FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Analýza dat ze senzorů s využitím SoC ESP32
Student:	Bc. Pavel Zajíc
Vedoucí:	Ing. Peter Macejko
Studijní program:	Informatika
Studijní obor:	Návrh a programování vestavných systémů
Katedra:	Katedra číslicového návrhu
Platnost zadání:	Do konce zimního semestru 2019/20

Pokyny pro vypracování

Analyzujte možnosti připojení a zpracování dat ze senzorů (mikrofon, ...) pomocí čipu ESP32. Zaměřte se na možnosti přenosu těchto dat pomocí vestavěné WiFi a sériové linky. Porovnejte tyto možnosti se starším čipem ESP8266. Ověřte možnosti přenosu dat v praxi na obou čipech (ESP8266 a ESP32). Prozkoumejte možnosti předzpracování těchto dat přímo v rámci samotného čipu, případně s využitím pomocných výpočetních jednotek.

Seznam odborné literatury

Dodá vedoucí práce.

doc. Ing. Hana Kubátová, CSc.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 15. března 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Analýza dat ze senzorů s využitím SoC ESP32

Bc. Pavel Zajíc

Katedra Číslicového návrhu
Vedoucí práce: Ing. Peter Macejko

7. ledna 2019

Poděkování

Rád bych poděkoval především mému vedoucímu práce, panu Ing. Peteru Macejkovi za jeho velkou ochotu mi vždy pomoci při řešení problémů a za to, že mě vždy dokázal nasměrovat správným směrem. Dále bych rád poděkoval své rodině a přátelům, kteří mě vždy podporovali a nepochybovali, že tuto diplomovou práci úspěšně dokončím.

Prohlášení

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

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

V Praze dne 7. ledna 2019 .....

Odkaz na tuto práci

Zajíc, Pavel. *Analýza dat ze senzorů s využitím SoC ESP32*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Tato diplomová práce se zabývá analýzou připojení a zpracování dat ze senzorů pomocí čipu ESP32 a zejména analýzou možností přenosu dat pomocí vestavěné WiFi a sériové linky a předzpracování dat ze senzorů (mikrofonu) v rámci samotného čipu. Následně jsou tyto možnosti porovnány se starším čipem ESP8266. Možnosti přenosu dat jsou v rámci této práce prakticky ověřeny pomocí aplikací v jazyce C pro oba moduly. Součástí programového vybavení je také aplikace tunelu UART-WiFi pro oba moduly napsána v jazyce C, která přijímá data ze sériové linky, vytváří z těchto dat UDP pakety a následně odesílá tyto pakety pomocí WiFi na zařízení v síti. Tento tunel je následně analyzován a je stanoven limit propustnosti pro bezchybný přenos společně s určením chybovosti při překročení tohoto limitu. Nakonec je srovnána výkonnost obou modulů v praxi. Zdrojové kódy aplikací je možné nalézt na přiloženém CD této práce.

Klíčová slova ESP32, ESP8266, analýza možností přenosu dat, UART, WiFi, tunel UART-WiFi, propustnost, jazyk C

Abstract

This diploma thesis deals with the analysis of connection and processing sensoric data using ESP32 chip and espeially with analysis of data transfer possibilities using embedded WiFi and serial link. Analysis of a sensoric data preprocessing within the chip ESP32 alone is also included. This analysis is followed by a comparison of these possibilities with the older chip ESP8266. The data transfer possibilities is in terms of this thesis verified in practice using applications in C programming language for both modules. Application for both modules in C programming language of a tunnel UART-WiFi is also included in the software. This application receives data from serial link, forms them into 1KB UDP packets and sends these packets over WiFi to other device in the same network. This tunnel is then analyzed and the limit of a throughput for data transfer with no errors is determined as well as the error rate when the limit is exceeded. In the end the performance of both modules in practice is compared. Source codes of applications can be found in the attachment of this work.

Keywords ESP32, ESP8266, analysis of data transfer possibilities, UART, WiFi, tunnel UART-WiFi, throughput, C programming language

Obsah

Seznam obrázků

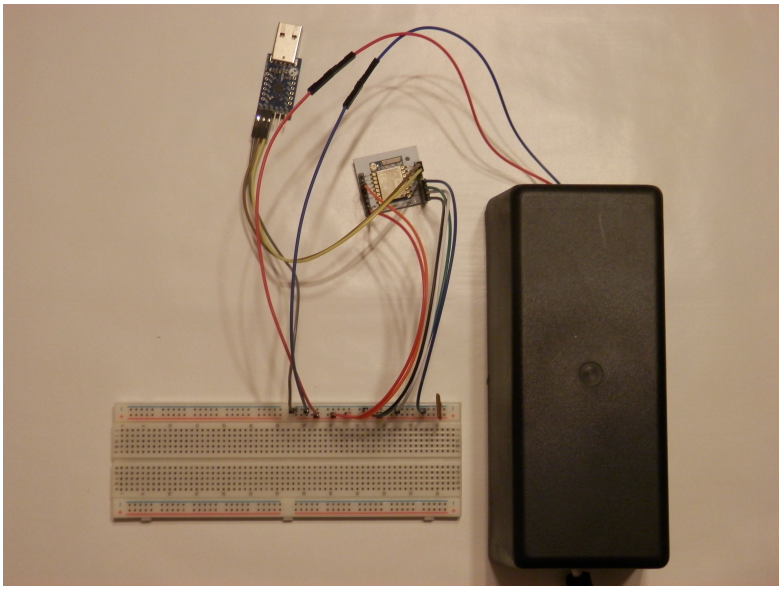
Seznam tabulek

Úvod

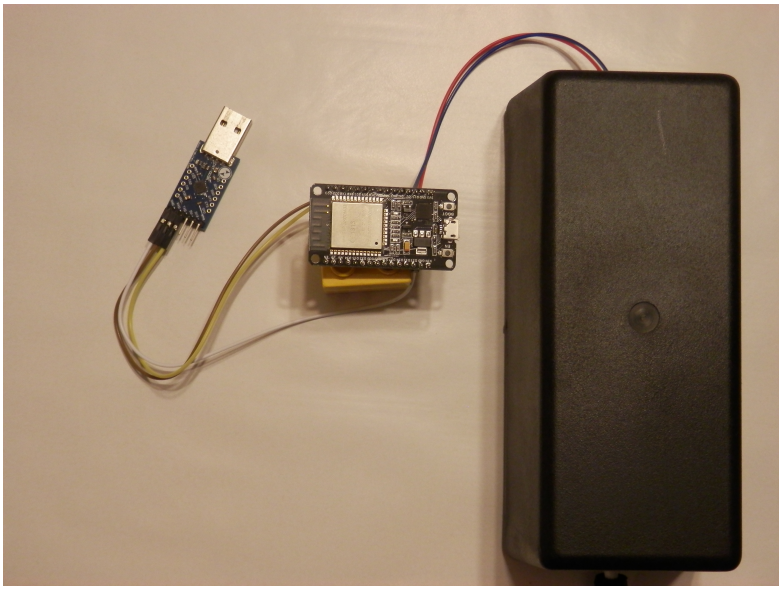
Dnešní svět informatiky se stále více rozrůstá a inženýři z celého světa přinášejí každým rokem čím dál tím více poznatků. Výjimkou není ani sféra internetu věcí, které čím dál tím více vzrůstá popularita. Díky této popularitě se dnes už s internetem věcí setkáváme prakticky na každém kroku, ať už se jedná o inteligentní hodinky nebo pouze o vzdálenou regulaci teploty bytu přes internet.

V mé diplomové práci se zabývám analýzou možností připojení a zpracování dat ze senzorů pomocí čipu ESP32 a zejména pak analýzou možností přenosu dat pomocí vestavěné sériové linky a WiFi. Součástí této analýzy je také prozkoumání možností předzpracování dat ze senzorů (mirkofonu) v rámci samotného čipu a následné porovnání těchto možností a výkonnosti se starší verzí čipu ESP8266. Dále také představím praktické ověření možností přenosu dat na obou čipech. Tato práce vytvoří základní podklady pro projekt Fakulty elektrotechnické ČVUT, který má za cíl testovat přenos audio dat v nekomprimované formě pomocí WiFi modulů.

Nejprve se v první části práce věnuji obecným specifikacím modulů ESP32 a ESP8266, jejich výkonnostním možnostem, možnostem připojení senzorů a možnostem zpracování dat ze senzorů v rámci čipu a dále pak také rozhraní UART, standardům WiFi spjatých s oběma moduly a teoretickým porovnáním obou modulů. Ve druhé části popisuji návrh zaspojení pro oba moduly včetně návrhu programů pro praktické ověření vlastností samotného rozhraní UART modulů, vestavěné WiFi a také návrhu programu tunelu UART-WiFi, který bude důležitou součástí projektu pro Fakultu elektrotechnickou ČVUT. Tento program přijímá data ze sériové linky, následně z nich vytváří UDP pakety a tyto pakety poté odesílá pomocí WiFi na příslušné zařízení v síti. Ve třetí části představím realizaci těchto programů. Ve čtvrté části se již věnuji testování a praktické analýze možností přenosu modulů ESP32 a ESP8266 společně se stanovením limitů pro propustnost sériové linky a tunelu UART-WiFi a chybovosti při překročení těchto limitů. V této části také stanovím praktickou propustnost WiFi obou modulů a další vlastnosti. Nakonec srov-



Obrázek 0.1: Zapojení modulu ESP8266



Obrázek 0.2: Zapojení modulu ESP32

návám praktický výkon obou modulů. Jako ukázku uvádím zapojení modulu ESP8266 na obrázku 0.1 a na obrázku 0.2 zapojení modulu ESP32.

Kapitola 1

Cíl práce

Cílem literární rešerše této práce je nastudování specifikací a technických parametrů čipů ESP32 a ESP8266 společně s analýzou možností připojení a zpracování dat ze senzorů. Dále analyzuji možnosti předzpracování dat ze senzorů (mikrofonu) v rámci samotného čipu ESP32. Následuje pak analýza možností přenosu dat pomocí vestavěné WiFi a sériové linky a také porovnání technických možností obou modulů.

Cílem praktické části je pak návrh a implementace programů, které pomohou otestovat možnosti přenosu dat pomocí vestavěné WiFi a sériové linky na modulech ESP8266 a ESP32. Součástí této části je také aplikace tunelu UART-WiFi pro oba moduly, který má za úkol přijímat data ze sériové linky, následně z těchto dat formovat UDP pakety a poté vysílat tyto pakety pomocí WiFi. Součástí testování tohoto programu je identifikace limitů propustnosti pro celý tunel, při kterých je možný bezchybný přenos a dále také analýza chybovosti při překročení těchto limitů a nakonec porovnání výkonnosti obou modulů v praxi.

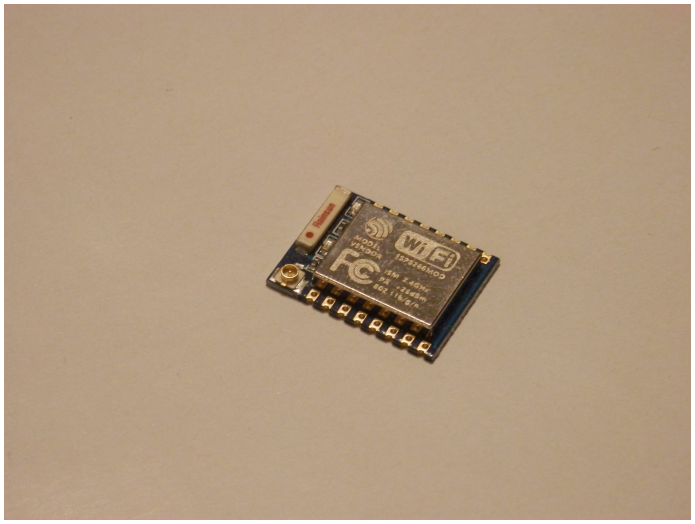
Kapitola 2

Analýza

Pro pochopení současného stavu problematiky je důležité čtenáře obeznámit s architekturou SoC ESP8266, architekturou modulu ESP32, s vlastnostmi těchto modulů a s jejich vzájemnými rozdíly. Dále je také důležité seznámit se se standardy Wi-fi a jejich vlastnostmi. Nedílnou součástí jsou také možnosti a vlastnosti sériové linky, která je jedním z hlavních předmětů analýzy této diplomové práce.

2.1 Wi-fi modul ESP8266

ESP8266 je SoC od firmy Espressif Systems. Modul obsahuje nízkopříkonový 32 bitový mikrokontrolér a podporuje hodinovou frekvenci 80 a 160 MHz. Modul má dále integrovanou Wi-fi anténu s podporou standardů IEEE 802.11b,



Obrázek 2.1: Reálná podoba modulu ESP-07

IEEE 802.11g a IEEE 802.11n. Modul lze využít k vytvoření samostatné sítě nebo ho lze využít jako další zařízení v existující síti. Jeho reálnou podobu si můžete prohlédnout na obrázku 2.1.

ESP8266 je vysoce integrovaný bezdrátový systém na čipu navržený pro přenosné platformy s nároky na omezenou plochu a spotřebu. Nabízí řadu možností, jak využít schopností Wi-fi v různých systémech. Modul může také fungovat jako samostatná aplikace s minimálními nároky na cenu a plochu.

Tento modul může sloužit také jako Wi-fi adaptér, což umožňuje začlenění bezdrátového přístupu k počítačové sítí, případně na internet, do jakéhokoliv návrhu obsahující mikrokontorlér. Propojení takového návrhu a modulu ESP8266 je jednoduše zprostředkováno pomocí sériových rozhraní jako je například SPI, UART nebo I^2C.

Kromě výše uvedených rozhraní zařízení disponuje 10 bitovým A/D převodníkem, integrovaným TCP/IP zásobníkem, integorvaným fázovým závěsem, různými regulátory a jednotkami na řízení spotřeby, pulzně šířkovou modulací nebo i rozhraním I^2S.

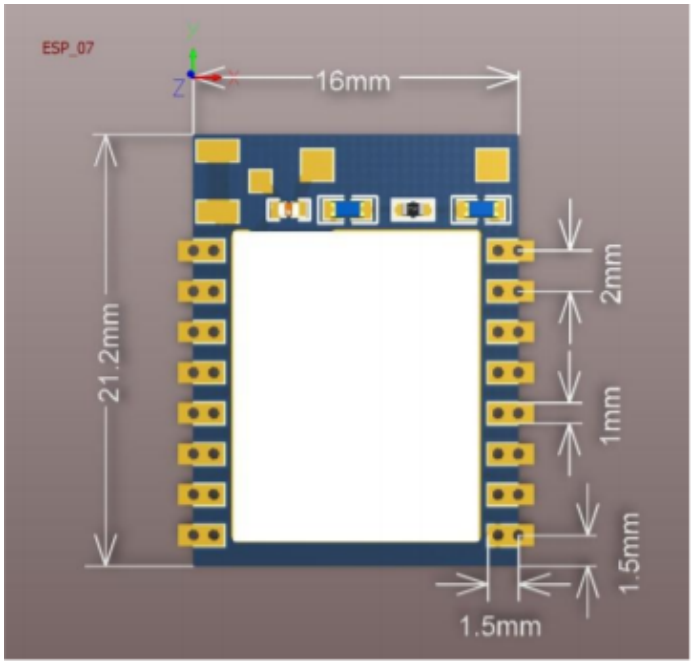
Dále je důležité zmínit, že provozní napájecí napětí se pohybuje mezi 3 a 3,6V a frekvence Wi-fi mezi 2,4 a 2,5GHz. Pro Wi-fi je podporováno zabezpečení WPA a WPA2 a šifrování AES nebo například WEP.

Pro práci se sítí pak můžeme využít síťové protokoly TCP, UDP, HTTP nebo FTP s podporou IPv4. Jiné protokoly pro tento modul nejsou podporovány.

Zařízení obsahuje celkem 16 pinů. Je zde samozřejmě pin pro napájecí napětí, uzemnění a reset. Dále pak také pin pro vysílač sériové linky TX0, přijímač sériové linky RX0, pin A/D převodníku ADC v měřítku 0-1024, pin CH-PD (chip power-down), který značí, jestli s modulem pracujeme v normální módu nebo v úsporném režimu. Nakonec je zde 9 pinů pro všeobecné využití (GPIO piny). Na obrázku 2.2 je znázroněn modul ESP8266, kde je možné vidět rozmístění všech pinů popsaných v tomto odstavci. Rád bych ještě podotknul, že zde uvádím informace spjaté s ESP-07, což je způsob zapojení ESP8266 a dalších podpůrných obvodů. Toto zapojení a označení ESP-07 bylo zhotoveno týmem AI-Thinker.

v tabulce 2.1 je očíslování jednotlivých pinů, které jsou vidět na obrázku (pin číslo 1 je v levém horním rohu modulu a pin číslo 16 je v pravém horním rohu modulu), dále název pinů a jejich funkce.

Další vlastnosti tohoto modulu přímo nesouvisí s mou diplomovou prací, a proto bych čtenáře rád odkázal na příslušný dokument, kde jsou popsány veškeré další detaily, a který můžete nalézt na [1]. V tomto dokumentu můžete nalézt také použité obrázky v této práci a také tabulku v originálním anglickém znění. Funkce některých pinů zde záměrně nepopisuji do detailů, protože pro



Obrázek 2.2: Rozložení pinů modulu ESP-07

číslo pinu	název pinu	funkce
1	RST	Resetuje modul
2	ADC	A/D převodník. Vstupní napětí 0-1 V, měřítko: 0-1024
3	CH_PD	1) logická úroveň 1: normální provoz; 2) logická úroveň 0: úsporný režim
4	GPIO16	GPIO16; Může být použit k probuzení modulu z hlubokého sleep módu
5	GPIO14	GPIO14; HSPI_CLK
6	GPIO12	GPIO12; HSPI_MISO
7	GPIO13	GPIO13; HSPI_MOSI; UART0_CTS
8	VCC	Zdroj napětí 3,3V (VDD)
9	GND	GND
10	GPIO15	GPIO15; MTDO; HSPICS; UART0_RTS
11	GPIO2	GPIO2; UART1_TXD
12	GPIO0	GPIO0
13	GPIO4	GPIO4
14	GPIO5	GPIO5
15	RXD0	UART0_RXD; GPIO3
16	TXD0	UART0_TXD; GPIO1

Tabulka 2.1: Popis jednotlivých pinů modulu ESP-07

tuto práci jsou nepodstatné. Detailní popis je však také uveden ve stejném dokumentu.

2.2 Wi-fi modul ESP32

Modul ESP32 je stejně jako ESP8266 vyvinutý firmou Espressif Systems s ultra-low-power 40 nm technologií. Tento modul kombinuje Wi-Fi s frekvencí 2,4 GHz a bluetooth v rámci jednoho čipu. Modul je navržen tak, aby dosáhl co možná největší robustnosti, přizpůsobivosti, spolehlivosti a vysokého výkonu pro radiotechniku. ESP32 je navržen hlavně pro přenosné aplikace a aplikace Internetu věcí. Dobře se ale také hodí pro návrh takzvané nositelné elektroniky.

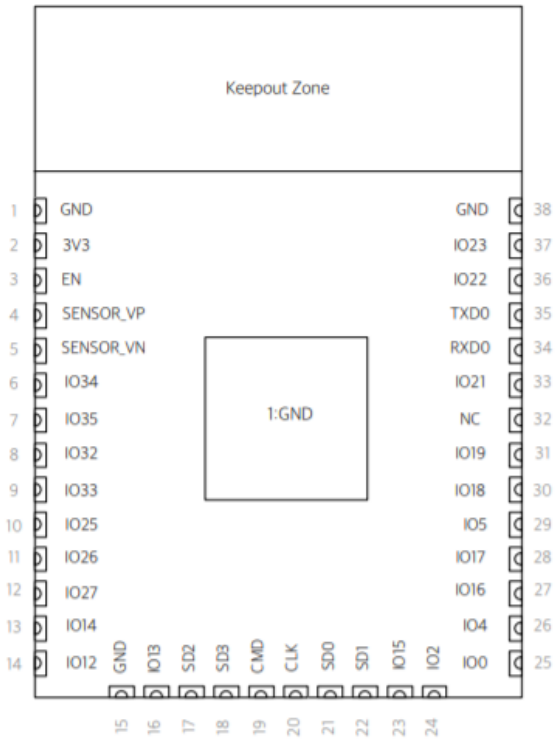
Série ESP32 zahrnuje čipy ESP32-D0WDQ6, ESP32-D0WD, ESP32-D2WD a ESP32-S0WD. v mé diplomové práci pracuji pouze s čipem ESP32-D0WDQ6, a proto se následující část textu může v některých detailech lišit od ostatních typů čipu.

Čip ESP32-D0WDQ6 je navržen tak, aby byl škálovatelný a adaptivní. Tento čip obsahuje dvě procesorová jádra, která lze řídit samostatně. Frekvence jednotlivých jader je nastavitelná a to od 80 MHz do 240 MHz. Uživatel má také možnost odpojit procesor od napájení a využít tak nízkopříkonového co-procesoru k pozorování změn na periferiích či pozorování překročení daných prahových hodnot.

ESP32 integruje bohatou škálu periférií jako například rozhraní pro SD karty, vysokorychlostní SPI, 3 rozhraní UART, rozhraní I^2S a I^2C. Modul disponuje celkem 38 piny. Kromě pinů pro napájení a uzemění je důležité zmínit piny RXD0 a TXD0, které představují přijímač a vysílač rozhraní UART0, dále na pinech GPIO16 respektive GPIO17 nalezneme přijímač respektive vysílač rozhraní UART2. Přijímač respektive vysílač rozhraní UART1 se nachází na pinech SHD/SD2 respektive SWP/SD3. Tyto 2 piny společně s piny SCK/CLK, SDO/SD0, SDI/SD1 a SCS/CMD jsou připojeny k SPI flash paměti integrované na modulu, a proto není doporučeno je používat k jiným účelům. Pokud bychom tedy chtěli využít piny rozhraní UART1, musíme je namapovat na jiné volné GPIO piny. Detailní rozložení všech pinů modulu je znázorněno na obrázku 2.3.

Integrace Bluetooth, Bluetooth LE (Typ Low Energy. Zajišťuje menší spotřebu napájení a cenu, přičemž nabízí podobný komunikační rozsah jako původní Bluetooth) a WiFi poskytuje celou řadu možností pro aplikace. Vestavěná WiFi nabízí velký fyzický rozsah a přímé připojení k počítačové síti či k internetu, zatímco použití Bluetooth umožňuje uživateli připojení k mobilnímu telefonu nebo vysílat signály pro detekci zařízení.

