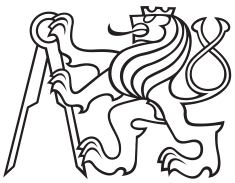
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra elektromagnetického pole

Univerzální GUI pro osciloskopické PC aplikace

Jiří Maier

Vedoucí práce: doc. Ing. Jan Fischer, CSc.
Studijní program: Elektronika a komunikace 2018
Květen 2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Maier** Jméno: **Jiří** Osobní číslo: **483896**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra elektromagnetického pole**

Studijní program: **Elektronika a komunikace**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Univerzální GUI pro osciloskopické PC aplikace

Název bakalářské práce anglicky:

The Universal GUI for PC Based Oscillographs

Pokyny pro vypracování:

Navrhněte a realizujte program s GUI pro univerzální použití při realizaci osciloskopů za pomoci mikrořadičů a PC. Návrh orientujte tak, aby výsledný produkt byl využitelný přímo v projektově orientované výuce a v dalších studentských projektech při realizaci SDI (softwarově definovaných přístrojů) typu osciloskop, analyzátor signálu, generátor, s mikrořadiči řady STM32. Vytvořte potřebné varianty komunikačního protokolu a s využitím nástroje „Qt framework“ realizujte kompletní PC aplikaci pro zobrazení průběhu signálů.
Navrhněte způsoby ovládání přístrojů za pomoci integrovaného textového zobrazovacího a komunikačního modulu s podporou ANSI sekvencí. S využitím hotové aplikace prověřte možnost realizace osciloskopů a generátorů s mikrořadiči STM32F303, STM32L412, STM32L072, případně dalších typů podle Vašeho výběru.

Seznam doporučené literatury:

[1] Yiu, J.:The Definitive Guide to ARM® Cortex®-M3 and Cortex®-M4 Processors, Newnes 2014
[2] STMicroelectronics: RM0316, STM32F3 Reference manual, dostupné z http://www.st.com
[3] Qt: Cross-platform software development for embedded & desktop [online] https://www.qt.io/

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Jan Fischer, CSc., katedra měření FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **21.01.2021** Termín odevzdání bakalářské práce: **21.05.2021**

Platnost zadání bakalářské práce: **30.09.2022**

doc. Ing. Jan Fischer, CSc. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

Poděkování

Tímto děkuji svému vedoucímu doc. Janu Fischerovi za odborné vedení práce a za čas, který mi věnoval při konzultacích. Dále děkuji své rodině a přátelům za podporu během celého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 21. května 2021

Jiří Maier

Abstrakt

Tato práce se zabývá návrhem a tvorbou grafické aplikace pro zobrazení a zpracování měřených dat přicházejících z mikrokontroleru. Aplikace je navržena s ohledem na univerzální použití se softwarově definovanými přístroji jako například osciloskop nebo logický analyzátor. Významnou součástí je komunikační terminál, podporující ANSI escape sekvence, umožňující vytvořit pseudografické rozhraní pro zobrazení měřených hodnot a nastavení přístroje. Terminál dokonce umožňuje vytvoření interaktivního menu pro ovládání přístroje. Součástí aplikace je i výpočet spektra pomocí FFT, interpolace signálu a výpočet frekvence. Aplikace je vytvořena v programovacím jazyce C++ s použitím Qt framework a komponenty QCustomPlot. V rámci práce byl také vytvořen osciloskop založený na mikrokontroleru STM32.

Klíčová slova: Nucleo osciloskop, STM32 osciloskop, STM32, Qt graf, Qt serial port, QCustomPlot, ANSI escape, FFT, interpolace signálu, STM32F3, Nucleo-F3, STM32L0, STM32L4, SDI

Vedoucí práce: doc. Ing. Jan Fischer, CSc.

Abstract

This thesis describes designing and developing an application for plotting and processing measured data sent by a microcontroller. Application is designed regarding universal use with software–defined instruments like oscilloscopes or logic analyzers. An important feature is a communication terminal, which supports ANSI escape sequences. That allows creating pseudo–graphical user interface for displaying measured values and device settings. It is even possible to create an interactive menu using the terminal. Application is also capable of computing FFT, interpolating signal, and calculating frequency. This application is developed using C++, Qt framework, and QCustomPlot. This thesis also includes creating an oscilloscope using an STM32 microcontroller.

Keywords: Nucleo osciloscope, STM32 osciloscope, STM32, Qt plot, Qt serial port, QCustomPlot, ANSI escape, FFT, signal interpolation, STM32F3, Nucleo-F3, STM32L0, STM32L4, SDI

Title translation: The Universal GUI for PC Based Oscillographs

Obsah

Obrázky

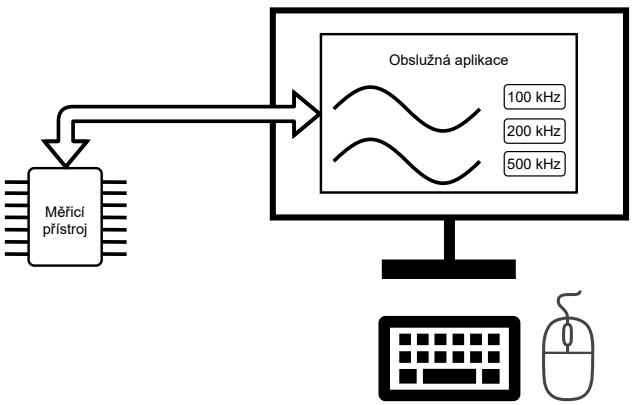
Tabulky

Kapitola 1

Úvod

Moderní mikrokontrolery z řady STM32 jsou vybavené rychlými analogově digitálními převodníky a disponují dostatečným výpočetním výkonem k tomu, aby bylo možné s pomocí nich vytvořit prakticky použitelné laboratorní přístroje. Takovéto přístroje samozřejmě nemohou konkurovat moderním profesionálním přístrojům, ale jsou dostatečné pro amatérské použití či využití ve výuce. Nízká cena takovýchto přístrojů umožňuje poskytnout přístroje všem studentům i na domácí použití, což je výhodné při projektově orientované výuce a distanční výuce.

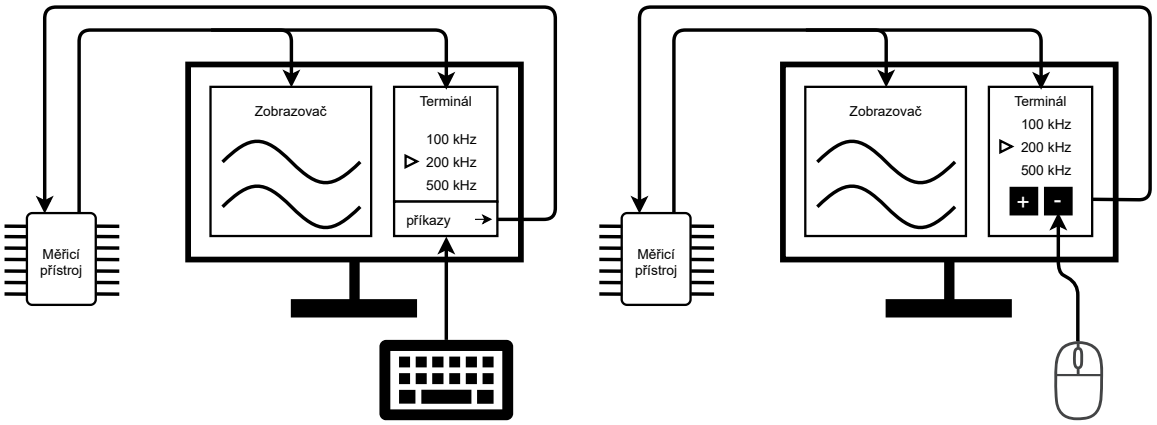
V minulosti již na katedře měření FEL ČVUT vzniklo několik projektů na téma vytvoření jednoduchých laboratorních přístrojů s využitím mikrokontrolerů STM32. Jako příklad lze uvézt platformu LEO (Little Embedded Oscilloscope [1]), umožňující použít vývojovou desku Nucleo jako osciloskop či generátor signálu. Takovéto projekty jsou vždy kombinací mikrokontroleru a zobrazovací aplikace pro počítač a jsou vytvořeny pro konkrétní typ mikrokontroleru (například platforma LEO je dostupná pouze pro mikrokontrolery STM32F303 a STM32F401).



Obrázek 1.1: Klasický přístup k tvorbě SDI pomocí mikrokontroleru

Takový přístup není optimální z toho důvodu, že různá pracoviště disponují různými typy mikrokontrolerů a je tedy nutné vytvořit firmware i zobrazovací aplikaci pro každý typ zvlášť. Navíc jsou vyvíjeny stále novější typy mikrokontrolerů, čímž se z tvorby osciloskopů pro každý z nich stává nikdy nekončící proces tvorby nejen firmwarů, ale i zobrazovacích aplikací.

Tato práce se snaží situaci řešit tím, že se soustředí na vytvoření univerzální aplikace, kterou by bylo možné využít s mnoha druhy přístrojů na různých typech mikrokontrolerů. Hlavní myšlenkou tohoto nového přístupu je co nejvíce oddělit zobrazování měřených dat od ovládání přístroje. Odesílání dat k zobrazení v grafu je plně v režii mikrokontroleru. Není vyžadováno, aby mikrokontroler přijímal jakékoli zprávy z aplikace. V případě ovládání samozřejmě je nutné komunikovat i směrem z počítačové aplikace do mikrokontroleru, ale konkrétní podoba ovládacího rozhraní je ponechána na tvůrci firmwaru pro mikrokontroler. Toho je dosaženo tím, že aplikace obsahuje terminál s podporou ANSI escape sekvencí, který umožní vytvoření pseudo–grafického uživatelského rozhraní pomocí textových příkazů z mikrokontroleru.



(a) : Ovládání pomocí terminálu **(b) :** Interaktivní terminál

Obrázek 1.2: Nový přístup k tvorbě SDI pomocí mikrokontroleru

Samozřejmě je na místě obava, že ovládání s pomocí terminálu bude mnohem méně pohodlné, než ovládání s pomocí plnohodnotného GUI v případě aplikace vytvořené pro konkrétní zařízení. Zdá se totiž, že jedinou možností ovládání je zadávání textových příkazů. Existují ale i způsoby, jak v terminálu vytvořit interaktivní menu pro ovládání pomocí klikání myší. A to bez újmy na univerzálnosti a jednoduchosti použití z pohledu firmwaru pro mikrokontroler.

V této práci se budu zabývat nejen tvorbou samotné aplikace, ale i vývojem osciloskopu s využitím několika různých typů mikrokontrolerů řady STM32. Pokusím se tak prokázat, že tímto přístupem lze vytvořit přístroj na stejné úrovni použitelnosti jako již existující projekty (např. LEO), který bude zároveň snadné portovat na jiné typy mikrokontrolerů.

Aby byla aplikace skutečně užitečná, musí nabízet funkce stejné či lepší, jako již existující platformy. Pokusím se proto implementovat pokročilé zpracování přijatých dat, jako například automatická měření charakteristik signálů, výpočet spektra algoritmem FFT nebo interpolaci signálu pro rekonstrukci harmonického průběhu z malého počtu vzorků na periodě.

Kapitola 2

Rozbor zadání

Cílem této práce je vytvoření počítačového programu pro zobrazování dat přicházejících z připojeného mikrokontroleru. Hlavním požadavkem je univerzálnost, tedy aby byl program do budoucna využitelný pro v podstatě jakýkoliv softwarově definovaný přístroj. Předpokládá se, že půjde především (ale ne výhradně) o osciloskopy či voltmetry na procesorech STM32. Takovéto přístroje budou vyvíjeny v rámci studia jakožto dostupnější (a pro potřeby měření jednoduchých obvodů dostačující) náhrada za profesionální přístroje. Pro přenos dat do programu je tedy nutné navrhnout takový komunikační protokol, který splní tyto požadavky:

- Bude vyhovovat potřebám mnoha typů přístrojů, zejména těchto: osciloskop, logický analyzátor, voltmetr (se záznamem), záznam neelektrických veličin
- Odeslání dat z mikrokontroleru musí být snadno implementovatelné a nemělo by vyžadovat nadbytečnou softwarovou režii (overhead) na straně mikrokontroleru.
- Přenos dat musí iniciovat mikrokontroler, nikoli PC aplikace. Komunikace směrem z PC do zařízení by měla zahrnovat jen příkazy které zadá sám uživatel.

Kromě zobrazování údajů z měřicích přístrojů má být program použitelný i pro jejich ovládání. Z toho důvodu je požadováno zahrnutí terminálu podporujícího ANSI sekvence. To uživateli umožní (prostřednictvím příkazů ze zařízení) vytvořit v tomto terminálu pseudografické rozhraní pro zobrazení měřených hodnot a nastavených parametrů zařízení. Důležitá je také možnost vyexportovat data do souboru ve vhodném formátu, aby mohla být zpracována v jiném programu či v případě využití ve výuce vložena do protokolu ze zpracované úlohy. Požadována je funkčnost na platformách Linux a Windows (včetně starších verzí, zejména Windows XP). Z tohoto důvodu se jako nejvýhodnější pro vytvoření grafického rozhraní zdá být multiplatformní framework Qt.

V zadání je také požadováno prověřit možnosti realizace osciloskopů a generátorů signálu s mikrokontrolery STM32. S ohledem na vývoj a testování aplikace je v podstatě nutností mít i přístroj s kterým ji budu používat. Pro účely testování zobrazovací aplikace by se nemuselo jednat o skutečný měřicí přístroj, použitelnost zobrazovacího programu lze prověřit i tím, že by mikrokontroler generoval simulovaná data. Rozhodně se však pokusím vytvořit funkční a prakticky použitelný osciloskop. Co se týče generátorů signálu, nepovažuji realizaci za potřebnou. V případě generátorů tkví úloha

zobrazovací aplikace pouze v odesílání příkazů pro nastavení a možnosti ovládání lze předvést i při nastavení parametrů osciloskopu. Je však vhodné do osciloskopu začlenit jednoduchý generátor signálu, jako je PWM s nastavitelnou frekvencí a střídou.

2.1 Stanovení cílů

Zobrazovací aplikace musí být snadno použitelná, s minimální nutností ručního nastavení před použitím. Ovládání by mělo být intuitivní, aby se dalo pochopit „za běhu“ bez potřeby rozsáhlého návodu k použití. Návod k použití však je potřeba vytvořit, a to především s ohledem na řádné zdokumentování komunikačního protokolu.

Terminál s podporou ANSI sekvencí musí podporovat sekvence užitečné pro tvorbu pseudo–grafického uživatelského rozhraní, jde zejména o pohyb textového kurzoru a vybarvení textu či jeho pozadí. Dále prověřím možnosti, jak terminál co nejlépe využít pro ovládání zařízení. Nutné minimum je možnost odeslat textový příkaz, pokusím se však vymyslet univerzální způsob ovládání kliknutím myší.

Požadavky na funkce zobrazovací aplikace nejsou velké (v podstatě jde o zobrazení hodnot v grafu a odečet hodnot pomocí kurzorů). Aby však aplikace byla skutečně užitečná, musí být minimálně na úrovni již existujících alternativ. Proto nad rámec zadání implementuji tyto funkce:

- Kurzory pro odečet hodnot ovladatelné pomocí myši
- Funkce „autoset“ pro snadné rozvržení více kanálu na obrazovku
- Matematické kanály (součet/rozdíl kanálů)
- XY režim
- Výpočet spektra signálu algoritmem FFT
- Automatické výpočty
- Interpolace signálu
- Průměrování měřených průběhů

Realizace osciloskopu s pomocí SM32 by měla sloužit nejen jako demonstrace použitelnosti zobrazovací aplikace, ale i jako skutečně použitelný přístroj. Pokusím se vytvořit takový přístroj, který může přímo konkurovat již existujícím platformám. Při tom se budu snažit efektivně využít periferie mikrokontroleru pro dosažení co nejlepších výsledků. Velmi užitečné by mohlo být využit AWDG (analog–watchdog) jako trigger.

Kapitola 3

Tvorba zobrazovací aplikace

K vývoji zobrazovací aplikace využiji framework Qt. Knihovny Qt umožňují vytvoření aplikace s grafickým uživatelským rozhraním. Pro programování s Qt lze využít programovací jazyk C++ nebo Python. V mém případě je jednoznačnou volbou C++ se kterým již mám zkušenosti ze studia (včetně použití Qt). K vývoji aplikací v Qt je k dispozici integrované vývojové prostředí (IDE) Qt Creator, které umožňuje návrh GUI v grafickém prostředí. Qt je k dispozici zdarma pro open-source projekty pod licencí LGPL.

3.1 Programování s Qt

Jazyk C++ je objektově orientovaný programovací jazyk. Knihovna Qt zavádí typ QObject, který (oproti klasickým C++ třídám) navíc zavádí signály a sloty. „Signály a sloty jsou používány pro komunikaci mezi objekty. Mechanismus signálů a slotů je centrální vlastnost Qt a pravděpodobně součást kterou se nejvíce odlišuje od jiných frameworků. Signály a sloty jsou umožněny meta-objekt systémem v Qt.“[2]

Program v Qt, stejně jako všechny C++ aplikace, začíná funkcí `main`. V této funkci je vytvořen objekt `QApplication` a hlavní okno aplikace. Běh aplikace je zahájen funkcí `app.exec()`, v které probíhá zpracování signálů a slotů po celou dobu běhu aplikace. Další funkce jsou již volány prostřednictvím signálů vyslaných z prvků GUI.

```
1 int main(int argc, char *argv[])
2 {
3     QApplication app(argc, argv);
4     MainWindow w;
5     w.show();
6     return app.exec();
7 }
```

3.1.1 Signály a sloty

Signály a sloty se deklarují podobně jako funkce. Slot je velmi podobný běžné funkci, jen má k modifikátoru viditelnosti (public, private) přidané klíčové slovo `slot`. Signál se pouze deklaruje, nemá definici. Pro vyslání signálu se použije klíčové slovo `emit` následované voláním funkce signálu. Aby mohl objekt využívat signály a sloty, musí být založen na třídě QObject a v `private` sekci deklarace musí mít makro `Q_OBJECT`. Argument signálu může být libovolný datový typ nebo objekt, ten však musí být registrován v třídě QMetaType [3]:

```
Q_DECLARE_METATYPE(MyObject); // Před funkcí main
qRegisterMetaType<MyObject>(); // Ve funkci main
```

Propojení signálu se slotem lze provést pomocí funkce `QObject::connect`, pro signály od prvků GUI lze navíc použít automatické propojení. Signál z objektu GUI se automaticky propojí s funkcí s názvem `void on_objectName_signalName(argument)` deklarovanou v objektu hlavního okna.

3.1.2 Vícevláknová aplikace

Je vhodné, aby zpracovaní příchozích dat probíhalo v jiném vláknu než GUI, aby nedošlo k narušení plynulosti ovládání v případě, že je zpracování dat časově náročné. Tvorba vícevláknové aplikace v Qt je poměrně snadná. Jak již bylo řečeno, objekty v Qt spolu komunikují prostřednictvím takzvaných signálů a slotů. Každý QObjekt (dále jen „objekt“) má svůj `event loop`: smyčku která kontroluje, jestli byl pomocí signálu zavolán některý se slotů objektu. Pokud je objekt přemístěn do jiného vlákna, jeho event loop a tedy i funkce slotů budou probíhat v tomto novém vláknu. V článku[4] je uveden postup vytvoření a zrušení objektu v jiném vláknu. Postup, který jsem použil vypadá takto:

```
1     // Začátek programu
2     QThread thread; // Vytvoření vlákna
3     MyObject *object = new MyObject; // Vytvoření instance objektu
4     object->moveToThread(&thread); // Přesun objektu do vlákna
5     thread.start(); // Spuštění vlákna (objekt je schopen přijímat signály)
6     // Běh programu
7     // Ukončení programu
8     object->deleteLater(); // Objekt bude smazán, jakmile to půjde.
9     thread.quit(); // Ukončí se event loop.
10    tread.wait(); // Počká na ukončení funkcí běžících ve vláknu.
```

Jsou dva typy signálů: `direct`, pro komunikaci mezi objekty ve stejném vláknu a `queued` pro objekty v různých vláknech. Signály `direct` se chovají v podstatě jako zavolání funkce. V případě `queued` signálů se signály řadí do fronty a funkce slotu bude vykonána až to bude možné. Pokud při propojení není specifikován typ, je zvolen automaticky. „Typ signálu je zvolen až při jeho vyslání, ne hned při propojení, proto nevadí, pokud je signál se slotem propojen ještě před přesunutím objektu do nového vlákna“[5]

3.2 Vykreslování grafu

Nejvýznamnější částí GUI je graf, do kterého budou přijatá data vykreslena. Pro vykreslování grafů je v Qt obsažena komponenta QChart, existují však i volně dostupné knihovny nabízející více funkcionalit. Rozhodl jsem se použít knihovnu QCustomPlot [6].

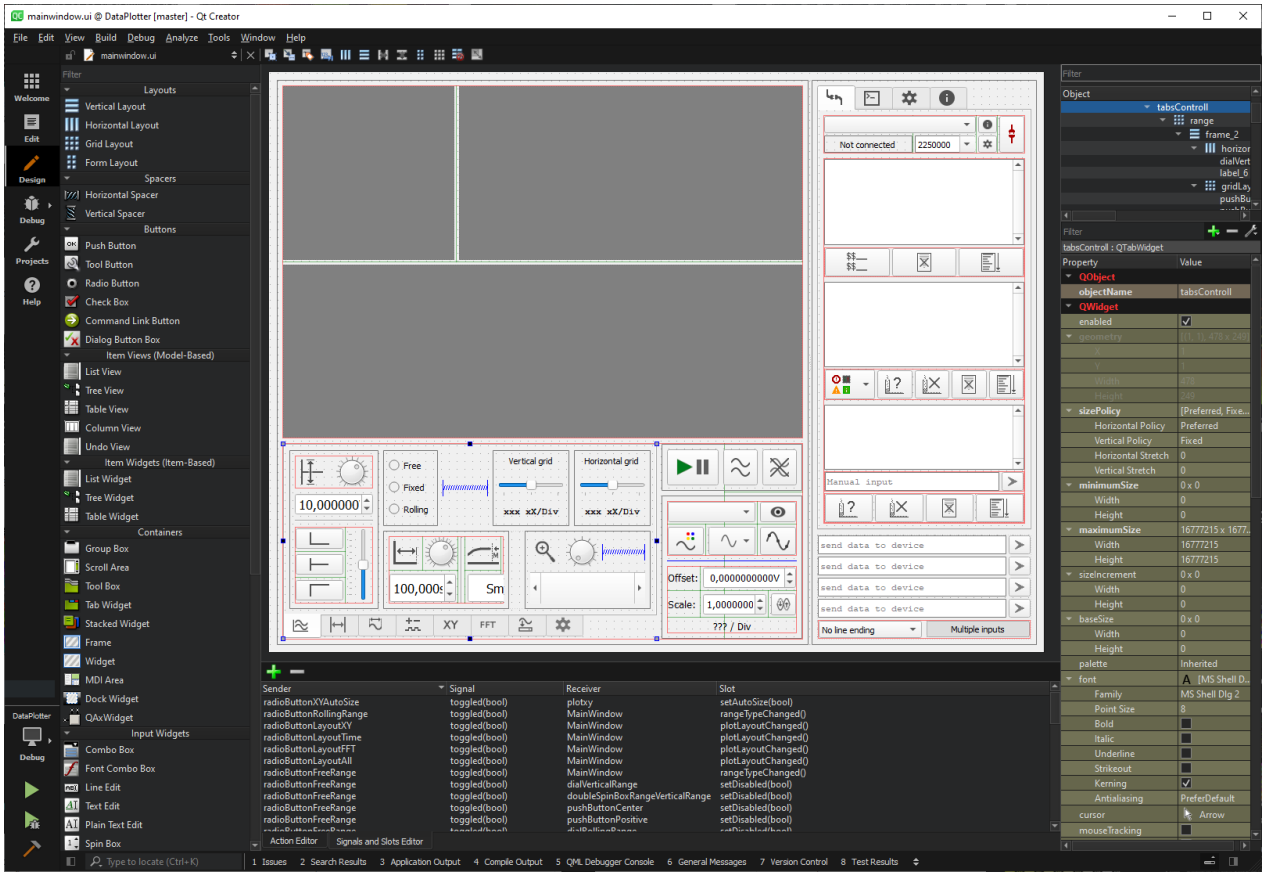
3.3 Terminál s podporou ANSI sekvencí

Další významnou částí projektu je vytvoření terminálu pro výpis textu. Jednoduché programy typu „serial monitor“ vypisují přijatý text jednoduše v pořadí, v jakém přišel a není možné nijak pohybovat textovým kurzorem a přepisovat stávající text. To je nepraktické pro zobrazení hodnot uživateli, protože nová hodnota by se musela vypisovat vždy na nový řádek. Lepším řešením je umožnit ovládání textového kursoru prostřednictvím příkazů obsažených ve vypisovaném textu, tak aby bylo možné přepsat libovolnou pozici v terminálu. To umožní zobrazit měnící-se hodnotu na jednom, stálém, místě.

Ovládání kursoru (a další funkce jako například změna barvy textu a pozadí) lze provést pomocí ANSI escape sekvencí. Tyto sekvence jsou přehledně popsány v článku [7]. Jako příklad aplikace, která podporuje ANSI escape sekvence lze uvést například PuTTY [8]. Pokusil jsem se najít již hotový prvek GUI pro Qt (takzvaný QWidget), který by toto umožňoval, ale žádný jsem nesehnal. Pokusím se ho tedy vytvořit sám.

3.4 Komunikace s přístrojem

Vývojové desky jako například Nucleo či Arduino jsou vybaveny USB-UART převodníkem. Některé mikrokontrolery kromě rozhraní UART mají i USB, které lze využít v režimu Virtual Com Port. V obou případech se zařízení v počítači chovají jako sériový port. Pro komunikaci prostřednictvím sériového portu je v frameworku Qt k dispozici knihovna QSerialPort.



Obrázek 3.1: Vývojové prostředí QT Creator

Kapitola 4

Grafické uživatelské rozhraní

Okno aplikace (zobrazeno na konci této kapitoly) jsem rozvrhl na tři základní části:

- V hlavní části je graf. Plocha grafu je buď zaplněna časovým grafem, nebo je rozdělena na tři části: Časový graf, XY graf a spektrum.
- V pravé části okna je panel s nastavením připojení, nastavením programu a terminálem.
- Ve spodní části je ovládání grafu a další funkce souvisejících se zpracováním dat.

Hranici mezi grafem a panelem lze popotahovat myší a tím zvětšit šířku terminálu na úkor grafu (vytvořeno s pomocí `QSplitter`).

Při návrhu GUI jsem se snažil (úspěšně) vyhovět požadavku na možnost zmenšit okno na velikost nižší než 1024×768 kvůli nízkému rozlišení dataprojektorů v některých učebnách.

4.1 Ikony

Ukázalo se výhodné pro tlačítka a podobné prvky použít ikony namísto textu. Ikony jsou výhodné z těchto důvodů:

- Ikony zabírají méně místa než text
- Správně navržená ikona má větší vypovídající hodnotu než stručný popisek
- Hezčí design

Pro dovysvětlení funkce tlačítka lze doplnit text pro takzvaný „tooltip“.

Ikony jsem kreslil sám (s výjimkou vlajek u výběru jazyka) v programu GIMP.

4.2 Ovládací prvky

Knihovny Qt nabízejí velké množství ovládacích prvků uživatelského rozhraní, samozřejmostí jsou tlačítka, textová pole, popisky, výběr ze seznamu („combo box“)...

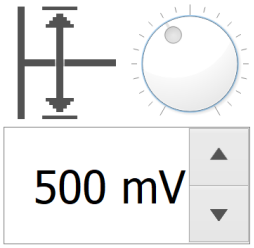
4.2.1 Nastavení číselných hodnot

Pro zadání číselné hodnoty je nejvhodnější prvek `QDoubleSpinbox`, ten umožňuje zadání desetinného čísla, které lze zvyšovat či snižovat tlačítky nebo kolečkem myši. Nastavení rozsahu grafu ale vyžaduje velký rozsah hodnot, nastavování v lineárním měřítku by bylo nepraktické. `QDoubleSpinbox` lze přepnout do „adaptivního“ režimu, kdy se krok změn hodnoty mění podle nastaveného čísla (vždy o řád nižší, než je číslo, například pokud je nastaveno 3.3, bude krok 0.1). Tento způsob je vhodný pro precizní nastavení hodnoty, ale pro pohotové měnění hodnoty ve větším rozsahu je stále příliš pomalý.

U mnoha přístrojů (i v jiných oblastech života, jako např. hodnoty mincí a bankovek) se setkáváme s pseudo-logaritmickou řadou `1 2 5 10 20 50...`. Ta je výhodná i pro tento účel. Ke každému `QDoubleSpinBox` nastavujícímu velký dynamický rozsah hodnot jsem přidal prvek `QDial` („kolečko“), pomocí kterého přepínám hodnotu podle zmíněné řady. `QDial` se chová jako nastavení celého čísla u určitém rozsahu, ovládat ho lze kolečkem myši (najet na něj ukazatelem a točit). Toto číslo poté použiji jako index hodnoty v řadě

```
1e-9, 2e-9, 5e-9, 1e-8, 2e-8, 5e-8, ... 1e8, 2e8, 5e8, 1e9, 2e9, 5e9, 1e10
```

Ta je po změně pozice `QDial` nastavena do `QDoubleSpinBox`. Hodnotu v `QDoubleSpinBox` lze dále detailněji měnit, čímž se „desynchronizuje“ s `QDial`. Pokud je později změněna pozice `QDial`, je hodnota nastavena na hodnotu z řady `1 2 5 ...` která je neblíže původní hodnotě.



Obrázek 4.1: QDial a upravený QDoubleSpinBox

Také jsem upravil prvek `QDoubleSpinBox` tak, aby namísto desetinných čísel používal jednotky s předponami (mili, mikro ...). Vnitřní chování upraveného `MyDoubleSpinBoxWithUnits` je stejné jako u původního, pouze jsou změněny funkce `QString textFromValue(double val)` pro převod nastavené hodnoty na zobrazené číslo a `double valueFromText(const QString& text)` pro převedení ručně zadaného čísla na hodnotu.

4.3 Hodnoty v jednotkách s předponami

Předpokládá se, že čísla vložená do grafu jsou vždy v základních jednotkách (více k tomuto v kapitole 5.2.3). Pro přehledné zobrazení je však vhodné použít jednotky s předponou (například u grafu z osciloskopu budou časy v řádech μs).

Pro převod čísla na hodnotu v jednotkách jsem vytvořil vlastní funkci. Použitá předpona jednotky je určena na základě řádu[1] hodnoty. Například pokud je hodnota mezi 0.001 a 1, bude jednotkou milivolt. Hodnota je vynásobena (případně vydělena) pro převedení na danou jednotku. Poté je zaokrouhlena na daný počet platných cifer a převedena na text, za který je následně přidáno písmeno odpovídající předponě jednotky a jednotka.

4.3.1 Zaokrouhlení na počet platných cifer

Standardní funkce převodu čísla na text `QString::number(double)` umožňuje pouze zadání pevného počtu desetinných míst. Pro zaokrouhlení na daný počet platných cifer jsem vytvořil vlastní funkci.

Základní myšlenkou je vynásobení čísla mocninou deseti tak, aby se příslušný počet cifer posunul před desetinnou tečku. Následuje zaokrouhlení na celé číslo, převedení na text a nakonec vložení desetinné tečky na příslušné místo.

```
1  QString toSignificantDigits(double x, double prec) {
2      // x: hodnota, která má být převedena na text
3      // prec: počet platných cifer
4      if (x == 0) return "0";
5      int log10ofX = intLog10(x * 1.0000001);
6
7      QString result;
8      if (log10ofX >= prec - 1) {  // číslo je celé, nemá desetinnou část
9          result = QString::number((int)round(x));
10     } else {
11         // Převedu na text jako celé číslo
12         result = QString::number((int)round(x * (pow10(prec - log10ofX - 1))));
13
14         // Na příslušné místo vloží desetinnou tečku
15         int decimalPoint = result.length() - prec + log10ofX + 1;
16         if (decimalPoint > 0) {
17             result.insert(decimalPoint, '.');
18         } else {
19             // Pokud číslo nemá celou část, je před něj přidána nula s desetinou tečkou
20             // Případně další nuly za tečkou
21             for (; decimalPoint < 0; decimalPoint++)
22                 result.push_front('0');
23             result.push_front('.');
24             result.push_front('0');
25         }
26     }
27     return result;
28 }
```

[1] Tím myslím nejbližší nižší mocninu deseti, například číslo 333 má řád 10^2 (stovky)

Funkce `intLog10` vypočítá desítkový logaritmus a zaokrouhlí ho dolů. Vlivem nepřesnosti v typu `double` může hodnota x být trochu menší, než měla být, například `1.000000` se může změnit na `0.99999999`, proto je číslo mírně zvětšeno, aby se zajistilo, že řád nevyjde o jedna menší, než měl být.

4.3.2 Nastavení jednotky

Předpokládá se, že hodnoty na vodorovné ose představují čas v sekundách a svislá osa je napětí. To však není zcela v souladu s požadavkem na univerzálnost. Proto jsem přidal možnost jednotku změnit (jednoduše se mění text který je přidán za číslo s předponou jednotky).

Pro některé jednotky použití předpony není vhodné, například pro decibely: „milidecibel“ by nedával smysl. Stejně tak v případě že uživatel ručně zadá jednotku, která již předponu obsahuje. Proto tyto případy je použit klasický převod čísla na text. Zadání jednotky s předponou však není doporučeno. Předpokládá se, že čísla vložená do grafu jsou vždy v základních jednotkách. Možnost posílání hodnot v například milivoltech je dále rozvedena v kapitole 5.2.3.

4.4 Vícejazyčné uživatelské rozhraní

Framework Qt umožňuje poměrně snadno vytvořit překlad GUI do různých jazyků. Rozhodl jsem se GUI vytvořit v angličtině a češtině. Pomocí příkazu `lupdate` se vygeneruje soubor `.ts` obsahující texty z GUI i textové řetězce z kódu uvozené funkcí `tr("text")`. Tento soubor poté načtu programem Qt Linguist v kterém texty přeložím. Příkazem `lrelease` je z `.ts` souboru vytvořen soubor `.qm`, který lze poté načíst do objektu `QTranslator` zajišťujícího překlad textů GUI.

```
1 translator->load(":/translations/translation_cz.qm");
2 qApp->installTranslator(translator);
3 ui->retranslateUi(this);
```

4.5 Načítání souborů a jejich přiložení k programu

Aplikace k běhu potřebuje soubory jako například překlad a ikony. Tyto soubory by bylo možné přidat do adresáře programu, ale výhodnější je použít systém „resources“ v Qt. Jedním ze zdrojových souborů programu je soubor `Resource Collection Files (.qrc)`, obsahující seznam souborů, které mají být k aplikaci přidány. Pří kompilaci jsou tyto soubory přibaleny do `exe` souboru a v kódu se k nim přistupuje pomocí cesty začínající dvojtečkou, například `:/translations/translation_cz.qm`.

Soubory, které má být možně upravit (například soubor s nastavením) je nutné umístit do adresáře programu. V aplikaci také nabízím možnost kliknutím na tlačítko otevřít uživatelskou příručku. Příručka je ve formátu PDF a má být otevřena v externím prohlížeči (což znamená, že musí existovat jako samostatný soubor, ne v `resources`). To lze zařídit funkcí `QDesktopServices::openUrl()`.

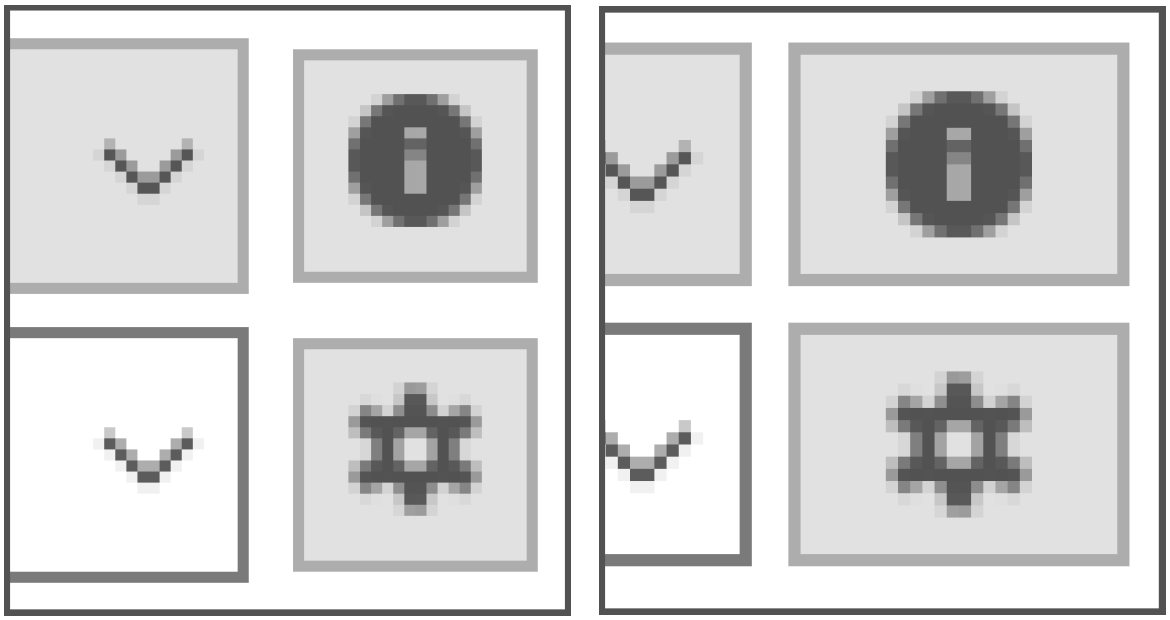
4.6 Nekonzistence ve velikosti prvků GUI

Povšiml jsem si, že tlačítka `QPushButton` se zobrazují menší než ostatní prvky GUI o jeden pixel z každé strany. To jsem vyřešil tím, že velikost prvku změním pomocí `stylesheet`[2].

```
QPushButton {
    margin:-1px;
    padding-left: 6px;
    padding-right: 6px;
    padding-top: 4px;
    padding-bottom: 4px;
}
```

Problém se vyskytuje pouze na Windows 10, u jiných verzí tato úprava není použita. Verzi Windows lze po spuštění programu zjistit pomocí

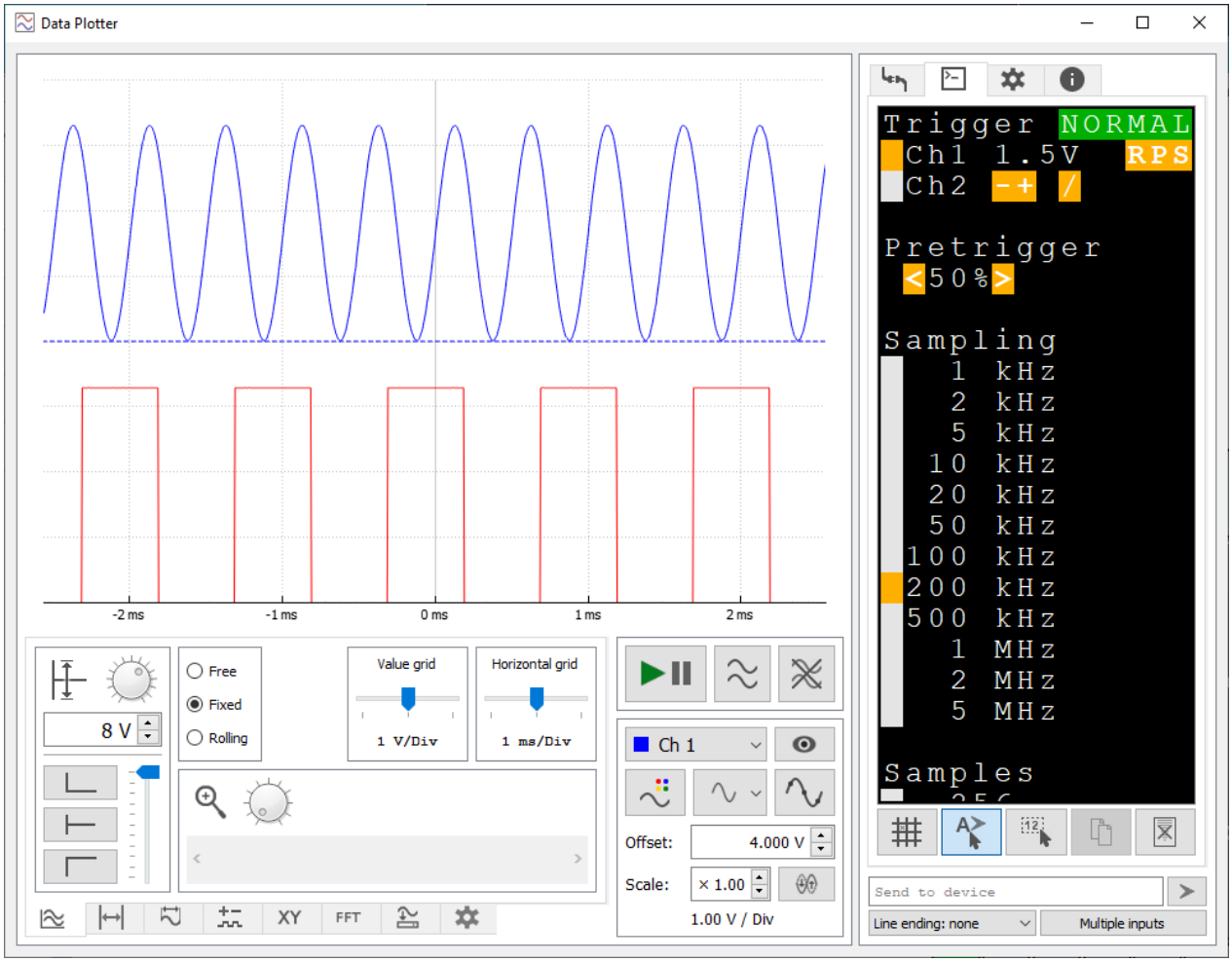
```
    if (QSysInfo::productVersion() == "10")
        // Windows 10
```



(a) : Výchozí nastavení **(b) :** Po úpravě

Obrázek 4.3: Zobrazení tlačítka v porovnání s jinými prvky GUI

[2]Stylesheet je kód (v podobě textového řetězce) umožňující upravit vzhled prvků GUI [9].



Obrázek 4.2: Okno aplikace

Kapitola 5

Komunikační protokol

Komunikační protokol určuje, v jakém formátu jsou do aplikace posílána data ze zařízení. Pro komunikaci se předpokládá použití UART (s UART-USB převodníkem), který posílá data po jednotlivých bajtech. Začátek a konec zprávy musí být označen sekvencí znaků (bajtů), která nemůže být zaměněna s daty obsaženými v těle zprávy. Zprávy, které by tato aplikace měla být schopná zpracovat, lze rozdělit na tyto základní typy:

- Hodnoty, které mají být vykresleny do grafu
- Text k vypsání do terminálu
- Příkazy k nastavení grafu
- Informační zprávy, které mají být zobrazeny uživateli

Posílání čísel by bylo nejsnadněji realizovatelné převodem čísla na text (například funkce `printf`) a odesláním v této textové formě. Vhodnějším způsobem je však posílání hodnot v binárním formátu (po bajtech, tak jak jsou reprezentovány ve svém datovém typu). To umožní rychlejší přenos a zpracování.

