UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2024 Jakub Prášek

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Jednoduchý frekvenční analyzátor na bázi Raspberry PI

Bakalářská práce

2024 Jakub Prášek

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jakub Prášek**
Osobní číslo:	**I21064**
Studijní program:	**B0714A150008 Automatizace**
Téma práce:	**Jednoduchý frekvenční analyzátor na bázi Raspberry PI**
Zadávající katedra:	**Katedra řízení procesů**

Zásady pro vypracování

Cílem práce je sestrojení zařízení, které bude provádět základní frekvenční analýzu měřeného signálu pomocí DFT, s využitím počítače Raspberry PI a vhodného rozšiřujícího modulu s A/D převodníkem. Bude navrženo jednoduché rozhraní, které umožní měřit napěťové signály v rozsahu 0-5V s ochranou proti přepětí, případně umožní rozsah zvolit. Dále bude v jazyku C/C++ v Raspberry PI napsán program, který bude provádět měření signálu a jeho frekvenční rozklad pomocí DFT, s využitím definičních vztahů. Výsledky budou vypisovány na obrazovce a ukládány do textového souboru, odkud je bude možné zobrazit v grafické podobě např. tabulkovým procesorem. V práci bude diskutována výpočetní náročnost v závislosti na zvolených parametrech. Testovací signály pro ověření výsledků mohou být generovány stejným zařízením.

Rozsah pracovní zprávy: **cca 40 stran**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

DAVÍDEK, V., SOVKA P. *Číslicové zpracování signálů a implementace*. Praha: ČVUT 1996.
TRANTER, J. *How to Control GPIO Hardware from C or C++* [online]. Dostupné z: https://www.ics.com/blog/how-control-gpio-hardware-c-or-c .
LÁNÍČEK, R. *Elektronika, obvody, součástky, děje*, Praha: BEN – technická literatura, 1998.

Vedoucí bakalářské práce: **doc. Ing. Jan Cvejn, Ph.D.**
Katedra řízení procesů

Datum zadání bakalářské práce: **15. prosince 2023**
Termín odevzdání bakalářské práce: **10. května 2024**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Daniel Honc, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 12. ledna 2024

Prohlašuji:

Práci s názvem Jednoduchý frekvenční analyzátor na bázi Raspberry PI jsem 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., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, 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.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 15. 05. 2024

Jakub Prášek v.r.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu práce panu doc. Ing. Janu Cvejnovi, Ph.D. za odbornou pomoc a připomínky, které mi pomohli se zpracováním práce. Dále bych chtěl poděkovat panu Ing. Pavlu Rozsívalovi za pomoc při shánění součástek a poskytnutí podmínek pro spájení plošného spoje. A v neposlední řadě bych chtěl poděkovat rodině za podporu při vytváření práce.

ANOTACE

Cílem práce je sestrojení zařízení, které provádí základní frekvenční analýzu měřeného napěťového signálu v rozsahu 0-5V pomocí diskrétní Fourierovy transformace. K tomuto účelu je využito počítače Raspberry PI a rozšiřujícího modulu ADC-DAC Pi Zero. Program realizující měření signálu a jeho frekvenční rozklad v Raspberry PI je napsán v jazyku C++.

KLÍČOVÁ SLOVA

Frekvenční analýza, DFT, Raspberry PI, C++, A/D převodník

TITLE

Simple frequency analyzer based on Raspberry PI

ANNOTATION

The purpose of this work is to build a device that performs a basic frequency analysis of a measured voltage signal in the range of 0-5V using a discrete Fourier transform. The Raspberry PI computer and the Pi Zero ADC-DAC expansion module are used for this purpose. The program implementing the signal measurement and its frequency decomposition in Raspberry PI is written in C++ language.

KEYWORDS

Frequency analysis, DFT, Raspberry PI, C++, A/D converter

OBSAH

SEZNAM ILUSTRACÍ A TABULEK

SEZNAM ZKRATEK A ZNAČEK

RPI – Raspberry PI

DFT – diskrétní Fourierova transformace

IDFT– inverzní diskrétní Fourierova transformace

FFT – fast Fourier transform (rychlá Fourierova transformace)

FT – Fourierova transformace

ARM – Advanced RISC Machines

SDRAM – Synchronous Dynamic Random Access Memory

V/V – vstupně výstupní

USB – Universal serial bus

HDMI – High-Definition Multimedia Interface

GPIO – General-purpose input/output

SPI – Serial Peripheral Interface

I2C – Inter-IC

UART – Universal asynchronous receiver-transmitter

OS – operační systém

D/A – Digitálně-analogový

A/D – Analogově-digitální

NAS – Network Attached Storage

ÚVOD

Frekvenční analýza je důležitý nástroj pro charakterizaci elektrických signálů v různých aplikacích elektrotechniky. Umožňuje rozklad signálu na jeho jednotlivé harmonické složky, čímž poskytuje informace o jejich frekvenci, amplitudě a fázovém posunu. Tyto informace jsou nezbytné při diagnostice a analýze poruch a anomálií v elektrických obvodech a zařízeních, jako jsou šumy, interferenční jevy nebo harmonické zkreslení. Výsledkem je efektivnější diagnostika a údržba elektrických zařízení, což přispívá ke zlepšení jejich spolehlivosti a výkonnosti. Frekvenční analýza má i aplikace v řadě dalších oblastí. Např. ve strojírenství se využívá pro detekci poruch.

Cílem práce je sestrojení zařízení, které provádí základní frekvenční analýzu měřeného napěťového signálu v rozsahu 0-5V pomocí diskrétní Fourierovy transformace. Součástí práce je rovněž testování výpočetní výkonnosti zařízení Raspberry PI(RPI) pro zpracování signálu a provedení frekvenční analýzy. Frekvenční analýza vyžaduje výpočetně náročné operace s potenciálně velkými soubory dat, je proto nezbytné posouzení, zda je Raspberry PI pro tuto úlohu vhodné. Testy jsou zaměřeny na měření doby potřebné k provedení frekvenční analýzy různě velkých datových souborů. Tímto způsobem je v práci posouzena efektivita a vhodnost použití RPI pro tuto úlohu.

V této práci je využita pouze základní verze diskrétní Fourierovy transformace (DFT). Základní verze DFT je zvolena záměrně, protože její realizace je jednodušší ve srovnání s rychlou Fourierovou transformací (FFT), která je sice efektivnější z hlediska výpočetního času, ale zároveň využívá složitější algoritmickou implementaci. Tato volba dává práci možnost většího porozumění principu frekvenční analýzy. Také usnadňuje vývoj a ladění programového kódu.

1 TEORETICKÁ ČÁST

1.1 Diskrétní Fourierova transformace

Fourierova transformace (FT) je matematický nástroj pro analýzu signálů. FT převádí signál z časové oblasti do frekvenční oblasti. Tento algoritmus se používá pro rozklad signálu na jednotlivé harmonické složky. Výstupem FT je funkce s významem komplexní amplitudy frekvenční složky obsažené v signálu, závislá na frekvenci. Výsledkem FT sinusového signálu je Diracův impuls posunutý o hodnotu odpovídající frekvenci měřeného signálu. Pokud je signál složený z více harmonických signálů, bude těchto impulsů ve výsledné funkci více a budou mít různou amplitudu a frekvenci.

Diskrétní Fourierova transformace (DFT) je upravená pro použití pro diskrétní signál, který vznikne navzorkováním spojitého signálu. Výsledkem DFT je soubor záznamů o stejném počtu složek, jako počet vzorků signálu. Proto je tímto algoritmem možno rozlišit pouze tolik frekvencí, kolik je vzorků signálu. Tyto složky jsou reprezentovány komplexními čísly, kde každému záznamu odpovídá jedna frekvenční složka. Z každého záznamu je možné získat odpovídající amplitudu a fázový posuv. Původní signál lze získat sečtením jednotlivých harmonických složek. Příslušný vztah se nazývá inverzní diskrétní Fourierova transformace (IDFT).

Vztahy pro výpočet DFT a IDFT (O. Smith, 2007) jsou následující:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{\frac{-j2\pi nk}{N}} = \sum_{n=0}^{N-1} x(n)[cos\left(\frac{2\pi nk}{N}\right) - j\ sin\left(\frac{2\pi nk}{N}\right)] \quad (1)$$

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k) e^{\frac{j2\pi nk}{N}} = \frac{1}{N}\sum_{k=0}^{N-1} X(k)[cos\left(\frac{2\pi nk}{N}\right) + j\ sin\left(\frac{2\pi nk}{N}\right)] \quad (2)$$

kde

k - index DFT ve frekvenční oblasti, k=0,1,2,..,N-1

n – číslo vzorku, n=0,1,2,..,N-1

$X(k)$ – komplexní frekvenční spektrum

$x(n)$ – diskrétní vstupní signál

N – počet vzorků.

Ze vztahu pro výpočet DFT vyplývá, že se jedná o součet součinů všech vzorků vynásobených exponenciální funkcí danou koeficientem nk/N. Tento součet charakterizuje frekvenční složku $X(k)$.

Frekvenci k-té složky f_k lze vypočítat pomocí vztahu:

$$f_k = \frac{kf_s}{N},\ k=0,1,2,..,N-1 \tag{3}$$

kde

f_k – frekvence k-té složky (Hz)

f_s – vzorkovací frekvence (Hz)

N – počet vzorků.

Frekvence f_k odpovídá úhlové frekvenci ω_k, kterou lze vypočítat pomocí vztahu (O. Smith, 2007):

$$\omega_k = k\Omega = \frac{k2\pi}{NT},\ k=0,1,2,..,N-1 \tag{4}$$

kde

ω_k – úhlová frekvence k-té složky (rad/s)

Ω – úhlová vzorkovací frekvence (rad/s)

N – počet vzorků

T – vzorkovací perioda (s).

Algoritmus DFT je reprezentován dvěma vnořenými cykly o k a n opakováních. Uvnitř těchto cyklů je provedeno násobení vzorku $x(n)$ s exponenciální funkcí a sečtení těchto násobků. Tyto operace jsou provedeny N^2 krát. Protože počet frekvenčních složek k je stejný jako počet vzorků výpočetní náročnost odpovídá kvadratické funkci N.

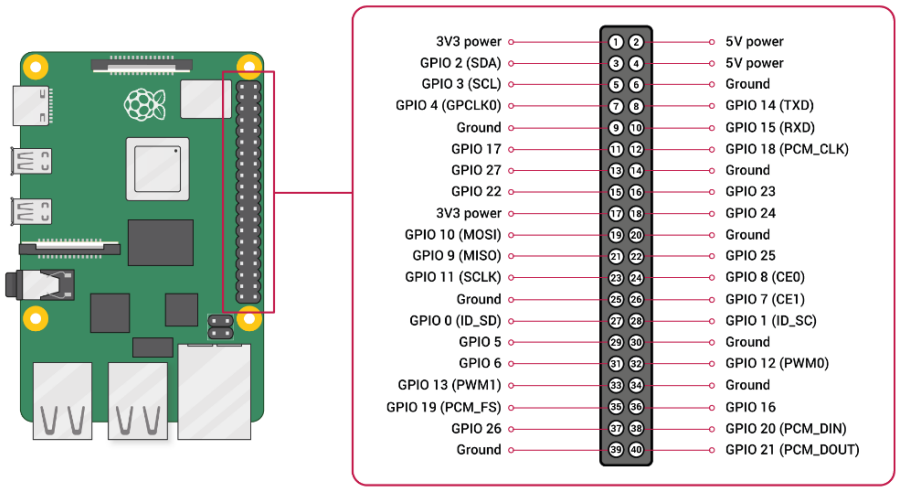
Rychlá Fourierova transformace (FFT) je efektivnější DFT z hlediska počtu operací, kterých neprovádí N^2, ale pouze $N \log N$. Pro FFT existuje několik různých algoritmů. Algoritmus Cooleyho a Tukeyho (Univerzita Palackého v Olomouci, cit. 2024), také nazývaný redukce času, využívá dělení signálu na dvě části a rekurzivního způsobu rozkladu. Je však potřeba, aby délka signálu byla mocnina 2.

1.2 Raspberry Pi

Raspberry Pi (RPI) je výkonný malý počítač, který je pro svou výkonnost, cenu a množství dostupných rozšiřujících modulů, využíván v řadě složitějších projektů. Konkrétně je využíván v automatizaci domácností, vzdálené komunikaci se zařízeními v rámci Internetu věcí (IoT), také je možné ho využít jako síťový server pro domácí úložiště. Za schopnost provádět složité výpočty a algoritmy vděčí 1.2 GHz 64-bitovému čtyřjádrovému procesoru ARM Cortex A53 a SDRAM paměti o velikosti 1GB.

RPI má řadu V/V periférií. Disponuje čtyřmi USB 2.0 porty na něž lze připojit klávesnice, myš případně externí pevné disky nebo flash disky pro přenos dat. HDMI port je

k dispozici pro připojení monitoru. V neposlední řadě je k dispozici GPIO (General Purpose Input/Output) rozhraní, které je možné nakonfigurovat pro práci s různými typy signálů nebo na něj lze připojit různé rozšiřující moduly. Obrázek 1 ilustruje GPIO rozhraní a rozložení periférií RPI.



Obrázek 1: GPIO rozhraní (Raspberry Pi Ltd, © 2012-2024)

Internetovou komunikaci zajišťuje Ethernet port nebo zabudovaná wifi. K dispozici je také možnost komunikace přes Bluetooth 4.1. Podporuje také sériovou komunikaci přes SPI (Serial Peripheral Interface), I2C (Inter-Integrated Circuit) a UART (Universal Asynchronous Receiver-Transmitter).

Operační systém (OS) RPI je uložen vždy na microSD kartě. Instalaci OS zajistí Raspberry Pi Imager. RPI je založeno na OS Linux. RPI disponuje vlastním OS - Raspberry Pi OS, který obsahuje širokou škálu softwaru, včetně nástrojů pro programování, práci s internetem, kancelářské aplikace a další. Je však možnost použít i další různé OS. Volba OS je zpravidla závislá na typu použití RPI. Zde je několik další příkladů OS a typu použití RPI: Volumio – přehrávání a streamování hudby, RetroPie – klasické herní konzole, LibreELEC – multimediální centrum a OpenMediaVault – síťové úložiště NAS (Network Attached Storage) (Smetanová, 2022)

1.3 Programování RPI

Programování pro Raspberry Pi nabízí širokou škálu možností a flexibilitu díky podpoře různých programovacích jazyků a knihoven pro práci s hardwarem. Mezi nejpoužívanější programovací jazyky patří Python a C++, které poskytují snadnou syntaxi a širokou podporu

komunity. Python je často preferovaný pro svou jednoduchost a rychlost vývoje, zatímco C++ nabízí lepší výkon a kontrolu nad hardwarem. Kromě těchto jazyků lze použít také další, jako je například Java nebo JavaScript. Pro práci s hardwarem, jako je komunikace s GPIO (General Purpose Input/Output) rozhraním, jsou k dispozici různé knihovny a frameworky, jako například WiringPi, RPi.GPIO nebo pigpio. Tyto knihovny umožňují snadnou manipulaci s GPIO rozhraním, čtení a zápis hodnot, řízení periferií a komunikaci s různými senzory a zařízeními připojenými k Raspberry Pi. Celkově je Raspberry Pi ideálním prostředím pro vývoj a experimentování díky podpoře různých programovacích jazyků a široké škále dostupných knihoven pro práci s hardwarem.

