Středoškolská technika 2024

Setkání a prezentace prací středoškolských studentů na ČVUT

Světlo ovládané DMX protokolem

Jan Červenka

Střední průmyslová škola a Vyšší odborná škola, Písek, Karla Čapka 402; se sídlem: Karla Čapka 402, 397 01 Písek

Obsah

1 Úvod

1.1 Ovládání světla DMX protokolem

Cílem práce je vytvořit DPS[i] pro příjem dat protokolu DMX512, jejich přečtení a zpracování pro následné zobrazení na světelném zdroji. Součástí musí být také možnost nastavení uživatelských parametrů a případné manuální nastavení.

1.2 Teoretický rozbor

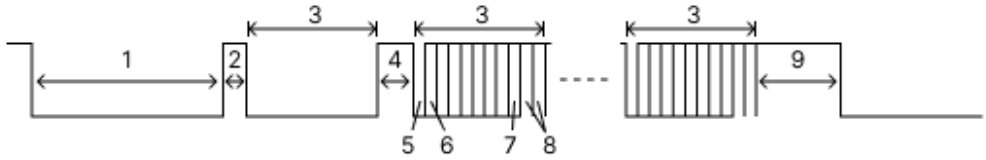
1.2.1 DMX512

Protokol DMX512 je standardizovaný digitální komunikační protokol vyvinutý pro řízení osvětlovací techniky. Umožnuje přenos dat mezi řídícím zařízením a osvětlovacími prvky, jako jsou stmívače, otočné hlavice, LED PAR a jiné, přes kabely z kroucené dvojlinky standardně zakončené XLR konektorem. DMX512 je založený na principu sériového přenosu dat označovaného jako RS485. Řídící zařízení vysílá opakovaně hodnoty všech kanálů, a to i když nebyla provedena žádná změna. Protokol se stal standardem v této oblasti díky své spolehlivosti, jednoduchosti a širokému rozsahu implementace v různých typech světel.

DMX512 má rozsah 512 kanálů, tedy až 512 jednoduchých (jednokanálových) osvětlovacích prvků. Z těchto 512 kanálů je každému přiřazena jedna z 256 hodnot (0 - 255). Tyto hodnoty mohou ovládat různé nastavení světel (stmívání, barva, zoom, rotace, ale i funkce zapnutí lampy, reset, reverzace směru otáčení, …).

Po sběrnici se posílají data bez adresy. Každé zařízení na sběrnici má nastavenou svou počáteční adresu a od té přečte požadovaný počet kanálů/rámců. Na sběrnici může být zapojeno i více zařízení se stejnou počáteční adresou, pokud bude jejich činnost stejná.

Protokol DMX512 má stanovenou přenosovou rychlost 250 kBit/s, přičemž časování znázorňuje *Obrázek 1.1: Časování protokolu DMX512* a příslušné hodnoty uvádí *Tabulka 1.1: Příslušné hodnoty časování protokolu DMX512*. Přenos probíhá asynchronně, přičemž jeho začátek je synchronizován pomocí „Break“ signálu (Reset), který musí trvat minimálně 88 µs, následovaný synchronizační mezerou MAB (Mark After Break) s vysokou úrovní a minimální délkou trvání 8 µs. Za tímto signálem následuje první poslaný rámec (start code), za kterým následuje zbývajících 512 datových rámců. Každý rámec (jednotlivý přenesený bajt) se skládá z jednoho start bitu, osmi datových bitů bez parity[ii] a dvou stop bitů. Mezi jednotlivými rámci mohou být mezery MTBF (Mark Time Between Frames) a MTBP (Mark Time Between Packet) v délce nejvíce 1 s. Tyto mezery se používaly dříve, dnes jsou z důvodu zrychlení přenosu téměř nulové (některá vysílací zařízení využívají tuto mezeru pro výpočet následně přenášených dat), případně je využita pro přenos doplňujících dat.



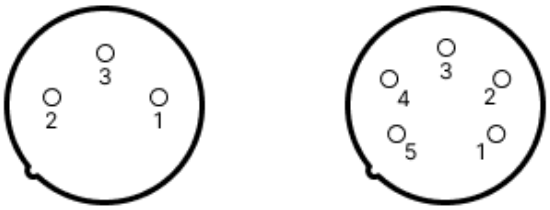
Obrázek 1.1: Časování protokolu DMX512

Číslo	Popis	Min. [µs]	Typ. [µs]	Max.

1	Break (Reset)	88	176	1 s
2	MAB (synchronizační mezera)	8	-	1 s
3	Rámec (jeden kanál)	43,12	44	44,48 µs
4	MTBF (mezera mezi rámci)	0	0	1 s
5	Start bit	3.92	4	4.08 µs
6	LSB (první datový bit)	3.92	4	4.08 µs
7	MSB (poslední datový bit)	3.92	4	4.08 µs
8	Stop bit (po 2)	3.92	4	4.08 µs
9	MTBP (mezera mezi pakety)	0	0	1 s

Tabulka 1.1: Příslušné hodnoty časování protokolu DMX512

Jako standardní konektor se využívá pětipinový XLR konektor, ovšem v praxi se spíše využívá třípinový XLR konektor, protože se může využít u mikrofonních kabelů, pokud tedy mají zapojené stínění na konektoru na pin 1. Kdyby bylo připojeno na kostru konektoru, mohlo by u některých zařízení dojít ke zkratu, nebo nevyzpytatelnému chování. Zapojení konektoru XLR 5-pin a XLR 3-pin znázorňuje *Obrázek 1.2: Piny na 3-pin a 5-pin XLR* konektorech a *Tabulka 1.2: Zapojení kabelu ke konektoru XLR*. Na konec každé sběrnice se umisťuje zakončovací 120Ω odpor (terminátor), což vyplývá z požadavků zapojení MAX485 (1).



