FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Chytrý dům 4.0
Student:	Bc. Jakub Kyzek
Vedoucí:	Ing. Martin Daňhel, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Návrh a programování vestavných systémů
Katedra:	Katedra číslicového návrhu
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Rozšiřte svou původní bakalářskou práci, která se zabývala návrhem a implementací chytrého domu.

Rozšíření bude spočívat v následujících klíčových bodech:
- Proveďte analýzu možných vylepšení systému a soustřeďte se na možnost využití IoT a zvažte využití protokolu MQTT.
- Přechod z RS-485 na ethernet/internet pro možné použití IoT (IPV4 resp. IPV6 dle analýzy).
- Návrh a realizace vlastního plošného spoje pro řízení chytrého domu.
- Vylepšení vzdáleného ovládání periferií bez nutnosti aktualizace firmwaru při použití dalších modulů.
- Dbejte na jednoduchost následné instalace a údržby celého systému.

Chytrý dům bude implementovat funkčnosti z původního projektu bakalářské práce.
Ovládání bude dostupné přes webové rozhraní.
Navržené řešení implementujte v reálných podmínkách (na skutečném domě).
Vytvořený systém otestujte.

Elektronicky schválil/a prof. Ing. Hana Kubátová, CSc. dne 30. června 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Chytrý dům 4.0

Bc. Jakub Kyzek

Katedra číslicového návrhu

Vedoucí práce: Ing. Martin Daňhel, Ph.D.

6. května 2022

Poděkování

Chtěl bych poděkovat vedoucímu práce Ing. Martinovi Daňhelovi, Ph.D. za věcné připomínky a rady.

Dále své rodině za podporu, kterou mi poskytla při studiu. A v neposlední řadě bych rád poděkoval svým přátelům. Zejména pak Ing. Martinovi Vitouškovi za pomoc při vzniku této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisu. Dále prohlašuji, že jsem s Českým vysokým učením technickým v Praze uzavřel dohodu, na jejímž základě se ČVUT vzdalo práva na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona. Tato skutečnost nemá vliv na ustanovení § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisu.

V Praze dne 6. května 2022 ……

České vysoké učení technické v Praze

Fakulta informačních technologií

Odkaz na tuto práci

Kyzek, Jakub. *Chytrý dům 4.0*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Abstrakt

Tato práce se zabývá návrhem a implementací systému pro chytrou domácnost jako součást projektu společnosti Kyvit s.r.o. Projekt volně navazuje na myšlenky a realizaci představenou v bakalářské práci *Návrh a implementace chytrého domu*. Diplomová práce se dále zabývá celkovou restrukturalizací projektu, který již není zaměřen na nízké pořizovací náklady, nýbrž na efektivitu a uživatelské pohodlí. Oproti původnímu návrhu v bakalářské práci jsou zde navržena a realizována mnohá vylepšení jako např. využití IoT nebo návrh a tvorba vlastní řídící jednotky – plošného spoje, který je osazený výkonným mikrokontrolérem STM32. Projekt si dále klade za cíl respektovat hlavní myšlenky představené konceptem Průmysl 4.0. Výstupem této práce je reálný prototyp funkčního systému.

Klíčová slova chytrý dům, Internet věcí, mikrokontrolér STM32, databáze, desky plošných spojů, vestavné systémy, ethernet

Abstract

This thesis presents the design and implementation of own smart home system as part of the project of the company Kyvit s.r.o. The project loosely follows the ideas and implementation introduced in the bachelor's thesis *SmartHome – Design and Implementation*. The thesis also deals with the overall restructuring of the project, which is no longer focused on low acquisition costs but on efficiency and user comfort. Compared to the original design in the bachelor's thesis, many improvements are designed and implemented here, such as the usage of IoT or the design and creation of own control unit – a printed circuit board, which is equipped with a powerful microcontroller STM32. The project also aims to respect the main ideas presented by Industry 4.0. The output of this work is a real prototype of a functional system.

Keywords smart home, Internet of Things, microcontroller STM32, database, printed circuit board, embedded systems, ethernet

Obsah

Úvod **1**
Motivace 2
Cíle práce 3

1 Teoretická část a analýza problému chytrých domů **5**
1.1 Popis systému z předchozí práce 6
1.2 Analýza možností vylepšení původního systému 6
1.3 Definování pojmu – Internet of Things (IoT) 8
1.4 Principy komunikace po internetu 8
1.4.1 Ethernet 10
1.4.2 Internet Protocol (IP) 10
1.4.3 Transmission Control Protocol (TCP) 11
1.4.4 Transport Layer Security (TLS) 12
1.4.5 MQTT protokol a jeho využití v IoT 12
1.5 Porovnání rozdílů mezi verzemi internetového protokolu (IP) . 13
1.6 Napájení přes síťový kabel – Power over Ethernet (PoE) 14
1.7 Zapojení RJ45 podle ANSI/TIA-568 a druhy kabeláže 15
1.8 Návrhový software pro vývoj plošných spojů 16
1.9 Volba nového mikrokontroleru 16
1.9.1 Popis mikrořadiče STM32F103C8T6 17
1.9.2 Zdroje hodinového signálu u vybraného mikrořadiče . . 17
1.10 Hardwarové síťové řadiče 18
1.11 Nástroj FastAPI pro tvorbu backendu 19
1.12 Webový Framework Flask 19
1.13 Možnosti zabezpečení jednotlivých komponent systému 19
1.13.1 Hardwarová jednotka chytrého domu 20
1.13.2 Zabezpečení síťové infrastruktury 20
1.13.3 Požadavky na zabezpečení backendu 21

2 Návrh jednotlivých komponent systému **23**
2.1 Celkový návrh infrastruktury systému – příklady použití 23
2.1.1 Interakce uživatele s webovým rozhraním 23
2.1.2 Propojení jednotek se serverem v rámci systému 24
2.2 Úvod k návrhu plošného spoje 25
2.3 Zdroj napájení pro desku 26
2.3.1 Analýza maximálního zatížení napájení desky 26
2.3.2 Návrh vlastního zdroje napájení 27
2.4 Návrh obvodu – hlavní mikrořadič 28
2.4.1 Vedlejší potřebné komponenty 28
2.4.2 Obvod vysokorychlostních hodin 29
2.4.3 Hardware na desce potřebný pro ladění 30
2.5 Návrhy obvodů zajišťující konektivitu jednotky 30
2.5.1 Internetový řadič W5500 31
2.5.2 Ethernetový konektor 31
2.5.3 Konektory pro připojení ostatních modulů a senzorů . . 32
2.6 Dokončení návrhu plošného spoje 33
2.7 Návrhy vylepšení databáze 35
2.7.1 Vhodné použití indexů 37
2.7.2 Výsledné zlepšení výkonu 37

3 Implementace **39**
3.1 Fyzická architektura síťové infrastruktury 39
3.2 Tovární výroba plošného spoje a náklady s tím spojené 40
3.3 Firmware pro jednotku 42
3.3.1 Náhrávání firmware do desky 43
3.3.2 Otázky zabezpečení firmware a zranitelnosti použitelného mikrořadiče 44
3.4 Použití externích knihoven ve firmware 44
3.4.1 Ethernet 45
3.4.2 NTPClient 45
3.4.3 SSLClient/BearSSL 45
3.4.4 arduino-mqtt 46
3.5 Vnější ochranný obal 47
3.6 Finální zapojení 47
3.7 Tvorba webového rozhraní 49
3.8 Zabezpečení frontendu a backendu 49

4 Testování **51**
4.1 Poučení z testování jednotek původního systému 51
4.2 Testování plošného spoje 52
4.2.1 Problémy při měření – zemní smyčky 52
4.2.2 Prvotní ověření funkčnosti plošného spoje po výrobě . . 52
4.2.3 Zvýšená teplota PCB 54

4.2.4 Důkladnější kontrola elektronické funkčnosti desky . . . 55
4.3 Stará a nová verze webového rozhraní (frontend) 61

Závěr **63**

Literatura **65**

A Seznam použitých zkratek **67**

B Schémata navržené desky **69**

C Obsah přiložené paměťové karty **77**

Seznam obrázků

1.1 Návrhové schéma původního systému 6
1.2 Deska používaná v první verzi 7
1.3 Vrstvy při MQTT komunikaci 9
1.4 Surový MQTT packet se všemi vrstvami 9
1.5 MQTT packet - vrstvy dekódované 10
1.6 Zapojení ethernetového kabelu podle TIA-568B 15
1.7 Schéma rozvodu nízko rychlostních hodin u STM32 18

2.1 Připojení z webového klienta na server 24
2.2 Připojení jednotky na server 25
2.3 Prototyp s vývojovou deskou NUCLEO-F103RB 26
2.4 Výsledný návrh napájecí sekce desky 28
2.5 Schéma zapojení oscilátoru u STM32 29
2.6 Návrh plošného spoje – zapojení oscilátoru u STM32 30
2.7 Zapojení konektorů pro ladění 31
2.8 Zapojení konektoru HY931147C 32
2.9 Ukázka vybraných konektorů 33
2.10 Finální návrh PCB (přední strana) 34
2.11 Finální návrh PCB (zadní strana) 34
2.12 Výsledná podoba návrhu desky jako 3D modelu 35

3.1 Architektura síťové infrastruktury 41
3.2 Vyrobená deska 41
3.3 Nastavení hodinového signálu v STM32CubeIDE 43
3.4 Vnější ochranný obal chránící plošný spoj 48
3.5 Zapojení jednotek do switche 48

4.1 Měření na osciloskopu – napětí na oscilátoru pro hlavní procesor . 53
4.2 Připojení desky k debuggeru – programování a ladění 54
4.3 Schéma k měření 4.4 56

4.4 Měření napětí na oscilátoru pro hlavní procesor 56
4.5 Schéma k měření 4.6 57
4.6 Měření napětí na oscilátoru pro W5500 57
4.7 Schéma k měřením 4.8 a 4.9 58
4.8 Měření napětí na záporném pólu diody SK33 (D2) ve zdroji 58
4.9 Měření napětí na diodě ve zdroji – detail 59
4.10 Schéma k měření 4.11 60
4.11 Měření hodinového signálu na komunikační sběrnici SPI 60
4.12 Nové uživatelské rozhraní pro vykreslování detailních grafů 61

Seznam tabulek

1.1 Kategorie síťových kabelů 16

2.1 Databázová tabulka pro data z původního návrhu 35
2.2 Dvě datové tabulky po dekompozici 36

Seznam zdrojových kódů

3.1 Kód přepisující chování WEAK funkce inicializující hodiny . . 44
3.2 Použití knihovny NTPClient 46
3.3 Ukázka rozhraní knihovny arduino-mqtt 47
3.4 Kód vykreslující hlavní stránku 49

Úvod

Název této práce je drobnou narážkou na technologický koncept známý také jako Průmysl 4.0. Ještě před nástupem na vysokou školu, a potom také během svého studia na Fakultě informačních technologií jsem se zabýval několika projekty týkající se vestavných systémů. Tento směr mi byl po celou dobu velmi blízký, a proto jsem se rozhodl věnovat se vývoji vlastního systému pro takzvaný Chytrý dům. V rámci tohoto mého původně *hobby projektu* vznikly postupně tři iterace s tím, že druhá z nich byla použita jako téma mé bakalářské práce. Rozhodl jsem se tedy podpořit tento začínající trend a čtvrtou iteraci použít pro svou závěrečnou diplomovou práci.

A nyní proč ta narážka na Průmysl 4.0 a o co že to vlastně jde? Slovní spojení Průmysl 4.0 (anglicky *Industry 4.0*, zkráceně *I4.0*) je označení pro takzvanou 4. průmyslovou revoluci. Heslo je to sice úderné, ale co to v praxi znamená? Jedná se obecně o jakýsi souhrnný název pro současný trend digitalizace a s ní spojený hlubší rozvoj propojování inteligentních systémů v průmyslu. Hlavními koncepty jsou kyber-fyzikální systémy (pokročilá propojení „počítač-fyzický objekt“), internet věcí (IoT – vzájemné propojení a integrace zařízení), internet služeb (jako IoT, ale pro služby a lidi) a digitální ekonomika. Kompletní propojení chytrých systémů pak umožňuje decentralizované rozhodování, založené na vzájemné komunikaci jednotlivých zařízení a sdílení dat [1]. Co je pro účely této práce důležité, je fakt, že koncept *chytrých domácností* vychází právě z myšlenek Průmyslu 4.0. A proto také chci, aby práce dostála svému jménu i ve vztahu k Průmyslu 4.0.

Záměr pro vytvoření takového systému je od začátku jednoduchý. Lidem se líbí možnosti, které Chytrý dům nabízí – regulace teploty, zatahování žalu-

[1] Samo sebou se jedná o další soubor ještě údernějších hesel, která si laskavý čtenář v případě zájmu jistě snadno sám vyhledá

zií, rozsvícení a zhasínání, automatické měření kvality vody a krmení rybiček v akváriu nebo sledování obsahu ledničky. Dále jsou pak takové systémy velmi zajímavé z pohledu získávaných dat a jejich dalšího možného využití pro analýzy a predikce. Výše uvedené příklady se tak dají rozvinout do mnohem zajímavějších aplikací. Například nám data mohou napovědět, jak ušetřit za topení, kterou zeď lépe izolovat kvůli únikům tepla, lépe dávkovat krmení rybičkám tak, aby se voda kazila pomaleji, nebo generovat nákupní seznamy na základě stavu našich zásob v lednici a rychlosti spotřeby jednotlivých výrobků. Možností je nepřeberné množství a zajímavá aplikace se dá vymyslet pro mnoho z nich.

Čím se však tato práce bude lišit od předchozích verzí? Všechny předchozí iterace mého Chytrého domu fungovaly na svých vlastních sběrnicích, nicméně tentokrát jsem se rozhodl využít výhod IoT a tím usnadnit instalaci a zvýšit praktičnost celého řešení. Další neméně důležitou částí práce bude návrh a výroba vlastní desky plošného spoje, která by měla sloužit jako základní hardwarová jednotka pro budoucí vývoj tohoto projektu. Celkově je však práce pojata především jako základní návrh vlastního konceptu, který by měl být dále rozvíjen. Rozhodl jsem se totiž, že se chci tomuto tématu dále věnovat, a to na profesionální úrovni. S mým kolegou a kamarádem *Ing. Martinem Vitouškem* jsme založili firmu *Kyvit s.r.o.*, jejíž hlavním obchodním záměrem bude (alespoň prozatím) rozvoj systémů inteligentních domů a jejich propojení s dalšími systémy. Martin se v rámci svého doktorského studia na Ústavu přístrojové a řídící techniky na Fakultě strojní ČVUT v Praze zabývá tématem pokročilého řízení strojů a procesů.

Stejně jako všechny iterace bude i tato vyvíjena a testována na rodinném domě mých rodičů, kteří po celou dobu trpělivě snášeli veškeré strasti spojené s laděním předchozích pokusů. Tato skutečnost je pro mě velmi důležitá, neboť já sám nevlastním žádnou nemovitost. V případě pozitivní budoucnosti celého projektu budu rád, když se moje rodina stane prvním čestným zákazníkem firmy Kyvit s.r.o.

Motivace

Na závěr úvodu bych chtěl ještě v krátkosti promluvit o osobní motivaci pro celou tuto práci. Jako nejjednodušší a nejpragmatičtější důvod by zajisté byla samotná potřeba tématu pro diplomovou práci. Nicméně, toto téma mi bylo blízké ve více ohledech, a tudíž i moje motivace se skládá z více faktorů. Jak jsem již zmínil, dlouhodobě mě zajímá a baví práce s vestavnými systémy, ať už se jedná o nejmenší mikrořadiče s 512 byty RAM, či větší desky s vyšším výkonem jako Raspberry Pi. Vždycky jsem si chtěl navrhnout a následně nechat vyrobit vlastní plošný spoj pro některou z mých aplikací. Dalším bodem

mého zájmu je plánovaný přechod od méně běžných sběrnic pro komunikaci, které jsem dosud měl tu možnost poznat, k něčemu, co je v praxi mnohem více rozšířené. Mám na mysli Ethernet, který je těsně napojen na pojem IoT, do kterého bych chtěl hlouběji proniknout. Této změny bych také rád využil v našem domě, neboť realizace dosavadní kabeláže je poněkud „nestandardní“, a proto by si jistě zasloužila menší obnovu a vylepšení. A na konec asi nejdůležitější důvod, v projektu a jeho budoucím vývoji vidím velký potenciál a byla by škoda nepokusit se jej rozvinout a využít v maximální možné míře. Z toho důvodu je celá práce koncipovaná tak, aby jí dále mohla převzít má začínající firma Kyvit s.r.o., která by se měla zasadit o pokračování této myšlenky v rámci komerční praxe.

Cíle práce

Tato práce si dává za cíl navázat, rozšířit a případně doplnit původní obhájenou bakalářskou práci. Ta se zabývala návrhem a implementací chytrého domu za použití vlastního protokolu a webovým rozhraním.

Rozšíření by mělo zahrnovat minimálně následující klíčové body:

- Provedení analýzy možností vylepšení chytrého domu. Zvláštní péče by měla být věnována využití IoT (a v jeho kontextu protokolu MQTT).
- Výměna hlavního protokolu chytrého domu, přechod z RS-485 na internetový protokol (IPv4 resp. IPv6, dle analýzy).
- Aby bylo možné dosáhnout předchozího bodu, bude potřeba navrhnout a realizovat vlastní řídící jednotku vlastní konstrukce pro řízení chytrého domu.
- Je třeba vylepšit (zjednodušit) vzdálené ovládání periférií – aby nebylo nutné aktualizovat firmware pro jiný druh zařízení na stejné sběrnici (přenesení ovladačů ze zařízení do serveru).
- Systém by měl dbát na jednoduchost instalace a následné údržby.

Mezi primární cíle této práce patří zachování stávající funkcionality, možnost přidávání dalších funkcí, další vylepšování efektivity, a to vše při zachování uživatelské přívětivosti. Téměř všechny základní koncepty z bakalářské práce zůstávají stejné, tedy ovládání by mělo být dostupné skrze webové rozhraní. Velmi významnou změnou oproti bakalářské práci je fakt, že již nebude nejpodstatnějším kritériem minimalizace ceny, především ne na úkor zvýšení kvality.

Poslední cíl je změřen na možnou budoucí rozšiřitelnost celého systému. Z toho důvodu je třeba vytvořit také popis topologie navrženého systému z pohledu serveru pro zajištění dostatečné škálovatelnosti.

Kapitola 1

Teoretická část a analýza problému chytrých domů

Nejprve nastíním obsah následující kapitoly, pojmy použité v rámci tohoto úvodu budou vysvětleny později.

První část této kapitoly je věnována základním informacím o původní bakalářské práci [1]. Jsou zde shrnuta základní fakta a dále zde rozebírám a popisuji různé možnosti vylepšení původního systému.

Druhá část kapitoly pojednává o internetu (z technického hlediska). Nejprve krátce popisuji co to je IoT (*Internet of Things*). Následuje popis vrstev na kterých je postaven moderní internet (od podsekce Ethernet (1.4.1) až po podsekci TLS (1.4.4). Hned po popisu jednotlivých vrstev internetu je popsán protokol MQTT (velice hojně využívaný v IoT). Následuje debata ohledně výběru verze internetového protokolu – zda vybrat starší IPv4, či novější IPv6. Dále jsou popsány způsoby napájení po síti (PoE – *Power over Ethernet*) a zapojení vodičů v *ethernetové* kabeláži.

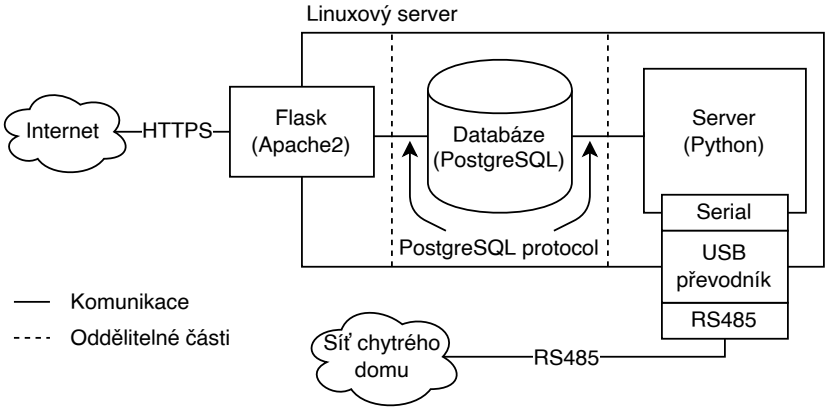
V sekci 1.8 jsou probrány v současné době dostupné návrhové programy pro navrhování plošných spojů. Dále je popsán mikrořadič, který bude při návrhu použit, včetně popisu rozvedeného hodinového signálu. Následuje popis síťových řadičů – primárně čipu W5500.

Kapitolu uzavírá popis ostatních softwarových nástrojů, které lze použít pro tvorbu backendu a frontendu. V posledním tématu kapitoly je věnováno pár slov k bezpečnosti[2] hardwarových jednotek chytrého domu, jejich přístupu k serveru přes internet a zabezpečení serverů samotných.

[2]Ve smyslu anglického slova *security*

1.1 Popis systému z předchozí práce

Ve své bakalářské práci [1] jsem se zabýval analýzou možností jak navrhnout chytrý dům. Následoval výběr jednoho z návrhů a jeho implementace.



Obrázek 1.1: Návrhové schéma původního systému – návrh používá pro agregaci jednotlivých softwarových komponent PostgreSQL databázi, části oddělené šrafováním lze provozovat na separátních linuxových serverech – převzato z bakalářské práce [1]

Jednotlivé ovládací jednotky byly postaveny na platformě AVR (procesory ATmega328 [2] od společnosti Atmel[3]). Systém jsem navrhnl tak, aby umožňoval přístup k informacím a ovládání domu přes internet. Nicméně jednotky samotné přímý přístup na internet neměly. Síť senzorů byla postavena na sběrnici RS-485 [3]. Webové rozhraní bylo implementováno ve Flasku [4]. Rozhraní je propojeno s chytrým domem databází PostgreSQL [5]. Práce se dále věnovala detailnímu popisu tvorby bootloaderu pro procesor AVR a nastavení serveru pro chytrý dům na linuxové distribuci Debian [6].

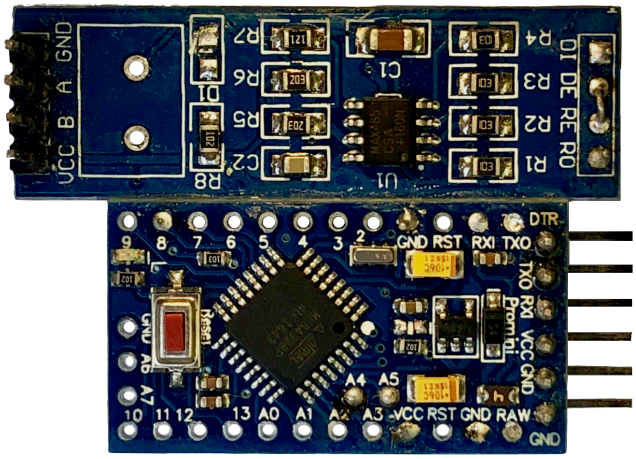
1.2 Analýza možností vylepšení původního systému

Během dvouletého testování původní verze chytrého domu vznikla spousta nových požadavků ze strany majitele či uživatelů domu. Jedním z hlavních problémů staré verze byl nedostatek možností pro vizualizaci dat. I když byla zaznamenávána všechna měření[4], tak uživatel měl přístup (alespoň z webového rozhraní) pouze k posledním 24 hodinám, a to pouze u senzorů zobrazených na hlavní stránce.