Zvolený operační systém pro ESP32 je freeRTOS s LwIP (Lightweight IP). Také je podporováno bezpečné (šifrované) OTA (over the air) nahrávání firmware, aby vývojář mohl snadno nahrát novou verzi firmware na zařízení i po jeho uvedení do provozu. Více detailů je dostupných na [2].



Obrázek 2.3: Rozložení pinů modulu ESP32

2.2.1 Možnosti připojení senzorů

Modul ESP32 poskytuje mnoho možností pro připojení senzorů. v této sekci se budu věnovat těm nejpodstatnějším z nich. Naopak zde nebudu uvažovat například IR řadič, pulsně šířkovou modulaci, řadiče SD karet a podobně, protože tato diplomová práce má pomoci zejména projektu zabývajícím se audio daty.

První a nejdůležitější možností pro připojení seznorů jsou samozřejmě GPIO piny. ESP32 má celkem 34 GPIO pinů, které mohou být využity pro různé účely. Modul má několik druhů GPIO pinů jako například piny, které jsou pouze digitální. Dále pak piny, které mají povolený analogový vstup/výstup. Druhý zmíněný typ pinů může být konfigurován také jako digitální. Při volbě pinů pro práci je nutné vzít v úvahu to, že GPIO piny 6-11 nelze využít, jelikož jsou napojené na SPI flash modulu ESP32.

Druhou možnost, jak pracovat se senzory, poskytuje SPI. Je to jeden ze standardů sériové periferní sběrnice, která slouží k propojení a sériové komunikaci obvodů. Sběrnice má celkem 4 vodiče: SCLK (sériové hodiny), SS (slave select), MOSI (master out slave in) a MISO (master in slave out). SPI na ESP32 má celkem 3 rozhraní: SPI, HSPI a VSPI. Písmena H a V jsou zde jen jako rozlišovací znaky. Všechna rozhraní mohou pracovat v master a slave

módu. SPI poskytuje 4 přenosové formáty, které závisí na polaritě (CPOL) a fázi (CPHA) SPI hodin. Detailnější popis pro účel této práce nepovažuji za důležitý, a proto bych čtenáře rád odkázal na přednášku Ing. Miroslava Skrbka, Ph.D., kterou je možné naleznout na [3].

Další možností, kterou ESP32 nabízí, je rozhraní I^2C. Sběrnice I^2C má pouze dva vodiče SCL (serial clock line) a SDA (serial data line). ESP32 poskytuje celkem 2 tato rozhraní, která mohou pracovat jako I^2C master či slave. Záleží na nastavení uživatele. I^2C na ESP32 podporuje standardní mód s přenosovou rychlostí do 100Kbit/s a fast mód s přenosovou rychlostí 400Kbit/s. Také podporuje sedmibitový nebo desetibitový adresní mód. O tom, jak funguje I^2C, je také možné se dočíst na [3].

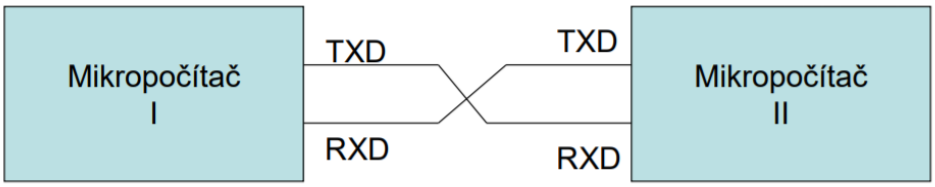
K přenosu digitálních audio dat nejlépe slouží sběrnice I^2S. Tato sběrnice byla vyvinuta firmou Philips Semiconductors. Sběrnice má celkem 3 vodiče: SCK (serial clock), WS (word select) a SD (serial data). Výhodou této sběrnice je, že může zpracovávat pouze data, zatímco ostatní signály (například řídící) může zpracovávat zvlášť. Data jsou přenášena v dvojkovém doplňku od nejvíce významného bitu. ESP32 poskytuje dvě standardní rozhraní I^2S. Ta mohou operovat v master nebo slave módu a mohou být konfigurována na 8, 16, 32, 48 nebo 64 bitové rozlišení. Rozhraní podporuje hodinovou frekvenci od 10kHz do 40MHz. Více o tomto typu sběrnice můžete nalézt na [4].

Abychom mohli v rámci mikrokontroléru dále pracovat s analogovými daty, která jsou produkována některými elektronickými součástkami (například mikrofonem, senzory teploty a tlaku), je nutné tato data převést do digitální podoby. Proto je nedílnou součástí většiny mikrokontrolérů (ESP32 nevyjímaje) A/D převodník, který tento převod zajišťuje. ESP32 integruje dvanáctibitový aproximační A/D převodník, který podporuje měření na celkem 18 kanálech (analogových pinech). Aproximační převodníky využívají k převodu komparátor, který srovnává vzorkované napětí na vstupu a výstup vnitřního D/A převodníku, což velmi často bývá polovina rozsahu napětí. Měření začíná od nejvíce významného bitu. Jestliže rozdíl obou hodnot je <0, uloží se do aproximačního registru 0 a následně se pokračuje na výstupu D/A převodníku s hodnotou o polovinu menší, než byla předchozí hodnota. V opačném případě se do aproximačního registru uloží 1 a k předchozí hodnotě výstupu D/A převodníku se přičte jeho polovina. Měření trvá po dobu délky aproximačního registru. Pro další detaily o funkci A/D převodníků bych rád odkázal na [5].

Detaily související přímo s modulem ESP32 jsou samozřejmě k dispozici na [2].

2.3 UART

Universal Asynchronous Receiver/Transmitter (UART) je definován jako základní sériové komunikační rozhraní počítače/mikrokontroléru. UART přenáší bajty dat po jednotlivých bitech sekvenčním způsobem. Na straně přijímače



Obrázek 2.4: Propojení dvou mikrokontrolérů pomocí rozhraní UART

druhý UART jednotlivé přijaté bity zase složí v celé bajty. Sériová komunikace se hojně využívala pro připojení modemů a dodnes se používá pro komunikaci mezi dvěmi zařízeními, jako jsou počítače, terminály a mikrokontroléry, které spolu nejsou propojené v síti. Jak vypadají dva kontroléry propojené pomocí rozhraní UART, si můžete prohlédnout na obrázku 2.4. Obě zařízení mají jak singál TXD (vysílač), tak signál RXD (přijímač). Tyto signály jsou zapojeny křížem z jednoho zařízení do druhého.

Jak už vyplývá z názvu, UART využívá asynchronní komunikace, což znamená, že se přenáší data bez toho, aniž by vysílač musel posílat hodinový signál přijímači. Místo toho se obě strany musí předem shodnout na časovacích parametrech a ke každému vyslanému slovu jsou přidány speciální bity určené k synchronizaci vysílací a příjimací jednotky.

Na začátek každého slova je přidán speciální bit jménem Start bit, který je použit k upozornění přijímače, že bude vysláno slovo dat. Tento bit také přinutí hodiny přijímače se synchronizovat s hodinami vysílače. Rychlosti obou hodin se nesmí lišit, jinak data nedojdou v pořádku.

Po vyslání Start bitu jsou přenášeny jednotlivé bity dat. Začíná se vždy od nejméně významného bitu. Každý bit je potom přenášen po úplně stejnou dobu, jako ostatní bity. Nutno podotknout, že vysílač neví, kdy se přijímač „podíval“ na hodnotu bitu. Ví jen, kdy hodiny vydávají pokyn k vyslání dalšího bitu. Proto je tolik důležité, aby se rychlosti obou hodin nelišily a data došla v pořádku.

Jakmile se přenese celé slovo dat, vysílač může ke slovu přidat bit parity. Tento bit slouží k odhalení jedné chyby ve slově. Poté přijímač „hledá“ takzvaný Stop bit. Pokud tento bit nedorazí, UART vyhodnotí toto slovo jako zkomolené a odešle hostitelskému procesoru zprávu, že nastal takzvaný Framing error. Nejčastější příčina této chyby jsou odlišné rychlosti hodin přijímače a vysílače nebo byl přerušen signál vysílání. Pokud je připraveno další slovo k odeslání, Start bit nového slova může být okamžitě vyslán, jakmile byl vyslán Stop bit předchozího slova.

Ještě bych rád dodal, že ve slově je možné poslat 7-9 datových bitů, přičemž standardem je 8 datových bitů + bit parity nebo 7 datových bitů + bit parity. Veškeré podrobnosti lze najít na [6]. Také jsem čerpal informace z přednášek

pana Ing. Miroslava Skrbka, PhD. Přednáška o sériové komunikaci je dostupná z [3].

2.4 WiFi

WiFi (Wireless Fidelity) je technologie pro zařízení využívající lokální bezdrátové sítě. Je založená na standardech IEEE 802.11, kterých je celá řada. Tyto standardy mají každý trochu odlišnou specifikaci. V bezdrátové lokální síti exitují 2 základní typy stanic. První z nich je takzvaný Access point. Tento typ stanice slouží jako prostředník přístupu k dalším počítačovým sítím, případně na internet. Druhým typem stanic jsou klienti. Komunikace v síti pak probíhá mezi klienty a tímto Access pointem. Existuje ještě také komunikace pouze mezi klienty. Takové síti se říká ad-hoc network. Tento typ sítě ovšem není důležitý pro tuto diplomovou práci a proto se jím nebudu dále zabývat.

Komunikace v síti využívá protokol CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance). Narozdíl od Ethernetu, který používá protokol CSMA/CD, rádiové systémy nejsou schopny detekovat kolize naslouchání při vlastním vysílání. Protokol CSMA/CA předchází těmto kolizím, které u Ethernetu vznikají tak, že se všechny stanice po uvolnění média snaží vyslat čekací rámce. Předchází jim tak, že v rámci okna „sváru“, které CSMA/CA zahájí po určité povinné době po uvolnění přenosového kanálu, si stanice náhodně zvolí svojí vlastní dobu čekání, která je násobkem dohodnutého časového úseku. Pokud nedojde k situaci, kdy dvě stanice začnou přesně ve stejný okamžik najednou vysílat, jedna stanice detekuje rádiový signál té druhé a odloží své vysílání. Dalším podrobnostem se zde již nebudu dále věnovat. Pro zájemce doporučuji náhlednout do [7].

Přístup na Access point WiFi je řešen několika způsoby. Prvním z nich je nešifrovaný volný přístup. Dále může být přístup na Access point řešen autentizačními protokoly. Historicky prvním z nich pro standard IEEE 802.11 je algoritmus WEP (Wired Equivalence Privacy). Tento algoritmus byl zodpovědný jak za autentizaci přístupu, tak za šifrování. WEP z počátku využívalo klíč dlouhý pouze 40 bitů, což se nakonec ukázalo jako velice náchylné k útokům hrubou silou. Z tohoto důvodu mnoho výrobců navrhlo rozšíření délky klíče na 104 bitů. Samotné šifrování je pak založeno na proudové šifře RC4.

Druhým protokolem je protokol WPA (WiFi protected access). Je to nástupce protokolu WEP a poprvé se objevil při dokončování standardu IEEE 802.11i. Problém útoků hrubou silou u WEP byl vyřešen tak, že u WPA byl implementován algoritmus TKIP (Temporal Key Integrity protocol). Ten ve zkratce zajišťoval generování klíče o délce 128 bitů pro každý paket zvlášť. Protože protokol WPA nepokrýval celý standard 802.11i z důvodů stále ještě probíhajících úprav, byl vyvinut později protokol WPA2. Ten už pokrýval celý tento standard a obsahoval i některá vylepšení oproti WPA. Pro detaily o autentizačních protokolech bych čtenáře rád odkázal na [8]

Typ standardu	Pásmo v GHz	Propustnost v Mbit/s
802.11	2,4	1/2
802.11b	2,4	5,5/11
802.11g	2,4	až 54
802.11n	2,4/5	>100

Tabulka 2.2: Srovnání parametrů standardů IEEE 802.11

Na závěr této kapitoly bych rád věnoval několik vět některým standardům IEEE 802.11. Jelikož mnoho z nich není důležitých pro tuto diplomovou práci, zmíním pouze IEEE 802.11b/g/n, které se týkají modulu ESP8266 a ESP32. Rád bych také doplnil, že následující text obsahuje pouze základní parametry, protože ty považuji pro účely práce za nejpodstatnější.

2.4.1 Standardy IEEE 802.11b/g/n

První standard, o kterém se chci zmínit, je 802.11b. Tento standard je rozšířením původního 802.11. Byl navržen tak, aby podporoval vyšší rychlost fyzické vrstvy pro práci v pásmu 2,4 GHz. Poskytuje datovou propustnost 5,5 Mbit/s a 11 Mbit/s oproti původnímu 802.11, který poskytoval ve stejném pásmu pouze 1 Mbit/s a 2 Mbit/s.

Další rozšíření 802.11g bylo poté navrženo pro ještě větší datovou propustnost, a sice až 54 Mbit/s. Pásmo je pak stejné jako u specifikace 802.11b.

Protože se později začala vyrábět WiFi zařízení, která podporují dvojí pásmo (jsou schopné pracovat ve dvou pásmech), byl vyvinut standard 802.11n. Ten tedy podporuje pásma 2,4 GHz a 5 GHz. S touto technologií se také zvýšila i datová propustnost, která může dosahovat i více než 100 Mbit/s. Pro přehlednost uvádím tabulku 2.2, která srovnává parametry jednotlivých zmíněných standardů.

Ke zpracování této sekce jsem čerpal z mnoha zdrojů. Jedním z nich byla i přednáška z Počítačových sítí pana doktora RNDr. Ing. Vladimíra Smotlachy, Ph.D., která je k dispozici na [9]. Dále jsem využil informace z [10], [11] a [12].

2.5 Možnosti předzpracování dat z mikrofonu v rámci čipu ESP32

Jestliže pracujeme s mikrofonem připojeným k modulu ESP32, je nutné vyřešit zejména způsob zpracování dat. Nejjednodušším způsobem je samozřejmě využití A/D převodníku, který modul poskytuje. Na aplikační úrovni je tento problém možné vyřešit tak, že jednoduše ve smyčce čteme analogová data z pinu, který je připojen k datovému pinu mikrofonu. Tato analogová data jsou

následně převedena do digitální podoby. Převedené hodnoty by bylo dále nutné ukládat do pole v pořadí, v jakém přijdou. Bohužel je nutné vzít v úvahu omezenou velikost paměti modulu, takže nelze ukládat příliš velká data najednou. Velká data by se musela přeposílat po částech na zařízení s větší pamětí, kde by se následně složila v jeden celek a dala se zkontrolovat.

Pokud bychom chtěli zaznamenávat obyčejnou mluvenou řeč, rozlišení 12 bitů tohoto A/D převodníku by mělo stačit, nicméně pokud by bylo potřeba zaznamenávat hudbu nebo by byla potřeba vyšší kvalita zvuku například k řízení nějakého robota, tento převodník není bohužel příliš vhodný. Obecně je známo, že pro záznam zvuku se využívá rozlišení 16 bitů a vzorkovací frekvence 44,1kHz. Takový A/D převodník ESP32 bohužel nemá k dispozici. Proto by bylo vhodné předzpracování těchto zvukových dat nechat na jiném mikrokontroléru.

Pro porovnání, ESP8266 poskytuje pouze desetibitový A/D převodník a tedy ani tento modul není vhodné použít pro předzpracování zvukových dat z mikrofonu

2.6 Porovnání modulů ESP8266 a ESP32

Protože hlavním cílem této diplomové práce je prozkoumat možnosti připojení senzorů, zpracování dat ze senzorů a přenos těchto dat pomocí modulů ESP8266 a ESP32, budu se v této kapitole zabývat teoretickými možnostmi obou modulů a také jejich vzájemným porovnáním. Nejprve se pojďme podívat na možnosti připojení různých senzorů.

Jak už jsem zmínil v předchozích kapitolách, ESP8266 má celkem 16 pinů. Z těchto pinů je celkem 5 pinů rezervovaných. Dále zde máme piny RXD0 a TXD0 pro sériovou linku. Zbývá tedy celkem 9 pinů, které by mohly být využity pro připojení senzorů. Pokud bychom si vzali například mikrofon GY 4466, který má celkem 3 piny (napájení, uzemění a výstup) a který má napájecí napětí 2,4 - 5,5 V, neměl by být s připojením vůbec žádný problém. Také u jednodušších senzorů teploty a vlhkosti, jako je například senzor DHT11, není s připojením žádný problém. Tento senzor využívá celkem 3 piny ze svých 4 a napájecí napětí je 3-5,5 V. Bohužel díky tomu, že ESP8266 má operační napětí 3-3,6 V, jsme omezeni vybírat pouze ze senzorů, které můžeme napájet 3,3 V. Také pokud bychom chtěli připojit více senzorů najednou, jsme omezeni počtem pinů, které můžeme teoreticky využít a tudíž lze připojit maximálně 3 jednoduché senzory najednou.

Z hlediska napájení to má ESP32 podobně jako starší verze ESP8266 uvedená v předchozím odstavci. Ovšem z hlediska počtu pinů je na tom ESP32 podstatně lépe. Tento modul má celkem 38 pinů, z nichž můžeme využít 22, případně i 24, pokud bychom využívali UART0 pouze pro nahrávání firmware.

Pokud bychom chtěli zpracovávat data ze senzorů přímo na modulech, je důležité se zaměřit zejména na 2 klíčové parametry modulů a sice na velikost

paměti dostupnou pro programátora a na výkon procesoru. Zatímco na modulu ESP32 je k dispozici teoreticky až 520 kB paměti, ESP8266 nabízí pouze 80 kB. I když v obou případech nebude jistě možné využít celou tuto kapacitu (z tohoto volného místa se například naalokuje paměť pro WiFi případně Bluetooth), je z těchto údajů jasné, že u ESP8266 nebude možné uchovávat příliš velká data a tím pádem celý program bude muset být mnohem úspornější, než kdybychom použili novější modul ESP32.

Důležitá je také rychlost zpracování dat. Pokud bychom potřebovali zpracovávat větší množství dat najednou a přeposílat je dále, nemusí se použití ESP8266 jevit jako dobré řešení. Tento modul má pouze jeden procesor o frekvenci 80 MHz, což dnešním aplikacím také nemusí stačit. ESP32 má k dispozici celkem 2 jádra o frekvenci 160 MHz s možností taktování až na 240 MHz. Z tohoto pohledu se pro větší aplikace hodí více tento modul.

Co se týká rozhraní UART je na tom ESP32 také o něco lépe. Tento modul disponuje celkem třemi těmito rozhraními, které všechny dosahují teoretické propustnosti až 5 Mbit/s. Oproti tomu starší modul ESP8266 má pouze jedno kompletní rozhraní UART a přídavný vysílač sériové linky, který je namapován na pin GPIO2. Přes UART se na tento modul také nahrává firmware a tedy z celkového pohledu jsme omezeni, pokud bychom k modulu chtěli přes UART připojit ještě jedno zařízení. Propustnost je ovšem srovnatelná a dosahuje teoreticky až 4,5 Mbit/s.

Schopnosti WiFi jsou však u obou modulů stejné. Oba moduly totiž disponují standardy IEEE 802.11 b/g/n, o kterých jsem se zmínil v předchozí sekci. Přesto i zde několik rozdílů najdeme. Kupříkladu ESP32 poskytuje modernější techniky šifrování a autentizace, má rozmanitější škálu síťových protokolů. Oproti ESP8266 například poskytuje protokol SSL a IPv6, což tento starší typ nemá.

Z celkového pohledu se zdá, že ESP32 je oproti ESP8266 modernější, rychlejší a rozmanitější, ať už se týká rychlosti procesoru, síťových protokolů nebo počtu pinů. Tomu ovšem odpovídá cena modulů. Cena staršího modulu ESP8266 se k datu 10.12.2018 pohybovala okolo 120 Kč za kus, zatímco cena ESP32 se pohybovala kolem 520 Kč za kus. Cena ESP32 však není nijak likvidační a to ani v případě masového použití, takže podle mého názoru se investice do ESP32 podle tohoto srovnání vyplatí.

Otázkou však zůstává, co se stane, pokud bychom chtěli nějakým způsobem propojit práci rozhraní UART a WiFi. Bude skutečně ESP32 výrazně rychlejší a spolehlivější než ESP8266, nebo vlivem zpracování dat či některých prvků modulů se rozdíly výrazně sníží či úplně smažou? Tomuto problému se budu dále věnovat v následujících kapitolách.

Kapitola 3

Návrh

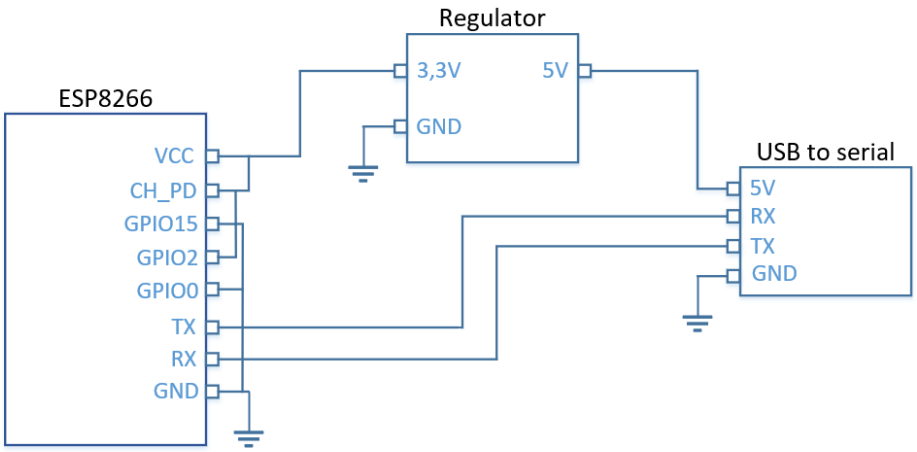
Protože praktickým cílem této práce je ověřit možnosti přenosu dat na obou modulech, je nutné rozdělit návrh řešení na dvě části. Nejprve je nutné navhrnout zapojení obou modulů tak, aby vyhovovala požadavkům pro tuto práci a následně je možné navrhnout samotný program respektive části programu. Této problematice se věnuji v následujícím textu.

