
# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

## FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

## ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY
INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS


# NÁVRH A REALIZACE MONITORU SÉRIOVÉ KOMUNIKACE
DESIGN AND REALIZATION OF SERIAL COMMUNICATION MONITOR

**DIPLOMOVÁ PRÁCE**
MASTER'S THESIS


**AUTOR PRÁCE** — **Bc. Jan Šnajder**
AUTHOR

**VEDOUCÍ PRÁCE** — **doc. Ing. Jiří Krejsa, Ph.D.**
SUPERVISOR



**BRNO 2020**

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

# Zadání diplomové práce

| | |
|---|---|
| Ústav: | Ústav mechaniky těles, mechatroniky a biomechaniky |
| Student: | **Bc. Jan Šnajder** |
| Studijní program: | Aplikované vědy v inženýrství |
| Studijní obor: | Mechatronika |
| Vedoucí práce: | **doc. Ing. Jiří Krejsa, Ph.D.** |
| Akademický rok: | 2019/20 |

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

## Návrh a realizace monitoru sériové komunikace

**Stručná charakteristika problematiky úkolu:**

Sériové komunikační rozhraní UART (Universal Asynchronous Receiver–Transmitter) je jeden z nejčastějších způsobů komunikace elektronických systémů. Analýza komunikace je nezbytná při odlaďování chyb během vývoje. Jednou z možností je použití logického monitoru, jehož návrh a realizace je cílem práce. Součástí práce je propojení s počítačem a prezentace dat ve vhodném softwaru.

**Cíle diplomové práce:**

1. Prostudujte problematiku analýzy UART komunikace.
2. Vyberte vhodný mikrokontrolér, navrhněte a realizujte hardware zařízení.
3. Vytvořte algoritmus pro sběr dat a realizujte připojení monitoru k počítači pomocí USB protokolu.
4. Navrhněte a implementujte propojení se softwarem Wireshark.

**Seznam doporučené literatury:**

PINKER, J. Mikroprocesory a mikropočítače. První vydání. Praha, BEN: 2004. ISBN 80-7300-110-1

COMBS, G. Wireshark. Elektronické dokumenty dostupné na http://www.wireshark.org/

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2019/20

V Brně, dne

L. S.

| | |
|---|---|
| prof. Ing. Jindřich Petruška, CSc.<br>ředitel ústavu | doc. Ing. Jaroslav Katolický, Ph.D.<br>děkan fakulty |

**Abstrakt**

Obsahem práce je návrh a realizace sériového monitoru pro UART rozhraní. Práce popisuje vytvoření hardwarového zařízení starajícího se o sběr dat z analyzovaných komunikací a jejich distribuci prostřednictvím USB sběrnice. Dále se text věnuje propojení sériového monitoru se softwarem *Wireshark* pomocí rozhraní *extcap*. Pro zobrazení dat v čitelné podobě je vytvořen dekódovací skript, který je následně testován na vybraném protokolu. Poslední část práce se věnuje vyhodnocení propustnosti hardwarového zařízení v závislosti na zaplněnosti datové linky.

**Summary**

Aim of this thesis is design and realization of serial communication monitor. The thesis describes creation of hardware device responsible for collecting data from analyzed communications and for distribution of these data by USB bus. Thesis also explains connection between serial monitor and software *Wireshark* by *extcap* interface. Visualization of data in readable format is done by decoding script, which is tested by selected protocol. The last part of the study is focused on evaluation of data permeability of hardware device according to data line fullness.

**Klíčová slova**

sériový monitor, sériová komunikace, UART, kompozitní zařízení, USB, Wireshark, extcap, disektor, Lua

**Keywords**

serial monitor, serial communication, UART, composite device, USB, Wireshark, extcap, dissector, Lua

**Bibliografická citace**

ŠNAJDER, Jan. *Návrh a realizace monitoru sériové komunikace.* Brno, 2020. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/124657. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky. Vedoucí práce Jiří Krejsa.

**Čestné prohlášení**

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím uvedené literatury.

Jan Šnajder, Brno, 2020

**Poděkování**

Tímto bych chtěl poděkovat všem, kteří mi při vytváření této práce pomáhali. Především děkuji vedoucímu diplomové práce doc. Ing. Jiřímu Krejsovi Ph.D. Dále bych chtěl poděkovat kolegům z Bender Robotics s.r.o. za jejich čas a cenné rady, které mi věnovali.

# Obsah

# 1 Úvod

Za dobu existence elektronických systémů udělalo lidstvo v této vědní disciplíně obrovský pokrok. Od zařízení řešících pouze jeden jednoduchý úkol jsme se dostali až k systémům, jež jsou součástí komplexních soustav a jsou schopny vykonávat nejrůznější úlohy bez jakéhokoliv zásahu člověka. Aby mohly tyto systémy spolupracovat a přenášet mezi sebou nejrůznější data, bylo potřeba zavést pravidla a způsoby komunikace. V elektronice existují dva základní typy přenosů: sériový a paralelní. Sériová komunikace je proces, kdy se jednotlivé signálové prvky (např. bity) téhož datového proudu předávají za sebou. Opakem je přenos paralelní, který přenáší určitý počet prvků současně.

Právě sériový přenos je nejčastějším způsobem komunikace elektronických systémů. Oproti paralelnímu přenosu přináší úsporu v množství použitých vodičů a díky tomu i úsporu finanční. Dalšími výhodami vyplývajícími z nenáročnosti na počet vodičů je možnost jejich použití na delší vzdálenosti, anebo v situacích, kdy je omezen prostor, například na deskách plošných spojů. Všechny tyto vlastnosti spolu se zmenšováním veškeré elektroniky činí ze sériového přenosu lukrativní způsob komunikace.

Díky množství inženýrských aplikací, které využívají sériový přenos, je tento druh komunikace častým objektem nejrůznějších analýz. Tyto analýzy jsou prováděny především při vývoji, kdy je potřeba odladit chyby a zkontrolovat správnou funkcionalitu elektronických zařízení. Jednou z možností, jak analyzovat sériovou komunikaci, je využití sériového monitoru či logického analyzátoru. Zařízení provádějící takovou analýzu bývá v anglické literatuře často označováno jako *sniffer* (z anglického *sniff* – čenichat). Za svůj název vděčí faktu, že tyto nástroje bývaly často využívány k odcizení citlivých informací. Jedná se o zařízení, které je připojeno paralelně k datovým linkám a pouze odposlouchává komunikaci. Vzhledem k tomu, že nezasahují do přenosu dat, je velice těžké je odhalit. Jejich prvotní využití bylo především v síťových komunikacích, nicméně jsou vhodné i pro sériovou komunikaci.

Tato práce se zabývá návrhem sériového monitoru pro rozhraní UART (*Universal asynchronous receiver/transmitter)*. Cílem práce je realizace tohoto zařízení, jeho propojení s počítačem a následná prezentace analyzované komunikace ve vhodném softwaru. Využití by sériový monitor mohl nalézt především při vývoji vestavěných systémů. S jeho pomocí půjde analyzovat jednotlivé prvky komunikace, což usnadní odhalení chyb a jejich odladění. Součástí práce je i testování v praxi využívaným protokolem složeným z HDLC (*High-level Data Link Control*) a vyšší informační vrstvy CBOR (*Concise Binary Object Representation*).

# 2 Rozbor zadání

Hlavním cílem této práce je realizace sériového monitoru. To znamená, že v závěru bude vytvořeno zařízení, ke kterému bude možné připojit dva analyzované systémy komunikující pomocí rozhraní UART. Zařízení musí být schopné tuto komunikaci zachytit a distribuovat ji pomoci USB sběrnice, díky čemuž půjdou analyzovaná data sledovat na počítači. Data je nutné prezentovat v čitelné podobě a ve vhodném softwaru.

Pro úspěšný návrh je vhodné znát princip a vlastnosti sériových komunikací, především pak UART rozhraní. Dalším logickým krokem je rešerše možností, pomocí kterých je možné analyzovat sériovou komunikaci. Vzhledem k tomu, že se práce zabývá propojením hardwarového zařízení s počítačem, je nutné mít znalosti i o způsobu komunikace těchto dvou částí – USB sběrnici.

Po teoretické části následuje zkonkretizování požadavků sériového monitoru. Na jejich základě je vybrán vhodný hardware a provedena analýza možných řešení. Analýza shrnuje veškeré dostupné informace a vytváří jasnou představu o nejvhodnějším způsobu realizace.

Samotná realizace je rozdělena na dvě části. Její první část zahrnuje vytvoření firmwaru starajícího se o správný chod hardwarového zařízení. Následně je realizováno softwarové rozhraní pro analýzu dat v počítači. Pro testování je vybrán vhodný protokol, který ověřuje jak funkčnost hardwarového zařízení, tak správnost dekódovacích skriptů.

Po zhotovení sériového monitoru je přikročeno k vyhodnocení jeho funkčnosti a spolehlivosti v závislosti na nejrůznějších parametrech analyzované komunikace.

# 3 Rešeršní část

Tato kapitola je rešerší zabývající se sériovou komunikací. V první části je popsáno rozhraní UART (*Universal Asynchronous Receiver/Transmitter*) a možnosti jeho analýzy. Druhá část se věnuje popisu sběrnice USB (*Universal Serial Bus*) a všech jejích součástí.

## 3.1 Sériová komunikace

Jak již bylo napsáno v úvodu, u sériového přenosu platí, že jednotlivé signálové prvky téhož datového proudu se předávají za sebou – sériově. Naproti tomu paralelní přenos je zpravidla takový přenos, kdy je určitý počet signálových prvků přenášen současně. Tento určitý počet (8, 16, 32 atp.) je označován jako binární slovo. Paralelní přenos se používá především na krátké vzdálenosti, a to hlavně z finančních důvodů, protože je potřeba mnohonásobně víc vodičů než u sériového přenosu. Vzdálenost paralelního přenosu je dále ovlivněna rušením, které vzniká mezi jednotlivými vodiči. Kvůli rušení je nutné snížit rychlost nebo vzdálenost přenosu. [1]

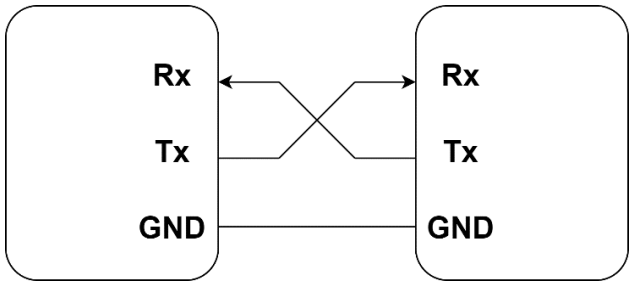
Sériový přenos dat se dá dále rozdělit podle použité synchronizace na synchronní, arytmický a asynchronní. Synchronní přenos využívá odděleného synchronizačního signálu, který určuje, kdy jsou data platná. U arytmického přenosu je synchronizace zajištěna pomoci předávání tzv. synchronizačních znaků nebo jejich posloupností, podle které se přijímač vždy s jejím přijetím synchronizuje. Výhoda arytmického přenosu je především v absenci dalšího synchronizačního vodiče. Nicméně na přijímací straně je nutný oscilátor a synchronizační posloupnost také snižuje přenosovou rychlost. [2][3]

Asynchronní přenos je koncepčně nejjednodušší. Odesílatel při něm explicitně sdělí příjemci, kdy začíná a kdy končí jednotlivé intervaly. Díky tomuto způsobu je možné, aby jednotlivé bitové intervaly trvaly různě dlouho, a tudíž přenosová rychlost nebyla konstantní. V praxi se takový způsob nepoužívá, protože pro jeho realizaci je nutná třístavová logika. Dva stavy vyjadřují binární hodnoty a třetí stav vymezuje začátek a konec bitového intervalu. Kvůli nízkému využití tohoto přenosu je arytmický často chybně označován jako asynchronní. [2][3]

Dalším kritériem pro rozdělení sériového přenosu je směr komunikace. V tomto ohledu se dají spojení dělit na jednosměrná (simplexní) a obousměrná (duplexní). Obousměrná komunikace se dále dělí na poloviční (*half-duplex*) a plnou (*full-duplex*). Poloviční obousměrný provoz přenáší data buďto jedním, anebo druhým směrem, avšak nikoli současně. Plný obousměrný přenos podporuje oba směry komunikace v jeden moment. [4]

## 3.2 Rozhraní UART

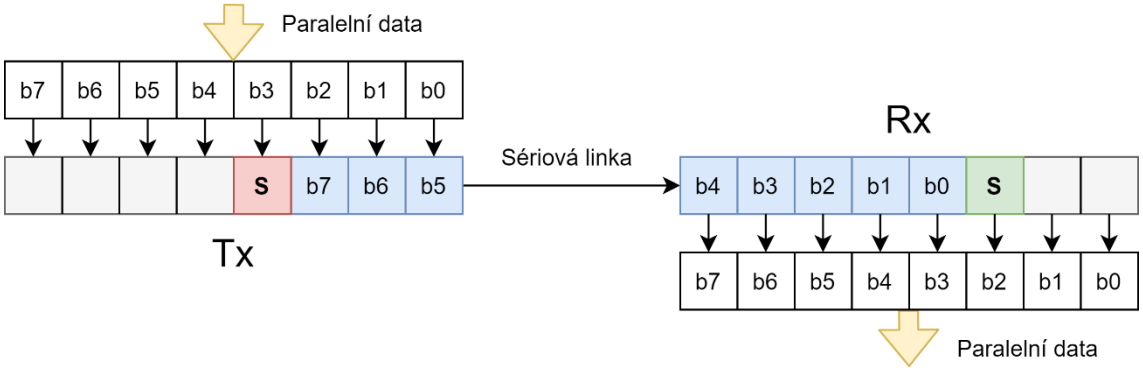
*Universal asynchronous receiver/transmitter* (dále jen UART) je typ obvodu realizující sériovou komunikaci. Bývá součástí řídících systémů (mikrokontrolerů). Představuje základ pro některé další standardy například RS-232, RS-485 atd. Pro komunikaci se využívají dvě jednosměrné datové linky – Rx, která je určena pro příjem dat a Tx, starající se o vysílání. Schéma zapojení lze vidět na obrázku č. 1. [5]




**Obrázek č. 1: Zapojení linek UART rozhraní**
**(**Zdroj: Vlastní zpracování dle: 5)

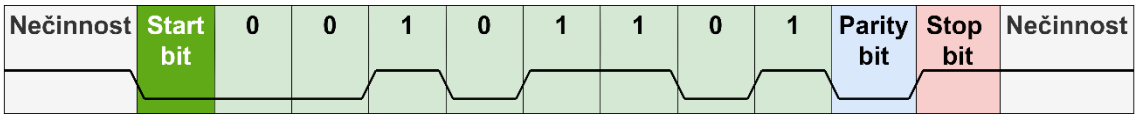
Jak lze vyčíst z názvu, UART je typ asynchronní (správně arytmické) komunikace. Není zde žádný synchronizační signál. Přijímač musí již před přijetím vědět, jakou rychlostí mu vysílač bude data posílat. Tato rychlost je označena jako baud *rate*, který je definován počtem změn stavu za jednu sekundu. Synchronní nástavbou rozhraní UART je USART. Ten používá kromě datových linek také hodinovou neboli synchronizační linku. V této konfiguraci jde použít pouze obousměrnou poloviční komunikaci.

Ve své podstatě UART představuje převodník paralelních dat na sériovou formu a naopak. Převod je realizovaný pomocí posuvného registru. Pro každý směr přenosu je potřebný jeden takový registr. Celý proces převodu lze vidět na obrázku č. 2.




**Obrázek č. 2: Převod paralelních dat na sériové**
**(**Zdroj: Vlastní zpracování dle: 6)

Na obrázku č. 3 je znázorněn přenos dat přes UART. Je možné ho rozdělit na několik částí. Přenos začíná start bitem, který je reprezentovaný změnou polarity ze stavu nečinnost. Start bit slouží přijímací straně ke synchronizaci. Po něm následuje samotný přenos dat, který může být dlouhý 5 až 9 bitů. Jako první se vždy posílá bit s nejnižší váhou (*LSB – Least Significant Bit*). Datové bity může následovat bit paritní. Paritní bit zajišťuje, že celkový počet jedničkových bitů v bitovém slově je sudý (sudá parita) nebo lichý (lichá parita) a slouží k detekci chyby. Jeho použití je volitelné. Celý přenos zakončuje stop bit. Ten může být jeden, jeden a půl nebo dva bity dlouhý. Stop bit má stejnou polaritu jako stav nečinnosti a poskytuje přijímací straně více času před přenosem dalšího bitového slova. [5]




**Obrázek č. 3: Časový přenos bytu 10110100 přes UART**
(Zdroj: Vlastní zpracování dle: 5)

UART poskytuje možnost vzájemného propojení koncových stanic formou *point-to-point* nebo *point-to-multipoint*. První zmíněný představuje vzájemnou komunikaci mezi dvěma koncovými zařízeními. Druhý případ je propojení, kde jedna koncová řídící stanice posílá data několika koncovým řízeným stanicím pomocí společné sběrnice. [5]

## 3.3 Analýza sériových komunikací

Nástroje pro analýzu sériových sběrnic mohou mít různý charakter. Často bývají označovány jako *sniffery*. Mohou mít podobu počítačového programu nebo fyzického zařízení, které je použito k zachycení a uložení komunikace a k její případné další analýze. Většinou se tyto entity vzájemně doplňují, kdy hardwarová část provádí sběr dat a softwarová část je zodpovědná za jejich analýzu.

Cílem analýzy je dekódovat komunikaci v daných protokolech a poskytnout uživateli náhled v čitelné podobě. Nástroje pro analýzu mohou představovat i jisté bezpečnostní riziko. Vzhledem k obtížnosti jejich detekce se dají poměrně jednoduše využít k odposlouchávání cizí komunikace.

### 3.3.1 Hardwarové analyzátory

Hardwarovým analyzátorem je většinou pasivní zařízení připojené paralelně ke komunikaci. Pro sériové komunikace se takovému zařízení říká logický analyzátor. Ten pouze přijímá data na daném médiu a do komunikace nijak nezasahuje. Přijatá data si může dále uložit do vnitřní paměti, nebo je přímo poskytovat k další analýze. Ve své podstatě je logický analyzátor osciloskopem pro digitální signály. Dále má oproti osciloskopům více paměti a více měřících signálů. Často se stává, že je logický analyzátor prodáván jako nástavba pro osciloskop, anebo v případě dražších osciloskopů je přímo

jeho součástí. Na trhu je možné najít mnoho variací logických analyzátorů. Cena těch nejjednodušších se pohybuje v řádech stovek korun českých, nicméně lze nalézt i analyzátory za několik desetitisíců.

Jako zástupce levnější cenové kategorie lze považovat logický analyzátor od firmy *Saleae*. [7] Jedná se o univerzální osmikanálový analyzátor s šířkou pásma až 24 MHz. Nicméně tato šířka pásma je zároveň i maximální vzorkovací frekvencí. Proto je vhodný pouze pro signály s frekvencí nižší. Analyzátor podporuje spoustu protokolů a k produktu je poskytován i software. Jedná se o cenově velmi lákavou variantu, avšak není vhodná pro pokročilejší aplikace.

Druhým příkladem, tentokrát z dražší cenové kategorie, je *Hantek MS052029* [8] s cenou lehce pod dvacet tisíc korun českých. Jedná se kombinované zařízení poskytující digitální osciloskop a 16 kanálový logický analyzátor. Šířka pásma je až 200 MHz při vzorkování 0,5 až 1 GS/s (*gigasamples per second*). K dispozici má řadu výpočetních funkcí a možnost připojení k počítači pomocí USB nebo připojení flash disků.

### 3.3.2 Softwarové analyzátory

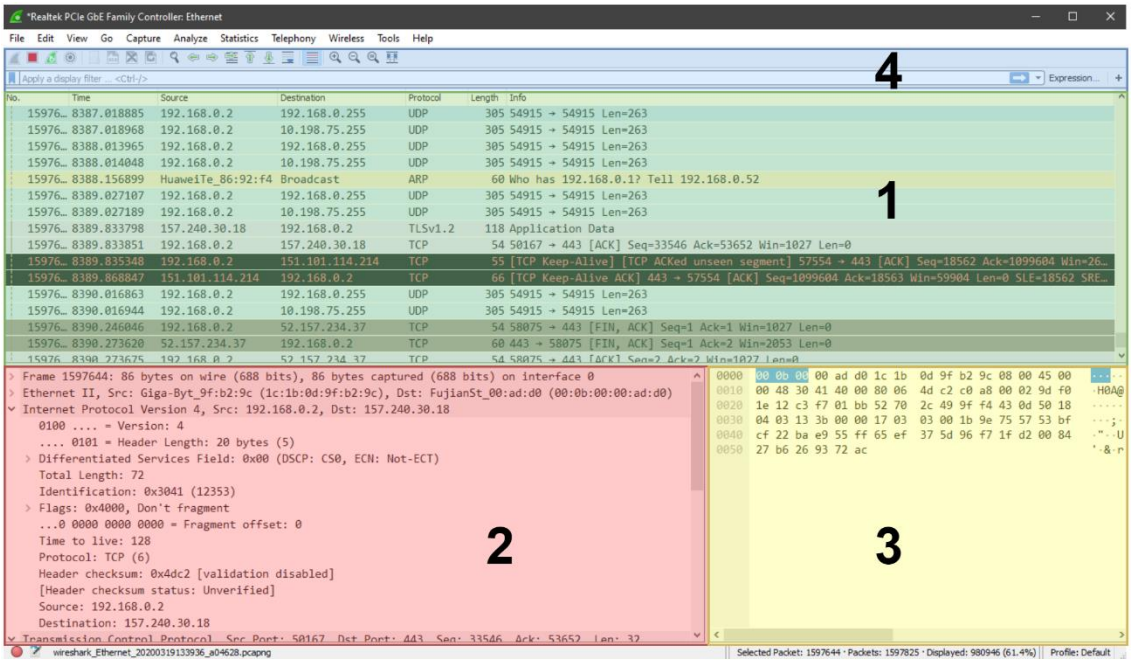
Softwarovým analyzátorem je myšlen počítačový program poskytující analýzu komunikačních protokolů. Funkčnost těchto softwarů je v dnešní době poměrně rozsáhlá a jednotlivé programy se dají klasifikovat podle následujících kritérii:

- počet podporovaných zařízení, ze kterých je možno přijímat data,
- počet podporovaných protokolů, které je možno dekódovat,
- detailnost informací o přenesených datech (čas, odesílatel, příjemce),
- možnost ukládat nebo načítat zachycenou komunikaci,
- možnost znovu sestavit rozdělené pakety,
- filtrování zachycených dat podle parametrů protokolu,
- identifikace chyb v komunikaci,
- možnost doplňovat dekódovací skripty pro zatím nepodporované protokoly.

