VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

INTELIGENTNÍ PŘÍSTUPOVÝ SYSTÉM S INTEGRACÍ DO PROSTŘEDÍ HOME ASSISTANT

INTELLIGENT ACCESS SYSTEM WITH INTEGRATION TO HOME ASSISTANT FRAMEWORK

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **DAVID PODESZWA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. VÁCLAV ŠIMEK**
SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

164717

Ústav:	Ústav počítačových systémů (UPSY)
Student:	**Podeszwa David, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Vestavěné systémy
Název:	**Inteligentní přístupový systém s integrací do prostředí Home Assistant**
Kategorie:	Vestavěné systémy
Akademický rok:	2024/25

Zadání:

1. Zabývejte se problematikou napájení zařízení umístěných ve venkovním prostředí. Blíže se zaměřte především na drátové technologie jako například PoE.
2. Prostudujte způsoby přenosu audio a video záznamu po síti. Zaměřte se na protokoly, které dosahují vysoké komprese s nízkým využitím výpočetních zdrojů.
3. Analyzujte možnosti autentizace uživatelů pro automatické otevření vstupních dveří bez nutnosti zadání přístupového kódu. Zhodnoťte výhody a nedostatky jednotlivých přístupů.
4. Na základě zjištěných informací navrhněte koncepci dálkově napájeného zvonku vybaveného kamerou, možností vzdáleného otevírání brány a funkcí manuální identifikace.
5. Zvolte vhodné technologie a komponenty pro realizaci kamerového zvonku dle koncepce z bodu 4) zadání. Následně proveďte realizaci na úrovni desky plošných spojů.
6. K prvkům realizovaným v bodu 5) zadání implementujte obslužný firmware s možností konfigurace za pomoci webové administrace.
7. Implementujte integraci pro open-source platformu HomeAssistant umožňující ovládání navrženého zvonku.
8. Proveďte praktické ověření funkčnosti a zhodnoťte dosažené výsledky.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:
Splnění bodů 1 až 4 zadání.

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

Vedoucí práce:	**Šimek Václav, Ing.**
Vedoucí ústavu:	Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání:	1.11.2024
Termín pro odevzdání:	21.5.2025
Datum schválení:	31.10.2024

Abstrakt

Tato práce se zabývá návrhem a realizací inteligentního přístupového systému s kamerou. Cílem je vytvořit dálkově napájený zvonek s kamerou, který umožní vzdálené otevírání a manuální identifikaci uživatelů RFID kartami. Napájení je provedeno technologií Power over Ethernet přičemž je použito právě ethernet rozhraní pro komunikaci po síti. Přenos audiovizuálních dat je zajištěn serverem protokolu RTSP a to s využitím komprimovaných kodeků jako je H.264 pro video a G.711 pro audio. Použitím elektromagnetického zámku připojenému k zařízení lze otevírat dveře či brány, a to vše díky integraci do systému chytré domácnosti HomeAssistant. Konfigurace zařízení je prováděna po síti zabudovaným webovým rozhraním. Kromě softwaru zařízení byla vytvořena taktéž deska plošných spojů využívající nový mikrokontrolér ESP32-P4.

Abstract

This work deals with the design and implementation of an intelligent access control system with a camera. The aim is to create a remotely powered doorbell with a camera that allows remote opening and manual identification of users with RFID cards. The power is supplied via Power over Ethernet technology that uses the same ethernet interface used for network communication. Audiovisual data transfer is handled using an RTSP protocol server together with compressed codecs such as H.264 for video and G.711 for audio. Using an electromagnetic lock connected to the device, doors or gates can be opened, all thanks to an integration into the HomeAssistant smart home system. Configuration of the device is done over the network using the built-in web interface. In addition to the device software, a printed circuit board has been developed using the new ESP32-P4 microcontroller.

Klíčová slova

mikrokontrolér, mikroprocesor, chytrá domácnost, zvonek, kamera, přístupový systém, HomeAssistant, RTSP, H.264, G.711, Power Over Ethernet, MQTT, ESP32, ESP32-P4, RFID

Keywords

microcontroller, microprocessor, smart home, doorbell, camera, access control system, HomeAssistant, RTSP, H.264, G.711, Power Over Ethernet, MQTT, ESP32, ESP32-P4, RFID

Citace

PODESZWA, David. *Inteligentní přístupový systém s integrací do prostředí Home Assistant.* Brno, 2025. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Václav Šimek

Inteligentní přístupový systém s integrací do prostředí Home Assistant

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Václava Šimka. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
David Podeszwa
20. května 2025

Poděkování

Rád bych na tomto místě upřímně poděkoval mému vedoucímu Ing. Václavovi Šimkovi za vedení práce a hlavně také za pomoc a rady při kompletaci výsledné desky plošného spoje.

Dále také kamarádům a rodině za psychickou a materiální pomoc bez které by nebylo možné tuto práci dokončit.

V neposlední řadě bych rád poděkoval také Fakultě informačních technologií Vysokého učení technického v Brně za poskytnutí technického zázemí.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Každá budova, ať už se jedná o rodinný dům nebo panelový dům, zpravidla disponuje nějakým způsobem, jak její obyvatele informovat o naší přítomnosti před vchodem. Typicky se tak jedná o zvonek, jehož podoba se během let velmi vyvíjela.

Co bylo v počátcích pouze generátor tónu, se však později proměnilo do jednoduchého obousměrného telefonu či přibylo možnosti vidět volajícího pomocí kamery. Posledním takovýmto vývojem je připojení těchto zařízení k síti pro integraci do chytrých domácností. Vznikla tak například možnost odpovídat na zazvonění, i když se člověk zrovna nenachází doma. Většina aktuálních zařízení na trhu však k využití těchto funkcí vyžaduje použití cloudových služeb, ve velké většině dokonce placených.

Cílem této práce je tedy vytvoření takového zvonku, jenž bude umožňovat přenos oboustranného hlasu společně s kamerovým záznamem. Protože bude toto zařízení umístěno ve venkovních prostorech, je zapotřebí, aby bylo napájeno adekvátním způsobem, ideálně tak, aby dokázalo fungovat 24 hodin denně, 7 dní v týdnu. Tou nejdůležitější součástí je však integrace do systému chytré domácnosti a to v podobě integrace do platformy Home Assistant, se schopností dálkově otevírat dveře.

Toto téma jsem si zvolil z praktických důvodů, a to neboť přesně takovéto zařízení potřebuji pro vlastní použití.

Práce má následující strukturu. Kapitola č. 2 popisuje různé způsoby napájení zařízení umístěných ve venkovních prostorech. Kapitola č. 3 pak navazuje přímo na první kapitolu a popisuje technologii Power over Ethernet. Systémy chytrých domácností a konkrétně popis systému Home Assistant s jeho vlastnostmi je v kapitole č.4. Nadále je v kapitole č. 5 popsáno, jakým způsobem je možné po síti přenášet video a audio efektivním způsobem. Návrh SW a HW zařízení je právě v kapitole 7, na níž pak navazuje samotná realizace a implementace zařízení v kapitolách 8 a 9. Finální zhodnocení a otestování zařízení je pak popsáno v kapitole 10.

Kapitola 2

Napájení zařízení v exteriéru

Tato kapitola se zaobírá možnostmi pro napájení zařízení, jenž se musí vyskytovat v prostorech mimo jednoduše dostupné zdroje elektrického proudu. Právě takovým zařízením je i zvonek s přístupem k síti.

Tyto možnosti se dají klasifikovat do tří různých kategorií. Prvním z nich je získávání energie z okolí, dále pak použití úložiště energie, a tou poslední možností přivedení elektrické energie vodiči přímo k zařízení [20].

2.1 Úložiště energie

V případech, kdy není možné dodat zařízením elektrickou energii vůbec nebo velmi zřídka, je možné použít jedno z úložišť energií, jako jsou baterie, či superkondenzátory. Energie je v nich uložena nabitím či při výrobě a využita v dobách, kdy je potřeba. Výhodou pak může být schopnost některých těchto úložišť nabíjet se za provozu.

Baterie

Pro IoT zařízení jsou baterie velice vhodným úložištěm energie, neboť mají vysokou energetickou hustotu. V případě těch nejlepších dobíjecích akumulátorů na základě technologie Li-Ion (Lithium-Iontové) se jedná o 180mWh/cm^3 a taktéž jsou schopny pokrýt krátkodobé odběrové špičky [20].

Jedná-li se však o použití v exteriérech, jejich kapacita se výrazně snižuje s klesající teplotou, a tak je jejich efektivita v zimním období snížena. Taktéž dochází k degradaci s každým nabíjecím cyklem, což omezuje životnost takového zařízení.

Alternativou může být použití jednorázových baterií, jakými jsou například NiMH (nikl-metal hybridový akumulátor), jejichž kapacita neklesá v chladných prostředích. Takovéto baterie ovšem již z principu vyžadují výměny uživatelem. [**?**]

Superkondenzátory

Použitím superkondenzátorů je možné dosáhnout životnosti v podobě více nabíjecích cyklů bez ztráty kapacity. Oproti Li-Ion článkům jsou také více odolné vůči nízkým teplotám, a jejich výkon tak není jimi ovlivněn.

Jejich energetická hustota je ovšem velmi malá, a tak jsou vhodné pouze pro zařízení s velmi nízkým odběrem. Nejlépe se tak uplatní v kombinaci s některým ze sběrů energie, jenž umožní dobíjení tohoto superkondenzátoru za chodu zařízení.

2.2 Sběr energie

Použitím z několika způsobů sběru energie, anglicky *Energy harvesting* je elektrická energie extrahována z prostředí, ve kterém se zařízení nachází, například ze slunce, vzduchu, tepla či z rádiových vln.

Solární energie

Sběr solární energie probíhající za pomoci fotovoltaických panelů je jeden z nejjednodušších, efektivních a hlavně velmi dobře prozkoumaných způsobů napájení.

Na $1m^2$ zemského povrchu dopadá zhruba 1000 wattů solární energie, což v průměru odpovídá 180-270 W/m^2 za den, počítáme-li s průměrným osvícením této plochy během celého roku. [20] Vezmeme-li v potaz efektivitu fotovoltaických panelů dostupných na trhu, jenž se pohybuje v rozmezí 15-20%, dostáváme se na 2,7-4 mW/cm^2.

Tyto panely však nefungují během noci a jejich výstup je taktéž závislý na počasí. V dobách, kdy solární panel není schopen zajistit dostatek energie pro provoz zařízení, je zapotřebí použití jednoho z výše zmíněných úložišť energií. [20]

Větrná energie

Dalším zdrojem energie může být také rotace způsobená tokem vzduchu, k čemuž se využívají větrné turbíny. Tento typ generace energie je však vysoce závislý na počasí a s použitím z jednoho z úložišť je velmi prostorově náročný. Hodí se tak spíše pro IoT (Internet of Things) senzory s velkým intervalem mezi zasíláním dat a nízkou energetickou náročností. [20]

2.3 Distribuce vodiči

Přivedením elektrických vodičů přímo k zařízení umístěnému v exteriéru není zapotřebí využít jakéhokoliv jiného zdroje napájení. Je tak zajištěna největší spolehlivost, neboť je toto napájení závislé pouze na elektrické síti.

Jednou z možností takového připojení je přivedení elektrické sítě, tj. střídavý proud 230V/50Hz přímo k venkovnímu zařízení. Ačkoliv tak odpadá nutnost umístění stejnosměrného zdroje v budově, ten musí být umístěn poblíž zařízení nebo přímo jeho součástí. Připojení vodičů k zařízení však vyžaduje přímou interakci s vodiči nízkého napětí, což dle §4 NV 194/2022 v České republice nemůže provádět osoba bez příslušné kvalifikace [1].

Alternativou tak může být umístění zdroje v budově a přivedení již stejnosměrného proudu. Vzhledem k velké vzdálenosti budovy od místa, kde má být zařízení umístěno (například na sloupu), která může dosahovat až desítek metrů, může během proudových špiček docházet k výrazným poklesům napětí na vodiči.

Tomuto jevu se dá zabránit použitím vyššího napájecího napětí, což ovšem na straně zařízení vyžaduje příslušný DC-DC měnič pro přeměnu na nižší napětí.

Variantou přenosu stejnosměrným proudem je pak využití standardu PoE (Power over Ethernet), o němž pojednává následující kapitola.

Kapitola 3

Standard Power over Ethernet

Technologie PoE, jak již lze z názvu odhadnout, slouží pro napájení zařízení pomocí kabelů, po nichž probíhá také samotná komunikace standardu Ethernet (IEEE 802.3).

Zařízení, jenž této technologie využívají, tak nepotřebují separátní vodiče pro napájení a datové spojení. Tímto způsobem je tedy ušetřeno jak na ceně, tak na práci instalace kabeláže, tak potenciálně také na zdrojích stejnosměrného napětí. [4] V typických instalacích je totiž jako zdroj pro PoE využíván síťový přepínač s podporou PoE, neboť je často instalováno těchto zařízení více najednou (např. bezpečnostní kamery a senzory, Wi-Fi přístupové body).

Délka kabeláže je dle standardu omezena na 100 m, ovšem zde se jedná pouze o zaručenou vzdálenost a prakticky lze dosáhnout vzdáleností větší v závislosti na typu UTP. kabeláže a rušení.

PoE jako takové původně vzniklo jako proprietární technologie vytvořená společností Cisco v roce 2000, kde původním záměrem bylo pouze napájení Cisco IP telefonů. Vzhledem k potenciálu této technologie však došlo později ke standardizaci institutem IEEE (Institute of Electrical and Electronics Engineers) a organizací Ethernet Alliance, čímž vznikl první standard *IEEE 802.3af*. Tyto dvě organizace se také starají o vývoj všech nových revizí tohoto standardu. [4]

3.1 Druhy zařízení

Součástí každé sítě využívající PoE musí být vždy minimálně jedno PSE (Power Sourcing Equipment) a jedno či více PD (Powered Device).

PD (Powered Device)

Powered Device, překladem *napájená zařízení* je zařízení, jež energii odebírá. Po připojení kabelem do sítě se musí adekvátně identifikovat a klasifikovat (viz sekce 3.4) a až poté může odebírat proud. [11]

PSE (Power Sourcing Equipment)

Power Sourcing Equipment (napájecí zařízení) jsou zařízení, která energii dodávají. V momentě správného připojení a detekce zařízení na druhém konci je tomuto PD zařízení přivedeno adekvátní napájecí napětí.

Napájecí zařízení se dále mohou dělit na dva typy podle toho, zda je PSE součástí aktivního síťového prvku (přepínač, směrovač), tzv. ENDSPAN, nebo zda jde o PSE umístěné mezi síťovým prvkem a PD, tzv. MIDSPAN. Příkladem tohoto MIDSPAN zařízení může být například PoE injektor. [11]

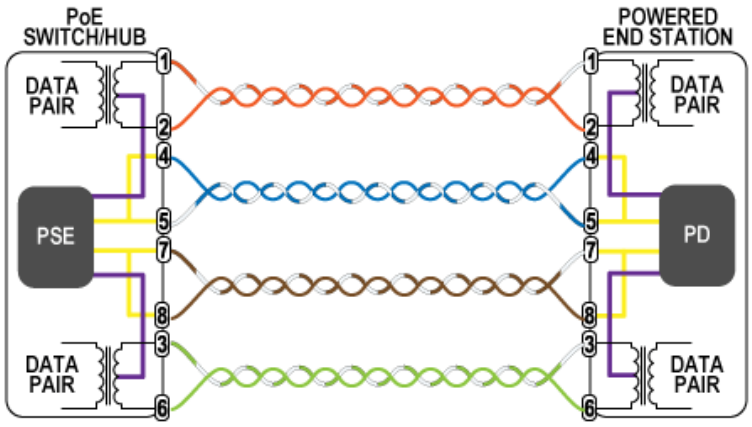
3.2 Přenos vodiči

PoE energii přenáší vždy v UTP kabelech po jednotlivých párech vodičů, tj. pozitivní napětí v jednom páru a negativní napětí v druhém. Tento přenos může být realizován ve dvou různých módech A nebo B.

V módu A jsou k přenosu použity datové páry a napětí je na páry vloženo za pomocí středové odbočky v transformátorech. Stejným způsobem je poté napětí z párů odebráno, a protože se jedná o diferenciální páry, datový signál není tímto napětím nijak ovlivněn. V tomto módu jsou tedy standardně využity páry 1-2 a 3-6, označovány Rx a Tx. [5]

Mód B k přenosu využívá pouze volné páry, jenž se nepoužívají k přenosu dat. Napětí je na tyto páry aplikováno přímo, bez použití jakýchkoliv transformátorů, neboť tyto páry nejsou jinak k ničemu jinému připojeny. Napětí je tedy přivedeno na diferenciální páry 4-5 a 7-8.

Obrázek 3.1 zobrazuje případné zapojení buď v módu A, nebo v módu B.



Obrázek 3.1: Schéma zapojení napájecího zařízení (PoE Switch) k napájenému zařízení (Powered end station).
Fialové vodiče – mód A, žluté vodiče – mód B. Převzato z [13]

V případě obou módů však nezáleží na správném zapojení polarity. UTP kabel, jenž propojuje napájecí zařízení s napájeným, totiž může být křížový, což znamená, že by mohlo po cestě kdykoliv dojít k výměně negativních a pozitivních vodičů. Z tohoto důvodu tak musí každé napájené zařízení využít usměrňovacího diodového můstku, než přivedené napětí jakkoliv využije.

Pasivní PoE

Pasivním PoE se rozumí takovým použitím PoE, kdy jsou jak napájecí, tak napájená zařízení funkční i bez jakékoliv komunikace (detekce a klasifikace). V takovéto situaci jsou napájená zařízení připojena k napětí i v případě, že se nemusí jednat o validní PoE zařízení a je využit výhradně mód B, tedy volné páry. Oficiálně se tedy nejedná o využití dodržující standard *IEEE 802.3af*.

Často takto fungují levné PoE injektory a připojení takto napájeného kabelu do jakéhokoliv zařízení, jenž není na PoE přizpůsobeno, může znamenat jeho nenávratné poškození. [19]

Aktivní PoE

Je-li použito aktivní PoE, musí vždy dojít k detekci a klasifikaci napájeného zařízení. Napájení může fungovat v obou dostupných módech A či B.

3.3 Výkonové třídy a standardy

Technologie PoE se časem vyvíjí, a tak existuje již několik standardů za účelem přenosu vyšších výkonů. Následující sekce se věnují popisu těchto standardů včetně výkonových tříd, jenž přidávají.

IEEE 802.3af

Původní standard *IEEE 802.3af* byl navržen pro maximální výkon 15,4 W, koncovému zařízení se však dostává pouze 12,95 W. Zařízení, jenž pracují pod tímto standardem, se nazývají také jako typ 1 zařízení. Tento typ se dále dělí na čtyři třídy dle maximálního dostupného výkonu, konkrétně 0-4 a popisuje je tabulka 3.2 níže. Napájené zařízení se tak může samo během klasifikace identifikovat jako jedna z těchto tříd, a omezit se tudíž na specifikovaný výkon (po překročení může ze strany PSE dojít k odpojení napájení).

Třída	**Výkon PSE**	**Výkon PD**
0	15,4 W	0,44-12,95 W
1	4 W	0,44-3,84 W
2	7 W	3,84-6,49 W
3	15,4 W	6,49-12,95 W

Tabulka 3.1: *IEEE 802.3af* Výkonové třídy typu 1 [5]

IEEE 802.3at

Protože 12,95 W je výkon, který již není dostačující pro některá zařízení, vznikl tento standard. Ten zvýšil maximální výkon na 30 W ze strany PSE resp. 25,5 W pro PD zařízení. Tato zařízení se značí jako typ 2 a vznikla tím také nová třída číslo 4 [5]. Vzhledem k takto zvýšenému výkonu a také faktu, že jsou stále využívány pouze 2 páry vodičů, došlo ke zvýšení minimálního doporučeného typu UTP kabelu na Cat.5 [21].

Třída	Výkon PSE	Výkon PD
4	30 W	25,5 W

Tabulka 3.2: *IEEE 802.3af* Výkonové třídy typu 2 [5]

IEEE 802.3bt

Poslední revize standardu vzniká roku 2018, a s označením PoE++ přidává schopnost poskytovat výkon až 90 W z napájecího zařízení (73 W pro napájené zařízení). Přibývají tak dva nové typy zařízení, typ 3 a 4, jenž se každá dále dělí na 2 výkonové třídy (5-7). K přenosu jsou nyní použity všechny 4 páry vodičů a je tedy vyžadováno užití módu A (napětí zároveň s datovými signály). Nově vzniklé třídy zobrazuje tabulka 3.3

Všechny napájecí zdroje jenž implementují jak *IEEE 802.3at* či *IEEE 802.3bt* musí být zpětně kompatibilní s nižšími standardy, a tedy umožnit napájenému zařízení provoz pokud si během klasifikace vyžádá výkonovou třídu nižší, než ty, které jsou v daném standardu specifikovány.

	Třída	Výkon PSE	Výkon PD
Typ 3	5	45 W	40 W
	6	60 W	51 W
Typ 4	7	75 W	62 W
	8	90 W	73 W

Tabulka 3.3: *IEEE 802.3bt* Výkonové třídy typu 3 a 4 [5]

3.4 Klasifikace zařízení

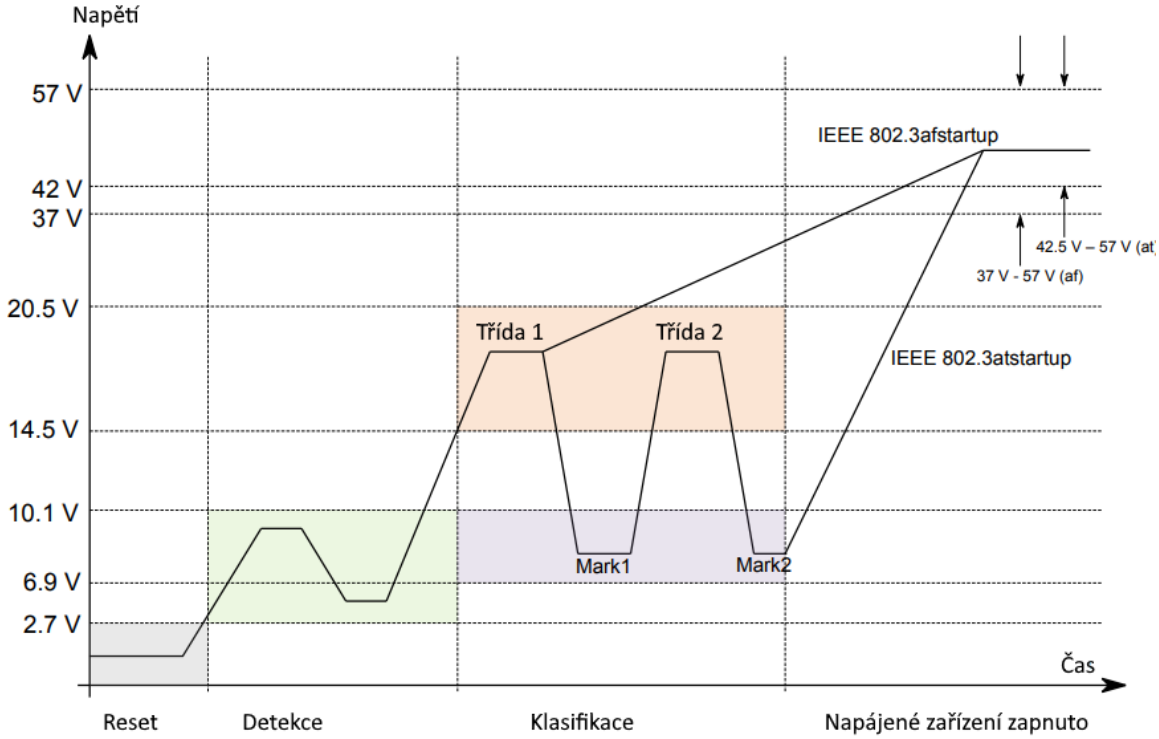
Protože veškerá zařízení dodržující standard musí použít aktivního PoE, napájecí zařízení (PSE) v klidovém stavu nejdříve provádí detekci koncového zařízení. Na svém výstupu (dle použitého módu) tedy poskytuje napětí v rozsahu 2,7 – 10,1 V a čeká, než se kabelem připojí koncové zařízení. Dojde-li k připojení, napájené zařízení (PD) musí vyvinout charakteristický detekční odpor 25 kΩ.

Tento odpor je detekován napájecím zařízením, jenž si může být v tuto chvíli jisté, že je na druhé straně opravdu napájené zařízení. Napětí na vodičích je tedy navýšeno na 14,5 V (klasifikační napětí) a pokračuje se klasifikací výkonové třídy.

Typ 2 napájecí zařízení provádí klasifikaci ve dvou fázích. V první fázi, a poté co bylo zmiňované napětí zvýšeno, měří odebíraný proud. Pro každou z výkonových tříd je specifikován proud, jenž jí reprezentuje. Pokud koncové zařízení odpoví správnou proudovou zátěží, napájecí zařízení sníží úroveň napětí na tzv. *mark* (7 – 10 V).

V případě, že se jedná o typ 1 PSE, a daná proudová zátěž je mimo podporovanou třídu, je automaticky přidělena třída 0, tedy 15,4 W a napětí zvýšeno na provozní (37 – 57 V). Pokud se jedná o typ 2 PSE a tento proud specifický pro třídu 4, dojde opět ke zvýšení napětí na klasifikační, koncové zařízení musí odpovědět nevalidní proudovou zátěží, a poté dojde opět na snížení na hodnotu *mark*. Tímto dvojitým pulsem na úroveň *mark* PSE signalizuje PD, že mu byla přidělena třída 4 a následně zvýší napětí již na provozní (v případě typu 2 se jedná o 42,5 – 57 V). [22] [16]

Následující obrázek 3.2 zobrazuje proces detekce a signalizace typ 2 PSE s typ 2 PD zařízeními.



Obrázek 3.2: Průběh detekce a klasifikace typ 2 PSE a PD zařízení. Převzato a upraveno z [22]

Kapitola 4

Systémy chytré domácnosti

V posledních letech se rozmáhá koncept chytrých domácností. Úkolem takové domácnosti je zjednodušit lidem život, případně také zefektivnit využití zdrojů. Díky tomu lze také domácnosti provozovat s menšími náklady. [9] Důležitým prvkem je možnost automatizace a vzdálená správa zařízení umístěných v domácnosti [2].

Právě nízká cena způsobená masovou výrobou a dostupností Wi-Fi mikrokontrolérů, jako je ESP32, umožnila rozšíření různých druhů senzorů, které je schopen instalovat i běžný člověk. Příkladem jsou chytré zásuvky, jenž stačí vložit do existujících zásuvek.

Tyto senzory společně s dalšími zařízeními následně tvoří systémy chytrých domácností. Senzory a zařízení komunikují mezi sebou a předávají si data. Rozšířenější je však v moderní době připojení všech těchto prvků k centrálnímu řídícímu systému, na němž běží automatizace. Typicky se jedná o malý počítač umístěný uvnitř budovy či centrální server nějaké služby v cloudu. [15].

Mezi cloudové služby chytrých domácností, do kterých je možné připojit zařízení, patří například Apple HomeKit či Amazon Alexa. Po přidání je možné pomocí mobilních aplikací či hlasových asistentů tyto zařízení ovládat.

4.1 Přenosové technologie

Protože jsou používány senzory a zařízení různých výrobců, neexistuje žádný jeden konkrétní standard, kterým by se prvky chytrých domácností dorozumívaly. Standardů, jež se používají pro komunikaci, je tedy několik.

Wi-Fi a lokální síť

Zařízení jsou připojena ke stejné Wi-Fi síti, přes kterou se dorozumívají s řídícím systémem nebo mezi sebou. Můžou být takto připojena také k internetu, a navíc se jedná o uživatelsky nejpřívětivější možnost, neboť Wi-Fi je v dnešní době dostupná téměř v každé domácnosti. Toto provedení je tedy ideální pro cloudová řešení.

Nevýhodou je však, že většina těchto chytrých zařízení využívá pouze pásmo 2.4 GHz, a v zastavěnějších oblastech tak bývá problém vzájemného zarušení. Zařízení s podporou pásma 5 GHz se teprve dostávají na trh.

Zigbee

Technologie Zigbee využívá primárně pásmo 2.4 GHz, tedy bezlicenční pásmo. Její výhodou je nízká energetická náročnost společně s dosahem v ideálních podmínkách až 300 metrů. Jednotlivá Zigbee zařízení jsou taktéž schopná komunikovat mezi sebou a tvořit tzv. *mesh* sítě, kde každé ze zařízení může fungovat jako opakovač signálu. Pokud ovšem potřebujeme se zařízeními komunikovat například z internetu, je nutné pořídit Zigbee bránu, která se připojí do sítě a která komunikuje se zigbee zařízeními v *mesh* síti. [15]

Z-Wave

Podobně jako Zigbee se jedná o *mesh* síť s podporou opakovačů. Rozdílem je však použití nižších frekvenčních pásem, v případě Evropy 868 MHz, díky čemuž se vyhýbá kolizím s Wi-Fi sítěmi. Mezi zařízeními různých výrobců je taktéž zajištěna lepší kompatibilita, neboť jakékoliv zařízení, jenž chce používat značku Z-Wave, musí projít certifikací. [15] [29]

4.2 Systém Home Assistant

Tato podkapitola čerpá z [10]. Home Assistant je jeden z nejrozšířenějších systémů pro chytré domácnosti. Jeho hlavní výhodou je možnost ho nechat běžet kompletně lokálně, například na domácím serveru, mini počítačích, jako je Raspberry Pi, či specializovaných zařízeních právě pro Home Assistanta, jako je Home Assistant Green [1].

