Univerzita Hradec Králové

Fakulta informatiky a managementu
Katedra informačních technologií

Implementace KNX modulu pomocí Arduino knihoven

Diplomová práce

Autor: Bc. David Tláskal
Studijní obor: Aplikovaná informatika (ai2-p)
Vedoucí práce: Ing. Pavel Kříž, Ph.D.

Hradec Králové srpen 2023

Prohlášení

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 10. 8. 2023 Bc. David Tláskal

Poděkování

Děkuji vedoucímu diplomové práce Ing. Pavlovi Křížovi, Ph.D. za odborné vedení, výpomoc a rady při zpracování této práce.

Abstrakt

Práce se zabývá základními principy systému KNX (Konnex), mezi které patří například jeho komunikační vrstvy, adresování, přenosová média, topologie atd. Dále je probíráno přenosové médium KNX TP (KNX Twisted Pair), které je jedním z hlavních předmětů práce. Krom základního porozumění KNX, je dále navržen a implementován software pro KNX zařízení (desku *H8I8O*).

Cílem práce bylo prozkoumat možnosti zmíněné desky a knihovny pro implementaci softwaru pro KNX, čehož bylo dosaženo díky implementovanému softwaru.

Abstract

Title: KNX module implementation using Arduino libraries

The thesis deals with the basic principles of the KNX (Konnex) system, including its communication layers, addressing, transmission media, topology, etc. Furthermore, the transmission medium KNX TP (KNX Twisted Pair) is discussed, which is one of the main subjects of the thesis. In addition to the basic understanding of KNX, the software for the KNX device (*H8I8O*) is also designed and implemented.

The aim of this thesis was to explore the capabilities of the mentioned board and library for implementing KNX software, which was achieved through the implemented software.

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

ADC Analog to Digital Converter. 31

API Application Programming Interface. 36

BCU Bus Coupling Unit. 26

CAN Controller Area Network. 31

CSMA/CA Carrier Sense Multiple Access with / Collision Avoidance. 6

EIB European Installation Bus. 2

ETS Engineering Tool Software. 22

GPIO General Purpose Input/Output. 30

HAL Hardware Abstraction Layer. 36

HBA Home and Building Automation. 2

I^2C Inter-Integrated Circuit. 30

IDE Integrated Development Environment. 58

IOT Internet Of Things. 3

IP Internet Protocol. 11

JSON JavaScript Object Notation. 54

JTAG Joint Test Action Group. 34

KNX IP KNX Internet Protocol. 11

KNX PL KNX Power Line. 11

KNX RF KNX Radio Frequency. 11

KNXD KNX Deamon. 26

LLC Logical Link Control. 6

LSB Least Significant Bit. 32

MAC Medium Access Control. 6

MAU Medium Attachment Unit. 15

MCU Microcontroller Unit. 30

OSI Open Systems Interconnection. 3

PCI Protocol Control Information. 4

PDU Protocol Data Units. 3

PWM Pulse Width Modulation. 29

SDU Service Data Units. 3

SELV Safety Extra Low Voltage. 10

SPI Serial Peripheral Interface. 30

SWD Serial Wire Debugging. 34

TCP Transmission Control Protocol. 26

TTL Time To Live. 18

UART Universal Asynchronous Receiver-Transmitter. 15

UDEV userspace / device. 38

USB Universal Serial Bus. 26

Wi-Fi Wireless Fidelity. 11

1 Úvod

V dnešní době, kdy se technologie neustále posouvají kupředu, se stává automatizace budov klíčovým faktorem pro dosažení vyšší účinnosti, komfortu a bezpečnosti v prostředí, ve kterém žijeme a pracujeme. S rostoucím zaměřením na udržitelnost a energetickou účinnost se očekává, že budovy budou vyžadovat sofistikovanější řízení a monitoring, aby byly schopny lépe reagovat na potřeby uživatelů a optimalizovat spotřebu energie.

Systém KNX se stal jedním z předních standardů v oblasti automatizace budov, díky své flexibilitě, otevřenosti a širokému spektru kompatibilních zařízení. Tento systém umožňuje integrovat a řídit různé funkce a zařízení v budově, jako jsou osvětlení, topení, klimatizace, žaluzie, bezpečnostní prvky a další, což vede ke zvýšení komfortu a efektivity provozu budovy.

1.1 Cíl práce

Cíl této práce je prozkoumat možnosti KNX a způsoby implementace KNX zařízení. Tento úkol je poměrně rozsáhlý a začíná u pochopení samotného systému KNX a principů na kterých je založený. Dále tento cíl vyžaduje zapojení testovací sběrnice KNX, která umožní testování zařízení včetně jeho konfigurace. Následně prozkoumat možnosti implementace softwaru zařízení (a jeho možnosti), tento software navrhnout, implementovat a otestovat. Následující kapitoly práce budou tedy věnovány technickým aspektům implementace zařízení KNX, včetně konfigurace, programování a komunikace mezi jednotlivými zařízeními.

2 KNX

KNX je standard otevřeného komunikačního protokolu pro domácí automatizaci a řízení budov, neboli HBA (Home and Building Automation). Jedná se o celosvětově uznávaný standard, který umožňuje propojení a komunikaci mezi různými zařízeními v inteligentních budovách. Mezi tato zařízení například patří osvětlení, topení, klimatizace, žaluzie, bezpečnostní systémy a další. [1]

Vzhledem k flexibilitě je KNX využíván ve všech možných budovách počínaje běžnými domy či byty, po nákupní centra, nemocnice, univerzity atd. Zatím co u běžných domů může být hlavní motivací pro použití KNX pohodlí, tak u větších budov může být motivací účinnost tohoto systému a fakt, že díky automatizaci, kterou KNX nabízí, je možné dosáhnout úspory energie. [2]

KNX je decentralizovaný systém, který je tzv. data driven. To znamená, že systém nepotřebuje žádnou řídící jednotku a při jeho vytváření byl kladen důraz na komunikace mezi zařízeními a na data předávaná touto komunikací. Jednou z hlavních myšlenek KNX je konfigurovatelnost zařízení. Díky této vlastnosti je dosaženo toho, že je tento sytém decentralizovaný. [1]

2.1 Historie KNX

Historie KNX se datuje od počátku 90. let 20. století. V té době existovalo několik izolovaných systémů domácí automatizace, avšak nedocházelo k jejich vzájemné kompatibilitě. V roce 1990 byla založena iniciativa EIB (European Installation Bus), která se snažila vytvořit sjednocený standard pro komunikaci mezi domácími zařízeními.

Během následujících let, mezi roky 1990 a 1999, se tato iniciativa EIB postupně vyvinula a přijala řadu technických specifikací a protokolů, které umožnily propojení a kompatibilitu různých zařízení. V roce 1999 byla iniciativa přejmenována na KNX, čímž se stala samostatnou organizací s mezinárodním dosahem.

KNX sjednotil a integroval několik existujících protokolů, včetně EIB, BatiBUS a EHS, a vytvořil tak komplexní a standardizovaný systém pro domácí automatizaci. Tento standard byl přijat a podporován stále více výrobci, což vedlo ke značnému rozšíření KNX v prvním desetiletí 21. století.

V roce 2011 byla přijata nová verze KNX s označením KNX TP1-256,

která umožňuje komunikaci až mezi 256 zařízeními v rámci jedné instalace. Tento standard byl dále standardizován jako evropská norma (EN 50090), mezinárodní norma (ISO/IEC 14543-3) a čínská norma (GB/Z 20965).

V posledních letech se KNX začal adaptovat na trendy IOT (Internet Of Things). Byly vyvinuty brány a rozhraní umožňující propojení KNX s jinými IoT zařízeními a cloudovými platformami, což uživatelům poskytuje větší flexibilitu a možnosti při řízení a ovládání jejich domovů a budov.

Dnes je KNX považován za jeden z nejrozšířenějších a nejúspěšnějších standardů pro domácí automatizaci a řízení budov na celém světě. Jeho spolehlivost, kompatibilita a flexibilita jej činí preferovanou volbou pro profesionály v oblasti automatizace a inženýrství budov. [3, 4, 1, 5]

2.2 Vrstvy KNX

Podobně jako model OSI (Open Systems Interconnection) je i KNX navržen ve vrstvách. Tyto vrstvy jsou hierarchickým modelem a každá z nich hraje při komunikaci určitou roli. Oproti OSI KNX nevyužívá všech 7 vrstev, ale je jich využito pouze 5, viz. tabulka 1. Dále platí, že každé přenosové médium má svoji linkovou vrstvu, tzn. poslední dvě vrstvy bývají specifické. [6, 7]

Tabulka 1: Vrstvy OSI a KNX. Zdroj: [Autor]

	Vrstva	OSI	KNX
7	Aplikační	✓	✓
6	Prezentační	✓	✘
5	Relační	✓	✘
4	Transportní	✓	✓
3	Síťová	✓	✓
2	Linková	✓	✓
1	Fyzická	✓	✓

2.2.1 Komunikace mezi vrstvami

Vrstvy KNX jsou mezi sebou provázány a každá vrstva má své tzv. *služby*, které jsou používány jako rozhraní pro komunikaci mezi nimi. Tyto služby jsou také označovány jako SDU (Service Data Units). Data přenášena mezi vrstvami se nazývají PDU (Protocol Data Units) a obsahují

pro každou vrstvu specifické informace označované jako PCI (Protocol Control Information). Základní primitiva při komunikaci mezi vrstvami jsou zmíněna v seznamu níže. Služby nemusí nutně využít všechny ze zmíněných primitiv.

- *request* - označováno jako *req*, vyjadřující požadavek
- *indication* - označováno jako *ind*, vyjadřující indikaci
- *confirmation* - označováno jako *con*, vyjadřující potvrzení
- *response* - označováno jako *res*, vyjadřující odpověď

Dále jde tyto služby dělit do třech kategorií. První kategorií jsou *lokálně potvrzené služby*. Tato kategorie vyžaduje *request*, *indication* a *confirmation*. Komunikace v tomto případě probíhá tak, že jsou PDU vygenerovány lokálně (v aktuální vrstvě *n*) a jsou postupně přenášeny mezi vrstvami, dokud nedojde k jejich přenosu přes fyzickou vrstvu. Na straně příjemce je indikací aktivována protější vrstva *n* a zaslaná PDU jsou dekódována a následně zasílána nižšími vrstvami příjemce, dokud nedorazí k již zmíněné příjemcově vrstvě *n*. Následně je odesílateli zasláno potvrzení (*confirmation*) z lokální nižší vrstvy *n-1*.

Druhou kategorií těchto služeb jsou *potvrzené služby*. Požadavky těchto služeb se nemění. Je tedy potřeba *request*, *indication* a *confirmation*. Komunikace v případě této kategorie funguje podobně jako v případě první kategorie s tím rozdílem, že potvrzení nepřichází z nižší vrstvy *n-1*, ale v odpovídající příjemcově vrstvě je vygenerováno potvrzení, které je zasláno odesílateli.

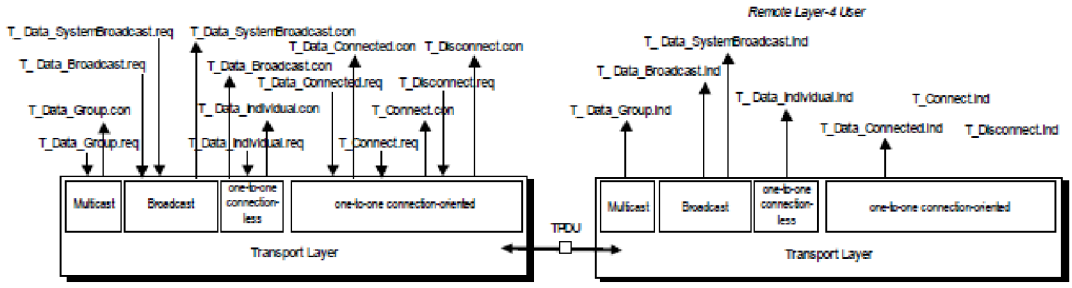
Poslední kategorií jsou *služby s odpovědí*. Tato kategorie vyžaduje všechny komunikační primitiva, tzn. *request*, *indication*, *confirmation* a *response*. Počátek komunikace je v tomto případě podobný jako u předchozích dvou kategorií. Odlišný je opět způsob zaslání potvrzení. V tomto případě totiž vzdálená vrstva *n* vygeneruje odpověď (*response*), která je zaslána odesílateli, kde je odpovídající vrstva aktivována indikací a dále zpracována odpověď, která je brána jako potvrzení. [2]

2.2.2 Aplikační vrstva

Aplikační vrstva představuje nejvyšší úroveň komunikace v KNX. Definuje standardizované aplikační služby a funkce, které umožňují ovládání a monitorování KNX zařízení. Aplikační vrstva zahrnuje například ovládání osvětlení, topení, žaluzií a dalších funkcí domácí automatizace. [8]

2.2.3 Transportní vrstva

Transportní vrstva je zodpovědná za spolehlivý přenos dat mezi odesílatelem a příjemcem. Zajišťuje, že data jsou doručena bez chyb a v pořadí, v jakém byla odeslána. Tzn. při odesílání dat se stará o to, aby data byla rozdělena do jednotlivých paketů a správně odeslána. V případě přijímání dat se zodpovídá za to, aby byly přijaté pakety správně srovnány a data v nich mohla být opět složena.



Obrázek 1: Transportní vrstva KNX. Zdroj: [9]

V obrázku 1 je možné vidět dělení komunikačních módů, které KNX podporuje. Tato vrstva totiž umožňuje zjednodušit komunikaci tím, že dovoluje komunikaci mezi dvěma uzly sítě tak, jako by šlo o přímé spojení (End to end) a díky tomu není nutné brát v potaz skutečnou topologii sítě. Pro tuto komunikaci KNX podporuje několik módů, viz seznam níže:

1. Point-to-Multipoint, Connectionless (Multicast)
2. Point-to-Domain Connectionless (Broadcast)
3. Point-to-all-Points Connectionless (System Broadcast)
4. Point-to-Point Connectionless
5. Point-to-Point Connection-Oriented

Prvním módem je Point-to-Multipoint, který slouží ke komunikaci mezi skupinami zařízení, které jsou označeny svojí skupinovou adresou, tzn. jedná se o multicast. Více o skupinách zařízení a skupinových adresách bude zmíněno v nadcházejících sekcích práce, viz. sekce 2.3.2. Druhým komunikačním módem je Point-to-Domain Connectionless. Jedná se o mód, kdy jedno zařízení vysílá a zbytek zařízení přijímá, tzn. jedná se o broadcast. Třetím módem je Point-to-all-Points Connectionless,

který je alternativně nazýván system brodcast. Tento mód podobně jako klasický brodcast umožňuje komunikaci jednoho zařízení k vícero zařízením, nicméně specifikace uvádí, že cíleni jsou komunikační partneři. Čtvrtým podporovaným módem je Point-to-Point Connectionless, který umožňuje komunikaci mezi dvěma zařízeními. Tento mód nevyžaduje žádné potvrzení příjmu dat, či udržování spojení. Posledním módem je Point-to-Point Connection-Oriented, který též slouží ke komunikaci mezi dvěma zařízeními, nicméně je nutné udržovat spojení, dokud je třeba. [9]

2.2.4 Síťová vrstva

Síťová vrstva se zabývá adresací a směrováním dat v rámci rozsáhlejších KNX sítí. Zajišťuje, aby data byla správně směrována mezi různými linkovými segmenty. Síťová vrstva také podporuje konfiguraci adresování a správu sítě. KNX používá ke směrování podobně jako IP sítě tzv. směrovače, které jsou zodpovědné za směrování v podsítích. [10]

2.2.5 Linková vrstva

Linková vrstva se stará o řízení přenosu dat mezi zařízeními v rámci KNX sítě. Zajišťuje, aby data byla správně adresována, řízena a přijímána. Linková vrstva také obsahuje mechanismy pro detekci a opravu chyb v přenosu dat. K tomu je využíváno LLC (Logical Link Control) a MAC (Medium Access Control).

LLC je zodpovědné za kontrolu chyb. Dále také převádí tzv. *datagram* na tzv. *data frame*. Datagram je základní jednotka, která může být zaslána příjemci, nebo příjemcům, nicméně neobsahuje informace o adrese, nebo adresách. Z tohoto důvodu je datagram přetvořen na data frame, který tyto informace obsahuje.

Vzhledem k tomu, že je KNX sběrnicový systém, je potřeba řešit detekce kolizí při komunikaci. Samotná zařízení by měla komunikační médium monitorovat a kolizím předcházet. Nicméně ne vždy se to podaří a proto je potřeba implementovat MAC. MAC je v případě KNX implementováno pomocí CSMA/CA (Carrier Sense Multiple Access with / Collision Avoidance). Tato metoda v praxi pracuje tak, že zařízení při odesílání dat zároveň naslouchá na přenosovém médiu a pokud se slyšená data liší od dat odeslaných, víme, že došlo ke kolizi. Tzn. pokud zařízení A vyšle logickou 1 a zařízení B logickou 0, zařízení A diagnostikuje kolizi a s odesláním dat setrvá, čímž upřednostní komunikaci zařízení B. [11]

2.2.6 Fyzická vrstva

Fyzická vrstva představuje nejnižší úroveň komunikace v systému KNX. Zahrnuje fyzické médium jako je kabeláž (např. kroucená dvojlinka a rozvodová síť budov) nebo bezdrátový přenos. Fyzická vrstva definuje elektrické a mechanické parametry pro přenos signálu mezi zařízeními. [12]

2.3 Adresovaní v KNX

KNX podporuje dva druhy adresování. Díky tomu lze adresovat individuální zařízení, jejich specifické funkcionality, nebo lze tyto funkcionality seskupit a adresovat tyto skupiny. [13]

2.3.1 Individuální adresy

Při navrhování instalace KNX je nutné každému ze zařízení přidělit jeho individuální adresu. Je nutné, aby tato adresa byla unikátní napříč celou instalaci KNX.

Individuální adresa se skládá ze dvou oktetů. První oktet je rozdělen na dvě části. Jeho první 4 bity reprezentují oblast, ve které je zařízení umístěno. Druhé 4 bity reprezentují číslo linie v oblasti, kde je zařízení umístěno. Poslední oktet (bajt) reprezentuje číslo zařízení na linii. Jako separátor v případě individuální adresy je používána tečka.

Tzn. počet oblastí se pohybuje v rozsahu (0-15), stejně tak jako počet linií. A počet zařízení na linii se pohybuje v rozsahu (0, 255). Takže individuální adresa může vypadat například takto: 1.1.10

Rozdělení adresy na oblasti a linie bude podrobněji objasněno v sekci 2.6. [13]

2.3.2 Skupinové adresy

Skupinová adresa může být přiřazena více než jednomu zařízení, respektive jejich (skupinovým) komunikačním objektům. Komunikační objekty jsou přiblíženy v sekci 2.4. Tato adresa není vázána na fyzické umístění zařízení, tudíž zařízení můžou být seskupena napříč oblastmi a liniemi.

- Podobně jako individuální adresa i tato adresa může být tvořena trojicí čísel. První část má rozsah (0-31), druhá (0-7) a poslední (0-255).
- Druhá reprezentace je tvořena dvojicí čísel. První je v rozsahu (0--2047) a reprezentuje hlavní funkci podskupiny. Druhé je v rozsahu (1-31) a určuje oblast úkolů skupiny.

V případě skupinové adresy je jako separátor použito dopředné lomítko. Tzn. lze jednoznačně určit, že adresa 1.1.10 je adresa individuální a adresa 1/1/50 je adresa skupinová. [13]

2.4 Komunikační objekty a jejich typy

Komunikační objekty v KNX reprezentují specifické funkcionality zařízení, respektive jejich vstupy, výstupy a funkce, které jsou těmito prvky podporované. Příkladem může být zařízení, jehož úkolem je ovládat osvětlení v místnosti. Tzn. zařízení musí mít výstup, který toto osvětlení bude rozsvěcet, zhasínat, stmívat atd. Každá z těchto funkcionalit musí mít svůj komunikační objekt a případně i komunikační objekt reprezentující aktuální stav výstupu pro zpětné doptávání se na aktuální stav.

Tabulka 2: Ukázka komunikačním objektů a jejich funkcí. Zdroj: [Autor]

Komunikační objekt	Číslo objektu	Typ data pointu	Název typu
outputOnOff	1	1.001	Switch
outputScale	2	5.001	Percentage 0, 100%
outputOnOffStatus	3	1.001	Switch
outputScaleStatus	4	5.001	Percentage 0, 100%

V tabulce 2 jsou uvedeny komunikační objekty pro tento příklad. Velmi důležitou částí tabulky je sloupec *Číslo objektu.* Toto číslo reprezentuje daný komunikační objekt zařízení a jeho název je čistě orientační.

2.4.1 Příznaky komunikačních objektů

Každý komunikační objekt má šest příznaků viz. seznam níže.

První příznak reprezentuje, zdali je komunikační objekt přenášen po sběrnici, či nikoli.

Příznak *READ* značí, jestli může být tento komunikační objekt čten ze sběrnice. Pokud bude možné dotázat se na aktuální stav hodnoty, bude tento příznak nastavený na *true*.

Následující příznak (*WRITE*) podobný příznaku předchozímu s tím rozdílem, že slouží pro zápis. Tzn. pokud bude tento příznak nastavený na *true*, bude možné pomocí tohoto komunikačního objektu zapisovat novou hodnotu.

Příznak *TRANSMIT* udává, zdali zařízení aktivně tento komunikační

objekt vyšle na sběrnici. Tzn. pokud půjde například o komunikační objekt reprezentující stisk tlačítka, musí mít tento příznak nastavený na *true*, aby mohlo zařízení aktivně informovat o stisku tlačítka.

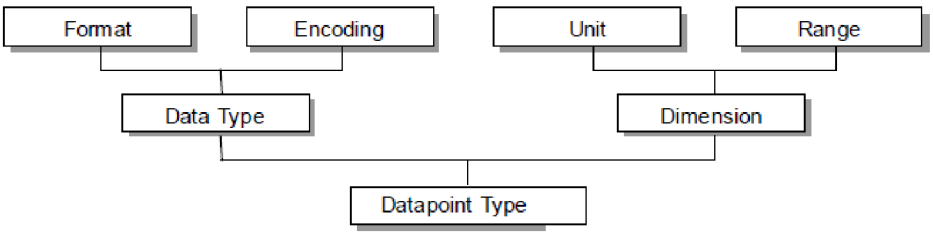
Příznak aktualizace (*UPDATE*) udává, že příznak čtení z jiného komunikačního objektu zapíše přečtenou hodnotu do komunikačního objektu, který je *čtenářem*, tzn. dojde k aktualizaci.

Poslední příznak udává, zdali má být hodnota komunikačního objektu inicializována po spuštění zařízení. Tzn. například při spuštění se zařízení dotáže na aktuální stav. [14, 6]

1. Komunikace (Communication)
2. Čtení (READ)
3. Zápis (WRITE)
4. Vysílání (TRANSMIT)
5. Aktualizace (UPDATE)
6. Čtení při inicializaci (READ ON INIT)

2.4.2 Typy komunikačních objektů

Každý z komunikačních objektů musí mít i svůj typ, respektive typ jeho hodnoty. Jedná se o normalizovaná data. Každý datový typ je vytvořen dle stromové struktury, viz. obr. 2.



