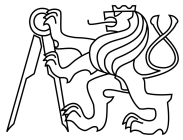
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Aplikace pro spolehlivé ovládání hardware s pomocí sériové komunikace
Student:	Bc. Martin Chudoba
Vedoucí:	Ing. Pavel Kubalík, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Cílem práce je návrh protokolu pro bezpečnou komunikaci po sériových linkách a návrh a implementace modulu/jádra, který komunikaci řídí.

Základní funkční a nefunkční požadavky jsou:
1. Analyzujte požadavky a proveďte návrh protokolu a modulu pro komunikaci tak, aby bylo možné přenášet různě velké struktury dat a to jak binárně, tak i textově.
2. Diskutujte a zvolte vhodnou implementační platformu.
3. Návrh implementujte.
4. Navrhněte a implementujte demonstrační aplikaci, která začlení a použije komunikační modul.
5. Zhodnoťte přínos řešení.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 2. listopadu 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Aplikace pro spolehlivé ovládání hardware s pomocí sériové komunikace

Bc. Martin Chudoba

Vedoucí práce: Ing. Pavel Kubalík, Ph.D.

30. června 2017

Poděkování

Chtěl bych poděkovat vedoucímu práce panu Ing. Kubalíkovi, Ph.D. za pomoc a rady s tvorbou této práce. Dále děkuji své rodině a přátelům za podporu v průběhu studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 30. června 2017 .....

Odkaz na tuto práci

Chudoba, Martin. *Aplikace pro spolehlivé ovládání hardware s pomocí sériové komunikace*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Tato práce se zabývá návrhem a implementací knihovny pro bezpečnou komunikaci po sériové lince. Sériová linka je tvořena USB/UART převodníkem s čipem FT232R.

Klíčová slova FT232R, spolehlivý software, seriová komunikace

Abstract

This thesis deals with design and implementation of library for reliable communication using serial line. Serial line is composed of USB/UART converter with FT232R chip.

Keywords FT232R, reliable software, serial communication

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V dnešní době existuje mnoho zařízení, které lze ovládat pomocí sériové linky. Zmiňme například Arduino nebo Raspberry PI a hned se nabízí nezměrné množství zařízení.

Pro některá z těchto zařízení bude důležitá spolehlivost přenosu dat po sériové lince, protože v případá chyba může způsobit finanční ztráty, ohrožení životního prostředí nebo lidského života.

Cílem této práce je poskytnout řešení pro spolehlivou komunikaci po sériové lince. Řešením bude postavení knihovny, která zaručí integritu a spolehlivost přenášených dat.

Členění textu

Tato práce je členěna na kapitoly. První kapitola (1) obsahuje cíl práce. Druhá kapitola (2) se zabývá obecně vývojem bezpečného softwaru, dále se věnuje spolehlivé komunikaci a protokolu komunikace. Nakonec obsahuje seznam požadavků. Třetí kapitola (3) se zabývá návrhem knihovny a aplikace. Začátek kapitoly obsahuje popis architektury, dále kapitola obsahuje celkový návrh knihovny a aplikace. Čtvrtá kapitola (4) se zabývá realizací, je zmíněn ovladač převodníku a popsáno rozhraní protokolu. Pátá kapitola (5) se zabývá testováním, obsahuje popsané testovací případy a test výkonu.

Kapitola 1

Cíl práce

Cílem této práce je návrh, implementace a testování knihovny, která umožní spolehlivou komunikaci po sériové lince. Protokol, podle kterého bude komunikace probíhat musí umožnit přenášet různě velké datové struktury, musí být spolehlivý a zajistit integritu dat. Sériové linky budou tvořeny převodníkem USB/UART s čipem FT232R. Součástí práce bude i aplikace, která demonstruje fungování knihovny.

Hlavní body práce:

- návrh spolehlivého protokolu komunikace
- návrh knihovny
- implementace knihovny
- testování knihovny
- návrh aplikace
- implementace aplikace

Kapitola 2

Analýza

2.1 Vývoj bezpečného software

V této sekci se zaměříme na metody vývoje bezpečného softwaru.

2.1.1 Best practices pro vývoj bezpečného softwaru[1]

V této části se podíváme na některé metody a postupy, jejichž použití povede ke snížení rizika chyby v kódu.

2.1.1.1 Unit testy

Unit testy přinášejí velké benefit každému projektu. Ideálně jsou spuštěny okamžitě po každém sestavení projektu a umožňují tak rychle objevit velkou část chyb.

Unit testy se dají rozdělit na dvě kategorie – white box a black box. U white box testu je předpoklad, že známe vnitřek testované komponenty. Takové testy umožňují testování jednotlivých větví programu. Black box testy zkoumají pouze vstup a výstup.

2.1.1.2 Refaktorování

Refaktorování kódu je technika pro provádění změn v kódu. Hlavním cílem je kód, který je lépe udržovatelný a efektivnější.

2.1.1.3 Analýza kódu

Analýza kódu je proces, při kterém hledáme části, které jsou proti best practices, nebo by případně mohly být problémové. Analýza je typicky prováděna automatickým nástrojem. Typicky analýza kódu zkoumá: pojmenování jednotlivých prvků (třídy, proměnné, funkce, ...) odporující konvenci, nedosaži-

telný kód, duplicitní kód, příliš mnoho větvení ve funkci, nezavolání metody pro uvolnění zdrojů, a další.

2.1.1.4 Analýza pokrytí kódu

Analýza pokrytí kódu zkoumá jak velká část kódu je volána testy.

Existují různé druhy pokrytí:

pokrytí příkazů zkoumá, jestli byl každý příkaz proveden alespoň jednou.

pokrytí rozhodování zkoumá, jestli byl booleovský výraz v každém větvení (if, for, ...) vyhodnocen alespoň jednou $false$ a alespoň jednou $true$.

pokrytí podmínek zkoumá, jestli booleovský výraz v každém větvení (if, for, ...) nabyl všech možných vstupních hodnot.

pokrytí cest zkoumá, jestli byla sledována každá možná cesta od začátku funkce do konce.

2.1.1.5 Využití formálních metod

Formální metody jsou matematické metody, pomocí kterých je možné dokázat některé vlastnosti programu. Například: hodnota proměnné nikdy nedosáhne dané hodnoty, parametr funkce nikdy nebude *null*, nikdy nepřistupujeme na neexistující index pole a další.

2.1.2 Návrhové vzory pro spolehlivé systémy[2]

2.1.2.1 Homogení duplex

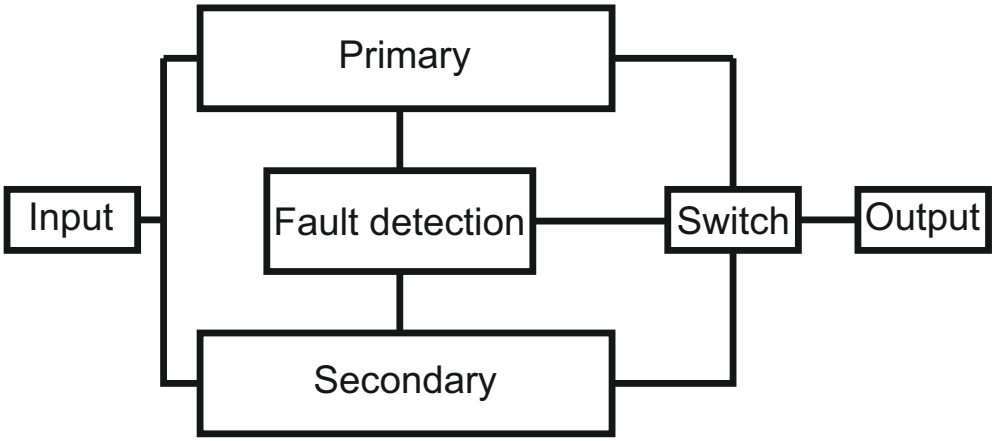
Homogení duplex (obrázek 2.1) je hardwarový návrhový vzor, který řeší situaci, kdy potřebujeme zajistit běh systému i v případě výpadku jedné z komponent.

Vzor situaci řeší přidáním redundance. Kritické komponenty potom obsahuje dvě, primární a sekundární. Navíc obsahuje modul pro detekci chyb. Modul pro detekci chyb kontroluje primární modul a v případě chyby přepne na sekundární.

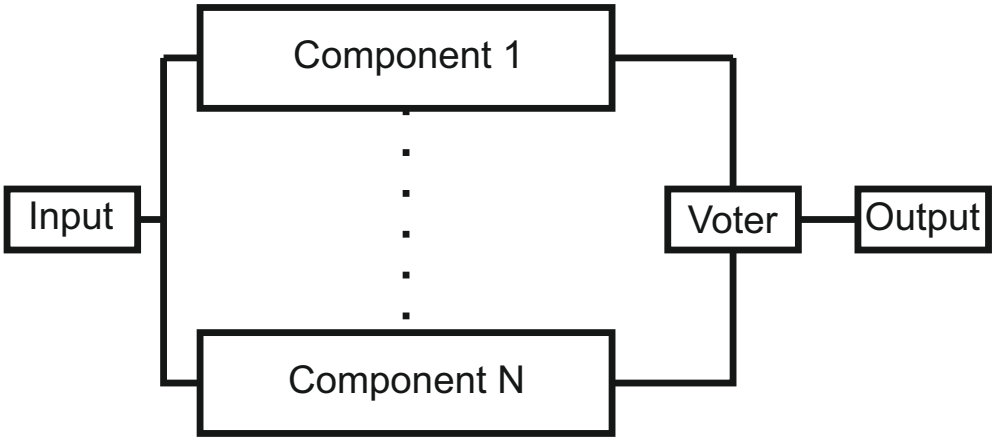
2.1.2.2 M z N

M z N (obrázek 2.2)je hardwarový návrhový vzor, který řeší situaci, kdy potřebujeme zajistit běh systému i v případě výpadku jedné z komponent.