Obrázek 1.2: Piny na 3-pin a 5-pin XLR konektorech

Pin	Barva vodiče	Signál
1	Stínění/černá	GND
2	Černá/Červená	Data-
3	Bílá	Data+
4 (nepoužívá se)	Zelená	Data2-
5 (nepoužívá se)	Červená/Modrá	Data2+

Tabulka 1.2: Zapojení kabelu ke konektoru XLR

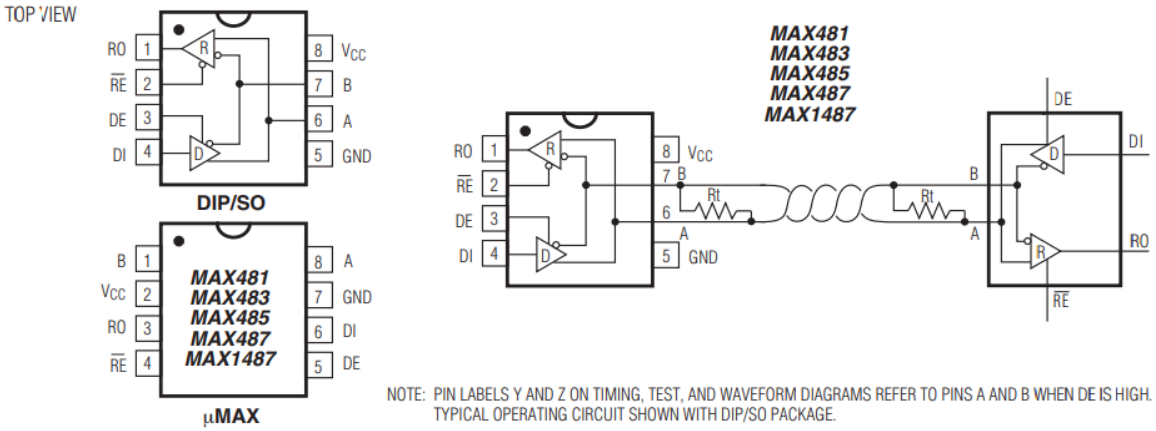
1.2.2 MAX485

MAX485 je integrovaný obvod používaný jako převodník pro komunikaci v sériových sběrnicích, jako je například RS485. Má 8 pinů (znázorňuje *Obrázek 1.3: Zapojení 1-1 MAX485*) a fyzicky se vyskytuje v mnoha druzích pouzder, například: 8 PDIP, 8 SO, 8 CERDIP. Jeho funkcí je umožnit přenos dat mezi různými zařízeními prostřednictvím kroucené dvojlinky.

MAX485 je schopen přenášet data rychlostí až 2,5 Mbps. Jeho spotřeba proudu se pohybuje v rozmezí 120 µA až 900 µA (typicky 500 µA), a to i při plném zatížení. V režimu nízkého proudu vypnutí spotřebuje pouze 0,1 µA, což přispívá k úspornému provozu. Ovladač je chráněn proti zkratu a nadměrnému přehřátí, čímž zajišťuje

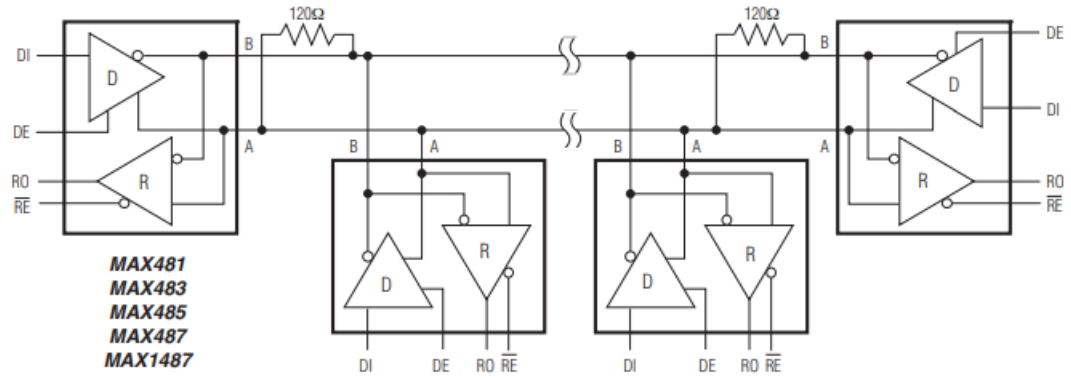
spolehlivý provoz v různých podmínkách. Napájen může být stejnosměrným zdrojem napětí 4,75 V až 5,25 V. Může být napájen maximálně 12 V, ovšem při takovém napětí už výrobce nezaručuje správnost signálu.

Při zapojení 1-1 (*Obrázek 1.3: Zapojení 1-1 MAX485 (1)*) je jeden IC[iii] zapojený jako vysílač a druhý jako přijímač. Vzájemně jsou propojeny kroucenou dvojlinkou se stíněním (GND), na níž se nachází zakončovací odpor [Rt], který se využívá k minimalizaci odrazů a rušení signálu na koncích linky.



