VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

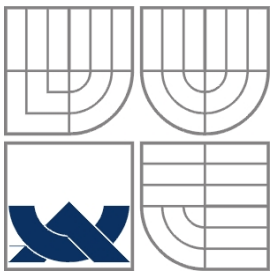
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

VSTUPNÍ ZÁSUVNÝ MODUL DO ANALYZÁTORU PROTOKOLŮ

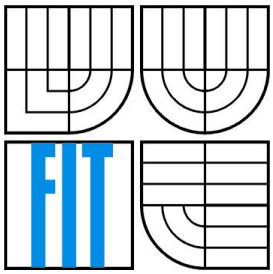
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE TOMÁŠ OCELÍK
AUTHOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

VSTUPNÍ ZÁSUVNÝ MODUL DO ANALYZÁTORU PROTOKOLŮ

IMPORT PLUGIN FOR NETWORK PROTOCOL ANALYZER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ OCELÍK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETER JURNEČKA

BRNO 2010

Abstrakt

Tato práce se zabývá návrhem a implementací zásuvného modulu do síťového analyzátoru protokolů, představuje jeho jednotlivé části a jejich účel. Ukazuje základní princip jeho fungování a je popsáno jeho rozhraní pro vstupní a výstupní zásuvné moduly. Dále je představeno prototypové řešení pro trunkovou síť MPT-1327. Kromě toho práce teoreticky pojednává o základních typech modulací a vyjádření signálu pomocí fázové a kvadraturní složky.

Abstract

This Bachelor's thesis deals with design and implementation of plug-in for Protocol analyzer and shows its main parts and its purposes. Thesis presents the basic principle of operation of Protocol analyzer and describes interface for input and output plug-ins. Then prototype solution for trunk network MPT-1327 is shown. Moreover thesis theoretically deals with basic kinds of modulation and expression of wireless signals by in-phase and quadrature component.

Klíčová slova

Analyzátor protokolů, bezdrátové sítě, demodulace, MPT-1327, I/Q data.

Keywords

Protocol analyzer, wireless networks, demodulation, MPT-1327, I/Q

Citace

Ocelík Tomáš: Vstupní zásuvný modul do analyzátoru protokolů, bakalářská práce, Brno, FIT VUT v Brně, 2010

Vstupní zásuvný modul do analyzátoru protokolů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Petera Jurnečky.
Další informace mi poskytli doc. Dr. Ing. Petr Hanáček, Ing. Jiří Šebesta, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Tomáš Ocelík
19. 5. 2010

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Peteru Jurnečkovi za dobré vedení práce a za připomínky k návrhu softwaru během konzultací.
Dále bych chtěl poděkovat panu docentu Petru Hanáčkovi a panu doktoru Jiřímu Šebestovi z Ústavu radioelektroniky Fakulty elektrotechniky a komunikačních technologií za rady týkající se bezdrátových sítí a frekvenční demodulace.

© Tomáš Ocelík, 2010
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod

Bezdrátové sítě jsou v dnešní době nedílnou součástí informačních a komunikačních technologií. Dávno to již není jen rádio nebo televize, které využívají přenosu informací pomocí elektromagnetického vlnění. Bezdrátové přenosy našly obrovské využití zejména s rozvojem internetu (*Wi-Fi*), mobilních telefonních sítí (*GSM*) [5] nebo komunikačních sítí sloužících specifické skupině lidí nebo organizaci, například policii. Jejich potenciál ještě není zdaleka vyčerpán a do budoucna lze očekávat nasazení do stále více zařízení, především díky snadné použitelnosti a stále rostoucím požadavkům na bezdrátové spojení různých komponent nějakého systému namísto spojení pomocí vodičů.

Tato práce se zabývá návrhem zásuvného modulu do analyzátoru protokolů, jehož autorem je Ing. Peter Jurnečka. Software je navržen pro analyzování datových toků na sítích různých technologií. Například se může jednat o výše zmíněné GSM nebo klasické *ethernetové* sítě a s tím související sadu protokolů *TCP/IP* [4].

Zásuvný modul popisovaný touto prací propojuje zmiňovaný analyzátor protokolů se spektrálním analyzátorem Rohde&Schwarz FSQ8 a tím poskytuje možnost analýzy bezdrátových signálů. Je ale možné jej použít s kterýmkoliv spektrálním analyzátorem podporujícím rozhraní, které tato práce také stručně vysvětluje. Funkčnost celého systému je nakonec ukázána na prototypovém řešení pro *trunkovou* síť *MPT-1327* [2].

Práce nejprve popisuje základní architekturu celého systému a podrobněji uvádí čtenáře do celé problematiky. Je ukázáno rozhraní mezi zásuvným modulem a analyzátorem protokolů a vysvětlena jeho sémantika. Kapitola také popisuje rozhraní mezi zásuvným modulem a spektrálním analyzátorem.

Třetí kapitola se po teoretické stránce zabývá základními modulacemi, vyjádřením signálu pomocí fázové a kvadraturní složky a vztahu tohoto vyjádření s polární reprezentací pomocí amplitudy a fáze. Dále zjednodušeně popisuje princip fungování I/Q modulátorů a demodulátorů.

Čtvrtá kapitola se zabývá filosofií, ze které vycházel návrh zásuvného modulu a dopodrobna tento návrh popisuje. Kapitola detailně prezentuje jeho implementaci a způsob činnosti.

Pátá kapitola potom stručně pojednává o prototypovém řešení pro *trunkovou* síť *MPT-1327.* Nejprve jsou představeny obecné principy fungování trunkových sítí a poté je popsán protokol MPT-1327, který technologie prototypového řešení používá ke komunikaci. Na základě popisu protokolu je pak stručně popsána implementace prototypu pro potřeby této práce.

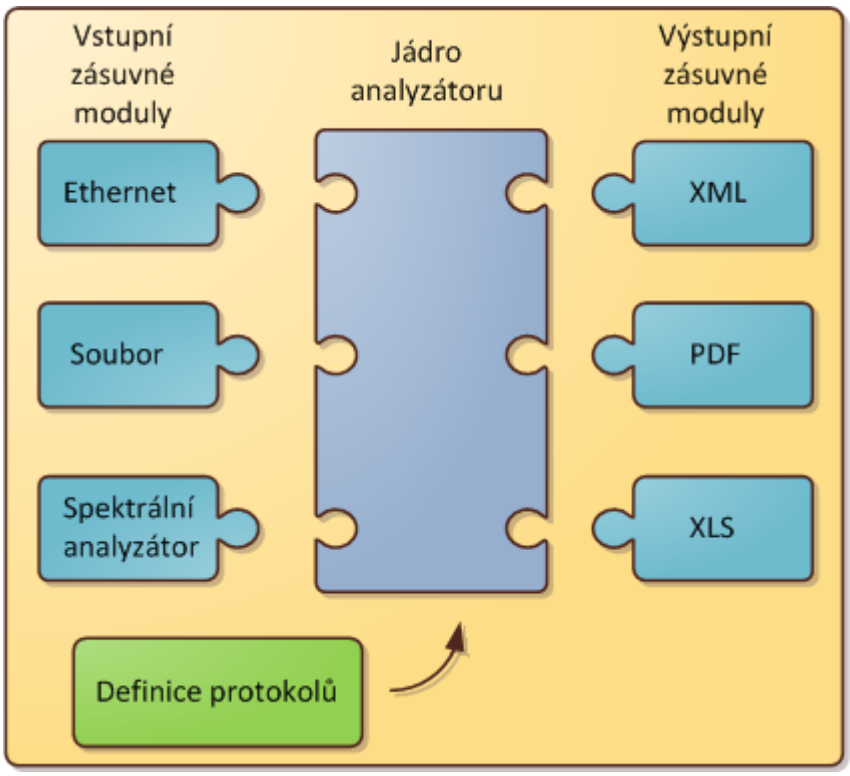
Poslední kapitola shrnuje výsledky práce a nastiňuje možnosti dalšího vývoje.

2 Architektura analyzátoru protokolů

Analyzátor protokolů byl koncipován jako modulární systém, jehož funkcionalita je z větší části implementována prostřednictvím připojovaných *dynamických knihoven* (*DLL*). Je napsán pomocí programovacího jazyka *C#* a technologie *Microsoft .NET*. Díky modularitě analyzátoru lze dosáhnout jeho vysoké obecnosti a tím umožnit jeho široké nasazení v různých oblastech počítačových sítí.

2.1 Popis architektury analyzátoru protokolů

Základní filosofií návrhu analyzátoru bylo oddělit načítání a vykreslování dat od samotné analýzy. Software se proto skládá ze tří základních částí, viz obrázek 1. Obecně se dá říci, že základ programu tvoří jádro analyzátoru, kterému poskytují data vstupní zásuvné moduly. O vykreslování výsledků analýzy se zase starají výstupní moduly. Tím je zajištěna jistá nezávislost programu jak na požadavcích na prezentaci výsledků, tak na zdroji dat. Podrobné informace o architektuře analyzátoru protokolů lze získat v literatuře [1].



Obrázek 1: Blokové schéma analyzátoru

Jádro analyzátoru se stará o vlastní analýzu dat na základě definovaných a přeložených pravidel. Dále provádí načítání vstupních a výstupních modulů, definici pravidel, jejich kompilaci do interní reprezentace a koordinuje celý proces analýzy. Rovněž poskytuje i grafické uživatelské rozhraní, umožňující uživateli například definovat pravidla pro nový protokol.

Výstupní moduly slouží k reprezentaci výsledků analýzy v požadovaném formátu. O rozhraní pro výstupní moduly stručně pojednává kapitola 2.1.3.

Oproti tomu vstupní moduly slouží k zajištění vstupu analyzovaných dat z libovolného zdroje dle požadavků. Příkladem může být textový soubor, *XML*, síťové rozhraní nebo vstup ze spektrálního analyzátoru, který poskytuje zásuvný modul, jímž se tato práce zabývá. Rozhraní pro vstupní moduly je popsáno v kapitole 2.1.2.

Důležité je zde zmínit, že moduly běží ve stejném vlákně, jako analyzátor protokolů.

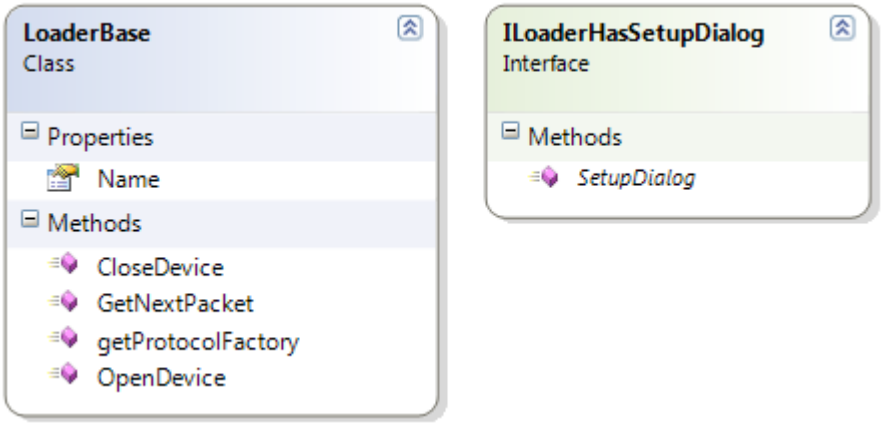
2.1.1 Definice protokolů

Protokoly a jejich pravidla jsou definovány pomocí jazyka založeného na technologii *XML* [11]. Přesná specifikace elementů je popsána v literatuře [1].

Základní filosofií definice pravidel bylo umožnit uživateli snadno definovat novou sadu protokolů pomocí vestavěného editoru, aniž by musel provádět opětovnou kompilaci celého systému. Kompilátor pravidel, který je součástí jádra analyzátoru pak provede převod pravidel z XML do jazyka C# a jeho následnou kompilaci do *Assembly.* Vzniklá dynamická knihovna je poté připojena k jádru analyzátoru. Tato knihovna nese objektový model, vytvořený podle *XML* definice pravidel. Každému protokolu potom odpovídá sada tříd, obstarávajících samotnou analýzu dat.

2.1.2 Rozhraní pro vstupní moduly

Pro vstupní moduly byla definována dvě rozhraní, která zajišťují jak řízení načítání dat, tak jejich základní zpracování. Třída reprezentující vstupní zásuvný modul musí být potomkem třídy `LoaderBase`, tvořící hlavní rozhraní pro zásuvný modul. Z této se pomocí vlastnosti `Name` zjišťuje jméno zásuvného modulu. Podobu rozhraní ukazuje diagram na obrázku 2.



Obrázek 2: Diagram rozhraní pro vstupní moduly

Třída definuje metodu `OpenDevice()`. Metoda se volá na začátku každé analýzy. V jejím těle by měl zásuvný modul provést veškerou svou inicializaci, která je nutná před začátkem každého načítání dat. Takovou inicializací může být například otevření souboru se vstupními daty, či připojení k síťovému rozhraní počítače. Metoda jako svůj výsledek vrací logickou hodnotu, kdy *true* znamená, že inicializace proběhla v pořádku a *false*, že nastala nějaká chyba. V tom případě je analýza předčasně ukončena. Způsob informování uživatele o vzniklé chybě je čistě věcí zásuvného modulu.

Druhou metodou je `CloseDevice()`, která se volá bezprostředně po dokončení analýzy. V rámci ní by měl zásuvný modul uvolnit veškeré zdroje alokované pomocí `OpenDevice()` a uvést se do stavu, kdy je možné znovu volat `OpenDevice()`a provádět další měření.

Další metodou je `getProtocolFactory()`. Tato metoda vrací objekt, který je potomkem třídy `ProtocolFactoryBase`, což je druhé rozhraní pro zásuvný modul a je popsáno níže.

Poslední a nejdůležitější metodou tohoto rozhraní je funkce `GetNextPacket()`, která vrací další paket, či datový rámec protokolu nejnižší vrstvy dané síťové technologie. Data jsou vrácena jako `InformationElementBase`, což je třída reprezentující právě datový rámec protokolu nejnižší vrstvy. Vstupní zásuvný modul může nad datovým rámcem provádět interně nějaké další akce typu počítání kontrolního součtu apod. Rozhraní tyto akce nijak nespecifikuje a záleží jen na tvůrci konkrétního zásuvného modulu, jaký přístup zvolí. Při dosažení konce proudu vstupních dat metoda vrací `null`. Konec proudu dat není nijak explicitně definován, protože je silně závislý na typu vstupu. Obecně se dá říci, že jím může být buď dosažení konce souboru s daty, či například odchycení předem nastaveného množství datových vzorků. V druhém případě by měl zásuvný modul poskytovat nějaký prostředek pro nastavení počtu vzorků, například uživatelský dialog (viz následující odstavec).

Volitelně může potomek třídy `LoaderBase` implementovat i rozhraní `ILoaderHasSetupDialog`, kterým zásuvný modul říká, že obsahuje nějaký nastavovací dialog. Toto rozhraní definuje pouze jednu metodu a to `SetupDialog()`. V rámci ní by implementace měla sama vytvořit daný dialog a po jeho uzavření vrátit řízení zpět do jádra analyzátoru. Nastavení zásuvného modulu jsou čistě jeho interní záležitostí a jádro žádným způsobem tato nastavení neověřuje.

Druhé rozhraní tvoří potomek třídy `ProtocolFactoryBase`. Tato třída řeší identifikaci protokolu nejnižší vrstvy a vytvoření instance příslušného objektového modelu. Ten používá jádro analyzátoru a nad ním pak probíhá vlastní analýza dat. Třída definuje metody s následující sémantikou:

Metoda `ParseProtocol()`slouží ke specifikaci typu protokolu nejnižší úrovně pro datový paket vrácený metodou `GetNextPacket()`. V rámci této metody se poté nad objektem třídy `ProgramInfo`() zavolá metoda `GetObject()`, která dle zadaného názvu objektového modelu vytvoří instanci objektu reprezentujícího protokol nejnižší vrstvy tohoto modelu. V konstruktoru vytvořeného objektu se pak provádí vlastní analýza dat, která je již záležitostí zvoleného modelu.

Přesný popis rozhraní je dostupný v literatuře [1].

2.1.3 Rozhraní pro výstupní moduly

Každý výstupní modul musí implementovat definované rozhraní `IInformationElementRender` [1]. Jeho podoba je ukázána na obrázku 3.