3.1 Návrh zapojení modulu ESP8266

K tomu, aby bylo vůbec možné nahrát firmware do modulu ESP8266, musel jsem se se při návrhu zapojení vypořádat hned s několika problémy. Nahrání firmware do modulu je možné pouze přes sériovou linku, a protože samotný program jsem vyvíjel na stolním počítači popřípadě na notebooku, musel jsem zajistit propojení počítače a modulu. Nejjednodušším způsobem, jak realizovat toto propojení, je pomocí převodníku USB-to-serial. Bohužel modul ESP8266 nedisponuje takovým konektorem, proto je nutné použít externí převodník. Tento převodník se propojí s modulem přes rozhraní UART. Převodník je napájen 5V přímo přes USB konektor počítače, takže externí napájecí zdroj není potřeba.

Otázku zdroje napájecího napětí bylo v tuto chvíli možné řešit dvěma způsoby. Buď použít externí zdroj napětí nebo využít zdroje napětí od převodníku USB-to-serial. Při využití zdroje od převodníku je nutné brát v potaz, že modul ESP8266 je napájen pouze 3,3V, a proto převodník musí mít integrovaný regulátor napětí nebo je nutné využít externí regulátor.

V datasheetu pro tento modul [1] je také napsáno, jak pracovat s některými piny v různých módech. Módy jsou zde uvedeny dva. Mód UART a mód Flash boot. Mód Flash boot je určen pro nahrávání nového firmwaru a mód UART zase pro využívání sériové linky a normální provoz. K těmto účelům bylo nutné správně zapojit 3 piny. GPIO0, GPIO2 a GPIO15. Zapojení pinu GPIO2 a GPIO15 je stejné pro oba módy. Rozdíl je pouze v tom, že GPIO2 se připojí ke zdroji napájecího napětí a GPIO15 se připojí k zemi. Pin GPIO0 má být



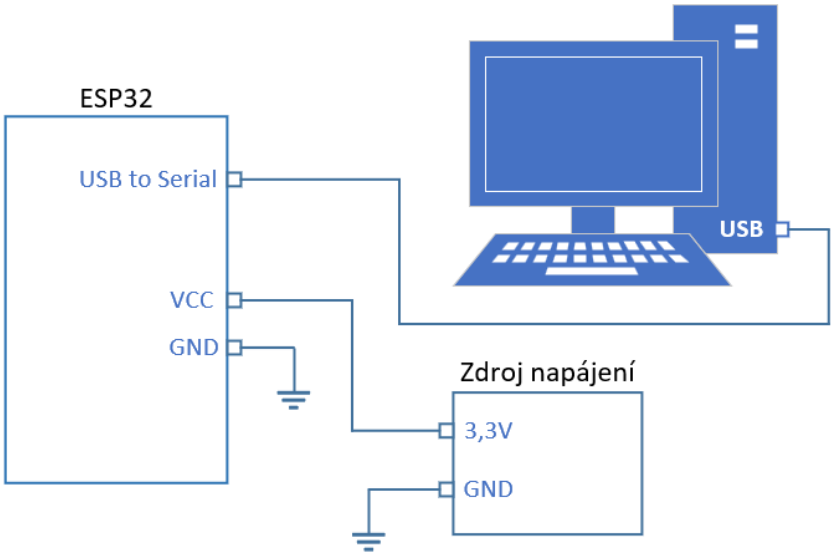
Obrázek 3.1: Blokové schéma návrhu zapojení modulu ESP8266

připojen pro Flash boot mód ke zdroji napájecího napětí a pro mód UART zase k zemi. Tato situace se dá řešit dvěma způsoby. První možností je tento pin připojit přes rezistor (o velikosti 1kΩ) k napájecímu napětí a přes tlačítko k zemi. V takovém případě při nahrávání firmware musí být vzhledem k zemi rozepnuto a sepnuto v opačném případě. Druhou možností je připojit pin GPIO0 trvale k zemi. V tom případě si režim Flash boot Arduino IDE, které jsem pro tuto práci použil, zapíná samo. Vývojovému prostředí Arduino IDE se detailněji věnuji v sekci 4.2.1.

Posledním pinem, který je nutno správně zapojit je pin CH_PD. Jak jsem již uvedl v předchozích kapitolách této práce, tímto pinem říkáme, jestli má zařízení pracovat v úsporném režimu nebo v normálním. Vzhledem k tomu, že úsporný režim v této práci neuvažuji, připojíme pin CH_PD ke zdroji napájení. Tímto je modul ESP8266 připraven k použití. Jak je již z tohoto textu patrné, zbývá pouze 6 pinů, které jsou k dispozici uživateli například pro připojení různých senzorů. Zde popsané kompletní zapojení modulu ESP8266 znázorňuje blokové schéma 3.1.

3.2 Návrh zapojení modulu ESP32

V případě ESP32 je práce, co se týká zapojení, o poznání jednodušší. ESP32 má totiž integrován převodník USB-to-UART, takže otázku připojení k počítači lze vyřešit velmi jednodušše pouze připojením USB kabelu. Touto cestou je také možné vyřešit otázku napájení modulu, jelikož ESP32 je takto napájen přímo z počítače přes USB kabel a tedy není potřeba žádný externí zdroj. Přesto se ale toto řešení u některých aplikací náročných na napájení (napří-



Obrázek 3.2: Blokové schéma zapojení modulu ESP32

klad využívání WiFi a Bluetooth zároveň) může ukázat jako nedostatečné a je potom nutné využít externí zdroj napájení jako výkonnější zdroj proudu.

Narozdíl od staršího modulu ESP8266, není potřeba nastavovat žádné módy u žádných pinů, takže pro uživatele jsou k dispozici všechny piny. Také pro nahrávání firmwaru není potřeba využít přímo piny pro UART0, protože nahrávání probíhá také přes USB rozhraní. To znamená, že po nahrání firmwaru může uživatel použít UART0 celkem dvěma způsoby. Může posílat data z počítače přes sériovou linku na modul nebo k pinům UART0 na modulu může připojit jakékoliv externí zařízení schopné této komunikace.

Celkově tedy ESP32 budí mnohem sofistikovanější dojem, než starší verze ESP8266. Jednoduché blokové schéma tohoto zapojení je možné vidět na obrázku 3.2.

3.3 Návrh programu

Jakmile jsem vyřešil otázku zapojení modulů, pustil jsem se do samotného návrhu programu. Cílem praktické části je prozkoumat možnosti přenosu dat na obou čipech, takže je nutné ověřit zejména, jakým způsobem se dají data přenášet do modulu, z jakých zařízení je to možné, jaké protokoly využít k řešení této problematiky a v neposlední řadě, jaké maximální propustnosti je

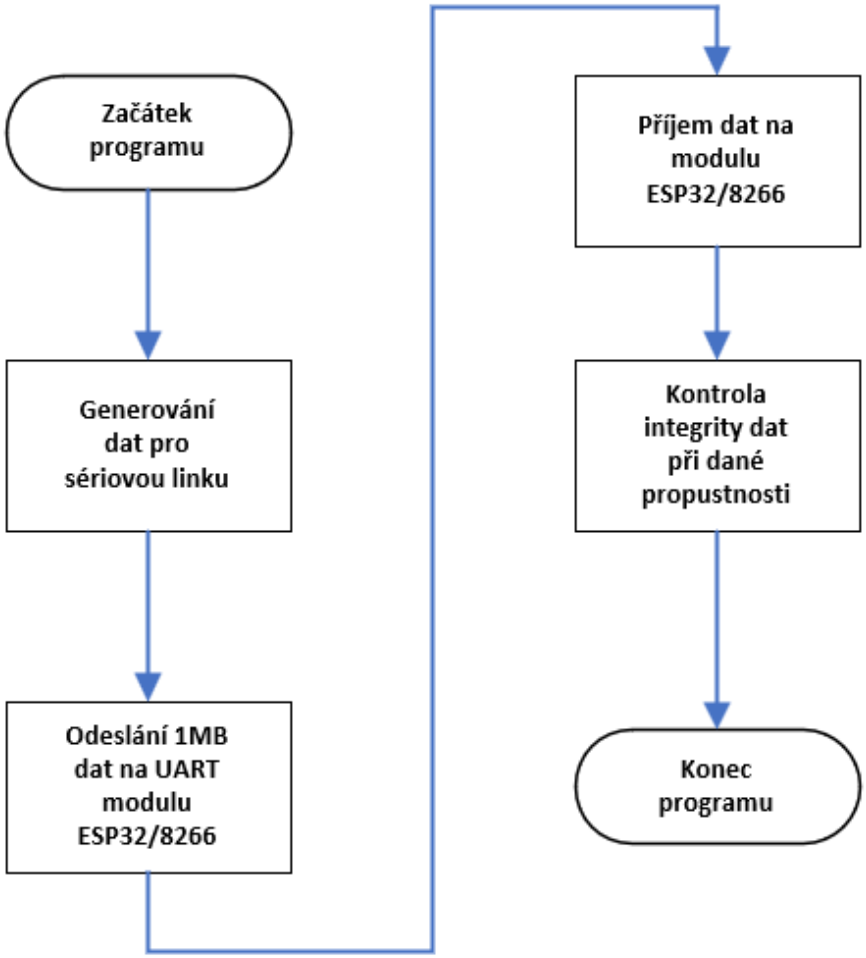
možné při přenosu dat dosáhnout.

Pro oba moduly jsou k dispozici celkem dva možné způspoby přenášení dat, UART a WiFi. Na modulu ESP32 je možné přenášet data také pomocí Bluetooth verze 4.2, který je moderní a velice výkonný, ale po dohodě s vedoucím práce a protože by výsledky z pozorování přenosu dat pomocí této technologie nebyly s čím porovnat, dále jsem se touto problematikou nezabýval. Na modulu ESP8266 totiž Bluetooth není podporován. Proto je návrh programu zaměřen pouze na UART a WiFi. Abych správně otestoval vlastnosti těchto možností přenosu, musel jsem se zaměřit především na tyto tři základní přístupy: Samostatné otestování sériové linky, Samostatné otestování WiFi a vytvoření tunelu UART-WiFi.

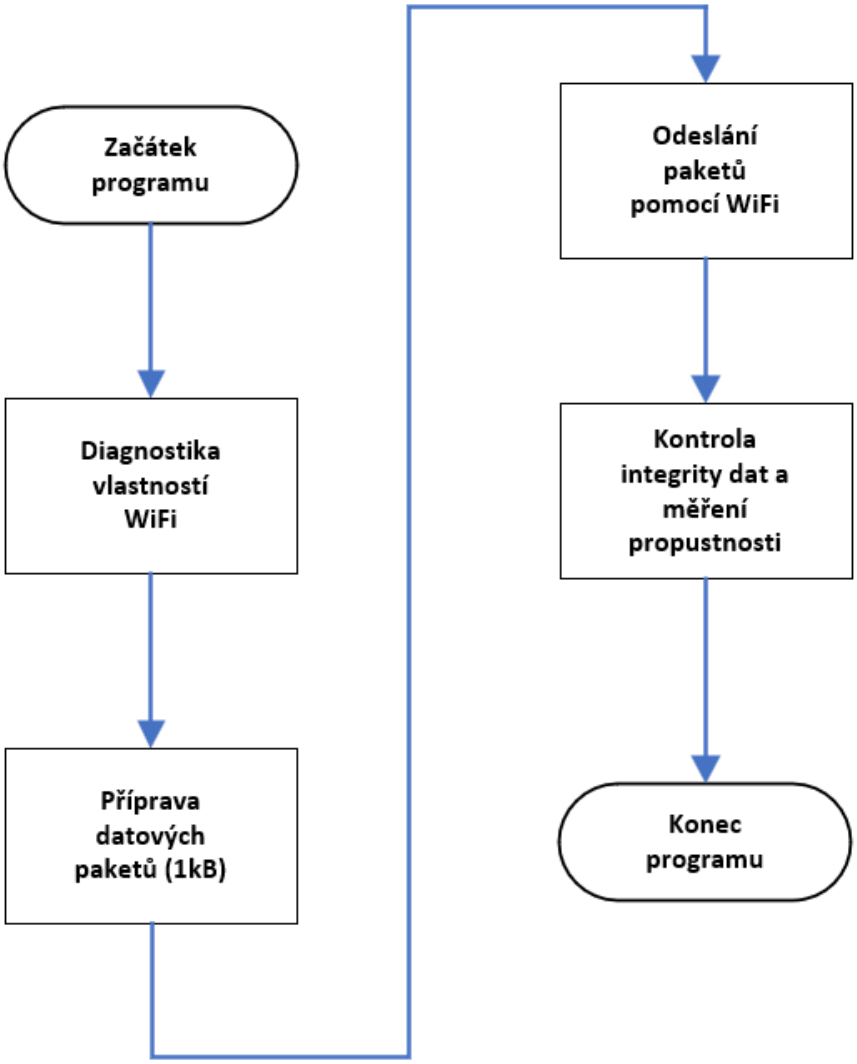
Při testování sériové linky je potřeba se zaměřit na příjem dat na modulu, jestli při dané propustnosti je schopen modul data přijímat všechna a dále také, jestli data, které modul přijal, neobsahují chyby jako jsou například přeházené bajty, menší objem přijatých dat, než bylo odeslaných atd. V průběhu testování také ověřím, zda chyby nenastávají i ve směru UART modulu - PC, aby bylo jasné, jestli na jedné straně nevzniká více chyb, než na té druhé. Program tedy bude vypadat velmi jednoduše a to tak, že externí zařízení bude vysílat pomocí sériové linky data o zvolené velikosti a modul bude tato data přijímat a následně určité množství dat vypisovat do terminálu či logovacího souboru. Jako externí zařízení jsem zvolil stolní počítač, protože se jeví jako nejjednodušší řešení, zejména, co se týká sběru a analýzy dat z modulu. Samozřejmě, přenosové rychlosti modulu i počítače musí být stejné. Tento koncept jsem použil pro oba moduly. Na obrázku 3.3 je znázorněn vývojový diagram popsaného programu.

Pro program pro testování WiFi máme na výběr ze tří standardů. Ovšem všechny tři standardy jsou, co se týče propustnosti, rychlejší, než maximální propustnost sériové linky. Takže výběr standardu by neměl mít vliv na třetí program, tedy na program testující tunel UART-WiFi. V tomto případě budeme testovat schopnosti a vlastnosti vestavěné WiFi. Nejdůležitější bude zjistit, jakým způsobem se dají nastavovat jednotlivé protokoly, kolik kanálů mají k dispozici oba moduly, zda a jakým způsobem se dají nastavit moduly do módu stanice nebo AP a jiné vlastnosti. Bude se tedy jednat o celkovou diagnostiku WiFi.

Dále je nutné otestovat schopnost modulu odesílat dané množství předpřipravených dat a tedy reálnou propustnost WiFi. Velikost datového paketu jsem určil po dohodě s vedoucím práce na 1KB. Celkový objem odesílaných dat poté na 1MB. Dále bylo nutné zvolit vhodný síťový protokol pro takový test. Pro takovéto testování se nejlépe jeví protokoly TCP a UDP. Problém s TCP ale spočívá v tom, že je to potvrzovaný protokol a čekání na potvrzení došlého paketu nebo dokonce opakování přenosu paketu, který nedošel, by mohlo značně ztěžovat test a tím pádem by pozorování nemělo vypovídající hodnotu. Cílem je totiž zjistit, jaká je možná maximální propustnost, za které bude možné spolehlivě přenést všechna data nebo alespoň jejich výraznou



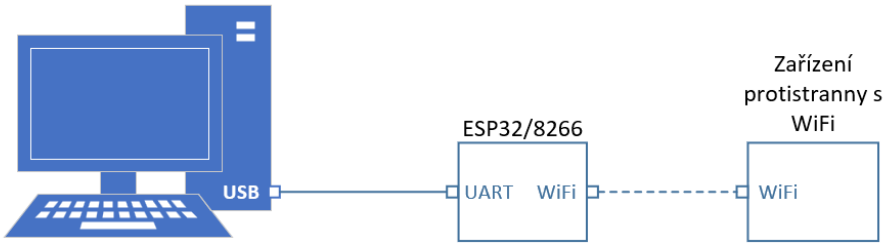
Obrázek 3.3: Vývojový diagram programu testující nejvyšší možnou propustnost UART



Obrázek 3.4: Vývojový diagram programu testující WiFi

většinu. Proto jsem zvolil protokol UDP, který není potvrzovaný. Vývojový diagram tohoto programu je patrný z obrázku 3.4.

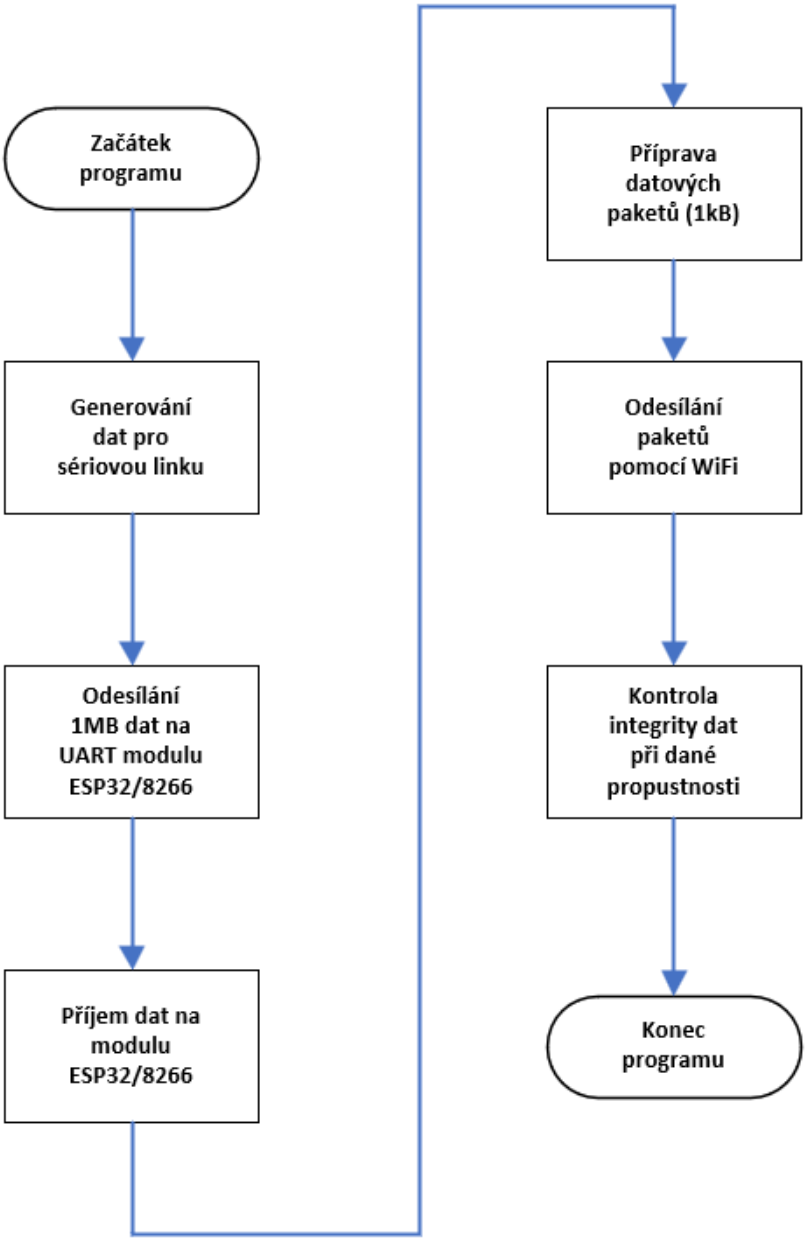
Posledním krokem bylo navrhnout program, který by integroval obě části dohromady. Navrhnul jsem tedy tunel UART-WiFi, který má za úkol na modulu přijímat data z externího zařízení, která přijdou ze sériové linky, na modulu se předzpracují a pomocí WiFi se odešle datový paket dané velikosti na příslušnou protistranu. Pro tento účel mohou být data generována pomocí



Obrázek 3.5: Schéma funkce programu tunelu UART-WiFi

počítače, odesílána na modul a přijímána a kontrolována buď na jiném nebo stejném externím zařízení. Vzhledem k povaze práce jsem zvolil stejné externí zařízení. Při předzpracování dat na modulu je potřeba si dát pozor především na omezenou velikost paměti a tedy neukládat na modul příliš velká pole dat. Z tohoto důvodu a z důvodu toho, že se dále po WiFi odesílají datové pakety o velikosti 1KB, předzpracovaná data na modulu budou také o velikosti 1KB. Schéma funkce tohoto programu je možné vidět na obrázku 3.5. Dále pro větší názornost přikládám také vývojový diagram tohoto programu, který je zobrazen na obrázku 3.6

Nadále však zůstává otázka, jakým způsobem generovat data a jak data vyslaná po WiFi kontrolovat a také jak přesně měřit nejvyšší možnou propustnost pro oba moduly. Tato problematika bude diskutována v nálsedujících kapitolách.



