VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

PLATFORMA PRO AUTOMATICKÉ TESTOVÁNÍ VESTAVĚNÝCH ZAŘÍZENÍ

PLATFORM FOR AUTOMATIC TESTING OF EMBEDDED DEVICES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **OTO DUŠEK**
AUTHOR

VEDOUCÍ PRÁCE **Doc. Ing. ZDENĚK VAŠÍČEK, Ph.D.**
SUPERVISOR

BRNO 2018

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Dušek Oto**
Obor: Informační technologie
Téma: **Platforma pro automatické testování vestavěných zařízení**
A platform for Automatic Testing of Embedded Devices
Kategorie: Vestavěné systémy

Pokyny:

1. Seznamte se s problematikou automatického testování funkčnosti moderních vestavěných zařízení jako jsou tiskárny, scannery, kopírovací stroje apod.
2. Navrhněte systém umožňující monitorovat a modifikovat stav zvoleného multifunkčního zařízení. Modifikací se rozumí např. emulace přiložení čipové karty, stisk tlačítek apod. Při návrhu dbejte na možnost rozšíření o nové senzory a aktory. Zaměřte se především na implementaci senzorů a aktorů. Mechanická část (stisk tlačítek pomocí robota, upevnění, apod.) není předmětem zadání.
3. Zpracujte studii na výše uvedené téma.
4. Navržený systém implementujte formou hardwarového prototypu sestávajícího z centrálního prvku, ke kterému jsou připojeny senzory a aktory (např. emulátor čipových karet).
5. Vyhodnoťte a diskutujte parametry navrženého řešení.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 a 2 zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Vašíček Zdeněk, doc. Ing., Ph.D.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

prof. Ing. Lukáš Sekanina, Ph.D.
vedoucí ústavu

Abstrakt

Tato bakalářská práce popisuje návrh a implementaci platformy využitelné k testování vestavěných zařízení, jako jsou například tiskárny a skenery. Tato platforma je tvořena jak serverem, tak i prvky umožňující interakci s testovaným zařízením. Tyto prvky lze ovládát přes server, který umožňuje posílat požadavky přes REST API. Výsledkem této práce je tedy funkční aplikace systému zahrnující vytvoření hardwarového prototypů emulátoru karty a sensorů papíru pro umožnění testování základní funkcionality tiskáren. Ovšem v budoucnu můžou přibýt požadavky pro vývoj dalších testovacích prvků, proto byla platforma vyvinuta s ohledem na snadnou rozšiřitelnost o nové sensory a aktory.

Abstract

The bachelor thesis describes design and implementation of a platform which can be used for testing embedded devices like printers and scanners. The platform is composed of the server and concrete testing elements interacting with devices. The elements can be controlled via the server, which hosts REST API. The result is the fully functional application system including creation of hardware prototype of card emulator and paper sensors for testing the basic functionality of printer devices. However, in future new requirements can be added for testing, so the platform has been developed with easy extensibility by new sensors and actuators.

Klíčová slova

Vestavěný systém, Testování, Emulace karty, Sensory, Aktory, C#, C, .NET Core, ASP.NET Core, Swagger, CAN

Keywords

Embedded system, Testing, Card emulation, Sensors, Actuators, C#, C, .NET Core, ASP.NET Core, Swagger, CAN

Citace

DUŠEK, Oto. *Platforma pro automatické testování vestavěných zařízení*. Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. Zdeněk Vašíček, Ph.D.

Platforma pro automatické testování vestavěných zařízení

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Doc. Ing. Zdeňka Vašíčka, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Oto Dušek
18. května 2018

Poděkování

Rád bych poděkoval panu Doc. Ing. Zdeňkovi Vašíčkovi, Ph.D. za vedení této bakalářské práce. Také bych rád poděkoval Jirkovi Kyzlinkovi za cenné připomínky a rady, které práci usměrnily tím správným směrem. Dále Martinu Duškovi a Janovi Dudysovi za jazykovou korekturu textu.

Obsah

Kapitola 1

Úvod

Kromě vývoje nových produktů je důležité i jejich testování. Avšak manuální testování je často zdlouhavé a finančně náročné, obzvláště při opakovaní stejných testů na více platformách. Často ani nelze ručně odhalit některé chyby. Chyba nastane například jednou z tisíce pokusů. Proto je důležité a někdy nezbytné využívat výhody spojené s automatickým testováním. Kromě klasického testování výskytu chyb lze systém pro automatické testování využít například i pro měření času odezvy aplikace.

V minulosti nebylo jiného zbytí než testovat vyvinuté aplikace ručně, dnes jsou však technologie natolik pokročilé, že vývoj automatizovaného testování se výrazně posouvá dopředu, proto pro testování lze využít specializovaný systém pro automatické testování vestavěných zařízení. Existující systém se v současné době skládá z robotické ruky a kamery. Robotická ruka se stylusem umožňuje klikat na dotykový displej a ze zpětné vazby z kamery lze zjistit, zda se aplikace chová, jak bylo zamýšleno. Ale u mnoha případů testovacích scénářů nestačí pouze robotická ruka a kamera. Proto je cílem této práce vytvořit doplňující část pro již existující systém, který umožní používat při testování různé sensory a prvky interagující se zařízením, a ne pouze kameru a robotickou ruku. Důležitým prvkem při testování např. tiskárny může být počítadlo vytisknutých papíru nebo emulátor karty, který umožní kontrolu ověřování přístupu osob. Ale protože toto nebudou jediné prvky využitelné při testování, tak důležitým kritériem při vývoji této platformy je i jeho snadná rozšiřitelnost o další sensory a aktory.

V první části práce jsou popsány informace a vědomosti důležité pro plné porozumění problematiky. Například odlišné způsoby testování, popis různých vhodných sběrnic pro použití v systému. Dále knihovny, které usnadní vývoj jak aplikací pro počítač, tak i tvorbu firmwaru pro mikroprocesory.

V další části se bakalářské práce zaobírá návrhem kompozice systému a komunikace v rámci celého doplňujícího systému sensorů a aktorů. Dále návrhem konkrétních testovacích prvků využitelných hlavně u testování tiskáren a konečně také samotnou realizací systému.

Kapitola 2

Automatické testování vestavěných zařízení

Testování je dnes nedílnou součástí vývoje nových produktů, a protože automatizace přináší úspory, tak lze očekávat, že se zabývá mnoho lidí vývojem nových metodik a nástrojů pro co nejsnadnější a nejrychlejší otestování softwaru. Je mnoho druhů testování, zde jsou představeny jen zcela základní způsoby.

Testování vestavěných zařízení můžeme klasifikovat například podle typu automatizace. Existují dva odlišné přístupy. Prvním z nich je manuální testování. Tento druh testů provádí lidé, s tím souvisí i jeho nevýhody, které můžou být reprezentovány nedbalostí člověka nebo rychlostí testování. Tento přístup je však často nenahraditelný, protože je obtížné vyvinout takové testy, které by z části nebo zcela nahradily člověka. Dalším případem, kde vyniká manuální testování, spočívá v tom, že pokud výsledný systém má být využíván lidmi, tak můžou manuální testy lépe simulovat to, jak bude systém v budoucnu využíván. Člověk může vnést do testů i určitou náhodu v tom, jak bude provádět testy, a díky tomu může objevit více chyb.

Dalším druhem testování je testování automatické. To probíhá tak, že člověk napíše ve specializovaném testovacím prostředí seznam úkonů potřebných pro vykonaní testů a poté může tyto předepsané testy opakovaně spouštět, například při každém upravení zdrojového kódu testovaného softwaru. Automatickými testy však nemusíme testovat jenom jednu část softwaru (jako jednotkové testy), ale můžeme testovat celý systém, to znamená, že robotem můžeme ověřovat správnou funkčnost obrazovky, na kterou kliká, a poté ověřovat, zda doteky vyvolaly správné akce, např. kontrolovat, zda z tiskárny vyjel papír předpokládané velikosti.

Tento druh testování se dnes hodně prosazuje. Šetří práci lidem, kteří pak můžou vykonávat užitečnější úkoly. Takže pokud to jde, má smysl veškeré testy automatizovat. Tento přístup je v některých případech však až několikanásobně složitější realizovat než přístup manuálního testování. Kupříkladu při testování vestavěných zařízení musíme mít vytvořenou velmi složitou infrastrukturu, která bude provádět testy.

Často se testování skládá z verifikace a validace. Cílem verifikace je ověřit, zda testovaný systém vyhovuje návrhu (specifikacím). Neboli ověřujeme, zda produkt, který vytváříme, splňuje všechny požadavky zákazníka. To znamená, že má systém všechny vlastnosti, které jsou požadovány [22]. Naproti tomu pomocí validace se ověřuje, zda testovaný systém má všechny vlastnosti, které by uživatel od něj očekával, ale nejsou ve specifikacích. Mezi

nejčastěji ověřované vlastnosti patří rychlá odezva systému nebo zda určitou operaci může uživatel vykonat do daného času [22].

Kapitola 3

Komunikační rozhraní vestavěných zařízení

Protože bude nutné v rámci systému komunikovat mezi jednotlivými komponenty, tak zde jsou představeny nejběžnější komunikační periférie.

3.1 USB

USB je univerzální sériová sběrnice (Universal Serial Bus). Je to jedna z nejvhodněji použitelných sběrnic pro připojení jednoduchých i složitějších periférii k počítači. A to díky podpoře ve všech moderních počítačích a nenáročné softwarové aplikaci.

Komunikace na sběrnici je koncipována jako master-slave. To znamená, že přenos dat řídí vždy master. Pokud chce zařízení na sběrnici poslat data, musí počkat až do okamžiku, kdy k tomu bude vyzván od mastera.

Přenos dat je logicky rozdělen do koncových bodů (endpoint). Každé zařízení může obsahovat až 16 výstupních a 16 vstupních koncových bodů. Logické spojení mezi hostitelem a konkrétním koncovým bodem se nazývá kanál. Kanály se dále dělí na proudový a kanál zpráv. Kanál zpráv umožňuje u komunikaci a lze ho využit u ovládacího přenosu (viz níže). Naproti tomu proudový kanál je jednosměrný a používá se u přenosu hromadného, izochronního nebo u přenosu přerušení. Tyto typy přenosů jsou popsány v sekci 3.1.2. [14]

3.1.1 Deskriptory

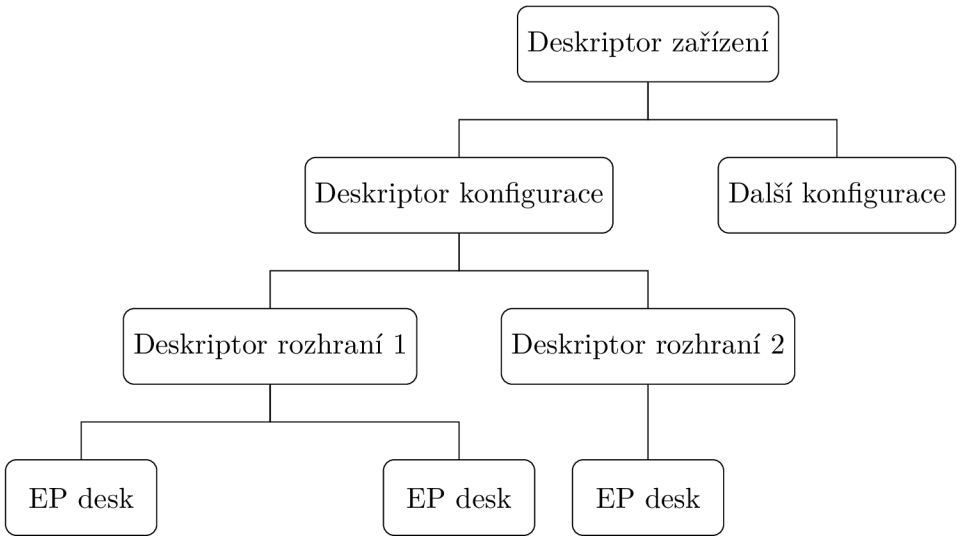
Každé USB zařízení musí mít nadefinovány struktury, které popisuje, jak se zařízení chová nebo jak s ním lze interagovat. Těmto strukturám se říká deskriptory. V zásadě existují 4 druhy deskriptorů, jenž jsou hierarchicky uspořádány (viz diagram 3.1). Nejvýše položeným typem je deskriptor zařízení (Device Descriptor), ten může mít každé fyzické zařízení pouze jedno. Obsahuje identifikátor USB produktu (PID) a výrobce (VID), ještě lze zjistit, jakou verzi USB zařízení podporuje, a další základní informace o zařízení.

Dále zařízení může obsahovat jeden nebo více konfiguračních deskriptorů. Aktivní však může být pouze jeden. To, která konfigurace bude zvolena, si může zvolit samotné zařízení nebo operační systém, např. podle spotřeby energie jednotlivých konfigurací (pokud zařízení připojíme k mobilu nebo k jinému bateriově napájenému hostiteli, tak si operační systém zvolí méně energeticky náročnou konfiguraci). Tyto konfigurace můžou být naprosto odlišné, mohou mít jinak definované deskriptory rozhraní (viz níže). Jedna konfigurace se může chovat jako myš a druhá jako klávesnice a mezi nimi lze podle potřeby přepínat vol-

bou určité konfigurace. Deskriptor obsahuje způsob napájení (vlastní nebo ze sběrnice), maximální spotřebu energie, množství obsahujících deskriptorů rozhraní této konfigurace a další atributy.

Deskriptor koncového bodu (endpointu) je dále nedělitelný, popisuje typ a způsob přenosu dat (jednotlivé typy jsou popsány níže). Obsahuje informace o směru toku dat, maximální velikost paketu, interval dotazování na nová data (maximální interval odesílaní zpráv) a také adresu identifikující, díky kterému poznáme k jakému endpointu data přísluší. Jsou dva druhy koncových bodů, jeden se označuje IN pro posílaní dat směrem ze zařízení do hostitele a druhý OUT pro posílání dat opačným směrem.

Deskriptor rozhraní sdružuje jednotlivé koncové body (endpointy) do funkčních celků. Například to může být USB reproduktor s tlačítky ovládající přehrávání hudby, ten tedy bude obsahovat endpoint, který zprostředkovává přenos zvuku, a dále endpoint, který bude zajišťovat přenos stisků kláves. Tyto dva koncové body mohou být umístěny v rámci jednoho deskriptoru rozhraní. To znamená, že se bude zařízení v operačním systému zobrazovat jako jedno. Pokud máme zařízení, které má více funkcí, tak můžeme použít více deskriptorů rozhraní. Například jeden popisující tiskárnu a druhý skener. Narozdíl od konfiguračního deskriptoru zde není omezení na počet aktivních rozhraní najednou, takže bychom mohli zároveň využívat funkcionalitu skeneru i tiskárny. [14]



Obrázek 3.1: Příklad uspořádání USB deskriptorů (EP desk = Deskriptor koncových bodů)

3.1.2 Typy přenosů

Jak bylo popsáno výše, tak typ přenosu je atributem koncového bodu. Tento atribut určuje, k čemu může přenos sloužit a jakým způsobem jsou data přenášena. To znamená, že určují prioritu přenosu, maximální možnou přenosovou rychlost, popřípadě jestli zajišťují při komunikaci znovu odeslání ztracených paketů.

Ovládací přenos (Control transfer)

Tento druh přenosu je jediný obousměrný. Slouží pro konfiguraci USB zařízení těsně po přípojení na sběrnici, ale lze je používat i pro aplikačně závislou konfiguraci.

Hromadný přenos (Bulk transfer)

Používá se pro spolehlivý přenos většího množství dat. Pokud je to možné, využije veškerou volnou šířku pásma, kterou nevyužily ostatní přenosy.

Přenos přerušení (Interrupt transfer)

Tento přenos používají zařízení, která potřebují mít zajištěnou rychlou odezvu. Typickým příkladem takového zařízení je myš. Ta posílá pouze malé pakety, ale musí být zaručen co nejrychlejší přenos, aby myš okamžitě reagovala na pokyny uživatele.

Při použití tohoto typu přenosu můžeme očekávat garanci latence a dále detekci chyb. Chyba se může detekovat např. pomocí kontrolního součtu, který je přítomný ve všech paketech. Pokud se taková chyba detekuje, USB zařízení se pokusí paket poslat data znovu, dokud neuspěje.

Přesto, že se přenos nazývá přenos přerušení, tak nejde o stejný princip přerušení jako v mikrokontrolérech, kde se jedná o asynchronní události. Zde se musí USB master dokola dotazovat, zda zařízení nemá nová data k dispozici. Pokud tak nastane, až potom může USB zařízení zaslat požadovaná data. Díky tomuto principu se může snadno zjistit nutná přenosová rychlost pro obsluhu všech zařízení na sběrnici. Přenosová rychlost alokovaná pro všechny přenosy na sběrnici USB však nesmí přesáhnout 80 - 90 % celkové šířky pásma. Zbytek se rezervuje pro ovládací přenosy. A poté, co se alokují zdroje i pro ně, tak tuto rezervu mohou využívat i hromadné přenosy.

Data při použití tohoto druhu přenosu mohou být zasílaná každou milisekundu a velikost paketu může být až 64 bytů. Tyto údaje platí pro full-speed USB. U high-speed USB lze posílat každých 125 μs až 1024 bytů. Jak už bylo zmíněno intervaly, jak často mohou být data zasílána jsou definována v deskriptoru koncového bodu, který byl popsán v předcházející sekci. Tento interval koresponduje s tím, jak často se USB master dotazuje zařízení připojeného do USB. [14]

Izochronní přenos (Isochronous transfer)

Často používaný pro přenos zvuku. Zde totiž musí být zaručena co nejmenší latence, ale při přenosu nevadí, pokud dojde ke ztrátě malé části dat, protože ve výsledném poslechu hudby to nemusí být rozpoznáno. Naopak by bylo horší, pokud by se zvuk zpožďoval.

3.1.3 Třídy zařízení

Zařízení se dále dělí na různé třídy, které mají rozdílné využití, jako tiskárny, skenery, klávesnice, apod. Každý typ zařízení musí mít určitou třídu, jako kupříkladu klávesnice je třídy HID.

Je standardizováno mnoho tříd zařízení, zde jsou vyjmenovány pouze dva relevatní pro tuto práci.

HID (Human interface device)

Doslovný překlad názvu třídy znamená: zařízení interagující s člověkem. Do této třídy patří například myši, klávesnice nebo joysticky. Využívá ovládacího přenosu pro konfiguraci a přenosu přerušení pro přenášení informace o stavu zařízení. Díky využití přenosu přerušení je zaručena nízká odezva zařízení, ale za cenu nízké rychlosti přenosu dat, která je maximálně 64 kB za sekundu. Ale tento limit většině zařízení tohoto typu stačí.

CDC (Communication class device)

Používá se pro zajištění telefonních nebo síťových služeb. Často je tato třída také využívána u převodníku sériového portu, ale i jiných převodníků. Při připojení se často v operačním systému objeví jako sériový port, jehož použití už je jednoduché. Pro komunikaci není potřeba žádný dodatečný ovladač, proto je používání této třídy velice oblíbené.

