TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Komunikační driver pro integraci čidel SDI-12

Bakalářská práce

Studijní program: B2612 Elektrotechnika a informatika
Studijní obor: Elektronické informační a řídicí systémy

Autor práce: **Marek Hejduk**
Vedoucí práce: Ing. Daniel Kajzr
Ústav mechatroniky a technické informatiky

Liberec 2021

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání bakalářské práce

Komunikační driver pro integraci čidel SDI-12

Jméno a příjmení: **Marek Hejduk**
Osobní číslo: M17000036
Studijní program: B2612 Elektrotechnika a informatika
Studijní obor: Elektronické informační a řídicí systémy
Zadávající katedra: Ústav mechatroniky a technické informatiky
Akademický rok: **2020/2021**

Zásady pro vypracování:

1. Seznamte se s hardwarovými a softwarovými vlastnostmi sběrnice SDI-12 a porovnejte je s ostatními komunikačními sběrnicemi využívanými pro sběr dat ze senzorů v budovách i průmyslu. Uveďte výhody a nevýhody jednotlivých standardů.
2. Navrhněte a sestavte komunikační driver protokolu SDI-12 ve vývojovém prostředí pro systémy řízení technologií např. Merbon IDE. Řešení otestujte nejprve simulačně a následně na reálném zařízení a konkrétní aplikaci ve spolupráci s dodavatelem vývojového prostředí.
3. Na základě zkušeností z praktické části vyhodnoťte možnosti použití standardu SDI-12 pro další oblasti, jako je například řízení budov z hlediska technických vlastností a náročnosti na uvádění do provozu i servis.
4. Součástí návrhu komunikačního driveru bude dokumentace v českém a anglickém jazyce.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30–40 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] A Serial-Digital Interface Standard for Microprocessor-Based Sensors. Version 1.4', Prepared By SDI-12 Support Group (Technical Committee). USA., July 2019.

[2] MERZ, Hermann, HANSEMANN, Thomas a HÜBNER, Christof. Automatizované systémy budov: sdělovací systémy KNX/EIB, LON a BACnet. 1. vyd. Praha: Grada, 2008, 261 s. ISBN 978-80-247-2367-9.

[3] JOHN, Karl-Heinz a Michael TIEGEKAMP. IEC 611311-3: Programming Industrila Automation Systems [online]. New York: Springer, 2010 [cit. 2014-05-08]. 2. ISBN 978-3-642-12014-5. Dostupné z: doi: 10.1007/978-3-642-12015-2

Vedoucí práce:	Ing. Daniel Kajzr Ústav mechatroniky a technické informatiky
Datum zadání práce:	9. října 2020
Předpokládaný termín odevzdání:	17. května 2021

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 9. října 2020

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

15. května 2021 Marek Hejduk

Poděkování

Tímto bych rád poděkoval firmě Domat Control System, která mi poskytla přístřeší a propůjčila potřebný hardware, zejména pak panu Janu Vidimovi, který byl mojí velkou oporou. Dále moje poděkování patří Ing. Danielu Kajzrovi za odborné vedení mé práce, jeho vstřícnost a cenné rady při vypracování této práce.

Abstrakt

Cílem této práce je návrh komunikačního driveru pro standard SDI-12, který vznikl ve spolupráci se společností Domat Control System. Práce se skládá z teoretické a praktické části. V teoretické části je přiblížena teorie průmyslové komunikace. Ta vysvětluje pojmy a uvádí v praxi nejpoužívanější standardy. Ve druhé, praktické části je navržen měřicí algoritmus, který byl následně zrealizován ve vývojovém prostřední Merbon IDE. Použit byl programovací jazyk ST. Součástí praktické části je také test funkčnosti programu a následná HMI vizualizace měřených hodnot.

Výsledkem bakalářské práce je funkční blok, který znatelně ulehčuje uživateli práci a šetří čas při uvádění čidel pracujících s SDI-12 do provozu. Konkrétně si program již nalezl uplatnění v zemědělství.

Klíčová slova

SDI-12, sběrnice, standard, driver, PLC

Abstract

The focus of this thesis is the design of a communication driver for the SDI-12 standard in cooperation with the company Domat Control System. The first part deals with the theory of industrial communication, explains the concepts and the most commonly used standards. The second part contains the design of the driver, the development of the measurement algorithm and the actual writing of code in the development environment Merbon IDE with language ST. The practical part also includes a test of the functionality and HMI visualization of the measured values.

The result is a function block that significantly simplifies the customer's work and saves time when using SDI-12 sensors. Specifically, the program has already found application in agriculture.

Keywords

SDI-12, bus, standard, driver, PLC

Obsah

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

TUL – Technická univerzita v Liberci

FM – Fakulta mechatroniky, informatiky a mezioborových studií Technické univerzity v Liberci

BP – Bakalářská práce

SDI-12 – Serial Digital Interface at 1200 Baud

GND – Ground – nulový potenciál (označováno také jako uzemnění)

PLC – Programmable Logic Controller

RT – Runtime

ST – Structured Text

LD – Ladder Diagram

FBD – Function Block Diagram

FB – Function Block

CPU- Central Processor Unit

RAM – Random Acess Memory

EEPROM – Electricaly Erasable Programmable Read Only Memory

HMI – Human Machine Interface

SCADA – Supervisory Control and Data Acquisition

DR – Data Recorder

ASCII – American Standard Code for Information Interchange

DC – Direct Current

IP – Internet Protocol

TCP – Transmission Control Protocol

UDP – User Datagram Protocol

HTTP – Hypertext Transfer Protocol

Úvod

Komunikace mezi zařízeními je nezbytnou součástí pokroku dnešního průmyslu, někdy zvaného také jako čtvrtá průmyslová revoluce nebo Industry 4.0. Řízení technologických procesů, automatických výrobních linek, měřicích systémů, inteligentních budov, systémů vozidel, tam všude je zapotřebí vzájemné komunikace zařízení mezi sebou. Ať už se jedná o spojení fyzické či bezdrátové, každý typ komunikace se řídí tzv. standardem, který předepisuje všechny potřebné parametry, aby si zařízení mezi sebou dokázala „porozumět“. Jedním z těchto standardů je také, ne příliš známý, leč zajímavý, standard SDI-12. Používán je zejména v zemědělství pro sledování environmentálních veličin, jako je teplota, vlhkost půdy, pH půdy apod.

Cílem této práce je představení tohoto standardu, porovnání s dalšími v praxi běžně využívanými standardy a následný vývoj programu pro PLC. Ten bude sloužit pro zajištění komunikace a automatický sběr dat ze sensorů pracujících s SDI-12.

Program je vyvíjen pro konkrétní aplikaci, avšak pro lepší přehlednost a následně možnost použití i v jiné aplikaci, bude měřicí algoritmus zakomponován do funkčního bloku, který se stane součástí knihovny vývojového prostředí. Ten by měl potom uživateli znatelně zjednodušit obsluhu, jelikož knihovny programovatelných automatů prozatím nedisponují žádnými vhodnými funkcemi. Každý takový funkční blok by měl zajišťovat komunikaci právě s jedním čidlem. Uživatel si podle potřeby určí vstupní parametry a na výstupu FB získá měřené hodnoty, se kterými může dál pracovat.

1 Průmyslová komunikace

Průmyslová komunikace je velmi obsáhlý pojem, proto je důležité si uvést alespoň některé její základní myšlenky. Jejím hlavním cílem je v podstatě zajištění přenosu dat mezi zařízeními, a to zcela bez lidského zásahu. Každé komunikující zařízení, jako je počítač, terminál nebo sensor, je určitým způsobem propojeno s ostatními a společně tak vytváří síť. Aby bylo možné rozeznat, komu je zpráva určena a kdo je odesilatelem, má každé zařízení vlastní adresu. Tím je možné vytvářet plně automatizované systémy řízení.

Díky průmyslové komunikaci můžeme sbírat data, monitorovat stavy technologických procesů, výrobních linek, budov, venkovního prostředí apod. Zároveň je možné ze stejného místa tyto aplikace také řídit. To je možné nejen z místa pracoviště, ale díky síti internet, pomocí osobního počítače či mobilního telefonu, prakticky odkudkoliv. Možnost vzájemné komunikace výrazně zjednoduší kontrolu, umožní předcházení i následné řešení problémů.

Dnešní doba žádá propojení a integraci různých druhů systémů a zjištění jejich kompatibility a následné spolupráce. Každá komunikace se řídí určitým standardem, který detailně předepisuje její formu. Takových standardů již v dnešní době existuje mnoho. To má své výhody ale i nevýhody. Hlavní nevýhodou je právě obtížná kolaborace zařízení, pracujících s odlišnými standardy. V takovém případě bývá nutné použití určitého převodníku. Z uvedených informací je zřejmé, že jeden typ komunikace v průmyslu zásadně zjednoduší celou aplikaci. Avšak existují výjimky, kdy takové řešení není možné. Bylo by velmi praktické, kdyby se v budoucnu přešlo k použití jednoho univerzálního standardu (jako tomu bylo při rozšíření USB u počítačů a mobilních telefonů), jenže v tomto odvětví to není zcela možné. Každý standard má totiž jiné parametry a tím je vhodný pro jiný typ aplikace. V neposlední řadě hraje roli i konkurence [1].

1.1 Rozdělení komunikací a základní termíny

Průmyslová komunikace je termín poměrně obecný a její forma se může v mnoha ohledech lišit. Nyní je důležité uvést základní rozdělení a termíny, se kterými se můžeme setkat.

- Rozdělení komunikace podle směru toku dat
    - jednosměrná – **simplexní**
    - obousměrná – **duplexní**, která se dále dělí na:
        - **plně duplexní** – komunikace probíhá oběma směry najednou
        - **poloduplexní** – komunikace probíhá pouze v jednom směru

- Rozdělení podle způsobu přenosu dat
    - **sériový** – data jsou odesílána po jednom vodiči postupně za sebou
    - **paralelní** – data jsou posílána najednou pomocí mnoha vodičů, které se dělí na řídicí, adresové a datové

- Rozdělení podle pravidelnosti vysílání:
    - **synchronní** – jeden z vodičů je vyhrazen pro přenos hodinového signálu, aby zápis a čtení probíhaly ve stejných časových intervalech, tzn. synchronizovaně
    - **asynchronní** – synchronizace probíhá pomocí tzv. start bitu (vysílač i přijímač mají každý své hodiny)

- Rozdělení přenosu informace podle změny veličiny
    - **el. napětí** – jednodušší, lehce narušitelné
    - **el. proud** – složitější, odolnější proti rušení

- Rozdělení přenosu podle pořadí odesílaných bitů
    - od nejméně významného bitu **LSB** – Less Significant Bit (Ascending order)
    - od nejvýše významného bitu **HSB** – Higher Significant Bit (Descending order)

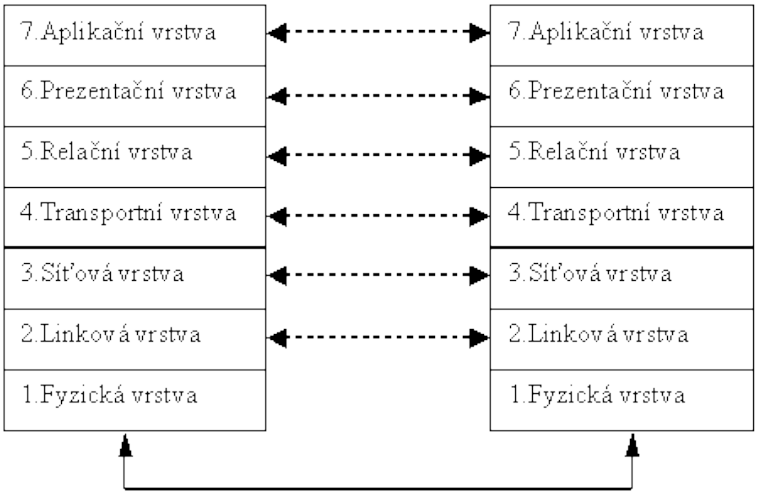
Pro jednoduchou detekci chyb, která dokáže zabránit ve zpracování chybné hodnoty, se používá tzv. **parita**. Ta může být **sudá** nebo **lichá**. Její základní princip spočívá v přidání jednoho bitu na konec zprávy, který se při vyhodnocování přičte k bitům předchozím. Tento tzv. paritní bit může mít hodnotu buď 0, nebo 1, podle toho, jestli výsledný součet má být lichý nebo sudý.

Příklad: Pokud je parita sudá, ale součet je lichý, nastala v přenosu chyba. Pokud jsou však chybné už 2 bity, tak pomocí parity chybu neodhalíme. Z toho důvodu se často využívají složitější zabezpečení. [2] [3]

1.2 Referenční model ISO/OSI

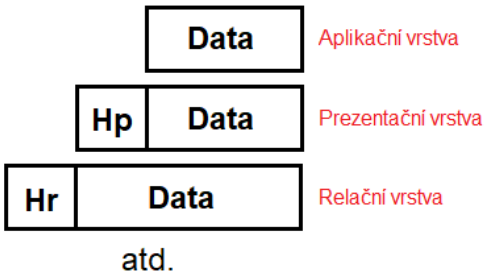
Dále je k pochopení přenosu dat důležité zmínit model ISO/OSI (Open Systems Interconnection). Tento model popisuje komunikaci mezi dvěma členy pomocí rozdělení do sedmi vrstev, z nichž má každá speciálně vymezené úkoly. Tyto vrstvy se nazývají:

- **Fyzická vrstva (Physical layer)** – zprostředkovává vlastní přenos bitů, definuje konektory, kabely či jiná přenosová média a způsob kódování dat čili rozlišení log.1 a log.0
- **Linková vrstva (Data-Link layer)** – stará se o řízení a logiku přenosu, adresování na nejnižší úrovni, členění dat do paketů, kontrolu kolizí, celistvosti dat a detekce chyb
- **Síťová vrstva (Network layer)** – zajišťuje řízení komunikace po síti, směřování a výběr trasy, propojení odlišných sítí pomocí routerů apod.
- **Transportní vrstva (Transport layer)** – řídí tok dat a zaručuje správnost přenosu
- **Relační vrstva (Session layer)** – zajišťuje navázání a ukončení spojení, synchronizaci a bezpečnostní procedury
- **Prezentační vrstva (Presentation layer)** – zabývá se strukturou přenášených dat (nikoli obsahem), řeší kódování, šifrování, komprese apod.
- **Aplikační vrstva (Application layer)** – tvoří rozhraní pro uživatele, zpracovává příchozí data do takové podoby, aby s nimi mohl pracovat člověk



Obrázek 1: Referenční model ISO/OSI [4]

Komunikace probíhá postupně přes všechny vrstvy, od nejvyšší k nejnižší ze strany odesilatele, poté od nejnižší k nejvyšší na straně příjemce. Totožné vrstvy na obou stranách používají stejný protokol, proto spolu zdánlivě komunikují přímo. Z hlediska obsahu dat každá vrstva při odesílání přidá vždy ke zprávě nějakou informaci, tzv. hlavičku. V případě přijmu hlavičku odebere.



Obrázek 2: Přidání hlaviček v jednotlivých vrstvách modelu ISO/OSI

Tento model byl vysoce propagován od roku 1984, kdy vznikl, avšak v praxi se plně neuchytil, a proto slouží jen jako obecný model. Reálně tak každá komunikace nemusí nutně obsahovat všechny z těchto vrstev. Např. Profibus obsahuje pouze 3 vrstvy (vrstvy 1,2,7). [4]

1.3 Sběrnice a komunikační protokoly

V průmyslové komunikace se často setkáváme s pojmy sběrnice a komunikační protokol. Ty se v běžné mluvě však velmi často zaměňují, proto je důležité znát mezi nimi rozdíl. Existují však i pojmenování standardu, který zahrnuje jak sběrnici, tak komunikační protokol.

