Bakalářská práce

České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Sada nástrojů pro řízení modelů světelné instalace LINKY

Matěj Mužátko

Vedoucí práce: Ing. Roman Berka, Ph.D.
Srpen 2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Mužátko** Jméno: **Matěj** Osobní číslo: **483809**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Sada nástrojů pro řízení modelů světelné instalace LINKY

Název bakalářské práce anglicky:

A set of tools for controlling models of LINKY light installation

Pokyny pro vypracování:

Na základě poznatků získaných v rámci projektu v předmětu B6B36PRO navrhněte univerzální softwarový balíček, který poslouží jako základ modelů světelné fasády Linky FEL ČVUT. Modely mohou být různých velikostí i hardwarového vybavení.
Balíček musí být použitelný na mikrokontrolérech STM32 a platformě Arduino.
V konceptu počítejte s internetovým připojením pro získání informací o aktuálním stavu světelné instalace. Ke komunikaci s instalací využijte veřejně dostupného rozhraní LinkyAPI na serveru https://linky.fel.cvut.cz.
Navrhněte více variant implementace HTTPS klienta pro vybranou platformu. Zvolte nejvhodnější variantu z hlediska náročnosti na operační paměť a implementujte ji.
Definujte rozhraní, které bude balíček poskytovat a zdokumentujte ho.
Vlastnosti návrhu ověřte implementací zvolené varianty a zhodnoťte případné nedostatky a případná rizika použití balíčku. Navrhněte a zdůvodněte sadu testovacích scénářů a následně po jejich realizaci zhodnoťte dosažené výsledky.

Seznam doporučené literatury:

1. Wang, Guijun, and Casey K. Fung. „Architecture paradigms and their influences and impacts on component-based software systems.“, 37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of the. IEEE, 2004, pp. 10 pp.-, doi: 10.1109/HICSS.2004.1265643.
2. Škvor, M., „Sběr dat ze senzorů pro monitorování budov.“, Bakalářská práce, 2015, České vysoké učení technické v Praze. https://core.ac.uk/download/pdf/47181522.pdf, 15.1.2021.
3. Fryč, J.: „Popis rozhraní pro komunikaci s instalací LINKY“. https://linky.fel.cvut.cz/Api. ČVUT FEL, 2018.
4. HYBLER, Jakub. „Světelný design v kostce – Část 24: Světelná fasáda –projekt Linky“, Světlo: časopis pro světelnou techniku a osvětlování. 2016,1998-, 2016(3), 5. ISSN 1212-0812.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Roman Berka, Ph.D., Katedra počítačové grafiky a interakce

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

Datum zadání bakalářské práce: **15.02.2021** Termín odevzdání bakalářské práce: **13.08.2021**

Platnost zadání bakalářské práce: **19.02.2023**

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

III. PŘEVZETÍ ZADÁNÍ

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

. _____
Datum převzetí zadání

_____
Podpis studenta

Poděkování

Děkuji svým rodičům za podporu nejen při tvorbě práce, ale i během celého studia. Dále děkuji vedoucímu práce, panu Ing. Romanu Berkovi, za možnost pracovat na zajímavém projektu.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a v souladu s Metodickým pokynem o dodržování etických principů pro vypracování závěrečných prací, a že jsem uvedl všechny použité informační zdroje.

V Praze, 13. srpna 2021

Matěj Mužátko

Abstrakt

Tato bakalářská práce se zabývá problematikou tvorby fyzických modelů světelné instalace Linky, tedy pěti světelných pásů na fasádě dejvické budovy FEL ČVUT. Cílem práce bylo vytvořit koncept, pomocí kterého bude možné vytvářet řídící software pro modely, které jsou po vizuální stránce navrhované studenty Fakulty architektury ČVUT. Po analýze problému byl vytvořen vysokoúrovňový návrh, byly implementovány dvě varianty modelu a síťové rozšíření pro získání dat aktuálního stavu skutečné instalace Linky v Dejvicích a jejich zaslání do modelu. Testování ukázalo, že nejužitečnějším výstupem této práce je implementační varianta B, pomocí které lze přehrát jakoukoliv animaci až do 850 snímků za sekundu. Síťové rozšíření je na reálné využití příliš pomalé, ale byly navrženy změny celého systému (včetně serverové strany), které usnadní jeho budoucí vývoj.

Klíčová slova: Model, světelná instalace, Linky FEL ČVUT, adresovatelné LED, WS2812B, ESP8266, STM32, Arduino, PWM, DMA

Vedoucí práce: Ing. Roman Berka, Ph.D.
Technická 1902/2,
16000 Praha,
místnost: H1-24a

Abstract

This bachelor thesis focuses on the topic of creation of physical models of Linky installation. Linky installation consists of 5 vertical light panels situated on CTU FEE building in Prague. The main goal of this thesis was to create a high level concept which would be used to create software controlling Linky physical models, designed by students of Faculty of Architecture CTU. The high-level design was created after analysis. Two variants of software for the physical model were implemented. Network extension was implemented in order to be able to get the current state of real Linky installation and send it to the model. Testing has shown the good usability of variant B, which can be used to play animations from SD card up to 850 frames per second. On the other hand, the network extension is too slow for real usage. The suggestions have been raised for changes of the whole system (including server side) to make the future development of network version easier.

Keywords: Model, light installation, Linky FEE CTU, addresable LED, WS2812B, ESP8266, STM32, Arduino, PWM, DMA

Title translation: A set of tools for controlling models of LINKY light installation

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

1.1 Projekt Linky

Světelná instalace Linky zdobí fasádu dejvické budovy Fakulty elektrotechnické ČVUT v Praze od roku 2016 [1]. Jedná se o 5 vertikálních, 17 m vysokých, světelných pásů – každý z nich se totiž skládá ze 17 LED svítidel Elar o délce 1 m [2].

Instalace může zobrazovat prostou animaci, ale je vytvořena tak, aby mohla např. reagovat na nějaký aktuální stav (znečištění ovzduší, hluk) [3] nebo interagovat s uživatelem (hry) [1]. V současnosti je možné instalaci ovládat přímo z Institutu intermédií (pouze členové týmu Linek), pomocí webového API (lidé s účtem na webovém portálu Linek a povolením pro ovládání skutečné fasády) [4], nebo volbou některého z hotových programů webové aplikace (pro veřejnost, bez účtu) [3].

1.2 Motivace

Projekt Linky je již od svého počátku společným projektem Fakulty elektrotechnické a Fakulty architektury ČVUT v Praze [3]. Tato spolupráce se uplatnila při realizaci projektu, ale pokračuje i při jeho rozšiřování. Někteří studenti studijního programu Design na Fakultě architektury ČVUT navrhují různé výrobky s motivem světelné instalace Linky. Tyto výrobky jsou koncipovány jako funkční modely instalace Linky. Mohou mít různé rozměry a počty aktivních světelných bodů, různé podoby. Může se jednat o designové doplňky do domácnosti, ale fantazii se zde meze nekladou – může se jednat např. o svítící tričko s motivem instalace Linky.

Motivací pro tvorbu této práce je skutečnost, že zatím nebyla vytvořena metodika vývoje funkční softwarové a hardwarové části těchto výrobků. Neproběhla analýza, jaké technologie využít a jakým způsobem je využít. Bez tohoto kroku není možné návrhy studentů FA ČVUT dále rozvíjet a případně některé z těchto výrobků produkovat a propagovat tím nejen instalaci Linky, ale i celou Fakultu elektrotechnickou ČVUT v Praze.

1.3 Cíle práce

Cílem této práce je analyzovat možnosti řešení fyzických modelů instalace Linky. Bude vytvořen koncept pro tvorbu takovýchto modelů ve formě vysokoúrovňového návrhu. Kvalita vysokoúrovňového návrhu bude ověřena implementací dvou variant modelu a implementací síťového rozšíření. Bude dbáno na přepoužívání kódu, který může být společný pro obě dvě varianty modelu, i pro síťové rozšíření.

Obrázek 1.1: Světelná instalace Linky na fasádě FEL ČVUT [5].

Obrázek 1.2: Model světelné instalace z této práce.

Kapitola 2
Návrh řešení

Tato kapitola popisuje návrh řešení software a hardware pro fyzické modely světelné instalace Linky. Zejména důležitý je vysokoúrovňový návrh (sekce 2.1 a 2.4), tj. společná kostra implementační varianty A, B i síťového rozšíření. Také jsou porovnány různé způsoby ovládání chytrých LED WS2812B a je popsáno, jaké způsoby jsou v každé variantě využity a proč (sekce 2.5).

2.1 Přístup

Systém je navržen pomocí komponent, které dodržují SOLID zásady návrhu. Takový návrh obsahuje komponenty, které jsou od sebe dobře oddělené, tj. každá má určitou zodpovědnost (*Single-responsibility princip ze SOLID* [6]). Návrh s oddělenými komponentami poskytne tvůrci modelu flexibilitu, kdy nebude nucen použít implementaci jako celek, ale bude schopen libovolnou komponentu implementovat jinak. Také bude možné přidat systému další funckionality přidáním dalších komponent, bez změny stávající funkcionality (*Open-closed princip ze SOLID* [6]).

2.2 Současný stav LinkyAPI

Na oficiálních stránkách projektu Linky je dostupný popis rozhraní LinkyAPI k ovládání internetového debuggeru, ale i skutečné světelné fasády a získání jejího aktuálního stavu [4]. Stručný popis rozhraní se nachází v Tabulce 2.1. Jsou v ní popsány všechny endpointy definované na oficiálních stránkách LinkyAPI [4]. Využití LinkyAPI je možné pouze pomocí HTTPS protokolu – po pokusu o navázání spojení prostým HTTP vrací LinkyAPI HTTP odpověď `301 Moved Permanently` a klient je přesměrován na HTTPS verzi API.

Endpoint vhodný pro získání dat aktuálně zobrazených na instalaci Linky je `DATASTATE`, který vrací aktuální snímek v JSON formátu [4]. Velikost odpovědi je vždy přibližně 11 kB (zjištěno pokusem), ale může se pokaždé lišit, jelikož se může lišit i počet cifer u hodnot jednotlivých barev.[1]

[1] Hodnoty každé barvy jsou v rozpětí 0 - 255 [4].

Metoda	Endpoint	Popis funkcionality
`POST`	`/TOKEN/CREATE`	Vytvoří uživateli token pro přehrávání na webu či skutečné fasádě.
`POST`	`/TOKEN/REFRESH`	Prodlouží platnost tokenu, pokud není ve frontě žádný jiný token.
`GET`	`/TOKEN/QUEUE`	Vrátí informace o aktuální frontě tokenů, včetně konce platnosti posledního tokenu ve frontě.
`POST`	`/PLAY`	Přehraje uživatelem zaslanou animaci.
`GET`	`/DATASTATE/PNG`	Vrátí současný stav na instalaci LINKY v png formátu.
`GET`	`/DATASTATE`	Vrátí současný stav na instalaci LINKY v json formátu.
`GET`	`/STATE`	Vrátí informace o aktuálním stavu instalace LINKY.

Všechny cesty jsou psány relativně k https://linky.fel.cvut.cz/Api/v1/.

Tabulka 2.1: Stručný popis rozhraní LinkyAPI [4].

Dokumentace API na oficiálních stránkách projektu Linky je matoucí. Endpointy jsou psány celé velkými písmeny, ale ve skutečnosti jsou case-sensitive a je třeba je volat dle konvence UpperCamelCase, tzn. první písmeno každého slova musí být velké (zjištěno pokusem). Také není zdokumentováno, kde se samotné API nachází, tato základní cesta (uvedena v Tabulce 2.1) byla zjištěna z ukázky kódu v bakalářské práci [7]. Opravit tyto drobné avšak velmi matoucí chyby v dokumentaci není náročné, stačí přidat několik vět do dokumentace [4]. Podle informací od členů týmu Linky ale na tomto API již nebude marněn čas, jelikož ho v brzké době nahradí nové jádro systému s pokročilejšími možnostmi [7] [8].

2.3 Budoucí podoba API

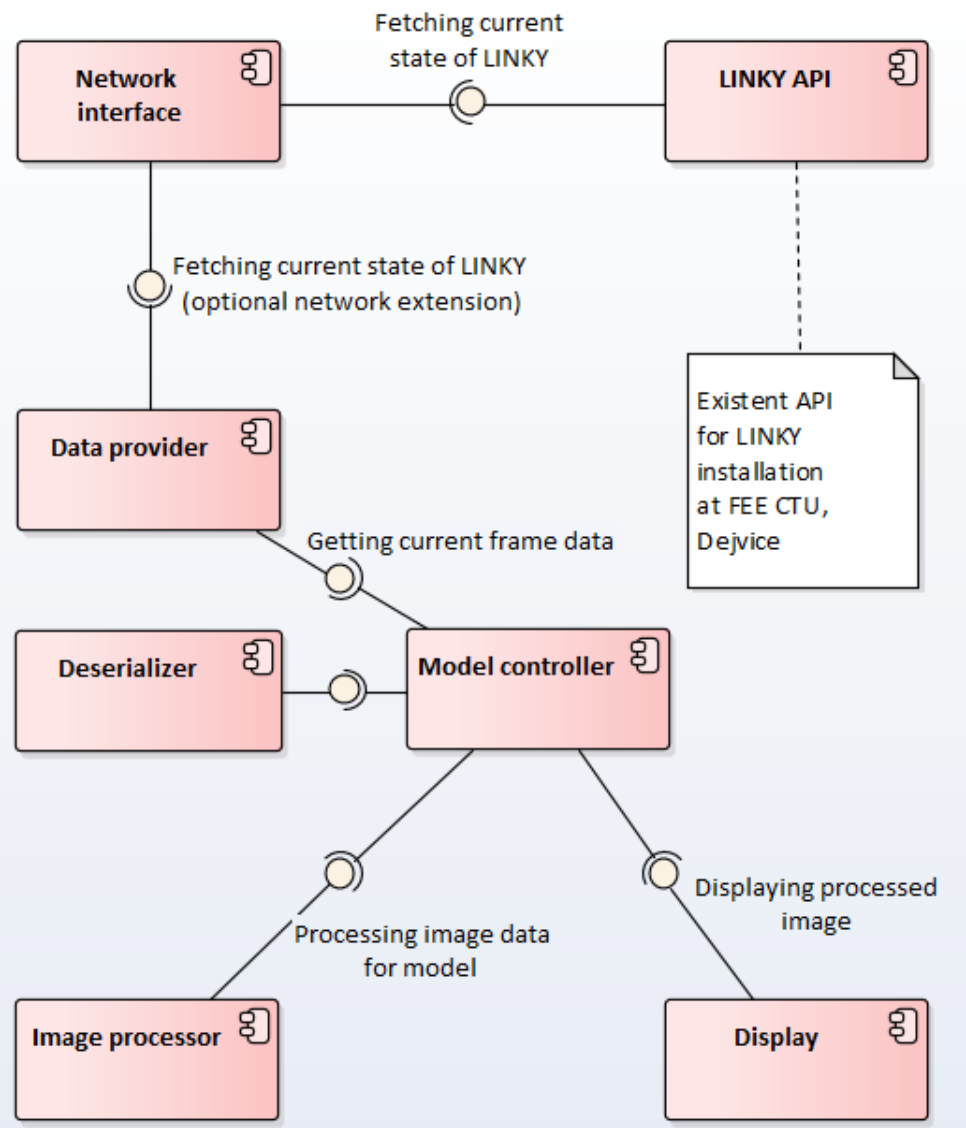
Jak bylo již zmíněno v předchozí sekci, v nedaleké době bude současné LinkyAPI vypnuto a bude nasazeno nové jádro systému Linky (nazváno LinkyCore [8]) a nová grafická knihovna s veřejným API [7]. API nového jádra bude sloužit především k administraci serverových aplikací a jiných dat [8]. API grafické knihovny bude hlavně umožňovat zobrazovat na Linky obraz určený geometrickými tvary, bez nutnosti vyplňovat jednotlivé pixely [7].

Součástí grafické knihovny je ale také koncept simulátoru, který může mít více instancí [7]. Instance simulátoru může být např. widget na webové stránce, který ale bude neustále spojený se serverem pomocí WebSocketu a bude získávat od serveru obrazová data [7]. Toto půjde v budoucnu využít i pro fyzické modely. Model bude možné brát jako simulátor, který bude v případě změny dostávat obrazová data od serveru po WebSocketu. Komunikace s modely tak bude efektivnější, než neustálé dotazování směrem k serveru

LinkyAPI implementované v této práci (podrobněji rozebráno v sekci 4.2.3).

2.4 Koncept

Součástí této práce jsou dvě implementační varianty modelu a síťové rozšíření. Tato sekce popisuje high-level návrh komponent a určení jejich zodpovědností.



Obrázek 2.1: Component diagram

Celkem se systém skládá z 5 softwarových komponent. Diagramy popisují jednu z možných konfigurací. Konfiguraci je ale možné v případě tvorby nové implementační varianty změnit, např. některé komponenty z celého procesu úplně vypustit. To může být nutné např. kvůli optimalisaci rychlosti či operační paměti. Na Obrázku 2.1 je možné vidět *Component diagram*, na

Obrázku 2.2 pak specifičtější *Class diagram*, který se ale stále snaží o zachování co možná největší nezávislosti na platformě v této fázi návrhu.