Asi největší výhodou oproti hardwarovým zařízením je univerzálnost softwarových analyzátorů. Ta spočívá ve velkém množství protokolů, které lze dekódovat, a také v modularitě a rozšiřitelnosti o další protokoly. Tato modularita vyplývá z faktu, že většina komunikačních protokolů má vždy nějaké společné vlastnosti (čas příchodu paketu, délka paketu, data paketu). Proto stačí, aby rozhraní softwarového analyzátoru byla navržena jen pro tyto základní parametry a o zbytek se staraly jednotlivé dekodéry.

Většina existujících softwarových řešení má proto velmi podobné rozhraní. Náhled na něj je možné vidět na obrázek č. 4. Základem tohoto rozhraní je seznam zachycených paketů, aktualizujících se v reálnem čase, doplněný o fundamentální informace o daném paketu, viz oblast č. 1. Dále analyzátor umožňuje zobrazit každý paket detailněji a prohlížet jeho dekódované informace, viz oblast č. 2. Současně

s dekódovanými informacemi lze prohlížet i pakety v nezpracované podobě, viz oblast č. 3. Celou analýzu je možné ovládat a filtrovat pomocí ovládacích lišt, viz oblast č. 4.




**Obrázek č. 4: Náhled uživatelského rozhraní software *Wireshark*.**

Z praktického hlediska se dají softwarové analyzátory rozdělit do dvou skupin podle realizace uživatelského rozhraní. První skupinou jsou softwary využívající pouze příkazovou řádku, druhá skupina využívá plnohodnotné grafické uživatelské rozhraní (GUI). Příkazový řádek je vhodný pro automatizované zpracování zachycených protokolů, zatímco programy s grafickým rozhraním přinášejí především přehlednost a pohodlnost a jsou většinou nástavbou pro první skupinu.

V současné době existuje nespočet softwarových analyzátorů většina z nich je uzpůsobena především síťovým komunikacím. V posledních letech se však funkčnost těchto programů rozšířila i o sériovou komunikací. Za průkopníka v této oblasti se často označuje software *Wireshark*. Za nejpoužívanější nástroj využívající pouze příkazovou řádku se dá považovat software *WinDump*. Za zmínku stojí také donedávna fungující *Microsoft Message Analyzer*. Jednalo se o přímou konkurenci *Wireshark*, avšak v listopadu 2019 byl ukončen vývoj a odstraněny všechny oficiální cesty ke stažení. [9]

## Wireshark

Software *Wireshark* je především síťový analyzátor, který je zadarmo a jehož zdrojový kód je dostupný online (například z webhostingových stránek *GitHub*). [10][11] Zakladatelem je *Gerald Combs* a software je vyvíjen od roku 1998, původně pod jménem *Ethereal*. Je vyvíjen jako multiplatformová aplikace a funguje v operačních systémech *Windows, Linux, OS X, BSD* a *Solaris*. Na webu je dostupná také verze využívající pouze příkazovou řádku s názvem *T-Shark*.

V současné době *Wireshark* podporuje přes 1500 protokolů a toto číslo stále roste. Pro svou analýzu vyžaduje knihovnu *pcap*. Pro *Windows* je tato knihovna implementována jako *WinPcap,* pro *Linux* jako *libcpap*. Náhled grafického rozhraní lze vidět na obrázku č. 4.

*Wireshark* podporuje dva typy filtrování. Prvním je *Capture filter,* který filtruje pakety v okamžiku jejich přijetí a nevyhovující pakety jsou zcela zahozeny. Druhým typem je *Display filter*, který je aplikován až po přijetí. Z toho vyplývá že nevyhovující pakety pouze nejsou zobrazeny v hlavním výčtů paketů. *Capture filter* se zadává před začátkem analýzy a není možné jej později modifikovat. Používá se k redukci množství přijatých paketů. Příkladem může být filtrace komunikace pouze z a na určitou IP adresu. Pro filtraci komunikace obsahující pouze IP adresu 8.8.8.8 uživatel zadá příkaz `host 8.8.8.8`. Naproti tomu *Display filter* se zadává během analýzy a je možné jej průběžně měnit. Ta samá funkce, která byla předvedena u *Capture filter*, by v tomto případě vypadala jako `ip.addr == 8.8.8.8`. Dostupná informační pole jsou interaktivně našeptávána, což z něho činí velice intuitivní a pohodlný způsob filtrace.

*Wireshark* obsahuje mnoho pokročilých funkcí, mezi něž patří například:

- analýza VoIP telefonátů, včetně funkce pro přehrání obsahu,
- čtení a zápis do mnoha formátů definovaných jinými analyzátory,
- rekonstrukce jednotlivých paketů do celkové komunikace u vybraných protokolů,
- *real-time* analýza dat z Ethernet, IEEE 802.11, PPP/HDLC, ATM, Bluetooth, USB, Token Ring, Frame Relay, FDDI a další,
- automatická dekomprese dat v *gzip* formátu,
- možnost vytvoření vlastních skriptů pro dekódování neznámých protokolů,
- možnost vytvoření vlastního rozhraní pro příjem dat (*extcap*),
- rozsáhlá dokumentace ulehčující práci se softwarem. [10][17]

**WinDump**

Asi nejpoužívanějším softwarovým analyzátorem pouze s příkazovou řádkou je *tcpdump*. [12] Jedná se o síťový analyzátor pro operační systémy *Linux*. Jeho využívanost dala za vznik programu *WinDump*, jež je kompatibilní verzí *tcpdump* pro operační systém *Windows*. [13] *WinDump* má otevřený zdrojový kód s aktivní vývojářskou komunitou. Stejně jako *Wireshark* je funkčně závislý na knihovně *pcap*.

Funkcionalita softwaru je značně omezena, a to především kvůli chybějícímu grafickému rozhraní. Z tohoto důvodu je vhodný pouze pro základní analýzy. Velkou výhodou je použití vzdáleného přístupu v rámci SSH (*Secure Shell*). Z dalších funkcí za zmínku stojí především možnost použití *Capture filter*, volitelné zobrazení celého obsahu paketů nebo ukládání zachycené komunikace do .cap formátu. Bohužel podporuje jen základní protokoly Ethernet, FDDI, IP, IPv6, ARP, TCP a UDP. [13]

## 3.4 USB

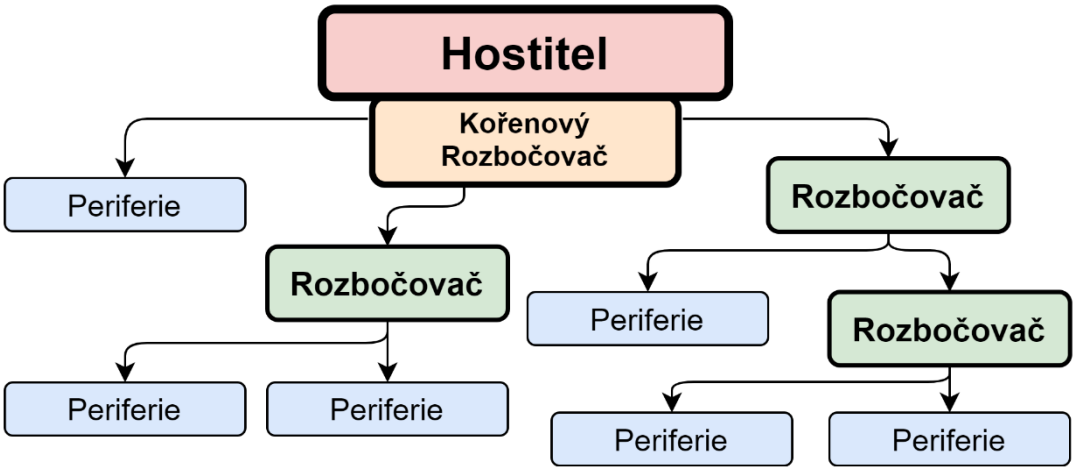
USB (*Universal Serial Bus*) je standardizovaná sériová sběrnice pro připojení periferií (*device*) k hostitelskému zařízení (*host*). Tento standard definuje architekturu sběrnice, její elektrické a mechanické vlastnosti, přenosový protokol a také zařízení pro větvení sběrnice – USB rozbočovače. USB umožňuje připojení zařízení pomocí jednotného standardizovaného rozhraní bez nutnosti restartovat počítač (*Plug-and-play)*.

Začátek vývoje této sběrnice byl v 90. letech a první oficiální verze vyšla v lednu roku 1996. Od té doby byly vyvinuty 4 verze. První USB 1.x podporovala dva základní typy rozhraní: zařízení s nízkou rychlostí (dále *low-speed*) – 1,5 Mbit/s a zařízení s plnou rychlostí (dále *full-speed*) – 12Mbit/s. Mezi *low-speed* zařízení se řadí především klávesnice, myši a nejrůznější herní ovladače. Pod *full-speed* spadají zařízení určená pro přenos dat jako je komunikace s tiskárnou, zvuk nebo video. Verze USB 2.0 rozšířila standart o další rozhraní – zařízení s vysokou rychlostí (dále *high speed*) – 480 Mbit/s. Verze USB 3.x zvýšila rychlost rozhraním zvaným *SuperSpeed* s rychlostí až 5 Gbit/s. V srpnu 2019 byly představeny specifikace USB 4.0, nicméně první zařízení vybavená tímto rozhraním by měla být dostupná až na konci roku 2020. [14][15]

V této práci bylo využito *full-speed* konfigurace, z toho důvodu se další podkapitoly věnují převážně této konfiguraci.

### 3.4.1 Topologie

USB využívá vrstvenou hvězdicovou topologii (*tiered-star topology)*, viz obrázek č. 5. Ve středu této topologie se nachází hostitel, který definuje USB. Každá USB sběrnice smí mít pouze jednoho hostitele. Ten je implementovaný jak hardwarově, firmwarově, tak softwarově. Hostitel obsahuje jeden nebo více kořenových rozbočovačů (*Root hub*), které poskytují přípojky pro další rozbočovače nebo koncové periferie. [14][16]



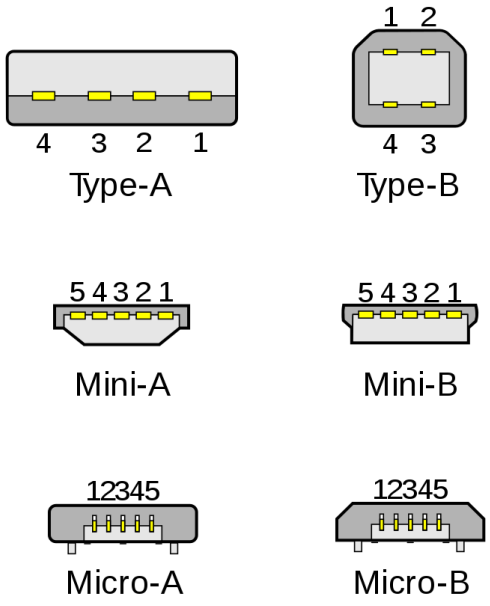

**Obrázek č. 5: Topologie USB sběrnice**
(Zdroj: Vlastní zpracování dle: 16, s. 68)

Přestože jsou USB periferie připojovány ve hvězdicové topologii, tak hostitel komunikuje s každým zařízením, jako by bylo přímo připojeno ke kořenovému rozbočovači. Každý rozbočovač může mít pouze jedno propojení s vyšší vrstvou (*upstream connection)* a až sedm propojení s vrstvou nižší (*downstream connection*). Pomocí řetězové architektury (*daisy – chain*) lze k jednomu hostiteli připojit až 127 zařízení v maximálně sedmi vrstvách. Tato limitace zajišťuje přijatelnou časovou odezvu od nejvzdálenějšího zařízení po hostitele podle USB specifikací. [14]

### 3.4.2 Fyzická vrstva

V dnešní době se pro připojení zařízení používá celá řada konektorů, viz obrázek č. 6. Ty se dají rozdělit na typ A a typ B. Typ A se používá u hostitele a rozbočovačů, například na základních deskách. Typ B můžeme nalézt na zařízeních, v dnešní době například na tiskárnách. Tímto způsobem se mělo původně zabránit vytvoření elektrické smyčky na rozbočovači. Nicméně většina výrobců nabízí kabely s konektory typu A na obou koncích. [14][15]

Druhou možností jak rozdělit konektory je podle velikosti na standardní, mini a mikro. Velikosti mini a mikro byly zavedeny především pro přenosná zařízení a kvůli zmenšování veškeré elektroniky. Právě kvůli potřebě mobilního průmyslu byla vydána tzv. OTG (*On-The-Go)* specifikace. Ta umožňuje dynamické přepnutí funkce hostitelského a koncového zařízení. U mini a mikro velikostí byl proto zaveden třetí typ konektoru – AB. Tento typ obsahuje 5. pin – ID pin. Pokud je na připojeném kabelu tento pin uzemněn, OTG zařízení je v hostitelském režimu. Naopak když je tento pin nezapojen, OTG zařízení je v režimu periferie. [14][19]

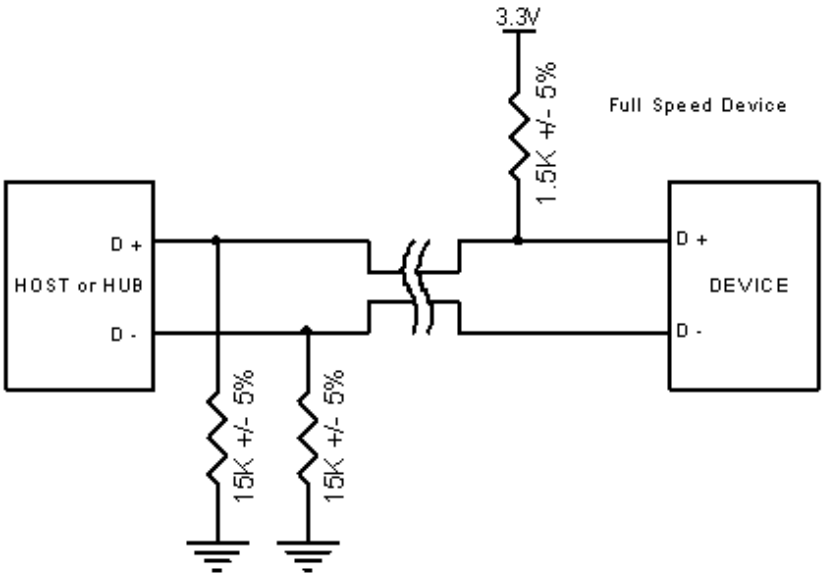



**Obrázek č. 6: Typy USB konektorů**
(Zdroj: 18)

Maximální délka USB kabelu je 5 metrů a je složen ze 4 vodičů. Jeden pár je použit pro napájení, druhý pár je datový. Datový pár musí být kroucený pro konfiguraci *full-speed* a *high-speed*. Napájecí vodiče poskytují napětí 5 V a proud minimálně 100 mA. V případě potřeby je možné poskytnout až 500 mA. [14]

Signál se převádí v diferenční podobě. Ke kódování se používá NRZI (*Non-Return-To-Zero-Inverted*). Toto kódování vyjadřuje logickou „0“ jako změnu stavu a logickou „1“ jako setrvání ve stavu předešlém. Pro prevenci ztráty synchronizace se po každých šesti po sobě jdoucích logických „1“ vkládá „0“ (*bit-stuffing)*. Logická „1“ je určena napětím větším než 2,8 V na lince D+ a menším než 0,3 V na lince D-. V případě logické „0“ lze použít stejnou definici s prohozenými datovými linkami. Přijímací zařízení definuje „1“ když je napětí na lince D+ o 200 mV větší než na D- a logickou „0“ při D+ o 200 mV menší než D-. Polarita může být invertována podle typu rychlosti sběrnice. [19][14]

Fyzická vrstva také určuje rychlost komunikace a to připojením 3,3 V na některou z datových linek přes *pull-up* rezistor. Připojením tohoto rezistoru na linku D+ se komunikace konfiguruje na *full-speed*. *Low-speed* je nastaven připojením rezistoru na linku D-. *High-speed* rychlost se inicializuje jako full-speed a během resetování zařízení se provede *chirp* sekvence. Avšak určení rychlosti komunikace není jediný účel těchto *pull-up* rezistorů. Dále se pomocí nich detekuje nově připojené zařízení a také se určuje, zda jde o periferii nebo hostitele. Hostitelské zařízení má obě datové linky připojeny přes *pull-down* rezistory k zemi. Na obrázku č. 7 lze vidět připojení *full-speed* zařízení. Při prohození datových linek bychom dostali *low-speed* konfiguraci. [19]




**Obrázek č. 7: Full-speed zařízení s pull-up rezistorem na D+**
(Zdroj: 19)

### 3.4.3 Linková vrstva

Linkovou vrstvu USB lze rozdělit na rámce (*frames)*, které se skládají z paketů. Každý rámec začíná přenosem SOF *(start of frame),* v případě *low-speed* rámec končí symbolem EOP *(end of packet)*. USB rámce se zpravidla skládají z pověřovacího (*token),* datového (*data)* a potvrzovacího *(status)* paketu. Pověřovací paket obsahuje popis typu a směru výměny dat, jedná se o hlavičku celého rámce. Jak název napovídá, datový paket přenáší kýžená data. Posledním paketem je potvrzovací, někdy též *handshake* paket. Ten nese informaci o úspěšnosti přenosu. V ojedinělých případech se používá tzv. speciální *(special)* paket. [19]

Shrnutí všech typů paketů lze vidět v tabulce 1. S příchodem *high-speed* konfigurace bylo vytvořeno několik dalších typů, především ve skupině datových paketů. V této práci se s nimi nepracovalo.

**Tabulka č. 1: Přehled typů paketů**
(Zdroj: Vlastní zpracování dle: 19)

| Skupina | Typ | Popis |
|---|---|---|
| **Pověřovací (Token)** | OUT | Přenos od hostitele k periferii. |
| | IN | Přenos od periferie k hostiteli. |
| | SOF | Začátek rámce. |
| | SETUP | Konfigurační přenos hostitele. |
| **Datový (Data)** | DATA0 | Sudý datový paket. |
| | DATA1 | Lichý datový paket. |
| **Potvrzovací (Status)** | ACK | Potvrzení úspěšného přijetí dat. |
| | NAK | Potvrzení neúspěšného přijetí dat. |
| | STALL | Brána je pozastavena, nutný zásah hostitele. |
| **Speciální (Special)** | PRE | Označuje *low-speed* přenos. |

Pro adresování dat se používají tzv. koncové body (*endpoints)*. Ty představují vyrovnávací paměť (dále *buffer*) pro přijímání a odesílání dat. Ačkoliv bezpochyby souvisejí s hardwarem, a tudíž s fyzickou vrstvou, jsou nedílnou součástí vrstvy linkové. Každý rámec obsahuje adresu koncového bodu, od kterého či ke kterému putuje. Koncové body se nacházejí na jednotlivých periferiích. Každá periferie musí mít alespoň jeden obousměrný nebo dva jednosměrné koncové body – EP0, resp. EP0 OUT a EP0 IN. Ten slouží k identifikaci zařízení při připojení, nastavení a provádění kontrolních operací.

USB zařízení může mít až 32 koncových bodů (16 IN a 16 OUT) včetně nultého. Jejich počet a konfigurace určuje funkci periferie. [14][19]

Logická spojení mezi hostitelem a koncovými body periferií se nazývají roury (*pipes)*. Roury jsou definované parametry souvisejícími s přístupem, šířkou pásma sběrnice, typem přenosu a charakteristickými vlastnosti koncového bodu (např. maximální velikost paketu). Mohou být jednosměrné nebo obousměrné. Roura spojující nulové koncové body bývá označována jako výchozí (*default control pipe)*. USB definuje dva typy rour: proudovou rouru a rouru zpráv. [14][15]

Proudové roury *(stream pipes)* nemají definovaný formát a slouží především pro sekvenční přenos dat. Tyto roury jsou jednoznačně určeny koncovým bodem a jeho směrem. Z toho vyplývá, že jsou pouze jednosměrné. Mohou být ovládány jak hostitelem, tak periferii. Mezi podporované typy přenosů patří hromadné, izochronní a přerušovací viz kapitola 3.4.4.

Naproti tomu roura zpráv (*message pipe)* má předem jasně stanovenou strukturu a je ovládána hostitelem. Ten inicializuje celou komunikaci žádostí, na kterou periferie odpovídá daty v požadovaném směru. Z toho vyplývá, že tyto roury musí být obousměrné, nicméně tok je převážně jednosměrný. Jediným podporovaným typem přenosu je přenos řídící a výchozí roura je vždy tohohle typu.

### 3.4.4 Typy přenosů

Ačkoliv konkrétní formát přenášených dat není nijak určen, USB poskytuje několik různých typu přenosů. Každý z nich je charakterizován vlastnostmi, které jsou pevně dané při vzniku roury a neměnné do jejího zrušení. Mezi klíčové parametry jednotlivých typů přenosu patří např. omezení velikosti paketu, požadovaný sled dat nebo řešení chyb v komunikaci.

USB specifikace definují čtyři základní typy přenosů:

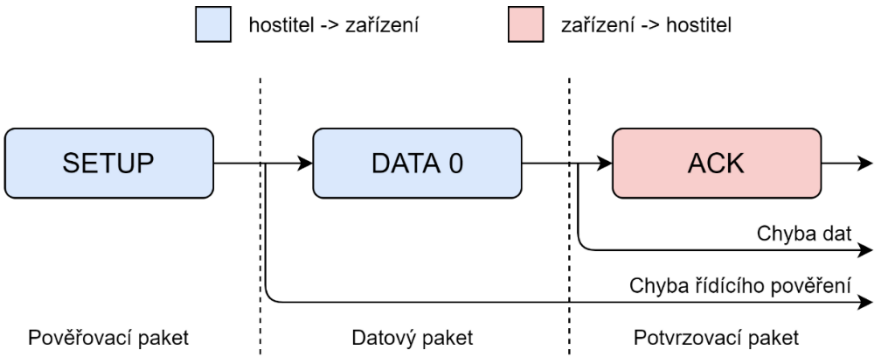
- řídící (*control transfer*),
- hromadný (*bulk tranfser*),
- přerušovací (*interrupt transfer*),
- izochronní (*isochronous transfer*).