Sběrnice (BUS) je skupina vodičů pro řízení a přenos dat mezi zařízeními v průmyslu. Každá sběrnice má definované vlastnosti fyzické vrstvy, tzn. předepsanou rychlost přenosu (Bd), napěťové úrovně a konektor.

Port označuje spojující bod, fyzický konektor, ke kterému se připojuje kabel externího zařízení. Konektory bývají mezi standardy sběrnic odlišné kvůli rozdílnému počtu vodičů, ale také záměrně, aby nebylo možné připojit k sobě dvě rozdílné sběrnice.

Komunikační protokol (Communication protocol) je souhrn pravidel pro přenos dat. [5]

1.4 Úrovně komunikace mezi zařízeními

Strukturu komunikací mezi zařízeními lze pro přehlednost hierarchicky rozdělit do 3 úrovní.

- Úroveň sensorů a aktuátorů
- Úroveň Field
- Úroveň počítačů

První (nejnižší) úroveň používá jednoduchý a cenově nenákladný přenos. Data a napájení bývají přenášeny najednou. Např. u AS-interface.

Úroveň Field tvoří distribuovaná zařízení, jako jsou I/O moduly, převodníky, řídicí jednotky analyzující zařízení, rozvaděče nebo operátorské terminály komunikující v aktuálním čase (Real Time). Např. Profibus DP/PA, Profinet, Can.

Nejvyšší úroveň komunikace probíhá mezi průmyslovými počítači PLC, případně i kancelářskými PC a jinými zařízeními, která pracují s tokem informací o velkém objemu dat. Používá se např. Ethernet s protokolem TCP/IP.

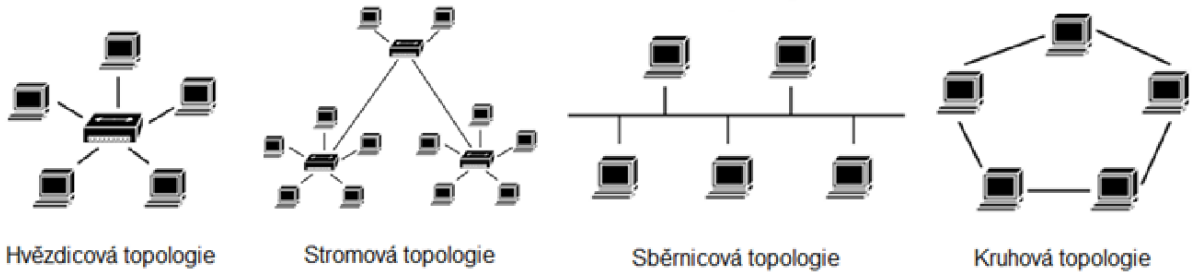
V praxi se pak komunikace odehrává jak horizontálně na každé úrovni, tak vertikálně skrz všechny hierarchické úrovně. Správně vrstvené a koordinované komunikační systémy nabízejí ideální předpoklady pro vytváření sítí ve všech oblastech výrobního procesu. [6]

Práce je dále soustředěna na systémy sběru dat a systémy pro řízení budov. Zmíněný Profibus či CAN jsou používané zejména ve výrobě.

1.5 Topologie

Zařízení, která mezi sebou komunikují, je potřeba nějakým způsobem propojit, tyto způsoby se nazývají topologie. [8]

- **hvězdicová** – všechna zařízení spojena do jednoho centrálního prvku, tzv. modemu
- **stromová** – vzniká propojením více modemů - např. lokální sítě LAN
- **sběrnicová** – všechna zařízení připojena na jednu sběrnici – převážně používané v průmyslu
- **kruhová topologie (Token ring)** – zařízení propojena do kruhu
    - Rámec zprávy se nazývá Token, který putuje v kruhu skrz zařízení, než nalezne příjemce. Hlavní nevýhodou je, že když dojde k přerušení obvodu či výpadku jednoho zařízení, nastávají komplikace.
- **vzájemné propojení** – všechna zařízení propojena tzv. každé s každým
    - Představuje nejrychlejší a nejspolehlivější přenos. Nevýhodou je cena, kvůli potřebě velkého množství kabeláže a organizace (některé cesty mohou být velmi dlouhé).



Obrázek 3: Topologie sítí [8]

2 Nejpoužívanější standardy

Tato kapitola je zaměřena na ty nejznámější standardy běžně nasazované v průmyslu. Nejdřív se specializuje na sběrnice (RS232, USB, Ethernet), poté na konkrétní komunikační protokoly (MODBus, M-Bus, BACNet).

2.1 RS232

RS232 (1969), obecně označována také jako *sériový port* nebo *sériová linka*, je pravděpodobně nejznámějším příkladem průmyslové sběrnice na světě. Dříve se používala např. i u osobních počítačů, kde je dnes však nahrazena sběrnicí USB (pro připojení komponentů k PC) nebo ethernetem (komunikace s ostatními PC/PLC). V průmyslu má však i nadále využití. RS232 zajišťuje komunikaci pouze mezi 2 zařízeními, tzv. point to point. Koncové zařízení však může být propojeno s dalšími zařízeními. Např. konkrétně v naší práci je RS232 použita pro komunikaci mezi PLC a převodníkem. K převodníku může být potom připojeno několik snímačů, které jsou adresovány. V tom případě je však nutné vyřešit kolize, tzn. aby nevysílalo více sensorů současně. [9]

2.1.1 Vlastnosti RS232

Komunikace RS232 je asynchronní a přenos dat probíhá od nejméně významného bitu (LSB). Maximálně dosažitelná rychlost přenosu je 115200 Bd. Další používané rychlosti jsou např. 57600, 38400, 19200, 9600, 4800, 2400 Bd. Maximální délka vedení se liší dle průměru a kvality použitých vodičů a rychlosti přenosu. Standard připouští max. 15–20 m. Konektory jsou CANON 9 nebo CANON 25. [10]

Používané vodiče sběrnice RS232 je možné rozdělit na 2 skupiny:

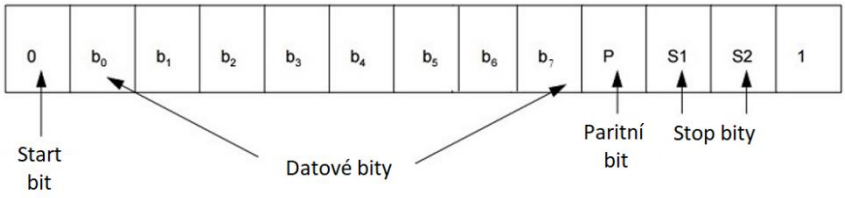
1) Datové signály – **RXD** a **TXD** (logická 0 je +3 V až +15 V, logická 1 je -3 V až -15 V)

2) Řídicí signály – **RTS, CTS, DTR, DSR** atd. (logická 0 je -3 V až -15 V, logická 1 je +3 V až +15 V)

Zmíněné řídicí signály mohou, ale nemusí být používány. Základní 3 vodiče (RXD, TXD a GND) jsou pro základní komunikaci zcela postačující.

2.1.2 Formát rámce RS232

Rámec přenášené zprávy se skládá ze start bitu označujícího začátek zprávy, datových bitů (typicky 8), paritního bitu (nepovinný) a jednoho nebo dvou stop bitů. [9] Viz obrázek 4.



Obrázek 4: Formát rámce komunikace přes RS232 [31]

2.1.3 RS485

V dnešní době se také často používá modifikace RS485(1983). Od RS232 se liší napěťovými úrovněmi, ale hlavní její výhodou je možnost propojení více zařízení a vytvoření sítě. Podle původního standardu bylo možné připojit až 32 účastníků (zařízení), dnes však existují integrované obvody, které jich umožňují propojit až 250. RS232 propojuje pouze 2. [10]

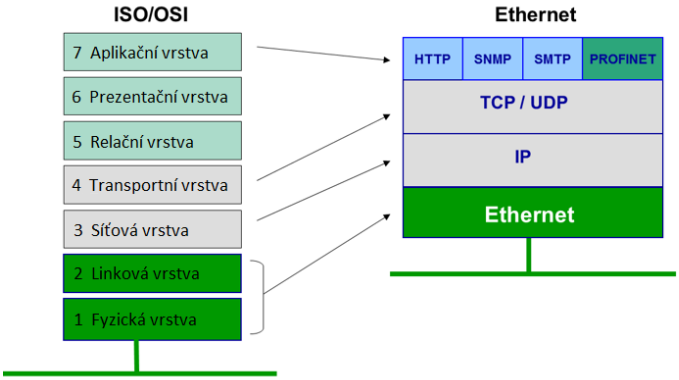
2.2 USB

Jednou z typicky používaných sběrnic v každodenním životě je sběrnice USB. Ta byla vyvinuta zejména pro připojení periferií k osobním počítačům, kde slouží pro přenos dat a zároveň pro jejich napájení. V tomto odvětví USB postupně nahradila všechny sériové i paralelní sběrnice využívané dříve např. pro myši, klávesnice, tiskárny, kamery atd. Jednou z těchto starších sběrnic mohla být např. IEEE 1394 FireWire. Výhodou USB je jednoduché použití a nízká cena. Nevýhodou je přenosová rychlost. Ačkoliv pro mnohá použití může být více než dostačující, zpravidla se však nedokáže vyrovnat např. Ethernetu, hojně používanému v komunikaci mezi zařízeními v průmyslu. [11]

2.3 Ethernet

Ethernet je standard, známý hlavně díky použití pro připojení zařízení k celosvětové síti internet. Jeho využití může být však i ve vlastních, lokálních sítích (LAN), především těch průmyslových. Právě Ethernet se totiž stal hlavním komunikačním prostředkem mezi PC, PLC, terminály apod. V dnešní době je však již i většina kancelářských a domácích zařízení vybavena Ethernet porty pro připojení na lokální síť (např. tiskárny).

Z hlediska správného zařazení do kontextu je důležité upozornit, že Ethernet je standard, který však realizuje pouze fyzickou a linkovou vrstvu. Síťovou vrstvu potom většinou definuje tzv. IP (Internet Protocol). Další vrstva, konkrétně transportní, může být realizována protokoly TCP nebo UDP. TCP se používá tam, kde nezáleží na rychlosti ale na bezchybnosti přenosu. UDP je používán v aplikacích, kde je důležitá rychlost a chyby nepředstavují takový problém (např. přenos videa). Poslední aplikační vrstva už může být různá, např. http. Z uvedeného vyplývá, že celková komunikace po Ethernetu obsahuje 4 vrstvy modelu ISO/OSI. [13]



Obrázek 5: Ethernet v rámci ISO/OSI modelu [13]

2.4 MODBus

MODBus je typický příklad komunikačního protokolu pro komunikaci elektrických zařízení s programovatelnými automaty PLC. Využíván je hojně v různých odvětvích průmyslu, především v automatizaci budov a SCADA systémech. MODBus je tzv. open protocol, to znamená, že pro jeho využití, není potřeba licence. Díky tomu a jeho praktičnosti je podporován mnoha výrobci na trhu. [14]

Komunikace je postavena na architektuře Master-Slave využívající sběrnicovou topologii. Na sběrnici je jedno zařízení Master (PLC), které vysílá adresované dotazy, a ostatní zařízení Slave, která na dotazy odpovídají. Slave (např. Sensor) nevytváří dotazy samostatně. MODBus komunikace může probíhat např. přes Ethernet nebo sériovou linku RS (většinou RS485). Charakteristickým znakem protokolu MODBus je to, že každé Slave zařízení obsahuje paměťové registry, do kterých zapisuje data nebo z nich data čte (hodnoty vstupů a výstupů). [15] [16]

2.4.1 Rámec komunikace MODBus

MODBus komunikace (MODBus RTU) má přesně definovaný rámec přenášené zprávy. Dotaz se skládá z:

- **Slave Adress** – 8 bitů (00000000 = *broadcast* – vysílá informaci všem zařízením, nikdo na ni neodpovídá)
- **Function Code** – 8 bitů – požadovaná operace (čtení/zápis)
- **Data** – n x 8 bitů, např. adresa registrů, počet bitů k přečtení, hodnota k nastavení
- **CRC** (Error check) – kontrolní součet
- mezera mezi vysíláním – vždy alespoň 28 bitů

Odpověď je tvořena stejným způsobem, s tím rozdílem, že pokud se při přenosu vyskytla chyba, zapíše se na první (nejvyšší) bit ve Function Code jednička. [16]

2.4.2 Omezení

ModBus dokáže adresovat až 255 Slave zařízení připojených na sběrnici. Podle původního standardu je předepsaný maximální počet připojitelných zařízení k jednomu kabelu sběrnice 32. Pro rozšíření ModBus sítě o další zařízení (případně připojení na vyšší vzdálenost) se používají tzv. opakovače (anglicky repeatery), které zesilují signál a vytváří nový segment, možný pojmout až 32 dalších zařízení. Dnes však existují i hardwarová řešení umožňující připojit na jednu linku až 250 zařízení.

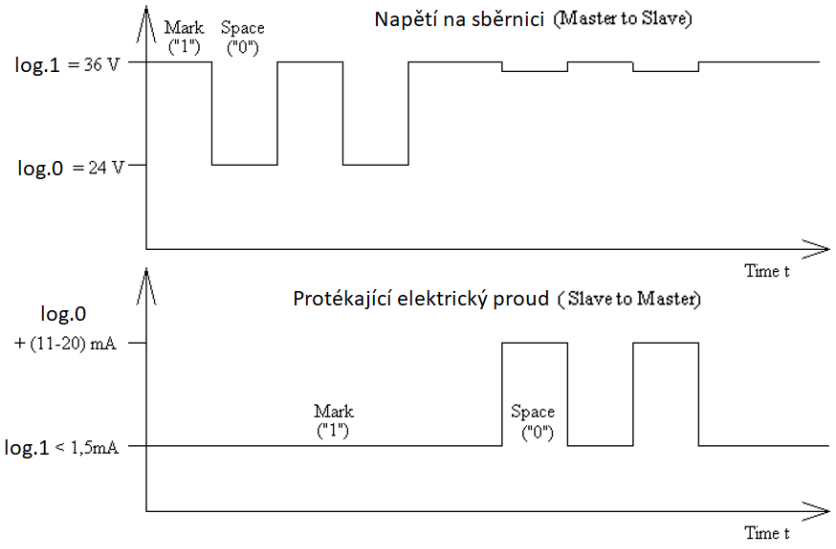
Maximální možná délka kabelu je teoreticky 1000 až 1200 m. Reálná maximální možná délka však závisí na počtu připojených zařízení, komunikační rychlosti a použitých vodičích (kvalita a průřez). Proto bude vždy nižší. [15] [16]

2.5 M-Bus

Za zmínku stojí také komunikační protokol M-Bus (Meter-Bus) určený pro dálkový odečet hodnot měřičů spotřeby vody, plynu a elektrické energie. Komunikace probíhá po 2 vodičích, což M-Bus činí cenově velmi efektivním. Používá se na velké vzdálenosti a v aplikacích, kde není potřeba vysoké rychlosti, zato však spolehlivosti. Proudový signál, vysílaný ze zařízení Slave, je totiž, na rozdíl od napěťových signálů, odolný vůči rušení, takže přenos probíhá téměř bezchybně. M-Bus definuje fyzickou, linkovou a aplikační vrstvu. Komunikace probíhá na úrovni Master-Slave. [17]

2.5.1 Fyzická vrstva M-Bus

M-Bus komunikace je poněkud netypická, avšak velmi zajímavá. Pro zařízení Master je log.1 definována 36 V, zatímco log.0 odpovídá 24 V. Tato napětí slouží zároveň k napájení. Zařízení slave odpovídá modulací spotřeby elektrického proudu (částečné zkratování). Log.1 pro slave zařízení je definována 1.5 mA, zatímco při log.0 se zvýší proud na 11-20 mA. Na obrázku můžete vidět, že změna proudu způsobí i lehké kolísání napětí.



