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 ACCELERATING THE DEVELOPMENT OF EMBEDDED DEVICES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MAREK SECHRA**
AUTHOR

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

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Sechra Marek**
Program: Informační technologie
Název: **Emulátor modemu pro akceleraci vývoje vestavěných zařízení**
Cellular Modem Emulator for Accelerating the Development of Embedded Devices
Kategorie: Počítačové sítě

Zadání:

1. Seznamte se s protokolem AT používaným pro komunikaci s modemy sloužícími k přístupu 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 ESP-IDF. Rozhraní emulátoru navrhněte tak, aby bylo možné specifikovat různé testovací scénáře a zvolit konkrétní model emulovaného modemu.
3. Navržený emulátor implementujte s využitím objektově orientovaného přístupu k programování formou Python modulu. Dbejte na nezávislost na použité platformě a možnost rozšiřitelnosti o další typy modemů. Implementujte minimálně podporu pro modem BG96 a režim PPP.
4. Vytvořte demonstrační aplikaci, ukazující způsob využití navrženého emulátoru a diskutujte jeho parametry.
5. Shrňte dosažené výsledky a diskutujte další možná rozšíření práce.

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

Úlohou této práce je seznámit čtenáře s rozmanitostí trhu a podpory jednotlivých modemů pro AT příkazy. Seznámit ho s používanými sítěmi a s nimi související protokoly a uvést čtenáře do problematiky AT příkazů. Navrhnout a implementovat emulátor modemu, který bude snadno rozšiřitelný. Běh emulátoru je zaznamenáván do terminálové konzole. Emulátor je implementovaný v jazyce Python.

Abstract

The task of this work is to acquaint readers with the diversity of the market and the support of individual modems for AT commands. Get acquainted with other networks and related protocols and introduce readers to the issue of AT commands. Design and implement a modem emulator that will be easily extensible. The run of the emulator is recorded in the terminal console. The emulator is implemented in Python.

Klíčová slova

Emulátor, Modem, Quectel, BG96, AT příkazy, GSM, LTE, NbIoT, LTE-M, PPP

Keywords

Emulator, Modem, Quectel, BG96, AT commands, GSM, LTE, NbIoT, LTE-M, PPP

Citace

SECHRA, Marek. *Emulátor modemu pro akceleraci vývoje vestavěných zařízení*. Brno, 2021. 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.

.....
Marek Sechra
10. května 2022

Poděkování

Rád bych poděkoval vedoucímu práce doc. Ing. Zdeňku Vašíčkovi Ph.D. za cenné rady, návrhy, připomínky a skvělou komunikaci při vedení bakalářské práce.

Obsah

Kapitola 1

Úvod

Velký technologický pokrok v mobilních sítích přinesl spoustu možností a inovací. Svět mobilních sítí se dostal do fáze, kdy existuje mnoho specifických řešení pro jednotlivé typy sítí, které vznikaly plynule s technologickým pokrokem. Prostředek, který spojuje analogický svět s digitálním je modem. Slouží k převodu analogického signálu do digitální podoby. Modem se nejprve rozvíjel na akademické úrovni a až po dosažení technologického pokroku vstoupil i do komerční sféry. I přes snahu udržení standardů vznikaly a vznikají v komerční sféře různorodá řešení, která se navzájem mohou výrazně lišit. Důsledky tohoto technologického rozvoje můžeme zaznamenat v nekompatibilitě modemů. Dalším důsledkem technologického vývoje je nátlak na řešení v rámci IoT. Proto vznikají zařízení typu mikrokontrolér, u kterých je jedna z nejdůležitějších vlastností v komerční sféře cena a rychlý vývoj. Důsledkem může být nekvalitní firmware, který může obsahovat chyby, které se ve vývoji složitě odhalují, jelikož mohou nastat až při zvláštních situacích. V současnosti můžeme zaznamenat různorodé modemy pro různorodé sítě. Například modem GSM je určen pro sítě typu GSM. Náš referenční modem, který se bude emulovat se jmenuje BG96, který byl vyvinut firmou Quactel a je určen pro LTE Cat M1, Cat NB1, EGPRS sítě. Byl vybrán pro jeho širokou použitelnost a univerzálnost. Obecně platí, že síťová zařízení jsou velice náročná na otestování z pohledů zdrojů.

Kapitola 2

Typické modemy v IoT

Zde si uvedeme typické modemy vhodné pro IoT řešení, jelikož k dnešnímu dni existuje velké množství takových modemů. Uvedeme si pouze ty nejzajímavější z pohledu dodržování standardů a podpory více typů telekomunikačních sítí. Komplexní tabulky pro přehled budou přiložené v příloze této práce. B Firma Quectel, U-blox, SimCom jsou jedni z největších firem, které vyrábí modemy pro širokou škálu datových sítí. Firma Sierra wireless, Nordic, Murata se především zabývají výrobou modemů pro IoT sítě a jejich portfolium produktů je značně omezené.

Zvolený modem BG96

Modem BG96 je navržený firmou Quectel. Maximální datový přenos je stanoven na rychlost na 375kb/s. Podporuje globální frekvenční pásma a zároveň je energeticky úsporný. Je navržený pro sítě LTE Cat M1/Cat NB1 a EGPRS. Jeho rozměry jsou 22.5 x 26.5 x 2.3 (mm). Podpora AT příkazů zahrnuje kombinaci standardů 3GPP 27.005 a 3GPP 27.007 s nadstavbou od firmy Quactel. Nabízí také TCP/IP stack s podporou protokolů TCP, UDP nebo PPP. Rozhraní pro BG96 zahrnuje 2 USB porty a 1 UART [16].

BG95-M5

Modem BG95 se vyskytuje v sériích M1, M2, M3, M4, M5. Jelikož série M5 je nejnovější, uvedeme si právě tuhle variantu. Stejně jako modem BG96 je navržený firmou Quectel. Datový přenos pro upload je stanoven na maximální hranici 588kbp/s a pro download 1119kbp/s pro síť LTE Cat M1. Je navržený pro sítě LTE Cat M1, Cat NB2 a EGPRS s integrací pro GNSS. Rozměry: 23.66 x 19.9 × 2.2 (mm). Podpora AT příkazů zahrnuje kombinaci standardů 3GPP 27.005 a 3GPP 27.007 s nadstavbou od firmy Quactel. Rozhraní obsahuje 1 USB port a 1 UART. Nabízí také TCP/IP stack s podporou protokolů TCP,UDP a PPP [15].

SARA-R422M8S

Modem SARA-R422M8S spadá pod sérii modemů SARA-R4 od firmy U-blox. Jedná se o nejnovější variantu téhle série, proto si představíme tuhle variantu. Datový přenos v sítí LTE Cat M1 pro upload je stanoven na maximální hranici 588 kbit/s pro download 1119 kbit/s. V sítí LTE Cat NB2 je hranice pro upload 127 kbit/s a pro download 158.5 kbit/s Rozměry: 26 x 16 x 2.5 (mm). Podpora AT příkazů zahrnuje kombinaci standardů 3GPP 27.005, 3GPP 27.007 a 3GPP 27.010. Rozhraní obsahuje 1 USB port a 1 UART. Nabízí podporu pro protoly: FTP,HTTP, TCP, UDP, MQTT [14].

SIM800

Modem je navržený firmou SIMCom. Modem SIM800 si představíme z důvodu podobných parametrů jako má náš zvolený modem BG96. Maximální datový přenos je stanoven na rychlost na 85.6 kbp/s Rozměry: 24 x 24x 3 (mm). Podpora AT příkazů zahrnuje kombinaci standardů 3GPP 27.005 a 3GPP 27.007 s nadstavbou od firmy SIMcom. Rozhraní obsahuje 1 USB port a 1 UART. Nabízí také TCP/IP stack s podporou protokolů TCP,UDP a PPP [11].

EM7455

Tento modem je od firmy Sierra wireless. Modem EM7455 spadá pod sérii modemů EM74 od firmy Sierra wireless. Jedná se o nejnovější variantu, proto si uvedeme tuhle variantu. Datový přenos v sítí LTE Cat 6 pro upload je stanoven na maximální hranici 50Mbp/s a pro download 300Mb/s. Rozměry : 42 x 30 x 2.3 (mm). Podpora At příkazů je pouze vlastní sadou AT příkazů od firmy Sierra wireless. Nabízí podporu pro TCP/IP stack [14].

nRF9160

Zařízení navržené firmou Nordic. Zde se nejedná pouze o modem. Jedná se o komplexní řešení pro IoT, které obsahuje kromě modemu i mikrokontrolér. Podpora AT příkazů zahrnuje standardy 3GPP 27.005, 3GPP 27.007 a 3GPP 27.010. Modem má rozměry : 10 x 16 x 1.04 (mm). Modem podporuje TCP/IP stack pro IPv4 i pro IPv6 [18].

2.1 Přístup k síti

GSM

GSM představuje celosvětový standard pro komunikaci v telekomunikačním odvětví. GSM pochází z názvu pracovní skupiny, která tento standard vyvíjela (Groupe Special Mobile). GSM síť funguje na několika frekvencích například v České republice jsou pro 2G sítě používány frekvence v rozsahu od 900 až 1800 [MHz]. Tento standard umožňuje oboustrannou komunikaci mezi příjemcem a volaným. Základním prvkem pro správnou funkci je SIM karta, která umožňuje komunikaci mezi zařízeními. GSM bylo stavebním kamenem pro další rozvoj komunikačních standardů jako EDGE, UMTS, LTE. Roku 1998 bylo zformované 3GPP, které mělo původně vytvořit podhoubí pro novou generaci komunikace v 3G. 3GPP znamená 3. generace partnerství projektu. V tomto partnerství se objevili mimo jiné japonské, americké, čínské, evropské organizace. Tato skupina již dnes pokračuje na vývoji 4G, 5G sítí. GSM modem umožňuje odesílat a přijímat zprávy. K provádění těchto úkonů musí GSM modem podporovat příkazy AT. Pro základní nastavení pro odesílání a přijímání např. SMS zpráv musí modem podporovat rozšířenou sadu AT příkazů k tomu se používá standard 3GPP TS 27.005 nebo známý jako ETSI TS 27.007 V10. Použití AT příkazu lze použít i pro MMS a FAX zprávy. GSM modemy se používají dodnes kvůli jejich dostupné ceně [28].

LTE

Sítě GSM vlivem technologického vývoje nedosahovali dostatečné rychlost vzhledem k datovému objemu přenášených po síti. Proto mobilní operátoři volali po nahrazení stávajících GSM technologiích. První komerční nasazení probíhalo zlomem roku 2009 a 2010 .[12] Systém LTE je postavený na architektuře SAE. Celý systém se skládá z nové rádiové sítě LTE a jádra SAE. LTE síť je založena na ortogonálním frekvenčním multiplexu OFDM. Ten je vhodný pro širokopásmové aplikace. Nastavení šířky rádiového pásma je velmi flexibilní, jedná se o rozsah 1,4MHz až 20MHz. LTE obecně posouvá hranice datových sítí a přidává techniky jako paketové přepínání a přepínání okruhů. V rámci specifikace 3GPP se rychlost přenosu na rádiovém pásmu pohybuje na 20MHz [22]. LTE umožňuje vzájemné propojení s již existujícími sítěmi 3GPP, také podporuje spolupráci s jinými sítěmi. LTE splňuje téměř všechny požadavky pro 4G sítě jako například LTE-A. Při použití LTE můžeme snížit náklady na připojení a zvýšit její rychlost. LTE je ideální pro uživatele, kteří nemohou získat konvenční ADSL připojení nebo kabelové připojení [29].

NbIoT

NbIoT můžeme volně přeložit jako úzkopásmovou síť (Narrow-Band IoT) pro internet věcí. Jedná se o LPWA rádiovou technologii, jejíchž standard je vyvíjen organizací 3GPP. Tato technologie je definována v dokumentu 3GPP Release. Koncepce vychází ze sítě LTE. NbIoT využívá stejnou modulaci, kanálové kódovaní, překládání apod [13]. Cílem bylo vylepšení komunikace zařízení oproti GSM, LTE pro použití v IoT. Síť NbIoT se specializuje na venkovní přenos s důrazem na nízkou cenu a nízkou spotřebou energie. Tato síť je určena pro mobilní zařízení a vestavěné systémy. NBIoT využívá spíše pásma v nižších frekvenčních pásmech, kvůli očekávání použití v náročných rádiových podmínkách [19]. S tímto pojmem souvisí Cat-NB1 a současně Cat-NB2, což jsou standardy pro tuto síť, které upřesňují přenosovou rychlost, latenci, šířku pásma. V Evropě tuto službu poskytuje firma Vodafone.

NbIoT využívá licencované pásmo, takže tato síť je dostupná v podstatě všude, kde se nachází signál od operátora. Nevýhodou licencovaného pásma je spotřeba energie a také modem musí být v sítí synchronizovaný [9].

LTE-M

LTE-M je bezdrátová technologie typu LPWAN, který se rovněž vyvinul pomocí organizace 3GPP. Výhodou oproti NbIoT je vyšší rychlost přenosu dat a mobilita, také ale potřebuje větší šířku pásma. LTE-M je podstatně levnější technologie a podporuje větší výdrž baterie u zařízení, proto se LTE-M drží větší oblibě než NbIoT [17]. Pro pochopení rozdílů mezi těmito technologiemi si uvedeme tabulku, která nám tuto problematiku přiblíží.

Porovnání LTE-M a NbIoT.

Technologie	Maximum Uplink Speed	Maximum Downlink Speed	Latence	Maximální ztráta	Bandwidth
LTE-M	1 [Mbit]	1 [Mbit]	10-15 [ms]	156 [dBs]	1.4 MHz to 5 [MHz]
NB-IoT	127 [Kbit]	159 [Kbit]	1.6-10 [s]	164 [dBs]	180 [KHz]

Tabulka 2.1: Porovnání LTE-M a NbIoT

[1]

Kapitola 3

Příkazy AT

3.1 Historie

V roce 1981 firma Hayes Communication vyvinula zařízení Smartmodem, které bylo možno ovládat pomocí knihovny, která obsahovala jednoduché textové řetězce. Před tímto převratem se spojení řídilo manuálně. Mobilní operátoři museli ručně přepínat kontexty nebo ručně vytáčet číslo a propojovat hovory. Sada příkazů Hayes obsahuje příkazy pro různé operace na telefonií lince jako je zavěšování či vytáčení hovorů. Dále obsahuje příkazy pro nastavení modemu. Původní sada byla koncipovaná pro modemy s 300 Baud. Při technologickém pokroku v tomhle odvětví a zvednutí rychlosti komunikace na 1200 až 2400 Baud bylo nutné rozšířit sadu příkazů. To dalo za vznik standardu TIA/EIA-602, kterému se přezdívalo Hayes command set.