Obrázek 2: Stromová struktura dělení data pointů. Zdroj: [15]

Díky normalizaci je zaručena kompatibilita mezi zařízeními. Tyto typy jsou řazeny do skupin a tyto skupiny jsou označovány jako *{číslo skupiny}.{číslo podskupiny}*. Důležité je také podotknout, že číslo skupiny udává i velikost daného data pointu, zatím co číslo podskupiny udává pouze způsob reprezentace dat.

Ukázka typů data pointů viz. seznam níže: [15]

- 1.xxx 1-bit
  - 1.001 switch, 1.002 boolean, 1.003 enable
  - atd.
- 2.xxx 1-bit controlled
  - 2.001 switch control, 2.002 boolean control, 2.003 enable control
  - atd.
- atd.
- 5.xxx 8-bit unsigned value
  - 5.001 percentage (0..100%), 5.003 angle (degrees), 5.004 percentage (0..255%)
  - atd.
- atd.

2.5 Přenosová média KNX

KNX podporuje řadu komunikačních médií pro přenos dat mezi zařízeními. Tyto média je možné vzájemně kombinovat. Tato vlastnost značně přispívá k flexibilitě tohoto standardu a umožňuje jednoduché budování KNX systémů. [3]

2.5.1 KNX TP

KNX TP využívá ke komunikaci kroucenou dvojlinku a je nejčastěji používaným médiem pro KNX. Toto médium využívá ke komunikaci a k napájení zařízení na sběrnici 30 V_{DC}. Alternativně může kabel kroucené dvojlinky obsahovat pár pro napájení silové části zařízení. Napětí KNX TP je označováno za SELV (Safety Extra Low Voltage), tzn. jedná se o bezpečné napětí a tato vlastnost usnadňuje práci s tímto médiem. Pro toto médium existují dva standardy. Prvním je TP-64, který umožňuje zapojit maximálně 64 zařízení na jeden fyzický segment. Druhým je TP-256, který umožňuje zapojit až 256 zařízení na jeden fyzický segment. TP-64 existuje z historických důvodů. TP-64 a TP-256 jsou plně kompatibilní. [13, 1]

Toto médium bude hlavním předmětem této práce a bude přiblíženo v nadcházející části práce.

2.5.2 KNX PL

KNX PL (KNX Power Line) využívá jako médium existující rozvodovou síť budov. Proto je toto médium vhodné pro starší budovy, které nemají rozvod KNX TP. Signály KNX jsou modulovány na standardní napětí 230 nebo 400 V_{AC} a je možné využít všechny 3 fáze. Oproti KNX TP práce s tímto médiem již není tak snadná a při neopatrnosti může být i nebezpečná. Dále je nutné dodržovat normy a bezpečnostní pravidla země, ve které je instalace prováděna. Existují dva druhy KNX PL, nicméně nejznámější a nejpoužívanější je tzv. *PL110*. [13]

2.5.3 KNX RF

KNX RF (KNX Radio Frequency) využívá rádiové vlny pro bezdrátovou komunikaci mezi KNX zařízeními. Tato technologie je vhodná pro situace, kdy není možné nebo praktické použít kabeláž. Opět existují dvě implementace KNX RF, první je RF Ready a druhá je RF Multi. Frekvence používaná RF Ready je 868,3 MHz a typ modulace je frekvenční. Vzhledem k tomu, že RF Ready používá jen jednu frekvenci, může docházet ke kolizím s ostatními RF systémy (ať už KNX, nebo s jinými) a proto byl vytvořen RF Multi. RF Multi využívá celkem 5 kanálů, z toho jsou 3 rychlé a 2 pomalé. Spolehlivost RF Multi je mnohem vyšší, díky možnosti střídání kanálů. RF Ready a RF Multi jsou spolu kompatibilní, vzhledem k tomu, že RF Multi může používat kmitočet RF Ready. [16]

2.5.4 KNX IP

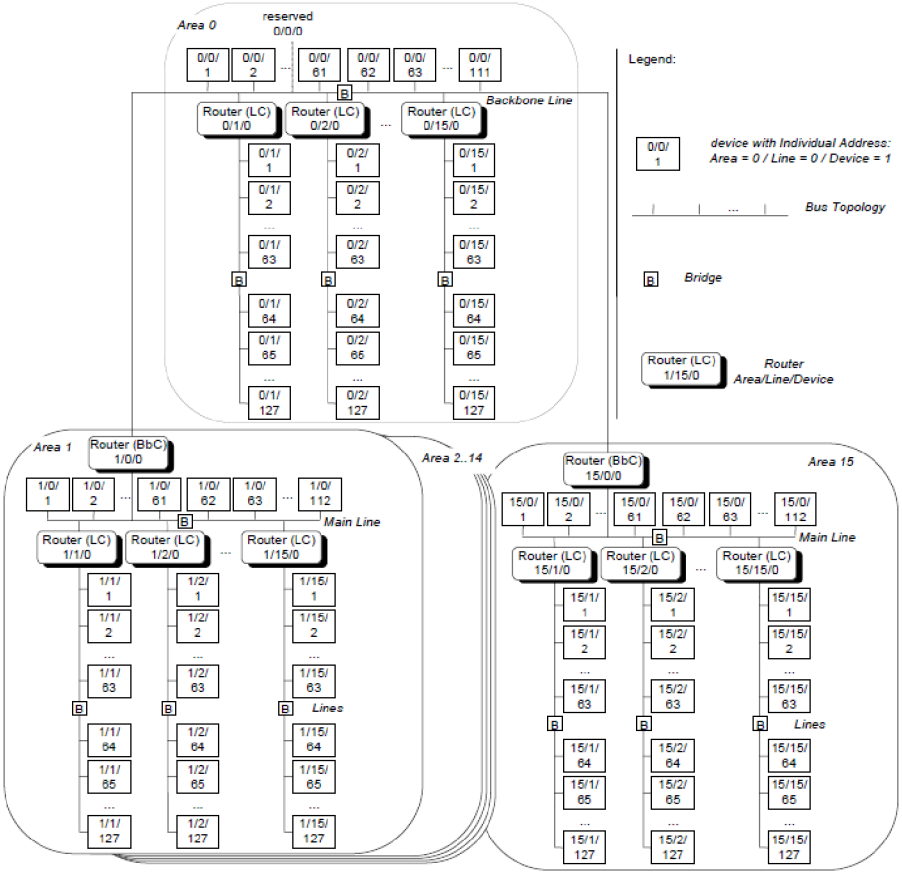
KNX IP (KNX Internet Protocol) (občas také označovaný jako KNXnet) využívá standardní IP (Internet Protocol) síť pro přenos dat mezi KNX zařízeními. Tímto způsobem je možné integrovat KNX do existujících internetových sítí a využívat výhod moderních IP technologií. Je podporován jak Ethernet, tak i Wi-Fi (Wireless Fidelity). Komunikace následovně probíhá pomocí UDP protokolu a samotné telegramy jsou obaleny standardním IP rámcem. KNX IP může být použito i jako páteřní linie KNX systému, což umožňuje propojení na velké vzdálenosti a tím dokáže zjednodušit jinak složité rozlehlé sítě. Díky tomuto médiu se KNX prosazuje v IoT. [17]

2.6 Topologie KNX

Vzhledem k tomu, že je KNX distribuovaná síť, má také svoji topologii. Na tuto topologii lze nahlížet z více úhlů. Prvním pohledem je logické uspořádání, které na síť nahlíží z pohledu adresování. Druhým pohledem na tuto problematiku je fyzické uspořádání. [3, 13]

2.6.1 Logická topologie KNX

Jak již bylo zmíněno, adresy v KNX jsou 16 bitové (dva oktety). Tato vlastnost KNX umožňuje připojit až 65’536 zařízení. Nicméně jak již bylo zmíněno v sekci 2.3, musí být tato zařízení rozdělena do oblastí a linií, viz. obr. 3. Jak vyplývá z velikostí jednotlivých částí individuální adresy, oblastí může být maximálně patnáct a linií taktéž. Co se týče počtu zařízení na lince, závisí jejich počet na přenosovém médiu. Pro KNX TP platí, že starší verze KNX TP-64 může mít na jedné linii maximálně 64 zařízení, potom je nutné použít tzv. router. Pro KNX TP-256 je možné připojit 256 zařízení přímo. [3, 13]

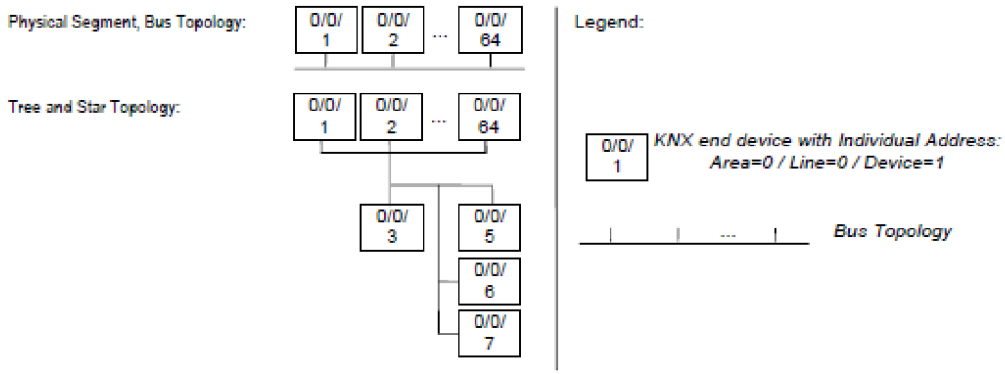


Obrázek 3: Logická topologie KNX. Zdroj: [13]

2.6.2 Fyzická topologie KNX TP

Fyzická topologie KNX může být naprosto rozdílná od té logické. Obecně opět záleží na přenosovém médiu, které je pro komunikaci použito. Pro KNX TP například platí, že zařízení na sběrnici mohou být zapojena lineárně, ve hvězdě, ve stromu, nebo kombinací všech těchto možností.

V obrázku 4 jsou zobrazeny příklady možných zapojení. V horní části obrázku se nachází lineární zapojení. Pokud síť nevyžaduje žádný směrovač, je každé nadcházející zapojení bráno v podstatě tímto způsobem. V dolní části obrázku je možné vidět spojení topologie hvězda a strom.

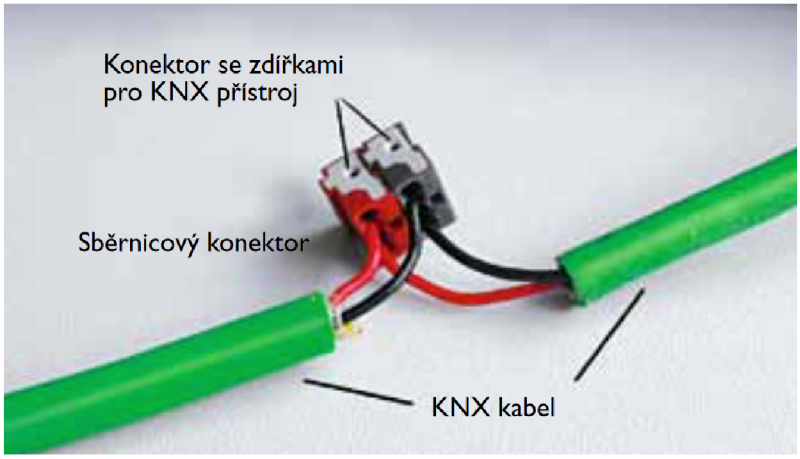


Obrázek 4: Topologie KNX TP. Zdroj: [13]

Vzhledem k tomu, že jde o sběrnicový systém, jsou dokonce povoleny smyčky, nicméně to není doporučeno. Rizikové to může být z několika důvodů. V případě dlouhé smyčky může dojít ke zpoždění signálů způsobené fyzikálními vlastnostmi přenosového média, což může signály vzájemně rušit. Dále může dojit například k větvení mezi nepovolenými částmi sítě. Tím se rozumí, že smyčka nesmí vzniknout například mezi liniemi, oblastmi, viz. sekce 2.6.2. [3, 13]

3 KNX TP

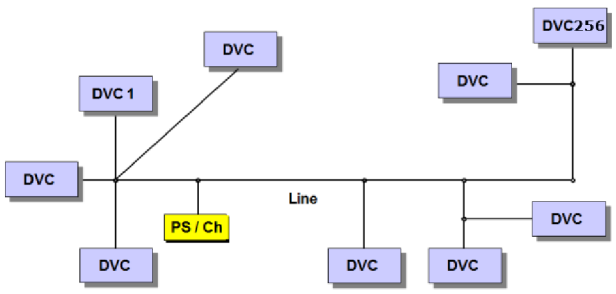
Jak již bylo zmíněno, KNX TP je jedním z přenosových médií KNX. Toto médium využívá ke komunikaci kroucenou dvojlinku s označením YCYM nebo JY(ST)Y (2,5 kV). Pro spojení kabeláže se zařízením KNX TP využívá konektor se zdířkami pro zařízení KNX, viz. obr. 5. Díky tomu, že konektor obsahuje i více zdířek pro kabeláž KNX, je větvení sítě poměrně jednoduché a umožňuje odpojení či připojení zařízení bez přerušení sběrnice. [13]



Obrázek 5: Ukázka kabeláže KNX TP. Zdroj: [6]

3.1 Zařízení na KNX TP

Jak již bylo zmíněno, KNX TP je sběrnice, tím pádem můžeme zařízení na sběrnici připojit, viz. sekce 2.6.2. Maximální délka fyzického segmentu je 1000 m s tím, že maximální vzdálenost zařízení je 700 m. Maximální vzdálenost zařízení od zdroje je 350 m. [13]



;

Obrázek 6: Ukázka kabeláže KNX TP. Zdroj: [6]

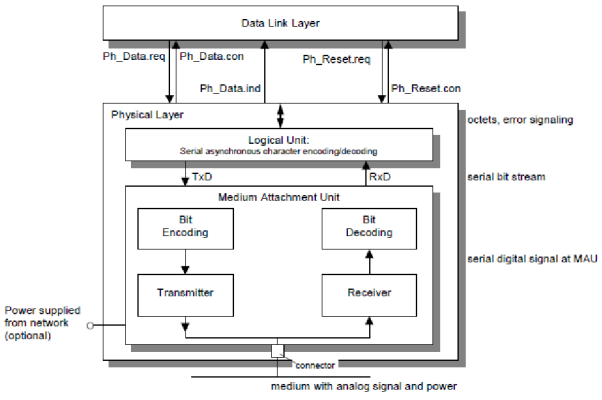
Vzhledem k tomu, jakým způsobem probíhá komunikace na KNX TP je nutné, aby každá instalace měla svůj zdroj, *PS / Ch*, viz. obr. 6. Tento zdroj má specifické vlastnosti. Jeho napětí musí být cca 30 V_{DC} (specifikace udává rozpětí od 21 do 32 V_{DC}). Dále musí být zdroj vybaven tzv. *choke*, tím je myšlena tlumivka, která dělá tento zdroj měkký. Z tohoto důvodu je v obrázku 6 popsán právě jako PS (Power Supply) / Ch (choke) a bloky DVC značí zařízení.

Co se týče proudové charakteristiky zařízení, pohybuje se od 3 mA do 12 mA. Pokud není součástí zařízení energeticky náročný prvek, celé zařízení může být napájeno ze sběrnice. Alternativně může být silová část zařízení napájena separátně. Tzn. kroucená dvojlinka může být v případě KNX TP použita jak ke komunikaci, tak k napájení. Aby toto bylo možné, musí být zařízení vybaveno kondenzátorem, který plní funkci zásobníku energie po dobu komunikace.

Všechna standardní zařízení na KNX TP jsou vybavena také tlačítkem a kontrolkou. Tlačítko umožňuje přepnout zařízení do programovacího režimu a kontrolka značí, že je tento režim spuštěn. Alternativně je možné kontrolku rozblikat pomocí softwaru pro tvorbu aplikací, viz. sekce 4.3.1.

3.1.1 KNX TP Medium Attachment Unit

MAU (Medium Attachment Unit) musí být součástí každého zařízení na KNX TP. Jeho hlavním úkolem je překládat znaková data tak jak je známe z komunikace pomocí UART (Universal Asynchronous Receiver-Transmitter) na analogový proud dat typický pro KNX TP. S MAU komunikuje přímo linková vrstva KNX, tzn. MAU spojuje linkovou vrstvu a fyzické médium (kroucenou dvojlinkou), viz. obr. 7. Jeho vedlejší funkcí je napájení samotného zařízení. [13]



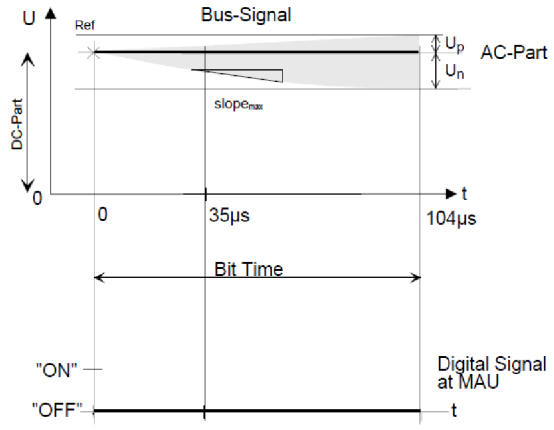
Obrázek 7: Spojení linkové vrstvy KNX TP, MAU a fyzického média. Zdroj: [6]

3.2 Komunikace na KNX TP

Jak již bylo zmíněno, komunikace na KNX TP probíhá prostřednictvím MAU. Rychlost komunikace je 9600 bitů za sekundu, tzn. přenesení jednoho bitu trvá 104 μs. [13]

3.2.1 Logická 1 na sběrnici KNX TP

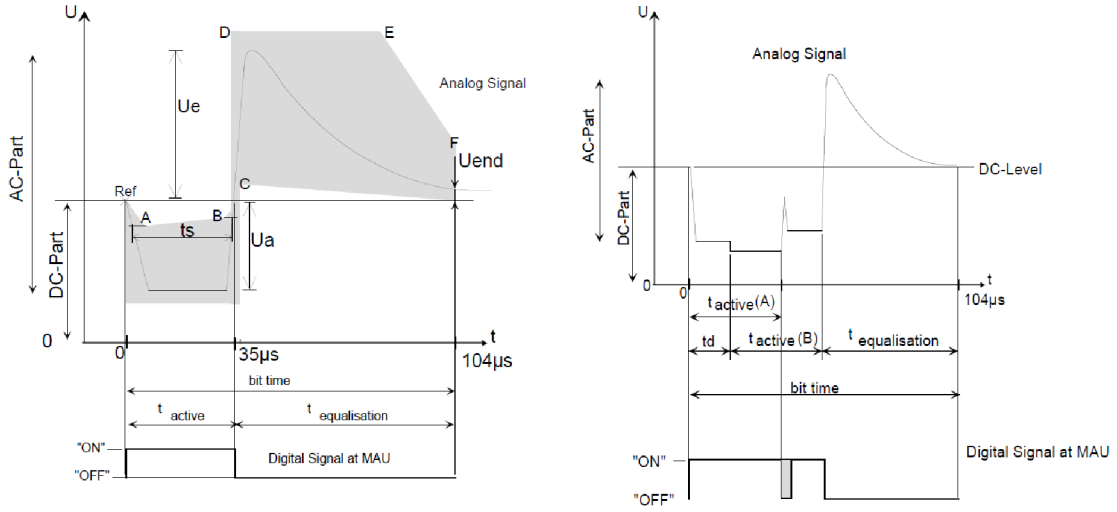
Pokud chce zařízení na sběrnici poslat logickou jedničku, je to podobná situace jako když zařízení nevysílá nic. Viz. ***t*** v obr. 8. Maximální sklon napětí je 400 $\frac{mV}{ms}$ (slope_{max}) a odchylka napětí musí být v rozsahu +0.3 V až -2 V. [13]



Obrázek 8: Reprezentace logické 1 na sběrnici KNX TP. Zdroj: [13]

3.2.2 Logická 0 na sběrnici KNX TP

Situace při zaslání logické 0 je poněkud složitější. Logická nula je reprezentována poklesem napětí (maximálně na hodnotu 21 V) na sběrnici po dobu 35 μs, viz. levá část obr. 9. Respektive napětí musí být sraženo po dobu 25 μs a zbylých 10 μs je vyhrazeno pro sestupnou a vzestupnou hranu signálu. Ve zbylých 70 μs dojde k rapidnímu stoupání napětí na sběrnici (až o 13 V). Tento efekt je působen induktivními vlastnostmi tlumivky zdroje a je využit k opětnému dobití kondenzátoru na zařízení, který napájel zařízení při komunikaci. [13]

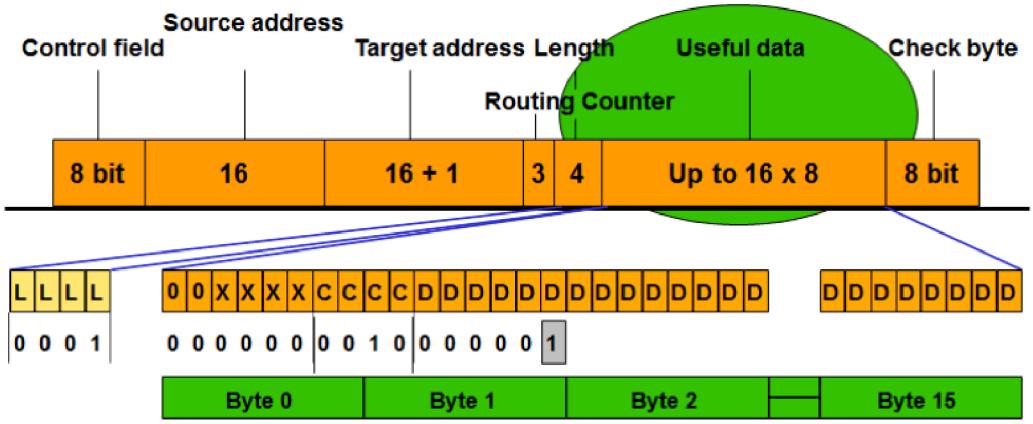


Obrázek 9: Reprezentace logické 0 na sběrnici KNX TP. Zdroj: [13]

Pokud jsou na sběrnici zařízení daleko od sebe, může dojít ke zpoždění signálu. V důsledku zpoždění signálu na přenosovém médiu (kroucené dvojlince) může dojít k tomu, že dvě zařízení začnou vysílat přibližně ve stejný čas (například s rozdílem 10 μs). Tato situace je zobrazena v pravé části obrázku 9. Pokud se zaměříme na digitální signál MAU, vidíme šedě vyznačenou oblast, která reprezentuje vzniklou chybu. [13]

3.2.3 KNX TP Telegram

Telegram by se dal přirovnat k datové jednotce na transportní vrstvě ISO/OSI modelu. Na obrázku 10 je zobrazena struktura KNX TP telegramu.



Obrázek 10: KNX TP telegram. Zdroj: [6]

První bajt telegramu obsahuje tzv. *control field*. Control field udává několik podstatných údajů. Prvním je typ rámce, respektive jestli jde o standardní rámec, či prodloužený. Je nutno podotknout, že ne všechna zařízení podporují prodloužené rámce. Následuje tzv. *repeat flag*, neboli příznak opakování. Tento údaj určuje, zdali se jedná o první zaslání telegramu, nebo opakované. Posledním důležitým údajem je priorita. Priority můžou být následující: systémová, urgentní, normální a nízká.

Druhé dva bajty telegramu obsahuje individuální adresu odesílatele. Dále následují další dva bajty, ke kterým je přidán jeden bit. Tato část telegramu reprezentuje adresu příjemce a přidaný bit rozlišuje, jestli se jedná o adresu individuální, nebo o adresu skupinovou.