Obrázek 6: Fyzická vrstva protokolu M-Bus – Napěťové a proudové úrovně [17]

Síť M-Bus je nejčastěji tvořena topologií hvězdicovou nebo sběrnicovou, je však možné použít i topologie jiné. Do sítě je možné připojit až 250 Slave zařízení. S délkou použité kabeláže tento počet klesá. Konektror M-Bus není specifikovaný. Za zmínku stojí také to, že existuje i bezdrátová verze zvaná *Wireless M-Bus*. [17]

2.5.2 Linková vrstva M-Bus

Linková a aplikační vrstva M-Bus už nejsou příliš zajímavé a popis protokolu je celkem složitý, proto je zde uveden pouze vzorec rámce. Komunikace probíhá po skupinách 11 bitů: start bit, 8 datových bitů, paritní bit (sudá parita), stop bit. Viz obrázek 13.

Obrázek 7: Linková vrstva protokolu M-Bus [17]

Důležitým charakteristickým rysem M-Bus komunikace je to, že Slave zařízení (měřič) v odpovědi na jednoduchý dotaz poskytuje kompletní řadu údajů včetně měřených hodnot, fyzikálních jednotek a jejich významu (např. teplota, celková energie, okamžitý výkon atd.). [17]

2.6 BACNet

Posledním příkladem a zároveň tím na nejvyšší úrovni je BACNet (Building and Automation Network). BACNet je určen pro komunikaci systémů v automatizaci budov. Konkrétně např. řízení topení, ventilací, klimatizací, osvětlení, bezpečnosti, detekce požárů apod. Jeho hlavní a nepřekonatelnou výhodou je to, že umožňuje integraci systémů a zařízení různých výrobců. BACNet se spíše než tím, jak přenášet data, zabývá tím, co je obsahem komunikace. Jeho cílem je nahradit různé formy komunikace jedním společným jazykem, aby všechna zařízení byla vzájemně kompatibilní a používala stejné datové struktury. Pro svůj přenos používá širokou škálu fyzických a linkových vrstev. [19]

2.6.1 Objekty, vlastnosti, služby

BacNet zavádí tzv. **objekty**, které obsahují určité informace a lze k nim přistupovat ze sítě standardizovaným způsobem. Objekt může reprezentovat např. fyzikální vstup, výstup, souhrn hodnot, softwarovou funkci, příkaz, výpočet, smyčku, časový plán, kalendář, soubor atd. Každý objekt obsahuje soubor vlastností (anglicky „properties“**)**, které charakterizují jeho stav a chování. Konkrétní BacNet zařízení je potom skupina objektů, které reprezentují jeho funkci. Zařízení tak může mít např. několik objektů pro konkrétní vstupy a výstupy (digitální nebo analogové), jeden nebo více časových rozpisů, příkazů, funkcí atd. S touto objektovou architekturou lze velmi jednoduše pracovat, není pevně daná a je rozšiřitelná podle potřeby.

Zařízení mezi sebou vysílají zprávy, tzv. služby (anglicky „services“), které určují, co má zařízení vykonat. Jsou jimi typicky např. ReadProperty, WriteProperty, CreateObject, DeleteObject atd. Výhodou BACNetu je to, že je volně bezlicenčně přístupný a nabízí možnost vytváření nových uživatelsky definovaných properties. [19]

2.7 Porovnání standardů

Sběrnice RS232, RS485 i Ethernet jsou v průmyslové komunikaci velmi často používané. USB se v průmyslu prakticky nepoužívá. ModBus je velmi jednoduchý jak na pochopení, tak na implementaci. Pro každé zařízení je však potřeba znát tabulku s popisy registrů a významy dat. Oproti tomu M-Bus odpovídá veškerými údaji o proměnných a jednotkách, přijímaný řetězec je však potřeba rozebrat. Absolutní špičkou je BACNet, který definuje objekty a jejich vlastnosti a disponuje službami pro jejich doptávání. Toto rozdělení je pak velmi přehledné pro vlastní aplikaci. Nevýhodou je však poměrně náročná implementace BACNet sítě.

V předchozím textu jsme byli seznámeni s těmi nejpoužívanějšími komunikačními standardy v systémech budov, provedli jejich porovnání a získali tím určitý rozhled, který je v tomto oboru nezbytný. Nyní se práce bude věnovat konkrétně standardu SDI-12.

3 SDI-12

SDI-12 je komunikační standard používaný pro sensory měřící různé environmentální veličiny (teplota, vlhkost, tlak apod.). Používají se převážně v zemědělství. Název SDI-12 je zkratka pro *serial/digital interface at 1200 baud*. Nese tak informace o tom, že komunikace je sériová při baudové rychlosti 1200 bitů za sekundu. Historicky sběrnice SDI-12 poprvé spatřila světlo světa roku 1988. Vyvinuta byla skupinou firem ve spolupráci s U. S. Hydrologic Instrumentation Facility (Správa hydrologických zařízení USA). Ačkoliv SDI-12 se za svojí existenci nestala příliš rozšířenou, stále je v procesu dalšího vývoje aktualizací, které pravidelně vychází (poslední verze 30. ledna 2021). Základ komunikačního protokolu zůstává nezměněn, proto všechny aktuální i dřívější verze protokolu SDI-12 spolu bezchybně spolupracují. Nasazení SDI-12 je zamýšleno pro finančně a energeticky nenáročné aplikace. Předností tohoto standardu je odolnost vůči rušení okolními vlivy díky relativně nízké rychlosti přenosu. To je jedním z důvodů, proč je SDI-12 používaná převážně v zemědělství, jelikož právě tam se může stát, že sběrnice přijde do styku s agresivním prostředím. Standard zaručuje připojení až 10 sensorů na vzdálenost 200 stop, což odpovídá přibližně 61 metrů. Při vyšším počtu sensorů vzdálenost klesá. [20]

3.1 Sensory SDI-12

Sensory pracující s SDI-12 jsou založené na mikroprocesorech. Díky tomu obsahují vlastní unikátní samokalibrační algoritmus. Díky adresaci pak umožňují jednoduchou záměnu sensorů bez přeprogramování vyhodnocování dat. Dále sensory obsahují nízkonákladové paměti EEPROM (pro ukládání dat při složitějších operacích a kalibracích), regulátor napájení atd. Hlavní vlastností těchto sensorů je však možnost režimu spánku. Při měření environmentálních veličin není většinou třeba sbírat data často, proto čidla „spí“. Ty se probudí až při jejich vyvolání, provedou měření a odešlou data. To samozřejmě vede k nižší spotřebě, na kterou je zde velmi kladen důraz, neboť sensory SDI-12 se používají převážně v aplikacích napájených z baterie. Čidla není třeba zvlášť napájet, jelikož sběrnice obsahuje napájecí vodič s napětím 12 V. [20]

3.2 Přenosové parametry SDI-12

SDI-12 využívá sběrnicovou topologii. Sběrnice obsahuje 3 vodiče.

1) Serial data line = sériová komunikace

2) Ground line = zemnicí vodič s nulovým potenciálem

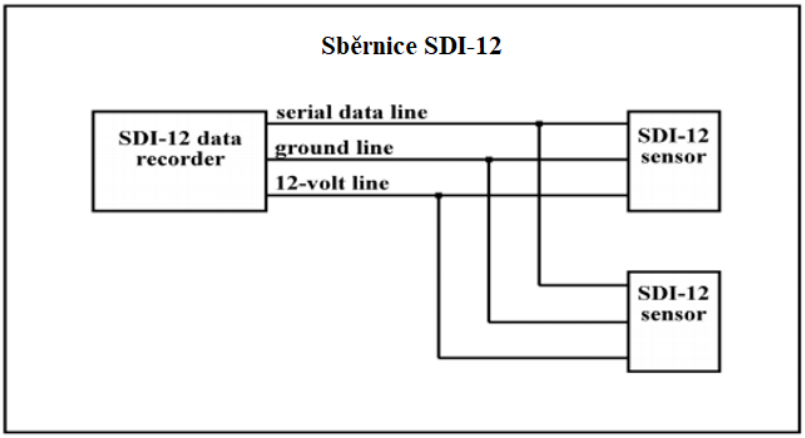
3) 12Volt line = napájecí vodič s elektrickým napětím 12 V pro napájení sensorů

Napěťové úrovně na datovém vodiči [20]:

-0,5 až 1 V – Log. 1 („*Marking*“)

3,5 až 5,5 V – Log. 0 („*Spacing*“)

1 až 3,5 V – Nedefinováno („*Transition*“)

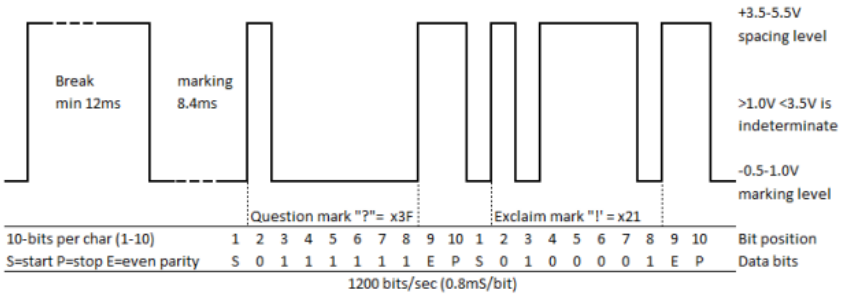


Obrázek 8: Sběrnice SDI-12 [20]

SDI-12 nemá žádný definovaný konektor pro přenos. [20]

3.2.1 Vzorec rámce

Komunikace probíhá pomocí ASCII znaků zakódovaných a přenášených na vodiči DATA-LINE. Rámec se skládá z jednoho start bitu, 7 datových bitů (přenos od LSB), jednoho paritního bitu a jednoho stop bitu. Příklad komunikace s vyznačenými napěťovými úrovněmi je vidět na obrázku 17. [20]



Obrázek 9: Příklad průběhu komunikace SDI-12 [29]

3.3 Komunikační protokol

Ačkoliv je SDI-12 označován obecně jako průmyslová sběrnice, jeho standard definuje kromě fyzické vrstvy také příslušný komunikační protokol linkové vrstvy. Pro přehlednost je základní používaný komunikační cyklus rozdělen do pěti po sobě jdoucích kroků: [20]

1. Data Recorder (v našem případě PLC) probudí všechny sensory na sběrnici tzv. *SPACE*, čili mezerou (logickou nulou).

2. DR odešle konkrétní příkaz adresovaný příslušnému sensoru. (Např. dotaz na měření hodnot)

3. Sensor odpoví maximální čas potřebný k naměření hodnot a jejich počet.

4. DR odešle příkaz ke čtení hodnot.

5. Sensor odpoví jednou nebo více hodnotami.

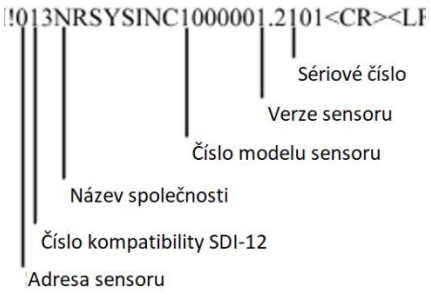
3.3.1 Adresace zařízení

Stejně jako u jiných standardů, má i zde každé zařízení svoji unikátní adresu, kterou lze podle potřeby měnit. Pro sensory jsou v rámci komunikačního protokolu vyhrazené adresy 1–9 (ASCII: 49–57), případně podle potřeby lze využít i a–z (97–122) nebo A–Z (65–90). Tato adresa pak tvoří první znak každého odeslaného příkazu či odpovědi pro snadné rozpoznání a ověření, jestli přišla odpověď od správného sensoru. Defaultně má každý sensor z výroby nastavenou adresu "0". [20]

3.3.2 Používané příkazy

Zde jsou uvedeny základní možné příkazy, na které sensory dokážou reagovat. Kromě níže uvedených existují samozřejmě příkazy další, pro naše využití však nepodstatné.

- **Acknowledge Active Command (a!)** – testování odezvy sensoru pro ověření funkčnosti
- **Send Identification Command (aI!)** – požadavek identifikace sensoru



Obrázek 10: Rozbor odpovědi čidla na příkaz aI! [20]

- **Address Query Command (?!)** – dotaz všech připojených zařízeních na jejich adresy, např. pokud neznáme adresu sensoru
- **Change Address Command (aAb!)** – změna adresy (z "a" na "b")

- **Start Measurement Command (aM!)** – požadavek na měření

Při přijetí příkazu *aM!* sensor ještě nemá požadované hodnoty změřené, proto vrací zpět počet sekund, za jak dlouho bude měření připravené, a počet měřených hodnot. Až sensor hodnoty změří a připraví, je nutné o tyto hodnoty zažádat. Tzn. nevrací je automaticky po uplynutí měřicího času. Příklad vyslaného příkazu *aM!* je na obrázcích 11 a 12.

Obrázek 11: Rozbor odpovědi čidla na příkaz *aM!* v textové podobě

50 77 33 50 48 48 49 50 13 10

Obrázek 12: Odpověď čidla na příkaz *aM!* v ASCII kódu

Poslední dva znaky v odpovědi se nazývají Carriage return a Line Feed, které jsou v podstatě neviditelné. Jejich význam je v přesunu kurzoru na začátek nového řádku, a tím ukončení odpovědi.

- **Send Data Command (aD0!, aD1! . . . aD9!)** – dotaz na měřené hodnoty (následuje např. po příkazu M)

Sensor může obsahovat až deset čísly postupně označených paměťových registrů (D0-D9). Každý z nich může obsahovat až 3 uložené hodnoty. Pro čtení každého registru musí být vyslán zvláštní příkaz. Každý sensor může využít libovolné množství zmíněných registrů/hodnot, proto je nutné se řídit datasheetem produktu, kde jsou jednotlivé hodnoty v registrech vysvětleny. Využití registrů bývá zpravidla od nejnižšího po nejvyšší, tzn. začíná od D0. U většiny sensorů je ale využit právě jen tento registr, kde jsou uloženy 2 nebo 3 hodnoty. [20]

4 Použitý hardware a software

Pro vývoj driveru a jeho testování bude z hlediska hardware speciálně zapotřebí:

- PLC (Merbon Runtime)
- Převodník RS232/SDI-12
- Sensory SDI-12

Z hlediska softwarového:

- Vývojové prostředí Merbon IDE
- Softwarový terminál (např. Tera Term)

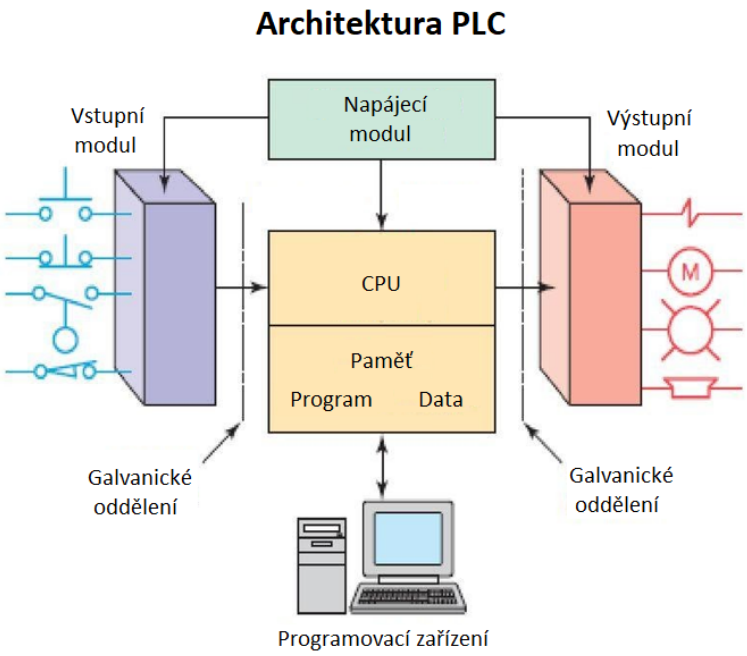
Nyní se práce detailně věnuje popisu PLC a způsobům jeho programování. V textu jsou obsaženy informace, které jsou pro práci s ním zásadní.

