VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

FIRMWARE TESTOVACÍ PLATFORMY JAQT
FIRMWARE FOR TESTING PLATFORM JAQT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JIŘÍ VEVERKA
AUTHOR

VEDOUCÍ PRÁCE — Ing. PETR MALANÍK
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Veverka Jiří**
Program: Informační technologie
Název: **Firmware testovací platformy JaQT**
Firmware for Testing Platform JaQT
Kategorie: Vestavěné systémy

Zadání:

1. Seznamte se s problematikou automatického testování HW. Nastudujte protokol USB pro účely komunikace s podpůrným HW. Popište hardwarové vybavení využité pro testování.
2. Definujte požadavky na automatický systém pro testování HW. Navrhněte rozhraní a komponenty systému pro řízené testování HW.
3. Navržený systém implementujte, vyberte konkrétní případy testování a demonstrujte jejich řešení.
4. Výsledky shrňte, popište možné využití v praxi. Diskutujte možné změny a vylepšení jak z pohledu firmwaru tak i HW testovacího zařízení.

Literatura:

- Yahav, E. (2014). Hardware and Software: Verification and Testing, Springer, ISBN: 978-3-319-13338-6
- Wunderlich, H. (2010), Models in Hardware Testing, Springer, ISBN: 978-90-481-3282-9

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Malaník Petr, Ing.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 3. listopadu 2021

Abstrakt

Tato bakalářská práce se zabývá platformou JaQT (Just a Quick Test), která byla vyvinuta za účelem jednoduchého a rychlého řešení problematiky softwarového testování hardware. V teoretické části se práce věnuje popisu použitého mikrokontroléru, řídící desky, rozhraní a periférií. Dále se v teorii zabývá popisem protokolu USB a použité třídy CDC ACM. V praktické části je popsán způsob implementace, ladění, samotné provádění testů a jejich vyhodnocení, nasazení celého systému a konečné zhodnocení výsledků.

Abstract

This bachelor thesis is about the JaQT (Just a Quick Test) platform which was developed for simple and fast software-hardware testing. The theoretical part of this thesis is describing used MCU, control board, peripherals and interfaces. The USB protocol and its CDC ACM class which is used for interfacing the device to enable communication with the board is described in this part as well. In the practical part is described how the JaQT was implemented, debugged, tested, used and evaluated.

Klíčová slova

MCU, mikrokontrolér, expanzní deska, hardware, SPI, I2C, RS232, RS485, Wiegand, USB, CDCACM, libopencm3, STM32, STM32F1, Bluepill, GPIO, testování hardware, softwarové testování hardware, Cortex M3, ARM, GPIO expander

Keywords

MCU, microcontroller unit, expansion board, hardware, SPI, I2C, RS232, RS485, Wiegand, USB, CDCACM, libopencm3, STM32, STM32F1, Bluepill, GPIO, hardware testing, software-hardware testing, Cortex M3, ARM, GPIO expander

Citace

VEVERKA, Jiří. *Firmware testovací platformy JaQT*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Petr Malaník

Firmware testovací platformy JaQT

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Petra Malaníka. Další informace mi poskytli konzultanti Ing. Adam Trhoň a Ing. Radim Pavlík z firmy Touchless Biometric Systems s.r.o. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Jiří Veverka
11. května 2022

Poděkování

Děkuji vedoucímu Ing. Petru Malaníkovi za vedení práce a podnětné rady při jejím vypracovávání. Také děkuji konzultantům z firmy TBS, bez jejichž odborné pomoci by tato práce neexistovala.

Obsah

Kapitola 1

Úvod

Pro účely testování správné funkcionality hardwaru byla vyvinuta aplikace (firmware) pod názvem JaQT (Just a Quick Test), která je určená pro zařízení s MCU (mikrokontrolerem) STM32F1 (ARM architektura Cortex M3). Byla vyvinuta za účelem jednoduchého a rychlého testování hardware bez potřeby jiných specializovaných prostředků. Pro uspokojení požadavků na hardware byla použita vývojová deska nazývaná „Bluepill“, jež obsahuje požadovaný MCU STM32F1. Ta je následně zasazena do převzaté řídící desky.

Řídící deska umožňuje testování rozhraní U(S)ART, RS-232, RS-485, Wiegand, I2C a SPI. Pomoci těchto rozhraní a nastavitelných GPIO („General-purpose input/output“ vstupů je také možné provést testování jednotlivých připojených periférií a vyhodnotit, zda-li fungují správně.

Pro komunikaci MCU s počítačem byl využit protokol USB („Universal Serial Bus“), jehož komunikace byla implementována pomoci USB třídy CDC ACM („Communication Device Class Abstract Control Model“), která byla využita pro realizaci několika virtuálních USB rozhraní (PC↔U(S)ART, PC↔RS-232, PC↔RS-485, PC↔Wiegand/Control Panel).

V praktické části práce je popsán způsob implementace firmware, k čemuž byla využita knihovna libopencm3. Následně je popsán způsob programování MCU a samotné provádění testů nad výsledným zařízením. Na závěr jsou provedené testy vyhodnoceny a je popsán proces nasazení celého systému.

Kapitola 2

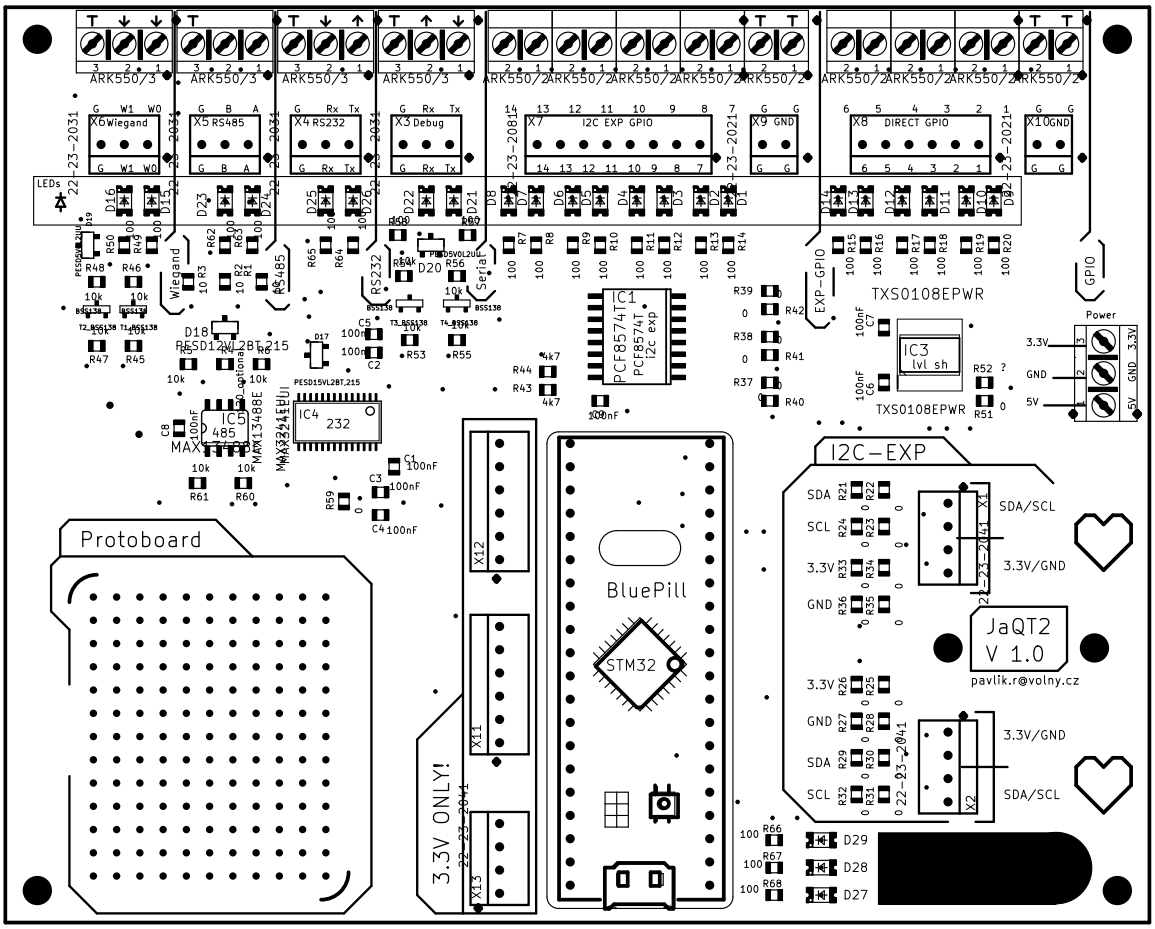
Teoretický rozbor platformy

Pracujeme-li v prostředí, kde neuvažujeme konvenční vývoj zahrnující simulování, může při vývoji hardware může docházet k problémům, kterým by se dalo předejít. Včasným testováním se sníží riziko vzniku vadných kusů na konci výrobního procesu. K problémům může dojít, například, když se v systému změní určitá komponenta (návrh nové součástky či změna v kódu) nebo při vadné výrobě a osazování PCB („Printed Circuit Board“ - deska plošných spojů). Díky automatizaci daného testovacího procesu lze dosáhnout vyšší efektivity při testování. Proces automatizovaného testování hardware by se dal připodobnit k CI („Continuous Integration“ - proces automatizovaného překladu a testování kódu) - testování automaticky probíhá po každé potvrzené změně a vyhodnotí stav systému po jejím aplikování.

Aby bylo možno splnit požadavek na jednoduchou použitelnost a připojitelnost rozšiřujících periférií či testovaných zařízení, je využit návrh expanzní desky[18] od konzultanta Ing. Radima Pavlíka, která je založená nad vývojovou deskou „Bluepill“. Firmware zařízení JaQT je následně implementován pro danou konfiguraci - část dostupné funkcionality zůstane zachována i bez připojené expanzní desky.

2.1 Expanzní deska

Schéma expanzní desky[18] bylo převzato z veřejně dostupných a volně použitelných nákresů jednoho z konzultantů této práce - Ing. Radima Pavlíka z firmy TBS. Pro použití této desky bylo rozhodnuto na základě rychlé podpory v případě problémů a její jednoduchosti pro použití. Deska poskytuje rozšíření o 8 GPIO vstupů pomoci I2C GPIO expanderu, dále 6 vstupů GPIO vyvedených přímo z použitého MCU, sériové rozhraní pro ladění realizované pomoci rozhraní U(S)ART. Vstup a výstup rozhraní RS-232 a RS-485 a připojení pro komunikační rozhraní protokolu Wiegand. Expanzní deska dále obsahuje vstupy pro připojení periférií komunikujících pomoci sběrnice I2C či SPI či vstupy pro ADC, které převádí vstupní analogový signál na digitální. Osazovací nákres s popisem jednotlivých součástí lze vidět na obrázku 2.1.



Obrázek 2.1: Osazovací nákres expanzní desky JaQT, převzato z: [18]

2.2 Minimalistická vývojová deska „Bluepill“

„Bluepill“ je neoficiální komunitní název označující minimalistickou výkonnou vývojovou desku s mikrokontrolerem STM32F103CxTx. Jedná se o mikrokontroler s architekturou jádra Cortex-M3[33].
Vyrábí se ve dvou verzích STM32F103C6T6 a STM32F103C8T6. Verze využitá v této

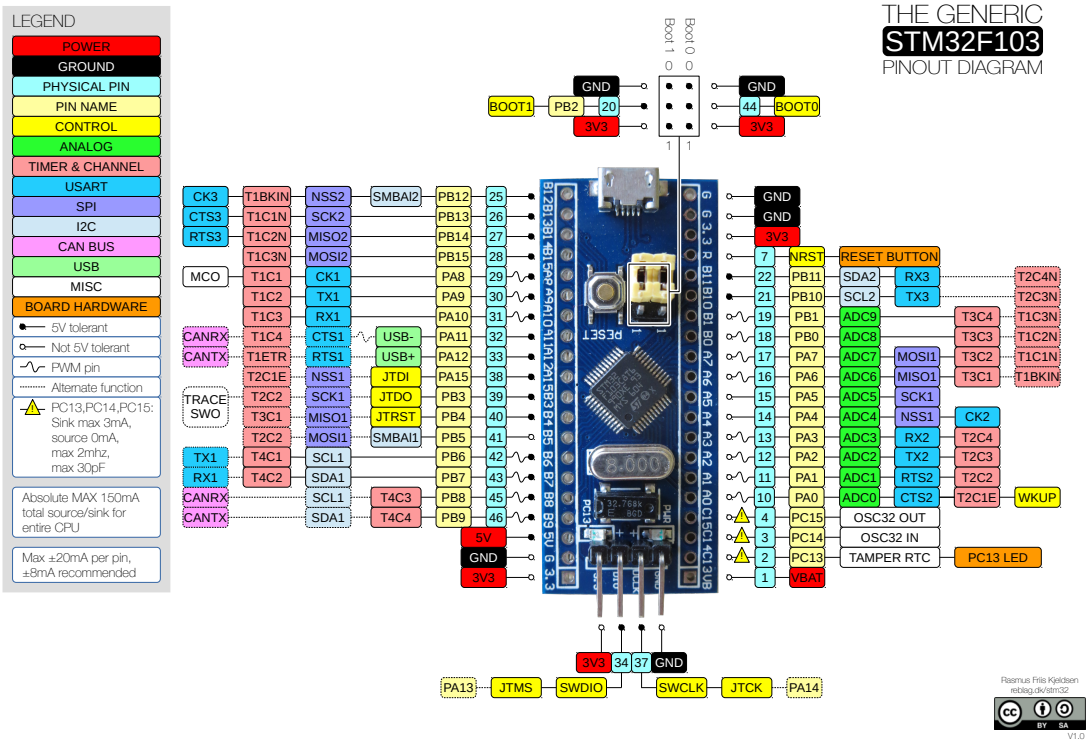
práci je STM32F103C8T6, jelikož disponuje vyšším počtem sériových portů a větší velikosti FLASH paměti. V tabulce 2.1 lze vidět porovnání obou verzí, data byla převzata z datasheetů, konkrétně z [27] pro STM32F103C6T6 a z [33] pro STM32F103C8T6. Použitý mikrokontroler může fungovat při maximálním taktu 72MHz, čehož bylo při implementaci plně využito. Vývojová deska obsahuje jeden konektor typu mikro USB pro připojení k počítači a celkem 37 GPIO vstupů, které lze naprogramovat s přerušením a časovačem, případně jako některé z následujících interně nastavitelných rozhraní:

- 2x SPI
- 3x U(S)ART
- 2x I2C
- 1x CAN
- 2x ADC

„Bluepill“ rovněž disponuje rozhraními JTAG a SWD, které lze využít k ladění aplikace běžící na mikrokontroléru či k nahrání firmware do paměti. V této práci bylo využito rozhraní SWD k nahrání bootloaderu s podporou USB DFU (device firmware upgrade), jež umožnil nahrávání či aktualizace firmware přes integrovaný mikro USB konektor.[6]

Porovnání verzí MCU STM32F1		
Verze MCU	STM32F103C6T6	STM32F103C8T6
FLASH	32 kB	64 kB
SRAM	10 kB	20 kB
Časovače	3	4
Sériové porty	2	3

Tabulka 2.1: Porovnání používaných MCU řady STM32F1.