Obrázek 1.3: Zapojení 1-1 MAX485 (1)

Pro účely přenosu signálu DMX512 se využívá zapojení 1-n (*Obrázek 1.4: Zapojení 1-n MAX485*), které je velmi podobné zapojení 1-1, jen jsou zde paralelně ke sběrnici přidány další IC[iii] MAX485 (bez zakončovacího odporu [Rt]), sloužící jako přijímače, kterých může být neomezené množství. Vysílací může být pouze jeden. Zakončovací odpor [Rt] je zapojen pouze na vysílacím IC[iii] a posledním přijímacím na sběrnici.



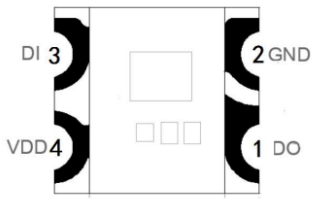
Obrázek 1.4: Zapojení 1-n MAX485 (1)

1.2.3 WS2812B

WS2812B je inteligentní LED světelný zdroj. Tento světelný zdroj kombinuje řídicí obvod s RGB[iv] diodami v jednom balení (2020, nebo 5050). Uvnitř se nachází integrovaný obvod WS2812B a 3 LED diody (červená, zelená a modrá). Komunikace je zajištěna protokolem NZR, který využívá jednodrátovou cestu. Samotný čip má 4 piny (Vcc, GND, Data in a Data out). Každá barva na jednom pixelu může být nastavena až na 256 hodnot (1. hodnota je vypnuto) Může tedy zobrazit až 16 777 216 jednotlivých

barev. Obnovovací frekvence je až 2 kHz. Doporučené napájení je od 3,3 V do 5,5 V a proud při maximálním rozsvícení dosahuje maximálně 18 mA, standardně 12 mA.

PIN Configuration

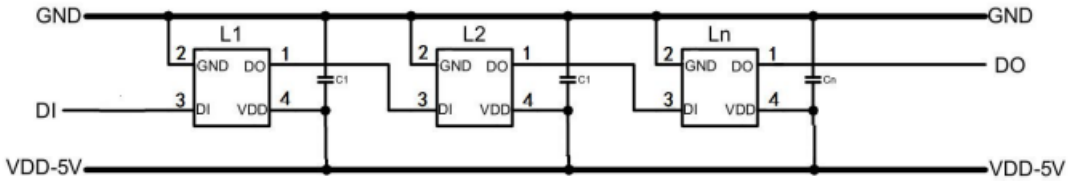


PIN Function

NO.	Symbol	PIN	Function description
1	DO	DATA OUT	Control data signal output
2	GND	GROUND	Ground, data & power grounding
3	DI	DATA IN	Control data signal input
4	VDD	POWER SUPPLY	Power supply

Obrázek 1.5: Rozložení pinů na WS2812B (2)

Typical Application Circuit



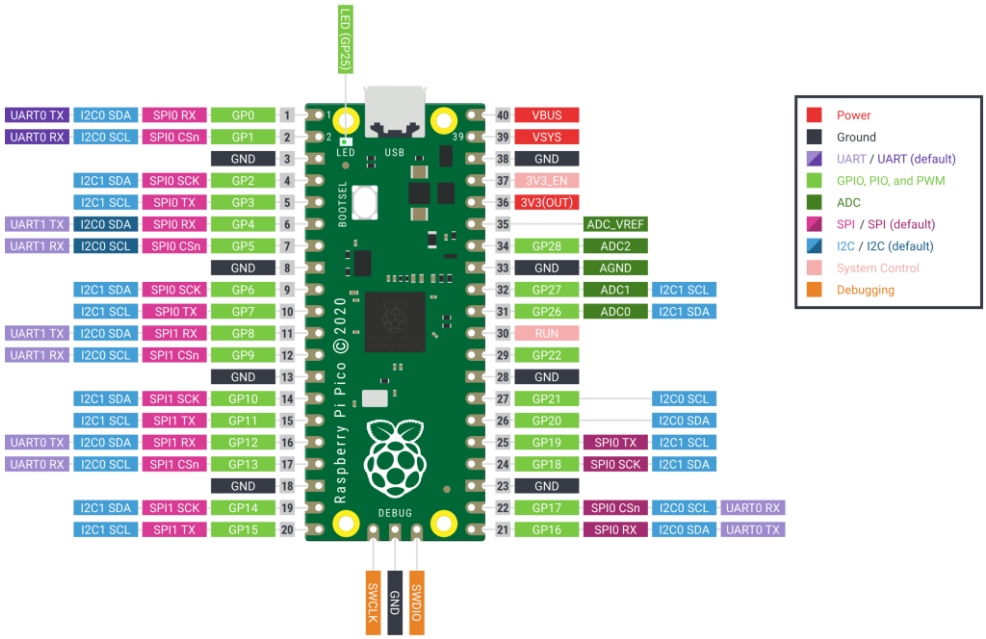
Remarks: C1 is the filter capacitor for VDD, its value of 100nF.

Obrázek 1.6: Typické zapojení WS2812B (2)

1.2.4 Raspberry Pi Pico

Raspberry Pi Pico je cenově dostupná vývojová deska mikrokontroléru, založeného na čipu RP2040, která je vybavena dvoujádrovým procesorem Arm Cortex M0+, flexibilními hodinami s možností nastavení na vysokou rychlost až 133 MHz, 264kB pamětí SRAM. Dále disponuje vestavěnou 2MB flash pamětí pro ukládání kódu a dat, s kterou se dá pracovat podobně jako s USB flash diskem.

