FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Aplikace pro analýzu průmyslových sběrnic včetně hardwarového trigeru
Student:	Matúš Olekšák
Vedoucí:	Ing. Pavel Kubalík, Ph.D.
Studijní program:	Informatika
Studijní obor:	Počítačové inženýrství
Katedra:	Katedra číslicového návrhu
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

1) Prozkoumejte existující řešení.
2) Navrhněte a zrealizujte vlastní aplikaci pro analýzu a zpracování dat z průmyslových sběrnic.
3) Systém musí umožnit zachycení komunikace přes CAN sběrnici v režimu CAN FD, sériové linky, HTTP streamu a veškerou komunikaci přes Ethernet.
4) Aplikace musí být ovladatelná přes webové API.
5) Aplikace musí být schopná konfigurovat jednotlivé platformy, které budou obsahovat uživatelem zvolené sběrnice.
6) Pro synchronizaci a vytváření značek navrhněte a zrealizujte jednoduchý hardware, umožňující spouštět analýzu a vytvářet značky na základě stisknutého tlačítka.
7) Výsledné řešení otestujte.

Seznam odborné literatury

Dodá vedoucí práce.

doc. Ing. Hana Kubátová, CSc.
vedoucí katedry

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

V Praze dne 6. února 2020

ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F8

Fakulta informačních technologií
Katedra číslicového návrhu

Bakalářská práce

Aplikace pro analýzu průmyslových sběrnic včetně hardwarového trigeru

Matúš Olekšák
Počítačové inženýrství

Květen 2020
Vedoucí práce: Ing. Pavel Kubalík, Ph.D.

Poděkování / Prohlášení

Chtěl bych poděkovat vedoucímu práce Ing. Pavlovi Kubalíkovi, Ph.D. za rady při tvorbě této bakalářské práce. Dále společnosti Digiteq Automotive s.r.o. za poskytnutí tématu a potřebného vybavení. A nakonec mé rodině za podporu při studiu.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 1. 6. 2020

.....

Abstrakt / Abstract

Tato práce se zabývá návrhem a implementací nástroje pro záznam dat ze sběrnic používaných v automobilech. Během vývoje řešení byl kladen důraz na univerzálnost a škálovatelnost.

Výsledkem je nástroj pro jednotný způsob záznamu sběrnic a hardwarový trigger pro vytváření časových značek. Podporuje sběrnice typu CAN FD, RS232, Ethernet a HTTP stream.

Přínosem této práce je analýza dílčích řešení pro záznam jednotlivých sběrnic, ale také lehce rozšiřitelný výsledný framework, který lze využít v automatizovaných testech.

Klíčová slova: zachytávání komunikace po sběrnici, jednotný způsob záznamu sběrnic, automatizované testy, CAN FD, Ethernet, 1000BASE-T1, RS232, HTTP stream, SocketCAN, Wireshark

This work deals with the design and implementation of a tool for data recording from buses used in automobiles. During the development of the solution, emphasis was placed on versatility and scalability.

The result is a tool for unified way to record buses and a hardware trigger to create timestamps. It supports bus type CAN FD, RS232, Ethernet and HTTP stream.

The contribution of this work is the analysis of partial solutions for recording individual buses, but also an easily extensible resulting framework, which can be used in automated tests.

Keywords: capturing of bus communication, unified way to record buses, automated tests, CAN FD, Ethernet, 1000BASE-T1, RS232, HTTP stream, SocketCAN, Wireshark

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

V automobilech je čím dál tím více elektronických zařízení, která potřebují mezi sebou komunikovat. Zvyšuje se nejen jejich počet, ale také množství dat, která je potřeba mezi nimi přenést. Začalo to standardizovanou sběrnicí CAN, pak přišla její evoluce ve formě CAN FD, která zvyšovala přenosové rychlosti a množství přenesených dat. Jenže ani to nestačilo pokrýt všechny požadavky, a proto aktuálně vyvíjené automobily obsahují také gigabitový Ethernet.

Neoddělitelnou součástí vývoje je testování, a proto také testovací část musí být připravena na zachytávání a analýzu všech používaných sběrnic. Pro účely testování jednotlivých komponent nebo dokonce celé architektury vozidla by bylo ideální mít jedno zařízení, které by zvládalo zachytávání všech použitých sběrnic.

Hlavní motivací, proč jsem si toto téma vybral, je nedostupnost škálovatelných zařízení určených pro zachytávání a analýzu aktuálně využívaných sběrnic. Všechna jsou dostupná jen jako hotová řešení s předem daným počtem kanálů. To způsobuje, že buď si koupíte zařízení, které obsahuje zbytečně mnoho kanálů, a zaplatili jste za něj pak více, než kolik jste potřebovali. Nebo naopak jedno zařízení nestačí, a musíte zakoupit více takových zařízení, co už není finančně výhodné.

Výsledek této práce je určen do testovacích center, která testují prototypy. Nepůjde o jedno konkretní zařízení, ale o nástroj pro sjednocený přístup k různým sběrnicím. S příchodem nové sběrnice postačí rozšířit zařízení o přídavnou kartu nebo převodník a dopsat implementaci všeobecných příkazů pro danou sběrnici.

Obsahem této práce je v úvodu analýza existujících řešení, kde porovnávám podobné produkty od renomovaných firem. Následuje pak analýza jednotlivých stavebních častí, jak po stránce hardwaru, tak i softwaru. Po analyzování přichází na řadu návrh řešení, kde jsem zvolil pro moje řešení, podle mně, nejvíc vhodné stavební prvky.

Realizace je pak rozdělena na softwarovou část a hardwarový trigger. Softwarová část je tvořená implementací frameworku, který je pak použit při realizaci nástroje splňujícího zadání. Část zaobírající se hardwarovým triggerem pak popisuje jeho realizaci od návrhu DPS, přes vyvíjení firmwaru, až po tisk krabičky na 3D tiskárně.

Když je realizace dokončená, tak se celé řešení otestuje. Ze začátku se otestuje po částech každá sběrnice samostatně, a pak se testují celé platformy s hardwarovým triggerem.

Kapitola 2
Cíle

Cílem této práce je navrhnout zařízení schopné zachytávat komunikaci přes CAN sběrnici v režimu CAN FD, sériové linky, HTTP streamu a veškerou komunikaci přes Ethernet. Pro účely automatizovaného testování, kam především toto zařízení míří, musí být ovladatelné přes webové API. Pro zvýšení uživatelského komfortu při testování, aplikace musí být schopna konfigurovat jednotlivé platformy, které budou obsahovat uživatelem zvolené sběrnice a jejich nastavení. Součástí zařízení bude hardwarový trigger, umožnující spouštět analýzu a vytvářet značky na základě stisknutého tlačítka.

Při implementaci bude kladen důraz na univerzálnost a škálovatelnost. Tím se bude odlišovat od existujících řešení. Uživatel si bude moct podle svých potřeb nakonfigurovat konečnou podobu zařízení. Při návrhu bude bráno v potaz také zvýšení nároků na počet jednotlivých kanálů v budoucnu. Samozřejmostí bude jednoduché rozšíření o nové sběrnice.

Navržené cíle budou rozděleny do jednotlivých dílčích etap. První a nejdůležitější etapou je návrh rozhraní, které budou využívat jednotlivé sběrnice a jejich unifikaci. Dále bude následovat navrhnutí rozhraní pro webové API a jeho následná implementace. Na konci bude implementace podpory platforem.

Další cíl bude ověření návrhu hardwarového triggeru na nepájivém poli. Po ověření funkčnosti bude vytvořen návrh DPS, s její následnou výrobou a osazením potřebnými komponenty. Na závěr se navrhne plastová krabička jako 3D model a vytiskne se na 3D tiskárně.

Kapitola 3
Existující řešení

Prvním z cílů zadání bylo prozkoumat existující řešení. Proto jsem hledal podobná All-In-One zařízení na zaznamenávání vícerých typů sběrnic. Hlavním požadavkem byla podpora Ethernetu 1000BASE-T1, ale vyžadován byl také CAN FD.

3.1 MAGNA Telemotive BLUEPIRAT Rapid

Společnost MAGNA Telemotive nabízí zařízení BLUEPIRAT Rapid. Jedná se o All-In-One řešení podporující 20 Ethernetových kanálů standardu 100BASE-T1 a 4 přídavné kanály standardu 1000BASE-T. Pro standard 1000BASE-T1 potřebujeme dokoupit převodník. Dále zde najdeme 16 portů sběrnice CAN FD a 22 portů sběrnice LIN. Zařízení je schopné zachytávat komunikaci ze 4 sériových portů. Kapacita úložiště dosahuje až 3 TB. [1] Cena se pohybuje kolem 20 tisíc eur. [2]

Obrázek 3.1. MAGNA Telemotive BLUEPIRAT Rapid [1]

3.2 X2E XORAYA N4000

Další řešení nabízí společnost X2E, konkrétně jde o produkt XORAYA N4000. Hlavní rozdíl oproti BLUEPIRAT Rapid je, že je částečně rozšiřitelný, ale také obsahuje jenom Ethernety standardu 1000BASE-T. Proto musíte dokoupit další převodníky z 1000BASE-T na 1000BASE-T1 pro každý port. K dispozici je takových portů osm. Dále obsahuje 4 porty sběrnice CAN FD a 8 portů RS232. Zařízení lze rozšířit přes 3 volné sloty, do kterých lze umístit například 4 další CAN FD porty nebo 24 portů sběrnice LIN. Pro záznam dat je úložiště o kapacitě až 1 TB. [3] Cena se pohybuje kolem 20 tisíc eur. [4]

Obrázek 3.2. X2E XORAYA N4000 [3]

3.3 Intrepid Control Systems RAD-Gigalog

RAD-Gigalog od společnosti Intrepid Control Systems je nejlépe vybavené zařízení co se kapacity úložiště týče. Lze na něj totiž uložit až 12 TB záznamu. Na druhou stranu co se týče dostupných portů, tak už to není tak obdivuhodné. Pro CAN FD je zde jen 8 portů, a žádná sériová linka. Dále máme 2 porty pro Ethernet 100BASE-T1, takže pro gigabitový Ethernet je potřeba použít externí převodník. [5] Cena se pohybuje kolem 18 tisíc eur. [6]

Obrázek 3.3. Intrepid Control Systems RAD-Gigalog [5]

3.4 Shrnutí

Při hledání existujících zařízení se ukázalo, že zatím na trhu není žádné zařízení, které přímo podporuje Ethernet 1000BASE-T1, ale je potřeba použít externí převodník. Jelikož většina zařízení podporuje buď jenom 100BASE-T1 nebo 1000BASE-T. CAN FD je již standardně používán a implementován ve všech zařízeních. Sériová linka vzhledem ke své historické roli také nepředstavovala téměř žádný problém.

Většina zařízení není vůbec rozšiřitelná, výjimku tvoří jen XORAYA N4000, která je jen částečně rozšiřitelná. Jejich cena se pohybuje kolem 20 tisíc eur, co je jeden z parametrů, v čem by mělo být toto řešení lepší než existující řešení. Na druhou stranu mají kompaktnější rozměry, než jaké bude mít toto řešení. To by ale nemělo mít velkou váhu pro cílovou skupinu, kam toto řešení cílí.

Kapitola 4
Analýza hardwaru

V této kapitole podrobněji rozeberu jednotlivé možné stavební prvky celého řešení. Půjde o PCI Express kartu pro zaznamenávání CAN FD, převodník Ethernetu 1000BASE-T1 na 1000BASE-T a mikroprocesor pro hardwarový trigger.

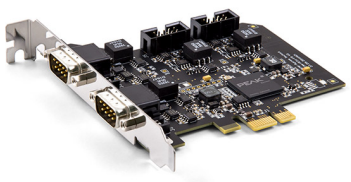
Mezi hlavní kritéria pro výběr patří jednoduchost nasazení, aby se co nejvíc zkrátil čas potřebný pro vývoj. Další neméně důležité kritérium je škálovatelnost pro budoucí potřeby. Ale nesmíme zapomenout na to, že je důležitá také jeho cena, aby se výsledné řešení udrželo cenou na přijatelné úrovni.

4.1 CAN FD

Pro zachytávání sběrnice CAN FD použiji přídavnou kartu do PCI Express. Mezi požadavky patří podpora rozhraní SocketCAN [7] a co nejvyšší počet CAN FD kanálů. SocketCAN byl zvolen proto, aby řešení nebylo závislé na jednom konkretním rozhraní od nějakého výrobce. To zajišťuje, že v případě potřeby používání jakékoli jiné karty, která podporuje SocketCAN, není nutný zásah do programu.