Další standardizaci přivedla síť GSM a vznikl standard GSM 07.07(3GPP 27.007), na který navázal standard GSM 07.05(3GPP 27.005). Tyto standardy zavedli příkazy pro práci se zprávami SMS [8].

Dnes již má každá firma svoji vlastní rozšiřující sadu příkazů. Většinou jsou specifikované v manuálové stránce modemu nebo v manuálové stránce pro AT příkazy pro daný modem či sérii modemů.

3.2 Syntaxe

Každý AT příkaz začíná prefixem AT nebo at poté následuje zkratka nějakého typicky anglického slova, který popisuje sémantiku daného příkazu. Jedná se o jednořádkové příkazy. Odpověď m§že být ve tvaru <CR><LF>[odpověď]<CR><LF> K potvrzení tedy slouží znak konce řádku <CR>, ale ve většině případech se používá kombinace <CR><LF> tahle varianta funguje i na OS Microsoft Windows [24].

Obecná syntaxe pro AT příkazy

- **AT<příkaz>?** Příkaz pro čtení aktuální nastavené hodnoty.
- **AT<příkaz>=?** Příkaz pro dotaz na podporu pro dané zařízení.
- **AT<příkaz>=<hodnota>** Příkaz pro provedení.

Syntaxe příkazů pro modem BG96

- **AT + <příkaz>** Příkaz pro pro provedení bez parametrů.
- **AT + <příkaz>= <parametry>** Příkaz pro provedení
- **AT + <příkaz >?** Příkaz pro čtení aktuální nastavené hodnoty.
- **AT + <příkaz>=?** Příkaz pro dotaz na podporu pro dané zařízení

3.3 Typy příkazů

3.3.1 Příkazy definované výrobcem

Typické příkazy:

- **ATI** Příkaz vrací informace o modemu
- **AT+GMI** Příkaz vrací název výrobce.
- **AT+GMM** Příkaz vrací název modemu.
- **AT+CFUN** Příkaz pro nastavení funkcionální úrovně.
- **AT+CPIN** Příkaz pro práci s pinem karty SIM.
- **AT+COPS** vrací aktuální nastavení parametrů pro příkazy.
- **A/** Příkaz pro znovu provedení předchozího příkazu.

Příklad pro zjištění informací definované výrobcem:

ATI
Quectel
BG96
Revision: BG96MAR01A01M1G
OK
AT+CFUN=0
OK
AT+COPS?
+COPS:2
OK
AT+CPIN?
+CME ERROR: 13
AT+CFUN=1
OK
+CPIN SIM PIN
AT+CPIN=1111
OK
+CPIN: READY
+QUSIM:1
+QIND: SMS DONE

Obrázek 3.1: AT Příklad pro zjištění informací

V příkladu můžeme vidět sekvenci příkazů a odpovědi na ně. Tučně jsou zvýrazněny AT příkazy. Odpovědi se vždy nachází za AT příkazem, kde se odpověď skládá ze dvou částí. Pro jednoduchost si je pojmenujeme jako odpověď a potvrzení správnosti. Vidíme, že u prvních třech příkazů nenastává žádný problém. U příkazu **AT+CPIN?** dochází k erroru číslo 13, který značí chybu SIM karty. Dochází poté k přepnutí funkcionální úrovně, kde se v následujících příkazech SIM karta úspěšně aktivuje.

3.3.2 Příkazy pro ovládání sériového rozhraní

Typické příkazy:

- **AT&C** Příkaz řídí detekci nosiče dat.
- **AT+IFC** Příkaz určuje chování sériového portu.
- **AT&W** Příkaz ukládá nastavení komunikace sériového portu.
- **AT+IPR** Příkaz nastavuje rychlost přenosu.

Příklad nastavení přenosového pásma:

AT+IPR=115200
OK
AT&W
OK
AT+IPR?
+IPR:115200
OK

Obrázek 3.2: AT příklad přenosového pásma

První příkaz slouží pro nastavení přenosového pásma. Následující příkaz ukládá hodnotu přenosového pásma do nastavení modemu. Toto nastavení tedy bude stejné i po restartu modemu. Poslední příkaz je pro výpis hodnoty rychlosti přenosového pásma.

3.3.3 Příkazy pro práci se sítí

Typické příkazy:

- **AT+CREG** Příkaz pro informaci o registraci zařízení
- **AT+COPS** Příkaz pro výpis aktuálních operátorů.
- **AT+COPN** Příkaz pro výpis jmen operátorů.
- **AT+CPOL** Příkaz pro editaci pole preferovaných operátorů.
- **AT+QPSMS** Příkaz od firmy Quectel pro PSM nastavení.

Příklad registrace k operátorovi

AT+CREG=2
OK
AT+COPS=0,"Telie Telia",9
OK
AT+COPS?
+COPS:0,0,"Telie Telia",9

Obrázek 3.3: AT Příklad registrace k operátorovi

U prvního příkazu vidíme přiřazení čísla 2, což značí následující změnu operátora. Poté dochází k nastavení nového operátora a kontroly, jestli se operátor skutečně nastavil.

3.3.4 Příkazy pro TCP/IP

Typické příkazy:

- **AT+QICSGP** Příkaz pro nastavení parametrů komunikace.
- **AT+QIACT** Příkaz pro aktivaci PDP kontextu.
- **AT+QIOPEN** Příkaz pro otevření socketu.
- **AT+QICLOSE** Příkaz pro zavření socketu
- **AT+QISTATE** Příkaz pro se stavem socketu.
- **AT+QISEND** Příkaz po odeslání dat.

Příklad jednoduchého odeslání dat

AT+QIOPEN=1,0,"TCP","220.200.200.200",8880,0
OK
AT+QISEND=0,5
>test1
SEND OK
AT+QISEND=0,0
QISEND: 9,9,0
OK
AT+QICLOSE=0
OK

Obrázek 3.4: AT příklad jednoduchého poslání dat

První AT příkaz otevírá TCP spojení pro IPv4 adresu. V druhém příkazu se indikuje poslání dat, číslo pět značí počet fixní délku dat v bytech. Poté se data odesílají. V dalším příkazu probíhá dotaz na odeslanou délku dat a poté zavření spojení.

Příklad příkazu ping na danou doménu

AT+QPING=1,"www.google.com"
+QPING 0,"8.8.8.8",32,192,255
+QPING 0,"8.8.8.8",32,178,255
+QPING 0,"8.8.8.8",32,156,255
OK

Obrázek 3.5: AT příklad příkazu ping na danou doménu

AT příkaz obsahuje dva parametry, číslo kontextu a cílovou doménu. V odpovědi číslo nula značí úspěch příkazu, dále je zaznamenána adresa, délka dotazu, čas délky odpovědi v milisekundách a TTL.
Příklad získání chybové hlášky

AT+QIOPEN=1,"TCP","220.180.239.212",8009,0,1
ERROR
AT+QIGETERROR
+QIGETERROR: 552, invalid parameters
OK

Obrázek 3.6: AT příklad jednoduchého poslání dat

První AT příkaz se pokouší otevřít TCP spojení se zařízením, které má danou IPv4 adresu. Spojení dopadlo neúspěšně. Další příkaz se doptává, o jakou konkretní chybu se jednalo.

3.3.5 Příkazy pro zahájení komunikace PPP

Typické příkazy:

AT+CGREG Příkaz pro nastavení či dotázaní na nastavení sítě
AT+CGDCONT Příkaz pro definici PDP kontextu
ATD Příkaz pro odchozí hlasové nebo datové volání. Pro indikaci zahájení komunikace skrz PPP se využívá s parametrem *99#. Celý příkaz je tedy ATD*99#.

3.3.6 Příklad pro nastavení PPP

AT+CGREG?
CGREG:0,1
AT+CGDCONT=1,"IP","CMNET"
OK
ATD*99#
CONNECT 115200
7EFF23C0217D..

Obrázek 3.7: AT příklad jednoduchého poslání dat

Kapitola 4

PPP

4.1 Úvod

PPP - Point to Protocol poskytuje metodu spojení mezi dvěma body na sériové lince. Řadí se tedy na druhou vrstvu ISO/OSI modelu. Tento protokol pracuje obousměrně tzv. full duplex za předpokladu, že pakety budou doručeny ve správném pořadí. Používá se pro získávání identity uživatele a jeho IP adresy. PPP obsahuje několik dalších protokolů. Pro úvodní část vyjednávání spojení se používá protokol LCP - Link Control Protocol. PPP se používá pro ustanovení spojení mezi dvěma stanicemi. Používá se na sériové lince.

Protokoly

Protokol PPP obsahuje sadu protokolů, které slouží k vyjednání potřebných parametrů ke komunikaci.

- Linkový řidicí protokol (LCP) - pomocí tohoto protokolu se nastavují základní parametry přenosu.
- Autentifikační protokoly PAP, CHAP, které mají za úkol zabezpečit komunikaci.
- Internetový řídicí protokol (NCP), který vyjednává informace pro vyšší síťový protokol. Například pro protokol IP slouží IPCP, který vyjednává IP adresu a IP adresy pro DNS servery.

[20]

Stavový automat

Proces vyjednávání je kontrolovaný stavovým automatem, který zabezpečuje správný chod příkazů v jednotlivých situacích [26].

Vyjednávací proces

Vyjednávací proces protokolu PPP by měl dospět do bodu shody ve vyjednávání konfiguračních možností, které budou shodou pro obě strany. Pokud jedna strana odmítne klíčovou konfigurační možnost, nedojde k úspěchu vyjednávacího procesu [26].

Spolehlivost

Přenos se považuje za nespolehlivý, jelikož neprobíhá žádné potvrzení komunikace. Pokud se nepodaří přijmout odpověď, paket s dotazem se musí vyslat znovu [26].

Data

Nutná část je vyjednávání, kde musí dojít k dohodě, aby došlo k přenosu dat na vyšší síťové vrstvě. Protokoly vyšší síťové vrstvy mohou zabalovat data, která obsahují data vyšších síťových protokolů [26].

4.1.1 SLIP

SLIP - Serial Line Internet Protocol. Jedná se o protokol, který zabaluje IP pakety do datagramů jednoduchým způsobem. Protokol je navržený pro komunikaci pomocí sériových portů a modemových spojení. Byl navržen v roce 1988 a je definován v RFC 1055 [5]. Dneska se již jedná o historický předvoj protokolu PPP, který tento protokol u osobních počítačů téměř nahradil. V současnosti se používá u jednočipových počítačů kvůli nízké režii. Jedna z jeho nevýhod je absence detekce chyb, tato povinnost se deleguje na protokoly vyšších vrstev [5].

4.1.2 CSLIP

CSLIP - Compressed SLIP. Tento protokol vychází z protokolu SLIP a přidává možnost komprese TCP/IP záhlaví. TCP/IP hlavička má 40 bajtů a CSLIP ji komprimuje na 3-16 bajtů. Je důležité zmínit, že komprese probíhá pouze u hlavičky paketu ne u dat. Podstata vychází z myšlenky, že hlavička TCP/IP během TCP spojení zůstává ve spoustu polí statická, lze tedy přenášet pouze přírůstky příznaků. Ke kompresi nedochází v případě, že se používá protokol ICMP nebo UDP. Pokud probíhá handshake na úrovni TCP, také nedochází ke kompresi. Takové pakety mají příznaky Syn,Rst,Fin. Protokol CSLIP je popsán v RFC 1444 [6].

4.2 HDLC paket

Během binárního toku dat po sériové lince je potřeba rozlišovat jednotlivé pakety. Každý jednotlivý paket obsahuje počáteční a koncovou značku (Flag). Pro tento účel se používá protokol HDLC - High level Data Link Control. V následující tabulce je vyobrazena obecná struktura HDLC paketu a konkrétní PPP paket.

HDLC	Flag	Adresa	Control	Informace	FCS	Flag
	1 [Byte]	1 [Byte]	1-2 [Byte]	0-1502 [Byte]	1 [Byte]	1 [Byte]
PPP	0x7E	0xFF	0x03	Data	8	0x7E

Tabulka 4.1: Struktura paketu HDLC

Z důvodu, že tato práce se bude zabývat výhradně PPP pakety můžeme říct, že takový paket bude vždy začínat **0x7EFF03** a končit **0x7E**.

Složka Informace obsahuje přenášena data. Složka FCS slouží jako kontrolní součet.

4.3 Ustanovení spojení

Ustanovení spojení je nejpodstatnější část PPP. První ustanovení probíhá jako domluva na parametrech spojení jako adresování nebo autentifikace. Ustanovení začíná tím, že jedna ze stran komunikace vysílá požadavek, na který odpovídá protistrana potvrzením či zamítnutím. Poté probíhá komunikační dialog. První požadavek posílá vysílací stanice. Uzavření spojení probíhá na základě dotazu o ukončení a potvrzení o ukončení druhou stranou spojení.

4.3.1 LCP

Protokol LCP se využívá při ustanovení spojení v PPP. Využívá se pro konfiguraci parametrů komunikace. komunikace.Komunikace PPP vždy bude začínat tímto protokolem. Je zabalený v základním paketu PPP v poli informace 4.1.

Protokol	Kód	ID	Délka paketu	Data
2 [Byte]	1 [Byte]	1 [Byte]	2 [Byte]	1496 [Byte]

Označení pro protokol LCP je **0xC021**. Kód označuje typ příkazu v protokolu LCP např. hodnota **0x01** označuje konfigurační dotaz. Identifikátor označuje jednotlivé dotazy. Tazatel předpokládá odpověď se stejnou hodnotou v tomto poli. Délka paketu je součet ostatních položek v bytech. Položka data v tabulce je popsána níže.

Kód	Název	Význam
1	Configure Request	Paket, který obsahuje požadavky na konfiguraci.
2	Configure Ack	Paket nesoucí potvrzení požadavku.
3	Configure Nack	Paket obsahuje zamítnutí požadavku a informace s čím nesouhlasí.
4	Configure Reject	Paket obsahuje zamítnutí všech požadavků.
5	Terminate Request	Požadavek na ukončení spojení.
6	Terminate Ack	Paket obsahuje potvrzení ukončení spojení.
7	Code Reject	Odmítnutí požadavku, kvůli neznámému kódu.
8	Protokol Reject	Druhá strana nepodporuje domlouvaný protokol.
9	Echo Request	Podpora testovací smyčky.
10	Echo Reply	Odpověď na podporu testovací smyčky.
11	Disckard Request	Zahození paketu.

Tabulka 4.2: Konfigurační Kódy LCP
[26]

Typ volby	Délka	Data	Typ volby	Délka	Data
1 [Byte]	1 [Byte]	1-n [Byte]	1 [Byte]	1 [Byte]	1-n [Byte]

[26]

Důležitá je část paketu typ volby. Jednoznačně identifikuje konfigurační informaci čili o jaký typ konfigurace se jedná.