Nejdůležitější komponentou je *Model controller*. Využívá rozhraní ostatních komponent k získání, zpracování a zobrazení obrazových dat. Sama tvůrci poskytuje důležitou metodu `refresh()`, která zajišťuje aktualizaci stavu modelu, k čemuž koordinuje všechny ostatní komponenty.

2.4.1 Jednoprůchodové zpracování

Systém je navržen tak, aby jím data procházela po jednotlivých částech, přičemž vstup a výstup pro každou komponentu může být různě velký. Výhodou tohoto přístupu je, že data nemusí být získána všechna najednou. Je tak šetřeno operační pamětí, např. u síťové varianty nemusí být stažena všechna data pro instalaci Linky, jsou stahována po malých bufferech a zpracovávána průběžně. Byl vzat v potaz předpoklad, že modely instalace budou mít mnohem méně pixelů, než instalace Linky samotná.[2] Nevýhodou je méně přímočará implementace.

Jedná se o variantu návrhového vzoru *Iterátor* [9], komponenty disponují funkcemi `hasNext()` pro zjištění, zda jsou na jejich výstupu připravena další data, `next()` pro získání těchto dat, ale některé implementují i funkci `process()`, která naopak funguje jako vstup pro příchozí data ke zpracování (viz Class diagram na Obrázku 2.2). Komponenta *Network Interface* dodává JSON data po jednom znaku, *Deserializer* je přijímá a na výstupu vrací pixely, *Image Processor* přijímá tyto pixely a na výstupu vrací zpracované byty pro model. Komponenta *Display* má v sobě uloženou referenci na buffer hotových dat pro displej ve formě *LinkyModelViewData*, po zavolání funkce je zobrazí na fyzický model. Komponenty mohou pro uchování stavu ve funkcích využívat bufferů a flagů ve formě svých vlastních lokálních statických proměnných v případě jazyka C, nebo ve formě datových položek třídy v případě jazyka C++ [10].

2.5 Způsoby ovládání LED WS2812B

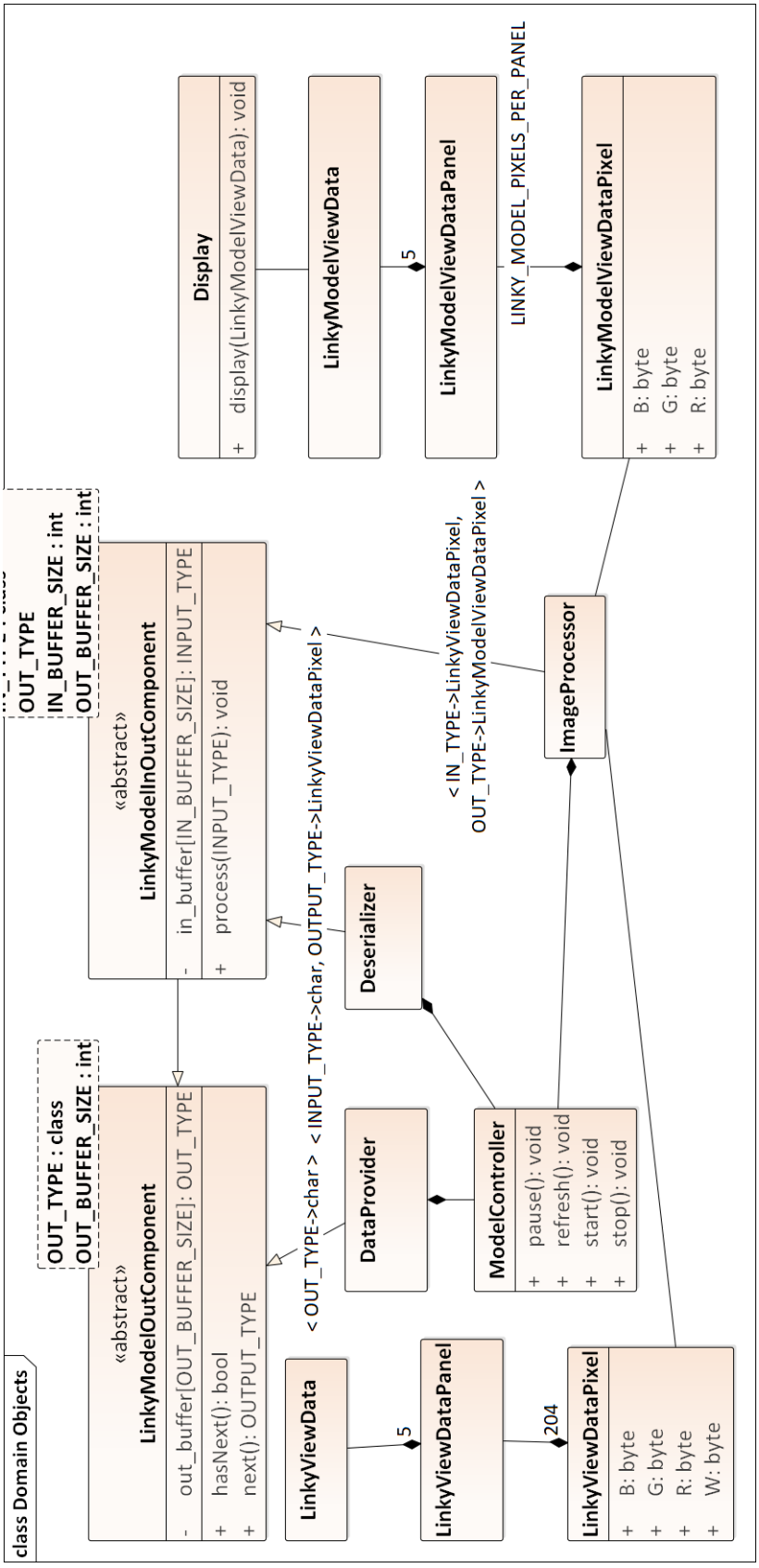
Tato sekce popisuje možné způsoby ovládání LED WS2812B, porovnání těchto způsobů a výběr vhodných způsobů pro obě implementační varianty realizované v této práci.

2.5.1 WS2812B a její protokol

WS2812B je elektronická součástka, která v pouzdře *SMD LED 5050*[3] obsahuje červenou, zelenou a modrou LED a řídící obvod, pomocí kterého lze měnit zářivý výkon (hodnoty intenzity 0-255) jednotlivých LED jediným datovým vstupem *DIN* pro všechny 3 LED [12]. Do WS2812B mohou být

[2] Např. model implementovaný v této práci má 170 světelných bodů, 6× méně než počet světel instalace Linky.

[3] Jedná se o typ LED čipu o velikosti 5.0 mm × 5.0mm [11].



Obrázek 2.2: Class diagram

odeslána po datech dané WS2812B data navíc pro další WS2812B [12]. Tato data jsou pak odeslána na datový výstup *DOUT*, na který může být připojena další WS2812B [12].

Obrazová data musí být do vstupu *DIN* přiváděna po bytech v pořadí barev G, R, B[4], přičemž nejprve se přenáší MSB (nejvýznamnějšího bitu) a bity jsou zakódované pomocí speciálního protokolu definovaného v datasheetu [12]. Kód každého bitu začíná úrovní HIGH po dobu $T_{x\mathrm{H}}$, následuje úroveň LOW po dobu $T_{x\mathrm{L}}$, přičemž x je bit 1, nebo bit 0, dle kódu. Čas odeslání jednoho bitu je vždy $T_{x\mathrm{H}} + T_{x\mathrm{L}} = 1.25$ µs v ideálním případě dle datasheetu [12]. Po odeslání všech bitů obrazových dat (jedné nebo více WS2812B) je zaslán RESET kód, který je realizován jako úroveň LOW minimálně po dobu T_{RESET} [12]. Hodnoty $T_{x\mathrm{H}}$, $T_{x\mathrm{L}}$ a T_{RESET} jsou z datasheetu [12] vypsané v Tabulce 2.2.

	Minimální	Ideální	Maximální
$T_{0\mathrm{H}}$	250	400	550
$T_{1\mathrm{H}}$	650	800	950
$T_{0\mathrm{L}}$	700	850	1000
$T_{1\mathrm{L}}$	300	450	600
T_{RESET}	50000	-	-

Tabulka 2.2: Tabulka hodnot WS2812B protokolu v ns dle datasheetu [12].

Na trhu je možné se v kategorii adresovatelných RGB LED založených na podobném principu setkat se součástkami od firmy Worldsemi (např. WS2811, WS2812 nebo právě WS2812B) [13]. Mezi další LED tohoto typu se řadí součástky výroby Shenzen LED Color, např. SK6812 [14]. WS2812B byla zvolena, jelikož ostatní LED pásky nalezené v době výběru měly příliš velké rozestupy mezi jednotlivými LED komponentami (hustota 60 LED/m), ale LED pásek s WS2812B bylo možné sehnat i s hustotou 144 LED/m. Díky velké hustotě LED komponent na metr lze dosáhnout vysokého rozlišení i u malého modelu, např. model v této práci má pásky vysoké 236 mm a obsahují 34 chytrých LED WS2812B.

2.5.2 Bit banging

Prvním způsobem je využití výstupních pinů (tedy GPIO) bez pokročilejších periferií ke generování digitálního signálu. V literatuře a na Internetu je pro tento způsob používán slangový výraz *Bit banging* [15] [16]. WS2812B protokol je prostý, je třeba implementovat 3 kódy [12], viz sekci 2.5.1. Jedná se tedy o nejpřímočařejší způsob ovládání, který ale přináší svá úskalí.

Hlavním problémem je nutnost odměřovat velmi krátké časové úseky, konkrétně u WS2812B v řádu stovek nanosekund [12]. Jednou z možností, jak krátkého zpoždění docílit, je využít skutečnosti, že má procesor určitou frekvenci a každá instrukce pak trvá určitý čas. Dále v tomto odstavci bude tento čas značen T_{INS}, požadovaný čas zpoždění bude značen T_{DELAY}. Pro dosažení požadované krátké prodlevy je pak nutné vykonat počet instrukcí

[4] 3 byty na jednu WS2812B, každý byte značí intenzitu dané barvy od 0 do 255

$t_{\mathrm{DELAY}}/T_{\mathrm{INS}}$. Kratších prodlev než T_{INS} touto metodou nelze dosáhnout. Jako vykonávané instrukce mohou být využity např. instrukce NOP vložené do C nebo C++ kódu pomocí inline assembleru, např. příkazem `asm("NOP");`, nebo je prodlevy určitého počtu instrukcí docíleno pomocí *cyklu for*, kde se pouze inkrementuje/dekrementuje proměnná, tedy např.

```
for (volatile int i = 0; i < n; i++);.
```

V případě cyklu for by měla být inkrementovaná proměnná označena jako `volatile`, aby kompilátor skutečně čtení a inkrementaci do programu zahrnul a nevypustil ji v rámci případné optimalisace [17]. Hodnotu n je potřeba zvolit tak, aby byla výsledkem kýžená prodleva. Lze ji zvolit např. pomocí datasheetu, případně ladit pomocí měření logickým analyzátorem na nějakém jednoduchém programu.

Při tvorbě prodlev tímto způsobem je vhodné vědět, jak vypadá program před sestavením assemblerem. To lze v případě gcc kompilátoru (dle dokumentace [18]) zjistit takto:

```
gcc -S program.c
```

Pomocí datasheetu používaného mikrokontroléru lze poté zkontrolovat délku instrukcí, zda dosahuje program správné prodlevy, případně upravit hodnotu n v cyklu for.

Ovládání chytrých LED pomocí GPIO a krátkých prodlev je výhodné z hlediska universality, lze použít jakýkoliv GPIO pin, nejedná se o periferii s pevně přiřazeným pinem či malou množinou pinů. Mezi nevýhody tohoto přístupu patří plýtvání procesorovým časem, který by šel využít k získání a zpracování obrazových dat, což by v případě této práce mohlo způsobovat problémy u vyššího počtu LED nebo při budoucím rozšiřování modelu o další funkcionality.

2.5.3 DMA

DMA je periferie CPU, která dle požadavků CPU zajišťuje přenos dat z místa A na místo B [19]. Místa A a B mohou být v principu adresy v paměti nebo např. registry [20]. Výhodou DMA je možnost souběhu přenosu dat s jinou činností CPU. U některých mikrokontrolérů je navíc možné nastavit přenos určitého počtu bytů (resp. wordů, dwordů), s případnou inkrementací zdrojové nebo cílové adresy po každém přenosu jednoho bytu (resp. wordu, dwordu) [21]. U některých mikrokontrolérů může DMA při přenosu komunikovat s CPU pomocí přerušení, např. v případě STM32F0 pomocí nastavitelných přerušení TC (*Transfer complete*, všechna data jsou přenesena) a HT (*Half transfer*, polovina dat je přenesena) [21]. U DMA může být také nastaveno, aby přenos probíhal cyklicky, bez nutnosti jej nastavovat při každém TC, např. u mikrokontroléru STM32F072RB [20].

Pro ovládání chytrých LED je možné využít DMA v kombinaci s nějakou další periferií, sloužící ke generování digitálního signálu odpovídajícího protokolu WS2812B (viz sekci 2.5.1). Obrazová data tak mohou být v nějaké

formě přenášena do registrů ovlivňujících průběh výstupního signálu, nebo na předem nastavené místo v RAM (záleží na generující periferii). Pro generování signálu je možné použít např. I^2S, SPI, PWM, pokud pro danou periferii mikrokontrolér nabízí použití s DMA. V následujících sekcích jsou tyto vybrané periferie popsány.

2.5.4 PWM

Pro ovládání chytrých LED WS2812B lze využít i PWM, neboli Pulse Width Modulation. PWM je druh modulace, kde je výstupem digitální signál o určité frekvenci, přičemž v každé periodě je po určitou dobu aktivní jedna úroveň (např. HIGH), zbytek periody je aktivní úroveň druhá (např. LOW) [22]. Poměr doby trvání první úrovně k celé periodě se nazývá střída, je udávána např. v procentech.

PWM má mnoho využití. Kromě možnosti využít PWM pro digitalisaci analogového signálu a jeho přenos, kdy se na straně přijímače pomocí dolnopropustného filtru objeví signál podobný původnímu analogovému signálu [22], je známé využití pro regulaci výkonu jakéhokoliv spotřebiče, který má pouze dva stavy – zapnuto a vypnuto. Jako příklad lze uvést regulaci zářivého výkonu LED, kterou ovlivňuje střída PWM [22]. Člověk totiž dle *Talbotova zákona* při dostatečně velké frekvenci střídání dvou odlišných hodnot jasu nevnímá blikání, ale vnímá střední hodnotu těchto jasů [22] [23, str. 753]. Dostatečně vysoká frekvence, kdy člověk přestává střídání hodnot vnímat jako blikání se nazývá *kritická frekvence splývání* [24, str. 65]. U lidí se kritická frekvence splývání pohybuje v desítkách Hz [24, str. 65]. Některá využití PWM vyžadují řádově vyšší frekvenci, např. generování digitálního signálu pro WS2812B vyžaduje frekvenci 800 kHz.

Základní myšlenka využití PWM pro ovládání WS2812B vychází z protokolu WS2812B (viz sekci 2.5.1). Odeslání jednoho bitu do této chytré LED trvá vždy $T_B = 1,25$ µs, přičemž odesílání vždy začíná logickou úrovní HIGH a poté se změní na logickou úroveň LOW [12]. Kódy pro bit 1 a 0 se liší pouze v poměru doby trvání úrovně HIGH k době trvání celého kódu [12]. Zde se naskýtá možnost použít PWM o frekvenci $T_{\mathrm{B}}^{-1} = 800$ kHz a střídou tohoto signálu je pak poměr $T_{\mathrm{HIGH}}/T_{\mathrm{B}}$, kde T_{HIGH} je doba trvání úrovně HIGH v daném kódu (1 nebo 0). Hodnoty střídy pro kód bitu 1 a kód bitu 0 se nachází v Tabulce 2.3. Pro RESET kód, který je definován jako logická hodnota LOW po dobu minimálně 50 µs, je nutné minimálně na 2 periody nastavit střídu 0 %. Jednodušeji to lze vnímat tak, že kdykoliv nejsou odesílána data, je střída 0 %, pro dobu trvání RESET kódu totiž není dle datasheetu určena žádná horní hranice (viz sekci 2.5.1, [12]).

Mezi přední výhody použití PWM patří vysoká přesnost načasování díky použití konstantní frekvence a jednoduchá implementace daných kódů (prostá změna střídy). Nevýhodou je, že pro každý odesílaný bit je potřeba alespoň 1 byte paměti (hodnota do registru střídy, záleží na daném mikrokontroléru).

	Doba trvání HIGH	Střída
Celá perioda	1.25 µs	100 %
Kód 1	800 ns	64 %
Kód 0	400 µs	32 %

Tabulka 2.3: Hodnoty střídy pro kódy bitu 1 a 0 protokolu WS2812B, odvozeno z [12].

