VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

DEPARTMENT OF CONTROL AND INSTRUMENTATION

OBSLUŽNÁ APLIKACE PRO SYSTÉM VÝSTUPNÍ KONTROLY ELEKTRONIKY

APPLICATION FOR QUALITY CHECK SYSTEM OF A DEVICE

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — Petr Tlustoš

AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. Jakub Arm, Ph.D.

SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Student: Petr Tlustoš ***ID:*** 240458

Ročník: 3 ***Akademický rok:*** 2023/24

NÁZEV TÉMATU:

Obslužná aplikace pro systém výstupní kontroly elektroniky

POKYNY PRO VYPRACOVÁNÍ:

Úkolem je vytvořit PC aplikaci, která bude obsluhovat testovací zařízení pro výstupní kontrolu elektroniky. Aplikace bude komunikovat s cloud službou obsahující data o typech kontrolovaných zařízeních, firmware, přičemž po provedení kontroly nahraje reporty. Aplikace bude komunikovat s testovacím zařízením pomocí USB, přičemž bude i nahrávat firmware. S testovaným zařízením bude komunikovat pomocí Bluetooth pro nastavení konfigurace.

1. Navrhněte testovací scénáře a workflow procesů.
2. Definujte požadavky na aplikaci.
3. Navrhněte architekturu aplikace.
4. Vytvořte aplikaci.
5. Otestujte funkčnost a proveďte validaci aplikace.

DOPORUČENÁ LITERATURA:

ČADA, O. Objektové programování. Grada, 2009. 200 s. ISBN: 978-80-247-6699-7.

Termín zadání: 5.2.2024 ***Termín odevzdání:*** 22.5.2024

Vedoucí práce: doc. Ing. Jakub Arm, Ph.D.

Ing. Miroslav Jirgl, Ph.D.
předseda rady studijního programu

ABSTRAKT

Tato bakalářská práce se zaměřuje na vývoj obslužné aplikace pro systém výstupní kontroly elektroniky, což je zásadní pro zajištění spolehlivosti elektronických zařízení a snížení zbytečných nákladů. Aplikace umožňuje testování desek plošných spojů (DPS) pomocí testovací stanice. Zajišťuje komunikaci s testovacím hardwarem přes USB a Bluetooth. Práce také řeší možnost nahrávání firmware zařízení a využití REST API pro komunikaci s databázovou službou. Obsahuje návrh testovacích scénářů, definici požadavků a následnou realizaci aplikace

KLÍČOVÁ SLOVA

Testovací scénář, obslužná aplikace, testování DPS, Flet, REST API, Python, Visual Studio Code, automatizace testování

ABSTRACT

This bachelor's thesis focuses on the development of an application for a quality control system of electronic devices, which is crucial for ensuring the reliability of electronic devices and reducing unnecessary costs. The application will provide testing of printed circuit boards (PCBs) using a testing station. It facilitates communication with the test hardware via USB or Bluetooth. The thesis also addresses the capability of uploading and modifying the device firmware and the use of REST API for communication with a cloud database service. It covers the design of test scenarios, definition of requirements and subsequent implementation of the application.

KEYWORDS

Testing scenario, Application for quality check device, Flet, REST API, Python, Visual Studio Code, test automation

Vysázeno pomocí balíčku `thesis` verze 4.09; `https://latex.fekt.vut.cz/`

TLUSTOŠ, Petr. *Obslužná aplikace pro výstupní kontrolu elektroniky*. Bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2024. Vedoucí práce: Doc.Ing. Jakub Arm, Phd.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Petr Tlustoš
VUT ID autora:	240458
Typ práce:	Bakalářská práce
Akademický rok:	2023/24
Téma závěrečné práce:	Obslužná aplikace pro výstupní kontrolu elektroniky

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu doc. Ing. Jakubovi Armovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Úvod

V dnešní době, kdy elektronika hraje zásadní roli ve většině aspektů našich životů, je kritické zajistit, že elektronická zařízení jsou spolehlivá a bezpečná. Jedním z klíčových prvků pro zajištění kvality a bezpečnosti elektronických komponent je výstupní kontrola elektroniky, která se provádí pomocí testovacích zařízení. S rostoucím počtem různých druhů elektronických zařízení se stává stále důležitějším, mít efektivní nástroje pro tuto kontrolu. Tímto způsobem se tak minimalizuje riziko selhání a zvyšuje se bezpečnost používání těchto zařízení.

Tato práce se zabývá návrhem a vývojem obslužné aplikace pro systém výstupní kontroly elektroniky. Cílem je vytvořit univerzální nástroj, který bude schopen automatizovaně testovat různé typy desek plošných spojů (DPS) za pomoci výstupní stanice výroby. Díky automatizovanému testovaní se celý proces zefektivní a zároveň se tak sníží riziko lidského faktoru, kde díky rostoucí komplexnosti zařízení, je již manuální kontrola často nedostatečná.

Samotná aplikace bude s testovací stanicí komunikovat prostřednictvím komunikace USB, kde bude provádět testy jednotlivých periférií testované desky a nahrávat konfiguraci (*firmware*). Při návrhu a následné implementaci bude zohledněna možnost budoucího rozšíření systému a případných změn v logice komunikací. Důležitou součástí práce bude návrh a implementace testovacích scénářů, které budou ověřovat specifické vlastnosti desky. Uživatel pak bude mít možnost nadefinovat, které periferie se budou testovat.

Pro účely vyskladnění bude možné nastavit konfiguraci testovaného zařízení pomocí Bluethooth. Firmware, včetně všech parametrů potřebných pro testování a pro proces vyskladnění bude uchováván v databázové službě. Po dokončení testování příslušného zařízení se vygeneruje závěrečná zpráva, která bude, pro účely budoucího zpracování, také nahrána do databáze. Tento centralizovaný přístup, umožní tak uživateli nastavovat a upravovat jednotlivé parametry bez nutnosti zásahu do implementace aplikace.

Cíle práce

Cílem této bakalářské práce je vytvoření návrhu obslužné aplikace pro výstupní kontrolu elektroniky pro výstupní stanici výroby. Samotný návrh aplikace byl vytvářen pro zařízení, které již bylo navrhnuto a realizováno. V rámci návrhu byly nadefinovány požadavky na aplikaci:

- Zajištění univerzálnosti - možnosti využiti pro testovaných více druhů desek plošných spojů
- Možnost otestovat a následně vyhodnotit stav testované desky
- Zajištění vytvoření výsledné zprávy testu
- Stažení firmware z databázové služby a schopnost aplikace nahrát firmware do příslušné desky plošných spojů.
- Navrhnutí a vytvoření testovacího scénáře
- Komunikace s databázovou službou za pomocí nástroje REST API
- Uložení výsledné zprávy testu do databáze
- Zajištění funkce tzv.vyskladnění (tj. uvedení již otestované desky pro vývoz)
- Navržení architektury aplikace pro budoucí vývoj (např. možnost využití na různých platformách)

1 Teoretická část

Následující podkapitoly shrnují základní pojmy a definice využívané v této práci a shrnutí základních znalostí pro hlubší pochopení problematiky práce.
První podkapitola definuje pojem deska plošných spojů.
Druhá podkapitola definuje proč se desky plošných spojů testují a jakým způsobem se provádí testovaní desek popisováno v této práci.
Třetí a podkapitola se zaobírá samotným principem testovacích scénářů a jejich následným návrhem. Pátá a poslední kapitola teoretické části pojednává o různých typech komunikací využívané v této práci.

1.1 Deska plošných spojů

Deska plošných spojů (DPS), známá také jako tištěný spoj, je klíčovým prvkem moderní elektroniky, který propojuje různé komponenty pomocí měděných cest.

DPS je vyrobena z izolačního materiálu, běžně sklolaminátu, známého pro svou pevnost, odolnost a izolační vlastnosti. Na tento substrát je nanesena vrstva mědi, která vytváří cesty pro elektrický proud mezi komponenty, buď leptáním, nebo potiskem.

1.2 Testování DPS

Při návrhu a vývoji DPS se vždy dbá na jeden klíčový faktor. Tím je, aby desky plošných spojů, jako základní kamen pro mnoho elektronických zařízení, byly funkční a spolehlivé.
Dříve kdy byly desky poměrně jednoduché, tak stačilo pro označení vadného produktu pouze manuální vizuální kontroly (MVI). Tato metoda samozřejmě skýtá určité nedostatky. Lidé se brzy unaví, chyby se přehlížejí a vadné desky nejsou zachyceny až do pozdějších fází procesu, kdy je jejich oprava dražší nebo dokonce nemožná.
Při snaze odstranit chybu lidského faktoru bylo využití automatické optické inspekce (AOI). Metoda se využívá při samotném výrobním procesu desky. AOI kontroluje zdali jsou všechny komponenty na správném místě či zdali deska neobsahuje špatný spoj.
Příchodem technologie povrchové montáže (SMT), která zavedla menší komponenty, inovativní balení čipů a zvýšila komplexnost jednostranných i vícevrstvých desek. S rostoucí hustotou komponent na deskách se metoda AOI potýká s obtížemi při rozpoznávání všech pájecích spojů a některé komponenty se tak stávají pro prakticky neviditelnými. Řešením se staly automatizované rentgenové inspekční systémy

(AXI). AXI dokáže "prohlédnout"skrze různé vrstvy a zjistit stav spojů pod nimi.

Pro bližší ukázaní proč je testování desek plošných spojů tak důležité, je v následující tabulce č 1.1 znázorněna míra výskytu běžných vad u PCB:

VADA	PODÍL	TYP	SOUVISÍ S PÁJENÍM
Otevřený obvod	25%	Kostrukční	Ano
Chybějící vodivá cesta	18%	Kostrukční	Ano
Zkrat	13%	Kostrukční	Ano
Chybějící elektrická součástka	12%	Kostrukční	Ne
Špatné zarovnání	8%	Kostrukční	Ano
Vadná elektrická součástka	8%	Elektrický	Ne
Nesprávná součástka	5%	Elektrický	Ne
Nesprávná orientace	2%	Elektrický	Ne

Tab. 1.1: Přehled vad a jejich charakteristik - převzato a upraveno z [3]

Jakmile deska plošných spojů byla vyrobena a tím i prošla všechny předešlé testy je na řadě otestováni celkové funkcionality desky. Ač existuje spoustu metod, tak mezi nejběžnějšími metodami se řadí testování uvnitř obvodu, známé pod anglickou zkratkou ICT (In-Circuit Testing)

Testování v obvodu (ICT) ověřuje, zdali jednotlivé komponenty správně fungují a zdali jsou správně osazeny. ICT využívá testovacího zařízení, které pomocí sond testuje DPS na přítomnost zkratů, přerušení, odporu, kapacity a dalších faktorů, aby určila, zda byla deska správně vyrobena.

V této kapitole bylo využito znalostí z [3].

Tato práce se zaobírá návrhem a realizací obslužné aplikace tohoto testovacího zařízení. V následující podkapitole je blíže vysvětlen princip jak zařízení pracuje.

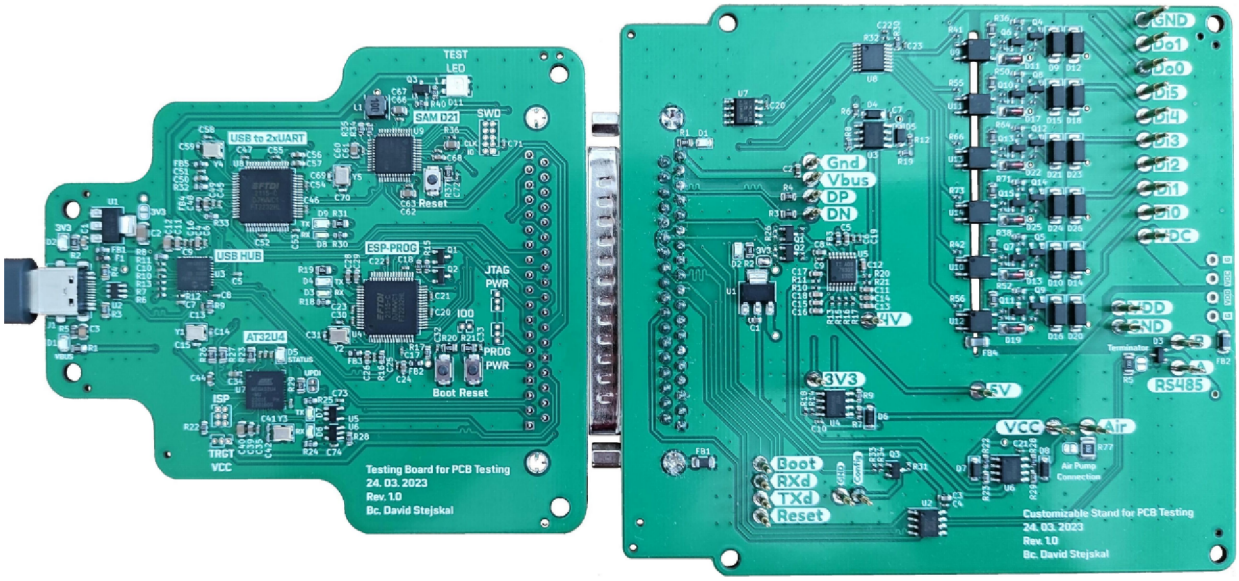
1.3 Testovací stanice

Návrh a následná realizace testované stanice je blíže popsána v práci [1]. Pro bližší pochopení problematiky, jak funguje testovací stanice jsou v této podkapitole zmíněné základní principy fungovaní stanice. Zařízení se skládá ze dvou hlavních komponent: univerzální testovací DPS a přizpůsobovací DPS (testovací podstavec).

- **Univerzální Testovací DPS**: Tato deska je v ideálním případě neměnná. Je navržena tak, aby byla schopna pracovat s různými typy testovaných DPS, což zajišťuje její flexibilitu a univerzálnost.
- **Přizpůsobovací DPS:** Na rozdíl od univerzální testovací DPS je přizpůsobovací DPS navržena speciálně pro konkrétní verzi testovaného produktu. Tím je

umožněno testovací stanici efektivně pracovat se specifickými částmi a účinné testování jednotlivých funkcí testované desky.

Následující obrázek 1.1 ukazuje testovací DPS (vlevo), která je univerzální a přizpůsobovací DPS, která je vyrobena přímo pro testovanou desku plošných spojů.



