Tester modulu měření vibrací

Bc. Adam Mynařík

Diplomová práce
2020

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

Ústav informatiky a umělé inteligence

Akademický rok: 2019/2020

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Adam Mynařík**
Osobní číslo:	**A18251**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Forma studia:	**Prezenční**
Téma práce:	**Tester modulu měření vibrací**
Téma práce anglicky:	**A Vibration Monitoring Module Tester**

Zásady pro vypracování

1. Prostudujte vlastnosti a principy funkce modulu pro měření vibrací leteckých motorů.
2. Vypracujte požadavky pro návrh testeru tohoto modulu.
3. Na základě požadavků navrhněte architekturu testeru pro modul měření vibrací.
4. Navrhněte a realizujte plošný spoj elektroniky testeru.
5. Implementujte firmware testeru a SW nástroj pro jeho ovládání přes PC.

Rozsah diplomové práce:
Rozsah příloh:
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. PINKER, Jiří. Mikroprocesory a mikropočítače. 1. vyd. Praha: BEN – technická literatura, 2004, 159 s. ISBN 80-7300-110-1.
2. VÁŇA, Vladimír. ARM pro začátečníky. 1. vyd. Praha: BEN – technická literatura, 2009, 195 s. ISBN 978-80-7300-246-6.
3. BODEN, Fritz. Advanced in-flight measurement techniques. New York: Springer, 2013, 344 s. ISBN 978-3-642-34738-2.
4. BUZDUGAN, Gheorghe. Vibration measurement.Netherlands: Springer, 2010, 345 s. ISBN 978-90-481-8287-9.
5. BARR, Michael a Anthony J. MASSA. Programming embedded systems: with C and GNU development tools. 2nd ed. Sebastopol: O'Reilly, 2006. ISBN 978-0-596-00983-0.

Vedoucí diplomové práce: **Ing. Tomáš Dulík, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **28. listopadu 2019**
Termín odevzdání diplomové práce: **15. května 2020**

L.S.

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

Ve Zlíně dne 9. prosince 2019

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomové práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky. Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 5.8.2020

Bc. Adam Mynařík, v.r

podpis autora

ABSTRAKT

Cílem této práce je vytvořit hardwarovou a softwarovou implementaci testeru modulu měření vibrací turbovrtulových motorů. První část diplomové práce popisuje problematiku měření a monitorování vibrací leteckých motorů v širším kontextu, včetně historie vývoje leteckých motorů. V práci je pak představena specifická aplikace - modul pro monitorování vibrací turbovrtulového motoru včetně popisu konstrukce daného motoru. Na závěr teoretické části je navržena architektura testeru modulu měření vibrací na základě vstupních požadavků. V praktické části práce je popsána implementace funkčního základu testeru modulu měření vibrací jako generátoru dvou nezávislých zdrojů vibrací. Dále je navržena architektura testeru, která byla posléze realizována jako deska plošných spojů vestavěná do 3D tištěné krabičky. Práce dále popisuje implementaci firmware testeru a také softwarový nástroj pro jeho ovládání. Závěrem je porovnán výstup modulu měření vibrací při generovaní vstupních signálů z testeru oproti generátoru signálů.

Klíčová slova: Vibrace, Letecký motor, Modul měření vibrací, Tester

ABSTRACT

The aim of this project was to create hardware and software implementation of a tester for vibration monitoring module for an aircraft turboprop engine. The first part of the thesis describes principles of vibrations monitoring in broader context, including the history of aircraft engines. The vibration monitoring module used in this diploma project is also described here. At the end of the theoretical part, the design of my tester for the vibration monitoring module is discussed, based on the requirements for the module. The second part of thesis describes implementation of two channels vibration signals generator as the basic component of the tester. Then, the tester architecture and its implementation as a printed circuit board along with a 3D-printed box is described. The implementation of the tester firmware, as well as desktop software tool for tester configuration and control is documented as well. Finally, the output of the vibration measurement module when generating input signals from the tester was compared against a signal generator.

Keywords: Vibration, Aircraft engine, Vibration measurement module, Tester

Děkuji panu vedoucímu mé diplomové práce, Ing. Tomáši Dulíkovi, Ph.D., za cenné rady během vypracovávání této práce. A také rodičům a přítelkyni za podporu.

Firmě UNIS, a.s. děkuji za poskytnutí zajímavého tématu diplomové práce a potřebného vybavení a materiálu.

Tato práce byla podpořena Ministerstvem průmyslu a obchodu (MPO) v rámci grantového projektu „Pokročilé technologie modulárních řídicích a diagnostických systémů“, č. projektu: FV20043.

OBSAH

1 Úvod

Monitorování vibrací je nezbytným prostředkem k zajištění spolehlivosti a bezpečnosti leteckých motorů. Poskytuje možnost sledování aktuálního rozpoložení stroje a vyvarování se vibracím zhoršujícím stav stroje. Získaná data se mohou použít pro diagnostiku motoru a jeho následnou údržbu, díky tomu se lze vyvarovat neplánovaným odstávkám a následným nákladným opravám. Monitorování vibrací tedy výrazně přispívá k životnosti pohonné jednotky.

Cílem této diplomové práce bylo vytvoření zařízení umožňující testovaní modulu měření vibrací. V této práci se nejprve zabývám vibracemi a jejich teoretickým popisem. Dále také stručnou historií vývoje leteckých motorů s danými významnými vynálezci. Je zde popsáno z jakých částí se turbovrtulový motor skládá. Dále také vlastnosti a principy funkce modulu pro měření vibrací. V práci jsou vypracované požadavky pro návrh testeru tohoto modulu a následně popsaný výběr vhodného HW na základě požadavků pro tester.

Praktická část začíná implementací generátoru dvou nezávislých zdrojů vibrací v programu Scilab. Dále je v práci popsána navržená architektura testeru pro modul měření vibrací, včetně výběru konkrétních komponent a popisu jejich propojení.

Výstupem praktické části je navržená, osazená a oživená deska plošných spojů spolu s následným vytvořením vhodné krabičky pro tester. Je zde popsáná implementace firmware testeru a také SW nástroje pro jeho ovládaní. Na závěr práce jsou porovnány výstupní hodnoty modulu měření vibrací při generovaní vstupních signálů pomocí testeru a generátoru signálů.

I. TEORETICKÁ ČÁST

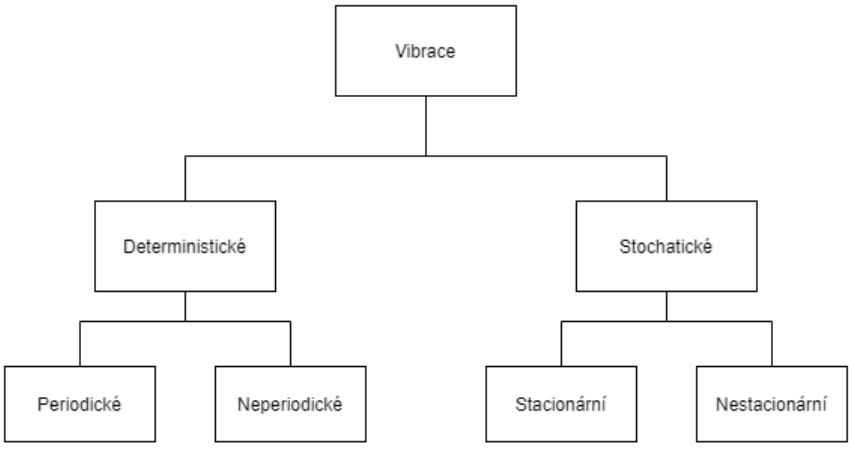
2 Vibrace

Studie o vibracích se týká oscilačních pohybů těles a sil s nimi spojených. Všechna tělesa, která mají hmotu a pružnost, jsou schopna vibrace. Existují dvě obecné třídy vibrací: volné a nucené. [1]

K volným vibracím dochází, když systém kmitá působením sil, které jsou vlastní systému, a když chybí vnější působící síly. Systém při volných vibracích bude vibrovat na jedné nebo více svých přirozených frekvencích, což jsou vlastnosti dynamického systému stanovené jeho rozložením hmoty a tuhosti. [1]

Vibrace, ke kterým dochází při buzení vnějšími silami, se nazývají nucené vibrace. V tomto případě vnější síla nepřetržitě dodává energii systému. Když je buzení oscilační, je systém nucen vibrovat na budicí frekvenci. Pokud se frekvence excitace shoduje s jednou z přirozených frekvencí systému, dochází k rezonanci a může dojít k nebezpečně velkým kmitáním. [1]

Vibrační systémy jsou do jisté míry vystaveny tlumení, protože energie je rozptylována třením a jinými odpory. Pokud je tlumení malé, má velmi malý vliv na přirozené frekvence systému, a proto se výpočty pro přirozené frekvence obvykle provádějí bez žádného tlumení Na druhé straně má tlumení velký význam při omezování amplitudy kmitání při rezonanci. [1]

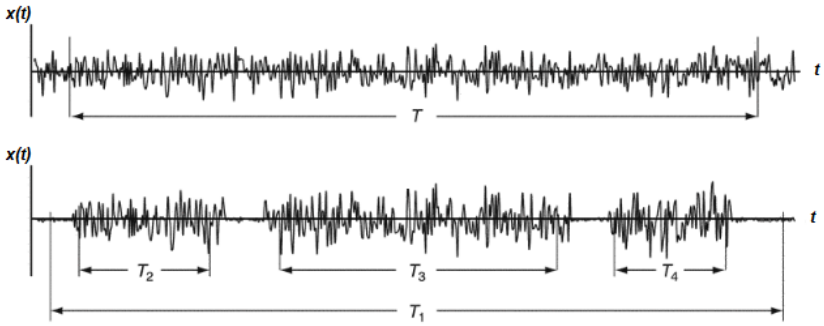


Obr. 2.1 Rozdělení vibrací podle časového průběhu [2]

Všechny pohyby, které lze popsat explicitním matematickým vztahem se nazývají deterministické vibrace. Lze tedy určit jejich okamžitou hodnotu v čase, dříve než daný časový okamžik nastane. [1]

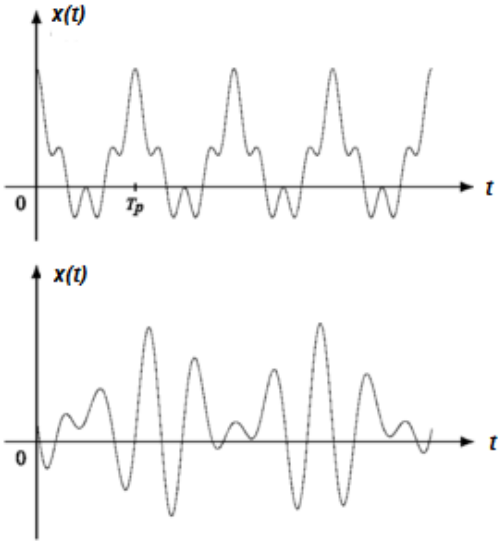
Na rozdíl od deterministických vibracích, pohyby, jejichž okamžitá hodnota není specifikována explicitním matematickým vztahem, nelze tedy určit okamžitou hodnotu v čase, se nazývají náhodné vibrace. [1]

Stacionární proces je stochastický proces, jehož statistické vlastnosti jako směrodatná odchylka, střední hodnota a autokorelace, se v průběhu času nemění. Nestacionární proces je signál, jehož statistické vlastnosti se mění s časem. Je obtížné zpracovat nestacionární proces. Velmi často jsme nuceni rozdělit proces na segmenty, z nichž každý se považuje za stacionární. [3]



Obr. 2.2 Stacionární a nestacionární průběh [3]

Když se pohyb opakuje ve stejných intervalech času T, nazývá se periodický pohyb. Pokud je pohyb určen časovou funkcí $x(t)$, pak jakýkoli periodický pohyb musí splňovat vztah $x(t) = x(t + T)$. [3]



Obr. 2.3 Periodický a neperiodický průběh [4]

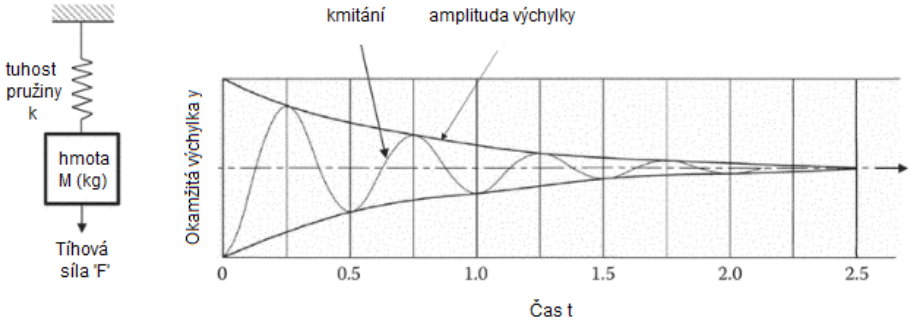
2.1 Harmonický pohyb

Nejjednodušší forma periodického pohybu je harmonický pohyb, tedy sinusová či kosinová funkce. Většina vibračních systémů obsahuje pohyby, které jsou téměř harmonické. To může být demonstrováno hmotou zavěšenou na pružině. Pokud bude hmota přemístěna ze své klidové polohy a uvolněna, bude kmitat nahoru a dolů. [5]

Harmonický pohyb může být popsán rovnicí:

$$x(t) = x_o \sin \omega t \tag{2.1}$$

kde $x(t)$ je pozice částice nebo těla v čase měřená od referenční polohy, amplituda posunu x_o a ω uhlová frekvence. [5]



Obr. 2.4 Vychýlení hmoty a výsledná křivka [5]

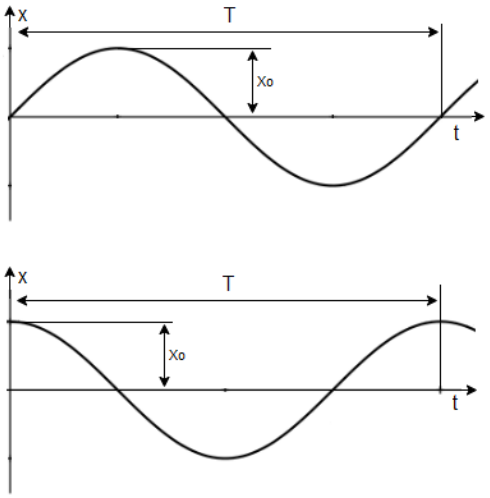
Vztahy mezi periodou $x(t)$, frekvencí f a úhlovou frekvencí ω jsou:

$$\omega = 2\pi f = \frac{2\pi}{T} \tag{2.2}$$

$$f = \frac{1}{T} = \frac{\omega}{2\pi} \tag{2.3}$$

$$T = \frac{1}{f} = \frac{2\pi}{\omega} \tag{2.4}$$

Perioda T opakování oscilace se obvykle měří v sekundách, frekvence f v cyklech za sekundu a ω úhlová rychlost, která je úměrná frekvenci, v radiánech za sekundu. [5]



Obr. 2.5 Harmonický pohyb se stejnou amplitudou a frekvencí, ale s jiným počátkem času t

Podle obrázku 2.5 můžeme určit, že se stále jedná o harmonický pohyb i když jsme začali měřit v jiném časovém okamžiku. Toto relativní posunutí se nazývá fázový rozdíl, jeho velikost udávaná v radiánech se nazývá fázový úhel mezi křivkami. Pro rovnici $x(t) = x_o \sin(\omega t + \varphi_o)$ kde za φ_o dosadíme $\pi/2$, lze vidět, že můžeme harmonický pohyb zapsat více způsoby. [5]

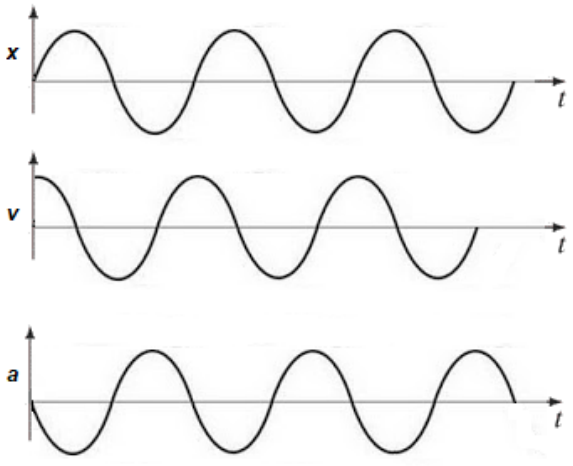
$$x = x_o \sin(\omega t + \frac{\pi}{2}) = x_o \cos(\omega t) \tag{2.5}$$

Pohyb se v mechanice popisuje pomocí výchylky, rychlosti nebo zrychlení. Rychlost vibrací popisuje rychlost změny výchylky a je první derivací výchylky podle času. Je-li posun vibrační částice dán rovnicí $x(t) = x_o \sin \omega t$ bude rychlost částice:

$$v = \frac{dx}{dt} = x_o \omega cos(\omega t + \varphi_o) \tag{2.6}$$

Zrychlení popisuje změnu rychlosti a je tedy druhou derivací výchylky podle času. [5]

$$a = \frac{dv}{dt} = x_o \omega^2 sin(\omega t + \varphi_o) \tag{2.7}$$



Obr. 2.6 Periodické harmonické signály [7]

Lze si všimnout, že kdykoli dosáhla hodnota posunu maximální hodnoty, rychlost v tom samém okamžiku nabývá hodnoty nula. Zrychlení dosáhne maximální úrovně, když je rychlost nulová. Nejvýhodnějším parametrem je určení rychlosti vibrací zvláště pro diagnostiku opotřebení pravidelně se rotujících částí. Při měření výchylky či zrychlení by bylo potřeba zaznamenávat i otáčky stroje, při jakých byly tyto hodnoty naměřeny. [7]

3 Historie létaní

Mezi první průkopníky, kteří se zasloužili o pokrok v oblasti letectví, patří sir Isaac Newton, který objevil a sepsal jeho tři zákony pohybu v roce 1686 v díle Principia Mathematica Philosophiae Naturalis. Zaměřil se na to, jaké síly působí na pohybující se předmět a ovlivňují jeho pohyb. [8]

Henri Giffard postavil vzducholoď, která v roce 1852 byla poháněna parním motorem o výkonu tří koní, jenž poháněl třílistou vrtuli. Jednalo se o první řízený let přepravující cestující na světě, kdy uletěl vzdálenost 27 km. [9]

Félix Du Temple v roce 1874 provedl krátký let dolů z kopce pomocí letadla, které navrhl a bylo poháněno parním strojem. Byl to spíše skok než opravdový let, ale někteří historici to považují za první poháněný let. [10]

Samuel Langley vyráběl letadla poháněná parními motory. V roce 1896 jeho bezpilotní letoun uletěl 1200 metrů po vypuštění katapultem z lodi. Tato vzdálenost byla několikrát delší než předešlé pokusy s letadly těžší než vzduch. [11]

Bratři Wrightovi, Willbur a Orville, se pokoušeli navrhnout křídla pro let a inspirovali se u ptáků, kde si všimli, že ptáci sklání křídla pro kontrolu rovnováhy při letu a navrhli koncept zvaný "deformace křídla". Dále přidali pohyblivé kormidlo a vyvinuli ovládaní letadla okolo tří os. Dne 17. prosince 1903 měli několik pokusů o let a nejlepší výsledek se povedl Willburovi, který s benzínovým motorem pohánějící vrtuli o výkonu dvanáct koní za 59 vteřin uletěl vzdálenost 259 metrů. Byl to první skutečný řízený let poháněný motorem v historii. [12]

Až do konce 30. let se používal plynový spalovací motor s vrtulí jako jediný prostředek používaný k pohánění letadel. Frank Whittle napsal během svých studií práci, kde tvrdil, že ve vysokých nadmořských výškách by byly pístové motory a vrtule nedostatečné pro dlouhé a rychlé lety. V roce 1930 navrhl první model proudového motoru. Letadlo s tímto motorem poprvé úspěšně létalo v roce 1941, kde se motor skládal z vícestupňového kompresoru, spalovací komory, jednostupňové turbíny a trysky. [13]

Hans von Ohain pracoval v Německu ve stejné době na podobném designu jako Whittle v Anglii. Po studiích se připojil k firmě Heinkel, kde z počátku prováděl své experimenty v tajnosti. Postupně vyústily v testovací zkoušky a prvním letounem, který úspěšně letěl s motorem s tímto typem pohonu byl letoun Heinkel He 178. Dne 27. srpna roku 1939 toto letadlo provedlo první úspěšný let s proudovým pohonem. [14]

4 Skladba proudového motoru

Většina moderních leteckých motorů s výkonem nad 300 kW vychází z koncepce proudového motoru, který obsahuje komponenty uvedené níže.