Obrázek 3.6: Vývojový diagram programu tunelu UART-WiFi

Kapitola 4

Implementace

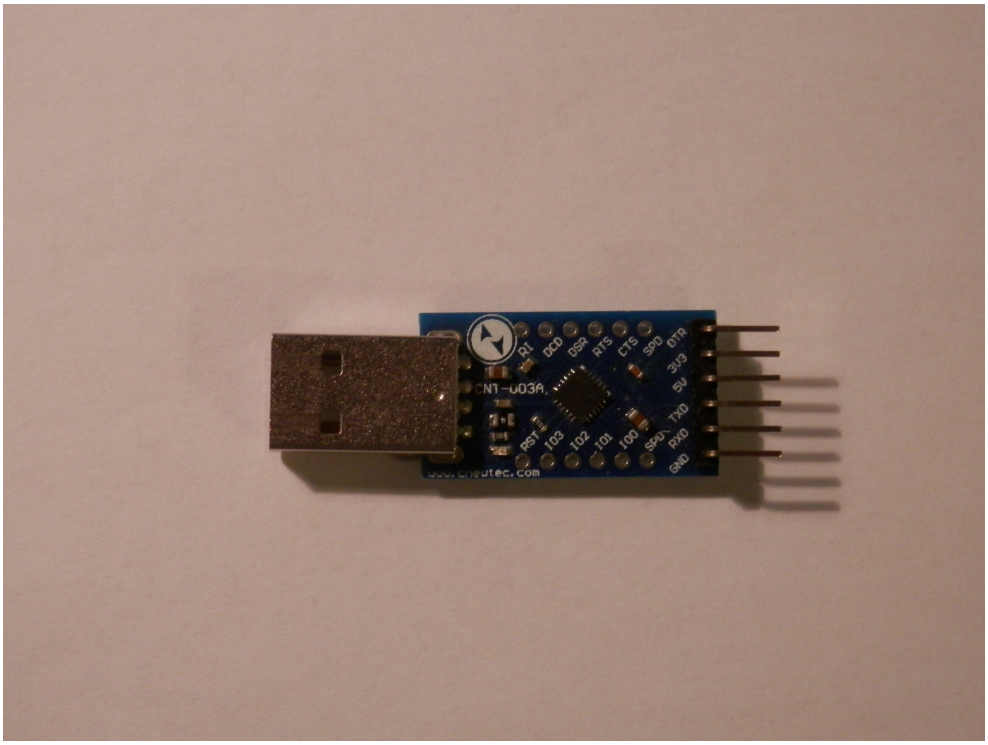
V této kapitole bych rád čtenáře seznámil s postupy, které jsem použil k ověření možností přenosu dat na modulech ESP8266 a ESP32. Uvádím zde popis a fragmenty vytvořeného kódu a dále také použitý hardware a nástroje nutné k vypracování praktické části diplomové práce. Kromě toho kapitola také obsahuje veškeré mnou použité knihovny kompatibilní s WiFi moduly. Neuvádím zde však výsledky, ke kterým jsem došel, jelikož to je předmětem až následující kapitoly.

4.1 Použitý hardware k realizaci zapojení modulů

Ještě než začnu popisovat softwarovou část, je důležité čtenáře seznámit také s hardwarovými součástkami, bez kterých by testování možnsotí přenosu v praxi nebylo vůbec možné.

4.1.1 Převodník CP2104

Jak už jsem zmínil v kapitole návrhu, k propojení modulu ESP8266 a počítače je potřeba převodník USB to UART, abychom mohli nahrávat firmware. Protože tento modul žádným takovým převodníkem nedisponuje, využil jsem externí převodník CP2104. Tento převodník vyrábí firma Silicon Labs. Převodník disponuje USB konektorem a celkem 6 piny. Mezi ně patří piny pro UART TXD a RXD, tedy přijímač a vysílač, dále pak pin pro uzemění, piny pro napájení 3,3V a 5V a nakonec pin DTR (Data Terminal Ready). Poslední zmíněný pin se využívá pro rozhraní RS-232, se kterým však nepracuji, proto se jím nebudu dále zabývat. Protože USB z počítače poskytuje napájení 5V a modul ESP8266 je nutné napájet 3,3V, je potřeba napětí z USB nějakým způsobem regulovat. U některých jiných převodníků je proto ještě potřeba připojit externí regulátor, avšak zde je práce uživateli ulehčena, jelikož tento typ obsahuje regulátor interní a výše zmíněný pin pro 3,3V. Jeho teoretická přenosová rychlost dosahuje až 2Mbit/s. Pro názornost přikládám obrázek 4.1



Obrázek 4.1: Reálná podoba převodníku CP2104

s reálnou podobou tohoto převodníku. Více informací o tomto převodníku je možné nalézt v datasheetu [13].

4.2 Použitý software

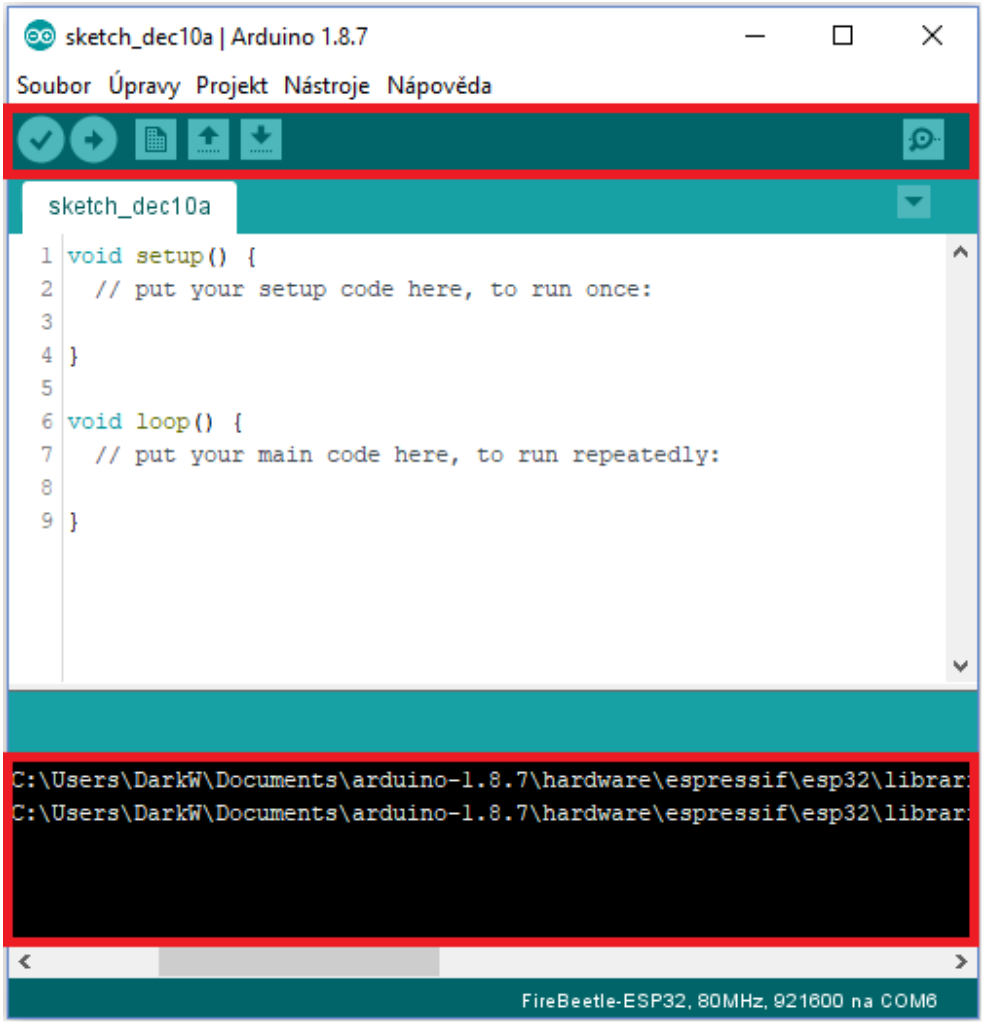
Abych mohl přejít k samotné implementaci programu a poté k testování možností modulů, musel jsem nejprve vyřešit několik problémů, a sice jakým způsobem odesílat data do modulů, jak bude možné nahrávat firmware do modulů co možná nejjednodušším způsobem a v neposlední řadě, jak kontrolovat správnost dat odesílaných a zpracovaných při dané propustnosti. K vyřešení těchto dílčích úkolů jsem použil následující nástroje.

4.2.1 Arduino IDE

Arduino IDE je open-source vývojové prostředí poskytované firmou Arduino. Tato firma kromě tohoto prostředí prodává také nejrůznější Arduino desky včetně příslušných přídavných součástek a shieldů. Vývojové prostředí Arduino IDE je kompatibilní nejen s deskami Arduino, ale právě také s moduly ESP. Vývoj pro desky Arduino, respektive v mém případě pro moduly ESP, je realizován v jazyce C++. Arduino IDE je poměrně hojně používaný nástroj

firmami, které se zabývají prací s embedded zařízeními a Internetem věcí. Velkou výhodou tohoto prostředí je snadný vývoj a přehlednost zdrojových kódů. Pro mou diplomovou práci jsem použil Arduino IDE verze 1.8.7, které je volně dostupné ke stažení na adrese `https://www.arduino.cc/en/Main/OldSoftwareReleases`. Inicializace hardware je řešená v příslušných knihovnách, které jsou volně dostupné ke stažení, což programátorovi ušetří mnoho práce. V zásadě vše, o co se programátor musí zajímat, jsou dvě funkce: Funkce setup() a funkce hlavní smyčky loop(). Prázdný projekt v Arduino IDE je znázorněn na obrázku 4.2. Pro přehlednost jsem vyznačil dvě nejdůležitější místa. Na vyznačené horní liště jsou zleva tlačítka pro kompilaci projektu, kompilaci projektu + nahrání firmware do modulu, vytvořit nový soubor, otevřít existující soubor, uložit soubor a poslední tlačítko slouží k otevření sériového monitoru, který je také možné otevřít z položky menu „Nástroje“. Ve spodním červeně označeném černém okně jsou pak vidět výpisy kompilátoru pro příslušnou platformu.

Funkce setup() slouží programátorovi k inicializaci různých rozhraní, proměnných nebo čehokoliv, co potřebuje pro svou práci a co není implicitně nastaveno v příslušných knihovnách. Samozřejmě je možné přidávat do projektu i vlastní funkce, proměnné a jiné užitečné věci podle potřeby. Obě tyto funkce, setup() i loop(), jsou pak volány z hlavní funkce programu main() umístěné v sobouru main.cpp. Jelikož ESP32 využívá operační systém FreeRTOS, je soubor main.cpp odlišný od souboru main.cpp, které využívají desky Arduino bez operačního systému či modul ESP8266. Hlavním důvodem této odlišnosti je to, že FreeRTOS je založen na funkcích nazývaných Task. Protože FreeRTOS ani systémy reálného času nejsou předmětem této diplomové práce, nebudu se tím podrobněji zabývat. Užitečné informace ohledně tohoto operačního systému jsou k dispozici na [14] a na [15]. Pouze bych rád uvedl, že klíčovým slovem „xTaskCreate“ říkáme systému FreeRTOS, aby založil nový Task s odkazem na funkci, kterou má spustit. Tento fakt a hlavní rozdíly mezi zmíněnými soubory main.cpp jsou patrné z následujících fragmentů zdrojového kódu. První fragment se týká Arduino kompatibilních desek.



Obrázek 4.2: Prázdný projekt v Arduino IDE

```
int main(void)
{
  init();

  initVariant();

#if defined(USBCON)
  USBDevice.attach();
#endif

  setup();
```

```
  for (;;) {
    loop();
    if (serialEventRun) serialEventRun();
  }

  return 0;
}
```

Následuje fragment zdrojového kódu souboru main.cpp pro ESP32.

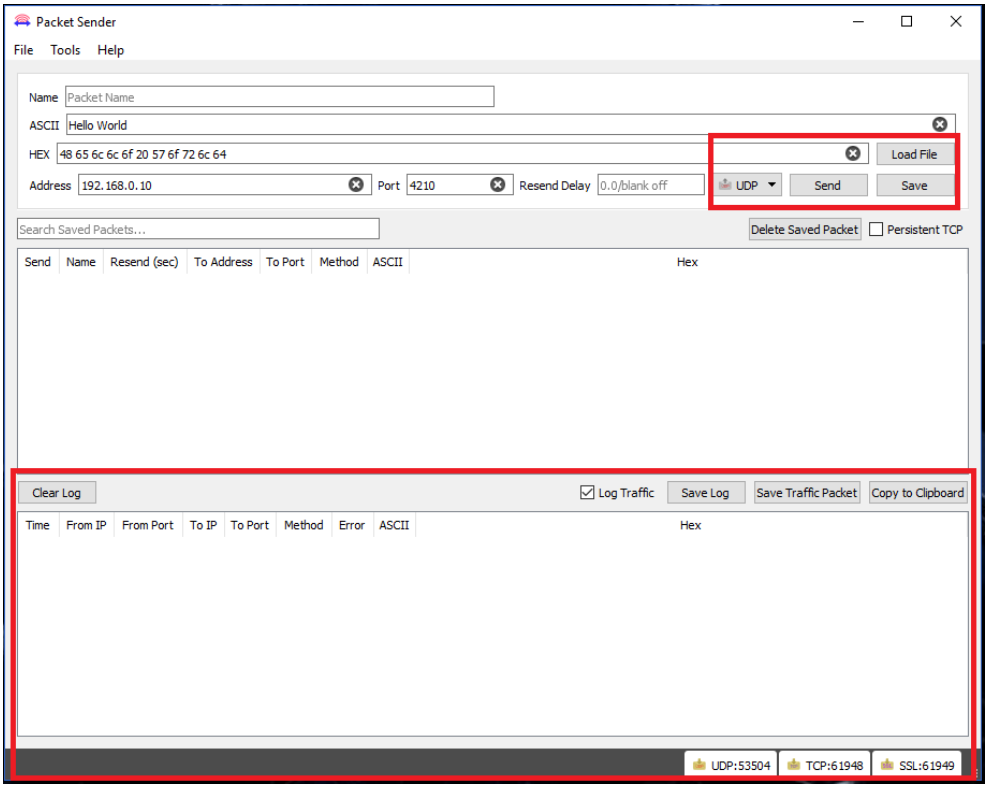
```
void loopTask(void *pvParameters)
{
    setup();
    for(;;) {
        loop();
    }
}

extern "C" void app_main()
{
    initArduino();
    xTaskCreatePinnedToCore(loopTask,
    "loopTask", 8192, NULL, 1,
    NULL, ARDUINO_RUNNING_CORE);
}
```

Samozřejmě existuje mnoho různých možností, jak pracovat s moduly ESP. Existuje například nástroj ESP-IDF (Espressif IoT Development Framework), který, jak už sám název vypovídá, byl vyvinut firmou Espressiv (která také vyrábí moduly ESP). Tento framework je volně dostupný na adrese `https://github.com/espressif/esp-idf`. Tento framework se po instalaci sváže s příslušným Toolchainem pro moduly ESP a programátor může vyvíjet firmware v jakémkoliv editoru (například může využít Eclipse). Drobnou nevýhodou práce s ESP-IDF je, že je nutné dát dohromady mnoho věcí už jen k tomu, abychom vůbec mohli vyvíjet pro moduly ESP. Dále je nutné inicializaci hardware provést vlastními silami. Jelikož jsou tyto úkony v Arduino IDE vyřešeny za nás a cílem této diplomové práce není naučit se pracovat s ESP-IDF, zvolil jsem Arduino IDE. Navíc s prací s Arduino IDE už mám určité zkušenosti, takže i to přispělo k mému výběru. Zájemci se mohou detailněji seznámit s ESP-IDF na [16].

4.2.2 Packet Sender

Jak už jsem se v předchozích kapitolách této práce zmínil, důležitou součástí při zkoumání možností přenosu dat, a maximální možné propustnosti, je sledování integrity dat. Co se týká části UART, k tomu lze jednoduše využít



Obrázek 4.3: Grafické rozhraní programu Packet Sender

sériový monitor, který má k dispozici Arduino IDE nebo je možné využít kterýkoliv jiný nástroj poskytovaný operčním systémem nebo pro operační systém dostupný. V případě tunelu UART-WiFi je situace jiná a je potřeba nějakým nástrojem odchytávat datové pakety tak, abychom měli možnost sledovat správnost jejich obsahu. K tomuto účelu jsem zvolil program Packet Sender. Je to jednoduchý open source nástroj od firmy NagleCode, který je schopen přijímat, ale i odesílat TCP, UDP a SSL (šifrované TCP) pakety. V současnosti tento software podporuje protokoly IPv4 i IPv6 a má širokou škálu využití v oblasti sítí, mezi kterou patří například i analýza malware. Firma poskytuje software v mnoha různých variantách pro různé operační systémy, například Windows, Mac OS a Linux. Pro Windows má i rozšíření do příkazové řádky. Existuje i varianta jako mobilní aplikace. Pro účely mé práce mi zcela postačila desktopová verze pro operační systém Windows. V mé diplomové práci jsem pracoval s aplikací Packet Sender verze 5.6.2, kterou naleznete na přiloženém CD. Aktuální verzi naleznete na `https://packetsender.com/download#show`. Podobu programu přikládám na obrázku 4.3.

Jak je z obrázku patrné, v horní části dialogového okna můžeme nastavit název paketu, ASCII podobobu datového obsahu paketu, jeho hexadecimální

podobu, cílovou adresu a port, na který hodláme odeslat příslušný paket a pak také zpoždění, které je aplikováno na opětovné odeslání paketu v případě, že jsme využili protokol TCP či SSL a prvotní odeslání paketu se nezdařilo. Ve vyznačené horní části se pak nachází tlačítko pro volbu síťového protokolu, tedy TCP, UDP nebo SSL, tlačítko pro odeslání paketu, uložení paketu nebo načtení obsahu souboru, jehož obsah chceme odeslat. V okně pod těmito tlačítky a pod nastaveními, které jsem právě zmínil, můžeme mít uložené různé pakety, které potřebujeme pro práci. Ve vyznačené spodní části je možné sledovat příchozí pakety společně s jejich časovou značkou, zdrojovou adresou a portem, cílovou adresou a portem, protokolem svázaným s tímto paketem. Dále pak typ chyby, pokud nějaká při přenosu nastala a nakonec ASCII a hexadecimální podobu obsahu paketu. Tento datový tok je také možné uložit do zvoleného logovacího souboru. V pravé spodní části jsou pak vyznačeny porty pro příslušné protokoly. Tyto porty je pak důležité předat aplikaci, která odesílá pakety právě tomuto programu. Z mého pohledu je tento nástroj velice šikovnou pomůckou pro testování jakýchkoliv síťových API, proto jsem neváhal při jeho volbě. Pro více podrobností bych čtenáře rád odkázal na příslušnou dokumentaci, která je k dispozici na [17].

4.2.3 Aplikace SerialSend

Kromě nástrojů pro otestování samotné WiFi, bylo potřeba najít vhodné nástroje k otestování rozhraní UART na modulech ESP. Jak už jsem zmínil v sekci Arduino IDE, toto vývojové prostředí má integrovaný sériový monitor, takže problém se sledováním příchozích dat na UART modulů ESP je vyřešený. Bylo však nutné najít vhodné řešení pro odesílání dat na sériovou linku modulů ESP. Jestliže je modul ESP připojen k počítači například pomocí převodníku CP2104, hlasí se zařízení v operačním systému jako virtuální COM port. Odeslání dat na tento COM port lze uskutečnit v operačním systému Windows následujícím jednoduchým příkazem:

```
echo "text to send" > COM1
```

Číslo COM portu se samozřejmě vždy liší a záleží na operačním systému, které číslo zařízení přidělí. Číslo v tomto příkazu je zde jen pro ilustraci. Bohužel i takový příkaz má mnoho nedostatků, které jsou pro účely této práce nežadoucí. Jedním z nich je, že se v tomto případě přenáší na COM port znaky <CR> a <LF>. Dále je také nutné zkontrolovat, popřípadě nastavit požadovaný baudrate pro příslušné testy sériové linky. Toto všechno je možné vyřešit buď ručním nastavením nebo například pomocí připraveného skriptu, který by tato nastavení prováděl. Při analýze tohoto problému jsem však narazil ještě na třetí možnost. Tou je aplikace příkazové řádky SerialSend pro operační systém Windows, kterou vyvinul profesor Ted Burke z Dublinského institutu technologie. Tento jednoduchý nástroj slouží k odesílání textových řetězců na COM port a jak sám autor této aplikace uvádí, je tento program

vyvinut hlavně pro odesílání dat mikrokontrolérům, takže hlavně z tohoto důvodu jsem zvolil tuto variantu. Navíc tento program řeší i problematiku potřebných nastavení, pro které bych jinak musel zhotovit skript. Software je volně dostupný ke stažení včetně jeho zdrojového kódu, což otevírá možnost i k jistým úpravám, které jsem nakonec v průběhu tohoto projektu využil.

SerialSend posílá pomocí jednoho jednoduchého příkazu zadaný text na hardwarové zařízení pomocí virtuálního COM portu. V tomto příkazu musíme nastavit baudrate a poté volitelně také číslo COM portu. Pokud ho nenastavíme, program automaticky zvolí nejvyšší číslo. Pokud není žádoucí odesílat text na zařízení a záměrem je posílání jiných netisknutelných znaků, je možné tuto možnost v příkazu nastavit.

Následující příkaz zobrazuje základní využití nástroje SerialSend s nastavením příslušného baudrate a čísla COM portu:

```
SerialSend.exe /baudrate 9600 /devnum 1 "Hello␣World"
```

V následujícím příkazu pro informaci uvádím variantu s nastavením hexadecimální podoby odesílaných dat. Podotýkám, že v uvozovkách musí být ještě pomocí escape sekvence „\x“ označen každý jednotlivý odesílaný bajt. Příkaz potom má například následující podobu:

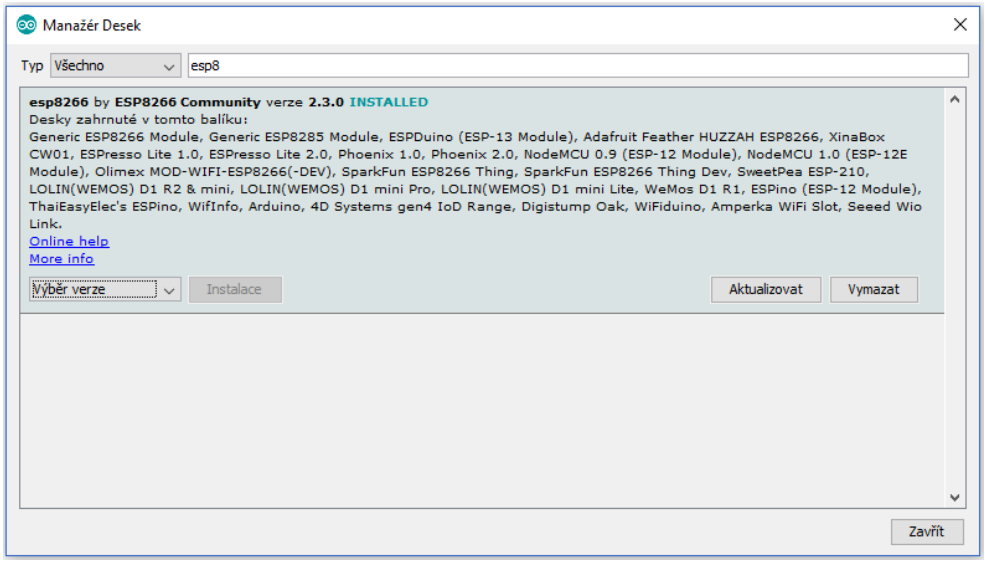
```
SerialSend.exe /baudrate 9600 /devnum 1 /hex "\xC8\x13"
```

Více o tomto nástroji je možné se dozvědět na [18].