Obr. 1.1: Testovací stanice - převzato z [1]

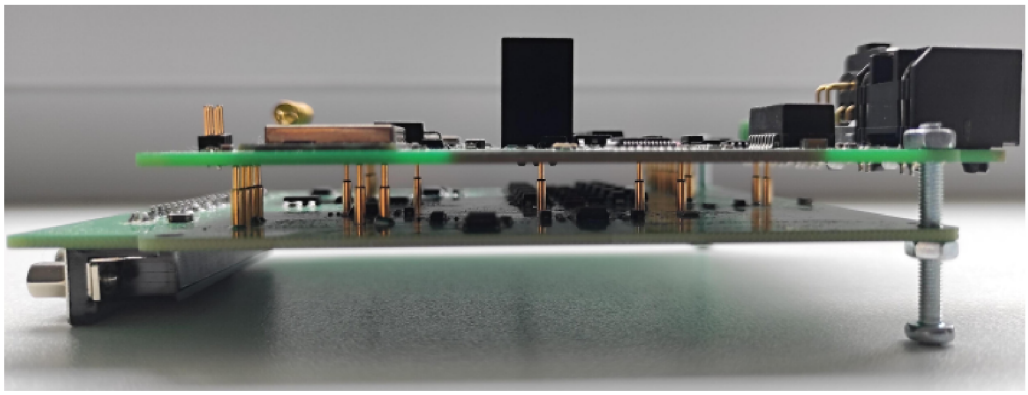
Testovaná deska je propojena s přizpůsobovací deskou prostřednictvím takzvaných testovacích bodů, známých také jako POGO piny. Tyto piny jsou speciálně navrženy, aby poskytovaly dočasný, avšak spolehlivý elektrický kontakt s testovanou DPS. Při testování je testovaná DPS umístěna tak, aby kontaktní místa (plošky) na testované DPS přesně odpovídala POGO pinům na přizpůsobovací desce. Celé zařízení je potom ukotveno proti nežádoucímu pohybu pomocí distančních můstků. Tímto způsobem je zajištěno přesné a efektivní měření napětí a komunikace mezi testovanou a přizpůsobovací DPS.

POGO piny hrají klíčovou roli v přenosu signálů, což zahrnuje nejen měření, ale i komunikační spojení. Tyto piny umožňují propojení s různými rozhraními, jako jsou USB a RS-485 či programování čipu ESP32.

Samotné propojení můžeme vidět na obrázku č.1.2

Obsluha zařízení:

Zařízení pracuje na principu, že se uživatel připojí k testovací DPS pomocí obslužné aplikace a spustí automatické testy, které otestují testovanou DPS a na základě testů vytvoří zprávu o stavu testovaných částí. Automatické testy se provedou na základě tzv. testovacích scénářů. Jejich princip přiblíží kapitola 1.4 . Uživatel



Obr. 1.2: Propojení testované a přizpůsobovací desky - převzato z [1]

bude schopen provést tzv. "vyskladnění". Tím proběhne nastavení vnitřních parametrů testované DPS. Testovaná deska tak bude připravena pro další montáž do zařízení či pro následný export. [1] [2]

1.4 Testovací scénáře

Testovací scénáře je nástroj, který umožní testeru (osobě), která kontroluje stav DPS, zjistit zdali výrobek funguje dle očekávání a splňuje požadavky. Scénáře kontrolují funkci konkrétních komponentů desky zadáním vstupům a pozorování reakce výstupů na tyto vstupy. Některé testy vyžadují splnění předpokladů, aby mohl test proběhnout.

Prostřednictvím těchto scénářů je tester schopen odhalit potenciální chyby na testovaných oblastech DPS, které byly předem definovány jako kritické nebo náchylné k problémům. Tyto oblasti mohou zahrnovat jak specifické komponenty, tak i celkové funkční aspekty desky.

Každá DPS je navržena s určitými specifickými funkcemi a charakteristikami, a proto je pro každému typu testované desky náleží odpovídající testovací scénář. Tyto scénáře jsou zpracovány s ohledem na unikátní vlastnosti a požadavky každé DPS, aby bylo zajištěno, že testy jsou relevantní a efektivní. [6] Pro pochopení problematiky testovacích scénářů si potřebujeme stanovit následující pojmy:

- **Definování testovacích požadavků**
  Aby se mohlo stanovit zdali daný test proběhl v pořádku či nikoliv jsou stanoveny požadavky, které musí testovaná část dodržet. Požadavkem se rozumí například úroveň napětí, rychlost přenosu dat či hodnota odporu. [6]

- **Testovací procedury**
  Testovací scénář se skládá z jednotlivých testovacích procedur (test case). Pro stanovení zdali testovaná část funguje dle očekávání se využívá tzv. Pass/Fail metriky. Tato metrika je založena na předem definovaných testovacích požadavcích. Pokud výsledky testu splňují stanovené kritérium, test je hodnocen jako "Pass"(prošel). Naopak, pokud výsledky nesplňují očekávané parametry, test je označen jako "Fail"(neprošel).[6]

1.5 Komunikace

V následujících podkapitolách jsou popsány základy komunikací, které jsou využívané při návrhu a následné implementaci aplikace.

1.5.1 Komunikace po sériové lince

Komunikace po sériové lince mezi počítačem a mikrokontrolérem je proces přenosu dat, který jak název napovídá využívá sériových dat. Tento způsob komunikace je běžný u mikrokontrolérů a je obvykle realizován pomocí protokolů jako je UART, SPI nebo I2C. V této podkapitole bude nastíněno, jak komunikace po sériové lince funguje, se zaměřením na UART (*Universal Asynchronous Receiver/Transmitter*).

UART je asynchroní komunikace, to znamená, že nevyužívá, žádný hodinový signál. Místo toho je přenos synchronizován pomocí předem dohodnuté přenosové rychlosti (*baud rate*). Ta se pohybuje v rozmezí od 110 bps do 230400 bps. Skládá se z přijímacího modulu a vysílacího modulu. Přijímací modul přijímá sériové signály a převádí je do paralelní formy pro další zpracování, zatímco vysílací modul provádí opačný proces – převádí data z paralelní formy do sériové a vysílá je.

Proces komunikace začíná inicializací, kde obě zařízení (počítač a mikrokontrolér) musí být nakonfigurována na stejnou přenosovou rychlost (*baud rate*), počet datových bitů (*typicky 8 bitů*), počet stop bitů a paritu. Start bit signalizuje začátek přenosu dat, následují datové bity, které obvykle začínají nejméně významným bitem (LSB). Paritní bit, pokud je použit, slouží k základní kontrole chyb, a stop bit ukončuje přenos, čímž zajišťuje synchronizaci mezi vysílačem a přijímačem.

Obr. 1.3: Rámec komunikace

Ač se UART už v dnešní době moc nevyužívá, tak jsou na něm postaveny standardy jako jsou RS-232, RS-485 či RS-422, které určují specifické parametry a použití.[8]

1.5.2 Bluethooth LE zařízení

Bluetooth Low Energy, často označované jako BLE, je technologie navržená pro efektivní bezdrátovou komunikaci s minimální spotřebou energie. Bylo poprvé představeno ve specifikaci Bluetooth verze 4.0, zaměřuje se primárně na aplikace, které vyžadují pravidelné přenášení malých datových paketů, a to s cílem maximalizovat úsporu energie. BLE operuje na frekvenci 2.4 GHz a je optimalizováno pro systémy, které vysílají data ve velmi malých množstvích, což výrazně redukuje spotřebu energie a prodlužuje životnost baterie zařízení.

GATT protokol

GATT (Generic Attribute Profile) představuje klíčovou vrstvu v tzv. protokolovém zásobníku BLE. Tato vrstva specifikuje, jak jsou služby a charakteristiky strukturovány a jak probíhá komunikace mezi BLE zařízeními. Základem GATT jsou služby a charakteristiky, které jsou blíže popsané níže.

Služby: Definovány jako kolekce charakteristik, služby poskytují specifické funkcionality a jsou identifikovány unikátním UUID (Universally Unique Identifier). Tyto mohou zahrnovat různé typy charakteristik, které společně vykonávají definovanou funkci.

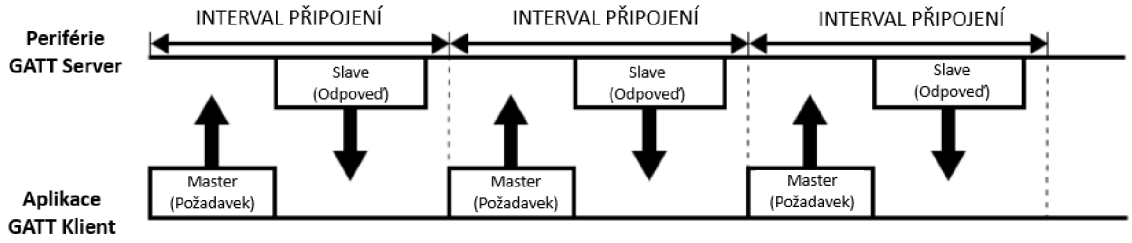
Charakteristiky: Jsou základními stavebními prvky služeb a obsahují hodnoty, které reprezentují specifické informace, jako jsou teplotní data, srdeční tep, nebo jiné senzorové údaje. Každá charakteristika má přidružené deskriptory, které poskytují další metadata o charakteristice, včetně informací o jednotkách, rozsahu hodnot, a dalších relevantních vlastnostech.

Principy komunikace

Komunikace v rámci GATT protokolu funguje na principu Server-Klient, kde klient (typicky mobilní zařízení nebo jiné inteligentní zařízení) iniciuje požadavek na server (často senzor nebo jiné BLE zařízení). Požadavky mohou zahrnovat čtení nebo zápis hodnot charakteristik. Server poté reaguje odpovědí, která může zahrnovat potřebná data nebo potvrzení o úspěšném zápisu.

Čtení a zápis: Klient může požádat o čtení hodnot z charakteristiky nebo poslat data k zápisu do charakteristiky na serveru.

Notifikace a indikace: Pro dynamickou komunikaci GATT umožňuje serveru automaticky posílat aktualizace (notifikace) nebo vyžadovat potvrzení přijetí dat (indikace) od klienta, což zvyšuje interaktivitu a zabezpečení přenosu. Pro bližší porozumění je komunikace znázorněna na následujícím obrázku: (1.4) [9][10]



Obr. 1.4: Komunikace - GATT protokol : Převzato a upraven z [10]

1.5.3 REST API

Representational State Transfer Application Programming Interface, zkráceně REST API, je nástroj pro komunikaci mezi různými systémy na internetu. V dnešní době se používá pro svůj efektivní způsob pro výměnu dat mezi klienty a servery, například mezi webovými aplikacemi a databázovými službami.

REST API využívá standardní metody a protokoly webu, zejména HTTP (Hypertext Transfer Protocol), což je základní komunikační protokol používaný na internetu. HTTP definuje metody (jako jsou GET, POST, PUT, DELETE), které určují, jakou akci chceme na serveru provést. Například metodou GET lze získat data ze serveru, zatímco POST se používá k odeslání dat na server. REST je nejčastěji používán jako správcovské API pro CRUD (Z angličtiny - Vytvoření, Čtení, Aktualizace, Smazání) k implementaci komunikace s databází.

Vzhledem k tomu, že REST API je založeno na stateless komunikaci, tak každý HTTP požadavek od klienta k serveru musí obsahovat všechny informace potřebné k porozumění požadavku. To eliminuje potřebu ukládání stavu mezi jednotlivými požadavky na serveru a tím je kladen i menší nápor na databáze.

V případně přenášení citlivých dat se data šifrují pomocí protokolu HTTPS a použití tokenů. [4]

2 Analýza požadavků

Tato kapitola se zaměřuje na analýzu požadavků pro vytvoření obslužné aplikace pro systém výstupní kontroly elektroniky. .

2.1 Funkční požadavky

Funkční požadavky definují, co systém musí umět, aby splnil očekávání uživatelů a zadání projektu.

- **Komunikace s testovacím zařízením**: Aplikace musí být schopná komunikovat s testovacím zařízením pomocí USB a Bluetooth. USB se bude používat pro přenos dat a nahrávání firmware, zatímco Bluetooth bude sloužit k nastavení konfigurace zařízení.[13]
- **Nahrávání firmware**: Aplikace musí umožnit nahrání firmware do testovaných zařízení. Tento firmware bude specifický pro každý typ zařízení a bude stažen z cloudové služby.
- **Vytváření a odesílání testovacích reportů**: Po dokončení testování musí aplikace generovat detailní report o výsledcích testu a odeslat jej zpět do cloudové databáze. Report bude obsahovat informace o provedených testech a jejich výsledcích.
- **Správa uživatelů**: Aplikace musí podporovat autentizaci a autorizaci uživatelů, aby byla zajištěna bezpečnost dat. Použití API klíčů a šifrování pomocí HTTPS zajistí, že přístup k datům bude omezen na autorizované uživatele.

2.2 Nefunkční požadavky

Nefunkční požadavky definují kvalitu a omezení systému, které musí být splněny, aby byl systém efektivní a použitelný. [13]

- **Multiplatformnost**: Aplikace musí být schopna běžet na různých operačních systémech, včetně Windows, Linux a macOS. To zajišťuje dostupnost aplikace pro širokou škálu uživatelů .
- **Výkon a škálovatelnost**: Systém musí být navržen tak, aby zvládal velké množství dat a uživatelů bez výrazného snížení výkonu. Databázová komunikace musí být efektivní a schopná rychle zpracovávat požadavky na čtení a zápis dat.
- **Bezpečnost**: Data přenášená mezi aplikací a cloudovou službou musí být šifrována a chráněna proti neoprávněnému přístupu. Použití API klíčů a HTTPS zajistí, že citlivá data budou chráněna během přenosu.

2.3 Analýza scénářů a použití

Analýza scénářů a použití poskytuje detailní pohled na to, jak bude aplikace používána v reálných situacích, a pomáhá identifikovat konkrétní požadavky na funkčnost a uživatelské rozhraní.[13]

- **Zahájení testování**: Uživatel vybere zařízení, které chce testovat, a aplikace automaticky stáhne odpovídající firmware z cloudové databáze. Firmware je nahrán do zařízení a spustí se testovací scénář.
- **Provedení testů**: Testovací scénář se skládá z jednotlivých testovacích procedur, které ověřují funkčnost zařízení. Výsledky každé procedury jsou zaznamenány a vyhodnoceny na základě předem definovaných kritérií.
- **Generování reportu**: Po dokončení testování aplikace generuje detailní report obsahující výsledky všech testovacích procedur. Tento report je následně odeslán do cloudové databáze pro další zpracování a archivaci.
- **Vyskladnění zařízení**: Pokud testy proběhnou úspěšně, aplikace stáhne finální firmware pro zařízení a pomocí Bluetooth jej nahraje do zařízení. Tento proces minimalizuje manuální zásahy a zvyšuje automatizaci celého testovacího procesu.