Tento mikrokontrolér nabízí bohatou sadu funkcí, včetně 26 multifunkčních GPIO pinů, rozhraní USB pro programování komunikaci a napájení, 2 SPI, 2 I2C, 2 UART, 16 PWM a mnoho dalších. Podporuje různé programovací jazyky, jako je MicroPython a C/C++. Raspberry Pi Pico je ideální pro vývoj široké škály projektů, od jednoduchých až po složitější aplikace. Díky své malé velikosti a vysokém výkonu je snadno integrovatelný do různých elektronických zařízení. Existuje také varianta Raspberry Pi Pico W, která je vybavena navíc wifi čipem, což rozšiřuje možnosti komunikace a připojení k síti.



Obrázek 1.7: Rozložení a funkce jednotlivých pinů Raspberry Pi Pico (3)

1.2.5 OLED ssd1306 128x64 0,96“

OLED displej SSD1306 128x64 je kompaktní, nízkoenergetický displej s rozlišením 128x64 pixelů. Tento displej využívá organické LED (OLED) technologie, která umožňuje vysoký kontrast a jasný obraz i při nízké spotřebě energie. Displej je řízen čipem SSD1306, který zajišťuje snadnou integraci a ovládání pomocí sériového rozhraní, jako je I2C nebo SPI. Jeho výhody zahrnují vysoký kontrast, rychlou odezvu a nízkou spotřebu energie. To z něj činí populární volbou pro různé elektronické projekty a zařízení. Dodává se v modré barvě, bílé barvě a kombinované modré se žlutým pruhem v horní části. Může být napájen 3,2 V až 15 V. Spotřeba se při běžném využití (3,3V napájení) pohybuje kolem 5 mA, při plném svitu 20,7 mA, při nulovém svitu 469 µA a při programovém vypnutí 26 µA.

2 Hardware

Tento oddíl popisuje zvolené součástky, jejich zapojení a zdůvodňuje jejich zvolení.

2.1 Zvolený mikrokontrolér

Pro tuto aplikaci jsem zvolil mikrokontrolér Raspberry Pi Pico. Druhou možností byl MCU[v] Arduino Nano, který je sice rozměrově menší a již dlouhou dobu je standardem mezi mikrokontroléry, ovšem nenabízí tolik možností jako Raspberry Pi Pico a je o něco dražší.

Každý DMX universe potřebuje minimálně 512B paměti RAM, Pi Pico disponuje 264kB RAM pamětí, což je dostatečné pro veškeré očekávané operace. Příjem DMX protokolu je také velmi citlivý na časování, které je téměř vždy řešené interrupty.

2.2 MAX485

MAX485 je standard pro příjem DMX signálu, alternativním IC[iii] by mohl být sn75176, který má nižší napájení a rychlejší přenosovou rychlost, konkrétně 10 MB/s, což při přenosové rychlosti DMX (250 kBit/s) nehraje roli. Zároveň se dá zaměnit za MAX485, to znamená, že by nevadilo mít vysílací zařízení s MAX485 a přijímací s sn75176. Oba mají stejné rozložení pinů a u obou se doporučuje 120Ω zakončovací rezistor. Rozhodující je nejspíše cena, ač je rozdíl v jednotkách korun, ve velkém množství už je cenový rozdíl znatelný.

Při vyhledávání informací k tomuto tématu jsem objevil velmi oblíbený modul obsahující právě MAX485. Při porovnání s dokumentací MAX485 (1) jsem zjistil, že tento levný modul pomocí 10k pull-up rezistorů udržuje na vysoké hodnotě piny RO, RE, DE a DI, díky čemuž je defaultně nastaven jako vysílací člen. Zároveň je mezi piny A a B 120Ω rezistor, což by bylo v pořádku, kdyby bylo počítáno se zapojením podle schématu vyobrazeného výše (viz *Obrázek 1.3: Zapojení 1-1 MAX485*). Ovšem DMX počítá s více zařízeními na jedné sběrnici. Proto se také tento „terminační“ odpor nachází v samostatných XLR M konektorech a zapojuje se na konec signálové cesty.

2.3 Odpory u MAX485

Z dokumentace k MAX485 (1) je zřejmé, že se doporučuje 120Ω zakončovací rezistor na obě strany sběrnice mezi A a B. Ve skutečnosti je u vysílače ideální 133Ω.

Dále se z dokumentace (1) dá vyčíst, že pro logickou „1“ by mělo platit $V_{AB} \geq +200$ mV a pro logickou „0“ $V_{AB} \leq -200$ mV. Pokud bude platit: $-200 \text{ mV} < V_{AB} < +200$ mV, nastane hazardní stav a IC[iii] by náhodně volil, zda bude počítat s log. „0“, či „1“. MAX485 je na tuto situaci nastaven tak, aby se automaticky definovala log. „1“.

Pro zapojení jako přijímač jsem použil 100kΩ rezistor jako pull-up pro A a jako pull-down pro B, čímž zajistím korektně definovanou logickou hodnotu v době, kdy není definována vstupem DMX. Zároveň jsem přidal 100Ω rezistor v sérii k A i B, pro ochranu před nežádoucími přechodovými jevy.

2.4 Displej

Displej je potřeba pro zobrazení nastavovaných hodnot, jako je třeba adresa, mód, a další. U starších či levnějších světel se využívají čtyři sedmi-segmentové displeje, které

postačují pro zobrazení číselných hodnot a základních hesel (např.: Adr jako adresa). Při velmi členitém menu je nastavení světla složité, a pokud je potřeba nastavit více různých druhů menu, proces nastavení se prodlužuje. Proto, i za dvounásobnou cenu a komplexnější nastavení, je ideální variantou displej s více řádky, jako je například OLED SSD1306. Jeho využitím se zrychlí a zpřehlední nastavení světla.