Následují tři bity reprezentující tzv. *routing counter*. Tato hodnota je podobná TTL (Time To Live) hodnotě u klasického OSI modelu.

Další 4 bity reprezentují velikost dat telegramu. Tato hodnota udává počet znaků (bajtů) a je v rozsahu 1 až 16.

Následujících *n* bajtů, kde *n* odpovídá velikosti dat telegramu, reprezentuje samotná data telegramu.

Poslední bajt telegramu je kontrolní bajt, který udává paritu dat. [6, 13]

3.3 Data telegramu KNX TP

Jak bylo popsáno v sekci 3.2.3, telegram obsahuje data (*Useful data* viz. obr. 10). Nejčastěji přenášenými daty jsou komunikační objekty. Jak již bylo zmíněno v sekci 2.4, tyto komunikační objekty mají vždy definovaný standardizovaný formát pro reprezentaci jejich dat, neboli svůj *data point*. [6, 15]

3.3.1 Reprezentace data pointu switch

První ukázkou jakým způsobem jsou data v telegramu formátována je data point s názvem *Switch* a číselným označením *1.001*. V tabulce 3 je zobrazena jeho struktura. Data tohoto telegramu mají délku dvou bajtů. První dva bity musí být nastaveny na 0. Bity označené písmenem X nejsou vyhodnocovány. Bity označené písmenem C značí příkaz. V případě zobrazeném v tabulce se jedná o příkaz zápisu. Poslední bit označený písmenem A značí hodnotu. Reprezentace hodnoty je zřejmá, 0 značí vypnuto, 1 značí zapnuto.

Tabulka 3: Struktura data pointu switch. Zdroj: [Autor]

Bit č.	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Význam	**0**	**0**	**X**	**X**	**X**	**X**	**C**	**C**	**C**	**C**	**X**	**X**	**X**	**X**	**X**	**A**
Hodnota	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

Pro ovládání žaluzií a jiných obdobných akčních členů jsou použity data pointy 1.008 pro operaci nahoru, nebo dolů a 1.007 pro krokování. Dle notace zmíněné v sekci 2.4 lze poznat, že patří do stejné skupiny data pointů. Tzn. jejich struktura je stejná, nicméně význam jejich hodnoty se liší.

3.3.2 Reprezentace data pointu switch control

Data point s názvem switch control a číselným označením 2.001 slouží k ovládání akčních členů společně s data pointem switch. Jeho struktura je zobrazena v tabulce 4. Bity příkazu (C) zůstávají stejné jako u předchozího příkladu, tzn. stále jde o příkaz zápisu. Nicméně hodnota má nyní dva bity (bity jsou označeny písmenem A).

Tabulka 4: Struktura data pointu switch control. Zdroj: [Autor]

Bit č.	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Význam	**0**	**0**	**X**	**X**	**X**	**X**	**C**	**C**	**C**	**C**	**X**	**X**	**X**	**X**	**A**	**A**
Hodnota	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0

Význam hodnoty je popsán v seznamu níže. Tento data point může být použit například u akčních členů, jejichž spínání je z důvodu bezpečnosti například vybaveno pojistkou sepnutí. V tomto případě by tato pojistka hrála roli spínače s prioritou a pokud by nebyla sepnuta, tak by bylo sepnutí bez priority ignorováno.

- 00_{BIN} - Vypnutí klasickým komunikačním objektem se switch data pointem
- 01_{BIN} - Sepnutí klasickým komunikačním objektem se switch data pointem
- 10_{BIN} - Vypnutí s prioritou
- 11_{BIN} - Sepnutí s prioritou

[6, 15]

3.3.3 Reprezentace data pointu scaling

Dalším data pointem je tzv. *scaling* s označením 5.001. Tento data point udává absolutní hodnotu v rozsahu 0 % až 100 %. Tento rozsah je vyjádřen pomocí jednoho bajtu, tzn. 0 až 255 s tím, že 0 odpovídá 0 % a 255 odpovídá 100 %. V tabulce 5 je možné vidět skupinu osmi bitů s označením A, která reprezentuje právě tuto hodnotu. Zbylá struktura zůstává stejná jako u ostatních data pointů, nicméně celková délka dat odpovídá 3 bajtům. [6, 15]

Tabulka 5: Struktura data pointu scaling. Zdroj: [Autor]

Bit č.	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	➡
Význam	**0**	**0**	**X**	**X**	**X**	**X**	**C**	**C**	**C**	**C**	**X**	**X**	**X**	**X**	**X**	**X**	
Hodnota	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	

➡	8	7	6	5	4	3	2	1
	A	**A**	**A**	**A**	**A**	**A**	**A**	**A**
	1	0	0	1	1	0	0	1

3.3.4 Reprezentace data pointu float 16

Reprezentace 16 bitového floatu je popsána v tabulce 6 a odpovídá data pointům s označením 9.00x. První dva bajty struktury se opět nemění (bity s označením 0, X a C). Druhé dva bajty reprezentují samotný 16 bitový float. Bity s označením M reprezentují tzv. *mantisu* a bit s označením S její znaménko. Rozlišení mantisy je vždy 0.01. Bity s označením E reprezentují exponent o základu 2. Samotná hodnota je vypočtena pomocí vzorce $X_{float16} = (-1)^S * (0.01 * M) * 2^E$. Bit S udává znaménko. Specifikace dále uvádí, že hodnota $7FFF_{HEX}$ je vyhrazena pro značení chybné hodnoty a kompletní rozsah je od -671’088,64 do 670’433,28.

Tento data point je využíván například k reprezentaci teploty, vlhkosti a tlaku, kdy je přesnost 0.01 naprosto dostačující. Nicméně v KNX existuje i 32 bitový float pro přesnější hodnoty. Jeho reprezentace se řídí normou IEEE 754. [15]

Tabulka 6: Struktura data pointu dvou bajtový float. Zdroj: [Autor]

Bit č.	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17
Význam	**0**	**0**	**X**	**X**	**X**	**X**	**C**	**C**	**C**	**C**	**X**	**X**	**X**	**X**	**X**	**X**
Hodnota	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
S	**E**	**E**	**E**	**E**	**M**	**M**	**M**	**M**	**M**	**M**	**M**	**M**	**M**	**M**	**M**
1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1

3.3.5 Reprezentace data pointu RGB

Reprezentace RGB data pointu je popsána v tabulce 7. Tento data point je označován jako 232.600 a jedná se o 3 bajtové vyjádření RGB barvy. Tzn. prvních 8 bitů označených písmenem B reprezentuje modrou barvu. Dalších osm bitů označených písmenem G reprezentuje zelenou. Následujících 8 bitů reprezentuje červenou barvu. Zbylá pole mají stejný význam jako u předchozího data pointu.

Vzhledem k tomu, že jednotlivé barvy jsou reprezentovány jedním bajtem, jedná se tedy o 24 bitovou barvu. Nevýhoda tohoto data pointu je vlastnost, že není blíže určeno jakým způsobem má být s hodnotou zacházeno. Proto může například v situaci, kdy je barevné světlo stmíváno dojít k tomu, že stmíváním je posunuta hodnota jednoho z barevných kanálů k nule a tím je v podstatě změněna barva. Pokud tuto situaci jednotlivá zařízení řeší, nemusí tento problém řešit stejně a proto může opět dojít k odlišnému výsledku. Z tohoto důvodu existuje i 6 bajtový data point RGBW. [15]

Tabulka 7: Struktura data pointu RGB. Zdroj: [Autor]

Bit č.	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25
Význam	**0**	**0**	**X**	**X**	**X**	**X**	**C**	**C**	**C**	**C**	**X**	**X**	**X**	**X**	**X**	**X**
Hodnota	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9
R	**R**	**R**	**R**	**R**	**R**	**R**	**R**	**G**	**G**	**G**	**G**	**G**	**G**	**G**	**G**
1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1

8	7	6	5	4	3	2	1
B	**B**	**B**	**B**	**B**	**B**	**B**	**B**
1	0	0	1	1	0	0	1

4 ETS

ETS (Engineering Tool Software) je softwarový nástroj používaný ve spojení s KNX pro programování a konfiguraci zařízení v síti KNX. ETS poskytuje uživatelské rozhraní, prostřednictvím kterého mohou instalatéři, projektanti a technici vytvářet, editovat a spravovat konfigurace a programy pro zařízení v síti KNX. [18, 19, 1]

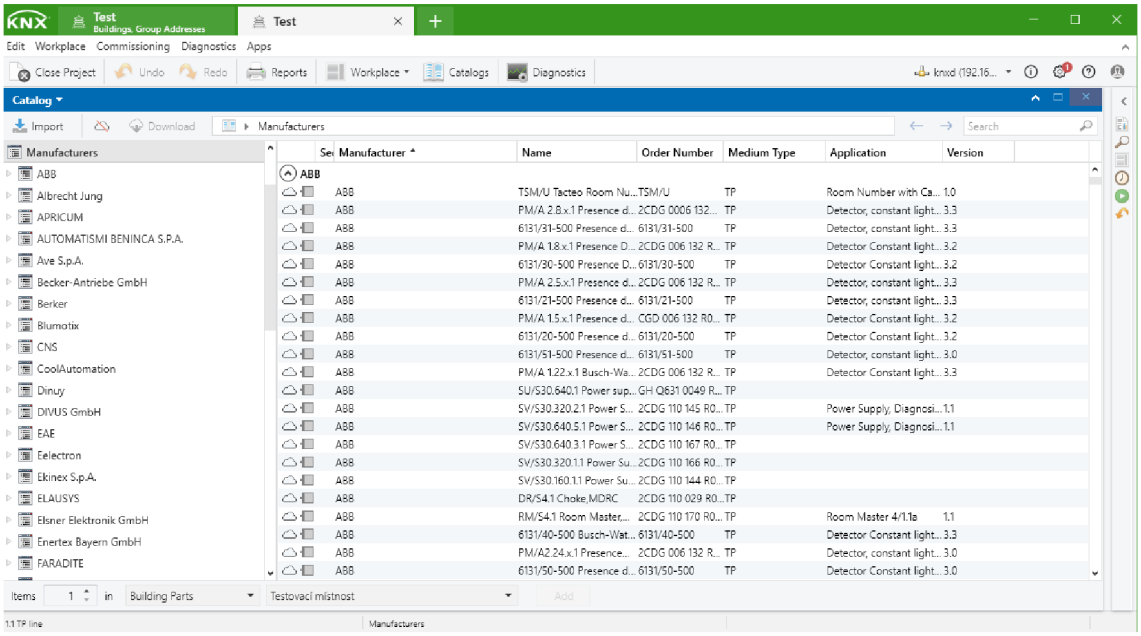
4.1 Reprezentace budovy

ETS umožňuje vytvořit realistickou reprezentaci budovy, nebo více budov. Tato reprezentace může využívat logické topologie, viz. sekce 2.6. Tzn. podlaží budovy mohou být reprezentována jako oblast a místnosti v jednotlivých podlažích jako linie.

4.2 Katalogy, zařízení a adresy

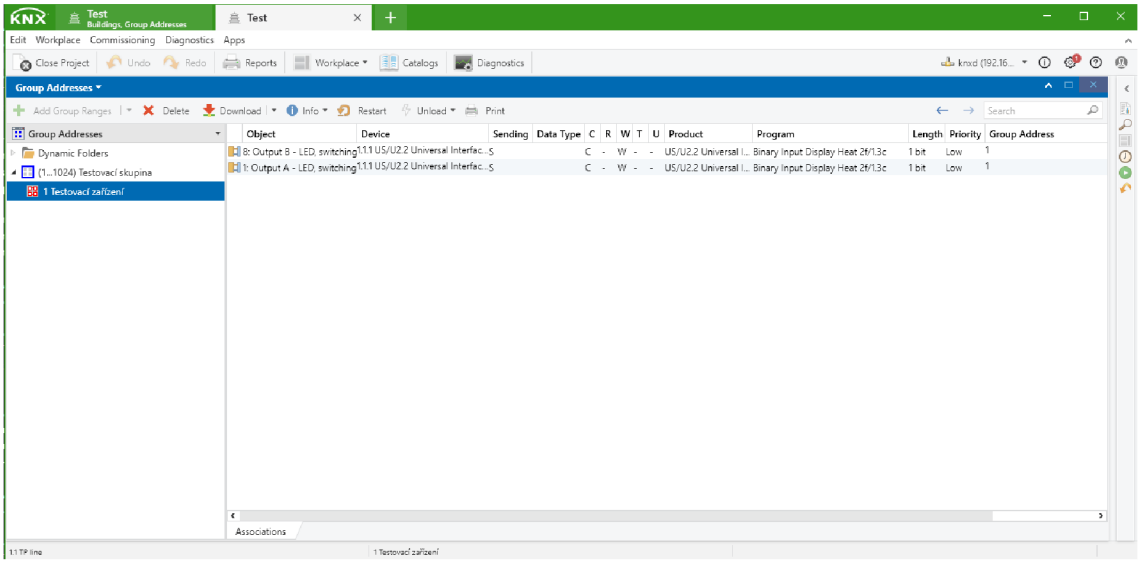
Do zhotovené reprezentace budovy lze přidat zařízení. K tomu v ETS slouží *katalogy*. Katalogy jsou hierarchicky uspořádané seznamy zařízení, které jsou řazené dle výrobců a jejich účelu, viz obr. 11. Příkladem může být zařízení od společnosti ABB s názvem *Universal Interfaces US/U 2.2*, které je zařazeno následovně:

Katalogy ➔ *ABB* ➔ *Universal interfaces* ➔ *Universal Interfaces US/U 2.2*.



Obrázek 11: ETS Katalogy. Zdroj: [Autor]

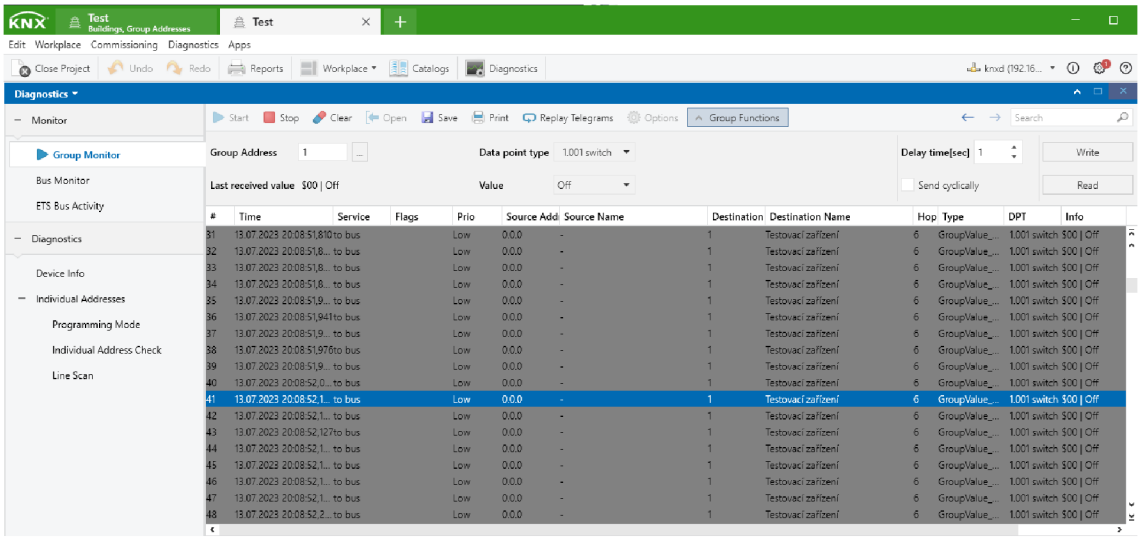
Po přidání zařízení ETS zařízení automaticky přidělí individuální adresu. Tuto adresu je možné změnit dle vybrané topologie. Poté je možné zařízení nastavit jeho parametry. Tyto parametry jsou pro každý druh zařízení odlišné a závisí na jeho funkci. Po nastavení parametrů ETS umožní seskupit komunikační objekty a tím zařízení propojit, viz. obr. 12. Poté už jen zbývá nahrát tuto konfiguraci do jednotlivých zařízení pomocí ETS a stisknutí programovacího tlačítka na zařízení. Po nahrání individuální adresy již není nutné stisknout programovací tlačítko, vzhledem k tomu, že poté je možné zařízení identifikovat pomocí adresy a ne stiskem tlačítka. [20]



Obrázek 12: ETS Ukázka skupin. Zdroj: [Autor]

4.3 Diagnostika

Při instalaci KNX či vývoji KNX zařízení je dobré mít přehled o dění na sběrnici. Z tohoto důvodu má ETS záložku s diagnostikou a monitoringem sběrnice, viz. obr. 13. [21, 22]



Obrázek 13: ETS Diagnostika. Zdroj: [Autor]

4.3.1 Diagnostika adres a zařízení na sběrnici

ETS umožňuje získat kompletní přístrojové informace o jednotlivých zařízení na sběrnici včetně komunikačních (skupinových) objektů podle skupinové adresy. Dále ETS umožňuje zkontrolovat dosažitelnost zařízení podle individuální adresy a případně rozblikat či rozsvítit signalizační světlo na zařízení. Tyto funkcionality umožňují jednoznačně identifikovat jednotlivá zařízení v obsáhlejších systémech KNX.

Dále ETS umožňuje naskenovat celou linii KNX. Tato funkcionalita je velmi užitečná k odhalení chyb ve fyzické instalaci KNX, nebo při mapování existující instalace KNX. Sken linie ETS zároveň umožňuje porovnat s projektem. [21]

4.3.2 Aktivita na sběrnici

Krom diagnostiky jednotlivých zařízení a skenu linie instalace KNX, ETS umožňuje i sledovat, nebo zaznamenat kompletní dění na sběrnici pomocí monitoru. Samotný monitor je rozdělen do tří částí, viz. seznam níže:

- Bus Monitor
- Group Monitor
- ETS Bus Activity Monitor

Bus monitor a group monitor jsou si velmi podobné, ovšem group monitor má více funkcionalit. Při záznamu dění na sběrnici bus monitor vyžaduje kompletní přístup ke sběrnici, to znamená, že není možné diagnostikovat jednotlivá zařízení, viz. sekce 4.3.1.

Group monitor oproti tomu nevyžaduje exkluzivní přístup ke sběrnici, tzn. může sledovat i přidělování skupinových adres, je možné přidělovat a kontrolovat skupinové adresy či skenovat celou linii sběrnice. Dále také umožňuje přepisovat hodnoty komunikačních objektů.

ETS bus activity monitor je používán spíše k diagnostice spojení ETS se sběrnicí. Tento monitor nenavazuje spojení s instalací, ale jen *naslouchá* existujícím spojením. Toto je užitečné pokud je ETS spojen se sběrnicí více způsoby (například přes KNX IP a KNX TP), protože lze jednoznačně určit jaké spojení je k čemu použito. [22]

5 KNXD

KNXD (KNX Deamon) je serverový software, který umožňuje přistupovat ke sběrnici KNX. Jeho hlavní myšlenka je, že by měl fungovat jako router či brána, která umí komunikovat pomocí všech KNX rozhraní.

Vzhledem k tomu, že jsou podporována veškerá rozhraní, lze KNXD propojit s ETS, tzn. KNXD umožňuje programovat a diagnostikovat zařízení na KNX a také nastavovat a propojovat jiné instance KNXD, nebo KNX IP.

Dále je nutno zmínit, že tento software podporuje pouze operační systém Linux. Ačkoli se tento fakt může zdát poměrně omezující, tak při aplikaci v reálném světě by tento program nejspíše běžel na linuxovém serveru, nebo v případě domácího užití na jednodeskovém počítači, kterým je například Raspberry Pi. [23, 24]

5.1 Fyzické rozhraní KNXD

Jak již bylo zmíněno, KNXD umožňuje propojit sběrnici KNX. Z tohoto důvodu podporuje několik fyzických rozhraní, viz. tab. 8.

Mezi nejpoužívanější fyzická rozhraní patří **ip** a jeho podvarianty **ipt** a **iptn**. Tato rozhraní slouží k připojení KNX IP, nebo k tunelování mezi dvěma instancemi KNXD. Možnost tunelování je opravdu výhodná v případě, že je KNX IP použitý jako páteřní linie KNX, vzhledem k tomu, že umožňuje zjednodušit (někdy dost komplexní) strukturu velkých KNX sítí.

Dále jsou velmi používaná rozhraní **tpuart** a **usb**. Rozhraní **tpuart** umožňuje připojit tzv. BCU (Bus Coupling Unit) pomocí UART. *Usb* rozhraní slouží k připojení specializovaných USB (Universal Serial Bus) zařízení pro komunikaci s KNX.

Rozhraní tpuart má svou tzv. TCP (Transmission Control Protocol) verzi, která je určena pro přenášení dat pomocí standardních tcp socketů. Toto je využitelné v případě, kdy je KNXD spuštěn na serveru, který není připojen ke sběrnici KNX a ke sběrnici je připojeno vzdálené zařízení. Za předpokladu, že operační systém tohoto vzdáleného zařízení je Linux, je možné použít například příkaz *socat* k připojení vzdáleného socketu k lokálnímu **tpuart**.

Tabulka 8: Přehled fyzických rozhraní KNXD. Zdroj: [Autor]

Rozhranní	Popis
ft12	Připojení pomocí FT1.2 protokolu k BCU 2
ip	Připojení k KNX IP bráně nebo k jiné instanci knxd
ipt	Připojení tunelovacího protokolu k KNX IP bráně
iptn	Starší alias pro **ipt** rozhraní, kdy je nastaveno *nat* na hodnotu *true*
bcu1s	Připojení pomocí PEI16 protokolu a BCU ke sběrnici (již nepodporováno)
tpuarts	Připojení pomocí TP-UART (a sériového rozhraní)
usb	Připojení pomocí KNX USB rozhraní (HID-mód)
ncn5120	Připojení k KNX pomocí NCN5120 s 38400 bitů za sekundu a 8 bit módem
tpuarttcp	Používá stejný způsob komunikace jako ne-tcp verze s tím, že je určená ke komunikace na dálku
ncn5120tcp	Viz. **tpuarttcp**
dummy	Používáno čistě pro testování

5.2 Síťové rozhraní KNXD

V tabulce 9 jsou popsány všechny síťové rozhraní, které KNXD podporuje. Nejzajímavější z nich jsou **multicast server**, **tunneling server** a klienti (*multicast client* a *non-multicast client*).

Multicast server je právě to rozhraní, které se využívá při připojení ETS k síti KNX spolu s jedním z fyzických rozhraní z předchozí části práce. Pokud není nastavení KNXD změněno, toto rozhraní je otevřeno na *224.0.23.12* a portu *3671*.

Rozhraní **tunneling server** je používáno k připojení KNX IP. Jak již bylo zmíněno, při komunikaci tímto způsobem jsou KNX telegramy obaleny standardními IP rámci a zasílány po IP síti.

Jak už samotný název napovídá, klientská rozhraní slouží k připojení jejich určeným serverům.

