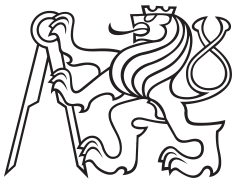
Diplomová práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra radioelektroniky

Systém pro funkční testování desek plošných spojů

František Beránek

Školitel: doc. Ing. Stanislav Vítek, Ph.D.
Květen 2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Beránek** Jméno: **František** Osobní číslo: **492027**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra radioelektroniky**

Studijní program: **Elektronika a komunikace**

Specializace: **Technologie internetu věcí**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Systém pro funkční testování desek plošných spojů

Název diplomové práce anglicky:

PCB Functional Testing System

Pokyny pro vypracování:

1) Navrhněte systém pro provádění funkčních testů nově vyrobených desek plošných spojů (DPS).
2) Systém bude disponovat co největším množstvím standardních rozhraní (jako jsou GPIO, analogové vstupy a výstupy, sběrnice jako UART, SPI, I2C) a bude v tomto ohledu rozšiřitelný.
3) Pro každý typ testované DPS bude možné vytvořit nový průběh testu definovaným a zdokumentovaným způsobem v závislosti na funkcích dané DPS. O průběhu bude na konci testu vytvořen report.
4) Součástí zpracování bude i vzorové využití systému pro libovolnou DPS (včetně SW pro testovanou DPS, bude-li potřeba).

Seznam doporučené literatury:

[1] WHITE, Elecia. Making Embedded Systems: Design Patterns for Great Software. " O'Reilly Media, Inc.", 2011.
[2] MALÝ, Martin. Data, čipy, procesory: vlastní integrované obvody na koleni. CZ. NIC, zspo, 2020.
[3] REORDA, Matteo Sonza; PENG, Zebo; VIOLANTE, Massimo (ed.). System-level test and validation of hardware/software systems. Springer Science & Business Media, 2006.

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Stanislav Vítek, Ph.D. katedra radioelektroniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **06.02.2024** Termín odevzdání diplomové práce: **24.05.2024**

Platnost zadání diplomové práce: **21.09.2025**

doc. Ing. Stanislav Vítek, Ph.D. podpis vedoucí(ho) práce	doc. Ing. Stanislav Vítek, Ph.D. podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. Datum převzetí zadání

Podpis studenta

Poděkování

Rád bych poděkoval firmě TSE spol. s.r.o. za možnost práce na zajímavých projektech a kolegům Ing. Radku Lustovi a Bc. Radimu Sejkovi za cenné rady. Dále bych rád poděkoval vedoucímu práce za příjemnou spolupráci při tvorbě bakalářské a diplomové práce i za veškerou výuku v průběhu studia. V neposlední řadě bych rád poděkoval své rodině a přítelkyni za ohromné množství trpělivosti jež se mnou měli, mají a snad i nadále budou mít.

Prohlášení

„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.“

V Praze, 16. května 2024

Abstrakt

Práce se zabývá návrhem a realizací systému pro automatické funkční testování DPS. Systém se skládá z jednodeskového počítače Raspberry Pi 4B a vlastního rozšiřujícího modulu. Tento modul slouží jako rozhraní mezi Raspberry Pi a testovanou DPS. Testování je založeno na nástroji Robot Framework. Veškerý vzniklý SW v této práci je tvořen pomocí jazyka Python. Hlavním cílem projektu je modularita systému pro možnost přizpůsobení různým testovaným DPS. Součástí práce je i ukázka použití pro zvolenou DPS.

Klíčová slova: testování, automatizace, Raspberry Pi, Python, Robot Framework, DPS

Školitel: doc. Ing. Stanislav Vítek, Ph.D.

Abstract

This thesis describes the design and implementation of a system for automatic functional testing of PCBs. The system consists of a Raspberry Pi 4B single-board computer and custom expansion module. This module serves as an interface between the Raspberry Pi and the PCB under test. Testing is based on the Robot Framework. All created SW in this work is created using the Python language. The main goal of the project is the modularity of the system for the possibility of adaptation to various tested PCBs. The work also includes a demonstration of use for the selected PCB.

Keywords: testing, automation, Raspberry Pi, Python, RobotFramework, PCB

Title translation: PCB Functional Testing System

Obsah

Obrázky

Seznam zkratek

ADC Analogově digitální převodník (analog to digital converter)

AOI Automatická optická inspekce (automatic optical inspection)

API Aplikační programové rozhraní (application programming interface)

DAC Digitálně analogový převodník (digital to analog converter)

DPS Deska plošných spojů

EEPROM Elektricky vymazatelná programovatelná paměť pouze pro čtení (electrically erasable programmable read-only memory)

GUI Grafické uživatelské rozhraní (graphical user interface)

GPIO Univerzální vstupní/výstupní pin (general purpose input/output)

JSON Datový formát (JavaScript Object Notation)

HW Hardware

HWAL Vrstva hardwareové abstrakce (hardware abstraction layer)

HWDL Vrstva hardwareové definice (hardware definition layer)

I2C Typ synchronní sériové sběrnice (Inter-integrated circuit)

ICT Metoda pro testování DPS (in-circuit testing)

IT Informační technologie (information technologies)

MCU Mikrokontroler (microcontroller unit)

OS Operační systém (operating system)

PC Osobní počítač (personal computer)

PSP Vlastnosti závislé na periferii (peripherie specific properties)

PTC Pozitivní teplotní koeficient (positive temperature coefficient)

PWM Pulzně šířková modulace (pulse width modulation)

RF Robot Framework

SPI Typ synchronní sériové sběrnice (serial peripheral interface)

SSH Protokol vzdálené příkazové řádky (Secure Shell)

SW Software

TSP Vlastnosti závislé na typu (type specific properties)

UART Typ asynchronní sériové sběrnice (universal asynchronous receiver/transmitter)

USB Typ asynchronní sériové sběrnice (universal serial bus)

Kapitola 1
Úvod

Při výrobě jakéhokoliv produktu je zásadní kontrola všech jednotlivých komponent. Čím dříve je závadná komponenta nalezena tím lépe, protože případná demontáž a výměna v pozdější fázi kompletace stojí ve výrobě čas i peníze. Toto platí i pro výrobu DPS.

Pro testování DPS je mnoho metod s různou úrovní náročnosti (ať už z pohledu doby trvání, tak i technologické) i pravděpodobností odhalení chyb. Každá z metod je odlišná a dokáže identifikovat různé chyby. Některé z nich ovšem často vyžadují zásah obsluhy v případě nálezu a díky tomu může dojít k schválení chybného kusu. Dalším problémem je, že lze relativně snadno narazit na neodhalitelné závady. Proto bývá požadován i funkční test DPS. Ten často sestává z oživení (první připojení napájení a nahrání SW, je-li potřeba) a připojení do systému, ve kterém bude výrobek pracovat, případně jeho náhrady.

Příkladem takové náhrady mohou být jednoúčelová plata. Ty se skládají ze sady elektroniky (kabelové svazky, DPS a případně i elektromechanické podsestavy), která ve výsledném zařízení testovanou DPS obklopuje a tím je simulováno výsledné zařízení. Obsluha pak během testů připojí testovanou DPS a dle návodu vyzkouší všechny funkce. Tento přístup je časově velmi náročný, jelikož obsluha musí nasimulovat spoustu stavů externích periferií. Proto začaly snahy o větší automatizaci procesu. Tento projekt popisuje vznik zařízení, které by mělo takovéto automatizované testy provádět.

Cílem práce je vytvoření univerzálního systému pro provádění funkčních testů DPS. Průběh testu by mělo být možné definovat zdokumentovaným způsobem, tzn. mělo by být relativně snadné zaškolit pracovníka s alespoň základní znalostí HW a SW k psaní nových testů a údržbě již existujících. k přizpůsobení zařízení pro novou DPS bude dále potřeba vytvořit i nový HW sloužící k propojení. Tato část by měla být také co nejjednodušší.

Součástí práce by měla být ukázka použití zařízení pro libovolnou DPS, která by měla sloužit jako vzor pro další využití.

V kapitole 2 jsou uvedeny vybrané metody používané pro testování DPS. Rozbor zadání je proveden v kapitole 3. Praktická část práce začíná kapitolou 4, kde jsou uvedeny zvažovné architektury systému včetně architektury SW v podkapitole 4.4. Konfigurací použitého jednodeskového počítače se zabývá kapitola 5. Implementace SW je popsána kapitolou 6. Dále je popsána zvolená

ukázková testovaná DPS v kapitole 7 a návrh propojovacího HW v kapitole 8 a testovacího skriptu v kapitole 9. Dosažené výsledky shrnuje kapitola 10 a dále jsou v kapitole 11 diskutována některá možná budoucí rozšíření.

Kapitola 2
Metody pro testování DPS

V této kapitole budou popsány některé významné metody používané pro testování DPS. Budou zmíněny klady a zápory jejich použití. Jako poslední bude popsána metoda funkčních testů, kterou má provádět zařízení vytvářené v rámci této práce, a bude zdůvodněna volba této metodiky.

2.1 Vizuální prohlídka

Tato metoda je technologicky z pochopitelných důvodů nejjednodušší. Vizuální prohlídku provádí pracovníci ve výrobě v průběhu celého procesu. z principu jde samozřejmě o metodu s velmi nízkou spolehlivostí, ale některé závady lze pomocí ní odhalit včas a velmi snadno (např. chybějící nebo špatně orientované komponenty atd.). Naopak existuje množství závad, které ani pečlivá prohlídka neodhalí (např. špatné hodnoty komponent bez jednoznačného potisku, špatné pájené spoje pro některá pouzdra atd.). Velký vliv má pak samozřejmě i lidský faktor.

2.2 AOI

Další metodou je AOI (automatická optická inspekce) [1]. Tento typ testu je založen na visuálním srovnání vzoru a kontrolovaného kusu. Jde tedy o automatizaci předešlé metody, a proto se používají k odhalování stejných závad. s automatizací roste pravděpodobnost zachycení vadného kusu, ale ta stále není stoprocentní. Chyby, které systém nalezne, jsou předávány obsluze k vyhodnocení, při kterém stále může dojít k selhání. Pro nalézání závad, které nelze vidět běžným způsobem (např. připájení u pouzder typu BGA nebo jiných typů pouzder s vývodem pod komponentou) se využívá kontrola pomocí rentgenového záření, která je nákladnější a používá se převážně při prvotním nastavení technologie, případně až při konkrétním podezření.

2.3 Flying probes

Mezi často používané metody patří například testování pomocí tzv. létajících sond (flying probes) [2], při které se sondy připevněné na robotickém

rameni nakontaktují přímo na cestu na DPS buďto pomocí k tomu určeného testovacího bodu nebo pomocí plošky součástky, pokud je to možné. z toho vyplývá, že s touto metodou je potřeba počítat již při návrhu DPS. Sonda poté měří parametry obvodu v daném místě a srovnává je se vzorem. Pro větší DPS může jít o časově náročnější proces (záleží na počtu sond konkrétního zařízení). Tento testovací přístroj je plně konfigurovatelný a je proto vhodný i pro použití v menších sériích.

2.4 In-Circuit testování (ICT)

Tento typ testu je v zásadě podobný předchozímu, ale jehly jsou rozmístěny pevně dle testované DPS a všechny testovací body jsou připojeny najednou [2]. Vzhledem k tomu, že je vždy potřeba vytvořit pro každou testovanou DPS nové propojovací rozhraní, které bývá komplikované a tedy i drahé (je potřeba zajistit správný přítlak testovacích jehel), je tento přístup vhodný spíše pro větší série. Díky pevnému rozmístění kontaktních jehel a jejich hromadnému připojení je testování oproti létajícím jehlám výrazně rychlejší.

