VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

EMULÁTOR MODEMU PRO AKCELERACI VÝVOJE VESTAVĚNÝCH ZAŘÍZENÍ

CELLULAR MODEM EMULATOR FOR ACCELERATION OF EMBEDDED SYSTEM DEVELOPMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **VÍT HORKÝ**
AUTHOR

VEDOUCÍ PRÁCE **doc. Ing. ZDENĚK VAŠÍČEK, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Horký Vít**
Program: Informační technologie
Název: **Emulátor modemu pro akceleraci vývoje vestavěných zařízení**
Cellular Modem Emulator for Acceleration of Embedded System Development
Kategorie: Počítačové sítě

Zadání:

1. Seznamte se s protokolem AT používaným pro komunikaci s modemy pro přístup k mobilní síti. Proveďte rešerši typických modemů používaných v oblasti IoT pro přístup k síti GSM, LTE a NbIoT a seznamte se s AT příkazy podporovanými těmito modemy.
2. Navrhněte emulátor protokolu AT komunikující skrze sériovou linku, který bude možné použít pro automatické testování kódu využívajícího knihoven pro obsluhu modemů používaných ve vestavěných zařízeních využívajících např. framework Zephyr. Rozhraní emulátoru navrhněte tak, aby bylo možné specifikovat různé testovací scénáře a zvolit konkrétní model emulovaného modemu.
3. Navržený emulátor implementujte s využitím OOP v podobě Python modulu. Dbejte na nezávislost na použité platformě a možnost rozšiřitelnosti o další typy modemů. Implementujte minimálně podporu pro modem BG96 včetně podpory tzv. GSM muxing.
4. Vytvořte demonstrační aplikaci, ukazující způsob využití navrženého emulátoru.
5. Diskutujte parametry navrženého řešení a možnosti dalšího rozšíření.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 a 2 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Vašíček Zdeněk, doc. Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 29. října 2021

Abstrakt

Cílem práce je vytvořit emulátor určený pro testování a vývoj knihoven pro komunikaci s mobilními modemy pomocí protokolu AT. Emulátor je implementován ve formě knihovny jazyka Python, která poskytuje podporu AT příkazů a GSM multiplexingu. Emulátor dosahuje podpory testování jiných knihoven a zařízení pomocí testovacích scénářů specifikovaných uživatelem. V základu je implementována emulace modemu Bg96 firmy Quectel s tím, že je kladen důraz na rozšiřitelnost o podporu dalších modemů.

Abstract

The purpose of this work is to create an emulator intended for testing and development of libraries for communication with cellular modems with support for AT protocol. This emulator is implemented as a python library and supports basic AT commands and GSM multiplexing. Support for testing other devices and libraries is achieved by user specified testing scenarios. By default, support for emulating modem Bg96 by Quectel is implemented but an emphasis is placed on the ability to support other modems.

Klíčová slova

modem, mobilní modem, IoT modem, emulátor modemu, protokol AT, GSM multiplexing

Keywords

modem, cellular modem, IoT modem, modem emulator, AT protocol, GSM multiplexing

Citace

HORKÝ, Vít. *Emulátor modemu pro akceleraci vývoje vestavěných zařízení.* Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Zdeněk Vašíček, Ph.D.

Emulátor modemu pro akceleraci vývoje vestavěných zařízení

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Ing. Zdeňka Vašíčka, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Vít Horký
11. května 2022

Poděkování

Rád bych poděkoval svému vedoucímu práce doc. Ing. Zdeňku Vašíčovi, Ph.D, který mi poskytnl velice cenné rady a nápady při pravidelných konzultacích.

Obsah

Kapitola 1

Úvod

Průběhem let přináší moderní technologie pro lidi čím dál více výhod. v současné době nastává snižování cen a rychlé rozšiřování tzv. IoT zařízení, které podporují vzájemnou komunikaci přes různé druhy sítí. Tato komunikace může probíhat přes klasické WiFi sítě, proprietární sítě s velkým dosahem (LoRa) a nebo sítě mobilní, jako je 4G LTE. Tato práce se zabývá vytvářením emulátoru pro tzv. GSM modem, což je zařízení často používané k poskytnutí konektivity k internetu, pomocí již zmíněných mobilních sítí.

Cílem práce je navrhnout tento emulátor jako software, který by zjednodušil testování knihoven pro obsluhu GSM modemu a zlepšil podporu ladění firmware, který s modemy pracuje. Doposud bylo nutné modem fyzicky vlastnit a jelikož různé modemy se liší v podporované funkcionalitě, může se vývoj značně prodražit, pokud jich knihovna nebo firmware chce správně podporovat více. Kromě samotné obsluhy modemu byla nad rámec přidána podpora chování a protokolů, které dovolují zařízení se pomocí emulátoru připojovat k internetu, přesněji PPP. Díky již zmíněnému odlišnému chování modemů, které jsou na trhu, je jednou z prioritou práce lehká rozšiřitelnost o novou funkcionalitu. Samotný emulátor by měl také podporovat testovací funkčnost, která dovoluje uživateli upravovat chování modemu tak, aby například bylo možné dosáhnout různých chybových stavů, které je u existujících modemů těžké vyvolat napřímo.

Práce se skládá z několika kapitol rozdělených podle tématu. Kapitola *Modem* 2 se zaměřuje na informace o modemech, technologiích a způsobech komunikace, které jsou nimi podporovány. Text této kapitoly obsahuje informace o postupném vývoji modemů přes léta a rozdílné technologie, které byly postupně přidávány. Kapitola *Existující řešení* 3 se zabývá emulátory modemů, které jsou už na trhu dostupné a zmiňuje, jaké mají výhody a nevýhody. v kapitolách 4 a 5 je řešena problematika návrhu emulátoru, informace o přístupu k jeho architektuře a samotná jeho implementace. Součástí detailů o implementaci je také zmíněno, jaké rozhraní a využití pro testování emulátor podporuje. Poslední kapitola 6 se zaměřuje na ověření funkčnosti emulátoru pomocí již existujícího zařízení, které implementuje funkcionalitu pro komunikaci s modemem. Otestována a zhodnocena je správnost komunikace s emulátorem a podpora připojení k internetu.

Kapitola 2

Modem

Modem je název tvořený zkrácením kombinace slov modulace a demodulace. z toho vyplývá že je toto zařízení používáno pro modulaci určitých signálů (většinou digitálních) a posílání přes různé druhy sítí (často analogové), kde jsou následně signály přijaté druhým modemem demodulovány do původní formy. Historicky se nejčastěji používaly modemy komunikující přes kabely používané telefonními sítěmi. v moderní době jsou vyráběny modemy podporující komunikaci přes mobilní sítě, například 3G, 4G LTE apod.

2.1 Typy modemů

Prvním komerčním typem modemů byly používané pro zasílání počítačových dat přes veřejné telefonní sítě, které tento způsob zasílání normálně nepodporovaly. Tyto modemy používaly modulaci dat do frekvencí podporovaných sítěmi telefonních operátorů. Často bylo po uživateli vyžadováno vytočit číslo a až následně začít zasílat data, tento problém byl v pozdějších letech vyřešen automatickým vytáčením. První modemy vznikaly v dvacátých letech v Americe hlavně za účelem zasílání informací mezi zpravodajskými agenturami[19]. Trvalo do šedesátých let, než byl vydán první komerční modem firmou AT&T. [4]

Druhou skupinou modemů jsou ty, které vznikaly zároveň s nástupem mobilních sítí. Tento druh modemů, také nazývané bezdrátové nebo mobilní, podporují přístup internetu bez nutnosti připojení kabelem k existující telefonní infrastruktuře. Tyto modemy také podporují mnohem vyšší rychlosti než jejich starší protějšky, které byly i ke konci své existence většinou limitovány na 56 kbit/s. Moderní mobilní sítě podporují rychlosti 100 mbit/s a s nástupem sítí 5G je teoreticky možné dosáhnout rychlosti až 10 gbit/s[14].

Mobilní modemy se rozdělují na několik kategorií:

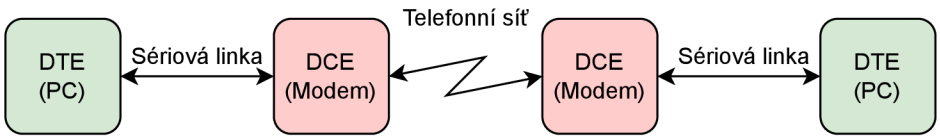
- Integrovaný - jedná se o modem přidaný do nějakého dalšího zařízení, nejčastěji WiFi routeru. Většinou není uživatelem konfigurovatelný, jedná se o jednoduchý způsob připojení k internetu bez nutnosti řešit nastavování modemu
- Smartphone - každý mobilní telefon obsahuje modem, často integrovaný na samotném SoC. Některé smartphony dovolují uživateli komunikovat přímo s modemem a tímto způsobem přistupovat k internetu z jiného zařízení. Tato funkcionalita se nazývá tethering a je často podporována pomocí USB nebo Bluetooth
- Samostatný - tento typ modemů je určený k přímému připojení k PC nebo podobnému zařízení. Většinou se připojení přes USB, ale starší typy podporovaly například PCI

- IoT - s nástupem IoT zařízení začaly vznikat modemy zaměřené na malou velikost a nízkou spotřebu. Jako samostatné modemy podporují komunikaci přes AT příkazy. Komunikace s nimi je prováděna jako s Dial-Up modemy přes sériovou linku, ale často podporují další rychlejší způsoby připojení k zařízení

2.2 Sériová linka

Historicky prvním a stále nejčastěji používaným způsobem komunikace mezi modemem a zařízením je sériová linka. Jedná se o jednoduchý způsob pro zasílání dat přes několik konektorů pomocí asynchronního komunikačního protokolu, často mezi dvěma modemy, viz 2.1. Data jsou rozdělena na bity, které jsou po jednom zasílány za sebou [2]. Pro správnou funkčnost sériového spojení jsou nutné alespoň tři vodiče, RX (recieve data), TX (transmit data) a GND (země). Při propojování zařízení je nutné tyto piny překřížit při propojování nebo speciálním konektorem, tedy aby byly spojené TX → RX a stejně z druhé strany.

Komunikace s modemem je specifikována ve standardu **RS-232** vydaném v roce 1960[7], který specifikuje počet vodičů, časování a podporované konektory. Také rozděluje dvě komunikující zařízení na DTE (Data Terminal Equipment) a DCE (Data Communications Equipment)[10]. DCE specifikuje modem se kterým DTE komunikuje. DTE je nejčastěji PC nebo v moderní době například mikroprocesor. Kromě vodičů RX a TX tento standard specifikuje i další vodiče, například ty co přenášení informace o schopnosti přijmout další data (CTS a RTS) nebo připravenost modemu zasílat data přes telefonní linku. Posledním vodičem je RI, který informuje DTE o příchozím hovoru.



Obrázek 2.1: Propojení dvou PC přes telefonní síť

Pro úspěšnou komunikaci s modemem ovšem není nutné implementovat funkčnost všech těchto vodičů. Piny modemu, které je nutné zapojit jsou pouze RX, TX a GND. Tímto se připravíme například o tzv. hardware flow control (piny CTS a RTS), ale dovoluje nám to připojit modem k různým zařízením, např. některým mikroprocesorům, které podporují pouze jednoduchou verzi sériové komunikace.

Sériové linky mají velké množství podporovaných rychlostí, historicky mezi 110 až 115200 bit/s. Vyšší z těchto rychlostí nebyly dříve dosažitelné, tak byla zařízení nastavena většinou na rychlosti pomalejší, například 9600. Starší modemy, které byly pomalejší než 9600 bit/s měly při jejich používání omezenou rychlost sériové linky na rychlost, kterou tento modem dokázal posílat data druhému zařízení[16]. Moderní implementace sériové komunikace podporují vyšší rychlosti mnohem lépe a velká část modemů má svoji rychlost při startu nastavenou na 115200 b/s. Díky novějším technologiím byly také podporované rychlosti rozšířeny a to nejčastěji až po 921600 b/s. v závislosti na délce vodičů a podle možného rušení nemusí být všechny rychlosti praktické z důvodu zvýšené chybovosti.

2.3 AT příkazy

AT příkazy byly vyvinuty pro produkt Smartmodem firmy Hayes vydaný v roce 1981. Jedná se o sadu jednoduchých textových příkazů kterými se dá modem přes sériovou linku ovládat. Před jejich vznikem bylo nutné, aby byl modem konfigurován manuálně a bylo očekáváno, že vytáčení telefonních čísel bude prováděno uživatelem. Pokud bylo nutné vytáčení nebo příjímání telefonních hovorů automatizovat, musel uživatel modemu zakoupit speciální zařízení které to dovolovalo. Jednou z inovací AT příkazů bylo právě to, že dovolovaly vytáčet telefonní čísla přímo z PC.

2.3.1 Historie a standardizace

První AT příkazy, tak pojmenovány, protože většina z nich začíná znaky 'AT' neboli 'attention' anglicky[8], implementovaly pouze základní funkčnost modemu. Vytáčení čísel, přijetí hovoru apod. jsou příkladem těchto prvních jednoduchých příkazů. Další funkčností, podporovanou modemy z této doby, jsou různé konfigurační příkazy měnící chování modemu, nastavení formátu zasílaných odpovědí na příkazy a konfigurace telefonní linky pomocí tzv. s registrů.

S příchodem novějších modemů začaly být přidávány také další AT příkazy, které je možné rozpoznat od těch základních pomocí rozdílné syntaxe. Většina příkazů přidaných v této době obsahuje znak `&`, který hned následuje řetězec `AT`. Mnoho z těchto příkazů sloužilo ke konfiguraci a jejímu ukládání do tzv. profilů. Tyto příkazy byly přidávány individuálně každým výrobcem a často se stávalo, že modemy od několika výrobců mezi sebou nebyly kompatibilní, proto došlo k vzniku formálního standardu TIA/EIA-602.