[3]Firma Atmel byla v roce 2016 koupena (a následně i integrována) společností Microchip Technology

[4]Systém snímal všechny senzory každých 15-30 sekund

Další nedokonalost spočívala ve „vlastním řešení“ zapojení jednotek, jinými slovy v absenci konektorů – všechny spoje byly letovány ručně, což na jednu stranu zvýšilo spolehlivost spoje, ale na druhou stranu to bohužel znesnadnilo následnou instalaci.



Obrázek 1.2: Deska používaná v první verzi – slouží k získávání dat ze senzorů a následné odesílání přes sběrnici RS-485 do serveru – z důvodu úspory peněz byly v první verzi byly použity separátní moduly místo vlastního plošného spoje – nahoře lze vidět převodník RS-485 na UART a dole desku Arduino mini – převzato z bakalářské práce [1]

Původní návrh počítal s jedním serverem (kompletní vlastní instancí pro každý chytrý dům) pro každého uživatele. Nespornou výhodou toho řešení bylo, že chytrý dům byl schopný fungovat po lokální síti i bez přístupu k internetu. Nevýhoda tohoto přístupu začne vyvstávat, pokud klient chce používat systém na různých místech[5] a musí kompletní server pořídit vícekrát – je potřeba vcelku výkonný stroj, aby zvládl obsluhovat klienta tak, aby byla aplikace ještě dostatečně responzivní (například vykreslování grafů na pomalém stroji trvá velmi dlouhou dobu).

Z pohledu uživatelského komfortu a především možných servisních zásahů nabízí původní řešení spoustu možností k vylepšení. Například zjednodušení instalace systému je velkou výzvou. Tedy v ideálním případě jen zapojit dva až tři konektory (podle množství použitých senzorů) a to by mělo být vše (samozřejmě následně registrovat novou jednotku, aby server věděl, ke kterému klientovi patří). Pokud by tedy například systém přecházel ze sběrnice RS-485 na ethernet, tak by se nabízelo využití PoE jako systému pro napájení a tím by se počet potřebných konektorů mohl ještě snížit.

[5]Například v bytu a zároveň na chatě

Jako další vylepšení navrhuji zvýšení tolerance u jednotek vůči zkratům (dalo by se uvažovat i o voděodolnosti). Toho by šlo docílit například pomocí kombinace nástřiku plochy plošného spoje a ochranného obalu. Obal by zároveň snížil šanci na náhodný zkrat, který by v závislosti na provedení PoE v připojeném switchi mohl klidně vyřadit všechny jednotky.

1.3 Definování pojmu – Internet of Things (IoT)

V moderní době, jak postupem času dochází k rozvoji technologií, je připojení různých „věcí“ do internetu stále levnější a snadnější. Jak u elektroniky pokračuje miniaturizace a postupně se snižuje spotřeba, tak je možné umisťovat postupně menší a menší mikropočítače na čím dál specifičtější místa. Například v poslední době jsou značně oblíbené automatické žaluzie, které se roztahují a zatahují podle venkovních podmínek, případně chytré televize, ledničky a v posledních pár letech se objevuje podobný trend například i u aut (kdy jsou automobilky schopné – minimálně elektrické – závady detekovat na dálku).

Ale pokud dovedeme tuto myšlenku dále, tak si uvědomíme, že i většina z nás je zapojena do sítě IoT. Během posledních několika let lze pozorovat postupný rozvoj přístupu na internet v mobilních telefonech (mobilní datová síť) – i tato běžná zařízení vlastně spadají do kategorie Internetu věcí (téměř každý telefon má dnes minimálně GPS a akcelerometr).

V této práci bych chtěl IoT využít pro standardizovaný přístup k jednotlivým jednotkám, hlavní výhodou je univerzálnost a možnost v jednotkách využít služby dostupné na internetu.

1.4 Principy komunikace po internetu

Pokud se chceme bavit o IoT, je dobré vysvětlit základní koncepty z technické stránky fungování internetu. Popíši jedno z nejznámějších pojetí internetu – rozdělení do vrstev v TCP/IP modelu[6]. Velká výhoda tohoto modelu je „rozdělení do vrstev“, což zvyšuje přehlednost a snižuje implementační náročnost. Pokud například budu chtít místo čtvrté verze Internetového protokolu použít verzi šestou, stačí upravit jen tuto jednu vrstvu a implementace ostatních vrstev zůstává stejná.

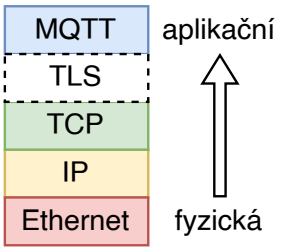
Na obrázku 1.3 lze vidět[7] uspořádání vrstev při posílání MQTT[8] packetu.

[6] Nejedná se o ISO/OSI model, ale o rodinu protokolů použitých při běžné TCP komunikaci

[7] Barvy jsou konzistentní s obrázkem 1.4

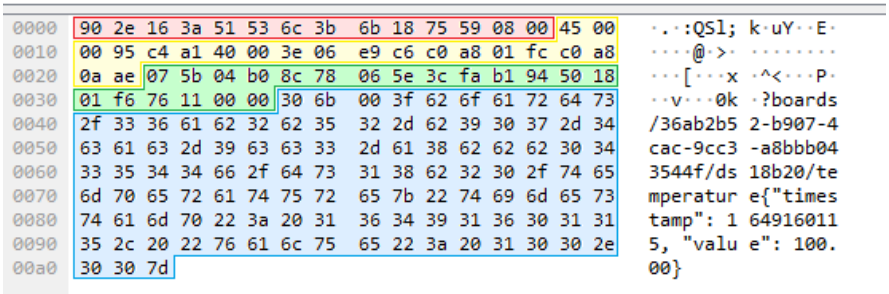
[8] Protokol MQTT bude podrobněji rozebrán v sekci 1.4.5

Obrázek výstižně reprezentuje rozšiřitelnost vrstevnatého modelu – například šrafováním vyznačená vrstva (TLS) může/nemusí být přítomna a podle toho je/není spojení pro MQTT na vyšší vrstvě šifrované. Ostatní vrstvy zůstávají beze změny.



Obrázek 1.3: Vrstvy při MQTT komunikaci

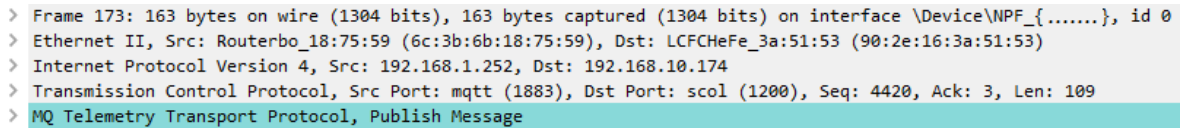
Pokud přejdeme od modelu ke konkrétním datům, tak na obrázku 1.4 můžeme pozorovat „surová data“. Vlevo reprezentovány číselně jako jednotlivé byty a vpravo jejich ASCII reprezentaci. V příkladu na obrázku není TLS vrstva – pokud by byla přítomna, tak by již nebylo možné pouhým pohledem na data (v pravé části) zjistit jejich obsah.



Obrázek 1.4: Surový MQTT packet se všemi vrstvami – úplně vlevo (šedivé pozadí) jsou označeny adresy jednotlivých řádků, uprostřed jsou barevně označena data specifická pro konkrétní protokoly (spolu s jejich reprezentací v šestnáctkové soustavě) a vpravo jsou data reprezentována v ASCII

Pro vizualizaci dat používám nástroj Wireshark [7], který je schopen zachytávat surová data a automaticky je interpretovat. Tuto skutečnost lze pozorovat na obrázku 1.5. Nejsou zobrazena přímo data (jako na obrázku 1.4), ale lze vidět dekódované „meta“ informace, jako je například MAC adresa zařízení na nejnižší vrstvě, IP adresy na vrstvě vyšší a ostatní dekódované informace.

V několika dalších sekcích se budu věnovat každé vrstvě z obrázků 1.3 a 1.4 podrobněji. U každé vrstvy platí, že pokud bude zmíněna hlavička, tak jsou myšlena metadata – ta předchází datům.



Obrázek 1.5: MQTT packet - vrstvy dekódované

1.4.1 Ethernet

Začněme tou nejnižší vrstvou a to jest ethernet. Ten u rodiny TCP/IP protokolů nařizuje mimo jiné i podobu vrstvy fyzické[9] – tedy specifikuje „jak jsou zapojené vodiče a co na nich běhá“. Dále ethernet definuje komunikaci na linkové vrstvě – ta má zajistit komunikaci mezi zařízeními v lokálním segmentu sítě.

Aby spolu zařízení mohla komunikovat, je potřeba, aby byla nějak adresovatelná. V případě Ethernetu k tomuto účelu slouží MAC adresy (jindy také nazývané jako fyzické). Ta by měla jednoznačně identifikovat zařízení v lokálním segmentu sítě – (naštěstí) nemusí být unikátní v celém internetu. Zařízení používají ke zjištění MAC adresy protistrany protokol ARP. Aby bylo možné přes ARP pomocí IP adresy (která je zmíněna v dalším textu) získat adresu fyzickou, využívá se tzv. linkového broadcastu – odesílatel nastaví jako příjemce fyzickou adresu `ff:ff:ff:ff:ff:ff`, na které poslouchají všechna zařízení v lokálním segmentu.

Mezi zařízeními dochází ke komunikaci za pomoci „rámců“ – ty obsahují informace jako například: odkud komunikace jde, kam má dojít, délku rámce, kontrolní součet a další.

1.4.2 Internet Protocol (IP)

Internetový Protokol se nachází o jednu vrstvu výše než Ethernet (v ISO/OSI modelu tedy stoupáme na vrstvu síťovou). Díky tomuto protokolu již lze komunikovat v rámci celého internetu (proto to trefné jméno).

Podobě jako u nižší vrstvy hlavička obsahuje informace odkud a kam je potřeba data přenést, dále obsahuje délku dat, protokol, kontrolní součet a několik dalších polí. Co se týká adres, tak již nemluvíme o adresách fyzických, nýbrž o adresách IP. Mezi další důležité pole, které hlavička obsahuje, se řadí verze použitého protokolu (důležité pro rozlišení IPv4 a jeho nástupce IPv6 – v této sekci ovšem budu dále mluvit jen o IPv4).

IP adresy jsou tvořené čtyřmi byty. Většinou jsou reprezentované jako 4 čísla

[9] Tak jak je reprezentována v ISO/OSI modelu

v desítkové soustavě oddělená tečkami. Příklad typické adresy: `192.168.1.1`. Síťování na této vrstvě už je složitější, zařízení musí vědět jak je daná „podsíť“ nastavená – k tomu potřebuje údaje jako adresu sítě, délku prefixu a adresu výchozí brány. Ukázka vymyšlené sítě:

- adresa sítě: `192.168.1.0`
- maska sítě (délka prefixu): `255.255.255.0` (24)
- adresa výchozí brány: `192.168.1.1`

V síti z předchozího příkladu pak lze také vysílat pro všechna zařízení pomocí broadcast adresy `192.168.1.255` – je to obdoba broadcast MAC adresy, ovšem vysílání nebude dostupné pouze zařízením v lokální segmentu, ale zařízením v lokální síti – LAN (síť může být složena z více segmentů).

K automatickému nastavení parametrů sítě je většinou používán protokol DHCP (*Dynamic Host Configuration Protocol*).

1.4.3 Transmission Control Protocol (TCP)

Internetový Protokol umožňuje „hloupě“ přenášet surová data. Nicméně, pro účely efektivní komunikace by bylo vhodnější, kdyby přenášení bylo sofistikovanější. Například by bylo vhodné, aby se data ztracená cestou (například v důsledku krátkého výpadku služeb), dokázala nakonec dostat ke svému cíli. Další dobrou funkčností by bylo, aby data chodila po pořadě, tzn. tak jak byla odeslána. Pokud by se surové IP packety cestou prohodily, není možné určit, který packet má být první a který druhý a příjemce si nemůže být jistý, zda přijal data správně.

O tyto výhodné funkce se v případě internetu, ve valné většině případů, stará právě protokol TCP. Podle ISO/OSI modelu leží na transportní vrstvě. V hlavičce se nachází informace o zdrojovém a cílovém portu (to pomáhá rozlišit jednotlivé aplikace nad TCP postavené), dále pořadové a potvrzovací číslo (tato pole se starají o uspořádání a nové odeslání nedoručených dat), dalším polem je kontrolní součet. V hlavičce se nachází ještě několik méně zajímavých polí.

Při komunikaci je nejprve potřeba otevřít spojení. Aby server i klient měli přehled o komunikaci, tak dochází k několikanásobnému potvrzování. Pak lze spojení využívat, kdy se protokol stará o různé eventuality, jako právě onen výpadek komunikace. Nakonec je potřeba spojení zavřít, kdy opět dochází k několikanásobnému ověření.

TCP se používá například pro HTTP/HTTPS komunikaci. Na těchto protokolech funguje velká část internetu (hlavně z uživatelského hlediska). Nyní by

bylo vhodné podívat se na cenu, kterou je za toto pohodlí nutno zaplatit (myšleno v objemu přenášených dat). Každá vrstva přidá pár bytů navíc. Přesněji to mohu určit součtem bytů jednotlivých vrstev:

- **Ethernet** – 14 bytů
- **IP** – 20-60 bytů[10]
- **TCP** – 20 bytů
- **uživatelská data** – kolik odešleme

A není v tom započítáno množství samostatných packetů, které je třeba na otevření, udržení a následné uzavření TCP spojení. V součtu nám jen hlavičky jednotlivých protokolů dají 54 bytů. Jak je vidět, tak využití TCP není úplně nejpřívětivější k paměti.

Na stejné úrovni jako je TCP existuje běžně užívaný protokol UDP. Ten slouží k posílání dat bez zajišťování „pěkných“ vlastností, které poskytuje TCP. Výhodou je nižší náročnost pro udržení spojení.

1.4.4 Transport Layer Security (TLS)

Tento protokol chci zmínit jen krátce. TLS je zkratka pro *Transport Layer Security* a vyvinulo se jako nástupce SSL (*Secure Sockets Layer*).

Protokol sloužící k zabezpečení komunikace – poskytuje důvěrnost, integritu a autentičnost dat (pomocí certifikátů).

V dnešní době se používá opravdu hojně od zabezpečení emailů až po HTTPS (na kterém běží například internetové bankovnictví). Protokol umožňuje serveru i klientovi podporovat část šifer a pokusí se spolu domluvit na nějakém kompromisu pro společnou komunikaci.

Nyní, když máme popsané všechny „nižší“ protokoly, je čas podívat se na jeden v kontextu IoT velice často zmiňovaný protokol MQTT.

1.4.5 MQTT protokol a jeho využití v IoT

Jedná se nenáročný protokol, který slouží k výměně zpráv mezi klienty skrz centrální bod, nazývaný broker. Tento protokol je definován standardem [8], který specifikuje všechny detaily. Pokud nahlédneme pod kapotu (do standardu), tak zjistíme, že komunikace mezi serverem a klientem probíhá trochu složitěji – různé systémové zprávy atp. Naštěstí toto je před programátory, kteří využívají jednu z mnoha existujících knihoven, kompletně skryto. Implementace pomocí knihovny je jednoduchá. MQTT definuje hierarchická témata

[10] Ve výpočtech budu používat nejběžnější případ (tedy 20 bytů)

(v angličtině *topics*) a klient, který chce „odebírat“ některé téma, se přihlásí k odběru (*subscribe*) a následně jiný[11] klient zveřejní (publish) do tématu data. Ta se pak podle nastavení QoS (*Quality of Service*), které klient specifikuje při odesílání zprávy, doručí přihlášeným (*subscribed*) klientům.

Jako téma (*topic*) se většinou používá schéma, které pro absenci lepšího pojmu nazvu „souborový systém“. Jednotlivá témata jsou „rozdělena do složek“, ty jsou od sebe odděleny lomítky a v poslední úrovni jsou „uložena data“. Například ukázka tématu: `kuchyně/topení/teplota`.

Pro témata existuje několik doporučení – nedefinuje je přímo standard, ale je rozhodně vhodné je dodržovat. Například nepoužívat (na rozdíl od souborového systému) lomítko na začátku (absolutní cesta), přidává to „zbytečnou prázdnou složku“ na začátek. Je vhodné nepoužívat mezery a ideálně se držet v ASCII. Dále je více než vhodné na začátku tématu nepoužívat znak dolaru (`$`) – to je rezervováno pro systémová témata a mohlo by to způsobovat problémy.

Kromě dolaru jsou ještě rezervované znaky plus (`+`) a hashtag (`#`), ty umožňují klientům provést subscribe k více tématům zároveň. Znak `+` nahrazuje jednu úroveň – tedy například u `kuchyně/+/teplota` dojde k nahrazení jakýmkoli existujícím slovem. Znak `#` nahradí více úrovní – například: `kuchyně/#` znamená, že se klient přihlašuje k úplně všem podtématům, které se nachází za úrovní `kuchyně`.

MQTT explicitně nevyžaduje u dat žádný specifický formát. Ovšem v praxi nejčastěji figurují standardní formáty jako například JSON a XML. Samozřejmě pro vestavné systémy je vhodnější použít úspornější formáty, jako jsou například BSON [9], nebo MessagePack[12]. Bohužel při použití méně známého formátu člověk přichází o možnosti nabízené běžně používanými nástroji, které se zaměřují na nejklasičtější „use-case“. Příkladem takového nástroje je *MQTT Explorer*, který umožňuje reprezentaci dat v surové podobě, ve formátu JSON nebo XML.

1.5 Porovnání rozdílů mezi verzemi internetového protokolu (IP)

V následujících pár odstavcích je věnován primárně důraz na rozdíly mezi verzí 4 a verzí 6[13].

[11] Může se přihlásit případně i on sám

[12] Velice úsporný, pěkně strukturovaný formát [10]

[13] popis IP protokolu viz sekce 1.4.2

Internetový protokol verze 6 je pomalu nastupující náhrada za zastaralý IPv4. Jedním z jeho hlavních přínosů je zvětšení počtu[14] adres z 2^{32} na 2^{128}. Ve staré verzi je v posledních letech nedostatek veřejných adres. Tento problém lze naštěstí snadno[15] vyřešit přechodem na IPv6. To je pro potřeby této práce, kde se snažím šetřit každý byte místa, ale zároveň nevýhoda protože velikost hlavičky v IPv6 je fixních 40 bytů, což může (neplatí úplně vždy) být více než u IPv4 (kde je velikost hlavičky určena podle obsahu v rozmezí 20-40 bytů).

Dále protokol IPv6 může umožňovat připojení klienta přímo na jiného klienta (u IPv4 problém s NAT – překlad adres).

Nevýhoda IPv6 je bohužel jeho (zatím) nízká rozšířenost. To by znevýhodňovalo většinu lidí bez možnosti připojit se do takové sítě. Existují možnosti, jak lze toto omezení obejít, bohužel to přidává požadavky na schopnosti routeru a dělá to přesný opak „jednoduché instalace“.

1.6 Napájení přes síťový kabel – Power over Ethernet (PoE)

Běžně se v IT setkáváme s malými vestavnými systémy, které vyžadují napájení a zároveň internet (prakticky všechny) – primárně myslím špatně dostupná zařízení jako například IP kamery, síťové vysílače na střechách a další. Vedení pouze jednoho kabelu zajišťuje jednodušší instalaci a vyšší spolehlivost. Proto bylo vymyšleno a později standardizováno PoE.

Existuje několik druhů. Jako první bych chtěl zmínit tzv. pasivní PoE. Přes „nepoužívané“ vodiče[16] je přivedeno napájení a zařízení má k dispozici tolik elektrické energie, kolik dokáže vyvinout kombinace zdroje a vedení. Pasivní PoE většinou operuje na několika specifických napěťových úrovních (podle druhu) – typicky 12, 24 nebo 48 voltů. Existují switche, které integrují pasivní PoE napájení, ovšem pro chytré domy, kde se očekává velké množství jednotek je výhodnější použít levný switch a do racku přidat převodník na PoE.

Druhou skupinou jsou zařízení s aktivním PoE. Ty již musí být standardizované, protože dochází ke komunikaci mezi napájeným zařízením a zdrojem. Výhodou, ale zároveň i nevýhodou, je právě ona aktivní komunikace. Její výhodou je možnost vykomunikovat si dostatek energie, nevýhodou je zvýšené množství potřebných součástek a to znamená dražší hardware na obou stranách.

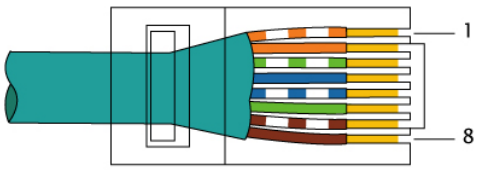
[14] Počty jsou přibližné – musí se odečíst rezervované rozsahy atd.
[15] Snadno pro uživatele, nikoli pro internetové poskytovatele
[16] Podle toho, jaký je použitý standard pro komunikaci, může být několik vodičů volných

Z logiky věci má PoE několik nevýhod – jako vodiče se používají jednotlivé žíly v síťovém kabelu, které mají ovšem relativně malé průměry a kabely mohou být stovky metrů dlouhé – to omezí maximální možný proud, ovšem v aplikaci pro chytré domy výhody značně převyšují nevýhody.

1.7 Zapojení RJ45 podle ANSI/TIA-568 a druhy kabeláže

V dnešní době nejběžnější koncovka pro internetové kabely je RJ45. Pokud má nějaká domácnost internet, tak téměř jistě bude používat RJ45 – pro *uplink* může být použitý občas i koaxiální kabel, ale propojení mezi routerem a počítači bude RJ45 zcela jistě.

„Commercial Building Telecommunications Cabling Standard“ (standard pro zapojení kabelů v komerčních budovách) specifikuje, jak zapojovat kabely u klasických RJ45 zásuvek. Jde o správné poskládání barev jednotlivých žil v kabelu. Na obrázku 1.6 se nachází ukázka zapojení ethernetového kabelu podle standardu T5689B. Díky tomuto standardu je dnes v zapojení menší zmatek. Například pokud budu mít dva lidi, co zapojují stejný kabel, a vím, že ve firmě používáme jen T568B, tak mohou oba připravit kabel na svojí straně, aniž by se museli domlouvat.



Obrázek 1.6: Zapojení ethernetového kabelu podle TIA-568B

Posledním důležitým prvkem je pak vlastní výběr druhu kabelu. Je třeba používat (alespoň v drtivé většině případů[17]) 8 samostatných vodičů – ty jsou spojeny do čtyř párů. Každý síťový kabel by měl mít svou kategorii (standard který splňuje) viz tabulka 1.1. Kategorie kabelu specifikuje maximální frekvenci – a tím i maximální rychlost kterou je po kabelu možné komunikovat.