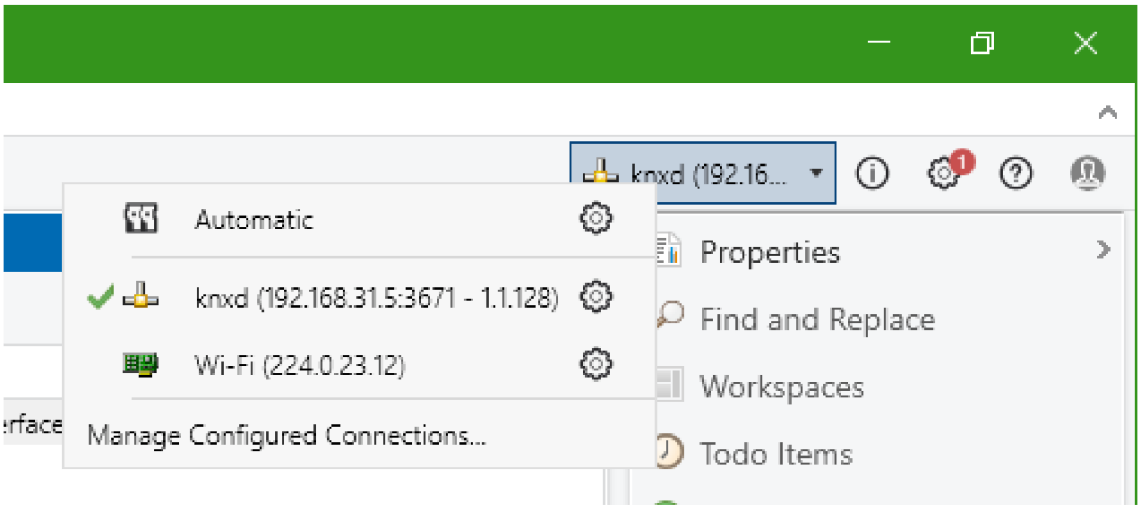
Tabulka 9: Přehled Síťových rozhraní KNXD. Zdroj: [Autor]

Rozhranní	Popis
Local server	Server umožňující lokální připojení, či připojení ze specifické adresy
Multicast server	Server umožňující zasílat multicastová data
Tunneling server	Tunelovací server
Multicast client	Multicastový klient - přiojení k multicast serveru
Non-Multicast client	Ne-multicastový klient

5.3 Spojení KNXD s ETS

Jak již bylo zmíněno, KNXD umožňuje propojit sběrnici KNX a ETS. V obrázku 14 je možné vidět okno výběru připojení ETS. Krom automatického výběru se v seznamu nacházejí další dvě možnosti. První možnost je tunneling, který je zde popsán jednoduše jako *knxd* a druhou možností je multicast, který je zde popsán jako *Wi-Fi*.

Jak už název napovídá, první možnost je na výběr tehdy, když je dostupné rozhraní *tunneling server*. Druhá možnost je na výběr, pokud je dostupné rozhraní *multicast server*, viz. sekce 5.2.



Obrázek 14: KNX TP telegram. Zdroj: [Autor]

6 Deska *H8I8O*

Deska *H8I8O* je vstupní a výstupní modul KNX. Země původu tohoto modulu je Čína. K modulu je také dodávána dokumentace a soubor ETS projektu, ze kterého lze kopírovat katalogový popis zařízení.

Je nutné zmínit, že dokumentace dodaná k desce *H8I8O* je jediným zdrojem informací o ní a že dokumentace nemá uvedeného autora nebo datum publikace, ale pouze kontaktní emailovou adresu. A na závěr je nutné zmínit, že je dokumentace psána v čínštině a při vypracování této práce byl použit automatický překlad.

Deska má 9 konfigurovatelných vstupů a 9 konfigurovatelných výstupů. Vstupy desky jsou označovány jako DI0 až DI8 s tím, že poslední pin je značen jako DIAUX. Co se týče výstupů, značení je podobné, začíná tedy DO1 až DO8, opět s tím, že poslední výstup je značen DOAUX. Piny DIAUX a DOAUX jsou značeny jinak z toho důvodu, že je možné je nakonfigurovat tak, že DIAUX může fungovat jako alternativa programovacího tlačítka a DOAUX jako alternativa programovací kontrolky.

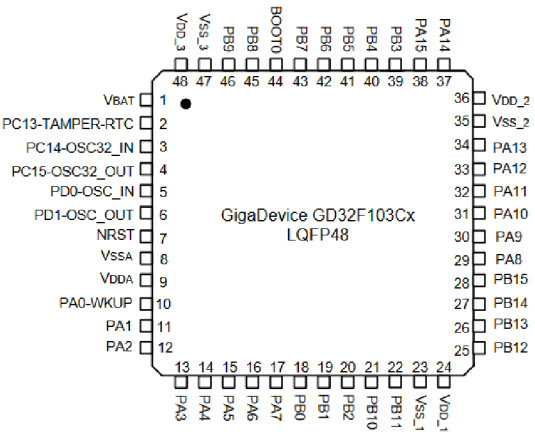
Funkcionality pinů popsané v dokumentaci lze rozdělit podle toho, jestli se jedná o vstupní, nebo výstupní funkce. Co se týče vstupních funkcí, deska umí po připojení tlačítka na jeden z nastavitelných vstupů rozlišit krátké a dlouhé stisknutí. Na základě toho může provést jednu z akcí viz. seznam níže:

- On / Off - Zaslání komunikačního objektu reprezentujícího vypnutí, nebo zapnutí.
- Stmívání - Jak už název napovídá, jedná se o zaslání komunikačního objekt reprezentující stmívání osvětlení.
- Rolety - Odesílání komunikačních objektů reprezentujících data pro akční členy rolet, nebo jim podobným.
- Fixed value - Umožňuje zasílat komunikační objekt s nastavenou hodnotou.
- Multi value cycle - Umožňuje zasílat komunikační objekty s jednou z přednastavených hodnot (hodnota je určena cyklicky).

Co se týče výstupních funkcí, deska podporuje standardní zapnutí, či vypnutí výstupních pinů s tím, že je možné konfigurovat jaký stav pinu desky bude logická hodnota reprezentovat. Dále také podporuje PWM (Pulse Width Modulation), které může být například použito pro stmívání světel. [25]

6.1 MCU

Deska *H8I8O* je vybavena MCU (Microcontroller Unit) označeným jako ***GD32F103CBT6***. Výrobcem tohoto MCU je společnost GigaDevice. Co se týče architektury, jedná se o ARM® Cortex™-M3 a šířka datové sběrnice je 32 bitů. Při návrhu a výrobě tohoto MCU bylo cílem, aby tato platforma byla levná, energeticky nenáročná a měla rychlou odezvu na přerušení. Maximální frekvence jádra je 108 MHz. Co se týče pamětí, tak programová paměť má 128 Kbit a RAM má 20 Kbit. [26]



Obrázek 15: *GD32F103CBT6* MCU. Zdroj: [27]

GD32F103CBT6 je v pouzdře *LQFP48*, viz. obr. 15. Toto pouzdro má celkem 48 pinů, z nichž lze použít až 37 jako GPIO (General Purpose Input/Output). Piny jsou 5 V tolerantní, tzn. přesto, že MCU používá 3,3 V logiku, lze použít i 5 V logiku.

Jak již bylo zmíněno, tento MCU má až 37 GPIO, nicméně většina pinů má specifické alternativní funkce. Tyto funkce se většinou týkají podporovaných periferií. Tzn. například piny PA9 a PA10 mohou sloužit jako běžné vstupy, či výstupy, nicméně pokud bude využitý UART1, tak bude pin PA9 použit pro vysílání dat (neboli jako *Tx*) a pin PA10 pro přijímání dat (neboli jako *Rx*). Seznam všech podporovaných periferií je níže: [26]

- 3× UART
- 2× I^2C (Inter-Integrated Circuit)
- 2× SPI (Serial Peripheral Interface)

- 1× CAN (Controller Area Network) 2.0B
- 1× USB 2.0 full speed
- 2× 12 bitový ADC (Analog to Digital Converter) s 10 kanály

6.1.1 Pinout *GD32F103CBT6* vzhledem k desce *H8I8O*

V tabulce 10 jsou popsány piny MCU a jejich funkcionality na desce *H8I8O*. Tato tabulka bude klíčová pro praktickou část práce. Piny označené DI1 až DIAUX ve sloupci *Funkce na desce H8I8O* slouží jako vstupy senzorů, zatím co piny označované DO1 až DOAUX slouží jako výstupy akčních členů.

TP-UART 2 (blíže popsaný v následující sekci 6.2) připojen na UART1 MCU, tzn. piny PA9 a PA10. Dále je z *TP-UART 2* připojen i tzv. *SAVE* pin. Tento pin signalizuje výpadek napětí na KNX sběrnici. Jeho funkce bude blíže popsána v sekci 6.2 a také pin *RESET* sloužící k resetování *TP-UART 2* obvodu.

Dalším pinem je pin PA12, na který je připojena kontrolka signalizující programovací režim, viz. sekce 3.1.

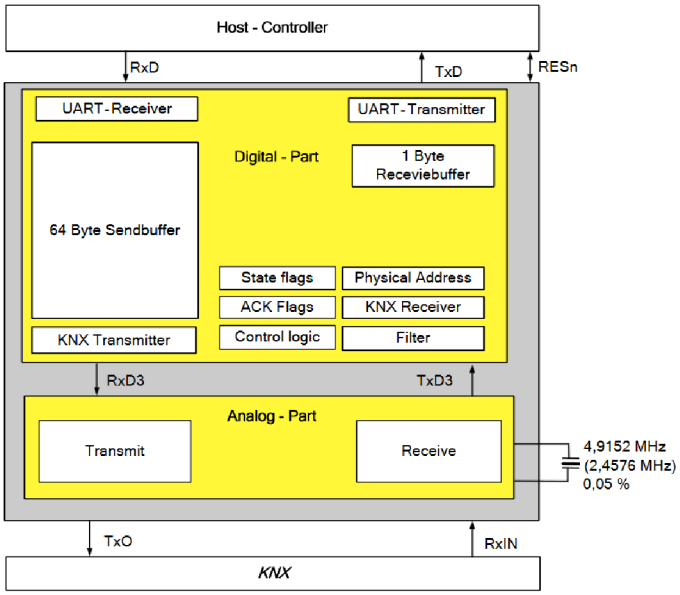
Následuje pin programovacího tlačítka PA11, také viz. sekce 3.1. Kontakty tlačítka jsou zapojeny z jedné strany k pinu MCU a z druhé k zemi, tzn. pro čtení stavu tlačítka je nutné použít interní *pull up* rezistor. [26, 25]

Tabulka 10: Piny *GD32F103CBT6* využité na desce *H8I8O*. Zdroj: [Autor]

Pin MCU	**Funkce na desce *H8I8O***	**Popis**
PA15	DI1	Vstupy a výstupy desky *H8I8O*
PB3	DI2	
PB4	DI3	
PB5	DI4	
PB6	DI5	
PB7	DI6	
PB8	DI7	
PB9	DI8	
PC13	DIAUX	
PA0	DO1	
PA2	DO2	
PA1	DO3	
PA3	DO4	
PA4	DO5	
PA5	DO6	
PA6	DO7	
PA7	DO8	
PB0	DOAUX	
PA9	UART1 Tx	Tx UART rozhraní použitého ke komunikaci s *TP-UART 2*
PA10	UART1 Rx	Rx UART rozhraní použitého ke komunikaci s *TP-UART 2*
PB15	SAVE PIN	Save signál z *TP-UART 2*
PA8	TP-UART RESET	Reset *TP-UART 2* integrovaného obvodu
PA12	LED PIN	Programovací kontrolka
PA11	BUTTON PIN	Programovací tlačítko

6.2 *TP-UART 2*

TP-UART 2 je integrovaný obvod v pouzdře *QFN36X36* sloužící k převodu UART komunikace na klasické KNX TP médium, viz. obr. 16. Při komunikaci s UART rozhraním je brán jako první LSB (Least Significant Bit) a při nečinnosti je očekávána logická 1. Co se týče rychlosti komunikace na UART rozhraní, *TP-UART 2* podporuje dva režimy. Prvním je klasických 9600 a druhý 19200 bitů za sekundu. Nutno podotknout, že komunikace po KNX TP je vždy rychlostí 9600 bitů za sekundu, viz. sekce 3.2, tzn. při komunikaci rychlostí 19200 bitů za sekundu po UART jsou data uložena do bufferu o velikosti 64 bajtů (viz. obr. 16) a následně pomaleji odeslána po KNX TP rychlostí 9600 bitů za sekundu. UART rozhraní podporuje jak 3,3 V, tak 5 V napětí, tzn. v případě použití na desce *H8I8O* s MCU *GD32F103CBT6* se jedná o 3,3 V režim. [28]



Obrázek 16: *TP-UART 2* blokový diagram. Zdroj: [28]

Co se týče komunikace ze strany KNX TP (viz. analog part v obr. 16), *TP-UART 2* zastává linkovou vrstvu, včetně LLC a MAC, viz. sekce 2.2.5.

Dále také *TP-UART 2* umí generovat 20 V_{DC} (maximální proud je 30 mA) pro případné akční členy atd. a 3,3 nebo 5 V pro MCU čistě z napětí na sběrnici KNX TP, tzn. právě *TP-UART 2* umí zajistit napájení celého zařízení skrze napětí na sběrnici. [28]

6.2.1 Pinout *TP-UART 2* vzhledem k desce *H8I8O*

V tabulce 11 jsou popsány piny *TP-UART 2* na desce *H8I8O*. Tato tabulka bude opět klíčová v praktické části práce. Piny *VB+* a *VBV–* jsou určené k připojení sběrnice KNX TP. Pin *VBV–* je zároveň používán jako tzv. zem.

Piny *V20* a *VCC* jsou použity pro napájení desky a případného příslušenství z napětí na sběrnici KNX. V případě desky *H8I8O* je *VCC* nastaven na 3,3 V.

Piny *RxD* a *TxD* jsou použity pro komunikaci s UART1 MCU. Pokud dojde k nenapravitelné chybě v komunikaci, pin *RESn* umožňuje případně resetovat *TP-UART 2*.

Pin SAVE signalizuje pokles napětí na sběrnici, například při výpadku rozvodové sítě. Pokud tato situace nastane, jeho napětí je sraženo k zemi a značí MCU, že po vyčerpání zásob energie v kondenzátoru (viz. sekce 3.1) dojde k vypnutí způsobeném nedostatkem energie. Tzn. tento signál dává MCU šanci uložit potřebné parametry do paměti předtím, než bude vypnut. [28, 25]

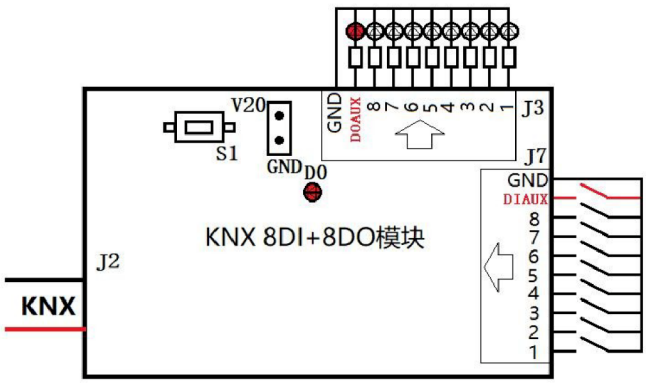
Tabulka 11: Piny *TP-UART 2* využité na desce *H8I8O*. Zdroj: [Autor]

Pin *TP-UART 2*	**Funkce na desce *H8I8O***	**Popis**
VB+	Konektor + KNX TP	Připojení + KNX TP
VB-	Konektor - KNX TP	Připojení - KNX TP
V20	Zdroj 20 V	Interně generovaný zdroj 20 V
VCC	3,3 nebo 5 V	Externí zdroj 3,3 nebo 5 V
RxD	*TP-UART 2* Rx	*TP-UART 2* Rx - MCU Tx
TxD	*TP-UART 2* Tx	*TP-UART 2* Tx - MCU Rx
RESn	Reset *TP-UART 2*	Resetování integrovaného obvodu *TP-UART 2*
SAVE	Save signál	Signalizace výpadku napětí na sběrnici KNX TP

6.3 Pinout a rozložení desky *H8I8O*

V dokumentaci desky *H8I8O* je popsáno její rozložení a některé její vstupy a výstupy, viz. obr. 17. V levé straně obrázku je možné vidět konektor *J2*, který slouží pro připojení KNX TP svorky, viz. obr. 5. V pravé části obrázku se nacházejí konektory *J3* a *J7*. Konektor *J3* obsahuje všechny *DO* výstupy a konektor *J7* všechny *DI* vstupy. Dále se v horní části obrázku nachází neoznačený konektor, který je výstupem zmíněného 20 V zdroje pro případné akční členy. A také tlačítko S1, které slouží jako programovací tlačítko a kontrolka D0, která slouží jako programovací kontrolka.

Dále je nutné zmínit, že na desce se nachází ještě jeden v dokumentaci nezmíněný konektor. Piny na tomto konektoru jsou *3,3 V*, *GND*, *PA14 (SWCLK)*, *PA13 (SWDIO)*. Vzhledem k tomu, že piny *SWDIO* a *SWCLK* u MCU slouží k programování samotného MCU, lze se domnívat, že tento konektor sloužil k nahrání softwaru při vývoji či výrobě desky pomocí JTAG (Joint Test Action Group), nebo SWD (Serial Wire Debugging). Vzhledem k cíli této práce, bude tento konektor klíčový. [25, 27, 29]



Obrázek 17: Pinout desky *H8I8O*. Zdroj: [25]

7 Programování desky *H8I8O*

Vzhledem k cíli práce je nutné zjistit, jaké jsou možnosti ohledně programování desky *H8I8O* a zdali to bude vůbec možné. Následně je nutné zvážit, jaké technologie lze případně použít a jak při samotném programování desky postupovat.

7.1 Programování desky s *STM32 Cube Programmer*

Vzhledem k tomu, že MCU desky je alternativa k řadě MCU *STM32F103* od společnosti *STMicroelectronics*, oficiální způsob programování spočívá v použití *STM32 Cube Programmer* vývojového prostředí a USB zařízení s názvem *ST Link (V2)*.

Samotné USB zařízení musí být připojeno k desce, viz. sekce 6.3. Vzhledem k tomu, že deska obsahuje i integrovaný obvod *TP-UART 2*, je nutné desku nejprve připojit ke KNX zdroji. Pokud by se tak nestalo, může dojít k permanentnímu poškození *TP-UART 2*. Připojený zdroj KNX však může být rizikem ať už desky, nebo počítače, ke kterému je připojeno programovací USB zařízení. Nastat můžou dvě rizikové situace. První je vytvoření tzv. zemnící smyčky, která může při kompenzaci parazitních proudů a potenciálních rozdílů v zemních potenciálech poškodit ať už USB zařízení, nebo desku samotnou. Druhou hrozbou je fakt, že napětí KNX zdroje je přibližně 30V_{DC}. Toto napětí sice spadá do bezpečných hodnot pro člověka, ovšem když je deska spojena přes USB zařízení k portu počítače, může případně dojít k jeho poškození. Z těchto důvodů není mezi deskou a USB zařízením připojen pin GND (zem) a USB zařízení je připojeno k počítači vždy skrze USB izolátor, který v případě nehody počítač ochrání.

Po připojení desky, které bylo popsáno výše a prozkoumání informací ve vývojovém prostředí *STM32 Cube Programmer* bylo zjištěno, že MCU má bohužel zapnutý příznak, který značí ochranu před čtením paměti. To z pohledu programování není až takový problém, nicméně to znamená, že není možné zálohovat originální software desky a při nahrání vlastního softwaru je originální nenávratně ztracen. I přes tuto komplikaci byl na desku nahrán testovací software, který rozblikal programovací kontrolku na desce a tím bylo potvrzeno, že desku programovat lze.

7.2 Programování desky s *PlatformIO*

PlatformIO je open-source ekosystém pro vývoj embedded a IoT aplikací. Je navržen tak, aby usnadnil a zjednodušil proces vývoje softwaru pro různé MCU, platformy a vývojové desky. *PlatformIO* poskytuje nástroje pro vývoj, sestavování, nahrávání a ladění aplikací, které běží na MCU a dalších embedded systémech.

Po prozkoumání podporovaných platforem bylo zjištěno, že platforma STM32 je podporovaná. Dále, že je podporovaný i *STM32F103CB*, což sice není přímo *GD32F103CBT6*, nicméně jak už bylo zmíněno, tyto MCU jsou spolu kompatibilní a proto lze *PlatformIO* využít namísto *STM32 Cube Programmer*, což značně usnadní práci a otevře spoustu nových možností.

Krom nástrojů potřebných k sestavování a nahrávání aplikací do MCU, *PlatformIO* umí také spravovat knihovny a jejich verze. Nicméně je nutné podotknout, že ne všechny knihovny v registru jsou dobře udržované a často se stává, že obsahují nějakou nedokonalost, kterou jejich autor sice opravil, ovšem změny nahrál pouze do Git repozitáře a nová verze knihovny v registrech *PlatformIO* chybí. Naštěstí *PlatformIO* podporuje i vkládání knihoven jako odkazy na repozitáře, či přímo do struktury projektu. Tato řešení jsou poměrně praktická. Vzhledem k tomu, že oboje řešení nabízí možnost přidat i specifickou větev, nebo tzv. *commit* repozitáře, lze tak i zabránit nepředvídaným změnám z pohledu knihoven a tím zajistit poměrně dobrou kompatibilitu v rámci projektu.

PlatformIO také podporuje řadu frameworků. V případě desky *H8I8O* padají v úvahu dva. Prvním je *STM32Cube*, který ve spojení s *PlatformIO* tvoří v podstatě náhražku k *STM32 Cube Programmer* a obsahuje základní API (Application Programming Interface) a HAL (Hardware Abstraction Layer) pro abstrakci mezi tzv. *low level* funkcemi různých STM32 MCU. Druhým frameworkem je poměrně populární Arduino. Vzhledem k jeho popularitě je tento framework využíván v široké řadě platforem a STM32 je jedna z nich. Dále je tento framework díky jeho jednoduchosti rozšířen o nespočet knihoven umožňujících velmi jednoduchou práci s ním a obecně je velmi uživatelsky přívětivý. To je jeden z mnoha důvodů, proč byl tento framework zvolen pro tento projekt. [30]

7.3 Programování desky ze strany ETS

Vzhledem k cíli práce a faktu, že zařízení KNX by měly mít nastavitelné parametry a měla by být možnost seskupovat jejich komunikační objekty do skupin, je nutností, aby šla i tato funkcionalita otestovat. K tomuto účelu lze využít několika již zmíněných poznatků, viz. následující sekce.

7.3.1 Instalace KNXD na Raspberry Pi

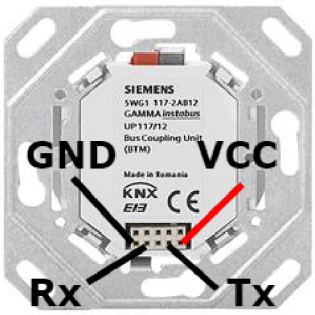
Jak již bylo zmíněno v sekci 5, KNXD umožňuje propojení ETS se sběrnicí KNX. Dále bylo také zmíněné, že KNXD podporuje operační systém Linux. Raspberry Pi je malý jednodeskový počítač, který podporuje řadu operačních systémů a většina těchto systémů je právě formou Linuxu. Z tohoto důvodu bylo právě Raspberry Pi vybráno jako vhodné zařízení pro instalaci KNXD a připojení ke KNX sběrnici.

Jako operační systém Raspberry Pi byl vybrán tzv. *Raspbian*, který je oficiální adaptací Linuxové distribuce *Debian*. Instalace tohoto systému na Raspberry Pi je podrobně popsána na oficiálních stránkách Raspberry Pi.

