ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta elektrotechnická
Katedra elektroniky a informačních technologií

BAKALÁŘSKÁ PRÁCE

Řídicí software pro DMX controller s využitím Raspberry Pi

Autor práce: **Petr Noháč**
Vedoucí práce: **Ing. Jaroslav Fiřt, Ph.D.**

2023

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Petr NOHÁČ**
Osobní číslo:	**E19B0101P**
Studijní program:	**B2612 Elektrotechnika a informatika**
Téma práce:	**Řídicí SW pro DMX controller s využitím Raspberry Pi**
Zadávající katedra:	**Katedra elektroniky a informačních technologií**

Zásady pro vypracování

1. Proveďte rešerži již realizovaných dostupných návrhů.
2. Využijte a případně upravte nejvhodnější řešení.
3. Navrhněte a realizujte SW řízení scén osvětlení.
4. K ovládání aplikace použijte primárně řízení TCP/IP sockety.

Rozsah bakalářské práce: **30 – 40**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **elektronická**

Seznam doporučené literatury:

1. https://www.raspberrypi.org/
2. http://www.dmx-512.com/

Vedoucí bakalářské práce: **Ing. Jaroslav Fiřt, Ph.D.**
Katedra elektroniky a informačních technologií

Datum zadání bakalářské práce: **7. října 2022**
Termín odevzdání bakalářské práce: **26. května 2023**

L.S.

Prof. Ing. Zdeněk Peroutka, Ph.D.
děkan

Doc. Ing. Jiří Hammerbauer, Ph.D.
vedoucí katedry

V Plzni dne 7. října 2022

Abstrakt

Práce je zaměřena na prostudování různých způsobů řízení scénického osvětlení, jednočipového počítače Raspberry Pi, komunikačního protokolu DMX512, návrh a realizace softwarového ovládaní. Snadné použití a komunikaci zajištuje protokol DMX512, který se široce uplatňuje v digitální realizaci pro jevištní techniku. Raspberry Pi slouží pro příjem příkazů ve formě paketů přes TCP/IP socket, zpracovává přijaté příkazy a následně posílá data pro osvětlení po UART sériové lince do poskytnuté vyrobené desky. Při návrhu řídicího softwaru byl využit software OLA, který realizuje přenos dat DMX512 protokolu. Pro příjem příkazu, jejich zpracování a celkovou automatizaci byly vytvořeny programy pro Raspberry Pi v programovacím jazyce Python.

Klíčová slova

Scénické osvětlení, Raspberry Pi, DMX512, TCP/IP, OLA, UART, Python

Abstract

This bachelor thesis is focused on studying different ways of controlling stage lighting, single chip computer Raspberry Pi, communication protocol DMX512, design and implementation of controlling software. Communication with the stage lights is provided by the DMX512 protocol, which is widely used in digital implementation for stage technology. Raspberry Pi is used to receive commands, in the form of packets over to TCP/IP socket, process the received commands and then to send the data for lighting over the UART serial link using the provided fabricated board. In the design of the controlling software, the OLA software was used to implement the DMX512 protocol data transfer. Programs to receive the commands, process them and for overall automatization were created on Raspberry Pi in Python programming language.

Key Words

Stage lighting, Raspberry Pi, DMX512, TCP/IP, OLA, UART, Python

Poděkování

Chtěl bych poděkovat vedoucímu bakalářské práce Ing. Jaroslavu Fiřtovi, Ph.D. za poskytnuté konzultace a především možnost testování realizace návrhu v laboratorních prostorech katedry KEI.

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci, zpracovanou na závěr studia na Fakulty elektrotechnické Západočeské univerzity v Plzni.

Prohlašuji, že jsem svou závěrečnou práci vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledku vyplývajících z ustanovení § 270 trestného zákona č. 40/2009 Sb.

Také prohlasuji, že veškerý software, použitý při řešení teto bakalářské práce, je legální.

V Plzni dne 25. května 2023

Petr Noháč

.....

Podpis

Obsah

Seznam symbolů a zkratek

TCP		Transmission Control Protocol
UDP		User Datagram Protocol
IP		Internet Protocol
UART		Universal asynchronous receiver/transmitter
GPIO		General purpose input/output
DMX		Digital Multiplex
RDM		Remote Device Management
ARM		Acorn RISC Machine
USITT		United States Instutute for Theatre technology
USB		Universal Seriál Bus
RAM		Random access memory
OLA		Open Lighting Architecture
QLC+		Q Ligh Controller Plus
XLR		External Line Return
U	V	elektrické napětí
I	A	elektrický proud
R	Ω	elektrický odpor

Úvod

V oblasti jevištního umění, ať už filmových nebo divadelních představení, hudebních vystoupení, kulturních akcí, či televizního natáčení, se v dnešní době využívá osvětlovací technika téměř vždy. Scénické osvětlení umožňuje divákovi dokreslit podtext daného vystoupení a dodává vnímavosti na atmosféře. To ovšem vyžaduje náročnou techniku a zároveň i osobu pro její obsluhu. Díky digitalizaci v této oblasti došlo ke značnému zjednodušení samotného řízení osvětlovací techniky. Pro toto řízení se standardně používají osvětlovací pulty s digitálním výstupem DMX 512, díky němuž probíhá komunikace mezi pultem a výkonovými jednotkami. Tyto pulty jsou ovšem často velmi drahé a pro menší pořadatele finančně nedosažitelné.

Tato práce má za úkol provést návrh a realizaci řídicího softwaru, který umožní ovládání osvětlení v jednosálovém kině. Jedná se o jednoduché využití, při kterém se bude plynule přecházet mezi pěti scénami a bude tak moci být nahrazen osvětlovací pult. Tato práce navazuje na bakalářskou práci „DMX controller s využitím Raspberry Pi“[1], jejíž cíl byl návrh desky DMX controlleru. V mém návrhu bude tato deska použita.

Raspberry Pi je vhodné pro tuto realizaci z důvodu možnosti příjmu TCP/IP paketů, možnosti připojení vytvořené desky na GPIO rozhraní a snadného vytvoření programu pro řízení přímo na samotném zařízení.

Pro pochopení této práce a problematiky spojené s digitalizací osvětlovací techniky, je zapotřebí vysvětlit několik technických pojmů a popsat nejčastěji používané metody pro obdobné realizace.

1 Protokol DMX512

DMX512 je povelový protokol pro digitální přenosy řídicích informací. Byl vyvinut v roce 1986 institutem USITT pro nahrazení analogového řízení světelné a jevištní techniky, případně jejich efektů. Do té doby analogové řízení mělo jako řídicí veličinu konkrétní hodnotu napětí na řídicím kabelu, což vedlo k mnoha potížím. Pro každý řízený vstup bylo zapotřebí vodiče a nebyla vždy přesně dána řídící veličina, protože z důvodu výrobních tolerancí mohla různá zařízení chápat nebo vydávat mírně odlišné povely. Zároveň analogový přenos byl náchylnější na rušení, které bylo způsobováno několika kilowattovými výkonovými jednotkami.

Základem protokolu DMX 512 je jeho elektrická specifikace, která vychází z průmyslového standardu EIA485. Díky použití tohoto standardu v průmyslu, jsou i technické prostředky pro jeho implementaci levné a přizpůsobené pro náročné podmínky. [2] [3]

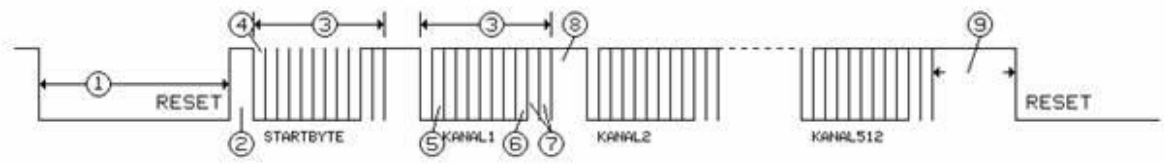
1.1 Technické specifikace EIA485

- diferenciální napěťový přenos pracující od napětí +5 V,
- rozsah napětí přípustného na sběrnici od -7 V do +12 V,
- možnost připojení až 32 přijímačů/vysílačů na jedné sběrnici,
- počet segmentů není limitován,
- impedanční přizpůsobení linky odporem/terminátorem 120 Ω,
- minimální zatěžovací impedance vysílače je 60 Ω,
- maximální zkratový proud vysílače je 150 mA proti zemi, 250 mA proti 12 V,
- maximální délka vedení sběrnice je 1200 metrů, při maximální přenosové rychlosti 400 kBit/s,
- budič musí dodat na sběrnici rozdílové vstupní napětí od 1,5 do 5 V,
- budič musí mít ochranu pro případ, kdy by se více budičů pokoušelo vysílat na sběrnici,
- přijímač by měl mít minimální vstupní impedanci 12 kΩ,
- logická „1“ je definována napěťovou úrovní A – B < -200 mV,
- logická „0“ je definována napěťovou úrovní A – B > +200 mV. [2]

1.2 Popis datového formátu DMX512

Protokol DMX512 má stanovenou rychlost přenosu na 250 kBit/s. Data jsou po sběrnici posílána sériově, paketem o maximální velikosti 512 datových bajtů (rámců). Po samotné sběrnici se posílají pouze data bez adresy. Zařízení připojená na sběrnici mají na počátku přidělené adresy a při přijímaní si přečtou jak adresu, tak i požadovaný počet bitů. Počáteční adresa tedy nabývá hodnotu 0 až 511. Pokud by zařízení byla shodná a byla jim přidělena identická adresa, budou na přijímaná data reagovat identicky. Jedná se o jeden způsob jak připojit na sběrnici více zařízení, než je podle technické specifikace možné. Dalším způsobem by bylo využití splitterů a repeaterů.