Při nástupu mobilních sítí a modemů je podporující se objevila nutnost podporovat speciální funkčnost přidanou těmito sítěmi. z tohoto důvodu začaly vznikat standardy, přesněji `GSM 07.07`, který specifikuje AT příkazy pro ovládání GSM telefonu nebo modemu a `GSM 07.07`, který popisuje AT příkazy pro podporu zasílání SMS zpráv. Příkladem příkazů přidaných v těchto standardech mohou být třeba ty na zadání pinu, jeho změna, zjištění stavu modemu, čísla IMEI, síly signálu, zasílání SMS apod.

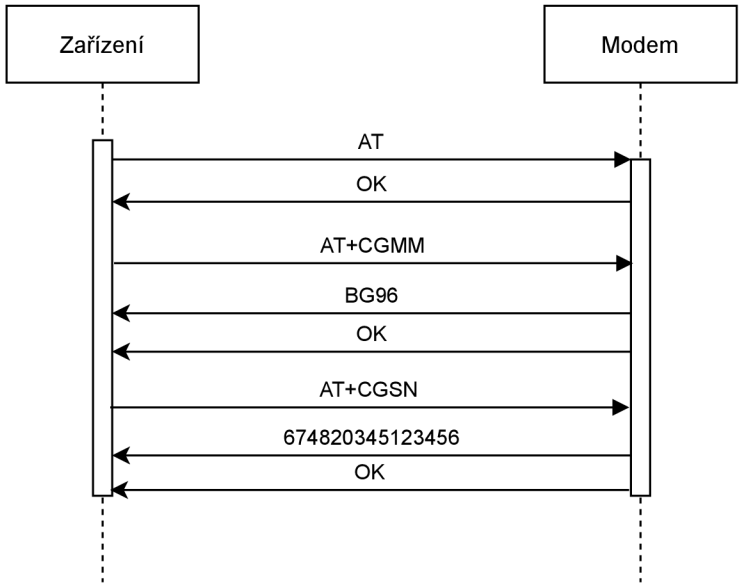
Současné modemy podporují alespoň AT příkazy ze specifikace `GSM 07.07`, i když kolik příkazů je doopravdy implementováno záleží na výrobci a nemusí být podporovány všechny. Mnoho firem vyrábí modemy, které podporují navíc sadu proprietárních příkazů, které jsou specifikovány výrobcem. Často se jedná o speciální konfigurační příkazy, které například dovolují nastavovat režimy nízké spotřeby nebo další funkce podporované pouze jedním výrobcem nebo modelem. Jelikož nejsou tyto příkazy v žádné specifikaci, jsou často obsažené v manuálu, který výrobce vydává. Tento dokument obsahuje nejen další příkazy podporované výrobcem, ale i speciální chování a další technologie které modem obsahuje. Existují také rozšíření pro AT příkazy, u kterých výrobci nechtějí, aby se o nich informace volně šířily, proto nejsou v manuálech volně dostupných na internetu a často jejich dokumentace závisí na podepsání dohody o mlčenlivosti[20]. Proprietární příkazy můžeme poznat pomocí speciální syntaxe. Často se stává, že začínají na určité písmeno. Například příkazy od výrobce modemů Quectel začínají písmenem Q.

2.3.2 AT spojení

Před příchodem Smartmodemu firmy Hayes na trh podporovaly modemy pouze tzv. manuální režim. Každé zařízení mělo dva módy, mezi kterými se dalo přepínat. První mód se označoval originate, do kterého uživatel modem přepnul pokud chtěl zahájit spojení. Ná-

sledně bylo nutné ručně vytočit číslo modemu se kterým chceme navázat spojení. Druhý mód se nazývá answer a je určený k příjímání spojení pocházejících od druhého zařízení. Před nástupem osobních počítačů nebyl tento způsob ovládání modemu problematický, protože modemy byly často napřímo propojené spolu a tedy bylo nutné zapamatovat si pouze jedno telefonní číslo. To se později ovšem změnilo s nástupem osobních počítačů a každý modem dovoloval komunikaci s tisíci dalšími zařízeními. Díky tomu bylo nutné, aby se dala čísla vytáčet z osobního počítače připojeného k modemu. Původně vzniklo mnoho způsobů jak toho dosáhnout, ovšem mnoho návrhů požadovalo dedikovaný port pouze pro nastavování modemu. Další návrhy se zabývaly použitím speciálních pinů na konektoru modemu, které by byly určené pro jeho nastavování. Nakonec žádný z těchto přístupů neuspěl a začaly se využívat AT příkazy, které dovolovaly komunikovat přes klasickou sériovou linku, kterou měl každý osobní počítač alespoň jednu.

Vznikem AT příkazů byla komunikace s modemem rozdělena na dva režimy. Command mode, viz 2.2, využíván pro posílání samotných příkazů pro operování s modemem. Druhý režim, data mode, který je určen k zasílání všech příchozích dat ze sériové linky na vzdálený modem. Při startu je modem v režimu Command, kdy uživatel může volně posílat AT příkazy a provádět konfiguraci. Následně je možné přepnout do datového režimu pomocí vytočení čísla vzdáleného modemu. Pokud je spojení úspěšně navázáno, modem zašle zprávu `CONNECT` následovanou číslem specifikující rychlost spojení. Všechna další data zaslaná modemu přes sériovou linku jsou přeposlána vzdálenému modemu. Pokud chceme datový režim opustit, zašleme escape sekvenci `+++`. Modem se následně vrátí do režimu command a můžeme opět zasílat AT příkazy. Tato akce ovšem pouze zastavila zasílání dat a spojení se vzdáleným modemem je stále aktivní. Pokud ho chceme obnovit, zašleme modemu příkaz `ATO` který spojení obnoví nebo příkaz `ATH` který ho přeruší.



Obrázek 2.2: Zjištění modelu a sériového čísla modemu pomocí AT příkazů

Příkazy v command režimu jsou po sobě posílány jako řetězce oddělené znakem `<CR>`. Každý příkaz může mít definovanou odpověď, například modem odpoví informacemi o jeho výrobci. Tyto odpovědi přichází ve formátu `<CR><LF><odpověď><CR><LF>`. Následně modem zašle informaci o úspěchu nebo neúspěchu příkazu ve formě kódu informující o stavu.

Výsledný kód může být ve více formátech. Některé modemy jsou v základu nastavena na detailní režim výpisu, angl. verbose, který kód vypisuje v lidsky čitelném formátu. Většinou se jedná o odpověď `OK` při úspěchu nebo `ERROR` při chybě. Existují ovšem další kódy, které jsou používány pro jiné případy. Příkladem může být `CONNECT`, který informuje uživatele o navázání spojení s dalším modemem a jeho rychlosti. Formátování těchto kódů je stejné jako ostatní odpovědi modemu, tedy z obou stran jsou odděleny znaky `<CR><LF>`. Modem má také režim neverbózní, který zasílá kódy v kratším formátu vhodným spíše pro zařízení. Na rozdíl od čitelných zpráv jsou zasílané číslice, které mají stejný význam. Na rozdíl od verbózního formátu odpovědí mají tyto zprávy jinou syntaxi. Modem je zasílá ve formátu `<kód><LF>`. Existuje ještě jeden formát zasílání chybových kódů přidaný standardy GSM, který je podporován pouze některými příkazy ze stejného standardu. Formát těchto odpovědí je `<příkaz>: <chybový kód/zpráva>` a stejně jako ve verbózním módu jsou odpovědi oddělovány znaky `<CR><LF>`. Podobně jako klasické kódy má tento syntax dva formáty. Jeden lidsky čitelný a druhý zasílající číslice namísto zpráv. Modemy implementují velké množství těchto chybových kódů dovolujících zařízení zjistit detailní důvod chyb. To je na rozdíl od klasických kódů, kterých je pouze malé množství a jejich zprávy nejsou velmi popisné.

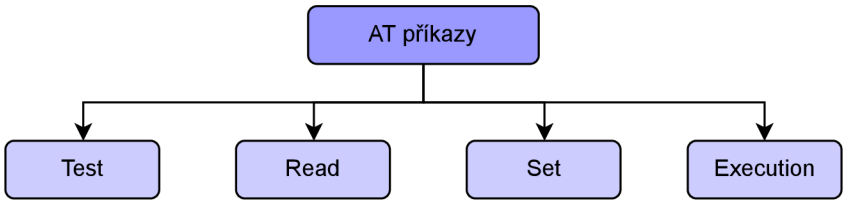
2.3.3 Syntaxe

Existují tři syntaxe příkazů: basic, S-syntax a extended syntax. Typ příkazů Basic a S-syntax byly implementovány už zařízením za doby Smartmodemu a tedy byly standardizované v Hayes Command Set. Extended příkazy byly přidány s nástupem GSM sítí a standardizovány v `GSM 07.07`.

Basic syntax má tvar `AT<x><n>` nebo `AT&<x><n>`, kde `<x>` je příkaz a `<n>` je jeho parametr ve formě číslice. Tyto parametry nejsou povinné a pokud nebudou specifikovány, tak je použita výchozí hodnota. Příkladem těchto příkazů může být `ATE`, který specifikuje, jestli by modem měl posílat příchozí příkazy zpět odesílateli přes sériovou linku. Má parametr s hodnotami 0 a 1. Základní hodnota tohoto příkazu a všech ostatních je uvedena v manuálu daného modemu.

Příkazy typu S-syntax jsou určené k nastavování hodnot tzv. s registrů a jsou ve formátu `ATS<n>=<m>`, kde `<n>` je číslo registru a `<m>` je hodnota na kterou ho chceme nastavit. Po spuštění modemu mají s registry předdefinované hodnoty, které můžeme následně změnit při běhu.

Extended příkazy se rozdělují do čtyř skupin podle typu funkčnosti, každá s odlišnou syntaxí[6], viz 2.3.



Obrázek 2.3: Typy extended AT příkazů

Operace test

Testovací operace je určena ke zjištění, jestli je daný AT příkaz podporován GSM modemem. Všechny příkazy s extended syntaxí podporují tuto operaci. Syntaxe zaslaného AT příkazu pro tuto operaci je `<příkaz>=?`, například `AT+CGMI=?`. o podpoře příkazu je zařízení informováno pomocí zprávy `OK`, když je podporován. Pokud modem příkaz nepodporuje, tak odpoví kódem `ERROR`

Pokud testovaný příkaz má parametry, jsou všechny jejich možné hodnoty zaslány jako první, až následně je zaslán kód `OK`. Syntaxe této informace je `<příkaz>: <parametr 1>,<parametr 2>`. Samotné pole parametrů obsahují seznam všech možných hodnot, který parametr podporuje oddělené čárkami. Pokud se jedná o nepřerušenou řadu čísel, může modem zaslat interval vyznačený znakem `-` mezi dvěma hodnotami.

Operace read

Operace *read* je určena pro získání hodnot parametrů příkazu. Pro její provedení je zaslán AT příkaz ve formě `<příkaz>?`. Stejně jako operace *test* má odpověď formu jména příkazu odděleného od seznamu parametrů dvojtečkou. Na rozdíl od ní jsou seznamy zobrazující všechny podporované možnosti nahrazeny hodnotami současnými. Pro některé AT příkazy může platit, že operace *read* vrátí více hodnot než pouze ty, které jsou uživatelem nastavitelné. Příkladem může být příkaz `AT+QCSCON`, který je určený k zobrazení současného stavu připojení k mobilní síti. Jeho operace *read* kromě nastavitelného parametru zasílá také informaci o stavu připojení.

Operace set

Operace *set* je určena k nastavování hodnot parametrů extended AT příkazů. Parametry příkazu zašleme jako pole oddělené čárkami následující znak `=` po názvu příkazu. Pokud není zařízení komunikující s modemem informované o počtu a podporovaných hodnotách parametrů, může je zjistit pomocí operace *test*. Pokud zašle hodnoty příkazu *set* jiné, než jsou specifikované v manuálu a vráceny testovacím příkazem, nebude operace provedena a bude zpět zaslána informace o chybě. o úspěšném provedení *set* příkazu je zařízení informováno přijetím odpovědí `OK`, což znamená, že hodnota je úspěšně uložena do paměti modemu.

Operace execution

Asi nejčastěji používanou operaci je *execution*. Syntaxe této operace je stejná jako jméno příkazu. Například pro použití operace *execution* příkazu `AT+CSQ` stačí modemu zaslat stejný řetězec znaků. z jejich názvu už vychází, že jsou tyto operace určené hlavně k prováděný různých akcí, které můžou například měnit stav modemu. Dalším častým využitím pro tyto operace je zjišťování informací o modemu nebo jeho interním stavu. Příkladem je třeba příkaz `AT+CSQ`, který odpovídá informacemi o síle signálu a chybovosti spojení. Zajímavě je tato informace zaslána stejně jako odpověď při zavolání operace *read*. Tyto operace se z hlediska sémantiky dají zpracovávat jako příkazy typu `basic`. Jediným rozdílem mezi nimi je, že příkazy typu extended podporují jiný typ chybových hlášek.

2.3.4 Řetězení

Modemy podporující Hayes command set také podporují tzv. řetězení příkazů. Jedná se o spojení více AT příkazů do jednoho. Tato funkce se používá nejčastěji na inicializaci modemu pomocí při začátku spojení. Určité modemy mají často omezení jaké příkazy a kolik jich je možné odeslat za sebou. Může se jednat například o limit 40 znaků díky omezení velikosti vyrovnávací paměti pro příjem.

Zřetězené AT příkazy jsou odděleny pomocí znaku ‘;‘ a na rozdíl od komunikace, kdy zasíláme příkazy po jednom, začíná pouze první příkaz řetězcem `AT`. Všechny ostatní příkazy v sekvenci mají tento řetězec odstraněn a pokud by modem detekoval jeho existenci někde jinde než na začátku, ohlásí zařízení chybu.