Metoda `SetUp()` slouží k inicializaci zásuvného modulu a v rámci ní by se měl vytvářet kontext výstupního zařízení. Může se jednat například o vytvoření nového okna a získání jeho *Device kontextu* pro kreslení, nebo vytvoření nějakého souboru a jeho otevření pro zápis.

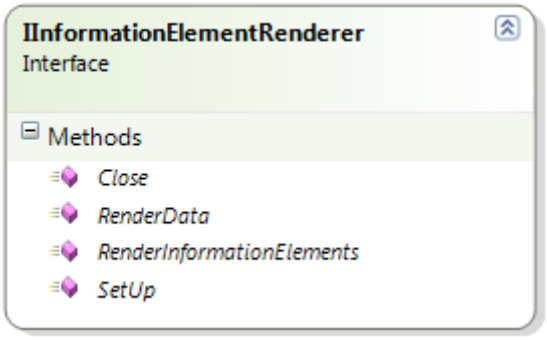
Oproti tomu metoda `Close()` by měla uvolňovat veškeré prostředky alokované v metodě `SetUp()` a uvést výstupní modul do stavu, v jakém se nacházel před jejím zavoláním při inicializaci.

Součástí rozhraní jsou dále metody sloužící pro samotné vykreslování. Metoda `RenderData()` je určena k vykreslení celého datového rámce jako pole bytů. Tato metoda je užitečná v případě, že je požadováno, aby byl k dispozici bytový výpis celého paketu, například v HEX formátu. V případě, že není tento výpis nutný, je možné nechat tělo metody prázdné.

Oproti tomu metoda `RenderInformationElements (InformationElementBase renderedObject, Object args)` slouží k rekurzivnímu hierarchickému vykreslení

analyzovaných protokolů jednotlivých vrstev. Nejprve je pro všechny objekty objektového modelu, vytvořeného na základě definice protokolu z XML, nastaven výstupní vykreslovací modul.

Samotné vykreslování dat probíhá až ve druhé fázi. Jádro analyzátoru zavolá nad každým objektem reprezentujícím protokol nejnižší vrstvy metodu `RenderInformationElements (Object args)`. Každý takto vykreslovaný objekt zavolá funkci `RenderInformationElements (InformationElementBase renderedObject, Object args)` modulu, který má nastavený jako výstupní. Tato funkce se postará o vlastní vykreslení daného objektu. Jestliže vykreslovaný objekt obsahuje nějaké vnořené objekty, metoda opět zavolá jejich funkci `RenderInformationElements (Object args)` a ty opět zavolají vykreslovací funkci svého výstupního modulu (většinou se jedná stále o ten samý modul). Vše se tak stále opakuje, dokud se nedosáhne nejvyšších vrstev protokolu, které již neobsahují žádné zanořené objekty. Tímto se rekurzivně vykreslí celý analyzovaný paket.



Obrázek 3: Diagram rozhraní pro výstupní moduly

2.2 Popis rozhraní spektrálního analyzátoru

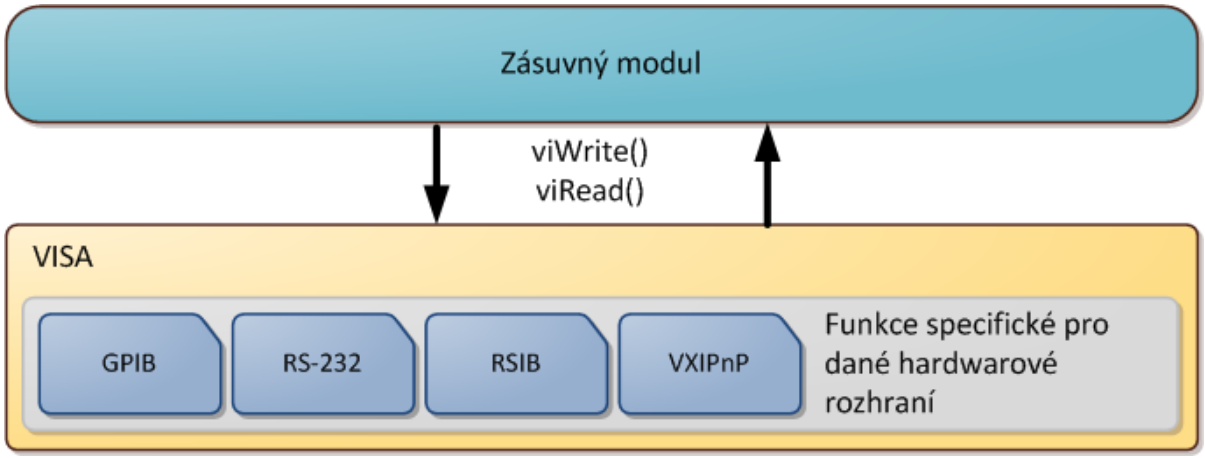
Na fakultě nám byl k dispozici spektrální analyzátor firmy Rohde&Schwarz, konkrétně typ FSQ8. Daný analyzátor dokáže pracovat na frekvenčním rozsahu od 20 Hz do 8Ghz. Přesnou technickou specifikaci poskytuje oficiální dokument od výrobce [9]. Informace o obsluze analyzátoru poskytuje oficiální manuál [3].

Pro účely této práce a pro činnost zásuvného modulu, který tato práce popisuje, je klíčová možnost použít spektrální analyzátor v režimu demodulace zachyceného signálu ve formě I/Q dat. I/Q data popisuje podrobněji kapitola 3.3. Takto naměřená data mohou být ve spektrálním analyzátoru uložena do textového souboru a následně přenesena standardním způsobem na jakýkoliv jiný počítač. Pro I/Q data má analyzátor k dispozici vyrovnávací paměť s kapacitou 16 milionů vzorků jak pro signál *I*, tak pro signál *Q*.

Analyzátor obsahuje síťovou kartu s *ethernetovým* rozhraním a rozhraní *GPIB*. Přes tato dvě rozhraní může být řízen vzdáleně a zachycený signál může přímo posílat do nějakého software na vzdáleném počítači, kde je dále zpracováván.

Po technické stránce je pro komunikaci použito rozhraní *VISA (Virtual Instrument Software Architecture)*, což je standardizované rozhraní poskytující vstupně výstupní funkce pro různá zařízení sloužící k analýze, vyhodnocování apod. Smyslem tohoto rozhraní je poskytovat jednotný přístup k přístrojům různých výrobců a minimalizovat množství úsilí, vydaného na implementaci komunikace s takovým přístrojem. Díky tomu by měl zásuvný modul, popisovaný touto prací, bez problému, či jen s minimálními úpravami kódu komunikovat s jinými přístroji, dodržujícími dané rozhraní.

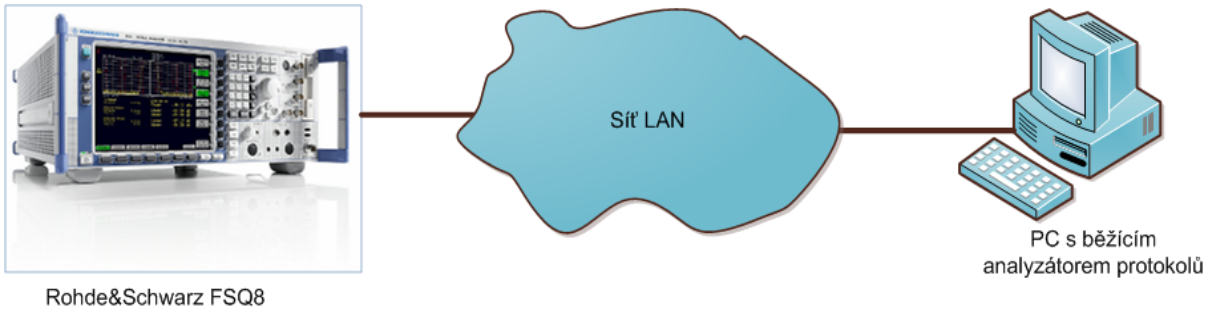
VISA je navrženo tak, aby většina I/O funkcí byla nezávislá na hardwarovém rozhraní použitém pro komunikaci. Klíčové části VISA rozhraní ukazuje obrázek 4.



Obrázek 4: Rozhraní VISA

Obrázek obsahuje některá hardwarová rozhraní, použitelná s VISA. Na obrázku není ukázána sběrnice USB, která je však pro přenos také použitelná. V našem případě bylo počítáno především s rozhraními RSIB a VXIPnP, která umožňují komunikaci prostřednictvím sítě *LAN*. Rozhraní RSIB bylo navrženo firmou Rohde&Schwarz a pracuje nad sadou protokolů TCP/IP. Toto rozhraní je podporováno pouze v implementaci VISA od firmy National Instruments.

Obrázek 5 ukazuje obecný způsob zapojení spektrálního analyzátoru Rohde&Schwarz FSQ8 a zásuvného modulu, popisovaného touto prací. Popis instalace ovladačů a potřebného software popisuje příloha 1.



Obrázek 5: Schéma zapojení spektrálního analyzátoru a PC, na kterém běží analyzátor protokolů

Aby bylo použití rozhraní VISA ještě jednodušší, poskytuje firma Rohde&Schwarz wrapper *RSSpecAn* pro platformu .NET. V našem případě byla použita verze pro jazyk C#, která je tvořena třídou `rsspecan`. Třída obsahuje širokou paletu metod, které jsou již specifické pro analyzátory této firmy, a lze ji použít pro většinu analyzátorů od tohoto výrobce. Teprve v rámci metod třídy `rsspecan` jsou pak volány funkce rozhraní VISA.

V našem případě byly využity především funkce týkající se režimu získávání signálu ve formě I/Q dat. Dokumentace k třídě `rsspecan` je součástí jejího zdrojového kódu. Zde budou popsány ty funkce, které byly v zásuvném modulu použity.

Wrapper a ovladač se inicializuje v rámci těla konstruktoru třídy `rsspecan`. Jako parametr se mu předává *Resource name*, což je řetězec specifikující cílové zařízení a způsob, jakým bude připojeno. Příklady Resource name uvádí tabulka 1. Kromě toho konstruktor obsahuje ještě dva logické parametry. První specifikuje, zda má být provedeno ověření, že dané zařízení je ovladačem podporováno. Druhý parametr říká, zda má být analyzátor při inicializaci resetován.

Další důležitou metodou této třídy je `ConfigureFrequencyCenter()`, pomocí které se nastavuje frekvence nosné vlny snímaného signálu. Funkce bere jako parametr hodnotu frekvence v Hz. Druhým parametrem je číslo vyhodnocovacího okna, které představuje virtuální pracovní plochu. Analyzátor obsahuje dvě a v našem případě je vždy použita ta první.

Resource name	Popis
`RSIB::192.168.1.33::INSTR`	Analyzátor je dostupný na specifikované adrese prostřednictvím rozhraní RSIB
`GPIB::1::0::INSTR`	Přístup prostřednictvím rozhraní GPIB
`ASRL1::INSTR`	Zařízení je připojeno na sériový port 1
`USB::0x1234::125::A22-5::INSTR`	Zařízení je připojeno přes USB. Čísla udávají ID výrobce, kód modelu a sériové číslo.
`TCPIP::192.168.1.33::INSTR`	Analyzátor je dostupný na specifikované adrese prostřednictvím rozhraní VXIPnP

Tabulka 1: Příklady Resource name

Funkce `ConfigureTraceIQDataAcquisition()` určuje, zda se má analyzátor přepnout do režimu získávání signálu ve formě fázové a kvadraturní složky.

Další použitou metodou třídy `rsspecan` je `TraceIQSet()`. Pomocí této metody se nastavuje šířka pásma vstupních analogových filtrů, které jsou předřazeny A/D převodníku v analyzátoru. Funkce rovněž nastavuje vzorkovací frekvenci pro získávání signálu, nastavení triggerů a počet odchycených vzorků. Možnosti nastavení vzorkovací frekvence jsou závislé na konkrétním typu analyzátoru a u některých typů může být napevno stanoveno několik hodnot. Před voláním této funkce by měl být analyzátor nastaven do režimu odchytávání signálu ve formě I/Q dat pomocí funkce `ConfigureTraceIQDataAcquisition()`, viz výše.

Metoda `ConfigureReferenceLevel()` slouží k nastavení referenční hodnoty v dBm pro měření amplitudy signálu. Ve výchozím stavu je nastavena na -20 dBm. Uživatel si však může zadat vlastní hodnotu v dialogu nastavení zásuvného modulu (viz dále).

Poslední důležitou funkcí je `ReadTraceIQData()`. Metoda provádí vlastní zachycení dat v reálném čase. Signály *I* a *Q* jsou vráceny ve formě dvou polí čísel v plovoucí desetinné čárce. Kromě toho je nutné specifikovat *timeout*, po kterém je zachycení ukončeno. Timeout je ve výchozí hodnotě nastaven na 15000 milisekund, ale v případě odchytávání velkého množství vzorků by měl být zvětšen, aby nedošlo k předčasnému ukončení měření.

Všechny výše zmíněné metody třídy `rsspecan` vrací logickou hodnotu podle výsledku operace. Při činnosti ovladače může být vyhozena výjimka `ExternalException` značící chybový stav.

3 Modulace

Modulace je proces, při kterém jedním signálem (*modulační signál*, *baseband*) ovlivňujeme jiný signál (*modulovaný signál*, *nosná vlna*). Obecně můžou být oba signály různého typu (například optický a elektrický). Výsledný signál má potom vlastnosti modulovaného signálu. V našem případě se vždy bude jednat o signály elektrické, které jsou modulovány na elektromagnetické vlnění.
Bezdrátový přenos se dá obecně rozdělit do tří kroků [6]:

1) Generuje se nosná vlna o určité frekvenci.
2) Tato vlna je modulována pomocí přenášené informace. Přičemž platí, že každou spolehlivě detekovatelnou změnu v charakteristice signálu lze využít pro přenos informace.
3) Na straně přijímače jsou změny charakteristiky detekovány a je z nich rekonstruována původní informace.

Protože se jedná o vlnění, existují tři signálové charakteristiky, které lze měnit a využívat k přenosu informací [6]:

- Amplituda
- Fáze
- Frekvence

Od těchto tří charakteristik jsou odvozeny tři základní typy analogových modulací, které se používají pro přenos informací – *amplitudová* (*AM*), *frekvenční* (*FM*) a *fázová* (*PM*). Pomocí modulace se posouvá frekvence *baseband* signálu na frekvenci nosné vlny. Přesněji o modulacích pojednává [6], [7] nebo [13].

Pro přenos digitálního signálu se používají buď digitální modulace anebo se dá digitální signál přímo modulovat na nosnou vlnu. Digitální modulace se někdy též nazývá *klíčování* (*keying*). Existují tři základní varianty tvořící obdobu analogových modulací – *ASK* (*Amplitude Shift Keying*), *FSK* (*Frequency Shift Keying*) a *PSK* (*Phase Shift Keying*).

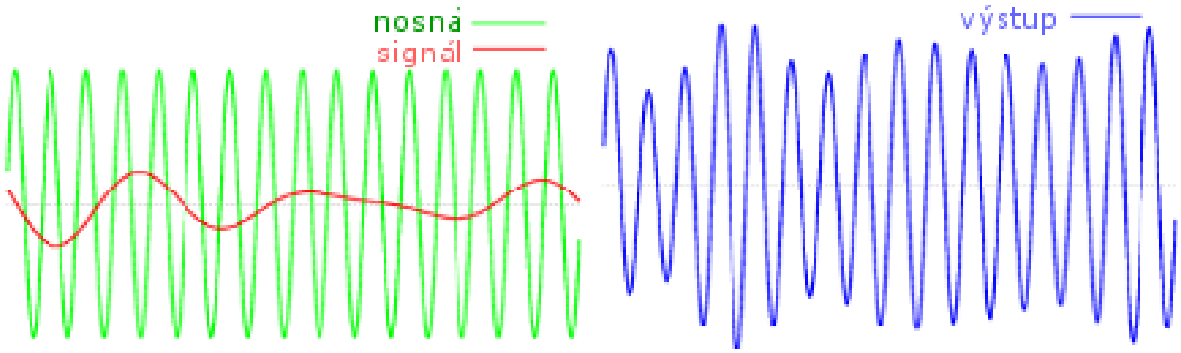
Následující kapitoly stručně pojednávají o základních typech modulací.

3.1 Základní typy analogových modulací

U analogové modulace obecně platí, že nosnou vlnou je signál s harmonickým průběhem v čase a modulačním signálem je nějaký analogový signál (například zvukový).