1.4 Programování RPI v jazyce C++

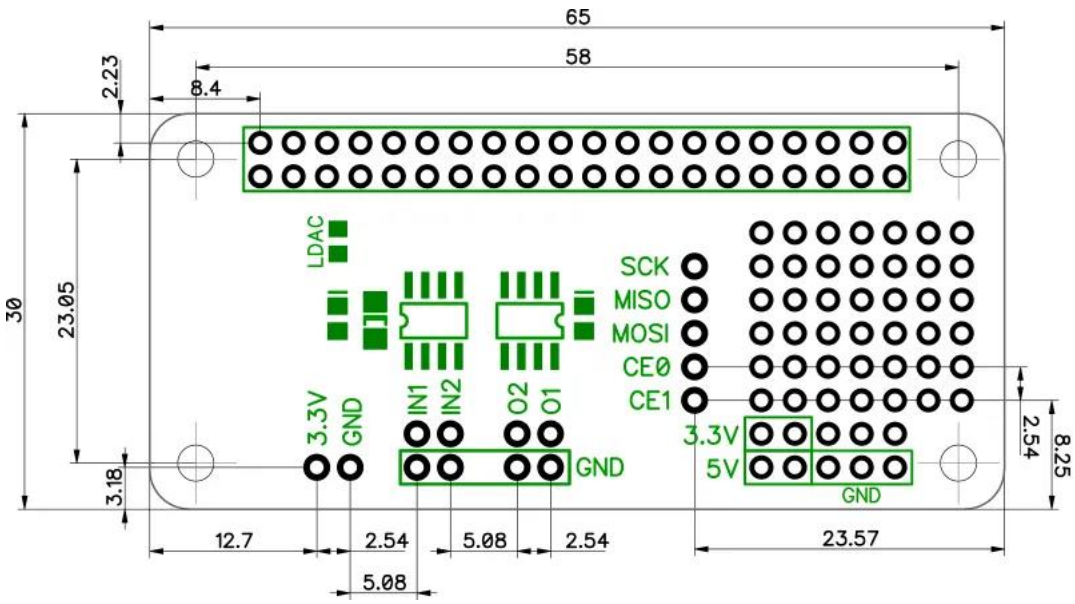
Při vývoji programů pro Raspberry Pi v jazyce C++ je důležité mít k dispozici vhodné nástroje pro psaní kódu, kompilaci, ladění a profilování. V prostředí Linuxu, které je běžně používáno s Raspberry Pi, jsou tyto nástroje součástí vývojového prostředí. Textový editor slouží k psaní kódu. K dispozici jsou standardní nástroje pro psaní textu jako nano, vim a emacs. Kompilátor překládá zdrojový kód do strojového kódu, který je pochopitelný pro Raspberry Pi. Linker je nástroj, který spojuje přeložené objektové soubory do spustitelného programu. Debugger umožňuje ladění programů a profiler slouží k analýze výkonu a identifikaci potenciálních úzkých míst v kódu. Pro ladění programů se standardně využívá utilita gdb a jako profiler gprof nebo perf. Kromě toho se často používá nástroj make pro automatizaci procesu kompilace a sestavení programu.

Pro pohodlnější tvorbu programů v různých jazycích je možné využít nástroj Geany, který umožňuje pohodlně psát kód, kompilovat ho a spouštět bez nutnosti opouštět editor. Tím poskytuje uživatelům jednoduchý a efektivní způsob vývoje aplikací pro Raspberry Pi v jazyce C++.

1.5 Modul ADC-DAC Pi Zero

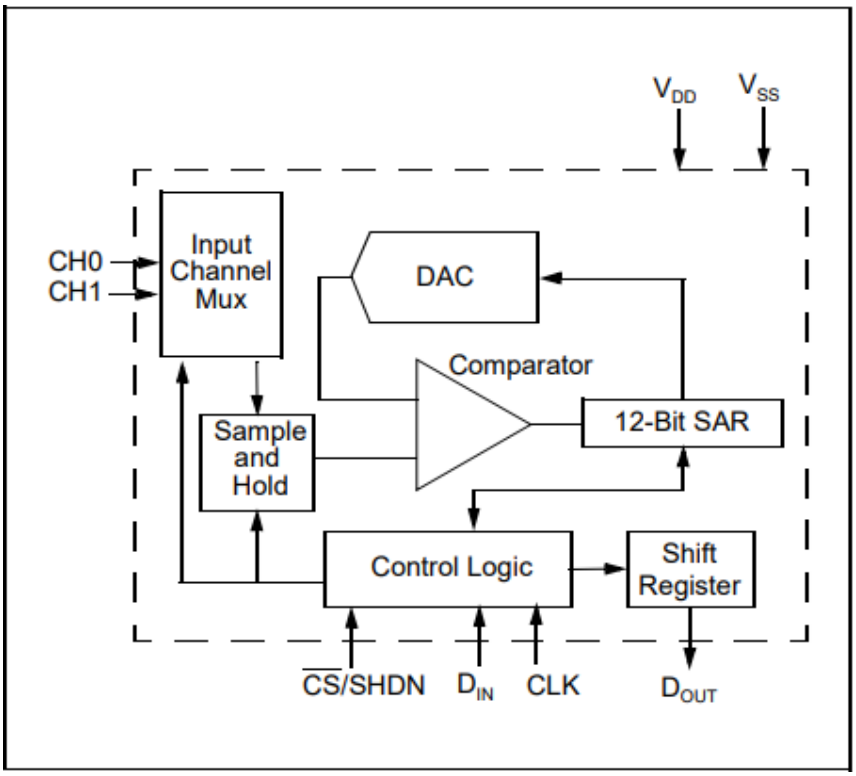
Rozšiřující modul ADC-DAC Pi Zero je rozšíření pro RPI, které se nasadí na GPIO konektor. Je to dvou kanálový dvanácti bitový D/A převodník (převodník digitálního na analogový signál) a A/D převodník (převodník analogového na digitální signál) převodník. Je plně kompatibilní s RPI 3B, který je použit v tomto projektu. Tento modul používá mikročip MCP3202 pro A/D převodník a mikročip MCP4822 pro D/A převodník. Komunikace s RPI probíhá přes sériovou SPI sběrnici. SPI je sběrnice typu master-slave, přenos je synchronní a probíhá na 4 pinech: MOSI, MISO, SCLK a CE0. Modul zvládne napětí do 3,3V. Maximální

vzorkovací frekvence udávaná výrobcem modulu je 17 500 vzorků za sekundu při použití C++ a RPI 4. Obrázek 2 zobrazuje mechanický výkres modulu.

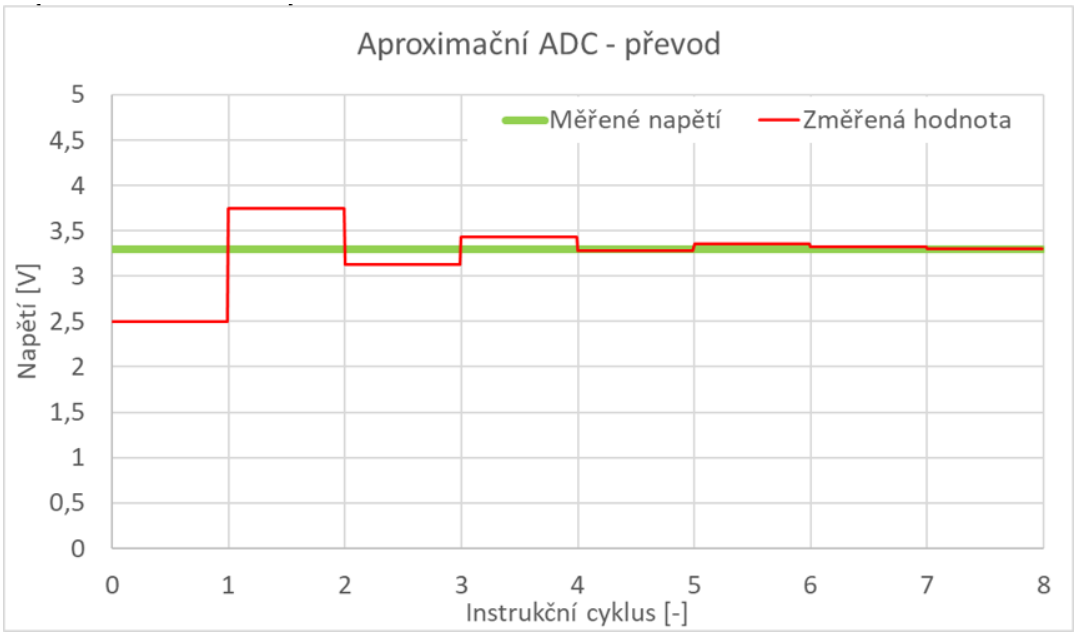


Obrázek 2: Mechanický výkres modulu ADC-DAC Pi Zero (ABElectronicsUK, © 2012-2024)

Mikročip MCP3202 – A/D převodník pracuje na principu postupné aproximace (successive approximation). Obrázek 4 tento princip ilustruje. Z dataheetu je patrno, že maximální vzorkovací frekvence na 5V je 100 000 vzorků za sekundu. SAR je registr postupné aproximace a pomocí nastavování bitů do 1 nebo 0 od nejvíce významného bitu (zprava) posílá signál do D/A převodníku. Signál z D/A převodníku se v komparátoru porovná se vstupním signálem a pokud je vstupní signál větší zůstane bit v SAR na 1, pokud je menší, nastaví se do 0. Takto projde převodník všech dvanáct bitů. Tento algoritmus je ekvivalentní metodě půlení intervalu známé z matematiky. Obrázek 3 zobrazuje blokový diagram mikročipu MCP3202.



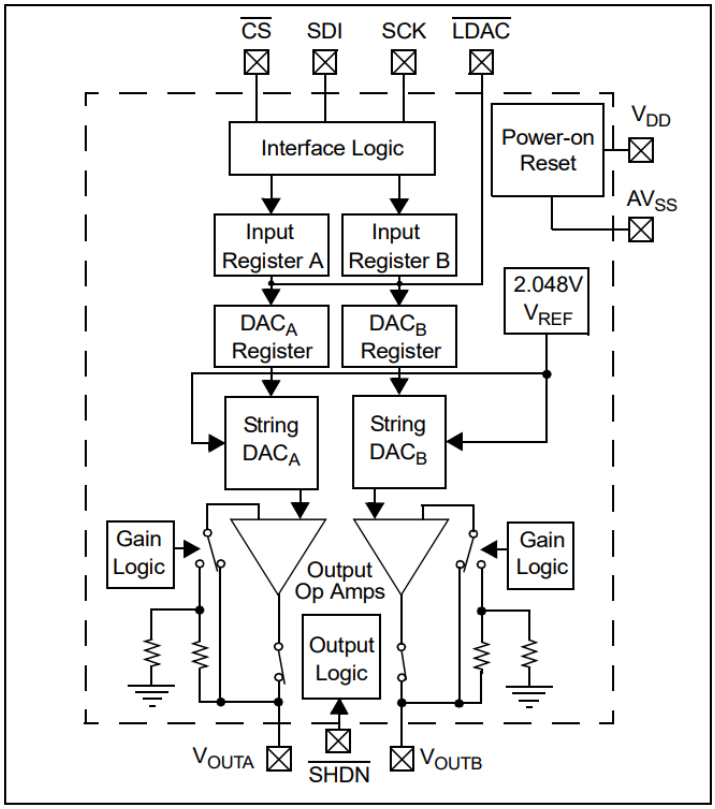
Obrázek 3: A/D převodník MCP3202 – blokový diagram (Microchip Technology Inc., 2006)



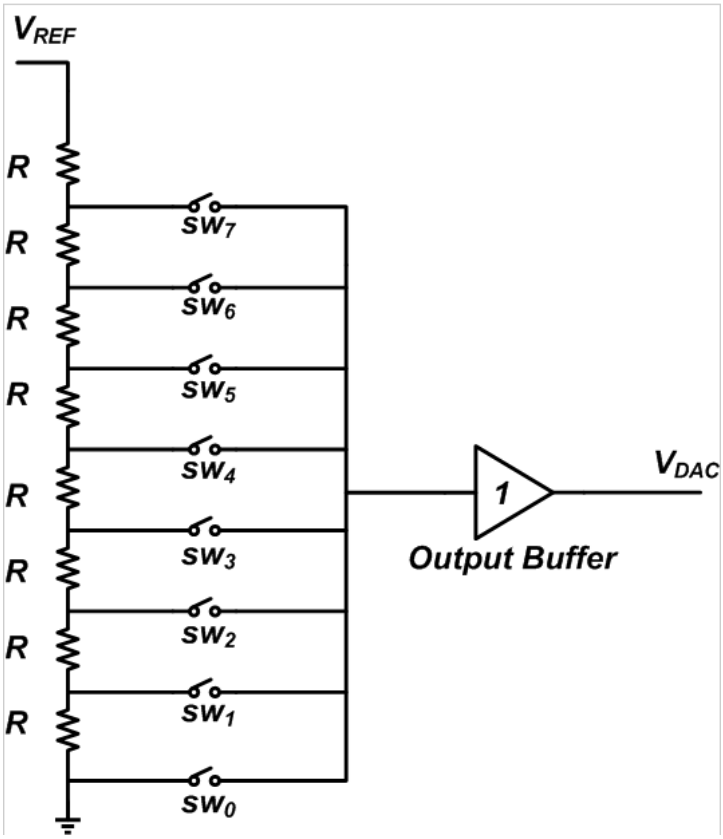
Obrázek 4: Princip aproximačního A/D převodníku (Rejfek, L. 2021)

Mikročip MCP4822 – D/A převodník používá odporový řetězcový D/A převodník (Kelvinův dělič), jehož princip ilustruje Obrázek 6. Jedná se o velký odporový dělič s 2^{12}

výstupy, které jsou pomocí spínačů nebo multiplexoru připojeny na výstup. Je tedy potřeba 2^{12} odporů. Úbytek napětí na jednom rezistoru odpovídá hodnotě jednoho kroku. Ten se vypočítá ze vztahu maximální napětí/2^{12}. Pro 3,3V je jeden krok 0.8mV. Obrázek 5 zobrazuje blokový diagram mikročipu MCP4822.



Obrázek 5: D/A převodník MCP4822 – blokový diagram (Microchip Technology Inc., 2005)



Obrázek 6: Princip řetězcového D/A převodníku (Steve, 2019).

2 PRAKTICKÁ ČÁST

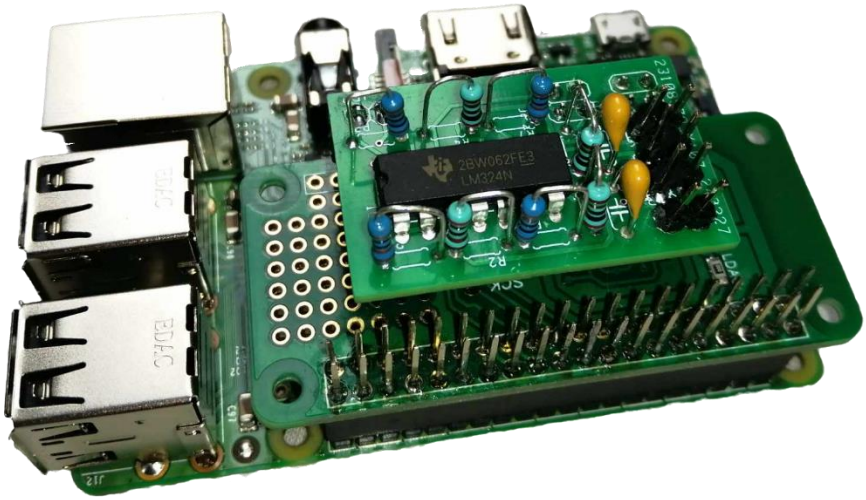
2.1 Elektrické zapojení a plošný spoj

Využitý modul ADC-DAC Pi Zero dokáže pracovat se signály pouze do 3,3V. Toto omezení je dáno jeho konstrukcí. Zařízení má umožňovat měření signálů v rozsahu 0-5V. Proto bylo potřeba navrhnout rozhraní, které přizpůsobí rozsah signálu a zaručí, že vstupní rozsah napětí převodníku nebude překročen a výstupní signál bude 5V.