2.3.5 Příkazy od výrobců

Přestože existuje několik specifikací AT příkazů, výrobci si často přidávají vlastní příkazy určené pouze pro své modemy. Tyto příkazy lze často rozpoznat pomocí speciální syntaxe. Historicky se používaly symboly %, $, #, *, & a \pro jejich vyznačení. Po standardizaci příkazů pro GSM s názvy začínající na znak + se začalo pro rozpoznání těchto příkazů používat první písmeno názvu příkazu. Například firma Quectel používá příkazy začínající znaky +Q.

Příkladem výrobcem implementovaných příkazů můžou být ty, které přidávají způsoby ovládání hardware, které nebyly stanoveny ve specifikaci. Například příkaz pro vypnutí modemu nebo speciální režimy nízké spotřeby pro modemy typu IoT. Dalším příkladem může být rozšíření funkčnosti SMS, například zřetězené zprávy, které jsou v textovém režimu modemem Bg96 podporovány díky příkazu `AT+QCMGS` dovolující specifikovat identifikátor a jeho mnohonásobným voláním zřetězit zaslané zprávy. Opačná funkcionalita je implementována pomocí příkazu `AT+QCMGR` určeného ke čtení SMS zpráv po jednotlivých segmentech.

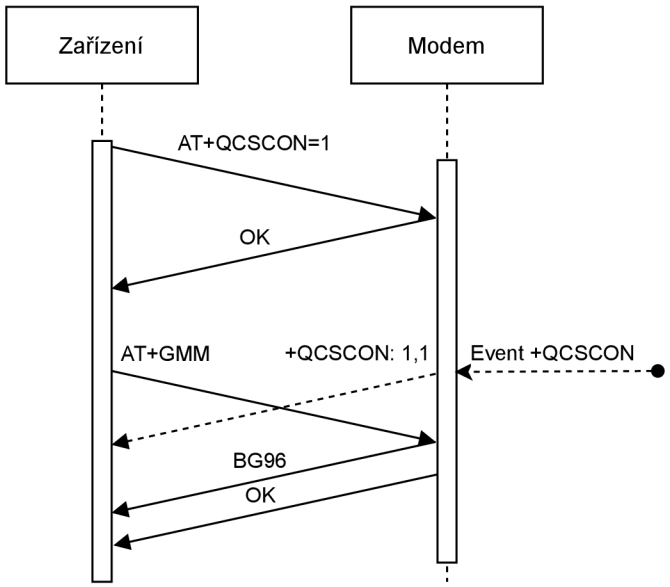
2.3.6 Asynchronní informace zaslané modemem

Kromě odpovědí vyžádaných ze strany zařízení existují také zprávy, které zasílá modem nezávisle na AT příkazech zaslaných ze strany zařízení. To tedy znamená, že nezávisle na stavu komunikace může modem zaslat zprávu informující zařízení o nějaké události. Často se jedná o zprávy informující o změně stavu sítě, připojení SIM karty a také o příchozích telefonních hovorech. Modem je možné konfigurovat pomocí různých příkazů, které nastavují jestli by se tyto informace měly zasílat. Při vypnutí této funkcionality se můžeme připravit o důležité informace o stavu modemu, jako je například odpojení od sítě.

Jelikož tyto zprávy přichází nezávisle na komunikaci a nejde předvídat kdy budou doručeny, je často jejich zpracování problematické. Není triviální implementovat parser pro AT komunikaci, který dokáže zpracovávat i tyto nečekané stavové kódy[12]. Jedinou garancí danou modemem je, že nikdy nepošle nevyžádaný stavový kód v průběhu zpracování jednoho AT příkazu. Tedy v časovém rozmezí mezi jeho přijetím a odesláním odpovědi máme zaručeno, že modem se nepokusí zasílat žádné asynchronní informace. Tato záruka ovšem moc neznamená, jelikož je nutné počítat se zpožděním komunikace přes sériovou linku. Někdy se může stát, že zařízení a modem zašlou AT příkaz a nevyžádaný stavový kód ve velmi malém časovém rozmezí. V tomto případě si může zařízení myslet, že informace patří k zaslanému AT příkazu a může dojít k problémech při zpracování, viz 2.4.

Tato skutečnost komplikuje situaci jednoduchým parserům, které očekávají pouze jeden příchozí stavový kód pro AT příkaz. Může se také stát, že má zařízení omezenou vyrovná-

vací paměť pro příjem odpovědí a kombinace dlouhé zprávy a více stavových kódů může vést k přetečení. Pro tento problém existuje více způsobu řešení s různou náročností pro implementaci. Nejjednodušším je zasílat AT příkazy ve větším rozmezí (>100ms), což zvyšuje šanci, že URC nebude přijat mezi příkazem a jeho odpovědí. Nevýhodou ovšem je, že tento přístup zpomaluje komunikaci. Lepší volbou je nevypínat echo modemu, jak často zařízení dělají z důvodu snížení komunikace přes sériovou linku. Díky tomu zařízení vidí průběh komunikace ze strany modemu. a kvůli garanci nezasílat URC při zpracování AT příkazu si můžeme být jistí, že ze strany modemu nebude žádný kód zaslán mezi AT příkazem a jeho odpovědí.



Obrázek 2.4: Přijetí URC před odpovědí na AT příkaz

2.4 GSM multiplexing

GSM multiplexer (také CMUX) je režim modemu určený pro současnou komunikaci se zařízením na více nezávislých kanálech. Jak již bylo zmíněno, modem podporuje režimy command a data. Je ovšem nutné mezi nimi přepínat a tak přerušit komunikaci pokud chceme např. zjistit sílu signálu. Toto historicky nebyl problém, protože se inicializace prováděla před přepnutím do data režimu a do command režimu jsme se často nevraceli. Problém to může být například pokud zařízení používající modem chce zobrazovat sílu signálu při jeho používání. Museli bychom tedy modem přepnout zpátky do command režimu a zjistit informace o připojení tímto způsobem. Přepínáním mezi režimy dochází k výpadku datového spojení a může například vypadnout hovor pokud k tomuto modem používáme. Také může být narušena komunikace přes TCP/IP, pokud modem držíme v command režimu déle než je maximální prodleva pro komunikaci, může být spojení přerušeno.

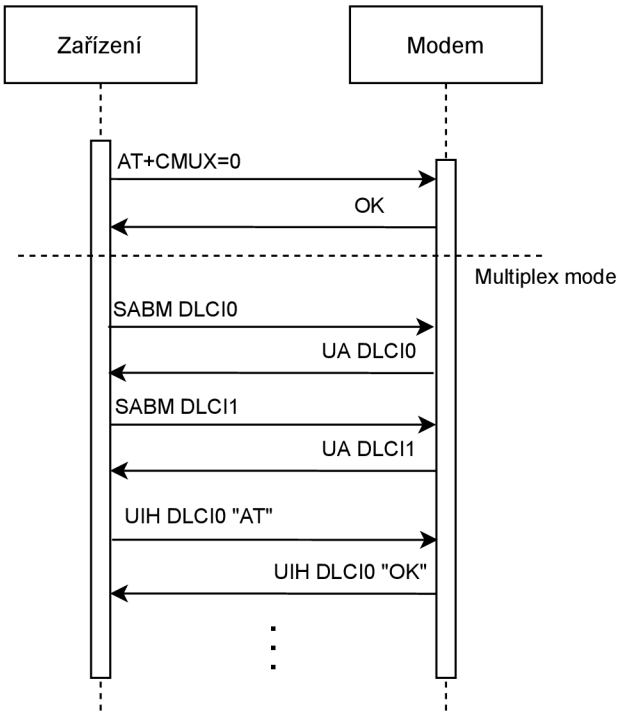
Všechny tyto problémy řeší GSM multiplexing, který dovoluje komunikovat naráz na více kanálech. Nejjednodušším příkladem jeho využití je právě využívání command a data módu zároveň, ale tento protokol podporuje mnohem více funkčnosti, hlavně z ohledu konfigurace. Jelikož byl tento režim designován pro mobilní modemy, bere důraz na podporu režimu nízké spotřeby.

2.4.1 CMUX protokol

Kanály pro GSM multiplexing jsou každé označené pomocí unikátního identifikátoru. Kromě unikátního identifikátoru jsou zasílána další pole, viz 2.5. Identifikátor je šestibitové číslo které je posíláno v každém rámci, aby zařízení poznalo jak by měl být paket interpretován. Počet DLCI je závislý na implementaci, ale většinou je podporováno několik pro data a AT příkazy. Často jsou také specifické DLCI, které podporují například SMS. Tyto kanály mají pevně dané číslo a uživatel modemu jej musí najít v manuálu.

Obrázek 2.5: Struktura paketu GSM multiplex

Pro vytvoření spojení je použit příkaz `AT+CMUX`, který přepne modem do multiplexing módu. Od zařízení se následně očekává poslání speciálního rámce, který inicializuje kanál podle zaslaného DLCI. Příklad této inicializace na obrázku 2.6. Kanál se většinou otevře v AT režimu, ale výrobci toto chování přesněji specifikují v manuálu a existují kanály pracující v jiných režimech než AT.



Obrázek 2.6: Navázání spojení přes GSM multiplex protokol

Protokol rozlišuje dva typy paketů pro posílání dat. Rozdíl mezi nimi je ve způsobu výpočtu kontrolního součtu. Podle specifikace musí každá implementace podporovat alespoň tzv. UIH pakety, jejichž kontrolní součet je počítán pouze pro hlavičku. Druhý typ paketů je UI, u kterého je nutné spočítat kontrolní součet pro celý paket, tedy i data v něm obsažená. Tento způsob zasílání dat zvyšuje spolehlivost, ale může být náročný na hardware, proto ho nemusí všechny modemy implementovat. Také nemusí být vhodný pro použití,

pokud zasílaná data implementují svojí vlastní korekci chyb, což by znamenalo že by byla vykonávána práce několikrát.

2.4.2 Frame check sequence

Každý rámec zasílaný v režimu GSM multiplexingu obsahuje 8 bitů tzv. Frame check sequence (FCS). Algoritmus používaný pro tuto potřebu je CRC (Cyklický redundantní součet). PPP režim používá stejný algoritmus, ale zasílá 16 bitů namísto 8.

Cyklický redundantní součet je algoritmus pro detekci chyb používaný digitálními sítěmi a úložnými zařízeními pro detekci a opravu neúmyslných změn dat. Tento algoritmus závisí na výpočtu zbytku dělení polynomů[5], který je přidaný na konec každé zprávy. Při jejím přijetí je proveden stejný výpočet a pokud výsledky nesedí můžeme poté následně provést opravu chyb.

CRC je specificky určeno na ochranu proti častým typům chyb nacházejících se v komunikačních kanálech, kde nabízejí rychlý a dostatečný způsob kontroly integrity zprávy. Ovšem není vhodné pro ochranu proti úpravě dat, jelikož útočník může jednoduše upravit zprávu a opět vypočítat součet bez způsobu detekce. Dalším problémem v tomto ohledu je, že CRC je lehce reverzibilní funkce, což znamená že není vhodná pro digitální podpisy.

Výpočet n-bitového CRC probíhá výběrem polynomu n-tého řádu, kde je n jak počet bitů, tak i řád polynomu. Přepíšeme polynom do formy, kde jsou jeho koeficienty vyjádřeny jako číslice binárního čísla u kterého je nejvyšší bit nejvyšším řádem polynomu. Sekvenci dat, pro kterou chceme CRC vypočítat doplníme o n bitů zprava s hodnotou nula. Sekvenci bitů polynomu přiřadíme k sekvenci dat tak, aby k sobě nejvyšší bity obou sekvencí seděly. Následně provedeme pro každý bit ze sekvence dat operaci XOR s bitem z polynomu, všechny ostatní bity ignorujeme. Výsledek si uložíme, posuneme sekvenci bitů reprezentující polynom o tolik bitů, aby seděla sekvence s nejbližší binární číslicí 1. Tato sekvence kroků je prováděna dokud nejsou všechny bity výsledku, kromě těch přidaných v začátečních krocích, 0. Tyto přidané výsledky jsou následně výsledkem dělení polynomů a hodnota n-bitové CRC funkce. Ověření zprávy se provádí stejným procesem, ale na rozdíl od přidání n počtu číslic 0, je přidaná vypočítaná hodnota CRC funkce. Pokud po opětovném výpočtu vychází hodnota celého výsledku 0, tak si můžeme být jisti že nedošlo k žádným detekovatelným chybám.

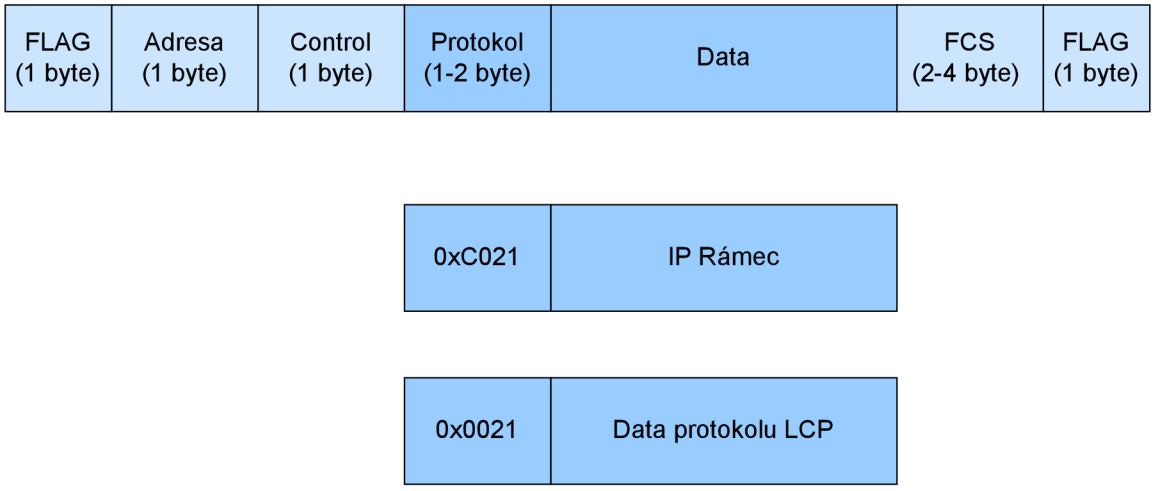
Pro výběr vhodného polynomu na provedení výpočtu existují různé standardy. Algoritmus se často zrychluje pomocí předvypočítání tzv. lookup table. Tato tabulka, indexovaná nejvyššími koeficienty výsledku dělení, odstraňuje nutnost vyhodnocování operací pro každý bit. Nejčastěji se používá tabulka velikosti 256.