3.1.1 Amplitudová modulace (AM)

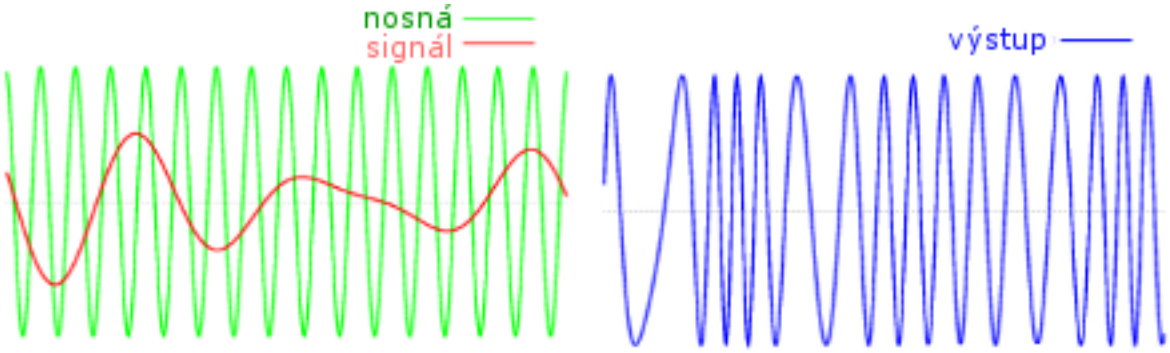
Jedná se historicky o nejstarší typ modulace. Při amplitudové modulaci se mění amplituda nosné vlny na základě přenášeného signálu. Fáze ani frekvence se v čase nemění. Amplitudovou modulaci ukazuje obrázek 6, který byl převzat z [7]. Červenou barvou je zobrazen průběh přenášeného signálu, zelenou barvou je ukázán průběh nosné vlny. Modrou barvou je pak nakreslen průběh výsledného modulovaného signálu. Jak je z obrázku zřejmé, v tomto případě se vyšší hodnota signálu projeví vyšší hodnotou amplitudy.



Obrázek 6: Amplitudová modulace

3.1.2 Frekvenční modulace (FM)

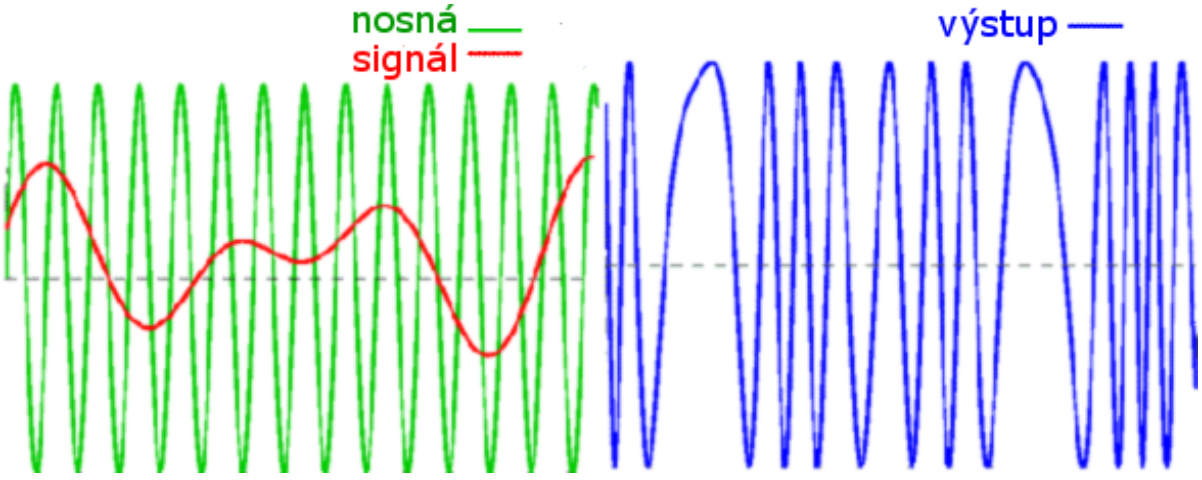
Frekvenční modulace využívá změny frekvence pro přenos informace. V tomto případě zůstává neměnná amplituda signálu. Maximální amplitudě napětí modulačního signálu pak odpovídá maximální změna frekvence nosné vlny. Tato změna se nazývá frekvenční zdvih Δf [7]. FM modulace se často využívá v mobilních systémech.



Obrázek 7: Frekvenční modulace

3.1.3 Fázová modulace (PM)

U fázové modulace je informace přenášena pomocí změny fáze nosného signálu. Je podobná frekvenční modulaci s malým frekvenčním zdvihem.



Obrázek 8: Fázová modulace

3.2 Základní typy digitálních modulací

V případě digitálních modulací je nosným signálem sinusoida a modulačním signálem je digitální signál.

U digitálních modulací je potřeba rozlišovat symbolovou rychlost a bitovou rychlost. Zatímco první zmíněná se vztahuje k množství přenesených symbolů za jednotku času, druhá souvisí s přenášenými daty. Obecně platí, že bitová rychlost je větší nebo rovna symbolové. V nejjednodušších případech (viz níže) jeden symbol odpovídá jednomu bitu. Používají se ale i jiné typy modulací, kdy jeden symbol nese dva bity. Tomu pak odpovídá dvojnásobná bitová rychlost oproti symbolové a nutnost použít čtyři typy symbolů. Existují však i modulace, které používají více symbolů, například 16-QAM, kde jeden symbol nese 4 bity a symbolů je 16.

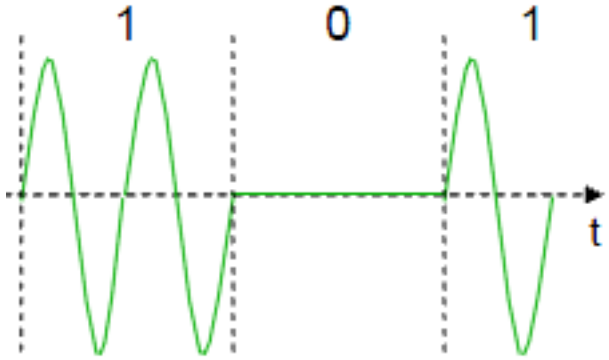
Pro výpočet symbolové rychlosti platí vztah:

$$R_S = \frac{R_B}{N_s}$$

Kde R_B značí bitovou rychlost, N_S je počet bitů na jeden symbol a R_S je symbolová rychlost.

3.2.1 Amplitude shift keying (ASK)

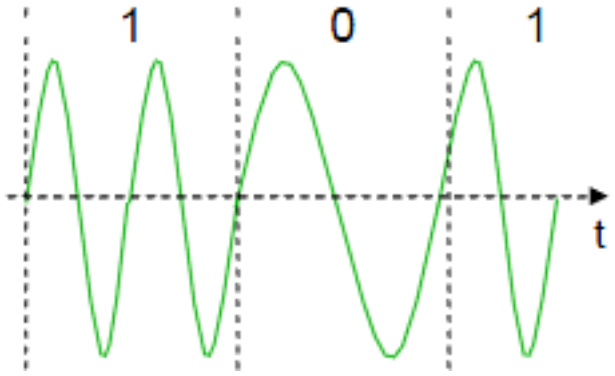
Tato modulace patří k nejjednodušším digitálním modulacím. Ve své základní verzi pracuje tak, že přítomnost nosné vlny indikuje binární jedničku a absence nosné vlny označuje binární nulu. Modulaci ilustruje obrázek 9. ASK je náchylné k interferencím, ale zase má malé nároky na šířku pásma.



Obrázek 9: ASK modulace

3.2.2 Frequency shift keying (FSK)

U FSK jsou v nejjednodušší variantě dvě různé frekvence, z nichž jedna představuje logickou 0 a druhá logickou 1. Obdobně jako u jiných modulací, i zde může jeden symbol nést více jak jeden bit. Tomu pak odpovídá více symbolů a tím pádem více frekvencí. Speciálním případem FSK je *MSK* (*Minimum Shift Keying*), které má spojitou změnu fáze a tím potřebuje užší spektrum než klasické FSK. Obrázek 10 ukazuje nejjednodušší variantu FSK, kterou je BFSK se dvěma frekvencemi.

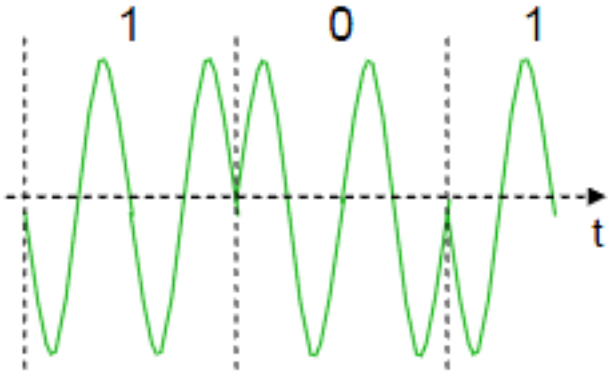


Obrázek 10: FSK modulace

3.2.3 Phase shift keying (PSK)

Fázová modulace patří ke komplexnějším typům modulací a je založena na změně fáze nosné vlny. Obecně existují dva přístupy. Klasické PSK je založeno na zjišťování fáze signálu. Aby mohl přijímač v tomto případě fázi správně detekovat, musí mít k dispozici referenční signál, jehož fázi porovnává s fází přijatého signálu.

Druhou možností je detekce změn fáze. Zde již referenční signál nepotřebujeme a pouze porovnáváme, o kolik se fáze změnila a podle toho detekujeme symboly.



Obrázek 11: PSK modulace

3.3 Fázová a kvadraturní složka signálu

Tato kapitola se stručně zabývá matematickou a grafickou reprezentací harmonického signálu a ukazuje jeho vyjádření jak pomocí polárních, tak kartézských souřadnic – pomocí *fázové* a *kvadraturní* složky signálu (*I/Q*).

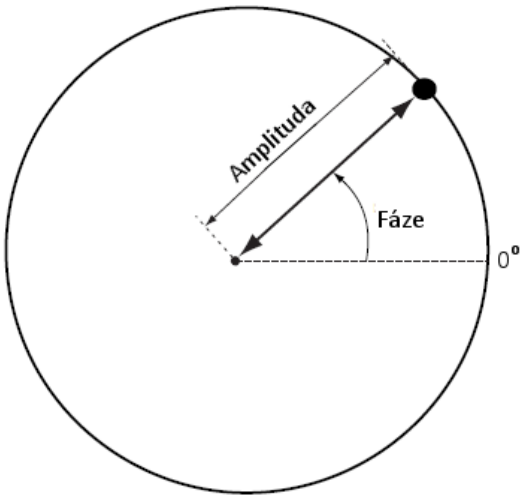
Pro harmonický signál platí rovnice (1), kde *X* představuje výsledný signál, *A* je amplituda signálu, f_C je frekvence, *t* je čas a *Φ* je počáteční fáze.

$$X(t) = A\cos(2\pi f_c t + \phi) \qquad (1)$$

3.3.1 Reprezentace pomocí polárních souřadnic

Pro reprezentaci signálu lze použít *komplexní rovinu* a *polární souřadnice* – *polární diagram*, jako to ukazuje obrázek 12. Na obrázku vzdálenost bodu od počátku reprezentuje amplitudu signálu

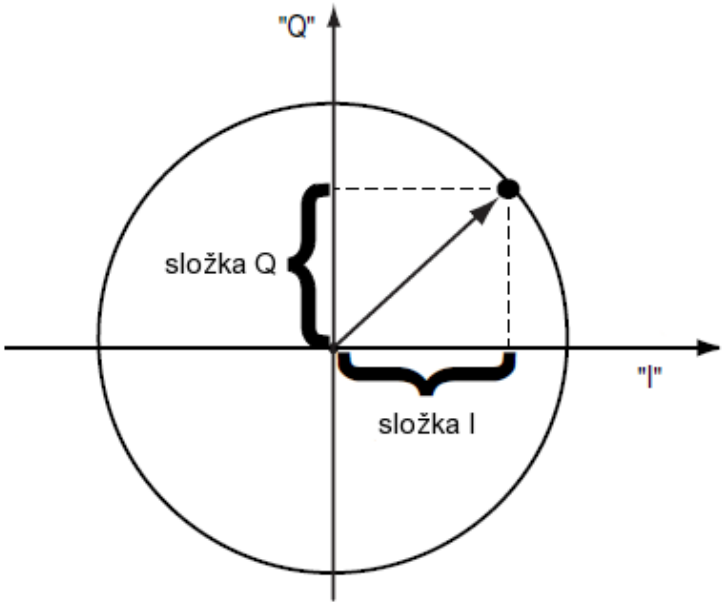
a úhel od kladné části osy x reprezentuje fázi. Fáze bývá většinou relativní vzhledem k referenčnímu signálu a amplituda může být buď absolutní, nebo relativní.



Obrázek 12: Ukázka vyjádření signálu pomocí polárních souřadnic

3.3.2 Reprezentace pomocí kartézských souřadnic

Jinou možností je vyjádřit signál pomocí kartézských souřadnic. Vodorovná osa se pak jmenuje *fázová složka* signálu a značí se *I* (z anglického *in-phase*). Svislá osa se jmenuje *kvadraturní složka* signálu a značí se *Q* (z anglického *quadrature*). V této práci se pak pro body signálu reprezentovaného pomocí *I* a *Q* používá název *I/Q data*. Obrázek 13 ukazuje reprezentaci signálu pomocí fázové a kvadraturní složky.



Obrázek 13: Reprezentace signálu pomocí kartézských souřadnic

Z obrázku je zřejmé, že vztah mezi polárními a kartézskými souřadnicemi je dán vzorci (φ je úhel od kladné části osy *I*):

$$I = A\cos(\varphi) \quad \text{a} \quad Q = A\sin(\varphi) \qquad (2)$$

Pokud vyjdeme ze základního vztahu pro harmonický signál (1), aplikujeme na něj vzoreček pro cosinus dvou úhlů (3) a použijeme vzorce z (2), získáme vztah pro reprezentaci signálu pomocí fázové a kvadraturní složky (5).

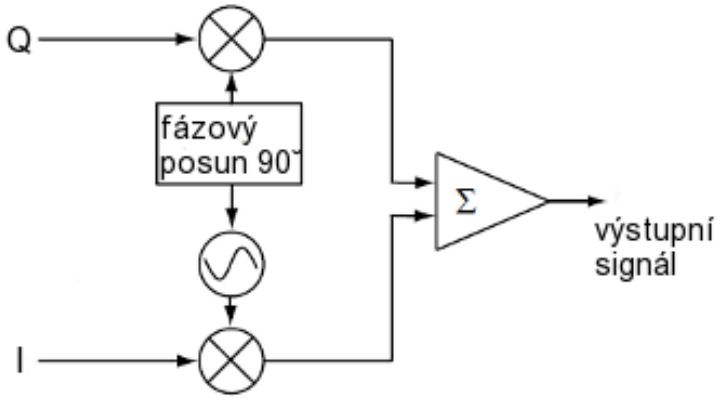
$$\cos(\alpha+\beta)=\cos(\alpha)\cos(\beta)-\sin(\alpha)\sin(\beta) \quad (3)$$

$$A\cos(2\pi f_c t+\varphi)=A\cos(2\pi f_c t)\cos(\varphi)-A\sin(2\pi f_c t)\sin(\varphi) \quad (4)$$

$$A\cos(2\pi f_c t+\varphi)=I\cos(2\pi f_c t)-Q\sin(2\pi f_c t) \quad (5)$$

Jak plyne ze vztahu (5), výsledný signál je složen ze dvou harmonických signálů o frekvencích f_C, které jsou vůči sobě posunuty o 90°. Odtud také plyne název pro fázovou a kvadraturní složku.

Výhodou tohoto přístupu je, že můžeme ovlivňovat amplitudu, fázi i frekvenci výsledného signálu pouze tím, že manipulujeme s amplitudami dvou samostatných signálů, které posléze sčítáme. Tímto způsobem lze snadno vytvořit modulátor, na jehož vstupu jsou dva, obvykle diskrétní signály, které se mixují s nosným signálem, přičemž pro signál Q je nosný signál zpožděn o 90° [16]. Tyto signály se poté sčítají, čímž vznikne výsledný signál, který je již reálný a ve frekvenčním spektru obsahuje dvě pásma, která jsou symetrická podle nuly. Princip modulace ilustruje obrázek 14.