3 Návrh obslužné aplikace

Tento systém obslužné aplikace je navržen tak, aby byl co nejvíce flexibilní a připravený na budoucí technologické změny. Důležitou podstatou aplikace je její multiplatformnost, která umožňuje aplikaci fungovat na různých operačních systémech a zařízeních, což zvyšuje její dostupnost a uživatelskou přívětivost.

Z hlediska rozšířitelnosti je zásadní integrace s databází. Tato databáze slouží jako úložiště pro informace o testovaných deskách pro přizpůsobování (DPS) a výsledcích testů. Komunikace s databází je zajištěna pomocí REST API, což je moderní a flexibilní způsob, jak umožnit aplikaci efektivně komunikovat s daty.

Uživatelský workflow je navržen tak, aby byl co nejméně náročný a co nejvíce intuitivní. Tester (uživatel) si vybere desku, kterou chce testovat, a aplikace automaticky stáhne odpovídající firmware z databáze. Tento firmware bude obsahovat . Po nahrání firmware do DPS a spuštění testu, aplikace generuje výslednou zprávu s podrobným výčtem provedených testů, která je následně uložena zpět do databáze.

Krom samotného testování a přehrávání firmware, aplikace nabízí možnost "vyskladnění". V této fázi aplikace bude uživatel schopen pomocí BLE technologie nastavit parametry spojené s exportem desky.

Celkově je tento systém navržen s myšlenkou na snadnou aktualizaci a adaptaci na nové technologie a standardy. Jeho modularita a flexibilita zajišťují, že bude moci růst a rozvíjet se spolu s pokrokem v oblasti technologií a testování DPS.

Funkcionalitu aplikace znázorňuje následující vývojový diagram.

Následující podkapitoly pojednávají o návrhu testovacího scénáře, komunikace po sériové lince s testovací stanicí a komunikace s databází.

3.1 Návrh testovacího scénáře

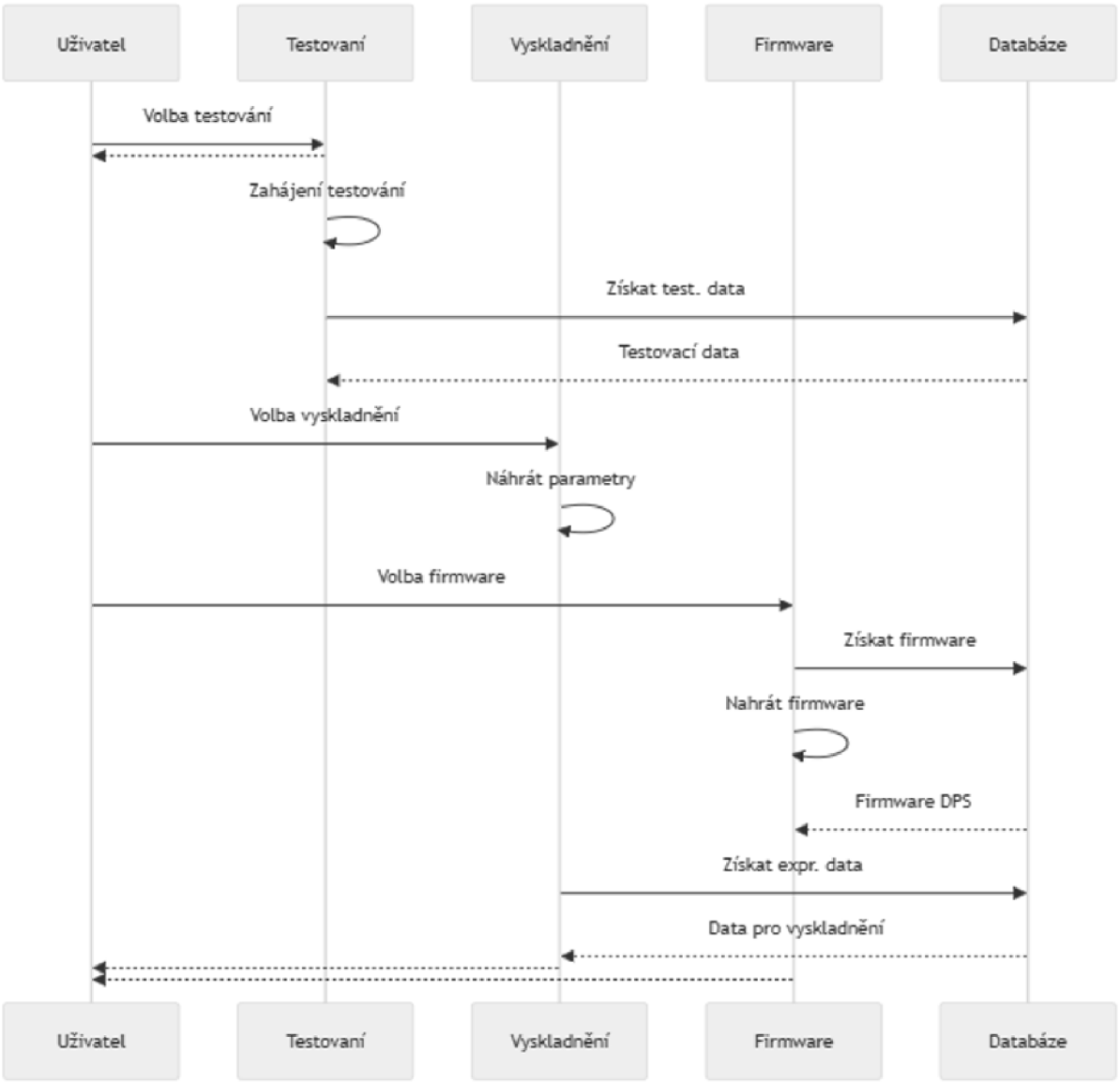
Testovací scénář se navrhuje pro testovanou desku plošných spojů, která je vyobrazena na obrázku č.3.2.

Při navrhnutí scénáře lze testování rozdělit do několika procedur. Procedury byly vytvořeny, dle logických funkčních celků testované desky, které je zapotřebí otestovat.

- Komunikace po sběrnici RS-485
- Digitální vstupů/výstupů
- Úrovně jednotlivých napěťových kanálů AD převodníku

Pro zkontrolování funkčnosti digitálních vstupů a výstupů testované desky je nejprve potřeba ověřit zdali funguje komunikace po sběrnici RS-485.

Tato komunikace se stará o aktivaci digitálních výstupů 8-kanálového převodníku testované desky. (Změření napětí zajistí přizpůsobovací DPS). Pomocí RS-458 se

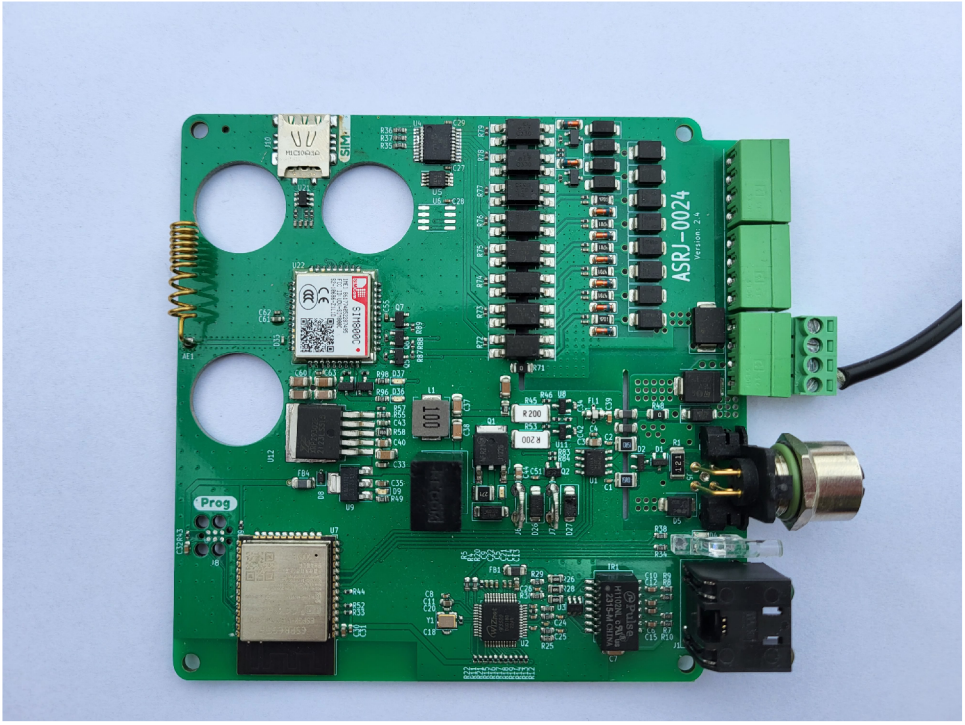


Obr. 3.1: Návrh funkcionality aplikace

zároveň předává informace o stavu digitálních vstupů testované DPS. Pokud by komunikace mezi testovanou deskou a přizpůsobovací DPS byla nefunkční, tak se změří pouze napěťové úrovně AD převodníku. [1]

Níže je návrh jednotlivých procedur podrobně rozepsaný, pro větší názornost je celý návrh testovacího scénáře ještě v podobě výbojového diagramu. (na obrázku č. 3.3 **Zkontrolovat stav RS-485:**

- **Odeslání zprávy:**
  Iniciuje se komunikace posláním předdefinované zprávy po sériové lince.
- **Čekání na odpověď:**
  Systém počká na odpověď od zařízení, s kterým komunikuje. Čekání je ome-



Obr. 3.2: Testovaná DPS

zeno časovým rozmezím, aby nedošlo k zbytečnému zdržení v případě, že zařízení neodpovídá.

- **Kontrola zprávy:**
  Testovací DPS zprávu zpracuje a navrátí předdefinovanou odpoveď.Po přijetí odpovědi se zkontroluje, zda zpráva odpovídá očekávanému formátu a obsahu a následně se vyhodnotí, zda je stav RS-485 v pořádku nebo zda jsou zjištěny nějaké problémy.

Zkontrolovat stav Digitálních výstupů (Air, D00, D01): *Pro každý výstup se bude procedura vykonávat samostatně*

- **Měření výstupu:**
  Prostřednictvím sériové komunikace se požádá o měření digitálního výstupu. Přizpůsobovací DPS následně aktivuje specifický digitální výstup na testované DPS
- **Čekání na odpověď:**
  Po aktivaci výstupu a následného měření odpovídajícího vstupu na přizpůsobovací desce, systém očekává zpětnou vazbu od testovací stanice, opět s časovým omezením.
- **Kontrola a vyhodnocení:**

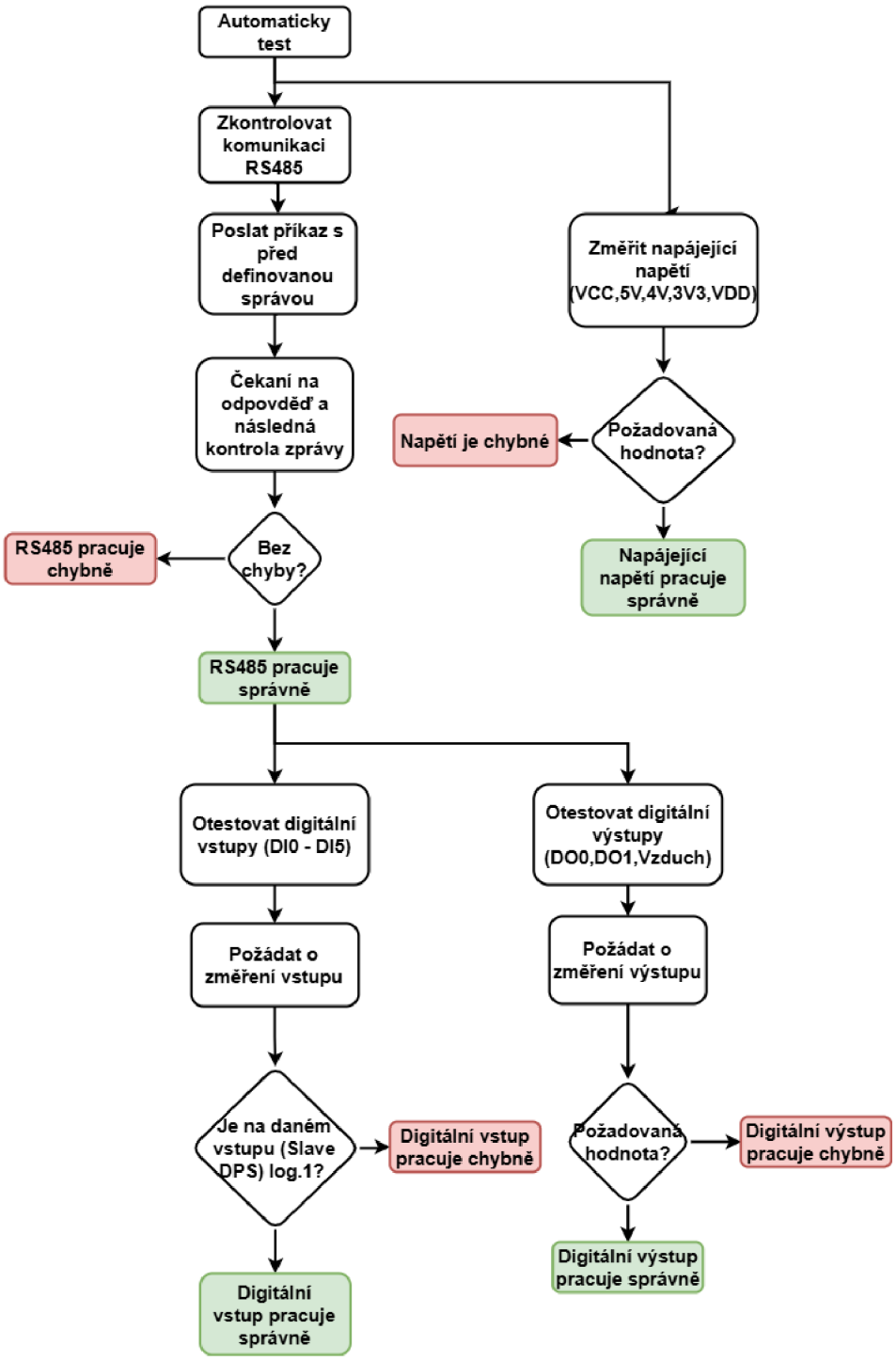
Následuje kontrola zdali naměřené napětí či logická úroveň na vstupech přizpůsobovací DPS, odpovídá hodnotě v očekávaném rozmezí. Následně se vyhodnotí stav tohoto výstupu.