Celý systém je vyvíjen open source a vyvíjen tak, aby poskytoval co největší soukromí jeho uživatelům. Veškerá data jsou ukládána lokálně a systém nepotřebuje pro své fungování ve výchozím stavu žádné cloudové služby. Systém je psán v jazyce Python, a ke konfiguraci často využívá značkovací jazyk YAML.

Do systému je možné přidávat zařízení, které se tímto systémem dají ovládat či z nich číst data. To, jaké funkce tato zařízení poskytují systému, je řízeno tzv. integracemi. Každé ze zařízení poskytované integrací může obsahovat několik entit.

Na základě těchto entit pak může uživatel vytvářet ovládací panely zobrazující data či reagovat na události zařízení ve formě automatizací.

Entity a její typy

V systému Home Assistant jsou zařízení zobrazována jako entity. Entita může systému poskytovat například data ze senzorů, či umožnit systému dané zařízení ovládat. Pro úspěšné ovládání musí entita poskytovat akce, které jsou pak zasílány zařízení.

Data, jenž jsou entitou poskytována, jsou součástí jejího aktuálního stavu. Tento stav je v systému zaznamenáván v databázi, a je tak možné v uživatelském rozhraní systému zobrazit například průběh hodnot dané entity.

Entity mohou být různých typů, na čemž pak záleží také hodnoty, kterých může nabývat, či akce, které poskytuje. Těchto typů je velmi hodně, a popisovat je všechny je zbytečné. Příkladem však můžou být následující:

- **Switch** – entita pro spínač. Její aktuální stav může být buď Zapnuto či Vypnuto. Poskytuje akce Zapnout, Vypnout či Přepnout

[1] https://www.home-assistant.io/green/

- **Sensor** – jednoduchá entita jejímž jediným stavem je její hodnota. Může se jednat například o aktuální teplotu či procenta aktuální spotřebu chytré zásuvky. Entita má také ještě specifickou třídu která udává jakých jednotek nabývá. Tato třída pak usnadňuje zobrazení v uživatelském rozhraní
- **Fan** – entita ventilátoru. Její aktuální stav může být buď Zapnuto či Vypnuto. Krom klasických akcí zapnutí a vypnutí však poskytuje také možnost nastavovat rychlost či směr pohybu ventilátoru

Automatizace

Poskytují možnost reagovat na změny stavů a událostí entit. Každá z automatizací musí mít alespoň jeden či více spouštěčů. Může se jednat například o změnu hodnoty senzoru, či přepnutí spínače do konkrétního stavu.

Ke spouštěči je možné také přidat podmínku, která musí být splněna, aby byla automatizace spuštěna. Může se jednat například o jednoduché porovnání, zda má nějaká jiná entita konkrétní hodnotu.

Jako poslední položka automatizace se definuje seznam akcí, jenž bude proveden při spuštění automatizace. Lze tak provést akci na nějaké jiné z entit.

Automatizace se v systému Home Assistant vytváří s použitím uživatelského rozhraní nebo jako konfigurace v jazyce YAML.

Jako příklad automatizace lze uvést zhasnutí všech světel v domě, pokud jsou zamknuty vchodové dveře.

Integrace

Integrace jsou stěžejní součástí systému Home Assistant. Bez podpory výrobců chytrých zařízení by celý systém postrádal smysl. Tyto integrace jsou tedy vytvářeny výrobci či komunitou, a to vždy pro konkrétní zařízení. Mohou však existovat integrace univerzálnější, například pro senzory využívající technologie Z-Wave či ZigBee, kdy je pak na výrobci konkrétního zařízení, aby dodržel formát komunikace po těchto bezdrátových technologiích.

Integrace jsou vytvářeny v jazyce Python, a je možné je dynamicky načítat. Každá z integrací se sama stará o správu jejích zařízení, a poté, co nějaké zařízení poskytne systému Home Assistant, přiřadí mu také seznam entit. Integrace dostává od Home Assistenta příkazy pro zjištění stavu či vykonání akce, a je již na integraci, jakým způsobem se se zařízením komunikuje.

Ovládací panel

Ovládací panely **Lovelace**, anglicky *Dashboards*, jsou částí uživatelského rozhraní, v nichž si uživatel může přidávat tzv. karty. Tyto karty jsou poskytovány jak Home Assistentem samotným, tak také integracemi, které můžou definovat jejich vlastní vzhled. Na každém ovládacím panelu může být uspořádáno několik těchto karet.

Typicky se karty používají na zobrazení dat ze senzorů (aktuální hodnota, graf) či na vykonávání akcí po kliknutí.

Spuštění Home Assistenta

Systém Home Assistant je nabízen ve třech formách instalace. Některé z nich jsou omezené funkcionalitou, což mnohdy ale kompenzují jednoduchostí instalace.

- **Home Assistant Core** – systém je nainstalován jako Python program pod libovolným operačním systémem. Krom Pythonu a použitých balíčků nemá žádné další požadavky. Aktualizace je nutné provádět manuálně
- **Home Assistant Container** – systém je distribuován a spuštěn v kontejneru, například ve službě Docker. Veškeré závislosti jsou zabaleny již v kontejneru, který stačí spustit. Aktualizace je možné provádět stáhnutím nové verze kontejneru
- **Home Assistant OS** – instalace je provedena ve formě celého operačního systému pod kterým běží Home Assistant Core. Aktualizace je možné provádět přímo přes webové rozhraní, kterým lze aktualizovat jak operační systém samotný, tak Home Assistant Core

Kapitola 5

Přenos audiovizuálních záznamů po síti

Tato kapitola se věnuje protokolům, díky kterým lze po síti přenášet video a audio, a to s co nejmenší náročností na výpočetní výkon, latenci a v neposlední řadě co nejmenším datovým tokem. Dále se také zabývá existujícími kodeky pro efektivní kódování těchto médií na mikroprocesorech, včetně těch s hardwarovou akcelerací.

5.1 Real-Time Streaming Protocol (RTSP)

Jedná se o protokol pro ovládání přenosů médií po síti v reálném čase. V ISO/OSI modelu se řadí do aplikační vrstvy a jeho fungování je popsáno v RFC 2326 z roku 1998. Tato a následující dvě podkapitoly 5.2 a 5.3 tedy vycházejí z [6].

Samotný protokol byl inspirován protokolem HTTP/1.1 a využívá tedy také TCP spojení s jednotlivými klienty. Narozdíl od HTTP musí RTSP server v paměti udržovat aktuální stav, ovšem jednotlivá TCP spojení jsou bezstavová. Komunikace se serverem probíhá v textové podobě a stejně jako v případě HTTP je nejdříve zaslán dotaz na nějž druhá strana odpovídá.

Veškerá komunikace a ovládání je založeno na architektuře klient-server, tedy běží jeden server, který média poskytuje, a jeden či více klientů tento server ovládá.

Důležité je také zmínit, že RTSP jako takové nepřenáší audio a video data, pouze umožňuje tento přenos řídit. K samotnému přenosu dat se využívá protokol RTP popsaný v sekci 5.4.

5.2 Formát komunikace RTSP

Jak již bylo zmíněno, komunikace probíhá formou dotazů. Na každý dotaz musí druhá strana odpovědět právě jednou validní odpovědí. Tělem dotazů a odpovědí je text kódovaný pomocí UTF-8 kódování rozdělený do řádků. Ty se od sebe rozlišují zasláním bílých znaků CR a LF (*Carrriage Return a Line Feed*) za sebou.

Každý dotaz se vždy skládá z prvního řádku popisující aktuální metodu, URL přistupovaného prostředku a použité verze RTSP zakončené CRLF.

Následují řádky s hlavičkami, každá ve formátu `NazevHlavicky: Hodnota` opět zakončeny CRLF. Počet hlaviček není omezen, ovšem duplicita není povolena. Konec dotazu je pak signalizován zasláním prázdného řádku, neboli znaků CRLF dvakrát za sebou.

Součástí dotazu však může být také obsah v libovolném formátu. Protože se nemusí jednat o text a jeho délka může být libovolná, musí být součástí seznamu hlaviček právě jedna `Content-Length` hlavička specifikující délku tohoto obsahu v bajtech. Příjemce tak musí tento obsah od odesílatele přečíst.

První řádek odpovědi na dotaz musí vždy začínat verzí RTSP protokolu, následuje stavový kód a zakončení zprávou, jenž tento kód zdůvodňuje. Dále jsou uvedeny opět hlavičky, které příjemce dotazu uvádí. Celá odpověď je opět zakončena dvojicí CRLF. Stejně jako v případě dotazu, za odpovědí může následovat obsah, jehož délka musí být taktéž správně specifikována hlavičkou `Content-Length`.

Důležité je také zmínit, že každý z dotazů musí vždy obsahovat hlavičku `CSeq` (Sekvenční číslo) jejíž hodnota je zopakována v odpovědi. Takto je odesílateli zřejmé, na jaký z jeho dotazů dostal odpověď.

Tento příklad sekvence dotazu a následné odpovědi zobrazuje dotaz s metodou `OPTIONS` na URL `rtsp://127.0.0.1/`:

```
OPTIONS rtsp://127.0.0.1/ RTSP/1.0\r\n
CSeq: 22\r\n
\r\n
RTSP/1.0 200 OK\r\n
CSeq: 22\r\n
Public: OPTIONS, DESCRIBE, SETUP, TEARDOWN, PLAY\r\n
\r\n
```

Lze vidět, že server odpověděl stavovým kódem `200 OK` a v hlaviččce `Public` zaslal informace o dostupných RTSP metodách.

5.3 Metody protokolu RTSP

Klienti mohou po připojení na server vykonávat některou z metod. Metoda může být zaslána buď ze strany klienta, nebo také ze serveru, avšak k tomu dochází velmi vzácně. Následuje popis zásadních metod, jenž musí podporovat minimální implementace RTSP serveru.

OPTIONS

Metoda `OPTIONS` se využívá ke zjištění schopností druhé strany. Při použití není zapotřebí nastavovat URL dotazu, neboť se požadavek týká schopností serveru/klienta jako takového, a taktéž je možné tento dotaz zaslat ve směru server→klient. Druhá strana musí na dotaz odpovědět seznamem dostupných metod v hlavičce `Public`.

DESCRIBE

Získává popis poskytovaných medií datových proudů serveru. Pouze jednostranná metoda klient→server, kdy server musí odpovědět popisem médií v příslušném formátu. Tento popis je v příslušném formátu navrácen jako obsah odpovědi, tedy nastavena příslušná `Content-Length` hlavička, a také `Content-Type` pro informování klienta o jaký formát se jedná.

Klient může v hlavičce `Accept` uvést seznam akceptovaných formátů, pokud tak neudělá, volba zůstává na serveru. Nejčastějším zasílaným formátem je však SDP (Session Description Protocol).

SETUP

Jednou z velmi důležitých metod je právě `SETUP`. Jeho zasláním klient připraví server na zasílání konkrétního proudu média a sebe na příjem. URL uvedená v dotazu se musí týkat konkrétního proudu.

Dotaz musí obsahovat informace týkající se transportního kanálu, jenž je vyhrazen pro tento proud. Tyto informace jsou obsaženy v hlavičce `Transport` a je-li tento typ transportního kanálu podporován, server si jej inicializuje. Data však nesmí být zatím odesílána.

Aby bylo možné daný datový proud, a zároveň kanál později ovládat, server si jej asociuje s tzv. *Session identifier*. Tento identifikátor je alfanumerický text libovolné délky a je generován náhodně serverem při prvním použití metody `SETUP` daným klientem. Server jej vrací v hlavičce `Session`, avšak může se vyskytnout také v hlavičce dotazu, a v tom případě musí server nově vytvořený kanál přibalit k existujícím kanálům.

PLAY

Byl-li vytvořen příslušný transportní kanál, a klient je připraven přijímat datový tok média, zašle dotaz `PLAY`, kterým se spustí zasílání dat. Díky použité URL v dotazu a uvedenému identifikátoru sezení v hlavičce `Session` dokáže server přesně vybrat transportní kanál a datový proud média, na němž má dojít k zahájení přenosu.

Hlavička `Range` může specifikovat časový úsek k přehrání ve formátu `npt=start-konec` kde start a konec udává čas média v sekundách. Alternativou je formát `clock=start-konec`, kde je start a konec hodnota v UTC čase. V obou formátech je možné čas konce vynechat, což vyznačuje přehrávání až do konce média. V případě, že hlavička není uvedena, server přehrává obsah média od začátku až po konec.

TEARDOWN

Zasláním dotazu s touto metodou je ukončen přenos datového proudu média specifikované URL. Server pro tuto URL a dané sezení ukončí zasílání dat a vytvořený transportní kanál je deinicializován. Jakékoliv volání `PLAY` na tuto URL v budoucnu bude nevalidní, a klient, pokud potřebuje, musí transportní kanál obnovit pomocí `SETUP`. V případě, že klient nespecifikuje v URL konkrétní datový proud, dojde k této deinicializaci pro všechny transportní kanály asociované s použitým sezením.

Hlavička Require

Každý požadavek taktéž může obsahovat jednu či více hlaviček `Require`. Tím se snaží dotazující strana zjistit, zda je podporována specifická funkcionalita (ne metoda). Jestliže je veškerá funkcionalita podporována, server odpoví jako vždy seznamem metod. Pokud-li však ne, musí odpovědět chybovým stavovým kódem `551 Option not supported` a informovat o této chybějící funkcí hlavičkou `Unsupported` obsahující příslušný název funkce.

Typy transportních kanálů pro RTP

Metodou `SETUP` lze vytvořit transportní kanál různých typů. Podporován je jak unicast na protokolech TCP a UDP, tak multicast za pomocí UDP. Následuje popis těchto typů.

UDP unicast

Nejjednodušší způsob přenosu dat využívá otevření UDP socketu přímo ke klientovi. K otevření tohoto socketu je zapotřebí port serveru a kombinace adresy a portu klienta. Port serveru si server vybírá sám a měl by být unikátní napříč všemi spojeními. Port klienta je zasílán v hlavičce `Transport`.

Po vytvoření těchto socketů jsou RTP zprávy odesílány jako jednotlivé UDP datagramy. Výhodou zasílání tímto transportním kanálem je nízká latence.

Prokládané TCP (interleaved)

Druhou nejčastější možností je prokládání přenosu dat přímo uvnitř otevřeného RTSP spojení mezi klientem a serverem. Tento způsob sice může zvyšovat latenci, na lokálních sítích je však tento problém minimální.

Klient při zvolení tohoto typu transportního kanálu specifikuje hlavičkou `Transport` virtuální port, na němž komunikace probíhá. Tento port je vázaný na RTSP spojení, na němž byla metoda `SETUP` zavolána, a musí být v rámci tohoto spojení unikátní. Tento port pak slouží k identifikaci konkrétního transportního kanálu.

Zasílání pak probíhá tak, že je nejdříve odeslán speciální znak (**$**) signalizující prokládaná data. Následuje jeden bajt virtuálního portu, dva bajty délky zprávy a poté zpráva samotná. Příklad takovéto zprávy zobrazuje tabulka 5.1

Popis	**Znak**	**Port**	**Délka**	**Zpráva**
Hex	`0x24`	`0x08`	`0x00 0x05`	`0x48 0x65 0x6c 0x6c 0x6f`
Hodnota	$	8	5	Hello

Tabulka 5.1: Příklad zaslané zprávy přes prokládaný kanál

5.4 Real-time transport protocol (RTP)

Tento protokol vznikl za účelem přenosu dat v reálném čase, jako je audio či video. Specifikuje, jak se mají tato data správně zabalit a poslat; sám je však kompletně nezávislý na transportní vrstvě. Tato podkapitola vychází z [8].

Formát zprávy

Jedná se o paketový protokol, zasílaná data jsou tedy rozdělena do bloků a odesílána jako jednotlivé zprávy. Každá zpráva se skládá z příslušné RTP hlavičky, za níž jsou jednoduše vložena celá data média. Následující tabulka 5.2 zobrazuje formát této hlavičky.

Tabulka 5.2: Formát RTP hlavičky

Bity	Pole	Název
0-1	V	Version
2	P	Padding
3	X	Extension
4-7	CC	CSRC Count
8	M	Marker
9-15	PT	Payload type
16-31	Sequence Number	Packet number
32-63	Timestamp	Packet timestamp
64-95	SSRC	Synchronization Source
96-...	CSRC	Contributing Source

Version (V) – 2 bity

Specifikuje použitou verzi RTP. Nejnovější, a tudíž nejčastější verze 2.

Padding (P) – 1 bit

Je-li tento bit nastaven, na konci paketu se nachází libovolný počet bajtů jako výplň. Velikost této výplně udává úplně poslední bajt součástí zprávy.

Extension (V) – 1 bit

Je-li tento bit nastaven, za hlavní RTP hlavičkou se nachází ještě jedna rozšiřující hlavička.

CSRC Count (CC) – 4 bity

Počet CSRC hodnot nacházejících se za hlavní RTP hlavičkou.

Marker (M) – 1 bit

Značka jejíž význam silně závisí na typu obsahu zprávy. Typicky však signalizuje příjemci, že se jedná o poslední zprávu nějakého časového úseku. V případě videa například o poslední jednotku nutnou pro vykreslení jednoho snímku.

Payload type (PT) – 7 bitů

Označuje typ přenášených dat. Může se jednat o jeden z předem specifikovaných typů, nebo o dynamicky definovaný typ. V takovém případě musí být předem definován jiným komunikačním kanálem.

Sequence number – 16 bitů

Označuje pořadové číslo zprávy. Protokol předpokládá, že použitá transportní vrstva může být nespolehlivá, a že tak může docházet k přehození zpráv či jejich nedoručení. Uvedení této hodnoty tak pomáhá příjemci rekonstruovat pořadí zpráv.

Timestamp – 32 bitů

Časová značka dat zprávy. Určuje čas, v němž došlo k navzorkování prvního bajtu dat média. Tento čas se řídí hodinami specifikovanými použitým typem přenášených dat.

Synchronization source (SSRC) – 32 bitů

Identifikuje zdroj dat napříč více RTP spojeními. Klient je tak schopen rozpoznat, které datové proudy médií synchronizovat s kterými.

Contributing source (CSRC) – 32 bitů × CC

Jedná se o seznam s délkou specifikovanou v poli *CC*. Speciální pole využívané pouze v případě, že jsou data odesílána z tzv. mixéru. Ten může mixovat více RTP proudů do jednoho a v tomto seznamu se pak nachází originální *SSRC* těchto proudů.

5.5 H264 kodek

H.264, jinak nazývaný MPEG Advanced Video Coding, je standard pro kódování videa vytvořený skupinami *MPEG Group* a *International Telecommunications Union*. Jedná se o ztrátový kodek, a vzhledem k nastavitelnosti úrovně komprese se hodí pro různá použití, začínaje televizními přenosy, uložení filmů či přenosy z IP kamer. [17]

Typický kodér H.264 se skládá ze dvou hlavních částí. Těmi jsou VCL (video coding layer), která se stará o práci s vizuálními daty, a NAL (network abstraction layer), jenž její výsledky převádí na formát snadno přenositelný sítí.

VCL s videem pracuje v blocích o velikosti 16×16 pixelů, tzv. makrobloky. Jeho výstupem pak jsou různé typy snímků: *Intra*, *Predicted* a *Bi-predictive* frames. Intra snímky jsou vždy vytvářeny přímo z obrazu v daný moment. Predictive snímky jsou pak vypočítávány na základě změn v obraze od posledního intra snímku. Speciální variantou jsou bi-predictive snímky, které počítají i s daty z budoucích intra snímků. [14]

Formát NAL jednotky

Výstupem kodéru jsou tzv. NAL units (jednotky) obsahující informace o snímcích. Každý snímek, ať už intra či predictive, je pak složen z několika těchto jednotek. Jednotka může také obsahovat informace o videu jako takovém, např. jeho rozlišení či snímkovací frekvenci. [27]

Obsahem jednotky je vždy jeden bajt s hlavičkou následovaný daty. Tuto hlavičku zobrazuje následující tabulka 5.3.

7	6	5	4	3	2	1	0	...
F	R		T					NAL data

Tabulka 5.3: Hlavička NAL jednotky [27]

Forbidden bit (F) – 1 bit

Hodnota tohoto bitu musí být vždy nula. Zamýšleno jako způsob, jak může síťový prvek během přenosu informovat dekodér, že danou NAL jednotku není zapotřebí zkoušet dekódovat, protože obsahuje chyby.

Reference idc (R) – 2 bity

Umožňuje definovat, jak moc je jednotka důležitá pro správné dekódování. Čím vyšší hodnota, tím důležitější.

Type (T) – 5 bitů

Určuje typ jednotky. Hodnoty 1 – 12 jsou definovány kodekem a definují skutečné typy. Rozsah 24-31 je využitelný pro jakékoliv jiné účely definované uživatelem jednotek, např. pokud jsou jednotky přenášeny sítí, umožňuje tento rozsah fragmentaci jednotek bez použití hlaviček navíc. Uživatel si jednoduše zvolí hodnotu z tohoto rozsahu, aby signalizoval fragmentaci či agregaci.

Hodnoty 0 či rozsah 13–23 jsou rezervovány pro budoucí použití a neměly by být využívány.

Uložení NAL jednotek

Sekvenci NAL jednotek je zapotřebí ukládat či přenášet v nějaké podobě. Existují tři způsoby přenosu:

1. **Annex A** – celá jednotka se zabalí do jedné zprávy a zašle. Příkladem je uložení v UDP datagramu
2. **Annex B** – využívá se tzv. *startcodes*. Před každou jednotkou je umístěňa sekvence bajtů (`0x000001` nebo `0x00000001`) značící začátek jednotky. Detekcí těchto sekvencí je pak možné nalézt začátky a konec jednotek jdoucích za sebou. Protože se můžou tyto sekvence vyskytovat v jednotkách samotných, je před jakýkoliv jejich výskyt při kódování vsunuta hodnota `0x03`
3. **AVC** – před každou jednotku je vložena její délka

Přenos H264 pomocí RTP

NAL jednotky kodeku H.264 je možné do RTP zprávy vložit několika způsoby. Jednotky se do zpráv vkládají bez zbytečných hlaviček navíc, pokud to není nutné, a proto je prvním bajtem každé z těchto zpráv validní hlavička NAL jednotky. [28]

Paketizace

Jedná se o formu **Annex A** uložení, každá NAL jednotka je jednoduše vložena do RTP zprávy a zaslána. Může být neefektivní při velkém počtu menších jednotek (zbytečná režie hlaviček). V případě velkých jednotek a použití UDP můžou být při překročení MTUsíťe takto velké datagramy zahozeny. [28]

Fragmentace

Jednotka je rozdělena do více RTP zpráv a jako signalizační prostředek pro informování příjemce o fragmentaci použit jeden z nevyužitých typů v hlavičce NAL jednotky. Za touto hlavičkou musí následovat hlavička fragmentace vyznačující, kde se aktuální zasílaný fragment nachází. [28]

Agregace

V případě velmi malých NAL jednotek je možné je agregovat do jedné RTP zprávy. Agregace se dělí na dva způsoby: *single-time aggregation packets*, jenž obsahuje jednotky se stejnou časovou značkou, nebo *multi-time aggregation packets* v nichž se mohou časy jednotek lišit. K signalizaci příjemci je použito stejně jako v případě fragmentace jednoho z nevyužitých typů. [27]

5.6 G.711 kodek

Kodek G.711 je rozšířený kodek v telekomunikacích, neboť je vhodný pro kódování mluveného slova. V mnohých komunikačních protokolech jde o minimálně podporovaný kodek pro přenos audio záznamů, neboť je jeho implementace jednoduchá.

Jeho výstupem je 8bitová hodnota a v kombinaci s typickou vzorkovací frekvencí 8 kHz pak dosahuje datového toku 64 kbit/s. Využívá logaritmické komprese a má dvě varianty, na jejichž základě se liší velikost vstupního signálu.

První variantou je μ-law, který je používán spíše v severní Americe a Japonsku. Vstupem této varianty je 13bitová hodnota se znaménkem. Varianta A-law je určena pro státy v Evropě a Austrálii a jejím vstupem je 14bitová hodnota se znaménkem. [12]

Přenos G.711 pomocí RTP

Přenos tohoto kodeku pomocí RTP je triviální. Jednotlivé 8bitové vzorky jsou, tak jak byly vyjmuty z kodéru, vloženy do RTP zprávy. Typicky se odesílá po blocích 80, 160 či 240 vzorků, což při vzorkovací frekvenci 8 kHz tvoří bloky o 10 ms, 20 ms či 30 ms záznamu.

Jako *Payload type (PT)* RTP zprávy se volí hodnota 8 pro A-law variantu a hodnota 0 pro μ-law. [7]

Kapitola 6

Autentizace uživatelů pro přístup

Tato kapitola se zabývá způsoby autentizace uživatelů pro přístup použitím něčeho, co uživatel vlastní. Cílem je zrychlit proces otevření dveří, tak aby uživatel nemusel zadávat kód.

Použitím těchto technologií typicky dochází ke komunikaci mezi autentizačním předmětem (klíčenka, karta, apod.) a terminálem. Klíčenka či karta si takto vymění svůj unikátní identifikátor s terminálem a ten na základě své databáze ověří jejího vlastníka. [24]

Tato komunikace může probíhat buď bezdrátově, či v případě některých technologií také přímým kontaktem čipu se čtečkou.

6.1 Technologie RFID

RFID (Radio Frequency Identification) je skupina bezdrátových technologií sloužících k identifikaci předmětů bez nutnosti kontaktu. Dosah těchto technologií se liší v závislosti na použité frekvenci a může se pohybovat v rozsahu několika centimetrů až metrů v případě použití velkých antén. Výhodou je tedy funkčnost skrze pevné objekty, jako jsou plasty či oblečení. [26] [25]

Po přiložení karty ke čtečce a následné komunikaci mezi nimi dochází k výměně dat mezi čtečkou a tzv. RFID tagem (nejčastěji RFID karta či klíčenka) a to za pomocí elektromagnetických vln. Obě strany tudíž vyžadují adekvátní anténu navrženou pro použitou frekvenci. [3]

Jednotlivé tagy se dělí na dva typy, aktivní a pasivní.

Aktivní RFID

Aktivní RFID tagy mají svůj vlastní zdroj napájení, což je typicky baterie. Jejich životnost je tedy omezena, avšak výhodou je pak větší dosah. Nespoléhají na rozdíl od pasivních RFID tagů na blízkou přítomnost čtečky a mohou tak vysílat na vyšší vzdálenosti.

Pasivní RFID

Tyto RFID tagy jsou napájeny elektromagnetickým polem přímo ze čtečky. Čtecí zařízení neustále generuje za pomocí své cívky elektromagnetické pole a čeká, než je do něj vložen RFID tag. Po jeho přiložení je postupně nabíjen vnitřní kondenzátor umístěný uvnitř tohoto tagu a ve chvíli, kdy dojde k jeho nabití, tag je dostatečně napájen a může tak dojít ke

komunikaci. [23] Takto provozované karty či čipy jsou mnohem levnější než aktivní tagy, protože nevyžadují vlastní zdroj energie.

Nízkofrekvenční RFID

Tyto RFID tagy fungují ve frekvenčních rozsazích od 125 kHz po 134 kHz. Díky takto nízké frekvenci musí být anténa tvořena větším počtem (stovky) vinutí. Dalším omezením je také maximální velikost unikátního identifikátoru, která tvoří pouhých 64 bitů. [18]

Tyto tagy jsou však vyráběny historicky delší dobu a ve větších množstvích, a tudíž jsou ze všech druhů ty nejlevnější.

Vysokofrekvenční RFID

Nejznámějším zástupcem moderní doby jsou právě vysokofrekvenční RFID tagy pracující na frekvencích 13,56 MHz. Na této frekvenci pracuje právě také NFC (Near Field Communication) společně s bezkontaktními platbami.

Tyto technologie jsou definovány standardem ISO/IEC 14443. Ten specifikuje použitou modulaci pro komunikaci mezi čtečkou a tagem, komunikační protokol a antikolizní funkce. Jedná se však pouze o obecný standard a konkrétní implementace včetně využití jsou ponechány na výrobcích.

Jedním z těchto výrobců je společnost *NXP Semiconductors* a jejich čipy MIFARE, konkrétně pak řada MIFARE Classic. Čipy této řady mohou obsahovat buď 4 nebo 7 bajtové unikátní identifikátory. Výběr, která z těchto dvou možností bude použita, je zapotřebí udělat již při výrobě, neboť hodnota tohoto identifikátoru je při ní nepřepisovatelně uložena.

Existují však čipy třetích stran, které se chovají v souladu s MIFARE classic implementací a dají se přepsat.