2.5 Funkční testování

Metodou, která bude implementována v rámci této práce je funkční testování. Jak již bylo zmíněno v úvodu, při této metodě je testovaná DPS připojena buďto přímo do výsledného produktu, nebo jeho co nejpodobnější náhrady a ideálně by měly být otestovány všechny funkce této DPS. Oproti testování ICT nebo létajícími jehlami by mělo jít o ještě spolehlivější metodu (např. z vnějšího ohledání se může integrovaný obvod jevit jako bezproblémový, ale je vadný).

Při použití zmíněného testovacího plata je proces testování velmi náročný, protože obsluha musí nasimulovat velké množství stavů, jinak je test neúplný. Proto je i u funkčních testů velká snaha o co největší automatizaci.

Funkční testy jsou nejvíce významné pro menší série, kdy je každá vadná DPS ještě závažnější problém, jelikož jsou DPS často objednávány na přesný počet kusů a každá závada pak může znamenat větší zpoždění ve výrobě.

Zařízení pro provádění automatizovaných funkčních testů není na trhu mnoho a bývají drahá a složitá. Vývoj vlastního zařízení také přináší velký vhled do jeho funkce a tím pádem možnost jeho upravení na míru případným novým požadavkům. Další nespornou výhodou je fakt, že není potřeba se spoléhat na vybavení dodavatele DPS a i případná změna výrobce by měla být mnohem snazší.

Kapitola 3

Rozbor zadání

V zadání jsou jasně definovana rozhraní, které musí zařízení být schopné testovat. Jde o následující: GPIO, UART, I2C, SPI, ADC, DAC. Výhodou by dále byla i možnost použití sběrnice USB. Díky různým převodníkům je možné výsledný seznam podporovaných periferií do budoucna i dále rozšiřovat. Mělo by být přihlédnuto i k možným rozdílným napěťovým úrovním zařízení a testované DPS.

Zásadním požadavkem napříč celým projektem je modularita a rozšiřitelnost, tzn. počet zmíněných rozhraní by mělo být možné jednoduše měnit.

Při řešení koncepce by mělo dále být přihlédnuto k funkcím, které nejsou udávané v zadání, ale s přihlédnutím ke zkušenostem z předchozí praxe by mohli být v budoucnu užitečné. Do této kategorie spadá například možnost programování SW pro mikrokontroler na testované DPS. Jak zadání naznačuje, může být pro některé DPS potřeba součinnost testovaného HW v průběhu testu. Takováto funkce sice nebude součástí práce, ale je výhodou, pokud je na ni pamatováno.

Jelikož by samotné testování měl provádět pracovník s nižší odborností, je potřeba ovládání tvořit co nejjednodušší a nejintuitivnější. Bližší požadavky na uživatelské rozhraní nejsou popsány.

Podobně je potřeba smýšlet i u koncepce zbytku přístroje, tj. rozhraní směrem k vývojáři zařízení. Ideální by bylo, aby větší část HW i SW byla pevně definována a nemuselo dojít k rozsáhlému vývoji nového zařízení pro každou testovanou DPS.

Kapitola 4

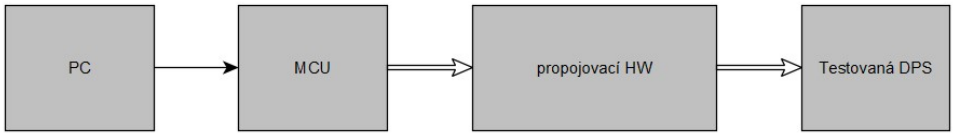
Návrh architektury systému

Na začátku celého projektu bylo utvořeno několik konceptů celého systému. Jednotlivé návrhy architektury se liší rozdělím kompetencí mezi jednotlivými bloky. Vybrané koncepce budou popsány a porovnány v následující kapitole.

4.1 Varianta s PC

Tento návrh počítal s použitím PC s obslužnou aplikací. Pomocí této aplikace by bylo možné spouštět testy, ukládat jejich výsledky, případně testy pomocí grafického nebo textového rozhraní i editovat. Se zbytkem systému by PC komunikovalo pomocí sběrnice USB.

Prvotním záměrem bylo, aby zbytek zařízení tvořila jediná DPS s obsluhujícím MCU. To by přijímalo instrukce od PC a na základě těchto instrukcí obsluhovalo periferie. Jelikož by měl blok MCU být vždy stejný včetně SW, bylo rozhodnuto o rozdělení na dvě DPS, kde by jedna obsahovala samotné MCU a druhá potřebné obvody rozhraní. Díky tomu by mělo být možné minimalizovat výrobní náklady zařízení například při menších změnách testované DPS nebo díky použití bloku MCU z testovacího zařízení pro již dále nevyráběnou DPS. Blokové schéma je naznačeno na obrázku 4.1.



Obrázek 4.1: Architektura s PC

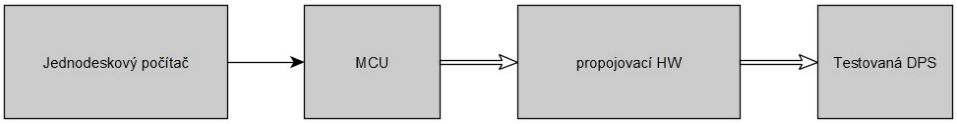
Blok propojovací HW je společný pro všechny koncepce. Tento blok slouží jako rozhraní mezi testovanou DPS a zbytkem systému. Mělo by jít o sestavu prvků jako jsou odporové děliče pro ADC, ochranu GPIO, převodníky logických úrovní, GPIO expandery a dalších na základě potřeb testované DPS. Dále by měl tento blok obsahovat nějaký prvek pro možnost identifikace. Pomocí něj by mělo být možné určit k jakému typu testované DPS patří a tím by mělo být zabráněno spuštění špatného testu.

Tenká šipka na obrázku 4.1 naznačuje sběrnici USB a širší bez výplně kombinaci různých rozhraní dle zadání (GPIO, UART, SPI...).

Hlavní nevýhodou systému je potřeba komunikace s dodavatelem DPS při nasazení zařízení. Je totiž potřeba kromě předání fyzického přípravku i předání obslužné aplikace. Počítali bychom při tom s využitím PC dodavatele a instalací ve spolupráci s jeho IT oddělením.

4.2 Varianta s jednodeskovým počítačem

Tato koncepce je prakticky identická s předchozí. Jediným rozdílem by bylo použití jednodeskového počítače na rozdíl od PC. Tím by se mělo urychlit případné nasazení dalšího zařízení na jiné pracoviště (k výrobci DPS by byl dodávaný kompletní systém s minimálními nároky na jeho součinnost). Blokové schéma je zobrazeno na obrázku 4.2.



Obrázek 4.2: Architektura s jednodeskovým počítačem

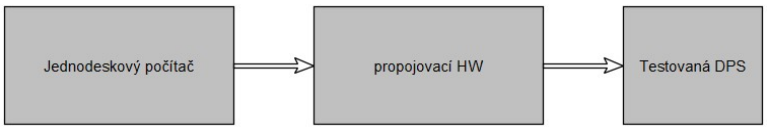
4.3 Varianta s jednodeskovým počítačem bez bloku MCU

Poslední popisovaná varianta vychází z předchozí, ale nepoužívá blok MCU. Namísto toho jsou přímo využity periferie jednodeskového počítače. Tím se výrazně snižuje komplexnost HW celého systému. Blokové schéma je naznačeno na obrázku 4.3.

Po dokončení SW pro jednodeskový počítač je možné vygenerovat image operačního systému a nasazení nového kusu zařízení se tak výrazně zjednodušuje. Nevýhodou může být fakt, že je potřeba o něco více času věnovat návrhu SW pro jednodeskový počítač, který zařizuje podstatně více funkčnosti než v předchozích konfiguracích.

Další nevýhodou tohoto řešení může být náročnější implementace testů s požadavkem na přesné časování (např. proměření strmosti hran signálů). Tento problém je možné řešit například pomocí MCU umístěném na propojovacím HW. Toto MCU může pak mít SW při kterém provede takovéto testování na povel jednodeskového počítače.

Tato varianta bude dále použita. Pro vývoj zařízení byl použit jednodeskový počítač Raspberry Pi ve verzi 4B.

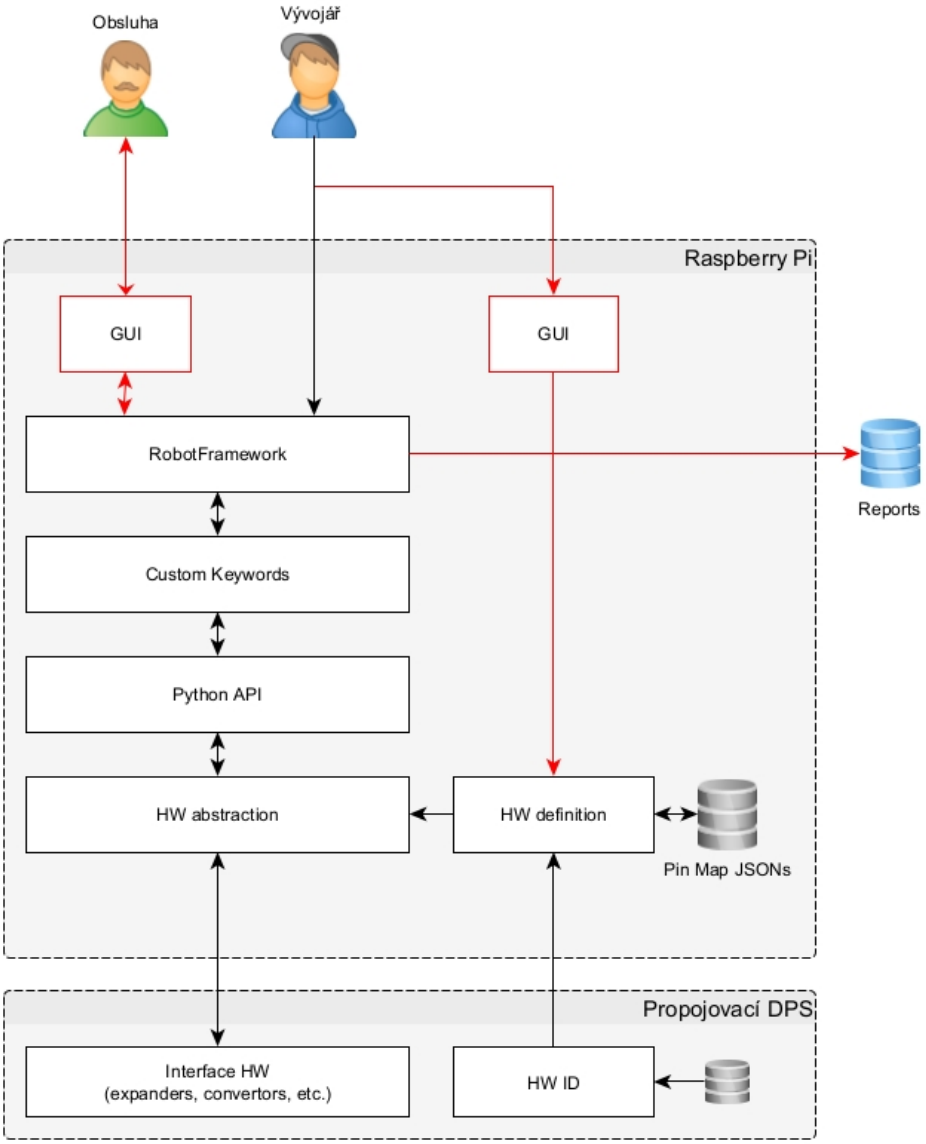


Obrázek 4.3: Architektura s jednodeskovým počítačem bez bloku MCU

4.4 Architektura SW

Vzhledem ke komplexnosti zařízení z pohledu SW bylo potřeba před jeho tvorbou vytvořit rozdělení SW do jednotlivých funkčních bloků. Ty tvoří určité vrstvy celého systému a budou dále popsány.