5.1 Binární reprezentace dat

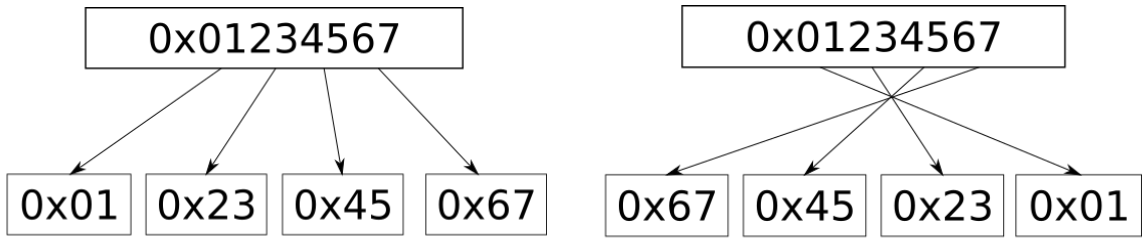
Programovací jazyky rozlišují tyto tři základní číselné datové typy:

- Unsigned integer: kladné celé číslo. Zapsané je v podstatě jako číslo v dvojkové soustavě (i když pořadí bitů se může lišit, viz obrázek 5.1). Počet bajtů se pro různé platformy liší, na PC je obvykle 4. V případě mikrokontrolerů může být menší a obvykle je přímo specifikován (např. `uint16_t` je 16bitový).
- Signed Integer: celé číslo se znaménkem. Podobné jako předchozí případ, ale záporné číslo je zapsáno formou dvojkového doplňku.

- Floating point: desetinné číslo v plovoucí řádové čárce (vědecká notace). Umožní zapsat prakticky libovolné číslo ve velkém rozsahu za cenu nepřesnosti v zaokrouhlení. V případě mikrokontrolerů procesor často není vybaven jednotkou FPU pro hardwarové operace s těmito čísly, takže použití je velmi neefektivní.

5.1.1 Big-endian a Little-endian

Jako přirozené pořadí bajtů se jeví začít bitem nejvyššího řádu (most significant bit, MSB) a skončit nejnižším řádem (least significant bit, LSB), jako bychom psali číslo v dvojkové soustavě (z leva doprava). Z hlediska výpočtů v procesoru je však o něco výhodnější pořadí bajtů prohodit. Většina platforem tedy používá little-endian.



(a) : Big-endian **(b)** : Little-endian

Obrázek 5.1: Reprezentace datových typů

5.2 Verze komunikačního protokolu

Komunikační protokol jsem během vývoje aplikace několikrát přepracoval. Stručně zde popíšu i předchozí, zavržené, verze a důvody proč nebyly použitelné.

5.2.1 První verze komunikačního protokolu

První verze protokolu byla udělána tak, že každá zpráva začínala znakem `$` po kterém následovalo záhlaví obsahující typ zprávy a případně další parametry. Záhlaví je zakončeno dvojtečkou, poté následují data. Zpráva je zakončena znakem `*`.

$$\$typ, parametry : data*$$

Program čekal až v bufferu bude celý takto ohraničený úsek a ten poté dále zpracoval.

Zásadní nevýhodou této verze bylo, že je použitelná jen pro čísla v textové podobě, ne pro binární hodnoty. Při přenosu binárních dat se v posílaných číslech běžně vyskytují všechny možné znaky ASCII, a tedy dochází k předčasnému ukončení kdykoli se vyskytne znak $*$.

Další nevýhodou je že přicházející data nejsou zpracovávána průběžně, ale až po přijetí celé zprávy, pokud by uživatel například testoval výpis do terminálu po jednotlivých znacích, musel by každý znak být poslán jako nová zpráva.

5.2.2 Druhá verze komunikačního protokolu

Druhá verze protokolu se snažila řešit některé výše uvedené nedostatky první verze: kompatibilitu s přenosem binárních dat a průběžné zpracování. Zprávy jsou rozděleny na dva základní typy: příkaz pro nastavení typu příchozích dat (nahrazuje záhlaví) a samotná data. Příkaz má tvar:

$$< \mathbf{cmd} > mode, parameters < \mathbf{end} >$$

Pokud je v datech přijatých ze sériového portu nalezen takový úsek, je zpracován jako záhlaví nové zprávy, které definuje režim v jakém jsou následující data zpracována. Vše ostatní je zpracováno jako data podle aktuálně zvoleného režimu.

Úsek dat je zakončen:

- Výrazem `<end>`
- Začátkem dalšího příkazu `<cmd>`
- Automaticky, pokud po nastavené době nepřijde nic nového.

Tato verze stále má nedostatky: spoléhá na zakončení úseku dat sekvencí znaků `<end>`, ta se však stálé může v binárních číslech objevit náhodně[1]. Zakončení dat časovou prodlevou se ukázalo být velmi nespolehlivé.

5.2.3 Třetí, konečná, verze komunikačního protokolu

Tato verze zpracovává data průběžně ihned po přijetí, tedy nečeká na zakončení zprávy. Zpráva začíná dvojicí znaků `$$` po níž následuje jedno písmeno označující typ zprávy. Další zpracování se řídí tímto typem. Binární hodnoty jsou započaty kódem označujícím datový typ, viz tabulka 5.1.

Typy zpráv jsou popsány v následující kapitole.

Zpracování čísel

Pokud program očekává číselnou hodnotu, nejprve přečte první znak této hodnoty. Pokud se jedná o číslici nebo znaménko mínus, jde o číslo zapsané textově[2]. Jinak jde o binární hodnotu, v tom

[1] Při použití desky Nucleo s 12bitovým AD převodníkem toto nastat nemůže, protože když je 12bitové číslo zapsáno dvěma bajty, tak první bajt má hodnotu maximálně $2^4 - 1 = 15$, což je méně než všechna čísla odpovídající znakům použitým v ukončovací sekvenci, tyto znaky se tedy nikdy neobjeví vedle sebe.

[2] V textovém zápisu lze použít i vědeckou notaci (např. `15e-6`).

případě je přečten typ (který obsahuje údaj o počtu bajtů hodnoty) a přečten příslušný počet bajtů následujících za označením typu. Tím je zajištěno, že i kdyby bajty čísla obsahovali sekvenci `$$`, nebude zaměněna za začátek nové zprávy.

Binární data

Převod sekvence bajtů na číslo je poměrně jednoduchý, stačí pole znaků přečíst jako hodnotu příslušného typu. Hodnoty se do grafu ve výsledku vždy přidávají v typu double.

```
return (double) *((uint16_t*)bytes);
```

Může nastat situace, že uživatel bude chtít poslat údaj o napětí v podobě celého čísla, ale v jednotkách mV. To by se dalo řešit změnou jednotky na ose grafu, ale přidal jsem možnost zadání jednotky přímo do označení typu. Před první písmeno typu lze napsat znak označující předponu jednotky. Možnosti jsou: `T, G, M, k, h, D, d, c, m, u, p, f, a`. Například `mU2` je 16bitový `unsigned integer` v milivoltech (hodnota bude vydělena tisícem).

Tabulka 5.1: Označení datových typů

little-endian		big-endian	
u1	8bitový unsigned integer	U1	8bitový unsigned integer
u2	16bitový unsigned integer	U2	16bitový unsigned integer
u3	24bitový unsigned integer	U3	24bitový unsigned integer
u4	32bitový unsigned integer	U4	32bitový unsigned integer
i1	8bitový signed integer	I1	8bitový signed integer
i2	16bitový signed integer	I2	16bitový signed integer
i4	32bitový signed integer	I4	32bitový signed integer
f4	float	F4	float
f8	double	F8	double

Řešení syntaktických chyb

Syntaktické chyby v přijatých datech by samozřejmě neměli způsobit selhání aplikace. Zároveň je vhodné uživateli poskytnout informaci o tom k jaké chybě došlo, aby bylo možné ji odhalit a opravit.

Pokud je nalezen problém (např. neplatné označení typu, nebo neplatné číslo), najde program nejbližší následující `$$` a zahodí všechna data před ním. Poté pokračuje od začátku nové zprávy[3]. Při chybě je v programu zobrazena informace, že k chybě došlo.

[3]Není zaručeno, že nalezená sekvence `$$` je začátek nové zprávy, mohla se náhodně vyskytnout v binární hodnotě. To by ale pouze vedlo na další chybu, po které by byl znovu hledán začátek.

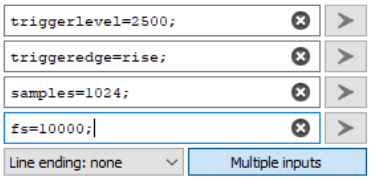
5.3 Odesílání příkazů do zařízení

Kromě příjmu dat ze zařízení je také potřeba jej ovládat. Základním způsobem ovládání je napsání textu do textového pole a jeho odeslání. To lze dále vylepšit tím, že je v programu připraveno několik takovýchto textových polí (obrázek 5.2), takže uživatel může mít předpřipraveno několik příkazů a snadno je upravovat a odeslat. Pohodlnější možností ovládání je vytvoření interaktivního menu v terminálu, viz kapitola 12.2.

Lze předpokládat, že zařízení po spuštění pošle nějaké příkazy pro nastavení a výpis do terminálu (protože by bylo nepraktické pravidelně přepisovat celý terminál). K připojení zařízení k aplikaci však typycky dojde až po jeho zapnutí a zprávy které odeslalo ihned po spuštění nebudou přijaty. To lze řešit několika způsoby:

- Po připojení mikrokontroler resetovat pomocí tlačítka.
- Mikrokontroler může ověřovat připojení pomocí zpráv `echo` (`$$E`, viz další kapitola) a prvotní nastavení odeslat až po potvrzení připojení.
- Poslat do mikrokontroleru příkaz, aby poslal prvotní nastavení.

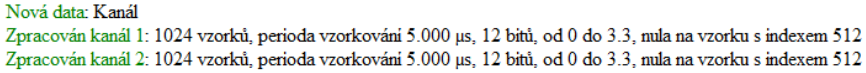
Poslední z možností lze provést ručním zadáním příkazu, zavedl jsem však i funkci která umožní v programu předem zadat takovýto „resetovací“ příkaz který bude po připojení odeslán automaticky.



Obrázek 5.2: Textová pole pro odeslání příkazů

5.4 Výpis přijatých dat

Pro případné řešení problémů s komunikací je vhodné mít možnost zobrazit přijatá data. Jako nejjednodušší možnost se jeví prosté vypsání všech přijatých znaků do textového pole (takzvaný „serial monitor“). To však naráží na zásadní problém, výpis tak velkého množství textu je velmi výpočetně náročný a je v podstatě nemožné ho vypisovat tak rychle jak přichází. Proto jsem zavrhl variantu vypisování všech přijatých dat a vypisuji pouze základní informace o typu přijatých dat a nikoli celou zprávu.



Obrázek 5.3: Příklad výpisu informací o přijatých datech

Režim vypisování informací lze navíc změnit tak, aby se zobrazovali pouze zprávy o problémech.

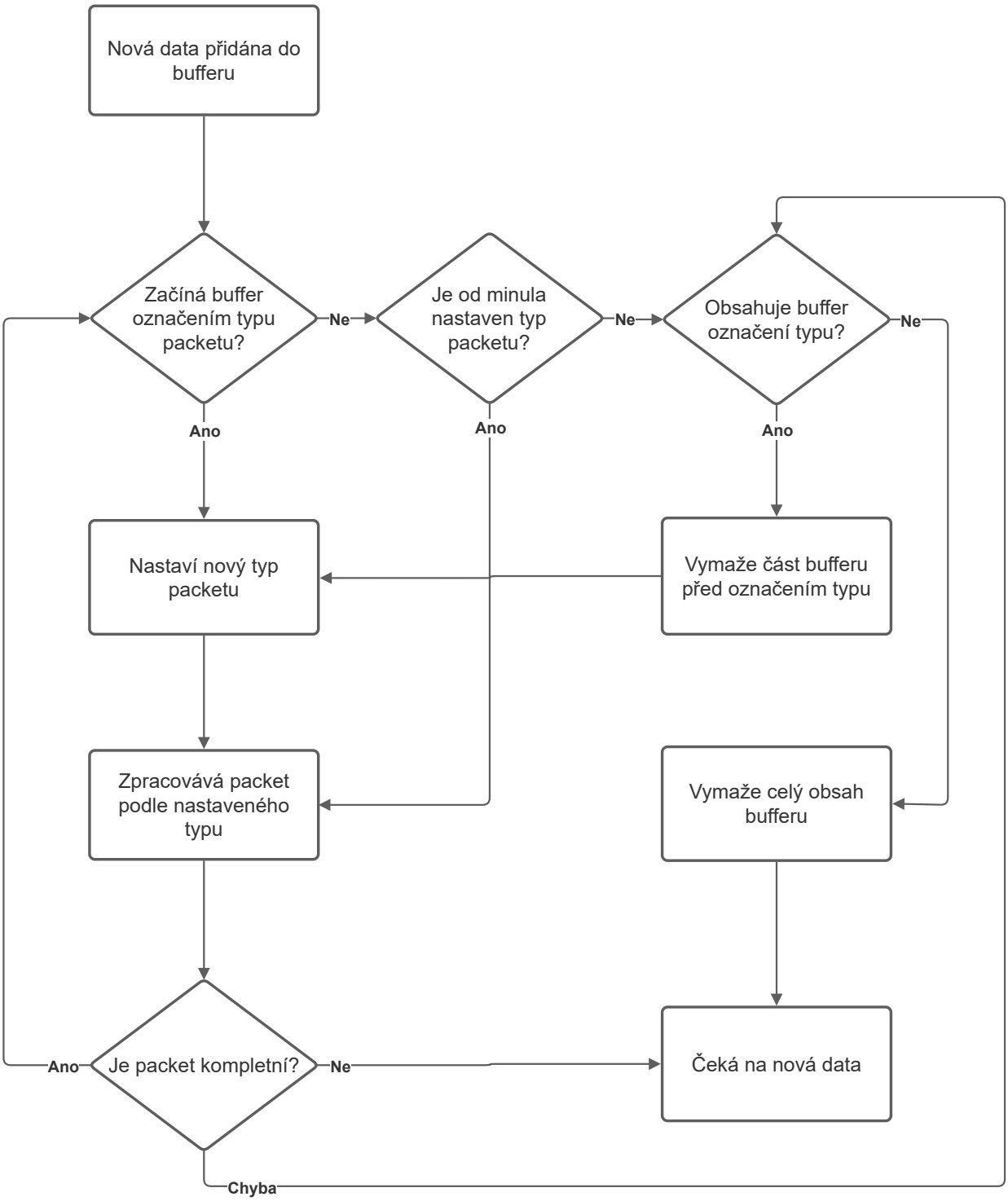
Přidal jsem i funkci zmíněného „serial monitoru“ kdy jsou vypsány všechny znaky tak, jak byly přijaty. Tuto funkci však nedoporučuji používat po delší dobu v případě velkého množství dat. Funkčnost jsem zlepšil tím, že k přidání textu nedochází ihned po přijetí, ale text se ukládá v bufferu a do textového pole se vkládá jen několikrát za sekundu po větších úsecích.

5.4.1 Textová pole v Qt

V Qt jsou dva druhy textových polí: `QPlainTextEdit` pro zobrazení prostého textu a `QTextEdit` pro formátovaný text. Jelikož jsem pro přehlednost chtěl mít možnost text vybarvit, použil jsem nejprve `QTextEdit`. Přidávání textu do něj je však velmi pomalé a docházelo k problémům i při zestručněném výpisu. Následně jsem zjistil, že `QPlainTextEdit` (navzdory svému názvu) podporuje barevný text (pomocí funkce `appendHtml(const QString)`) a přidávání textu do nej je výrazně rychlejší než v případě `QTextEdit`.

V případě „serial monitoru“ je situace ještě trochu složitější, protože funkce `append` přidá text na nový řádek. Pokud chci text přidat bez odřádkování, je nutné ho vložit pomocí textového kurzoru.

```
1 auto cursor = ui->plainTextEdit->textCursor();
2 cursor.movePosition(QTextCursor::End);
3 cursor.insertText(text);
```



Obrázek 5.4: Postup zpracování příchozích dat

Kapitola 6

Typy zpráv v komunikačním protokolu

V této kapitole stručně popíšu typy zpráv s ohledem na důvody zavedení a postup zpracování. Přehlednější popis použití i s příklady je součástí uživatelské příručky (příloha B).

Program rozlišuje tyto typy zpráv:

- `$$P` Přidání bodu do grafu
- `$$C` Přidání celého kanálu do grafu
- `$$B` Přidání bodu do logického kanálu
- `$$L` Přidání celého logického kanálu
- `$$T` Výpis do terminálu
- `$$S` Nastavení
- `$$I` Informační zpráva
- `$$W` Varovná zpráva
- `$$X` Chybová zpráva
- `$$E` Echo
- `$$U` Neznámý (data jsou zahozena - nezpracují se, použito jako výchozí typ po připojení)

6.1 Přidání bodů do grafu

Nejjednodušším způsobem přidání hodnot do grafu je přidání jednoho bodu. To je vhodné pro pomaloběžné průběhy, například voltmetr se záznamem, záznam teploty a podobně.

6.1.1 Formát zprávy typu bod

Zpráva má tvar:

$$\$\$\mathbf{P}t, ch_1, ch_2, ch_3;$$

- t: čas (souřadnice na vodorovné ose)
- ch_n: hodnota kanálu n v tomto čase (souřadnice na svislé ose)

Hodnoty lze zapsat číselně nebo v binárním tvaru. Kanálů může být maximálně 16, kterýkoli z kanálů lze vynechat tím, že se na jeho pozici napíše `-`.

Pokud jsou čísla zapsaná textově, musí být oddělena čárkami. U binárních hodnot je oddělení čárkami volitelné. Číselné a binární hodnoty lze kombinovat, v takovém případě je nutné čárku napsat alespoň za hodnoty napsané číselně.

Lze napojit více bodů za sebe, aniž by bylo nutné znovu psát `$$P`.

Speciální hodnoty v pozici pro čas

Do pozice pro čas lze kromě hodnoty zapsat i speciální příkazy:

- `-` Čas je o 1 vyšší než v předchozím bodě. Souřadnice na vodorovné ose tedy odpovídá indexu bodu.
- `-auto` Bude automaticky doplněn čas od připojení zařízení (měřený zobrazovacím programem)
- `-tod` Bude automaticky doplněn čas dne (v sekundách od půlnoci)

6.1.2 Zpracování zprávy typu bod

Nejprve je načteno první číslo postupem uvedeným v části 5.2.3 na straně 17. Mohou nastat dvě možnosti: pokud je číslo zapsané binárně, je načteno a jeho konec je pevně stanoven počtem bajtů. Pokud jde o číslo zapsané textově, je nutné nalézt jeho konec. To může být:

- Čárka, po které následuje další hodnota
- Středník, kterým bod končí
- Sekvence `$$`, (začátek další zprávy v případě chybějícího středníku na konci této)

Nejbližší z těchto znaků je vybrán a úsek před ním převeden na číslo. Případně jde o speciální příkaz (proto začíná pomlčkou/mínusem, aby byl dle prvního znaku považován za textové číslo).

Schematicky je postup znázorněn na obrázku 6.1.

6.2 Přidání bodu do grafu pro logický kanál

Kromě analogových hodnot je možné přidat i hodnoty logického analyzátoru.

$$\$\$\mathbf{B}t, value, bits;$$

- t: čas, stejně jako u analogového bodu
- $value$: hodnota typu unsigned integer (maximálně 32 bitů)
- $bits$: počet využitých bitů v čísle (od LSB). Pokud je vynecháno, je zobrazený počet bitů určen velikostí datového typu.

6.3 Přidání kanálu do grafu

Typ zprávy vhodný pro osciloskopy, data jsou poslána jako po sobě jdoucí vzorky v binárním tvaru, které nejsou nijak odděleny. Lze tedy do sériového portu zapsat přímo pole hodnot (buffer osciloskopu).

6.3.1 Základní formát

Hodnoty jsou vždy v binárním tvaru. Před samotnými daty je záhlaví s údaji pro zpracování. Zpracování záhlaví se řídí stejnými pravidly jako zpracování bodu (6.1), údaje v záhlaví lze zadat formou napsaného čísla, nebo binárně.

$$\$\$\mathbf{C}ch, T, N; type\square\square\square\square\square\square\square\square\square\square\square\square\square\square\square\square;$$

- ch: číslo kanálu do kterého budou hodnoty zapsány
- T: časový interval mezi dvěma po sobě jdoucími hodnotami ($\frac{1}{f_s}$)
- N: počet hodnot (tedy čísel, ne bajtů) v této zprávě
- $type$: datový typ hodnot (viz tabulka 5.1 na straně 18)

Zpráva je vždy zakončena středníkem. Použití středníku zde není vyloženě nutné (konec zprávy už je jednoznačně určen počtem bajtů které mají následovat po záhlaví), ale slouží jako kontrola, že uživatel správně zadal počet hodnot. Pokud na konci není středník, je uživateli zobrazeno varování.

6.3.2 Přidání kanálu do grafu s přemapováním hodnot

Hodnota z AD převodníku je vždy celé číslo o daném počtu bitů (rozlišení převodníku). Tuto hodnotu je následně potřeba přepočítat na napětí. Například mikrokontroler STM32 má 12bitoví ADC, kde hodnoty v rozsahu 0 až 4095 je třeba přepočítat na rozsah 0 až 3.3 voltů. Takový přepočet lze udělat přímo v mikrokontroleru, ale je výhodné hodnotu odeslat v původní podobě a přepočítat až v zobrazovacím programu. Tento typ záhlaví je dostupný jen pro hodnoty typu `unsigned integer`.

$$\textbf{\$\$C}ch, T, N, n, min, max; type\square\square\square\square\square\square\square\square\square\square\square\square\square\square\square\square;$$

- *ch*: číslo kanálu do kterého budou hodnoty zapsány
- *T*: časový interval mezi dvěma po sobě jdoucími hodnotami
- *N*: počet hodnot (tedy čísel, ne bajtů) v této zprávě
- *type*: datový typ hodnot (viz tabulka 5.1 na straně 18), pouze typ `unsigned integer`.
- *n*: počet bitů (využito pro přepočet)
- *min*: minimální skutečná hodnota (V_{ref-}), lze vynechat[1] - implicitně 0
- *max*: maximální skutečná hodnota (V_{ref+})

Přepočet probíhá podle rovnice

$$x = \frac{max - min}{2^n} \cdot x_{\text{raw}} + min, \tag{6.1}$$

kde x je výsledná hodnota zobrazená v grafu a x_{raw} je hodnota přijatá ze zařízení (proměnné n, max a min jsou definovány v seznamu výše).

6.3.3 Pozice triggeru

Digitální osciloskopy obvykle mají funkci pretrigger, kdy je zaznamenána i část průběhu před triggerem. Je vhodné nějak zobrazit kde k triggeru došlo tím, že příslušný vzorek umístím na čas 0. Proto jsem do záhlaví přidal ještě údaj *zeroIndex* o indexu vzorku (počítáno od 0), který má být umístěn na čase 0:

$$\textbf{\$\$C}ch, T, N, n, min, max, zeroIndex; type\square\square\square\square\square\square\square\square\square\square\square\square\square\square\square\square;$$

Údaj lze vynechat (první vzorek je v čase 0).

[1] Vynecháním se v tomto případě myslí, že záhlaví bude kratší o jeden údaj, ne vynechání napsáním pomlčky jako v případě hodnot bodu.

6.3.4 Více kanálů na přeskáčku

Pokud je v mikrokontroleru čteno více kanálů na jednom AD převodníku a hodnoty jsou ukládány do bufferu pomocí DMA, jsou v bufferu hodnoty uloženy na přeskáčku. Příklad:

```
ch1 ch2 ch3 ch4 ch1 ch2 ch3 ch4 ch1 ch2 ch3 ch4 ch1 ch2 ch3 ch4 ch1 ch2 ch3 ch4
```

Rozdělení tohoto na čtyři samostatné kanály není triviální úkol. V mikrokontroleru totiž buffer zabírá většinu paměti RAM, a tedy není možné hodnoty z něj kopírovat do nových bufferů, protože by se do paměti nevešly. Zpřeházení členů na místě lze provést, v podstatě se jedná o transpozici matice [10]

$$[1\ 2\ 3\ 4\ 1\ 2\ 3\ 4\ 1\ 2\ 3\ 4] \Longrightarrow \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{bmatrix} \Longrightarrow \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ 4 & 4 & 4 \end{bmatrix} \Longrightarrow [1\ 1\ 1\ 2\ 2\ 2\ 3\ 3\ 3\ 4\ 4\ 4]$$

na což určitě existují efektivní algoritmy, ale je jednodušší buffer odeslat ve stavu v jakém je a zpracovat až v počítačové aplikaci.

Takto na přeskáčku seřazené hodnoty několika kanálů lze poslat stejným formátem jako jeden kanál, jen se namísto čísla kanálu zadají čísla obsažených kanálů oddělené znaménkem plus (např: `1+2+3+4`).

6.3.5 Zpracování zprávy typu kanál

Záhlaví (od začátku po středník) je zpracováno stejným postupem jako bod. Poté je načten typ binárních dat. Počet bajtů za označením typu je roven počtu hodnot (údaj v záhlaví) vynásobenému počtem bajtů na jednu hodnotu (v označení datového typu). Program čeká, dokud v bufferu není počet bajtů vyšší než délka dat a poté je přečte. Dále je zkontrolováno, jestli jsou zakončena středníkem. Pokud je ve zprávě více kanálů (typ `1+2+3+4`), jsou data rozdělena na jednotlivé kanály.

6.4 Přidání logického kanálu do grafu

Logické kanály se přidávají podobně jako analogové.

$$L*T*, *N*, *bits*, *zeroIndex*; *type*□□□□□□□□□□□□□□□□;

- *T*: časový interval mezi dvěma po sobě jdoucími hodnotami (převrácená hodnota vzorkovací frekvence)
- *N*: počet hodnot (tedy čísel, ne bajtů) v této zprávě
- *bits*: specifikuje počet využitých bitů v čísle (od LSB). Pokud je vynecháno, je zobrazený počet bitů určen velikostí datového typu.

- *zeroIndex*: indexu vzorku (počítáno od 0), který má být umístěn na čase 0. Údaj lze vynechat (v tom případě je první vzorek v čase 0).
- *type*: datový typ hodnot (pouze typ `unsigned integer`)

6.5 Výpis do terminálu

Data poslaná touto zprávou se vypíší do terminálu.

$$T*text*

Zpráva není nijak zakončena, příchozí znaky se vypisují do terminálu, dokud nezačne nová zpráva jiného typu. Zpráva není zakončena středníkem, výpis textu je průběžný a končí začátkem jiné zprávy. Text vypsaný do terminálu může obsahovat jakékoli znaky s výjimkou dvou po sobě jdoucích znaků dolar `$$` (sekvence pro začátek nové zprávy). Terminál podporuje ANSI escape sekvence.

6.6 Zpráva pro uživatele

Text poslaný tímto typem zprávy bude zobrazen uživateli v konzoli kam se vypisují informace o přijatých datech. Zpráva začíná `$$I` pro informační zprávu a `$$W` pro varovnou (zvýrazněnou) zprávu. Zpráva není zakončena středníkem, výpis textu je průběžný a končí začátkem jiné zprávy. Text zprávy může obsahovat jakékoli znaky s výjimkou dvou po sobě jdoucích znaků dolar `$$` (sekvence pro začátek nové zprávy).

Typickým využitím těchto zprav je debug firmwaru pro mikrokontroler. Pro běžné používání je výhodnější použít terminál.

6.7 Chybová zpráva

Začíná sekvencí `$$X` po níž následuje text zakončený středníkem. Po přijetí celé zprávy je text zobrazen ve vyskakovacím okně a dojde k odpojení portu.

6.8 Zpráva typu echo

Tento typ zprávy je zahájen `$$E`. Při přijetí zprávy typu echo (ozvěna) je text zprávy odeslán zpět do připojeného zařízení. Například pokud je z mikrokontroleru odesláno `$$EABC`, počítač pošle odpověď `ABC`. Text může obsahovat jakékoli znaky s výjimkou dvou po sobě jdoucích znaků dolar `$$` (sekvence pro začátek nové zprávy).

Tento typ zprávy lze použít pro ověření spojení se zobrazovací aplikací. Také je možné tuto zprávu použít pro ověření, jestli je aplikace připravena přijmout další data. Pokud by totiž mikrokontroler nepřetržitě odesílal velké množství dat, mohlo by dojít k zahlcení programu. Zprávy jsou zpracovávány v pořadí, v jakém přichází. Pokud je například nejprve odeslán kanál `$$C.....` a poté `$$EX`, obdrží mikrokontroler odpověď `X` až poté co byl kanál zpracován. Lze takto realizovat jednoduchý „handshake“, kdy se mikrokontroler dotáže programu, jestli je připraven přijmout další data.

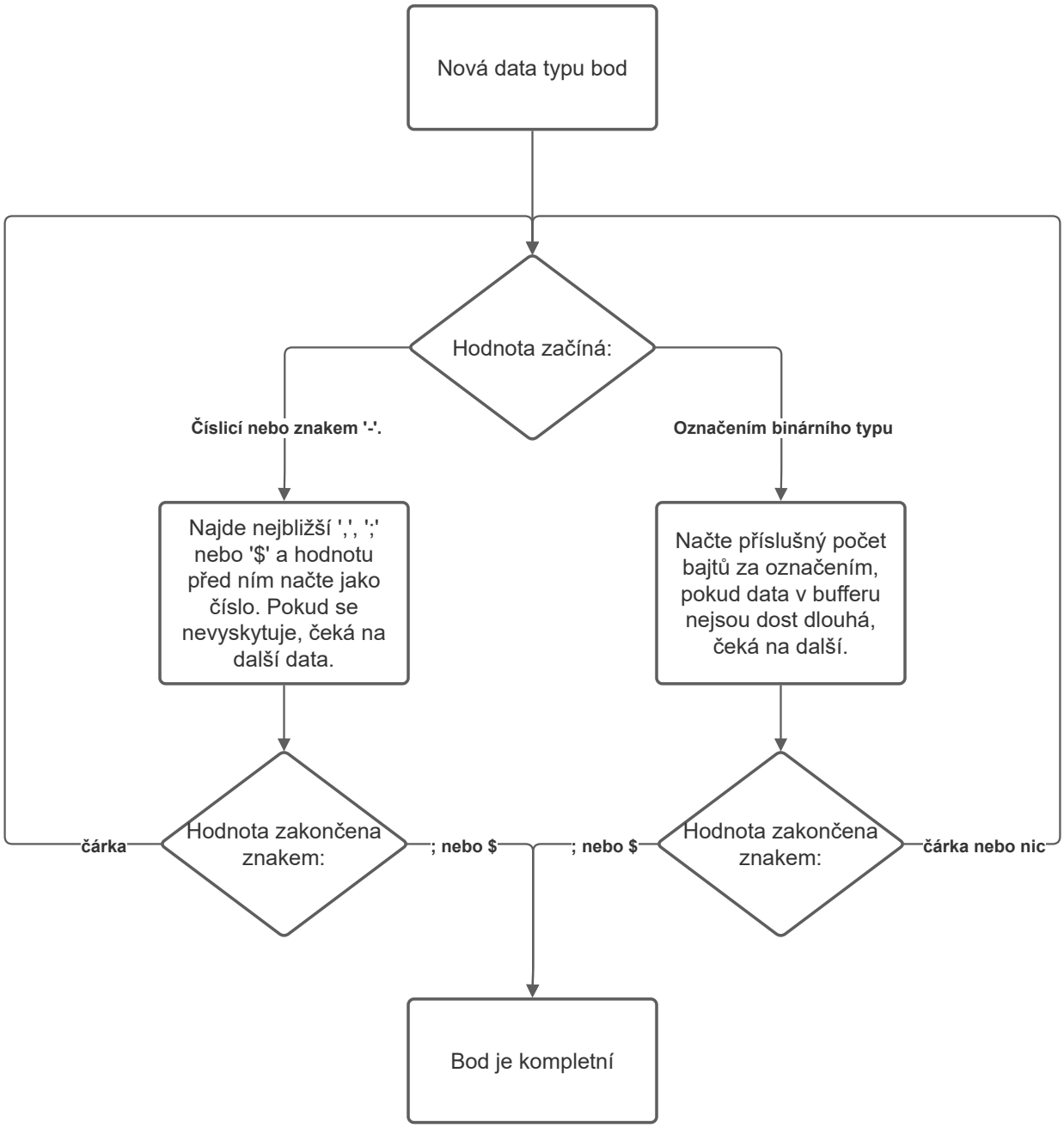
6.9 Nastavení

S pomocí těchto příkazů je možné nastavit parametry uživatelského rozhraní (například rozsahy grafu, popisky os grafu, ...). To umožňuje, aby připojené zařízení samo přednastavilo rozsah grafu tak aby odpovídal posílaným datům. Struktura zprávy je následující:

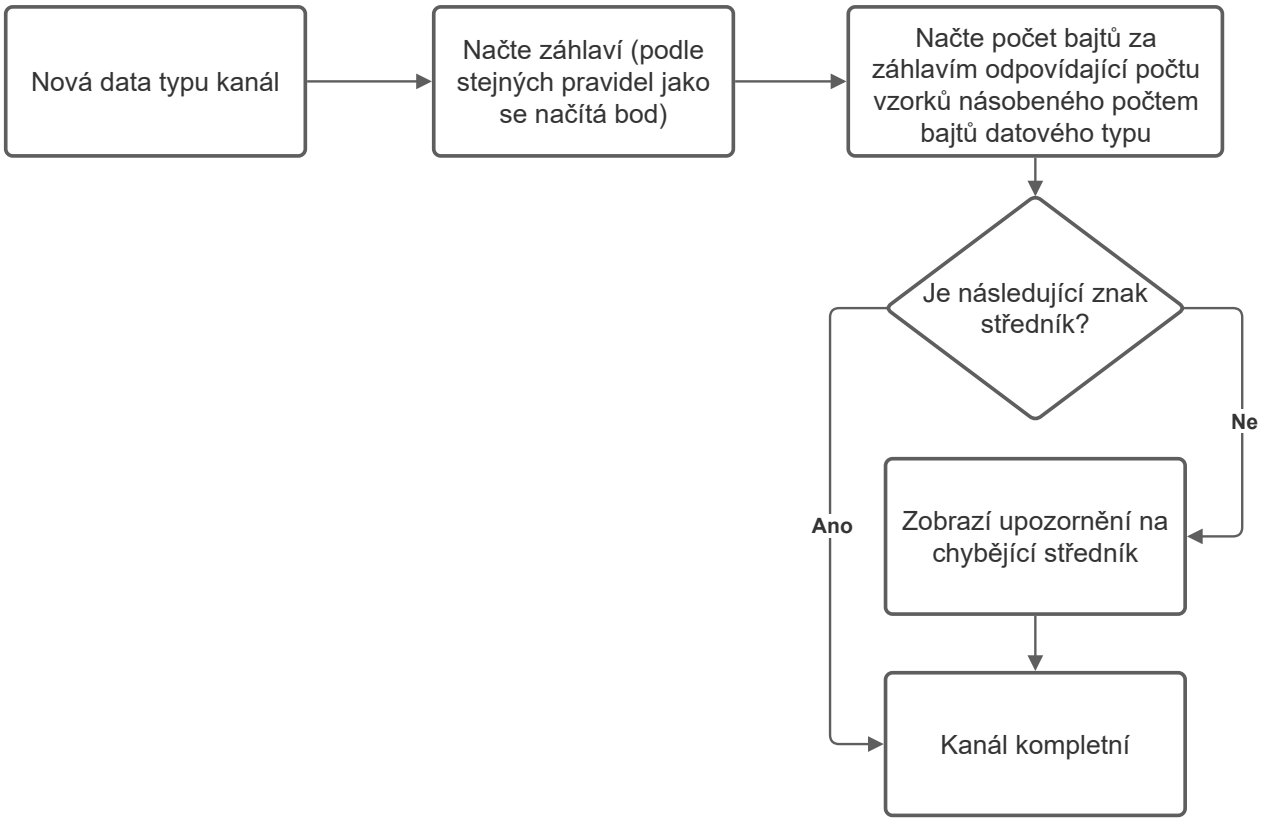
$$S*parameter* **:** *value***;**

- *parameter*: Nastavovaný parametr
- *value*: Hodnota, na jakou má být nastaven

Seznam nastavitelných parametrů a příkazů je v tabulce na konci uživatelské příručky k programu (příloha B). Čísla lze zadat jen jako napsané číslo, nikoli binárně.



Obrázek 6.1: Postup zpracování bodu



Obrázek 6.2: Postup zpracování kanálu

Kapitola 7

Zpracování příchozích dat

Odesílání dat je plně v režii mikrokontroleru a jejich množství může být prakticky libovolné. Zpracování velkého množství dat a zejména jeho vykreslení do grafu může trval nezanedbatelný čas a může negativně ovlivnit odezvu GUI (program by „se zasekávat“) Proto je důležité zajistit že:

- Zpracování dat probíhá v jiném vláknu než procesy spojené s GUI
- Graf není překreslován zbytečně často
- Pravidelné časově náročné výpočty (FFT, automatická měření, interpolace) nezačnou, dokud není dokončen předchozí výpočet, aby se nenahromadily požadavky[1] na výpočet.

7.1 Objekty a komunikace mezi nimi

Postup vytvoření objektů v různých vláknech programu a komunikaci mezi nimi jsem popsal v kapitole 3.1.2. Schéma 7.2 na straně 36 znázorňuje, jak spolu komunikují hlavní objekty v mém programu. V každém objektu ve schématu je napsáno, které soubory kódu se ho týkají.

7.2 Překreslování grafu

Graf nemá smysl překreslovat častěji, než jakou frekvenci člověk vnímá a monitor zobrazí. Za nejvyšší smysluplnou vykreslovací frekvenci se v obvykle považuje 60 Hz což zároveň bývá frekvence obyčejných monitorů. Pro zobrazení grafu však může dostačovat i frekvence mnohem menší (například 30 Hz).

Nejproblematičtější je z tohoto hlediska „rolling“ režim, kdy se s přibývajícími daty celý graf posouvá. Technicky není problém, aby mikrokontroler posílal stovky bodů (`$$P`) za sekundu. Rozhodně není žádoucí, aby byl graf překreslován po každém přijatém bodu. Program jsem tedy

[1] signály typu `queued` , viz 3.1.2

vytvořil tak, že překreslování grafu je prováděno v pravidelných intervalech (30 Hz) nezávisle na rychlosti přidávání dat.

7.3 Automatická měření a výpočet spektra

Automatická měření (popsány v kapitole 10) a výpočet spektra (kapitola 9) jsou výpočetně náročné a není žádoucí je provádět při každém přijetí nových dat. Měření postačuje aktualizovat několikrát za sekundu, v případě FFT je vhodné použít podobnou frekvenci jako v případě hlavního grafu.

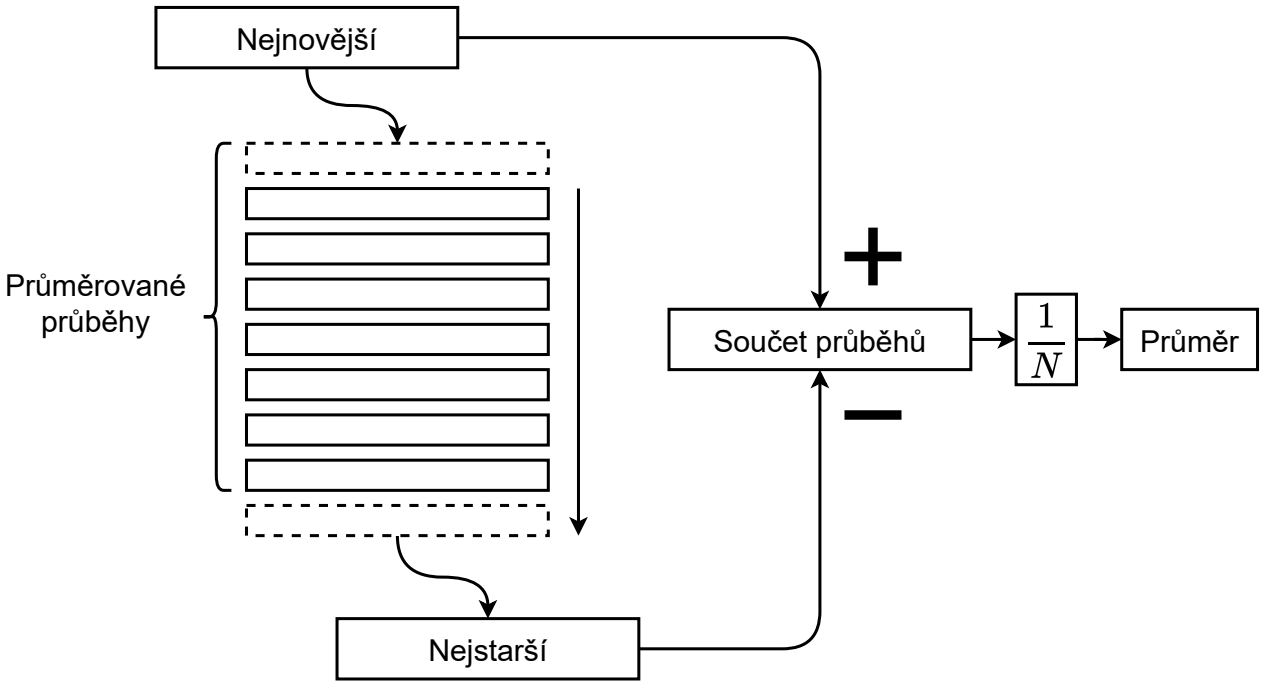
Samotné nastavení pevné obnovovací frekvence v tomto případě není dostatečné. Na pomalém počítači a nebo při velmi velkém množství vzorků může nastat situace, kdy výpočet trvá déle, než je perioda obnovování. Tomu jsem zabránil tak, že frekvence není pravidelná, ale po dokončení jednoho výpočtu je s pomocí časovače odměřen určitý čas (perioda požadované přibližné obnovovací frekvence) a potom teprve zahájím další výpočet.

7.4 Průměrování průběhů

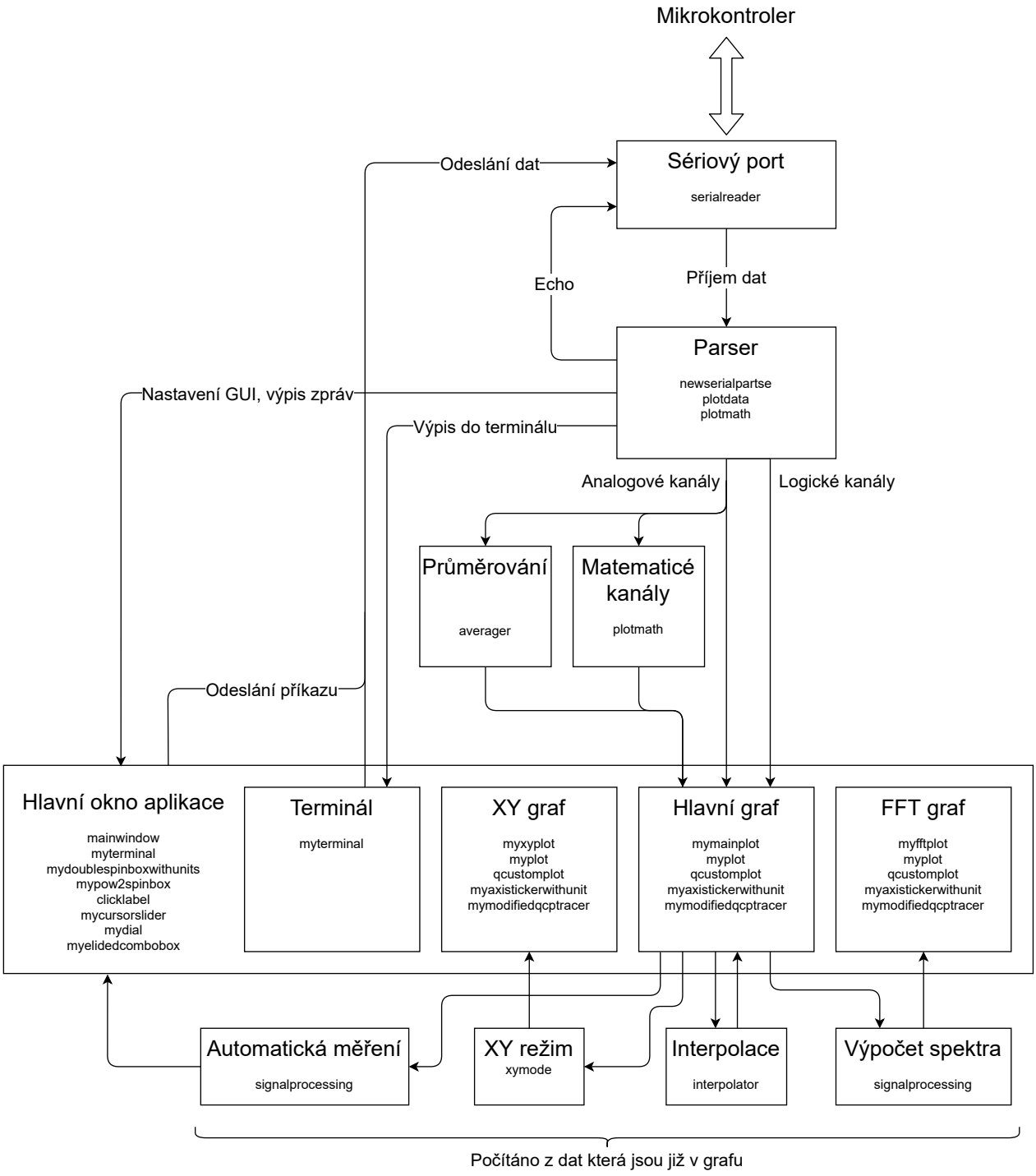
Pro snížení šumu je užitečné kanál průměrovat z více průběhů. Toho lze dosáhnout snadno tím, že program ukládá několik nejnovějších průběhů pro každý kanál a po přijetí nového průběhu pro daný kanál tyto průběhy zprůměruje. Kanály jsou uloženy v `QList`, nejnovější je přidán funkcí `append` a nejstarší odebrán pomocí `pop_front`. Průměr lze vypočítat sečtením vzorků z průběhů v seznamu a vydělením počtem průběhů.