2.5 PPP režim

Point-to-Point je protokol linkové vrstvy určený pro přímou komunikaci mezi dvěma routery bez jakékoliv sítě nebo zařízeními mezi nimi. Tento protokol podporuje mnoho funkcí, včetně šifrování, autentifikace a komprese dat.

PPP je používán na více typech fyzických sítí, proto má několik implementací podle způsobu přenosu dat na nižší vrstvě. Nejpoužívanější jsou Point-to-Point Protocol over Ethernet (PPPoE) a Point-to-Point Protocol over Serial (PPPoS), který je používán modemy pro komunikaci se zařízením snažící se přistupovat k internetu.

PPP je často používaný protokol linkové vrstvy pro synchronní a asynchronní komunikační kanály. Nahrazuje zastaralé protokoly sériových linek, jako je například SLIP a protokoly standardizované telefonními společnostmi (LAPB). PPP protokol byl designován s podporou mnoha protokolů síťové vrstvy, původně IP, TRILL, IPX, atd. Jako SLIP, tento protokol dovoluje plné internetové spojení přes telefonní linky, nebo mobilní sítě, použitím modemu. Na rozdíl od protokolu SLIP podporuje detekci chyb zaslaných paketů a poškozené pakety zasílá znovu. Pro využití PPP protokolu je na modemu nutné vytočit speciální



Obrázek 2.7: Struktura PPP paketu

telefonní číslo, kdy se poté modem přepne do režimu kdy přijímá a zpracovává PPP pakety. Samozřejmě je možné používat PPP režim v jednom z kanálů CMUX.

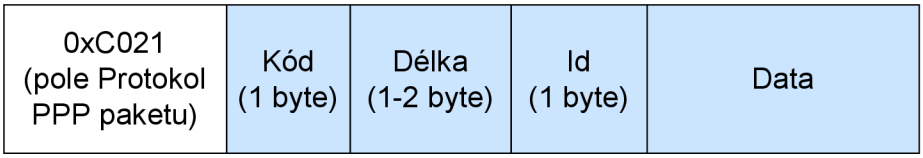
PPP podporuje několik protokolů, které se pomocí něj dají využívat, viz 2.7. Často používanými jsou LCP (Link Control Protocol) a IPCP (Internet Protocol Control Protocol), které jsou používané ke konfiguraci spojení. Dále je podporován protokol IP, určený k zasílání dat. Každý paket zaslaný pomocí PPP protokolu má pole obsahující kód, který označuje, jaký protokol je paketem přenášen. Kromě již zmíněných podporovaných protokolů, existuje velké číslo dalších, každý označený vlastním kódem.

Kód protokolu	Jméno
0021	Internet Protocol
002b	Novell IPX
002d	Van Jacobson Compressed TCP/IP
002f	Van Jacobson Uncompressed TCP/IP
8021	Internet Protocol Control Protocol
802b	Novell IPX Control Protocol
8031	Bridging NC
C021	Link Control Protocol
C023	Password Authentication Protocol
C223	Challenge Handshake Authentication Protocol

Tabulka 2.1: Protokoly, které možné zasílat s použitím PPP

2.5.1 Ustanovení spojení

PPP ustanovuje spojení pomocí protokolu LCP, dovolujícího zařízením k nastavení základních hodnot ovládajících chování komunikace. Protokol dovoluje automatickou konfiguraci rozhraní každé strany. Příkladem může být nastavovaní velikosti rámců, tzv. magic number a autentifikace. Struktura LCP paketu viz 2.8. LCP je součástí PPP protokolu, proto jsou jeho zprávy zasílány pomocí PPP paketů označené kódem `0xC021`. [1]



Obrázek 2.8: Struktura PPP LCP paketu

Po ustanovení základních parametrů spojení přes LCP nastává konfigurace IP vrstvy. Nejčastějším pro to používaným protokolem je IPCP, který dovoluje konfigurovat parametry IP pro použití přes protokol PPP. Existují i další protokoly pro konfiguraci spojení, například IPXCP a ATCP, ty ovšem už v současné době nejsou používány. Pro účely konfigurace komunikace přes IP verze 6 vznikl protokol IPv6CP, který se s rozšiřování IPv6 začíná používat.

2.5.2 IPCP

IPCP je protokol pro konfiguraci IP over PPP. Tento protokol nastavuje IP adresy zřízení a další konfigurační hodnoty. IPCP stejně jako LCP komunikuje pomocí PPP rámců se speciálním číslem protokolu `0x8021`. Samotný protokol poté podporuje několik zpráv určených k zasílání nastavení a jejich zamítnutí, potvrzení atd. Typy zpráv jsou rozeznávány podle pole *kód*, angl. *code*. V každém požadavku je zasílán i seznam nastavení, které jsou jím ovlivněna. V závislosti na typu paketu jsou zasílány i jejich hodnoty, které by například zařízení preferovalo.

Pro ustanovení spojení je zařízením zaslán paket typu *Configure-Request*, který informuje o jeho preferovaných nastaveních. Zařízení zaslat preferovanou IP adresu, a nebo má možnost zažádat, aby mu ji druhé zařízení přiřadilo zasláním adresy `0.0.0.0`. Po zaslání této adresy ho to druhé informuje zasláním paketu *Configure-Nak*, který informuje o preferenci adresy zaslané jako součásti tohoto paketu. Původní zařízení opět zažádá o IP adresu, specificky o tu, kterou by to druhé preferoval.

Kód	Jméno
0	Vyhrazený pro výrobce
1	Configure-Request
2	Configure-Ack
3	Configure-Nak
4	Configure-Reject
5	Terminate-Request
6	Terminate-Ack
7	Code-Reject

Tabulka 2.2: Kódy označující typy IPCP paketů

Kromě nastavení IP adresy protokol také podporuje konfiguraci ostatních parametrů. Příkladem může být komprese IP paketů určená hlavně pro pomalejší spojení. Dalším nastavovaným parametrem je DNS server. Pro účely přístupu k internetu je často nutné mít funkční DSN server pro účely zasílání na url namísto IP adresy. Stejně jako konfigurace adresy IP je možné zaslat nevalidní adresu `0.0.0.0` a tím si vyžádat její preferovanou hodnotou. Druhé zařízení následně zašle preferovanou hodnot a to první může zaslat stejný požadavek, který nyní obsahuje validní adresu DNS serveru.

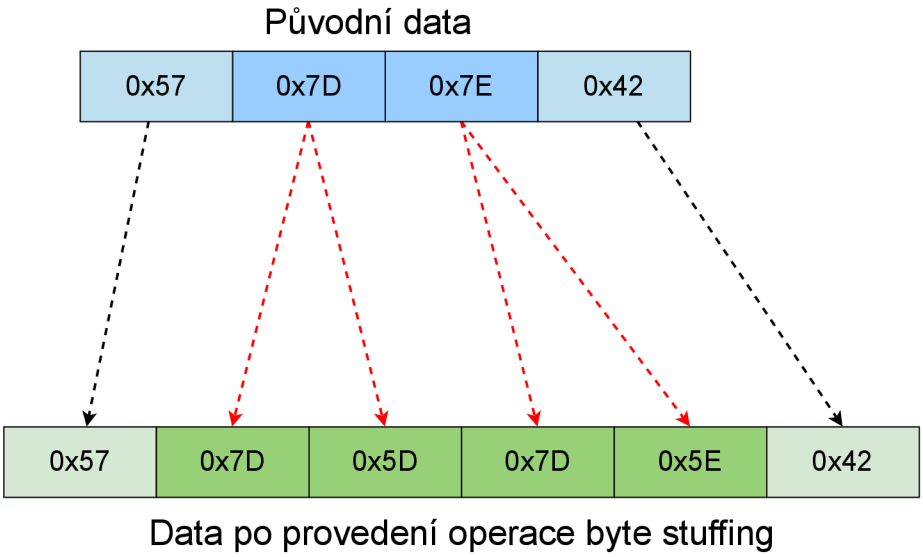
2.5.3 Byte stuffing

PPP rámce na obou koncích obsahují tzv. PPP flag. Je to specifický byte s hodnotou 0x7E odpovídající ASCII znaku , který je určený pro označení začátku a konce rámce. Někdy se může stát, že se tento znak bude vyskytovat v datech, které chceme přes tento protokol posílat. Abychom se vyhnuli chybám při čtení PPP paketů ze sériové linky, tak je nutné tento problém nějak řešit.

Proto se před zasíláním dat PPP paketu používá byte stuffing. v implementaci pro PPP je jakýkoli byte obsahující sekvenci bitů PPP flag nahrazen speciálním escape znakem a za něj je vložen originální byte, na který byla aplikována binární operace XOR s hodnotou 0x20[11]. Příklad tohoto procesu viz obrázek 2.9. Po provedení této operace na každý byte obsažený v PPP paketu ho můžeme zaslat na vzdálené zařízení.

Strana která pakety přijímá musí před zpracováním byte stuffing odstranit, aby se data vrátila do původní formy. Tak jde dosáhnout hledáním všech výskytů escape znaku a jeho odstranění. Na následující znak je použita operace XOR opět s číslem 0x20. Po této sekvenci operací jsou všechna data PPP paketu vrácena do původní formy, se kterou se dá pracovat.

Díky byte stuffingu je možné dosáhnout transparentnosti této vrstvy OSI modelu. To znamená, že technická nemožnost posílat určité bajty je řešena už v samotném protokolu a vyšší vrstvy nemusí tuto problematiku řešit.



Obrázek 2.9: Příklad použití byte stuffingu na sekvenci bajtů

Kapitola 3

Existující řešení

Na trhu existují zařízení[21] a software[13] určené k emulaci modemů. Většina z těchto řešení podporuje základní AT příkazy a některé z nich dovolují uživateli také pracovat s rozšířeními přidanými standardy GSM a některými proprietárními příkazy.

3.1 Hardwarové emulátory

Některé hardware emulátory jsou navrženy jako 1:1 náhrada starších analogových zařízení[21][17]. Jsou tedy určeny pro firmy, které mají velké množství zařízení určených pro komunikaci pouze s Hayes-compatible modemem. Tedy tato zařízení nepodporují žádné modernější protokoly, jako je například TCP/IP. Jelikož jsou analogové sítě vyřazovány a většina míst poskytuje síťovou infrastrukturu, jsou tyto zařízení vhodná náhrada za historicky využívané analogové modemy. Tato zařízení často komunikují přes virtuální tunely vytvořené přes TCP/IP komunikace. Pro komunikaci jsou tedy potřeba dvě zařízení, které uživatel spojí. Po navázání spojení se chovají jako dva Hayes-compatible modemy k sobě připojené. Příkladem využití může být například vzdálené monitorování industriálních zařízení, které původně komunikovalo přes telefonní sítě a po zrušení telefonního spojení by je chtěl majitel modernizovat na ehternetovou síť bez nutnosti nákupu nových zařízení, které tento typ komunikace podporují.

Tento typ emulátorů je pro využití k testování knihoven pro komunikaci s modemem nevhodný z více důvodů. Jedním je cena zařízení a jeho zaměření pouze na podporu pouze starších verzí Hayes command setu. Pokud je uživatel ochotný si pořídit nějaké zařízení emulující modem, může si samotný modem pořídit místo něj. Toto by nebyl takový problém, kdyby emulátor tohoto typu dovoloval emulaci několika existujících modemů z jednoho zařízení. Ovšem tato zařízení často podporují pouze minimální počet příkazů nutných k navázání spojení. To znamená, že by jimi šlo validovat funkčnost komunikace přes sériovou linku a syntax základních AT příkazů, ale nemůžeme od nich čekat podporu standardů nebo způsob jak testovat příkazy různých výrobců.

Další nutnou vlastností emulátoru modemu pro testování knihoven je zjišťování informací o interním stavu. Jelikož je modem implementovaný v hardwaru, uživatel má pouze omezené informace o interní struktuře zařízení. Často neexistuje ani debugovací výstup, který je asi nejdůležitější součástí práce se zařízením, pokud uživatel nemá přístup k implementaci.

3.2 Softwarová řešení

Na rozdíl od hardware emulátorů jsou ty implementované v software vhodnější pro využití jako testovací platformy. Důležitá výhoda je, že není nutné pořizovat zařízení, které implementuje danou funkčnost, což snižuje cenu a dovoluje použít existující osobní počítač jako emulátor.

Software emulátory jsou také často určené pro podporu starších zařízení, které jsou určené pro komunikaci pomocí telefonního modemu. Velká část z nich také posílá komunikaci přes sériovou linku v datovém režimu pomocí TCP/IP streamů. Tyto emulátory se rozdělují ještě do dvou skupin, a to open source a closed source. Closed source emulátory jsou často placené a podléhají podobným nevýhodám jako emulátory hardwarové.

Open source emulátory mají mnoho výhod, které znamenají, že jsou více vhodné pro použití na testování knihoven implementujících AT komunikaci. Přístup k zdrojovému kódu je jednou z jejich nejsilnějších stránek. Dovoluje uživateli podívat se na implementaci a zjistit důvody chování emulátoru. Ovšem mezi jejich nevýhody patří jejich částečně nedostatečná implementace všech AT příkazů podporovaných moderními modemy. Jelikož je jejich cílem emulovat modem z důvodu propojení starších zařízení, často podporují jen základní AT příkazy nutné pro ustanovení spojení a následnou komunikaci. Často neexistuje podpora například PPP protokolu nebo GSM multiplexing. Většina také nepodporuje ani GSM standard určený pro mobilní sítě.

