlastnosti jsou (Rouse, © 1999 - 2016).

- Load/store architektura
- Ortogonální instrukční sada
- Provádění instrukce v jednom cyklu
- Energeticky úsporný provoz
- 64bitový a 32bitový spouštěcí stav
- Podpora virtualizace hardwaru

Load/store architektura je taková, kde jsou veškeré instrukce rozděleny do dvou skupin. První skupina se zabývá pouze správou paměti. Jedná se o načítání a ukládání dat mezi registry a operační pamětí. Druhou skupinou jsou veškeré aritmetickologické operace. Důležitým znakem

ALU operací je fakt, že můžou pracovat pouze s operandy, které jsou uloženy registrech procesoru. Právě tato vlastnost poskytuje vyšší rychlost prováděných operací oproti procesorům založených na CISC architektuře, kde tak být nemusí (Flynn, c1995).

Ortogonální instrukční sada je taková, která povoluje používat libovolný adresovací režim pro všechny instrukce. Toto znamená, že daná instrukce nemusí využívat konkrétní registr procesoru. Plně ortogonální sada se v praxi moc nevyužívá, protože většina programů využívá jen zlomek ortogonálních adresních režimů. Díky absenci některých ortogonálních adresovacích režimů je možné ušetřené bity využít efektivnějším způsobem (Jackson, c1999).

Provádění instrukce v jednom cyklu je vlastnost, kdy všechny instrukce mají stejnou délku v bitech. Všechny instrukce se provádějí stejně dlouho a to jeden cyklus. Jeden cyklus se skládá z pěti fází.

- Načtení instrukce
- Dekódování instrukce
- Provedení instrukce
- Čtení z paměti
- Zápis výsledků do paměti nebo registrů

Energeticky úsporný provoz souvisí s použitím jednotné bitové šířky pro všechny instrukce. Pro vykonávání složitějších operací je možné použít instrukce menší bitové šířky, které poskytují možnost většího objemu vykonávaného kódu a pořád stejně ekonomický provoz. Tato vlastnost je nesmírně důležitou pro možnost použití v mobilních zařízeních (Marwedel, 2011).

64bitový a 32bitový spouštěcí stavy a přepínání mezi nimi je užitečné například v případě, že máme nainstalovaný 64bitový operační systém a potřebujeme spustit 32bitovou aplikaci. Mezi stavy lze přepínat pomocí Secure monitoru, hypervizoru nebo operačního systému. Důležitou vlastností je, že na 32bitové architektuře nelze spustit 64bitovou aplikaci, zatímco obráceně je to možné (ARM Cortex-A Series Programmer's Guide for ARMv8-A, © 1995-2016).

Podpora virtualizace hardwaru je realizována pomocí ARM Architecture virtualization extension a Large Physical Address Extension. Toto umožnilo používat takzvaných hypervisorů (virtuálních strojů). Jedná se o možnost poskytovat komplexní softwarové prostředí pro hypervisory (Virtualization Extensions, © 1995-2016).

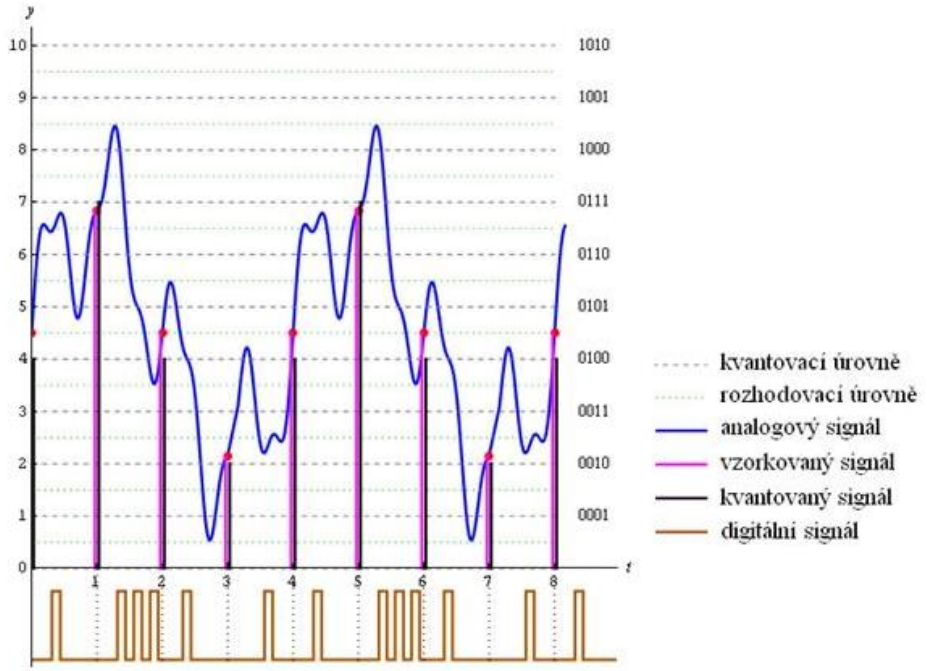
4 ZPRACOVÁNÍ SIGNÁLU

Zvuk se v běžném světě nachází v podobě mechanického vlnění. K tomu, abychom mohli se zvukem pracovat, je nutné ho převést na elektrický signál. K převodu akustického vlnění na elektrický signál slouží například mikrofon. Po nasnímání zvuku mikrofonem máme k dispozici spojitý analogový signál. K tomu abychom mohli se signálem pracovat na číslicovém stroji, jakým je počítač, je třeba spojitý analogový signál převést do digitální podoby, která je vyjádřena konečným počtem diskrétních hodnot.

V běžných osobních počítačích jsou k tomuto používány zvukové karty, které mohou být jak integrované, tak externí. Tyto karty nám poskytují možnost připojení zvukových zařízení, jakými jsou například reproduktory nebo mikrofony. Mají v sobě zabudovaný A/D a D/A převodníky pro převod analogového signálu do digitálního a zpět.

4.1 A/D převodníky

Převod analogového signálu do diskrétní podoby nazýváme digitalizace. Tento proces lze rozdělit do dvou kroků, vzorkování a kvantování spojitého signálu. Při vzorkování je třeba určit omezený počet vzorků, které budeme snímat s danou periodou, kterou nám určí vzorkovací frekvence. V druhém kroku kvantování jde o tzv. úrovňovou diskretizaci. Zjednodušeně se jedná o zaokrouhlení hodnot nasnímaných vzorků na předem určené hodnoty, které jsou vyjádřeny nějakým binárním kódem (Reichl & Všetička, © 2006 - 2016).



Obrázek 2 Digitalizace signálu, zdroj: (Reichl & Všetička, © 2006 - 2016)

Vzorkovací frekvence je hodnota, která určuje, jak často budeme snímat hodnotu daného vzorku analogového signálu za jednu sekundu. Pro určení minimální hodnoty vzorovací frekvence se používá tzv. Nyquistova podmínka. Toto pravidlo říká, že vzorkovací frekvence musí být dvakrát větší než maximální frekvence snímaného signálu. Pokud se tak nestane, bude docházet ke zkreslení, které by způsobilo rušení struktury výsledného signálu. Z tohoto plyne, že pro snímání lidským uchem slyšitelného spektra, kde se pohybuje maximální frekvence kolem 20 kHz, je nutné použít vzorkovací frekvenci 44,1 kHz, což je standardní hodnota pro hudební CD nosiče. Dalšími často používanými hodnotami vzorkovací frekvence je 32 kHz pro telekomunikační přenosy a 48 kHz pro záznam zvuku do televize. Dále se používají násobky uvedených vzorkovacích frekvencí. Vyšší vzorkovací frekvence zajišťuje vyšší kvalitu výsledného signálu (Vzorkování signálu, © 2006 - 2016).

Bitová hloubka je dalším důležitým parametrem pro digitalizaci, který určuje, kolik bude používáno kvantizačních úrovní. Při použití 16bitové hloubky jsou dané hodnota vyjádřena 16 bity, což znamená, že je možné definovat 65 536 hodnot. Tato bitová hloubka je standardem pro hudební CD nosiče. Používají se i vyšší hodnoty. Například v hudebních DSP se používají 24bitové převodníky. Čím je dané hodnota vyšší, tím je menší kvantizační krok, což znamená, že dostáváme kvalitnější výsledný signál s menším zkreslením (Vzorkování signálu, © 2006 - 2016).

4.2 D/A převodníky

Převod z diskrétní podoby na analogový signál probíhá obvykle pomocí vytváření elektrického napětí odpovídající hodnoty (možné i elektrického proudu). Pro výstup nelze nastavit libovolnou hodnotu. Signál může nabývat pouze diskrétních hodnot, což má za následek schodovitou strukturu výsledného signálu. Tato chyba vzniká z důvodu omezeného počtu kvantizačních úrovní vstupního digitálního signálu. Vzniklé zkreslení se označuje jako kvantizační chyba (Převodníky analogových a číslicových signálů, 2014).

5 SPEKTRÁLNÍ ANALÝZA

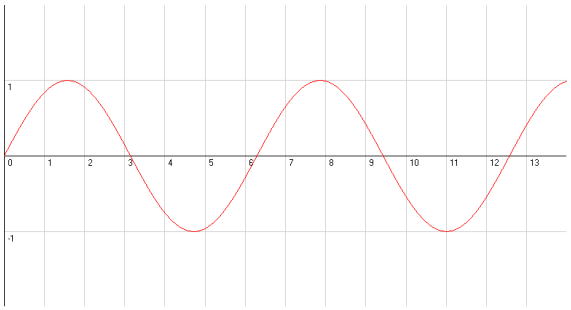
Jakékoliv signály jsou v běžném světě tvořeny množstvím signálů rozdílných frekvencí. K tomu abychom zjistili přesnější složení takového signálu, musíme docílit rozdělení těchto frekvenčních částí. K tomuto nám slouží metoda, kterou nazýváme spektrální analýza. Jedná se o postup, při kterém dochází k převodu signálu ve formě vzorku snímaných v určitém čase, do signálu z pohledu jeho spektrálního složení. Algoritmy, které slouží k řešení této problematiky, jsou nazývány jako Fourierova analýza. Pomocí této analýzy lze graficky znázornit signál dle složení frekvenčního spektra (Holčík, 2009).