Vzor situaci řeší přidáním redundance. Konkrétně přidáme N identických komponent, které poběží paralelně. Jejich výstup se potom agreguje do další komponenty. Ta podle nějakého hlasovacího algoritmu vybere správný výsledek.



Obrázek 2.1: Schéma vzoru homogení duplex

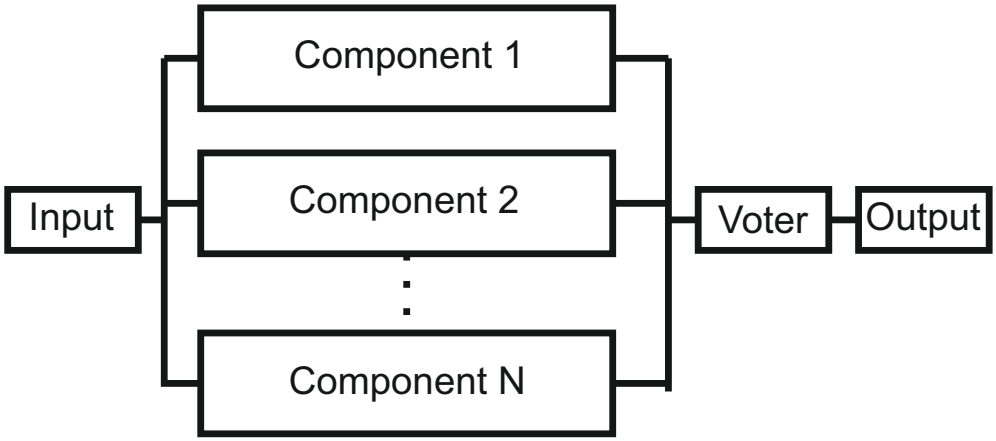


Obrázek 2.2: Schéma vzoru M z N

2.1.2.3 N verzové programování

N verzové programování (obrázek 2.3) je sofwarový návrhový vzor, který řeší situaci, kdy potřebujeme zajistit běh systému i v případě výpadku jedné z komponent.

Vzor situaci řeší přidáním redundance. Konrétně přidáme N komponent, kde každá z komponent je vyrobena nezávislým týmem podle stejné specifikace. Komponenty potom běží paralelně a jejich výstup se agreguje do další komponenty, která podle nějakého hlasovacího algoritmu vybere správný výsledek.

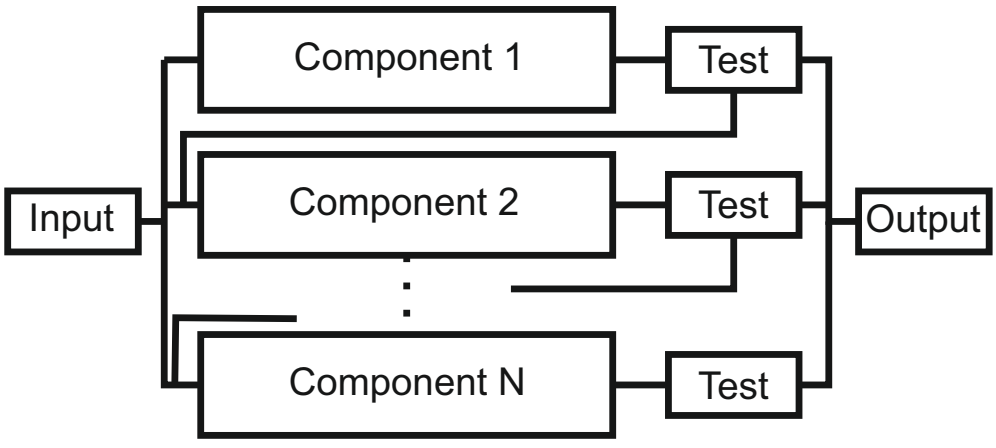


Obrázek 2.3: Schéma vzoru N verzové programování

2.1.2.4 Obnovovací blok

Obnovovací blok (obrázek 2.4) je softwarový návrhový vzor, který řeší situaci, kdy potřebujeme zajistit běh systému i v případě výpadku jedné z komponent.

Vzor situaci řeší podobně jako N verzové programování. Zásadním rozdílem proti předchozí metodě však je, že jednotlivé komponenty neběží paralelně, ale sekvenčně. A výstup komponenty je vždy porovnán proti testu. Pokud test splní je poslán na výstup. Pokud test nesplní, původní vstup dostane další komponenta.

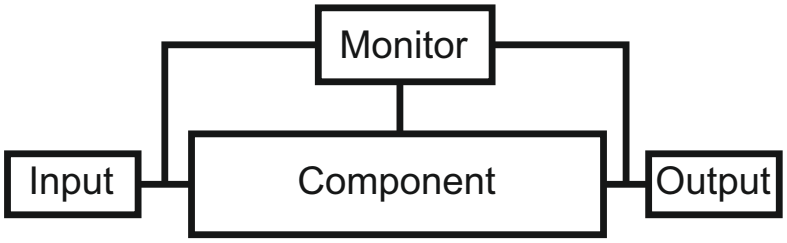


Obrázek 2.4: Schéma vzoru obnovovací blok

2.1.2.5 Externí monitor

Externí monitor (obrázek 2.5) je softwarový návrhový vzor, který řeší situaci, kdy dojde k chybě v jedné z komponent.

Vzor se skládá ze dvou částí – komponenty a monitoru. Komponenta dostává vstup a transformuje ho na výstup. Monitor sleduje vstup, výstup a interní stav komponenty. Účelem monitoru je detekovat chyby a v případě jejich detekce udržet systém v bezpečném stavu.



Obrázek 2.5: Schéma vzoru externí monitor

2.2 Hardware

Sériová linka je realizována pomocí převodníku USB/UART s čipem FT232R od firmy FTDI (Future Technology Devices International). Vybrané vlastnosti[3] čipu FT232R:

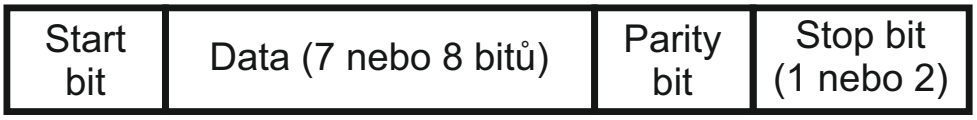
- přenosová rychlost sériové linky od 300Bd až po 3MBd
- každý čip má unikátní id
- výstup hodinového signálu o 48Mhz, 24Mhz, 12Mhz nebo 6Mhz
- paměť EEPROM pro uložení USB VID, PID, sériového čísla a dalších informací

Zařízení dokáže pracovat s 3.3V nebo s 5V. Data jsou posílána v packetech viz. obrázek 2.6. Každý paket začíná 1 bitem, který určuje začátek paketu. Následují data, pro která je vyhrazeno 7 nebo 8 bitů. Paket je ukončen volitelnou paritou a 1 nebo 2 bity, které určují konec packetu.

2.3 Spolehlivá komunikace

Cílem práce je zajistit spolehlivou komunikaci po sériové lince. V této části budeme definovat, co pro nás pojem „spolehlivá komunikace“ znamená.

Protokol, podle kterého se bude komunikace řídit, musí mít jasně definované stavy, ve kterých se může nacházet. Dále musí být jasně určeny podmínky pro přechod z jednoho stavu do druhého.



Obrázek 2.6: UART packet

Protokol musí obsahovat mechanismus, který zajistí, že byla přenášená data doručena (spolehlivost). Dále musí zajistit, že data nebyla nijak změněna (integrita).

Aby byla spolehlivá komunikace možná, musí být celá knihovna spolehlivá – tedy psaná s využitím metod, které snižují riziko zanesení chyby. Dále musí být dostatečně otestovaná a mít zdokumentované chybové stavy.

2.4 Protokol

V této sekci se zaměříme na komunikační protokol. Konkrétně na mechanismy zajištění spolehlivosti přenosu a integrity dat, na druhy zpráv a stavy protokolu. A nakonec na konkrétní ukázky typických akcí, jako otevření a uzavření linky a přenos dat.

2.4.1 Spolehlivost

Pro zajištění spolehlivosti musí protokol obsahovat mechanismus pro ověření doručení jednotlivých zpráv. To zajistíme tím, že každé zprávě přiřadíme sekvenční číslo. Sekvenční čísla umožňují přijaté zprávy uspořádat do stejného pořadí, v jakém byly odeslány. Zároveň je příjem takové zprávy možné potvrdit zasláním zprávy Ack s daným sekvenčním číslem.

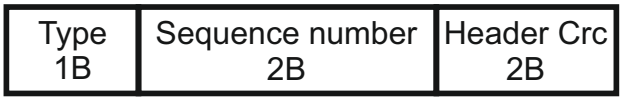
Aby bylo možné sekvenční čísla využít musí se nich obě strany komunikace dohodnout. Proto je nutné nutné, aby protokol spojení vytvořil, po celou dobu komunikace udržoval a nakonec uzavřel.

2.4.2 Integrita

Pro zajištění integrity můžeme použít buď kontrolní součet, nebo hašovací funkci. Obě metody zajistí odolnost proti náhodným změnám, hašovací funkce je navíc odolná i proti cíleným změnám dat (tedy u kontrolního součtu je možné data změnit a kontrolní součet při tom zůstane stejný). Nicméně kontrolní součet bude typicky méně náročnější na výpočet a proto vybereme právě kontrolní součet – konkrétně variantu CRC 16 CCITT.