Obrázek 14: Princip I/Q modulátoru

Demodulace pak funguje opačným způsobem. Nejprve je nutné vstupní reálný signál navzorkovat a převést do komplexní podoby. Tímto se spektrum signálu posune doleva a není již souměrné podle nuly. Matematicky to lze vyjádřit vztahem (6).

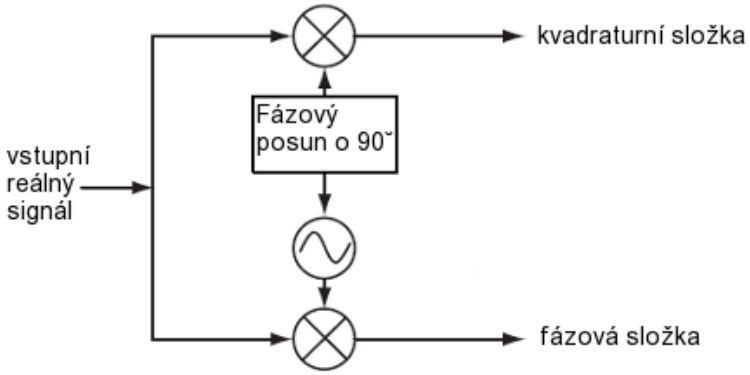
$$X_{I/Q}(t)=X_{RF}(t)e^{-i2\pi ft} \quad (6)$$

X_{RF} je vstupní realný signál. $X_{I/Q}$ je výstupní komplexní signál.

Protože však platí vztah (7), lze převod realizovat tak, že se vstupní signál mixuje se dvěma harmonickými signály o stejné frekvenci, které jsou vůči sobě posunuty o 90°. Tím získáme zpět fázovou a kvadraturní složku, ze kterých pak můžeme odvozovat vlastní data. Odvození dat už je závislé na konkrétním typu modulace. Pro FSK modulaci, se kterou pracuje síťová technologie prototypového řešení, bude odvozování dat probráno v následující kapitole.

$$e^{-i\omega t}=\cos(-\omega t)+i\sin(-\omega t)=\cos(\omega t)-\sin(\omega t) \quad (7)$$

Obrázek 15 ilustruje princip demodulace. Je patrné, že princip je přesně opačný oproti modulaci na obrázku 14.



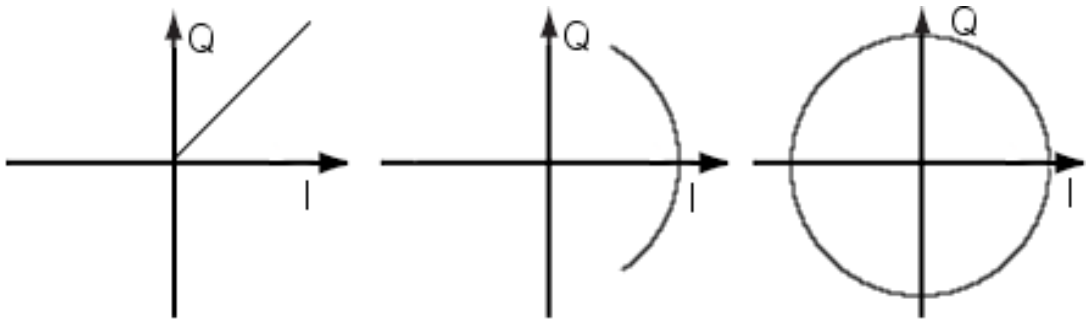
Obrázek 15: Princip I/Q demodulátoru

3.3.3 Základní typy modulací v komplexní rovině

U amplitudové modulace by se za ideálních podmínek měla měnit pouze amplituda signálu. Při zobrazení signálu do komplexní roviny bychom pak měli vidět úsečku jdoucí směrem od počátku do některého kvadrantu, jako to ukazuje obrázek 16a. Při trojrozměrném zobrazení i s časovou osou bychom pak viděli, že průběh signálu leží v části roviny dané touto úsečkou.

Obdobně v případě fázové modulace je amplituda stejná a dochází pouze ke změně fáze. V ideálním stavu by pak body signálu měly vytvořit část kružnice v případě PM, nebo shluky v případě PSK. Tyto shluky se nazývají konstelační body a jsou charakteristické pro jednotlivé typy PSK (platí také pro ASK). PM modulaci ukazuje obrázek 16b.

U změny frekvence je to podobné jako u změny fáze, protože tyto dva parametry spolu úzce souvisí. Změna frekvence se totiž také projeví změnou fáze. Platí, že zvýšení frekvence o 1Hz znamená, že fáze se konstantě zvyšuje rychlostí $2\pi\, rad/s$ vzhledem k fázi signálu s nezměněnou frekvencí. V komplexní rovině tak body v ideálním případě utvoří kružnici, jako to ukazuje obrázek 16c.



a) **b)** **c)**

Obrázek 16: Různé typy digitální modulací v komplexní rovině

3.4 Demodulace FSK

Jak již bylo zmíněno v předchozích kapitolách, u FSK modulace se mění frekvence nosné vlny, přičemž amplituda zůstává stále stejná. Protože změna frekvence znamená také změnu fáze, odvozujeme frekvenci pomocí derivace fáze podle času (8) [13].

$$s = \frac{d}{dt}\left[\arctan\left(\frac{Q}{I}\right)\right] \tag{8}$$

Pro výpočet derivace je vhodné vztah (8) upravit. Protože je derivace v tomto případě rozdíl fází dvou po sobě jdoucích bodů, je možné vztah přepsat do podoby (9), přičemž *oldI* a *oldQ* jsou předchozí vzorky.

$$\frac{d}{dt}\left[\arctan\left(\frac{Q}{I}\right)\right] = \arctan\left(\frac{Q}{I}\right) - \arctan\left(\frac{oldQ}{oldI}\right) \tag{9}$$

Vztah (9) lze dále upravit přidáním funkce tangens v kombinaci s funkcí arkus tangens (10) tak, aby šlo použít vztah (11).

$$\arctan\left(\frac{Q}{I}\right) - \arctan\left(\frac{oldQ}{oldI}\right) = \arctan\left\{\tan\left[\arctan\left(\frac{Q}{I}\right) - \arctan\left(\frac{oldQ}{oldI}\right)\right]\right\} \tag{10}$$

$$\tan(x - y) = \frac{\tan x - \tan y}{1 + \tan x * \tan y} \tag{11}$$

Po aplikaci (11) na část pravé strany (10) a matematických úpravách dostaneme.

$$\tan\left[\arctan\left(\frac{Q}{I}\right) - \arctan\left(\frac{oldQ}{oldI}\right)\right] = \frac{oldI * Q - oldQ * I}{oldI * I + oldQ * Q} \tag{12}$$

Zpětným dosazováním do (10) a (9) získáme konečný vzorec pro výpočet derivace fáze podle času (13).

$$\frac{d}{dt}\left[\arctan\left(\frac{Q}{I}\right)\right] = \arctan\left(\frac{oldI * Q - oldQ * I}{oldI * I + oldQ * Q}\right) \tag{13}$$

Je zřejmé, že výpočet derivace se změnil na výpočet rozdílu úhlů ze dvou po sobě jdoucích bodů. Počítání úhlu je z důvodu efektivity možné realizovat pomocí algoritmu *CORDIC* [12].

4 Popis architektury zásuvného modulu

Zásuvný modul popisovaný touto prací byl koncipován jako software, který propojuje spektrální analyzátor popsaný v kapitole 2.2 a softwarový analyzátor protokolů (kapitola 2.1). V rámci své činnosti modul obdrží diskrétní signál ve formě I/Q dat, ze kterých musí v konečné fázi vytáhnout datový rámec protokolu nejnižší vrstvy a předat jej jádru analyzátoru. Musí tedy mimo jiné provádět i odvozování bitů z přijímaných I/Q dat. Při počátečních návrzích se nabízely dvě základní varianty řešení.

První variantou bylo vytvořit zásuvný modul jako jeden celek, který bude provádět veškerou požadovanou funkcionalitu. Nevýhodou tohoto řešení je jeho nízká obecnost a nutnost modifikovat zdrojové kódy celého modulu v případě požadavků na použití u jiné technologie. Výhodou pak byl jednodušší návrh a implementace.

Druhou možností bylo nahlížet na zásuvný modul pouze jako prostředek propojující analyzátor protokolů se spektrálním analyzátorem, přičemž věci specifické pro různé bezdrátové technologie budou uloženy zvlášť a k zásuvnému modulu se budou připojovat ve formě dynamických knihoven. Tím by byla zajištěna vysoká obecnost celého systému a možnost použít zásuvný modul i na jiné technologie. Nevýhodou by byl složitější návrh a implementace, která by měla vyhovovat různým požadavkům. Nakonec byla zvolena tato varianta řešení. Celý systém je navržen tak, aby bylo možné relativně snadným způsobem dodělat implementaci požadované technologie a tím rozšířit funkcionalitu zásuvného modulu.

Byly identifikovány jednotlivé fáze celého přenosu a na základě nich byl zásuvný modul rozdělen do několika částí, z nichž každá má svůj specifický účel.

Zmíněné části zásuvného modulu se nazývají *provideři*. Provideři se dělí na dva základní typy podle jejich umístění a práce s nimi:

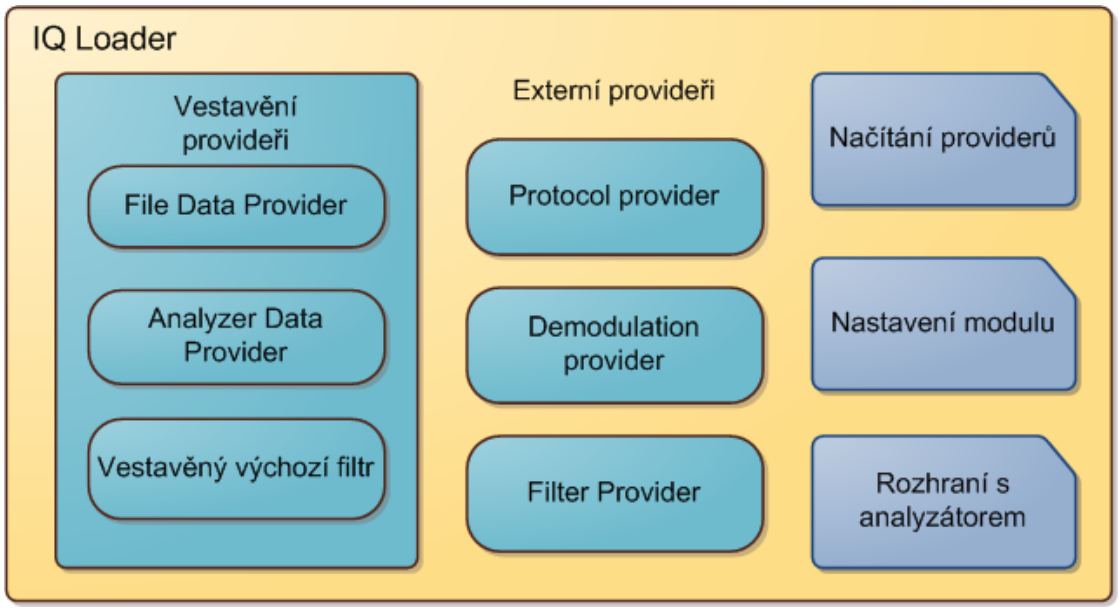
a) *Vestavění* (embedded) – Tito provideři jsou pro svou obecnost vestavěni přímo do zásuvného modulu a poskytují jeho základní funkcionalitu. Jejich přesnější popis bude uveden níže.
b) *Externí* – Externí provideři nesou funkcionalitu, spojenou s konkrétní bezdrátovou technologií. Jsou uloženi v dynamických knihovnách, které se podle potřeby linkují k zásuvnému modulu. Takto implementovaní provideři musí podle svého typu dodržovat rozhraní, která jsou podrobně popsána v následujících kapitolách.

Obrázek 17 ukazuje architekturu zásuvného modulu a jeho klíčové části. Jak je z obrázku patrné, zásuvný modul obsahuje tři vestavěné providery – *File Data Provider* a *Analyzer Data provider* poskytují vstupní data pro analýzu. Jejich přesný popis udává kapitola 4.2. Posledním vestavěným providerem je výchozí filtr, který pouze kopíruje svůj vstup na svůj výstup.

Dále se k zásuvnému modulu načítají tři další typy providerů – *Filter Provider*, *Demodulation Provider* a *Protocol Provider*, kteří se starají (v tomto pořadí) o filtraci vstupních dat, jejich demodulaci a vyhledání datového rámce protokolu nejnižší vrstvy, který je pak odeslán do analyzátoru k vlastní analýze. Fungování těchto providerů a jejich rozhraní se zásuvným modulem je podrobně popsáno v následujících kapitolách.

Kromě toho zásuvný modul obsahuje část, která se stará o načítání, inicializaci, běh a ukončování providerů. Tato část bude dopodrobna popsána v kapitole 4.6. Dále jsou součástí zásuvného modulu funkce zabezpečující rozhraní s analyzátorem protokolů. Zejména se jedná o

třídu `IQLoaderPlugin`, která je základní třídou zásuvného modulu. Tato třída implementuje i volitelné rozhraní `ILoaderHasSetupDialog` a proto obsahuje také prostředky poskytující dialog pro uživatelská nastavení.



Obrázek 17: Blokové schéma zásuvného modulu

Poslední důležitou součástí je třída `PluginSettings`, která nese nastavení celého modulu a objekt této třídy je jako reference předáván i do jednotlivých providerů. Smyslem bylo poskytnout částem modulu všechny dostupné informace o současném nastavení a o vlastnostech zpracovávaného signálu. Třída obsahuje například informace o právě načtených providerech nebo aktuálním nastavení vzorkovací frekvence, od které si může Protocol provider odvodit, kolik bitových vzorků odpovídá jednomu bitu apod. Třída je ukázána na obrázku 23 a zběžně popsána v kapitole 4.6.

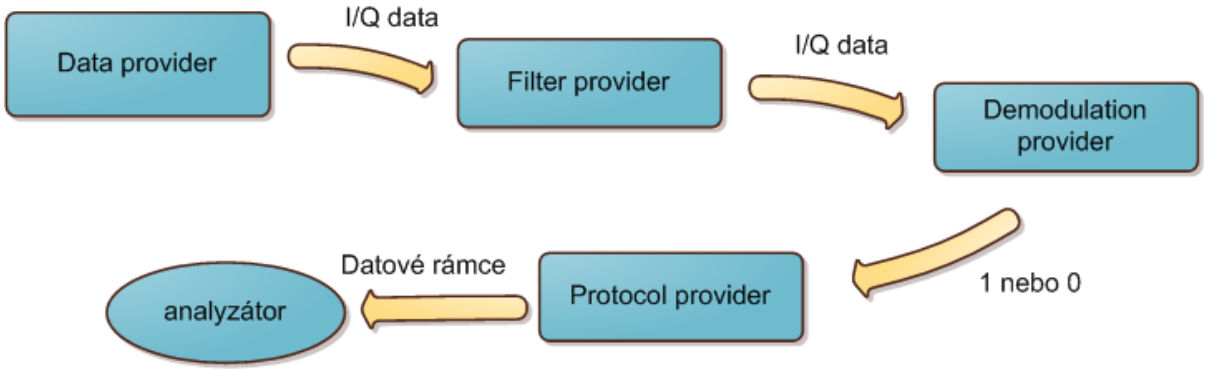
4.1 Pořadí providerů a datové toky

Při návrhu zásuvného modulu se vycházelo z myšlenky, že jednotliví provideři budou poskytovat určité fáze přenosu dat do analyzátoru a tím, že tyto části budou snadno nahraditelné, bude možné modul snadno používat pro různé bezdrátové technologie. Na druhou stranu tyto požadavky zvyšovaly náročnost návrhu jádra samotného modulu tak, aby byl opravdu obecný a poskytoval skutečné možnosti pro různé typy bezdrátových sítí. Z toho důvodu také každý provider dostává referenci na všechna dostupná nastavení ve formě objektu třídy `PluginSettings`. Informace v něm obsažené potom může použít ke své činnosti.

Jednotliví provideři jsou v zásuvném modulu řazeni za sebou, jak ukazuje obrázek 18. Na tomto obrázku jsou u šipek také napsány typy datových toků.