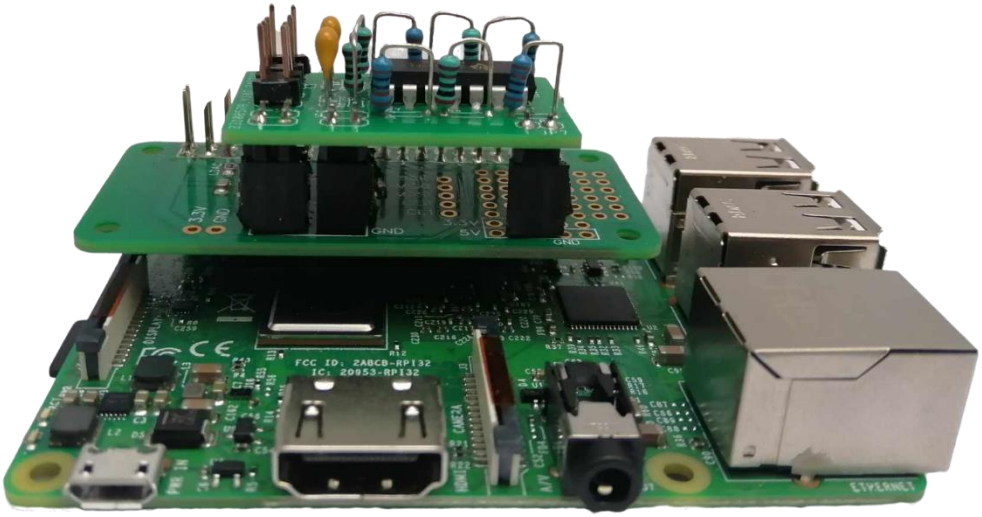
Pro realizaci rozhraní bylo zvoleno neinvertující zapojení s operačním zesilovačem, ilustrované na Obrázek 11, který ze své podstaty nemůže na svém výstupu vytvořit napětí větší než své napájecí. Pro výstupní signál bylo použito neinvertující zapojení se zesílením 1,5. Zajistí, že na výstupních pinech bude maximální napětí 4,95V, výpočty těchto hodnot jsou v rovnicích (5) a (6). Na vstupní piny bylo použito zapojení „opakovače“ v kombinaci s napěťovým děličem, ilustrované na Obrázek 12. Rezistory v napěťovém děliči jsou v poměru 1:2. Proto maximální vstupní napětí je 3.34 V, výpočet je proveden v rovnici (7).

Nejlepší podmínky pro realizaci tohoto zapojení poskytuje integrovaný obvod LM324N. Tento obvod dovoluje napájení v rozmezí 3-30V single (kladný pin a GND pin), ±1,5 – ±15V dual (kladný pin a záporný pin). Toto zapojení používá pro napájení 5V pin z ADC-DAC modulu. Obrázek 9 obsahuje elektrické zapojení plošného spoje. Obrázek 10 zobrazuje schéma zapojení vytvořené v programu EAGLE.

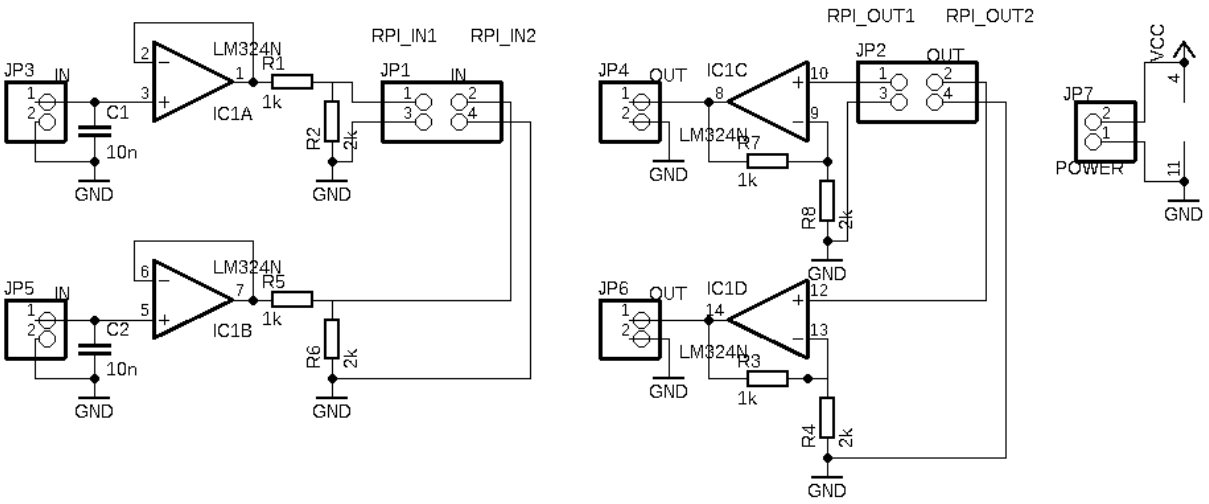
Plošný spoj byl tvořen na tento konkrétní rozšiřující modul. Bylo potřeba najít přesné rozměry pro umístění pájecích plošek u vstupu a výstupu z převodníku a u napájení spoje. Rozměry poskytl Obrázek 2: Mechanický výkres modulu ADC-DAC Pi Zero. Plošný spoj je navržen tak, aby se pouze nasadil na Modul ADC-DAC Pi Zero bez potřeby pevného spoje. Obrázek 7 a Obrázek 8 jsou fotografie, které ilustrují, jak zařízení vypadá.



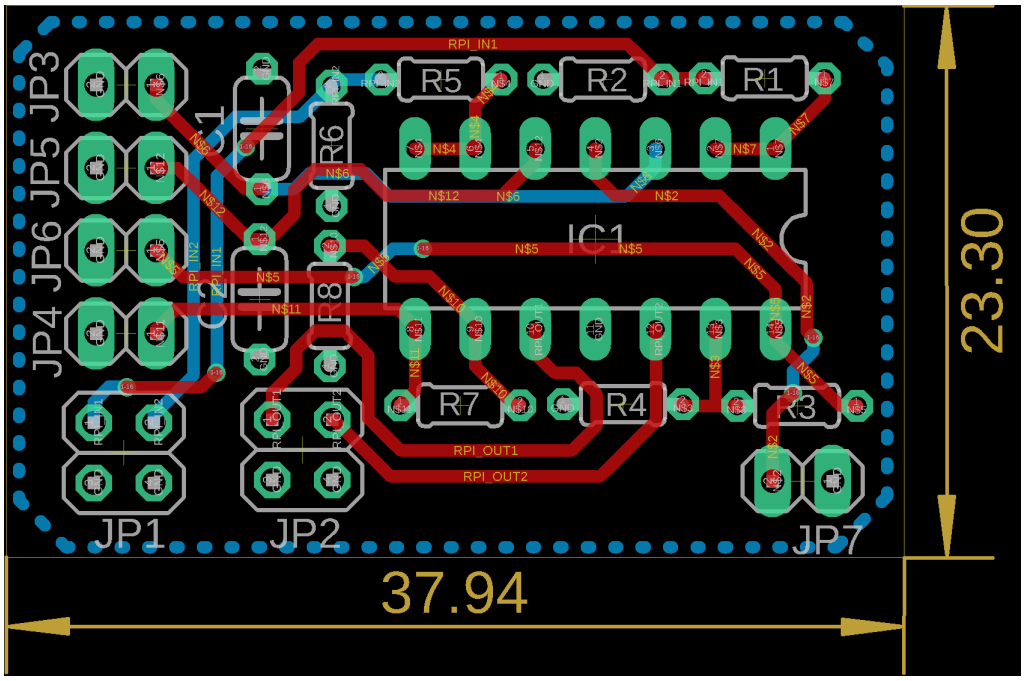
Obrázek 7: Fotografie zařízení – pohled z hora



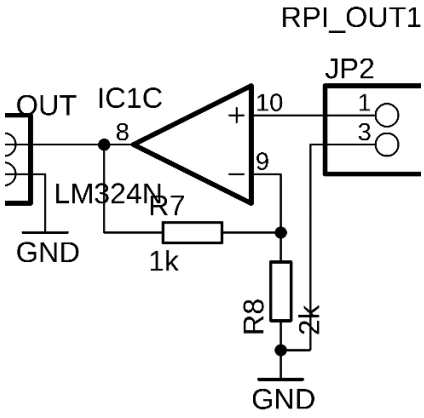
Obrázek 8: Fotografie zařízení – pohled z boku



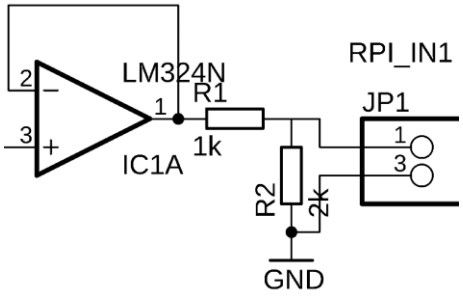
Obrázek 9: Elektrické zapojení



Obrázek 10: Schéma zapojení plošného spoje



Obrázek 11: Detail neinvertujícího zapojení operačního zesilovače z elektrického zapojení



Obrázek 12: Detail zapojení operačního zesilovače jako "opakovač" z elektrického zapojení

Rovnice (5) a (6) určují zesílení A_u neinvertujícího zapojení operačního zesilovače a max. výstup. napětí U_{8max}.

$$A_u = 1 + \frac{R_2}{R_1} = 1 + \frac{1}{2} = 1{,}5 \tag{5}$$

$$U_{8max} = U_{10max} \times A_u = 3{,}3 \times 1{,}5 = 4{,}95V \tag{6}$$

kde

A_u– hodnota zesílení neinvertujícího zapojení operačního zesilovače

R_1– rezistivita rezistoru R_1 (kΩ)

R_2 – rezistivita rezistoru R_2 (kΩ)

U_{8max}– napětí na výstupu (pin 8) zesilovače IC1C (V)

U_{10max}– napětí na kladném vstupu (pin 10) zesilovače IC1C (V).

Výpočet maximálního napětí U_{JP1max} na svorkovnici JP1:

$$U_{JP1max} = U_{3max}\frac{R_2}{R_1 + R_2} = 5\ \frac{2}{1+2} = 3.34V \quad (7)$$

kde

U_{JP1max}– maximální napětí na pinu 1 svorkovnice JP1 (V)

U_{3max} – maximální napětí na kladném vstupu (pin 3) zesilovače IC1A (V)

R_1– rezistivita rezistoru R_1 (kΩ)

R_2 – rezistivita rezistoru R_2 (kΩ).

2.2 Programová realizace

Na začátku programu je provedena inicializace knihoven, které jsou dále v programu použity. Program je složen z funkce main() a šesti uživatelských funkcí. Funkce main() má za úkol získat od uživatele data pro nastavení funkcí programu. Tyto funkce jsou ošetřeny, aby uživatel nemohl zadat nesmyslná data. Uživatelské funkce jsou blíže rozebrané v podkapitolách 2.3.2- 2.3.7

2.3 Použité knihovny

Program využívá standardní knihovnu jazyka C++ a knihovnu pro ovládání modulu ADC-DAC RPi zero, dodanou výrobcem. Do programu jsou vloženy následující hlavičkové soubory pomocí #include:

- unistd.h
  - standartní knihovna pro jazyk C++
- iostream
  - třídy pro standartní V/V
- fstream
  - třídy pro práci se soubory
- cmath
  - třídy pro standartní matematické funkce
- complex
  - třídy pro práci s komplexními čísly
- vector
  - třídy pro práci s poli proměnné velikosti
- thread
  - třídy pro práci s vlákny procesoru
- chrono
  - třídy pro přesné měření času
- limits
  - třídy pro práci s limity proměnných
- ABE_ADCDACPi.h
  - knihovna pro ovládání modulu DAC RPi zero (Abelectronicsuk, © 2024)

2.3.1 Struktura programu

Program je standardně spuštěn vyvoláním funkce main(), která realizuje hlavní smyčku programu, která čeká na parametry od uživatele a následně provede měření. Program je schopen

signál generovat. Tato možnost je jedním z dotazů na parametry měření v úvodu programu. Signál, který má být generován je potřeba mít připravený v samostatném textovém formátu. Je potřeba, aby soubor obsahoval pouze hodnoty signálu uspořádané v jednom sloupci. Podrobněji se tomuto tématu věnuje kapitola 3.1.1

V programu jsou definovány následující globální proměnné:

- Period
  - Hodnota určuje délku signálu – délku měření v ms
- numSamples
  - počet vzorků za jedno měření
- numMeasurements
  - počet měření
- signalFilename
  - název textového souboru s daty pro generování signálu
- generate
  - logická proměnná, která určuje, zda se bude signál generovat
- measurements
  - proměnné pole, které ukládá výsledky měření
- average
  - proměnné pole, které ukládá výsledek zprůměrování měření
- spectrum
  - proměnné pole, které ukládá komplexní hodnoty po provedení DFT

V programu jsou definovány následující uživatelské funkce:

- getParameters()
  - získá od uživatele data pro nastavení měření
- readAndSaveADCData()
  - pomocí ADC-DAC modulu provede měření
- ProcessMeasuredData()
  - zpracuje naměřená data
- computeDFT()
  - provede výpočet DFT

Všechny funkce jsou ve zdrojovém souboru mereni_a_DFT.cpp.

2.3.2 Funkce readAndSaveADCData()

Úkolem funkce readAndSaveADCdata() je pomocí knihovny ABE_ADCDACPi.h, která ovládá ACD-DAC modul, zajistit přečtení dat na vstupu převodníku a uložení výsledku měření. Jejími vstupy jsou uživatelem zadané hodnoty uložené v proměnných period a numSamples a odkaz na proměnnou measurements.

Je rozdělená na čtyři části: zprovoznění komunikace s převodníkem, vypočítání vzorkovacího intervalu, přečtení a uložení hodnot z převodníku a ukončení komunikace s převodníkem.

Data jsou uložena do proměnné measurements. Proměnná measurements je typu std::vector<std::vector<double>>. Tuto možnost přidává knihovna vector, která zjednodušuje práci s poli. Proměnná measurements je dvourozměrné pole o numMeasurements řádcích a numSamples sloupcích. Ve funkci main je definována jako vector ve kterém je numMeasurements vectorů, které mají nula sloupců(členů). Až ve funkci readAndSaveADCdata se členy postupně přidávají. Za každé jedno měření se jeden vector naplní numSamples členy. K tomu je použit příkaz:

measurements.push_back(adcdac.read_adc_voltage(1, 0)).

Tento příkaz přidá změřenou hodnotu na konec vectoru measurements. Následuje příkaz: usleep(samplingInterval * 1000). Ten zajistí uspání procesu na samplingInterval milisekund.

Obrázek 13 zobrazuje část programu s funkcí readAndSaveADCData.

```
using namespace ABElectronics_CPP_Libraries;
//funkce pro čtení z ADC převodníku
void readAndSaveADCData(int period, int numSamples, std::vector<double>& measurements, int mer) {
    ADCDACPi adcdac; //definování objektu adcdac
    adcdac.open_adc(); //zahájení komunikace s ADC
    // Vypočítat interval vzorkování
    double samplingInterval = period / numSamples; //výpočet intervalu vzorkování
    std::cout << "sample interval: "<< samplingInterval << std::endl; //informování uživatele o i

    for (int i = 0; i < numSamples; ++i) {
        //double value = adcdac.read_adc_voltage(1, 0); //uložení výsledku měření
        measurements.push_back(adcdac.read_adc_voltage(1, 0)); //Uložení amplitudy, __(kanál, mód
        usleep(samplingInterval * 1000); //počkání samplingInterval milisekund
    }
    adcdac.close_adc(); //ukončení komunikace s ADC
}
```

Obrázek 13: Ukázka části programu s funkcí readAndSaveADCData()

2.3.3 Funkce generateAndWriteSignal()

Funkce generateAndWriteSignal() také využívá knihovnu ABE_ADCDACPi.h. Využívá část pro ovládání D/A převodníku. Pomocí této funkce lze vygenerovat předem připravený signál ve formě dat. Tyto data lze připravit například v MATLABu. Vstupem této funkce je název souboru s vygenerovanými daty a časový úsek, za který mají být data vygenerovány. Časový úsek je zadávaný v milisekundách ve funkci getParameters() a je uložen v proměnné period.

Funkce je rozdělena na čtyři části: zprovoznění komunikace s převodníkem a otevření souboru s daty, vypočítání generovacího intervalu, přečtení hodnot ze souboru a nastavení hodnot na převodník, ukončení komunikace s převodníkem a ukončení čtení souboru.