Bloková architektura je znázorněna na obrázku 4.4. Červeně jsou označeny rozhraní nebo bloky, které nejsou přímo součásti zadání, ale mohli by být nad jeho rámec realizovány.



Obrázek 4.4: Architektura SW

4.4.1 HW ID

Tento blok je z jedné strany připojen k paměťovému prvku. Ten může být různého typu. Preferovaná možnost je paměťový čip s pamětí typu EEPROM. Další možnost může být i sada přepínačů.

Z druhé strany je tento blok připojen k řídícímu počítači a na vyžádání mu předává ID.

4.4.2 Interface HW

Tento blok znázorňuje různé prvky potřebné k bezpečnému propojení řídícího jednodeskového počítače a testované DPS. Spolu s HW ID tvoří propojovací DPS popsanou výše.

4.4.3 HW Abstraction

Vzhledem k požadavku na univerzálnost a modulárnost celého zařízení je potřeba vytvořit alespoň základní míru abstrakce nad HW (HW Abstraction Layer, dále HWAL). Vrstvy SW nad touto by měli přistupovat naprosto stejným způsobem například k digitálnímu pinu na pin headeru Raspberry Pi tak i k pinu na expanderu. Tato vrstva bude pravděpodobně nejnáročnější na implementaci, jelikož je potřeba zaručit robustnost a poskytnout co nejširší nabídku kompatibilních periferií pro možnost škálování na základě potřeb dané testované DPS a to vše při zachování snadné čitelnosti a rozšířitelnosti zdrojového kódu.

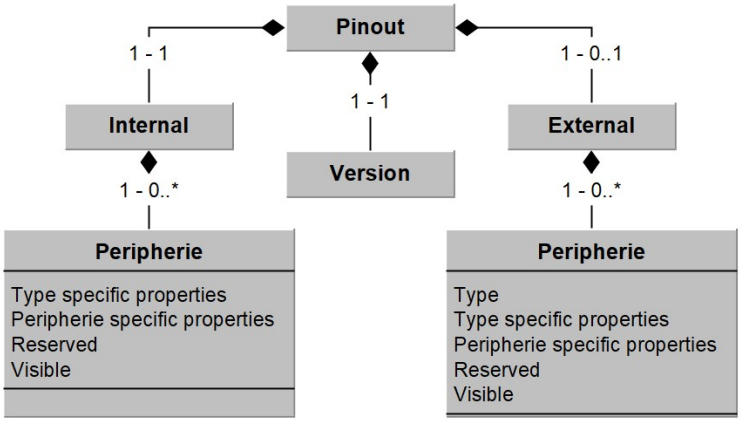
4.4.4 HW Definition

Tento blok označovaný jako HW Definition Layer (dále HWDL) by se měl spustit vždy před startem testu. Jeho úkolem je přečíst ID připojeného HW a dle něj vytvořit vrstvu HW abstrakce. Za tímto účelem je potřeba vytvořit pro každé ID soubor, který bude definovat použité periferie. v tomto souboru by mělo být zapsáno o jakou periferii jde a jakým způsobem je připojena.

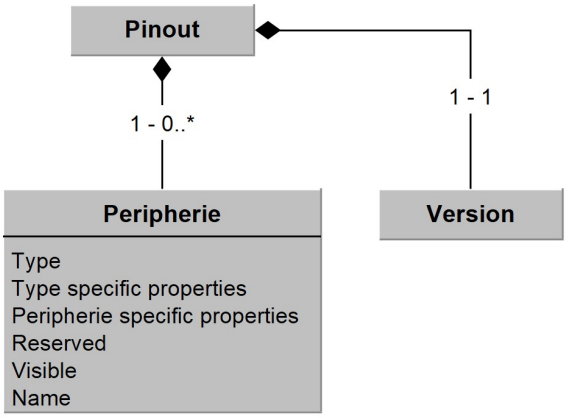
4.4.5 PinMap JSON

Tento blok je úložiště obsahující soubory popsané v předchozím odstavci. Pro snadnější implementaci a přehlednost souboru byl zvolen formát JSON. Struktura souboru by mohla být popsaná schématem 4.5.

Celá třída Pinout je v souboru rozdělená na části internal a external. Internal jsou vlastní periferie jednodeskového počítače a external jsou periferie, které jsou připojeny nepřímo (např. piny expanderu, externí ADC a DAC atd.). k tomuto rozdělení došlo kvůli potřebě definování interních periferií v první řadě tak, aby bylo možné dále použít tyto periferie k definování externích (např. je potřeba v první řadě definovat třídu pro obsluhu I2C sběrnice, ke které je dále připojen expander).



Obrázek 4.5: Struktura PinMap souboru



Obrázek 4.6: Struktura PinMap souboru, bez oddělení

Další možností je spojení interních i externích periferií do jediné třídy. Schéma se pak zjednoduší dle 4.6.

V takovém případě musí vývojář tvořící tento soubor zajistit, že periferie jsou definovány ve správném pořadí. Tento problém by mohl být řešen volitelným GUI, které by pomáhalo s tvorbou pinMap souboru s definicí periferií. Vzhledem k možné délce souboru při větším množství periferií by byla manuální tvorba velmi náročná, což je další motivace k tvorbě GUI. Ve finální implementaci bude počítáno pouze s druhou variantou struktury souboru.

Periferie (digitální/analogový výstup/vstup, UART atd.) je definována několika atributy. v první řadě je definován typ (např. pro digitální výstup může být typu interní pin, pin na I2C expanderu atd.). V závislosti na typu periferie je počítáno s dalšími argumenty (označeno jako `Type specific properties`, dále TSP), které odpovídají argumentům konstruktoru třídy abstrakce pro daný typ periferie (např. číslo pinu pro interní digitální výstupní pin). Argument `Peripherie specific properties` (dále PSP) je závislý na periferii (např. výchozí stav pro výstupní digitální pin). Hodnota `Visible` by

měla nabývat stavu `False` pro periferie, které nejsou přímo řízené testovacím skriptem (např. pin pro ovládání chip select signálu SPI sběrnice), nebo `True` v opačném případě (tj. pro periferie přímo ovládané testovacím skriptem).

Hlavní třída celého pinMap souboru (v diagramu označeno jako `Pinout`) dále obsahuje hodnotu `version`. Díky této hodnotě by v budoucnu mělo být možné upravit podobu souboru bez nutnosti změny všech již vytvořených souborů.

4.4.6 Robot Framework

Dalším krokem při tvorbě celého systému je vytvoření nástroje pro tvorbu testů samotných. Ten má být dle zadání definovaný a zdokumentovaný. Tvorba takového systému od základu by byla velmi náročná a navíc zbytečná. Mezi množstvím hotových nástrojů pro automatizaci a testování byl vybrán Robot Framework. Hlavním důvodem k jeho použití byla předchozí zkušenost s tímto nástrojem. Dále byla rozhodujícím faktorem této volby strmá učící křivka (RF skripty jsou snadno čitelné).

Robot Framework (dále jen RF) je open-source nástroj pro automatizaci a testování [3]. Celý framework je založen na jazyce python a v tomto jazyce je i možné psát vlastní knihovny [4]. Díky tomu je možné využít tento framework pro téměř libovolný účel.

Vzhledem k jednoduché syntaxi je snadné zaškolit pro psaní testovacích skriptů i méně zkušeného pracovníka. Přímo RF má navíc na svých stránkách množství interaktivních tutoriálů a díky široké uživatelské základně je k dispozici i velké množství informací na oficiálním fóru projektu.

4.4.7 Python API

Jak bylo zmíněno, RF je možné dále rozšiřovat pomocí vlastních knihoven napsaných v jazyce Python a tím tvořit vlastní klíčová slova a používat je při psaní testů. v rámci projektu je potřeba vytvořit takové API, které bude přistupovat k vytvořené HWAL a HWDL přímo z kódu testu.

4.4.8 Custom Keywords

V tomto bloku by měli být vytvořeny vlastní klíčová slova pro Robot Framework pro často využívané funkce. Mělo by se jednat o kombinace klíčových slov RF a vytvořeného Python API, které budou tvořit elementární testy a bude je tak možné opakovaně používat.

4.4.9 Reports

Ve výchozím stavu se reporty vytvořené na konci testu v RF ukládají do pracovního adresáře [5]. Pro použití v tomto projektu by bylo výhodné mít možnost ukládat výsledky testů dohledatelným a vývojáři přístupným způsobem pro možnost zpětného vyhodnocení výsledků. k tomu by mělo sloužit vzdálené úložiště. Tento blok je volitelnou součástí projektu.

4.4.10 GUI

Další volitelnou součástí projektu je grafické uživatelské rozhraní pro editaci a spouštění testů a zobrazení a ukládání reportů. Pomocí grafického rozhraní by mělo být editování testů jednodušší, ale již použití RF a zdokumentovaného API splňuje zadání.

Jelikož samotné spouštění testů by měla provádět obsluha s relativně nízkou odborností, měl by tento proces být co nejjednodušší. z tohoto důvodu by bylo dobré vytvořit aplikaci, která by po spuštění systému zjistila ID připojeného HW a vytvořila alespoň jednoduché okno pro spuštění testu a dále signalizovala jeho průběh.

4.4.11 Uživatel

V tomto návrhu jsou rozlišovány dva druhy uživatelů: obsluha a vývojář. Obsluha má za úkol připojit testovanou DPS a spustit test, ideálně pomocí GUI. Vývojář má možnost přímo vytvářet nebo editovat testy a definice HW.

Zatímco obsluha musí být fyzicky přítomná u zařízení, u vývojáře je vysoce žádoucí umožnit i určitou formu vzdáleného přístupu.

Kapitola 5

Konfigurace Raspberry Pi

Jako operační systém byl použit Raspberry Pi OS ve verzi 64-bit [6]. Vývoj probíhal v dedikovaném virtuálním prostředí, aby bylo zabráněno konfliktům verzí použitých knihoven.

Po celou dobu vývoje byl používán nástroj Visual Studio Code Remote Development. Jde o rozšiřující balíček pro editor Visual Studio Code, který umožňuje editování zdrojového kódu a přístup k terminálu na vzdáleném zařízení [7]. V tomto případě bylo použito připojení pomocí protokolu SSH. Díky tomuto nástroji bylo možné veškerou práci provádět na běžném PC a zároveň interagovat s cílovým HW.

Použitá rozhraní (SSH, UART, I2C a SPI) byla aktivována pomocí nástroje pro konfiguraci raspi-config [8]. Zvažováno bylo i použití vzdálené plochy, od čehož se nakonec ustoupilo díky použití již zmíněného Visual Studio Code Remote. Na hotových zařízeních, která budou předávána k výrobci DPS by mělo dojít k deaktivaci SSH.

Níže popsané uživatelské GUI by mělo být spuštěno hned po startu zařízení. Jelikož jde o grafickou aplikaci, je potřeba, aby byl grafický systém OS již v provozu. Proto byla využita metoda označována jako autostart [9].