To se negativně projeví především při použití DMA a velkých bufferů. Na každý odesílaný byte je totiž zapotřebí přinejmenším 8 bytů bufferu, což v případě velkého počtu LED či nároků na šetření RAM nemusí být schůdné.

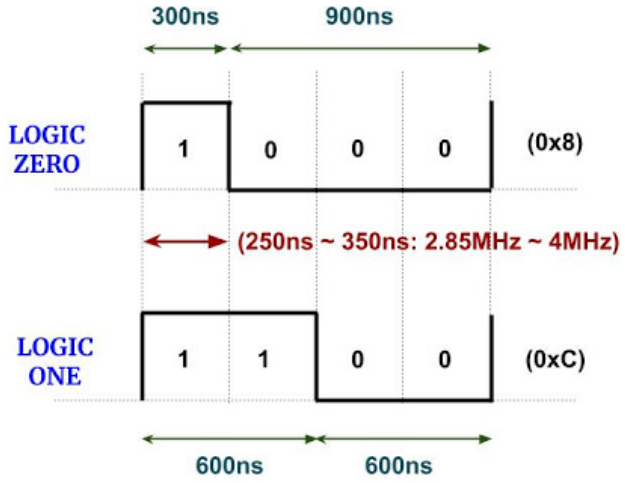
2.5.5 SPI

SPI je rozhraní zajišťující synchronní komunikaci [25]. Komunikace se v danou chvíli účastní dvě strany (mikrokontroléry, integrované obvody, apod.), z nichž se jedna nazývá *Master,* druhá *Slave* [25]. SPI umožňuje duplexní přenos, má oddělené vodiče *MOSI* (Master out, Slave in) a *MISO* (Master in, Slave out) [25]. Součástí rozhraní je vodič *SCLK*, který vede hodinový signál (generovaný Masterem), při jehož náběžné (resp. sestupné, dle konfigurace) hraně jsou data Masterem odesílána na výstup MOSI (úroveň HIGH či LOW) a zároveň čtena na vstupu MISO. U Slave komponenty jsou při této hraně data čtena na vstupu *SDI* a zároveň odeslána na výstup *SDO* [25]. V SPI je také možné dynamicky přepínat, se kterou Slave periferií Master v danou chvíli komunikuje, pomocí výstupu *CS* (Chip Select) [25].

Pro ovládání LED WS2812B však stačí vodič MOSI, připojený na datový vstup WS2812B. Pro dosažení požadovaných hodnot dle datasheetu [12] je vhodné zajistit frekvenci hodinového signálu 3 MHz, jeden SPI bit tak bude odeslán přibližně za $(3\ \text{MHz})^{-1} = 0.333\ \mu\text{s}$. Odpovídajícího poměru hodnot HIGH a LOW (viz sekci 2.5.1) je pak dosaženo posíláním skupin tří SPI bitů.

U SPI bývá na straně vysílače využíváno posuvných registrů, kde dochází k posunu registru a zároveň odeslání jednoho bitu při každém taktu hodinového signálu [26]. Při odeslání celého posuvného registru je třeba jej znovu naplnit, při samotném odesílání však není třeba činnost CPU. U některých mikrokontrolérů je také možné využít DMA k přenosu dat z bufferu v RAM do posuvného registru SPI.

Tato metoda je s použitím hodinového signálu o frekvenci 3 MHz relativně nepřesná (např. oproti PWM), ale dle datasheetu [12] je tato přesnost dostačující, hodnoty se totiž od ideálního stavu mohou lišit o ±150 ns. Vyšší přesnosti lze dosáhnout zvýšením frekvence hodinového signálu, tím i zvýšení potřebného počtu SPI bitů na 1 bit obrazových dat. To znamená častější doplňování výstupního posuvného registru, v případě použití DMA s bufferem v RAM to znamená nutnost použití většího bufferu pro stejnou velikost odesílaných obrazových dat. Odpovídajícím způsobem je pak nutné změnit poměr jedničkových a nulových SPI bitů pro jednotlivé kódy WS2812B bitů 1 a 0. Na obrázku 2.3 je vyobrazen princip fungování této metody.



Obrázek 2.3: Princip ovládání adresovatelných LED pomocí SPI.

Mezi výhody SPI patří vysoká dostupnost napříč mikrokontroléry [20] [27] [20] [28], šetření času CPU oproti bit-bangingu a možnost využití DMA na některých mikrokontrolérech [20]. Jako nevýhodu lze uvést např. nutnost relativně vysoké frekvence SCLK (např. oproti frekvenci PWM, viz sekci 2.5.4).

2.5.6 I^2S

Další popsanou technologií je I^2S, neboli Inter-IC Sound. I^2S je rozhraní, které slouží pro přenos zvukových vzorků mezi zařízeními [29]. Kromě vodičů SCK a SD, které plní podobnou funkci jako SCLK a MOSI u SPI [25] [29], obsahuje I^2S sběrnice ještě vodič WS (word select), který ovlivňuje interpretaci dat – určuje jestli následující vzorek bude interpretován jako zvuk levého (WS = 0) či pravého (WS = 1) kanálu [29]. I^2S je možné použít např. pro přenos zvuku mezi dvěma mikrokontroléry, nebo pro výstup zvuku z mikrokontroléru do reproduktoru přes speciální I^2S-DAC převodník [30].

Jelikož je funkce vodičů SCK a SD prakticky stejná jako funkce odpovídajících vodičů u SPI, je možné použít stejný trik pro ovládání WS2812B jako u ovládání pomocí SPI (viz sekci 2.5.5). Důvod, proč je I^2S zmíněno v této práci je možnost jej v ESP8266 využít společně s DMA [27], mohlo by tedy být možné celý model implementovat na ESP8266. Tato práce nejde implementačně do této hloubky, je zde kladen důraz na implementace A a B (mikrokontroléry ze zadání) a ESP8266 je zde využíván pouze jako síťový

prostředník. ESP8266 však kromě schopnosti připojit se k WiFi nabízí širokou škálu možností, což dosvědčuje i tato elegantní metoda ovládání WS2812B bez nutnosti neustálého vytížení CPU. V budoucnu tak bude možné mimo rámec této práce modely zjednodušit a využít ESP8266 (či pokročilejšího nástupce ESP32 [31]) na implementaci funkcionality celého modelu.

2.5.7 Výběr vhodných metod

Součástí této práce jsou 2 implementace modelu, varianta A a B. Varianta A je zaměřená na efektivní elegantní řešení s nízkou úrovní abstrakce nad hardwarem. Varianta B je zaměřená na rychlost vývoje za použití již hotových známých knihoven a na dobrou přenositelnost mezi jednotlivými Arduiny. Tato filosofie obou variant byla brána v potaz při výběru vhodných metod.

Pro variantu A bylo tedy nutné zvolit metodu s co nejvyšší přesností a s co nejnižším využitím CPU. Požadavek nízkého vytížení CPU splňují všechny metody co lze kombinovat s DMA, tedy v případě STM32F072RB se jedná o PWM, SPI a I^2S [20]. SPI a I2S poskytují nižší náročnost na paměť než PWM. Na druhou stranu PWM zajišťuje vysokou přesnost (viz sekci 2.5.4). Ze toho důvodu bylo pro variantu A zvoleno použití PWM v kombinaci s DMA.

Pro variantu B bylo nutné pouze vybrat některou z dostupných knihoven pro ovládaní chytrých LED. Byly voleny knihovny z Library Manageru v Arduino IDE. Subjektivně nejpříjemnější použití nabízí knihovny *Adafruit Neopixel* a *FastLED*. Obě dvě tyto knihovny používají bit banging (viz sekci 2.5.2), v případě LED WS2812B je tedy nutné počítat s cca. 30 µs blokovaného procesorového času při překreslení jedné LED (24 bitů po 1.25 µs, viz sekci 2.5.1). Knihovna *Adafruit Neopixel* stále obsahuje redundantní kroky a konfigurace, např. nastavení pořadí barev a frekvence signálu [32]. Tyto konfigurace se v knihovně FastLED dějí automaticky dle typu LED, předaného do parametru šablony metody `FastLED::addLeds` [33]. Pro implementaci B tedy byla zvolena knihovna FastLED, která má vysokou míru abstrakce a umožňuje snadný a rychlý vývoj.

2.6 Varianta A – STM32

V této variantě je k řízení modelu použit mikrokontrolér STM32F072RB. Na úplném začátku práce na projektu byl použit mikrokontrolér STM32F042F6P6, který byl k této práci zdarma k disposici od garantů předmětu LPE (Laboratoře z průmyslové elektroniky a senzorů) na ČVUT FEL, kde byl využíván k výuce. Později byl zakoupen mikrokontrolér STM32F072RB, konkrétně na desce Nucleo-F072RB. Jedním z důvodů pro změnu mikrokontroléru byla větší operační pamět STM32F072RB oproti STM32F042F6P6 [20] [34]. Dalším důvodem byla možnost jednoduššího ladění – Nucleo-F072RB má totiž k disposici ST-LINK, pomocí kterého lze jednoduše ladit program, sledovat hodnoty proměnných po jakémkoliv řádku zdrojového kódu programu [35] [36]. Konkrétní model Nucleo-F072RB byl zvolen z toho důvodu, že STM32F072RB

je jediný mikrokontrolér řady STM32F0 (stejné jako STM32F042F6P6, pro jednodušší portování), které jsou k dispozici na svých deskách a zároveň byl v době psaní práce k sehnání na českém trhu.

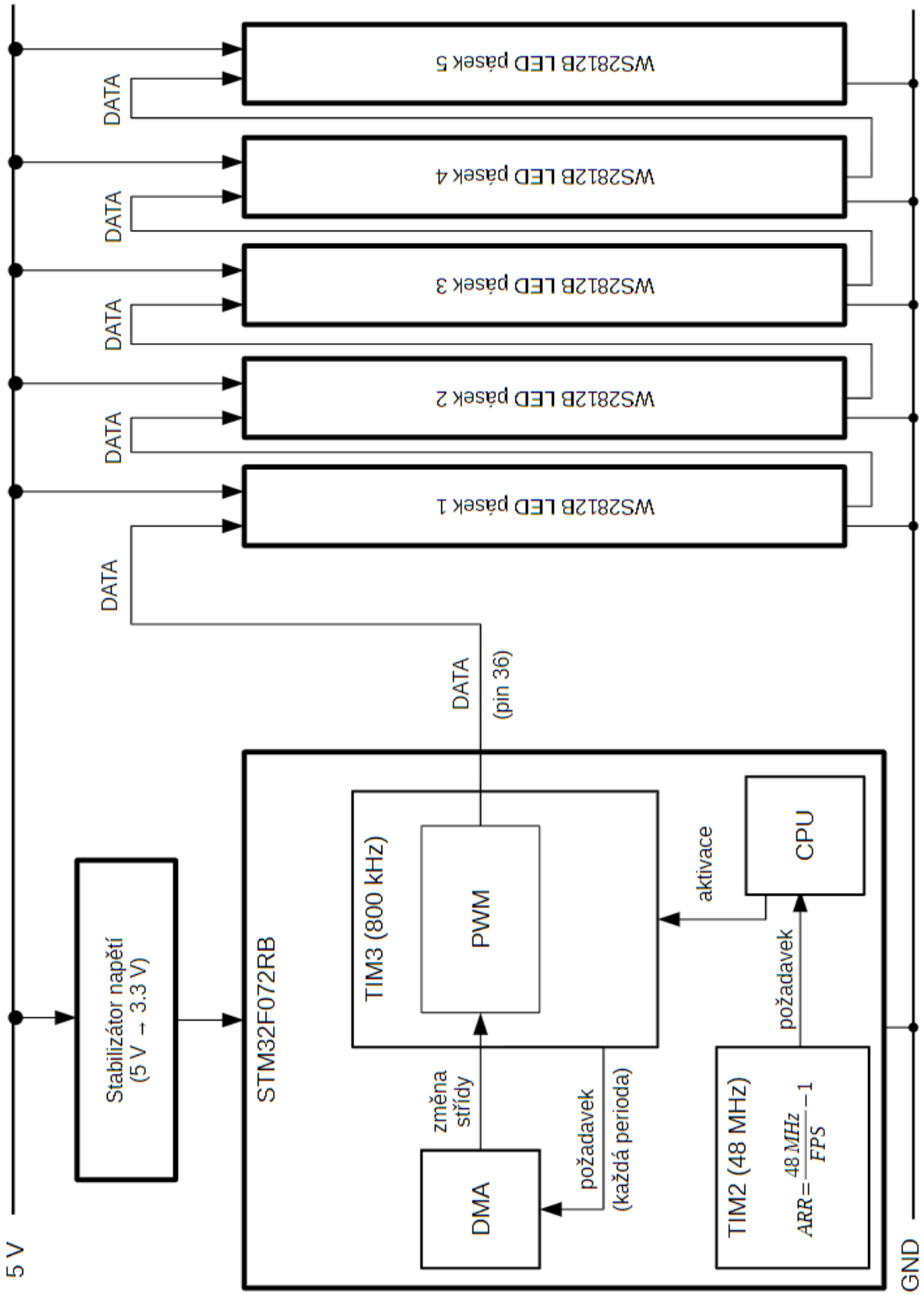
Komunikace s LED páskem zde funguje na principu změny střídy PWM. PWM výstup z mikrokontroléru je přiveden na datový vstup LED pásku. Signál pro odeslání každého bitu je dlouhý 1.25 µs takže je třeba použít PWM o frekvenci 800 kHz, kde 1 perioda trvá oněch 1.25 µs. Změnou střídy, v tomto případě hodnoty registru CCR4 je pak určováno, jestli je odeslán bit 1 (odpovídá střídě 64 %), či bit 0 (odpovídá střídě 32 %). Tento způsob ovládání WS2812B je podrobněji popsán v sekci 2.5.4.

Potřebná obrazová data pro model (v tomto případě RGB) jsou nejprve bit po bitu převedena na byty, které odpovídají hodnotám střídy pro dané bity. Ty jsou poté pomocí DMA převáděny do registru, který určuje střídu PWM. Tento přenos pomocí DMA je uskutečňován o stejné frekvenci, kterou má PWM, tj. 800 kHz. Každá perioda PWM tedy značí jeden bit, a může mít různou střídu. Princip animace v této variantě je znázorněn na blokovém diagramu na Obrázku 2.4.

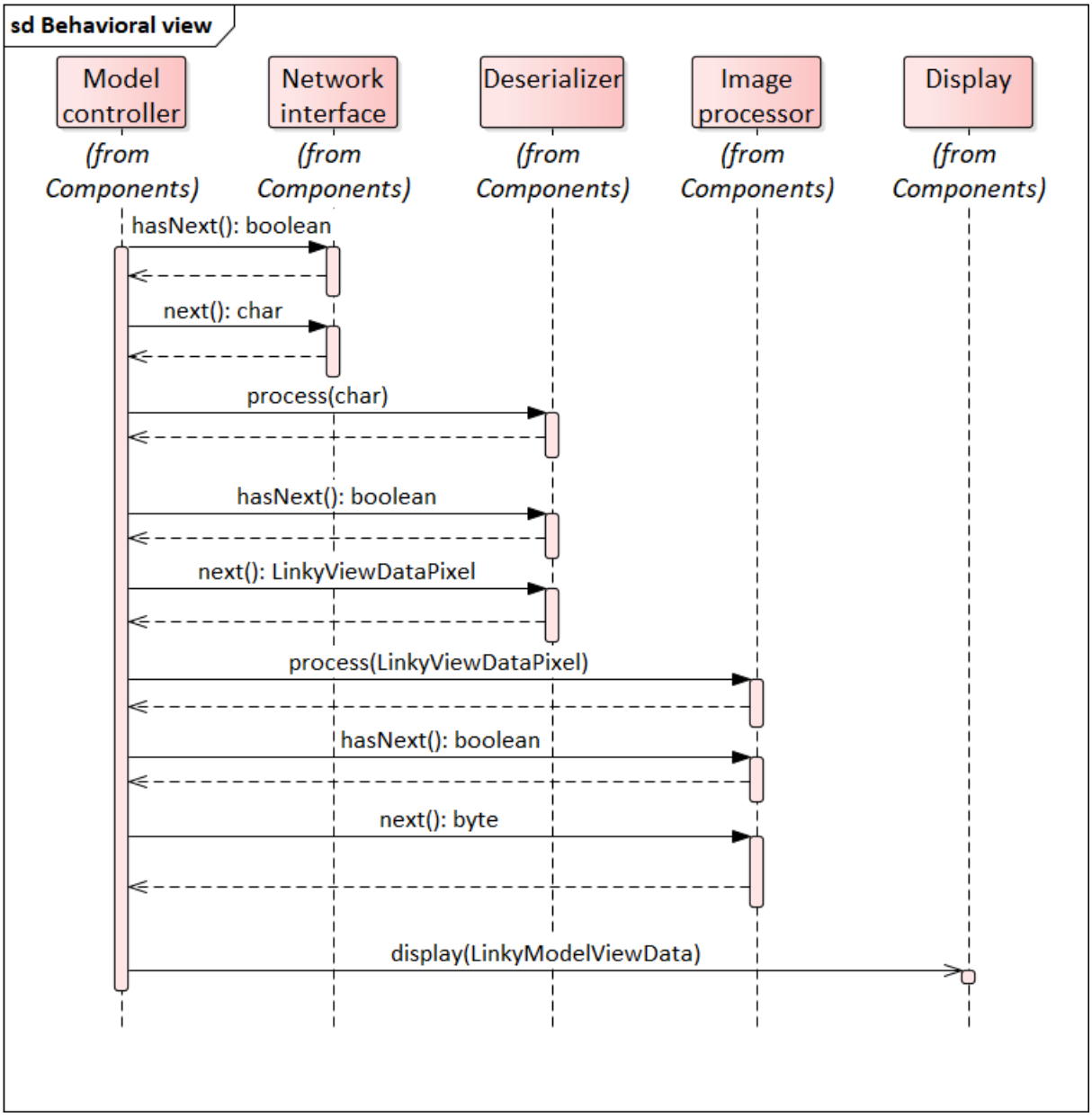
Konkrétně je nejdůležitější čítač TIM3, o frekvenci 800 kHz. Na TIM3 je jednak napojeno PWM (má stejnou frekvenci jako TIM3), jednak vyvolává přenos pomocí DMA. V paměti je buffer, pole bytů, jehož každý byte odpovídá odesílanému bitu. Před vykreslením každého snímku je celý buffer naplněn správnými hodnotami střídy. Poté je aktivováno DMA, které každou periodu TIM3 změní střídu PWM podle toho, jaký bit bude v příští periodě odesílán. TIM2 se FPS-krát za sekundu vynuluje, vyvolá přerušení, a tím i výše popsanou proceduru vykreslení jednoho snímku. Díky TIM2 probíhá animace.