Generování hodnot probíhá ve smyčce while. Podmínka „inputFile >> value“ zajišťuje, že generování hodnot bude probíhat dokud nedojde funkce na konec souboru. I zde je použita funkce usleep(), která byla vysvětlena v kapitole 2.3.2

Pro běžné funkce byly vytvořena data o tisíci vzorcích. Pro generování dat ze zvukového souboru byl počet dat větší a přesná vzorkovací frekvence. Tyto hodnoty je potřeba zadat na řádcích 41 a 42 do proměnných numPoints a period1. Také je potřeba odkomentovat řádek 44 a zakomentovat řádek 43, aby byla pro výpočet použita proměnná period1.

Obrázek 14 zobrazuje část programu s funkcí generateAndWriteSignal().

```
void generateAndWriteSignal(const std::string& filename, int period) {

    std::ifstream inputFile(filename);
    if (!inputFile.is_open()) {
        std::cerr << "Nelze otevrit soubor " << filename << " pro cteni." << std::endl;
        return;
    }
    double value;
    ADCDACPi adcdac; //definování objektu adcdac
    adcdac.open_dac(); //zahájení komunikace s ADC
    adcdac.set_dac_gain(2); // nastavení DAC na hodnoty mezi 0 a 3.3V

    int numPoints = 1000; // Počet bodů v souboru s daty pro generování, původně 1000, p
    //int period1 = 8868; //manualni nastaveni periody
    double interval = period / numPoints; // Interval vzorkování - period1 nebo period p
    std::cout << "generate interval: "<< interval << std::endl; //inofrmování uživatele

    while (inputFile >> value) { //čtení pracuje se streamem, dokud budou v souboru data
        adcdac.set_dac_voltage(value, 2); // Nastavení hodnoty na DAC
        usleep(interval * 1000); // Pauza mezi generováním výstupů na DAC
    }
    inputFile.close(); //zavření souboru pro čtení
    adcdac.close_dac(); //ukončení komunikace s DAC
    std::cout << "generate end" << std::endl; //inofrmování uživatele
}
```

Obrázek 14: Ukázka části programu s funkcí generateAndWriteSignal()

2.3.4 Funkce computeDFT()

Funkce computeDFT() provede výpočet DFT, výsledky uloží do proměnné spectrum, spočítá dobu výpočtu a proměnou spectrum uloží do souboru dft.txt ve formátu

Re: x,xxxx Im: x,xxxx

Funkce má čtyři části: uložení aktuálního času do proměnné start, výpočet a uložení výsledků DFT, uložení času do proměnné end a výpočet doby trvání výpočtu DFT, zapsání výsledků do souboru a ukončení zápisu do souboru.

Měření času zajišťuje knihovna chrono. Její funkce high_resolution_clock() počítá mikrosekundy od startu programu. Tyto hodnoty můžeme uložit do proměnných start a stop a jejich odečtením získáme čas, za který se vykonal kód mezi uložením těchto hodnot. Tento čas po výpočtu převede na milisekundy a sekundy a vypíše na konzoli pro uživatele.

Vzorec pro výpočet DFT (1), je ve funkci zapsán pomocí dvou for cyklů. Ukázka zapsání vzorce v programu je na Obrázek 15. Obrázek 16 zobrazuje část programu s funkcí computeDFT().

```
for (int k = 0; k < N; ++k) {
    spectrum[k] = 0;
    for (int n = 0; n < N; ++n) { //výpočet DFT
        double angle = -2.0 * M_PI * k * n / N;
        std::complex<double> complex_exp(cos(angle), sin(angle));
        spectrum[k] += signal[n] * complex_exp;
    }
}
```

Obrázek 15: Ukázka zapsání DFT v programu

```cpp
void computeDFT(const std::vector<double>& signal, std::vector<std::complex<double>>& spectrum) {
    std::cout << "dft" << std::endl;
    auto start = std::chrono::high_resolution_clock::now();//zahájení počítání času
    int N = signal.size(); //proměnná N ukládá hodnotu počtu naměřených vzorků
    for (int k = 0; k < N; ++k) {
        spectrum[k] = 0;
        for (int n = 0; n < N; ++n) { //výpočet DFT
            double angle = -2.0 * M_PI * k * n / N;
            std::complex<double> complex_exp(cos(angle), sin(angle));
            spectrum[k] += signal[n] * complex_exp;
        }
    }
    auto end = std::chrono::high_resolution_clock::now(); //ukončneí počítání čase
    auto duration = std::chrono::duration_cast<std::chrono::microseconds>(end - start); //výpočet ča
    auto mills = std::chrono::duration_cast<std::chrono::milliseconds>(duration); //převod na milise
    auto sec = std::chrono::duration_cast<std::chrono::seconds>(duration); //převod na sekundy
    std::cout << "cas vypoctu: " << mills.count() << " milisekund" << std::endl; //informování uživa
    std::cout << "to je: " << sec.count() << " sekund" << std::endl; //informování uživatele v [s]

    std::ofstream outputFile("mer_2_dft.txt");
        if (!outputFile.is_open()) {
            std::cerr << "Nelze otevrit soubor mer_2_dft.txt pro zapis." << std::endl;
            return;
        }
        for (int i = 0; i < N; ++i) {
            outputFile << "Re " << spectrum[i].real() << " Im " << spectrum[i].imag() << std::endl;
        }
        outputFile.close();
}
```

Obrázek 16: Ukázka programu s funkcí computeDFT()

2.3.5 Funkce ProcessMeasuredData()

Funkce ProcessMeasuredData() zapisuje data z proměnné measurements do souboru a počítá průměr z naměřených hodnot. Vstupy funkce tvoří hodnoty numSamples a numMeasurements a odkazy na proměnné measurements a average. Soubor s naměřenými hodnotami je použit při zpracování dat v MATLABu.

Zápis naměřených hodnot do souboru je proveden ve dvou for cyklech. Ve vnitřním cyklu je postupně poskládán řetězec v proměnné prompt. Ten je ve vnějším cyklu zapsán do souboru. Takto projde cyklus každý řádek souboru. Data jsou zapsána ve formátu, který ukazuje Obrázek 17.

```
1.1:0.479370 1.2:0.481787 1.3:0.480981
2.1:1.687866 2.2:1.687866 2.3:0.803247
3.1:1.293091 3.2:1.295508 3.3:1.621802
4.1:2.550732 4.2:2.553955 4.3:2.540259
5.1:0.311792 5.2:0.307764 5.3:0.974048
6.1:1.904590 6.2:1.905396 6.3:1.905396
7.1:2.559595 7.2:2.562817 7.3:2.566040
8.1:0.959546 8.2:0.959546 8.3:0.960352
9.1:2.526562 9.2:2.526562 9.3:2.527368
10.1:2.188184 10.2:2.188184 10.3:2.187378
```

Obrázek 17: Ukázka ze souboru measurements

V Druhé části jsou vypočítány průměry hodnot z jednotlivých měření a uloženy do proměnné average. Pro výpočet průměrů hodnot jsou použity dva for cylky. Ve vnitřním cyklu jsou data sečteny. Tyto data jsou poté ve vnějším cyklu vyděleny počtem měření numMeasurements.

Obrázek 18 zobrazuje část programu s funkcí ProcessMeasuredData().

```
void ProcessMeasuredData(int numSamples, int numMeasurements, std::vector<std::vector<double>>& measurements, std::vecto

    // zapsání výsledků jednotlivých měření do souboru measurements.txt
    std::ofstream outputFile("measurements_2.txt");
    if (!outputFile.is_open()) {
        std::cerr << "Nelze otevrit soubor measurements.txt pro zapis." << std::endl;
        return;
    }
    for(int i = 0; i < numSamples; ++i) {
    std::string prompt;
        for (int j = 0; j < numMeasurements; ++j) {
            prompt += std::to_string(i+1) + "." + std::to_string(j+1) + ":" + std::to_string(measurements[j][i]) + " ";
            //std::cout << prompt << std::endl;
        }
    outputFile << prompt << std::endl;
    }
    outputFile.close();

    // Výpočet průměru z jednotlivých měření
    for (int j = 0; j < numSamples; ++j) {
        for (int i = 0; i < numMeasurements; ++i) {
            average[j] += measurements[i][j];
        }
    average[j] /= numMeasurements;
    }
}
```

Obrázek 18: Ukázka programu s funkcí ProcessMeasuredData()

2.3.6 Funkce validateInput()

Funkce validateInput() je šablonová funkce. Tato funkce může pracovat s různými datovými typy. V tomto kódu pracuje s typy: int, double, char a string. Vstupem této funkce je string prompt. Tato proměnná obsahuje text, který bude vypsán do příkazového řádku. Tento text informuje uživatele, jaká data má zadat.

Zadaná data jsou uložena do proměnné value. Datový typ této proměnné je určen při použití této funkce v kódu. Pokud data nelze do proměnné uložit, je uživatel funkcí ve smyčce while znovu vyzván k zadání dat. Tato smyčka je opakována, dokud uživatel nezadá správný typ dat. Příkaz: std::cin.clear() zajistí, že může uživatel zadat nová data. Příkaz

std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n')

zajistí, že jsou ignorována všechna data až do znaku '\n', který značí nový řádek. Tímto je zajištěno, že se uloží až nová data zadaná uživatel. Pokud jsou zadána správná data je smyčka proražena příkazem break a funkce ukončena.

Obrázek 19 zobrazuje část programu s funkcí validateInput().

```
template<typename T> // šablonový parametr - možnost pracovat s různými typy(int
T validateInput(const std::string& prompt) {
    T value;
    while (true) {
        std::cout << prompt; //vypsání výzvy k zadání vstupu
        if (!(std::cin >> value)) {
            std::cerr << "Chybny vstup. Zadejte platnou hodnotu." << std::endl;
            std::cin.clear(); //odstranění předchozího vstupu
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        } else {
            break;
        }
    }
    return value;
}
```

Obrázek 19: Ukázka funkce validateInput()

2.3.7 Funkce getParameters()

Ve funkci getParameters() jsou získávána a kontrolována data od uživatele, kterými je celý program nastaven. Vstupy jsou odkazy na proměnné: period, numSamples, numMeasurements, signalFilename a generate.

Většina dat je kontrolována ve dvou úrovních. V první úrovni je kontrolován typ zadaných dat pomocí šablonové funkce validateInput(). Typ dat je specifikován ve formátu

validateInput<typ>(„text pro uživatele“)

V druhé úrovni jsou kontrolovány meze, ve který se mají data nacházet. Tyto meze jsou součástí výzvy uživateli. Pokud data tyto meze nesplňují jsou opakovány výzvy na zadání dat. Tyto výzvy jsou opakovány, dokud nejsou zadána správná data. U výzvy o zadání znaku 'A' nebo 'N' jsou přijmuty i znaky 'a' nebo 'n'. Pokud uživatel zadá u výzvy

Chcete vstup generovat? (A/N):

znak 'A' nebo 'a', je vyzván k zadání názvu souboru s daty pro generování signálu. U tohoto vstupu je provedena v druhé vrstvě kontrola, zda název souboru s daty pro generování existuje. Kontrola je provedena pokusem o otevření souboru pro zápis. Pokud soubor lze otevřít je funkce ukončena, jinak je opakován proces, dokud není zadán název souboru, který lze otevřít.

Obrázek 20 zobrazuje část programu s funkcí getParameters().

```
// Funkce pro získání parametrů od uživatele
void getParameters(int& period, int& numSamples, int& numMeasurements, std::string& signalFilename, bool& generate) {
    period = validateInput<int>("Zadejte periodu signalu (v ms): "); //zkontoroluje, zda uživatel zadal číslo int
    numSamples = validateInput<int>("Zadejte pocet vzorku (100-256): "); //zkontoroluje, zda uživatel zadal číslo int
    while (numSamples < 100 || numSamples > 256) { //pokud není zadané číslo v daném limitu opakuje se výzva
        std::cerr << "Neplatny pocet vzorku. Zadejte hodnotu mezi 100 a 256." << std::endl;
        numSamples = validateInput<int>("Zadejte pocet vzorku (100-256): ");
    }
    numMeasurements = validateInput<int>("Zadejte pocet mereni (1-10): "); //zkontoroluje, zda uživatel zadal číslo ir
    while (numMeasurements < 1 || numMeasurements > 10) { //pokud není zadané číslo v daném limitu opakuje se výzva
        std::cerr << "Neplatny pocet mereni. Zadejte hodnotu mezi 1 a 10." << std::endl;
        numMeasurements = validateInput<int>("Zadejte pocet mereni (1-10): ");
    }
    char generateInput; //proměnná do které se uloží uživatelský vstup
    while (true) {
        generateInput = validateInput<char>("Chcete vstup generovat? (A/N): "); //zkontoroluje, zda uživatel zadal cha
        if (generateInput == 'A' || generateInput == 'N' || generateInput == 'a' || generateInput == 'n') { //ověří zd
            generate = (generateInput == 'A' || generateInput == 'a');
            break;
        } else {
            std::cerr << "Neplatny vstup. Zadejte 'A' pro Ano nebo 'N' pro Ne." << std::endl;
        }
    }
    if(generate){
        while(true){
            signalFilename = validateInput<std::string>("Zadejte cely nazev souboru (ve formatu ____.txt) s generovany
            std::ifstream file(signalFilename); //otevře soubor se zadaným názvem
            if(file.is_open()){
                file.close(); //lze otevřít -> ukončí smyčku while
                break;
            }else{
                std::cerr << "Soubor " << signalFilename << " nelze otevrit" << std::endl; //nelze otevřít -> opakuje
            }
        }
    }else{
        signalFilename = "";
    }
}
```

Obrázek 20: Ukázka funkce getParameters()

2.4 Použití programu

Program na základě zadaných parametrů měření načte data poskytovaná ADC-DAC modulem, provede DFT transformaci a výsledek zapíše do textového souboru, který je uložený ve složce s programem, odkud je možné data zkopírovat na flashdisk pro další zpracování.

Program je spuštěn v příkazovém řádku. Před vykonáním hlavní části programu je uživatel několikrát vyzván, aby zadal jednotlivé parametry měření.

Parametry měření jsou:

- perioda signálu
- počet vzorků
- počet měření
- možnost signál generovat
- název souboru s daty pro generování signálu.

Periodou signálu je myšlena délka měření a je zadávána v milisekundách. Počet vzorků je omezen a uživatel může zadat číslo od 100 do 256 vzorků. Program umí změřit i více vzorků po drobné úpravě programu, pro základní analýzu však stačí toto rozmezí. Počet měření je též omezen na počet do deseti měření. Pokud je provedeno více než jedno měření je výsledkem

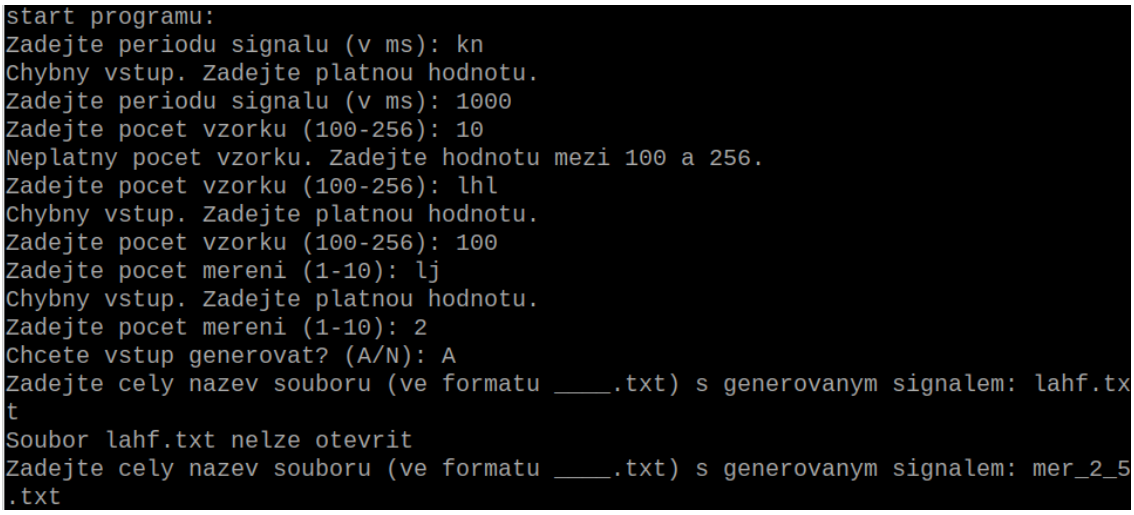
měření průměrná hodnota ze všech měření. Na dotaz, zda uživatel chce vstup generovat, odpoví znakem A pro ano nebo N pro ne. Pokud zadá znak A, je vypsána výzva na zadání názvu souboru s daty pro generování signálu.