4.1.1 PEAK-System PCAN-PCI Express FD

Výrobce PEAK-System nabízí kartu PCAN-PCI Express FD. Ovladač je součástí hlavní větve Linuxového jádra, což zaručuje podporu rozhraní SocketCAN. Vrcholem nabídky je model se 4 kanály CAN FD vyvedenými z jedné PCI Express linky. Cena 4 portové karty je 350 eur. [8]



Obrázek 4.1. PEAK-System PCAN-PCI Express FD [8]

4.1.2 Ixxat CAN-IB600/PCIe

Jako alternativa je karta CAN-IB600/PCIe od výrobce Ixxat. Karta podporuje rozhraní SocketCAN, ale ovladač není součástí hlavní větve Linuxového jádra, a proto je ho potřeba stáhnout ze stránky výrobce. To zbytečně komplikuje nasazení. Dále tento model je dostupný maximálně se 2 CAN FD kanály. [9] Cena takové konfigurace se pohybuje kolem 750 eur. [10]

Obrázek 4.2. Ixxat CAN-IB600/PCIe [9]

4.2 Ethernet

Jelikož na trhu zatím není žádná síťová karta do PCI Express, která by podporovala standard 1000BASE-T1, tak použiji běžně dostupnou gigabitovou síťovou kartu a převodník. Naštěstí u převodníků je z čeho vybírat. I když s ohledem na jejich jednoúčelové použití, se téměř (kromě ceny) není podle čeho rozhodovat. Všechny totiž jen převádí Ethernet 1000BASE-T1 na 1000BASE-T. Liší se jenom použitými konektory a vstupním napájením.

4.2.1 Intrepid Control Systems RAD-Moon 2

Převodník RAD-Moon 2 od společnosti Intrepid Control Systems používá jako základ čip Marvell 88Q2112 A2. Pro 1000BASE-T1 používá konektor TE MATEnet. Napájen může být přes USB typu C nebo přes Barrel Jack. Pracovní napětí je 4,5 V až 40 V. [11] Cena převodníku se pohybuje kolem 800 eur. [12]

Obrázek 4.3. Intrepid Control Systems RAD-Moon 2 [11]

4.2.2 Technica Engineering 1000BASE-T1 MediaConverter

Společnost Technica Engineering nabízí model s názvem 1000BASE-T1 MediaConverter, který také používá čip Marvell 88Q2112 A2. Konektor pro 1000BASE-T1 byl použit TE NanoMQS, v kterém je obsažené i napájení. To může být od 6,5 V do 16 V. Zařízení také splňuje stupeň krytí IP20. [13] Cena převodníku se pohybuje kolem 700 eur. [14]

Obrázek 4.4. Technica Engineering 1000BASE-T1 MediaConverter [13]

4.2.3 X2E XORAYA Automotive Ethernet Converter

Jako další alternativa je zde převodník XORAYA Automotive Ethernet Converter od společnosti X2E. Tak jako všechny předešlé převodníky i on používá čip Marvell 88Q2112 A2. Na zařízení se nachází 8 pinový konektor, který kombinuje Ethernet 1000BASE-T1, CAN a napájení. Pracovní napětí musí být v rozmezí od 6 V do 28 V. [15] Cena převodníku se pohybuje kolem 650 eur. [16]

Obrázek 4.5. X2E XORAYA Automotive Ethernet Converter [15]

4.2.4 GÖPEL electronic EasyCON 1000Base-T1 Media Converter

Poslední řešení nabízí společnost GÖPEL electronic, a to model EasyCON 1000Base-T1 Media Converter. Na stránce výrobce není uvedeno, jaký čip byl použit pro fyzickou vrstvu Ethernetu 1000BASE-T1, ale nepředpokládám, že by to mělo být něco jiného než Marvell 88Q2112 A2. Jako konektor pro 1000BASE-T1 byl použit poněkud neobvyklý pro průmyslové použití RJ-45, ale v balení je také adaptér na častěji využívaný 9 pinový D-Sub. Zařízení je napájeno z Barrel Jacku, do kterého může být přivedeno napětí od 5 V do 30 V nebo přes USB konektor. Zařízení splňuje stupeň krytí IP20. [17] Cena převodníku se pohybuje kolem 500 eur. [18]

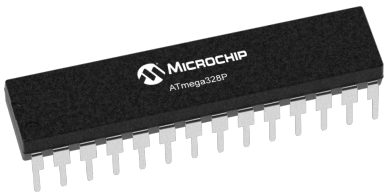
Obrázek 4.6. GÖPEL electronic EasyCON 1000Base-T1 Media Converter [17]

4.3 Hardwarový trigger

Pro použití v hardwarovém triggeru potřebujeme najít vhodný mikroprocesor. Mezi požadavky patří možnost připojit displej, podpora sériové linky a podpora přerušení. Důležitá je dostupnost knihoven pro práci s displejem a sériovou linkou pro daný mikroprocesor.

4.3.1 Microchip ATmega328P

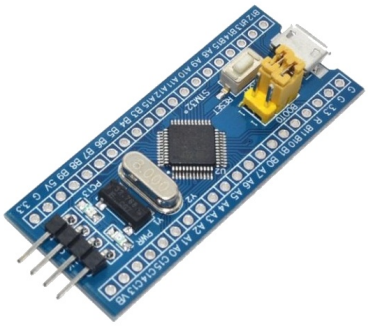
Jako první možnost se nabízí 8-bitový mikroprocesor ATmega328P od společnosti Microchip. Původně se jedná o produkt společnosti Atmel, která ale byla odkoupena společností Microchip. Jeho velkou výhodou je použití v projektu Arduino [19]. Díky tomu existuje velké množství knihoven a rozšiřujících zařízení určeno pro tento mikroprocesor. [20] Cena se pohybuje kolem 70 Kč. [21]



Obrázek 4.7. Microchip ATmega328P [20]

4.3.2 STMicroelectronics STM32F103C8T6

Alternativa je model STM32F103C8T6 od společnosti STMicroelectronics. Jedná se o vývojovou desku s procesorem STM32F103C8, co je 32-bitové jádro ARM Cortex-M3. I když se neobjevil v žádném oficiálním Arduino zařízení, existuje podpora pro tento mikroprocesor v Arduino IDE. [22] Cena se pohybuje kolem 200 Kč. [23]



Obrázek 4.8. STMicroelectronics STM32F103C8T6 [24]

Kapitola 5
Analýza softwaru

Požadavky zůstávají podobné jako v případě hardwaru, a to jednoduchost nasazení, otestovanost softwaru, a z toho plynoucí spolehlivost. Budeme analyzovat jednotlivé knihovny potřebné pro podpůrné části nástroje. Jedná se o HTTP server pro webové API a knihovnu na práci s JSON. Ale také i možnosti zaznamenávání jednotlivých typů sběrnic, a to CAN FD, Ethernetu, HTTP streamu a sériové linky.

5.1 HTTP Server

Pro potřeby webového API budeme potřebovat HTTP server. Použijeme proto nějakou již hotovou knihovnu. Rozhodovat se budeme podle dokumentace a jednoduchosti používaní. Neméně důležité je, zda je knihovna dostatečně otestována a aktivně vyvíjená. Součástí repozitářů Debianu jsem přímo žádnou takovou knihovnu nenašel, proto další kritérium bude i to, jak pracné ji bude zakomponovat do projektu.

5.1.1 cpp-httplib

Knihovna cpp-httplib [25] od Yuji Hirose je tvořená jediným hlavičkovým souborem httplib.h, proto by nasazení mělo být jednoduché. Knihovna obsahuje všechny potřebné funkcionality pro moje řešení. Dokumentace obsahuje dostatečné množství příkladů použití. Jenom práce s regulárními výrazy by mohla být více uživatelsky přívětivá.

5.1.2 libhttpserver

Knihovna libhttpserver [26] od Sebastiana Merlina je postavená na základu knihovny libmicrohttpd [27], kterou můžeme považovat za dostatečně otestovanou, díky počtu projektů, které ji využívají. Dokumentace s ukázkami použití je dostupná a úplně postačující pro naše účely. Zakomponování knihovny do projektu by neměl být problém.

5.2 JSON

Jelikož webové API bude odpovídat ve formátu JSON, tak použiji nějakou z dostupných knihoven. Jedno z kritérií pro výběr bylo, aby knihovna byla součástí repozitářů distribuce Debian, kvůli jednoduchosti nasazení. Dále byla požadovaná co nejobsáhlejší dokumentace pokrývající co nejvíc případu užití s příklady.

5.2.1 rapidjson

Společnost Tencent vyvíjí knihovnu rapidjson [28]. Jak je z názvu patrné, je zaměřená na rychlost. Je součástí repozitářů Debianu, kde ji najdeme jako balíček rapidjson-dev [29]. Dokumentace je obsáhlá, ale je trochu obtížnější se v ní zorientovat a najít co hledáte. Také její rozhraní by mohlo být trochu intuitivnější, ale to je jen můj subjektivní názor.

5.2.2 json

Jako další alternativa je knihovna json [30] od Nielse Lohmanna. V repozitářích Debianu ji najdeme jako balíček nlohmann-json3-dev [31]. Na rozdíl od knihovny rapidjson její prioritou není výkon, ale jednoduchost používaní a otestovanost. Také je výborně zpracovaná dokumentace s ukázkovými příklady použití, díky které je seznámení s knihovnou velice rychlé.

5.3 Zaznamenávání sběrnic

V této sekci si popíšeme, jak zachytávat každou sběrnici jednotlivě. Bude tam podrobněji vysvětleno jaký nástroj na to bude použitý a v jakém formátu budou data uložena. Budou převážně využívané již existující nástroje, z důvodu jejich lepší otestovanosti díky širokému nasazení.

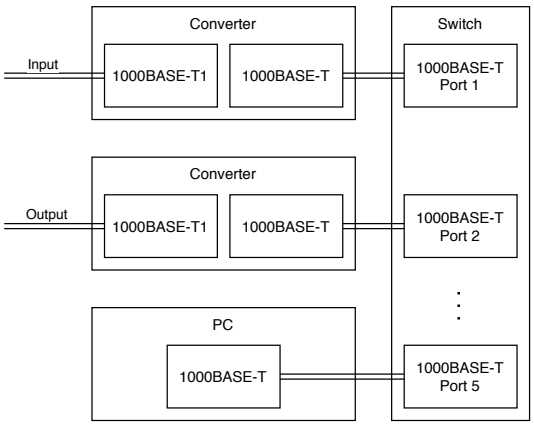
5.3.1 CAN FD

Pro ukládání dat ze sběrnice CAN byl zvolen formát ASCII Logging Files (.asc). Tento formát byl zvolen pro to, že byl požadavek na možnost otevření záznamu v programu CANoe od společnosti Vector. Tento formát je jedním z podporovaných, a zároveň dostatečně čitelný pro člověka.

Protože námi použitá karta podporuje rozhraní SocketCAN, tak se nabízí použití can-utils, jenže tyto utility podporují formát ASCII Logging Files (.asc), až jako konverzi z jejich log formátu. To by zbytečně komplikovalo práci s daty, proto se vyvinul samostatný program can-trace. Jako can-utils, také používá rozhraní SocketCAN, jenže produkuje výstup už přímo v požadovaném formátu.

5.3.2 Ethernet

Pro zaznamenávání Ethernetu 1000BASE-T1 použijeme dva převodníky z Ethernetu 1000BASE-T1 na 1000BASE-T a jeden 5 portový gigabitový switch s funkcí zrcadlení portů. Switch nastavíme, aby porty 1 až 4 zrcadlil na port 5. Řídící počítač připojíme do portu 5 a převodníky do portů 1 a 2.



Obrázek 5.1. Schéma zapojení převodníků 1000BASE-T1 a switche.

Pro záznam komunikace byla zpočátku zvolena utilita tshark, což je rozhraní příkazového řádku nástroje Wireshark. Později byla změněna na utilitu dumpcap, kterou interně tshark používá. Bylo to kvůli snížení počtu zahozených paketů, ke kterému mohlo dojít v důsledku zatížení systému [32]. Výstupní formát dat je pcapng.