3.2 CAN

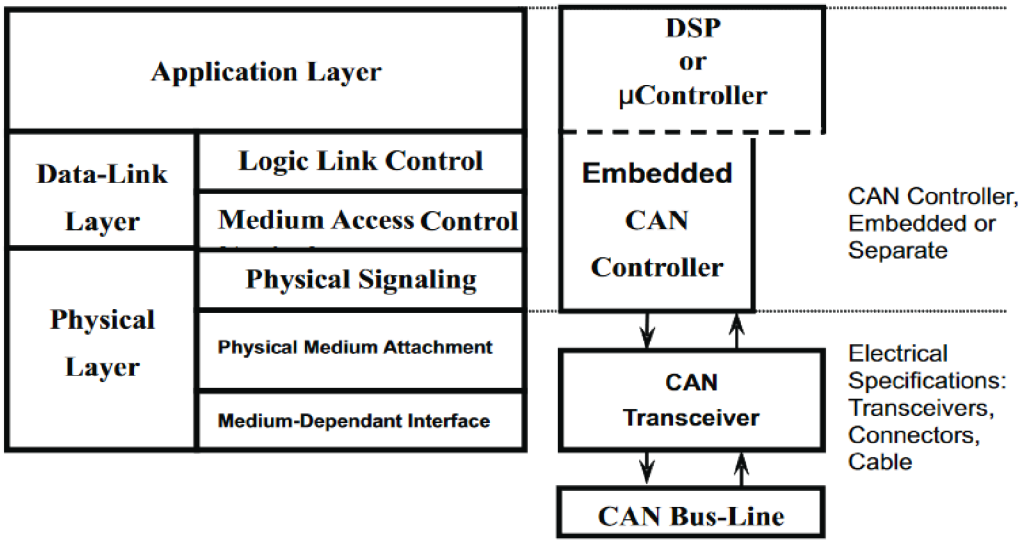
Sběrnice a protokol CAN byly vyvinuty firmou BOSCH už roce 1986 ([5]) primárně pro automobilový průmysl. V této době probíhala elektronizace aut, začaly se integrovat různá čidla a senzory. Před vývojem této sběrnice ovšem neexistovala jednotná komunikační linka, po které by mohly komunikovat všechny komponenty auta. Díky sběrnici CAN se všechna propojení nahradily jedinou linkou, přes kterou komunikovala zařízení v autě. Tím se v mnoha automobilech ušetřila značná část kabeláže.

Sběrnice se díky svým vlastnostem brzy velmi rozšířila do několika průmyslových odvětví. Nejrozšířenější se samozřejmě stala v automobilovém průmyslu, kde naprostá většina výrobců ve svých autech pro komunikaci mezi komponentami používá právě tuto sběrnici. Ale sběrnice CAN se nepoužívá jenom zde, je velmi rozšířená i v průmyslové automatizaci nebo v lékařství [5].

Sběrnice CAN je standardizována normou ISO pod označením ISO 11898. Standard specifikuje 2 vrstvy ISO/OSI modelu: fyzickou a linkovou. Tento standard počítal s 11 bitovým identifikátorem (standard CAN 3.2.3). Postupem času tato velikost identifikátoru nestačila, a tak se vytvořilo rozšíření normy ISO 11898, která identifikátor rozšířila na 29 bitů (Extended CAN 3.2.4) [5].

Po sběrnici CAN se komunikuje sériově pomocí dvou diferenciálních vodičů, které zajišťují vysokou odolnost vůči elektromagnetickému rušení a chybám. Tato odolnost je obvzláště nutná v automobilovém průmyslu. Další důležitou vlastností je, že na sběrnici nemusí být tzv. master, který by musel řídit komunikaci na sběrnici. To umožňuje zařízením komunikovat i mezi sebou a mohou také začít vysílat kdykoliv nezávisle na řídící jednotce. Jsou zde pouze důležité omezující podmínky, které zajišťují, aby uzly nenarušily stávající komunikaci.

Typu sběrnice, kterou nemusí řídit master, se říká multimaster. Díky této vlastnosti může být celý systém daleko spolehlivější, protože při poruše jednoho uzlu může komunikace probíhat nadále, což u opačného typu sběrnice, kde musí být řídící člen, neplatí. Pokud by přestal tento člen fungovat, tak nebude fungovat celá sběrnice.



Obrázek 3.2: CAN architektura, zdroj [17]

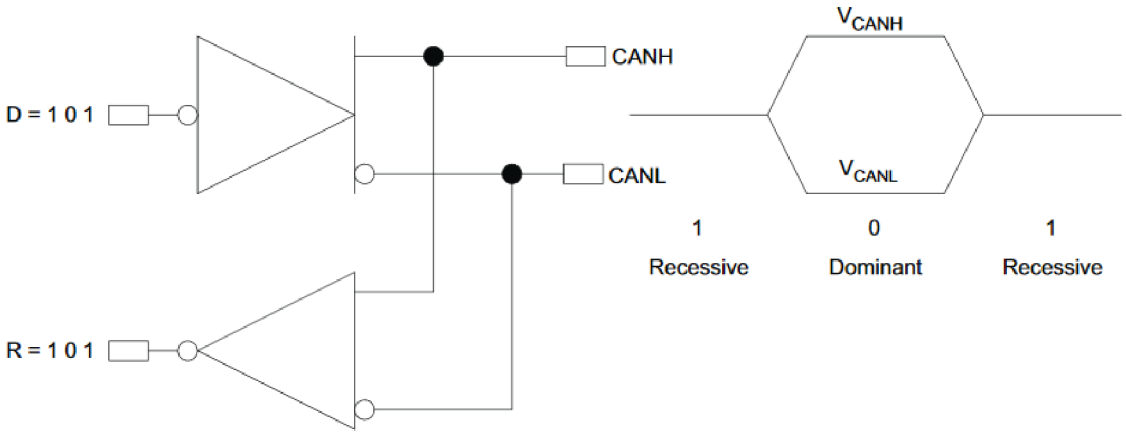
Obrázek 3.2 popisuje jednotlivé vrstvy sběrnice CAN a jakými komponentami mohou být vrstvy implementovány. Fyzická a linková vrstva je definována ISO standardem. Fyzická vrstva je realizována CAN transceiverem a kontrolérem (většinou je součástí mikroprocesoru), ten budí CAN transceiver, který se stará o vysílání signálu na vodiče. CAN kontrolér implementuje také linkovou vrstvu. Takže se stará o detekci a řešení o chyb, provádí acknowledgement zpráv, atd. Nejvýše položená je vrstva aplikační, na této vrstvě může být postaven jakýkoli protokol či aplikace.

Kontrolér také implementuje filtrování zpráv, které je důležité obzvlášť při existenci více uzlů na sběrnici a jejich časté komunikaci. Filtrování se provádí na základě identifikátoru zprávy. V kontroléru můžeme specifikovat, které zprávy nás zajímají. Pokud nás zajímá více zpráv, můžeme specifikovat bitovou masku, která bude aplikována na identifikátory. Pokud zpráva projde touto maskou, může být vyvoláno přerušení zpracovávající zprávy CAN. Díky filtrování zpráv se ušetří výkon mikroprocesoru, který by tuto operaci musel provádět sám.

V následujících sekcích bude detailně popsáno, jak jednotlivé vrstvy pracují, jejich požadavky a čím jsou tvořeny.

3.2.1 Fyzická vrstva

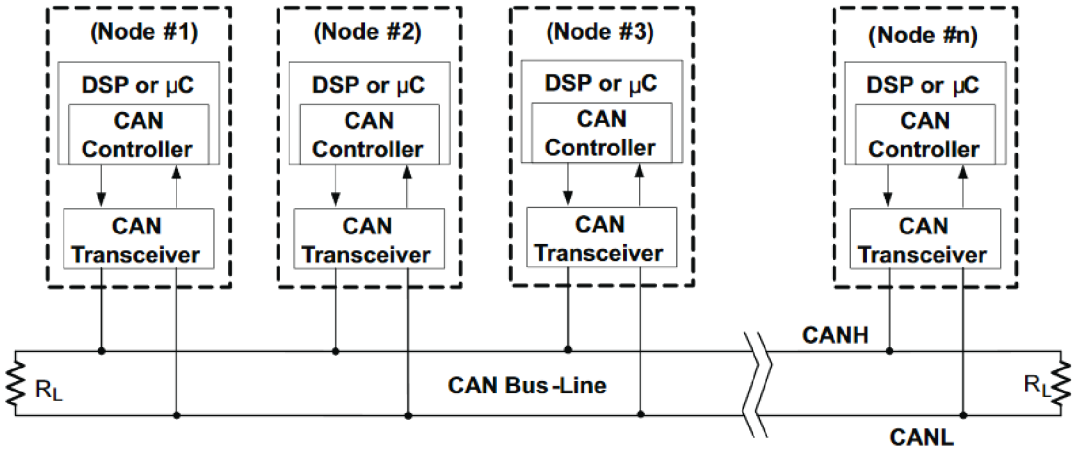
CAN sběrnice definuje i fyzickou vrstvu, díky tomu může zaručovat co největší spolehlivost pro přenos dat. Jedním z elementárních požadavků je, aby fyzické médium realizovalo logický součin všech signálů vyslaných na sběrnici. S tím standard definuje dva stavy, ve kterých se může sběrnice nacházet, a to stavy recesivní a dominantní. Tyto stavy jsou ekvivalencí k logickým úrovním a záleží jen na implementaci, jakým napěťovým úrovním budou odpovídat. Musí být pouze zajištěno, že v klidu bude linka v recesivním stavu [17].



Obrázek 3.3: Logika sběrnice CAN, zdroj [17]

Pro fyzickou realizaci přenosového média se nejčastěji používá dvou vodičů označovaných jako CANH a CANL, na které je signál vysílán diferenciálně a výsledný stav sběrnice je dán rozdílem těchto dvou vodičů. Obrázek 3.3 ukazuje příklad možného vysílaného signálu a jeho průběh. Lze si všimnout, že recesivní úroveň je reprezentována nulovým rozdílem potenciálů mezi vodiči a dominantní nenulovým rozdílem. Jak název stavů napovídá, tak pokud dva uzly vyšlou dvě různé úrovně, tak na sběrnici vítězí dominantní bit. Této vlastnosti je využíváno u detekce kolize.

Jak je znázorněno na obrázku 3.4, sběrnice CAN musí být zakončena dvojící rezistorů R_L. Tyto odpory zajišťují dvě důležité vlastnosti. Tou první je, že pokud vysílající uzel vyslal dominantní úroveň a následuje recesivní úroveň, tak odpor zajistí vybití náboje a přechod zpátky do recesivního stavu. Druhým důležitým úkolem těchto rezistorů je snížit odrazy na vedení. Čím delší vodiče, tím větší jsou odrazy na vedeních, což při větších přenosových rychlostech může způsobovat problémy a tehdy jsou i rezistory na sběrnici nutné. Při kratších vzdálenostech do desítek centimetrů se sběrnice bez těchto rezistorů obejde, stačí je nahradit jedním rezistorem s menším odporem.



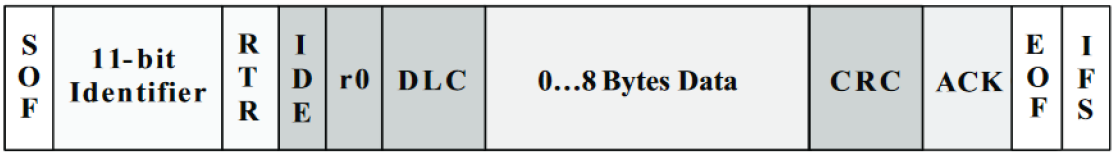
Obrázek 3.4: CAN sběrnice, zdroj [17]

3.2.2 Linková vrstva

Jak je vidět na obrázku 3.2, linková vrstva se děli na dvě podčásti. Jednou částí je MAC (Medium Access Control). Jejím úkolem je kódovat data do vysílaných rámců, vkládat bity, které minimalizují stejnosměrnou složku vysílaného signálu (pro zvýšení odolnosti proti rušení), řídit přístup k médiu, detekovat kolize a jiné chyby či potvrzování úspěšně přijatých zpráv. Druhou podčástí je LLC (Logical Link Control), ta se stará o filtrování přijatých zpráv či podávání hlášení o přetížení.

Různé protokoly řeší kolize různými způsoby, existuje nespočet způsobů detekce kolize i zotavení se z ní. CAN tento problém řeší protokolem **CSMA/CD+AMP**. CSMA (carrier-sense, multiple-access) definuje, že pokud chce uzel vyslat zprávu, tak musí nejprve zjistit, zda nějaký uzel již nevysílá na sběrnici. Poté pokud je volná sběrnice, tak počká náhodný interval a začne vysílat. Pokud se i tak stane, že začnou vysílat dva uzly najednou, tak se detekuje kolize. Tím, že výsledný signál na sběrnici je daný logickým součinem všech vysílaných signálů (viz obrázek 3.3) a odesílající uzly poslouchají na sběrnici i po dobu odesílání a detekují, zda odeslaný signál je roven signálu na sběrnici. Pokud není roven, tak zařízení ví, že je detekována kolize, a uzel okamžitě přestane odesílat data. Protože se všechny signály vysílané na sběrnici získávají operací logického součinu, tak větší prioritu má zpráva s nejmenším identifikátorem [17]. [17]

3.2.3 Standard CAN



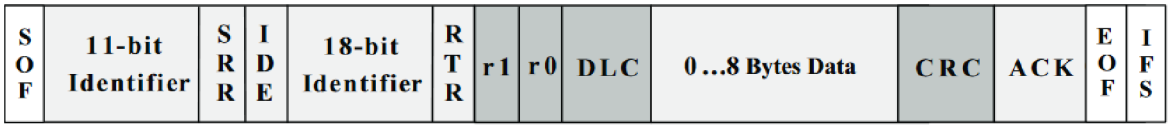
Obrázek 3.5: CAN 11bitový identifikátor, zdroj [17]

Popis jednotlivých bitových polí:

- SOF - Dominantní bit, který označuje začátek zprávy a slouží k detekování, zda je sběrnice volná (viz CSMA)
- 11bit identifier - Určuje prioritu zprávy (čím nižší hodnota, tím vyšší priorita)
- RTR (Remote transmission request)- Bit je dominantní, pokud odesílatel zprávy očekává odpověď od příjemce; tato vlastnost se většinou nepoužívá
- IDE - Označuje typ zprávy CAN, pokud je dominantní značí Standard CAN, jinak označuje zprávu Extended CAN
- r0 - Rezervovaný bit
- DLC (Data length code) - Čtyřbitové neznaménkové číslo, definuje délku přenášených dat
- Data - Obsahuje přenášená data, maximální délka je 64 bitů
- CRC (Cyclic redundancy check)- Obsahuje kontrolní součet pro detekci bitových chyb

- ACK - Tento bit je ve výchozím stavu recesivní, pokud však příjemci nedetekují chybu, tak tento bit nastaví na dominantní úroveň, a tím odesílateli potvrdí úspěšné odeslání zprávy
- EOF (End of frame) - Označuje konec zprávy
- IFS - Používá se pro navázání odpovědi pokud je bit RTR dominantní

3.2.4 Extended CAN



Obrázek 3.6: CAN 29bitový identifikátor, zdroj [17]

Extended CAN je velice podobný, jediné rozdíly jsou v:

- Identifier - Přidáno další 18bitové pole pro identifikaci zprávy; dohromady umožnující přiřadit zprávě 29bitový identifikátor
- SRR - Vždy recesivní, je zde kvůli kompatibilitě se Standard CAN
- r1 - Přidán další rezervovaný bit

3.2.5 Typy rámců

CAN zařízení se můžou dorozumívat pomocí až čtyř různých typů rámců:

Datový rámec

Datový rámec je nejpoužívanějším typem rámce při běžné komunikaci. Obsahuje všechna pole, jak jsou popsána v Extended CAN (3.2.3) nebo Standard CAN (3.2.4). Od ostatních rámců se liší především bitem RTR, který má recesivní hodnotu. Dále rámec může obsahovat až osm bytů pro data.

Rámec žádosti o data

Tento rámec se používá v případě, že uzel na sběrnici potřebuje data od jiného uzlu. To znamená, že uzel pošle zprávu s bitem RTR nastaveným na dominantní hodnotu. Uzel, který má požadovaná data, naváže ihned na odeslanou zprávu pomocí nastavení bitu IFS. Tímto se zaručí, že odpověď je dodána ihned po odeslání. Pozor, rámec žádosti o data sám nemůže obsahovat data, pouze rámec odpovídající na tuto zprávu.

Chybový rámec

Chybový rámec má důležitou funkci při detekování chyby. Pokud jakýkoli uzel rozpozná, že vysláná zpráva má špatný formát nebo nesouhlasí kontrolní součet, tak pošle tento typ rámce. Odesílající uzel díky těmto zprávám pozná, že byla zpráva porušena a zprávu může

poslat znovu. Pokud uzel posílá chybné zprávy častěji než nad danou mez, tak se na čas umlčí. Může to značit, že je zařízení porouchané, a tak se zaručí chybami nenarušená komunikace.

Rámec indikující přetížení

Uzel nestíhající zpracovat přijímané zprávy vygeneruje tento typ rámce pro informaci ostatním uzlům. Ostatní uzly by se poté měly snažit posílat rámce s většími časovými rozestupy, aby zařízení stihlo rámce zpracovávat.

3.2.6 CAN transceiver

Protože většina mikroprocesorů má integrovaný pouze CAN kontrolér, je nutné pro komunikaci připojit k MCU CAN transceiver, který zajišťuje převod logiky mikroprocesoru na diferenciální úrovně používané na sběrnici.

Pro toto použití je například vhodný transceiver IFX1050GVIO především díky tomu, že podporuje 5V i 3,3V logiku komunikace s procesorem. Požadovanou napěťovou úroveň musíme připojit na pin V_{33V}. Dále je zde pin T_xD. Tento pin ovládá mikroprocesor, přijímané bity integrovaný obvod převede do diferenciálního signálu, který vyšle na piny CANH a CANL. Naopak na R_xD posílá obvod veškeré přijímané data ze sběrnice. To znamená, že i to, co mikroprocesor pošle na pin T_xD, tak se zpropaguje i na pin R_xD. Díky této vlastnosti může MCU zároveň naslouchat komunikaci na sběrnici a detekovat případné kolize. [9]

3.2.7 Sběrnice I^2C

I^2C je sériová poloduplexní sběrnice určená pro komunikaci na kratší vzdálenosti, většinou v rámci jedné desky plošných spojů. Na tuto sběrnici může být připojeno více zařízení, adresují se pomocí adresy, která je uvedená na počátku sekvence posílaných dat. Komunikace probíhá po dvou vodičích, po prvním vodiči je přenášen signál hodin, po druhém data. Výstupy budičů I^2C jsou v provedení otevřený kolektor. Na obou signálech tak musí být připojeny pull-up odpor, které v klidu definují na sběrnici logickou jedničku. [13]

Tuto sběrnici lze s výhodou použít tam, kde je potřeba minimalizovat počet vodičů. Lze totiž mít na sběrnici více zařízení, každé s jinou adresou. I^2C umožňuje přenášet data v různých módech [13]:

- Standard-mode - až do 100 kbit/s
- Fast-mode - až do 400 kbit/s
- Fast-mode Plus - až do 1 MBit/s
- High-speed mode Plus - až do 3,4 MBit/s
- Ultra Fast-mode - až do 5 MBit/s, ale pouze jednosměrně

3.2.8 SPI sběrnice

Podobně jako I^2C i SPI se většinou používá pouze pro komunikaci na kratší vzdálenosti, avšak je plně duplexní a neadresuje se pomocí adresy, ale každé zařízení má zvláštní vstup (CS - chip select) pro aktivaci. Pokud zařízení není aktivováno, nemůže vysílat data na sběrnici. SPI sběrnice využívá čtyři piny pro komunikaci. Na prvním pinu je signál hodin

(CLK), ten vysílá master na sběrnici pokud komunikuje on nebo chce, aby komunikovalo jiné zařízení. Dalšími využitelnými piny jsou MOSI (master output - slave input) a MISO (master input slave output). Ty slouží pro vlastní přenášení dat. Signál MOSI (Master Output Slave Input) vysílá master a poslouchá na něm zařízení (slave), u signálu MISO (Master Input Slave Output) je tomu naopak. [21]

SPI může být využito pro daleko rychlejší komunikaci než I^2C , díky tomu, že nevyužívá pull-up odporů. Jeho implementace je velice jednoduchá, prakticky se totiž skládá jen z dvou posuvných registrů. Jedním vstupním, do něj se zapisují data ze sběrnice, a druhým výstupním, do kterého se zapisují data pro odeslaní. Signál CLK určuje frekvenci posouvání dat posuvných registrů.