Ventilátor je první komponentou v motoru, díky němuž se nasává velké množství vzduchu. Většina lopatek ventilátoru je vyrobena z titanu. Nasátý vzduch je rozdělen na dvě části. Jedna část pokračuje „jádrem“ neboli středem motoru, kde na něj působí jiné součásti motoru. Druhá část obchází jádro motoru. Prochází potrubím, které obklopuje jádro k zadní části motoru, kde produkuje velkou část síly, která pohání letoun dopředu. Tento chladnější vzduch přispívá k chlazení a tichosti motoru a zvyšuje jeho tah. [8]

Kompresor je první součástí jádra motoru. Kompresor se skládá z ventilátorů s mnoha lopatkami připojenými k hřídeli spolu s turbínou, která kompresor pohání. Kompresor stlačuje vzduch, který do něj vstupuje, což má za následek zvýšení tlaku vzduchu. Výsledkem je zvýšení energetického potenciálu vzduchu. Stlačený vzduch je tlačen do spalovací komory. Dřívější proudové motory používaly odstředivé kompresory, modernější proudové motory používají výhradně axiální kompresory kvůli jejich vyšší kompresní účinnosti. [8]

Ve spalovací komoře se vzduch mísí s palivem a poté se zapálí. Je zde několik trysek pro vstřikování paliva do proudu vzduchu. Směs vzduchu a paliva se vznítí, hoření generuje proud plynů s vysokou energií: palivo hoří s kyslíkem ve stlačeném vzduchu a vytváří horké expandující plyny. Vnitřek spalovací komory je vyroben z materiálů, které poskytují dostatečnou tepelnou odolnost. Teplota plynů při spalovacím procesu dosahuje průměrně 1500°C, ale může dosáhnout až 2100 °C. [8]

Vysokoenergetický proud plynů vycházející ze spalovací komory prochází turbínou a roztáčí její lopatky. Turbína je spojena hřídelí s kompresorem a ventilátorem v přední části motoru. Rotace turbíny z vysokoenergetického proudu odebírá určitou energii, která se používá k pohonu ventilátoru a kompresoru. Zbylá energie je využita pro tah motoru. [8]

Tryska je výfuková trubka motoru. Toto je část motoru, která ve skutečnosti vytváří tah pro letadlo. Energeticky ochuzený proud vzduchu, který prošel turbínou, kromě chladnějšího vzduchu, který obcházel jádro motoru, při výstupu z trysky vytváří sílu, která pohání motor a tím i letadlo dopředu. Trysce může předcházet směšovač, který kombinuje vzduch o vysoké teplotě přicházející z jádra motoru se vzduchem o nízké teplotě, který prošel jen ventilátorem. Směšovač pomáhá zajistit tišší provoz motoru. [8]

4.1 Turbovrtulový motor

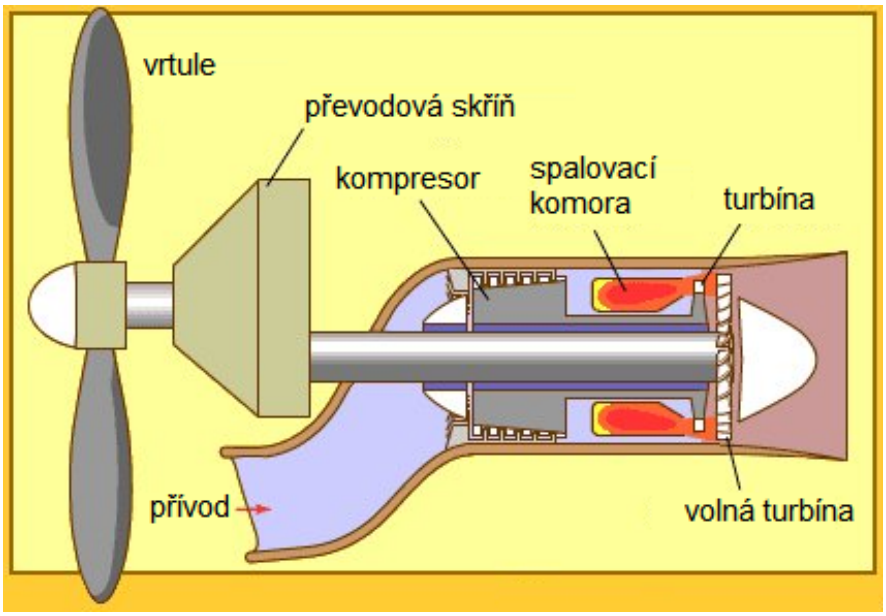
Turbovrtulový motor (Turboprop) je kombinace vrtulového a proudového motoru, kde se energie proudového motoru používá k pohonu vrtule. [15]

Pro potřebu návrhu modulu měření vibrací lze popsat turbovrtulový motor jako mechanickou sestavu tří částí:

- Gas generator(GG) - kompresor a turbína
- Power turbine(PT) - volná turbína
- Vrtule

Jádro motoru je velmi podobné proudovému motoru: obsahuje kompresor, spalovací komoru a turbínu. Turbína spojená s kompresorem na stejné hřídeli tvoří tzv. gas generator. Na výstupu z turbíny však prochází plyn ještě jednou turbínou, ta se nazývá volná turbína. Většina energie výfukových plynů se používá pro poháněnní volné turbíny. Mezi kompresorem a volnou turbínou neexistuje pevná mechanická vazba. Otáčky volné turbíny a kompresoru jsou tedy na sobě nezávislé, nemusí se v určitém okamžiku shodovat. [15]

Vysokoenergetický proud vzduchu vycházející ze spalovací komory jde do turbíny, což způsobuje otáčení lopatek kompresoru. Uvnitř hřídele, která spojuje hlavní turbínu a kompresor, prochází hřídel volné turbíny. Tato hřídel je spojená s převodovkou, která zajišťuje otáčení vrtule s redukovanými otáčkami. Převodovka je spojena s vrtulí, jenž vytváří většinu tahu. [15]



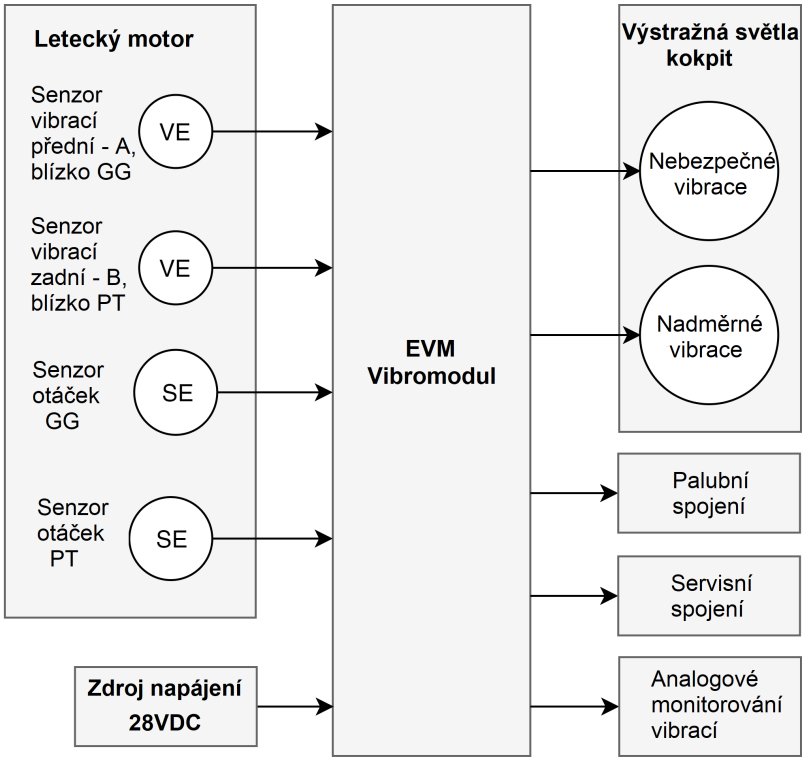
Obr. 4.1 Skladba turbovrtulového motoru [16]

Historie vývoje turbovrtulových motorů začínala přímočarým konceptem, kde vrtule s převodovkou byla připojena přímo k turbíně. Toto uspořádaní se ukázalo jako méně efektivní než použití oddělené hřídele pro vrtuli, která je poháněná volnou turbínou, jejíž výstupní výkon a rychlost lze efektivněji přizpůsobit. Jako jedinou hlavní překážku by se dalo označit spojení hřídele s vrtulí, otáčky turbíny jsou totiž asi 10 krát větší, než jsou otáčky vhodné pro vrtuli. [17]

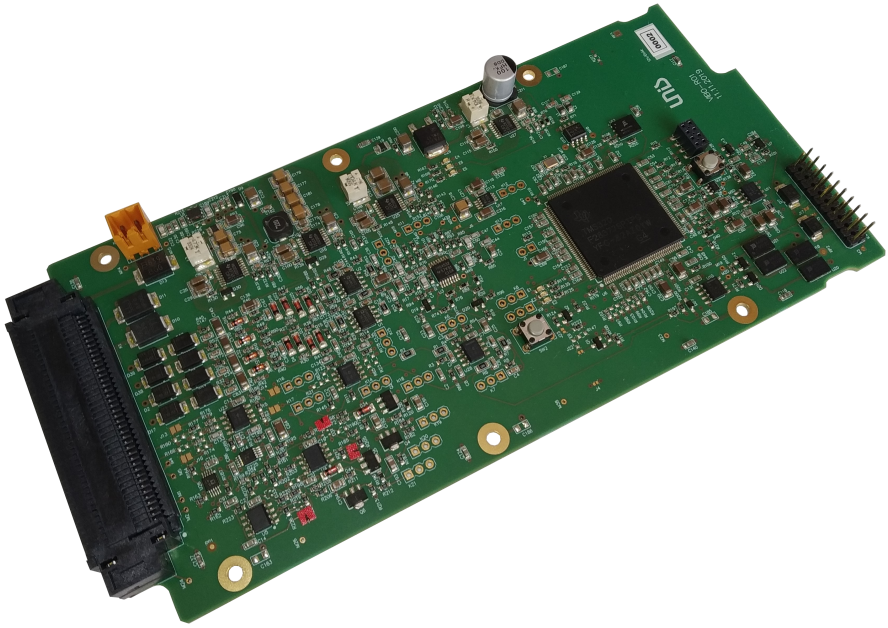
Je tedy zapotřebí převodová jednotka mezi vrtulí a turbínou. Hmotnost takových jednotek poněkud snížila přitažlivost tohoto motoru, avšak dostatečné výhody pro kombinaci proudového motoru a vrtule stále přinášejí jejich použití pro letadla, která nemusí létat rychleji než 555 kilometrů za hodinu. Jelikož se vrtule stávají méně efektivní pro vyšší rychlosti letadla, je tento typ motoru používán pro kratší a nízkonákladové letové trasy. [17]

5 Modul měření vibrací

Vibrační diagnostika výrazně přispívá k prodloužení životnosti pohonné jednotky, proto se stává standardní součástí řídicích jednotek motorů. Hlavní funkce modulu měření vibrací je neustálé měření vibrací po dobu letu a nastavení výstrahy do pilotní kabiny při případném překročení povolené amplitudy vibrací. Modul měření vibrací (dále jen „Vibromodul“) by měl také sloužit k získávaní dat pro diagnostiku motoru a jeho následnou údržbu. Tato data by měla být získávána při pozemních či letových zkouškách. Díky informacím, které dostane pilot skrz výstrahu o překročení povolených vibrací, může snížit otáčky motoru nebo jej vypnout a tím zamezit poškození nebo destrukci motoru. Architektura Vibromodulu, pro nějž mám vytvořit tester, je popsána v následujících podkapitolách a obrázcích. [15]



Obr. 5.1 Blokové schéma modulu měření vibrací [15]



Obr. 5.2 Fyzická podoba Vibromodulu [15]

5.1 Snímače vibračního zrychlení

Jednotka Vibromodulu provádí měření vibrací v reálném čase pomocí dvou snímačů vibračního zrychlení. Jeden ze snímačů je umístěn na přední části motoru v oblasti, kde se nachází převodovka. Tento snímač slouží k určení vibrační rychlosti vrtule a vibračního zrychlení kompresoru ve spojení s turbínou („Gas Generator“, dále jen GG). [15]

Druhý ze snímačů je umístěn na zadní části motoru v oblasti, kde se nachází volná turbína („Power Turbine“, dále jen PT). Tento snímač slouží k měření vibračního zrychlení volné turbíny. Měření vibračního zrychlení probíhá ve frekvenčním pásmu nastaveném podle aktuální hodnoty otáček kompresoru s turbínou, či volné turbíny. [15]

Snímač umístěný v přední části může být vystaven teplotám až 250°C. Druhý snímač umístěný v zadní části je vystaven teplotám až 400°C. Na základě těchto specifikací jsou použity speciální vysokoteplotní piezokeramické snímače bez zabudované elektroniky. [15]

Tyto snímače poskytují data s vysokou přesností v krátkém čase. Běžně používaný senzor při testování vibrací je piezoelektrický snímač zrychlení. Piezokeramické snímače generují výstup v podobě elektrického náboje, který je pro další zpracování nutné převést na napěťový signál pomocí nábojového zesilovače. [15, 18]

5.2 Snímače otáček

Analýza signálu vibračního zrychlení popsaná výše vyžaduje znalost aktuálních rychlostí otáčení hřídelí GG a PT, protože modul měření vibrací vyhodnocuje amplitudy vibrací jen na dvou diskrétních frekvencích, odpovídajících frekvencím otáčení těchto dvou hřídelí. Pro měření otáček se používají tachogenerátory pracující na indukčním principu, kde frekvence a amplituda závisí na rychlosti otáčení. Proto pro převod signálu tachogenerátoru do digitální podoby je potřeba analogový obvod tvarující impulzy. Tento obvod také zajišťuje diagnostiku stavu senzorů a jejich možné poruchy. Kmitočet pulzů tachogenerátoru se pohybuje od 100 Hz do 3000 Hz. Jeden ze snímačů měří otáčky pro volnou turbínu, druhý pro kompresor ve spojení s turbínou. Otáčky vrtule jsou počítány z otáček volné turbíny převodovým poměrem použité převodovky. [15]

5.3 Komunikační rozhraní

Vibromodul obsahuje komunikační rozhraní CAN bus. Toto rozhraní je určeno k přenášení naměřených hodnot do řídicí jednotky pomocí protokolu CANAerospace, kde v každém datovém kanálu je zasílána vypočítaná velikost amplitudy příslušné spektrální složky vibrací. Současně protokol slouží k přenosu základních diagnostických dat. Sběrnice je obousměrná, takže umožňuje i ovládat či nastavovat Vibromodul nebo řídicí jednotku motoru. Pro účely servisního typu je Vibromodul vybaven také sériovou sběrnicí RS-485. [15]

5.4 Výstraha

Vibromodul disponuje jednoduchým spínacím výstupem pro signalizaci překročení limitů u hodnot vibrací. Tímto způsobem bude signál předán do pilotní kabiny či nadřízenému systému. [15]

5.5 Napájení

Vibromodul bude napájen z palubní sítě letadla, tj. napětím +28V. Spolehlivá činnost modulu musí být zaručena ve všech testovacích režimech. [15]

5.6 Algoritmus pro výpočet vibrací

Pro zpracování výpočetního algoritmu je použit digitální signálový procesor (DSP). Pro zpracování digitálního signálu je použit algoritmus založený na diskrétní fourierově transformaci (DFT) dle následujícího vztahu: [15]

$$S(k) = \sum_{n=0}^{N-1} s(n)c_k(n) \tag{5.1}$$

Kde $s(n)$ je n-tý vzorek vstupního signálu, $S(k)$ je k-tý bod ve frekvenčním spektru vstupního signálu a $c_k(n)$ jsou vzorky komplexního sinusoidu při frekvenci k/N, definované:

$$c_k(n) = e^{-jk\frac{2\pi}{N}n} = \cos(k\frac{2\pi}{N}n) - j\sin(k\frac{2\pi}{N}n) \tag{5.2}$$

Algoritmus zpracovává první harmonickou složku vibrací pro hřídel volné turbíny (PT) a první harmonickou složku pro vibrace hřídele kompresoru s turbínou (GG). Výpočty DFT na DSP dle vztahu (5.1) jsou velmi efektivní, protože využívají speciální instrukci MAC - „Multiply And Accumulate“. [15]

Dvě komplexní sinusiody $c_{kGG}(n)$ a $c_{kPT}(n)$ jsou generovány z tabulky. Frekvence každého sinusoidu k_{GG}/N a k_{PT}/N jsou proměnné - jsou odvozené od senzorů pro otáčky hřídelí volné turbíny (PT) a kompresoru s turbínou (GG). [15]

Algoritmus je tedy schopný spočítat hodnoty vibrací dvou nezávislých senzorů vibrací: senzoru umístěného u volné turbíny (PT) a senzoru u kompresoru s turbínou (GG). Dohromady jsou tedy výsledky zasílány na pět kanálů, kde pátý kanál slouží pro hodnoty vibrací vrtule. [15]

6 Požadavky pro návrh testeru tohoto modulu

Požadavkem pro návrh testeru je generovaní dvou nezávislých signálů, které simulují výstup otáčkoměrů volné turbíny (PT) a kompresoru spojeného s turbínou (GG). Dále generování analogového signálu, simulujícího výstup senzoru vibrací umístěného na motoru. Tester musí být možné ovládat přes PC pomocí softwarového nástroje, který umožní měnit některé parametry těchto signálů. Zároveň bude softwarový nástroj přijímat výsledná data úrovní vibrací z Vibromodulu a ověřovat, zda na straně Vibromodulu dochází ke správnému zpracovaní signálů generovaných testerem.

Požadavek pro signály simulující výstup otáčkoměrů je takový, aby tyto signály byly na sobě časově nezávislé a byly co nejvíce podobné signálům, které tyto senzory spolu s tvarovacím obvodem generují. To bude dosaženo generováním obdélníkového signálu pomocí pulzně šířkové modulace.

Další požadavek pro generování analogového signálu simulujícího výstup senzoru vibrací motoru je, aby výsledný signál byl složen ze dvou harmonických signálů, kde parametr frekvence prvního signálu budou otáčky volné turbíny a parametr frekvence druhého signálu zase otáčky turbíny kompresoru s turbínou a také, aby šlo jednotlivým signálům měnit velikost amplitud.