3.2.1 CelerSMS: AT Emulator

Emulátor modemu od firmy CelerSMS je určený jako náhrada pro komunikační software při nedostupnosti toho fyzického nebo jako testovací nástroj, jelikož díky němu není nutné riskovat poškození reálného hardwaru. [3]

Emulátor je open-source, napsaný v jazyce Java. Podporuje dvě rozhraní pro komunikaci, TCP/telnet a console. Console je způsob ovládání emulátoru, který dovoluje uživateli zasílat AT příkazy přes terminálovou aplikaci. Po spuštění emulátoru interpretuje řetězce zaslané do standardního vstupu jako AT příkazy. Druhý režim dovoluje aplikacím připojit se přes port, na který následně zasílají textové AT příkazy. Emulátor nativně nepodporuje komunikaci se zařízeními připojenými přes sériové porty. Autoři doporučují tento problém řešit pomocí virtuálního sériového portu a programu, který bude komunikaci přes něj přeposílat na port emulátor přes TCP.

Emulátor na rozdíl od těch předešlých podporuje základní AT příkazy a GSM rozšíření pro SMS. Jeho nevýhodou je ovšem špatná rozšiřitelnost. Ze struktury programu lze odhadnout, že by bylo těžké přidat různé chování pro specifické modemy nebo příkazy nové. Také vyniká naprostý nedostatek logování a možnosti komunikaci se zařízením debugovat. Modem je vhodný pro otestování základní funkčnosti, ale nedovoluje zjistit podporu speciálních příkazů přidávaných výrobci nebo chování při vzniku chyb.

3.2.2 TCPSER

TCPSER je implementace modemu v programovacím jazyce C. Emulátor změní jeden ze sériových portů PC na modem podporující komunikaci přes AT příkazy. Jako hardware emulátory je tato funkčnost určena hlavně z důvodu podpory starších zařízení, jejichž komunikaci následně posílá přes TCP/IP dalšímu modemu, který běží na vzdáleném počítači. Na rozdíl od předešlého emulátoru nepodporuje rozšířené příkazy standardu GSM.

Výhodou tohoto softwaru je že dovoluje široké možnosti konfigurace, jako je konfigurace různých způsobů chování podle nutnosti uživatele. Další důležitou funkčností je podpora trasování a logování. Emulátor podporuje několik režimů trasování přijímaných a zasílaných dat, také podporuje záznam IP paketů které posílá. Logování je konfigurovatelné na několik úrovní, což je vhodné pro testování a modem dovoluje zapisovat informace do logovacího souboru nebo terminálu.

Nevýhodou tohoto emulátoru je to, že nedovoluje jednoduchou rozšiřitelnost a tedy by bylo těžké přidat příkazy od specifických výrobců. Také implementuje pouze část příkazů nutných pro komunikaci se vzdáleným zařízením. Podle manuálu speciální příkazy rozezná a tedy nevznikne chyba, pokud je emulátoru pošleme, ale velkou část z nich neimplementuje. Na rozdíl od mnoha dalších emulátorů podporuje velké množství logovacích možností, proto je alespoň částečně vhodný pro použití jako testovací platforma pro vývoj.

Kapitola 4

Návrh a architektura emulátoru

Návrh emulátoru důraz na podporu jednoduché rozšiřitelnosti o další typy modemů, bez nutnosti měnit interní strukturu emulátoru. Díky asynchronnímu chování AT příkazů zasílaných GSM modemy, a GSM multiplexingu, je kladen důraz na architekturu emulátoru týkající se běhu interních procesů a jejich komunikace.

4.1 BG96

Cílem práce je dovolovat minimálně emulaci existujícího modemu Bg96 od firmy Quectel, neboť je tento modem často přítomen v řadě IoT zařízení. Ovšem dalším požadavkem je dovolovat jednoduchou rozšiřitelnost pro ostatní modemy od různých firem. Všichni výrobci podporují AT příkazy z originální specifikace a specifikace týkající se mobilních modemů. Ovšem výrobci často přidávají své vlastní AT příkazy, např. různé režimy spánku.

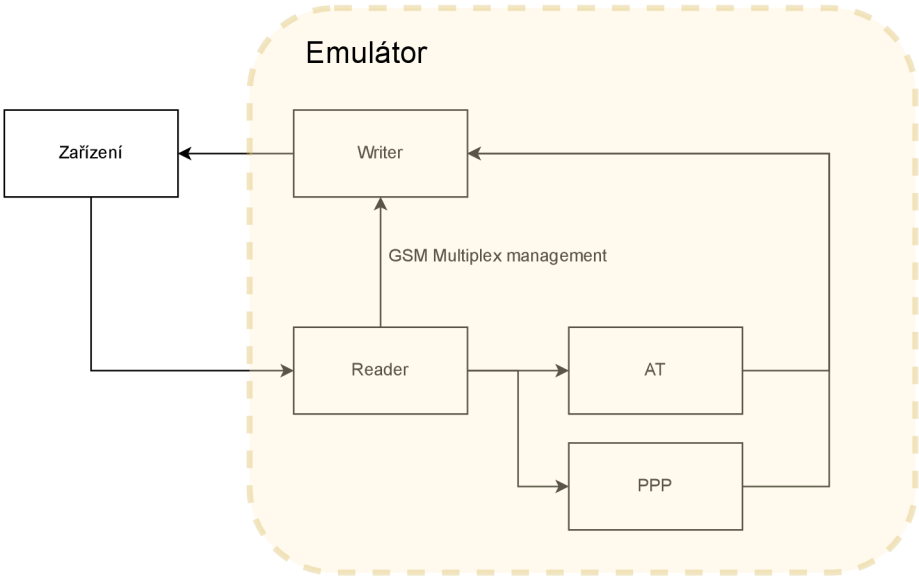
Modem BG96 od firmy Quectel je jeden z populárních LTE modemů na trhu. Jedná se o tzv. IoT modem, výrobce se tedy zaměřuje hlavně na nízkou spotřebu a malou velikost na úkor rychlosti, která je omezená na 375 kbps oběma směry.

Toto zařízení podporuje několik různých technologií. v základu se drží specifikace a podporuje CMUX, PPPoS a rozšířené AT příkazy pro volání, SMS a přístup k internetu. Modem také podporuje velké množství komunikačních rozhraní: USB/UART/I2C[18]. Tato práce je zaměřena na podporu komunikace přes sériovou linku.

4.2 Sériová komunikace

Díky rozdělení emulátoru na skupinu podprocesů které spolu komunikují asynchronně je nutné, aby byla komunikace se zařízením přes sériovou linku nějakým způsobem organizována, viz 4.1. Kdyby nebyla takto synchronizována, tak se může stát, že se dva moduly pokusí zapisovat do sériové linky naráz. O tuto režii se stará hlavní modul, který je rozdělen na dva nezávislé procesy.

Přesněji se tyto procesy starají o synchronizované čtení a zápis do sériové linky. Při čtení je vybrán správný modul a zpráva je mu zaslána přes asynchronní kanály. Se zprávou je mu také zaslán identifikátor, kterým se dají ke zprávám přiřadit odpovědi. Tento přístup je zvolen z důvodu podpory GSM multiplexingu, který dovoluje mít několik otevřených komunikačních kanálů naráz, z nichž má každý unikátní identifikátor. Proces určený pro zápis naslouchá na několika asynchronních kanálech naráz a zpracovává příchozí zprávy. Buďto jsou to interní zprávy pro komunikaci mezi moduly, nebo se jedná o data, která by



Obrázek 4.1: Komunikace mezi interními procesy emulátoru

měla být zapsána na sériovou linku. S daty vždy přichází informace o tom, na který kanál GSM multiplexeru by měla být zaslána, pokud je emulátor v tomto režimu.

4.3 Architektura

Jednoduché změně chování a možnosti přidat nové modemy, které podporují jiné části funkcionality, bylo dosaženo rozdělením emulátoru do několika modulů, které mezi sebou navzájem komunikují. Asi nejdůležitějším je hlavní modul, který se stará základní funkcionalitu, kterou mají modemy podobnou. Čtení a zápis do sériové linky, práce s GSM multiplexingem a hlavně řízení komunikace mezi moduly.

Hlavní modul má přístup ke dvěma dalším, se kterými může koordinovat funkčnost. Jedná se o moduly implementující funkčnost spojenou s AT příkazy a komunikací přes protokol PPP. Hlavní modul je navržen tak, aby nezávisel na specifické implementaci modulu, ale na jeho rozhraní. Toto dovoluje přidávat implementace modulů, a tím měnit chování emulátoru. Díky tomu je možné přidávat nové implementace modulů, které mají chování jako specifický typ modemu.

Pro jednodušší implementaci nových AT příkazů byl modul podporující tuto funkčnost navržen tak, aby se daly vytvářet nové moduly podporující specifickou funkčnost pomocí jednoduché dědičnosti. Existuje základní třída, která je navržena na podporu základní funkcionality. Následně z ní dědí ostatní a samotné implementují AT příkazy. Tímto způsobem jde dosáhnout rozdělení druhů funkcionality na specifické moduly a jednoduché přidávání nových modemů, je pouze nutné implementovat jejich specifické příkazy.

Druhý modul je navržen pro podporu komunikace přes PPP. Podporuje ustanovení PPP spojení přes protokol LCP a nastavení přes IPCP. Následně podporuje komunikaci pomocí protokolu IP, která je zpracována pomocí zasílání IP paketů na určenou adresu pomocí internetového připojení PC, na kterém emulátor běží.

4.4 Asynchronní komunikace mezi moduly

Díky složitosti komunikace s emulátorem modemu, hlavně díky GSM multiplexing, který dovoluje zařízení zasílat více zpráv obsahujících různé protokoly, bylo rozhodnuto komunikaci mezi moduly řešit pomocí komunikačních kanálů. Kanály fungují jako fronty zpráv, které dovolují modulům zasílat a přijímat z několika různých zdrojů. Každý modul je spuštěn jako nezávislý proces, jehož jediný způsob komunikace jsou kanály pro příjem a zasílání dat. Hlavní modul se následně stará o zpracování dat přicházejících z těchto asynchronních kanálů. Tento přístup dovoluje modulům nezávisle na sobě posílat data a příkazy a umožňuje jim provádět akce nezávisle na stavu komunikace se zařízením. Zasílání dat pomocí asynchronních komunikačních kanálů má také další výhodu, a to, že na rozdíl od komunikace pomocí sdílené paměti nevznikají situace jako je vzájemné čekání, apod[15].

Kapitola 5

Implementace

Pro implementaci emulátoru byl zvolen jazyk Python díky jeho dobré podpoře pro více platforem. Jazyk Python má také dobrou podporu pro asynchronní kód. Pro práci s touto funkcionalitou a také díky podpoře velkého množství asynchronních metod byla použita knihovna `asyncio`.

5.1 Sériová komunikace

Ke komunikaci se zařízením byl použit modul `pyserial-asyncio`. Jedná se o jednoduchou knihovnu, která rozšiřuje modul `pyserial` o podporu asynchronních funkcí. o čtení dat ze sériové linky se stará funkce `run_reader()`, která má odlišné chování podle stavu emulátoru. Podporuje tři stavy, které mění způsob načítání dat ze sériové linky. AT režim, ve kterém se modem spustí, je určený k načítání AT příkazů rozdělených znakem `<CR>`. PPP režim mění oddělovací znak na PPP flag a také počítá s tím, že rámce PPP mají tento oddělovací znak z obou stran. Posledním režimem je ten podporující GSM multiplexing, který načítá CMUX rámce podobně jako předešlé režimy. Na rozdíl od nich se také stará o jejich zpracování a, pokud se jedná o rámce UIH, jejich zaslání správnému modulu. k těmto datům je přiřazen identifikátor, který je určený k rozpoznání odpovědi v režimu GSM multiplexingu. v tomto režimu je jako identifikátor použito DLCI příchozího rámce, ale když modem není v tomto režimu, je použito číslo -1. Protože je identifikátor určen pouze pro režim s GSM multiplexingem, tak je při zápisu v jiných režimech ignorován.

Jelikož je možné, že data z více míst v kódu mohou být zasílána zároveň, musí se funkce `run_writer()` starat o čtení z více komunikačních kanálů naráz bez blokování těch ostatních. Toho je dosaženo použitím funkce `asyncio.wait`, která dovoluje čekat na více asynchronních funkcí zároveň, přičemž vrátí výsledek té, která byla první dokončená. Metoda pro zápis na sériovou linku se také stará o příjem speciálních příkazů, které můžou moduly zasílat za účelem změny parametrů modemu, o které se stará třída `Modem`. Důležitou součástí funkcionality této metody je také schopnost zabalovat příchozí zprávy od modulů do paketů CMXU a jejich zasílání na sériovou linku.

5.2 Rozhraní

Emulátor před spuštěním dovoluje nastavit různé hodnoty a připravit metody pro testování. Při samotném běhu podporuje mnoho způsobů, jak informovat uživatele o svém interním stavu a o tom, jaká data jsou zasílána přes sériovou linku.

5.2.1 Konfigurace

Jedním z důvodů, proč byl emulátor implementován jako knihovna jazyku Python, je jednoduchost jeho konfigurace před spuštěním. Pro jednoduché případy by stačily parametry příkazové řádky nebo konfigurační soubory. Ovšem pro pokročilejší použití a jednoduchou implementaci složitějších testovacích scénářů by tento přístup nebyl vhodný. Použití jazyka Python jako konfigurační nástroj uživateli dovoluje implementovat chování, které nemusí emulátor v základu podporovat. Knihovna, kromě specifických metod pro testování, dovoluje uživateli také používat svůj vlastní kód, který je při běhu spuštěn. Tato architektura dovoluje přidat velmi specifickou funkčnost, což může být například vypsání interního stavu modemu a podobně.

```
from modems.bg96 import Bg96

async def main():
    port = argv[1]
    baud = 115200
    modem = await Bg96(port, baud)
    await modem.run()
```

Výpis 5.1: Import a nastavení emulátoru modemu BG96

5.2.2 Běh emulátoru

Emulátor v základu podporuje několik způsobů informování uživatele o svém stavu a o datech zasílaných pomocí sériové linky. Nejjednodušším z nich je výpis dat na chybový výstup `STDERR`. Pro tuto funkcionalitu byla zvolena knihovna `loguru`, která, kromě barev a formátování, podporuje přidání informací o čase udání události. Tato informace dovoluje uživateli porovnávat výstup ze zařízení s informacemi z modemu a tím porovnat události, které se na obou stranách sériové linky udávaly.