5.1 Úvod do Fourierovy analýzy

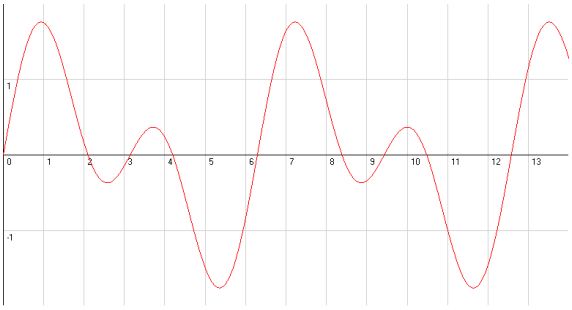
Fourierova analýza je souhrnné označení pro matematické postupy, které slouží k rozkladu signálu na sinusoidy. Jedná se metody, které jsou pojmenovány po francouzském matematikovi a fyzikovi, který se jmenoval Jean Baptista Josef Fourier (1768-1830). Tento vědec se během svého života, mimo jiné, zabýval šířením tepla a možností jeho reprezentace tepelných variací pomocí sinusových vln (Holčík, 2009).

Pro pochopení této problematiky budou uvedeny pojmy od základů a způsob výpočtu Fourierovi transformace, včetně její inverzní varianty.

Tóny jsou takové zvuky, které lidské ucho vnímá jako příjemné. Patři mezi ně například zvuk jednotlivých hudebních nástrojů. Z fyzikálního hlediska se jedná o takové signály, které mají periodicky se opakující výchylku nebo intenzitu v závislosti na čase. Nejjednodušším znázorněním tónu je graf funkce sinus.

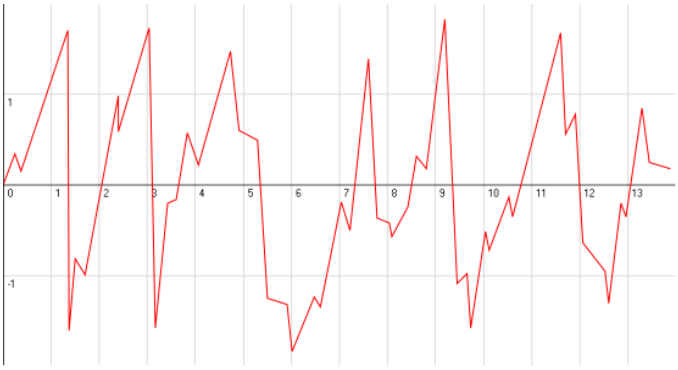


Obrázek 3 Graf průběhu jednoduchého tonů



Obrázek 4 Graf průběhu složeného tónu

Ruchy jsou takové zvuky, které jsou pro lidské ucho nepříjemné. Jedná se například o ruch z ulice nebo bouchání dveřmi. Z fyzikálního hlediska se však jedná o zvuk, který nemá periodický průběh výchylky nebo intenzity v závislosti na čase.



Obrázek 5 Graf průběhu ruchu

Jednoduchý tón je se zvuk, který je dán právě jednou frekvencí a je možné ho znázornit funkcí sinus. Tyto tóny jsou ochuzeny a tzv. barvu zvuku. Toto znamená, že při dvou stejných zvucích, nejsme schopni od sebe tyto zvuky rozpoznat. Je to z důvodu absence vyšších harmonických, které dodávají zvuku barvu na základě zdroje zvuku (Reichl & Všetička, © 2006 - 2016).

Složený tón je složený z více jednoduchých tónů. Tyto zvuky mají svojí barvu. Reprezentací takového signálu už není sinusoida. Z matematického hlediska však lze tuto funkci vytvořit pomocí skládání více funkcí sinus (Reichl & Všetička, © 2006 - 2016).

Fourierova řada je vzorec pro vyjádření složených tónu pomocí skládání jednotlivých sinusoid. Tedy jednotlivých jednoduchých tónů resp. jednotlivých obsažených frekvencí v daném složeném tónu (Reichl & Všetička, © 2006 - 2016).

$$y = y_{m1} \cdot \sin \omega_1 \cdot t + \ y_{m2} \cdot \sin \omega_2 \cdot t + \ \cdots + \ y_{mn} \cdot \sin \omega_n \cdot t \qquad (1.1)$$

Tento vzore lze vyjádřit elegantněji pomocí sumy.

$$\sum_{i=0}^{n} y_{m(2i+1)} \cdot \ \sin\big((2i+1)\omega_i t\big) \qquad (1.2)$$

Fourierova transformace je metoda pro převod mezi časovým a frekvenčním znázorněním složených tónů. Základní vztah pro možný převod ve spojitém čase je vyjádřen pomocí následujícího vzorce (Klíč, Dubcová, & Volka, 2002):

$$S(\omega) = \int_{-\infty}^{\infty} s(t) e^{-i\omega t} dt \qquad (1.3)$$

Při použití na výpočetních zařízení je nutné používat obměnu, kterou lze spočítat transformací v diskrétním tvaru. Poté bude vypadat postup následovně (Klíč, Dubcová, & Volka, 2002):

$$D(n) = \sum_{k=0}^{N-1} d(k) e^{\frac{-ink2\pi}{N}}, n = 0, \dots, N-1 \tag{1.4}$$

Inverzní Fourierova transformace je metoda pro zpětný převod z frekvenčního vyjádření na vyjádření časové. Tato metoda našla své upotřebení především, když se začaly používat digitální zvukové procesory pro úpravu signálů. Pro obecný převod platí následující vztah (Klíč, Dubcová, & Volka, 2002):

$$s(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} S(\omega) e^{i\omega t} d\omega \tag{1.5}$$

Varianta pro výpočet inverzní transformace v diskrétním tvaru potom vypadá takto (Klíč, Dubcová, & Volka, 2002):

$$d(k) = \frac{1}{N} \sum_{n=0}^{N-1} D(n) e^{\frac{ink2\pi}{N}}, k = 0, \dots, N-1 \tag{1.6}$$

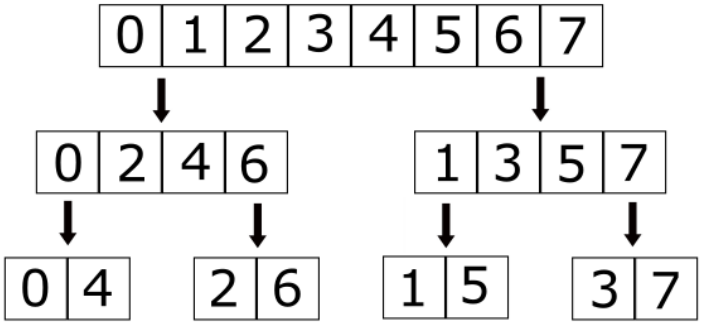
5.1.1 Rychlá Fourierova transformace

Jedná se o rychlejší metodu výpočtu Fourierovy analýzy, označovanou jako FFT z anglického Fast Fourier Transformation. Při použití klasického postupu výpočtu Fourierovy analýzy algoritmus dosahuje složitosti, která je rovna $O(N^2)$ počtu operací. Nejedná se tedy o příliš efektivní algoritmus. Naproti tomu algoritmy FFT dosahují i složitosti O(N log N). Nejznámější verzi algoritmu pro výpočet FFT uvedli v roce 1965 pánové J. W. Cooley and J. W. Tukey (VANDERPLAS, © 2012-2015).

Cooley-Tukey algoritmus je založený na rozdělení vstupní sady dat, spočítání Fourierovy transformace nad těmito částmi a následného sečtení hodnot pro získání původní požadované Fourierovy transformace. Je tedy využíváno tohoto vztahu (VANDERPLAS, © 2012-2015):

$$F_n = \sum_{k=0}^{\frac{N}{2}-1} \left(e_n \cdot e^{-j\frac{2\pi}{N}(2k)n} + o_n \cdot e^{-j\frac{2\pi}{N}(2k)n} \right) \tag{1.7}$$

Dělení vstupu na separované části se uskutečňuje dle toho, zda se jedná o sudé nebo liché prvky. Tento děj probíhá rekurzivně, dokud nedosáhneme části o velikosti dvou prvků. Poté začneme opačným směrem vypočítávat jednotlivé hodnoty transformací. Pro názornost rozkladu je níže uvedeno schéma.



Obrázek 6 Schéma rozkladu vstupních dat při FFT

6 TEORIE ŠUMŮ

V této kapitole bude představeno, co to takový šum je, kde se s ním můžeme setkat, jaké druhy speciálních šumů jsou známi a jakým způsobem jsou možné využít pro zvukovou optimalizaci.

6.1 Co je to šum

Šum je vlastně jiné pojmenování pro hluk. Hluk můžeme definovat jako zvukové variace. Z běžného života tedy můžeme mluvit o jakémkoliv zvuku, který je při označení hluk, spíše nežádoucím. Tedy zvuky, často hlasité, které narušují slyšitelnosti žádoucích zvuků. Jako příklad těchto zvuků je možné uvést hlasitou hudbu od sousedů, prolétající letadlo, hluk běžné dopravy, ale také například vítr nebo šumící vodu v řece.

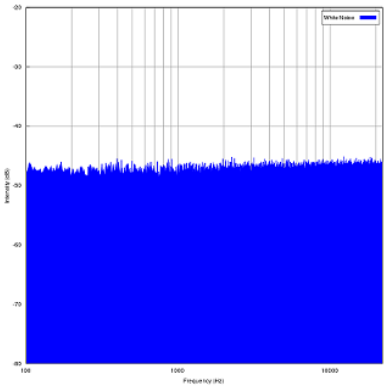
Hlukem tedy může být cokoliv od tichého příjemného zvuku přírody po téměř nesnesitelný hluk v podobě tryskového motoru při startu letadla nebo sirény hasičského vozu.

Při vnímání hluku, či poslechu hudby je nutné také přemýšlet o dvojím pojetí vnímání okolního zvuku. Jedná se o vnímání psychické a fyziologické. V prvním případě dochází k vnímání a zpracování zvuku našimi smysli a nervovou soustavou. Jedná se například o proces poslouchání hudby a vyhodnocování estetického působení na pocity člověka, pomocí vyhodnocení mozku a nervové soustavy. Ovšem je třeba zmínit i druhou možnost vnímání a to fyziologickou. Člověk neslyší svým uchem celé hudební spektrum, ale tělo jako celek tyto části pořád vnímá. Neslyšitelné spektrum pořád může ovlivnit lidský organismus například změnou nálad nebo změnou činností, které jsou prováděné podvědomě (například funkce orgánů) (Barthes, 1991).

Nyní konkrétněji k šumům v oblasti hudby. Při hraní na hudební nástroj, nahrávání, přehrávání atd. vznikají v pozadí nežádoucí zvukové nečistoty, které lze slyšet v klidnější okamžiky průběhu jedné ze zmíněných činnosti. Zde zmíněný jev byl definován jako tzv. bílý šum (White Noise, © 2015).