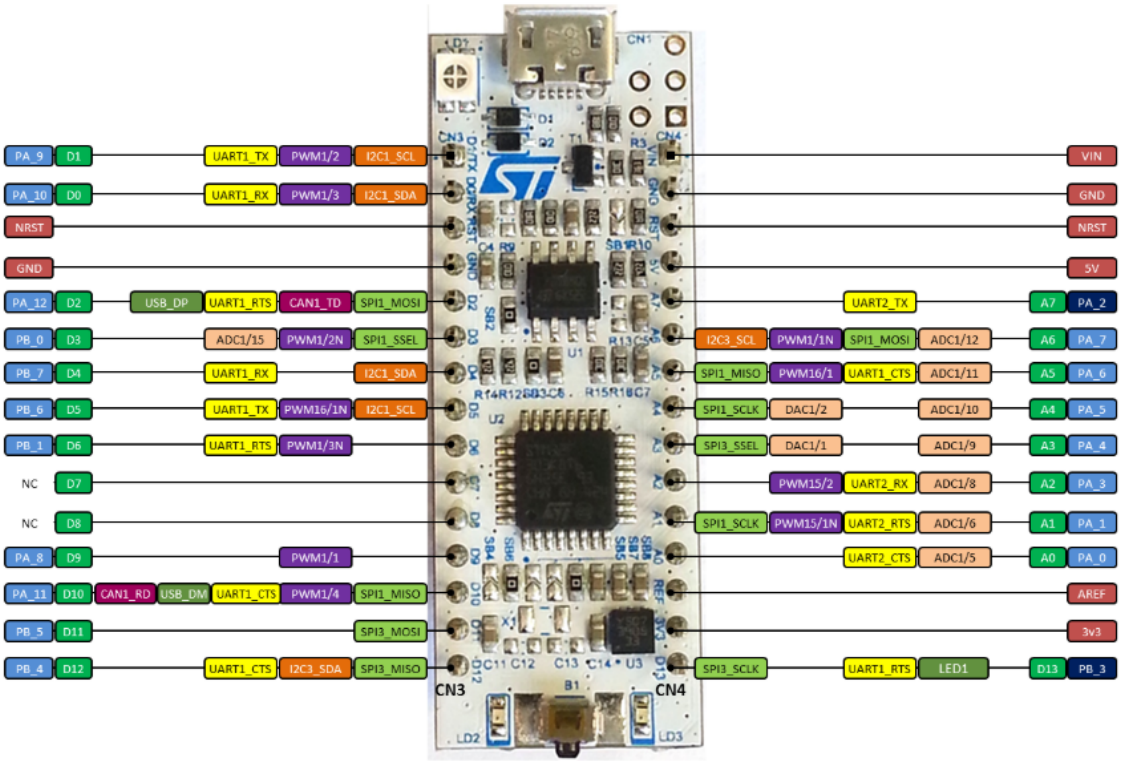
Dále je potřeba navrhnout softwarový nástroj pro tester. Zde jsou požadavky, aby tester bylo možné ovládat pomocí softwarového nástroje z PC. Je potřeba, aby byl tester schopný přijímat data z PC. Například, aby šlo měnit hodnoty parametrů jako frekvence jednotlivých obdélníkových signálů simulujících výstup z otáčkoměrů. Nebo také amplitudy složeného analogového signálu simulujícího výstup ze senzoru vibrací. Tento softwarový nástroj by měl také umožňovat přijímat data z modulu měření vibrací a zobrazit je do čitelné podoby, aby si uživatel mohl v reálném čase zkontrolovat, zda signál generovaný testerem je zpracován modulem měření vibrací správně.

6.1 Výběr HW pro tester na základě požadavků

Jako základní komponenta testeru byl vybrán kit Nucleo-L432KC. Procesor na tomto kitu je STM32L432KC založený na 32-bitové architektuře RISC ARM Cortex-M4. Tento procesor pracuje s frekvencí až 80MHz a obsahuje jednotku Floating Point Unit (FPU), která podporuje všechny instrukce a datové typy potřebné pro implementaci testeru. [19, 20]

Vlastnosti mikrokontroléru:

- STM32L432KC v UFQFPN32 pouzdře [20]
- ARM®32-bit Cortex®-M4 CPU [20]



Obr. 6.1 Kit Nucleo-L432KC [20]

- 80 MHz max CPU frekvence [20]
- Napájecí napětí od 1.65 V do 3.6 V [20]
- 256 KB paměti Flash [20]
- 64 KB paměti SRAM [20]
- 4 časovače [20]
- 2 rozhraní SPI/I2S [20]
- 2 rozhraní I2C [20]
- 2 rozhraní USART [20]
- 12-bit ADC s 10-ti kanály [20]
- 20 pinů GPIO s možností externího přerušení [20]
- RTC [20]
- kompatibilita s Arduino Nano [20]

Vlastnosti vývojového kitu:

- uživatelská LED (LD3) [20]
- tlačítko RESET [20]
- flexibilní napájecí zdroj desky [20]
  - USB nebo externí zdroj (3.3 V, 5 V, 7 - 12 V) [20]
- Tři podporovaná rozhraní pro USB [20]
  - Virtuální sériový port [20]
  - USB Mass storage pro drag'n'drop programování [20]
  - Debugovací port [20]

6.1.1 Komunikace mezi PC a testerem

Mezi PC a testerem probíhá sériová komunikace, toto připojení probíhá přes port USB, což nám umožňuje jednoduše komunikovat s hostitelským PC. Sériové spojení má dva kanály, jeden pro přijímaní a druhý pro odesílání. Spojení je asynchronní, a proto nastavení obou směrů musí být nakonfigurováno stejně. [21]

Tyto kanály mají řadu konfigurovatelných parametrů:

- přenosová rychlost
- délka dat
- paritní bit
- stop bit

Existuje celá řada standardních přenosových rychlostí od několika stovek bitů za sekundu až po megabity za sekundu. Výchozí nastavení pro sériové připojení mikrokontroléru mbed je 9600 baudů. [22]

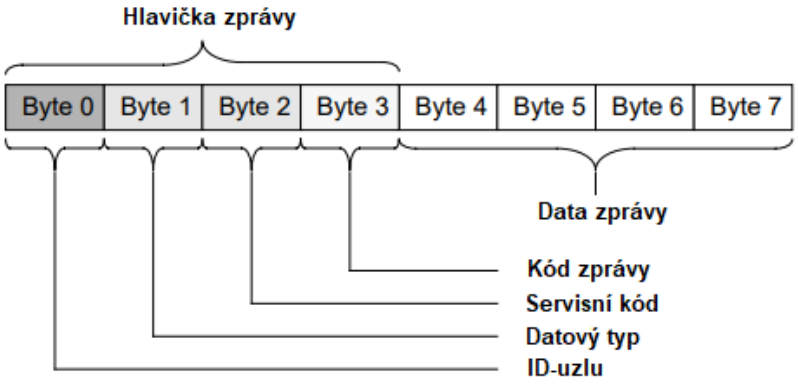
Přenesená data mohou být dlouhá buď 7 nebo 8 bitů, výchozí nastavení pro sériové připojení mikrokontroléru mbed je 8 bitů. [22]

Lze nastavit volitelný paritní bit, tento bit bude automaticky nastaven tak, aby počet jedniček v datech byl lichý či sudý. Nastavení parity je tedy liché, sudé nebo žádné. Výchozí nastavení sériového připojení mikrokontroléru mbed je nastavena parita na None, tedy žádná. [22]

Po přenesení datových a paritních bitů se na řadu dostává stop bit, kterým se do rámce vloží buď jeden nebo dva stop bity. Výchozí nastavení sériového připojení mikrokontroléru mbed je přidaní jednoho stop bitu. [22]

6.1.2 Komunikace mezi Vibromodulem a PC

Vibromodul bude umístěn v řídicí jednotce letadla a pro komunikaci s ostatními moduly nebo jednotkami v letadle používá protokol CANarospace. Tento protokol byl navržen pro vysoce spolehlivou komunikaci mikropočítačových systémů ve leteckých aplikacích přes sběrnici CAN bus. Definice protokolu je široce otevřená, aby umožňovala implementaci uživatelem definované zprávy. [23]



Obr. 6.2 Formát Canaerospace zprávy [23]

Jak je vidět na obrázku všechny zprávy CAN se skládají ze 4-bajtové hlavičky, která obsahuje bajty pro identifikaci uzlu, datový typ, kód zprávy a servisní kód a další jeden až čtyři bajty jsou pro skutečná data. [23]

Vibromodul tyto zprávy obsahující hodnoty svých jednotlivých vnitřních registrů posílá automaticky v pravidelných cyklech. V těchto registrech jsou uložené také hodnoty vibračních signálů, které Vibromodul zpracoval. Tato data se posílají jak po CAN sběrnici, tak zároveň po sériové komunikaci po RS-485, čehož při návrhu architektury využijeme a použijeme převodník USB-RS485 pro přijímaní dat z modulu měření vibrací. Vibromodul tedy využívá sériovou komunikaci typu RS-485.

Pro implementaci testeru budeme potřebovat pouze přijímat data z RS-485 servisního rozhraní Vibromodulu. Parametry, které budou potřeba nastavit pro připojení převodníku USB-RS485 k PC jsou následující:

- přenosovou rychlost 38400 baudů, tedy 38400 bitů za sekundu
- délka dat je 8 bitů
- paritní bit je nastavený na None, tedy žádný
- stop bit je nastavený na jeden bit.

II. PROJEKTOVÁ ČÁST

7 Generátor vibrací v programu Scilab

Moje práce začala implementací generátoru vibrací v programu Scilab, jehož cílem bylo simulovat dva nezávislé zdroje vibrací, tedy vibrace volné turbíny (PT) a vibrace kompresoru s turbínou (GG). Vygenerované signály jsem použil pro prvotní testování Vibromodulu. Kromě toho, poznatky z této simulace prohloubily znalosti dané problematiky, kterých jsem pak využil při návrhu testeru.

7.1 Vytváření sekvencí provozních režimů motoru

Generátor bude vytvářet sekvenci o celkové délce 30 sekund. Protože studujeme dva nezávislé zdroje vibrací, vytváří generátor dvě rozdílné sekvence. Na základě vzorkovací frekvence vytvoříme dané časové úseky pro jednu z turbín.

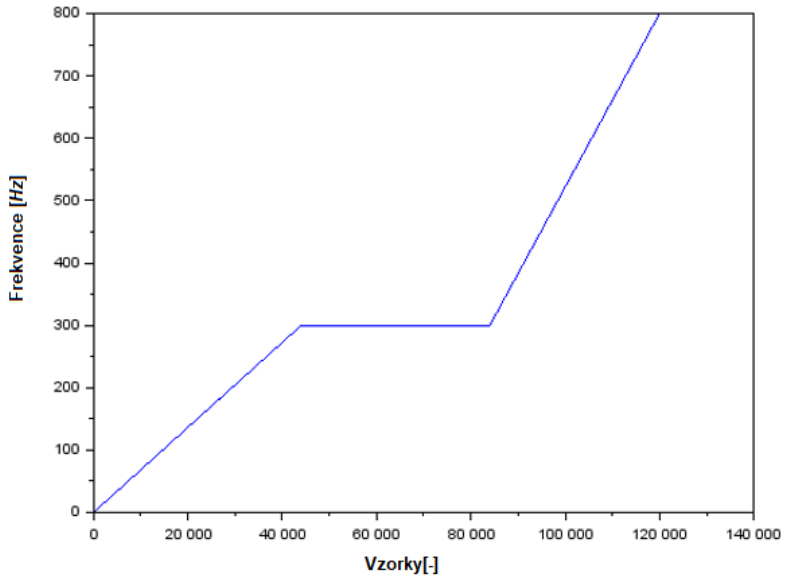
```
fs = 4000; //sample frequency
t_10=[0:1/4000:10]; // interval 10s
t_11=[0:1/4000:11]; // interval 11s
t_9=[0:1/4000:9];   // interval 9s
```

Ukázka vytvoření jedné sekvence hodnot frekvencí, simulující následující děj :

- fáze 10 sekund při vzletu – rostoucí otáčky turbíny z 0 na 300 Hz,
- fáze 11 sekund – otáčky turbíny ustálené na 300 Hz,
- fáze 9 sekund – otáčky turbíny rostoucí z 300 Hz na 800 Hz.

```
fn_1 = [0:300/size(t_11,2):300]; //creating values of increasing
    frequency
fn_2 = zeros(1,44000); //creating values of steady frequency
for i = 1:40000
    fn_2(i) = 300;
end
fn_3 = [300:500/size(t_9,2):800]; //creating values of increasing
    frequency
freqt = [fn_1(1:44000) fn_2(1:40000) fn_3(1:36001)]; // creating a
    frequency response
ifreqt = cumsum(freqt)/fs; // frequency integration
```

Vytvořenou sekvenci hodnot pak zobrazíme v grafu, kde můžeme vizuálně zkontrolovat, že je průběh sekvencí daných frekvencí správně. Vytvořená sekvence hodnot frekvencí simuluje vibrační signál jedné z turbín při vzletu letadla .



Obr. 7.1 Sekvence hodnot volné turbíny

7.2 Vytváření složeného signálu

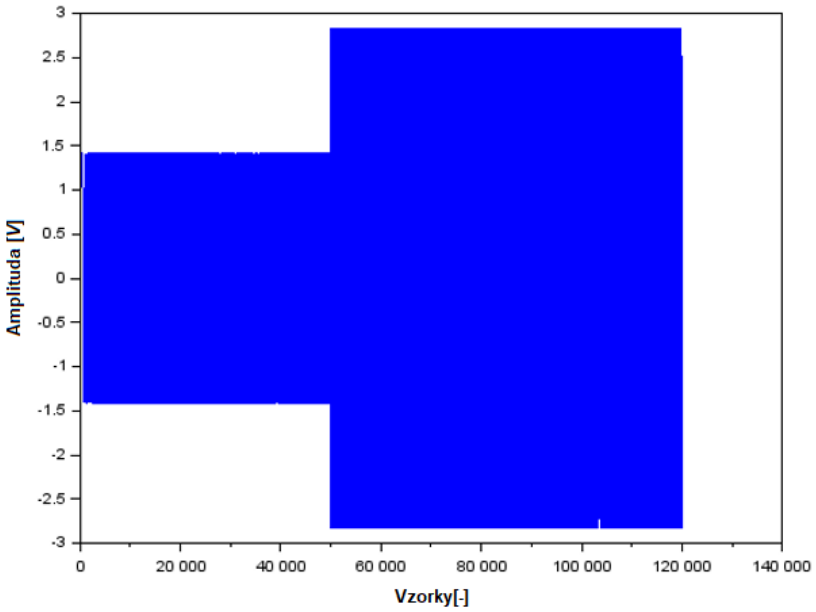
Dále byly vytvořeny dva harmonické signály s fází 45°, jejímž sečtením získáme signál složený. Také byly vytvořeny referenční signály sin *sv* a cos *cv* s konstatní amplitudou a stejnou sekvencí změny frekvence, které se použijí pro simulaci harmonické analýzy, tedy výpočet Fourierovy transformace.

```
data = sin(%pi*2*ifreqt) + cos(%pi*2*ifreqt); // composite signal from
    two signals
sv = sin(%pi*2*ifreqt);
cv = cos(%pi*2*ifreqt);
```

Do simulace byla přidána skoková změna amplitudy, tedy zvětšení vibrací v čase 50000/4000 sekund a to vytvořením obálky harmonického signálu, nakonec byla obálka signálu vynásobena s výsledným signálem.

```
amp = zeros(1,120001); //creating an amplitude envelope for the signal
for i = 1:120001
    if(i >= 50000) then
        amp(i) = 2;
    else
        amp(i) = 1;
    end
end
vys_sign = data.*amp; // resulting signal
```

Na grafu lze vidět skokovou změnu amplitudy signálu v čase 50000/4000 sekund.



Obr. 7.2 Amplitudová obálka výsledného signálu

7.3 Ověření simulace

Byla vypočítána amplituda obou signálů pro ověření správnosti. Bylo zde využito referenčních signálů *sv* a *cv*, které jsem použil pro výpočet imaginární a reálné složky u obou harmonických signálů.

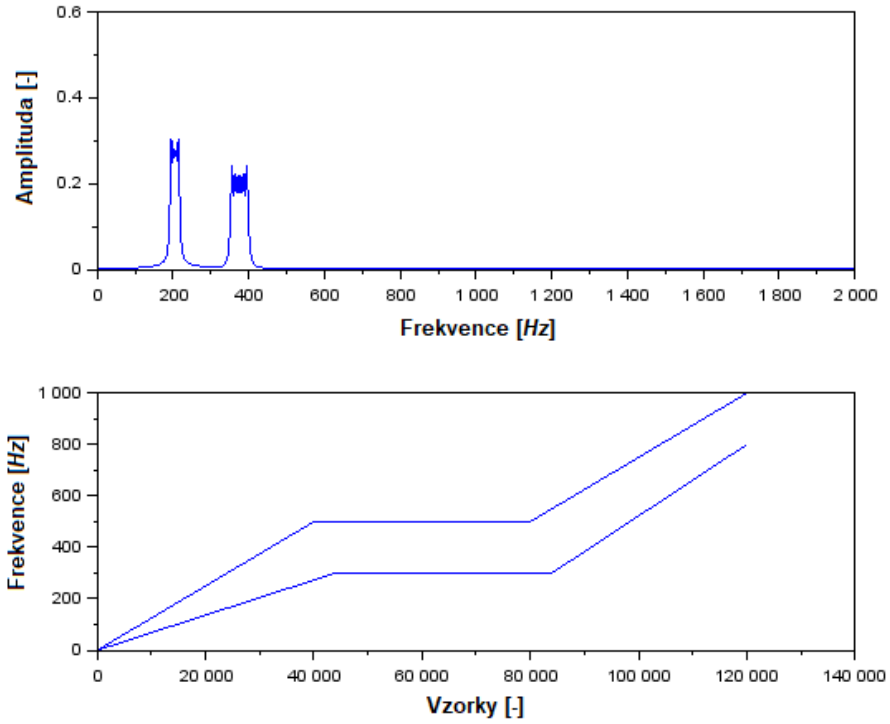
```
//amplitude calculation
for j = 1:30
        re = vys_sign((4000*j)-3999:4000*j)*sv((4000*j)-3999:4000*j);
        im = vys_sign((4000*j)-3999:4000*j)*cv((4000*j)-3999:4000*j);
        am = re*re + im*im;

        re2 = vys_sigv((4000*j)-3999:4000*j)*sv2((4000*j)-3999:4000*j);
        im2 = vys_sigv((4000*j)-3999:4000*j)*cv2((4000*j)-3999:4000*j);
        am2 = re2*re2 + im2*im2;
        printf('\n%f %f',am,am2);
end
```

Stejně byl vytvořen i signál pro druhou turbínu, pouze s jinými frekvenčními hodnotami. Pro ověření zda jsou data správně vygenerována, jsem použil harmonickou analýzu za pomoci algoritmu rychlé Fourierovy transformace.

```
count_of_m = 4000; //count of measures

f = fs * ( 0 : (count_of_m / 2 )) / count_of_m;
for i = 1:3
//display fft after one second along with frequency characteristics
    for j = 1:30
        fft_data = fft((vys_sigv((4000*j)-3999:4000*j)+vys_sign((4000*j)-
    3999:4000*j))/(count_of_m / 2 ));
        subplot(2,1,1);
        xlabel("Frequency[rpm]");
        ylabel("Amplitude[-]");
        a = gca();delete(a.children);
        plot(f,abs(fft_data(1:size(f, '*'))));
        subplot(2,1,2);
        ylabel("Frequency[rpm]");
        xlabel("Samples[-]");
```

```
16            plot(freqt);
17            plot(freqt_2);
18            sleep(500);
19        end
20    end
```

Nakonec jsem vytvořil animaci harmonické analýzy, kde jsou zobrazeny sekvence frekvencí simulující volnou turbínou (PT) a kompresor s turbínou(GG). Animace postupuje po jedné sekundě, kde se délka kroku rovná vzorkovací frekvenci, tedy 4 kHz.