Bylo by však velmi neefektivní po přidání nového průběhu sčítat přes všechny průběhy v seznamu. Řešení je zobrazeno na obrázku 7.1: Kromě samotného seznamu průběhů je uložen i jejich součet. Po přidání nového průběhu je nový průběh přičten k součtu a nejstarší (který bude odebrán ze seznamu) je naopak odečten. Takto lze efektivně přepočítávat součet průběhů pro libovolnou délku seznamu (počtu průměrovaných průběhů). Vzorky součtu jsou poté vyděleny počtem průběhů, z kterých byl vypočítán.

Postup je použitelný pouze pro variantu přidávání celých průběhů (`$$C`). V případě přidávání po bodech (`$$P`) není k dispozici více hodnot pro jeden čas. Funkce průměrování se pro tento případ chová jako klouzavý průměr.



Obrázek 7.1: Průměrování vzorků kanálu z N průběhů



Obrázek 7.2: Schéma propojení objektů programu

Kapitola 8

Graf

Graf je založen na knihovně QCustomPlot [6]. Počet průběhů, které lze do takového grafu vykreslit, není prakticky omezen. Program jsem ale navrhl tak, že umožňuje maximálně 16 analogových kanálů, 3 matematické a 3 skupiny logických kanálů po 32 bitech. Dynamické přidávání kanálů do grafu by vedlo k přílišným komplikacím a nepřehlednosti, proto jsem se rozhodl pro pevný maximální počet. Nepředpokládám, že by někdo chtěl zobrazit více než 16 analogových kanálů současně.

Ovládání grafu jsem navrhl tak, aby se podobalo ovládání osciloskopu, díky tomu by mělo být intuitivní a lehce pochopitelné. Jednotlivé kanály jsou odlišeny různou barvou. Kanály lze v grafu svisle posouvat tak, aby se nepřekrývali, ale byli zobrazeny nad sebou. Pokud je kanál svisle posunut, a tedy jeho nulová hodnota neodpovídá nule na svislé ose grafu, je pozice nuly daného kanálu zobrazena přerušovanou čarou. Graf je ukázán na obrázku 8.7 na straně 45. Čísla na osách grafu mohou být skryta (u posunutých anebo zvětšených kanálů jsou hodnoty na svislé ose nerelevantní) hodnoty je možné odečíst s pomocí kursorů.

8.1 Režimy rozsahu grafu

Graf má tři režimy:

- Pevný režim: Graf zobrazuje celý přijatý úsek hodnot. Zobrazení lze přiblížit pomocí funkce zoom. Tento režim je vhodný pro osciloskopy.
- Posuvný (rolling) režim: Graf zobrazuje nastavený časový úsek před posledním přijatým bodem (nové hodnoty přibývají z pravé strany grafu a starší se odsouvají vlevo). Tento režim je vhodný pro taková měření, kde data přicházejí po jednotlivých bodech s nízkou frekvencí, například voltmetr se záznamem.
- Volný režim: Graf je možné přibližovat a posouvat myší.

8.1.1 Pevný režim

V tomto režimu je vodorovný rozsah nastaven tak, aby obsahoval všechny body v grafu. Svislý rozsah uživatel nastaví.

Zoom lze nastavit v rozsahu 100 % až 0.1 %. Přiblížený graf lze posouvat do stran posuvníkem.

8.1.2 Rolling režim

Je zobrazen nastavený časový úsek před posledním vzorkem, celý graf se tedy posouvá vpravo s tím, jak přibývají nová data. Pokud je průběh v grafu kratší než nastavený interval, je zobrazen úsek od prvního vzorku směrem do kladných časů (prázdný prostor je tedy vpravo od průběhu).

Posunutí grafu s každým novým vzorkem nemusí být v určitých situacích ideální (při malém tempu přidávání vzorů je pohyb trhaný a hůře se odečítají hodnoty). Zavedl jsem tedy funkci, která graf posunuje po úsecích s náskokem oproti signálu: Pokud je funkce nastavena například na 30 %, tak poté co signál dojde na pravý okraj grafu se graf posune o 30 % rozsahu a čeká až signál opět dojde na okraj.

Svislý rozsah se nastavuje stejně jako v pevném režimu.

8.1.3 Volný režim

Funkce tažení grafu myší a zoom myší je již implementována v knihovně QCustomPlot, stačí ji pouze aktivovat.

```
1 this->setInteraction(QCP::iRangeDrag, true);
2 this->setInteraction(QCP::iRangeZoom, true);
```

8.2 Použití QCustomPlot

Použití grafu QCustomPlot je zdokumentováno v [11]. Pro přidání průběhu[1] do grafu se použije objekt `QCPGraph` (pro XY graf se použije `QCPCurve`). Po každé[2] změně je nutné graf překreslit pomocí funkce:

```
1 replot(QCustomPlot::RefreshPriority::rpQueuedReplot);
```

Argument funkce značí, že překreslení neproběhne ihned při zavolání funkce, ale až při následující iteraci `event loop`. Pokud by tedy byla funkce zavolána vícekrát za sebou, nedojde zbytečně k opakovanému vykreslení.

[1] V angličtině se rozlišují pojmy „plot“: graf jako celek a „graph“: jeden průběh (křivka) v grafu. „plot“ zde nazývám „graf“ a „graph“ nazývám „průběh“ nebo „kanál“.

[2] V některých případech je překreslení automatické, například při posouvání nebo přibližování myší.

8.2.1 Analogové kanály

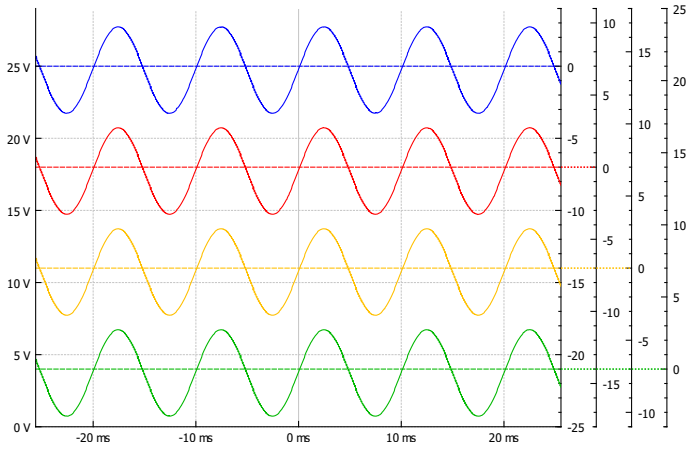
Aby zobrazení napodobovalo klasický osciloskop, je nutné implementovat možnost svisle posouvat kanály (rozložit více kanálů nad sebe) a také jejich vzájemné zvětšení nebo zmenšení. Objekt `QCPGraph` nenabízí možnost nastavit „offset“ ani změnit měřítko. Jako možné řešení jsem nejprve zkusil posuv implementovat tak, že jsem velikost posunutí přičítal k hodnotám všech vzorků. To však bylo velmi neefektivní.

Nakonec jsem objevil způsob, jak zajistit manipulaci s kanálem bez nutnosti změn hodnot v něm uložených. QCustomPlot umožňuje mít v grafu více os a různým průběhům přiřadit jiné osy. Měřítko os lze nezávisle měnit a stejně tak mohou mít nulu na různé pozici. Přidal jsem tedy do grafu samostatnou osu pro každý kanál, jak je zobrazeno na obrázku 8.1. Tyto osy je samozřejmě nutné skrýt, což lze provést tímto kódem:

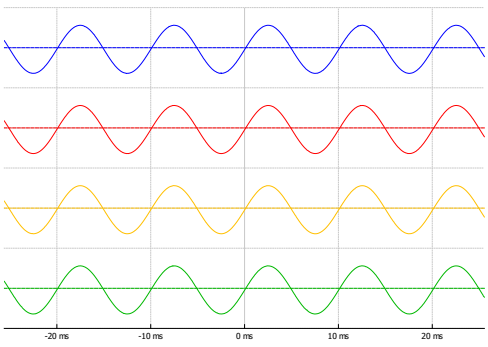
```
1 axis->setTicks(false);
2 axis->setBasePen(Qt::NoPen);
3 axis->setOffset(0);
4 axis->setPadding(0);
5 axis->setLabelPadding(0);
6 axis->setTickLabelPadding(0);
7 axis->setTickLength(0, 0);
```

Také je žádoucí skrýt hodnoty na svislé ose:

```
1 void MyPlot::setShowVerticalValues(bool enabled) {
2     this->yAxis->setTicks(enabled);
3     this->yAxis->setBasePen(enabled ? Qt::SolidLine : Qt::NoPen);
4 }
```



(a) : Dodatečné osy ponechány zobrazené pro názornost



(b) : Skryté osy

Obrázek 8.1: Použití více svislých os pro svislý posuv kanálů

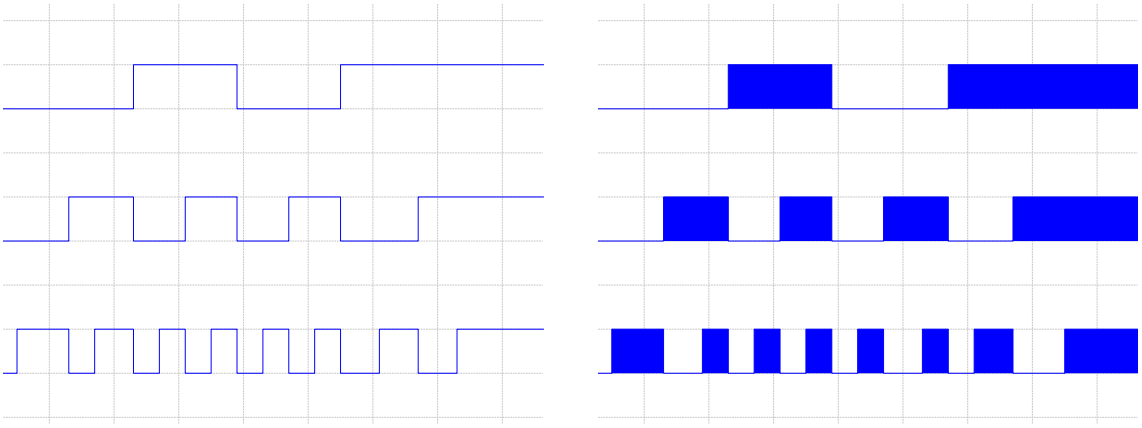
8.2.2 Logický analyzátor

Kromě analogových kanálů lze do grafu přidat i kanály logického analyzátoru. Pro každý kanál (1 bit) je využit samostatný `QCPGraph`. Tyto jednotlivé kanály (bity) mají však přiřazenu stejnou osu, a tedy se jejich svislý posun a zvětšení ovládá společně. Rozložení jednotlivých bitů na sebou je řešeno přičtením konstanty. Nízká úroveň každého bitu je na hodnotě o 3 vyšší než předchozí bit. Nultý bit má nízkou úroveň na nule (vysokou na 1), první na hodnotě 3, druhý na 6, a tak dále. Převod hodnoty na skutečný stav bitu (0 nebo 1) lze snadno provést jako zbytek po dělení třemi.

Podbarvení průběhu v grafu

Pro přehlednější zobrazení bitů logického analyzátoru je možné graf podbarvit jak je ukázáno na obrázku 8.2. QCustomPlot sice umožňuje vybarvit plochu pod grafem, ale pouze mezi průběhem a hodnotou 0. To pro tento účel není použitelné, protože nízká úroveň logického kanálu není na nule (s výjimkou bitu 0). Proto jsem kód QCustomPlot mírně upravil, abych umožnil nastavit jako spodek podbarvení i jinou hodnotu než 0.

Do třídy `QCPGraph` jsem přidal proměnnou `fillBaseLine` pomocí které nastavím spodní hranici podbarvení. Ve funkci `QPointF QCPGraph::getFillBasePoint(QPointF matchingDataPoint) const` jsem řádek `result.setY(valueAxis->coordToPixel(0));`
změnil na `result.setY(valueAxis->coordToPixel(this->fillBaseLine));`



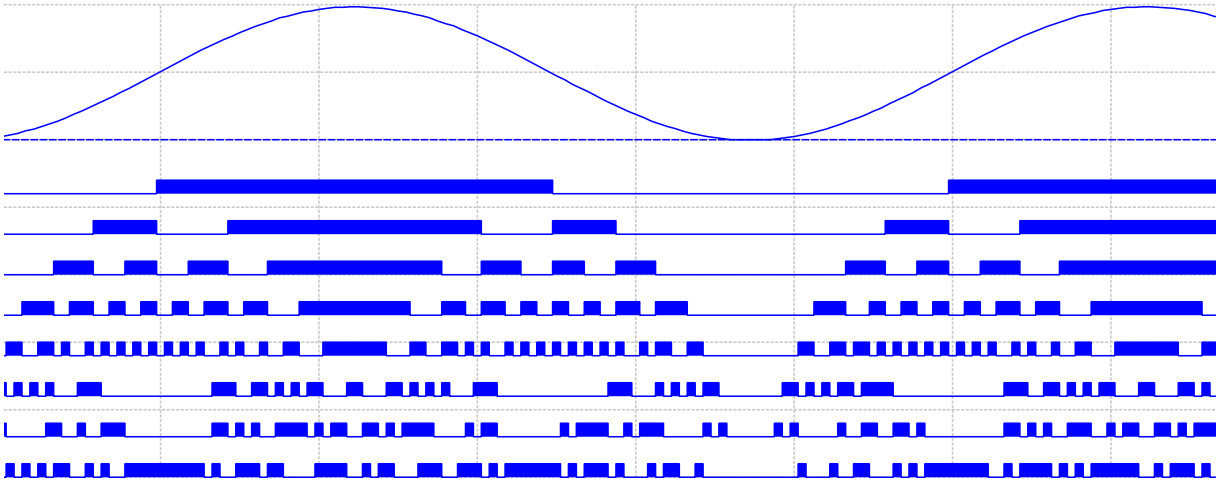
(a) : Základní zobrazení **(b) :** S podbarvením

Obrázek 8.2: Zobrazení logického analyzátoru

8.2.3 Kanály výpočtů

Program umožňuje provádět základní matematické operace (sčítání, odčítání, násobení a dělení kanálů). Výpočet je proveden vždy po přidání nového průběhu (`$$C`) nebo bodu (`$$B`), po zapnutí

je zpětně dopočítán pro již zobrazená data. Také jsem přidal možnost převést analogoví kanál přidaný v datovém typu `unsigned integer` na bity logického analyzátoru.



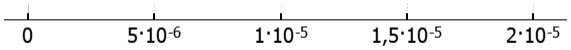
Obrázek 8.3: Zobrazení kanálu a jednotlivých bitů AD převodníku

8.3 Mřížka a osy grafu

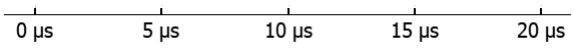
Hustota hodnot na osách a tím i mřížky se automaticky nastavuje podle rozsahu a to tak, že krok mřížky je přibližně desetina rozsahu (zaokrouhluje se na „hezké“ hodnoty `1, 2, 5, 10, 20, 50...`). V závislosti na velikosti okna a rozlišení displeje toto může být příliš nebo málo. Proto je možné hustotu mřížky nastavit na tři úrovně (poloviční a dvojnásobná)[3]. U nastavení mřížky se zobrazuje krok mřížky v jednotkách na dílek, jako je běžné u osciloskopů. To má význam zejména při zobrazení více kanálů rozložených nad sebou, kdy absolutní hodnoty na svislé ose jsou irelevantní (svislou osu lze zcela skrýt).

8.3.1 Zobrazení hodnot na osách

QCustomPlot umožňuje zobrazit hodnoty s pevným počtem desetinných míst nebo ve vědecké notaci (s hezky vysázeným 10^x, ne s „e+x“), neumožňuje však použití jednotek a jejich předpon (mili, mikro...). Tuto funkci jsem přidal.



(a) : Základní zobrazení



(b) : S jednotkami

Obrázek 8.4: Zobrazení hodnot na ose grafu

[3]Opět se drží řady `1, 2, 5, 10, 20, 50...`, takže „poloviční“ z 5 je 2 a „dvojnásobná“ z 2 je 5.

Hodnoty s jednotkami

Vytvořil jsem třídu založenou na třídě `QCPAxisTickerFixed` sloužící pro vytvoření popisků hodnot na osách. Algoritmus pro převod čísla na text s předponou jednotky je popsán v sekci 4.3. Pro účel os grafu má však určité nevýhody:

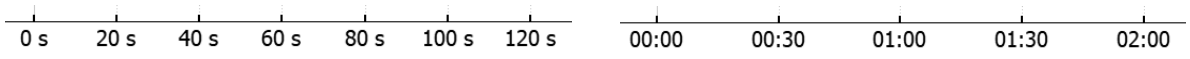
- Předpona je určena pro každou hodnotu zvlášť. Pokud by například na ose grafu bylo: `0, 0.5, 1...`, bylo by toto zobrazeno jako: `0 V, 500 mV, 1~V...`. Nekonzistence v jednotkách, kdy jedna hodnota je v milivoltech, kdežto ostatní ve voltech vypadá nehezky, bylo by lepší mít vše ve stejných voltech: `0.0 V, 0.5 V, 1.0 V...`
- Ztráta rozlišení ve velkých číslech. Pokud bych graf přiblížil nikoli na okolí nuly ale například na čase 2 sekundy tak, že by krok mřížky byl milisekunda, hodnoty by se zobrazovaly v sekundách a v případě malého počtu desetinných míst by se mohli ztratit v zaokrouhlení.

Řešení je však velmi snadné, stačí pro určení předpony namísto čísla samotného použít nastavený krok mřížky. Za předponu je přidána nastavená jednotka. Některé jednotky předponu nevyužívají (bezrozměrné, decibely), pro ně se moje verze `QCPAxisTicker` chová stejně jako původní `QCPAxisTickerFixed`.

Časy v hodinách, minutách a sekundách

QCustomPlot kromě obyčejných os umožňuje použít osy které zobrazují čas ve formátu MM:SS, případně HH:MM:SS. Toto zobrazení je vhodné pro pomaloběžné průběhy a záznamy po delší dobu. Čas je v tomto případě zadáván v sekundách. Pokud je jednotky vodorovné osy nastavena na `s`, je umožněno přepnout osy do těchto režimů.

Nastavení mřížky funguje obdobně, ale čáry mřížky se v tomto případě automaticky umísťují na „kulaté“ hodnoty z hlediska tohoto způsobu zobrazení (celé sekundy/minuty), nelze tedy dodržet přesné hodnoty sekund na dílek, informace o jednotkách na dílek je proto skryta.



(a) : Sekundy **(b) :** Minuty a sekundy

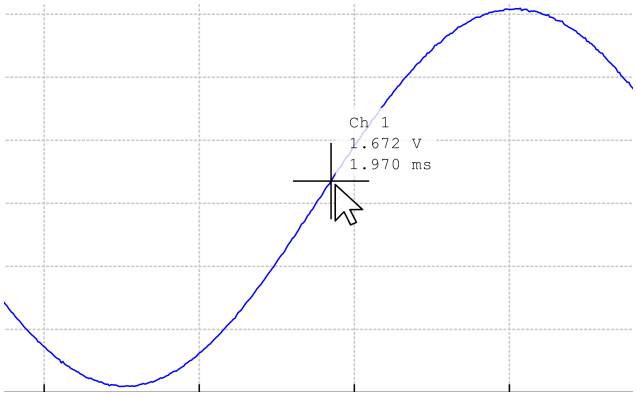
Obrázek 8.5: Zobrazení času na ose grafu

8.4 Kurzory

Kurzory umožňují jednoduchý odečet hodnot kanálu z grafu. Jak je obvyklé, jsou k disposici dvě dvojice kurzorů: časové a napěťové. Časový kurzor lze umístit na vzorek signálu (v takovém případě se napěťový kurzor umístí na hodnotu toho vzorku), nebo lze časové a napěťové kurzory umístit na libovolný čas, respektive hodnotu.

8.4.1 Zobrazení hodnoty po najetí myší na kanál

Pro rychlé zobrazení hodnoty vzorku jsem vytvořil funkci, která zobrazí hodnotu po najetí ukazatele myši na kanál, jak je zobrazeno na obrázku 8.6. Postup je založený na objektu `QCPTracer` který je součástí knihovny QCustomPlot. Jedná se o značku (např. křížek) která se umístí na vzorek průběhu v grafu a s měnící se hodnotou se automaticky posouvá. Musel jsem však udělat několik úprav, proto jsem na základě třídy `QCPTracer` vytvořil upravený `MyModifiedQCPTracer`.



Obrázek 8.6: Zobrazení hodnoty kanálu po najetí kurzorem myši

Pohyb ukazatele myši po grafu vyvolá signál `mouseMoved(QMouseEvent* event)`. V reakci na tento signál zjistím, ke kterému kanálu je ukazatel myši nejblíže (případně že k žádnému není dostatečně blízko). Pro zjištění vzdálenosti kanálu od ukazatele myši je v QCustomPlot funkce `QCPGraph::selectTest`

```
1  // Najde nejbližší kanál k myši.
2  // Pokud žádný není blíže než 20 pixelů, zůstane proměnná nearestIndex na -1.
3  int nearestIndex = -1;
4  unsigned int nearestDistance = 20;
5  for (int i = 0; i < NUMBER_OF_CHANNELS; i++) {
6    if (graph(i)->visible()) {
7      unsigned int distance = graph(i)->selectTest(event->pos(), false);
8      if (distance < nearestDistance) {
9        nearestIndex = i;
10       nearestDistance = distance;
11     }
12   }
13 }
```

Následně je `QCPTracer` umístěn na takto nalezený nejbližší kanál a vzorek nejblíže k pozici ukazatele myši. Původní `QCPTracer` pro určení nejbližšího vzorku využívá pouze vodorovnou souřadnici. To má zásadní nevýhody:

- Zejména v případě strmého průběhu je vzorek s nejbližší vodorovnou souřadnicí výrazně odlišný od skutečně nejbližšího.
- Nekompatibilita s XY grafem, kde vodorovná souřadnice není jednoznačně určující.

Ve svém `MyModifiedQCPTracer` jsem tedy postup hledání nejbližšího vzorku upravil tak, že bere v úvahu obě souřadnice (součet druhých mocnin rozdílů). To navíc umožňuje snadné hledání lokálních extrémů průběhu, stačí najet myší přibližně nad vrchol, který poté bude vyhodnocen jako nejbližší bod. To lze využít zejména v grafu FFT. Důležitý poznatek je, že pro posouzení vzdálenosti je nutné použít souřadnice v pixelech na obrazovce, nikoli v souřadnicích grafu, protože osy grafu mohou mít velmi odlišná měřítka. Pokud by totiž například svislá osa byla řádově v jednotkách voltů a vodorovná v tisícinách sekundy, byla by jakákoli vodorovná vzdálenost zanedbatelná vůči svislé.

K samotné značce `QCPTracer` jsem přidal i popisek který zobrazí hodnotu. Pro umístění popisku ke značce je výhodné použít systém ukotvení („anchor“), kdy není nutné pozici textu ručně měnit, ale je vázaná ke značce. Popisek je standardně umístěn vpravo dole od značky. Přidal jsem však funkci, která ho přemístí na vhodnější stranu, pokud by se nevešel do zobrazení.

8.4.2 Ovládání kurzorů myší

Po kliknutí na graf (signál `mousePress(QMouseEvent*)`) je možné určit vzorek nejblíže k místu kliknutí stejným postupem jako bylo zmíněno v předchozí sekci (8.4.1). Umístit kurzor na tento vzorek je poté triviální záležitost, stačí mu nastavit příslušný kanál a vzorek.

Protože mám dva kurzory, je třeba je nějak odlišit. První kurzor se tedy umístí kliknutím levým tlačítkem myši, druhý pravým tlačítkem. Obě kliknutí vyvolají signál `mousePress(QMouseEvent*)` a použité tlačítko je možné určit z proměné `QMouseEvent`:

```
1 if (event->button() == Qt::RightButton)
2     // Pravé tlačítko
3 else
4     // Levé tlačítko
```

Tažení kurzru myší

Po kliknutí myší do grafu kromě blízkosti ke kanálu posuzuji i blízkost ke kurzoru. Pokud vyhodnotím, že uživatel klikl na kurzor, přejde program do režimu tažení kurzoru myší. Při pohybu myši (signál `mouseMoved`) je měněna pozice kurzoru. V případě časového kurzoru je přesunut na vzorek s časem nejbližším k vodorovné pozici myši. Napěťový kurzor se po popotažení přepne do režimu libovolné hodnoty (přestane být vázaný k hodnotě vzorku na kterém je časový kurzor) a přemístí se na svislou pozici myši. Tažení kurzoru je ukončeno puštěním tlačítka (signál `mouseReleased`).

Podobným postupem lze také táhnout čáru označující nulovou úroveň kanálu a tím měnit jeho offset.

8.5 Zobrazení pozice triggeru

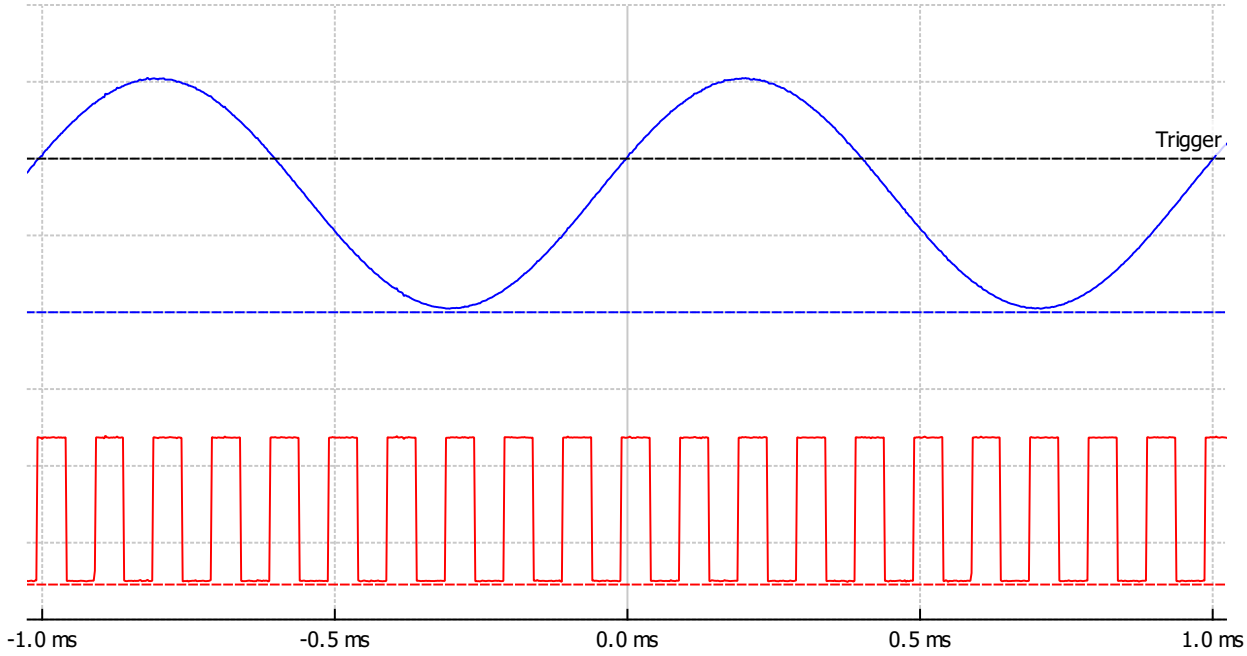
Osciloskopy obvykle na displeji zobrazují nastavenou úroveň pro trigger. To je v tomto případě problematické, protože nastavení triggeru je záležitostí samotného přístroje (mikrokontroleru) nikoli

zobrazovací aplikace. Zobrazení úrovně a její nastavení je možné přes terminál (více o ovládání pomocí terminálu v kapitole 12.2). Přidal jsem i možnost zobrazení úrovně přímo v grafu, jak je vidět na obrázku 8.7. K ovládání jsou použity příkazy typu `$$S` (nastavení). Příkazy jsou:

- `trigline` zobrazí či skryje čáru označující trigger
- `trigch` nastaví na kterém kanálu se trigger nachází
- `trigpos` nastaví úroveň (napětí) na které je čára umístěna

Kromě zobrazení a skrytí lze nastavit i automatický režim, v kterém je čára dočasně zobrazena po změně úrovně či kanálu.

Podrobnější popis příkazů pro nastavení je na konci Uživatelské příručky (příloha B).



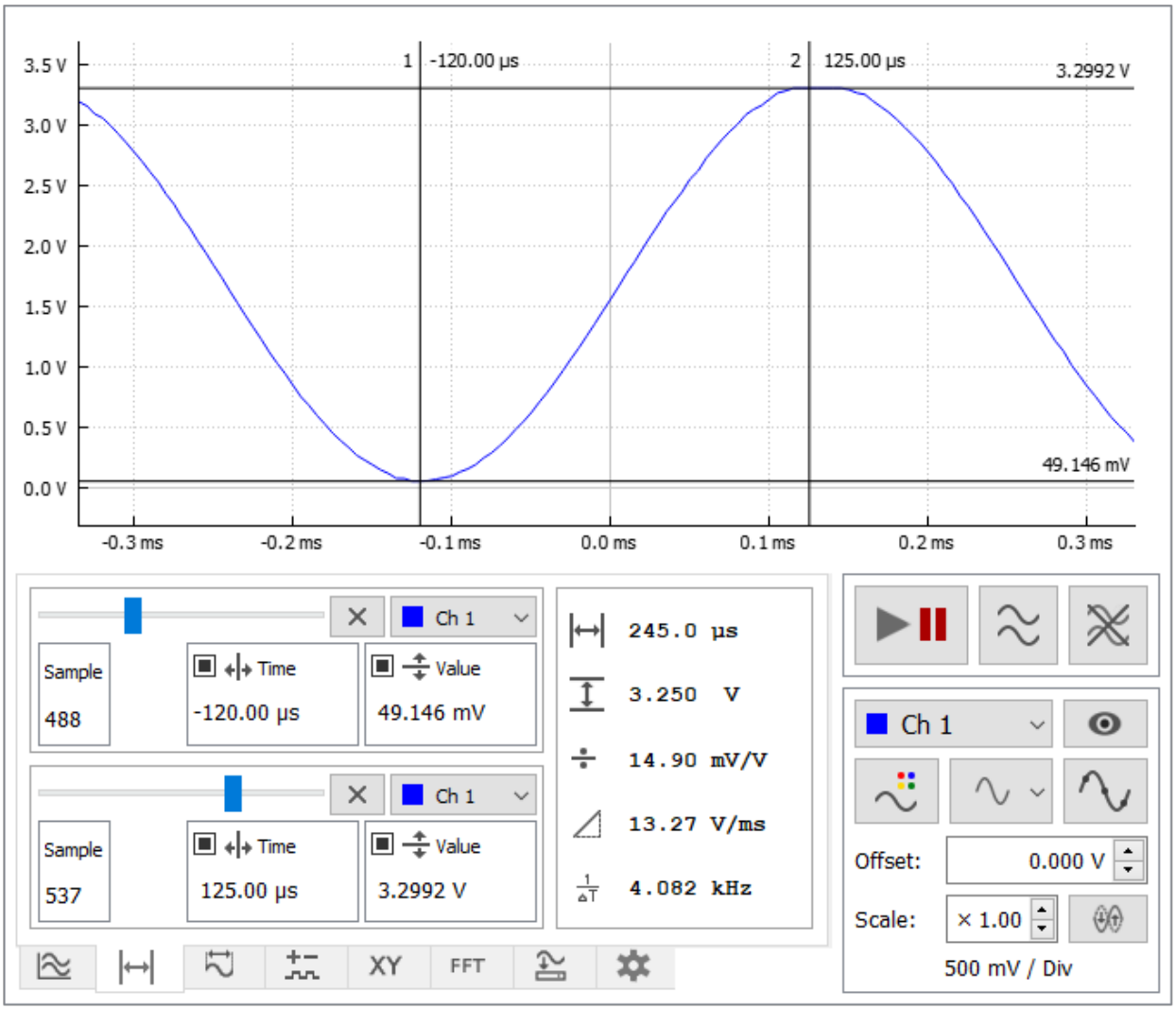
Obrázek 8.7: Zobrazení dvou kanálů v grafu a vyznačení úrovně triggeru

8.6 Automatické rozložení kanálů nad sebe

Program umožňuje kliknutím na tlačítko rozložit více kanálů nad sebe tak, aby se nepřekrývaly. Automatické nastavení offsetů kanálů je uděláno takto: (n označuje aktuálně nastavovaný kanál, $n-1$ je předchozí kanál, který je v grafu pod ním)

$$offset[n] = offset[n-1] + max[n-1] - min[n]. \tag{8.1}$$

Minimální a maximální hodnoty kanálů jsou zaokrouhleny nahoru (směrem od nuly, dolů pro záporná čísla) na „hezké“ hodnoty: `1, 2, 5, 10, 20, 50...`. V případě, že kanál je přidán pomocí zprávy typu `$$C` s přemapováním hodnot, tedy jsou uvedeny parametry *min* a *max*, jsou tyto hodnoty použity namísto zaokrouhlených extrémů průběhu. Program postupuje od posledního kanálu k prvnímu, první je tedy nejvýše a poslední má spodek na nule. Pro vzorec 8.1 tedy platí, že $n = N - ch$, kde N je maximální počet kanálů (16) a ch je číslo kanálu (1 až 16).



Obrázek 8.8: Kurzory v grafu

Kapitola 9

Výpočet spektra signálu

Spektrum signálu je graf ve frekvenční oblasti, který popisuje, jak výrazně jsou v signálu zastoupeny jednotlivé frekvence. Proces přeměny signálu v časové oblasti do frekvenční oblasti se nazývá Fourierova transformace, jedná se o integrální transformaci, definovanou vztahem

$$F(\omega) = \int_{-\infty}^{\infty} f(t) \cdot e^{-i\omega t} dt. \tag{9.1}$$

Výsledkem je graf s komplexními hodnotami, amplitudové spektrum je jeho absolutní hodnota. Pro reálný signál je spektrum symetrické okolo nulové frekvence. V případě navzorkovaného signálu se použije diskrétní varianta, označovaná DFT (Discrete Fourier Transform).

9.1 Diskrétní Fourierova transformace

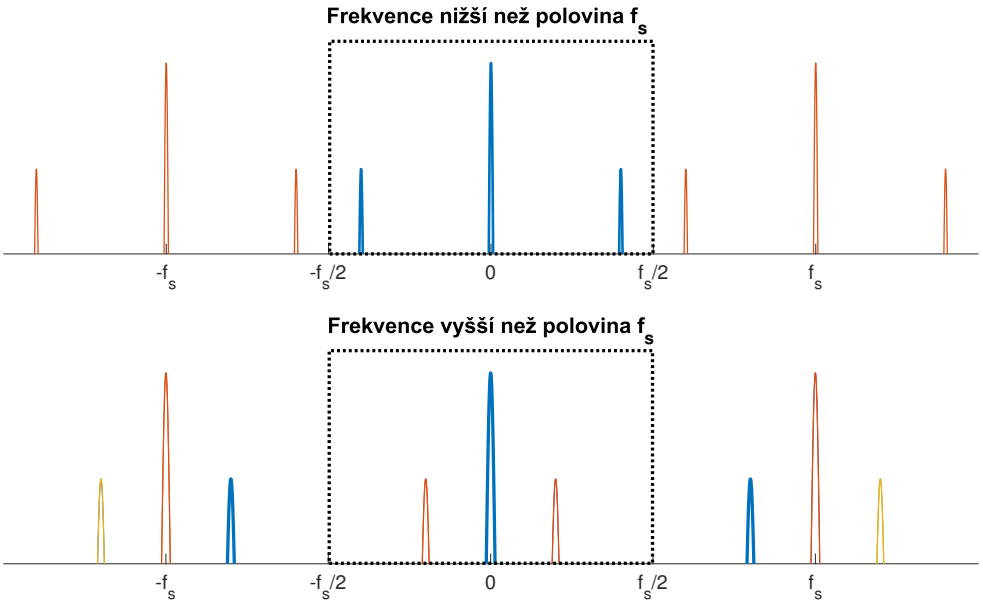
Diskrétní Fourierova transformace je matematický nástroj pro výpočet spektra ze vzorků signálu. Tato transformace je popsána vztahem

$$X_k = \sum_{n=0}^{N-1} x_n \, e^{-i\frac{2\pi}{N}nk}, \tag{9.2}$$

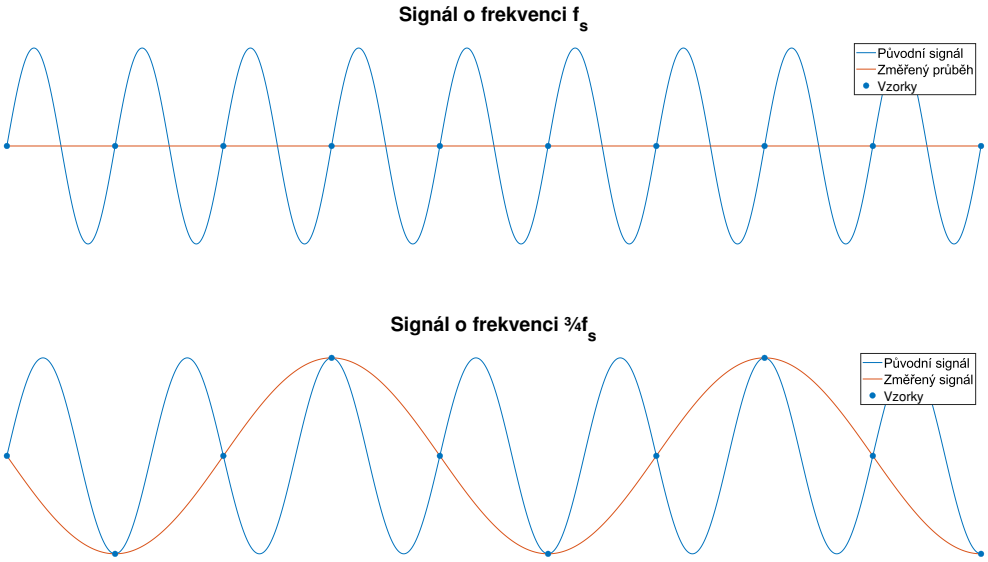
kde vzorky spektra X_k a vzorky v čase x_n jsou indexovány od nuly do $N-1$. Vzorek s indexem N (který už je mimo vypočtený rozsah) odpovídá frekvenci f_s. Spektrum se periodicky opakuje s periodou N vzorků, tedy například hodnota v f_s se rovná hodnotě v 0. To vyplývá z vlastností funkce $e^{-i\frac{2\pi}{N}nk}$.

Pokud signál obsahuje frekvence větší než $\frac{f_s}{2}$, dojde k tomu, že odpovídající spektrální čára v prvním periodickém opakování spektra pronikne do základní části spektra, jak je znázorněno na obrázku 9.1. Tím se ve spektru objeví složky, které v původním signálu nejsou, tento efekt se nazývá aliasing. V časové oblasti jsem tento efekt znázornil na obrázku 9.2: první graf ukazuje jak lze signál o frekvenci f_s zaměnit za stejnosměrný, druhý graf zobrazuje, jak se shodují vzorky signálů o frekvencích $\frac{1}{4}f_s$ a $\frac{3}{4}f_s$.

Pro správné navzorkování signálu nesmí signál obsahovat frekvenční složky vyšší, než je polovina vzorkovací frekvence, to se nazývá vzorkovací podmínka nebo Nyquistovo kritérium.



Obrázek 9.1: Periodické opakování spektra diskrétního signálu a aliasing



Obrázek 9.2: Vzorkování signálu s frekvencí vyšší, než je polovina vzorkovací frekvence

Výpočet DFT z tohoto definičního vztahu je velmi neefektivní, má výpočetní náročnost $O(N^2)$ (pro každý z N vzorků spektra násobíme N vzorků signálu komplexní exponenciálou). Pro efektivnější výpočet lze využít algoritmus FFT (Fast Fourier Transform).

9.2 Rychlá Fourierova transformace

Algoritmus FFT umožňuje vypočítat DFT mnohem efektivněji než z definičního vzorce. Tento algoritmus byl poprvé publikován v roce 1965 Jamesem Cooleym a Johnem Tukeym, ale výpočet na tomto principu prý již dlouho před tím použil Gauss v nepublikované práci [12].