Obrázek 2.2: Vstupy a výstupy vývojové desky „Bluepill“, převzato z: [21].

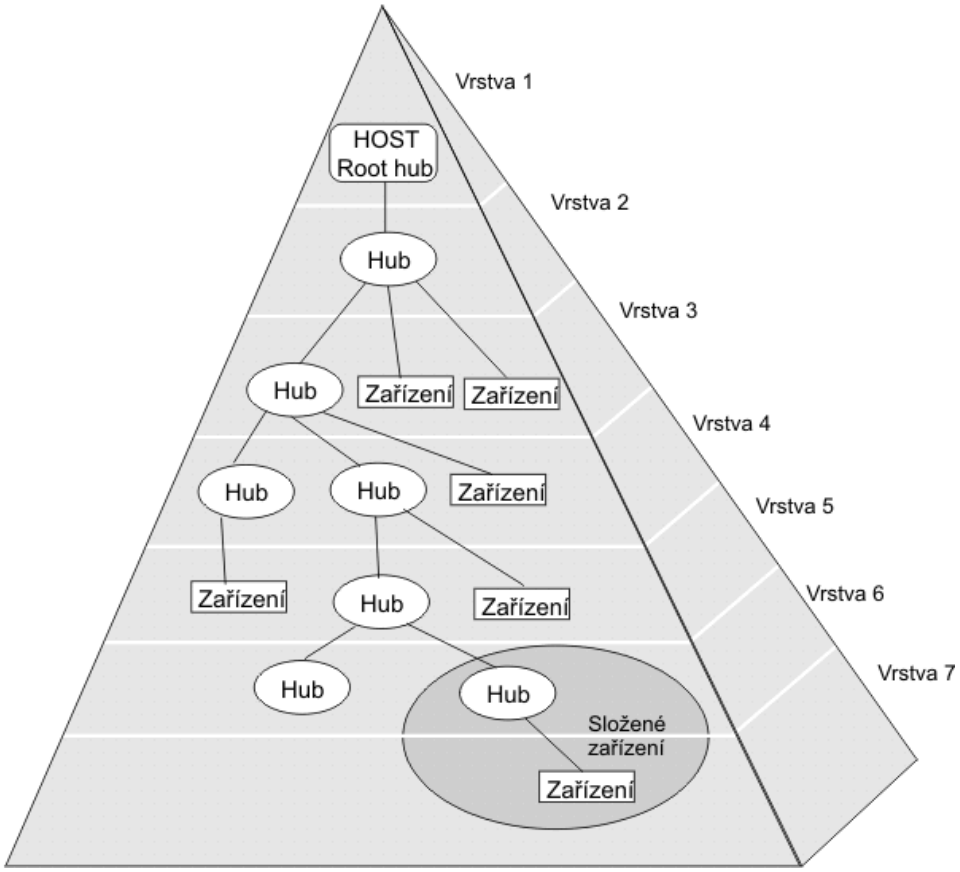
2.3 Rozhraní USB

Jelikož existuje mnoho druhů sběrnic a konektorů, bylo za účelem jejich sjednocení vytvořeno USB („Universal Serial Bus“). Došlo k tomu pomoci spolupráce několika velkých firem v IT oboru (převážně pod vedením Intelu) a koncem roku 1995 začaly být počítačové desky vybavovány vyvinutým univerzálním konektorem[5]. Aktuální verze standardu je USB 3.2 která je zpětně kompatibilní se všemi předchozími verzemi. USB je flexibilní a tudíž se dá použít celou řadou způsobů - pro komunikaci s periferiemi (klávesnice, počítačové myši, tiskárny, atp.), jako HUB (rozvětvení jednoho konektoru na více), pro připojení k internetu, pro napájení či dobíjení zařízení a dále.

Dnes již existuje celá řada konektorů USB, přičemž mezi nejrozšířenější patří typ USB-A (obrázek 2.3). Dále je velmi rozšířený konektor typu μUSB (mikro USB) a USB-C, které se využívají například k dobíjení a přenosu dat z paměti mobilních telefonů, fotoaparátů, laptopů a v neposlední řadě také pro komunikaci a programování integrovaných obvodů (mikrokontrolery či složitější vestavěná zařízení), čehož je v této práci využito. Příklad jednotlivých konektorů lze vidět na obrázku 2.3.

Obrázek 2.3: Druhy USB konektoru, převzato z: [3].

Po technické stránce USB využívá vrstvenou hvězdicovou topologii[10] (obrázek 2.4).



Obrázek 2.4: Topologie sběrnice USB, převzato z: [10].

Podle Michaela Gooka se tudíž jako sběrnice se dá označovat pouze po logické nikoliv elektrické stránce. Je řízená hostitelským počítačem / zařízením, které zpravidla uskutečňuje komunikaci v režimu polling, aby se žádné zařízení nemohlo inicializovat samo od sebe

a muselo vyčkat na patřičný požadavek od hostitele. Pojem polling znamená, že zařízení se neustále ve smyčce dotazuje na nová příchozí data. Aby nedocházelo ke ztrátě dat během dotazování na jiná zařízení, je nutná implementace vnitřních FIFO bufferů (typ bufferu kde první příchozí bit jej opouští jako první, také se mu říká „řada“) na zařízení.[5]

Předtím, než host může začít komunikovat s zařízením, je potřeba dodat informace o zařízení a přiřadit správný ovladač. Tomuto procesu se říká „enumerace“ a jedná se o počáteční výměnu informací o typu a vlastnostech zařízení.

Během tohoto procesu dochází k:

- přiřazení adresy zařízení,
- vyčtení datových struktur ze zařízení,
- přiřazení a načtení odpovídajícího ovladače zařízení,
- výběr konfigurace ze seznamu možností dodaných zařízením.

Po provedení výše zmíněných kroků je zařízení připraveno začít přijímat a vysílat data na kterémkoliv ze svých definovaných koncových bodů („endpointů“) - jedná se o adresu bufferu v zařízení pro příjem dat či data čekající na odeslání[2]. Existují celkem 3 druhy „endpointů“ popsaných v tabulce níže (tabulka 2.2).

Typ transakce	Zdroj dat	Typ přenosů využívajících transakci	Obsah dat
IN	Zařízení	Všechny	Obecná data
OUT	Host	Všechny	Obecná data
Setup	Host	Inicializace a ovládání komunikace	Požadavky

Tabulka 2.2: Druhy USB endpointu, převzato z: [2]

Endpoint „Setup“ je specifický tím, že se používá k inicializaci a následnému ovládání datového přenosu, přičemž host nemůže daná data odmítnout.[2]

V této práci používám verzi USB standardu 2.0 a fyzické propojení pomoci typů USB a μUSB, jelikož vývojová deska disponuje druhým zmíněným konektorem, pro komunikaci (podkapitola 2.2).

2.3.1 Třídy USB zařízení

Různé druhy USB zařízení často sdílí stejné vlastnosti. Když takové zařízení pak sdílí i stejné požadavky na komunikaci, tak se dají seskupit do společné třídy, která je popisuje - proto vznikly třídy zařízení USB (USB device classes), jež obecně definují skupiny takovýchto zařízení[2].

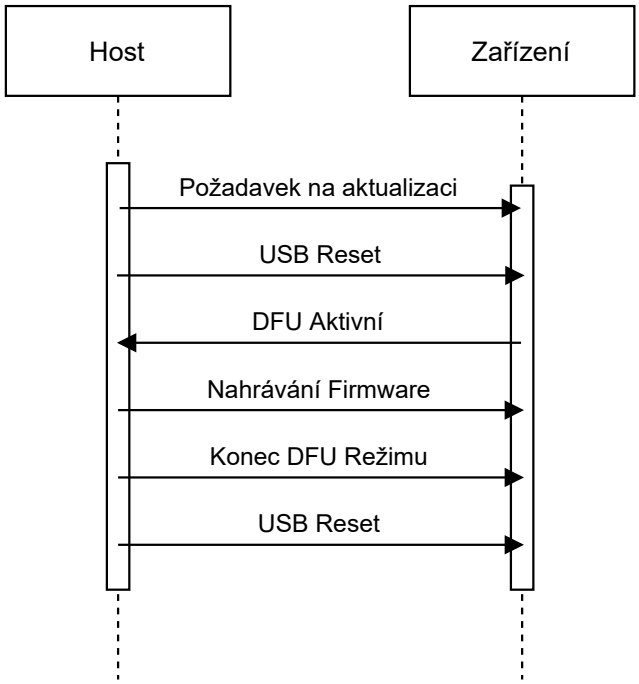
Tříd USB zařízení existuje vícero druhů, v této práci jsou však popsané pouze dvě, které byly při vývoji použity. Jedná se o třídy DFU („Device firmware upgrade“) a abstraktní řídící model komunikačních zařízení (CDC ACM).

DFU

Třída DFU (Device Firmware Upgrade), jak napovídá její název, slouží k aktualizaci programu (firmware) v zařízení. Zařízení, které vyžadují tuto funkcionalitu pomoci USB musí tedy implementovat a nakonfigurovat třídu DFU.

Jelikož je nežádoucí, aby během běhu programu zároveň probíhala i jeho aktualizace, je třeba zajistit, aby tyto dvě operace neprobíhaly současně - to se řeší tak, že po dobu aktualizace dané zařízení přestane vykonávat svou činnost (běh programu), přepne svůj režim do DFU a umožní provedení aktualizace. Zařízení není schopno započít aktualizaci samo - je potřeba, aby dostalo pokyn od uživatele nebo řídícího zařízení.[36]

Vizualizaci relace DFU aktualizace lze vidět na obrázku 2.5 níže.

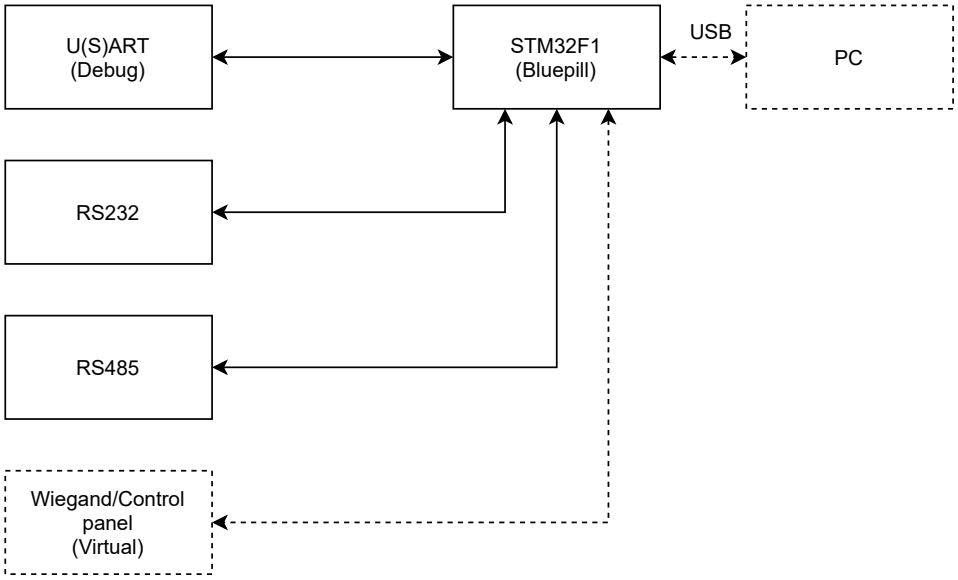


Obrázek 2.5: Příklad relace DFU, převzato z: [36]

CDC ACM

CDC („Communication Device Class“) je třída která umožňuje telekomunikačním a síťovým zařízením se připojit a komunikovat s hostem (řídícím zařízením). Specifikuje vícero modelů pro podporu různých druhů komunikace. Jedním z nich je ACM (Abstract Control Model), jehož vlastností je emulace sériového rozhraní, pomoci čehož je možné vytvořit softwarovou abstrakci nad USB zařízením pro komunikaci[37]. V rámci práce je tato třída využita pro abstrakci sériových rozhraní, které použitá vývojová deska využívá a na kterých probíhá sériová komunikace. Vizualizaci ACM rozhraní pro komunikaci s počítačem lze vidět na obrázku 2.6 níže, přičemž rozhraní, které mají svůj fyzický protějšek jsou zvýrazněný plnou čarou. Poslední rozhraní (znázorněno přerušovanou čarou) je virtuální a má dvě funkcionality:

- Abstrakce nad rozhraním protokolu Wiegand (podkapitola 2.7).
- Kontrolní panel přijímající příkazy pro řízení systému.



Obrázek 2.6: Vizualizace rozhraní a směr jejich komunikace,

2.4 GPIO

GPIO („General purpose input/output“) je druh vstupu, jehož způsob využití není nijak předem definován. Jeho základní funkcionalitou je jednoduché čtení a zápis binárních hodnot (pomoci signálů „zapnuto“ - 1 a „vypnuto“ - 0). Této vlastnosti lze využít vícero způsoby - mimo již zmíněný zápis a čtení jednoduchých binárních hodnot lze nad GPIO implementovat i plnohodnotné sběrnice (například SPI či v této práci použitý Wiegand). Dvoustavové řízení lze použít například pro rozsvěcování stavových LED světel (například pro účely vizuální detekce, zda-li je systém aktivní), pro spínání relé ovládající vysokonapěťové zařízení či například rozsvěcování individuálních segmentů LCD displeje. Na expanzní desce je dostupných celkem 14 vstupů GPIO, přičemž 6 jich je vyvedených napřímo z vývojové desky (podkapitola 2.2) přes měnič napěťových úrovní z 3.3V na 5V a zbylých 8 je realizovaných pomoci rozhraní I2C (podkapitola 2.6) a GPIO expanderu PCF8574[17]. Konkrétní GPIO vstupy vývojové desky „Bluepill“ lze vidět na obrázku 2.2.