2.4.3 Struktura zpráv

Zprávy posílané protokolem se dají rozdělit na dva typy – kontrolní zprávy a datové zprávy. Kontrolní zprávy slouží řízení stavu spojení a potvrzování



Obrázek 2.7: Struktura control framu

Obrázek 2.8: Struktura datového framu

přijatých zpráv. Datové zprávy jsou pouze ty zprávy, které nesou zasílaná data.

Hlavička kontrolní zprávy (viz obrázek 2.7) obsahuje 1B pole pro typ zprávy, 2B pole pro sekvenční číslo a 2B pro kontrolní součet.

Hlavička datové zprávy (viz obrázek 2.8) obsahuje 1B pole pro typ zprávy, 2B pole pro sekvenční číslo, 2B pro délku dat, 2B pro kontrolní součet hlavičky, 2B pro kontrolní součet dat a nakonec data.

Sémantika jednotlivých polí:

Typ zprávy určuje typ zprávy, typy zpráv popsány v sekci 2.4.4.

Sekvenční číslo slouží jako identifikátor zprávy, zprávám se přiděluje sekvenčně, tedy první poslaná zpráva bude mít sekvenční číslo x a druhá zpráva $x + 1$ atd. Jedinou výjimkou je zpráva Ack, kde pole obsahuje sekvenční číslo potvrzované zprávy.

Kontrolní součet hlavičky obsahuje kontrolní součet hlavičky zprávy (pro kontrolní zprávy první 3B a pro datovou prvních 5B).

Kontrolní součet dat obsahuje kontrolní součet dat.

Délka dat obsahuje počet datových bajtů následujících za hlavičkou. Vzhledem k tomu, že velikost bufferu pro příjem dat u převodníku je 1024B byla maximální délka určena 1015 (tedy maximální délka celé zprávy je 9B hlavičky a 1015B dat = 1024B), pokud jsou posílaná data delší dojde k jejich rozdělení na více zpráv.

Data obsahuje posílaná data.

2.4.4 Typy zpráv

Open zpráva použitá při požadavku na otevření spojení.

OpenOk zpráva použitá pro potrvzení zprávy Open.

Ack zpráva slouží pro potvrzení přijatých zpráv.

Data zpráva nesoucí data.

Close zpráva použitá pro ukončení spojení.

Rst zpráva použitá pro indikaci chyby a okamžité ukončení spojení.

2.4.5 Stavy

V této sekci se zaměříme na stavy protokolu. Podíváme se na všechny stavy ve kterých se protokol může nacházet. Následně se podíváme na přechody mezi stavy (viz obrázek 2.9) a jak konkrétně vypadá otevření a uzavření spojení a přenos dat.

Closed Výchozí stav, ve kterém neprobíhá žádná komunikace.

Listening Stav po zavolání funkce *Listen*. Protokol přijímá zprávy a pasivně čeká na požadavek na otevření spojení (zprávu Open).

Opening Stav po zavolání funkce *Open*. Protokol se aktivně snaží otevřít spojení (zasílá zprávu Open).

Open Stav, kdy je komunikační linka otevřená a je možné posílat data.

Closing Stav, kdy se komunikační linka zavírá (je poslána zpráva Close).

Faulted Stav, kdy došlo k chybě, není možná další komunikace.

Pro následující scénáře vždy předpokládáme dvě strany komunikace – A a B.

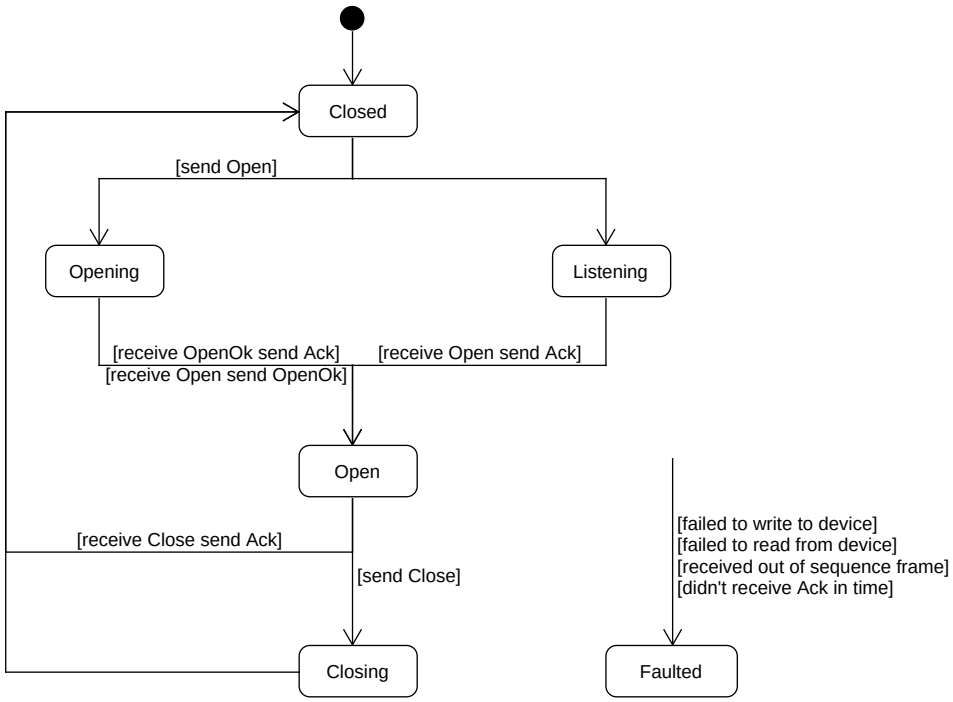
Jako první se podíváme na scénář, kdy jedna strana komunikace pasivně čeká na inicializaci spojení (obrázek 2.10) a strana druhá se snaží spojení otevřít. Obě strany se nachází ve výchozím stavu Closed.

1. Na A je zavolána metoda *Listen*. A přejde do stavu Listening. Zavoláním metody je vybráno náhodné iniciální sekvenční číslo (A_{seq}) a protokol začne přijímat zprávy.

2. Na B je zavolána metoda *Open*. B přejde do stavu Opening. Zavoláním metody je vybráno náhodné iniciální sekvenční číslo (B_{seq}) a protokol začne přijímat zprávy. Zároveň je poslána zpráva Open s B_{seq}.

3. A přijme zprávu a při jejím zpracování se dozví sekvenční číslo B. Jako odpověď je poslána zpráva OpenOk s A_{seq}.

4. B přijme zprávu a jako odpověď pošle zprávu Ack s A_{seq} a přejde do stavu Open.



Obrázek 2.9: Diagram stavů protokolu

5. A přijme zprávu a přejde do stavu Open.

V následujícím scénáři se obě strany snaží spojení aktivně otevřít. Obě strany se nachází ve výchozím stavu Closed.

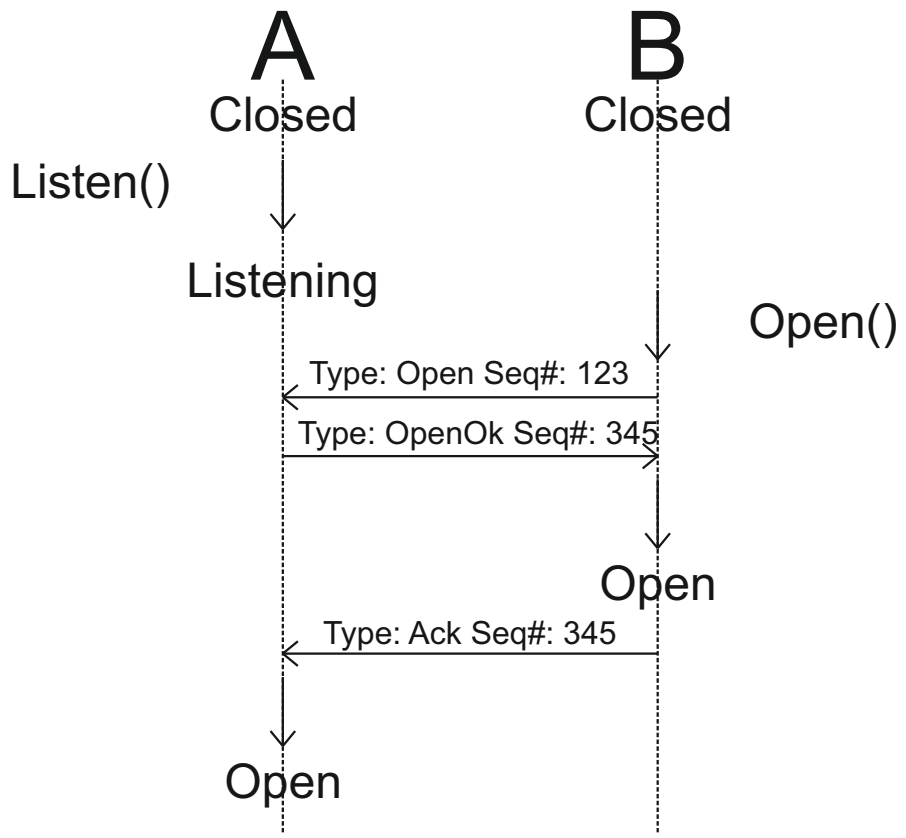
1. Na A je zavolána metoda *Open*. A přejde do stavu Opening. Zavoláním metody je vybráno náhodné iniciální sekvenční číslo (A_{seq}) a protokol začne přijímat zprávy. Zároveň je poslána zpráva Open s A_{seq}.