5.3.3 HTTP stream

Potřebujeme zaznamenávat data, která zařízení vysílá na svojí IP adrese na určeném portu. Komunikace probíhá přes TCP protokol. Pro tyto potřeby jsem použil utilitu netcat. Jde o lety ověřenou utilitu pro čtení a zápis dat přes TCP nebo UDP protokol. Data se nijak neupravují a jen se uloží na disk.

5.3.4 Sériová linka

Požadavek byl na ukládání komunikace ze sériové linky. Protože jsme na Linuxu, tak port sériové linky je dostupný jako soubor, který můžeme číst pomocí příkazu cat. Předem je jen potřeba nastavit správnou rychlost. Data se také nijak neupravují.

Kapitola 6
Návrh řešení

Po dokončení analýzy se začalo s návrhem možného řešení. Byly vybrány stavební prvky jak na straně hardwaru, tak i softwaru. Pak nasledoval návrh architektury. Při návrhu se kladl důraz na jednoduché používaní API z pohledu automatizovaného testování. Ale také na co největší modularitu konečného řešení.

6.1 Výběr stavebních prvků

Na základě analýzy se rozhodlo o použití následujících prvků. Dále jsou uvedeny důvody, které byly klíčové pro rozhodnutí použít je v našem řešení.

6.1.1 CAN FD

Pro zachytávání sběrnice CAN FD byla zvolena karta PCAN-PCI Express FD od výrobce PEAK-System. Bylo to jednak kvůli lepší podpoře ovladače v jádru, ale také díky vyššímu počtu CAN FD kanálů, to má za následek, že na pokrytí vyššího počtu kanálů je potřeba méně linek PCI Express. Nezanedbatelná je také o dost nižší pořizovací cena oproti alternativě.

6.1.2 Ethernet

Jak bylo vzpomenuto v předešlé kapitole, tak mnoho rozdílů mezi převodníky není. A jelikož by každý vyhovoval našemu použití, tak byl zvolen EasyCON 1000Base-T1 Media Converter od společnosti GÖPEL electronic především kvůli jeho nejnižší ceně. Jako příjemný bonus považuji stupeň krytí IP20.

6.1.3 HTTP Server

Byla zvolena knihovna libhttpserver, i když zatím není součástí repozitářů Linuxové distribuce Debian, tak vytvořit instalační balíček bylo jednoduché. Rozhodl jsem se pro tuto knihovnu z důvodu, že je postavená na otestované knihovně libmicrohttpd. Také jako další její výhodu považuji jednoduchou práci s regulárními výrazy.

6.1.4 JSON

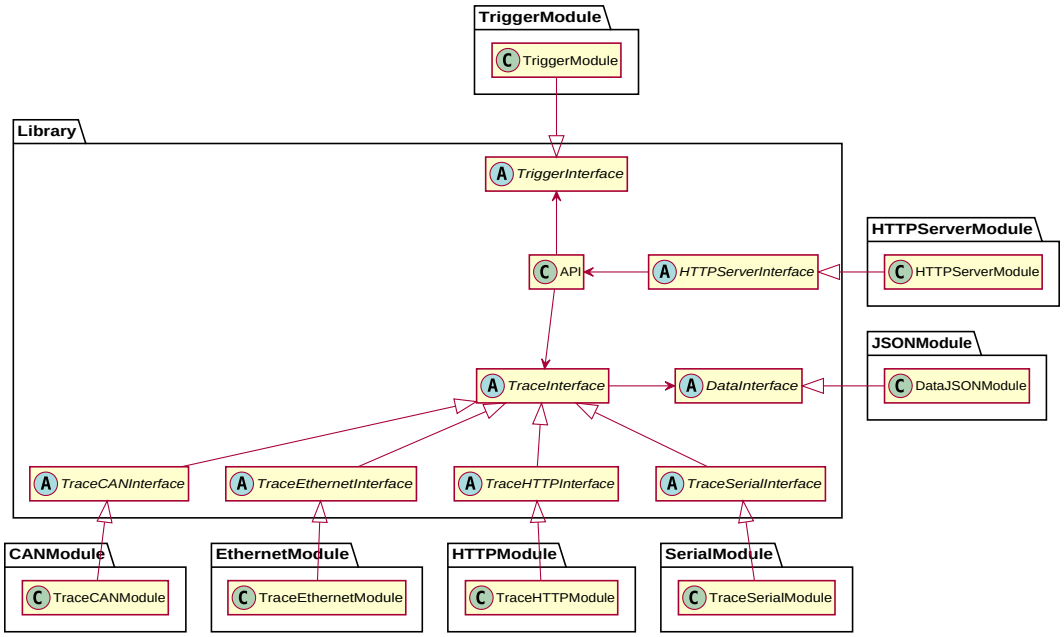
Jelikož výkon není prioritou pro naše zpracování formátu JSON, tak jsem zvolil knihovnu json pro její intuitivní používaní. Jako další důvod byla její rozšířenost, a z toho plynoucí otestovanost. Také jsem byl ohromen perfektní dokumentaci, která pokrývá široký záběr případů užití. Integrace knihovny do projektu byla také bezproblémová, protože se nachází v oficiálních repozitářích.

6.2 Architektura

V této kapitole bude podrobněji popsáno navrhování architektury řešení. Jedná se o výběr programovacího jazyku, rozložení programu do modulů a způsob implementace uživatelského rozhraní.

6.2.1 Návrh architektury

Konečný návrh je realizován pomocí nativní aplikace, která se spouští po startu systému. Součástí řešení je vytvoření frameworku, který bude aplikace využívat. Framework bude sloužit na abstrakci implementací jednotlivých sběrnic. Samotná implementace požadovaného rozhraní je až na úrovni modulu. Mezi potřebné moduly patří HTTP server pro potřeby webového API a JSON modul na práci s daty.



Obrázek 6.1. Diagram knihovny.

6.2.2 Programovací jazyk a prostředí

Pro implementaci byl zvolen jazyk C++ kvůli standardní knihovně, výkonu a množství existujících dostupných knihoven. Pro automatizaci překladu programu byl použit software CMake, protože zajišťuje kompatibilitu mezi různými vývojovými prostředími a je lehce integrovatelný do balíčkovacího procesu pro Linuxovou distribuci Debian.

6.2.3 Uživatelské rozhraní

Uživatelské rozhraní je implementováno formou webové stránky, která pomocí Javascriptu komunikuje s API. Webová stránka byla zvolena pro širší pokrytí zařízení schopných ovládat řídící program oproti nativní aplikaci. Pro snazší ovládání na mobilech je použitý responzivní design.

6.3 Hardwarový trigger

Po dokončení návrhu architektury, se přešlo k návrhu hardwarového triggeru. Začalo se zvolením použitého hardwaru, s ohledem na co nejjednodušší konečný návrh DPS. Co se týče softwaru, tak nebylo potřeba použít žádné externí knihovny, protože všechen potřebný kód byl součástí Arduino IDE.

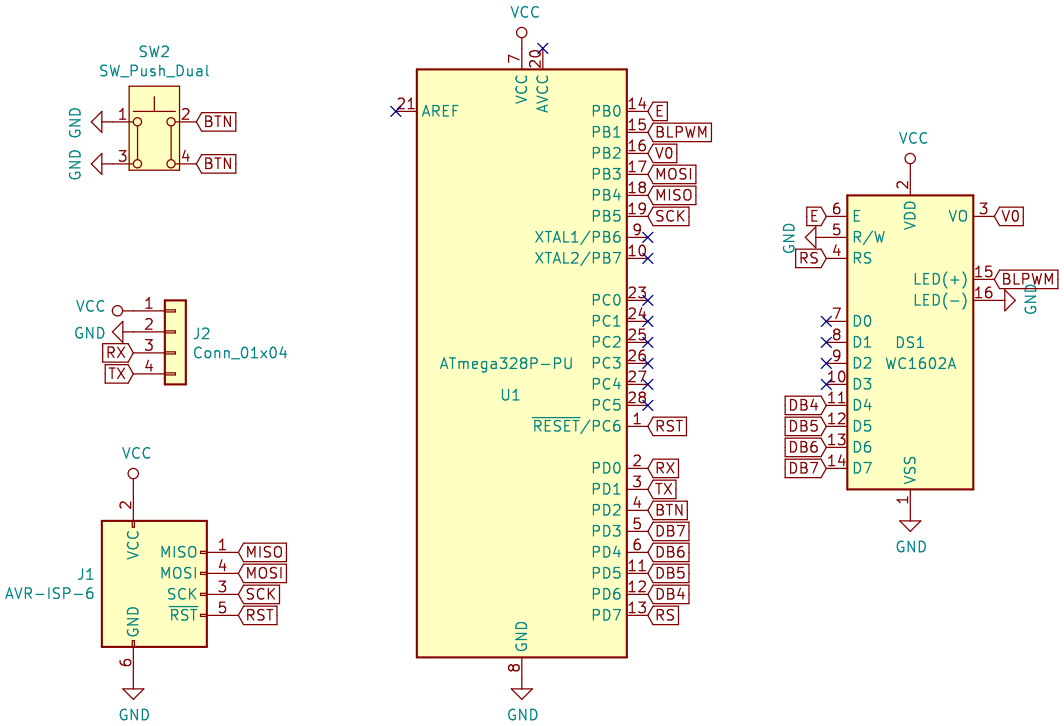
6.3.1 Hardware

Byl použitý mikroprocesor ATmega328P od společnosti Microchip. Zvolen byl hlavně kvůli jednoduššímu osazení na DPS, a také kvůli mým zkušenostem s tímto mikroprocesorem. Pro zjednodušení návrhu nebyl použit externí oscilátor, jako tomu obvykle bývá při použití tohoto mikroprocesoru, ale byl využit interní, který sice nemá takovou přesnost, ale pro naše použití to postačuje. Celé zařízení pracuje na napětí 3,3 V.

Kromě mikroprocesoru zde nalezneme LCD modul LCD1602 s rozlišením 16x2 znaků. Dále je zde tlačítko, na které reaguje mikroprocesor zasláním signálu řídícímu počítači. Nalezneme zde také AVR ISP 6 pin konektor pro jednoduché nahrávání firmwaru do mikroprocesoru.

6.3.2 Návrh schématu

Pro tyto účely byl použit program KiCad [33]. Všechny potřebné součástky kromě použitého tlačítka už měly v KiCadu připravený model na použití. Podle technické specifikace tlačítka byl navržen vlastní model. Pak následovalo označení jednotlivých pinů ve schématu.



Obrázek 6.2. Schéma.

6.3.3 Software

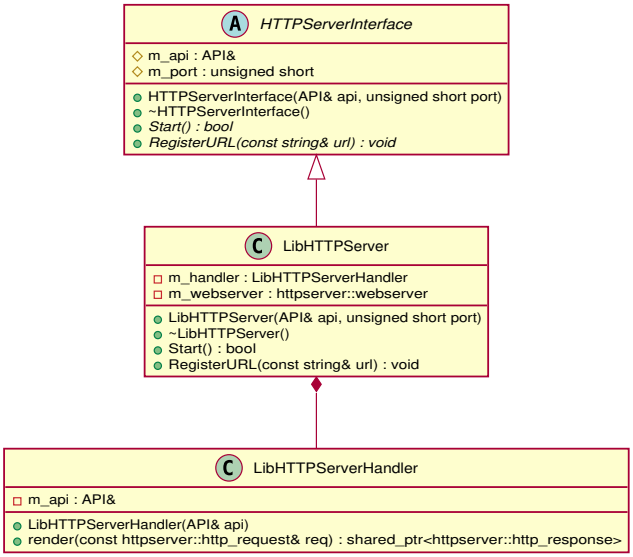
Jak bylo vzpomenuto v předešlé sekci, je možné využít knihovny a vývojové prostředí projektu Arduino. Pro práci s LCD modulem byla použita knihovna LiquidCrystal [34]. Dále pro potřebu komunikace mikroprocesoru s řídicím počítačem jsem využil třídu Serial [35].

Kapitola 7
Realizace softwaru

Když byl návrh řešení dokončený, tak se začalo s realizací softwarové části. To zahrnuje realizaci knihovny, jednotlivých modulů a webového rozhraní. V této kapitole jsou podrobněji popsány nejdůležitější třídy, které se používají pro zaznamenávání nebo pro podpůrné funkcionality jako webové API nebo trigger. Pro zachování přehlednosti byla vynechána většina gettrů a settrů.