Zkontrolovat stav Digitálních vstupů (D10, D11, D12, D13, D14, D15): *Pro každý vstup se bude procedura vykonávat samostatně*

- **Aktivace vstupu:** Systém pomocí sériové linky zažádá o zjištění stavu daného digitální vstupu. Testovací stanice pak aktivuje port na testované DPS pro žádaný vstup a aktivuje příslušný výstup.
- **Kontrola a vyhodnocení:**
  Systém pak po přijetí odpovědi, provede kontrolu zdali, žádaný se vstup chová dle očekávání.

Kontrola napětí na všech kanálech A/D převodníku (VCC, 5V, 4V, 3V3, VDD):

- **Měření napětí:** Obdobně jako u předchozích procedur systém zažádá testovací stanici o měření určitého testovací bodu (*test pointu*) po seriové lince. Stanice pak provede se měření napětí na příslušném kanálu analogově-digitálního převodníku.
- **Kontrola a vyhodnocení**
  Tyto kanály zahrnují různé napěťové úrovně, které by měly být v určitých předem specifikovaných rozmezích. Kontroluje se, zda naměřené napětí odpovídá očekávané hodnotě.



Obr. 3.3: Testovací scénář

3.2 Návrh seriové komunikace s deskou

Při převzetí testovací desky a pokusu o její zapojení do existujícího fyzického systému bylo zjištěno, že předchozí implementace desky není plně funkční, přičemž hlavním

nedostatkem byla absence implementace komunikace přes sériovou linku. Po hlubší analýze problému a následné konzultaci s vedoucím projektu bylo rozhodnuto, že původní testovací deska bude nahrazena zařízením, které bude schopno simulovat její funkcionalitu.

3.2.1 Struktura Rámce

Každá zpráva odeslaná pomocí protokolu SkuLweP [12] se skládá z několika klíčových částí:

- **Start Symbol (0xAA):** Každý rámec začíná tímto specifickým bajtem, který signalizuje začátek zprávy.
- **Address:** Toto pole určuje adresu cílového zařízení. Hodnota '0' je použita pro broadcast zprávy vysílané na všechna zařízení, zatímco hodnoty '1-255' specifikují konkrétní zařízení.
- **Length:** Délka rámce od startovního do koncového symbolu, včetně všech mezi nimi ležících částí.
- **Command:** Příkaz určuje operaci, která má být provedena. Například příkaz pro získání ID zařízení nebo pro testování komunikace na sběrnici.
- **Payload:** Obsah zprávy, který nese data související s příkazem. Délka a obsah payloadu se liší v závislosti na typu příkazu.
- **Checksum:** Kontrolní součet slouží pro zajištění integrity dat. Vypočítá se jako aritmetický součet všech předchozích částí rámce, maskovaný na jeden bajt.
- **End Symbol (0x55):** Symbol uzavírající rámec, signalizující jeho konec.

3.2.2 Proces Komunikace

Komunikační proces mezi master a slave zařízeními probíhá ve formě dotazu a odpovědi. Master zařízení pošle příkaz na slave zařízení, které následně zpracuje požadavek a vrátí odpověď. Odpověď má obdobný rámec jako původní příkaz, ale v poli příkazu je hodnota maskována hodnotou 0x80 pro indikaci, že se jedná o odpověď.

3.2.3 Příklady Příkazů

- **Get device id (CMD = 0x01):** Tento příkaz neobsahuje payload a slouží k získání identifikátoru zařízení. Odpověď obsahuje 16-bajtový identifikátor.
- **TP measurement (CMD = 0x11):** Změří napětí na testovacím bodě a vrátí 16-bitovou hodnotu měření.
- **Test bus communication (CMD = 0x21):** Odesílá data na RS-485 sběrnici a přijímá odpověď.

- **TEST DO (CMD = 0x31):** Měří napětí na zadaném výstupu před a po sepnutí a vrátí 2x 16-bitovou hodnotu měření pomocí ADC.
- **TEST DI (CMD = 0x41):** Získá hodnotu zadaného vstupu před a po nastavení výstupu a vrátí získané hodnoty ve formátu 0x00 nebo 0x01. (log. 0 / log.1)
- **Reset (CMD = 0x02):** Restartuje běžící operace a celý testovací subsystém. [12]

3.3 Návrh komunikace s databází

Komunikace obslužné aplikace s vzdálenou databází je uskutečněna prostřednictvím REST API (blíže popsána v kapitole 1.5.3).

Při návrhu výměny dat mezi obslužnou aplikací a databázovou službou je zapotřebí, aby nástroj splňoval následující požadavky:

- **Schopnost získání informací o testované desce:** Získání informací o testované desce a parametrů potřebné k testování. Tato funkcionalita umožní obslužné aplikaci načíst a zpracovat potřebné údaje o hardwaru, které jsou uloženy v databázi.
- **Stažení firmware pro přizpůsobovací desku:** Možnost stáhnout příslušný firmware pro přizpůsobovací desku, která je určena k testované desce. Tento proces zahrnuje vyhledání správného firmware v databázi a jeho dostupnost pro obslužnou aplikaci.
- **Získání dat pro vyskladnění:**Získání dat potřebných pro vyskladnění zařízení, zejména pro komunikaci přes Bluetooth Low Energy (BLE). To zahrnuje přístup k nezbytným parametrům zařízení a odkazu na příslušný firmware.
- **Stažení firmware testované desky za účelem její vyskladnění:** Pro účely vyskladnění testované desky musí nástroj umožňovat stažení aktuálního firmware. Zpřístupnění správného souboru firmware uloženého v databázi, který je nutný pro správnou funkci desky po vyskladnění.
- **Uložení výsledné zprávy po úspěšném testování:** Po úspěšném provedení všech testů by nástroj měl umožňovat uložení výsledné zprávy do databáze. Tato zpráva by měla obsahovat všechny relevantní údaje o průběhu a výsledcích testů, což je důležité pro dokumentaci a případné budoucí analýzy.

Komunikace mezi obslužnou aplikací a databázovou službou musí být navržena tak, aby byla bezpečná a efektivní. Data se budou přenášet ve formátu JSON, což zajišťuje snadnou čitelnost a kompatibilitu s různými systémy. Bezpečnostní opatření zahrnují autentizaci a autorizaci uživatelů pomocí API klíčů a šifrování dat pomocí HTTPS, což chrání citlivé informace před neoprávněným přístupem.

Celkově je důležité, aby návrh nástroje podporoval hladkou a efektivní výměnu dat mezi obslužnou aplikací a databázovou službou, splňoval všechny specifikované požadavky a umožňoval snadnou správu a údržbu systému.

4 Realizace obslužné aplikace

4.1 Nástroje obslužné aplikace

V následujících podkapitolách jsou zadefinovány důležité pojmy a definují se zde nástroje, které budou využívány při vytváření obslužné aplikace testovací stanice.

4.1.1 Python

Python je vysokoúrovňový, objektově orientovaný programovací jazyk, který byl zvolen pro svoji schopnost multiplatfomní podporu. Jazyk je tzv. přenositelný (schopnost spustit na různých platformách jako je windows, linux či maxOS)

Jazyk nabízí velikou podporu z hlediska knihoven a dostupných zdrojů.

Python byl zvolen díky své univerzálnosti a schopností zprácovát data.

4.1.2 Visual Studio Code

Visual Studio Code (často zkracováno jako VS Code) je populární open-source editor zdrojové kódu vyvinutý společností Microsoft. Je dostupný pro Windows, macOS a Linux a je široce používán vývojáři pro programování v různých jazycích. Toto prostředí nabízí bohatý ekosystém rozšíření, které usnadňuje vývojářům vytváření aplikací. V této práci se VS code využívá k vývoji samotné aplikaci v Pythonu. Vs Code umožňuje využít virtuálního prostředí, což je izolovaný pracovní, který umožňuje oddělit knihovny nebo balíčky třetích stran (dependency) a nastavení pro různé Python projekty.

4.1.3 Flet

Při výběru nástroje pro vývoj grafického uživatelské rozhraní aplikace (GUI) byl zvolen **Flet**. Stejně jako python, aplikace vyvinuté s využitím Fletu může být vyexportována pro jakoukoliv platformu. Společně s jazykem Python, který nabízí nepřeberné množství knihoven a různé podpory pro aplikace třetích stran tvoří ideální nástroj pro potřeby této aplikace.

Flet je postaven na frameworku Flutteru. Flutter je open-source framework od Googlu pro vývoj nativních aplikací pro Android, iOS, web a desktop z jednoho kódu (obdobně jak je tomu u knihovny Flet). Narozdíl od Flet, Flutter využívá programovacího jazyku Dart a základním stavebním blokem je widget, zatímco Flet využívá tzv. controls.

Flet CLI

Flet framework obsahuje modul Flet CLI, který po úspěšné implementaci aplikaci

umožňuje tzv. balení Flet aplikace do samostatného spustitelného souboru nebo instalačního balíčku připraveného pro distribuci na různé platfromy. V případě desktopové aplikace pro platformy Windows, Linux, macOS.

Během balení aplikace do spustitelného souboru nebo instalačního balíčku se provádí následující kroky:

1. **Vytvoření nového Flutter projektu** v dočasném adresáři na základě předdefinované šablony.
2. **Kopírování vlastních ikon a obrázků úvodní obrazovky** z adresáře `assets` do Flutter projektu.
3. **Generování ikon pro všechny platformy** pomocí specifických nástrojů.
4. **Zabalování Python aplikace**, kde jsou všechny Python soubory zkompilovány a přidány jako aktiva do balíčku.
5. **Spuštění kompilace pro cílovou platformu** za účelem vytvoření spustitelného souboru nebo instalačního balíčku.
6. **Kopírování výsledků kompilace** do specifikovaného výstupního adresáře.

Při vývoji a distribuci aplikací je důležité udržovat přehled v jednotlivých verzích. Během balení lze specifikovat verzi spustitelného souboru nebo balíčku, což umožňuje rozlišování jednotlivých *buildů* a jejich zobrazení uživatelům.[15]

Výpis 4.1: Příklad verzování distribuce aplikace pro Windows

```
flet build windows --build-number 2 --build-version 1.1.0
```

Ukládání dat

Za běhu programu, je důležité uchovávat dočasné informace(např. jméno testované desky). Flet poskytuje API pro ukládání těchto dočasných informací ve formmátu *klíč-hodnota* na straně serveru během uživatelské relace (*session storage*). Data jsou přechodná a nejsou zachována mezi restartováním aplikace, což znamená, že po ukončení relace budou odstraněna.

Příklad využití session storage lze vydět na výpisu 4.2, kde znázorněna redukce počtu serverových požadavků, kde se načítají data o testovaných zařízeních. Tím, že se dočasná data ukládají na straně klienta, tak se nemusí se znovu načítat z databáze.[15]

Výpis 4.2: Příklad uchování přechodných dat

```
page.session.set("hw_data",  api_client.get_hardware_data())
value = page.session.get("hw_data")
```

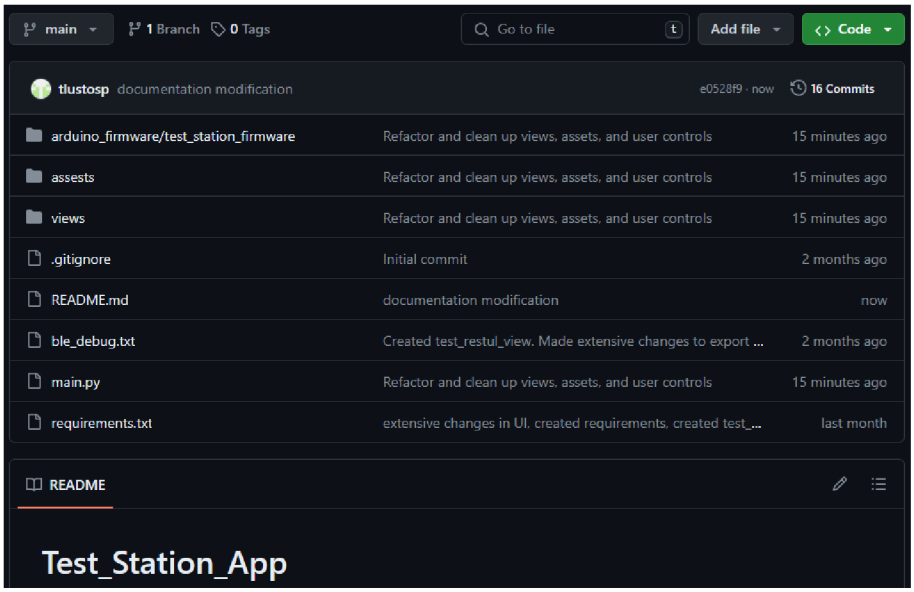
4.1.4 Github

GitHub je platforma pro vývojáře, která poskytuje webovou službu pro vývoj softwaru a podporuje verzování pomocí systému Git. Usnadňuje týmovou spolupráci na vývoji softwaru a správu zdrojového kódu, zejména zálohování a návrat ke starším verzím kódu.

Git je distribuovaný systém pro správu verzí, který sleduje změny v kódu a podporuje spolupráci. Udržuje historii všech změn pomocí "commitů"(soubory změn) ukládaných do "repozitářů"(úložišť).

GitHub, založený na Gitu, poskytuje webové rozhraní pro snadnou správu repozitářů. Komunikace s GitHubem probíhá prostřednictvím vývojového prostředí VS Code (viz kapitola 4.1.2).

Vývojáři ukládají změny jako "commit"s popisem, což umožňuje zpětnou kontrolu a návrat ke starším verzím. Při práci ve větších týmech se využívá "branching"pro práci na různých částech projektu bez ovlivnění hlavní verze kódu. Po dokončení práce na větvi (branch) může vývojář otevřít pull request, aby požádal ostatní členy týmu o revizi a schválení změn před jejich sloučením (merged) do hlavní větve, což podporuje týmovou spolupráci a zajišťuje důkladnou kontrolu změn. [17]



Obr. 4.1: Ukázka repozitáře aplikace

4.2 Implementace komunikačních modulů

V následujících podkapitolách je popsáno, jak bylo postupováno při implementace komunikace mezi jednotlivými periferiemi.

Za účelem budoucí rozšířitelnosti aplikace a přehlednosti implementovaného kódu byla celá komunikační část rozdělena do tzv. modulů (soubory s třídami a metodami).