2.5 Testovací rozhraní

JTAG

Testovací rozhraní JTAG („Joint Test Action Group“) je standardizované rozhraní pro testování komplexních logických obvodů (procesory, FPGA, atp.). Na základě toho, že JTAG TAP (tabulka 2.3) lze nalézt a je definovaný ve většině digitálních integrovaných obvodů lze předpokládat, že se jedná pravděpodobně o nejrozšířenější testovací rozhraní integrovaných obvodů[34]. Jedná se o synchronní a paralelní rozhraní s 4 povinnými a 1 nepovinným signálem. Povinné signály jsou - TCK (testovací hodiny udávající synchronizační frekvenci), TMS (testovací mód - vybírá jej testující zařízení/řadič), TDI (vstup testovacích dat v binárním kódu, LSB („Least Significant Bit“ - nejméně důležitý bit je první), TDO (výstupní data z testovaného zařízení). Nepovinný signál TRST se stará o reset testovaného uzlu a generuje jej testující zařízení. Testované zařízení se k řadiči připojuje pomoci portu TAP,

přes který přijímá testovací signály, předem definované programátorem a následně generované řadičem. Rozhraní kromě testování také umožňuje programování zařízení do RAM („Random Access Memory“ - operační paměť, ukládá data pouze po dobu kdy je zařízení napájeno) či FLASH (Programovatelná paměť - ukládá data natrvalo, lze je přepisovat)[5]. Ačkoliv dané rozhraní nikde později ve své práci nepoužívám, použité MCU jej poskytuje a je možné jej použít jako alternativu SWD (podkapitola 2.5) k programování zařízení.

JTAG TAP (Test Access Port)		
Signál	Směr komunikace	Popis
TCK	Z řadiče	Hodiny udávající synchronizační takt
TMS	Z řadiče	Výběr testovacího režimu
TDI	Z řadiče	Vstupní data testování v binární podobě
TDO	Do řadiče	Výstupní data testování v binární podobě
TRST	Z řadiče	Nepovinný resetovací signál

Tabulka 2.3: Port JTAG TAP

SWD

Jedná se o synchronní rozhraní sloužící k ladění, testování a programování integrovaných obvodů a čipů - podobně jako JTAG (podkapitola 2.5). Využívá však jen jeden vodič pro přenos dat a jeden vodič přenášející hodinový signál udávající synchronizační takt. Dále je ještě potřeba uzemňovací vodič GND a případně napájecí vodič, jestliže testované zařízení není již napájeno a uzemněno jiným způsobem (tabulka 2.4).

Signál	Popis
Napájení (volitelný)	Provozní napětí čipu
SWDIO	Datový vodič
SWCLK	Vodič s hodinovým signálem
GND	Uzemnění čipu

Tabulka 2.4: SWD port

Průběh komunikace mezi řídícím zařízením a testovaným zařízením probíhá ve dvou až třech fázích:

- Packet request: Řídící zařízení vyšle testovanému požadavek na SWD port
- Acknowledge response: Testované zařízení potvrdí řídícímu jeho požadavek
- Přenos dat: Tato fáze probíhá pouze pokud je na požadavek zodpovězeno potvrzující zprávou (OK) nebo je nastavený příznak v registru, který vynutí odpovídání na všechny příchozí požadavky

Samotný přenos dat může obsahovat pouze pakety čtení z testovaného zařízení nebo zápis na něj, nikoliv naopak[1]. SWD je podobně jako JTAG také velice rozšířené, avšak jedním z hlavních důvodů je jeho menší počet vstupů oproti druhému zmíněnému - výrobci mohou vzhledem k tomu uvolnit vyšši počet vstupů pro jiné funkcionality než testování či programování daného integrovaného obvodu.

2.6 Vstupní a výstupní rozhraní

Použitá expanzní deska rozšiřuje již existující rozhraní MCU o několik dalších, ve výsledku je tedy dostupné:

- 1x SPI,
- 1x U(S)ART (jako sériová sběrnice pro debug připojeného zařízení),
- 1x RS-232,
- 1x RS-485,
- 1x Wiegand,
- 2x I2C.

Z hlavních komunikačních rozhraní které jsou vestavěny v MCU nebyl využit AD převodník a sběrnice CAN.

U(S)ART

U(S)ART znamená „Universal Synchronous/Asynchronous Receiver-Transmitter“ (Univerzální synchronní/asynchronní přijímač/vysílač). Jedná se o široce využívané rozhraní, které se používá k sériové komunikaci[5]. Hlavním rozdílem mezi UART a U(S)ART je, že U(S)ART podporuje jak asynchronní tak i synchronní přenos, kdežto UART pouze asynchronní.

Podle autora Hardwarového průvodce [5] přenos dat rozlišujeme na:

- Synchronní přenos
  - synchronizační informace přesně udávají příští platný bit (kontrola validity),
  - používá hodinový a datový signál,
  - může komunikovat při různých přenosových rychlostech (přijímač je schopen se dynamicky přizpůsobit) až do maximální možné frekvence,
  - přenos typicky probíhá po blocích avšak může i po bytech,
  - podporuje plně duplexní i polo-duplexní komunikaci,
  - umožňuje vyšší rychlosti přenosu.
- Asynchronní přenos
  - minimum synchronizačních informací (start bit, stop bit),
  - vysílač a přijímač musejí pracovat se stejnou, předem domluvenou taktovací frekvencí,
  - přenos probíhá po bytech,
  - plně duplexní komunikace,
  - omezená přenosová rychlost.

Synchronizační informace asynchronního přenosu se skládají ze start bitu indikujícího začátek datového rámce a alespoň jednoho stop-bitu označujícího jeho konec. Dále se jak v synchronním tak i v asynchronním přenosu mimo datových bitů vyskytují bity paritní sloužící pro omezenou validaci integrity datového rámce, což do jisté míry pomáhá detekovat a chránit vůči chybám při přenosu[8].

Občas se chybně uvádí, že UART je jenom alternativní název pro rozhraní RS232, avšak toto tvrzení je mylné - UART definuje podobu komunikačního protokolu (datová rychlost, podobu dat při přenosu, atd.) kdežto RS232 definuje způsob samotného přenosu (definuje například úrovně napětí přenosových signálů). Signál z UART je však možno přenášet pomoci RS232, RS485 či RS422.

I2C

Součástí jak MCU tak i expanzní desky jsou dva vstupy/výstupy na sběrnici I2C, avšak na expanzní desce je prostor pro rozšíření o další. Tato sběrnice umožňuje zařízení v režimu Master (v tomhle případě platforma JaQT) komunikaci s více zařízeními v režimu Slave.

Rozpoznání jednotlivých zařízení v režimu Slave probíhá pomoci adresace - každé takové připojené zařízení má vlastní 7bitovou adresu pomoci které ji Master rozlišuje a komunikuje. Samotná komunikace na sběrnici funguje v režimu halfduplex - data ze zařízení v režimu Master mohou být odeslána či přijata kdykoliv avšak zařízení v režimu Slave pouze odpovídají na příchozí požadavky z Masteru.

Pro propojení a přenos dat na sběrnici I2C se využívá dvou vodičů - datový vodič (SDA) a vodič s hodinovým signálem (SCL), jehož frekvence zpravidla bývá 100kHz nebo 400kHz. Vývojová deska „Bluepill“ umožňuje komunikaci při napětích 3.3V a 5V a hodinovém taktu v rozmezí 100kHz - 800kHz.[5]

SPI

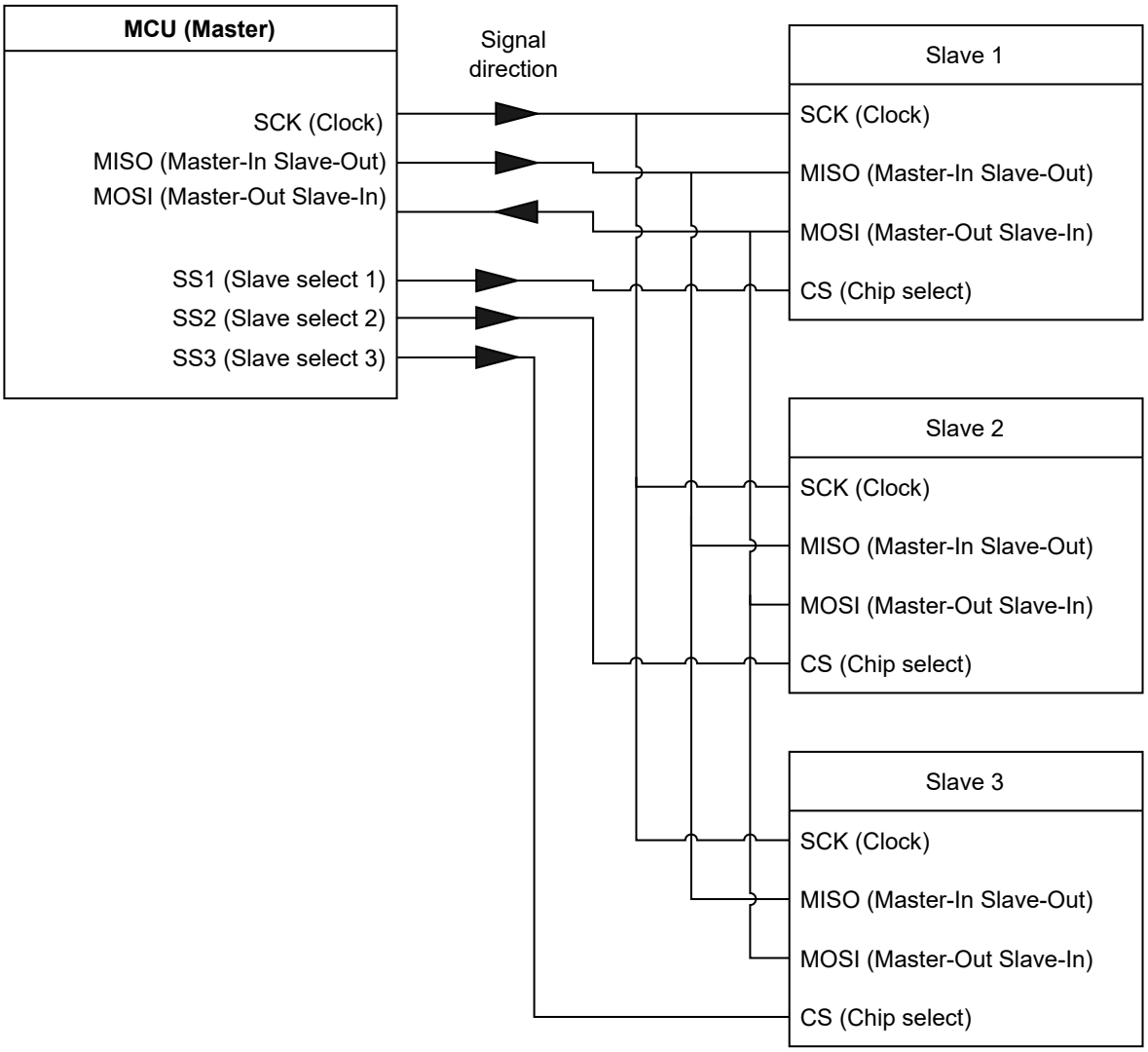
Rozhraní SPI, které se také občas nazývá „Microwire“ je synchronní rozhraní o 4 povinných a 1 nepovinném vodiči. Disponuje samostatnými linkami pro vstup a výstup a umožňuje nejen komunikaci mezi řadiči, periferiemi, mikročipy ale i například programovat FPGA obvody. SPI je rychlejší než již zmíněné I2C (2.6), nejen díky tomu, že během každého impulsu hodinového signálu může zároveň číst i zapisovat ale i díky samotné synchronizační frekvenci. Výhodou použití je, že podobně jako u I2C může být na jednu sběrnici připojeno více zařízení, přičemž nejčastěji se používá topologie o jednom řídícím zařízení (master) a více periférií (slaves). V praxi se můžeme setkat i s variantou připojení, kdy se na jedné sběrnici nachází více zařízení typu master - v takovém případě ale protokol přístupu k datům není obecně určen[5].
Povinné vodiče SPI:

- SCK - Hodinový taktovací signál udávající vzorkovací frekvenci,
- MOSI - „Master-Out Slave-In“ datový signál ve směru od řídícího zařízení,
- MISO - „Master-In Slave-Out“ datový signál ve směru do řídícího zařízení,
- GND - Uzemnění, vůči kterému se odvíjí hodnota ostatních signálů.

Dále se můžeme setkat s nepovinným signálem označeným jako SS („slave select“ - na straně master zařízení) nebo CS („chip select“ - na straně slave), který slouží pro výběr periférie se kterou v danou chvíli chce řídící zařízení komuniovat. V případě použití signálu

SS periférie odpovida (je aktivní) pouze v případě kdy je daný signál stažený na nizkou logickou uroveň[5]. Příklad propojení řídícího zařízení a periférií lze vidět na obrázku 2.7.

Rozhraní je plně duplexní, přičemž jednou z nevýhod je, že neexistuje moc možností, jak kontrolovat chybovost přenosu, jelikož neprobíha potvrzeni doručení dat. Dalším omezením při použití SPI je maximální vzdálenost přenosu, která je například oproti RS-485 nízká.[14].



Obrázek 2.7: Příklad připojení periférií k zařízení pomoci rozhraní SPI, převzato a upraveno z: [9]

RS-232

Cílem standardu rozhraní RS-232 je propojit dvě zařízení pro vzájemnou sériovou komunikaci pomoci přijímacího signálu na vstupu zařízení (Rx) a vysílacího signálu na výstupu zařízení (Tx).

„Tento způsob je vhodný pro point-to-point komunikaci při nízkých rychlostech. Například na port COM 1 u pc může být připojená myš, na COM 2 modem atd. To je příklad point-to-point komunikace, jeden port, jedno zařízení.“[24]

Ve verzi RS-232-C je v případě signálu Rx hodnota logické 1 udávaná úrovní napětí mezi -12 a -3V, přičemž hodnota logické 0 je mezi úrovněmi napětí 3 a 12V. Pro signál Tx jsou tyto logické hodnoty převráceny - tedy mezi 3 a 12V se nachází logická 1 a mezi -12 a -3V logická 0. Rozsah napětí mezi -3V a 3V je tzv. mrtvá zóna - změna stavu signálu je rozpoznána pouze při překročení jedné z mezních úrovní napětí.

Uzemňovací vodič GND slouží pro správnou detekci rozdílu v napětí - jeho úroveň na vodičích Rx a Tx se vztahuje vůči uzemňujícímu vodiči.

Jelikož oba signály vyžadují sdílený uzemňovací vodič, není vhodné tento standard používat pro přenosy na vyšší vzdálenosti - kvůli narůstajícímu odporu a rušení se stoupající délkou vedení.
Připojení zařízení je silně doporučeno provádět pouze ve stavu, kdy jsou obě zařízeni vypnutá, jelikož nevyvážený rozdíl v napětí by se mohl v okamžiku připojení projevit negativně na signálu výstupu nebo dokonce poškodit interní obvody zařízení.[5]
Na expanzní desce je použit integrovaný obvod MAX3241EUI, který operuje v hladinách napětí od +3/-3V do +5,5/-5,5V, a stará se o převod signálu o maximální rychlosti 120kb/s z rozhraní RS-232 na UART rozhraní mikrokontroleru.[16] Standard RS-232 umožňuje komunikaci v synchronním i asynchronním režimu.