4.2.4 Knihovna pro modul ESP8266 do Arduino IDE

Ještě, než přejdu k implementaci samotných programů, je důležité se seznámit s knihovnami, které jsou nezbytné pro implementaci aplikací a jejich nahrání do modulů, kterých se týká tato diplomová práce. První z nich je knihovna pro modul ESP8266. Tato knihovna poskytuje veškeré důležité hlavičkové soubory a zdrojové kódy nutné pro práci se všemi rozhraními, se kterými je modul schopen pracovat. Knihovna je samozřejmě vytvořena tak, aby byla kompatibilní s vývojovým prostředím Arduino IDE. Instalace této knihovny je velice jednoduchá. Stačí v hlavním menu v položce nastavení do pole označeného textem „Správce dalších desek URL“ zadat adresu `http://arduino.esp8266.com/stable/package_esp8266com_index.json`. Dále pak přejdeme na položku menu Nástroje, kde zvolíme možnost Vývojová deska a následně Manažér desek. Zde už jen stačí vyhledat příslušnou knihovnu a nainstalovat ji přesně, jak znázorňuje obrázek 4.4.

Knihovna kromě potřebných zdrojových kódů a jiných souborů poskytuje také potřebnou konfiguraci modulu nutnou ke správnému nahrávání firmware a to je tedy poslední věc, kterou je nutné vyřešit, než je možné se pustit do samotného programování. V položce menu Nástroje se opět vybere možnost Vývojová deska a v nově nainstalovaných položkách se nastaví „Generic ESP8266 Module“. Zbytek nastavení je možné ponechat jako výchozí kromě

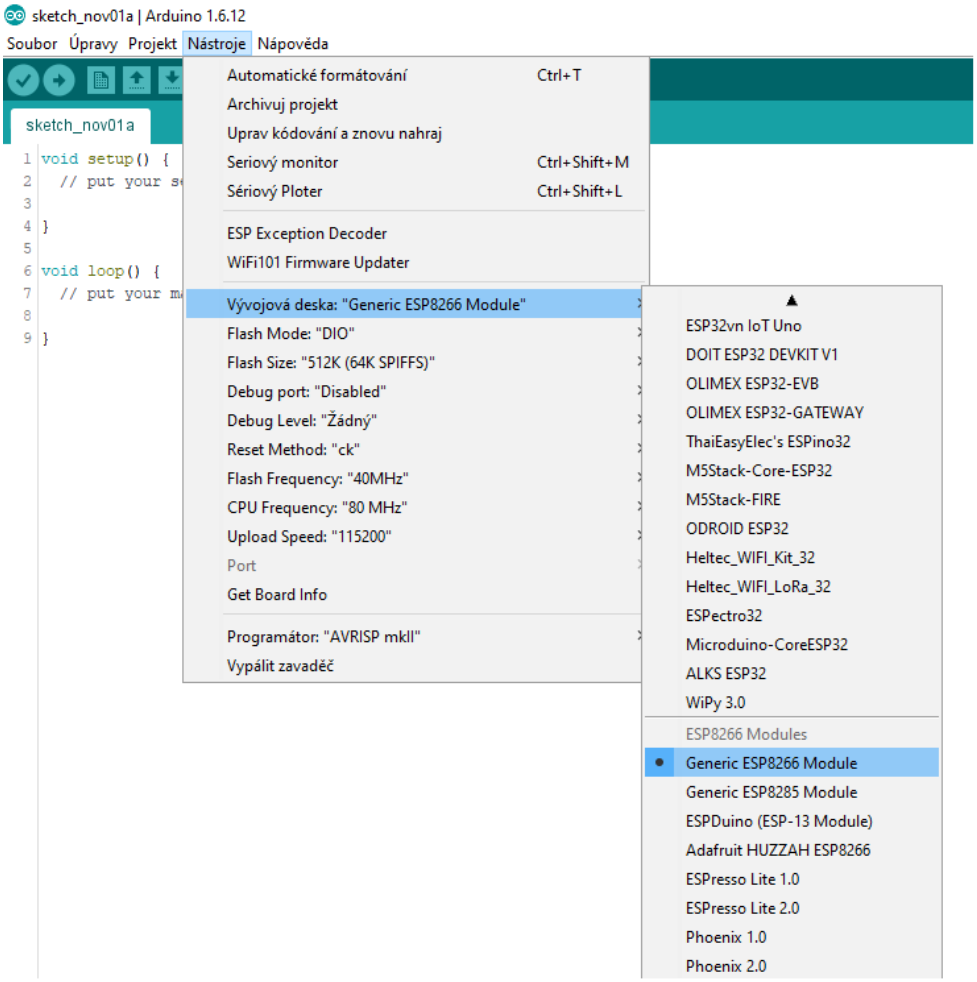


Obrázek 4.4: Instalace knihovny pro modul ESP8266 v Arduino IDE

rychlosti nahrávání, kterou, pokud tak již není nastavena, je vhodné nastavit na 115200 baudů, což je doporučená rychlost pro nahrávání firmware. Postup tohoto nastavení je patrný z obrázku 4.5.

4.2.5 Knihovna pro modul ESP32 do Arduino IDE

Také pro druhý modernější modul ESP32 existuje knihovna do vývojového prostředí Arduino IDE. Tato knihovna obsahuje opět všechny potřebné zdrojové soubory a hlavičkové soubory, které umožňují vyvíjet aplikace pro tento modul. Instalační proces může být úplně stejný jako je tomu v případě ESP8266. Podle zkušeností mnoha uživatelů ale ne vše v takto nainstalované knihovně je zcela funkční, proto jsem raději zvolil ještě jiný postup. Z adresy `https://navody.arduino-shop.cz/navody-k-produktum/vyvojova-deska-esp32.html` se dá stáhnout příslušný archiv pro modul ESP32, který se rozbalí do adresářové struktury Arduino/hardware/espressif/esp32, přičemž poslední dva adresáře musíme vytvořit. Dále poté v podadresáři „tools“ se spustí aplikace „get“, která dokončí instalaci knihovny. Konfiguraci pro nahrávání firmware jsem zvolil podle obrázku 4.6, jelikož specifikace „FireBeetle-ESP32“ nejvíce odpovídá typu ESP32, který mám k dispozici. Na závěr této sekce bych rád uvedl, že zdrojové kódy jak u knihovny pro ESP32, tak u knihovny pro ESP8266 je možné editovat, kdybychom například potřebovali přemapovat některá rozhraní na jiné piny, než je tomu ve výchozím případě apod.



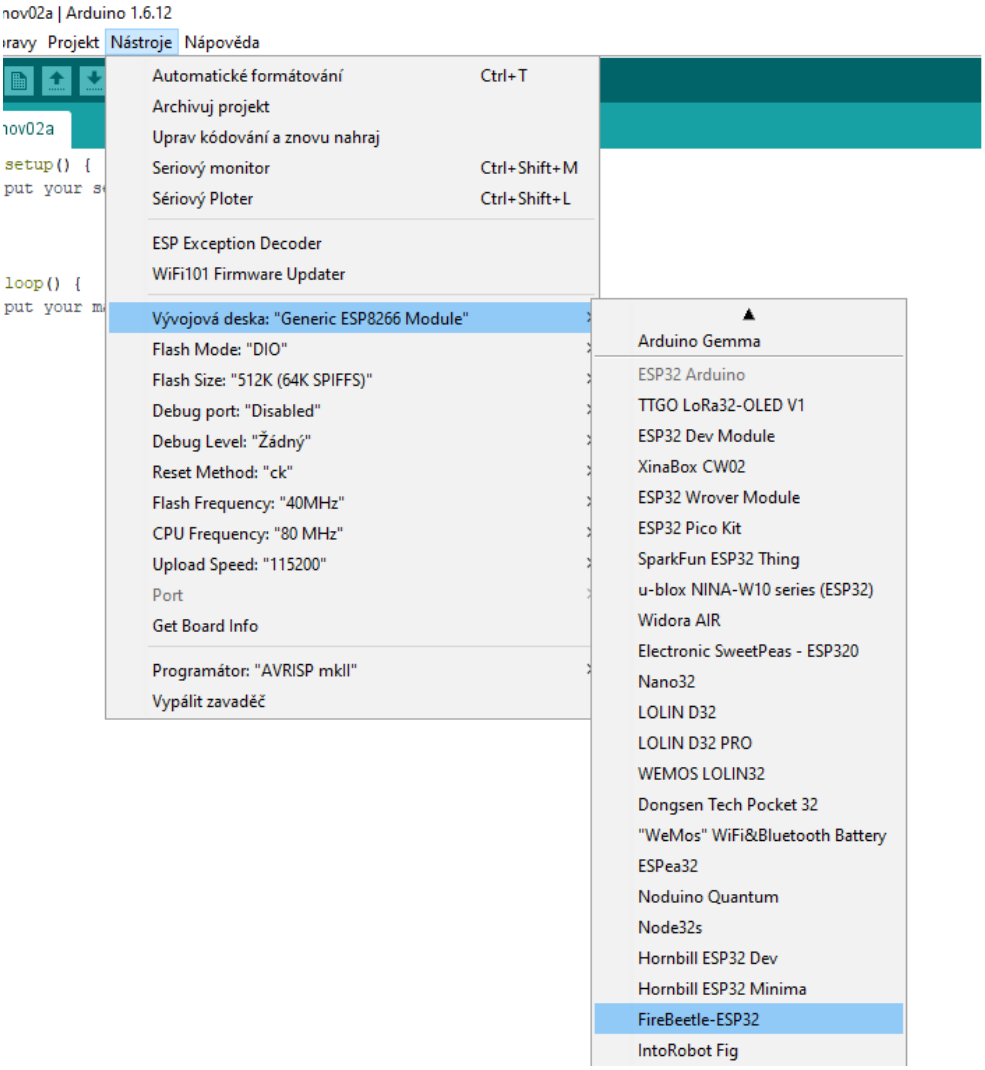
Obrázek 4.5: Konfigurace modulu ESP8266 v Arduino IDE

4.3 Implementace programu

V této části textu seznámím čtenáře s tím, jak jsem postupoval při zkoumání možností přenosu na modulech ESP. Popisuji zde hlavně implementační část všech tří programů, o kterých jsem hovořil v kapitole Návrhu. Dále jsem zde zařadil také úpravy převzatých kódů. Závěry, které z této implementace vyplynuly, jsou obsahem následující kapitoly.

4.3.1 Program pro testování sériového rozhraní UART

Jak jsem se již zmínil v sekci 3.3, u programu, který testuje rozhraní UART jsem se zaměřil na příjem dat z jiného zařízení, v mém případě ze stolního PC, odkud posílám data na příslušný COM port pomocí aplikace SerialSend. Implementace tohoto programu je velice jednoduchá a pro oba moduly je



Obrázek 4.6: Konfigurace modulu ESP32 v Arduino IDE

z aplikačního hlediska zcela totožná.

Pro práci s rozhraním UART se využívá třída Serial, která je součástí knihoven pro moduly ESP. Z příslušných metod této třídy jsem potřeboval využít pouze 4. První je metoda begin(), která inicializuje příslušné rozhraní UART. Tato metoda přijímá jako parametr baudrate, na který chceme rozhraní konfigurovat. Baudrate je datového typu unsigned long. Druhou důležitou metodou je metoda available(). Tato metoda získá počet bajtů, které jsou k dispozici pro čtení ze sériového portu. Jsou to data, která jsou již uložená v přijímacím bufferu sériové linky, který drží maximálně 64 bajtů. Třetí důležitou metodou pro tento program je metoda read(). Tato metoda má za úkol přečíst jeden bajt z receive bufferu sériové linky. Pokud je tento buffer prázdný, vrací metoda -1.

Pomocí těchto metod jsem tedy sestavil jednoduchý test rozhraní UART tak, že jsem ze stolního PC vyslal 1MB dat na příslušný COM port při daném baudrate a na straně modulu jsem tato data přijímal jako zprávy o velikosti 1KB. Tuto velikost jsem zvolil po dohodě s vedoucím práce, jelikož posledním krokem této práce je program tunelu UART-Wifi a tato velikost pro paket, který bude odeslán pomocí WiFi, se zdála ideální.

Abych mohl danou propustnost správně otestovat, musel jsem zkontrolovat, zda dojde celý 1MB a zda jsou data v těchto 1KB zprávách správná, tedy nijak nezpřeházená, nepřekrývající se nebo jestli zprávy nejsou neúplné. Co se týká testování přijatých dat přímo v modulu, mají oba moduly značně omezenou paměť, a tak není možné uložit celý 1MB, ale jen jeho určitou část. Nelze také testovat integritu dat po každém přijatém 1KB, jelikož algoritmus, který tuto integritu kontroluje, by omezoval přenos dat a přineslo by to zkreslené výsledky. Cílem tedy je vždy otestovat jen určitou porci dat po skončení celého přenosu. Bohužel tato část dat nebude nijak velká vzhledem k omezené paměti, proto jsem se rozhodl otestovat toto množství dat vzhledem k některé části zmíněného 1MB. To znamená, že odeberu vždy vzorek dat z dané části 1MB a ten následně otestuji. Tento způsob testování mi dá alespoň hrubou představu, zda v některé části dat nevznikají chyby více či méně.

Testovanou část dat také vypíši na příslušný sériový port, abych ověřil, jestli ve směru UART modulu - PC nevznikají ještě další chyby. K tomuto účelu poslouží poslední metoda, kterou jsem použil a tou je metoda println(). Ta vypíše data na sériový port v ASCII podobě a následně odřádkuje. Parametrem této metody je jakýkoliv datový typ. Existuje také varianta printf(), která, stejně jako v klasickém jazyce C, zařizuje formátovaný výstup. Vypsaná data je možné pak kontrolovat na sériovém monitoru pro příslušný COM port. Kromě vypsané části dat sleduji, zda přišla všechna data a také zda testovaná část dat na straně příjmu v modulu neobsahuje chyby. Funkci compareData(), která kontroluje data na straně příjmu, se detailněji věnuji v sekci 5.3. Jako úkázku uvádím část zdrojového kódu popisující hlavní smyčku tohoto programu.

```
void loop() {

    while ((siz = Serial.available()) > 0) {
        while(cnt < siz){
            c = Serial.read();

            if(config_flag == 1){
               baudRate[baud_pos++] = c;

              if(baud_pos == 4){
                 setBaudrate(baudRate, 4);
                 config_flag = 0;
                 baud_pos = 0;
                 data_cnt = 0;
                 KB_cnt = 0;
              }
            }
            else{

               if (c == '3'){
                   compareData(dictionary, data_buf,
                               256, data_size,
                               holes_cnt,
                               lost_bytes,
                               avg_lost_bytes);

                   Serial.println(KB_cnt);
                   Serial.println(holes_cnt);
                   Serial.println(lost_bytes);
                   Serial.println(avg_lost_bytes);
                   Serial.println(data_buf);

                   baud_pos = 0;
                   data_cnt = 0;
                   KB_cnt = 0;
                   buff_pos = 0;
                   config_flag = 1;

               }
               else{
                   if(data_cnt < data_size
                      && i > 750){

                     data_buf[data_cnt++] =
```

```
                        buff[buff_pos++] = c;
                    }
                    else{
                      buff[buff_pos++] = c;
                    }
                }
                if (buff_pos == 1024){
                    KB_cnt++;
                    buff_pos = 0;
                }
            }
            cnt++;
        }
        cnt = 0;
    }
}
```

Tento kód využívá vlastnosti metody available. Pomocí ní si zjistí počet bajtů v receive bufferu a následně jich přesně tolik načte postupně po jednom bajtu a uloží do testovacího bufferu. Volání metody available() po každém načteném bajtu není samozřejmě efektivní a pro účel této práce přináší nežádoucí výsledky, kde díky pomalému zpracování dat nelze přesáhnout rychlost 9600 baudů.

První část smyčky je zaměřena na dynamické nastavení baudrate, aby bylo možné za běhu měnit propustnost, aniž bychom museli kvůli změně nahrávat nový firmware. Ve druhé části je vidět načítání dat do 1KB bufferu a do bufferu představující testované množství dat v určité části 1MB, v tomto případě ve třetí třetině dat. Jakmile skončí přenos dat, vyšle se ukončovací znak (zvolil jsem číslici 3, protože číslice se nevyskytují v mých testovacích datech) a začne kontrola zvoleného množství dat. Na sériový port následně vypíši počet přijatých KB (daných proměnnou „KB_cnt“), dále počet vzniklých chyb, počet celkově ztracených bajtů na dané množství dat, průměrný počet ztracených bajtů na chybu a nakonec testovaná data.

4.3.2 Program pro testování WiFi

Dalším klíčovým programem je program pro testování WiFi. V dnešní době a zvláště v oblasti internetu věcí se bez WiFi, jako prostředku k přenosu dat, nedokážeme obejít. Tento program dává větší náhled na to, jak samotná WiFi na modulech ESP funguje v praxi. Pro tuto práci je zejména důležité otestovat schopnosti WiFi u modulů nastavených do módu stanice. Jelikož ani jeden modul nemá poskytovat přístup na internet ani do žádné jiné sítě, nýbrž mají sloužit jako prostředek k posílání dat, možnost módu „Access point“ jsem v této práci neuvažoval.

Program, který jsem implementoval, nejprve nastaví mód, ve kterém se modul připojí do sítě, tedy jako stanice, poté nastaví protokol (podporovány jsou IEEE 802.11 b, IEEE 802.11 g nebo IEEE 802.11 n, jak jsem uvedl v sekci 2.4 kapitoly Analýza) a následně se připojí do sítě. Pro kontrolu správného provedení těchto kroků vypisuji informace o aktuálním připojení na sériový port a zobrazuji na sériovém monitoru. Kromě těchto informací vypisuji také, zda se podařilo správně vybrat protokol, nastavit mód stanice, kolik bylo alokováno kanálů pro daný protokol atd. Pro tuto konkrétní část mi byla nápomocná metoda printDiag(Serial) třídy WiFi, která dokáže tyto informace zobrazit.

Bohužel program není pro oba moduly zcela totožný. Zatímco u modulu ESP8266 lze nastavit každý jednotlivý protokol zvlášť, ESP32 poskytuje pouze módy 802.11 b, 802.11 bg, 802.11 bgn a dva Espressif specifické módy, které jsem do této práce nezahrnul. Zatímco mód 802.11 b je na první pohled jednoznačný, u ostatních dvou to nemusí být zcela zřejmé. Podle oficiální dokumentace firmy Espressif, která je výrobcem modulu ESP32, modul v těchto módech zvolí nejlepší možný protokol podle možností modulu a AP, pomocí kterého je připojen do sítě. Detailní informace o implementaci WiFi na modulu ESP32 jsou k dispozici na [19]. Také metoda printDiag(Serial) není schopná podat všechny informace. Například status a nastavený protokol je nutné zobrazit jiným způsobem. Konkrétní podobě tohoto způsobu se věnuji v dalších částech této sekce.

Protože tento program také posílá data, bylo nutné zvolit vhodný protokol pro jejich přenos. V kapitole Návrh v sekci 3.3 jsem se zmínil, že jsem zvolil protokol UDP. Potom, co program vypíše důležité informace týkající se WiFi daného modulu, vytvoří lokální UDP port pro modul. Výběr konkrétního čísla portu jsem provedl náhodně. Poté ve smyčce pomocí tohoto protokolu aplikace odesílá UDP pakety o velikosti 1KB, které obsahují data z bufferu, který jsem předem naplnil testovacím textem. Tyto pakety jsou odesílány na IP adresu počítače, kde pomocí aplikace napsanou v jazyce C# data přijímám, počkám, až přijde 1MB dat a následně spočítám, jak dlouho trvalo přijetí tohoto 1MB dat. Takto jsem testoval všechny 3 WiFi protokoly na obou modulech. Z těchto údajů pak snadno určím reálnou propustnost pro každý protokol a zjistím, zda se alespoň tato propustnost blíží teoretickým předpokladům.

Na závěr této sekce uvádím některé důležité fragmenty zdrojových kódů programu pro oba moduly. Úplné zdrojové kódy naleznete na přiloženém CD.