Obr. 7.3 Ukázka výsledné animace

8 Architektura testeru pro Vibromodul

Na základě požadavků pro návrh testeru byla navržena architektura, která bude splňovat tyto požadavky, umožňující funkční testovaní Vibromodulu.

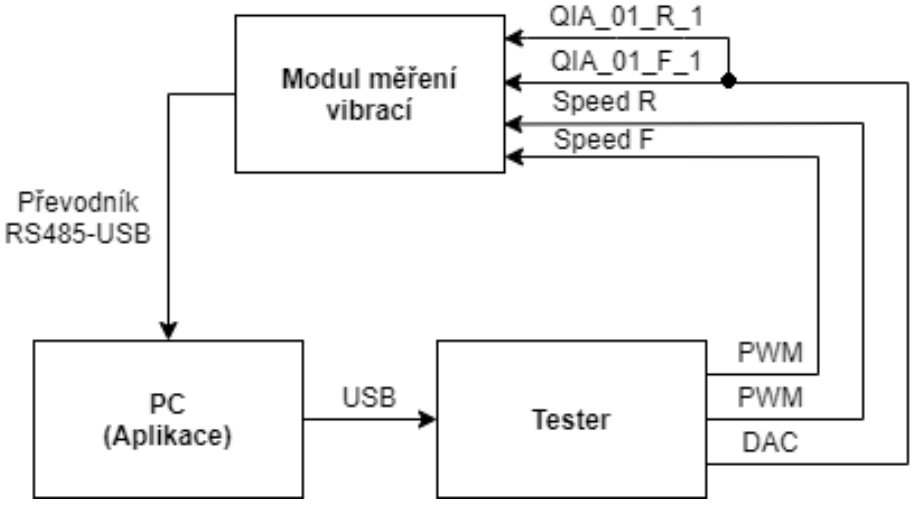
Architektura je navržená z následujících bloků:

- Vibromodul
- Tester
- PC s aplikací pro tester

První blok - „Tester“ - generuje signály simulující výstupy otáčkoměrů volné turbíny a také kompresoru s turbínou na stejné hřídeli. Tento obdélníkový signál je generován pomocí pulzně šířkové modulace. Dále tento blok generuje signál simulující výstup ze snímače měření vibrací pomocí číslicově-analogového převodníku (DAC). Tester bude ovládán pomocí aplikace na PC skrze USB rozhraní.

Druhý blok - „Modul měření vibrací“ - zpracovává příchozí generovaný signál z testeru a periodicky posílá rámce (CanAero), obsahující hodnoty zpracovaných dat, které jsou posílány po převodníku RS485-USB do PC.

Třetí blok - „PC“ - obsahuje aplikaci, která ovládá tester přes USB virtuální sériový port a nastavuje dané parametry generovaných signálů. Zároveň tato aplikace přijímá data z modulu měření vibrací a zobrazuje je do čitelné podoby. Uživatel tedy může zkontrolovat, zda signál, který generuje tester, je zpracován správně.



Obr. 8.1 Blokové schéma architektury testeru

8.1 Návrh využití periferií a pinů MCU

V této kapitole je popsáno využití a propojení pinů a periferií MCU s dalšími komponentami testeru. Jedná se o signály, které se generují pro Vibromodul. Dále pak textový LCD, který uživatele informuje o nastavených parametrech generovaných signálů a EEPROM paměť pro nahrání a generování vzorků reálných vibračních signálů.

Tab. 8.1 MCU piny pro generování signálů

MCU Pin	Periferie
PB_0	PWM
PB_6	PWM
PA_4	DAC

Tab. 8.2 Propojení pinů MCU a LCD

MCU Pin	LCD Pin
PB_4	RS
PB_5	E
PA_8	D4
PB_1	D5
PB_7	D6
PA_11	D7
PA_0	Pot.

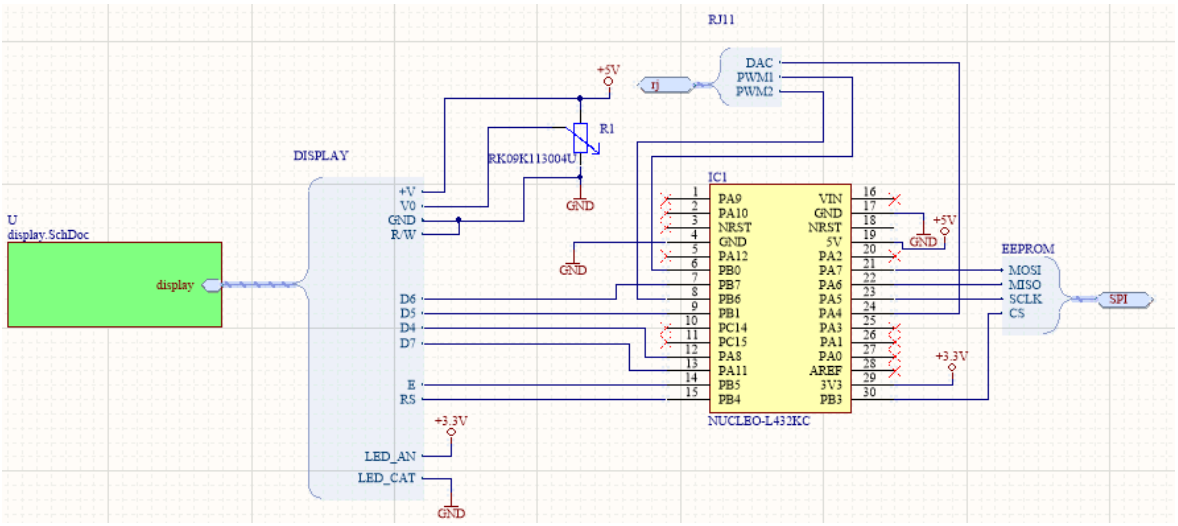
Tab. 8.3 Propojení pinů MCU a EEPROM

MCU Pin	EEPROM Pin
3V3	VCC
PA_7/MOSI	SI
PA_6/MISO	SO
PA_5/SCLK	CLK
PB_3/CS	CS
GND	GND

8.2 Návrh a realizace plošného spoje testeru

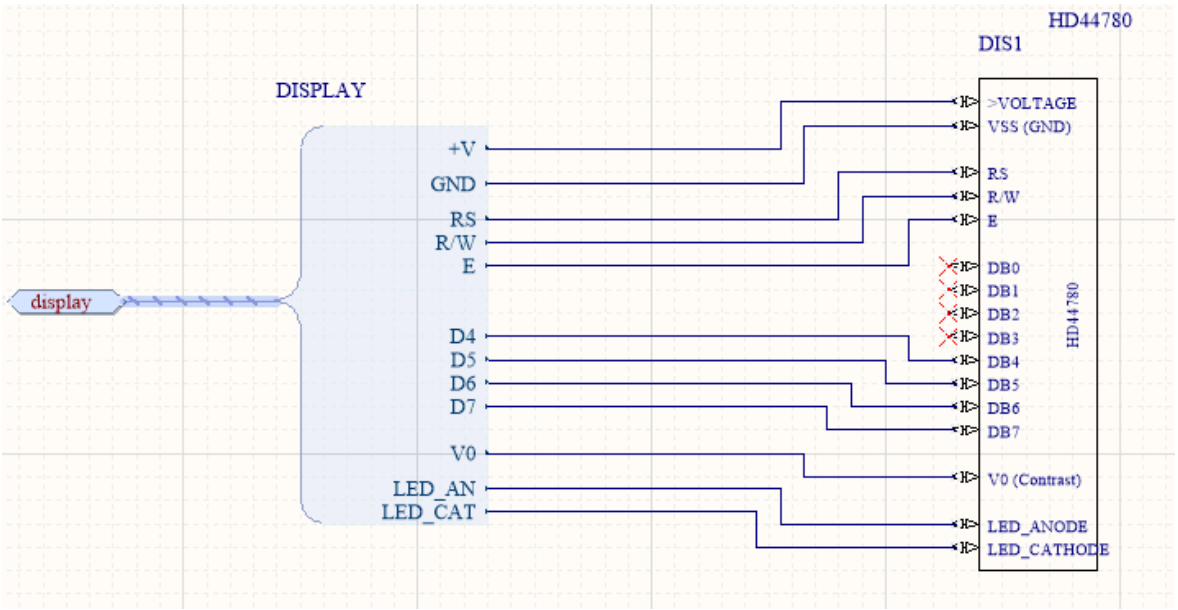
Pro návrh desky plošných spojů byl použit editor plošných spojů Altium Designer 18.0. Před návrhem desky plošných spojů bylo potřeba navrhnout schéma testeru se všemi potřebnými komponentami. Schéma testeru jsem rozdělil do tří schématických souborů.

První schéma obsahuje MCU Nucleo-L432KC, ze kterého jsou vyvedeny dané signály do ostatních bloků.



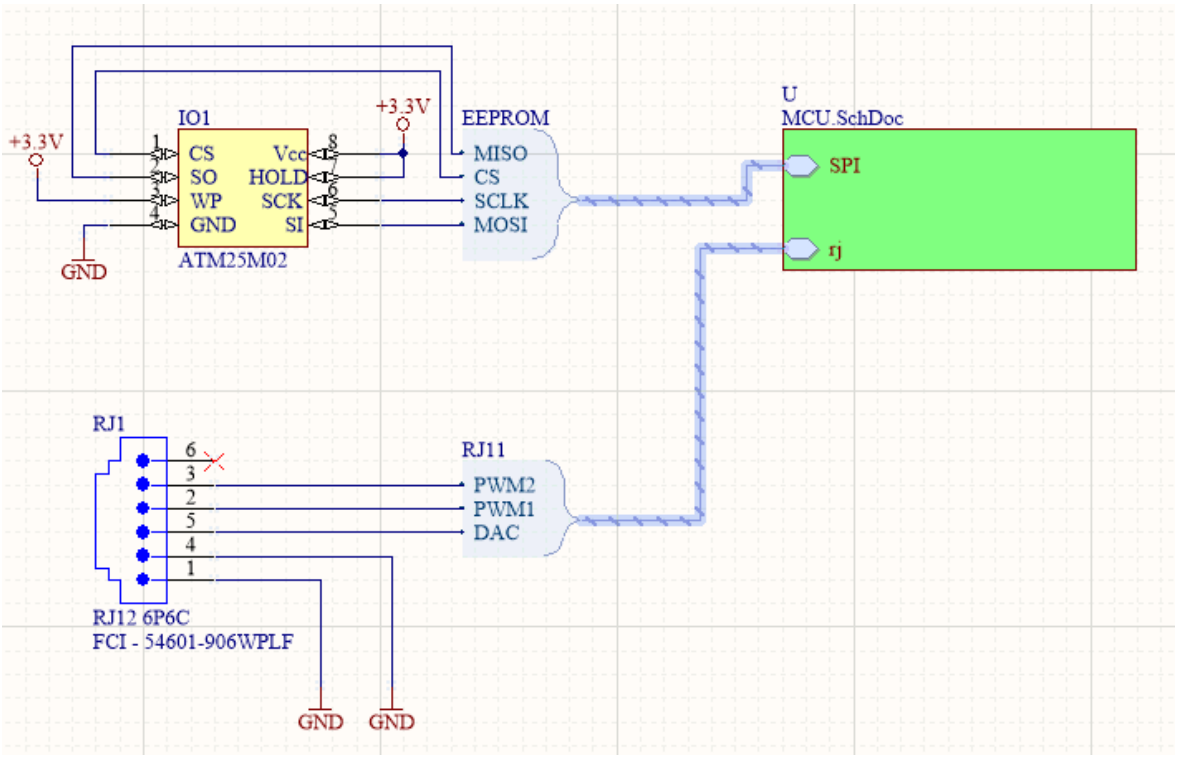
Obr. 8.2 Schéma zapojení MCU

V druhém bloku jsou vyvedeny signály pro zapojení textového LCD displeje, který zobrazuje uživateli velikost nastavených hodnot parametrů frekvencí PWM signálů a napětí V_{pp} výsledného signálu z DAC.



Obr. 8.3 Schéma zapojení LCD

V posledním bloku jsou vyvedeny potřebné signály ke komunikaci s EEPROM pamětí přes SPI rozhraní. Dále jsou zde na konektor RJ12 vyvedeny signály simulující výstupy otáčkoměrů dvou turbín a výstup senzoru vibrací umístěného na motoru.



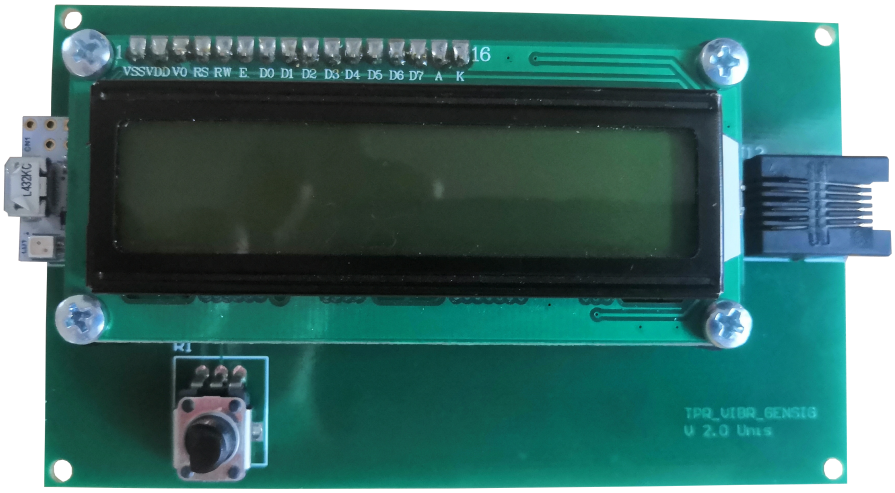
Obr. 8.4 Schéma zapojení EEPROM paměti a konektoru RJ12

Rozmístění komponent na desce plošných spojů je takové, že Nucleo kit je umístěn jednou stranou na okraji desky, aby bylo možné připojit k jeho microUSB portu kabel pro nahrávaní zkompilovaného binárního kódu a také pro komunikaci s PC.

Nad kitem je umístěn textový LCD displej HD44780, jehož jas lze ovládat pomocí připojeného potenciometru na desce. Tento displej drží pomocí čtyř distančních sloupců přišroubovaných skrz LCD panel k desce. Piny z LCD jsou umístěny v zasouvací liště, na které už vedou cesty desky plošných spojů.

Dále deska obsahuje paměť EEPROM, která komunikuje přes SPI rozhraní. Do této paměti se nahrají reálná navzorkovaná data ze senzoru vibrací naměřených při letu a budou se generovat na výstupu DAC. Tento signál obsahuje celé spektrum kmitočtů, které vytváří proudový motor, na rozdíl od „syntetického“ signálu generovaného jako součet dvou harmonických signálů.

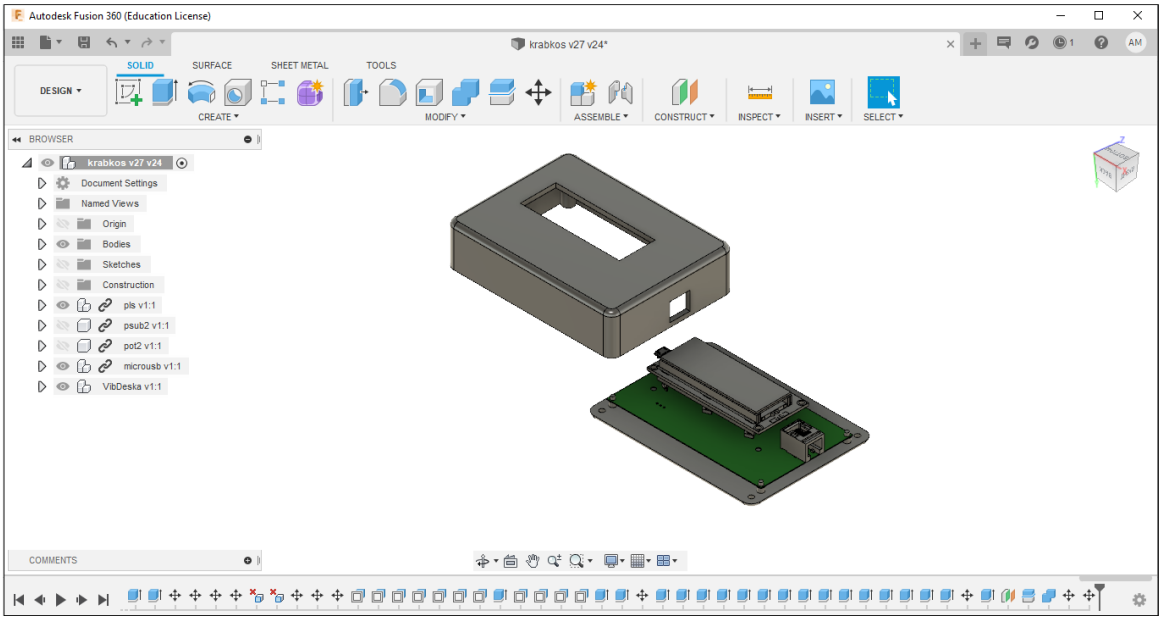
Signály z periferií kitu jako obdélníkový či analogový signál jsou vyvedeny na konektor RJ12 společně se zemí obvodu. Z konektoru RJ12 vedou signály kabelem do dvou konektorů, které se zapojí na vstup modulu měření vibrací.



Obr. 8.5 Ukázka desky plošných spojů

8.3 Návrh krabičky pro desku plošných spojů

Z programu Altium Designer 18.0 jsem si nechal vyexportovat 3D model desky ve formátu STL, abych mohl vytvořit pro desku plošných spojů krabičku, ve které bude umístěna. Tuto krabičku jsem navrhl a vymodeloval v programu AutoDesk Fusion 360 a nechal vytisknout na 3D tiskárně.



Obr. 8.6 Ukázka modelování krabičky

Tato krabička se dělí na dvě části, do první části můžeme položit desku skrze čtyři plastové kolíky, které znemožňují posouvání desky v krabičce a druhá část obsahuje otvor pro LCD displej. Otvory na konektory jsou přesně uprostřed dvou částí krabičky.

Po nasazení vrchní části krabičky na spodní část, kde je přichycená deska plošných spojů, je potřeba tyto dvě části sešroubovat. Pomocí závitníku jsem upravil vnitřní závity čtyř válcových otvorů pro lepší zašroubování obou částí krabičky do jednoho celku.