Kapitola 7

Návrh technického řešení

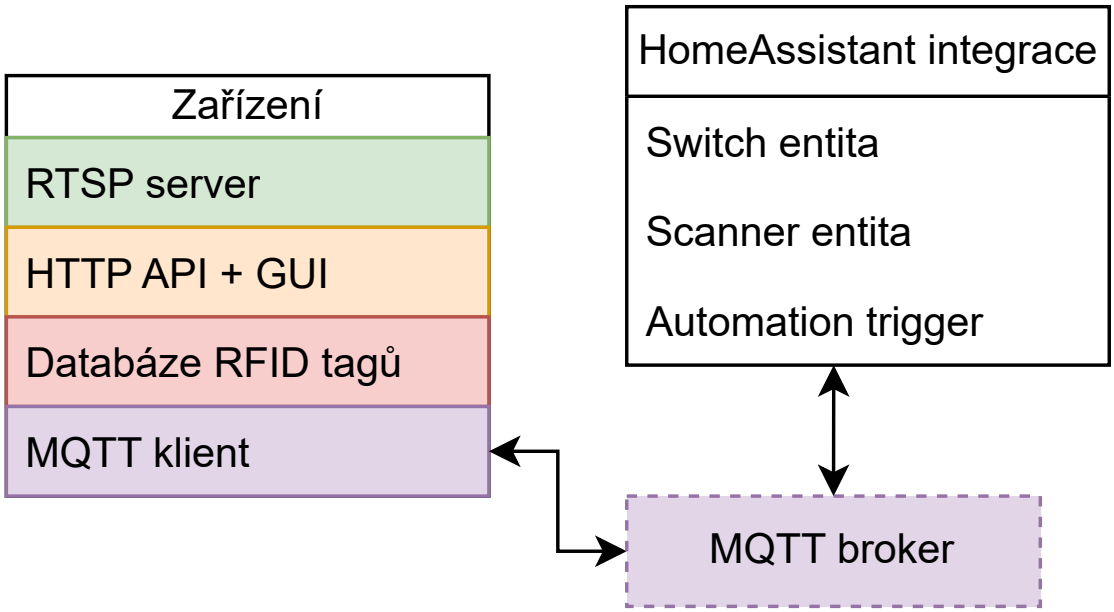
Důležitou částí návrhu je stanovení požadovaných cílů celé práce. Tím je vytvoření hardwaru a obslužného softwaru pro zařízení. Jedná se o kombinaci zvonku a přístupového systému v jednom. Toto zařízení bude doplněno integrací pro open source platformu Home Assistant, pomocí níž může uživatel dálkově zařízení ovládat, a také webovým rozhraním běžícím přímo na zařízení pro jeho správnou konfiguraci.

Zařízení by tedy mělo splňovat následující cíle:

- **Funkce zvonku** – cizí osoba před vchodovými dveřmi/bránou po zmáčknutí tlačítka informuje obyvatele. To proběhne komunikací s integrací Home Assistant
- **Kamerový přenos** – zařízení bude umožňovat přenos kamerového záznamu z kamery a to včetně přenosu zvuku z mikrofonu v zařízení. Dále také podpora zpětného audio přenosu z klientské aplikace směrem k zařízení a přehrání reproduktorem
- **Otevření dveří** – na základě požadavku ze strany integrace či v případě správné autentizace uživatelem dojde adekvátním způsobem k otevření dveří či vrat
- **Autentizace uživatele** – přiložením klíčenky s identifikátorem je tento identifikátor porovnán s uloženou databází a v případě jeho nalezení musí dojít k otevření dveří
- **Připojení k síti** – zařízení bude k síti připojeno drátově za pomocí standardu ethernet a tudíž UTP kabely
- **Napájení** – napájení bude probíhat přivedenými vodiči. Vzhledem k již použitému připojení k síti UTP kabelem je ideální volbou technologie Power Over Ethernet (PoE)
- **Konfigurační rozhraní** – přímo na zařízení poběží webový server poskytující rozhraní s možností konfigurace a taktéž ovládání zařízení

Díky stabilnímu síťovému připojení ethernet bude veškerá komunikace mezi integrací Home Assistant a zařízením probíhat pomocí protokolu MQTT, jenž je ideální pro IoT [1] aplikace. Přeos audio a video záznamů pak bude zajišťovat protokol RTSP. Následující obrázek 7.1 zobrazuje obecné schéma situace.

[1] IoT- Internet Of Things



Obrázek 7.1: Obecné schéma zařízení a integrace

7.1 Návrh hardware

Jako hlavní prvek zařízení jsem se rozhodl použít mikroprocesor firmy Espressif, konkrétně ESP32-P4. K jeho výběru mě přiměl jeho vysoký výkon vhodný pro živé přenosy video záznamů. Jedná se o dvoujádrový procesor s frekvencí 400 MHz a také rozsáhlým souborem periferií. Tou významnou je právě rozhraní MIPI CSI[2], díky kterému lze snadno připojit velkou škálu kamerových senzorů. Další obsaženou periferií je hardwarový akcelerátor kódování H.264 s podporou rozlišení Full HD (1920×1080).

Napájení

Napájení celého systému jsem se rozhodl provést technologií Power over Ethernet. Díky tomu bude vedeno napětí zároveň s daty a zároveň docíleno galvanického oddělení zařízení od napájecího zdroje. Proto jsem se rozhodl použít integrovaný modul AG5412 firmy Silvertel, který se postará o správné provedení klasifikace s napájecím zdrojem. Součástí tohoto modulu je napěťový měnič s nastavitelným napětím v rozsahu 10,8 V – 12,8 V, jenž je galvanicky izolován od přívodních vodičů s izolací až 1500 Vdc.

Ve výchozí konfiguraci, jenž je pro mé využití ideální, bude tedy výstup nastaven na 12 V. Těchto 12 V bude nadále lineárními regulátory přeměněno na nižších 5 V a 3.3 V pro napájení samotného mikroprocesoru.

Tlačítko a ovládání dveří

Tlačítko, kterým bude prováděno zvonění, jsem se rozhodl provést formou kapacitivního dotykového tlačítka. Periferie pro správnou detekci dotyků je již součástí mikroprocesoru

[2]MIPI CSI – Camera Serial Interface vytvořen orgranizací Mobile Industry Processor Interface. Sjednocuje komunikaci mezi procesory a kamerovými senzory

ESP32-P4 a bude tedy stačit napojit plochu na desce plošných spojů k jednomu z GPIO[3] pinů.

Otevření vrat pak bude prováděno přivedením jednoho z napětí na výstupní svorky zařízení. Toto napětí bude spínáno za pomoci relé, jehož vstup bude připojen na výstup hlavního měniče generujícího 12 V.

Kamera

Připojení kamerového senzoru bude provedeno sběrnicí MIPI CSI dostupnou v mikrokontroléru ESP32-P4. Dostupný je tak široký výběr senzorů, a pro konkrétní model pak stačí pouze implementace ovladače pro jeho řízení. Tento ovladač s kamerou komunikuje po sběrnici I2C. Tato řídící sběrnice společně s obrazovou sběrnicí MIPI CSI a taktéž napájení je kamerovému modulu přivedeno v jednotném konektoru.

Čtečka pro autentizaci

Pro autentizaci uživatel, tedy čtečku čipů, bude použito RFID na frekvenci 13,56 MHz. Konkrétně standard ISO 14443 společně s čipy MIFARE Classic. Tento typ je velmi rozšířený, levný a vzhledem k použité frekvenci je možné navrhnout příslušnou anténu jako trasu na desce plošného spoje.

Pro čtení RFID karet tohoto standardu jsem se rozhodl použít čip MFRC522. Ten podporuje jak čtení, tak zápis do těchto RFID karet ve vzdálenosti až 50 mm s adekvátní anténou. Pro účely řízení poskytuje tři různé typy sběrnic: I2C, SPI či sériový port UART, ze kterých použiji právě I2C kvůli nízkému počtu vyžadovaných GPIO pinů a taktéž protože je tato sběrnice již využita u ostatních připojených periferií.

Ethernet

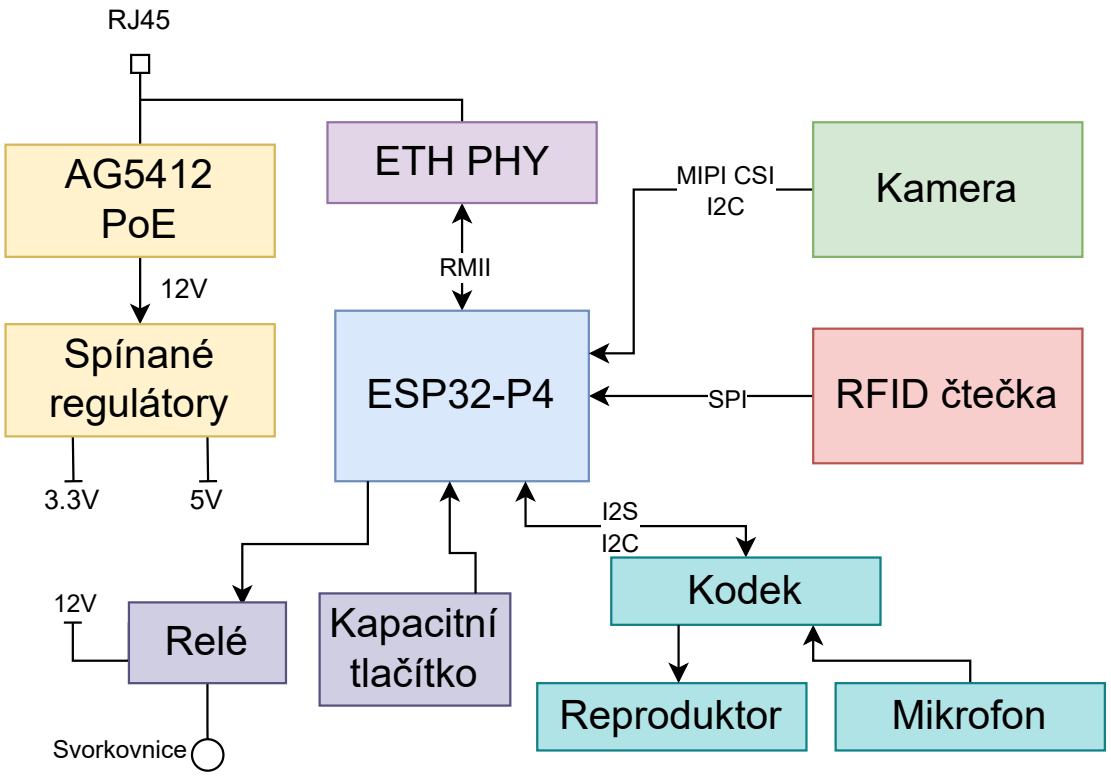
Součástí mikrokontroléru ESP32-P4 je již ethernet MAC periferie. K plnému fungování je však potřeba připojení tzv. ethernet PHY čipu, jenž převádí digitální signály ethernet rámců na diferenciální signály standardu ethernet a zpět. Komunikace mezi ethernet MAC a PHY vrstvami pak probíhá sběrnicí RMII (Reduced Media-independent Interface). Pro dnešní doby by také bylo vhodné, aby takovýto čip podporoval technologii Auto-MDIX, což je automatická detekce zapojení párů v kabelech UTP. Eliminuje se tak potřeba křížených kabelů.

Všechny tyto požadavky splňuje vybraný PHY čip IP101GRI firmy IC Plus. Nejen že má prvotřídní podporu v oficiálním vývojovém prostředí pro mikrokontrolér ESP32-P4, ale také je cenově dostupný a nevyžaduje velké množství součástek.

[3] GPIO – General Purpose Input Output

Blokové schéma HW

Následující obrázek 7.2 zobrazuje blokové schéma zařízení. Jsou naznačeny použité periferie a sběrnice použité pro komunikaci mezi nimi.



Obrázek 7.2: Blokové schéma hardwaru zařízení

7.2 Návrh software

Software zařízení se bude skládat ze dvou částí: obslužný firmware a poté webové administrační rozhraní. Úkolem obslužného firmwaru bude správná inicializace periferií a následné fungování, zatímco administračním rozhraním bude možné měnit parametry tohoto fungování.

RTSP server

Jednou ze základních funkcí bude právě přenos video záznamu z kamery, tak oboustranný přenos zvuku (ze zařízení směrem k Home Assistant integraci, tak z integrace do zařízení), a o tuto funkcionalitu se postará RTSP server v kombinaci s RTP pro přenos na správných transportních kanálech.

Právě díky dostupnosti hardwarového akcelerátoru pro kodek H.264 v mikrokontroléru ESP32-P4 je ideální jako kodek snímaného kamerového záznamu použít právě jej. Pro kódování audia je pak kodek G.711 A-Law vhodnou variantou vzhledem k jeho nízké náročnosti na výpočetní výkon.

Nastavení zařízení

Některé funkcionality zařízení vyžadují změnu parametrů za běhu, a taktéž na dálku. Aby tato nastavení přežila restart zařízení, je zapotřebí je uložit do nevolatilní paměti. V případě použitého mikrokontroléru ESP32-P4 k tomu bude vyhrazeno několik sektorů v paměti flash, v níž se nachází také samotný program.

Nastavitelné budou následující parametry:

- **Síťové rozhraní** – volba použití DHCP klienta nebo statické konfigurace IP adresy, výchozí brány a masky sítě rozhraní. Nastavení `mDNS hostname` pro snadnější vyhledání zařízení na síti
- **Zabezpečení** – heslo pro přihlášení do webové administrace
- **MQTT klient**
  - `URI` MQTT brokeru na nějž se má klient připojovat včetně volby použití zabezpečení `MQTTS`. Společně s touto URI také přihlašovací údaje k brokeru
  - nastavení MQTT témat, které bude klient používat pro komunikaci
  - jméno zařízení, kterým se bude v rámci MQTT komunikace identifikovat
- **RFID**
  - chování čtečky po načtení čipu (verifikace lokálně či `MQTT`, viz v jedné z následujících sekcí *RFID čtečka*
  - seznam identifikátorů RFID karet společně s informací zda-li mají právo otevírat a také uživatelsky nastavitelným jménem pro lepší orientaci
- **Časování** – doba otevření dveří v případě načtení RFID karty a také maximální možná doba, po kterou může být relé sepnuto jako takové

Home Assitant integrace

Integrace pro software Home Assistant umožní ovládání tohoto zařízení dálkově po síti. Základní funkcionalita se bude skládat ze tří Home Assistant entit popsaných v následujícím seznamu:

- **Zámek** – entita typu `switch` reprezentující aktuální stav odemknutí dveří a tedy zda-li je sepnuté relé. Může mít dva stavy: `ON` či `OFF`
- **Skener karet** – entita typu `tag`. Nemá stav ale pouze vyvolává událost na kterou je možné reagovat automatizací. Součástí této události je vždy načtený identifikátor karty
- **Zvonkové tlačítko** – entity typu `device_automation` s podtypem `trigger`. Taktéž nemá žádný stav, událost je vyvoláná po zmáčknutí tlačítka na zařízení

Pro implementaci této integrace jsem se rozhodl využít funkcionality `MQTT discovery`. Není tedy zapotřebí implementovat žádný kód pro běh v Home Assistantu, veškerá konfigurace proběhne zaregistrováním entit po protokolu MQTT, a to zasláním konfiguračních zpráv.

Tato zpráva je zasílána na MQTT téma `{discoveryprefix}/device/{id}/config`. Kde `id` je volitelné a slouží primárně k snažší identifikaci MQTT zpráv (ideální by tedy mělo být unikátní napříč celým MQTT brokerem). Obsahem musí být základní informace o zařízení jako takovém, společně se seznamem entit, jenž budou do Home Assistenta přidány. Součástí definice těchto entit je vždy také jedna či více cest specifikující využité MQTT témata pro komunikaci mezi klientem a Home Assistant integrací.

Firmware zařízení se tedy připojí k nastavenému MQTT brokeru, zašle konfigurační zprávy, nastaví odběr na všechna uvedená témata a následně už jen reaguje/odesílá změny.

Webová API

Komunikaci zařízení s webovou administrací bude zajišťovat HTTP API s jednoduchým REST rozhraním. Díky ní bude možné měnit konfiguraci či například vydat příkaz pro otevření dveří.

Vzhledem k tomu, jakou kontrolu nad zařízením tato API má, je zapotřebí ji chránit před zneužitím autentizačními mechanismy, jako je přihlášení. Ověření probíhá zasláním hesla, na jehož základě je vrácen autentizační token – ten je následně přikládán ke každému dalšímu požadavku. V případě nevalidního či chybějícího tokenu musí být požadavek odmítnut.

Protože se neočekává, že by konfiguraci provádělo více osob najednou, přihlášení automaticky způsobí zrušení platnosti předchozího autentizačního tokenu.

Webové administrační rozhraní

Pro správu zařízení bude firmware poskytovat přehledné administrační rozhraní. Díky použití mDNS pak bude jednoduše dostupné přímo na URL `http://hostname/`.

Celé rozhraní bude poskytováno stejným HTTP serverem, který už poskytuje webové API a z důvodů jednodušší udržitelnosti a nižší vytíženosti samotného mikrokontroléru se bude jednat o single-page aplikaci. Zařízení tak při první návštěvě zašle webovému prohlížeči veškeré potřebné soubory pro její fungování. Požadavky pro načtení a uložení hodnot budou zasílány právě dříve zmiňovanému REST API, přičemž každá provedená operace bude doplněna zpětnou vazbou v podobě oznámení o úspěchu či chybě.

Celé rozhraní by mělo být přizpůsobitelné velikosti obrazovky, aby bylo možné zařízení spravovat také na mobilních telefonech. Součástí rozhraní budou sekce pro konkrétní kategorie konfigurace – každá se svou vlastní cestou URL a navigace mezi těmito sekcemi bude probíhat skrze menu. Mírně složitější sekcí pak bude správa databáze RFID identifikátorů, která zahrnuje funkcionalitu pro jejich vytvoření, modifikaci a také odstranění.

Protože API vyžaduje autentizaci, bude součástí rozhraní také přihlašovací stránka se vstupem pro heslo a také tlačítko pro odhlášení.

RFID čtečka

Pro otevírání dveří autorizovanými osobami bude sloužit právě RFID čtečka. Ta bude neustále skenovat a v případě přiložení validní RFID karty je získán její unikátní identifikátor. Následující postup pak závisí na aktuálně nastaveném režimu verifikace. K dispozici jsou tři režimy ověřování:

- **Lokální** – po načtení je RFID identifikátor porovnán s lokální databází. Pokud má karta právo k přístupu, je zaslán příkaz k otevření stavovému automatu (viz. následující sekce *Chování zvonku*
- **MQTT** – po načtení je identifikátor karty odeslán Home Assistant entitě *Skener karet*. Poté záleží na uživatelem nastavených automatizacích zda zašle entitou *Zámek* příkaz k otevření
- **Kombinovaný režim** – spojení obou předchozích. Karta je ověřena lokálně a zároveň zaslán její identifikátor Home Assitant entitě. V této konfiguraci je toto zaslání spíše pro informační účely, například pro účely logování

Chování zvonku

Pro správné otevírání dveří a také reakce na zvonek se budou starat dva jednoduché asynchronní stavové automaty řízené událostmi.

Automat obstarávající spínání relé (a tedy otevření dveří) bude mít dva stavy, zavřeno (relé vypnuto), jenž je stav výchozí, a otevřeno (relé sepnuto). K přepnutí do stavu otevřeno může dojít pouze třemi událostmi. První z těchto událostí je načtení validní RFID karty s právem otevírat. Druhou událostí je pak změna *MQTT switch* entity na `ON` a poslední možností otevření je příkaz z webového rozhraní.

Vypnutí manuálně lze provést pouze přepnutím *MQTT* entity nebo opět webovým rozhraním. Ve všech případech je však maximální doba ve stavu `Otevřeno` omezená na konkrétní časový interval. Ten je zvolen podle použitého způsobu otevření (RFID kartou, MQTT či rozhraním), je konfigurovatelný, a po této době musí být stav přepnut zpět na `Zavřeno`. Tento čas slouží především k ochraně v situaci, kdy dojde k náhodnému přerušení spojení s `MQTT brokerem`. Aktuální stav je však za všech okolností synchronizován se stavem *MQTT switch* entity.

Kapitola 8

Obvodová realizace hardware

Výsledný hardware zařízení byl navržen v programu Altium Designer. Použité schematické značky společně s půdorysy součástek byly vytvořeny za pomocí nástroje *IPC Compliant footprint wizard* jenž je součástí programu Altium Designer.

8.1 Návrh desky plošného spoje

Proveden byl schematický návrh, jenž byl rozdělen do celkově 7 schematických souborů kvůli lepší přehlednosti. Byl zvolen tzv. *flat* (plochý) design, v němž jsou jednotlivé soubory na stejné úrovni a propojení mezi nimi se dělá za pomocí pojmenovaných portů.

Power Over Ethernet

Jak již bylo zmiňováno v návrhu, pro napájení zařízení byl zvolen modul AG5412. Jeho vstup však vyžaduje již usměrněné napětí, čehož bylo docíleno použitím dvou integrovaných usměrňovacích můstků, konkrétně modelu `MB1S`. Ty jsou zapojeny na páry CT1,CT2 a SP1,SP2 vyvedené z konektoru RJ45. V mém případě jsem použil konektor s integrovanou magnetikou pro ušetření místa na desce.

Na výstup PoE modulu byl zapojen elektrolytický kondenzátor s kapacitou $470\mu F$ a maximálním napětím 16 V, tak jak doporučuje katalogový list. Pro indikaci stavu napájení je pak přidána červená LED.

Napájecí větve

Zařízení vyžaduje dvě různá napájecí napětí pro své fungování. Prvním z nich jsou 3,3 V pro napájení mikrokontroléru, RFID čtečky, fyzického rozhraní ethernet a audio kodeku. Audio zesilovač na výstupu kodeku pak požaduje 5 V.

O jejich tvorbu na zařízení byly navrženy dva spínané regulátory napětí s použitím čipu AP62250. Jedná se o čip v poměrně malém TSOT26 pouzdře a společně s nízkým počtem externích součástek zabírají na desce méně místa.

Jedná se o nastavitelné regulátory, kde výstupní napětí závisí na použité napěťové děličce připojené k zpětné vazbě. Hodnota rezistoru R2 je typicky určena fixní a požadovaný rezistor R1 je dopočítán dle následujícího vzorce č. 8.1, kde V_{OUT} je požadované výstupní napětí.

$$R_1 = R2 \times \left(\frac{V_{OUT}}{0,8V} - 1\right) \tag{8.1}$$

Jako R2 jsem tedy zvolil 10 kΩ, díky čemuž po dosazení vychází rezistory R2 pro daná napětí následovně, tak jak ukazují rovnice 8.2 a 8.3.

$$\text{Pro } 3,3V: \quad R_1 = 10k\Omega \times \left(\frac{3,3V}{0,8V} - 1\right) = 31,25k\Omega \tag{8.2}$$

$$\text{Pro } 5V: \quad R_1 = 10k\Omega \times \left(\frac{5V}{0,8V} - 1\right) = 52,5k\Omega \tag{8.3}$$

Další důležitou součástí obvodu regulátoru je cívka, jejíž induktanci je možné v závislosti na vstupním a výstupním napětí a také očekávaným proudovým odběrem. Doporučenou hodnotou induktance pro výstupní napětí 5V a 3,3V je dle katalogového listu $2,2\mu H$.

Zapojení ESP32-P4

Mikrokontrolér vyžaduje pro své fungování 3,3 V, které jsou přivedeny z každé strany pouzdra. Kvůli stabilitě napájení jsou u každého přívodu použity lokální oddělovací kondenzátory o velikostech $100nF$ a $10\mu F$.

Součástí pouzdra jsou také čtyři regulátory se softwarově nastavitelným výstupním napětím. Tyto regulátory jsou z pouzdra vyvedeny na pinech VO1-VO4 a každý z výstupů je také opatřen oddělovacími kondenzátory. Jejich zapojení jsou provedena následovně:

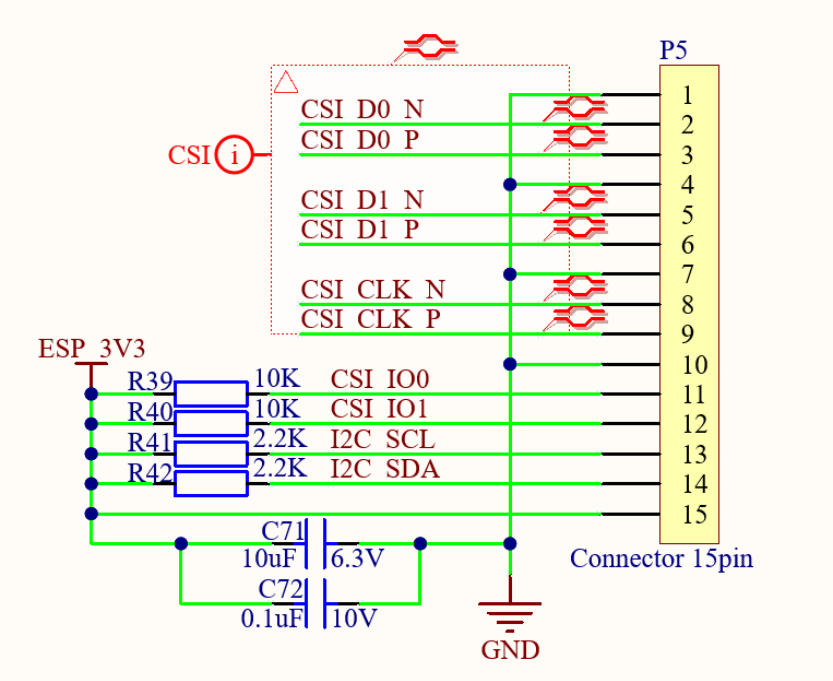
- **VO1** – napájí připojenou NOR flash (3,3V)
- **VO2** – přiveden do pouzdra jako zdroj napětí pro interní PSRAM (1,8V)
- **VO3** – přiveden do pouzdra pro napájení MIPI periferie (2,5V)
- **VO4** – napájí GPIO piny vyhrazené pro komunikaci s RFID čtečkou (3,3V)

Mikrokontrolér ESP32-P4 neobsahuje vlastní oscilátor a tudíž je povinné připojení externího 40MHz krystalu společně se zátěžovými kondenzátory na vyhrazené piny `XTAL`. Použitý krystal má pouzdro SMD3225-4P.

Součástí desky musí být také paměť pro firmware zařízení ve formě NOR flash. Zvolena byla flash výrobce GigaDevice GD25Q128E o velikosti 16 MiB a s mikrokontrolérem komunikuje po sběrnici Dual SPI.

Do schématu byl přidán 15pinový konektor, na nějž jsou přivedeny sběrnice I2C a MIPI CSI. Poslední zmiňované rozhraní pracuje na vyšších frekvencích (až stovky MHz) a využívá diferenciálních párů. Každý z těchto párů (CSI_D0, CSI_D1 a CSI_CLK) byl ve schématu pojmenován tak, aby jej program Altium Designer považoval právě za páry diferenciální a mohlo se s nimi poté lépe pracovat při návrhu desky.

Jméno jednoho z vodičů tak musí končit příponou **_P** zatímco druhý je pojmenován se zakončením **_N**. Na každý vodič je pak ještě nutné umístit modifikátor *Differential pair*. Zapojení celého konektoru s uvedenými diferenciálními páry ukazuje následující obrázek 8.1. Modifikátory *Differential pair* jsou červené objekty vyskytující se u každého z CSI vodičů.



Obrázek 8.1: Zapojení MIPI CSI párů konektoru

Relé

Protože je relé proudově náročnější součástka, nelze jí spínat jednoduše z GPIO pinu mikrokontroléru. Rozhodl jsem se tedy použít N-Channel mosfet AO3400, jenž spíná zem a na jehož bránu je připojen výstup z mikrokontroléru. Mezi bránou a mikrokontrolérem se ještě nachází rezistor o hodnotě 200Ω pro omezení tekoucího proudu z GPIO pinu. Protože je relé ve své podstatě cívka, dochází po odpojení napájení k indukci napětí, jenž může poškodit spínací mosfet. Z toho důvodu je paralelně s relém připojena dioda v opačném směru. Indukované napětí je tak svedeno přes diodu a nemůže dojít k ohrožení zbytku obvodu.

Ethernet PHY

Čip fyzické vrstvy ethernet IP101GRI komunikuje s mikrokontrolérem rozhraním RMII (*Reduced media-independent interface*). V RMII režimu je tento čip interně nastaven tak, aby generoval na svém výstupu `50M_CLKO` hodinový signál o frekvenci 50 MHz. Ten je následně používán jako zdroj signálu pro oboustrannou komunikaci s mikrokontrolérem – je přiveden jak do pinu `50M_CLKI` tak do mikrokontroléru. Externě je k čipu IP101GRI připojen 25 MHz krystal, na jehož základě je právě tento hodinový signál generován.

Konfigurace čipu je prováděna připojením specifických pinů za pomocí pull-up či pull-down rezistorů. Konkrétně pin `COL/RMII` musí být připojen pull-up rezistorem k napájení, aby byl použit RMII režim. Dále piny `PHY_AD3` a `PHY_AD0` nastavují adresu, se kterou se bude s čipem komunikovat.