Kapitola 4

Prostředky pro realizaci vestavěných zařízení a prvků pro testování

Pro řízení a komunikaci mezi vestavěnými zařízeními lze využít různých komponent. Například lze využít mikroprocesoru s ARM jádrem, proto jsou zde popsány základy ARM architektury a mikroprocesor využívající této architektury. Dále se lze dozvědět informace o čipech, které by mohli být využity pro realizaci testovacích sensorů a aktorů.

4.0.1 ARM Cortex-M4

ARM je typ architektury hojně využíván ve vestavěných systémech a menších zařízeních, jako jsou chytré mobilní telefony, tablety, síťová zařízení a dokonce i klávesnice. A to především díky tomu, že vynikají vysokým poměrem výkonu a spotřeby.

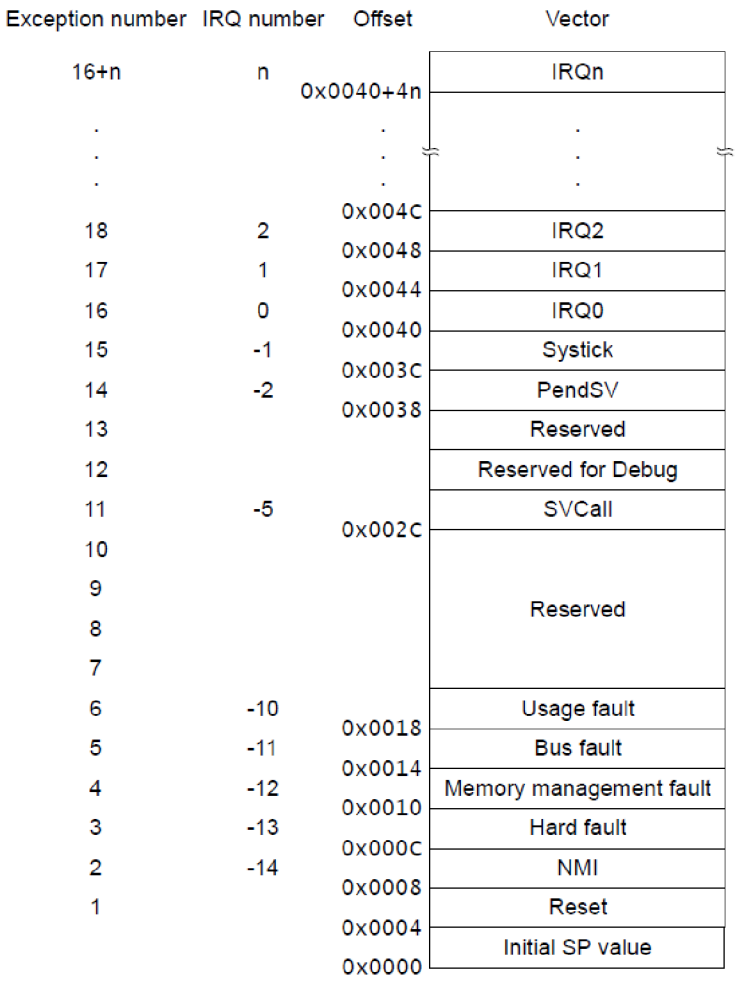
Jádro ARM vychází z RISC (Reduced Instruction Set Computing) architektury, která se snaží minimalizovat počet instrukcí a jejich složitost. Využívá nedestruktivního přístupu provádění instrukcí, což znamená například u operace sčítání využití 3 operandů (registr pro uložení výsledku a dva registry pro operandy). Při využití destruktivního přístupu by měla instrukce dva operandy s tím, že by se přepsal první operand výsledkem operace. Toto má negativní vliv na rychlost exekuce kódu, pokud bychom potřebovali znovu přepsaný operand, tzn. že bychom ho museli znovu vyčítat z paměti. Nedestruktivní vykonávání tak může minimalizovat přístup do paměti. Díky této vlastnosti mají i procesory ARM velký soubor registrů.

ARM Cortex-M4 využívá výhod jádra Armv7-M a doplňuje ho několika důležitými perifériemi jako je řadič přerušení (NVIC - Nested Vectored Interrupt Controller), řadič starající se o probuzení ze spánku, jednotka paměťové ochrany (MPU - Memory Protection Unit) nebo jednotky umožnující debuging a mnoho dalších.

Jeden z nejdůležitějších částí na porozumění je periférie NVIC, který se stará o obsluhu přerušení. Podporuje nastavení až 255 různých priorit různým přerušením, kterých může být až 240. Dále NVIC podporuje vnořené vykonávání přerušení. To znamená, že pokud se vykonává přerušení a vyvolá se přerušení s větší prioritou, tak se začne vykonávat nově vyvolané a po dokončení se vrátí do předchozího přerušení. [4]

Dalším důležitým pojmem ve světě ARM architektury je tabulka vektorů. Ta bývá nejčastěji uložena na začátku zkompilovaného programu v paměti FLASH. Na počátku je záznam, který obsahuje ukazatel na začátek zásobníku, a adresu instrukce, kterou mikro-

procesor začne vykonávat po resetu. Dále obsahuje nemaskovatelné přerušení. To jsou ty, jenž nejdou zakázat. Poté následují vektory různých typů chyb nebo systémových přerušení. Vektory s číslem IRQ 0 až n jsou využívány pro signalizaci přerušení periférií jako je například GPIO, SPI a dalšími.



Obrázek 4.1: Tabulka vektorů ARM Cortex-M4, zdroj [4]

4.0.2 STM32L432KC

Řada mikroprocesorů STM32L4xx od společnosti STMicroelectronics vyniká svou nízkou spotřebou, přičemž je založena na jádře ARM Cortex-M4 [18]. Tato 32bitová RISC architektura je jedna z nejrozšířenějších a nejpoužívanějších ve všech mikroprocesorech používaných ve vestavěných systémech. Konkrétně MCU STM32L432KC má parametry následujicí [18]:

- Pouzdro: UFQFPN-32
- Maximální frekvence: 80 MHz
- Velikost paměti Flash: 256 kB
- Velikost paměti RAM: 64 kB

- Počet časovačů: 7 16bitových a 1 32bitový
- Počet DA převodníků: 2
- Počet AD převodníků: 1
- Počet I^2C periferií: 2
- Počet SPI periferií: 2
- Počet CAN periferií: 1
- Počet USART a UART periferií: 3

Tento mikroprocesor má dostatek periférii a dostatečný výkon. Takže by měl stačit pro řízení většiny sensorů a aktorů. Další nespornou výhodou je, že společnost STMicroelectronics vyrábí vývojové kity Nucleo, které už obsahují debuger a programátor ST-LINK/V2-1, takže je ideální pro rychlé vytvoření a zprovoznění prototypu vyvíjeného systému.

Společnost STMicroelectronics dále usnadňuje vývoj vestavěných aplikací. Je možnost využívat konfiguračního nástroje STM32CubeMX, pomocí kterého lze nakonfigurovat registry mikroprocesoru v grafickém rozhraní. Například lze nakonfigurovat zdroj hodin pro jednotlivé sběrnice, inicializovat komunikační periférie jako je SPI nebo CAN. Tento nástroj po nastavení všech parametrů umožňuje vygenerování projektů pro různá integrovaná vývojová prostředí.

4.0.3 PN532

Modul PN532 umožňuje bezdrátovou komunikaci ve frekvenčním pásmu 13.56 MHz, toto je standardní frekvence pro RFID komunikaci. RFID (Radio frequency identification v překladu: Identifikace pomocí rádiových vln) se používá hlavně pro identifikaci osob a zboží. PN532 podporuje 6 operačních módů [12]:

- ISO/IEC 14443A/MIFARE Reader/Writer
- FeliCa Reader/Writer
- ISO/IEC 14443B Reader/Writer
- ISO/IEC 14443A/MIFARE Card MIFARE Classic 1K or MIFARE Classic 4K card emulation mode
- FeliCa Card emulation
- ISO/IEC 18092, ECMA 340 Peer-to-Peer

Módy se liší použitím různých standardů, ale hlavně rozdílnými úlohami při komunikaci. Čip umožňuje provozovat tyto druhy úloh: čtení karet, emulace karty a peer-to-peer komunikaci. V módu čtení karet vysílá čip periodicky rámec, kterým zjišťuje přítomnost karty. Pokud je karta v blízkosti, komunikace započne a čip vyčte data uložená na kartě. Existují i RFID čipy, do kterých lze zapisovat. Posledním případem je peer-to-peer mód, kde spolu můžou komunikovat dva uzly dat nebo specifický datový stream. [12]

Pro emulaci karty, lze využít pouze dva módy, a to mód ISO/IEC 14443A/MIFARE emulace karty nebo mód FeliCa Card emulation. Ovšem nejčastěji používaný a implementovaný standard v kartách je první zmiňovaný.

4.0.4 VL6180x

Sensor VL6180x vyvinula firma STMicroelectronics, využívá patentované technologie FlightSense. Tato technologie neměří množství odraženého IR záření, ale měří délku letu paprsku od sensoru k nejbližšímu objektu a zpátky. Díky tomu není vzdálenost závislá na ambientním osvětlení, ani na povrchu a barvě objektu, od kterého se světlo odrazí. Se sensorem lze komunikovat pomocí I^2C rozhraní a je napájen napětím 2,8 V. [19]

Kapitola 5

Možnosti využití knihoven a frameworků

Při vývoji komplexního systému je vhodné řešené problémy správně rozdělit mezi hardware s omezenými možnostmi a hardware, kde lze řešit jednodušeji složitější úlohy. V další části jsou tedy popsány knihovny a frameworky usnadňující vývoj jak firmwaru, tak i obslužné aplikace.

5.1 Vývoj firmwaru

Existuje veliké množství operačních systémů pro vývoj vestavěných aplikací. FreeRTOS je zde popsán hlavně z důvodu jeho rozšířenosti a široké podpory od výrobců mikroprocesorů.

5.1.1 FreeRTOS

FreeRTOS je operační systém reálného času. Zjednodušuje tak vývoj vestavěných aplikací, které kladou důraz na zpracování operací v reálném čase. A to jak ve správném logickém pořadí, tak i správném načasování.

Operační systém reálného času se nejčastěji používají při aplikaci vestavěných systému, kde čas odezvy je kritickou sledovanou veličinou. Při nedodržení kritickým limitů by například mohlo dojít k znehodnocení výrobků nebo v horším případě k újmě na zdraví obsluhy. **[[zdroj prezentace IMP]]**

Jeho nesporná výhoda tkví v tom, že jádro je napsáno pouze v pár zdrojových souborech a v paměti ROM i RAM tak zabírá pouze minimálně místa (běžně desítky kilobytů ROM paměti). Paměťová náročnost je odvislá od využití jednotlivých prvků operačního systému.

Tento operační systém umožňuje vytvářet úlohy (obdoba vlákna v normálních operačních systémech), přidělovat jim priority a spouštět je. Při vytvoření více úloh je pak klíčová jejich synchronizace, která je možná díky použití semaforů a mutexů. Pokud chceme periodicky vykonávat danou činnost, dají se využít časovače implementované programově v jádře systému.

Ve verzi systému FreeRTOS V10.0.0 přibylo několik dalších použitelných primitiv, které lze využít při vývoji vestavěného systému. FreeRTOS 10 představil proudovou vyrovnávací paměť (Stream Buffer) a vyrovnávací paměť zpráv (Message Buffer). Tyto paměti slouží pro komunikaci mezi úlohami nebo přerušeními a jsou optimalizovány pouze pro případ, že existuje jeden zapisující a jeden čtenář paměti. Běžným využitím této funkcionality může být zapisování příchozích požadavků a následné zpracovávání v úloze, pro urychlení práce

vykonávané v přerušení. Pokud je nutné mít více čtenářů a zapisovatelů, je nutné operace pro čtení a zápis do paměti dát do kritické sekce, kde nebude plánovač operačního systému přepínat procesor na jinou úlohu. Toto se docílí voláním API funkcí. [8]

Primitivum Message Buffer je postaveno nad Stream Buffer. Zatímco proudová paměť umožňuje posílat souvislý proud dat, paměť zpráv nám může posloužit pro posílání struktur, které můžou být různě dlouhé. [2]

Tato primitiva jsou velice užitečné pro zpracovávání požadavků přicházejících z přerušení, protože přerušení by se mělo provádět po co nejmenší dobu, kvůli blokování jiných přerušení.

5.2 Vývoj obslužné aplikace

Pro usnadnění vývoje obslužné aplikace lze využít mnoho knihoven a frameworků, které značné zjednodušují implementaci. Zde je několik z nich popsáno.

5.2.1 .NET Core

.NET Core je opensource platforma vytvořená společností Microsoft. Je to alternativa k .NET Framework, který byl vyvíjen primárně pro systém Windows. Podmnožina .NET Frameworku byla portována i na jiné platformy díky projektu Mono. .NET Core je oproti tomu od začátku koncipován tak, aby mohl běžet na všech běžných platformách. Primárně je cílen na vývoj webových aplikací, které lze jednoduše hostovat v cloudu, například v Azure, který nabízí přímo Microsoft. Díky multiplatformnosti frameworku si lze zvolit, zda chceme webový server mít na systému Windows, Linux nebo na jiném.

Kromě webových aplikací podporuje samozřejmě i standardní konzolové aplikace. Díky tomu si lze jednoduše vytvářet nástroje, které budou podporovány všech platformách. Vývoj grafických desktopových aplikací zatím jednoduše není možný. V současné době neexistuje žádná knihovna podporovaná Microsoftem, která by to umožňovala. Existuje pouze komunitní projekt Avalonia, který je zatím v beta verzi, takže není možné vytvářet plnohodnotné grafické aplikace.

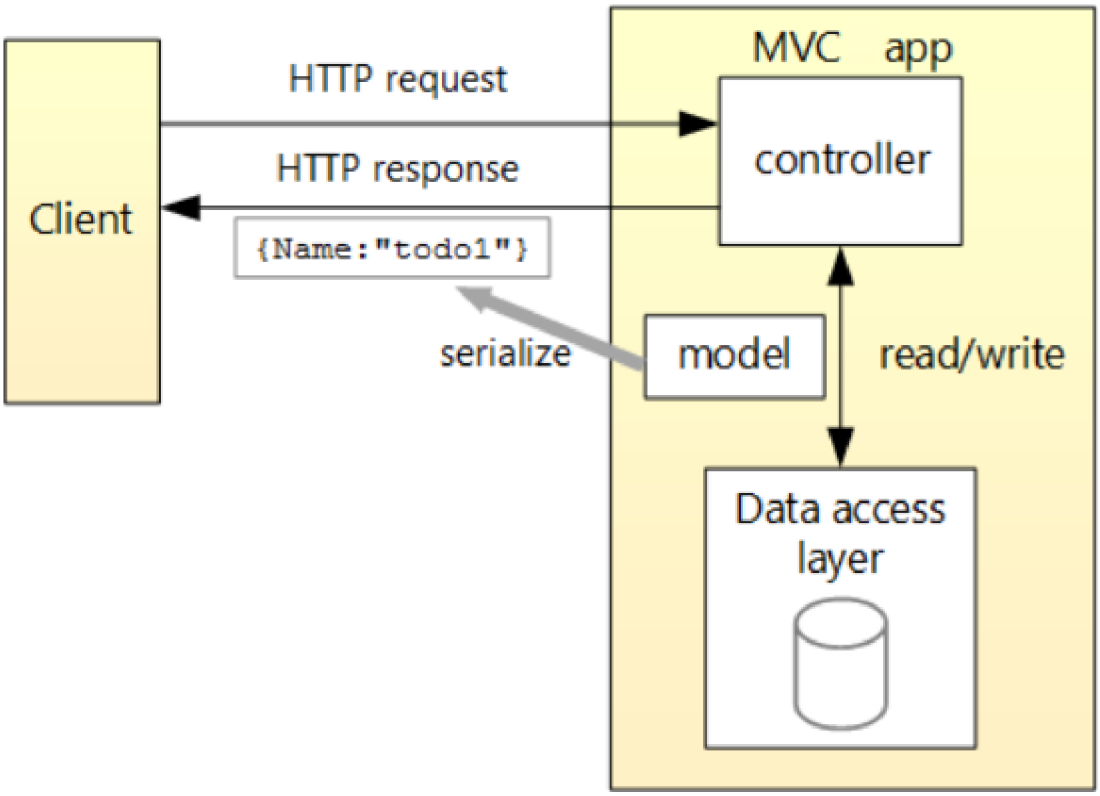
5.2.2 ASP.NET Core

ASP.NET Core je opensource framework pro vytváření webových aplikací. Může běžet na .NET Framework i na .NET Core, což umožňuje vytvářet webové aplikace pro všechny nejpoužívanější operační systémy. [6]

Předností tohoto frameworku je jednoduchý návrh a realizace webového serveru. Toho je dosažené díky MVC (Model-View-Controller) architektuře. Tato architektura je obecně využívaná při vytváření grafických aplikací, avšak v ASP.NET Core ji lze využít pro stavbu REST API serveru, s tím že nebudeme využívat složku vytvářející grafické rozhraní (View). [6]

Jak je znázorněno na obrázku 5.1, REST API aplikace se může skládat ze dvou částí. První částí je klient, který klade dotazy serveru. Druhou částí je server, který obsahuje tzv. kontrolér a datovou vrstvu. Ta předává data kontroléru ve formě modelu (model je objekt, který lze serializovat, v jazyce C# je reprezentovaný instancí třídy). Kontrolér obsluhuje dotazy a na tyto dotazy patřičně odpovídá například ve formě serializovaného modelu. Serializování dat lze provádět do více možných formátů, například do JSON, XML nebo ve formě prostého textu.

Datová vrstva může být implementována různě. Nejčastějším případem je použití relační databáze, ale může být použitý jakýkoliv zdroj relevantních dat.



Obrázek 5.1: Architektura Web API aplikace v ASP.NET Core, zdroj [10]

5.2.3 Swagger

Swagger je souhrn nástrojů a knihoven pro návrh, vytváření, testování, nasazování a dokumentování REST API služeb. Swagger je postaven nad OpenAPI. Tato specifikace popisuje rozhraní nezávislé na programovacím jazyku a použitelné pro popis REST API, které umožňuje uživateli jednoduše porozumět a využívat REST API služby popsané touto specifikací. [3]

Tato OpenAPI specifikace se popisuje nejčastěji pomocí YAML nebo JSON formátu. Může být ručně psána člověkem nebo se může vygenerovat specializovaným softwarem jako je například Swashbuckle (více 5.2.3) podle již vytvořeného serveru.