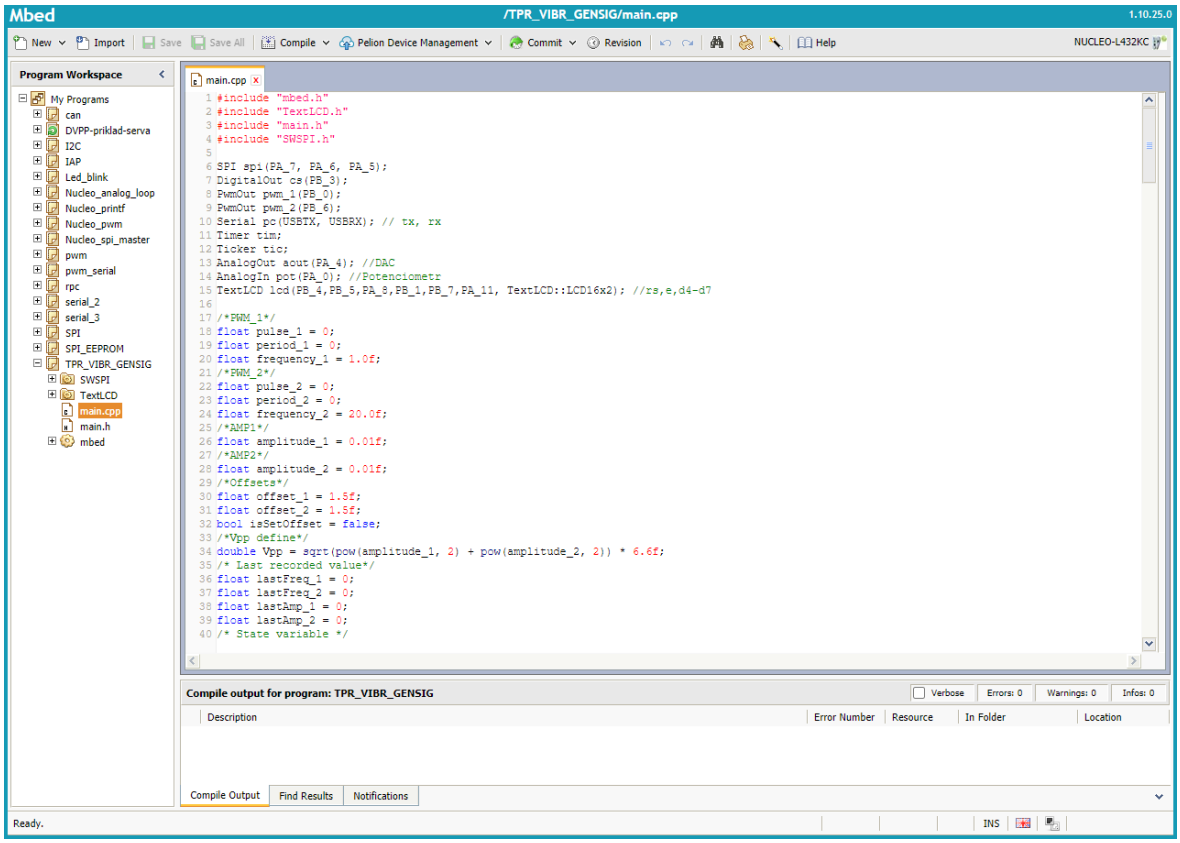
Obr. 8.7 Ukázka testeru v krabičce

9 Implementace firmware testeru

Pro vývoj firmwaru testeru byla využita bezplatná platforma Mbed. Tuto platformu vyvíjí společnost ARM ve spolupráci s několika významnými technologickými společnostmi. Tato open source platforma umožňuje zrychlený vývoj jakéhokoli zařízení, založeného na mikrokontroléru s architekturou ARM s jádrem Cortex-M.

Skládá se z hlavních knihoven, které využívají hardwarovou abstrakční vrstvu neboli HAL a poskytují periferní ovladače pro mikrokontroléry. Firmware lze psát pomocí kombinace jazyků C a C++ v online IDE Mbed. Jediné, co je potřeba, je webový prohlížeč, kde můžeme editovat a kompilovat kód, který se kompiluje na vzdáleném serveru za pomocí kompilátoru ARMCC.

Také lze kód importovat, exportovat nebo sdílet s ostatnímu uživateli. Kromě online IDE lze využít i jiná vývojová prostředí jako je Keil μVision, Eclipse atd. Nedávno bylo vydáno desktopové IDE s názvem Mbed Studio pro Mbed OS 5 zahrnující všechny nástroje v jednom balíčku pro vytváření, ladění a kompilaci kódu.



Obr. 9.1 Ukázka online vývojového prostředí pro platformu Mbed

9.1 Hlavní program

Na základě navržené architektury pro tester byl firmware implementován následujícím způsobem.

Výstupy dvou snímačů otáček, a to pro volnou turbínu a pro kompresor ve spojení s turbínou, budou simulovány pomocí pulzně šířkové modulace o dané frekvenci.

Výstup z D/A převodníku bude emulovat výstupní signál senzoru vibrací, který snímá vibrace dvou nezávislých hřídelí GG a PT. Emulace dosáhneme tak, že výstup z D/A převodníku bude obsahovat součet dvou harmonických signálů, podle následujících vzorců.

$$x_1(t) = A_1 \sin(2\pi f_1 t + \varphi_o) \tag{9.1}$$

$$x_2(t) = A_2 \cos(2\pi f_2 t + \varphi_o) \tag{9.2}$$

Frekvence obou harmonických signálů musí odpovídat frekvencím pulzně šířkové modulace, emulující výstupy dvou otáčkoměrů. Díky tomu algoritmus DFT uvnitř Vibromodulu dokáže správně zpracovat a vyhodnotit úroveň emulovaných vibrací z obou zdrojů (PP i GT).

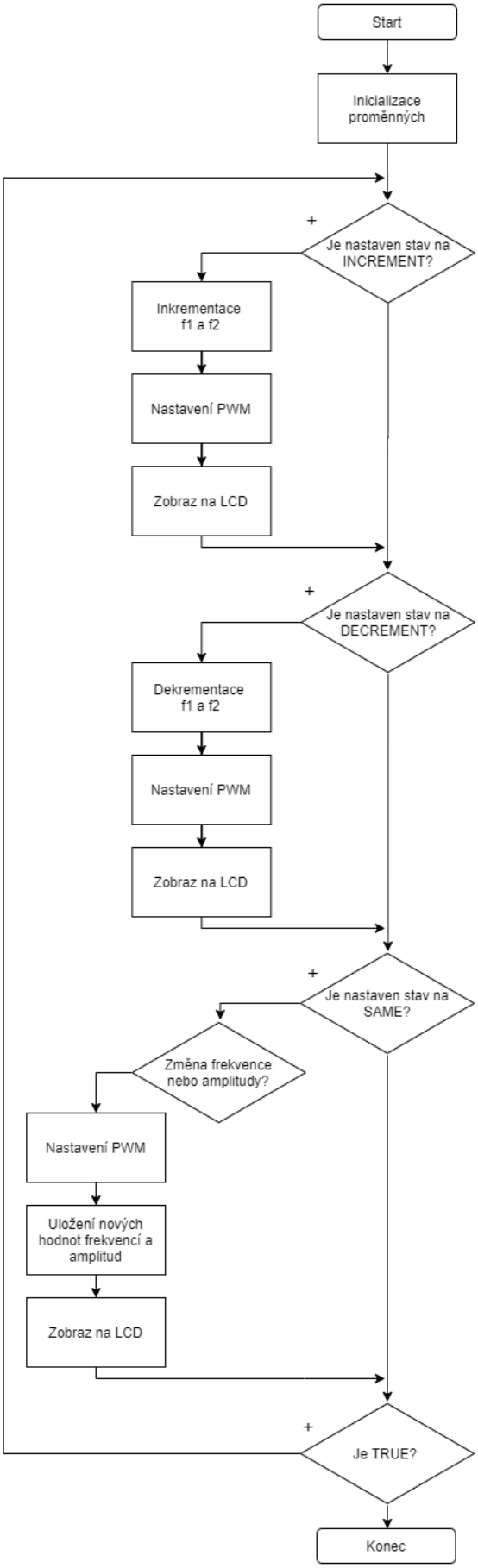
Jednotlivé hodnoty frekvencí f_1 a f_2 lze nastavit jako desetinná čísla s jednou desetinnou čárkou od hodnoty 1 do 1000. Odpovídající parametry amplitud A_1, A_2 lze v programu také samostatně nastavit. Velikost výsledného napětí V_{pp} je určena podle vzorce.

$$V_{pp} = \sqrt{A_1^2 + A_2^2} * 6.6 \tag{9.3}$$

Pohybuje se tedy v rozmezí 93mV až do 3,26V a aktuální hodnota se zobrazuje na textovém LCD. Fázový posun je pro obě harmonické funkce stejný, mění se automaticky při běhu programu, a to podle toho, zda je napětí V_{pp} větší či menší než 2,25V.

Firmware obsahuje tři frekvenční módy:

- Rostoucí frekvenci – simulující rostoucí frekvenci otáček turbín při vzletu z hodnoty 1 Hz do hodnoty 1000 Hz.
- Stabilní frekvenci – simulující stabilní frekvenci otáček turbín při letu.
- Klesající frekvenci – simulující klesající frekvenci otáček turbín z hodnoty 1000 Hz do hodnoty 1 Hz.



Obr. 9.2 Zjednodušený diagram hlavního programu

9.2 Vytváření instancí

Firmware testeru začíná vytvářením instancí, tedy objektů podle třídy, které už obsahují implementované metody pro nastavení parametrů pro generovaní PWM, ovládaní analogového výstupu D/A převodníku, či analogové vstupu pro potenciometr. Je zde také inicializace pinů pro SPI rozhraní, a to pro komunikaci mezi vývojovým kitem a EEPROM pamětí.

Dále také obsahuje inicializaci pro textový LCD a sériovou komunikaci mezi vývojovým kitem a počítačem, dále také pro vytvoření instancí pro časovače a čítače. Při vytváření některých instancí je potřeba jako parametr zadat název pinu, na kterém chceme, aby obsahoval vstup/výstup dané periferie.

9.3 Inicializace proměnných

Následuje inicializace proměnných, potřebných pro generovaní obdélníkového signálu za pomoci pulzně šířkové modulace. Tento signál bude emulovat výstupy dvou otáčkoměrů, je zde také potřeba inicializovat proměnné jako frekvence, perioda či střída, a to všechny jako racionální čísla. Pro signál emulující výstup ze senzoru vibrací je zde inicializace proměnných amplitud harmonických signálů.

Protože není volná turbína a kompresor s turbínou mechanicky spojena a jsou tedy nezávislé, nebudou mít při zapnutí firmwaru stejnou frekvenci otáček. Proto jsem zvolil, aby počáteční rozdíl byl 20 Hz. Byl také inicializován fázový posun obou signálů, výstup z D/A převodníku se totiž generuje jen v kladných napětích a při určitém zvýšeném napětí se začne signál ořezávat, proto je fázový posun měnitelný, aby bylo možno generovat signál co v největším možném napěťovém rozsahu. Nakonec je definováno napětí signálu V_{pp}, které se pro složené harmonické signály vypočítá podle daného vzorce.

```
1  /*PWM_1*/
2  float pulse_1 = 0.0f;
3  float period_1 = 0.0f;
4  float frequency_1 = 1.0f;
5  /*PWM_2*/
6  float pulse_2 = 0.0f;
7  float period_2 = 0.0f;
8  float frequency_2 = 20.0f;
9  /*Amplitudes*/
10 float amplitude_1 = 0.01f;
11 float amplitude_2 = 0.01f;
12 /*Offsets*/
13 float offset = 1.5f;
14 bool isSetOffset = false;
15 /*Vpp define*/
16 double Vpp = sqrt(pow(amplitude_1, 2) + pow(amplitude_2, 2)) * 6.6f;
```

9.4 Funkce pro přepočet PWM

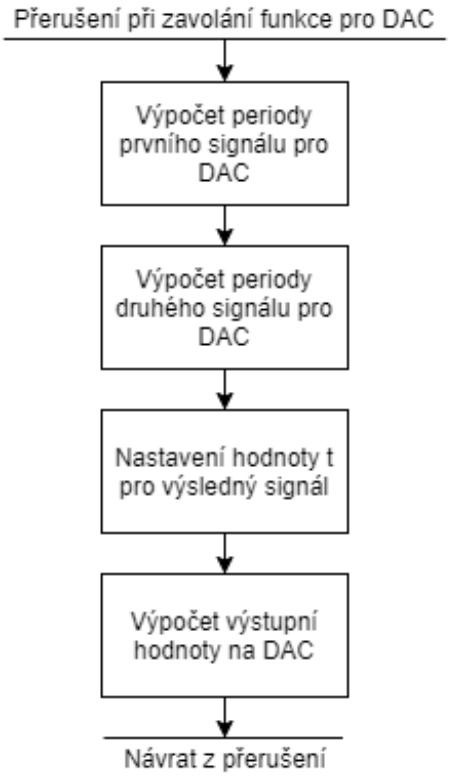
Firmware obsahuje funkci k přepočítávání pulzně šířkové modulace. Tato funkce se volá, pokud je firmware nastaven v rostoucím, či klesajícím módu otáček a dochází k průběžné změně frekvencí, nebo pokud je některá z frekvencí změněna v módu stabilním. Na základě frekvence se vypočítá perioda a podle periody se vypočítá střída pulzně šířkové modulace, střída signálu má být vždy 50%. Na konci funkce se vždy zavolají metody obou instancí pro PWM, kde se předají jako parametry nově vypočítané hodnoty, díky čemuž bude docházet ke generovaní obdélníkového signálu podle zadané frekvence.

```
void setPWM()
{
    /*calculate the period and witdh of the pulse*/
    period_1 = 1.0f / (frequency_1);
    pulse_1 = (period_1 / 2.0f);
    period_2 = 1.0f / (frequency_2);
    pulse_2 = (period_2 / 2.0f);
    /*setting PWM outputs to required values*/
    pwm_1.period(period_1);
    pwm_1.pulsewidth(pulse_1);
    pwm_2.period(period_2);
    pwm_2.pulsewidth(pulse_2);
}
```

9.5 Funkce pro generování výstupu z D/A převodníku

Pro generování analogového signálu využijeme instance časovače a tzv. tickeru. Rozhraní ticker je použito k nastavení opakovaného přerušení pro opakované volání funkce s danou periodou, což je využito pro funkci generující výstup na D/A převodníku. Pokaždé, když je funkce zavolána, dojde k přepočtu periody na základě nastavených frekvencí, periody jsou vypočítány v mikrosekundách. Rozhraní časovače je použito pro vytvoření hodnoty v čase t. Ta je vypočítána jako zbytek po dělení naměřeného času v mikrosekundách pomocí časovače a součtu dvou period. Potřebné proměnné jsou dosazeny a výsledná výstupní hodnota D/A převodníku je dána součtem dvou harmonických funkcí podle vzorců 7.1 a 7.2.

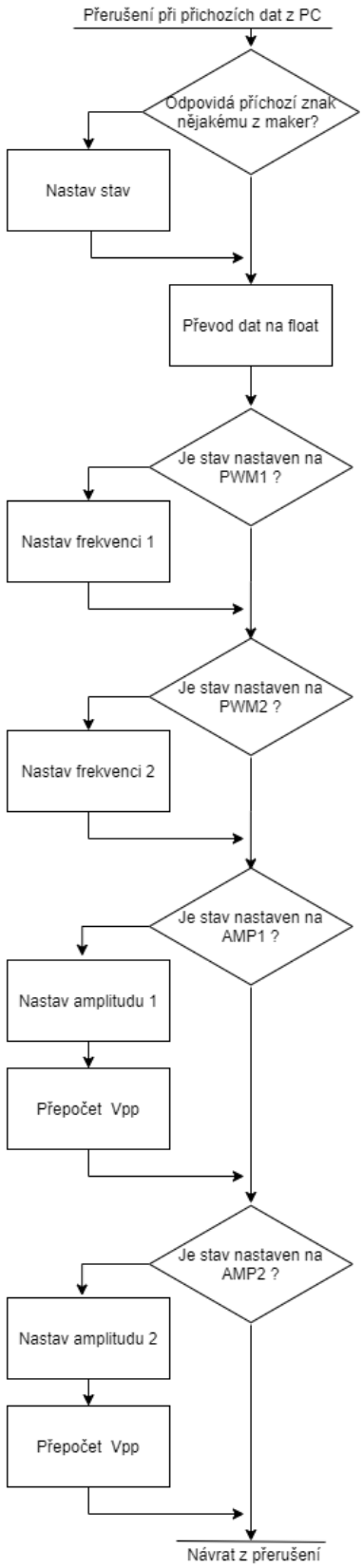
```
tim.start();
tic.attach_us(&DAC_Int, 1000000 / DAC_Frequency);

void DAC_Int()
{
    int dac_period_1 = 1000000 / (frequency_1);
    int dac_period_2 = 1000000 / (frequency_2);
    int pos = tim.read_us() % (dac_period_1 + dac_period_2);
    aout = ((amplitude_1*(sin(pos*3.14f*2/dac_period_1)+offset)) +
            (amplitude_2*(cos(pos*3.14f*2/dac_period_2)+offset)));
}
```



Obr. 9.3 Diagram přerušení při příchozím znaku

9.6 Funkce při přijímaní nových dat

Dále byla vytvořena funkce, která je zavolána za pomocí přerušení pokaždé, když přijdou po sériové lince jakákoliv data z PC. Tato funkce porovnává přijatý znak se všemi definovanými makry, a pokud je nalezena shoda, je nastaven daný stav. Pokud příchozí data nenesou informace o změně stavu frekvenčního módu nebo o změně stavu proměnné signálu, jejíž hodnotu chceme změnit, tedy jednu z frekvencí nebo hodnotu jedné z amplitud dvou harmonických signálů, tato data pravděpodobně obsahují hodnotu, jakou má být jedna vybraná proměnná harmonického signálu nastavena. Toto pole znaků je převedeno na datový typ float, a pokud splňuje dané podmínky, je nastavena buď velikost dané amplitudy jednotlivého harmonického signálu, nebo frekvence jednotlivých pulzně šířkových modulací.



Obr. 9.4 Diagram přerušení při příchozím znaku

9.7 Inicializace paměťi pro nahrání reálných vzorků

Firmware také umožňuje načítání uložených reálných vzorků z EEPROM paměti a jejich generování na výstupu D/A převodníku.

Pro uložení vzorků byla použita paměť AT25M02 od výrobce ATMEL, komunikující přes rozhraní SPI, podporující SPI mód 0(0,0) a 3(1,1).

Výrobce uvádí možnost 1 000 000 cyklů zápisu a uchování dat po dobu 40 let. Kromě toho zařízení pracuje od napětí 1,7 V do 5,5 V a jeho maximální frekvence hodinového signálu může dosahovat až 5MHz. Velikost této paměti je 2 Mbity a je organizována jako 262144 slov, kde jedno slovo má 8 bitů. Jeden reálný vzorek je typu float a má tedy velikost 4 bajty.

Než se začne do paměti zapisovat či z ní číst hodnoty, je potřeba nejprve poslat vhodný příkaz a hned za ním adresu o velikosti 24 bitů, z jakého místa v paměti se bude číst či zapisovat. Umožňuje tedy zápis po jednom bajtu nebo po jedné stránce o velikosti 256 bajtů.

V jednom zapisovacím cyklu lze zapsat maximálně 256 bajtů, po překročení se hodnoty začnou přepisovat od začátku, což je díky internímu čítači, který po každém zapsaném bajtu inkrementuje 8 nejméně významných bitů adresy.