RS-485

Komunikační standard rozhraní RS-485 je odlišný od RS-232. Sdílí sice stejný základ, avšak logická úroveň napětí se určuje odlišně. Pro přenos jednoho signálu se používá dvou vodičů, typicky kroucených okolo sebe (tzv. kroucená dvoulinka - v angličtině TwistedPair) - jeden z vodičů je označen jako A a druhý jako B.[24]

Pro určení logické hodnoty se využívá rozdílového napětí mezi oběma vodiči - logická úroveň je definována dle rozdílu napětí mezi vodičem A a vodičem B. Je-li výsledné napětí A - B $<$ -200mV, pak se jedná o stav logické 0, jestliže však je výsledné napětí $>$ 200mV, signál se nachází ve stavu logické 1. V průmyslu se nejčastěji využívají úrovně 0V - 5V (0V na vodiči A a 5V na vodiči B) v klidovém stavu.[24]

Rozdíl oproti RS-232 je také v tom, že namísto od point-to-point komunikace se jedná o typ sběrnice, u které lze připojit více zařízení či uzlů (Slave) na jeden vysílací vodič či zařízení (Master). Jedná se o nejrozšířenější způsob použití RS-485, kdy každý uzel má vlastní adresu na kterou se řídicí zařízení následně při komunikaci s konkrétním uzlem odkazuje - slave nikdy nemůže započít komunikaci sám od sebe, odpovídá pouze na dotazy mastera. Další častý způsob použití je, že řídicí zařízení odesílá zprávy všem a připojené uzly na ně pak případně patřičně reagují. V praxi se občas také využívá dvou párů přenosových vodičů tzv. „Double TwistedPair“, jejichž funkcionalita je pak mírně odlišná - zatímco na jedné dvojici vodičů probíhá zasílaní rámců od řídícího zařízení k uzlům, na druhé dvojici uzly odpovídají nazpět.[24]

Maximální počet uzlů na jedné sběrnici je 32, přičemž maximální délka přenosu se pohybuje okolo 1200m - to je také jeden z důvodů, proč je tento standard především využíván v průmyslu[14].

Samotný přenos dat probíhá v rámcích o velikosti 7 nebo 8bitů. Začátek rámce je označen 1bitem (start-bit) a konec rámce 1 nebo 2bity (stop-bit).

Pro zajištění správné funkcionality při testování, je na expanzní desce využit integrovaný obvod MAX13488E, který se vyznačuje garantovanou bezchybnou rychlostí přenosu až 500kb/s a má na starost správný převod signálu ze sběrnice RS-485 na UART rozhraní mikrokontroleru.

Další podstatnou vlastností použitého integrovaného obvodu je eliminace nebezpečí při zapojení pod napětím, podporuje totiž tzv. „hot-swap“ a nehrozí tudíž poškození interních obvodů zařízení či zkreslení přijímaného signálu.[15]

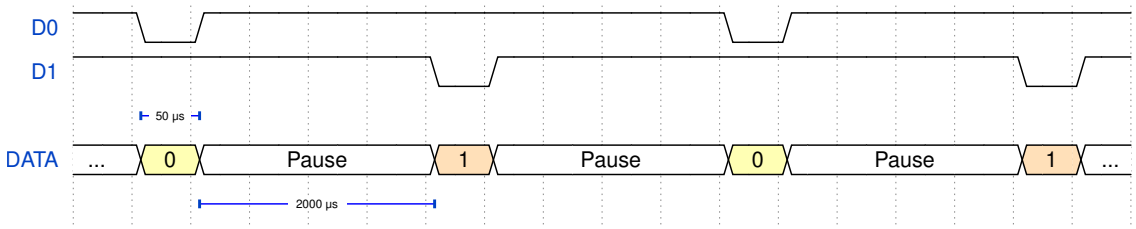
2.7 Komunikační protokol Wiegand

Dle zdroje [19] se označení Wiegand se v průmyslu využívá pro více technologií, mezi něž patří:

- rozhraní,
  - Čtečka - Karta
  - Čtečka - MCU
- komunikační sběrnice a protokoly,
- magnetické karty (karty s magnetickým proužkem),
- elektromagnetický jev tzv. Wiegandova drátu (V praxi se člověk může setkat také s označením „Wiegandův jev“ či „Wiegandův efekt“).

V této sekci se budu zabývat rozhraním Čtečka↔MCU a komunikačním protokolem k sběrnici Wiegand. Samotné rozhraní mezi čtečkou a MCU je realizováno pomoci tří vodičů, přičemž jeden z nich je zemnící (GND) vůči kterému se odvíjí potenciál napětí na zbylých dvou. Zbylé dva vodiče jsou označeny jako D0 (Data 0) a D1 (Data 1). Oproti sběrnicím, kde se jeden vodič stará o komunikaci směrem k zařízení (Rx) a druhý o komunikaci ze zařízení (Tx), zde se oba vodiče starají o příjem bitových rámců do zařízení zpracovajícího příchozí signál, přičemž na vodiči D1 přicházejí bity s logickou hodnotou 1 a na D0 s logickou hodnotou 0. V klidovém stavu jsou oba vodiče ve stavu HIGH, který se zpravidla nachází na napětové úrovní 5V (avšak občas se můžeme setkat i s vyššími hodnotami, např. 12V či 24V). Příjem dat následně probíhá stažením příslušného vodiče do stavu LOW, který se nachází na nulové napěťové úrovni. Mimo napěťové úrovně HIGH a LOW je také definován čas, po který dochází k příjmu validní hodnoty (typicky 50μs) a definovaný mezičas určující dobu do příjmu hodnoty následné (typicky 2000μs). Jestliže dojde k příjmu dat v mezičase či k příjmu dat na obou vodičích najednou, jedná se o chybu systému, kterou je nutné ošetřit.[7]

Grafický příklad průběhu signálů na vodičích D0 a D1 a reprezentaci přijatých dat lze vidět na obrázku 2.8.



Obrázek 2.8: Vizualizace signálů na vodičích a data

Komunikační protokol definuje datagram o velikosti 26bitů. Jeho první bit je MSB („Most significant bit“) a poslední LSB („Least significant bit“). Parita MSB je vypočtena z první poloviny celého 26bitového datagramu (tzn. prvních 12bitů po MSB), přičemž hodnoty 1 nabývá, když se v ní nachází **lichý** počet jedniček. Jinak 0. Podobným způsobem je vypočten i LSB, avšak z druhé poloviny datagramu (posledních 12bitů před LSB) - hodnoty 1 nabývá, když se v ní nachází **sudý** počet jedniček, jinak 0.

Parita v rámci datagramu																									
Bit datagramu	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
Význam / parita	MSB	1												2												LSB

Tabulka 2.5: Parita v rámci datagramu

Kapitola 3

Požadavky a návrh firmware pro JaQT

V praxi se často stává, že při vývoji zařízení dochází ke změnám v návrhu a dané změny je následně třeba otestovat (například změna hardware součástek či aktualizace kernelu).

Pro testování správné funkcionality hardwaru je potřeba navrhnout a implementovat firmware pro expanzní desku(2.1) dle specifikovaných požadavků. Před samotnou tvorbou návrhu byl proveden průzkum vhodných technologií pro implementaci, ze kterých je jako nejvhodnější vybrána knihovna LibOpenCM3(3.2) a jako základ celého návrhu je využito informací z článku[22] a kódu[23] uživatele satoshinm.

3.1 Požadavky na výsledný systém

Obecné požadavky na systém jsou následující:

- Potřeba testovat periferie připojené k zařízení - v důsledku změn v hardware nebo software může docházet k problémům.
- Testování časově omezených požadavků (například Wiegand).
- Možnost rychlé a spolehlivé aktualizace firmware testovací platformy.

Konkrétní požadavky na systém vyplývají z použité expanzní desky:

- Podpora rozhraní protokolu Wiegand,
- Podpora komunikace přes sériové rozhraní U(S)ART, RS-232 a RS-485,
- Podpora ovládání GPIO,
- Podpora periférií komunikujících přes I2C.

Expanzní deska dále obsahuje rozhraní SPI, avšak tato problematika se nenachází v požadavcích na výsledný systém.

3.2 Existující knihovny a přístupy

Pro vývoj firmware na architektuře ARM STM32 existuje již několik knihoven nabízejících abstrakci nad registry mikrokontroleru. Tyto knihovny se liší způsobem použití, mírou abstrakce a kvalitou dokumentace. V podkapitolách níže jsou vyjmenovány a popsány knihovny běžně používané pro vývoj firmware na architekturu použitého mikrokontroleru STM32.

LibOpenCM3

LibopenCM3 je komunitní projekt, který má za cíl vytvořit volně přístupnou knihovnu sloužící k vývoji firmware pro různé mikrokontrolery postavené na architektuře ARM Cortex[12] a mezi podporované patří i použitý STM32F103C8T6. Knihovna je rozsáhle dokumentována a má k dispozici celou řadu příkladů implementace, ze kterých se při vývoji dá vycházet. Míra abstrakce nad registry je dostatečně odstíněná od funkcí nejnižší úrovně - programátor nepotřebuje znát názvy registrů se kterými pracuje, avšak pro případ potřeby zároveň umožňuje jednoduchý zásah do systému přímo na úrovni registrů. Nevýhodou knihovny je, že podporuje pouze vybrané mikrokontrolery a je stále ve vývoji, tudíž může docházet ke změnám v rozhraní (což může způsobit nekonzistenci v kódu a potenciální problémy při překladu).

STM32 HAL

STM32 HAL („Hardware Abstraction Layer“) je oficiálně podporovaný způsob programování mikrokontroleru od firmy ST. Poskytuje vrstvu abstrakce nad nízko-úrovňovými registry a umožňuje programátorovi rychle navrhnout a naprogramovat výslednou aplikaci. Každá série mikrokontroleru má vlastní manuál, který dopodrobna popisuje všechny aspekty dané série[29]. Firma ST nabízí volně ke stažení nástroj STM32CubeMX, který dle grafické konfigurace umí vygenerovat základní kód implementující požadované chování[32]. Výhodou je již zmíněná oficiální podpora a podrobná dokumentace. Nevýhodou je komplikovanost použití.

STM32duino

Jedná se o přístup implementující rozhraní pro implementaci v ArduinoIDE[1]. Základem je standardní knihovna pro STM32 periférie dodávaná výrobcem[2] nad kterou jsou implementovány abstrakce umožňující jednoduchý vývoj a podporu v ArduinoIDE. Jde o komunitní projekt, který je rozvíjen v rámci fóra[3] a podporuje široké spektrum mikrokontrolerů STM32. Výhodou je aktivní komunita, což umožňuje dobrou podporu při práci na projektu. Nevýhodou je minimální dokumentace.

3.3 Způsoby programování paměti zařízení

Pro naprogramování zařízení jsou nutné dva nástroje - programátor (zařízení sloužící k programování či debugování hardware) a příslušný software pro nahrávání. Jelikož vývojová deska disponuje vyvedenými vstupy rozhraní SWD (obrázek 2.2), je jednodušší použít tento standard namísto JTAG při programování firmware do paměti čipu. V rámci této práce je

[1] https://www.arduino.cc/en/software
[2] https://www.st.com/en/embedded-software/stsw-stm32054.html
[3] https://www.stm32duino.com/

použitý oficiální programátor ST-LINK/V2 od firmy ST[30]. Během vývoji bylo zjištěno, že v nouzovém případě lze využít i běžný FTDI převodník nebo kabel připojený na primární U(S)ART rozhraní (vstup A9 a A10) a manuálně uvést čip do programovacího módu přesunutím svorky na vstupu BOOT0 do stavu HIGH - v dané konfiguraci lze nahrát program pomoci vestavěného bootloaderu od výrobce[26]. Variace konfigurací BOOT vstupů jsou znázorněny v tabulce 3.1.
Při průzkumu existujících řešení pro naprogramování STM32 byly objeveny následující nástroje:

- Open OCD (On-Chip Debugger) - otevřený software pro debug a programování hardware[20].
- Oficiální nástroj od firmy ST - STM32CubeProgrammer, který existuje ve variantách s grafickým rozhraním a rozhraním v příkazové řádce[28].
- Texane STLINK - Open-source sada nástrojů pro programování a debug STM32[25].

Aktualizaci programu přes rozhraní USB DFU (podkapitola 2.3.1) obstarává nástroj **dfu-util**[4].

Konfigurační vstupy		Konfigurace	Popis
BOOT1	BOOT0		
X	0	Flash paměť	Po zapnutí se načítá program v paměti Flash
0	1	Systemová paměť	Po zapnutí se načítá program ze systemové paměti (vestavěný bootloader)
1	1	Vestavěná SRAM	Po zapnutí se načítá program ve vestavěné SRAM

Tabulka 3.1: Varianty konfigurací BOOT pinů, převzato a upraveno z: [29]

3.4 Vysokoúrovňový návrh systému JaQT

Výsledný návrh představuje systém, složený z dvou hlavních součástí - bootloaderu a aplikace. Zatímco bootloader má za úkol správné spuštění aplikace a umožnění její aktualizace, účelem aplikace je poskytnout funkcionalitu definovanou požadavky kladenými na systém. Výsledný systém by měl být být schopen stabilně zpracovávat požadavky (bez zasekávání či pádů aplikace) a být jednoduchý na použití. Pro tyto účely je v rámci aplikace třeba navrhnout a implementovat ovládací rozhraní, skrze který půjde jednoduše zařízení ovládat. Zařízení by mělo být schopné komunikovat přes sériové rozhraní, kde každá sběrnice má přiděleno své vlastní rozhraní. K sériovým rozhraním se dá připojit pomoci programů jako je například **Putty** (Windows) nebo **Screen** (Linux).

Rozhraní Wiegand

Protože všechny sériové rozhraní které použitý mikrokontroler STM32F103 nabízí jsou již přiřazeny na jiné, je potřeba realizovat funkcionalitu rozhraní Wiegand jiným způsobem.

[4] http://dfu-util.sourceforge.net/

Díky tomu, že mikrokontroler umožňuje přiřazení přerušení na konkrétní vstup (v momentě, kdy procesor zaznamená aktivitu na daném vstupu, přeruší svou aktuální činnost a začne zpracovávat danou aktivitu) a zároveň poskytuje funkcionalitu časovače, je možné softwarově implementovat nové sériové rozhraní tímto způsobem a tím vytvořit nové sériové rozhraní. Komunikační vodiče D0 a D1 (podkapitola 2.7) vyžadují dva volné vstupy, ke kterým se přiřadí jeden registr přerušení a jeden časovací registr. Kontrola intervalu příchodu jednotlivých datových bitů by měla být realizována právě pomoci již zmíněného časovacího registru, a pomoci něhož bude jednotlivé bity při přerušení možno validovat. Zařízení má být schopno pouze validovat příchozí komunikaci, nikoliv zapisovat, tudíž stačí data pouze číst.