Přestože existují knihovny v C++ pro výpočet FFT, rozhodl jsem se tuto funkci implementovat sám, abych algoritmu lépe porozuměl. Informace jsem čerpal z článku [13] a videa [14], kde je uveden i příklad implementace v programovacím jazyce Python. Nyní se pokusím shrnout princip tohoto algoritmu:

9.2.1 Algoritmus FFT

Vyhodnocení DFT v jednom bodě je vlastně výpočet hodnoty polynomu. Zavedu-li substituci $W_N^k = e^{-i\frac{2\pi}{N}k}$ pak můžu sumu 9.2 zapsat jako polynom s proměnnou W_N^k.

$$P(W_N^k) = x_0 {W_N^k}^0 + x_1 {W_N^k}^1 + x_2 {W_N^k}^2 + x_3 {W_N^k}^3 + x_4 {W_N^k}^4 + x_5 {W_N^k}^5 + x_6 {W_N^k}^6 + x_7 {W_N^k}^7 \tag{9.3}$$

Základní myšlenkou tohoto algoritmu je rozdělení výpočtu na sudé a liché členy. Nejprve změním pořadí členů tak aby se sudé a liché oddělili:

$$P(W_N^k) = x_0 {W_N^k}^0 + x_2 {W_N^k}^2 + x_4 {W_N^k}^4 + x_6 {W_N^k}^6 + x_1 {W_N^k}^1 + x_3 {W_N^k}^3 + x_5 {W_N^k}^5 + x_7 {W_N^k}^7 \tag{9.4}$$

poté z druhé poloviny vytknu W_N^k:

$$P(W_N^k) = x_0 {W_N^k}^0 + x_2 {W_N^k}^2 + x_4 {W_N^k}^4 + x_6 {W_N^k}^6 + W_N^k(x_1 {W_N^k}^0 + x_3 {W_N^k}^2 + x_5 {W_N^k}^4 + x_7 {W_N^k}^6) \tag{9.5}$$

a zapíšu jako součet dvou polynomů

$$P(W_N^k) = P_e(W_N^k) + W_N^k \cdot P_o(W_N^k) \tag{9.6}$$

$$P_e(W_N^k) = x_0 {W_N^k}^0 + x_2 {W_N^k}^2 + x_4 {W_N^k}^4 + x_6 {W_N^k}^6 \tag{9.7}$$

$$P_o(W_N^{2k}) = x_1 {W_N^k}^0 + x_3 {W_N^k}^2 + x_5 {W_N^k}^4 + x_7 {W_N^k}^6 \tag{9.8}$$

Tyto polynomy mají pouze sudé mocniny, lze je tedy zapsat jako funkci proměnné W_N^{2k}. Původní polynom měl stupeň $N-1$, tyto dva nové jsou stupně $\frac{N}{2}-1$.

$$P_e(W_N^{2k}) = x_0 W_N^{2k0} + x_2 W_N^{2k1} + x_4 W_N^{2k2} + x_6 W_N^{2k3} \tag{9.9}$$

$$P_o(W_N^{2k}) = x_1 W_N^{2k0} + x_3 W_N^{2k1} + x_5 W_N^{2k2} + x_7 W_N^{2k3} \tag{9.10}$$

Díky sudé symetrii P_e a liché symetrii $W_N^k \cdot P_o$ lze vypočítat hodnoty pro dva vzájemně opačné argumenty s použitím stejných hodnot P_e a $W_N^k \cdot P_o$.

$$P(W_N^k) = P_e(W_N^{2k}) + W_N^k \cdot P_o(W_N^{2k}) \tag{9.11}$$

$$P(-W_N^k) = P_e(W_N^{2k}) - W_N^k \cdot P_o(W_N^{2k}) \tag{9.12}$$

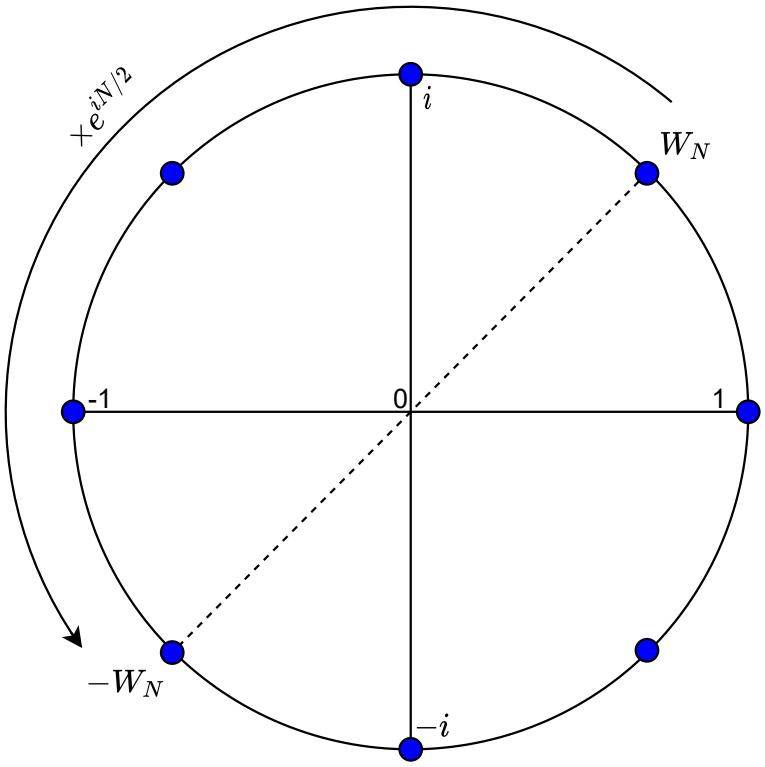
Hodnoty W_N^k jsou Nté kořeny jedničky (anglicky „roots of unity“). Jde o N čísel rovnoměrně rozložených na jednotkové kružnici. Jestliže je N sudé číslo (viz podmínky v dalším odstavci), tak ke každému kořenu existuje další kořen, který je naproti němu (středově symetrický kolem nuly) a tedy je jeho opačnou hodnotou. Tyto dva kořeny jsou od sebe půl kruhu daleko, tedy mocnina se liší o $\frac{N}{2}$. Proto platí:

$$-W_N^k = W_N^{k+\frac{N}{2}} \tag{9.13}$$

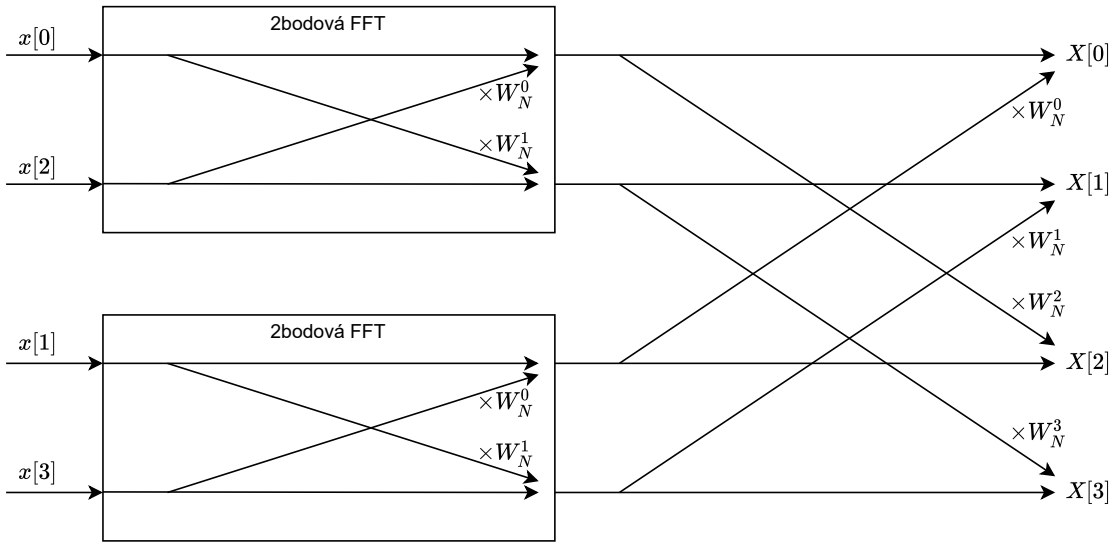
Hodnoty $P_e(W_N^{2k})$ a $P_o(W_N^{2k})$ se počítají rekurzivně (stejným postupem jakým nyní počítám $P(W_N^k)$) dokud se nedojde k polynomu stupně 0 (tam je řešení triviální - hodnota jediného vzorku). Výpočet Nbodové DFT se tedy změnil na dvě $\frac{N}{2}$bodové DFT, které jsou opět rozděleny na dvě poloviční a tak dále až do jednobodové, kde rekurze končí.

Aby rekurze fungovala, musí být splněny tyto podmínky:

- Počet vzorků N (tedy stupeň polynomu) musí být mocnina 2 (v každém kroku se snižuje na polovinu až do hodnoty 1).
- Symetrie použitá ve vztazích 9.11 a 9.12 vyžaduje, aby ke každému W_N^k existovalo $-W_N^k$. Aby totéž fungovalo v dalším kroku rekurze, musí totéž platit pro W_N^{2k}. To je zaručeno právě tím, že $W_N^k = e^{-i\frac{2\pi}{N}k}$ jsou vždy sudé komplexní kořeny jedničky (vždy jsou dva naproti sobě).

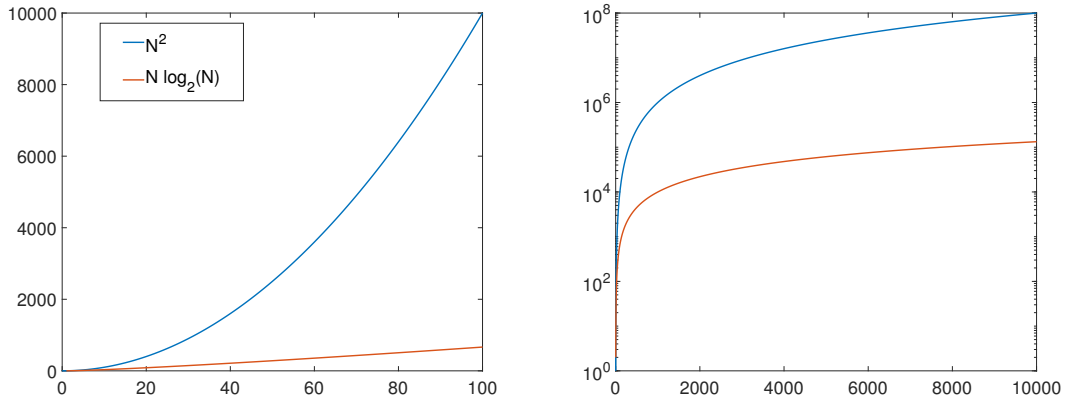


Obrázek 9.3: Osmé odmocniny z jedné v komplexních číslech



Obrázek 9.4: Schéma čtyřbodové FFT

Pro výpočet Nbodové DFT tímto postupem je potřeba dvakrát vypočítat $\frac{N}{2}$bodovou DFT a ještě provést N násobení číslem W_N^k. To vede na výpočetní náročnost $O(N \cdot log_2(N))$, což je pro velké hodnoty N téměř lineární (logaritmus roste velmi pomalu). Jedná se tedy o výrazné zlepšení oproti vypočtu z definičního vztahu DFT jak je vidět z grafu 9.5.



Obrázek 9.5: Srovnání výpočetní náročnosti DFT a FFT

Normování hodnot v amplitudovém spektru

Výsledné spektrum je lineárně úměrné počtu vzorků N (což je nejlépe vidět na $f = 0$ kde jde o pouhý součet všech vzorků, jak je vidět z rovnice 9.2 pro $k = 0$). Pokud spektrum ještě vydělím počtem vzorků, bude se hodnota v $f = 0$ rovnat stejnosměrné složce. Ostatní čáry (vrcholy) ve spektru ukazují amplitudu (ve smyslu $A \cdot cos(\dots)$, ne V_{pp}). Hodnoty jsou oproti skutečné amplitudě poloviční, protože každá složka se objeví dvakrát (na f a $-f$). V případě doplnění nulami (9.2.3) dělím původním počtem vzorků před doplněním, aby nemělo vliv na amplitudu.

9.2.2 Implementace FFT v C++

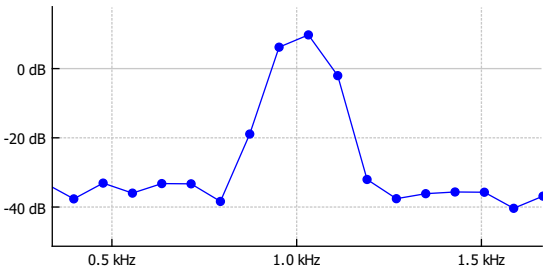
Toto je můj kód pro výpočet DFT algoritmem FFT:

```
QVector<std::complex<double>> fft(QVector<std::complex<double>> x) {
    int N = x.size();
    if (N == 1) return x; // Konec rekurze

    QVector<std::complex<double>> Pe, Po; // Sudé a liché koeficienty
    for (int n = 1; n < N; n += 2) {
        Pe.append(x.at(n - 1)); // Sudé
        Po.append(x.at(n));     // Liché
    }

    QVector<std::complex<double>> Xe = fft(Pe), Xo = fft(Po); // Rekurze
    QVector<std::complex<double>> X;
    X.resize(N);
    for (int k = 0; k < N / 2; k++) {
        // exp(i*2*Pi*k/N)
        double arg = M_PI * 2 * k / N;
        std::complex<double> WNk(cos(arg), sin(arg));
        X[k] = Xe[k] + WNk * Xo[k];
        X[k + N / 2] = Xe[k] - WNk * Xo[k];
      }
    return X;
}
```

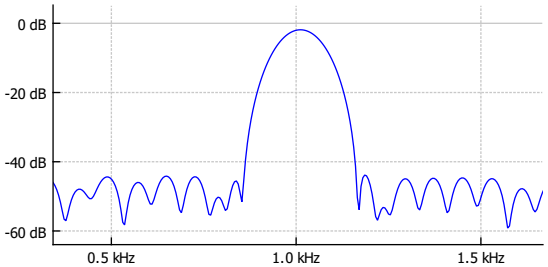
9.2.3 Doplnění signálu nulami

Použitý algoritmus FFT vyžaduje, aby počet vzorků signálu byl mocnina dvou. Pro signál, jehož počet vzorků není mocnina dvou toto řeším doplněním nulami na nejbližší vyšší mocninu dvou.

Doplnění nulami také umožňuje zlepšit rozlišení ve spektru, jak je ukázáno na obrázku 9.6: V prvním případě nelze přesně lokalizovat polohu maxima, protože leží mezi dvěma vzorky. V druhém případě je použito doplnění nulami, což zvýší počet vzorků spektra a tím sníží vzdálenost mezi nimi (zjemní rozlišení ve frekvenci), to umožní přesněji určit polohu maxima.



(a) : Signál délky 128 vzorků



(b) : Signál délky 128 vzorků, doplněný nulami na 1024 vzorků

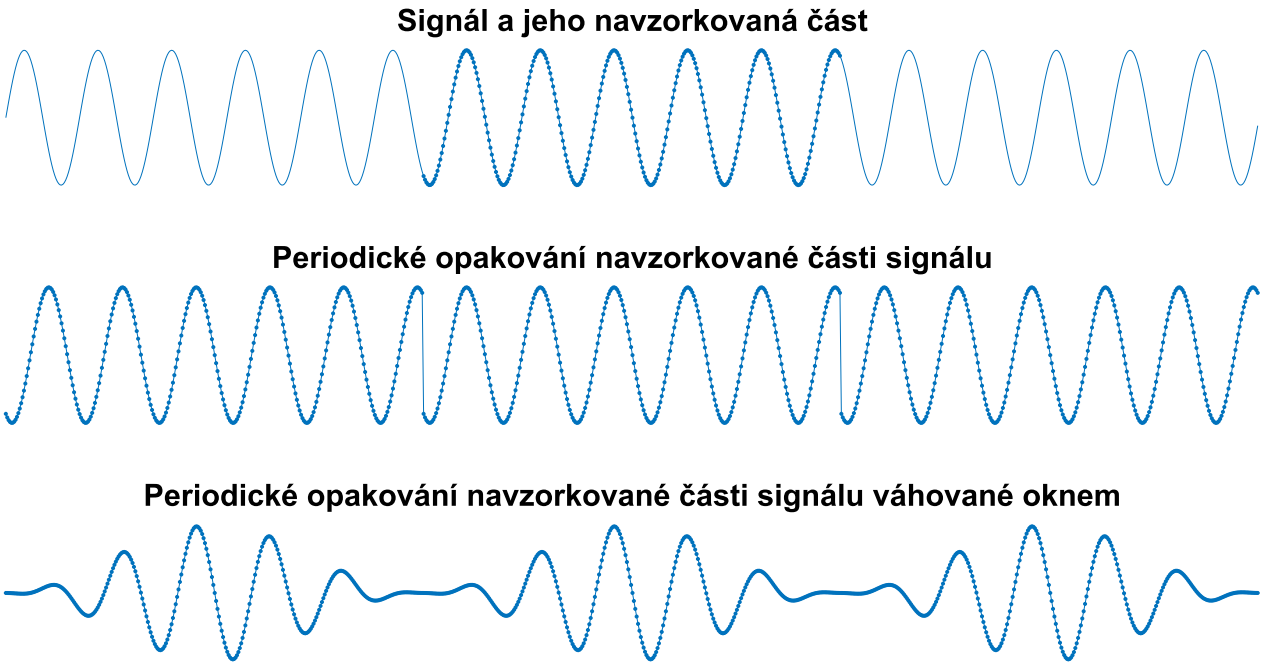
Obrázek 9.6: Zvýšení rozlišení ve frekvenci pomocí doplnění nulami

9.3 Váhování oknem

Fourierova transformace je definována pro signál nekonečně dlouhý v čase. V praxi však máme k dispozici pouze úsek konečné délky. Diskrétní Fourierova transformace (DFT) se chová jako by se úsek, z kterého ji počítáme, periodicky opakoval. Pokud bychom v naměřeném úseku měli celý počet základních period signálu, bylo by to v pořádku. Problém nastává, pokud máme necelý počet period, jak je znázorněno na prvním grafu v obrázku 9.7. Pokud se tento úsek periodicky zopakuje, úseky na sebe nenavazují plynule (druhý graf, 9.7). Tyto nespojitosti se ve spektru projeví jako další frekvenční složky, které ale v původním signálu nebyly (a tedy jsou nežádoucí). Tento jev se nazývá prosakování ve spektru.

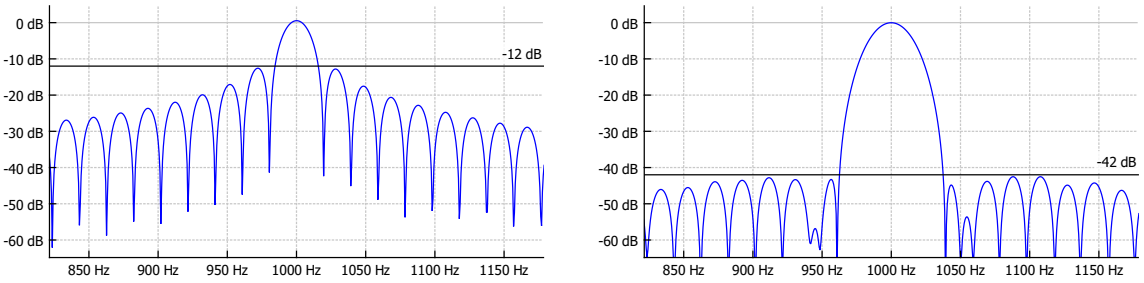
Řešením je použití váhovacího okna. Základní (neváhované) okno se nazývá obdélníkové. Váhovací okna se od něho liší tím, že vzorky na okrajích okna (navzorkovaného úseku) jsou potlačeny, a tedy na sebe lépe navazují, jak je vidět z třetího grafu v obrázku 9.7.

V ideálním případě (nekonečně dlouhý periodický signál) by pro harmonický signál ve spektru byl pouze jeden Diracův impuls na příslušné frekvenci. Pokud použijeme okno (obdélníkové i váhovací), tak provádíme násobení signálu oknem v časové oblasti. Násobení v čase se projeví jako konvoluce ve spektru. Konvoluce obrazů okna a signálu se projeví tak, že na místě, kde měl být Diracův impulz je posazen obraz okna. V případě obdélníkového okna je obrazem funkce sinc, ta má uprostřed hlavní lalok o výšce 1 a okolo něho jsou postranní laloky[1] 12 dB pod úrovní hlavního, jak je vidět na obrázku 9.8. Cílem váhovacího okna je co nejvíce potlačit tyto postranní laloky. [15, s. 694-710]



Obrázek 9.7: Periodické opakování konečného úseku signálu

[1] Pokud je navzorkován přesně celý počet period, laloky od kladné a záporné frekvenční složky a jejich periodických opakování se vzájemně odečtou, proto se prosakování v takovém případě neprojeví, což odpovídá intuici s plynulým napojením periodických kopií.



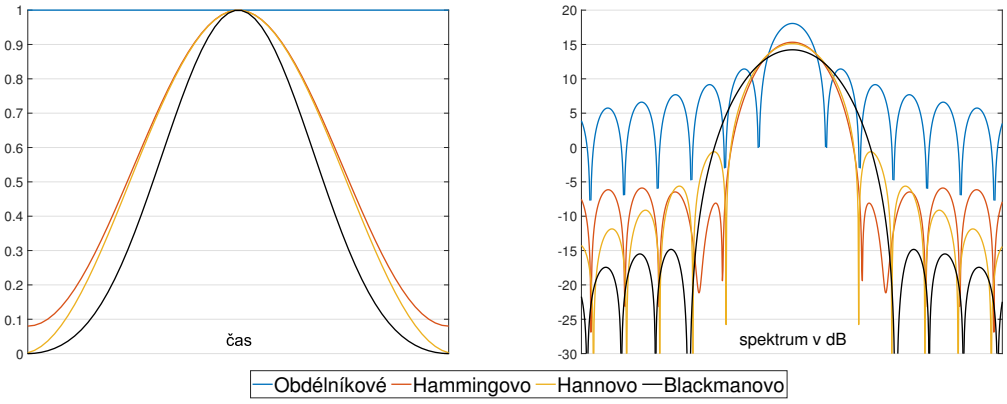
(a) : Použito obdélníkové okno **(b) :** Použito Hammingovo okno

Obrázek 9.8: Prosakování ve spektru

Nejčastěji používaná váhovací okna jsou Hammingovo, Hannovo a Blackmanovo. Tato tři okna jsem v tomto programu použil (kromě nich lze zvolit i obdélníkové). Vzorce pro výpočet těchto oken jsem si zjistil z dokumentace programu Matlab (která se odkazuje na knihu od Oppenheima [15]).

- Hammingovo: $0.54 - 0.46\cos(2\pi\frac{n}{N})$
- Hannovo: $0.5 - 0.5\cos(2\pi\frac{n}{N})$
- Blackmanovo: $0.42 - 0.5\cos(2\pi\frac{n}{N}) + 0.08\cos(4\pi\frac{n}{N})$

Tyto vzorce generují okno na rozsahu 0 až $N-1$ (s vrcholem v $\frac{N}{2}$).



Obrázek 9.9: Váhovací okna

Použití okna ovlivní velikosti hodnot ve spektru, aby bylo dodrženo normovaní z 9.2.1, je třeba vliv okna kompenzovat. To lze udělat vydělením vzorků spektra průměrnou hodnotou okna. Tu lze pro Hammingovo okno vypočítat jako

$$\frac{1}{N}\sum_{n=0}^{N-1}\left(0.54 - 0.46\cos\left(2\pi\frac{n}{N}\right)\right). \tag{9.14}$$

Pro velká N lze hodnotu $\frac{n}{N}$ považovat za číslo které se spojitě mění od 0 do 1. Proto lze průměrnou hodnotu okna pro dostatečně velké N přibližně zapsat jako

$$\lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} \left(0.54 - 0.46\cos\left(2\pi\frac{n}{N}\right)\right) = \int_0^1 (0.54 - 0.46\cos(2\pi x))\ dx, \tag{9.15}$$

což se zjevně rovná 0.54, protože $\cos(2\pi x)$ integrujeme přes jednu celou periodu, což vyjde 0. Obdobně pro ostatní okna.

9.4 Periodogram

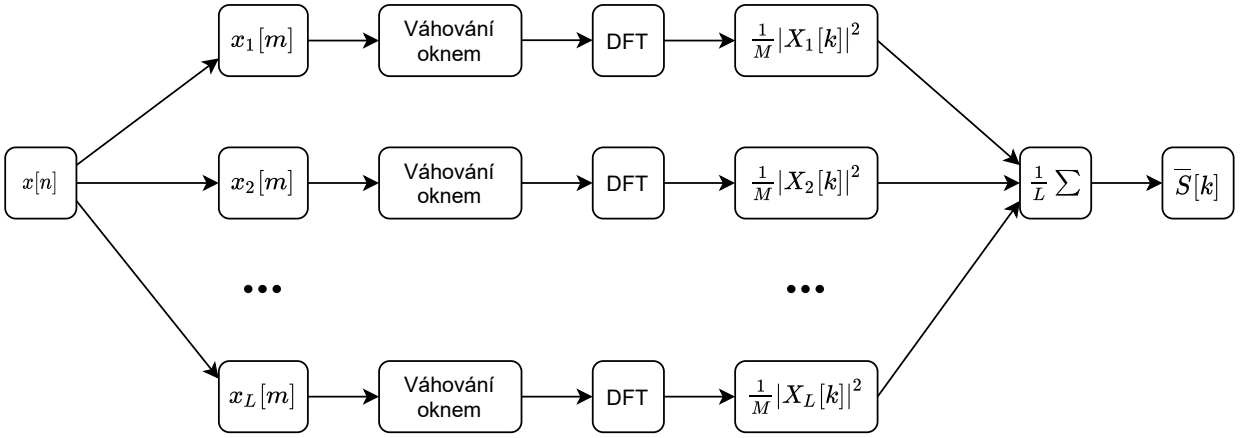
Přepočet spektra X_k na periodogram je obvykle definován jako $\frac{1}{N}|X_k|^2$. Já však již provádím dělení hodnotou N při výpočtu spektra, aby hodnoty byly nezávislé na počtu vzorků. Periodogram v decibelech tedy počítám jako

$$10 \cdot log_{10}|X_k|^2. \tag{9.16}$$

Hodnotu $|X_k|^2$ lze efektivně vypočítat jako součin X_k s komplexně sdruženým X_k^*. Tím se ušetří zbytečné počítání odmocniny při výpočtu absolutní hodnoty, která by se poté znovu umocnila. Hodnota 0 dB odpovídá jednomu voltu (jednotce na svislé ose v lineárním měřítku podle 9.2.1).

9.4.1 Welchova metoda

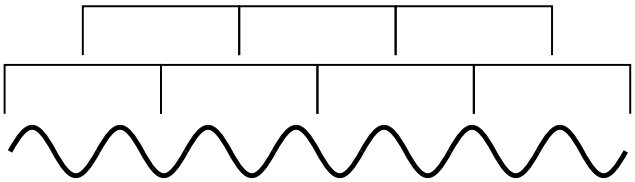
Periodogram poskytuje odhad spektrální výkonové hustoty, jeho rozptyl ale nezávisí na délce záznamu (větší počet vzorků nezlepší rozptyl periodogramu). Řešením je rozdělit signál na několik kratších úseků, vypočítat periodogram pro každý zvlášť a následně je zprůměrovat [15, 548–554]. Postup výpočtu je znázorněn na schématu 9.10.



Obrázek 9.10: Welchova metoda

Nejjednodušší způsob rozdělení signálu na segmenty by bylo rozdělení signálu délky N na L úseků délky $\frac{N}{L}$. Takovýto způsob je však neoptimální, protože v důsledku váhování segmentů oknem

jsou vzorky na okrajích segmentů potlačeny a tím se ztrácí informace. Řešením je segmentování s překryvem, při použití 50% překryvu jsou vzorky u hranice dvou segmentů zároveň ve středu jiného segmentu, jak je znázorněno na obrázku 9.11.



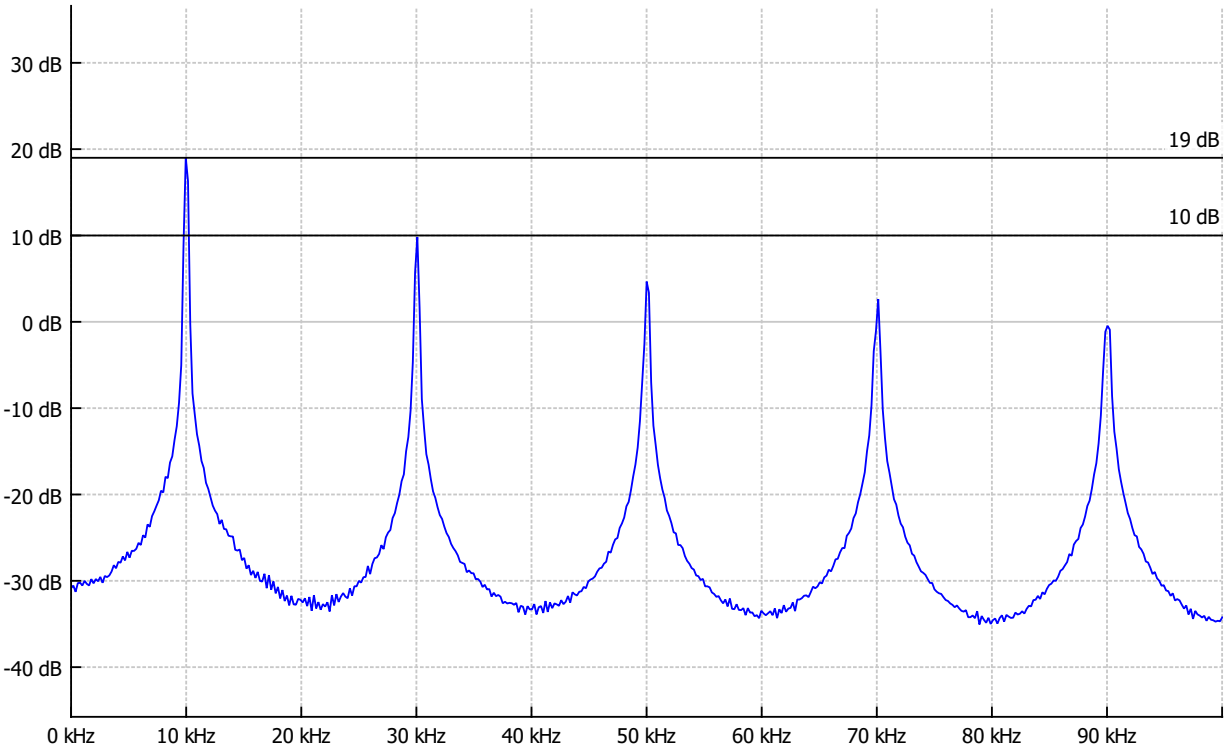
Obrázek 9.11: Segmentace v čase s 50% překryvem

9.5 Příklad vypočteného periodogramu

Na obrázku 9.12 je zobrazen periodogram obdélníkového signálu o frekvenci 10 kHz. Rozklad obdélníkového signálu do Fourierovy řady vypadá takto:

$$f(t) = \frac{4}{\pi} \sum_{n=0}^{\infty} \frac{1}{2n+1} \sin(2\pi f(2n+1)t). \tag{9.17}$$

Amplituda složky s frekvencí $3f$ je třikrát menší než základní harmonická složka f, $\frac{1}{3} \approx -9dB$. To odpovídá vypočtenému spektru zobrazenému v grafu.



Obrázek 9.12: Periodogram obdélníkového signálu

Kapitola 10

Automatické měření parametrů signálu

Základním způsobem měření parametrů signálu (například perioda, minimální a maximální hodnota, doba náběžné nebo sestupné hrany) je odečet hodnot s pomocí kurzorů a následný výpočet. Pro uživatele je však pohodlnější, pokud jsou tyto hodnoty zjištěny automaticky. Další charakteristiky, jako například stejnosměrná složka a efektivní hodnota, vyžadují sečíst velké množství hodnot, což vyžaduje zpracování počítačem.

Hodnoty jsou počítány z dat zobrazených v grafu a aktualizují se přibližně čtyřikrát za sekundu (po dokončení výpočtu program počká 250 ms, než zahájí nový výpočet, aby nedocházelo k nahromadění požadavků na výpočet, pokud by jeden trval déle než oněch 250 ms).

Výpočet lze nastavit pro dva kanály současně a obnovuje se pouze pokud je zobrazena stránka výpočtů, aby se hodnoty nepřepočítávali zbytečně když je uživatel nemá zobrazené.

Program umí vypočítat a zobrazit následující charakteristiky signálu:

- Frekvence a perioda
- Amplituda (špička–špička)
- Efektivní hodnota
- Počet vzorků
- Vzorkovací frekvence
- Minimální a maximální hodnota
- Stejnosměrná složka
- Doba náběhu a sestupu

Minuimum a maximum jsou jednoduše globální extrémy průběhu (nebo jeho části). Amplituda (ve smyslu peak–peak) je jejich rozdíl.

Stejnosměrná složka je aritmetický průměr hodnot:

$$V_{DC} = \frac{1}{N} \sum_{n=0}^{N-1} x[n] \tag{10.1}$$

Efektivní hodnota (nazývaná také RMS, podle „root mean square“) je odmocněný aritmetický průměr druhých mocnin hodnot:

$$V_{RMS} = \sqrt{\frac{1}{N} \sum_{n=0}^{N-1} (x[n])^2} \tag{10.2}$$

10.1 Frekvence a perioda

Pomocí osciloskopu obvykle měříme periodické signály a jejich frekvence je jedním z nejdůležitějších údaje které o nich můžeme chtít zjistit.

Základní frekvence (základní harmonická) je nejnižší frekvenční složka signálu. Nemusí se nutně jednat o nejvýraznější frekvenci. K výpočtu základní periody lze využít autokorelační funkci:

10.1.1 Výpočet pomocí autokorelační funkce

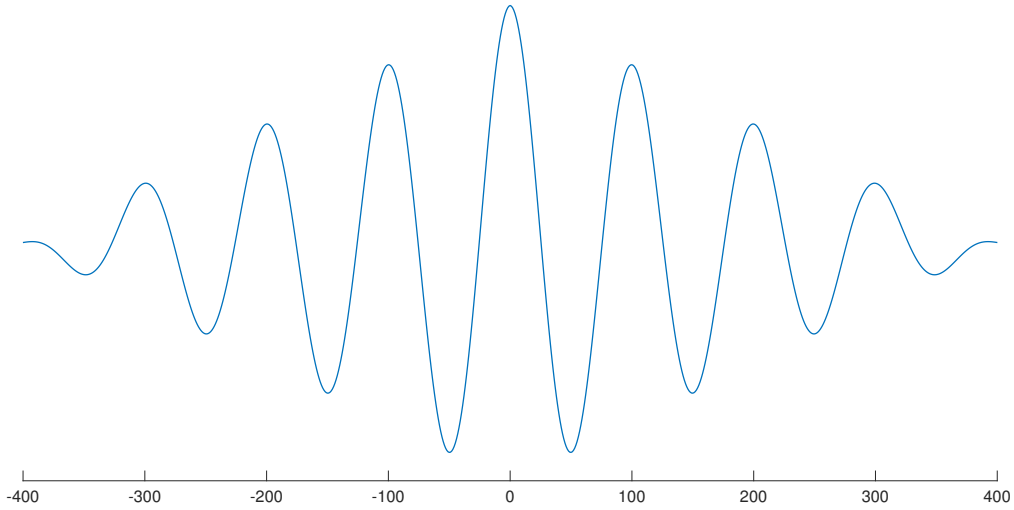
Autokorelační funkce je v podstatě výpočet vzájemné energie signálu a jeho v čase posunuté kopie.

$$\hat{R}_x[k] = \frac{1}{N} \sum_{n=0}^{N-1-|k|} x[n+k] \cdot x^*[n], |k| < (N-1) \tag{10.3}$$

Maximum autokorelační funkce je samozřejmě při nulovém posuvu k, tam je překryv dokonalý. Pokud je signál periodický, dochází k velmi blízkému překryvu i při posunutí o celý násobek periody. To je znázorněno na obrázku 10.1: Lokální maxima nastávají, když posuv (vodorovná osa) je roven celému násobku periody. Velikosti maxim se směrem od středu snižují, to je dáno tím, že jsou vypočteny z průběhu konečné délky (jde o superpozici sinu a obdélníkového okna, jehož autokorelace je trojúhelníková), tomuto odhadu autokorelační funkce se říká vychýlený. Perioda signálu je poloha prvního maxima vynásobená periodou vzorkování.

Nevýhody výpočtu frekvence pomocí autokorelační funkce

- Jeden krok posunu má velikost $\frac{1}{f_s}$, to pro vyšší frekvence vede k vysoké relativní odchylce vlivem diskretizace. Tato metoda je tedy použitelná jen pro signály s frekvencí řádově nižší, než f_s.
- Nalezení „toho správného“ lokálního maxima je velmi netriviální úkol.
- Výpočet autokorelační funkce (v časové oblasti, z definičního vztahu) má kvadratickou výpočetní složitost.



Obrázek 10.1: Autokorelační funkce sinu s periodou 100 vzorků

10.1.2 Jiné možnosti zjištění frekvence

Základní frekvence sice obecně nemusí být nejvýraznější z obsažených frekvenčních složek, ale u jednoduchých signálů, o kterých se dá předpokládat, že se budou tímto programem měřit (sinus, obdélníkový) tomu tak je. Proto jsem se rozhodl, že se zaměřím na hledání nejvýraznější obsažené frekvence.

Toho lze docílit mnohem jednodušeji, stačí vypočítat spektrum (funkci k tomu již mám, viz kapitola 9) a najít globální maximum. Ještě před tím je potřeba vypočítat a odečíst stejnosměrnou složku, jinak by globální maximum mohlo být v nule. Pokud je signál krátký (malý počet vzorků), doplním jej nulami pro zlepšení rozlišení (viz podkapitola 9.2.3).

Přesnost výpočtu jsem otestoval s pomocí mého osciloskopu na STM32L412 (popsán dále v této práci) a generátoru signálu UNI-T UTG932. Frekvenci signálu jsem nastavoval tisíckrát nižší než vzorkovací frekvenci, což při délce záznamu 8192 vzorků znamená asi 8 period. Výsledky jsou v tabulce 10.1 (f_{set} je frekvence nastavená na generátoru, f_{calc} je vypočtená frekvence).

Tabulka 10.1: Test výpočtu frekvence pro harmonický signál (neúspěšný)

f_s[kHz]	1	2	5	10	20	50	100	200	500	1000	2000	5000
f_{set}[Hz]	1	2	5	10	20	50	100	200	500	1000	2000	5000
f_{calc}[Hz]	0.977	1.953	4.883	9.766	19.53	48.83	97.66	195.3	488.3	976.6	1953	4883
$odchylka$[%]	2.34	2.35	2.34	2.34	2.35	2.34	2.34	2.35	2.34	2.34	2.35	2.34

Relativní odchylky jsou zjevně podezřele podobné. Nejprve jsem měl podezření na nepřesné časování vzorkování v mikrokontroleru, to jsem však vyloučil změřením periody obdélníkového signálu pomocí kurzorů (relativní chyba byla v řádu tisícin). Příčinou se ukázalo být diskrétní spektrum. Rozlišení

ve spektru je:

$$\Delta f = \frac{f_s}{N_{FFT}} \tag{10.4}$$

Pro 8192 je rozlišení ve spektru $\frac{f_s}{8192}$, frekvenci jsem nastavoval $\frac{f_s}{1000}$, relativní chyba tedy může být až

$$\Delta f = \frac{1000 \cdot f}{8192} \implies \frac{\Delta f}{f} = \frac{1000}{8192} = 0.12 = 12\%, \tag{10.5}$$

to je zcela nepřijatelné.

10.1.3 Vylepšení metody

Předchozí postup zjevně selhává pro frekvence, které jsou řádově nižší než vzorkovací a tedy mají málo period na změřeném úseku. Možné řešení by bylo doplnit signál nulami pro zlepšení rozlišení (to již bylo zmíněno, ale doplňoval jsem jen na minimální délku 4096, což bylo jen zhruba odhadnuté číslo, o kterém jsem myslel, že „musí stačit“ a nebylo podloženo žádným výpočtem). Pokud bych požadoval odchylku v řádu desetin procenta, musel by počet bodu FFT být pro tento signál (který měl 8 period v záznamu, což považuji za dostatečný počet pro přesný výpočet) být stonásobně vyšší.

Je zřejmé, že zatímco použitá metoda funguje dobře pro vysoké frekvence (velký počet period v záznamu), pro nižší frekvence je vhodnější použít postup s autokorelační funkcí. V případě nižších frekvencí je tedy nutné použít autokorelaci, je však potřeba vyřešit otázku hledání lokálního maxima. Vymyslel jsem postup, který kombinuje obě metody:

1. Předchozím postupem (ze spektra) vypočtu frekvenci jakožto hrubý odhad.
2. Podle tohoto odhadu vypočtu autokorelační funkci na okolí předpokládané polohy lokálního maxima.
3. Zjistím globální maximum na tomto úseku autokorelační funkce.

Nejprve je potřeba určit nutný počet bodů FFT pro dostatečně přesný odhad. Jako rozumný cíl zvolím maximálně 10% odchylku pro pouhé dvě periody na záznamu, tedy $\frac{f_s}{f} = \frac{N}{2}$.

$$\frac{\Delta f}{f} = \frac{N}{2 \cdot N_{FFT}} \leqq 10\% \implies N_{FFT} \geqq 5 \cdot N \tag{10.6}$$

Je tedy potřeba doplnit nulami na alespoň pětinásobek původní délky.

Nyní je třeba určit, jestli je výhodné hodnotu upřesnit pomocí autokorelační funkce, nebo jestli pouhé určení ze spektra poskytuje lepší přesnost. Autokorelace má rozlišení $\Delta T = \frac{1}{f_s}$ a tedy pro relativní odchylku platí:

$$\frac{\Delta T}{T} = \frac{1}{f_s \cdot T} \tag{10.7}$$

$$\frac{\Delta f}{f} = -\frac{f}{f_s + f} \tag{10.8}$$

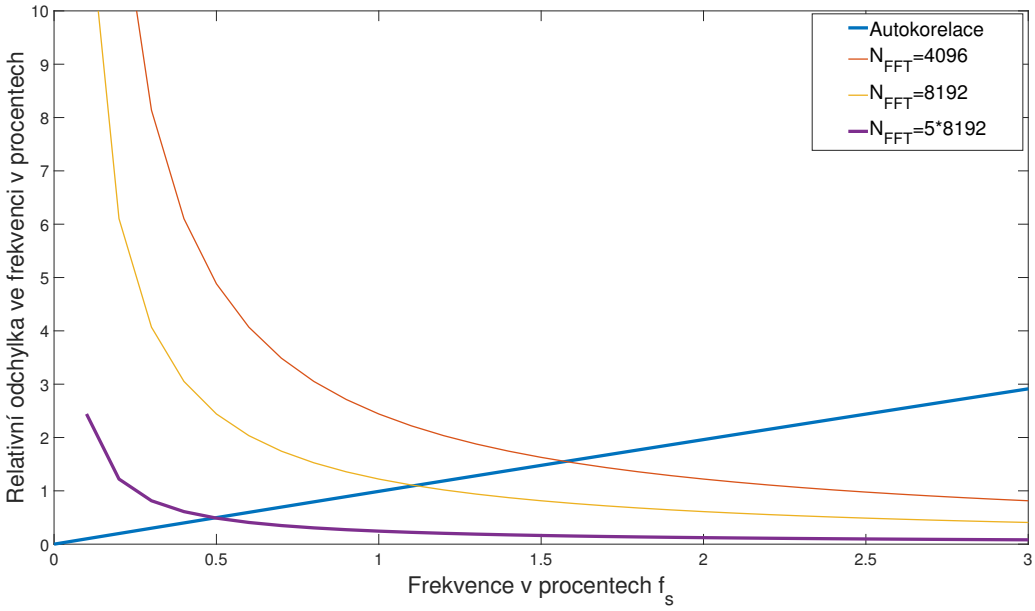
Ze vstahů 10.4 a 10.8 lze určit při jaké frekvenci nastává zlom mezi tím kdy se vyplatí autokorelace a kdy je výhodnější použít jen spektrum.

$$|\frac{f}{f_s + f}| = |\frac{f_s}{N_{FFT} \cdot f}| \tag{10.9}$$

z čehož lze vyjádřit f a tím určit, že postup s autokorelací se vyplatí, pokud

$$f < \frac{f_s(1 + \sqrt{4 \cdot N_{FFT} + 1})}{2 \cdot N_{FFT}}. \tag{10.10}$$

Pokud bych například měl $N = 8192$ a použil $N_{FFT} = 5 \cdot N$, byla by hranice přibližně $\frac{5}{1000} f_s$. To je znázorněno v grafu 10.2, kde je navíc dokresleno, jak by odchylka ve spektru vypadala pro nižší hodnoty N_{FFT}. Tento výpočet je založen na předpokladu, že odchylka ve výpočtu pochází pouze



Obrázek 10.2: Porovnání relativní odchylky dané rozlišením při výpočtu frekvence

z omezeného rozlišení ve spektru a v autokorelační funkci, což nemusí nutně být pravda. Dle grafu 10.2 lze očekávat odchylku maximálně půl procenta. Kvůli dalším zdrojům chyb může být vyšší.

10.1.4 Konečná verze algoritmu

Nejprve od signálu odečtu stejnosměrnou složku. Poté doplním nulami na pětinásobek původní délky zaokrouhlený na nejbližší vyšší mocninu dvou. Spočítám spektrum (FFT) a najdu jeho globální maximum. Pokud signál nesplňuje podmínku 10.10, je tento výsledek konečný. V opačném případě je spočítána autokorelační funkce pro posuny v rozsahu ±10% okolo hodnoty odpovídající frekvenci odhadnuté v předchozím kroku. Na tomto úseku naleznu globální maximum (vzhledem k celé autokorelační funkci jde o lokální maximum, ale na tom úseku je globální) a přepočítám jej na výslednou frekvenci.