Kapitola 6

Implementace SW

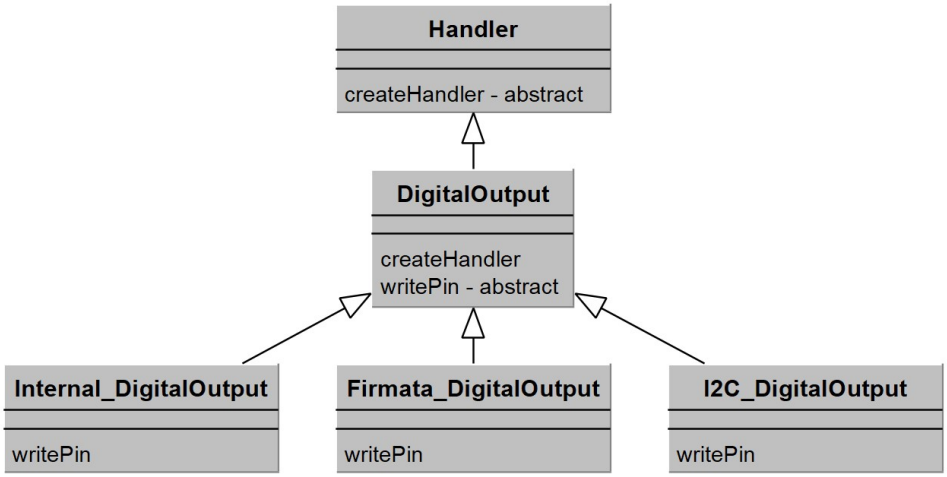
Vzhledem k použité platformě a množství dostupných informací byl veškerý software tvořen pomocí jazyka Python. Dalším rozhodujícím faktorem je i snadné propojení mezi vytvořeným SW a RF.

6.1 HW Abstraction Layer

Implementace byla zahájena směrem od nejnižších vrstev k vyšším.

Ve vrstvě HWAL jsou ve velkém rozsahu použity abstraktní třídy. Ty jsou výhodné pro tvorbu šablon pro třídy, které po nich dědí [10]. Metody abstraktních tříd nejsou přímo vykonávány, ale musí být definovány dědící třídou. Díky tomu je možné definovat univerzální rozhraní (např. třída DigitalOutput definuje metodu `writePin`, kterou musí obsahovat všechny implementace pro periferii tohoto typu).

Každá třída pro obsluhu periferie dědí po třídě Handler. Tato třída definuje jedinou abstraktní metodu `createHandler`, která slouží pro vytvoření instance obslužné třídy pro daný typ periferie. Přiklad struktury pro digitální výstup je na obrázku 6.1.



Obrázek 6.1: Class diagram pro HWAL

V současné době jsou implementovány následující periferie a jejich typy:

- DigitalOutput
  - Internal
  - Firmata
  - I2C
- DigitalInput
  - Internal
  - Firmata
  - I2C
- AnalogOutput
  - Internal (PWM)
  - Firmata (PWM)
- AnalogInput
  - Firmata
  - I2C
- Serial (UART)
  - Internal
- I2C
  - Internal

6.2 HW Definition Layer

Na rozdíl od předchozího bloku je tento mnohem méně rozsáhlý, a proto je tvořen pouze jedním zdrojovým souborem, a nikoliv celým modulem. Pro zbytek systému je relevantní pouze funkce `get_pinout`. Ta vrací obslužné třídy všech periferií, které jsou v souboru `pinMap.json` definovány a mají parametr `Visible` s hodnotou `True`, ve formě slovníku (datový typ dictionary jazyka Python). Všechny periferie je tak možné volat pomocí jména.

V prvním kroku při definování periferií je přečteno ID uložené na EEPROM paměti na připojeném HW. Použit je integrovaný obvod AT24C256, pro který byla vytvořena vlastní knihovna dostupná pomocí nástroje pip.

Získané ID je dále použito pro načtení příslušného pinMap souboru. Tyto soubory jsou ukládány v adresáři `/home/dev/DPS_tester/tests/`. Tento adresář je inicializován jako git repozitář. Díky tomu, že má tento repozitář definován vzdálený origin na platformě GitHub, mělo by být možné provést případné úpravy vzdáleně. HWDL pak pro automatické aktualizace před každým testem provádí metodu `pull`, ovšem pouze v případě, že je aktivní větví větev `main` (zbylé větve jsou brány jako pracovní a vývojář by tak přišel o dosavadní práci). V adresáři `tests` je pak dále adresář pojmenovaný dle ID pro všechny možné testované DPS a obsahuje hledaný soubor pinMap a testovací skripty.

Obsah pinMap souboru je pak v dalším kroku zpracován dle verze souboru. Získané parametry jsou předány funkci `createHandler` importované z modulu HWAL a ta vytvoří instanci obslužné třídy pro danou periferii.

Během vývoje byl vytvořen skript `test.py`, pomocí kterého byly testovány implementované funkce bloků HWAL a HWDL. Následující kód zobrazuje testovací skript pro výstupní a vstupní digitální piny a pro výstupní a vstupní analogové piny.

```
from time import sleep
from HW_definition import get_pinout
import threading

def thread_Blink():
    while True: # Run forever
        pinout["I2Cpin"].writePin(1)
        sleep(1) # Sleep for 1 second
        pinout["I2Cpin"].writePin(0)
        sleep(1) # Sleep for 1 second
        value = pinout["I2CpinIn"].readPin()
        print(value)

def thread_Analog():
    while True:
        value = pinout["FirmataInAnalog"].readPin()
        if value != None:
            pinout["FirmataPWMOut"].writePin(value)

pinout = get_pinout()

blink_thread = threading.Thread(target=thread_Blink)
blink_thread.start()

analog_thread = threading.Thread(target=thread_Analog)
analog_thread.start()
```

6.3 Python API

Tento modul má, jak už bylo zmíněno, za úkol definovat klíčová slova pro RF, která budou přistupovat k nižším vrstvám. v rámci modulu je vytvořen soubor `RF_API.robot`, který by měl být importován v každém testovacím souboru přistupujícím k HW. Tento soubor v první řadě importuje všechny další soubory modulu a dále definuje klíčové slovo `Initialize HW`. Toto slovo se stará o volání funkce `getPinout` z HWDL a jeho návratovou hodnotu nastavuje jako globální proměnou. Díky tomu je možné ve zbylých klíčových slovech přistupovat k HW a jediným povinným argumentem je jméno periferie. Posledním krokem tohoto klíčového slova je vyvolání dialogového okna, které po obsluze požaduje připojení testované DPS.

Ve zbylých souborech jsou definována klíčová slova odpovídající danému typu periferie, tj. např. soubor `Digital.py` definuje slova `SET PIN` pro digitální výstup a `READ PIN` pro digitální vstup. Samotný kód klíčového slova obsahuje v zásadě pouze získání instance obslužné třídy prostřednictvím

jména periferie, zavolání odpovídající funkce z HWAL a ošetření chybových stavů (periferie s daným jménem neexistuje nebo nemá odpovídající typ).

V následující ukázce je uveden příklad definice klíčového slova pro nastavení hodnoty digitálního výstupního pinu.

```
@keyword(types=[str, bool])
def set_pin(name, value) -> None:
    try:
        pinout = BuiltIn().get_variable_value("${pinout}")
        pin = pinout[name]
    except KeyError:
        raise Exception("Object " + name + " not found! Check
    spelling")
    if isinstance(pin, DigitalOutput):
        pin.writePin(value)
    else:
        raise Exception("Selected object is not digital output")
```

6.4 Custom Keywords

V současné době je definován jen malý počet vlastních klíčových slov. Je plánováno, že jejich počet bude postupně narůstat dle často využívaných funkcí.

Klíčové slovo `Should Be In Range` kontroluje, zda je zadaná hodnota v zadaném rozmezí. Druhé klíčové slovo `Check Pin In Loop` kontroluje správné propojení pinů zadaných jako argumenty `send` a `return`. V případě, že je propojení po cestě invertované, je potřeba nastavit argument `invert` na hodnotu `True`. Pro propojení s vyšší mírou latence (například je vyžadována interakce MCU nebo je na testované lince větší kapacita) je možné vyhodnocení zpomalit argumentem `delay`. Dále je definováno klíčové slovo `Check Analog Pin`, které přečte hodnotu ze vstupního analogového pinu a pomocí klíčového slova `Should Be In Range` zkontroluje, zda je načtené napětí v udaných mezích. Měřené napětí je pro možnost budoucí kontroly uloženo do logu. Kód těchto klíčových slov je zobrazen v následující ukázce.

```
*** Settings ***
Library     HW_init.py
Library     Digital.py
Library     Analog.py

*** Keywords ***
Should Be In Range
    [Arguments]     ${input}     ${min}     ${max}
    Run Keyword If     ${input} < ${min}
    ...     Fail     The value ${input} too low (min: ${min})
    Run Keyword If     ${input} > ${max}
    ...     Fail     The value ${input} too high (max: ${max})

Check Analog Pin
    [Arguments]     ${pin}     ${min}     ${max}
    ${voltage}=     Analog.Read Pin     ${pin}
    Log To Console     PWR voltage: ${voltage}
```

```
18      Should Be In Range      ${voltage}      ${min}      ${max}
19
20  Check Pin In Loop
21      [Arguments]     ${send}     ${return}     ${invert}=False     ${
        delay}=${0}
22      Digital.Set Pin     ${send}     False
23      Sleep     ${delay}
24      ${value}=     Digital.Read Pin     ${return}
25      Run Keyword If     ${value} != ${invert}
26      ...     Fail     Pin ${send} set to False. Pin ${return}
        should be ${invert}
27      Digital.Set Pin     ${send}     True
28      Sleep     ${delay}
29      ${value}=     Digital.Read Pin     ${return}
30      Run Keyword If     ${value} == ${invert}
31      ...     Fail     Pin ${send} set to True. Pin ${return} should
         not be ${value}
32      Digital.Set Pin     ${send}     False
```

6.5 GUI pro tvorbu pinMap souboru

Nad rámec zadání bylo vytvořeno jednoduché GUI pro tvorbu pinMap souborů. Jejich tvorba manuální cestou by byla velmi náročná a vedla by k častým chybám. Toto GUI bylo tvořeno pomocí knihovny guizero [11]. Tvorba nebo editace pinMap souboru by neměla probíhat na hlavní větvi git repozitáře. Proto je při startu větev ověřena a program dále nepokračuje, pokud odhalí problém.

GUI tohoto nástroje je tvořeno tak, aby se automaticky aktualizovalo dle zvolených hodnot. Toho je docíleno pomocí souboru `peripheries.json`. v tomto souboru jsou vyjmenovány všechny periferie, všechny jejich typy a veškeré PSP a TSP včetně definice jejich datového typu (viz příklad pro digitální výstup).

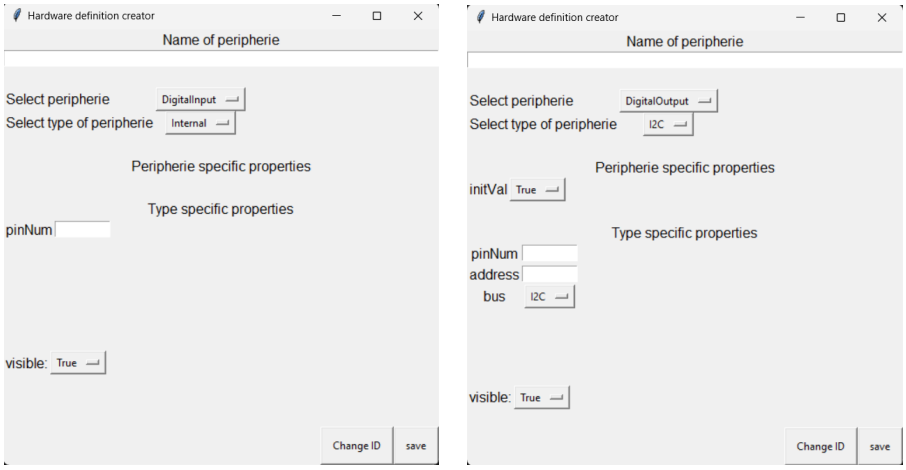
```
1  "DigitalOutput" : {
2          "type" :{
3              "Internal":{
4                  "pinNum" : "int"
5              },
6              "Firmata":{
7                  "pinNum" : "int",
8                  "firmata" : "Firmata"
9              },
10             "I2C":{
11                 "pinNum" : "int",
12                 "address" : "string",
13                 "bus" : "I2C"
14             }
15         },
16         "PSP" :{
17             "initVal" : "bool"
18         }
19     },
```

Po potvrzení vložených hodnot dojde k jejich jednoduché kontrole (je ověřeno, že jsou všechna pole vyplněna) a po uložení se GUI vrátí do výchozího stavu.