Typ volby	Název
0	Rezervovaný
1	Maximum přijatých složek
2	Asynchronní kontrolní mapa
3	Autentizační protokol
4	Protokol kvality
5	Magické číslo
6	Rezervovaný
7	Komprese datového pole paketu PPP
8	Komprese kontrolního a datového pole

Tabulka 4.3: Konfigurační možnosti LCP
[26]

4.3.2 Konfigurační možnosti spojení

Maximum přijatých složek (MRU) Tato konfigurační možnost se využívá v případě, že chceme informovat druhou stranu komunikace o změně velikosti paketu. Tato hodnota je přednastavená na 1500 [Byte]. Pokud chceme nastavit menší velikost paketů, musíme být připraveni i na variantu přijmutí paketu původní délky, kvůli možnosti ztráty spojení. Pole typ volby a délka má velikost 1 byte. Pole pro maximum přijatých složek má velikost 2 byty. [26]

Typ volby	Délka	MRU
1	4	

Asynchronní kontrolní mapa (ACCM) Konfigurační volba dovoluje změnu mapování kontrolních znaků. Původní PPP pakety mapují všechny kontrolní znaky. Pomocí operace XOR na dvojitou délku. Informace tedy obsahuje, který znak musí zůstat mapovaný a který nikoliv. Druhá strana však může nadále mapovat zakázané znaky z důvodu omezené implementace na zařízení. Pole asynchronní kontrolní mapy je velké 4 byty. Typ volby a délka zůstává jako u předchozích nastavení [26].

Typ volby	Délka	ACCM
2	6	

Autentizační protokol (AP) Toto nastavení obsahuje informaci o tom, který autentizační protokol se bude používat např. PAP(0xC023), CHAP(0xC223). Tedy obsahuje informaci o tom, jaký protokol se pokusí strana vyjednat. Druhá strana nemusí daný protokol podporovat. Pole pro autentizační protokol má velikost 2 byty a pole data mohou obsahovat dodatečné informace o protokolu např. verzi. Nelze tedy určit jeho statickou velikost [26].

Typ volby a délka zůstává jako u předchozích nastavení.

Typ volby 3	Délka >= 4	AP	Data

Protokol kvality (QP) Nastavení obsahuje informaci o určení kdy a jak často má linka přestat přenášet data. Implicitně je toto nastavení vypnuté. Pole pro protokol kvality má velikost 2 byty a data mohou obsahovat dodatečné informace o protokolu např. verzi. Nelze tedy určit jeho statickou velikost. Typ volby a délka zůstává jako u předchozích nastavení [26].

Typ volby 4	Délka >= 4	QP 0xc025	Data

Magické číslo Jak už název napovídá, jedná se o konfigurační volbu, která obsahuje náhodné číslo, které slouží k detekci smyček a anomálií v komunikaci. Je žádoucí, aby číslo bylo co nejvíce náhodné. Doporučeným způsobem k získání jedinečného čísla je zabudování do výpočtu dynamicky se měnící proměnou. Typickým příkladem takové proměnné je čas. Pokud je vyslaný paket s takovým nastavením, který je typu *Configure Request*, poté se čeká na *Configure Request* od proti strany, který by měl obsahovat jinou hodnotu magického čísla. Pokud je magické číslo identické značí to výskyt smyčky v komunikaci a vyšle se zamítnutí dotazu s jiným magickým číslem. V konfigurační odpovědi *Configure Ack* by se naopak mělo magické číslo shodovat s hodnotou, která byla v dotazu. Problém spočívá v tom, že tohle nastavení slouží pouze pro identifikaci smyček nikoliv k řešení. Tedy pokud by došlo ke smyčce budou se stále odesílat dotazy a jejich zamítnutí ve smyčce. Velikost pole pro magické číslo jsou 4 byty. Typ volby a délka zůstává jako u předchozích nastavení [26].

Typ volby 5	Délka >= 6	Magické číslo

Komprese datového pole(FCS) Tato konfigurační volba vyjednává kompresi ohledně PPP protokolu. Implicitně v PPP každé číslo např. 0x01 se kóduje jako 0xFD21. Některá hodnota však může být zkomprimovaná do 1 bytu. Paket v rámci komunikace skrz protokol LCP není nikdy komprimovaný, tak lze odlišit LCP od ostatních protokolů v rámci PPP.

Typ volby a délka zůstává jako u předchozích nastavení [26].

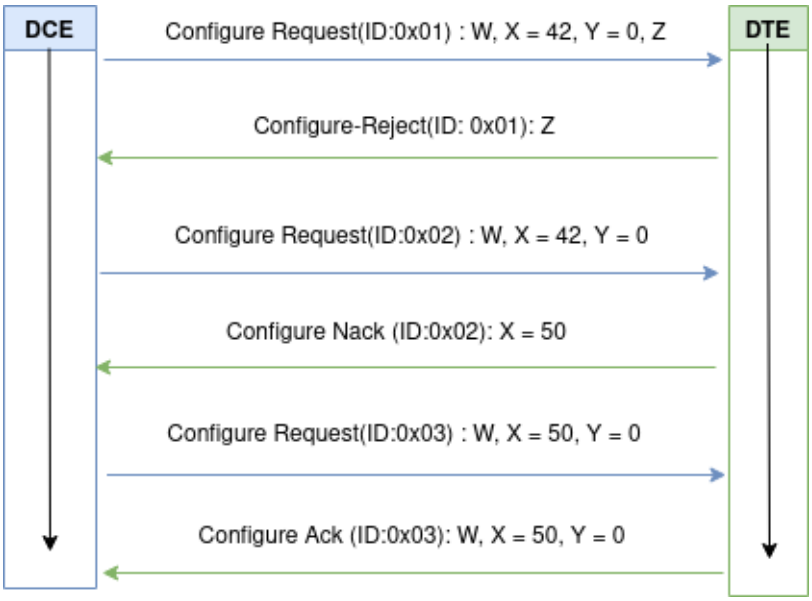
Typ volby 7	Délka 2

Komprese adres a kontrolních polí (ACFC) Tato konfigurační možnost poskytuje nastavení na vyjednání komprese adres a kontrolních polí. Implicitně je komprese nastavená na kompresi využívajícího protokolu. Posílá se tedy informace o skutečnosti, že vysílací strana je schopna přijímat zkomprimované adresy a pole. Typ volby a délka zůstává jako u předchozích nastavení [26].

Typ volby 7	Délka 2

4.3.3 Vyjednávací proces u LCP

LCP vyjednávací postup je sériová výměna paketů mezi účastníky komunikace. Důvodem vyjednávání je dohoda na parametrech komunikace. Vyjednávaní jsou ve skutečnosti dva samostatné dialogy mezi účastníky komunikace [25].



Obrázek 4.1: LCP vyjednávací proces
[25]

Obrázek zobrazuje hypotetickou komunikaci mezi zařízeními DTE - Data Terminal Equipment a DCE - Data Communications Equipment.

- DCE posílá dotaz s navrhovaným nastavením pro komunikaci.
- DTE odpovídá zamítnutím dotazu a posílá nastavení, s kterým nesouhlasí.
- DCE posílá dotaz s upraveným nastavením.
- DTE odpovídá nesouhlasem, posílá vhodné nastavení pro X.
- DCE posílá dotaz s upraveným nastavením.
- DTE posílá potvrzení.

Potvrzením(Configure Ack) dojde k dohodě na domlouvaném nastavením [25].

4.3.4 IPCP

IPCP - IP Control protocol. Jedná se o řídící protokol pro internetový protokol IP. Pokud chceme přenášet data pomocí IP je potřeba vyjednat spojení. V tomhle směru nám pomůže protokol IPCP, který spadá pod kategorii NCP jedná se o soubor řídících protokolů pro vyjednání spojení [23]. Formát paketu se podobá paketu LCP a vypadá následovně.

Protokol	Kód	Identifikátor	Délka paketu	Data
2 [Byte]	1 [Byte]	1 [Byte]	2 [Byte]	0-5270[Byte]

Tabulka 4.4: Paket LCP

Pole protokol nese hodnotu, která identifikuje tento protokol a patří jí číslo **0x8021**. Pole označené kód označuje hodnotu, který vypovídá typu dotazu nebo odpovědi. Pole identifikátor slouží k identifikaci jednotlivé otázky a její odpovědi. Délka paketu je součet ostatních polí, tedy kódu a identifikátoru a dat [25].

Kód	Název	Význam
1	Configure Request	Paket, který obsahuje požadavky na konfiguraci na změnu parametru linky.
2	Configure Ack	Paket nesoucí potvrzení požadavku. Všechny požadavky jsou přijaté.
3	Configure Nack	Paket obsahuje zamítnutí požadavku a informace s čím nesouhlasí.
4	Configure Reject	Paket obsahuje zamítnutí všech požadavků.
5	Terminate Request	Požadavek na ukončení spojení.
6	Terminate Ack	Paket obsahuje potvrzení ukončení spojení.
7	Code Reject	Odmítnutí požadavku, kvůli neznámému kódu.

Tabulka 4.5: Konfigurační kódy IPCP
[23]

Podobně jak to bylo u předchozího protokolu LCP, pole data pro IPCP protokol obsahuje konfigurační možnosti pro IPCP. Jednotlivé volby jsou řazeny sekvenčně za sebou [23].

Typ volby	Název
2	Protokol komprese (IPCP)
3	IP adresa
129	Primární DNS server
131	Sekundární DNS server

Tabulka 4.6: Konfigurační možnosti IPCP
[23]

Protokol Komprese (IPCP) Tato konfigurační volba obsahuje identifikátor volby, délku a data, kde se nacházejí parametry komprese [23].

Typ volby	Délka	Data
2	6 >=	0x02d

Tabulka 4.7: Konfigurační volba - Komprese dat

IP adresa Jedná se o konfigurační volbu, při které lze dynamicky nastavit konkrétní adresu pro protokol IP. Pokud druhá strana nesouhlasí, pošle preferovanou IP adresu v *Configure-Nak* [23].

Typ volby	Délka	Data
3	6	0.0.0.0

Tabulka 4.8: Konfigurační volba - IP adresa

Primární DNS server Tato konfigurační volba nastavuje primární jmenný server DNS [23].

Typ volby	Délka	Data
129	6	0.0.0.0

Tabulka 4.9: Konfigurační volba - Primární DNS server

Sekundární DNS server Tato konfigurační volba nastavuje sekundární jmenný server DNS [23].

Typ volby	Délka	Data
131	6	0.0.0.0

Tabulka 4.10: Konfigurační volba - Sekundární DNS server

4.3.5 Příklad navázaní spojení PPP

Uvedeme si příklad počátku komunikace mezi naším emulovaným modemem BG96 a řídicí jednotkou ESP32 Wemos R32. V počátku proběhne sekvence AT příkazů, kde emulátor odpovídá takovým způsobem, aby došlo k navázání PPP spojení. Pro jednoduchost si uvedené pouze počáteční a koncový AT příkaz.

Zařízení	Příkaz/Data	Popis
ESP32	AT <CR><LF>	Testování podpory AT příkazů
BG96	AT <CR><LF>	Očekávána odpověď pro řídicí jednotku
BG96	<CR><LF> OK <CR><LF>	Potvrzení úspěšnosti příkazu
ESP32	ATD*99***# <CR><LF>	Příkaz inicializujíce počátek PPP
BG96	CONNECT 115200 <CR><LF>	Potvrzení očekávaného přechodu na PPP a informace ohledně rychlosti přenosu
BG96	<CR><LF> OK <CR><LF>	Potvrzení úspěšnosti příkazu

Tabulka 4.11: Příklad navázaní spojení PPP

4.3.6 Příklad příchozího LCP paketu

Pokud se proces úspěšně dostal do počátku PPP komunikace ESP32 pošle HDLC rámec, který obsahuje PPP protokol, který používá pro počáteční domluvu protokol LCP.

Význam	HEX hodnota
Flag	0x7E
Adresa	0xFF
Control	0x03
LCP	0xC0 0x21
Kód, ID, Délka	0x7D 0x21 0x7D 0x21 0x7D 0x20 0x7D 0x34
Asynchronní kontrolní znaková mapa	0x7D 0x22 0x7D 0x26 0x7D 0x20 0x7D 0x2A 0x7D 0x20 0x7D 0x20
Magické číslo	0x7D 0x25 0x7D 0x26 0x31 0x2A 0x82 0x5A
Nastavení	0x7D 0x27 0x7D 0x22
CRC	0xAD 0x50
Flag	0x7E

Tabulka 4.12: Příklad příchozího LCP paketu

Je důležité zmínit, že pokud bychom chtěli pracovat s čistými daty v rámci protokolu LCP zabaleného v PPP, tak příchozí data jsou kódována, tak aby se zamezilo výskytu ukončovací sekvence. Tento pojem můžeme znát pod *bit stuffing* nebo pod *escaping*. 0x7D a 0x7H jsou ukončovací sekvence, pokud by se tato hodnota použila uprostřed paketu, tak by si zařízení nebo aplikace mohla myslet, že zde paket končí a došlo by ke ztrátě dat a chybě. Vycpávání paketů probíhá ve třech různých variantách.

1. Byte 0x7E se převede na 2 bytovou sekvenci 0x7D5E. Jedná se o ukončovací *flag byte*.
2. Byte 0x7D se převede na 2 bytovou sekvenci 0x7D5E. Jedná se o řešení *escape byte*
3. Byte menší než 0x20 se také převádí na 2 bytovou sekvenci. Například byte 0x01 se převádí na sekvenci 0x7D21.

Důvod pro převod u hodnoty menší než 0x20 je takový, že se jedná o ASCII kontrolní znaky, cílem je zabránit tomu, aby se tyto bajty objevily jako řídící znaky ASCII v sériovému ovladači na hostiteli či modemu, kde se tyto znaky interpretují speciálně. Lze také použít protokol pro nastavení viz. 4.3.2. Ve výchozím nastavení je všech 32 hodnot kódováno [7].

4.3.7 Příklad příchozího IPCP

Začátek vyjednávaní IPCP je podobné jako u LCP v tom, že každá strana najednou pošle svůj konfigurační požadavek druhé straně. Je zde použitý stejný stavový automat, který má odlišné stavy a průběh.

Význam	HEX hodnota
Flag	0x7E
PPP hlavička	0xFF 0x03
IPCP Configure Request, ID = 2	0x80 0x21 0x01 0x02 0x00 0x0A
Délka	0x18
IP adresa	0x03 0x06 0x00 0x00 0x00 0x00
Primární DNS server	0x81 0x06 0x00 0x00 0x00 0x00
Sekundární DNS server	0x83 0x06 0x00 0x00 0x00 0x00
CRC	0x25 0x56
Flag	0x7E

Tabulka 4.13: IPCP - úvodní dotaz

Z důvodu, že máme konfiguraci pro dynamickou IP adresou, tedy očekáváme, že nám nějakou IP adresu přiřadí. IP ve tvaru 0.0.0.0. V rámci naší aplikace se používá i DNS server, proto jsem ho uvedl jako příklad. v IPCP dotazu není vždy tento parametr nutný. Déle následuje další dotaz z druhé strany.