7.1 Třída HTTPServerInterface

Třída HTTPServerInterface popisuje rozhraní pro HTTP server zprostředkující API. Tady bylo implementováno rozhraní pro knihovnu LibHTTPServer. Ta vyžaduje další třídu pro zpracování HTTP požadavků, to bylo realizované pomocí třídy LibHTTPServerHandler. Tato třída pak přistupuje k třídě API.



Obrázek 7.1. Třída HTTPServerInterface s implementací.

7.1.1 Metoda Start

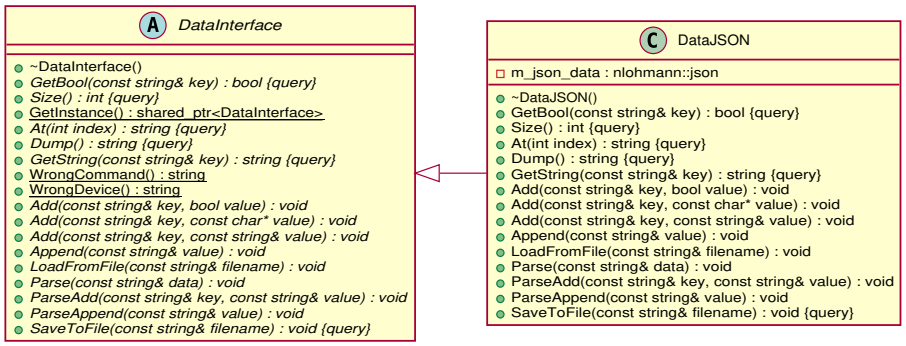
Metoda Start má na starost spuštění HTTP serveru na předem zvoleném portu.

7.1.2 Metoda RegisterURL

Metoda RegisterURL registruje URL adresy webového API, které chceme obsluhovat přes HTTP server.

7.2 Třída DataInterface

Třída DataInterface tvoří rozhraní sloužící na práci s daty, využívající se při generování odpovědí funkcí nebo při práci platformami. V tomto řešení bylo implementováno pomocí JSON knihovny, ale je možné ho implementovat například pomocí XML knihovny. V takovém případě by metody vracely odpovědi ve formátu XML a platformy by byly rovněž tak uloženy na disku ve formátu XML.



Obrázek 7.2. Třída DataInterface s implementací.

7.2.1 Metoda Parse

Metoda Parse se stará o rozparsování stringu na vstupu do objektu, který je indexovatelný podle klíče nebo na iterovatelný seznam. Tato metoda se používá při deserializaci zařízení.

7.2.2 Metoda Dump

Metoda Dump slouží na serializaci vytvořeného objektu do podoby stringu. Používá se při tvorbě odpovědi v metodách zařízeních, ale také v metodě Serialize.

7.2.3 Metoda Add

Metoda Add pracuje na principu mapy, to znamená, že se přidá položka na místo určené klíčem. Jelikož položka může být různého typu, tak je podle toho přetěžovaná metoda.

7.2.4 Metoda ParseAdd

Metoda ParseAdd je podobná metodě Add, jen s tím rozdílem, že se před přidáním pokusí rozparsovat položku na vstupu.

7.2.5 Metoda Append

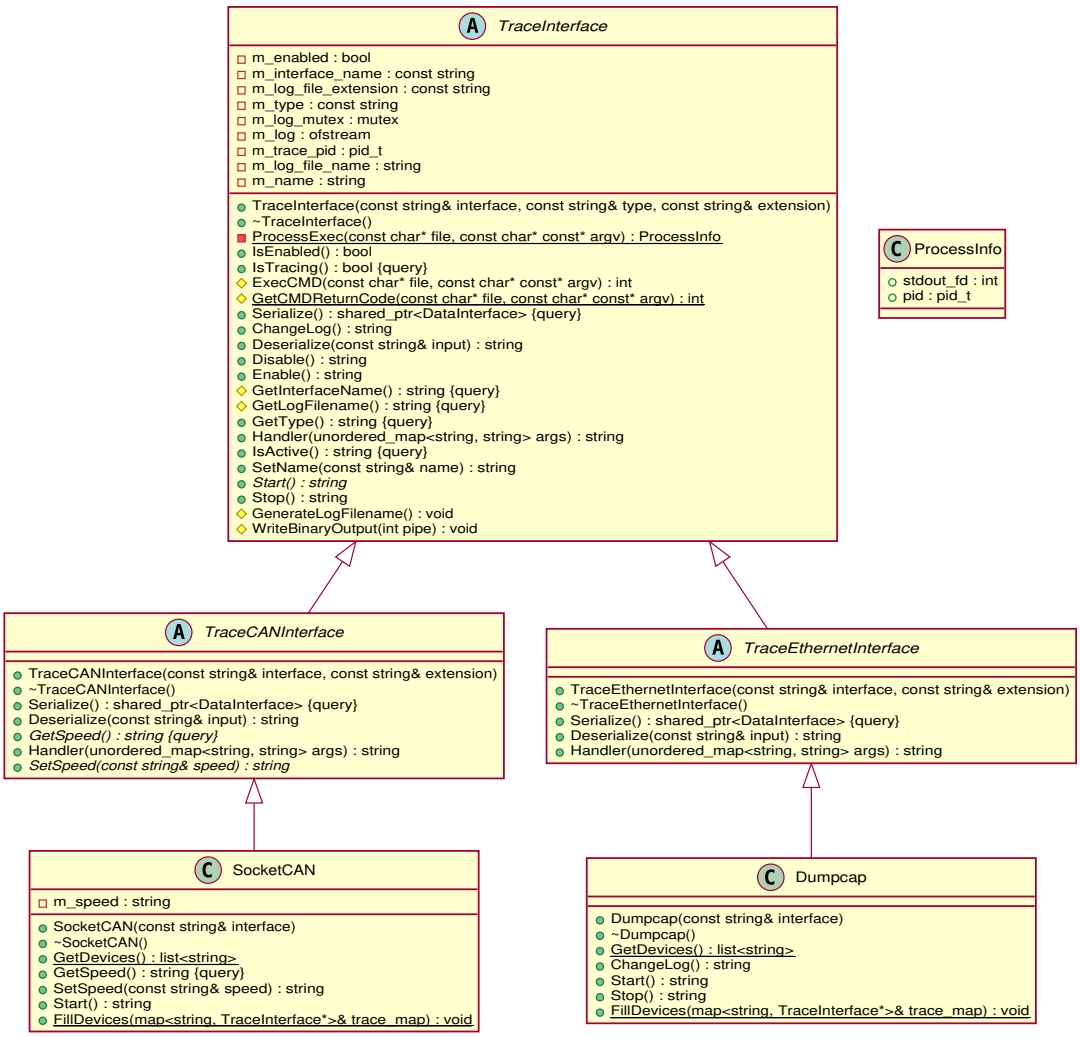
Metoda Append přidává položku na konec seznamu.

7.2.6 Metoda ParseAppend

Metoda ParseAppend je podobná metodě Append, jen s tím rozdílem, že se před přidáním pokusí rozparsovat položku na vstupu.

7.3 Třída TraceInterface

Tato třída popisuje základní rozhraní pro implementaci podpory jednotlivých sběrnic. Z třídy TraceInterface jsou pak zděděné třídy pro jednotlivé typy, které mohou mít víc implementací. To má za následek, že můžete zachytávat jeden typ sběrnice pomocí různých nástrojů, ale pořád budete používat jednotné API. Tyto třídy vyžadují implementaci funkcí, potřebných pro daný typ sběrnice (například u CANu jde o nastavení rychlosti, což u HTTP streamu není potřeba).



Obrázek 7.3. Třída TraceInterface s implementacemi.

7.3.1 Metoda Handler

Virtuální metoda Handler se stará o volání dalších metod podle parametrů získaných z HTTP požadavku. Metoda je navržená tak, aby se na každé úrovni starala jen o metody, které byly implementované na dané úrovni. Pokud nedokáže obsloužit požadavek, tak zavolá metodu Handler o úroveň výše se stejnými parametry. Pokud to nedokáže obsloužit Handler na úrovni třídy TraceInterface, tak vrací chybovou hlášku.

V třídách dědících z TraceInterface má příznak final, to zaručuje, že už další třída, která ji bude dědit, nemůže přepsat tuto metodu. Takový přístup byl zvolen pro zachování jednoty rozhraní.

7.3.2 Metoda Serialize

Další virtuální metoda, tato ale provádí serializaci objektu. Používá se při ukládání platformy, kdy se objekt serializuje a uloží na disk. Její výstup je také k dispozici při použití příkazu info nad zvoleným zařízením v API. Také jako metoda Handler, i tato se stará jen o prvky implementované v její třídě a pak zavolá Serialize třídy TraceInterface. Stejně je použitý příznak final jako v případě metody Handler.

7.3.3 Metoda Deserialize

Metoda Deserialize zabezpečuje proces nastavení zařízení podle předem uložených nastavení. Je tam použitý stejný princip jako v metodě Serialize, a to že se načítají hodnoty z konfiguračního souboru jen pro ty členské proměnné, které jsou přidané oproti třídě z které dědíme. Také je použitý příznak final v třídě dědící z třídy TraceInterface.

7.3.4 Metoda Start

Abstraktní metoda Start se stará o spuštění zaznamenávání komunikace. Implementovat tuto metodu má za úkol až třída dědící z třídy rozhraní pro jednotlivé typy sběrnic. Její implementace typicky pozůstává z volání metody ExecCMD a následně WriteBinaryOutput.

7.3.5 Metoda Stop

Tato metoda zastavuje zaznamenávání komunikace. Předvolené chování této metody je, že pošle signál SIGTERM na proces s PID uloženým v proměnné m_trace_pid. Pak vyčká až se proces zastaví a nastaví hodnotu m_trace_pid na nulu. Metoda je deklarovaná jako virtuální, takže nic nebrání jejímu přepsání podle potřeb.

7.3.6 Metoda ChangeLog

Virtuální metoda ChangeLog mění soubor, do kterého se zapisují zachycená data. Předvolená implementace je, že pomocí mutexu si zabezpečí, že jako jediná metoda bude v tom okamžiku přistupovat ku otevřenému souboru. Zavře soubor, a pak otevře nový soubor s jiným názvem. Jelikož byl použitý std::lock_guard, tak k odemčení mutexu dochází automaticky po opuštění bloku, v kterém byl definovaný.

7.3.7 Metoda GenerateLogFilename

Metoda GenerateLogFilename se stará o vygenerování nového názvu souboru, kde se ukládá zaznamenávaná komunikace. Název souboru se skládá z typu sběrnice, názvu zařízení, aktuálního času a přípony souboru.

7.3.8 Metoda ProcessExec

Privátní metoda ProcessExec, slouží ke spouštění procesů a získávání jejich standardního výstupu. Metoda vrací strukturu ProcessInfo, která obsahuje PID a file descriptor roury ze standardního výstupu daného procesu. Metoda funguje na principu, že nejprve vytvoří rouru, a pak zavolá funkci fork, jejíž návratová hodnota se uloží do proměnné pid v struktuře, která se bude vracet jako návratová hodnota metody.

V procesu potomka pak nahradí standardní výstup za konec roury určený pro zapisování, zavře všechny konce roury, vyprázdní všechny buffery a zavolá funkci execvp, to způsobí přepsání programu v paměti binárním souborem s názvem uloženým v parametru file. Pak už proces potomka pokračuje ve vykonávaní instrukcí z nově nahraného binárního souboru.

Proces rodiče zavře konec roury určený pro zápis a konec určený pro čtení uloží do struktury, kterou metoda vrací jako návratovou hodnotu. Implementace metody je následovná.

```
ProcessInfo TraceInterface::ProcessExec(const char*        file,
                                        const char* const* argv) {
  int out_pipe[2];
  pipe2(out_pipe, 0);

  ProcessInfo result;

  result.pid = fork();

  if(result.pid == 0) {
    dup2(out_pipe[1], STDOUT_FILENO);

    close(out_pipe[0]);
    close(out_pipe[1]);

    fflush(nullptr);

    execvp(file, const_cast<char* const*>(argv));
  }

  close(out_pipe[1]);

  result.stdout_fd = out_pipe[0];

  return result;
}
```

7.3.9 Metoda ExecCMD

Metoda ExecCMD se stará o spuštění procesu, který produkuje výstup, který se bude zaznamenávat, a vytvoření roury ze standardního výstupu daného procesu. Metoda vrací file descriptor roury, který se použije v metodě WriteBinaryOutput. Interně se využívá privátní metoda ProcessExec.