Ke konektoru RJ45 je připojen čip přímo, a to dvěma diferenciálními páry. Stejně jako v případě rozhraní MIPI CSI je tedy pro tyto diferenciální páry ve schématu použito správné pojmenování společně s modifikátorem *Differential pair*.

Ačkoliv je čip podporuje, rozhodl jsem se nepoužívat RJ45 konektor se signalizačními LED, neboť zařízení bude z velké části přiloženo ke stěně a diody nebudou viditelné.

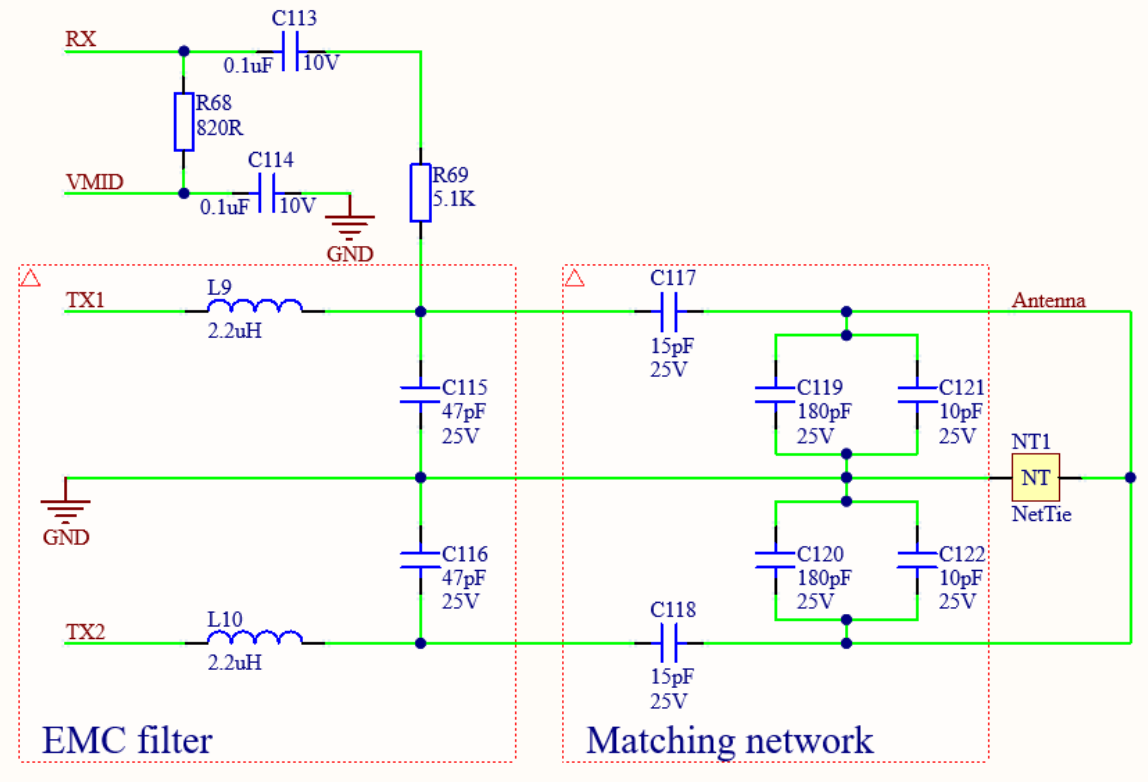
RFID

Pro komunikaci s RFID kartami byla vybrána velmi rozšířená čtečka karet MFRC522 firmy NXP. Čtečka je napájena 3,3V, jenž je přivedeno celkově na čtyři napájecí piny čipu. Toto napájení je doplněno lokálními oddělovacími kondenzátory.

Kromě připojení sběrnicí SPI je součástí schématu také přiveden vodič pro reset zařízení připojený pull-up rezistorem k 3,3V větvi. Čtečka je schopna komunikovat také za pomoci sběrnice I2C, což v mém případě není zapotřebí. Je však nutné připojit pin `I2C` k zemi, aby byl tento režim vypnut a bylo možné použít pouze SPI.

Podobně jako v některých z předchozích periferií je také k RFID čteččce připojen krystal, tentokrát o frekvenci 27,12 MHz. Z výstupu oscilátoru je pak interně generována nosná vlna o frekvenci 13,56 MHz použitá pro vysílání.

Velmi důležitou částí čtečky je její anténa, jejíž design doplňuje také několik pasivních součástek pro správné fungování na daných frekvencích. Toto zapojení je zobrazeno na následujícím obrázku 8.2.



Obrázek 8.2: Schéma zapojení RFID antény

Jak lze vidět, součástí designu je EMC filtr pro snížení nežádoucích emisí rušení do okolního prostoru. Jedná se o dolní propust s mezní frekvencí vypočtenou na základě vzorce 8.4 s hodnotami dosazenými ve vzorci 8.5. Výslednou frekvenci lze vidět ve vzorci 8.6.

$$f_c = \frac{1}{2\pi\sqrt{L \times C}} \tag{8.4}$$

$$f_c = \frac{1}{2\pi\sqrt{(2.2\mu H) \times (47pF)}} \tag{8.5}$$

$$f_c = 15.65MHz \tag{8.6}$$

Konkrétní hodnoty induktorů a kondenzátorů byly vybrány experimentací s vysoce dostupnými hodnotami součástek, a to tak, aby byla mezní frekvence vyšší než frekvence vysílání.

Dále je součástí obvodu kompenzační síť (*matching network*) jejímž úkolem je maximalizace přenosu energie z vysílače do antény. Použité hodnoty kondenzátorů závisí na impedanci použité antény. Dle katalogového listu by měla být celková impedance na výstupu vysílače právě 50Ω. Kondenzátory C119 a C121 jsou zapojeny tak, aby bylo možné jejich výslednou kapacitu doladit paralelním zapojením různých hodnot. Stejné platí také pro dvojici C120 a C122.

Výpočet těchto hodnot kompenzační sítě je však složitý a pro přesné odladění vyžaduje velmi dobrou znalost vlastností konkrétní použité antény. Firma NXP poskytuje pro tyto účely online nástroj[1] v němž je možné zadat přímo fyzický vzhled antény, syntetizovat její vlastnosti a na jejich základě vypočíst hodnoty součástek.

Syntetizované vlastnosti se však mohou výrazně lišit od skutečných, které je ideální změřit vektorovým analyzátorem. Protože se jedná o dražší nástroj, rozhodl jsem se použít fyzický návrh antény z již existujících modulů MFRC522 a to včetně použitých hodnot kompenzační sítě.

Audio kodek a zesilovač

Pro přehrávání audio záznamů byl využit kodek ES8311 společně se zesilovačem NS4150B. Kodek kromě audio výstupu integruje také analogově-digitální převodník, na nějž je připojen diferenciální mikrofon.

Protože je kodek citlivé analogové zařízení, je pro jeho převodníky vyhrazen regulátor napětí s nízkým úbytkem (LDO) s výstupním napětím 3,3V. Je tak zamezeno napájení převodníků napětím s velkým šumem, který by se na napájecí větvi vyskytoval v případě použití spínaného regulátoru. Dále je celá napájecí větev pro audio zařízení oddělena přemosťovacími rezistory tak, aby nedocházelo k šíření šumu z digitálních částí zařízení.

Zesilovač je pak napájen 5 V vyvedenými ze spínaného regulátoru popsaného v sekci o napájecích větvích.

8.2 Rozvržení plošného spoje

Finální deska zařízení má velikost 6,8 cm na šířku a 12,7 cm na výšku. V horní části se nachází výřez pro prostorově úspornější umístění kamery.

Vytvořena byla 4-vrstvá deska s jádrem z materiálu FR-4. Horní a spodní vrstva byly vyhrazeny pro trasy signálů, příležitostně pro napájení. Druhá vrstva od zhora je vyhrazena čistě pro zemi, konkrétně pro společnou digitální a také analogovou zem. Třetí vrstvou jsou pak vedeny napájecí větve, jako jsou 5V, 3,3V, či 12V, přivedené z PoE modulu.

Země je v druhé vrstvě rozvedena za pomocí rozlité mědi – v programu Altium Designer pojmenováno *Polygon pour*. Hlavní měď pro digitální komponenty pokrývá takřka celou plochu desky vyjma prostoru pod anténou,částí s RJ45 konektorem a PoE modulem. Druhá rozlitá měď se nachází v pravé části desky pod audio komponentami a je oddělená přemosťovacím rezistorem z důvodu izolace šumu.

[1] https://community.nxp.com/t5/NFC/bd-p/nfc

Na třetí vrstvě je použita rozlitá měď pouze pro 3,3V větev a zbytek napěťových větví je rozveden použitím širších tras. Například 12V větev využívá šířky trasy 1,27 mm a 5V pro změnu 0,762 mm.

Na horní a spodní vrstvě je také z důvodu lepšího chlazení PoE modulu použit kus rozlitě mědi u jeho výstupu, kde zároveň usnadňuje proudu cestu mezi modulem a jeho výstupním kondenzátorem.

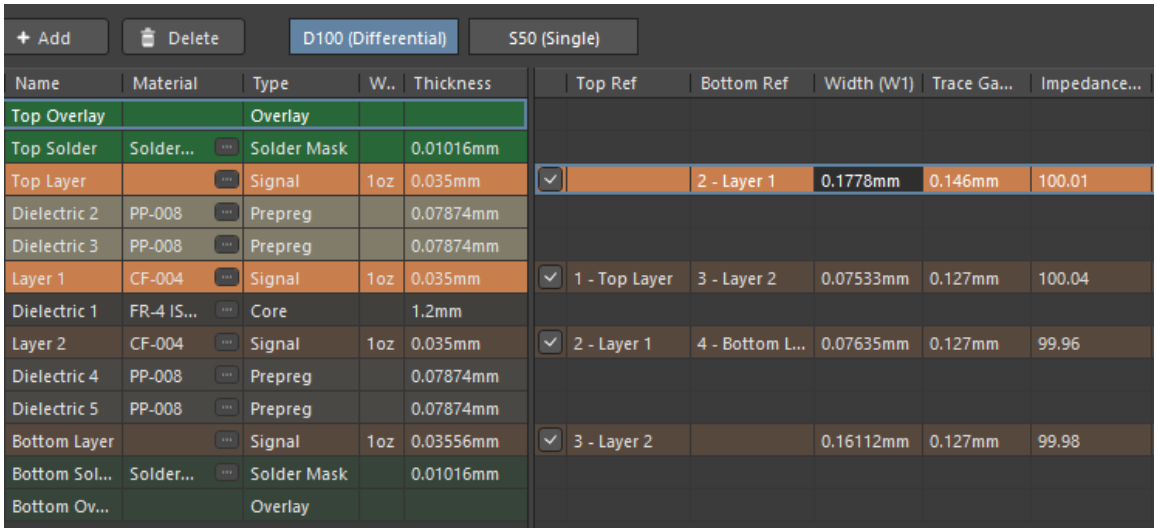
Spínané regulátory

Rozvržení spínaných regulátorů napětí pro jednotlivé větve bylo provedeno dle doporučení v jejich dokumentaci. Jejich cívka byla umístěna hned vedle čipu regulátoru společně s vstupními a výstupními kondenzátory o velikosti 0603. Pro cívky s indukčností $2.2\mu H$ bylo zvoleno pouzdro o velikosti 3mm x 3mm. Pro propojení vstupního napětí, výstupu a také cívky se spínaným pinem regulátoru bylo využito rozlitých mědí pro zlepšení tepelných vlastností regulátoru. Výstupní napětí je pak propojeno s napájecí větví za pomoci prokovů v desce.

Diferenciální páry

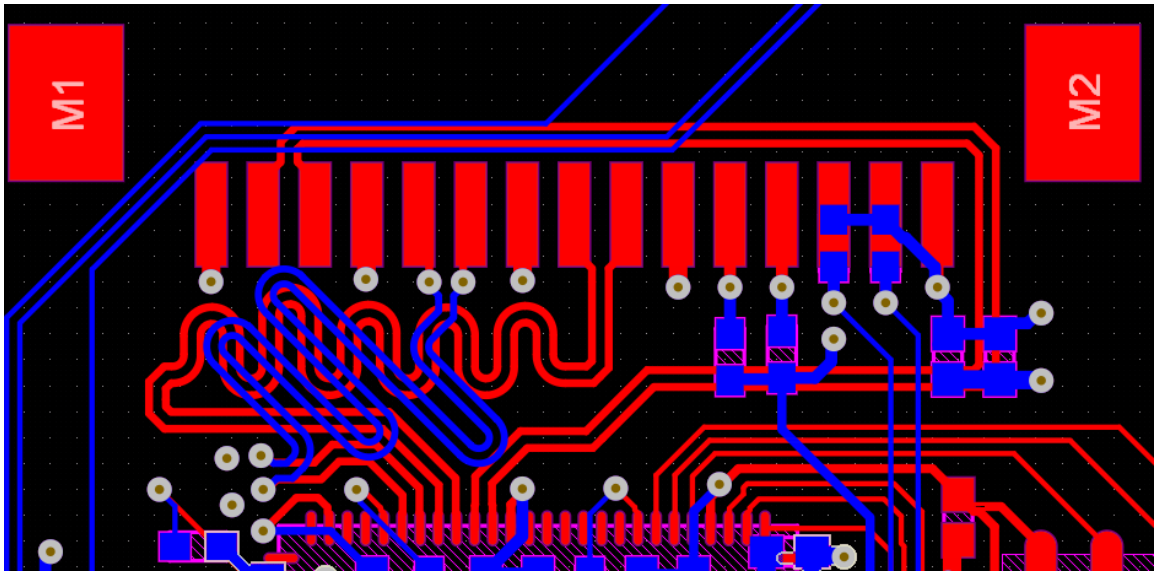
Protože je MIPI CSI vysokofrekvenční datové rozhraní a integrita přenosu tak závisí na správném návrhu signálových tras, bylo při rozvržení těchto tras použito nástrojů pro výpočet impedance. Program Altium Designer obsahuje pro tyto účely již připravené utility. Do tzv. *Layer stack manageru* bylo vloženo očekávané uspořádání vrstev výrobce desky společně s vlastnostmi a tloušťkami použitých materiálů.

Na jejich základě pak byla nástrojem pro výpočet impedance vypočítána jak šířka tras, tak mezera mezi nimi. Rozhraní MIPI CSI požaduje diferenciální impedanci 100Ω. Po dosazení této impedance do nástroje byla vypočtena šířka 0,1778 mm s mezerou 0,146 mm. Jako referenční plocha byla zvolena právě rozlitá zem na druhé vrstvě desky. Na základě výpočtu byl vytvořen impedanční profil `D100 (Differential)`. Následující obrázek 8.3 zobrazuje použitý nástroj výpočtu společně s uspořádáním vrstev desky. Výpočet impedance je na zvýrazněném řádku v pravé části nástroje.



Obrázek 8.3: Uspořádní desky a výpočet impedance

Při trasování všech diferenciálních párů nástrojem *Interactive Differential Pair Routing* byl vytvořen impedanční profil, zvolen a nástroj se postaral o to, aby byla dodržena daná šířka a mezera tras. Dodržení impedance však není jediné, co je potřeba pro správné rozvržení těchto signálů. Jednotlivé páry musí navzájem dosahovat také stejné délky, aby při komunikaci docházelo k synchronnímu příchodu signálů. Proto bylo také využito nástroje pro interaktivní optimalizaci délky jednotlivých párů. Tato optimalizace lze vidět na následujícím obrázku 8.4 kde jsou použity meandry pro jejich prodloužení.



Obrázek 8.4: Trasy MIPI CSI rozvrženy dle vypočtené impedance a optimalizovány na stejnou délku

RFID anténa

Anténa čtečky karet byla převzata z návrhu hobbisty velmi oblíbeného Arduino modulu MFRC-522 RC522. Pro toto řešení jsem se rozhodl z toho důvodu, že návrh a následné odladění kompenzační sítě vyžaduje velké množství úsilí a také vektorový analyzátor.

Použitá anténa má tedy rozměry 4 cm x 4 cm, nachází se na spodní vrstvě desky, směrem k uživateli. Na stejné vrstvě byla také položena kompenzační síť vytvořená z kondenzátorů.

Finální rozvržení desky po jednotlivých vrstvách lze vidět v příloze D.

Kapitola 9

Implementace programové části

Implementaci obslužného firmwaru jsem se rozhodl provést ve frameworku ESP-IDF poskytovaném přímo výrobcem mikrokontroléru, tedy firmy Espressif. Tento framework obsahuje kromě kompilátorů pro příslušnou architekturu také nástroje pro ladění a nahrání firmwaru do mikrokontroléru.

Každý firmware využívající ESP-IDF v základu obsahuje operační systém reálného času FreeRTOS. Využívá jej většina vestavěných komponent pro práci s periferiemi nebo službami. S jeho poskytnutými synchronizačními prostředky je tak snadnější psát bezpečný víceúlohový kód.

Primárním jazykem mé implementace je C++, pro který má framework velmi dobrou podporu, a to včetně moderního standardu C++23. Díky této podpoře je k dispozici například práce s vlákny třídou `std::thread` ze standardní knihovny. Tuto třídu jsem využil obzvláště v situacích, kdy je zapotřebí vytvořit vlákno, které je součástí C++ třídy. Standardní FreeRTOS úlohy totiž počítají pouze s jazykem C a vytvoření úlohy s ukazatelem na instanci třídy tak vyžaduje vytváření tzv. wrapper funkcí.

V práci využívám primárně vestavěné komponenty, společně s komponentami dostupnými z *ESP Component registry*. Velká část z nich je vyvíjena širší komunitou kolem platformy ESP32.

Mezi tyto komponenty patří:

- *esp_cam_sensor* – poskytuje funkce pro detekci a správu kamerových senzorů připojených přes sběrnici I2C. Pro každý z podporovaných senzorů jsou k dispozici funkce a konfigurační struktury umožňující jeho nastavení
- *esp_h264* – ovladač hardwarového akcelerátoru kódování H.264. Společně s ním je obsažena také softwarová implementace, tu však v práci nepoužívám, neboť je pomalejší. Ovladač je použit pro kódování videa z kamerového senzoru
- *abobija/rc522* – ovladač senzoru MFRC522 pro čtení RFID karet
- *espressif/mdns* – služba mDNS pro snadnější nalezení zařízení na síti pomocí názvu zařízení
- *etl* – embedded template library. Alternativa velké části standardní knihovny C++ (STL) pro vestavěné zařízení. Poskytuje alternativy kontejnerových tříd s fixním využitím paměti – neprovádí žádné dynamické alokace – velikost musí být dopředu známa a definována jako parametr šablony třídy.

Výhodou je také třída `delegate<TReturn(TParams...>`, což je bezalokační náhrada `std::function`. Takto jsem schopen předávat ukazatele obslužných rutin jednotlivých událostí a to i těch jenž se vyskytují v třídách

- *ArduinoJSON* – malá a efektivní knihovna pro de/serializaci JSON řetězců

Využité periferie

Firmware musí při svém startu nastavit několik sběrnic či periferií nezbytných pro správné fungování systému. U některých z nich je konfigurace prováděna přímo mnou samotným použitím vestavěných komponent, zatímco ostatní jsou spravovány uvnitř ovladačů pro některá z externích zařízení. Některé z těchto periferií jsou dokonce sdíleny mezi externími zařízeními.

Následující seznam obsahuje všechny použité a konfigurované periferie společně s jejich účelem:

- **I2C** – použit I2C port 0 ke komunikaci s audio kodekem a zároveň nastavení parametrů kamerového senzoru
- **I2S** – přenos digitálního zvuku mezi mikrokontrolérem a audio kodekem
- **SPI** – SPI3 ke komunikaci s čtečkou RFID karet
- **MIPI CSI** – přenos obrazových dat z kamerového senzoru. Rozlišení přenášeného obrazu je nastaveno sběrnicí I2C
- **ETH MAC** – ethernet MAC periferie je přímo připojena k externímu čipu fyzické vrstvy. Během inicializace je zapotřebí správně nakonfigurovat jejich propojení a použitý hodinový signál

Struktura projektu

Vzhledem k tomu, že se jedná o projekt ve frameworku ESP-IDF, je při založení projektu připravena základní struktura. Jedná se o dvě složky s názvy `components` a `main`.

Obsaženy v projektu jsou pak také soubory definující kompilační parametry, jako třeba `CMakeLists.txt`. Velmi důležitou částí je soubor `sdkconfig`. Ten konfiguruje prostředí, jehož obsah je ve formátu `PROMENNA=hodnota`, a každá z těchto proměnných je pak pro C/C++ kód definována makrem `#DEFINE`.

Za zmínku stojí také `partitions.csv` ve formátu CSV specifikující rozdělení flash paměti na jednotlivé oddíly. Použité rozdělení popisuje následující tabulka 9.1:

Č.	Jméno	Typ	Subtyp	Velikost
1	nvs	data	nvs	128 KiB
2	otadata	data	ota	8 KiB
3	factory	app	factory	4 MiB
4	ota_0	app	ota_0	4 MiB
5	ota_1	app	ota_1	4 MiB

Tabulka 9.1: Tabulka rozdělení paměti flash

Oddíly factory, ota_1 a ota_2 jsou `app` oddíly obsahující firmware zařízení, zatímco oddíl nvs slouží jako úložiště nastavení zařízení.

Složka `main`

Obsahuje hlavní funkčnost celého zařízení. Nacházejí se zde vstupní bod programu s funkcí `main`, a to v souboru `bell_main.cpp`. V něm jsou vytvářeny všechny potřebné třídy obsahující služby a úlohy, jenž poté na zařízení běží.

Soubor `idf_component.yml` kromě minimální použité verze ESP-IDF frameworku udává požadované závislosti jako právě dříve zmiňované komponenty z *ESP Component registry*.

Zbytek souborů ve složce se týká už samotné implementace jednotlivých funkcí firmwaru a věnují se jím příslušné sekce.

Složka `components`

Tato složka je určena pro jednotlivé komponenty, převážně pro zlepšení organizace zdrojového kódu. Téměř všechny komponenty v této složce byly vytvořeny mnou s výjimkou právě `etl` a `arduinoJson`, což jsou komponenty, které je zapotřebí instalovat manuálně, neboť je jejich autoři neposkytují v registru komponent. Mé komponenty jsou taktéž popsány v příslušných sekcích textu dále.

Před samotnou kompilací, v momentě kdy se spustí příkaz `build` ESP-IDF build systému, je v kořenové složce projektu vytvořena složka `managed_components`. Do ní jsou z registru komponent staženy všechny deklarované komponenty, aby mohly být nadále zahrnuty při kompilaci. Do zdrojových kódů v těchto komponentách není doporučeno nic měnit – kdykoliv může dojít k vymazání těchto změn build systémem.

9.1 RTSP server

Pro přenos video a audio záznamů je implementován RTSP server. Před implementací jsem prozkoumal možné knihovny, jenž by RTSP server pro platformu ESP32 a framework ESP-IDF poskytovaly, ovšem žádná nesplňovala mé požadavky na souběžný přenos video a audio záznamů. Jednou z nich je například knihovna Micro-RTSP[1] od autora s *geeksville*. Samotná se však specializuje pouze na přenos videozáznamů. Jako upravená verze této knihovny s podporou audia se nabízí Micro-RTSP-Audio[2], ovšem její autor pouze upravil původní knihovnu tak, že přenos videa kompletně nahradil. Nabízela se tedy možnost využít obou knihoven pro vytvoření kombinace s funkcionalitou obou. Po zhlédnutí kódu obou knihoven jsem však zjistil, že kód není snadno rozšiřitelný, a tak jsem se rozhodl RTSP server implementovat vlastní.

Pro implementaci je vyhrazena komponenta *rtsp_server* a celý server se skládá z následujících částí: soketový server, parser požadavků, zpracovávač požadavků a transportní vrstvy pro video a audio. Každá ze tříd je psána v C++ a všechny třídy mají odstraněný výchozí konstruktor kopírování. Je tak zajištěno, že nedojde omylem ke kopii instance, která obsahuje ukazatele na nějaké z jiných objektů. Kompilátor vždy vyhodí chybu v situacích, kdy se o předání instancí pokusím jinak než referencí.

K serveru je možné před jeho startem přidružit jednotlivé stopy, tzv. *tracks*. Stopy můžou být různého typu – H.264 video či audio stopy kodeku G.711. Každá ze stop musí

[1] https://github.com/geeksville/Micro-RTSP
[2] https://github.com/pschatzmann/Micro-RTSP-Audio

být potomkem třídy `Track` a musí sama implementovat funkce pro zasílání dat přidruženou transportní vrstvou.

Soketový server a příjem klientů

Implementován třídou `RtspControlServer`, tento server se stará o správu připojených klientů a komunikaci s nimi. Framework ESP-IDF pro práci se sokety využívá knihovnu lwIP (*A Lightweight TCP/IP stack*), jenž poskytuje standardní API ve formě BSD soketů. Po startu je tedy vytvořen TCP soket a příkazy jako `bind` a `listen` zahájí příjem klientů.

Pro čekání na klienty je vytvořeno zvlášť vlákno, jenž příkazem `accept()` vrátí vždy nově připojeného klienta. Pokud počet připojených klientů nepřesahuje limit, je vytvořena nová instance třídy `RtspPeer` se socketem nově připojeného klienta a pro zpracovávání je mu z fondu vláken jedno pracovní vlákno přiřazeno. Pokud žádné není dostupné, klient je odpojen se zprávou `Server busy`.

Ve svém pracovním vlákně je nejdříve prováděno čekání na požadavek ze strany klienta. Po přečtení je požadavek parsován a je spuštěna obslužná rutina `RequestHandler` s daty požadavku a instancí pomocného objektu pro zasílání odpovědí `RtspResponder`. Toto chování se opakuje, dokud klient neukončí spojení, nedojde k interní chybě serveru, či server vyžádá ukončení spojení na základě chybného formátu požadavku.

Vzhledem k podpoře prokládáného TCP transportního spoje je nutné před každým začátkem zpracování zjistit, zda-li klient nezaslal znak `$`. V takovém případě se totiž nejedná o požadavek, ale o data vložená do prokládaného kanálu. S takto přijatými daty se pracuje mimo základní smyčku a více jsou popsána později v sekci 9.1 o přenosu videa a audia.

Parsing požadavků

Pro snadnější parsing požadavků jsem implementoval pomocnou třídu `RtspFormatReader`, která si uvnitř sebe zachovává aktuální stav čtení. Stav je tvořen textovým řetězcem o maximální délce 1024 znaků a pohledem do tohoto řetězce vyznačujícím poslední přečtená data (proměnná `lastViewEnd`). Každá instance patří právě jednomu připojenému klientovi, z jehož soketu jsou data čtena. K tomuto čtení je mu poskytnuta v konstruktoru funkce `int recv(char*,int)`.

Implementovány byly následující funkce:

- `PeekOne(char& out)` – podívá se jaký znak se nachází na vstupu. Tento znak je na vstupu ponechán
- `ReadUntilCRLF(etl::string_view& outputLine)` – přečte řádek ze vstupu. Musí končit kombinací bílých znaků `CRLF`
- `ReadContent(size_t length, etl::istring& output)` – přečte specifikovaný počet znaků ze vstupu do daného výstupu
- `DumpBuffer(size_t length)` – ze vstupu odstraní specifikovaný počet znaků. Data jsou kompletně zahozena

Všechny tyto funkce pracují s interním řetězcem a pokud možno, data jsou čtena z něj. V případě, že dojde k vyprázdnění tohoto řetězce, kód se pokusí o přečtení nových dat ze soketu klienta za pomocí `recv`. Ze soketu je vždy čteno maximální možné množství dat tak, aby došlo k naplnění řetězce – druhý argument funkce `recv` je délka určená odečtením aktuální délky od té maximální. Přečten však může být menší počet znaků.

Všechny výše uvedené čtecí funkce fungují v základu stejně – nejprve se pokusí ke splnění svého cíle využít data z interního řetězce. Například v případě `ReadUntilCRLF` je proveden pokus o nalezení bílých znaků. Pokud cíl splnit nelze, využije `recv` k přečtení dat dalších, ta vloží do řetězce a proces se opakuje. Po splnění jsou daná data vrácena výstupním argumentem, a zároveň je interní pohled do řetězce `lastViewEnd` nastaven na stejný rozsah. Takto jsem se rozhodl, aby nebylo při volání zapotřebí alokovat žádnou paměť navíc – s daty se pracuje přímo. Protože je samozřejmě po jejich využití nepotřebujeme, při dalším volání jakékoliv funkce je z řetězce daný rozsah znaků odstraněn na základě `lastViewEnd`.

Na základě formátu RTSP požadavků, tak jak je uvedeno v sekci 5.2, je nejdříve přečten úvodní řádek obsahující verzi RTSP, použitou metodu a také URI. Toho se docílí právě `ReadUntilCRLF`. Informace o počátečním řádku jsou uloženy do objektu `RtspRequestParam`. Poté následuje smyčka, jenž čte další řádky, jenž představují hlavičky požadavku.

Každý z řádků s hlavičkami je rozdělen na název a hodnotu, název porovnán se seznamem podporovaných hlaviček, a pokud se jedná o hlavičku jenž je relevantní, je uložena do instance třídy `RtspRequest`. Velmi důležitou hlavičkou, na jejíž zpracování je zapotřebí si dát záležet, je hlavička `Transport`. Ta může vzhledem k různým transportním vrstvám podporovaným standardem RTSP nabývat několika variant a klient jí určuje, jakým způsobem bude probíhat přenos audio a video záznamů.