Zopakoval jsem měření stejným způsobem jako předtím (předchozí měření: tabulka 10.1). Jak je vidět v tabulce 10.2, výsledky byly v tomto případě přesné (až na drobné odchylky ve vysokých vzorkovacích frekvencích, které jsou velmi malé a mohou být způsobeny i nedokonalostmi osciloskopu, ne výpočtem jako takovým)

Tabulka 10.2: Test výpočtu frekvence pro harmonický signál (úspěšný)

f_s[kHz]	1	2	5	10	20	50	100	200	500	1000	2000	5000
f_{set}[Hz]	1	2	5	10	20	50	100	200	500	1000	2000	5000
f_{calc}[Hz]	1.000	2.000	5.000	10.00	20.00	50.00	100.0	200.0	500.0	999.0	2000	5005
$odchylka$[%]	0	0	0	0	0	0	0	0	0	0.1	0	0.1

10.2 Vzestupná a sestupná hrana

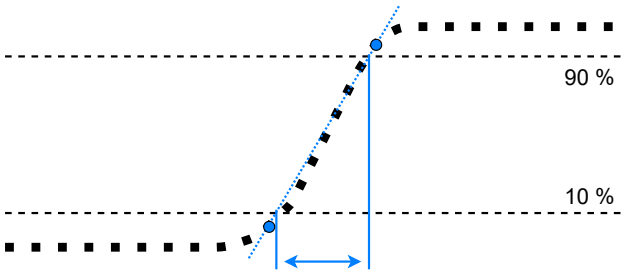
Doba vzestupné či sestupné hrany je definovaná jako čas za který se úroveň signálu změní z 10 % na 90 % vysoké úrovně (nebo opačně pro sestupnou).

10.2.1 Postup výpočtu

Při detekci vzestupné hrany postupuji následovně: Signál je procházen po jednotlivých vzorcích. Dokud jsou hodnoty pod 10 % vysoké úrovně, program si uloží vždy poslední zkontrolovaný vzorek, pokud je následující vzorek nad 10% úrovní, minulý vzorek se již nepřepíše a program si pamatuje poslední vzorek pod 10% úrovní. Když signál vystoupá nad 90% úroveň (a již byl nalezen vzorek pod 10 %), je tento vzorek použit jako konec vzestupné hrany.

Rozdílem těchto dvou vzorků lze dobu vzestupu odhadnout, ale ne příliš přesně, protože vzorky neleží přesně na 10% a 90% úrovni. Proto tyto dva body proložím přímkou a najdu její průsečíky s 10% a 90% úrovněmi, jak je znázorněno na obrázku 10.3. Pokud je vzestup a sestup tak rychlý, že nastane mezi dvěma vzorky, je uživateli hodnota zobrazena jako „menší než“.

Při výpočtu program postupuje od konce signálu (tedy se vždy počítá poslední hrana). Minimální a maximální úroveň je zjištěna jednoduše jako globální extrémy průběhu na úseku v kterém je hrana hledána (poslední dvě periody).

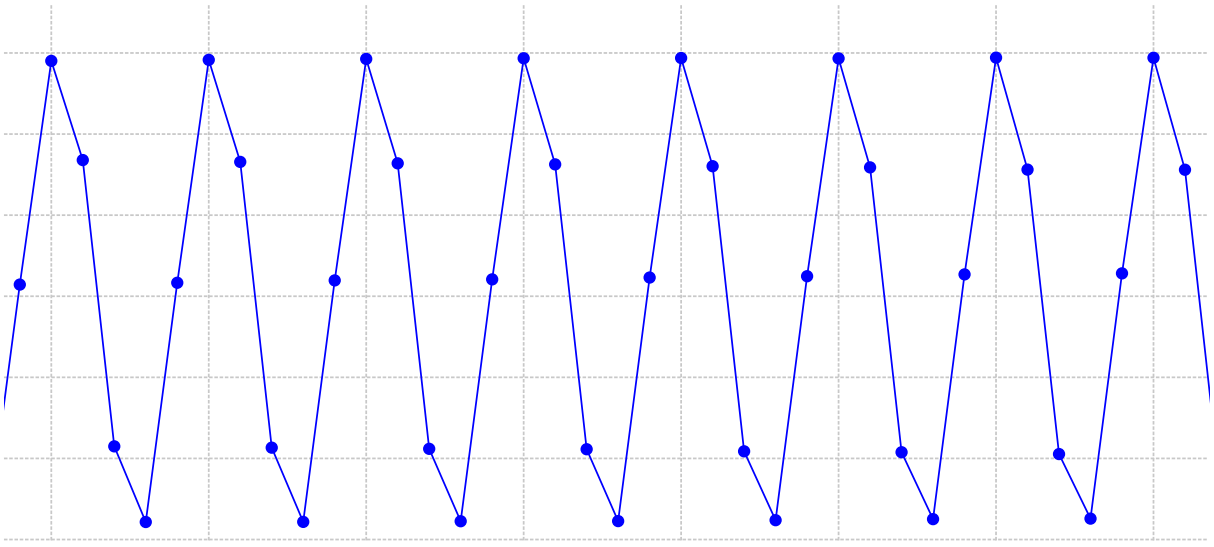


Obrázek 10.3: Výpočet vzestupné hrany signálu

Kapitola 11

Interpolace signálu

Pro přesné zaznamenání harmonického signálu stačí mít dva vzorky na periodu (Vzorkovací podmínka, viz 9). V grafu je však pro vykreslení průběhu použito prosté spojení vzorků rovnými čarami. To je použitelné při velkém počtu vzorků na periodu (alespoň 30 aby tvarem připomínal funkci sinus). Aby bylo možné zobrazit hladký sinus i pro řádově jednotky vzorků na periodu, je potřeba mezi ně doplnit a dopočítat další body, tedy je potřeba signál převzorkovat a vyhladit.



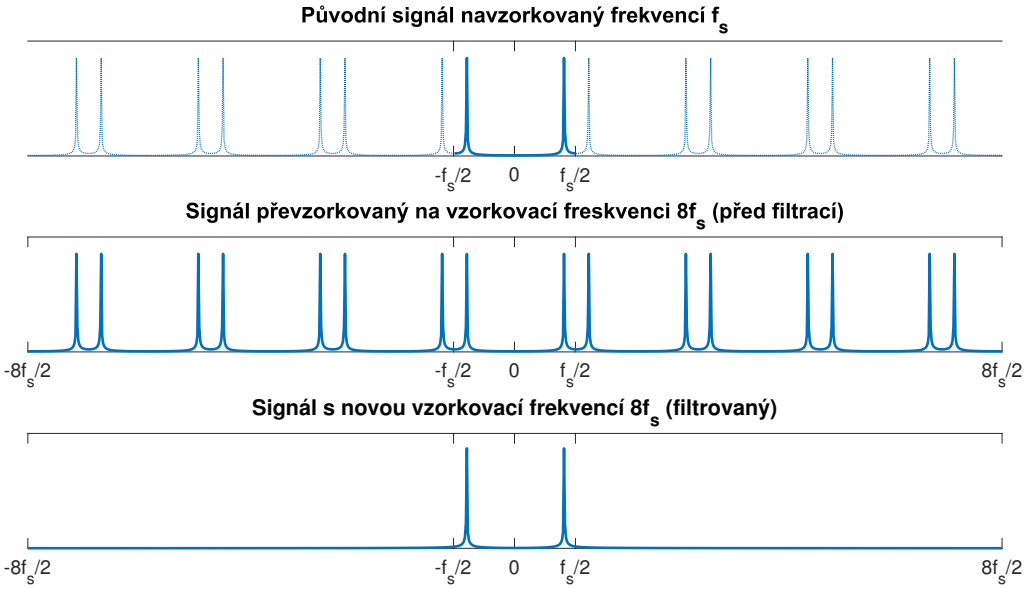
Obrázek 11.1: Harmonický signál s malým počtem vzorků na periodu

Postup interpolace spočívá v převzorkování signálu na vyšší vzorkovací frekvenci a následné filtraci dolnopropustným filtrem. Postup znám z předmětu Číslicové zpracování signálů a dále je popsán kupříkladu v článku [16].

11.1 Převzorkování

Zvýšení vzorkovací frekvence lze provést jednoduše, stačí mezi existující vzorky vložit další vzorky s nulovou hodnotou. Rozhodl jsem se vzorkovací frekvenci zvyšovat osmkrát[1], takže mezi každé dva vzorky vložím sedm dalších s nulovou hodnotou.

Přidání nul do signálu nevnáší žádnou novou informaci, a tedy nezpůsobí změnu spektra, pouze se zvýší vzorkovací frekvence f_s a tím se v rozsahu 0 až f_s objeví více periodických zopakování spektra, ty je potřeba odstranit (filtrovat dolní propustí), aby v intervalu od nuly do nové vzorkovací frekvence vyskytovaly pouze frekvenční složky původního signálu (frekvence nižší než polovina původní vzorkovací frekvence). Postup je zobrazen na obrázku 11.2.



Obrázek 11.2: Postup interpolace signálu

11.2 Filtrace dolní propustí

Je potřeba odstranit frekvenční složky vyšší než polovina původní vzorkovací frekvence, což je šestnáctina nové vzorkovací frekvence. K tomu použiji filtraci v časové oblasti. V číslicové filtraci rozlišujeme dva druhy filtrů: IIR filtry a FIR filtry.

Filtry s konečnou impulsovou odezvou (FIR) jsou realizovatelné pouze v číslicové filtraci (nemají analogovou obdobu) a mají tyto výhody:

- Je zaručena stabilita (protože nemají zpětnou vazbu)

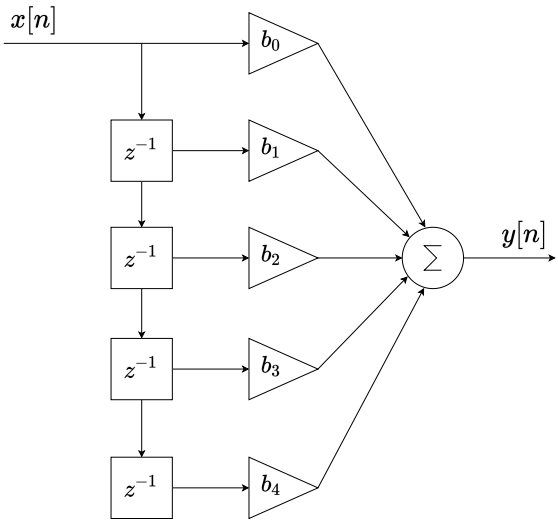
[1] Dodatečně jsem doplnil možnost zvolit i 16× a 32×.

- Lineární fázová charakteristika v propustném pásmu (všechny frekvence se zpožďují o stejný čas, nedochází ke zkreslení)

Nevýhodou je větší řád (počet koeficientů) oproti IIR se srovnatelnými vlastnostmi a s tím spojené větší zpoždění. Zpoždění pro můj účel není nijak zásadní, pouze způsobí, že bude chybět několik vzorků na začátku a konci interpolovaného signálu. Rozhodl jsem se použít filtr typu FIR.

11.2.1 FIR filtr

Filtry s konečnou impulsovou odezvou nemají zpětnou vazbu, výstup je pouze lineární kombinace zpožděných verzí vstupního signálu, jak je znázorněno na obrázku 11.3. Z toho vyplývá, že pokud je na vstup přiveden jednotkový impulz (Kronekerova delta), na výstupu se objeví několik zpožděných jednotkových impulzů, násobených koeficienty pro příslušné zpoždění. Impulsová odezva je tedy totožná se seznamem koeficientů. Řád filtru je roven nejvyššímu použitému zpoždění, koeficientů je tedy o jeden více než je řád.



Obrázek 11.3: FIR filtr

Nejjednodušší variantou FIR dolní propusti je klouzaví průměr, tedy součet několika vzorků vydělený jejich počtem. To odpovídá konvoluci signálu s obdélníkem o $M + 1$ vzorcích a výšce $\frac{1}{M+1}$ (M je řád filtru, o jedna menší než počet koeficientů). Konvoluce s obdélníkem v čase se ve spektru projeví jako násobení funkcí sinc, která má tvarem hodně daleko k ideální (obdélníkové) charakteristice filtru. Sice potlačí vyšší kmitočty, ale potlačení není příliš výrazné a naopak dochází k nežádoucímu potlačení v propustném pásmu.

Jednou z metod návrhu FIR filtrů je metoda okna. Jak je vysvětleno v článku [17]: Ideální filtr by ve frekvenční oblasti měl tvar obdélníku (v propustném pásmu 1, mimo něj 0). To potřebujeme převézt do časové oblasti (impulzní odezva), což samozřejmě vede na funkci sinc. Ta je nenulová na nekonečné délce. Aby byl filtr použitelný, je potřeba tuto odezvu filtru oříznout. Oříznutí vede na prosakování

ve spektru, proto je vhodné oříznutí provézt váhovacím oknem (viz část 9.3). Výhodné je použití parametrizovaného okna (například Kaiserovo), jehož tvar lze optimalizovat pomocí parametru.

11.2.2 Návrh filtru pro interpolaci

Filtr jsem navrhl s pomocí programu Matlab. Pro vytvoření FIR filtru lze použít funkci `fir1`, minimální parametry pro tuto funkci jsou řád filtru a zlomová frekvence[2]. V takovém případě je navržena dolní propust s pomocí Hammingova okna[18]. Další možností je do funkce vložit jiné okno, například Keiserovo. To lze vypočítat z tolerančního pole funkcí `kaiserord`.

Pokud převzorkovávám na osminásobek původní vzorkovací frekvence, pak zlomová frekvence bude $\frac{1}{16}f_s$. Normovaná frekvence je tedy $\frac{1}{8} = 0.125$. Z toho je také vidět, že filtr není závislý na vzorkovací frekvenci, pouze na poměru zlomové a vzorkovací, proto stačí mít jeden filtr a bude fungovat se všemi signály (pro stejný násobek převzorkování).

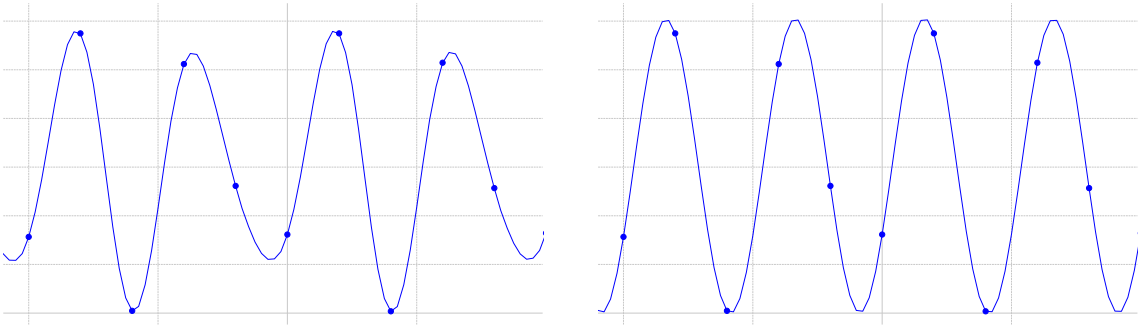
Výsledkem je vektor koeficientů b_n. Filtr má obecně přenosovou funkci tvaru

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3} + \dots}{a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} + \dots}, \tag{11.1}$$

pro FIR filtr je však $a_0 = 1$ a další koeficienty a_n jsou nulové (nemá zpětnou vazbu).

Nejprve jsem navrhl filtr s pomocí Hammingova okna s řádem 50, funkčnost byla dostatečná, ale s viditelnými nedokonalostmi. Poté jsem navrhl další filtr s pomocí funkce `kaiserord`, ten je viditelně lepší, ale řád je výrazně větší (194). Tento druhý filtr jsem následně napočítal i pro převzorkování 16x a 32x, uživatel může zvolit jeden z těchto čtyř filtrů (ten první, řádu 50, může být výhodný na pomalém počítači, kdyby se program pro větší filtry zasekával. Převzorkování 32x by naopak obvykle bylo zbytečně velké).

Na další straně jsou vypsány použité příkazy v Matlabu a výsledky.



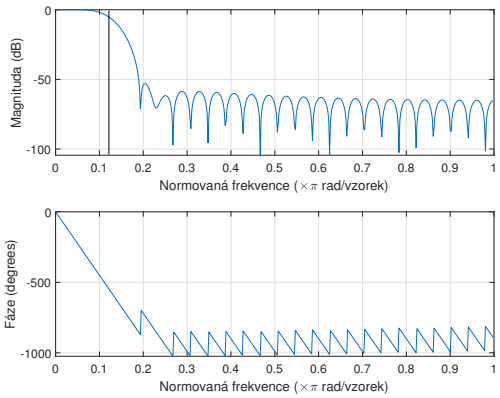
(a) : FIR řádu 50, Hammingovo okno **(b) :** FIR řádu 194, Kaiserovo okno

Obrázek 11.4: Interpolace harmonického průběhu o frekvenci 200 kHz, navzorkovaného frekvencí 500 kHz

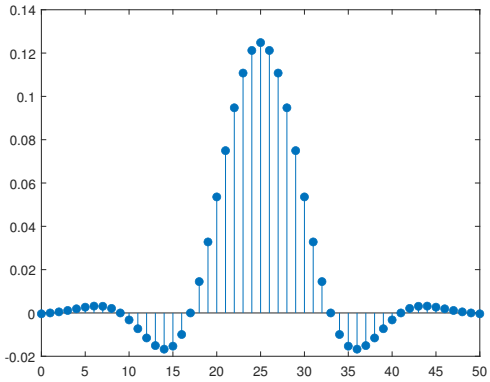
[2] Zadává se jako normovaná frekvence, což je hodnota v rozsahu 0 až 1, kde 1 odpovídá $\frac{f_s}{2}$

FIR řádu 50, Hammingovo okno

```
upsample = 8;
M=50;
b=fir1(M,1/upsample,'low');
```



(a) : Přenos

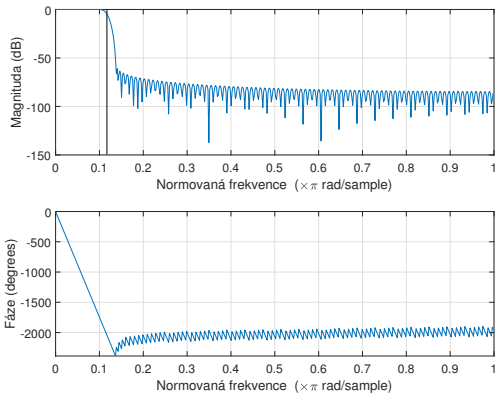


(b) : Impulzová odezva

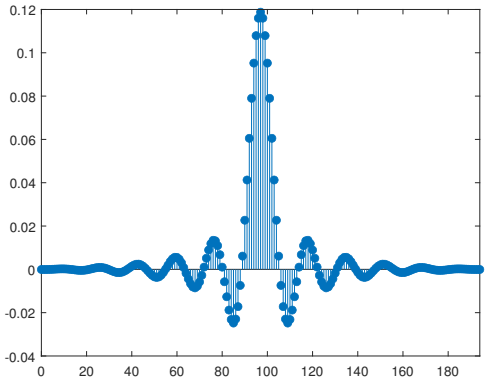
Obrázek 11.5: FIR řádu 50, Hammingovo okno

FIR řádu 194, Kaiserovo okno

```
fs=1000;
upsample = 8;
[M,Wn,beta,ftype]=kaiserord([fs/2*0.8,(fs/2)*1.1],[1 0],[0.05 0.001],fs*upsample);
b=fir1(M,Wn,ftype,kaiser(M+1,beta),'noscale');
```



(a) : Přenos

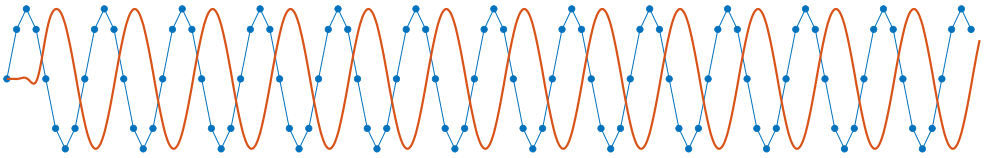


(b) : Impulzová odezva

Obrázek 11.6: FIR řádu 194, Kaiserovo okno

11.2.3 Zpoždění filtru

Aby byl filtr kauzální, musí hodnota v čase n používat pouze hodnoty z časů starších nebo rovných n. Výsledek filtrace FIR filtrem řádu M je oproti původnímu signálu zpožděn o $\frac{M}{2}$ vzorků. To je potřeba kompenzovat, aby interpolovaný graf odpovídal původním vzorkům. Z grafu 11.7 je také vidět, že na začátku jsou vzorky zkreslené přechodovým jevem (filtr potřebuje vzorky ze záporných časů které neexistují a pro účel výpočtu jsou považovány za nulové).



Obrázek 11.7: Filtarace FIR filtrem

11.2.4 Implementace filtrace v kódu

Filtrace se provádí konvolucí filtrovaného signálu s impulzovou odezvou filtru. Konvoluce diskrétních signálů je definována:

$$y[k] = \sum_{n=-\infty}^{\infty} h[n] \cdot x[k-n] \tag{11.2}$$

Protože signál x i impulsová odezva filtru h jsou konečné délky, meze sumy lze zmenšit tak že se vynechají hodnoty k kdy se x a y nepřekrývají (tam je hodnota nulová). Také je vhodné nepočítat začátek a konec, který je zatížený přechodovým jevem, protože vzorky mimo změřený úsek jsou nahrazeny nulovou hodnotou. Odebrání přechodového jevu na začátku (M vzorků) způsobí, že výsledek se naopak předbíhá o $\frac{M}{2}$ vzorků oproti původnímu. Korektní výsledek vypadá tak, že na začátku i konci chybí $\frac{M}{2}$ vzorků, protože ty už by potřebovali vzorky které jsou mimo rozsah a tedy by byly zkresleny.

Kód pro výpočet konvoluce (bez zkresleného začátku a konce)

```
1  QVector<float> Interpolator::filter(QVector<float> x, QVector<float> h) {
2    QVector<float> y;
3    int N = x.length();      // Délka signálu
4    int M = h.length();      // Délka odezvy filtru
5    y.resize(N - M + 1);
6    for (int n = M - 1; n < N; n++) {
7      for (int k = 0; k < M; k++) {
8        y[n - M + 1] += x[n - k] * h[k];
9      }
10   }
11   return y;
12 }
```

11.3 Začlenění funkce do programu

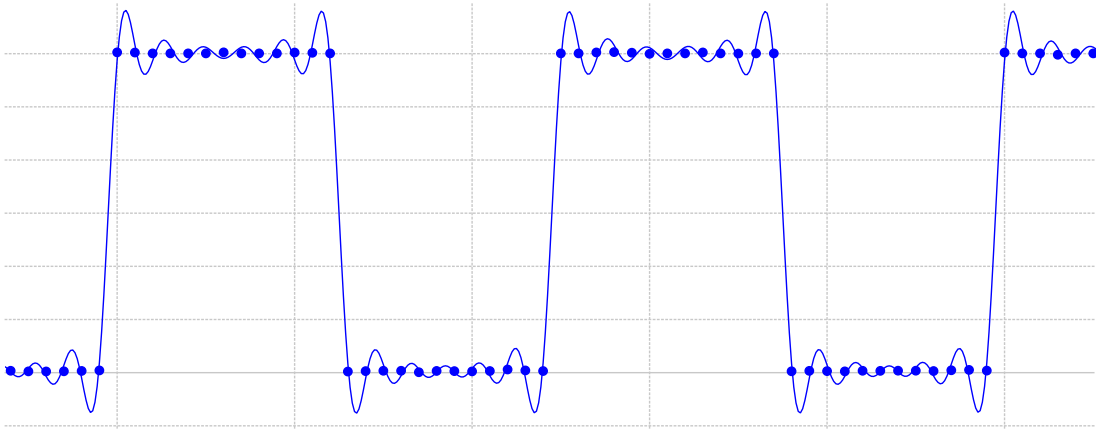
Pro ušetření výpočetního výkonu je interpolace počítána jen pro viditelný rozsah časů (ne pro celý signál). Pro výpočet je potřeba vzít i několik (alespoň $\frac{M}{2}$) vzorků mimo tento rozsah na každé straně, aby měl výsledek hodnoty na celém zobrazeném intervalu, navíc je vhodné nechat větší rezervu, protože při posunu grafu do stran se interpolace musí přepočítat pro nový interval, což chvilku trvá a interpolovaný graf by po tu dobu nebyl kompletní. Rezerva po stranách je: $\frac{M}{2}$ vzorků, nebo 50 % viditelného intervalu (podle toho co je vyšší).

Interpolovaný průběh se v grafu vykresluje jako samostatný průběh, který se překrývá s původním, je tak možné zároveň s interpolovaným průběhem zobrazit i původní vzorky.

11.4 Chovaní v případě neharmonických signálů

Jakýkoli signál lze reprezentovat jako součet harmonických průběhů (Fourierova řada). Pro nekonečně velký počet členů řady se řada rovná původnímu signálu. Pokud jsou vzorky spojeny rovnými čarami (bez interpolace), není problém vykreslit obdélníkový signál. Přímka spojující vzorky může být libovolně strmá a nemá tedy problém napodobit velmi strmou hranu obdélníkového signálu.

Interpolace se však chová jako proložení původních vzorků kombinací harmonických složek o maximální frekvenci odpovídající polovině původní vzorkovací frekvence. Omezená frekvence omezuje schopnost napodobit ostré hrany, proto se na hranách obdélníkového signálu vyskytuje takzvaný Gibbsův jev: překmit, jehož velikost navíc neklesá se zvyšujícím se počtem členů Fourierovi řady.



Obrázek 11.8: Gibsuv jev na obdélníkovém signálu

Kapitola 12

Terminál

Ve své aplikaci jsem vytvořil terminál, do kterého lze vypisovat text s využitím ANSI escape sekvencí, podobně jako funguje například PuTTY. Jak již bylo zmíněno v rozboru zadání, nepodařilo se mi sehnat již hotový prvek grafického rozhraní (takzvaný QWidget) který by toto umožňoval, proto jsem terminál sám vytvořil.

Nejprve jsem se pokusil ho vytvořit na základě textového pole, ale narazil jsem na mnohá omezení (zejména nemožnost přemístit kursor na prázdné místo, kde ještě není žádný text). Nakonec jsem vytvořil terminál založený na tabulce (prvek QTableWidget), kde v každém poli tabulky je jeden znak. Tento postup se ukázal být velmi výhodný. Jedinou nedokonalostí je, že pokud se vypisují znaky s podtržením, tak je čára podtržení přerušena na hranici políček, není tedy možné spojitě podtrhnout slovo.

Je možné zvolit ze dvou velikostí písma (12 nebo 18 bodů). Při minimální šířce pole terminálu se do něj na šířku vejde přesně 14 znaků při velkém písmu, nebo 21 při malém písmu, v případě potřeby lze terminál zvětšit (posunutím hranice mezi pravým panelem nástrojů a grafem). Výška záleží na velikosti okna a v terminálu lze kolečkem myši posouvat zobrazení nahoru a dolů.

12.1 Podporované znaky a sekvence

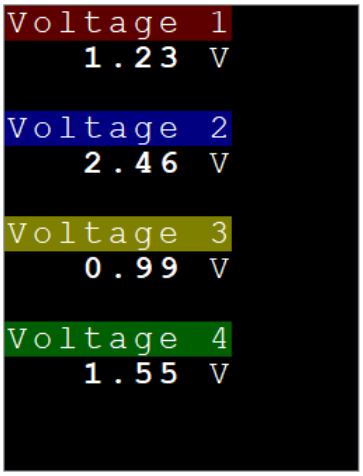
Terminál samozřejmě podporuje všechny znaky z ASCII, kromě toho umožňuje vypsat i znaky v UTF-8.

Speciální znaky

Kromě obyčejných písmen a číslic jsou podporovány tyto znaky:

- Nový řádek (line feed, `\n`) posune kurzor na další řádek, ale na stejném sloupci jako byl původně (neposune ho na začátek řádku).

- Návrat na začátek řádku (carriage return, `\r`).
- Tabulátor (horizontal tab, `\t`) posune kursor na nejbližší další pozici (sloupec) která je násobkem osmi.
- Zvonek (bell, `\a`) vyvolá systémový zvuk upozornění (pomocí `QApplication::beep()`).



Obrázek 12.1: Zobrazení textu v terminálu

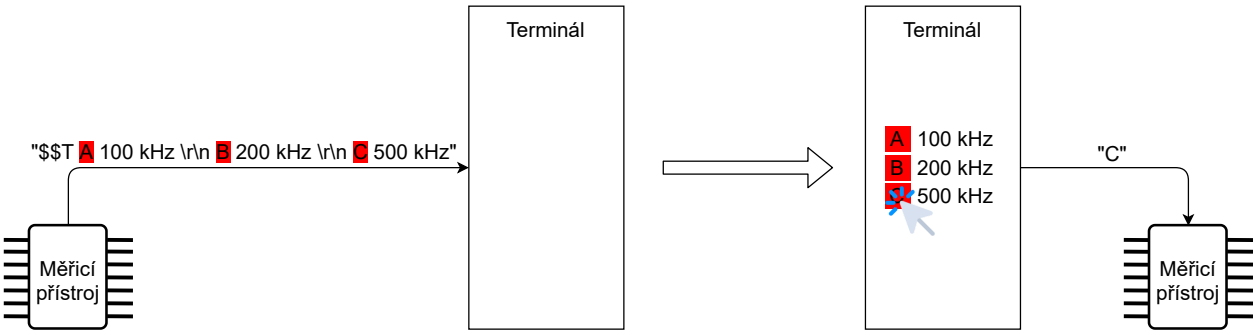
12.2 Interaktivní menu pro ovládání zařízení

Základním způsobem ovládání zařízení (mikrokontroleru) prostřednictvím aplikace je zadání textu do textového pole a odeslání. Tento způsob však není příliš pohodlný pro uživatele, lepší by bylo umožnit ovládání například s pomocí tlačítek.

Tlačítka pro ovládání v GUI aplikace by však byla docela v rozporu s požadovanou univerzálností aplikace. Možné řešení by bylo navrhnout způsob, jak pomocí příkazů dynamicky vytvářet tlačítka a jiné prvky (pole pro nastavení čísla a podobně) a nastavit jim příkazy které by odeslaly po kliknutí či změně hodnoty. Toto by však bylo velmi složité implementovat jak v aplikaci, tak i ve firmware pro mikrokontroler.

Nakonec jsem vymyslel velmi jednoduchý, a přitom dobře fungující způsob, jak alespoň částečně dosáhnout možnosti ovládat zařízení klikáním na tlačítka. Podstata této funkce je velmi jednoduchá: pokud uživatel klikne na znak (písmeno) v terminálu, je tento znak odeslán do zařízení.

Použití je následující: zařízení bude ovládáno pomocí jednoduchých příkazů tvořených jen jedním znakem (například `+` pro zvýšení rozsahu a `-` pro snížení). Poté stačí do terminálu vypsat znaky `+` a `-` a formátovat je tak, aby působily dojmem tlačítka (například vybarvit pozadí). Uživatel poté může na tento znak kliknout a tím ho odeslat do mikrokontroleru, který ho zpracuje jako příkaz. Znak musí samozřejmě být pro každý příkaz jedinečný (nelze například použít `+` a `-` pro různá nastavení), pro větší množství nastavovaných hodnot tedy bude nutné použít i znaky které příliš



Obrázek 12.2: Interaktivní menu v terminálu

nevypovídají o svém účelu. Je však možné znak skrýt tím, že bude mít stejnou barvu jako pozadí, čímž vznikne zdánlivě prázdné tlačítko, ke kterému lze připsat popisek.

Je vhodné, aby uživatel měl zpětnou vazbu, že ke kliknutí došlo, proto se po kliknutí na zlomek sekundy změní barva pozadí znaku (ztmaví nebo zesvětlá, podle toho, jestli byl původně spíše světlý či tmavý).

Kliknout lze samozřejmě i na jiná písmena než ta, která jsou zamýšlena jako tlačítka. Tomu nelze zcela zabránit, protože odlišení „tlačítek“ od obyčejných písmen by vyžadovalo komunikaci nad rámec obyčejného výpisu s pomocí ANSI sekvencí. Přidal jsem však možnost nastavit seznam výjimek: „blacklist“ barev pozadí znaku, pro které je odeslání zakázáno. Ve výchozím nastavení je takto znemožněno odeslat znaky s černým pozadím. Další zakázané barvy muže mikrokontroler přidat pomocí nastavovacího příkazu `$$Snoclickclr` následovaného seznamem ANSI kódů těchto barev (nemusí obsahovat `\e[` na začátku a `m` na konci). Toto nastavení přepíše původní seznam, je tedy potřeba odeslat všechny najednou. Drobným problémem je, že escape sekvence obsahují znak `;`, který je ale použit jako zakončení zprávy `$$S`, proto jsou středníky v kódu barvy pro tento účel nahrazeny tečkami. Výsledná podoba příkazu může vypadat například takto: `$$Snoclickclr:40,41.1,48.5.34;` (černá, červená a světle zelená).

12.2.1 Příklad ovládání přístroje

Při tvorbě osciloskopu (založeném na STM32 mikrokontroleru) jsem navrhl rozhraní zobrazené na obrázku 12.3a.

- Pro nastavení úrovně triggeru jsou použity znaky `+` a `-`, které zjevně vypovídají o svém účelu.
- Pro nastavení hrany triggeru jsou použity znaky lomítko (tvarem připomínající vzestupnou hranu) a zpětné lomítko. Zobrazený znak ukazuje aktuální stav, pochopení příkazů v zařízení je tedy samozřejmě opačné: kliknu-li na dopředné lomítko, chci nastavení změnit na klesající hranu a vice versa.
- V nastavení pretriggeru jsou použity znaky `<` a `>`, vypadající jako šipky.

(a) : Základní nastavení osciloskopu **(b) :** Zadávání číselně hodnoty

Obrázek 12.3: Interaktivní menu v terminálu

- Výběr hodnoty ze seznamu (vzorkovací frekvence, kanál triggeru) je udělán tak, že vlevo vedle seznamu jsou čísla nebo písmena ukrytá (barva písma stejná jako pozadí) v bílých políčcích, aktuálně zvolená možnost má políčko vybarvené oranžově. Je vhodné seřadit čísla či písmena v pořadí v jakém jsou v ASCII, aby bylo snadné je v mikrokontroleru přepočítat na index zvolené hodnoty.

- Tlačítka (v tomto případě `Run`, `Pause` a `Single-trigger`) jsou jednoduše písmeno s barevným pozadím.

Výběr frekvence ze seznamu je dostačující pro volbu vzorkovací frekvence, avšak v případě generování signálu je vhodné umožnit mnohem přesnější volbu. Jednou z možností (zobrazená na obrázku 12.3b) je vytvoření „klávesnice“ ze znaků `0` až `9`. S ohledem na větší přehlednost a zejména na to, že každý znak lze pro „tlačítko“ použít je jednou, je vhodné terminál pomyslně rozdělit na více stránek.

Příklad: V základním menu osciloskopu je uvedena nastavená frekvence PWM generátoru a vedle ní je tlačítko pro její nastavení. Po kliknutí na tlačítko mikrokontroler reaguje vymazáním terminálu a vykreslením stránky pro zadání frekvence. Protože mikrokontroler „si pamatuje“, že je nyní v režimu zadávání frekvence, zpracovává přijaté znaky (číslice) jinak, než jaký význam měly v základním menu. Po zadání hodnoty a potvrzení mikrokontroler opět vymaže terminál a vykreslí původní menu.

12.3 Kopírování textu z terminálu

QTableWidget, na kterém je terminál založen, umožňuje označit skupinu políček (v tomto případě jednotlivých znaků), toho lze využít pro zkopírování textu do schránky. Jednoduše vytvořím textový řetězec, do kterého přidám postupně všechny znaky z vybraných políček. Označit lze více řádků, v takové případě přidám na konec řádku znak `\n`.

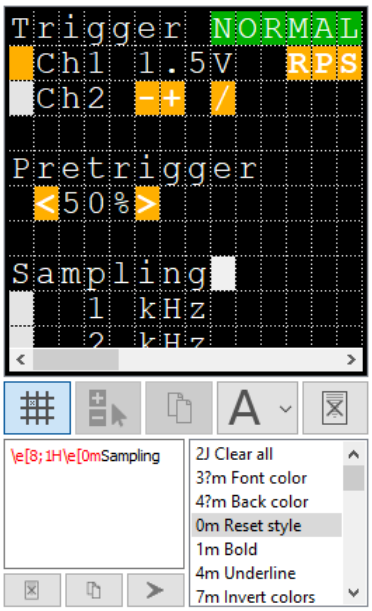
Umístění textu do schránky lze snadno provézt pomocí kódu:

```
1     QClipboard* clipboard = QGuiApplication::clipboard();
2     clipboard->setText(text);
```

Užitečným vedlejším efektem označování textu je, že pokud je takto označeno políčko, v kterém je skrytý znak (barva písma stejná jako pozadí), tak se vlivem změny barev označením tento znak odkryje. Toho lze využít při odladňování.

12.4 Režim pro návrh terminálu

Vytvořit pseudografické uživatelské rozhraní v terminálu může být obtížné. Uživatel by musel znát (nebo mít po ruce) seznam escape sekvencí a každá změna či oprava chyby by vyžadovala znovu zkompilovat a nahrát firmware pro mikrokontroler. Z toho důvodu jsem do aplikace zahrnul i návrhový režim, v kterém lze do textového pole zadat text (včetně escape kódů) a následně ho vypsat do terminálu.



Obrázek 12.4: Režim pro návrh terminálu

Uživatel tak může snadno pozměňovat text a ihned vidí výsledek v terminálu. Escape sekvence lze vkládat ze seznamu, není tedy potřeba je znát či hledat. Pokud je ze seznamu vložena změna barvy textu nebo pozadí, je zobrazen dialog pro výběr barvy[1]. V návrhovém režimu se v terminálu zobrazí mřížka řádků a sloupců a vybarví se pozice kurzoru. Kliknutím do kteréhokoli místa se do textového pole s návrhem kódu vloží příkaz pro přesun kurzoru na danou pozici.

Zkopírováni textu pomocí tlačítka kromě prostého zkopírování textu z textového pole také nahradí odřádkování znaky `\r\n` a znaky, které nejsou v ASCII, nahradí příslušným kódem v UTF-8 (například `\xc3""\xa1""` pro „á“), takže lze text přímo vložit do kódu.

[1] Terminál podporuje jen sadu barev xterm-256, ne celé obvyklé RGB, pokud je zvolena barva, kterou terminál nepodporuje, je tato barva „zaokrouhlena“ na nejbližší podporovanou (blízkost je posuzována jako součet druhých mocnin rozdílů složek R, G a B).

Kapitola 13

Export dat

Funkce uložení grafu do souboru umožní data dále zpracovat například v Excelu, případně vložit do laboratorního protokolu.

13.1 Export kanálů do tabulky

Pro jednoduché ukládání dat se používá formát souboru `.csv`, název „comma separated values“ značí, že hodnoty jsou odděleny čárkou (Zkratka se někdy interpretuje jako „character separated values“, protože se nemusí nutně jednat o čárku). Toto oddělení čárkou (či jiným znakem) se týká hodnot v řádku, jednotlivé řádky jsou odděleny znakem `\n` (nový řádek).

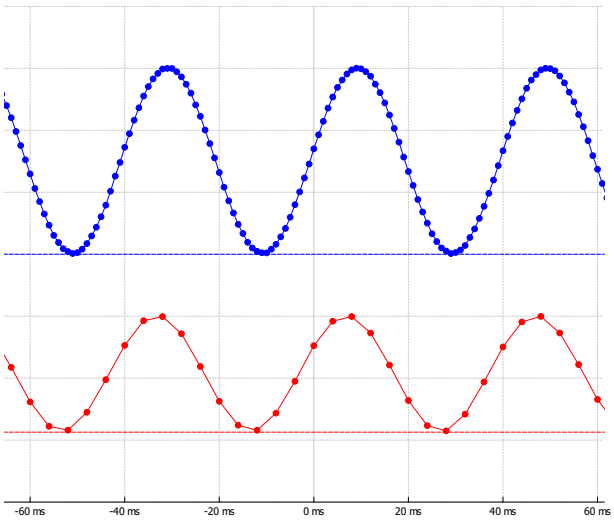
Při otvírání souboru v Excelu je nutné brát zřetel na to, že Excel může vyžadovat buďto desetinnou čárku, nebo tečku, v závislosti na nastavení jazyka. Obvyklí způsob je použití desetinné tečky, pokud je však Excel v češtině, je potřeba použít desetinnou čárku. To samozřejmě vylučuje použití čárky pro oddělení hodnot, v tomto případě tedy použiji středník.

Exportovat lze samostatný kanál, nebo všechny kanály současně. V případě exportu všech kanálů je vytvořena tabulka, v které je jen jeden sloupec časů (společný pro všechny kanály). Algoritmus pro export zohledňuje i situaci, kdy kanály mají vzorky v různých časech: například pokud mají kanály různou vzorkovací frekvenci, jak je znázorněno na obrázku 13.1

13.1.1 Kopírování dat do schránky

Pro přenesení dat do Excelu či podobného programu může být rychlejší a pohodlnější data zkopírovat do schránky a poté vložit do tabulky. To umožní vložit data do již existujícího souboru a i pro založení nového souboru to může být rychlejší, než ukládat a otevírat CSV soubor.

Způsob uložení tabulky ve schránce se podobá CSV souboru s tím rozdílem, že hodnoty v řádku jsou odděleny tabulátorem (`\t`). To jsem zjiztil z [19].

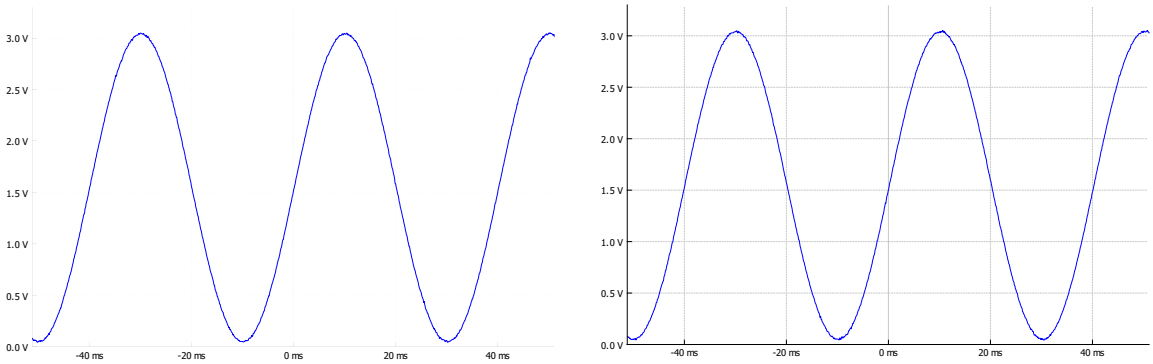


time	Ch 1	Ch 2
0.000	1.702	1.395
0.001	1.928	
0.002	2.145	
0.003	2.358	
0.004	2.538	1.79
0.005	2.693	
0.006	2.813	
0.007	2.909	
0.008	2.975	1.866
0.009	3.003	
0.010	2.993	
0.011	2.946	
0.012	2.874	1.6
0.013	2.745	
0.014	2.608	
0.015	2.443	
0.016	2.247	1.083
0.017	2.031	
0.018	1.81	
0.019	1.569	
0.020	1.335	0.512
0.021	1.111	
0.022	0.883	
0.023	0.682	

Obrázek 13.1: Export dvou kanálů s různou vzorkovací frekvencí

13.2 Export obrázku

Graf QCustomPlot nabízí funkce pro export v podobě rastrové grafiky ve formátech PNG, BMP a JPEG. Kromě toho nabízí i funkci funkci exportu do vektorové grafiky formátu PDF, což jsem využil i pro obrázky v této práci. Drobným nedostatkem exportu do PDF je, že mřížka na pozadí se vykresluje velmi tenkou čarou, takže v tištěné podobě není zřetelná. Tento problém lze však vyřešit použitím parametru `QCP::epNoCosmetic` při volání funkce `QCustomPlot::savePdf`.



(a) : Výchozí nastavení funkce `savePdf`

(b) : S parametrem `QCP::epNoCosmetic`

Obrázek 13.2: Export grafu do PDF

Kapitola 14

Příprava programu pro distribuci

Kromě samotného spustitelného souboru vzniklého kompilací kódu potřebuje aplikace k běhu také knihovny Qt. Tyto knihovny je nutné přidat do adresáře se spustitelným souborem[1]. Přidání knihoven do adresáře je možné provézt automaticky pomocí nástroje `windeployqt` který je součástí instalace Qt. Pro Linux existuje nástroj `linuxdeployqt` [21].