4.2.1 Seriová komunikace

Tento modul je navržen pro komunikaci s testovací deskou prostřednictvím sériové linky, která je založena na protokolu SkuLweP (blíže v kapitole 3.2). Třída obsažená v modulu nabízí základní metody mezi které se řadí možnost připojení, odesílání příkazů pomocí kterých se zažádá o testování různých periferií a následné čtení odpovědí.

Po odeslání příkazu modul čeká na odpověď od testovací desky. Při čekání na odpověď modul kontroluje příchozí data a hledá specifický startovací symbol. Jakmile je nalezen, modul pokračuje ve čtení dat až do koncového symbolu nebo do vypršení časového limitu, který je definován v testovacích parametrech v kapitole 4.3.1. Pokud je zpráva přijata úspěšně, je vrácena jako pole bajtů, jinak nic nevrátí. O následné zpracování dat se starají metody uvedeny ve výpisu 4.3

Celý modul slouží pro získání testovaných dat a jejich následné předzpracování pro testovací modul, který je blíže popsán v kapitole 4.3

Výpis 4.3: Metody pro získání testovacích dat

```
def get_device_id(self)->str                                    1
def get_tp(self, tp_idx)->float                                 2
def get_bus_communication(self, message)->str                   3
def get_do(self, do_idx, ad_idx)->float                         4
def get_di(self, di_idx, to_idx)->int                           5
def reset(self)->bool                                           6
def get_state(self)->bool                                       7
```

Tento výpis ukazuje metody pro získávání testovacích dat z testovací desky. Modul umožňuje získat ID zařízení ve formě řetězce, měřit napětí na testovacích bodech, které navrací napětí ve formátu float, a získat stav sběrnicové komunikace, přičemž navrací zprávu ve formě řetězce. Při měření digitálních výstupů navrací napětí ve formátu float a v případě digitálních vstupů ve formátu int (log. 0 nebo log. 1). Modul také umožňuje restartovaní subsystému a získávání stavu subsystému. Podrobnější kódu, naleznete v příloze A

4.2.2 Bluethooth komunikace

Pro nastavení parametrů testované desky při procesu vyskladňování bylo zapotřebí implementovat BLE komunikaci (podrobněji popsáno v kapitole 1.5.2). Modul je postaven na knihovně **Bleak** a zahrnuje metody pro připojení, čtení a zápis dat různých typů do charakteristik zařízení. Knihovna Bleak byla zvolena především pro své multiplatformní vlastnosti a schopnost využívat integrovaný Bluetooth čip zařízení, na kterém bude aplikace spuštěna.

Při navazování spojení se zařízením se provádí metoda, která se opakovaně pokouší o připojení s nastaveným zpožděním mezi pokusy. Důvodem jsou intervaly ve kterých zařízení vysílá (je aktivní). Po úspěšném připojení se vrací informace o úspěšném spojení. Pokud se zařízení nepodaří připojit, pokusy se opakují podle stanoveného počtu.

Zapisování, popřípadě čtená data musí být správně zakódována/dekodována a převedena na příslušný datový typ. Při zápisu do každé charakteristiky je zapotřebí dodržet datový typ ve který jsou původní data uchovávána. Proto byly imlepmentovány následující metody:

Výpis 4.4: Metody pro nastavení parametrů

```
def __init__(self, device_address,services)                              1
            .                                                            2
            .                                                            3
            .                                                            4
async def write_number(self, uuid, value)->bool                          5
async def write_float(self, uuid, value)->bool                           6
async def write_text(self, uuid, value)->bool                            7
async def write_bool(self, uuid, value)->bool                            8
async def write_byte_array(self, uuid, value)->bool                      9
async def read_characteristic(self,uuid, retries, delay, type)           10
```

Metody přijaté UUID normalizují na 128-bitový tvar, který používá knihovna Bleak. Při úspěšném zakódovaní hodnoty do příslušného tvaru se celá hodnota ještě zabalí do bytové sekvence a probíhá pokus o zápis do charakteristiky.

UUID služeb a charakteristik jsou načítaný do aplikace ze vzdálené databáze (blíže popsáno v následující podkapitole *RestAPI*. Pro nalezení těchto charakteristik a služeb bylo využito modulu **BleakScanner**, který pomohl definovat nejen UUID služeb a charakteristik, ale také k čemu jednotlivé charakteristiky slouží. Na základě konzultace s vedoucím práce bylo určeno, které parametry budou modifikované: název testované desky, ip adresa, Sériové číslo zařízení, verze revize desky, maximální povolené napětí a parametr, který nastavuje zdali se má deska restartovat. [11]

4.2.3 RestAPI

Implementace REST API pro modul zařízení zahrnuje několik klíčových funkcí, které umožňují získávání a ukládání dat prostřednictvím HTTP požadavků. API má základní adresu (adresa databáze), kterou využívá každý požadavek, a všechny požadavky vyžadují specifický API klíč v hlavičce. Tento klíč zajišťuje bezpečnou a autorizovanou komunikaci mezi klientem a serverem.

Modul zahrnuje tři hlavní operace, které komunikují s různými koncovými body (endpointy):

Výpis 4.5: Metody pro komunikaci s databází

```
def __init__(self, base_url)->None                              1
def get_hardware_data(self, device_id)->dict                    2
def post_test_report(self, device_id, report_data)->bool        3
def get_dispatch_data(self, device_id)->dict                    4
```

Výše vypsané metody jsou zodpovědné za specifické úkony, které jsou klíčové pro správu zařízení v různých fázích běhu aplikace:

- **Přijímání dat pro testování:** První operace odesílá GET požadavek na konkrétní koncový bod s parametrem identifikujícím zařízení. Cílem této operace je přijmout parametry zařízení a odkaz na testovací firmware.
- **Odesílání testovacího reportu:** Druhá operace odesílá POST požadavek na jiný koncový bod s parametrem identifikujícím zařízení a tělem požadavku obsahujícím data reportu. Tato operace slouží k odeslání výsledků testování zařízení. Po úspěšném odeslání požadavku je kontrolován status kód a v případě úspěchu operace vrací potvrzení o správném uložení reportu.
- **Přijímání dat pro vyskladnění:** Třetí operace, obdobně jako první, odesílá GET požadavek na další koncový bod s parametrem identifikujícím zařízení. Cílem této operace je přijmout data potřebná pro vyskladnění zařízení, včetně odkazu na příslušný firmware testované desky.

Při zpracování požadavků metody zasílají nezbytné hlavičky a parametry. U metody POST se připojují také zasílaná data. Všechna data jsou strukturována ve formátu JSON, který zajišťuje jasnou a konzistentní reprezentaci informací potřebných pro komunikaci mezi klientem a serverem.

V případě budoucího rozšíření implementace těchto JSON záznamů, definovaných v databázi, je nezbytné dodržovat jejich pevně stanovenou strukturu. Aplikace následně dynamicky zpracovává data s ohledem na tuto strukturu, což podporuje modularitu a tím snadnou integraci nových zařízení.Níže jsou uvedeny ukázky dat a její struktury, které jsou načítaná z databáze. Výpisy celých záznamů jsou zmíněny v příloze C

Data pro vyskladnění

Výpis 4.6: Úkazka načítaných dat pro vyskladnění

```
{
    "ASRJ": {
        "address": "---",
        "data": {
            "ip_adress": {
                "UUID_char": "---",
                "UUID_serv": "---",
                "default_value": "---",
            }
        ...
```

Název testovaného zařízeni je, pro ukázku, "*ASRJ*". MAC adresa zařízení je uvedena pod klíčem "address"a slouží k jednoznačné identifikaci v síti. Objekt "data"obsahuje jednotlivé atributy zařízení, každý s příslušnými UUID charakteristikami, službami a výchozími hodnotami. Mezi tyto atributy patří IP adresa, restart (bool), název desky, maximální napětí, hardwarová verze a softwarová verze.

Data pro testování

Výpis 4.7: Úkazka načítaných dat pro testování

```
{
    "pca9557": {
        "baudrate":---,
        "timeout_limit":--,
        "criteria_range_V":--,
        "DO_limit_V":--,
        "tp_limit_V":--,
        "testing_areas": {
            "gpio_input": {
                "choosen": --,
                "ids":{
                    "DI0": "--",
            ...
```

V ukázce kódu je definována přizpůsobovací deska "*pca9557*", zahrnující testovací parametry jako je rychlost přenosu dat, časový limit, testovací kritérium, limity pro digitální výstupy či napětí. Blíže rozebráno v následující kapitole 4.3. Dále definuje testovací oblasti, včetně komunikace přes RS-485, digitálních vstupů, analogových výstupů a napěťových výstupů. Každá oblast má parametr *choosen* či se bude testovat a identifikátory, které mapují TP na fyzické piny na desce.

4.3 Implementace testovacího modulu

Testovací modul je jednou z nejdůležitějších částí aplikace. Slouží k testování periférií testované desky na základě vybraných testovacích oblastí a poskytuje podrobný přehled o výsledcích těchto testů. Modul je postaven na implementaci modulu, který slouží pro připojení k hardwaru pomocí sériové linky (popsán v kapitole 4.2.1). Odděluje tak samotnou komunikační část a testovací část. V případě změny v komunikačním protokolu či v logice testovaní se upraví pouze jeden z těchto modulů. Dále modul pracuje s daty potřebné pro testování (načtené z databáze), názvem zvolené přizpůsobovací desky a několika testovanými parametry, včetně tolerance hodnot při testování, limitů napětí a časového limitu po jakou dobu testovat. Průběh testování je blíže popsán v podkapitole *Průběh testování*.

Metody pro testování:

Výpis 4.8: Úkázka metod pro testování

```
def _testvoltage(self, tp_idx:int, pin_name:str,                    1
                 test_param:int) -> bool                            2
def _testRS485(self, message:str, pin_name:str) -> bool             3
def _testDO(self, do_idx:int, ad_idx:int, pin_name:str)-> bool      4
def _testDI(self, di_idx:int, to_idx:int, pin_name:str)->bool       5
def _generate_report(self) -> None                                  6
def StartTest(self) -> dict                                         7
            ...                                                     8
```

1. **Testování napětí**
   Metoda testuje napětí na zadaném testovacím bodě. Vypočítá dolní a horní limit na základě tolerance a zadaného limitu napětí. Pokud naměřené napětí leží v tomto rozmezí, test je považován za úspěšný. Výsledek testu je zaznamenán.
2. **Testování komunikace**
   Metoda testuje komunikaci po sběrnici zasláním zprávy a kontrolou odpovědi. Pokud odpověď odpovídá očekávání, test je považován za úspěšný. Výsledek testu je zaznamenán.
3. **Testování digitálního výstupu**
   Metoda testuje digitální výstup na specifikovaných indexech. Vypočítá dolní a horní limit napětí na základě tolerance a zadaného limitu napětí. Pokud naměřené napětí leží v tomto rozmezí, test je úspěšný. Výsledek testu je zaznamenán.
4. **Testování digitálního vstupu**
   Metoda testuje digitální vstup porovnáním dvou hodnot. Pokud se hodnoty liší, test je považován za úspěšný. Výsledek testu je zaznamenán.

4.3.1 Průběh testování

Hlavní metoda spouští všechny příslušné testy na základě načtené konfigurace. Uživatel si může v aplikaci zvolit, které oblasti budou testovány. Nejprve otestuje komunikaci po sběrnici, pokud uživatel tuto volbu aktivoval, a následně pokračuje dalšími testy podle výsledků komunikace a zvolené konfigurace. Po dokončení všech testů je vygenerována finální zpráva a výsledky testů jsou vráceny ve formátu JSON, který je potom dále aplikací zpracováván (odesílán do databáze) .

Při spuštění testů metoda kontroluje, zda uživatel vybral testovací oblast RS-485. Pokud je tato oblast vybraná, provede test komunikace a výsledky uloží. Poté prochází ostatní testovací oblasti, které uživatel vybral v konfiguraci. Pokud test RS-485 selže nebo není vybrán, ostatní testy mimo testování napětí jsou přeskočeny. (viz. kapitola *Návrh testovacího scénáře* 3.1).

Po proběhnutí testů se zavolá metoda, která generuje závěrečnou zprávu s výsledky všech provedených testů a ukládá ji do souboru ve formátu .log. Soubor obsahuje datum, čas, název testované desky, testované parametry a podrobnosti o výsledcích testů. Pokud složka `logs` neexistuje, je vytvořena.