Konec zpracování požadavku nastává v moment, kdy se po sobě nacházejí dva prázdné řádky, neboli následující kombinace bílých znaků: `<CR><LF><CR><LF>`. Po něm může následovat také tělo požadavku, ovšem pro mnou implementované metody RTSP jeho obsahu není zapotřebí, a pokud se tedy v požadavku nachází hlavička `Content-Length`, je toto tělo zahozeno za pomocí `DumpBuffer(length)`.

Zpracování příkazů

Pokud je celý požadavek úspěšně naparsován a předán ke zpracování, je proveden vstup do kritické sekce RTSP serveru použitím utility `std::lock_guard<std::mutex>`. Příkazy totiž může zasílat více klientů najednou, a bez tohoto zámku by mohlo dojít k souběžnému přístupu k vnitřním proměnným serveru. Protože implementace podporuje také prokládání RTP dat v TCP soketu klienta (popsáno v sekci 5.3), je tímto zámkem zaručeno, že nedojde k zápisu odpovědi na RTSP příkaz, zatímco jsou stejnému klientovi zasílána data obrazu či videa – využívá se totiž stejný soket. Jak je již z předchozí věty zřejmé, před jakýmkoliv odesíláním RTP dat je tedy zamknutí taktéž povinností.

Implementace podporuje tyto požadované RTSP metody: `OPTIONS`, `DESCRIBE`, `SETUP`, `PLAY` a `TEARDOWN`. První z nich je zpracována jednoduše, pouze je odeslána odpověď obsahující podporované metody.

DESCRIBE

Metoda `DESCRIBE` už je však složitější – vyžaduje sestavení řetězce formátu SDP (Session Description Protocol) popisujícího dostupné stopy. Informace o stopě jsou vytvořeny instancí stopy samotnou, a to přepsatelnou (virtual) funkcí `BuildSDP(SDPMessageBuilder&)`. V případě, že klient podporuje oboustranné audio, je v hlavičce `Requires` uveden řetězec `www.onvif.org/ver20/backchannel`. Tímto dává klient najevo, že máme ve výsledném SDP řetězci uvést právě také audio stopy, jenž umožňují zaslání dat od klienta směrem k serveru.

SETUP

Příkaz `SETUP` tvoří první krok klienta k zahájení přenosů dat. Je vytvořeno transportní spojení pro konkrétní kombinaci RTSP sezení a stopy. Sezení jsou vytvářena dynamicky, a slouží k tomu, aby mohl klient ovládat přenosy i po odpojení a znovupřipojení k RTSP serveru. Server tato sezení odlišuje pomocí *Session identifier*. Typ transportní vrstvy je určován hlavičkou `Transport`, a má implementace podporuje buď UDP, či prokládané TCP. Součástí vytvoření UDP spojení je také alokace portů z předem daného seznamu. Tyto porty jsou po uzavření spojení vždy uvolněny. Při vytvoření spojení pro prokládané TCP je zaznamenána reference na TCP soket klienta a také číslo kanálu, jenž klient uvedl v hlavičce `Transport` požadavku.

Seznam všech transportních spojení je ukládán ve vektoru (použita třída `etl::vector` knihovny etl) přímo uvnitř dané stopy společně s informací, kterému sezení je toto spojení přiřazeno. Každá ze stop má tak k nim při zasílání dat přímý přístup a nemusí je nijak složitě vyhledávat.

Seznam všech sezení celého serveru se pak nachází ve vektoru fixní velikosti třídy `RtspServer`.

PLAY

Implementace tohoto příkazu změní stav stopy na stav `PLAYING`, tedy přehrávání. Přehrávání se týká pouze konkrétního transportního spojení, jenž je asociováno se sezením a danou stopou. V případě, že požadavek neobsahuje žádnou stopu, jenž se má začít přehrávat, je zahájeno přehrávání všech stop daného sezení. Existující sezení se hledají podle identifikátoru předaného pomocné funkci `findSession` a každá ze stop má také funkcí `FindConnectionBySessionId` pro vyhledání konkrétního transportního sezení.

V případě, že existuje minimálně jedno transportní spojení uvnitř stopy ve stavu přehrávání, je celá stopa taktéž ve stavu `PLAYING`. Tato změna stavu je oznámena delegátem `OnPlayingStateChangedCallback` a kód pak může na tuto událost reagovat – například inicializací kamery a začnutím zasílání video přenosu. V případě vypnutí přehrávání pak může dojít k deinicializaci kamery pro ušetření prostředků.

TEARDOWN

Vyhledá dané sezení a odstraní ho. V případě, že se v sezení vyskytují nějaká transportní spojení, je zapotřebí je odstranit také. V případě prokládaného TCP spojení je pouze instance odstraněna z vektoru stopy. Pokud se jedná o UDP spojení, je taktéž odstraněno, avšak navíc jsou uvolněny také jeho použité porty.

Odpověďi na požadavky

Pro konstrukci odpovědí byla implementována třída `RtspResponder` jejímž úkolem je sestavit odpověď na požadavky, jenž bude po dokončení zpracování odeslána klientovi. Zaslání jednoduché odpovědi s RTSP status kódem lze použít `Respond(RtspStatusCode)`. `ResponseStart(RtspStatusCode)` pak zahájí konstrukci odpovědi s daným kódem, neodesílá však nic klientovi – k tomu je zapotřebí použít `ResponseEnd()`. Mezi těmito dvěma voláními je pak možné přidávat hlavičky použitím `ResponseHeader(etl::string_view headerName, etl::string_view headerValue)`.

Pokud je zapotřebí odesílat jakýkoliv obsah v těle, je nutné využít `ResponseContent( etl::string_view content, etl::string_view contentType)`. Ta nastaví hlavičky jako `Content-Length` a `Content-Type`, ukončí odpověď a zašle uvedený obsah.

Přenos videa a audia

Přenos jednotlivých stop vždy závisí na jejím typu. Zaslání jednoho paketu protokolu RTP se skládá z hlavičky zmíněné v sekci 5.4. Každý RTP paket může mít maximální délku pouze 65535 bajtů, a tak je součástí každé ze stop předem alokováno pole o této fixní velikosti, které je při stavbě paketů používáno.

Protože je součástí každého paketu jeho hlavička, je k jejímu vyplnění dostupná utilita `RtpUtil::WriteRtpHeader` s argumenty jako typ stopy, sekvenční číslo paketu či časová značka. Další obsah paketů se pak již liší podle typu stopy.

V případě přenosu videa kodeku H.264 je o správné zaslání zodpovědná funkce `SendNAL`, kdy z jedním z jejích argumentů je právě pravdivostní hodnota `lastNALforFrame`. Tím dáváme klientovi přijímajícímu náš obraz najevo, že byla odeslána poslední NAL jednotka. Pokud se jedná o menší NAL jednotku, je vložena do paketu samostatně nakopírováním dat.

V případě, že se jedná o jednotku větší než limit RTP paketu, bylo potřeba implementovat fragmentaci typu A. NAL jednotka je rozdělena na menší části, a postupně jsou vytvářeny RTP pakety. V případě této fragmentace je však nutné před dané části vložit speciální hlavičku oznamující klientovi, že k fragmentaci vůbec dochází. Jedná se o upravenou hlavičku NAL jednotky s typem 28, za níž následuje fragmentační hlavička o velikosti jednoho bajtu, informující, v jaké části původní NAL jednotky se nacházíme – tedy start bit nastaven na 1, pokud se jedná o začátek, a end bit, pokud o konec.

Při přenosu audia kodeku G.711 je zasílající funkce pojmenována `SendSlice` a přijímá v argumentu vzorky audio záznamu, jenž jsou již zakódovány. Jako typ stopy v RTP hlavičce je uvedeno číslo 8 a za hlavičku jsou rovnou kopírovány vzorky.

Poté co stopa sestaví RTP pakety, zavolá funkci `SendPacket`, která rozešle data všem přehrávaným transportním spojením. Pokud je transportní spojení typu prokládané TCP, jsou data položena za hlavičku se znakem `$`, číslem kanálu, délkou paketu a následně odeslána na TCP soket klienta.

Speciálním typem jsou stopy pro příjem audia. Pokud nějaký z transportních spojů přijme data, je vyvolána funkce `HandleBackDataReceived` přímo na dané stopě, jíž se transportní spoj týká. Z dat se odstraní RTP hlavička a zbytek dat je předán obslužné rutině uživatele stopy pro další zpracování.

9.2 Čtení dat z kamery a jejich přenos

Tato podkapitola se zabývá inicializací a čtením dat z kamerového senzoru. Dále je také popsáno, jakým způsobem jsou tato data zasílána RTSP klientům.

Inicializace kamery

Před samotným čtením je nutné kameru a také periferii MIPI CSI správně inicializovat. Inicializace modulu provádí třída `CameraDevice`. Její funkce `Init()` vytváří instanci objektu `esp_cam_sensor_device_t` na základě konfigurační struktury, kde je nutné uvést použi-

tou I2C sběrnici a adresu senzoru. Na tomto objektu je prováděno nastavování parametrů senzoru, tedy vlastností jako aktuální formát (rozlišení a snímkovací frekvence) či zisk.

Funkce `CameraDevice::SetFormat(formatSelector)` slouží k nastavení formátu senzoru a její argument `formatSelector` je delegátní funkce. Vybrat aktuální formát totiž lze jen tak, že se nejdříve iteruje přes všechny podporované formáty dostupné v instanci `esp_cam_sensor_device_t`, vybere se formát, který požadujeme, a následně je ukazatel na něj předán funkci `esp_cam_sensor_set_format`, která jej nastaví na senzoru. Má funkce tedy pro každý z iterovaných formátů volá delegátní funkci, jenž v případě shody vrací pravdivostní hodnotu `true`.

Součástí formátu senzoru je však krom rozlišení a snímkovací frekvence také použitá frekvence komunikační sběrnice MIPI, či přímo jaká sběrnice se má používat. Některé senzory totiž krom MIPI CSI dokáží komunikovat také po sběrnici DVP, jenž má jiné hardwarové zapojení. Aby nemohlo dojít k nastavení špatného formátu, jsou při iteraci filtrovány jen ty formáty, co využívají právě MIPI CSI, komunikují po dvou datových linkách a jejichž výstupní formát pixelů je `RAW10`.

Po nastavení formátu může uživatel třídy využít funkcí `CameraDevice::StartStream()` kterým je senzoru sděleno, že může začít po sběrnici MIPI CSI zasílat data.

Příjem snímků

Příjem a počáteční zpracování dat z kamery provádí třída `CameraSource`. Je inicializována MIPI CSI sběrnice, konkrétně použitím konfigurační struktury `esp_cam_tlr_csi_config_t` do níž je nastaveno použité rozlišení kamery, vstupní a výstupní formát pixelů a také frekvence sběrnice. Vstupní formát je nastaven na `RAW10` a výstupní na `YUV420`, což je formát, se kterým pak pracuje také H.264 enkodér.

Krom sběrnice je také nastavena periferie ISP (*Image Signal Processor*), neboť data přijatá sběrnicí jsou vždy nejdříve zpracována. Jako zdroj dat je zapotřebí nastavit právě sběrnici MIPI CSI společně se stejnými parametry formátu jako u sběrnice. Uvnitř mikrokontroléru je zvolením zdroje vytvořena přímá cesta ze sběrnice do ISP a tyto data před zpracováním neprocházejí ani pamětí či jádrem procesoru. Konverzi jednotlivých formátů pixelů provádí právě ISP a je nutné, aby se vstupní a výstupní formáty napříč sběrnicí a ISP shodovaly. Jinak budou výstupní data poškozena či nebude čtení fungovat vůbec.

ISP funguje na principu tzv. *pipeline*. Po přijetí dat sběrnicí MIPI CSI jsou tato data postupně zpracovávána jednotlivými kroky, které jsou samostatně nastavitelné. Vytvoření ISP je provedeno funkcí `esp_isp_new_processor(config, handle)` a po vytvoření je třeba jej povolit `esp_isp_enable(handle)`. Právě tuto *pipeline* využívám také k aplikování matice korekce barev. Je tedy pomocí `esp_isp_ccm_configure(config, handle)` nastaven krok *pipeline* pro tuto korekci s maticí doporučenou výrobcem.

Poté co je vše nastaveno a kamera je přepnuta do režimu zasílání dat, mohou být snímky přijímány do paměti. Předem jsou alokovány dvě pole v paměti PSRAM o velikosti závislé na rozlišení. Dvě pole jsou použita, neboť příjem probíhá střídavě. Protože jsou výsledná data ve formátu `YUV420`, je velikost těchto polí pouhých 1,5 × (šířka × výška).

Kopírování každého snímku po dokončení zpracování v ISP je prováděno za pomocí DMA (*direct memory access*), která je automaticky nastavena po inicializaci MIPI CSI sběrnice. Po dokončení přenosu je vyvolána obslužná rutina `s_camera_get_finished_trans` kde je předána informace o poli, do nějž byl snímek uložen. Tato rutina následně za pomocí synchronizační struktury typu semafor oznámí voláním `xSemaphoreGiveFromISR` úspěšný

příjem. Voláním funkce `CameraSource::WaitForData()`, která čeká na uvolnění semaforu, pak může další kód čekat na přijetí snímků.

Kódování snímků

Pro kodování je využita vestavěná hardwarová periferie, jenž je konfigurována komponentou *espressif__esp__h264* a její použití je provedeno ve třídě `H264Encoder`. Konfigurace se skládá jak z rozlišení a snímkovací frekvence kamery, tak také požadovaného datového toku (*bitrate*).

Jako výstup enkodéru je použito pole o stejné velikosti jako jsou ta vstupní. Pole je fixní velikosti, neboť velikost po kódování není možné znát. Jisté je však minimálně to, že kódovaná data nemohou být větší než velikost vstupu.

Pro vstup enkodéru je vytvořena struktura `esp_h264_enc_in_frame_t` s ukazateli na vstupní pole a jeho velikost. Stejně tak je vytvořen `esp_h264_enc_out_frame_t` pro výstup. Obě tyto struktury jsou následně předány funkci `esp_h264_enc_process(encoder, in, out)` jenž daný vstup zpracuje na základě uloženého stavu enkodéru.

9.3 MQTT klient

Vzhledem k použití MQTT discovery bylo zapotřebí implementovat právě takového klienta, jenž bude provádět registraci jednotlivých Home Assistant entit. Tuto funkcionalitu zastřešuje třída `HaMqttClient` jejímž úkolem je připojení a komunikace s MQTT brokerem. K tomu využívá vestavěné komponenty pro MQTT klienta z frameworku ESP-IDF. Podporováno je připojení jak čistým TCP spojením (url začínající `mqtt://`) tak také šifrováním s použitím SSL (schéma `mqtts://`). V případě, že je komunikace prováděna šifrovaným spojením, je povinné specifikovat SSL certifikát MQTT brokeru. Je tak spojení chráněno proti útokům typu MITM (*man-in-the-middle*).

Klientu jsou tedy předány informace o MQTT brokeru, přihlašovací údaje a MQTT témata. Po připojení je potřeba přihlásit se k odběru *birth* tématu, a také témat, po kterých bude klient dostávat aktualizace o entitách. Ten první z nich je specifikován v konfiguraci zařízení, ty ostatní jsou odebírány jednotným odběrem s cestou `{deviceId}/#`, kde `deviceId` značí konfigurovatelné ID zařízení a znak `#` je zde zástupným znakem pro všechny podcesty.

Dále je funkcí `publishConfiguration()` sestavena konfigurační zpráva zahrnující informaci o zařízení a také o všech entitách. Ta se skládá z textového řetězce JSON a k jejímu sestavení používám právě knihovnu *ArduinoJSON*. Daný JSON objekt je konstruován v soukromé proměnné třídy, neboť má předem danou velikost, která by mohla při některých voláních vyvolat přetečení zásobníku, pokud by se jednalo o proměnnou lokální.

Pro programovou definici entit zařízení byla jako základ vytvořena třída `HaEntity`. Každá instance musí mít své unikátní ID a musí implementovat funkci `populateDiscoveryMessage(JsonObject)`. Do předaného JSON objektu, jenž je vyhrazen právě této entitě, zapíše informace specifické pro daný druh entity. V případě entity typu `switch` například obsah zprávy, která se zašle na dané téma po přepnutí do stavu zapnuto.

Každou z entit ve výsledném programu je zapotřebí nejdříve vytvořit, ideálně na haldě, a poté připnout k MQTT klientovi funkcí `AttachEntity(HaEntity&)`. Připnutí by mělo být provedeno před spuštěním klienta a jeho připojením k brokeru – jedině tak je zajištěna správná registrace entity v registrační zprávě. V opačném případě je entita zaregistrována až při dalším připojení k brokeru, například z důvodu výpadku síťového připojení. K uložení

připnutých entit klienta je využíváno vnořeného spojového seznamu (*intrusive linked list*) – odkaz na dalšího potomka na stejné úrovni je již součástí základní třídy `HaEntity`).

9.4 RFID čtečka

O načítání jednotlivých RFID karet a jejich identifikátorů se stará použitá komponenta *abobija/rc522*. Konfigurace komponenty vyžaduje pouze definici použitých SPI pinů, kterými je připojen čip MFRC522, společně s parametry úlohy, jenž komponenta interně využívá pro své fungování. Tyto parametry, jakými jsou např. velikost zásobníku či priorita, byly ponechány na výchozích hodnotách.

Kód využívající komponentu je součástí třídy `RfidReader`. Při startu čtečky je na inicializovaném objektu registrována obslužná rutina `OnPiccStateChanged` (stav čtené karty se změnil). Stavů, které mohou nastat, je několik, ovšem pro účely této práce stačí pouze stav `RC522_PICC_STATE_ACTIVE` při kterém došlo k přiložení čteného zařízení. Společně se stavem je rutině předána také informace o čtené kartě – hlavně její identifikátor, jenž je pro nás důležitý.

Protože se může identifikátor skládat z libovolného počtu bajtů, vytvořil jsem nad ním strukturu společně s pomocnými funkcemi pro konverze mezi různými formáty. Ve třídě `RfidTagId` lze tak nalézt `ToString()` jenž provede konverzi na hexadecimální řetězec. Pro opačný směr pak lze využít `FromString(inStr)`.

Pro konverze týkající se reprezentace v Base64 pak byly vytvořeny funkce `ToBase64` a `FromBase64(inStr)`. Interně využívají pro své fungování příslušné implementace z knihovny mBedTLS, avšak výsledky jsou ukládány do textových řetězců komponenty `etl::string`. Ukládání do nevolatilní paměti probíhá pak právě v této reprezentaci viz. sekce 9.5.

Stejně jako pro identifikátor, tak pro kartu jako takovou existuje struktura, kde je kromě identifikátoru uloženo také její jméno a hodnota práva otevírat. Pro snadné vyhledávání v databázi je pak implementován operátor pro porovnání s další instancí karty nebo identifikátoru samotného. Hodnoty identifikátorů jsou porovnávány tak, jak jsou uloženy v paměti struktury, tedy pole bajtů se specifikovanou délkou.

9.5 Ukládání konfiguračních hodnot

Ukládání konfigurace zařízení jsem se rozhodl provést za použití vestavěné komponenty *esp_nvs* pro NVS (non volatile storage). Její jednoduché API podporuje základní datové typy, jakými jsou znaménkové či bezznaménkové čísla (`uintX_t / intX_t`), pravdivostní hodnoty (`bool`), či textové řetězce ukončené nulovým znakem (`char[]` hh[1]. Pro každou ukládanou položku je pak stanoven unikátní klíč o maximální délce 15 znaků vyjma nulového znaku. Položky jsou ukládány do vyhrazeného oddílu v paměti flash.

Protože je konfigurovatelných hodnot v zařízení větší počet, a neustálé opakování kódu pro každou z hodnot by bylo mírně nepraktické, vytvořil jsem si systém vlastností (*properties*) a kontejnerů, ve kterých tyto vlastnosti leží. Každý z kontejnerů může těchto vlastností obsahovat libovolný počet a kontejnery lze zanořovat.

Základem tohoto systému je třída `SettingBase` z níž následně dědí jak vlastnosti (třída `Property<T>`) tak kontejnery (`SettingContainer`). Každá z instancí má svého rodiče a přiřazený klíč a poskytuje následující funkce:

- `Str15 GetFullKey()` – vypočte na základě klíče a rodiče plný řetězec klíče, který bude použit k uložení do NVS. Rodiče prochází rekurzivně, konkatenuje jejich klíče dokud nenarazí na konec (instance bez rodiče)
- `NvsPtr* GetHandle()` – navrací ukazatel na aktuálně otevřenou instanci NVS. Je virtuální, ovšem s výchozí implementací jenž zavolá `GetHandle()` rodiče. Kořenový prvek by tak měl tuto funkci přepsat
- `void Load()` – virtuální funkce pro načtení obsahu. Odvozené třídy jí musí implementovat.

Třída `Property<T>` má právě jeden parametr šablony, a tím je datový typ, který drží. Lze jí tak použít pro ukládání libovolného datového typu. Její implementace `Load()` nejdříve získá hodnotu celého klíče, ukazatel na instanci NVS a provede načtení. Dále definuje funkci `Set(T)`, jenž provede téměř stejný postup, akorát ve svém finálním kroku provede uložení. Pro podporu jiných než základních datových typů však obě využívají pro operace načtení a uložení prostředníka – `TypeHandler<T>` s funkcemi `LoadType` a `SaveType`. Jejich implementaci pro základní typy si kompilátor dokáže odvodit sám (generická varianta je součástí definice), pro ty ostatní stačí danou třídu implementovat.

Aby nedocházelo k zbytečným opakováním v kódu, byl přepsán operátor přiřazení, společně s operátorem `T()`. Přiřazení či přečtení uložené hodnoty je pak snadné, jak ukazuje výpis 9.1 na příkladu.

```
1 Property<int>& prop = ...//reference na existujici vlastnost
2 prop = 42; //operator =(T)
3 int storedValue = prop; //operator T()
```

Výpis 9.1: Ukázka přiřazení a přečtení do vlastnosti

Implementace funkce `Load()` kontejnerovou třídou `SettingContainer` se od vlastnosti liší tím, že provede načtení potomků zavoláním stejné funkce.

V proměnné `children` typu `etl::intrusive_list` z komponenty *etl* je pro každý kontejner ukládán seznam jeho potomků. Tento seznam je tedy v paměti ukládán jako vnořený spojový seznam. Vzhledem k absenci reflexe v jazyce C++, je zapotřebí tento seznam sestavovat manuálně, a to vždy předáním ukazatele na rodičovský kontejner v konstruktoru potomka. Ten sám sebe vloží do seznamu a nastaví si rodiče. Díky použití tzv. *uniform initialization* jazyka C++ je možné předat ukazatel na rodiče již v definici proměnné. Společně s ním je zapotřebí předat jméno klíče, případně nepovinnou výchozí hodnotu.

Poslední důležitou třídou je `NamespaceContainer`. Ta musí být povinně použita jako kořenový kontejner. Její implementace načítací funkce otevře instanci NVS. K otevření použije přiřazený klíč jmenného prostoru `namespace`. Ten umožňuje dělit celý oddíl paměti na menší logické celky – v každém z nich musí být klíče položek unikátní, ale stejný klíč se může vyskytovat ve více těchto prostorech.

Použití celého tohoto systému kontejnerů a tvorbu mírně složitějšího objektu ukazuje příklad ve výpisu 9.2.

```
class NetworkSettings : public NamespaceContainer
{
public:
    using NamespaceContainer::NamespaceContainer;
    class MdnsSettings : public SettingContainer
    {
    public:
        using SettingContainer::SettingContainer;
        Property<bool> Enabled{"enabled", true, this};
        Property<Str32> Hostname{"hostname","doorbell", this};
    } Mdns{"mdns", this};
    class EthernetSettings : public SettingContainer
    {
    public:
        using SettingContainer::SettingContainer;
        Property<bool> UseDHCP{"dhcp", true, this};
        Property<uint32_t> Ip{"ip",0, this};
    } Ethernet{"eth", this};
};
NetworkSettings network{"network"};
network.Load(); //nacte vsechny vlastnosti z NVS
bool isEnabled = network.Mdns.Enabled; //pouziti hodnoty
```

Výpis 9.2: Ukázka využití celého systému pro tvorbu složitějších objektů

Ukládání databáze RFID karet

Bohužel na ukládání seznamů prvků nelze rozumně aplikovat výše uvedený postup. Ukládání identifikátorů karet tedy probíhá jiným způsobem. Pro každou z uložených karet je za pomoci `RfidTagId::ToBase64()` vypočtena Base64 reprezentace a ta je použita jako klíč položky v NVS. Protože je délka identifikátorů standardem omezena na 10 bajtů, může po použití Base64 délka tohoto klíče dosáhnout maximálně 14 znaků, což je dostačující vzhledem k limitu délky klíče NVS (15 znaků).

Informace o RFID kartě (uživatelem nastavené jméno a právo otevřít) jsou pak do položky vloženy dohromady jako textový řetězec oddělený středníkem.

9.6 Obsluha zvonku a zámku

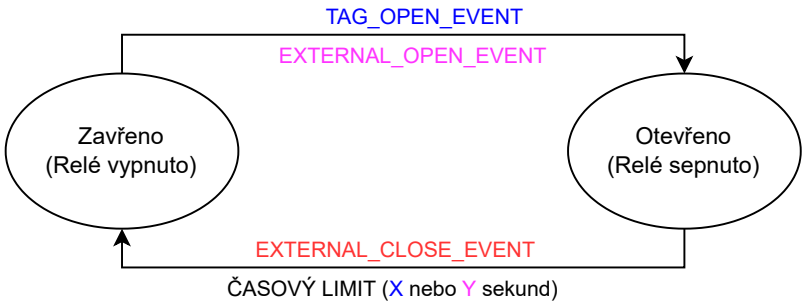
Veškerá logika týkající se správného ovládání spínání relé a také reakce na zmáčknutí zvonkového tlačítka je soustředěna v souborech `LockApp.*`. Definovány jsou tři třídy: `LockApp`, `Output`, a `UserButton`. `Output` vytváří lehkou abstrakci nad výstupním GPIO pinem, `UserButton` pak inicializuje kapacitní dotykový senzor zabudovaný v mikrokontroléru.

Jádrem obsluhy je však třída `LockApp`, jejíž jedinými argumenty jsou reference na konfiguraci zařízení a instanci MQTT klienta připojeného k Home Assistantovi. V konstruktoru jsou vytvořeny a přidruženy ke klientovi tři používané entity: *zámek*, *skener karet* a *zvonkové*

tlačítko. Pro účely komunikace mezi úlohami jsou také vytvořeny dvě instance FreeRTOS synchronizačního prostředku nazvaného skupina událostí s příznaky (*event groupy*).

Tak jak bylo uvedeno v návrhu, stav relé řídí asynchronní automat řízený událostmi. Ten je zde implementován vláknem se smyčkou `while`. Vlákno začíná ve stavu `Zavřeno`, tedy vypne relé, a za pomocí dříve vytvořených skupin událostí čeká na nastavení příznaků `TAG_OPEN_EVENT_BIT` nebo `EXTERNAL_OPEN_EVENT_BIT`. Jako časový limit tohoto čekání je nastaveno efektivní nekonečno, tedy kromě nastavení příznaků nemůže být čekání nijak přeskočeno. Po úspěšném čekání je automat ve stavu `Otevřeno` a na výstupní GPIO pin relé vyslán signál pro sepnutí.

Použitý příznak je zaznamenán a na jeho základě stanovena doba, po kterou automat v tomto stavu setrvá. Tyto doby jsou nakopírovány z konfigurace zařízení. Aby šlo během otevření provést také okamžité zavření, je místo čekání použitím časovačů zahájeno čekání na příznak `EXTERNAL_CLOSE_EVENT_BIT`, avšak nyní je časový limit nastaven na maximální dobu setrvání. Takto je čekání dokončeno buď vypršením limitu, či nastavením příznaku. Daný automat zobrazuje následující obrázek 9.1. Závislost časového limitu na způsobu otevření je vyznačena barevně.



Obrázek 9.1: Asynchronní konečný automat stavu relé

Součástí `LockApp` je také instance `RfidReader` jenž nemá žádné závislosti a krom obsluhy zámku není nikde jinde využívána. Pro událost načtení RFID karty je vytvořena příslušná obslužná rutina. Ta se podle aktuálního režimu verifikace uloženého v nastavení rozhoduje, zda provede ověření v databázi identifikátorů či zaslání MQTT entitě *skener karet*. Je-li identifikátor oprávněn k otevření, je výše uvedenému automatu nastaven příznak `TAG_OPEN_EVENT_BIT`.

Další událostí, konkrétně změnou stavu entity *zámek* jsou pro změnu nastavovány příznaky `EXTERNAL_OPEN_EVENT_BIT` při stavu `ON` a `EXTERNAL_CLOSE_EVENT_BIT` při stavu `OFF`.