Stav komunikace se zařízením je asi nejdůležitější částí informací zapisovaných do terminálový výstup. Každý přijatý nebo zaslaný AT příkaz je se svým směrem vyznačeným vypsán z důvodu informování uživatele o stavu komunikace, viz 5.1. Další data vypisovaná do terminálu jsou ty o stavu komunikace přes GSM multiplexing. Všechny rámce zaslané a přijaté od zařízení jsou po parsování vypsány pro debugovací účely. Posledními informacemi vypisovanými do terminálového výstupu jsou informace o změnách interního stavu modemu. Jedná se o například přepnutí do PPP nebo GSM multiplex režimu. Uživatel je také informován o chybách při zpracování AT příkazů, což také zahrnuje to, že emulátor přijme nepodporovaný příkaz.

```
15:28:24.130 | INFO     | Device: AT
15:28:24.132 | INFO     | Modem:  OK
15:28:24.192 | INFO     | Device: ATE0
15:28:24.194 | INFO     | Modem:  OK
15:28:24.285 | INFO     | Device: AT+CGMM
15:28:24.287 | INFO     | Modem:  BG96
15:28:24.287 | INFO     | Modem:  OK
15:28:24.486 | INFO     | Device: AT+CPIN?
15:28:24.488 | INFO     | Modem:  READY
15:28:24.489 | INFO     | Modem:  OK
15:28:24.687 | INFO     | Device: AT+CGSN
15:28:24.689 | INFO     | Modem:  00100000000123
15:28:24.689 | INFO     | Modem:  OK
15:28:24.889 | INFO     | Device: AT+CIMI
15:28:24.890 | INFO     | Modem:  460023210226023
15:28:24.891 | INFO     | Modem:  OK
15:28:25.092 | INFO     | Device: AT+COPS?
15:28:25.093 | INFO     | Modem:  +COPS: 0,0,"T-Mobile CZ T-Mobile CZ",7
15:28:25.094 | INFO     | Modem:  OK
```

Obrázek 5.1: AT příkazy zobrazené v terminálu

Informace o průběhu komunikace přes PPP protokol nejsou zasílány pomocí výstupu na terminál. Tak je z důvodu toho, že PPP je protokol binární a data jím zasílaná jsou často také v binárním formátu. Proto byl zvolen nástroj Wireshark, který podporuje zobrazení těchto paketů. Při běhu emulátoru jsou příchozí i odchozí pakety zapisovány do interního pole a po jeho ukončení jsou uživateli všechny zobrazeny. Data zapisována pro zobrazení se týkají všech paketů zaslaných přes sériovou linku. Tedy Wiresharkem bude zobrazen i začátek PPP komunikace obsahující LCP a IPCP handshake. Také budou zobrazeny všechny zaslané pakety, nejen ty, které modem přepošle dále do internetu.

Při testování je také dobré vědět stav zařízení, které s modemem komunikuje. Proto je implementována jednoduchá funkčnost, která dokáže zapsat výstup zařízení připojených přes sériovou linku. Často se jedná o debugovací výstup knihovny implementující komunikaci přes AT příkazy, ale také dovoluje uživateli přidat kód který vypisuje informace o interním stavu zařízení.

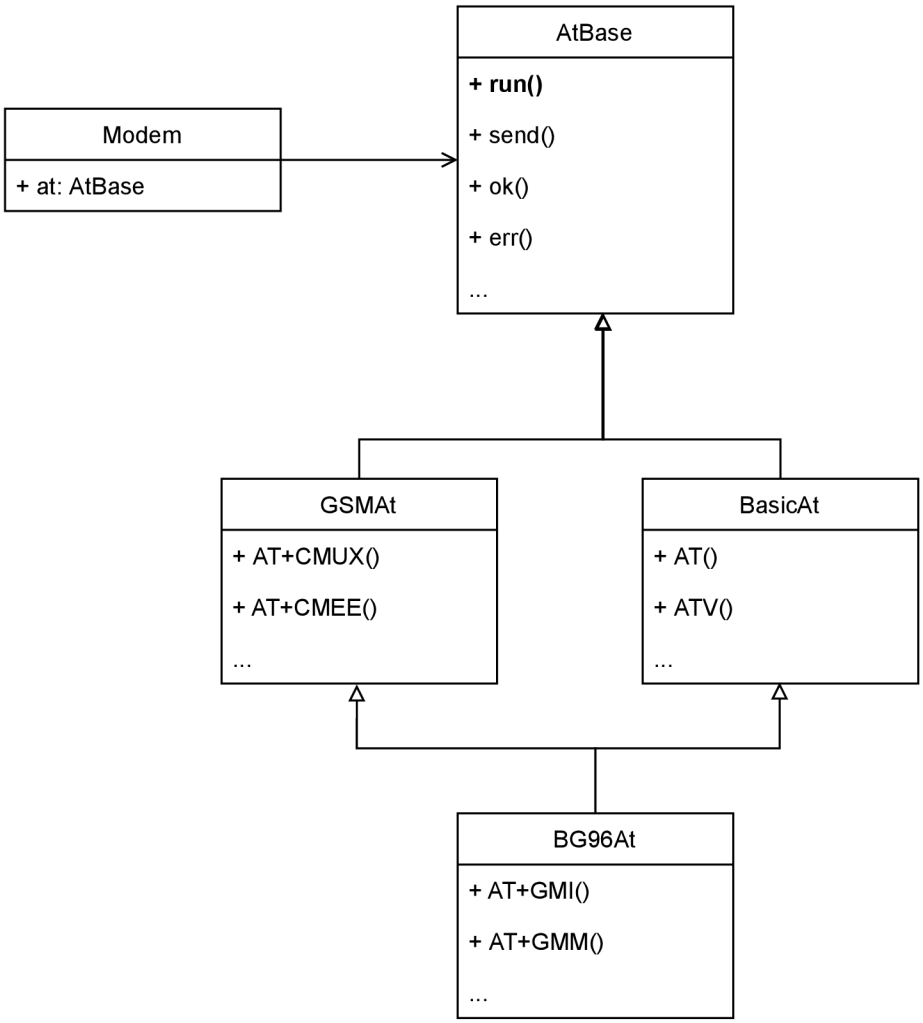
5.3 Moduly

Pro jednoduchou rozšiřitelnost byl modem rozdělen na dva moduly, které je možné nahradit pro jednoduchou změnu chování modemu. Jedná se o moduly AT a PPP. Jelikož je nutné modem vytvořit ze svých podmodulů, které mají nutnost se také inicializovat, byla implementována speciální funkce pro vytvoření objektu implementující funkcionalitu Bg96 se kterým může uživatel pracovat.

5.3.1 AT

Jelikož různé modemy implementují jiné skupiny AT příkazů a často podporují i své vlastní, které žádný jiný modem nebo výrobce nepodporují, byl modul implementující komunikaci přes AT rozdělen na několik podmodulů, ze kterých finální implementace dědí, viz 5.2. Nejdůležitějším modulem, který implementuje proces čtení a zpracování AT příkazů je třída `AtBase`. Obsahuje funkce pro formátování AT příkazů, testování apod.

Při vytváření objektu modemu by měla být vybrána jedna z tříd jako ta, která bude používána pro implementaci všech podporovaných AT příkazů. Tyto příkazy jí budou posílány přes asynchronní kanály. Pokud tato třída neobsahuje jejich implementaci, tak je základním chováním oznámit uživateli do terminálu chybu s informacemi o příkazu a zařízení zaslat chybový kód.



Obrázek 5.2: Závislosti mezi třídami implementující AT příkazy

Individuální AT příkazy

AT příkazy, například `AT+CGMI`, jsou implementované ve třídách rozdělených podle funkčnosti dědící z `AtBase`. Pro vytvoření celé funkčnosti modemu se vzájemným děděním na sebe "nabaluje" funkčnost. Příkladem může být třída `Bg96At`, která dědí základní AT příkazy a k nim přidává příkazy pro modem BG96 firmy Quectel. Tímto způsobem jsou také implementovány příkazy, které mají jiné implementace závislé na emulovaném modemu, jako jsou například příkazy na získání IMEI nebo jména výrobce.

Individuální AT příkazy jsou implementovány pomocí asynchronních metod. Každá metoda implementující jeden příkaz se stará o jeho zpracování a zaslání odpovědi nebo chyb.

Jelikož jsou metody obsaženy v třídě dědící `AtBase`, mají přístup k internímu stavu tohoto modulu a mohou pomocí pomocných funkcí zasílat odpovědi, měnit konfiguraci apod.

```
@at_write("AT+CMEE")
async def cmee_w(self, params):
    if len(params) != 1:
        await self.ex_err(518, "CMEE")
        return
    val = int(params[0])
    if val < 0 or val > 2:
        await self.ex_err(518, "CMEE")
        return

    self.cmee_val = val
```

Výpis 5.2: Implementace příkazu write AT+CMEE

Na každou metodu, která implementuje jeden z AT příkazů, musí být použit dekorátor specifikující informace o AT příkazu, pro který je určena. Dynamická povaha jazyku Python dovoluje jednoduše upravovat existující objekty, proto je pomocí dekorátoru přiřazen funkci samotný AT příkaz, který bude implementovat. Přesněji je to string, se kterým jsou příchozí AT příkazy jednoduše porovnávány pomocí ekvality řetězců. Příkazy, které mají parametry, a tedy je nutné je parsovat složitěji, používají speciální dekorátor proto určený. Parsované parametry jsou následně předány metodě jako seznam řetězců ve stejném pořadí parametrů příkazu.

Při vytváření objektu AT modulu je volána metoda `__init__` třídy `AtBase`, která implementuje kolekci metod označených pomocí speciálních dekorátorů. Kód v této funkci iteruje všechny metody implementující AT příkazy z výsledné třídy. Následně jsou tyto metody rozděleny do dvou polí podle použitého dekorátoru. Při běhu modemu jsou tyto pole postupně procházeny a první metoda odpovídající příchozímu příkazu je spuštěna. Pokud se jedná o příkaz s parametrem, je výsledek parsování předán metodě.

5.3.2 PPP

PPP modul implementuje základní funkčnost LCP a IPCP handshake. Pro parsování a vytváření paketů je využita knihovna `scapy`. Implementována byla podpora fcs a byte stuffing, která není součástí knihovny `scapy`.

Hned po spuštění modem čeká na první PPP paket zaslaný ze strany zařízení, což by pro ustanovení spojení měl být paket protokolu LCP, přesněji typu *configure*. Konfigurace různých hodnot zaslaná tímto paketem je uložena do paměti, následně je zpět zasláno potvrzení tohoto nastavení. Stejné kombinace operací je provedena ze strany emulátoru. Po ustanovení spojení přes LCP je konfigurován protokol IPCP. Důležitou součástí tohoto protokolu je správné nastavení IP adres obou zařízení a DNS serverů pro zasílání požadavků do internetu. Emulátor podporuje automatické nastavení IP zařízení, pokud si o to zažádá.

Po konfiguraci a ustanovení spojení přes LCP a IPCP modem čeká na zařízení aby zaslalo IP pakety obsahující odchozí data. Tyto pakety jsou následně parsovány a extrahovány knihovnou `scapy` z PPP rámců a následně jsou zasílány na cílovou IP adresu. Při parsování je nutné, aby emulátor měnil některá pole IP paketů, aby přicházely odpovědi na správnou adresu. Toto chování můžeme přirovnat k chování routeru, i když je mnohem jednodušší např. díky komunikaci pouze s jedním zařízením přes PPP protokol. u paketů tedy musí

být měněna IP adresa odesílatele a při přijímání musí být změněna zpět. Stejně tak kód parsuje TCP a UDP informace a mění port paketu. Pro jejich zasílání jsou využity funkce knihovny `scapy` pro zasílání paketů a příjem odpovědí na ně.

5.4 Využití knihovny pro potřeby testování

Testování s použitím emulátoru modemu probíhá pomocí testovacích scénářů implementovaných v jednotlivých souborech jazyka Python. Uživatel má při vytváření a konfiguraci modemu možnost nastavit různé testovací chování pomocí metod, jak na samotném objektu modemu, tak na jeho AT modulu. Metody určené pro testování dovolují také do modemu vložit různé funkce implementující chování dané uživatelem. Po zavolání metody `run()` se emulátor spustí s danou konfigurací a uživatelem přidanými funkcemi.

```
from loguru import logger
from modems.bg96 import Bg96

async def main():
    async def t(modem):
        logger.info(modem.state)

    modem = await Bg96(argv[1])
    modem.test_run_periodic(t, 0.5)
    await modem.run()
```

Výpis 5.3: Uživatelem poskytnutá funkce periodicky vypisující stav modemu

5.4.1 Časově závislé události

Události závislé na čase jsou rozdělené na dvě kategorie. Události, které se provedou jednou a ty, které se provádí za sebou až po ukončení modemu. Spouštění událostí pro různé časy je dosaženo pomocí vytvoření nového `asyncio.Task`, která se stará o běh asynchronní metody na event loopu. Funkčnosti implementující opakované volání metod je dosaženo pomocí `while` loopu a funkce `asyncio.sleep`. Všechny funkce, které jsou takto modemu přiřazeny, jsou uloženy do speciálního pole a pokud dojde k dokončení běhu modemu dříve než funkce doběhnou, je jejich exekuce předčasně přerušena. Pro často používané scénáře byly implementovány funkce, které na určitý čas dají modem do chybového režimu, kdy nezpracovává příkazy a odpovídá chybovými hláškamy. a funkce, která na nějaký čas přidá zpoždění zpracování příchozích dat, pro simulaci zatížení na danou dobu.