6.2 Bílý šum

Při zpracování signálu, je jako bílý šum označovaný náhodný signál, který má konstantní spektrální hustotu. Toto znamená, že libovolné zvukové pásmo daného signálu určité šířky disponuje stejnou hustotou jako jiné. Pojmenování bílý šum je odvozeno od tzv. bílého světla. Je důležité si ale uvědomit, že nekonečnost frekvenčního rozsahu je pouze teoretická. Pokud by tak nebylo, pak by se jednalo o signál neomezeného výkonu, což je nemožné (Carter, 2013).



Obrázek 7 Frekvenční spektrum bílého šumu, zdroj: (Colors of noise, 2001-)

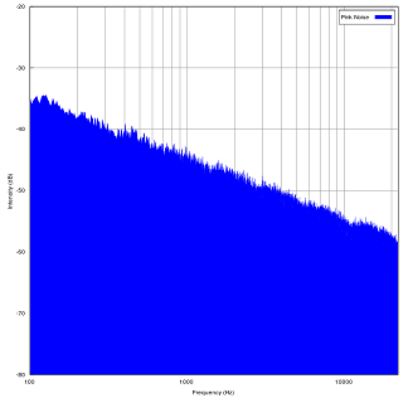
Bílý šum má v praxi mnoho využití. Používá se například v sirénách pohotovostní vozidel, a to díky jeho snadnému šíření v okolním prostředí. Velkou roli hraje při produkci elektronické hudby, kde je možné tento šum využít při zvukové syntéze a to například při úpravě, či obnově zvukové stopy některých nástrojů. Také se využívá v informatice pro generování náhodných čísel nebo ve zdravotnictví pro léčbu onemocnění sluchu. Hlavním a stěžejním použitím, na kterém je postavena tato práce, je použití pro testování mikrofonů, reproduktorů a jiných zvukových zařízení a také možnost využití pro optimalizaci zvukového výstupu pro kompenzaci špatných akustických vlastností auditoria například při hudebních produkcích. Pro specifické výsledky se často používají jiné, k tomuto účelu uzpůsobené šumy.

6.3 Barvy šumů

Existuje množství různých šumů, které byly vytvořeny pro odlišné způsoby použití.

Růžový šum, také označovaný jako 1/f šum, je takový signál, jehož výkon jednotlivých frekvenčních hustot je přímo úměrný převrácené hodnotě dané frekvence. Toto znamená, že při zdvojnásobení frekvence dochází k útlumu výkonu frekvenční hustoty o 3 dB. Výkon je tedy stejný v logaritmických souřadnicích pro všechna stejně široká pásma, tedy v jednotlivých oktávách. Toto je důvodem, proč růžový šum poměrně dobře odpovídá logaritmickému vnímání zvuků lidským sluchem. Právě toto je důvodem pro použití při kalibraci zvukových aparátů například na hudebních produkcích (Pink Noise, 2016).

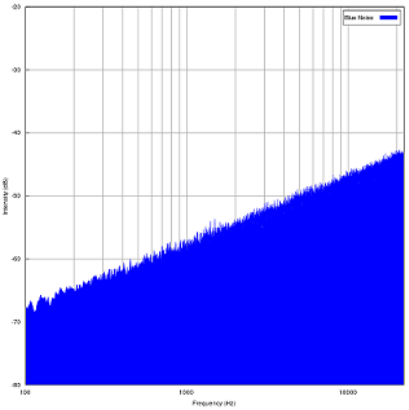
Červený šum, často také nazývaný jako šum hnědý z anglického Brownian noise, který je pojmenovaný po Robertu Brownovi. Tento šum je svoji povahou velmi podobný šumu růžovému. Dochází zde o větší útlum výkonu frekvenční hustoty na frekvenci, a to o hodnotu 6 dB, což znamená, že je na poslech hlubší, protože má více utlumené vyšší frekvence (Violet Noise, 2016).



Obrázek 8 Frekvenční spektrum růžového šumu, zdroj: (Colors of noise, 2001-)

Modrý šum disponuje výkonem frekvenční hustoty stoupajícím o 3 dB na oktávů se stoupající frekvencí. V grafice se často zaměňuje název za volnější název „šum s minimální nízkou frekvencí a bez výkyvu výkonu“. Používá se například pro opravy zvukových záznamů nebo transformacím digitálních obrázků (Blue Noise, 2016).

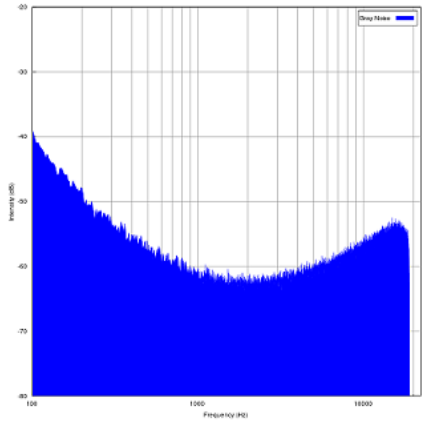
Purpurový šum je podobný svou charakteristikou modrému šumu. Charakteristický je svým větším zesílením výkonu frekvenční hustoty signálu, a to o 6 dB na oktávu (Violet Noise, 2016).



Obrázek 9 Frekvenční charakteristika modrého šumu, zdroj (Colors of noise, 2001-)

Šedý šum je signál, který je svojí frekvenční charakteristikou navržený dle psychoakustického vnímání zvuku. Jedná se o šum, který má rozdílný výkon frekvenční hustoty pro jednotlivá frekvenční pásma. Výkon je přesně takový, aby šum byl jako celek lidským uchem vnímán jako zvuk, který má výkon frekvenční hustoty v celém svém spektru stejně veliký. Tedy výkon

frekvenční hustoty je silnější v nižších frekvencích, utlumen v dobře slyšitelném středním pásmu a pomalu zesilován v oblasti frekvencí vysokých. Tento šum je možné použít například pro testování lidského sluchu (Gray Noise, 2016).



Obrázek 10 Frekvenční spektrum šedéo šumu, zdroj: (Colors of noise, 2001-)

7 GENEROVÁNÍ RŮŽOVÉHO ŠUMU

Vzhledem k charakteru je růžový šum ideální volbou pro použití při optimalizaci audio výstupu. V této kapitole tedy budou představeny metody, které nám umožňují tento šum získat. Pro generování růžového šumu existuje celá řada možných použitelných algoritmů a velká většina jich je založena na filtraci z bílého šumu. V této části se odkazuje na (WHITTLE, 1999).

Robert Bristow-Johnson's three pole and three zero filter je metoda využívající speciálně navržený signálový filtr typu FIR. Jedná se o číslicový filtr, který regresním výpočtem potlačuje frekvence bílého šumu pro vznik šumu růžového. Pro správnou funkci filtru je třeba dosadit správné hodnoty pólů a nul, které jsou definovány autorem této metody. Tyto hodnoty znázorňuje následující tabulka.

Póly	Nuly
0.99572754	0.98443604
0.94790649	0.83392334
0.53567505	0.07568359

Tabulka 1 Koeficienty Bristow-Johnsonova filtru

Následný vzorec slouží pro spočítání výkonu frekvenčního hustoty n-tého frekvenčního pásma (Bristow-Johnson).

$$y[n] = \frac{p0}{z0}x[n] + \frac{p1}{z0}x[n-1] + \frac{p2}{z0}x[n-2] - \frac{z1}{z0}y[n-1] - \frac{z2}{z0}y[n-2] \qquad (1.8)$$

Při použití vyššího počtu sdružených pólu a nul získáváme vyšší přesnost výsledku, ale také nevýhody v podobě vyššího rozsahu nefiltrovaných frekvencí v dolním frekvenčním pásmu a vyšší výpočetní složitosti použitého algoritmu.

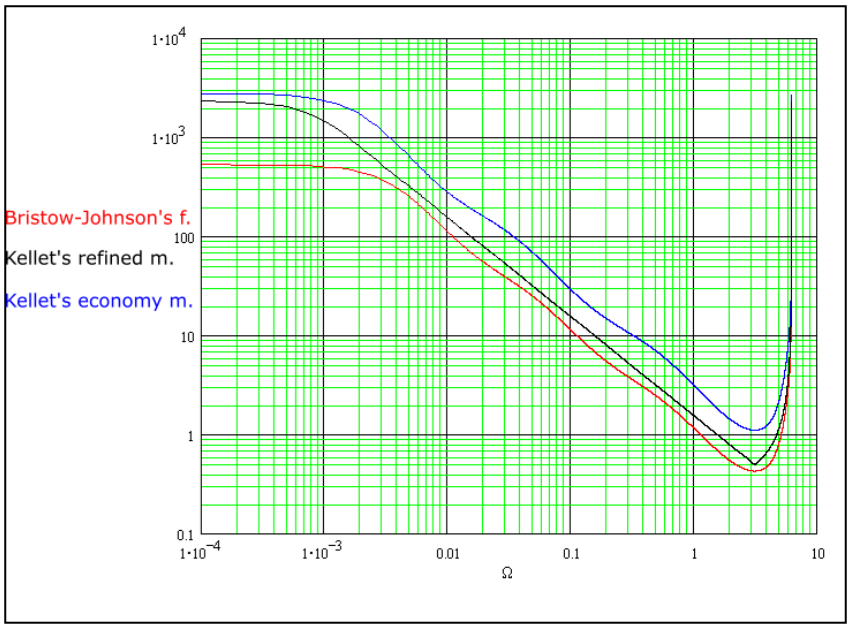
Paul Kellet's refined method je algoritmus, který filtruje vstupní bílý šum pomocí postupného použití šesti koeficientů na každý vzorek vstupního bílého šumu, přičemž dochází k závislosti výpočtu hodnoty aktuálního prvku na hodnotách, které byli vypočteny při výpočtu prvku předchozího. Tato metoda není nijak složitá, jak ukazuje následná ukázka zdrojového kódu. Pro úplnost ukázky je důležité zmínit, že se jedná o část kódu, která se nachází v cyklu, kde počtem opakování je počet potřebných vzorků vstupně/výstupního šumu.

```
b0 = 0.99886 * b0 + white * 0.0555179;
b1 = 0.99332 * b1 + white * 0.0750759;
b2 = 0.96900 * b2 + white * 0.1538520;
b3 = 0.86650 * b3 + white * 0.3104856;
b4 = 0.55000 * b4 + white * 0.5329522;
b5 = -0.7616 * b5 - white * 0.0168980;
pink = b0 + b1 + b2 + b3 + b4 + b5 + b6 + white * 0.5362;
b6 = white * 0.115926;
```

Paul Kellet's economy method je postup velmi podobný předchozímu. Jedná se však o více ekonomický postup vzhledem k výpočetnímu času a výkonu výpočetního stroje.

```
b0 = 0.99765 * b0 + white * 0.0990460;
b1 = 0.96300 * b1 + white * 0.2965164;
b2 = 0.57000 * b2 + white * 1.0526913;
pink = b0 + b1 + b2 + white * 0.1848;
```

Následující graf znázorňuje porovnání frekvenčních charakteristik růžových šumů získaných pomocí výše popsaných metod.