Swagger UI

Swagger UI je webová dokumentace REST API vygenerovaná z OpenAPI. Tento web přehledně vypíše všechny operace, které lze posílat serveru. U každé zobrazí její cestu, parametry a jejich popis, a dále možné návratové kódy. UI umožňuje také interaktivní posílání požadavků serveru a následné zobrazení jeho odpovědi. [3]

Swagger Codegen

Swagger Codegen projekt slouží pro automatické generování klienta nebo serveru na základě OpenAPI Specifikace. Tato specifikace může být napsána ručně nebo vygenerovaná například pomocí knihovny Swashbuckle 5.2.3. Tento nástroj vygeneruje zdrojové kódy, které lze libovolně upravit. Generování zdrojových souborů je možné do mnoha jazyků, například: C#, C++, Java, Python, Rust a mnoho dalších. Přidání podpory jiných jazyků je velmi jednoduché díky šablonovému systému. [3]

Swashbuckle

Swashbuckle je port Swaggeru pro prostředí ASP.NET Core. Tato knihovna zajistí automatické vygenerovaní dokumentace k REST API definovaného v ASP.NET Core pomocí architektury MVC. Umožňuje také vytvoření webové stránky, která obsahuje interaktivní dokumentaci REST API, více 5.2.3. Díky této vlastnosti je vývoj a testovaní API jednodušší a rychlejší.

Swashbuckle při spuštění vygeneruje OpenAPI specifikaci z XML dokumentace vytvořené kompilátorem C#. To znamená, že obsah komentářů jednotlivých funkcí reprezentující REST API požadavky budou i ve webové dokumentaci. Díky tomu můžeme okomentovat REST API pouze jednou v kódu. Po vygenerování specifikace už stačí otevřít stránku s dokumentací. Tato stránka nahraje OpenAPI specifikaci a vygeneruje pro každé REST API volání pole, které obsahuje informace a parametry důležité pro posílání REST API žádosti. Tato dokumentace je i interaktivní, což znamená, že můžeme zadat parametry a odeslat žádost serveru. Následně se nám zobrazí návratový kód i tělo odpovědi (vygeneruje i příkaz pro program cURL umožňující posílání dotazu serveru z příkazové řádky). Toto velice usnadňuje debugování při vývoji serveru.

5.2.4 HIDAPI

HIDAPI je multiplatformní knihovna pro komunikaci s USB zařízeními třídy HID (Human interface device), jako jsou klávesnice a myši, mimo jádro operačního systému. Tuto knihovnu lze využívat na systémech Windows, Linux FreeBSD, and Mac OS X. Pokud vytváříme vlastní negenerická HID zařízení, tak díky této knihovně není potřeba psát nový ovladač pro různé systémy. [16]

HIDAPI nabízí konzistentní rozhraní pro každou platformu v jazyce C, pomocí které lze zapisovat, číst a zobrazit všechna HID zařízení s jejich podrobnostmi, jako je výrobce, název atd.

Pro použití této knihovny v .NET prostředí je potřeba vytvořit obal pro toto prostředí. Docílit tohoto v jazyku C# je velice jednoduché: stačí vytvořit třídu, ve které budou deklarované struktury a funkce nutné pro použití knihovny ,a poté pomocí atributu definovat v jaké knihovně DLL se funkce nachází. Poté už je možné volat vytvořené funkce. Tato volání se zpropagují na volání nativních metod.

5.2.5 Vkládání závislostí (Dependency injection)

Mechanismus vkládání závislostí je dnes nepostradatelnou součástí při vytváření softwaru v pokročilejších programovacích jazycích. Jeho princip spočívá v tom, že se nejdříve zaregistrují objekty (závislosti) a ty jsou automaticky dodány všem objektům využívající dané závislosti. Tyto závislosti se dodávají při vytváření instance třídy. Vkládání závislostí tak

často ušetří vytváření návrhových vzorů typu továrna, a tak není potřeba instanciovat závislost přímo v objektu. Tímto se také zaručí, že více objektů se stejnou závislostí, dostanou identickou implementaci dané závislosti.

Autofac

Komunitní knihovna Autofac slouží pro vkládání závislostí na platformě Microsoft .NET. Díky Autofac je mnohem jednodušší kontrolovat všechny závislosti ve vyvíjené aplikaci. V první fázi použití knihovny je potřeba vytvořit kontejner, přes který se zaregistrují všechny potřebné závislosti. V této knihovně existuje mnoho způsobů jak je zaregistrovat. Buď konkrétní instanci nebo můžeme také specifikovat, pod jakým rozhraním se má implementace zaregistrovat. Autofac umožňuje i proskenování aplikace a automatické registrování všech typů se specifikovanou vlastností.

Po zaregistrování všech potřebných závislostí se vytvoří strom vazeb mezi objekty a odhalí se případné kruhovité závislosti, které by nebylo možné vyřešit. Poté můžeme vytvořit instanci jednoho objektu a Autofac automaticky doplní všechny jeho potřebné závislosti. Doplňuje je rekurzivně. To znamená, že doplní i závislosti závislostí atd.

Při vytvoření nějakého celku, který má například stejné závislosti, je výhodné použít tzv. modulárního systému. V jenom modulu zaregistrujeme všechny spolu související závislosti. A poté, pokud chceme využít tento celek, tak můžeme zaregistrovat celý modul a ne pouze jednotlivé závislosti. Tímto přístupem se může minimalizovat chyba, jejíž příčinou můžou být chybějící zaregistrované závislosti.

Autofac podporuje i konfiguraci a registrování závislosti přes konfigurační XML nebo JSON soubor. Takže stejným přístupem, jako se dají zaregistrovat závislosti v kódu, tak lze využít možnosti registrace v ve formě XML nebo JSON. Tento přístup k registraci ušetří čas tam, kde je nutné často měnit závislosti. Tato konfigurace umožňuje, že je lze měnit bez nutnosti rekompilace kódu. [1]

5.3 Moderní způsoby vývoje aplikací

Pro vývoj systému, který musí nějakým způsobem přijímat požadavky od uživatele, lze využít spoustu hotových přístupů. Jedním z dnes nejrozšířenějšího přístupu je REST.

5.3.1 REST

Princip návrhu REST definuje několik základních principů pro stavbu webových aplikací. Vyvinul ho Roy Fielding na Kalifornské univerzitě v roce 2000 v rámci disertační práce Architectural Styles and the Design of Network-based Software Architectures. Zde popisuje základní omezení a principy využívané pro návrh webových aplikací. Díky těmto stanovených pravidlům se zaručí škálování a konzistentnost rozhraní napříč různými aplikacemi i operačními systémy. [15]

Roy Fielding definuje několik důležitých omezení. Prvním z nich je Client-Server, díky tomuto omezení jsou jasně rozdělené úlohy jednotlivých komponent. Dalším omezením je bezstavovost. Ta zaručí spolehlivost a škálování, které je zajištěno díky tomu, že se nemusí udržovat stav mezi požadavky, a tak server může uvolnit rychle všechny zdroje. Navíc zjednodušuje implementaci díky tomu, že nemusí udržovat tyto zdroje.

Pro zajištění vyšší efektivity přenosu dat přes síť bylo přidáno omezení zajišťující mezipaměť mezi serverem a klientem. Pokud klient pošle požadavek a odpověď může být uložena

do mezipaměti (např. statický obrázek), tak se při příštím požadavku použijí data z této mezipaměti.

Díky volitelnému omezení, které zajišťuje rozložení úloh mezi různé vrstvy v systému, lze například v systému jednoduše rozložit zátěž mezi více prvků. Další omezení umožňuje. aby byla funkcionalita rozšířena pomocí stažení a spuštění kódu ve formě skriptů (například Javascript). REST definuje i několik dalších omezení, avšak popsáno zde bude pouze jedno, z důvodu jeho důležitosti. Tímto omezením je identifikace zdrojů. Klíčovou abstrakcí v REST je zdroj. Jakákoliv informace, která může být získána ze serveru může být zdroj. Mezi ně patří webová stránka, dokument, obrázek, ale i mnoho dalších. Tyto zdroje musí být nějakých způsobem identifikovány. Správce si tak může zvolit adresu pomocí, kterého bude zdroj identifikován (tato adresa se nezmění ani poté co se informace změní). [7]

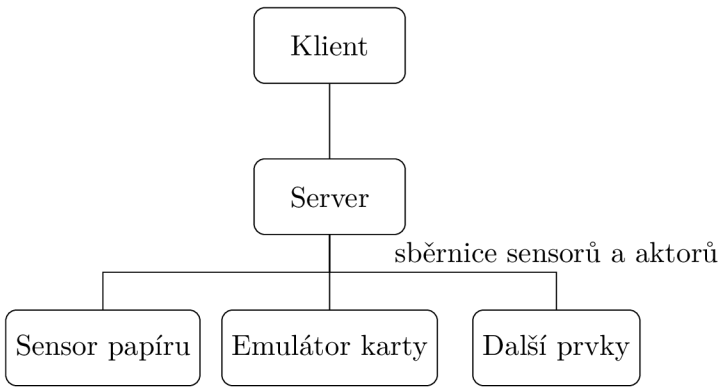
Pro manipulaci se zdroji RFC 2616 definuje několik základních metod. Díky těmto metodám je může uživatel vytvářet zdroje (metoda POST), získat zdroje (metoda GET), změnit zdroj (metoda PUT) nebo může smazat zdroj s pomocí metody DELETE. Tyto metody můžou vyžadovat i argumenty, které mohou být zadány jak v odkazu (např. /adduser?name=Roy), tak i v těle posílaného požadavku. [15]

Kapitola 6

Návrh systému

6.1 Kompozice systému

Systém se skládá ze 3 částí (viz obrázek 6.1). První částí jsou sensory a aktory, které interagují přímo s testovaným zařízením. Ty budou ovládány prostřednictvím serveru, který může běžet na desktopovém nebo jednodeskovém počítači. Server bude sbírat data ze sensorů a po zpracování je může poslat klientovi, který si o ně zažádá. Bylo by vhodné, aby klient mohl posílat požadavky přes internetovou síť pro co nejjednodušší vývoj a použití. Protože bude potřeba připojovat do systému více sensorů a aktorů, je nutné vybrat sběrnici, po které budou komunikovat. Kvůli různým požadavkům na připojení sensorů je vyhrazena následující sekce popisu výběru vhodné sběrnice pro komunikaci se sensory a aktory. Po nalezení vyhovující sběrnice bude nutné také navrhnout jednotlivé prvky systému a vymyslet protokol realizující komunikaci mezi sensory. Tím se zaobírá část 6.8.



Obrázek 6.1: Diagram systému

6.2 Výběr sběrnice a protokolu pro komunikaci se sensory a aktory

Jedním z nejzásadnějších bodů při vývoji systému je výběr nejvhodnější sběrnice pro komunikaci mezi počítačem a sensory. Následující seznam sdružuje nejdůležitější kritéria, které je nutné zvážit při výběru:

- odolnost vůči rušení - systém může být použit i v průmyslu

- multimaster sběrnice - zařízení budou v budoucnu moci komunikovat mezi sebou bez zásahu mastera
- jednoduchost použití

Prvním použitelným kandidátem by mohla být sběrnice RS485 (modifikace standardu RS232 - sériový port), což je poloduplexní sériová sběrnice, která umožňuje komunikovat rychlostí až 100 Mbps. Oproti RS232 vysílá signál diferenciálně pomocí dvou vodičů. Tento standard ovšem nepočítá s multimaster variantou, proto zde musí být řídící prvek a implementovaný řídící protokol jako například MODBUS.

Vhodnější bude použití sběrnice CAN, která definuje fyzickou vrstvu podobně jako RS485, avšak standard CAN uvažuje i připojení více než dvou komunikačních uzlů bez nutnosti řídícího prvku. Protože na tomto druhu sběrnice může nastat kolize, tak tento standard uvádí i způsoby její detekce a zotavení se z ní.

Dále bude nutné navrhnout protokol postavený na sběrnici CAN. Existují již hotové protokoly, ale většinou jsou buď proprietární, nebo je protokol velmi abstraktní, a tudíž režie nutná pro přenos dat není zanedbatelná. Například komunikační protokol CANopen díky jeho generičnosti využívá 50 % dat v jednom rámci na režijní informace. V systému může být časem potřeba přenášet větší množství dat nebo zaručení přenosu v co nejkratším čase. Z těchto důvodu bude nutné navrhnout vlastní protokol, který optimalizuje využití přenesených dat.

Protože CAN rozhraní dnes má jen málokterý počítač, proto je nutné navrhnout převodník komunikace, kterému bude možné zaslat data z počítače, a převodník se postará o správnou propagaci dat na sběrnici CAN sensorům a aktorům. Dalším důležitým úkolem bude vybrat použitelnou sběrnici, pomocí které se převodník připojí k počítači.

V dnešním světě existuje mnoho sběrnic pro připojení periférií k počítačí. Avšak některé z nich jsou složité na integraci do zařízení. Jedním ze snadno použitelných kandidátu na použití by byla sériová sběrnice UART. Sériovou sběrnici má téměř každý mikrokontrolér, takže implementace komunikace by nebyla komplikovaná. Ovšem tato sběrnice má jedno velké negativum, a to, že dnes už většina počítačů nemá vyvedenou právě sériovou linku.

Vhodnějším kandidátem pro komunikaci s převodníkem bude sběrnice USB, dnes nejrozšířenější sběrnice pro připojování uživatelských periférií k počítači. Výhodou je její existence na všech dnešních počítačích a i to, že při použití běžné třídy zařízení USB nepotřebuje vlastní ovladače. Dále také existence periférie USB v mnoha dostupných mikrokontrolérech z něj dělá ideální volbu pro tvorbu zařízení připojitelného k jakémukoliv počítači.

Protože byla vybrána sběrnice USB je nutné zvolit konkrétní třídu USB zařízení, kterou bude převodník implementovat. Volba třídy vždy záleží na způsobu použití a požadavcích systému. Ty budou v našem případě dva, což je rychlost a spolehlivost doručení dat a absence potřeby psát vlastní ovladač pro zařízení. Rychlost doručení dat je nutná s ohledem na sensory vyvíjené v budoucnu, které by mohly klást podmínky na dodání dat v reálném čase.

Těmto podmínkám vyhovují dvě třídy - CDC a HID (ty jsou popsány v sekci 3.1.3 a 3.1.3). Výhoda třídy CDC je jednoduché použití bez použití dalších knihoven, avšak někdy může docházet ke zpožďování příchodu dat. V tomto případě excelují zařízení třídy HID, která zaručují dodání dat v minimálním možném čase. Použití této třídy není tak přímočaré, jako použití CDC, ale existuje knihovna HIDAPI, která jejich použití maximálně zjednodušuje (knihovna je popsána v sekci 5.2.4).

6.3 CAN převodník

Pro umožnění komunikace sensorů se serverem je nutné navrhnout převodník, který bude připojitelný k počítači a převede komunikaci ze sběrnice USB na sběrnici CAN a naopak. Důležité parametry pro návrh a realizaci převodníku budou především tyto:

- Jednoduchost použití
- Snadná připojitelnost sensorů a aktorů.
- Umožňující 5V a 12V napájení sensorů.

Jak bylo nastíněno v kapitole 6.2 byla vybrána sběrnice USB s rychlostí full speed. Převodník bude implementovat zařízení třídy HID (Human input device), díky tomu nebude potřebovat doinstalovat ani programovat USB ovladač pro převodník a bude zaručovat včasné doručení dat.

Full speed USB může přenášet data o maximální velikosti 64 bytů s minimálním intervalem 1 milisekunda. To znamená, že maximální možná rychlost komunikace je 64 kB/s. Pokud by bylo nezbytné toto omezení obejít tak lze použít více USB koncových bodů nebo high speed USB (popsáno v 3.1), ale tato přenosová rychlost bude pro většinu sensorů a aktorů stačit.

Protože každé USB zařízení musí být popsáno pomocí deskriptorů, základním úkolem bude navrhnout, co budou jednotlivé deskriptory obsahovat. V tomto případě to bude jednoduché. Každé zařízení musí definovat jeden deskriptor zařízení, ten bude obsahovat námi zvolené VID (Vendor ID) a PID (Product ID) s ohledem na vyvarování se existujících identifikátorů kvůli možné kolizi při enumeraci zařízení. Dále musí definovat konfigurační deskriptor obsahující informaci o maximálním příkonu, dalších důležitých atributech, jako třeba záznam o počtu a typu deskriptorů rozhraní. V tomto použití bude obsahovat zařízení jedno rozhraní a dva koncové body. Jeden bude typu IN pro komunikaci směrem do hostitele a druhý typu OUT pro posílaní dat opačným směrem. Velikost posílaného reportu oběma endpointy zvolíme 64 bytů, protože to je maximální možná velikost dat posílaná skrze přenos typu přerušení. Tímto docílíme maximální možné přenosové rychlosti dat ze sensorů, jak je zmíněno u popisu USB přenosu typu přerušení. Více o USB v sekci 3.1.

Převodník bude obsahovat jeden USB micro konektor pro připojení k počítači a dvojici konektorů RJ12 pro možnost připojení sensorů a aktorů. Konektor RJ12 byl dříve využíván pro telefonní komunikaci a byl zvolen z důvodu jednoduché výroby kabelů a jeho robustnosti. Obsahuje celkem 6 pinů, což je ideální počet pro účely systému, protože potřebuje dva piny CANH a CANL pro komunikaci na sběrnici CAN, dva piny pro 5V a 12V napájení, a dva zemnící vodiče. 12V napájení se bude využívat v situacích, kdy může být potřeba vyvinout sensory nebo aktory s vyšším příkonem.

Tento převodník bude řídit mikroprocesor STM32F415RG od společnosti STMicroelectronics. V tomto mikroprocesoru je již implementována fyzická vrstva full-speed USB sběrnice, tudíž nejsou potřeba další součástky zabírající prostor na desce plošných spojů. MCU obsahuje i CAN kontrolér a měl by být dostatečně rychlý pro propagaci dat z USB na CAN a naopak. Takže by měl tento procesor vyhovovat všem požadavkům pro realizaci převodníku.

Výhoda mikroprocesorů řady STM32 tkví v tom, že mají svůj vlastní zavaděč (bootloader), který lze aktivovat přivedením příslušné logické úrovně na pin BOOT0. Díky tomu lze firmware nahrát do procesoru i bez použití externího programátoru. Toto se může hodit v případě, že systém již bude nasazený na mnoha místech a objeví se chyba v aplikaci nebo

bude potřeba doděla novou funkcionalitu do stávající aplikace. Díky zavaděči lze firmware jednoduše aktualizovat.

Protože bude s převodníkem nutné komunikovat přes USB a CAN rámec neobsahuje pouze jeden druh dat, ale je strukturovaný, musíme navrhnout protokol, pomocí kterého budeme zasílat převodníku rámce nebo naopak převodník bude nám přeposílat rámce přijaté na svém rozhraní CAN.