2.7 Varianta B – Arduino

Model je v této variantě řízen pomocí Arduino Leonardo. Na rozdíl od varianty A, kde je použita vlastní implementace komunikace, je pro komunikaci s LED páskem využita knihovna FastLED. Není zde využito žádných přerušení, zajištění správné snímkové frekvence animace je řešeno prostými prodlevami Arduino metodou `delay`. Komunikace řídící softwarové komponenty Model Controller s ostatními komponentami je znázorněna na Obrázku 2.5.



Obrázek 2.4: Blokový diagram principu animace varianty A



Obrázek 2.5: Sekvenční diagram – příklad komunikace komponent

Použití knihovny FastLED (běžným způsobem dle dokumentace) omezuje velikost modelu, jelikož již při inicializaci je třeba určit buffer v RAM, kam budou vkládána obrazová data [33]. Maximální počet LED lze určit z následujícího vztahu:

$$N_{\mathrm{LED\_MAX}} = \frac{\mathrm{RAM}\times 8}{\mathrm{BPP}},$$

kde $N_{\mathrm{LED\_MAX}}$ je vypočtený maximální počet LED, RAM je velikost volné RAM mikrokontroléru v bytech a BPP je barevná hloubka, tedy počet bitů na jeden pixel.

V případě RGB LED, bitové hloubky 8 bit na každý kanál a volných 2000 bytů RAM, což přibližně odpovídá podmínkám této práce, je maximální počet LED modelu dle dříve uvedeného vztahu $666 > 175$, a model tvořený v této práci o velikosti 175 LED (170 aktivních) je tedy z tohoto hlediska realizovatelný.

Kapitola 3

Popis realizace

3.1 Struktura projektu

Součástí projektu jsou kromě kódu implementačních variant i testovací animace, Python generátory animací, užitečné skripty, zdrojový kód textu bakalářské práce v LaTeX aj. Tyto části jsou členěny do podadresářů kořenového adresáře, jejich popis se nachází v Tabulce 3.1.

Soubor	Popis
analysis	Diagramy – projekt v Enterprise Architect
animations	Animace v .json a .lmn formátu (viz sekci 3.4.3)
code	Vlastní implementace a utility (více v Tabulce 3.2)
report	Text bakalářské práce, se zdrojovými kódy a grafikou

Tabulka 3.1: Popis kořenového adresáře

Dále bude popsán adresář `code`, jako stěžejní adresář této práce. Obsahuje především adresáře se zdrojovými kódy jednotlivých implementačních variant a implementace síťového rozšíření. V adresáři `examples` se nachází příklady využití jednotlivých implementací, v adresáři `common` se nachází společné zdrojové kódy pro všechny implementace, včetně dvou platformně nezávislých komponent – *Deserializeru* a *Image Processoru*, ty data zpracovávají bez vedlejších efektů a nevyužívají žádné knihovny mimo C++ STL, tedy standardní knihovnu šablon. Zajímavé utility jsou v adresáři `utils`. Patří mezi ně generátory animací, generátor konfigurace nebo *json2lmn* – vlastní převodník animací v json formátu dle stávajícího LinkyAPI do LMN – optimálního formátu pro model 3.4.3. Celý popis je v Tabulce 3.2.

3.2 Kombinace jazyků C a C++

Jazyk C++ obsahuje na rozdíl od staršího jazyka C koncept tříd, čímž umožňuje vývojáři řešit problémy pomocí objektově orientovaného programování, neboli

Soubor	Popis
common	Kód společný pro více implementačních variant
examples	Příklady využití implementačních variant A a B
impl_a_stm32	Implementační varianta A
impl_b_arduino	Implementační varianta B
impl_networking	Síťové rozšíření pro varianty A a B
test	Testy rychlosti jednotlivých variant
utils	Utility

Tabulka 3.2: Popis adresáře code

OOP. Z tohoto důvodu byla práce implementována v jazyce C++. Bylo tak možné vytvořit třídu pro každou komponentu (viz sekci 2.4). Existují způsoby, jak programovat objektově i v jazyce C. Jazyk C++ řeší tento problém elegantněji – obohatit jazyk C o objektově orientované programování bylo jedním z důvodů vzniku jazyka C++ [37]. Pro všechny mikrokontroléry použité v této práci také existují kompilátory jazyka C++, programy pro Arduino jsou navíc při využití Arduino IDE psány v jazyce Wiring, který je v podstatě jazykem C++ obohaceným o užitečné funkce a knihovny pro vývoj vestavěných systémů [38]. U varianty A a u síťového rozšíření je ale vhodné, aby byly některé části, zejména vstupní bod programu, implementovány v jazyce C.

Důvodem u varianty A je použití utility CubeMX (viz sekci 3.3.2) ke konfiguraci mikrokontroléru. CubeMX po nakonfigurování pomocí grafického rozhraní generuje kód do různých souborů. CubeMX v době psaní práce generuje zdrojové kódy pouze v jazyce C, tedy soubory s příponou `.c`. Po změně přípony na `.cpp` lze projekt zkompilovat, STM32CubeIDE pozná, že se jedná o zdrojový kód v jazyce C++ a použije správný kompilátor. Problém je ale u úprav existujících souborů. Pokud se jedná o úpravu existujících souborů, CubeMX je opraví a přepíše. Bere ale v potaz pouze soubory s příponou `.c`. Soubor se změněnou příponou není brán v potaz, je vygenerován nový soubor, stávající se změněnou příponou není upraven. Takové chování není žádoucí, projekt pak obsahuje dva soubory se stejnými názvy funkcí. Problém by šel vyřešit smazáním původního souboru s příponou `.cpp` a změnou přípony nového souboru na `.cpp`. Vývojář by tak ale mohl přijít o svůj hotový kód, který je umístěn mezi určitými komentáři a při úpravách konfigurace v CubeMX je zachován. Tento problém lze vyřešit změnou přípony před generováním kódu zpět na `.c`, přegenerovat kód v CubeMX a po generování příponu opět změnit na `.cpp`. Nejedná se ale o elegantní řešení, neboť každá změna konfigurace zabírá čas vývojáře a je možné, že nějaké soubory přejmenovat zapomene, což povede k chybám.[1] Mnohem lepší je tedy zdrojové kódy generované CubeMX ponechat v jazyce C.

U síťového rozšíření je s jazykem C nutné počítat z důvodu využití ESP8266 RTOS SDK. Většina komponent zahrnutých v této vývojové sadě je totiž psána v jazyce C, jak se lze přesvědčit po nahlédnutí do adresáře

[1] V této práci se jedná pouze o dva soubory, soubor `main.c` a `stm32f0xx_it.c`, přesto ale může vývojář udělat chybu.

`components`, v kořenovém adresáři vývojové sady ESP8266 RTOS SDK, a jeho podadresářů. Většina zdrojových kódů má příponu `.c`. Vývojová sada ale po přepsání přípony na `.cpp` použije správný kompilátor. Některé komponenty navíc jsou implementovány v jazyce C++. Tyto skutečnosti dávají příslib toho, že jazyky C a C++lze v ESP8266 RTOS SDK kombinovat, což bylo ověřeno způsobem popsaným v následujících odstavcích.

Kombinace jazyků C a C++ je v této práci řešena voláním C++ funkce, implementované v jiném souboru, ze zdrojového kódu v jazyce C. V souboru s C funkcí musí být prototyp C++ funkce, deklarovaný buď přímo v daném souboru, nebo zahrnutý direktivou `#include` z hlavičkového souboru. Jelikož se každá z těchto dvou funkcí nachází v jiném souboru, jsou kompilovány zvlášť. Podle přípony souboru se zdrojovým kódem je automaticky zvolen správný kompilátor a výsledkem jsou dva objektové soubory. Objektové soubory obsahují zkompilovaný binární kód funkcí, každá funkce je označena svým symbolem, vygenerovaným dříve při kompilaci. Tyto objektové soubory jsou následně pomocí linkeru převedeny na jeden spustitelný soubor přímo pro mikrokontrolér, přičemž pro volání jiných funkcí jsou využívány právě názvy symbolů. Jelikož objektové soubory obsahují ve funkcích hotový binární kód pro daný mikrokontrolér a pro C i C++ byl dříve zvolen správný kompilátor, neměl by být problém tyto funkce kombinovat. Problém nastává až ve fázi linkování pomocí linkeru. C funkce zde volá funkci pomocí symbolu vytvořeného z prototypu funkce. Tento symbol ale nebyl definován, názvy symbolů jsou totiž při kompilaci v jazyce C generovány jinak než v jazyce C++. Názvy symbolů v objektovém souboru lze zjistit např. pomocí programu `objdump` z balíčku *binutils*. Je nutné jej spustit s přepínačem `-t`. Rozdíl názvu symbolu po kompilaci v jazyce C a C++ lze vidět v Tabulce 3.3, na příkladové funkci `calculate`.

Prototyp	`int calculate(int a, int b);`
Název symbolu v C	`calculate`
Název symbolu v C++	`_Z9calculateii`

Tabulka 3.3: Rozdíl názvů symbolů C a C++ funkcí

Tento problém lze řešit pomocí konstruktu `extern "C"`, který umožňuje s C++ programem propojit objektové soubory vytvořené pomocí jazyka C. Po umístění před deklaraci funkce je název symbolu do objektového souboru generován stejným způsobem jako v C. Oprava dřívějšího příkladu s příkladovou funkcí `calculate` je demonstrována v Tabulce 3.4.

Prototyp	`extern "C"int calculate(int a, int b);`
Název symbolu v C++	`calculate`

Tabulka 3.4: Název symbolu při použití `extern "C"`.

V případě této práce je tento postup použit zejména u vstupních bodů varianty A[2] a síťového rozšíření.[3] V obou případech je volána C++ funkce `linky_model_main_task`, která je deklarována jako `extern "C"`, implementována je v jiném souboru, aby byla zkompilována zvlášť. Té je předána kontrola a začíná vytváření a ovládání objektů – softwarových komponent modelu. Implementace této funkce vždy obsahuje nekonečnou smyčku. Další využití lze u varianty A vidět u řešení přerušení.[4]

3.3 Varianta A – STM32

Tato sekce popisuje implementaci varianty A, která využívá mikrokontrolér z rodiny STM32.

3.3.1 STM32F072RB

STM32F072RB je mikrokontrolér z rodiny STM32 výrobce *STMicroelectronics* [39] [40] [41]. Mikrokontroléry z rodiny STM32 jsou založené na 32bitovém procesorovém jádře architektury ARM [39]. Existují různé řady mikrokontrolérů STM32, např. F0, G0 či W [39]. Mikrokontroléry dané řady spojuje rodina architektury ARM jejich jádra, např. mikrokontroléry řady STM32F0 jsou založené na procesorovém jádře architektury ARM Cortex-M0 [39]. Do této řady spadá i mikrokontrolér použitý v této práci, tedy STM32F072RB [40].

3.3.2 Vývojové prostředí

K vývoji firmware pro mikrokontroléry STM32 existuje několik přístupů. Nabízí se využít nástroje od STMicroelectronics, výrobce tohoto mikrokontroléru [20]. Mezi tyto nástroje se řadí STM32CubeMX, pomocí kterého je možné pomocí grafického rozhraní nakonfigurovat mikrokontrolér a vygenerovat kód do začátku [42]. V takovém kódu jsou pak komentáři `/* USER CODE BEGIN */` a `/* USER CODE END */` označena místa, kam je možné psát vlastní uživatelský kód, tudíž je možné kdykoliv projekt v CubeMX upravit a přegenerovat, přičemž vlastní uživatelský kód zůstane neporušen [42]. Dalším nástrojem je STM32CubeIDE, integrované vývojové prostředí, které umí generovat projekty s vlastní strukturou [43]. Společně s STM32CubeIDE je nainstalován kompilátor pro mikrokontroléry STM32 (GNU ARM Toolchain), takže není potřeba jej stahovat manuálně [43, str. 51]. Stejně tak STM32CubeIDE obsahuje i integrované STM32CubeMX [43]. Díky automatické konfiguraci projektu a díky CubeMX i generování kódu se tak vývojář nemusí starat o konfiguraci mikrokontroléru, strukturu projektu a detaily kompilace. Posledním jmenovaným nástrojem od STMicroelectronics je STM32CubeProgrammer, který umožňuje nahrát zkompilovaný kód do flash paměti konkrétního mikrokontroléru STM32.

[2] Soubor `code/examples/stm32_offline/Core/Src/main.c`
[3] Soubor `code/impl_networking/main/main.c`
[4] Soubor `code/examples/stm32_offline/Core/Src/stm32f0xx_it.c`

Výhodou využití těchto nástrojů je jednoduchost konfigurace a dobrá kompatibilita těchto nástrojů s mikrokontroléry STM32, jelikož jsou tyto nástroje vydávány přímo výrobcem mikrokontrolérů STM32 [42] [43] [44]. Nevýhodou je, že konfigurace je v podstatě black-box,[5] v případě chyby způsobené generovaným kódem může být složité tuto chybu najít a opravit ji. Je vhodné, aby vývojář alespoň částečně rozuměl vygenerovaným řádkům kódu, v případě potřeby tak může snáze nalézt chybu. Další nevýhodou je využití SW spravovaného komerčním výrobcem. Přestože jsou nyní nástroje od STMicroelectronics zcela zdarma i pro komerční využití [45], není zaručeno, že tomu tak bude i nadále.

Další možností je využití Mbed OS, tedy jednoduchého operačního systému pro mikrokontroléry založené na jádře ARM Cortex M [46]. Mbed OS nabízí dva profily – „full profile“, který je založený na operačním systému pro ARM procesory – Keil RTX, a „bare metal profile“, který neobsahuje Keil RTX, a je zaměřený na aplikace s nutností co nejmenší velikosti zkompilovaného programu nebo aplikace bez komplexního souběhu mnoha úloh. Mbed OS je možné využívat společně s nástroji od ARM – mezi ty patří např. Mbed Studio, vývojové prostředí pro offline použití, dále Mbed Online Compiler, tedy webová aplikace pro psaní a kompilaci zdrojových kódů pro Mbed OS, vhodná např. pro nezávazné zkoušení Mbed OS bez nutnosti stahování či instalace čehokoliv [47]. Poslední možností využití MbedOS je Mbed CLI, rozhraní kompilátoru pro příkazovou řádku [47]. Mezi výhody využití Mbed OS patří vysokoúrovňový přístup nebo možnost využití webové aplikace bez nutnosti instalace IDE. Nevýhodou může být nedostatečná kontrola kvůli přílišné abstrakci nad hardwarem.

GNU ARM toolchain, zmiňovaný v předchozích odstavcích je možné nainstalovat i samostatně, je volně k dostání na serveru ARM [48]. Toolchain obsahuje především gcc kompilátor pro cross kompilaci C programů pro ARM, stejně tak obsahuje g++ kompilátor, pro cross kompilaci C++ programů.[6] Vývojář si může GNU ARM toolchain nainstalovat a používat jej společně s nástroji (textovými editory apod.), na které je zvyklý. Výhodou tohoto přístupu je např. možnost využití nástrojů, které již vývojář využíval dříve. Nevýhodou je především složitost a časová náročnost vytvoření a nastavení vlastního vývojového prostředí.