Význam	HEX hodnota
Flag	0x7E
PPP hlavička	0xFF 0x03
IPCP Configure Request, ID = 1	0x80 0x21 0x01 0x01 0x00 0x2A
Délka	0x18
IP adresa	0x03 0x06 0x7F 0x00 0x00 0x01
Primární DNS server	0x81 0x26 0x08 0x08 0x08 0x08
Sekundární DNS server	0x81 0x06 0x08 0x08 0x04 0x04
CRC	0x36 0x22
Flag	0x7E

Tabulka 4.14: IPCP - úvodní dotaz druhé strany

V tomto případě jsme poslali informaci o tom jakou IP adresu má přístupový bod. Můžeme začít vyjednávat. Dojde k schválení.

Význam	HEX hodnota
Flag	0x7E
PPP hlavička	0xFF 0x03
IPCP Configure Ack, ID = 1	0x80 0x21 0x02 0x01 0x00 0x0A
Délka	0x18
IP adresa	0x7d 0x23 0x06 0x7F 0x00 0x00 0x01
Primární DNS server	0x81 0x06 0x08 0x08 0x08 0x08
Sekundární DNS server	0x81 0x06 0x08 0x08 0x04 0x04
CRC	0x2F 0x57
Flag	0x7E

Tabulka 4.15: IPCP - potvrzení dotazu

Současně se zašle odmítnutí, které obsahuje IP adresu zařízení a IP adresy serverů.

Význam	HEX hodnota
Flag	0x7E
PPP hlavička	0xFF 0x03
IPCP Configure Nak, ID = 2	0x80 0x21 0x03 0x02 0x00 0x0A
Délka	0x18
IP adresa	0x7d 0x23 0x06 0x7F 0x00 0x00 0x01
Primární DNS server	0x81 0x06 0x08 0x08 0x08 0x08
Sekundární DNS server	0x81 0x06 0x08 0x08 0x04 0x04
CRC	0x2a 0x57
Flag	0x7E

Tabulka 4.16: IPCP - zamítnutí dotazu

Druhá strana dostala IP adresu, na přijetí reaguje konfiguračním dotazem.

Význam	HEX hodnota
Flag	0x7E
PPP hlavička	0xFF 0x03
IPCP Configure Request, ID = 3	0x80 0x21 0x01 0x03 0x00 0x0A
Délka	0x18
IP adresa	0x7d 0x23 0x06 0x7F 0x00 0x00 0x01
Primární DNS server	0x81 0x06 0x08 0x08 0x08 0x08
Sekundární DNS server	0x81 0x06 0x08 0x08 0x04 0x04
CRC	0x2F 0x57
Flag	0x7E

Tabulka 4.17: IPCP - konfigurační dotaz

Pro potvrzení shody, příjde konfigurační schválení.

Význam	HEX hodnota
Flag	0x7E
PPP hlavička	0xFF 0x03
IPCP Configure Ack, ID = 3	0x80 0x21 0x02 0x03 0x00 0x0A
Délka	0x18
IP adresa	0x7d 0x23 0x06 0x7F 0x00 0x00 0x01
Primární DNS server	0x81 0x06 0x08 0x08 0x08 0x08
Sekundární DNS server	0x81 0x06 0x08 0x08 0x04 0x04
CRC	0x2F 0x57
Flag	0x7E

Tabulka 4.18: IPCP - potvrzení dotazu

4.4 IP protokol

V předcházející sekci jsme si popsali protokol PPP, pokud bychom po vytvoření spojení chtěli přenášet data, tak na to nám poslouží rodina protokolů IP - Internet protocol. Linkové protokoly jako SLIP a PPP poskytují komunikaci mezi konkretními dvěma zařízeními po sériové lince. IP nám poskytne možnost komunikace skrze internet mezi dvěma zařízeními přes internet. Takový paket přidává IP hlavičku, která obsahuje především cílovou a zdrojovou IP adresu v síti. Taková hlavička má standardně 20 bytů, uvažujeme IPv4. V hlavičce kromě adres se ještě vyskytuje verze protokolu IP(IPv4, IPv6). Konkrétněji si IP hlavičku ukážeme v tabulce [3].

Verze IP	Délka	Typ služby	Celková délka
Identifikátor	Příznaky		Odsazení
Doba života(TTL)	Protokol vyšší vrstvy		Kontrolní součet
IP adresa odesílatele			
IP adresa příjemce			

Tabulka 4.19: Hlavička protokolu IP
[3]

Položka typ služby má velikost 1 byte. V historickém podtextu měla sloužit jako priorita. Celková délka obsahuje počet bytů v paketu. Jeho velikost nabývá hodnoty 2 bytů, což znamená, že velikost paketu může nabít maximální velikost 65 535 bytů. Identifikátor vkládá operační systém odesílatele. Délka života paketu(TTL) udává dobu života. Slouží k tomu aby nedošlo k nekonečnému zacyklení paketu v internetu. Každý směrovač dobu života snižuje o 1. Pokud je hodnota ve směrovači 0 paket je zahozen a vygeneruje se ICMP paket, který signalizuje odesílateli, že patřičný paket byl zahozen, má velikost 1 byte, takže jeho maximální hodnota je 255. Protokol vyšší vrstvy obsahuje identifikaci protokolu, který pracuje na vyšší vrstvě. Typicky se zde objevuje protokol TCP nebo UDP. Kontrolní součet obsahuje součet hlavičky ne celého paketu. Každý směrovač, pokud změní část hlavičky, musí dopočítat nový kontrolní součet. Jinak by paket byl identifikován jako neplatný a byl

by zahozen. Hlavičku můžou obsahovat volitelné položky, které se zpravidla nepoužívají, jelikož směrovače IP pakety s jinou velikostí hlavičky než 20 u protokolu IPv4 zahazují [3].

4.5 TCP

TCP - Transmition Control Protocol. Protokol pracující na 4. transportní vrstvě ISO/OSI modelu. Je definován v dokumentu RFC 793. Jedná se tedy o transportní protokol, paket na této vrstvě se nazývá segment. Při vytváří spojení probíhá navazování pomocí inicializačních zpráv. V hlavičce každého segmentu nebo blok segmentů je pořadové číslo. Každý úspěšně příchozí blok či segment je zpětně potvrzován příjemcem. Z tohoto důvodu se jedná o spolehlivý protokol. Nevýhoda takového protokolu spočívá ve vysoké režii, může tedy docházet k zahlcení sítě potvrzovacími segmenty. Výhodou protokolu je detekce nedoručení dat, jelikož nám nepřijde potvrzovací segment. Výhodou je také záruka správného pořadí, jelikož jednotlivé segmenty obsahují pořadové číslo. TCP využívají protokoly jako HTTPS, MQTT, FTP [4].

4.6 UDP

UDP - User Diagram Protocol. Protokol pracující na 4. transportní vrstvě ISO/OSI modelu. Je definován v dokumentu RFC 768. PDU se nazývá datagram. Protokol UDP, jak již bylo zmíněno, pracuje na transportní vrstvě oproti protokolu IP rozšiřuje paket o číslo portu. To znamená, že protokol IP slouží na propojení dvou bodů v internetu a UDP to upřesňuje na konkretní aplikaci. To znamená, že port adresuje konkretní bežící aplikaci. Úvodní komunikace se neustanovuje. Tento protokol nepoužívá pořadová čísla a ani nejsou zpětně potvrzována. Říkáme tedy, že tento protokol je nespolehlivý. Nevýhodou je nespolehlivost, nemáme žádnou záruku o úspěšném doručení dat. Výhodou je nízká režie čili menší zahlcení sítě. UDP využívá např. TFTP, VoIP [2].

4.7 Protokoly vyšších vrstev

V přecházející kapitole jsme si popsali transportní protokoly, pokud bychom se chtěli připojit na server, tak nám tyto protokoly nestačí. Pro připojení k serveru potřebujeme znát jeho IP adresu. K tomu se používá protokol DNS. Po připojení bychom chtěli komunikovat pro nějakou aplikaci. Jedním z velmi rozšířených aplikačních protokolů v rámci IoT je MQTT.

4.7.1 DNS

DNS - Domain Name Service. Jedná se o systém překladu doménových jmen na IP adresy. Jedná se o decentralizovanou databázi překladů. Klient požádá o překlad adresy svého primárního DNS serveru, pokud IP adresu zná pošle odpověď jinak se server dále doptává jiných DNS serverů. Výsledkem je odpověď v podobě IP adresy pro danou doménu [27].

4.7.2 MQTT

MQTT - Message Queing Telemtry Transport. Jedná se o jednoduchý, nenáročný protokol pro přenos zpráv. Kde jedna strana publikuje a druhá odebírá. Přenos dat probíhá pomocí TCP protokolu [21]. Tento protokol má na svědomí Andy Stanford-Clark, který v té době pracoval pod společností IBM a Arlen Nipper pracující pod firmou Eurotech. První verze

vznikla v roce 1999. Cílem bylo vytvořit protokol pro užší pásma, aby zařízení nemusela spotřebovávat tolik energie na napájení. MQTT ve většině případech funguje přes TCP/IP, ale pro senzorové sítě se využívá i protokol UDP. MQTT definuje dvě strany komunikace server (broker) a několik klientů (subscribers). MQTT broker je server, který přijímá všechny dotazy od klientů a poté směřuje odpovědi všem příslušným klientům. MQTT klient může být jakékoliv zařízení, které dokáže provozovat MQTT knihovnu a dokáže se připojit k serveru [10].

Kapitola 5

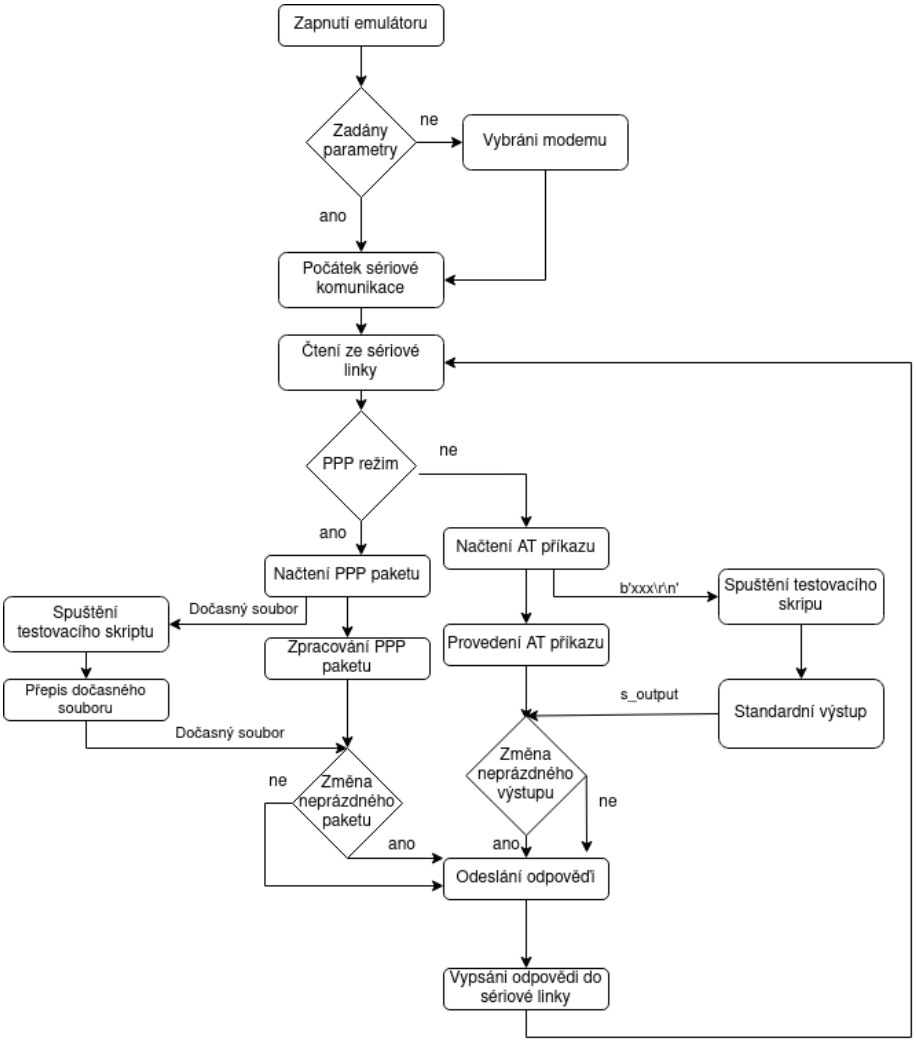
Návrh emulátoru modemů

5.1 Koncept emulátoru navrhovaného řešení

Emulátor bude napsaný v jazyce Python. Program bude orientovaný objektově pro snadnou údržbu a jednoduchou rozšiřitelnost. Emulátor by neměl využívat žádné specifické knihovny pro určitý operační systém, aby byl jednoduše přenositelný. Celý emulátor se bude moci nastavit pomocí parametrů nebo pomocí komunikačního dialogu. Emulátor bude zahrnovat možnost si vybrat konkrétní modem, kde bude implementována minimálně volba pro modem BG96 od firmy Quectel. Rozhraní by mělo být navrženo, tak aby rozšíření o další modem bylo triviální. Komunikace mezi mikrokontrolérem a emulovaným modemem bude probíhat přes sériovou linku. Rozhraní emulátoru a všechny jeho probíhající výstupy a vstupy budou zobrazeny v terminálu. Předpokládá se, že zařízení bude s emulátorem komunikovat výhradně přes sériovou linku. Jako parametr modulu bude soubor nebo absolutní cesta k souboru, která bude obsahovat cestu ke skriptu, který bude paralelně spouštěn s modemem, kde si uživatel bude moci pomocí jazyka Python definovat testovací scénář. Na vstupu ze sériové linky se budou očekávat na počátku příkazy AT, které daný modem podporuje. AT příkazy budou přicházet v binárním formátu přes sériovou linku. Po počátečním dialogu AT příkazů se přejde na komunikaci pomocí PPP protokolu.

5.2 Vývojový diagram

Diagram popisuje nejdůležitější prvky chodu programu a jeho větvení. Největší pozornost je v diagramu věnována části pro testovací scénář a jeho přístupu k režimu PPP a AT příkazům.