Výpis 4.9: Úkazka výslednzprávy nahrávané do databáze

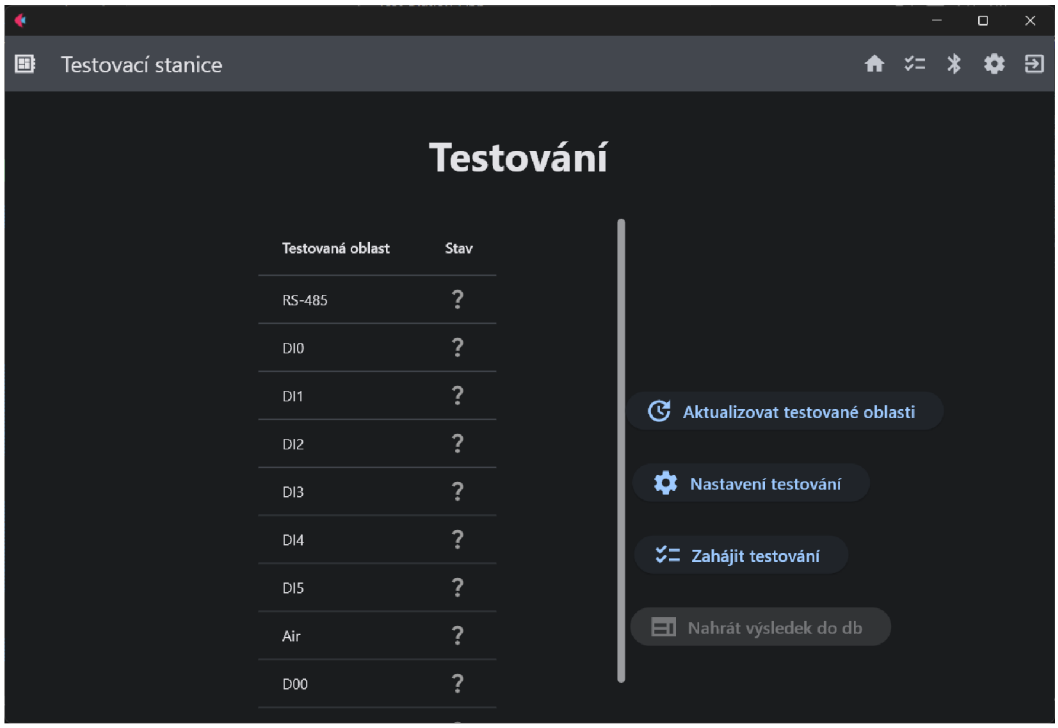
```
{                                                                    1
    "pca9557": {                                                     2
        "criteria_range_V":--,                                       3
        "DO_limit_V":--,                                             4
        "tp_limits_V":  { ... },                                     5
        "testing_areas":{                                            6
            "RS-485": {                                              7
                "RS-485": true                                       8
            },                                                       9
            "gpio_input": {                                          10
                "DI0": true,                                         11
                "DI1": false,                                        12
                "DI2": true                                          13
            },                                                       14
            "ad_output": {                                           15
                "Air": false,                                        16
                "D00": true,                                         17
            },                                                       18
            "voltage": {                                             19
                "VCC": false,                                        20
                "5V": true                                           21
                ...                                                  22
```

4.4 Implementace GUI

Tato kapitola nabízí ucelený pohled na funkcionalitu aplikace. Popisuje hlavní části aplikace (testování (4.2), nastavení testování (4.3) a vyskladnění(4.4)). Zbylé části aplikace jsou k nalezení v příloze E

Testování

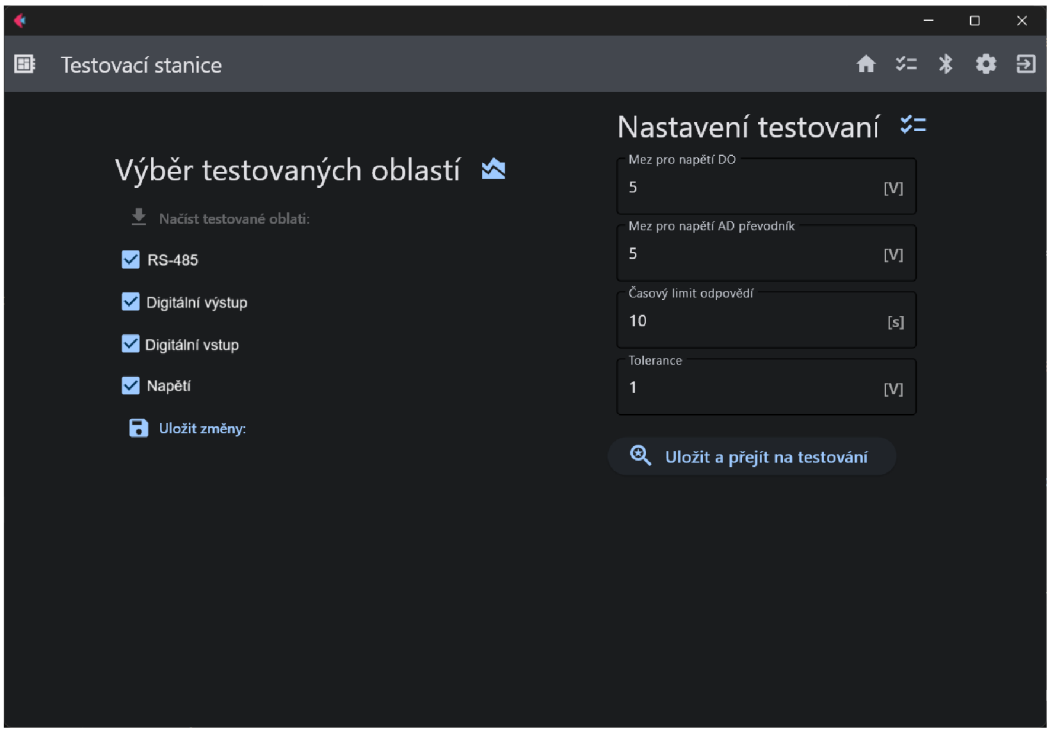
Uživateli se zobrazí testované periferie na základě konfigurace načtené z databáze. V průběhu testování se stav aktualizuje na základě pass/fail kritéria, které následně může nahrát do databáze. Uživatel si může zvolit, které oblasti se budou testovat (viz. 4.3). Kromě zvolení testovaných oblastí, má uživatel možnost upravit parametry testovaní.



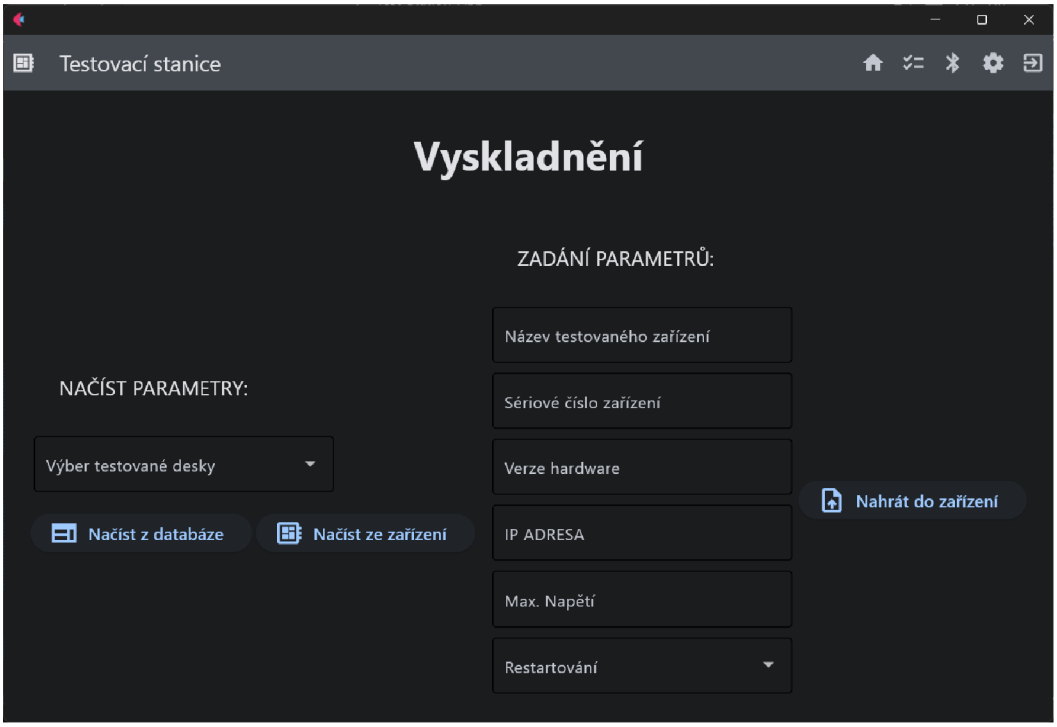
Obr. 4.2: Obrazovka pro testování

Vyskladnění

Obrazovka pro vyskladnění (viz. 4.4 umožňuje zadávání a načítání parametrů zvoleného testovaného zařízení. Uživatel může vybrat testovanou desku z nabídky a načíst parametry buď z databáze, nebo přímo ze zařízení. Data o testovaných zařízeních, včetně UUID parametrů potřebné pro komunikaci, jsou obdobně jako u testování, načítané z databáze.



Obr. 4.3: Obrazovka pro nastavení testování



Obr. 4.4: Obrazovka vyskladnění

5 Simulátor

Jako simulující zařízení bylo zvoleno Arduino kvůli jeho jednoduché architektuře a snadné implementaci požadovaných funkcionalit.

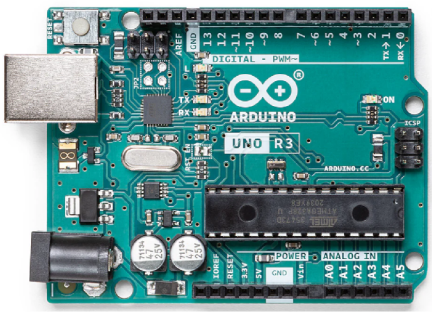
5.1 Arduino UNO

Arduino UNO je mikrokontrolérová platforma založená na 8bitové mikrokontrolérové jednotce (MCU) ATmega328 od firmy Atmel. Tato deska nabízí 14 digitálních vstupně-výstupních pinů, z nichž 6 může být použito jako PWM výstupy, a 6 analogových vstupů. Kromě toho má USB port, který slouží k propojení s počítačem, což umožňuje jak programování mikrokontroléru, tak i komunikaci po sériové lince, která je pro tuto práci klíčová.

Pro zajištění zobrazení stavů během testování byl k Arduinu připojen displej pomocí rozhraní I2C (dvouvodičový komunikační protokol). Zapojení displeje není v kontextu této práce podstatné, avšak jeho použití zvyšuje přehlednost a kontrolu nad probíhajícím testování. Samotné zařízení je pro ukázku znázorněno na obrázku č. 5.1

5.2 Arduino IDE

Programování samotné desky je realizováno prostřednictvím Arduino IDE, což je vývojové prostředí. Samotný kód je implementován v jazyce založeném na principu jazyků C/C++. Arduino poskytuje širokou škálu knihoven, což výrazně ulehčuje práci s již dříve zmíněnými periferiemi, bez nutnosti psát nízkoúrovňový kód pro každou periferii.[18]



Obr. 5.1: Arduino

5.3 Implementace komunikace

Aby bylo možné otestovat implementaci obslužné aplikace, bylo zapotřebí, aby simulované zařízení co nejvíce napodobovalo testovací desku. Z tohoto důvodu, bylo implementace provedena na základě protokolu SkuLweP definované v kapitole 3.2.

Výpis 5.1: Ukázka implementace seriové komunikace

```
if (Serial.available() > 0) {
    delay(10);
    byte startByte = Serial.peek();
    if (startByte == 0xAA && Serial.available() >= 3) {
      Serial.read();  // Consume the start byte
      byte address = Serial.read();
      byte length = Serial.read();
      if (Serial.available() >= length - 3) {
        byte data[length];
        data[0] = startByte;
        data[1] = address;
        data[2] = length;
        for (int i = 3; i < length; i++) {
          data[i] = Serial.read();
        }
        if (data[length - 1] == 0x55) {
          lcd.clear();
          lcd.setCursor(0, 0);
          switch (data[3]) {
            case 0x01: handleGetDeviceId(data); break;
            case 0x11: handleTPMeasurement(data);break;
            case 0x21: handleTestBusCommunication(data);break;
            case 0x31: handleTestDO(data);break;
            case 0x41: handleTestDI(data);break;


                ...

                ...
```

Výše zmíněný kód zpracovává data ze sériového portu mikrokontroléru. Jeho hlavním cílem je zajistit, aby přijaté zprávy byly kompletní a správně formátované před jejich zpracováním. Bylo implementováno malé zpoždění, které dává bufferu čas se naplnit. Kontrola startovacího bajtu ověřuje, že zpráva začíná očekávaným bajtem

(0xAA), čímž se signalizuje začátek platné zprávy. Poté čte adresu a délku zprávy a zajišťuje, že je dostupný celý obsah zprávy. Ukládá zprávu do pole a ověřuje, že je správně ukončená (0x55). Na základě čtvrtého bajtu zprávy se pak volá odpovídající funkce pro zpracování celého příkazu.

Pro bližší pochopení jak se celá zpráva zpracovává je níže ukázka jedné z funkcí. Konkretně funkce *handleTPMeasurement*, která simuluje měření napětí na testovacích bodech.

Výpis 5.2: Ukázka zpracování příkazu měření napětí

```
void handleTPMeasurement(byte data[]) {
  delay(1000);
  float floatVoltage = 0;
  if (data[4] == 0) { floatVoltage = 23.5; }
  else if(data[4] == 1) { delay(5000); floatVoltage = 5.2; }
                ...
    else if(data[4] == 3)  {delay(3000); floatVoltage = 24;}
    uint16_t encodedValue = floatTo16Bit(floatVoltage);

    byte hiByte = highByte(encodedValue);
    byte loByte = lowByte(encodedValue);
    byte checksum=0xAA + 0x01 + 0x08 + 0x91 + hiByte + loByte;
    Serial.write(0xAA);
    Serial.write(0x01);
    Serial.write(0x08);
    Serial.write(0x91);
    Serial.write(hiByte);
    Serial.write(loByte);
    Serial.write(checksum);
    Serial.write(0x55);
    lcd.setCursor(0, 0); lcd.print("TPMeasurement");
    lcd.setCursor(0, 1); lcd.print("Voltage:");
    lcd.print(floatVoltage);
}
```

Při implementaci funkce bylo bráno v potaz, že máme více tzv.*test pointů*, které můžou vykazovat v případě poruchy, různá napětí. Na základě hodnoty v pátém bajtu vstupního pole, která určuje, který test point se má testovat, se přiřazuje proměnné floatVoltage specifickou hodnotu napětí, která je pak zakódována do 16bitového formátu. Kontrolní součet pak slouží k ověření integrity dat. LCD displej je aktualizován pro informaci o stavu testu.

6 Validace aplikace

Tato kapitola představuje výstupy aplikace a validace je funkcionality společně s testovaným zařízením. V této práci byla komunikace s testovaným zařízením nahrazena simulátorem (blíže v kapitole 5)

Validace aplikace se prováděla na následujícím příkladu, kde parametry pro simulátor jsou:
Pozn: Číslování pinů se odvíjí od mapovaní pinů v konfiguraci testování získané z databáze

- **Digitální výstupy**:
  $DO0 = 23.5V; DO1 = 0V; DO2 = 24V$
- **Digitální vstupy**:
  $DI0 = HIGH; DI1 = HIGH; DI2 = LOW; DI3 = HIGH; DI4 = HIGH;$
  $DI5 = LOW$
- **Napětí**:
  $VCC = 23.5V; 5V = 5.2V; 4V = 4V; 3V3 = 10V; VDD = 24V$
- **RS-485**:
  Očekává zprávu "$TEST$", pokud úspěšně odpoví OK

Nastavení parametrů v konfiguraci pro testování:

- Tolerance: 1 V
- Limit napětí pro DO: 5 V
- Timeout: 10 s
- Napětí: $VCC = 24V; 5V = 5V; 4V = 4V3V3 = 3.3V; VDD = 24V$

Pro ukázku testovaní byly zvoleny všechny dostupné testovací oblasti. V simulátoru byly implementovány časové zpoždění, které by nastalo při měření na reálné periferii.

Níže, ve výpisech 6.1 a 6.2, naleznete výčet z konzole aplikace při komunikaci se simulátorem a vygenerovanou závěrečnou zprávu, která je vytvořena po úspěšném dokončení všech vybraných testů.