V této práci bylo pro vývoj zvoleno STM32CubeIDE, jelikož je vydáváno přímo výrobcem použitého mikrokontroléru [43]. Slibuje tak rychlý a pohodlný vývoje pro tuto platformu. Jedním z rozhodujících faktorů byla také přítomnost vestavěného CubeMX, na který je možné kdykoliv přepnout a mikrokontrolér překonfigurovat [43].

[5] Black box je systém, u nějž známe pouze vstupy a výstupy, ale ne vnitřní implementaci.

[6] Cross kompilace je způsob kompilace zdrojového kódu na určitém procesoru, jehož výstupem je spustitelný soubor pro jiný typ procesoru [49]. Příkladem může být kompilace kódu pro mikrokontrolér STM32 prováděná na osobním počítači s Intel procesorem. Cross kompilace se děje ve všech výše zmíněných přístupech vývoje firmware pro STM32.

3.3.3 Implementační detaily

V adresáři `code/impl_a_stm32` se nachází implementace komponenty *Display* a *DataProviderCalc*. Také se zde nachází soubor `.ioc`, který umožňuje jednoduše pomocí STM32CubeMX (viz sekci 3.3.2) vytvořit kód se správným nastavením mikrokontroléru [43].

Nejdůležitější soubory se nachází v adresáři `src`, jedná se o zdrojové kódy implementace. V souboru `Display.cpp` je implementován protokol WS2812B (viz sekci 2.5.1), v metodě `Display::display(LinkyModelViewData*)`. V souboru `DataProviderCalc.cpp` je pak implementováno prolínání barev počítané v reálném čase.

Konfigurace mikrokontroléru stejná jako v příkladu se nachází v souboru `code/impl_a_stm32/linky_model_stm32_config.ioc`, pro vytváření podobných projektů. Toho lze docílit pomocí menu *File* | *New* | *Project from .ioc*. Dále bylo nutné přidat do projektu příkladu společné komponenty a komponenty implementované v `code/impl_a_stm32/`. Toho bylo docíleno pravým kliknutím na projekt, dále *Properties* a v nastavení pak *C/C++ General* | *Paths ans Symbols*. Zde jdou v jednotlivých tabech přidat např. include adresáře, tedy adresáře, které budou prohledávány při hledání hlavičkového souboru zahrnutého direktivou `#include`, v záložce *Includes*. Dále sem jdou přidat adresáře se zdrojovými soubory s implementací těchto hlavičkových souborů, v záložce *Source Location*. Tímto způsobem byly zahrnuty adresáře `code/common` a `code/impl_a_stm32`.

3.4 Varianta B – Arduino

Tato sekce popisuje implementaci varianty B, ve které bylo k řízení modelu využito Arduino.

3.4.1 Arduino

Arduino je značka jednočipových počítačů, založených převážně na mikrokontrolérech AVR výrobce Atmel [50]. Projekt Arduino začal v roce 2005 v Itálii, za účelem vyrobit intuitivní výukovou pomůcku k výuce elektrotechniky a programování [51]. Brzy však Arduino přesáhlo svůj původní účel, je využíváno např. v mnohých volnočasových projektech po celém světě [51] [50].

V této práci má Arduino význam platformy, která je známá široké veřejnosti. V tomto projektu je využito Arduino Mega 2560, ale neměl by být problém řešení portovat např. pro jiné desky Arduino, které jsou podporovány v Arduino IDE [52]. Problémem při portování by mohla být velikost RAM[7], programové paměti nebo využívání periferie, kterou daná deska Arduino nedisponuje.

[7] Arduino Mega má mezi Arduiny jednu z největších RAM [53]

3.4.2 Vývojové prostředí

Jelikož je v této práci varianta s Arduinem zahrnuta pro jeho všeobecnou rozšířenost, bylo pro vývoj a testování zvoleno vývojové prostředí Arduino IDE, které je vydávané přímo společností Arduino [52]. Arduino IDE má kromě textového editoru obsahuje také kompilátor, sériový monitor, správce knihoven i správce nainstalovaných desek (druhů Arduina) [52].

Alternativou by bylo brát Arduino pouze jako AVR mikrokontrolér (viz sekci 3.4.1), okolo kterého je dané Arduino postavené, a kód kompilovat pomocí kompilátoru pro tento AVR mikrokontrolér. V tomto případě by navíc bylo možné stále využít Arduino funkcí, zahrnutím hlavičkového souboru `Arduino.h`. Ten je v případě použití Arduino IDE zahrnut implicitně, bez nutnosti jej zahrnout direktivou `#include`. Jinou alternativou je použití Arduino CLI, pomocí kterého lze kompilovat programy pro Arduino pomocí příkazové řádky.

Vzhledem k filosofii varianty B, tedy využití nástrojů hojně používaných veřejností, bylo vybráno Arduino IDE. Díky tomu může být balíček testován v tom prostředí, ve kterém má být používán. Arduino IDE však skýtá různá úskalí, jako je např. absence napovídání nebo netradiční barvení syntaxe.[8] V Arduino IDE byl tedy vytvořen pouze kód, kterým je softwarový balíček pro model využíván, tedy kód příkladu (`code/examples/arduino_offilne`) a testovací kód (`code/test/arduino_speedtest`). Kód samotného balíčku (`code/impl_b_arduino`) byl tvořen v textovém editoru *Visual Studio Code*. Výhodou *Visual Studio Code* je kromě pohodlného rozhraní také jeho rozšiřitelnost o doplňky pro práci s různými typy souborů [55]. Díky tomu mohl být editor *Visual Studio Code* v této práci využit nejen pro zdrojové kódy programu, ale např. i pro tvorbu textu samotné bakalářské práce v LaTeX.

3.4.3 Definice formátu LMN

Formát LMN byl vytvořen pro tuto práci v rámci zefektivnění komunikace s SD kartou. Zkratka LMN byla vymyšlena pouze pro odlišení souborů pomocí přípony `.lmn`, a značí *LinkyModel native*.

Jedná se o binární formát pro ukládání animací. První byte značí FPS, tedy snímkovou frekvenci dané animace. Další byty jsou již data samotné animace, již zpracované pro daný model. Je tedy je možné přímo přečíst do struktury LinkyModelViewData a snímek zobrazit. Snímků může být v souboru za sebou libovolný počet.

V případě této práce se jedná o pořadí barev GRB, pořadí panelů *zleva doprava* a pořadí jednotlivých LED *shora dolů*.

[8] Obarvena jsou pouze ta slova, která jsou zapsána v některé knihovně v `keywords.txt` [54], přičemž Arduino IDE neřeší, jestli je tato knihovna v danou chvíli zahrnuta, či nikoliv (zjištěno pokusem).

3.4.4 Utilita json2lmn

Úkolem utility *json2lmn* je převádět animace z json formátu současného LinkyAPI do formátu LMN (viz sekci 3.4.3). Vstupní data přebírá ze standardního vstupu, a binární data posílá na standardní výstup, je tedy nutné je přesměrovat do souboru. Příklad použití je na Linuxu je na Obrázku 3.1, jedná se o převod animace s názvem `linky_api.json` na animaci `linky_model.json` s 24 snímky za sekundu.

```
cat linky_api.json | ./json2lmn 24 > linky_model.lmn
```

Obrázek 3.1: Použití utility *json2lmn*

Utilita *json2lmn* byla testována pouze na Ubuntu ve WSL, ale pomocí přiloženého CMakeLists.txt by neměl být problém jej zkompilovat i nativně pro Windows [56]. Zdrojový kód totiž využívá pouze knihovny ze Standardní knihovny šablon C++ (*string* a *iostream*) a multiplatformní komponenty z této práce (*Deserializer* a *ImageProcessor*).

3.4.5 Instalace knihovny LinkyModel.h

Aby mohla být knihovna využívána v Arduino IDE, musí být hlavičkové soubory a soubory s implementací umístěny v jednom adresáři [54]. Tento adresář pak musí být umístěn v adresáři `libraries`, umístěném ve sketchbooku [54].[9]

V této práci ale nebylo vhodné vyvíjet vše v jednom adresáři. Varianta s Arduinem totiž kromě kódu specifického pro Arduino využívá i kód společný pro obě varianty. Také je přehlednější udržovat konsistentní členění pro celou práci, tj. členit hlavičkové soubory a soubory s implementací do oddělených adresářů. Soubory byly tedy při vývoji členěny do adresářů bez ohledu na strukturu požadovanou prostředím Arduino IDE, nasazení pak bylo řešeno pomocí skriptů. Nasazovat je potřeba vždy před testováním, případně před případnou distribucí knihovny v ZIP archivu.

Při instalaci se pouze zkopírují všechny potřebné soubory do potřebného adresáře. Instalace je spouštěna pomocí prostředí *make*, v adresáři `impl_B_Arduino` je nutné zavolat `make install ARDUINO_LIB_PATH=libraries`. Při volání tohoto příkazu je třeba definovat proměnnou `ARDUINO_LIB_PATH` jako cestu k adresáři libraries ve sketchbooku. Definici přímo při volání `make install` je možné vynechat, pokud je v shellu zadefinována globální proměnná `ARDUINO_LIB_PATH` s cestou k adresáři libraries ve sketchbooku. Instalace byla testována na Ubuntu ve WSL.

Výsledný adresář obsahuje ještě soubor `LinkyModel.h`, který má jako jediný příponu `.h`. Je tomu tak z toho důvodu, že jinak docházelo k problémům při zahrnutí knihovny v Arduino IDE pomocí menu *Sketch* | *Include library*.

[9] Umístění sketchbooku se v Arduino IDE nastavuje v menu *File* | *Preferences* | *Sketchbook location*).

Toto menu totiž automaticky vkládá direktivu `#include` pro všechny soubory, které končí na `.h` v adresáři dané knihovny (zjištěno pokusem). Jedná se o drobný detail, který často knihovny neřeší (např. knihovna SdFat zahrne více souborů než jeden), ale může přinést větší přehlednost do zdrojového kódu programu pro Arduino.

3.4.6 Spuštění

Pro využití varianty B je třeba zapojit čtečku SD karet na SPI piny, přičemž v základu je třeba zapojit pin CS čtečky na pin 4 Arduina. Výstup dat pro WS2812B pásky je na pinu 10. K přepínání animací slouží pin 12. Když se na pinu 12 objeví hladina HIGH, je přepnuta animace. Je tedy vhodné jej přes rezistor spojit s hladinou LOW a přes tlačítko s hladinou HIGH. Kompilovat a flashovat kód je vhodné pomocí Arduino IDE, ať už kód příkladu,[10] nebo testovací kód.[11] Je nutné mít správně nainstalovanou knihovnu *LinkyModel.h* dle sekce 3.4.5.

3.5 Síťové rozšíření

Tato sekce popisuje implementaci síťového rozšíření, které je v podstatě další implementační variantou, zpracovaná data ale nezobrazuje, ale vypisuje je na konzoli pro kontrolu uživateli přes UART. Pro propojení s jinou variantou by bylo možné např. posílat data pomocí SPI. Jde tak použít i s mikrokontroléry z rodiny STM32 i s Arduinem, stačí pouze podpora SPI [20] [28].

3.5.1 ESP8266

ESP8266 je mikrokontrolér od firmy Espressif Systems, známý pro svou schopnost připojení k Wi-Fi síti [57]. Je založen na 32-bit procesorovém jádře Tensilica Xtensa LX106 [27]. Tensilica je již zaniklá firma, v roce 2013 byla pohlcena společností Cadence Design Systems [58].

S výchozím továrním firmwarem je ESP8266 možné ovládat pomocí speciálních AT instrukcí, definovaných výrobcem, které jsou do ESP8266 posílány pomocí UART, např. z jiného mikrokontroléru [59]. Pomocí nich lze mikrokontrolér např. připojit k síti Wi-Fi, navázat TCP spojení nebo komunikovat pomocí TCP [60]. Je to jednoduchý, ovšem velmi omezující způsob ovládání ESP8266. AT instrukční sada je omezená, není např. možné jednoduše navázat zabezpečené SSL spojení [60]. Pomocí výchozího firmware tedy není možné efektivně implementovat HTTPS klienta, který je pro tuto práci potřeba (viz sekci 2.2).

Při řešení netriviálních problémů je mnohem výhodnější naprogramovat ESP8266 jako samostatný mikrokontrolér, tím opadne nutnost posílat instrukce přes UART a ušetří se čas, především se ale vývojáři otevře pestrá škála nových možností nad rámec AT instrukční sady, díky kterým bude moci

[10] `code/examples/arduino_offline`
[11] `code/test/arduino_speedtest`

řešit problémy, jejichž řešení by s pouhou AT instrukční sadou bylo velmi obtížné (HTTPS klient), nebo zcela nemožné (ovládání některých periferií, např. SPI nebo I^2S) [60].

ESP8266 je prodáváno různými výrobci integrované v různých modulech, které se liší např. fyzickou velikostí, uspořádáním pinů, přítomností USB-Serial převodníku apod [57]. Tato práce byla vyvíjena pomocí modulu WeMos D1 mini, protože má pro uživatele vyvedena všechna potřebná rozhraní (SPI a v budoucnu potřebné I^2S – viz sekci 2.5.6) a obsahuje USB-Serial převodník, pomocí kterého je možné flashovat firmware [61]. Opadá tak nutnost kupovat a zapojovat USB-Serial převodník navíc.

3.5.2 Vývojové prostředí

Jedna z možností, jak vyvíjet firmware pro ESP8266 je použití Arduino IDE [62]. Jeho hlavní výhodou je jednota, po přidání desky ESP8266 je možné jej začít využívat s podobným rozhraním jako Arduino. Jeho nevýhodou ale může být nedostatečná kontrola nad chováním mikrokontroléru způsobená vysokou mírou abstrakce nebo pevně daná struktura projektu v Arduino IDE (viz sekci 3.4.2), která má spoustu nevýhod. Není např. elegantně možné direktivou `#include` zahrnout hlavičkové soubory odjinud, než z adresáře libraries, či projektového adresáře. Pro variantu B (s Arduinem) toto bylo nutné vyřešit, zde ale bylo možné se problému zcela vyhnout a použít některou z dostupných SDK.

Jednou z SDK je *ESP8266 nonOS SDK* [63]. Tento druh SDK je na pozadí využit i při programování ESP8266 pomocí dříve zmíněného Arduino IDE [62]. Výrobce Espressif Systems však na gitu *ESP8266 nonOS SDK* varuje, že už tento druh SDK není dále vyvíjen a není doporučeno ho využívat pro nové projekty [64].

Espressif Systems naopak doporučují využít jejich sadu *ESP8266 RTOS SDK* [64], která je založená na operačním systému reálného času FreeRTOS [65, str. 129]. Díky němu funguje v *ESP8266 RTOS SDK* na rozdíl od *ESP8266 nonOS SDK* preemptivní multitasking [66] [67]. Ten umožňuje docílit zdánlivého běhu více úloh naráz při jednom vlákně [66]. Při preemptivním multitaskingu operační systém spravuje běžící úlohy, přičemž je může pozastavovat a opětovně spouštět dle jejich předem nastavené priority [66].

Ve vstupním bodu programu je v ESP8266 RTOS SDK tedy možné nastavit jednu či více úloh, ve které poběží uživatelská funkce [67]. To dává příslib škálovatelnosti do budoucna. Pokud např. vznikne požadavek na spuštění jednoduché webové stránky na ESP8266, kde by bylo možné model konfigurovat, bude tak možné celkem jednoduše učinit přidáním další úlohy s implementací webového serveru. Výhodou *ESP8266 RTOS SDK* je především zmíněný preemptivní multitasking, který může být ovšem i nevýhodou, jelikož potenciálně vytvoří prostor pro těžko odhalitelné, těžko opravitelné chyby. Doporučení od výrobce a příslib škálovatelnosti do budoucna nad tímto rizikem ale zvítězily, a pro vývoj byl zvolen *ESP8266 RTOS SDK*.

3.5.3 Využití IDF s ESP8266

ESP-IDF je vývojová sada od Espressif Systems [68]. Název ESP-IDF vznikl spojením zkratky firmy Espressif se zkratkou pro IoT Developement Framework [68]. Tento framework je určen pro vývoj SW pro mikrokontroléry řady ESP32 [68]. Mikrokontroléry ESP32 mají s ESP8266 četné podobnosti: WiFi funkcionalitu, sériová rozhraní UART, SPI, I2S a jiná, ale mikrokontroléry ESP32 jsou výkonnější než ESP8266 a poskytují více typů rozhraní, např. CAN sběrnici [31] [27].

Stejně jako výše zmíněné *ESP8266 RTOS SDK* a *ESP8266 nonOS SDK*, poskytuje ESP-IDF různé softwarové komponenty pro zjednodušené využití periferií mikrokontroléru (SPI, WiFi apod. [69]) a různé utility – např. pro kompilaci kódu nebo pro flashování programu do paměti mikrokontroléru [68]. ESP-IDF se ovládá pomocí python skriptu `idf.py` a využívá prostředí *CMake* [68], které slibuje lepší platformní nezávislost [56].