5.4.2 Události závislé na stavu

Pro implementaci některých testovacích scénářů je vhodné uživateli dovolit přidat možnost provést nějakou funkčnost závislou na interním stavu emulátoru modemu. Může se například jednat o příchozí AT příkaz, jehož funkčnost uživatele debuguje a chtěl by se dozvědět o interním stavu modemu. Stejně jako pro časové události byly implementovány nejčastější využití této funkčnosti. Pro AT příkazy je možné přidat čas před zpracování příkazu a pro vypsání chyby namísto normálního zpracování.

Jelikož AT příkazy nejsou jedinou funkčností modemu, byla třídou `Modem` implementována funkčnost pro přidání zpoždění před čtením nebo zápisem dat na sériovou linku. Stejně

jako u předešlých testovacích metod je tato funkčnost implementována takovým způsobem, že je možné, aby uživatel poskytnul vlastní funkci implementující specifičtější funkcionalitu při zápisu nebo čtení. Těmto funkcím jsou v parametru poskytnuta data ze sériové linky, což uživateli dovoluje implementovat různé chování v závislosti například na typu přicházejících dat.

5.4.3 Funkčnost URC

Jak již bylo zmíněno v kapitole 2.3.5, URC je typ zpráv zasílaných modemem v závislosti na určitých událostech. Jelikož emulátor modemu nemá přístup k SIM kartě nebo mobilní síti, je nutné tyto události simulovat. Toho je dosaženo rozšířením funkcionality z předešlých kapitol o testování. Modemem jsou implementovány funkce, které zasílají stavové kódy v závislosti buďto na interním stavu modemu nebo v různých časových intervalech. Funkčnost modemu implementující možnost vypínat tento typ stavových kódů je modemem ignorována z důvodu zjednodušení chování při běžících testech. Uživatel si může být jistý, že zařízení neních schopné tento typ zpráv vypnout a tím omezit rozmezí testování funkčnosti.

```
from modems.bg96 import Bg96

async def main():
    port = argv[1]
    baud = 115200
    modem = await Bg96(port, baud)

    modem.test_add_urc_once("+CPIN: READY", 1)
    await modem.run()
```

Výpis 5.4: Nastavení modemu na zaslání URC po 1 sekundě běhu

5.5 Demonstrační aplikace

Součástí práce je také vytvořit tzv. demonstrační aplikaci, která dovoluje ukázat způsob využití emulátoru. Aplikace, implementována v souboru `demo.py`, ukazuje import a použití na několika testovacích scénářích. Dovoluje uživateli jeden z těchto testovacích scénářů vybrat a také zvolit, na kterém portu bude emulátor naslouchat. Oba tyto parametry jsou nastavitelné pomocí parametrů příkazové řádky. Testovací scénáře implementují různé chování modemu v závislosti na jeho nastavení, např. chyby při přijetí specifického AT příkazu, zpoždění zpracování příchozích dat apod. Po přečtení zdrojového kódu této aplikace a jejím krátkém používání by měl mít uživatel částečný přehled o tom, jak s emulátorem pracovat a jak psát vlastní scénáře pro testování.

Kapitola 6

Ověření korektní funkcionality emulátoru

Při vývoji byl využit mikrokontroler ESP32 firmy Espressif pro ověření funkcionality emulátoru. Byly vybrány různé programy používající Hayes Command Set pro komunikaci s modemem, aby se zjistila správnost implementace. Pro použití byla zvolena deska s procesorem firmy DOIT ESP32 Devkit V1.

6.1 ESP32

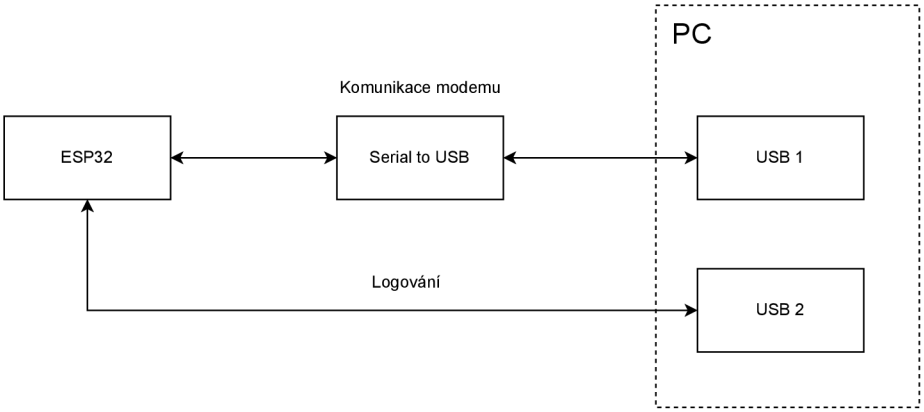
ESP32 je populární skupina mikroprocesorů firmy Espressif vyznačující se podporou WiFi a Bluetooth. Kromě těchto vlastností se platforma vyznačuje také velmi nízkou cenou, proto je jako jedním z jejich využití IoT[9]. Také z důvodu nízké ceny se tato platforma stala populární pro hoby účely. Proto je jednoduché sehnat tzv. vývojové desky, které dovolují používat mikroprocesor bez znalostí elektrotechniky nutných pro vytvoření vlastní tištěné desky propojující mikroprocesor a zařízení se kterými komunikuje.

ESP32 podporuje několik frameworků pro vývoj, které mohou programy běžící na mikroprocesory využívat. Oficiální vývojové prostředí je Espressif IoT Development Framework (esp-idf), které dovoluje vytvářet aplikace v C/C++. Pro vývoj existují i další neoficiální vývojové frameworky, jako například porty platforem Arduino, Micropythonu a Mongoose OS.

6.2 Sériová komunikace

Pro komunikaci s deskou ESP32 Devkit jsou používány dvě sériové linky, které musí být pro komunikaci s počítačem převedeny na USB, viz 6.1. Ta první, která má svůj integrovaný převodník na desce ESP32 Devkit, je určena k nahrávání programů a následně výpisu debugovacích informací při jejich běhu. Mnoho knihoven implementuje různé úrovně logování pro tuto linku a uživatel tuto funkčnost může využít k zjištění stavu běžícího programu na desce ESP32.

Druhá sériová linka je určena k samotné komunikace zařízení a modemu přes AT příkazy a další protokoly. Jelikož deska ESP32 Devkit neobsahuje dva převodníky ze sériové linky na USB, byl při vývoji použit převodník USB to Serial od firmy Arduino, ale pro práci s modemem je možné použít jakýkoli převodník podporující dostatečné rychlosti.



Obrázek 6.1: Sériová komunikace s ESP32

6.3 Ověření komunikace s emulátorem

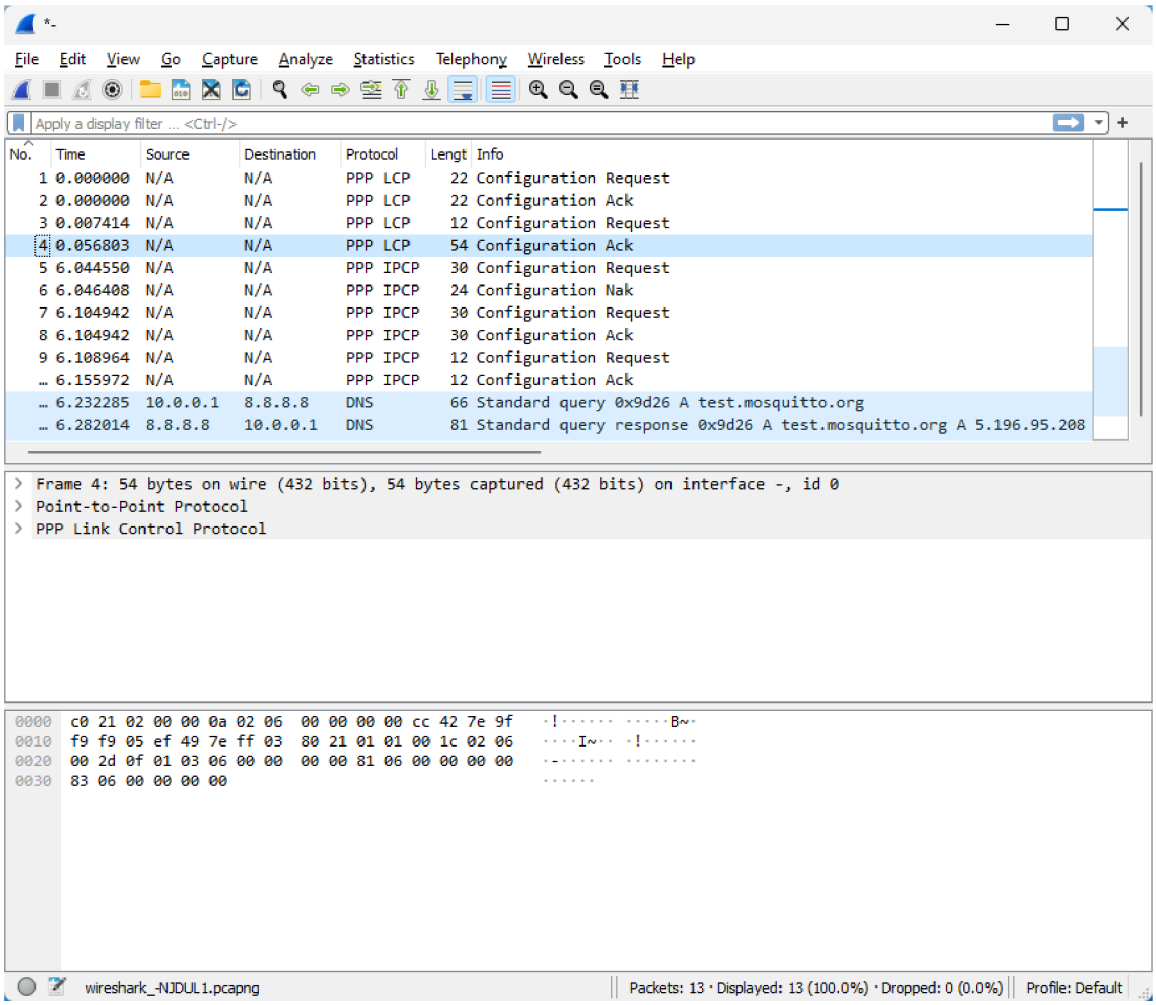
Pro ověření funkčnosti emulátoru byl zvolen framework esp-idf. Takto bylo rozhodnuto z důvodu jeho oficiální podpory výrobcem a poskytnutí kvalitních příkladů použití pro komunikaci přes AT příkazy. Framework obsahuje podporu části AT příkazů, PPP komunikace a GSM multiplexingu. Pro ověření funkčnosti byly vybrány příklady z oficiálních repozitářů a jiných zdrojů na internetu.

6.3.1 Inicializace PPP komunikace

Jako první scénář pro ověření funkcionality byl zvolen příklad použití z oficiálních repozitářů pro esp-idf, "PPP over Serial (PPPoS) client example". Jak z názvu vyplívá, tento program se pokusí o navázání spojení přes protokol PPP s modemem. Následně po jeho navázání se pokusí kontaktovat MQTT server.

Tímto příkladem se otestuje několik částí funkcionality emulátoru. Prozačátek to je správná komunikace přes sériovou linku a podpora některých AT příkazů. Specificky je nutné aby pro správnou funkčnost emulátor podporoval minimálně základní příkazy, které program posílá pro inicializaci a příkazy pro navázání spojení přes PPP.

Další část emulátoru, která je ověřována je podpora protokolu PPP a to jak protokoly LCP pro inicializaci, tak IPCP handshake a samotná komunikace přes tento protokol. Inicializace a začátek komunikace přes protokol IP je zobrazena nástrojem Wireshark na obrázku 6.2. Je také ověřeno, jestli emulátor správně pracuje s příchozími pakety ze strany zařízení, které je při přijetí nutné upravit před přeposláním přes protokol IP.