7.3.10 Metoda GetCMDReturnCode

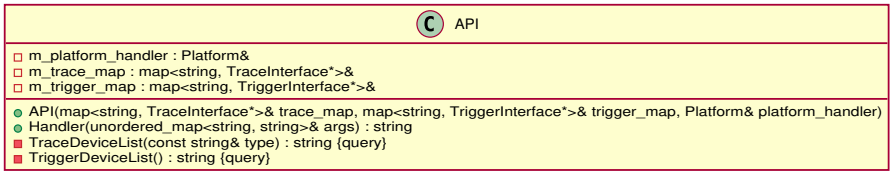
Metoda GetCMDReturnCode slouží na spuštění procesu a zachycení jeho návratového kódu, který je návratová hodnota této metody. Interně využívá metodu ProcessExec. Primární využití této metody je pro ulehčení práce s procesy v jednotlivých modulech, při nastavovaní zařízení, kdy je potřeba volat externí příkazy.

7.3.11 Metoda WriteBinaryOutput

Metoda WriteBinaryOutput zapisuje data, která získá z roury, kterou dostane parametrem. Práce se souborem je chráněná mutexem, jelikož metoda ChangeLog může také manipulovat s cílovým souborem. V případě, že proces, kterého výstup získáváme skončí, tak převezme jeho návratovou hodnotu a nastaví m_trace_pid na nulu, aby bylo jasné, že už není aktivní zaznamenávání.

7.4 Třída API

Třída API vyřizuje HTTP požadavky přijaté HTTP serverem. Především volá metodu Handler na objekt, kterého se požadavek týká. Ale také zjišťuje názvy připojených zařízení.



Obrázek 7.4. Třída API.

7.4.1 Metoda Handler

Metoda Handler je volána HTTP serverem po obdržení HTTP požadavku. Její úkol je zjistit podle parametrů HTTP požadavku kterou další metodu má zavolat pro získaní odpovědi na požadavek. V případě, že jde o neplatný příkaz nebo zvolené zařízení neexistuje, tak vrací chybovou hlášku.

7.4.2 Metoda TraceDeviceList

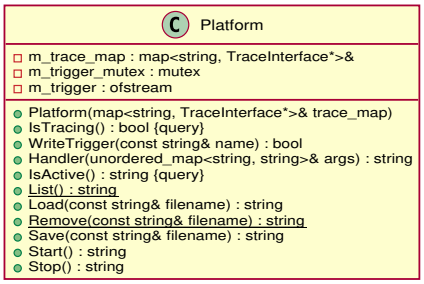
Metoda TraceDeviceList vrací seznam zařízení určitého typu, který dostane parametrem. Jelikož jsou všechny zařízení pro zaznamenávání uloženy v jedné mapě, je potřeba ji projít celou a zkontrolovat, jestli dané zařízení je toho typu, který chceme.

7.4.3 Metoda TriggerList

Metoda TriggerList má stejný úkol jako metoda TraceDeviceList, jenže vrací seznam triggerů. Rozdíl je ten, že vrací každý prvek mapy, v které jsou uloženy triggery.

7.5 Třída Platform

Tato třída se stará o práci s platformami – tj. načítaní a ukládání. Platformy jsou realizované pomocí souborů, které jsou uloženy v `/var/tracebox/platforms/`. Formát dat je odvozený od modulu implementujícího třídu DataInterface, v tomto případě to je formát JSON. Platformy definují parametry jednotlivých zařízení, mezi kterýma lze jednoduše a rychle přepínat.



Obrázek 7.5. Třída Platform.

7.5.1 Metoda List

Metoda List vrací string obsahující seznam uložených platforem. Data získává z filesystému, kterého se dotáže na seznam souborů v `/var/tracebox/platforms/`.

7.5.2 Metoda Start

Metoda Start se stará o spuštění zaznamenávání na všech zařízeních, které jsou součástí platformy.

7.5.3 Metoda Stop

Metoda Stop zastavuje všechna aktivní zaznamenávání.

7.5.4 Metoda Load

Metoda Load slouží na načítaní uložených dat platforem. Načítá zvolený soubor, a snaží se rozparsovat data. Pak zavolá metodu Deserialize pro každé zařízení, které je popsáno v daném souboru.

7.5.5 Metoda Save

Metoda Save funguje na opačném principu jako metoda Load. Pro každé zařízení zavolá metodu Serialize a její výstup uloží do souboru se zvoleným názvem.

7.5.6 Metoda Remove

Metoda Remove slouží na odstranění zvolené platformy. Její realizace je jen o odstranění souboru se zvoleným názvem ze složky `/var/tracebox/platforms/`.

7.5.7 Metoda Handler

Metoda Handler je typicky volána z třídy API, a stará se o volání metod podle parametrů z HTTP požadavku.

7.5.8 Metoda WriteTrigger

Metoda WriteTrigger slouží pro zápis časového razítka triggeru do souboru. Typicky je volána z metody TriggerAction třídy TriggerInterface. Při přístupu k souboru, kde se má informace zapsat se používá mutex pro zabezpečení výhradního přístupu.

7.5.9 Metoda IsTracing

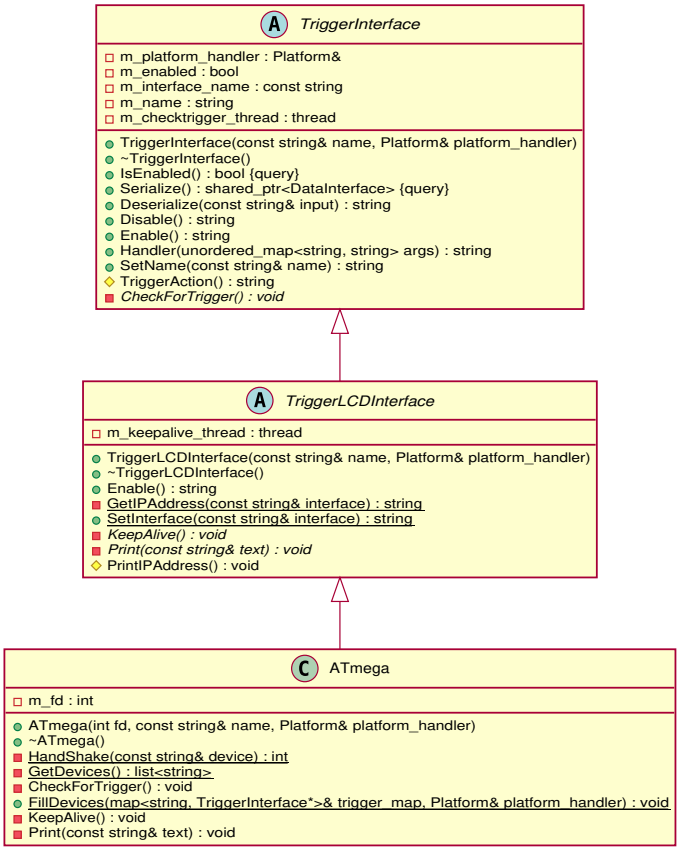
Metoda IsTracing vrací bool hodnotu indikující jestli je aktivní zaznamenávání nějakého zařízení, které je součástí platformy.

7.5.10 Metoda IsActive

Metoda IsActive vrací string obsahující stav jestli je aktivní zaznamenávání. Tato metoda se volá z webového API.

7.6 Třída TriggerInterface

Třída TriggerInterface poskytuje rozhraní pro trigger, z které je pak odvozená třída TriggerLCDInterface pro trigger obsahující displej. Tato třída má na starost vykonávání akcí způsobených triggerem. V případě, že trigger obsahuje displej, tak může být použita na vypsání důležitých informací na displeji, jako je IP adresa počítače na kterém běží.



Obrázek 7.6. Třída TriggerInterface s implementací.

7.6.1 Metoda Handler

Metoda Handler je typicky volána z třídy API, a stará se o volání metod na základě parametrů z HTTP požadavku.

7.6.2 Metoda IsEnabled

Metoda IsEnabled vrací hodnotu proměnné m_enabled, jestli je trigger povolený. Hodnota se mění po zavolání metod Enable a Disable.

7.6.3 Metoda CheckForTrigger

Abstraktní metoda CheckForTrigger běží v samostatném vláknu, kde typicky čte výstup z triggeru a reaguje na něj. Může se jednat o informaci, že došlo ke stisknutí tlačítka nebo požadavek na zaslaní IP adresy. V případě, že metoda IsEnabled vrátí hodnotu false, tak se metoda má ukončit.

7.6.4 Metoda TriggerAction

Metoda TriggerAction se stará o zapsání časového razítka a názvu triggeru z kterého byla zavolána do příslušného souboru, a případného spuštění zaznamenávání. Metodu TriggerAction zavolá buď metoda CheckForTrigger nebo API.

7.6.5 Metoda Enable

Metoda Enable se stará o spuštění vlákna s metodou CheckForTrigger a úpravu hodnoty proměnné m_enabled na true. To zajistí, že bude možné vyvolat akci triggeru přes zavolání metody TriggerAction. V případě třídy TriggerLCDInterface se spouští také vlákno vykonávající metodu KeepAlive pokud již nebylo spuštěno.

7.6.6 Metoda Disable

Metoda Disable upravuje hodnotu proměnné m_enabled na false a vyčká až se vlákno s metodou CheckForTrigger ukončí. To způsobí, že nebude možné vyvolat akci triggeru přes zavolání metody TriggerAction. Vlákno s metodou KeepAlive se neukončuje, jen by se měla změnit hodnota kterou odesílá.

7.6.7 Metoda HandShake

Metoda HandShake slouží pro detekci hardwarového triggeru. Otevře sériové zařízení, nastaví ho na rychlost 9600 baudů, a pošle bajt s hodnotou `0x05`, která symbolizuje dotaz na handshake. Pak očekává odpověď ve formě bajtu s hodnotou `0x06`. Pokud nedostane žádnou odpověď do 1 sekundy nebo dostane jinou odpověď, tak obnoví jeho původní nastavení. V případě úspěchu metoda vrací file descriptor na dané zařízení, jinak vrací hodnotu -1.

7.6.8 Metoda KeepAlive

Metoda KeepAlive periodicky informuje hardwarový trigger o tom, že spojení s řídicím počítačem je aktivní, a jestli je daný trigger povolený. V případě, že trigger je povolený tak odesílá bajt s hodnotou `0x06`, jinak odesílá hodnotu `0x15`. Trigger podle této hodnoty zobrazuje stav na displeji.

7.6.9 Metoda Print

Metoda Print dostane přes parametr string, který pošle hardwarovému triggeru k zobrazení na displeji. Typicky by se mělo jednat jen o zavolání funkce write na file descriptor triggeru.

7.6.10 Metoda PrintIPAddress

Metoda PrintIPAddress slouží pro vypsání IP adresy řídicího počítače na displeji hardwarového triggeru. Metoda interně využívá metodu Print. Název zařízení, kterého IP adresu vypíše je uložený v nastaveních, a je konfigurovatelný přes API.

7.7 Zaznamenávání CAN sběrnice

Pro naše potřeby bylo rozhodnuto vyvinout jednoduchý program na zachytávání CAN FD komunikace. Využívat budeme rozhraní SocketCAN pro zabezpečení kompatibility s jinými kartami. Program bude vypisovat na standardní výstup data ve formátu ASCII Logging Files (.asc). Chování programu bude následovné – otevře socket pro zadaný CAN port, do ukončení procesu bude číst získaná data, a vypisovat je v ASCII formátu. Všechny potřebné kroky budou popsány níže.