Fragment zdrojového kódu pro připojení modulu ESP8266 do sítě pomocí WiFi včetně výpisu všech důležitých informací:

```
#include <WiFi.h>

WiFi.mode(WIFI_STA);

WiFi.setPhyMode(WIFI_PHY_MODE_11B);
```

```
WiFi.begin("SSID", "PASSWORD");
Serial.print("Connecting");
while (WiFi.status() != WL_CONNECTED)
{
  delay(500);
  Serial.print(".");
}
Serial.println();
Serial.printf("Current IP is %s, UDP port %d\n",
              WiFi.localIP().toString().c_str(),
              localUdpPort);
WiFi.printDiag(Serial);
```

Fragment zdrojového kódu pro připojení modulu ESP32 do sítě pomocí WiFi včetně výpisu všech důležitých informací:

```
#include <WiFi.h>
#include <esp_wifi.h>

wifi_interface_t current_wifi_interface;
uint8_t current_protocol;
esp_err_t error_code;

WiFi.mode(WIFI_STA);

esp_wifi_set_protocol(current_wifi_interface,
                      WIFI_PROTOCOL_11B);

WiFi.begin("SSID", "PASSWORD");
Serial.print("Connecting");
while (WiFi.status() != WL_CONNECTED)
{
  delay(500);
  Serial.print(".");
}
Serial.println();
Serial.printf("Current IP is %s, UDP port %d\n",
              WiFi.localIP().toString().c_str(),
              localUdpPort);

WiFi.printDiag(Serial);

error_code =
   esp_wifi_get_protocol(current_wifi_interface,
```

```
                                &current_protocol);

esp_err_to_name_r(error_code,error_buf1,100);
Serial.print("esp_wifi_get_protocol error code: ");
Serial.println(error_buf1);
Serial.print("Current protocol code is ");
Serial.println(current_protocol);

if ((current_protocol&WIFI_PROTOCOL_11B)
    == WIFI_PROTOCOL_11B)
  Serial.println("Protocol is WIFI_PROTOCOL_11B");

if ((current_protocol&WIFI_PROTOCOL_11G)
    == WIFI_PROTOCOL_11G)
  Serial.println("Protocol is WIFI_PROTOCOL_11G");

if ((current_protocol&WIFI_PROTOCOL_11N)
    == WIFI_PROTOCOL_11N)
  Serial.println("Protocol is WIFI_PROTOCOL_11N");

if ((current_protocol&WIFI_PROTOCOL_LR)
    == WIFI_PROTOCOL_LR)
  Serial.println("Protocol is WIFI_PROTOCOL_LR");
```

Jak je možné si povšimnout, hlavním rozdílem je to, že knihovna pro WiFi u modulu ESP32 již neobsahuje metodu setPhyMode() a musíme tedy využít funkci esp_wifi_set_protocol(), která je součástí esp_wifi.h. Dále, jak jsem se zmínil, ne všechny informace jsme schopni získat pomocí metody printDiag(Serial). Tyto informace však zjišťuje kód pod touto metodou. Zjistí chybový kód, pokud se nepodařilo nastavit protokol a dále získá kód protokolu, který je dále převeden a vypsán v textové podobě. Kód je koncipován tak, že pokud je nastaven protokol 802.11 BGN, potom jsou na sériový port postupně vypsány protokoly B, G a N. Pro úplnost na závěr úvádím ještě část kódu pro posílání UDP paketů. Jediný rozdíl v kódu mezi moduly ESP8266 a ESP32 je ten, že pro ESP8266 se v metodě write nemusí uvádět velikost pole, jehož obsah se má odeslat v UDP paketu. To zřejmě zjistí překladač z deklarace bufferu.

```
char buff[] = "text to send.....1024 bytes";

void loop() {
        Udp.beginPacket(remoteIp, remoteUdpPort);
        Udp.write((uint8_t *)buff, b_size);
        Udp.endPacket();
}
```

4.3.3 Program tunel UART-WiFi

Posledním programem, který testuje možnosti přenosu dat pomocí modulů ESP8266 a ESP32, je program tunelu UART-WiFi. Tato aplikace integruje testovací program pro UART a část programu testující WiFi. Tento program má za úkol přijímat data ze sériové linky, zpracovávat je a na modulech z těchto dat vytvářet UDP pakety a odesílat je na adresu protistrany (v mém případě stolního PC), která je ve stejné síti jako moduly ESP. Co se týká integrace, program testující UART zůstal prakticky beze změny. Z této části jsem odstranil buffer pro testování určitého množství dat (větších než 1KB) a také funkci, která kontrolovala správnost dat uvnitř tohoto bufferu. Tato funkce totiž na modulech ESP8266 a ESP32 nebude třeba, protože všechna data jsou odchytávána pomocí aplikace Packet Sender a jsou kontrolována jako celek na stolním PC stejnou funkcí. Také veškeré výpisy na sériový port jsou vynechány. Místo toho je ponechán buffer o velikosti 1KB, ze kterého se vytvoří UDP paket, který se následně odešle do WiFi sítě. Z původního programu pro testování WiFi jsem odstranil předpřipravený buffer s testovacími daty, jelikož potřebná data poskytne sériová linka. Odesílání paketů je ponecháno v hlavní smyčce programu v kombinaci s přijímáním dat ze sériové linky. Výpis informací o nastavení WiFi a připojení k síti jsem ponechal. Samozřejmě část týkající se WiFi je odlišná pro oba moduly přesně tak, jak jsem uvedl v minulé sekci a část pro UART je pro oba moduly stejná.

Takto připravený program je schopen testovat maximální propustnost celku s ohledem na to, že je možné nastavovat baudrate rozhraní UART a také různé protokoly WiFi podle potřeby. Toto dobře poslouží ke stanovení limitů propustnosti, při které jsou data stále neporušená. Také lze sledovat procento chybovosti, pokud se propustnost začne zvyšovat nad stanovený limit.

Následující část zdrojového kódu zobrazuje zmíněnou hlavní smyčku programu (úplný zdrojový kód tohoto programu zájemce nalezne na přiloženém CD):

```
while ((siz = Serial.available()) > 0) {
    while(cnt < siz){
        c = Serial.read();

        if(config_flag == 1){
            baudRate[baud_pos++] = c;

            if(baud_pos == 4){
                setBaudrate(baudRate, 4);
                config_flag = 0;
                baud_pos = 0;
                buff_pos = 0;
            }
        }
```

```
        else{
            buff[buff_pos++] = c;

            if (buff_pos == 1024){
                Udp.beginPacket(remoteIp, remoteUdpPort);
                Udp.write((uint8_t *)buff, buff_size);
                Udp.endPacket();
                buff_pos = 0;
            }
        }
        cnt++;
    }
    cnt = 0;
}
```

Použité proměnné „remoteIp“ a „remoteUdpPort“ jsou deklarovány na začátku programu. Ostatní části programu zde neuvádím, protože jsou stejné, jako v původních programech testujících UART a WiFi a jsou uvedené v předchozích sekcích této kapitoly.

Při implementaci všech programů a jejich následným testování jsem se dozvěděl mnoho zajímavých informací ohledně testovaného hardware i software. Také jsem při implementaci a sestavování obvodů narazil na různá úskalí. O tom všem už ale pojednává následující kapitola.

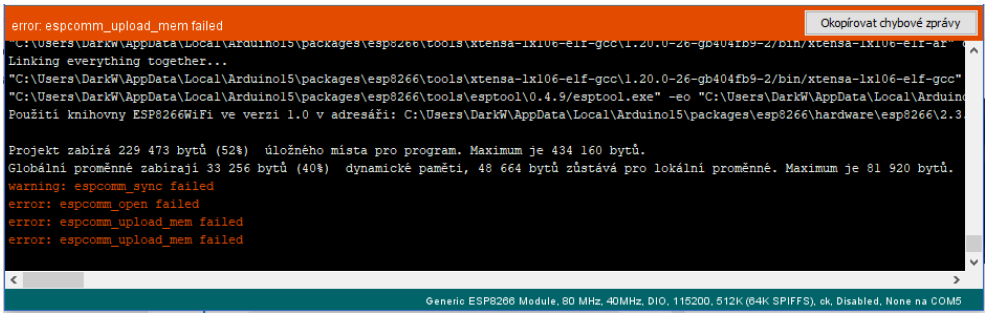
Kapitola 5

Pozorování a výsledky

Tato kapitola pojednává o výsledcích pozorování, ke kterým jsem dospěl v průběhu implementace a testování hotového software. Je zde popsáno, jak jsem při testování postupoval, jaké nástroje jsem vybíral pro testování a k jakým limitům jsem došel. Také zde popisuji porovnání praktických výsledků pozorování a teoretických limitů popsaných v příslušných dokumentech daných hardwarových součástek. V neposlední řadě se čtenář dozví, s jakými problémy jsem se při sestavení obvodů a implementaci programů potýkal a jakými způsoby jsem je vyřešil.

5.1 Napájení modulů ESP8266 a ESP32

První věcí, kterou bylo potřeba vyřešit, byl kvalitní napájecí zdroj. Oba moduly sice lze napájet přímo z USB, nicméně je potřeba si dát pozor na odběrové špičky obou modulů. V datasheetu pro ESP32 dostupného na [20] je doporučeno, aby výstupní proud napájecího zdroje byl nejméně 500mA. V datasheetu pro ESP8266 to takto jednoznačně popsáno není, avšak z přiložené tabulky pro spotřebu a z pozorování při nahrávání některých programů lze usuzovat, že je tato hranice podobná. USB verze 2.0 poskytuje výstupní proud právě 500mA, což by teoreticky stačit mělo, ale v praxi to ve většině případů zejména u modulu ESP8266 nestačí. Pokud totiž například modul potřebuje využít WiFi, stává se, že pro úspěšné připojení je potřeba odebrat více, než 500mA. Tato proudová špička zapříčíní nižší napětí, než je stanovená operační mez. Tuto situaci detekuje speciální obvod, který se jmenuje „watchdog“ a modul z této příčiny restartuje. Protože po restartu se spustí nahraná aplikace znovu, nastane znovu pokles napětí a tento typ restartu se opakuje do nekonečna. Z tohoto důvodu je dobré mít kvalitní napájecí zdroj, který poskytuje výstupní proud větší, než je zmiňovaných 500mA, jinak je jakákoliv práce s moduly ESP velmi obtížná. Pro tuto práci jsem tedy nakonec zvolil napájecí zdroj MEAN WELL PM-05-3,3, který poskytuje výstupní proud 1,25 A. Zdroj tento pro-



Obrázek 5.1: Výpis kompilátoru při opětovném nahrání firmware pro modul ESP8266 v Arduino IDE

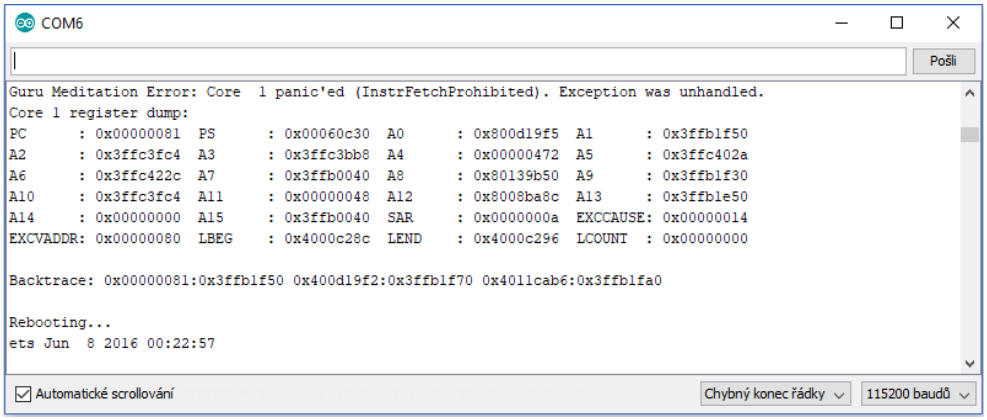
blém nakonec odstranil. Přestože modul ESP32 tímto mírným nedostatkem tolik netrpí, využil jsem tento zdroj i při zapojení tohoto modulu.

5.2 Úskalí implementace programů na modulech ESP a jejich řešení

Žádný program nelze napoprvé naprogramovat zcela správně a bez chyb a tedy i já jsem se při implementaci testovacích programů potýkal s několika problémy, které jsem musel odstranit. V této sekci prezentuji pouze několik těchto problémů, u kterých si myslím, že způsob jejich řešení, který jsem aplikoval by mohl v budoucnu pomoci programátorům, kteří nejsou příliš obeznámeni s moduly ESP8266 a ESP32 a také s vývojovým prostředím Arduino IDE. Popisuji zde i některé vlastnosti modulů, které jsem doposud v této práci nezmínil.

Nejprve bych rád zmínil rozdíl v procesu nahrávání nového firmware u modulů ESP8266 a ESP32. U modulu ESP32 nejde prakticky o nic zvláštního. Po připojení USB kabelu se klasicky nahraje nový firmware do modulu a pokud je funkční, program se nahraje do paměti. Pokud je nutné tento program nahradit jiným, je možné tak bez potíží učinit novým nahráním firmware, aniž by bylo potřeba odpojovat modul od počítače či jinak měnit nastavení modulu. U modulu ESP8266 to však takto není možné provést. Pokud se provede náhrada starého firmware novým, v okně pro výpisy kompilátoru se objeví zpráva, která je vidět na obrázku 5.1.

Toto se dá jednoduše odstranit tím, že se odpojí napájení od modulu ESP8266, znovu připojí a firmware se nahraje. Po odpojení napájení ale také dochází ke smazání původního programu z paměti ESP8266. To je velmi nepraktické, zvláště pak u aplikací, které běží 24 hodin denně a kde krátkodobý výpadek napájení způsobí vymazání programu a následně je vyžadován vnější zásah a opětovné nahrání firmware. Oproti tomu ESP32 těmito nedostatky netrpí, program je v paměti uložen i po odpojení napájení. Odpojení a opě-



Obrázek 5.2: Příklad pádu aplikace na modulu ESP32 v Arduino IDE

tovné připojení pak modul ESP32 vnímá pouze jako restart podobně jak je tomu u obyčejných stolních počítačů.

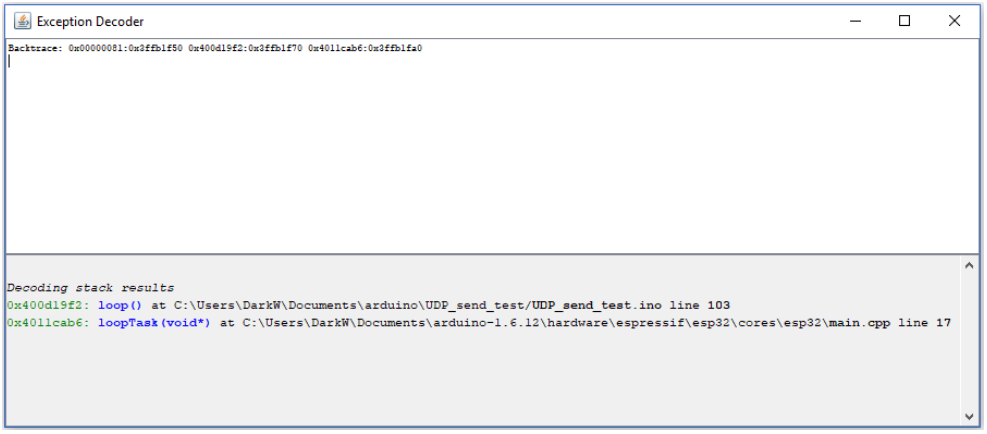
Na závěr této sekce bych rád zmínil, jak nejlépe řešit ve vývojovém prostředí Arduino IDE velmi častý problém všech programátorů, se kterým jsem se během implementace také potýkal. Tím je pád aplikace. Bohužel při programování vestavných systémů či internetu věcí a v některých vývojových prostředí není příliš snadné programy takzvaně debuggovat. Pro vývojové prostředí Arduino IDE a pro moduly ESP však existuje nástroj, který se jmenuje ESP exception decoder.

Jestliže dojde k pádu aplikace, moduly ESP vypíší na sériový monitor nejprve obsah zásobníku, který obsahuje inkriminované adresy paměti, kde k pádu došlo, a následně modul restartuje. Obrázek 5.2 je ukázkou příkladu pádu programu na modulu ESP32. V tomto případě došlo k provedení instrukce na neplatné adrese a tedy byla vyvolána výjimka a došlo k restartu modulu. Je zde vidět i výpis obsahu registrů modulu a dále pak už zmiňovaný obsah zásobníku uvozený slovem „backtrace“.

Tento backtrace je pak vstupem pro nástroj ESP exception decoder. Příklad, jak vypadá tento nástroj je patrný z obrázku 5.3. Do horní poloviny okna uživatel vloží backtrace, který poskytl modul ESP (musí se vložit i s klíčovým slovem „backtrace:“, jinak nástroj vstup nedokáže zpracovat) a následně se ve spodní části objeví funkce a konkrétní řádek zdrojového kódu, kde k pádu aplikace došlo.

5.3 Testování rozhraní UART

Testování rozhraní UART u obou modulů je ve své podstatě velmi jednoduché. Jde o to především otestovat, zda funguje přenos dat jako takový, jestli přijdou všechna data a zkontrolovat alespoň některý z přijatých bufferů, zda obsahuje ta data, která byla na sériový port odeslána.



Obrázek 5.3: Příklad funkce nástroje ESP exception decoder v Arduino IDE

Z aplikačního hlediska je testování rozhraní UART poněkud omezené několika faktory. Tím prvním je terminál, na kterém je možné pozorovat přenos dat. Ať už se použije jakýkoliv terminál, vždy je možné zde nastavit pouze několik vybraných propustností a ne jakoukoliv, která by byla pro pozorování vhodná. Terminálů pro sledování sériových portů existuje celá řada, bohužel mnoho z nich je omezeno pouze na 250Kbit/s, což v mém případě představuje problém. Nicméně lze použít interní sériový monitor vývojového prostředí Arduino IDE, který má od novějších verzí škálu propustností poměrně slušnou. Bohužel i zde, jakmile je potřeba nastavit vyšší propustnost, než je škála standardizovaných (to jest 1Mbit/s a vyšší), máme k dispozici jen tři rychlosti a to 1, 2 a 3Mbit/s a nelze využít nějaký mezistupeň například mezi 1 a 2Mbit/s. O něco lépe tuto situaci řeší Parallax Serial Terminal, se kterým jsem měl tu možnost se setkat, který nabízí 1,5Mbit/s, ale pak také nic navíc.

U ESP8266 dále komplikuje situaci fakt, že k jedinému rozhraní UART je připojen převodník CP2104, pomocí kterého se nahrává i nový firmware, a na jehož COM portu nelze zároveň odesílat data a zároveň sledovat přijímaná data. Tudíž je nutné vypsat informace o tom, jak přenos dopadl, až po dokončení přenosu. Informace, které si necháme vypsat, jsou: počet zpráv o velikosti 1KB, kterých by mělo být přesně 1024 (pro data o velikosti 1MB), namátkou vybraný úsek přijatých dat uložených do bufferu v modulu a také informace o chybách vzniklých při příjmu dat do tohoto bufferu. Vzhledem k ne příliš velké paměti modulu nelze uložit všechna data. Experimentálně jsem zjistil, že na modulu ESP8266 lze uložit maximálně 26KB dat. Bohužel práce s takto velkým bufferem způsobuje nestabilitu modulu, proto jsem musel velikost bufferu zmenšit až na 20KB.

ESP32 má o něco lepší možnosti. Přijímač a vysílač rozhraní UART0 jsou vyvedeny na piny modulu. Kromě toho lze přistupovat na UART0 i přímo přes USB kabel. To dává možnost přistupovat k jednomu rozhraní přes 2 různé virtuální COM porty. Toho jsem také využil při zkoumání možností sériové

linky. Lze samozřejmě využít i UART1 nebo UART2, ale potom už není možné takto sledovat data přímo v průběhu vysílání a možnosti jsou pak stejné jako u ESP8266. Při využití UART1 je také nutné dbát na to, že piny, na které je namapován přijímač a vysílač, jsou napojené na flash paměť modulu a tedy modul nedovoluje manipulaci s těmito piny. Přijímač a vysílač je v tomto případě pak nutné softwarově přemapovat na některé z volných GPIO pinů. I na tomto modulu jsem vytvořil buffer s co možná největší velikostí. Paměť tohoto modulu ale není o tolik větší, než v případě ESP8266. Největší velikost bufferu, kterou jsem mohl zvolit tak, aby nezpůsobovala nestabilitu, je 90KB.

Nyní již k samotnému měření. Při přijímání dat jsem si zavedl proměnnou, která počítá přijaté KB. Vzhledem k tomu, že posílám celý 1MB dat, počet KB, který napočítá tato proměnná, musí být 1024. Tato proměnná má z tohoto aplikačního pohledu největší váhu. Pomůže totiž stanovit přibližnou mez propustnosti pro bezchybný přenos dat. Pro určení chyb v přijatých datech na modulech jsem naprogramoval funkci compareData() jejíž zdrojový kód uvádím níže:

```
void compareData(char dictionary[256][4], char * data_buf,
                 int row_size, int data_size,
                 int & holes_cnt, int & lost_bytes,
                 int & avg_lost_bytes)
{
  int ok_flag;
  int err_flag = 0;
  int last_good_idx = 0;
  int new_good_idx = 0;

  for(int i = 0; i < data_size - 4; i++){
    ok_flag = 0;
    for(int j = 0; j < row_size; j++){
      if(strncmp(&data_buf[i], dictionary[j], 4) == 0){
        if(err_flag == 0){
          last_good_idx = j;
        }
        else{
          new_good_idx = j;

          if(last_good_idx >= 252){
            last_good_idx -= 256;
          }

          if(new_good_idx < last_good_idx){
              new_good_idx += 256;
          }
```

```
            holes_cnt++;

            lost_bytes += new_good_idx -
                          (last_good_idx + 4);
            last_good_idx = new_good_idx;
            err_flag = 0;
          }
          ok_flag = 1;
          break;
        }
      }
      if(ok_flag == 0){
        err_flag = 1;
      }
    }
    if(holes_cnt > 0)
      avg_lost_bytes = lost_bytes / holes_cnt;
    else
      avg_lost_bytes = 0;
}
```

Tato funkce porovnává testovaná data vůči předem připravenému slovníku. Tento slovník obsahuje všechny možné čtveřice písmen, které v datech mohou nastat. Velikost čtyři jsem zvolil záměrně proto, protože každá čtveřice písmen je v testovacím textu unikátní. Menší spojení písmen již unikátní není. Algoritmus tedy hledá shody čtveřic znaků a postupně si ukládá index poslední shodné čtveřice. Pokud algoritmus pro danou čtveřici znaků v datech nenalezne shodu, zaznamená chybu a v datech nalezne první následující čtveřici, která vyhovuje a uloží si její index. Z rozdílů indexů pak algoritmus určí počet bajtů, které se v rámci této jedné chyby ztratily. Nakonec algoritmus ještě spočítá průměrný počet ztracených bajtů na 1 chybu. Tento algoritmus má ale své nevýhody. Jestliže se mezi 2 chybami vyskytnou 1-3 znaky, které do kontextu patří, algoritmus je ignoruje a započítá do ztracených bajtů. Jestliže se tato situace vyskytuje v datech velmi často, algoritmus poskytuje velmi nepřesné výsledky. Z pozorování dat ale vím, že často tato situace nenastává, proto je tento algoritmus dostačující. Po provedení algoritmu jsou všechny příslušné údaje vypsány na sériový port. Dále jsem ještě nechal vypsat obsah testovaného bufferu (o velikosti 20KB v případě ESP8266 a o velikosti 90KB v případě ESP32), abych měl představu, jaké chyby nastávají v průběhu přenosu při vyšších rychlostech i ve směru UART modulu - PC.