Struktura protokolu komunikace mezi převodníkem a serverem bude vypadat následovně. Obsahuje celkem 5 polí určených pro vytvoření CAN rámce:

1. 1 byte - Délka dat v rámci
2. 1 byte - Typ rámce
3. 2 byty - Adresa (Adresa a ID sensoru nebo aktoru)
4. 1 byte - Číslo rámce
5. 2 byte - Rezervováno
6. 8 bytů - Data rámce

Pole typ rámce, adresa a číslo rámce jsou určeny pro sestavení identifikátoru CAN rámce, který bude vždy 29bitový. Poté je zde pole délka, která může být v rozmezí 0-8 stejně jako u rámce CAN.

Díky tomu, že jsme si zvolili velikost HID reportu 64 bytů, tak můžeme do jedné zprávy zapsat až 4 CAN rámce, protože popsaná struktura má pouze 16 bytů. Ale protože velikost reportu musí být vždy stejná, tedy 64 bytů, musíme ošetřit případ, pokud budeme posílat menší počet rámců. Ostatním nevalidním rámcům tedy nastavíme délku větší než 8. Jelikož délka CAN rámce může být maximálně 8 bytů, tak můžeme v převodníku vyfiltrovat všechny příchozí rámce v reportu, které mají délku dat větší než těchto 8 bytů.

6.4 Server

Server by se měl starat o propagaci požadavků od klienta k sensorům nebo aktorům, vyčítání dat ze sensorů a o objevení dostupných zařízení na sběrnici. Se sensory a aktory bude komunikovat skrze převodník převádějící komunikaci z USB na CAN a naopak. S převodníkem se bude komunikovat skrze multiplatformní knihovnu HIDAPI, která zajistí přímočaré použití HID USB převodníku, přes který bude přenášen jednoduchý protokol popsaný v předcházející časti, kde je popisován CAN převodník.

Protože je žádoucí, aby mohl být server spuštěn na všech běžně používaných platformách, tak byl vybrán jazyk C#, který může být přeložen do tzv. bytecodu a spuštěn na frameworku .NET Core. Tento framework běží na většině běžných platforem.

Server nebude vykonávat žádné výpočetně náročné úkoly a není paměťově náročný, proto aplikace může běžet jak na desktopovém počítači, tak i na jednodeskovém počítači s ARM procesorem, jako je například Raspberry Pi. Což je velká výhoda z důvodů mobility, ceny a spotřeby elektrické energie.

Dalším úkolem serveru bude prezentovat data klientovi, a pro to byl vybrán protokol HTTP, konkrétně přístup REST API pro zpracovávání požadavků od klienta a posílání odpovědí ze sensorů a aktorů. Webové API bude realizováno pomocí frameworku ASP.NET

Core (více o frameworku v sekci 5.2.2), který usnadňuje vytváření webových aplikací. V serveru však využijeme pouze jeho malou podmnožinu funkcí, a tím je architektura MVC (Model View Controller). Vzhledem k tomu, že nepotřebujeme grafické rozhraní, tak nebudeme potřebovat všechny složky této architektury.

6.5 Klient

Klient bude moci posílat příkazy nebo vyčítat data ze serveru pomocí standardního REST API. Díky tomu je vyčítání jednoduché a jednoduše integrovatelné do systému, který by měl tuto platformu využívat. Při vývoji, debugování této platformy nebo při debugování testů je možné vyčítat data ručně. Toho lze docílit například pomocí prohlížeče, kam zadáme adresu serveru, identifikaci operace a jeho parametrů. Potvrdíme odeslání žádosti a přijde nám odpověď s výsledkem operace. Tento přístup je jednoduchý a účinný, pokud server obsahuje pouze malý počet operací, ale existuje knihovna, která tuto práci ještě usnadňuje. Je to knihovna pro ASP.NET Core Swagger (popsána v sekci 5.2.3) a je používaná pro generování dokumentace pro REST API aplikace. Tato knihovna vygeneruje všechny operace, které aplikace umožňuje využívat skrze REST API. U každé operace je její popis, parametry a její typický formát návratové hodnoty. Navíc je u každé operace ještě možnost odeslání požadavku se zadanými parametry a okamžité zobrazení výstupu operace. Toto významně zjednodušuje práci s REST API při debugování.

Kromě vytváření dokumentací knihovna Swagger podporuje i vygenerování klienta k danému serveru. Takže stačí napsat server ve frameworku ASP.NET Core, vygenerovat OpenAPI popis serveru a pomocí jednoduchého nástroje lze z tohoto popisu vygenerovat automaticky knihovnu obsahující všechny požadavky použitelné pro komunikaci se serverem. Tato vlastnost značně zjednodušuje práci, protože nemusí být odkazy natvrdo napsané přímo v kódu, ale stačí volat metody z této vygenerované knihovny. Navíc nástroj generující klienta nepodporuje jenom jazyk C#, ale mnoho dalších, takže lze vygenerovat a využívat knihovny v dnes všech běžně rozšířených programovacích jazycích.

6.6 Sensory a aktory

Pro účely testování a vývoj bude tato skupina prvků systému nejzajímavější. Do budoucna může být potřeba vyvinout mnoho druhů sensorů a aktorů, to záleží podle aktuálně zadaných požadavků na testování daného systému. Proto bude důležité, aby vývoj následujících testovacích prvků byl co nejjednodušší. Z těchto důvodů bude navržena jedna deska plošných spojů připojitelná do sběrnice CAN, z které bude vyvedeno několik nejdůležitějších periférii s více úrovněmi napětí (3,3 V a 5 V). To zaručí kompatibilitu s většinou potřebných sensorů a aktorů a také, že ve většině případů nebude potřeba vyvíjet další hardware společné části pro komunikaci na sběrnicí CAN. Pokud ovšem tato deska nebude splňovat očekávání u vývoje nového prvku, může se navrhnout nová deska plošných spojů, jako je tomu například u návrhu emulátoru karty.

Na univerzální desce plošných spojů budou celkem dva konektory RJ12 podobně jako u převodníku CAN pro snadné řetězení sensorů a aktorů za sebe. Dále bude obsahovat konektor s vyvedeným rozhraním SPI, další konektor s I^2C (s logickými úrovněmi 3,3 V) a jeden konektor s SPI (s 5V logickou úrovní).

Další nutností je implementace adresování sensorů a aktorů, protože u jednoho testovaného objektu může být více sensorů. Proto každý sensor bude mít své vlastní identifikační

číslo, které se bude moct nastavit skrze enkodér. Byl vybrán enkodér FR01KR16H od firmy NKK Switches. Tento enkodér je 4bitový, díky tomu bude možné adresovat až $2^4 = 16$ sensorů a aktorů.

Enkodér byl vybrán z důvodu jeho velikosti a jednoduchého použití. Pomocí otočné osy lze nastavit konkrétní číslo, které bude sensor používat pro komunikaci na sběrnici, díky tomu lze identifikovat zdroj nebo cíl zprávy. Jak je vidět z obrázku 6.2, tak enkodér má 4 výstupní piny označené čísly 1, 2, 4 a 8 a celkem 16 nastavitelných pozic: 0-F. Body v jednotlivých sloupcích a řádcích tabulky značí v jaké pozici jsou jaké piny propojené s pátým společným pinem (v našem případě bude připojen na zem). Po připojení jednotlivých pinů k mikrokontroléru lze na základě této tabulky tedy zjistit jaké číslo je navolené. Pozor na vstupech mikrokontroléru musí být pullup odpory.

TRUTH TABLES (CIRCUITS & POSITIONS)

Actuator Position ● = ON — Terminal No. (Output) — 16 Hexadecimal

R Real Coded Model Numbers: FR01FR, FR01KR, FR01SR, FR01AR	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1		●		●		●		●		●		●		●		●
2			●	●			●	●			●	●			●	●
4					●	●	●	●					●	●	●	●
8									●	●	●	●	●	●	●	●

Obrázek 6.2: Logická tabulka enkodéru FR01KR16H, zdroj [11]

Pro řízení byl vybrán mikroprocesor STM32L432KC, který má nízkou spotřebu, je dostatečně výkonný a má veškeré periférie nutné pro komunikaci se sensory a aktory, tudíž by měl stačit pro většinu běžných sensorů potřebných pro testování.

6.7 Způsob zapisování a čtení dat ze sensorů a aktorů

Z důvodu jednoduché manipulace se sensory a aktory, je nutné navrhnout abstrakci nad těmito úkony. Tato abstrakce také výrazně zjednoduší tvorbu nových sensorů a aktorů. Nejjednodušší implementace abstrakce spočívá v náhledu na všechny akce jako na zápis a čtení z polí. Například čtení ze sensorů může být simulováno jako čtení z pole nebo emulace karty může být realizována jako zápis do pole.

Pole můžou být umístěny ve dvou druzích paměti: perzistentním a neperzistentním. Pokud zapíšeme do pole v neperzistentním úložišti, tak se po resetu zařízení zapsané hodnoty ztratí. Pokud potřebujeme zachovat hodnoty polí i po resetu, musíme pole umístit do tzv. perzistentní paměti. V tomto typu paměti budeme nejčastěji ukládat různé konfigurační hodnoty, abychom je nemuseli konfigurovat při každém resetu zařízení. Příklad takové konfigurace může být vzdálenost sensoru papíru, při které potřebujeme detekovat papír.

6.7.1 Typy polí

Pro abstrakcí bude potřeba několik typů polí, které budou použitelné pro čtení dat ze sensorů, ovládání aktorů, či jejich konfigurace. Bude možné používat tři typy polí:

- Pole pouze pro čtení
- Pole pouze pro zápis
- Pole pro zápis i čtení

Pole pouze pro čtení může být využito při několika různých případech. Prvním případem samozřejmě bude již zmíněné čtení naměřené hodnoty ze sensorů, dalším použitím může být například čtení konfigurace nebo informace o zařízení, která byla zapsána při zápisu firmwaru. V konkrétní implementaci sensorů ovšem budou nalezeny další způsoby použití tohoto typu pole.

Pole sloužící pouze pro zápis může být použito například v situacích, kdy potřebujeme vyvolat určitou akci (např. emulaci karty). Tato akce může být vyvolána pouze s jedním argumentem o velikosti zapisovaného pole. Tento argument se bude hodit i u emulace karty, kde parametrem bude identifikátor emulované karty. Limitem zápisu je jeden parametr. Toto lze obejít dvěma způsoby. Prvním je rozdělení argumentů do struktury a zapsání více argumentů do jednoho pole. Druhým možným přístupem bude přidání dalšího konfiguračního pole, do kterého nahrajeme další parametr.

Posledním typem polí je pole pro zápis i čtení, to bude nejčastěji sloužit pro konfiguraci zařízení jako například časový interval, po který se bude daná karta emulovat. Pole nicméně může sloužit i pro další účely závislé na konkrétní implementaci daného sensoru nebo aktoru.

6.8 Návrh protokolu

Jedním z klíčových úkolu pro umožnění komunikace se sensory je návrh protokolu nad protokolem CAN. Pro komunikaci byl vybrán Extended CAN z důvodu většího identifikátoru zprávy. Datová pole využitelná pro přenos dat jsou následující: 29bitový identifikátor a 8bytové pole pro data. Ostatní pole jsou specifická pouze pro CAN a nelze je využít pro přenos dat (více informací 3.2.4).

Pole identifikátoru bude využito pro identifikaci typu zprávy, adresaci sensorů a fragmentaci či defragmentaci zpráv. Fragmentace zpráv je v tomto případě nezbytná, protože délka dat v jednom rámci může být maximálně osm bytů. Tato délka bude ve většině případu nedostatečná, proto bude nezbytné fragmentovat zprávy delší než osm bytů. K fragmentaci bude využívano pole "Číslo rámce". Zde je konkrétní rozčlenění 29bitového identifikátoru:

Obrázek 6.3: Využití 29 bitového identifikátoru

- Typ zprávy (seřazeno podle priority):
  - Acknowledgement - potvrzení zprávy a v případě odpovědi na požadavek čtení může obsahovat požadovaná data
  - Discovery reply - odpověď na discovery zprávu
  - Discovery - žádost pro objevení všech zařízení na sběrnici
  - Notification data - asynchronní zpráva - např. překročení prahu teploty apod.
  - Read - požadavek na čtení ze zařízení

  - Write - zápis do zařízení

- Adresa - adresa zařízení na sběrnici
- Id sensoru nebo aktoru - adresa sensoru v rámci jednoho zařízení (jedno zařízení na sběrnici může mít více snímačů)
- Číslo rámce - identifikátor rámce při fragmentaci a defragmentaci zprávy
- Rezervováno - toto pole je rezervováno pro budoucí použití

Jednotlivá pole a jejich využití jsou detailně vysvětleny v následujících sekcích.

6.8.1 Typy zpráv

Typ zprávy určuje prioritu ale také význam zprávy. Typů zprávy je několik (viz výše). Pokud chceme zjistit, jaká zařízení jsou připojena na sběrnici CAN, vyšleme zprávu typu discovery. Pokud zařízení na síti zachytí tuto zprávu, tak odpoví zprávou discovery reply s informací o typu sensoru nebo aktoru.

Dalším typem zprávy je read a write. Zmíněné typy zde byly zavedeny z důvodu využití abstrakce pole. Zprávy můžou sloužit pro konfiguraci, pro vyslání příkazu aktoru nebo naopak pro vyčtení hodnoty sensoru na vyžádání. Tyto dva typy vyžadují potvrzení zprávy pomocí acknowledgement zprávy, ovšem v případě acknowledgementu požadavku na čtení může odpověď obsahovat i data, pokud operace proběhla úspěšně. Pokud acknowledgement nedorazí, server ví, že nastala nějaká chyba a může se pokusit poslat požadavek znovu.

Notification data zpráva bude využívána v případech, kdy sensor detekuje nějakou událost. Například pokud snímač detekuje papír, tak odešle tento typ zprávy s časovou délkou průjezdu papíru. Tento typ zprávy se bude obvzláště hodit pro detekci změny testovaného systému.

6.8.2 Adresování

Z důvodu výskytu více zařízení na jedné sběrnici CAN je nutné závest adresování sensorů a aktorů. Každé připojené zařízení musí mít svoji unikátní adresu v rámci jedné sběrnice. Díky této adrese lze poslat zprávu výlučně jednomu fyzickému zařízení na sběrnici, ostatní zařízení budou tuto zprávu ignorovat. Tato adresa je 11bitová, takže teoreticky může být na sběrnici připojeno 2^{11} zařízení, avšak prakticky to nebude využitelné ani realizovatelné. Rozsah této adresy byl zvolen tak velký z jednoho prostého důvodu. A to, aby každý typ zařízení měl svůj vlastní rozsah adres. To znamená, že pokud budeme počítat se 128 druhy zařízení, tak vyjde $\frac{2^{11}}{128} = 16$. To znamená, že můžeme adresovat na sběrnici 16 zařízení stejného typu. Tímto docílíme minimálního překryvu adres a minimalizaci konfliktů v komunikaci.

Další hodnotou používanou při adresaci je id sensoru nebo aktoru. Ke každému zařízení připojeného na sběrnici může být připojeno více sensorů nebo aktorů (například více snímačů papíru). Tato hodnota jednoznačně určuje číslo prvku přípojenému k zařízení.

6.8.3 Fragmentace a defragmentace

Fragmentace zprávy bude nutná při odesílání zprávy delší než osm bytů. Pokud bude zpráva delší, tak se zpráva fragmentuje na $\frac{\text{délka dat}+7}{8}$ rámců, protože do CAN rámce se vejdou pouze osm bytů dlouhá data. První rámec bude mít pole Číslo rámce vyplněno hodnotou

celkový počet rámců − 1, každý následující bude mít hodnotu dekrementovanou o jedničku, to znamená, že poslední rámec bude mít hodnotu nula. Pole Číslo rámce je 8bitové. To znamená, že maximální možná velikost zprávy může být $8 \cdot 2^8 = 2048$ bytů. Takováto délka zprávy bude využita pouze v extrémních případ nebo nebude využita vůbec. Většinou bude délka zprávy maximálně několik desítek bytů, avšak v budoucnu může tato délka využitelná například pro aktualizaci firmwaru.

Defragmentace je opačnou operací než fragmentace. Zařízení, které přijímá fragmentovanou zprávu, nejdříve příjme zprávu s číslem rámce celkový počet rámců − 1. Zařízení tedy ví, kolik má ještě očekávat rámců. Pokud neobdrží všechny rámce, tak po určité době by mělo zařízení signalizovat chybu. Poté co je úspěšně přijata celá zpráva, zařízení vyšle zprávu typu Acknowledgement.

6.9 Návrh sensorů na papír

Při testování správné funkcionality tiskárny je zásadní ověřit, zda se po zaslání tiskové úlohy opravdu vytiskl daný dokument. Proto musí být navržen sensor detekující informaci o tom, zda projel vytisknutý papír. Ten bude mít za úkol sledovat stav výstupní přihrádky tiskárny a pokud detekuje papír, tak by měl poslat serveru, jak dlouho papír projížděl kolem sensoru. Z této informace se dá určit velikost papíru.

Nejjednodušší možností sledování tisku papírů je pomocí infračerveného sensoru. Ten obsahuje IR led a IR fotocitlivý tranzistor (přijímač). Dioda pořád svítí a pokud se sensor namíří na odrazivý materiál, tak se světlo odrazí a dopadne na fotocitlivý tranzistor. Díky tomu, že je papír bílý, tak má vysokou odrazivost, proto je infračervený sensor jednoduše použitelnou volbou.

Další alternativou pro detekci papíru je použití snímače VL6180x (popsaný v sekci 4.0.4), který byl použit v konečném návrhu. Tento snímač měří vzdálenost a ne pouze informaci o tom zda se nachází objekt před snímačem (jako u IR optočlenu). Díky tomu, že měří vzdálenost, tak může být sensor použit i pro jiné účely než jen pro detekci průjezdu papíru. Například i pro určení přibližného stavu papíru v zásobníku.

Pro vyčítání dat ze snímače VL6180x bude mít sensor jedno pole pouze pro čtení, ze kterého lze vyčítat aktuální vzdálenost objektu od sensoru. Avšak bylo by neefektivní se pořád dotazovat sensoru skrze CAN na vzdálenost a na základě toho určovat, zda projíždí papír před sensorem, takže se využije notifikační zpráva (více o typech zpráv 6.8.1). Skrze tento typ zprávy lze informovat o asynchronní události, která nastala (v tomto případě průjezd papíru). Zásluhou tohoto přístupu se ušetří šířka pásma sběrnice CAN.

V zařízení tedy bude smyčka kontrolující stav snímačů a pokud zaznamená průjezd papíru, tak odešle zprávu s informací o časové délce této události. Server na základě toho určí velikost projetého papíru. A klient může skrze REST API zjistit, zda papír úspěšně projel.

Protože je klíčové určovat, zda tiskárna vytiskla papír oboustranně nebo jenom jednostranně, je nutné, aby bylo možné k jednomu zařízení připojit minimálně dva snímače VL6180x. Pro ušetření konektorů bude vhodné, pokud bude možné snímače řetězit za sebou, a tím pádem ušetřit místo na DPS. Toto bude možné díky tomu, že snímač má pin, kterým ho lze deaktivovat. Pro ovládání pinu bude vhodné využít klopného obvodu typu D, díky kterému si nastavíme jednoduše bitovým posunem logickou jedničku na snímači, ze kterého chceme vyčítat data. Proto bude nutné mít na univerzální desce plošných spojů konektor jak s komunikačními signály SCK, SCL, tak i dva další vodiče - jeden pro vysílání

pulzů pro překlopení vstupu klopného obvodu na výstup a druhý pro vstupní signál pro klopný obvod.