Struktura projektu v původním ESP8266 RTOS SDK (při využití make) a ESP-IDF je podobná. Stejně jako každý projekt v ESP8266 RTOS SDK obsahuje ve svém kořenovém adresáři soubor `Makefile` [70], mají ESP-IDF projekty ve svém kořenovém adresáři soubor `CMakeLists.txt` [71]. Stejně jako každá komponenta v ESP8266 RTOS SDK je adresář, který obsahuje soubor `component.mk` [70], komponenta v ESP-IDF je adresář, který obsahuje `CMakeLists.txt`, ve kterém je použito `idf_component_register()` [71]. Obě porovnávané SDK jsou navíc založeny na FreeRTOS [71] [65].

Pro lepší přenositelnost kódu mezi ESP8266 a ESP32 se podnik Espressif Systems rozhodl zmigrovat svůj ESP8266 RTOS SDK na *IDF styl*. Na gitu ESP8266 RTOS SDK je v README[12] uvedeno, že po verzi 2.0.0 migrace začne a stanovuje cíl zmigrovat na IDF styl do verze 3.0.0. V době psaní této práce byla verze 3.4 poslední distribuovanou verzí ESP8266 RTOS SDK, tudíž by měla být plně zmigrovaná na IDF styl. Nachází se ale v jakémsi přechodném stavu, v komponentách je jak soubor `component.mk`, tak `CMakeLists.txt`.

I přes matoucí README, kde není výslovně uvedeno, že již lze IDF-styl používat, bylo později zjištěno, že je to možné [65, str. 5]. Pokusem bylo ověřeno, že po přidání potřebných `CMakeLists.txt` souborů lze i síťové rozšíření kompilovat a flashovat pomocí IDF utilit. Projekt byl tedy předělán do IDF stylu, což přineslo následující výhody:

- Příslib jednodušší případné migrace na ESP32 v budoucnu (viz sekci 2.5.6).
- Možnost využití WolfSSL, lepší implementace (viz sekci 3.5.4).
- Možnost případně vyvíjet kdekoliv, kde lze nainstalovat CMake.
- Subjektivně jednodušší použití utilit.

[12] Nachází se zde: `https://github.com/espressif/ESP8266_RTOS_SDK/blob/master/README.md`.

3.5.4 Výběr vhodné implementace TLS

Komunikace se současným LinkyAPI je možná pouze pomocí zabezpečeného HTTP protokolu, neboli HTTPS (viz sekci 2.2). Protokol HTTPS je realizován pomocí zabezpečovací vrstvy TLS [72]. Implementaci TLS není nutné vytvářet, vzhledem ke složitosti implementace je lepší využít některou z hotových implementací TLS, které ESP8266 RTOS SDK nabízí.

Implementace je možné přepnout v nastavení projektu, které je možné otevřít pomocí `idf.py menuconfig` volaného z kořenového adresáře ESP8266 RTOS SDK projektu.[13] V nastavení je pak možné přepnout implementaci v menu *Component config* | *ESP-TLS* | *Choose SSL/TLS library for ESP-TLS*. V základu je na výběr ze dvou implementací SSL – MbedTLS (výchozí varianta) a WolfSSL. V rozhraní se neliší, vývojář u každé implementace používá stejnou sadu funkcí z hlavičkového souboru `esp-tls.h` z ESP8266 RTOS SDK.

Tento projekt byl ze začátku vyvíjen s implementací MbedTLS, která je v ESP8266 RTOS SDK nastavena ve výchozím stavu a je v základu ESP8266 RTOS SDK již obsažena. Později byly vzaty v potaz výhody WolfSSL (hlavně nižší náročnost na RAM dle zadání). Podrobnější srovnání obou implementací se nachází v Tabulce 3.5. Porovnání bylo provedeno společností Espressif Systems na programu, využívajícím funkci `https_request` se serverovou autentikací [73].

Vlastnost	WolfSSL	MbedTLS
Využití paměti na haldě	19	37
Využití paměti na zásobníku	2.2	3.6
Velikost programu ve flash paměti	858	736

Tabulka 3.5: Srovnání WolfSSL a MbedTLS (v kB, přibližné hodnoty) [73].

Mezi nevýhody WolfSSL patří nutnost jej stáhnout jako externí komponentu a ukončená distribuce zkompilované knihovny pro ESP8266 [73]. Dříve bylo možné knihovnu stáhnout a použít ji v jakékoliv SDK na ESP8266, jednalo se o staticky linkovanou knihovnu, která obsahuje signatury funkcí a již zkompilovaný kód spustitelný na procesoru [73]. Nyní je k disposici pouze zdrojový kód WolfSSL, společnost Espressif Systems na svém gitu udržuje pouze port pro ESP-IDF (viz sekci 3.5.3). Tento port je nutné stáhnout z gitu a použít jako komponentu v ESP-IDF. Do této práce byla komponenta WolfSSL naklonována do adresáře `components/esp-wolfssl`. WolfSSL bylo hlavním důvodem pro migraci projektu na styl ESP-IDF (viz sekci 3.5.3).

3.5.5 Konfigurace síťového rozšíření

Konfigurace síťového rozšíření je možná dvěma způsoby. Pokud má vývojář k dispozici ESP8266 RTOS SDK v IDF-stylu, může před kompilací zdrojo-

[13] V případě této práce se jedná o adresář `code/impl_networking`.

vého kódu v příkazové řádce zavolat `idf.py menuconfig`. Otevře se grafické rozhraní konfigurátoru, kde je možné nastavit i přístupové údaje k Wi-Fi síti, změnit server nebo endpoint LinkyAPI, nebo změnit pin pro vstup do dynamické konfigurace (vysvětlena posléze). Všechna tato nastavení se nachází v menu *LINKY model settings*.

Pokud má vývojář k dispozici pouze zkompilovaný program, nebo z jiného důvodu nemůže konfigurovat pomocí menuconfigu, je zde možnost dynamické konfigurace. Ta se aktivuje přivedením úrovně HIGH na pin vstupu do dynamické konfigurace (ve výchozím stavu se jedná o GPIO 4). Poté si stačí připravit sériový monitor (testováno na idf.py monitor), ESP8266 restartovat (na Wemos D1 mini stiskem reset tlačítka) a vývojář bude vyzván k vyplnění SSID a hesla sítě, pomocí UART. Takto si může program vyzkoušet kdokoliv i bez potřeby zdrojového kódu.

Kapitola 4
Testování

Tato kapitola popisuje testovací fázi práce. Testovací fáze této práce sestávala z několika manuálních testů a z automatického měření rychlosti, která je důležitým kritériem pro plynulé fungování modelů instalace Linky. Bylo využito skutečnosti, že systém sestává z oddělených komponent, které je možné testovat samostatně.

4.1 Manuální testování

Manuální testy byly prováděny jednak u funkcionalit, které vyžadují interakci člověka, jednak při kontrole správnosti dat odesílaných na výstup mikrokontroléru, tedy i na vstup LED pásků. Byly vytvořeny testovací scénáře týkající se implementace varianty B, ale i síťového rozšíření. Správnost odesílaných dat byla testována pouze u varianty A, jelikož u ní na rozdíl od varianty B nebyla pro zobrazování využita knihovna, ale protokol WS2812B byl implementován od úplného základu (viz sekci 2.5.1 a 2.6).

4.1.1 Přepínání animací na SD kartě

Tento scénář testuje funkčnost tlačítka na přepínání animací z SD karty u příkladu využití Varianty B. Tlačítko musí být připojeno na pin definovaný symbolem `LINKY_ANIMATION_BUTTON_PIN` v tomto příkladu, ve výchozím nastavení se jedná o pin 12.

Testovaný program je příkladový kód pro offline využití Varianty B. Nachází se v adresáři `code/examples/arduino_offline`. Zde je možné otevřít soubor `arduino_offline.ino` v *Arduino IDE* a nahrát kód do Arduina. Arduino použité v této práci je Arduino Mega. Je nutné také mít na paměti, že pro použití tohoto příkladu je potřeba mít správně nainstalovanou knihovnu *LinkyModel.h* dle sekce 3.4.5.

Před začátkem testování je potřeba zkopírovat testovací *lmn* animace z adresáře `/animations/lmn` na SD kartu do adresáře `/linky`, případně jiného adresáře, pokud je definovaný symbolem `LINKY_SD_ANIMATION_DIR`. Pro tento scénář byly použity pouze animace `color_stripes_test.lmn` a `individual_stripes_test.lmn`. Je možné použít i více animací, nebo je např. vložit do jiného adresáře a zkontrolovat, že nebudou přehrány.

Animace `color_stripes_test` by měla být přehrána jako první, vizuálně ji lze od druhé animace odlišit tak, že se střídají pouze dva snímky a na každém z pěti pásků svítí barva, nebo je zhasnutý. Na prvním snímku svítí na páscích (jmenováno zleva): červená, zelená, modrá, zbylé dva pásky jsou zhasnuté. Na druhém snímku svítí barvy: azurová, purpurová, žlutá, zbylé dva pásky svítí bíle.

Oproti tomu na druhé animaci, `individual_stripes_test`, svítí vždy pouze jeden vertikální pásek, a to celý červeně. Po každém snímku se červený pásek posune o jeden pásek doprava, po prostřídání všech pěti pásků začíná opět zleva.

Scénář pro otestování je následovný:

1. Vložit SD kartu s animacemi do čtečky připojené k Arduinu.
2. Připojit Arduino ke zdroji napětí (např. USB kabelem).
3. *V tuto chvíli by se měla automaticky spustit první animace, tedy* `color_stripes_test`.
4. Stisknout tlačítko připojené na pin `LINKY_ANIMATION_BUTTON_PIN` (pin 12 ve výchozí konfiguraci)
5. *Animace by se měla přepnout na animaci* `individual_stripes_test`.
6. Stisknout tlačítko znovu.
7. *Jelikož v adresáři nejsou žádné další animace, měla by se opět začít přehrávat první animace,* `color_stripes_test`.

Tento test dopadl úspěšně, přepnutí je okamžité. Při vývoji byl problém, že při přepínání animace podivně problikávala, to bylo opraveno zastavením animace po dobu přepínání. V případě příliš rychlé nebo pomalé odezvy tlačítka je možné chování ladit pomocí změny definice `BUTTON_PRESS_MS`, která udává očekávanou dobu trvání stisku tlačítka v milisekundách. Nachází se v používaném příkladovém kódu, `arduino_offline.ino`. Ve výchozím nastavení je tato konstanta nastavena na 150 ms.

4.1.2 Chybové hlášky SD karty

Při využívání Arduino knihovny *LinkyModel.h* mohou nastat různé potíže se čtením dat z SD karty, způsobené špatným zapojením SPI rozhraní, vadnou čtečkou karet, nebo chybnou SD kartou. Problémem může být také špatně zformátovaná SD karta, formáty podporované použitou knihovnou *SdFat* jsou *FAT32* a *FAT16*, přičemž doporučený formát je *FAT16*. Při tvorbě této práce byla také vyměněna 64GB SD karta za 2GB SD kartu, jelikož větší kartu nebylo *Arduino* schopné přečíst, přestože byla správně zformátována.

Pokud nastane některá z těchto chyb, je vhodné vývojáře informovat, že se jedná o chybu spojenou s SD kartou. Kdyby tak nebylo učiněno, mohl by

marnit čas hledáním chyby ve svém kódu nebo v kódu knihovny. O tyto chybové hlášky se stará komponenta *DataProviderSdCard* při vytváření spojení s SD kartou v konstruktoru. V této sekci jsou chybové hlášky otestovány.

Testovaný program je stejný jako v minulé sekci, `arduino_offline.ino`. Před začátkem testu je nutné odstranit z SD karty adresář `/linky`, pro otestování všech chybových hlášek.

Scénář je následovný:

1. Otevřít si sériový monitor (např. přímo v *Arduino IDE* | *Tools* | *Serial Monitor*).
2. Pokud je ve čtečce karet připojené k Arduinu SD karta, je třeba ji vyjmout.
3. Připojit Arduino ke zdroji napětí, nebo restartovat program pomocí *RESET* tlačítka.
4. *Po pár sekundách by se na Serial Monitoru měla objevit zpráva o problému s inicializací SD karty.*

Mimo fyzickou chybu SD karty je možné také špatně pojmenovat či umístit adresář s animacemi na SD kartě. Scénář pro otestování hlášení chyb je následovný:

1. Vložit do čtečky karet připojené k Arduinu SD kartu s chybějícím adresářem `/linky`
2. Připojit Arduino ke zdroji napětí, nebo restartovat program pomocí *RESET* tlačítka
3. *Po pár sekundách by se na Serial Monitoru měla objevit zpráva o chybě čtení adresáře.*

Oba testy týkající se chybových hlášek proběhly úspěšně.

4.1.3 Dynamická konfigurace připojení k síti

U síťového rozšíření je SSID a heslo přístupového bodu možné nastavit buď staticky před kompilací nebo dynamicky pomocí UART 3.5.5. V tomto scénáři bude otestována dynamická konfigurace.

Před testováním pomocí tohoto scénáře je zapotřebí mít zprovozněnou sériovou komunikaci s ESP8266. Je možné použít nějaký sériový terminál, např. *RealTerm*[1] nebo *TeraTerm*.[2] V případě této práce byl ale použit přímo příkaz `idf.py monitor` – sériový monitor, který je součástí balíčku nástrojů IDF (viz sekci 3.5.3). Scénář je následovný:

[1] https://sourceforge.net/projects/realterm/
[2] https://ttssh2.osdn.jp/index.html.en

1. Připojit ESP8266 ke zdroji napětí a spojit zdroj napětí s GPIO 4 (aktivace konfiguračního módu, viz sekci 3.5.5).
2. Připravit si sériový terminál, spojený s ESP8266
3. Resetovat ESP8266 (na Wemos D1 Mini pomocí RESET tlačítka).
4. *Na sériovém terminálu by se měla objevit výzva k zadání SSID sítě.*
5. Pomocí sériového terminálu zaslat SSID sítě, zakončit jej odřádkováním `\n`.
6. *Na sériovém terminálu by se měla objevit výzva k zadání hesla.*
7. Pomocí sériového terminálu zaslat heslo Wi-Fi sítě, zakončit odřákováním `\n`.
8. *Na sériovém terminálu by se měla zobrazit zpráva o začátku pokusu o připojení.*
9. *Na sériovém terminálu by se měla zobrazit zpráva o výsledku připojení k síti.*

Test dynamické konfigurace připojení dopadl úspěšně.

4.1.4 Správnost odesílaných dat ve variantě A

Varianta A obsahuje implementaci protokolu WS2812B (viz sekci 2.5.1) od úplného základu, nelze se tedy bez testování bezmezně spoléhat na správnost odesílaných dat. Byty barev totiž bylo nutné převádět bit po bitu na buffer dat odesílaných do WS2812B, přičemž při tomto převodu mohlo nastat porušení správnosti dat, např. zavedením špatného pořadí bitů (viz sekci 2.6 a 2.5.1).

Správnost byla zpočátku testována pouze vizuální kontrolou modelu po zasílání základních barev modelů RGB a CMYK. Toto testování bylo úspěšné, ale netestuje správnost příliš dobře, jelikož nepokrývá případné špatné pořadí bitů v jednotlivých bytech – tyto byty totiž obsahovaly pouze samé jedničky, nebo samé nuly.

Vhodnější je využít logického analyzéru.[3] Pro tento test byl využit logický analyzér neznámé značky se vzorkovací frekvencí 24 MHz, který je kompatibilní s programem Saleae Logic. Saleae Logic umí přečtená data i interpretovat, obsahuje různé SW analyzéry na konkrétní protokoly. Mezi ně patří např. běžné protokoly SPI, UART, ale mimo to i protokoly adresovatelných LED včetně WS2812B (viz sekci 2.5.1).

Díky logickému analyzéru, který zobrazí přesné hodnoty odesílaných pixelů se není nutné spoléhat na pouhou vizuální kontrolu triviálních barev, ale je možné volit jakékoliv hodnoty, a ty kontrolovat. Testování bylo provedeno pomocí programu prolínání barev umístěného v adresáři `bin/stm32_example.bin`. Tímto způsobem byla odhalena chyba posílání bitů ve špatném pořadí, kterou by při testování triviálních barev pouhým pohledem bylo nemožné najít.

[3] Logický analyzér je přístroj, který umožňuje vzorkovat digitální signál [74].

4.2 Měření rychlosti

Jelikož jsou modely schopny přehrávat animace, je rychlost důležitým ukazatelem použitelnosti, aby animace byly co nejplynulejší. To platí zejména u síťového rozšíření, kde je tento aspekt negativně ovlivněn komunikací s Internetem.

Rychlost byla měřena na úrovni jednotlivých softwarových komponent. Na jejich samostatném využití při měření byla demonstrována dobrá samostatnost komponent, jeden z aspektů SOLID návrhu (*Interface segregation principle* [6]).

Testy se nacházejí v adresáři `code/test`. Testovací programy využívají společný kód a kód dané implementace, musí tedy tyto závislosti zahrnout do kompilace, podobně jako příkladové programy (viz sekci 3.1).