U čtení z paměti po poslání příkazů nastavujících EEPROM paměť ke čtení stačí poslat jen adresu a znovu pomocí interního čítače se čtou hodnoty až k poslední možné adrese, nejsme tedy omezeni 256 bajty při jednom čtecím cyklu, tak jak je to u cyklu zapisovacího.

Při implementaci jsem použil mbed knihovnu SWSPI pro vytvoření dané instance. Definoval jsem jednotlivé operační kódy a nahrál pole reálných vzorků z druhého souboru a určil jeho velikost.

```
1
2  SWSPI spi(PA_7, PA_6, PA_5);
3  DigitalOut cs(PB_3);
4  #define WREN          (0x06)
5  #define RDSR          (0x05)
6  #define WRDI          (0x04)
7  #define READ_LOW      (0x03)
8  #define WRITE_LOW     (0x07)
9
10  extern float  EEPROM_data_write[];
11
12  int arr_size = sizeof(EEPROM_data_write)/sizeof(EEPROM_data_write[0]);
```

9.8 Funkce pro ovládaní paměti

Pro správnou komunikaci mezi MCU a EEPROM pamětí byla implementována funkce jako inicializace SPI sběrnice, kde se před začátkem komunikace, určí velikost posílaných dat a vztah mezi daty a hodinovým signálem nebo také frekvenci hodinového signálu.

```
void spi_init(){
    spi.format(8,0);
    spi.frequency(1000000);
}
```

Další funkce jsou pro povolení či zablokovaní zápisu na EEPROM paměť. Obě funkce nastaví chip select na logickou úroveň 0 pro posílání dat a pošlou vhodný operační kód. Jakmile jsou data poslána, nastaví se chip select znovu na logickou jedničku.

```
void write_enable(){
    cs = 0;
    spi.write(WREN);
    cs = 1;
}

void write_disable(){
    cs = 0;
    spi.write(WRDI);
    cs = 1;
}
```

Nebo také funkce zajišťující čtení registru statusů pro ověření, že EEPROM paměť komunikuje s MCU.

```
int read_status_reg(){
    cs = 0;
    spi.write(RDSR);
    int value = spi.write(0xFF);
    printf("%d\r\n",value);
    cs = 1;
    return value;

}
```

Hodnoty, které se do paměti zapisují, jsou typu float s velikostí 4 bajty. Potřeboval jsem tedy tento datový typ převést na čtyři bajty. K tomu jsem použil datový typ union, který jsem definoval dvěma datovými typy - jako pole kladných znaků o velikosti 4 bajty a jako jeden prvek typu float.

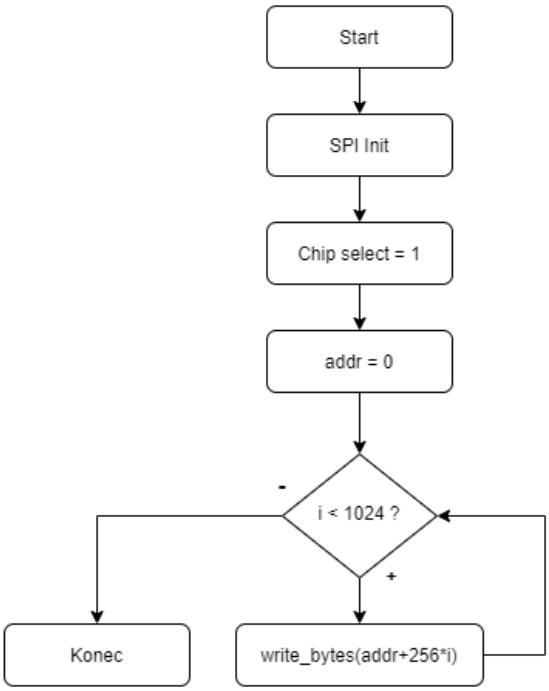
```
union one_sample {
    unsigned char buff[4];
    float number;
} one_sample;
```

Dále jsem vytvořil funkci, kde se jako parametr předá desetinné číslo, které se převede na 4 kladné znaky, které je už možné zapsat do EEPROM paměti.

```
void fillOneFloat(float number){
 one_sample.number = number;
}
```

Tato funkce slouží pro zápis jedné stránky o velikosti 256 bajtů. Do funkce se předává celá kladná 32 bitová adresa. Nejprve se pošlou vhodné operační kódy, které nastaví paměť k zápisu, následuje poslání adresy, které je vhodně vynulována a bitově posunuta. Pak je použita funkce, která z datového typu float udělá čtyři bajty a tyto bajty jsou za sebou poslány. Toto se opakuje, dokud není posláno všech 256 bajtů dat.

```
void write_bytes(uint32_t addr){
    write_enable();
    wait_ms(100);
    cs = 0;
    spi.write(WRITE_LOW);
    spi.write((addr & 0xFF0000) >> 16);
    spi.write((addr & 0x00FF00) >> 8);
    spi.write((addr & 0x0000FF));
        for(int j = 0; j < 64; j++){
            fillOneFloat(EEPROM_data_write[g]);
             g++;
             for(int i = 0; i < 4 ; i++){
                 spi.write(test.buff[i]);
             }
        }
        wait_ms(50);
        cs = 1;
}
```

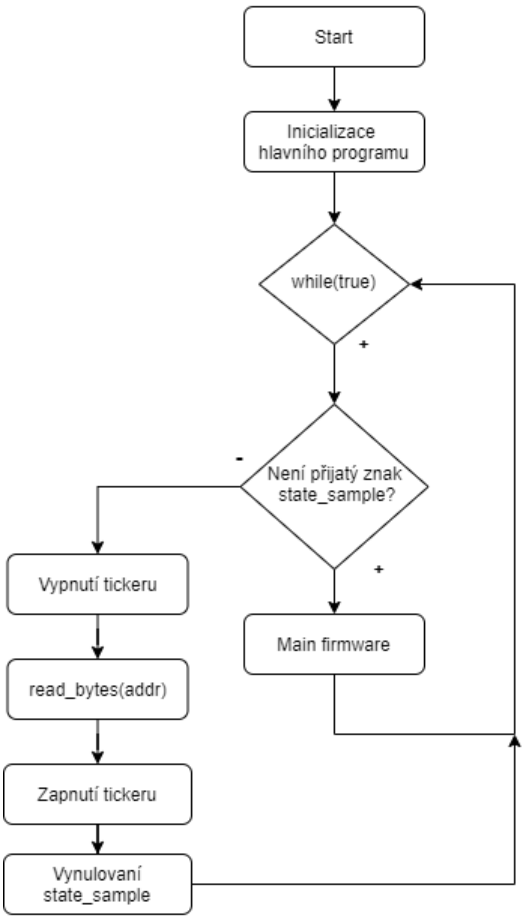


Obr. 9.5 Zjednodušený diagram zápisu do EEPROM

Další funkce slouží pro čtení nahraných vzorků z paměti. Do funkce se předává celá kladná 32 bitová adresa. Nejprve se pošlou vhodné operační kódy, které nastaví paměť ke čtení, následuje poslání adresy, která je vhodně vynulována a bitově posunuta. Tato adresa bude vždy nastavená na hodnotu 0x0000h a pomocí interního čítače je adresa inkrementována až na konečnou hodnotu 0x3FFFFh v hexadecimální soustavě, čímž tedy využijeme všechno volné místo v paměti. Dále následuje cyklus, který v každé iteraci získá čtyři bajty, které jsou převedeny na hodnotu datového typu float, tenhle cyklus se opakuje přesně 65536 krát.

```
void read_bytes(uint32_t addr){
    write_disable();
    wait_ms(100);
    cs = 0;
    spi.write(READ_LOW);
    spi.write((addr & 0xFF0000) >> 16);
    spi.write((addr & 0x00FF00) >> 8);
    spi.write((addr & 0x0000FF));
    wait_ms(1);
    for(int j = 0; j < 65536; j++){
        for(int i = 0; i < 4 ; i++){
          test.buff[i]= spi.write(0xFF);
        }
    }
    wait_ms(100);
    cs  = 1;
}
```

Nejprve jsem tedy zapsal všechny možné reálné vzorky do paměti, což znázorňuje první diagram. Dále jsem potřeboval implementovat čtení z paměti do hlavního programu, který zobrazuje následující diagram.

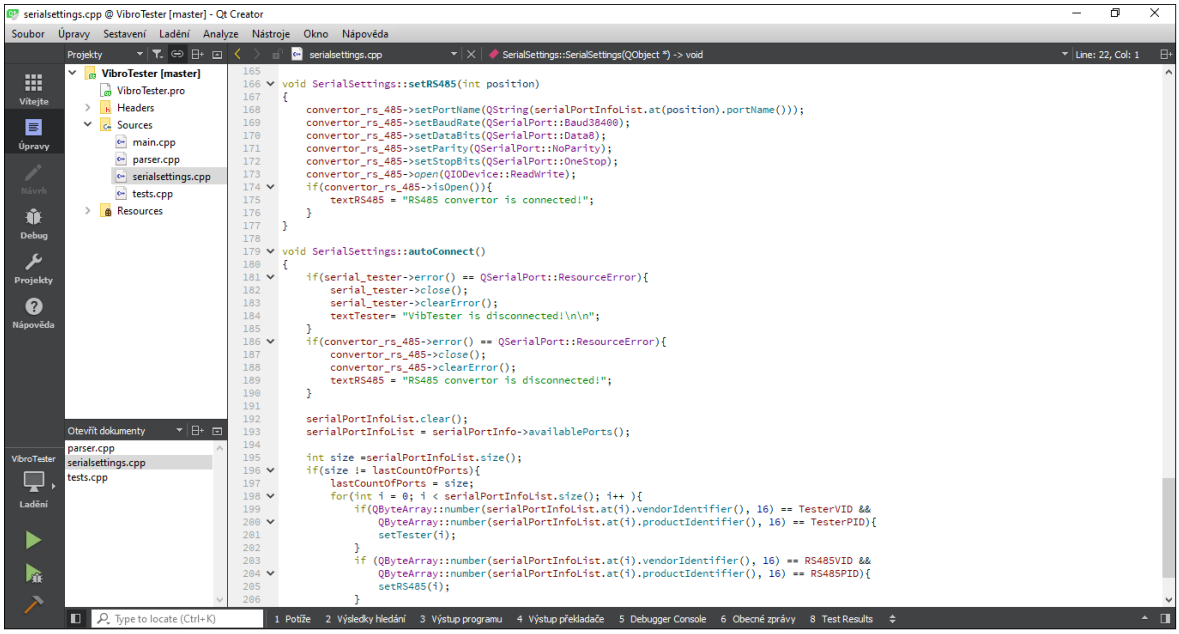


Obr. 9.6 Zjednodušený diagram čtení z EEPROM

10 Softwarový nástroj pro ovládaní testeru

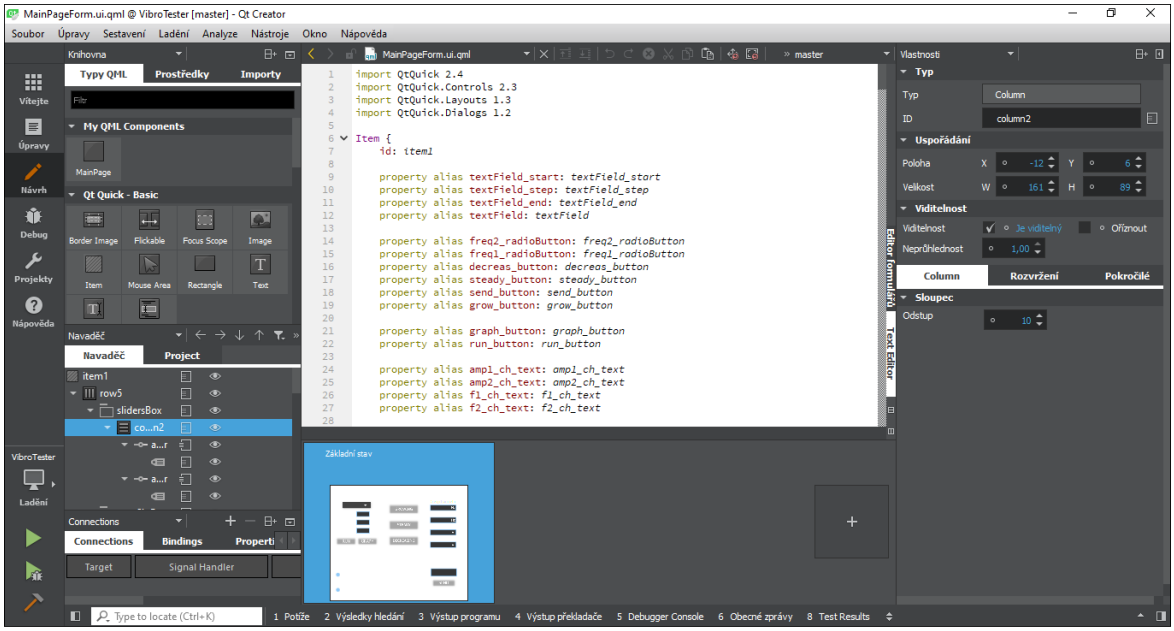
Softwarový nástroj pro ovládání testeru byl implementován v jazyce C++ za pomocí frameworku Qt. Qt je platforma pro vývoj multiplatformních aplikací jak pro desktopové, embedded a mobilní zařízení. Mezi podporované platformy patří Linux, Windows, Android, iOS, BlackBerry a další.

Qt není programovací jazyk, je to framework napsaný v C++. Je dostupný jako open-source, ale i jako komerční verze. Používá například MOC (Meta-Object Compiler) preprocesor, který se používá k rozšíření jazyka C++ o funkce, jako jsou signály a sloty. Framework může být kompilován standardními C++ kompilátory jako je MinGW, MSVC, GCC a další. Qt má také vlastní IDE, tedy integrované vývojové prostředí s názvem Qt Creator.



Obr. 10.1 Ukázka vývoje v IDE Qt Creator

Na vytváření grafického uživatelského rozhraní, tedy GUI jsem použil modul QtQuick. GUI používající QtQuick jsou psány v QML, což je deklarativní jazyk popisující objekt, který integruje JavaScript pro proceduální programování. QtQuick poskytuje sadu ovládacích prvků, které lze použít k vytváření kompletních GUI s QML. Aplikace je možné psát jen v QML, většinou se však QML používá jako frontend a backend aplikace je implementován v C++. QtCreator má integrovaný QtQuick GUI designer, kde lze vyvíjet GUI buď v textovém editoru, kde můžeme pomocí kódu nastavovat jednotlivé prvky nebo pomocí editoru formuláře, kde už objekty přetahujeme z nabídky a skládáme do hlavního okna a tento kód se automaticky generuje do textového editoru.



Obr. 10.2 Ukázka vývoje GUI pomocí QtQuick Designer

GUI testeru jsem rozdělil na šest částí. Každou část jsem vložil do komponenty zvané skupinový rámeček, která označuje a ohraničuje dané prvky. Pro přidávání více komponent se umisťují prvky buď do řádků, nebo sloupců, díky čemuž jsou tyto komponenty zaobaleny a lze nastavovat vzdálenost mezi jednotlivými komponentami.

Obr. 10.3 Ukázka GUI pro ovládání testeru

10.1 Skupinový rámeček Tests

V tomto skupinovém rámečku se nachází jedno kombinované pole s daným popiskem, kde si uživatel může vybrat, jaký typ testu spustí. Jsou zde nahrané názvy testů z XML souboru a také test s reálnými vzorky. Pro test s názvem "Sweep"jsou umístěny pod sebou tři textová pole. První určuje počáteční frekvenci, druhý po jakém kroku se frekvence budou generovat a poslední určuje konečnou hodnotu frekvence. Jako poslední komponenty jsou vedle sebe umístěny tlačítka Run a Graph. První jmenované spouští test, který je vybraný v kombinovaném poli. Druhé tlačítko umožňuje zobrazit graf naměřené frekvenční charakteristiky po dokončení Sweep testu.

10.2 Skupinový rámeček Value of amplitudes

V tomto skupinovém rámečku jsou dva posuvníky pro nastavování velikosti amplitudy. Pomocí posuvníků lze ve výsledku měnit velikost Vpp výsledného signálu od 93mV až do 3,26V.

10.3 Skupinový rámeček Data from VIBR

V tomto skupinovém rámečku jsou popisky k jednotlivým čtyřem rámečkům, kde se v pravidelných intervalech zobrazují příchozí data, které zpracovává a posílá Vibromodul. Jde o hodnoty dvou amplitudových a frekvenčních kanálů Vibromodulu.

10.4 Skupinový rámeček Frequency mode

V tomhle skupinovém rámečku jsou pod sebou umístěna tři tlačítka umožňující měnit frekvenční módy, tedy zda jsou frekvence stabilní, rostou, nebo klesají.

10.5 Skupinový rámeček Type of signal

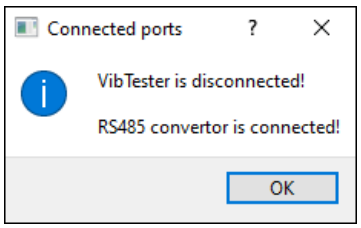
V tomto skupinovém rámečku jsou pod sebou umístěné čtyři přepínače. Tyto komponenty nám umožňují vybrat, jaký signál chceme měnit - zda jednu z frekvencí či amplitud. Lze vybrat jen jednu volbu. Frekvence můžeme měnit pomocí textového pole s tlačítkem ve skupinovém rámečku Value of frequency. Amplitudy lze pouze měnit pomocí posuvníků ve skupinovém rámečku Value of amplitude.

10.6 Skupinový rámeček Value of frequency

V tomto skupinovém rámečku je umístěno textové pole, do kterého se zapisuje hodnota frekvence, kterou chceme generovat testerem. Dále tlačítko, které po stisku pošle data, které mohou obsahovat desetinné číslo s jednou desetinnou čárkou, pokud tedy nabývají hodnot v rozmezí 1 až 1000. Zde jdou pouze nastavovat hodnoty frekvencí.

11 Implementace automatického připojení k virtuálnímu sériovému portu

Do počítačové aplikace bylo také implementováno automatické připojení zařízení jako je tester nebo převodník USB-RS485 k sériovému virtuálnímu portu daného PC. Kdykoliv během spuštěné aplikace, můžeme zařízení odpojovat či připojovat a každá zaznamenaná změna stavu je hlášena pomocí dialogového okna, které oznamuje, jaké zařízení je právě připojeno nebo odpojeno.



Obr. 11.1 Ukázka GUI pro ovládání testeru

Každé zařízení, které se připojuje do PC pomocí USB, by mělo mít jedinečný identifikátor, v tomto případě se jedná o Vendor ID (VID) a Product ID (PID). Jednotlivé identifikátory se skládají ze 16 bitového čísla. Vendor ID slouží k identifikaci výrobce, zatímco Product ID slouží k identifikaci specifického výrobku. Naše aplikace se tedy umí automaticky připojit k testeru a k převodníku od výrobce FTDI.