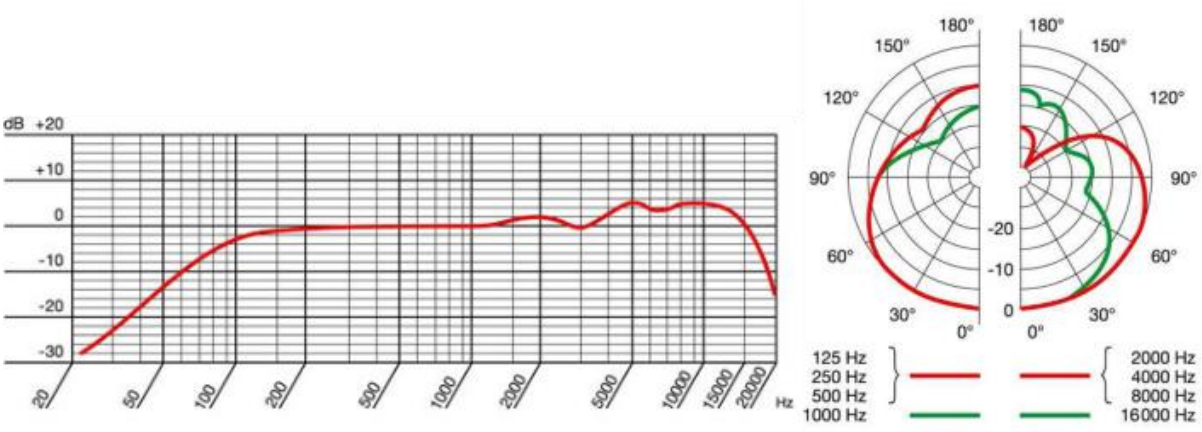


Obrázek 11 Porovnání výsledků jednotlivých metod, zdroj: (WHITTLE, 1999)

8 VYUŽITÍ ŠUMŮ VE ZVUKOVÉM INŽENÝRSTVÍ

Využívání šumů v oboru zvukového inženýrství slouží například k testování zařízení pro přeměnu zvukového vlnění na elektrický signál a zpět (například mikrofon a reproduktory) nebo pro analyzování špatné akustiky prostoru a následné kompenzace výsledků pomocí ekvalizérů.

Při testování mikrofonů je mikrofon vystaven neustále znějícímu šumu a to ze všech možných úhlů. Na mikrofonů se sleduje hlavně směrová charakteristika a ovlivnění výkonů pro snímání jednotlivých frekvencí, které se pak vykreslují do grafů. Jedná se o grafy přenášeného frekvenčního spektra a směrové charakteristiky zařízení.



Obrázek 12 Směrová a frekvenční charakteristika, zdroj: (TV Freak, © 1998-2015)

Testování reproduktorů a celých zvukových systémů probíhá podobným způsobem. Do reproduktorů je puštěn šum a následně se snímá a analyzuje. Výstupem je graf, který vyjadřuje frekvenční charakteristiku přenosu daného reproduktoru, a informace o směrovosti. Tento způsob lze též využít pro kompenzaci akustiky prostoru.

Použití pro kompenzaci špatných akustickým vlastností prostoru probíhá pomocí spuštění krátkého zvukového impulsu šumu a jeho následného snímání z několika míst auditoria. Nasnímané signály poté prochází spektrální analýzou. Posledním krokem je nastavení ekvalizéru použitého zvukového systému pomocí speciálních algoritmů. Celý tento mechanismus již bývá součástí moderních digitálních zvukových zařízení, především mixážních pultů.

9 PRAKTICKÉ ŘEŠENÍ

9.1 Použité komponenty

V rámci bakalářské práce bylo zkompletováno zařízení, založené na platformě Raspberry Pi, které umožnuje uživateli přehrávat zvukové soubory WAV a po přepnutí do módu kalibrace i optimalizovat zvukový výstup zařízení pro kvalitnější poslech hudby. Pro realizaci práce bylo použité standardní zařízení Raspberry Pi řady 2 s připojeným 3,5" LCD TFT dotykovým displejem, externí zvukovou USB kartou Steinberg c12 a nainstalovaným operačním systémem Raspbian Jessie.

9.2 Instalace operačního systému a ovladačů

Pro instalaci byla vybrána varianta instalace pomocí obrazu operačního systému a nástroje Win32DiskImager. Zvolený byl standardní operační systém pro Raspberry Pi a to systém založený na distribuci Debian optimalizovaný pro Raspberry Pi Raspbian Jessie. V současné době se jedná o nejnovější verzi. Tento operační systém byl vybrán především z důvodu podpory ovladačů zvoleného dotykového displeje.

Po úspěšném nainstalování operačního systému na SD kartu a následném spuštění nemá systém k dispozici víc než několik desítek MB. Veškeré místo je zaplněno operačním systémem. Pro možnost využívat veškeré místo užívané paměťové karty je nutné udělat několik dále popsaných kroků.

- Spustit terminál
- Zadat příkaz: `sudo raspi-config`
- V grafickém rozhraní spustit volbu: `Expand Filesystem`
- Restartovat systém

Při kompletaci zařízení a všech jeho standardně podporovaných funkcí bylo též nutné nainstalovat ovladače pro použitý dotykový displej. Tyto ovladače byli stahovány ze serveru prodejce použitého zařízení Raspberry Pi pomocí příkazu:

```
wget http://rpishop.cz/files/LCD-show-151020.tar.gz
```

Následně bylo třeba stažený balík rozbalit a to pomocí příkazu:

```
tar -zxvf LCD-show-151020.tar.gz
```

Poté už nám stačí spustit instalaci, což provedeme přemístěním do složky s instalačním skriptem a následným spuštěním. Tyto úkony se provádí následnými dvěma příkazy:

```
cd LCD-show
```

```
sudo ./LCD35-show
```

Po instalaci daných ovladačů již stačí pouze restartovat operační systém a připojený dotykový displej bude připraven k použití. Pro případné přepnutí grafického výstupu zpět na HDMI konektor je nutné restartovat systém po zadání příkazu

```
sudo LCD-show/LCD-hdmi
```

Přepnutí zpět se provádí obdobným způsobem.

```
sudo LCD-show/hdmi-LCD
```

9.3 Volba programovacích nástrojů

Zde bude popsáno několik možností, které by bylo možné použít k realizace implementace aplikace a dále bude seznámeno se závěrečnou volbou.

Pro implementaci byly zvažovány tři alternativy. Prvním jazykem byl jazyk Python, který je podporován a doporučován pro programování na platformě Raspberry Pi. Druhou volbou byl jazyk Java a JavaFx. JavaFx umožnuje poměrně dobré rozhraní pro práci se zvukem. Též v programovacímu jazyku Java je možně používat externí frameworky pro práci se zvukem jako jsou například The Beads Project nebo Java Media Framework. Poslední alternativou byl jazyk C++, který sám neposkytuje žádnou podporu jak pro zvuk, tak ani pro vývoj grafických aplikací. Je zde tedy nutnost použít externích frameworků. Pro podporu audia by bylo možné použít například Clam, Nsound nebo JUCE.

Při výběru jazyka je důležitým ukazatelem hlavně rychlost, protože se jedná aplikaci běžící v reálném čase, a náročnost na výkon, protože je Raspberry Pi trochu limitující zařízení. Především z těchto důvodů můžeme Python a Javu vyřadit. Jedná se o interpretované jazyky a jejich nízká rychlost by se mohla stát neodstranitelnou překážkou. Zbylo tedy projít možnosti C++ a jeho frameworků. Po nastudování nejvíce vyhovoval pro potřeby aplikace implementované aplikace framework JUCE. Tento Framework je volně šiřitelný pro nekomerční účely. Poskytuje nejen příjemnou podporu pro práci se zvukem, ale také dovoluje vytvářet grafické rozhraní a vyvíjet multiplatformní aplikace.

Možnosti vývoje multiplatformních aplikací tohoto frameworku bylo plně využito. Vzhledem k tomu, že vyvíjet aplikace přímo na platformě Raspberry Pi je trochu nešikovné, tak celý vývoj, včetně prvního testování, byl realizován na osobním počítači s operačním systémem Windows 10 v programovacím prostředí Visual Studio 2013. Díky této skutečnosti byla vytvořena aplikace, která dovoluje použití nejen na zařízení realizovaném na platformě Raspberry Pi, ale i na osobních počítačích s operačním systémem Windows i Linux.

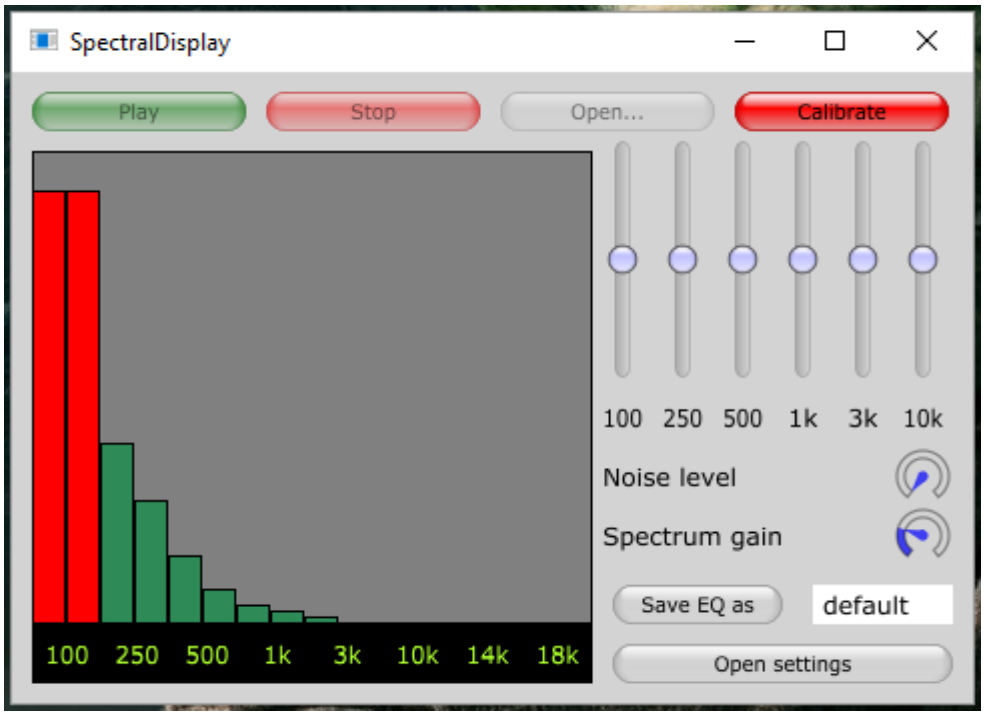
9.4 Design aplikace

Při této činnosti je za potřebí neustále brát v potaz, že se musí jednat design, který by měl splňovat intuitivnost a přehlednost zobrazení a to i na malém displeji výsledného zařízení. Z tohoto důvodu jsou absolutně vymezeny některé rozměry a hranice. V podstatě jediné, co při změně velikosti aplikačního okna mění svojí velikost, je zobrazovací část zvukového spektra.