V této aplikaci využívám I2C komunikační protokol pro spojení s MCU[v]. Pin SCL představuje hodinový signál a SDA samotná data jsou připojena na MCU[v] takto: SCL > GP5 a SDA > GP4. Pi Pico nabízí možnost displej zapojit i na jiné piny (viz *Obrázek 1.7: Rozložení a funkce jednotlivých pinů* Raspberry Pi Pico), ovšem tyto jsou defaultní a v mém rozložení DPS[i] nebylo nutné zapojení měnit.

Kvůli ušetření místa jsem mezi displej a DPS[i] přidal pinhead typu F, aby se vyvýšila jeho poloha, bylo pod něj možné umístit MCU[v] a dosahoval otvoru na krytu. Při mých testech měl displej nejlepší poměr kontrastu a nerušení, když měl modrou barvu.

2.5 Tlačítka

Kvůli umístění součástek na horní stranu DPS[i] a přítomnosti displeje jsem zvolil tlačítka typu TS1112 12x12x15 DIP. Jedná se o čtvercová tlačítka s vyvýšenou dotykovou plochou. Tlačítka mají 4 piny, které jsou po dvou vnitřně spojeny. V této aplikaci by postačovaly 2 vývody, ale přidáním dvou dalších vývodů je zajištěno udržení stability při stisku tlačítka

2.6 Konektor JST

Pro připojení signálu z XLR konektoru jsem zvolil konektor JST-XH 3-pin. Jedná se o velmi oblíbené konektory využívané na DPS[i], a to především kvůli spolehlivosti jednoduché instalaci, nízké ceně a nízkému profilu, který je důležitým faktorem, zejména pokud je omezený prostor. Nejčastěji se vyskytují v bílé, šedé či světle hnědé barvě.

2.7 Ovládané světlo

Jako ovládané světlo jsem použil NeoPixel Ring 24 RGB[iv] WS2812B. Světlo je tvořeno 24 diodami WS2812B zapojenými podle výše uvedeného schématu (viz *Obrázek 1.6: Typické zapojení WS2812B*), dohromady tvořícími kruh o vnějším průměru 66 mm. Díky vyvedenému DOUT je možnost připojit další WS2812B diody. Je vyžadována minimální obnovovací frekvence 8 MHz. Při plném rozsvícení jedna LED spotřebovává běžně 12 mA a maximálně 18 mA, tedy celý kruh spotřebovává maximálně 432 mA.

2.8 Zdroj

Pro napájení výše zmíněných součástek je potřeba 5V zdroj, který bude schopný dodávat až 0,7 A. Tuto hodnotu jsem vypočítal sečtením všech maximálních proudů součástek a přibližně 100mA rezervy.

432 + 20,7 + 0,9 + 150 = 603,6 mA ≈ **0,7 A**

U MCU[v] se sice uvádí spotřeba přibližně 20 mA, ovšem když jsem si tento údaj ověřoval, zjistil jsem, že při zapnutí spotřeba krátkodobě dosahuje až 150 mA.

Pro účely tohoto projektu bude tedy postačovat zdroj Liteon PA-1100-17IU, s hodnotami 5,2 V a maximálně 2 A.

3 Software

3.1 Zvolený způsob programování

Rozhodoval jsem se mezi programováním v MicroPythonu (vývojové prostředí Thonny), nebo v C. Na základě více podnětů jsem se rozhodl program vytvořit v C ve vývojovém prostředí Arduino IDE. Hlavním důvodem byla větší kontrola nad časováním MCU[v] (PIO), což je u časově citlivého příjmu dat důležité. Také jsem již zvyklý jak na C, tak na vývojové prostředí Arduino IDE. Při programování je vhodné psát komentáře v anglickém jazyce, protože samotná programovací řeč má anglickou logiku. Zároveň není potřeba psát bez diakritických znamének a angličtině rozumí většina programátorů. Proto jsem zvolil psaní komentářů v angličtině.

3.2 Použité knihovny

3.2.1 Adafruit_SSD1306 (4)

https://github.com/adafruit/Adafruit_SSD1306
Verze 2.5.9

Knihovna pro připojení a komunikaci s displeji na základě ovladačů SSD1306. Umožňuje komunikaci I2C, nebo SPI pomocí 2 až 5 pinů. V dokumentaci zatím není mezi podporovanými čipy uveden RP2040, který je součástí Pi Pico, ovšem nenarazil jsem na žádný problém při připojení.

3.2.2 Adafruit_GFX (5)

https://github.com/adafruit/Adafruit-GFX-Library
Verze 1.11.9

Tato knihovna musí být společně s knihovnou Adafruit_SSD1306, aby byla zajištěna funkčnost zobrazení na displeji. Obsahuje základní grafické prvky pro více různých displejů (jedním z nich i SSD1306), například body, kruhy, čáry, fonty a další.

3.2.3 Wire (6)

https://www.arduino.cc/reference/en/language/functions/communication/wire/

Knihovna je součástí Arduino IDE a jejím účelem je zajištění komunikace se zařízeními, která využívají komunikační protokol I2C.

3.2.4 Adafruit_NeoPixel (7)

https://github.com/adafruit/Adafruit_NeoPixel
Verze 1.12.0

Tato knihovna je určena pro ovládání jednovodičových LED pixelů a LED pásků, jako je například v tomto projektu využitý WS2812B.

V dokumentaci (7) zatím není mezi podporovanými čipy uveden RP2040, který je součástí Pi Pico. Jediný problém, který jsem zaznamenal, bylo nefunkční nastavení maximální intenzity světla, tedy *setBrightness()*, které lze nastavit pouze na hodnotu 0, nebo 255. To v tomto projektu, dokud bude fungovat rozsvícení (255), není zásadní problém, protože se hodnota nastaví pouze jednou, a to při zapnutí (setup).

3.2.5 Pico-DMX (8)

https://github.com/jostlowe/Pico-DMX

Konkrétně část DmxInput je v tomto projektu využita pro příjem DMX signálu. Knihovna umožňuje jak čtení, tak vysílání dat a je přímo uzpůsobena pro Pi Pico.

V tomto projektu je potřeba pouze čtení dat, a to je umožněno hned dvěma způsoby: čtení celého universu (metoda *.read()*), nebo pouze několik specifikovaných kanálů (metoda *.read_async()*), což výrazně urychluje běh celého programu, ale znemožňuje nastavení vlastní adresy uživatelem.

Metoda *.read()* přečte celý DMX paket a posoudí, zda nedošlo k chybnému přenosu dat. Pokud je celý paket bez chyb, uloží se, v opačném případě se zahodí a zůstává uložený původní.

Metoda *.read_async()* oproti předchozí čte a ukládá pouze konkrétní adresy, ale nekontroluje již jejich správnost. Tedy ukládá i případné chybně přijaté pakety. Proto je zde vytvořena metoda *.dmxInput.latest_packet_timestamp()*, která určuje kolik času uběhlo od příjmu posledního paketu.

3.3 Uživatelské nastavení

3.3.1 Displej

Nejdříve bylo potřeba naimportovat knihovny Adafruit_SSD1306, Adafruit_GFX a Wire. Poté jsem definoval konstanty pro počet pixelů displeje na šířku (128) a výšku (64), které jsou potřeba pro deklarování displeje, kde se navíc zadává propojení ke knihovně Wire, ta nastavuje komunikaci s displejem (I2C).

V setupu ověřuji, zda je displej správně připojen a nakonfigurován. Pokud není, tak se (jestliže je MCU[v] připojeno k nějaké sériové lince) zobrazí chybová zpráva například na terminále. Jelikož nevyužívám originální displej, ale čínskou kopii, zadávám I2C adresu 0x3C a ne 0x3D, jak se nastavuje u originálních.

Pro zobrazení informací na displeji je dobré pro jistotu vždy vymazat jeho dosavadní obsah. Poté nastavit požadovanou velikost, barvu, pozici kurzoru a případně font textu (to umožňuje knihovna Adafruit_GFX). Nyní je možnost například vypsat konkrétní text, proměnnou, případně vytvořit tvar, či zobrazit obrázek. Veškerá tato nastavení se ovšem pouze ukládají do mezipaměti a na displeji se zobrazí až po vyvolání metody *display()*, přiřazené ke konkrétní deklaraci displeje.

Z estetických důvodů jsem zarovnal některé části menu na střed, což bylo obtížné u proměnných, které mění svou velikost. V dokumentaci (4) jsem našel funkci *getTextBounds()*, pomocí které zjistím šířku textu (bohužel funguje pouze pro datový typ string). Ve funkci *setCursor()* bylo nyní možné vypočítat nastavení polohy kurzoru tak, aby výsledný text byl ve středu obrazovky. Od poloviny maximální šířky obrazovky se odečte polovina vypočtené délky textu. Polovina maximální šířky obrazovky se zadává manuálně, pro možnost vycentrování textu i na části displeje.

3.3.2 Tlačítka

Každé tlačítko má v setupu přiřazené odpovídající číslo GPIO pinu a je nastavené na vnitřní pullup. V loopu se s nimi nejdříve pracuje ve funkci *displaymenu()*, kde se z každého tlačítka přečte hodnota vstupu a ta se zapíše do odpovídající proměnné.

Poté se testuje, zda je tlačítko menu stisknuté (tedy LOW) a zároveň je pomocná proměnná *btn_menu_clicked* rovna nule, pokud obě podmínky platí, přičte se k proměnné *item_selected* 1, což v důsledku uživatele posune o jednu nabídku v menu dál. Pokud by mělo dojít k překročení počtu existujících nabídek v menu, automaticky se přejde na první položku menu. Pomocná proměnná *btn_menu_selected* se zde nachází, aby byla požadovaná akce po stisknutí tlačítka vykonána pouze jednou. Po uvolnění tlačítka se tato proměnná vynuluje a je možné tlačítko znovu stisknout a provést požadovanou akci znovu.

Tento systém je použit pro všechna tlačítka s úpravou u tlačítek up a down, u kterých by tato funkce byla u nastavování větších čísel spíše na obtíž (200x zmáčknout tlačítko není uživatelsky přívětivé). Přidal jsem tedy tuto podmínku: pokud bude tlačítko přidrženo cca 2 sekundy, bude při každém cyklu MCUv zvýšena, či snížena hodnota nastavované veličiny automaticky o 1, dokud bude tlačítko stisknuté. Funkci *delay()* nevyužívám, protože přeruší průběh cyklu do doby, než doběhne.

3.3.3 Manuální nastavení

Aby měl uživatel možnost nastavení světla i bez signálu DMX, je jednou z položek menu manuální nastavení. Umožňuje uživateli manuálně nastavit hodnoty jednotlivých barev a strobe (blikání).

Tato položka menu se skládá z dalších dvou položek, pomocí první se vybírá nastavovaná veličina a druhou hodnota této veličiny. Mezi těmito položkami se přepíná tlačítkem enter. Aby uživatel viděl, se kterou veličinou právě pracuje, zobrazuje se před aktivní veličinou šipka. Samotnou veličinu poté nastavuje tlačítky up, nebo down. U nastavení konkrétní veličiny je možnost přidržení tlačítka, a tím zrychlení výběru mezi hodnotami. (popsáno v části *3.3.2 Tlačítka*).

3.3.4 Smyčka

MCUv může pracovat podobně jako PLC, tedy vytvářet cyklus, který se stále opakuje. V tomto cyklu mohou být určité podmínky, příkazy, funkce, na základě kterých se rozhoduje o výstupních veličinách. Na tomto způsobu pracuje například Arduino.

V mém programu se právě taková smyčka vyskytuje. Nejprve spustí funkci *displaymenu()* (popsáno v části *3.3.1 Displej*) a poté rozhoduje, jestli se uživatel právě nachází v nastavení DMX, či v manuálním nastavení.

V nastavení DMX, což je adresa, mód a DMX fail, se rozhoduje o výstupu na LED podle DMX vstupu. Nejdříve je ošetřena možnost, že nejsou přijímána data, což většinou znamená odpojení od signálu. V tom případě se rozhoduje, zda uživatel nastavil DMX fail na „Hold“, či „Blackout“. Možnost „Hold“ při nepřijatých datech ponechá poslední správně přijatou hodnotu. „Blackout“ na výstup vyšle nuly (zhasne). Pokud jsou přijímána správná data, program rozhoduje o současně nastaveném módu a adrese a na základě nastavení pro každý mód vypočítá pro ten právě nastavený výstupní hodnoty.

`3ch`

Tedy tříkanálový mód přečte hodnoty 3 kanálů od prvního uživatelem nastaveného. Tyto hodnoty se zapíšou do funkce *setfled()*, která nastaví korespondující barvy na všech pixelech kruhu.

`5ch`

Pětikanálový mód nastavuje, jako v tříkanálovém, jednotnou barvu na celém kruhu, ovšem tento mód disponuje navíc dvěma kanály (master dimmer a strobe). Master dimmer (první kanál) limituje celkovou svítivost kruhu. To je velmi užitečné při postupném stmívání barvy, složené z více než jedné. Strobe (jinak řečeno blikání) určuje rychlost blikání, respektive čas mezi probliknutím, nastavené barvy. Nula znamená vypnuté strobe a se zvyšující se hodnotou se také zvyšuje rychlost blikání. Toto je docíleno výpočtem podrobněji popsaným níže (viz Rovnice 1: výpočet konstanty blikání).

72ch

Sedmdesáti dvou kanálový mód obsazuje velké množství kanálů (512 v jednom universu). Představuje možnost plné kontroly nad všemi pixely individuálně. Pro nastavení je využit for loop. Počítá se, že kruh má 24 pixelů (LED_COUNT) a každý pixel má tři barvy. Proměnná „k“ představuje pořadové číslo právě nastavovaného pixelu a „b“ určuje počáteční kanál právě nastavovaného pixelu.

V případě manuálního nastavení se nastavuje hodnota výstupu na základě dvoudimenzionální proměnné *man_valx[][]*. Nejdříve se zkontroluje, jestli je hodnota blikání větší než nula. Pokud ano, tak se porovná současná hodnota *strobe_time* (jež určuje „čas“, který již uběhl od doby, kdy se naposledy na výstup vyslaly hodnoty barev) s konstantou vypočítanou podle *Rovnice 1: výpočet konstanty blikání*, kde $\boldsymbol{x}$ je výsledná hodnota a $\boldsymbol{v}$ je hodnota strobe zadaná uživatelem. Pokud je konstanta větší než *strobe_time*, na výstup se vyšlou nulové hodnoty (zhasnuto), v opačném případě se podobu minimálně jednoho cyklu na LED zobrazí požadovaná barevná kombinace.

Rovnice 1: výpočet konstanty blikání

$$x = \frac{|v - 270|}{10}$$

3.4 Čtení dat DMX pomocí MAX485

DMX přenese 513 datových rámců, z toho první je tzv. start code a zbylých 512 obsahuje hodnoty každého z kanálů. Start code se standartně značí 0.

Součástí RP2040 jsou 2 základní komunikační protokoly pro periferie, a to I2C a SPI. Tyto protokoly ovšem nemohou číst ani vysílat DMX, proto je potřeba vytvořit program právě pro práci s DMX. Na většině MCU[v] se pro tyto situace využívá tzv. proces bit banging, který umožňuje velmi rychle vypínat a zapínat určité piny v přesné časy. Tento proces vyžaduje velmi přesné interrupty v procesoru, jejichž využití ho velmi rychle zahltí.

RP2040 disponuje funkcí PIO, která tyto náročné procesy může od procesoru převzít a uvolnit tak výpočetní výkon pro jiné procesy. Tyto PIO bloky se zde nacházejí 2, z nichž každý obsahuje 4 stavové automaty. To umožňuje pracovat až s 8 DMX universy v jednu chvíli. Programují se v řeči podobné asembleru a podporují 9 instrukcí.

3.5 Zápis dat na LED

K ovládání LED kruhu využívám knihovnu Adafruit_NeoPixel. Při jejím deklarování je potřeba zadat počet LED, GPIO pin na MCU[v] a typ LED (RGB[iv], RGBW, …). Poté je potřeba v setupu zadat příkaz *begin()*, ihned poté se doporučuje spustit příkaz *show()*, čímž se

vynulují hodnoty na LED, tedy nebudou svítit, a pak nastavení maximální intenzity světla (0-255). Tím je vše potřebné nastaveno.

Pro rozsvícení jednoho pixelu na určitou barvu je nutné zadat příkaz *setPixelColor()*, ve kterém jsou vyžadovány dva parametry, a to pořadové číslo pixelu a hodnoty jednotlivých barev nastavené příkazem *Color()*, jenž v mém případě obsahuje 3 číselné hodnoty od 0 do 255, udávající intenzitu jednotlivých barevných LED diod. S rozsvícením celého kruhu je nutné vytvořit loop (například for), který bude toto nastavení opakovat pro každé pořadové číslo pixelu, díky tomu, že se při každém cyklu zvýší o jeden. Tyto příkazy se, podobně jako u obrazovky, pouze ukládají do mezipaměti a fyzicky se zobrazí na LED až po spuštění příkazu *show()*.

4 Spojení

4.1 Kryt

Jako zadní část krytu jsem využil rozebranou část nefunkčního reflektoru, na základě které jsem vytvářel jednotlivé součásti, jako například rozmístění tlačítek, či umístění displeje na DPS[i]. Protože je kryt kovový, je propojen se zemnícím kabelem.

Přední část je tvořena hrdlovou zátkou KG pro odpadní potrubí průměr 200 mm. Na tomto dílu jsem vyměřil a vyvrtal potřebné otvory pro umístění vrutů a usazení WS2812B ringu. Následně jsem aplikoval nástřik matně černou barvu.

4.2 Konektory

Na zadní straně krytu jsou umístěny čtyři konektory, a to dva napájecí a dva signálové.

4.2.1 Napájecí

Jedná se o konektory typu PowerCon. Modrý je určen pro vstup a bílí pro výstup (průchozí). Mezi sebou jsou propojeny sériově a napájení zařízení je paralelně ke vstupnímu (modrému) konektoru. Navíc fáze do zařízení vede přes 10A tavnou pojistku.

4.2.2 Signálové

Signálové konektory jsou blíže popsané v části 1.2.1 DMX512. XLR konektory jsou mezi sebou propojeny sériově a vývod signálu do zařízení je proveden paralelně k tomuto propojení (viz *Obrázek 1.4: Zapojení 1-n MAX485*) zakončen konektorem JST.

4.3 DPS

Deku plošného spoje jsem tvořil pomocí programu Autodesk Fusion 360 (dříve známý jako Eagle). Vytvářel jsem si svou vlastní knihovnu, a to pro IC[i] MAX485. Výstup jsem využil pro potřeby dokumentace a jako předlohu pro výrobu DPS.

5 Závěr

V této práci byl prezentován návrh a implementace systému pro řízení osvětlení pomocí protokolu DMX512. Celý systém byl navržen s ohledem na robustnost, flexibilitu a uživatelskou přívětivost. Práce je zaměřena na implementaci softwaru pro mikrokontrolér (Raspberry Pi Pico), správné čtení a zpracování dat z DMX a efektivní ovládání LED osvětlení.

V první a druhé části práce byl popsán teoretický základ protokolu DMX512 a jeho využití pro řízení osvětlení. Dále byly představeny technologie a součásti použité pro realizaci systému, včetně mikrokontroléru Raspberry Pi Pico s čipem RP2040, komunikačních rozhraní a knihoven použitých pro práci s displejem a LED osvětlením.

Třetí část práce detailně popisuje implementaci softwaru pro mikrokontrolér, včetně nastavení displeje, ovládání tlačítek, manuálního nastavení osvětlení a hlavní programové smyčky pro čtení dat z DMX a ovládání LED osvětlení. Byl kladen důraz na efektivitu čtení DMX dat pomocí PIO funkcionality RP2040 a na přesné ovládání LED osvětlení pomocí knihovny Adafruit_NeoPixel.

Navržený systém splňuje požadavky pro řízení osvětlení pomocí protokolu DMX512. Jeho implementace byla úspěšně provedena a výsledný systém je schopen spolehlivě a efektivně číst přijatá data z XLR konektoru na světle, na základě kterých je řízeno osvětlení. Uživatel má možnost nastavit adresu a mód pro čtení DMX. Také má možnost nastavit, co se stane při přerušení příjmu dat a manuálně nastavit jednotlivé barvy a rychlost blikání přímo na světle (bez DMX).

Další možností rozšíření systému by mohlo být přidání dalších funkcionalit, jako například podpora RDM, možnost komunikace mezi světly. Program byl tvořen tak, aby bylo jednoduché přidat tyto funkcionality a případně využití jiného světelného zdroje.

Tato práce přináší užitečný příspěvek v oblasti řízení osvětlení a demonstruje úspěšnou integraci moderních a ustálených technologií pro efektivní a spolehlivé řízení osvětlení. Je zaměřena na oblast, která není dostatečně zdokumentována pro běžné uživatele, a to příjem DMX signálu a jeho zpracování.

[i] DPS je zkratka – Deska Plošného Spoje

[ii] Parita je jeden bit přidávaný na konec zpráv. Indikuje, jestli je počet log. „1“ ve zprávě lichý, nebo sudý.

[iii] IC je anglická zkratka – Integrted Cyrcuit, česky: Integrovaný Obvod

[iv] RGB je anglická zkratka – Red Green Blue, česky: Červená Zelená Modrá

[v] MCU je anglická zkratka – MicroController Unit, česky: jednotka mikrokontroléru