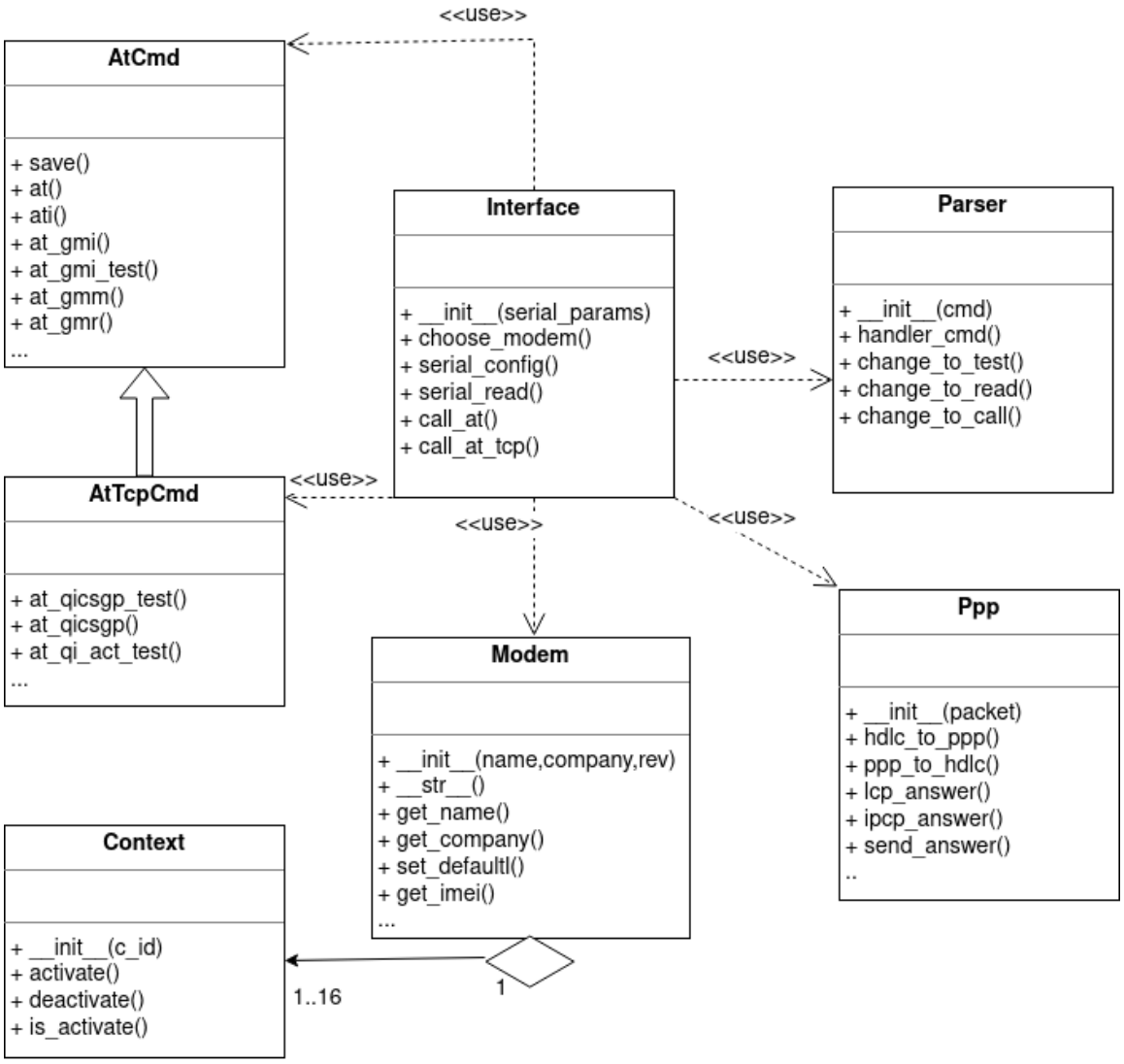


Obrázek 5.1: Obecný flowchart diagram emulátoru

Při spuštění aplikace bude potřeba si zvolit daný modem primárně bude nabídnut BG96. Rozhraní bude také umožňovat vložit absolutní cestu ke skriptu, kde bude sepsán testovací scénář. Poté se provede nastavení sériové linky se zařízením. Poté se čeká na příchozí vstup ze sériové linky, který obsahuje AT příkaz nebo paket PPP v binární formě. Pokud vstup byl AT příkaz, začne se provádět v emulátoru, současně se tento příkaz pošle na standardní vstup daného testovacího skriptu. Po provedení AT příkazu se výstup z emulovaného příkazu a výstup ze skriptu porovnají, pokud je rozdílný výstup skriptu, který je neprázdný, použije se tento výstup na standardní výstup emulátoru a vyšle odpověď skrz sériovou linku. Jinak se použije výstup emulovaného příkazu na standardní výstup a vyšle se odpověď sériovou linkou. Pokud příchozí data byla v binární formě PPP paketu, vloží se obsah paketu do dočasného souboru, který si otevře testovací skript. Po provedení testovacího skriptu se odpověď se vloží do dočasného souboru a jeho obsah si přečte emulátor, který porovná testovací odpověď s odpovědí emulovanou. Pokud je odpověď jiná, tak ji odešle na sériový výstup. Jinak použije emulovanou. Poté se přejde na čekání vstupu na sériové lince.

5.3 Diagram tříd

Diagram popisuje provázaní tříd a jejich závislosti. Za hlavní třídu, bych označil třídu **Interface**, která řídí celý proces emulace.



Obrázek 5.2: Diagram tříd emulátoru

Spouštění emulátoru bude lze vykonat pomocí *./emul.py*. Tento soubor neobsahuje žádnou třídu, je zde pouze zpracování parametrů a předání jako instancí pro třídu **Interface**. Pokud nedojde k předání žádných parametrů vytvoří se třída **Interface** s výchozím nastavením pro sériovou linku, při téhle skutečnosti probíhá komunikační dialog na doptání modemu, který se bude emulovat. Třída **Interface** slouží pro vstup a výstup programových částí. Současně se zde volají další třídy. Po příchodů AT příkazů se zavolá metoda na třídě **Parser**, která slouží k převodu AT příkazů, který je uložen jako řetězec. Příčina převodu spočívá ve skutečnosti, že AT příkazy obsahují znaky, které se v jazyce Python

nedají použít pro volání stejnojmenné funkce či metody. Například mezera v AT příkazu se nahrazuje podtržítkem. Také se zde zpracovávají argumenty příkazu, které jsou jednotlivě uloženy do pole. Po úspěšném převodu se volají upravené AT příkazy jako metoda třídy **AtCmd** nebo třídy **AtTcpCmd**. Tyto dvě výše zmíněné třídy mohou využívat třídu **Modem**, kde se ukládají aktuální data o modemu a jsou zde předdefinované výchozí hodnoty modemu. Třída **Modem** vytváří kontexty komunikace, každý kontext je vytvořen jako objekt **Context**. Kontextů může být maximálně šestnáct. Pokud na třídě **AtTcpCmd** je volána metoda, která reprezentuje AT příkaz, který nastavuje kontext komunikace, použije se třída **Modem**, která pomocí svých metod přenastaví kontext komunikace čili objekt **Context**. Pro komunikaci na úrovni PPP se z **Interface** využívá třída **Ppp**, která si vstup ze sériové linky vezme jako HDLC rámec a ten rozbalí a poté vykonává patřičné zpracování. Po zpracování paketu PPP a vytvoření jeho odpovědi se paket opět zabalí do HDLC rámce. Pokud přijde odpověď do třídy **Interface**, pošle se sériovou linkou a opět se čeká na příchozí data. Pokud dojde k ukončení PPP komunikace. přepne se řízení programu do stavu, kdy na vstupu očekává AT příkazy.

Kapitola 6

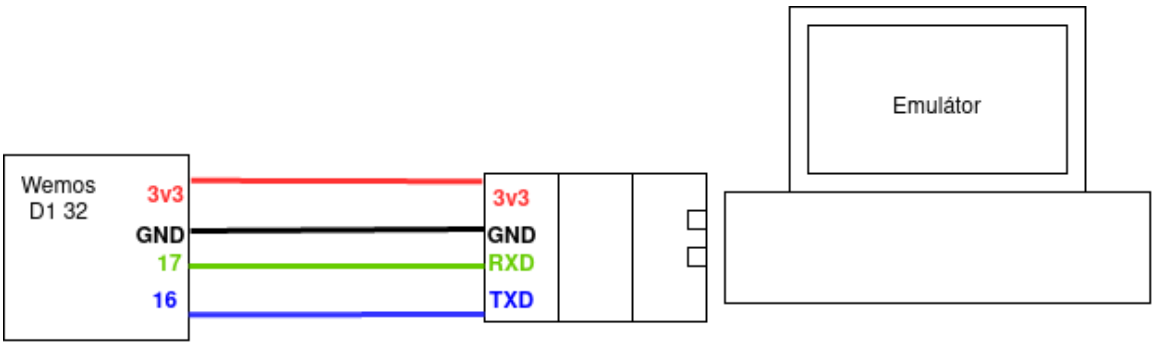
Implementace

6.1 Úvod

Pro snadnější rozšiřitelnost a udržovatelnost je žádoucí, aby samotná implementace oddělovala jednotlivé třídy do jednotlivých souborů. Pro otestování funkčnosti emulátoru je potřeba vytvořit prostředí ve kterém ho lze testovat. Také je důležité vytvořit prostředí ve kterém by se daný emulátor mohl používat. Tedy je potřeba připojit přes sériovou linku mikrokontrolér, který si popíšeme v další podkapitole.

6.1.1 Příprava řešení

Do počítače jsem tedy připojil externí TTL převodník konkrétně CP2102, který obsahuje piny pro napájení(3v3, GND), příjmu dat (RXD - Recieved) a přenosu dat (TXD - Transmited), pracovní status (DTR). Piny pro napájení a příjem jsem připojil na propojovací kablíky, které jsem napojil na mikrokontrolér R32 Wemos D1 R32.



Obrázek 6.1: Schéma testovacího prostředí

6.2 Použité technologie

6.2.1 Arduino IDE

Arduino IDE je open-source aplikace, která slouží k vývoji programu pro mikrokontroléry. Obsahuje i testový editor pro editaci a vývoj programu. Pomocí této aplikace, lze i takový program nahrát do mikrokontroléru. Patří mezi nejznámější zástupce takového vývojového prostředí. Jak už název napovídá toto vývojové prostředí není primárně určeno pro mikrokontroléry pro platformu ESP32. Přesto toto rozšířené vývojové prostředí nabízí podporu. Jak si uvedeme, tak podpora není zcela komplexní. [1] První kroky vedly tedy k frameworku Arduino IDE, kde jsem využíval knihovnu *TinyGSM* pro základní otestování jednotlivých AT příkazů. Poté jsem se pokoušel najít knihovnu, která mi pomůže s komunikaci na úrovni PPP. Bohužel jsem žádnou takovou knihovnu nenašel. Došel jsem tedy k závěru, že tento framework není pro moji aplikaci vhodný. [2]

6.2.2 ESP-IDF

Tedy Espressif IoT Development Framework. ESP-IDF je knihovna, která slouží pro vývoj mikrokontroléru především pro platformu ESP-32. Knihovna lze nainstalovat jako rozšíření pro Visual Studio Code. Prostředí ESP-IDF nabízí řadu nástrojů jako CMake nebo Ninja build tools. Tyto nástroje jsou určeny pro překlad a následné spouštění kódu. Toto prostředí obsahuje širokou škálu knihoven pro jednotlivé komponenty vývojové desky nebo pro speciální protokoly, jako v našem případě pro protokol PPP. Výhodou této knihovny jsou příklady pro danou knihovnu a užití. Já jsem zvolil příklad, který je pro komunikaci PPP. V příkladu na počátku proběhne série AT příkazů a poté se přejde na komunikaci PPP, kde dochází k úvodnímu nastavení parametrů a poté ke komunikaci se serverem MQTT. Po ukončení komunikace se přechází opět to režimu, kde se posílájí AT příkazy. [3] [4]

6.2.3 Knihovna ScaPy

Knihovna Scapy je multiplatformní nástroj, který zapouzdřuje řadu protokolů. Z velké části dokáže nahradit programy jako tcpdump nebo nmap. Slouží k manipulaci síťových paketů včetně odposlechu na nějakém rozhraním. Je psána pro jazyk Python, současná verze podporuje Python verze 2 a Python ve verzi 3. Výhodou je dostupnost zdrojových kódu na Githubu. Scapy má v sobě zabudovaný interaktivní interpret, který můžete využít pro jednoduché odzkoušení nebo pro práci v reálném čase. V mém případě knihovnu používám jako přídavný modul. Scapy využívám především na parsování paketů. V implementaci především pracuji s pakety PPP, které mohou obsahovat data, které představují další protokoly jako TCP 4.5, UDP 4.6, ICMP ,které dále mohou obsahovat protokoly z vyšších vrstev. V mém případě jsem se setkal s protokoly DNS a MQTT 4.7.2. [5]

[1] Převzaté z https://www.arduino.cc/en/software
[2] TinyGSM knihovna dostupná na https://github.com/vshymanskyy/TinyGSM
[3] Převzaté z https://docs.espressif.com/projects/esp-idf/en/latest/esp32/
[4] PPPOS dostupné na https://github.com/espressif/esp-idf/tree/master/examples/protocols
[5] https://scapy.net/

6.3 Koncepce

Z podstaty toho, že náš modul má fungovat jako emulátor modemu či modemů je důležité zajistit jednoduchou rozšiřitelnost. Současně je vhodné, aby vstupy a výstupy emulátoru byly viditelné, tak aby se uživatel mohl zorientovat, co se právě v emulaci děje. Nejjednodušším způsobem, jak toho lze docílit je posílat komunikaci na standardní výstup do terminálové konzole. Důvodem užití výpisů do terminálu je skutečnost, je že předpokládám, že budoucí uživatel bude pokročilý s informačními technologiemi a nebude pro něj přítěž mít takový výstup v konzoli. Snaha o jednoduchou rozšiřitelnost by zároveň měla jít ruku v ruce s přenosností programu čili modul by neměl být závislý na používaném operačním systému a neměl by využívat specifické programy a funkce operačního systému. Modul jsem vyvíjel na operačním systému Linux s distribucí Ubuntu 20.04. Dále rozšiřitelnost jsem plnil pomocí rozčlenění jednotlivých části modulu do tříd.

6.3.1 Seznam tříd

Interface

Implementace této třídy se věnuje zpracováním vstupu a výstupu programu a následné volání dalších tříd a kontroly jejich výstupů. Současně zde třída řídí testovací scénář, který je implementován jako pod proces programu.