Na začátku celého řetězce je Data provider, který je do modulu vestavěný. Výstupem tohoto provideru jsou dva diskrétní signály (fázová a kvadraturní složka původního signálu). Data dále pokračují do filtru, který může provádět například eliminaci šumu. Výstupem filtru jsou opět I/Q data, která jdou posléze do demodulátoru. Demodulátor ze vstupních I/Q dat odvozuje digitální diskrétní signál tvořený posloupností jedniček a nul. Tyto vzorky však neodpovídají bitům původního signálu, ale jsou to jen vzorky získané převodem I/Q bodů. Jeden bit dat je tedy obecně tvořen více

vzorky. Bitová synchronizace a bitové intervaly jsou rekonstruovány až v rámci Protocol provideru, který pak v proudu bitů hledá počátek datového rámce. Poté co jej nalezne a zachytí celý rámec, jej předá do zásuvného modulu. Ten potom pošle rámec dále do analyzátoru.

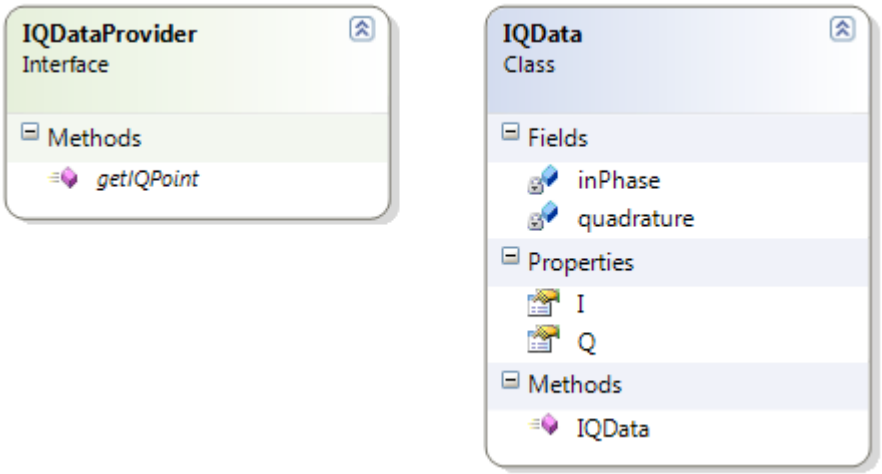


Obrázek 18: Řazení providerů do řetězce a datové toky mezi nimi

Konec proudu dat se oznamuje pomocí výjimky vyhozené v Data provideru. Tato výjimka pak prochází postupně celým řetězcem a je zachycena až na úrovni zásuvného modulu, který provádí její zpracování a oznámí analyzátoru, že bylo dosaženo konce vstupních dat. Podrobněji o životním cyklu zásuvného modulu a o ošetřování chybových stavů pojednává kapitola 4.6.

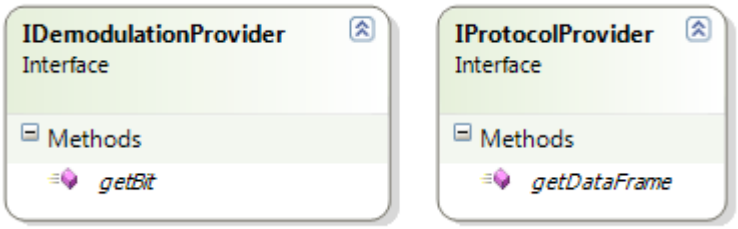
Implementačně jsou provideři navrženi jako potomci tříd definovaných v zásuvném modulu, přičemž každý typ provideru musí být potomkem specifické třídy. Kromě toho jsou definována tři rozhraní podle typu přenášených dat, viz obrázky 19 a 20. Jedná se o rozhraní, pomocí kterých komunikuje každý provider, kromě těch počátečních, se svým předchůdcem v řetězci.

Při návrhu implementace jsme vycházeli z myšlenky, že zásuvný modul potřebuje nahlížet na providery jiným způsobem než provideři na své předchůdce a zároveň bude využívat také jiné metody. Proto zásuvný modul používá při komunikaci s providery třídy, ze kterých jsou odvozeni, a provider komunikuje se svým předchůdcem pomocí rozhraní, které musí implementovat. Výchozími třídami pro popis providerů se zabývají následující kapitoly.



Obrázek 19: Rozhraní IQDataProvider a třída IQData

Rozhraní `IQDataProvider` je určeno k získávání I/Q dat. Je proto použito jak mezi Data providerem a Filter providerem, tak mezi Filter providerem a Demodulation providerem. Rozhraní definuje pouze jednu metodu a to `getIQPoint()`, která vrací jeden I/Q bod. Tyto body jsou v rámci modulu reprezentovány jednoduchou třídou `IQData`.



Obrázek 20: Rozhraní IProtocolProvider a IDemodulationProvider

Rozhraní `IDemodulationProvider` je použito mezi demodulátorem a Protocol providerem. Je určeno k získávání digitálních vzorků demodulovaných ze vstupních dat. Rozhraní definuje metodu `getBit()`, která vrací jeden digitální vzorek (buď 1, nebo 0).

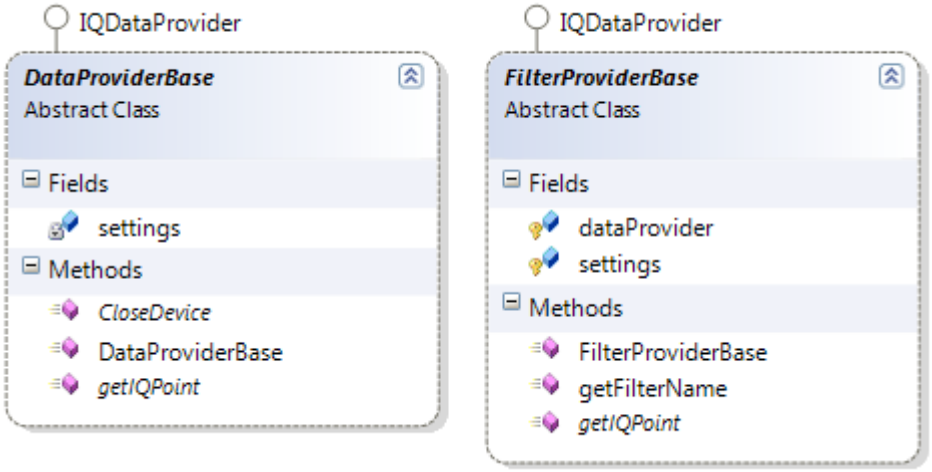
Posledním rozhraním, je rozhraní mezi Protocol providerem a zásuvným modulem, `IProtocolProvider`. Toto rozhraní rovněž obsahuje pouze jednu metodu – `getDataFrame()`, která vrací pole bytů představujících datový rámec protokolu nejnižší vrstvy. Obě rozhraní ukazuje obrázek 20.

4.2 Data provider

Jak bylo zmíněno výše, tito provideři jsou vestavění přímo do zásuvného modulu. Důvodem je jejich vysoká obecnost a také to, že tvoří stěžejní část celého zásuvného modulu a bez nichž by systém neměl smysl.

Data provideři jsou v zásuvném modulu na samém začátku řetězce, přes který data procházejí, a jejich smyslem je získávat tato data z nějakého externího zdroje. V současnosti zásuvný modul obsahuje dva Data providery. Jeden poskytuje možnost čtení I/Q bodů z textového souboru, který vygeneroval spektrální analyzátor. Druhý provádí načítání přímo ze spektrálního analyzátoru prostřednictvím rozhraní VISA (viz kapitola 2.2), typicky skrz počítačovou síť, ke které je analyzátor připojen pomocí ethernetového rozhraní.

Základem pro každý Data provider je abstraktní třída `DataProviderBase`, která implementuje rozhraní `IQDataProvider`, jež bylo popsáno v kapitole 4.1. Podoba třídy je ukázána na obrázku 21.



Obrázek 21: Třídy DataProviderBase a FilterProviderBase

Konstruktor této třídy by měl provést inicializaci vstupního zařízení. Například se může jednat o otevření souboru nebo inicializaci ovladače pro komunikaci se spektrálním analyzátorem.

Oproti tomu metoda `CloseDevice()` by měla vstupní zařízení uzavřít a uvolnit všechny alokované zdroje.

Metoda `getIQPoint()` pak vrací další I/Q bod ze vstupního proudu dat, nebo hází výjimku `EndOfStreamException` v případě dosažení konce dat.

Kromě těchto metod obsahuje třída ještě i referenci na nastavení celého zásuvného modulu. Toto nastavení se předává do konstruktoru jako parametr.

4.2.1 File data provider

File data provider poskytuje možnost načítat navzorkovaný a zaznamenaný signál z textového souboru ve formě I/Q bodů. Tento soubor je možné snadno vytvořit pomocí spektrálního analyzátoru a samotnou analýzu signálu provádět až dodatečně offline. Jméno souboru je převzato z objektu nastavení zásuvného modulu, který byl předán při inicializaci provideru.

Textový soubor v tomto formátu obsahuje na začátku hlavičku s informacemi o měření. Protože jsou však tyto informace nastavovány v uživatelském dialogu, rozhodli jsme se, že bude hlavička souboru při načítání přeskočena. Samotné I/Q body jsou v souboru uloženy každý na vlastním řádku a fázová složka je od té kvadraturní oddělena středníkem. Pokud není dodržena správná syntaxe, je vyhozena výjimka `FileFormatException`, která postupně projde celým řetězcem, na úrovni zásuvného modulu je odchycena a analýza tímto předčasně končí.

4.2.2 Analyzer data provider

Analyzer data provider umožňuje načítat informace přímo z analyzátoru pomocí rozhraní VISA, například prostřednictvím počítačové sítě. Zde již není možné načítat data ze sítě postupně, protože bychom pravděpodobně neobdrželi souvislou posloupnost I/Q bodů, tak jak byl signál vysílán. Je to způsobeno tím, že činnost analyzátoru protokolů si vyžádá nějakou, byť krátkou dobu a některé úseky signálu v čase by tím byly vynechány.

Provider proto provede inicializaci ovladače a načtení dat do lokální cache v okamžiku své inicializace, kdy je celý zásuvný modul ve stádiu zavolání funkce `OpenDevice()`. Při vlastním načítání dat jsou pak vraceny body z naplněné cache. V okamžiku, kdy je cache prázdná, se vyhazuje výjimka `EndOfStreamException`, obdobně jako když se narazí na konec vstupního souboru.

Timeout pro načítání dat je stanoven na pevno na 20000 milisekund, což odpovídá předpokládané maximální době zachytávání dat.

Ostatní parametry pro nastavení analyzátoru jsou převzaty z objektu třídy `PluginSettings`, který byl předán jako parametr konstruktoru při vytváření provideru.

4.3 Filter provider

Smyslem tohoto provideru je poskytovat možnost předřadit demodulaci filtr. Ten by sice bylo možné vytvořit jako součást demodulátoru, ale systém by byl zbytečně složitý a přišli bychom o možnost aplikovat různé filtry na stejný typ demodulace. Jak již bylo zmíněno výše, zásuvný modul přímo obsahuje výchozí filtr, který pouze kopíruje vstup na svůj výstup.

Vstupní filtry se odvozují od třídy `FilterProviderBase` implementující rozhraní `IQDataProvider` (viz obrázek 21). Nad touto třídou je implementován i výchozí filtr, který, je do zásuvného modulu zabudován.

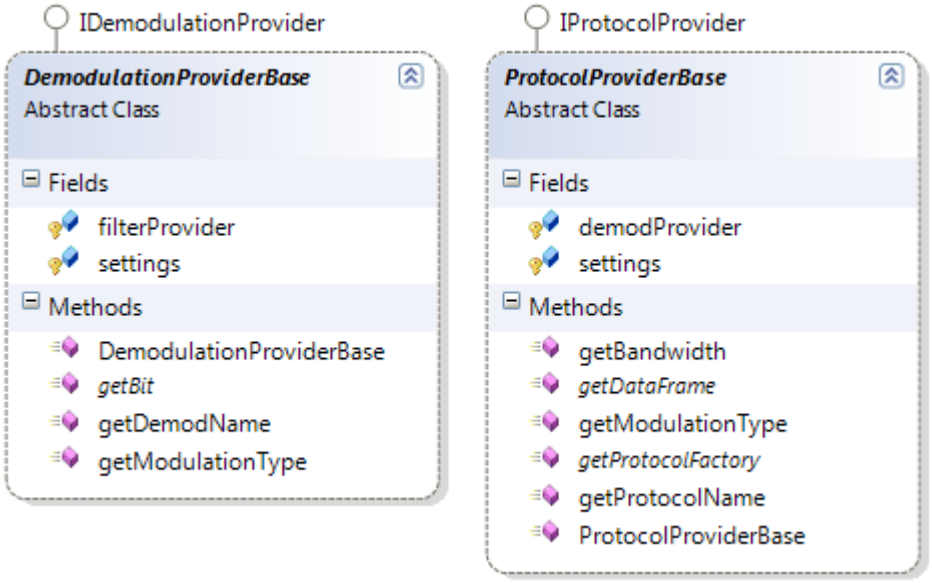
Statická metoda `getFilterName()` vrací název filtru. Ten se následně zobrazí v seznamu dostupných filtrů v nastavení zásuvného modulu. Metoda v této třídě hází výjimku o absenci implementace, a tudíž musí být v implementovaném provideru předefinována.

Metoda `getIQPoint()` vrací další bod ze vstupního proudu dat nebo hází výjimku informující o konci dat.

4.4 Demodulation provider

Demodulátor je v řetězci zařazen hned za filtrem. Jeho úkolem je získávat z dodávaného proudu I/Q dat bitové vzorky, které se pak posílají dále na zpracování do Protocol provideru.

Demodulátory se odvozují od třídy `DemodulationProviderBase`. Tato třída implementuje rozhraní `IDemodulationProvider`, protože demodulátor „sousedí“ s Protocol providerem. Třída je ukázána na obrázku 22.



Obrázek 22: Třídy DemodulationProviderBase a ProtocolProviderBase

Kromě reference na nastavení zásuvného modulu a reference na Filter provider třída obsahuje několik metod. Statická metoda `getDemodName()`vrací jméno demodulátoru. Toto jméno je pak použito v seznamu demodulátorů v dialogu nastavení zásuvného modulu. Třída dále obsahuje ještě statickou metodu `getModulationType()`, která vrací typ modulace. Tento údaj je pak při načítání providerů před samotnou analýzou porovnán s typem modulace, požadovaným Protocol providerem. V případě neshody je vyhozena výjimka a proces zahájení přenosu dat je předčasně ukončen. Obě zmiňované statické metody v případě této abstraktní třídy hází výjimku o absenci implementace. Je tedy nutné je v potomcích předefinovat.

Konstruktor třídy by měl obsahovat inicializaci demodulátoru a provádět ověření správné vzorkovací frekvence a dalších nastavení, která jsou pro daný demodulátor klíčová.

Poslední metodou je `getBit()`, která vrací další bitový vzorek získaný ze vstupních I/Q dat, nebo hází výjimku `EndOfStreamException` v případě konce proudu dat.

4.5 Protocol provider

Poslední v řetězci zpracování je Protocol provider. Od demodulátoru dostává bitové vzorky a jeho smyslem je jak nalézt bitovou synchronizaci, tak načíst jeden rámec protokolu nejnižší vrstvy a předat jej zásuvnému modulu. Kromě toho může nad nalezeným rámcem provádět některé dodatečné operace jako například výpočet kontrolního součtu pro ověření správnosti dat.

Základem pro vytváření Protocol providerů je třída `ProtocolProviderBase`. Třída je ukázána na obrázku 22. Jak je vidět, tak obdobně jako ostatní provideři nese referenci na svého předchůdce v řetězci zpracování a dále nese referenci na nastavení celého zásuvného modulu.

Konstruktor provideru by měl provést všechny potřebné inicializace, včetně případných kontrol na správná nastavení. Typicky je například nutné ověřit, zda je nastavena správná vzorkovací frekvence tak, aby vycházel celý lichý počet bitových vzorků na jeden bit.