9.7 Webová REST API

Tato API je používána pro konfiguraci zařízení z webového rozhraní. Pro implementaci jsem se rozhodl použít vestavěné komponenty HTTPS serveru, ten dokáže fungovat také v režimu HTTP bez šifrování, což je v případě prvotního zapnutí zařízení výchozí stav. K povolení SSL je zapotřebí nahrát jak privátní klíč, tak certifikát serveru.

HTTP server je nastaven tak, aby byl schopen přijímat až 10 spojení v jeden moment. Pro plynulé fungování HTTPS bylo také povolení *Enable server session tickets* v konfiguraci frameworku ESP-IDF. Takto je umožněna recyklace SSL sezení a každý z nových

HTTP požadavků klienta tak nemusí při připojení tato sezení zahajovat znova. Většinou se totiž jedná o vyšší stovky milisekund, což by značně zpomalovalo práci s administračním rozhraním.

Registrace koncových bodů

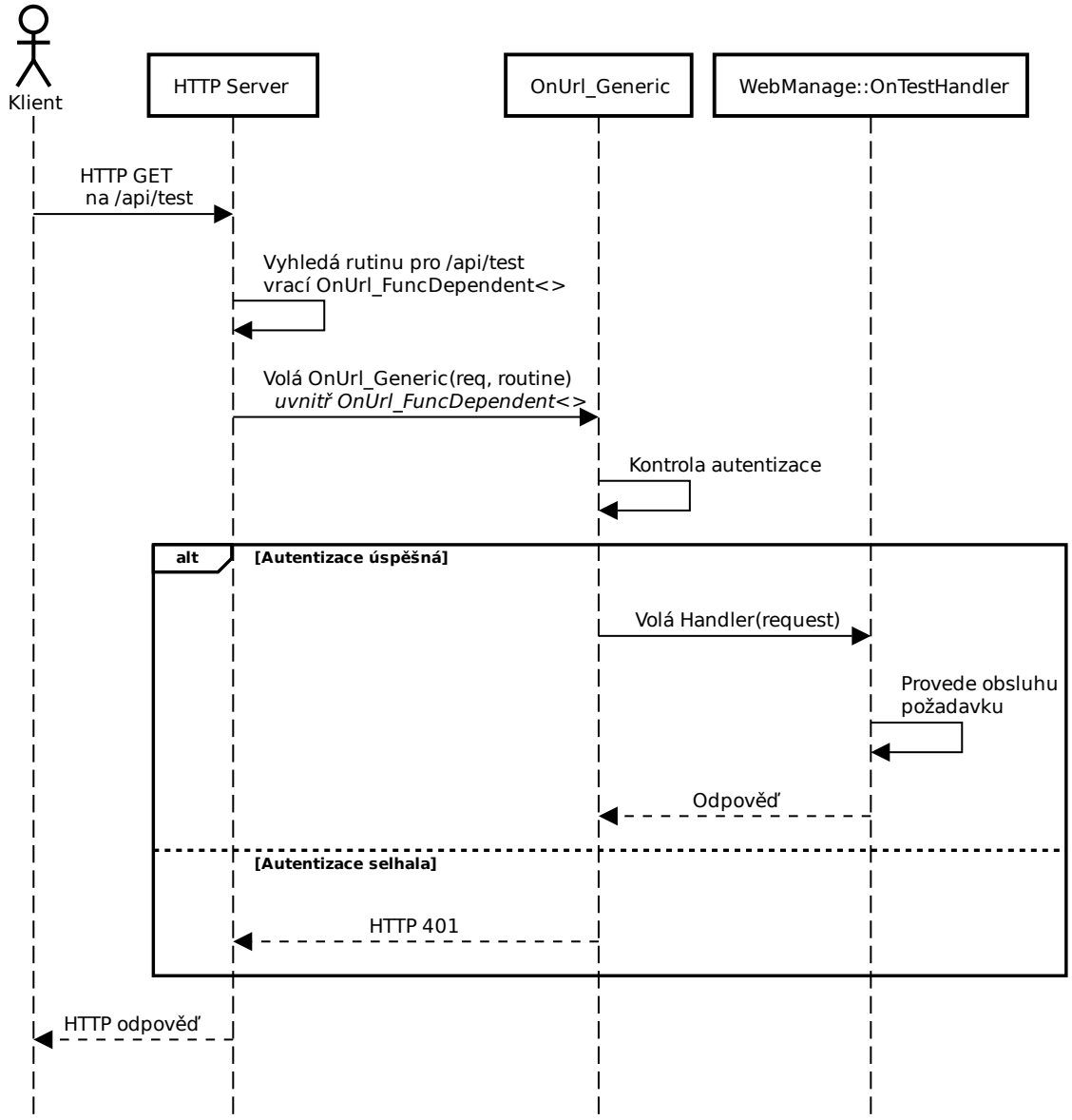
Pro každou z cest koncových bodů API je při inicializaci serveru registrována funkce, tzv. *URI handler*. Pro danou cestu je uvedena povolená HTTP metoda společně s obslužnou rutinou, jenž se postará o zpracování požadavku. K umožnění registrace těchto obslužných rutin byly definovány funkce `RegisterXXXX(const char* path, bool requireAuth)`, kde `XXXX` značí použitou HTTP metodu. Jedná se o šablonové funkce s jedním argumentem `auto function`. Obslužnou rutinu lze tedy specifikovat jako tento argument.

Dále je definována pomocná funkce `OnUrl_FuncDependent(httpd_req_t* req)`, taktéž se stejnou šablonou. Příkladem procesu registrace obslužné rutiny může být následující volání:

```
1 RegisterGET<&WebManage::OnTestHandler>("api/test", true);
```

Kompilátor vytvoří implementaci `OnUrl_FuncDependent<&WebManage::OnTestHandler>`. Ukazatel na tuto funkci je pak uložen do konfiguračních struktur HTTP serveru společně s cestou `api/test` a metodou GET. Provedením HTTP požadavku na tuto cestu je implementace spuštěna, ukazatel původní obslužné rutiny, tedy `OnTestHandler`, je přetypován na typ `void*`, a zavolána generická funkce `OnUrl_Generic(httpd_req_t* request, void* routine)`.

Hlavním úkolem této generické funkce je kontrola autentizace požadavku. Pokud je vše v pořádku, je obsluha předána již skutečné obslužné rutině registrované v původním příkladu. V opačném případě je požadavek zamítnut s HTTP kódem `401 Unauthorized`. Následující obrázek 9.2 zobrazuje průběh těchto volání.



Obrázek 9.2: Sekvenční diagram průběhů volání funkcí při HTTP požadavku

Autentizace

Všechny kromě jednoho z koncových bodů API vyžadují autentizaci. Pro tento účel je využívána HTTP hlavička `Authorization` jejímž obsahem musí být tzv. *Bearer token*. V mém případě jsem zvolil přístupový token o délce 32 znaků, jenž je po přihlášení uložen v paměti serveru.

Token je po úspěšném přihlášení generován použitím `esp_random()`, jenž by dle dokumentace měla být kryptograficky bezpečná, neboť využívá v případě mnou zvoleného mi-

krokontroléru hardwarový generátor pravých náhodných čísel. Přihlášení probíhá zasláním přístupového hesla na koncový bod API s cestou `api/login` s JSON řetězcem obsahujícím položku `password`. Přihlašovací jméno není zapotřebí.

V jeden moment však může být platný pouze jeden přístupový token. Dojde-li k přihlášení k API vícekrát po sobě, předchozí přístupové tokeny budou automaticky zneplatněny.

Koncové body

Data jsou API předávána ve formě JSON, a to jak obsah požadavků, tak také odpovědi na něj. Implementace obslužných rutin koncových bodů se tedy skládá z vytvoření JSON objektu a naplněním dat. V případě požadavků s metodami POST pak načtení tohoto objektu z dat HTTP požadavku. V obou případech je využíváno knihovny *ArduinoJSON*.

Pro načítání bylo vytvořeno C makro `LOAD_JSON(doc,size)`, jenž definuje lokální proměnnou typu `StaticJsonDocument<size>` a pokusí se o načtení těla HTTP požadavku do tohoto JSON objektu. V případě chyby automaticky ukončí obslužnou rutinu s příslušným chybovým HTTP kódem (například `HTTP 400 Bad Request` v případě neplatného JSON řetězce). Pokud měl požadavek obsah ve formě JSON a jeho odpověď má být taktéž ve stejném formátu, daný JSON objekt bude nejdříve vyčištěn a následně zrecyklován pro tvorbu odpovědi.

Celá API pracuje s instancí `Settings` na které přímo nastavuje jednotlivé konfigurační hodnoty. Obslužné rutiny požadavků mají k dispozici reference instancí MQTT klienta a síťového rozhraní, kterými můžou provést restart služby po nastavení nových hodnot.

Přehled všech koncových bodů, jenž jsou v API implementovány, lze nalézt v tabulce B.1 uvedené v příloze B.

9.8 Webové administrační rozhraní

Rozhraní provedené ve formě *single-page* webové aplikace bylo vytvořeno s použitím frameworku SvelteKit. Ten byl vybrán obzvláště kvůli tomu, že na rozdíl od ostatních webových frameworků nepoužívá virtuální DOM a je tedy rychlejší. Důležitou vlastností je také malá velikost výstupních souborů po kompilaci projektu. Díky tomu je snadné uložit celé rozhraní do flash paměti zařízení.

Krom samotného frameworku byla použita sada kaskádových stylů TailwindCSS společně s knihovnou Flowbite-Svelte pro snadnější práci s webovými prvky. Kód pro obsluhu GUI byl napsán v jazyce TypeScript s použitím knihovny Redaxios pro volání API zařízení.

Souborová struktura projektu s webovým rozhraním se skládá ze dvou hlavních složek. První z nich, `src/lib` obsahuje logiku volání API oddělenou od grafického rozhraní a také několik GUI komponent sdílených mezi stránkami. Složka `src/routes` pak obsahuje jednotlivé pohledy aplikace. Navigace ve frameworku SvelteKit pracuje s cestami pohledů tak, jak se nacházejí v adresářové struktuře, tedy URL `/administration` přímo mapuje na rozhraní ve složce `administration`.

Volání API

Pro zprostředkování volání REST API metod byl vytvořen modul `ApiClient` jenž exportuje jednotlivé HTTP metody. První z nich se nazývá `PostNoAuth` a provede HTTP požadavek typu `POST` bez použití autentizace. Další z nich již autentizaci vyžadují – je nutné, aby byl nastaven správný přístupový token použitím `SetAccessToken`. Při spuštění programu je tento token nastaven na hodnotu `null` a každý požadavek, jenž autentizaci vyžaduje, automaticky selže.

Dostupná je obslužná rutina `unauthorizedCallback` jenž může uživatel nastavit. V případě, že při vykonávání jakékoliv z metod vyžadujících autentizaci vrátí API chybový kód `401 Unauthorized`, dojde k vykonání této obslužné rutiny.

API klienta následně využívají ostatní moduly, jejichž úkolem je zavolat konkrétní koncový bod API podle URL, společně se správně připravenými daty ve formátu JSON. Volání koncového bodu API s danou URL zobrazuje následující výpis kódu 9.4 modulu `SettingsNetwork.ts`.

```
1  import { Get, Post } from "./ApiClient";
2  ...
3  interface InterfaceSettings {
4    useDhcp: boolean;
5    ipAddress: string;
6    subnetMask: string;
7    gateway: string;
8  }
9
10 export async function GetEthernetSettings(): Promise<InterfaceSettings> {
11   const response = await Get("/api/ethernet");
12   return response as InterfaceSettings;
13 }
14
15 export async function SetEthernetSettings(useDhcp: boolean, ipAddress: string,
       subnetMask: string, gateway: string): Promise<void> {
16   const response = await Post("/api/ethernet", {useDhcp, ipAddress, subnetMask
       , gateway});
17   return response as void;
18 }
```

Výpis 9.3: Rozhraní a volání koncových bodů nastavení sítě

Exportované funkce jsou asynchronní a využívají *Promises* pro vrácení výsledku. Klient také zaručí, že obsah odpovědí API je automaticky převeden na JavaScript objekt z navráceného JSON řetězce.

Přihlašování a stav autentizace

Aktuální stav přihlášení drží modul `AuthService` jehož úkolem je obstarat správnou změnu stavu v případě zadání hesla či vykonání odhlášení. Aplikace se může nacházet v jednom ze tří následujících stavů:

- **LoggedOut** – aplikace kompletně odhlášena
- **LoggedIn** – uživatel je přihlášen a přístupový token by měl být validní
- **Unsure** – došlo k restartu aplikace a není jasné zda daný přístupový token je stále validní

Aktuální stav je pro využití zvenčí exportován jako nezapisovatelná reaktivní proměnná a při každé změně je tedy v rozhraní možné na změnu reagovat.

Modul si po přihlášení ukládá poslední přístupový token do úložiště *Local storage* v prohlížeči. Po restartu/načtení stránky s aplikací však nemusí být jasné, zda je daný token stále validní a je potřeba jej zkontrolovat. Aplikace tak v takovém případě začíná ve stavu `Unsure` dokud nedojde k ověření. Změny stavu přihlášení jsou signalizovány uživatelskému rozhraní a to na něj adekvátně reaguje. V případě neznámého stavu zobrazením načítacího kolečka.

V modulu je přímo použita dříve zmíněná obslužná rutina `unauthorizedCallback` API klienta. Pokud dojde k jejímu vyvolání klientem, aplikace je automaticky kompletně odhlášena, a to včetně odstranění přístupového tokenu z úložiště prohlížeče.

Aplikace využívá reaktivních proměnných stavu navigace pro zabránění přístupu uživatele k rozhraní v případě, že není přihlášen. Konkrétně se jedná o proměnnou `page.route.id` jenž vždy obsahuje aktuální navigovanou URL. Pokud je uživatel odhlášen a dojde k její změně na jakoukoliv jinou adresu než je přihlašovací formulář (`/login`), je uživatel na něj ihned přesměrován. Původní URL je však uložena a je na ní po úspěšném přihlášení přesměrováno.

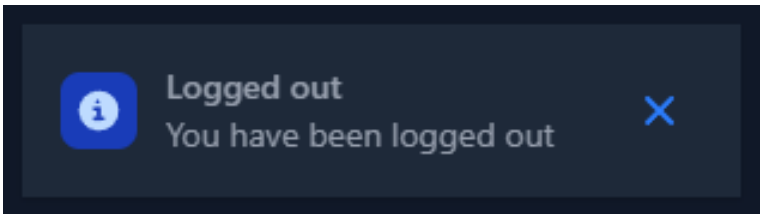
Zobrazení jednotlivých URL a reakce na změny stavu přihlášení jsou plně v kompetenci Svelte modulu `routes/+layout.svelte`. Zde jsou obsaženy také základní HTML prvky celé stránky, jako například navigační řádek s odkazy či tlačítko odhlášení.

Oznámení uživateli

Pro zlepšení uživatelské přívětivosti byl vytvořen systém oznámení, jenž se po vyvolání zobrazí na určitou dobu (ve výchozím stavu 5 sekund) uživateli v pravém dolním rohu obrazovky. Seznam aktuálních oznámení je spravován modulem `ToastService` a každá z notifikací má vždy titulek, zprávu a také typ (úspěch, chyba, varování, informace) na jehož základě je určena barva pozadí.

Seznam oznámení je uložen jako reaktivní proměnná typu `writable<Toast[]>` do níž jsou nové prvky vkládány exportovanou funkcí `DisplayToast(title, message, type, timeout?)` jenž prvek vloží na první pozici v poli. Součástí prvku je vždy také proměnná `shown` jenž může uživatelské rozhraní samotné nastavit a prvek tak skrýt. Po vytvoření oznámení je také spuštěn časovač po dobu `timeout` po jehož uplynutí je prvek skryt, aby mohlo dojít k animaci zavření. Po další sekundě je prvek kompletně odstraněn z pole. Oba časovače využívají vestavěné utility `setTimeout`.

Příklad jednoho ze zobrazených oznámení v aplikaci lze vidět na obrázku 9.3.



Obrázek 9.3: Informační oznámení o odhlášení z administrace

Konfigurační stránky

Webové rozhraní se skládá z celkově šesti podstránek, každá se svou URL a několika sekcemi konfigurace, které nastavuje.

Společnou částí všech stránek je komponenta rozhraní nazvaná `SetCard` jenž poskytuje pro jednotlivé sekce sjednocený vzhled, nadpis a také ukládací tlačítko, jenž může být deaktivováno nastavením příslušné vlastnosti komponenty. Deaktivace tlačítka je závislá na třech proměnných, a to `!isSaving`, `isLoading` a `isValid`. Pokud alespoň jedna z proměnných nabývá pravdivostní hodnoty `true`, je tlačítko deaktivováno. Všechny tyto proměnné jsou taktéž nastavovány v místě, kde je komponenta využita, samotná nad nimi nemá žádnou

kontrolu. Tlačítko ukládání vyvolává událost `save` a pokud je zrovna nastaveno `isSaving`, je uvnitř tlačítka zobrazeno načítací kolečko.

V horní části rozhraní se vyskytuje navigační řádek, jenž využívá komponent `NavBar` z knihovny Flowbite. Celé rozhraní se automaticky přizpůsobuje velikosti zařízení a také vyžádanému barevnému tématu prohlížeče. Přepínání na základě tématu je umožněno díky TailwindCSS třídám s předponou `dark:` jenž se při tmavém režimu automaticky aplikují na prvky.

Status stránka

Tato úvodní stránka zobrazuje aktuální stav otevření dveří/relé společně s tlačítkem na otevření/zavření. Pod tlačítkem je zobrazena informace o aktuální IP adrese síťového rozhraní a stavu připojení MQTT klienta.

Síťová konfigurace

Obsahuje dvě sekce – nastavení mDNS a nastavení rozhraní.

Pro nastavování IP adres byla vytvořena komponenta `IpInput` jejíž součástí je prvek `Input` s nastavenou minimální a maximální délkou vstupu. Vlastnost komponenty `isValid` je automaticky závislá na aktuálním zadaném vstupu. Kontrola je prováděna regulárním výrazem pro IP adresu, a v případě, že je vstup zadán špatně, je pod ním zobrazena chyba.

V sekci nastavení rozhraní je pole adresy pro výchozí bránu nastaveno tak, aby umožňovalo zadání prázdného vstupu.

HTTPS konfigurace

Pro povolení HTTPS serveru administrace a API je nutné nahrát větší soubory (certifikát a klíč). Vytvořena byla tedy komponenta `LargeFileInput` skládající se z velkého textového pole a také dialogu pro výběr souboru. Obsah těchto větších souborů je možné zadat buď ručně do textového pole, nebo vybráním souboru z disku, po němž je jeho obsah vypsán právě v textovém poli.

Výběr souborů je vytvořen prvkem `FileInput` jehož vlastnost `files` obsahuje seznam vybraných souborů. Mezi touto vlastností a lokální proměnnou `selectedFiles` je vytvořen tzv. *binding* (vazba). Po vybrání však dostaneme pouze odkaz na soubor, který je nutné přečíst třídou `FileReader`. Seznam povolených přípon a MIME typů vstupu je nastaven na `text/plain,application/x-pem-file,.pem,`

Seznam RFID karet

Zobrazení uložených RFID karet a jejich identifikátorů je provedeno ve formě tabulky. Zobrazeny jsou dva sloupce – identifikátor karty a její jméno nastavené uživatelem. Třetí sloupec obsahuje tlačítko odstranění karty ze zařízení. Protože je možné nechat kartu uloženou v zařízení, ale zároveň jí nepovolit otevírání, je pro každou kartu zobrazen barevný pruh v levé části řádku. Vybarvení zeleně znamená, že lze kartou otevírat, barva červená pak odmítnutí.

Pro vytváření a úpravu karet slouží komponenta `TagModifyDialog` jenž je vytvořena ve formě vyskakovacích dialogů.Pro zobrazení jako dialog je využit prvek `Modal` z knihovny Flowbite.

Zda se jedná o dialog vytvoření nebo úpravy, určuje vlastnost `isNewTagDialog`. Na jejím základě je určen zobrazený text tlačítka a titulku dialogu, případně také, zda je vstup pro identifikátor karty modifikovatelný.

Protože je vstup zadáván ve formě hexadecimálního čísla, je na něm taktéž prováděna validace za pomoci regulárního výrazu `/^[0-9A-F]{1,14}$/i`

Kliknutí tlačítka odstranění vyvolá vždy také vyskakovací dialog s informací o dané kartě a potvrzením akce. Vždy, když je provedena nějaká z akcí, jako odstranění, přidání či modifikace existující karty, je seznam karet načten kompletně znova ze zařízení pomocí API.

Uložení webového rozhraní do zařízení

Protože musí být webové rozhraní hostováno přímo ze zařízení samotného a ne webová stránka volně na internetu, je zapotřebí výstup kompilace SvelteKit projektu zakomponovat do firmwaru zařízení.

Prvním důležitým krokem je už jen správné nastavení kompilace celého webového projektu. SvelteKit běžně předpokládá, že projekt bude částečně vykreslován na serveru (technologie Server Side Rendering). Pro mé účely je však nutné toto serverové vykreslování vypnout. Soubor `svelte.config.js` tak bylo zapotřebí upravit a přidat do něj následující kus kódu:

```
import adapter from '@sveltejs/adapter-static';
...
const config = {
    ...
    kit: {
        adapter: adapter({
            pages: 'build',
            assets: 'build',
            fallback: "index.html", //vychozi stranka
            precompress: true, //automaticky provest kompresi vystupu
            strict: true
        }),
    },
    ...
};
```

Výpis 9.4: Nastavení statického adaptéru pro rendering

Po této úpravě jsou po kompilaci projektu připraveny všechny soubory, jenž stačí umístit na jakýkoliv HTTP server. Díky povolené kompresi jsou vytvořeny také komprimované verze souborů ve formátu GZip.

Pro zabudování rozhraní do firmwaru byl vytvořen skript `generateOutput.py` jenž provádí následující kroky:

1. Vytvoření komponenty `webgui` v ESP-IDF projektu
2. Vygeneruje hlavičkový soubor `WebGui.hpp` s definicí struktury `StaticFileEntry` a seznamem MIME typů souborů
3. Vytvoří zdrojový soubor `WebGui.cpp` s implementací funkce `WebGuiHandler::HandleRequest` a také inicializační funkcí `InitializeMap()`

4. Pro každý soubor výstupu kompilace webového rozhraní vygeneruje hlavičkový a zdrojový C soubor s jeho obsahem a. Zdrojový oubor se skládá z pole `const unsigned char file_name[]` a také proměnné `const size_t size_file_name` s uvedenou velikostí daného pole

5. Vygeneruje soubor `CMakeLists.txt` s informacemi o všech hlavičkových souborech a zdrojových C souborech v komponentě

Struktura `StaticFileEntry` obsahuje ukazatel na pole s daty, velikost těchto dat, informaci o tom, zda soubor používá kompresi (GZip) a také ukazatel na textový řetězec v poli MIME typů. Informaci o MIME typu souboru přiřazuje již generační skript na základě přípony souboru.

Seznam všech souborů je za chodu uložen v kontejneru typu `etl::unordered_map` a je vytvářen právě zavoláním funkce `InitializeMap()`. Jako klíč kontejneru byl zvolen textový řetězec názvu souboru.

Webový server pro API je upraven tak, aby všechny HTTP požadavky, jejichž URL nespadá pod žádný z koncových bodů API, automaticky vyřizoval funkcí `HandleRequest`. Ta provede vyhledání v kontejneru se soubory podle URL požadavku a v případě nalezení shody je obsah pole zaslán klientovi. Hlavička `Content-Type` nesoucí informaci o typu souboru je nastavena dle MIME typu uvedeného v záznamu.

Pokud je soubor již dopředu komprimován, je nastavena hlavička `Content-Encoding: gzip`. Zda-li je do firmwaru zabudována komprimovaná či nekomprimovaná verze souboru, rozhoduje generační skript podle toho, zda kompilace webového rozhraní vytvořila již předkomprimovanou verzi.

V aktuální verzi uživatelského rozhraní jsou předem komprimovány všechny soubory až na jeden. Díky tomu je v paměti FLASH zařízení rozhraním využito pouhých 135 KiB, což je v případě celkové velikosti paměti 16 MiB velmi malé procento. Tento způsob zasílání komprimovaných dat musí webový prohlížeč podporovat. Předpokládá se však, že každý prohlížeč s podporou JavaScriptu (jenž je potřeba pro chod rozhraní) je dostatečně moderní, aby akceptoval komprimované soubory.

Kapitola 10

Testování a vyhodnocení

Tato kapitola popisuje vyrobené zařízení a také jeho testování spolu s ověřením celkové funkčnosti se systémem Home Assistant. Taktéž jsou popsány problémy, na které jsem během vývoje narazil.

10.1 Problémové části tvorby zařízení

Jedním z největších úskalí celého vývoje byla slabá či nedostupná dokumentace samotného mikrokontroléru ESP32-P4. Jedná se totiž o velmi nový mikrokontrolér, jenž není oficiálně zaveden do sériové výroby a dostupné jsou pouze tzv. *engineering samples*. Postupné zveřejňování dokumentace tak probíhalo v době, co jsem na práci pracoval.

Protože je mikrokontrolér zatím dostupný pouze ve formě vývojových kitů, při výsledné desce byl čip z jednoho z kitů odpájen, aby bylo možné jej osadit na výslednou desku zařízení.

Dalším problémem byla obsluha kamery, konkrétně čtení dat sběrnicí MIPI-CSI, aby bylo možné je dále zpracovávat. Výchozí nastavení prostředí ESP-IDF a systému FreeRTOS však neumožňovalo dosahovat požadovaných snímkovacích frekvencí kamery. Později bylo zjištěno, že je potřeba zvýšit tzv. *tick rate* FreeRTOS na 1000 Hz, aby docházelo k častějšímu přepínání kontextu a tudíž bylo umožněno včasné předání dat mezi úlohami.

Posledním problémem bylo nastavení čtyř vnitřních regulátorů napětí zmíněných již v sekci 8.1. Jejich správné nastavení nebylo v dokumentaci specifikováno, a protože jsou GPIO piny použité pro komunikaci s RFID čtečkou napájeny právě jedním z těchto regulátorů, byla komunikace značně nestabilní. Po správném nastavení výstupního napětí regulátorů byly tyto problémy odstraněny.

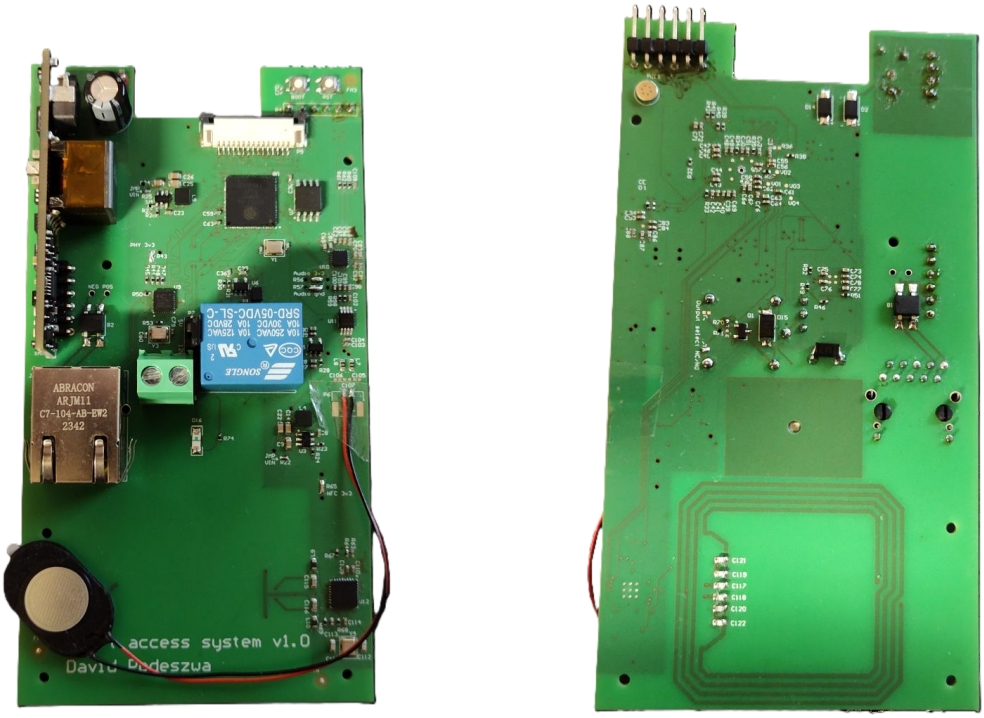
10.2 Kompletace desky

Výsledná deska byla vyrobena pouze jedna, a její kompletace byla poměrně zdlouhavá činnost. Nejdříve byly osazeny menší čipy v pouzdrech QFN a následně ručně všechny menší součástky. Před uložením byla aplikována pájecí pasta, na níž byly součástky položeny. Po zapájení byly všechny důležité spoje změřeny multimetrem, obzvláště byly zkontrolovány potenciální zkraty mezi napětovými větvemi a zemí.

Následně bylo připojeno testovací napájení 12 V na vstupy regulátorů, přičemž došlo k jejich poškození. Později byl identifikován problém se zapojením jejich vstupu `EN` (*enable*). Absolutní maximální napětí akceptovatelné na jejich vstupu je totiž pouhých 6 V, čehož

jsem si při návrhu schématu nevšiml. Zapojení bylo tedy opraveno jak ve schématu, tak na již vyrobené desce přepájením nových kusů a zvednutím vývodu z pouzdra tak, aby se nedotýkal plochy na desce.