2. Na B je zavolána metoda *Open*. B přejde do stavu Opening. Zavoláním metody je vybráno náhodné iniciální sekvenční číslo (B_{seq}) a protokol začne přijímat zprávy. Zároveň je poslána zpráva Open s B_{seq}.

3. A i B přijmou zprávu Open a pošlou jako odpověď zprávu OpenOk s příslušným sekvenčním číslem.

4. A i B přijmou zprávu OpekOk, pošlou jako odpověď zprávu Ack a přejdou do stavu Open.

V následujícím scénáři se podíváme jak probíhá posílání dat (obrázek 2.11). Obě strany se nachází ve stavu Open.



Obrázek 2.10: Proces otevření komunikační linky

1. A chce poslat data B. Pokud jsou data delší než 1024B, rozdělí se na části. K jednotlivým částem je přidána hlavička typu Data. Následně je poslána první část.

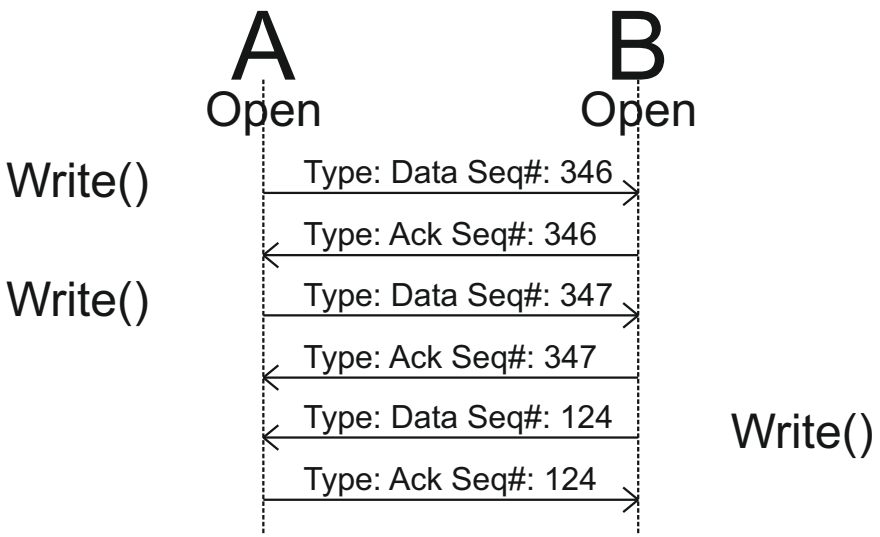
2. B přijme zprávu a pošle jako odpověd zprávu Ack s příslušným sekvenčním číslem. Pokud A nedostane do stanoveného limitu zprávu Ack se sekvenčním číslem zprávy, je zpráva poslána znovu. Zpráva je poslána maximálně třikrát. Potom protokol přejde do stavu Faulted.

3. A přijme Ack a pošle další část.

V následujícím scénáři se podíváme jak probíhá ukončení spojení (obrázek 2.12). Obě strany se nachází ve stavu Open.

1. Na A je zavolána metoda *Close*. A přejde do stavu Closing a pošle zprávu Close.

2. B přijme zprávu Close pošle jako odpověd zprávu Ack s příslušným sekvenčním číslem a přejde do stavu Closed. Pokud A nedostane do stano-



Obrázek 2.11: Proces zasílání dat

veného limitu Ack se sekvenčním číslem zprávy, je zpráva poslána znovu. Zpráva je poslána maximálně třikrát. Potom protokol přejde do stavu Closed a pošle zprávu Rst.

3. A přijme zprávu Ack a přejde do stavu Closed.

2.4.6 Vrstva redundance

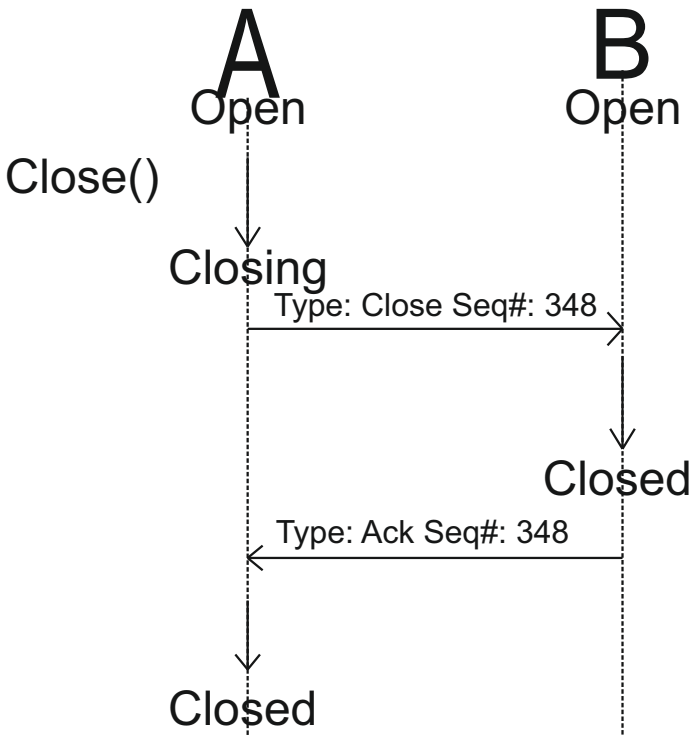
Protokol zajistí bezpečnou komunikaci, stále je tu ale riziko selhání samotného hardwaru. Pro snížení rizika využijeme návrhový vzor M z N. Jedna strana komunikace bude tedy tvořena více převodníky.

Komunikace v rámci jednoho převodníku bude řízená protokolem nezávisle na ostatních. Jednotlivé protokoly potom budou řízené vrstvou redundance. Ta bude zodpovědná za předávání vstupu a výstupu, za detekci chyb u jednotlivých protokolů a jejich následnou obnovu.

Vrstva redundance bude udržovat informace o všech spravovaných protokolech. Z jejího pohledu může být protokol v jednom ze tří stavů – *Ok*, *Recovering* a *Syncing*. Stav *Ok* znamená, že daný protokol pracuje v pořádku, stav *Recovering* znamená, že u daného protokolu došlo k chybě a probíhá proces obnovy a stav *Syncing* znamená, že byl daný protokol úspěšně obnoven, ale je ho ještě potřeba synchronizovat s ostatními.

2.4.6.1 Proces obnovy

Proces obnovy bude mechanismus, který obnoví fungování protokolu u kterého dojde k chybě. Proces obnovy nejdříve zkontroluje jestli je zařízení připojené



Obrázek 2.12: Proces ukončení spojení

Obrázek 2.13: Hlavičky redundance

a následně se bude snažit otevřít spojení.

Po otevření musíme protokol synchronizovat s ostatními. Nutnost synchronizace ilustrujeme na příkladu: mějme dva protokoly, oba přijmou 100B. Následně u prvního protokolu dojde k chybě a než je obnoven druhý protokol přijme dalších 50B. Při dalším čtení z prvního protokolu nebudeme vědět kolikátý bajt jsme přijali.

Pro řešení situace použijeme stejnou metodu jako u protokolu, tedy sekvenční číslo. Sekvenční číslo nám umožní určit pořadí jednotlivých bajtů. Hlavička, kterou přidáme je na obrázku 2.13.

2.4.7 Režijní náklady

Režijní náklady jsou důležitý údaj každého protokolu, který nám říká jak efektivní daný protokol je.

Každá poslaná zpráva má nějakou hlavičku a je na ní poslán acknowledgement. Pro výpočty budeme předpokládat, že jsou zprávy poslány vždy jen jednou. Zprávy poslané v rámci otevření a ukončení spojení budeme ignorovat.

LineProtocol umožňuje maximální velikost zprávy 1015B + 9B hlavičky, *ProtokolBundle* umožňuje maximální velikost zprávy 65535B + 4B hlavičky. Navíc na každou zprávu přichází acknowledgement 5B.

2.4.7.1 Poslání 1B

- 1B se vejde do 1 zprávy *LineProtocolu* a do 1 zprávy *ProtocolBundlu*
- $1 * 9 + 1 * 4 + 1 * 5 = 18$ celková režie je tedy 1800%

2.4.7.2 Poslání 1kB (1000B)

- 1000B se vejde do 1 zprávy *LineProtocolu* a do 1 zprávy *ProtocolBundlu*
- $1 * 9 + 1 * 4 + 1 * 5 = 18$ celková režie je tedy 1.8%

2.4.7.3 Poslání 20kB (20 000B)

- 20 000B se vejde do 20 zpráv *LineProtocolu* a do 1 zprávy *ProtocolBundlu*
- $20 * 9 + 1 * 4 + 20 * 5 = 284$ celková režie je tedy 1.42%

2.4.7.4 Poslání 20mB (1 000 000B)

- 1 000 000B se vejde do 986 zpráv *LineProtocolu* a do 16 zpráv *ProtocolBundlu*
- $986 * 9 + 16 * 4 + 986 * 5 = 13868$ celková režie je tedy 1.39%

2.5 Aplikace

Aplikace bude sloužit k demonstraci knihovny. Bude tedy muset podporovat tyto případy užití:

- vytvoření komunikačních linek.
- odeslání zprávy – obsah zprávy bude buď určený uživatelem nebo vygenerován náhodně. Zprávy bude možné posílat periodicky s nastavitelným intervalem.
- příjem zprávy.
- výběr formátu zprávy - uživatel bude mít možnost vybrat formát obsahu zprávy a to buď textový nebo binární.

2.6 Volba jazyka

Volba jazyka je pro každý projekt velmi důležitá.

Jako první se nabízí nízkoúrovňové jazyky jako C/C++. Jejich hlavní výhodou je efektivita. Program napsaný v C/C++ bude typicky rychlejší než program napsaný například v Javě. Mezi nevýhody zmiňme absenci garbage collectoru, složitost, která vede k zavedení chyb a obtížné ladění chyb. Z důvodu těchto nevýhod C/C++ vyřadíme.