[17] Při pomalejší komunikaci ethernet může specifikovat menší množství potřebných vodičů, čehož výrobci u některých levných produktů využívají a kabel z důvodu úspory ony nepotřebné vodiče neobsahuje

[18] Maximální na 100 metrů

kategorie	přenosová rychlost[18]	frekvence
cat. 5	100 Mbps	100 MHz
cat. 5e	1000 Mbps	100 MHz
cat. 6	1000 Mbps	250 MHz
cat. 6a	10 Gbps	500 MHz
cat. 7	10 Gbps	600 MHz

Tabulka 1.1: Kategorie síťových kabelů

1.8 Návrhový software pro vývoj plošných spojů

Součástí zadání je i tvorba vlastního plošného spoje. K tomu bude třeba specializovaný návrhový software. Na trhu existuje velké množství nástrojů, ze kterých lze vybírat (Altium Designer[19], Autodesk EAGLE[20], Fusion 360 a spoustu dalších) – tyto nástroje, i když hojně používány, jsou cenově bohužel značně nedostupné. Proto jsem začal hledat jinou alternativu, až jsem narazil na KiCad, který je dokonce zdarma.

KiCad [11] je software s otevřeným zdrojovým kódem určený k návrhu plošných spojů. Na trhu se vyskytuje již dlouhou dobu. První (neveřejné) vydání programu bylo už v roce 1992, vydání pro veřejnost bylo o několik let později. Z uživatelského hlediska s ním bylo za dobu jeho existence značné množství problémů, hlavními byly nečekaná ukončení a nižší uživatelská přívětivost. Ovšem během několika posledních let byl hodně vylepšen a poslední verze (6.0) spoustu problémů řeší. Nyní je program více než použitelný.

Software podporuje návrh vícevrstvých plošných spojů od návrhu schémat, až po export Gerber souborů pro výrobu. V poslední verzi byla přidána podpora zásuvných modulů (plugin manažer) a okamžitě se objevila spousta užitečných pluginů – například export souborů pro konkrétní výrobce (Gerber, POS a BOM soupisky pro výrobu ve specifickém formátu), což práci na plošném spoji drasticky usnadňuje.

1.9 Volba nového mikrokontroleru

I když se mi čip ATmega328P v bakalářské práci osvědčil jako velmi spolehlivý, ze 14 zapojených jednotek ani s jednou nebyl během tří let problém (co se týká funkcí mikrořadiče), na požadavky nové verze systému bohužel již svým výkonem nestačí. ATmega328P má pouze 2 kB RAM. Komunikace přes internet je paměťově velmi náročná (viz sekce 1.4). Stačí se podívat na množství odesílaného „balastu“ a lze snadno tvrdit, že na to mikrořadič s takto malou

[19] S cenou začínající na 295 eurech měsíčně – cena k dubnu 2022
[20] S cenou 372 dolarů za rok – duben 2022

RAM není vhodný – pro referenci čip W5500 má pro 8 socketů společnou paměť (RX/TX) o velikosti 32 kB.

Jakýkoli druh vysílání z původního systému by se vešel celý do samotných hlaviček jednotlivých protokolů.

Po mých předchozích osobních zkušenostech s mikrořadičem STM32F103C8T6 jsem se rozhodl ho vybrat. Oproti původnímu AVR má desetinásobek RAM (20 kB), je daleko flexibilnější a lze ho taktovat více než čtyřikrát rychleji.

1.9.1 Popis mikrořadiče STM32F103C8T6

Tyto mikrořadiče získaly svou velkou oblíbenost zejména díky deskám *BluePill* – velmi levné a softwarově kompatibilní s Arduinem. Vyrábí je společnost *STMicroelectronics*. Výkonem by se daly zařadit do střední třídy mezi nižší AVR (ATmega328) a vyšší ESP32 (dvoujádrové procesory s taktem mezi 160 a 240 MHz).

Jsou to 32 bitové mikrořadiče založené na jádře `Arm 32bit Cortex-M3` [12], podporují násobení v jednom cyklu a hardwarové dělení. Součástí pouzdra je i flash paměť, kde je umístěn program.

Pod kapotou ukrývají opravdu velké množství periferií. Zkrácený seznam:

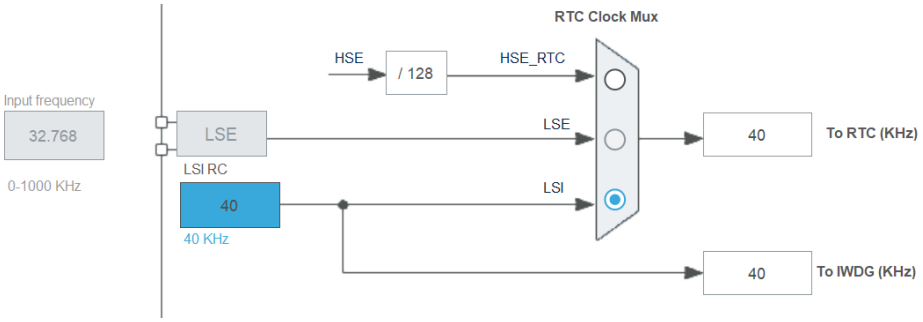
- dvakrát dvanáctibitový analogový převodník (ADC)
- velké množství GPIO pinů – z nichž je většina „5 V compatible“
- nástroje pro ladění – SWD (*Serial Wire Debug*) a JTAG
- sedm různých časovačů
  - třikrát obecný
  - dvakrát WatchDog (různé druhy)
  - PWM ovládání motorů
  - udržování času (SysTick 24-bit downcounter)
- velké množství komunikačních rozhraní: I^2C, USART, SPI, CAN, USB

Jedna z výhod, kterou poskytuje tento mikrořadič, je obrovská kontrola nad vlastním hardware. Na rozdíl od AVR se například dá přetaktovat, vypnout nepotřebné periferie, dynamicky měnit takt za běhu a spousta dalších zajímavostí. Díky tomu, pokud si na firmware dá vývojář opravdu záležet, lze vytvářet opravdu výjimečná řešení.

1.9.2 Zdroje hodinového signálu u vybraného mikrořadiče

V mikrořadičích STM32F103 lze vybírat různé zdroje hodin pro různé periferie (nastavení výběru hodin pro RTC a IWDG na obrázku 1.7):

- **LSI** – *Low-Speed Internal* – 40 kHz – je používaný jako zdroj hodinového signálu pro IWDG (*Independent WatchDog*) a může být zvolen dále i pro RTC (*Real-Time Clock*)
- **HSI** – *High-Speed Internal* – 8 MHz – integrovaný výchozí oscilátor
- **LSE** – *Low-Speed External* – 32 kHz – externí zdroj pro RTC
- **HSE** – *High-Speed External* – až 16 MHz – externí vysokorychlostní zdroj hodinového signálu



Obrázek 1.7: Schéma rozvodu nízko rychlostních hodin u STM32 – v IDE lze vybrat, který oscilátor používat pro které periferie

Po resetu mikrořadiče je jako zdroj hodin vždy používán HSI, po spuštění může uživatel nastavit parametry pro externí rezonátor a následně na něj přepnout. Při použití HSI bohužel nelze procesor taktovat na jeho maximální frekvenci 72 MHz – nejvíce lze dosáhnout 64 MHz. Tedy při použití externího krystalu je dosažen přibližně o deset procent vyšší výkon než s krystalem interním.

1.10 Hardwarové síťové řadiče

Na ukázku jsem si vybral dva čipy W5500 a W6100. Jedná se o čipy obsahující TCP/IP řadič[21]. Pokud použiji analogii, tak je to podobné, jako „síťová karta“ v PC, až na to, že nepoužívá PCI Express, ale SPI. Je k ní samozřejmě nutné přidat všechny potřebné součástky. Podporuje následující protokoly: TCP, UDP, IPv4, ICMP (ping), ARP a další. Vlastně je to taková „hardwarová ethernetová knihovna“. Umožňuje uživateli používat 8 nezávislých socketů v jednu chvíli – tento počet lze snížit a maximálně využít místo ve vestavěné paměti – hodí se například pro větší packety).

Podporuje ethernetové fyzické vrstvy: 10BaseT (10 Mbit/s, baseband, twisted pair, Manchester code) a 100BaseTX (100 Mbit/s, baseband, twisted pair, 8b/10b kódování bloků).

[21] Oba dva čipy jsou funkčně ekvivalentní, jen W5500 je pro IPv4 a W6100 pro IPv6 – dále budu mluvit jen o W5500 [13].

Čip vyžaduje 3,3 V napájení. Pomocí externích pinů lze vynutit konkrétní rychlosti Ethernetu (10/100 Mbit/s). V případě jejich nezapojení se použije automatické vyjednání rychlosti (anglicky: *auto-negotiation*). Lze použít v několika SPI režimech (variable/fixed length data mode), což umožňuje více klientů na jedné sběrnici, nebo zjednodušení zapojení. Na webu výrobce je několik ukázkových schémat [14], podle kterých lze navrhnout plošný spoj.

1.11 Nástroj FastAPI pro tvorbu backendu

Tento framework je designovaný pro snadný a rychlý vývoj API v programovacím jazyku Python. Autoři se snaží cílit především na snadnost programování a jeho výkonnost [15]. FastAPI používá zajímavý systém kontroly a vymáhání typů.

Obrovskou nevýhodou je absence řádné dokumentace u projektu. To činí jeho použití velmi nesnadné. Autoři se spoléhají na to, že z ukázkového kódu bude vývojářovi jasné, jak se má správně používat. Bohužel jsem při jeho použití nejednou narazil na části, kde by dokumentace velmi významně pomohla. Mimo jiné je její absence obrovským bezpečnostním rizikem, protože vývojářova představa toho, jak se má funkce chovat nemusí odpovídat realitě, což vytváří prostor pro dělání chyb ohrožujících bezpečnost kódu.

Další výhodou je systém generování automatické dokumentace.

1.12 Webový Framework Flask

Jedná se o sadu nástrojů (framework) pro vývoj a nasazení dynamických webových stránek [4]. Je napsaný v Pythonu. Celý koncept Flasku spočívá v jeho minimalismu. Na rozdíl od konkurenčních frameworků (například Django), je Flask spíše množina různých nástrojů (Jinja, Werkzeug a další), které Flask pouze spojuje do jednoho celku. Flask podporuje spíše vývojářovo vlastní řešení, než aby ho do něčeho nutil (například napojení na databázi).

1.13 Možnosti zabezpečení jednotlivých komponent systému

I když je toto téma velmi často zanedbávané (například v předchozí práci jsem mu nevěnoval příliš pozornosti), tak bych se rád v následujících pár podsekcích zaměřil na možnosti zabezpečení různých kritických komponent systému.

1.13.1 Hardwarová jednotka chytrého domu

Protože se práce zabývá návrhem jednotky – zařízením figurujícím v síti – je třeba analyzovat i možné otázky okolo jejího zabezpečení.

Nejprve nastíním co nejhoršího by útočník mohl se systémem dělat, pokud by získal absolutní kontrolu (platí pro šifrované i nešifrované spojení – díky absolutní kontrole). Pokud by například sledoval senzory, je schopen určit, zda někdo je, nebo není doma (vhodné, pokud je daná osoba nějak zajímavá, nebo pokud plánuje loupež). Chytrý dům aktuálně neimplementuje automatické odemykání dveří, tedy útočník nedokáže získat přímý přístup do domu (plánováno v dalších verzích – samozřejmě s vhodným zabezpečením).

Co chytrý dům ale již dělá, je ovládání různých zařízení. Aktuálně klimatizační jednotky, kamna a stará se také o ochranu rekuperační jednotky. V případě úspěšného napadení by měl útočník možnost ovládat jednotlivá zařízení. To sice není velice pravděpodobně likvidační (s výjimkou rekuperace), ale rozhodně to je nepříjemné a nežádoucí. Ohledně rekuperace, chytrý dům hlídá venkovní teplotu a v případě nízkých teplot přepne přívod vzduchu z venku na přívod zevnitř a tím zabrání zamrznutí tepelného výměníku a jeho následnému zničení.

V budoucnu by měl dům ovládat i tepelné čerpadlo a zde už by útočník dokázal napáchat škody za statisíce korun.

1.13.2 Zabezpečení síťové infrastruktury

Co tedy s tím? Nejjednodušším řešením by bylo kompletní oddělení všech jednotek do vlastní sítě (zabránění útočníkovi kompletně v přístupu). Toto „zabezpečení“ je ovšem vykoupeno tím, že si zákazník musí připlácet za další síťovou infrastrukturu. Levnějším řešením by bylo pořízení switche s podporou technologie VLAN (Virtual LAN) – toto řešení je velmi oblíbené. Jednotky chytrého domu zůstanou v jedné VLAN, další zařízení (například kamery) ve druhé a ostatní síťová zařízení ve třetí. To umožňuje použití jednoho „chytřejšího“ switche místo tří „hloupých“. Zařízení mezi jednotlivými VLAN na sebe nevidí a uživatel je spokojený, že nemusí kupovat více zařízení.

Dále by bylo možné vynutit konkrétní MAC adresy na konkrétních portech, to už ale ubírá na flexibilitě řešení a příliš bezpečnosti to nepřidá.

V dalším kroku při zabezpečení už musíme přejít k sofistikovanějším řešením. Hardwarovou jednotku by nemělo být možné kompromitovat (za použití běžných prostředků) i pokud má útočník fyzický přístup. Tedy zařízení by mělo být „zamknuté“, aby z něj nebylo jednoduché dostat firmware. Dále by bylo

vhodné, aby bylo odolné vůči odběrové analýze – lze například použít náhodné vkládání „dummy“ instrukcí atp.

Posledním krokem v zabezpečení je komunikace s vnějším světem. Jednotka bude jistě potřebovat přístup k internetu (serveru), proto IoT. Tato komunikace by v ideálním případě měla být kompletně zašifrována nějakou běžně používanou (osvědčenou) šifrou. Pokud zařízení šifrování neumožňuje – nemá hardwarovou podporu, nebo na to prostě jeho výkon nestačí – tak existuje slabší forma, kdy se o šifrování stará nějaké jiné zařízení na síti, například router, přes který tato komunikace stejně musí jít. To nám v kombinaci s VLAN dává celkem silné zabezpečení. Komunikace je nešifrovaná pouze v jedné VLAN (která je kompletně oddělená od skutečné LAN) a v moment kdy komunikace opouští svou VLAN, tak je automaticky zašifrována a odeslána na server. Asi nejznámější možností jak zajistit takové šifrované spojení je OpenVPN a nebo má méně známá osobní preference – WireGuard [16].

1.13.3 Požadavky na zabezpečení backendu

Na rozdíl od jednotek, servery zpracovávající požadavky musí být dostupné na internetu (mít veřejnou IP adresu). To znamená, že je třeba věnovat jejich zabezpečení daleko větší úsilí. Čistě prakticky by z internetu měly být přístupné jen tři porty. Port 80 pro HTTP kde připojujícímu server sdělí, že HTTP použít nelze a ať použije port 443. Na něm již server bude (zabezpečeně – TLS) odpovídat na HTTPS požadavky. A posledním portem je nějaký přístup do vnitřní sítě (pokud je potřeba) – vhodně zabezpečená VPN (šifrování a autentifikace). Pomocí této VPN lze server spravovat a přistupovat k ostatním potřebným z venku nepřístupným portům – jako například port 22 pro SSH nebo port 5432 pro přímý přístup do PostgreSQL databáze (pokud na serveru běží). Všechny požadavky na ostatní porty musí firewall[22] automaticky zahazovat. Občas je zajímavé pozorovat, o kolik pokusů k neoprávněnému přístupu (například na port 22) za den na serveru s veřejnou IP dojde.

Díky přístupu z předchozího odstavce omezíme možný vektor útoku pouze na 3 porty. Teoreticky potom zůstává ještě eventualita útoku přímo na jádro (například speciálně vytvořenými packety) se kterou bohužel nejde, kromě pravidelných aktualizací, příliš mnoho udělat. Naštěstí se závažné zranitelnosti linuxového jádra (konkrétně jeho síťové sekce) neobjevují zdaleka tak často. Většina útoků stejně není tak sofistikovaná a spočívá v tom, že robot narazí na otevřený port 22 (SSH) se jménem „root“ a běžně uhodnutelným heslem.

Pak už jde o software obsluhující dané otevřené porty. Na portech 80 a 443 poběží nějaká forma webového serveru (nejčastěji Apache, nebo Nginx), ten musí

[22] V linuxu například nftables

být samozřejmě správně nakonfigurován a udržován (aktualizace). Webový server bude předávat (například pomocí uWSGI, WSGI, …) požadavky procesu obsluhující požadavky – to může být nějaký webový framework v Pythonu například Django nebo Flask a v něm běží uživatelská aplikace – část o jejíž zabezpečení se musí postarat autor aplikací.

Uživatelská aplikace by měla dodržovat princip nejnižších oprávnění – tedy pokud musí být spouštěna pod uživatelem s vyššími oprávněními (root), tak se jich po vyřešení nejnutnějších částí (otevření socketu) vzdá. Systemd[23] používané ve většině dnešních linuxových distribucí nabízí celkem rozsáhlé možnosti dalšího zabezpečení – od omezení maximální dovolené paměti, přes spouštění služby pod náhodně generovaným uživatelem bez práv, až po omezení či dokonce kompletní schování systému souborů.

[23] Linuxový správce systému a služeb – moderní implementace *init systemu*

Kapitola 2

Návrh jednotlivých komponent systému

V úvodu této kapitoly věnuji prostor návrhu infrastruktury celého systému – kde a jak by mohla být umístěna zařízení v síti, atd. Nejrozsáhlejší část této kapitoly je věnována procesu navrhování plošného spoje.

V sekci o plošném spoji je rozebráno, s jakou spotřebou desky je třeba při navrhování zdroje počítat. Na to navazuje samotný návrh zdroje napájení. Další podsekce se věnují začleňování hlavního mikrořadiče na desku – to znamená, jak navrhnout zdroj externího hodinového signálu a jak na desku přidat podporu pro ladící nástroje. Dále je rozebrán návrh ohledně internetového čipu W5500 v kombinaci s konektorem RJ45.

Kapitolu uzavírá sekce o tom, jak vylepšit existující databázi z původní práce.

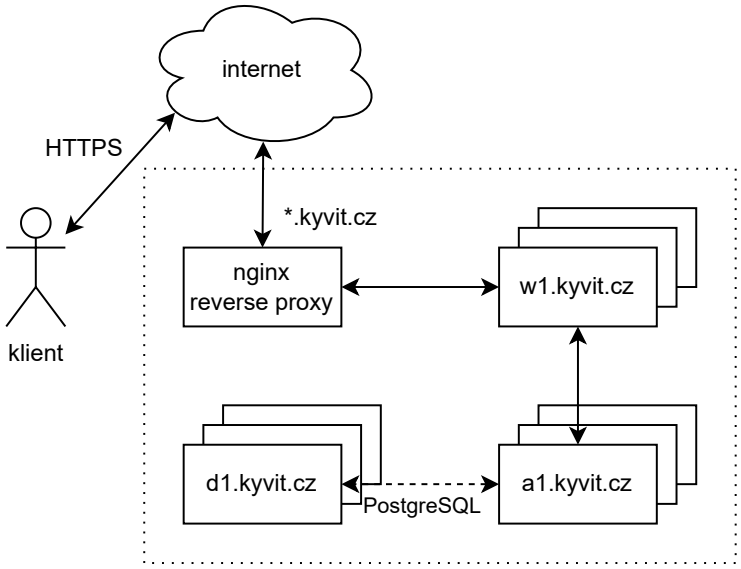
2.1 Celkový návrh infrastruktury systému – příklady použití

Nejprve je vhodné připravit si průběhy pro dva důležité scénáře, ze kterých následně bude vycházet vše ostatní. Prvním scénářem je komunikace uživatele (zákazníka) se serverem, druhým scénářem je komunikace mezi jednotkou chytrého domu a serverem.

2.1.1 Interakce uživatele s webovým rozhraním

Scénář komunikace uživatele a chytrého domu je vcelku přímočarý. Ukázka jedné z možností je na obrázku 2.1. Uživatel se připojuje svým telefon, počítačem nebo jiným zařízením pomocí internetu na vnější bránu sítě projektu.

Vnější brána má veřejnou IP adresu (na kterou odkazuje DNS) a na kterou se klient připojuje. Před, nebo přímo součástí tohoto hraničního zařízení by měla být nějaká forma firewallu. Na hraničním zařízení běží reverzní proxy – k tomu lze použít například webový server Nginx. Reverzní proxy přeposílá požadavky ostatním webovým[24] serverům ve vnitřní síti. Mimo jiné se tato reverzní proxy stará také o šifrování spojení s klientem. Webový server *w1* se pokusí obsloužit klienta. K tomu potřebuje přístup k API. To běží na stroji s označením *a1*. Stroj *a1* se následně napojí na databázový server *d1* a přes *a1* vrátí odpověď webovému serveru *w1*. Ten pošle data zpět pomocí reverzní proxy, která je zašifruje a vrátí zpět klientovi.



Obrázek 2.1: Připojení z webového klienta na server

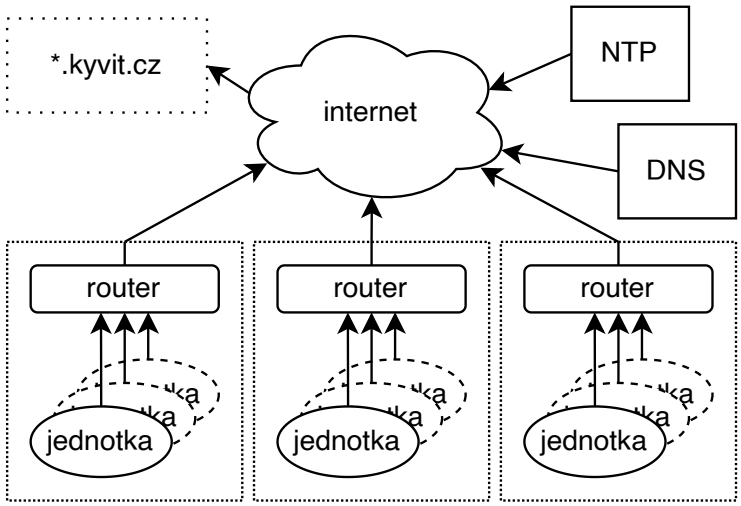
Výhodou přístupu podle předchozího odstavce je možnost rozdělení různých požadavků mezi více serverů (prostor pro vertikální škálování). V ukázce jsou použity 4 různé druhy serverů, ale v praxi lze jejich počet snížit třeba až na jeden.

2.1.2 Propojení jednotek se serverem v rámci systému

V případě druhého scénáře (z úvodu sekce) by komunikace jednotek se serverem mohla vypadat například jako na obrázku 2.2. Tento návrh poskytuje několik výhod. Router mezi jednotkou a serverem chytrého domu se může starat o šifrování (pokud by samotná jednotka neměla dostatek výkonu) a jednotlivé jednotky mohou využívat služeb volně dostupných na internetu, jako

[24] V obrázku 2.1 se jedná o stroj s označením *w1*

například DNS pro určení IP adresy serveru (vylepšení oproti statické adrese) nebo NTP server pro určení (a udržení) přesného času.