4.1 PLC

Programovatelný automat, čili PLC, je počítač navržený pro přímé použití v průmyslu, kde je požadavek na odolnost a spolehlivost. Takový přístroj by měl být odolný vůči fyzickému poškození jako jsou nárazy, dále proti vysokým teplotám, vlhkosti, prachu apod. Žádoucí je rovněž bezpečnost, např. zapamatování posledních stavů a jejich obnovení při výpadku.

4.1.1 Architektura PLC

PLC se vyrábí buď v provedení kompaktním, obsahujícím množství základních prvků, nebo v provedení modulárním, kdy uživatel sestaví PLC stanici pomocí jednotlivých modulů podle potřeb dané aplikace. PLC, stejně tak jako další typy počítačů, obsahuje mikroprocesor, paměť typu RAM a uživatelskou paměť typu EEPROM. Paměť EEPROM slouží k ukládání programu, jelikož u paměti typu RAM dochází při odpojení napájení ke ztrátě dat. Většina PLC má však záložní baterii, která dokáže zabránit zmíněnému vymazání dat. Přesto však spolehlivějším a optimálnějším místem pro program zůstává EEPROM a RAM, tedy zůstává pouze pro ostatní, při chodu měnící se, data.



Obrázek 13: Architektura PLC [22]

Pomocí vstupů je PLC schopné snímat a zaznamenávat hodnoty. V případě výstupů pak ovládat různé akční členy jako jsou pohony, regulační orgány, frekvenční měniče atd. Kromě těchto portů mívá PLC i porty komunikační, např. pro sběrnice RS232/485 nebo Ethernet. Díky nim může komunikovat se zařízeními, jako jsou další PLC, osobní počítače, terminály, řídicí jednotky nebo sensory. Výhodou sensorů připojení na sběrnici je úspora zapojení, ale hlavně digitální podoba vysílaného slova, obsahující přesné hodnoty. V případě měření přes analogové vstupy PLC je měřena velikost elektrického napětí nebo proudu, která je potřeba výpočetními operacemi převést na reálné hodnoty, např. na teplotu. Tento problém řeší zmíněné, v dnešní době již inteligentní sensory s vnitřní elektronikou, vracející přímo reálné hodnoty.

Software uvnitř PLC se dělí na dvě části, samotný uživatelský program a Firmware, který ho spouští. Firmware tvoří operační systém PLC a zajišťuje tak kompatibilitu s vývojovým prostředím, komunikaci mezi CPU, pamětí RAM a moduly a mnoho dalších úkolů, které jsou pro obyčejnou lidskou mysl těžce představitelné. Firmware instaluje výrobce PLC. [21] [22]

4.1.2 Pracovní cyklus PLC

PLC pracuje v cyklu (anglicky „Scan“), který tvoří:

- čtení vstupních hodnot a jejich zápis do paměti,
- vyhodnocení podmínek programu,
- diagnostika – např. ověření funkčnosti připojených modulů,
- přiřazení výstupních hodnot programu k reálným výstupům PLC.

Trvání jednoho cyklu, též *Scan Time*, se liší podle CPU, velikosti programu, složitosti výpočtů, počtu vstupů a výstupů atd. Může tak být od jednotek až po stovky milisekund. [23]

4.1.3 Programovací jazyky

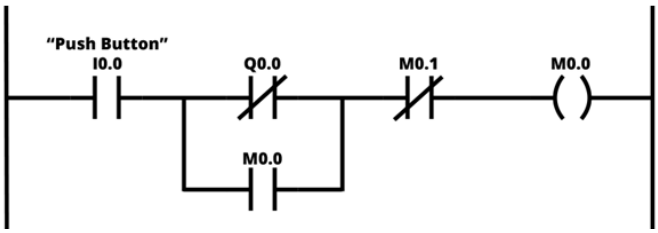
Programovací jazyky vychází z normy *IEC 1131-3*. Česká verze je označována *ČSN EN 61131-3 Programovatelné řídicí jednotky – Část 3: Programovací jazyky*. Tato norma byla přijata většinou významných výrobců PLC na světě. Díky přesně stanoveným programovacím jazykům, dodržujícím jednotnou syntaxi, jsou uživatelé snadno schopni přecházet k používání produktů jiného výrobce.

Tyto jazyky jsou:

- LD (Ladder Diagram)
- FBD (Function Block Diagram)
- ST (Structured Text)
- SFC (Sequential Function Chart)
- IL (Instruction List)

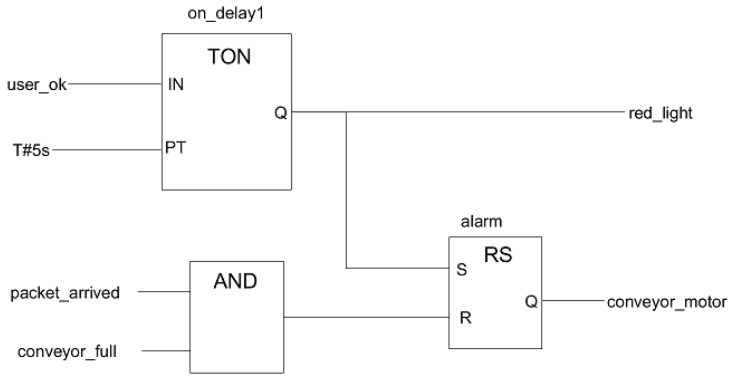
Nyní následuje stručné přiblížení zmíněných programovacích jazyků.

Ladder Diagram neboli **Ladder Logic**, česky také **Příčkový Diagram** nebo **Reléové Schéma,** disponuje svojí jednoduchostí. Využíván je zejména pro nenáročné aplikace, používající pouze kombinační logiku. Viz obrázek 14. Ladder Diagram je možné kombinovat i s funkčními bloky, jelikož sám o sobě nabízí práci pouze s digitálními proměnnými.



Obrázek 14: Ukázka programovacího jazyku LD

FBD neboli diagram funkčních bloků se používá již pro sofistikovanější aplikace, jelikož obsahuje řadu speciálních bloků jako jsou čítače, časovače, regulátory apod. Podle potřeby lze do vývojového prostředí stáhnout doplňkové knihovny s dalšími funkčními bloky či si vytvořit vlastní (většinou v jazyce ST).



Obrázek 15: Ukázka programovacího jazyku FBD

ST neboli strukturovaný text umožňuje maximální kontrolu nad programem, jeho výhodou je snadná práce s proměnnými, poli, podmínkami a cykly. Program probíhá cyklicky shora dolů, tím umožňuje např. umístění Breakpointů pro lepší Debugging. Pro složitější úlohy je použití jazyka ST jasnou volbou. Na druhou stranu může být ST pro nezkušené programátory PLC méně přehledný a názorný. Zejména při dlouhém kódu může být náročné se v programu zorientovat. Jazyk ST je velmi podobný většině textových programovacích jazyků. Nejvíce společných znaků má s jazykem Pascal.

```
1  IF #start = 1 THEN
2      //comment
3      "Max_nr" := #Array[0];
4      FOR #i := 1 TO 10 DO
5          // Statement section FOR
6          IF #Array[#i] > "Max_nr" THEN
7              "Max_nr" := #Array[#i];
8          END_IF;
9      END_FOR;
10 END_IF;
11
```

Obrázek 16: Ukázka programovacího jazyku ST

Dalšími programovacími jazyky jsou Sequential Function Chart (SFC) podobný vývojovému diagramu a Instruction List (IL) ve formě jednoduchého seznamu příkazů. Používají se však velmi málo. Většina PLC programů je typu LD nebo FBD, avšak používané funkční bloky mají často strukturu psanou textově. Pro návrh driveru byl zvolen jazyk ST. [25]

4.1.4 Datové typy

Pro všechny z těchto jazyků jsou dle normy IEC 61 131-3 definovány tzv. elementární datové typy. Ty nejpoužívanější jsou, s jejich významem a rozdíly mezi nimi, uvedeny v tabulce. Neexistuje univerzální datový typ, při deklaraci každé proměnné je nutné zvolit pouze jeden z nabídky.

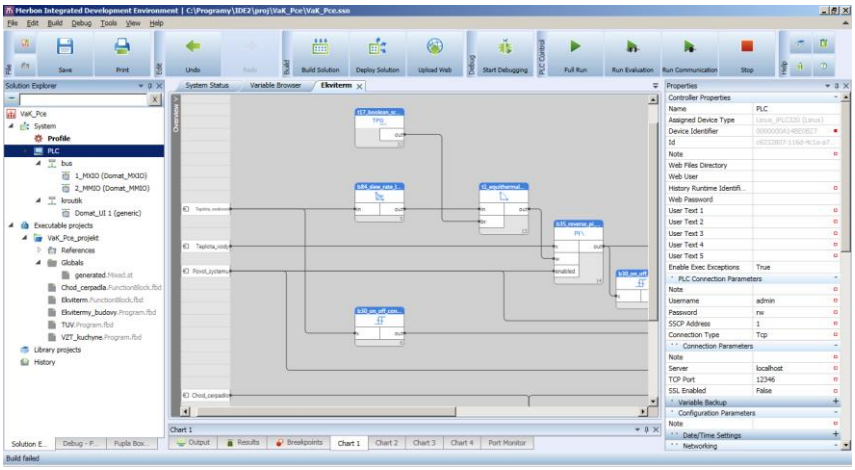
Tabulka 1: Elementární datové typy dle normy IEC 61 131-3 [26]

Datový typ	Význam	Počet bitů v paměti	Rozsah hodnot
BOOL	Boolovské číslo	1	0,1
INT	Celé číslo	16	–32 768 až +32 767
UINT	Nezáporné celé číslo	16	0 až 65 535
DINT	Celé číslo dvojnásobné délky	32	-2 147 483 648 až +2 147 483 647
SINT	Celé číslo poloviční délky	8	-128 až +127
REAL	Reálné číslo, pracující s desetinnými čárkami	32	-3.4E+38 až +3.4E+38
BYTE	Řetězec bitů	8	00000000 až 11111111
STRING	Řetězec znaků	Nespecifikováno	Max. 255 znaků
TIME	Časový údaj (doba trvání)	Nespecifikováno	Nespecifikováno
DATE	Datum	Nespecifikováno	Nespecifikováno

Kromě těchto elementárních datových typů, které mohou nabývat pouze jedné hodnoty, existují tzv. pole (Array). To jsou struktury, obsahující více prvků jednoho datového typu. Prvky jsou v poli seřazeny a adresovány tzv. indexem. Datové typy je rovněž možné mezi sebou převádět. K tomu existují příslušné funkce, např. INT_TO_STRING, převádějící celé číslo do textové podoby. [26]

4.1.5 Vývojové prostředí Merbon IDE

Jelikož cílem práce je vytvořit komunikační driver pro PLC firmy Domat Control System, které pracují s Merbon Runtimem, bude programování probíhat ve vývojovém prostředí Merbon IDE. To v základu ve svých knihovnách obsahuje drivery pro základní nejpoužívanější protokoly, jako jsou ModBus či M-Bus. Zároveň však umožňuje vytvářet drivery vlastní, pokud to složitost komunikačního protokolu a jeho popis uživateli dovolí. SDI-12 je jedním z příkladů, kde by neměl být problém takový program vytvořit. Programování PLC je z hlediska použití programovacích jazyků normalizované podle zmíněného standardu IEC 61131-3. Vývojové prostředí má však každý výrobce své. To se liší v designu, přehlednosti, nabízených funkcích či obsáhlosti knihoven. Domat Control System používá své vlastní vývojové prostředí Merbon IDE, které je k dispozici zdarma a nabízí přehlednou a pohodlnou obsluhu PLC, včetně možnosti tvorby uživatelského rozhraní HMI. [27]



Obrázek 17: Ukázka vývojového prostřední Merbon IDE [27]

4.2 Převodník SDI-12/RS232

Vyvíjená aplikace je mířená pro PLC obsahující port RS232. Díky v podstatě stejné stavbě rámce přenosu sběrnic SDI-12 a RS232 se na trhu vyskytují cenově dostupné převodníky, které převádí napěťové úrovně a Baudovou rychlost, tudíž fyzickou vrstvu přenosu. Protokol, který je součástí již vrstvy linkové, však zůstává zachován. V této práci byl použit převodník C1023 od firmy icMK, tudíž se není třeba zabývat tvorbou celého rozhraní od základu, pouze programováním a kódováním vysílaných příkazů a dat. Převodník je napájen 12 V stejnosměrného napětí, které slouží i jako napájení sensorů na sběrnici SDI-12. Navíc disponuje i funkcemi odeslání "No Response" v případě, že nepřišla odpověď od žádného sensoru, a "Low Voltage" v případě, že je na převodníku nedostatečné napětí. Všechny podstatné parametry převodníku jsou zobrazeny v tabulce 5. [28]

Obrázek 18: Převodník icMK C1023 [28]

Tabulka 2: Parametry převodníku

Napájecí napětí	10-17 V (typicky **12 V**)
Proudová spotřeba bez připojených čidel	60 mA
Pracovní teplota	–30 až +60 °C
Parametry RS232	9600 Bd, 8 datových bitů, 1 stop bit, bez parity
Počet možných připojených zařízení	10
Maximální délka kabelů čidel	60 m
Rozměry	140×118×64 mm
Hmotnost	250 g

5 Návrh driveru

Nyní se práce věnuje samotnému vývoji driveru. Před programováním je důležité důkladně promyslet posloupnost vykonávání programu. Následuje tak rozdělení programu na dílčí části, které byly v rámci psaní programu postupně řešeny.

5.1 Postup

Proces vývoje byl rozdělen do následujících kroků:

1. Promyšlení, návrh měřicího cyklu, analýza potřebných proměnných
2. Rozdělení programu na části (pro postupné vypracování funkčních celků)
3. Zajištění komunikace přes RS232
4. Zakódování adresy a příkazu, převod pomocí ASCII tabulky
5. Sestavení příkazu ve vstupním bufferu, následné vyslání příkazu na port
6. Čtení portu – ukládání do vstupního bufferu
7. Rozebraní odpovědi, převody datových typů
8. Čekání – využití časovače TON
9. Rozebrání odpovědi
10. Ukončení cyklu, zahájení odpočtu do dalšího měření
11. Použití Funkčního bloku ve FBD programu, ověření funkčnosti
12. Řešení kolizí při použití více bloků, obsazenost portu, čekání, největší obtíž při prvním spuštění, kdy každý blok žádá měření, propojení bloků, určení priority
13. Testování s reálným PLC a sensory

V kroku č. 1 byla podle popisu standardu SDI-12 navržena struktura celého měřicí cyklu. Jeho funkce je znázorněna ve vývojovém diagramu (Příloha 1). V kroku č. 2 byl program rozdělen na několik částí, kterým se bude práce postupně věnovat.