Po instalaci *Raspbianu* a následném spuštění Raspberry Pi, přihlášení atd. je nutné vytvořit uživatele se jménem *knxd* a přidělit mu příslušná práva. Pro zjednodušení tomuto uživateli byla přidělena práva administrátora. Dále pomocí příkazu `git clone` byl naklonován repozitář KNXD. Repozitář KNXD obsahuje soubor bootstrap.sh, který je nutné spustit pomocí `sudo bash ./bootstrap.sh`. Po tomto kroku následuje standardní sestavení pomocí `sudo ./configure` (zde je možné přidat případné příznaky konfigurace), `sudo make` a `sudo make install`. Po provedení instalace je již možné KNXD spustit pomocí příkazu `knxd`, nebo jako tzv. démon pomocí *systemd*, přesněji příkazu `systemctl` s odpovídajícími argumenty.

7.3.2 Spojení KNXD na Raspberry PI s KNX TP

Po instalaci KNXD zbývá už jen poslední krok k připojení ke KNX TP. Jak je již zmíněno v sekci 3.1, k připojení ke KNX je potřeba tzv. BAU. V případě připojení Raspberry Pi k KNX TP byl jako BAU využit Bus coupling unit od společnosti SIEMENS, viz. obr. 18. Jedná se o modul, který je používán k připojení speciálních spínačů a zařízení jim podobných ke KNX TP. [31]

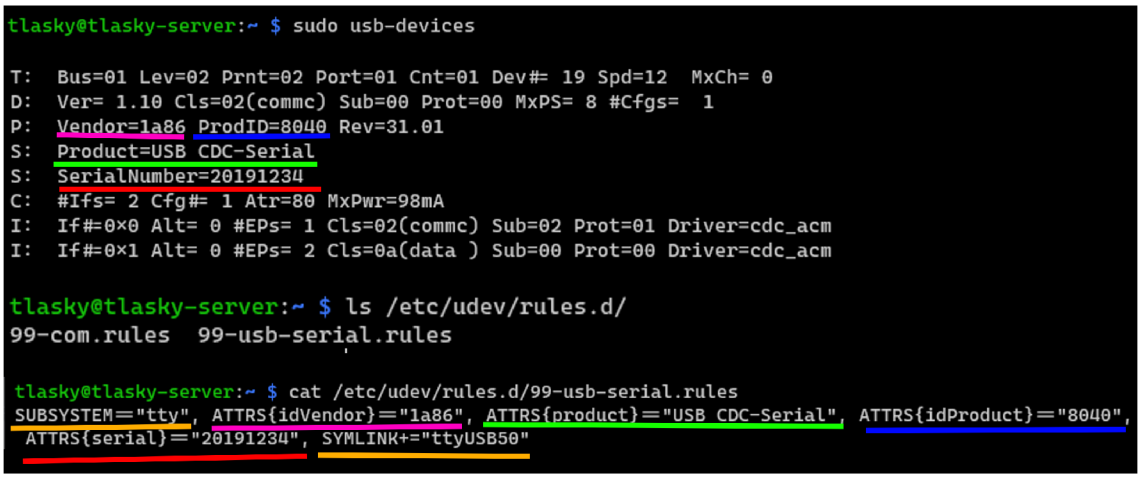


Obrázek 18: Bus coupling unit Zdroj: [31]

Tento modul v podstatě obsahuje jen integrovaný obvod *TP-UART 2* a komponenty potřebné k jeho provozu. V obrázku 18 je částečně popsán pinout tohoto zařízení.

Naskýtají se dvě možnosti jak tento modul k Raspberry Pi připojit. První možností je použití UART rozhraní, které je vyvedeno na GPIO, respektive na piny 14 pro Tx a 15 pro Rx. Toto řešení má však několik problémů. Prvním je fakt, že například u Raspberry Pi 3 je tento UART využíván pro připojení Bluetooth. Tento problém lze jednoduše vyřešit vypnutím této funkcionality v nastavení Raspberry Pi. Druhým problémem je, že logika Bus coupling unit je 5 V a logika GPIO pinů Raspberry Pi je 3,3 V. Tento problém lze také vyřešit a to pomocí převodníku mezi 5 V a 3,3 V logikou. Třetím problémem je, že Bus coupling unit je připojen ke KNX TP a to je podobně jako u desky *H8I8O* rizikem pro Raspberry Pi a jakákoliv nehoda, či selhání tohoto modulu může být fatální pro Raspberry Pi. Proto by mělo být i toto spojení galvanicky odděleno podobně jako spojení mezi deskou *H8I8O* a počítačem z nějž je deska programována, viz. sekce 7.1.

Druhou možností jak tento modul k Raspberry Pi připojit je využití externího USB na UART převodníku. Tato možnost je výhodná v tom, že podobně jako při programování desky *H8I8O* lze využít tzv. USB izolátor. Dále také v tom, že díky USB rozhraní odpadá problém s nekompatibilní úrovní logiky a i problém s nutností vypnutí Bluetooth rozhraní Raspberry Pi. Díky těmto vlastnostem byla použita právě tato metoda. Nicméně i externí USB na UART převodník má svou nevýhodu. Tato nevýhoda spočívá v tom, že Linux reprezentuje většinu zařízení jako soubor (respektive tzv. file-like objekt) prostřednictvím UDEV (userspace / device). Pokud chybí definice zařízení v konfiguraci UDEV, Linux má tendenci připojovat USB na UART převodník na různá místa v rámci tohoto sytému. Z toho důvodu byl definován záznam pro UDEV na základě příkazu `sudo usb-devices`, viz. obr. 19.



Obrázek 19: Připojení USB na UART převodníku pomocí UDEV. Zdroj: [Autor]

Řádky v obr. 19 podtržené barvami růžová, červená, modrá a zelená reprezentují atributy záznamu, které jsou nutně shodné s výsledkem příkazu `sudo usb-devices`, zatím co první podtržení oranžovou barvou reprezentuje typ připojeného zařízení, tzn. *tty* a druhé podtržení oranžovou barvou reprezentuje umístění, kam bude zařízení připojeno, tzn. `/dev/ttyUSB50`.

Nyní po připojení Bus coupling unit k USB na UART převodníku, převodníku a izolátoru k Raspberry Pi bylo možné spouštět KNXD tak, aby skutečně spojovalo KNX TP a ETS. Toto bylo dosaženo příkazem:

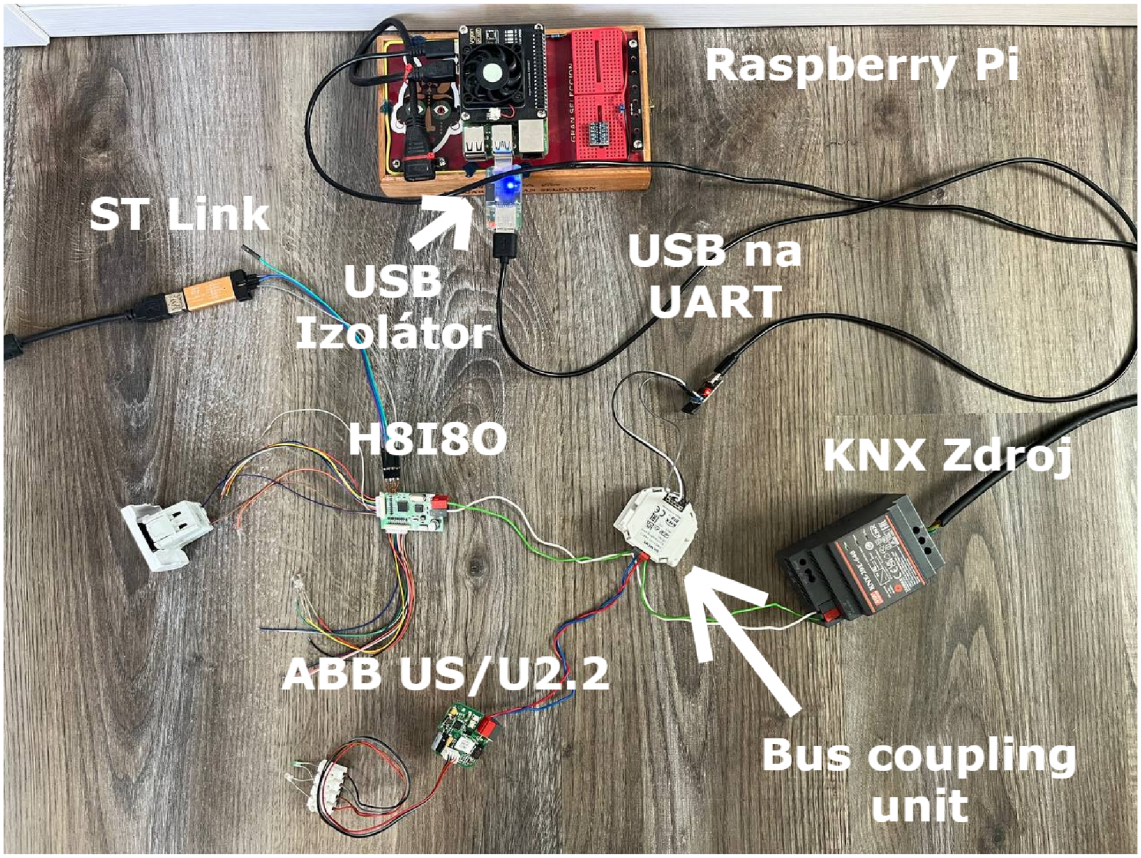
```
knxd --eibaddr="1.1.128" --client-addrs="1.1.129:8" -D -T
   -R -S -i -b "tpuarts:/dev/ttyUSB50" -t 0xffc -f 9
```

Kdy argumenty `-t 0xffc -f 9` zapínají debugovací výstup KNXD, který se hodí při hledání chyby v případě, že spojení nefunguje. Argumenty `--eibaddr --client-addrs` nastavují adresy na kterých je KNXD provozován. Argumenty `-D, -T, -R, -S a -i` spouští discovery (automatické hledání rozhraní v ETS), tunneling (viz. sekce 5), routing (v rámci KNX IP), multicast server (viz. sekce 5) a naslouchání na TCP portu (6720). Poslední argument `-b tpuarts:/dev/ttyUSB50` nastavuje jaké rozhraní má být použito pro komunikaci s KNX TP. V tomto případě se tedy jednalo o rozhraní popsané v UDEV záznamu, viz. předchozí odstavec.

7.4 Zapojení KNX TP pro účely vývoje a testování

Pro testování programu na desce *H8I8O* byl zapojen jednoduchý KNX systém, viz. obr. 20. Na obrázku je možné vidět celkem 4 KNX zařízení. Prvním je zdroj, který je naprosto nezbytný pro každou KNX TP instalaci. Druhým zařízením je Bus coupling unit, který je již zmíněným postupem připojen k Raspberry Pi. Třetím je samotná deska *H8I8O*, která je přes ST Link připojena k počítači a dále má na piny také připojené tlačítko a dvě LED kontrolky určené k testování. Posledním zařízením je již zmíněný modul *ABB US/U2.2*, viz. sekce 4.2, který sloužil k testování kompatibility s jiným zařízením KNX. K tomuto zařízení jsou také připojené dvě LED kontrolky.

Vzhledem k tomu, že součástí zapojení je KNX zdroj, je nutné dbát na opatrnost, při manipulaci se zapojenými komponenty. I přes fakt, že dražší zařízení jsou izolována, může dojít k poškození ostatních zařízeni.



Obrázek 20: Zapojení testovací KNX TP sběrnice. Zdroj: [Autor]

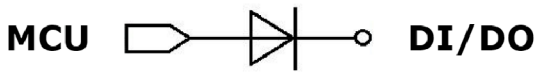
8 Návrh softwaru pro desku *H8I8O*

Po prozkoumání možností, frameworků a možném postupu k dosažení cíle práce, bylo již možné zaměřit se na program pro desku *H8I8O* jako takový. Vzhledem k tomu, že program musí počítat s možností parametrizace desky, tato problematika byla rozdělena do dvou částí. První část se zabývá problematikou aplikace desky *H8I8O* pro ETS, která udává rozhraní, parametry a komunikační objekty KNX. Druhá část řeší problematiku programu desky.

8.1 Požadované funkcionality

Před začátkem návrhu aplikace pro ETS a návrhu programu pro desku *H8I8O* bylo nutné ujasnit si, jaké funkcionality budou podporovány a jakým způsobem budou implementovány. Hlavním důvodem je fakt, že aplikace ETS a program desky musí být po stránce parametrů a čísel komunikačních objektů naprosto totožné a jakákoliv změna vyžaduje většinou radikální změnu jak v aplikaci ETS, tak v programu desky.

Podobně jako v sekci 6, bylo na tyto funkcionality nahlíženo jako na vstupní a výstupní. Originální software desky rozlišoval krom vstupních a výstupních funkcí i vstupní a výstupní piny. Proto jsou piny označeny jako DI(n) a DO(n). Při vývoji vlastního softwaru bylo na desku nahlíženo jinak. Po bližším zkoumání dokumentace a desky samotné bylo zjištěno, že všechny piny DI(n) a DO(n) jsou zapojeny stejným způsobem, viz. obr. 21.



Obrázek 21: Vstupy a výstupy na desce *H8I8O*. Zdroj: [25]

Tzn. ať už se jedná o vstupní, nebo výstupní piny, vždy se jedná o pin MCU, za kterým je připojena dioda a její výstup je vyveden na konektor J3, nebo J7, viz. sekce 6.3. Na základě tohoto faktu bylo rozhodnuto, že ať už se jedná o piny patřící do skupiny DI(n), nebo do skupiny DO(n), bude je možné nastavit jako vstup, nebo výstup. V případě výstupu není nutné řešit jakékoliv problémy vzhledem k tomu, že dioda výstupní signály propustí. Pokud budeme chtít nastavit pin jako vstup, bude nutné pin nastavit jako tzv. ”input pullup”, což bude objasněno v nadcházející části práce.

Co se týče výstupních funkcí, bylo rozhodnuto, že každý pin bude podporovat standardní On / Off funkcionalitu, která může být užitečná například při rozsvěcení, či zhasínání světel a dále také PWM, což může být použito například při stmívání. Dále, že piny DO6, DO7 a DO8 bude možné seskupit pro ovládání RGB osvětlení.

Realizovány tedy budou následující vstupní funkcionality, viz. seznam níže:

1. Spínání pomocí přepínače
2. Spínání pomocí tlačítka (s alternativními funkcemi)
3. Spínání pomocí dvojkliku tlačítka
4. Spínání pomocí dlouhého stisku tlačítka
5. Čtení impulzů elektroměru
6. Ovládání scén (4 funkce)

První funkcionalita slouží ke snímání standardních přepínačů. Tzn. přepínač drží stav On / Off, takže je snímána ať už vzestupná, nebo sestupná hrana signálu. Druhá funkcionalita se zabývá snímáním tlačítek. Tzn. spínače, který nedrží svůj stav. Tyto spínače jsou v rámci KNX běžnější. Dále tato funkcionalita umožňuje nastavit alternativní funkce na dvojité stisknuti, nebo na dlouhé stisknutí, čímž umožňuje rozšířit funkce jednoho tlačítka. Následující dvě funkcionality umožňují nastavit akce pouze na dvojité stisknutí, nebo na dlouhé stisknutí. Následující funkcionalita umožňuje číst počet pulzů vydávaných elektroměrem, které reprezentují počet zaznamenaných kWh. Poslední funkcionalita se zabývá scénami. Těmi je v podstatě myšleno vyvolání několika akcí (v tomto případě 4 akcí) v rámci jednoho vstupu.

8.2 Návrh aplikace pro ETS

Každá z funkcionalit zmíněných v předchozí sekci by měla mít své parametry a komunikační objekty. Na základě parametrů je možné navrhnout uživatelské rozhraní, které následně slouží k nastavování hodnot parametrů v ETS. Toto uživatelské rozhraní je vždy dodáváno prostřednictvím *knxprod* souboru společně se zařízením, viz. sekce 4.2.

8.2.1 Nastavení desky

Obecně bylo uživatelské rozhraní navrženo tak, že první obrazovka obsahuje nastavení desky. Jako nastavení desky je bráno nastavení RGB, které při zapnutí skryje nastavení výstupů rezervovaných pro RGB a zobrazí dva komunikační objekty. Prvním je komunikační objekt, který nastavuje barvu RGB kanálů. Jeho data point je nastaven na 232.600. Tento data point má velikost tří bajtů, kdy první bajt reprezentuje červenou barvu (R), druhý zelenou barvu (G) a třetí modrou (B). Druhý komunikační objekt nastavuje stav světla, tzn On / Off funkce. Proto je jeho data point nastaven na 1.001 (Switch).

8.2.2 Nastavení pinů desky

Dále uživatelské rozhraní obsahuje záložky, uvnitř kterých se skrývá nastavení jednotlivých pinů desky. V těchto záložkách lze nastavit jakou funkci z dříve uvedeného seznamu má pin plnit, toto nastavení je umožněno pomocí parametru *mode*. Pokud tento parametr nastavíme na funkci *Spínání pomocí přepínače*, je dále umožněno specifikovat následující parametry:

- Debounce
- Init state
- Switch type

Parametr *debounce* značí prodlevu při čtení stavu pinu, která zabraňuje chybnému čtení v případě poskakujících kontaktů spínače a jedná se o hodnotu v milisekundách. Dále je možné nastavit hodnotu parametru *init state*. Tento parametr udává, jaký má být stav (On / Off) po inicializaci zařízení. V úvahu se nabízejí 4 možnosti. První je Low, neboli Off. Druhou možností je High, neboli On. Třetí možností je přečíst aktuální stav pinu a analogicky nastavit tuto hodnotu podle něj. Čtvrtá možnost je podobná možnosti třetí, nicméně je čtená hodnota negována. Posledním parametrem je *switch type*. Tento parametr udává, zdali je připojený spínač v klidovém stavu otevřený, nebo spojený. V případě nastavení této funkcionality je zobrazen komunikační objekt s data pointem typu 1.001, který je zaslán v případě zaznamenání změny stavu přepínače.

Pokud je nastaven parametr *mode* na funkcionalitu *Spínání pomocí tlačítka (s alternativními funkcemi)*, parametry z předchozí funkce zůstávají a jsou rozšířeny o následující:

- Double click enabled

- Long press enabled
  - Long press delay

Parametr *double click enabled* umožňuje zapnout první alternativní funkci tlačítka, která je spuštěna dvojitým stisknutím. Zatím co parametr *long press enabled* umožňuje zapnout alternativní funkci spuštěnou dlouhým stiskem tlačítka. Pokud je tato možnost vybrána, je dále zobrazen parametr *long press delay*, který umožní nastavit délku dlouhého stisku. Tento parametr je v milisekundách. Podobně jako předchozí funkcionalita má i tato komunikační objekt typu 1.001, který reprezentuje stav On / Off. Ovšem pokud je vybrána jedna, nebo obě alternativní funkce, jsou dále zobrazeny další komunikační objekty typu 1.001, dle výběru.

Dále je možné nastavit parametr *mode* na jednu z funkcionalit *Spínání pomocí dvojkliku tlačítka*, nebo *Spínání pomocí dlouhého stisku tlačítka*. Pokud je vybrána jedna z těchto funkcionalit, je možné nastavit parametry stejně jako i funkcionality první (*Spínání pomocí přepínače*), ovšem v případě *Spínání pomocí dlouhého stisku tlačítka* bude k dispozici v předchozí sekci zmíněný parametr *long press delay*.

Pokud je parametr *mode* nastaven na funkcionalitu *Čtení impulzů elektroměru* je umožněno nastavit následující parametry:

- Pulses per kWh
- Send cyclically / per kWh
  - Cycle delay
  - kWh count

Parametr *pulses per kWh* vyjadřuje kolik pulzů elektroměru vyjadřuje jednu kWh. Tento počet se může lišit dle typu elektroměru. Dále je možné pomocí parametru *end cyclically / per kWh* nastavit způsob zasílání hodnot. Pokud je zvolena možnost zasílání cyklicky, data budou zaslána periodicky s prodlevou nastavenou parametrem *cycle delay*, který je v milisekundách. Pokud bude zvolena možnost zasílání za počet kWh (*per kWh*), budou data zasílána až po docílení nárůstku nastaveným parametrem *kWh count*. Pokud je tato funkcionalita zvolena, je zobrazen komunikační objekt s data pointem 13.013, který vyjadřuje aktivní energii v kWh.

Pokud je parametr *mode* nastaven na funkcionalitu *Ovládání scén (4 funkce)*, lze nastavit parametry *debounce*, *init state* a *switch type*, viz předchozí funkcionality. Dále je možné nastavit 4 různé akce. Každá z těchto

akcí může být nastavena na jeden ze dvou druhů, a každý druh má možnost nastavit specifickou hodnotu, viz. seznam níže:

- 1bit mode
  - On / Off
- 8bit mode
  - Hodnota v rozsahu 0 až 255

Tato funkcionalita nabízí 4 separátní komunikační objekty, které reprezentují jednotlivé akce, které jsou buďto typu 1.001, nebo typu 5.001 s ohledem na druh akce. Oproti předešlým funkcionalitám, tyto komunikační objekty musí být vysílány při stisku tlačítka a zároveň musí aktualizovat své hodnoty na základě akcí ostatních zařízení (tzn. musí číst hodnoty ze sběrnice). To je z toho důvodu, že při odvolání scény je chtěné aby byla ovládaná zařízení nastavena do stavu před spuštěním scény. Dále jsou tyto hodnoty také čteny při inicializaci zařízení.

8.2.3 Soubory *knxprod* a jejich možná realizace

Jak již bylo zmíněno, soubory *knxprod* definují aplikaci a její rozhraní v ETS. Tyto soubory jsou v podstatě obyčejnými *zip* archivy, které mají však pevně definovanou vnitřní strukturu a musí obsahovat soubory, viz. seznam níže:

1. knx_master.xml
2. M-XXXX.signature
3. M-XXXX/Catalog.xml
4. M-XXXX/Hardware.xml
5. M-XXXX/M-XXXX_A-XXXX-XX-XXXX.xml

První soubor v seznamu musí obsahovat informace o výrobci zařízení, typy data pointů, verzi masky zařízení, atd. Verze masky udává jaké přenosové média zařízení podporuje.

Druhý soubor obsahuje tzv. podpis, který zajišťuje, že jde importovat jen aplikace oficiálních zařízení, nicméně existuje mnoho aplikací, které tento podpis umí vygenerovat.

Zbylé soubory jsou obsaženy ve složce M-XXXX, kde všechny X odpovídají identifikátoru výrobce. Prvním souborem v této složce je

soubor, který obsahuje katalogové informace o zařízení. Druhý popis hardwaru zařízení a třetí kompletní popis aplikace, včetně komunikačních objektů, parametrů atd. Formát jeho jména je následující:

M-{identifikátor výrobce}_A-{číslo aplikace}-{verze aplikace}-{hash aplikace}

Kdy je obsah složených závorek společně s nimi nahrazen položkami uvedenými mezi nimi. Hash aplikace je možné vypočítat na základě obsahu souboru.

Vytvoření *knxprod* souboru je možné několika způsoby. Prvním je vytvoření všech souborů a struktury ručně, nebo pomocí scriptu, vše zabalit do zip souboru a následně mu změnit koncovku ze *.zip* na *.knxprod*. Tato možnost má hned několik nevýhod.