Pomocí tohoto GUI je možné i zapsat HW ID do připojeného paměťového čipu při oživení nového zařízení.

Současná verze aplikace umí pouze přidávat definice nových periferií. v budoucnu by mělo dojít k rozšíření a umožnění procházení a editace již definovaných položek pinMap souboru. Dále by mělo dojít k úpravě aplikace tak, aby bylo možné ji spustit i mimo Raspberry Pi. To by mělo být relativně jednoduše proveditelné díky použití jazyka Python (aplikaci je pouze potřeba nějakým způsobem předat správnou cestu k souborům).

Příklady vzhledu nástroje pro tvorbu pinMap souborů pro různé periferie a jejich typy zobrazují obrázky 6.2(a) a 6.2(b).



(a) (b)

Obrázek 6.2: GUI nástroje pro tvorbu pinMap souborů

6.6 Uživatelské GUI

Stejně jako předchozí GUI bylo i toto tvořeno pomocí knihovny guizero.

Jelikož není kladen žádný nárok na odbornost uživatele, musí být toto rozhraní co nejjednodušší. v současné době je tvořenou velkým tlačítkem pro spuštění testu a sadou textových polí informujících o průběhu a následně výsledku testu. Pro uživatele je výsledek testu pouze binární (tj. Pass/Fail). Pro hlubší vyhodnocení slouží report automaticky generovaný RF, ale k tomu by měl mít přístup pouze vývojář. Samotné uživatelské rozhraní je zobrazeno na obrázku 6.3.

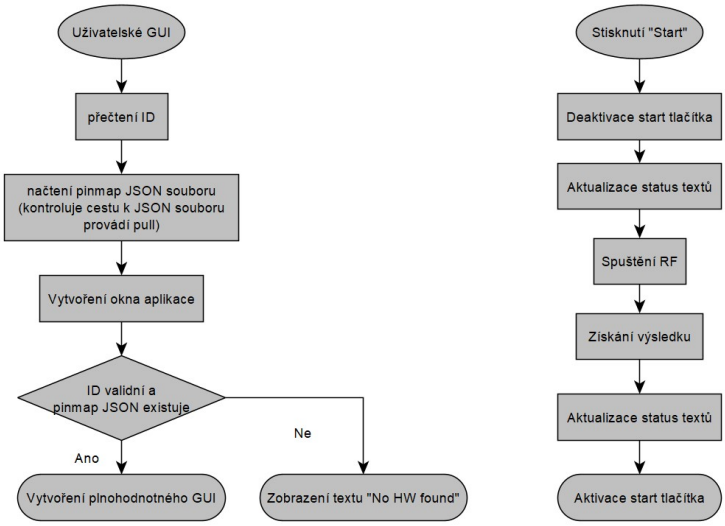
Uživatelské rozhraní se automaticky spouští po startu zařízení, a navíc v režimu fullscreen a uživatel by neměl mít možnost ho opustit.

Pro realizaci testů, kde je potřeba interakce uživatele (např. zadání sériového čísla pro účely evidence výsledků), by měla být použita knihovna RF

Dialogs. Ta definuje klíčová slova pro vyvolání jednoduchých dialogových oken například pro zadání textu, volby možnosti typu Ano/Ne atd. Díky použití této knihovny jsou dialogová okna testu plně oddělena od uživatelské aplikace.

Funkci uživatelské aplikace popisuje vývojový diagram na obrázku 6.4.

Obrázek 6.3: uživatelské GUI



Obrázek 6.4: Vývojový diagram pro uživatelské GUI

Kapitola 7

Popis vzorové testované DPS

Pro první otestování systému byla vybrána DPS izolace čidel. Tato DPS je v jednom z produktů firmy TSE spol. s.r.o. použita pro galvanické oddělení obsluhy a pacienta s požadovanou izolační pevností 4 kV. Potřeba tohoto galvanického oddělení vyplívá z požadavků norem pro zdravotnické přístroje, konkrétně ČSN EN 60601 [12].

Dle logiky z výsledného zařízení bude o galvanicky oddělené části zvolené DPS (část směřující k pacientovi) referováno jako o výstupní straně a část v zařízení připojená k jeho zbylým částem bude označována za vstupní. Toto označení nijak nesouvisí s orientací toku signálů na DPS (signály jsou orientovány oběma směry).

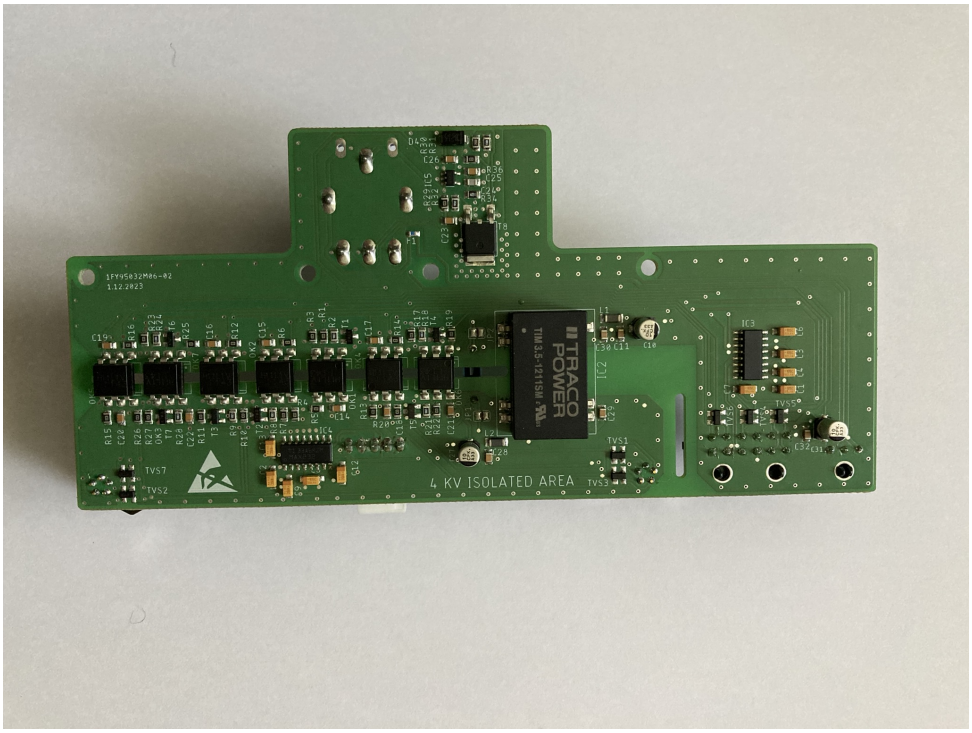
Vybraná DPS dále obsahuje i obvod pro řízení vyšetřovacího světla pomocí potenciometru s tlačítkem v kombinaci s operačním zesilovačem v zapojení převodník napětí na proud.

Pro zvolenou DPS bude potřeba otestovat následující funkce:

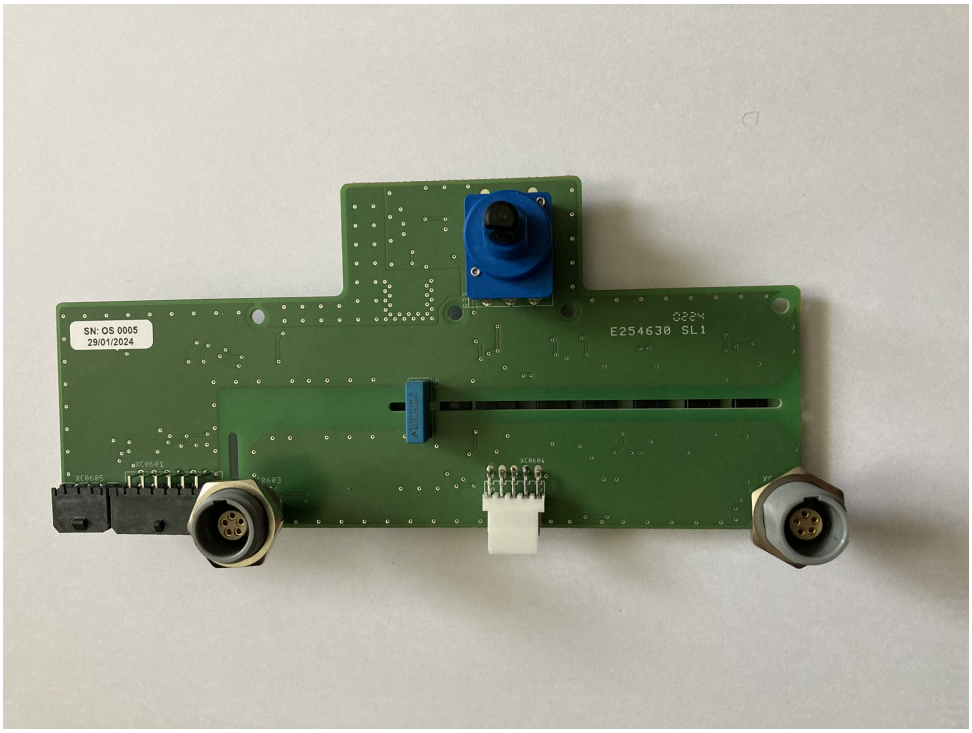
- Napětí výstupu izolovaného zdroje
- Přenesení všech digitálních signálů
- Správnou funkci sběrnice RS-232 ze vstupu na výstup
- Řízení vyšetřovacího LED světla

Poslední bod testování (tedy řízení LED světla) by byl vzhledem ke své povaze náročně automatizovatelný. Aby bylo předejito relativně složité elektromechanické soustavě, měl by tento test provést uživatel, který bude muset následně předat informaci o výsledku testu.

Vybraná testovaná DPS je specifická mimo jiné i tím, že vyžaduje napájení 12 a 5 V, které bude také potřeba zajistit.



Obrázek 7.1: Testovaná DPS



Obrázek 7.2: Testovaná DPS

Kapitola 8
Návrh propojovacího HW

Testování elektrické pevnosti je samostatným úkonem po zkompletování celého přístroje a není součástí funkčního testu samotné DPS, a proto bude možné pro zjednodušení propojovacího HW vstupní a výstupní stranu galvanicky opět spojit. Možnost tohoto propojení vstupní a výstupní strany zdroje byla předem ověřena.

V průběhu celého projektu bylo přihlíženo k co největšímu usnadnění při implementaci dalších testování. Návrh propojovacího HW by v tomto ohledu neměl být výjimkou. Jelikož se tento HW skládá z množství samostatně funkčních bloků, jsou tyto bloky pro budoucí použití tvořeny jako design block v použitém nástroji pro tvorbu DPS Autodesk Eagle [13]. Návrh propojovací DPS se pak náročností blíží nákupu vývojových modulů a jejich propojení na nepájivém poli.

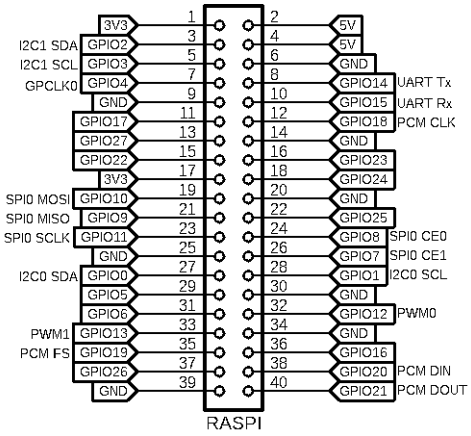
Dále budou popsány design blocky, které jsou v současné chvíli vytvořeny. u všech design blocků byl při tvorbě kladen důraz na dokumentaci (např. popis funkce pinů integrovaných obvodů, adresy použitých I2C periferií atd.). Všechny použité integrované obvody byly voleny tak, aby bylo možné sehnat vývojové kity a aby bylo možné je napájet napětím 3,3 v i 5 V.