DFU Bootloader

Pro jednoduchost aktualizování firmware platformy je potřeba zařídit, aby to bylo možné pomoci integrovaného konektoru USB přes který bude probíhat veškerá komunikace se zařízením. Jelikož vestavěný bootloader vývojové desky „Bluepill“ (podkapitola 2.2) neumožňuje jednoduchou aktualizaci přes USB port (je potřeba manuálně uzemnit BOOT pin - viz pinout 2.2 nebo na úrovní registrů obejít nastavení jak je popsáno v manuálu [31]), je potřeba použít bootloader jiný. K této činnosti se výborně hodí bootloader STM32duino uživatele rogerclarkmelbourne na stránkách GitHub[4]. Tento bootloader umožňuje aktualizaci firmware pomoci implementované USB třídy DFU a potřebuje minimum změn (možnost zápisu a čtení z konkrétního bloku paměti, změna výchozího stavu vstupů), aby vykonával činnost vyžadovanou požadavky této práce. Přeložený bootloader má definovanou počáteční adresu ve Flash paměti na „*0x08000000*“ a spouští program uložený na adrese „*0x08002000*“.

Aplikace JaQT

Mezi již existujícími řešeními je projekt „pill_serial“[23], který již implementuje převodník tří sériových rozhraní do jednoho USB. Pro použití k řešení problematiky této práce je zmíněná implementace ideální, jelikož poskytuje funkční základ výsledného požadovaného systému. Pro splnění všech požadavků je potřeba odstranit přebytečné funkcionality a doplnit implementaci těch chybějících. V první řadě je třeba přidat implementaci čtvrté sériové rozhraní, které bude sloužit pro ovládání systému a čtení protokolu Wiegand. Aby nedocházelo ke konfliktům mezi zmíněnými dvěma funkcionalitami, a zároveň byla zachována jistá míra ovladatelnosti, je potřeba implementovat způsob, jak toho docílit - například pomoci nastavování příznaku aktivního čtení protokolu Wiegand. Vyžadována je pouze schopnost čtení a kontroly správně přijatého počtu datových bitů v časovém rozmezí. Dále je zapotřebí implementovat funkcionalitu sběrnice I2C, nastavit GPIO vstupy do vhodného režimu a zajistit správnou funkcionalitu sběrnic (U(S)ART, RS-232 a RS-485) nad sériovým rozhraním. Vhodným řešením implementace I2C je použít a modifikovat již existující příklad z použité knihovny libopencm3[11]. Pro zajištění správného fungování sběrnic je potřeba přenastavit jejich konfiguraci, například sjednotit a zvýšit modulační rychlost na 115000 baud jednotek.

Aby bylo možné zařízení ovládat a zasílat mu příkazy k vykonání, je potřeba implementovat kontrolní rozhraní, pomoci kterého by bylo možno takové příkazy přijímat a zpracovávat. Systém musí obsahovat kontrolu provedení zadaného příkazu a navrátit odpověď, zda-li se příkaz úspěšně provedl či nikoliv. Tato zpětnou vazbu musí být přítomna, aby bylo možné automatizovaně kontrolovat stav provedení příkazu. Tabulku příkazů 3.2, které by mělo zařízení podporovat lze vidět níže.

Název příkazu	**Parametry**	**Popis příkazu**	**Příklad použití**
help	-	Zobrazí seznam příkazů s nápovědou k použití	help
info	-	Zobrazí informace o desce (verzi firmware a datum přeložení systému)	info
port info	-	Zobrazí informace o USB rozhraních	info
clear	-	Vymaže předchozí výstupy v termínalu	clear
reset device	-	Restartuje zařízení	reset device
gpio [n] set [val]	[n] = číslo vstupu (1-14 | all) [val] = (1 - On, 0 - Off)	Nastaví vstup [n] do stavu [val]	gpio 1 set 1
gpio [n] get	n = číslo vstupu (1-14)	Navrátí stav vstupu [n] - 1 = on, 0 = off	gpio 1 get
wiegand (continuous) (size) [state]	continuous = nepřetržité čtení, volitelný size = (1-200), volitelný, výchozí: 26 state = stav (start|stop)	Spustí nebo zastaví čtení sběrnice protokolu Wiegand	wiegand continuous 20 start
i2c probe [addr]	addr = adresa v hexadecimálním tvaru	Prozkoumá přítomnost I2C zařízení na dané adrese	i2c probe 0x20
board_no set [n]	n = sériové číslo (0-255)	Nastaví sériové číslo desky na hodnotu n a v případě úspěchu restartuje zařízení	board_no set 5
board_no get	-	Navrátí sériové číslo desky	board_no get
board_no checksum	-	Navrátí kontrolní součet uloženého sériového čísla	board_no checksum

Tabulka 3.2: Seznam příkazů

Analýza výsledného návrhu

Už se tu podruhé mluví o úpravách bootloaderu, ale pořád nevím co je potřeba udělat Zatímco bootloader vyžaduje minimum úprav, aplikace je z větší části celá potřeba implementovat - převážně ovládací rozhraní a rozhraní Wiegand. Pro zajištění abstrakce nad systémem a umožnění řízení jednoduché automatizace z hosta (počítač či jiné řídící zařízení, ke kterému je JaQT připojen), je nutné implementovat sadu testů, které budou spouštěny nad řídícím sériovým rozhraním zařízení. Jelikož není potřeba žádné složité řešení, postačí implementovat testovací skript v jazyce Python s využitím modulu pySerial[13], který lze využít pro čtení komunikace na sériových rozhraních.

3.5 Postup pro testování FW/HW pomoci JaQT

Otestovaný postup pro testování zařízení pomoci systému JaQT by měl probíhat v následujícím pořadí:

1. Fyzické připojení JaQT k testovanému zařízení,
2. Připojení JaQT pomoci USB k počítači,
3. Lokalizace adresaře s testovacími skripty (adresář se musí nacházet na počítači, ke kterému je JaQT připojen) a výběr požadovaného testu dle následujících variant:
   - Již existující testovací scénář,
   - Implementace vlastního za využití dodaného testovacího modulu v jazyce Python.
4. Spuštění skriptů,
5. Průběh testů, čekání na dokončení,
6. Vyhodnocení výsledků.

Testovací skript s JaQT komunikuje pomoci sériového rozhraní a vysílá příkazy na otestování. Přijímané odpovědi o stavu jejich provedení jsou následně zpracovány a vyhodnoceny testovacím skriptem. V případě, že během testování nastane situace, kdy test nebyl úspěšně proveden, je na individuálním zvážení programátora, zda-li bude testovací skript takovou chybu vyhodnocovat jako kritickou a implementuje ukončení systému či přeskočí daný testovací případ a pokračuje dalším. O průběhu testování skript informuje uživatele na výstupu příkazové řádky, ve které byl test spuštěn.

V rámci testovací infrastruktury je efektivní, když jeden počítač řídí průběh testování na více JaQT najednou. To umožní nejen vyšší efektivitu testování ale při správné konfiguraci i dálkový přistup k zařízení, programátor tudíž nemusí být u zařízení fyzicky přítomen, aby mohl testování zahájit. Řízení testovaného zařízení může probíhat pomoci sériového rozhraní Debug integrovaného v expanzní desce.

Kapitola 4

Implementace FW a testování

V této kapitole je popsána implementace systému, provedené testy a jejich výsledky a rozšíření. Také je zde popsáno, jakým způsobem je systém nasazen v reálném prostředí. Při implementaci je dodržen návrh systému uvedený v kapitole 3, který splňuje definované požadavky. Výsledné součástí se dělí do dvou kategorií - firmware a testovací skripty. Firmware je implementován v jazyce C s použitím knihoven STM32duino a libopenCM3. Testovací skripty v jazyce Python. Struktura JaQT aplikace je rozdělena do podadresářů, které představují jednotlivé moduly systému a díky tomu je systém jednoduše rozšiřitelný o další funkcionalitu (moduly).

4.1 Systém JaQT

Tato část popisuje součásti systému, které je nutné dle návrhu implementovat nebo upravit. Zaměřuje se na převzatý DFU bootloader, JaQT aplikaci a testovací sadu skriptů v jazyce Python. Funkcionalita bootloaderu je realizována za pomoci knihovny STM32duino a aplikace JaQT využívá knihovnu libopenCM3. V případě obou součástí firmware je potřeba implementovat funkcionalitu zápisu a čtení z konkrétního bloku paměti, kde se nachází sériové číslo JaQT. Sériové číslo slouží pro identifikaci konkrétního JaQT, čehož je využito například pro rozlišení jednotlivých rozhraní. Možnost rozlišit jednotlivá zařízení umožňuje na jednom počítači provozovat více připojených JaQT najednou a v rámci testování se odkazovat na konkrétní jednotku.

4.1.1 Bootloader

Po převzetí bootloaderu byly z projektu odstraněny nepotřebné součástí, jako jsou například binární soubory určené pro jinou architekturu než je využita (projekt je již obsahuje předkompilované pro každou podporovanou variantu). K převzetí bylo přistoupeno, jelikož se jedná o efektivní řešení, které plně vyhovuje požadavkům a umožňuje jednoduchou aktualizaci bez nutnosti zásahu uživatele (popsáno v 4.2). Jak již bylo zmíněno v úvodu, je potřeba změnit obsah řetězců obsahujících informace o zařízení, připojit GPIO expander pomoci I2C a implementovat čtení z bloku paměti obsahujícího sériové číslo JaQT.

Změna řetězců v USB rozhraní Aby se po připojení do počítače zobrazily informace o zařízení, je třeba nastavit (v tomhle případě změnit) USB řetězce, které zařízení popisují. Byly změněny celkem 3 řetězce:

- Výrobce,
- Produkt,
- Sériové číslo.

Pro změnu sériového čísla je implementovaná funkce, ve které při startu zařízení probíhá čtení dat z paměti, přičemž pokud je sériové číslo desky uloženo, tak je z paměti vyčteno a zobrazeno v popisu USB rozhraní. Jestliže není sériové číslo uloženo, pak se jako výchozí hodnota nastaví číslo „0“. Velikost sériového čísla je limitována na rozmezí 0-255.

```
#define EEPROM_BOARD_NO ((u32)0x0801F7DC)
void init_board_no() {
    u8 *pmem_data = (u8 *)(EEPROM_BOARD_NO);
    u8 *pchecksum_data = (u8 *)(EEPROM_BOARD_NO) + 1;

    if (*pmem_data + *pchecksum_data == 255) {
        int mem_data = *pmem_data;

        ...
        //Nastavi hodnotu serioveho cisla
        ...

    }
}
```

Výpis 4.1: Nastavení sériového čísla

Řetězce „Produkt“ a „Výrobce“ jsou změněny na „JaQT bootloader“ a „Jiri Veverka“, výsledný popis USB rozhraní bootloaderu lze vidět na výpisu 4.2.

```
[202956.422533] usb 3-2: Product: JaQT bootloader
[202956.422536] usb 3-2: Manufacturer: Jiri Veverka
[202956.422539] usb 3-2: SerialNumber: 2
```

Výpis 4.2: Výsledek volání příkazu **dmesg** po připojení zařízení

Nastavení rozhraní I2C s pomocnými funkcemi a nastavení GPIO Jelikož je potřeba zajistit, aby všechny vstupy byly při připojení v režimu čtení (aby nedošlo k případnému poškození integrovaných obvodů připojených zařízení), a 8 vstupů je umístěno na GPIO expanderu ovladaného I2C, je potřeba rozhraní povolit a nastavit, jelikož ve výchozím stavu jsou vstupy GPIO expanderu ve stavu „HIGH“[17]. Implementace je inspirována a částečně převzata z knihovny libopencm3. Po povolení I2C komunikace již na úrovni bootloaderu, je již možné všechny vstupy nastavit do režimu pro čtení. Příklad zápisu do registru pro nastavení konkrétního vstupu do režimu pro čtení lze vidět ve výpisu 4.3

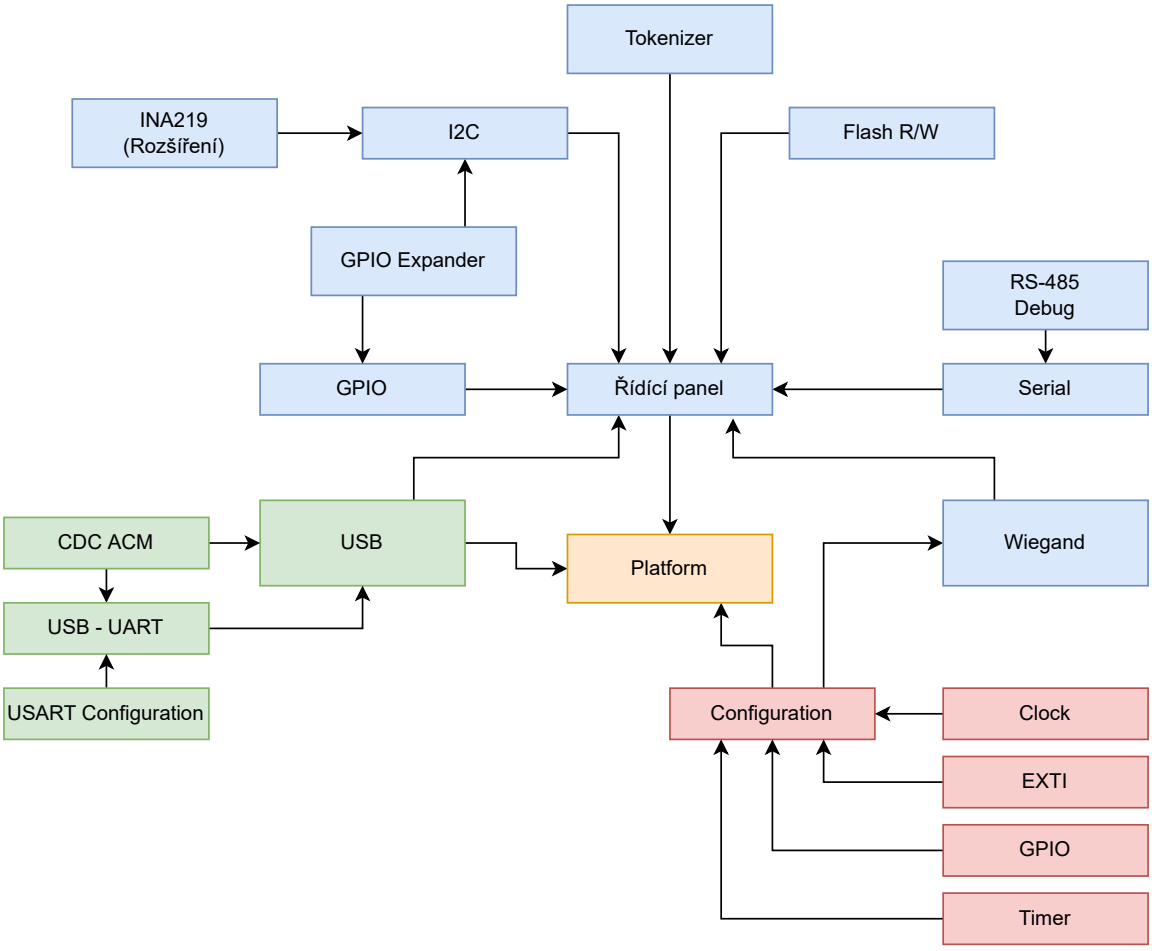
```
//SET_REG(addr,val) - Nastavi hodnotu val do registru na adrese addr
//GET_REG(addr) - Navrati hodnotu registru na adrese addr
//GPIO_CR(port, num) - Navrati adresu vstupu num na zadanem portu
//crMask(num) - Navrati bitovou masku kontrolniho registru pro vstup num
//CR_SHITF(num) - Navrati bit kontrolniho registru
//CR_OUTPUT_PP - Vystupni kontrolni registr

SET_REG(GPIO_CR(GPIOB,3),(GET_REG(GPIO_CR(GPIOB,3)) & crMask(3)) \\
              | CR_OUTPUT_PP << CR_SHITF(3));
```