Po ověření, že je toto zapojení již funkční a výstupy regulátorů odpovídají požadovaným napětím, byly připájeny všechny ostatní součástky jako hlavní mikrokontrolér, PoE modul či 15pinový konektor pro připojení kamery. Dále byl otestován PoE modul, přičemž bylo změřeno jeho výstupní napětí, které dosahovalo 12,068 V. Desku tak, jak vypadá po dokončení osazování, lze vidět na obrázku D.3. Zobrazeny jsou jak přední, tak zadní strana desky.



(a) Deska plošných spojů z přední strany (b) Deska plošných spojů ze zadní strany

Obrázek 10.1: Výsledné desky plošných spojů s osazenýma součástkama

Finální deska byla připojena programovacím modulem k počítači a byl otevřen monitor sériového portu, kde se již objevila hláška s úspěšným startem mikrokontroléru a informací, že se čeká na nahrání nového firmwaru do paměti flash.

10.3 Testování obslužného firmwaru

Po nahrání kompletního firmwaru bylo zařízení připojeno UTP kabelem do PoE injektoru, jenž byl připojen do lokální sítě. Zařízení je ve výchozí konfiguraci nastaveno tak, aby získalo svou IP adresu síťového rozhraní použitím DHCP.

Po úspěšném získání adresy byla otevřena URL `http://doorbell.local` v prohlížeči, čímž byla otevřena webová administrace. Tato URL je odvozena od výchozího jména služby mDNS zařízení. Po zadání výchozího hesla `admin` došlo k přihlášení.

Následovalo nastavení MQTT klienta, kde stačilo zadat URL MQTT brokeru společně s přihlašovacími údaji. Ostatní nastavení lze ponechat ve výchozím stavu, nebyla-li nijak

modifikována nastavení Home Assistenta. Po uložení se zařízení úspěšně připojilo k MQTT brokeru, zaslalo registrační zprávy a požadované entity se objevily v administraci Home Assistenta. Změnou stavu entity **Zámek** bylo sepnuto relé.

Dále byl v administraci změněn režim verifikace RFID karet na režim **Oba** a do databáze karet přidán identifikátor jedné z karet s povolením otevírat. Po přiložení došlo ke správnému ověření karty a opět bylo sepnuto relé na dobu specifikovanou v nastavení.

Funkcionalita kamery a audia byla ověřena již komplexnějším způsobem. Home Assistant sám o sobě pracovat s kamerami sice umí, ovšem efektivnější jsou externí integrace, které RTSP přenos konvertují na WebRTC. Pro otestování jsem tedy použil jednu z těchto integrací nazvanou *WebRTC Camera*[1]. Pro prostředí Home Assistenta poskytuje zobrazitelný prvek do ovládacího panelu Lovelace, kterým lze zobrazit RTSP přenosy. Tyto přenosy integrace interně převádí na WebRTC, zobrazitelné v prohlížeči.

Do ovládacího panelu jsem tedy přidal kartu dle YAML definice zobrazené ve výpisu 10.1.

```
1 type: custom:webrtc-camera
2 url: rtsp://doorbell.local:8080
3 mode: webrtc
4 media: video, audio, microphone
```

Výpis 10.1: Karta se zobrazením přenosu z kamery

Vždy, když je tato karta zobrazena v ovládacím panelu Home Assistenta, dojde na pozadí k otevření RTSP spojení směrem k zařízení.

Při testování bylo zjištěno, že přenos audio záznamů je momentálně nespolehlivý a velmi často se stává, že pokud je zapnut přenos audio a video stop najednou, dochází ke zpomalení přehrávání obou těchto stop. Pokud jsou stopy spuštěny samotné, tento problém nenastává. Aktuálně se mi nepodařilo tento problém vyřešit. Pravděpodobně se však jedná o špatnou synchronizaci RTP časových značek.

10.4 Cena zařízení

Tato sekce se zabývá odhadem celkové ceny zařízení na základě zakoupených součástek použitých k výrobě. Součástky byly primárně kupovány od distributorů LCSC.com a mouser.com s cenami uváděnými v eurech.

Velkou část ceny tvoří samotná deska plošných spojů, jenž v případě výroby v České republice tvoří zhruba 2000 Kč. Pokud by byla objednána od výrobců ze zahraničí, je možné cenu desky redukovat až na 500 Kč. Obecně by šla tato cena redukovat ještě více výrobou ve více kusech.

Odhadovanou cenu těch nejdražších a nejdůležitějších položek lze spatřit v tabulce 10.1. Uvedené ceny jsou za nákup v nejmenším možném množství, tak jak byly nakoupeny pro tvorbu finálního zařízení. Původní cena

[1] https://github.com/AlexxIT/WebRTC

Komponenta	**Cena**
HD G OV5647 5Mpx MIPI CSI kamera	22,50 €
AG5412 PoE modul	16,00 €
ESP32-P4 mikrokontrolér	11,00 €
MFRC522 RFID transceiver	3,14 €
GD25Q128ESIG paměť NOR flash	0,78 €
IP101GR ethernet PHY	0,93 €
ES8311 audio kodek	0,52 €
AP62250 spínaný regulátor	0,42 €
NS4150B zesilovač	0,12 €
TLV62569 spínaný regulátor	0,06 €
ME6211C33 LDO regulátor	0,05 €
Kondenzátory	0,97 €
Induktory	0,18 €
Rezistory	0,10 €
Ostatní součástky	5 €

Tabulka 10.1: Seznam některých z použitých komponent a jejich ceny

Protože není mikrokontrolér ESP32-P4 zatím volně dostupný, jeho cenu nelze jednoznačně určit. Ceny u distributorů se však pohybují okolo 11 €.

Výsledná cena součástek je tedy 65,17€ (1625 Kč), s přičtením ceny výroby desky plošných spojů je celková cena zařízení 3625 Kč. Výrobou desky u levnějšího výrobce, například u výrobců z Číny, by bylo možné cenu desky snížit alespoň na stovky korun. Cenu součástek je také možné redukovat nákupem součástek ve větších množstvích, ovšem největší část ceny bude vždy tvořit cena kamery a PoE modulu.

10.5 Další možný vývoj

Zařízení rozhodně lze ještě mnohem vylepšit, hlavně co se týče samotného hardwaru. Velikost zařízení by se dala redukovat rozdělením desky plošných spojů na dvě části, tak aby obvodové řešení audio části, RFID čtečky a dotykovou plochou tlačítka bylo umístěno odděleně. Tato druhá deska by pak mohla být umístěna přímo nad tu první.

Pro reálné využití zařízení je také nutné vše uzavřít do vodotěsné krabičky, což mám určitě v plánu. Jak mikrofon, tak reproduktor jsou připraveny tak, aby bylo možné obojí provést. Mikrofon se totiž nachází v horní části desky, bez větších součástek v okolí a stačí tak tedy v krabiččce vytvořit výřez s vodotěsnou membránou. Reproduktor je pro změnu vyveden směrem ke spodní části zařízení, tak aby mohl směřovat směrem dolů a být tak odstíněn od případného deště.

Po softwarové části by určitě bylo vhodné přidat další možnosti konfigurace zařízení, například přidat možnost autentizace RTSP serveru. Mikrokontrolér také obsahuje rozšíření instrukční sady pro akceleraci výpočtů s 128bitovými vektorovými daty. Je tak možné lokálně provádět inferenci menších modelů umělé inteligence. Firmware by se tedy dal rozšířit například o detekci tváří či podobné.

Kapitola 11

Závěr

Cílem této práce bylo vytvořit přístupový systém s kamerou, jenž lze připojit k systému chytré domácnosti Home Assistant za pomoci vytvořené integrace, a jehož konfigurace bude dostupná ve webovém prohlížeči. Všechny tyto cíle byly splněny. Vlastní vytyčený cíl, a to přidání přenosu také audio záznamů, byl však splněn pouze částečně, neboť audio přenos vyžaduje další odladění pro reálné použití.

Během práce byla analyzována problematika napájení exteriérových zařízení se zaměřením na drátové technologie. Dále byl proveden popis systémů chytrých domácností se zaměřením na systém Home Assistant a také popis způsobů přenosu audiovizuálních dat po síti.

Na základě získaných poznatků byl proveden návrh řešení, kde byla zvolena technologie Power over Ethernet pro napájení celého zařízení. Pro přenos audiovizuálních dat byl zvolen protokol RTSP společně s protokolem RTP. Pro provedení integrace se systémem Home Assistant byl zvolen proces MQTT discovery. Zvolena byla technologie RFID na frekvenci 13,56 MHz pro autentizaci uživatelů za pomocí karet.

Dle provedeného návrhu řešení byla vytvořena obvodová realizace ve formě desky plošných spojů s mikrokontrolérem ESP32-P4 a všemi klíčovými částmi jak Power over Ethernet napájení, kamerou, RFID čtečkou a audio kodekem. Celé zařízení bylo postupně osazeno součástkami a zprovozněno. Pro danou desku byl implementován obslužný firmware ve frameworku ESP-IDF s webovým administračním rozhraním zabudovaným přímo do zařízení. Toto rozhraní bylo vytvořeno s použitím frameworku SvelteKit.

Po dokončení byly otestovány jednotlivé části zařízení, obzvláště pak jeho připojení k systému Home Assistant. V aktuální formě stačí zařízení vytvořit vodotěsnou krabici a bylo by použitelné.

Tvorba této práce mě naučila lépe porozumět návrhu desek plošných spojů, obzvláště složitějších mikrokontrolérů jenž pro své fungování vyžadují větší množství součástek. Také jsem se naučil rozvržení tras datových signálů s vyššími frekvencemi, jenž pro korektní fungování vyžadují specifickou impedanci. Z ohledu tvorby software jsem se pak naučil tvorbu komplexnějších projektů ve frameworku ESP-IDF s více komponenty.

Literatura

[1] Česko. Nařízení vlády č. 194 ze dne 22. června 2022 o požadavcích na odbornou způsobilost k výkonu činnosti na elektrických zařízeních a na odbornou způsobilost v elektrotechnice. In: *Sbírka zákonů České republiky*. 2022, částka 89, s. 2396–2409. ISSN 1211-1244.

[2] Chakraborty, A.; Islam, M.; Shahriyar, F.; Islam, S.; Zaman, H. U. et al. Smart Home System: A Comprehensive Review. *Journal of Electrical and Computer Engineering*, 2023, sv. 2023, č. 1, s. 7616683. Dostupné z: https://onlinelibrary.wiley.com/doi/abs/10.1155/2023/7616683.

[3] Checkpoint Systems. *What is RFID? Everything you need to know about this technology* online. Dostupné z: https://checkpointsystems.com/uk/blog/what-is-rfid/. [cit. 2025-05-16].

[4] Cisco. *What is Power over Ethernet (PoE)?* online. Dostupné z: https://www.cisco.com/c/en/us/solutions/enterprise-networks/what-is-power-over-ethernet.html. [cit. 2025-05-16].

[5] David D. Coleman and David A. Westcott. Power over Ethernet (PoE). In: *CWNA Certified Wireless Network Administrator Study Guide: Exam CWNA-107*. John Wiley Sons, 2018, s. 443–469. Dostupné z: https://sci-hub.se/https://doi.org/10.1002/9781119549406.ch12.

[6] H. Schulzrinne and A. Rao and R. Lanphier. Real Time Streaming Protocol (RTSP). *RFC 2326* Request for Comments, April 1998. Dostupné z: https://www.rfc-editor.org/rfc/rfc2326.html. Obsoleted by RFC 7826.

[7] H. Schulzrinne and S. Casner. RTP Profile for Audio and Video Conferences with Minimal Control. *RFC 3551* Request for Comments, July 2003. Dostupné z: https://www.rfc-editor.org/rfc/rfc3551.html. Obsoletes RFC 1890.

[8] H. Schulzrinne and S. Casner and R. Frederick and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. *RFC 3550* Request for Comments, July 2003. Dostupné z: https://www.rfc-editor.org/rfc/rfc3550.html. Obsoletes RFC 1889.

[9] Hasan, M.; Biswas, P.; Bilash, M. T. I. a Dipto, M. A. Z. Smart Home Systems: Overview and Comparative Analysis. In: *2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*. 2018, s. 264–268.

[10] Home Assistant Community. *Home Assistant Documentation* https://www.home-assistant.io/docs/. 2025. Dostupné z: https://www.home-assistant.io/docs/. Accessed: 2025-05-19.

[11] Intellinet Europe. *Power over Ethernet – Everything you need to know* online. Dostupné z: https://intellinetnetwork.eu/pages/power-over-ethernet. [cit. 2025-05-16].

[12] International Telecommunication Union. *Pulse Code Modulation (PCM) of Voice Frequencies*. Recommendation G.711. ITU-T, 1988. Dostupné z: https://www.itu.int/rec/T-REC-G.711.

[13] Jaroslav Kováč. *PoE IEEE 802.3af, IEEE 802.3at a IEEE 802.3bt* online. Dostupné z: https://jkovac.cz/poe-ieee-802-3af-ieee-802-3at-ieee-802-3bt/. [cit. 2025-05-16].

[14] Kalva, H. The H.264 Video Coding Standard. *IEEE MultiMedia*, 2006, sv. 13, č. 4, s. 86–90.

[15] Kertous, M.; Zerroug, A. a Zerroug, N. Review of Technics Used In Smart House. In: *2024 2nd International Conference on Electrical Engineering and Automatic Control (ICEEAC)*. 2024, s. 1–5.

[16] Kevin Kilbane. *Understanding the IEEE 802.3bt PoE Standard*. White Paper. Skyworks Inc., 2022. Dostupné z: https://www.skyworksinc.com/-/media/SkyWorks/SL/documents/public/white-papers/understanding-the-ieee-8023bt-poe-standard.pdf. [cit. 2025-05-16].

[17] Marpe, D.; Wiegand, T. a Sullivan, G. The H.264/MPEG4 advanced video coding standard and its applications. *IEEE Communications Magazine*, 2006, sv. 44, č. 8, s. 134–143.

[18] Microchip Technology Inc.. *microID 125 kHz RFID System Design Guide*. Microchip Technology Inc., 1998. Dostupné z: https://ww1.microchip.com/downloads/en/devicedoc/51115f.pdf. System Design Guide; Copyright 1998.

[19] NETGEAR Team. *Active or Passive PoE, That is the Question* online. Dostupné z: https://www.netgear.com/hub/business/network/active-or-passive/. [cit. 2025-05-16].

[20] Raj, A. a Steingart, D. Power sources for the internet of things. *Journal of The Electrochemical Society*. IOP Publishing, 2018, sv. 165, č. 8, s. B3130.

[21] Sheldon. *How to Choose PoE Cable for Power Over Ethernet Network?* online. Dostupné z: https://www.fs.com/blog/useful-tips-for-selecting-poe-cables-61.html. [cit. 2025-05-16].

[22] Silicon Laboratories. *Si3406x Family Data Sheet*. Silicon Laboratories, 2022. Dostupné z: https://www.skyworksinc.com/-/media/Skyworks/SL/documents/public/data-sheets/si3406x-datasheet.pdf. Datasheet, Revision 1.1.

[23] Spiceworks contributors. *What is RFID* online. Dostupné z: https://www.spiceworks.com/tech/tech-general/articles/what-is-rfid/. [cit. 2025-05-16].

[24] Thopate, K.; Shinde, S.; Mahajan, R.; Bhagat, R.; Joshi, P. et al. Keyless security: the smart solution for home with a smart door lock. *International Journal on Recent and Innovation Trends in Computing and Communication*, 2023, sv. 11, 8s, s. 170–174.

[25] TT Electronics. *RFID: The Technology Making Industries Smarter* online. Dostupné z: https://www.ttelectronics.com/blog/rfid-technology/. [cit. 2025-05-16].

[26] Wang, J.; Zhang, J.; Saha, R.; Jin, H. a Kumar, S. Pushing the range limits of commercial passive {RFIDs}. In: *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 2019, s. 301–316.

[27] Wenger, S. H.264/AVC over IP. *IEEE Transactions on Circuits and Systems for Video Technology*, 2003, sv. 13, č. 7, s. 645–656.

[28] Y.-K. Wang and R. Even and T. Kristensen and R. Jesup. RTP Payload Format for H.264 Video. *RFC 6184* Request for Comments, May 2011. Dostupné z: https://www.rfc-editor.org/rfc/rfc6184.html. Obsoletes RFC 3984.

[29] Z-Wave Alliance. *Certification Process* online. Dostupné z: https://z-wavealliance.org/development-process-overview-2/. [cit. 2025-05-16].

Příloha A

Obsah přiloženého média

```
/
├── MainFirmware/ ..... hlavní firmware zařízení
│   ├── components/ ..... implementované a stáhnuté komponenty
│   └── main/ ..... zdrojové kódy logiky firmwaru
├── PCB/ ..... deska plošných spojů
│   ├── Motherboard/ ..... schémata a rozložení desky
│   └── Components/ ..... použité symboly a půdorysy součástek
├── WebManage/ ..... zdrojové kódy webového rozhraní
├── latex/ ..... zdrojové kódy technické zprávy
└── zprava.pdf ..... výsledná technická zpráva
```

Příloha B

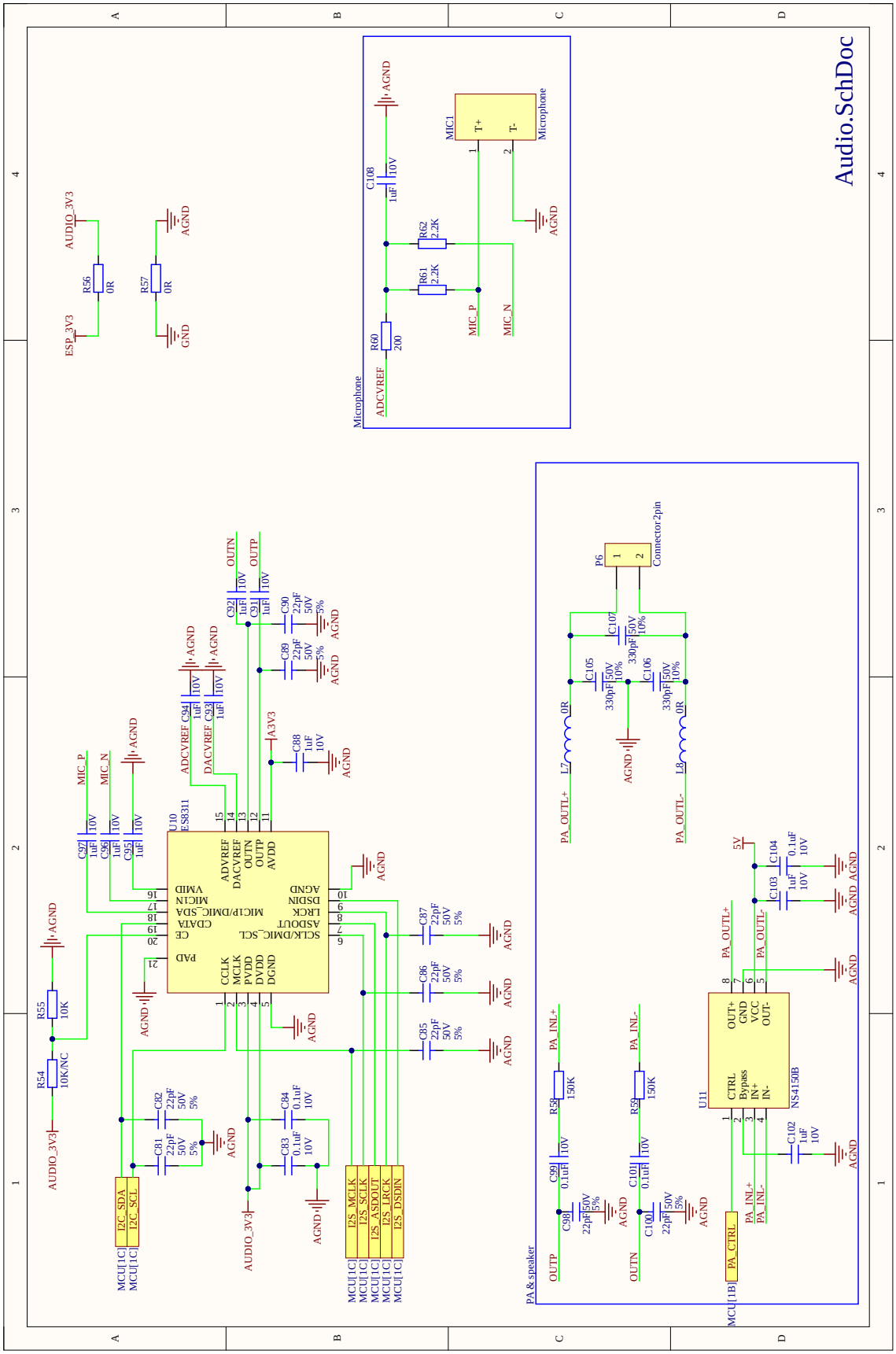
Koncové body API

Cesta	Metoda	Popis
/api/login	POST	Provede přihlášení a vrací přístupový token
/api/test	GET	Otestuje validitu přístupového tokenu
/api/logout	POST	Odhlásí/zneplatní použitý přístupový token
/api/changePassword	POST	Mění přístupové heslo administrace
/api/http	GET	Vrací konfiguraci HTTP serveru
/api/http	POST	Nastavuje konfiguraci HTTP serveru
/api/http/cert	GET	Vrací aktuální nastavený HTTPS certifikát
/api/http/cert	POST	Nastavuje certifikát HTTPS serveru
/api/http/privkey	POST	Nastavuje privátní klíč HTTPS serveru
/api/mdns	GET	Vrací konfiguraci MDNS služby
/api/mdns	POST	Nastavuje MDNS službu
/api/ethernet	GET	Vrací konfiguraci síťového rozhraní
/api/ethernet	POST	Nastavuje síťové rozhraní
/api/mqtt	GET	Vrací konfiguraci MQTT klienta
/api/mqtt	POST	Nastavuje MQTT klienta
/api/mqtt/certificate	GET	Vrací aktuální certifikát MQTT brokera
/api/mqtt/certificate	POST	Nastavuje certifikát MQTT brokera
/api/rfid	GET	Vrací aktuální validační režim RFID
/api/rfid	POST	Nastavuje validační režim RFID
/api/rfid/tags	GET	Vrací seznam uložených RFID karet
/api/rfid/tags	POST	Přidává novou RFID kartu
/api/rfid/tags/id	DELETE	Odstraňuje RFID kartu
/api/rfid/tags/id	PUT	Upravuje RFID kartuL
/api/intervals	GET	Vrací nastavení časových intervalů relé
/api/intervals	POST	Nastavuje časové intervaly obsluhy relé
/api/reboot	POST	Restartuje celé zařízení
/api/status	GET	Vrací aktuální stav otevřenosti relé
/api/relay	POST	Nastavuje stav otevřenosti relé

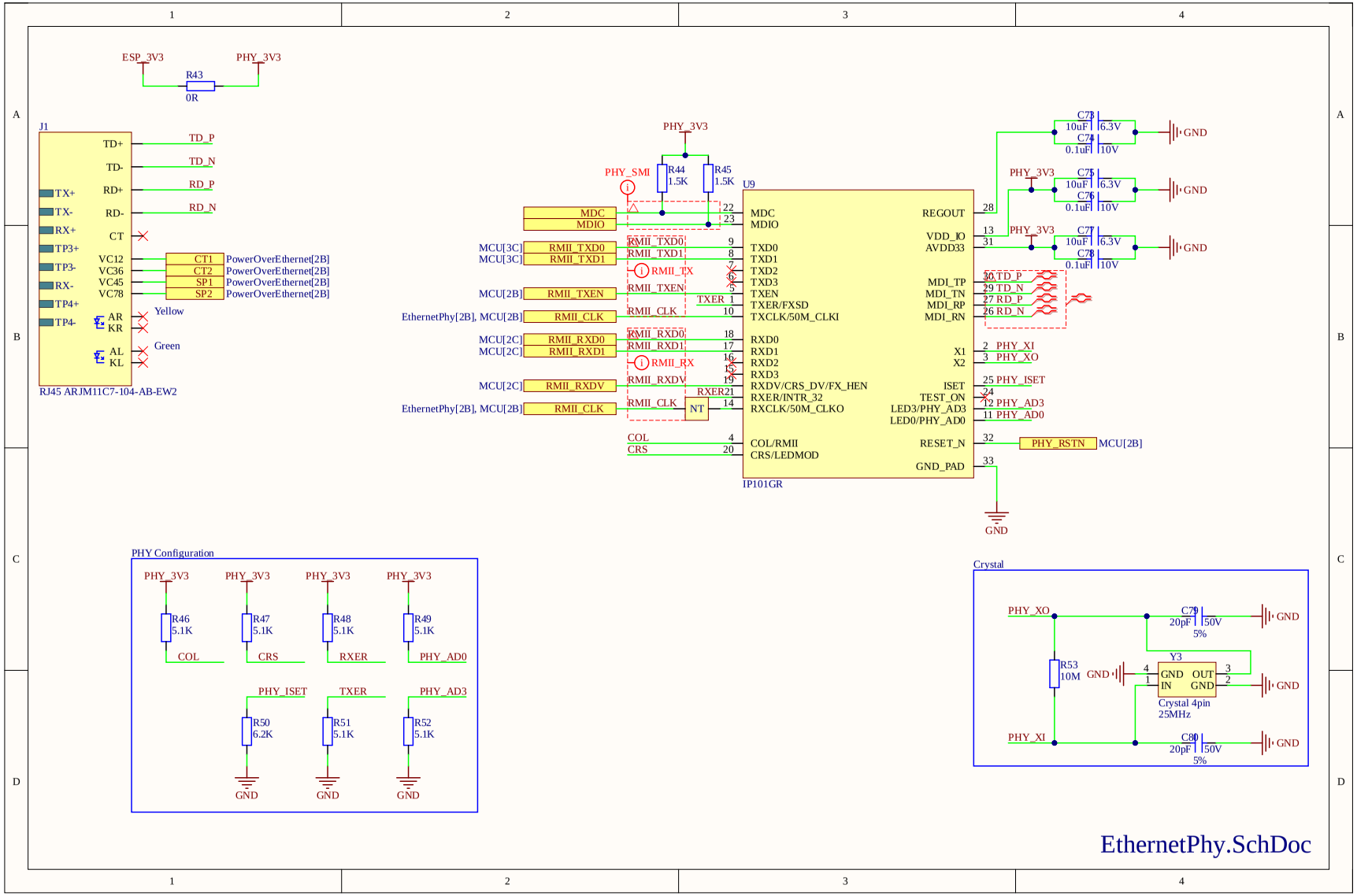
Tabulka B.1: Koncové body webové REST API zařízení