5.2 Struktura programu

Hlavní celek měřicího cyklu byl postaven na struktuře Case. Díky tomu byl program rozdělen do jednotlivých kroků, díky kterým se stal přehlednějším. V každém z těchto kroků se vždy vykoná určitá operace. Poté program přistoupí k dalšímu kroku. Program měl původně kroků více, ale za účelem úspory místa v paměti byl zjednodušen pouze do osmi kroků. V některých krocích se pak program ocitne dvakrát, jelikož se v rámci jednoho cyklu na sběrnici vždy vysílají právě 2 příkazy. Bylo by tak zbytečné, aby program obsahoval dvě téměř totožné části kódu. Změněn byl tak pouze vysílaný příkaz. Kroky č. 1 a 4 obsahují tak další vnitřní *case*. V kroku č. 1 se program rozhoduje, který příkaz bude vysílán. Nejprve

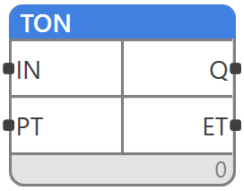
je vždy vysílán příkaz *M!*. Při navrácení do toho kroku je vysílán příkaz *D0!*. V kroku č. 4 se program, podle toho, který ze zmíněných příkazů byl naposledy vyslán, rozhodne, zda bude pokračovat ke kroku 5 nebo 6. V případě M! se přesune do kroku 5, kde setrvá určitý čas pro dokončení měření. Tento potřebný čas je známý z odpovědi čidla. V případě vyslání příkazu D0! se přesune do 6. kroku, kde bude řetězec rozebrán. Těm nejdůležitějším částem programu se bude práce nyní detailněji věnovat. Vložená ukázka kódu znázorňuje použití struktury Case, která tvoří hlavní část programu.

```
CASE com_state OF

0: //Ověření dostupnosti portu (přechod do kroku 1)

1: //Vymazání bufferů, uložení příkazu do vstupního bufferu
(přechod do kroku 2)

2: //Zápis na port (přechod do kroku 3)

3: //Čtení z portu (přechod do kroku 4)

4: //Po obdržení odpovědi program rozhodne, zda přistoupí ke kroku 5
nebo 6

5: //čekání na měření (návrat do kroku 1)

6: //Rozebrání řetězce měřených hodnot, příprava na převod
(přechod do kroku 7)

7: //Převod obdržených hodnot na datový typ Real, přiřazení
k výstupu (přechod do kroku 8)

8: //Ukončení měření, vynulování vnitřních proměnných
(návrat ke kroku 0)

END_CASE;
```

Ukázka kódu 1: Struktura měřicího cyklu

5.2.1 Časování

Spouštění měření podle stanoveného času je řešeno použitím funkčního bloku TON. Ten je předřazen celému měřicímu cyklu. Celá hlavní struktura Case je pak umístěna do podmínky IF, reagující na výstup časovače Q. Vstupní signál IN spouští odpočet. Parametr PT udává čas zpoždění aktivace výstupu. Po uplynutí zmíněného času PT se aktivuje výstup Q, který spustí měření. Vstup IN je resetován v okamžiku ukončení měření. Výstup ET udává uběhlý čas od aktivace vstupu IN. [25]



Obrázek 19: FB TON

Tabulka 3: Parametry funkčního bloku TON [25]

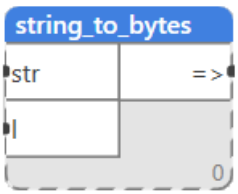
Vstup	Datový typ	Význam
IN	BOOL	Vstupní signál
PT	TIME	Nastavený čas zpoždění

Výstup	Datový typ	Význam
Q	BOOL	Výstupní signál
ET	TIME	Uběhlý čas od aktivace vstupu *IN*

Čekání na změření hodnot uvnitř cyklu je rovněž řešeno použitím FB TON.

5.2.2 Formování příkazů

Vysílané příkazy na sběrnici je možné rozdělit do dvou částí. První část tvoří adresa sensoru, kterou uživatel nastaví na vstupu FB. Ta je pak v rámci kódu převedena funkcí *string_to_bytes* na Ascii kód. Druhá část je již konkrétní příkaz, který je už připravený v proměnné *str_m* (M!) nebo *strd_d0* (D0!). Tato část je stejným způsobem rovněž převedena do Ascii kódu. Obě tyto části jsou společně uloženy do bufferu *glob.inBuffer*, odkud budou vysílány na sběrnici.



Obrázek 20: Funkce string_to_bytes

Tabulka 4: Parametry funkce string_to_bytes [25]

Vstup	Datový typ	Význam
STR	STRING_REF	Reference vstupního řetězce
L	DINT	Počet znaků k převedení

Výstup	Datový typ	Význam
=>	BYTEARRAY	Pole, obsahující ASCII hodnoty

5.2.3 Otevření portu

Pro otevření portu je potřeba nejprve sestavit řetězec specifikující sériovou komunikaci, tj. číslo portu, Baudovou rychlost a počty bitů. Řetězec ve formátu string vypadá následovně:
„serial: číslo portu: Baudová rychlost: počet datových bitů: počet paritních bitů: počet stop bitů“

serial:4:9600:8:N:1

Obrázek 19: Textový řetězec, specifikující sériovou komunikaci

Při pohledu na tento řetězec lze vidět, že bude využit port č. 4 s Baudovou rychlostí 9600 (stanoveno podle parametru převodníku). Slovo se bude skládat z 8 bitů, ve kterých je vždy zakódován znak v ASCII kódu. N značí „NO“, tudíž přenos bez paritního bitu, a nakonec 1 stopbit. Tento řetězec pak tvoří vstup funkce pro otevření portu *io.openport*, která vytváří na svém výstupu tzv. handle objekt obsahující parametry probíhající komunikace. S tímto, námi pojmenovaným, *com_handle* se dále pracuje při zápisu a čtení.

Obrázek 20: Funkce io.openport

5.2.4 Zápis do portu

Zápis probíhá funkcí *io.writeport*, jejíž vstupy a výstupy můžete vidět v následující tabulce. Vstupem je již zmíněný Handle (ukazatel na pole, kde je uložen řetězec), který má být zapsán a jeho délka. Při úspěšném zapsání vrací funkce na výstupu log.1. Program pak může pokračovat do dalšího kroku, tj. ke čtení obdržené odpovědi.

Obrázek 21: Funkce io.writeport

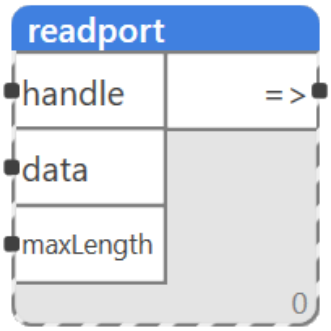
Tabulka 5: Parametry funkce io.writeport [25]

Vstup	Datový typ	Význam
HANDLE	IO.COMHANDLE	Parametry komunikace
DATA	BYTE_PTR	Ukazatel na data, která mají být zapsána
LENGTH	INT	Velikost zapisovaného řetězce (počet znaků)

Výstup	Datový typ	Význam
=>	BOOL	0... Data nebyla zapsána / 1... Data zapsána

5.2.5 Čtení z portu

Funkce pro čtení *io.readport* má podobné parametry jako předchozí funkce *io.writeport*. Jejich význam je opět vidět v tabulce. Funkce při čtení ukládá hodnoty do námi zvoleného pole. Řídí se podle zvolené maximální délky. Ta může být větší, než je reálná velikost odpovědi, tím se tak předchází i problémům při různě dlouhých odpovědích. Tato hodnota byla ze zkušeností získaných z testování sensorů zvolena 40. Čtení je ukončeno posledním znakem odpovědi (Carriage return, v ASCII hodnota 10).



Obrázek 22: Funkce io.readport

Tabulka 6: Parametry funkce io.readport [25]

Vstup	Datový typ	Význam
HANDLE	IO.COMHANDLE	Parametry komunikace
DATA	BYTE_PTR	Ukazatel na pole, kam se mají data ukládat
MAXLENGTH	INT	Maximální velikost ukládaného řetězce

Výstup	Datový typ	Význam
=>	BOOL	0... Data obdržena / 1... Data neobdržena

5.2.6 Zpracování přijatých textových řetězců

Po vyvolání funkce *io.readport* se do námi zvoleného pole *outBuffer* uloží přijatý řetězec znaků (odpověď čidla). Podle toho, v které části se program nachází (který příkaz byl před obdržením odpovědi vyslán), musí program rozhodnout, jak bude s odpovědí pracovat. Mohou nastat 2 případy:

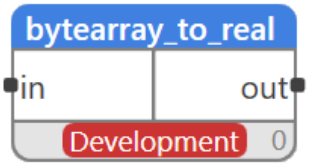
1. Odpověď obsahuje informaci o tom, kolik hodnot sensor změří a odešle (jejich význam v datasheetu sensoru) a čas potřebný pro změření hodnot (jak dlouho program má čekat, než bude žádat hodnoty).
2. Odpověď obsahuje už samotné výsledky měření.

V obou případech je nutné určitým způsobem rozebrat řetězec a převést proměnné datového typu *byte* na typ *real* nebo *int*.

Rozdělení řetězce podle specifických oddělovacích znaků je řešeno v hlavním programu. Pro samotný, v našem případě velmi specifický převod neexistovaly v knihovnách žádné funkce, tudíž bylo nezbytné je vymyslet. Z důvodu rozsáhlého kódu a možného použití i v jiné části kódu či jiném programu, nebyl převod umístěn přímo do programu, ale do dvou nových funkčních bloků, každého s jiným typem převodu. Tyto funkční bloky se pak jednoduše externě vyvolají v rámci hlavního FB.

5.2.7 Funkce bytearray_to_real

Převod z typu *bytearray* na typ *real* byl vytvořen konkrétně pro převod získané odpovědi sensoru, zakódované v poli typu Byte, na reálné číslo. Převodní funkce byla realizována strukturou case. Způsobů, jak převod provést, je více, např. odečet hodnoty 48 (mohlo by být programátorsky optimálnější, avšak mé řešení je zřetelnější). Konkrétně byl tento převod použit pro již přijaté výsledky měření na konci měřicího cyklu a pro čas, potřebný ke změření hodnot. Ačkoliv tento čas je v podobě celého čísla, ponechat ho v typu Real bylo jednodušší než vytvářet další speciální funkční blok. Zdrojový kód je umístěn v příloze.



Obrázek 23: Funkce bytearray_to_real

5.2.8 Funkce byte_to_integer

Tento FB byl vytvořen pro převod jednociferného celého čísla typu *byte* na *integer*. V našem případě je konkrétně použit pro navrácený počet měření v odpovědi na příkaz započetí měření M! Zdrojový kód FB je možné nalézt v příloze.

Obrázek 24: Funkce byte_to_integer

5.2.9 Rozložení řetězce naměřených hodnot

Z popisu komunikačního protokolu je známo, že sensor odpovídá jedním řetězcem, který obsahuje všechny naměřené hodnoty. Proto bylo nutné ho nějakým způsobem rozdělit na více řetězců, z nichž každý ponese právě jednu měřenou hodnotu. Rozdělení bylo provedeno podle charakteristických znaků, konkrétně znamének „+“ a „-“. Pokud je před číslem znaménko „-“ (Ascii hodnota: 45), nastaví pomocnou proměnnou *minus*, která po převodu udělá z čísla záporné. Poslední číslo je ukončeno tzv. Carriage Return (Ascii hodnota: 13).

```
IF(glob.outBuffer[I]=45) THEN minus:=1; END_IF;
...
IF minus=1 THEN  vysledek:=-vysledek; END_IF;
```

Ukázka kódu 2: Rozlišení záporného čísla

5.2.10 Ukládání výstupních hodnot

Po rozdělení řetězce podle charakteristických znaků jsou jednotlivé části převedeny na hodnoty typu Real (funkce *bytearray_to_real*) a ukládány na výstupy FB. Pokud jsou všechny hodnoty zpracovány, převodní cyklus je ukončen (pomocí příkazu *exit*). Následně program přechází do dalšího, posledního kroku, kde proběhne vynulování vnitřních proměnných a resetování hlavního časovače TON.

```
CASE converted_values OF

      0: out1:=vysledek;
      1: out2:=vysledek;
      2: out3:=vysledek;

END_CASE;

converted_values:=converted_values+1;

IF converted_values=number_of_measures THEN  exit; END_IF;
```

Ukázka kódu 3: Ukládání hodnot na výstupy FB

5.2.11 Ošetření situací, kdy čidlo neodpovídá

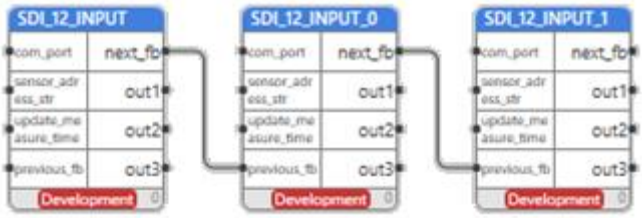
Při testování nastala situace, kdy čidlo nereagovalo. Převodník poté vyslal na sběrnici „No response“. Obdržení a zpracování takové odpovědi by mohlo ohrozit chod programu, proto byla zavedena část kódu, která při obdržení této odpovědi vymaže vstupní i výstupní buffer a ukončí cyklus. Nově vytvořená podmínka reaguje na první dvě písmena „N“ (Ascii hodnota:78) a „o“ (Ascii hodnota:111). Program zareaguje stejným způsobem, pokud dostane odpověď nesmyslnou nebo nedostane vůbec žádnou. Celý zdrojový kód je k dispozici v příloze.

```
IF (glob.outBuffer[0]=78 AND glob.outBuffer[1]=111) THEN
      FOR I:=0 TO glob.outBufferlen BY 1
            DO glob.outBuffer[I]:=0;
      END_FOR;
      glob.outBufferlen:=0;
      com_state:=8;

END_IF;
```

Ukázka kódu 4: Detekce "No Response"

5.2.12 Návaznost bloků

Zde se jedná se o mnou navržený způsob ošetření kolizí na sběrnici. Ty nastávaly zejména při spuštění programu, kdy se všechny použité instance funkčních bloků *SDI_12_INPUT* snaží navázat komunikaci najednou. Mé řešení spočívá ve vzájemném propojení jednotlivých bloků do série (podle priority). První blok vždy zahájí komunikaci, uloží hodnoty a předá pověření dalšímu. Další blok v pořadí může začít komunikovat teprve tehdy, když předchozí skončí.

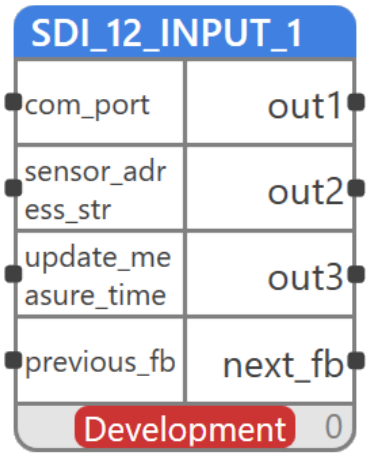


Obrázek 25: Řetězení funkčních bloků

Právě tato část vývoje driveru působila největší komplikace. Mé řešení tohoto problému se nemusí zdát příliš elegantní, ale je funkční a spolehlivé. Původní nápad byl toto předávání pověření nějakým způsobem zakomponovat přímo do funkčního bloku Driveru, např. při každém vytvoření bloku nového mu přiřadit číslo, podle kterého by byl vyvolán, bohužel se toto provedení nedokázalo zrealizovat. Dalším řešením by bylo postavit tzv. „řadič“, další funkční blok, který by postupně předával pověření jednotlivým funkčním blokům. Nevýhodou tohoto řešení je to, že uživatel musí myslet na to, aby bloky spolu správně propojil.