Obrázek 2.2: Připojení jednotky na server

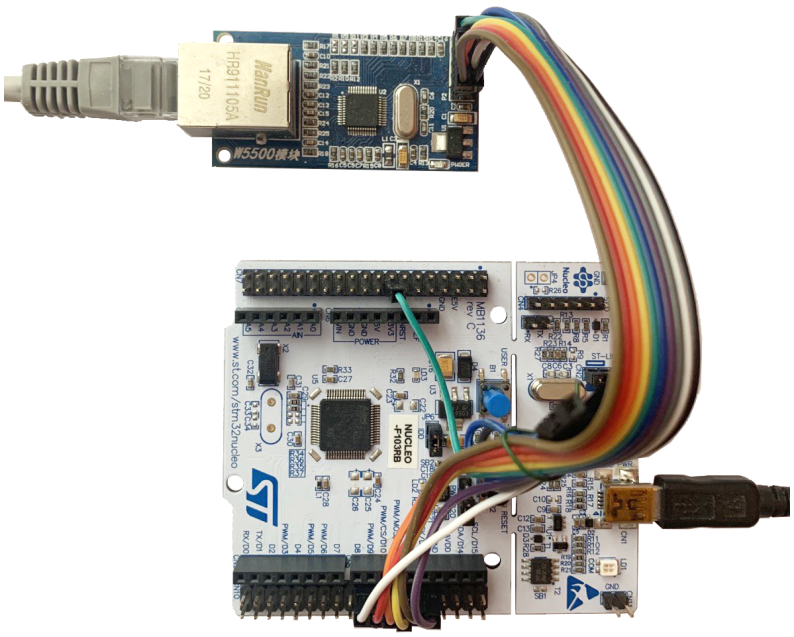
2.2 Úvod k návrhu plošného spoje

Při navrhování plošného spoje je možné využít vývojových desek, které STMicroelectronics vydává ke svým procesorům. Díky komunitě lidí kolem projektu Arduino vznikl modul s čipem W5500. Tyto dvě části mohu propojit a otestovat (alespoň částečně) jejich vzájemnou kompatibilitu (v tomto případě primárně softwarovou). Obě tyto části jsou zachycené po jejich testování na obrázku 2.3.

Abych si usnadnil vývoj, tak mohu rozdělit desku do několika logických částí. Ty mohou být navrhovány každá zvlášť a ke spojení dojde až v konečném návrhu. Těmito logickými celky jsou:

- hlavní mikrořadič (STM32) a součásti potřebné k jeho zprovoznění (kondenzátory, rezistory, krystal, a další)
- čip starající se o komunikaci (W5500), součástky k jeho zprovoznění a konektor RJ45
- zdroj napájení
- ostatní – GPIO konektory (vstup a výstup) a ostatní součástky nezařazené jinde

Každé z kategorií se budu věnovat v následujících podkapitolách.



Obrázek 2.3: Prototyp s vývojovou deskou NUCLEO-F103RB – W5500 modul propojený pomocí SPI k vývojové desce Nucleo

2.3 Zdroj napájení pro desku

Několik dalších podsekcí bude věnováno návrhu vlastního zdroje napájení pro jednotku chytrého domu.

2.3.1 Analýza maximálního zatížení napájení desky

Před návrhem zdroje pro desku je dobré vědět, kolik by mohla být maximální spotřeba celé desky. Pokud budeme počítat maximální možné zatížení, tak čip W5500 by v normálním režimu podle datasheetu [13] měl mít spotřebu okolo 132 mA (na 3.3 V), tedy 440 mW. Mikrořadič STM32 by neměl překročit 50.3 mA na 3.3 V, to je spotřeba 166 mW. Na desce se nachází 3 LED. Dvě jsou v konektoru a jedna zvlášť ovládána hlavním mikrořadičem. Pokud opět budu brát horní limit (skutečná spotřeba bude mnohem menší), tak všechny diody dohromady budou odebírat do 200 mW. Kromě výše jmenovaných se na desce nenachází žádné jiné výrazné spotřebiče. To znamená, že teoretická maximální (nereálná) spotřeba desky je 806 mW. Z toho plyne, že zdroj musí být schopen (na 3.3 V) dodat alespoň 245 mA.

V úvahu se musí vzít i případná spotřeba připojených periferií. Teplotní a jiné senzory mají typicky velmi nízkou spotřebu. Spotřebič s nejvyšší spotřebou elektrické energie, který by mohl být podporován je relé. To jsou spotřeby

v řádech stovek miliampér. Tedy design by měl být mířen na maximální proud mezi dvěma a třemi ampérami (na 3.3 V).

2.3.2 Návrh vlastního zdroje napájení

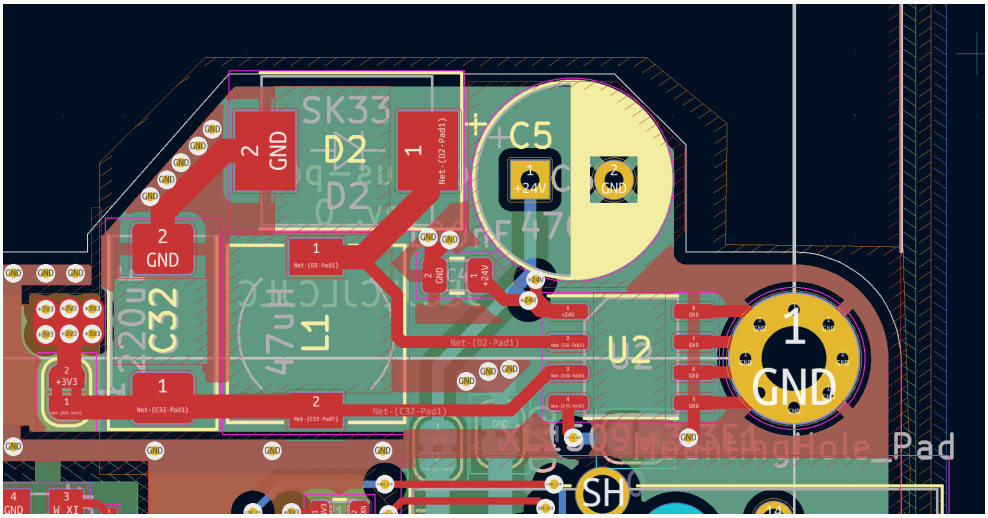
Pasivní PoE dodává do systému 24 V. Toto napětí je pro komponenty na desce příliš vysoké a je třeba ho snížit na přijatelnou úroveň. Například pro STM32F103C8T6 je přijatelné napětí v rozmezí 2.0-3.6 V. Typicky se používá napětí 3.3 V, které plně postačuje k napájení mikrořadiče i při nejvyšším možném zatížení (jádro procesoru na maximální frekvenci 72 MHz a zapnuté všechny periferie).

Čip pro zdroj jsem vybíral s ohledem na několik parametrů. Mnou vybraný výrobce plošných spojů u každé součástky udává, zda je „základní“ nebo „rozšířená“. U základních součástek není poplatek navíc za nastavení strojů. Podle tohoto parametru jsem filtroval různé zdroje – zůstalo deset různých druhů. Následně jsem zpřísnil kritéria na cenu – součástka musela být dostatečně levná (pod 0,40 $). To snížilo počet druhů na tři s tím, že dva z nich byly stejný čip, jen s jinými parametry. Kandidátem byl tedy čip `XL1509-ADJE1`, ale nakonec jsem se rozhodl – z důvodu jednoduššího návrhu desky – radši použít rozšířenou součástku `XL1509-3.3E1`.

Vybraný čip je vyráběn v pouzdře SOP-8 a podle datasheetu [17] zvládne zpracovat vstupní napětí v rozsahu 4,5-40 V. Při výstupním napětí 3,3 V by měl být schopen udržet maximální konstantní zatížení 2 A. Pro převedení napětí používá pevnou frekvenci 150 kHz. V datasheetu je ukázka schématu pro typické implementace, ze které lze vyjít a navrhnout vlastní schéma upravené pro mé potřeby. Drobnou nevýhodou je, že čip nezvládne o přibližně 10 V vyšší vstupní napětí. Bylo by potom možné podporovat větší množství routerů s vyšším rozsahem napájecího napětí u PoE.

Plošný spoj jsem navrhl tak, aby bylo možné zdroj oddělit od zbytku elektroniky na desce – jak na vstupu (PoE), tak na výstupu (+3,3 V napájecí větev). To umožuje zkontrolovat správnost vstupu a výstupu napájení u testovací desky, což snižuje šanci na trvalé poškození při chybě v návrhu.

Při výběru několika komponent při navrhování plošného spoje nastal problém s miniaturizací. Většina těchto komponent musí vydržet vysoké napětí a proudy, to se bohužel odráží na jejich velikosti. Například pro vstupní kondenzátor pro PoE (na obrázku 2.4 označený jako C5) jsem z dostupných komponent vybral nejmenší možný. Ovšem při jeho parametrech – maximální napětí 35 V s kapacitou 470 μF stejně na desce zabírá značné množství místa a to



Obrázek 2.4: Výsledný návrh napájecí sekce desky – na obrázku lze vidět fyzické umístění jednotlivých součástek na navrženém plošném spoji, kondenzátor C5 je umístěný na vstupní straně (PoE in), čip XL1590-3.3E1 (U2) se stará o konverzi za pomoci cívky L1 a diody D2 – k uchování převedené elektrické energie slouží kondenzátor na výstupní straně (C32)

nejen v ploše, ale i ve výšce[25]. Hned po konektoru RJ45 je tento kondenzátor nejvyšší použitá součástka.

2.4 Návrh obvodu – hlavní mikrořadič

Srdcem jednotky je mikrořadič. Troufal bych si říct, že se jedná o nejdůležitější komponentu na celé desce. Jeho začlenění do návrhu bude věnován prostor v několika dalších podsekcích.

2.4.1 Vedlejší potřebné komponenty

Mikrořadiče řady STM32F103 jsou, alespoň co se týká požadavků při návrhu desky, vcelku nenáročné. U čipu je doporučeno použití tzv. „decoupling capacitors“. Ty slouží k odstranění šumu[26], který by se normálně propagoval zbytkem obvodu. V tomto případě je ke každému napájecímu (V_{DD}) pinu doporučeno [18] přiřadit jeden 10 μF a jeden 100 nF kondenzátor.

Dále je vhodné u mikrořadiče zafixovat výběr startovacího režimu (piny označené jako `BOOT`) – v případě této desky stačí zafixovat pin `BOOT0` na hodnu 0.

[25] To se bohužel podepsalo na designu vnějšího obalu – bude zmíněno v další kapitole
[26] Ten je způsoben běžným fungováním mikrořadiče – konkrétně jeho spotřebou

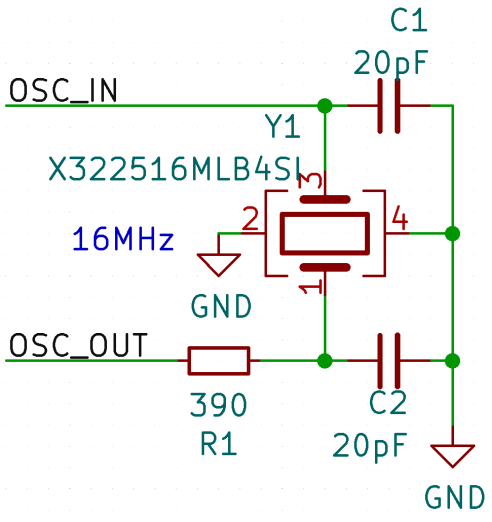
Ke splnění požadavků z datasheetu je potřeba přidat kondenzátor na `RESET` pin mikrořadiče (100 nF označovaný v mém schématu jako C3).

Mikrořadič sice obsahuje integrovaný oscilátor[27], ale pokud vývojář chce dosáhnout maximální frekvence, je nutné použít externí oscilátor. Důvod proč nelze využít interní oscilátor je ten, že jeho hodinový signál je před vstupem do PLL zpomalen na polovinu a maximální činitel při násobení hodinové frekvence v PLL je 16 – což po výpočtu dává maximální hodnotu 64 MHz. Konrétnějšímu nastavení hodin bude věnován prostor v sekci 3.3 v příští kapitole.

Posledním krokem návrhu obvodů kolem mikrořadiče je příprava vodičů potřebných k nahrání firmware, případně k jeho ladění, to bude rozebráno podrobněji až v sekci 2.4.3.

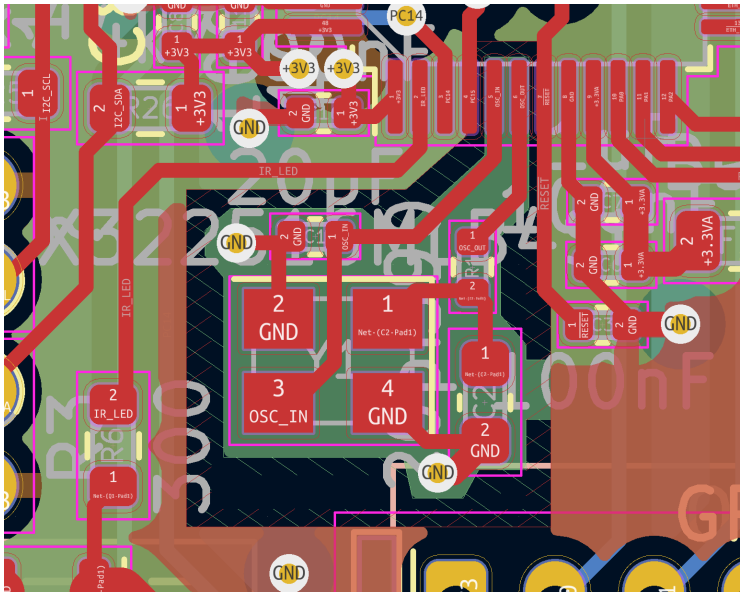
2.4.2 Obvod vysokorychlostních hodin

Ohledně externích krystalů vydalo STMicroelectronics velmi užitečné materiály [19], kde je popsáno vše od základních principů až po doporučení při návrhu plošných spojů. Snažil jsem se doporučení z materiálů držet. Z těch nejdůležitějších: vést co nejkratší cesty a pod oscilátor umístit oddělený „ground plane“ připojený k hlavnímu jen v jednom místě. Samotné schéma zapojení oscilátoru je vcelku jednoduché – viz obrázek 2.5. Na obrázku 2.6 se nalézá výsledný návrh plošného spoje pro externí rezonátor s dodrženými doporučeními – ten už je složitější.



Obrázek 2.5: Schéma zapojení oscilátoru u STM32

[27] 8 MHz



Obrázek 2.6: Návrh plošného spoje – zapojení oscilátoru u STM32

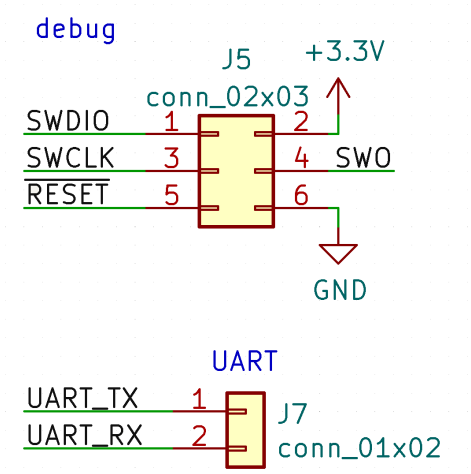
2.4.3 Hardware na desce potřebný pro ladění

I když je ladění za použití tzv. „printf metody“ velmi osvědčené a v praxi často používané, tak je občas vhodné používat propracovanější nástroje. Při programování v linuxovém prostředí lze například používat nástroj GDB (GNU Debugger). Tento rozšířený nástroj je přítomný prakticky v každé linuxové distribuci. Nabízí velké množství různých grafických rozhraní. GDB dokáže vypisovat části paměti, přepisovat hodnoty, měnit tok programu, a další. Díky jeho oblíbenosti se dočkal podpory pro mnoho různých platforem, a to právě i pro ARM Cortex-M3.

Pro připojení GDB k procesoru je potřeba, aby procesor měl připravené ladící rozhraní (SWD nebo JTAG) a k němu připojenou ladící desku. Ta následně komunikuje s počítačem (například přes USB). Toto zapojení je standardizované mezi mnoha procesory s ARM architekturou. Ve většině případů používá ale zbytečně velký konektor. Proto jsem z důvodu úspory místa konektor upravil a používám zapojení jak je vyobrazené na schématu z obrázku 2.7.

2.5 Návrhy obvodů zajišťující konektivitu jednotky

Aby jednotka chytrého domu byla schopná odesílat k analýze prováděná měření a jinak interagovat s reálným světem, musí být zajištěna nějaká forma propojení. Nejprve se budu věnovat čipu starajícímu se spolu s konektorem RJ45 o připojení k internetu a následně konektorům připojujícím jednotlivé



Obrázek 2.7: Zapojení konektorů pro ladění

rozšiřující moduly k hlavní desce.

2.5.1 Internetový řadič W5500

V dokumentaci k W5500 [14] je několik příkladů použití tohoto čipu v různých případech. Zvolil jsem správnou variantu a podle dokumentace navrhl obvod. „Decoupling capacitors“ se u tohoto čipu používají stejně jako u STM32 a taktéž obvod s oscilátorem (jen vyžaduje 25 MHz místo 16 MHz u procesoru). Analogová část – tedy propojení mezi čipem W5500 a transformátory (v mém případě integrováno v RJ45 konektoru, viz 2.5.2) – je to nejsložitější z celého návrhu. Je potřeba vypočítat impedance drah a několik dalších věcí. Naštěstí základní zapojení je popsáno v dokumentaci. Zbytek zapojení je přímočarý – několik kondenzátorů a rezistorů na speciální piny.

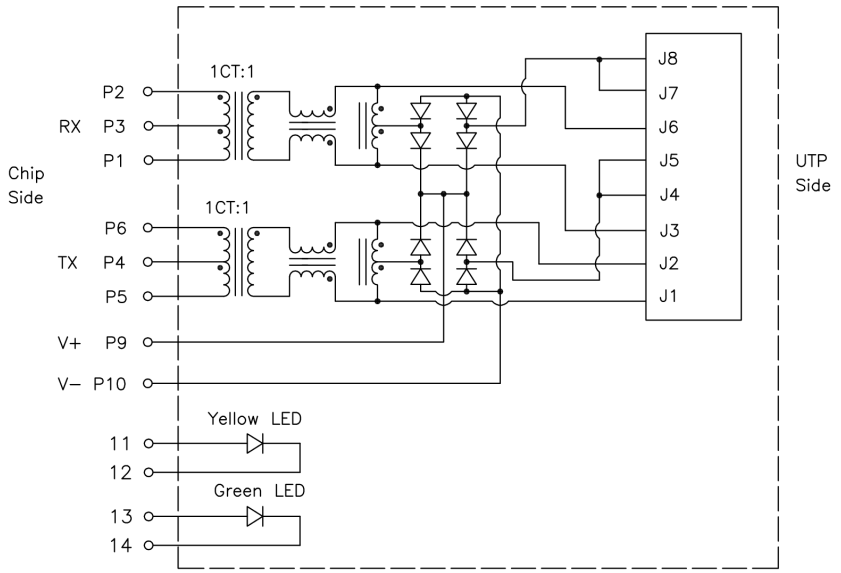
2.5.2 Ethernetový konektor

V ethernetové síti nejsou signály posílány přímo, ale za pomoci transformátorů – tím dochází ke galvanickému oddělení jednotlivých příjemců. V tomto případě lze vybírat mezi konektory bez transformátorů (ty musejí být následně umístěné externě), nebo vybrat konektor s integrovanými transformátory (to je většinou případ u Arduino modulů).

Tento projekt vyžaduje, aby bylo možné použít pasivní PoE. Proto je důležité zvolit správný konektor – pokud bych použil například HR911105A – de facto standard používaný v modulech pro Arduino – tak bych nebyl úspěšný (všechny piny jsou propojené přes sadu rezistorů vzájemě k sobě a ty by ná-

sledně nevydržely zatížení, další věcí je, že potřebné piny vůbec nejsou vyvedeny ven z konektoru).

Při procházení seznamu komponent u výrobce jsem nalezl jen jeden konektor splňující všechny požadavky (RJ45 s integrovanými transformátory podporující pasivní PoE) a to konektor HY931147C.



Obrázek 2.8: Zapojení konektoru HY931147C – obrázek převzat z datasheetu k čipu [20] – je dobré si povšimnout osmi usměrňovacích diod napojených na piny *P9* a *P10* (tyto piny slouží k vyvedení PoE z konektoru)

Schématická značka tohoto konektoru bohužel v databázi KiCadu chyběla, musel jsem jí proto během tvorby schématu přidat. Footprint pro součástku byl naštěstí stejný jako u jiné existující součástky, to práci usnadnilo, jinak by bylo potřeba ho podle datasheetu manuálně doplnit.

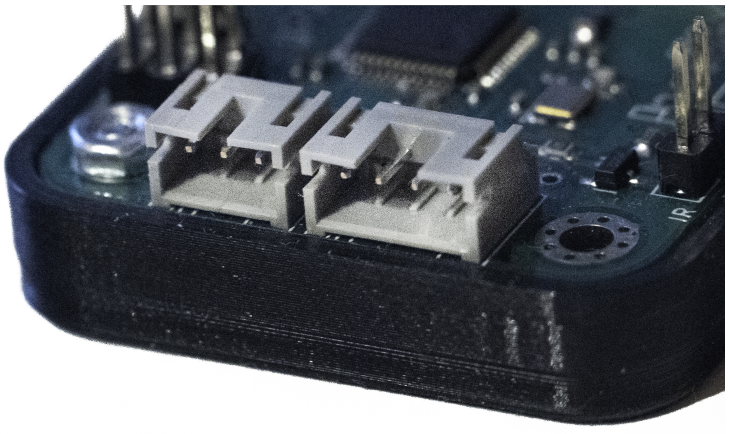
2.5.3 Konektory pro připojení ostatních modulů a senzorů

Deska samotná neobsahuje žádný senzor (tedy kromě čidla teploty uvnitř STM32, ten ale není přesný a je ovlivňován zatížením procesoru). Je tedy vhodné na desku umístit konektory, aby bylo možné senzory nebo jiná zařízení snadno zapojovat a odpojovat. Tím bude zároveň snížena pravděpodobnost chyby uživatele, jako například obrácená polarita. Vzhledem k množství využitelných pinů na mikrořadiči, jsem se rozhodl na desku umístit celkem 6 konektorů. Z toho dva GPIO konektory, jeden SPI (který lze ale také použít jako GPIO), jeden pro sběrnici OneWire a jeden pro I^2C (toto jsou tři nejčastěji používané sběrnice pro připojení senzorů). Poslední konektor je primárně

zamýšlený pro spínání IR diody (například pro ovládání klimatizací), lze ho ovšem využít k jakémukoli spínání, které vyžaduje mírně vyšší proudy než zvládnou dodat piny mikrořadiče. Důvodem je, že konektor není přímo napojený na mikrořadič, ale předchází mu MOSFET (číslo součástky: 2N7000), který dokáže zvládnout vyšší proudy (až 200 mA).