7.7.1 SocketCAN

Rozhraní SocketCAN používá Berkeley socket API a implementuje CAN zařízení jako síťové rozhraní. Pro otevření CAN portu se proto používají funkce socket, ioctl a bind. Následné načtení CAN rámce se realizuje přes funkci read. Funkce CanOpen sloužící na otevření CAN portu má následovní implementaci.

```
int CanOpen(const char* ifname) {
  int                 s;
  struct sockaddr_can addr;
  struct ifreq        ifr;

  if((s = socket(PF_CAN, SOCK_RAW, CAN_RAW)) < 0) {
    perror("opening failed");
    return -1;
  }

  strcpy(ifr.ifr_name, ifname);
  ioctl(s, SIOCGIFINDEX, &ifr);

  addr.can_family  = AF_CAN;
  addr.can_ifindex = ifr.ifr_ifindex;

  if(bind(s, (struct sockaddr*) &addr, sizeof(addr)) < 0) {
    perror("bind failed");
    return -2;
  }

  return s;
}
```

7.7.2 ASCII Logging Files formát

Získaná data potřebujeme zobrazit ve formátu ASCII Logging Files. Předpokládá se, že zaznamenávaná data budou otevírána v programu CANoe od společnosti Vector. Výstup začíná hlavičkou, která může vypadat následovně.

```
date Mon Apr 27 14:14:29.000 PM 2020
base hex  timestamps absolute
internal events logged
// version 10.0.0
Begin TriggerBlock Mon Apr 27 14:14:29.000 PM 2020
0.000000 Start of measurement
```

Pak následuje komunikace, jejíž výstup může být následovní.

```
39.4521543 1         31 Rx d 2 5B B0
39.8515138 1        4FF Rx d 8 FD 3B BF 0D 12 7E 6E 42
40.0516851 1        42B Rx d 7 66 09 D0 37 17 37 EA
40.6524292 1        2CC Rx d 8 BE D1 68 34 78 9B 6C 3C
40.8526551 1        74C Rx d 1 8D
```

Detailně popsaný řádek komunikace se nachází v tabulce 7.1.

Tabulka 7.1. Popis jednotlivého řádku ve formátu ASCII Logging Files.

Čas	Kanál	ID	Tx/Rx	Typ Framu	DLC	Data
39.4521543	1	31	Rx	d	2	5B B0

Během testování kompatibility vytvořených souborů s programem CANoe, bylo zjištěno několik detailů, které bylo potřeba opravit. První byl, že identifikační číslo kanálu musí být vyšší nebo rovno 1. První návrh programu totiž číslo kanálu určoval podle názvu zařízení, takže zařízení `can0` mělo nastavený kanál 0. To se vyřešilo inkrementací čísla kanálu o jedničku.

Jako další implementační detail, se ukázalo nastavovaní EFF flagu. Ten se nastavuje v 32. bitu ID zprávy. To způsobovalo, že se zobrazovaly zprávy s nesprávnými ID. Korektní řešení je, vypisovat vymaskované ID zprávy s CAN_EFF_MASK za pomocí logického součinu, a pak v případě, že je nastavený 32. bit v původním ID, co zjistíme za pomoci vymaskování ID zprávy a CAN_EFF_FLAG, vypíšeme za ID zprávy znak `x`.

7.8 API

Pro jednoduchost integrace do automatizovaných testů bylo zvoleno webové API přístupné přes HTTP GET požadavky. Odpovědi jsou ve formátu JSON, který byl vybrán kvůli jeho čitelnosti jak pro běžného člověka, tak i aplikaci.

Tabulka 7.2. Minimální API pro každou sběrnici.

Adresa	Popis
`/${DeviceType}/device_list`	Vypíše seznam zařízení
`/${DeviceType}/${DeviceName}/info`	Vypíše informace o zařízení
`/${DeviceType}/${DeviceName}/start`	Spustí záznam
`/${DeviceType}/${DeviceName}/stop`	Zastaví záznam
`/${DeviceType}/${DeviceName}/is_active`	Vráti, jestli je záznam aktivní
`/${DeviceType}/${DeviceName}/change_logfile`	Zaznamenávání do nového souboru
`/${DeviceType}/${DeviceName}/enable`	Povolí zařízení v platformě
`/${DeviceType}/${DeviceName}/disable`	Zakáže zařízení v platformě

7.8.1 Základní příkazy

API garantuje určitou základní množinu příkazů, které musí být implementovány. Ukázka takových adres je v tabulce 7.2. Většinu je potřeba implementovat na úrovni rozhraní dané sběrnice nebo modulu. Výjimkou jsou příkazy enable a disable, které už jsou implementovány na úrovni třídy TraceInterface, a nejsou virtuální.

7.8.2 Ukázka komunikace

V této sekci budou popsány základní příkazy API s ukázkou možných odpovědí.

Začneme požadavkem na adresu `/ethernet/device_list`, ten vrací seznam zařízení schopných zachytávat Ethernetovou komunikaci. Odpověď může vypadat následovně.

```
[
    "enp0s3",
    "enp0s29",
    "enp0s30"
]
```

Pokud chceme zjistit všechny informace o konkrétním zařízení, například o zařízení enp0s3 typu ethernet, tak pošleme požadavek na adresu `/ethernet/enp0s3/info`. Odpověď, obsahuje všechny potřebné informace pro daný typ zařízení.

```
{
    "device_name": "enp0s3",
    "enabled": true,
    "interface": "enp0s3",
    "type": "ethernet"
}
```

Pro spuštění zaznamenávání zařízení enp0s3, pošleme požadavek na adresu `/ethernet/enp0s3/start`. Odpověď obsahuje jen návratový kód a status.

```
{
    "return": "0",
    "status": "OK"
}
```

Pro zastavení zaznamenávání zařízení enp0s3, pošleme požadavek na adresu `/ethernet/enp0s3/stop`. Odpověď obsahuje kromě návratového kódu a statusu také název souboru, do kterého se doteď zapisovalo, a je připravený na další akce.

```
{
    "log_filename": "ethernet_enp0s3_20200408155344.pcapng",
    "return": "0",
    "status": "OK"
}
```

Pro zjištění stavu zaznamenávání zařízení enp0s3, pošleme požadavek na adresu `/ethernet/enp0s3/is_active`. Jsou jen dvě možné odpovědi, a to buď

```
{
    "return": "0",
    "status": "NOT_ACTIVE"
}
```

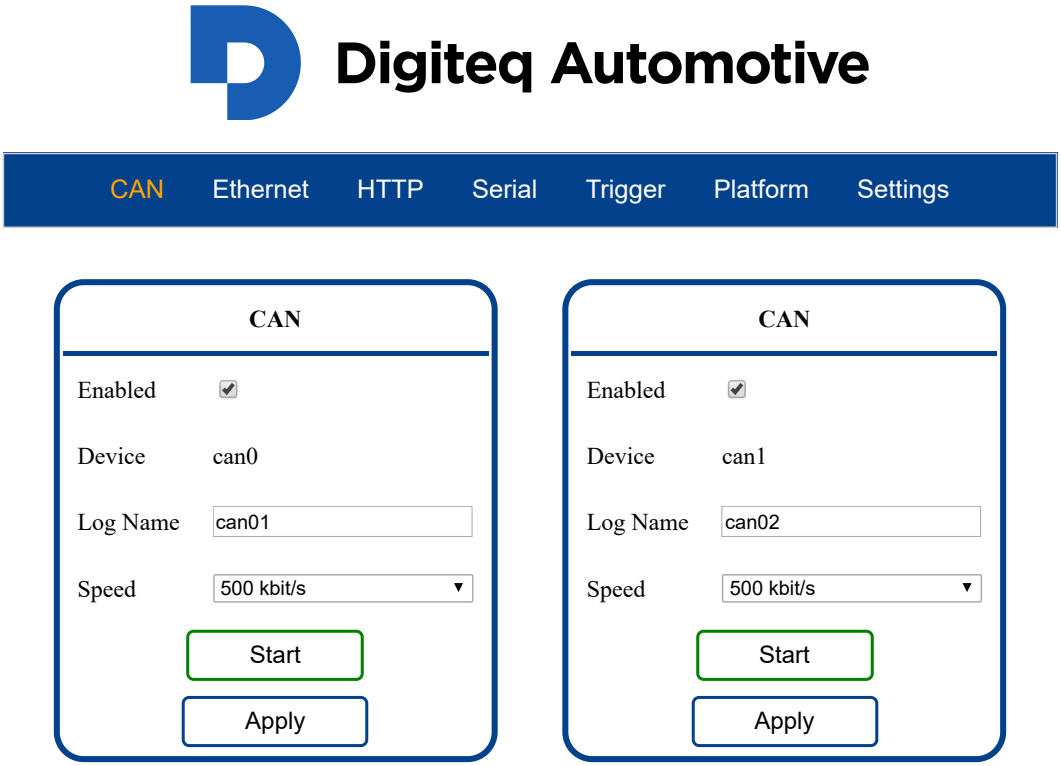
nebo

```
{
    "return": "1",
    "status": "ACTIVE"
}
```

Pro změnu cílového souboru pro zaznamenávání je potřeba udělat požadavek na adresu `/ethernet/enp0s3/change_logfile`, to způsobí, že se další komunikace bude zapisovat do nového souboru. Odpověď má stejný formát jako příkaz stop, jen s tím rozdílem, že se komunikace zachytává i nadále.

7.9 Webové rozhraní

Pro pohodlnější ovládání z pohledu uživatele bylo vyvinuto webové rozhraní. Všechny funkce dostupné přes API jsou ovladatelné přes webové rozhraní. Předpokládá se, že na řídicím počítači je spuštěný HTTP server, který bude obsluhovat tuto stránku. Webové rozhraní je naprogramováno v HTML, CSS a Javascriptu. Z toho plyne, že se server zatěžuje, jenom režií potřebnou pro posílaní potřebných souborů.



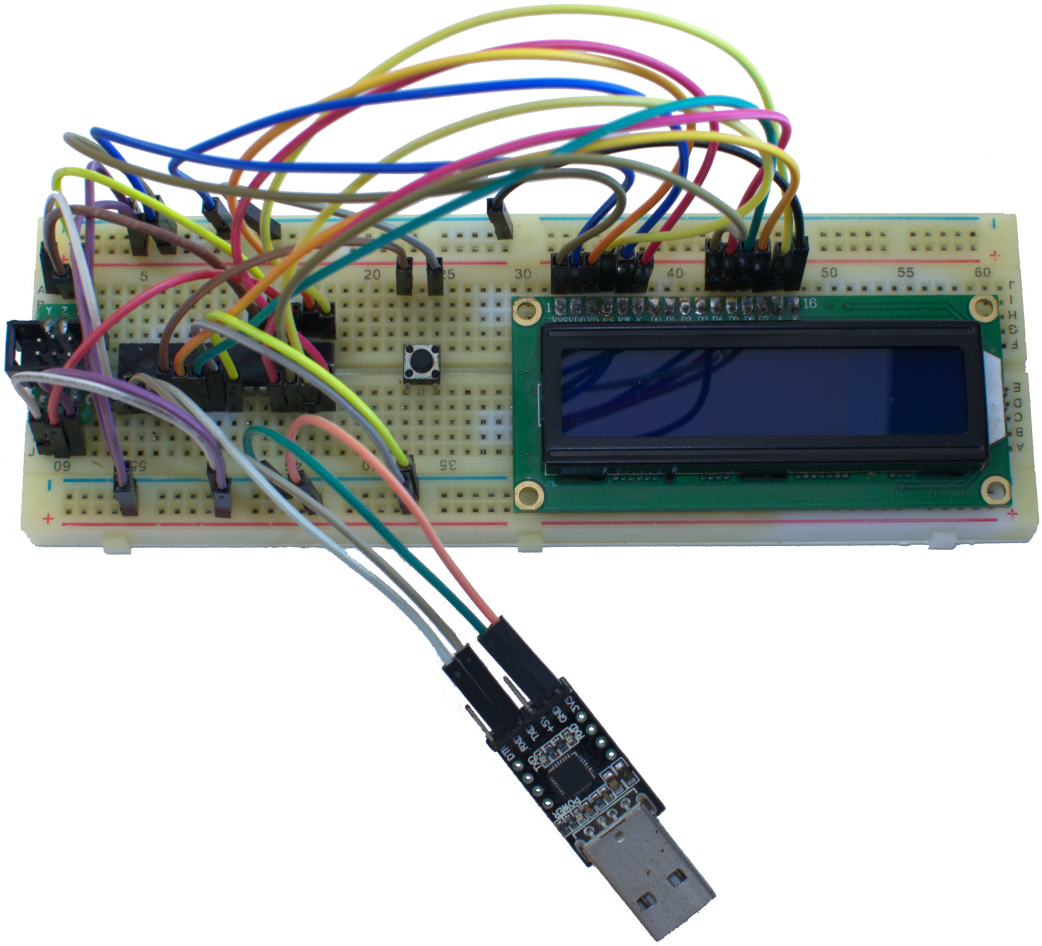
Obrázek 7.7. Webové rozhraní.