První nevýhodou je fakt, že vnitřní struktura veškerých xml souborů obsažených v *knxprod* je poměrně složitá a v případě jakékoliv chyby ETS odmítne importovat zhotovený *knxprod* soubor bez jakéhokoli hlášení o chybě.

Druhou nevýhodou je, že pro úsporu paměti samotného zařízení je nutné parametry definované v *knxprod* souboru zarovnat tak, aby se co nejlépe vměstnaly do políček o velikosti jednoho bajtu, čímž dojde k jejich přeházení a následné získávání těchto parametrů je dále poměrně složité.

Z těchto a několika dalších důvodů bylo rozhodnuto, že k vytvoření *knxprod* souboru bude použit program *Kaenx-Creator*[1] . Tento program umožňuje pomocí poměrně jednoduchého uživatelského rozhraní definovat všechny možné parametry, komunikační objekty atd. na základě kterých následně vygeneruje *knxprod* soubor a k němu taky odpovídající *.h* soubor. Uvnitř souboru jsou popsány definice pomocí kterých lze získat veškeré definované parametry a komunikační objekty na základě vygenerovaného *knxprod* souboru. Tato vlastnost je neskutečným ulehčením pro software desky. Ukázka informací o parametru níže:

```
#define APP_DI1_DEBOUNCE    0x0001
#define APP_DI1_DEBOUNCE_Shift  1
#define APP_DI1_DEBOUNCE_Mask 0x007F
// Offset: 1, Size: 7 Bit, Text: DI1 debounce
#define ParamAPP_DI1_DEBOUNCE ((uint)((knx.paramByte(1) >>
   APP_DI1_DEBOUNCE_Shift) & APP_DI1_DEBOUNCE_Mask))
```

[1] *Kaenx-Creator* je dostupný z: `https://github.com/OpenKNX/Kaenx-Creator`

8.3 Návrh programu desky

Jak již bylo zmíněno v sekci 7.2 k programování byl zvolen ekosystém *PlatformIO* a Arduino framework. I přes tato ulehčení by bylo teoreticky nutné implementovat všechny síťové vrstvy KNX od linkové nahoru, viz. sekce 2.2. Naštěstí existuje knihovna pro Arduino framework, která tyto vrstvy implementuje. Jedná se o knihovnu *knx*, jejímž autorem je *Thomas Kunze* (a přispěvatelé). Tato knihovna zajišťuje implementaci všech vrstev KNX a podporuje TP, IP a RF média. Dále se tato knihovna stará o ukládání parametrů přijímaných ze strany ETS a o zasílání, či přijímání komunikačních objektů, čímž tvoří výborný základ pro tvorbu softwaru určenému pro KNX zařízení. Z tohoto důvodu bylo rozhodnuto, že bude tato knihovna použita. [32]

Dále je nutné řešit problematiku s debounce, dvojklikem a dlouhým stisknutím tlačítka. Bylo rozhodnuto, že k tomuto účelu bude použita knihovna *AceButton* od autora *Brian Park* (a přispěvatelů), která umožňuje pomocí jednoduchého callbacku tyto požadavky realizovat. [33]

Dále bylo rozhodnuto, že piny desky *H8I8O* budou reprezentovány objekty třídy *KnxBoardIO*, která zastřeší funkce pinů. Tato třída dále zastřešuje veškeré atributy potřebné pro jednotlivé funkcionality pinů viz. sekce 8.2.2. Následně bylo rozhodnuto, že veškeré funkcionality budou realizovány pomocí callbacků, které na základě atributů objektů *KnxBoardIO*. Tento přístup zajistil možnost rozšiřování funkcionalit. Ze zvoleného přístupu vyplývá, že tato třída musí minimálně obsahovat atribut, který jednoznačně identifikuje, o jaký pin desky se jedná. Dále je také nutné aby na základě tohoto atributu bylo možné jednoznačně určit o jaké číslo pinu z pohledu MCU se jedná.

Vzhledem k počtu parametrů a komunikačních objektů bylo rozhodnuto, že každý typ parametru bude mít svůj *getter* a stejně tak i komunikační objekt. Tyto *gettery* umožňují na základě čísla pinu jednoduše přistoupit k požadovanému parametru, či komunikačnímu objektu, což vedlo k jednoduchému způsobu manipulace s nimi.

9 Implementace softwaru pro desku *H8I8O*

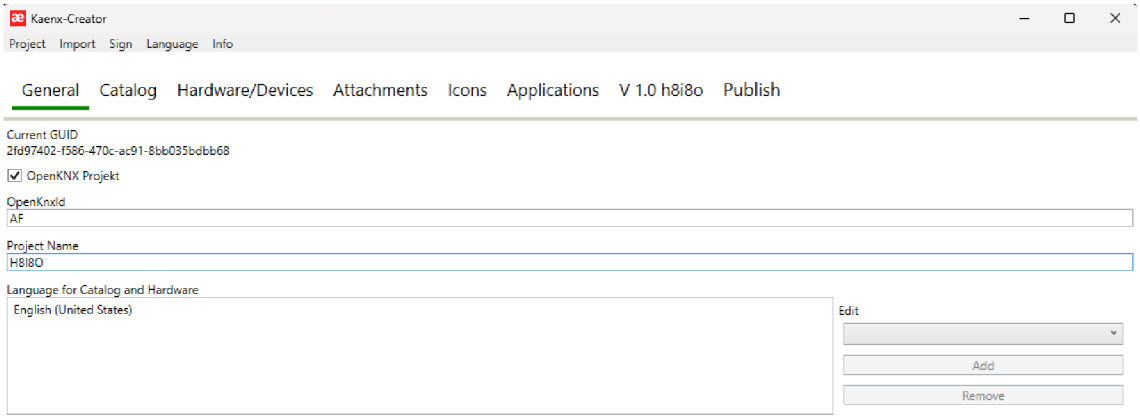
Po rozmyšlení základnách požadavků na software desky a aplikaci pro ETS, bylo možné začít s implementací. Podobně jako u návrhu musí být i implementace rozdělena na dvě části, z nichž první popisuje implementaci aplikace pro ETS a druhá popisuje implementaci sowtwaru pro desku *H8I8O*.

9.1 Implementace aplikace pro ETS

Jak již bylo zmíněno v sekci 8.2.3, pro usnadnění implementace byl zvolen program *Kaenx-Creator*. V tomto programu bylo nejprve nutné vytvořit nový projekt a vyplnit základní informace o projektu, čímž se zabývá následující kapitola.

9.1.1 Vytvoření projektu v *Kaenx-Creator*

Po vytvoření projektu se v programu *Kaenx-Creator* otevře záložka s názvem *General*, viz. obr. 22. V této záložce je nutné vyplnit několik informací. První je identifikátor výrobce zařízení. Program samotný umožňuje zvolit možnost *OpenKNX*, čímž je tento identifikátor nastaven na hodnotu *AF*. Vzhledem k tomu, že se jedná o neoficiální projekt, byla vybrána tato možnost. Dále je nutné nastavit název projektu a jazyk aplikace pro ETS. V tomto případě byl jako název vybrán *H8I8O* a jako jazyk byla vybrána angličtina, viz. obr. 22.



Obrázek 22: Obecné nastavení projektu v *Kaenx-Creator*. Zdroj: [Autor]

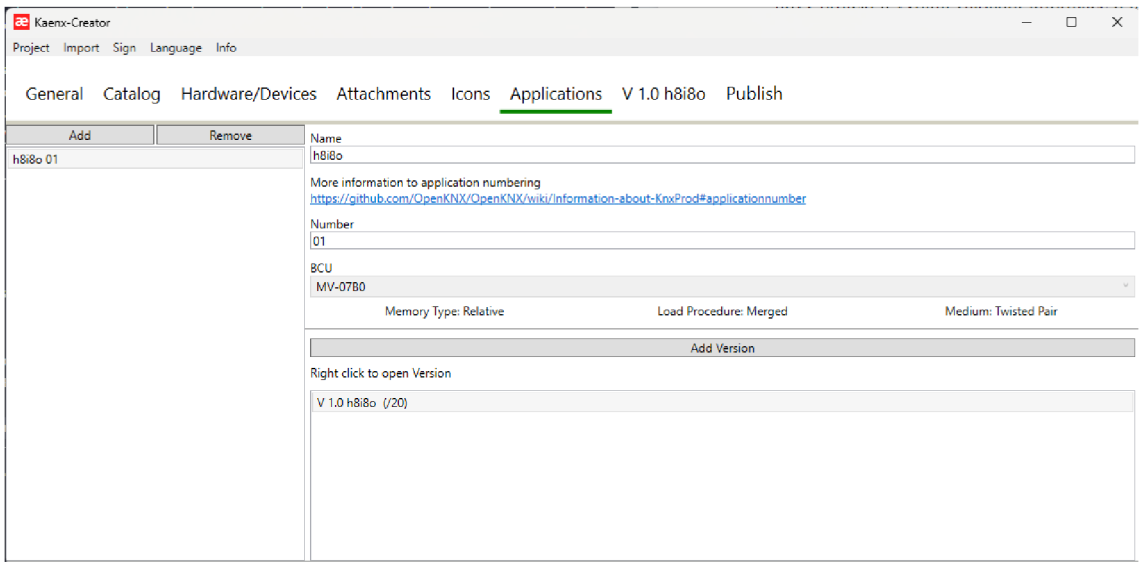
Po nastavení obecných informací o projektu je nutné vytvořit novou aplikaci. Tuto aplikaci lze v tomto kontextu chápat jako jakýsi obal pro všechny parametry a komunikační objekty.

Prvním parametrem při vytváření aplikace je její jméno. V tomto případě bylo opět zvoleno *H8I8O*. Dalším parametrem je číslo aplikace. Obecně platí, že intervaly tohoto čísla jsou rezervovány různým výrobcům, respektive různým zařízením. Vzhledem k tomu, že se jedná o demonstrativní aplikaci, bylo zvoleno jednoduše číslo 1, viz. obr. 23.

Dalším parametrem označovaným *BCU* je maska značící jaký typ přenosového média zařízení používá. Pro desku byla zvolena hodnota *MV-07B0*, která značí, že deska podporuje přenosové médium KNX TP. Hodnoty pro další přenosová média podporovaná Arduino knihovnou *knx* jsou zmíněna v seznamu níže:

- KNX TP: MV-07B0
- KNX IP: MV-57B0
- KNX RF: MV-27B0
- KNX IP/TP: MV-091A
- KNX TP/RF: MV-2920

Po vyplnění aplikace, bylo nutné přidat její verzi. Jedná se o verzi ETS aplikace, která musí odpovídat verzi softwaru na desce *H8I8O*. K tomu slouží tlačítko *Add version*, viz. obr. 23.



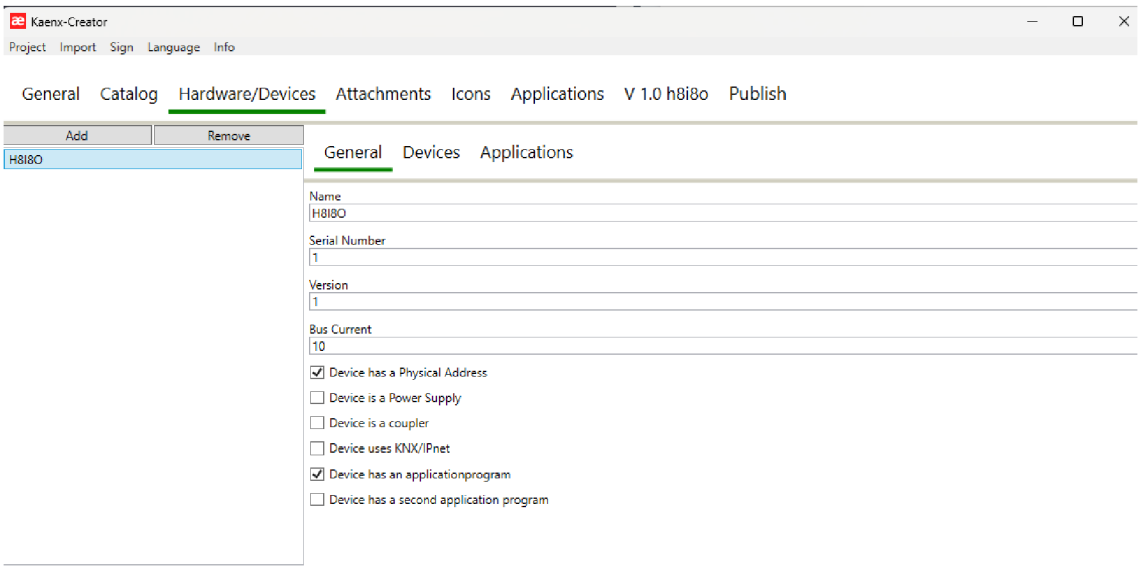
Obrázek 23: Vytvoření aplikace v *Kaenx-Creator*. Zdroj: [Autor]

Po vytvoření verze aplikace je nutné v záložce *Hardware/Devices* vytvořit zařízení, které reprezentuje desku *H8I8O*. Tato záložka má své vnitřní záložky *General*, *Devices*, *Applications*.

V záložce *General* je zapotřebí opět nastavit jméno. Nyní se jedná o jméno zařízení, z tohoto důvodu bylo vybráno *H8I8O*. Dále je nutné vyplnit sériové číslo a verzi hardwaru. Vzhledem k tomu, že se jedná o demonstrační projekt, byly zvoleny hodnoty 1 a 1. S ohledem na to, že se jedná o KNX TP zařízení, je potřeba specifikovat jeho proudovou charakteristiku. Tato charakteristika desky *H8I8O* činí do 10 mA. Z tohoto důvodu byla doplněna hodnota 10. V poslední fázi bylo nutné specifikovat, jaké vlastnosti zařízení má, či ne. Pro toto zařízení byly zaškrtnuty *Device has Physical Address* a *Device has application program*, viz. obr. 24.

V záložce *Devices* bylo následně nutné přidat opět jméno zařízení a jeho popis. Jméno bylo opět vyplněno jako *H8I8O* a popis zařízení jako *Universal IO*. Dále je velmi důležité aby byla v této záložce zaškrtnuta možnost **Device is for REG**. Bez této možnosti by aplikace pro toto zařízení nešla exportovat do *knxprod* souboru.

Ze tří záložek uvnitř záložky *Hardware/Devices*, zbývá už jen *Applications*. Zde je nezbytné přiřadit aplikaci vytvořenou v předchozím kroku. Pokud by se tak nestalo, opět by nebylo možné vytvořit soubor *knxprod*.



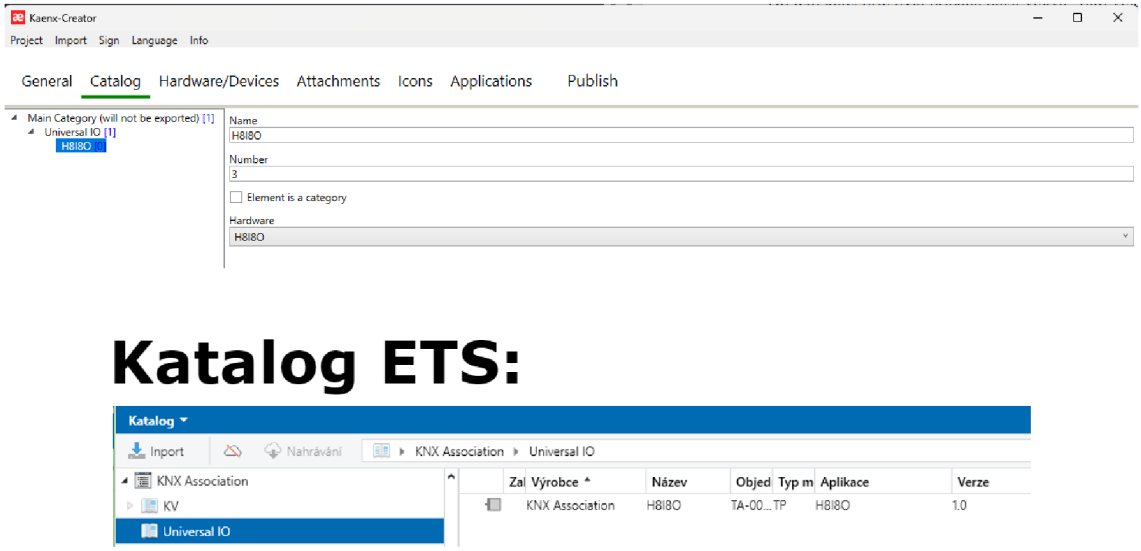
Obrázek 24: Vytvoření zařízení v *Kaenx-Creator*. Zdroj: [Autor]

Jak již bylo zmíněno v sekci 4.2, každé zařízení by mělo být zařazeno do katalogu. Pokud tomu tak není, vygenerovaný soubor *knxprod* by sice šel pomocí ETS importovat, nicméně nebylo by možné zařízení přidat, vzhledem k tomu, že se zařízení přidávají právě z něj.

V obrázku 25 je možné vidět, že *Kaenx-Creator* umožňuje vytvoření větvené struktury odpovídající katalogu ETS. Kořen této struktury je zde nazván *Main Category*. Tato kategorie reprezentuje výrobce zařízení. V případě desky *H8I8O* byla vybrána asociace KNX (*KNX Association*).

Druhá kategorie reprezentuje kategorii zařízení. V tomto případě bylo vybráno jméno Universal IO, vzhledem k funkcionalitám desky. Vzhledem k tomu, že větvení katalogu musí mít minimálně tři úrovně, je tato kategorie nutná, viz. sekce 4.2.

Do této kategorie byla přidána další vrstva. Tato vrstva již reprezentuje samotné zařízení. Z tohoto důvodu je nutné vypnout možnost *Element is a category*. Následně je nutné vybrat jaký hardware toto zařízení reprezentuje. V případě desky *H8I8O* byl vybrán hardware vytvořený v předchozí části práce.



Katalog ETS:

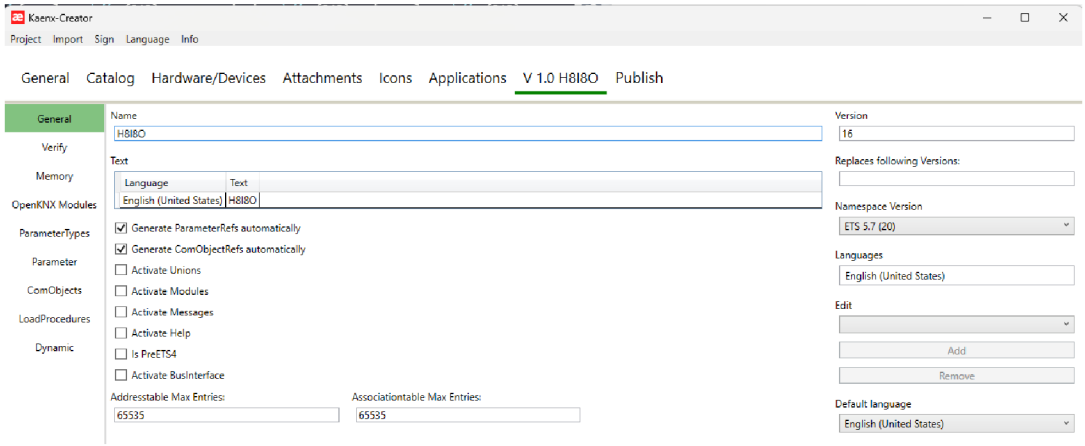
Obrázek 25: Vytvoření katalogové položky v *Kaenx-Creator*. Zdroj: [Autor]

V dolní části obrázku 25, je možné vidět, jakžm způsobem je zařízení v katalogu umístěno. Tzn. po importu *knxprod* souboru do ETS zařízení bylo umístěné v katalogu následovně:

Katalogy ➔ *KNX Association* ➔ *Universal IO* ➔ *H8I8O*.

9.1.2 Vytvoření aplikace a paměti v *Kaenx-Creator* projektu

Po vytvoření základního projektu v *Kaenx-Creator* bylo možné začít s tvořením aplikace jako takové.



Obrázek 26: Základní informace o aplikaci v *Kaenx-Creator*. Zdroj: [Autor]

V obrázku 26 je možné vidět jaké základní parametry byly v aplikaci pro desku *H8I8O* nastaveny. Krom běžných parametrů jako je jméno, jazyk atd. byly zapnuty dvě velmi ulehčující funkce.

První funkcí je tzv. *Generate ParameterRefs authomatically*. Vzhledem k tomu, že na parametry definované v aplikaci je nutné se odkazovat, existují v rámci *Kaenx-Creator* tzv. *ParametrRefs*, které slouží právě k tomu. Obecně se jedná o entity držící referenci na Parametr a dále je odkazováno na tyto entity. Tyto entity lze vytvářet uvnitř *Kaenx-Creator* manuálně, nicméně vzhledem k počtu parametrů by vytváření těchto referencí bylo velmi náročné a mohlo by jednoduše dojít k chybě. Proto *Kaenx-Creator* umožňuje tyto reference generovat automaticky.

Druhá funkce, která byla zapnuta je *Generate ComObjectRefs authomatically*. Podobně jako u parametrů, i na komunikační objekty je potřeba vytvářet reference. V případě komunikačních objektů se tyto reference jmenují *ComObjecRef*.

Ostatní funkcionality nabízené nastavení aplikace uvnitř *Kaenx-Creator* nebudou v projektu aplikace desky *H8I8O* využity.

9.1.3 Vytvoření paměti aplikace v *Kaenx-Creator* projektu

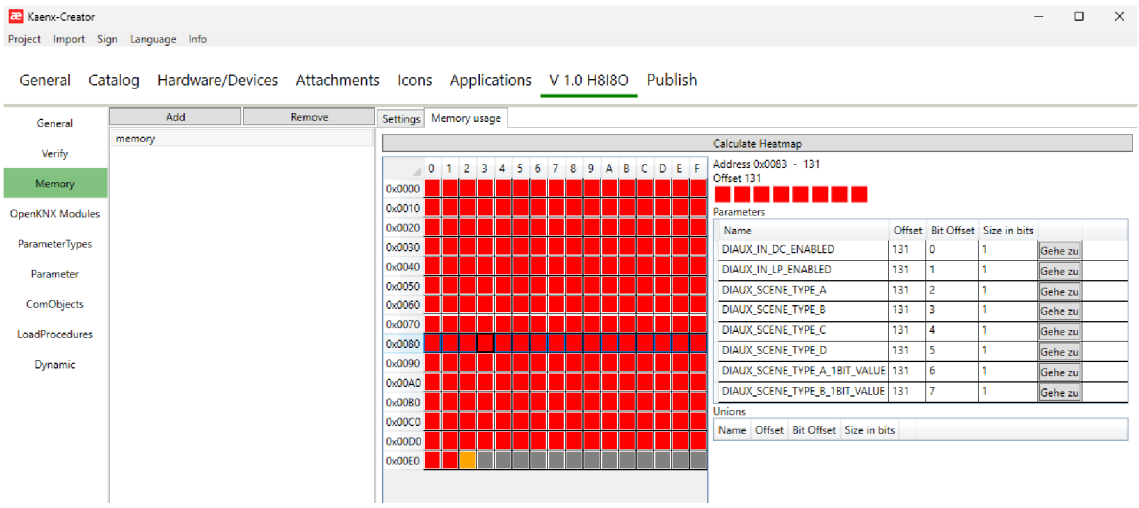
Po nastavení základních parametrů aplikace bylo možné vytvořit oddíl paměti do které budou ukládány parametry aplikace. Proto byla vytvořena

paměť s názvem *Memory*. Pro tuto paměť bylo zapnuto hned několik funkcionalit, viz. seznam níže:

- *Calculate size automatically*
- *Add size in LoadProcedure*
- *Order Parameters/Unions/Modules automatically*
- *Order automatically even if they have an address*

První funkcionalita zajistí, že velikost paměti bude automaticky vypočítána na základě parametrů v ní uložených a velikosti jejich typů. Následující funkcionalita zajistí, že do popisu procedury, která slouží k nahrání parametrů na desku *H8I8O*, bude automaticky přidaná velikost paměti. Následující funkcionalita je zároveň jeden z hlavních důvodů pro výběr tohoto nástroje, viz. sekce. 8.2.3. Tato funkcionalita tedy zajistí zarovnání parametrů do jednotlivých bajtů paměti, jak je možné vidět v obr. 27. Parametry zarovnané ve vybraném bajtu paměti jsou zobrazeny v pravé části obrázku. Poslední funkcionalita pouze vynutí funkcionalitu předchozí i za podmínky, že parametru byla adresa již přidělena jiným způsobem.