Nutno bohužel konstatovat, že to není jedinečné řešení. Překreslování obsahu je totiž nejnáročnější operace a při zvětšení dochází k překreslování větší plochy a na pomalejších zařízeních se může stávat, že při zobrazení na celou obrazovku, přestane být zobrazování plynulé. S tímto problémem se potýká i Raspberry Pi 2 s připojeným externím monitorem.

9.5 Rozvržení

Zde si představíme základní rozvržení implementované aplikace, která bude sloužit pro optimalizace zvukového výstupu a přehrávání hudby. Pro umožnění všech funkcionalit daného zařízení byla implementována aplikace poskytující intuitivní grafické rozhraní pro, co možná nejjednodušší, ovládání pomocí připojeného dotykového displeje. Grafické rozhraní poskytuje základní tři části pro uživatele



Obrázek 13 Grafické rozvržení aplikace

Část pro ovládání přehrávání, skládající se ze třech tlačítek.

Část pro zobrazení měnícího se frekvenčního spektra, závislého na aktuálním zvukovém vstupu pomocí připojeného snímacího zařízení.

Část sloužící pro optimalizování zvukového výstupu zařízení, která se skládá z přepínače mezi přehrávacím a kalibračním módem, šesti pásmového grafického ekvalizéru, ovladačů citlivostí a tlačítek pro ukládání resp. načítání námi vytvořeného nastavení grafického ekvalizéru.

9.6 Funkce aplikace

Zde bude seznámeno se všemi podporovanými funkcemi vyvíjené aplikace a metodikou používání tohoto nástroje pro optimalizaci zvukového výstupu.

První část je věnována ovládání jednoduchého přehrávače zvukových souboru WAV. Disponuje pouze třemi prvky, což značí, že ovládání bude velmi triviální.

První tlačítko s názvem „Play“ slouží ke spuštění předem načteného souboru. Pokud není zvukový soubor načtený, tlačítko je neaktivní.

Druhé tlačítko popsané jako „Stop“ Slouží obdobným způsobem jako první. Místo první však zastavuje již spuštěnou zvukovou stopu. Pokud aplikace nepřehrává, je dané tlačítko neaktivní.

Třetí tlačítko „Open“ po stisknutí uživateli zobrazí dialog pro výběr zvukového souboru WAV, který chce uživatel přehrát. Po úspěšném otevření se zpřístupní první tlačítko „Play“ a vzniká tu možnost si daný soubor přehrát.

Druhá část je věnována spektrální analýze zvukové vstupu. Slouží pro grafické zobrazení hlasitosti v oblasti jednotlivých frekvenčních pásmech. Zobrazování je realizováno pomocí různých velikostí sloupců, které mění barvu podle toho, zda přesáhnou hlasitost předgenerovaného růžového šumu uloženého v paměti aplikace, či nikoli. Pokud ne, je barva zelená. Při překročení daných hodnot sloupec zčervená. Ve spodní části se nachází legenda k zobrazovanému spektru. Nachází se zde několik čísel, která jsou na pozicích více vpravo obohacena o písmeno k, které značí, že se jedná o tisíc. Legenda je uvedená v jednotkách Hertz. Zobrazujeme tedy zvukové spektrum od přibližně 60 Hz do 20 kHz.

Třetí část slouží pro optimalizace zvukového výstupu aplikace. Protože se jedná o multifunkční část s mnoha komponentami, bude rozepsána po částech.

Tlačítko s názvem „Calibrate“ slouží k zapnutí módu pro kalibraci. Vypíná možnost přehrávání a povoluje spustit růžový šum, který je nezbytný pro využití mechanizmu pro optimalizování zvukového výstupu.

Šesti-pásmový grafický ekvalizér, pomocí kterého jsme schopní optimalizovat výstup. Každý z horizontálních posuvníků představuje ovládání hlasitosti v dané části zvukového spektra, přesněji specifikované dle popisků pod každým z těchto posuvníků. Každá část spektra je ovlivňována na centrální uvedené frekvenci v popisku a jejího okolí širokého jednu oktávu. Jedná se o ekvalizér aktivní, což znamená, že je možné frekvenci zesílit resp. zeslabit dle potřeby akustiky prostoru. Hodnoty frekvence byli vybrány dle toho, jak obvyklé jsou jejich úpravy ve zvukařské praxi.

Otočný ovladač s popiskem „Noise level“ slouží pro určení výstupní hlasitosti růžového šumu. Tento ovládací prvek funguje pouze v kalibračním módu.

Otočný ovladač s popiskem „Spectrum gain“ neslouží jako posílení hlasitosti vstupu, ale pouze jako nástroj pro zvětšení zobrazovaných sloupců zvukového spektra. Tento ovladač slouží pro optimální zobrazení zvukového spektra k velikosti aplikačního okna.

Tlačítko s názvem „Save EQ as“ slouží pro uložení aktuálního nastavení grafického ekvalizéru do externího textového souboru pod názvem, který je uveden v přilehlém textovém poli,

standardně vyplněném jako „default“. Uložené nastavení najdeme ve složce SpectralDisplaySettings, která se nachází v dokumentech přihlášeného uživatele.

```
void  saveButtonClicked()
{
  getEqValues();

  File currentDir = File::getSpecialLocation(File::userDocumentsDirectory);
  String path = currentDir.getFullPathName();

  File dir(path + "/SpectralDisplaySettings");
  if (!dir.exists()) {
      dir.createDirectory();
  }
   File f(path + "/SpectralDisplaySettings" + "/"+ nameOfSetting.getText()
                                                                  ".txt");
  if (f.exists()) {
      f.deleteFile();
  }
   FileOutputStream myfile(f, 1000);
   for (size_t i = 0; i < 6; i++) {
      myfile.writeString(slider100Hz.getTextFromValue(eqSettings[i]));
      myfile.writeString("");
  }
  NativeMessageBox::showMessageBox(AlertWindow::AlertIconType::InfoIcon,
                                          "Info",
                                          "Succesfully save!",
                                          nullptr
                                          );
}
```

Tlačítko s názvem „Open settings“ slouží pro načtení uloženého uživatelského nastavení ze souboru.

9.7 Postup optimalizace zvukového výstupu

Použití hlavní funkce aplikace není nijak složité, ale pro elegantní a rychlé nastavení je doporučeno používat definovaný postup, skládající se z těchto tří kroků.

- Nastavení hlasitosti šumu na požadovanou hlasitost zvuku v auditoriu.
- Zesílení vstupní úrovně signálu tak silně, aby část zobrazovaného spektra začala problikávat červenou barvou. V tomto kroku je možno použít ovladač „Spectrum gain“ pro lepší zobrazení grafického výstupu.
- Nastavení grafického ekvalizéru tak, aby červeně problikávalo, co možná nejvíce sloupečků, které zobrazují frekvenční spektrum.

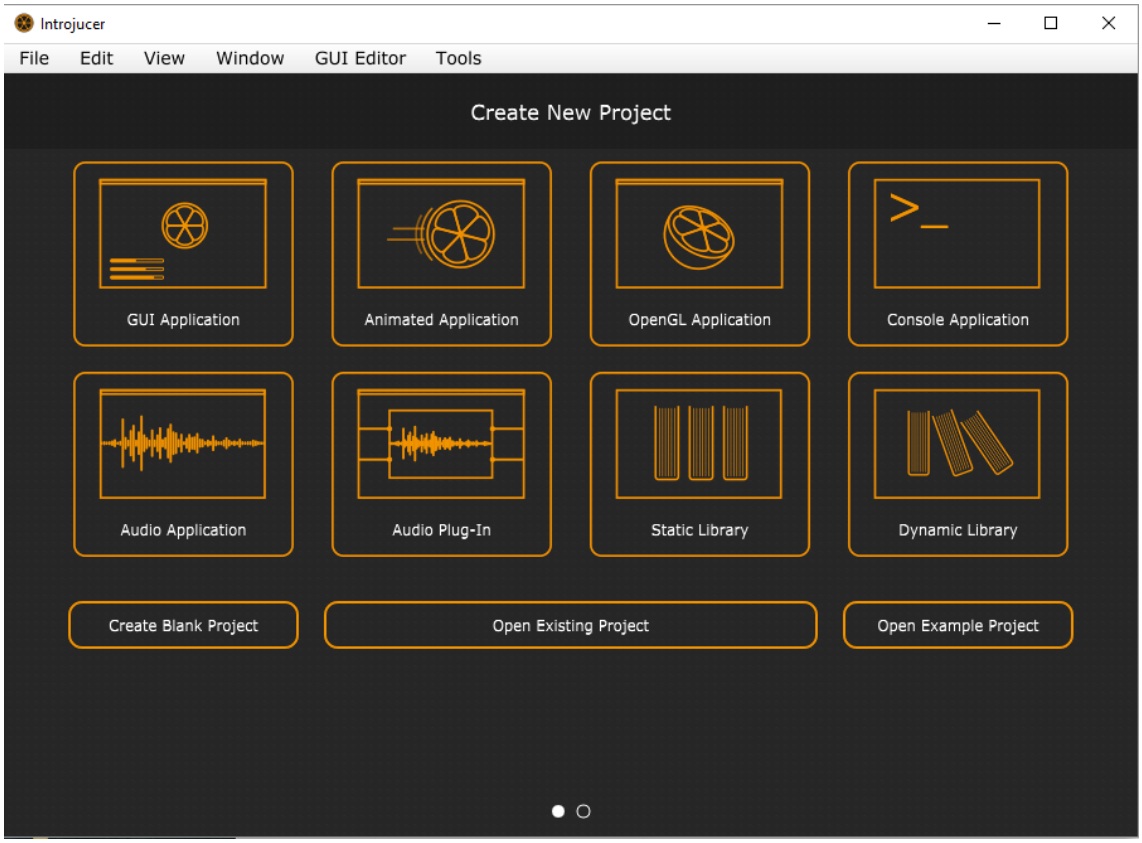
Nutné připomenout, že pro většinu potřebných funkcí pro optimalizaci je nutné mít zapnutý příslušný mód, který je signalizován červenou barvou tlačítkového přepínače.