AtCmd

Implementace třídy má jednoduchou koncepci v tom, že každý AT příkaz má svoji metodu. Zároveň bych chtěl upozornit, že se v programu žádný objekt této třídy nevytváří. Jednotlivé metody se volají na této třídě bez instance třídy. Toto řešení se může zdát jako netradiční, ale vzhledem k povaze úlohy je to vhodné řešení. Jednotlivé AT příkazy mohou mít několik použití jedná se o testovací, čtecí, zapisovací a příkaz pro spuštění. Příkazy pro testování a zápis mají svůj specifický sufix. Pro testování *test* pro zápis je zvolený sufix *read*.

AtTcpCmd

Implementace této třídy v podstatě kopíruje princip, který jsme si popsali u předchozí třídy. Rozdíl spočívá v jiných AT příkazech, čili metodách, které se věnují příkazům práci s TCP stackem.

Parser

Implemetace se věnuje převodu syntaxe příkazů do syntaxe takové, která se dá použít pro volání metody.AT příkazy mohou obsahovat znaky, které se v jazyce Python nedají použít pro volání funkce či metody. Jak popisuji ve třídě AtCmd, pro jednotlivé AT příkazy existují varianty použití, některé obsahují argumenty. Proto je důležité takový AT příkaz řádně zpracovat a připravit pro volání. Třída tedy zpracovává sériový vstup následovně. Vstup se předá metodě, která převede vstup na řetězec, který se následně oddělí na příkaz a jeho argumenty. Příkaz může obsahovat mezery nebo jiné znaky jako & nebo +, proto je nutné je převést na znak, který se dá použít pro volání. Já jsem ho převedl na podtržítko. Následně je potřebné přidat patřičný sufix. Syntaxe AT příkazu je popsána zde 3.2, za sufix se považují znaky za příkazem. Sufix se tedy odebere a nahradí se textovým sufixem, který vypovídá o funkci příkazu.

Do syntaxe argumentů příkazu se nijak nezasahuje pouze se oddělí od příkazu a uloží se do řetězce. Třída tedy vrací dvojici, která představuje metodu a argumenty.

Ppp

Implementace se věnuje zpracování komunikace PPP, která může obsahovat úvodní konfigurační dialog dále potom protokoly jako UDP, TCP, které mohou zapouzdřovat protokoly z vyšší vrstvy jako DNS nebo MQTT. Po vytvoření objektu, kde jako instance třídy je jednotlivý paket je paket zabalen v HDLC rámci, který popisuji zde 4.2. Po odstranění HDLC hlavičky jsou data escapované 4.3.6. Je tedy potřeba je převést na surové data. Po převedení se zjišťuje jaké protokoly paket PPP obsahuje. Podle daného protokolu se vytváří paket s reakcí. K výslednemu PPP paketu se přidá kontrolní součet neboli *FCS* a HDLC hlavička. Mohou nastat situace, kdy je potřeba zaslat druhé straně více paketů najednou v tomto případě se zabalí do pole. V rámci komunikace PPP byla implementována podpora pro protokoly TCP, UDP, DNS, MQTT.

Modem

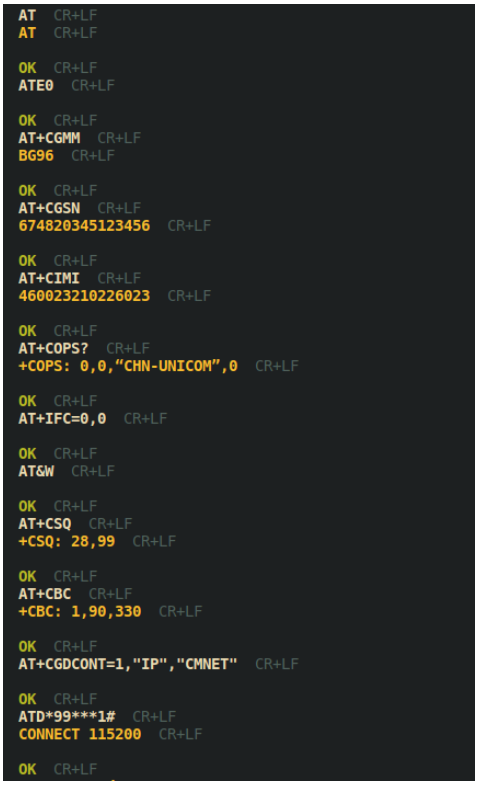
Implementace se věnuje třídě, která zaštiťuje konkrétní modem v dané emulaci v naší třídě se jedná o modem BG96, který může obsahovat spoustu dat, jako název, název výrobce nebo může obsahovat komunikační kontext, kterému se věnuje další třída.

Context

Implementace se věnuje jednotlivému kontextu připojení a jeho informacích jako je IP adresa či přihlašovací údaje, potenciál této třídy není zcela naplněn jelikož takový kontext se věnuje AT příkazům, které souvisí s práci se SIM kartou, této problematice se tato práce nevěnuje, ale pokud by se jednalo o rozšíření emulátoru, jeví se tato třída jako užitečná.

6.4 Popis fungování

Před spuštěním programu lze pomocí parametrů nastavit vlastnosti emulátoru, žádný z parametrů není povinný. Avšak emulátor pro svůj chod potřebuje znát daný modem, který bude emulovat. Pokud ho uživatel nezadá pomocí parametru. Proběhne komunikační dialog. Důležitým parametrem je cesta k testovacímu scénáři, který si podrobně popíšeme dále. Tedy uvažujeme o situaci, kdy jsme nezadali testovací scénář a chceme si jenom ověřit správné chování mikrokontroléru a jeho nahraného programu. Já konkrétně jsem pracoval s programem, který po sekvenci AT příkazu komunikuje v rámci PPP. V úvodu se tedy zpracovávají AT příkazy, jejich výstup v konzoli vypadá následovně.



Obrázek 6.2: Výpis sekvence AT příkazů

Vstupy ve formě AT příkazů jsou v bílé barvě. Odpověď se skládá ze dvou částí. Pro jednoduchost si je pojmenujeme jako odpověď a potvrzení správnosti. Odpověď je tedy v barvě bílé a potvrzení správnosti v barvě zelené. Pokud by AT příkaz nebyl implementován nebo by došlo k chybě, vypíše se chybová hláška, která se neodesílá přes sériovou linku, ale odešle se chybové potvrzení správnosti (ERROR), které v konzoli bude označené červeně. Na obrázku můžeme vidět příkaz *ATD*, který nám značí počátek komunikace PPP. Po tomto příkazu se očekává na vstupu programu komunikace pomocí PPP, která může trvat libovolně dlouhou dobu. Ukončovací sekvence PPP je *****. Po tomto příkazu program opět očekává na vstupu AT příkazy. V našem příkladu se mikrokontrolér pokusí náš emulátor restartovat pomocí příkazu *AT+QPOWD=1* načež odpovíme pouze *OK*. Poté se celý proces programu opakuje.

Testovací scénář

Jedním z důležitých faktorů emulátoru je vložení testovacího scénáře, který by měl být v podobě python skriptu. Daný testovací scénář se připojuje k emulátoru za pomocí parametrů před spouštěním emulátoru, kde se zadává jako absolutní cesta ke skriptu. Spuštění testovacího scénáře probíhá v rámci podprocesu emulátoru, kde se každý AT příkaz vloží na standardní vstup a poté se jeho výstup porovná s výstupem emulovaného příkazu. Pokud je z příkazu výstup neprázdný a rozdílný vůči emulovanému, vezme se na výsledný výstup, výstup ze skriptu, jinak se vezme emulovaný. Lze tedy, takto změnit výstup každého jednotlivého příkazu nebo změnit jeho reakční dobu. Průběh je implementovaný, tak že pokud změníme výstup AT příkazů na chybný, tak se to stane pouze pro první konkrétní příklad. Pokud by totiž docházelo k takovému případu vždy. Program by se zacyklil.

Pokud bychom chtěli nastavit takovéto zacyklení, je možné ho nastavit pomocí parametru - *-repeat* nebo *-r*. Testovací scénář lze použít i pro testování v rámci PPP komunikace. Jelikož binární forma PPP paketu nelze jednoduše posílat skrze standardní vstup, jako to bylo možné u AT příkazů. Ukládá se paket do dočasného souboru. V testovacím skriptu se přečte obsah souboru a zpracuje se odpověď. Odpověď se uloží do dočasného souboru, kde si ji emulátor přečte. Poté porovná testovací odpověď a emulovanou odpověď. Pokud jsou odpovědi rozdílné použije na výstup pro sériovou linku testovací odpověď. Jinak se použije emulovaná odpověď. Výhodou této implementace vidím především v tom, že pokud uživatel narazí na AT příkaz, který není v rámci emulátoru implementovaný, může si ho doplnit pomocí testovacího scénáře.

6.4.1 Ukázka použití

Pro použití emulátoru tedy potřebujeme mikrokontrolér, který je připojený skrze sériovou linku k zařízení, které dokáže spustit emulátor. Pro podrobnější popis fyzických komponent doporučuji předchozí podkapitolu. 6.1.1. Situace, kde budeme používat testovací scénář si popíšeme v další kapitole. Zde budu použití díla demonstrovat na příkladu, kde proběhne sekvence příkazu a poté se přejde do komunikace skrz PPP protokol a poté se znova přejde do příjmu AT příkazů. Po spuštění proběhne sekvence AT příkazů.

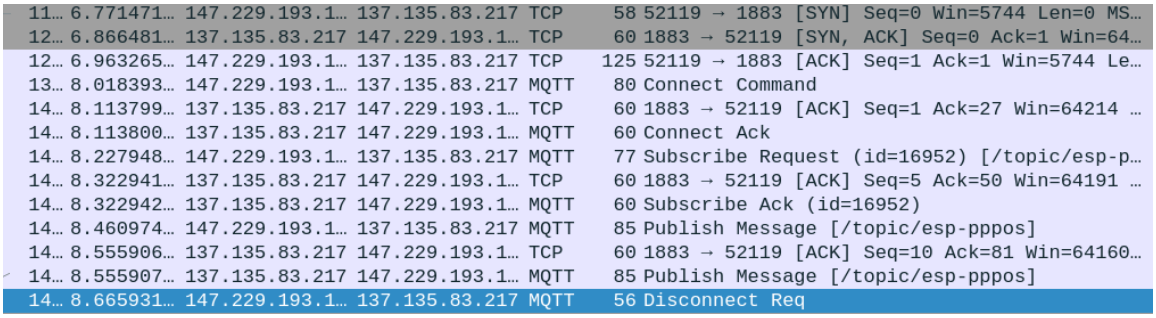
Během této komunikace nevypisuji žádné výstupy, pokud komunikace probíhá úspěšně. V počátku probíhá komunikační dialog protokolu LCP 4.3.1. Poté dialog pokračuje, ale v protokolu IPCP, ve kterém nastavuji IP adresu a DNS servery. IP adresu nastavím na IP adresu zařízení, kde běží emulátor. DNS servery nastavuji staticky na běžné hodnoty 8.8.8.8 a 8.8.4.4. Pro představu doporučuji 4.3.7. Dále proběhne DNS dotaz na MQTT server, který je nastavený v programu běžící na mikrokontroléru. Pro lepší představu přidávám záznam obrazovky z aplikace Wireshark.

854	7.336876…	147.229.193.1…	147.229.191.1…	DNS	98	Standard query 0x67ab A safebrowsing.googl…
855	7.337122…	147.229.193.1…	147.229.191.1…	DNS	98	Standard query 0x1e72 AAAA safebrowsing.go…
856	7.337266…	147.229.191.1…	147.229.193.1…	DNS	114	Standard query response 0x67ab A safebrows…
857	7.337402…	147.229.191.1…	147.229.193.1…	DNS	126	Standard query response 0x1e72 AAAA safebr…
30…	28.46365…	147.229.193.1…	8.8.8.8	DNS	83	Standard query 0x8cfc A mqtt.eclipseprojec…
30…	28.57618…	8.8.8.8	147.229.193.1…	DNS	99	Standard query response 0x8cfc A mqtt.ecli…

Obrázek 6.3: DNS dotaz

Záznam zobrazuje komunikaci mezi počítačem a sítí. Ukazuje průběh rekurzivního DNS dotazu, který končí odpovědí, která obsahuje IP adresu na dotazovaný server.

Po úspěšném DNS se naváže spojení s daným MQTT serverem. Pro lepší představu přidávám záznam obrazovky z aplikace Wireshark.



Obrázek 6.4: MQTT komunikace

Prvně proběhne navázání spojení pomocí protokolu TCP. Poté proběhne krátká výměna dat v protokolu MQTT, jelikož program nahraný v mikrokonroléru slouží jako příklad.

6.5 Využití testovacího scénáře

Jeden z hlavních cílů této práce je umožnit uživateli simulovat různé druhy chyb mezi AT příkazy, které v reálném světě mohou nastat za delší časový úsek. Umožníme tedy uživateli otestovat svůj mikrokontrolér a odhalit chyby, které by mohly nastat až při nasazení.

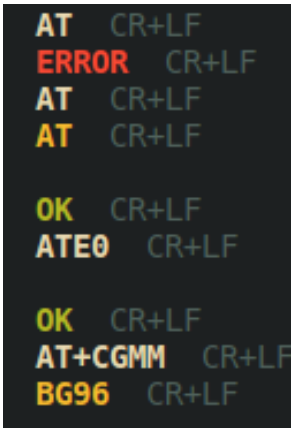
6.5.1 Příklad použití testovacího scénáře

V předešlých kapitolách jsem popisoval, že testovací skript bere na vstupu sériový výstup. Na vstupu skriptu se tedy objeví konkrétní AT příkaz a je pouze na uživateli, jakým způsobem bude tento skript strukturovat. Pro představu zde vložím jednoduchý příklad takového skriptu.

```
def test(cmd):
    if(cmd == 'AT\r'):
        print("ERROR")
    else:
        print("")
if __name__ == '__main__':
    cmd = sys.argv[1]
    test(cmd)
```

Obrázek 6.5: Jednoduchý testovací scénář

Tento skript jsem spouštěl bez vstupního argumentu *–repeat* v důsledku toho, se nám testovacího scénář provedl pouze jednou. Následně si ukážeme, jak tento testovací scénář měl vliv na průběh emulace.



Obrázek 6.6: Výstup po testování

6.5.2 Metodika testovacích scénářů

Metodika mých testů je poměrně prostá. Budu měnit pomocí skriptů výstupy emulátoru tak, abychom mikrokontrolér donutili k nějaké neočekávané akci čili se pokusíme najít vnitřní chybu. Pomocí skriptů je také možné měnit i odezvu příkazu. Z počátku budu testovat změnu jen jednoho příkazu, kde budu zkoušet jednotlivé scénáře. Poté budu testovat kombinaci změn u různých příkazů.