Pokud uživatel v jakémkoli kroku zadá špatná data, vypíše program výzvu znovu a opakuje výzvu, dokud nejsou zadána správná data. Na obrázku 2 je výpis z programu, kdy byly záměrně zadávána chybná data a byla testována reakce programu.

Po zadání správných dat program informuje o každém zahájeném měření a vypisuje interval vzorkování a interval mezi generovanými vzorky- oboje v ms. Program oznámí, že začíná výpočet DFT, oznámí, jak dlouho výpočet trval a oznámí, že vše proběhlo v pořádku. Obrázek 21 znázorňuje tento proces. Obrázek 22 ukazuje reakci programu na chybné vstupy.

```
start programu:
Zadejte periodu signalu (v ms): 1000
Zadejte pocet vzorku (100-256): 100
Zadejte pocet mereni (1-10): 3
Chcete vstup generovat? (A/N): n
Neplatny vstup. Zadejte 'A' pro Ano nebo 'N' pro Ne.
Chcete vstup generovat? (A/N): N
Mereni 1
sample interval: 10
Mereni 2
sample interval: 10
Mereni 3
sample interval: 10
dft
cas vypoctu: 10 milisekund
to je: 0 sekund
Vsechno probehlo v poradku :)


------------------
(program exited with code: 0)
Press return to continue
```

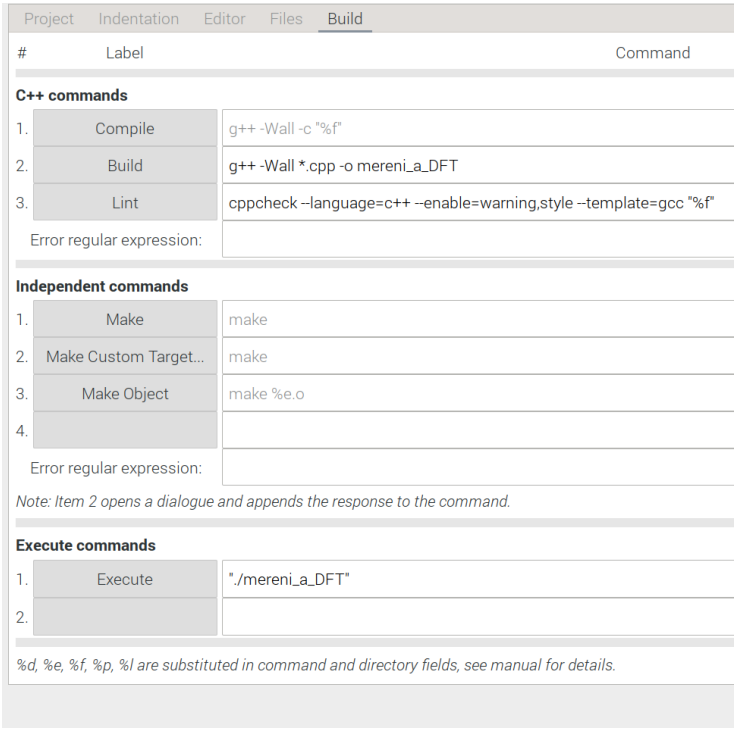
Obrázek 21: Ukázka komunikace programu s uživatelem



Obrázek 22: Reakce programu na chybné vstupy

2.5 Nastavení záložky Build pro správnou kompilaci

Pro sestavení a spouštění kódu v programovacím jazyce C++ na RPI byl použit program Geany. Program používá knihovnu ABE_ADCDACPi.h a soubor ABE_ADCDACPi.cpp. Tyto dva soubory je potřeba mít ve složce s napsaným programem. Protože tyto soubory jsou používány při sestavování kódu, je potřeba, aby byla správně nastavena v Project Properties záložka build. Obrázek 23 zobrazuje toto nastavení. Zvlášť velkou pozornost je třeba věnovat řádku Build v části C++ commands.

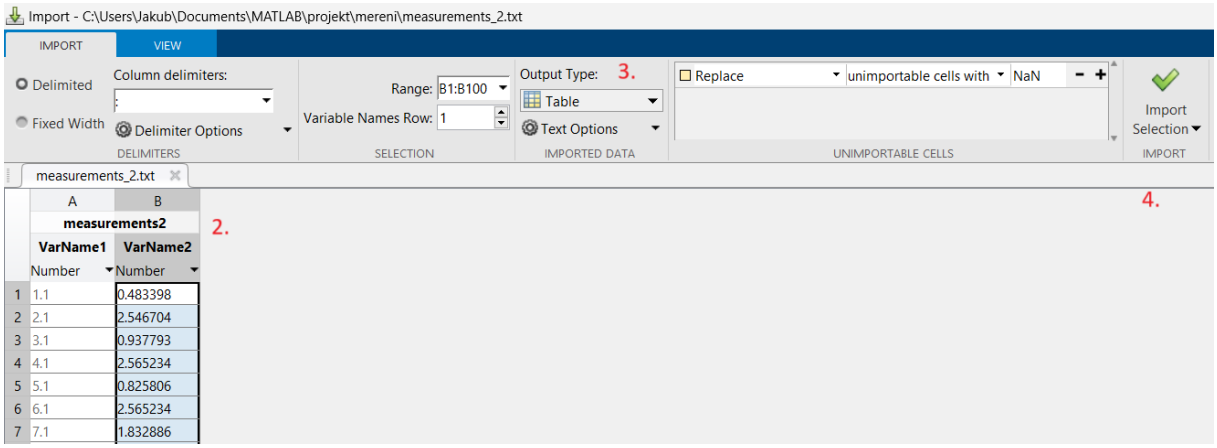


Obrázek 23: Nastavení properties build v programu Geany

2.6 Importování naměřených dat do MATLABu

Data ze souboru measurements.txt je třeba do MATLABu naimportovat. V kartě domů je záložka Import Data. Postup pro importování je zobrazuje Obrázek 24 a je následující:

1. Zvolit importovaný soubor
2. Označit sloupce, které obsahují naměřená data a přepsat název sloupce VarName2 na data
3. Pokud je sloupců s naměřenými daty víc je potřeba změnit Output Type na Column vectors
4. Importovat od MATLABu



Obrázek 24: Postup importu dat do MATLABu

2.7 Vytvoření grafu naměřených hodnot v MATLABu

Pro vytvoření grafu naměřených hodnot jsou potřeba data, která byla v kapitole 2.6 naimportována. Pomocí skriptu zobrazeni_dat.m je graf hodnot vytvořen a zobrazen. Je potřeba ve skriptu správně nastavit frekvenci vzorkování. Vytvořený graf zobrazuje např. Obrázek 33.

```
clear
% potřeba načíst data pomocí importu

% Vzorkovací frekvence
sampling_frequency = 200;

% Časový vektor
t = (0:length(data)-1) / sampling_frequency;

% Vykreslení grafu
plot(t, data, 'bo-');
xlabel('Čas, s');
ylabel('Napětí, V');
title('Graf naměřených dat');
grid on;

% zvětšení fontu
ax = gca;
ax.FontSize = 15;
```

Obrázek 25: Ukázka z matlab skriptu zobrazeni_dat

2.8 Vytvoření grafu frekvenčních složek DFT

Pro vytvoření grafu frekvenčních složek jsou potřeba data ze souboru mer_2_dft. Pomocí skriptu zpracovani_dat.m je graf hodnot vytvořen a zobrazen. Je potřeba ve skriptu správně nastavit název souboru s výsledky DFT a frekvenci vzorkování f_samp. Skript zobrazuje Obrázek 26. Vytvořený graf zobrazuje např. Obrázek 34.

```
clear
data = import_datafile('mer_2_4dft.txt');%import dat z textového souboru
ampl = abs(data); %amplituda signálu
N = length(data); % délka signálu
f_samp = 200; % frekvence vzorkování (v Hz)
f = (0:N-1) * f_samp / N; % frekvenční osa

% Vykreslení frekvenčního spektra
figure;
stem(f,ampl,'LineStyle','-.', 'MarkerFaceColor','red','MarkerEdgeColor',
xlabel('Frekvence, Hz');
ylabel('Amplituda, 1');
title('Frekvenční spektrum');

% zvětšení fontu
ax = gca;
ax.FontSize = 15;
```

Obrázek 26: Ukázka z MATLAB skriptu zpracovani_dat

3 EXPERIMENTÁLNÍ ČÁST

3.1 Signály použité pro měření

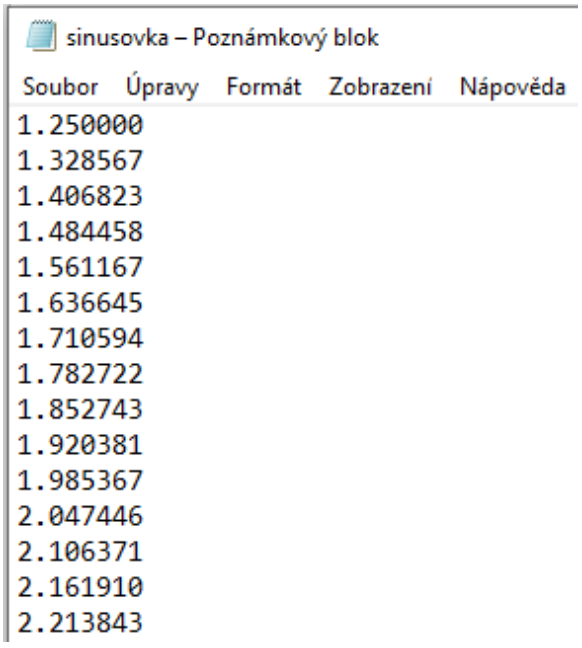
Byl změřen výstup usměrňovače síťového napětí 50Hz, bez filtračního kondenzátoru. Výsledky měření jsou popsány v kapitole Usměrňovač 50Hz, sinusový signál popsaný v kapitole Sinusový signál, obdélníkový signál popsaný v kapitole Obdélníkový signál a dva zvukové signály popsané v kapitole Zvukový signál. Kromě usměrněného signálu byly všechny signály uměle vytvořeny v MATLABu, zapsány na D/A převodník a změřeny pomocí A/D převodníku.

3.1.1 Vytvoření dat pro generování signálu

Vytvoření dat pro generování signálu je možné pomocí skriptu signal_gen_and_save.m. Je potřeba pouze odkomentovat potřebnou funkci, kterou je potřeba vygenerovat, nebo napsat vzorec jiné funkce. Také je potřeba nastavit počet vzorků. Počet vzorků je vhodné volit mnohem větší než počet plánovaných vzorků, které je potřeba naměřit. Obrázek 27 ukazuje skript signal_gen_and_save.m v MATLABu. Obrázek 28 ukazuje výsledná uložená data ze souboru sinusovka.txt.

```
% Počet bodů signálu
numSamples = 1000;

% Frekvence sinusovky
freq = 8;

% Amplituda signálu
amplituda = 1.25;

% Časový interval
t = linspace(0, 1000, numSamples);

% Sinusový signál
sinus_signal = amplituda + sin(2 * pi * freq * t) * amplituda;

% Uložení do souboru sinusového signálu
fileID = fopen('sinusovka.txt', 'w');
fprintf(fileID, '%f\n', sinus_signal);
fclose(fileID);

% Obdélníkový signál
% rect_signal = amplituda + square(2 * pi * freq * t) * amplituda;

% Uložení do souboru obdélníkového signálu
% fileID = fopen('obdelnik.txt', 'w');
% fprintf(fileID, '%f\n', rect_signal);
% fclose(fileID);

plot(t,sinus_signal);
```

Obrázek 27: Ukázka ze souboru signal_gen_and_save.m



Obrázek 28: Ukázka ze souboru sinusovka.txt

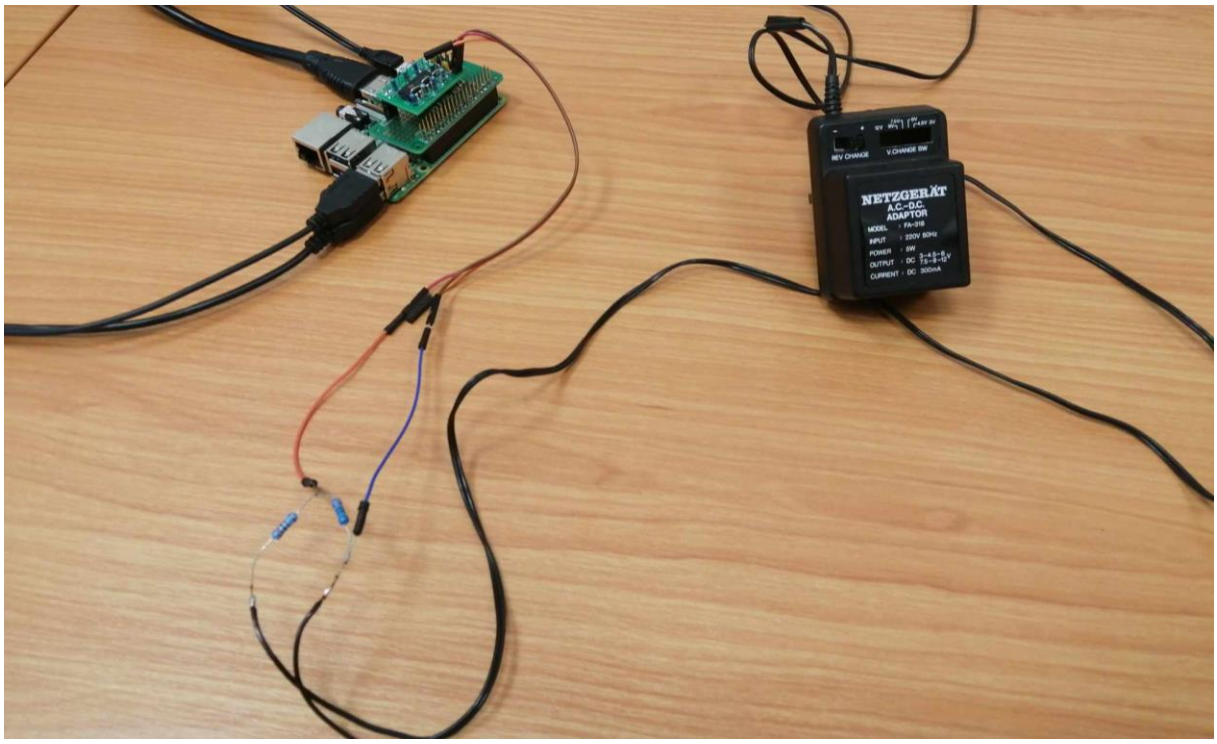
3.2 Výsledky měření

Usměrňovač 50Hz

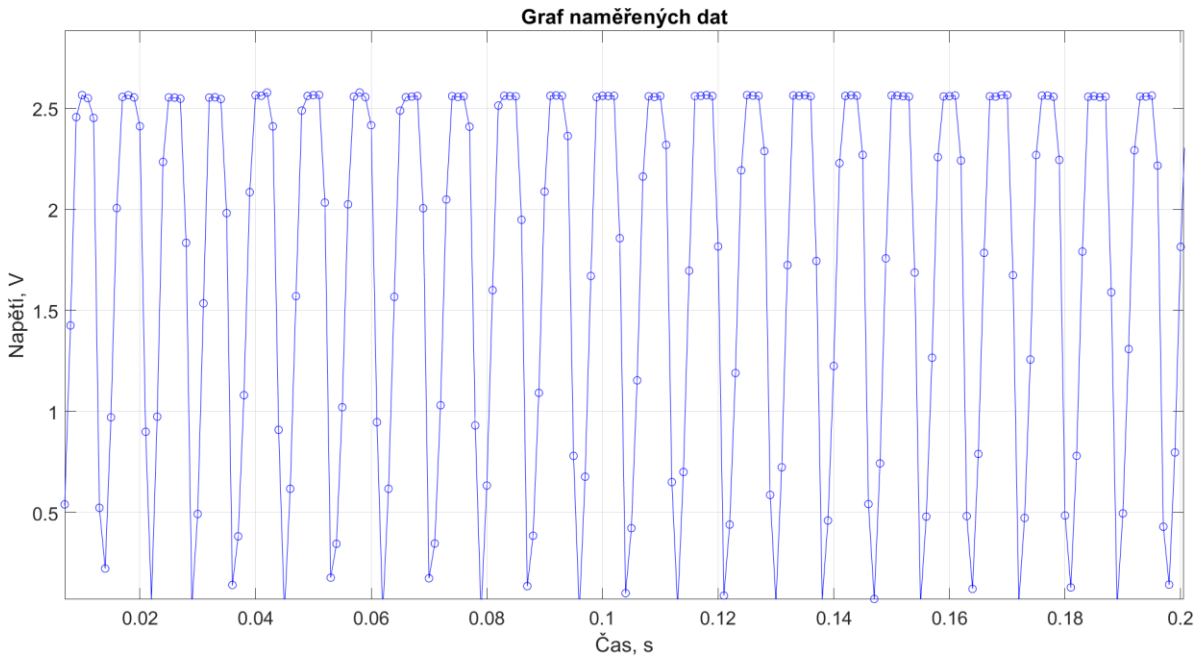
Byl použit usměrňovač síťového napětí 50Hz s přidaným děličem napětí, protože výstupní napětí bylo 12,9V. Dělič byl složen z 1,2kΩ a 330Ω rezistorů, což umožnilo výstupní napětí transformovat do rozsahu 0-2,8V. Obrázek 29 je fotografie z měření. Signál byl navzorkován frekvencí 1kHZ a změřen byl časový úsek 1s.