11.1 Funkce pro automatické připojení na COM port

Tato funkce nejprve kontroluje, zda nejsou zařízení odpojena, pokud ano, porty vybraných zařízeních jsou uzavřeny a je nastaven daný text označující tuto skutečnost. Dále je načten seznam všech dostupných portů. Tato funkce je volána automaticky každých 5 vteřin. Pokud je za tuhle dobu změněn počet dostupných portů, funkce tyto porty načte a pomocí cyklu porovnává jejich identifikátory VID a PID, jestli se neshodují se zařízeními, která chceme připojit. Pokud ano, jsou zavolány funkce, kde je předána pozice v seznamu všech dostupných portů zajištující správné připojení k zařízením a je nastaven text označující tuto skutečnost.

```
1  void SerialSettings::autoConnect()
2  {
3     if(serial_tester->error() == QSerialPort::ResourceError){
4           serial_tester->close();
5           serial_tester->clearError();
6           textTester= "VibTester is disconnected!\n\n";
7       }
8       if(convertor_rs_485->error() == QSerialPort::ResourceError){
9           convertor_rs_485->close();
10          convertor_rs_485->clearError();
11          textRS485 = "RS485 convertor is disconnected!";
12      }
13
14      serialPortInfoList.clear();
```

```
15      serialPortInfoList = serialPortInfo->availablePorts();
16
17      int size =serialPortInfoList.size();
18      if(size != lastCountOfPorts){
19          lastCountOfPorts = size;
20          for(int i = 0; i < serialPortInfoList.size(); i++ ){
21              if(QByteArray::number(serialPortInfoList.at(i).
    vendorIdentifier(), 16) == TesterVID &&
22                  QByteArray::number(serialPortInfoList.at(i).
    productIdentifier(), 16) == TesterPID){
23                  setTester(i);
24              }
25              if (QByteArray::number(serialPortInfoList.at(i).
    vendorIdentifier(), 16) == RS485VID &&
26                  QByteArray::number(serialPortInfoList.at(i).
    productIdentifier(), 16) == RS485PID){
27                  setRS485(i);
28              }
29          }
30          sendDialog("Connected ports", textTester + textRS485, "
    Information");
31      }
32  }
```

11.2 Funkce pro nastavení COM Portu

Do funkce je předána pozice vybraného sériového virtuálního portu, který odpovídá daným identifikátorům VID a PID. Dále jsou na tento port zavolány metody, které zařídí správné nastavení pro komunikaci mezi aplikací a zařízením. Také je nastaven text pro dialogové okno, že zařízení je připojeno.

```
1   void SerialSettings::setTester(int position)
2   {
3       if(!serial_tester->isOpen()){
4           serial_tester->setPortName(QString(serialPortInfoList.at(position).
    portName()));
5           serial_tester->setBaudRate(QSerialPort::Baud9600);
6           serial_tester->setDataBits(QSerialPort::Data8);
7           serial_tester->setParity(QSerialPort::NoParity);
8           serial_tester->setStopBits(QSerialPort::OneStop);
9           serial_tester->open(QIODevice::ReadWrite);
10          if(serial_tester->isOpen()){
11              textTester = "VibTester is connected!\n\n";
12          }
13      }
14  }
15
16  void SerialSettings::setRS485(int position)
17  {
18      convertor_rs_485->setPortName(QString(serialPortInfoList.at(position)
    .portName()));
19      convertor_rs_485->setBaudRate(QSerialPort::Baud38400);
20      convertor_rs_485->setDataBits(QSerialPort::Data8);
21      convertor_rs_485->setParity(QSerialPort::NoParity);
22      convertor_rs_485->setStopBits(QSerialPort::OneStop);
23      convertor_rs_485->open(QIODevice::ReadWrite);
24      if(convertor_rs_485->isOpen()){
25          textRS485 = "RS485 convertor is connected!";
26      }
27  }
```

12 Implementace automatizovaných testů testeru

Tester umožňuje tři automatizované testy:

- Test pomocí reálných vzorků
- Sweep test
- Test z xml souboru

12.1 Funkce k načítaní testů z konfiguračního souboru

Tester umožňuje nahrání uživatelského scénáře z XML souboru. Uživatel si může nadefinovat test pro několik kroků, kde se každý krok skládá ze čtyř nastavitelných hodnot. Musí být dodržen formát a pořadí nastavitelných hodnot. Pokud je nějaká z hodnot nastavena s překlepem nebo mimo rozsah, upozorní na to uživatele dialogové okno s daným textem. V jednom XML souboru může být definováno více testů oddělených pomocí elementu test name.

Hodnoty elementu variable může nabývat hodnot jako:

- f1 - frekvence prvního harmonického signálu
- f2 - frekvence druhého harmonického signálu
- amp1 - amplituda prvního harmonického signálu
- amp2 - amplituda druhého harmonického signálu

```
<?xml version = "1.0"?>
<tests>
<test name = "Test1">
        <variable>f1</variable>
        <value>350</value>
        <variable>f2</variable>
        <value>300</value>
        <variable>amp1</variable>
        <value>0.2</value>
    <variable>amp2</variable>
        <value>0.3</value>
</test>

<test name = "Test2">
        <variable>f1</variable>
        <value>100</value>
        <variable>f2</variable>
        <value>120</value>
        <variable>amp1</variable>
        <value>0.1</value>
    <variable>amp2</variable>
        <value>0.1</value>
</test>
</tests>
```

12.1.1 Funkce k načítaní hodnot z konfiguračního souboru

Tato funkce nejprve vytváří cestu k XML souboru, dále se načítá daný XML soubor. Pro přístup ke zpracování XML souboru je použit přístup přes DOM parser, který vezme XML dokument a vytvoří z něj stromovou strukturu v paměti, tady je využita funkce, která rekurzivně projde celou strukturu. Dále následuje ověření zda jsou načtené hodnoty správně napsané a v daném rozsahu.

```
void Tests::loadDataXML()
{
    QString directory = QDir::currentPath() + "/test1.xml";
    QDomDocument doc;
    QFile file(directory);
    if(file.open(QIODevice::ReadOnly)){
        if(doc.setContent(&file)){
            QDomElement elem = doc.documentElement();
                traverse(elem);
        }
        file.close();
    }
    for(int i = 0; i < values.size() / 2; i += 2){
        if(((values.at(i) == "f1" || values.at(i) == "f2") && (values.at(
    i+1).toFloat() > 1.0f && values.at(i+1).toFloat() <= 1000.0f))
            ||((values.at(i)== "amp1" || values.at(i)== "amp2") && (
    values.at(i+1).toFloat() > 0 && values.at(i+1).toFloat() <= 0.35f))){
        }
        else{
            isValidXml = true;
        }
    }
}
```

12.1.2 Funkce k rozřazovaní podle názvu elementů

Tato rekurzivní funkce prochází stromovou strukturu po jednom listu, kde na základě toho, že není prázdný rozřazuje podle názvu elementů. Hodnoty se buď přidávají do pole řetězců obsahující názvy testů, nebo pokud už je test vybrán, funkce přidává hodnoty elementů do pole řetězců, které se dále zpracují.

```
void Tests::traverse(const QDomNode &node)
{
    QDomNode domNode = node.firstChild();
    while(!domNode.isNull()){
        if(domNode.isElement()){
          QDomElement domElement = domNode.toElement();
          if(domElement.tagName() == "test" ){
              name.append(domElement.attribute("name",""));
              if(domElement.attribute("name","") == nameOfTest ){
                  isTest = true;
              }
              else{
                  isTest = false;
              }
          }
          else{
              if(isTest){
                values.append(domElement.text());
              }
          }
        }
```

```
22            traverse(domNode);
23            domNode = domNode.nextSibling();
24        }
25    }
```

12.2 Funkce pro Sweep test

Tato funkce nejprve vhodně upraví počáteční zadané hodnoty, pomocí podmínky se kontroluje, zda zadané hodnoty pro sweep test je možné vygenerovat. Je tady ošetřeno, aby počáteční frekvence nebyla větší než konečná a zároveň, aby zbytek rozdílu konečné a počáteční hodnoty s krokem, po kterém se frekvence bude generovat, byl roven nule. Pokud jsou tyto podmínky splněny, je pomocí cyklu naplněno pole hodnot. Pokud nejsou, je uživatel informován pomocí dialogového okna o špatně zadaných vstupních hodnotách pro test.

```
1  void Tests::getDataForSweep(QString start, QString step, QString end)
2  {
3      if(!start.contains('.')) start = start + ".0";
4      if(!step.contains('.')) step = step + ".0";
5      if(!end.contains('.')) end = end + ".0";
6      float Start = start.toFloat() * 10;
7      float Step = step.toFloat() * 10;
8      float End = end.toFloat() * 10;
9
10     if((Start < End) && (fmod (End - Start, Step) == 0.0f)){
11         noSweep = false;
12         for(float i = Start; i <= End; i+= Step){
13             sweepTestVal.append(i/10);
14         }
15     }
16     else{
17         sweepTestVal = {0.0,0.0};
18         waitForTest = 9;
19         noSweep = true;
20         sendDialog("Wrong input value", "Input values for sweep test are
       wrong", "Warning");
21     }
22 }
```

12.2.1 Vytváření grafu v aplikaci

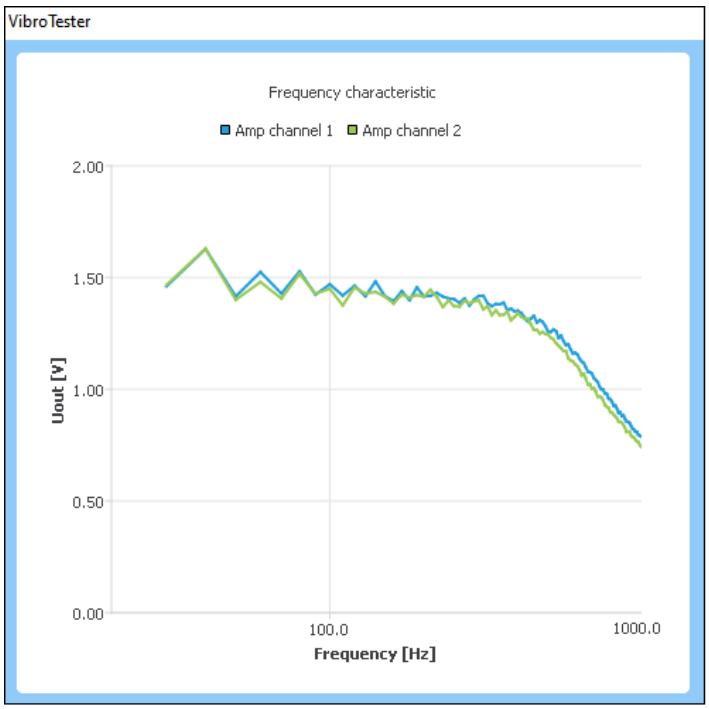
Aplikace také umožňuje vytvoření grafu frekvenční charakteristiky po dokončení Sweep testu, díky tomu může uživatel hned zkontrolovat, zda implementace analýzy vibračních signálů uvnitř Vibromodulu má odpovídající frekvenční charakteristiku. Po dokončení testu jsou všechny hodnoty uloženy do listů. Po stisknutí tlačítka uživatelem se načte nově vytvořené okno s grafem. Okno s grafem lze zavřít a dokud nebude spuštěn nový test, který aktuální hodnoty grafu přepíše, může se uživatel ke grafu kdykoliv vrátit.

Pro vytvoření grafu bylo potřeba implementovat osy x a y. Osa x je implementovaná pro frekvenci jako logaritmická o základu 10, kde minimální hodnota je 1 a maximální 1000. Osa y je implementovaná pro velikost výstupního napětí modulu měření vibrací, rozsah osy se pohybuje od 0 do 2 V.

```
axes: [
  LogValueAxis{
      id: xAxis
        base: 10
        min : 1
        max : 1000
        titleText : "Frequency [Hz]"
    },
    ValueAxis{
      id: yAxis
        min: 0.0
        max: 2.0
        titleText : "Uout [V]"
    }
]
```

Funkci, která zajišťuje vykreslení hodnot, jsou nejprve předány všechny hodnoty uložené do listů během běhu testu. Dále jsou vytvořeny dva průběhy pro oba amplitudové kanály, které jsou pomocí cyklu for naplněny naměřenými hodnotami.

```
function insertValuesToGraph() {
  var frequencies = tests.getFreq();
    var amplitude_1 = tests.getAmp1();
    var amplitude_2 = tests.getAmp2();
    var amp_Series_1 = chart.createSeries(ChartView.SeriesTypeLine, "Amp
    channel 1", xAxis, yAxis);
    var amp_Series_2 = chart.createSeries(ChartView.SeriesTypeLine, "Amp
    channel 2", xAxis, yAxis);
    for (var i in voltages_map){
      amp_Series_1.append(frequencies[i], amplitude_1[i])
        amp_Series_2.append(frequencies[i], amplitude_2[i])
    }
}
```



Obr. 12.1 Ukázka vygenerované frekvenční charakteristiky v aplikaci

12.3 Ukládání naměřených hodnot do CSV

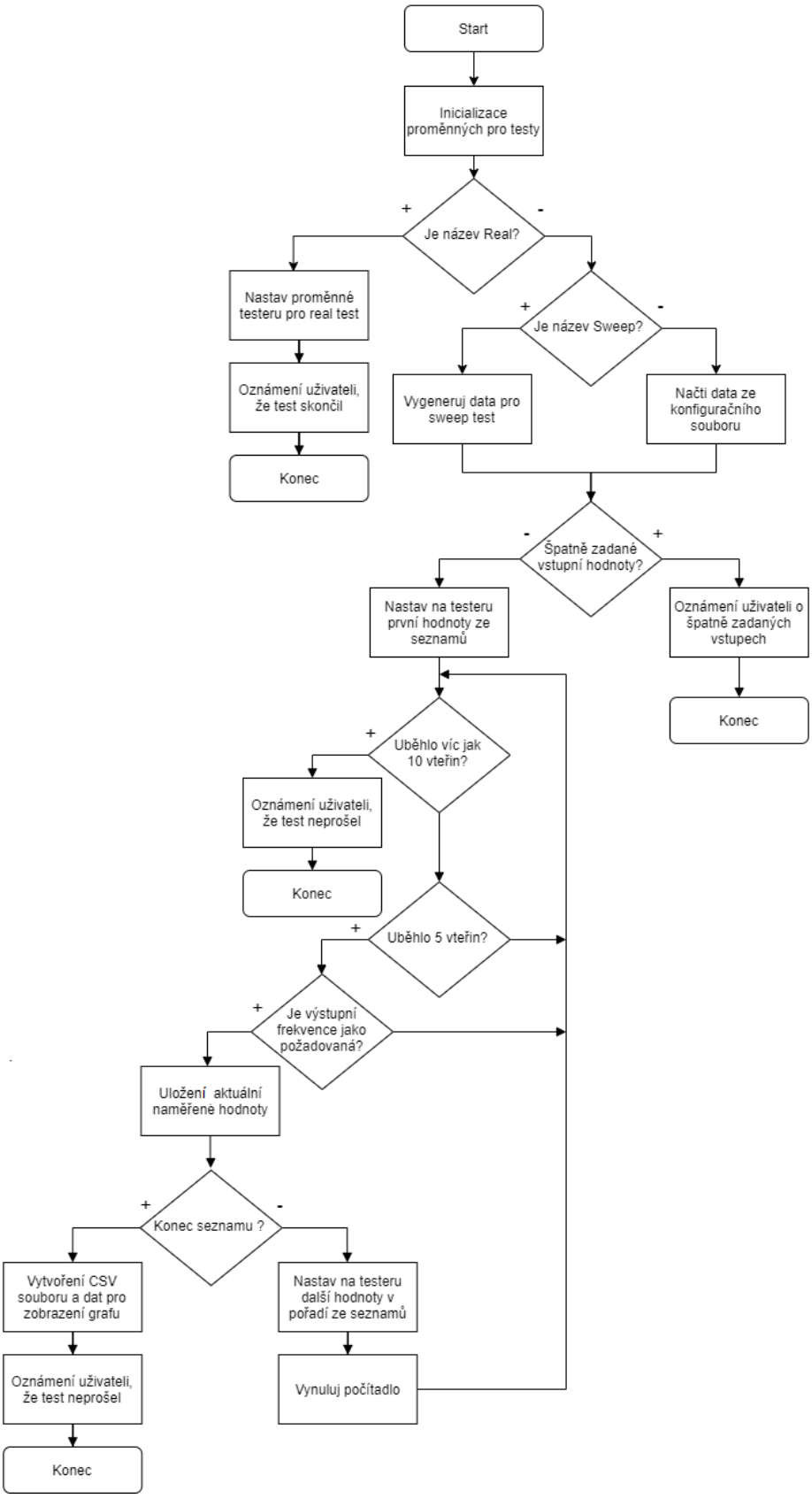
Aplikace také po dokončení Sweep testu nebo testu nahraného z konfiguračního souboru vytvoří CSV soubor s uloženými hodnotami během testu.

Než se zavolá funkce pro zápis do CSV souboru, je nutné data v listech upravit do formátu, aby se s nimi dalo v CSV souborech dál pracovat. Funkce na základě názvu vybraného testu vytvoří instanci QFile pro zápis do CSV souboru. Dále tato instance využívá QTextStream, kde pomocí cyklu for zapisujeme ze seznamů naměřené hodnoty, které vložíme ve vhodně upraveném formátu.

```
1  void Tests::WriteToCSV()
2  {
3      QFile file(nameOfTest + ".csv");
4          file.open(QIODevice::Append | QIODevice::Text);
5          modifyDataForCSV();
6          QTextStream stream(&file);
7          for (int i = 0; i < f1.size(); ++i){
8              stream << f1.at(i) << ";" << f2.at(i) << ";"  << a1.at(i) <<
       ";" << a2.at(i) << endl;
9          }
10         stream.flush();
11         file.close();
12 }
```

12.4 Funkce při výběru testu

Při spuštění funkce zajišťující automatické testování se inicializují všechny proměnné pro testování do defaultní hodnoty a na základě názvu testu jsou volány další funkce k jeho pokračování. Když je vybrán test z nahraných vzorků z paměti, vypnou se všechna tlačítka a přes sériovou linku jsou poslány příkazy ke spuštění testu trvajícího 16 vteřin. Pokud je vybrán test z konfiguračního souboru nebo sweep test, je nejprve zkontrolováno, zda jsou data k generování na testeru správná. Funkce využívá signálů a slotů, kde pomocí časovače se každou vteřinu pošlou naměřená data z výstupu modulu měření vibrací. Díky tomu se po jedné vteřině inkrementuje počítadlo. Každý krok trvá pět sekund, potom se generují další hodnoty ze seznamu vybraného testu až do konce. Aplikace kontroluje, zda nastavené frekvence zpracoval Vibromodul s tolerancí ±1Hz, pokud do deseti sekund nezaznamená tuto hodnotu, uživatelem vybraný test se ukončí. Když tyto naměřené hodnoty zaznamená, jsou tyto hodnoty uloženy do seznamu spolu s výstupním napětím modulu měření vibrací, kde seznam je využit pro generování grafu nebo CSV souboru a zároveň jsou další hodnoty v pořadí seznamu poslány po sériové lince testeru. Každý test kromě testu s reálnými vzorky lze zastavit během jeho průběhu.



Obr. 12.2 Zjednodušený diagram při spuštení testu

13 Implementace jednotlivých funkcí

Třída s názvem SerialSettings se stará o sériovou komunikaci a její nastavení. Nejdříve jsem implementoval konstruktor, kde jsem inicializoval instance pro dva sériové porty. Jeden pro tester, druhý pro Vibromodul a také instanci poskytující informace o existujících sériových portech.