6.5.3 Jednotkové testovací scénáře

Příkaz AT+CGMM

V testovacím scénáři jsem měnil výstupy příkazu *AT+CGMM*. Správný průběh je výpis názvu modemu v našem případě **BG96**. Následným potvrzením **OK**.

Název modelu	Potvrzení	Opakovaně	Neočekávané chování	Počet pokusů
BG97	OK		žádné	5
SIM900	OK		žádné	5
445454554542	OK		žádné	5
BG96	ERROR	ano	žádné	5

Tabulka 6.1: Příkaz AT+CGMM

Pole pro název modelu představuje hodnotu, která se měnila pro tento scénář a která se poslala do mikrokontroléru. Pole potvrzení je hodnota, která se rovněž posílala sériovou linkou. Pole opakovaně představuje jestli se daná situace cyklicky opakovala v emulátoru, pro potvrzení s hodnotou OK nemá smysl. Jedná se o parametr, který nastavuje v emulátoru. Pole pro neočekávané chování označuje neočekávanou změnu, kterou způsobil testovací scénář.

Příkaz ATE0

V testovacím scénáři jsem měnil výstupy pro příkaz *ATE0*. Odpověď je pouze jako potvrzení ve tvaru **OK**.

Potvrzení	Opakovaně	Neočekávané chování	Počet pokusů
ERROR	ne	ano	5
ERROR	ano	žádné	5

Tabulka 6.2: Testovací scénář ATEO

Očekávané chování mikrokontroléru na zamítnutí tohoto příkazu je dotaz pomocí příkazu *AT*, kde se zeptá jestli zařízení podporuje příkazy AT a po potvrzení podpory by se měla sekvence opakovat.

Obrázek 6.7: Výstup pro neopakovanou chybu ATE0

Na obrázku můžeme vidět příkaz chybovou odpověď, kterou jsme způsobili našim testovacím skriptem reakcí na příkaz *ATE0*. Dále jsme tedy očekávali příchozí příkaz *AT*, kde došlo k anomálii a sériová linka příjmula pouze příkaz *T*, který emulátor nezná a proto ho označil za vadný. V následujicím kroku, ale dojde k úspěšnému přijmutí příkazu *AT* a program pokračuje bez chyby. Tuto anomálii nelze označit za chybu, jelikož nedošlo k porušení chodu programu. Zároveň podobnost písmena *T* a příkazu *AT* je evidentní. Z toho usuzuji, že mohlo dojít k chybě čtení ze sériové linky v rámci emulátoru.

Příkaz AT&W

V testovacím scénaři jsem měnil výstupy pro příkaz *AT&W*. Odpověď je pouze jako potvrzení příkazu ve správném případě **OK**.

Potvrzení	Opakovaně	Neočekávané chování	Počet pokusů
\n\r OK\n\r		žádné	5
\n\r\n\r OK \n\r\n\r		žádné	5
\n\r ERROR \n\r	ne	žádné	5
\n\r ERROR \n\r	ano	žádné	5
\n\r\n\r ERROR \n\r\n\r	ne	žádné	5
\n\r\n\r ERROR \n\r\n\r	ano	žádné	5

Tabulka 6.3: Testovací scénář AT&W

Pro tento příkaz se neobjevilo žádné neočekávané chování. Přijmutí potvrzení s přidanými bílými znaky bylo bezproblémové. Odmítnutí v podobě **ERROR** bylo také bezproblémové. Po odmítnutí se vždy předchozí sekvence AT příkazů zopakovala podle očekávaní. V případě, že se odmítnutí opakuje v cyklu, sekvence se rovněž opakovala podle očekávaní.

Příkaz AT+CIMI

V tomto testovacím scénáři jsem měnil výstup pro příkaz **AT+CIMI**, který očekává hodnotu *IMSI*, což je hodnota, která jednoznačně identifikuje zařízení. Poté následuje hodnota pro potvrzení.

Hodnota IMSI	Potvrzení	Opakovaně	Neočekávané chování	Počet pokusů
456465.5756	OK		žádné	5
+CME ERROR: 0		ne	žádné	5
+CME ERROR: 3		ne	žádné	5
+CME ERROR: 4		ne	žádné	5
+CME ERROR: 0		ano	žádné	5
+CME ERROR: 3		ano	žádné	5
+CME ERROR: 4		ano	žádné	5

Tabulka 6.4: Testovací scénář pro AT+CIMI

V první sérii testů jsem se snažil vložit zvláštní hodnotu IMSI, která by potencionálně mohla vyvolat chybu. Na průběh programu to nemělo žádný vliv. V dalších testech jsem odpovídal chybovou hláškou. Chybová hláška 0 značí selhání zařízení. Chybová hláška 3 značí operaci, která není povolena. Chybová hláška 4 značí operaci, která není podporována. Po odeslání chybové hlášky se předchozí sekvence AT příkazů zopakovala a pokud byl testovací scénář nastaven pouze pro první průběh program bezproblémově pokračoval dále. Pokud byl testovací scénář nastaven na cyklické opakování, tak se sekvence příkazů opakovala. Lze tedy říci, že nedošlo k žádnému neočekávanému chování.

Příkaz ATD*99***1#

V tomto testovacím scénáři jsem měnil výstup pro příkaz **Příkaz ATD*99***1#**, který očekává zprávu ve tvaru **CONNECT 115200**, poté potvrzení ve tvaru **OK**.

Zpráva připojení	Potvrzení	Opakovaně	Neočekávané chování	Počet pokusů
CONNECT 9600	OK		žádné	5
CONNECT 256000	OK		žádné	5
	NO CARRIER	ne	ano	5

Tabulka 6.5: Testovací scénář pro ATD

V prvních dvou sériích testů jsem měnil zprávu připojení. Na chod programu takové změny však neměly žádný vliv na chod aplikace. V posledním testu jsem na místě potvrzení vracel hodnotu **NO CARRIER**. Výsledky tohoto testovacího scénáře byly rozhodně příkladem pro neočekávané chování. Po odeslání tohoto chybové potvrzení se mikrokontrolér odmlčel a přestal komunikovat s modemem. Očekávané chování bylo přepnutí do režimu AT a vyslání příkazu **ATE0**.

Testovací scénář v PPP

Zatím jsem simuloval chybu pouze u AT příkazů. Po úspěšné sekvenci AT příkazů, začne komunikace probíhat v PPP. V tomto testovacím scénáři jsem zasáhl do této binární komunikace a snažil jsem se vyvolat chybu. V testovacím skriptu jsem otevřel soubor obsahující binární formu PPP paketu a zpracoval pomocí metody emulátoru, která nám binární data převede do formátu, ve kterém ho lze zpracovat pomocí knihovny ScaPy 6.2.3. Po zpracování jsem využil metodu emulátoru pro převedení do správného binárního formátu a uložil do dočasného souboru. Výsledky tohoto testu popisuje následující tabulka.

Protokol	Typ dotazu	Odpověď	počet opakování	Neočekávané chování
LCP	Configure-Request	Discard packet	3	ne

Tabulka 6.6: Testovací scénář PPP

V testu jsem zachytil PPP paket, který obsahoval protokol LCP 4.3.1. Na dotaz ohledně konfigurace jsem reagoval zahozením paketu. Po zahození se dotaz vždy opakoval a po správné odpovědi program úspěšně pokračoval.

6.5.4 Komplexní testovací scénáře

Zatím jsem se snažil simulovat chybu vždy u jednoho konkrétního příkazu. Zde popíšu scénáře, kde se chyby objeví u více příkazů najednou.Poslední testovací scénář se bude věnovat kombinaci chyby několika AT příkazů a chyby pro PPP režim.

Komplexní testovací scénář 1.

V tomto scénáři testuji situaci pro chybovou odpověď u více AT příkazů. Následuje tabulka použitých příkazů a změn jejich vstupů.

AT příkaz	Odpověď	Potvrzení odpovědi	Neočekávané chování
AT	zpozděná odpověď 350[ms]	OK	ne
AT&W		ERROR	ne
ATE0		ERROR	ne
AT+CGMM		ERROR	ne

Tabulka 6.7: Testovací scénář 1.

V rámci tohoto testovacího scénáře proběhlo pět pokusů. Každý pokus dopadl stejně. Chování programu mělo vždy stejný průběh a to takový, že po chybném potvrzení odpovědi, proběhla znovu předchozí sekvence AT příkazů, jelikož jsem tento scénář spouštěli bez opakování. Závěrem k tomuto scénáři bych chtěl upozornit na skutečnost, že u příkazů **AT** jsem zpozdil odpověď o hodnotu větší než je maximální reakční doba u tohoto příkazu, která je 300 [ms]. Program i přes takovéto zpoždění na odpověď reaguje a posílá následující AT příkazy.

Komplexní testovací scénář 2.

Předchozí scénář se věnoval pouze situaci, kdy se jednotlivé testovací odpovědi vloží pouze jednou. V tomto scénáři jsem otestoval situaci, pro stejné AT příkazy jako v předchozím příkladě s rozdílem, že počet opakování zadá uživatel na každém příkazu zvlášť. Následuje tabulka popisující tento test.

AT příkaz	Odpověď	Potvrzení odpovědi	Počet opakování	Neočekávané chování
AT	AT	OK	3	ne
AT&W		ERROR	2	ne
ATE0		ERROR	5	ne
AT+CGMM	BG96	ERROR	2	ne

Tabulka 6.8: Testovací scénář 2.

V rámci tohoto testu proběhlo 5 opakování, kdy každý test dopadl stejně. Po vypsání chyby se vždy sekvence opakovala, po dovršení počtu opakování u jednotlivého příkazu program pokračoval dále.

Komplexní testovací scénář 3.

Předchozí scénář se věnoval pouze situaci, kdy se jednotlivé testovací odpovědi věnují pouze AT příkazům. V tomto testovacím scénáři jsem testoval AT příkazy i PPP komunikaci. Pro představu zde vložím příklad takového skriptu.

```
import sys
from scapy.all import *
sys.path.insert(1, '../src')
from ppp import *

def test(cmd):
    if(cmd == 'AT\r'):
        print("AT")
        print("ERROR")
        print(10)
    elif(cmd == 'AT&W\r'):
        print("ERROR")
        print(4)
    elif(cmd == 'ATE0\r' ):
        print("ERROR")
        print(7)

    elif(cmd.startswith('AT') == False):
        name_file = cmd
        try:
            f = open(name_file,'rb+')
        except:
            return
        data = f.read()
        f.seek(0)
        f.truncate()
        p = Ppp(data)
        p.hdlc_to_ppp()

        if(p.packet[PPP].proto == 0xc021):
            p.packet[PPP].code = 12

        p.ppp_to_hdlc()
        f.write(p.packet)
        f.close()
        print(3)
    else:
        print("")

if __name__ == '__main__':
    cmd = sys.argv[1]
    test(cmd)
```

Obrázek 6.8: Testovací scénář pro AT příkazy a PPP

Testovací scénář je vložen do funkce *test()*, která jako argument přebírá parametr, ze standardního vstupu. Pokud se jedná o AT příkaz, tak obsahuje prefix *AT*. Při kladném vyhodnocení AT příkazů u větvení se odpovědi odesílají pomocí funkce *print()*. Pokud chce uživatel určit počet opakování pro konkrétní případ, napíše tento počet opakování do funkce *print()*. Jestliže se jedná o binární komunikaci je v argumentu název souboru, kde je uložen surový PPP paket. Pro práci s takovým paketem je vhodné využít metody třídy *Ppp*, které uživateli umožní zpracovávat paket pomocí knihovny ScaPy a poté ho převést do surové binární podoby. Počet opakování lze nastavit pomocí funkce *print()*.

První část se věnovala testovacím vstupům pro AT příkazy, její výsledky jsou popsané následující tabulkou.

AT příkaz	Odpověď	Potvrzení odpovědi	Počet opakování	Neočekávané chování
AT	AT	OK	10	ne
AT&W		ERROR	4	ne
ATE0		ERROR	7	ne

Tabulka 6.9: Testovací scénář 3. část AT

Během této části scénáře nedošlo k žádnému neočekávanému chování. Testovací scénář mohl přejít na část věnované PPP komunikaci.

Protokol	Typ dotazu	Odpověď	počet opakování	Neočekávané chování
LCP	Configure-Request	12	3	ne

Tabulka 6.10: Testovací scénář 3. část PPP

V této části jsem na dotaz protokolu LCP odpovídal neznámým kódem. V protokolu LCP se nachází 11 kódů 4.3.1, čili taková hodnota v protokolu LCP nemá žádný význam. Program reagoval na neznámý kód tak, že zopakoval vždy předchozí dotaz. Poté program úspěšně pokračoval.

Kapitola 7

Závěr

Cílem této práce bylo vytvořit emulátor pro modem BG96 a implementovat podporu pro protokol PPP, ve kterém lze automaticky testovat kód bežící na mikrokontroléru. Tento záměr byl splněn. Při řešení práce jsem strávil přespříliš času ve vývojovém prostředí Arduino, které se ukázalo jako nevhodné pro mou implementaci. Během implementace mi největším časovým oříškem byl protokol PPP a jeho vnitřní protokoly i protokoly vyšších vrstev jako DNS nebo TCP. V rámci této práce se mi nakonec povedlo implementovat emulátor pro modem BG96, kde jednotlivé AT příkazy jsou zaznamenávané v terminálu. Podporu pro PPP protokol jsem řešil svědomitě a snažil jsem dodržet standardy RFC. Jako jeden z největších přínosů vidím, že se jedná o poměrně ojedinělé řešení takového problému. Další přínos vidím ve skutečnosti, že pomocí testovacího scénáře se dají doplnit jednotlivé AT příkazy, které mohou uživateli chybět. Jako jeden z úspěchu považuji skutečnost, že se mi povedlo pomocí jednoho testovacího scénáře vložit takovou odpověď pro mikrokontrolér, s kterou si nevěděl rady a dále se nechoval podle očekávání. Další rozšíření práce by mohlo spočívat v implementaci dalších modemů a v rozšíření o grafické rozhraní, kde by uživatel nemusel přistupovat k příkazové řádce. Práce mi poskytla prostředí pro zlepšení svých schopností v programování. Zároveň mi pomohla si odzkoušet řadu technologií s kterými jsem neměl žádnou zkušenost. Jako největší přínos beru skutečnost, že během realizace jsem si prošel celým síťovým modelem od nastavení sériové linky po komunikaci na vzdáleném MQTT serveru.

Literatura

[1] *What Is LTE-M? Long Term Evolution for Machines Explained*. Www.emnify.com: Emnify, 13.4.2021. Dostupné z: https://www.emnify.com/blog/lte-m.

[2] *User Datagram Protocol* [RFC 768]. RFC Editor, srpen 1980. DOI: 10.17487/RFC0768. Dostupné z: https://www.rfc-editor.org/info/rfc768.