**Řídící přenos**

Při připojení je USB periferie vždy označena za obecné zařízení. Jediný možný způsob komunikace v tomto stavu je pomocí řídících přenosů. Ty jsou podporovány pouze na nultém koncovém bodu. Z toho vyplývá, že pro přenos je možné použít pouze rouru zpráv a je použit obousměrný přenos dat. Řídící přenos slouží k zjištění informací o připojené periferii a její enumeraci. Enumerace je proces, kdy hostitel přiřazuje periferii konfiguraci. Řídící přenos může mít až tři části:

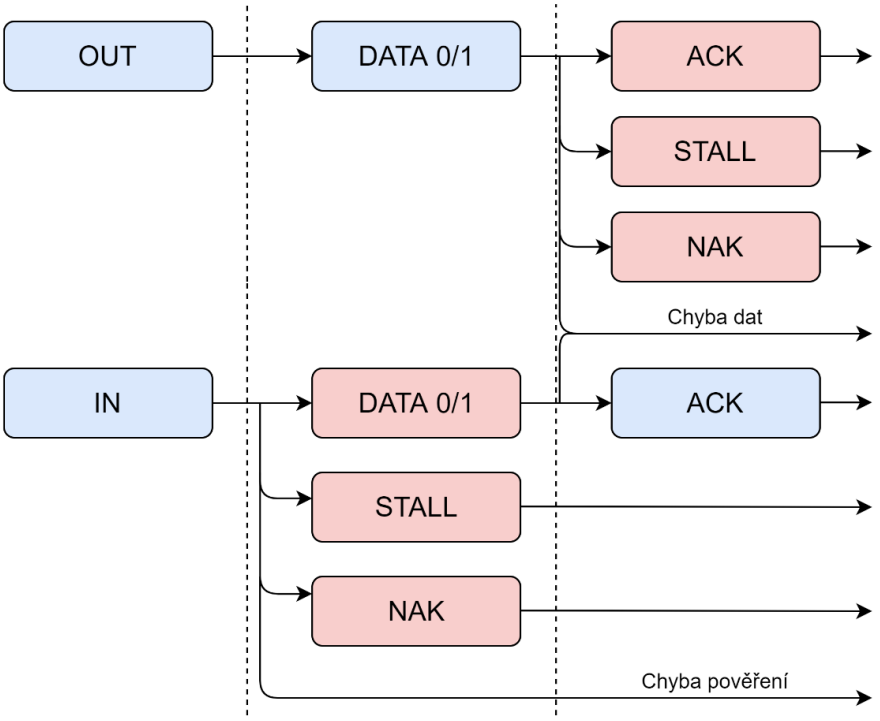
První část je pověřovací. V té hostitel posílá pověřovací paket typu SETUP následovaný datovým paketem, obsahující detaily a typ žádosti. Tento paket je

standardizovaný a existuje osm typů žádostí. Po úspěšném obdržení dat periferie odpovídá potvrzovacím paketem ACK. V případě neúspěchu periferie neodpovídá. Schéma pověřovací části lze vidět na obrázku č. 8.




**Obrázek č. 8: Schéma pověřovací části řídícího přenosu**
(Zdroj: Vlastní zpracování dle: 20, s. 63)

Druhá část je datová. Tato část je volitelná a není vždy součástí. Nicméně v některých případech je využita a může se i několikrát opakovat. Může mít dvě podoby, buďto se data do periferie zapisují, anebo se z ní čtou. Množství dat přenesených v této fázi je určen pověřovací fází. Pakliže je požadované množství dat větší než maximální délka paketu (u zařízení s plnou rychlostí až 64 bytů), jsou tato data rozdělena do více paketů. V takovém případě se musí střídat sudý a lichý datový paket (DATA0 a DATA1 viz tabulka č. 1). Díky tomuto střídání je detekována případná ztráta paketu. Schéma této části lze vidět na obrázku č. 9.




**Obrázek č. 9: Schéma datové části řídícího přenosu**
(Zdroj: Vlastní zpracování dle: 20, s. 64)

Poslední částí řídícího přenosu je stavová část (*status stage)*. Ta má obdobnou strukturu jako datová část. Rozdílem je, že poslaný datový paket má nulovou délku a směr komunikace je opačný oproti datové části. Stavová část je provedena vždy a je nutná pro úspěšné dokončení řídícího přenosu. [14][19][20]

**Hromadný přenos**

Hromadný přenos se používá pro přenos dat, například u tiskáren nebo skenerů. Obsahuje 16bitovou CRC korekci (*Cyclic Redundancy Check* – cyklický redundantní součet) a v případě chyby se data posílají znovu. Existují dva typy hromadného přenosu, přenos z hostitele na periferii (OUT) a v opačném směru (IN).

Pravidla přenosu jsou stejná jako ve druhé fázi řídícího přenosu, viz obrázek č. 9. Hostitel vždy začíná transakci buďto oznámením, že je připraven přijmout data nebo oznámením, že má data připravena pro transakci. V po sobě jdoucích transakcích se střídají datové pakety DATA0 a DATA1 s tím, že přenos vždy začíná paketem DATA0.

Hromadný přenos nemá nijak definovaný počátek ani konec, data jsou posílány podle potřeby. Přenos je uskutečněn na nevyužité šířce pásma po tom, co byly všechny ostatní přenosy alokovány. V případě, že je sběrnice zaneprázdněna, mohou se hromadné přenosy zpomalit. Z toho vyplývá, že jsou vhodné jen při časově nezávislých komunikacích, protože zde není zaručeno časování.

Pro přenos se používá proudová roura a tento přenos je podporován pouze *full-speed* a *high-speed* konfigurací. Pro *full-speed* je maximální velikost paketu až 64 bytů, pro *high-speed* 512 bytů. V případě, že je paket menší, nemusí být doplněn nulami. Přenos je považován za úplný ve dvou případech: při přenesení předpokládaného objemu dat nebo při transakci s nulovou délkou dat či s objemem menším než povolené maximum koncového bodu. [14][19][20]

**Přerušovací přenos**

Přerušovací přenos má využití v případě, kdy je potřeba dodržet u transakcí nějaké časování. Typickou aplikací jsou klávesnice nebo herní kontroléry. Většinou se jedná o periferie nevysílající data často nebo periodicky, ale vyžadující maximální dobu pro vyřízení požadavku.

Pravidla jsou stejná jako u hromadného přenosu, viz obrázek č. 9. V případě směru z periferie k hostiteli se hostitel periodicky dotazuje, zda nebylo vyvoláno přerušení. Perioda dotazování je popsána v deskriptoru koncového bodu. V opačném směru hostitel oznámí, že zahajuje přerušovací přenos a toto oznámení je ihned následováno datovým paketem.

Pro přenos je opět použita proudová roura a přenos je podporován všemi rychlostními konfiguracemi. Pro *full-speed* zařízení je velikost paketu omezena na 60 bytů, pro *high-speed* naopak zvýšena na 1024 bytů. V případě, že zařízení nepošle potvrzovací paket, přenos se opakuje. Dotazování na přerušení od hostitele nemá limit opakování. [14][19][20]

**Izochronní přenos**

Izochronní přenos je využíván především pro přenos dat v reálném čase požadujících konstantní šířku pásma. Objektem takového přenosu je ve většině případů zvuková nebo obrazová stopa. Je zde detekce chyb, nicméně pakety nejsou opakovány v případě chyby. Z toho důvodu zde ztrácí smysl potvrzovací paket. Data jsou přijaty s chybami i bez. Struktura izochronních přenosu je obdobná jako u hromadných přenosů, avšak s absencí potvrzovacího paketu.

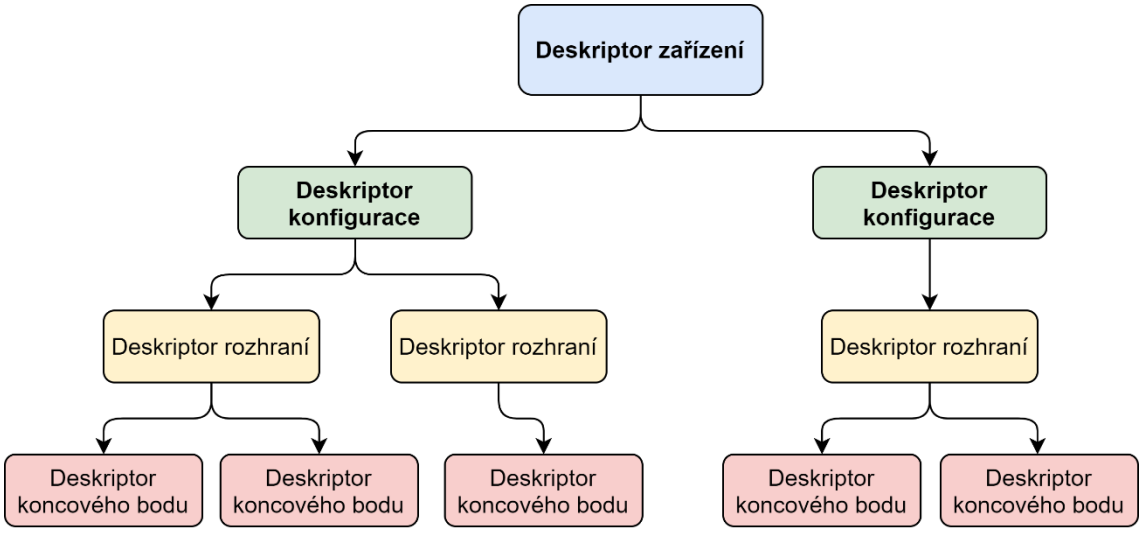
Maximální délka paketu je specifikována deskriptorem koncového bodu a může nabývat až 1023 bytů u *full-speed* a 1024 pro *high-speed* konfigurací. *Low-speed* zařízení izochronní přenos nepodporují. [14][19][20]

### 3.4.5 Deskriptory

Všechny USB zařízení mají určitou hierarchii deskriptorů. Pomocí těch jsou poskytovány informace o periferii hostiteli. Jedná se nejrůznější informace např. o jaké se zařízení se jedná, kdo jej vyrábí, jakou verzi USB podporuje, počet koncových bodů atd. Typů deskriptorů je mnoho, nicméně tato práce se věnuje pouze těm nejběžnějším, mezi něž patří:

- deskriptor zařízení (*device descriptor*),
- deskriptor konfigurace (*configuration descriptor*),
- deskriptor rozhraní (*interface descriptor*),
- deskriptor koncových bodů (*endpoint descriptor*).

Větvení deskriptorů lze vidět na obrázku č. 10. Na vrcholu je vždy jeden deskriptor zařízení. Ostatních deskriptorů může být více.




**Obrázek č. 10: Schéma větvení deskriptorů**
(Zdroj: Vlastní zpracování dle: 16)

### Deskriptor zařízení

Jak již bylo napsáno, deskriptor zařízení může být pouze jeden a má za účel reprezentovat celou periferii. Specifikuje základní, ale důležité informace o zařízení. Dále specifikuje, kolik má periferie deskriptorů konfigurace. Deskriptor zařízení je první deskriptor, který se hostitel snaží získat. Jedná tak z důvodu potřeby parametru *bMaxPacketSize0,* který určuje maximální velikost paketu. Před jeho zjištěním se nemůže hostitel dotázat na více jak osm bytů. Celková délka deskriptoru zařízení je 18 bytů. Jednotlivá pole lze vidět v tabulce č. 2.

**Tabulka č. 2: Struktura deskriptoru zařízení**
(Zdroj: Vlastní zpracování dle: 19)

| Offset [B] | Jméno pole | Délka [B] | Popis |
|---|---|---|---|
| 0 | *bLength* | 1 | Délka deskriptoru. |
| 1 | *bDescriptorType* | 1 | Kód deskriptoru zařízení (0x01). |
| 2 | *bcdUSB* | 2 | Nejvyšší verze USB kompatibilní se zařízením (BCD kódování). |
| 4 | *bDeviceClass* | 1 | Identifikátor třídy zařízení. |
| 5 | *bDeviceSubClass* | 1 | Identifikátor podskupiny třídy zařízení. |
| 6 | *bDeviceProtocol* | 1 | Identifikátor protokolu. |
| 7 | *bMaxPacketSize* | 1 | Maximální délka paketu nultého koncového bodu. |
| 8 | *idVendor* | 2 | Identifikátor výrobce – přiděluje USB-IF. |
| 10 | *idProduct* | 2 | Identifikátor výrobku – přiděluje výrobce. |
| 12 | *bcdDevice* | 2 | Verze výrobku (BCD kódování). |
| 14 | *iManufacturer* | 1 | Index textového řetězce popisující výrobce. |
| 15 | *iProduct* | 1 | Index textového řetězce popisující výrobek. |
| 16 | *iSerialNumber* | 1 | Index textového řetězce sériového čísla. |
| 17 | *iNumConfigurations* | 1 | Počet možných konfigurací. |

Pole *bDeviceClass, bDeviceSubClass* a *bDeviceProtocol* využívá operační systém především k nalezení správných ovladačů zařízení. Nicméně většinou jsou tyto parametry definovány až na úrovni deskriptoru rozhraní a v deskriptoru zařízení jsou uvedeny jako nulové (0x00). Díky tomu může mít jedno zařízení více tříd. [14][16][19]

### Deskriptor konfigurace

Periferie může mí několik různých deskriptorů konfigurací, avšak většina využívá pouze jeden. Pomoci něj se specifikuje napájení zařízení, maximální spotřeba a počet rozhraní. Některá zařízení využívají více těchto deskriptorů, například může být periferie buďto napájena ze sběrnice, anebo mít vlastní zdroj. Nicméně při enumeraci zařízení

hostitel vždy vybírá právě jednu konfiguraci a zbytek během dané relace zůstane nevyužito. Délka deskriptoru konfigurace je 9 bytů, viz tabulka č. 3.

**Tabulka č. 3: Struktura deskriptoru konfigurace**
(Zdroj: Vlastní zpracování dle: 19)

| Offset [B] | Jméno pole | Délka [B] | Popis |
|---|---|---|---|
| 0 | *bLength* | 1 | Délka deskriptoru. |
| 1 | *bDescriptorType* | 1 | Kód deskriptoru konfigurace (0x02). |
| 2 | *wTotalLength* | 2 | Počet bytů deskriptoru konfigurace a jemu podřazených deskriptorů |
| 4 | *bNumInterfaces* | 1 | Počet rozhraní v konfiguraci |
| 5 | *bConfigurationValue* | 1 | Hodnota, kterou hostitel identifikuje konfiguraci při jejím výběru |
| 6 | *iConfiguration* | 1 | Index textového řetězce popisující konfiguraci. |
| 7 | *bmAttributes* | 1 | Nastavení napájení. |
| 8 | *bMaxPower* | 1 | Maximální proud odebíraný zařízením v této konfiguraci. Jednotkou jsou 2 mA. |

Pole *bmAttributes* říká hostiteli, zda je zařízení napájeno ze sběrnice, nebo má vlastní napájení. Dále se zde dá nastavit podpora vzdáleného probuzení (*remote wake up*). Pole *bMaxPower* určuje polovinu maximálního požadovaného odběru (jednotka 2 mA) proudu ze sběrnice. Pokud není požadované množství možné poskytnout, hostitel zamítne konfiguraci zařízení. [14][16][19]

### Deskriptor rozhraní

Na deskriptor rozhraní lze nahlížet jako na hlavičku sdružující koncové body do skupin provádějící danou funkci periferie. Dále je zde určena třída periferie, viz kapitola 3.4.6. Třídy umožňují použití generických ovladačů, díky kterým nemusí výrobce psát ovladače pro každé zařízení zvlášť. Nastavení třídy se provádí v polích *bInterfaceClass*, *bInterfaceSubClass* a *bInterfaceProtocol*.

Deskriptory rozhraní umožňují multifunkčnost periferii, například multifunkční tiskárna se skenerem. První deskriptor rozhraní může popisovat funkci skeneru, zatímco druhý popisuje funkci tiskárny. Na rozdíl od konfiguračních deskriptorů zde není limitace použití více rozhraní simultánně.

Další nespornou předností deskriptorů rozhraní je pole *bAlternativeSetting*. To umožňuje přiřadit jednomu rozhraní jeho alternativu a díky tomu měnit jeho nastavení za chodu. [14][16][19]

Délka deskriptoru rozhraní je 9 bytů a jeho strukturu lze vidět v tabulce č. 4.

**Tabulka č. 4: Struktura deskriptoru rozhraní**

(Zdroj: Vlastní zpracování dle: 19)

| Offset [B] | Jméno pole | Délka [B] | Popis |
|---|---|---|---|
| 0 | *bLength* | 1 | Délka deskriptoru. |
| 1 | *bDescriptorType* | 1 | Kód deskriptoru rozhraní (0x04). |
| 2 | *bInterfaceNumber* | 1 | Identifikátor rozhraní. Číslováno od nuly |
| 3 | *bAlternateSetting* | 1 | Identifikátor alternativního rozhraní. |
| 4 | *bNumEndpoints* | 1 | Počet koncových bodů využitý rozhraním (bez nultého koncového bodu). |
| 5 | *bInterfaceClass* | 1 | Identifikátor třídy zařízení. |
| 6 | *bInterfaceSubClass* | 1 | Identifikátor podskupiny třídy zařízení. |
| 7 | *bInterfaceProtocol* | 1 | Identifikátor protokolu. |
| 8 | *iInterface* | 1 | Index textového řetězce popisující rozhraní. |

### Deskriptor koncového bodu

Každý koncový bod musí mít svůj deskriptor. Výjimkou je nultý koncový bod, ten je standardizován, a kromě jeho délky (parametr *bMaxPacketSize*, který hostitel získává z deskriptoru zařízení) jsou všechny jeho parametry dané. Deskriptor koncových bodů slouží k určení typu přenosu, šířce pásma a periodě dotazování. Jeho délka je 7 bytů a strukturu lze vidět v tabulce č. 5.

**Tabulka č. 5: Struktura deskriptoru koncového bodu**

(Zdroj: Vlastní zpracování dle: 19)

| Offset [B] | Jméno pole | Délka [B] | Popis |
|---|---|---|---|
| 0 | *bLength* | 1 | Délka deskriptoru. |
| 1 | *bDescriptorType* | 1 | Kód deskriptoru koncového bodu (0x05). |
| 2 | *bEndpointAdress* | 1 | Adresa koncového bodu v rámci zařízení. |
| 3 | *bmAttributes* | 1 | Typ podporovaného přenosu. |
| 4 | *wMaxPacketSize* | 2 | Maximální velikost paketu. |
| 6 | *bInterval* | 1 | Interval přenosu dat. |

Pole *bmAttributes* určuje typ přenosu. První dva bity určují jeden ze 4 druhů přenosu a zbylých šest je rezervováno pro nastavení synchronizace při použití izochronního přenosu. Pole *bEndpointAddress* určuje adresu koncového bodu. První tři bity určují číslo koncového bodu a poslední bit jeho směr – 0 pro směr OUT a 1 pro směr IN. [14][16][19]

### 3.4.6 Třídy zařízení

Většinu USB zařízení lze rozdělit do skupin podle funkcí. Tyto skupiny se nazývají třídy. Jejich hlavním účelem je definovat jeden standard pro celou skupinu periferií. Díky těmto standardům je jednodušší upravovat firmware a těží z toho i výrobci, kteří nemusí ke každému zařízení vydávat ovladače. Většina tříd je definována neziskovou organizací USB-IF, nicméně mnoho výrobců definuje vlastní třídy, které mají tak velké využití, že se stanou součástí standardu. Příkladem může být specifikace *Bluetooth USB*, kterou vytvořila skupina *Bluetooth Special Interest Group*. V současné době je několik desítek tříd.

Norma je dělí podle místa definice. Některé třídy je třeba definovat v deskriptoru zařízení, jiné v deskriptoru rozhraní a existují i třídy, které mohou být definovány v libovolném z těchto deskriptorů. Každá třída má definovaný požadovaný a volitelný počet koncových bodů. Dále může třída specifikovat formát dat a hodnoty některých polí v deskriptorech. Některé třídy definují i vlastní deskriptory, například třída rozbočovačů musí obsahovat deskriptor rozbočovačů.

Náhled všech tříd lze nalézt na oficiálních webových stránkách USB. [22] Vzhledem k zaměření této práce bude podrobněji rozebrána pouze třída *Communication Device Class* (dále jen CDC).

#### Communication Device Class

Tato třída byla vytvořena za účelem podpory telefonní komunikace, nicméně během vývoje byla přidána i síťová komunikace a další komunikační funkce včetně virtuálních sériových portů.

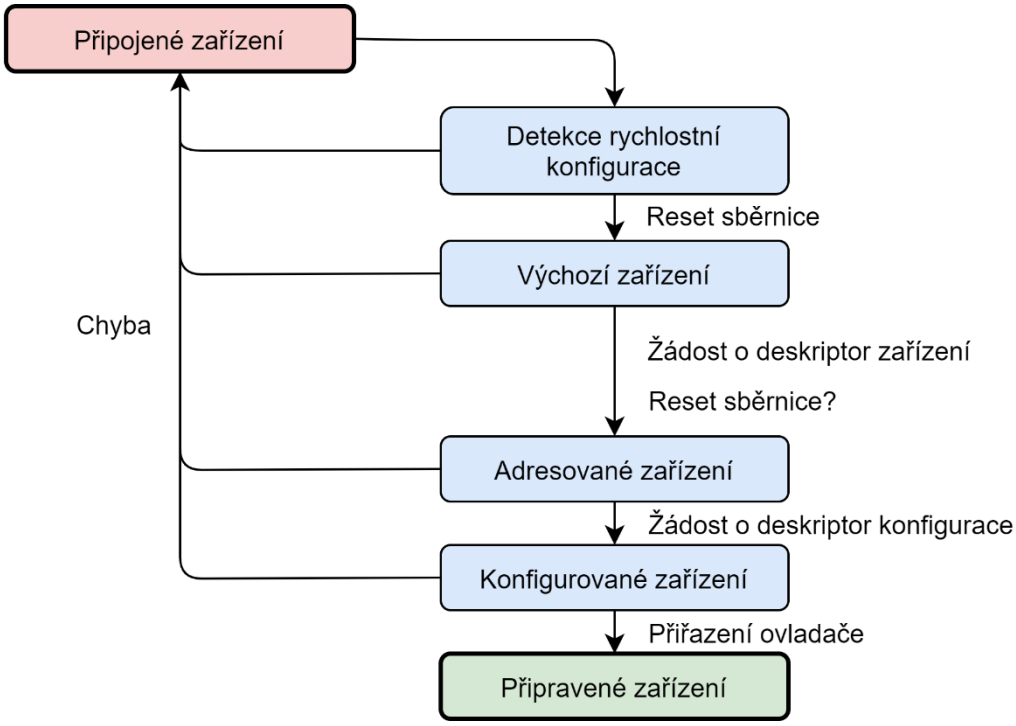
CDC je zodpovědná za řízení periferie, komunikace a transakci dat. Řízení periferie zahrnuje kontrolu a konfiguraci periferie a upozorňování hostitele na výskyt nejrůznějších událostí. Řízení komunikace znamená především navazování a ukončování komunikací. Všechny řídící a upozorňovací příkazy jsou standardizovány a jejich seznam je dostupný na stránkách nástroje *USBlyzer*. [23] Tato třída může být specifikována v deskriptoru zařízení nebo deskriptoru rozhraní. Jestliže je definována již v deskriptoru zařízení, pak všechna rozhraní musí náležet této třídě.