Další se nabízí interpretované jazyky jako jsou Java a C#. Oba jazyky jsou multiplatformní (pro C# je to Mono). Oba jazyky mají také podporu velké knihovny funkcí, která zvládne vše od databází po uživatelské rozhraní. Mezi hlavní výhody těchto jazyků patří spravovaná paměť, jednoduchá práce s vlákny a v neposlední řadě i pokročilé možnosti ladění.

Pro tento projekt zvolíme jazyk C# a to z důvodu výhod interpretovaných jazyků, které byly zmíněny výše a s přihlednutím na osobní zkušenosti s tímto jazykem.

2.7 Požadavky

Tato sekce obsahuje požadavky vyplývající ze zadání a cíle práce.

- komunikační protokol musí umět přenášet různě velké datové struktury
- komunikační protokol musí podporovat binární a textový režim
- komunikační protokol musí zajistit integritu dat
- komunikační protokol musí být spolehlivý

Nefunkční požadavky:

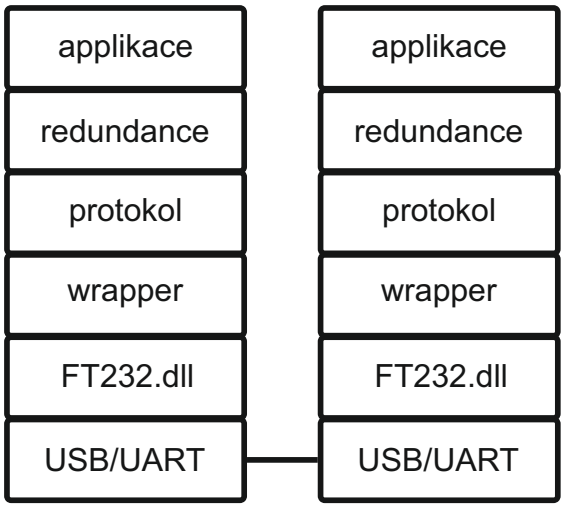
- sériové linky budou tvořeny zařízeními s čipem FT232R

Kapitola 3

Návrh

3.1 Architektura

Vyjdeme z ovladače převodníku, kolem kterého bude bezpečný wrapper (viz sekce 4.1.0.1). Nad wrapperem vybudujeme protokol, který bude zajišťovat komunikaci (viz sekce 2.4) a nad protokolem bude ještě vrstva redundance. Ta zajistí fungování komunikace v případě, že na jedné z komunikačních linek dojde k zásadní chybě, například fyzickému přerušení. Celá idea architektury naznačena na obrázku 3.1.



Obrázek 3.1: Obecná představa architektury

3.2 Finální návrh knihovny

V této části se podíváme na výsledný návrh knihovny (příloha C)a zmíníme důležité třídy.

NativeMethods realizuje ovladač převodníku

DeviceFactory umožňuje detekci a přístup k připojeným zařízením.

DeviceStream umožňuje zápis a čtení do konkrétního zařízení.

LineProtocol realizuje protokol. Postavená nad *DeviceStream*.

ProtocolBundle poskytuje vrstvu redundance pro protokol. Postavená nad *LineProtocol*.

Logger poskytuje metody pro loggování událostí.

3.3 Aplikace

Aplikaci navrhneme tak, aby byla co nejjednodušší. A to jak z pohledu používání, tak implementace. Pro návrh použijeme vzor MVVM.

3.3.1 MVVM

MVVM je návrhový vzor, který umožňuje oddělit logiku uživatelského rozhraní a aplikace od samotného uživatelského rozhraní. Vzor pro to definuje tři druhy tříd.

Model tyto třídy obsahují logiku aplikace.

ViewModel tyto třídy fungují jako prostředník mezi *View* a *Modelem*. Poskytují data a akce pro *View* a aktualizují *Model* v reakci na akce *View*.

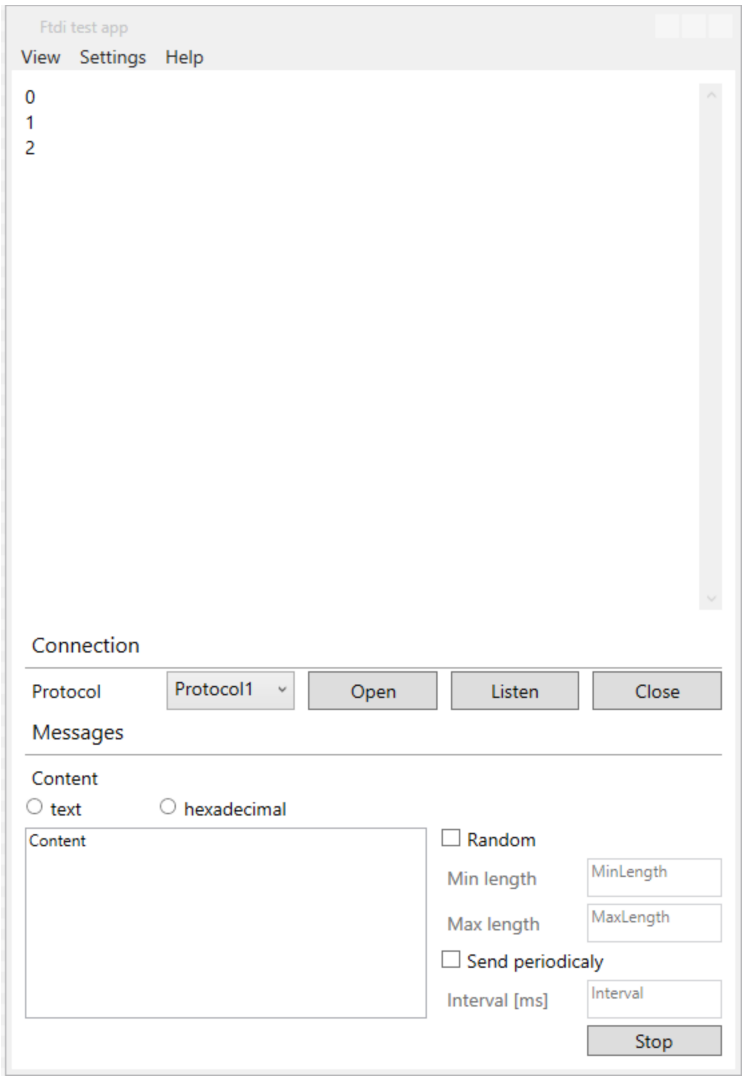
View tyto třídy obsahují prvky uživatelského rozhraní.

3.3.2 Uživatelské rozhraní

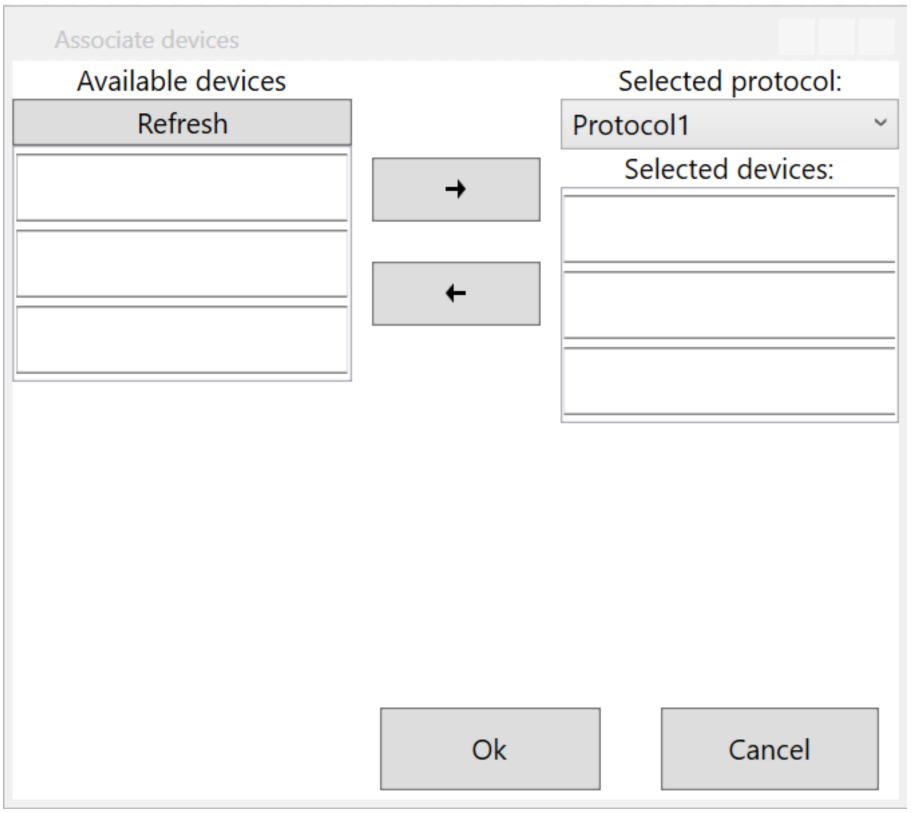
Aplikace bude obsahovat celkem tři okna.

Jedno hlavní okno (obrázek 3.2), které umožní příjem a posílání zpráv a otevírání a ukončování spojení. Druhé okno (obrázek 3.3), které umožní přiřadit zařízení jednotlivým protokolům. A třetí okno (obrázek 3.4), které zobrazí log.