Výpis 4.3: Nastavení čtecího režimu na vstupu B3

4.1.2 Aplikace

Při implementaci aplikace se částečně vycházelo z převzatého projektu, který byl již zmíněn v kapitole 3.4. Jelikož daný projekt již implementuje tři sériové rozhraní, je třeba přidat jedno poslední - virtuální (všechny rozhraní jsou znázorněny na obrázku 2.6). Následně je potřeba zprovoznit rozhraní I2C pro komunikaci s GPIO expanderem, implementovat ovládání GPIO vstupů, zápis a čtení vyhrazeného bloku paměti, nastavení vstupů a testovaní dat přijatých rozhraním protokolu Wiegand a způsob zpracování příkazů včetně tokenizace přijímaných zpráv (tabulka 3.2). Také je potřeba změnit definovanou počáteční adresu Flash paměti z „*0x08000000*“ na „*0x08002000*“ aby nedocházelo ke kolizi s bootloaderem a byla zajištěna jeho správná funkcionalita.

V práci je využita knihovna libopenCM3 a jí nabízené funkcionality. Jak již bylo zmíněno výše, aplikace je pro umožnění jednoduché rozšiřitelnosti rozčleněná do modulů - graf závislostí a členění do modulů lze vidět na obrázku 4.1.



Obrázek 4.1: Graf závislostí a rozčlenění do modulů. Modře - moduly řídícího panelu, zeleně - moduly USB, červeně - konfigurační moduly, oranžově - jádro systému

Sériové rozhraní Nastavení sériových rozhraní je fixní. Pro změnu parametrů některého z rozhraní je třeba upravit následující řádky kódu (výpis 4.4):

```
void setUSART(int baudrate, int USBUSART)
{
  usart_disable(USBUSART);

  //Nastavi modulacni rychlost na zadane baud jednotky
  usart_set_baudrate(USBUSART, baudrate);
  usart_set_databits(USBUSART, 8); //Nastavi pocet datovych bitu
  usart_set_stopbits(USBUSART, USART_STOPBITS_1); //Nastavi pocet stopbitu
  usart_set_mode(USBUSART, USART_MODE_TX_RX);
  usart_set_parity(USBUSART, USART_PARITY_NONE); //Nastavi paritu
  usart_set_flow_control(USBUSART, USART_FLOWCONTROL_NONE);

  usart_enable(USBUSART);
}
```

Výpis 4.4: Nastavení sériových rozhraní.

Ovládácí panel a jeho příkazy Ovládací rozhraní zpracovává komunikaci odehrávající se na virtuálním rozhraní. K zprávám přicházejícím na toto rozhraní se přistupuje jako k příkazům a při každé probíhá zpracování znaků. Jestliže je příkaz nalezen, provede se odpovídající akce a systém vygeneruje patřičnou odpověď na výstupu. Součástí odpovědi je i prefix indikující stav provedení příkazu - [**OK**] jestli akce byla vykonána úspěšně a [**ERROR**] pokud došlo k chybě. V případě, že příkaz není nalezen nebo není zadán správně, systém vypíše hlášení o neexistujícím příkazu a vyzve uživatele aby zobrazil nápovědu pomoci příkazu „help“.

Příchozí komunikace je ukládána znak po znaku do paměti a až po příjmutí ANSI escape sekvence pro klávesu Enter je výsledná zpráva předána k zpracování. Pro zpracování příchozích zpráv se využívá tokenizace - příchozí zpráva je rozdělená na jednotlivé tokeny podle mezer a ty jsou následně postupně zpracovávány.

Ovládací rozhraní podporuje klávesu backspace pro případ, že se uživatel při zadávání přepsal a potřebuje příkaz opravit. Tato klávesa odesílá ANSI escape sekvenci, kterou je možné během příchozí zprávy odchytit a následně patřičně zpracovat. Další escape sekvence podporovány nejsou.

Nastavení sériového rozhraní I2C Konfigurace rozhraní I2C je implementována podle příkladu z knihovny libopenCM3[11]. Podobným způsobem je implementována i funkce pro vyhledávání zařízení na požadované adrese (probe). Rozhraní je vyžadováno pro komunikaci s GPIO expanderem, který rozšiřuje počet vstupů o dalších 8. Výpis 4.5 zobrazuje implementaci funkce probe, jejíž příkaz v ovládacím panelu je tvaru: *i2c probe [adresa]*.

```
uint8_t i2c_probe(uint8_t addr)
{
	i2c_send_start(I2C1);

/* Zacatek prevzate casti */

	/* Wait for the end of the start condition,
	   master mode selected, and BUSY bit set */
	while ( !( (I2C_SR1(I2C1) & I2C_SR1_SB)
	&& (I2C_SR2(I2C1) & I2C_SR2_MSL)
	&& (I2C_SR2(I2C1) & I2C_SR2_BUSY) ));

/* Konec prevzate casti */

	i2c_send_7bit_address(I2C1, addr, I2C_WRITE);

	uint8_t probecounter = 0;
	while(!(I2C_SR1(I2C1) & I2C_SR1_ADDR) && probecounter != 255) {
		probecounter++;
	}

	(void)I2C_SR2(I2C1);

	i2c_send_stop(I2C1);

	return (probecounter == 255) ? 0 : 1;
}
```

Výpis 4.5: Implementace funkce I2C probe

Ovládání GPIO Při spuštění zařízení jsou ve výchozím stavu všechny vstupy nastaveny do režimu čtení. Jak již bylo zmíněno, je to z důvodu, aby nedošlo k případnému poškození interních obvodů připojeného zařízení. K následnému ovládání vstupů jsou dostupné dva příkazy:

- *gpio [num] get* - Zjistí aktuální logickou hodnotu vstupu číslo [num]. Vstup je uveden do režimu pro čtení,
- *gpio [num] set [state]* - Nastaví logickou hodnotu specifikovanou parametrem [state] (1 nebo 0) na vstupu číslo [num]. Vstup je uveden do režimu pro zápis.

Vstupy které jsou vyvedené z vývojové desky přímo se ovládají pomocí interních funkcí knihovny libopenCM3, přičemž vstupy které jsou součástí GPIO expanderu jsou ovládány pomoci funkcí rozhraní I2C.

Zápis a čtení z Flash paměti Jak již bylo zmíněno v předchozích kapitolách, systém umožňuje identifikaci konkrétních JaQT zařízení pomoci sériového čísla. Sériové číslo lze nastavit pouze z ovládacího panelu aplikace a to příkazem *serial_no set [1-255]*. Pro tyto potřeby jsou implementovány 3 funkce - dvě pro čtení (sériového čísla a kontrolního součtu) a jedna pro zápis. Zatímco funkce pro čtení (výpis 4.6 a 4.8) se svou funkcionalitou nijak neliší od té popsané v bootloaderu, funkce pro zápis (výpis 4.7 je odlišná. Kontrolní součet se počítá vždy při nastavení nového sériového čísla a jeho hodnota je rovná zbytku po odečtení sériového čísla od hodnoty 255 (*checksum = 255 - serial number*).

```
#define FLASH_EEPROM_ADDRESS  ((uint32_t)0x0801F7DC)

uint8_t* flash_read_board_no() {
    return (uint8_t *)FLASH_EEPROM_ADDRESS;
}
```

Výpis 4.6: Navrácení sériového čísla

```
#define FLASH_EEPROM_ADDRESS  ((uint32_t)0x0801F7DC)

uint8_t flash_write_board_no(uint16_t halfword) {

        flash_unlock();

        flash_erase_page(FLASH_EEPROM_ADDRESS);

        flash_program_half_word(FLASH_EEPROM_ADDRESS, halfword);

    /* verify the write */
    if (*(volatile uint16_t *)FLASH_EEPROM_ADDRESS != halfword) {
        return 1;
    }

    return 0;
}
```

Výpis 4.7: Zapsaní hodnoty do Flash paměti

```
#define FLASH_EEPROM_ADDRESS  ((uint32_t)0x0801F7DC)

uint8_t* flash_read_checksum() {
    return (uint8_t *)FLASH_EEPROM_ADDRESS + 1;
}
```

Výpis 4.8: Navrácení kontrolního součtu

Rozhraní protokolu Wiegand Rozhraní pro příjem komunikace je realizováno pomoci dvou vyčleněných GPIO vstupů nakonfigurovaných do čtecího režimu („input“), na které jsou pomoci funkcí knihovny libopenCM3 napamovány přerušení (registry *EXTI8* a *EXTI9*), které se generuje vždy, když dojde k detekci změny logické úrovně na konkrétním vstupu (výpis 4.9).

```
void extiWiegandSETUP(uint16_t *exti8_direction, uint16_t *exti9_direction)
{
	nvic_enable_irq(NVIC_EXTI9_5_IRQ);

	exti_select_source(EXTI8, GPIOB);
	exti_select_source(EXTI9, GPIOB);
	*exti8_direction = 0;
	*exti9_direction = 0;
	exti_set_trigger(EXTI8, EXTI_TRIGGER_FALLING);
	exti_set_trigger(EXTI9, EXTI_TRIGGER_FALLING);
	exti_enable_request(EXTI8);
	exti_enable_request(EXTI9);
}
```

Výpis 4.9: Nastavení přerušení

Pokud je vygenerováno přerušení, je aktivován časovač *TIM1*, jehož časová perioda je nastavena na 2010μs (maximální délka periody mezi jednotlivými bity + 10μs). Při každém dalším přerušení na rozhraní Wiegand je kontrolováno, zda-li je detekovaná náběžná hrana signálu opačná, než poslední přijatá a nachází se na stejném vodiči, a pokud ano, patřičný bit je uložen do zásobníku, časovač je vynulován a čeká na příjem dalších dat. Jestliže dojde k vypršení časové periody (zpoždění na lince nebo nižší velikost datagramu), detekci neočekávané náběžné hrany nebo příjmu signálu na opačném vodiči než je očekáván, dojde k okamžitému ukončení detekce přerušení, odečet časovače je pozastaven a provede se okamžité vyhodnocení přijatých dat. V případě, že během přenosu nedojde k chybě, po přijetí všech datových bitů dojde k vyhodnocení přijatého datagramu. V obou případech je výsledek vypsán na sériové rozhraní ovládacího panelu.

4.2 Nasazení výsledného zařízení s firmware JaQT v provozu

V reálném prostředí je systém nasazen v počtu několika zařízení JaQT připojenych k jednomu počítači RaspberryPi[1] a testovanému zařízení. RaspberryPi následně slouží jako řídící jednotka, která nad konkrétními JaQT spouští automatizované testy. Počítač je připojený k lokální síti, tudíž programátor, který právě provádí testování může řídit celý proces na dálku.

Využití SWD pro nahrání bootloaderu a aplikace

popis jak se nahrává bootloader, zmínit program který využívám k tomu, obrázek zapojení

Pro nahrání programu přes rozhraní SWD je potřeba připojit programátor ST-LINK k vývojové desce „Bluepill“. Vstupy rozhraní SWD jsou z desky vyvedeny zvlášť, tudíž není potřeba celý komplet demontovat. Po připojení programátoru k počítači lze využít zmíněného nástroje od firmy ST, který přes specifikované rozhraní nahraje program do paměti na určenou adresu. Kromě nástroje od firmy ST je možné využít program OpenOCD s konfiguračním *openocd.cfg* souborem popsaným ve výpisu 4.10.

```
source [find interface/stlink-v2.cfg]

transport select "hla_swd"

set WORKAREASIZE 0x5000

set CHIPNAME STM32F103C8Tx

set ENABLE_LOW_POWER 1

set STOP_WATCHDOG 1

set CLOCK_FREQ 100

reset_config none

source [find target/stm32f1x.cfg]
```

Výpis 4.10: Open OCD konfigurační soubor

Aktualizace firmware

V případě, že JaQT zařízení již obsahuje nahraný bootloader umožňující aktualizaci pomoci USB rozhraní DFU, je možné použít nástroj *dfu-util*. Jelikož je rozhraní DFU aktivní přibližně první tři vteřiny po startu zařízení, je potřeba zahájit aktualizaci v daném časovém okně. Pro případ, že je potřeba aktualizovat zařízení na dálku, je implementován příkaz *reset device*, který vyvolá restart zařízení a umožní provedení operace nad DFU rozhraním.

[1] https://www.raspberrypi.org/

4.3 Výstupy testování rozhraní Wiegand a jejich vyhodnocení

Při testování rozhraní Wiegand se počítá s výchozí hodnotou délky datagramu 26 a časovým intervalem 2010μs, který se skládá z času pro příjem hodnoty (50μs), času pro pauzu mezi příjmem jednotlivých bitů (1950μ) a časové tolerance 10μs.

Název testu: Test správného přenosu dat - Test Wiegand #1.

Popis testu: Testované zařízení odešle data na rozhraní Wiegand. Odeslaná data jsou validní a očekávaný výsledek testu je kompletní přenos všech dat ve správném časovém intervalu.

Parametry: Datagram: 10111001100011010101101011
Délka datagramu: 26
Očekávaná délka: 26

Výsledek: Přijaté data: 10111001100011010101101011
Status: OK

Vyhodnocení: Data byla přijatá všechny a ve správném časovém intervalu. Test skončil úspěchem. Výpis z terminálu 4.11 zobrazuje průběh testu.

```
wiegand start
[OK] Wiegand reading has started, please connect the device
*****************************
[OK]
Wiegand done

Data: 10111001100011010101101011
Data times: 51 50 50 50 50 50 51 50 49 50 50 50
            50 50 50 50 50 50 50 50 49 52 50 50 50 50
Pause times: 1952 1951 1949 1950 1950 1952 1950 1950 1950
             1951 1951 1953 1950 1951 1950 1951 1950 1949
             1950 1950 1952 1950 1949 1950 1950
*****************************
```

Výpis 4.11: Výpis z terminálu, test Wiegand #1

Název testu: Test nesprávného přenosu dat - Test Wiegand #2.