Ve variantě A je implementován pouze jednoduchý příklad prolínání barev, na důkaz funkčnosti low-level implementace. Není zde tedy motivace měřit variantu A, jelikož výsledná čísla by nebyla s ostatními variantami dobře porovnatelná. Zajímavější je měření rychlosti načítání předem připravených obrazových dat z SD karty ve variantě B a měření rychlosti stahování dat ze sítě v implementaci síťového rozšíření.

4.2.1 Varianta B

Tato sekce popisuje měření rychlosti implementační varianty B, která pro svou činnost využívá SD kartu. Měření varianty B je spouštěno ze souboru `adruino_speedtest.ino` a je k němu využita třída *LinkyModelSpeedTest*.

Měření byla prováděna vícekrát, s různými velikostmi bufferu a s různým množstvím dat. Měření dle velikosti bufferu v Bytech jsou uvedena v Tabulce 4.1.

Buffer [B]	Přečteno [B]	Rychlost [B/s]
1	20000	27000
512	20000	344000
1024	20000	434000
1024	10000	416000
1024	1024	256000

Tabulka 4.1: Měření rychlosti čtení dat z SD karty.

Měření s jiným počtem čtených bytů nebyla prováděna s žádným předpokladem, pouze pro zajímavost, ale je vidět, že velký rozdíl zde není. Předpoklad u bufferu se potvrdil, čím více bytů je čteno naráz, tím je rychlost čtení vyšší.

Z měření plyne, že rychlost čtení SD karty je pro model dostatečná, pokud se použije formát *lmn* (viz sekci 3.4.3). Maximální snímková frekvence f_{MAX} je totiž přibližně:

$$f_{MAX} = \frac{r}{n_{PIXEL} \cdot B_{PIXEL}},$$

kde r je rychlost čtení v [B/s], n_{PIXEL} je počet pixelů v jednom snímku a B_{PIXEL} je počet bytů v jednom pixelu.

Pro konkrétní model z této práce v případě 1024 bytového bufferu tedy přibližně platí:

$$f_{MAX} = \frac{434000}{170 \cdot 3} \approx 850 \text{ snímků za sekundu},$$

přičemž Linky v současné době akceptují maximálně 50 snímků za sekundu [4], tudíž je rychlost dostatečná.

4.2.2 Síťové rozšíření

V rámci této práce proběhly dva druhy měření rychlosti síťového rozšíření: měření rychlosti celé obnovy jednoho snímku[4] a měření rychlosti samotného získávání dat z LinkyAPI. Oba testy jsou implementovány v jednom programu, který nachází v adresáři `code/test/networking_speedtest`.

Výsledkem jednoho z testů je doba trvání obnovy jednoho snímku. Výpis pixelů pro uživatele při testování není prováděn, je pomalý a zkresloval by výsledky. Doba trvání je vytištěna pro uživatele pomocí UART. Program byl spuštěn desetkrát a výsledky těchto běhů byly zprůměrovány. Výsledkem byl průměr 3359 ms, tedy více než tři sekundy na obnovu jednoho snímku. Tento čas je velmi nedostatečný pro animace, jelikož animace pak dosahuje snímkové frekvence přibližně 0.3 FPS, přičemž lidské oko potřebuje minimálně animace o snímkové frekvenci minimálně 10 FPS, aby vznikla iluze pohybu. Naměřené časy obnovy jednoho snímku je možné vidět v Tabulce 4.2.

Pokus	Čas obnovy [ms]
1.	3305
2.	3308
3.	2310
4.	4262
5.	2858
6.	3535
7.	2491
8.	4034
9.	4749
10.	2741
Průměr	3359

Tabulka 4.2: Měření doby trvání obnovy jednoho snímku.

[4] Jedná se o metodu `ModelController::refresh`, tedy včetně zpracování.

Při měření rychlosti pouhého získávání dat z LinkyAPI byla výsledkem doba trvání načítání v milisekundách a z ní spočítaná rychlost získávání dat v bytech za sekundu. Tyto informace jsou uživateli po pokusu vytištěny pomocí UART. Tento program byl také spuštěn desetkrát a výsledky těchto běhů byly zprůměrovány. Výsledkem je přibližně 2170 milisekund, rychlost získávání dat byla přibližně 5190 bytů za sekundu. Naměřené hodnoty se nachází v Tabulce 4.3.

Pokus	Čas obnovy [ms]	Rychlost přenosu [B/s]
1.	4.932	3305
2.	1.520	3308
3.	1.864	2310
4.	3.697	4262
5.	1.855	2858
6.	1.384	3535
7.	2.381	2491
8.	1.810	4034
9.	4.349	4749
10.	2.098	2741
Průměr	2.266	5190

Tabulka 4.3: Měření rychlosti samotného získávání dat.

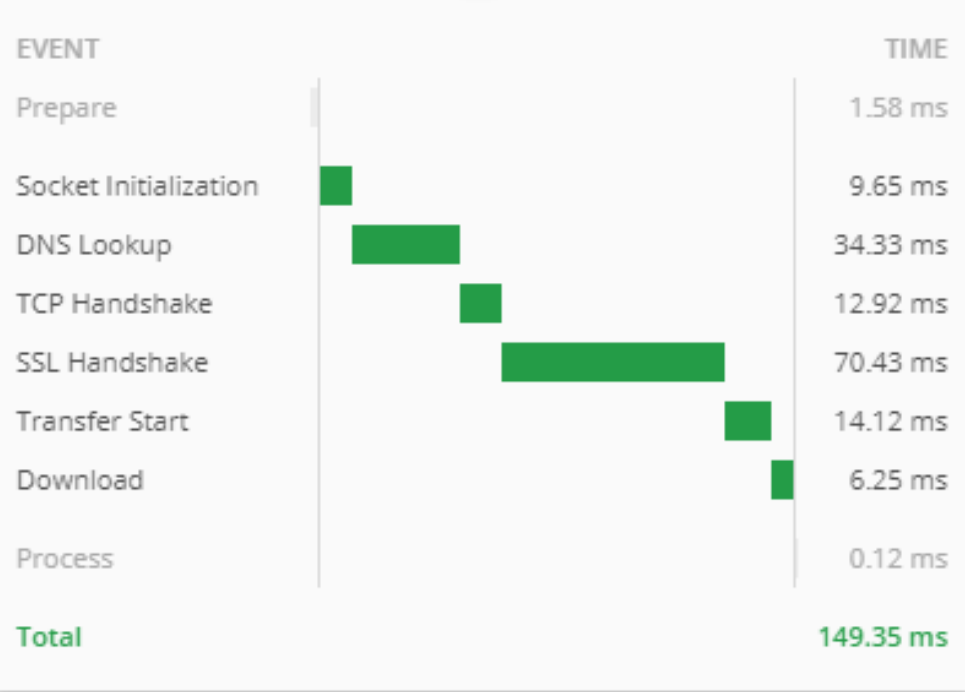
Implementace síťového rozšíření v této práci dokazuje, že je možné data z LinkyAPI stahovat, ale v praxi tato implementace není pro animace použitelná. V sekci 4.2.3 budou probrány návrhy na zlepšení konceptu síťové komunikace do budoucna, aby byla snímková frekvence maximalizována i při použití síťového rozšíření.

4.2.3 Návrh na zlepšení

Ze sekce 4.2.2 plyne, že síťové rozšíření implementované v této práci není v praxi použitelné pro přehrávání animací s aktuálním LinkyAPI. Tato sekce popíše možné směry zlepšení na straně LinkyAPI pro zvýšení frekvence při použití síťového rozšíření.

Pro lepší pochopení příčiny příliš dlouhé doby trvání získávání dat byl požadavek na LinkyAPI endpoint `DataState` proveden znovu, pomocí počítačového programu *Postman*. Postman je nástroj na testování API, který mimo jiné umožňuje analyzovat průběh požadavku – rozpadne jej na jednotlivé kroky a ukáže, který krok trval jaké množství času. Výsledek analýzy jednotlivých kroků požadavku na LinkyAPI lze vidět na Obrázku 4.1.

Z analýzy plyne, že nemalou část požadavku probíhá navazování SSL spojení. Část tohoto problému by šla vyřešit přesunem LinkyAPI (nebo alespoň



Obrázek 4.1: Analýza požadavku na LinkyAPI pomocí programu Postman.

některých jeho endpointů) na nezabezpečené HTTP. Odpadly by tak všechny části spojené s SSL, ale stále by muselo opakovaně docházet k navazování SSL spojení při obnově každého snímku. Vhodnější tedy bude opustit myšlenku opakovaného aktivního dotazování, tedy TCP spojení navázat pouze jednou, pasivně čekat na obrazová data od serveru a TCP spojení po každém snímku nezavírat.

Udržení TCP spojení je možné např. pomocí techniky *Long polling*, kdy je HTTP požadavku nastaven velmi dlouhý timeout. TCP spojení tak zůstává otevřené a server může aktivně posílat data klientovi. Jedná se ale o zastaralou technologii, vhodnější bude použít WebSockety. Websocket je protokol, který umožňuje obousměrnou komunikaci mezi klientem a serverem pomocí jediného TCP spojení.

Další možností by bylo implementovat celou komunikaci pouze na transportní vrstvě pomocí TCP socketu. Takové řešení ale dobře nezapadá do konceptu budoucího LinkyAPI [8] [7], jednalo by se spíše o jednoúčelový server pro modely. Websockety budou ale již v budoucím systému použity v implementaci webových simulátorů [7], které v současné době fungují podobně jako fyzické modely v této práci (na principu opakovaného aktivního dotazování na LinkyAPI), ale v novém LinkyAPI budou simulátory pasivní a budou dostávat data od serveru bez vyžádání [7]. Principiálně tedy bude vhodné chápat fyzický model jako druh simulátoru.

Dalším způsobem, jak ušetřit čas, je přesun zpracování dat pro model na serverovou část a jejich následné odeslání v binárním formátu *LMN* (viz sekci 3.4.3) či podobném formátu. Zpracováním se rozumí především úprava

dat instalace Linky na data pro model správné velikosti, tj. zodpovědnost komponenty *ImageProcessor*. Zcela by odpadla nutnost deserializovat a zpracování by se podstatně zrychlilo – oproti procesoru Tensilica Xtensa v ESP8266 a jeho 80 MHz frekvenci mají procesory PC dnes běžně frekvence v řádech jednotek GHz. Také by se radikálně snížilo zasílané množství dat – jednak použitím binárního formátu oproti textovému v současném LinkyAPI [4], jednak zasíláním správného počtu pixelů pro model (v této práci 170 pixelů oproti 1020 [2], tj. 6× méně pixelů).

Kapitola 5
Závěr

V této kapitole jsou shrnuty výstupy práce a náměty pro budoucí tvorbu fyzických modelů Linek.

5.1 Shrnutí výstupů

V rámci práce byl vytvořen vysokoúrovňový návrh systému pro modely instalace Linky (sekce 2.4). Ten obsahuje důležitou myšlenku jednoprůchodového zpracování, které je vhodné pro zařízení s nízkou RAM. Dále byly implementovány dvě varianty modelu, popsané v sekcích 2.6 a 2.7, a síťové rozšíření, popsané v sekci 3.5. Implementované varianty a síťové rozšíření sdílí část kódu, struktura projektu je popsána v sekci 3.1. Implementace ukázala použitelnost konceptu na různých platformách, některé komponenty byly dokonce přepoužívány u obou dvou variant i síťového rozšíření (viz sekci 3.1).

Správnost implementačních variant pak byla ověřena manuálními testy (sekce 4.1). Byla změřena rychlost některých komponent (sekce 4.2). Testování proběhlo úspěšně, ale měření ukázalo nepoužitelnost síťového rozšíření pro přenos animací (viz sekci 4.2.2) při současném LinkyAPI.

Pro tvorbu modelů je možné využít hotových implementací, nebo některé komponenty poupravit či vytvořit zcela novou implementaci podle vysokoúrovňového návrhu ze sekce 2.4. Velmi užitečným výstupem je implementace varianty B, neboť její pomocí je možné na modelu zobrazit jakoukoliv animaci z SD karty, navíc s velmi vysokou snímkovou frekvencí díky použití formátu LMN (viz sekci 3.4.3).

5.2 Náměty do budoucna

V této práci byl vytvořen základ, pomocí kterého je možné vytvářet modely instalace Linky na různých platformách. Offline modely lze provozovat na dostatečné úrovni, např. pomocí již hotové implementace varianty B (viz sekci 2.7). Budoucí posun modelů bude tedy směřovat k vytvoření funkční implementace síťového rozšíření a integraci budoucí verze LinkyAPI, která bude využívat WebSockety. Také bude vhodné do systému na server přesunout zpracování dat pro model, aby se snížila doba přenosu i zpracování dat. Návrhy

na budoucí změny serverové strany systému jsou podrobněji popsány v sekci 4.2.3.

Online model by měl také v budoucnu být koncipován jako další implementace, která bude obsahovat i zobrazení dat na model (přímo v ESP8266 nebo ESP32, viz sekci 2.5.6 a 3.5.3), bez nutnosti dalšího běžícího mikrokontroléru s jinou implementací. Odpadne tak čas potřebný ke komunikaci s další hardwarovou komponentou a celý systém se zjednoduší na jeden mikrokontrolér i v případě použití síťové varianty.

Příloha A

Seznam použité literatury

1. ČVUT. *Dejvické Vítězné náměstí oživí interaktivní fasáda Fakulty elektrotechnické ČVUT v Praze* [online]. Praha: ČVUT, c2015 [cit. 2021-08-10]. Dostupné z: `https://aktualne.cvut.cz/tiskove-zpravy/20160219-dejvicke-vitezne-namesti-ozivi-interaktivni-fasada-fakulty-elektrotechnicke`.
2. HYBLER, Jakub. Světelný design v kostce – Část 24: Světelná fasáda – projekt Linky. *Světlo: časopis pro světelnou techniku a osvětlování*. 2016, roč. 2016, č. 3, s. 5. ISSN 1212-0812.
3. VOJTĚCHOVSKÝ, Miloš. *Linky - Historie* [online]. Praha: ČVUT FEL, c2015-2018 [cit. 2021-01-09]. Dostupné z: `https://linky.fel.cvut.cz/About`.
4. FRYČ, J. *Popis rozhraní pro komunikaci s instalací LINKY* [online]. Praha: ČVUT FEL, c2015-2018 [cit. 2021-08-10]. Dostupné z: `https://linky.fel.cvut.cz/Api`.
5. *Světelná instalace Linky na fasádě FEL ČVUT v Dejvicích.* [Online]. Praha: ČVUT FEL, c2015-2018 [cit. 2021-08-13]. Dostupné z: `https://linky.fel.cvut.cz/img/linky01.jpg`.
6. MARTIN, Robert C.; GREENING, James; BROWN, Simon. *Clean Architecture: A Craftman's Guide to Software Structure and Design*. United States of America: Prentice Hall, 2017. ISBN 978-0-13-449416-6.
7. KOŠATA, Jiří. *Linky - Grafická knihovna*. Praha, 2019. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, Katedra softwarového inženýrství.
8. PLÍŠEK, Libor. *Linky - Jádro systému*. Praha, 2019. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, Katedra softwarového inženýrství.
9. GAMMA, Erich. *Design patterns: Elements of Reusable Object-Oriented Software*. Boston: Addison-Wesley, c1995. ISBN 02-016-3361-2.
10. LOUIS, Dirk; MEJZLÍK, Petr; VIRIUS, Miroslav. *Jazyky C a C podle normy ANSI/ISO: kompletní kapesní průvodce*. Praha: Grada, 1999. ISBN 80-716-9631-5.