9.8 Popis implementace

Popisem implementace budeme v této části rozumět založení projektu, samotný proces kódování s popis použitých tříd a také kompilace pod jednotlivými platformami s použitím frameworku JUCE.

9.8.1 Založení projektu

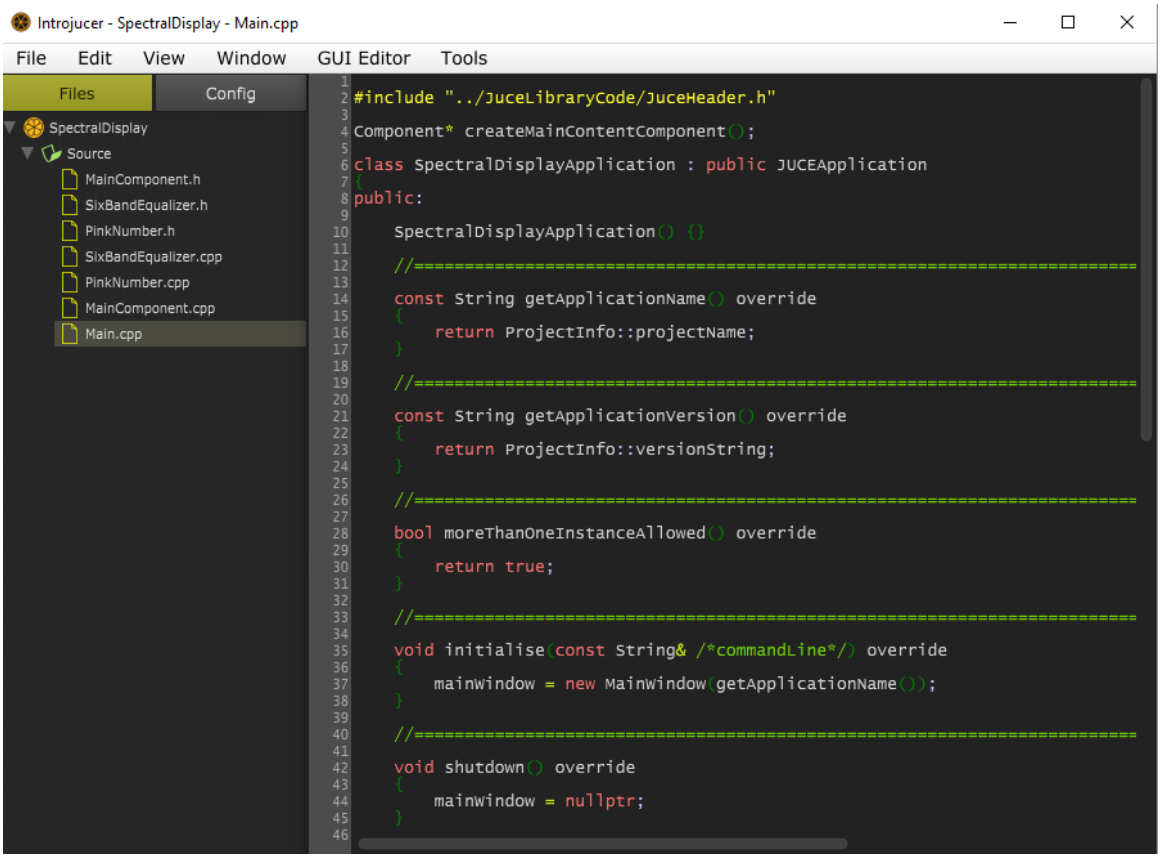
Pro založení projektu, JUCE poskytuje grafický nástroj s názvem Introjucer. Jedná se o grafického průvodce zakládáním projektu s možností nastavení projektu a v neposlední řadě o jednoduché vývojové prostředí pro psaní kódu aplikace. Při zakládání projektu je hned v první nabídce možnost vybrání typu projektu. Možností je tu více. Samozřejmě můžeme vytvořit prázdný projekt. Dále tu jsou možnosti „GUI Application“, „OpenGL Application“, „Audio Application“ a jiné. Tyto možnosti se od sebe liší připojenými knihovnami daného frameworku JUCE a předgenerovaným kódem k danému typu aplikace, který má za úkol zjednodušit a urychlit zakládání projektu.



Obrázek 14 Introjucer

Po vybrání šablony projektu se ukáže nabídka pro nastavení umístění a názvu projektu. Pravé části se poté nastavuje, pro jaké platformy se bude aplikace vyvíjet. Po odsouhlasení vybraných voleb se již ukáže okno, kde je možné připojit další soubory frameworku, upravovat cesty k souborům. V levé dolní části se zobrazují dvě tlačítka. Jedno slouží pro uložení projektu a druhé

pro otevření projektu námi zvoleném vývojovém prostředí. V záložce „Files“ je možné se dostat do vývojového prostředí pro psaní kódu. U grafických komponent, jako jsou formuláře, se zde nachází i grafický návrhář, který následně vygeneruje kód pro vzhled grafického rozhraní. Slouží pro ulehčení tvorby interaktivních oken, avšak používání tohoto nástroje není moc šťastnou volbou. Pokud bude použit, doporučuje se projít si kód a případné chyby doopravit ručně.



Obrázek 15 Vývojové prostředí Juce

9.8.2 Použitá šablona a její struktura

Pro implementaci byla použita šablona „Audio application“. Jedná se o implementaci třídy, která dědí z třídy AudioAppComponent. Proto je nutné přetížit tři základní virtuální funkce děděné třídy.

Metoda **prepareToPlay()** je zavolána minimálně jednou před začátkem zvukového procesu, pro inicializaci potřebných zvukových zdrojů, které jsou potřebné pro provádění operací v metodě getNextAudioBlock(). Také se zde získávají potřebné informace o očekávaném množství samplů a vzorkovací frekvenci používaného hardwaru.

```
void  prepareToPlay(int samplesPerBlockExpected, double sampleRate)
{
    transportSource.prepareToPlay(samplesPerBlockExpected, sampleRate);
    this->sampleRate = sampleRate;
}
```

Metoda **getNextAudioBlock()** je cyklicky opakovaná metoda pro dodávání dat zvukovému hardwaru v čase kdy je třeba. Tato činnost je spuštěna ve vláknu aplikace s největší prioritou vykonávání.

```
void getNextAudioBlock(const AudioSourceChannelInfo& bufferToFill)
{
     if (bufferToFill.buffer->getNumChannels() > 0) {
         const float* channelData = bufferToFill.buffer->
                                    getWritePointer(0, bufferToFill.startSample);

         for (int i = 0; i < bufferToFill.numSamples; ++i)
             pushNextSampleIntoFifo(channelData[i]);
     }

     if (modeCalibrate) {
         float level = (float)noiseVolumeSlider.getValue();

         if (bufferIndex == BUFFER_SIZE) {
             bufferIndex = 0;
         }

         for (int channel = 0; channel < bufferToFill.buffer->
                                              getNumChannels(); ++channel) {
             float* const buffer = bufferToFill.buffer->
                           getWritePointer(channel, bufferToFill.startSample);
             for (int sample = 0; sample < bufferToFill.numSamples; ++sample) {
                 buffer[sample] = getNextPinkNoiseNumber(sample) * level;

             }
         }
         bufferIndex++;
     } else {
         if (readerSource == nullptr) {
             bufferToFill.clearActiveBufferRegion();
             return;
         }
         transportSource.getNextAudioBlock(bufferToFill);
     }
}
```

Metoda **releaseResources()** je párová k první zmíněné a slouží pro uvolnění používaných zvukových zdrojů po skončení provádění zvukového procesu.

```
void  releaseResources()
{
    transportSource.releaseResources();
}
```

9.8.3 Použité třídy

Pro implementaci kromě šablony projektu bylo třeba umožnit další funkce. Proto byly použité další třídy. Některé JUCE třídy děděny celou třídou programované aplikace, jiné jako atributy.

Pro zajištění vypnutí při dokončení přehrávání souboru bylo nutné dědit ze třídy **ChangeListener**. Při tomto kroku je nutné přetížit virtuální metodu

changeListenerCallback(ChangeBroadcaster* source) a to tak, že při změně stavu zdroje zkontrolujete, zda hraje. Pokud ano, nechává se stav jako zapnutý a přehráváme dál. Pokud ne, zdroj se zastaví.

```
void  changeListenerCallback(ChangeBroadcaster* source)
{
    if (source == &transportSource) {
        if (transportSource.isPlaying())
            changeState(Playing);
        else
            changeState(Stopped);
    }
}
```

Další důležitá funkce, která byla nutná vyřešit, bylo cyklické opakování zobrazování frekvenčního spektra do okna aplikace. Pro zajištění této funkce byla zděděna funkčnost třídy **Timer**. Tato třída má jednu virtuální metodu timerCallback(). V této metodě se zkontroluje, zdali je kompletní dávka dat ze vstupu zvukové karty a následně se spustí kreslení spektra do obrázku a překreslení celého aplikačního okna. Pro zajištění optické plynulosti obrazu je třeba zobrazit 24 snímků za vteřinu. Pro potřeby aplikace však bylo z tohoto nároku upuštěno, protože překreslení obrázku spektra je nejdražší operací na výkon. Byl stanoven počet na 10 zobrazovaných snímků za sekundu. Tento interval pro potřeby aplikace postačí a přináší nám velmi dobrou úsporu výkonu.

```
void timerCallback()
{
    if (nextFFTBlockReady) {
        drawSpectrogram();
        nextFFTBlockReady = false;
        repaint();
    }
}
```

Poslední zděděnou třídou je **Button::Listener**, který nám umožnuje zachytávat události jednotlivých tlačítek a příslušně reagovat.

```
void  buttonClicked(Button* button)
{
    if (button == &openButton)  openButtonClicked();
    if (button == &playButton)  playButtonClicked();
    if (button == &stopButton)  stopButtonClicked();
    if (button == &modeButton)  modeButtonClicked();
    if (button == &saveSettingsButton)  saveButtonClicked();
    if (button == &openSettingsButton)
            openSettingButtonClicked();
}
```

Nepostradatelnou vlastní implementovanou třídou je třída **PinkNumber**. Jedná se o třídu, která má za úkol převést bílý šum na růžový. Pro tuto operaci využívá Paul Kellet's economy method (7). Byla použita právě tato metoda, z důvodu nízké náročnosti na hardware, avšak při

optimalizování kódu bylo neustále generování šumu stejně odstraněno a tedy vliv na výkon není zřejmý. Místo toho byl použit předgenerovaný buffer, ze kterého již pouze čteme hodnoty, což vedlo ke zrychlení celé aplikace. Následující ukázka kódu zobrazuje metodu pro filtrování bílého vstupního vzorku na výstupní růžový.

```
float PinkNumber::GetNextValue(float white)
{
    b0 = 0.99765f * b0 + white * 0.0990460f;
    b1 = 0.96300f * b1 + white * 0.2965164f;
    b2 = 0.57000f * b2 + white * 1.0526913f;
    float pink = b0 + b1 + b2 + white * 0.1848f;
    return pink;
}
```

Další důležitou třídou je **FFT**. Jak zkratka napovídá, jedná se třídu založené na Rychlé Fourierovy transformaci (5.1.1). Tato třída implementuje základní využití FFT. V aplikaci je využita pro převedení pole s hodnotami vstupního zvukového signálu na pole, které zobrazuje hlasitost jednotlivých frekvenčních pásem. V našem případě je použita velikost pole o 1024 hodnotách, což dobře stačí pro zobrazení základních problematických frekvencí zvukového vstupu.