Co se týká podoby konektorů, tak jsem se rozhodl pro využití tzv. JST (*Japan Solderless Terminal*). Ty jsou vyráběny v několika různých sériích – navržené pro různě vysoká napětí, různě vysoké proudy, zamykání a jiné vlastnosti. Vybral jsem PH sérii, která je navržena, aby splňovala následující vlastnosti:

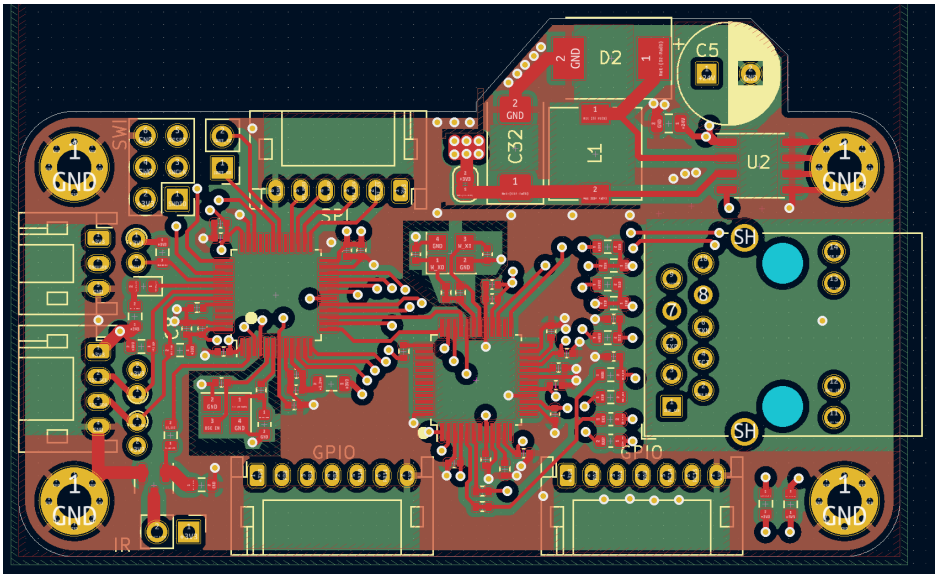
- 2,0 mm od pinu k pinu
- piny jsou v jedné řadě
- maximální proud – 2 A
- maximální napětí – 100 V
- bez uzamykání samce v samici



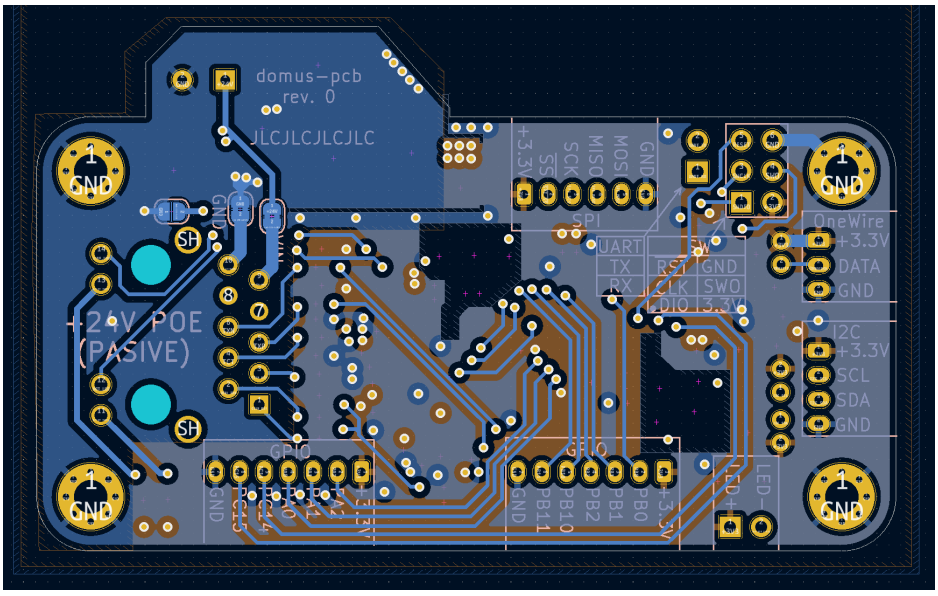
Obrázek 2.9: Ukázka vybraných konektorů – konektor *S3B-PH-KL(LF)(SN)* vlevo a konektor *S4B-PH-KL(LF)(SN)* vpravo

2.6 Dokončení návrhu plošného spoje

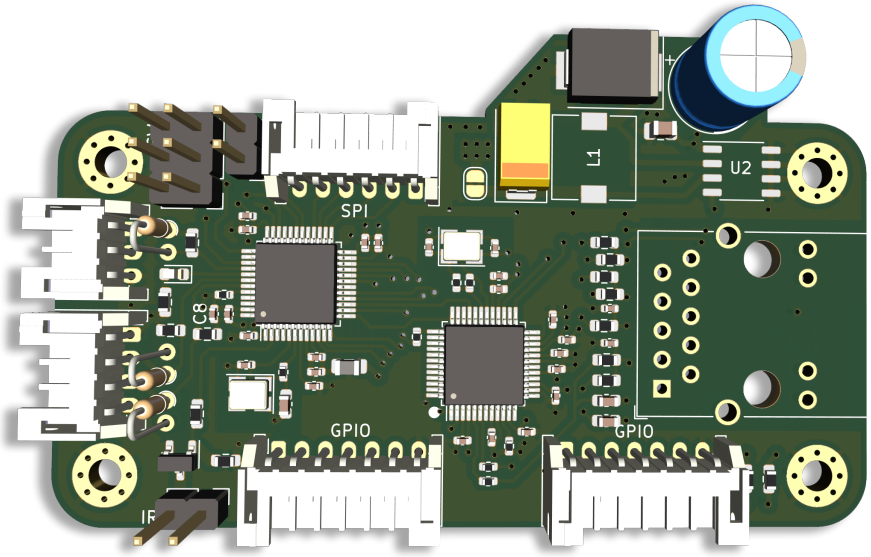
Veškeré routování cest na plošném spoji jsem dělal ručně (KiCad nepodporuje žádnou formu automatického routování). Během vytváření návrhu desky jsem volil různé tloušťky cest na PCB podle toho, jaký byl očekávaný proud na konkrétní cestě. Pro většinu cest digitálních signálů jsem použil tloušťku 0,3 mm – to odpovídá tloušťce pinů u obou čipů (pouzdro LQFP-48). U několika cest jsem vybral tloušťku 0,5 mm – například napájení signalizačních diod. Napájecí cesty, u kterých by byl očekáván nejvyšší možný proud, jsem zesílil až na tloušťku 1,0 mm.



Obrázek 2.10: Finální návrh PCB (přední strana) – na obrázku jsou vyobrazeny dvě horní vrstvy – červeně je znázorněna vrchní signálová vrstva plošného spoje a zeleně druhá napájecí (3,3 V) vrstva



Obrázek 2.11: Finální návrh PCB (zadní strana) – na obrázku jsou vyobrazeny spodní dvě vrstvy – modrá signálová vrstva je fyzicky na desce umístěna nejníž – žlutě je reprezentována třetí napájecí (GND) vrstva – na obrázku si lze povšimnout prázdného místa pod oscilátory (podle doporučení z datasheetu)



Obrázek 2.12: Výsledná podoba návrhu desky jako 3D modelu – vygenerováno v návrhovém software KiCad

Poslední důležitou věcí, kterou bych chtěl ohledně návrhu desky zmínit, je má snaha o jednoduchou upravitelnost designu. Pro správnou funkci sběrnic I^2C a OneWire jsou potřeba pull-up rezistory. Na desce jsou vybrány rezistory s nejtypičtějšími hodnotami. Konkrétně u OneWire je potřeba 4,7 kΩ rezistor. Během mých testů se ukázalo, že rezistor s takto vysokou rezistencí může být u delšího vedení problém. Proto jsem na desku přidal vedle SMD rezistorů také neosazené THC rezistory. To umožňuje velmi snadnou úpravu.

2.7 Návrhy vylepšení databáze

Návrh původní databáze byl koncipován tak, že v každé instanci chytrého domu bude server s lokální databází. Tento přístup je jednoduchý a funkční. V databázovém modelu se pro data ze senzorů nacházela pouze jedna tabulka a v ní několik sloupců, viz model podle tabulky 2.1.

jméno sloupce	datový typ	velikost [21]	komentář
`timestamp`	timestamp	8 bytů	datum a čas záznamu
`area`	varchar(45)	1 byte + délka	označení oblasti (pokoj)
`sensor`	varchar(45)	1 byte + délka	typ senzoru
`type`	varchar(45)	1 byte + délka	fyzikální veličina
`value`	real	4 byty	hodnota (desetinná)

Tabulka 2.1: Databázová tabulka pro data z původního návrhu

Pokud se zaměříme na paměťovou náročnost, tak po sečtení velikostí sloupců dostaneme 15 bytů + součet délek všech řetězců. Průměrná délka jednoho řetězce je okolo hodnoty 7. To znamená, že jeden řádek tabulky (pouze data bez indexů) obsadil v průměrném případě 36 bytů.

To poukazuje na několik problémů. Za prvé, řádek zabírá zbytečně příliš paměti, což rozhodně půjde vylepšit. Za druhé, při procházení tabulky, například při filtrování dat (pro vykreslení do grafu), musí probíhat operace pro porovnávání řetězců (zbytečně pomalé).

Jako elegantní řešení lze využít dekompozici a rozdělit data z jedné do více tabulek. V tom případě by bylo možné mít v jedné tabulce sloupce: `area`, `sensor`, `type` a do druhé tabulky přesunout pouze sloupce s hodnotou a datem. Tyto dvě tabulky bude potřeba propojit relací (tím obsadíme další paměť). Ukázka takového návrhu databáze je popsána v tabulce 2.2.

jméno sloupce	datový typ	velikost [21]	komentář
`timestamp`	timestamp	8 bytů	datum a čas záznamu
`value_type_id`	integer	4 byty	typ hodnoty
`value`	real	4 byty	hodnota (desetinná)
`value_type_id`	integer	4	typ hodnoty (PK)
`sh_instance_id`	integer	4	identifikace domu
`area`	varchar(45)	1 byte + délka	označení oblasti (pokoj)
`sensor`	varchar(45)	1 byte + délka	typ senzoru
`type`	varchar(45)	1 byte + délka	fyzikální veličina

Tabulka 2.2: Dvě datové tabulky po dekompozici

Pokud budeme uvažovat databázový model podle tabulky 2.2, tak přidáním jednoho měření[28] dojde k obsazení pouhých 16 bytů paměti. To je značná úspora oproti původním 36 bytům. Dále se ušetří strojový čas při porovnávání a filtrování. Místo porovnávání tří řetězců se provádí porovnávání jednoho čísla[29]. Sice je přidána další tabulka a v součtu by velikost byla větší, ovšem jedna tabulka je používána daleko více. Výsledkem je úspora místa a zvýšení výkonu upraveného návrhu.

Posledním krokem k úspoře místa by bylo data proložit nějakou matematickou funkcí a zapamatovat si její parametry pro určitý časový úsek – tomu se v této práci nevěnuji, mohlo by to ale ušetřit velké množství místa.

Do nového systému bude přidáno ještě několik dalších tabulek:

[28] Při porovnání množství druhů senzorů a množství záznamů pro senzor, můžeme přidávání dat do druhé tabulky zanedbat

[29] Opět můžeme zanedbat činnost, která je prováděna nepoměrně méněkrát

- uživatelské údaje (jméno, heslo)
- asociace domů s uživateli
- různé datové tabulky pro jiné typy

Pro většinu tabulek budou přidány pohledy (*views*), kde bude docházet k agregaci s ostatními tabulkami. To umožní nahlížet na databázový model podobným způsobem jako v původním návrhu.

Na ovládání klimatizací, kamen a ostatních prvků byly původně použity separátní tabulky, ale vzhledem k jejich malé velikosti je při tomto rozboru zanedbávám.

2.7.1 Vhodné použití indexů

Databázové indexy mohou zásadně ovlivnit výkon[30], za cenu využití prostředků navíc [22] – především paměti. PostgreSQL nabízí několik druhů indexů. Nejběžnějším typem je tzv. *B-Tree index* – ten je vhodný například při procházení rozsahu v seřazených datech.

Dalším velice používaným typem je *Hash index*. Převede dlouhou hodnotu na její otisk (*hash*) – který má jen 4 byty – a ten používá při porovnávání (pro mě nepříliš použitelný).

Typem ze kterého jsem v práci ovšem schopný dostat nejvyšší výkon je tzv. *BRIN index* (*Block Range INdexes*). Ten si pro každý (sekvenčně uložený) blok dat ukládá různé statistické údaje, díky kterým může snadno přeskakovat velké části paměti, o které v dotazu není zájem. Výhodou je, že velikost bloku lze nastavit a tím dostat požadované vlastnosti.

2.7.2 Výsledné zlepšení výkonu

Nejvyššího zvýšení výkonu bylo ovšem dosaženo jinou metodou. Při prvotním testování jsem používal pro databázi síťový disk[31], to se příliš neosvědčilo. I přes to, že síťová komunikace probíhala pouze lokálně (mezi jednotlivými virtuálními počítači na jednom stroji), databáze byla stále brzděna. Po několika testech jsem dospěl k názoru, že magnetické disky jsou pro databázi s více uživateli nedostatečné a rozhodl jsem se přejít na *NVME SSD*. Tím jsem dosáhl téměř 80× původního výkonu (počítáno vzhledem k počtu obsloužených transakcí, testováno pomocí nástroje *pgbench*).

[30] Při správném použití k lepšímu

[31] Který se skládal z několika magnetických disků v *RAIDu*

Kapitola 3

Implementace

První část této kapitoly obsahuje popis fyzické a virtuální síťové architektury. V další sekci se nachází popis schopností vybrané výrobní společnosti a detailnější popis finanční stránky výroby navrženého plošného spoje. Dále jsou v kapitole rozebrány otázky ohledně firmware, jeho nahrávání do jednotky a jeho zabezpečení proti odcizení. Následuje popis použitých knihoven s ukázkami kódu. V neposlední řadě kapitola obsahuje pár slov k vyrobenému vnějšímu ochrannému obalu a následném zapojení jednotek do systému. Kapitolu uzavírá několik sekcí zabývající se backendem, frontendem a jejich zabezpečením.

3.1 Fyzická architektura síťové infrastruktury

Pokud by projekt byl realizován větší firmou s dostatečným kapitálem, jistě by bylo investováno nemalé množství finančních prostředků do řádné síťové infrastruktury (případně by mohli provozovat servery v *cloudu* – což je pohodlnější, nicméně to přináší mnoho nevýhod[32]). Aby bylo možné nasimulovat situaci, že mám k dispozici dostatek hardware, rozhodl jsem se využít možností virtualizace. To znamená, že na jednom fyzickém serveru bude běžet několik virtuálních strojů, které budou vzájemně provázány do různých virtuálních sítí.

Na projekt vyhrazený fyzický server se skládá z následujícího hardware:

- Procesor *AMD Ryzen 7 3800X* – relativně výkonný procesor pro pracovní stanice (není přímo určený pro servery, nicméně zpracovává 16

[32]Především nutnost platit za pronájem serverů, menší kontrola nad uživatelskými daty a další.

vláken s maximální frekvencí 4.5 GHz), ten lze případně v budoucnu[33] vyměnit až za *Ryzen 5950X* (32 jader na 3.2 GHz)
- Základní deska *ASRock X570D4U-2L2T* – (před Koronavirovou krizí) relativně levná deska s chipsetem *X570* s podporou vzdálené správy (lze se dostat do BIOSu i pokud operační systém neběží – velmi vhodné pro servery)
- RAM Transcend *JM3200HLE-32G* – bohužel základní deska podporuje pouze napětí 1,2 V pro napájení RAM – tedy tato paměť není příliš rychlá a ani nepodporuje *ECC* (není vhodná pro servery). RAM jsou zapojeny v dual-channel konfiguraci 2× 32 Gb (na DDR4 CL22)
- Disků je v něm zapojeno požehnaně, dvojice NVMe disků (pro virtuální stroje) a 5 dalších SATA disků na data

I na takto „levném“ a nespecializovaném hardware si lze vyhrát s architekturou. Na serveru aktuálně běží kolem desítky různých virtuálních strojů, každý se specifickým účelem. Prvním je firewall, který se stará o blokování nechtěné komunikace a přeposílání chtěné dalším serverům. Dalších několik strojů obsluhuje lokální síť (nepotřebné vzhledem k této práci). Následuje server, na kterém běží Nginx v konfiguraci reverzní proxy. Ten se stará o veškeré šifrování a přeposílání požadavků jednotlivým serverům podle poddomény. Dále jsou virtualizovány „cílové“ servery, jeden s databází a jeden který se stará o frontend zpracování požadavků (zjednodušená architektura je naznačena na obrázku 3.1, všechny virtuální servery jsou umístěny v *LAN kyvit*).

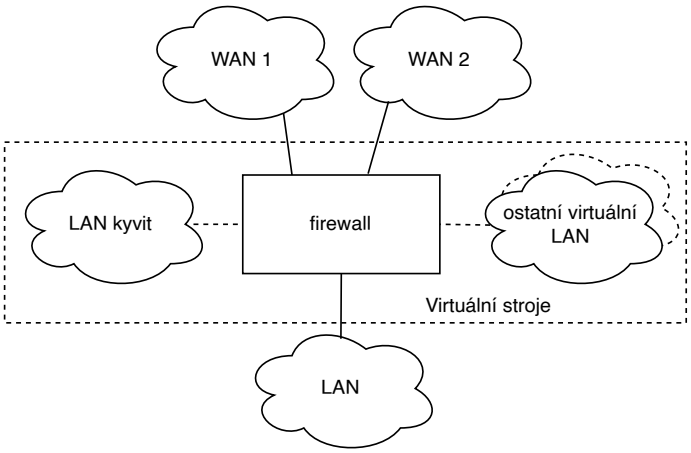
Všechny virtuální stroje jsou rozdělené do různých sítí a oddělené od zbytku (domácí) LAN – kde probíhá komunikace běžných uživatelských zařízení. Za pomoci operačního systému[34] *Proxmox* jsem nastavil síťování jako na schématu z obrázku 3.1. Obsah sítě s virtuálními stroji (*LAN kyvit*) je nastíněn v návrhovém schématu z předchozí kapitoly (obrázek 2.1).

3.2 Tovární výroba plošného spoje a náklady s tím spojené

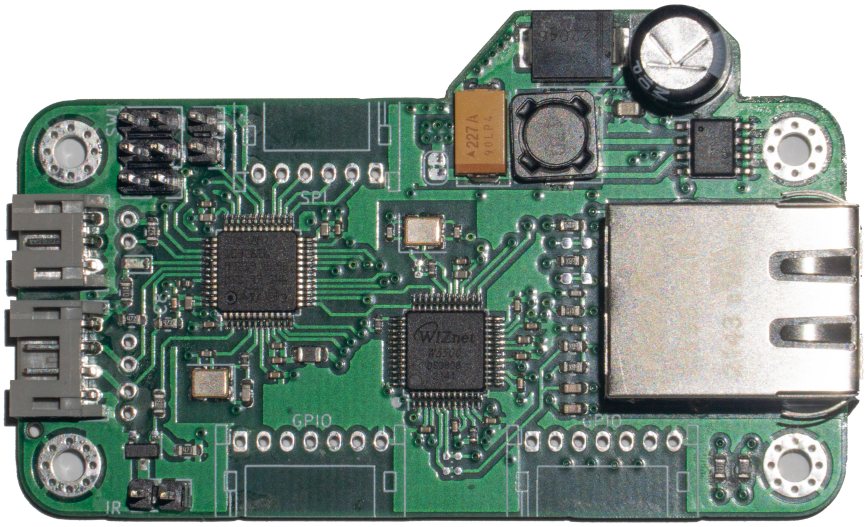
K objednání prototypu plošných spojů jsem se rozhodl využít jedné z mnoha čínských firem. Konkrétně se jedná o společnost JLCPCB. Patří k nejlevnější na trhu, ze všech které se mi podařilo dohledat. Podporuje jak výrobu samotného plošného spoje, tak jeho následné osazení SMD součástkami a případně ruční doletování komponent, které nelze osadit automaticky.

Seznam výrobních schopností vybrané továrny [23]:

[33] Při zachování všech ostatních komponent.
[34] Je to spíše softwarový nástroj vydávaný spolu s OS – používají Debian
[35] Bohužel nešly v dostatečně krátkém termínu objednat ani z lokálních zdrojů v Evropě



Obrázek 3.1: Architektura síťové infrastruktury – vyšrafovaný čtverec reprezentuje komponenty umístěné uvnitř jednoho fyzického serveru – šrafované čáry reprezentují komunikaci po virtuálních síťových kartách a nešrafované reprezentují komunikaci pomocí fyzických rozhraní – dále jsou na obrázku zobrazeny dva *uplinky* od různých internetových dodavatelů zajišťují zálohování systému proti výpadku internetu – jako *firewall* je označený virtuální stroj starající se o dělbu komunikace a udržování spojení – *LAN kyvit* je označena lokální virtuální síť pro virtuální firemní servery – LAN je klasická domácí síť



Obrázek 3.2: Vyrobená deska – téměř kompletní – chybí osadit 3 konektory, to bohužel kvůli nedostatku součástek nebylo v době výroby možné[35]

- vícevrstvé spoje – 1, 2, 4 nebo 6 vrstev
- maximální rozměry desky – 400 mm × 500 mm
- tolerance rozměrů pro CNC – ±0,2 mm
- průměr vrtaných otvorů – 0,2 mm-6,33 mm (s tolerancemi +0,13/−0.08)

Díky vhodně vybranému návrhovému programu nebyl se zadáním výroby žádný problém. Vše, co je potřeba udělat, je nahrání tří souborů vygenerovaných doplňkem KiCadu. To jsou Gerber soubory (zkomprimované ve formátu zip), rozpis součástek (BOM[36] – *Bill of materials*) a soubory s polohou součástek (POS).

Náklady

Vzhledem k výběru továrny v Číně je bohužel třeba navíc počítat (na rozdíl od lokální výroby) s vyšší cenou za dopravu (a je třeba zaplatit celní poplatky). Celkem mě výroba s dopravou prototypů přišla na 315,69 $[37]. Rozpis jednotlivých cenových položek:

- výroba desek plošných spojů – 8,00 $
- osazení součástkami – 213,33 $
  - poplatek za přípravu – 8,00 $
  - vytvoření šablony – 1,50 $
  - cena komponent – 171,98 $
  - poplatek za rozšířené komponenty – 21,00 $
  - SMT montáž – 3,69 $
  - ruční pájení součástek – 3,50 $
  - ruční osazení součástkami – 3,66 $
- poštovné (včetně celních poplatků) – 92,83 $
- poplatek společnosti PayPal – 1,53 $

3.3 Firmware pro jednotku

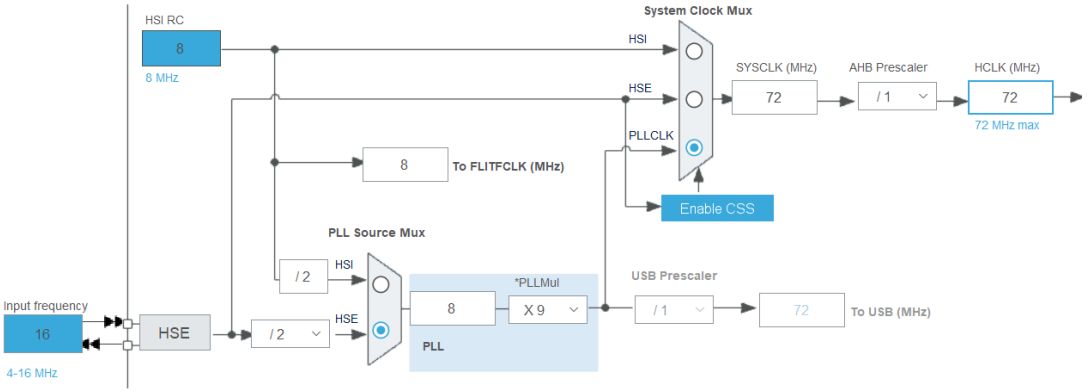
Pro implementaci software v jednotce jsem se rozhodl využít Arduino Core STM32 (stm32duino). Výhodou je krátký vývojový čas a hlavně stabilní API (stabilní přes všechny platformy, od ATtiny85, přes ATmega328P a STM32F103 až po ESP32). Další výhodou je množství (a přenositelnost) existujících knihoven, které hojně využívám.