11. FLEXFIRE LEDS. *Comparison between 3528 LEDs and 5050 LEDs: What are SMD 3528 and SMD 5050 LEDs?* [Online]. Costa Mesa (California): Flexfire LEDs., c2021 [cit. 2021-08-10]. Dostupné z: `https://www.flexfireleds.com/comparison-between-3528-leds-and-5050-leds/`.
12. WORLDSEMI. *WS2812B datasheet* [online]. GuangDong: WORLDSEMI, [2017] [cit. 2021-01-09]. Dostupné z: `https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf`.
13. WORLDSEMI. *Digital RGB LED* [online]. Dongguan: WORLDSEMI CO., LIMITED, c2017 [cit. 2021-08-10]. Dostupné z: `http://www.worldsemi.com/solution/list-4-1.html`.
14. SHENZHEN LED COLOR. *SK6812 SPECIFICATION: INTEGRATED LIGHT SOURCE INTELLIGENT CONTROL OF CHIP-ON-TOP SMD TYPE LED* [online]. Shenzen: Shenzhen LED Color, c2013-2019 [cit. 2021-08-10]. Dostupné z: `http://www.szledcolor.com/download/SK6812%20LED.pdf`.
15. SOLVANG, Henrik Olav. *Instruction Set for Bit-Banging Operations*. Gløshaugen, Trondheim, 2016. Diplomová práce. Norwegian University of Science a Technology.
16. DASGUPTA, Ranjan; HALDER, A; CHEPADA, J; DASH, N P. Implementation of MCU Invariant I2C Slave Driver Using Bit Banging. In: *ICONS 2011: The Sixth International Conference on Systems Proceedings*. St. Maarten (Netherlands): IARIA, 2011, s. 1. ISBN 978-1-61208-114-4.
17. STROUSTRUP, Bjarne. *The C++ Programming Language*. 4th edition. United States of America: Addison-Wesley, 2013. ISBN 978-0-321-56384-2.
18. STALLMAN, Richard M. *Using the GNU Compiler Collection* [online]. Boston: Free Software Foundation, 2021 [cit. 2021-08-10]. Dostupné z: `https://gcc.gnu.org/onlinedocs/gcc-11.2.0/gcc.pdf`.
19. CATSOULIS, John. *Designing embedded hardware*. 2nd ed. Sebastopol: O'Reilly, 2005. ISBN 978-0-596-00755-3.
20. STMICROELECTRONICS. *STM32F072x8 STM32F072xB Datasheet* [online]. Geneva: STMicroelectronics, c2021 [cit. 2021-08-10]. Dostupné z: `https://www.st.com/resource/en/datasheet/stm32f072rb.pdf`.
21. STMICROELECTRONICS. *STM32F0x1/STM32F0x2/STM32F0x8 Reference Manual* [online]. Geneva: STMicroelectronics, c2021 [cit. 2021-08-10]. Dostupné z: `https://www.st.com/resource/en/reference_manual/dm00031936-stm32f0x1stm32f0x2stm32f0x8-advanced-armbased-32bit-mcus-stmicroelectronics.pdf`.
22. VYŠINSKÝ, Tomáš. *PWM regulátor* [online]. Praha: ČVUT FEL [cit. 2021-08-11]. Dostupné z: `https://www.vovcr.cz/odz/tech/586/page18.html`.

23. COLMAN, Andrew M. *Dictionary of psychology*. 4th ed. Oxford: Oxford University Press, 2015. ISBN 978-0-199-65768-1.

24. SYNEK, Svatopluk; SKORKOVSKÁ, Šárka. *Fyziologie oka a vidění*. 2., dopl. a přeprac. vyd. Praha: Grada, 2014. ISBN 978-80-247-3992-2.

25. DHAKER, Piyu. *Introduction to SPI Interface* [online]. Wilmington (Massachusetts): Analog Devices, c1995-2021 [cit. 2021-08-11]. Dostupné z: `https://www.analog.com/media/en/analog-dialogue/volume-52/number-3/introduction-to-spi-interface.pdf`.

26. MICROCHIP TECHNOLOGY. *Overview and Use of the PICmicro Serial Peripheral Interface* [online]. Chandler (Arizona): Microchip Technology, c1998-2021 [cit. 2021-08-10]. Dostupné z: `http://ww1.microchip.com/downloads/en/devicedoc/spi.pdf`.

27. ESPRESSIF SYSTEMS. *ESP8266EX - Datasheet* [online]. Shanghai: Espressif Systems, c2021 [cit. 2021-08-10]. Dostupné z: `https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf`.

28. ARDUINO. *Arduino MEGA 2560 & Genuino MEGA 2560* [online]. Monza: Arduino s.r.l., c2021 [cit. 2021-01-09]. Dostupné z: `https://www.arduino.cc/en/Main/Arduino_BoardMega2560`.

29. LEWIS, Jerad. *Common Inter-IC Digital Interfaces for Audio Data Transfer* [online]. Wilmington (Massachusetts): Analog Devices, c1995-2021 [cit. 2021-08-10]. Dostupné z: `https://www.analog.com/media/en/technical-documentation/technical-articles/MS-2275.pdf`.

30. ADAFRUIT. *UDA1334ATS: Low power audio DAC with PLL* [online]. New York: Adafruit [cit. 2021-08-11]. Dostupné z: `https://cdn-shop.adafruit.com/product-files/3678/UDA1334ATS.pdf`.

31. ESPRESSIF SYSTEMS. *ESP32 Series - Datasheet* [online]. Shanghai: Espressif Systems, c2021 [cit. 2021-08-10]. Dostupné z: `https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf`.

32. ADAFRUIT. *Adafruit NeoPixel Library* [online]. Adafruit [cit. 2021-08-10]. Dostupné z: `https://github.com/adafruit/Adafruit_NeoPixel/blob/master/README.md`.

33. FASTLED. *FastLED documentation* [online]. [2010] [cit. 2021-01-09]. Dostupné z: `https://github.com/FastLED/FastLED/wiki/Overview`.

34. STMICROELECTRONICS. *STM32F042x4 STM32F042x6* [online]. Rev 5. Amsterdam: STMicroelectronics, 2017 [cit. 2021-08-13]. Dostupné z: `https://www.st.com/resource/en/datasheet/stm32f042f6.pdf`.

35. STMICROELECTRONICS. *STM32 Nucleo Boards* [online]. Geneva: STMicroelectronics, c2021 [cit. 2021-08-10]. Dostupné z: `https://www.st.com/en/evaluation-tools/stm32-nucleo-boards.html`.

36. STMICROELECTRONICS. *NUCLEO-XXXXRX NUCLEO-XXXXRX-P: Data Brief* [online]. Geneva: STMicroelectronics, c2021 [cit. 2021-08-10]. Dostupné z: https://www.st.com/resource/en/data_brief/nucleo-f072rb.pdf.

37. STROUSTRUP, Bjarne. *Bjarne Stroustrup's FAQ* [online]. Bjarne Stroustrup [cit. 2021-08-10]. Dostupné z: https://www.stroustrup.com/bs_faq.html.

38. BARRAGÁN, Hernando. *Wiring FAQ* [online]. Italy: Hernando Barragán, [2003] [cit. 2021-01-09]. Dostupné z: http://wiring.org.co/faq.html.

39. STMICROELECTRONICS. *STM32 32-bit Arm Cortex MCUs* [online]. Geneva: STMicroelectronics, c2021 [cit. 2021-08-10]. Dostupné z: https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html.

40. STMICROELECTRONICS. *STM32F0 Series* [online]. Geneva: STMicroelectronics, c2021 [cit. 2021-08-10]. Dostupné z: https://www.st.com/content/st_com/en/products/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus/stm32-mainstream-mcus/stm32f0-series.html.

41. STMICROELECTRONICS. *STM32F0x2* [online]. Geneva: STMicroelectronics, c2021 [cit. 2021-08-10]. Dostupné z: https://www.st.com/en/microcontrollers-microprocessors/stm32f0x2.html.

42. STMICROELECTRONICS. *STM32CubeMX for STM32 configuration and initialization C code generation - User manual* [online]. Geneva: STMicroelectronics, c2021 [cit. 2021-08-10]. Dostupné z: https://www.st.com/resource/en/user_manual/dm00104712-stm32cubemx-for-stm32-configuration-and-initialization-c-code-generation-stmicroelectronics.pdf.

43. STMICROELECTRONICS. *STM32CubeIDE user guide* [online]. Geneva: STMicroelectronics, c2021 [cit. 2021-08-10]. Dostupné z: https://www.st.com/resource/en/user_manual/dm00629856-stm32cubeide-user-guide-stmicroelectronics.pdf.

44. STMICROELECTRONICS. *STM32CubeProgrammer software description* [online]. Geneva: STMicroelectronics, c2021 [cit. 2021-08-10]. Dostupné z: https://www.st.com/resource/en/user_manual/dm00403500-stm32cubeprogrammer-software-description-stmicroelectronics.pdf.

45. STMICROELECTRONICS. *Free Integrated Development Environment from STMicroelectronics Further Expands Popular STM32Cube Microcontroller Ecosystem* [online]. Geneva: STMicroelectronics, c2021 [cit. 2021-08-10]. Dostupné z: https://www.st.com/content/st_com/en/about/media-center/press-item.html/p4154.html.

46. ARM LIMITED. *An introduction to Arm Mbed OS 6* [online]. Cambridge (Velká Británie): Arm Limited, c2021 [cit. 2021-08-10]. Dostupné z: `https://os.mbed.com/docs/mbed-os/v6.12/introduction/index.html`.

47. ARM LIMITED. *Mbed: Free open source IoT OS and development tools from Arm* [online]. Cambridge (Velká Británie): Arm Limited, c2021 [cit. 2021-08-10]. Dostupné z: `https://os.mbed.com/`.

48. ARM LIMITED. *GNU Arm Embedded Toolchain* [online]. Cambridge (Velká Británie): Arm Limited, c1995-2021 [cit. 2021-08-10]. Dostupné z: `https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm`.

49. FREE SOFTWARE FOUNDATION. *Cross-Compilation* [online]. Boston: Free Software Foundation, c2020 [cit. 2021-08-10]. Dostupné z: `https://www.gnu.org/savannah-checkouts/gnu/automake/manual/html_node/Cross_002dCompilation.html`.

50. ARDUINO. *What is Arduino?* [Online]. Monza: Arduino s.r.l., c2021 [cit. 2021-08-10]. Dostupné z: `https://www.arduino.cc/en/Guide/Introduction`.

51. KUSHNER, David. *The making of Arduino: How five friends engineered a small circuit board that's taking the DIY world by storm* [online]. New Jersey: IEEE, c2021 [cit. 2021-08-11]. Dostupné z: `https://spectrum.ieee.org/the-making-of-arduino`.

52. ARDUINO. *Arduino Software (IDE)* [online]. Monza: Arduino, c2021 [cit. 2021-08-11]. Dostupné z: `https://www.arduino.cc/en/guide/environment`.

53. ADAFRUIT. *Arduino Comparison Chart* [online]. New York: Adafruit [cit. 2021-08-11]. Dostupné z: `https://learn.adafruit.com/adafruit-arduino-selection-guide/arduino-comparison-chart`.

54. ARDUINO. *Writing a Library for Arduino* [online]. Monza: Arduino, c2021 [cit. 2021-08-11]. Dostupné z: `https://www.arduino.cc/en/Hacking/libraryTutorial`.

55. MICROSOFT. *Visual Studio Code: Code Editing. Redefined* [online]. Washington: Microsoft, c2021 [cit. 2021-08-12]. Dostupné z: `https://code.visualstudio.com/`.

56. CMAKE. *About CMake* [online]. Https://cmake.org/: CMake, c2021 [cit. 2021-08-12]. Dostupné z: `https://cmake.org/overview/`.

57. SCHWARTZ, Marco. *Internet of Things with ESP8266*. Birmingham: Packt Publishing, July 2016. ISBN 978-1-78646-802-4.

58. CADENCE. *Cadence to Acquire Tensilica* [online]. San Jose (California): Cadence, c2021 [cit. 2021-08-10]. Dostupné z: `https://ip.cadence.com/news/427/330/Cadence-to-Acquire-Tensilica`.

59. ESPRESSIF SYSTEMS. *What is ESP-AT* [online]. Shanghai: Espressif Systems, c2020 [cit. 2021-08-12]. Dostupné z: https://docs.espressif.com/projects/esp-at/en/latest/Get_Started/What_is_ESP-AT.html.

60. ESPRESSIF SYSTEMS. *ESP8266 AT Instruction Set* [online]. Shanghai: Espressif, c2021 [cit. 2021-01-09]. Dostupné z: https://www.espressif.com/sites/default/files/documentation/4a-esp8266_at_instruction_set_en.pdf.

61. ESCAPEQUOTES. *WeMos D1 mini pins and diagram* [online]. c2021 [cit. 2021-08-12]. Dostupné z: https://escapequotes.net/esp8266-wemos-d1-mini-pins-and-diagram/.

62. ESP8266 COMMUNITY FORUM. *ESP8266 core for Arduino* [online]. San Francisco: GitHub, c2021 [cit. 2021-08-12]. Dostupné z: https://github.com/esp8266/Arduino.

63. ESPRESSIF SYSTEMS. *ESP8266 Non-OS SDK: API Reference* [online]. Shanghai: Espressif Systems, c2021 [cit. 2021-08-12]. Dostupné z: https://www.espressif.com/sites/default/files/documentation/2c-esp8266_non_os_sdk_api_reference_en.pdf.

64. ESPRESSIF SYSTEMS. *ESP8266 nonOS SDK* [online]. San Francisco: GitHub, c2021 [cit. 2021-08-12]. Dostupné z: https://github.com/espressif/ESP8266_NONOS_SDK.

65. ESPRESSIF SYSTEMS. *ESP8266 RTOS SDK User Manual* [online]. Shanghai: Espressif Systems, c2021 [cit. 2021-08-12]. Dostupné z: https://docs.espressif.com/_/downloads/esp8266-rtos-sdk/en/release-v3.3/pdf/.

66. BARR, Michael. *Introduction to Preemptive Multitasking* [online]. Cambridge (Massachuttes): AspenCore [cit. 2021-08-12]. Dostupné z: https://www.embedded.com/introduction-to-preemptive-multitasking/.

67. ESPRESSIF SYSTEMS. *FreeRTOS* [online]. Shanghai: Espressif Systems, c2016-2021 [cit. 2021-08-10]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html.

68. ESPRESSIF SYSTEMS. *Get Started* [online]. Shanghai: Espressif Systems, c2016-2021 [cit. 2021-08-10]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html.

69. ESPRESSIF SYSTEMS. *ESP-IDF API Reference* [online]. Shanghai: Espressif Systems, c2016-2021 [cit. 2021-08-10]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/index.html.

70. ESPRESSIF SYSTEMS. *Build System (Legacy GNU Make)* [online]. Shanghai: Espressif Systems, c2016-2021 [cit. 2021-08-10]. Dostupné z: `https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/build-system-legacy.html`.

71. ESPRESSIF SYSTEMS. *Build System* [online]. Shanghai: Espressif Systems, c2016-2021 [cit. 2021-08-10]. Dostupné z: `https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/build-system.html`.

72. GOOGLE. *Secure your site with HTTPS* [online]. Mountain View (California): Google [cit. 2021-08-12]. Dostupné z: `https://developers.google.com/search/docs/advanced/security/https?hl=en%5C&visit_id=637643544089294350-2280798395%5C&rd=1`.

73. ESPRESSIF SYSTEMS. *ESP-WOLFSSL README* [online]. Shanghai: Espressif Systems, [2019] [cit. 2021-08-10]. Dostupné z: `https://github.com/espressif/esp-wolfssl/blob/master/README.md`.

74. REYES, Tim; HYMEL, Shawn. *What Is a Logic Analyzer?* [Online]. South San Francisco: Saleae, [2018] [cit. 2021-08-10]. Dostupné z: `https://articles.saleae.com/logic-analyzers/what-is-a-logic-analyzer`.

Příloha B

Seznam použitých zkratek

API Application Programming Interface. 1, 5, 6

CAN Controller Area Network. 31

CLI Command Line Interface. 25, 27

CMYK Cyan Magenta Yellow Key. 38

CPU Central Processing Unit. 11, 13, 14, 15

CS Chip Select. 13, 29

DAC Digital-to-Analog Converter. 14

DMA Direct Memory Access. 11, 12, 13, 14, 15, 16

FEL Fakulta elektrotechnická. 15

GPIO General Purpose Input-Output. 10, 11

HTTP Hypertext Transfer Protocol. 5

HTTPS Hypertext Transfer Protocol Secure. 5

I^2S Inter-IC Sound. 14, 15, 30

IDE Integrated Developement Environment. 15, 22, 25, 26, 27, 28, 30, 35, 37

JSON JavaScript Object Notation. 5, 8

LMN Linky Model Native. 21, 27, 28, 42, 45

LPE Laboratoře průmyslové elektroniky a senzorů. 15

MISO Master In, Slave Out. 13

MOSI Master Out, Slave In. 13, 14

MSB Most Significant Bit. 10

OOP Object Oriented Programming. 22

PC Personal Computer. 43

PWM Pulse Width Modulation. 12, 13, 14, 15, 16

RAM Random Access Memory. 12, 13, 19, 26, 32, 45

RGB Red Green Blue. 10, 16, 19, 38

SCLK Serial Clock. 13, 14

SD Secure Digital. viii, 27, 35, 36, 37, 39, 45

SDI Serial Data In. 13

SDK Software Developement Kit. 30, 31, 32

SDO Serial Data Out. 13

SPI Serial Peripheral Interface. 12, 13, 14, 15, 29, 30, 31, 36, 38

SSID Service Set Identifier. 37, 38

SSL Secure Socket Layer. 29, 32, 41, 42

STL Standard Template Library. 21

SW Software. 25, 31, 38

TCP Transmission Control Protocol. 29, 42

TLS Transport Layer Security. 32

UART Universal Asynchronous Receiver-Transmitter. 29, 31, 37, 38, 40, 41

USB Universal Serial Bus. 30, 36

WSL Windows Subsystem for Linux. 28

Příloha C

Seznam přiložených souborů

K této práci byly přiloženy 3 soubory:

- `muzatmat_BP2021_projekt.zip` – archiv, obsahuje kořenový adresář projektu, struktura projektu dle sekce 3.1.
- `muzatmat_BP2021_bin.zip` – archiv, obsahuje zkompilované programy a README.md soubor s pokyny k flashování.
- `muzatmat_BP2021_video.mp4` – video s testováním animací o vysokých rychlostech s variantou B.