Obrázek 30 ukazuje naměřené hodnoty zpracované v MATLABu do grafu. Na přiblíženém grafu naměřených hodnot je patrné, že do A/D převodníku vstupovalo pouze napětí do 2,5V. Toto omezení zapříčinila neschopnost operačního zesilovače dodat napětí do 5V, ale pouze do 3,9V při napájení 5V z ADC-DAC modulu. Proto není A/D převodník schopen měřit do max. hodnoty 3,3V, ale pouze 2,5V.

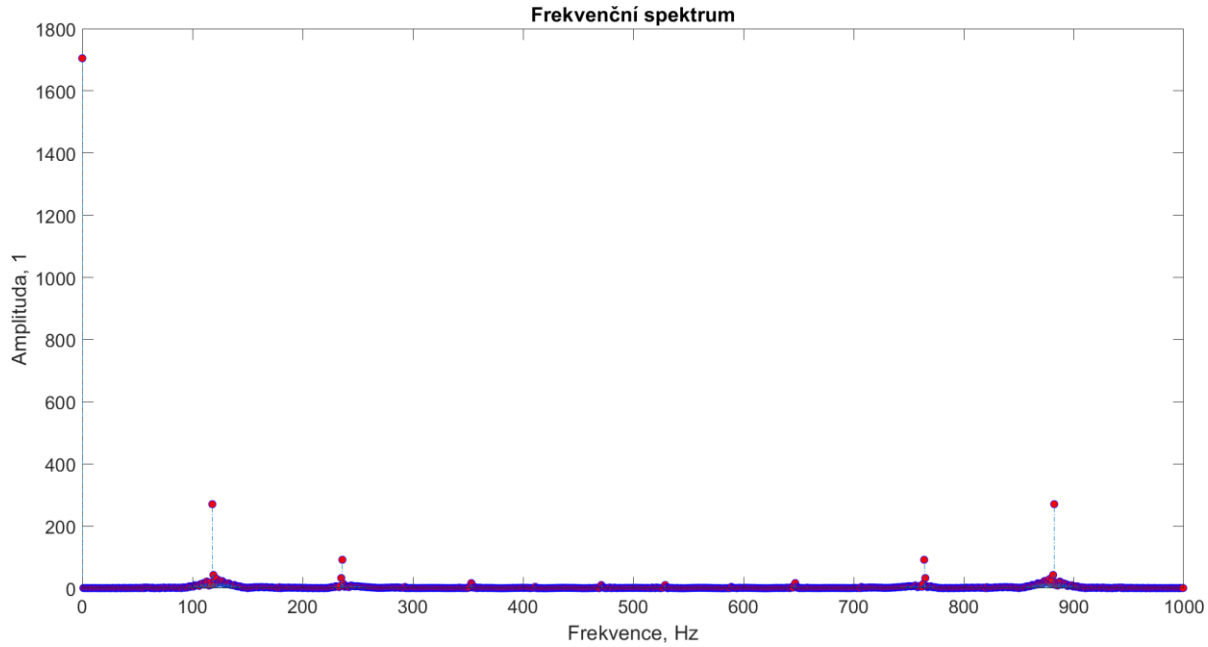
Obrázek 31 a Obrázek 32 zobrazují frekvenční graf výsledků DFT. Na svislé ose je vynesena amplituda a na vodorovné ose je frekvence složky spočítaná podle vztahu (3). Amplituda nulté frekvenční složky odpovídá stejnosměrné složce signálu. Kromě první složky je spektrum symetrické kolem středu. Na přiblíženém frekvenčním grafu je složka s největší frekvencí složka nesoucí informaci o frekvenci změřeného signálu a ostatní složky jsou způsobeny oříznutím signálu na 2.5V. Frekvenční spektrum je podobné obdélníkovému signálu.



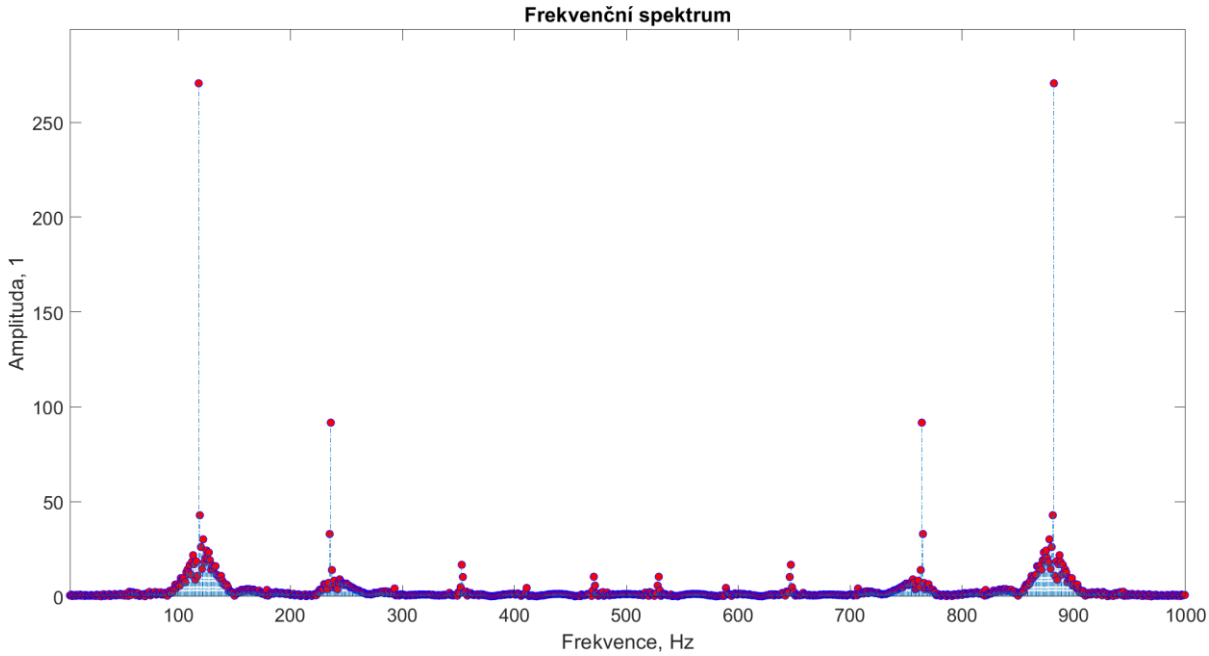
Obrázek 29: Fotografie zapojení usměrňovače



Obrázek 30: Přiblížený graf naměřených hodnot usměrňovače



Obrázek 31: Frekvenční graf usměrňovače



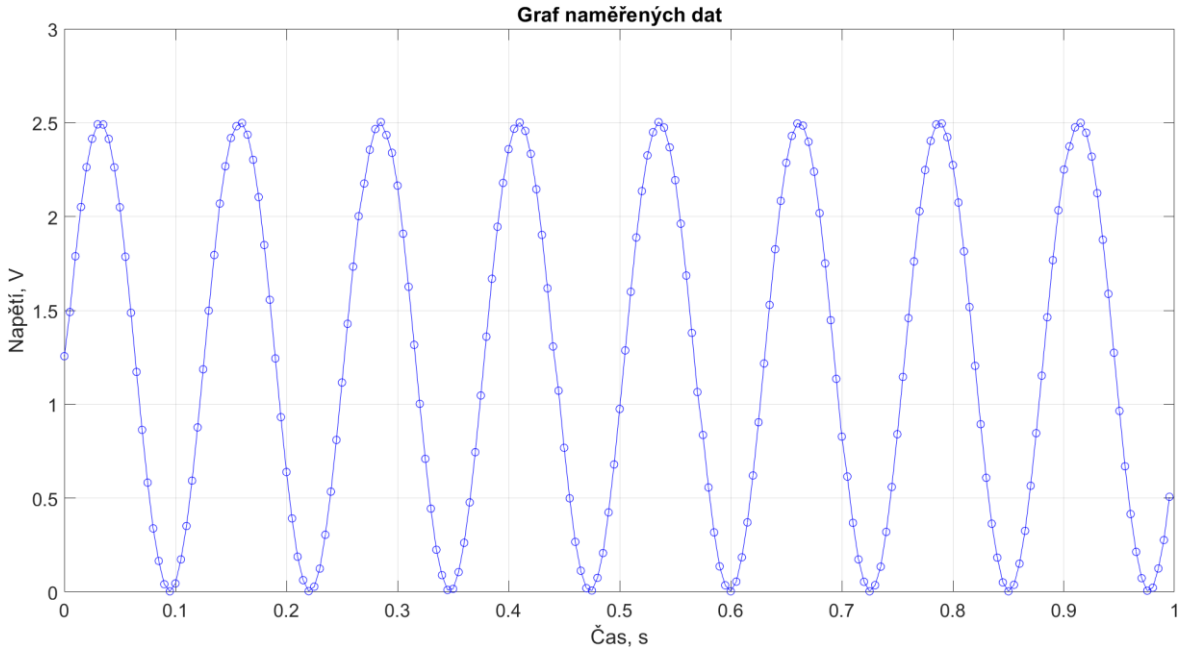
Obrázek 32: Přiblížený frekvenční graf usměrňovače

Sinusový signál

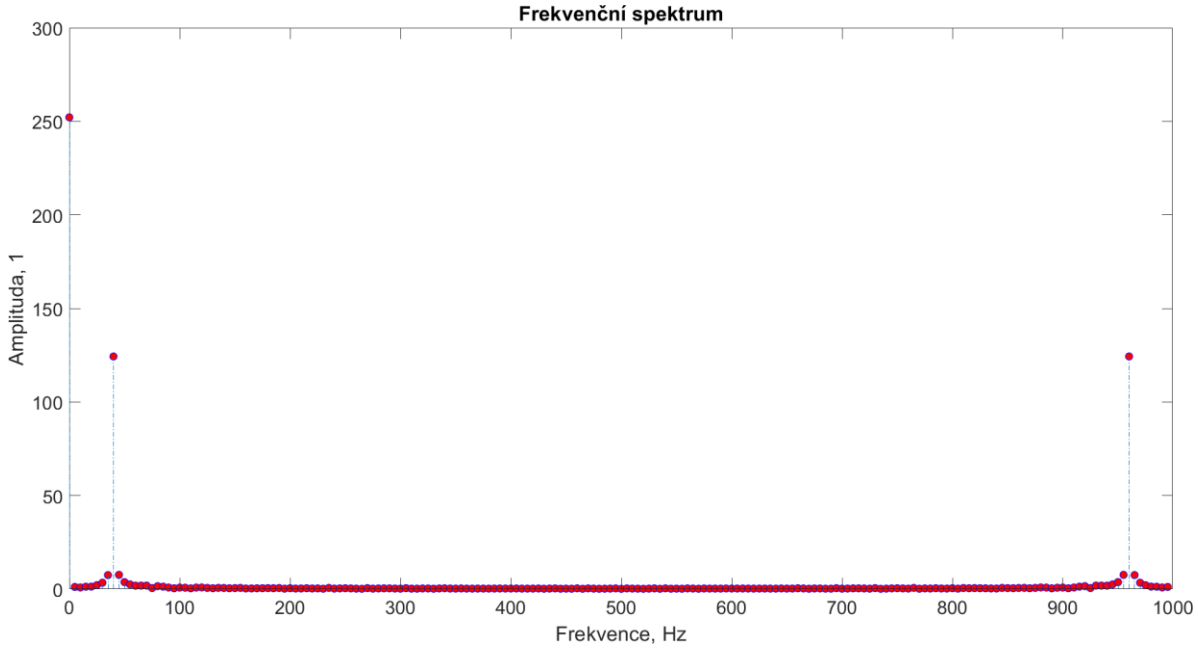
Tento signál byl vytvořen pomocí skriptu popsaném v kapitole 3.1.1. Pro jeho změření byla data generována na D/A převodníku a následně změřena A/D převodníkem. Délka signálu byla 1s, navzorkováno bylo 1000 vzorků a frekvence vzorkování byla 200Hz. Perioda

původního signálu byla 1/8s. Obrázek 33 zobrazuje graf naměřených hodnot a Obrázek 34 zobrazuje frekvenční amplitudový graf. Obrázek 35 zobrazuje frekvenční fázový graf.

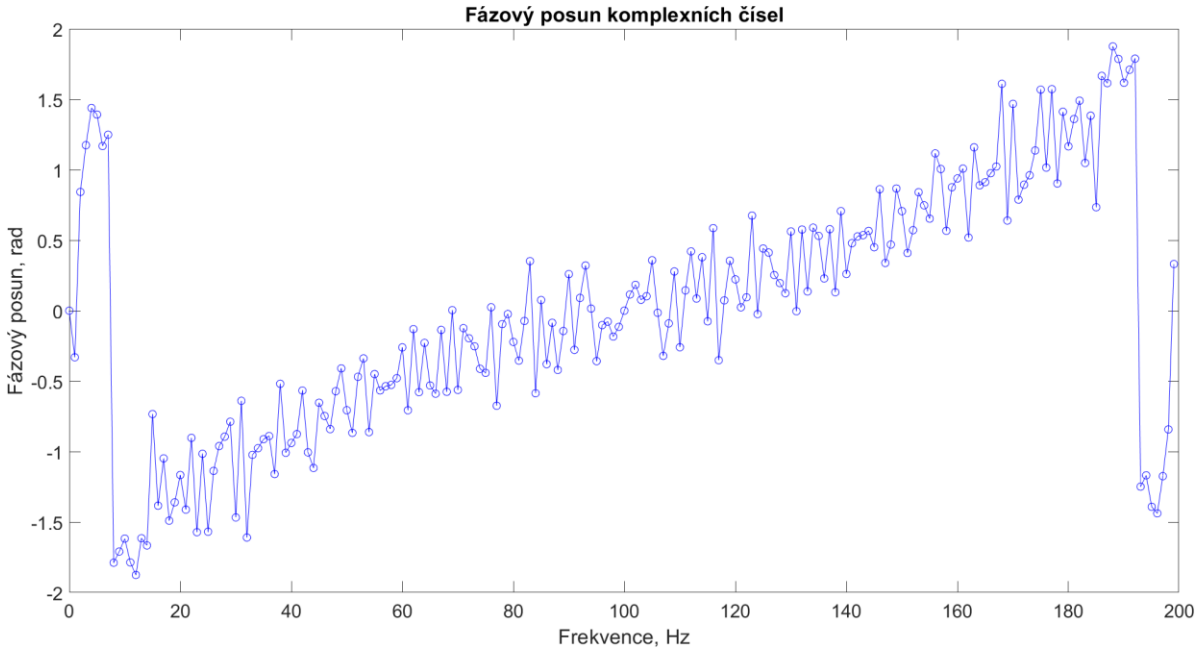
I zde je signál stejnosměrně posunut a první frekvenční složka má velkou amplitudu, příčiny byly vysvětleny v kapitole 3.2 Usměrňovač 50Hz. Frekvenční graf dokazuje, že signál je složen pouze z jedné frekvenční složky a stejnosměrné složky.