Pro optimalizaci potřebnou třídou je **SixBandEqualizer**, která poskytuje šesti pásmový grafický ekvalizér. Tato třída je založená třídách IIRFilter a IIRCoeficient frameworku JUCE. Tyto třídy umožnují vytvořit zvukový filtr, který je schopen pracovat v režimech high-pass a low-pass filtr, high-pass shelf a low-pass shelf filtr nebo zde použitý peak filtr, kde je možné nastavit centrální frekvenci, faktor Q, pro šířku upravovaného frekvenčního pásma a úpravu hlasitosti, která může být jak kladná, tak záporná. Následný kód slouží pro aplikování filtru na výstupní tok dat.

```
float *dataLeft = bufferToFill.buffer->
                  getWritePointer(0, bufferToFill.startSample);
float *dataRight = bufferToFill.buffer->
                  getWritePointer(1, bufferToFill.startSample);
int nSamples = bufferToFill.numSamples;

getEqValues();
eq.setCoeficient(eqSettings,sampleRate);
eq.processMethod(dataLeft, dataRight, nSamples);
```

9.8.4 Implementace zobrazování

Část pro zobrazování je implementována pomocí dvou objektů Image. První je v kódu nazván spectrogramImage a je částí, kde se v určené periodě mění zobrazovaný obsah. Druhý je jen legenda a v implementaci je pojmenován jako descriptionImage.

Zobrazování jednotlivých frekvenčních pásem je realizováno kreslením obdélníků za pomoci třídy Graphics. Velikost jednotlivých obdélníků je dána absolutně dle hlasitosti pásma ve vstupních datech. Pro možnost tuto velikost změnit je zde implementován ovladač Spectrum gain. Tento ovladač nepracuje s ničím jiným, než s velikostí vizualizace.

```
if ((spectrogramImage.getHeight() - avarageLevel) <
                                                    DRAWABLE_LIMIT) {
      drawBandwidth(bandwidthOutset,
                        DRAWABLE_LIMIT,
                        widthOfBandwidth,
                        spectrogramImage.getHeight(),
                        g,
                        stateOfVolumeIsHigher);
} else {
      size_t y0 = (size_t)round(spectrogramImage.getHeight()
                                                        - avarageLevel);
      drawBandwidth(bandwidthOutset,
                        y0,
                        widthOfBandwidth,
                        spectrogramImage.getHeight(),
                        g,
                        stateOfVolumeIsHigher);
}
```

Při návrhu části pro zobrazení zvukového spektra bylo třeba vymyslet, jaké pásma budou zobrazována. Problém je zde skutečnost, že oktáva se rovná dvojnásobku frekvence aktuální. Pokud použijeme zobrazování po oktávách, budou basové frekvence obsaženy pouze v jednou až dvou sloupečkách, stejně jako nižší středy. Toto by způsobilo jen velmi omezenou použitelnost výsledné aplikace, vzhledem k tomu, že bychom se pohybovali téměř neustále za hranicí slyšitelného spektra, pro většinu populace.

Bylo tedy použito řešení absolutně definovaných frekvencí zobrazovaných v grafickém výstupu. Pro zobrazení se počítají průměrné hodnoty hlasitostí hlavní frekvence a jejího okolí. K manipulaci se zvukem je použito pole o velikosti 1024 položek. Frekvence na jednotlivých indexech jsou určeny poměrem k vzorkovací frekvenci systému. Interval pro výpočet průměrné hodnoty výkonu frekvenčního pásma s centrální frekvencí na n-tém indexu poté bude $\left(n + \frac{n}{4}\,; n - \frac{n}{4}\right)$.

```
void setRangeOfBandwidths()
{
  size_t bandHalfWidth[17];
  for (size_t i = 0; i < 17; i++) {
      bandHalfWidth[i] = (size_t)round(bandFrequency[i] / 4);
    }

    int x = 0;
    for (size_t i = 0; i < 17; i++) {
        range[x] = (size_t)round((bandFrequency[i] -
                                        bandHalfWidth[i]) / 22);
        range[x + 1] = (size_t)round((bandFrequency[i] +
                                        bandHalfWidth[i]) / 22);
        x += 2;
    }
}
```

9.8.5 Způsob implementace zvukové diagnostiky

Pro možnost optimalizace bylo nutné vymyslet mechanizmus, který bude indikovat, zda optimalizujeme správně nebo nikoliv. Jak již bylo zmíněno, pro zvukový výstup bylo z důvodu optimalizace zavedeno dvou rozměrné pole, které slouží jako buffer růžového šumu. Nevyřešenou otázkou však zůstalo, s čím porovnávat získané hodnoty z mikrofonního vstupu o aktuální situaci zvukového spektra v auditoriu.

```
void  MainContentComponent::fillPinkNoiseBuffer()
{
    for (size_t i = 0; i < BUFFER_SIZE; i++) {
        for (size_t j = 0; j < fftSize; j++) {
            pinkNoiseBuffer[i][j] =
pinkNumber.GetNextValue(random.nextFloat() * 2.0f - 1.0f);
        }
        pinkNumber.RefreshParameters();
    }
}
```

Pro tento problém byly navrženy dvě metody. První se zakládala na matematické výpočtu průběhu funkce, zatímco druhá využívala průměrné hodnoty zvukového bufferu, který již v aplikaci používáme místo neustálého generování nových hodnot. Pro výslednou aplikace byla nakonec použita metoda průměru několika vzorků šumu. Z důvodu dalšího snížení náročnosti na výkon stroje, se počet vzorků pro získání průměrných hodnot snižoval na, co možná, nejmenší číslo. Stabilních výsledků bylo dosahováno s minimálním množství 5 polí s hodnotami šumu.

Tedy pro získání vzorových dat růžového šumu je vygenerováno pět polí s hodnotami šumu, na které je použitá FFT pro získání hodnot hlasitosti jednotlivých frekvencí. Z tohoto pole je dále vypočítán průměr šestnácti hodnot zvolených frekvenčních pásem a tyto hodnoty jsou zapsány do pole s názvem pinkNoiseStandard. Je zde uložený stejný počet hodnot, jako je

zobrazovaných pásem. Pro výpočet průměrů je použit stejný princip jako při zobrazování do obrázku zvukového spektra.

```
void makePinkNoiseStandard()
{
    float counter = 0.0f;
    size_t min = 0;
    size_t max = 1;
    size_t bandwidthOutset = 0;
    for (size_t x = 0; x < AUDIO_BANDWIDTH_NUMBER; x++) {
            for (size_t i = 0; i < 5; i++)
            {
                  counter =
            countBandwidthAvarageLevel(fftPinkNoiseStandard[i], min, max);
            }
        pinkNoiseStandard[x] = counter / 5 * 0.75f;
        bandwidthOutset += widthOfBandwidth;
        min += 2;
        max += 2;
    }
}
```

Před grafickým zobrazením jsou hodnoty naměřené porovnány se vzorovými a na základě výsledku je daná část zvukového spektra zobrazena zelenou barvou při nedosažení stejné hlasitosti pásma, nebo červenou při přesáhnutí. Proto se při zvukové optimalizace snažíme, aby zobrazované obdélníky blikali dvěma barvami. Znamená to totiž, že se pohybujeme v blízkosti požadovaných hodnot.

9.9 Kompilace na platformě Raspberry Pi

Pro vývoj aplikace zvoleným způsobem bylo třeba zajistit compilaci aplikace na výsledné zvolené platformě Raspberry Pi. Je nutné stáhnout upravenou verzi JUCE pro Linux na disk paměťové medium Raspberry, tedy SD kartu. Následně je nutné zkopírovat zdrojové kódy vyvíjené aplikace na Raspberry. Poté už stačí jen zkompilovat a aplikace poběží na cílové platformě. Což není možné bez doinstalovaných knihoven. Tuto překážku lze jednoduše vyřešit pomocí nástroje apt-get. Následný příkaz nám veškeré potřebné knihovny doinstaluje.

```
sudo apt-get -y install freeglut3-dev libasound2-dev
libfreetype6-dev libjack-dev libx11-dev libxcomposite-dev
libxcursor-dev libxinerama-dev mesa-common-dev
```

Po doinstalování knihoven je ještě třeba mírně upravit makefile naší aplikace. Jedná se o upravení jednoho řádku z důvodu použité ARM architektury procesoru v zařízení Raspberry Pi. Jedná se o následující část:

```
ifeq ($(TARGET_ARCH),)
  TARGET_ARCH := -march=native
Endif
```

Je nutno přepsat prostřední řádek na správnou architekturu procesoru. V případě vyvíjeného zařízení se jedná armv6. Tedy to bude vypadat takto.

```
TARGET_ARCH := -march=armv6
```

Po doinstalování a úpravě je třeba se v terminalu přemístit do složky projektu a zde se dostat do podsložky `Builds/Linux`. Nyní je nutné provést kompilaci pomocí příkazu make. Po kompilaci se v této složce vytvoří podsložka `Builds/Linux/build`, ve které se již nachází spustitelný soubor.

9.10 Shrnutí praktické části

Výsledné zařízení je velmi malé a je možné jej ovládat pomocí dotykového displeje. K možnosti přehrávání a kalibrace je nutné použít externí zvukové karty, která musí podporovat USB rozhraní a mikrofon. Další možností je rozšířit Raspberry Pi o zvukovou kartu, kterou lze přidat jako periferii.