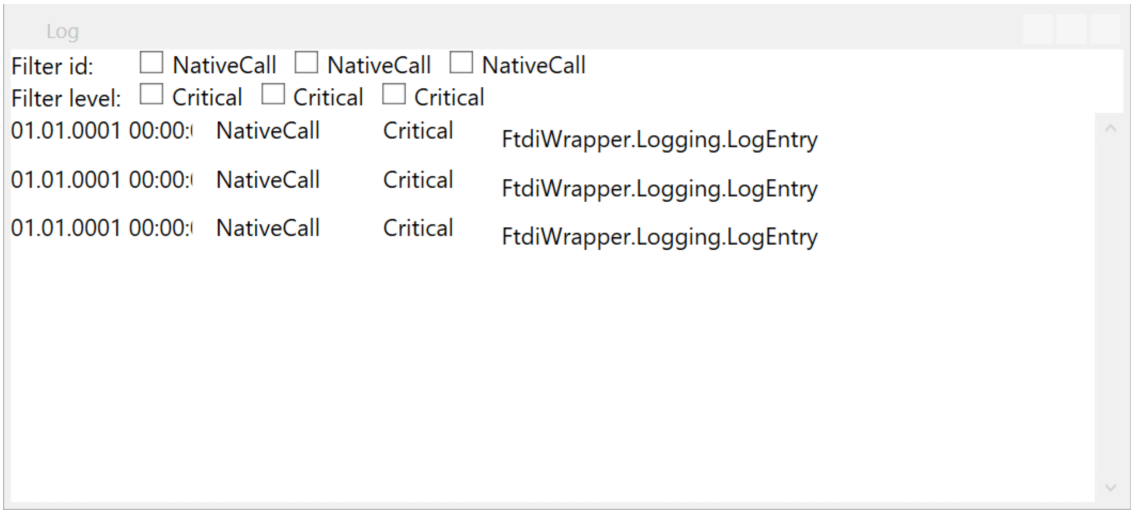
Hlavní prostor okna je obsazen prostorem pro výstup – zde se budou zobrazovat změny stavů protokolu a odeslané a přijaté zprávy. Ve spodní části okna budou ovládací prvky pro otevření a uzavření spojení. Dále bude spodní část okna obsahovat prvky umožňující posílání zpráv.



Obrázek 3.2: Hlavní okno



Obrázek 3.3: Okno pro přiřazení zařízení



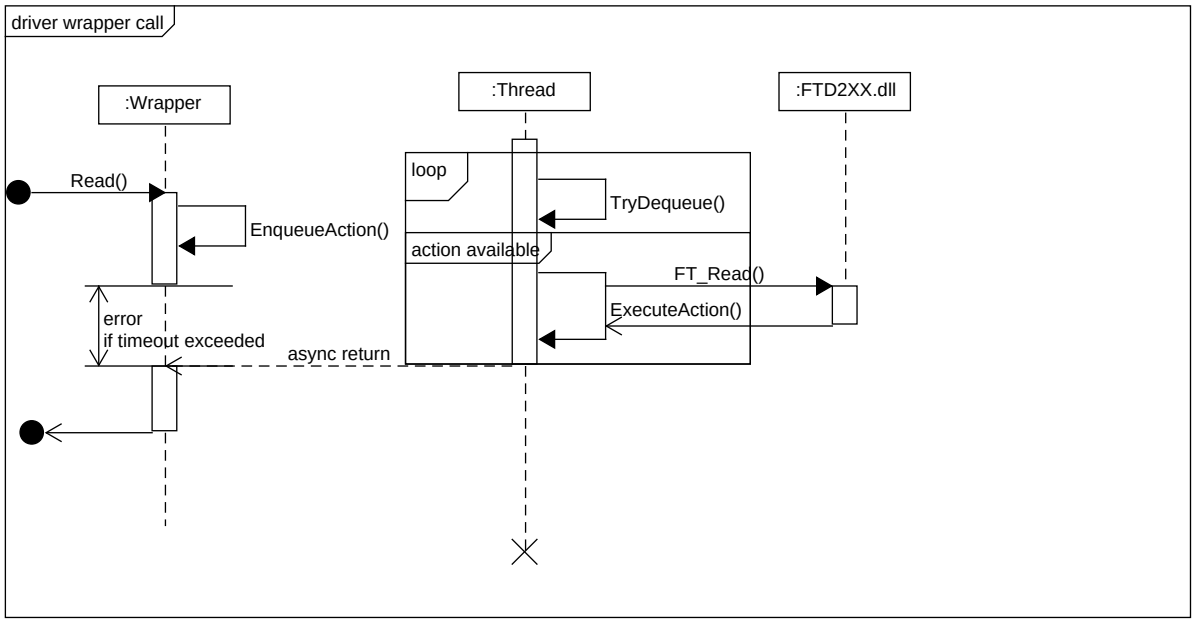
Obrázek 3.4: Okno logu

Kapitola 4

Realizace

4.1 Ovladač převodníku

Volání funkce ovladače bude potřeba izolovat od zbytku knihovny, tím se budeme bránit případnému zamrznutí volání.



Obrázek 4.1: Volání funkce z ovladače převodníku

Wrapper bude obsahovat vlákno, které bude vyřizovat volání funkcí ovladače. Potom při volání nějaké funkce wrapper přidá požadavek do fronty a bude omezenou dobu čekat na signál o dokončení akce. Pokud nedostane signál včas, operace skončí chybou. Vlákno bude pracovat ve smyčce, kde vždy

vybere požadavek z fronty zpracuje ho, a pošle signál o dokončení. Volání potom vrátí. Celý proces jen naznačen na obrázku 4.1.

Případné zamrznutí nebo jiná chyba se potom na fungování wrapperu neprojeví. V případě zamrznutí vyprší čas čekání na signál, a volání skončí chybou, a zamrzlé vláknu je zrušeno.

Pro přístup k zařízení slouží knihovna FTD2XX.dll. Knihovna obsahuje funkce potřebné pro otevření linky, čtení, zápis, nastavení parametrů přenosu a další.

4.1.0.1 Ovladač převodníku

FTD2XX.dll je knihovna napsaná v jazyku C. Zmíníme zde funkce[4], které využije naše knihovna.

Každá funkce z FTD2XX.dll vrací hodnotu FT_STATUS, která určuje zda funkce skončila úspěšně, případně obsahuje kód chyby.

- `FT_STATUS FT_CreateDeviceInfoList (LPDWORD lpdwNumDevs)`

  Tato funkce vytvoří seznam připojených zařízení a vrací jeho délku.

- `FT_STATUS FT_GetDeviceInfoList ( FT_DEVICE_LIST_INFO_NODE *pDest, LPDWORD lpdwNumDevs)`

  Tato funkce zkopíruje seznam připojených zařízení do parametru *pDest*.

- `FT_STATUS FT_Open (int iDevice, FT_HANDLE *ftHandle )`

  Tato funkce otevře zařízení a vrací handle pro další přístup k zařízení.

- `FT_STATUS FT_Close (FT_HANDLE ftHandle)`

  Tato funkce uzavře otevřené zařízení.

- `FT_STATUS FT_Read (FT_HANDLE ftHandle, LPVOID lpBuffer, DWORD dwBytesToRead, LPDWORD lpdwBytesReturned)`

  Tato funkce přečte data ze zařízení.

- `FT_STATUS FT_Write (FT_HANDLE ftHandle, LPVOID lpBuffer, DWORD dwBytesToWrite, LPDWORD lpdwBytesWritten)`

  Tato funkce zapíše data do zařízení.

- FT_STATUS FT_SetBaudRate (FT_HANDLE ftHandle, DWORD dwBaudRate)

  Tato funkce nastaví přenosovou rychlost zařízení.

- FT_STATUS FT_SetDataCharacteristics (FT_HANDLE ftHandle, UCHAR uWordLength, UCHAR uStopBits, UCHAR uParity)

  Tato funkce nastaví parametry přenosu (délku přenášeného slova, počet stop bitů a paritu)

- FT_STATUS FT_GetQueueStatus (FT_HANDLE ftHandle, LPDWORD lpdwAmountInRxQueue)

  Tato funkce vrací počet bajtů připravených ke čtení.

- FT_STATUS FT_SetEventNotification (FT_HANDLE ftHandle, DWORD dwEventMask, PVOID pvArg)

  Tato funkce umožňuje vytvořit synchronizační objekt, který umožní vláknu blokovat, dokud nenastane daná událost (například nová data).

- FT_STATUS FT_Purge (FT_HANDLE ftHandle, DWORD dwMask)

  Tato funkce vymaže buffery pro příjem a odesílání.

4.2 Rozhraní protokolu

Protokol bude realizován třídou *LineProtocol* s následujícím veřejným rozhraním:

- `void Listen()`

  Volání této funkce způsobí vybrání iniciálního sekvenčního čísla, zároveň dojde k otevření zařízení a spuštění úlohy pro příjem dat (viz funkce Read).

- `bool Open(int maxWaitTime)`

  Volání této funkce způsobí vybrání iniciálního sekvenčního čísla, zároveň dojde k otevření zařízení a spuštění úlohy pro příjem dat (viz funkce Read). Nakonec je poslána zpráva Open a vlákno blokuje (po dobu $maxWaitTime$) na signálu o otevření spojení. Funkce vrací *true* v případě, že je spojení v časovém limitu, daném parametrem $maxWaitTime$, otevřeno.

- `void Close()`

  Volání této funkce způsobí poslání zprávy Close, která uzavře spojení.

- `void Write(byte[] buffer, int offset, int length)`

  Tato funkce vezme data k odeslání (tedy $buffer[offset]$ až $buffer[offset+length]$), pokud jsou data delší, než maximální možná velikost zprávy (1024), tak data rozdělí. K získaným částem je přidána hlavička a data jsou odeslána.