8.1 Raspberry Pi pinheader

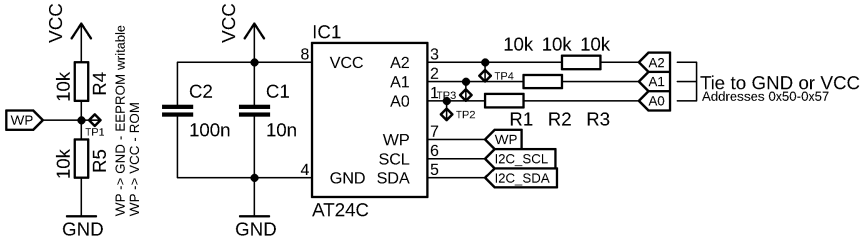
Tento design block je tvořen pouze jediným konektorem. Jde o konektor pro plochý kabel s roztečí, počtem pinů a pinoutem odpovídajícím rozšiřujícímu konektoru na Raspberry Pi [14]. Veškeré piny jsou popsány včetně možné alternativní funkce. Tento blok je sice velmi jednoduchý, ale při obkreslování pinoutu je velmi snadné udělat chybu, a proto bude pro další návrhy velmi užitečný. Použitý konektor pro plochý kabel je možné nahradit i klasickou pinovou lištou se shodnou roztečí (je možné výslednou DPS použít jako shield pro Raspberry Pi).

8.2 I2C EEPROM

Dalším blokem společným pro všechny budoucí propojovací DPS je I2C EEPROM. Jak již bylo zmíněno, je použit integrovaný obvod AT24C256C [15].



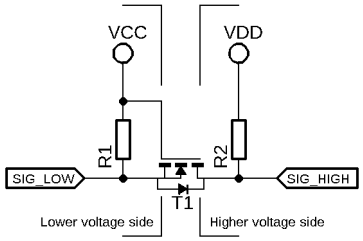
Obrázek 8.1: Schéma bloku Raspberry Pi pinheader



Obrázek 8.2: Schéma bloku I2C EEPROM

Výhodou tohoto integrovaného obvodu může být i možnost použití pinu Write Portection, který po připojení na napájecí napětí zabraňuje zápisu do paměti obvodu. Při oživení zařízení by měl být osazen rezistor R5 (dle značení na obrázku 8.2) a pozice R4 by měla být zachována neosazená. Po zápisu HW ID do paměti by naopak mělo dojít k demontáži R5 a osazení R4, čímž bude zabráněno nechtěnému přepsání HW ID.

8.3 Převodník napěťových úrovní

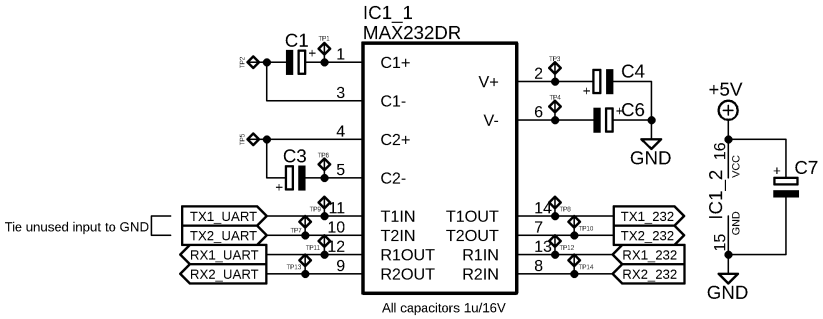


Obrázek 8.3: Schéma bloku Převodník napěťových úrovní

Jak název napovídá, slouží toto jednoduché zapojení k převodu napěťových úrovní pro digitální signály. v případě této DPS jde o přechod mezi napětím 3,3 v na straně Raspberry Pi na 5 v na straně testované DPS.

Pokud je uzemněna strana nižšího napětí (`SIG_LOW` je log. 0), napětí `UGS` otevře tranzistor a tím dojde i k uzemnění na straně vyššího napětí (`SIG_HIGH`). v opačném případě (`SIG_HIGH` je uzemněn) dochází k uzemnění pomocí substrátové diody tranzistoru [16].

8.4 Převodník UART -> RS232



Obrázek 8.4: Schéma bloku Převodník UART -> RS232

Testovaná DPS obsahuje i sběrnici RS232. Tato sběrnice je na testované DPS galvanicky oddělena pomocí optočlenů, ale jinak s ní není nijak zacházeno, tj. data by na výstup měla projít beze změny. k testování by tedy mělo dojít tím, že budou signály Rx a Tx na výstupní straně testované DPS propojeny. Vzhledem k problémům při implementaci kódu pro práci s interní UART sběrnicí jsou pro toto testování připraveny dvě varianty:

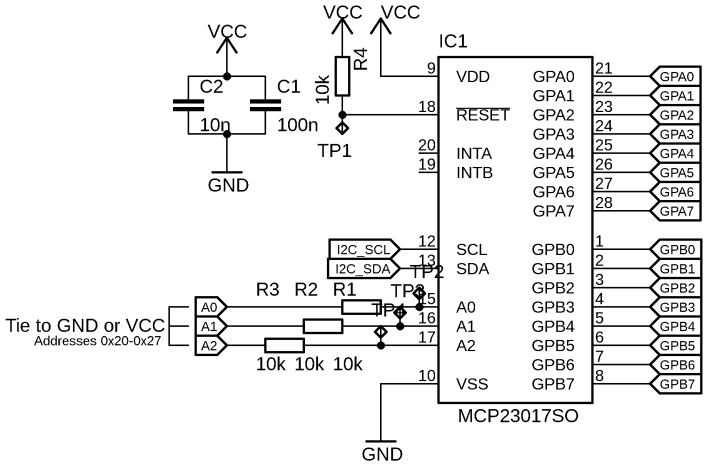
Testování pomocí sběrnice UART Skrze převodník MAX 232 [17] jsou do testované DPS posílána testovací data, která by měla být díky propojení výstupu opět přijata.

Testování pomocí GPIO Skrze převodník MAX 232 [17] je nastavována hodnota log. 0 a log. 1 na pinu Tx, která by měla být díky propojení na výstupu opět přečtena na pinu Rx

Způsob testování by měl být určen pomocí pájecího SMD jumperu. v testovacím skriptu je počítáno s testováním pomocí GPIO. Testování pomocí sběrnice UART by mělo mít větší výpovědní hodnotu, jelikož testuje i možné zpomalení hran signálu. Pokud by ovšem docházelo k zpomalení hran signálů, šlo by o chybu návrhu testované DPS, jehož odhalení není cílem tohoto zařízení.

8.5 I2C GPIO Expander

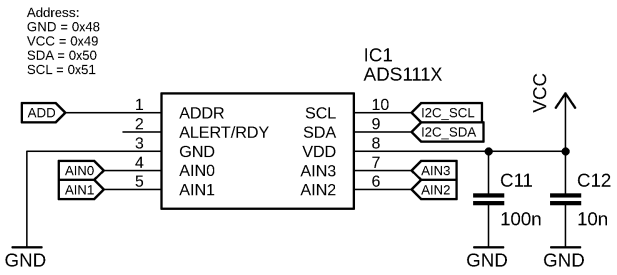
Tento blok měl sloužit převážně v případě, že by byl počet GPIO Raspberry Pi nedostatečný. V prezentované DPS je použit, protože veškeré piny testované DPS pracují s napětím 5 V. Použití převodníků logických úrovní pro všechny piny by bylo nepraktické. Proto byl použit tento blok a převodníky



Obrázek 8.5: Schéma bloku I2C GPIO Expander

napěťových úrovní jsou umístěny pouze na sběrnici I2C. V tomto bloku je použit integrovaný obvod MCP23017 [18]. Tento obvod umožňuje nastavení každého pinu na vstupní nebo výstupní a díky třem adresním pinům je možné použít až osm integrovaných obvodů připojených na jedinou sběrnici.

8.6 I2C ADC



Obrázek 8.6: Schéma bloku I2C ADC

Jako externí ADC byl zvolen integrovaný obvod ADS1115 [19]. Jedná se o čtyř kanálové ADC s rozlišením šestnáct bitů. Nastavení adresy umožňuje připojení až čtyř integrovaných obvodů na jednu sběrnici.

8.7 Výsledná propojovací DPS

Vzhledem k nižšímu množství součástek potřebných pro vytvoření propojovacího HW pro vybranou testovanou DPS byly rozměry voleny tak, aby bylo možné vzniklou DPS použít jako shield k Rapsberry Pi. Díky tomu bude možné minimalizovat velikost výsledného zařízení. Není jisté, zda bude podobný přístup použit i pro další zařízení.

Raspberry Pi má dvě I2C sběrnice. Zatímco I2C1 je určena pro běžné použití, I2C0 je určena pro použití s paměťovými čipy pro detekci různých shield modulů [20]. Toho by bylo možné využít i v tomto projektu pro detekci HW ID. Prozatím je paměťový prvek připojen na sběrnici I2C1, jelikož je pak jednodušší přístup k datům z vlastního kódu (I2C0 je čtena při startu OS a neměla by být přístupná uživateli).

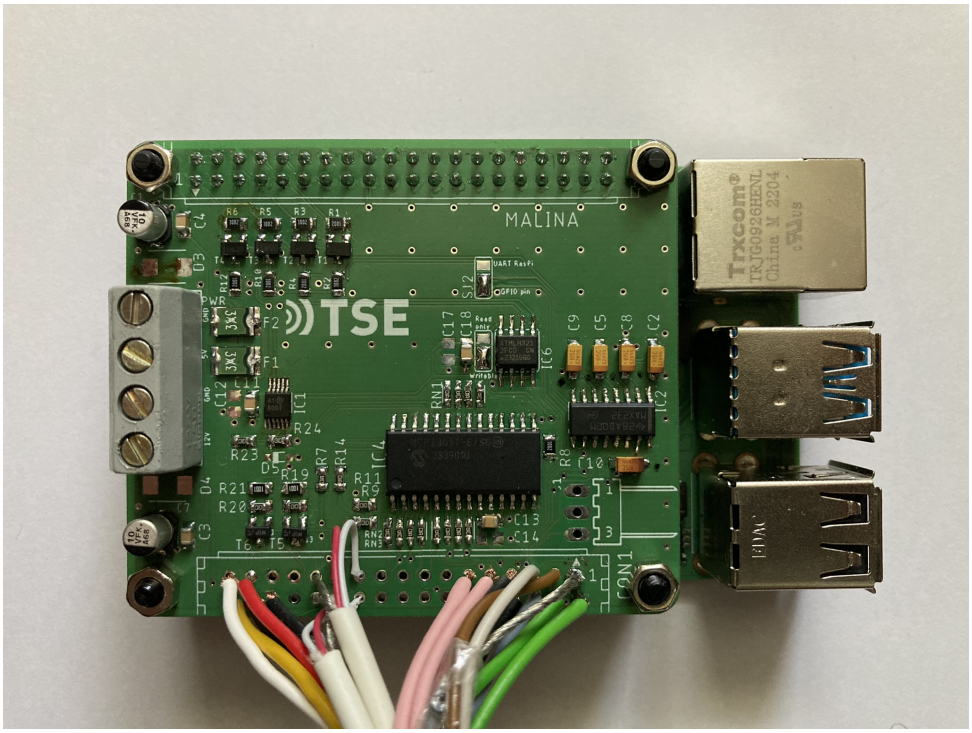
Jako výstupní konektor pro připojení testované DPS byl zvolen konektor pro plochý kabel (ribbon cable). Výroba kabelového svazku by tak měla být jednodušší, avšak v prototypovém kusu se počítá se zapájením ustřiženého kabelu ze sériové výroby přímo do desky (vybraná testovaná DPS v době vzniku této práce stále prochází vývojem a může dojít k její změně). Testovaná DPS bude připojena pomocí stejných konektorů, které budou použity i ve výsledném přístroji. v průběhu byla zvažována i možnost výroby fixtury (podobně jako při In-Circuit testování, viz kapitola 2.4), ale to by zařízení zdražilo. Další výhodou je, že dojde i k otestování konektorů samotných (v minulosti se objevily případy teplem zdeformovaných konektorů, nebo i došlo k jejich neosazení). Naopak nevýhodou je, že bude docházet k opotřebení kabelů testovacího zařízení.