Třída dále obsahuje tři statické metody. Metoda `getProtocolName()` vrací název protokolu/provideru a tento je pak použit v dialogu pro nastavení zásuvného modulu. Metoda `getBandwith()` vrací šířku pásma požadovanou protokolem v jednotkách Hz. Tato informace je zobrazena v dialogu nastavení modulu a má čistě informační charakter. Konkrétní šířku pásma si nastaví uživatel sám v případě, že nastaví jako zdroj dat analyzátor. Pokud je jako zdroj dat použit soubor, je nutné nastavit vhodnou šířku pásma přímo na spektrálním analyzátoru.

Poslední statickou metodou je `getModulationType()`, která vrací typ modulace používané daným protokolem. Tato informace je jednak vyplněna v nastavení zásuvného modulu, ale především je porovnávána při načítání providerů s typem modulace zvoleného demodulátoru. Obdobně jako u předchozích providerů, i zde implementace metod házejí výjimku a je nutné je v potomcích předefinovat.

Další metodou je `getProtocolFactory()`. Metoda vrací Protocol factory pro daný protokol (viz kapitola 2.1.2). Vytvořenou továrnu potom vrací zásuvný modul jádru analyzátoru při volání metody `getProtocolFactory()` nad třídou `IQLoaderPlugin`.

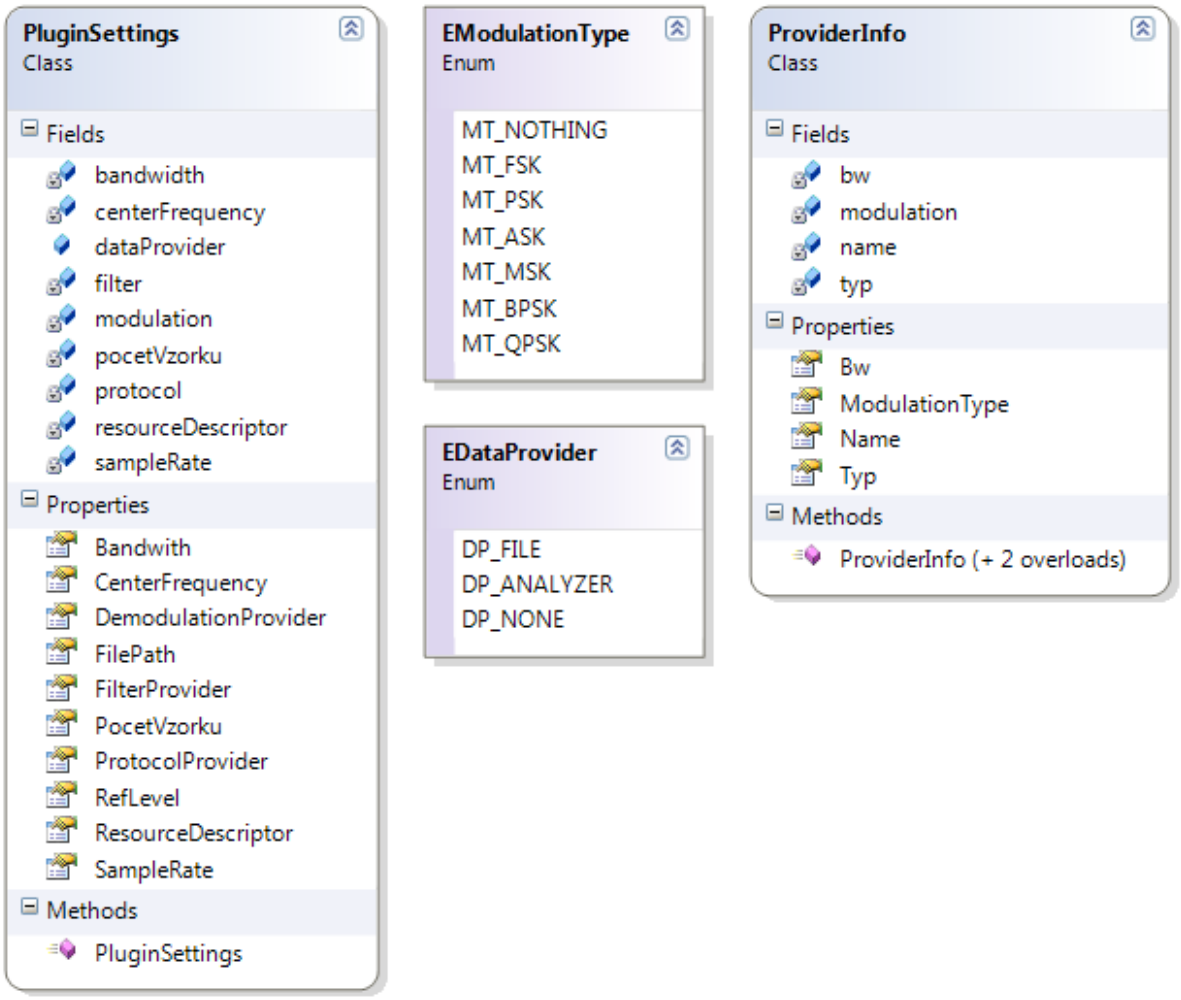
Nejdůležitější metodou je `getDataFrame()`. Metoda vrací nový datový rámec protokolu nejnižší vrstvy jako pole bytů, nebo hází výjimku v případě dosažení konce dat. V rámci metody může být také při prvním zavolání hledána bitová synchronizace.

4.6 Životní cyklus zásuvného modulu

Analyzátor protokolů při svém spuštění prohledává specifické adresáře s cílem nalézt dynamické knihovny nesoucí nějaké zásuvné moduly. Následně se provede načtení jejich datových typů, podle kterých jsou potom informace o modulech uloženy do tří seznamů – seznam pro datové typy vstupních modulů, pro datové typy výstupních modulů a pro `ProtocolFactory` (viz kapitola 2). Inicializace zásuvného modulu pak proběhne při jeho prvním použití.

Základní třídou zásuvného modulu, popisovaného touto prací je třída `IQLoaderPlugin` (viz kapitola 4). Jádro analyzátoru tedy vytvoří tuto třídu a pomocí její vlastnosti `Name` zjistí název zásuvného modulu. Protože se objekt této třídy vytváří v okamžiku, kdy ještě není jisté, zda bude použit k načítání dat nebo ne, provádí konstruktor jen základní inicializace. Jedná se o vytvoření objektu třídy `PluginSettintgs`, který ponese nastavení celého modulu a zjištění informací o dostupných providerech.

Provideři se musí nacházet v adresáři `Plugins`, umístěném v pracovním adresáři analyzátoru protokolů. Obdobně jako u celého analyzátoru, i zde jsou nejprve z knihoven načteny informace o datových typech providerů. Získané informace jsou poté roztříděny do tří seznamů podle toho, od které výchozí třídy typ dědí. Zde se jedná o seznamy Filter providerů, Demodulation providerů a Protocol providerů. V těchto seznamech jsou informace neseny v objektech třídy `ProviderInfo`. Třída je ukázána na obrázku 23.



Obrázek 23: Třída PluginSettings nesoucí nastavení zásuvného modulu. Výčtové typy pro druhy modulací a zdroje dat. Třída ProviderInfo obsahující informace o vlastnostech jednoho provideru.

Tato třída kromě samotného datového typu provideru a jeho textového názvu nese také informace zjištěné pomocí statických metod, které musí provider implementovat (viz předchozí kapitoly). Zejména se jedná o typ modulace, vyjádřený pomocí výčtového typu `EModulationType`.

Poté, co uživatel nastaví zásuvný modul pomocí dialogu, je aktualizováno nastavení zásuvného modulu v objektu třídy `PluginSettings`. Tato třída obsahuje tři objekty třídy `ProviderInfo` – pro každý typ externího provideru jeden. Do nich se uloží informace o skutečně nastavených providerech. Kromě toho třída nese ještě informace o zdroji dat, reprezentovaném výčtovým typem `EDataProvider`.

4.6.1 Inicializace providerů

Samotná inicializace se provádí až v metodě `OpenDevice()`, která je jádrem analyzátoru volána těsně před samotnou analýzou. V rámci této metody se nejprve provede ověření veškerého nastavení. Ověřování je nutné provádět jak v nastavovacím dialogu, tak i zde. Vzhledem ke způsobu ovládání analyzátoru totiž uživatel vůbec nemusí nastavovací dialog vyvolat. Zásuvný modul pak zůstane nastaven do výchozích hodnot, které ovšem neumožňují jeho činnost. V tom případě `OpenDevice()` vrací `false` a k vlastnímu načítání dat vůbec nedojde.

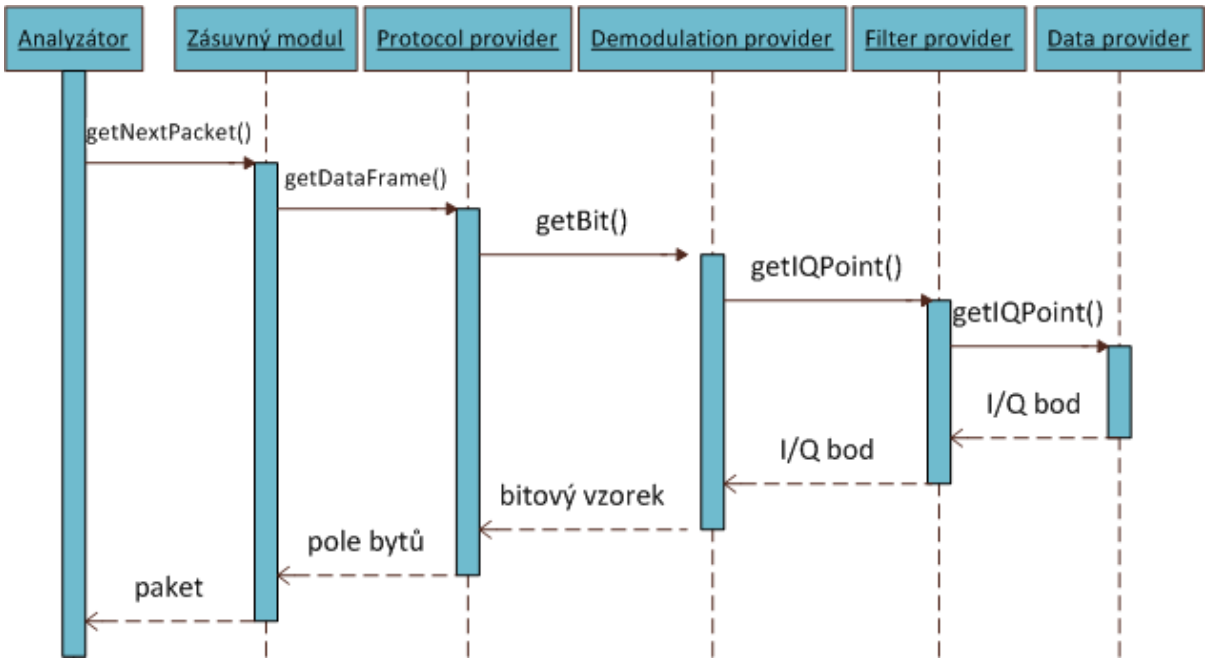
Pokud při ověřování nastavení nebyl nalezen žádný problém, je provedena vlastní inicializace providerů. V situaci, kdy dojde k chybě během inicializace, provider dle stanoveného rozhraní vyhodí výjimku `ProviderInitializationException`. Tato byla vytvořena pro potřeby zásuvného modulu a měla by se vyhazovat vždy, pokud dojde k chybě například v konstruktoru provideru.

Inicializace je prováděna postupně od začátku řetězce zpracování dat. Nejprve je tedy vytvořen příslušný Data provider a je provedena jeho inicializace. Poté je vytvořen Filter provider a je mu v podobě rozhraní `IQDataProvider` předána reference na již vytvořený Data provider. Dále je vytvořen Demodulation provider a je mu také v podobě rozhraní `IQDataProvider` předána reference na již inicializovaný filtr. Nakonec je vytvořen a inicializován Protocol provider, kterému je obdobně předána reference na demodulátor prostřednictvím rozhraní `IDemodulationProvider`.

Reference na inicializované providery jsou kromě toho uchovány i v třídě `IQLoaderPlugin`. Zde ale ve formě jejich bázových tříd.

4.6.2 Způsob zpracování dat

Vlastní načítání dat je spuštěno po úspěšném dokončení funkce `OpenDevice()` (viz předchozí kapitola) a probíhá tak, že jádro analyzátoru stále dokola volá metodu `getNextPacket()` dokud tato nevrátí `null`, což znamená konec dat. Hierarchii volání jednotlivých částí modulu zjednodušeně ilustruje sekvenční diagram na obrázku 24.



Obrázek 24: Postup volání providerů během načítání dat

Protože má každý provider referenci na svého předchůdce v řetězci, volají se postupně sami v opačném směru, než jak tečou zpracovávaná data. Jak je patrné z obrázku, volání `getNextPacket()` vede na volání `getDataFrame()` nad Protocol providerem. V rámci `getDataFrame()` se musí postupně načíst celý datový rámec protokolu nejnižší vrstvy, volá se tedy opakovaně `getBit()`. Voláním `getBit()` se aktivuje demodulátor, který potřebuje na odvození jednoho bitového vzorku například dva I/Q body v případě FSK modulace. Dochází tedy k dvojímu zavolání filtru, konkrétně jeho metody `getIQPoint()`. Ta následně vede na volání metody `getIQPoint()` Data provideru. Jeho chování je pak závislé na tom, jestli se načítá ze souboru nebo přímo z analyzátoru.

V prvním případě se přečte jeden řádek z fyzického souboru, zpracuje se, vytvoří se objekt třídy `IQPoint` a ten se pošle filtru jako návratová hodnota.

V druhém případě má Analyzer data provider již načten požadovaný počet I/Q bodů ve své vyrovnávací paměti a čte body pouze z ní.

Zpracovaná data se pak předávají v řetězci ve formě návratových hodnot volaných metod.

4.6.3 Ukončení práce

Poté, co je dosaženo konce načítaných dat, je Data providerem (ten tuto skutečnost zjistí) vyhozena výjimka `EndOfStreamException`, která postupně projde všemi úrovněmi až na úroveň zásuvného modulu do metody `getNextPacket()`. Tato metoda výjimku zachytí a jako návratovou hodnotu vrátí `null`, čímž informuje jádro analyzátoru o dosažení konce dat.

Analyzátor následně volá metodu `CloseDevice()`. V této metodě se uvolňují zdroje alokované v souvislosti s načítáním dat v rámci jedné analýzy. Provideři jsou postupně zrušeni, zásuvný modul se dostává do stavu, v jakém byl před zavoláním `OpenDevice()` a je připraven pro další analýzu.

Nastavení zásuvného modulu zůstane zachováno, protože jádro analyzátoru by mělo instanci zásuvného modulu držet načtenou v operační paměti. Modul je zrušen až v okamžiku, kdy uživatel zvolí pro vstup dat nějaký jiný. V tom případě je modul popisovaný touto prací bez problémů uvolněn, protože po dokončení volání `CloseDevice()` by neměly zůstat alokovány žádné dodatečné systémové prostředky.

5 Popis prototypového řešení

V rámci ověření funkčnosti námi navrženého zásuvného modulu jsme implementovali prototypové řešení pro *trunkovou* bezdrátovou síť *MPT-1327.* Tuto síťovou technologii využívá Dopravní podnik města Brna pro komunikaci s jednotlivými vozidly v rámci svého řídícího informačního systému. Neměl by tedy být problém se získáváním signálu sítě. O *trunkových* sítích obecně pojednává následující kapitola.

5.1 Trunkové sítě

Základní filosofie *trunkových* sítí by se dala shrnout citátem [10]: „Balík kanálů je dostupný všem uživatelům systému, když o něj požádají. Kanál je na žádost přidělen uživateli a po uvolnění je vložen zpět do balíku volných kanálů“. Toto je také hlavní rozdíl oproti klasickým bezdrátovým sítím, kdy má každý uživatel (nebo skupina uživatelů) systému přidělen svůj kanál.

Výhodami klasického řešení jsou nízká cena, jednoduchá technologie a snadnost použití. Mezi nevýhody patří například neefektivní využití a problematické sdílení přenosových kanálů.

Oproti tomu trunkové sítě využívají tzv. *simulcast* – všechny základnové stanice v dané oblasti využívají stejné frekvence. Zároveň všechny dostupné frekvence jsou sdíleny všemi uživateli – *trunking*. Potom existuje jeden kontrolní kanál, který je spravován kontrolérem. Přes tento kanál je řízen provoz na ostatních komunikačních kanálech, kde již probíhá vlastní přenos dat. Na kontrolním kanále jsou naladěni všichni uživatelé, kteří právě neprovádějí hovor.