5.3 Finální podoba funkčního bloku driveru

Výstupem práce je jednoduše použitelný funkční blok v knihovně programovacího prostředí Merbon, který zajišťuje komunikaci a čtení podle nastavitelných parametrů. Popis všech proměnných včetně datového typu, který je nutno dodržet, je možné vidět v tabulce 8.



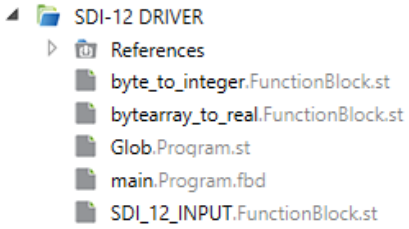
Obrázek 28: Finální podoba funkčního bloku

Tabulka 7: Vstupy a výstupy FB driveru

Vstup	Význam	Datový typ
com_port	Číslo portu	Int
sensor_adress_str	Adresa čidla	String
update_measure_time	Interval měření	Time
previous_fb	Propojení s předchozím FB	Bool

Výstup	Význam	Datový typ
out1, out2, out3	Naměřené hodnoty	Real
next_fb	Propojení s dalším FB	Bool

Výsledný projekt v Merbon IDE se skládá ze souborů, které jsou k vidění na obrázku 31. *Byte_to_integer* a *bytearray_to_real* jsou definice pomocných funkčních bloků. Jejich význam byl vysvětlen v kapitolách 5.3.8 a 5.3.9. Namespace *Glob* obsahuje definici globálních proměnných. Soubor *main* je diagram funkčních bloků, kde uživatel vytváří program již pro konkrétní aplikaci. Právě v této části je možné použít již naprogramované funkční bloky *SDI_12_INPUT*. Jejich definici obsahuje poslední, nejobsáhlejší soubor.



Obrázek 29: Obsah projektu v Merbon IDE

6 Testování funkčnosti

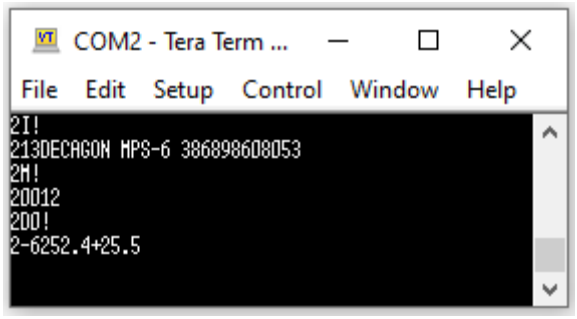
Testování probíhalo od samotného začátku vývoje programu až do úplného konce, před nasazením do reálného provozu. Prováděno bylo na sensoru MPS-6 měřícím vodní tlakový potenciál půdy (kPa) a její teplotu (°C). Později i na dalších poskytnutých sensorech, jako jsou MPS-2, Teros32 a Sentek SDI-12 Series II.

Obrázek 31: Fotografie z prvotního testování

Obrázek 30: Sensor MPS-6 [30]

- **Ověření funkčnosti čidla**

Čidlo bylo nejprve testováno na softwarovém terminálu *Tera Term*, kde můžeme sledovat odezvu čidla. Díky tomu pak můžeme ověřit funkčnost a správnost hodnot na výstupu funkčních bloků. Na obrázku je zobrazena komunikace mezi terminálem a snímačem. Příkazy (konkrétně identifikace, dotaz na měření a požadavek na hodnoty) byly psány do terminálu ručně, po nich následovala vždy odpověď sensoru.

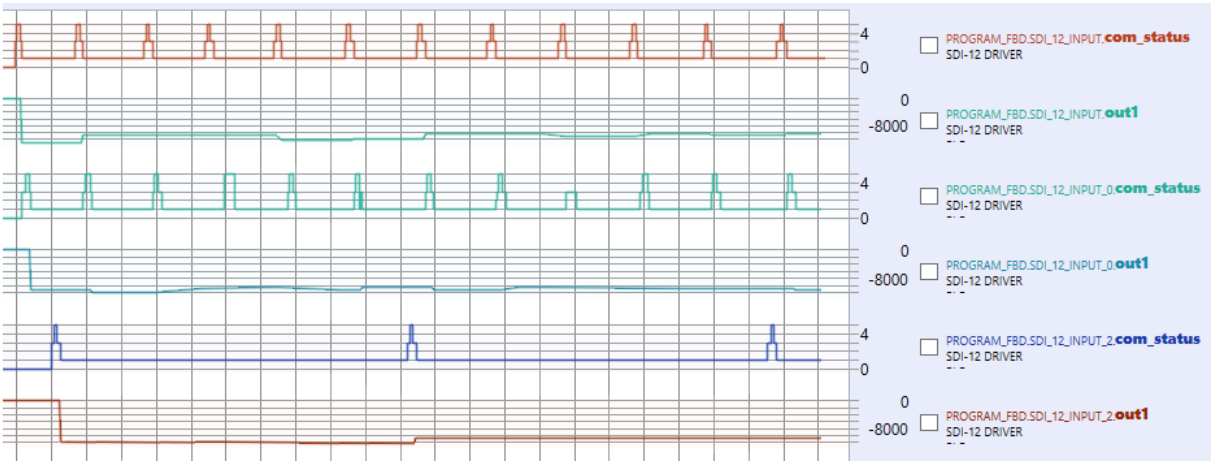


Obrázek 32: Komunikace se sensorem MPS-6 přes terminál

Přes tento terminál je samozřejmě možné i přenastavit adresu čidla podle potřeby, aby se v aplikaci nenacházelo více čidel se stejnou adresou. To se provede samostatným připojením čidla a příslušným příkazem *Change Address Command (aAb!)*.

- **Debugging**

První plnohodnotné testování programu probíhalo připojením čidla a převodníku ke komunikačnímu portu počítače, na kterém byl spuštěn Merbon Runtime (došlo k vytvoření tzv. softPLC, do kterého byl nahrán vytvořený testovací program). Sledované průběhy proměnných z debuggingu je možné vidět v grafu na obrázku 35.



Obrázek 33: Sledování hodnot testovacího programu

V grafu můžeme vidět průběhy měření a změny výstupních hodnot, konkrétně tlakové potenciály. Test proběhl za použití 4 funkčních bloků. Dva z nich mají nastavený stejný intervalem měření 10 sekund. Třetí blok má interval měření 45 sekund. Rovněž byl použit čtvrtý blok, který reakci programu testuje chybně zadanou nebo nezadanou adresu čidla. Při neobdržení odpovědi se činnost tohoto bloku ukončí a následuje další. Debugging není určen pro sledování dat, pouze pro ověření správnosti chodu programu a hledání chyb. Při rychlých změnách proměnných lehce zaostává. To je způsobeno díky velkému rychlostnímu rozdílu mezi chodem programu a vzorkováním hodnot. V grafu pak není vidět hladký průběh se všemi kroky, vidíme pouze ty, na kterých setrval program nejdéle, tzn. čekání na měření. Peaky v grafu značí vždy okamžik probíhající komunikace, měření a uložení hodnoty.

- **Reálné zapojení Hardwarového PLC a čidel**

Další testování bylo provedeno již na 3 rozdílných čidlech. Prvním (nesoucím adresu 1) je Teros32 měřící teplotu a vodní potenciál půdy. Druhým (s adresou 2) je MPS-2 měřící rovněž teplotu a vodní potenciál. Poslední čidlem (adresa 3) je *Sentek SDI-12 Series II Probe* poskytující informace o vlhkosti a slanosti půdy. Měření probíhalo v laboratorních podmínkách, kde nebyly měřeny vlastnosti půdy, ale okolního vzduchu v laboratoři. Interval měření byl nastaven na 15 sekund. Časování můžeme vidět na skenovaném výstupu z terminálu.

```
8:44:54.440> 1M!10015
8:44:56.316> 1D0!1+98.190+18.7+0
8:44:59.005> 2M!20012
8:45:00.877> 2D0!2-21168.0+23.3
8:45:01.566> 3M!30016
8:45:03.594> 3D0!3+0.000000+0.000000+0.000000
8:45:11.755> 1M!10015
8:45:13.630> 1D0!1+98.192+18.7+0
8:45:16.318> 2M!20012
8:45:18.191> 2D0!2-19877.9+23.3
8:45:19.378> 3M!30016
8:45:21.378> 3D0!3+0.000000+0.000000+0.000000
8:45:29.129> 1M!10015
8:45:30.942> 1D0!1+98.189+18.7+0
8:45:33.693> 2M!20012
8:45:35.503> 2D0!2-19877.9+23.3
8:45:37.254> 3M!30016
8:45:39.192> 3D0!3+0.000000+0.000000+0.000000
```

Obrázek 34: Skenovaný výstup terminálu

6.1.1 Tvorba vizualizace

Nejprve je nutné se seznámit s pojmy HMI a SCADA. Ačkoliv se velmi často zaměňují, je důležité znát rozdíl mezi nimi.

HMI – umožňuje jednoduché sledování a řízení procesů na Webserveru PLC nebo dotykovém displeji v rámci lokální sítě LAN

SCADA – umožňuje řízení a sledování procesů, sběr dat, jejich zobrazení a analýzu přes internet

Výstup této práce měl být aplikován konkrétně pro měření parametrů půdy na České zemědělské univerzitě v Praze. Bohužel však nebyla instalace čidel dokončena (ze strany ČZU), proto je zde vidět pouze hrubý návrh HMI. To obsahuje hlavní stranu, která zobrazuje aktuální hodnoty všech 9 použitých čidel a pro každé čidlo speciální stranu, zobrazující změny měřených hodnot v grafu.

Obrázek 35: Návrh prostředí HMI

6.2 Řešení problémů během vývoje

Práci provázela celá řada problémů, které musely být řešeny. Jedním z nich byla např. situace, kdy z převodníku přišla v ASCII zakódovaná zpráva „No Response“. Proměnné závislé na odpovědi čidla pak začaly nabývat nesmyslných hodnot. Vytvořena byla jednoduchá část kódu, která při obdržení „No Response“ celý měřicí proces ukončí a přistoupí k dalšímu funkčnímu bloku. Navíc v tu chvíli vyšlo najevo, že program počítal pouze se smysluplnými odpověďmi nebo žádnými. Jiné by teoreticky nastat neměly, ale pro tento případ jsem ošetřil reakci programu na odpovědi, které by nedávaly smysl. Program na ně nebude reagovat, ukončí měřicí cyklus a žádné proměnné tak neovlivní. Výsledky zůstávají uložené z předchozího měření. Většina dalších problémů vyplývala pouze z neúplné znalosti programovacího jazyka a funkcí, které si bylo potřeba nastudovat. Rovněž vývojové prostředí je pro nové uživatele poměrně složité na obsluhu, např. při nahrávání programu se musí přiřadit program do „Tasku“ PLC, kde může běžet i více programů najednou. Na druhou stranu může projekt pracovat s více PLC. Velmi obtížné bylo vytvořit také část programu, která obsluhuje port, tudíž jeho otevření, zápis, čtení a zavření.

6.3 Možná vylepšení

Stejně tak jako vytvoření funkcí pro převod, by bylo možné více částí programu zakomponovat do funkcí a lépe tak pro přehlednost program strukturovat. Dalším vylepšením by mohlo být praktičtější řešení ošetření proti kolizím. To se však prozatím nepodařilo jiným způsobem udělat.

7 Vyhodnocení

Nyní se práce zaměřuje na třetí bod zadání, a to uvedení praktických poznatků a porovnání s ostatním standardy, konkrétně MODBus a M-Bus. Zde se nabízí otázka: podle kterých kritérií lze tyto standardy porovnávat? Existuje mnoho faktorů. Osobně jsem porovnání rozdělil do dvou kategorií, a to z hlediska hardwarových a softwarových vlastností. V rámci každé z nich jsem pak standardy porovnal podle několika, dle mého názoru, nejdůležitějších vlastností. Nakonec jsem zhodnotil rovněž náročnost uvádění do provozu.

7.1 Vyhodnocení z hlediska hardwarových vlastností

Nejprve byly standardy porovnány podle hardwarových vlastností. Pro přehlednost byla porovnání rozdělena do více bodů.

- **Napájení**

SDI-12 pracuje s nízkým stejnosměrným napětím 12 V, které je běžně dostupné v každém rozvaděči automatizace budov, tudíž z tohoto hlediska je SDI-12 přijatelná. Zdroje 12 V DC jsou na trhu cenově dostupné. V tomhle ohledu je SDI-12 praktičtější než třeba M-Bus, pracující s 36 V. Napájení MODBus protokolu závisí na použité verzi.

- **Kabeláž**

SDI-12, stejně jako MODBus a M-BUS používají sériovou komunikaci, tudíž rozdíl v kabelech není tak závratný. SDI-12 používá 3 vodiče, zatímco oba zmíněné typy mohou pracovat pouze se dvěma vodiči. M-bus má navíc možnost záměny vodičů.

Tabulka 8: Porovnání počtu použitých vodičů mezi standardy [16] [17] [20]

Standard	Počet vodičů
SDI-12	3
MODBus	2–3
M-Bus	2

Další tabulka porovnává maximální délku sběrnice a možný počet připojitelných zařízení. Toto porovnání je velmi sporné, protože s délkou kabelu klesá možný počet připojitelných zařízení. Kromě toho maximální délka kabelu závisí na přenosové rychlosti a samozřejmě na typu kabelu (jeho impedanci, kvalitě a průřezu vodiče).

Tabulka 9: Porovnání maximální délky sběrnice a počtu připojitelných zařízení [16] [18] [20]

Standard	Maximální délka sběrnice	Maximální počet připojených zařízení k jednomu základnímu segmentu
SDI-12 při použití převodníku C1023	60 m	10
MODBus	700-1200 m	32
M-Bus	350-900 m	250

Nabízí se zde otázka, proč je i při takto malé přenosové rychlosti maximální délka kabelu SDI-12 tak nízká? Jedním důvodem může být to, že sběrnice neřeší impedanční přizpůsobení. Dalším může být to, že sběrnice je určená pro nasazení v zemědělství, kde vodiče mohou přijít do styku s agresivním prostředím a s kapalinami, jako je např. močovina. Působící vlhkost prostředí může rovněž vodiče degradovat korozí. Dalším důvodem by mohla být eliminace možného poklesu napětí při napájení sensorů, jelikož napětí 12 V je poměrně malé a při udávané maximálně spotřebě 0,5 A by při použití klasického vodiče 0,5 mm^2 na vzdálenost 60 kleslo o 2,112 V. Na vzdálenost 100 m až o 3,52 V, což by mohlo vést k nefunkčnosti čidla.

- **Spotřeba**

Čidla SDI-12 jsou optimalizována pro minimální spotřebu, jelikož se po většinu času nachází v režimu spánku, ze kterého jsou probuzena vždy, když obdrží příkaz jim adresovaný. Pro automatizaci budov je tento způsob zbytečně složitý, a ačkoliv může ušetřit náklady na provoz, tak ve srovnání s vysokou cenou sensorů SDI-12 se příliš nevyplatí.