Dále byl nastaven parametr *Offset* na hodnotu 0, čímž paměť bude začínat od nuly.



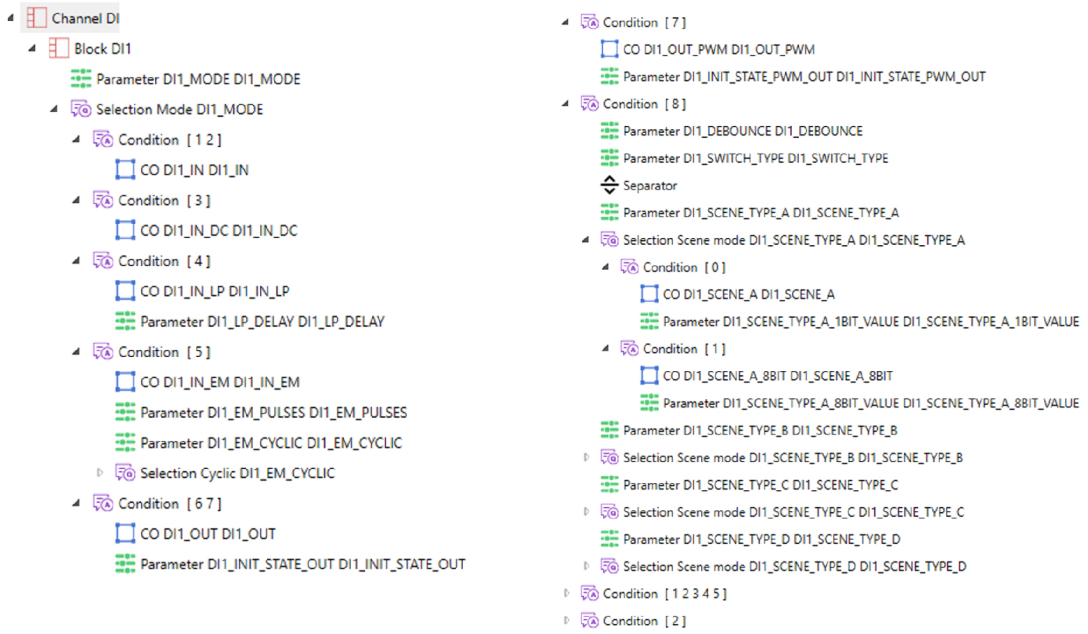
Obrázek 27: Paměť aplikace v *Kaenx-Creator*. Zdroj: [Autor]

Jak již bylo zmíněno v sekci 8.2.3, *Kaenx-Creator* umí na základě této paměti krom *knxprod* souboru vygenerovat i *.h* soubor. Zarovnání para-

metrů v obou souborech tedy odpovídá této paměti. Po přidání parametrů, *Kaenx-Creator* umožní vypočítat mapu paměti (barevná pole v obrázku 27), pomocí tlačítka *Calculate Heatmap*. Tato akce vynutí přerovnání parametrů do bajtů paměti.

9.1.4 Přidání parametrů, komunikačních objektů a rozhraní do *Kaenx-Creator* projektu

Po přidání parametrů, komunikačních objektů a prvků uživatelského rozhraní, které jsou v kontextu *Kaenx-Creator* nazývány *Dynamics*, pro jediný pin desky *H8I8O* bylo rozhodnuto, že realizace zbylých 17 pinů je poměrně náročný úkol, při jehož tvoření je obrovský prostor pro chyby. Odhadovaný počet parametrů je v řádu stovek a stejně tak tomu je i v případě komunikačních objektů. Dále by byla problematická i replikace uživatelského rozhraní jednoho pinu, jehož ne zcela rozevřená struktura je možná vidět v obr. 28.



Obrázek 28: Ukázka uživatelského rozhraní v *Kaenx-Creator*. Zdroj: [Autor]

Na základě těchto skutečností bylo rozhodnuto, že parametry, komunikační rozhraní i uživatelské rozhraní bude do stávajícího projektu dogenerováno. Tato možnost se jevila jako poměrně jednodušší řešení i vzhledem k tomu, že soubory projektů programu *Kaenx-Creator* mají sice koncovku *.ae-manu*, nicméně se jedná o obyčejný JSON (JavaScript Object Notation) soubor.

K tomuto účelu byl sepsán script, která zbylý obsah dogeneruje. Tento script byl sepsán v programovacím jazyce *Python* za použití knihovny *Pydantic*, která sloužila v případě tohoto scriptu k reprezentování jednotlivých entit, které byly následně jednodušeji serializovatelné do JSON formátu.

Krom entit rozhraní, byly definovány následující entity:

- *ParameterType* (*ParameterTypes* a *ParameterTypeEnum*)
- *Parameter* (a *ParameterRef*)
- *DataPointType* (a *DataPointSubType*)
- *ComObject* (a *ComObjectRef*)

Entita *ParameterType* slouží k specifikování typu parametru. Tyto typy mohou být například *string*, *uint*, *int*, *float* a *enum*. Tato entita slouží k upřesnění rozsahů parametru v případě číselných hodnot, nebo k definování možností enumerace pomocí entity *ParameterTypeEnum*.

Následující entita *Parameter* reprezentuje samotný parametr. Tato entita musí mít krom jiného i přiřazený typ pomocí entity předchozí. Dále tyto entity musí mít definovanou paměť, pro jejich uložení. Zde byla použita paměť definovaná v sekci 9.1.3. Tato entita má také definovanou metodu, která umožní generovat entitu *ParameterRef* na základě svých parametrů.

Entity *DataPointType* a *DataPointSubType* reprezentují data pointy viz. sekce 2.4.2. Respektive entita *DataPointType* reprezentuje hlavní skupinu a *DataPointSubType* podskupinu.

Poslední entitou je *ComObject*. Tato entita reprezentuje komunikační objekty a podobně jako u entity *Parameter*, umožňuje i tato entita generovat entitu *ComObjectRef* na základě svých parametrů. Dále tato entita musí mít přiřazený typ pomocí entity *DataPointType* a *DataPointSubType*.

V níže uvedeném seznamu jsou zachyceny všechny entity uživatelského rozhraní v programu *Kaenx-Creator*:

- *DynMain*
- *DynChannelIndependent*
- *DynChannel*
- *DynBlock*
- *DynSeparator* (a *DynSeparatorHint*)
- *DynChoose*

- *DynWhen*
- *DynParameter*
- *DynComObject*

Většina entit uživatelského rozhraní umožňuje vnořit jiné entity do sebe. Díky tomu lze tvořit stromovou strukturu, viz, obr. 28. Tato vlastnost je u těchto entit zprostředkována pomocí funkce *extend_items*. Tzn. tvorba uživatelského rozhraní byla uskutečněna vkládáním těchto entit viz. obr. 29.

```
channel.extend_items(DynBlock.from_args(io_name).extend_items(
    DynParameter.from_parameter(io_mode_param),
    DynChoose.from_param('Mode', io_mode_param).extend_items(
        DynWhen.from_condition('1 2').extend_items(
            DynComObject.from_com_object(in_co)
        ),
        DynWhen.from_condition('3').extend_items(
            DynComObject.from_com_object(in_dc_co)
        ),
        DynWhen.from_condition('4').extend_items(
            DynComObject.from_com_object(in_lp_co),
            DynParameter.from_parameter(long_press_delay_param)
        ),
```

Obrázek 29: Ukázka scriptu generujícího uživatelské rozhraní pro *Kaenx-Creator*. Zdroj: [Autor]

Entita *DynMain* tvoří základní entitu rozhraní. Tato entita musí být vždy jen jedna a všechny ostatní entity musí být vnořeny do ní.

Entity *DynChannelIndependent* a *DynChannel* reprezentují záložky uživatelského rozhraní s tím, že *DynChannelIndependent* je hlavní záložka, která je vždy zpřístupněna po otevření rozhraní a podobně jako *DynMain* může být jen jedna. Z tohoto důvody byla využita pro obecné informace a nastavení desky. Zatím co entita *DynChannel* může být označena vícekrát. Proto byla právě tato komponenta využita pro oddělení nastavení jednotlivých pinů desky.

Entita *DynBlock* je využívána k logickému oddělení jednotlivých prvků uživatelského rozhraní.

Entita *DynSeparator* má hned několik využití. Prvním je tvoření nadpisů. Pokud je této entitě přiřazena entita *DynSeparatorHint.HEADLINE*

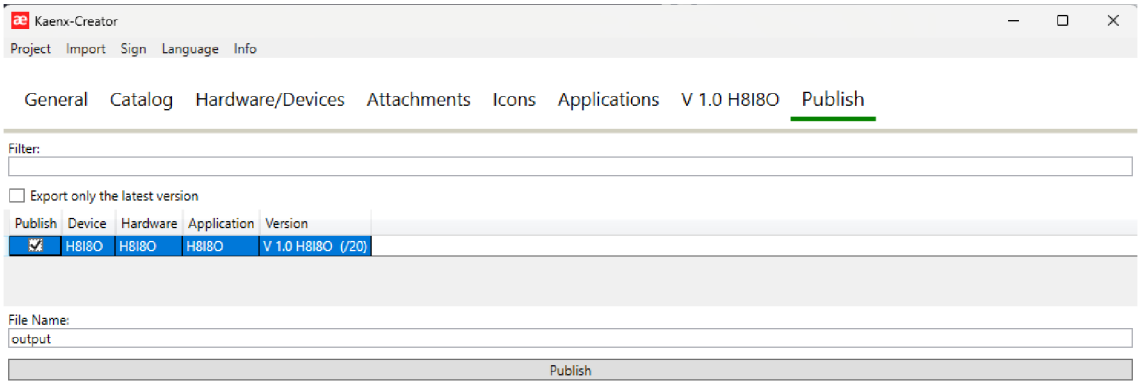
a text, bude tato entita reprezentována jako nadpis. Pokud jí bude přiřazena entita *DynSeparatorHint.INFO*, bude tato entita reprezentována jako informativné okno. Pokud jí bude přiřazena entita *DynSeparatorHint.ERROR*, bude tato entita reprezentována jako chybové okno. Dále podporuje také variantu s *DynSeparatorHint.RULER*, která zobrazuje tuto entitu jako horizontální separátor v podobě přímky. Pokud se přiřazení entity vynechá, nebo je přiřazena entita *DynSeparatorHint.NONE* a také text, bude tato entita reprezentována pouze jako čistý text.

Entity *DynChoose* a *DynWhen* jsou v podstatě používány jako klíčové slovo *switch* a klíčoví slovo *case*, jejichž význam můžeme znát ze *switch-case* konstrukce například v jazyce *C++*. Tzn. entita *DynChoose* má přiřazenou entitu *DynParameter* a na základě její hodnoty je možné uživatelské rozhraní větvit pomocí entity *DynWhen*, která má zase přiřazenou podmínku vyhodnocující parametr přiřazený *DynChoose*.

Zbývají už jen entity *DynParameter* a *DynComObject*, těmto entitám je analogicky přiřazena jedna z entit *Parameter* a *ComObject*. A na základě jejich reference reprezentují přidělené entity v rámci uživatelského rozhraní.

9.1.5 Generování *knxprod* souboru pomocí *Kaenx-Creator*

Samotný script *generate.py*, viz přílohy tedy doplnil vytvořenou aplikaci o všechny v něm definované parametry a uložil takto doplněnou kopii do vlastního souboru. Tento dogenerovaný soubor byl následně otevřen pomocí *Kaenx-Creator*. Dále byly v záložce aplikace pomocí tlačítka *Calculate Heatmap* (viz. sekce 9.1.3) adekvátně rozloženy všechny dogenerované parametry. Posledním krokem bylo tedy vygenerování samotného *knxprod* souboru. Tento proces je zachycen v obrázku 30.



Obrázek 30: Generování *knxprod* souboru pomocí *Kaenx-Creator*. Zdroj: [Autor]

9.2 Implementace softwaru desky

Po úspěšné realizaci aplikace pro ETS přišla řada na software desky jako takový. V návrhu bylo rozmyšleno několik základních myšlenek, kterými se tato sekce bude řídit.

9.2.1 Založení a konfigurace *PlatformIO* projektu

Jak již bylo popsáno v sekci 7.2, pro programování desky byl zvolen ekosystém *PlatformIO* a framework Arduino. *PlatformIO* podporuje několik tzv. *code editorů*, nebo IDE (Integrated Development Environment). Mezi tato vývojová prostředí patří například *Visual Studio Code* od společnosti *Microsoft*, nebo *CLion* od společnosti *JetBrains*. V případě tohoto projektu bylo zvoleno *Visual Studio Code*, pomocí kterého byl projekt založen. Samotný projekt má následující strukturu:

- *include/* - Hlavičkové soubory (*.h*)
- *lib/* - Knihovny
- *src/* - Zdrojové soubory (*.c*, *.ino* nebo *.cpp*)
- *test/* - Případné testy
- *platformio.ini* - Konfigurační soubor projektu, viz. obr. 31

Po založení projektu, které bylo poměrně jednoduché, přišla řada na jeho konfiguraci. Celá konfigurace je zachycena v obr. 31. První sekce konfigurace se zabývá platformou, deskou a MCU. V případě desky *H8I8O* se tedy jedná o platformu *ststm32*, generickou desku a z pohledu *PlatformIO* o MCU definované jako *stm32f103cbt6*. Jak již bylo zmíněno v sekci 6.1, MCU desky je ve skutečnosti *GD32F103CBT6*, nicméně tyto dva MCU jsou kompatibilní i přesto, že jsou některé jejich parametry rozdílné. Z tohoto důvodu je dále upřesněna frekvence MCU, která je zde nastavena na 108 MHz, podle specifikace *GD32F103CBT6*, viz. sekce 6.1.

V další sekci konfigurace byl nastaven framework na *arduino*. Dále také *upload_protocol* a *debug_tool* na hodnotu *stlink*, vzhledem k použitému nástroji pro programování desky, viz. sekce 7.2. Následně byl nastaven seznam potřebných knihoven pomocí parametru *lib_deps*, viz. seznam níže:

- *SPI* - Knihovna pro SPI, kterou vyžaduje knihovna *knx*

- *EEPROM* - Knihovna která umožní zapisovat to flash paměti MCU, vyžadovaná knihovnou *knx*
- *knx* - Samotná knihovna, tvořící API KNX.
- *AceButton* - Knihovna zpracovávající události spínačů, viz. sekce 8.3

Poslední sekce konfigurace se zabývá příznaky použitými při sestavování softwaru. Tyto příznaky jsou následně použity podobně jako definice v *C* nebo *C++*. Příznaky definované v konfiguraci nastavují několik věcí, první je zapnutí hardwarového UART (*serial*) rozhraní a nastavení jeho pinů. Dále jsou pak nastaveny piny programovacího tlačítka a kontrolky. Toto je nastaveno viz. tabulka 10. Dále je nastavena maska, která udává druh přenosového média. Poslední příznaky vypnout debugovací výstup knihovny *knx* a zapnout upozornění na definované příznaky, které jsou neznámé.

platformio.ini

```
[env:h8i8o]
platform = ststm32
board = genericSTM32F103CB
board_build.mcu = stm32f103cbt6
board_build.f_cpu = 108000000L

framework = arduino
upload_protocol = stlink
debug_tool = stlink
lib_deps =
    SPI # Required by KNX library
    EEPROM # Required by KNX library
    https://github.com/thelsing/knx # KNX library
    https://github.com/bxparks/AceButton # AceButton library
build_flags =
    -DENABLE_HWSERIAL1
    -DPIN_SERIAL1_TX=PA9
    -DPIN_SERIAL1_RX=PA10
    -DKNX_SERIAL=Serial1
    -DKNX_BUTTON=PA11
    -DKNX_LED=PA12
    -DMASK_VERSION=0x07B0
    -DKNX_NO_PRINT
    -Wno-unknown-pragmas
```

Obrázek 31: Konfigurace *PlatformIO* projektu. Zdroj: [Autor]

9.2.2 Instalace knihoven projektu

V předchozí sekci týkající se konfigurace projektu, byl zmíněn seznam knihoven. První dvě knihovny jsou v podstatě součástí Arduino frameworku. Z toho důvodu je jejich instalace automatická a stačí je jen zmínit v tomto seznamu.

Co se týče knihovny *knx*, tak tato knihovna je sice v registrech *PlatformIO*, nicméně se zde projevil problém se zastaralou verzí knihovny, který byl popsán v sekci 7.2. Z tohoto důvodu byla tato knihovna přidána jako odkaz na Git repozitář knihovny. To samé platí pro knihovnu *AceButton*.

9.2.3 Komponenty *PlatformIO* projektu

Vzhledem k rozsáhlosti projektu byl projekt rozdělen do jednotlivých komponent. Seznam těchto komponent je uveden níže:

- *main*
- *board_definition*
- *board_defaults*
- *board_io*
- *board_ios*
- *knx_init*
- *knx_actions*
- *energy_meter*
- *knx_params_cos*
- *knx_param_types*
- ✸ *knxprod_AF01_10*
- ✸ *generated*
- *knx_utils*

První komponentou je *main*. Jak již název napovídá, jedná se o hlavní soubor, kde jsou definovány funkce *void setup()* a *void loop()*. Tyto funkce jsou součástí Arduino frameworku, kdy *setup()* je proveden jednou při inicializaci MCU a slouží čistě k inicializaci, zatím co *loop()* je prováděn cyklicky a tím se stará o postupné vykonávání hlavních funkcionalit programu.

Komponenta *board_definition* dále definuje piny desky *H8I8O*. Pinů DI a DO byly následně seskupeny do polí *inputPins outputPins* a *ioPins* aby bylo možné se na ně odkazovat pomocí indexů. Tato definice je založena na tabulce 10 a je zobrazena v obr. 32.

```
// Inputs
#define DI1 PA15
#define DI2 PB3
#define DI3 PB4
#define DI4 PB5
#define DI5 PB6
#define DI6 PB7
#define DI7 PB8
#define DI8 PB9
#define DIAUX PC13

#define NUM_INP 9
const uint8_t inputPins[NUM_INP] = {···
```

```
// Outputs
#define DO1 PA0
#define DO2 PA2
#define DO3 PA1
#define DO4 PA3
#define DO5 PA4
#define DO6 PA5
#define DO7 PA6
#define DO8 PA7
#define DOAUX PB0

#define NUM_OUT 9
const uint8_t outputPins[NUM_OUT] = {···
```

```
// Inputs & Outputs combined
#define NUM_IOS (NUM_INP + NUM_OUT)
const uint8_t ioPins[NUM_IOS] = {···

// Button, LED, save
#define RESET_PIN PA8 // Active low
#define BUTTON_PIN PA11
#define LED_PIN PA12
#define SAVE_PIN PB15 // Active low
```

Obrázek 32: Komponenta *board_definition*. Zdroj: [Autor]

Komponenta *board_defaults* obsahuje pouze definice základních hodnot parametrů. Tyto hodnoty jsou použity při inicializaci zařízení a odpovídají základním hodnotám parametrů v *knxprod* souboru.

Čtvrtá a pátá komponenta, tedy *board_io* a *board_ios* zajišťují definici třídy *KnxBoardIO*. Dále také definují funkci *void initIos()*, která inicializuje pole těchto objektů na základě definice desky a *gettery*, které tyto objekty vracejí. Následně je také definována funkce *void loopIos()*, která iterativně provolává funkce *loop()* jednotlivých objektů *KnxBoardIO* v poli těchto objektů. Jako poslední je zde definována funkce *KnxRgbIos getRgbIos()*, která vrazí *struct* reprezentující seskupení pinů vyhrazených pro RGB.

Komponenty *knx_init* (a *knx_actions*) se stará o nastavení objektů třídy *KnxBoardIO* dle toho jaká funkcionalita tohoto piny byla nastavena prostřednictvím ETS. A dále přiřazením odpovídajících akcí definovaných v komponentě *knx_actions*.

Komponenta *energy_meter* obsahuje třídu *KnxEnergyMeter*, která je zodpovědná za počítání pulzů elektroměru a následný dopočet energie na základě těchto pulzů. Objekt této třídy je vždy asociován s objektem třídy *KnxBoardIO*, pokud je funkcionalita elektroměru zvolena v ETS.

Následující komponenty *knx_params_cos*, *knx_param_types*, *generated* a *knxprod_AF01_10* zajišťují definice pro získávání parametrů a komunikačních objektů. Respektive komponenta *knx_params_cos* zastřešuje celý tento úkol, zatím co komponenta *knx_param_types* definuje typy parametrů (enumerace odpovídající *knxprod* souboru). Komponenty v seznamu označené odrážkou „✸“ jsou komponenty generované. Komponenta *knxprod_AF01_10* byla vygenerována programem *Kaenx-Creator* při generování *knxprod* souboru. Zatím co komponenta *generated*, byla vygenerována pomocí scriptu *generate.py* a jejím úkolem je na získávání parametrů, nebo komunikačních objektů pomocí definic v komponentě *knxprod_AF01_10*.

Poslední komponenta *knx_utils* slouží pouze k definici pomocných funkcí. Mezi tyto funkce patří například funkce *uint8_t getNthByte(uint32_t number, uint8_t n)* a *void setNthByte(uint32_t *number, uint8_t n, uint8_t value)*, které slouží bitovou manipulaci čísel typu *uint32_t*. Tyto funkce jsou poměrně často využívány v komponentách *knx_init* a *knx_actions*.

9.2.4 Implementace třídy *KnxBoardIO*

Jak již bylo zmíněno třída *KnxBoardIO* má za úkol zastřešit veškeré atributy a funkce, které se týkají jednotlivých pinů desky a to včetně případného objektu třídy *AceButton*, nebo *EnergyMeter*.

Základní funkce této třídy jsou vyjmenivány v seznamu níže:

- *uint8_t getIndex()*
- *uint8_t getPin()*
- *void setupAsInput()*
- *void setupAsOutput()*
- *bool read()*
- *void writeDigital(bool value)*
- *void writeAnalog(uint8_t value)*
- *io_mode_t getMode()*
- *void setMode(io_mode_t mode_)*

První funkce slouží k získání indexu pinu. Tento index je vždy shodný s pozicí objektu *KnxBoardIO* v poli těchto objektů definovaném v komponentě *board_ios* a slouží k jednoznačnému určení o jaký pin se jedná.

Následující funkce *uint8_t getPin()* umožňuje získat číslo pinu z pohledu MCU. Tato funkcionalita je umožněna na základě pole všech pinů definovaném v komponentě *board_definition*. Tato funkce podstatně ulehčila práci, vzhledem k tomu, že tato informace je podstatná pokaždé, když je s pinem MCU jakkoli manipulováno.