Výhodou platformy STM32 (teď konkrétně bez Arduino nadstavby) je snadné nastavení některých důležitých parametrů. Například na obrázku 3.3 lze vidět,

[36] JLCPCB vyžadují zadání ve formátu CSV

[37] Když jsem zkoušel zadávat výrobu o pár týdnů později, tak rapidně klesla cena u některých součástek, tím by se celková výroba zlevnila o více než 50 $

jak snadné je nastavení hodin v STM32CubeIDE. Vývojové prostředí následně vygeneruje inicializační funkci, kterou lze jednoduše zavolat a nastavit tím procesor.



Obrázek 3.3: Nastavení hodinového signálu v STM32CubeIDE – pro přehlednost zkráceno

Stm32duino umožňuje přidat podporu vlastní desky vytvořením a vyplněním několika souborů podle šablony. Prakticky je třeba jen vyplnit potřebné údaje pro linker, dále které piny/periferie jsou používané, dodat nastavení PPL/HSE vygenerované v STM32CubeIDE a nastavení pro desku, aby jí Arduino IDE správně detekovalo a zobrazovalo. Toto v práci aktuálně řešeno nebude, desky jsou zatím jen funkčním prototypem, který se může měnit, a zatím na vše může být použita generická deska a díky *WEAK* C funkcím mohou být některá důležitá chování přepsána.

V ukázkovém kódu 3.1 lze pozorovat použití *WEAK* funkcí v praxi.

3.3.1 Náhrávání firmware do desky

V procesorech STM32F103 může být nahrán zavaděč, který umožňuje přepisování flash paměti pomocí sériové linky. Této vlastnosti jsem se rozhodl nevyužít a nahrávat program přímo přes SWD (*Serial Wire Debugging*) rozhraní.

Před nahráním programu pomocí Arduino IDE je potřeba správně nastavit parametry:

- Deska (upřesnění) – Generic STM32F1 series (Generic F103C8Tx)
- Podpora sériové linky – ano
- Veškerá podpora USB – vypnuta
- Optimalizace – pro produkci – optimalizace pro velikost (-Os)
- Symboly pro ladění – pro produkci – vypnuto

```
// ukázka ze souboru:
//   Arduino_Core_STM32/variants/STM32F1xx/
//     F103C8T_F103CB(T-U)/variant_PILL_F103Cx.cpp
WEAK void SystemClock_Config(void)
{
    // rest of function...
}

// soubor z arduino projektu
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    // rest of function...
}
```

Kód 3.1: Kód přepisující chování WEAK funkce inicializující hodiny – vrchní (defaultní) funkce je během kompilace nahrazena spodní (uživatelskou) funkcí místo toho, aby došlo ke kompilační chybě (dvakrát definovaná stejná funkce)

- Knihovna jazyka C – Newlib Nano
- Nahrávání – STM32CubeProgrammer (SWD)
- Port – podle přiřazení operačním systémem (např. COM3 na Windows)

3.3.2 Otázky zabezpečení firmware a zranitelnosti použitelného mikrořadiče

V případě finálního vydání produktu je vhodné uzamknout flash paměť obsahující program, aby nemohlo dojít k přečtení/přepisu útočníkem. Na to mikroprocesory a mikrořadiče od STMicroelectronics nabízí několik režimů uzamčení paměti, které lze využít.

Bohužel v systému mikrořadičů řady `STM32F1` byla objevena zranitelnost zneužívající otevřené ladící rozhraní, kdy pomocí výjimek lze postupně extrahovat obsah paměti.

3.4 Použití externích knihoven ve firmware

Knihovny jsou nedílnou součástí programování. Umožňují vývojáři přeskočit tu část, kde musí *znovu vynalézat kolo*, a dovolují mu soustředit se na ty

důležité části. Proto několik dalších podsekcí věnuji některým z důležitých knihoven, které používám.

3.4.1 Ethernet

Obsáhlá knihovna implementující základní funkcionalitu pro práci se síťovými službami (mimo jiné například DHCP). Umožňuje jednoduše připojit Arduino nebo jinou kompatibilní desku k internetu. Spolu s čipem W5500 poskytuje možnost mít najednou otevřených až osm nezávislých socketů – samozřejmě jen pokud na to vývojář má ve svém mikrořadiči dostatek paměti. Knihovnu používám pouze nepřímo, proto ukázku zdrojového kódu vypustím.

Drobný problém u této knihovny spočívá v nesnadném přepsání výchozího nastavení pro SPI komunikaci – upravit sice lze, ovšem úprava se musí přepsat přímo ve zdrojovém kódu knihovny – daleko vhodnější by bylo použití direktivy jazyka C `#IFNDEF` v kombinaci s „build_opt.h“ souborem. V základu se pro komunikaci po SPI používá frekvence 8 MHz, což zbytečně zdržuje komunikaci mezi hlavním mikrořadičem desky a čipem W5500. *STM32F103C8T6* podporuje komunikaci po SPI rychlostí s nosnou frekvencí až 18 MHz, čímž by se přenos mohl zrychlit téměř dvojnásobně – hlavní mikrořadič by tedy mohl delší dobu být v režimu spánku a to by snížilo celkovou spotřebu systému, případně by se procesorový čas mohl použít na něco jiného.

3.4.2 NTPClient

Jednoduchá knihovna, která používá NTP (*Network Time Protocol*) pro synchronizaci a udržení aktuálního času. Ve firmware jí používám, protože to je nejjednodušší cesta, jak v zařízení s internetem udržet přesný čas. Výhoda oproti použití hardwarového RTC je absence potřeby mít „záložní“ zdroj energie, který musí udržet RTC v běhu přes čas bez hlavního napájení (většinou se používají baterie, nejběžněji *CR2032*).

Metoda `ntpClient.begin()` v kódu 3.2 slouží k inicializaci knihovny (stejně jako ve většině Arduino knihoven). Důležité je v pravidelných intervalech volat funkci `ntpClient.update()`, která se stará o udržování aktuálního času v interní proměnné uvnitř knihovny. Tato metoda je navržena tak, aby zbytečně nezatěžovala vzdálený server. Proto i když bude volána vícekrát za sebou, tak ke komunikaci na serverem dojde jen jednou za čas – ten lze nastavit v knihovně. Poslední důležitou metodou je `ntpClient.getEpochTime()`. Ta vrátí aktuální časovou epochu (unix timestamp).

3.4.3 SSLClient/BearSSL

SSLClient je pouze wrapper okolo knihovny BearSSL (to je univerzální C knihovna). SSLClient přidává podporu TLS 1.2 pro kteroukoli síťovou kni-

```
NTPClient ntpClient;  // na začátku programu
ntpClient.begin();

ntpClient.update();  // funkce volána v pravidelných
↪  intervalech

// získání aktuálního času
ntpClient.update();
debug << "ntp: actual time: " <<
↪  String(ntpClient.getEpochTime()) << "\n";
```

Kód 3.2: Použití knihovny NTPClient

hovnu, která dodržuje původní Arduino (Client) API.

Minimální požadavky pro „celou“[38] knihovnu jsou 110 kB flash paměti a 7 kB RAM. To bohužel nesplňuje vybraný procesor, ale použití této knihovnu by během budoucího vývoje měla být priorita. Umožnilo by to používání samostatných senzorů i v nedůvěryhodných sítích – což rozhodně stojí za můj budoucí zájem. Po bližší inspekci BearSSL by možná nebylo nemožné do STM32F103 vmáčknout jeden podporovaný režim. To je dobrá otázka pro budoucí experimentování.

Aktuálně jedinou funkcí, která je ve firmware z této knihovny využita, je `SHA256`. Pomocí té je vypočítán unikátní 128 bit identifikátor jednotky a MAC adresa (za pomocí unikátního 96 bit identifikátoru a „soli“ zabudované ve firmware).

3.4.4 arduino-mqtt

Podobně jako SSLClient obaluje jinou knihovnu, v tomto případě je obalena knihovna lwmqtt (Light Weight MQTT). Tato knihovna implementuje MQTT verzi 3.1.1. Ukázka použití knihovny v kódu 3.3.

V ukázkovém kódu 3.3 je kromě ukázkového kódu knihovny názorně předvedeno, že Arduino používá plně funkční C++ kompilátor. V ukázce lze vidět použití standardních řetězců jazyka C++ (`std::string`) místo řetězců z Arduina (`String`). Dále je v kódu vidět použití metody *perfect-forwarding* – to je pokročilá funkcionalita jazyka C++.

[38] Minimální podmnožina protokolů, kterou autor knihovny uznal za minimální – lze zmenšit

```
MQTTClient mqttClient;

// metoda používající perfect-forwarding z c++ - ukázka toho,
↪  že kompilátor implementuje většinu funkcionalit z c++
template<typename T>
bool mqttPublish(const std::string& topic, T&& arg)
{
    return mqttClient.publish(("boards/" + uuid + "/" +
    ↪  topic).c_str(), std::forward<T>(arg));
}

// funkce volané během inicializace desky
mqtt_connect();
mqttPublish(
    "status", String() +
    "{"
        "\"connected_since\": " +
        ↪  String(ntpClient.getEpochTime()) + ", "
        "\"dhcp_address\": \"" +
        ↪  ip_to_string(Ethernet.localIP()) + "\""
    "}"
);
```

Kód 3.3: Ukázka rozhraní knihovny arduino-mqtt

3.5 Vnější ochranný obal

Vnější ochranný obal jsem nenavrhoval ani nevyráběl já, nýbrž kolega (a mimo jiné spoluvlastník firmy Kyvit s.r.o). Krabička byla vyrobena na 3D tiskárně, výsledný produkt (i s otvorem pro příliš vysoký kondenzátor) ve své finální podobě na obrázku 3.4.

3.6 Finální zapojení

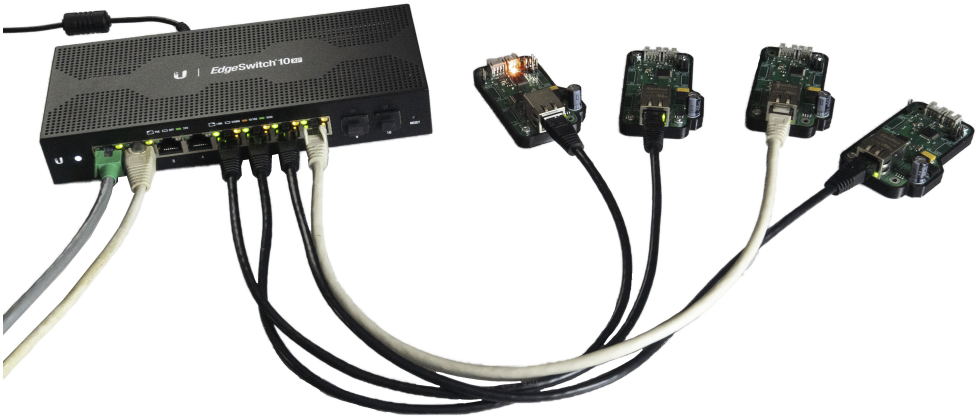
Po zkompletování návrhu a vyrobení plošného spoje bylo potřeba systém zapojit. Testování samotnému je věnována další kapitola (4).

Aby systém fungoval, je potřeba mít jednotky zapojené do *switche*[39], který podporuje PoE. Rozhodl jsem se použít *Ubiquiti EdgeSwitch 10XP, PoE*, protože byl zrovna k dispozici. Ale na konkrétním výrobci nezáleží. Je důležité, aby switch podporoval pasivní PoE v rozmezí od 5 V do 24 V a zároveň podpo-

[39] Síťový přepínač

Obrázek 3.4: Vnější ochranný obal chránící plošný spoj – složený ze dvou dílů (spojování pomocí šroubů a matic) – vytisknutá na 3D tiskárně

roval přinejmenším *10BaseT ethernet*. Čtyři jednotky zapojené do vybraného switche na obrázku 3.5.



Obrázek 3.5: Zapojení jednotek do switche – nejlevější kabel je internetový *uplink*, druhý zleva je připojený počítač (kvůli jednoduššímu ladění chyb), všechny 4 zapojené kabely na pravé straně slouží k napájení přes PoE a zajišťují připojení jednotek do sítě

3.7 Tvorba webového rozhraní

Backend byl implementován podle návrhu. FastAPI aplikace komunikuje s databází pomocí knihovny `psycopg_pool`. Tato knihovna během startu programu otevře několik desítek spojení a ty během obsluhování různých požadavků „recykluje“. Díky tomu nové požadavky nemusí čekat na vytvoření nového spojení a jsou obslouženy rychleji.

Flask aplikace obsluhuje požadavky skrze aplikační rozhraní vytvářené FastAPI. Propojení mezi Flaskem a API zajišťuje knikovna `requests`.

Na ukázku se lze podívat na kód 3.4 vykreslující hlavní stránku.

```
# main pages
@app.route('/')
@app.route('/layout')
@app.route('/layout/<string:layout>')
def index(layout: str = None):
    return render_template('index.html')
```

Kód 3.4: Kód vykreslující hlavní stránku

3.8 Zabezpečení frontendu a backendu

Většina operačních systémů na serverech projektu je minimálně „Debian-based“, pokud na Debianu neběží přímo. To znamená že většina z nich používá jako svůj *init system* software jménem *systemd*. Dle analýzy v 1.13.2 byly v *systemd* zapnuty příslušné zabezpečující volby. *Systemd* používá ke zjednodušené analýze „zabezpečení“ příkaz `systemd-analyze security`.

Zapnutá bezpečnostní nastavení pro aplikaci:

- spuštěná pod uživatem bez oprávnění (`User`)
- nemůže získavat nová oprávnění (`NoNewPrivileges`)
- má přístup do systému pouze pro čtení (`ProtectSystem`)
- aplikace v souborovém systému `/proc/` uvidí jen sebe (`ProtectProc`)
- žádný přístup k dočasným složkám ostatních procesů (`PrivateTmp`)
- žádný přístup k hardwarovým zařízením (`PrivateDevices`)
- žádný přístup k modulům jádra (`ProtectKernelModules`)
- a spoustu dalších

Tyto volby jsou zapnuté jak pro službu backendu, tak pro službu frontendu.

Kapitola 4

Testování

V první části následující kapitoly jsou rozebrány některé postřehy z dlouhodobějšího testování starého systému. V druhé, nejdelší části, je do detailu popsáno testování vyhotovených plošných spojů. Nejprve jsou rozebrány zemní smyčky (a jak se jim vyhnout). Dále prvotní testování desek, po té co dorazily z výroby. Následuje sekce o podezřele vysoké teplotě desek při běžném používání. Sekci o testování plošných spojů uzavírá popis několika měření na různých částech obvodů na desce. Kapitola končí testováním uživatelského rozhraní.

4.1 Poučení z testování jednotek původního systému

Rád bych věnoval pár slov tomu, jak se osvědčil starý systém (předchůdce této práce). Lze si z toho odnést několik zajímavých postřehů použitelných i pro tuto práci. S mikrořadiči (ATmega328P) a komunikační sběrnicí nebyl během posledních tří let žádný problém, ale některé senzory způsobovaly problémy. Čidla umístěná na sběrnicích SPI, nebo I^2C komunikovala bez problémů, to ovšem nelze říct o čidlech umístěných na sběrnici OneWire. Konkrétně u čipů *DS18B20* bylo nutné komunikaci i několikrát za sebou opakovat. Situaci zlepšilo použití „silnějších“ rezistorů[40] na OneWire sběrnici, což nasvědčuje tomu, že šlo o problém se zarušením (ale neproběhla příliš hluboká diagnóza). Tedy u tohoto nového systému bych spíše než čidla *DS18B20* doporučoval používat jejich SPI nebo I^2C alternativy.

[40] Místo doporučených 4,7 kΩ například 2,2 kΩ

4.2 Testování plošného spoje

Nejdůležitější částí testování bylo testování hardwarové stránky projektu, tedy testování vyrobeného plošného spoje. Protože na rozdíl od software se chyba na plošném spoji velice těžko opravuje na dálku.

4.2.1 Problémy při měření – zemní smyčky

Na začátek je dobré si rozebrat zapojení a nástroje použité k testování. Na měření budu používat osciloskop *Rigol DS1054Z* a pro napájení a zapojení jednotek budu používat switch *EdgeSwitch 10XP*.

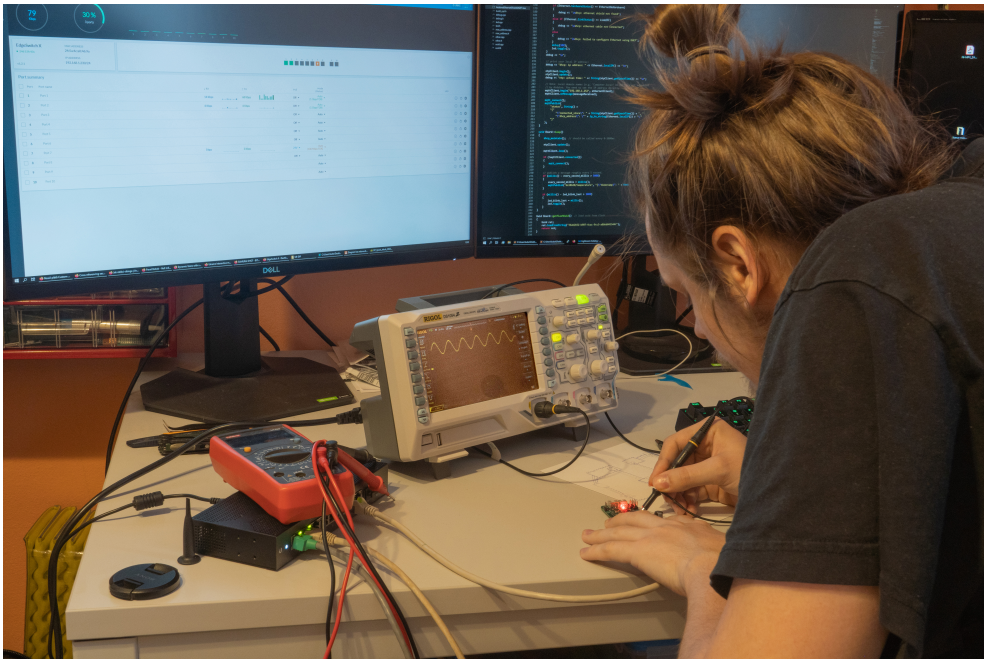
Měl bych čtenáři ve zkratce nastínit, co to vlastně je zemní smyčka (anglicky *ground loop* nebo *earth loop*). Pokud budu mít nějaká dvě zařízení, obě s vlastní zemí, a pomocí jednoho vodiče je propojím, vzniká mi uzavřený obvod (smyčka) skrz země obou zařízení. Tedy pokud mají rozdílná napětí (jiný potenciál), tak to může způsobovat velké množství problémů.

Bohužel kombinace mého osciloskopu – kde zemnění sondy je napojené na zemnění v napájecí zásuvce (220 V) – a switche takové zemní smyčky způsobuje. Rozhodl jsem se to proměřit a mezi zemí sondy (osciloskop) a zemí pasivního PoE ze switche je kolem tří čtvrtin voltu (při zkratu tam protéká přibližně 33 mA). V praxi je problém většinou řešen použitím diferenciální sondy, která je bohužel mimo můj rozpočet. Proto jsem se rozhodl, že přeruším zemnící drát[41] v napájecím kabelu osciloskopu – tím by mělo dojít k přerušení zemní smyčky (je nutné zkontrolovat, že zem osciloskopu opravdu „plave“ a že toto byla jediná cesta uzemňující osciloskop).

4.2.2 Prvotní ověření funkčnosti plošného spoje po výrobě

Deska byla navržena, aby ji bylo možné rozdělit do několika částí a ty postupně testovat. Konkrétně zdroj lze odpojit od vstupního napájení a od výstupu pro zbytek desky, to jsem provedl a postupně zkontroloval napětí – vstupní se pohybovalo mezi 21-23 V, to je očekávané (je dodáváno switchem). Následně jsem pomocí pájky opět propojil přerušený „solder bridge“ a naměřil na výstupu ze zdroje napětí mezi 3,2 V a 3,3 V, tedy i zdroj na desce fungoval podle očekávání. Pokračoval jsem v testování dále a po připojení hlavního napájení se W5500 okamžitě probral a připojil se ke switchi (signalizováno pomocí LED v konektoru). Čip z továrny bohužel nemá nahraný žádný firmware, tedy nebylo na první pohled zřejmé, zda funguje. Zapojil jsem tedy debugger jako na obrázku 4.2 a zkusil nahrát software.

[41] Není to bezpečné a sami to prosím nezkoušejte – důrazně doporučuji to nedělat



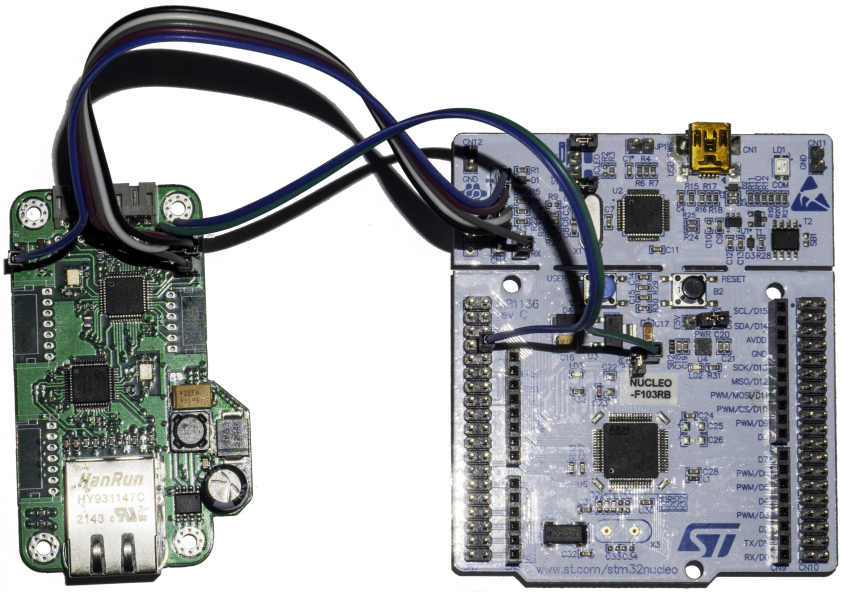
Obrázek 4.1: Měření na osciloskopu – napětí na oscilátoru pro hlavní procesor

Obrázek 4.2 může vypadat velice podobně jako obrázek 2.3, ovšem je zde mnoho zásadních rozdílů. Zatímco na obrázku 2.3 je používán procesor na vývojové desce, tak na obrázku 4.2 je z vývojové desky používána pouze horní část (debugger) a zdroj napájení – procesor na vývojové desce (čip vpravo uprostřed) je kompletně odpojen od napájení.