Výpis 6.1: Úkázka komunikace

```
Odeslano: bytearray(b'\xaa\x01\x081\x02\x02\xe8U')                    1
Prijato: bytearray(b'\xaa\x01\x0c\xb1\x00\x00\t\x92\x03U')            2
Dekodovana napeti DO1: 0.0 a 24.5                                    3
```

Výpis 6.2: Závěrečná vygenerovaná zpráva

```
====================Výsledný report testů=====================
                                                              
Dne: 17.05.2024                                               
Čas: 08:42:38                                                 
Testovaná deska: pca9557                                      
                                                              
============================================================= 
Testovací parametry:                                          
Tolerance: 1 V                                                
Limit napětí pro DO: 5 V                                      
Timeout: 10 s                                                 
Napětí:                                                       
   VCC : 24  V                                                
   5V  : 5   V                                                
   4V  : 4   V                                                
   3V3 : 3.3 V                                                
   VDD : 24  V                                                
============================================================= 
Výsledky testů:                                              
                                                              
08:42:28 | Sběrnice - RS-485                    | PROŠEL      
08:42:29 | Digitální vstup - DI0                | PROŠEL      
08:42:29 | Digitální vstup - DI1                | PROŠEL      
08:42:29 | Digitální vstup - DI2                | NEPROŠEL    
08:42:29 | Digitální vstup - DI3                | PROŠEL      
08:42:29 | Digitální vstup - DI4                | PROŠEL      
08:42:29 | Digitální vstup - DI5                | PROŠEL      
08:42:32 | Digitální výstup - Air: 23.0 V       | PROŠEL      
08:42:34 | Digitální výstup - DO0: 0.0  V       | NEPROŠEL    
08:42:36 | Digitální výstup - DO1: 24.5 V       | PROŠEL      
08:42:37 | Napětí - VCC: 23.5  V                | PROŠEL      
08:42:37 | Napětí - 5V:  5.2   V                | PROŠEL      
08:42:38 | Napětí - 4V:  0.0   V                | NEPROŠEL    
08:42:38 | Napětí - 3V3: 10.0  V                | NEPROŠEL    
08:42:39 | Napětí - VDD: 24.0  V                | PROŠEL      
============================================================= 
```

Závěr

Tato bakalářská práce se zabývala vývojem obslužné aplikace pro systém výstupní kontroly elektronických zařízení (desek plošných spojů). Hlavním cílem bylo navrhnout a implementovat aplikaci, která umožní efektivní testování a validaci DPS v rámci výrobního procesu.

V teoretické části práce byly popsány základní pojmy a principy související s testováním DPS, včetně vysvětlení problematiky testovacích scénářů. Dále zde byly čtenářovi přestaveny a následně objasněny nedůležitější typy komunikací, které se v práci využívají. Byla zdůrazněna důležitost spolehlivého testování pro zajištění kvality finálních výrobků a minimalizaci výrobních nákladů.

Analýza požadavků identifikovala klíčové funkční a nefunkční požadavky na aplikaci. Bylo nezbytné zajistit multiplatformní řešení, které umožní provoz na různých operačních systémech, a to včetně Windows, Linux a macOS. Dále byla kladena důraz na výkon, škálovatelnost a bezpečnost aplikace, zejména při komunikaci s cloudovou databází.

V rámci návrhu obslužné aplikace byly definovány testovací scénáře a workflow procesy, které umožní automatizované testování DPS. Byly navrhnuty různé komunikační moduly, včetně sériové komunikace, Bluetooth a REST API, které umožňuji komunikovat s různými typy testovacích zařízení a databázových služeb.

Implementace aplikace probíhala v několika fázích, počínaje vývojem komunikačních modulů, přes tvorbu grafického uživatelského rozhraní (GUI) až po integraci s Arduinem, který byl využit z pozice simulované desky. Důvodem, proč byl využit simulátor, namísto testovací stanice, bylo, že testovací stanice postrádala implementaci komunikace mezi čipe a počítačem. Vzhledem k povaze práce, bylo jako náhradní řešení využito právě Arduino. Při implementaci komunikace tohoto mikrokontroléru, bylo dbáno na to, aby implementace dodržovala všechny natrhnuté protokoly. Tím se tento mikrokontrolér výrazně přiblížil chování reálné testovací stanice.

Celkově tato práce přispěla k vytvoření užitečného nástroje pro výstupní kontrolu elektronických zařízení, který může být dále rozšiřován a upravován dle konkrétních potřeb výrobního procesu. Budoucí práce by mohla zahrnovat integraci dalších komunikačních protokolů, rozšíření podpory pro nové typy testovacích zařízení a optimalizaci výkonu aplikace. Vzhledem k nedostupnosti testovací stanice a potřeby implementace simulátoru, nebylo zprovozněno nahrávaní firmwaru.

Výsledky této práce mají potenciál významně zlepšit kvalitu a efektivitu výrobních procesů v oblasti elektroniky, což může vést k lepší spolehlivosti finálních produktů a snížení výrobních nákladů.

Literatura

[1] STEJSKAL, David. Testovací zařízení elektroniky pro výstupní stanoviště výroby [online]. Brno, 2023 [cit. 2023-10-20]. Dostupné z: https://www.vut.cz/www_base/zav_prace_soubor_verejne.php?file_id=254030.

[2] HART, Pierre. Using Pogo Pins to Add Electrical Connectivity to Your 3D Printed Fixtures [online]. 4 [cit. 2024-05-20]. https://www.javelin-tech.com/blog/2016/10/pogo-pins-3d-printed-fixtures/.

[3] HOUDEK, Cal. Inspection and testing methods for PCBs: An overview [online]. 2016, 7 . [cit. 2023-11-13] Dostupné z: https://static1.squarespace.com/static/62e3c6284ddd8f7896c21906/t/643ea484dd7ee32ad15a78b7/1681826950765/CD%26A+White+Paper+%23401.pdf.

[4] KORNIENKO, D V. Principles of securing RESTful API web services developed with python frameworks [online]. 2021, 12 [cit. 2023-12-20]. Dostupné z: https://iopscience.iop.org/article/10.1088/1742-6596/2094/3/032016/pdf.

[5] JANSEN, Bernard J. The graphical user interface. ACM SIGCHI Bulletin, 1998, 30.2: 22-26.[cit.2023-12-21]. Dostupné z: https://dl.acm.org/doi/pdf/10.1145/279044.279051.

[6] Hardware Test Plan – for complex or mission-critical products [online]. 4 [cit. 2024-01-04]. Dostupné z: https://www.viewpointusa.com/TM/ar/hardware-test-plan-for-complex-or-mission-critical-products/.

[7] VIRDEN, D.W., S.A. NEILD a D.J. WAGG. Emulator-based control for actuator-based hardware-in-the-loop testing. Control Engineering Practice [online]. 2008, 16(8), 897-908 [cit. 2024-05-21]. ISSN 09670661. Dostupné z: https://linkinghub.elsevier.com/retrieve/pii/S0967066107001876.

[8] SHARMA, Pranjal, Anup KUMAR a Naresh KUMAR. Analysis of UART Communication Protocol. 2022 International Conference on Edge Computing and Applications (ICECAA) [online]. IEEE, 2022, 2022-10-13, 323-328 [cit. 2024-05-20]. ISBN 978-1-6654-8232-5. Dostupné z: https://ieeexplore.ieee.org/document/9936199/.

[9] GOMEZ, Carles, Joaquim OLLER a Josep PARADELLS. Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology. Sensors [online]. 2012, 12(9), 11734-11753 [cit. 2024-05-20]. ISSN 1424-8220. Dostupné z: http://www.mdpi.com/1424-8220/12/9/11734.

[10] TOWNSEND, Kevin. Introduction to Bluetooth Low Energy [online]. [cit. 2024-05-21]. Dostupné z: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt.

[11] BLIDH, Henrik. Bleak Documentation [online]. 88 [cit. 2024-05-18]. Dostupné z: https://bleak.readthedocs.io/_/downloads/en/latest/pdf/.

[12] ARM, Jakub. Dokumentace protokolů testovacího systému [online]. 2024 [cit. 2024-05-18]. Dostupné v el. příloze.

[13] ROBERTSON, James a Suzanne ROBERTSON. Šablona pro specifikaci požadavků [online]. 64 [cit. 2024-05-21]. Dostupné z: https://www.volere.org/wp-content/uploads/2018/12/template_cz.pdf.

[14] P.E., Nzerue-Kenneth, Onu F.U., Denis A.U., Igwe J.S. a Ogbu N.H. Detailed Study of the Object-Oriented Programming (OOP) Features in Python. British Journal of Computer, Networking and Information Technology [online]. 2023, 6(1), 83-93 [cit. 2024-05-21]. ISSN 2689-5315. Dostupné z:https://abjournals.org/bjcnit/papers/volume-6/issue-1/detailed-study-of-the-object-oriented-programming-oop-features-in-python/.

[15] APPVEYOR SYSTEMS INC. Flet Introduction [online]. In: . [cit. 2024-05-19].Dostupné z:https://flet.dev/docs/.

[16] LIECHTI, Chris. PySerial Documentation [online]. [cit. 2024-05-19]. Dostupné zhttps://pyserial.readthedocs.io/_/downloads/en/latest/pdf/.

[17] GITHUB, INC. About GitHub and Git [online]. In: . [cit. 2024-05-20]. Dostupné z: [cit. 2024-05-19]. Dostupné zhttps://docs.github.com/en/get-started/start-your-journey/about-github-and-git.

[18] LOUIS, Leo. Working Principle of Arduino and Using it as a Tool for Study and Research. International Journal of Control, Automation, Communication and Systems [online]. 2016, 2016-04-30, 1(2), 21-29 [cit. 2024-05-20]. ISSN 24557889. Dostupné z: https://airccse.com/ijcacs/papers/1216ijcacs03.pdf.

Seznam příloh

A Implementace sériové komunikace

Následující podkapitoly prezentují podrobnější náhled na implementaci sériové komunikace založené na protokolu Skulwep

A.1 Čtení odpovědi

Výpis A.1: Ukázka implementace čtení dat po seriové lince

```
def read_response(self)->bytearray:
    """Čte celou odpověď od testovací desky,
    začíná číst pokud první bajt je start bit."""
    response = bytearray()
    start_time = time.time()

    # Čekání na start symbol
    while (time.time() - start_time) < self.read_timeout:
        if self.ser.in_waiting > 0:
            # Čtení jednoho bajtu
            byte = self.ser.read(1)
            if byte == b'\xAA':  # Start symbol nalezen
                response += byte
                # Čtení zbytku zprávy až do end symbolu
                response+=self.ser.read_until(expected=b'\x55')
                if response[-1] == 0x55:
                    print(f"Přijato:{(response)}")
                    return response
                else:
                    return None
            else:
                time.sleep(0.1)
        return None
```

A.2 Získání měření napětí

Výpis A.2: Ukázka implementace získání měření napětí

```
def get_tp(self, tp_idx)->float:
    """Získá měření napětí na zadaném testpointu."""
    self.send_command(0x11, [tp_idx])
    response = self.read_response()
    if response is None:
        return None
    voltage =(int.from_bytes((response[4:6]), byteorder='big'))
    voltage/=100
    print(f"Dekodované napětí:{voltage}")
    return voltage
```

A.3 Získání stavu sběrnice

Výpis A.3: Získání stavu sběrnice

```
def get_bus_communication(self, message)->str:
    """Získá stav sběrnicové komunikace."""
    self.send_command(0x21, list(message.encode()))
    response = self.read_response()
    if response is None:
        return None
    message_end = response[2] - 2
    decoded_message =(response[4:message_end]).decode('utf-8')
    decoded_message = decoded_message.replace('\x00', '')
    print(f"Dekodovaná zpráva:{decoded_message}")
    return decoded_message
```

B Implementace BLE komunikace

Následující podkapitoly prezentují podrobnější náhled na implementaci BLE komunikace s testovaným zařízením

B.1 Čtení odpovědi

Výpis B.1: Ukázka implementace čtení dat z charakteristiky

```
async def read_characteristic(self, characteristic_uuid,type):
    """Čtení z charakteristy"""
    if not self.client.is_connected:
        return None
    for attempt in range(self.retries):
        try:
            uuid=uuids.normalize_uuid_str(characteristic_uuid)
            value = await self.client.read_gatt_char(uuid)
            result = chardet.detect(value)
            encoding = result['encoding']
            decoded_str = value.decode(encoding)
            if type==float:
                param=type(decoded_str)
            elif type!=str:
                param = type.from_bytes(value,
                        byteorder="little",signed=False)
            else:
                param=decoded_str
            return param
        except Exception as e:
            await asyncio.sleep(self.delay)
        return None
```

B.2 Zápis celočíselné hodnoty do charakteristiky

Výpis B.2: Ukázka implementace zápisu celočíselného parametru

```
async def write_number(self, characteristic_uuid, value)->bool:
    """Zápis int hodnoty."""
    if not self.client.is_connected:
        return False
    try:
        value_bytes=value.to_bytes(length=4,
                    byteorder="little",signed=False)
        uuid=uuids.normalize_uuid_str(characteristic_uuid)
        await self.client.write_gatt_char(uuid,
                            bytearray(value_bytes))
            return True
        except Exception as e:
            return False
```

C Konfigurační soubory

Následující podkapitoly prezentují záznamy uložené v databázové službě a načítané aplikací pomocí REST API

C.1 Konfigurační záznam pro vyskladnění

Výpis C.1: Ukázka záznamu pro vyskaldnění

```
{                                                                    1
   "test": {                                                         2
       "address":"70:B8:F6:24:BD:82",                                3
         "data": {                                                   4
               "ip_adress": {                                        5
                   "UUID_char": "bdf9431f-1b63-11ec...",             6
                   "UUID_serv":"c0ef1d02-0000-1000...",              7
                   "default_value":"192.0.0.1"                       8
               },                                                    9
               "bool": {                                             10
                   "UUID_char": "bdf94e53-1b63-11ec...",             11
                   "UUID_serv":"c0ef1d03-0000-1000...",              12
                   "default_value":"False"                           13
               },                                                    14
               "name_board": {                                       15
                     "UUID_char": "00002A28-0000-1000...",           16
                     "UUID_serv":"0000180A-0000-1000...",            17
                     "default_value":"ASRJ-0025-000002"              18
               },                                                    19
               "max_voltage": {                                      20
                   "UUID_char": "bdf94f00-1b63-11ec...",             21
                   "UUID_serv":"c0ef1d04-0000-1000...",              22
                   "default_value":"32.4"                            23
               },                                                    24
               "hardw_r": {                                          25
                   "UUID_char": "00002A27-0000-1000...",             26
                   "UUID_serv":"0000180A-0000-1000...",              27
                   "default_value":"2.55"                            28
               }                                                     29
                   ...                                               30
```