Popis testu: Testované zařízení odešle nekompletní datagram na rozhraní Wiegand. Odeslaná data jsou odeslána ve správných časových intervalech, avšak chybí poslední bit. Očekávaný výsledek testu je, že systém detekuje chybu a oznámí ji.

Parametry: Datagram: 1011100110001101010110101
Délka datagramu: 25
Očekávaná délka: 26

Výsledek: Přijaté data: 10111001100011010101101101-
Status: ERROR

Vyhodnocení: Všechna odeslaná data byla úspěšně přijata ve správném časovém intervalu a byl detekován chybějící bit. Test skončil úspěchem, výpis z terminálu lze vidět níže (výpis 4.12). Chybějící bit je znázorněn šipkou a písmenem X („->X“).

```
*****************************
wiegand start
[OK] Wiegand reading has started, please connect the device
[ERROR]
Packet number: 25 - Timeout while waiting for next packet (Missing packets)

Data: 10111001100011010101101101-
Data times: 51 49 50 51 51 50 50 50 50 50 49 50
           50 50 50 50 50 50 50 50 50 50 50 50 50
Pause times: 1950 1951 1952 1951 1950 1950 1950 1951
           1949 1950 1950 1950 1951 1949 1950 1949
           1950 1951 1951 1949 1949 1950 1950 1950 ->X
*****************************
```

Výpis 4.12: Výpis z terminálu, test Wiegand #2

Název testu: Test nesprávného přenosu dat - Test Wiegand #3.

Popis testu: Testované zařízení odešle kompletní datagram na rozhraní Wiegand. Odeslaná data jsou odeslána v nevalidním časovém intervalu. Očekávaný výsledek testu je, že systém detekuje chybu a oznámí ji.

Parametry: Datagram: 10111001100011010101101011
Délka datagramu: 26
Očekávaná délka: 26

Výsledek: Přijaté data: 10111001100011010101101011
Status: ERROR

Vyhodnocení: Všechna odeslaná data byla úspěšně přijata ve správném časovém intervalu. Test skončil úspěchem, výpis z terminálu lze vidět níže (výpis 4.13). Chybějící bit je znázorněn šipkou a časovou hodnotou („->“).

```
****************************
wiegand start
[OK] Wiegand reading has started, please connect the device
[ERROR]
Packet number: 1 - Timeout while waiting for next packet (Timeout)

Data: 10111001100011010101101011
Data times: 51 51 52 51 50 50 50 50 52 49 50
            50 51 50 50 49 50 50 50 50 50 50
            49 50 50 50
Pause times: ->1974 1951 1951 1953 1949 1950 1950
            1951 1949 1949 1950 1951 1950 1949
            1950 1950 1951 1953 1951 1949 1949
            1950 1952 1951 1950
****************************
```

Výpis 4.13: Výpis z terminálu, test Wiegand #3

4.4 Implementovaná rozšíření

Mezi implementovanými rozšířeními je aktuálně pouze I2C modul pro měření proudu a napětí pomocí obvodu INA219[35]. Při pokusu o čtení dat ze senzoru systém prvně pomoci kontroly dostupných I2C adres zjistí, jestli je senzor připojen. Tento proces se nazývá „I2C probe“, a je-li senzor připojen a funkční, pak navrátí požadované hodnoty. V opačném připojení systém vypíše uživateli hlášení do příkazové řádky, že senzor nebyl nalezen. V řídícím panelu byly implementovány příkazy popsané v tabulce 4.1.

Příkaz	Popis
get ina219	Navrátí data z INA219 senzoru (je-li připojen)
set ina219 [address]	Nastaví I2C adresu senzoru INA219
init ina219 32v 1a	Inicializuje INA219 pro maximální napětí 32V a proud 1A
init ina219 16v	Inicializuje INA219 pro maximální napětí 16V a proud 400mA

Tabulka 4.1: Příkazy pro INA219.

Kapitola 5

Závěr

Výsledný systém je plně funkční a využívá se již pro testování při vývoji. Aplikace je modulární a tudíž jednoduše rozšířitelná o nové funkcionality.

JaQT oproti podobnému testovacímu systému postavenému na platformě Beaglebone Black[14] disponuje celou řadou výhod. První a nejzřetelnější výhodou je jeho jednoduchost - oproti svému složitějšímu protějšku nevyžaduje pro správné fungování operační systém. Implementovaný firmware totiž pracuje přímo s registry mikrokontroleru bez nutnosti složitější abstrakce, kterou použití operačního systému zpravidla přináší. Další výhodou je zavedení modulů - systém je jednoduše rozšířitelný o další funkcionality. Neposlední výhodou je, že oproti již zmíněné porovnávané platformě, je pořizovací cena vývojové desky „Bluepill“ a použité expanzní desky (včetně součástek) výrazně nižší.

Nepotvrzenou výhodou by měla být, i díky nízkoúrovňové podstatě implementovaného systému, rychlost zpracovávání příkazů a časově omezených požadavků.

Nevýhodou JaQT je, že kvůli omezení použité architektury není možno vykonávat více činností najednou (paralelně), a pro zajištění jisté úrovně paralelizace je tedy zapotřebí zajistit přepínání akci pomoci různých úrovní priorit přerušení.

Další problém nastává při čtení GPIO - kvůli problémům v návrhu expanzní desky použitý měnič logických úrovní neumožňuje správné čtení některých logických stavů, je tedy nutné manuálně umístit externí pull-down resistor na vstupy expanzní desky pro zajištění správné funkcionality. V opačném případě nastává problém při pokusu o čtení nízké úrovně za klidového stavu právě kvůli použitému měniči logických úrovní - hodnota je nezjistitelná.

Po vyhodnocení výstupů z hotového systému lze říci, že bylo dosaženo všech požadavků, avšak lze najít i případy, kde by šel rozšířit. Mezi dané případy se řadí:

- Integrace výsledného produktu do Labgridu[1], - jedná se o nástroj a knihovnu v jazyce Python, sloužící k automatizaci testování vestavěných zařízení.
- Integrovaná kontrola bitové parity při testování rozhraní Wiegand. Tuto problematiku aktuálně řeší programátor individuálně dle svého uvážení.
- Implementace rozhraní SPI.
- Testování komunikace na rozhraní I2C - aktuálně se používá pouze pro práci s moduly.
- Rozšíření nedostatečného paměťového prostoru pomoci externího úložiště typu EEPROM připojeného pomoci sběrnice I2C.

[1] https://github.com/labgrid-project/labgrid

- LED displej obsahující informace o aktuálním stavu zařízení, komunikující přes sběrnici I2C.
- Změna způsobu aktualizace firmware - využití interního bootloaderu namísto aktuálně použitého.

Dále díky existenci malé prototypovací plochy na plošném spoji označené jako „Protoboard“ lze uvažovat například o implementaci podpory pro spínání relé, majícího za úkol spínání vysokého napětí u testovaného zařízení. Co se zlepšení týče, určitě by bylo vhodné mírně upravit návrh expanzní desky a eliminovat nutnost externě připojeného „čtecího“ pull-down rezistoru. Zároveň by stálo za úvahu využít dalšího GPIO expanderu pro uvolnění některých z aktuálně zabraných GPIO vstupů, jež by se následně daly využít pro rozšíření testovacích kapacit (například rozšíření testovaní o sběrnici CAN).

Také by stálo za zvážení restrukturalizovat parser příkazů z řídícího panelu, aby umožňoval jednodušší a intuitivnější integraci nových modulů.

Dále by bylo vhodné integrovat DFU funkcionalitu přímo do JaQT aplikace, aby se eliminovala závislost na externím kódu. Samotná produkce testovací platformy - hardware + firmware je jednoduchá a podařilo se splnit všechny deklarované požadavky. Kromě samotné JaQT aplikace je implementován i základ modulu pro automatizované testování v jazyce Python3, který je jednoduše rozšířitelný a aktuálně se využívá pouze pro testování správného fungování samotné testovací platformy.

Literatura

[1] ARM Ltd.. *Arm Debug Interface ArchitectureSpecification ADIv6.0*. 2017. Document number: ID030122. Dostupné z: https://documentation-service.arm.com/static/62221ef4e6f58973271ebc1d.

[2] Axelson, J. *USB Complete: everything you need to develop custom USB peripherals*. 2. vyd. Lakeview Research, 2004. ISBN 0-9650819-5-8.

[3] Barnes, D. G. *USB connectors: an optimistic journey from USB 1.0 to USB 3.1*. Listopad 2021. Dostupné z: https://it-talk.org/usb-connectors-an-optimistic-journey-from-usb-1-0-to-usb-3-1/.

[4] Clark, R. *STM32duino-bootloader*. Melbourne: [b.n.], květen 2022. Dostupné z: https://github.com/rogerclarkmelbourne/STM32duino-bootloader.

[5] Gook, M. *Hardwarová rozhraní: průvodce programátora*. 1. vyd. Brno: Computer Press, 2006. ISBN 80-251-1019-2.

[6] Gravekamp, T. *Description of STM32F103C8T6 - Blue Pill*. Dostupné z: https://stm32-base.org/boards/STM32F103C8T6-Blue-Pill.html.

[7] Hankovec, D. *Popis protokolu Wiegand a řešení jeho čtení procesorem*. Prosinec 2009. Dostupné z: http://www.dhservis.cz/dalsi_1/wiegand.htm.

[8] Hans-Peter Messmer. *Velká kniha hardware: [architektura, funkce, programování]*. 1. vyd. 2005. ISBN 80-251-0416-8.

[9] Horowitz, P. *The art of electronics*. 3. vyd. Cambridge University Press, 2015. ISBN 978-0-521-80926-9.

[10] HW server s.r.o.. *USB 2.0 - Úvodní článek seriálu o sběrnici USB*. únor 2005. Dostupné z: https://vyvoj.hw.cz/navrh-obvodu/rozhrani/rs-485-rs-422/usb-20-dil-1.html.

[11] libopencm3. *Libopencm3 - Simple example projects showing how to use libopencm3*. libopencm3, květen 2022. Dostupné z: https://github.com/libopencm3/libopencm3-examples.

[12] libopencm3. *Libopencm3 project - Open source ARM Cortex-M microcontroller library*. libopencm3, květen 2022. Original-date: 2012-04-18T16:51:38Z. Dostupné z: https://github.com/libopencm3/libopencm3.

[13] Liechti, C. *pySerial - Python serial port access library*. Květen 2022. Dostupné z: https://github.com/pyserial/pyserial.

[14] Macháček, J. *Zařízení pro automatizované testování hardwaru.* Brno, 2016. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací.

[15] Maxim Integrated Products, Inc. *Half-Duplex RS-485-/RS-422- Compatible Transceiver with AutoDirection Control.* Datasheet 19-0740; Rev 1; 2/15. 2015. Dostupné z: https://datasheets.maximintegrated.com/en/ds/MAX13487E-MAX13488E.pdf.

[16] Maxim Integrated Products, Inc. *MAX3222/MAX3232/MAX3237/MAX3241 - 3.0V to 5.5V, Low-Power, up to 1Mbps, True RS-232 Transceivers.* Datasheet 19-0273; Rev 10; 5/19. 2019. Dostupné z: https://pdfserv.maximintegrated.com/en/ds/MAX3222-MAX3241.pdf.

[17] NXP Semiconductors. *PCF8574; PCF8574A Remote 8-bit I/O expander for I2C-bus with interrupt.* Datasheet PCF8574_PCF8574A. Květen 2013. Dostupné z: https://www.nxp.com/docs/en/data-sheet/PCF8574_PCF8574A.pdf.

[18] Pavlik, R. *JaQT2 - Functional testing plaform, hardware design project.* Květen 2022. GitHub repository. Dostupné z: https://github.com/RadimPavlik/JaQT2.

[19] Peška, R. *Wiegand – od karty po sběrnici.* Prosinec 2017. Dostupné z: https://automatizace.hw.cz/wiegand-od-karty-po-sbernici.html.

[20] Rath, D. *OpenOCD - Open On-Chip Debugger, Solution for Embedded Target Systems.* červenec 2005. Dostupné z: https://openocd.org/.

[21] Reblag. *Generic STM32 boards – Board Pinout diagram.* Květen 2022. Dostupné z: http://reblag.dk/stm32/.

[22] satoshinm. *Triple USB-to-serial adapter using STM32 blue pill (pill_serial).* Prosinec 2017. Dostupné z: https://satoshinm.github.io/blog/171223_stm32serial_triple_usb-to-serial_adapter_using_stm32_blue_pill.html.

[23] satoshinm. *Pill_serial: USB-to-serial adapter.* Březen 2022. Original-date: 2017-12-24. Dostupné z: https://github.com/satoshinm/pill_serial.

[24] Staněk, J. a Řehák, J. *RS 485 & 422 | Vývoj.HW.cz.* Leden 1998. Dostupné z: https://vyvoj.hw.cz/teorie-a-praxe/dokumentace/rs-485-422.html.

[25] stlink-org. *Open source version of the STMicroelectronics STLINK Tools.* stlink-org, květen 2022. Original-date: 2011-01-14T09:19:12Z. Dostupné z: https://github.com/stlink-org/stlink.

[26] STMicroelectronics. *Getting started with STM32F10xxx hardware development.* Application Note AN2586; Rev 7. STMicroelectronics, listopad 2011. 28 s. Dostupné z: https://www.st.com/resource/en/application_note/cd00164185-getting-started-with-stm32f10xxx-hardware-development-stmicroelectronics.pdf.

[27] STMicroelectronics. *STM32F103x4, STM32F103x6 - Low-density performance line, ARM-based 32-bit MCU.* Datasheet 15060; Rev 7. červen 2015. Dostupné z: https://www.st.com/resource/en/datasheet/stm32f103c6.pdf.

[28] STMicroelectronics. *STM32CubeProgrammer software for all STM32*. Data brief DB3420; Rev 4. únor 2019. Dostupné z: https://www.st.com/en/development-tools/stm32cubeprog.html.

[29] STMicroelectronics. *Description of STM32F1 HAL and low-layer drivers*. User manual UM1850; Rev 3. STMicroelectronics, únor 2020. Dostupné z: https://www.st.com/resource/en/user_manual/dm00154093-description-of-stm32f1-hal-and-lowlayer-drivers-stmicroelectronics.pdf.