Napájení by měl řešit zdroj Mean Well RD-35A. Jde o síťový spínaný zdroj s dvouhladinovým výstupem (5 a 12 V) určený k zástavbě. Napájení pro testovanou DPS je vedeno přes nelineární rezistor typu PTC, který by měl zabránit vypnutí zařízení vlivem nadproudové ochrany při závadě na testované DPS.

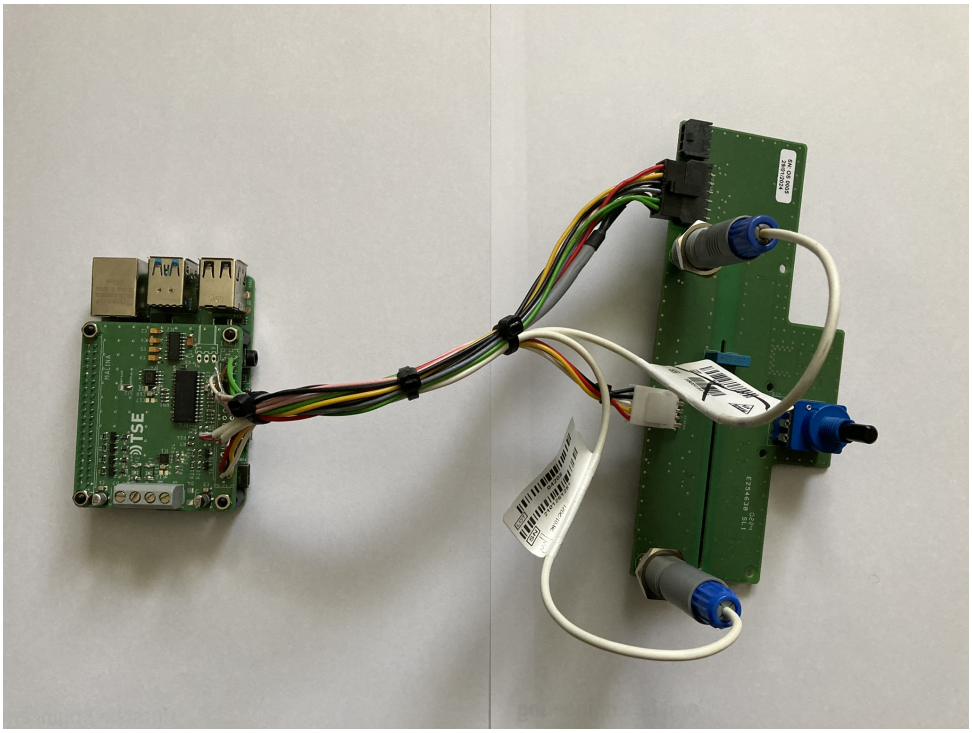
Pro testování LED světla bude použit LED modul shodný s tím ve výsledném přístroji. Tato volba se jevila jako nejjednodušší a test je díky tomu nejvíce průkazný.

Po ověření funkčnosti zařízení jako celku by mělo dojít i k vytvoření krytu. Pro zástavbu byl zvolen dotykový TFT displej s úhlopříčkou sedm palců a rozlišením 1024x600.

Navržená propojovací DPS připojená k Raspberry Pi je zobrazena na obrázku 8.7 a propojení s testovanou DPS ukazuje obrázek 8.8. z obrázků je patrné, že nejsou plně využity veškeré výstupní piny propojovací DPS. To je způsobeno probýhajícím vývojem testované DPS, jak bylo zmíněno výše.



Obrázek 8.7: Raspberry Pi s propojovací DPS



Obrázek 8.8: Navržený systém s připojenou testovanou DPS

Kapitola 9

RF skript pro testovanou DPS

Tato kapitola popisuje tvorbu testovacího skriptu pro zvolenou DPS. Dále blíže popisuje práci s RF v kombinaci s knihovnami vytvořenými v rámci této práce.

9.1 Společná šablona

Jak již bylo zmíněno, je adresář s testy inicializován jako samostatný git repozitář a každá testovaná DPS by v něm měla mít vlastní složku s testy a pinMap souborem. Pro usnadnění práce vývojáře byla vytvořena složka `template` obsahující soubory s minimálními náležitostmi.

Prvním souborem je `__init__.robot`. Pokud RF spouští více testovacích skriptů v jednom adresáři je využití takového souboru výhodné, pokud mají mít testovací skripty v adresáři společnou část. v tomto případě jde o inicializaci HW, která by měla proběhnout pouze jednou při startu testu. K takovému účelu slouží klíčové slovo `Suite Setup`. Argumentem klíčového slova je pak klíčové slovo, které má být vykonáno. Obdobně funguje klíčové slovo `Suite Teardown` pro definování průběhu ukončení testu.

```
1 *** Settings ***
2 Resource    RF_API/RF_API.robot
3 Suite Setup    Initialize HW
```

Dalším souborem je `RF_test.robot`. Jde o soubor RF, ve kterém je pouze importována knihovna `RF_API` podobně jako v předchozím popsaném souboru tak, aby bylo možné přistupovat k HW. Dále je importována knihovna CustomKeywords.

Posledním souborem je `pinMap.json`, který je také téměř prázdný a měl by být vyplněn za pomoci výše popsaného GUI.

9.2 Skript pro testovanou DPS

Pro zvolenou testovanou DPS byl vytvořen nový adresář uvnitř repozitáře `tests` zkopírováním adresáře `template` a přejmenováním dle interního označení. Pomocí GUI bylo stejné označení zapsáno do paměťového integrovaného

obvodu na propojujícím HW a dále byl upraven soubor pinMap. z definovaných periferií jsou použity následující: I2C, DigitalInput, DigitalOutput a AnalogInput.

Dále již bylo přistoupeno k tvorbě samotných testů. Pořadí testů je voleno dle závažnosti jejich chyby. Prvním je proto kontrola napájecího napětí na výstupu izolovaného zdroje na testované DPS, jelikož při nefunkčním zdroji by nefungovali ani všechny zbylé obvody na izolované části desky. Meze tohoto testu jsou určeny dle požadavků obvodů, které měřený zdroj napájí. Přísnější meze by mohli být nastaveny dle tolerance výstupního napětí použitého zdroje. Změna mezí je velmi jednoduchá, jelikož jsou krajní hodnoty definovány pomocí proměnných. Hodnota mezí musí být kvůli napěťovému děliči na vstupu ADC volena poloviční oproti požadovanému napětí.

Dále jsou testovány jednotlivé digitální signály pomocí výše popsaného klíčového slova `Check Pin In Loop`. Dle směru signálu jsou voleny piny send a return buďto na vstupní nebo výstupní straně testované DPS. Lehce odlišný přístup je pak v případě testování sběrnice RS232, kde je na výstupní straně DPS propojen signál Rx a Tx, čímž je vytvořena smyčka. Signál Tx na vstupní straně je pak použit jako send a Rx jako return.

Posledním testem je vyzkoušení ovládání LED světla. Pro tento krok je použito klíčové slovo `Execute Manual Step` z knihovny Dialogs. Toto klíčové slovo zobrazí dialogové okno s instrukcemi k otestování LED světla a uživatel poté zvolí možnost `PASS` nebo `FAIL` na základě výsledku testu. Při zvolení `FAIL` je dále zobrazeno další okno s textovým polem, kam může uživatel zadat podrobnější popis odhaleného problému.

V zkrácené podobě je kód testu zobrazen následující ukázkou.

```
1  *** Settings ***
2  Resource    RF_API/RF_API.robot
3  Resource    RF_API/CustomKeywords.robot
4
5  *** Variables ***
6  ${MAX_PWR_VOLTAGE}=    ${2.625}
7  ${MIN_PWR_VOLTAGE}=    ${2.375}
8
9  *** Test Cases ***
10 Check PWR
11     Check Analog Pin    PWR_CHECK    ${MIN_PWR_VOLTAGE}    ${
       MAX_PWR_VOLTAGE}
12
13 Check Data connection
14     Check Pin In Loop    DATA_T1    DATA_T_OUT
15     Check Pin In Loop    DATA_T2    DATA_T_OUT
16
17     ...
18 Check LED
19     Execute Manual Step    Vyzkousejte funkce rizeni LED svetla
```

Kapitola 10

Zhodnocení dosažených výsledků

Pro řešení zařízení byla zvolena architektura s jednodeskovým počítačem bez bloku MCU. Tato volba se prokázala jako správná díky výraznému zjednodušení HW výsledného zařízení. Zásadní výhodou tohoto řešení je možnost vytvoření samostatně pracujícího přístroje. Pravděpodobně také bude platit, že s touto koncepcí je možné docílit větší rychlosti testu, jelikož testovací skript přímo přistupuje k periferiím, zatímco ve zbylých architekturách bylo počítáno s komunikací mezi zařízením zpracovávajícím skript (PC nebo jednodeskový počítač) a zařízením interagujícím s testovanou DPS.

Díky zvážení architektury SW hned při začátku práce na projektu bylo docíleno rozdělení na jednotlivé vrstvy tak, aby byl testovací skript co nejvíce oproštěn od HW a byl co nejlépe čitelný. Vrstva HWAL zajišťuje absolutní nezávislost programu vyšších vrstev na HW. Nevýhodou současného řešení je náročnější proces doplnění nových podporovaných periferií. Ke zjednodušení tohoto procesu by přispělo, kdyby nástroj pro vytváření pinMap souborů načítal seznam podporovaných periferií přímo ze zdrojového kódu HWAL. HWDL umožňuje automatické rozpoznání připojeného HW a dále automatickou aktualizaci testovacích skriptů a pinMap souborů.

Zvolený nástroj pro automatizaci testů je díky kvalitní dokumentaci a velké uživatelské základně velmi snadný na používání. Zásadní výhodou je možnost přizpůsobení pro konkrétní použití díky podpoře uživatelských knihoven tvořených v jazyce Python. Kombinací takové knihovny a vlastních klíčových slov je možné dosáhnout snadno čitelného testovacího skriptu, jehož vytvoření zabere jen minimum času.

Návrh propojovacího HW byl zjednodušen využitím designových bloků v použitém nástroji Eagle. I takto je ovšem potřeba vyšší míra odbornosti vývojáře. Dalšího zjednodušení by bylo možné docílit vytvořením modulů. Velikost výsledného zařízení by byla v takovém případě pravděpodobně větší a snížila by se možnost přizpůsobení zařízení pro potřeby testované DPS.

Pro výrobu kabelového svazku pro zařízení byly využity upravené produkční kabelové svazky. Tímto způsobem bylo vytvoření kabelu nejrychelejší. Nevýhodou tohoto řešení je zapájení kabelu přímo do DPS propojovacího HW. Tento přístup by proto měl být použit pouze pro prototypový kus. Pro finální zařízení by měl být vytvořen výkres a kabelový svazek by měl být vyroben specializovanou firmou.

Vytvořený testovací skript ověřuje všechny funkce testované DPS uvedených v kapitole 7. Díky knihovně Dialogs je možné provádět i testy, které by byly náročně automatizovatelné.