14.1 Distribuce programu pro systém Windows

Celý vývoj aplikace jsem prováděl na počítači se systémem Windows 10. Kompilaci je možné provádět s pomocí kompilátorů MSVC (Microsoft Visual C++) nebo MinGW (GNU Compiler Collection portovaný pro Windows). Ze zkušeností při testování vyplynulo, že program zkompilovaný s MSVC funguje lépe (z hlediska výpočetního výkonu) a proto ho upřednostňuji. Pro spuštění programu zkompilovaného kompilátorem MSCV je potřeba mít v počítači nainstalované knihovny Microsoft Visual C++ Redistributable (VC redist).

Pro použití ve školních učebnách je však požadována „portable“ verze programu, ke které není nutné nic doinstalovat. Instalaci je možné obejít tím, že se `dll` knihovny VC redist přidají přímo do adresáře se spustitelným souborem stejně jako knihovny Qt. Jedná se o soubory:

```
1 msvcp140.dll
2 msvcp140_1.dll
3 msvcp140_2.dll
4 vcruntime140.dll
5 vcruntime140_1.dll
```

Které lze nalézt (na počítači s nainstalovaným VC redist) ve složce `C:\Windows\System32`. Vytvořil jsem tedy i portable verzi, do které jsem překopíroval tyto knihovny.

Požadována je i kompatibilita se systémem Windows XP, ten však nejnovějšími verzemi Qt již

[1] Možné je i statické linkování kdy by knihovny byly zahrnuty v exe souboru, jak je popsáno v [20]. Je to však složitější a vývojář Qt od toho odrazuje.

není podporován. Podle [22] je poslední verzí fungující na Windows XP verze 5.7 v kombinaci s MinGW 5.3., kterou jsem tedy použil. Drobným nedostatkem této verze je, že v ní chybí adaptivní režim změny hodnoty `QDoubleSpinBox` (zmíněný v 4.2.1).

14.1.1 Automatické instalace programu

Aplikaci lze samozřejmě distribuovat jako složku se spustitelným souborem a dalšími potřebnými soubory, tedy jako takzvanou „portable“ verzi. To je pro mnoho uživatelů preferované volba, zejména pokud program plánují použít jednorázově. Považuji však za pohodlnější využít možnost instalace která kromě umístění potřebných souborů:

- Vytvoří zástupce v nabídce Start, případně na ploše
- Automaticky nainstaluje Microsoft Visual C++ Redistributable

Instalační balíček jsem vytvořil s pomocí nástroje Inno Setup [23]. Před automatickou instalací Microsoft Visual C++ Redistributable je vhodné zkontrolovat, zdali již není v počítači nainstalován. Postup kontroly přítomnosti VC Redist je popsán v článku [24]. Použitý skript je součástí příloh.

14.1.2 Chování na monitoru s vysokým DPI

Na monitoru s vysokým DPI (velké rozlišení při malé ploše) by okna aplikaci byla velmi malá a texty by mohly být nečitelné. Z toho důvodu systém Windows umožňuje prvky grafického rozhraní zvětšit. Aplikace k tomuto může přistupovat dvěma způsoby, jak je popsáno v [25].

- Aplikace je schopna sama zvětšit texty a ikony
- Aplikace se nechá zvětšit systémem (vykreslí se v nižším rozlišení a obraz je roztáhnut)

První možnost jistě může vézt k nejlepším výsledkům, ale navrhnout GUI tak, aby bylo zvětšení schopno je problematické a nepodařilo se mi dosáhnout přijatelné funkčnosti bez překrývání textu a podobných problémů. Vydal jsem se tedy druhou cestou a aplikaci nastavil jako `DPI unaware`. Nevýhodou je mírně rozostřený vzhled, ale zcela odpadají problémy s proporcemi GUI.

Předpokládám, že většina uživatelů zvětšení nepoužívá (zvětšení nastaveno na 100 %), takže dokonalá kompatibilita se zvětšením GUI není nijak zásadní. Nastavení do režimu `DPI unaware` zajistí, že v případě použití zvětšení se nevyskytnou žádné chyby ve vykreslování (překrývající se texty) a lehké rozostření na monitoru s vysokým DPI není nijak výrazné. Nastavení je provedeno tak, že do adresáře se spustitelným souborem je vložen soubor s názvem `qt.conf` obsahující text:

```
1 [Platforms]
2 WindowsArguments = dpiawareness=0
```

Pokud by uživatel chtěl vyzkoušet chování v režimu kdy texty v GUI zvětšuje samotná aplikace, může toto změnit na:

```
[Platforms]
WindowsArguments = dpiawareness=1
```

Dle testování se zdá že v tomto režimu se GUI vykresluje správně i při zvětšení nastaveném na 125 %, při 150 % již dochází k problémům.

14.2 Distribuce programu pro systém Linux

Vývojové prostředí QtCreator má i verzi pro Linux, proto stačí zdrojový kód aplikace zkopírovat na počítač se systémem Linux a zkompilovat na něm. Pro přidání potřebných knihoven jsem použil již zmíněný nástroj `linuxdeployqt` [21]. Tento nástroj zároveň umí program zabalit do jednoho spustitelného souboru AppImage. Pro nejlepší kompatibilitu s různými verzemi systému je doporučeno pro sestavení balíčku použít starší verzi operačního sytému [26]. Použil jsem Ubuntu 16.04.

Použití je složitější než v případě `windeployqt`, ale je dobře zdokumentováno [21]. Příkaz pro sestavení jsem použil s těmito parametry:

```
-appimage -always-overwrite -bundle-non-qt-libs -no-translations
-extra-plugins=iconengines,platformthemes/libqgtk3.so
```

Produktem je jednak přidání potřebných knihoven do složky programu (což je použitelné stejně jako portable verze pro Windows), ale také balíček AppImage. Ten je vhodnější pro distribuci, protože se jedná o jeden jediný spustitelný soubor. Nevýhodou je, že v takovém případě program nemá žádnou stálou složku v které by mohlo být uloženo upravené výchozí nastavení.

Program jsem otestoval na různých verzích OS Ubuntu a fungoval bez problémů. Jedním drobným problémem je, že program nemá povolen přístup k sériovým portům. Přístup lze udělit pomocí příkazů:

```
sudo usermod -a -G tty $USER
sudo usermod -a -G dialout $USER
```

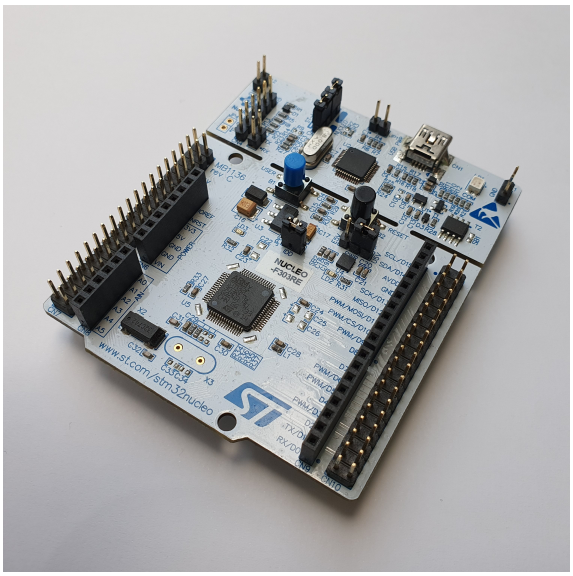
Kapitola 15

Softwarově definované přístroje na mikrokontrolerech

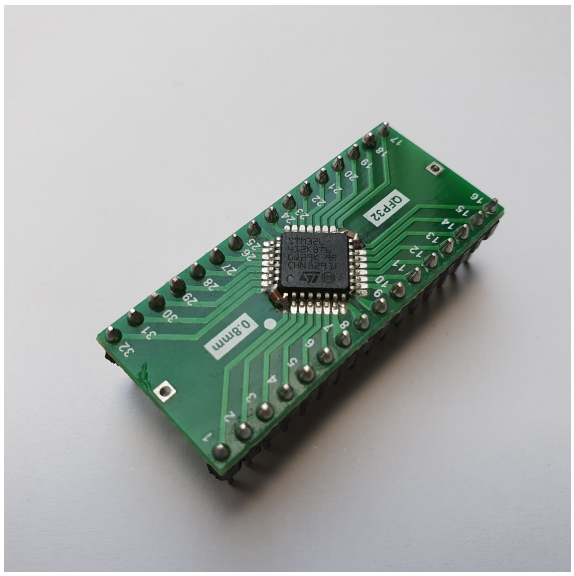
V následující části práce se zabývám tvorbou softwarově definovaných přístrojů na mikrokontrolerech. K disposici mám tyto typy mikrokontrolerů:

- STM32F303RE (Vývojová deska Nucleo): 72 MHz, 64 KB RAM, 4 ADC (5 Msps), DAC
- STM32L412KB: 80 MHz, 40 KB RAM, 2 ADC (5 Msps)
- STM32L072KZ: 32 MHz, 20 KB RAM, 1 ADC (1 Msps), DAC
- ATMega328p (Arduino UNO): 16 MHz, 1 KB, 1 ADC (15 kSps)

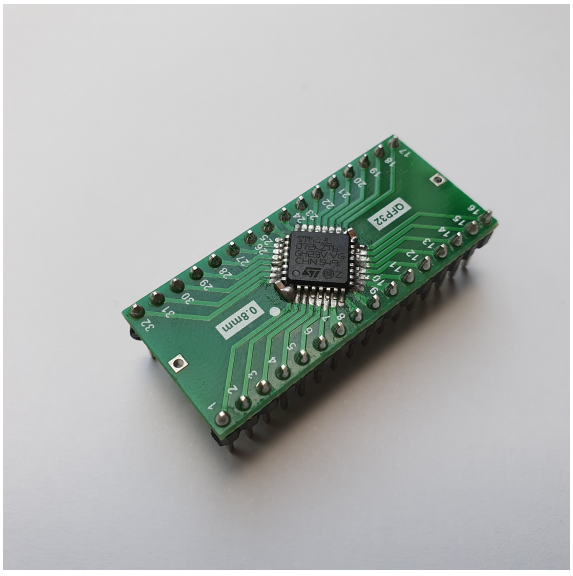
Přístrojů, které je možné realizovat s pomocí mikrokontrolerů, je mnoho, ale za nejužitečnější z nich lze s jistotou považovat osciloskop. Zaměřím se tedy především na vývoj osciloskopu. Mikrokontrolery z řady STM32 nabízejí rychlé AD převodníky a analog–watchdog, čímž jsou pro tento účel velmi vhodné. V případě desky Arduino nelze očekávat ani zdaleka takové výsledky, ale jedná se o velmi rozšířený typ vývojové desky, na které mnoho lidí začíná při učení se programovat mikrokontrolery.



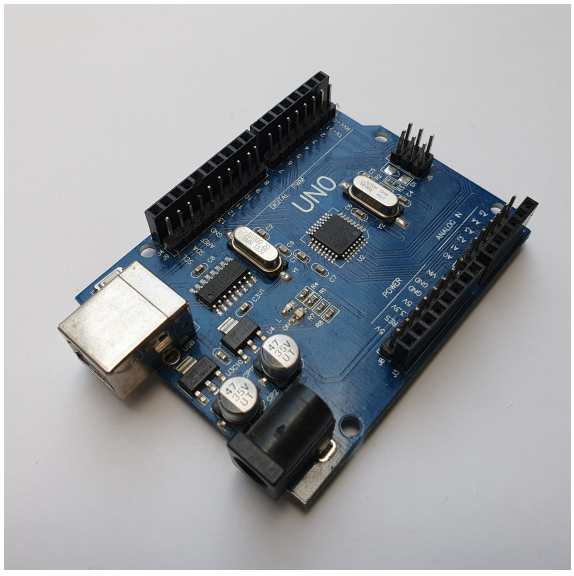
(a) : Nucleo STM32L303RE



(b) : STM32L412KB



(c) : Nucleo STM32L072KZ



(d) : Arduino UNO (ATMega328p)

Obrázek 15.1: Mikrokontrolery a vývojové desky použité v této práci

Kapitola 16

Použití mikrokontrolérů řady STM32

STM32 jsou 32bitové mikrokontrolery založené na jádrech Arm Cortex-M, vyráběné firmou ST Microelectronics.

16.1 Zapojení mikrokontroleru

Oba mikrokontrolery STM32L412 a STM32L072 mají stejné rozmístění vývodů a jsou v pouzdru QFP32, s pomocí adaptéru je lze zapojit do nepájivého pole. Adaptér má navíc na spodní straně u každého pinu pad na SMD součástku velikosti 0805, což umožní snadno přidat blokovací kapacitory a pull-down rezistor pro $BOOT0$ pin. Spodní plocha adaptéru je připojena k V_{SS} (GND) pinům pomocí 0Ω SMD „rezistorů“ (propojek).

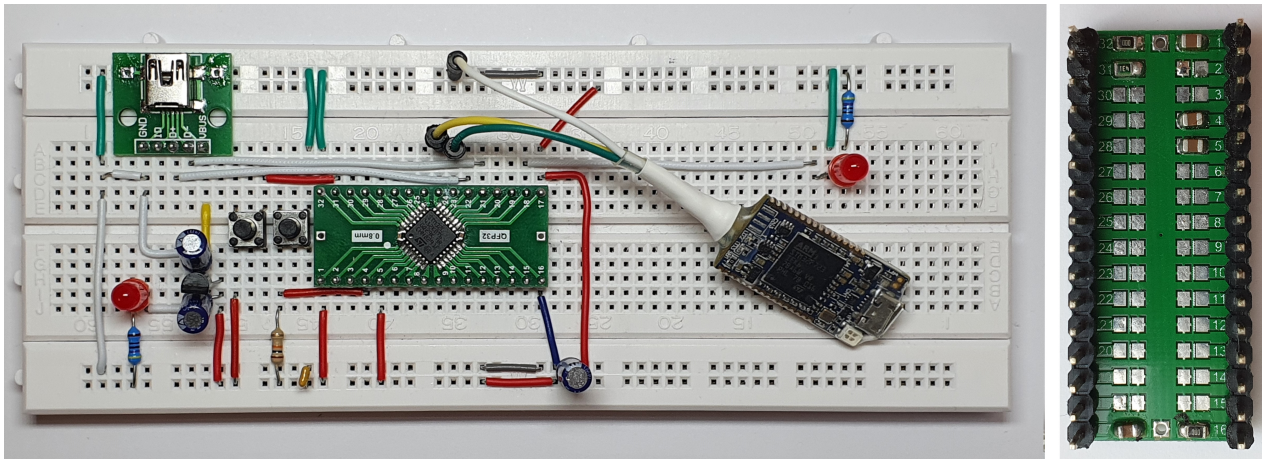
Pro napájení (3.3 V) je použit lineární zdroj HT7533-1 (pouzdro TO92). Resetovací tlačítko spíná pin $NRST$ na zem. $BOOT0$ naopak na V_{DD}.

16.1.1 Blokovací kapacitory

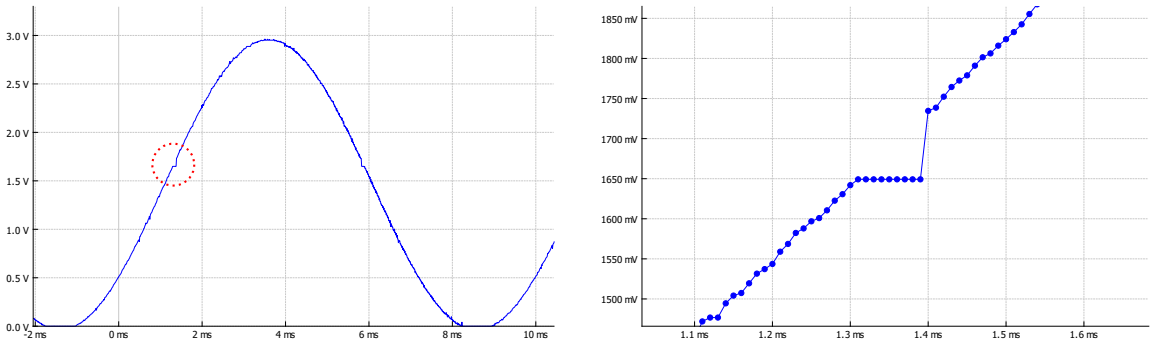
Na V_{DD} jsem použil 100 nF kondenzátor na spodní straně adaptéru, a ještě jeden velmi malý (rozměrově, také 100 nF) připájený přímo mezi piny pouzdra.

Analogová část mikrokontroleru je napájena pinem V_{DDA}, toto napětí je zároveň využito jako referenční pro ADC převodník. Pro správnou funkci je nutné umístit blokovací kondenzátor. Výrobce doporučuje 1 μF paralelně s 10 nF keramickým v těsné blízkosti pinu. Použil jsem jen 1 μF připájený na spodní straně adaptéru.

Na obrázku 16.2 je zobrazeno, jak vypadal záznam z ADC převodníku STM32L412KB bez blokovacího kondenzátoru.



Obrázek 16.1: Mikrokontroler STM32L412 na nepájivém poli a spodní strana adaptéru



Obrázek 16.2: Důsledek nepoužití blokovacího kondenzátoru na V_{DDA}/V_{ref}

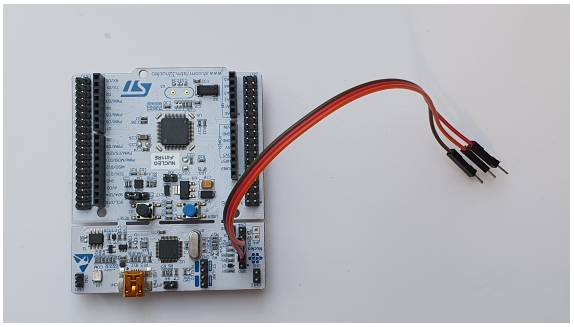
16.1.2 Programování a debug

K programování a ladění (debug) se používá nástroj ST-LINK. Ten umožňuje snadné nahrání firmware a debug ve vývojovém prostředí CubeIDE. Pro nahrání firmware je také možné použít bootloader. Do bootloader režimu se mikrokontroler přepne přivedením vysoké úrovně (3.3 V) na BOOT0 během restartu, poté lze firmware nahrát přes USB či UART s pomocí programu STCubeProgrammer.

V případě vývojové desky Nucleo je ST-LINK přímo součástí desky. Pro programování mikrokontroleru na nepájivém poli lze využít samostatný ST-LINK. Z čínských e-shopů se dají velmi levně sehnat padělky, ale můj čínský ST-LINK nefungoval s STM32L412 (s STM32L072 fungoval).

Proto jsem si pořídil STLINK-V3MINI na který jsem namísto pinové lišty pro plochý kabel jsem připájel tři kabely (GND, SWCLK, SWDIO). To naopak fungovalo pro STM32L412, ale nefungovalo pro STM32L072. Jako nejspolehlivější řešení se ukázalo použití ST-LINK na desce Nucleo (odstraněním „jumperů“ se ST-LINK odpojí od mikrokontroleru na desce).

(a) : Čínský



(b) : Na desce Nucleo

(c) : STLINK-V3MINI

Obrázek 16.3: ST-LINK pro programování samostatného mikrokontroleru

16.2 Vývojové nástroje

Použil jsem vývojové prostředí STM32CubeIDE. To umožňuje nastavit periferie v grafickém rozhraní a následně vygeneruje soubory zdrojového kódu s kódem pro inicializaci periferií.

16.2.1 Knihovny HAL

Knihovna HAL (Hardware abstraction layer) umožňuje ovládat periferie mikrokontroleru pomocí funkcí definovaných v knihovně namísto přímého zápisu do registrů. To usnadňuje psaní kódu a usnadňuje případné portování kódu na jiný typ mikrokontroleru. Velmi užitečný je driver pro USB. HAL má však také nevýhody: kód může být neoptimální za cenu univerzálnosti a dokumentace není moc přehledná [27].

Kapitola 17

Osciloskop s využitím STM32

Osciloskop je jedním ze základních přístrojů pro elektrická měření, umožňuje zobrazit průběh napětí v závislosti na čase.

17.1 Základní vlastnosti a funkce osciloskopu

Kromě samotného zaznamenání průběhu signálu jsou nezbytnou součástí digitálního osciloskopu tyto funkce:

- Trigger: pokud by každý záznam začal v nahodilý okamžik, byl by průběh pokaždé jinak posunutý a zobrazení by bylo nečitelné. Funkce trigger zahájí záznam v definovaném místě signálu, typicky jde o průchod zadanou napěťovou úrovní ve vzestupném či sestupném směru. Pokud signál nikdy nedosáhne nastavené úrovně (vlivem nesprávného nastavení nebo neočekávaného tvaru signálu), pak trigger v základním (normal) režimu nikdy nesepne a zobrazení zůstane prázdné. Pro diagnózu a opravu nastavení je vhodné v takové situaci sepnou alespoň v náhodný okamžik. Takový režim se nazývá auto-trigger.
- Pretrigger: funkce která umožní zaznamenat i průběh signálu i před okamžikem triggeru. Realizace takovéto funkce i analogového osciloskopu je velmi obtížná[1], v případě digitálních osciloskopů je však snadná. Vzorky jsou již před příchodem spouštěcího impulzu ukládány do kruhové fronty a po něm je přidán určitý počet vzorků (např. polovina délky záznamu) a zbylá část zůstane naplněna vzorky z doby před triggerem.
- Single trigger: pokud měříme přechodný děj, udělá přístroj jen jeden záznam a poté ze měření pozastaví.
- Nastavitelná vzorkovací frekvence: pro nezkreslený záznam je samozřejmě nutné, aby vzorkovací frekvence byla alespoň dvojnásobná oproti nejvyšší frekvenční složce obsažené v měřeném signálu.

[1] Podle [28] je to možné s využitím zpožďovací linky. Podle [29, s. 87] je zpožďovací linka potřebná v každém případě, protože rozběhnutí časové základny má nezanedbatelné zpoždění za spouštěcím impulsem (Velikost zpoždění 20 až 200 ns)

Ne vždy však platí, že vyšší vzorkovací frekvence je lepší. Počet vzorků je omezen velikostí RAM, vysoká frekvence tedy znamená zaznamenání kratšího časového úseku. Vzorkování také zatěžuje měřený obvod, protože je třeba naplnit kapacitor v převodníku. Pokud je doba vzorkování příliš krátká, tento přechodný děj nestihne odeznít (kapacitor se nabije na nižší než skutečné napětí) a hodnoty budou zkreslené.

17.2 Analogově digitální převodník

Analogově digitální převodník (ADC) převádí hodnotu napětí na binární číslo které je dále zpracováno v digitálním obvodu. Mikrokontrolery obvykle (včetně mnou používaných) používají převodník s postupnou aproximací.

17.2.1 ADC převodník s postupnou aproximací

Obecný princip převodníku s postupnou aproximací je ten, že vstupní napětí (uložené v kondenzátoru) je porovnáno s úrovněmi napětí odpovídajícím bitům výsledného čísla. Nejprve je porovnáno s hodnotou odpovídající `1000...` (polovina referenčního napětí). Pokud je vyšší, je bit ponechán na 1, jinak je nastaven na 0. V dalším kroku je napětí porovnáno s `1100...` respektive `0100...` dle výsledku předchozího kroku. Takto se postupuje pro všechny bity.

Princip převodníku v mikrokontrolerech STM32 je popsán v [30] a nepovažuji za účelné ho zde převyprávět.

Mnou použité mikrokontrolery z řad STM32 mají 12bitový převodník. Uvažuji-li, že referenční napětí V_{ref+} je stejné jako napájecí, tedy 3.3 V a V_{ref-} je 0, pak pro naměřenou hodnotu X platí, že napětí

$$V_{IN} = 3.3\ V \cdot \frac{X}{2^{12}}. \tag{17.1}$$

Maximální hodnota (dvanáct jedniček) odpovídá napětí o jeden krok rozlišení nižší, než 3.3 V.

17.2.2 Více kanálů

Mikrokontrolery mnohdy mají jen jeden AD převodník, ten však má několik vstupních kanálů. V takzvaném „scanning“ režimu převodník převede postupně všechny aktivované kanály.

Pochopitelně se tím sníží vzorkovací frekvence na jednotlivých kanálech. Pokud jedním 1 Msps převodníkem měřím střídavě 4 kanály, dosáhnu maximálně 250 ksps na jednom kanálu.

17.2.3 Vzorkování v pravidelných intervalech

Převod lze spouštět v pravidelných intervalech s pomocí časovače (timer). Hodnoty lze přímo ukládat do paměti s pomocí DMA.

Zdroj hodinového signálu

Mikrokontrolery STM32 mají v sobě integrovaný RC oscilátor označovaný HSI (high speed internal) s tolerancí ±1 %. Deska Nucleo (F303RE) je navíc vybavena externím krystalovým oscilátorem.

Frekvenci lze ověřit různými způsoby. Jedním ze způsobů je zobrazit na osciloskopu PWM výstup z mikrokontroleru společně se signálem z generátoru (který považuji za přesný) o stejné nastavené frekvenci. Pokud frekvence signálů není shodná, bude se měnit jejich fáze (budou se vůči sobě posouvat). Frekvenci generátoru lze poté měnit, dokud signály nezůstanou vůči sobě nehybné. V tom případě se bude skutečná frekvence signálu mikrokontroleru rovnat nastavené frekvenci na generátoru.

Další možností je použít čítač pro měření frekvence PWM výstupu. Touto metodou jsem udělal experiment s Nucleo F303RE. Nastavil jsem frekvenci PWM na 1kHz a změřil jsem ji přístrojem UNI-T UTG932 (generátor signálu vybavený čítačem).

- Při použití HSI byla odchylka 0.10 %.
- Při použití HSE byla odchylka 0.0005 %.

17.2.4 Doba vzorkování a převodu

U všech tří použitých mikrokontrolerů trvá převod 12.5 cyklů hodinového signálu ADC. Doba vzorkování je nastavitelná. V tabulkách 17.1 až 17.3 jsou uvedeny možné hodnoty doby vzorkování a k nim vypočtené vzorkovací frekvence pro maximální takt ADC.

Nejnižší doba vzorkování pochopitelně umožňuje nejvyšší vzorkovací frekvenci, má však i nevýhodu. Za dobu vzorkování se musí stihnout nabít kapacitory v převodníku, pokud je výstupní odpor zdroje (měřeného napětí) příliš velký, kapacitory v ADC se nestihnou nabít.

Nabíjení kapacitorů AD převodníku

Obrázek 17.1 zobrazuje zjednodušené schéma převodníku s kapacitou C na jehož vstup je připojeno napětí V_{IN} v výstupním odporem R. Nabíjení kapacitoru lze popsat diferenciální rovnicí:

$$\dot{u_c}(t) = \frac{1}{C} \cdot i(t) = \frac{1}{C} \cdot \frac{V_{IN} - u_c(t)}{R}. \tag{17.2}$$

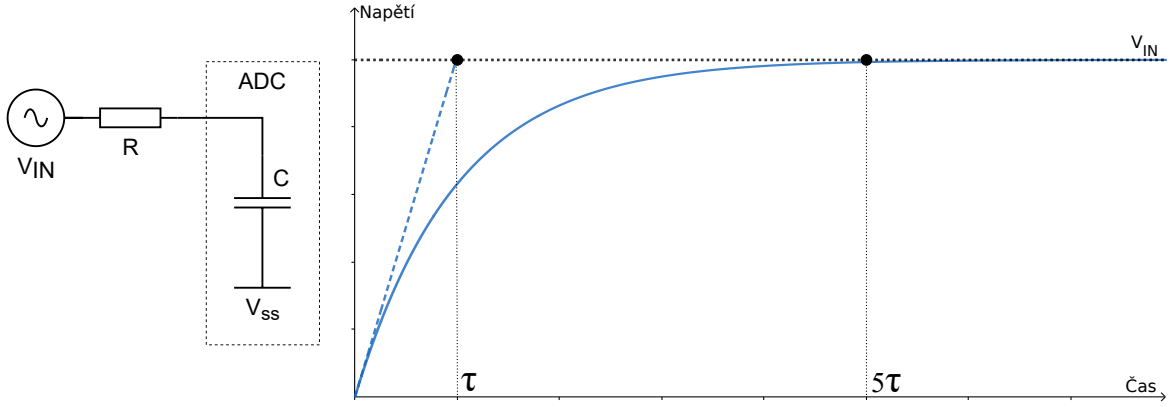
Jejím řešením je exponenciála, která se asymptoticky blíží k napětí V_{IN}, jak je zobrazeno v grafu 17.1.

$$u_c(t) = V_{IN}(1 - e^{-\frac{1}{RC}t}) \tag{17.3}$$

Udělám-li tečnu v bodě 0, získám přímku $u(t) = V_{IN} \cdot \frac{1}{RC} \cdot t$. Ta protne úroveň V_{IN} v čase RC, který se nazývá časová konstanta τ.

Za tento čas dosahuje napětí kapacitoru hodnoty $V_{IN}(1 - e^{-1})$, tedy asi 63 % vstupního napětí. Pro plné využití rozlišení převodníku je nutné volit takovou dobu, pro kterou je odchylka $e^{\frac{n \cdot \tau}{\tau}}$ menší než rozlišení převodníku, které je polovinou hodnoty odpovídající nejnižšímu bitu (LSB). Bit nejnižšího řádu dvanáctibitového převodníku odpovídá e^{-12} násobku rozsahu napětí. Dostatečná doba vzorkování je 9τ, která vede na odchylku $0.5 \cdot$ LSB 12bitového převodníku.

U osciloskopu však nepožadujeme takové rozlišení jako u voltmetrů, proto jsem se rozhodl jako kompromis zvolit pětinásobek časové konstanty. Ten vede na maximální chybu $e^{-5} = 6.7 \cdot 10^{-3}$, což odpovídá $27.5 \cdot$ LSB 12bitového převodníku. Maximální výstupní impedanci měřeného napětí tedy vypočítám pro 5τ.



Obrázek 17.1: Přechodný jev při nabíjení kapacitoru v ADC

Maximální impedanci měřeného zdroje lze vyjádřit ze vzorce

$$t_{sa} = 5 \cdot RC \implies R = \frac{t_{sa}}{5C}, \tag{17.4}$$

kde doba vzorkování t_{sa} je počet cyklů dělený frekvencí hodinového signálu. Velikost vzorkovacího kondenzátoru je 5 pF pro L412 a F303, v případě L072 je to 8 pF (viz příslušné datasheety [31] [32] [33]). Hodnoty vypočtené pro jednotlivé mikrokontrolery (pro 5τ) jsou uvedeny v tabulkách 17.1 až 17.3.

Tabulka 17.1: Vzorkovací frekvence STM32F303 (72 MHz)

Doba vzorkování [cykly]	1.5	2.5	4.5	7.5	19.5	61.5	181.5	601.5
Maximální F_s [MHz]	5.143	4.800	4.235	3.600	2.250	0.973	0.371	0.117
Maximální impedance [kΩ]	0.8	1.4	2.5	4.2	10.8	34.2	100.8	334.2

Tabulka 17.2: Vzorkovací frekvence STM32L412 (80 MHz)

Doba vzorkování [cykly]	2.5	6.5	12.5	24.5	47.5	92.5	247.5	640.5
Maximální F_s [MHz]	5.333	4.211	3.200	2.162	1.333	0.762	0.308	0.123
Maximální impedance [kΩ]	1.3	3.3	6.3	12.3	23.8	46.3	123.8	320.3

Tabulka 17.3: Vzorkovací frekvence STM32L072 (16 MHz)

Doba vzorkování [cykly]	1.5	3.5	7.5	12.5	19.5	39.5	79.5	160.5
Maximální F_s [MHz]	1.143	1.000	0.800	0.640	0.500	0.308	0.174	0.092
Maximální impedance [kΩ]	2.3	5.5	11.7	19.5	30.5	61.7	124.2	250.8

17.2.5 Analog watchdog

Analog watchdog je periferie, která při každém převodu porovná změřenou hodnotu s nastaveným oknem (minimální a maximální úroveň), pokud je hodnota mimo toto okno, je vyvoláno přerušení.

Tuto periferii lze efektivně využít jako trigger. Pokud chci sepnout na vzestupné hraně při polovině referenčního napětí (odpovídá hodnotě 2048), nejprve nastavím okno mezi 2048 a 4095. Jakmile signál opustí toto okno, je vyvoláno přerušení. V reakci na něj změním nastavení na 0 až 2048, jakmile hodnota vystoupá nad požadovaných 2048, je vyvoláno přerušení. Nakonec je okno nastaveno na plný rozsah (0 až 4095), aby už k dalšímu přerušení nedošlo.

Pokud je signál zašuměný, může se stát, že signál ihned po poklesu lehce pod úroveň triggeru opět vyskočí nad ni a dojde k falešnému sepnutí na sestupné hraně. Proto je vhodné přidat hysterezi: v první fázi umístit horní úroveň níž.

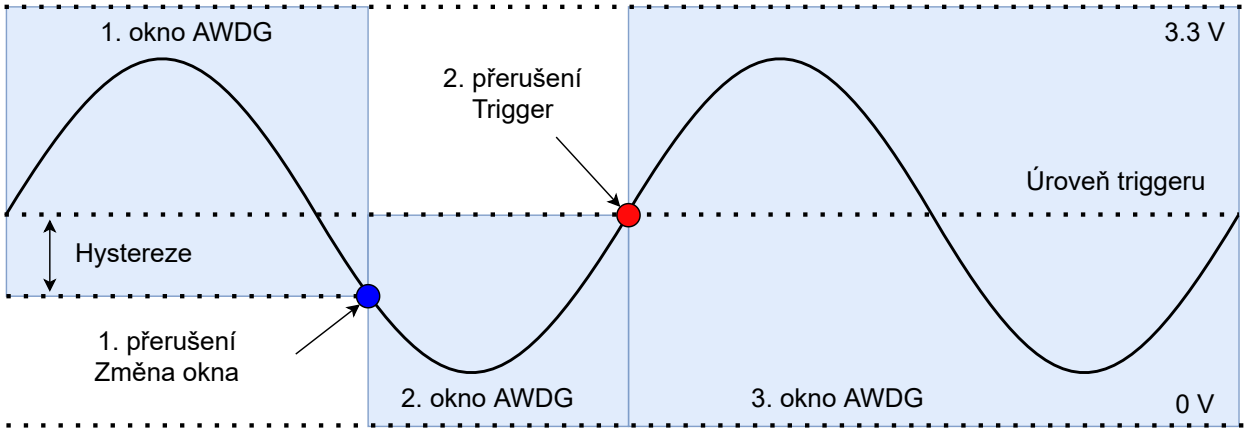
17.3 Implementace vzorkování a triggeru

Využívám 2 časovače: timer1 a timer2. Timer2 je použit ke spouštění AD převodu (Převodník má nastavený „regular conversion“ na timer2 output event). Timer1 je slave timeru 2 a počítá směrem dolů (s každým vzorkem se dekrementuje).

Na začátku vzorkování je perioda `ARR` timeru1 nastavena na maximální hodnotu (65 535), což je limit pro autotrigger. Převod zahájím zapnutím timeru2. Hodnoty jsou prostřednictvím DMA ukládány do kruhového bufferu. Když nastane trigger, přepíšu counter na hodnotu odpovídající počtu vzorků které mají následovat po triggeru. Po uplynutí timeru1 je vyvoláno přerušení v kterém zastavím timer2.

Trigger samozřejmě nesmí přijít dříve, než je změřen potřebný počet vzorků pro pretrigger. Proto analog watchdog povolím až po vyvolání `half-complete` interruptu od DMA (případně celý kompletní, pokud je pretrigger vyšší než 50 %).

Pokud je osciloskop v režimu single-trigger, je autotrigger vypnut tím, že pokud před zavoláním interruptu od timeru1 nedošlo k triggeru, nedojde k zastavení timeru2 a timer1 běží znovu od 65 535.



Obrázek 17.2: Využití analog watchdog jako triggeru

17.3.1 Problémy a nedostatky

Při vysokých vzorkovacích frekvencích nedojde k včasnému zastavení timeru2 (který spíná ADC) a po triggeru je nasbíráno více vzorků, než mělo být. Kromě zastavení timeru se na tomto může podílet i opožděné nastavení timeru při triggeru. Toto kompenzuji tím, že zjištěná zpoždění odečítám od hodnoty, kterou nastavím do timer1 po triggeru (pro zjištění zpoždění jsem měřil obdélníkový signál, kde je zřejmé, na kterém vzorku měl trigger nastat). Zpoždění však není konstantní, takže zejména při maximální rychlosti vzorkování je záznam nestabilní.

17.4 Nastavení vzorkovací frekvence

Vzorkovací frekvenci může uživatel nastavit prostřednictvím terminálu. Pro její změnu je potřeba změnit frekvenci timeru2. Tu lze měnit nastavením prescaleru `PSC` nebo periody `ARR`. Čísla zapsaná do `PSC` a `ARR` jsou vždy o jedna menší než zamýšlená hodnota.

$$F = \frac{F_{clk}}{(PSC + 1) \cdot (ARR + 1)} \tag{17.5}$$

Pokud se nenastavuje libovolná hodnota, ale jen se vybírá z nabídky, je nejjednodušší vytvořit tabulku příslušných hodnot `PSC` a `ARR` pro tyto frekvence.

V případě STM32F303 (72 MHz) neexistuje kombinace celočíselných hodnot která by vedla na přesných 5 MHz. Skutečná frekvence při nastavení 5 Msps se tedy mírně liší (což je zohledněno v záhlaví přenášených dat, ale v terminálu je pojmenována „5 MHz“).

17.5 Komunikace mikrokontroleru s počítačem

Data jsou posílána verzí protokolu pro přenos celého kanálu `$$C`. Pro ovládání je použit terminál, tedy každý příkaz je jen jeden znak.

17.5.1 Komunikace mikrokontroleru s počítačem prostřednictvím USB

Odesílání dat přes USB Virtual Com port se provádí pomocí funkce `CDC_Transmit_FS`. Při použití této funkce jsem narazil na dva problémy:

- Funkce nečeká na dokončení odeslání, pokud se pokusím odeslat dvě zprávy těsně po sobě, odešle se jen první. Druhé zavolání vrátí `USBD_BUSY`.
- Pokud není na straně počítače otevřen port, k odeslání nedojde (na rozdíl od UART který nepozná, jestli je připojen). Je tedy třeba při čekání na odeslání zavést timeout.

Upravil jsem definici této funkce. Po odeslání čekám na `hcdc->TxState == 0`, což značí dokončené odeslání. `For` cyklus slouží jako timeout.

```
uint8_t CDC_Transmit_FS(uint8_t* Buf, uint16_t Len)
{
    uint8_t result = USBD_OK;
    auto *hcdc = (USBD_CDC_HandleTypeDef*) hUsbDeviceFS.pClassData;
    if (hcdc->TxState != 0) return USBD_BUSY;
    USBD_CDC_SetTxBuffer(&hUsbDeviceFS, Buf, Len);
    result = USBD_CDC_TransmitPacket(&hUsbDeviceFS);

    for (uint32_t usbTimeout = 100000; usbTimeout > 0; usbTimeout--) {
        if (hcdc->TxState == 0)
            break;
    }
    return result;
}
```

17.5.2 Komunikace mikrokontroleru s počítačem prostřednictvím UART

Deska Nucleo sice má USB port, ten však není připojen přímo do mikrokontroleru, nýbrž do ST–LINK, který obsahuje USB–UART převodník, komunikace samotného mikrokontroleru probíhá přes UART. Komunikace je výrazně pomalejší než u USB. To omezuje počet odeslaných průběhů za čas, a také zhoršuje responzivitu interaktivního terminálu (změna terminálu se neodešle, dokud neskončí právě probíhající odesílání dat). Nejvyšší rychlost komunikace, kterou UART podporuje je 2250000 (2.25M), to však na některých počítačích není spolehlivé. Proto jsem umožnil změnit baudrate na 115200 stisknutím tlačítka na desce Nucleo.

Příjem ovládacích příkazů je v režimu využívajícím přerušení `HAL_UART_Receive_IT`. Výhoda terminálového ovládání je že příkaz je vždy jediný znak a zpracování tedy není nijak komplikované.

17.5.3 Detekce spojení mikrokontroleru se zobrazovací aplikací

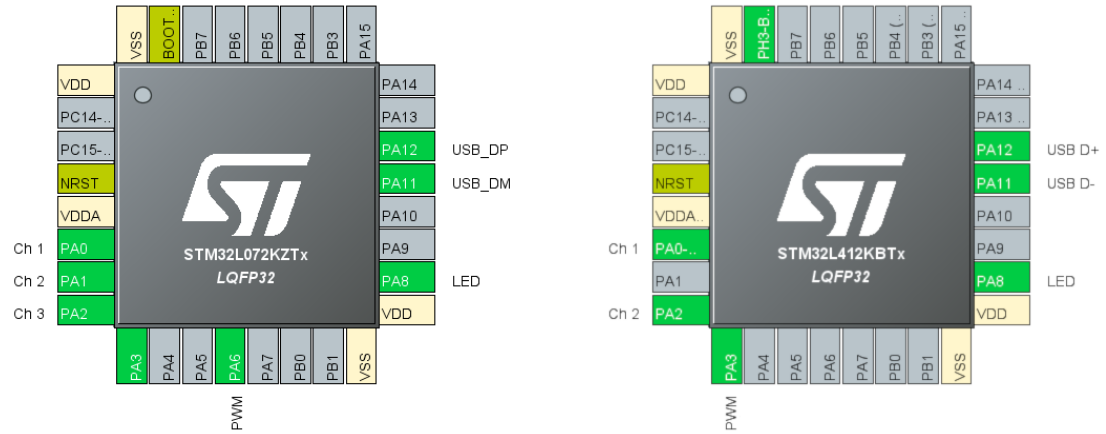
Mikrokontroler musí vědět kdy došlo k připojení k zobrazovací aplikaci, aby poslal základní text do terminálu a případné další nastavení. Pro účel kontroly připojení je v zobrazovací aplikaci implementován příkaz „echo“ `$$E`, který odešle přijatý text zpět do portu. Pokud mikrokontroler pošle `$$EX`, dostane odpověď `X` (pokud je spojení s aplikací aktivní).

Spojení je takto ověřováno v pravidelných intervalech. Pokud je za dobu před dalším odesláním přijata odpověď, je připojení potvrzeno. Pokud se minulé ověření nezdařilo a současné ano, došlo k novému připojení.

17.6 Výsledky tvorby osciloskopů na mikrokontrolerech STM32

Osciloskop jsem vytvořil na všech třech zmíněných mikrokontrolerech. Ve všech případě je ovládání zajištěno pomocí interaktivního terminálu. GUI v terminálu odpovídá tomu které je zobrazeno v kapitole o terminálu (obrázek 12.3 na straně 74). Za nejlepší bych považoval verzi pro STM32L412, která má 2 kanály s maximální rychlostí 5 Msps. Verze pro Nucleo STM32F303 má 4 kanály s 5 Msps, ale komunikace přes UART se ukázala být výrazným omezením oproti USB. Zejména v případě dlouhého záznamu se dlouhá doba přenosu výrazně projevuje nízkou obnovovací frekvencí a opožděnou reakcí na ovládání. Proto jako výchozí nastavení využívám jen 1024 vzorků na kanál a pouze dva aktivní kanály (ostatní lze aktivovat v terminálu).

STM32L072 nabízí mnohem menší možnosti: pouze jeden převodník s 1 Msps, zde jsem použil 4 kanály, tedy každý jen 250 ksps. To slouží spíše pro demonstraci použití komunikačního protokolu ve verzi pro více kanálů měřených jedním převodníkem (hodnoty kanálů v bufferu jsou střídavě, viz 6.3.4).



(a) : STM32L072 **(b) :** STM32L412

(c) : Pro Nucleo F303 jsem zvolil stejné piny, jaké jsou použity na LEO (viz [1])

Obrázek 17.3: Rozložení pinů mikrokontrolerů ve funkci osciloskopu

Kapitola 18

Osciloskop s využitím Arduino UNO

Mikrokontroler ATMega328p [34] na kterém je deska Arduino založena nenabízí ani zdaleka takové možnosti jako STM32 v předchozí kapitole. AD převodník dosahuje rychlosti pouhých 15 kSps a má rozlišení 10 bitů. Není zde k dispozici DMA ani analog–watchdog.