Třetí a čtvrtá funkce *void setupAsInput()* a *void setupAsOutput()* umožňují nastavit pin MCU buďto jako vstup, nebo jako výstup. Toto je provedeno na základě čísla pinu získaného z předchozí funkce *uint8_t getPin()*.

Funkce *bool read()*, *void writeDigital(bool value)* a *void writeAnalog(uint8_t value)* umožňují čtení a zápis stavů pinů MCU. Funkce *bool read()* tedy umožňuje čtení stavu pinu, pokud je nastaven jako vstup pomocí funkce *void setupAsInput()*. Funkce *void writeDigital(bool value)* a *void writeAnalog(uint8_t value)* umožňují zápis stavu pinu MCU s tím, že první funkce zapisuje tzv. digitální hodnotu (1 nebo 0) a druhá hodnotu PWM.

Poslední dvě funkce (*getter* a *setter*) *io_mode_t getMode()* a *void setMode(io_mode_t mode_)* umožňují získat a nastavit funkci pinu, dle parametru nastaveného v ETS. Tato funkce je reprezentována hodnotou enumerace, která je definovaná v komponentě *knx_param_types*.

Krom těchto základních funkcí třída *KnxBoardIO* dále obsahuje i funkce, které slouží k interakci s instancí třídy *AceButton* ze stejnojmenné knihovny. Seznam těchto funkcí je uveden níže:

- *void setupAceButton(...)*
- *void setAceButtonDebounceDelay(uint16_t delay)*
- *void setAceButtonLongPressDelay(uint16_t delay)*
- *void enableAceButtonClick()*
- *void enableAceButtonDoubleClick()*
- *void enableAceButtonLongPress()*
- *bool supportsDcAndLp()*

První z těchto funkcí umožňuje inicializovat objekt *AceButton* pokud to daná funkcionalita pinu vyžaduje. Tzn. pokud je pin nastaven jako vstup.

Funkce *void setAceButtonDebounceDelay(uint16_t delay)* a *void setAceButtonLongPressDelay(uint16_t delay)* umožňují nastavit debounce prodlevu a trvání dlouhého stisku. Tyto funkce jsou opět použity v případě, že to daný funkcionalita pinu požaduje.

Následující tři funkce *void enableAceButtonClick() void enableAceButtonDoubleClick()* a *void enableAceButtonLongPress()* umožňují zapnout sledování událostí objektem *AceButton*. Tzn. sledování těchto událostí je opět zapnuto na základě funkcionality pinu nastavené prostřednictvím ETS.

Poslední funkce *bool supportsDcAndLp()* umožňuje zjistit, zdali aktuální instance *KnxBoardIO* podporuje akce na dvojklik a dlouhé kliknutí. Tato informace je odvozena od aktuální funkce pinu, získané pomocí *io_mode_t getMode()*.

Dále třída *KnxBoardIO* obsahuje i funkce týkající se třídy *EnergyMeter*, viz. seznam níže:

- *void setupEnergyMeter(...)*
- *void setEnergyMeterCallbackPerConsumed(float value)*
- *void setEnergyMeterCyclic(bool enabled)*
- *void setEnergyMeterCyclicDelay(uint16_t delay)*
- *void pulseEnergyMeter()*

Funkce *void setupEnergyMeter(...)* slouží k inicializaci objektu *EnergyMeter* obdobně jako tomu bylo u objektu *AceButton*.

Následující tři funkce slouží k nastavení atributů objektu *EnergyMeter*. Tyto atributy jsou nastaveny podle konfigurace ETS.

Poslední funkce *void pulseEnergyMeter()* slouží k přičtení 1 k počtu pulzů. Což může dle nastaveného způsobu zasílání způsobit i zaslání aktuální hodnoty přepočítané energie po sběrnici KNX.

Funkce uvedené v seznamu níže slouží pro usnadnění logiky funkcionality scén.

- *bool supportsScenes()*
- *uint32_t *getRecallValuePtr()*
- *bool shouldUpdateRecallValue(uint8_t i)*
- *void setShouldUpdateRecallValue(uint8_t i, bool value)*
- *bool getSceneEnabled()*
- *void toggleSceneEnabled()*

Funkce *uint32_t *getRecallValuePtr()* vrací ukazatel na tzv. *recall* hodnotu. Tato hodnota je čtena při inicializaci zařízení, nebo při změně její hodnoty ze sběrnice KNX. Dále je tato hodnota použita k navrácení stavu při odvolání scény.

Funkce *bool shouldUpdateRecallValue(uint8_t i)* a *void setShouldUpdateRecallValue(uint8_t i, bool value)* jsou použity pro logiku, která zajištuje, že *recall* hodnota není přepsána hodnotou scény.

Poslední funkce *bool getSceneEnabled()* a *void toggleSceneEnabled()* jsou využívány pro zajištění logiky, která se stará o uchování stavu dané scény, respektive jestli je scéna zapnuta, či ne.

Poslední funkcí této třídy je funkce *void loop()*. Tato funkce je provolávána prostřednictvím funkce *void loopIos()* uvnitř cyklicky volané funkce *void loop()* hlavní komponenty *main*. Tato funkce se stará o aktualizování a kontrolování vnitřních stavů objektu *KnxBoardIO*.

Krom funkcí je dále nutní zmínit, že tento objekt má veřejně dostupný atribut *value*, který má datový typ *uint32_t*. Tento atribut je využíván k uchování aktuální hodnoty dle zvolené funkcionality. Jeho bity jsou děleny do sekcí, které reprezentují jednu, či více různých hodnot.

9.2.5 Implementace komponenty *knx_init*

Jak již bylo zmíněno, komponenta *knx_init* se stará o nastavení instancí *KnxBoardIO* na základě konfigurace z ETS. Uvnitř této komponenty se nacházejí 3 funkce, viz. seznam níže:

- *void handleInputEvent(...)*
- *void handleSave()*
- *void initConfiguration()*

Funkce *void initConfiguration()* zprostředkovává nastavení jednotlivých instancí třídy *KnxBoardIO* dle zvolené funkcionality. Tento postup lze shrnout do následujících kroků.

Nejprve podle parametru, který udává zdali je zavolena funkcionalita RGB nastaví piny příslušných objektům *KnxBoardIO* jako výstup. Následně komunikačním objektům reprezentujícím barvu a stav (On/Off) nastaví příslušné akce a příslušné typy data pointů.

V dalším kroku tato funkce iterativně nastavuje příslušné funkcionality příslušným objektům *KnxBoardIO*. Tzn. pokud je například pro daný pin desky zvolena funkcionalita *Spínání pomocí tlačítka (s alternativními funkcemi)* (viz. sekce 8.1), je daný pin nastaven jako vstup, dále je inicializována instance třídy *AceButton*, následně je zapnuto hlídání událostí stisknutí a nastavena prodleva debounce dle parametru z ETS. Poté je první bajt atributu *value* nastaven na hodnotu dle parametru *init state* a následně je přiřazen odpovídajícímu komunikačnímu objektu odpovídající data point. Obdobně jsou realizovány i nastavení ostatních funkcionalit jednotlivých instancí *KnxBoardIO*.

Další funkcí je *void handleInputEvent(...)*. Tato funkce se stará o vyřizování událostí vyvolaných instancí *AceButton*. Tzn. na základě vyvolané události a funkcionality přiřazené k dané instanci *KnxBoardIO*, zavolá adekvátní akci z komponenty *knx_actions*.

Poslední funkce *void handleSave()* je určena k ukládání aktuálních hodnot instancí *KnxBoardIO* v případě, že by tyto hodnoty měly být zachovány. Tato funkce je vyvolána za pomocí systémového přerušení, které je nastaveno na pin MCU označovaný jako *SAVE PIN* v tabulce 10 a na sestupnou hranu signálu.

9.2.6 Implementace akcí

Jak již bylo zmíněno, v komponentě *knx_actions* jsou definované jednotlivé akce, či *callbacky*, které reagují na různé události v softwaru desky. Tyto události lze rozdělit do dvou skupin. První skupiny řeší události vyvolané akcemi ze strany desky. Mezi ně patří například události vyvolané vstupními piny, nebo časované události. Hlavní úkol těchto událostí je aktualizovat stavy atributu *value* v instancích *KnxBoardIO* a na základě toho zasílat aktualizované hodnoty pomocí komunikačních objektů na sběrnici KNX. Proto je z pravidla argumentem těchto akcí daný objekt *KnxBoardIO*. Seznam těchto akcí, viz. níže:

- *void actionSwitch(...)*
- *void actionButton(...)*
- *void actionButtonDc(...)*
- *void actionButtonLp(...)*
- *void actionEnergyMeter(...)*
- *void actionSceneIn(...)*

Druhou skupinou těchto akcí jsou akce, které mají opačný úkol. Tzn. přijímat hodnoty z komunikačních objektů, na základě kterých aktualizují hodnoty *value* příslušných komunikačních objektů a dále nastaví piny MCU dle dané hodnoty. Argumentem těchto funkcí je vždy komunikační objekt zachycený ze sběrnice. Na základě jeho čísla je umožněno získat odpovídající instanci *KnxBoardIO*, pomocí *getteru* z komponenty *knx_params_cos*. Seznam těchto akcí je uveden níže:

- *void actionRGB(...)*
- *void actionRGBOnOff(...)*
- *void actionOut(...)*
- *void actionOutPwm(...)*
- *void actionOutOnOffPwm(...)*
- *void actionSceneA(...)*
- ...
- *void actionSceneD(...)*
- *void actionScene8BitA(...)*
- ...
- *void actionScene8BitD(...)*

Akce jsou posledním důležitým, či zajímavým prvkem, který je dobré zmínit. Detailnější pohled na problematiku implementace je možné získat prohlédnutím příloh práce.

10 Testování

Veškeré funkcionality a akce byly testovány několika způsoby. Tyto způsoby jsou popsány v následujících sekcích práce.

10.1 Testování bez ETS

Testování jednotlivých funkcionalit bez ETS bylo prováděno poměrně jednoduchým způsobem. Namísto aby parametry, které by byly jinak čtené z konfigurace ETS, byly definovány pevně. Tzn. pokud byla testována funkcionalita *Spínání pomocí tlačítka (s alternativními funkcemi)*, byl parametr reprezentující tuto funkcionalitu pevně nastaven právě na ni a ostatní parametry dle požadavků testu.

Následně bylo sledováno buďto fyzické chování desky, nebo zasílání komunikačních objektů na sběrnici. Sledování komunikačních objektů bylo realizované pomocí *group monitoru*, viz. sekce 4.3.2.

10.2 Testování s ETS

Po dokončení testování bez ETS byly jednotlivé funkcionality testovány s ním. Tento způsob testování je výhodný v tom, že na rozdíl od předchozího kontroluje i správnost *knxprod* souboru.

Samotné testování probíhalo tak, že zařízení bylo nejprve nakonfigurováno pomocí ETS. Tzn byly nastaveny parametry dané funkcionality a přiřazeny komunikační objekty testovacím skupinám. K výsledek testování byl vyhodnocován na základě pozorování a diagnostiky ETS.

Tento způsob testování je poněkud zdlouhavý, vzhledem k tomu, že vyžaduje i nastavení prostřednictvím ETS. Nicméně na druhou stranu zaručeně ověří funkčnost jak softwaru zařízení, tak i softwaru aplikace ETS a proto se stal nedílnou součástí procesu testování.

10.3 Testování s jiným zařízením KNX

Tento druh testů je asi nejnáročnější, vzhledem k tomu, že krom samotného nastavení testovaného zařízení vyžaduje i nastavení druhého zařízení.

Na druhou stranu toto testování zaručilo absolutní kompatibilitu v rámci KNX a to i s jinými zařízeními. Tzn. jedná se v podstatě o tzv. *end-to-end* testování.

Proces je podobný jako u předchozího způsobu testování, nicméně je tedy konfigurováno i druhé zařízení.

11 Závěr

Cílem této práce bylo prozkoumat možnosti KNX a způsoby implementace KNX zařízení za pomoci Arduino knihoven. Tento cíl by nebylo možné uskutečnit, bez bližšího porozumění systému KNX. Proto se počáteční kapitoly práce zabývaly principy a technickými aspekty celého systému.

Dále byla zkoumána deska *H8I8O*, která obsahuje potřebný hardware k realizaci KNX zařízení. Krom technických aspektů a hardwarových možností byla zkoumána i stránka softwarová. Tzn. zdali je možné desku programovat s pomocí Arduino frameworku. A jaké jsou případné možnosti z pohledu knihoven tohoto frameworku.

Při implementaci samotného softwaru desky byla využita knihovna *knx* od autora Thomase Kunze. Tato knihovna se ukázala jako opravdu dobře zpracovaná a realizaci KNX zařízení značně ulehčuje.

Pro realizaci softwaru aplikace byl využit program *Kaenx-Creator*, který práci taktéž ulehčuje. Nicméně je nutné podotknout, že některé jeho funkcionality ještě nejsou tak úplně doladěny.

Co se týče plánovaných funkcionalit, povedlo se realizovat všechny. Nicméně při realizaci funkcionality *Ovládání scén (4 funkce)* byla odhalena limitace desky *H8I8O*, respektive limitace MCU *GD32F103CBT6*.

Tato limitace spočívá ve velikosti EEPROM paměti MCU (nezaměňovat s flash pamětí, i když je jí v případě desky *H8I8O* emulována). Tato paměť je využívána k ne-volatilnímu ukládání parametrů konfigurovaných z ETS a relací mezi komunikačními objekty. V případě všech funkcionalit implementovaných na desce *H8I8O*, jsou všechny tato data poměrně rozsáhlá a velikost 1 KB EEPROM paměti bohužel nestačila. Z tohoto důvodu byla funkcionalita *Ovládání scén (4 funkce)* implementována pouze pro DI piny zařízení, díky čemuž byla tato realizace značně složitější.

Alternativně by tato limitace šla řešit pomocí vlastní definice platformy uvnitř *knx* knihovny a následné adaptace EEPROM knihovny. Toto bylo testováno zvýšením EEPROM paměti pomocí příznaků při sestavení programu na 4 KB a následnou úpravou všech okolností, bohužel toto testování nebylo úspěšné. Proto tato varianta řešení problematiky s nedostatečnou ne-volatilní pamětí zůstává otevřená.

Nicméně i přes tento zádrhel byla realizace úspěšně dokončena. Tento fakt byl následně i potvrzen při testování všech aspektů vytvořeného softwaru.

Tato práce tedy přinesla vhled do komplexní problematiky systému KNX se zaměřením na tvorbu softwaru KNX zařízení. Tento software je plně konfigurovatelný tak ‚jak by každý software KNX zařízení měl být. Díky tomu může být práce využita i jako návod při tvorbě softwaru zařízení KNX.

12 Seznam použité literatury

1. KNX ASSOCIATION. *KNX Základy*. 2013. Dostupné také z: https://www.knx.org/wAssets/docs/downloads/Marketing/Flyers/KNX-Basics/KNX-Basics_cz.pdf.
2. RUTA, Michele; SCIOSCIA, Floriano; LOSETO, Giuseppe; DI SCIASCIO, Eugenio. KNX: A Worldwide Standard Protocol for Home and Building Automation: State of the Art and Perspectives. In: ZURAWSKI, Richard (ed.). *Industrial Communication Technology Handbook* [online]. 2. vyd. CRC Press, 2017, s. 58–1–58–19 [cit. 2023-06-15]. ISBN 978-1-315-21548-8. Dostupné z DOI: 10.1201/b17365-60.
3. KNX ASSOCIATION. *KNX Standard: 3.1.1 KNX System Specifications, Architecture*. 2013.
4. RESIDIT S.R.O. *Historie systému KNX (1. část) - Časopis Elektro - Odborné časopisy* [Odbornecasopisy.cz] [online]. [cit. 2023-06-05]. Dostupné z: http://www.odbornecasopisy.cz/elektro/clanek/historie-systemu-knx-1-cast--2728.
5. KNX ASSOCIATION. *The legacy of KNX–KNX Association* [online]. [cit. 2023-06-12]. Dostupné z: https://www.knx.org/knx-en/for-professionals/What-is-KNX/KNX-History/.
6. KNX ASSOCIATION. *KNX Basic course*. KNX Association, 2013. Dostupné také z: http://knx.com.ua/attachments/article/132/KNX-basic_course_full.pdf.
7. PETERKA, Jiří. *Jiří Peterka: Referenční model ISO/OSI - sedm vrstev* [online]. 1992. [cit. 2023-06-05]. Dostupné z: https://www.earchiv.cz/a92/a213c110.php3.
8. KNX ASSOCIATION. *KNX Standard: 3.3.7 System Specifications, Communication , Application Layer*. 2013.
9. KNX ASSOCIATION. *KNX Standard: 3.3.4 System Specifications, Communication , Transport Layer*. 2013.
10. KNX ASSOCIATION. *KNX Standard: 3.3.3 System Specifications, Communication , Network Layer*. 2013.
11. KNX ASSOCIATION. *KNX Standard: 3.3.2 System Specifications, Communication , Data Link Layer General*. 2013.
12. KNX ASSOCIATION. *KNX Standard: 3.3.1 System Specifications, Communication , Physical Layer General*. 2013.
13. KNX ASSOCIATION. *KNX Standard: 3.2.2 System Specifications, Communication Media, Twisted Pair 1*. 2013.
14. PARTHOENS, Christophe. *Flags* [KNX Association] [online]. 2017-07-27. [cit. 2023-06-29]. Dostupné z: https://support.knx.org/hc/en-us/articles/115003188089-Flags.
15. KNX ASSOCIATION. *KNX Standard: 3.7.2 System Specifications, Interworking, Datapoint Types*. 2013.

16. KNX ASSOCIATION. *KNX Standard: 3.2.5 System Specifications, Communication Media, Radio Frequency*. 2013.
17. KNX ASSOCIATION. *KNX Standard: 3.2.6 System Specifications, Communication Media, KNX IP*. 2013.
18. KNX ASSOCIATION. *ETS6 Hlavní vlastnosti ETS6*. 2021. Dostupné také z: https://knxcz.cz/images/files/ETS6_Features_2021_CZ.pdf.
19. KNX ASSOCIATION. *ETS6 A KNX VIRTUAL PRO TESTOVÁNÍ A VÝUKU KNX*. 2021. Dostupné také z: https://knxcz.cz/images/files/ETS6%20and%20virtual%202021%20CZ.pdf.
20. HÄNEL, André. *ETS Catalogs* [KNX Association] [online]. 2020-03-10. [cit. 2023-06-06]. Dostupné z: https://support.knx.org/hc/en-us/articles/360011690620-Catalogs.
21. UNAL, Ufuk. *ETS Individual address* [KNX Association] [online]. 2023-01-30. [cit. 2023-06-05]. Dostupné z: https://support.knx.org/hc/en-us/articles/360018775719-Individual-address-.
22. UNAL, Ufuk. *Bus Monitor, Group Monitor and ETS Bus Activity Monitor* [KNX Association] [online]. 2023-02-08. [cit. 2023-06-06]. Dostupné z: https://support.knx.org/hc/en-us/articles/360018712880-Bus-Monitor-Group-Monitor-and-ETS-Bus-Activity-Monitor.
23. URLICHS, Matthias; ET AL. *knxd* [knxd] [online]. 2023-06-07. [cit. 2023-06-07]. Dostupné z: https://github.com/knxd/knxd.
24. RASPBERRY PI FOUNDATION. *What is a Raspberry Pi?* [Raspberry Pi Foundation] [online]. [cit. 2023-07-02]. Dostupné z: https://www.raspberrypi.org/help/what-is-a-raspberry-pi/.
25. ANON. *KNX*输入输出模块说明书 *H8I8O V0.3*. [B.r.]. Dostupné také z: https://en.webshare.cz/#/file/oJc05nklKG/knx-h8i8o-v0-3-pdf.
26. GIGADEVICE SEMICONDUCTOR INC. *PRODUCT SELECTION GUIDE GD32 MCU*. 2018. Dostupné také z: https://datasheet.octopart.com/GD32F103CBT6-GigaDevice-datasheet-120073833.pdf.
27. GIGADEVICE SEMICONDUCTOR INC. *GD32F103xx ARM® Cortex™-M3 32-bit MCU*. 2013. Dostupné také z: https://datasheet.octopart.com/GD32F103CBT6-GigaDevice-datasheet-103065885.pdf.
28. SIEMENS. *SIEMENS KNX EIB TP-UART 2-IC*. 2012. Dostupné také z: https://sid.siemens.com/v/u/A6V11933793.
29. STMICROELECTRONICS. *ST-LINK/V2 in-circuit debugger/programmer for STM8 and STM32*. 2012. Dostupné také z: https://docs.rs-online.com/679b/0900766b8148760b.pdf.
30. PLATFORMIO. *Professional collaborative platform for embedded development —PlatformIO latest documentation* [online]. 2014. [cit. 2023-08-03]. Dostupné z: https://docs.platformio.org/en/latest/.
31. SIEMENS. *SIEMENS Bus Coupling Unit (BTM) 5WG1117–2CB12*. 2020.
32. KUNZE, Thomas; ET AL. *Welcome to knx's documentation! —knx 1 documentation* [online]. 2019. [cit. 2023-07-31]. Dostupné z: https://knx.readthedocs.io/en/latest/.

33. PARK, Brian; ET AL. *AceButton* [online]. 2023. [cit. 2023-07-31]. Dostupné z: `https://github.com/bxparks/AceButton`.

13 Přílohy

Příloha 1
Obsah elektronické přílohy

Název souboru	**Popis souboru**
➥ knxprod	- Generátor *Kaenx-Creator* projektu
➥ models	- Modely generátoru *Kaenx-Creator* projektu
• H8I8O_V4.ae-manu	- Základní *Kaenx-Creator* projekt
• H8I8O_gen.ae-manu	- Dogenerovaný *Kaenx-Creator* projekt
• generate.py	- Generátor *Kaenx-Creator* projektu
• Atd.	
➥ resources	- Obrázky pro popis projektu
➥ include	- Zdrojové kódy projektu
➥ src	- Zdrojové kódy projektu
• README.md	- Popis projektu s odkazem na repozitář projektu
• firmware-blink.bin	- Software desky, který bliká s kontrolkou
• generate.py	- Generátor komponenty *generated*
• platformio.ini	- Konfigurace projektu
• Atd.	

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání diplomové práce

Autor: **Bc. David Tláskal**

Studium: I2100084

Studijní program: N1802 Aplikovaná informatika

Studijní obor: Aplikovaná informatika

Název diplomové práce: **Implementace KNX modulu pomocí Arduino knihoven**

Název diplomové práce AJ: KNX module implementation using Arduino libraries

Cíl, metody, literatura, předpoklady:

Cíl: Naportovat Arduino knihovnu thelsing/knx na desku H8I8O představující vstupně-výstupní modul pro sběrnici KNX pro řízení budov. Dále navrhnout a implementovat aplikační firmware demonstrující možnosti knihovny a modulu - tedy nezávisle konfigurovatelné vstupy a výstupy asociované s komunikačními objekty sběrnice KNX, parametrizovatelné standardním softwarem ETS.

Zadávající pracoviště: Katedra informatiky a kvantitativních metod,
Fakulta informatiky a managementu

Vedoucí práce: Ing. Pavel Kříž, Ph.D.

Datum zadání závěrečné práce: 26.1.2021