6.10 Návrh emulátoru karet

Nezbytné při testování velkých multifunkčních tiskáren, které se používají ve firmách, je i kontrola správnosti a funkčnosti autentizace osoby, která chce pracovat s tiskárnou. Pokud obsluha potřebuje například vytisknout papír, nejdříve musí přiložit svoji kartu. Tiskárna ověří, zda může osoba manipulovat s tiskárnou nebo zda má dostatečný kredit na tisk dokumentů, až poté může uživatel tiskárnu využívat.

Pro otestování správné funkčnosti autentizace je možné využít jednoduchého přístupu. Na motor upevníme kartu, která má přístup k tiskárně. Kartou pak lze pohybovat před čtečkou karet na vyžádání. Tento přístup je jednoduchý na výrobu, ale má však řadu nevýhod. Například nelze automatizovaně testovat více karet bez ruční výměny karty na motoru.

Díky těmto nevýhodám bude výhodnější využít jiného přístupu, a tím je emulace karty pomocí specializovaného integrovaného obvodu. Nutnou podmínkou pro výběr čipu je možnost emulovat karty s různými identifikátory. Tuto vlastnost podporuje obvod PN532 navržený firmou NXP, která je lídrem v oblasti RFID technologií. Integrovaný obvod PN532 byl také vybrán z důvodu jeho ceny a dostupnosti hotových modulů obsahující i anténu. To odstraňuje nároky na návrh citlivé analogové části.

Úkolem emulátoru tedy bude emulovat kartu s daným identifikátorem. Před samotnou emulací se ale musí zaregistrovat číslo karty v ověřovacím systému a poté lze testovat, zda byl přístup udělen se správným identifikátorem nebo zda byl zamítnut pokud emulujeme kartu s nevalidním číslem.

Úkolem mikroprocesoru bude pouze přijímat příkazy a poté započít samotnou emulaci karty. Díky tomu bude mít pouze jedno zapisovatelné pole o velikosti 8 bytů, které bude obsahovat identifikátor karty.

Kapitola 7

Realizace

Práce zahrnuje jak softwarovou a firmwarovou implementaci, tak i vytvoření prototypů hardwarových částí sensorového systému.

Protože se systém skládá i z hardwarových částí, tak byl navržen prototyp CAN převodníku, emulátoru karty i sensoru papíru. Veškerá schémata i návrhy desek plošných spojů byly vytvořeny v programu CircuitMaker od společnosti Altium.

Pro implementaci firmwaru do mikroprocesorů byl zvolen jazyk C s využitím operačního systému FreeRTOS. Díky použití proudové paměti se tak značně zjednodušuje obsluha příjímání a odesílání dat ze sběrnice USB a CAN.

7.1 Server

Aplikace serveru byla vyvinuta v jazyce C# v běhovém prostředí .NET Core s využitím softwarové podpory pro vývoj webových aplikací ASP.NET Core. Díky tomuto frameworku se jednoduše implementovala část REST API, která vystavuje funkce všech připojených sensorů a aktorů.

Server se skládá z několika částí zajišťujících rozdílné úkoly. Nejdříve se musí zajistit správná komunikace s převodníkem CAN na USB. Toho se docílilo použitím knihovny HIDAPI. Tato knihovna ovšem byla napsána v jazyce C, takže bylo potřeba napsat obalující vrstvu, která bude zajišťovat propagaci volání metod z C# na volání nativních funkcí. Po vytvoření obalové vrstvy bude potřeba implementovat komunikaci s převodníkem. Díky tomu budeme moci již plně komunikovat se zařízeními na sběrnici CAN.

Dále je zde implementovaný protokol (popsán v sekci 6.8) pro umožnění adresace sensorů a dalších důležitých vlastností, což zajistí pohodlnější další integraci sensorů.

Po ovládání sensorů a aktorů je nutné nějakým způsobem prezentovat jejich funkcionalitu, proto bylo využito přístupu REST API. REST API se implementovalo pomocí ASP.NET Core frameworku. Díky tomuto frameworku se jednoduše zpropagovaly požadavky klienta na dotazy konkrétním sensorům. Každý druh sensoru zde má vlastní kontrolér starající se o přeposílání požadavků nebo prezentaci nasbíraných dat. Každý kontrolér má ovšem další sadu metod pro získání dostupných sensorů. Lze se dotázat na všechny zařízení daného typu nebo pouze zařízení s daným identifikátorem.

Server hostuje i stránku vytvořenou pomocí knihovny Swagger, která umožňuje jednoduché posílání REST API požadavků. Toho bylo s výhodou využito při debugování, ale i při následném nasazení pro zjišťování dostupných sensorů nebo pro přehledné zobrazení podporované funkcionality kolegům využívající testovací platformu.

7.2 Převodník CAN na USB

Převodník neobsahuje žádnou složitou logiku, skládá se pouze z procesoru, CAN transceiveru, konektorů a dalších běžných součástek nutných pro napájení, běh, programování mikroprocesoru, atd.

Jak je vidět na části schématu převodníku v příloze, tak zapojení mikroprocesoru řídící převodník je jednoduché. Využívají se jen piny CAN RX, CAN TX připojené na CAN transceiver IFX1050GVIO, který vysílá data na sběrnici. Dále jsou využity piny D+ a D- pro komunikaci na USB. Pro programování nebo debugování mohou být využité piny SWDIO, SWCLK, RESET nebo lze na pin BOOT0 přivést logickou jedničku a připojit desku převodníku přes USB do počítače. Poté se spustí zavaděč v procesoru, přes který lze pohodlně programovat firmware. Pin BOOT0 lze ovládat pomocí tlačítka na desce plošných spojů.

Na desku byl přidán i 16MHz krystal starající se o dodání přesných hodin do mikroprocesoru. Tento krystal je nutný hlavně z důvodu použití USB. Pro časování této sběrnice je totiž potřeba přesné frekvence 48 MHz, která je získána následným vynásobením frekvence pomocí PLL.

Převodník se stará pouze o propagaci dat na sběrnici CAN a naopak. Takže transformuje rámce přijaté z USB od serveru na rámce, které je možné odeslat přes CAN. Inverzní transformace je využívána pro posílání dat ze sběrnice CAN do serveru přes USB.

7.3 Univerzální deska pro sensory a aktory

Tato deska by měla postačit pro vývoj běžných sensorů nebo aktorů. Zatím ovšem bude využívána pouze pro sensory papíru, a podle toho byla uzpůsobena. Pro tento účel byl vyveden konektor P3. Zde je vyvedená I^2C sběrnice, ale i piny CLK a D, které můžou být využity pro adresaci snímačů, pokud jich bude více na jedné sběrnici. Dále je zde konektor P4 s vyvedenou SPI sběrnicí a konektor P5, na kterém jsou dva piny v 5V logických úrovní, převedené převodníkem TXB0102 od firmy Texas Instruments. Pro zajištění ESD odolnosti zde byly přidány ESD diody D3 a D4.

Pro obsluhu snímačů a komunikace byl vybrán mikroprocesor STM32L432KC z důvodu jeho nízké ceny a dostatku periférií pro řízení většiny sensorů nebo aktorů.

7.4 Vytváření nových sensorů a aktorů

Při vývoji platformy byl kladen důraz na snadnou rozšiřitelnost o nové sensory nebo aktory, proto byl vytvořen jednoduchý modulární systém, kde se definují vlastnosti daného zařízení, jako je typ zařízení, počet endpointů a jakým způsobem jsou tyto endpointy obsluhovány. Proto na začátku musí být definován tzv. dispečer, který obsluhuje požadavky adresované danému zařízení. Tento dispečer je v podstatě struktura obsahující několik ukazatelů na funkce, které jsou zavolány pokud dojde k nějaké události. Při startu zařízení je zavolána funkce SetupCallback, v této funkci může být např. inicializace sensorů nebo alokace perzistentní paměti. Dále je zde WriteRequestCallback, který je závolán pokud dojde požadavek k zápisu nebo ReadRequestCallback pokud naopak byl přijat požadavek ke čtení.

Dispečer dále obsahuje pole Memory obsahující popis polí do kterých lze zapisovat, nebo z nich lze číst. Tento popis je generovaný pomocí maker. Nejdříve musí být zavoláno makro START_MEMORY_LAYOUT, po ukončení definování popisu musí následovat

makro END_MEMORY_LAYOUT. Obě makra musí obsahovat argument značící jméno popisu paměti. Mezi těmito makry můžeme definovat jednotlivá pole. Existují dva typy: normální pole, pole kolekce. Normální pole obsahuje pouze jednu položku daného typu a lze ho definovat pomocí makra ADD_FIELD. Pole kolekce může obsahovat několik položek zadaného typu a definuje se makrem ADD_COLLECTION_FIELD. Tento typ pole musí mít pevnou délku kolekce.

Tato sada maker vytvoří strukturu s jednotlivými prvky o dané délce. Struktura bude přímo využívaná pro zápis a čtení pole. Protože je nutné kontrolovat oprávněný přístup k poli, tak je nutné vygenerovat i popis obsahující typ (pouze pro čtení, pouze pro zápis, obě operace povoleny), číslo, začátek a velikost pole. Všechna tato pole jsou zkontrolována po přijetí požadavku a pokud došlo k neoprávněnému přístupu k poli, tak se operace ukončí. Tento popis je vygenerován s makry s předponou DEF_, tedy například makro DEF_START_MEMORY_LAYOUT pro definování začátku definice paměti.

Po vytvoření popisu pole, je nutné vytvořit pomocí makra DEFINE_MEMORY popis paměti jako celku. Tímto makrem můžeme popsat typ paměti (perzistentní, neperzistentní) nebo paměť která následuje za touto pamětí. Tímto můžeme jednoduše řetězit paměti za sebe. Aby tato paměť byla viditelná, musí se přiřadit její ukazatel do struktury dispečera.

Při příchozím požadavku firmware rozpozná dispečera, který by měl obdržet danou zprávu. Pomocí popisu polí paměti zjistí, zda dané pole existuje a zda nad ním chce uživatel vykonat podporovanou akci. Pokud je požadavek validní, tak u čtení zavolá nejdříve ReadRequestCallback, který může naplnit definovanou strukturu, a poté vrátí data uživateli. U zápisu nejdříve naplní strukturu a až poté zavolá WriteRequestCallback, který může vyvolat danou akci.

Po vytvoření firmwaru pro mikroprocesor je ovšem ještě nutné vytvořit ASP.NET Core kontrolér, který bude obsluhovat požadavky od klienta. Logiku sensoru lze implementovat jak ve firmwaru, tak i na serveru, oba přístupy mají svoje výhody i nevýhody. Po úpravě serveru lze již plně využívat funkcionalitu daného sensoru.

7.5 Sensory papíru

Sensor se snímačem VL6180x obsahuje celkem 6 součástek: 2 konektory, regulátor 2,8 V, klopný obvod typu D, převodník napěťové úrovně I^2C z 3,3 V na 2,8 V a samotný snímač VL6180x. Dva konektory J1 a J2 jsou identicky zapojeny jako konektor P3 na univerzální DPS. Je zde I^2C sběrnice a signály pro klopný obvod D. Signál D IN ovládá vstupní pin klopného obvodu a signál CLK na náběžnou hranu překlápí logickou úroveň z pinu D na Q. Pin Q je připojen na čip VL6180x (tímto signálem ho můžeme deaktivovat) a na výstupní konektor, ostatní piny jsou zapojeny identicky se vstupním konektorem. Pokud za sebe zřetězíme více snímačů, vznikne tak registr o velikosti počtu snímačů a piny D a CLK můžeme ovládat, která zařízení na sběrnici jsou aktivní.

Tento snímač se může zapojit do univerzální desky přes 6pinový konektor. Poté může firmware vyčítat aktuálně naměřené vzdálenosti. Pokud se detekuje papír před snímačem, tak se pošle asynchronní zpráva s časovým intervalem délky průjezdu papíru. Z tohoto intervalu server určí velikost papíru na základě pevně daných časových konstant.

Pro urychlení vývoje komunikace se snímačem VL6180x byla použita knihovna, kterou dodává přímo STMicroelectronics. Pouze byla portována nejnižší vrstva využívající I^2C rozhraní pro mikroprocesor STM32L432KC.

7.6 Emulátor karty

Pro emulaci karty byl vybrán čip PN532, protože existuje dostupný vývojový kit, který značně zjednodušuje vývoj a následné použití. Kit má vyvedeny 3 sběrnice: Sériový port UART, I^2C , SPI. Proto musíme na spínači na kitu navolit rozhraní, přes které budeme chtít s čipem komunikovat. Na výběr komunikačního prostředku s PN532 nejsou kladeny žádné požadavky, takže byla zvolena sběrnice SPI. Pro řízení emulace je použit čip STM32L432 podobně jako u univerzální desky.

Protože pro využití emulátoru není vhodná univerzální deska, a to hlavně z důvodu absence konektoru s roztečí 2,54 mm, tak byla navržena vlastní deska plošných spojů pro zasazení kitu s PN532. Do této desky plošných spojů lze jednoduše nasunout kit s PN532. Poté lze tuto desku s kitem připojit ke čtečce karet. A přes server lze zasílat příkazy k emulaci karty s daným identifikátorem.

Pro umožnění komunikace s čipem PN532 byla využita knihovna od firmy Seeed Technology, která ji zveřejnila na githubu. Tato knihovna umožňuje využívat PN532 jako emulátor karty, čtečku karet, ale i k peer-peer komunikaci. [20]

Kapitola 8

Závěr

Cílem této práce bylo vytvořit doplňující platformu pro testování, která umožňuje jednoduché použití a vytvoření sensorů nebo aktorů pro testování vestavěných systémů, dále také navrhnout a zhotovit jejich funkční prototyp. Byl realizován jak server obsluhující sensory a aktory, tak i konkrétní prvky, které lze využít pro testování.

Komunikace serveru se sensory a aktory probíhá po sběrnici CAN, takže je zajištěna odolnost vůči rušení, detekce chyb a detekce a zotavení se z kolize při vysílání několika zařízení naráz. Tyto vlastnosti a mnohé další zaručily snadnější implementaci protokolu nad touto sběrnicí, protože nebyla potřeba se zabývat základními požadavky nutnými pro komunikaci. Dále pro umožnění vyčítání dat ze serveru nebo posílání příkazů byl zvolen protokol HTTP s přístupem REST API implementovaného pomocí frameworku ASP.NET Core. Díky tomu je implementace obsluhy požadavků klienta přímočará, takže se při vytváření nového testovacího prvku stačí soustředit na realizaci logiky daného sensoru nebo aktoru.

Platforma byla navržena s ohledem na znovupoužitelnost a rozšiřitelnost kódu. Díky tomu lze firmware snadno portovat na jakýkoli mikroprocesor nebo jednoduše vytvořit nové sensory bez nutnosti detailní znalosti systému, a to jak na straně implementace sensoru v mikroprocesoru, tak i na straně serveru.

Navržen a sestaven byl emulátor karet a sensor průjezdu papíru. Oba prvky se už používají v praxi pro testování multifunkčních tiskáren. Při používání emulátoru karty se neprojevily žádné nedostatky, pouze u sensoru papíru by bylo vhodné mít možnost nastavování minimální vzdálenosti detekce papíru, protože u každého druhu tiskárny nelze sensory upevnit ve stejné vzdálenosti od papíru. Dalším objeveným nedostatkem je způsob rozpoznávání velikosti papíru. V současnosti se měří čas průjezdu a na základě konstant se přepočte na velikost papíru. Tento přístup znemožňuje spolehlivou detekci napříč různými výrobci. Proto bude potřeba mít databázi tiskáren a v ní uložené rychlosti průjezdu papíru.

Vytvořená platforma splňuje všechny základní požadavky, avšak v budoucnu bude potřeba doimplementovat další funkcionalitu jako třeba možnost připojení bezdrátových sensorů, zabezpečení serveru proti neoprávněnému využívání nebo vytvoření zavaděče pro možnost jednoduché aktualizace firmwaru po sběrnici CAN. Dále bude nutné navrhnout další sensory a aktory z důvodu otestování další funkcionality multifunkčních zařízení podle aktuálních požadavků. Pro ověření správné funkčnosti tiskárny již byla zjištěna nutnost implementace dalších dvou typů sensorů pro ověřování barevnosti vytištěného papíru a pro zjištění stavu kontrolek na tiskárně.

Literatura

[1] *Autofac*.
URL http://autofaccn.readthedocs.io/en/v4.0.0/

[2] FreeRTOS Version 10.
URL https://www.freertos.org/FreeRTOS-V10.html

[3] Swagger.
URL https://swagger.io/

[4] ARM: *Cortex-M4 processors*.
URL http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0553a/CIHIGCIF.html

[5] CIA, C.: History of CAN technology. *Can CIA*.
URL https://www.can-cia.org/can-knowledge/can/can-history/

[6] Daniel Roth, a. S. L., Rick Anderson: Introduction to ASP.NET Core. *Microsoft documentation*, 2018.
URL https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.1

[7] Fielding, R. T.: *Architectural Styles and the Design of Network-based Software Architectures*. Diplomová práce, UNIVERSITY OF CALIFORNIA, IRVINE, 2000.

[8] FreeRTOS: RTOS Stream & Message Buffers. *FreeRTOS*, 2017.
URL https://www.freertos.org/RTOS-stream-message-buffers.html

[9] Infeon: IFX1050GVIO.
URL https://www.infineon.com/dgdl/IFX1050GVIO_DS_10.pdf?fileId=db3a304330f68606013141d8eac7569f

[10] a Mike Wasson, R. A.: Create a Web API with ASP.NET Core MVC and Visual Studio Code on Linux, macOS, and Windows.
URL
https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api

[11] NKK Switches: *Series FR01 general specifications*.
URL https://www.nkkswitches.com/pdf/FR01-2.pdf

[12] NXP Semiconductors: *PN532*.
URL https://www.nxp.com/docs/en/user-guide/141520.pdf

[13] NXP Semiconductors: *I2C-bus specification and user manual*. 2014.
URL https://www.nxp.com/docs/en/user-guide/UM10204.pdf

[14] Peacock, C.: USB in a NutShell. *Beyond Logic*.
URL https://beyondlogic.org/usbnutshell/usb1.shtml#Introduction

[15] Rodriguez, A.: RESTful Web services: The basics. *IBM*, 2008.