Kontrolní kanál je většinou složen ze dvou frekvencí. Jedna je pro uplink (od uživatele na základnovou stanici) a druhá pro downlink (ze základnové stanice k uživateli). Celá komunikace se dá rozepsat do následujících kroků:

1) Uživatel posílá kontroléru požadavek na hovor.
2) Kontrolér přijme požadavek, vybere jeden z volných komunikačních kanálů a všem uživatelům z hovorové skupiny pošle příkaz, aby se přeladili na daný kanál.
3) Všechny uživatelské stanice poslouchající downlink kontrolního kanálu obdrží tento příkaz a ty, kterých se to týká, se přeladí na požadovaný kanál.
4) Po dokončení komunikace se stanice přeladí zpět na kontrolní kanál a použitý přenosový kanál je označen opět jako volný.

5.2 Protokol MPT-1327

MPT-1327 je standard pro pozemní bezdrátové trunkové sítě. Definuje způsob komunikace mezi uživatelskými stanicemi a řídícím kontrolérem (*TSC*). Standard specifikuje pouze způsob bezdrátové komunikace a na návrh samotného systému klade minimální požadavky. Odtud se taky odvíjí, že není nutné implementovat celý standard, ale jen jeho požadované části. Přesnou specifikaci standardu lze nalézt v [2].

Standard umožňuje jak klasickou hovorovou, tak datovou komunikaci. Pro přenos je použita FSK modulace s bitovou rychlostí 1200 bit/s. Jsou použity dvě frekvence – 1200 Hz pro logickou nulu a 1800 Hz pro logickou jedničku.

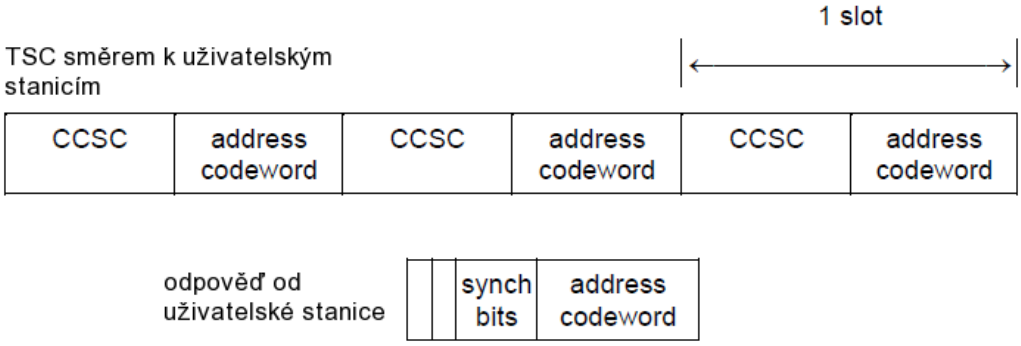
5.2.1 Struktura kontrolního kanálu

Aby bylo zajištěno, že se uživatelské stanice nebudou při komunikaci na kontrolním kanálu rušit, je čas na tomto kanálu rozdělen do časových slotů. Každý slot trvá 106.7 milisekund, což odpovídá přenosu 128 bitů. V každém slotu pak může být odeslána jedna zpráva, přičemž vysílání na downlinku probíhá nepřetržitě, aby mohly uživatelské stanice odvozovat časování slotů. Uživatelská stanice může přijmout zprávu v jednom slotu a odeslat odpověď v následujícím slotu. O přidělování slotů pojednává blíže kapitola 5.2.3.1.

Každý slot downlinku kontrolního kanálu obsahuje dvě kódová slova po 64 bitech (viz obrázek 25):

a) *Control Channel System Codeword (CCSC)* – CCSC obsahuje údaje o identifikaci základnové stanice a celé sítě. Dále poskytuje synchronizační bity pro následující kódové slovo.
b) *"Address" codeword* – Toto kódové slovo následuje bezprostředně po CCSC a nese informace o podstatě zprávy.

Některá základní kódová slova jsou popsána v kapitole 5.2.3.



Obrázek 25: Provoz na kontrolním kanálu.

Obě kódová slova můžou být vyměněna za datová slova v případě delších zpráv. Vždy však platí, že maximálně dvě CCSC (tedy dva sloty) za sebou mohou být nahrazena datovými slovy.

5.2.2 Způsob přenosu dat na technické úrovni

V základu se dá přenos specifikovat obrázkem 26. Nejprve je *Link Establishment Time (LET).* Zde je minimálně po dobu 6 bitových intervalů přenášen signál nedefinované modulace o síle alespoň 90% maximálního vysílacího výkonu.

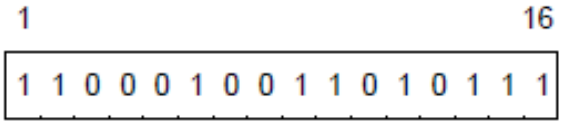
Dále následuje *Preamble* – hlavička, která je tvořena posloupností minimálně šestnácti střídajících se jedniček a nul. Posloupnost slouží k odvození bitové synchronizace a měla by končit nulovým bitem.

Poté již následuje samotná zpráva tvořená *codeword synchronization sequence* (SYNC) - viz níže. Za ní je odvysíláno adresové slovo a můžou následovat ještě nějaká data. Na konec je připojen jeden bit značící konec komunikace (H).

Obrázek 26: Přenos zprávy

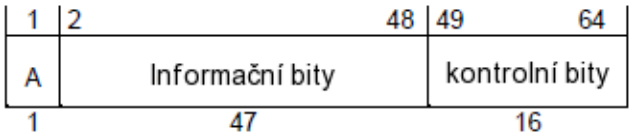
Jak bylo zmíněno výše, před adresovým slovem je odesláno *SYNC*, které slouží k označení začátku rámce – adresového slova na kontrolním kanálu. SYNC je posloupnost 16 bitů, ukázaná na

obrázku 27. Kromě SYNC existuje ještě *SYNT*, které má obdobný účel, ale používá se na komunikačním kanálu k označení začátku dat [2].



Obrázek 27: SYNC slovo

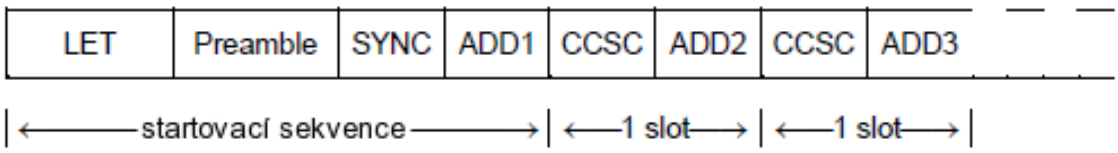
Obecný formát zprávy je na obrázku 28. První bit každé zprávy označuje, zda se jedná o adresové nebo datové slovo. Pokud je nastaven na 1, jedná se o adresové slovo, jinak je to datové slovo. Následuje 47 bitů vlastní zprávy a nakonec je připojen *kontrolní součet* (*CRC*) [2].



Obrázek 28: Obecný formát zprávy

Mezi základní kódová slova patří CCSC, které identifikuje základnovou stanici a poskytuje synchronizaci pro následující adresové slovo. V případě CCSC jsou kontrolní bity napevno nastaveny na SYNC a posledních 16 informačních bitů je nastaveno na střídání jedniček a nul, aby tvořily hlavičku (preamble). Zbylé informační bity jsou dopočítávány tak, aby jejich kontrolní součet dal právě SYNC. Konec CCSC slova je tedy tvořen stejnou bitovou posloupností, jako začátek přenosu.

Díky tomu je již možné řadit více zpráv za sebou, protože mezi každými dvěma zprávami je vloženo právě CCSC (pokud nebylo nahrazeno daty). Obrázek 29 ilustruje celý scénář od začátku vysílání. Z obrázku je zřejmé, že nejprve probíhá LET, dále je vysílána bitová synchronizace, následuje SYNC a poté již odvysíláno první adresové slovo. Pak již probíhá standardní vysílání, kdy je v první části slotu vysláno CCSC a za ním následuje adresové slovo. Obdobou CCSC je *DCSC*, které má stejný účel, ale používá se na datovém kanálu spolu se SYNT.



Obrázek 29: Celá sekvence zpráv na kontrolním kanálu

Pokud tedy začne některá uživatelská stanice sledovat provoz na downliku kontrolního kanálu později, než je zahájeno vysílání, měla by nejpozději po úspěšném odposlechnutí některého CCSC být schopná odvozovat bitovou synchronizaci a časování slotů.

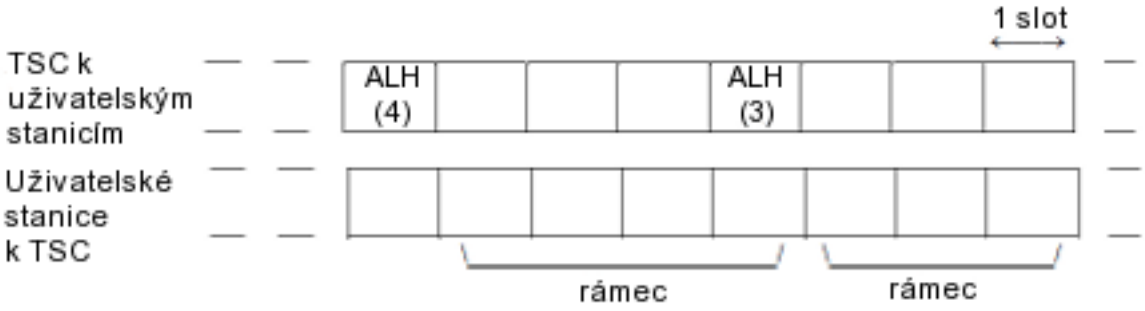
5.2.3 Přehled vybraných adresových kódových slov

Adresová slova se dají rozdělit do devíti kategorií. Jedná se například od zprávy „*ALOHA*“, jejich potvrzení (*ACK*), žádosti o provedení hovoru (*RQS*), kontroly dostupnosti uživatelských stanic (*AHY*) nebo požadavek na přechod na komunikační kanál (*GTC*). Úplný výčet zpráv a jejich přesný popis lze najít v oficiálním standardu [2].

Dále budou popsány některé vybrané zprávy, pro které byla definována pravidla v analyzátoru protokolů.

5.2.3.1 ALOHA

Tyto zprávy zasílá TSC na kontrolním kanálu. Zpráva *ALOHA* slouží k vyhrazení specifikovaného počtu slotů na uplinku kontrolního kanálu. Uživatelské stanice si pak můžou vybrat kterýkoliv z vyhrazených slotů a uskutečnit vysílání. Pokud dojde ke konfliktu, opakují zainteresované uživatelské stanice vysílání v některém z následujících slotů. ALOHA zpráv je více typů. Zde popsaná obecná ALOHA vyhrazuje sloty pro jakoukoliv komunikaci. Různé podtypy můžou vyhrazovat sloty například pouze pro tísňová volání. Obrázek 30 ukazuje příklad použití zprávy ALOHA.



Obrázek 30: Vyhrazení rámců pomocí zprávy ALOHA

Na obrázku je vidět, že nejprve je rezervován rámec o délce čtyři sloty. Během něj můžou uživatelské stanice vysílat svá kódová slova. V posledním slotu rámce TSC vyšle další zprávu ALOHA, tentokrát rezervující rámec o délce tři sloty.

5.2.3.2 Potvrzovací zprávy (ACK)

TSC zasílá ACK jako odpověď na různé zprávy zaslané uživatelskými stanicemi v průběhu sestavování hovoru. Dále můžou tuto zprávu zasílat i uživatelské stanice jako odpověď pro požadavek na odezvu. I zde platí, že kromě obecného potvrzení je několik podtypů specifických pro určité situace. Potvrzovací zpráva může plnit i funkci ALOHA zprávy a kromě potvrzení vyhrazovat i sloty pro další rámec.

5.2.3.3 Zpráva Go To Traffic Channel (GTC)

Tuto zprávu zasílá TSC konkrétním uživatelským stanicím a přikazuje jim přeladit se na specifikovaný komunikační kanál, kde pak probíhá vlastní hovor. Zpráva taky může být zaslána i již komunikujícím stanicím v situaci, kdy mají v komunikaci pokračovat na jiném komunikačním kanálu.

5.3 Implementace providerů

V prototypovém řešení jsme se zaměřili na analýzu datových toků na downlinku kontrolního kanálu sítě MPT-1327, který by se měl v našem případě nacházet na frekvenci 460.69 MHz. Zaměřili jsme se na analýzu několika nejčastějších zpráv, které jsou popsány v kapitole 5.2.3 a pro tyto zprávy definovali pravidla. Dále jsme implementovali Protocol provider pro rozpoznávání kódových slov kontrolního kanálu. Zároveň jsme napsali demodulátor FSK podle informací z kapitoly 3.4.

5.3.1 Implementace Protocol provideru

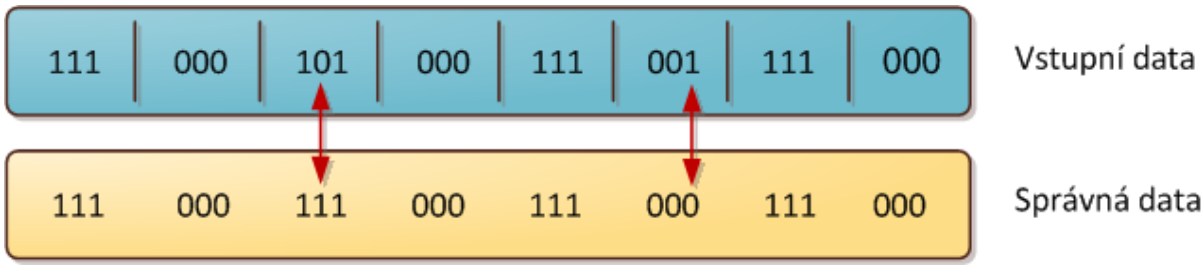
Jak bylo zmíněno v kapitole 4.5, Protocol provider přebírá od demodulátoru bitové vzorky, kdy obecně více vzorků dává dohromady jeden datový bit. Implementovaný poskytovatel z toho vychází.

Při jeho inicializaci se nejprve z dostupného nastavení vzorkovací frekvence a ze znalosti bitové rychlosti vypočte počet bitových vzorků na jeden bit. Pokud se zjistí, že počet vzorků na jeden bit není celé číslo, je vyhozena výjimka `ProviderInitializationException`. Stejná výjimka je vyhozena rovněž pokud se zjistí, že počet bitových vzorků na jeden bit je sudé číslo. Důvod, proč musí být toto číslo liché, je zmíněn níže. Zároveň, pokud vyjde jeden vzorek na bit, je vypsáno varovné hlášení o možnosti velké nepřesnosti. Obecně tedy platí, že vzorkovací frekvence by měla být lichým násobkem bitové rychlosti větším než jedna.

Základním problémem, který jsme řešili, bylo nalezení bitové synchronizace a SYNC, které označuje začátek kódového slova. Jednou z variant bylo použít konečný automat. Takové řešení bylo ovšem málo robustní a nedokázalo si poradit v situaci, kdy vstupní posloupnost bitových vzorků obsahovala chybu. Navíc bylo poměrně složité navrhnout takový automat, který by i v případě, že všechny bitové vzorky jsou správné, dokázal vystihnout všechny možné situace dle standardu.

Další možností bylo implementovat nějaký sofistikovaný algoritmus, který by ve vstupním proudu nejprve nalezl nejdelší vyhovující posloupnost střídání jedniček a nul. Z této pozice by se pak vyšlo a zjišťovalo by se, zda po takového posloupnosti následuje SYNC neboli začátek kódového slova. Tento algoritmu by rovněž bylo velmi problematické implementovat kvůli jeho komplikovanosti. Navíc by byl velmi pomalý a pro rychlé analyzování dat nevhodný.