Třída definuje mnoho podskupin tříd. Nejdůležitější z nich je pro aplikaci sériového monitoru tzv. *Abstract Control Model*. Tato podskupina vyžaduje a definuje oproti běžným deskriptorům následující: *CDC Header Functional Descriptor, CDC Union Functional Descriptor, Call Management Functional Descriptor* a *ACM Functional Descriptor*. [24]

CDC třída vyžaduje konfiguraci třech koncových bodů. Jeden koncový bod pro přerušovací přenos ve směru od periferie k hostiteli kvůli upozorňování hostitele. Dále dva koncové body pro hromadné přenosy v obou směrech.

### 3.4.7 Enumerace zařízení

Enumerace je proces rozpoznání zařízení a zvolení jedné z jeho konfigurací. Tento proces se dá rozdělit do několika fází a celý jeho průběh lze vidět na obrázku č. 11. Následující odstavce popisují enumeraci u operačního systému Windows, v němž byla tato práce realizována.




**Obrázek č. 11: Schéma enumerace USB zařízení**
(Zdroj: Vlastní zpracování dle: 25)

Proces začíná připojením nové periferie nebo zapnutím systému, kde je periferie připojena. Tím je zařízení považováno za připojené a může odebírat až 100 mA. Následuje detekce rychlostní konfigurace. Ta je popsána v kapitole 3.4.2 a jedná se o detekci *pull-up* rezistoru na jedné z datových linek. Po úspěšné detekci rozbočovač resetuje zařízení. V případě, že se zařízení nachází ve *full-speed* konfiguraci, je nutné zjistit, zda nejde o *high-speed*. Avšak každé zařízení v *high-speed* konfiguraci musí podporovat řídící přenosy i ve *full-speed* nastavení.

Po procesu zjištění rychlostní konfigurace se zařízení nachází ve stavu výchozím. Je mu přiřazena adresa 0x00 a pro komunikaci jsou použity řídící přenosy. Pomocí těch se hostitel dotazuje na jednotlivé deskriptory.

Nejprve se dotazuje na prvních osm bytů deskriptorů zařízení, viz kapitola 3.4.5. Obsahem těchto osmi bytů je i informace o maximální velikosti paketu. Po přečtení může následovat reset zařízení, nicméně od verze USB 2.0 tento reset není požadován. V případě neúspěchu enumerace se při dalších pokusech reset použít musí. Před další žádostí hostitel přiřadí periferii její unikátní adresu. Od tohoto momentu všechna

komunikace s tímto zařízením využívá tuto adresu. Následuje druhý dotaz na deskriptor zařízení, tentokrát už hostitel obdrží celý deskriptor.

Hostitel pokračuje v zjišťování informací od periferií pomocí dotazování na jednotlivé deskriptory konfigurace a deskriptory jim podřazené. První žádost je vždy na 255 bytů. V případě, že odpovědí od periferie je právě 255 bytů, dotazuje se hostitel znovu, tentokrát na délku obsaženou v položce *wTotalLength*. Ta upřesňuje, kolik bytů mají podřazené deskriptory včetně konfiguračního. Dřívější verze Windows se nejprve dotazovaly na 9 bytů k zjištění celkové délky a druhým dotazem obdržely celý strom deskriptorů. Po obdržení všech deskriptorů hostitel vybere konfiguraci a zařízení je považováno za konfigurované.

Následuje poslední krok, kterým je přiřazení ovladačů. To je uskutečněno na základě tzv. INF souborů. Jedná se o textové soubory, které popisují zařízení. Windows má databázi takových souboru a při výběru ovladačů hledá největší shodu dvou INF souborů. Ty byly vytvořeny na základě tříd zařízení nebo jsou dodány výrobcem. Pro již enumerované zařízení může mít systém uloženou informaci o předchozím přiřazení ovladačů. Po výběru ovladačů je zařízení považováno za připravené a je možné jej používat. [14][16][25]

# 4 Upřesnění cílů

Na základě rešeršní části a konzultací s vedoucím práce byly upřesněny cíle práce a stanoveny základní požadavky:

1. Hardwarové zařízení sériového monitoru je realizováno na platformě od firmy *STMicroelectronics*.
2. Algoritmus pro sběr dat musí být dostatečně robustní a nezatěžovat mikroprocesor při stavu nečinnosti.
3. Datové linky musí být vhodně odděleny, aby bylo možné rozeznat, ze kterého analyzovaného kanálu data přišla.
4. Parametry analyzovaných kanálů musí být nastavitelné, nejlépe z počítače.
5. Pro zobrazení dat je použit software *Wireshark* a je proto nutné realizovat propojení hardwarového zařízení s tímto softwarem.
6. Vytvoření vlastních dekódovacích skriptů pro zvolený protokol a jejich následné otestování.

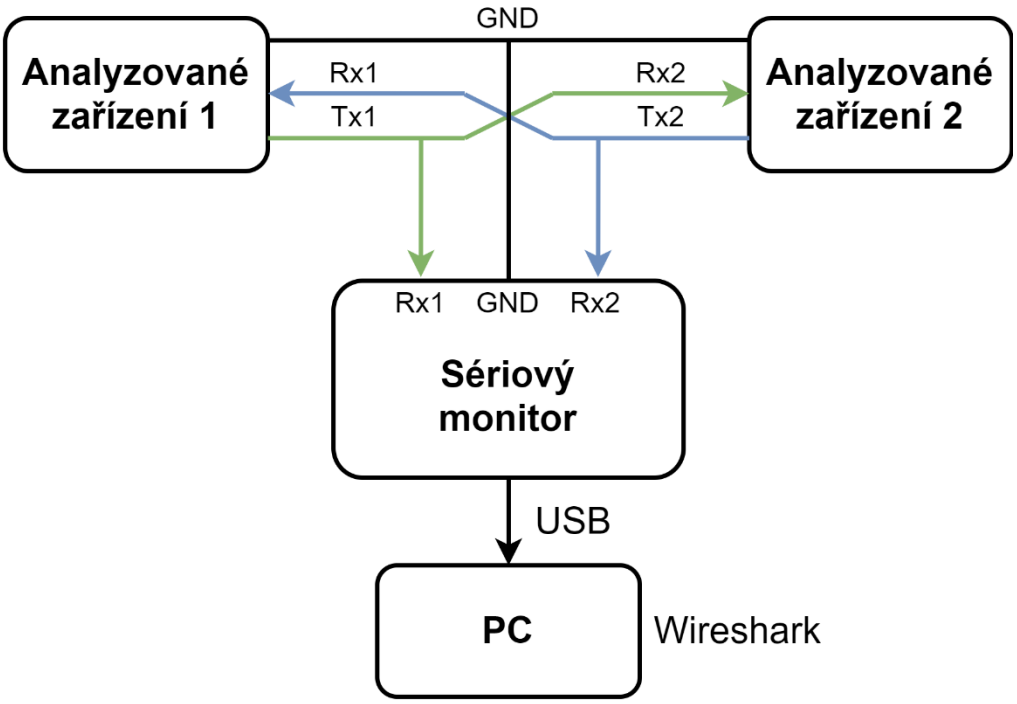
Všechny tyto body musí být splněny tak, aby byla dodržena co největší univerzalita zařízení. Tato vlastnost je klíčová pro zaručení širšího využití sériového monitoru a zároveň poskytuje prostor pro případná budoucí rozšíření.

# 5 Analýza řešení

Tato kapitola se zaobírá analýzou možných přístupů k vytvoření sériového monitoru. Součástí kapitoly je výběr hardwaru dle požadavků na zařízení. Dále jsou rozvedeny možnosti realizace jednotlivých funkcí monitoru, které jsou specifikovány v předchozí kapitole.

Základní schéma celého procesu analýzy lze vidět na obrázku č. 12. Stěžejní části je mikrokontroler s rozhraním UART, který je připojen paralelně k datové komunikaci dvou analyzovaných zařízení. Úkolem mikrokontroleru je sběr dat a jejich distribuce pomocí USB sběrnice. Data musí být vhodně oddělena, aby se dal poznat jejich původ a nedošlo k jejich záměně. Dále je nutné vytvořit vhodnou metodu nastavování parametrů UART periferie. Ty musí být možné nastavit uživatelem, aby bylo dosaženo maximální univerzality.

Na straně počítače je využit software *Wireshark*, který poskytuje několik možností pro zobrazení komunikace USB zařízení, a navíc umožňuje uživateli vytvořit vlastní rozhraní pro zachycení externí komunikace (*external capture – extcap).* V posledním kroku je nutné přijatá data dekódovat. *Wireshark* nabízí několik způsobů, jak dekódovací skript vytvořit. V tomto ohledu je *Wireshark* naprosto bezkonkurenční. Disponuje rozsáhlou dokumentací, plně podporuje vývojářskou komunitu a díky své univerzalitě a modularitě najde využití ve velkém počtu aplikací.




**Obrázek č. 12: Schéma sériového monitoru**

## 5.1 Výběr hardwaru

Součástí cílů práce byla volba vhodného hardwarového řešení. Ta spočívala ve výběru vhodného mikrokontroleru a dále rozhodnutí, zda bude použita již hotová vývojová sada, nebo vytvořena vlastní deska plošných spojů. Toto rozhodnutí záviselo především na dostupnosti a vhodnosti vývojové sady. Mezi hlavní požadavky, dle kterých byl zvolen mikrokontroler patřilo:

- mikrokontroler od výrobce *STMicroelectronics* (dále STM), který bude mít podporu v software *STM32CubeIDE*,
- dostatečný počet použitelných UART rozhraní, dvě z těchto rozhraní musí mít stejnou a nejlépe plnou funckionalitu,
- funkce *line idle detection* rozhraní UART či obdobná,
- podpora USB, možnost pracovat jako *full-speed* USB zařízení.

STM je výrobce polovodičů a elektroniky s dlouholetou tradicí. Jedná se o největšího výrobce mikroprocesorů v Evropě. Jejich software *STM32CubeIDE* je vývojový nástroj zahrnující nastavení periferii, generování koster kódů, kompilaci a odlaďování pro mikrokontrolery a mikroprocesory od STM. Tento ekosystém vznikl v dubnu roku 2019 díky akvizici IDE (*Integrated Development Enviroment) Attolic TrueSTUDIO*. STM má velice rozsáhlou nabídku zahrnující mikroprocesory, mikrokontrolery a nepřeberné množství modulů. Dále nabízí i nejrůznější vývojářské sady. [26]

Druhou podmínkou je počet UART rozhraní. Toto rozhraní je v dnešní době u mikrokontrolerů samozřejmostí, nicméně pro aplikaci sériového monitoru je nutné, aby mikrokontroler podporoval dvě rozhraní se stejnou funkcionalitou. Většina produktů má jedno plně funkční rozhraní, které je doplněno dalšími s omezenými funkcemi. Mezi velmi zajímavé funkce patří podpora simultánní komunikace využívající DMA (*Direct Memory Access*) a hardwarové řešení automatické detekce rychlosti komunikace (*ABRD/ABR – automatic* baud *rate detection)*. Tyto funkce nejsou podmínkou, nicméně jejich využití by mohlo mít přínos pro sériový monitor. Obě dvě funkce budou rozebrány podrobněji dále v práci.

Velice důležitou funkcí UART rozhraní je *idle line* detekce. Většina mikrokontrolerů od STM podporuje tuto detekci a zároveň ji velice podobnou funkci *receiver timeout*. Tyto funkce dokážou spustit přerušení UART rozhraní při neaktivitě komunikace. Rozdílem je skutečnost, že *idle line* spustí přerušení po neaktivitě po dobu jednoho rámce, zatímco *receiver timeout* má nastavitelný časový interval, po kterém je přerušení vyvoláno. Pro účely této práce mají tyto funkce veliké využití, viz kapitola 6.1.

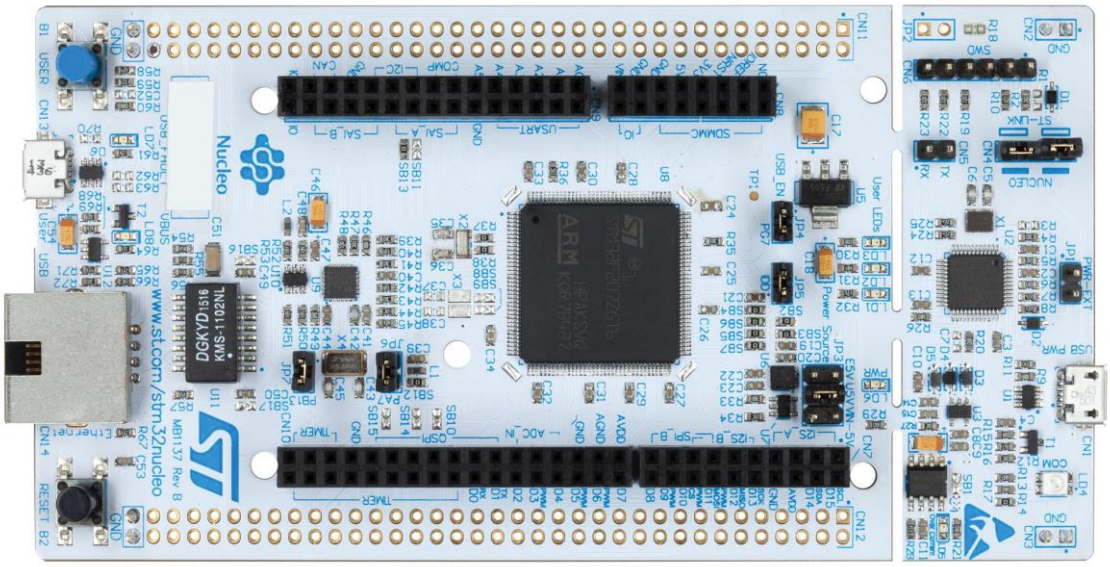
Poslední podmínkou je podpora USB. Mikrokontrolery od STM tuto funkci ve většině případů podporují. Velké množství mikrokontrolerů podporuje dokonce USB *On-the-go*, což pro aplikaci této práce není třeba. Sériový monitor bude po celou dobu fungovat jako USB zařízení.

Ke svým mikrokontrolerům poskytuje STM i nabídku hotových vývojových sad. Ty dělí na tři skupiny s názvy: *DISCOVERY, EVAL a NUCLEO*. Velká část *DISCOVERY* sad je určena pro aplikace s nízkou spotřebou, jako je například *IoT (Internet of Things)*. Dále obsahuje sady s již vestavěným LCD displejem pro tvorbu nejrůznějších aplikací s grafickým rozhraním. Skupina *EVAL* obsahuje sady pro měření. Jedná se o poměrně velká a drahá zařízení, která zahrnují mnoho vstupních a výstupních pinů, LCD displej a nejrůznější druhy konektorů od mikrofonů přes kamery až po Ethernetové rozhraní.

Třetí skupinou je *NUCLEO*, která má představovat cenově dostupnou skupinu pro tvorbu prototypů. *NUCLEO* vydává sady ve třech hlavních konfiguracích. Jedná se o jednotky s 32, 64 nebo 144 vstupy/výstupy. STM se s touto skupinou snaží o univerzalitu a jednoduchost, aby využití *NUCLEO* sad zahrnovalo co nejširší oblast. Ze všech tří skupin se *NUCLEO* jeví jako nejpoužitelnější pro sériový monitor. Konfigurace s 32 a 64 vstupy/výstupy bohužel nemá na desce zakomponované USB rozhraní. To zahrnuje jen řada se 144 vstupy/výstupy. Všechny tyto *NUCLEO* sady poskytují USB zařízení ve *full-speed* konfiguraci.

Poslední podmínkou je množství a funkcionalita UART rozhraní. V tomto ohledu se jako nejlepší volbou jeví mikrokontroler *STM32F303ZE*, ten nabízí až pět U(S)ART rozhraní, z nichž tři s plnou funkcionalitou. [28, s. 31] Tyto tři rozhraní podporuji *ABRD, recevier timeout interrupt* i komunikaci pomocí DMA. Vybraný mikrokontroler je navíc i součástí jedné z vývojových sad *NUCLEO*, konkrétně *NUCLEO-F303ZE*.

Tato sada má všechny funkce potřebné pro realizaci sériového monitoru a jedná se i o cenově velmi přívětivé řešení. Na internetových stránkách distributora *Premier Farnell* lze tento kus pořídit za cenu 466 Kč. [27] Součástí sady je i integrovaný *ST-LINK*, pomocí kterého se zařízení programuje a odlaďuje, a zároveň je sada i součástí *STM32CubeIDE*. Sadu lze vidět na obrázku č. 13.



**Obrázek č. 13: Použitá vývojová sada *NUCLEO-F303ZE***
(Zdroj: 26)

## 5.2 UART periferie

Jak již bylo zmíněno výše, vývojová sada *NUCLEO-F303ZE* obsahuje tři U(S)ART periferie se stejnou funkcionalitou. USART3 je ve výchozím nastavení použit pro komunikaci s integrovaným *ST-Link* programátorem. Zbylé dvě periferie je možné použít. Tyto rozhraní jsou schopné komunikovat až rychlostí 9 Mb/s. [28, s. 29] Sada dále obsahuje další dvě rozhraní pouze v asynchronní konfiguraci a bez některých funkcí. Všechna UART rozhraní je možno programovat pomocí knihoven *HAL* (*Hardware Abstraction Layer* nebo *Hardware Annotation Library*). Tato knihovna umožňuje tři způsoby komunikace pomocí UART rozhraní.

Prvním způsobem je tzv. *Polling mode*. Jedná se o nejjednodušší, ale zároveň nejméně vhodnou implementaci. Při jejím použití se procesor neustále dotazuje (*polling*), zda nebyl přijat nový byte. Z principu je jasné, že je to způsob zatěžující procesor. Velkou nevýhodou je šance ztráty informací a daná metoda může fungovat pouze v jednoduchých aplikacích a pro nízké rychlost (9600 baud *rate* nebo nižší). Pro komplexní aplikace je tato metoda nejméně vhodná.

Druhým způsobem je použití přerušení. Při přijetí bytu periferie spustí přerušení a procesor jej zpracuje. Jedná se o velmi častý způsob zpracování dat, který funguje pro běžné rychlosti (až 115200 baud *rate*). Nevýhodou může být provedení přerušovací rutiny při každém přijatém bytu, což může zapříčinit odložení ostatních procesů.

Třetí možností je přijímat data za použití DMA. *Direct Memory Access*, neboli přímý přístup do paměti, je funkce, která umožňuje přenášet data v rámci pamětí a periferii bez zásahu procesoru. Na rozdíl od normálního přenosu dat, kde strojové instrukce dává procesor, je DMA řízeno speciální řadičem, který je součástí mikrokontroleru. Procesor díky tomu není brzděn komunikací s periferii a může vykonávat jiné činnost. Ke konfliktu může dojít jedině při přístupu do paměti. Při použití DMA je procesor potřeba jen při prvotní inicializaci a při dokončení operace. Díky tomu dochází k paralelizaci a mikrokontroler může z hlediska uživatele vykonávat více procesů najednou. DMA je vhodné i pro ty nejvyšší rychlosti. Nevýhodou je, že musíme předem znát množství dat, které má být přijato/odesláno.

DMA může pracovat ve dvou konfiguracích: s normální nebo cirkulární (*circular*) vyrovnávací pamětí. Při použití normálního *bufferu* jsou data přijímány/odesílány, dokud není *buffer* naplněn, poté je komunikace ukončena. U cirkulárního se po jeho naplnění začíná zapisovat zase od začátku a předchozí data jsou přepsána. Konfigurace *bufferu* se zadává při inicializaci.

Z výše uvedených informací je jasné, že pro aplikaci sériového monitoru se nejlépe hodí použití DMA s cirkulárním *bufferem*. Zařízení musí pracovat co nejrychleji, aby bylo použitelné i pro přenosy s vysokou rychlostí. Neznalost množství dat kompenzuje použití cirkulárního *bufferu*.

## 5.3 Rozlišení datových linek

Z principu navrženého sériového monitoru, viz obrázek č. 12, lze vidět problém v rozlišení datových linek. Zatímco výstupem z monitoru je jedna USB linka, vstupem jsou dvě sady dat z UART periferií. Rozlišení původu těchto dat je nutné pro správnou analýzu.

Za nejběžnější metodu rozlišení lze považovat označování dat při jejich zpracování v mikrokontroleru. Tato metoda spočívá v sekvenci, ve které jsou data po přijetí z UART periferie označena buď nějakým kódem pro jednotlivé analyzované zařízení, nebo kódem a časovou značkou. Časová značka má využití při pozdější analýze. Velkou výhodou této metody je jednoduchá realizace. Nicméně hlavním cílem firmwaru sériového monitoru je minimální náročnost na mikroprocesor, což vkládání značek nesplňuje. Při každé zpracovávací rutině by musel algoritmus k přijatým datům přidat textový řetězec. V případě použití i časové značky by se musely inicializovat hodiny a přibyla by další přerušení, která by mohla ovlivnit práci procesoru. Označování dat a časovým značkám se podrobněji věnuje diplomová práce Ing. Jana Perného. [21]

Druhou možností je využití komplexnosti USB rozhraní. Ta poskytuje možnost vytvoření tzv. kompozitního zařízení. Kompozitní zařízení je periferie podporující víc než jednu třídu zařízení. Spousta zařízení na trhu spojuje více tříd do jednoho zařízení. Příkladem mohou být sety myši a klávesnice připojeny pouze jedním USB kabelem. Jejich deskriptory obsahují dvě rozhraní s třídou HID (*Human Interface Device*), jedno rozhraní pro myš a druhé pro klávesnici. Hostitel je pak schopen rozpoznat podle adresy koncových bodů, od které části zařízení přijímá data.