Obrázek 6.2: Záznam navázání spojení zobrazený pomocí nástroje Wireshark

6.3.2 PPP komunikace přes GSM multiplexor

Pro ověření podpory komunikace s podporou GSM multiplexingu byla zvolena upravená verze příkladu z předchozí sekce. Úpravy tohoto programu přidávají podporu pro protokol GSM multiplexingu a komunikaci přes něj pomocí PPP. Po spuštění se tento příklad pokusí registrovat na modemu dva kanály pro komunikaci. Jeden z nich určený pro zasílání AT příkazů a druhý, který je využíván pro komunikaci přes PPP s MQTT serverem. Jelikož se jedná o upravený předchozí příklad, tak komunikace probíhá stejným způsobem. Jediným rozdílem je, že je tak dosaženo přes GSM multiplexing rámce obsahující PPP pakety.

```
Device: AT+CMUX=0
Modem:  OK
ModemStateCommand(modem_state=<ModemState.CMUX: 3>)
Setting modem state to CMUX
Device: CmuxPacket {'frame_type': <CmuxFrameType.SABM: 0>, 'CR': <CommandResponseType.COMMAND: 1>, 'DLCI': 0, 'information': b'', 'PF': True}
Setup DLCI = 0
Device: CmuxPacket {'frame_type': <CmuxFrameType.SABM: 0>, 'CR': <CommandResponseType.COMMAND: 1>, 'DLCI': 1, 'information': b'', 'PF': True}
Setup DLCI = 1
Device: CmuxPacket {'frame_type': <CmuxFrameType.SABM: 0>, 'CR': <CommandResponseType.COMMAND: 1>, 'DLCI': 2, 'information': b'', 'PF': True}
Setup DLCI = 2
Device: CmuxPacket {'frame_type': <CmuxFrameType.UIH: 4>, 'CR': <CommandResponseType.RESPONSE: 0>, 'DLCI': 2, 'information': b'AT+IFC=0,0', '
Device: AT+IFC=0,0
Modem:  OK
Modem:  CmuxPacket {'frame_type': <CmuxFrameType.UIH: 4>, 'CR': <CommandResponseType.RESPONSE: 0>, 'DLCI': 2, 'information': b'\r\nOK\r\n', '
Device: CmuxPacket {'frame_type': <CmuxFrameType.UIH: 4>, 'CR': <CommandResponseType.RESPONSE: 0>, 'DLCI': 2, 'information': b'AT&W', 'PF': F
Device: AT&W
Modem:  OK
Modem:  CmuxPacket {'frame_type': <CmuxFrameType.UIH: 4>, 'CR': <CommandResponseType.RESPONSE: 0>, 'DLCI': 2, 'information': b'\r\nOK\r\n', '
Device: CmuxPacket {'frame_type': <CmuxFrameType.UIH: 4>, 'CR': <CommandResponseType.RESPONSE: 0>, 'DLCI': 2, 'information': b'AT+CSQ', 'PF':
Device: AT+CSQ
Modem:  +CSQ: 24,99
Modem:  OK
Modem:  CmuxPacket {'frame_type': <CmuxFrameType.UIH: 4>, 'CR': <CommandResponseType.RESPONSE: 0>, 'DLCI': 2, 'information': b'\r\n+CSQ: 24,9
Device: CmuxPacket {'frame_type': <CmuxFrameType.UIH: 4>, 'CR': <CommandResponseType.RESPONSE: 0>, 'DLCI': 2, 'information': b'AT+CBC', 'PF':
Device: AT+CBC
Modem:  +CBC: 0,50
Modem:  OK
Modem:  CmuxPacket {'frame_type': <CmuxFrameType.UIH: 4>, 'CR': <CommandResponseType.RESPONSE: 0>, 'DLCI': 2, 'information': b'\r\n+CBC: 0,50
Device: CmuxPacket {'frame_type': <CmuxFrameType.UIH: 4>, 'CR': <CommandResponseType.RESPONSE: 0>, 'DLCI': 2, 'information': b'AT+CGDCONT=1,"
Device: AT+CGDCONT=1,"IP","CMNET"
Modem:  OK
Modem:  CmuxPacket {'frame_type': <CmuxFrameType.UIH: 4>, 'CR': <CommandResponseType.RESPONSE: 0>, 'DLCI': 2, 'information': b'\r\nOK\r\n', '
Device: CmuxPacket {'frame_type': <CmuxFrameType.UIH: 4>, 'CR': <CommandResponseType.RESPONSE: 0>, 'DLCI': 1, 'information': b'ATD*99***1#',
Device: ATD*99***1#
Modem:  CONNECT 115200
ChannelStateCommand(channel_state=<ChannelState.PPP: 2>)
Setting dlci channel 1 state to PPP
Modem:  CmuxPacket {'frame_type': <CmuxFrameType.UIH: 4>, 'CR': <CommandResponseType.RESPONSE: 0>, 'DLCI': 1, 'information': b'\r\nCONNECT 11
Device: CmuxPacket {'frame_type': <CmuxFrameType.UIH: 4>, 'CR': <CommandResponseType.RESPONSE: 0>, 'DLCI': 1, 'information': b'~\xff}#\xc0!}!
Logging packet
Logging packet
Logging packet
Modem:  CmuxPacket {'frame_type': <CmuxFrameType.UIH: 4>, 'CR': <CommandResponseType.RESPONSE: 0>, 'DLCI': 1, 'information': b'~\xc0!}"}!} }4
Modem:  CmuxPacket {'frame_type': <CmuxFrameType.UIH: 4>, 'CR': <CommandResponseType.RESPONSE: 0>, 'DLCI': 1, 'information': b'~\xc0!}!} } }*
Device: CmuxPacket {'frame_type': <CmuxFrameType.UIH: 4>, 'CR': <CommandResponseType.RESPONSE: 0>, 'DLCI': 1, 'information': b'~\xff}#\xc0!}"
\x03\x06\x00\x00\x00\x00\x81\x06\x00\x00\x00\x00\x83\x06\x00\x00\x00\x00U\x15', 'PF': False}
```

Obrázek 6.3: Navázání spojení přes GSM multiplexing a přepnutí DLCI1 do režimu PPP

Tento příklad dovoluje ověřit správnost podpory pro GSM multiplexing, hlavně navázání spojení a zasílání datových paketů. Také jím jde zjistit, jestli jsou data korektně zasílány na správné modulu implementující AT a PPP funkcionalitu a jestli jsou správně zasílány odpovědi od těchto modulů. Také je ověřena podpora AT příkazu `AT+CMUX`, který přepne modem do GSM multiplexing režimu. Toto není triviální proces, protože AT modul musí informovat proces pro čtení o změně očekávaných dat přicházejících ze sériové linky. Testována je i schopnost parsování a hlavně vytváření GSM multiplexing rámců, které jsou zasílány jako odpovědi pro zařízení.

Kapitola 7

Možnosti rozšíření

Emulátor modemu byl navržen tak, aby bylo jednoduché ho rozšířit. Tím bylo dosaženo jeho implementací jako velmi modulární knihovnou. Rozšířit nebo upravit funkcionalitu je tedy triviální nahrazením nebo rozšířením implementovaných modulů.

7.1 Přidání dalších typů modemů

AT modul je asi nejjednodušeji měnitelnou části emulátoru. Je rozdělen na několik podmodulů, které se mezi sebou dědí funkčnost a každý z nich implementuje svoji část pro zpracování AT příkazů. Pokud bychom chtěli implementovat AT příkazy pro specifický modem, stačí přidat třídu, která je obsahuje a dědí z modulů implementujících zbytek podporované funkcionality. Příkladem je už implementovaný modul pro modem Bg96, který kromě své vlastní funkcionality dědí i implementace ze tříd pro podporu extended GSM příkazů a třídy podporující základní AT příkazy.

Jelikož jsou pro třídu `Modem` moduly AT a PPP přiřaditelné pomocí parametrů konstruktoru, je možné implementaci jednoho z nich nahradit úplně. Způsob komunikace pro emulátor byl úmyslně zvolen využívající generické objekty `asyncio.StreamReader` a `asyncio.StreamWriter`. Díky tomu podporuje modem zasílání dat jakýmkoli rozhraním, které podporuje komunikaci přes tyto dva objekty implementující asynchronní čtení a zápis. v základu je objekt modemu vytvářen s komunikací pro sériovou linku, ale bylo by možné lehce přidat podporu vstupu přes terminál nebo síťové rozhraní.

7.2 SMS funkcionalita

AT příkazy umožňující příjímání a zasílání krátkých textových zpráv (SMS) nebyly v této práci implementovány, jelikož nejsou typicky IoT zařízeními používány. Emulátor modemu by o tuto funkcionalitou nebylo složité rozšířit vytvořením třídy implementující tyto AT příkazy a jejím použitím v AT modulu. Samotná implementace funkcionality by byla složitější. Na rozdíl od protokolu PPP, který jednoduše implementujeme přeposíláním paketů do internetu, podpora zasílání a příjímání SMS je složitější. Pro její funkční implementaci by bylo nutné použít externí zařízení ovladatelné z PC nebo online API, které dovoluje zasílat SMS zprávy pomocí HTTP požadavků.

Kapitola 8

Závěr

Cílem této práce bylo nastudovat problematiku komunikace s modemy pro přístup k mobilním sítím, a to konkrétně přes protokol AT. Následně, po seznámení se s technologiemi používanými mobilními modemy, bylo cílem navrhnout a implementovat emulátor, který dovoluje komunikaci přes sériovou linku, podobně jako mobilní modem.

Práce zkoumala existující způsoby komunikace s modemem pomocí sériové linky. Důraz byl kladen na AT protokol, který byl historicky používán pro konfiguraci a komunikaci s modemem, a který je pořád důležitou součástí spojení přes sériovou linku s moderními modemy. Dále byl kladen důraz na protokol GSM multiplex, který dovoluje moderním modemům komunikovat na několika kanálech přes jednu sériovou linku. Jako částečné rozšíření zadání se bylo seznámeno s podporou protokolu PPP, který umožňuje zařízení připojenému k mobilnímu modemu komunikovat se zařízeními v internetu pomocí protokolu IP. Jedním z dalších cílů této práce bylo podporovat jednoduché rozšíření o více typů modemů, jako řešení tohoto problému bylo rozdělení emulátoru na více modemů, které je možné nahradit těmi, které implementují rozdílnou funkcionalitu.

Před návrhem emulátoru byla analyzována už existující řešení, která implementují emulaci modemu. Bylo rozhodnuto, že mají odlišné cíle od této práce a chybí jim důležité části funkčnosti. Následně byl implementován emulátor který, kromě komunikace se zařízením pomocí již zmíněných protokolů, dovoluje uživateli ovládat parametry této komunikace a simulovat různé scénáře za účelem otestování správné podpory těchto protokolů od připojených zařízení.

V poslední části této práce byla ověřena správnost podpory pro komunikaci přes sériovou linku. Bylo toho dosaženo pomocí připojení zařízení, které podporuje komunikaci s mobilním modemem, k PC s běžícím emulátorem. z výsledků těchto testů vyplývá, že se úspěšně podařilo implementovat emulátor, který podporuje různé AT příkazy, GSM multiplexing a nad rámec zadání implementuje také podporu protokolu PPP.

Literatura

[1] AFFILIATES, O. C. and/or its. *Link Control Protocol (LCP) Frames* [online]. 2010 [cit. 2022-05-09]. Dostupné z: https://docs.oracle.com/cd/E19096-01/sol.ppp301/805-4018/6j3qil165/index.html.

[2] BEAL, V. *Serial Port* [online]. Webopedia, 1996 [cit. 2022-05-06]. Dostupné z: https://www.webopedia.com/definitions/serial-port/.

[3] CELERSMS. *AT Emulator* [online]. 2021 [cit. 2022-05-06]. Dostupné z: https://www.celersms.com/at-emulator.htm.

[4] DAVID HANES, G. S. *Fax, Modem, and Text for IP Telephony*. 1. vyd. Cisco Press, 2008. ISBN 1587052695,9781587052699.

[5] DAVIES, J. *Understanding CRC32* [online]. 2011 [cit. 2022-05-09]. Dostupné z: https://commandlinefanatic.com/cgi-bin/showarticle.cgi?article=art008.

[6] DEVELOPERSHOME.COM. *AT Command Operations: Test, Set, Read and Execution* [online]. [cit. 2022-05-06]. Dostupné z: https://www.developershome.com/sms/atCommandOp.asp.

[7] EVANS, J. M., O'NEILL, J. T., LITTLE, J. L., ALBUS, J. S., BARBERA, A. J. et al. *Standards for computer aided manufacturing*. National Bureau of Standards, 1976.

[8] GPS/GSM, V. *Understanding AT commands* [online]. 2011 [cit. 2022-05-06]. Dostupné z: https://sites.google.com/site/vmacgpsgsm/understanding-at-commands.

[9] HÜBSCHMANN, I. *ESP32 for IoT: A Complete Guide* [online]. 2020 [cit. 2022-05-09]. Dostupné z: https://www.nabto.com/guide-to-iot-esp-32/.

[10] INETDAEMON. *RS232, EIA232, EIA/TIA232* [online]. 2018 [cit. 2022-05-06]. Dostupné z: https://www.inetdaemon.com/tutorials/networking/wan/serial/eia/eia232.shtml.

[11] JAMES F. KUROSE, K. W. R. *Computer Networking: A Top-Down Approach*. 5. vyd. Addison Wesley, 2009. ISBN 0136079679,9780136079675.

[12] KUNILKUDA. *Handling URC (Unsolicited Result Code) in Hayes AT Command* [online]. 2008 [cit. 2022-05-06]. Dostupné z: https://embeddedfreak.wordpress.com/2008/08/19/handling-urc-unsolicited-result-code-in-hayes-at-command/.

[13] LLP., F. *Virtual Modem* [online]. [cit. 2022-05-07]. Dostupné z: https://www.virtual-modem.com/.

[14] Looper, C. de a Jansen, M. *Is 5G as fast as they're saying? We break down the speeds* [online]. Digital Trends Media Group, 2022 [cit. 2022-05-06]. Dostupné z: https://www.digitaltrends.com/mobile/how-fast-is-5g/.

[15] Scott, M. L. *Shared-Memory Synchronization*. 1st. Morgan & Claypool, 2013. Synthesis Lectures on Computer Architecture. ISBN 160845956X,9781608459568.

[16] S.Lawyer, D. *Antique Modems* [online]. 2007 [cit. 2022-05-09]. Dostupné z: https://tldp.org/HOWTO/Modem-HOWTO-29.html.

[17] SmartWorx, A. B. *AT Modem Emulator* [online]. 2016 [cit. 2022-05-07]. Dostupné z: https://advantech-bb.com/technical-learning-support-center/verizon-at-modem-emulator/#1457466415734-962d0213-2ddf.

[18] s.r.o., S. electronic. *QUECTEL BG96 – IoT modul pripravený na budúcnosť* [online]. 2017 [cit. 2022-05-09]. Dostupné z: https://www.sos.sk/articles/quectel/quectel-bg96-iot-modul-pripraveny-na-buducnost-2036.

[19] Staff, H. C. *The Complete History of the Modem* [online]. 2021 [cit. 2022-05-06]. Dostupné z: https://history-computer.com/modem-complete-history-of-the-modem/.

[20] William, D. *That's when I reach for my revolver...* [online]. 2009 [cit. 2022-05-06]. Dostupné z: https://blogs.gnome.org/dcbw/2009/03/20/thats-when-i-reach-for-my-revolver/.

[21] *58666, AT-Modem-Emulator, 10/100BT* [online]. [cit. 2022-05-07]. Dostupné z: https://www.top-electronics.com/en/at-modem-emulator-10-100bt.

Příloha A

Obsah přiloženého paměťového média

Struktura souborů na paměťovém médiu je následující:

- `src/` - složka se zdrojovými soubory
- `doc/` - složka obsahující zdrojové soubory textu práce
- `xhorky33.pdf` - text práce
- `README.md` - návod k použití