- `int Read(byte[] buffer, int offset, int length)`

  Volání této funkce zapíše do parametru $buffer$ data z fronty přijatých dat. Funkce vrací počet přečtených bajtů.

  Přijímané zprávy je potřeba zpracovávat co nejrychleji. Některé zprávy vyžadují acknowledgement, který musíme poslat v daném časovém limitu. Proto musí být čtení ze zařízení řízeno vlastním vláknem. Toto vlákno bude obsahovat nekonečnou smyčku ve které bude čekat na signál od zařízení. Potom přečte data ze zařízení a následně se z nich pokusí deserializovat zprávu. Pokud je zpráva deserializována úspěšně je následně zpracována podle daných pravidel. Pokud zpráva obsahuje data, jsou data přidána do fronty.

4.3 Použité knihovny

4.3.1 MVVM light

MVVM light[5] je knihovna, která obsahuje funkce pro podporu MVVM vzoru. Mezi hlavní části knihovny patří třída $ViewModelBase$, které obsahuje metody pro notifikaci o změnách pro $View$. Další je třída $Messenger$, která umožňuje komunikaci mezi jednotlivými ViewModely a View bez vytvoření závislostí mezi nimi.

4.3.2 WPF

WPF je součást .NET frameworku a slouží k tvorbě uživatelského rozhraní. Hodí se pro použití MVVM vzoru (vzor byl vytvořen přímo pro použití ve WPF). Velkou výhodou WPF je možnost tvořit uživatelské rozhraní pomocí XAML. XAML je jazyk založený XML a slouží k inicializaci struktur a objektů. Další velká výhoda je „data binding“. Data binding umožňuje jednotlivým prvkům uživatelského rozhraní přiřadit jako zdroj nějaký objekt a stará se o následné aktalizace. Například naše View obsahuje pole textové pole, které s pomocí data bindingu pole spojíme textem, který je v ViewModelu.

Potom pokud uživatel začne text přepisovat, data binding se postará o aktualizaci ViewModelu a naopak, pokud se text ve ViewModelu změní, dojde k aktualizaci textového pole.

4.3.3 FakeItEasy

FakeItEasy[6] je knihovna pro vytváření mock objektů. Ty jsou využité v testování.

Kapitola 5

Testovaní

Testování je důležitá součást každého projektu.

Pro tento projekt byly vytvořeny unit testy pomocí Microsoft Visual Studio Unit Testing Frameworku a knihovny FakeItEasy.

Systémové testování bylo prováděno ručně podle případů užití.

5.1 Testovací případy spolehlivosti

Pro knihovnu byly navrženy testy, které vyzkouší chování knihovny v případě, že dojde k nějaké chybě.

5.1.1 Fyzické přerušení linky

K fyzickému přerušení linky dojde pokud dojde k odpojení nebo poškození jednoho z kabelů (RX, TX, GND) nebo odpojení celého zařízení z USB portu.

5.1.1.1 Výchozí stav

Předpokládáme, že máme *ProtocolBundle*, který obsahuje alespoň dva *LineProtocoly*.

5.1.1.2 Očekávaný průběh

1. Dojde k fyzickému přerušení linky – *LineProtokol* s ním spojený přejde do stavu Faulted.
   a) Pokud byl *LineProtokol* poslední aktivní, test končí chybou – *ProtocolBundle* vyhodí výjimku *IOException* se zprávou „All protocols are down.“.
   b) Pokud dojde k odpojení před nebo v průběhu volání metody *Open*, test končí chybou – *ProtocolBundle* vyhodí výjimku se zprávou „Protocol failed to open.“.
   c) *ProtocolBundle* detekuje Faulted stav protokolu a začne proces obnovy.

2. Dojde k opravě přerušené linky.

    a) Proces obnovy protokolu uspěje.

5.1.2 Chyba při přenosu

Chybou přenosu rozumíme změnu nebo úplnou ztrátu jednoho a více přenášených bitů. Tato situace může nastat například vlivem rušení.

5.1.2.1 Výchozí stav

Předpokládáme, že máme *ProtocolBundle* A a *ProtocolBundle* B.

5.1.2.2 Očekávaný průběh

1. A posílá B zprávu a při přenosu dojde k chybě.

2. B přijme poškozená data a následně se z nich pokusí deserializovat *Frame*.

    a) Pokud došlo ke změně některých bitů, deserializace selže, protože nebude souhlasit crc. B tedy nepošle acknowledgement.

    b) Pokud došlo ke ztrátě některých bitů, deserializace buďto nebude možná (nebude dostatek dat), nebo nebude souhlasit crc. V tom případě jsou data, která byla součástí poškozeného framu zahozena a B nepošle acknowledgement.

3. Na A vyprší časový limit a zpráva je poslána znovu.

    a) Pokud je zpráva doručena bez chyby, B pošle acknowledgement.

    b) Pokud znovu dojde k chybě, situace se opakuje. Zpráva je poslána celkem třikrát. Pokud se ani napotřetí nepodaří zprávu doručit, protokol přejde do stavu Faulted.

5.1.3 Zamrznutí volání funkce z FTD2XX.dll

Zamrznutí volání některé funkce z FTD2XX.dll (například FT_Open, FT_Close, FT_Read, FT_Write, ...) znamená, že volání funkce už nevrátí kontrolu programu.

5.1.3.1 Výchozí stav

Předpokládáme, že máme *ProtocolBundle*.

5.1.3.2 Očekávaný průběh

1. Dojde k zamznutí volání.
2. Dojde k vypršení časového limitu pro výpočet operací z FTD2XX.dll. Vypršení limitu způsobí *TimeoutException*.
   a) Pokud je *TimeoutException* zachycena v metodách pro posílání nebo čtení, tak daný protokol přejde do stavu Faulted. Dále se situace řeší stejně jako v případě fyzického přerušení. Tedy pokud byl protokol poslední aktivní, dojde k *IOException*, nebo začne proces obnovy protokolu.
   b) Pokud je *TimeoutException* zachycena v metodě Open, je zachycena v *ProtocolBundle* a dojde k *IOException*.

5.1.4 Chyby v procesu obnovení

Každý *ProtocolBundle* spouští úlohy pro obnovení protokolů, u kterých dojde k chybě. Situace, kdy by v úloze došlo k chybě, by mohla vést k tomu, že protokol nebude nikdy obnoven.

5.1.4.1 Výchozí stav

Předpokládáme, že máme *ProtocolBundle*.

5.1.4.2 Očekávaný průběh

1. Dojde k chybě v úloze pro obnovení protokolu.
2. Další volání funkce pro zápis nebo funkce pro čtení volá metodu, která řeší spouštění úloh pro obnovení. Ta detekuje, že úloha neběží a spustí ji znovu.

5.1.5 Chyby v úloze pro příjem dat

Každý *LineProtocol* má úlohu, která je zodpovědná pro čtení přijatých dat ze zařízení.

5.1.5.1 Výchozí stav

Předpokládáme, že máme *LineProtocol*.

5.1.5.2 Očekávaný průběh

1. Dojde k chybě úlohy pro čtení dat protokolu.
   a) Chyba je zaregistrována úlohou a protokol přejde do stavu Faulted.

Tabulka 5.1: Výsledky testu

Velikost zprávy [B]	Celkem posláno	Průměrná doba doručení [ms]	Průměrná přenosová rychlost [b/s]
1	100B	15.57	498
100	10kB	31.58	24922
500	50kB	31.45	125156
1000	100kB	37.62	209754
2000	200kB	73.32	216743
4000	400kB	136.3	233918
8000	800kB	266.13	240087
10000	1mB	330.18	241977

b) Pokud chyba není zaregistrována úlohou, tak každé volání funkce Read a Write kontroluje stav úlohy. A v případě zjištění chyby ji restartuje.

5.2 Měření výkonu

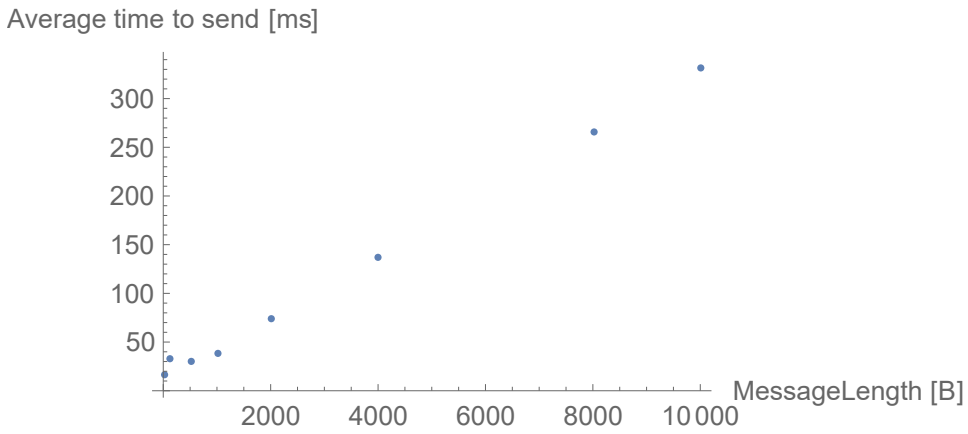
V této části se podíváme jak dlouho protokolu potrvá poslat různě dlouhé zprávy. Budeme sledovat, jak dlouho trvá zprávu doručit (za doručenou zprávu budeme považovat takovou, na kterou přijmeme acknowlegement).

Předpokládáme že, doba doručení je závislá hlavně na přenosové rychlosti, dalším faktorem bude zpracování zprávy a režie protokolu. Zpracování zprávy a přidaná režie bude pro různě dlouhá data v podstatě konstatní. Proto se dá předpokládat lineární závislost mezi dobou doručení a velikostí zprávy.