Pro aplikaci sériového monitoru by se daly využít dvě oddělená CDC. V praxi by to znamenalo, že by zařízení emulovalo dva virtuální komunikační porty (*VCP – Virtual Communication Port*). Díky dvěma portům by mohla být přijatá data rozdělena podle původu a každý port by přenášel data pouze z jedné UART periferie. V počítači by se sériový monitor identifikoval jako dva COM porty. Tato funkce by umožnila výběr analyzovaného zařízení dle situace a do budoucna by mohla poskytnout prostor pro další rozšíření. Tím by mohlo být například přidání i druhého směru komunikace. Díky tomu by uživatel mohl z počítače injektovat zařízení a testovat komunikaci jednoho zařízení simulací druhého. Vytvoření kompozitního zařízení se jeví jako vysoce efektivní a sofistikovaná metoda pro rozlišení datových linek.

## 5.4 Konfigurace UART parametrů

Pro správné fungování monitoru je nutné, aby uživatel byl schopen nastavit parametry UART periferií. Jedná se především o nastavení přenosové rychlosti, nicméně pro univerzálnost zařízení je vhodné takto nastavovat i další parametry: délku slova,

paritu a počet stop bitů. Způsobů, jak k tomuto problémů přistupovat, je mnoho. Níže uvedené podkapitoly popisují některé z možných přístupů.

### 5.4.1 Manuální přepínač

Jako nejjednodušší způsob nastavení parametrů by mohl být použit manuální přepínač neboli DIP *(Dual In-Line Package)* přepínač. Jedná se o součástku skládající se z několika dvojic kontaktů, které lze nezávisle na sobě spojit nebo rozpojit. Rozpojení a spojení je určeno malou páčkou.

Tento přepínač by mohl být zapojen do vývojové sady a pomoci něj by se volily konfigurace. Jedná se o velice efektivní a jednoduchou metodu, nicméně by konfigurace byla omezena počtem přepínačů. Například u přenosové rychlosti by musely být zvoleny jen význačné rychlosti a ostatní zůstat bez podpory. Navíc při použití přepínačů i pro délku slova, paritu a počet stop bitů by počet externích součástek vzrostl a zařízení by již nebylo robustní. Z těchto důvodů nebylo s touto možností dále pracováno.

### 5.4.2 Konfigurační rozhraní

Druhou vybranou metodou je vytvoření konfiguračního rozhraní. V druhé části kapitoly 5.3 je popsáno kompozitní zařízení. To by šlo využít i pro tento problém. Pomocí kompozitního zařízení by byla vytvořena tři rozhraní. Dvě pro komunikaci a jedno pro konfiguraci zařízení. Toto třetí konfigurační rozhraní by mohlo být třídou CDC a pro nastavení parametrů by byl použit vlastní protokol, který by v sobě nesl informaci o rychlosti, délce slova, paritě a počtu stop bitů. Na straně počítače by fungovala aplikace generující tento protokol na základě požadavků uživatele. Mikrokontroler by v případě přijetí dat z tohoto rozhraní dešifroval protokol a resetoval UART periferie s novým nastavením.

Toto řešení by zachovalo univerzalitu, nicméně se jedná o poměrně komplikovaný způsob. Třetí rozhraní určené ke konfiguraci celého sériového monitoru může nalézt své využití v budoucnu v případě rozšíření zařízení o další funkce.

Vlastní protokol by bylo možné využít i na jednom z rozhraní určených pro data. Při přijetí dat by mikrokontroler zkontroloval přítomnost označení. V případě, že by jej data obsahovala, spustil by nastavovací rutinu. To by ovšem značně snížilo univerzalitu a muselo by se předcházet situacím, kdy analyzovaný protokol obsahuje stejné označení jako vlastní protokol.

### 5.4.3 Přenesení VCP parametrů

Třetí variantou je možnost využít vlastností CDC třídy. Při využití podskupiny třídy *ACM* je v počítači emulován virtuální sériový port. Při otevírání tohoto portu v počítači lze nastavit parametry sériové komunikace. Ty jsou následně při řídících procesech CDC třídy distribuovány k zařízení. Příkaz, kterým hostitel posílá nastavené parametry se nazývá `SET_LINE_CODING`. V případě sériového monitoru tyto parametry

nepotřebujeme, protože využíváme sběrnici USB, nicméně lze je využít pro nastavení UART periferií.

Tato metoda se jeví jako velice univerzální a efektivní. Virtuální COM porty je před začátkem komunikace vždy nutné nejprve otevřít. Uživatel je může otevřít se stejnou konfigurací na jaké fungují analyzované zařízení a mikrokontroler dle této konfigurace reinicializuje své UART periferie. Z těchto důvodů byla tato metoda dále v práci použita.

### 5.4.4 Automatická baud rate detekce

Poslední možností je použití automatické baud *rate* detekce (dále jen ABR). Ta spočívá v determinaci rychlosti pomocí smluveného prvního bitu, nebo sekvence bitů (*syncword)*. Díky tomu je umožněno přijímat data z různých zařízení konfigurovaných na různé rychlosti bez jejich předešlé znalosti. Existují dvě možnosti implementace: hardwarová nebo softwarová.

Mikrokontroler *STMF303ZE* má tuto detekci implementovanou hardwarově a díky tomu umožňuje jednoduchou realizaci ABR. Implementace umožňuje čtyři možné konfigurace, jež se liší v podobě smluvených bitů. Celé nastavení ABR je popsáno v dokumentaci firmy *STM*. [29]

Druhým způsobem je implementace softwarová. Ta přináší volnost v realizaci použitého algoritmu. Celá detekce je tvořena periferiemi a funkcemi mikrokontroleru a je tudíž možné vytvořit ABR, které nebude závislé na smluvených sekvencích bitů. Algoritmus by mohl měřit časy mezi vzestupnou a sestupnou hranou a po určitém počtu měření vybrat minimální naměřenou hodnotu a vypočítat z ní baud *rate.* Taková implementace by využívala modularity jednotlivých pinů. Pro měření dob mezi hranami lze využít buďto externího přerušení (*external interrupt – EXTI*) nebo přímo časovače (*timer*) v módu *input capture*.

*EXTI* je konfigurace GPIO (*General-Purpose Input/Ouput*) pinu, která při detekci hrany vyvolá přerušení. To by mohlo být využito pro zjištění doby mezi jednotlivými hranami. Veliká výhoda tohoto řešení je skutečnost, že drtivá většina pinů vývojové sady má tuto funkci implementovanou. *STMF303ZE* má tuto funkci na všech pinech, na kterých je současně vyvedena i datová linka Rx. Nevýhodou takového řešení je nutnost použití dalšího hodinového signálu ke generování času.

Druhou možností je využití časovače v módu *input capture*. Fungování je obdobné jako v předchozím případě. Při detekci hrany je vyvoláno přerušení, nicméně zde jej vyvolává samotný časovač a proto v rutině přerušení stačí pouze přečíst hodnotu času a není nutné inicializovat další hodinový signál. Tento způsob se velice často používá při nejrůznějších analýzách pulzní šířkové modulace. Tato konfigurace je vyvedena na pinu datové linky Rx pouze u periferie USART2.

Z výše uvedených informací je patrné, že softwarová implementace ABR může mít pozitivní přínos pro nastavení UART periferie. Její hlavní výhodou je, že není potřebná předchozí znalost rychlosti analyzovaných zařízení a zároveň není nutné zajišťovat smluvní sekvenci bitů. Za nevýhodu lze považovat fakt, že při měření doby

mezi dvěma hranami nelze zaručit, že výsledná minimální doba skutečně odpovídá jednomu bitu. Další nevýhodou je, že ABR řeší pouze rychlost a ostatní parametry nenastavuje. Ve většině aplikací není parita použita a nastavuje se jeden stop bit, avšak v praxi mohou nastat situace, kdy je potřeba tyto parametry změnit. Proto je ABR závislé na dané aplikaci a jeho využití je situační.

## 5.5 Propojení se softwarem Wireshark

*Wireshark* je znám především jako síťový analyzátor, nicméně svou působnost průběžně rozšiřuje. Pro analýzu USB sběrnice je určen nástroj *USBPcap*. Dále *Wireshark* umožňuje vytvořit vlastní rozhraní pro jakoukoliv externí komunikaci. Tento nástroj se nazývá *extcap* a je možné jej použít i pro USB sběrnici.

### 5.5.1 USBPcap

*USBPcap* je aplikace třetích stran, která měla původně jen rozšířit *Wireshark*. Jednalo se o softwarový analyzátor bez grafického rozhraní, který využíval knihovny *Wireshark*. Nicméně tento nástroj byl natolik oblíbený, že se vývojáři rozhodli jej zakomponovat přímo do grafického rozhraní *Wireshark*. V dnešní době je tak jeho plnohodnotnou součástí.

*USBPcap* zobrazuje v softwaru *Wireshark* všechny kořenové rozbočovače přítomné v počítači. Pomocí tohoto propojení je *USBPcap* schopný odchytávat data z *USB Requests Blocks* (*URBs*), které jsou přenášeny v *I/O Request Packets* (*IRP*). *Wireshark* prezentuje tyto pakety jako rámce. Důležitým poznatkem je, že paket z *USBPcap* ne vždy odpovídá paketu USB specifikace a rámec ve *Wireshark* neodpovídá rámci USB. *USBPcap* zároveň není schopen zachytit veškeré elementy komunikace, mezi které patří:

- stavy sběrnice (*Handshake, Power ON, Power OFF, Reset, High Speed Detection*),
- *packet ID* (*PID*),
- rozdělené transakce (*CSPLIT, SSPLIT*),
- časy a doby přenosů a jeho rychlost (*full-speed, low-speed, high-speed*). [30]

Zároveň nelze pozorovat celou enumeraci USB. Řídící přenosy jsou zobrazeny až po přiřazení adresy zařízení.

*USBPcap* podporuje všechny čtyři druhy přenosů, viz kapitola 3.4.4. V aplikaci sériového monitoru se jedná hlavně o hromadný (*bulk*) přenos. Ten je podporován a *Wireshark* zobrazuje rámec jako skutečný rámec hromadného přenosu dle USB specifikací. To znamená, že spojuje jednotlivé datové pakety jednoho hromadného přenosu a zobrazuje je jako celek.

Obrovskou výhodou tohoto nástroje je jeho dostupnost. Jedná se o velmi užitečný nástroj, pomoci kterého lze rychle a jednoduše sledovat veškerou komunikaci na USB sběrnici. Nicméně má několik nedostatků. *USBPcap* je stále ve vývoji a má problémy s fungováním na některých kořenových rozbočovačích, kde nelze vybrat pouze jeden kýžený port. Druhým vývojovým problémem je neúplná podpora USB 3.0. Avšak jako hlavní nedostatek v aplikaci sériového monitoru je nutnost externího otevření virtuálních sériových portů. *USBPcap* funguje čistě jako analyzátor, tudíž pouze sleduje komunikaci a neumožňuje otevření portů. Bez jejich otevření žádná komunikace neprobíhá.

### 5.5.2 Extcap

Rozhraní *extcap* (*external capture*) je univerzální zásuvný modul (*plugin)*, který umožňuje zachycovat externí komunikace a analyzovat je ve *Wireshark.* Bývá použit při analýze netradičních zdrojů dat a sledování komunikace esoterických hardwarů. *Extcap* umožňuje, aby takové komunikace byly zahrnuty a konfigurovány přímo v grafickém rozhraní *Wireshark*. Jedná se o binární soubor či skript umístěný ve složce určené pro *extcap*. Při spuštění *Wireshark* spustí všechny *extcap* rozhraní za použití externích překladačů.

*Wireshark* poskytuje kostru a příklad kódu ve své dokumentaci. [31] Jedná se o script v jazyce *Python* a jsou zde uvedeny příklady argumentů a jednotlivých prvků rozhraní. Na webu je možné najít i hotové příklady komerčních zařízení, které ke svému hardwaru dodávají *extcap* rozhraní. Například firma *Nordic Semiconductors* poskytuje ke svým vybraným vývojovým sadám tzv. *nRF Sniffer tool*. Jedná se o nástroj pro odlaďování chyb u *Bluetooth Low Energy* (*BLE*) aplikací pomocí detekce paketů mezi analyzovanými zařízeními, a to i přes šifrování této komunikace. [32] Součásti tohoto nástroje je i *extcap.* který je volně ke stažení na stránkách *GitHub* a je neustále vyvíjen a zlepšován. [33]

*Extcap* rozhraní se jeví jako velice užitečný nástroj pro analýzu sériové komunikace ve *Wireshark*. Jeho hlavní výhodou je jeho univerzalita a možnost vytvoření vlastního rozhraní. Zatímco *USBPcap* zachycuje celou USB komunikaci, *extcap* by mohl pracovat pouze se sériovou komunikací a obsahem datových paketů. Další výhodou je vytvoření vlastní konfigurace, která by umožnila otevřít virtuální komunikační porty přímo ve *Wiresharku* a rovnou zde upřesnit i další parametry. Tím by odpadla nutnost použití externího softwaru pro otevření portů. *Extcap* by rovněž mohl být využit jako nástroj pro rekonstrukci rozdělených rámců. Tím by se ulehčila práce dekódovacím skriptům, které by vždy obdržely celistvý rámec. Celkově se *extcap* jeví jako univerzálnější nástroj, který přináší prostor pro vlastní realizaci a případnou modularitu do budoucna.

## 5.6 Dekódování protokolů

Velkou výhodou softwaru *Wireshark* je možnost vytváření vlastních dekódovacích skriptů (disektorů). Díky tomu je v dnešní době dostupných více než 1500 disektorů pro nejrůznější protokoly. Disektory mohou mít dvě podoby: modul přímo zakomponovaný v hlavním programu nebo *plugin* v podobě sdílené knihovny či *DLL* (*Dynamic-Link Library*). *Plugin* se hodí v případě vývoje, kdy je potřeba neustále testovat, zatímco vestavěný disektor dává smysl až ve finální podobě.

Úlohou disektorů je rozšifrovat protokol do čitelné podoby. Analyzovaný protokol je rozčleněn na jednotlivé elementy, kterým je následně přiřazena hodnota. Existuje mnoho způsobů, jak psát disektory, nicméně mezi tři základní a nejpoužívanější patří:

- *Wireshark Generic Dissector (*WSGD*),*
- skriptovací jazyk *Lua,*
- programovací jazyk *C.*

WSGD je *plugin*, díky kterému je možné poměrně rychle vytvořit disektor pro jednoduché protokoly. Jeho tvůrce je *Oliver Aveline* a veškerá dokumentace je dostupná na oficiálních webových stránkách WSGD. [34] Jedná se *plugin* spojující definice v čitelné podobě, které jsou interpretované nebo kompilované do disektoru. Avšak kvůli své jednoduchosti má množství limitací a není vhodný pro složitější protokoly.

Skriptovací jazyk *Lua* je odlehčený jazyk určený pro rozšiřující aplikace. *Wireshark* obsahuje vestavěný *Lua* překladač, který je možné použít pro psaní disektorů. Díky tomu stačí skripty v *.lua* formátu pouze vložit do složky */plugins* a *Wireshark* je při spuštění nebo při použití klávesové zkratky CTRL+SHIFT+L (*Reload Lua Plugnis*) sám přeloží. To činí z tohoto nástroje velice efektivní způsob vytváření disektorů. Další výhodou je rozsáhlá dokumentace na webu *Wireshark.* [17] Nevýhodou je, že všechny funkce *Wireshark* nejsou součástí *LuaAPI* (*Application Programming Interface*), avšak toto rozhraní je stále rozšiřováno.

Třetí základní možností je použití programovacího jazyka *C*. Jedná se o velice podobnou koncepci jako v případě skriptovacího jazyka *Lua* s tím rozdílem, že jsou zde dostupné všechny funkce softwaru *Wireshark.* Nevýhodou je nutná kompilace při každé úpravě kódu a absence hlubší dokumentace. Nicméně ta je nahraditelná stovkami již existujících disektorů.

*Wireshark* vydal na jedné ze svých událostí *SharkFest* porovnání těchto tří metod pro dekódování jednoduchého protokolu. [35][36] Z jejich výsledků je patrné, že WSGD a *Lua* jsou jednoduchými a rychlými způsoby vývoje, ale zároveň jsou i funkčně limitované. Disektory psané v *C* mají výhodu v podpoře všech funkcí, nicméně musí být zkompilovány a sestaveny se softwarem *Wireshark*. Vytvoření disektoru pro demonstrační protokol zabralo ve WSGD 124, v *C* 270 a v *Lua* pouze 89 řádků. Zároveň

autoři uvádějí, že pro malé množství dat není viditelný rozdíl v rychlosti zpracování. Avšak pro velké soubory (~ 1M pakety) jsou výsledky následující:

- WSGD*:* 1:42.1 – 9.1krát pomalejší než *C*,
- *Lua*: 0:27.0 – 2.4krát pomalejší než *C*,
- *C*: 0:11.2.

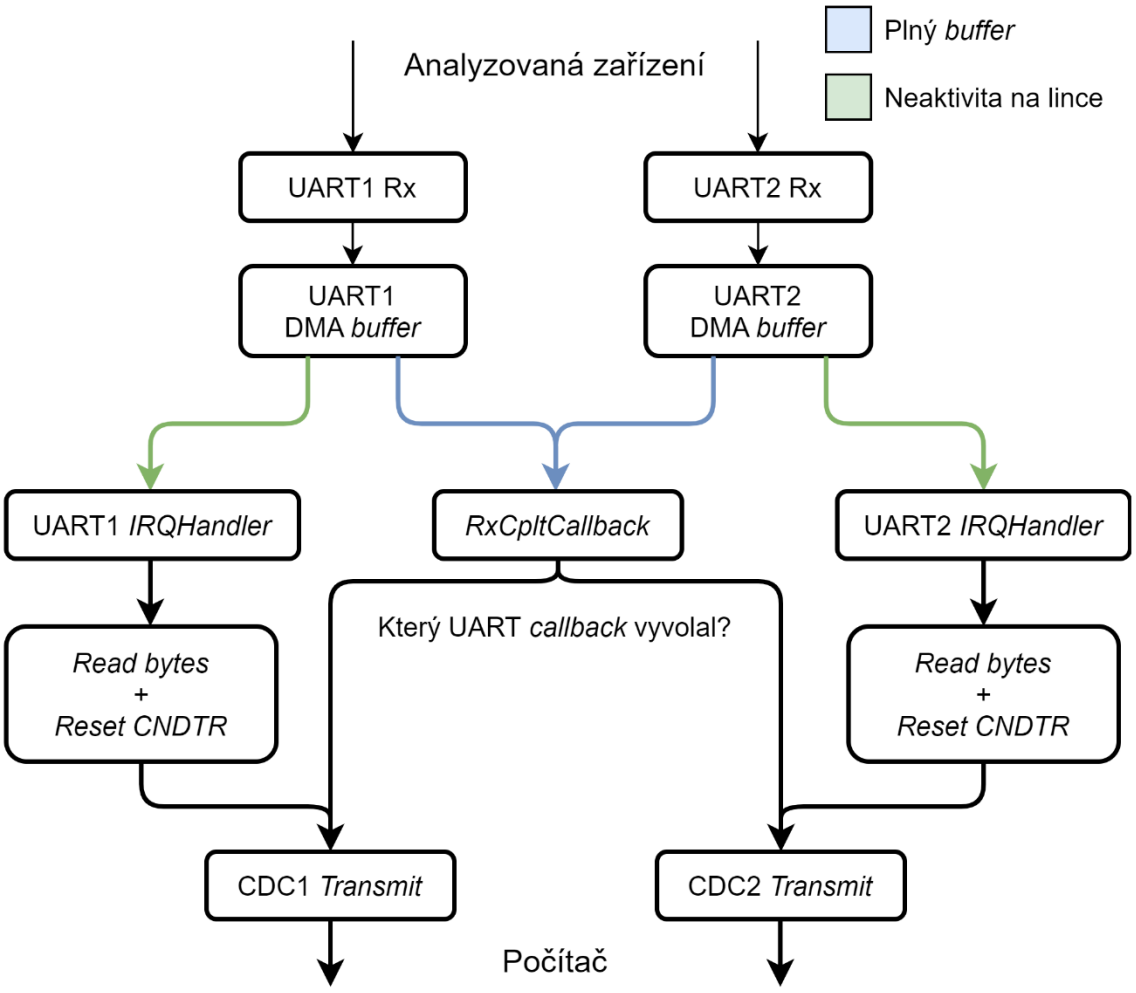
Ze všech uvedených výhod a nevýhod je patrné, že pro tuto práci bude nejvhodnějším způsobem využití skriptovacího jazyku *Lua*. Ten poskytuje rozsáhlou dokumentaci a dostatek funkcí. Zároveň je také vhodný pro prvotní vývoj, protože chyby jsou lehce opravitelné.

# 6 Realizace hardwarového zařízení

Tato kapitola se zabývá popisem realizace hardwarového zařízení. V první části je popsán algoritmus sběru dat z rozhraní UART. Další část se věnuje funkcím USB CDC zařízení a tvorbou kompozitního zařízení. V závěru kapitoly je popsána konfigurace UART parametrů a volba vhodné metody pro tuto problematiku.

## 6.1 Algoritmus pro sběr dat

Algoritmus pro sběr dat je stěžejní částí této práce. Stará se o čtení dat z analyzovaných zařízení a jejich vhodnou distribuci. Jeho cílem je spolehlivost při co největším zatížení analyzovaných datových linek a co nejnižší zátěž mikroprocesoru při stavu nečinnosti. Schéma algoritmu lze vidět na obrázku č. 14 a jeho popis je předmětem dalších odstavců.




**Obrázek č. 14: Schéma algoritmu pro sběr dat.**

Na začátku algoritmu je inicializace UART periferií. Ty jsou nastaveny pomocí softwaru *STM32CubeMX*. S nimi jsou nakonfigurovány i DMA *buffery*, konkrétně v cirkulárním módu, každá UART periferie má svůj *buffer*. Následující tabulka popisuje piny jednotlivých UART periferií.