Dále dva časovače, jeden pro čtení dat a druhý pro volaní funkce, která aktualizuje nalezené porty, aby se aplikace nemusela po odpojení znova zapínat. Tyto časovače jde spojit metodou *connect*, kde časovač po vypršení určitého času pošle signál *timeout*, který přijme funkce definovaná jako slot a vykoná se.

```
SerialSettings::SerialSettings(QObject *parent)  : QObject(parent)
{
    serialPortInfo = new QSerialPortInfo();
    serial_tester = new QSerialPort(this);
    convertor_rs_485 = new QSerialPort(this);
    parser = new Parser();
    timer_autoConnect = new QTimer(this);
    timer_readData = new QTimer(this);
    timer_autoConnect->start(2500);
    timer_readData->start(1000);
    connect(timer_readData, &QTimer::timeout, this, &SerialSettings::
    readData);
    connect(timer_autoConnect, &QTimer::timeout, this, &SerialSettings::
    autoConnect);
}
```

13.1 Funkce pro čtení ze sériového portu

Funkce slouží ke čtení dat ze sériového virtuálního portu, kde přichází data z modulu měření vibrací. Nejprve je zkontrolováno, zda převodník RS485-USB je připojen. Tato data jsou předána do funkce, kde se zkonvertují a po návratu z funkce jsou poslány jako signál, který se zobrazí v GUI. Dále se všechna data vymažou a znovu naplní v dalším volání funkce, které zajišťuje časovač s periodou jedna sekunda.

```
void SerialSettings::readData()
{
    if(convertor_rs_485->isOpen()){
        data = convertor_rs_485->readAll();
        emit updateFreq1(parser->parseString(data, f1));
        emit updateFreq2(parser->parseString(data, f2));
        emit updateAmp1(parser->parseString(data, a1));
        emit updateAmp2(parser->parseString(data, a2));
        data.clear();
    }
}
```

13.2 Funkce pro parsování příchozích dat

Tato funkce kontroluje, zda přijatá data obsahují a začínají danou hlavičku, pokud ano jsou tato data předány funkci k parsovaní. Pokud přijatá data hlavičku obsahují, ale

hlavičkou nezačínají, následuje cyklus while, ve kterém se budou data v poli posouvat dokud nebudou hlavičkou začínat, aby mohlo dojít k parsovaní.

```
QString Parser::parseString(QByteArray data, const QByteArray header)
{
    QString result;
    QByteArray tmp = data;
    if(tmp.contains(header)){
        if(tmp.startsWith(header)){
            tmp.remove(25,tmp.size());
           result = parseData(tmp);
        }
        else{
            while(!tmp.startsWith(header)) {
                int n = tmp.size();
                char temp = tmp.at(0);
                for(int i = 0; i < n-1; i++)
                {
                    tmp[i] = tmp[i+1];
                }
                tmp[n-1] = temp;
            }
            tmp.remove(25,tmp.size());
          result = parseData(tmp);
        }
    }
    return result;
}
```

Této funkci jsou předána data, které začínají jednou ze čtyř hlaviček. Tato funkce převede předaná data na jednotlivé znaky do seznamu řetězců v čitelné hexadecimální podobě, dále se jednotlivé znaky vhodně mění do podoby jednoho celého řetězce, aby data mohla být převedena na celé kladné 32 bitový číslo. To je vzápětí přetypováno na datový typ float a výsledná rozparsovaná hodnota je vrácena.

```

QString Parser::parseData(QByteArray data)
{
  QString str;
    QStringList array;
    for(int i = 0; i < data.size(); i++){
        array.append(QString::number(static_cast<unsigned char>(data[i]),
   16).toUpper().rightJustified(2,'0'));
        if(array.at(i).contains(QRegExp("[A-Z]"))){
            str += array.at(i) + " ";
        }
        else{
            QChar character  = data.at(i);
            QString pom = character;
            str += pom.toLower() + " ";
        }
    }
  str = modifyStr(str);
    bool ok;
    quint32 hex = str.toUInt(&ok, 16);
    const float *f = reinterpret_cast<const float *>(&hex);
    return  QString::number(double(*f)).remove(8,5);
}
```

13.3 Funkce zajištující ovládání testeru

Zde jsou funkce, které zajišťují posílání příkazů k ovládaní testeru, pokud je tester připojen. Tyto funkce zajišťují poslaní příkazů pro změnu frekvenčních módů. Zde je ukázka pro poslání příkazu pro inkrementaci frekvenci. Stejným způsobem jsou implementovány i zbylé frekvenční módy.

```
1
2 void SerialSettings::on_btnGrow_clicked()
3 {
4     if(serial_tester->isOpen()) writeData(INCREMENT);
5 }
```

Tyto funkce zajišťují poslání příkazů pro změnu typu signálu, který chceme nastavit. Zde je ukázka změny proměnné frekvence prvního harmonického signálu, stejným způsobem jsou implementovány i zbylé proměnné dvou harmonických signálů.

```
1 void SerialSettings::on_freq1_RB_clicked()
2 {
3     if(serial_tester->isOpen()) writeData(FREQ_1);
4 }
```

Ovládaní amplitud harmonických signálu je implementováno pomocí slideru. Zde je ukázka, kde je hodnota upravena a je předána do funkce zajištující přenos po sériové lince do testeru.

```
1 void SerialSettings::on_amp1_Slider_sliderMoved(QByteArray val)
2 {
3     val = val.remove(4,20);
4     setSliderData(val);
5 }
```

Tato funkce zajištuje správnost dat zapsaných do textového pole k nastavení frekvence jednoho z harmonických signálů. Pokud jedna z podmínek není splněna, je poslán signál s daným textem k vyvolaní dialogu zprávy oznamující tuto skutečnost.

```
1 void SerialSettings::on_btnSend_clicked(QByteArray val)
2 {
3     if(!val.contains('.')) val = val + ".0";
4     if(serial_tester->isOpen() && !val.isEmpty() && (val.length() - val.
    indexOf(".") - 1) == 1 ){
5         float value = val.toFloat();
6         if(value >= 1 && value < 100){
7                 val.append(SMALLER_NUM);
8                 writeData(val);
9         }
10        else if((value >= 100) && (value < 1000)){
11          writeData(val);
12        }
13        else if(value == 1000.0f){
14          writeData("1000.");
15        }
16        else{
17          sendDialog("Wrong input value", "Input value should be between
    1 and 1000 Hz", "Warning");
18        }
19    }
20 }
```

13.4 Propojení QML s třídami

Abych mohl používat metody z vybraných tříd na dané komponenty v grafickém uživatelském rozhraní, musel jsem nejdříve vytvořit instance ve třídě main. Dále jsem pomocí metody nastavil jejich název, abych k metodám instancí přistupovat i v QML.

```
1     SerialSettings serial;
2     Tests test;
3     Parser parser;
4     QObject::connect(&test, &Tests::sendDataToSerial, &serial, &
      SerialSettings::sendTestData);
5     context->setContextProperty("tests",&test);
6     context->setContextProperty("serialSettings",&serial);
7     context->setContextProperty("testsModel", QVariant::fromValue(test.
      getTests()));
```

QML má mechanismus signálu a obsluhy, kde signál je událost a ta je zpracována pomocí obsluhy signálu. Umístění logiky, jako je skript nebo jiné operace do obslužné rutiny umožňuje reagovat na událost. Objekt Connections může přijímat jakýkoli signál ze zadané instance. Když chceme zpracovat určitý signál, musí se definovat obsluhu signálu pojmenovanou jako on<signal>, kde signál je název signálu s velkým písmenem na začátku.

Toho jsem využil a implementoval zpracování pro signály z tříd Test a SerialSettings, kde vyvolávám signály, které posílají text informující uživatele o tom, co se stalo, i spolu s nastavením, jaký typ dialogového okna se má zobrazit, tedy varování nebo informativní. Dále jsem objekt Connections využil pro zpracování čtyř signálů ze třídy SerialSettings, které při emitování signálu posílají i hodnotu rozparsovaného signálu z modulu měření vibrací, která se pak v textovém poli přepíše na novou hodnotu.

```
1     Connections {
2         target: serialSettings
3
4         onUpdateFreq1 : {
5             f1_ch_text.text = val;
6         }
7
8         onUpdateFreq2 : {
9             f2_ch_text.text = val;
10        }
11
12        onUpdateAmp1 : {
13            amp1_ch_text.text = val;
14        }
15
16        onUpdateAmp2 : {
17            amp2_ch_text.text = val;
18        }
19
20        onSendDialog : {
21            messageDialogSerial.title = title;
22            messageDialogSerial.text = text;
23            if(type =="Warning"){
24                messageDialogSerial.icon = "Warning";
25            }
26            else{
27                messageDialogSerial.icon = "Information";
28            }
29            messageDialogSerial.open();
30        }
31    }
```

13.5 Implementace funkcí komponentů GUI

Dále zde jsou ukázky implementací funkce pro posuvník, která se zavolá kdykoliv se změní hodnota na posuvníku. Jeho hodnota se předá do metody implementované ve třídě SerialSettings.

```
amp1_slider.onValueChanged: {
  if(amp1_radioButton.checked){
            serialSettings.on_amp1_Slider_sliderMoved(amp1_slider.value);
    }
}
```

Následuje zpracovaní události, která zavolá metodu, když uživatel klikne na vybraný přepínač. Stejným stylem jsou implementované i zbylé přepínače.

```
freq1_radioButton.onClicked: {
  serial.on_freq1_RB_clicked();
}
```

Další ukázka je pro zpracování události stisku tlačítka pro zobrazení grafu. Když po vykonaném testu stiskneme tlačítko "Graph", nastaví se zdroj nové stránky na soubor, který vygeneruje graf frekvenční charakteristiky. Zároveň se změnil text tohoto tlačítka na "Close". Pokud klikneme na tlačítko znovu, zdroj stránky se přepíše na prázdné uvozovky, čímž dojde k zavření stránky s grafem.

```
graph_button.onClicked: {
  if(graph_button.text == "Close"){
      pageLoader.source = "";
        graph_button.text = "Graph"
    }
    else if(graph_button.text == "Graph"){
      graph_button.text = "Close";
        pageLoader.source = "Graph.qml";
    }
}
```

Dále například zpracovávám událost stisku tlačítka pro spuštění testu. Nejprve se ověří, že už aplikace zobrazuje rozparsované hodnoty, pokud tomu tak není, nejde test spustit. V opačném případě po stisku tlačítka je zavolána metoda, která spustí uživatelem vybraný test a nastaví text tlačítka na "Stop". Pokud test běží a uživatel stiskne tlačítko, test se zastaví. Dialogové okno nás bude informovat o přerušení testu a nastavení zpět na spuštění od začátku.

14 Porovnání testeru s generátorem signálů

Bylo potřeba zjistit, zda výstupy generované testerem jsou zkalibrované. Rozhodl jsem se tedy porovnat výsledky modulu měření vibrací s laboratorním generátorem signálů RIGOL DG2041A. Měření výstupního napětí modulu měření vibrací probíhalo při vstupní amplitudě harmonického signálu o velikosti 50mV.

Tab. 14.1 Porovnání výsledků výstupního napětí mezi generátorem a testerem

	Generátor signálů	**Tester**
Frekvence [Hz]	**Uout [V]**	**Uout [V]**
20	0,845	0,837
70	1,100	1,116
120	1,085	1,070
170	1,030	1,019
220	0,959	0,959
270	0,900	0,890
320	0,839	0,826
370	0,770	0,763
420	0,700	0,712
470	0,680	0,665
520	0,620	0,625
570	0,593	0,582
620	0,562	0,549
670	0,500	0,512
720	0,490	0,494
770	0,468	0,464
820	0,446	0,438
870	0,417	0,418
920	0,404	0,398
970	0,385	0,378
999	0,372	0,370

Z tabulky můžeme vidět, že výstupní hodnoty modulu měření vibrací vychází jak pro generátor signálů, tak pro tester podobně.

ZÁVĚR

Teoretická část mojí práce začíná úvodem do problematiky vibrací a jejich matematického popisu. Následuje pohled do historie leteckých motorů a inovátorů, kteří se zasloužili o jejich vývoj. Dále jsem popsal skladbu leteckého motoru a jeho jednotlivé části, relevantní pro měření vibrací. Zahrnul jsem zde také popis modulu měření vibrací leteckých motorů (Vibromodulu) a požadavky na návrh testeru tohoto modulu.

V praktické části se nejprve zabývám vytvořením generátoru dvou nezávislých zdrojů vibrací v programu Scilab. Dále popisem architektury hardwarové realizace testeru modulu měření vibrací. Hlavní část praktické části mojí diplomové práce spočívala v návrhu schématu testeru a vytvoření desky plošných spojů, k čemuž jsem použil program Altium Designer 18. Pro desku plošných spojů jsem vytvořil kryt - krabičku v programu Fusion 360, kterou jsem vytiskl na 3D tiskárně. Další částí této práce je implementace firmware testeru umožňující generovat signály simulující výstupy otáčkoměrů a senzoru vibrací. Firmware jsem implementoval v jazyce C++ za pomocí online IDE Mbed. Pro ovládání testeru jsem vytvořil program s grafickým uživatelským rozhraním v jazyce C++ v programu Qt Creator 4.7.2. Součástí implementace jsou také automatizované testy pro zjednodušení testovaní modulu měření vibrací pomocí mého testeru. Implementoval jsem tři typy automatických testů: ve formě uživatelského scénáře z konfiguračního souboru, ve formě testu ke zjištění frekvenční charakteristiky modulu měření vibrací a ve formě testu s využitím reálných vibračních signálů, uložených v EEPROM paměti mého testeru. Na závěr práce porovnávám výstupní hodnoty modulu měření vibrací při generovaní vstupních signálů pomocí testeru a generátoru signálů.

SEZNAM POUŽITÉ LITERATURY

[1] *BUZDUGAN, Gheorghe. Vibration measurement. Netherlands: Springer, 2010, 345 s. ISBN 978-90-481-8287-9.* [cit. 2020-7-11].

[2] *SMETANA, Ctirad, et al. Hluk a vibrace : Měření a hodnocení. 1. vydání. Praha 1 : Sdělovací technika, 1998. 188 s. ISBN 80-901936-2-5* [cit. 2020-7-11].

[3] *VEER, I. L. a Leo L. BERANEK. Noise and vibration control engineering: principles and applications. 2nd ed. Hoboken, N.J.: Wiley, c2006. ISBN 978-0-471-44942-3.* [cit. 2020-7-11].

[4] *CARIOLARO, Gianfranco. Unified Signal Theory. London: Springer, 2011, 927 s. ISBN 978-0-85729-463-0* [cit. 2020-7-11].

[5] *RICHARDS, Keith L. Design Engineer's Sourcebook. CRC Press, 2018, 1182 s. ISBN 978-1-4987-6341-7* [cit. 2020-7-11].

[6] *Sine Wave Background* [online]. [cit. 2020-7-11]. Dostupný z WWW: `https://www.kissclipart.com/sine-curve-png-clipart-line-sine-wave-c5y0om/`

[7] *DOYLE, James. F. Nonlinear Structural Dynamics Using FE Methods. Cambridge: Cambridge University Press, 2015. ISBN 978-1-107-04570-5.* [cit. 2020-7-11].

[8] *Royce, Rolls. The Jet Engine. 5th Edition. John Wiley & Sons, 2015, 288 s. ISBN 978-1-119-06599-9.* [cit. 2020-7-11].

[9] *CHAJDA, Radek. Dobrodružství vzduchoplavby. V Brně: Edika, 2018. ISBN 978-80-266-1300-8* [cit. 2020-7-11].

[10] *Félix Du Temple de la Croix (1823-1890)* [online]. [cit. 2020-7-11]. Dostupný z WWW: `http://www.ctie.monash.edu.au/hargrave/du_temple.html`.

[11] *Samuel Pierpont Langley* [online]. [cit. 2020-7-11]. Dostupný z WWW: `http://www.flyingmachines.org/lang.html`.

[12] *Wright Brothers* [online]. [cit. 2020-7-11]. Dostupný z WWW: `https://www.history.com/topics/inventions/wright-brothers`.

[13] *Frank Whittle (1907 - 1996)* [online]. [cit. 2020-7-11]. Dostupný z WWW: `http://www.bbc.co.uk/history/historic_figures/whittle_frank.shtml`.

[14] *Hans Joachim Pabst von Ohain* [online]. [cit. 2020-7-11]. Dostupný z WWW: `https://www.britannica.com/biography/Hans-Joachim-Pabst-von-Ohain`.

[15] *DULÍK, T. a POSPÍŠILÍK, M. a OPLUŠTIL, V. a BENEŠ, P. (2020). Aircraft Turboprop Engine Vibration Monitoring Module. [Manuscript submitted for publication]. Faculty of Applied Informatics, Tomas Bata University in Zlin* [cit. 2020-7-11].

[16] *Turboprops, propfans, and unducted fan engines* [online]. [cit. 2020-7-11]. Dostupný z WWW: `https://www.britannica.com/technology/jet-engine/Turboprops-propfans-and-unducted-fan-engines`.

[17] *SFORZA, P. M. Theory of aerospace propulsion. Waltham, MA: Butterworth-Heinemann, c2012. Elsevier aerospace engineering series. ISBN 978-1-85617-912-6.* [cit. 2020-7-11].

[18] *BODEN, F. Advanced in-flight measurement techniques. New York: Springer, 2013, 344 s. ISBN 978-3-642-34738-2.* [cit. 2020-7-11].

[19] *VÁŇA, Vladimír. ARM pro začátečníky. 1. vyd. Praha: BEN - technická literatura, 2009, 195 s. ISBN 978-80-7300-246-6.* [cit. 2020-7-11].

[20] *NUCLEO-L432KC* [online]. [cit. 2020-7-11]. Dostupný z WWW: `https://os.mbed.com/platforms/ST-Nucleo-L432KC`.

[21] *BARR, Michael a Anthony J. MASSA. Programming embedded systems: with C and GNU development tools. 2nd ed. Sebastopol: O'Reilly, 2006. ISBN 978-0-596-00983-0.* [cit. 2020-7-11].

[22] *Serial Handbook* [online]. [cit. 2020-7-11]. Dostupný z WWW: `https://os.mbed.com/handbook/Serial`

[23] *Interface specification for airborne CAN applications V 1.7* [online]. [cit. 2020-7-11]. Dostupný z WWW: `https://www.stockflightsystems.com/tl_files/downloads/canaerospace/canas_17.pdf`.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

HW	Hardware
SW	Software
GG	Gas Generator
PT	Power Turbine
CAN	Controller Area Network
DSP	Digitální Signálový Procesor
DFT	Diskrétní Fourierova transformace
MAC	Multiply–accumulate
PC	Personal Computer
RISC	Reduced Instruction Set Computer
FPU	Floating Point Unit
CPU	Central Processing Unit
SPI	Serial Peripheral Interface
I2C	Inter-Integrated Circuit
I2S	Inter-IC Sound
USART	Universal Synchronous / Asynchronous Receiver and Transmitter
GPIO	General-Purpose Input / Output
RTC	Real-Time Clock
LED	Light-Emitting Diode
USB	Universal Serial Bus
MCU	MicroController Unit
LCD	Liquid-Crystal Display
EEPROM	Electrically Erasable Programmable Read-Only Memory
PWM	Pulse-Width Modulation
SCLK	Serial Clock
CLK	Serial Clock
MOSI	Master Out Slave In
MISO	Master In Slave Out
SS	Slave Select
SO	Slave In
SI	Slave Out
IoT	Internet of Things
HAL	Hardware Abstraction Layer
IDE	Integrated Development Environment
OS	Operating System
MOC	Meta-Object Compiler

QML	Qt Modeling Language
GUI	Graphical User Interface
XML	eXtensible Markup Language
VIR	Vendor ID
PIR	Product ID
DOM	Document Object Model
CSV	Comma-Separated Values

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P VII. MOTIV PLOŠNÉHO SPOJE