[30] STMicroelectronics. *ST-LINK/V2 - ST-LINK/V2 in-circuit debugger/programmer for STM8 and STM32*. Data brief DB1275; Rev 6. STMicroelectronics, listopad 2020. Dostupné z: https://www.st.com/en/development-tools/st-link-v2.html.

[31] STMicroelectronics. *STM32 microcontroller system memory boot mode*. Application Note AN2606; Rev 54. STMicroelectronics, 2022. Dostupné z: https://www.st.com/resource/en/application_note/cd00167594-stm32-microcontroller-system-memory-boot-mode-stmicroelectronics.pdf.

[32] STMicroelectronics. *STM32CubeMX - STM32Cube initialization code generator*. Data brief DB2163; Rev 18. STMicroelectronics, únor 2022. Dostupné z: https://www.st.com/resource/en/data_brief/stm32cubemx.pdf.

[33] STMicroelectronics. *STM32F103x8, STM32F103xB, ARM based 32-bit MCU*. Datasheet DS5319; Rev 18. STMicroelectronics, březen 2022. Dostupné z: https://www.st.com/resource/en/datasheet/stm32f103c8.pdf.

[34] Stollon, N. *On-Chip Instrumentation: Design and Debug for Systems on Chip*. Springer Science+Business Media, LLC, 2011. ISBN 978-1-4419-7562-1.

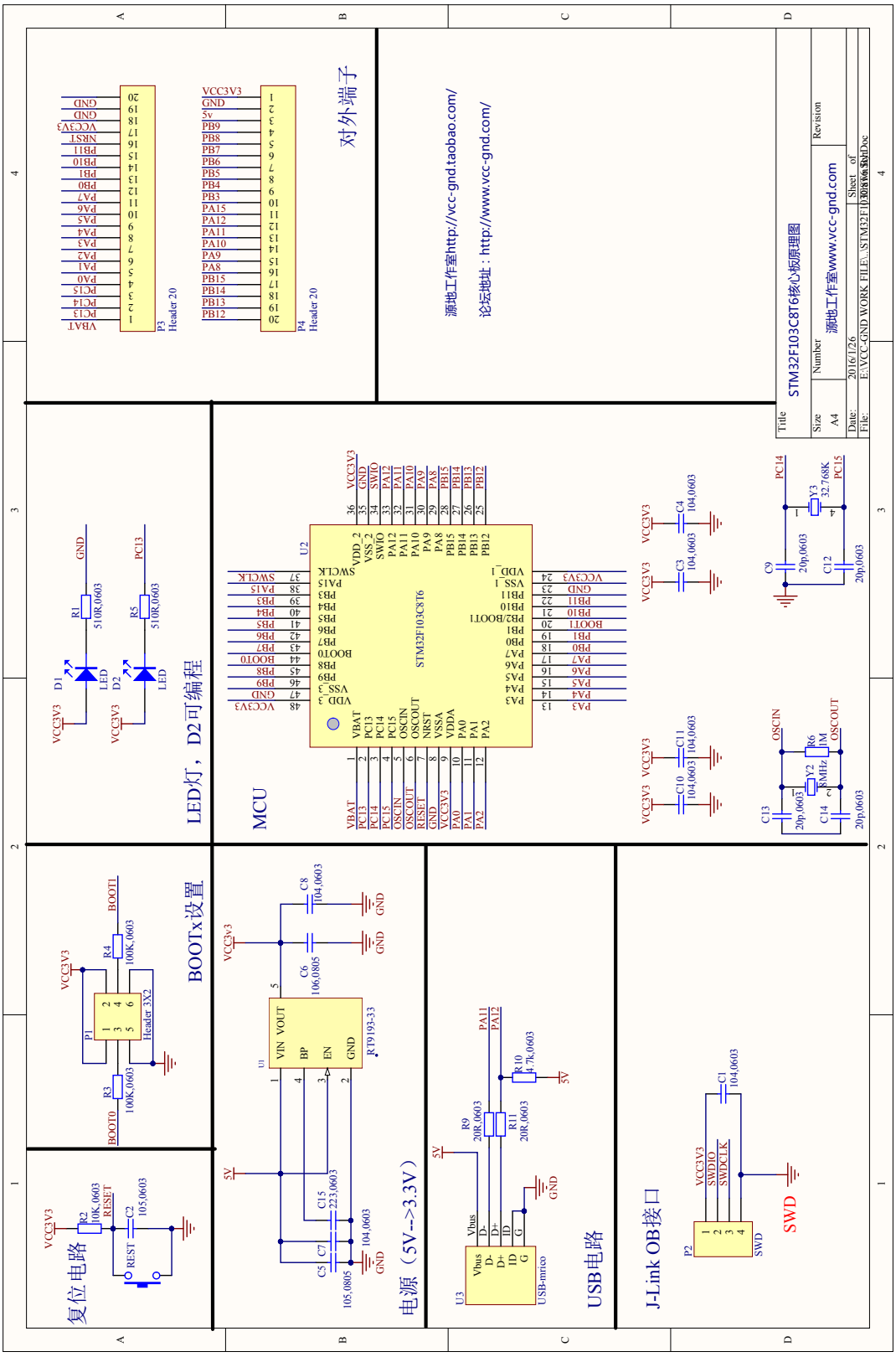
[35] Texas Instruments. *INA219 Zero-Drift, Bidirectional Current/Power Monitor with I2C Interface*. Datasheet SBOS448G. Prosinec 2015. Dostupné z: https://www.ti.com/lit/ds/symlink/ina219.pdf.

[36] USB Foundation. *Universal Serial Bus Device Class Specification for Device Firmware Upgrade*. 2004. Dostupné z: https://www.usb.org/sites/default/files/DFU_1.1.pdf.

[37] XMOS Ltd.. *USB CDC Class as Virtual Serial Port*. Application Note AN00124. 2016. Dostupné z: https://www.xmos.ai/download/AN00124:-USB-CDC-Class-as-Virtual-Serial-Port(2.0.2rc1).pdf.

Příloha A

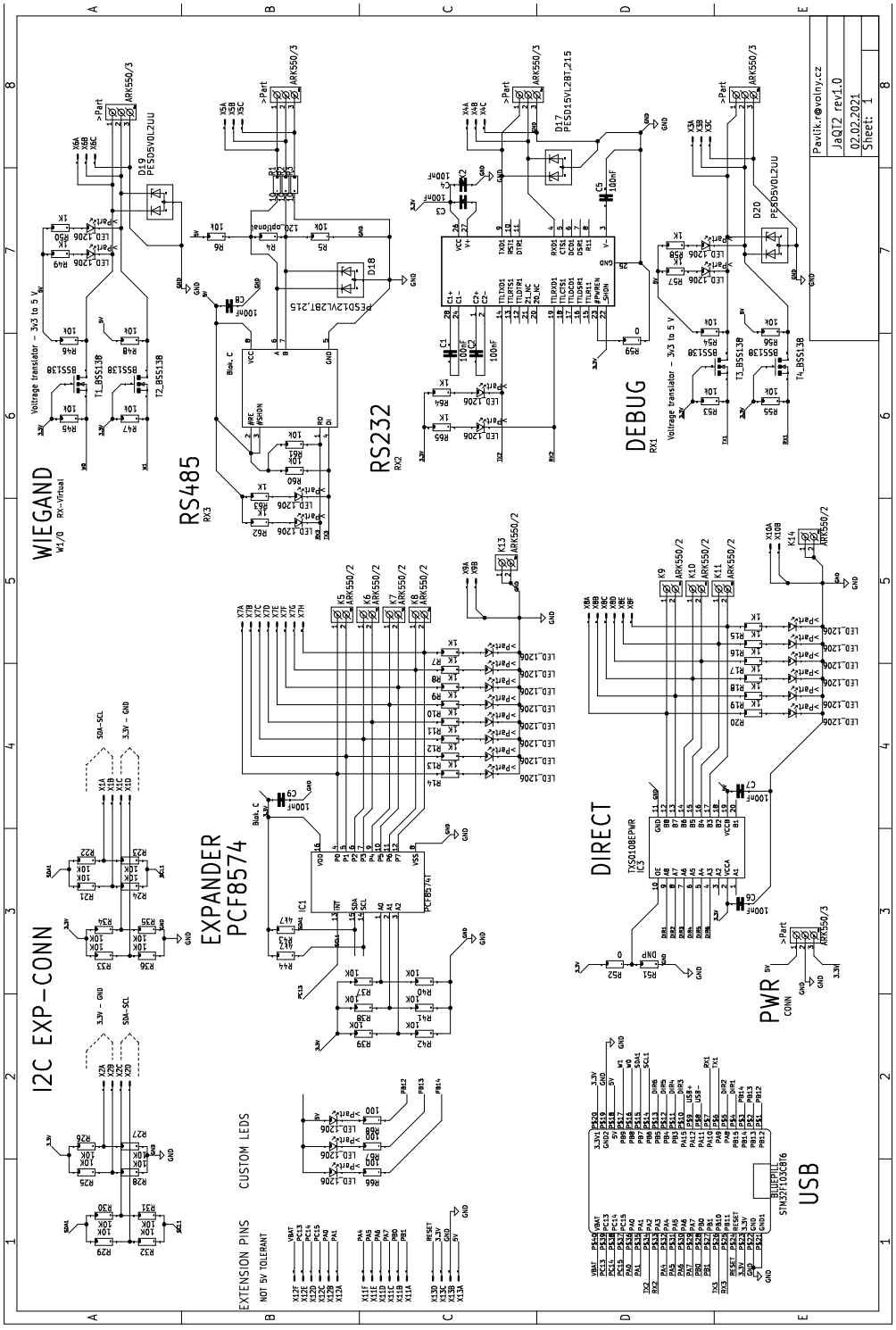
Schéma vývojové desky „Bluepill“



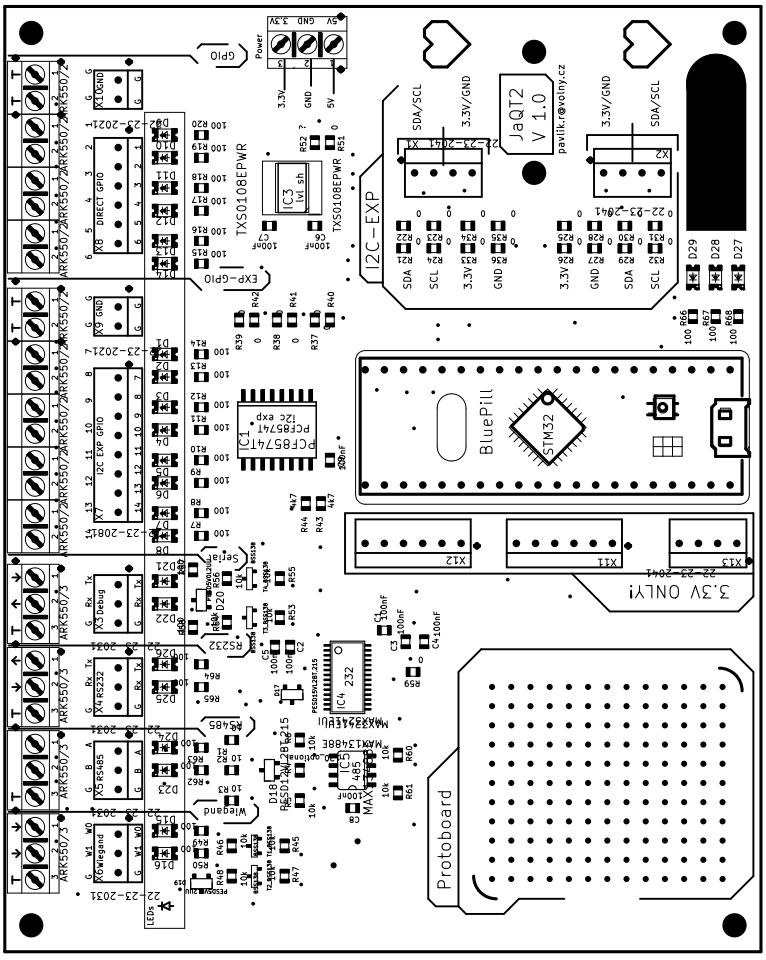
Obrázek A.1: Schéma vývojové desky „Bluepill“, převzato z: [6].

Příloha B

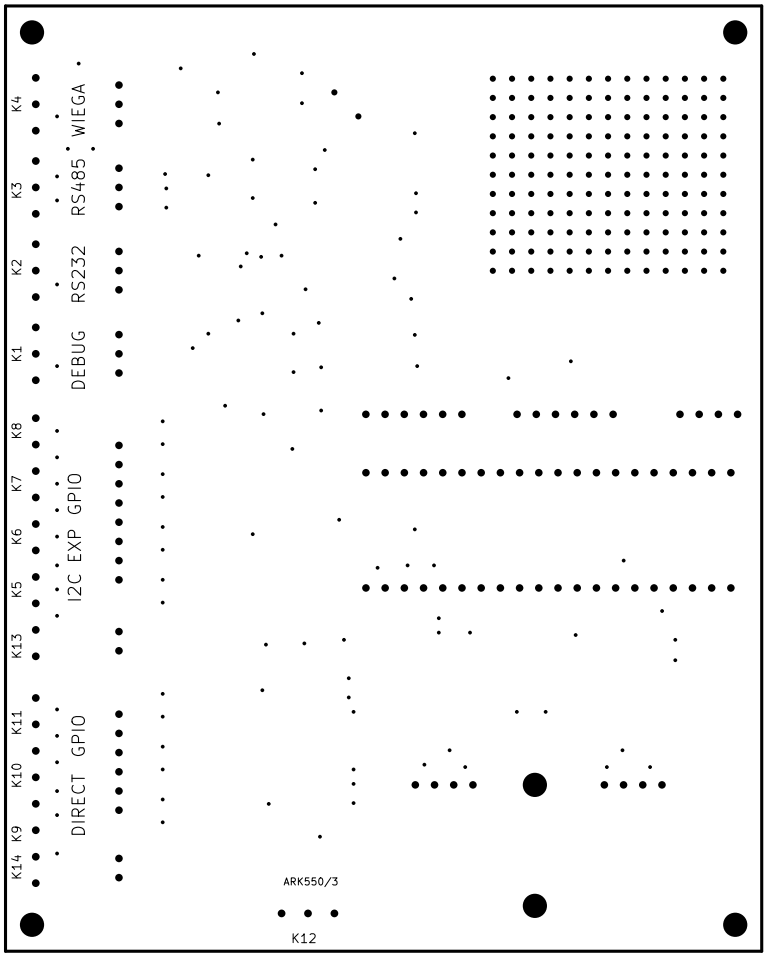
Schéma a osazovací plány expanzní desky



Obrázek B.1: Schéma expanzní desky, převzato z: [18].



Obrázek B.2: Osazovací plán expanzní desky, přední strana, převzato z: [18]



Obrázek B.3: Osazovací plán expanzní desky, zadní strana, převzato z: [18]