**Tabulka č. 6: Popis pinů UART periferií.**

| Rozhraní | Tx | Rx |
|---|---|---|
| **UART1** | PC4 | PC5 |
| **UART2** | PA2 | PA3 |
| **UART3** | *ST-link* | *ST-link* |

Při spuštění sériového monitoru je inicializováno přijímání dat z UART periferií pomocí DMA, viz kapitola 5.2. Velikost *bufferů* byla nastavena na 512 bytů. Dále je při spuštění rovněž inicializován *receiver timeout interrupt*. Toto přerušení je nastaveno pomocí následující sekvence kódu:

```
huart1.Instance->CR2 |= USART_CR2_RTOEN;
huart1.Instance->CR1 |= USART_CR1_RTOIE;
huart1.Instance->RTOR = 0.0011 * baud;
```

V prvních dvou krocích jsou do kontrolních registru (`CR1` a `CR2`) zapsány hodnoty pro zapnutí funkce *receiver timeout* a jejího přerušení. V kroku posledním je definován minimální počet bitů, po který musí být linka neaktivní, aby došlo k tomuto přerušení.

Tato hodnota byla nastavena na $0.0011 \cdot baud$. Jedná se o krajní dobu, pro kterou je mikroprocesor a USB rozhraní schopno zpracovat předešlý plný DMA *buffer*. Velikost DMA paměti a hodnota `RTOR` registru spolu úzce souvisí. Při zvětšení velikosti DMA *bufferu* musí být zvýšena hodnota `RTOR` registru. Zařízení musí být schopno odeslat 512 bytů za tuto dobu. Pro baud *rate* 115 200 byla tato hodnota zjištěna experimentálně jako 128 bitů. Pomocí přepočtu $\frac{128}{115200} = 0.0011$ lze získat univerzální konstantu pro libovolný baud *rate*.

Po inicializaci je algoritmus ve stavu, kdy mohou nastat dvě situace. Buďto se *buffer* zaplní, nebo bude linka neaktivní.

Při zaplnění *bufferu* je zavolána funkce `HAL_UART_RxCpltCallback`. Toto zavolání je provedeno automaticky UART periferií. Funkce sestává z jednoduché rutiny. Nejprve je zjištěno, která UART periferie tuto funkci zavolala a poté je obsah *bufferu* odeslán USB sběrnici.

Druhou situací je stav, kdy v DMA *bufferu* zůstanou data, ale linka již není aktivní. V tomto případě UART periferie vyvolá přerušení, a mikrokontroler začne

vykonávat funkci `USART_IRQHandler` (*Interrupt ReQuest Handler*). Obě UART periferie mají vlastní manipulátor pro přerušení. Obsahem této funkce je následující kód:

```
if(__HAL_UART_GET_FLAG(&huart1, UART_FLAG_RTOF)){
	uint16_t num_bytes_received;
	__HAL_UART_CLEAR_IT(&huart1, UART_CLEAR_RTOF);
	num_bytes_received = DMA_BUF_SIZE - huart1.hdmarx->Instance->CNDTR;

	if (num_bytes_received != 0){
		huart1.hdmarx->Instance->CCR &= ~DMA_CCR_EN;
		huart1.hdmarx->Instance->CNDTR = DMA_BUF_SIZE;
		huart1.hdmarx->Instance->CCR |= DMA_CCR_EN;
		CDC_Transmit_FS(0, rx1_buff, num_bytes_received);
	}
}
```

Po spuštění manipulátoru je nejprve zkontrolováno, co vyvolalo přerušení. V případě, že se skutečně jedná o neaktivní linku, je registr vyvolávající toto přerušení resetován do výchozí podoby. Následně je spočítáno, kolik nových bytů se nachází v DMA *bufferu*. K tomu slouží položka `CNDTR`, která udává počet zbývajících bytů do přetečení zásobníku. Pomocí této informace lze jednoduše zjistit, kolik bytů je třeba odeslat. V případě, že je tento počet nenulový, je přikročeno k resetu položky `CNTDR` a odeslání dat. Resetování je provedeno vypnutím kanálu, přepsáním hodnoty `CNDTR` a zapnutím kanálu. Po této operaci zbývá pouze odeslat neúplný buffer pomocí funkce `CDC_Transmit_FS`. Tato funkce bude popsána v kapitole 6.2, respektive 6.3.

## 6.2 USB CDC zařízení

Prvotní nastavení USB zařízení se opět provádí v softwaru *STM32CubeMx*. Zde se povolí USB a nastaví se třída CDC. Nicméně software neumožňuje možnost přidat více rozhraní a vytvořit tak kompozitní zařízení. To je potřeba vytvořit manuálně a tvorba je pospána v kapitole 6.3. Po povolení USB zařízení jsou do firmwaru zahrnuty ovladače pro tuto sběrnici a dále skripty obsahující základní funkce, pomocí kterých se USB periferie ovládá.

Generování kódu pomocí *STM32CubeMx* je bohužel stále ve vývoji a může obsahovat chyby. Pro úspěšné inicializování USB zařízení je nutné manuálně přidat *pull-up* rezistor na linku D+[1]. Ačkoliv je rezistor implementován v mikrokontroleru, jeho použití je nutné přidat do funkce `HAL_PCDEx_SetConnectionState` v souboru *usbd_conf.c*. Rezistor má výstup na pinu PG6. V případě, že je vstupující argument funkce `state` roven 1, což značí připojení USB kabelu, je na tento pin přivedeno napětí.

Rezistor je možné přidat také externě, a to mezi piny 3.3 V a PA12. PA12 je pin, kde je vyvedena linka D+. Díky tomuto zásahu dojde k úspěšné enumeraci USB zařízení.

[1] Viz kapitola 3.4.2 a obrázek č. 7

Pro ovládání USB CDC zařízení existují čtyři základní funkce: `CDC_Init_FS`, `CDC_Control_FS`, `CDC_Receive_FS` a `CDC_Transmit_FS`.

`CDC_Init_FS` slouží pouze pro inicializaci vyrovnávacích pamětí, pro každý směr komunikace jednu. Zde je definována i velikost těchto *bufferů*, která byla nastavena stejně jako velikost DMA *bufferů* – 512 bytů.

`CDC_Control_FS` má na starosti řídící transakce s hostitelem. Tato funkce obsahuje *switch* strukturu s vybranými řídícími funkcemi, které bývají provedeny až po enumeraci USB zařízení. Mezi tyto řídící funkce patří také `SET_LINE_CODING`, ve které jsou přijaty parametry sériové komunikace od hostitele.

Třetí funkcí je `CDC_Receive_FS`, která je v této práci využita jen okrajově. Jejím úkolem je přijímat data z USB periferie. Pro účely samotného sériového monitoru je tento směr komunikace nepotřebný, nicméně pro budoucí rozšíření tohoto zařízení může mít využití. Pomoci této funkce by uživatel mohl mít k monitoru připojeno jen jedno analyzované zařízení a v počítači simulovat zařízení druhé. Tímto způsobem by mohl testovat chování v nejrůznějších konfiguracích. Dále by pomocí této funkce mohla být injektována komunikace dvou zařízení a testována jejich odezva.

Stěžení funkcí pro sériový monitor je `CDC_Transmit_FS`. Ta má na starost odesílání dat do PC. Ve výchozí konfiguraci jsou vstupy do této funkce pouze data a jejich délka. Nejdříve je zkontrolován stav Tx linky. V případě že je linka zaneprázdněna je odeslání dat potlačeno. Právě kvůli této podmínce musí být nastavena délka pro přerušení *receiver timeout* podle velikosti DMA *bufferu.*

Funkce `CDC_Transmit_FS` ve svém výchozím tvaru obsahuje jednu chybu. V kapitole 3.4.4 v sekci o hromadných přenosech jsou popsány podmínky pro dokončení transakce. Ta je považována za úplnou v případě, že byl přenesen požadovaný objem dat nebo byl poslední datový paket nulový nebo menší než maximální velikost paketu. To znamená, že pro přenos dat o objemu stejném jako je maximální velikost paketu nebo její násobek, je potřeba za tyto data přidat paket nulový (*zero-length packet*). To bohužel vygenerovaný kód pomocí *STM32CubeMx* nevykonává a je nutné jej upravit. V ovladači pro řízení periferií (*PCD – Peripheral Control Driver) stm32f3xx_hal_pcd.c* ve funkci `PCD_EP_ISR_Handler`[2] je potřeba změnit podmínku pro dokončení transakce. Její rozšířená opravená podoba je následovná:

```
if (ep->xfer_len == 0 && (ep->xfer_count < ep->maxpacket)){
    /* TX COMPLETE */
    HAL_PCD_DataInStageCallback(hpcd, ep->num);
}
else{
    HAL_PCD_EP_Transmit(hpcd, ep->num, ep->xfer_buff, ep->xfer_len);
}
```

Kde `xfer_len` je délka současného paketu, `xfer_count` je délka minulého paketu a `xfer_buff` je *pointer* na transferovaná data.

[2] *Endpoint Interrupt Service Routine* (*EP_ISR*)

Původní podmínkou pro ukončení byla pouze nulová délka současného paketu. Nicméně v případě, že předchozí transakce měla maximální velikost paketu, nebyl odeslán nulový paket a transakce byla neúplná. Z toho důvodu byla vytvořena nová podmínka, která dovoluje ukončit transakci pouze v případě, že je délka současného paketu nulová a zároveň byl předchozí paket menší než maximální délka paketu. V případě, že tato podmínka splněna není, je odeslán nulový paket a v příštím průchodu již podmínka splněná bude.

## 6.3 Kompozitní USB zařízení

Pro rozlišení datových linek bylo využito kompozitního USB zařízení. To umožňuje na jednom USB zařízení provozovat více rozhraní. Tuto funkci je bohužel možné vytvořit pouze manuálně úpravou ovladačů pro USB zařízení.

### 6.3.1 Úprava deskriptorů

V kapitole 3.4.5 byly vysvětleny jednotlivé druhy deskriptorů. Vytvoření kompozitního zařízení zahrnuje úpravu původních deskriptorů USB zařízení s jednou CDC třídou na zařízení obsahující třídy dvě. Deskriptory lze nalézt ve skriptu *Middlewares\Class\CDC\Src\usbd_cdc.c*. Nachází se zde všechny deskriptory pro různé rychlostní konfigurace. *NUCLEO-F303ZE* nabízí pouze *full-speed* konfiguraci, proto je název upravované proměnné `USBD_CDC_CfgFSDesc`.

Deskriptor zařízení zůstává neměnný a první změna je až v deskriptoru konfigurace. Zde je nutné přidat počet rozhraní, které konfigurace nabízí. Dále je součástí deskriptoru konfigurace i celková délka všech deskriptorů „pod“ ním. Původní hodnota tohoto pole byla 75, ke které se musí přičíst počet bytů v druhém rozhraní - 66. Výsledná hodnota tohoto pole je tedy 141.

Deskriptory rozhraní jsou stěžejní částí tvorby kompozitního zařízení. Podle původního deskriptoru bylo vytvořeno rozhraní druhé. Tyto dvě rozhraní se liší pouze v položce *bInterfaceNumber*. Součástí těchto deskriptorů jsou i deskriptory, které vyžaduje CDC třída[3].

Posledním krokem konfigurace byla úprava deskriptorů koncových bodů. Každé CDC rozhraní vyžaduje tři koncové body. Dva z nich jsou určeny pro transakce dat z a do zařízení pomocí hromadných přenosů. Třetí slouží pro konfiguraci zařízení s využitím přerušovacích přenosů.

Koncové body jsou určeny parametrem *bEndpointAdress*. První tři bity tohoto parametru určují číslo koncového bodu. V hexadecimálním zápisu se jedná o část druhé cifry (od nuly do sedmi). Poslední bit obsahuje informaci o směru: 8 pro směr do zařízení,

[3] *CDC Header Functional Descriptor, CDC Union Functional Descriptor, Call Management Functional Descriptor* a *ACM Functional Descriptor.*

0 pro směr k hostiteli. K původním třem koncovým bodům byly přidány další tři. Adresy všech koncových bodů obou CDC rozhraní shrnuje tabulka č. 7.

**Tabulka č. 7: Adresy koncových bodů USB zařízení.**

| Rozhraní | Hromadný IN | Hromadný OUT | Přerušovací IN |
|---|---|---|---|
| **CDC0** | `0x81` | `0x01` | `0x82` |
| **CDC1** | `0x83` | `0x03` | `0x84` |

### 6.3.2 Úprava funkcí a přiřazení paměti

Druhou část tvorby kompozitního zařízení je úprava všech funkcí CDC třídy operujících s USB sběrnicí. Ty jsou v původním stavu použitelné pouze pro jedno rozhraní, avšak součástí této sběrnice jsou díky úpravě deskriptorů rozhraní dvě. Jedná se o funkce, zmíněné v kapitole 6.2 a všechny jejich podfunkce, které využívají. Podfunkce jsou součástí ovladačů *usbd_cdc.c*, zatímco funkce pro uživatele se nachází v *usbd_cdc_if.c*.

K realizaci je potřeba upravení definice struktury pro manipulaci s CDC rozhraním `USBD_CDC_HandleTypeDef`. Původní obsah této struktury byl vložen do nové struktury označující rozhraní, o které se jedná. Toto označení nabývá hodnot 0 a 1. Dále bylo nutné vytvořit pole, které tomuto označení přiřadí správné adresy koncových bodů. Všechny funkce přistupující do této struktury musely být upraveny dle její nové podoby[4].

Posledním krokem je přiřazení PMA (*Packet Memory Area*) adresy jednotlivým koncovým bodům. PMA je část SRAM (*Static Random Access Memory* – statická paměť), kterou má mikrokontroler vyhrazenou pro USB zařízení. Velikost této paměti je 512 bytů. Úkolem PMA je poskytnout uložiště pro pakety přicházející nebo odcházející z jednotlivých koncových bodů. Bohužel dokumentace k ní není příliš obsáhlá.

Přiřazení PMA adres se vykonává při inicializaci USB periferie ve funkci `USBD_LL_Init` ve skriptu *usbd_conf.c*. Ve výchozí podobě je tu přiřazena paměť nulovému koncovému bodu a třem koncovým bodům CDC třídy. Z toho vyplývá, že musí být přidána druhá trojice adres pro koncové body druhé CDC třídy. Toto přiřazení vypadá následovně:

```
HAL_PCDEx_PMAConfig((PCD_HandleTypeDef*)pdev->pData , 0x03 , PCD_SNG_BUF,
0xC0);
HAL_PCDEx_PMAConfig((PCD_HandleTypeDef*)pdev->pData , 0x83 , PCD_SNG_BUF,
0x100);
HAL_PCDEx_PMAConfig((PCD_HandleTypeDef*)pdev->pData , 0x84 , PCD_SNG_BUF,
0x140);
```

[4] Kromě funkce *CDC_Init_FS*, kde se inicializují vždy všechny *buffery* a není třeba znát, o které rozhraní jde.

Prvním argumentem funkce je manipulátor pro řízení periferií, druhým je adresa koncového bodu, třetí argument je volba mezi jednoduchou nebo dvojitou vyrovnávací pamětí. Dvojitá paměť umožňuje mít pořád jednu část paměti volnou, zatímco mikrokontroler využívá část druhou. To může mít přínos pro izochronní přenosy. [37, s. 1057] Jednoduchá vyrovnávací paměť je určena výrazem `PCD_SNG_BUF`, zatímco dvojitá `PCD_DBL_BUF`.

Poslední parametr konfigurační funkce je adresa v PMA. Rozdíly adres by měly odpovídat maximální délce paketu. To je v případě hromadných přenosů 64 bytů, zatímco u konfiguračních přerušovacích přenosů pouze 8 bytů. Adresy první trojice koncových bodů jsou pro hromadný přenos `0x38` a `0x78` a pro přerušovací `0xb8`.

Po připojení zařízení k počítači a úspěšné enumeraci se emulují dva virtuální komunikační porty, se kterými jde samostatně komunikovat. Na straně sériového monitoru je volba komunikačního portu závislá na hodnotě parametru `idx`. Tento parametr je využit především při nastavování parametrů UART periferie, viz kapitola 6.4 a dále při rozlišení datových linek v algoritmu pro sběr dat, viz kapitola 6.1

## 6.4 Konfigurace UART parametrů

Další nutnou součástí firmwaru je konfigurace UART periferii. Z analýzy možných přístupů vyplývá, že jako řešení, které zachová univerzalitu a zároveň bude plně funkční, se nabízí přenášení parametrů z virtuálních komunikačních portů. Nicméně při analýze zařízení s neznámou rychlostí by bylo vhodné využít automatickou baud *rate* detekci. Jako nejpraktičtější způsob ABR se jeví její softwarová implementace, při které není nutné zajišťovat přítomnost smluvených bitů.

Z těchto důvodů bylo využito manuálního *DIP* přepínače. Pomoci něj může uživatel přepnout mezi automatickou baud *rate* detekcí a přenesením parametrů z komunikačních portů. *DIP* přepínač byl zapojen mezi piny PA9 a PA8. První zmíněný byl zapojen na vstup a bylo na něj přivedeno napětí. Druhý pin byl připojen k výstupu *DIP* přepínače a bylo na něm zjišťováno napětí.

Pro komunikaci sériového monitoru s počítačem je vždy nutné nejprve emulované virtuální komunikační porty otevřít. Při otvírání si hostitel (počítač) a zařízení vyměňují konfigurační zprávy. Jednou z konfiguračních zpráv je i `SET_LINE_CODING`. V té hostitel posílá zařízení parametry sériové komunikace. Tato zpráva je součástí funkce `CDC_Control_FS`. Mikrokontroler touto funkcí prochází vždy při otevření sériových portů. Z toho vyplývá, že právě tato funkce je vhodná pro přečtení stavu *DIP* přepínače a rozhodnutí, kterou metodou se budou nastavovat UART parametry. V případě, že je na pinu PA8 stav „*low*“, budou se parametry číst z konfiguračních zpráv. V opačném případě se spustí automatická baud *rate* detekce.

### 6.4.1 Přenesení VCP parametrů

Při stavu „*low*“ je spuštěna následující sekvence. Nejprve mikrokontroler podle argumentu `idx` funkce `CDC_Control_FS` zjistí, které rozhraní mu posílá konfiguraci sériového přenosu. Parametry hostitel posílá ve struktuře `LineCoding`. První čtyři byty struktury nesou informaci o rychlosti, pátý byte určuje počet stop bitů, šestý paritu a sedmý délku slova. Po rozkódování této struktury následuje funkce `CDC_Params_UART_Init`, která reinicializuje danou UART periferii.

Reinicializace začíná vypnutím DMA a dané UART periferie. Poté se podle rozkódované struktury `LineCoding` nastaví inicializační parametry periferie společně s *receiver timeout* přerušením. Na konci funkce `CDC_Params_UART_Init` proběhne nová inicializace. Zároveň s ní je i znovu spuštěn příjem dat pomocí DMA.

### 6.4.2 Automatická baud rate detekce

Při stavu „*high*“ je započata ABR sekvence. Pro tu byla vybrána implementace pomocí časovače v módu *input capture*. Nejprve jsou vypnuty DMA a UART periferie. Poté je rozsvícena červená LED, která signalizuje probíhající ABR a její zhasnutí je uskutečněno až po úspěšném dokončení detekce.

Následuje samotná inicializace časovače. Na pinu PA3, který je pinem datové linky Rx periferie USART2 je současně vyveden i kanál 4 časovače číslo 2 (*TIM2CH4*). Ten lze inicializovat v módu *input capture*. Celé nastavení je provedeno v softwaru *STM32CubeMx*. Zde se definuje mód, perioda, dělič (*prescaler*), *trigger event* a mnoho dalších parametrů. Dělič určuje konečnou frekvenci časovače. Výchozí frekvence je 72 MHz a měřený baud *rate* bude dosahovat maximální frekvence 1 MHz. Z toho vyplývá, že výchozí frekvence je zbytečně vysoká, proto byl použit dělič s hodnotou 5 (respektive 6 je-li počítáno od jedné). Tím je snížena frekvence na 12 MHz, která by měla být dostačující pro všechny standartní rychlosti. Jako *trigger event* neboli „spouštěcí událost“ byly zvoleny oba typy hran, vzestupný i sestupný.

Po inicializaci je nutné časovač spustit příkazem `HAL_TIM_IC_Start_IT`. Přerušení vyvolané časovačem se nazývá `HAL_TIM_IC_CaptureCallback`. Tuto funkci mikrokontroler vyvolá při každé detekci hrany. Zde je zjištěna aktuální hodnota časovače a je od ní odečtena hodnota předchozí. Výsledek odpovídá době mezi posledními dvěma hranami a je uložen do pole. Toto pole má velikost 64 prvků a po jeho naplnění je časovač ukončen a je přikročeno k výpočtu baud *rate*.

Po naplnění pole je nalezena minimální hodnota z 64prvkového pole, ze které je výsledný baud *rate* spočten následovně:

$$baud = \frac{f \cdot prescaler}{x\_min} \tag{1}$$

Kde $f$ je frekvence mikrokontroleru, tedy 72 MHz a $x\_min$ je nejmenší naměřená doba mezi dvěma hranami.

Tímto je vypočten výsledný baud *rate* a algoritmus přechází k reinicializaci UART periferií. Ostatní parametry jsou ponechány v původním stavu tzn. bez paritního bitu, s jedním stop bitem a délkou slova 8 bitů. Současně je zde i opětovně nastaveno *receiver timeout* přerušení a jsou spuštěny DMA.

Výsledný algoritmus byl následně testován a dosahuje velice dobrých výsledků. Spolehlivě funguje pro rychlosti až do 576 000 baudů za sekundu. Z principu algoritmu vyplývá, že čím nižší rychlost, tím přesnější bude výsledek. Tento předpoklad se potvrdil a nejvyšší odchylka se pohybuje okolo 1 % právě při rychlosti 576 000 baudů za sekundu. Jediným problémem je skutečnost, že algoritmus je opřen o výskyt bitových posloupností, tedy sekvencích bitů 010 nebo 101. Pravděpodobnost výskytu těchto sekvencí zvyšuje přítomnost start a stop bitů, které mají pevnou hodnotu. Avšak vždy je zde šance, že se sekvence během detekce nevyskytla. To lze řešit jedině opětovným otevřením sériových portů a zopakováním detekce.

# 7 Realizace na straně počítače

Tato kapitola se zabývá zpracováním dat přijatých od hardwarového zařízení sériového monitoru. Zpracování zahrnuje přijetí dat, jejich dekódování a zobrazení v čitelné podobě. Ke všem krokům je použit software *Wireshark* poskytující potřebnou modularitu. První část kapitoly se věnuje vytvoření rozhraní pro sériový monitor a druhá se detailněji zaobírá tvorbou deskriptorů testovacího protokolu.

## 7.1 Extcap

*Extcap* neboli *external capture* je název zásuvného modulu umožňující zachytit a analyzovat externí komunikaci. Ačkoliv dokumentace pro *extcap* od vývojářů softwaru *Wireshark* není příliš obsáhlá, nabízí příklad kódu v programovacím jazyce *Python*. [13] Tento skript může být použit jako vzor pro vlastní řešení. Pro spuštění těchto zásuvných modulů *Wireshark* využívá externích překladačů.