[16] Software, S. .: HID API for Linux, Mac OS X, and Windows. *Signal 11 Software*, 2010.
URL http://www.signal11.us/oss/hidapi/

[17] Steve Corrigan, T. I.: Introduction to the Controller Area Network (CAN).
URL http://www.ti.com/lit/an/sloa101b/sloa101b.pdf

[18] STMicroelectronics: *STM32L432KC*.
URL http://www.st.com/en/microcontrollers/stm32l432kc.html

[19] STMicroelectronics: *VL6180x*.
URL http://www.st.com/en/imaging-and-photonics-solutions/vl6180x.html

[20] Technology, S.: NFC library for Arduino. 2012.
URL https://github.com/Seeed-Studio/PN532

[21] Texas Instruments: *Serial Peripheral Interface (SPI)*.
URL http://www.ti.com/lit/ug/sprugp2a/sprugp2a.pdf

[22] Zemek, I. P.: Zaměňované pojmy v oblasti SW inženýrství. *Blog Petr Zemek*, 2010.
URL https://cs-blog.petrzemek.net/2010-01-17-zamenovane-pojmy-v-oblasti-sw-inzenyrstvi

Seznam příloh

Příloha A

Převodník CAN na USB

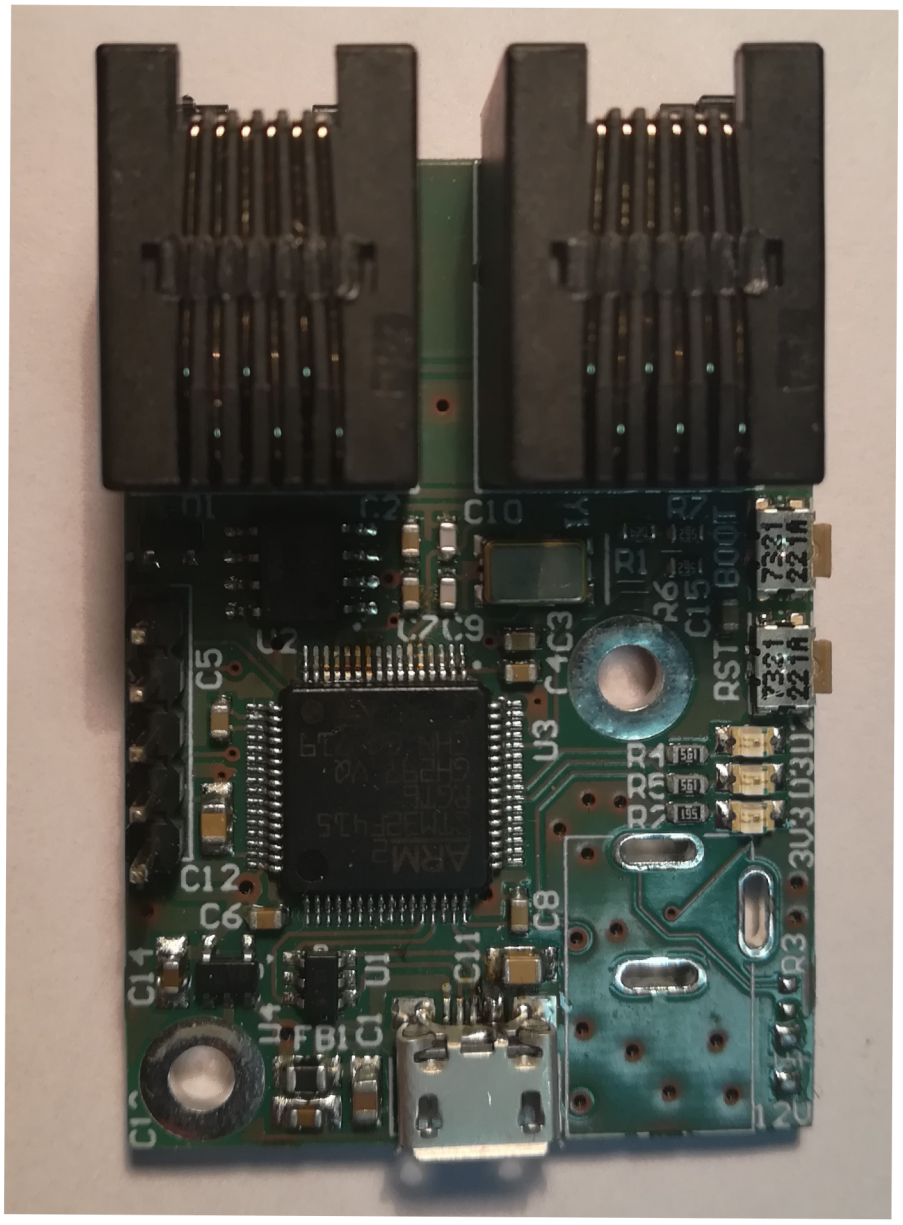


Schéma část první

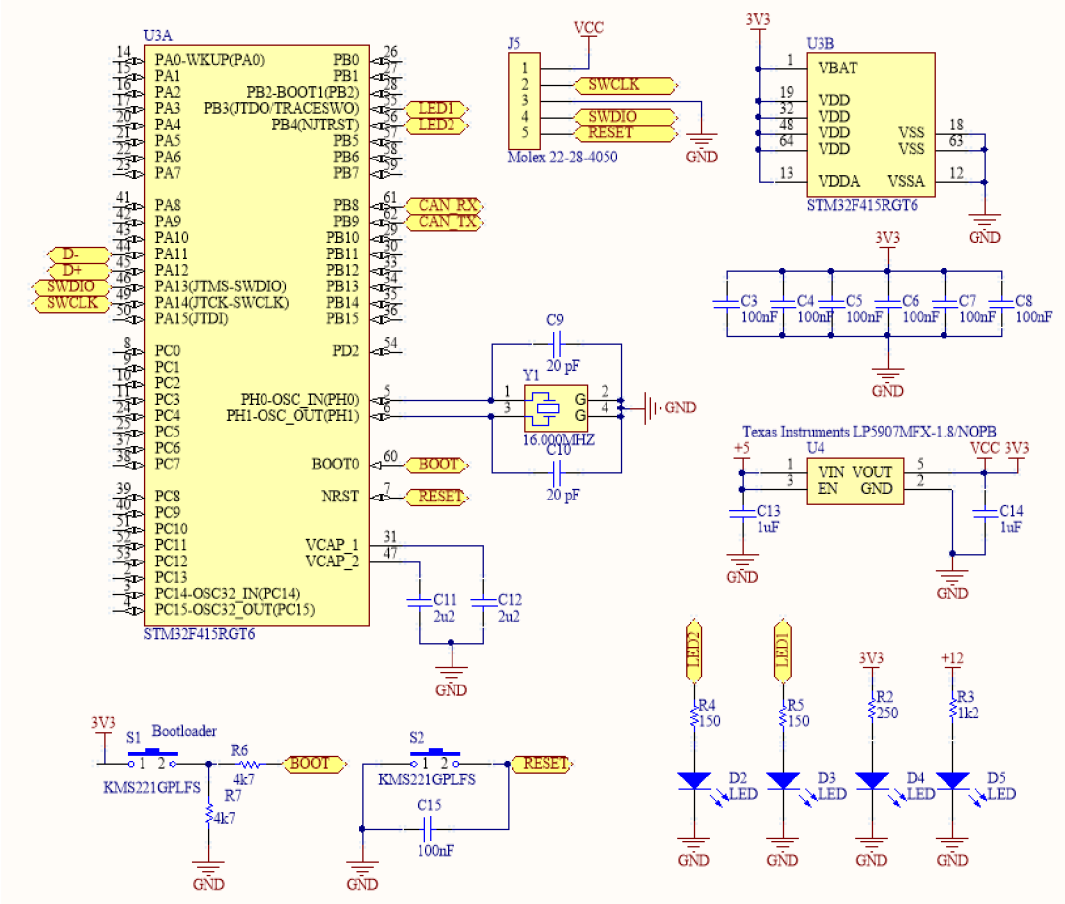
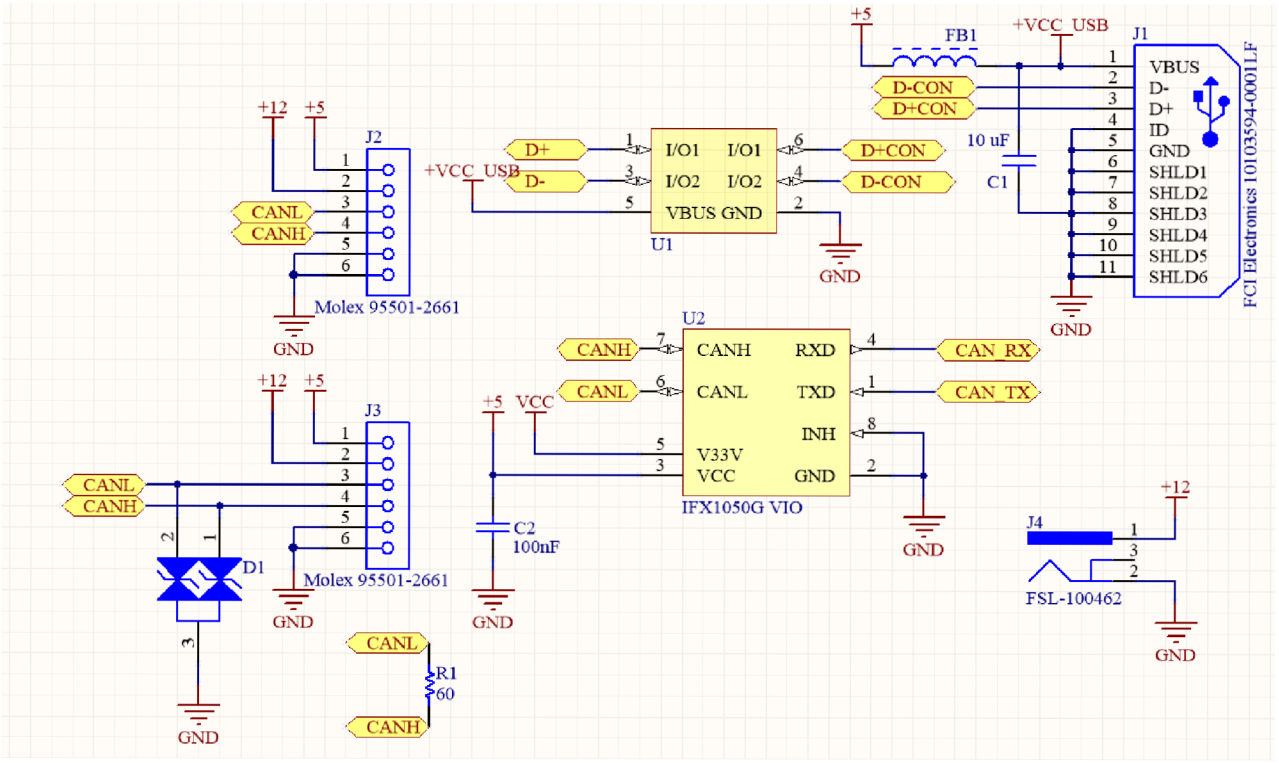


Schéma část druhá



Příloha B

Univerzální deska plošných spojů pro sensory nebo aktory

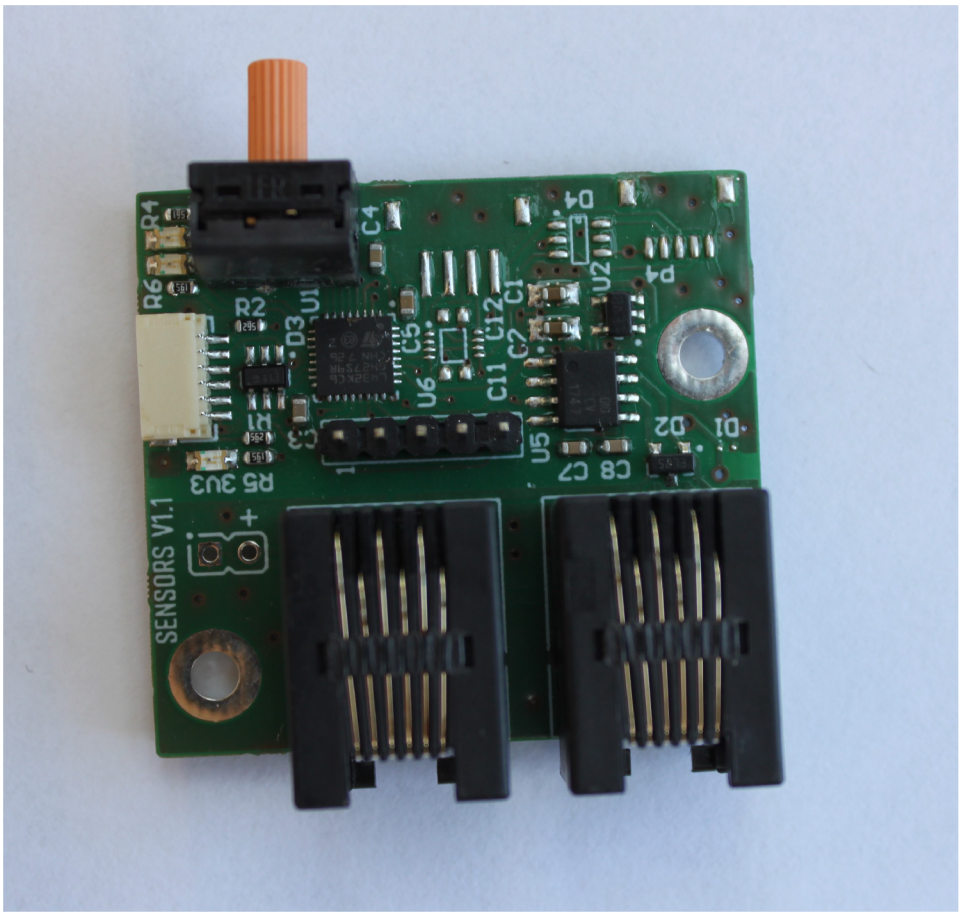


Schéma část první

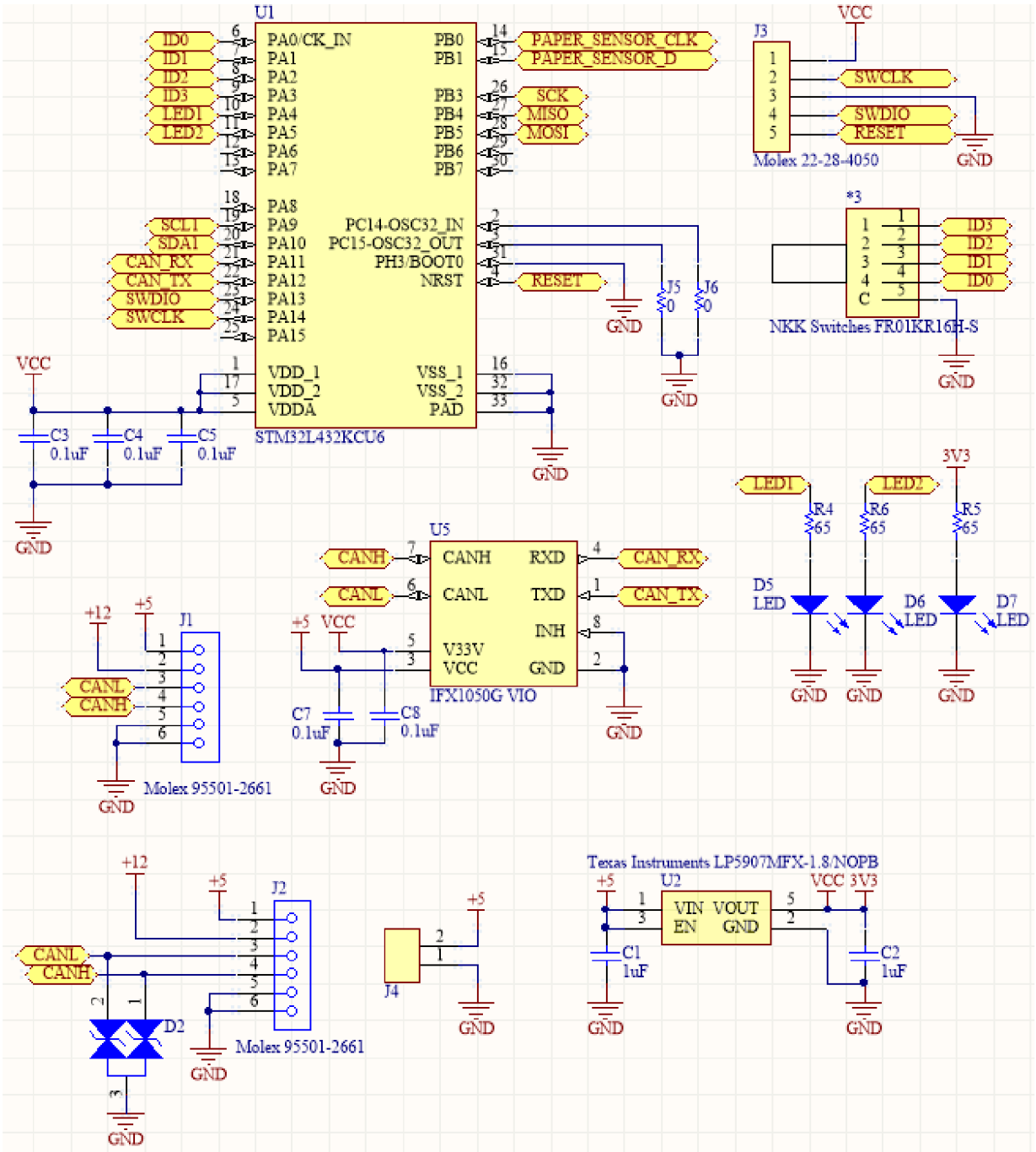
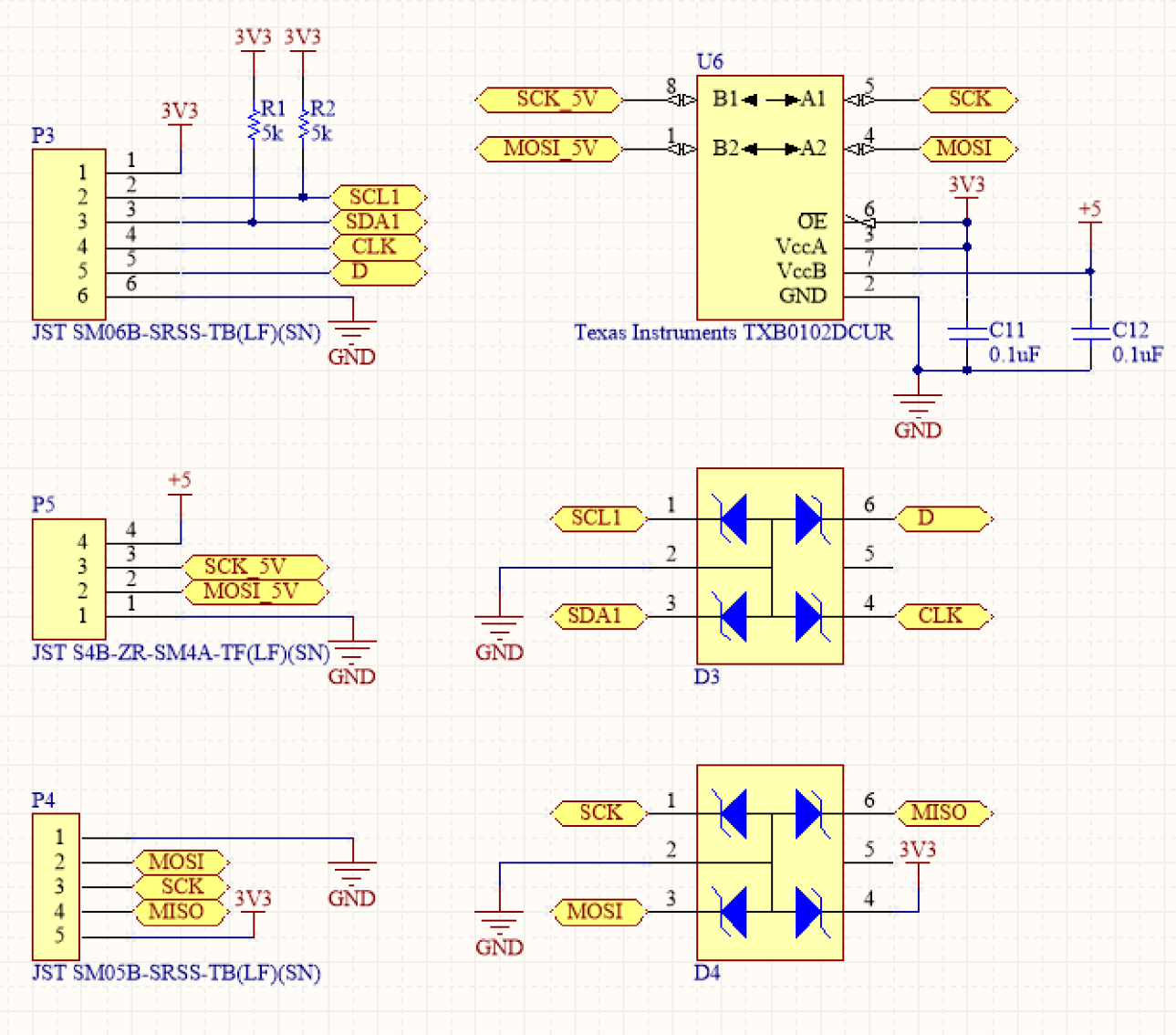