Obrázek 33: Graf naměřených hodnot sinusový signál



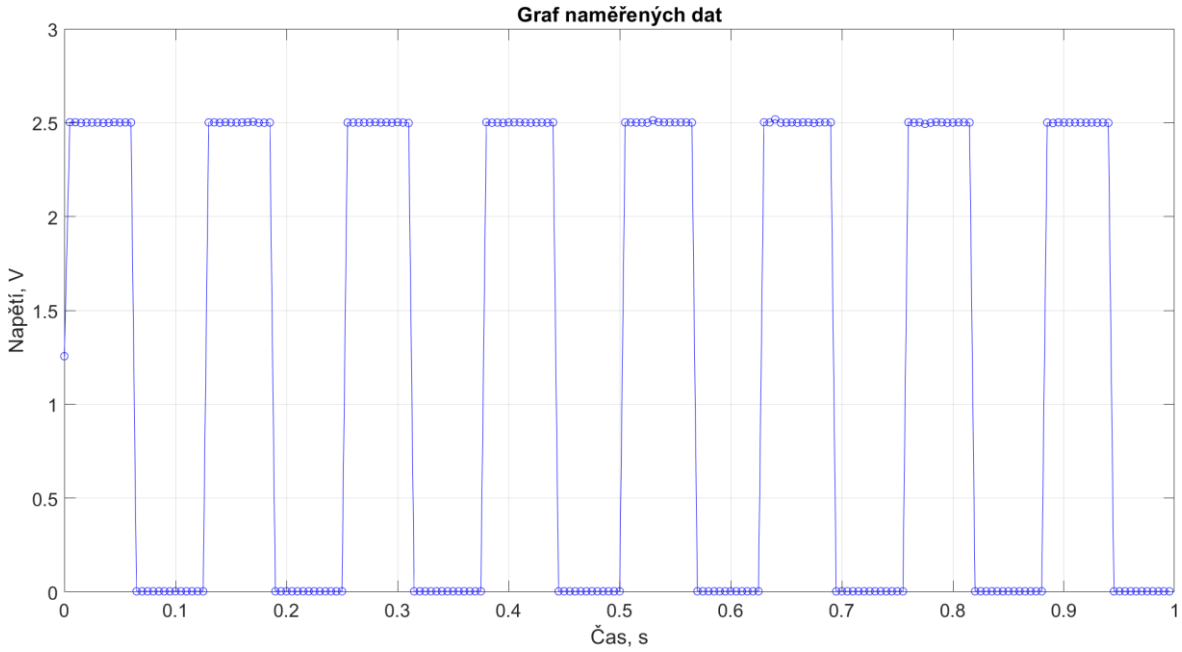
Obrázek 34: Frekvenční graf sinusového signálu



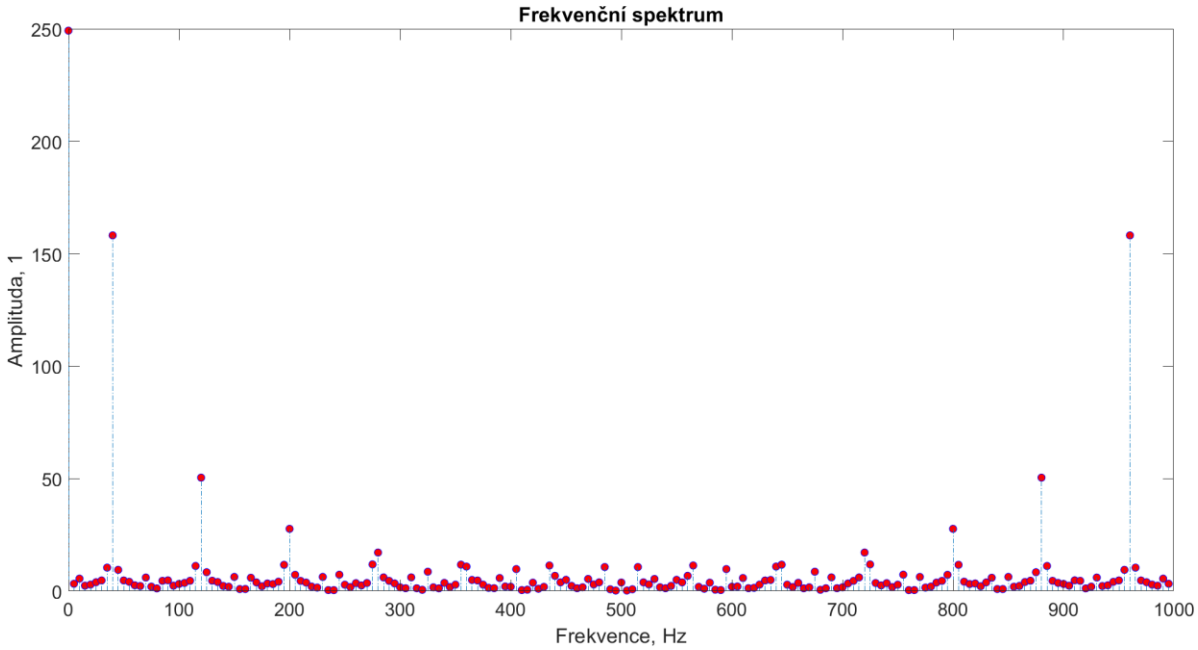
Obrázek 35: Graf fázových posunů sinusového signálu

Obdélníkový signál

Tento signál byl vytvořen pomocí skriptu popsaném v kapitole 3.1.1. Pro jeho změření byla data generována na D/A převodníku a následně změřena A/D převodníkem. Délka signálu byla 1s, navzorkováno bylo 1000 vzorků a frekvence vzorkování byla 200Hz. Perioda původního signálu byla 1/8s. Obrázek 36 zobrazuje graf naměřených hodnot a Obrázek 37 zobrazuje frekvenční graf. I zde je signál stejnosměrně posunut a první frekvenční složka má velkou amplitudu, příčiny byly vysvětleny v kapitole 3.2 Usměrňovač 50Hz.



Obrázek 36: Graf naměřených hodnot obdélníkový signál



Obrázek 37: Frekvenční graf obdélníkového signálu

Zvukový signál

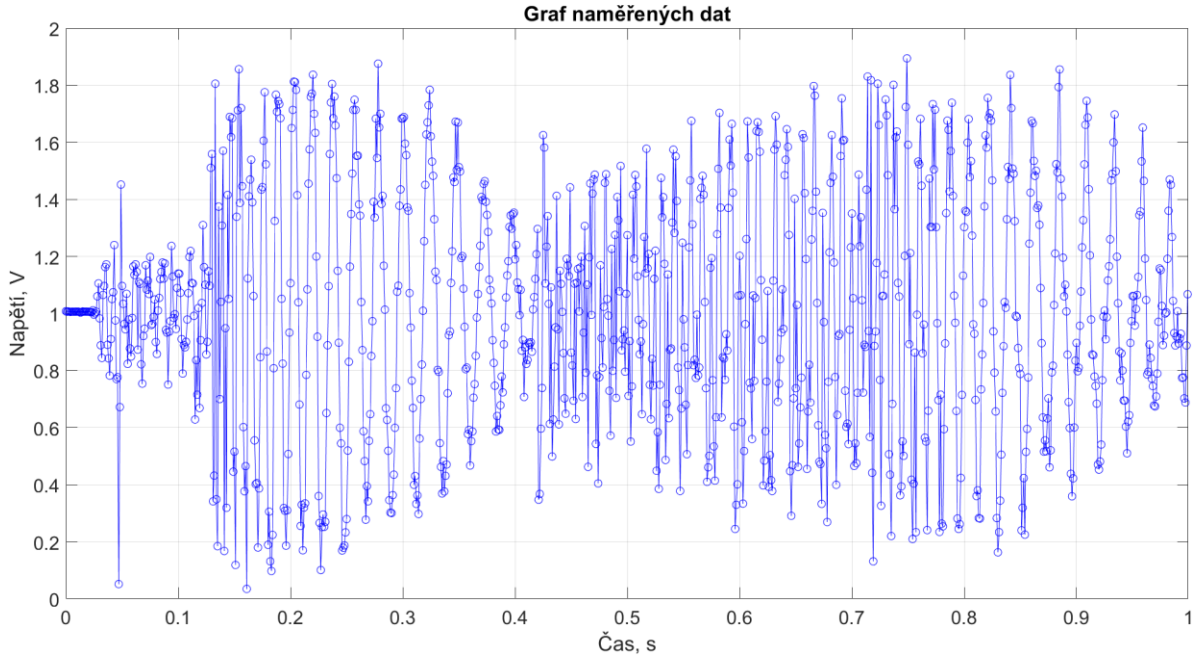
Ze zvukového souboru byla pomocí skriptu v MATLABu vytvořena data pro vygenerování signálu na D/A převodníku a změření A/D převodníkem. Obrázek 38 zobrazuje skript data_ze_zvuku. Zvukovými soubory byly zkrácené vyzváněcí tóny top_gun-short.mp3 a crab_rave-short.mp3. Soubory s uloženými daty pro měření byly sound1.txt a sound2.txt.

Vzorkovací perioda pro sound1 byla 8,9s a pro sound2 3,8s. U obou signálů byl signál navzorkován na 1000 vzorků.

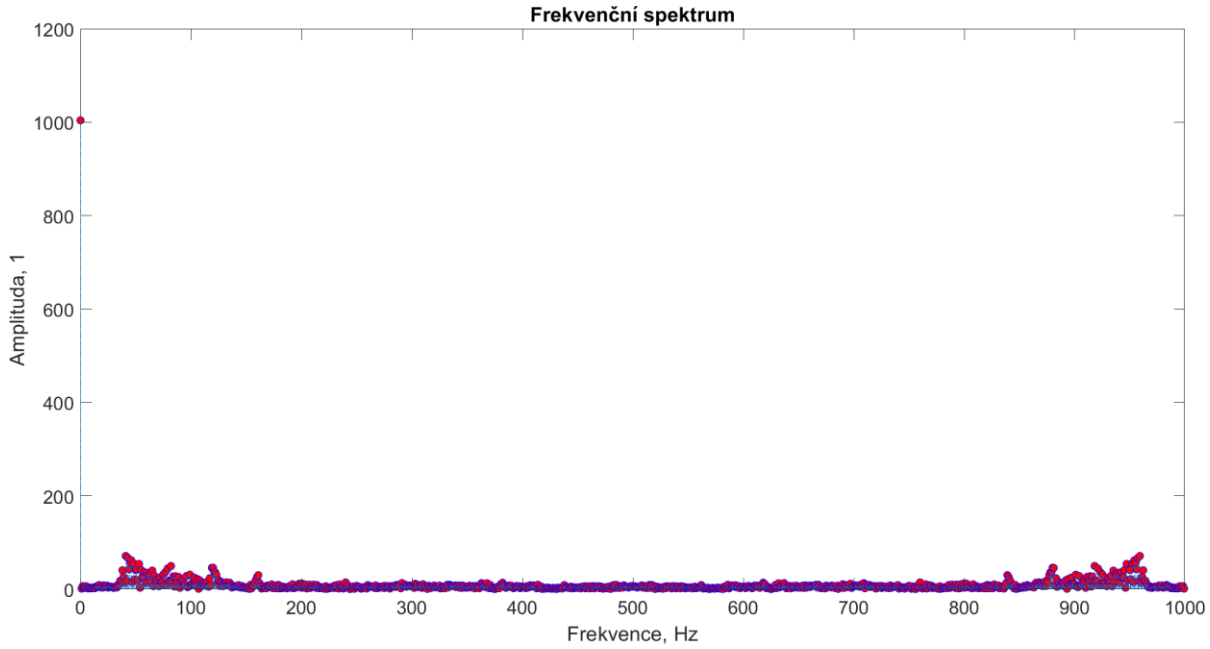
Obrázek 39 a Obrázek 42 zobrazují grafy naměřených hodnot. Obrázek 40, Obrázek 41, Obrázek 43 a Obrázek 44 zobrazují frekvenční a přiblížené frekvenční grafy. I zde je signál stejnosměrně posunut a první frekvenční složka má velkou amplitudu, příčiny byly vysvětleny v kapitole 3.2 Usměrňovač 50Hz.

```
[y, FS] = audioread('sounds\top_gun-short.mp3');
zvukovaDelka = length(y) / FS; %délka v sekundách
delka_ms = zvukovaDelka * 1000;
y = y+1;
y1 = y(:, 1);
% Uložení do souboru
fileID = fopen('sound2.txt', 'w');
fprintf(fileID, '%f\n', y1);
fclose(fileID);
%%sound(y,FS);
```

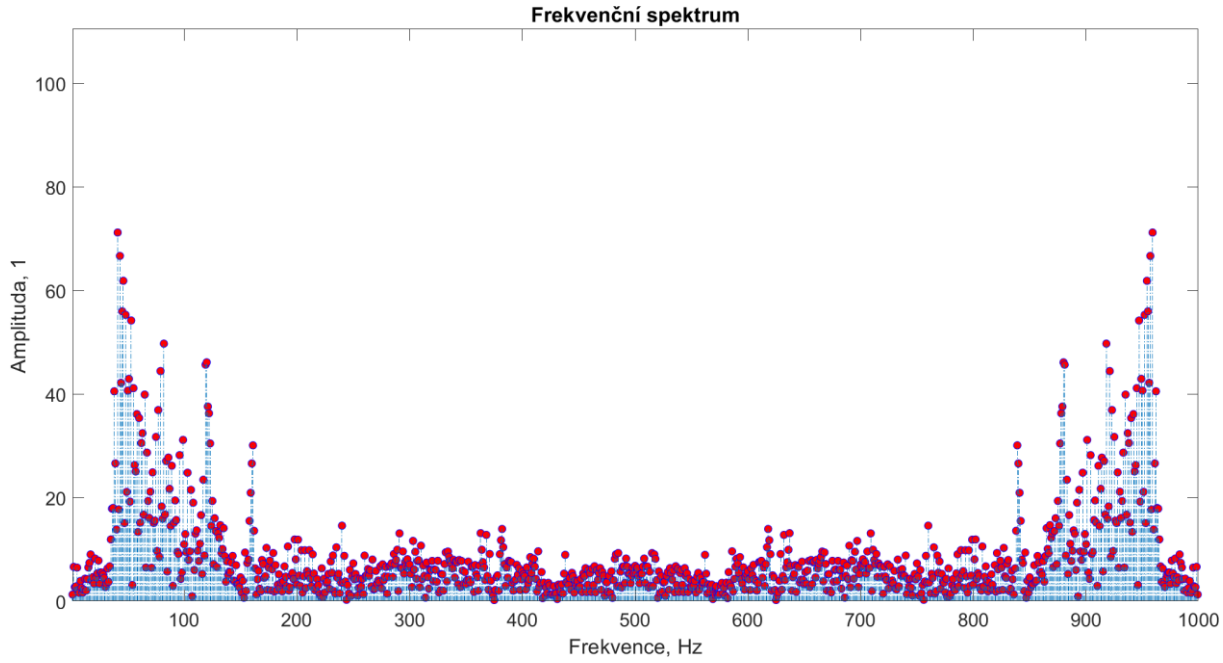
Obrázek 38: Ukázka ze skriptu data_ze_zvuku