18.1 Knihovna Arduino

Při programování desek Arduino se obvykle používá knihovna Arduino, které je pro začátečníky jednodušší, než přímí zápis do registrů. Ve svém kódu jsem se však snažil ji využívat co nejméně. Zápis do registrů je výhodnější z hlediska rychlosti běhu programu a velikosti kódu. Některé periferie s pomocí knihovny Arduino vůbec není možné přímo ovládat (zejména časovače). Pro odesílání a přijímání dat přes UART však využívám funkce z knihovny Arduino.

18.2 AD převodník

Doba převodu je řízena hodinovým signálem procesoru děleným předděličkou (prescaler). Funkce `analogRead` z knihovny Arduino používá prescaler nastavený na 32. S tím lze dosáhnout vzorkovací rychlosti okolo 10 kHz. Změnou hodnoty `prescaler` na 16 lze dosáhnout až 20 kHz a přestože je to více než výrobcem uváděných 15 Msps, vypadá výsledek stále slušně.

Převod lze ovládat pomocí časovače, nastavením `ADC trigger source` na `timer0 compare A`. Po dokončení každého převodu je vyvoláno přerušení s obslužnou funkcí `ISR(ADC_vect)`. V této funkci lze přečíst hodnotu z ADC. Ta se skládá z 10 bitů, z nichž nejvyšších 8 je v registru `ADCH` a zbylé dva v `ADCL` (pokud je v převodníku nastaveno `left adjust`). Vzhledem k velmi omezené velikosti paměti RAM ukládám pouze hodnoty z `ADCH`, lze tak použít buffer o velikosti 1700 vzorků.

Hodnoty ukládám do kruhového bufferu a při každém uložení kontroluji, zda došlo k překročení úrovně triggeru. Pokud nastane trigger, začnu při každém dalším uložení odpočítávat zbývající

vzorky. Převod poté zastavím zastavením `timer0`.

18.3 Komunikace desky Arduino s počítačem

Mikrokontroler ATMega328p nemá USB, pouze UART a deska Arduino je vybavena UART-USB převodníkem. Komunikace probíhá obdobně jako u desky Nucleo v předchozí kapitole. Vzhledem a absenci DMA a AWDG je však v průběhu vzorkování procesor zaměstnán ukládáním hodnot a kontrolováním triggeru a proto jsou příchozí data přečtena vždy až po skončení měření. Pro odeslání dat používám funkci `Serial.printf` z knihovny Arduino. Pro příjem dat je použit kód:

```
void checkSerialRx()
{
  while (Serial.available() > 0)
  {
    char c;
    Serial.readBytes(&c, 1);

    // ...

  }
}
```

Zejména pro výpis do terminálu je v kódu mnoho konstantních textových řetězců. Tyto textové řetězce by za normálních okolností byly uloženy v RAM a zabírali místo které by jinak bylo využitelné pro buffer na vzorky. Knihovna Arduino však nabízí makro `F()`, které textový řetězec uloží do paměti PROGMEM (flash) namísto RAM [35].

```
Serial.print("$$T\e[8;1H\e[38;5;254m\e[48;5;254m0\e[9;1H1"); // RAM

Serial.print(F("$$T\e[8;1H\e[38;5;254m\e[48;5;254m0\e[9;1H1")); // PROGMEM
```

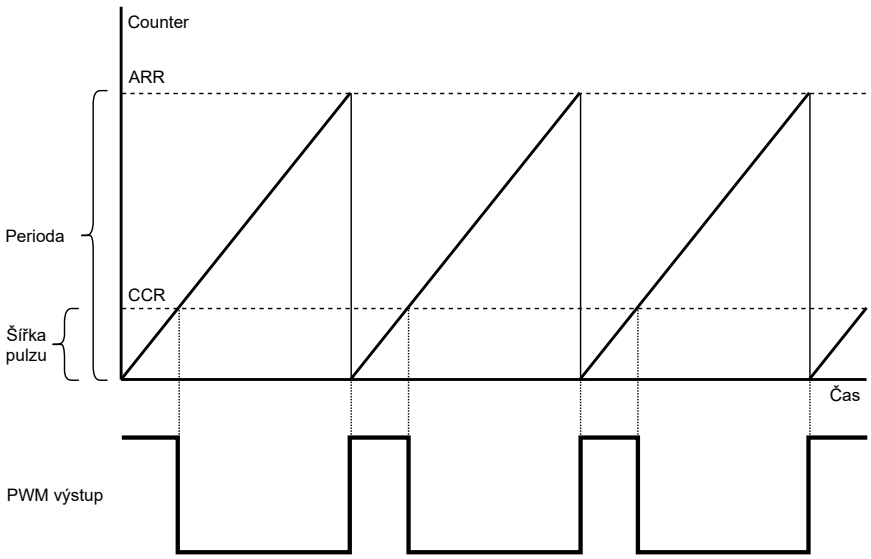
Kapitola 19

Generátory signálů na STM32

Přestože nemám v plánu vytvořit generátor signálu jakožto samostatný přístroj, pokusím se funkci základního generátoru přidat do osciloskopů. Rozhodl jsem se do osciloskopických přístrojů (popsaných v kapitole 17) přidat generátor PWM signálu s nastavitelnou frekvencí a střídou.

19.1 Generování PWM signálu

Většina časovačů v mikrokontrolerech nabízí funkci generování PWM signálu. Na obrázku 19.1 jsem znázornil jak toto funguje. Výstup je nejprve na vysoké úrovni, dokud `counter` nedosáhne hodnoty `capture compare`. Poté se výstup změní na nízkou úroveň, dokud nezačne nová perioda časovače.



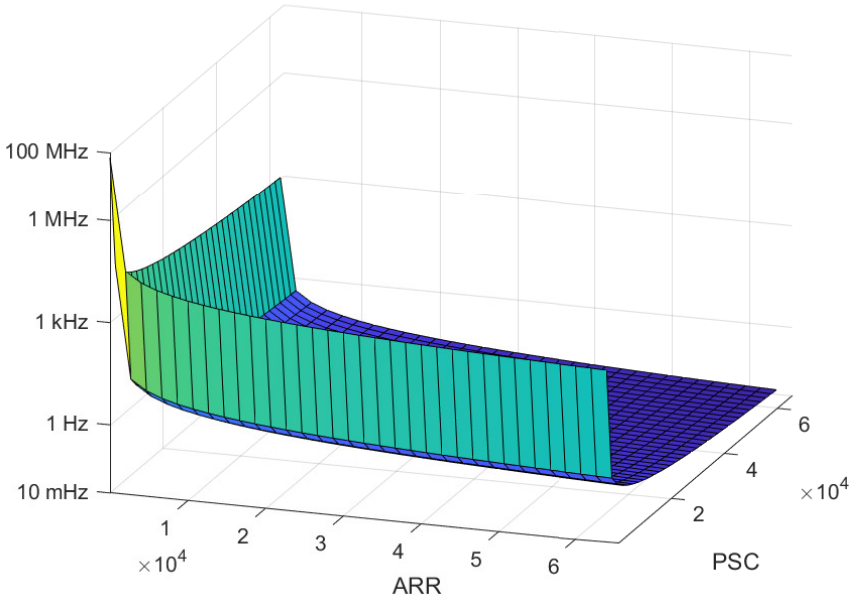
Obrázek 19.1: Princip generování PWM signálu pomocí časovače

19.2 Nastavení frekvence časovače

Časovač má při nastavení frekvence dva stupně volnosti: `prescaler` (PSC) a `period` (ARR). Vztah mezi frekvencí časovače f, těmito parametry a frekvencí hodinového signálu f_{osc} je

$$f = \frac{f_{osc}}{PSC \cdot ARR}. \tag{19.1}$$

což je vykresleno v grafu 19.2 pro STM32F303 (72 MHz). Do registrů `ARR` a `PSC` se zapisují hodnoty o jedna menší, než uvažuji ve výpočtech (například `PSC=0` znamená, že frekvence je dělena 1×).



Obrázek 19.2: Závyslost frekvence časovače na PSC a ARR

Nastavená frekvence nemůže být libovolná, a to ze dvou důvodů: hodnoty ARR a PSC jsou celá čísla a jejich hodnoty jsou omezené (pro STM32 obvykle 16 bitů). Požadovanou frekvenci může a nemusí být možné nastavit přesně. Nejobtížnější úkol je nalezení optimální kombinace PSC a ARR. Naivní přístup by byl vyzkoušet všechny možné kombinace a vybrat tu která se nejméně odlišovala (nebo byla přesná). Počet kombinací je však extrémně vysoký ($2^{16} \cdot 2^{16} = 4294967296$, 4 miliardy). Takový postup by sice umožnil nalézt přesnou kombinaci (pokud existuje), ale je prakticky nepoužitelný kvůli dlouhé době nutné pro výpočet.

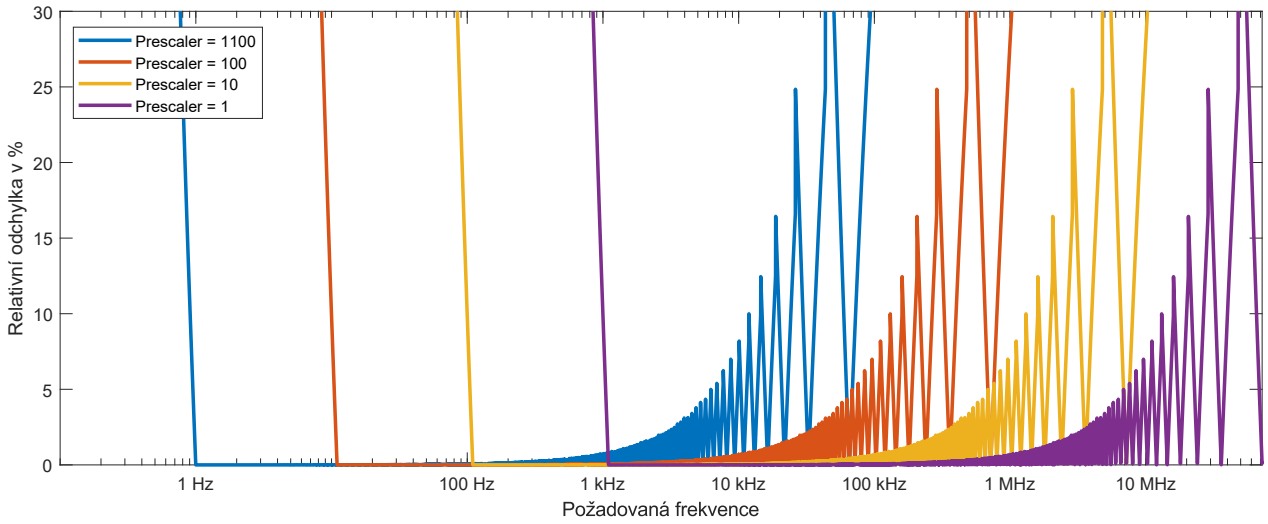
19.2.1 Nastavení při pevné hodnotě PSC

Ponechám–li PSC nastavený na pevnou hodnotu, lze ARR dopočítat jednoduše z rovnice

$$ARR = \frac{f_{osc}}{f \cdot PSC}. \tag{19.2}$$

Požadovanou hodnotu f v obecném případě nelze přesně dosáhnout, protože ARR musí být zaokrouhleno na celé číslo. Odchylka je tím vyšší, čím je frekvence f blíže hodnotě $\frac{f_{osc}}{PSC}$, jak je vidět z grafu 19.3. Navíc je možné, že pro dostatečně malou frekvenci hodnota ARR vyjde vyšší, než umožňuje její bitová velikost. Samozřejmě není možné takto dosáhnout frekvence vyšší, než je frekvence hodinového signálu z procesoru.

Jako požadovaný rozsah frekvencí jsem si stanovil 1 Hz až 1 MHz. V grafu 19.3 jsem vykreslil relativní odchylky frekvencí nastavených při pevném PSC od požadované frekvence. Je vidět, že přesnost se výrazně zhoršuje pro vyšší frekvence (malé ARR), u nižších frekvencí (velké ARR) je přesnost velmi dobrá, ale je náhle utnuta dosažením maximální hodnoty ARR. Je tedy zjevné, že je žádoucí udržovat co nejnižší PSC tak, aby ještě bylo možné dodržet maximální ARR. Pro $PSC = 1$ je minimální frekvence lehce nad 1 kHz (konkrétně $\frac{72MHz}{2^{16}} = 1099Hz$). Při $PSC = 1100$ lze dosáhnout požadovaného 1 Hz.



Obrázek 19.3: Odchylka nejbližší nastavitelné frekvence od požadované pro různé hodnoty PSC

19.2.2 Metoda výpočtu PSC a ARR

Z grafu je vidět, že by bylo optimální nalézt nejnižší hodnotu PSC potřebnou pro dosažení požadované frekvence a následně dopočítat ARR. Hodnotu PSC potřebnou pro dosažení frekvence f lze vypočítat z rovnice

$$PSC = \frac{f_{osc}}{f \cdot ARR_{max}}. \tag{19.3}$$

Hodnotu PSC je nutné zaokrouhlit nahoru. Odchylka ve frekvenci při výpočtu ze vztahu 19.2 je

$$\frac{\Delta f}{f} = \frac{1}{ARR}. \tag{19.4}$$

Při hodnotách ARR v řádu desetitisíců lze dosáhnout přesnost v řádu setin procenta. Nad 10 kHz se bude přesnost zhoršovat, protože PSC již bude na 1 a hodnota ARR tedy bude muset pro vyšší frekvence klesat.

19.2.3 Nastavení střídy PWM signálu

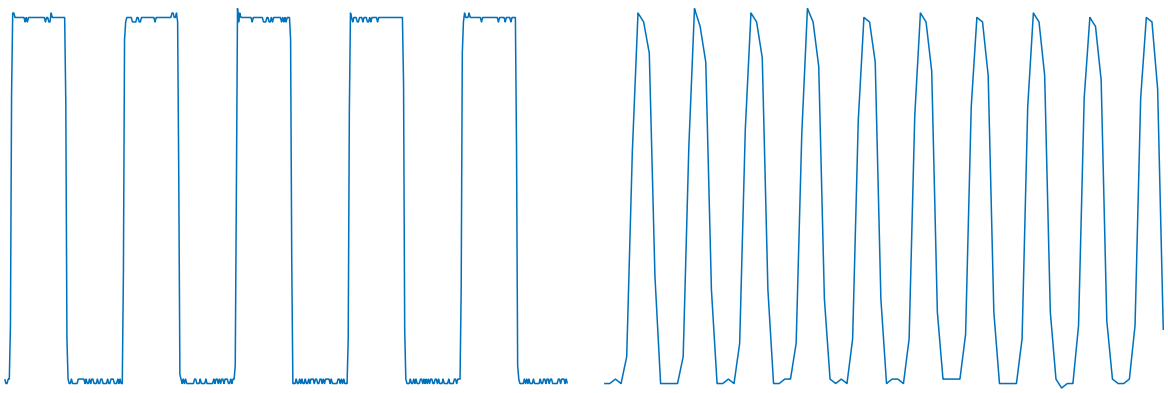
Dyty cycle[1] je doba vysoké úrovně v poměru k periodě PWM signálu, pro generovaný PWM signál jej lze vyjádřit jako

$$Duty\ cycle = \frac{CCR}{ARR} \tag{19.5}$$

a lze ho tedy nastavit změnou hodnoty CCR, jak již bylo zobrazeno na obrázku 19.1. Je zjevné, že vyšší hodnota ARR umožní větší rozlišení pro nastavení CCR, což odpovídá strategii volby nejnižšího možného PSC a tedy vysokého ARR.

19.2.4 Výsledek tvorby generátoru PWM signálu

Při testování s pomocí čítače v přístroji UTG932 se potvrdilo, že pro frekvence do 10 kHz jsou odchylky skutečně v řádu setin procenta, jak předpokládá rovnice 19.4. Pro vyšší frekvence rozlišení klesá, jak vyplývá z průběhu pro $PSC = 1$ v grafu 19.3. Stejně tak se snižuje rozlišení pro nastavení střídy. Pro frekvenci 1 MHz je signál navíc již výrazně zkreslený, jak je zobrazeno na 19.4b. Takto vysoké kmitočty však již nejsou prakticky použitelné, protože převyšují vzorkovací frekvenci osciloskopu. Způsob nastavení frekvence pomocí terminálu je popsán v kapitole 12.2.1.



(a) : Frekvence 1 MHz, duty cycle 50 %

(b) : Frekvence 10 MHz, duty cycle 50 %

Obrázek 19.4: Výstup PWM signálu z STM32L412KB (použit osciloskop Owon VDS1022I)

[1] Pojmy „duty cycle“ a „střída“ jsou často považovány za synonyma. Ale já se budu držet konvence, že střída označuje $\frac{T_{high}}{T_{low}}$, zatímco duty cycle je $\frac{T_{high}}{T}$.

Kapitola 20

Zhodnocení dosažených výsledků

V této kapitole jsou podrobně shrnuty výsledky této práce. Základní částí práce je počítačová aplikace sloužící jako univerzální zobrazovač pro měřicí přístroje realizované s pomocí mikrokontrolerů. Nad rámec zadání jsem přidal pokročilé funkce pro zpracování příchozích dat, zejména FFT, interpolaci a výpočet frekvence. Pro ovládání mikrokontroleru je použit terminál s podporou ANSI escape sekvencí, který navíc umožňuje vytvoření interaktivního menu. Kromě zobrazovací aplikace bylo v rámci práce vytvořeno i několik přístrojů na mikrokontrolerech STM32 a desce Arduino. Jedná se o osciloskopy, z nichž některé jsou dovybaveny i základním generátorem signálu.

20.1 Komnunikace s mikrokontrolerem

Základní úlohou programu je zobrazování dat přijatých z mikrokontroleru. Na začátku práce jsem stanovil podmínky, které má program splňovat, aby byl skutečně univerzální:

- Bude vyhovovat potřebám mnoha typů přístrojů, zejména těchto: osciloskop, logický analyzátor, voltmetr (se záznamem), záznam neelektrických veličin.
- Odeslání dat z mikrokontroleru musí být snadno implementovatelné a nemělo by vyžadovat nadbytečnou softwarovou režii (overhead) na straně mikrokontroleru.
- Přenos dat musí iniciovat mikrokontroler, nikoli PC aplikace. Komunikace směrem z PC do zařízení by měla zahrnovat jen příkazy které zadá sám uživatel.

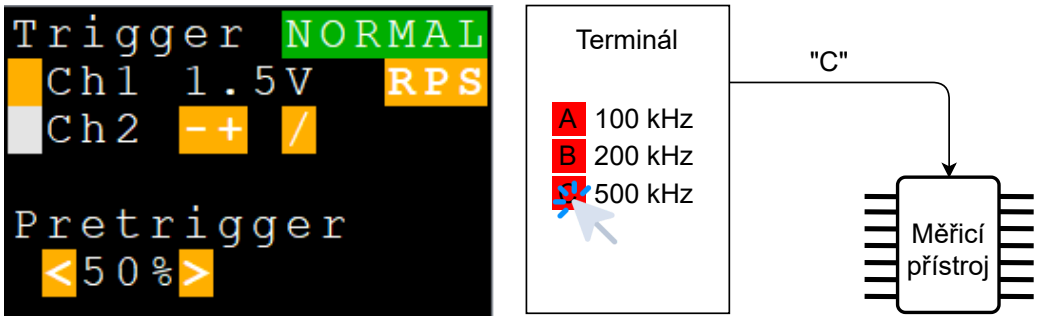
Tyto podmínky jsem dodržel, odesílání dat je řízeno čistě ze strany mikrokontroleru a komunikační protokol umožňuje předat data v podobě v jaké jsou uložena a další zpracování (přepočet hodnot z ADC na napětí, rozdělení bufferu multiplexovaného ADC na jednotlivé kanály) ponechat na počítačové aplikaci. Odeslání dat je možné v binárním i textovém formátu. Textový formát (s pomocí funkce `printf`) je snadněji uchopitelný pro začátečníky, programující například v Mbedu. Binární formát samozřejmě umožní výrazně rychlejší přenos a rychlejší zpracování.

Komunikační protokol umožňuje přidávat data do grafu buď po jednotlivých bodech, což je vhodné pro dlouhodobý záznam, nebo jako celý kanál najednou, to je vhodné pro osciloskopy. Data mohou být analogová i digitální (logický analyzátor). Kromě přidání dat lze také pomocí příkazů měnit nastavení (například rozsah grafu nebo jednotky na osách) a zobrazovat textové zprávy.

20.1.1 Ovládání přístroje

Jako velmi efektivní se ukázalo být využití terminálu podporujícího ANSI escape sekvence pro vytvoření pseudografického uživatelského rozhraní. Zdá se, že pro framework Qt doposud nebyl publikován žádní prvek GUI, který by umožňoval výpis textu s podporou ANSI escape sekvencí. Takový prvek jsem tedy vytvořil úpravou tabulky QTableWidget. Výsledný terminál podporuje sekvence pro pohyb textovým kurzorem a pro formátování textu (změna barvy, tučné písmo, podtržení). Terminál tak umožňuje přehledné zobrazení nastavených parametrů či měřených hodnot.

Pro ovládání přístroje se velmi osvědčil můj nápad na vytvoření tlačítek v terminálu. Funkce odeslání znaku kliknutím umožní s naprostou jednoduchostí vytvořit tlačítko: stačí do terminálu vypsat znak s vybarveným pozadím. Zpracování na straně mikrokontroleru je velmi jednoduché, protože jde vždy o přijetí jednoho bajtu.



(a) : Pseudografické uživatelské rozhraní **(b) :** Ovládání kliknutím myší

Obrázek 20.1: Terminál pro ovládání přístroje

20.2 Zobrazení a zpracování dat

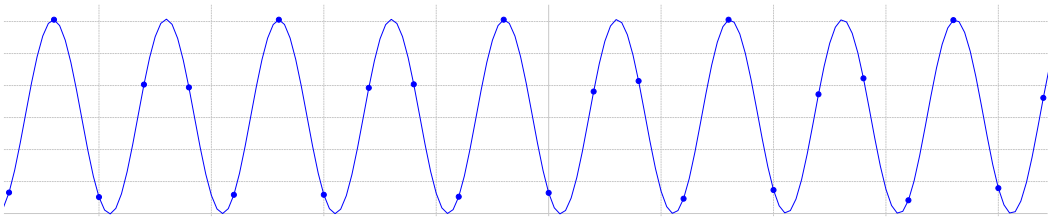
Graf umožňuje zobrazit až 16 analogových kanálů a 32 bitů logického analyzátoru. Pro vykreslování grafu jsem použil knihovnu QCustomPlot, který jsem vylepšil o další funkce jako například zobrazení čísel na osách v automaticky zvolených jednotkách. Také jsem vytvořil funkci „autoset“, která automaticky rozvrhne více kanálů do grafu. Ovládání a nastavení programu probíhá kompletně v grafickém uživatelském rozhraní a základní parametry (například rozsah grafu) lze nastavit i pomocí příkazů z mikrokontroleru

Hodnoty z grafu je možné odečíst pomocí kurzorů, pro které jsem dokonce vytvořil ovládání tažením myší. Samozřejmostí je zobrazení rozdílu hodnot kurzorů, kromě toho program provádí i automatická měření parametrů signálu. Navrhl a použil jsem postup pro optimální výpočet frekvence, využívající

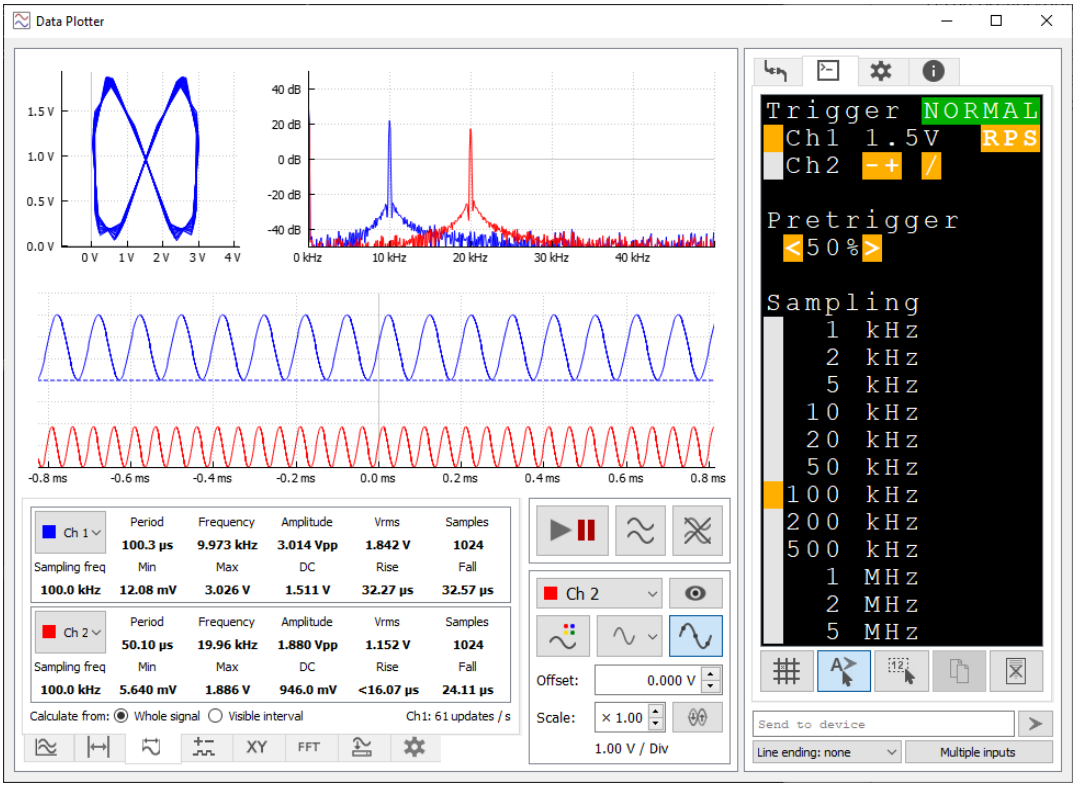
kombinaci určení ze spektra a z autokorelační funkce. Výpočet frekvence dosahuje přesnosti v řádu desetin procenta.

Implementoval jsem (vlastním kódem, bez použití cizí knihovny) výpočet spektra algoritmem FFT, což umožňuje spektrální analýzu signálu. Uživatel může zvolit z několika běžně používaných váhovacích oken (Hammingovo, Hannovo, Blackmanovo). Výpočet periodogramu lze provézt i s využitím Welchovi metody.

S využitím interpolačního filtru navrženého v Matlabu jsem implementoval funkci interpolace signálu, která spolehlivě zrekonstruuje harmonický signál už při pouhých čtyřech vzorcích na periodě. Obrázek 20.2 ukazuje interpolovaný signál o frekvenci 200 kHz navzorkovaný frekvencí 500 kHz, tedy má pouhých 2.5 vzorků na periodě. Na výběr je z několika různých filtrů, aby uživatel mohl zvolit kompromis mezi kvalitou a výpočetní náročností.



Obrázek 20.2: Harmonický signál zrekonstruovaný z malého počtu vzorků pomocí interpolačního filtru



Obrázek 20.3: Okno aplikace s aktivním FFT a XY režimem

20.3 Přístroje vytvořené v rámci projektu

V rámci práce jsem vytvořil osciloskopy založené na mikrokontrolerech STM32. K dispozici jsem měl STM32L072KZ a STM32L412KB jakožto samostatné procesory na nepájivém poli a STM32F303RE na desce Nucleo. Tyto osciloskopy využívají periferii AWDG (analog–watchdog) jako trigger a DMA pro ukládání vzorků do paměti. STM32L412 nabízí 2 AD převodníky o rychlosti 5 Msps, v případě STM32L072 je k dispozici pouze jeden převodník o rychlosti 1 Msps, který je multiplexovaný na čtyři kanály.

Verze využívající desku Nucleo s procesorem STM32F303 nabízí největší možnosti (4 AD převodníky, každý 5 Msps), bohužel je však omezena komunikací přes UART namísto USB použitého u mikrokontrolerů na nepájivém poli. Při použití nejvyššího možného baudrate (2.25 Mbps) je rychlost komunikace dostatečná, ale na některých počítačích toto není spolehlivé. Při přepnutí na nižší baudrate (115200) je komunikace spolehlivá, ale odezva je znatelně horší než při použití USB u ostatních mikrokontrolerů. Tento problém lze zmírnit tím, že je (pomocí ovládání v terminálu) umožněno snížit délku záznamu a vypnout nepoužívané kanály.

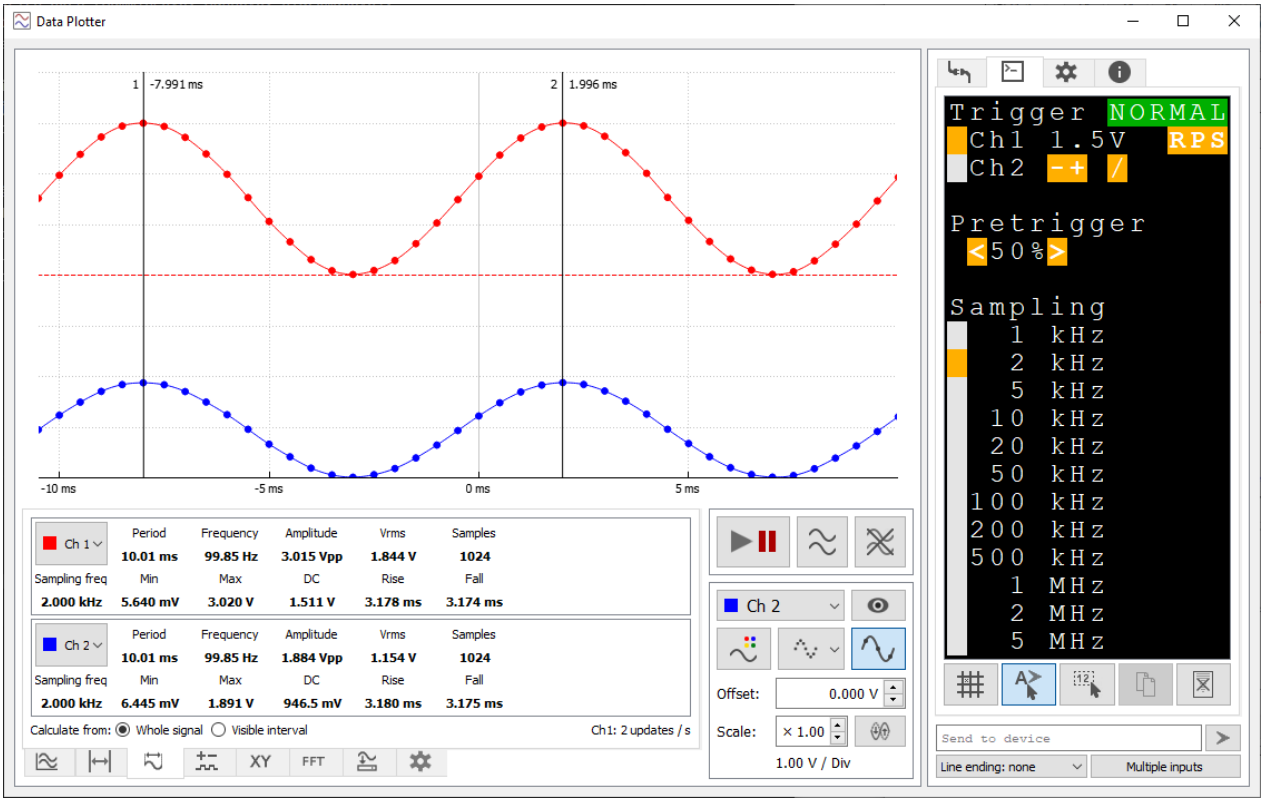
Každý z těchto osciloskopů je vybaven i PWM generátorem. Frekvenci lze (pomocí ovládání v terminálu) nastavit na libovolnou hodnotu od 1 Hz do 10 MHz[1], není však zaručeno, že se požadované frekvence podaří přesně dosáhnout (z důvodu diskrétnosti a omezenosti hodnot předděličky a periody). Po zadání požadované hodnoty je nalezena nejbližší možná kombinace předděličky a periody, poté je v terminálu zobrazena skutečná nastavená hodnota. Pro hledání této kombinace jsem navrhl algoritmus, který sice nezaručuje nalezení nejvhodnější kombinace, ale pro frekvence v rozsahu 1 Hz až desítky kHz dosahuje odchylky jen v řádu setin procenta (pro vyšší hodnoty se přesnost zhoršuje, pro požadovanou frekvenci v řádů několika stovek kHz už se skutečně nastavená frekvence může lišit o jednotky procent).

Osciloskop jsem naprogramoval i pro desku Arduino UNO. Při psaní kódu jsem se snažil vyhnout použití funkcí z knihovny Arduino a používat přímo zápis do registrů. Podařilo se dosáhnout vzorkovací frekvence až 20 ksps, což je velmi málo v porovnání s tím, co umožňují STM32, i tak je to však zajímavá ukázka toho, že i z obyčejného Arduina lze vytvořit funkční osciloskop. Komunikace opět probíhá přes UART, baudrate 250000.

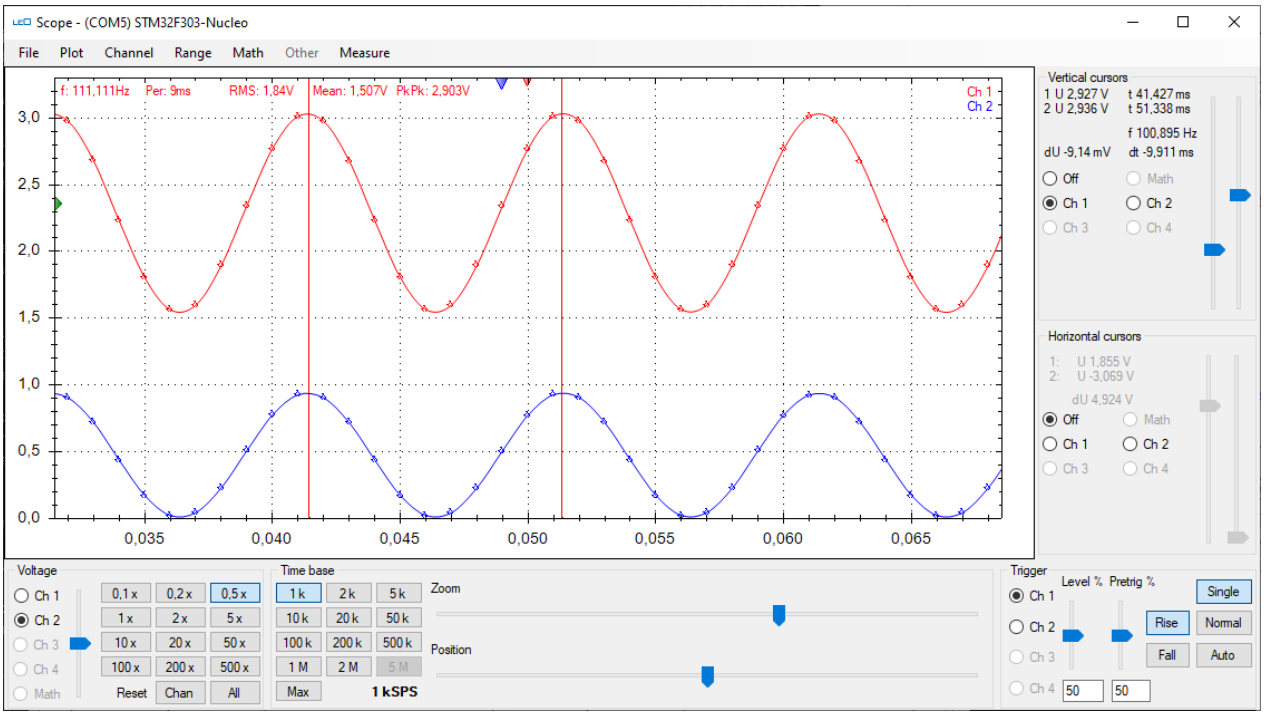
20.4 Srovnání s již existujícími platformami

Tato aplikace je svou univerzálností předurčena k tomu, aby s její pomocí bylo vytvořeno mnoho přístrojů, nabízí se však otázka, zda–li tato univerzálnost není na úkor funkčnosti a uživatelské přívětivosti. Tato obava se však zjevně nenaplnila, moje aplikace nabízí stejné možnosti jako již existující platformy a v mnoha případech dokonce více. Ovládání s pomocí terminálu je (navzdory své jednoduchosti) stejně pohodlné jako používání plnohodnotného GUI. Obrázek 20.4 srovnává tuto aplikaci (kterou jsem pojmenoval „Data Plotter“) s populární platformou LEO, vytvořenou také na FEL [1]. Je vidět, že ovládání v terminálu není o nic horší než tlačítka v grafickém rozhraní okna.

[1] pro STM32L072KZ jen 2 MHz



(a) : Data Plotter (aplikace vytvořená v této práci)



(b) : LEO, Little Embedded Oscilloscope [1]

Obrázek 20.4: Srovnání této aplikace a platformy LEO

Kapitola 21

Závěr

Cílem této práce bylo vytvoření aplikace, která poslouží jako univerzální prostředí pro zobrazování dat z měřicích přístrojů a pro jejich ovládání. To se nepochybně zdařilo. Komunikační protokol je přizpůsoben potřebám mnoha různých druhů přístrojů: osciloskopů, logických analyzátorů i dlouhodobého záznamu hodnot. Přenos dat do počítače nevyžaduje obousměrnou komunikaci mezi počítačem a mikrokontrolerem, komunikace spočívá jen v odeslání hodnot v příslušném formátu, díky čemuž je odesílání dat snadno implementovatelné.

Graf umožňuje zobrazit až 16 analogových kanálů a 32 bitů logického analyzátoru. Kurzory pro odečet hodnot je možné pohodlně ovládat myší, kromě kurzorů lze využít i automatické výpočty charakteristik signálu včetně výpočtu frekvence. Program rovněž umožňuje výpočet spektra algoritmem FFT a vyhlazení průběhu s malým počtem vzorků s pomocí interpolačního filtru.

Pro ovládání přístroje byl navržen systém využívající terminál pro výpis textu s pomocí ANSI escape sekvencí, které umožňují vytvoření pseudografického rozhraní. Terminál je doplněn o funkci odeslání znaku kliknutím, čímž je umožněno v terminálu vytvořit tlačítka pro ovládání přístroje klikáním myší. Funkce se velmi osvědčila jako pohodlný způsob ovládání, který je zároveň nenáročný na implementaci v mikrokontroleru.

Pro demonstraci využití programu byl vytvořen osciloskop na několika mikrokontrolerech z řady STM32. Osciloskopy na procesorech STM32L412 a STM32F303 dosahují vzorkovací frekvence až 5 MHz. Každý z osciloskopů obsahuje generátor obdélníkového signálu (PWM) s nastavitelnou frekvencí a střídou. Osciloskop byl vytvořen i pro desku Arduino UNO, dosahuje vzorkovací frekvence až 20 kHz.

Tato práce prokázala, že je možné vytvořit univerzální zobrazovací aplikaci dosahující stejných kvalit jako existující platformy využívající aplikaci vytvořenou pro konkrétní přístroj. Při tvorbě osciloskopů pro několik různých typů mikrokontrolerů se ukázalo, že kód vytvořený pro jeden mikrokontroler lze velmi jednoduše portovat na ostatní typy. S využitím této aplikace tak bude možné vytvořit osciloskop s použitím většiny mikrokontrolerů, zejména vývojových desek Nucleo, které jsou na FEL používány ve výuce. Program je již používán při výuce v předmětu B3B38LPE na FEL a předpokládá se i jeho využití pro budoucí práce týkající se tvorby softwarově definovaných přístrojů na STM32. Zadání jsem splnil v plném rozsahu a v mnoha směrech i přesáhl.

Bibliografie

1. *LEO: Little Embedded Oscilloscope* [online]. 2020 [cit. 2020-12-03]. Dostupné z: https://embedded.fel.cvut.cz/platformy/leo.
2. *Signals & Slots* [online]. Qt Company Ltd., 2020 [cit. 2021-01-16]. Dostupné z: https://doc.qt.io/qt-5/signalsandslots.html.
3. *QMetaType Class* [online]. Qt Company Ltd., 2020 [cit. 2020-11-30]. Dostupné z: https://doc.qt.io/qt-5/qmetatype.html.
4. ZHANG, Debao. How to use QThread in the right way (Part 1). *1+1=10* [online]. 2016 [cit. 2020-01-17]. Dostupné z: http://blog.debao.me/2013/08/how-to-use-qthread-in-the-right-way-part-1/.
5. D'ANGELO, Giuseppe. *Qt World Summit 2017*. Multithreading with Qt [online]. Berlín, 2017 [cit. 2020-11-27]. Dostupné z: https://www.youtube.com/watch?v=BgqT6SIeRn4.
6. EICHHAMMER, Emanuel. *QCustomPlot* [online]. 2018 [cit. 2020-11-28]. Dostupné z: https://www.qcustomplot.com/.
7. HAOYI, Li. Build your own Command Line with ANSI escape codes. *Haoyi's Programming Blog* [online]. 2016 [cit. 2020-11-28]. Dostupné z: https://www.lihaoyi.com/post/BuildyourownCommandLinewithANSIescapecodes.html.
8. TATHAM, Simon. *PuTTY* [online]. [B.r.] [cit. 2020-11-28]. Dostupné z: https://www.chiark.greenend.org.uk/~sgtatham/putty/.
9. *Qt Style Sheets* [online]. Qt Company Ltd., 2020 [cit. 2021-04-17]. Dostupné z: https://doc.qt.io/qt-5/stylesheet.html.
10. VHALLAC. *De-interleave an array in place* [online]. 2011 [cit. 2021-04-15]. Dostupné z: https://stackoverflow.com/a/7780962/5317795.
11. EICHHAMMER, Emanuel. *QCustomPlot documentation* [online]. 2018 [cit. 2020-11-28]. Dostupné z: https://www.qcustomplot.com/documentation/index.html.
12. WEISSTEIN, Eric W. Fast Fourier Transform. *MathWorld–A Wolfram Web Resource* [online]. 2021 [cit. 2021-03-21]. Dostupné z: https://mathworld.wolfram.com/FastFourierTransform.html.

13. MAKLIN, Cory. Fast Fourier Transform. *Towards Data Science* [online]. 2019 [cit. 2021-03-21]. Dostupné z: https://towardsdatascience.com/fast-fourier-transform-937926e591cb.
14. RAMAKRISHNAN, Nipun. *Reducible*. The Fast Fourier Transform (FFT): Most Ingenious Algorithm Ever? [Online]. Berkeley, 2020 [cit. 2021-01-06]. Dostupné z: https://www.youtube.com/watch?v=h7apO7q16V0&t=1345s.
15. OPPENHEIM, Alan V.; SCHAFER, Ronald W.; BUCK, John R. *Discrete-time signal processing*. 2. vyd. New Jersey: Prentice-Hall, 1998. ISBN 0-13-754920-2.
16. ROBERTSON, Neil. *Interpolation Basics* [online]. 2019 [cit. 2021-03-29]. Dostupné z: https://www.dsprelated.com/showarticle/1293.php.
17. ARAR, Steve. FIR Filter Design by Windowing: Concepts and the Rectangular Window. *All About Circuits* [online]. 2016 [cit. 2021-04-04]. Dostupné z: https://www.allaboutcircuits.com/technical-articles/finite-impulse-response-filter-design-by-windowing-part-i-concepts-and-rect/.
18. *Fir1* [online]. MathWorks, 2021 [cit. 2021-04-04]. Dostupné z: https://www.mathworks.com/help/signal/ref/fir1.html.
19. KUHN, Matthias. *Selected Rows in QTableView, copy to QClipboard* [online]. 2018 [cit. 2021-04-02]. Dostupné z: https://stackoverflow.com/a/1230530/5317795.
20. *Build Standalone Qt Application for Windows* [online]. Qt Company Ltd., 2020 [cit. 2021-01-16]. Dostupné z: https://wiki.qt.io/Build_Standalone_Qt_Application_for_Windows.
21. *linuxdeployqt*. 2020. Dostupné také z: https://github.com/probonopd/linuxdeployqt.
22. *The last Qt version that supported Windows xp?* [Online]. 2016 [cit. 2021-04-17]. Dostupné z: https://forum.qt.io/topic/73292/the-last-qt-version-that-supported-windows-xp.
23. JRSOFTWARE. *Inno Setup* [online]. 2021 [cit. 2021-04-18]. Dostupné z: https://jrsoftware.org/isinfo.php.
24. BELLOT, Gilles. Inno Setup - Installing Prerequisites. *bell0bytes* [online]. 2019 [cit. 2021-04-18]. Dostupné z: https://bell0bytes.eu/inno-setup-vc/.
25. *High DPI Displays* [online]. Qt Company Ltd., 2020 [cit. 2021-01-16]. Dostupné z: https://doc.qt.io/qt-5/highdpi.html.
26. TEAM, AppImage. *AppImage - Concepts*. 2020. Dostupné také z: https://docs.appimage.org/introduction/concepts.html.
27. *About STM32 HAL quality and performance* [online]. 2018 [cit. 2021-04-14]. Dostupné z: https://stackoverflow.com/a/50124885/5317795.
28. WOLF, Connor. *How does the oscilloscope trigger really work?* [Online]. 2019 [cit. 2021-04-11]. Dostupné z: https://electronics.stackexchange.com/a/452077/282444.
29. HAASZ, Vladimír; HOLUB, Jan; JANOŠEK, Michal; KAŠPAR, Petr; PETRUCHA, Vojtěch. *Elektrická měření: přístroje a metody*. 3. přepracované vydání. Praha: Česká technika - nakladatelství ČVUT, 2018. ISBN 978-80-01-06412-2.
30. *AN2834 Application note: How to get the best ADC accuracy in STM32 microcontrollers* [online]. Rev 6. ST Microelectronic, 2020 [cit. 2021-04-11]. Č. AN2834. Dostupné z: https://www.st.com/resource/en/application_note/cd00211314-how-to-get-the-best-adc-accuracy-in-stm32-microcontrollers-stmicroelectronics.pdf.