Kapitola 8
Realizace hardwarového triggeru

Tato kapitola popisuje potřebné kroky pro realizaci hardwarového triggeru. Jak po stránce hardwaru, tak i softwaru. Na začátku se sestrojil prototyp ověřující správnost schématu zapojení. Když bylo testování dokončené, tak se začal vyvíjet firmware. Pak se na závěr podle schématu vytvořila DPS a krabička na 3D tiskárně.

8.1 Prototyp

Když byl návrh schématu hotový, tak byl sestrojen prototyp na nepájivém poli pro ověření funkčnosti. Jediný rozdíl proti konečnému řešení bylo použití jiného tlačítka z důvodu osazení na nepájivé pole, co ale nemělo vliv na funkčnost.



Obrázek 8.1. Prototyp na nepájivém poli.

8.2 Firmware

Firmware triggeru byl vyvíjen v programu Arduino IDE. Nahrávání firmwaru je realizováno pomocí programátoru USBasp [36] přes AVR ISP 6 pin konektor. Během vývoje firmwaru bylo potřeba vyřešit dílčí problémy jako odstranění zákmitů a vypisování textu na displeji přijatého přes sériovou linky.

8.2.1 Odstranění zákmitů tlačítka

Jelikož docházelo ke zákmitům při stisknutí tlačítka, tak to bylo nutné vyřešit, aby se pro každé stisknutí tlačítka odesílala informace jen jednou. Povedlo se to bez dalšího hardwaru, a to aktivováním pullup rezistoru na příslušném pinu a částí firmwaru.

Při změně stavu pinu, na který je připojené tlačítko se vyvolá přerušení. Ve funkci obsluhující přerušení se uloží stav pinu a čas, kdy ke změně došlo. Periodicky se volá funkce debounce, která kontroluje jestli nedošlo ke změně a neuběhl už časový limit. V takovém případě ještě zkontroluje, jestli aktuální hodnota pinu tlačítka je HIGH, a v tom případě zavolá funkci button_action. Na závěr už jen zapíše hodnotu proměnné current_button_state do proměnné last_button_state.

```
#define BUTTON_PORT    2
#define DEBOUNCE_DELAY 50

unsigned int           last_button_state    = LOW;
volatile unsigned long last_debounce_time   = 0;
volatile int           current_button_state = LOW;

void button_change_isr() {
  current_button_state = digitalRead(BUTTON_PORT);
  last_debounce_time   = millis();
}

void debounce() {
  if(current_button_state != last_button_state
     && (millis() - last_debounce_time) > DEBOUNCE_DELAY) {
    if(current_button_state == HIGH) {
      button_action();
    }

    last_button_state = current_button_state;
  }
}
```

8.2.2 Vypsání IP adresy na displeji

Hardwarový trigger všechno, co přijme přes sériovou linku a co není kontrolní znak, uloží do bufferu o délce počtu znaků displeje v řádku. Pak v případě, že přijme znak `\n`, tak vypíše obsah bufferu na displej a zbytek řádku zmaže. Pokud přijme víc znaků, než je délka bufferu, tak přebytečné znaky neukládá do bufferu, ale zahazuje je.

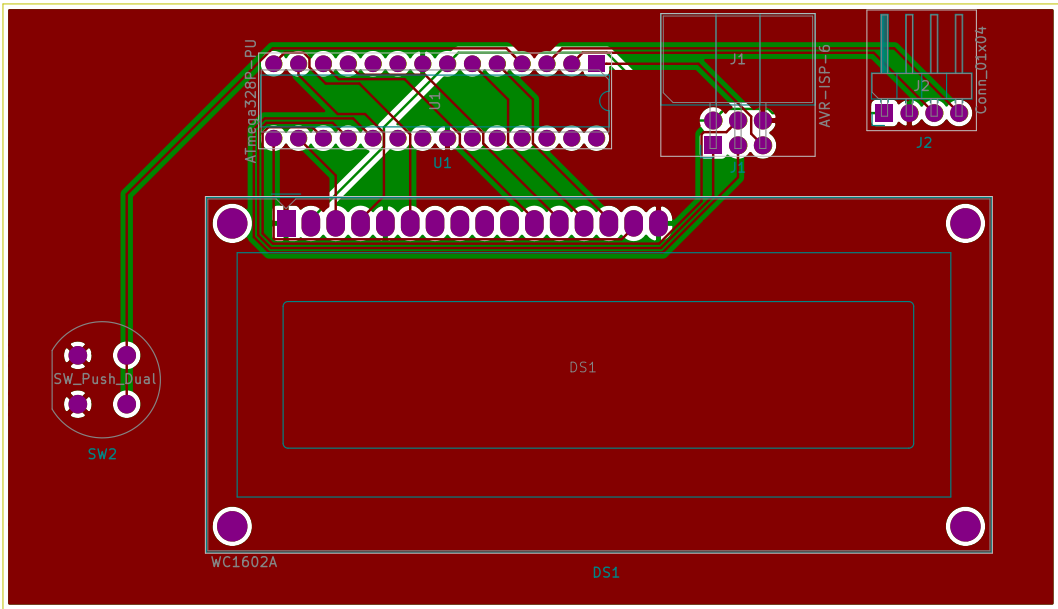
8.2.3 Vypsání stavu připojení na displeji

Hardwarový trigger může být v jedním ze třech stavů: connected, disconnected nebo disabled. Přechod mezi jednotlivými stavy je realizován přes keepalive bajty nebo časovým úsekem po který nepřišel žádný keepalive bajt.

Do stavu connected se dostane, když přijme bajt s hodnotou `0x06` s periodou jedné sekundy, to znamená, že řídící počítač je připojený a přijme případné triggery. Stav disabled nastane, když přijme bajt s hodnotou `0x15`, s periodou jedné sekundy, to znamená, že řídící počítač je připojený, ale nepřijme případné triggery. Stav disconnected nastává, když po dobu 3 sekund nepřijme žádný keepalive bajt.

8.3 Návrh DPS

Návrh počítá se dvěma vodivými vrstvami ve výsledné DPS. Na začátku se navrhlo rozložení součástek pro DPS. Pak se přidaly díry do DPS pro upevnění krabičky, které se nachází přímo pod dírami, které jsou součástí LCD displeje. Jako další krok bylo rozlití měděné plochy na obě vodivé vrstvy. Na vrchní vrstvě se potahaly všechny spoje kromě napájení, které bylo na dolní vrstvě.



Obrázek 8.2. DPS.

8.4 Výroba 3D modelu krabičky

Model byl vytvářený v programu OpenSCAD [37]. Pro přehlednost je každá součástka v samostatném souboru jako modul. Krabička se skládá ze třech částí - horní díl, spodní díl a krytka na tlačítko. Díly byly navrženy, aby byla potřeba co nejméně podpůrného materiálu při tisku.

Po vyrendrování byly vyexportovány stl soubory, které se importovaly do sliceru. Jako slicer byl použitý PrusaSlicer [38]. Byl zvolen kvůli jeho jednoduchosti používání, ale také velkému množství možností konfigurace tisku. Všechny díly se tiskly zvlášť, aby v případě nějaké chyby během tisku bylo co nejméně odpadu. Pro tisk byl použit materiál PLA.

Obrázek 8.3. Hotový hardwarový trigger.

Kapitola 9
Testování

Testování probíhalo z webového rozhraní, protože se simulovalo chování manuálního testera. Tím se najednou otestovalo webové rozhraní a API. Ze začátku se testovalo zaznamenávání jednotlivých sběrnic, pak se testovaly celé platformy a nakonec se ověřovala funkčnost hardwarového triggeru.

9.1 Zaznamenávání jednotlivých sběrnic

Test ověřující funkčnost zaznamenávání každé sběrnice samostatně. Očekává se připojení nějakého zařízení schopného vysílat komunikaci na dané sběrnici.

9.1.1 Průběh testu

Na začátku se nastaví potřebné parametry zařízení, které budou zaznamenávat a potvrdí se stisknutím tlačítka Apply. Zaznamenávání začne po stisknutí tlačítka Start. Po přibližně minutě se ukončí zaznamenávání stisknutím tlačítka Stop. Pak se stáhne záznam přes odkaz s názvem Download, a zkontroluje se jeho obsah. Tento postup se opakuje pro každou sběrnici.

9.1.2 Výsledek testu

Všechny sběrnice se povedlo úspěšně otestovat.

9.2 Tvorba a načtení platformy

Test sloužící na ověření funkčnosti tvorby a načtení platformy. Uživatel vytvoří platformu, kterou pak při dalším spuštění aplikace načte.

9.2.1 Průběh testu

Zvolí se zařízení, které má platforma obsahovat. Ostatní zařízení se zakážou. Nastaví se potřebné parametry zvoleným zařízením, a potvrdí se stisknutím tlačítka Apply. Kliknutím na položku Platform v menu se přejde na část konfigurující platformy. Tam se klikne na tlačítko Save As, a zvolí jméno nové platformy. Aplikace se ukončí, a znova spustí. Po spuštění se načte uložená platforma přes položku Platform v menu. Zvolením platformy ze seznamu, a kliknutím na tlačítko Load se platforma načte. Ověří se správnost načtených parametrů.

9.2.2 Výsledek testu

Test dopadl úspěšně. Nastavení platformy se po restartu aplikace korektně načetly.

9.3 Zaznamenávání platformy

Test ověřující funkčnost zaznamenávání celé platformy. Předpokládá se, že platforma už byla vytvořená a byla načtená.

9.3.1 Průběh testu

Na začátku se přejde na část konfigurující platformy kliknutím na položku Platform v menu. Zaznamenávání se spustí tlačítkem Start. Po přibližně minutě se ukončí zaznamenávání stisknutím tlačítka Stop. Pak se stáhne záznam přes odkaz s názvem Download, a zkontroluje se jeho obsah.

9.3.2 Výsledek testu

Test skončil úspěchem. Zaznamenávání se spustilo korektně na všech zařízeních platformy.

9.4 Spuštění zaznamenávání pomocí triggeru

Ověření funkcionality spuštění zaznamenávání pomocí triggeru. Test bude probíhat s hardwarovým triggerem, kde se otestuje zároveň funkčnost aplikace a hardwarového triggeru.

9.4.1 Průběh testu

Hardwarový trigger se připojí k řídicímu počítači a spustí se aplikace. Na hardwarovém triggeru se stiskne tlačítko. Ve webovém rozhraní se přejde na část konfigurující platformy kliknutím na položku Platform v menu. Zkontroluje se, že je aktivní zaznamenávání pomocí zobrazeného tlačítka Stop, a pak se stiskne pro ukončení zaznamenávání.

9.4.2 Výsledek testu

Tento test také proběhl korektně. Po stisknutí tlačítka na hardwarovém triggeru se spustilo zaznamenávání platformy.

9.5 Vytváření časového razítka pomocí triggeru

Test ověřující Vytváření časového razítka pomocí triggeru. Testování bude probíhat s hardwarovým triggerem.

9.5.1 Průběh testu

Hardwarový trigger se připojí k řídicímu počítači, a spustí se aplikace. Přes webové rozhraní se spustí zaznamenávání platformy. Na hardwarovém triggeru se desetkrát stiskne tlačítko. Zastaví se zaznamenávání, a pak se stáhne záznam časových značek triggerů přes odkaz s názvem Download. Zkontroluje se jestli jsou zachyceny všechny stisknutí tlačítka hardwarového triggeru.

9.5.2 Výsledek testu

Test dopadl úspěchem. Všech deset stisknutí tlačítka na hardwarovém triggeru bylo zaznamenáváno časovým razítkem v souboru.

Kapitola 10
Závěr

Všechny vytyčené cíle se podařilo splnit. Aplikace je schopná zaznamenávat všechny požadované typy sběrnic, a je připravená na další rozšíření podporovaných typů sběrnic. Součástí je webové API, které je realizováno přes HTTP GET požadavky, které odpovídá ve formátu JSON. Pro vylepšení uživatelského zážitku byla implementována podpora konfigurace platforem, které se rovněž ukládají ve formátu JSON na disk.

Další z cílů bylo navržení a realizace hardwarového triggeru pro spuštění zaznamenávání a vytváření časových značek. To bylo dosaženo pomocí mikroprocesoru ATmega328P. Oproti zadání byl navíc přidaný LCD displej pro zobrazovaní IP adresy řídicího počítače pro zvýšení uživatelského komfortu, v případě, že k počítači není připojený monitor a klávesnice. Pro tento obvod byla navržena DPS a krabička vytisknuta na 3D tiskárně.