Pro úpravu výstupního signálu je implementován grafický ekvalizér, přičemž jeho nastavení je možné ukládat a zpětně načítat, což se hodí při opakovaném použití ve stejném prostoru a stejném zvukovém systému.

Zařízení bylo otestováno na několika sestavách s uspokojivými výsledky pro přehrávání reprodukované hudby. Při využití daného zařízení s živou kapelou nastal problém s vysokou latencí systému, což mělo za následek zvýraznění přeslechů z podia jejich posunutí v čase. Tedy vznikal nežádoucí efekt zvukové smyčky.

ZÁVĚR

Cílem mé práce bylo využít platformu Raspberry Pi pro vývoj zařízení poskytující možnost optimalizace výstupu zvukových systémů.

V první části bylo seznámeno se s platformou Raspberry Pi. Zmíněny byly především jednotlivé generace a jejich hardwarové možnosti. Okrajově poté možné instalace operačních systémů optimalizovaných pro Raspberry Pi. Dále zde jsou popsány principy digitálního zpracování signálu, Fourierovy analýzy a základní možností implementace FFT. Nakonec jsou představeny barvy šumů a jejich možné využití v praxi.

Druhá část je potom věnována především implementaci multiplatformní aplikaci pro práci se zvukem s použitím frameworku JUCE. Je zde seznámeno s návrhem aplikace a postupem implementace, samotným frameworkem a kompilaci projektu JUCE pod výslednou platformou Raspberry Pi.

Výsledná multiplatformní aplikace nachází široké užití jak na platformě Raspberry Pi, tak osobních počítačích. Dovoluje kompenzovat akustické vlastnosti reproduktorů a poslechových prostor. Kompletní systém je poté možné použít jako spektrální analyzér pro prvotní nastavení zvukového aparátu na koncertech živých seskupení, průchozí ekvalizér při přehrávání reprodukované hudby nebo jako přehrávač hudby se zabudovaným modulem pro úpravu frekvenční charakteristiky signálu.

Na základech této práce by bylo možné implementovat vlastní audio/video přehrávač, který by poskytoval modul pro optimalizaci zvukového výstupu. Při této variantě by bylo dobré optimalizaci plně automatizovat a navrhnout i komplexnější řešení, které by více pracovalo s akustikou celkového prostoru poslechu. Výsledné zařízení by bylo ideální volbou například pro domácí kina.

POUŽITÁ LITERATURA

2013 RoEduNet International Conference 12th Edition: Networking in Education and Research. USA: Curran Associates, Inc., 2014. ISSN 9781479926015.

ARM Cortex-A Series Programmer's Guide for ARMv8-A. *ARM News & Events* [online]. ©1995-2016 [cit. 2016-04-26]. Dostupné z: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.den0024a/CHDIHCJE.html

RICHARDSON, Matt. Beer and wine fridge of awesomeness. In: *Raspberry Pi Foundation* [online]. 2015 [cit. 2016-04-26]. Dostupné z: https://www.raspberrypi.org/blog/beer-and-wine-fridge-of-awesomeness/

Blue Noise. *Techopedia* [online]. 2016 [cit. 2016-04-26]. Dostupné z: https://www.techopedia.com/definition/27896/blue-noise

BROSE, Moses. Broadcom BCM2835 SoC has the most powerful mobile GPU in the world? In: *GrandMAX* [online]. ©2010-2012 [cit. 2016-04-26]. Dostupné z: https://web.archive.org/web/20120413184701/http://www.grandmax.net/2012/01/broadcom-bcm2835-soc-has-powerful.html

JACKSON, John David. *Classical electrodynamics*. 3rd ed. New York, N.Y.: Wiley, c1999, xxi, 808 p. ISBN 047130932X.

Colors of noise. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2016-04-30]. Dostupné z: https://en.wikipedia.org/wiki/Colors_of_noise

FLYNN, Michael J. *Computer architecture: pipelined and parallel processor design*. Boston, MA: Jones and Bartlett, c1995, xix, 788 p. ISBN 0867202041.

BRISTOW-JOHNSON, Robert. Cookbook formulae for audio EQ biquad filter coefficients. In: *Music-DSP Source Code Archive* [online]. [cit. 2016-04-26]. Dostupné z: http://www.musicdsp.org/files/Audio-EQ-Cookbook.txt

REICHL, Jaroslav a Martin VŠETIČKA. Digitalizace analogového signálu. In: *Encyklopedie fyziky* [online]. ©2006-2016 [cit. 2016-04-26]. Dostupné z: http://fyzika.jreichl.com/main.article/view/1355-digitalizace-analogoveho-signalu

WHITTLE, Robin. DSP generation of Pink (1/f) Noise. In: *First Principles* [online]. 1999 [cit. 2016-04-26]. Dostupné z: http://www.firstpr.com.au/dsp/pink-noise/

MARWEDEL, Peter (ed.). *Embedded system design*. 2nd ed. Dordrecht: Springer, 2011. ISBN 9789400702578.

FAQS. *Raspberry Pi Foundation* [online]. 2016 [cit. 2016-04-26]. Dostupné z: https://www.raspberrypi.org/help/faqs/#introWhatIs

GRAZIANO, Dan. First boot with the Raspberry Pi. In: *CNET* [online]. 2014 [cit. 2016-04-25]. Dostupné z: http://www.cnet.com/how-to/first-boot-with-the-raspberry-pi/

KLÍČ, Alois, Miroslava DUBCOVÁ a Karel VOLKA. *Fourierova transformace: S příklady z infračervené spektroskopie*. Vyd. 3. Praha: VŠCHT, 2002, 196 s. ISBN 8070804785.

Gray Noise. *Techopedia* [online]. 2016 [cit. 2016-04-26]. Dostupné z: https://www.techopedia.com/definition/27898/gray-noise

Installing operating system images using Windows. *Raspberry Pi Foundation* [online]. 2016 [cit. 2016-04-26]. Dostupné z: https://www.raspberrypi.org/documentation/installation/installing-images/windows.md

MOORER, James Andel'son. *On the segmentation and analysis of continuous*. Standford, California, USA, 1975.

CARTER, Bruce. *Op amps for everyone*. Fourth edition. Oxford, UK ; Waltham, MA: Elsevier/Newnes, 2013, xxii, 281 pages. ISBN 0123914957.

Pink Noise. *Techopedia* [online]. 2016 [cit. 2016-04-26]. Dostupné z: https://www.techopedia.com/definition/14996/pink-noise

Převodníky analogových a číslicových signálů. *Fyzikální sekce Matematicko-fyzikální fakulty UK* [online]. 2014 [cit. 2016-04-26]. Dostupné z: http://physics.mff.cuni.cz/kfpp/skripta/elektronika/kap8/pevodnky.html

Raspberry Pi 2: Which OS is best? *Element14* [online]. ©2009-2016 [cit. 2016-04-26]. Dostupné z: https://www.element14.com/community/polls/2103

UPTON, Eben. Raspberry Pi 3 on sale now at $35. In: *Raspberry Pi Foundation* [online]. 2016 [cit. 2016-04-26]. Dostupné z: https://www.raspberrypi.org/blog/raspberry-pi-3-on-sale/

Raspberry Pi Foundation [online]. 2014 [cit. 2016-04-26]. Dostupné z: https://www.raspberrypi.org

REICHL, Jaroslav a Martin VŠETIČKA. Složené tóny a Fourierova transformace. In: *Encyklopedie fyziky* [online]. ©2006-2016 [cit. 2016-04-26]. Dostupné z: http://fyzika.jreichl.com/main.article/view/187-slozene-tony-a-fourierova-transformace

HOLČÍK, Lukáš. *Spektrální analýza hudební skladby*. Brno, 2009. Diplomová práce. Masarykova univerzita. Vedoucí práce MgA. Rudolf Růžička.

MEIER, Florian, Marco FINK a Udo ZOLZER. *The JamBerry: A Stand-Alone Device for Networked Music Performance Based on the Raspberry Pi*. Hamburg, 2014.

BARTHES, Roland. *The responsibility of forms: critical essays on music, art, and representation*. Překlad Richard Howard. Berkeley: University of California Press, 1991, viii, 312 s. ISBN 0520072383.

TV Freak [online]. ©1998-2015 [cit. 2016-05-04]. ISSN 1802-1328. Dostupné z: http://www.tvfreak.cz

VANDERPLAS, Jake. Understanding the FFT Algorithm. In: *Pythonic Perambulations: Musings and ramblings through the world of Python and beyond* [online]. ©2012-2015 [cit. 2016-04-30]. Dostupné z: https://jakevdp.github.io/blog/2013/08/28/understanding-the-fft/

Violet Noise. *Techopedia* [online]. 2016 [cit. 2016-04-26]. Dostupné z: https://www.techopedia.com/definition/27897/violet-noise

Virtualization Extensions. *ARM News & Events* [online]. ©1995-2016 [cit. 2016-04-26]. Dostupné z: https://www.arm.com/products/processors/technologies/virtualization-extensions.php

REICHL, Jaroslav a Martin VŠETIČKA. Vzorkování signálu. In: *Encyklopedie fyziky* [online]. ©2006-2016 [cit. 2016-04-26]. Dostupné z: http://fyzika.jreichl.com/main.article/view/1356-vzorkovani-signalu

ROUSE, Margaret. What is ARM processor? In: *TechTarget* [online]. ©1999-2016 [cit. 2016-04-26]. Dostupné z: http://whatis.techtarget.com/definition/ARM-processor

ADAMSKI, Krzysztof. What is the optimum split of main versus GPU memory? In: *Raspberry Pi Stack Exchange* [online]. 2016 [cit. 2016-04-26]. Dostupné z:

www.raspberrypi.stackexchange.com/questions/673/what-is-the-optimum-split-of-main-versus-gpu-memory

White Noise. *Merriam-Webster* [online]. 2015 [cit. 2016-04-26]. Dostupné z: http://www.merriam-webster.com/dictionary/white%20noise

HEATH, Nick. Windows 10 on the Raspberry Pi: What you need to know. In: *TechRepublic* [online]. 2016 [cit. 2016-04-26]. Dostupné z: http://www.techrepublic.com/article/windows-10-on-the-raspberry-pi-what-you-need-to-know/

COLLINSON, Andy. Winscope. In: *Circuit Exchange International* [online]. 2016 [cit. 2016-04-25]. Dostupné z: http://www.zen22142.zen.co.uk/Prac/winscope.htm

PŘÍLOHY

Příloha B – *UML diagram tříd*