Test bude prováděn na této konfiguraci: Intel i7-6700HQ @ 2.6GHz, 8 GB RAM, NVIDIA GTX 950M. Použijeme celkem 4 převodníky (2 pro každý *ProtocolBundle*). Vždy pošleme 100 zpráv dané velikosti a budeme měřit čas doručení každé zprávy a celkový čas poslání všech 100 zpráv.

Výsledky jsou v tabulce 5.1. Tabulka obsahuje čtyři sloupce – $Velikostzprávy$ obsahuje velikost posílané zprávy, $Celkemposláno$ obsahuje počet poslaných dat celkem (tedy velikost zprávy * 100), $Průměrnádobadoručení$ obsahuje průměr naměřených hodnot pro doručení a $Průměrnápřenosovárychlost$ obsahuje přenosovou rychlost počítanou jako $\frac{celkemposláno}{dobatestu}$. Graf na obrázku 5.1 zobrazuje závislost doby doručení na velikosti zprávy. Podle grafu je patrné, že je závislost lineární, což potvrzuje náš předpoklad.

Tabulka 5.1 obsahuje výsledky testu



Obrázek 5.1: Graf závislosti doby doručení na velikosti zprávy

Závěr

V rámci této práce jsme se podívali na vývoj spolehlivého software, zmínili jsme důležité metody pro vývoj spolehlivého software a návrhové vzory. Dále jsme navrhli protokol pro komunikaci po sériové lince. Spolehlivost protokolu je zajištěna přidáním sekvenčních čísel a kontrolního součtu. Dále jsme navrhli vrstvu redundance podle návrhového vzoru M z N, která slouží jako zabezpečení proti hardwarovým chybám. Navrhli jsme i testovací aplikaci, které umožňuje navržený protokol vyzkoušet.

Všechny navržené komponenty jsme implementovali a nakonec jsme knihovnu otestovali podle předem navrhnutých testovacích případů. Tímto jsme splnili body zadání.

Literatura

[1] National Aeronautics and Space Administration: *NASA Software Safety Guidebook*. 2004.

[2] Armoush, A.: *Design patterns for safety-critical embedded systems*. Dizertační práce, RWTH Aachen University, 2010.

[3] FT232R Product Page. [cit. 2017-06-21]. Dostupné z: `http://www.ftdichip.com/Products/ICs/FT232R.htm`

[4] Future Technology Devices International Ltd.: *Software Application Development D2XX Programmer's Guide*. [cit. 2017-06-20]. Dostupné z: `http://www.ftdichip.com/Support/Documents/ProgramGuides/D2XX_Programmer's_Guide(FT_000071).pdf`

[5] *MVVM Light*. [cit. 2017-06-26]. Dostupné z: `http://www.mvvmlight.net/`

[6] *FakeItEasy*. [cit. 2017-06-26]. Dostupné z: `https://fakeiteasy.github.io/`

[7] Rausand, M.: *Reliability of Safety-Critical Systems: Theory and Applications*. 2014, ISBN 9781118112724.

[8] Smith, D. J.; Simpson, K. G. L.: *Safety Critical Systems Handbook*. 2011, ISBN 9780080967813.

[9] Hobbs, C.: *Embedded Software Development for Safety-Critical Systems*. Boston, MA, USA: Auerbach Publications, 2015, ISBN 1498726704, 9781498726702.

Příloha A

Seznam použitých zkratek

UART Universal asynchronous receiver/transmitter

Ack Acknowledgement

MVVM Model View ViewModel

XAML Extensible Application Markup Language

SIL Safety integrity level

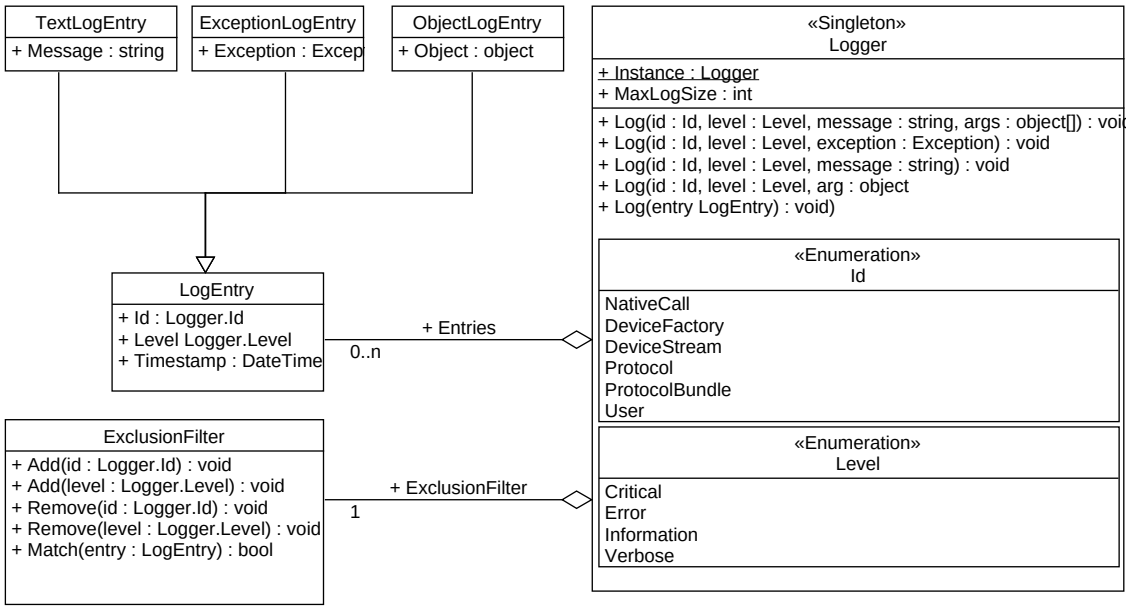
Příloha **B**

Obsah přiloženého CD

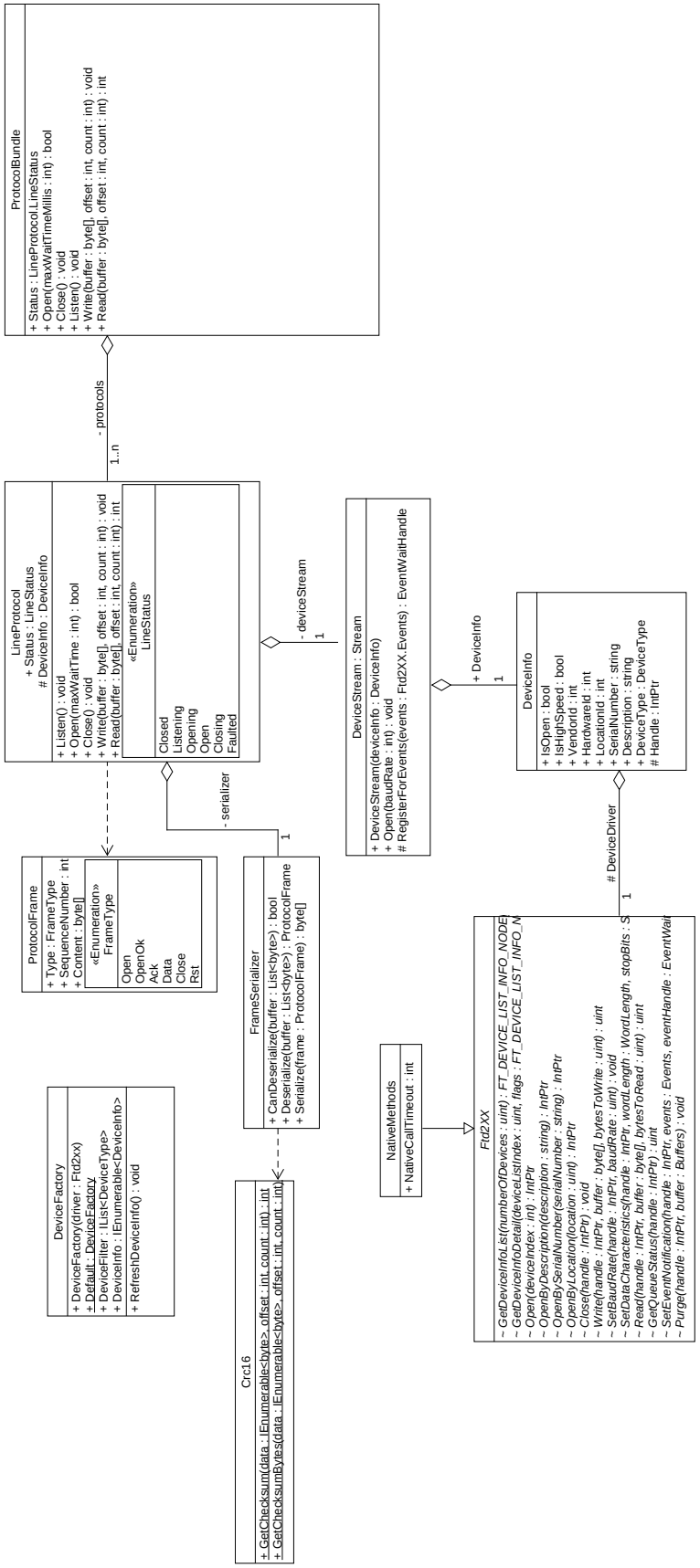
- `readme.txt` ..... stručný popis obsahu CD
- `bin` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ... projekt pro Visual Studio 2017 obsahující zdrojové kódy práce
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `documentation` ..... dokumentace zdrojového kódu

Příloha C

Diagram tříd knihovny



Obrázek C.1: Diagram tříd logu



Obrázek C.2: Diagram tříd knihovny