Nakonec tedy bylo navrženo kompromisní řešení, které se nachází někde mezi výše zmíněnými variantami. Implementovali jsme buffer, jehož velikost se odvíjí od velikosti synchronizační posloupnosti (16 bitů) a počtu vzorků na jeden bit. Tento buffer se nejprve celý naplní daty. Následně se zkusí aplikovat na načtená data bitová synchronizace a je spočítáno, kolik bitových vzorků nevyhovuje. Pokud je množství špatných vzorků pod stanovenou hranicí, je prohlášeno, že bitová synchronizace byla nalezena. V případě, že je množství špatných vzorků větší, je z bufferu smazán nejlevější (nejstarší) bitový vzorek, obsah celého bufferu je posunut o jedno místo doleva a na konec je načten nový vzorek. Celá procedura se pak opakuje od začátku. Rozhodovací práh je počítán dynamicky při inicializaci provideru jako 20% z celkové velikosti testovacího bufferu. Princip nalezení bitové synchronizace ilustruje obrázek 31, kde je pro zjednodušení použit buffer pro 8 bitů a každý bit je tvořen třemi vzorky. Jak je vidět, v tomto případě nevyhovují dva bitové vzorky z 24. Bitová synchronizace by tedy v tomto případě identifikována byla.



Obrázek 31: Hledání bitové synchronizace

Poté co byla nalezena bitová synchronizace, je nutné ověřit, že za ní následuje SYNC. Pro jednoduchost jsme přepokládali, že bitová synchronizace není delší než 16 bitů. Tento požadavek by měl být splněn téměř pokaždé, protože bitová synchronizace během již probíhajícího vysílání je zajištěna pomocí CCSC (viz kapitola 5.2.2).

Samotné hledání kódového slova probíhá obdobným způsobem jako u hledání bitové synchronizace. Nyní již ale známe bitové intervaly, můžeme se tedy v úrovni abstrakce posunout výše a nahlížet na data již jako na bity. Ty jsou načítány tak, že se z modulátoru načte počet vzorků

odpovídající jednomu bitovému intervalu. Protože jsou splněny podmínky definované v počátku této kapitoly (lichý počet vzorků na bit), můžeme na základě převažujícího typu vzorku rozhodnout, zda se jedná o logickou jedničku nebo nulu. Vstupní bity jsou poté porovnány se vzorovou posloupností a obdobně, jako u hledání bitové synchronizace, i tady se počítá množství špatných vzorků. Pokud je počet špatných vzorků pod stanovenou hranicí (20%), je prohlášeno, že začátek kódového slova byl nalezen. V případě, že tomu tak není, vrací se celý algoritmus zpět do fáze hledání bitové synchronizace, protože se předpokládá, že byla v proudu dat nalezena jen částečná, náhodná podobnost s počátkem kódového slova.

Jakmile je nalezen začátek kódového slova, je již možné začít načítat samotné kódové slovo, které je dlouhé 64 bytů. Dvě volání `getDataFrame()` by proto v tomto případě měla přečíst jeden slot. Zde je nejprve nutné ze vstupní posloupnosti bitů poskládat jednotlivé byty. To se děje pomocí jednoduchých operací bitových posuvů a logických bitových operací, kdy jsou načtené bity nasouvány do proměnné velikosti jeden byte. Jakmile je takto načteno celé kódové slovo, je předáno do zásuvného modulu v podobě pole bytů. Pro jednoduchost bylo vypuštěno počítání kontrolního součtu.

5.3.2 Implementace Protocol factory pro MPT-1327

V případě MPT-1327 je Protocol factory jednoduchá, protože existuje vždy pouze jeden protokol nejnižší datové vrstvy. Při zpracovávání kódového slova se tedy automaticky vytváří objektový model `Protocols.mptTest.mpt1327`, který je určen pro analyzování všech výše zmíněných typů kódových slov protokolu MPT-1327.

Pokud bychom požadovali možnost analyzovat veškeré typy kódových slov dle celého standardu, nabízelo by se rozdělit protokol na nejnižší vrstvě na adresová a datová slova.

5.3.3 Demodulátor a filtr

U demodulátoru jsme implementovali principy FSK demodulace, nastíněné v kapitole 3.4. Demodulation provider při prvním zavolání `getBit()` načte dva I/Q body z filtru. Z těchto dvou bodů spočítá pomocí vzorce z kapitoly 3.4 změnu fáze a podle výsledku provede přiřazení buď logické jedničky, nebo nuly. Při následujících voláních již načítá z filtru pouze jeden I/Q bod a využívá bod načtený v předchozím kroku.

Protože se nám nikdy nepodařilo zachytit na frekvenci 460.69 MHz signál uspokojivé intenzity a kvality, nebylo možné námi navržený demodulátor dostatečně otestovat a ověřit jeho funkcionalitu. Vytvořili jsme proto ještě jeden pomocný demodulátor, který bitové vzorky načítá z textového souboru. Aby nebylo nutné takový textový soubor vytvářet ručně, implementovali jsme jednoduchý nástroj *Frame Creator.* Nástroj dokáže vytvářet kódová slova popsaná v kapitole 5.2.3, včetně možnosti vkládat chybné bitové vzorky, čímž je možné simulovat rušení.

V našem prototypovém řešení jsme pro jednoduchost zvolili výchozí filtr, který je zabudován přímo do zásuvného modulu a provádí kopírování svého vstupu na svůj výstup.

6 Závěr

V rámci této práce bylo naším úkolem nastudovat problematiku bezdrátových sítí, seznámit se se spektrálním analyzátorem Rohde&Schwarz FSQ8 a navrhnout softwarový prostředek – zásuvný modul, který by zmíněný spektrální analyzátor propojoval s analyzátorem protokolů implementovaným v rámci diplomové práce, jejímž autorem je Ing. Peter Jurnečka. Aby se prokázala funkčnost řešení, bylo pro takto navrhnutý zásuvný modul implementováno zkušební prototypové řešení pro trunkovou síť MPT-1327.

Práce nejprve poskytnula ucelený pohled na rozhraní poskytované analyzátorem protokolů pro zásuvné moduly. Potom byly popsány dva typy podporovaných zásuvných modulů a jejich účel. Důraz byl kladen především na popis rozhraní pro vstupní moduly a jeho sémantiku. Dále byly zběžně načrtnuty ostatní části analyzátoru protokolů a nastíněna jejich funkčnost. Zde se zejména jedná o jádro analyzátoru a způsob definice pravidel meta modelu protokolů.

Další část práce popsala rozhraní spektrálního analyzátoru a způsob jeho použití. Na základě těchto poznatků byly implementovány softwarové prostředky pro komunikaci s tímto analyzátorem. Kromě toho byla implementována možnost načítat vstupní data z textového souboru. Obě možnosti byly otestovány v reálných podmínkách a byla tak ověřena jejich správná funkcionalita. Největší problém zde byl s implementací provideru poskytujícího vstup přímo z analyzátoru, protože je nutné na cílovém počítači zprovoznit ovladače rozhraní VISA a RSIB respektive VXI PnP. Navíc je nezbytné mít pro účely testování k dispozici zmíněný spektrální analyzátor.

Poté byly popsány základní typy analogových a digitálních modulací a způsob vyjádření signálů ve formě fázové a kvadraturní složky. Nakonec kapitola ukázala, jak v teoretické rovině funguje I/Q modulátor a demodulátor.

V další kapitole byl poskytnut detailní pohled na návrh zásuvného modulu. Modul byl navržen tak, aby byl snadno přenositelný na jinou síťovou technologii a bylo možné jednoduše rozšiřovat jeho funkcionalitu přidáváním dalších demodulací, filtrů a protokolů. Toho jsme dosáhli tak, že jsme identifikovaly některé klíčové fáze zpracování dat a tyto části vyčlenili do samostatných dynamických knihoven. Tyto knihovny jsou pak již zaměnitelné a je možné dopsat si vlastní za předpokladu dodržení stanoveného rozhraní.

Funkčnost systému jako celku byla poté ověřována v prototypovém řešení pro trunkovou síť MPT-1327. Práce zběžně popsala obecné principy fungování trunkových sítí a na základě těchto informací stručně prezentuje fungovaní protokolu MPT-1327. Zaměřili jsme se především na popis komunikace na kontrolním kanálu a technické záležitosti týkající označení začátků kódových slov a odvození bitové synchronizace. Na základě tohoto popisu jsou pak navrženy algoritmy pro načítání kódových slov, jakožto datových rámců protokolu nejnižší vrstvy.

V rámci prototypového řešení jsme mimo jiné implementovali FSK demodulátor, čímž jsme zužitkovali informace teoreticky představené v kapitole 3. Nepodařilo se nám však zachytit na frekvenci 460.69 MHz dostatečné kvalitní signál, takže nebylo možné tento demodulátor uspokojivě otestovat a odladit. Protože jsou demodulátory poskytovány v podobě dynamických knihoven, může si případný uživatel se znalostmi konstrukce demodulátorů snadno implementovat vlastní softwarový demodulátor.

Abychom umožnili alespoň částečnou funkcionalitu prototypového řešení, implementovali jsme ještě jeden pomocný demodulátor, který načítá bitové vzorky z textového souboru a předává je

Protocol provideru. Protože by bylo velmi pracné a komplikované vytvářet takový soubor, byl implementován pomocný nástroj, který dokáže snadno generovat kódová slova zadaných parametrů v podobě bitových vzorků a tyto pak ukládat do souboru, který je již přímo použitelný.

Jako další možnosti rozšíření a zdokonalení lze určitě zmínit lepší algoritmy pro odvozování bitové synchronizace a především pro hledání začátku kódového slova. Zde by se dala inspirace najít například v algoritmech pro vyhledávání podřetězce v řetězci.

Další možností by bylo rozšířit množinu zabudovaných providerů o demodulátory pro základní typy modulací. Spolu s tím, by se rovněž daly rozšířit možnosti uživatelských nastavení, zejména z oblasti nastavení spektrálního analyzátoru.

Literatura

[1] JURNEČKA, Peter. *Analyzátor protokolů řízený pravidly*. Brno, 2009. 48 s. Diplomová práce. Vysoké Učení Technické v Brně, Fakulta Informačních Technologií.

[2] MPT 1327. *A Signaling Standard for Trunked Private Land Mobile Radio Systems*. [s.l.] : [s.n.], January 1988. 290 s. Dostupné z WWW: <http://www.ofcom.org.uk/static/archive/ra/publication/mpt/mpt_pdf/mpt1327.pdf>.

[3] *R&S®FSQ Signal Analyzer: Operating Manual*. Munich (Germany): Rohde & Schwarz GmbH & Co. KG, 2009. 826 s. Dostupné z WWW: <http://www2.rohde-schwarz.com/file_5543/FSQ%20Operating%20Manual%20English%20FW%204.55.pdf>.

[4] Specifikace IP protokolu RFC 791 [online] Aktualizováno 1981-09-01 [cit. 2008-12-30]. Dostupné na URL: <http://www.faqs.org/rfcs/rfc791.html >

[5] Hillebrand F.: GSM and UMTS, The Creation of Global Mobile Communications, John Wiley & Sons, 2002

[6] Digital Modulation in Communications Systems: An Introduction. In . [s.l.] : [s.n.], 2001 [cit. 2010-04-28]. Dostupné z WWW: <http://cp.literature.agilent.com/litweb/pdf/5965-7160E.pdf>.

[7] HANÁČEK, Petr. *Základy bezdrátové komunikace* [online]. Brno, 2009. 58 s. Referát. Vysoké Učení Technické v Brně, Fakulta Informačních technologií.

[8] *NI Developer Zone* [online]. Apr 14, 2009 [cit. 2010-04-29]. What is I/Q Data?. Dostupné z WWW: <http://zone.ni.com/devzone/cda/tut/p/id/4805>.

[9] *Signal Analyzer FSQ : Specifications* [online]. Version 05.00. [s.l.] : Rohde&Schwarz, April 2005 [cit. 2010-05-02]. Dostupné z WWW: <http://www.testbuyer.com/pdf/specs.cfm?pdf_id=5C0159B7>.

[10] HANÁČEK, Petr. *Bezdrátové telekomunikační sítě* [online]. Brno, 2009. 57 s. Referát. Vysoké Učení Technické v Brně, Fakulta Informačních technologií.

[11] *World Wide Web Consorcium* (W3C) [online]. 2010, 2010/03/14 08:37:32 [cit. 2010-05-08]. Extensible Markup Language (XML). Dostupné z WWW: <http://www.w3.org/XML/>.

[12] Jack E. Volder, *The CORDIC Trigonometric Computing Technique* [online], IRE Transactions on Electronic Computers, pp330-334, September 1959. Dostupné z WWW: <http://www.jacques-laporte.org/Volder_CORDIC.pdf>

[13] TROJANOVIČ, Vladimír. *Spracovanie AM a FM signálov s využitím princípov softvérového rádia*. [s.l.], 2006. 63 s. Diplomová práce. TECHNICKÁ UNIVERZITA V KOŠICIACH, FAKULTA ELEKTROTECHNIKY A INFORMATIKY, Katedra elektroniky a multimediálnych telekomunikácií.

[14] *Microsoft Visual C# 2005 : krok za krokem*. vydání první. Brno : Computer Press, 2006. 528 s. ISBN 80-251-1156-3.

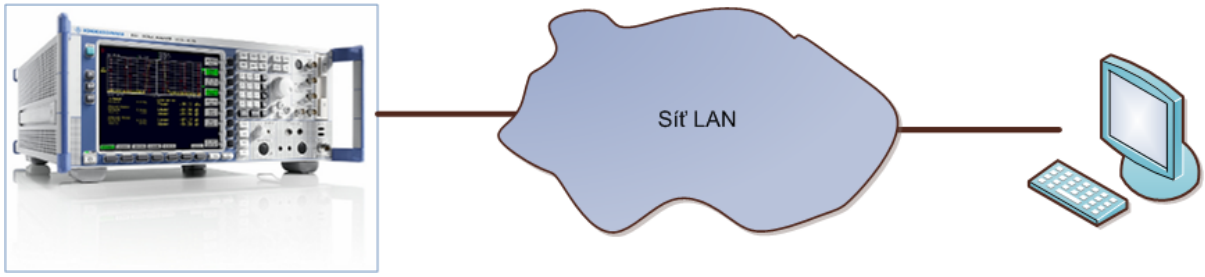
[15] NI-VISA : Programmer Reference Manual [online]. March 2003 Edition. [s.l.] : National Instruments, 2003 [cit. 2010-05-09]. Dostupné z WWW: <http://www.ni.com/pdf/manuals/370132c.pdf>.

[16] J. Kirkhorn, “Introduction to IQ-demodulation of RF-data,” Tech. Rep., IFBT, NTNU, September 15, 1999.

Seznam příloh

1) Způsob zapojení spektrálního analyzátoru a instalace ovladačů
2) Obsah přiloženého CD

1) Způsob zapojení spektrálního analyzátoru a instalace ovladačů



Pro připojení lze použít:

a) **Vzdálenou plochu**
   Na Spektrálním analyzátoru běží operační systém Windows XP firmy Microsoft. Je tedy možné použít vzdálenou plochu.

b) **Rozhraní VISA s ovladačem VXIplug&play**
   Spektrální analyzátor může být s PC propojen prostřednictvím ethernetové sítě. Na PC je nutné mít kromě Analyzátoru protokolů nainstalován ovladač rozhraní VISA a VXIplug&play.

Instalace rozhraní VISA

1) Ovladač je možné stáhnout z http://joule.ni.com/nidu/cds/view/p/id/1605/lang/en.
2) Pro stažení je nutná bezplatná registrace.
3) Po stažení ovladač nainstalujte klasickým způsobem.

Instalace ovladače VXIPnP a wrapperu RSSpecAn

Ovladač je k dispozici ke stažení na adrese:
http://www2.rohde-schwarz.com/en/service_and_support/Downloads/Drivers/?type=25&downid=1809
Zde se také nachází množství ukázkových programů, pro práci s wrapperem RSSpecAn.

Zjednodušený postup získávání signálu ve formě I/Q dat přímo pomocí spektrálního analyzátoru

1) Určit nosnou frekvenci (*Center frequency*)
2) Určit šířku pásma (*Bw*)
3) Nastavit *marker*
4) V sekci *Meas* nastavit režim I/Q dat.

2) Obsah přiloženého CD

Adresář `src` – Zdrojové kódy celého analyzátoru (adresář `AnalyzeThis`).

Významné podadresáře:

`Loaders.IQLoader` – zásuvný modul

`Loaders.IQLoader.mpt1327` – Protocol provider pro MPT-1327

`Loaders.IQLoader.mpt1327FSKdemod` – Demodulation provider pro MPT-1327

`Loaders.IQLoader.mptDemod` – Pomocný demodulátor

Adresář `text` – Elektronická verze textové části práce.

Adresář `bin` – Spustitelná verze analyzátoru včetně zásuvného modulu.