C.2 Konfigurační záznam pro testování

Výpis C.2: Ukázka záznamu pro testování

```
"pca9557": {
      "baudrate":38400,
      "timeout_limit":10,
      "DO_limit_V":24,
      "criteria_range_V" :1,
      "testing_areas": {
            "RS-485": {
                  "choosen": true,
                  "ids": {
                        "RS-485": "Test"
                  }
            },
            "gpio_input": {
                  "choosen": true,
                  "ids":{
                        "DI0": "0",
                        "DI1": "1",
                        "DI2": "2",
                        ....
                  }
            },
            "ad_output": {
                  "choosen": true,
                  ...
            },
            "voltage": {
                  "choosen": true,
                  "ids":{
                        "VCC": "0",
                        "5V": "1",
                        ...},
                  "ref_value":{
                        "VCC": 24,
                        "5V": 5,
                        "...
```

D Implementace Testovacího modulu

Následující podkapitoly prezentují důležité části implementace testovacího modulu

D.1 Test napěťových úrovní AD-převodníku

Výpis D.1: Ukázka metody pro test napěťových úrovní

```
def _testvoltage(self, tp_idx: int, pin_name: str,
                                    test_param:int) -> bool:
    """Testuje napětí na testovacím bodě."""
    voltage = self.hardware_conn.get_tp(tp_idx)
    lower_limit = test_param - self.tolerance
    upper_limit = test_param + self.tolerance
    passed = voltage is not None
            and lower_limit <= voltage <= upper_limit
    self._log_test_result("Napětí", pin_name, passed,voltage)
    return passed
```

D.2 Test sběrnice

Výpis D.2: Ukázka metody pro test napěťových úrovní

```
def _testRS485(self, message: str, pin_name: str) -> bool:
    """Testuje komunikaci po sběrnici."""
    response = self.hardware_conn.get_bus_communication(message)
    passed = response == "Ok"
    self._log_test_result("Komunikace po sběrnici", pin_name
                                                    , passed)
    return passed
```

D.3 Testovací scénář

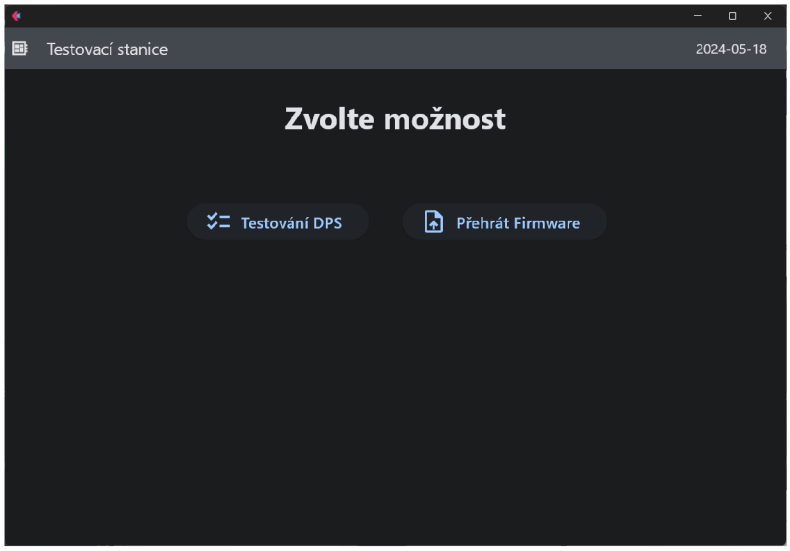
Výpis D.3: Ukázka metody pro test napěťových úrovní

```
 def StartTest(self)->dict:
"""Spustí zvolené testy, podporuje testování více položek."""
    results = {}
    config_areas = self.config[self.board]['testing_areas']
    rs485_passed = False
    if config_areas['RS-485']['choosen']:
    rs485_result = self._testRS485(config_areas['RS-485']
                                ['ids']['RS-485'], "RS-485")
    rs485_passed = rs485_result
    results['RS-485'] = {'RS-485': rs485_result}
    for area, details in config_areas.items():
       if area == 'RS-485' or not details['choosen']:
            continue
        if not rs485_passed and area != "voltage":
            continue
        area_ids = details['ids']
        area_results = {}
        for name, id in area_ids.items():
            result = None
            if area == "voltage":
                result = self._testvoltage(int(id), name,
                                    details["ref_value"][name])
            elif area == "gpio_input":
                result = self._testDI(int(id), int(id), name)
            elif area == "ad_output":
                result = self._testDO(int(id), int(id), name)
                area_results[name] = result
            results[area] = area_results
        self._generate_report()
        return results
```

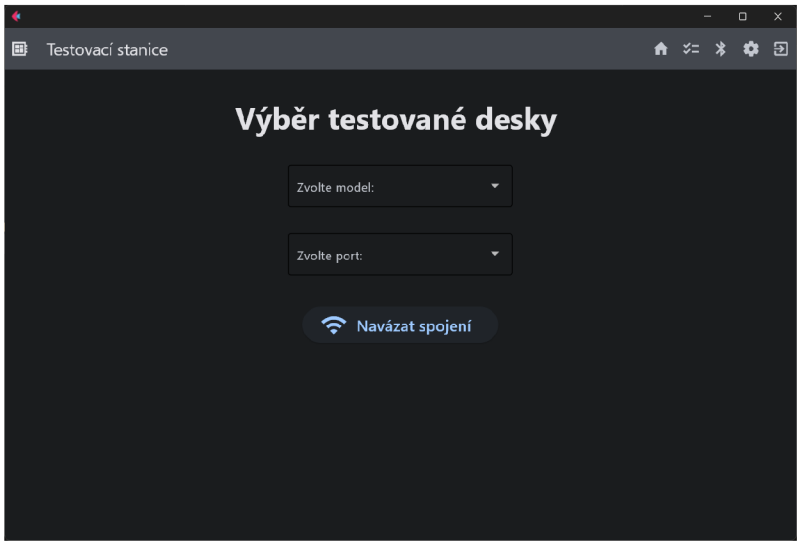
E Uživatelské rozhraní

Následující podkapitoly slouží pro prezentaci uživatelského rozhraní obslužné aplikace

E.1 Úvodní a připojovací obrazovka

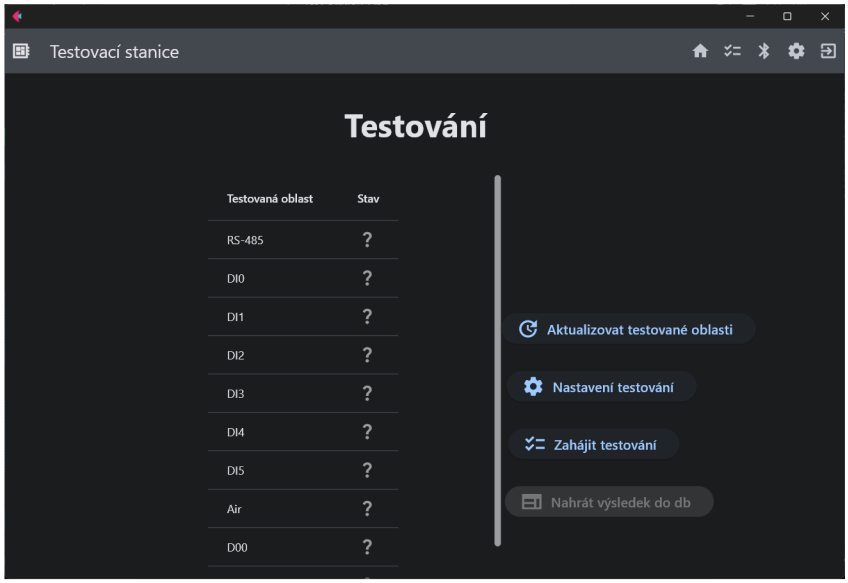


Obr. E.1: Uživatelské rozhraní - úvodní obrazovka



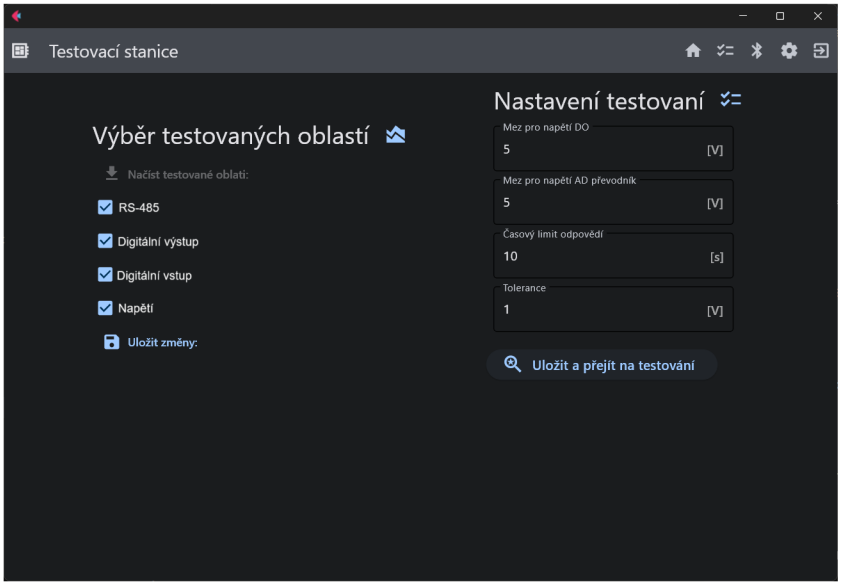
Obr. E.2: Uživatelské rozhraní - Připojovací obrazovka

E.2 Testovací obrazovka



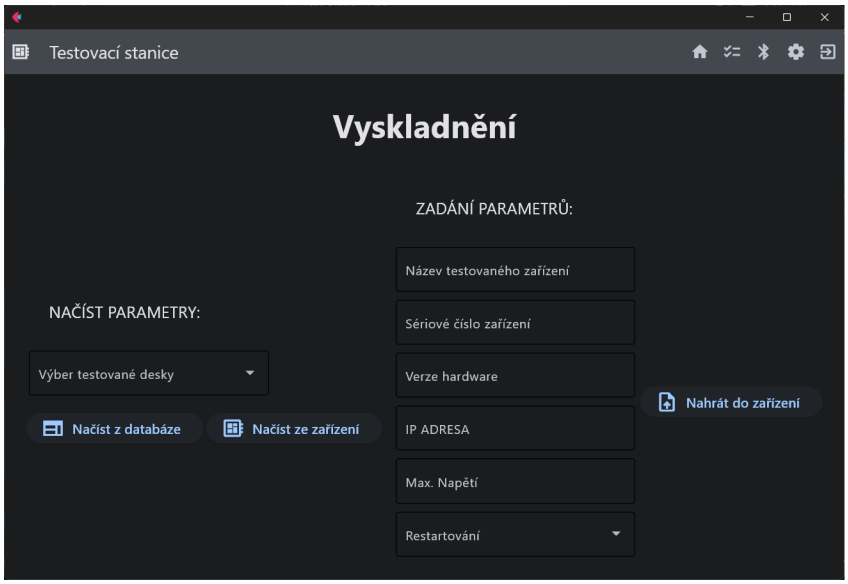
Obr. E.3: Uživatelské rozhraní - Testovací obrazovka

E.3 Obrazovka přizpůsobení testování



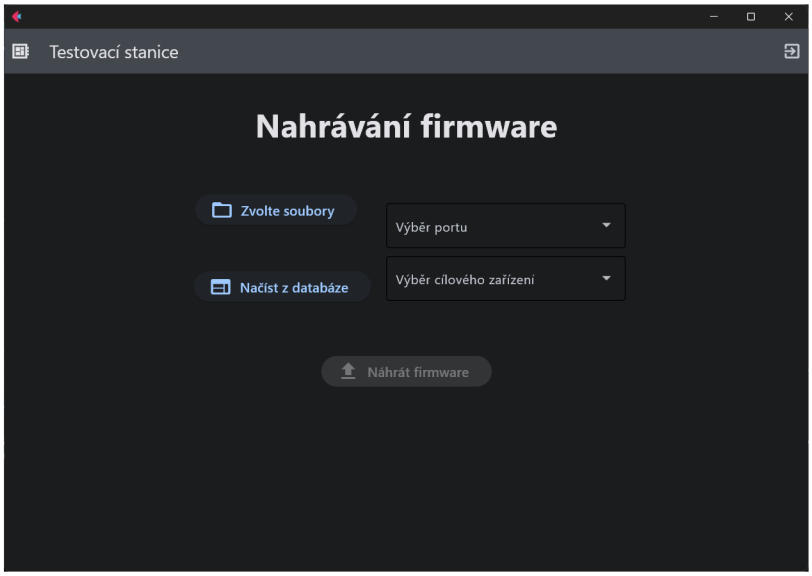
Obr. E.4: Uživatelské rozhraní - obrazovka přizpůsobení testování

E.4 Obrazovka vyskladnění



Obr. E.5: Uživatelské rozhraní - Obrazovka vyskladnění

E.5 Obrazovka firmware



Obr. E.6: Uživatelské rozhraní - Obrazovka firmware

F Obsah elektronické přílohy

- / .2 Komunikace\ — Dokumenty popisující komunikace
  - Popis_protokolů_0_9.pdf
  - ble_characteristic.txt
- Simulator_firmware\ — Zdrojový kód pro Arduino
  - test_station_firmware.ino
- Test_Station_App\ — Kořenový adresář obslužné aplikace
  - assests\ — Konfigurační soubory pro databázy
    - ble.json
    - data.json
  - logs\ — Výsledná zpráva testování
    - Test_Report_2024-05-21_163146.log
  - views\ — Zdrojové kódy aplikace
    - conn_adapter\ — Zdrojové kódy komunikačních modulů
      - __init__.py
      - api_client.py
      - ble_connection.py
      - globalvars.py
      - hardware_connection.py
      - test_module.py
    - user_controls\
      - app_bar.py
      - app_bar_firmw.py
      - app_bar_menu.py
    - __init__.py
    - export_view.py
    - fimware_view.py
    - home_view.py
    - menu_view.py
    - settings_view.py
    - test__view.py
    - viewsrouting.py
  - .gitignore
  - README.md
  - main.py
  - requirements.txt
- Text\ — Elektronická verze práce
  - BP_tlustos_240458.pdf