Kapitola 11

Budoucí rozšíření

Hlavním směrem pro budoucí rozšiřování tohoto projektu je bezpochyby vytvoření co nejdetailnější dokumentace. Součástí takové dokumentace by v ideálním případě mělo být i množství demo ukázek založených na již implementovaných testovacích zařízeních. Díky tomu by mělo být v budoucnu možné svěřit tvorbu nových zařízení dalšímu vývojáři v co nejkratším časovém horizontu.

Dalším krokem je pokračování v přidávání kompatibilních periferií tak, aby bylo možné zařízení ve fázi návrhu co nejvíce přizpůsobit testované DPS, např v současné době použité ADC je relativně drahé a rozlišení 16 bit je pro mnohé aplikace zbytečně velké. Bylo by také dobré implementovat i další periferie dostupné pomocí použitého protokolu Firmata (externí I2C a UART sběrnice a další).

V rámci správy SW bude potřeba po dostatečném otestování a implementaci některých zvolených typů periferií vytvořit release balíček, kterému budou i odpovídat verze pinMap souborů a souboru peripheries.json.

Stejně jako by mělo postupně docházet k rozšiřování kompatibilních periferií, mělo by postupně docházet i k vzniku nových klíčových slov RF pro často používané funkce.

Jak již bylo zmíněno v kapitole 3, bylo v návrhu počítáno s možností přidání funkce programování mikrokontrolerů na testovaných DPS. Tato funkce by mohla být velmi užitečná, jelikož programování je často prováděno obsluhou a může být i celkem zdlouhavé. Nově vyrobená DPS by takto mohla být rovnou připojená k testovacímu zařízení a po spuštění testu by se nahrál testovací SW. Mikrokontroler by poté spolupracoval se zařízením v průběhu testu a na jeho konci by byl nahrán již produkční SW. k samotnému programování mikrokontrolerů s architekturou AVR je možné použít nástroj AVRDUDE [21].

V současné chvíli je uživatel upozorněn na výsledek testu pouze ve formě vyhovuje/nevyhovuje. Pro zlepšení tohoto stavu jsou zvažovány dva možné přístupy. První možností by bylo zobrazit seznam neúspěšných testů spolu s výpisem chybové hlášky. Druhým řešením by bylo ukládání výstupních logů, které generuje RF, jak je naznačeno na diagramu architektury SW (obrázek 4.4). V případě ukládání protokolů by bylo potřeba doplnit inicializaci testu tak, aby uživatel zadal identifikační kód testované DPS.

Nejambicióznějším cílem pro budoucí rozšíření je úprava zařízení pro provádění HW in Loop testů, tzn. testováno by bylo celé zařízení a jeho SW namísto jediné DPS. V minulosti již byl pro účely testování nového SW vytvořen tzv. simulátor čidel. Jde o zařízení, které napodobuje komunikační protokoly senzorů použitých v přístrojích firmy TSE. Jeho obsluhu a vyhodnocení ale dále provádí pracovník vývoje. Pokud by zařízení popsané v této práci bylo doplněno o API pro řízení tohoto simulátoru a dále o API pro komunikaci se servisním USB testovaného přístroje, bylo by možné simulovat chybové stavy přístroje a vyhodnocovat odezvu testovaného SW. Automatizované testování tímto způsobem by poté mohlo být součástí procesu vydávání nové verze SW.

Kapitola 12

Závěr

Cílem práce bylo vytvoření kompletního systému pro provádění funkčních testů DPS. V práci byly uvedeny některé často používané metody pro testování DPS, aby byl uveden základní kontext této problematiky a byl více přiblížen záměr projektu.

Zmíněny byly různé koncepce základní architektury navrhovaného systém. U těchto zmíněných koncepcí byly zdůrazněny hlavní výhody i nevýhody a tím byla zdůvodněna volba použité architektury. Ta je založena na jednodeskovém počítači Raspberry Pi 4B s použitým operačním systémem Raspberry Pi OS. Druhým blokem HW systému je tzv. propojovací HW, tj. DPS tvořící rozhraní mezi rozšiřujícím konektorem použitého jednodeskového počítače a testovanou DPS, která se bude lišit dle potřeb testované DPS.

Jelikož hlavní důraz je napříč celým projektem kladen na modularitu systému, bylo potřeba důkladně zvážit architekturu SW ještě před jeho implementací. Nejvíce diskutovanými bloky systému jsou bloky pro abstrakci HW, Hardware Abstraction Layer (HWAL), a blok pro popis použitých periferií, Hardware Definition Layer (HWDL). Díky těmto blokům má vývojář tvořící zařízení pro testování nové DPS možnost využívat různé druhy podporovaných periferií a ve vyšších vrstvách k nim přistupovat bez rozdílu a snadno čitelným způsobem.

Pro tvorbu testovacích skriptů byl použit nástroj Robot Framework. Použití hotového a ověřeného nástroje s rozsáhlou dokumentací velmi usnadnilo a urychlilo vývoj systému jako celku. Díky přímé podpoře vlastních knihoven v jazyce Python bylo propojení RF a vlastního kódu velmi snadné. RF navíc nabízí i rozhraní pro řízení testování z jazyka Python, což umožnilo vznik jednoduchého GUI pro uživatele. Rozsáhlá dokumentace RF a dokumentace kódu ve vzniklých knihovnách by měla umožnit snadné zaškolení pracovníků i s nižší odborností pro tvorbu testovacích skriptů.

Další výhodou použití RF je možnost tvorby vlastních klíčových slov. To umožňuje zvýšení čitelnosti kódu testovacího skriptu, jelikož pro často testované funkce je možné vytvořit nové klíčové slovo.

Nad rámec zadání vznikly i dvě grafické aplikace. Jedna je určena pro uživatele a druhá pro vývojáře. První zmíněná aplikace spouští test a informuje uživatele o jeho průběhu a výsledku. Druhá je určena pro tvorbu pinMap souborů. Jde o soubory, se kterými pracuje blok HWDL a které popisují

použité periferie a jejich typy. Vytvoření grafického rozhraní pro tvorbu těchto souborů výrazně usnadňuje vývojáři přípravu systému pro testování nové DPS a snižuje pravděpodobnost chyby v tomto kroku díky omezením, která zavádí (pořadí definic musí být správné, je provedena základní kontrola vstupů, atd.).

Pro tvorbu propojovacího HW byl použit nástroj Autodesk Eagle. Aby byl proces návrhu takového HW pro další testované DPS v budoucnosti co nejvíce usnadněn, byly veškeré použité bloky implementovány jako tzv. design block. Díky tomu je možné elementární samostatně funkční celky v dalším návrhu znovu využít. Návrh propojovacího HW stále vyžaduje určitou míru odbornosti vývojáře.

Dále byl v práci popsán proces vzniku zařízení pro testování konkrétní zvolené DPS s použitím vytvořeného systému. Popsán byl vznik propojovacího HW i testovacího skriptu.

Systém, který byl implementován v rámci této práce, splňuje veškeré požadavky, které byly stanoveny v zadání. I přes to není v současné chvíli možné označit práci v tomto projektu za plně dokončenou. To je způsobeno vysokým potenciálem pro různá rozšíření, z nichž některá jsou v práci také popsány.

Literatura

[1] PCBWAY. *A.O.I.*. Online. C2024. Dostupné z: https://www.pcbway.com/pcb_prototype/Automated_optical_inspection__AOI_.html. [cit. 2024-04-29].

[2] PCBWAY. *Electronic test - flying probe test*. Online. C2024. Dostupné z: https://www.pcbway.com/pcb_prototype/Electronic_test___probe_test.html. [cit. 2024-04-29].

[3] ROBOT FRAMEWORK RY. *Robot Framework*. Online. C2008-2024. Dostupné z: https://robotframework.org. [cit. 2024-04-29].

[4] ROBOT FRAMEWORK RY. *Robot Framework libraries*. Online. C2008-2024. Dostupné z: https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#creating-test-libraries. [cit. 2024-04-29].

[5] ROBOT FRAMEWORK RY. *Robot Framework Output Files*. Online. C2008-2024. Dostupné z: https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#output-files. [cit. 2024-04-29].

[6] RASPBERRY PI FOUNDATION. *Raspberry Pi OS*. Online. [2012]. Dostupné z: https://www.raspberrypi.com/software/. [cit. 2024-04-29].

[7] MICROSOFT. *Visual Studio Code Remote*. Online. C2024. Dostupné z: https://code.visualstudio.com/docs/remote/remote-overview. [cit. 2024-04-29].

[8] RASPBERRY PI FOUNDATION. *Raspberry Pi Configuration*. Online. C2012-2024. Dostupné z: https://www.raspberrypi.com/documentation/computers/configuration.html. [cit. 2024-04-29].

[9] HYMEL, Shawn. *How to Run a Raspberry Pi Program on Startup*. Online. In: SparkFun Electronics. [2003]. Dostupné z: https://learn.sparkfun.com/tutorials/how-to-run-a-raspberry-pi-program-on-startup/all. [cit. 2024-04-29].

[10] GEEKS FOR GEEKS. *Python Abstract Classes*. Online. [2008], Last Updated : 12 Mar, 2024. Dostupné z: https://www.geeksforgeeks.org/abstract-classes-in-python/. [cit. 2024-04-29].

[11] *Guizero*. Online. [2017], Last release: 2023-12-03. Dostupné z: https://lawsie.github.io/guizero/. [cit. 2024-04-29].

[12] ČESKÝ NORMALIZAČNÍ INSTITUT. ČSN EN 60601-1-2, *Zdravotnické elektrické přístroje - Část 1: Všeobecné požadavky na základní bezpečnost a nezbytnou funkčnost*. Ed. 3. 2016.

[13] SATTEL, Sam. *What-s New in Autodesk EAGLE: Modular Design Blocks*. Online. In: AUTODESK. Autodesk. C2024. Dostupné z: https://www.autodesk.com/products/fusion-360/blog/whats-new-in-autodesk-eagle-modular-design-blocks/. [cit. 2024-04-29].

[14] RASPBERRY PI FOUNDATION. *Raspberry Pi Pinout*. Online. C2012-2024. Dostupné z: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html. [cit. 2024-04-29].

[15] MICROCHIP TECHNOLOGY INC. *AT24C256C*. Online. C1998-2024. Dostupné z: https://www.microchip.com/en-us/product/at24c256c. [cit. 2024-04-29].

[16] JOHANNECK, Don. *Logic Level Shifting Basics*. Online. In: DigiKey. [1996]. Dostupné z: https://www.digikey.cz/en/blog/logic-level-shifting-basics. [cit. 2024-04-29].

[17] TEXAS INSTRUMENTS. *MAX232DR*. Online. C1995-2024. Dostupné z: https://www.ti.com/product/MAX232/part-details/MAX232DR. [cit. 2024-04-29].

[18] MICROCHIP TECHNOLOGY INC. *MCP23017*. Online. C1998-2024. Dostupné z: https://www.microchip.com/en-us/product/mcp23017. [cit. 2024-04-29].

[19] TEXAS INSTRUMENTS. *ADS1115*. Online. C1995-2024. Dostupné z: https://www.ti.com/product/ADS1115. [cit. 2024-04-29].

[20] RASPBERRY PI FOUNDATION. *Raspberry Pi Hats*. Online. C2012-2024. Dostupné z: https://github.com/raspberrypi/hats. [cit. 2024-04-29].

[21] AVRDUDE. *AVRDUDE*. Online. [2003], Last release: 2024-02-07. Dostupné z: https://github.com/avrdudes/avrdude. [cit. 2024-04-29].

Příloha A

Seznam příloh

- Kompletní zdrojový kód pro Raspberry Pi
- Schéma a návrh propojovací DPS (formát Gerber a Eagle)