[3] *Internet Protocol* [RFC 791]. RFC Editor, září 1981. DOI: 10.17487/RFC0791. Dostupné z: https://www.rfc-editor.org/info/rfc791.

[4] *Transmission Control Protocol* [RFC 793]. RFC Editor, září 1981. DOI: 10.17487/RFC0793. Dostupné z: https://www.rfc-editor.org/info/rfc793.

[5] *Nonstandard for transmission of IP datagrams over serial lines: SLIP* [RFC 1055]. RFC Editor, červen 1988. DOI: 10.17487/RFC1055. Dostupné z: https://www.rfc-editor.org/info/rfc1055.

[6] *Compressing TCP/IP Headers for Low-Speed Serial Links* [RFC 1144]. RFC Editor, únor 1990. DOI: 10.17487/RFC1144. Dostupné z: https://www.rfc-editor.org/info/rfc1144.

[7] *2.6 PPP: Point-to-Point Protocol*. Flylib.com: Flylib.com, 2007. Dostupné z: https://flylib.com/books/en/3.223.1.37/1/.

[8] *The Complete History of the Modem*. https://history-computer.com/: History Computer, 2015. Dostupné z: https://history-computer.com/modem-complete-history-of-the-modem/.

[9] *Standardization of NB-IOT completed*. https://www.3gpp.org/: 3GPP, 2016. Dostupné z: https://www.3gpp.org/news-events/3gpp-news/1785-nb_iot_complete.

[10] *MQTT Version 5.0*. https://www.oasis-open.org/: Oasis-open, 2019. Dostupné z: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[11] *SIM800*. https://www.simcom.com/: SIMCom, 2020. Dostupné z: https://www.simcom.com/product/SIM800.html.

[12] *Základní informace o LTE*. Www.digi.ctu.cz: Lte ctu, 2020. Dostupné z: https://web.archive.org/web/20200219072002/https://digi.ctu.cz/lte-rk/olte.

[13] *Narrowband – Internet of Things (NB-IoT)*. https://www.gsma.com/: GSMA, 2021. Dostupné z: https://www.gsma.com/iot/narrow-band-internet-of-things-nb-iot/.

[14] *EM7455 M2M Module*. https://www.sierrawireless.com/: Sierra Wireless, 2022. Dostupné z: https://www.sierrawireless.com/iot-solutions/products/em7455/.

[15] *LPWA BG95 Cat M1/Cat NB2/EGPRS series*. https://www.quectel.com/: Quectel, 2022. Dostupné z: https://www.quectel.com/product/lpwa-bg95-cat-m1-cat-nb2-egprs-series.

[16] *LPWA BG96 Cat M1/NB1/EGPRS*. https://www.quectel.com/: Quectel, 2022. Dostupné z: https://www.quectel.com/product/lpwa-bg96-cat-m1-nb1-egprs.

[17] *LTE-M – the new GSM*. Www.rohde-schwarz.com: Rohde-schwarz, 2022. Dostupné z: https://www.rohde-schwarz.com/us/solutions/test-and-measurement/wireless-communication/iot-m2m/lte-m/lte-m-theme_234034.html.

[18] *NRF9160*. https://www.nordicsemi.com/: Nordic Semiconductor, 2022. Dostupné z: https://www.nordicsemi.com/Products/nRF9160.

[19] Hlavatý, J. *Priame porovnanie IoT technológií dostupných v ČR*. Praha, 2018. Bakalářská práce. ČVUT. Dostupné z: https://dspace.cvut.cz/bitstream/handle/10467/76452/F3-BP-2018-Hlavaty-Jergus-BachelorFinal.pdf.

[20] Hobby, R. a Perkins, D. D. *Point-to-Point Protocol (PPP) initial configuration options* [RFC 1172]. RFC Editor, červenec 1990. DOI: 10.17487/RFC1172. Dostupné z: https://www.rfc-editor.org/info/rfc1172.

[21] Kiedroň, K. *Rozšíření MQTT brokeru o archivaci a vizualizaci dat*. Ostrava, 2018. Diplomová práce. Vysoká škola báňská - Technická univerzita Ostrava. Dostupné z: http://hdl.handle.net/10084/128447.

[22] Matyáš, B. J. *Možnosti potenciálu nově dostupných služeb v sítích Long Term Evolution*. Zlín, 2010. Diplomová práce. Univerzita Tomáše Bati. Dostupné z: https://digilib.k.utb.cz/bitstream/handle/10563/14423/matyáš_2010_dp.pdf.

[23] McGregor, G. *The PPP Internet Protocol Control Protocol (IPCP)* [RFC 1332]. RFC Editor, květen 1992. DOI: 10.17487/RFC1332. Dostupné z: https://www.rfc-editor.org/info/rfc1332.

[24] Melík, F. *Knihovna pro ovládání GSM modulu mikrořadiči ATmega*. Liberec, 2012. Bakalářská práce. TECHNICKÁ UNIVERZITA V LIBERCI. Dostupné z: https://dspace.tul.cz/bitstream/handle/15240/49107/V_11812_Mb.pdf.

[25] Simpson, W. A. *PPP Authentication Protocols* [RFC 1334]. RFC Editor, říjen 1992. DOI: 10.17487/RFC1334. Dostupné z: https://www.rfc-editor.org/info/rfc1334.

[26] Simpson, W. A. *The Point-to-Point Protocol (PPP) for the Transmission of Multi-protocol Datagrams over Point-to-Point Links* [RFC 1331]. RFC Editor, květen 1992. DOI: 10.17487/RFC1331. Dostupné z: https://www.rfc-editor.org/info/rfc1331.

[27] Sotorník, L. *Vizualizace DNS dotazu*. Olomouc, 2019. Bakalářská práce. Univerzita Palackého v Olomouci, Přírodovědecká fakulta, Katedra Informatiky. Dostupné z: https://theses.cz/id/ioeamg/33396800.

[28] Spěvák, V. *Komunikační rozhraní modemů GSM*. Brno, 2018. Bakalařská práce. Masarykova univerzita. Dostupné z: https://is.muni.cz/th/fkpn8/Komunikacni_rozhrani_modemu_GSM.pdf.

[29] Čepl, R. *Použití mobilních zařízení při výuce odborných předmětů*. Brno, 2018. Bakalářská práce. Masarykova univerzita. Dostupné z: https://is.muni.cz/th/k369f/Bakalarska_prace_Roman_Cepl_447936__3_.pdf.

Příloha A

Obsah přiloženého paměťového média

- `latex_src/` - zdrojé soubory LaTeX
- `src/` - zdrojové soubory - emulátor
- `examples/` - zdrojové soubory - testovací scénáře
- `docs/` - dokumentace
- `xsechr00.pdf` - text práce ve formátu PDF
- `README.md` - informace o použití emulátoru

Příloha B

Modemy v IoT

Quectel 5G NR Module Family

Jedná se o nejnovější produkty, které vyrábí firma Quactel, tyto modemy jsou určeny pro 5G sítě. Moduly v této rodině se značí prefixem RG nebo RM. Moduly se vyskytují ve verzi **EA**,**EU**,**GL** a **AE**.
EA je konfigurace pro Evropu,Asii a Pacifik
EU je konfigutace pouze pro Evropu,Asii, Pacifik s Brazílii.
GL je konfigurace pro celý svět.
AE je konfigurace pro celý svět, kromě Číny.

Modem	AT příkazy	3GPP TS27.005	3GPP TS27.007	3GPP TS27.010
RG50xQ	ano	x	ano	x
RM5xxQ	ano	x	ano	x
RG52ON	ano	x	ano	x
RG52OF	ano	x	ano	x
RM52ON	ano	x	ano	x
RM53ON	ano	x	ano	x

Quectel LTE-A Module Family

Modemy z rodiny LTE-A což představuje název pro 4G sítě.

Modem	AT příkazy	3GPP TS27.005	3GPP TS27.007	3GPP TS27.010
EP06	ano	x	ano	x
EG06	ano	x	ano	x
EG060V	ano	x	ano	x
EG12	ano	x	ano	x
EG120K	ano	x	ano	x
EG18	ano	x	ano	x
EG512	ano	x	ano	x

Quectel M.2, Module Family

Modemy z rodiny M.2 je označení, který znamená kategorii síti 4G.

Modem	AT příkazy	3GPP TS27.005	3GPP TS27.007	3GPP TS27.010
EM05	ano	ano	ano	x
EM06	ano	ano	ano	x
EM060K	ano	ano	ano	x
EM12	ano	x	ano	x
EM120K	ano	x	ano	x
EM120R	ano	x	ano	x
EM121R	ano	x	ano	x

Quectel LTE, Module Family

Modem	AT příkazy	3GPP TS27.005	3GPP TS27.007	3GPP TS27.010
EC21/EC21 Mini PCIe	ano	x	x	x
C25/EC25 Mini PCIe	ano	x	x	x
EG21-G/ EG21-G Mini PCIe	ano	x	x	x
G25-G/ EG25-G Mini PCIe	ano	x	x	x
EG91	ano	x	x	x
EG95	ano	x	x	x
EC200S	ano	x	x	x
EC200A	ano	x	x	x
EC200U	ano	x	x	x
EG915U	ano	x	x	x

Quectel LPWA, Module Family

Modem	AT příkazy	3GPP TS27.005	3GPP TS27.007	3GPP TS27.010
BG96	ano	ano	ano	x
EC200S	ano	x	x	x
BG95	ano	x	x	x
BG600L-M3	ano	x	x	x
BG77	ano	x	x	x
BG770A	ano	x	x	x
BC95-G	ano	x	ano	x
BC68	ano	x	ano	x

Quectel GSM/GPRS, Module Family

Modem	AT příkazy	3GPP TS27.005	3GPP TS27.007	3GPP TS27.010
M66	ano	ano	ano	x
M65	ano	ano	ano	x
M08-R	ano	ano	ano	x
M95	ano	x	x	x
M95-R	ano	x	x	x
MC60	ano	x	x	x
MC60E	ano	x	x	x
MC90	ano	x	x	x

B.0.1 U-blox

LTE-M, NB-IoT

Série modemů	AT příkazy	3GPP TS27.005	3GPP TS27.007	3GPP TS27.010
ALEX-R	ano	x	x	x
UBX-R5	ano	x	x	x
SARA-R5	ano	ano	ano	ano
SARA-N2/N3	ano	x	x	x

CAT-1

Série modemů	AT příkazy	3GPP TS27.005	3GPP TS27.007	3GPP TS27.010
LARA-R2	ano	ano	ano	x
TOBY-R2	ano	ano	ano	x

4G

Série modemů	AT příkazy	3GPP TS27.005	3GPP TS27.007	3GPP TS27.010
TOBY-L2	ano	ano	ano	x
MPCI-L2	ano	ano	ano	x

3G

Série modemů	AT příkazy	3GPP TS27.005	3GPP TS27.007	3GPP TS27.010
SARA-U2	ano	ano	ano	ano
LISA-U2	ano	ano	ano	ano

2G

Série modemů	AT příkazy	3GPP TS27.005	3GPP TS27.007	3GPP TS27.010
SARA-G	ano	ano	ano	ano
SARA-G3	ano	ano	ano	ano

B.0.2 Simcom

5G

Modem	AT příkazy	3GPP TS27.005	3GPP TS27.007	3GPP TS27.010
SIM8210C	ano	x	x	x
SIM8210C-M2	ano	x	x	x
SIM8202X-M2	ano	x	x	x
SIM8300G-M2	ano	x	x	x
SIM8200G	ano	x	x	x
SIM8200EA-M2	ano	x	x	x

LPWA

Modem	AT příkazy	3GPP TS27.005	3GPP TS27.007	3GPP TS27.010
SIM7022	ano	x	x	x
SIM7090G	ano	x	x	x
SIM7070E	ano	x	x	x
SIM7080G	ano	x	x	x
SIM7070G	ano	x	x	x
SIM7060R	ano	x	x	x
SIM7060G	ano	x	x	x
SIM7000X	ano	x	x	x
SSIM7000E-N	ano	x	x	x

4G

Série modemů	AT příkazy	3GPP TS27.005	3GPP TS27.007	3GPP TS27.010
A76xx	ano	x	x	x

Modem	AT příkazy	3GPP TS27.005	3GPP TS27.007	3GPP TS27.010
SIM7500	ano	x	x	x
SIM7600	ano	x	x	x

LTE-A

Série modemů	AT příkazy	3GPP TS27.005	3GPP TS27.007	3GPP TS27.010
A7906	ano	x	x	x
SIM79xx	ano	x	x	x

3G

Modem	AT příkazy	3GPP TS27.005	3GPP TS27.007	3GPP TS27.010
A5360E	ano	x	x	x
SIM5320X	ano	x	x	x
SIM5310	ano	x	x	x

2G

Modem	AT příkazy	3GPP TS27.005	3GPP TS27.007	3GPP TS27.010
SIM800	ano	ano	ano	x

B.0.3 Sierra wireless

5G mmWave a Sub-6

Modem	AT příkazy	3GPP TS27.005	3GPP TS27.007	3GPP TS27.010
EM9190	ano	x	x	x
EM9190	ano	x	x	x

4G LTE Advanced Pro(Cat-12+)

Modem	AT příkazy	3GPP TS27.005	3GPP TS27.007	3GPP TS27.010
EM7690	ano	x	x	x
EM7565	ano	x	x	x
EM7511	ano	x	x	x

4G LTE Advanced (Cat-6 Cat-7)

Modem	AT příkazy	3GPP TS27.005	3GPP TS27.007	3GPP TS27.010
EM7411	ano	x	x	x
EM7421	ano	x	x	x
EM7431	ano	x	x	x
EM7430	ano	x	x	x
EM7455	ano	x	x	x

MC série (Mini Card)

Série modemů	AT příkazy	3GPP TS27.005	3GPP TS27.007	3GPP TS27.010
MC74x1	ano	x	x	x

Mini Card Accessory Board

Série modemů	AT příkazy	3GPP TS27.005	3GPP TS27.007	3GPP TS27.010
WP76xx	ano	x	x	x

B.0.4 Nordic

Modem	AT příkazy	3GPP TS27.005	3GPP TS27.007	3GPP TS27.010
nRF9160	ano	ano	ano	x

B.0.5 Murata

LTE Cat. M1

Modem	AT příkazy	3GPP TS27.005	3GPP TS27.007	3GPP TS27.010
LBAD0XX1SC	ano	x	ano	x
LBAD00XX1WG	ano	x	ano	x
LBAD0ZZ1SE	ano	x	x	x

NB-IoT

Modem	AT příkazy	3GPP TS27.005	3GPP TS27.007	3GPP TS27.010
LBAD0ZZ1SE	ano	x	x	x
LBAD0YW1SS	ano	x	x	x
LBAD0XX1SC	ano	x	ano	x
LBAD0XX1WG	ano	x	ano	x