Testování na začátku probíhalo pro každou sběrnicí zvlášť. Pak se testovaly platformy, kde bylo otestováno větší množství sběrnic najednou. Nakonec se testoval hardwarový trigger. Všechny testy dopadly úspěšně.

Jako vedlejší produkt práce vznikl modulární framework pro zaznamenávání sběrnic pomocí kterého byla aplikace implementována. Díky němu bude případné rozšíření o podporu další sběrnice jednoduché. Pro potřeby manuálního testování bylo naprogramováno webové rozhraní aplikace, které zprostředkovává všechny funkce webového API.

Literatura

[1] *blue PiraT Rapid* [online]. MAGNA Telemotive GmbH [cit. 26. 3. 2020]. Dostupné na
https://www.telemotive.de/en/products/hardware/blue-pirat-rapid/.

[2] Kevser Bekmezci. *AW: Price of BLUEPIRAT Rapid* [elektronická pošta]. Zasláno z:
kevser.bekmezci@magna.com. 6. 5. 2020 13:45 [cit. 6. 5. 2020]. Osobní komunikace

[3] *Data Logger XORAYA N4000 — X2E GmbH* [online]. X2E GmbH [cit. 26. 3. 2020]. Dostupné na
https://x2e.de/en/products/data-logger-xoraya-n4000.

[4] Matthias Müller. *Re: Price of DATA LOGGER XORAYA N4000* [elektronická pošta]. Zasláno z:
matthias.mueller@x2e.de. 27. 3. 2020 10:01 [cit. 27. 3. 2020]. Osobní komunikace

[5] *Intrepid Control Systems RAD-Gigalog - Intrepid Control Systems, Inc.* [online]. Intrepid Control Systems Inc. [cit. 26. 3. 2020]. Dostupné na
https://www.intrepidcs.com/products/data-loggers/rad-gigalog/.

[6] Fabio Leite. *Re: Intrepid Control Systems - Solution Proposal* [elektronická pošta]. Zasláno z:
fleite@intrepidcs.com. 30. 3. 2020 18:21 [cit. 30. 3. 2020]. Osobní komunikace

[7] *SocketCAN - Controller Area Network — The Linux Kernel documentation* [online]. The kernel development community [cit. 26. 3. 2020]. Dostupné na
https://www.kernel.org/doc/html/latest/networking/can.html.

[8] *PCAN-PCI Express FD: PEAK-System* [online]. PEAK-System Technik GmbH [cit. 26. 3. 2020]. Dostupné na
https://www.peak-system.com/PCAN-PCI-Express-FD.414.0.html?&L=1.

[9] *CAN-IB600/PCIe* [online]. HMS Industrial Networks AB [cit. 26. 3. 2020]. Dostupné na
https://www.ixxat.com/products/products-industrial/can-interfaces/pcie-can-interfaces/can-ib600-pcie.

[10] Ing. Jakub Samek. *Re: FW: Digiteq Automotive s.r.o: Price of CAN-IB600/PCIe* [elektronická pošta]. Zasláno z:
samek@fccps.cz. 30. 3. 2020 15:50 [cit. 30. 3. 2020]. Osobní komunikace

[11] *Intrepid Control Systems RAD-Moon 2 - Intrepid Control Systems, Inc.* [online]. Intrepid Control Systems Inc. [cit. 26. 3. 2020]. Dostupné na
https://www.intrepidcs.com/products/automotive-ethernet-tools/rad-moon2/.

[12] Monika Heidrich. *AW: Price of RAD-MOON 2 Media Converter for 100/1000BASE-T1* [elektronická pošta]. Zasláno z: mheidrich@intrepidcs.com. 9. 3. 2020 17:04 [cit. 9. 3. 2020]. Osobní komunikace

[13] *Technica Engineering - 1000BASE-T1 MediaConverter* [online]. Technica Engineering GmbH [cit. 26. 3. 2020]. Dostupné na https://technica-engineering.de/en/produkt/mediaconverter/.

[14] Order Team Technica Engineering GmbH. *AW: Price of 1000BASE-T1 MediaConverter* [elektronická pošta]. Zasláno z: order@technica-engineering.de. 10. 3. 2020 09:56 [cit. 10. 3. 2020]. Osobní komunikace

[15] *Automotive Ethernet Converter — X2E GmbH* [online]. X2E GmbH [cit. 26. 3. 2020]. Dostupné na https://x2e.de/en/products/automotive-ethernet-converter.

[16] Matthias Müller. *Re: Price of AUTOMOTIVE ETHERNET CONVERTER* [elektronická pošta]. Zasláno z: matthias.mueller@x2e.de. 24. 3. 2020 10:44 [cit. 24. 3. 2020]. Osobní komunikace

[17] *EasyCON 1000Base-T1 Media Converter* [online]. GÖPEL electronic GmbH [cit. 26. 3. 2020]. Dostupné na https://www.goepel.com/fileadmin/pdf/ats/en/ATS_EasyCON_1000Base-T1.pdf.

[18] Rico Heine. *WG: Price of EasyCON 1000Base-T1 Media Connverter* [elektronická pošta]. Zasláno z: r.heine@goepel.com. 10. 3. 2020 09:09 [cit. 10. 3. 2020]. Osobní komunikace

[19] *Arduino - Home* [online]. Arduino AG [cit. 26. 3. 2020]. Dostupné na https://www.arduino.cc/.

[20] *ATmega328P - 8-bit AVR Microcontrollers* [online]. Microchip Technology Inc. [cit. 26. 3. 2020]. Dostupné na https://www.microchip.com/wwwproducts/en/ATmega328P.

[21] *Mikrokontrolér AVR, EEPROM: 1kB, SRAM: 2kB, flash: 32kB, THT ATMEGA328P-PU SDIP28 — GM electronic, spol. s.r.o.* [online]. GM electronic, spol. s.r.o. [cit. 14. 5. 2020]. Dostupné na https://www.gme.cz/atmega328p-pu-sdip28-atmel.

[22] *STM32F103C8 - Mainstream Performance line, ARM Cortex-M3 MCU with 64 Kbytes Flash, 72 MHz CPU, motor control, USB and CAN - STMicroelectronics* [online]. STMicroelectronics International N.V. [cit. 26. 3. 2020]. Dostupné na https://www.st.com/en/microcontrollers-microprocessors/stm32f103c8.html.

[23] *Kontroler STM32F103C8T6 ARM STM32 — GM electronic, spol. s.r.o.* [online]. GM electronic, spol. s.r.o. [cit. 14. 5. 2020]. Dostupné na https://www.gme.cz/kontroler-stm32f103c8t6-arm-stm32.

[24] *STM32 základní vývojová deska s procesorem Cortex-M3, STM32F103C8T6 - ESP32* [online]. FZG Studio s.r.o. [cit. 26. 3. 2020]. Dostupné na https://www.esp32.cz/cortex-stm32/stm32-zakladni-vyvojova-deska-s-procesorem-cortex-m3--stm32f103c8t6/.

[25] *GitHub - yhirose/cpp-httplib: A C++ header-only HTTP/HTTPS server and client library* [online]. Yuji Hirose [cit. 26. 3. 2020]. Dostupné na `https://github.com/yhirose/cpp-httplib`.

[26] *GitHub - etr/libhttpserver: C++ library for creating an embedded Rest HTTP server (and more)* [online]. Sebastiano Merlino [cit. 26. 3. 2020]. Dostupné na `https://github.com/etr/libhttpserver`.

[27] *Libmicrohttpd - GNU Project - Free Software Foundation* [online]. Free Software Foundation, Inc. [cit. 26. 3. 2020]. Dostupné na `https://www.gnu.org/software/libmicrohttpd/`.

[28] *GitHub - Tencent/rapidjson: A fast JSON parser/generator for C++ with both SAX/DOM style API* [online]. Tencent Holdings Ltd. [cit. 26. 3. 2020]. Dostupné na `https://github.com/Tencent/rapidjson/`.

[29] *Debian – Details of package rapidjson-dev in buster* [online]. Debian Webmaster, webmaster@debian.org [cit. 26. 3. 2020]. Dostupné na `https://packages.debian.org/stable/rapidjson-dev`.

[30] *GitHub - nlohmann/json: JSON for Modern C++* [online]. Niels Lohmann [cit. 26. 3. 2020]. Dostupné na `https://github.com/nlohmann/json`.

[31] *Debian – Details of package nlohmann-json3-dev in buster* [online]. Debian Webmaster, webmaster@debian.org [cit. 26. 3. 2020]. Dostupné na `https://packages.debian.org/stable/nlohmann-json3-dev`.

[32] *Wireshark Packet Capture: Tshark Vs. Dumpcap* [online]. Tony Fortunato [cit. 26. 3. 2020]. Dostupné na `https://www.networkcomputing.com/networking/wireshark-packet-capture-tshark-vs-dumpcap`.

[33] *KiCad EDA* [online]. KiCad Developers Team [cit. 26. 3. 2020]. Dostupné na `https://www.kicad-pcb.org/`.

[34] *Arduino - LiquidCrystal* [online]. Arduino AG [cit. 26. 3. 2020]. Dostupné na `https://www.arduino.cc/en/Reference/LiquidCrystal`.

[35] *Arduino Reference* [online]. Arduino AG [cit. 26. 3. 2020]. Dostupné na `https://www.arduino.cc/reference/en/language/functions/communication/serial/`.

[36] *USBasp - USB programmer for Atmel AVR controllers - fischl.de* [online]. Thomas Fischl [cit. 26. 3. 2020]. Dostupné na `https://www.fischl.de/usbasp/`.

[37] *OpenSCAD - The Programmers Solid 3D CAD Modellerde* [online]. Marius Kintel [cit. 26. 3. 2020]. Dostupné na `https://www.openscad.org/`.

[38] *PrusaSlicer - Prusa Research a.s.. – 3D tisk a 3D tiskárny od Josefa Průši* [online]. Prusa Research a.s. [cit. 17. 5. 2020]. Dostupné na `https://www.prusa3d.cz/prusaslicer/`.

[39] FIŠEROVÁ, Kristina. *Grafický manuál identity Českého vysokého učení technického v Praze* [online]. [cit. 26. 3. 2020]. Dostupné na `https://www.cvut.cz/logo-a-graficky-manual`.

Příloha A
Zkratky a symboly

A.1 Zkratky

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CAN	Controller Area Network
CAN FD	Controller Area Network Flexible Data-Rate
CSS	Cascading Style Sheets
DLC	Data Length Code
DPS	Deska Plošných Spojů
EFF	Extended Frame Format
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ID	Identifier
IDE	Integrated Development Environment
IP	Internet Protocol
ISP	In System Programming
JSON	JavaScript Object Notation
LCD	Liquid Crystal Display
LIN	Local Interconnect Network
PID	Process Identifier
PLA	Polylactic acid
Rx	Receive
TCP	Transmission Control Protocol
Tx	Transmit
UDP	User Datagram Protocol
URL	Uniform Resource Locator
USB	Universal Serial Bus
XML	Extensible Markup Language

A.2 Obsah přiloženého USB flash disku

`ctustyle2.tex`	TEXová makra implementující šablonu ve verzi 2.
`ctulogo-new.pdf`	Logo ČVUT podle[39] v modré variantě.
`technika.tfm`, `Technika-Regular.pfb`	Metrika a kresby písma **Technika-Regular**.
`technika-it.tfm`, `Technika-Italic.pfb`	Metrika a kresby písma ***Technika-Italic***.
`technika-bk.tfm`, `Technika-Book.pfb`	Metrika a kresby písma Technika-Book.
`technika-bkit.tfm`, `Technika-BookItalic.pfb`	Metrika a kresby písma *Technika-BookItalic*.
`technika-bf.tfm`, `Technika-Bold.pfb`	Metrika a kresby písma **Technika-Bold**.
`technika-bi.tfm`, `Technika-BoldItalic.pfb`	Metrika a kresby písma ***Technika-BoldItalic***.
`xl2tech.enc`	Kódovací soubor speciálně pro rodinu Technika, protože toto písmo má obskurní názvy pro některé ligatury.
`bp-oleksmat.tex`	Hlavní zdrojový soubor tohoto dokumentu.
`images`	Složka obsahující obrázky použité v dokumentu.
`mybase.bib`	Údaje použité pro generování seznamu literatury.
`opmac-bib-iso690.tex`	Bibliografický styl iso690 používaný v balíčku opmac-bib.tex.
`src`	Složka obsahující zdrojové kódy.
`bp-oleksmat.pdf`	Tento dokument.