- **Cena sensorů**

Jelikož automatizace budov je vysoce konkurenční oblast, tak oproti jiným standardům je cena poměrně vysoká. Z toho důvodu nemá nasazení čidel SDI-12 v praxi příliš smysl. Dále je nutné zakoupit hardwarový převodník pro připojení k portu PLC, který je též poměrně drahý.

- **Komunikační rychlost**

Komunikační rychlost 1200 Baudů je pro použití v automatizaci budov, kde je potřeba vyšší propustnost, poměrně nízká. Při použití v zemědělství, pro které je tato sběrnice určena, je však nízký baudrate vyhovující, díky již zmíněné odolnosti proti rušení.

Tabulka 10: Porovnání komunikačních rychlostí zmíněných standardů [16] [18] [20]

Standard	Komunikační rychlost (Baud)
SDI-12	1200
MODBus	1200 až 11520
M-Bus	300 až 9600

- **Dostupnost periferií**

SDI-12 se prozatím nestala příliš populární a využívanou, a kvůli vysoké konkurenci pravděpodobně ani nestane. Nabídka periferií je proto velmi omezená. Aktuálně jsou dostupné pouze senzory pro půdní měření. Obzvlášť v České republice je dostupnost velmi špatná a většinu čidel je nutno objednat ze zahraničí. S případným rozšířením SDI-12 by se to mohlo zlepšit, nicméně to není příliš pravděpodobné.

7.2 Vyhodnocení z hlediska softwarových vlastností

- **Adresace**

Adresace čidel pomocí jednoho znaku, a to buď písmen nebo čísel, je jednoduchá, ale ne příliš praktická. To posouvá SDI-12 spíše do oblasti laboratorních měření. Celkový počet možných adres je 62 (26 velkých písmen, 26 malých písmen, 10 čísel). V tabulce můžeme znovu vidět porovnání s jinými standardy.

Tabulka 11: Porovnání možného počtu adres zmíněných standardů [16] [18] [20]

Standard	Možný počet adres
SDI-12	62
MODBus	255
M-Bus	250

- **Komunikační protokol**

Komunikační protokol SDI-12 je poměrně jednoduchý a uživatelsky přívětivý. Roli v tom samozřejmě hraje to, že přenášená data od sensorů nejsou složitá a obsáhlá. Zásadní nevýhodou je to, že sběrnice je, alespoň prozatím, omezena pouze na čidla. Neumožňuje tak řízení výstupních periferií, jako jsou např. ventily, relé. To ji činí pro nasazení v automatizaci budov ne příliš vhodnou.

7.3 Oživování a servis

Při prvotní instalaci je nutno sensory adresovat. To je možné buď pomocí univerzálních či výrobcem dodaných programů nebo za použití textového terminálu, kdy uživatel píše příkazy ručně. Nevýhodou je potřeba hardwarového převodníku, který je poměrně drahý. Obecně se postup uvádění čidel do provozu od jiných standardů příliš neliší.

7.4 Možnost použití v jiných oblastech

Z uvedených porovnání vyplývá, že použití SDI-12 v automatizaci budov a jiných oblastech nedává příliš smysl. Téměř ve všech kritériích byl standard překonán vhodnějšími protokoly, jako je MODBus nebo M-Bus. Z toho důvodu SDI-12 zůstává používána pouze pro specifické aplikace v zemědělství, kde, díky odolnosti vůči rušení a možnosti režim spánku, zmíněné sběrnice překonává.

Závěr

V rámci této bakalářské práce se podařilo úspěšně splnit všechny body zadání. Na základě podkladů byl pro standard SDI-12 vytvořen plně funkční komunikační driver. Ten je tvořen funkčním blokem, který je volně použitelný ve vývojovém prostředí Merbon IDE. Uživateli nabízí jednoduchou implementaci a možnost snadného odečtu měřených hodnot. Díky nastavitelným parametrům je rovněž možné pro každý sensor stanovit specifický interval měření přizpůsobený vlastní potřebě.

Teoretická část práce se věnuje některým komunikačním standardům běžně používaným v automatizaci budov. Vysvětluje princip jejich funkce a jejich použití. Rovněž detailně popisuje standard SDI-12, jehož znalost byla nezbytná pro následný vývoj driveru. Provedeno bylo také porovnání zmíněných standardů a SDI-12.

V rámci samotného vývoje byl poměrně složitý měřicí cyklus rozdělen do několika částí, jež byly postupně zpracovávány. Po jejich dokončení byly jednotlivé funkční celky spojeny do jednoho kódu. Program je založený na struktuře Case, která rozděluje celý měřicí cyklus na několik kroků, které jsou vykonávány postupně. Při návrhu byl kladen důraz na spolehlivost, proto byl program důkladně ošetřen proti kolizím na sběrnici nebo vůči zpracování nesmyslných hodnot. Dodatečně byl rovněž vytvořen předběžný návrh jednoduché vizualizace pro konkrétní aplikaci v zemědělství, která však při dokončování této práce ještě nebyla zrealizována.

Hlavním přínosem této práce je zhotovení funkčního, spolehlivého a snadno použitelného nástroje, který zajišťuje správnou a bezchybnou komunikaci čidel SDI-12 s programovatelnými automaty PLC. Toto řešení dokáže uživateli značně zjednodušit práci. Díky tomu již není třeba dlouhé studium standardu SDI-12 a vlastnoruční psaní poměrně složitého kódu, které by zabralo hodiny práce. Navíc byl celý kód psán univerzálně a umístěn do funkčního bloku, který lze v uživatelském programu libovolně používat. FB driveru nabízí vytvoření libovolného množství instancí. Při připojení dalšího čidla uživatel jednoduše přidá do programu další FB. Jejich počet je však omezen pamětí konkrétního PLC. Tento FB je samozřejmě možné použít i v jiném programu či projektu. Prozatím se sice nestal součástí základní knihovny Merbon IDE, je však možné ho snadným způsobem do svého projektu importovat. Práce v příloze obsahuje také dokumentaci driveru s podrobným popisem a návodem k použití.

Řešení bylo důkladně otestováno, případné nedostatky se mohou v provozu ukázat časem.

Seznam použité literatury

[1] Industrial Communication [online]. KUNBUS.com [cit. 2021-5-12]. Dostupné z: https://www.kunbus.com/industrial-communication.html

[2] TEACH COMPUTER SCIENCE: Simplex, Half Duplex, Full Duplex [online]. © 2021 Teach Computer Science [cit. 2021-5-12]. Dostupné z: https://teachcomputerscience.com/simplex-half-duplex-full-duplex

[3] PETERKA, Jiří. Základní formy přenosů [online]. [cit. 2021-5-12]. Dostupné z: https://www.earchiv.cz/a91/a140c110.php3

[4] RS-232, Wikipedie [online]. [cit. 2021-5-12]. Dostupné z: https://cs.wikipedia.org/wiki/RS-232

[5] Síťové modely a architektury [online]. [cit. 2021-5-6]. Dostupné z: http://site.borec.cz/02%20Architektura%20iso%20osi.htm

[6] Systémy používané v "inteligentních" budovách - přehled komunikačních protokolů [online]. © Copyright Topinfo s.r.o. 2001-2021 [cit. 2021-5-12]. Dostupné z: https://vytapeni.tzb-info.cz/mereni-a-regulace/6879-systemy-pouzivane-v-inteligentnich-budovach-prehled-komunikacnich-protokolu

[7] PROFIBUS Technology and Application. [PDF dokument], Karlsruhe, Profibus International Germany, october 2002, [cit. 2021-5-1]. Dostupné z: https://www.profibus.com/

[8] Sítě LAN a WAN [online]. [cit. 2021-5-6]. Dostupné z: http://www.cs.vsb.cz/grygarek/POS/lect/topologie.html

[9] Sériový port RS232 [online]. [cit. 2021-5-6]. Dostupné z*:* https://papouch.com/seriovy-port-rs232-p3740/

[10] Komunikace po průmyslových linkách RS485 a RS422 [online]. © 2019 Papouch.com [cit. 2021-5-12]. Dostupné z: https://papouch.com/komunikace-pro-prumyslovych-linkach-rs485-a-rs422-p3735/

[11] USB. Wikipedia [online]. [cit. 2021-5-10]. Dostupné z: https://en.wikipedia.org/wiki/USB

[12] USB On-The-Go. Wikipedia [online]. [cit. 2021-5-12]. Dostupné z: https://en.wikipedia.org/wiki/USB_On-The-Go

[13] PROFINET University: Network Reference Model [online]. © 2021 PROFINET University [cit. 2021-5-8]. Dostupné z*: https://profinetuniversity.com/industrial-automation-ethernet/network-reference-model/*

[14] The Modbus Organization [online]. Copyright © 2005-2021 Modbus Organization [cit. 2021-5-3]. Dostupné z: https://www.modbus.org/

[15] Ing. Otto Havle, C. M. [2/2009]. Jak na Modbus? Automa: odborný časopis pro automatizační techniku, stránky 46-47. ISSN 1210-9592. Praha: FCC public, 1994-.

[16] Modbus 101 - Introduction to Modbus [online]. © Copyright 2020 Control Solutions Minnesota [cit. 2021-5-12]. Dostupné z: https://www.csimn.com/CSI_pages/Modbus101.html

[17] M-Bus [online]. © M-Bus 2020 [cit. 2021-5-8]. Dostupné z: https://m-bus.com/

[18] M-Bus connection for energy and consumption meters via TwinCAT [PDF dokument]. © Beckhoff Automation GmbH, August 2010 [cit. 2021-5-8]. Dostupné z: https://download.beckhoff.com/download/document/Application_Notes/DK9322-0810-0036.pdf

[19] BACnet Tutorial Overview [online]. [cit. 2021-5-1]. Dostupné z: http://www.bacnet.org/Tutorial/HMN-Overview/sld001.htm

[20] A Serial-Digital Interface Standard for Microprocessor-Based Sensors. Version 1.4', Prepared By SDI-12 Support Group (Technical Committee). USA., July 2019.

[21] PLC Architecture and Types: With Comparison Table [online]. © 2021 | ladderlogicworld.com [cit. 2021-5-12]. Dostupné z: https://ladderlogicworld.com/plc-architecture/

[22] Alannah Thornton: Session III Architecture of PLC [online]. © 2021 SlidePlayer.com [cit. 2021-5-8]. Dostupné z*:* https://slideplayer.com/slide/13936021/

[23] Understanding the PLC Program Scan | Acc Automation [online]. [cit. 2021-5-8]. Dostupné z: https://accautomation.ca/understanding-the-plc-program-scan/

[24] JOHN, Karl-Heinz a Michael TIEGEKAMP. IEC 611311-3: Programming Industrial Automation Systems [online]. New York: Springer, 2010 [cit. 2021-04-05]. 2. ISBN 978-3-642-12014-5. Dostupné z: doi: 10.1007/978-3-642-12015-2

[25] Co by měl každý vědět o programovacích jazycích PLC: Elektro Průmysl [online]. Copyright © 2011 - 2020 ElektroPrůmysl.cz | ISSN 2571-0761 | [cit. 2021-5-1]. Dostupné z: https://www.elektroprumysl.cz/software/co-by-mel-kazdy-vedet-o-programovacich-jazycich-plc

[26] Programování PLC podle normy IEC 61 131-3 v prostředí Mosaic [PDF dokument]. jedenácté vydání, únor 2009 [cit. 2021-5-3]. Dostupné z: https://web.rcmt.cvut.cz/users/cerny/PLC_sup/TXV00321_(v11)_Programovani_PLC_TECOMAT_podle_IEC_61131-3.pdf

[27] Domat Control System - Energie pod kontrolou [online]. [cit. 2021-5-1]. Dostupné z: https://domat-int.com

[28] C1023: Převodník mezi RS232 a SDI-12 - Dokumentace k modulu [PDF dokument]. icMK s.r.o. [cit. 2021-5-1].

[29] SENTEK SDI-12 SERIES II Probe Interface Manual Version 1.1 [PDF dokument]. Copyright © 2001 – 2014 Sentek [cit. 2021-5-8]. Dostupné z: https://www.fondriest.com/pdf/sentek_sdi-12_manual.pdf

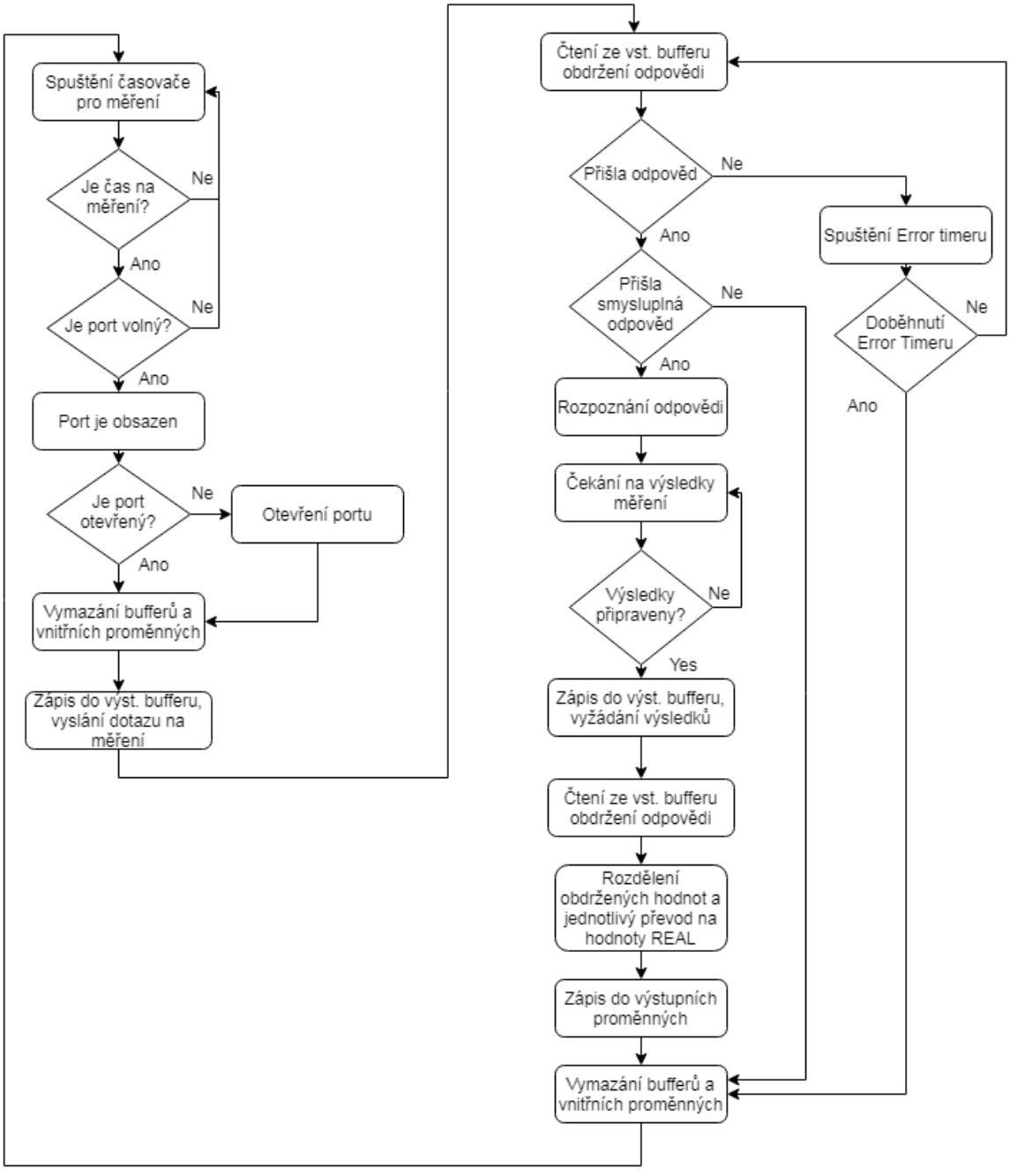
[30] TEROS 21 Water Potential Sensor | ICT International [online]. © Copyright ICT International [cit. 2021-5-1]. Dostupné z: https://www.ictinternational.com/products/mps-6/teros-21-water-potential-sensor/