Během ladění jsem si všiml, že popis sériové linky na spodní straně desky neodpovídá funkcím pinů na mikrořadiči (první nalezená chyba), ovšem to naštěstí nijak nebrání použitelnosti desky (piny stačí prohodit), jen je na to potřeba myslet během ladění.

Bohužel během měření signálu se mi podařilo jednu desku poškodit. Měl jsem připojenou zem od sondy k jedné z kotvících děr (jsou připojené k zemi) a během manipulace s druhou stranou sondy se mi podařilo vyzkratovat zemnění k 3,3 V napájení. Po incidentu přestal fungovat čip W5500, systému se nepodaří navázat spojení (*link* – proces zajištění spojení v Ethernetu), ovšem procesor je možno naprogramovat a na komunikační sběrnicí mezi hlavním procesorem a W5500 je vidět snaha hlavního mikrořadiče o komunikaci (bez odpovědi W5500).

Během měření základních funkcí jsem přišel na to, že externí krystal u STM32 není používaný (u W5500 byl signál naprosto zřejmý). Po dlouhém procházení



Obrázek 4.2: Připojení desky k debuggeru – programování a ladění – laděná (vyvíjená) deska se nachází v levé části obrázku a vývojová deska Nucleo v pravé – pro komunikaci se používá SWD v kombinaci s rozhraním UART, o napájení se stará vývojová deska

kódu jsem přišel na to, že generická implementace desky pro STM32F103 vynechává kód k inicializaci externího krystalu, po jeho dodání již byla na krystalu jasně měřitelná frekvence.

4.2.3 Zvýšená teplota PCB

Během dlouhodobějšího testování plošného spoje umístěného v krabičce, jsem si všiml, že dochází k nadměrnému zahřívání plošného spoje. Teplo je generováno primárně internetový čipem W5500, procesorem STM32 a součástkami v okolí zdroje. Pokud je PCB umístěno volně bez krabičky, tak součástky jsou na dotek sice teplé, ale nijak výrazně.

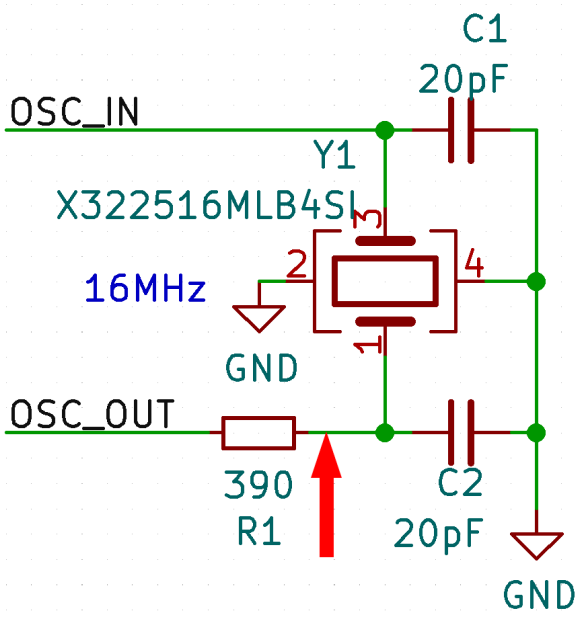
I když dochází k tomuto nechtěnému zahřívání, tak během týdenního testování nedošlo k žádným problémům a deska fungovala správně. Mohlo by se stát, že vyšší teplota sníží dlouhodobou spolehlivost a oslabí výdrž součástek. Proto by bylo vhodné se v budoucnu na problém zaměřit.

U mikrořadiče STM32 lze zahřívání značně omezit pomocí snížení hodinové frekvence, případně využíváním úsporných režimů v moment, kdy procesor „pouze čeká“ (nyní se využívá tzv. *busy waiting*). U W5500 je řešení složitější, protože čip nelze vypnout, musí odpovídat na externí požadavky. Co udělat

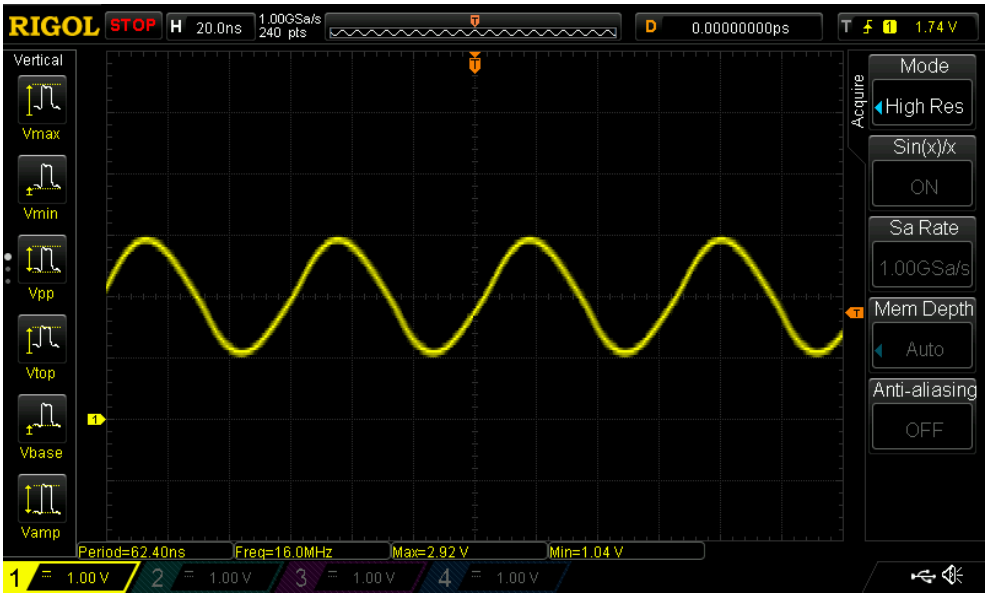
lze, je snížit rychlost z 100 Mbit/s na 10 Mbit/s, tím dojde o snížení spotřeby téměř o polovinu. Sada těchto opatření by v budoucnu mohla zabránit zvýšené teplotě plošného spoje.

4.2.4 Důkladnější kontrola elektronické funkčnosti desky

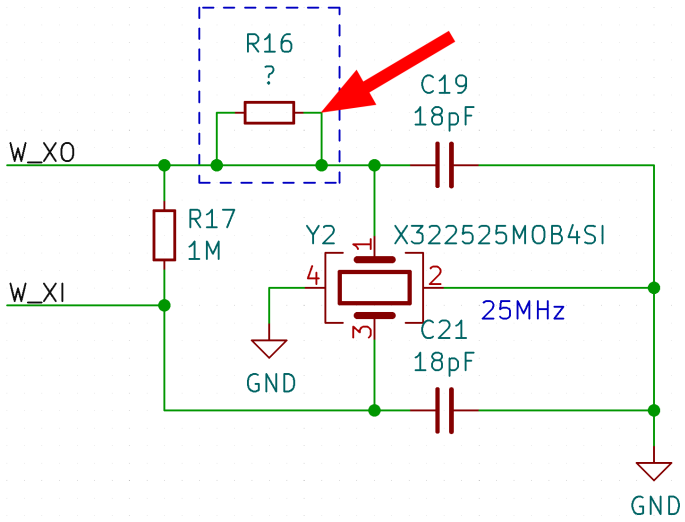
Abych ověřil správnou funkčnost plošného spoje, provedl jsem pomocí osciloskopu mnoho měření v různých částech desky. Primárně jsem se zaměřoval na správnou funkčnost obou použitých čipů – tedy správná hodnota napájení ze zdroje, správný signál na rezonátoru a nakonec jsem ověřil integritu signálu na sběrnici SPI mezi příslušnými čipy. Ve zbytku sekce se nachází vybraná měření s příslušnými popisky.



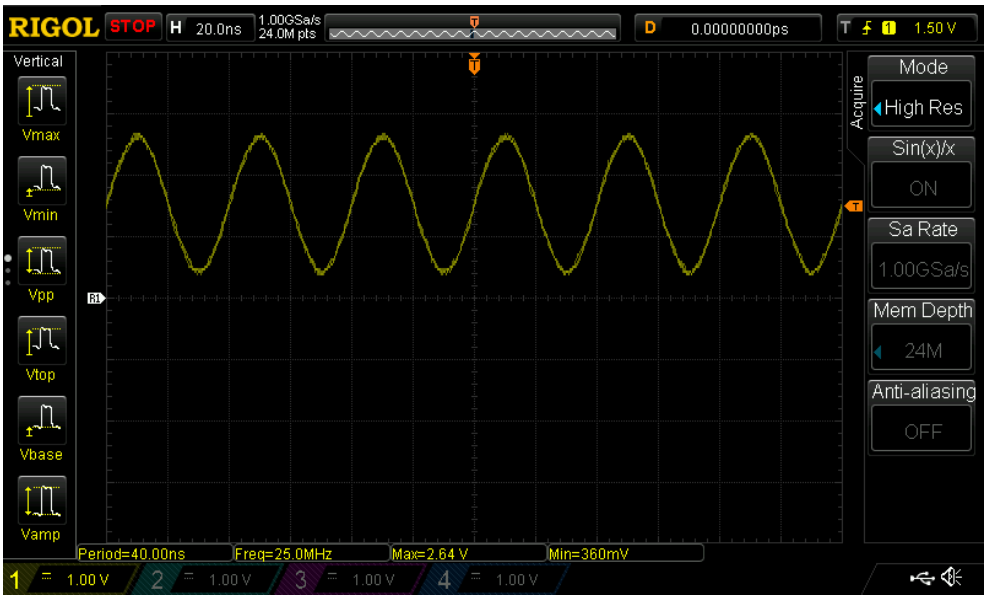
Obrázek 4.3: Schéma k měření 4.4 – měřené místo je označeno červenou šipkou



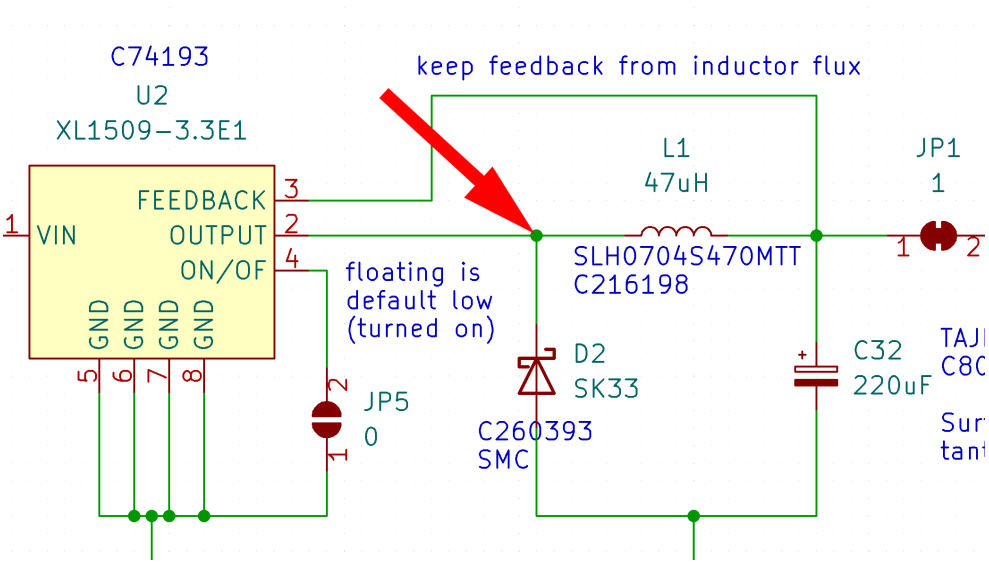
Obrázek 4.4: Měření napětí na oscilátoru pro hlavní procesor – signál snímán přímo mezi krystalem a procesorem STM32 – na obrázku můžeme pozorovat správný průběh signálu, a to že je generována frekvence přesně 16 MHz



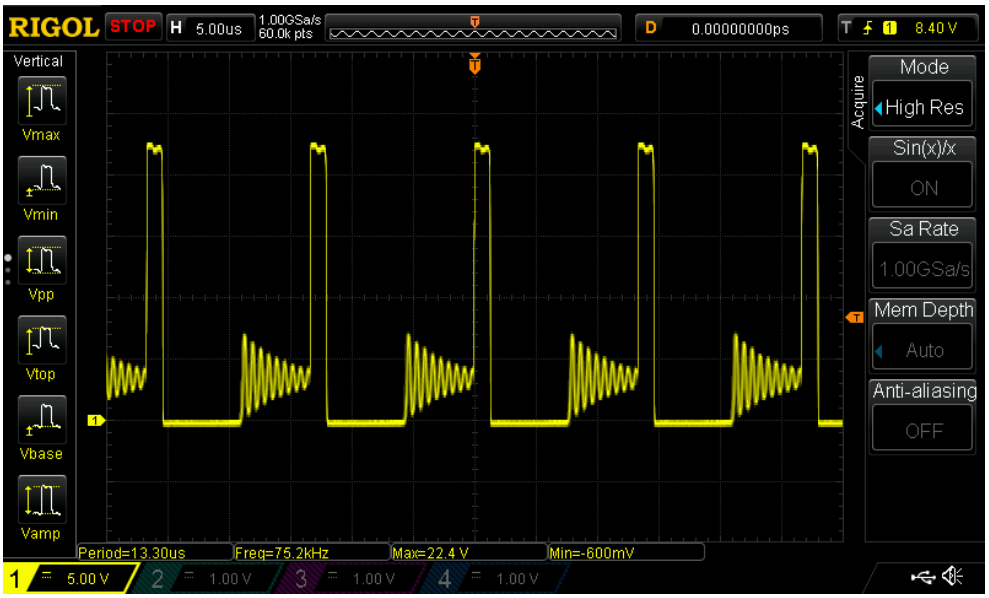
Obrázek 4.5: Schéma k měření 4.6 – měřené místo je označeno červenou šipkou – rezistor R16 s hodnotou *?* není na desce přiletován a po přerušení paralelní cesty může být přidán (vhodné pro testování různých hodnot)



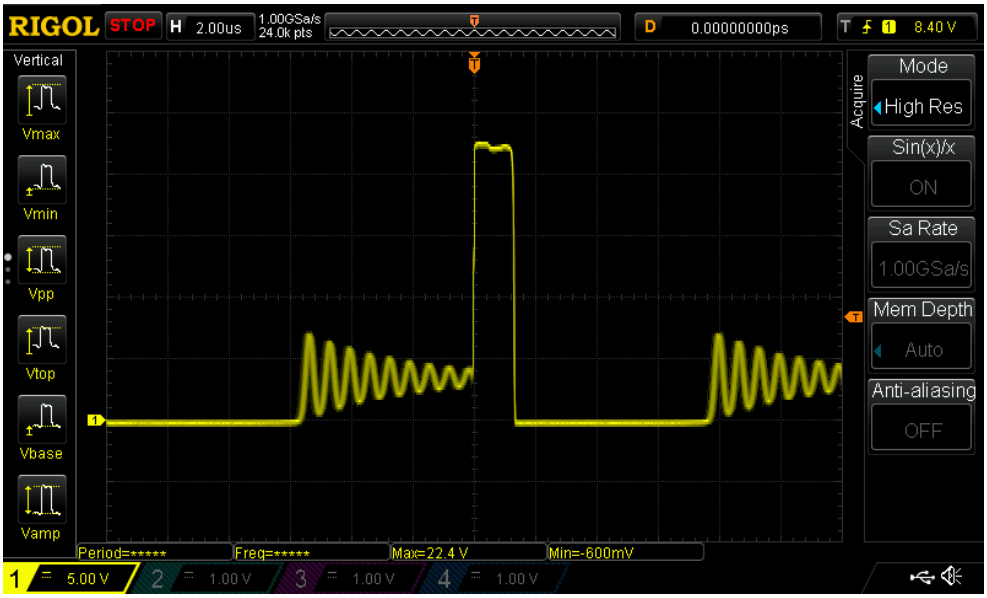
Obrázek 4.6: Měření napětí na oscilátoru pro W5500 – průběh signálu měřen mezi čipem W5500 a krystalem – stabilní frekvence 25 MHz, správný průběh



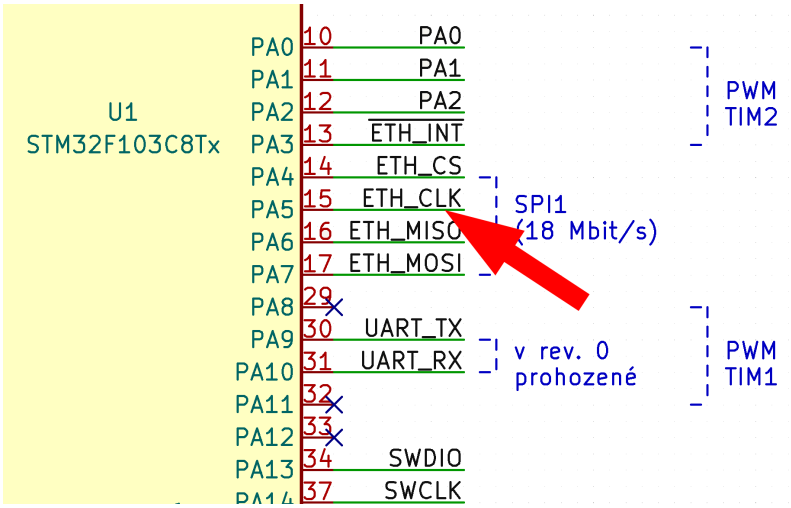
Obrázek 4.7: Schéma k měřením 4.8 a 4.9 – měřené místo je označeno červenou šipkou



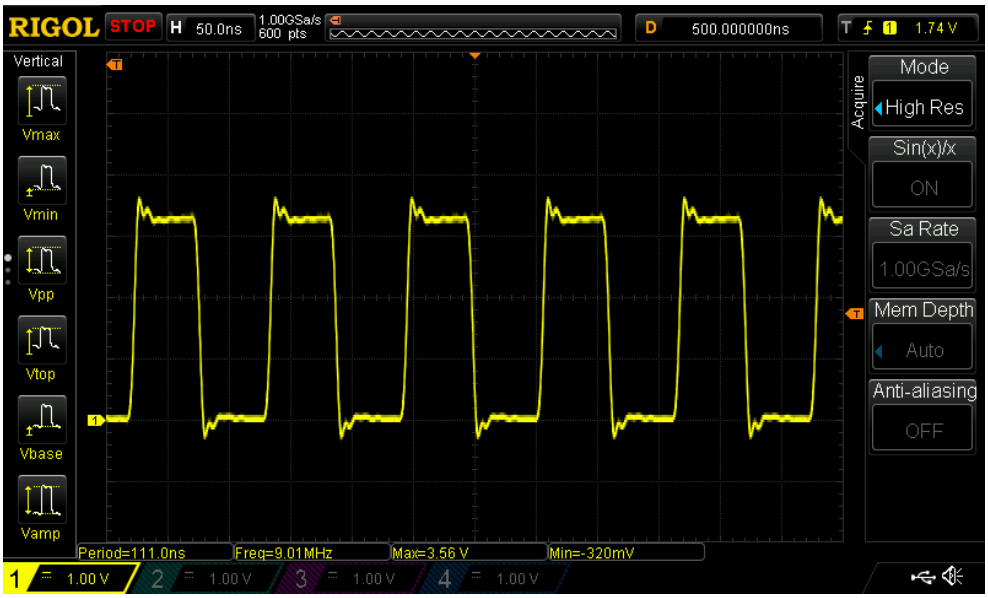
Obrázek 4.8: Měření napětí na záporném pólu diody SK33 (D2) ve zdroji – tedy v tomto případě na straně nepřipojené k zemi (fyzické zapojení z obrázku 2.4) – na obrázku lze pozorovat zatížení zdroje



Obrázek 4.9: Měření napětí na diodě ve zdroji – detail – přiblížení jednoho cyklu při spínání zdroje



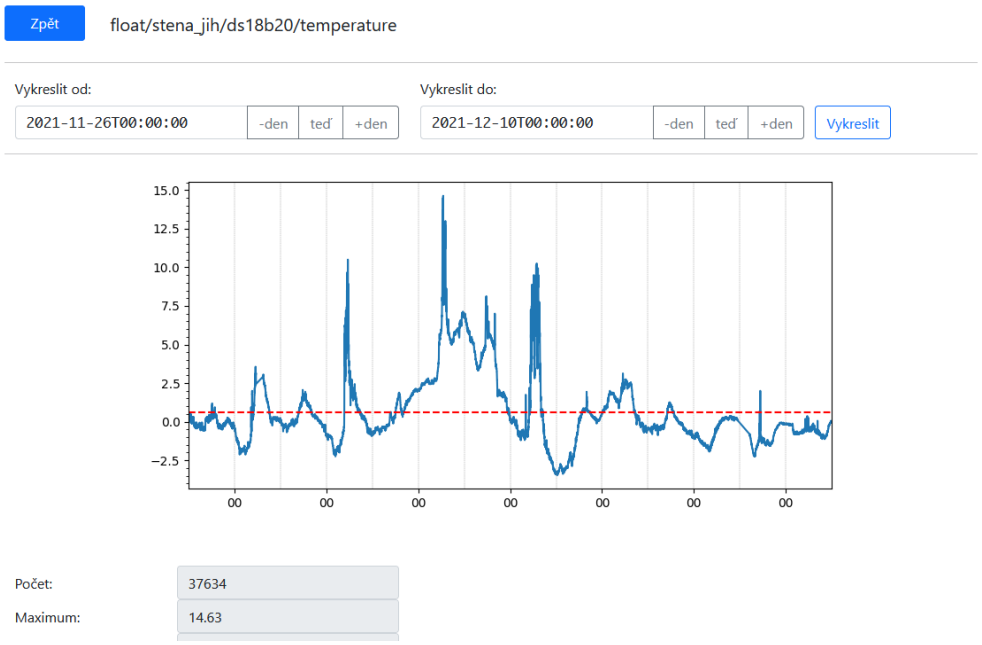
Obrázek 4.10: Schéma k měření 4.11 – měřené místo je označeno červenou šipkou



Obrázek 4.11: Měření hodinového signálu na komunikační sběrnici SPI – signál je měřen mezi procesorem STM32 a W5500 – na obrázku můžeme pozorovat drobné zákmity, nepřekračují maximální dovolené napětí na pinu a nezpůsobují žádné problémy při komunikaci, ovšem pokud by v budoucnu došlo ke zvýšení komunikační frekvence, bylo by dobré tyto konkrétní piny přeměřit

4.3 Stará a nová verze webového rozhraní (frontend)

K testování uživatelského rozhraní jsem jako testovací subjekt využil své rodiče. Nejvíce poznámek měl dlouhodobý uživatel chytrého domu, *pan otec*. Byl trochu zaskočen změnou designu – došlo ke sjednocení, aby podoba více odpovídala šabloně *Bootstrap 5* – což oproti staré verzi výrazně změnilo vzhled. Při zkoušení nového rozhraní ocenil hlavně rychlost vykreslování grafů (díky rychlejší databázi a cachování grafů). Dále byl do systému přidán přístup ke grafům za delší čas.