31. *STM32F303x6/x8 datasheet* [online]. Rev 17. ST Microelectronic, 2015 [cit. 2020-11-27]. Č. DocID13587. Dostupné z: https://www.st.com/resource/en/datasheet/stm32f303c6.pdf.

32. *STM32L412xx datasheet* [online]. Rev 8. ST Microelectronic, 2020 [cit. 2021-02-01]. Č. DS12469. Dostupné z: https://www.st.com/resource/en/datasheet/stm32l412c8.pdf.

33. *STM32L072x8 STM32L072xB STM32L072xZ datasheet* [online]. Rev 5. ST Microelectronic, 2019 [cit. 2020-02-01]. Č. DS10689. Dostupné z: https://www.st.com/resource/en/datasheet/stm32l072cz.pdf.

34. *ATmega328P* [online]. Rev 7810D–AVR–01/15. Atmel, 2015 [cit. 2020-04-29]. Dostupné z: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf.

35. *PROGMEM* [online]. Arduino, 2021 [cit. 2020-04-29]. Dostupné z: https://www.arduino.cc/reference/en/language/variables/utilities/progmem/.

Seznam zkratek

Zkratka	Význam
AD	Analogově digitální
ADC	Analogově digitální převodník
AWDG	Analog–watchdog
CD	Kompaktní disk, optické záznamové médium pro ukládání digitálních dat
DFT	Diskrétní Fourierova transformace
DMA	Periferie umožňující přímí přístup do paměti bez intervence procesoru
FFT	Rychlá Fourierova transformace
FPU	Koprocesor pro operace s čísly v pohyblivé řádové čárce
GUI	Grafické uživatelské rozhraní
LSB	Bit nejnižšího řádu
MSB	Bit nejvyššího řádu
OS	Operační systém
PC	Osobní počítač
SDI	Softwarově definovaný přístroj
UART	Sběrnice sloužící k asynchronnímu sériovému přenosu dat
USB	Univerzální sériová sběrnice

Několik poznámek k této práci

Tento dokument je vytvořený za pomoci (mírně upravené) LaTeX šablony CTUstyle od Petra Olšáka.

Použité obrázky jsou vlastní tvorba, není-li uvedeno jinak. Jedná se především o:

- Shémata vytvořená v editoru Draw.io
- Grafy vytvořené v Matlab nebo v GeoGebra, případně upravené v Inkscape
- Snímky obrazovky z vytvářené aplikace
- Graf vyexportovaný z mé aplikace (viz kapitola 13.2)

V textu se dopouštím použití anglicismů, které sice mohou mít ryze český ekvivalent, ale v terminologii dané oblasti se běžně používají. Také při psaní desetinných čísel používám tečku namísto čárky pro zachování konzistence s ukázkami kódu a s grafy.

Příloha A

Obsah přiloženého CD a USB flash disku

- Bakalářská práce ve formátu PDF
- Zdrojový kód zobrazovací aplikace
- Aplikace zkompilovaná pro OS Windows
- Aplikace zkompilovaná pro OS Ubuntu
- Zdrojové kódy osciloskopů na STM32
- Zdrojový kód osciloskopu pro Arduino UNO
- Skript pro vytvoření instalačního souboru v Inno Setup
- Uživatelská příručka ve formátu PDF

Program je k dispozici také na GitHubu

https://github.com/jirimaier/DataPlotter

Příloha B

Uživatelská příručka k programu

DATA PLOTTER

Uživatelská příručka

Jiří Maier

OBSAH

PROPOJENÍ SE ZAŘÍZENÍM

Program vyhledá dostupné COM porty a zobrazí je v seznamu (včetně názvu zařízení, pokud je k dispozici). Seznam se průběžně automaticky aktualizuje. Při spuštění se pokusí najít a vybrat port, který má v popisu "ST" (tedy se pravděpodobně jedná o Nucleo). Popis nebo název výchozího portu lze změnit (viz. nastavení)

Na Linuxu se může stát, že aplikace nemá přístup k portu. To lze vyřešit pomocí příkazů:

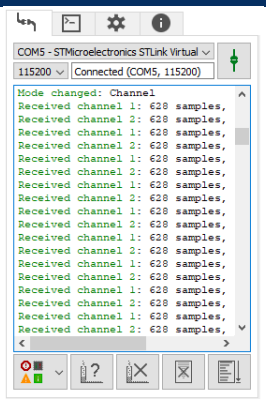
```
sudo usermod -a -G tty $USER
sudo usermod -a -G dialout $USER
```

Kromě přednastavených baudrate lze do pole napsat jakékoli jiné číslo. Také lze nastavit další parametry jako například paritu, kliknutím na tlačítko se symbolem nastavení. Pokud je port připojen, jakákoli změna nastavení vyvolá odpojení. Tlačítko informace zobrazí všechny zjistitelné údaje o portu.

PŘÍJEM DATA ZE ZAŘÍZENÍ

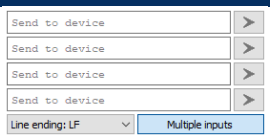
V textovém poli se zobrazují informace o přijatých zprávách a případná chybová hlášení a také informační a varovné zprávy ze zařízení. Lze nastavit úroveň výpisu:

- Jen zprávy ze zařízení (**`$$I`** a **`$$W`**)
- Zprávy ze zařízení a chyby
- Zprávy ze zařízení, chyby a varování
- Všechna oznámen



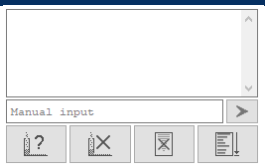
ODESLÁNÍ DO ZAŘÍZENÍ

Pro odeslání do zařízení stačí napsat text do textového pole a potvrdit klávesou enter, nebo kliknutím na tlačítko. Na konec textu je přidán znak zakončení řádku dle výběru, pokud je nějaký zvolen.
Odesílání má dva režimy: Jedno pole, které je vymazáno po odeslání, nebo čtyři pole, v kterých text zůstane (je tak možné mít připraveno několik základních příkazů.



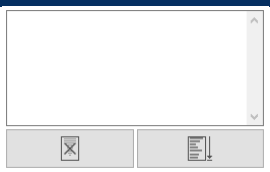
MANUÁLNÍ VSTUP PRO TESTOVÁNÍ

Manuální vstup lze zobrazit zaškrtnutím políčka na stránce nastavení. Umožňuje ručně zadat data do textového pole a zpracovat stejně jako by šlo o data ze zařízení. V informacích u manuálního vstupu se také zobrazují informace o nastavení načteném ze souboru. Znaky, které nejsou v základním ASCII se nemusí zpracovat správně (není vhodné pro testování binárních dat).



MONITOROVÁNÍ SÉRIOVÉHO PORTU

Pole kam se vypisuje veškerý text přijatý ze sériového portu lze zobrazit zaškrtnutím políčka na stránce nastavení.



PŘÍKAZ PO PŘIPOJENÍ

Po připojení může být žádoucí zařízení resetovat, například aby poslalo příkaz k vykreslení GUI do terminálu. K tomu lze využít tuto funkci. Pokud je políčko zaškrtnuté a textové pole není prázdné, bude text z pole po připojení odeslán do sériového portu. Text lze uložit do souboru s výchozím nastavením, aby se při spuštění programu nastavil. Do textového pole lze zapsat i znak nový řádek, stačí napsat „\n", případně „\r".

PROTOKOL PRO PŘENOS DAT

Zprava začíná dvojicí znaků **$$**, po nich následuje písmeno označující typ zprávy:

- **$$P** přidání bodu do grafu
- **$$C** přidání celého kanálu do grafu
- **$$L** přidání logického kanálu do grafu
- **$$B** přidání logického bodu do grafu
- **$$T** výpis do terminálu
- **$$S** nastavení
- **$$I** výpis informační zpráv
- **$$W** varovná zpráva
- **$$X** chyba zařízení (zobrazí zprávu a odpojí zařízení)
- **$$E** echo (pošle přijatý text zpět do zařízení, pro otestování připojení ze strany zařízení)
- **$$U** neznámý (data jsou zahozena, nezpracují se)

Písmeno označující typ zprávy není case-sensitive.

VÝPIS DO TERMINÁLU

Data jsou vypisována do terminálu podporujícího ANSI escape sekvence. Vypisují se průběžně tak jak přichází. Zakončeno je až začátkem další zprávy. Nesmí obsahovat sekvenci **$$**, jeden **$** se vyskytnout smí.

$$T\u001b[31;1mAAA \u001b[32;1mBBB \u001b[33;1mCCC\n\r
Vypíše AAA BBB CCC a odřádkuje.

VYPSÁNÍ INFORMAČNÍ VAROVNÉ A CHYBOVÉ ZPRÁVY

Text informační a varovné se zobrazí v textovém poli, kam se vypisují zprávy o zpracovaných datech. Informační je označena zeleně, varovná červeně. Chybová zpráva se zobrazí ve vyskakovacím okně a odpojí port.

Chybová zpráva je zakončena středníkem. Informační a varovné jsou zakončeny až začátkem další zprávy, středník se nepoužívá (takže může být obsažen v textu). Nesmí obsahovat sekvenci **$$**. Jeden **$** se vyskytnout smí.

```
$$IToto je informace      $$WToto je varování      $$XToto je error;
```

NASTAVENÍ

Nastaví parametry grafu a GUI. Seznam dostupných nastavení je na konci tohoto dokumentu.

- Nastavení má tvar: **identifikátor:hodnota**
- Pro nastavení analogového nebo math kanálu: **ch:čísloKanálu:identifikátor:hodnota**
- Pro nastavení logických kanálů **log:čísloLogiky:identifikátor:hodnota**

Každé nastavení je zakončeno středníkem. V jedné zprávě lze mít více nastavení. Identifikátory nejsou case-sensitive. Číselné hodnoty jsou vždy zapsány jako číslo, binární reprezentace zde není možná.

Kanály jsou číslované od 1, pro matematické kanály jsou čísla 17, 18, 19. Logické kanály jsou číslovány 1 a 2, pro hlavní logický kanál (v seznamu kanálů nemá číslo) použijte číslo 3.

$$Svrange:100;
Nastaví svislý rozsah na 100.

BOD

Přidá data do analogových kanálů po jednotlivých bodech v desítkové nebo binární reprezentaci.

$$P(čas),(ch1),(ch2),(ch3);

- Čas:
  - Hodnota (číslo nebo binárně): čas (souřadnice x) bodu
  - Speciální příkazy:
    - "-": Index vzorku od připojení
    - "-auto": Čas od připojení
    - "-tod": Čas dne (sekundy od půlnoci) (TOD = time of day)
- Ch1... Ch16 (maximálně 16 hodnot)
  - Hodnota (číslo nebo binárně): hodnota kanálu v tomto bodě
  - Speciální příkazy:
    - "-": Kanál nemá v tomto čase žádnou hodnotu

Po sobě jdoucí binární hodnoty není potřeba oddělit čárkou.

PŘÍKLADY

ČÍSELNÝ ZÁPIS

$$P123.00,1.10,2.20,3.30;

V čase 123.00 má kanál 1 hodnotu 1.10, kanál 2 má hodnotu 2.20 a kanál 3 hodnotu 3.30.

$$P123.00,1.10,-,3.30;

V čase 123.00 má kanál 1 hodnotu 1.10, kanál 2 nemá žádnou hodnotu a kanál 3 hodnotu 3.30.

$$P-,1.10,2.20,3.30;

Časová souřadnice tohoto bodu se rovná pořadí bodu od připojení (začne v čase 0 a pro každý následující bod se zvýší o jedna).

$$P-auto,1.10,2.20,3.30; nebo $$P-tod,1.10,2.20,3.30;

Časová souřadnice tohoto bodu se rovná času od připojení (nebo času dne v druhém případě), ve kterém byl tento bod přijat.

BOD ZAPSANÝ BINÁRNĚ

$$PU2??U2??U2??U2??;

Čas a tři hodnoty v unsigned integer typu. Všimněte si, že není nutné oddělovat hodnoty čárkou (ale čárky mohou být použity).

KOMBINOVANÝ ZÁPIS

$$PU2??U2??,123.00,U2??;

Čas a kanály 1 a 3 mají hodnoty v unsigned integer typu, kanál 2 má hodnotu 123.00, všimněte si, že číselná hodnota je oddělena čárkami.

$$PU2??U2??,-,U2??;

Čas a kanály 1 a 3 mají hodnoty v unsigned integer typu, kanál 2 je v tomto okamžiku prázdný.

$$P-,U2??U2??U2??;

Časová souřadnice tohoto bodu se rovná pořadí bodu od připojení (začne v čase 0 a pro následující bod se zvýší o jedna).

KANÁL

Přidá celou sadu dat do jednoho analogového kanálu, data jsou binární.

$$C(záhlaví);(datový typ)(data..........);

V závislosti na datovém typu lze použít různé typy záhlaví:

UNSIGNED INT

$$C(ch),(časový krok),(délka);U?(data..........);
$$C(ch),(časový krok),(délka),(bity),(max);U?(data..........);
$$C(ch),(časový krok),(délka),(bity),(min),(max);U?(data..........);
$$C(ch),(časový krok),(délka),(bity),(min),(max),(index nuly);U?(data....);

SIGNED INT NEBO FLOATING PIONT

$$C(ch),(časový krok),(délka);F?(data..........);
$$C(ch),(časový krok),(délka),(index nuly);F?(data..........);

- Ch: kladné celé číslo (číslo nebo binárně): kanál pro zápis dat v (1 ... 16), nebo více čísel oddělených ‚+‘
- Časový krok: hodnota (číslo nebo binárně): časový interval mezi po sobě jdoucími vzorky
- Délka: kladné celé číslo (číslo nebo binárně): počet vzorků (nikoli bajtů) v tomto kanálu
- Bity: kladné celé číslo (číslo nebo binárně): počet využitých bitů v hodnotě (pro výpočet min a max)
- Min: hodnota (číslo nebo binárně): hodnoty budou přemapovány tak, aby 0 odpovídala této hodnotě
- Max: hodnota (číslo nebo binárně): hodnoty budou přemapovány tak, aby 2^bity odpovídalo této hodnotě
- Index nuly: kladné celé číslo nebo nula (číslo nebo binárně): index vzorku, který odpovídá času 0. Pokud je vynecháno, první vzorek (index 0) je v čase 0. Užitečné pro pretrigger.

Po sobě jdoucí binární hodnoty není potřeba oddělit čárkou.

PŘÍKLADY

JEDNODUCHÁ VARIANTA S UNSIGNED INTEGER HODNOTAMI

$$C1,0.001,20;U2??;

Data kanálu 1, interval mezi vzorky je 0.001 sekundy (první vzorek je v čase nula), kanál má 20 vzorků v 16bitovém unsigned integer (40 bajtů za "U2").

HODNOTY V UNSIGNED INTEGER S PŘEMAPOVÁNÍM

$$C1,0.001,20,12,-1.5,1.5;U2????????????????????????????????????;

Data pro kanál 1, interval mezi vzorky je 0.001 sekundy (první vzorek je v čase nula), kanál má 20 vzorků v 16bitovém unsigned integer. Hodnoty jsou přemapovány tak, že hodnota 4096 (2^12) odpovídá 1.5V a hodnota 0 odpovídá -1,5.

HODNOTY VE FLOATING POINT

$$C1,0.001,10,5;F4????????????????????????????????????;

Data pro kanál 1, interval mezi vzorky je 0.001 sekundy (první vzorek je v čase nula), kanál má 10 vzorků v 32bitovém float. Vzorek s indexem 5 (počítáno od nuly) je v čase 0, vzorky před ním jsou v záporných časech.

VÍCE KANÁLŮ NA PŘESKÁČKU

$$C1+2+3+4,0.001,24;U2??;

Data jsou pro kanály 1, 2, 3 a 4. Hodnoty se v tomto pořadí střídají (první dvojice bajtů je pro kanál 1, druhá dvojice pro kanál 2...). Zadaná délka je počet vzorků všech kanálů dohromady (v tomto příkladu má každý kanál 8 vzorků).

LOGICKÝ KANÁL

Přidá celou sadu dat do logických kanálů, hodnoty jsou binární, typu unsigned int.

$$C(záhlaví);(datový typ)(data..........);

$$C(časový krok),(délka);U?(data..........);
$$C(časový krok),(délka),(bity);U?(data..........);
$$C(časový krok),(délka),(bity),(index nuly);U?(data..........);

- Časový krok: hodnota (číslo nebo binárně): časový interval mezi po sobě jdoucími vzorky
- Délka: kladné celé číslo (číslo nebo binárně): počet vzorků (nikoli bajtů) v tomto kanálu
- Bity: kladné celé číslo (číslo nebo binárně): počet bitů, které se mají zobrazit (počínaje LSB)
- Index nuly: kladné celé číslo nebo nula (číslo nebo binárně): index vzorku, který odpovídá času 0. Pokud je vynecháno, první vzorek (index 0) je v čase 0. Užitečné pro pretrigger.

Po sobě jdoucí binární hodnoty není potřeba oddělit čárkou.

PŘÍKLADY

$$C0.001,20;U2??;
Interval mezi vzorky je 0.001 sekundy (první vzorek je v čase nula), kanál má 20 vzorků v 16bitovém unsigned integer (40 bajtů za "U2"), zobrazí se všech 16 bitů.

$$C0.001,20,12;U2??;
Interval mezi vzorky je 0.001 sekundy (první vzorek je v čase nula), kanál má 20 vzorků v 16bitovém unsigned integer, je zobrazeno pouze posledních 12 bitů.

$$C0.001,20,16,10;U2??;
Interval mezi vzorky je 0.001 sekundy (první vzorek je v čase nula), kanál má 20 vzorků v 16bitovém unsigned integer, je zobrazeno všech 16 bitů. Vzorek s indexem 10 (počítáno od nuly) je v čase 0, vzorky před ním jsou v záporných časech.

LOGICKÝ BOD

Přidá data do kanálů logiky po jednotlivých bodech.

`$$P(čas),(hodnota),(bity);`

- Čas:
  - Hodnota (číslo nebo binárně): čas (souřadnice x) bodu
  - Speciální příkazy:
    - "-": Index vzorku od připojení
    - "-auto": Čas od připojení
    - "-tod": Čas dne (sekundy od půlnoci)
- Hodnota: hodnota v unsigned integer
- Bity: kladné celé číslo (číslo nebo binárně): počet bitů, které mají být zobrazeny (počínaje LSB)

Po sobě jdoucí binární hodnoty není potřeba oddělit čárkou.

PŘÍKLADY

`$$P123.00,U2??;`

Čas je 123.00, 16bitová logická hodnota.

`$$PU2??U2??;`

Čas jako unsigned integer. 16bitová logická hodnota.

`$$PU2??U2??,12;`

Čas jako unsigned integer. 12bitová logická hodnota.

`$$P-auto,U2??;` nebo **`$$P-tod,U2??;`**

Časová souřadnice tohoto bodu se rovná času od připojení (nebo času dne v druhém případě), ve kterém byl tento bod přijat.

ČÍSELNÉ HODNOTY

Používá se desetinná tečka.

```
123.45
```

Číslo je také možné zapsat ve vědecké notaci

```
1.23e-3
```

Vždy musí začínat číslicí:

`1e-3`: správně
`e-3`: špatně

PŘÍKLAD ODESLÁNÍ ČÍSELNÉ HODNOTY

MBED:

```
serial.printf("$$P%u.0,%.3f,%.3f;",millis(),value1,value2);
```

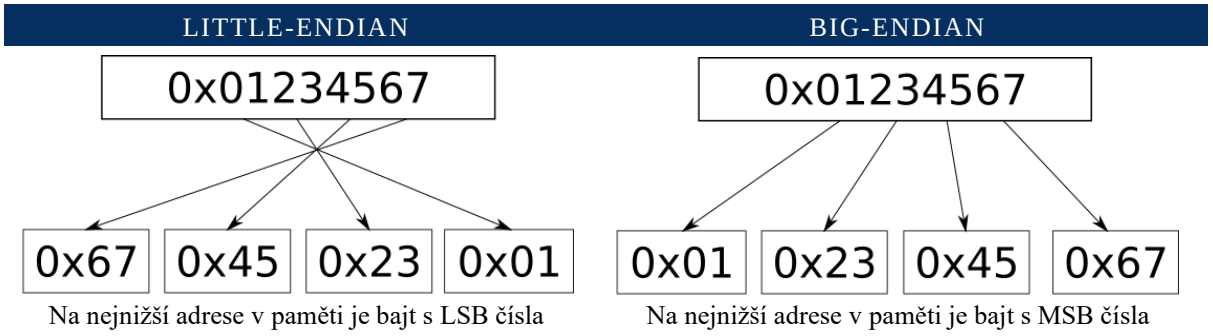
BINÁRNÍ HODNOTY

Při posílání dat v binárním tvaru je nutné před samotnými bajty čísla uvést datový typ:

Little-endian		Big-endian	
u1	8bitový unsigned integer	**U1**	8bitový unsigned integer
u2	16bitový unsigned integer	**U2**	16bitový unsigned integer
u3	24bitový unsigned integer	**U3**	24bitový unsigned integer
u4	32bitový unsigned integer	**U4**	32bitový unsigned integer
i1	8bitový signed integer	**I1**	8bitový signed integer
i2	16bitový signed integer	**I2**	16bitový signed integer
i4	32bitový signed integer	**I4**	32bitový signed integer
f4	float	**F4**	float
f8	double	**F8**	double

LITTLE-ENDIAN A BIG-ENDIAN

Většina platforem je little endian



Na nejnižší adrese v paměti je bajt s LSB čísla

Na nejnižší adrese v paměti je bajt s MSB čísla

PŘÍKLAD ODESLÁNÍ HODNOTY BINÁRNĚ

MBED:

```
float value = 123.45;
char *value_bytes = (char *)&value;
serial.printf("f4");
serial.putc(value_bytes[0]);
serial.putc value_bytes[1]);
serial.putc(value_bytes[2]);
serial.putc(value_bytes[3]);
```

HODNOTY S JEDNOTKOU

Pokud například chceme poslat celočíselnou hodnotu v mV, je možné to udělat tak, že před identifikátor datového typu přidám písmeno **m**, hodnota poslaná s touto předponou bude vydělena tisícem.

Možné předpony jsou: T, G, M, k, h, D, d, c, m, u, p, f, a.

PŘÍKLAD

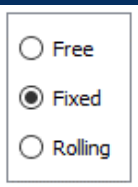
```
$$C1,uU2??,20;mU2????????????????????????????????????????;
```

Interval mezi vzorky je 16bitový unsigned integer v µs, hodnoty kanálu jsou 16bitový unsigned integer v mV.

GRAF

V grafu lze zobrazit až 16 analogových kanálů, 3 matematické a 3 skupiny logických kanálů s maximálně 32 bity. Dvě skupiny logických kanálů jsou určeny pro převedení celočíselného analogového vstupu na logické kanály (zobrazení bitů AD převodníku), třetí je určen pro přímí logický vstup pomocí zpráv **`$$L`** nebo **`$$B`**.

REŽIMY

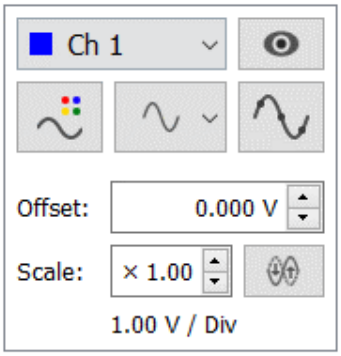


Free (volný): lze posouvat / přibližovat myší, přibližovat lze buď celý graf, nebo individuálně roztáhnout svislou nebo vodorovnou osu.

Fixed (pevný): zobrazí celý časový rozsah přijatého signálu. Vhodné pro průběhy, které se překreslují stále na stejném časovém úseku.

Rolling (posuvný): Zobrazí pouze úsek na konci, graf se odsouvá doleva. Vhodné pro průběhy, které přibývají dál v čase a starší hodnoty v grafu zůstávají.

NASTAVENÍ KANÁLU



Nahoře se vybírá nastavovaný kanál, je u něho zobrazena jeho barva pro snadnou identifikaci, barvu lze změnit. Také lze vybrat styl zobrazení (čára, body), kanál lze také skrýt.

Pro analogové kanály lze aktivovat funkci interpolace, kdy je průběh převzorkován na vyšší vzorkovací frekvenci. U nastavení matematických kanálů lze nastavit kvalitu.

Ve výchozím nastavení lze vybírat jen z kanálů které jsou aktuálně používány, to lze změnit v nastavení.

V poli uprostřed se nastavuje offset, tedy svislé posunutí. Pokud je nenulový, v grafu se zobrazí čárkovaná čára v barvě kanálu, která ukazuje kde se nulová hodnota kanálu právě nachází. Offset lze rovněž měnit tažením myší za čáru která označuje nulovou úroveň kanálu.

Pod offsetem se nastavuje svislé roztažení kanálu (vynásobení všech hodnot nastaveným číslem). Také lze kanál invertovat. Roztažení ani invertování nemá vliv na hodnoty, jaké zobrazí kurzory.

Dole se zobrazuje měřítko kanálu (rozdíl hodnot odpovídající jednomu kroku mřížky), to se mění podle nastavení mřížky, a roztažení kanálu.

OVLÁDÁNÍ GRAFU

Tlačítko pauza pozastaví (nebo rozeběhne pozastavený) graf. V průběhu pauzy jsou nová data i nadále zpracovávána a po ukončení pauzy budou přidána do grafu.

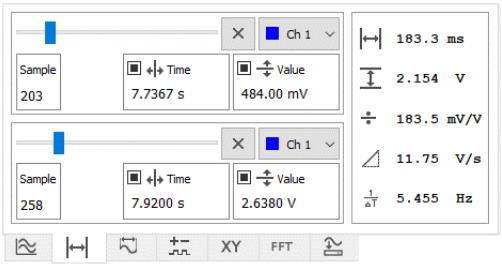
Prostřední tlačítko se pokusí automaticky nastavit rozsah a pomocí offsetů rozloží více kanálů nad sebe, lze použít i pro logické kanály. V záložce nastavení lze zvolit, aby se autoset provedl automaticky po připojení.

Tlačítko vpravo vynuluje offset a nastaví zvětšení na 1 u všech kanálů.

KURZORY

K dispozici jsou dva páry svislých a vodorovných kurzorů. Každý kurzor má tři režimy (podle třístavového zaškrtávacího políčka):

- ▪ Posazený na vzorku
- ✓ Volný
- ○ Skrytý



Ve volném režimu se nastaví čas, respektive hodnota (napětí), na jakém má časový nebo napěťový kurzor být. V režimu vázaném k vzorku se pomocí posuvníku nastaví vzorek, na kterém je posazen časový kurzor, pokud je napěťový kurzor také v tomto režimu, je posazen na hodnotu toho vzorku.

Časový kurzor může být ve volném režimu pouze pro analogové kanály (ne logické a FFT). Pro logický kanál je k dispozici pouze časový kurzor.

Kurzory lze také ovládat myší. Kliknutím na graf se dvojice kurzorů (časový a napěťový) přesune na dané místo (nejbližší vzorek) kanálu na který bylo kliknuto, levé tlačítko pro první, pravé pro druhý). Poté lze kurzory táhnout myší. Pokud je popotažen napěťový kurzor, přepne se do volného režimu. Časový kurzor při tažení zůstává v původním režimu.

Tyto kurzory jsou společné pro hlavní graf a FFT, XY režim má samostatné kurzory na své stránce.

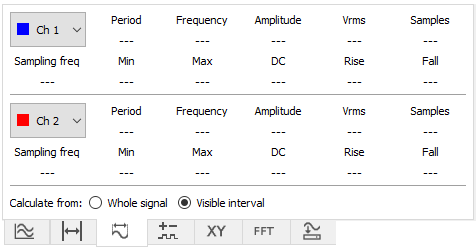
Hodnota napěťového kurzoru je relativně vůči offsetu zvoleného kanálu, ale nezávislá na nastavení zvětšení (zvětšení pouze roztáhne graf, ale hodnoty neovlivní) a na invertování kanálu (pokud je kanál invertovaný, jsou hodnoty kladné směrem dolů). Kurzory lze také nastavit do absolutního režimu, kde zobrazují hodnoty vůči osám grafu.

MĚŘENÍ

Na této stránce se zobrazují údaje o měřeném signálu.

Lze zvolit dva kanály (Kromě základních kanálů lze zvolit i kanál výpočtu), pro které se údaje budou počítat.

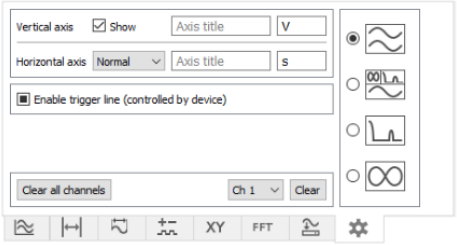
Hodnoty mohou být vypočteny z celého kanálu (Whole signal), nebo z rozsahu který je zobrazen v grafu (Visible interval).



Doba vzestupné a sestupné hrany se počítá vždy na poslední periodě signálu/zobrazeného úseku. Pokud je hodnota zobrazena se znaménkem „menší než“, je hrana kratší než interval mezi vzorky.

NASTAVENÍ GRAFU

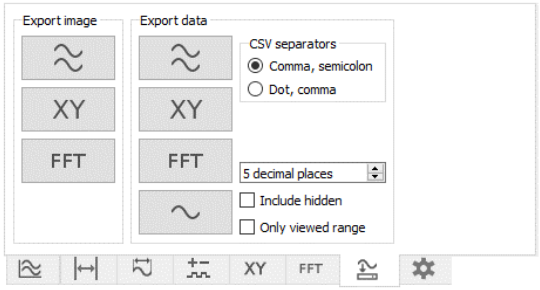
Lze nastavit popisky os a jednotku hodnot na ose (jednotku zadejte jako základní bez předpony, předpony jsou doplněny automaticky). Lze zobrazit nebo skrýt svislou osu (pokud je více průběhu nad sebou, jsou čísla na ose irelevantní). Horizontální osu lze také skrýt. Pokud je jednotka „s“ (sekunda), lze nastavit, aby se údaj v zobrazoval ve formátu MM:SS nebo i HH:MM:SS.



Čára triggeru slouží pro zobrazení úrovně triggeru pomocí příkazů ze zařízení (viz. tabulka nastavení) a má tři režimy: nezaškrtnuto – není zobrazena, zaškrtnuto – vždy zobrazena, čtvereček – zobrazí se dočasně při změně

Lze nastavit režim zobrazení grafů (hlavní, všechny, XY, FFT). Mezi hlavním a všemi se přepíná automaticky.

EXPORT



Exportovat do souboru ve formátu CSV lze jeden vybraný kanál, nebo všechny, nebo XY kanál.

V závislosti na nastavení systému program Excel používá buď desetinou tečku, nebo čáku, aby soubor načetl správně, je nutné správně vybrat typ oddělovače. Možnosti jsou:

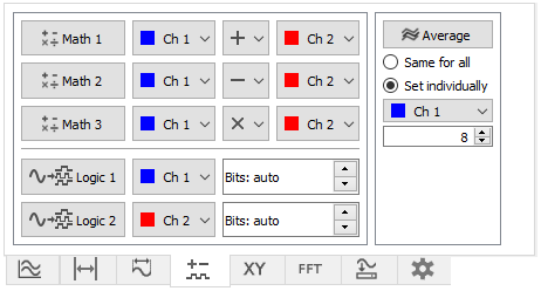
- desetinná tečka, odděleno čárkou
- desetinná čárka, odděleno středníkem

Je-li zaškrtnuto „Include hidden“, budou exportovány i skryté kanály. Je-li zaškrtnuto „Only viewed range“, bude exportován pouze aktuálně zobrazený úsek (dle úseku na vodorovné ose, svisle není omezeno).

Také je možné graf uložit jako obrázek ve formátu PNG.

Po kliknutí na tlačítko bude uživatel dotázán, jestli mají data být uložena do souboru nebo zkopírována do schránky (pro Ctrl+V). V případě kopírování CSV do schránky je jako oddělovač použit tabulátor (funguje pro vložení do Excelu a podobných programů).

VÝPOČTY A LOGICKÉ KANÁLY



Kanály je možno sčítat, odčítat, násobit, dělit

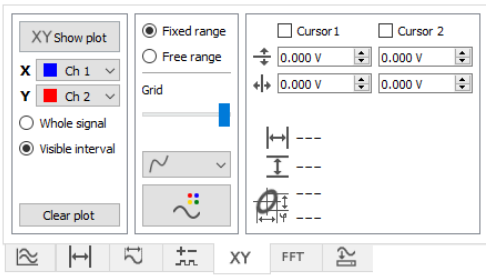
Výpočet se aktivuje kliknutím na tlačítko příslušného kanálu.

Kanály posílané v režimu celého kanálu s přepočtem (nebo jako celočíselné hodnoty) lze zobrazit i jako logický kanál. Pokud je počet bitů nastaven na automatický, je použit počet bitů uvedený v záhlaví zprávy.

Matematický kanál se po aktivaci dopočítá zpětně pro všechny předchozí body (pokud jsou data přidávána po bodech), logické kanály se zpracují jen pro nově příchozí data (nezobrazí se, pokud je graf pozastaven).

Také je možné průměrovat hodnoty kanálů. Průměrování se aktivuje tlačítkem, počet kanálů/bodů pro průměrování lze nastavit pro všechny stejné, nebo pro každý individuálně. Pro data přidávaná po celých kanálech (**$$C**) jsou průměrovány vzorky nejnovějších průběhů. Pro přidávání po bodech (**$$P**) se toto chová jako klouzavý průměr.

X-Y REŽIM



Graf X-Y režimu je zobrazen v samostatném grafu (automaticky se zobrazí při zapnutí tohoto režimu a skryje se při jeho vypnutí).

Kromě základních kanálů lze zvolit i kanál výpočtu (Math).

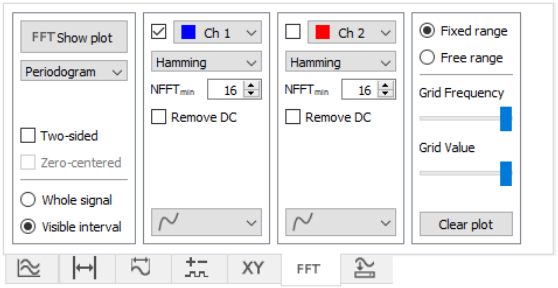
Lze zvolit pevný nebo volný rozsah, ve volném režimu lze posouvat a přibližovat myší.

Také lze nastavit krok mřížky a změnit zobrazení grafu (čára nebo body) a barvu grafu.

XY graf může být vypočten z celých kanálů (Whole signal), nebo z rozsahu který je zobrazen v hlavním grafu (Visible interval).

V XY grafu lze použít dva páry kurzorů. Lze je ovládat myší stejně jako kurzory v hlavním grafu.

FFT



Spektrum je zobrazeno v samostatném grafu (automaticky se zobrazí při zapnutí tohoto režimu a skryje se při jeho vypnutí). Je možné zobrazovat spektrum dvou průběhů současně.

Kromě základních kanálů lze zvolit i kanál výpočtu (Math).

Lze zvolit pevný nebo volný rozsah, ve volném režimu lze posouvat a přibližovat myší.

Také lze nastavit krok mřížky a změnit zobrazení grafu (čára nebo body), barva je zvolena automaticky podle grafu, z kterého je spektrum počítáno.

FFT graf může být vypočten z celého kanálu (Whole signal), nebo z rozsahu který je zobrazen v hlavním grafu (Visible interval).

Lze zvolit tři typy výpočtu: spektrum (lineární), periodogram v dB a periodogram vypočtený pomocí Welchovy metody.

TERMINÁL

Terminál umožňuje vypisování textu a podporuje ANSI escape sekvence, což umožňuje v terminálu vytvořit pseudografické uživatelské rozhraní pro přehledné zobrazení naměřených hodnot a podobně.

Zaručená minimální šířka terminálu je 14 znaků, doporučuji ji při návrhu nepřekročit (ale lze ho roztáhnout popotažením rozhraní mezi plochou grafu a panelem vpravo). Ve svislém směru lze terminál posouvat kolečkem myši.

Kromě samotného vypisování přijatého textu má terminál tři ovládací režimy:

INTERAKTIVNÍ OVLÁDÁNÍ

Kliknutím na písmeno (znak) v terminálu je tento znak odeslán do zařízení, lze takto vytvořit menu pro ovládání zařízení pomocí klikání myší.

Ve výchozím nastavení nelze odeslat znak s černým pozadím (předpokládá se, že jde o popisek). Seznam zakázaných barev pozadí lze upravit v nastavení a pomocí příkazu **`noclickclr`**, za kterým je seznam ANSI sekvencí barev. Sekvence nemusí obsahovat **`\e[`** a **`m`**. Protože příkaz je zakončen středníkem, není v seznamu možné použít středníky, a proto jsou nahrazeny tečkami:

```
$$Snoclickclr:40,41.1,48.5.34;
```

PŘÍKLAD VYTVOŘENÍ INTERAKTIVNÍHO MENU

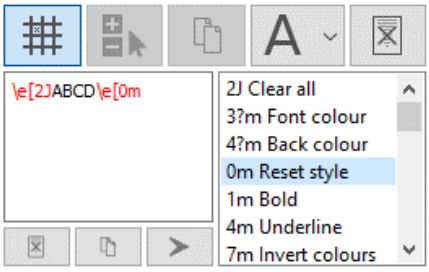
Kliknutím na **`+`** nebo **`-`** bude znak **`+`** respektive **`-`** odeslán do zařízení, které ho může zpracovat jako příkaz ke zvýšení nebo snížení nastavení hodnoty.

OZNAČENÍ A KOPÍROVÁNÍ

Pomocí myši lze označit text a tlačítkem ho zkopírovat do schránky.

NÁVRH A ODLADĚNÍ

Tento režim umožnuje ručně zadat text do terminálu, včetně zadání escape sekvencí a řídicích znaků vybraných ze seznamu. Text zadaný do textového pole zle kliknutím na tlačítko vypsat do terminálu. Také ho lze zkopírovat do schránky pro následné použití ve firmware pro zařízení.



V terminálu je v tomto režimu zobrazena pozice kursoru. Kliknutím do terminálu se kurzor přemístí na příslušnou pozici (a do textového pole je přidán odpovídající příkaz pro přesunutí kursoru).

To terminálu lze zapsat i znaky v UTF-8, pokud do textového pole zadáte znak, který není v ASCII, po zkopírování do schránky tlačítkem kopírovat je nahrazen sekvencí pro zápis tohoto znaku jako textového řetězce. Například písmeno **`á`** bude zkopírováno jako **`\xc3""\xa1""`**.

NASTAVENÍ

Nastavení programu je možné načíst ze souboru a uložit do souboru. Soubor s nastavením je prostý text s příponou **`.cfg`** (config). Nastavení je ve stejném formátu jako nastavovací příkazy (**$$S**). Jednotlivé příkazy jsou oddělené středníkem. V souboru může být každý příkaz na novém řádku, i tak je však potřeba použít středníky.

Po spuštění se program pokusí načíst výchozí nastavení (soubor **./settings/default.cfg**, vytvořen při prvním spuštění) ten může uživatel přepsat vlastním nastavením, také ho lze načíst ručně tlačítkem vlevo. Pokud soubor neexistuje, jsou použita výchozí nastavení zabudovaní v programu, která uživatel nemůže měnit. Druhé tlačítko zleva načte tato neměnná výchozí nastavení a zároveň jimi přepíše soubor s uživatelským výchozím nastavením.

Také je možné načíst analogové kanály ze souboru CSV.

SEZNAM PŘÍKAZŮ PRO NASTAVENÍ

Identifikátor	Význam nastavovaného parametru	Typ a rozsah hodnoty
autoautoset	Autoset po připojení	0/1
baud	Baudrate	(číslo)
clearch	Vymazat kanál	1…16
clearlog	Vymaže logický kanál (ten pro přímé přidání)	(žádný)
clearonrec	Vymazat graf po připojení	0/1
debuglvl	Úroveň výpisu	(index) 0~jen zařízení …3~vše
haxis	Typ časové osy	(index) 0~skrytá…3~HH:MM:SS
hlabel	Popisek časové osy	(text)
hrange	Rozsah času v rolling režimu	0.001…1000000
hunit	Jednotka vodorovné osy	(text)
layout	Zobrazené grafy	„time“, „all“, „xy“, „fft“
manualin	Zobrazit manuální vstup	0/1
multisend	Více řádků pro odeslání	0/1
noclickclr	Seznam barev pozadí znaků, které nelze odeslat	(viz kapitola Interaktivní ovládání)
noopengldialog	Nebude se zobrazovat upozornění na OpenGL	(žádný)
nofreeze	Automatické vypnutí výpisu, hrozí-li zaseknutí	0/1
opengl	OpenGL	0/1
presetport	Název nebo popis výchozího portu	(text)
rstcmd	Příkaz k poslání po připojení	(text)
send1	Předvyplní řádek pro odeslání	(text)
send2	Předvyplní řádek pro odeslání	(text)
send3	Předvyplní řádek pro odeslání	(text)
send4	Předvyplní řádek pro odeslání	(text)
sendend	Zakončení odeslaného řádku	(index) 0~nic…3~CRLF
sendonrec	Po připojení odeslat (rstcmd)	0/1
serialmon	Zobrazit serial monitor	0/1
vaxis	Zobrazit hodnoty na svislé ose	0/1
vlabel	Popisek svislé osy	(text)
vpos	Svislá pozice nuly v grafu	-100~jen záporné…100~jen kladné
vrange	Rozsah hodnot	0.000001…1000000
vunit	Jednotka hodnot na svislé ose	(text)
plotrange	Typ rozsahu grafu	fix, free, roll
terminal	Interaktivní režim terminálu	clicksend, select, nointeract
trigline	Režim čáry zobrazující úroveň triggeru	„on“, „off“, „auto“
trigch	Kanál, na kterém je trigger	1…16
trigpos	Hodnota triggeru	(číslo)
lang	Jazyk GUI	cz, en
csvsep	Oddělovače pro CSV	cs, dc
xyclr	Barva XY grafu	“0,0,0”…”255,255,255”
ch:?:sty	Styl kanálu (?=1…16 nebo 17…19 pro math)	0~line…5~squareFilled
ch:?:clr	Barva kanálu (?=1…16 nebo 17…19 pro math)	“0,0,0”…”255,255,255”
log:?:sty	Styl log. kanálů (?=1/2 = Logic1/2, ?=3=Logic)	0~line…5~squareFilled
log:?:clr	Barva log. kanálů (?=1/2 = Logic1/2, ?=3=Logic)	“0,0,0”…”255,255,255”