Hlavní výhodou při použití rozhraní *extcap* je plná kontrola nad tím, co bude softwaru *Wireshark* přeposláno. *USBPcap* využívá přístup přímo k rozbočovačům. Tento přístup v případě sériového monitoru není potřebný, protože jde především o analýzu datových paketů, které jde zachytit i funkcemi a knihovnami pro sériovou komunikaci. V jazyce *Python* se tato knihovna nazývá *pySerial*.

Tvorbu *extcap* lze rozdělit na dvě části: tvorba uživatelského rozhraní pro nastavení parametrů a rutina pro sběr dat. Jako vzor byl použit již zmiňovaný příklad kódu. [13]

Uživatelským rozhraním je myšleno rozhraní pro nastavování parametrů analýzy. Je vytvořeno knihovnou *argparse*, která bývá často používána pro tvorbu rozhraní v příkazové řádce. Nejprve je vytvořen objekt pomocí *ArgumentParser,* do kterého jsou metodou `add_argument` přidávány jednotlivé prvky. Jejich definice je následně rozšířena ve funkci `extcap_config`, ve které je vytvořen list, popisující vlastnosti jednotlivých prvků. Mezi tyto rozšiřující vlastnosti patří například:

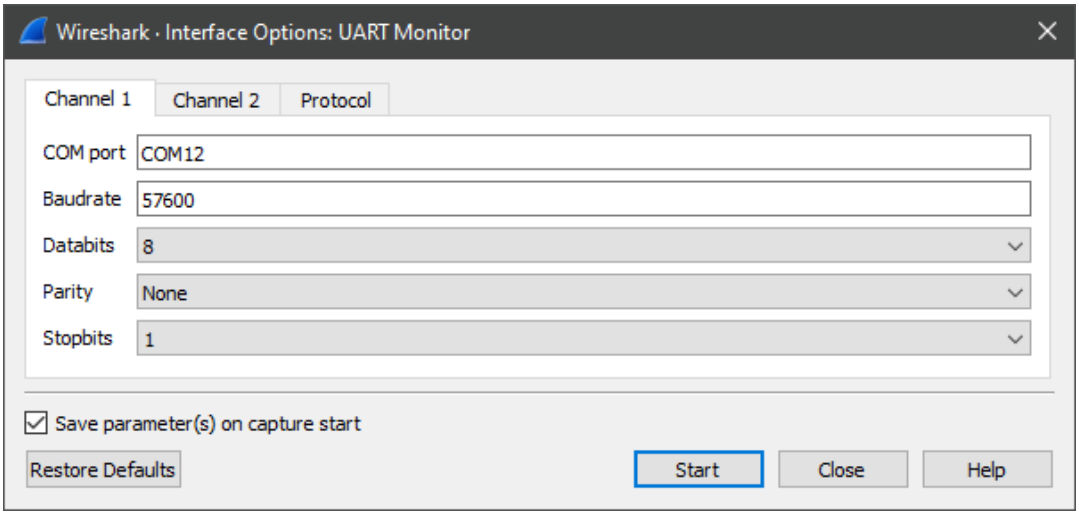
- název, zobrazený v grafickém rozhraní,
- tipy, zobrazené při přejetí myši,
- typ proměnné, například *string, password* nebo *selector* (rolovací nabídka hodnot),
- zařazení do skupiny – *Wireshark* vytvoří dle skupin jednotlivé záložky,
- omezení vstupu,
- výchozí hodnota.

Pomocí těchto dvou částí *Wireshark* vytvoří jednoduché grafické rozhraní, do kterého uživatel zadá všechny parametry.

Jako prvky byly přidány parametry sériového přenosu, mezi něž patří: sériový port, baud *rate*, délka slova, parita a počet stop bitů. Všechny jsou potřeba dvakrát pro oba kanály, které mají pro přehlednost každý svou záložku. U sériového portu je vhodné nastavit výchozí hodnotu, aby uživatel věděl, v jakém tvaru jej psát. Rychlost přenosu může být libovolná, nicméně by měla mít definované hranice. Poslední tři prvky musejí mít předem nadefinované hodnoty, ze kterých si uživatel vybere.

Třetí záložkou je výběr protokolu. Zde si uživatel může vybrat protokol, pomoci kterého bude *Wireshark* dekódovat přijatá data. Ten je přiřazen na základě čísla *DLT* (*Data Link Type*) viz kapitola 7.2.2. Tato záložka by v budoucnu mohla představovat způsob, jak měnit použité dekódovací skripty bez jakéhokoliv hlubšího zásahu uživatele.

Výsledný vzhled rozhraní lze vidět na obrázku č. 15.




**Obrázek č. 15: Vzhled vytvořeného rozhraní v softwaru *Wireshark*.**

Tlačítkem „Start“ je spuštěna funkce `extcap_capture`. V té jsou otevřeny sériové porty se zadanou konfigurací. V případě neúspěchu *Wireshark* přejde do svého hlavního rozhraní, viz obrázek č. 4, avšak bez spuštění funkce pro zachycení dat. Bohužel pro opětovné spuštění analýzy s jiným nastavením je nutné *Wireshark* restartovat. V současné době nejsou podporovány funkce zpětného volání od ovládacích tlačítek a není zde tudíž možnost jak restartovat pouze *extcap*.

Hlavní částí funkce `extcap_capture` je vytvoření třech vláken pomoci knihovny *threading*. Dvě vlákna slouží pro čtení dat z jednotlivých sériových portů. Jejich úkolem je otevřít daný port a poté číst příchozí data. Čtení funguje na obdobné bázi jako přerušení *receiver timeout*. Je sledován počet příchozích bytů a jakmile se tento počet nezmění po určitou dobu je řetězec bytů přečten. Po přečtení jsou data zabalena do bytového pole (*bytearray*) spolu s označením sériového portu, ze kterého přišla a několika dalšími informacemi. Posledním krokem je přidání tohoto pole do řady pomoci knihovny *queue*.

Třetí vlákno slouží k přeposílání dat softwaru *Wireshark*. Jedná se o jednoduché nahlížení do řady řetězců vytvořených z přijatých dat prvních dvou vláken. Po přečtení jednotlivých řetězců je funkce jako paket přeposílá softwaru *Wireshark* k dekódování.

Pro úspěšnou implementaci je nutné vytvořené *extcap* rozhraní vložit do složky *\Wireshark\extcap*. Spolu se skriptem musí být vytvořen a přiložen dávkový soubor s příponou *.bat* obsahující následující vzor:

```
@echo off
<Cesta k překladači skriptu> <Cesta k extcap skriptu> %*
```

Díky tomuto dávkovému souboru *Wireshark* zjistí cestu k překladači a je schopný jej spustit.

## 7.2 Tvorba disektoru

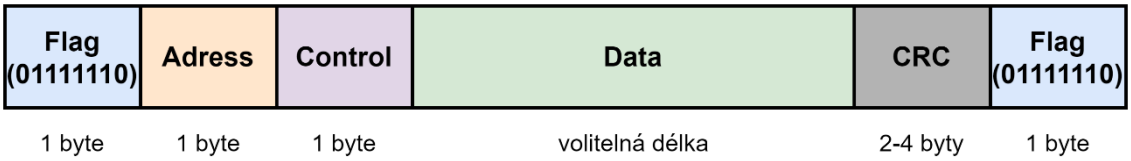
Pomocí vytvořeného rozhraní obdrží *Wireshark* data ze sériového monitoru. Ty jsou v nezpracované podobě a je nutné je dekódovat dle použitého protokolu. K tomu slouží skripty nazývající se disektory. Pro každý protokol musí být vytvořen vlastní disektor. To značně znesnadňuje situaci, nicméně *Wireshark* se snaží tvorbu disektorů co nejvíce zjednodušit. Mocným nástrojem je tvorba v jazyce *Lua*, který je velice intuitivní a dobře dokumentovaný. Podporuje všechny důležité funkce, což bylo otestováno na protokolu skládajícím se z HDLC (*High-Level Data Link Control*) a vyšší informační vrstvy CBOR (*Concise Binary Object Representation*).

### 7.2.1 Testovací protokol

Nižší vrstvu protokolu tvoří HDLC. Jedná se o protokol používaný v linkové vrstvě OSI (*Open Systems Interconnection*), která představuje sedmivrstvý standard komunikace v počítačových sítích vytvořený organizací ISO (*International Organization of Standardization*). [39]

V sériových přenosech se HDLC používá především pro zabalení dat jiných protokolů. Existují tři typy HDLC rámců: I-rámec, S-rámec a U-rámec. I-rámec (*Informative frame*) je hlavním rámcem pro přenášení dat mezi dvěma zařízeními. S-rámec *(Supervisory frame*) se používá pro řídící funkce, kterými jsou potvrzení o přijetí, požadavky o zopakování předchozího rámce atd. U-rámec (*Un-numbered frame*) je rezervovaný pro řízení linky a systému. [39]

Každý začátek a konec rámce je označený tzv. *flag* bytem. Druhý byte je *address* byte obsahující adresu přijímacího zařízení. Třetím bytem je *control* byte, kterým je určen typ rámce a další kontrolní údaje. Tuto hlavičku následuje v případě I-rámců datová část. S-rámec datovou část nemá a U-rámec ji využívá pro upřesnění řídících požadavků. Poslední dva až čtyři byty před koncovým *flag* bytem slouží k detekci chyby pomocí CRC metody. HDLC rámec lze vidět na obrázku č. 16.




**Obrázek č. 16: HDLC rámec.**
(Zdroj: Vlastní zpracování dle: 38)

Datovou část testovacího protokolu tvoří CBOR. Ten je založen na úspěšném modelu JSON (*JavaScript Object Notation*). Jedná se o způsob zápisu dat organizovaný v polích a objektech. Cílem tohoto zápisu je vytvoření datového formátu s jednoznačným kompaktním kódováním nejběžnějších standardů. [40]

CBOR se snaží o co největší kompresi. Této komprese je dosaženo za cenu čitelnosti, kterou disponuje JSON. Jedná se o velice efektivní kódování. Porovnání zápisů stejných objektů do formátu JSON a CBOR lze vidět na blogu společnosti *End Point Corporation*. [41] CBOR dosahuje ve většině příkladů maximálně poloviční délky zápisu v JSON. Současně zde autor uvádí i příklad kódování a dekódování obrázku ve formátu *.jpeg*. Jednalo se o obrázek velký 910 226 bytů a výsledky jsou shrnuty v následující tabulce.

**Tabulka č. 8: Porovnání CBOR a JSON.**
(Zdroj: Vlastní zpracování dle: 41)

| | JSON | CBOR |
|---|---|---|
| Medián kódovacího času [ms] | 3.983 | 0.538 |
| Medián dekódovacího času [ms] | 3.151 | 0.006 |
| Velikost kódu [byte] | 1 213 676 | 910 262 |

Výsledky ukazují, že využití CBOR je mnohonásobně rychlejší, obzvláště dekódování. Zároveň velikost kódu přesněji odpovídá velikosti původního obrázku.

### 7.2.2 Propojení disektoru a rozhraní

Pro psaní disektoru byl zvolen jazyk *Lua*. Ačkoliv je psaní disektorů tímto způsoben poměrně nová záležitost, její dokumentace je velice obsáhlá. Současně s dokumentací je na internetu možné najít spoustu příkladů disektorů. Většinou se jedná o disektory pro síťové komunikace, nicméně jejich princip je velice obdobný.

Hlavním rozdílem je přiřazení disektoru k dané analýze. Síťové disektory se přiřazují k číslu portu komunikace. Kompletní seznam čísel portů pro TCP, UDP a SCTP protokoly lze nalézt na IANA (*Internet Assigned Number Authority*).[43] *Wireshark* při spuštění vybere podle čísla portu daný disektor a používá jej. Při využití *USBPcap* se disektory váží na typ přenosu a *idProduct* číslo. Toto číslo je součástí deskriptoru zařízení. Výhodou tohoto propojení je skutečnost, že uživatel může napsat vlastní disektor pro každý typ přenosu jednoho USB zařízení.

U *extcap* rozhraní uživatel musí sám definovat, který disektor použít. To je uskutečněno přiřazením tzv. DLT (*Data Link Type*) čísla. Toto číslo v knihovně *pcap* definuje hlavičku linkové vrstvy. Seznam všech DLT lze nalézt na stránkách *tcpdump*. [42] Pro uživatelem definovaný typ jsou vyhrazena čísla od 147 do 162. Po přiřazení čísla v *extcap* skriptu stačí v nastavení programu *Wireshark* přiřadit danému DLT číslu disektor. Disektor pro protokol použitý v práci je přiřazen DLT číslu 148. V budoucnu by mohl uživatel v rozhraní *extcap* vybrat, který protokol chce analyzovat a na základě jeho výběru by *extcap* přiřadil protokolu jeho DLT číslo, na které by *Wireshark* navázal daný disektor.

### 7.2.3 Struktura disektoru

Pro psaní disektoru byl použit software *ZeroBrane Studio*. Jedná se o velice jednoduchý vývojářský nástroj. Vzhledem k tomu, že se chyby disektoru odlaďují rovnou ve *Wiresharku*[5], nejsou požadavky na software pro psaní skriptu vysoké. Strukturu disektoru lze rozdělit na dvě části: definice polí disektoru, a rozložení analyzovaných dat.

Každý disektor v jazyce *Lua* musí začínat vytvořením objektu `Proto`, jehož argumenty jsou název protokolu a název zobrazený u analýzy. První argument je název, kterému je přiřazeno DLT číslo.

Dalším krokem je definice a vytvoření objektů pro jednotlivá pole protokolu. K tomu je použit objekt `ProtoField`. Jako příklad je uvedena definice objektu, který zjišťuje typ HDLC rámce:

```
local frametype = ProtoField.uint8 ("hdlc.frametype", "Frame type",
base.DEC, FrameTypeVALS, 0xC0)
```

Jedná se o metodu `.uint8`, protože protokol je dělen po bytech. Prvním argumentem je název pole a druhým je název, který bude zobrazen v analýze. Poslední tři argumenty nejsou povinné. Argument `base` určuje, v jaké číselné soustavě má být pole zobrazeno (v tomto případě v desítkové). `FrameTypeVALS` je tabulka přiřazující na základě hodnoty tohoto pole textový řetězec. Posledním argumentem je tzv. maska. Typ rámce určují první dva bity (v případě I-rámce jen první) *control* bytu, proto je použita maska `0xC0` (binárně 11000000), pomocí které se pole zabývá jen prvními dvěma bity.

[5] *Wireshark* obsahuje integrovaný *Lua* překladač, viz kapitola 5.6.

Tímto způsobem je potřeba nadefinovat a vytvořit objekty pro všechna pole HDLC rámce. V závěru tohoto kroku je nutné všechny vytvořené objekty přidat jako pole do proměnné `fields`, které je součástí objektu pro protokol.

Druhou částí je rozkládání příchozích dat. To je uskutečněno metodou `dissector` definovanou objektem `Proto`. Funkce má tři argumenty: příchozí paket, informace o paketu a strom dělení paketu. Obsahem funkce je v podstatě průchod paketem a jeho rozložení. Nejprve je zjištěno, ze kterého zařízení data přišla. Této informaci odpovídá první byte, který je uměle přidán ve zpracovávací rutině rozhraní *extcap*. Následuje kontrola, zda druhý a poslední byte obsahují *flag* byte. Jestliže tomu tak není, jedná se o neúplný rámec, viz kapitola 7.2.4.

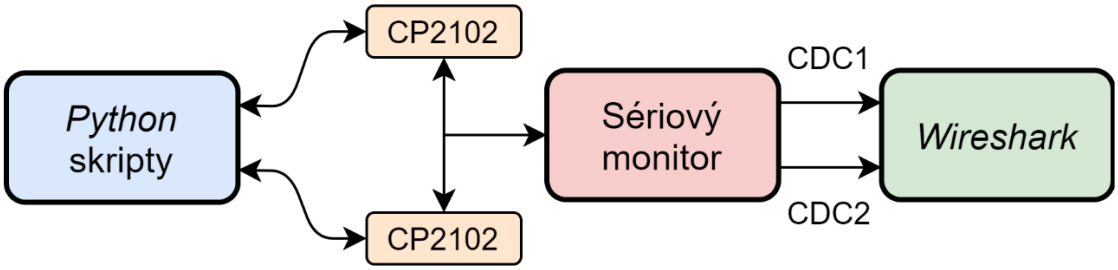
V opačném případě následuje přidání stromu a názvu protokolu do sloupce „Protocol“. Dále je přeložen adresní byte, CRC pole a je zjištěno o jaký typ rámce se jedná. Podle typu rámce je přeložen řídící (*control*) byte. Jestliže se jedná o I-rámec je vytvořen nový paket, který se skládá pouze z datové části informačního rámce. Pro tento nový paket je zavolán disektor pro CBOR, jenž je součástí softwaru *Wireshark*. V praxi se takový způsob nazývá *chained dissectors* neboli řetězení disektorů. Tato sekvence vypadá následovně:

```
if information_len > 0 then
        local new_buffer = buffer(offset, information_len)
        local new_dissector = Dissector.get("cbor")
        new_dissector:call(new_buffer:tvb(), pinfo, tree)
end
```

Kde `information_len` je délka datové části a `offset` je současná pozice v paketu. Díky těmto dvěma proměnným je vytvořen nový paket, pro který je zavolán již existující disektor.

### 7.2.4 Testování disektoru

Pro testování funkčnosti disektoru jsou použity skripty v jazyce *Python*, které mají za úkol simulovat komunikaci pomocí protokolů HDLC a CBOR. Oba skripty posílají data přes propojené převodníky USB/UART s čipem CP2102. Na jejich komunikaci je připojeno hardwarové zařízení sériového monitoru, které posílá data zpět do počítače, kde je pro zpracování dat připraven software *Wireshark* s *extcap* rozhraním a disektorem. Schéma testování lze vidět na obrázku č. 17.




**Obrázek č. 17: Schéma testování disektoru.**

Komunikaci obstarávají dva skripty, jeden funguje jako vysílač a druhý jako přijímač. Vysílač posílá HDLC I-rámce s datovou částí obsahující protokol CBOR, který popisuje pole o čtyřech elementech. Přijímač na tuto zprávu vždy odpovídá rámcem bez datové části. Řídící byte přijímače má hodnotu vždy o 32 bitů vyšší než jeho předchůdce. Tímto je dosaženo otestování funkčnosti disekce řídícího bytu. Po přijetí zprávy od přijímače se celá sekvence opakuje. Detail paketu vysílače a vybraného paketu přijímače lze vidět na obrázku č. 18, respektive 19.

```
DLT: 148, Payload: cbor_over_hdlc (UART Serial Monitor - STM32 - CBOR over HDLC)
▾ HDLC Frame
    Source/Destination address: 0xff
    00.. .... = Frame type: I-Frame (0)
  ▾ I-Frame
      .... .010 = Receive sequence no.: 2
      .... 0... = Poll/Final flag: 0
      .001 .... = Send sequence no.: 1
    Message length: 64
    Frame Checksum: 51758
▾ Concise Binary Object Representation
  ▾ Array: (4 elements)
      Text String: SPACE_C_H_operation_selection
      Text String: Heating
    ▾ Array: (2 elements)
        Text String: stringArray
        Text String: rw
      Text String: cooling
```

**Obrázek č. 18: Detail paketu vysílače testovacího protokolu.**

```
DLT: 148, Payload: cbor_over_hdlc (UART Serial Monitor - STM32 - CBOR over HDLC)
▾ HDLC Frame
    Source/Destination address: 0xff
    01.. .... = Frame type: I-Frame (1)
  ▾ I-Frame
      .... .001 = Receive sequence no.: 1
      .... 0... = Poll/Final flag: 0
      .100 .... = Send sequence no.: 4
    Message length: 0
    Frame Checksum: 2723
```

**Obrázek č. 19: Detail vybraného paketu přijímače testovacího protokolu.**

Z obrázku je patrné, že disektor funguje přesně jak je požadováno. Všechny příchozí pakety jsou správně rozsegmentovány. Z těchto důvodů je disektor prohlášen za validní.

Integrovaný disektor pro protokol CBOR přesně splňuje svůj účel. Toto zjištění má do budoucna velký potenciál, protože při využití integrovaných disektorů se značně zjednodušuje tvorba nových disektorů. HDLC rámce můžou přenášet libovolný protokol a struktura celého disektoru s výjimkou jednoho řádku zůstane neměnná.

### 7.2.5 Rekonstrukce rozdělených rámců

Kapitola 7.2.4 popisuje disekci protokolu za předpokladu, že přijatá data obsahují celý rámec. Disektor nicméně neřeší situaci, kdy je rámec rozdělený do více datových paketů[6]. Při analýze rozděleného rámce by disektor nerozsegmentoval ani jednu část, protože ani jedna část neodpovídá prvotní podmínce, že druhý a poslední byte jsou *flag* byty. Proto je nutné nějak podchytit tuto problematiku a rozšířit stávající disektor.

Software *Wireshark* nabízí rekonstrukci síťových protokolů s transportní vrstvou TCP (*Transmission Control Protocol*). U integrovaných disektorů stačí pouze povolit tuto funkci v nastavení. U vlastních disektorů pro tyto protokoly lze přes objekt `pinfo` (*packet information*) přistoupit do parametru `desegment_len` a `desegment_offset`. První zmiňovaný určuje zbývající počet bytů do kompletního paketu a druhým parametrem se určuje pozice v současném paketu, od kterého bude disektor při příštím zavolání pokračovat.

Zatímco TCP protokoly fungují v podstatě jako proud dat, USB sběrnice je založena na oddělených paketech. Z tohoto důvodu tato funkce není implementována. Bohužel stejná situace je i u externích přenosů. Jediným způsobem, jak provést rekonstrukci rozdělených rámců je použití manuálně vytvořené proměnné, která bude sledovat data. Do té se bude ukládat obsah příchozích dat v případě, že nebude splněna podmínka pro úplný rámec.