[31] Electronics Hub: RS232 Protocol – Basics [online]. Copyright © 2021 Electronicshub.org [cit. 2021-5-8]. Dostupné z: https://www.electronicshub.org/rs232-protocol-basics/

Materiály firmy Domat Control System

Přílohy

1. Vývojový diagram znázorňující funkci driveru



2. Zdrojový kód

```
FUNCTION_BLOCK SDI_12_INPUT
    VAR
        com_handle : io.commhandle;
        def:string;
        com_status:int :=0;//pro hlavní case, určuje v jakém stavu se komunikace nachází
        port_status:int;//hlásí stav portu
        read_status:int;//status čtení portu
        write_status:int;//status zápisu do portu
        str_m :string:= 'M!';
        str_d0 :string:= 'D0!'; //stringy v kterých jsou uloženy používané příkazy pro SDI12
        str_received:string;
        str_adress :string :='00';
        str_adress_ref: ref_to string; //reference na adresu
        statement, statement2, adress :bytearray; //ascii hodnoty stringů příkazů a adresy
        I,I2 : int; //pro For cykly
        step: int :=0; //identifikuje kolikátý příkaz se posílá
        byte_pro_prevod: byte;
        posledni_znak_ascii:int;
        number_of_measures: int;
        converted_values: int;
        minus:bool;
        konec_cisla:bool;
        pomocna_adresa:int;
        waiting_timer_IN,
        waiting_timer_OUT: BOOL;  //pro TON
        waiting_time: TIME;
        FB:TON;
        PREVOD:byte_to_integer;
        PREVOD2:bytearray_to_real;
        bytearray_pro_prevod: bytearray;
        vysledek: real;

        //proměnné pro časovaní programu, spouštěni v intervalu měřeni
        Measure_timer:TON;
        Measure_timer_set, Measure_time_counting:TIME;
        Measure_timer_in,Measure_timer_out: BOOL;
        Measuring: BOOL :=1;
        error:bool;
        Error_timer:TON;

    END_VAR
    VAR_INPUT
        com_port:int; //číslo portu
        sensor_adress_str: string; //adresa sensoru
        update_measure_time: TIME; //čas po jaké době se mají hodnoty aktualizovat
        previous_fb: bool; //pro inicializaci, obdržení pověření
    END_VAR
    VAR_OUTPUT
        next_fb: bool; //pro inicializaci, předání pověření
```

```
        out1,out2,out3: real; //výstupy
    END_VAR

      Error_timer(PT:=T#20s);

    //načtení adresy sensoru typu string je nutné pomocí reference
    str_adress:=sensor_adress_str;
    str_adress_ref := ref str_adress;

    IF update_measure_time<T#10s THEN // ošetření proti kolizi při moc krátkém casu
    update_measure_time:=T#10s;
    END_IF;

    IF previous_fb=1 THEN

    Measure_timer_set:=update_measure_time; //časovač pro spouštění intervalu měření
    Measure_timer(IN:=Measure_timer_in,PT:=Measure_timer_set,Q=>Measure_timer_out,ET=>Measure_
time_counting);

    IF Measure_timer_out=1 THEN //povolení programu podle časovače
    IF glob.port_busy=0 THEN
    Measuring:=1;
    glob.port_busy:=1;
    Measure_timer_in:=0;
    Measure_timer_out:=0;
    Error_timer.IN:=true;
    END_IF;
    END_IF;

    IF Measuring=1 THEN

FB(IN:=waiting_timer_IN,PT:=waiting_time,Q=>waiting_timer_OUT); //časovač pro čekání na
hodnoty, mezi příkazy M a D0

//zjištění statusu portu při každém cyklu
port_status:=io.getportstatus(com_handle);
//Výstup: 0 (OK) 25 (nepodařilo se otevřít port) 31 (empty - prázdný buffer) 34 (busy - jsou
odvysílávána data)

    if (glob.outBuffer[0]=78 AND glob.outBuffer[1]=111) THEN //když přijde odpověď NO Response
        FOR I:=0 TO glob.outBufferlen BY 1
                    DO
                        glob.outBuffer[I]:=0;
                    END_FOR;
                    glob.outBufferlen:=0;
                    com_status:=8;

    end_if;

    if (glob.outBuffer[0]>100 OR glob.outBuffer[1]>100 OR glob.outBuffer[2]>100 ) THEN //když
přijde nesmysl..
        FOR I:=0 TO glob.outBufferlen BY 1
```

```
                    DO
                        glob.outBuffer[I]:=0;
                    END_FOR;
                    glob.outBufferlen:=0;
                    com_status:=8;

    end_if;

    if Error_timer.Q=true then
        FOR I:=0 TO glob.outBufferlen BY 1
                    DO
                        glob.outBuffer[I]:=0;
                    END_FOR;
                    glob.outBufferlen:=0;
                    com_status:=8;

    end_if;

CASE com_status of

    0: //otevření portu + ověření komunikace
     IF glob.port_opened=1 THEN
     com_status:=1;

    ELSE
    glob.serial := concat(concat('serial:', lint_to_string(com_port)), ':9600,8,N,1');
//concat sloučí řetězce
              com_handle := io.openport(glob.serial);
    if com_handle >= 0 then
                com_status := 1; glob.port_opened:=1;

            end_if;

    END_IF;

    ////////////////////////////////////////////////////////////////////////////////////////
    1: //vymazání bufferů, poté sestavení příkazu a uložení do buffery k odeslání

                    FOR I:=0 TO 5 BY 1
                    DO
                        glob.inBuffer[I]:=0;
                    END_FOR;
                    glob.inBufferlen:=0;


                    FOR I:=0 TO glob.outBufferlen BY 1
                    DO
                        glob.outBuffer[I]:=0;
                    END_FOR;
                    glob.outBufferlen:=0;
```

```
        case step of // druhý case pro nastavování inBufferu - převody stringů na Bytearray
                     //0-dotaz na měřeni, 1-send data command

                0://nastaveni pro vyslani dotazu na mereni ...M!
                  statement:=string_to_bytes(str_m, 2);
                 glob.inBufferlen:=3;

                1://nastaveni pro odeslani prikazu pro mereni ...D0!
                 statement:=string_to_bytes(str_d0, 3);
                 glob.inBufferlen:=4;

            end_case;

    adress:=string_to_bytes(str_adress_ref, 1); //převod adresy na typ BYTE
    glob.inBuffer[0]:=adress[0]; //uložení adresy do bufferu k odeslání

    FOR i:= 1 TO (glob.inBufferlen-1) DO
    glob.inBuffer[i] := statement[i-1]; //uložení příkazů do bufferu k odeslání
    END_FOR;

    com_status:=2;

    ////////////////////////////////////////////////////////////////////////////////////
    2: //vysilani

            if (port_status = 0 or port_status = 31) and glob.inBufferLen > 0 then
                if io.writeport(com_handle, adr glob.inBuffer[0], glob.inBufferLen) then
                    com_status := 2;
                    // Shození vstupního bufferu
                    glob.inBufferLen := 0;
                end_if;
            elsif (port_status <> 0) then
                com_status := 99; //chyba
            else
            com_status := 3
            end_if;


    3: //cteni
            if glob.inBufferLen > 0 then
                glob.outBufferLen := 0;
            elsif (glob.outBufferLen < 40) then
                glob.outBufferLen := glob.outBufferLen + io.readport(com_handle, adr
glob.outBuffer[glob.outBufferLen], 40 - glob.outBufferLen);
            end_if;


            if (glob.outBuffer[glob.outBufferlen-1]=10)
            then com_status:=4;
```

```
            end_if;

            4: //rozhodnutí co dělat po obdržení odpovědi

                case step of //case pro rozhodnuti co bude dále dělat s odpovědÍ
                //0-dotaz na měřeni, 1-send data command

                0:                  //vymazání bytearray před převodem
                  FOR I:=0 TO 10 BY 1
                    DO
                        IF bytearray_pro_prevod[I]=0 THEN
                            EXIT;
                        END_IF;

                        bytearray_pro_prevod[I]:=0;
                    END_FOR;

                byte_pro_prevod:=glob.outBuffer[4]; //počet měření
                bytearray_pro_prevod[0]:= glob.outBuffer[1];
                 bytearray_pro_prevod[1]:= glob.outBuffer[2];
                bytearray_pro_prevod[2]:= glob.outBuffer[3];
                PREVOD(in:=byte_pro_prevod, out=>number_of_measures);
                PREVOD2(in:=bytearray_pro_prevod, out=>vysledek);

                waiting_timer_IN:=1;    //spuštění časovače čekání
                waiting_time:= mul_time_real(T#1s, vysledek); //nastavení času čekání

                com_status :=5;

                1: com_status:=6;
                end_case;


            5: //čekání než bude měření připravené

                  if(waiting_timer_OUT=1) then com_status:=1; step:=1; waiting_timer_IN:=0;
end_if;

        //měření připraveno, návrat na začátek case,  vysílaní nového příkazu

            6:
                      FOR I:=0 TO 10 BY 1 //vynulovaní pole
                    DO
                        IF bytearray_pro_prevod[I]=0 THEN
                            EXIT;
                        END_IF;

                        bytearray_pro_prevod[I]:=0;

                    END_FOR;
```

```
    if(glob.outBuffer[1]=45) then minus:=1; end_if; //minus
        com_status:=7;


////////////////////////////////////////////////////////////////////////////////

7:
        FOR I:=2 TO glob.outBufferlen-1 BY 1 DO

            IF (glob.outBuffer[I]>47 and (glob.outBuffer[I]<59)) THEN //číslo
            bytearray_pro_prevod[pomocna_adresa]:=glob.Outbuffer[I];
          pomocna_adresa:=pomocna_adresa+1;
          ELSE
          if(glob.outBuffer[I]=46) then //téčka
                bytearray_pro_prevod[pomocna_adresa]:=glob.Outbuffer[I];
             pomocna_adresa:=pomocna_adresa+1;
          else

          konec_cisla:=1;

          end_if;
            END_IF;

            IF konec_cisla=1 THEN

                PREVOD2(in:=bytearray_pro_prevod, out=>vysledek); //zavolaní fb
                IF minus=1 THEN
                    vysledek:=-vysledek;
                END_IF;
                CASE converted_values OF
                    0: out1:=vysledek;
                    1: out2:=vysledek;
                    2: out3:=vysledek;
                END_CASE;
                converted_values:=converted_values+1;
                pomocna_adresa:=0;
                konec_cisla:=0;
                minus:=0;

                FOR I2:=0 TO 10 BY 1 //vynulovaní pole
                  DO
                      IF bytearray_pro_prevod[I2]=0 THEN
                          EXIT;
                      END_IF;

                  bytearray_pro_prevod[I2]:=0;

              END_FOR;

                if(glob.outBuffer[I]=45) then minus:=1; end_if;

            END_IF;
```

```
            IF converted_values=number_of_measures THEN

        //zpracovaných výsledků=očekávaných vysledků
                            exit; //->ukončení cyklu for
                            END_IF;

                        END_FOR;
            com_status:=8;
            ////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////

            8: com_status:=1;
            step:=0;
            glob.port_busy:=0;
            Measuring:=0;
            next_fb:=1;
            Error_timer.IN:=false;
            converted_values:=0;
            minus:=0;
            pomocna_adresa:=0;

            FOR I:=0 TO glob.outBufferlen BY 1
                    DO
                        glob.outBuffer[I]:=0;
                    END_FOR;
                    glob.outBufferlen:=0;

        IF Measure_timer_in=0 THEN //znovu start počítaní - musí byt tady aby
projel cely cyklus s nulovým vstupem, jinak se nevynuluje
            Measure_timer_in:=1;
            END_IF;

            99: com_status:=0; glob.port_opened:=0; io.closeport(com_handle);

        else
            io.closeport(com_handle);
            com_status := 0;
            glob.port_opened:=0;

END_CASE;

END_IF;

END_IF;

END_FUNCTION_BLOCK
```

```
NAMESPACE glob
    VAR_GLOBAL
        serial: string;
        inBuffer: array[0..7] of byte;
        inBufferlen : int :=0    ;
        outBuffer: array[0..20] of byte;
        outBufferlen: int :=0 ;

        port_busy: bool :=0;
        port_opened: bool :=0;
    END_VAR
END_NAMESPACE

FUNCTION_BLOCK bytearray_to_real

    VAR
        num_ascii : int; //ascii hodnota čísla
        num_i: int; //převedené číslo z ascii
        I: int; //pomocná proměnná pro FOR cykly
        numbers :array[0..10] of int; //pole převedených čísel
        number: real; //výsledné číslo, postupně tvořené přičítáním jeho částí z
pole
        number_help: real; //pomocná proměnná pro desetiny, setiny....
        order: int; //řád čísla, pokud má desetinou část, pokud ji nemá tak je roven nule
        div: int; //pomocná proměnná, pro dělení čísel podle řádu
        length: int;
    END_VAR
    VAR_INPUT
        in : bytearray;
    END_VAR
    VAR_OUTPUT
        out : real;
    END_VAR
 //nulovaní
I:=0;
order:=0;
div:=10;
length:=0;
number:=0;

FOR I:=0 TO 10 BY 1
    DO    numbers[I]:=0;
END_FOR;
                    //cyklus pro převádění ascii hodnot na reálné číslo
FOR I:=0 TO 10 DO

num_ascii:=in[I];

case num_ascii of
```

```
48: num_i:=0;
49: num_i:=1;
50: num_i:=2;
51: num_i:=3;
52: num_i:=4;
53: num_i:=5;
54: num_i:=6;
55: num_i:=7;
56: num_i:=8;
57: num_i:=9;

46: order:=I; //tečka
0: length:=I; EXIT; //nastaveni délky

END_CASE;

IF num_ascii<>46 THEN //zapsání čísla do pole
numbers[I]:=num_i;
END_IF;


END_FOR;
 // násobeni a dělení podle radu a sestavovaní výsledku
//celá část čísla
if order  <> 0 THEN
FOR I:=0 TO (order-1) DO
number:= number*10 +  numbers[I];

END_FOR;

//desetinná část čísla
FOR I:=(order+1) TO (length-1) DO
number_help:=numbers[I];
number_help:=number_help/div;
number:=number+number_help;
div:=div*10;
END_FOR;

end_if;

//pokud je číslo celé, bez desetinné části
if order  = 0 THEN
FOR I:=0 TO (length-1) DO
number:= number*10 +  numbers[I];

END_FOR;
end_if;

out:=number; //konečné přiřazení

END_FUNCTION_BLOCK
```

```
FUNCTION_BLOCK byte_to_integer


    VAR
        num : int;
    END_VAR
    VAR_INPUT
        in : byte;

    END_VAR
    VAR_OUTPUT
        out : int;
    END_VAR

num:=in;

case num of

49: out:=1;
50: out:=2;
51: out:=3;
52: out:=4;
53: out:=5;
54: out:=6;
55: out:=7;
56: out:=8;
57: out:=9;
48: out:=0;

END_CASE;

END_FUNCTION_BLOCK
```