Obrázek 4.12: Nové uživatelské rozhraní pro vykreslování detailních grafů – ve vrchní části obrázku je vidět ovládací rozhraní pro nastavení vykreslovaného časového období, červená čára v grafu reprezentuje průměrnou hodnotu z (resamplovaných) dat, každá půlnoc je označena svislou šedivou čárou, pod grafem je umístěn výstup běžných statistických dat (pro přehlednost oříznuto)

Závěr

Cílem práce byl návrh a vývoj systému pro chytrý dům v souladu s myšlenkami Průmyslu 4.0. Před samotnou fází návrhu bylo nutné provést analýzu dostupných možností ohledně využití internetu, vývoje plošných spojů, nástrojů pro vývoj backendu a frontendu. Jelikož práce volně navazuje na mou původní bakalářskou práci „Návrh a implementace chytrého domu“, byla část této kapitoly věnována informacím o původním projektu, který posloužil jako odrazový můstek pro definování nových požadavků a možných zlepšení. Na základě obsáhlé analýzy bylo přistoupeno k jednotlivým částem návrhu. Nejprve je věnován prostor návrhu infrastruktury systému a jeho celkovému rozložení. Velkou část pak zabírá proces návrhu samotného plošného spoje, kde jsou představeny použité výpočty, zdroj napájení a další komponenty začleněné na desce. Závěr kapitoly doplňuje návrh vylepšení původní databáze s využitím nového schématu, které se lépe hodí pro účely aplikace. Na základě návrhu byla následně provedena implementace jednotlivých hardwarových a softwarových částí systému. Jmenovitě jde o síťovou architekturu, tovární výrobu plošného spoje a vyhodnocení její finanční stránky, dále pak instalace a zabezpečení firmware, použité knihovny a jejich využití při implementaci, backend, frontend a obecně zabezpečení systému.

V úvodu práce bylo nastíněno, že motivací pro celý projekt bylo hned několik. Chtěl jsem pracovat s vestavěnými systémy a navázat na svou bakalářskou práci. Ačkoliv Chytrý dům z původní práce fungoval skvěle, rozhodl jsem se ho vylepšit, neboť použité technologie zdaleka nepatřily k těm nejnovějším. Tato skutečnost mi dala podnět k tomu, abych se tímto tématem zabýval. Nakonec významným bodem mé motivace byla možnost práce na reálném projektu pro mou nově vzniklou firmu Kyvit s.r.o., kterou jsem spolu se svým kamarádem *Ing. Martinem Vitouškem* nedávno založil.

Jak to tak bývá, jednoduché věci se začaly zesložiťovat v momentě, kdy jsem

na nich začal pracovat. První bodem byl plán změn na původním systému. Předpokládal jsem, že bude možné jednotlivé části starého řešení pouze vylepšit, což jak se ukázalo, nebyla nakonec úplně pravda. Nakonec jsem dospěl k závěru, že bude lepší kompletně přepsat celý systém tak, aby nová implementace byla co nejčistší a netáhla si onu pověstnou „kouli na noze“. Věřím, že nová implementace je skutečně lepší a pro budoucí vývoj bude rozhodně přívětivější.

Bohužel všechno neprobíhalo hladce a práce se neobešla bez komplikací. Během testování vyrobených plošných spojů došlo z důvodu mé nepozornosti k poškození jednoho prototypu. Poškození bylo nevratné. Naštěstí ostatní desky vyvázly v pořádku. Nicméně s deskami plošných spojů se pojí další problémy, které provázely celkový průběh práce. V první řadě jejich samotné navržení mi zabralo více času než jsem předpokládal a současně mě brzdila starší verze programu KiCad, kterou jsem na začátku neprozřetelně zvolil a později již nebyl čas toto rozhodnutí měnit. Aby toho nebylo málo, tak jsem na závěr doplatil na současnou situaci ohledně výrobců elektroniky. Objednané desky z důvodu nedostatku součástek čekaly na jejich výrobu, a poté putovaly do České republiky mnohem déle, než bývá zvykem.

Naštěstí i přes všechny zmíněné útrapy se mi podařilo práci dokončit. Rád bych práci věnoval více prostoru, abych mohl rozvinout i jiné zajímavé části projektu, ale bohužel návrh a vyhotovení plošného spoje a další problémy mi zabraly daleko více času než jsem očekával, a tudíž na přidávání dalších částí již nezbyl čas. Na projektu však plánuji i nadále pracovat v rámci vývoje v naší firmě. Věřím, že na konci této cesty spatří světlo světa nový, skvělý a užitečný produkt, který zaujme své místo na trhu.

Literatura

1. KYZEK, Jakub. *Návrh a implementace chytrého domu*. Praha, 2019. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií.
2. *ATmega328P DATASHEET* [online]. 2015. 7810D–AVR [cit. 2019-05-12]. Dostupné z: `http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf`.
3. *The RS-485 Design Guide* [online] [cit. 2022-04-04]. Dostupné z: `https://www.ti.com/lit/an/slla272d/slla272d.pdf`. Application Report.
4. *Flask* [online] [cit. 2022-04-04]. Dostupné z: `https://flask.palletsprojects.com/en/2.1.x/`.
5. *PostgreSQL: Data Types* [online] [cit. 2022-04-04]. Dostupné z: `https://www.postgresql.org/`.
6. *Debian: Reasons to use Debian* [online] [cit. 2022-04-04]. Dostupné z: `https://www.debian.org/intro/why_debian`.
7. *Wireshark* [online] [cit. 2022-04-04]. Dostupné z: `https://www.wireshark.org`.
8. OASIS. *MQTT Version 5.0*. 2019. Dostupné také z: `https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.pdf`. Standard.
9. *BSON: Specification* [online]. Verze 1.1 [cit. 2022-04-28]. Dostupné z: `https://bsonspec.org/spec.html`.
10. *MessagePack specification* [online] [cit. 2022-04-28]. Dostupné z: `https://github.com/msgpack/msgpack/blob/master/spec.md`.
11. *About KiCad* [online] [cit. 2022-04-28]. Dostupné z: `https://www.kicad.org/about/kicad/`.

12. *STM32F103x8 DATASHEET* [online]. 2022. DS5319 Rev 18 [cit. 2022-04-03]. Dostupné z: https://www.st.com/resource/en/datasheet/stm32f103c8.pdf.

13. *W5500 Datasheet* [online]. 2019. Verze 1.0.9 [cit. 2022-04-03]. Dostupné z: https://docs.wiznet.io/img/products/w5500/w5500_ds_v109e.pdf.

14. *W5500* [online] [cit. 2022-04-03]. Dostupné z: https://www.wiznet.io/product-item/w5500/.

15. *FastAPI* [online] [cit. 2022-04-04]. Dostupné z: https://fastapi.tiangolo.com/.

16. *WireGuard* [online] [cit. 2022-04-27]. Dostupné z: https://www.wireguard.com/.

17. *XL1509 Datasheet* [online] [cit. 2022-04-10]. Dostupné z: https://datasheet.lcsc.com/lcsc/1809200015_XLSEMI-XL1509-3-3E1_C74193.pdf.

18. *Getting started with STM32F10xxx hardware development*. 2011. Verze 7. Dostupné také z: https://www.st.com/resource/en/application_note/cd00164185-getting-started-with-stm32f10xxx-hardware-development-stmicroelectronics.pdf. Application note. STMicroelectronics.

19. *Oscillator design guide for STM8AF/AL/S, STM32 MCUs and MPUs*. 2021. Verze 15. Dostupné také z: https://www.st.com/resource/en/application_note/an2867-oscillator-design-guide-for-stm8afals-stm32-mcus-and-mpus-stmicroelectronics.pdf. Application note. STMicroelectronics.

20. *HY931147C* [online] [cit. 2022-04-28]. Dostupné z: https://datasheet.lcsc.com/lcsc/1811141115_HANRUN-Zhongshan-HanRun-Elec-HY931147C_C91754.pdf.

21. *PostgreSQL: Data Types* [online] [cit. 2022-04-04]. Dostupné z: https://www.postgresql.org/docs/current/datatype.html.

22. *PostgreSQL: Indexes* [online] [cit. 2022-04-04]. Dostupné z: https://www.postgresql.org/docs/current/indexes.html.

23. *JLCPCB: Capabilities* [online] [cit. 2022-04-28]. Dostupné z: https://jlcpcb.com/capabilities/Capabilities.

24. *What is Industry 4.0?* [online] [cit. 2022-04-04]. Dostupné z: https://www.ibm.com/topics/industry-4-0.

25. SETRAPA, Pavel. *IPv6: Internetový protokol verze 6*. CZ.NIC, 2019. ISBN 978-80-88168-46-1. Dostupné také z: https://knihy.nic.cz/files/edice/IPv6-2019.pdf.

Dodatek A

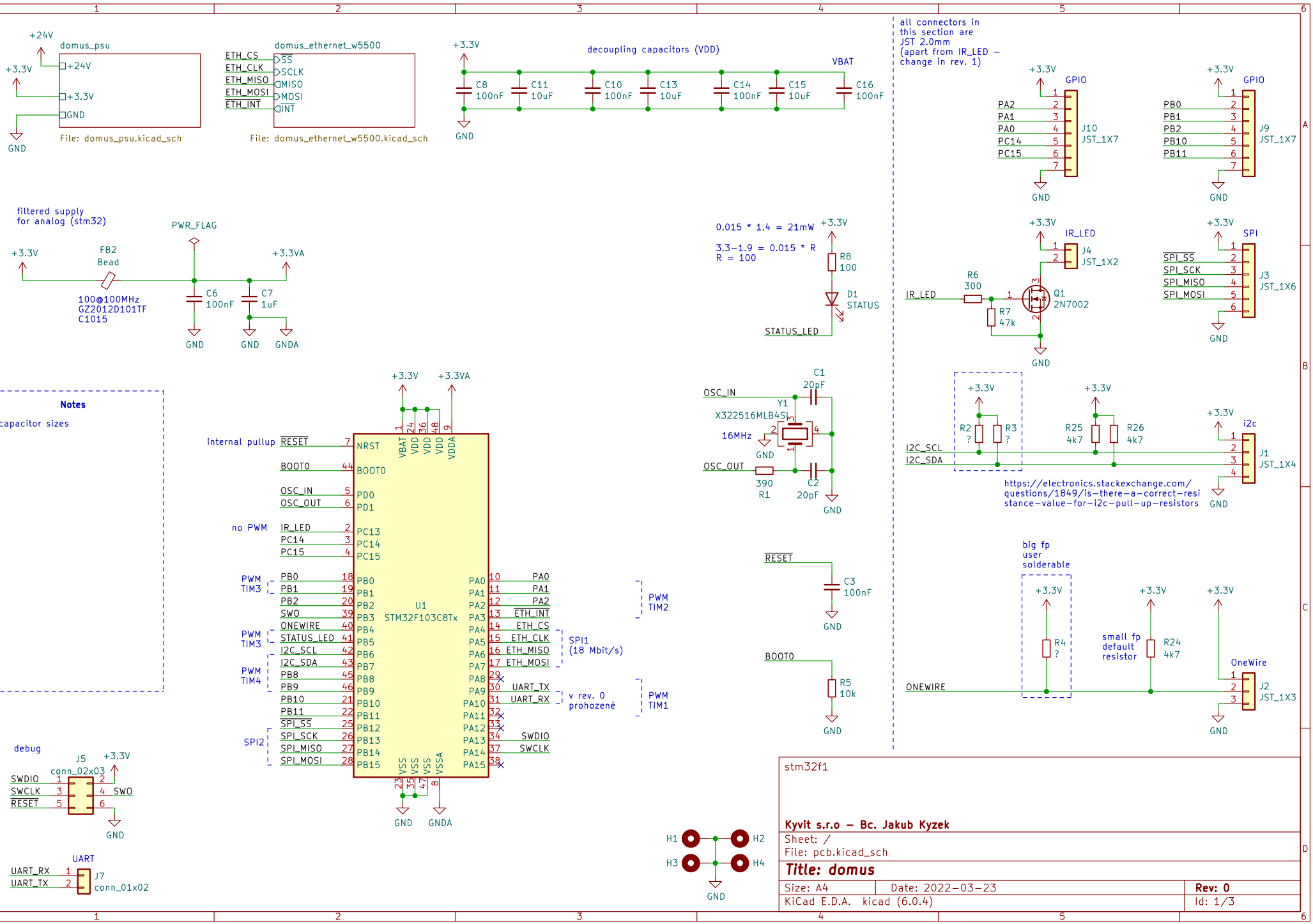
Seznam použitých zkratek

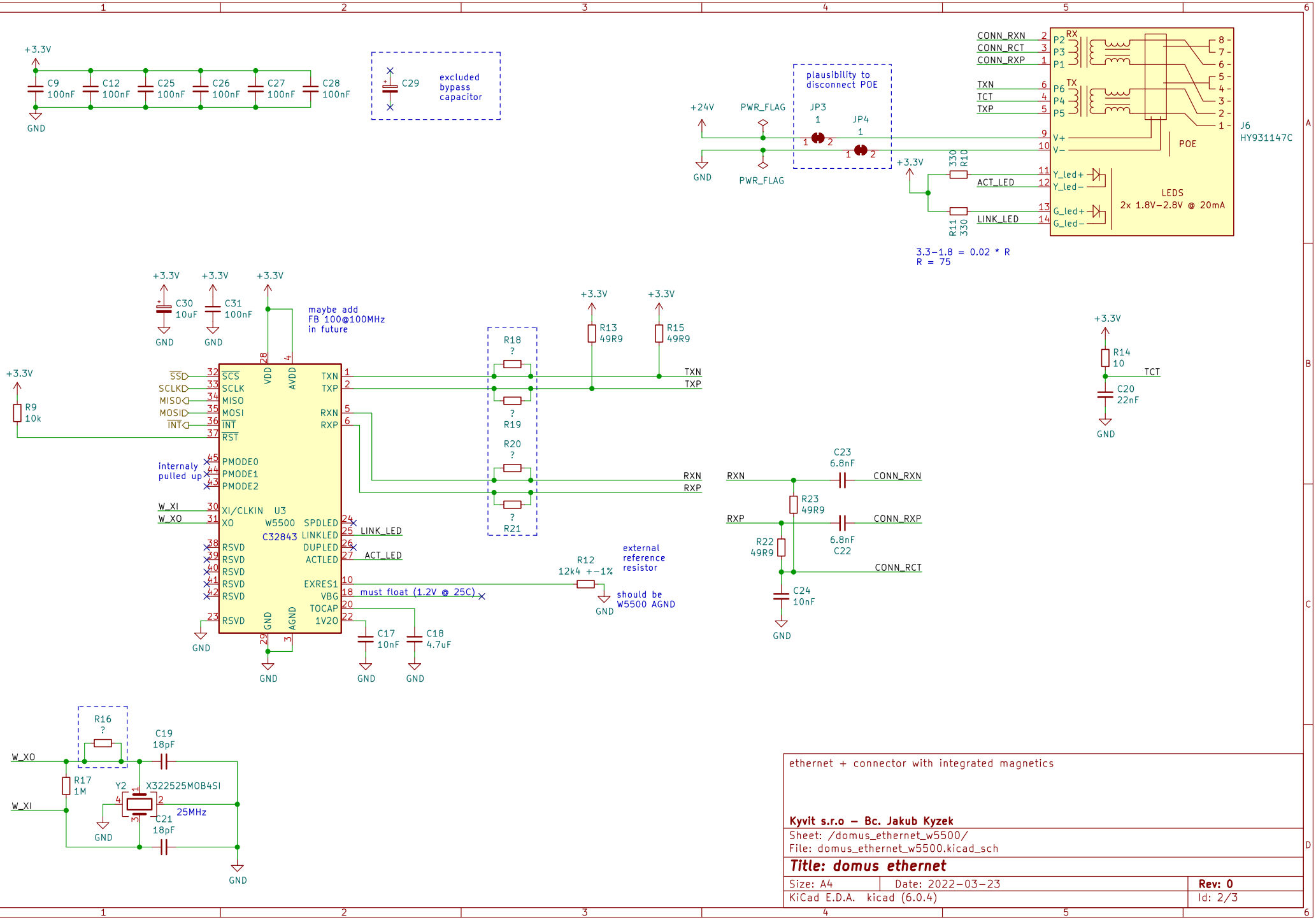
BRIN Block Range INdexes
BSON Binary JSON
ECC Error correction code
GPIO General-purpose input/output
HTTP Hypertext Transfer Protocol
HTTPS Hypertext Transfer Protocol Secure
IoT Internet of Things
JSON JavaScript Object Notation
NTP Network Time Protocol
NVMe NVM Express
PoE Power over Ethernet
RAM Random-access memory
SSD Solid-state drive
SSL Secure Sockets Layer
XML Extensible markup language

Dodatek B

Schémata navržené desky

Na následujících stránkách se nachází schéma navrhnuté desky. Jako první se tam nalézá stránka se schématem okolo procesoru, vstupně-výstupních konektorů a ostatních součástek pro běh procesoru. Na druhé stránce je schéma „internetové konektivity“ – konektor RJ45 a čip W5500 s potřebnými součástkami. Strana 3 obsahuje napájecí zdroj.


+24V
domus_psu
+3.3V
+24V
+3.3V
GND
GND
File: domus_psu.kicad_sch
domus_ethernet_w5500
ETH_CS
ETH_CLK
ETH_MISO
ETH_MOSI
ETH_INT
SS
SCLK
MISO
MOSI
INT
File: domus_ethernet_w5500.kicad_sch
+3.3V
decoupling capacitors (VDD)
VBAT
C8 100nF
C11 10uF
C10 100nF
C13 10uF
C14 100nF
C15 10uF
C16 100nF
GND
filtered supply
for analog (stm32)
PWR_FLAG
+3.3V
FB2
Bead
100@100MHz
GZ2012D101TF
C1015
C6 100nF
C7 1uF
+3.3VA
GND
GND
GNDA
0.015 * 1.4 = 21mW
3.3-1.9 = 0.015 * R
R = 100
+3.3V
R8 100
D1 STATUS
STATUS_LED
Notes
- capacitor sizes
+3.3V
+3.3VA
internal pullup
RESET
BOOT0
OSC_IN
OSC_OUT
no PWM
IR_LED
PC14
PC15
PWM TIM3
PB0
PB1
PB2
SWO
PWM TIM3
ONEWIRE
STATUS_LED
PWM TIM3
I2C_SCL
PWM TIM4
I2C_SDA
PB8
PB9
PB10
PB11
SPI2
SPI_SS
SPI_SCK
SPI_MISO
SPI_MOSI
U1
STM32F103C8Tx
PA0
PA1
PA2
ETH_INT
ETH_CS
ETH_CLK
ETH_MISO
ETH_MOSI
SPI1 (18 Mbit/s)
UART_TX
UART_RX
v rev. 0 prohozené
SWDIO
SWCLK
PWM TIM2
PWM TIM1
GND
GNDA
OSC_IN
OSC_OUT
Y1
X322516MLB4SI
16MHz
C1 20pF
C2 20pF
390
R1
GND
RESET
C3 100nF
GND
BOOT0
R5 10k
GND
debug
J5
conn_02x03
SWDIO
SWCLK
RESET
SWO
+3.3V
GND
UART
UART_RX
UART_TX
J7
conn_01x02
H1
H2
H3
H4
GND
all connectors in this section are JST 2.0mm (apart from IR_LED - change in rev. 1)
+3.3V
GPIO
PA2
PA1
PA0
PC14
PC15
J10
JST_1X7
GND
+3.3V
GPIO
PB0
PB1
PB2
PB10
PB11
J9
JST_1X7
GND
+3.3V
IR_LED
J4
JST_1X2
IR_LED
R6 300
R7 47k
Q1 2N7002
GND
+3.3V
SPI
SPI_SS
SPI_SCK
SPI_MISO
SPI_MOSI
J3
JST_1X6
GND
+3.3V
+3.3V
R2 ?
R3 ?
+3.3V
R25 4k7
R26 4k7
+3.3V
i2c
I2C_SCL
I2C_SDA
J1
JST_1X4
GND
https://electronics.stackexchange.com/questions/1849/is-there-a-correct-resistance-value-for-i2c-pull-up-resistors
big fp user solderable
+3.3V
R4 ?
+3.3V
small fp default resistor
R24 4k7
+3.3V
OneWire
ONEWIRE
J2
JST_1X3
GND
stm32f1
Kyvit s.r.o - Bc. Jakub Kyzek
Sheet: /
File: pcb.kicad_sch
Title: domus
Size: A4
Date: 2022-03-23
Rev: 0
KiCad E.D.A. kicad (6.0.4)
Id: 1/3



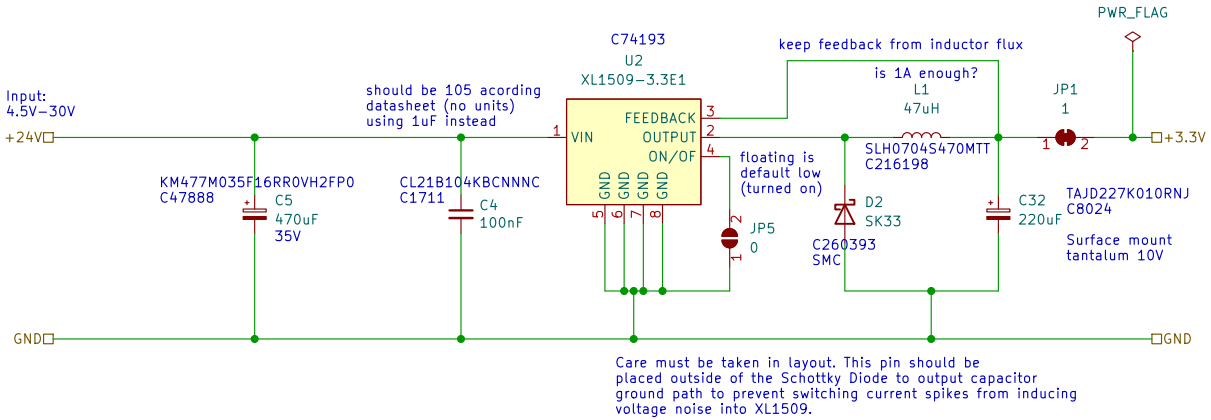
ethernet + connector with integrated magnetics

Kyvit s.r.o – Bc. Jakub Kyzek

Sheet: /domus_ethernet_w5500/
File: domus_ethernet_w5500.kicad_sch

Title: domus ethernet

Size: A4	Date: 2022-03-23	**Rev: 0**
KiCad E.D.A. kicad (6.0.4)		Id: 2/3



Kyvit s.r.o – Bc. Jakub Kyzek		
Sheet: /domus_psu/		
File: domus_psu.kicad_sch		
Title: domus power supply		
Size: A4	Date: 2022-03-16	**Rev: 0**
KiCad E.D.A. kicad (6.0.4)		Id: 3/3

Dodatek C

Obsah přiložené paměťové karty

- `README.txt` ..... stručný popis obsahu SD karty
- `zadani.pdf` ..... zadání práce ve formátu PDF
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `DP_Kyzek_Jakub_2022.pdf` ..... text práce ve formátu PDF