Tuto proměnnou je nutné definovat mimo metodu disektoru, k čemuž je využita metoda `init` objektu označujícím protokol. Tato proměnná se inicializuje pouze při spuštění disektoru. Proměnná nese název `partialBuffer`. Díky tomuto kroku se řešení rekonstrukce rozpadá do čtyř možností:

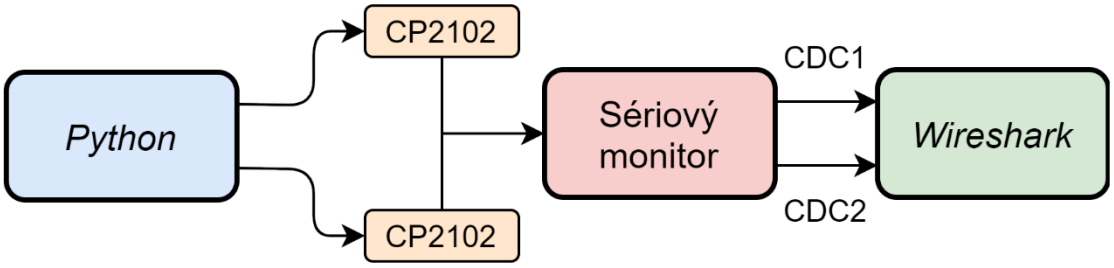
- druhý i poslední byte jsou *flag* – disekce probíhá normálně, `partialBuffer` není využit,
- druhý byte je *flag*, poslední není – datový paket je uložen do proměnné `partialBuffer` a paket je označen jako „*Partial Frame“*,
- druhý ani poslední byte nejsou *flag* – datový paket je přidán funkcí `append` do proměnné `partialBuffer` a paket je označen jako „*Partial Frame“*,
- druhý byte není *flag*, poslední je – datový paket je přidán funkcí `append` do proměnné `partialBuffer`, paket je označen jako „*Reassembled*“ a je přistoupeno k jeho disekci.

Touto algoritmizací je vyřešena rekonstrukce rozdělených rámců. Jedná se o značně experimentální metodu, protože tato problematika není dobře dokumentovaná. Jediným problémem rekonstrukce je skutečnost, že *Wireshark* při rozkliknutí jednotlivých paketů opět použije disektor. Z toho důvodu je nutné pro prohlídnutí rekonstruovaného paketu nejdříve prohlídnout všechny jeho části.

[6] Nejedná se o skutečný paket, ale o terminologii softwaru *Wireshark*.

# 8 Vyhodnocení propustnosti

Tato kapitola se zabývá testováním propustnosti sériového monitoru na základě rychlosti analyzovaných komunikací a zaplnění jejich datových linek. Zaplněnost je určena pomocí frekvence posílání zpráv a délky posílaných řetězců. Pro testování bylo použito schéma, které lze vidět na obrázku č. 18.




**Obrázek č. 18: Schéma testování spolehlivosti sériového monitoru**

Schéma je obdobné jako v případě testování disektoru. Rozdílem je, že čipy CP2102 nejsou propojeny. Sériový monitor je připojen na linku Rx obou čipů a přeposílá data zpět do počítače, kde je obsah kontrolován v softwaru *Wireshark*.

Pro experimenty byly vybrány následující baud *rate*: 9 600, 38 400, 115 200, 256 000, 460 800 a 921 600. Skutečná rychlost, tedy kolik baudů bylo ve skutečnosti posláno za jednu vteřinu se vypočítá jako:

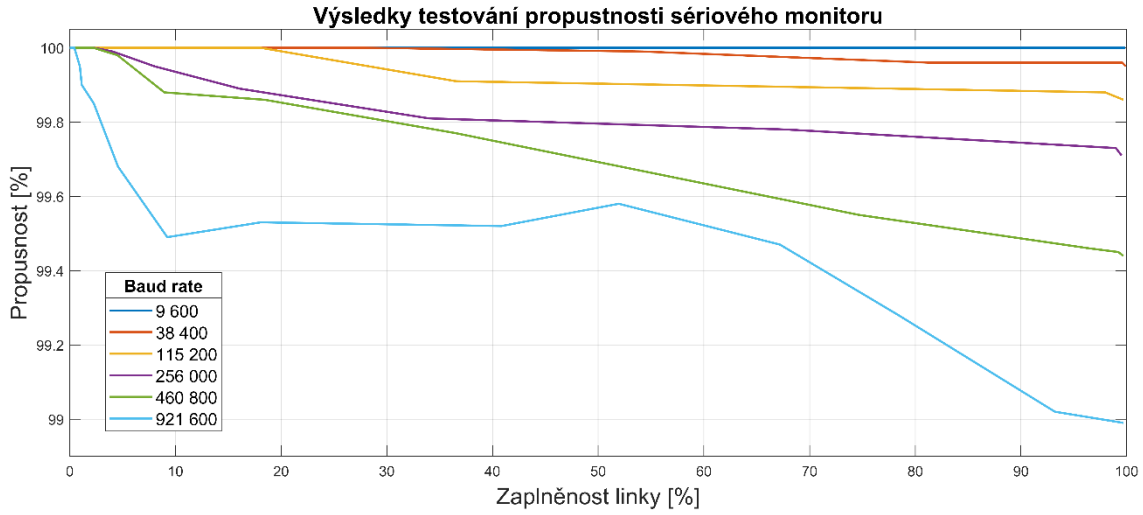
$$baud_{real} = \frac{n \cdot l \cdot (B + startbit + stopbit)}{t}. \qquad (2)$$

Kde
- *n* je počet poslaných řetězců,
- *l* je délka řetězce,
- *B* je počet bitů jednoho bytu, tedy osm,
- *t* je doba transakce.

Baud *rate* je definován jako možný počet změn signálu za jednu sekundu, tudíž je nutné ke každému bytu přičíst start a stop bit. Zaplněnost je určena jako poměr skutečného počtu baudů za sekundu ku použitému baud *rate*. Výsledná zaplněnost je uváděna v procentech.

Propustnost je vypočtena jako poměr doručených k počtu odeslaných znaků. Tento údaj je opět udáván v procentech. Zprávy byly kontrolovány v softwaru *Wireshark*. Pro potřeby vyhodnocení byl vytvořen jednoduchý disektor, který pouze zobrazoval obsah paketu ve formátu ASCII a postupně sčítal počet znaků v každém paketu.

Na obrázku č. 19 lze vidět výsledný graf vyhodnocení. Osa x představuje zaplněnost datových linek v procentech, zatímco osa y zobrazuje propustnost sériového monitoru. Pro každou rychlost přenosu je vynesena křivka spojující naměřené hodnoty.




**Obrázek č. 19: Vyhodnocení propustnosti sériového monitoru**

Z grafu lze vyčíst, že se zvyšující se rychlostí se snižuje propustnost sériového monitoru. Jediná rychlost, která je schopná udržet hodnotu je 9 600 baud/s. Při použití vyšších rychlostí propustnost od určitého zaplnění linky lineárně klesá. Příčina tohoto chování je s velkou pravděpodobností ve vytíženosti mikroprocesoru, který musí zároveň obsluhovat USB sběrnice a přijímat nová data. Až na nejvyšší použitou rychlost se propustnost pohybovala v rozmezí 99.5 až 100 %. Chování nejvyšší rychlosti 921 600 baud/s bylo značně nepředvídatelné, avšak po rychlém poklesu se propustnost začala ustalovat. Nejnižší naměřená hodnota při použití této rychlosti byla 98.99 %.

Pro ověření správné funkčnosti byl následně proveden ještě jeden experiment, a to pouze s jedním čipem CP2102. Výsledkem tohoto experimentu nebyla žádná ztráta dat a stoprocentní propustnost sériového monitoru u všech rychlostí. Tímto byla vyloučena nesprávná funkčnost UART a USB periferie. Jako příčina neúplné propustnosti při použití obou UART periferií byla označena vytíženost mikroprocesoru.

# 9 Závěr

Cílem této diplomové práce bylo navrhnout a realizovat sériový monitor pro rozhraní UART. V rešeršní části byly prozkoumány možnosti analýzy sériových komunikací a popsány jednotlivé teoretické prvky nutné pro realizaci zařízení. Na základě této části byly upřesněny požadavky a byla vypracována analýza možných řešení.

Samotnou realizaci lze rozdělit na dvě části. První část je věnována hardwarovému zařízení. Pro realizaci byla vybrána vývojová sada *NUCLEO-F303ZE* od firmy *STMicroelectronics*. Ta odpovídala všem požadavkům a zároveň se jednalo i o dostupnou variantu. Pro tuto sadu byl vytvořen firmware, který se stará o sběr dat z UART periferií a jejich distribuci pomocí sběrnice USB. Sběr je založen na přímém přístupu do paměti (*DMA – Direct Memory Access*). Pro distribuci bylo vytvořeno kompozitní USB zařízení skládající se ze dvou tříd CDC (*Communication Device Class*). Nastavování parametrů UART periferie je uskutečněno buďto přenášením parametrů virtuálních komunikačních portů z počítače za pomoci řídících přenosů při otevírání těchto portů nebo vytvořenou automatickou baud *rate* detekcí. Volba metody je provedena *DIP* přepínačem připojeným k vývojové sadě.

Druhá část realizace je věnována propojení hardwarového zařízení se softwarem *Wireshark* a zobrazení dat v čitelné podobě. Pro propojení bylo vytvořeno rozhraní *extcap* v programovací jazyce *Python*. Toto rozhraní umožňuje otevření virtuálních komunikačních portů přímo v softwaru *Wireshark* a zároveň se stará i o distribuci dat z hardwarového zařízení. Dále byl vytvořen dekódovací skript v jazyce *Lua*. Ten má za úkol zobrazení příchozích dat v čitelné podobě. Tento skript byl vytvořen pro testovací protokol skládající se z HDLC a vyšší informační vrstvy CBOR. Součástí skriptu byla i rekonstrukce rozdělených rámců. Dekódování bylo následně úspěšně testováno.

Poslední část práce byla věnována vyhodnocení propustnosti hardwarového zařízení. Propustnost byla testována na základě zaplněnosti obou datových linek pro různé rychlosti komunikace. Výsledky ukázaly, že propustnost pro běžné rychlosti do 115 200 baud/s neklesne pod 99,8 %, a to i při plném zaplnění obou datových linek. Pro ověření, že se nejedná o chybu UART periferií či USB sběrnice byla otestována propustnost pouze za použití jedné datové linky. Výsledkem byla stoprocentní propustnost. Z těchto důvodů byla za příčinu neúplné propustnosti označena vytíženost mikroprocesoru.

Zvolená realizace zachovala univerzálnost zařízení, které je díky tomu možné v budoucnosti rozšířit o další funkce. Těmi by mohlo být například rozšíření o další sběrnice nebo přidání módu pro injekci analyzovaných systémů. USB sběrnice hardwarového zařízení by mohla být rozšířena o třetí rozhraní, které by sloužilo ke konfiguraci módů a parametrů komunikace.

Výsledkem práce je kompletní realizace sériového monitoru včetně jeho propojení s počítačem, všechny cíle diplomové práce byly splněny.

# Seznam zkratek

| | |
|---|---|
| **ABR/ABRD** | Automatic Baud Rate Detection – automatická baud *rate* detekce |
| **ACM** | Abstract Control Model |
| **API** | Application Programming Interface |
| **ASCII** | American Standard Code for Information Interchange |
| **BCD** | Binary Coded Decimal – dvojkově reprezentované dekadické číslo |
| **BLE** | Bluetooth Low Energy |
| **CBOR** | Concise Binary Object Representation |
| **CDC** | Communication Device Class |
| **COM** | Communication Port – komunikační port |
| **CRC** | Cyclic Redundancy Check – cyklický redundantní součet |
| **DIP** | Dual In-line Package |
| **DLL** | Dynamic-Link Library |
| **DLT** | Data Link Type |
| **DMA** | Direct Memory Access |
| **EOP** | End of packet – symbol konce paketu |
| **EP0** | Endpoint 0 – nultý koncový bod |
| **EXTI** | External Interrupt – externí přerušení |
| **GPIO** | General-Purpose Input/Output |
| **GUI** | Graphical User Interface – grafické uživatelské rozhraní |
| **HAL** | Hardware Abstraction Layer/Hardware Annotation Library |
| **HDLC** | High-level Data Link Control |
| **HID** | Human Interface Device |
| **IANA** | Internet Assigned Number Authority |
| **IDE** | Integrated Development Enviroment |
| **IoT** | Internet of Things |
| **IP** | Internet Protocol |
| **IRP** | Input/Output Request Packet |
| **IRQ** | Interrupt ReQuest |
| **ISO** | International Organization of Standardization |
| **ISR** | Interrupt Service Routine |
| **JSON** | JavaScript Object Notation |
| **LCD** | Liquid Crystal Display |
| **LED** | Light-Emitting Diode |

| | |
|---|---|
| **LSB** | Least Significant Bit – nejméně významný bit |
| **NRZI** | Non Return To Zero Inverted |
| **OSI** | Open Systems Interconnection |
| **OTG** | On-The-Go |
| **PCD** | Peripheral Control Driver |
| **PID** | Packet ID |
| **PMA** | Packet Memory Access |
| **RTOR** | Receiver TimeOut Register |
| **SCTP** | Stream Control Transmission Protocol |
| **SOF** | Start of frame – paket začátku rámce |
| **SRAM** | Static Random Access Memory – statická paměť |
| **SSH** | Secure Shell |
| **TCP** | Transmission Control Protocol |
| **UART** | Universal Asynchronous Receiver/Transmitter |
| **UDP** | User Datagram Protocol |
| **URB** | USB Request Block |
| **USART** | Universal Synchronous/Asynchronous Receiver/Transmitter |
| **USB** | Universal Serial Bus |
| **USB-IF** | USB Implementers Forum |
| **VCP** | Virtual Communication Port |
| **WSGD** | Wireshark Generic Dissector |

# Zdroje


[1] VLČEK, Jiří. Přenos dat. In: *TZB-info* [online]. [cit. 2020-04-08]. Dostupné z: https://www.tzb-info.cz/docu/texty/0001/000102_mereni_a_sber_dat_pomoci_pc.pdf

[2] Komunikace po sběrnici. *Mendelova univerzita v Brně* [online]. [cit. 2020-04-08]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=9947

[3] PETERKA, Jiří. Synchronní, asynchronní a arytmický přenos. In: e*Archiv: Archiv článků a přednášek Jiřího Peterky* [online]. 10. prosince 1996 [cit. 2020-04-08]. Dostupné z: http://www.earchiv.cz/a96/a650k150.php3

[4] PETERKA, Jiří. Druhy přenosu – II. In: *eArchiv: Archiv článků a přednášek Jiřího Peterky* [online]. 3. listopadu 1998 [cit. 2020-04-08].
Dostupné z: https://www.earchiv.cz/a98/a845k180.php3

[5] Serial Communication. *SparkFun Electronics* [online]. [cit. 2020-04-08]. Dostupné z: https://learn.sparkfun.com/tutorials/serial-communication

[6] GAJDOŠ, M. Univerzální analyzátor sériových sběrnic. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009. 85 s. Vedoucí diplomové práce Ing. Petr Sysel, Ph.D.

[7] *Logic Analyzers from Saleae* [online]. [cit. 2020-04-08].
Dostupné z: https://www.saleae.com/

[8] *Qingdao Hantek Electronic Co., Ltd.* [online]. [cit. 2020-04-08].
Dostupné z: http://www.hantek.com/en/productdetail_84.html

[9] Microsoft Message Analyzer Blog. *Technická dokumentace, rozhraní API a příklady kódování* [online]. [cit. 2020-04-08].
Dostupné z: https://docs.microsoft.com/en-us/openspecs/blog/ms-inintbloglp/f

[10] *Wireshark Go Deep.* [online]. [cit. 2020-04-08].
Dostupné z: https://www.wireshark.org/

[11] *GitHub-wireshark/wireshark: Read only mirror.* The world's leading software development platform-GitHub [online]. [cit. 2020-04-08].
Dostupné z: https://github.com/wireshark/wireshark

[12] *TPCDUMP/LIBCAP public repository* [online]. [cit. 2020-04-08].
Dostupné z: https://www.tcpdump.org/

[13] *WinDump-Home* [online]. [cit. 2020-04-08]. Dostupné z: https://www.winpcap.org/

[14] AXELSON, Jan. *USB complete: the developer's guide*. Fifth edition. Madison: Lakeview Research, [2015]. ISBN 978-1931448284.

[15] USB-IF USB4 specifications annoucment. *USB-IF* [online]. Beaverton, OR, USA, 3. září 2019 [cit. 2020-04-08].
Dostupné z: https://www.usb.org/sites/default/files/2019-09/USB-IF_USB4_spec_announcement_FINAL.pdf

[16] ANDERSON, Don, Dave DZATKO, Mindshare a Inc. *Universal Serial Bus System Architecture*. 2. University of Michigan: Addison-Wesley, 2001. ISBN 9780201309751.

[17] Wireshark-Documentation. *Wireshark Go Deep.* [online]. [cit. 2020-04-08]. Dostupné z: https://www.wireshark.org/docs/

[18] USB hardware - Wikipedia. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2020-04-08].
Dostupné z: https://en.wikipedia.org/wiki/USB_hardware

[19] USB in a NutShell. *Beyondlogic-Electronics Hardware-Embedded Linux-Internet of Things* [online]. [cit. 2020-04-08].
Dostupné z: https://www.beyondlogic.org/usbnutshell/usb1.shtml

[20] A Transfer Type for Every Purpose. *Proyectosfie.tk* [online]. [cit. 2020-04-08]. Dostupné z: https://proyectosfie.webcindario.com/usb/libro/capitulo03.pdf

[21] PERNÝ, J. Zařízení pro monitorování sériové komunikace. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. 55 s. Vedoucí diplomové práce doc. Ing. Jaromír Kolouch, CSc

[22] Defined Class Codes |USB-IF. *USB-IF* [online]. [cit. 2020-04-08].
Dostupné z: https://www.usb.org/defined-class-codes

[23] USB Communication Device Class (CDC) Decoder. *USBLyzer-USB Protocol Analyzer and USB Traffic sniffer for Windows* [online]. [cit. 2020-04-08]. Dostupné z: http://www.usblyzer.com/usb-communication-device-class-cdc-decoder.htm

[24] Universal Serial Bus Class Definitions for Communication Devices. *Dr. C. Scott Ananian* [online]. 1999 [cit. 2020-04-08].
Dostupné z: https://cscott.net/usb_dev/ data/devclass/usbcdc11.pdf

[25] USB Enumeration-Developer Help. *Developer-Help* [online]. [cit. 2020-04-08]. Dostupné z: https://microchipdeveloper.com/usb:enumeration

[26] *STMicroelectronics* [online]. [cit. 2020-04-08]. Dostupné z: https://www.st.com/

[27] NUCLEO-F303ZE-STMICROELECTRONICS. *Farnell Česká republika-Distributor elektronických součástek* [online]. [cit. 2020-04-08].

Dostupné z: https://cz.farnell.com/stmicroelectronics/nucleo-f303ze/dev-board-arduino-mbed-nucleo/dp/2517895

[28] STM32F303xD STM32F303xE. *STMicroelectronics* [online]. 2016 [cit. 2020-04-08]. Dostupné z: https://www.st.com/resource/en/datasheet/stm32f303ze.pdf

[29] AN4908 Application note. *STMicroelectronics* [online]. 2016 [cit. 2020-04-08]. Dostupné z: https://www.st.com/resource/en/application_note/dm00327191-stm32-usart-automatic-baud-rate-detection-stmicroelectronics.pdf

[30] USBPcap. *Desowin's mainsite* [online]. [cit. 2020-04-08].
Dostupné z: https://desowin.org/usbpcap/

[31] Wireshark/extcap_example.py. *The world's leading software development platform-GitHub* [online]. [cit. 2020-04-08]. Dostupné z: https://github.com/wireshark/wireshark/blob/master/doc/extcap_example.py

[32] NRF Sniffer User Guide v2.2. *Nordic Semiconductor* [online]. [cit. 2020-04-08]. Dostupné z: https://infocenter.nordicsemi.com/pdf/nRF_Sniffer_UG_v2.2.pdf

[33] GitHub-NordicSemiconductor/nRF_Sniffer_802154. *The world's leading software development platform-GitHub* [online]. [cit. 2020-04-08].
Dostupné z: https://github.com/NordicSemiconductor/nRF-Sniffer-for-802.15.4/tree/master/nrf802154_sniffer

[34] *Wireshark Generic Dissector* [online]. [cit. 2020-04-08].
Dostupné z: http://wsgd.free.fr/index.html

[35] *SharkFest* [online]. [cit. 2020-04-08].
Dostupné z: https://sharkfestus.wireshark.org/

[36] BLOICE, Graham. SF19US - 03 Writing a Wireshark Dissector: 3 ways to eat bytes. In: *Youtube* [online]. 12. června 2019 [cit. 2020-04-08].
Dostupné z: https://www.youtube.com/watch?v=Fp_7g5as1VY

[37] RM0316 Reference manual. *STMicroelectronics* [online]. [cit. 2020-04-08]. Dostupné z: https://www.st.com/resource/en/reference_manual/dm00043574-stm32f303xbcde-stm32f303x68-stm32f328x8-stm32f358xc-stm32f398xe-advanced-armbased-mcus-stmicroelectronics.pdf

[38] *High-Level Data Link Control (HDLC)* [online]. [cit. 2020-04-08].
Dostupné z: https://www.tutorialspoint.com/high-level-data-link-control-hdlc

[39] High Level Data Link Control (HDLC). *Electronics Research Group-University of Aberdeen* [online]. [cit. 2020-04-08].
Dostupné z: https://erg.abdn.ac.uk/users/gorry/course/dl-pages/hdlc.html

[40] *RFC 7049-Concise Binary Object Representation (CBOR)* [online]. [cit. 2020-04-08]. Dostupné z: https://tools.ietf.org/html/rfc7049

[41] VOLLRATH, Matt. Extensible Binary Encoding with CBOR. In: *Secure Bussines Solutions* | *End Point* [online]. 18. března 2019 [cit. 2020-04-08]. Dostupné z: https://www.endpoint.com/blog/2019/03/18/extensible-binary-encoding-with-cbor

[42] Service Name and Transport Protocol Port Number Registry. *Internet Assigned Numbers Authority* [online]. [cit. 2020-04-08]. Dostupné z: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

[43] LINK-LAYER HEADER TYPES. *TCPDUMP/LIBPCAP public repository* [online]. [cit. 2020-04-08]. Dostupné z: https://www.tcpdump.org/linktypes.html

# Seznam příloh

## A Elektronické přílohy

Přiložený .zip soubor obsahuje:

- /serial_monitor_firmware – zdrojové soubory pro hardwarové zařízení sériového monitoru,
- /extcap – rozhraní *extcap* pro propojení hardwarového zařízení se softwarem *Wireshark,*
- /disektor – dekódovací skript pro testovací protokol použitý v práci.