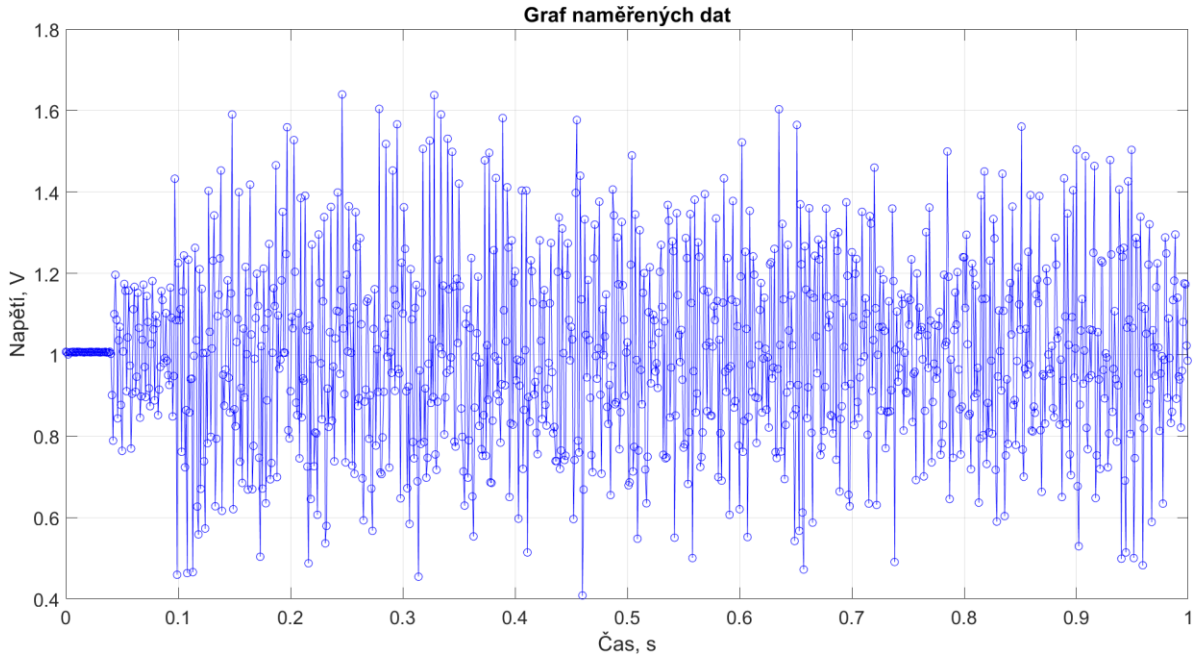
Obrázek 39: Graf naměřených hodnot zvukového signálu 1



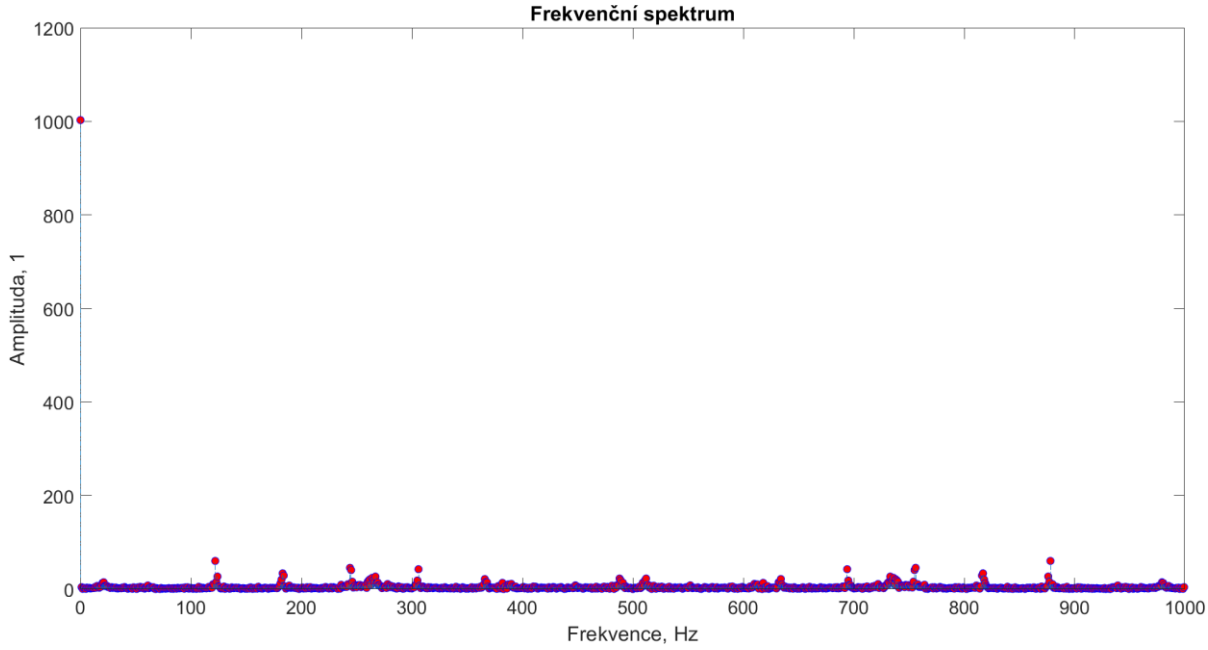
Obrázek 40: Frekvenční graf zvukového signálu 1



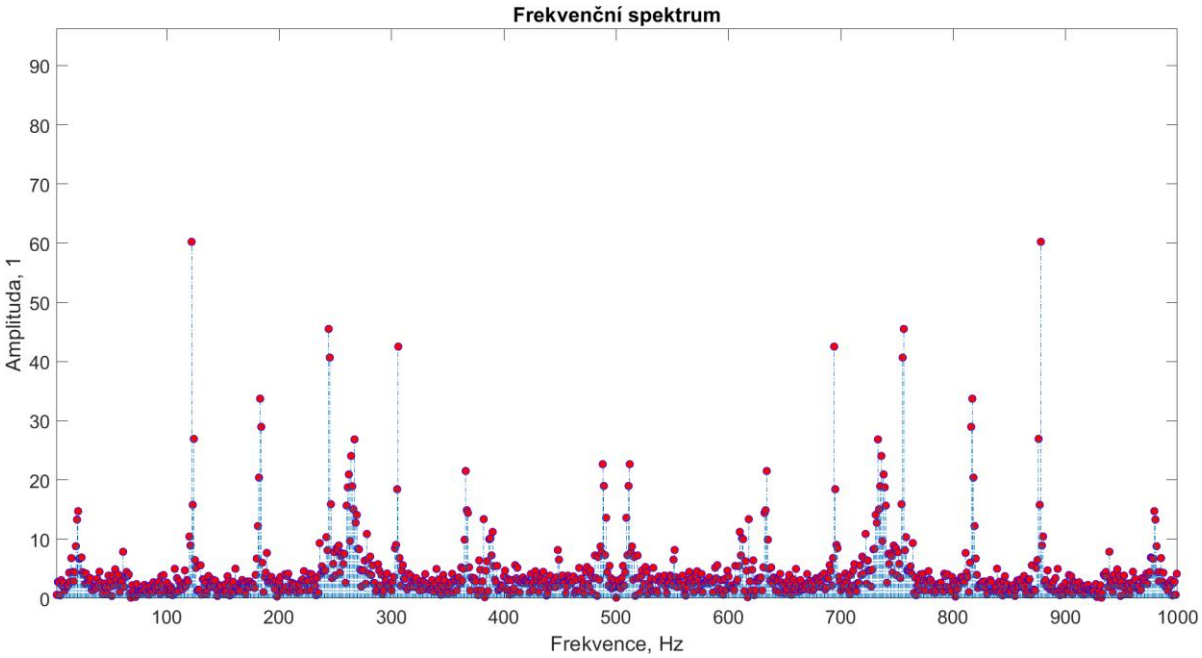
Obrázek 41: Přiblížený frekvenční graf zvukového signálu 1



Obrázek 42: Graf hodnot naměřených zvukového signálu 2



Obrázek 43: Frekvenční graf zvukového signálu 2



Obrázek 44: Přiblížený frekvenční graf zvukového signálu 2

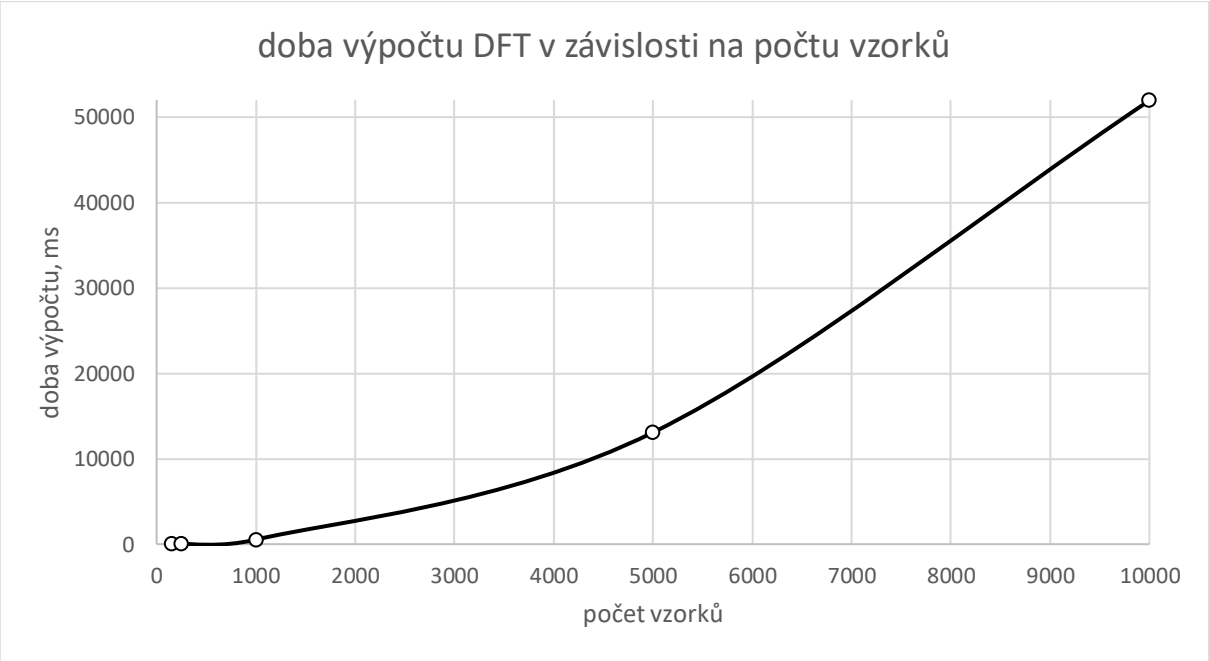
3.3 Výpočetní náročnost v závislosti na počtu vzorků

Bylo provedeno 5 měření pomocí knihovny chrono ve funkci computeDFT() s různým počtem zpracovaných vzorků. Tabulka 1 obsahuje počet vzorků signálu a dobu jejich zpracování v ms.

Tabulka 1: Výsledky měření výpočetní náročnosti

počet vzorků signálu	doba zpracování (ms)
150	17
250	49
1000	532
5000	13081
10000	51944

Po zpracování většího počtu vzorků bylo zjištěno, že náročnost výpočtu DFT je exponenciální ve vztahu doba výpočtu a počet vzorků. Obrázek 45 ilustruje tuto závislost na grafu.



Obrázek 45: Graf závislosti náročnosti výpočtu na počtu vzorků

ZÁVĚR

Práce se zabývá navržením a sestrojením zařízení, které provádí základní frekvenční analýzu s využitím počítače Raspberry PI a rozšiřujícího modulu ADC-DAC Pi Zero.

V teoretické části bylo vysvětlena Diskrétní Fourierova transformace a její výpočet. Bylo představeno Raspberry Pi a způsoby programování. Dále byl popsán rozšiřující modul ADC-DAC Pi Zero.

V praktické části byl sestrojeno rozhraní, které pomocí modulu ADC-DAC Pi Zero umožňuje měřit signály a chrání zařízení před přepětím. Dále byl v jazyce C++ napsán program, který měří signál a provádí diskrétní F. transf., výsledky zapisuje do souboru. Program má i funkci generování vstupního signálu pro testování.

V experimentální části bylo provedeno měření pro 5 různých signálů, z nichž 4 byly i vygenerované, a proběhlo vyhodnocení výsledků měření a výsledků Diskrétní Fourierovy transformace v MATLABu. Součástí vyhodnocení výsledků bylo vyhodnocení výpočetní náročnosti Diskrétní Fourierovy transformace pro různý počet naměřených vzorků.

Zvolené řešení ochrany proti přepětí pomocí operačního zesilovače ochrání rozšiřující modul, avšak neposkytuje možnost měřit signály až do 5V, jak bylo původně předpokládáno, ale pouze 3,9V. To je způsobeno napájením operačního zesilovače 5V, které odpovídá napájecímu napětí GPIO rozhraní počítače Raspberry PI. Toto omezení se projeví na změřeném signálu, protože v kombinaci s napěťovým děličem bude maximální amplituda peek-to-peek změřeného signálu 2.5V. Ostatní cíle práce byly splněny.

POUŽITÁ LITERATURA

ABELECTRONICSUK. *Mechanical Drawings*. Online. In: Abelectronics.co.uk. © 2012-2024. Dostupné z: https://www.abelectronics.co.uk/docs/stock/raspberrypi/adcdacpizero/adcdacpizero-mechanical_.webp. [cit. 2024-05-08].

ABELECTRONICSUK. *ABE_ADCDACPi.h*. Online. GITHUB, INC. Github.com. © 2024. Dostupné z: https://github.com/abelectronicsuk/ABElectronics_CPP_Libraries/blob/master/ADCDACPi/ABE_ADCDACPi.h. [cit. 2024-05-08].

DR. STEVE, Arar. *The Operation and Characteristics of Voltage-Mode R-2R DACs*. Online. In: Allaboutcircuits.com. March 11, 2019. Dostupné z: https://www.allaboutcircuits.com/technical-articles/voltage-mode-r2r-dacs-operation-and-characteristics. [cit. 2024-05-08].

MICROCHIP TECHNOLOGY INC. *MCP4822: 12-Bit DACs with Internal VREF and SPI™ Interface*. Online. © 2005. Dostupné také z: https://www.abelectronics.co.uk/viewpdf/mcp4822.

MICROCHIP TECHNOLOGY INC. *MCP3202: 2.7V Dual Channel 12-Bit A/D Converter with SPI Serial Interface*. Online. © 2006. Dostupné také z: https://www.abelectronics.co.uk/viewpdf/mcp3202.

O. SMITH, Julius. *Mathematics of the discrete fourier transform (DFT): with audio applications*. Online. 2nd ed. [s.l.]: W3K Publishing, 2007. ISBN 978-0-9745607-4-8. Dostupné z: https://ccrma.stanford.edu/~jos/mdft/Mathematics_DFT.html. [cit. 2024-05-07].

RASPBERRY PI LTD. *GPIO and the 40-pin header*. Online. In: Raspberrypi.com. © 2012-2024. Dostupné z: https://www.raspberrypi.com/documentation/computers/os.html#gpio-and-the-40-pin-header. [cit. 2024-05-08].

REJFEK, Luboš. *Elektrická měření. Pardubice – BELME Přednáška 3*. Univerzita Pardubice, FEI, 2021. Elektronický studijní materiál k předmětu Elektrická měření.

SMETANOVÁ, Romana; SHARMA, Shashank; PEERS, Nick a SHARMA, Mayank. *Nejlepší operační systémy pro Raspberry Pi v roce 2022*. Online. In: ZONEPI S.R.O. Blog.zonepi.cz. 2022. Dostupné z: https://blog.zonepi.cz/nejlepsi-operacni-systemy-pro-raspberry-pi-v-roce-2022/. [cit. 2024-05-12].

(Diskrétní)Fourierova transformace. Online. Univerzita Palackého v Olomouci. Dostupné z: http://apfyz.upol.cz/ucebnice/down/mini/fourtrans.pdf. [cit. 2024-05-12].

SEZNAM PŘÍLOH

Příloha 1: Zdrojové soubory navrženého plošného spoje

Příloha 2: Zdrojové soubory pro Raspberry PI

Příloha 3: Zdrojové soubory s příponou .m