Schéma část druhá



Příloha C

Emulátor karet

Foto emulátoru karet bez nasazeného kitu PN532

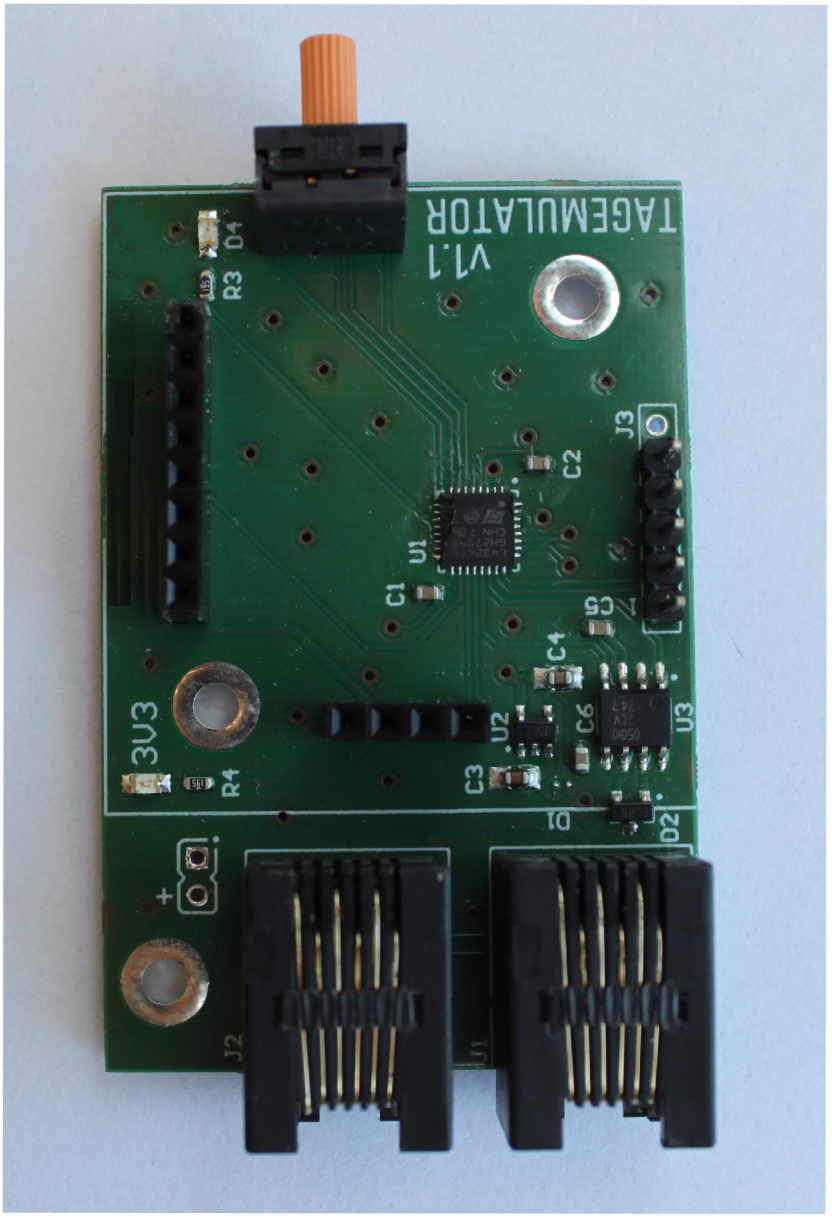
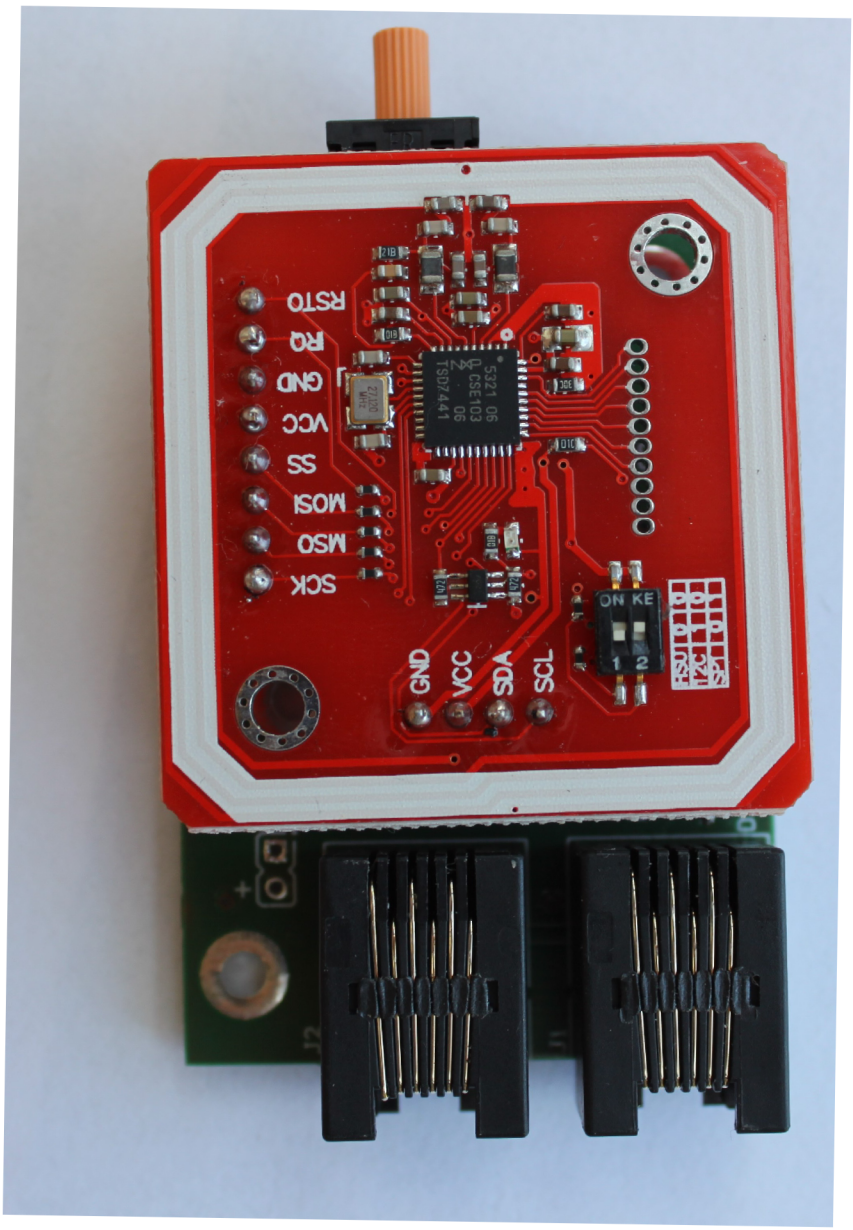
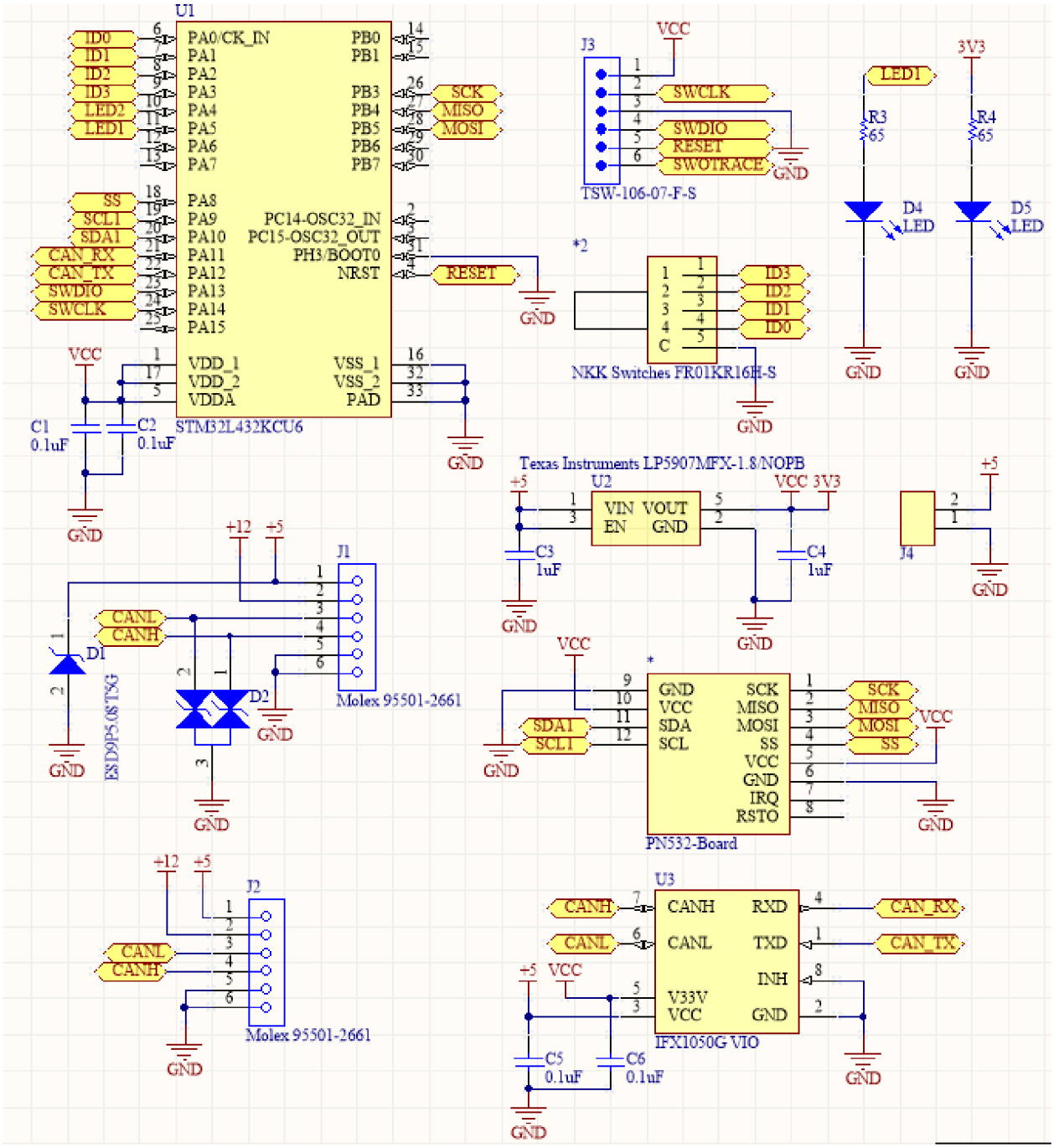


Foto emulátoru karet s kitem PN532



Schéma


U1
PA0/CK_IN
PA1
PA2
PA3
PA4
PA5
PA6
PA7
PA8
PA9
PA10
PA11
PA12
PA13
PA14
PA15
PB0
PB1
PB3
PB4
PB5
PB6
PB7
PC14-OSC32_IN
PC15-OSC32_OUT
PH3/BOOT0
NRST
VDD_1
VDD_2
VDDA
VSS_1
VSS_2
PAD
STM32L432KCU6
ID0
ID1
ID2
ID3
LED2
LED1
SS
SCL1
SDA1
CAN_RX
CAN_TX
SWDIO
SWCLK
SCK
MISO
MOSI
RESET
C1
0.1uF
C2
0.1uF
VCC
GND
J3
TSW-106-07-F-S
SWCLK
SWDIO
RESET
SWOTRACE
*2
NKK Switches FR01KR16H-S
3V3
R3
65
R4
65
D4
LED
D5
LED
Texas Instruments LP5907MFX-1.8/NOPB
U2
VIN
VOUT
EN
GND
C3
1uF
C4
1uF
J4
+5
+12
J1
Molex 95501-2661
CANL
CANH
D1
ESD9P5.0ST5G
D2
J2
Molex 95501-2661
*
PN532-Board
SDA1
SCL1
SCK
MISO
MOSI
SS
IRQ
RSTO
U3
IFX1050G VIO
CANH
CANL
RXD
TXD
INH
V33V
C5
0.1uF
C6
0.1uF
CAN_RX
CAN_TX

Příloha D

Sensory papíru

Foto DPS s osazeným snímačem VL6180x zespodu

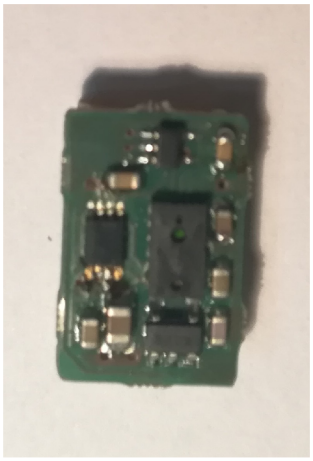
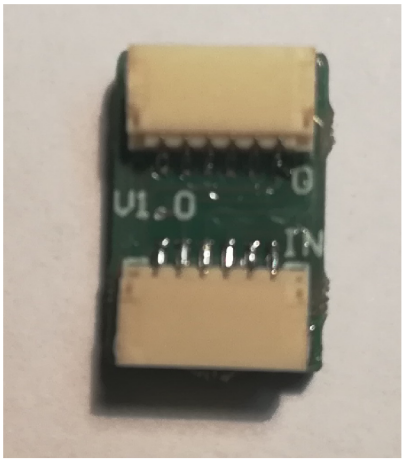
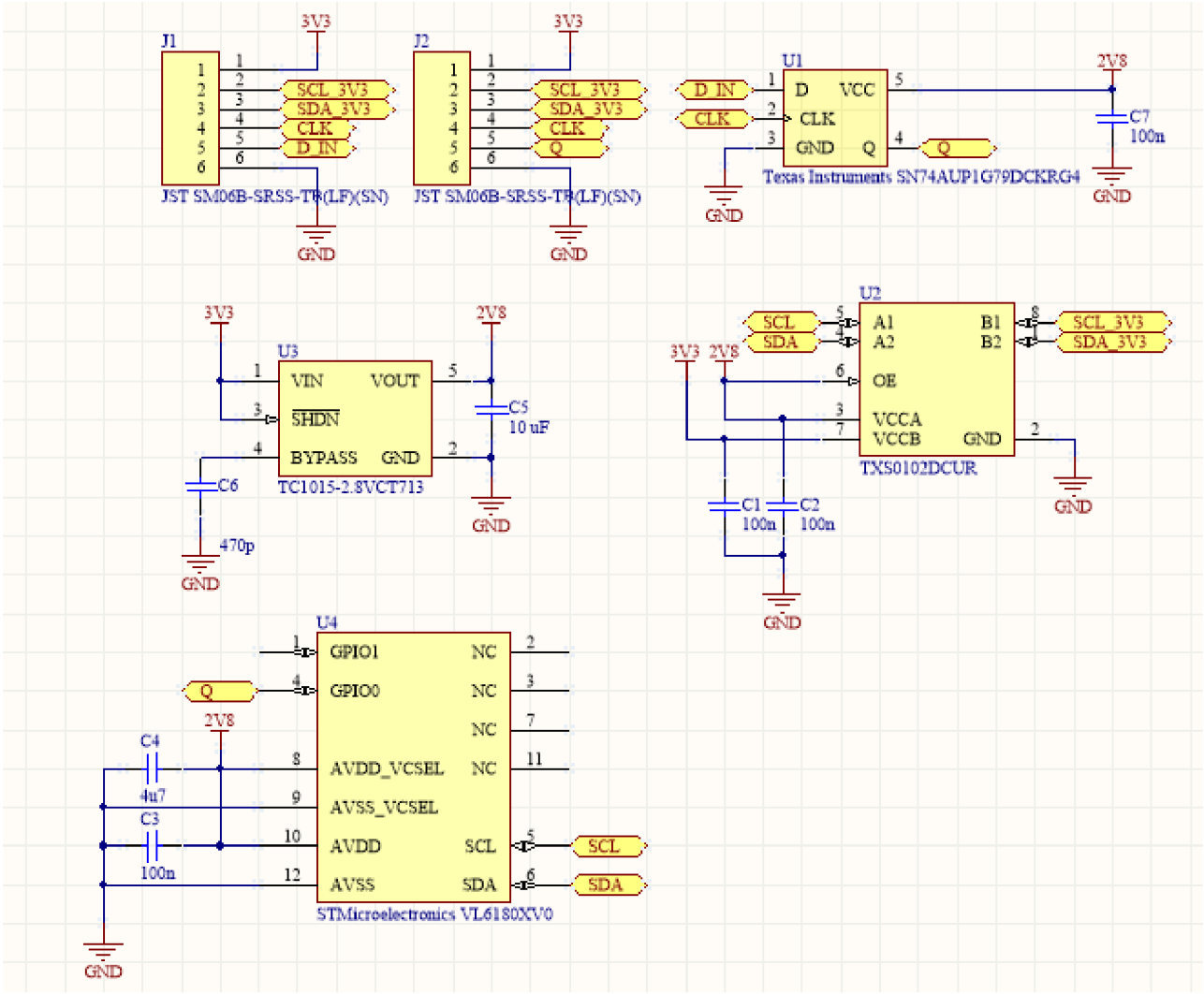


Foto DPS s osazeným snímačem VL6180x zvrchu



Schéma


J1
3V3
SCL_3V3
SDA_3V3
CLK
D_IN
GND
JST SM06B-SRSS-TR(LF)(SN)
J2
3V3
SCL_3V3
SDA_3V3
CLK
Q
GND
JST SM06B-SRSS-TR(LF)(SN)
U1
D_IN
CLK
D
VCC
CLK
GND
Q
Q
2V8
C7
100n
GND
Texas Instruments SN74AUP1G79DCKRG4
U3
3V3
2V8
VIN
VOUT
SHDN
BYPASS
GND
C5
10 uF
C6
470p
GND
TC1015-2.8VCT713
U2
SCL
SDA
A1
A2
B1
B2
SCL_3V3
SDA_3V3
OE
VCCA
VCCB
GND
TXS0102DCUR
C1
100n
C2
100n
GND
U4
Q
2V8
GPIO1
GPIO0
NC
AVDD_VCSEL
AVSS_VCSEL
AVDD
AVSS
SCL
SDA
C4
4u7
C3
100n
GND
STMicroelectronics VL6180XV0