Při pozorování jsem postupoval tak, že do testovacího bufferu jsem vždy postupně uložil data z první, druhé a třetí třetiny vstupního 1MB dat. Při dané propustnosti jsem pro každý vzorek dat provedl deset pozorování. Pro

Propustnost v Mbit/s	Počet chyb	ztracených bajtů	vypsaných bajtů
1,5	0	0	20480
2,0	3	127	19922
3,0	26	3374	2162

Tabulka 5.1: Hodnoty z pozorování ve směru UART modulu ESP8266 - PC pro první třetinu vstupních dat

Propustnost v Mbit/s	Počet chyb	ztracených bajtů	vypsaných bajtů
1,5	0	0	20480
2,0	2	66	20261
3,0	25	3218	2146

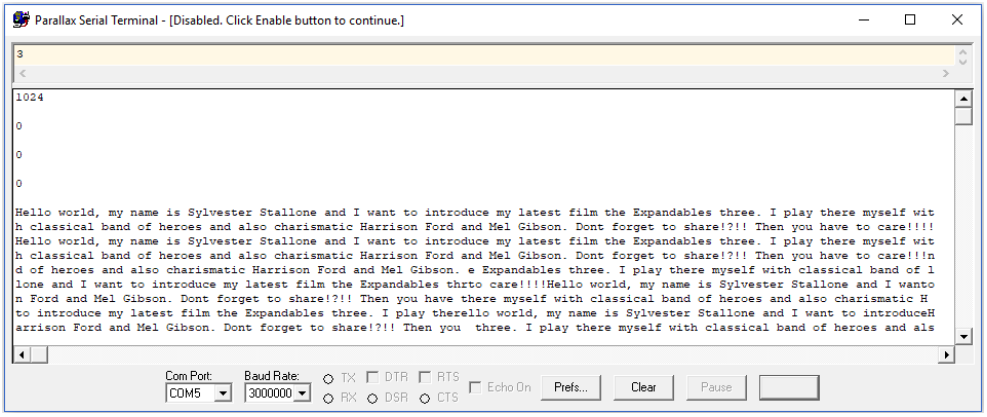
Tabulka 5.2: Hodnoty z pozorování ve směru UART modulu ESP8266 - PC pro druhou třetinu vstupních dat

Propustnost v Mbit/s	Počet chyb	ztracených bajtů	vypsaných bajtů
1,5	0	0	20480
2,0	2	57	20372
3,0	26	3448	2162

Tabulka 5.3: Hodnoty z pozorování ve směru UART modulu ESP8266 - PC pro třetí třetinu vstupních dat

ESP8266 jsem ve vzorcích přijatých dat nezaznamenal žádné chyby až do propustnosti 6,1Mbit/s. Při propustnosti 6,2Mbit/s ESP8266 již nereaguje na vyslané řídící znaky, které jsem do svých programů zařadil, abych měl kontrolu nad okamžikem výpisu údajů z přenosu dat. Z tohoto pozorování jsem usoudil, že zřejmě došlo k dosažení hardwareového limitu a modul ESP8266 už není schopen přenosovou rychlost 6,2Mbit/s nastavit. To se následně potvrdilo při testování tunelu UART-WiFi, o kterém pojednává sekce 5.5. Jako řídící znaky jsem pro jednoduchost zvolil číslici 3 pro výpis potřebných údajů a číslici 2 pro přenastavení baudrate, který je podporován terminálem, abych mohl vizuálně ověřit výsledky výpočtu.

Ve směru UART modulu - PC je situace však poněkud jiná. Drobné chyby se začaly objevovat při propustnosti 2Mbit/s a při 3Mbit/s obsahovaly vzorky už velmi mnoho chyb. Tato situace se dá vysvětlit tím, že převodník CP2104 má maximální teoretickou propustnost 2Mbit/s a tudíž jeho hardware vyšší rychlosti, než 2Mbit/s, nezvládá. Pro názornost přikládám tabulky 5.1, 5.2 a 5.3, které obsahují naměřené hodnoty pro směr UART modulu - PC pro jednotlivou část vstupních dat. Podotýkám, že se jedná o průměr z pozorování pro jednotlivé vzorky dat. Obrázek 5.4 znázorňuje výpis údajů při propustnosti 3Mbit/s.



Obrázek 5.4: Ukázka pozorování přenosu dat pomocí UART modulu ESP8266 při rychlosti 3Mbit/s

třetina vstupních dat	Počet chyb	ztracených bajtů	přijatých KB
1/3	359	1567	918
2/3	363	2460	916
3/3	364	3256	915

Tabulka 5.4: Hodnoty měření při propustnosti 3Mbit/s při příjmu na modulu ESP32

Co se týká modulu ESP32, je situace o něco málo horší. Zatímco u modulu ESP8266 na příjmu chyby nevznikají, u ESP32 nastávají problémy u propustnosti 3Mbit/s. V testovaném bufferu nastává poměrně mnoho chyb a zároveň modul z 1024KB přijme pouze okolo 918KB. Ve směru UART modulu - PC vznikají poměrově podobné chyby jako u ESP8266. Tabulka 5.4 znázorňuje chybovost při propustnosti 3Mbit/s při příjmu na modulu ESP32. Poslední sloupec znázorňuje celkový počet přijatých KB ze vstupních dat. Ostatní hodnoty souvisí s testovacím bufferem o velikosti 90KB. Jak je vidět, data ke konci přenosu obsahují více chyb než data ze začátku přenosu.

Dále přikládám tabulky 5.5, 5.6 a 5.7 znázorňující pozorované hodnoty ve směru UART modulu - PC pro modul ESP32. Jak je vidět, opět při rychlosti 2Mbit/s nastávají drobnější chyby, avšak při rychlosti 3Mbit/s opět vzniká velmi mnoho chyb.

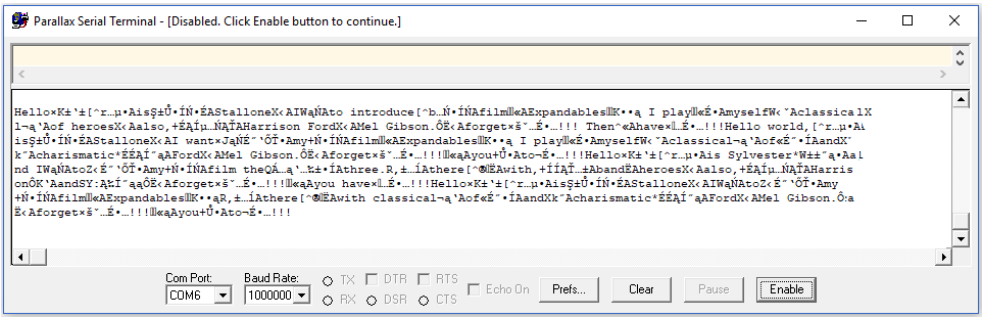
Jak jsem se již zmínil, ESP32 nám dává také možnost posílat data i přímo přes USB kabel a sledovat UART0 pomocí druhého virtuálního COM portu. Ukázalo se ale, že od rychlosti 1Mbit/s a vyšší se zobrazují nesmyslná data a netisknutelné znaky. To je dáno tím, že interním převodníkem ESP32 je převodník CP2102, který má teoretický limit 1Mbit/s, čehož v praxi fakticky ani nedosahuje. Poslední rychlostí, kterou je tento převodník schopen zvládnout, je 921,6Kbit/s. Na obrázku 5.5 je výpis obsahu dat v bufferu při rychlosti

Propustnost v Mbit/s	Počet chyb	ztracených bajtů	vypsaných bajtů
1,5	0	0	92160
2,0	7	410	91503
3,0	141	14916	7834

Tabulka 5.5: Hodnoty z pozorování ve směru UART modulu ESP32 - PC pro první třetinu vstupních dat

Propustnost v Mbit/s	Počet chyb	ztracených bajtů	vypsaných bajtů
1,5	0	0	92160
2,0	7	452	91561
3,0	145	15119	7849

Tabulka 5.6: Hodnoty z pozorování ve směru UART modulu ESP32 - PC pro druhou třetinu vstupních dat



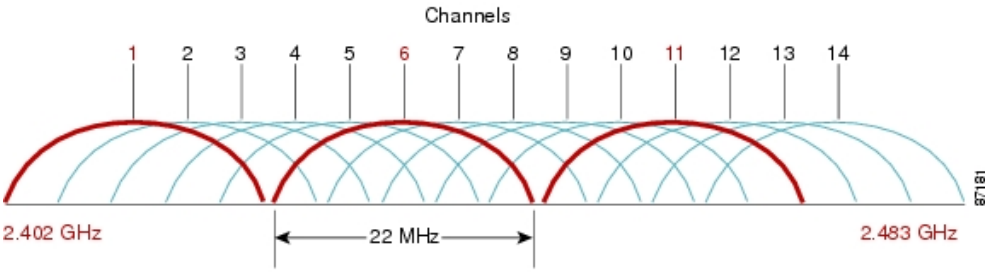
Obrázek 5.5: Ukázka pozorování přenosu dat pomocí UART modulu ESP32 při rychlosti 1Mbit/s za použití interního převodníku USB-UART

1Mbit/s pozorovaných na COM portu svázaných s převodníkem CP2102.

Toto pozorování mi vytvořilo představu, kde se zhruba pohybují limity propustností, které jsou klíčové pro program tunelu UART-WiFi. Při pozorování tohoto tunelu jsem pak získal mnohem přesnější výsledky, kde jsem také mohl pozorovat všechna data a mohl jsem tak určit jistou chybovost na celém objemu dat.

Propustnost v Mbit/s	Počet chyb	ztracených bajtů	vypsaných bajtů
1,5	0	0	92160
2,0	8	665	91179
3,0	141	14132	7809

Tabulka 5.7: Hodnoty z pozorování ve směru UART modulu ESP32 - PC pro třetí třetinu vstupních dat



Obrázek 5.6: Schéma alokací WiFi kanálů

5.4 Testování WiFi

Při tomto druhu testování je potřeba se zaměřit zejména na praktické schopnosti WiFi. Tedy zda se dají nastavit různé protokoly WiFi, kolika kanály disponují moduly a jiné vlastnosti.

Jak už jsem uvedl v kapitole Implementace v sekci Program pro testování WiFi, jedním z hlavních rozdílů v implementaci API pro WiFi je způsob nastavení jednotlivých protokolů. Zatímco u ESP8266 se dají protokoly nastavit každý zvlášť, u ESP32 jsou povolené jen módy IEEE 802.11B, IEEE 802.11BG a IEEE 802.11BGN. Ve zdrojovém souboru esp_wifi_types.h jsou pak tyto protokoly definovány jako WIFI_PROTOCOL_11B, WIFI_PROTOCOL_11G a WIFI_PROTOCOL_11N, přičemž každý má přiřazené číslo mocniny dvou řazené vzestupně. To znamená, že pokud je nastaven mód IEEE 802.11BGN, číslo pro tento mód je bitovým součtem jednotlivých protokolů a tedy hodnota tohoto módu je 7. Jelikož na ESP8266 jsou tyto protokoly řešeny jednodušeji, všem třem protokolům byla přiřazena čísla 1-3. Textová podoba těchto protokolů je pak WIFI_PHY_MODE_11B, WIFI_PHY_MODE_11G a WIFI_PHY_MODE_11N.

Další důležitou vlastností WiFi jsou kanály. V podstatě jde o určitou šířku pásma, ve které zařízení v síti operují. Standard IEEE 802.11 definuje celkem 14 kanálů pro low power WiFi komunikaci. Tyto kanály jsou pak od sebe rozmístněny po 5MHz kromě kanálu číslo 14, který je od kanálu čísla 13 vzdálen 12MHz. Toto je však nelicencované pásmo a obyčejně v módech 802.11 b/g/n je typický požadavek na operační pásmo 20MHz. Proto je potřeba brát v úvahu jistá omezení. Jestliže jsou WiFi kanály umístěny 5Mhz od sebe a kanál typicky vyžaduje 20MHz pásmo, může docházet k rušení. Proto pokud se nastaví moduly ESP jako AP, je dobré jeden modul nastavit například na kanál číslo jedna a druhý na kanál číslo šest. Pro ilustraci uvádím obrázek 5.6, který je možné společně s dalšími užitečnými informacemi nalézt na [21].

Dalším omezením je legislativa zemí. Kanály 1 až 11 mohou být bezpečně používány ve většině zemí (například v celé Evropě jsou povoleny), ale například kanál číslo 14 není v Evropě ani ve Spojených Státech povolený. Více

detailů o WiFi kanálech je možné se dozvědět na [22]. Co se týká hardwarových možností modulů ESP, tak pro ESP8266 i pro ESP32 platí, že jsou limitovány pouze na jeden kanál, takže možnost takzvaného „dual band“ zde nepřipadá v úvahu. Oba moduly se však dají nastavit do módu AP + Station. Situace se pak jednoduše řeší tak, že číslo kanálu pro AP je stejné jako pro Station.

Další důležité informace pak poskytuje metoda třídy WiFi printDiag(). Jak jsem uvedl v kapitole Implementace, tato metoda poskytne uživateli důležité informace o aktuálním WiFi připojení. V tomto případě se hodí praktická ukázka. Na obrázku 5.7 je uveden příklad výpisu diagnostiky WiFi pro modul ESP8266. Metoda vypsala, že modul je nastaven v módu stanice, protokol je nastaven na IEEE 802.11b. Dále pak byl stanici přidělen kanál číslo 6. Na tomto kanálu operuje AP, ke kterému byl ESP8266 připojen, tedy k mé testovací WiFi. Potom je uvedeno ID příslušného AP a dále položka Status. Tato položka ale není návratovou hodnotou metody status() třídy WiFi, jak by se mohlo na první pohled zdát, nýbrž tuto hodnotu vrací vestavěná funkce wifi_station_get_connect_status(), která má návratové kódy zcela jiné. Zatímco metoda status vrací jeden z těchto návratových kódů číslovaných od nuly:

```
enum {
  WL_IDLE_STATUS,
  WL_NO_SSID_AVAIL,
  WL_SCAN_COMPLETED,
  WL_CONNECTED,
  WL_CONNECT_FAILED,
  WL_CONNECTION_LOST,
  WL_DISCONNECTED
};
```

Funkce wifi_station_get_connect_status vrací zcela jiné návratové kódy také číslované od nuly:

```
enum{
  STATION_IDLE = 0,
  STATION_CONNECTING,
  STATION_WRONG_PASSWORD,
  STATION_NO_AP_FOUND,
  STATION_CONNECT_FAIL,
  STATION_GOT_IP
};
```

Jak je vidět, status s hodnotou 5 neznamená WL_CONNECTION_LOST, nýbrž STATION_GOT_IP a tedy s připojením je vše v pořádku. Ostatní hodnoty a další detaily jsou k dispozici na [23]. Nakonec jsou zde ještě doplňující

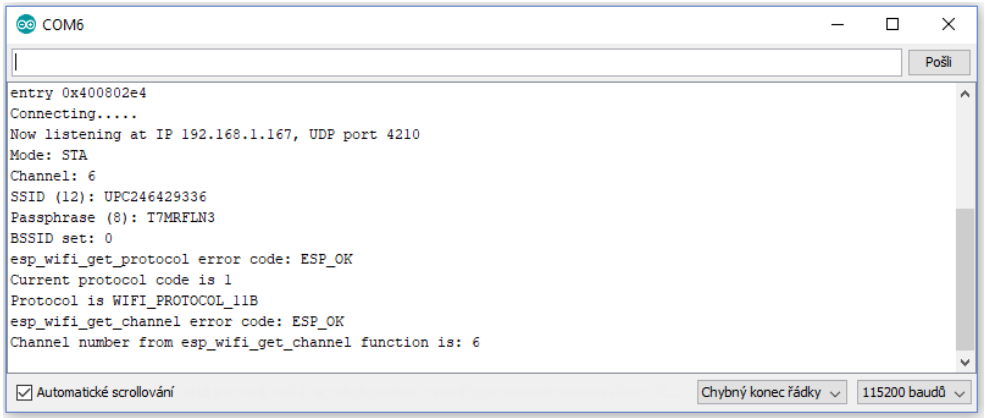
Obrázek 5.7: Příklad výpisu informací o WiFi připojení pro modul ESP8266

údaje jako SSID AP, ke kterému je modul připojen, heslo nebo například zda je nastaveno automatické připojování k síti.

V implementaci WiFi API pro modul ESP32 jsou oproti API pro ESP8266 drobné rozdíly. Kromě již zmíněných konfigurací protokolů, je například z implementace metody printDiag odstraněn výpis ID daného AP a funkce wifi_station_get_connect_status je z knihoven pro ESP32 zcela odstraněna. Je to dáno tím, že návratové kódy jsou nyní jednotné pro všechny funkce specifické pro ESP32 týkající se WiFi a funkce pro kontrolu statusu připojení zůstala pouze jedna a to metoda status třídy WiFi, jíž náleží příslušné návratové kódy platné i pro ESP8266. Metoda printDiag také už nevrací nastavený protokol. Ten se nyní musí zjistit samostatně a to pomocí metody esp_wifi_get_protocol. Ta kromě zjištěného kódu protokolu vrací jeden z následujících návratových kódů:

```
enum{
  ESP_OK,
  ESP_ERR_WIFI_NOT_INIT,
  ESP_ERR_WIFI_IF,
  ESP_ERR_INVALID_ARG
};
```

V tomto případě jsou tyto návratové kódy číslovány od jedničky, přičemž ESP_OK a ESP_ERR_INVALID_ARG se nevztahují pouze na funkce týkající se WiFi, ale také například na funkce určené pro práci s rozhraním I2S. Všechny popsané informace o WiFi na modulu ESP32 jsou patrné z obrázku 5.8. Na závěr jsem ještě využil funkce esp_wifi_get_channel a vypsal tak samostatně číslo kanálu. Tato funkce je začleněna také v metodě printDiag. Jak je vidět, stejně jako pro ESP8266, i zde bylo ověřeno, že mému AP náleží kanál číslo 6.



Obrázek 5.8: Příklad výpisu informací o WiFi připojení pro modul ESP32

Nakonec zbývalo ještě ověřit praktickou datovou propustnost všech protokolů pro oba moduly. V případě ESP32 se tak dá ověřit, zda nastavení IEEE 802.11BG a IEEE 802.11BGN vybere skutečně ten nejlepší možný protokol, který má k dispozici. Pro oba moduly a pro každý protokol jsem provedl 10 nezávislých pozorování. K měření jsem vytvořil malou aplikaci v jazyce C#, která poslouchá na mnou zvoleném portu. Aplikace poté přijme 1MB dat a změří, jak dlouho přenos trval. Z hodnot jsem následně stanovil praktickou propustnost protokolů. Zdrojový kód této aplikace přikládám níže:

```
class Program
{
  static void Main(string[] args)
  {
    /* Inicializace UDP klienta
      - predava se cislo portu */
    UdpClient udpClient = new UdpClient(59333);
    try
    {
      int dataLength = 0;
      byte[] receiveBytes;
      DateTime start = DateTime.Now;
      DateTime end = DateTime.Now;

      Console.WriteLine("Listening...");

      IPEndPoint RemoteIpEndPoint
          = new IPEndPoint(IPAddress.Any, 0);

      /* Prijem 1MB dat*/
```

```
            while (true)
            {
              /* Prijem 1KB dat */
              receiveBytes
                 = udpClient.Receive(ref RemoteIpEndPoint);
              if (dataLength == 0)
              {
                start = DateTime.Now;
              }
              dataLength += receiveBytes.Length;
              if (dataLength >= 1025*1024)
              {
                end = DateTime.Now;
                break;
              }
            }
            /* Kontrolni vypis
               posledniho paketu a jeho delky */
            string returnData
                  = Encoding.ASCII.GetString(receiveBytes);
            Console.WriteLine(returnData);
            Console.WriteLine(receiveBytes.Length.ToString());

            /* Vypis a vypocet delky trvani prenosu */
            Console.WriteLine(start.ToString());
            Console.WriteLine(end.ToString());
            TimeSpan x = end - start;
            Console.WriteLine(x.ToString());
            Console.ReadLine();
          }
          catch (Exception e)
          {
            Console.WriteLine(e.ToString());
          }
        }
    }
```

Pro toto pozorování by se dala využít i aplikace Packet Sender. Důvodem, proč jsem tuto aplikaci nepoužil, je to, že při výpisu paketů vždy na nějaký krátký okamžik aplikace zamrzne a časové známky, které aplikace ukáže, neodpovídají skutečnému času přijetí paketů. Tím vznikne výsledek, že například pro protokol IEEE 802.11b je naměřena propustnost 101Kbit/s, což jistě není správně ani při katastrofickém scénáři. Na závěr této sekce ještě přikládám tabulku 5.8 pro porovnání reálných propustností jednotlivých protokolů pro

Typ modulu	802.11b	802.11g	802.11n
ESP8266	5,52Mbit/s	11,80Mbit/s	12,42Mbit/s
ESP32	4,42Mbit/s	10,14Mbit/s	15,30Mbit/s

Tabulka 5.8: Porovnání reálných propustností jednotlivých protokolů WiFi pro moduly ESP8266 a ESP32

oba moduly.

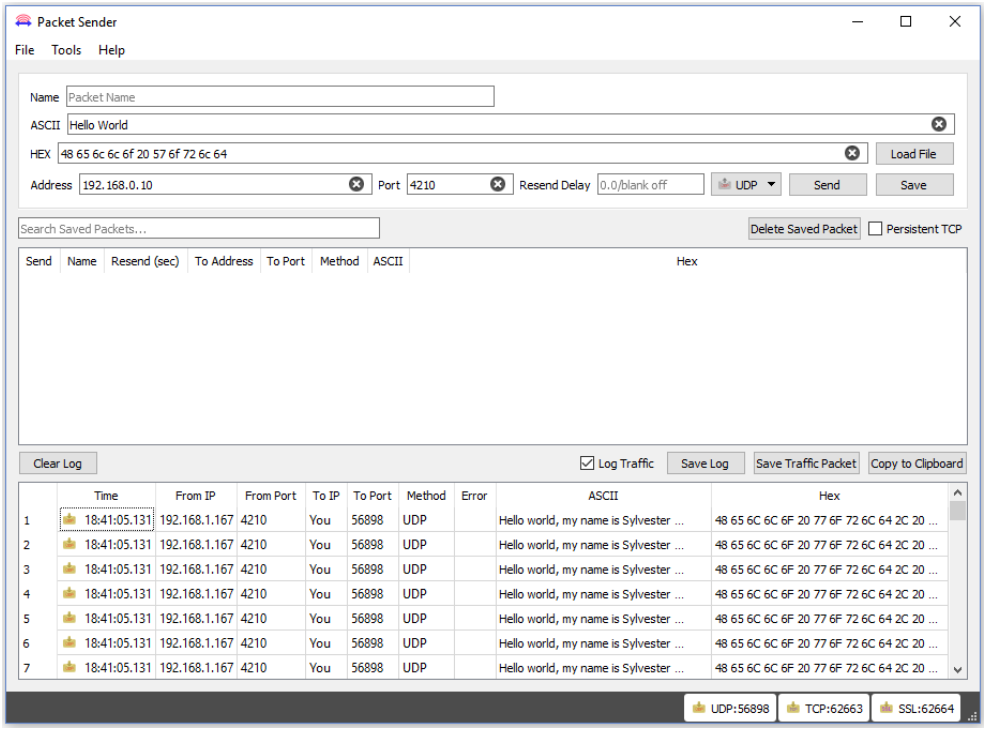
Z tohoto porovnání vidíme, že ESP8266 zhruba splňuje teoretické parametry protokolů až na protokol IEEE 802.11n, kde hardware zjevně nedokáže naplnit úplný potenciál tohoto protokolu. U ESP32 je patrné, že pro tento protokol je situace o poznání lepší, nicméně hardwareové řešení starších protokolů se zdá být trochu ošizené a vykaze horší výsledky než u modulu ESP8266.