Příloha C

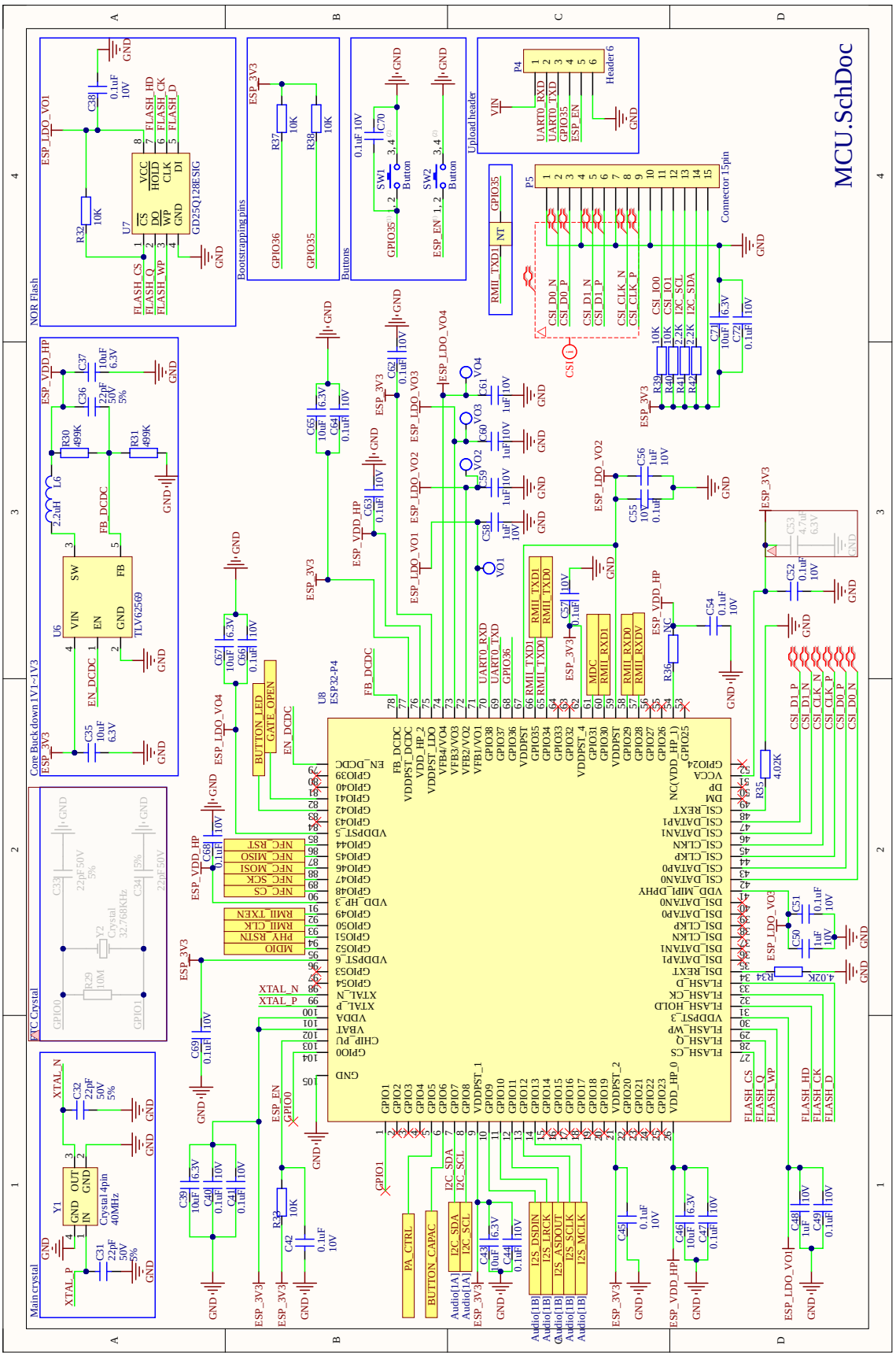
Schémata obvodového zapojení



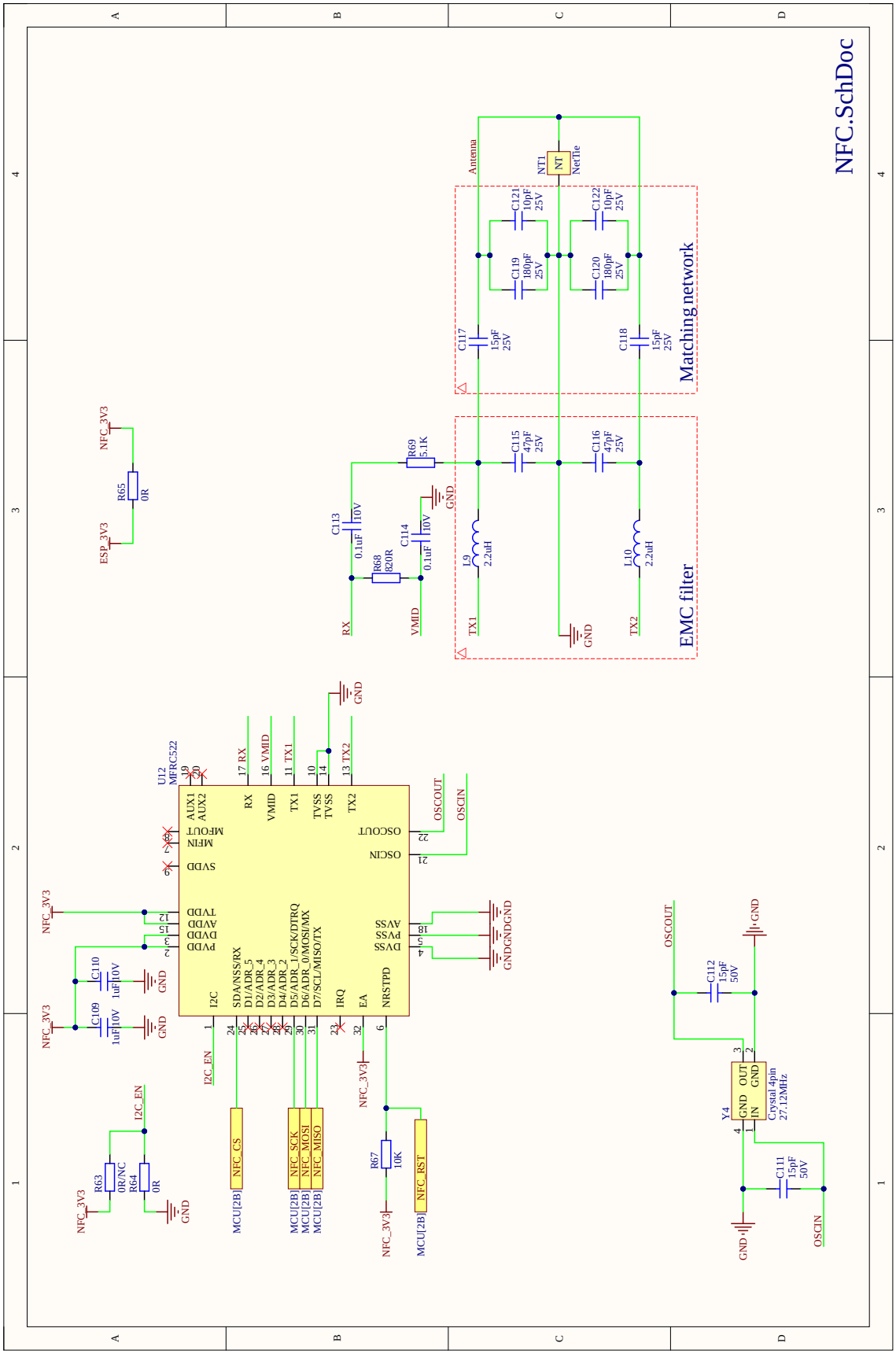
Obrázek C.1: Schéma zapojení audio kodeku, mikrofonu a reproduktoru



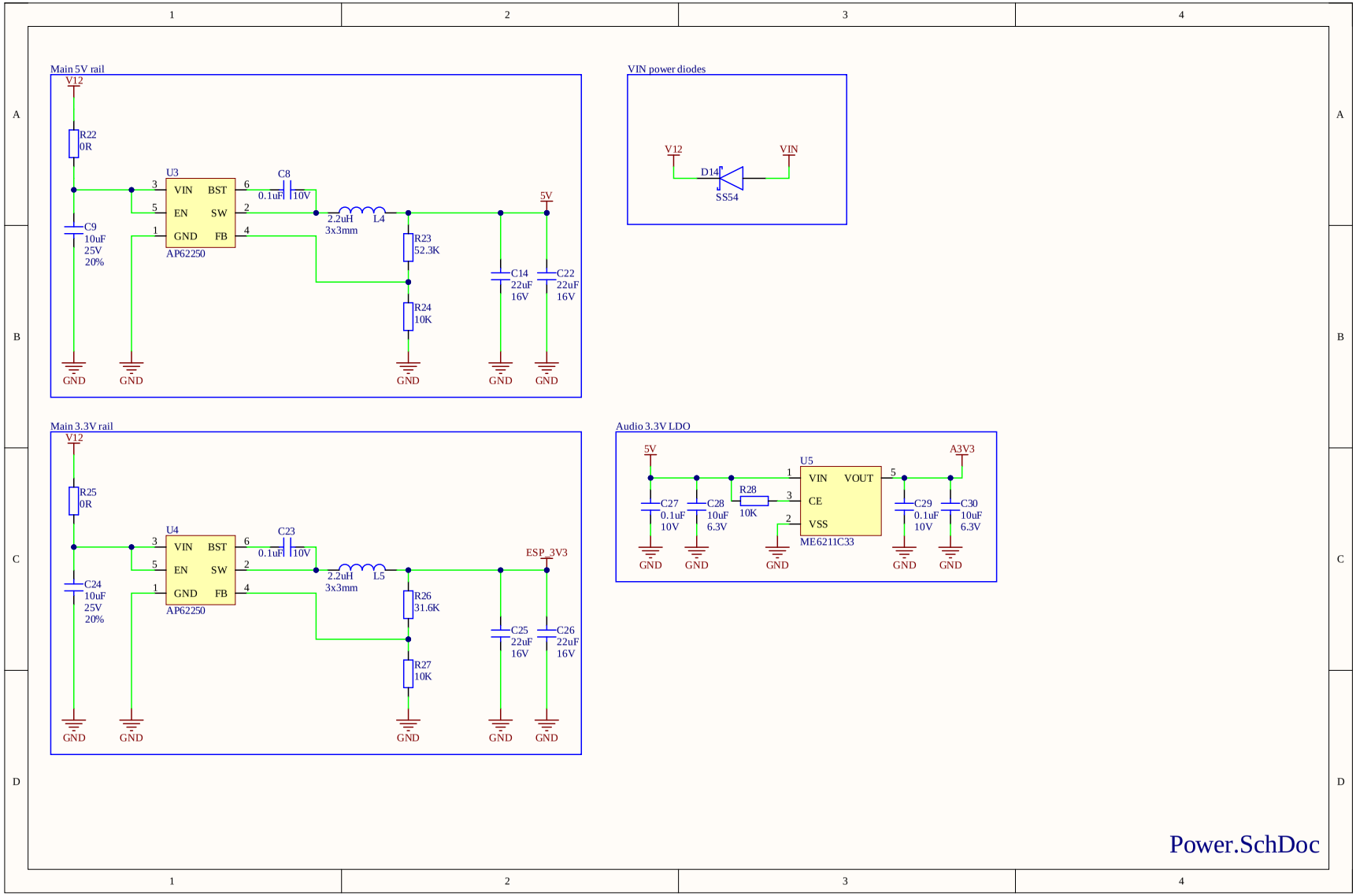
Obrázek C.2: Schéma zapojení ethernet PHY čipu a RJ45 konektoru



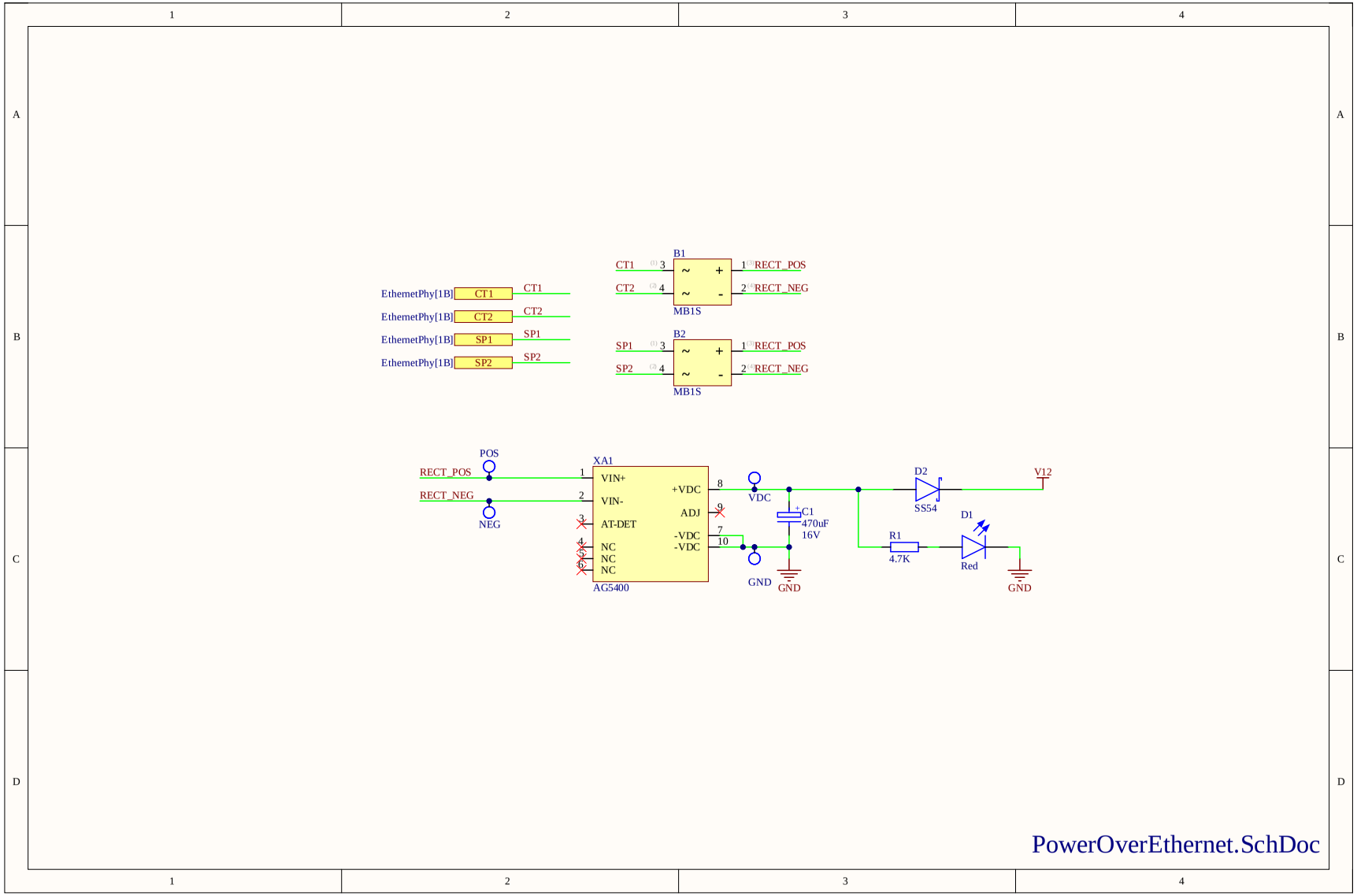
Obrázek C.3: Schéma zapojení mikrokontroléru ESP32P4, jeho podpůrných součástí a MIPI CSI konektoru



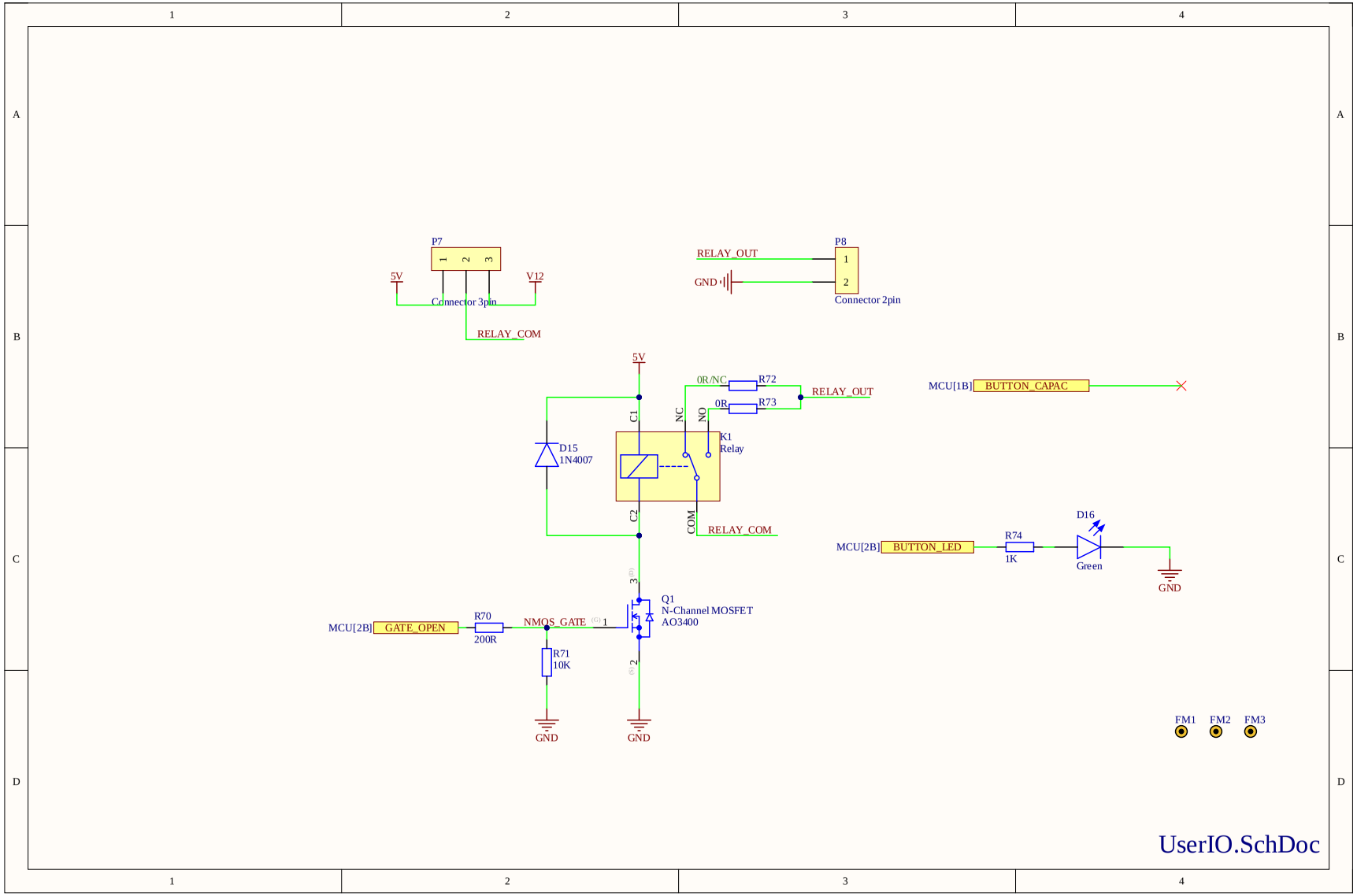
Obrázek C.4: Schéma zapojení RFID čtečky



Obrázek C.5: Schéma zapojení napěťových spínacích regulátorů



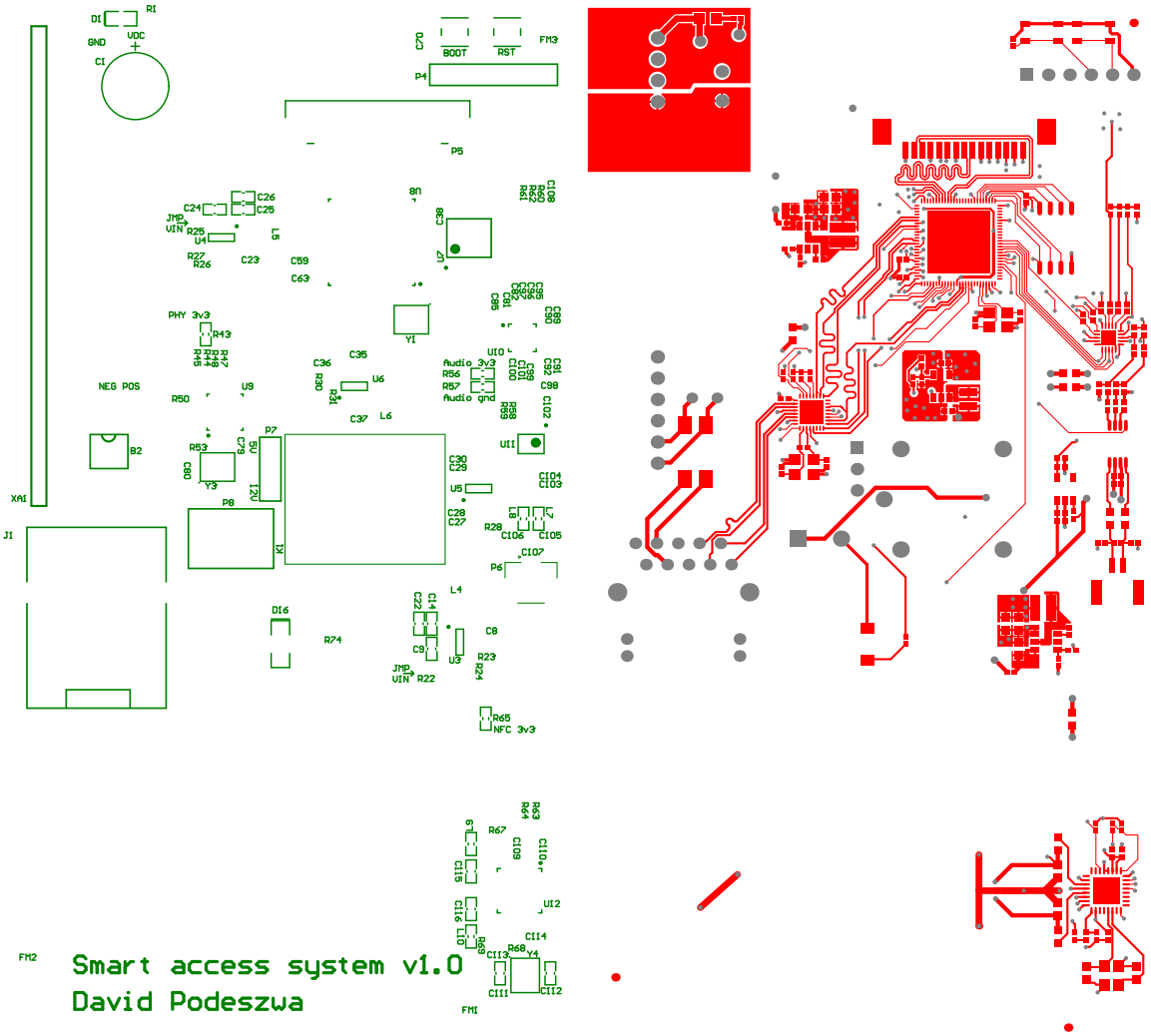
Obrázek C.6: Schéma zapojení Power over Ethernet modulu



Obrázek C.7: Schéma zapojení relé a LED

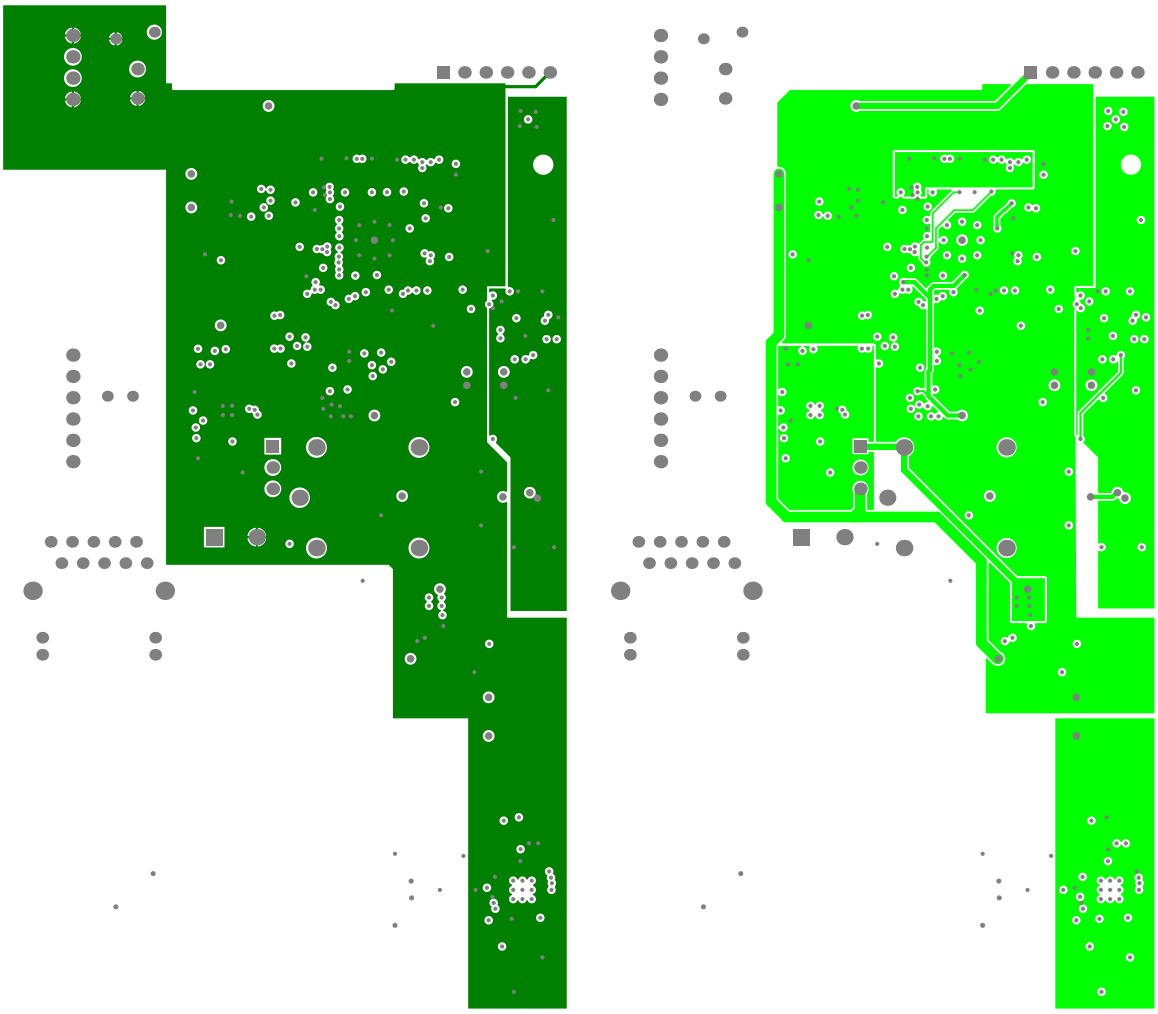
Příloha D

Vrstvy desky plošných spojů



(a) Popisky horní vrstvy desky (*Top overlay*) (b) Horní vrstva desky (*Top layer*)

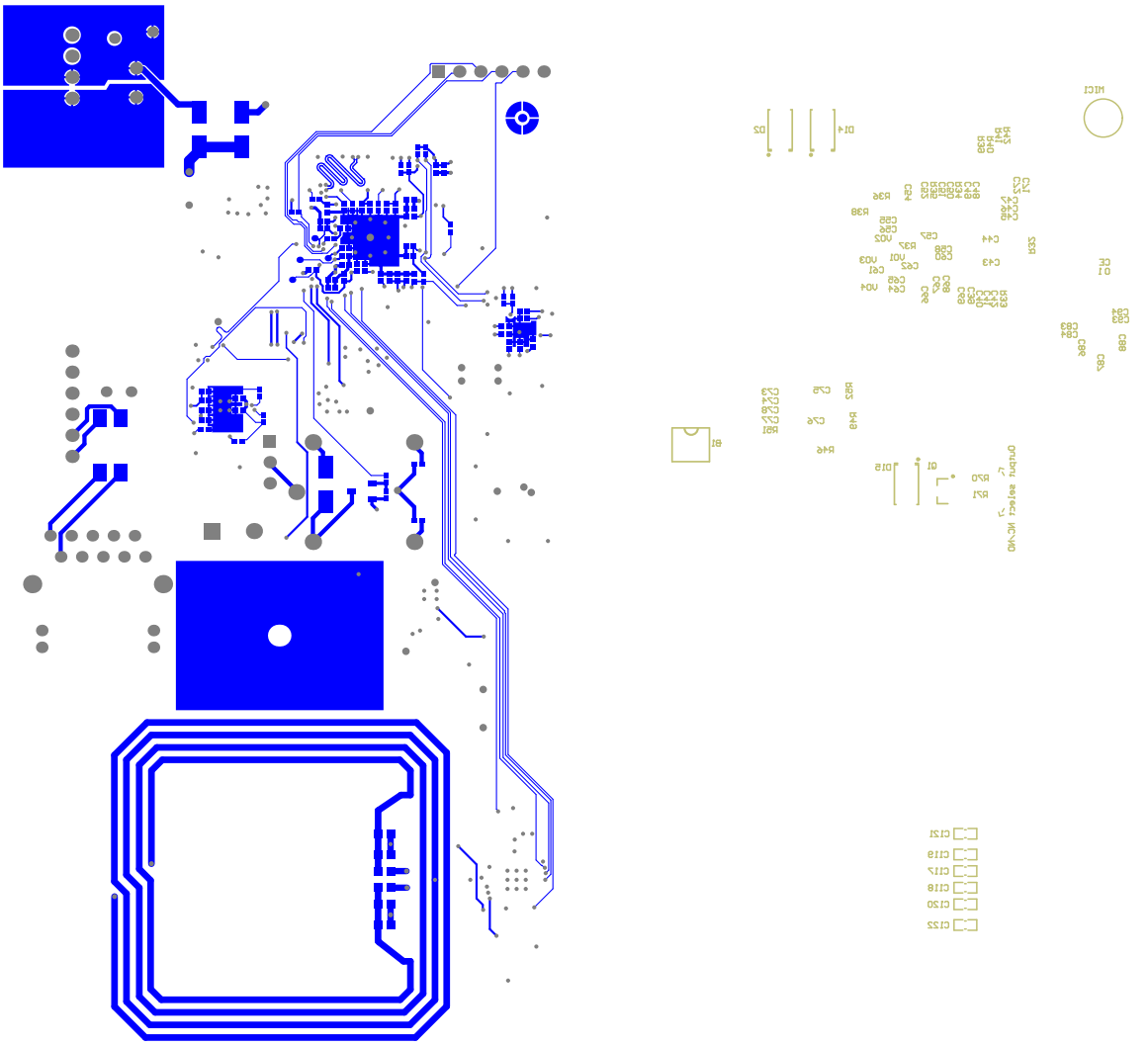
Obrázek D.1: Horní vrstvy desky



(a) Vnitřní vrstva se zemí (*Layer 1*)

(b) Vnitřní vrstva s napěťovými větvemi (*Layer 2*)

Obrázek D.2: Vnitřní vrstvy desky



(a) Spodní vrstva desky (*Bottom layer*) (b) Popisky spodní vrstvy desky (*Bottom overlay*)

Obrázek D.3: Spodní vrstvy desky