Přenos je realizován asynchronně. Začátek je ale synchronizován nulovou úrovní „Break“ a následnou synchronizační mezerou MAB (Mark After Break) s logickou úrovní High. Dále pak následuje první rámec (start code) a zbývajících 512 datových rámců. Datové rámce obsahují jeden start bit, osm datových bitů a dva stop bity. Mezi jednotlivými rámci se mohou vyskytovat mezery MTBF (Mark Time Between Frames) a MTBP (Mark Time Between Packet). Datový přenos je znázorněn na Obr. 1.1 a konkrétní časové údaje jsou uvedeny v tabulce č. 1. [2]



Obr. 1.1 Časový diagram přenosu jednoho paketu protokolu DMX 512

Díky známé přenosové rychlosti 250 kBit/s vyplývají časové údaje: doba trvání jednoho bitu je 4 us a doba trvání datového rámce je 44 us. Pro určení délky trvání celého přenosu s celkovým počtem datových bitů je dán následující vztah [2]:

$$Break + MAB + (1 + 512) \cdot rámec = 88 + 8 + 513 \cdot 44 = 22{,}668\ ms \qquad (1.1)$$

Tabulka 1 Přehled časování protokolu DMX512 [2]

Číslo	Popis	Min.	Typ.	Max.	Jednotka
1	Break (Reset)	88 us	176 us	1 s	V tabulce
2	MAB (synch. Mezera)	8 us	-	1 s	V tabulce
3	Rámec	43,12	44	44,48	us
4	Start bit	3,92	4	4,08	us
5	LSB (první datový bit)	3,92	4	4,08	us
6	MSB (poslední datový bit)	3,92	4	4,08	us
7	Stop bit	3,92	4	4,08	us
8	MTBF	0	0	1	s
9	MTBP	0	0	1	s

1.3 Možnosti Přenosu DMX přes IP síť

Při realizaci přenosu DMX dat pomocí internetové sítě se standardizovalo několik metod, jak přenos uskutečnit. Přestože v návrhu pro zadaný konkrétní případ jsem tyto metody nevyužil, je nezbytné metody stručně popsat a vysvětlit jejich funkce, aby bylo možné odůvodnit jejich nevyužití.

1.3.1 Art-Net

Art-Net je ethernetový protokol pro distribuci dat, který umožňuje přenášet data DMX512 (dále také DMX) a RDM osvětlení prostřednictvím ethernetové sítě. Používá jednoduchou strukturu paketů založenou na protokolu UDP. Art-Net je vlastněn společností Artistic Licence Holdings Ltd a má k němu autorská práva. Společnost však zveřejnila specifikaci Art-Netu a poskytla ji komukoli k bezplatnému použití.

Nejnovější verzí je Art-Net 4, která byla zveřejněna v roce 2016. Dokáže přenést až 32768 paketů a lze připojit až 1000 DMX portů za využití pouze jedné IP adresy. Art-Net umožnuje zároveň RDM (Remote Device Management). O samotný DMX512 přenos se stará datový paket ArtDmx. Ten je použit vždy, když dochází k DMX vysílání, které nastává při komunikaci od pultu k zařízení, ale i od zařízení k pultu zpět. K obnově vysílání dochází při změnách nebo každé 4 sekundy, pokud se výstup nemění. [4]

1.3.2 sACN

sACN (streaming ACN), také známý jako E1.31, je odlehčenou verzí ACN protokolu, určený pro přenos DMX512 dat pomocí ethernetové sítě. Podobně jako Art-Net využívá struktury packetů založené na UDP protokolu. sACN byl adaptován asociací ESTA (Entertainment Services Technology Association) a byla standardizována roku 2016. sACN je schopen přenést až 63999 paketů, ale neumožnuje RDM. [5][6]

Komunikace obou těchto metod probíhá velmi podobně a formáty paketů znázorním dále pouze na protokolu Art-Net a paketu ArtDmx.

Pakety, které se nazývají universe, jsou specifikovány podobně jako struktury programovacího jazyka C, ve kterých jsou všechny datové formáty považovány za celá čísla bez znaménka typu INT8, INT16 nebo INT32. Podle počtu bitů, neexistují žádné skryté výplňové bajty. Výjimkou jsou samotné bajty na konci paketu, které mohou být zaokrouhleny na násobky 2 nebo 4 bajtů. Tyto nadbytečné byty na konci jsou pak po přijetí platného paketu ignorovány. Mnoho bitových polí obsahuje nevyužité pozice, které mohou být použity v budoucích verzích protokolu. Měly by být přenášeny jako nula a přijímače by je neměly testovat. Všechny definice paketů jsou navrženy tak, aby bylo možné v budoucnu prodloužit jejich délku a zachovat tak kompatibilitu. Z tohoto důvodu je pouze minimální délka paketu kontrolována. [4]

Tabulka 2 popis ArtDmx Paketu [7]

	Velikost	Popis
ID	Int8	Identifikace jako Art-Net
OpCode	Int16	Druh zaslaného paketu
PortVer	Int16	Číslo verze Art-Net protokolu
Sequence	Int8	Pořadí paketu
Physical	Int8	Číslo fyzického vstupního portu
Universe	Int16	Adresa pro daný paket
Length	Int16	Počet rámců v paketu
Data	Int8	Data jednotlivých rámců

Tato skutečnost je pro náš konkrétní účel nevhodná. Požadavek na řídící příkaz, který bude posílán pomocí TCP protokolu, je takový, aby byl co nejjednodušší a mohl být poslán jako jednorázový impuls pro přechod z jedné scény na druhou. Pokud by byly využity výše zmiňované metody, znamenalo by to, že například plynulé přechody by musely být realizovány posíláním několika paketů, představující určité mezistavy přechodu mezi scénami.

Vytvořil jsem tedy alternativu, která tento problém řeší pro můj případ a je dále popsána v části 6.6 Příjem TCP/IP paketů.

1.4 Osvětlovací Pult

Jak už jsem zmínil, cílem této práce je navrhnout software, aby bylo možné nahradit osvětlovací pult. Proto by bylo vhodné toto zařízení lehce popsat a představit jeho funkce.

Osvětlovací pult slouží pro generování dat DMX512 a jejich ovládání. Data jsou posílány k výkonovým jednotkám pomocí XLR kabelu, který je možný přímo připojit na 3-pinový XLR DMX konektor osvětlovacího pultu. Pro řízení posílaných dat, respektive ovládání jednotlivých kanálů, se používají fyzické prvky, které mohou být různého druhu v závislosti na konkrétním pultu. Nejčastěji se jedná o tahové potenciometry, které poskytují plynulý přechod, ale může být také k dispozici řada tlačítek nebo u drahých variant dokonce dotykový displej. [8]

Konkrétně popíšu jednu z nejlevnějších variant osvětlovacího pultu Stairville DDC-12. Pult je na stránkách eshopu soh.cz ke dni citace [9] dostupný za cenu 2158 Kč. Ovládací metodu nabízí pouze v podobě tahových potenciometrů a umožnuje individuálně ovládat až 12 kanálu DMX. Zároveň je součástí potenciometr s názvem Master, který ovládá všechny kanály najednou. Pult dále poskytuje LCD display, který slouží pro zobrazení aktuálního stavu v digitální podobě, a na boční straně zpřístupňuje XLR DMX konektor. [8]

Obdobný typ osvětlovacího pultu budu chtít nahradit pomocí Raspberry Pi.

Obr. 1.2 Osvětlovací pult DDC-12 LCD [9]

2 Raspberry Pi

Pro realizaci návrhu softwaru mi byl poskytnut počítač Raspberry Pi 1. generace, který bude následně využíván pro danou konkrétní aplikaci v jednosálovém kině. Zároveň jsem měl k dispozici mnou vlastněný Raspberry Pi 3, který sloužil jako alternativní testovací zařízení. Je tedy vhodné popsat Raspberry Pi a jeho různé verze.

Jedná se o jednodeskový počítač malých rozměrů, s výkonem slabšího stolního počítače. Použitý mikroprocesor je z rodiny ARM. Počítač Raspberry Pi byl vyvinut ve Velké Británii nadací „Raspberry Pi Foundation“ společně s firmou Broadcom v roce 2012. Cílem byla snaha podpořit výuku základů počítačové technologie na školách a lépe seznámit studenty s využitím počítačů a možnosti řízení různých zařízení. Nízká cena, modularita a otevřený design však učinily tento originální model populárním a Raspberry Pi se rozšířilo po celém světě ať už v technických oblastech jako je například robotika, ale také školách nebo interaktivních muzeích. Samotné názvy počítačů jsou inspirovány firmou Arcon Computers. Název „Raspberry Pi“ vznikl tradičním pojmenování počítačových firem po ovoci, odtud „Raspberry“ a odkazem na programovací jazyk Python, tedy „Pi“. [10]

2.1 Rozdělení modelů

Počítač Raspberry Pi se od svého uvedení na trh dočkal několika verzí. V této části je stručné srovnání některých významných verzí počítače Raspberry Pi s důrazem na jejich klíčové vlastnosti a vylepšení.

- Raspberry Pi Model A

Raspberry Pi Model A má nižší spotřebu energie, je vybaven jedním portem USB a 256 MB paměti RAM. Je vhodný pro aplikace s nízkou spotřebou energie a projekty s minimálními požadavky na zdroje.

- Raspberry Pi Model B

Raspberry Pi Model B má dva porty USB, port Ethernetový a 512 MB paměti RAM. Díky vylepšeným možnostem připojení a zpracování je vhodný pro multimediální centra, systémy domácí automatizace a vzdělávací projekty.

- Raspberry Pi Zero

Raspberry Pi Zero je díky svým malým rozměrům, jednojádrovému procesoru a napájení pomocí micro-USB určen pro kompaktní a lehké projekty se základními výpočetními potřebami. Často je použito pro přenosná zařízení, jako jsou například retro herní konzole.

- Raspberry Pi 2

Raspberry Pi 2 disponuje čtyřjádrovým procesorem ARM Cortex-A7 a 1 GB paměti RAM. Oproti dřívějším modelům nabízí výrazné zvýšení výkonu a zvládá náročnější úlohy, jako jsou webové servery, robotika a záznam dat.

- Raspberry Pi 3

Raspberry Pi 3 je vybaven čtyřjádrovým procesorem ARM Cortex-A53, moduly pro Wi-Fi a Bluetooth připojení a 1 GB paměti RAM. Opět poskytuje vyšší výpočetní výkon a bezdrátové funkce. Běžně se používá v domácích multimediálních serverech nebo monitorovací aplikace.

- Raspberry Pi 4

Raspberry Pi 4 je má čtyřjádrový procesor ARM Cortex-A72, až 8 GB RAM, porty USB 3.0 a podporuje dva displeje s rozlišením 4K. Nabízí výrazný výkonnostní skok a vylepšené multimediální schopnosti, takže je vhodný pro náhradu stolních počítačů, multimediální centra s vysokým rozlišením a pokročilé projekty internetových síťí. [10]

Obr. 2.1 Raspberry Pi 1B[11]

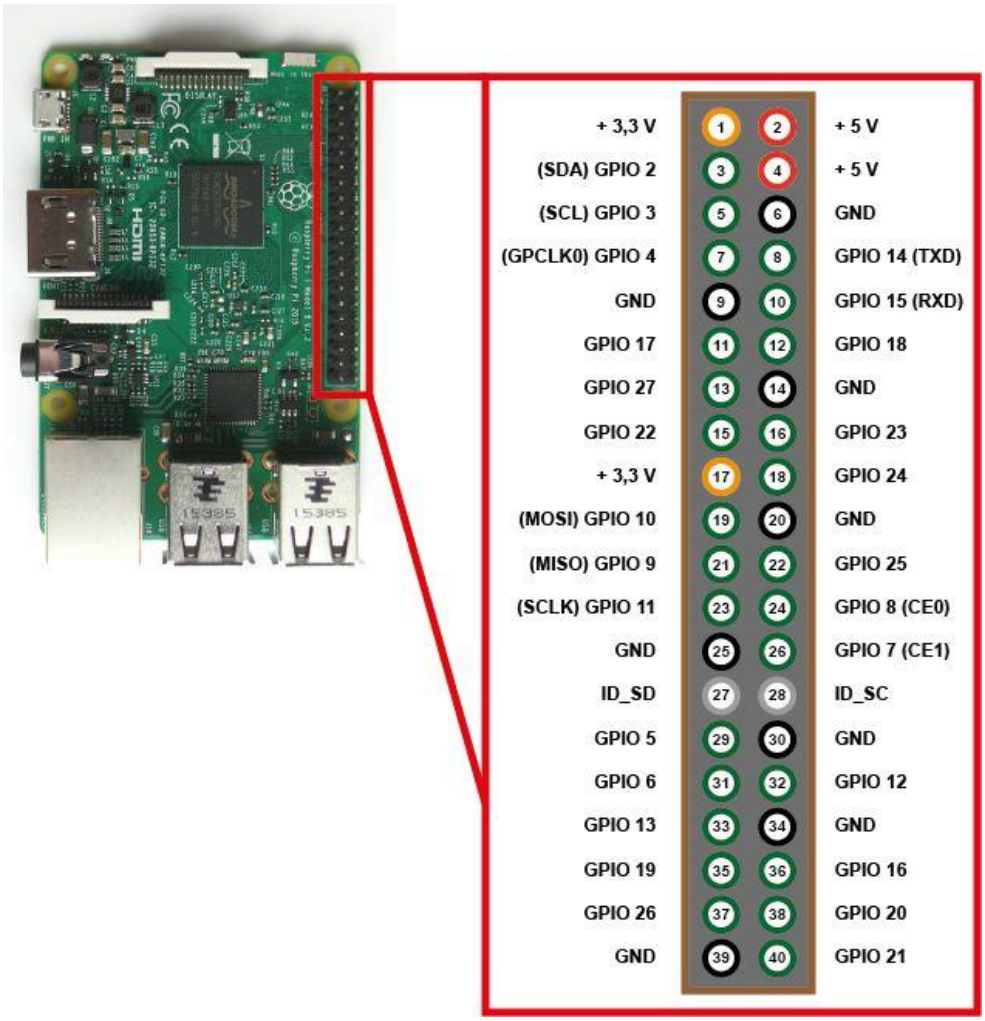
2.2 Software

Raspberry Pi má možnost instalace různého množství operačních systémů. Jako hlavní a officiální operační systém je Raspberry Pi OS, který je dostupný přímo na stránkách Raspberry Pi[10]. Je možné ale nainstalovat řadu dalších operačních systémů, které jsou určeny pro procesory ARM. Příkladem jsou RISC OS, Windows 10 IoT Core nebo systémy založené na Linux distribuci jako Debian a Ubuntu. Existuje ale i řada dalších operačních systému, které jsou ovšem určeny pro multimédia nebo hraní starších videoher. U Raspberry Pi 1. generace je pouze podporován operační systém Raspberry Pi OS a není možné nainstalovat jiné alternativy. Při prvotní realizaci Raspberry Pi se předpokládalo, že pro programy se bude využívat programovacího jazyka Python, nicméně v základním operačním systému Raspberry Pi OS je dnes předem nainstalována podpora také pro programovací jazyky Scratch, C a C++. [10][12]

2.3 GPIO Raspberry Pi

Jednou z velmi užitečných vlastností Raspberry Pi je jeho sada GPIO (General Purpose Input/Output) Pinů. Tyto piny jsou fyzickým prostředkem, jakým může komunikovat s okolním světem a umožnuje nespočet funkcí jako ovládání LED, motorů, ale i monitorování teploty, světla nebo zmáčknutí tlačítka. To vše v závislosti na připojené desce.

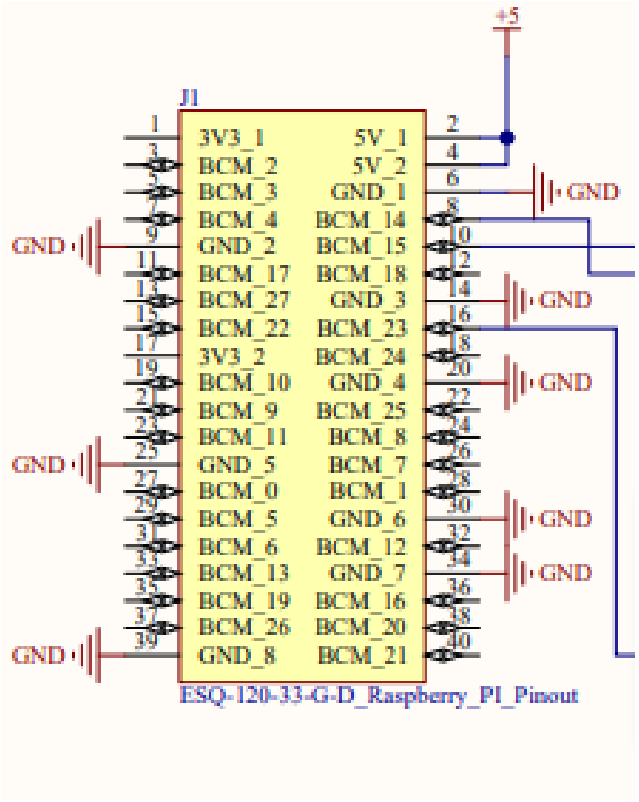
Raspberry Pi má k dispozici sadu 40 pinů, přičemž jejich funkce se odlišují. Jedná se o zdroje napětí 3,3 V nebo 5 V, uzemnění a samotné GPIO, které se dále dělí na piny se speciálním účelem a piny pro jednoduchý vstup/výstup. Speciální účely, které Raspberry Pi nabízí jsou SPI, I2C, UART a PCM. Přesné rozložení jednotlivých pinů je zobrazeno na obr. 2.1. [13][14]



Obr. 2.1 Rozložení pinů na rozhraní Raspberry Pi GPIO[15]

2.4 Přidělená vyrobená deska pro Raspberry Pi

Pro posílání DMX dat k příslušné osvětlovací technice byla katedrou KEI vyrobena a poskytnuta deska, navržená v bakalářské práci „DMX controller s využitím Raspberry Pi“ [1]. Deska se připojuje na GPIO rozhraní Raspberry Pi, a umožnila mi tak testování a dolaďování mého návrhu. Musel jsem zohlednit schéma desky, které mi poskytovalo informaci, kam a jaký typ signálu je zapotřebí přivést na piny GPIO z pohledu Raspberry Pi. Jedná se o přivedení logické „1“ na GPIO23, sloužící jako enable signál pro uvedení desky do vysílacího režimu, a samotné posílání DMX dat pomocí GPIO se speciálním účelem UART (GPIO14, GPIO15). Schéma a obrázky desky jsou v příloze [A].



Obr. 2.2 Raspberry Pi pinout vyrobené desky[A]

2.5 UART Raspberry Pi

Pro návrh posílání DMX512 dat jsem musel pracovat s GPIO piny, které mají speciální účel pro UART komunikaci. V následující části je tedy stručně popsána jeho funkce, a jakým způsobem je použit na Raspberry Pi.

UART neboli „Universal Asynchronous Receiver/Transmitter“ je sériový komunikační protokol. Data se přenášejí sériově bit po bitu, což se široce používá pro přenos orientovaný na Bajty. UART používá pro komunikaci dat jasně definovanou strukturu. Tato struktura v asynchronní komunikaci se skládá z:

- START bit: Jedná se o bit, který indikuje inicializaci sériové komunikace. Jeho hodnota je vždy logická „0“ nebo „Low“.
- Data bit paket: Datové bity mohou být v paketech pěti až devíti bitů. Normálou je 8 bitový datový paket, který je vždy poslán po START bitu.
- STOP bit: Tento bit je vždy za datovými bity a indikuje konec. Jeho hodnota je vždy logická „1“ nebo „High“.[16]

Obr. 2.3 Struktura rámce sériového přenosu UART[16]

Rámec asynchronního sériového přenosu nejčastěji obsahuje 10 bitů. Jedná se o počáteční START bit, který je následován 8 bity dat a jako poslední je jeden STOP bit. Jiné možné struktury rámce se lišší v počtu datových bitů, kterých může být 5 až 9, a případnými dvěma STOP bity místo jednoho. [16]

Samotné Raspberry Pi obsahuje dvě varianty UART. Jednou je PL011 UART a druhou pak mini UART. PL011 UART je založen na architektuře ARM a má lepší propustnost. Mini UART využívá frekvence jádra grafického procesoru, a proto pokud se mění frekvence grafického procesoru, mění se i frekvence UART. Tato skutečnost vede

k nestabilitě a možné ztrátě dat nebo jejich poškození. Možným řešením je uvést stálou frekvenci grafického procesoru a tím se vyhnout jeho změnám. Obě varianty mají své využití, nicméně PL011 UART se využívá mnohem častěji díky stabilnějšímu a vyššímu výkonu. [16]

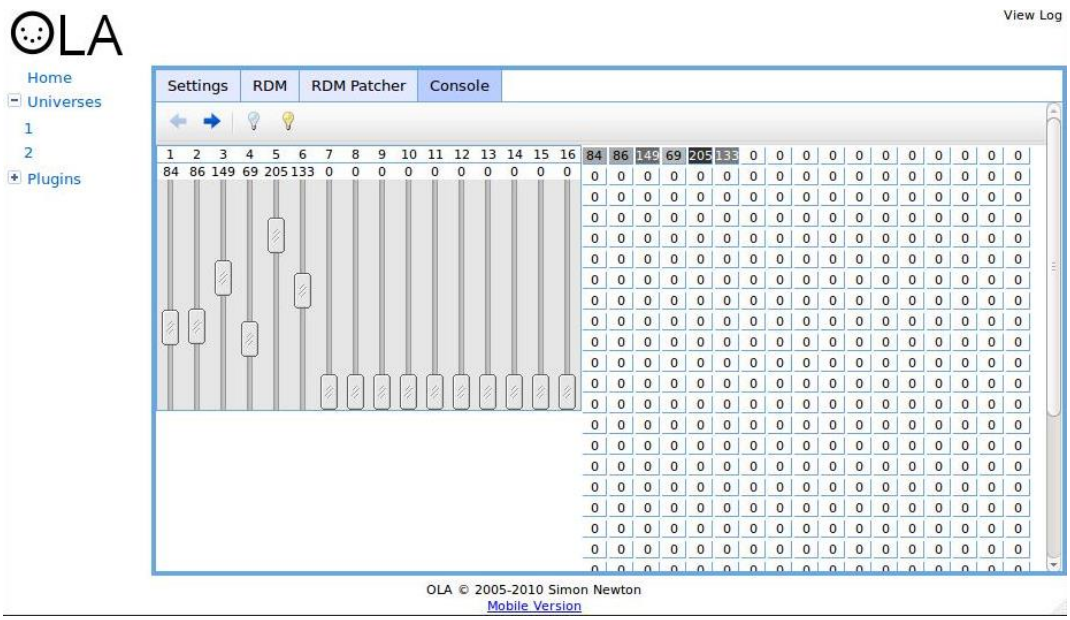
Při návrhu software jsem využil Raspberry Pi 3 pro odlaďování samotného softwaru díky jeho rychlejšímu procesoru. Po odladění byl software dále používán ve finální formě na Raspberry Pi 1. K odlišnosti mezi těmito dvěma zařízeními došlo pouze při konfiguraci zmíněného UARTu a u nainstalované verze operačního systému. Odlišná konfigurace je popsána v kapitole 5.2. Nainstalování softwaru OLA a konfigurace UART a odlišné operační systémy pak v kapitole 5.1 Instalace Raspberry Pi OS.

3 Možnosti generování DMX512 dat

V této části jsem se zabýval a popsal dvě nejvíce používané možnosti jak generovat DMX512 data pomocí počítače Raspberry Pi.

3.1 Open Lighting Architecture(OLA)

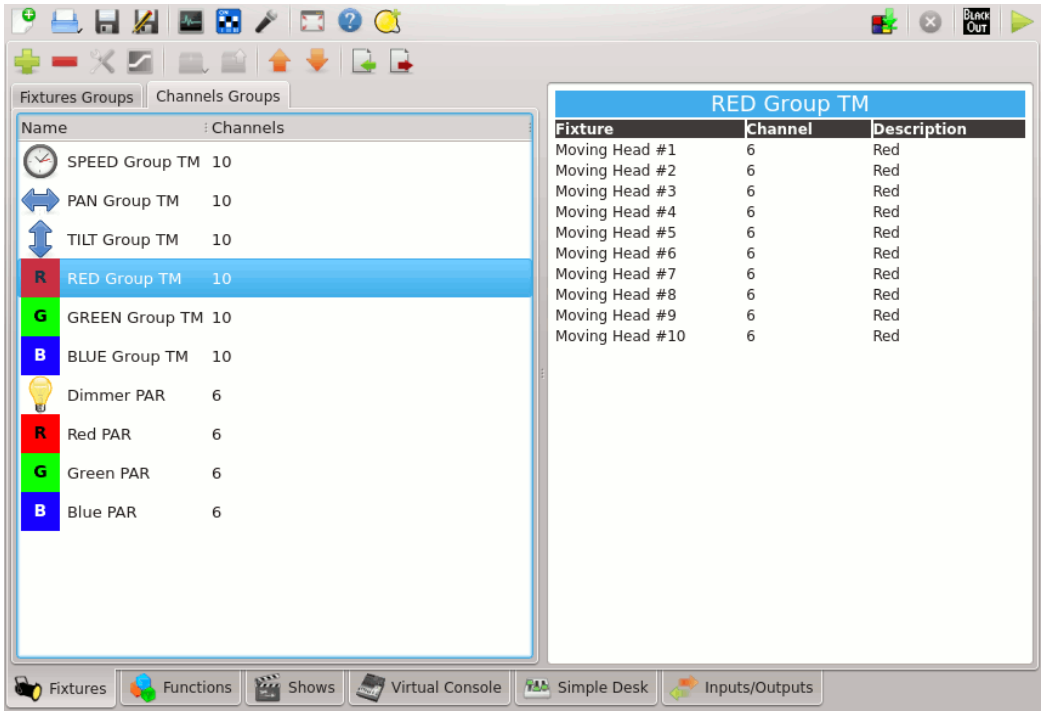
Jednou z velmi využívaných metod pro realizaci řízení DMX je OLA (Open Lighting Architecture). Jedná se o software, který je určený pro operační systémy distribuce Linuxu a Mac OS. Současně podporuje platformy ARM a je tedy vhodnou možností pro Raspberry Pi. Návody pro instalaci na jednotlivé platformy je možné najít přímo na oficiálních stránkách[18]. Aktuální verze OLA 0.10.9 umožnuje nejenom generování dat DMX512 protokolu, ale také může sloužit pro přijímání ethernetových protokolů sACN a Art-Net nebo připojení různých USB adaptérů. Pro jednoduché ovládání a konfiguraci řízení osvětlení poskytuje grafické prostředí, které je přístupné na lokální IP adrese Raspberry Pi s portem: 9090. Je tedy možné grafické prostředí obsluhovat přímo na Raspberry Pi, ale i na jiném zařízení připojeném na společnou lokální síť. Samotné grafické prostředí je zobrazeno na Obr. 3.1. OLA zároveň umožnuje řízení skrze příkazového klienta v příkazovém terminálu Raspberry Pi pomocí jednoduchých příkazů. [17][19]



Obr. 3.1 Grafické prostředí softwaru OLA

3.2 Q Light Controller Plus (QLC+)

Další možností je Q Light Controller Plus (QLC+). Tento software je možný využít na operačních systémech distribuce linuxu, Windows, Mac OS a mnoha platformách včetně Raspberry Pi. Podobně jako OLA aktuální verze QLC+ 4.12.6 podporuje příjem protokolů sACN a Art-Net a případné připojení USB adaptérů. Software se zaměřuje především na jednoduchost samotného řízení osvětlení a nabízí velmi intuitivní grafické prostředí, kterým je vše ovládáno a konfigurováno. Grafické prostředí je zobrazeno na Obr. 3.2. Návody pro použití tohoto softwaru jsou přístupné na oficiálních stránkách[21], ale návody pro instalaci dostupné na stránkách nejsou. [20]



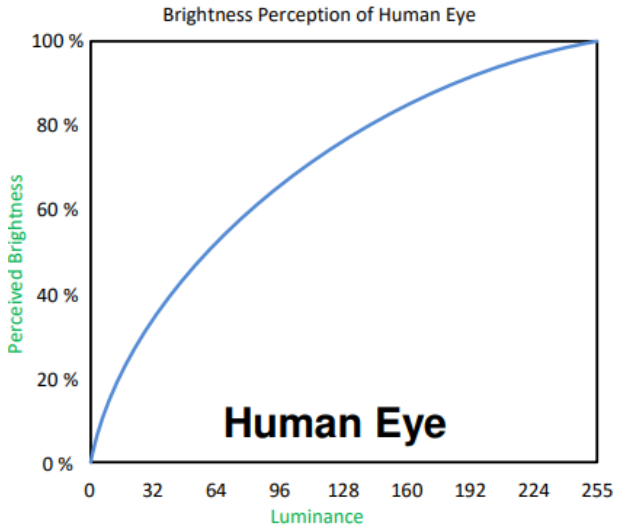
Obr. 3.2 Grafické prostředí softwaru QLC+[20]

Po prostudování již používaných metod realizace generování dat DMX512, jsem se rozhodl, využít software OLA. Důvodem je možnost využití jeho funkcí společně s mým vytvořeným softwarem pro přijímání TCP/IP paketů. Toho je možné docílit díky klientu v příkazovém terminálu na Raspberry Pi. Software QLC+, ačkoliv se jeví jako velmi intuitivní možnost, vyžaduje obsluhu řízení osvětlení pomocí grafického prostředí nebo přijímání protokolů sACN a Art-Net. Z tohoto důvodu je pro můj konkrétní případ tento software nevhodný.

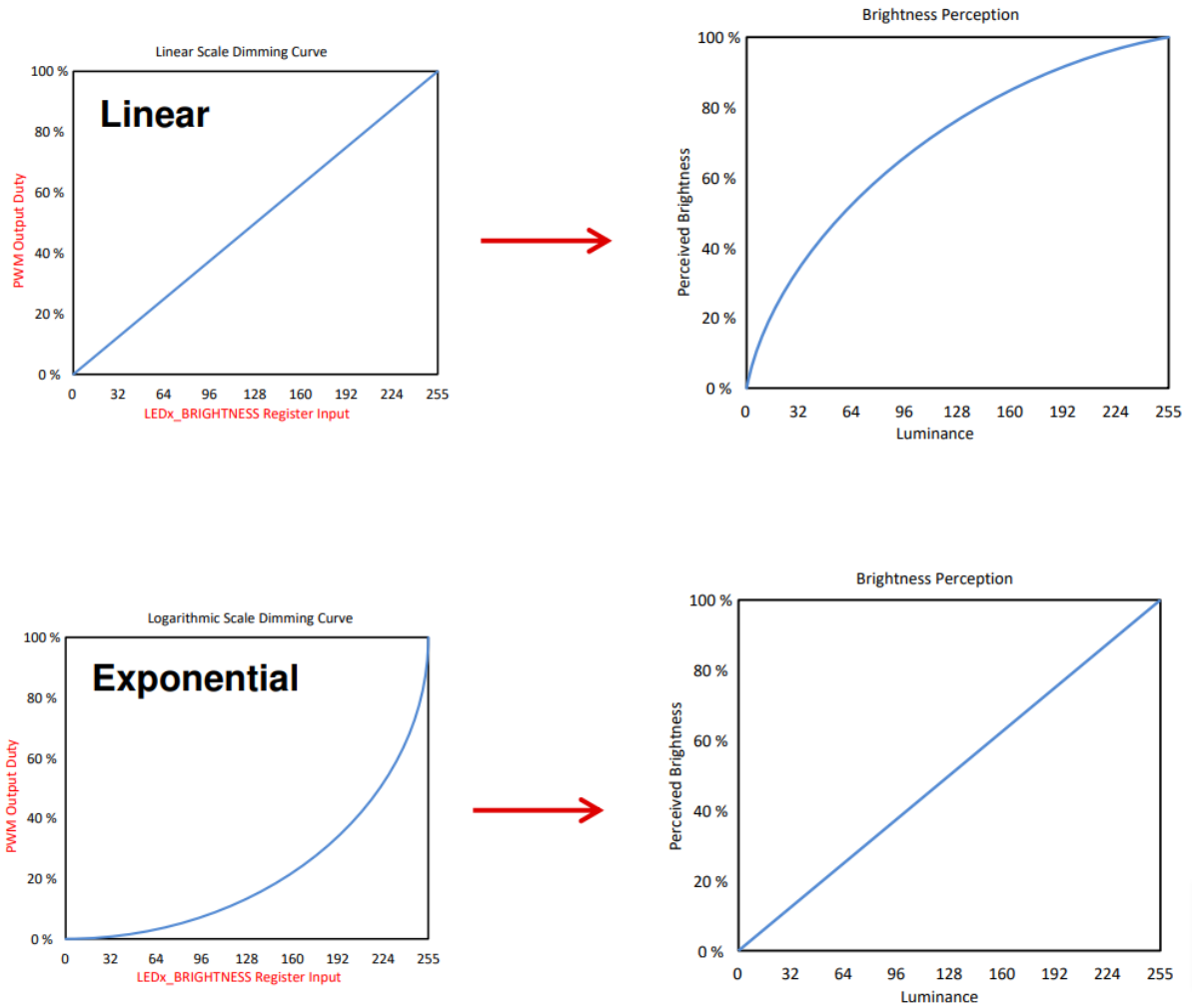
4 Křivky plynulých přechodů

Součástí návrhu softwaru je realizace plynulých přechodů mezi různými scénami. V této části jsou popsány různé metody plynulých přechodu u osvětlovací techniky.

U řídicí techniky pro osvětlení se často setkáváme s pojmy „stmívací křivky“, které představují, jakým způsobem dochází k přechodům mezi scénami. Cílem je zajistit co nejplynulejší přechod z pohledu lidského oka. Lidské vnímání jasu není lineární s jasem výkonových jednotek a je citlivější při nízkém jasu, přibližuje se tedy spíše křivce logaritmické. Pokud by byla tedy používaná metoda řízení stmívání lineárního charakteru, konečné vnímání změny jasu lidským okem není lineární. Pokud chceme tedy, aby vnímaný přechod byl lineární, musíme použít křivku exponenciální. V praxi se člověk přesto může setkat i s dalšími křivkami a záleží pouze na návrháři, jaký typ křivky přechodu je pro jeho konkrétní účel nejvhodnější. [22][23]



Obr. 4.1 vnímání lidského oka[23]



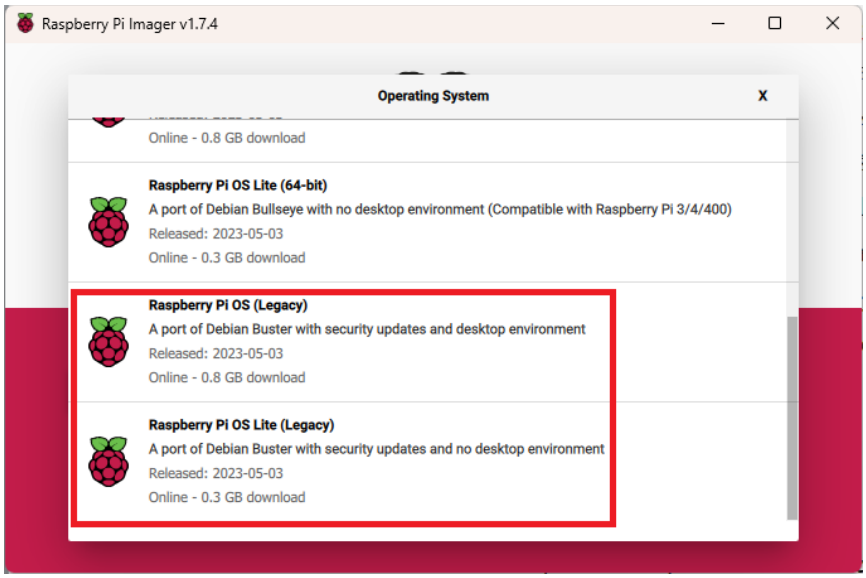
Obr. 4.2 Porovnání vnímání lineárního a exponenciálního přechodu[23]

5 Návrh softwaru

V předchozích kapitolách jsem shrnul podstatné technické pojmy a provedl rešerši již obdobných metod realizace. V následující části jsem se po samotném pochopení této problematiky věnoval realizaci softwaru pro Raspberry Pi, v součinnosti s vyrobenou deskou. Navržený program a skripty jsem vytvořil pomocí programovacího jazyka Python.

5.1 Installace Raspberry Pi OS

Jako první jsem se musel zvolit správnou verzi operačního systému Raspberry Pi OS, který bude kompatibilní se softwarem OLA. Zvolená verze operačního systému se pak vždy musí nejprve předinstalovat na mikroSD kartu pomocí aplikace Raspberry Pi Imager na externím počítači s operačním systémem Windows. Minimální požadavek na kapacitu použité mikroSD karty je alespoň 4 GB. Po nahrání operačního systému na kartu je možné ji vložit do Raspberry Pi a spustit ho. Od roku 2021 jsou k dispozici novější verze operačního systému zvané Bullseye a starší verze Legacy. Software OLA s aktuální verzí 0.10.9 je postaven pro Legacy Raspberry Pi OS a přestože je možné software nainstalovat i na novější Bullseye, není zaručeno, že všechny funkce budou pracovat, jak mají. Nainstaloval jsem tedy Legacy Raspberry Pi OS, přičemž na Raspberry Pi 1 se jednalo o verzi bez grafického rozhraní.



Obr. 5.1 Raspberry Pi Imager

5.2 Nainstalování softwaru OLA a konfigurace UART

Před instalací softwaru je vhodné provést aktualizaci systému pomocí příkazu v příkazovém terminálu:

```
sudo apt-get update
```

Následně pak samotnou instalaci jsem provedl dalším příkazem, pomocí kterého se nainstaluje aktuální verze OLA 0.10.9:

```
sudo apt-get install ola
```

Pro zprovoznění UART sériové komunikace bylo zapotřebí přidat řádek:

```
enable_uart=1
```

do `/boot/config.txt`, aby byl port použitelný. Základní přenosová rychlost je ovšem 115200bit/s, což je méně než požadovaných 250kbit/s pro DMX512. bylo zapotřebí přidat do stejného souboru řádek, aby byl základní limit zvýšen:

```
init_uart_clock=16000000
```

U Raspberry Pi 3 a 4 je v základním nastavení pro Bluetooth využíván rychlejší UART a pro sériovou komunikaci pomalejší UART. To bylo pro mne nevhodné, a proto bylo zapotřebí uvolnit rychlejší UART vypnutím funkce Bluetooth. Toto bylo možné docílit přidáním dalšího řádku do stejného souboru, jako v předchozím kroku:

```
dtoverlay=disable-bt
```

Tento příkaz bylo možné přeskočit na Raspberry Pi 1.

Aby UART nebyl využíván jinými aplikacemi, bylo zapotřebí vypnout službu, která dodává k němu přístup. Dospěl jsem toho odstraněním části řádku:

```
console=ttyAMA0,115200
```

ze souboru `/boot/cmdline.txt` a provedením příkazu v příkazovém terminálu:

```
sudo systemctl disable serial-getty@ttyAMA0.service
```

Jako poslední bylo zapotřebí přidat uživatele `pi` do skupiny, aby měl přístup k UART portu:

```
sudo usermod –a –G dialout pi.
```

5.3 Nastavení pinů pro posílání dat po UART lince

Ze schématu vyrobené desky je zřejmé, že je nutné přivést na GPIO 23 logickou „1“. Zároveň bylo potřeba uvést GPIO 14 a 15, které jsou určeny pro UART komunikaci, do módu ALT0, který umožnuje sériové posílání dat. Toto bylo uskutečněno pomocí WiringPi knihovna, který umožnuje snadné ovládání všech GPIO pinů. V Legacy verzi Raspberry Pi OS je tato knihovna předinstalována a mohl jsem ji rovnou používat. Aby byla vyváděna na vyrobenou desku logická „1“ po GPIO 23, bylo zapotřebí příkazů:

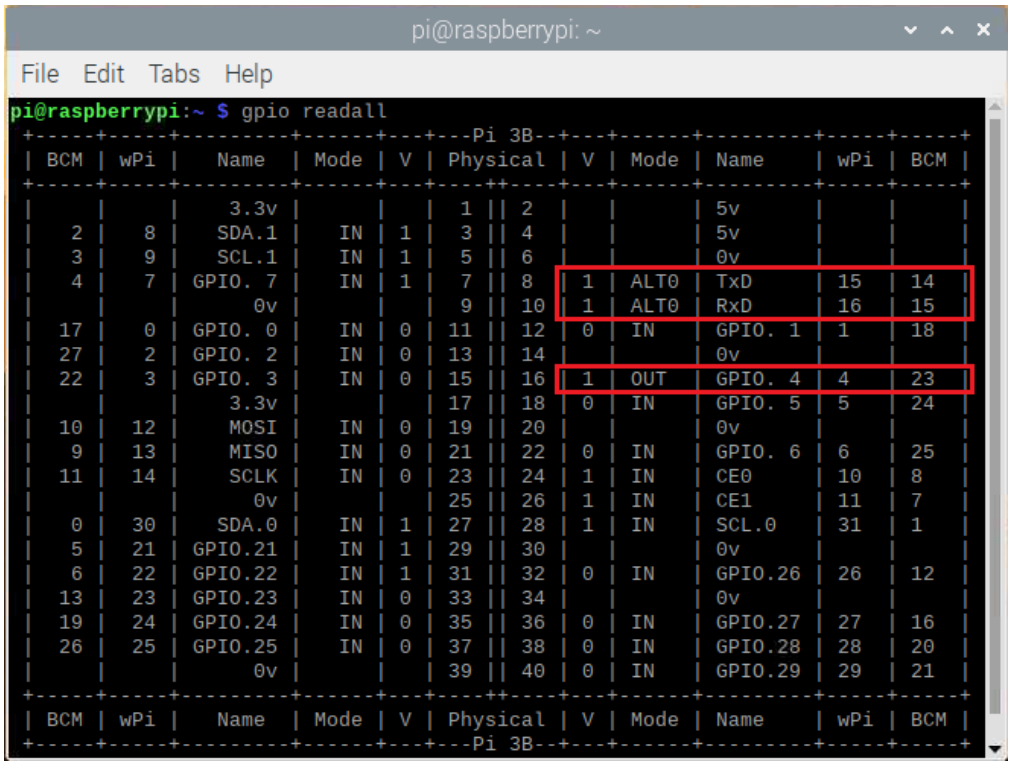
```
gpio -g mode 23 out
gpio -g write 23 1
```

Pro uvedení GPIO 14 a 15 do režimu ALT0 jsou pak příkazy:

```
gpio -g mode 14 alt0
gpio -g mode 15 alt0
```

Stav všech pinů lze pak zkontrolovat pomocí příkazu:

```
gpio readall
```



Obr. 5.2 Ukázka příkazu gpio readall

Všechny příkazy byly možné díky knihovně WiringPi.

Následně můžeme vytvořit skript, který provede tyto příkazy. Skript jsem nazval `uartenable.py` [B] a byl vytvořen přímo na Raspberry Pi pomocí editoru nano:

```
import os

os.system('gpio -g mode 23 out')
os.system('gpio -g write 23 1')
os.system('gpio -g mode 14 alt0')
os.system('gpio -g mode 15 alt0') [B]
```

5.4 Nastavení a konfigurace softwaru OLA a UART pluginu

Aby samotný software OLA posílal DMX512 data po UART sériové lince správně, bylo nutné nejprve upravit konfigurační soubor OLA pluginu, určený pro UART DMX komunikaci `/etc/ola/ola-uartdmx.conf`. Zde bylo zapotřebí určit vnitřní UART zařízení a hodnoty pro „break“ a „malf“.

```
device = /dev/ttyAMA
enable = true
/dev/ttyAMA-break = 100
/dev/ttyAMA0-malf = 24000
```

5.5 Příjem TCP/IP paketů

Pro příjem jednoduchého příkazu v podobě TCP/IP paketů na určitém portu Raspberry Pi jsem použil knihovnu NetCat. Knihovna mi umožnila provádět odposlech na mnou zvoleném TCP portu a následně data dále zpracovat. Pokud dojde při odposlechu k přijetí dat, odposlech je ukončen. Proto jsem pro můj konkrétní účel vytvořil nekonečnou smyčku, protože se předpokládá, neustálý odposlech.

```
while True:
    process = subprocess.Popen(['nc', '-l', '4444'],
    stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    output, error = process.communicate()
    PortData = output.decode('utf-8')
```

5.6 Vytvoření scén

Pro vytvoření scén jsem vytvořil konfigurační soubor, ve kterém lze určit jednotlivé scény. Při návrhu se počítalo s jasným počtem 5 scén, ale jejich hodnoty mohou být nastavitelné. Konfigurační soubor tedy obsahuje 5 scén (S1-S5) a proměnné určené pro výpočet mezistavů při přechodech mezi nimi. Konkrétně proměnná c1 představuje kolik mezistavů bude v časovém úseku jedné sekundy, proměnná c2 představuje kolik sekund daný přechod trvat. Představují tedy plynulost přechodu a jeho rychlost. Zároveň jsem definoval proměnnou fade, která určuje jaký typ přechodu se používá. Například scéna 1(S1) má nastavenou hodnotu pro kanál 1 a 4 hodnotu 255, mezistavů za 1 sekundu bude 24, přechod bude trvat 2 sekundy a samotný přechod bude lineárního typu. Konfigurační soubor jsem nazval `program1.conf` [C] a pro popsaný příklad vypadá následovně:

```
[Section1]
S1=255,0,0,255
S2=0,255,0,255
S3=0,0,255,255
S4=128,128,0,255
S5=128,128,128,128

[Section2]
c1=24
c2=2

[Section3]
fade=linear [C]
```

Konfigurační soubor je pak použit v programu prostřednictvím knihovny konfigparser. Díky knihovně jsou vytvořeny proměnné pro program a jsou dále zpracovány.

```
def create_variables_from_config(config_file):
    config = configparser.ConfigParser()
    config.read(config_file)
    variables = {}
    for section in config.sections():
        for option in config.options(section):
            value = config.get(section, option)
            if ',' in value:
                values = value.split(',')
                variables[option.upper()] = [int(v)
                for v in values]
            else:
                try:
                    variables[option] = int(value)
                except ValueError:
                    variables[option] = value
    return variables
```

5.7 Realizace přechodů mezi scénami

Při vytváření přechodů mezi scénami jsem využil již definovaných proměnných scén a proměnných c1 a c2. Zároveň jsem vytvořil nové pracovní proměnné c3, cha (channelA), chb (channelB) a chdiff (channelDiff), které slouží pro výpočty mezistavů přechodů.

Přechod funguje tak, že mám známou aktuální scénu, ve které se nacházím, a z příkazu přijatého na TCP portu převezmu cílovou scénu, do které se chci dostat. Tyto hodnoty jsou přiděleny proměnným cha (aktuální scéna) a chb (cílová scéna). Rozdíl hodnot těchto proměnných je pak reprezentován proměnou chdiff. Výpočet mezistavů jsem pak získal pomocí této proměnné a proměnné c3.

Přechody mohou proběhnout ve 3 variantách: lineárně, logaritmicky a exponenciálně. Při výpočtech pro logaritmický a exponenciální přechody jsem využil knihovnu numpy, která slouží pro matematické výpočty tohoto druhu. K jakému přechodu

dojde, je pak zcela závislé na volbě v konfiguračním souboru. Funkce jednotlivých přechodů jsou zobrazeny níže.

- Lineární přechod

```
def fade_linear(ch, cha, chb, cas):
    chdiff = [(x - y) / cas for x, y in zip(cha, chb)]
    for i in range(cas-1):
        ch = [x + y for x, y in zip(ch, chdiff)]
        output = ",".join(str(x) for x in ch)
        os.system(f'ola_streaming_client -u 1 -d
        {output}')
```

- Logaritmický přechod

```
def fade_log(ch, cha, chb, cas):
    chdiff = [(x - y) for x, y in zip(cha, chb)]
    x = np.linspace(0.1, 10, cas)
    y = np.log10(x)
    y_scaled = (np.round((y - np.min(y)) *
    (100 / (np.max(y) - np.min(y)))) / 100)
    for i in range(cas-1):
        chlog = [x * y_scaled[i] for x in chdiff]
        ch = [x + y for x,y in zip(ch, chlog)]
        output = ",".join(str(x) for x in ch)
        os.system(f'ola_streaming_client -u 1 -d
        {output}')
        ch = [x - y for x,y in zip(ch, chlog)]
```

- Exponenciální přechod

```
def fade_exp(ch, cha, chb, cas):
     chdiff = [(x - y) for x, y in zip(cha, chb)]
     x = np.linspace(0.1, 10, cas)
     y = np.exp(x / 10)
     y_scaled = (np.round((y - np.min(y)) *
     (100 / (np.max(y) - np.min(y)))) / 100)
     for i in range(cas-1):
          chexp = [x * y_scaled[i] for x in chdiff]
          ch = [x + y for x,y in zip(ch, chexp)]
          output = ",".join(str(x) for x in ch)
          os.system(f'ola_streaming_client -u 1 -d
          {output}')
          ch = [x - y for x,y in zip(ch, chexp)]
```

Následně pak pro mě bylo již snadné vytvořit několik jednoduchých podmínek pro příchozí příkaz z TCP portu a proměnné fade, které následně provedou určený přechod. Ukázka pro přechod do S1 (scény 1) je zobrazen níže:

```
if PortData == "S1":
     cha = S1
     chb = ch
     if fade == "log":
          fade_log(ch, cha, chb, c3)
     elif fade == "exp":
          fade_exp(ch, cha, chb, c3)
     else:
          fade_linear(ch, cha, chb, c3)
     ch = S1
     output = ",".join(str(x) for x in ch)
     os.system(f'ola_streaming_client -u 1 -d {output}')
```

Všechny funkce pro příjem TCP příkazu, jejich zpracování a provedení přechodů jsem následně převedl do jednoho skriptu, který nese název `program1.py`. Jeho finální forma je dostupná v příloze [D].

5.8 Automatizace spouštění skriptů

Aby nebylo nutné vytvořené skripty provádět po každém spuštění Raspberry Pi, je vhodné tento proces zautomatizovat. Do této doby jsem vždy musel spuštění skriptů provést po zapnutí Raspberry Pi příkazy:

```
sudo python3 /home/pi/uartenable.py
sudo python3 /home/pi/program1.py
```

Tyto příkazy jsem pak vložil do souboru `/etc/rc.local` a díky tomu jsou automaticky provedeny při spuštění Raspberry Pi. Do stejného souboru jsem také vložil řádek, aby se automaticky spouštěl software OLA:

```
Su pi -c olad
```

Zhodnocení a závěr

V této bakalářské práci jsem se zabýval problematikou pro návrh a realizaci řídicího softwaru osvětlení pro jednosálové kino. Díky digitalizaci jevištní techniky je realizace oproti analogovým variantám levnější a umožňuje více metod pro jejich řízení. Pro přenos byl standardizován protokol DMX512 a v roce 2016 byly vydánu nejnovější verze protokolů pro jejich přenos pomocí ethernetové sítě. Tyto protokoly dále rozšiřují metody řízení a jejich finanční dostupnost. Dodnes nejčastější metodou je ovládání pomocí osvětlovacích pultů, které přímo podporují zmíněné protokoly, ale vyžadují obsluhu, nepatří k nejlevnějším metodám. Pro můj konkrétní případ byl tedy nastaven cíl navržení softwaru pro počítač Raspberry Pi s využitím poskytnuté desky, která je připojena na GPIO Raspberry Pi, aby bylo možné nahradit osvětlovací pult.

V první části jsem popsal technické pojmy, které pro pochopení práce byly nezbytné, a zároveň provedl rešerši obdobných již realizovaných metod. Bylo zřejmé, že pro můj konkrétní případ, žádná z těchto metod nesplňuje všechny podmínky, které na práci byly kladeny. Především se jednalo o možnost posílání jednoduchého příkazu na TCP/IP socket Raspberry Pi.

Ačkoliv žádná z metod nesplňovala nastavené podmínky, bylo možné alespoň využít pro část návrhu software OLA. Ten mi poskytl generování dat DMX512 po sériové lince UART z Raspberry Pi do poskytnuté desky.

V další části jsem pak popsal jednotlivé postupy při návrh samotného softwaru. Jednalo se o instalaci a nastavení operačního systému pro Raspberry Pi, nainstalování a konfiguraci softwaru OLA a vytvoření mnou navržených 2 skriptů. První skript slouží pro správné nastavení GPIO, aby byla možná sériová komunikace od Raspberry Pi na poskytnutou desku, a druhý skript pak realizuje příjem příkazů na TCP/IP socketu, jejich zpracování a realizaci přechodů mezi definovanými scénami. Jako poslední bylo popsáno, jakým způsobem jsem zajistil automatizaci celého softwaru.

Řízení osvětlení tedy funguje tak, že z externího počítače připojeného na stejnou ethernetovou sít pošle jednoduchý příkaz (např S1), počítač Raspberry Pi tento příkaz přijme a provede plynulý přechod z aktuální scény do scény nové.

Jako nevýhodu mého návrhu vnímám, že definice scén a jejich přechodů je prováděna na samotném počítači Raspberry Pi. Pokud bychom chtěli změnit charakter předdefinovaných scén, je nutné tak učinit v mnou vytvořeném konfiguračním souboru. Opodstatnění této nevýhody je v požadavku, aby posílaný příkaz byl co nejjednodušší a jednalo se o jednorázový impulz.

Tento návrh byl realizován a testován v laboratorním prostředí katedry KEI. Zároveň bude použit v jednosálovém kině jako náhrada za osvětlovací pult. Bakalářská práce mi umožnila prohloubit si znalosti, které jsem nabyl v průběhu studia, ale zároveň také umožnila získat znalosti nové, ať už z hlediska programovacího jazyka Python, se kterým jsem do té doby nepracoval, ale i v oblasti jevištní techniky.

Použité prameny a literatura

[1] Filip Brož, DMX controller s využitím Raspberry Pi, 2021 [online] [cit. 3.12.2022] dostupné z: https://dspace5.zcu.cz/bitstream/11025/44519/1/BP_DMX_controller_s_vyuzitim_Raspberry_Pi.pdf

[2] SOH, Protokol DMX512 [online] [cit. 3.12.2022] dostupné z: http://www.soh.cz/podpora/teorie

[3] DMX 512, DMX protocol [online] [cit. 3.12.2022] dostupné z: http://www.dmx-512.com/dmx-protocol/dmx-protocol-basics/

[4] Art-Net 4, Specification for Art-Net 4 Ethernet Communication Protocol [online] [cit. 3.12.2022] dostupné z: https://www.artisticlicence.com/WebSiteMaster/User%20Guides/art-net.pdf

[5] Wayne Howell, sACN in large systems, Pt. 1 [online] [cit. 8.1.2023] dostupné z: https://artisticlicence.com/WebSiteMaster/Publicity/HelpDesk18-sACN-in-large-systems-Part1-Nov2018.pdf

[6] Doityourselfchristmas, E1.31 (Streaming-ACN) Protocol [online] [cit. 8.1.2023] dostupné z: https://www.doityourselfchristmas.com/wiki/index.php?title=E1.31_(Streaming-ACN)_Protocol

[7] Art-Net, ArtTimeCode [online] [cit. 8.1.2023] dostupné z: https://art-net.org.uk/how-it-works/time-keeping-triggering/arttimecode/

[8] Wikiimproliga, Osvětlovač, Osvětlovací pult [online] [cit. 12.1.2023] dostupné z: https://wiki.improliga.cz/wiki/Osv%C4%9Btlova%C4%8D

[9] Eshop SOH, DDC-12 DMX ovládací pult s LCD displejem [online] [cit. 12.1.2023] dostupné z: http://eshop.soh.cz/dmx-pulty/dmx-pulty/i752-ddc-12-dmx-ovladaci-pult

[10] Wikipedia, Raspberry Pi [online] [cit. 25.1.2023] dostupné z: https://en.wikipedia.org/wiki/Raspberry_Pi

[11] RPishop, Raspberry Pi 1 Model B [online] [cit. 25.1.2023] dostupné z: https://rpishop.cz/raspberry-pi-1b/74-raspberry-pi-707565827133.html

[12] Raspberry Pi, Operating system images [online] [cit. 13.2.2023] dostupné z: https://www.raspberrypi.com/software/operating-systems/#raspberry-pi-os-32-bit

[13] Raspberry Pi, GPIO pins [online] [cit. 13.2.2023] dostupné z: https://projects.raspberrypi.org/en/projects/physical-computing/1

[14] Pinout, Raspberry Pi Pinout [online] [cit. 13.2.2023] dostupné z: https://pinout.xyz/

[15] Trikarus Project Overview, Raspberry Pi 3 B – GPIO config [online] [cit. 13.2.2023] dostupné z: https://stadtfabrikanten.org/display/TH/Raspberry+Pi+3+B+-+GPIO+config

[16] ElectronicWings, Raspberry Pi UART Communication using Python and C [online] [cit. 13.2.2023] dostupne z: https://www.electronicwings.com/raspberry-pi/raspberry-pi-uart-communication-using-python-and-c

[17] Open Lighting Architecture, Open Lighting Project [online] [cit. 7.3.2023] dostupné z: https://www.openlighting.org/

[18] Open Lighting Architecture, Tutorials [online] [cit. 7.3.2023] dostupné z: https://www.openlighting.org/ola/tutorials/

[19] Github, The Open Lighting Architecture – The Travel Adaptor for the Lighting Industry [online] [cit. 7.3.2023] dostupné z: https://github.com/OpenLightingProject/ola

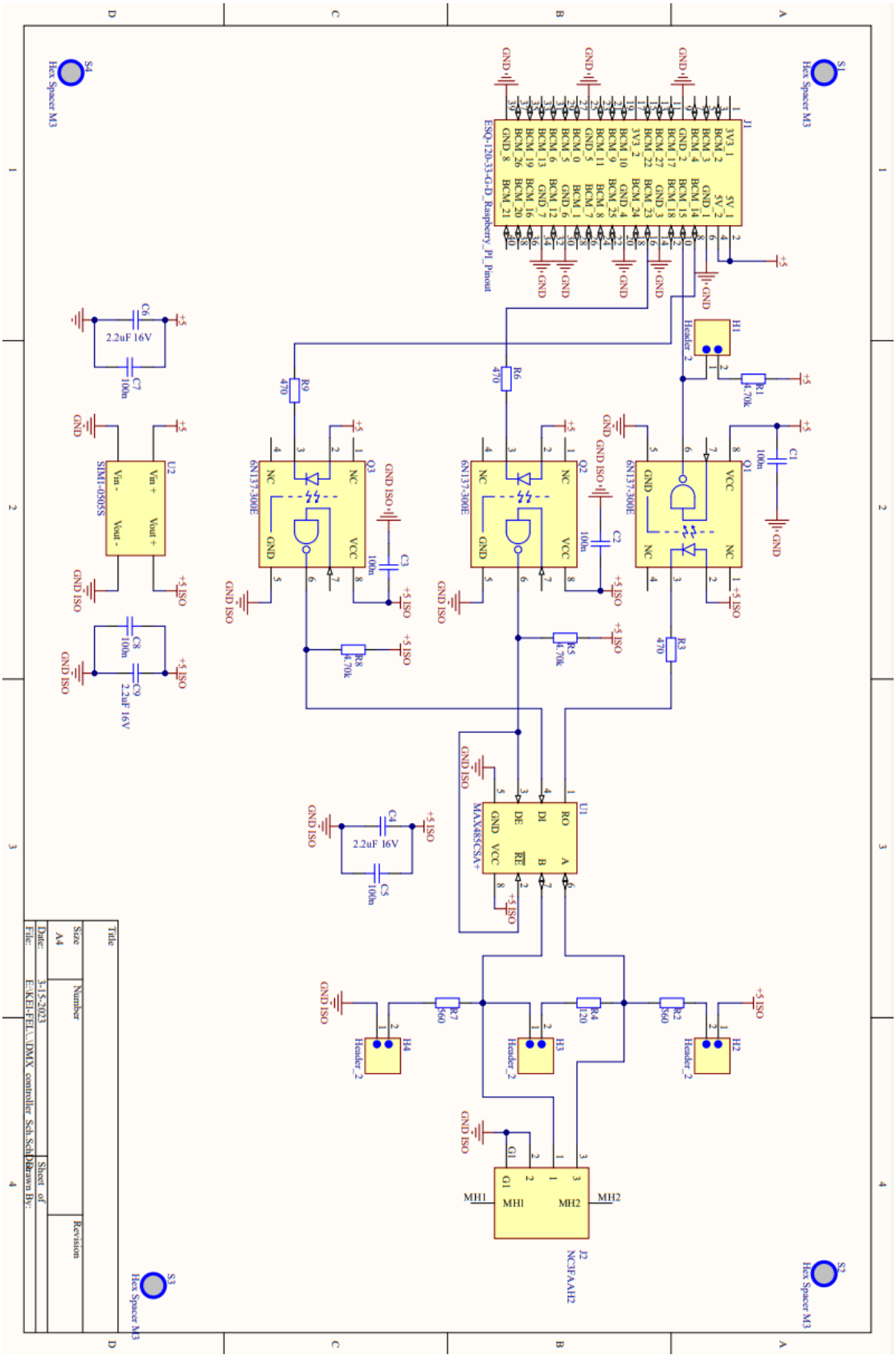
[20] QLCplus, QLC+ introduction [online] [cit. 14.3.2023] dostupné z: https://www.qlcplus.org/index.php

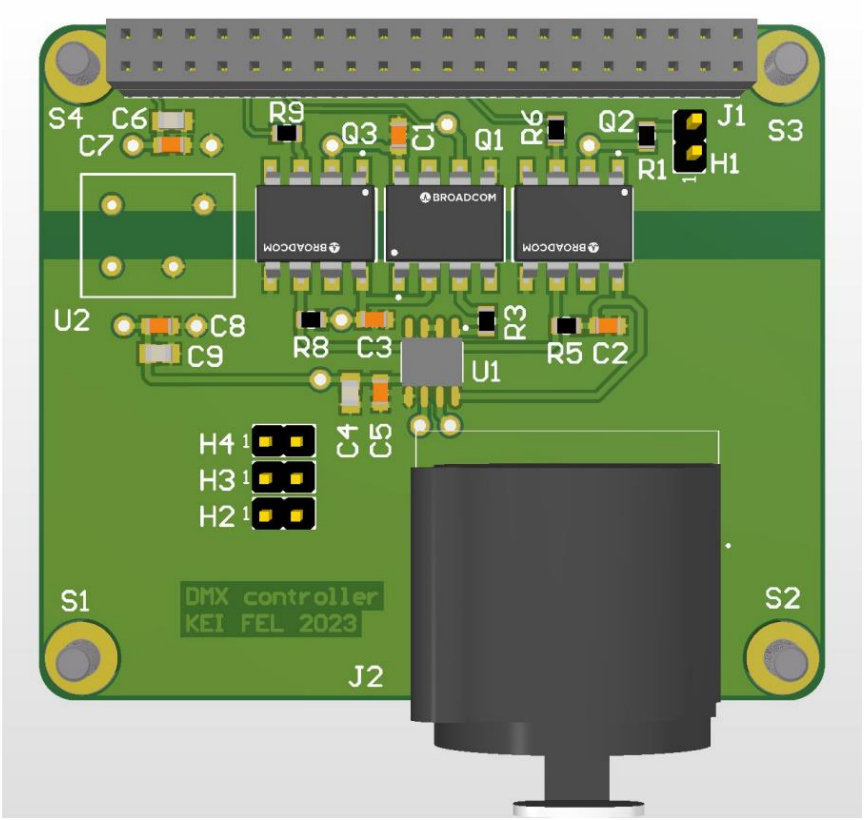
[21] QLCplus, Official QLC+ tutorials [online] [cit. 18.3.2023] dostupné z: https://www.qlcplus.org/tutorials.php

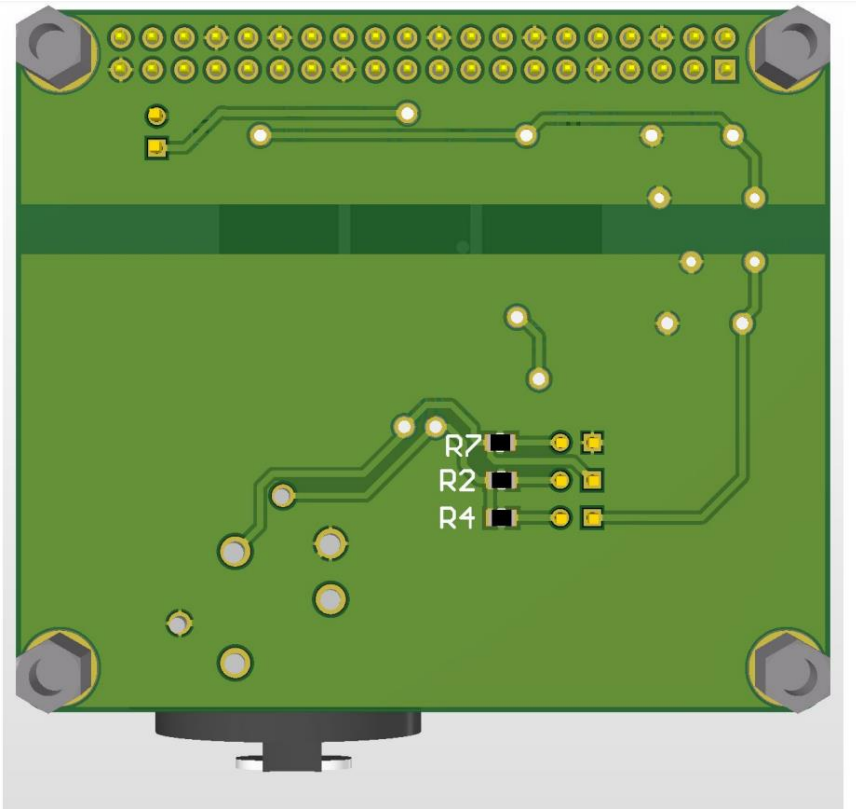
[22] uPowerTek, What are dimming Curves and How to Choose? [online] [cit. 20.4.2023] dostupné z: https://www.upowertek.com/what-are-dimming-curves-and-how-to-choose/

[23] Texas instruments, How to Select a RGB :ED Driver [online] [cit. 20.4.2023] dostupné z: https://www.ti.com/lit/an/slvaef3/slvaef3.pdf?ts=1684917583043&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FLP5009

Příloha A


ESQ-120-33-G-D_Raspberry_PI_Pinout
SIM1-0505S
6N137-300E
MAX485CSA+
NC3FAAH2
Hex Spacer M3
2.2uF 16V
100n
Header_2
Title
Size
A4
Number
Revision
Date:
3-15-2023
Sheet of
File:
E:\KEI-FEL\...\DMX controller Sch.SchDoc
Drawn By:


S4
C6
C7
R9
Q3
C1
Q1
R6
Q2
J1
H1
S3
R1
BROADCOM
U2
C8
C9
R8
C3
R3
R5
C2
U1
H4
H3
H2
C4
C5
S1
DMX controller
KEI FEL 2023
S2
J2


R7
R2
R4

Příloha B

uartenable.py

```
import os

os.system('gpio -g mode 23 out') #BCM23 mode out
os.system('gpio -g write 23 1') #BCM23 output 1
os.system('gpio -g mode 14 alt0') #BCM14 mode alt0
os.system('gpio -g mode 15 alt0') #BCM15 mode alt0
```

Příloha C

program1.conf

```
[Section1]
S1=255,0,0,255
S2=0,255,0,255
S3=0,0,255,255
S4=128,128,0,255
S5=128,128,128,128

[Section2]
c1=24
c2=2

[Section3]
fade = linear
```

Příloha D

program1.py

```
import os
import time
import subprocess
import configparser
import numpy as np

def create_variables_from_config(config_file):
    config = configparser.ConfigParser()
    config.read(config_file)
    variables = {}
    for section in config.sections():
        for option in config.options(section):
            value = config.get(section, option)
            if ',' in value:
                values = value.split(',')
                variables[option.upper()] = [int(v) for v in
values]
            else:
                try:
                    variables[option] = int(value)
                except ValueError:
                    variables[option] = value
    return variables

config_file = '/home/pi/program1.conf'
variables = create_variables_from_config(config_file)

c1 = variables['c1']  # rychlost posilani za sekundu
c2 = variables['c2']  # Doba prechodu v sekundach
c3 = (c1 * c2)
```

```
fade = variables['fade']

ch = [0] * 512
S0 = [0] * 512

S1 = variables['S1'] + [0] * (512 - len(variables['S1']))
S2 = variables['S2'] + [0] * (512 - len(variables['S2']))
S3 = variables['S3'] + [0] * (512 - len(variables['S3']))
S4 = variables['S4'] + [0] * (512 - len(variables['S4']))
S5 = variables['S5'] + [0] * (512 - len(variables['S5']))

def fade_linear(ch, cha, chb, cas):
    chdiff = [(x - y) / cas for x, y in zip(cha, chb)]
    for i in range(cas-1):
        start_time = time.perf_counter()
        ch = [x + y for x, y in zip(ch, chdiff)]
        output = ",".join(str(x) for x in ch)
        os.system(f'ola_streaming_client -u 1 -d {output}')
        while time.perf_counter() - start_time < 1/c1:
            pass

def fade_log(ch, cha, chb, cas):
    chdiff = [(x - y) for x, y in zip(cha, chb)]
    x = np.linspace(0.1, 10, cas)
    y = np.log10(x)
    y_scaled = (np.round((y - np.min(y)) * (100 / (np.max(y)
- np.min(y)))) / 100)
    for i in range(cas-1):
        start_time = time.perf_counter()
        chlog = [x * y_scaled[i] for x in chdiff]
        ch = [x + y for x, y in zip(ch, chlog)]
        output = ",".join(str(x) for x in ch)
        os.system(f'ola_streaming_client -u 1 -d {output}')
        ch = [x - y for x,y in zip(ch, chlog)]
```

```
        while time.perf_counter() - start_time < 1/c1:
            pass

def fade_exp(ch, cha, chb, cas):
    chdiff = [(x - y) for x, y in zip(cha, chb)]
    x = np.linspace(0.1, 10, cas)
    y = np.exp(x / 10)
    y_scaled = (np.round((y - np.min(y)) * (100 / (np.max(y)
- np.min(y)))) / 100)
    for i in range(cas-1):
        start_time = time.perf_counter()
        chexp = [x * y_scaled[i] for x in chdiff]
        ch = [x + y for x,y in zip(ch, chexp)]
        output = ",".join(str(x) for x in ch)
        os.system(f'ola_streaming_client -u 1 -d {output}')
        ch = [x - y for x,y in zip(ch, chexp)]
        while time.perf_counter() - start_time < 1/c1:
            pass

while True:
    process = subprocess.Popen(['nc', '-l', '4444'],
stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    output, error = process.communicate()
    PortData = output.decode('utf-8').strip()

    if PortData == "S1":
        cha = S1
        chb = ch
        if fade == "log":
            fade_log(ch, cha, chb, c3)
        elif fade == "exp":
            fade_exp(ch, cha, chb, c3)
        else:
            fade_linear(ch, cha, chb, c3)
```

```
        ch = S1
        output = ",".join(str(x) for x in ch)
        os.system(f'ola_streaming_client -u 1 -d {output}')
    elif PortData == "S2":
        cha = S2
        chb = ch
        if fade == "log":
            fade_log(ch, cha, chb, c3)
        elif fade == "exp":
            fade_exp(ch, cha, chb, c3)
        else:
            fade_linear(ch, cha, chb, c3)
        ch = S2
        output = ",".join(str(x) for x in ch)
        os.system(f'ola_streaming_client -u 1 -d {output}')
    elif PortData == "S3":
        cha = S3
        chb = ch
        if fade == "log":
            fade_log(ch, cha, chb, c3)
        elif fade == "exp":
            fade_exp(ch, cha, chb, c3)
        else:
            fade_linear(ch, cha, chb, c3)
        ch = S3
        output = ",".join(str(x) for x in ch)
        os.system(f'ola_streaming_client -u 1 -d {output}')
    elif PortData == "S4":
        cha = S4
        chb = ch
        if fade == "log":
            fade_log(ch, cha, chb, c3)
        elif fade == "exp":
            fade_exp(ch, cha, chb, c3)
```

```
        else:
            fade_linear(ch, cha, chb, c3)
        ch = S4
        output = ",".join(str(x) for x in ch)
        os.system(f'ola_streaming_client -u 1 -d {output}')
    elif PortData == "S5":
        cha = S5
        chb = ch
        if fade == "log":
            fade_log(ch, cha, chb, c3)
        elif fade == "exp":
            fade_exp(ch, cha, chb, c3)
        else:
            fade_linear(ch, cha, chb, c3)
        ch = S5
        output = ",".join(str(x) for x in ch)
        os.system(f'ola_streaming_client -u 1 -d {output}')
    elif PortData == "OFF":
        cha = S0
        chb = ch
        if fade == "log":
            fade_log(ch, cha, chb, c3)
        elif fade == "exp":
            fade_exp(ch, cha, chb, c3)
        else:
            fade_linear(ch, cha, chb, c3)
        ch = S0
        output = ",".join(str(x) for x in ch)
        os.system(f'ola_streaming_client -u 1 -d {output}')
    else:
        continue
```