5.5 Testování tunelu UART-WiFi

Jak už jsem se zmínil v kapitole Implementace, program tunelu UART-WiFi je posledním a největším programem. V rámci jeho testování jsem se zaměřil zejména na určení maximální propustnosti programu jako celku, při níž data na protistranu přichází úplná a neporušená. Jelikož tato práce poslouží jako základ pro projekt, jehož cílem bude posílat audio data přes tento tunel, je přípustná určitá tolerance chybovosti při přenosu dat. Z toho důvodu jsem také udělal několik pozorování, kde jsem konstantně zvyšoval propustnost a sledoval, jak velký dopad to má na přenos dat. Pozorování jsem prováděl jak na modulu ESP8266, tak na modulu ESP32 a výkonnosti obou modulů jsem následně porovnal.

Samotný test probíhá tak, že se z počítače na UART modulu ESP vyšlou data pomocí aplikace SerialSend při dané propustnosti. Modul tato data zpracuje, vytvoří z nich UDP paket a pomocí WiFi tento paket odešle zpět na počítač. Vznikne tím jednoduchá testovací smyčka. Celkově jsem vyslal data o velikosti 1MB s tím, že UDP pakety měly velikost 1KB. Přijaté pakety jsem odchytával pomocí aplikace Packet Sender. Na obrázku 5.9 je možné vidět, jak průběh odchytávání vypadá.

Po každém přenosu dat jsem přijaté pakety uložil do logovacího souboru, kde jsem data mohl snadno pozorovat. Zaměřil jsem se zejména na chybovost na celý 1MB dat a také chybovost v rámci jednoho paketu. Jako testovací data jsem zvolil prostý text v anglickém jazyce, abych mohl snadno identifikovat typy chyb, které v datech nastávají. Z pozorování dat jsem zjistil, že ani u jednoho z modulů nedochází v případě tunelu UART-WiFi k přehazování bajtů. V případě ESP8266 dochází pouze k občasným ztrátám malých bloků vstupních dat při vysokých rychlostech přenosu dat (ne všechna data se stíhají zpracovávat). V případě ESP32 tyto chyby nastávají také, avšak u některých vyšších rychlostech přenosu dat se občas objevují v datech znaky, které do

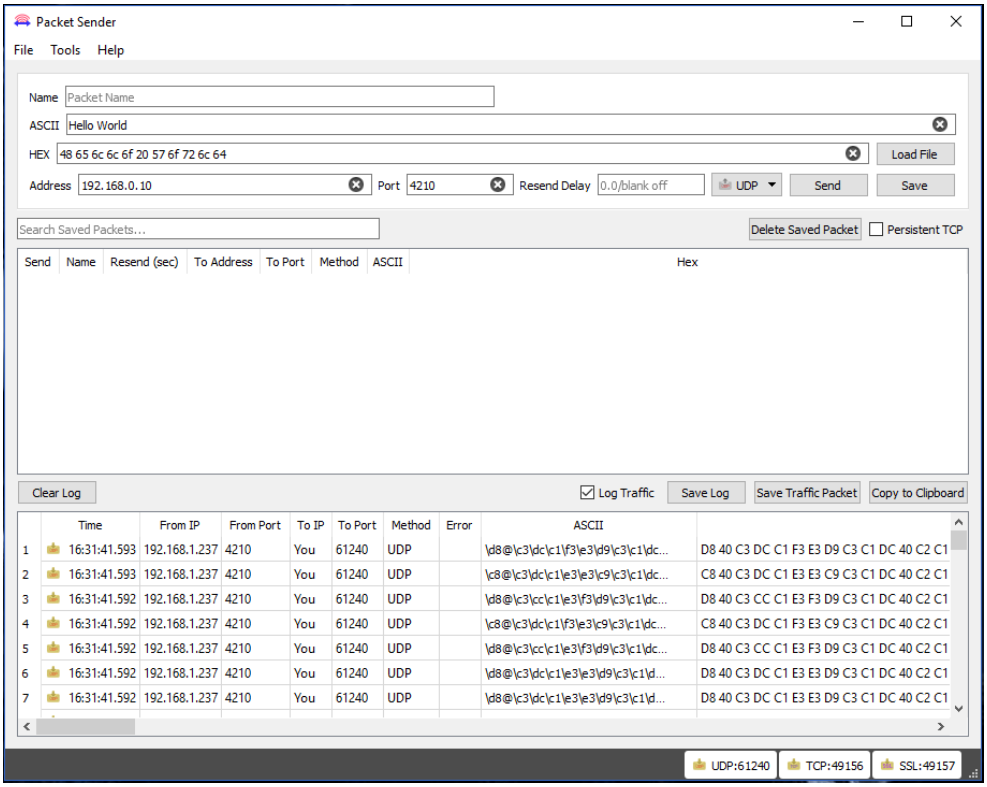


Obrázek 5.9: Ukázka funkce aplikace Packet Sender

správného kontextu nepatří. Z pozorování se zdá, že tato chyba vyplývá ze špatného zpracování dat při vytváření UDP paketu v metodě write() třídy UDP. Usoudil jsem tak, protože tato chyba při zpracování dat ze sériové linky nenastává.

Při pozorování jsem si nejprve stanovil výchozí hodnotu propustnosti 1,5Mbit/s. Vycházel jsem při tom z pozorování rozhraní UART, kde pro příjem dat na modulu vyhovovala oběma modulům propustnost 2Mbit/s pro bezchybný přenos. Pro jistotu jsem tuto hodnotu propustnosti ještě o 0,5Mbit/s snížil. V průběhu testování jsem propustnost nejprve zvyšoval o 0,1Mbit/s až do 2Mbit/s a následně jsem zvolil krok 0,5Mbit/s. Protože jsem při pozorování u modulu ESP32 zjistil, že chybovost pro propustnost 3Mbit/s je již neúnosná (více než 75% chyb v datech), dále jsem její hodnotu nezvyšoval. Poslední propustnost, která vykazovala rozumnou chybovost byla 2,5Mbit/s, a proto jsem se snažil odhalit bod zlomu, kde přijatelná chybovost přechází v neúnosnou. Zjistil jsem, že tento bod zlomu se nachází již u propustnosti 2,6Mbit/s.

Zajímavé je, že u modulu ESP8266 jsem takovéto problémy nezaznamenal. Pouze při vysokých rychlostech se náhodně vyskytovala poměrně malá chybovost ve smyslu ztráty malých bloků vstupních dat. Chybovost se příliš neměnila až do propustnosti 6,1Mbit/s, která se ukázala jako poslední, kterou



Obrázek 5.10: Ukázka přijatých paketů v aplikaci Packet Sender při propustnosti 6,2Mbit/s na modulu ESP8266

modul ESP8266 dokáže zvládnout. Propustnost 6,2Mbit/s vykazovala znaky toho, že hardware modulu ESP8266 již tuto rychlost nedokáže zvládnout, což jsem předpokládal již při testování samotného rozhraní UART. Data v bufferu totiž nebyla čitelná a evidentně se tedy na tuto rychlost modul již nepodařilo nakonfigurovat. Tuto situaci znázorňuje obrázek 5.10.

ESP32 má celkem 3 rozhraní UART, a proto bylo na místě vyzkoušet, zda UART1 nebo UART2 nevykazuje lepší výsledky, než UART0, na kterém jsem přenos dat původně testoval. Bohužel obě rozhraní poskytla stejné výsledky jako UART0. Chybovost se nesnížila u žádné hodnoty propustnosti ani když jsem měnil nastavení WiFi protokolů. Dá se tedy předpokládat, že hardware sériové linky u ESP32 není tak dobře navržen jako u ESP8266 nebo mohou být chybně implementované knihovny pro sériovou linku pro modul ESP32.

Celkově jsem pozoroval na modulu ESP8266 15 různých hodnot propustností a pro každou propustnost jsem provedl 10 nezávislých pozorování. Pro modul ESP32 jsem pozoroval 10 různých hodnot propustností a pro každou propustnost jsem taktéž provedl 10 nezávislých pozorování. Poslední hodnotou propustnosti, která u modulu ESP32 zajišťovala bezchybný přenos dat byla 1,5Mbit/s. Co se týká modulu ESP8266, tak například při rychlosti 5Mbi-

propustnost v Mbit/s	chybovost v rámci paketu	celková chybovost na 1MB	největší rozptyl v B
4,5	1,6%	0,02%	23
4,6	0,6%	0,015%	18
5,0	1,4%	0,017%	23

Tabulka 5.9: Pozorování chybovosti při přenosu dat na modulu ESP8266

propustnost v Mbit/s	chybovost v rámci paketu	celková chybovost na 1MB	největší rozptyl v B
1,6	>90%	>90%	128
1,8	50%	48%	128
2,0	3%	6%	73
2,5	5%	0,1%	115
2,7	>90%	>90%	3
3,0	>90%	>90%	2

Tabulka 5.10: Pozorování chybovosti při přenosu dat na modulu ESP32

t/s se objevilo několik náhodných chyb v 10 pozorováních, nicméně při testech propustnosti 6,1Mbit/s se žádná chyba v datech neprojevila. Dá se tedy usuzovat, že tunel na tomto modulu je i při této propustnosti na modulu ESP8266 zcela použitelný. Pro porovnání výsledků uvádím tabulku 5.9, která znázorňuje chybovost pro různé propustnosti pro modul ESP8266, u kterých nějaká chybovost nastala a největší rozptyl v bajtech mezi chybami. Dále uvádím tabulku 5.10 vybraných propustností, která taktéž znázorňuje chybovost a rozptyl mezi chybami, pro modul ESP32. V některých případech chybovost přesáhla 90%, a proto již dále neuvádím přesné číslo. Co se týká pozorování na ESP8266, bylo možné nastavit propustnost sériové linky až do výše 6,1Mbit/s a s touto rychlostí program tunelu neměl žádný problém (odeslal všechna data bez chyby) i při nastavení WiFi protokolu IEEE 802.11b, který reálně dosahuje na tomto modulu 5,52Mbit/s. Z toho vyplývá, že zpracování dat ze sériové linky a následné vytvoření UDP paketu vytvoří dostatečnou prodlevu mezi odesíláním jednotlivých paketů tak, aby bylo možné použít i nejslabší protokol WiFi, tedy již zmíněný IEEE 802.11b. Zdá se, že na návrh sériové linky (či implementace jejích knihoven) u modulu ESP32 nebyl kladen takový důraz jako například na návrh procesoru, moderních komunikačních rozhraní a spolehlivosti modulu.

Závěr

Náplní této diplomové práce byla analýza možností připojení a zpracování dat ze senzorů pomocí čipu ESP32 a zejména zaměření se na možnosti přenosu těchto dat pomocí vestavěné WiFi a sériové linky. Součástí této práce bylo také porovnání těchto možností se starším čipem ESP8266 a také ověření možností přenosu dat v praxi na obou čipech. Teoreticky byla také rozebrána možnost předzpracování dat v rámci samotného čipu.

V praktické části práce jsem navrhl a implementoval aplikace v jazyce C pro moduly ESP32 a ESP8266, které mi pomohly analyzovat praktické možnosti přenosu dat u obou modulů pomocí vestavěné WiFi a sériové linky. V návrhové části jsem také vyřešil zapojení pro oba moduly, které je klíčové pro správný provoz obou modulů. Součástí programového vybavení je také aplikace tunelu UART-WiFi, která přijímá data ze sériové linky, vytváří z těchto dat UDP pakety o velikosti 1KB a tyto pakety následně odesílá pomocí WiFi na zařízení v síti.

Nejdůležitější součástí této práce byla praktická analýza možností přenosu dat obou modulů společně s analýzou použitelnosti tunelu UART-WiFi. Už jen teoretická analýza specifikací modulů napovídá, že ESP32 je poměrně výkonnější a použitelnější, než starší čip ESP8266. ESP32 má mnohem více pinů, které uživatel může využít pro připojení různých senzorů. Naopak ESP8266 má menší počet pinů a díky tomu k němu lze připojit mnohem méně senzorů současně. Dále ESP32 disponuje dvoujádrovým procesorem o frekvenci 240MHz, kdežto ESP8266 má pouze jednojádrový procesor o frekvenci 80MHz. U ESP32 lze také využít celkem tří rozhraní UART, kdežto ESP8266 má pouze jedno. Výkon rozhraní UART u ESP8266 je kupodivu o dost lepší než u ESP32. Co se týká vestavěné WiFi, jsou možnosti obou modulů stejné.

Praktická analýza ukázala, že z hlediska starších komunikačních rozhraní je na tom ESP8266 lépe, než novější typ ESP32. Zejména u tunelu UART-WiFi se ukázalo, že ESP8266 zvládá bezchybný přenos i při propustnosti sériové linky 6,1Mbit/s, zatímco ESP32 v tomto případě dosahuje pouze 1,5Mbit/s. U ESP8266 již ale dále nelze tuto propustnost zvyšovat, jelikož nám brání hard-

warový limit pro sériovou linku. ESP32 na druhou stranu při vyšší propustnosti než 1,5Mbit/s vykazuje velmi vysokou chybovost v datech a při 3Mbit/s je tato chybovost už neúnosná (data obsahují více než 75% chyb). Analýza samotné WiFi odhalila, že pro protokoly IEEE 802.11b a IEEE 802.11g vykazuje ESP8266 o něco vyšší praktickou propustnost než ESP32, nicméně propustnost u protokolu IEEE 802.11n je o hodně vyšší u ESP32 než je tomu u ESP8266. Z tohoto pohledu se zdá, že vývoj ESP32 se zaměřil spíše na modernější protokoly, větší nabídku různých komunikačních rozhraní, velikost paměti, výkon procesoru a celkovou spolehlivost modulu ovšem na úkor například zmíněné sériové linky. Slabým bodem tohoto tunelu je u obou modulů sériová linka, jelikož WiFi při nastavení na protokol IEEE 802.11n dosahovala propustnosti 12,42Mbit/s pro ESP8266 a pro ESP32 dokonce 15,3Mbit/s.

Co se týká předzpracování dat z mikrofonu, není vhodný ani jeden z modulů. ESP32 sice disponuje 12 bitovým A/D převodníkem, zatímco ESP8266 pouze 10 bitovým, nicméně pro dobrou kvalitu zvuku, která je dnes v mnoha systémech žádána, tyto převodníky nestačí. Z tohoto důvodu by bylo dobré nechat předzpracování těchto dat na nějaký jiný mikrokontrolér s alespoň 16 bitovým A/D převodníkem, který dokáže vzorkovat data s frekvencí 44,1kHz.

Z hlediska celkového porovnání je ESP32 v mnoha ohledech lepší. Při návrhu různých systémů s využitím ESP32 je důležité rozmyslet si, která rozhraní a funkce z ESP32 využít, a která raději delegovat na jiné mikrokontroléry, které jsou pro dané funkce vhodnější.

Tato práce poskytuje potřebné informace pro práci s moduly ESP8266 a ESP32, zejména pak pro využívání vestavěné WiFi a sériové linky včetně zpracování dat v rámci modulu a mohla by být přínosem pro mnoho projektů zabývajících se touto problematikou. V budoucnu by se tato práce mohla využít v prostředí, kde se zpracovávají a posílají reálná audio data jako například v systému, kde bude možné vzdáleně pomocí WiFi ovládat robota prostřednictvím hlasových příkazů. Rozšířením této práce by mohlo být nasazení tunelu UART-WiFi do projektu. Bylo by ovšem potřeba vyřešit zejména vzdálenou konfiguraci modulu, například pomocí konfiguračního souboru a skriptu a prostřednictvím sériové linky by se do modulu poslaly parametry jako například rychlost sériové linky, SSID a heslo příslušné WiFi, číslo UDP portu. Součástí tohoto rozšíření by bylo zahájení a ukončení přenosu dat pomocí modulů ESP.

Literatura

[1] AI-Thinker Team: *ESP07 Datasheet Version 1.0 [online]*. 2015, [cit. 2018-11-30], Anglická verze. Dostupné z: https://www.mikrocontroller.net/attachment/338570/Ai-thinker_ESP-07_WIFI_Module-EN.pd

[2] Espressif Inc.: *ESP-WROOM-32 Datasheet Version 2.7 [online]*. 2018, [cit. 2018-11-30], Anglická verze. Dostupné z: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf

[3] SKRBEK, M.: Sériová rozhraní, sérioví komunikace, sérioví sběrnice. LS 2014/2015, Fakulta Informačních Technologií ČVUT v Praze. Dostupné z: https://edux.fit.cvut.cz/courses/BI-VES/_media/lectures/bives_pred_05.pdf

[4] Philips Semiconductors: *I2S bus specification [online]*. June 1996, [cit. 2018-11-30]. Dostupné z: https://www.sparkfun.com/datasheets/BreakoutBoards/I2SBUS.pdf

[5] SKRBEK, M.: A/D převodníky. LS 2014/2015, Fakulta Informačních Technologií ČVUT v Praze. Dostupné z: https://edux.fit.cvut.cz/courses/BI-VES/_media/lectures/bives_pred_06.pdf

[6] FreeBSD: *Serial and UART tutorial, revision 44692 [online]*. April 2014, [cit. 2018-11-30], Anglická verze. Dostupné z: https://www.freebsd.org/doc/en_US.ISO8859-1/articles/serial-uart/

[7] Pužmanová, R.: *Moderní komunikační sítě od A do Z*. Brno: Computer Press, 1996, ISBN 80-7226-049-098-7.

[8] BENTON, K.: *The Evolution of 802.11 Wireless Security [online]*. April 2010, [cit. 2018-11-30]. Dostupné z: https://benton.pub/research/benton_wireless.pdf

[9] SMOTLACHA, V.: Linková vrstva, metody přístupu. LS 2013/2014, Fakulta Informačních Technologií ČVUT v Praze. Dostupné z: https://edux.fit.cvut.cz/courses/BI-PSI/_media/lectures/p2-link.pdf

[10] The Institute of Electrical and Electronics Engineers Inc.: *IEEE 802.11b [online]*. January 2000, [cit. 2018-11-30], Anglická verze. Dostupné z: http://ant.comm.ccu.edu.tw/course/92_WLAN/1_Papers/IEEE%20Std%20802.11b-1999.pdf

[11] The Institute of Electrical and Electronics Engineers Inc.: *IEEE 802.11g [online]*. June 2003, [cit. 2018-11-30], Anglická verze. Dostupné z: http://www.diegm.uniud.it/tonello/MATERIAL/STANDARDS/802.11/802.11g-2003.pdf

[12] Cisco Systems Inc.: *802.11n Wireless Technology Overview [online]*. 2007, [cit. 2018-11-30], Anglická verze. Dostupné z: http://www.ventevinfra.com/pdf/802.11n%20Wireless%20Technology%20Overview.pdf

[13] Silicon Labs: *Single-chip USB-to-UART Bridge CP2104, Revision 1.2 [online]*. 2017, [cit. 2018-11-30], Anglická verze. Dostupné z: https://www.silabs.com/documents/public/data-sheets/cp2104.pdf

[14] BARRY, R.: *Mastering the FreeRTOS Real Time Kernel [online]*. 2016, [cit. 2018-11-30]. Dostupné z: https://www.freertos.org/Documentation/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf

[15] Amazon Inc: *The FreeRTOS Reference Manual [online]*. 2017, [cit. 2018-11-30]. Dostupné z: https://www.freertos.org/Documentation/FreeRTOS_Reference_Manual_V10.0.0.pdf

[16] Espressif Inc.: *ESP-IDF Programming Guide [online]*. 2016, [cit. 2018-11-30], Anglická verze. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/latest/

[17] NagleCode LLC.: *Packet Sender Documentation [online]*. 2018, [cit. 2018-11-30], Anglická verze. Dostupné z: https://packetsender.com/documentation

[18] Ted Burke: *SerialSend [online]*. 2018, [cit. 2018-11-30], Anglická verze. Dostupné z: https://batchloaf.wordpress.com/serialsend/

[19] Espressif Inc.: *Wi-Fi Driver [online]*. 2016, [cit. 2018-11-30], Anglická verze. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/latest/api-guides/wifi.html?highlight=802.11bgn%20mode

[20] Espressif Inc.: *ESP32 Datasheet [online]*. 2018, [cit. 2018-11-30], Anglická verze. Dostupné z: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[21] Cisco Systems Inc.: *WLAN Radio Frequency Design Considerations [online]*. [cit. 2018-11-30]. Dostupné z: https://www.cisco.com/en/US/docs/solutions/Enterprise/Mobility/emob30dg/RFDesign.html

[22] Espressif Inc.: *ESP8266 Wi-Fi Channel Selection Guidelines [online]*. June 2016, [cit. 2018-11-30], Anglická verze. Dostupné z: https://www.espressif.com/sites/default/files/esp8266_wifi_channel_selection_guidelines.pdf

[23] Espressif Inc.: *ESP8266 SDK API Guide [online]*. June 2015, [cit. 2018-11-30], Anglická verze. Dostupné z: http://nonnoise.github.io/ESP-WROOM-02/_downloads/2C-ESP8266__SDK__Programming%20Guide__EN_v1.1.1.pdf

Příloha A

Seznam použitých zkratek

SoC System on Chip

WiFi Wireless Fidelity

UART Universal Asynchronous Receiver/Transmitter

SPI Serial Peripheral Interface

IIC Inter Integrated Circuit

USB Universal Serial Bus

MISO Master In Slave Out

MOSI Master Out Slave In

SCLK Serial clock

SS Slave select

SCL Serial clock line

SDA Serial data line

WS Word select

GPIO General Purpose Input/Output

UDP User Datagram Protocol

CSMA/CA Carrier Sense Multiple Access with Collision Avoidance

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF