KATEDRA ELEKTRICKÝCH POHONŮ A TRAKCE

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA ELEKTROTECHNICKÁ

ŘÍZENÍ POHONU VÝTAHU POMOCÍ PLC

BAKALÁŘSKÁ PRÁCE

LEDEN 2023

JOSEF VESELÝ

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Veselý** Jméno: **Josef** Osobní číslo: **492141**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra elektrických pohonů a trakce**

Studijní program: **Elektrotechnika, energetika a management**

Specializace: **Aplikovaná elektrotechnika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Řízení pohonu výtahu pomocí PLC

Název bakalářské práce anglicky:

Control of Pasasnger Lift on PLC

Pokyny pro vypracování:

1) Popište komunikační standarty používané v automatizaci
2) Navrhněte logiku řízení pro výtah 5 podlažního domu
3) Do logiky řízení zahrňte funkce paměti zvolených pater a logiku zastavování podle směru jízdy
4) Řízení implementujte na modelu výtahu s PLC Simatic S1200

Seznam doporučené literatury:

dodá vedoucí

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Jan Bauer, Ph.D. katedra elektrických pohonů a trakce FEL

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

Datum zadání bakalářské práce: **13.09.2022** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **19.02.2024**

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

III. PŘEVZETÍ ZADÁNÍ

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

Datum převzetí zadání	Podpis studenta

PODĚKOVÁNÍ

Poděkování patří vedoucímu práce doc. Ing. Janu Bauerovi, Ph.D. za odborné rady a vedení této práce, také mému otci za technickou a psychickou podporu, která vedla ke zvýšení kvality závěrečné práce. Též bych chtěl poděkovat rodině a přátelům za neustálou podporu nejen během psaní této bakalářské práce ale celého průběhu studia.

PROHLÁŠENÍ

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 10. ledna 2023

.....

ABSTRAKT

Hlavním účelem této bakalářská práce je seznámit se s programovatelným logickým automatem firmy Siemens a prostředím TIA portal. Výsledným produktem této bakalářské práce je program umožňující logické řízení modelu výtahu a následná implementace na PLC Simatic S1200, které řídí servopohon výtahu. Součástí práce je i stručná teorie vztahující se k použitému typu pohonu, frekvenčním měničům, PLC a jeho programování a komunikačním standardům.

Klíčová slova: PLC S1200, TIA Portal, SCL, servomotory, výtah

ABSTRACT

The main purpose of this bachelor thesis is to get acquainted with the Siemens programmable logic automaton and the TIA portal software. The final product of this bachelor thesis is a program that enables the logical control of an elevator model through a Simatic S1200 PLC. This program is then implanted on the elevator model controlled by the servo. The thesis also includes a brief theory related to the used type of drive, frequency converters, PLC and its programming and communication standards.

Keywords: PLC S1200, TIA Portal, SCL, Servomotors, Lift

OBSAH

SEZNAM OBRÁZKŮ

ÚVOD

Tato bakalářská práce pojednává o řízení pohonu výtahu pomocí průmyslového logického automatu. V dnešním světě, který je silně závislý na technologiích, nacházejí systémy řízené průmyslovými počítači široké uplatnění. Díky své univerzálnosti, snadnému servisu a dalším vlastnostem nahradila PLC mechanická relé a používají se především v průmyslové automatizaci, pro dopravní světelnou signalizaci nebo i výtahy.

V tomto projektu budeme využívat PLC firmy Siemens z řady Simatic S1200 k řízení servopohonu, který uvádí do pohybu model výtahu pro pětipodlažní dům. První část závěrečné práce se věnuje servomotorům, frekvenčním měničům nezbytných v dnešních aplikacích, PLC a s ním spojeným pojmům a komunikačním standardům. Druhá část práce popisuje program vytvořený v softwaru TIA Portál, který umožňuje logické řízení modelu výtahu. Za účelem zlepšení uživatelské přívětivosti jsme k použité sestavě byla připojena dvojici HMI panelů, která umožňuje stisknutí tlačítek ve výtahu a jednotlivých podlažích.

V Praze dne 10.01.2023

Josef Veselý

KAPITOLA 1: ELEKTROMOTORY

Zařízení sloužících k elektromechanické přeměně energie neboli elektrických strojů existuje několik druhů. Základními typy jsou synchronní, asynchronní a stejnosměrné stroje, které nalezneme na různých místech v různých aplikacích. Nicméně existují práce například v průmyslu nebo v domácnostech, pro které tyto stroje pro všeobecné použití nemají vhodné vlastnosti. Prudký rozvoj v oblasti výkonové elektroniky dal za vznik speciálním typům elektrických motorů, které díky vhodným úpravám v jejich konstrukci mohou tyto specifické práce vykonávat. Mezi takové elektrické motory řadíme i servomotory. [1]

1.1 Servomotory

Přesný význam termínu „servo“ v kontextu elektrických pohonů lze těžko vystihnout. Obecně řečeno, pokud je součástí popisu pohonu „servo“, obsahuje uzavřenou smyčku a je určen pro zpětnovazební řízení. Servopohon prostřednictvím uzavřené smyčky umožňuje řízení točivého momentu, rychlosti nebo pozice.

První servomechanismy vznikly pro aplikace ve vojenském průmyslu. Nejprve se používaly běžné DC motory, které se ale brzy ukázaly jako nevhodné pro přesné řízení zejména z důvodu, že bylo zapotřebí vysokého poměru točivého momentu ku setrvačnosti bez zvlnění momentu. Motory byly tedy dále vyvíjeny tak, aby splňovaly tyto požadavky, až vznikly speciální typy elektrických motorů. Stejně jako u pohonů pro všeobecná použití, i na trhu servopohonů došlo možná k ještě k výraznějšímu odklonu od stejnosměrných pohonů ve prospěch motorů s permanentními magnety a asynchronních motorů, ačkoliv stejnosměrné serva si udržely některé specifické aplikace. [2]

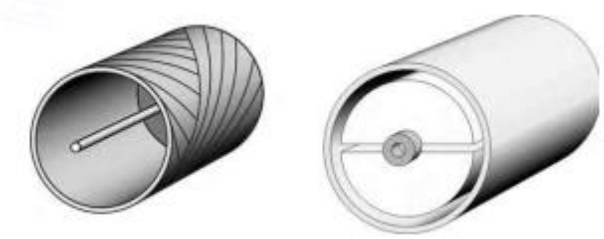
1.1.1 DC Servomotory

Přestože neexistuje pevně stanovená hranice mezi tím, co je servomotor a běžný motor, servomotory jsou určeny pro aplikace, které si vyžadují rychlé rozběhy a zastavení. Proto jsou motory konstruovány tak, aby odolávaly mnohonásobně větším přerušovaným proudům (tudíž i momentu) než jsou jejich jmenovité hodnoty. Protože většina servomotorů je menších rozměrů, jsou jejich kotevní odpory poměrně vysoké. Proud nakrátko při plném napětí v kotvě dosahuje pouze několikanásobku jmenovitého proudu. Tato skutečnost umožňuje velice rychlý rozběh motoru z klidového stavu. [2] Takový provozní stav je pro velké stejnosměrné stroje neproveditelný, což vysvětluje napěťová rovnice stejnosměrného motoru:

$$U = R_a I_a + U_i. \tag{1-1}$$

Stejnosměrné motory nesmí být spouštěny přímým připojením na zdroj, jelikož při nulových otáčkách je indukované napětí U_i rovné nule a odpor kotvy $R_a$velmi malý. To by při nesníženém svorkovém napětí znamenalo obrovský proudový ráz a došlo by k nevratnému poškození motoru. U servomotorů však díky vyššímu kotevnímu odporu je takový rozběh zcela běžný.

Za účelem maximalizace zrychlení je třeba minimalizovat setrvačnost rotoru. V takovém motoru by se měl pohybovat pouze elektrický obvod na rotoru a magnetická část zůstává nehybná. Tento princip je adoptován u motorů s rotorem bez železného jádra a tištěnou kotvou. U motorů s rotorem bez železa nebo pohyblivou cívkou (Obr. 1-1) jsou vodiče kotvy tvořené tenkostěnným válcem, který se skládá pouze z lakovaných drátů vinutých šikmo. Uvnitř kotvy je umístěn dvoupólový permanentní magnet, který zajišťuje radiální tok, a vně je ocelový válcový plášť, který uzavírá magnetický obvod. Absence drážek pro podporu vinutí kotvy má za následek poměrně křehkou konstrukci, která je proto omezena na průměry nepřesahující 1 cm. Vzhledem k malým rozměrům se často označují jako mikromotory. Užívají se ve fotoaparátech, kamerách, čtečkách karet atp. [2]



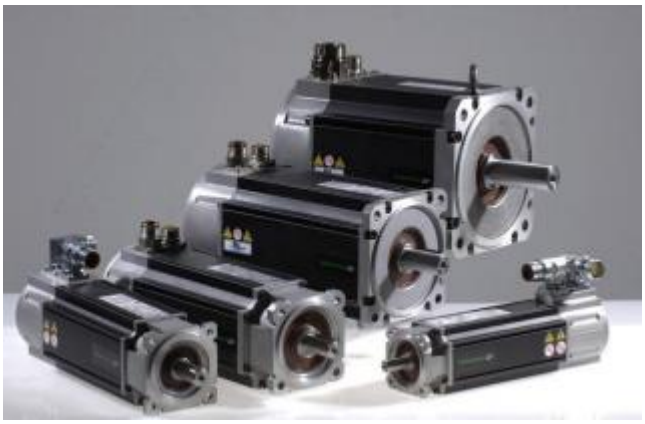
Obr. 1-1 Rotor stejnosměrného motoru bez železného jádra [2]

Typ motoru s tištěnou kotvou je výrazně robustnější a vyrábí se ve velikostech až do několika kW obvykle ve tvaru kotouče nebo placky. Protože je v kotvě typicky nejméně 100 vodičů, zůstává točivý moment při otáčení rotoru téměř konstantní, což umožňuje velmi plynulé otáčení i při nízkých otáčkách. Zároveň jsou setrvačnost a indukčnost kotvy nízké pro rychlou dynamickou odpověď. Díky svému krátkému tvaru a velkým průměrům jsou tyto motory vhodné pro aplikace jako obráběcí stroje a diskové pohony, kde je omezen axiální prostor. [2]

Výhodami stejnosměrných servomotorů jsou vysoký poměr točivého momentu vůči setrvačnosti a velký rozběhový moment. AC motory vynikají svou spolehlivostí a menší hlučností, jelikož u bezkomutátorové konstrukce nedochází k mechanickému kontaktu, tudíž hlučnosti a opotřebení kartáčů.

1.2 Synchronní motory s permanentními magnety

Dále se přesuneme k synchronním motorům a jejich aplikaci v oblasti střídavých servomotorů. Vzhledem k různorodosti konstrukce a širokému použití synchronních motorů v polohovacích aplikacích a aplikacích s proměnnou rychlostí, zejména synchronních motorů s permanentními magnety pro přesné polohování (servopohony) v kombinaci s regulátory otáček výkonové elektroniky jsou charakteristiky motorů a výkonové rovnice obvykle popsány specifickým způsobem, s nímž se v tradiční teorii synchronních motorů příliš nesetkáme. [4]



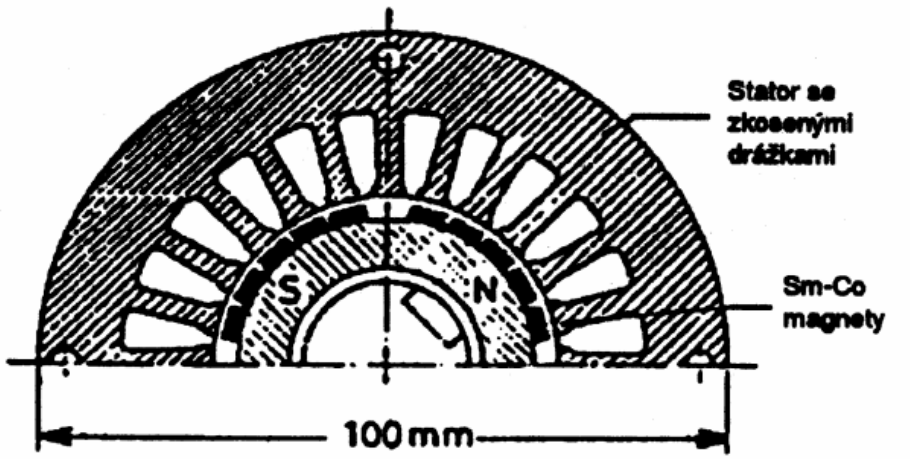
Obr. 1-2 Servomotory s permanentními magnety [2]

Permanentní magnet lze využít jako zdroj magnetomotorického napětí, čímž nahrazuje funkci budicího vinutí stejnosměrných i střídavých synchronních motorů. [3] Kromě podstatného zjednodušení motoru mají i další výhodu, neboť nevyžadují budicí výkon pro vytvoření magnetického toku v magnetickém obvodu stroje. Motor pracuje s podstatně nižším účiníkem než srovnatelný asynchronní motor, protože neodebírá ze sítě magnetizační proud. Navíc v rotoru nevznikají ztráty ani v budicím vinutí jako u klasického synchronního motoru, ani v rotorové kleci jako u asynchronního motoru. Důsledkem je, že motor o stejném výkonu má podstatně menší rozměry než klasický asynchronní motor a lepší účinnost. [5] Vysoká účinnost je žádoucí např.

u pomaluběžných regulovaných pohonů. Motory s permanentními magnety, které mají rotory s nízkou setrvačností, nacházejí široké uplatnění ve vysoce výkonných servo-aplikacích, kde jsou vyžadovány rychlé a přesné pohyby spíše než účinnost motorů. [2]

1.2.1 Konstrukce PMSM

Na obrázku 1.2. můžeme vidět řez typickým provedením motoru s PM. Stator má shodné provedení jako běžný asynchronní motor, tedy třífázové vinutí se shodným počtem pólů jako má rotor. Obvykle se používá provedení se šikmým drážkováním statoru. [5] Na rotoru se střídají póly tvořené permanentními magnety, které mají vysoká sycení od 1 T do 1,2 T. Velice dobrými materiály pro permanentní magnety se ukázaly být vzácné zeminy SaCo nebo NdFeB. Existují různá konstrukční řešení rotoru od permanentních magnetů upevněných na jhu rotoru v místech pólových nástavců až po příčně umístěné permanentní magnety v drážkách rotoru. Poměrná permeabilita materiálu permanentních magnetů je vysoká: μ = 1. Pro prvé provedení rotoru platí $L_d = L_q$. Proto lze pro náhradní schéma PMSM použít náhradní schéma synchronního stroje s hladkým rotorem viz Obr. 1-3. [3]



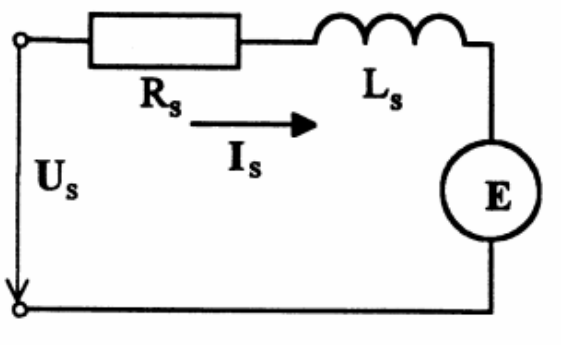
Obr. 1-3 Příčný řez synchronním motorem s permanentními magnety 1,2 kW [5]

1.2.2 Napájení motoru

Zatímco synchronní motory, jejichž rotory obsahují budicí a rozběhové vinutí, mohou být rozbíhány asynchronně, konstrukce motorů s permanentními magnety tento rozběh neumožňuje. Proto je možné tento typ motoru rozbíhat a řídit pouze v kombinaci s napájením z frekvenčního měniče obvykle s napěťovým meziobvodem s šířkovou pulzní modulací. Nelze použít běžný frekvenční měnič, ale frekvenční měnič se speciálním softwarem určeným pro řízení synchronního motoru s permanentními magnety, který má výstupní sinusový proud, vysokou spínací frekvenci a proudové regulátory výstupních proudů. [3]

1.2.3 Řízení motoru

Řízení pohonu je podobné vektorovému řízení asynchronního motoru s tím rozdílem, že poloha rotoru zde slouží jako vztažný úhel. Základní princip řízení vychází z náhradního schématu pro jednu fázi, který se nachází na Obr. 1-4, kde R_S je odpor jedné fáze statorového vinutí, L_S je náhradní indukčnost reakce kotvy, E je napětí indukované magnetickým tokem Φ_f permanentních magnetů a U_S je fázové napájecí napětí. [5]



Obr. 1-4 Náhradní schéma motoru s permanentními magnety [5]

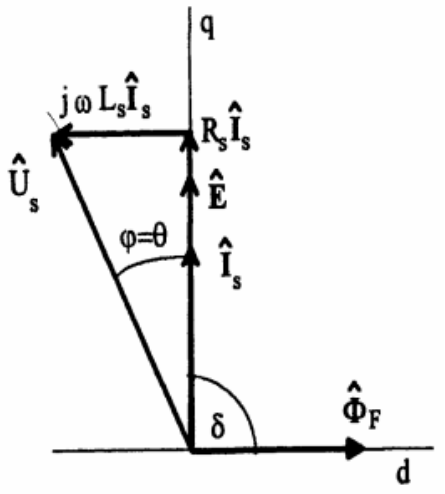
$$(R_S + j\omega L_S)I_S + E = U_S. \tag{1-2}$$

$$E = j\frac{1}{\sqrt{2}}\omega\Phi_f. \tag{1-3}$$

U synchronních motorů s permanentními magnety se používají dva principy řízení, přičemž rozhodujícím kritériem pro volbu způsobu řízení je okamžitá rychlost otáčení. [5]

1.2.3.1 Řízení v oblasti nízkých otáček

Pro provozní stav s nízkými otáčkami platí, že indukované napětí E je úměrné elektrické úhlové rychlosti ω, jelikož magnetický tok Φ_f permanentních magnetů lze považovat za konstantní ve zvolené soustavě os (d, q). Fázor $\hat{\phi}_f$ leží v ose d a proudu $\hat{I}_S$ v ose q. Fázový posun fázoru napájecího napětí $\hat{U}_S$ vzhledem k fázoru $\hat{E}$ je dán parametry R_S, L_S a velikostí E a I_S. Velikost U_S určíme rovněž z fázorového diagramu. [5]



Obr. 1-5 Fázorový diagram pro oblast nízkých otáček $n < n_N$ [5]

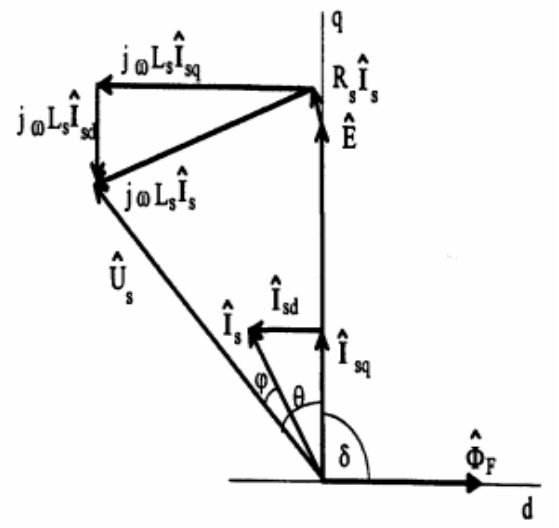
Pro vyvinutí maximálního momentu motoru musí být fázory toku $\hat{\phi}_f$ a statorového proudu $\hat{I}_S$ na sebe kolmé, což je patrné ze vzorce pro moment synchronního stroje.

$$M = \frac{3\,p_p}{\omega_1}\left(\frac{UU_{ief}}{X_d}\sin\beta + \frac{U^2}{2}\left(\frac{1}{X_q} - \frac{1}{X_d}\right)\sin 2\beta\right). \tag{1-4}$$

Pokud se jedná o motor s téměř shodnou příčnou a podélnou reaktancí, nevzniká pak reluktanční moment a v rovnici zůstane pouze synchronní složka momentu, která je přímo úměrná součinu napájecího napětí, vnitřního napětí U_{ief}, sinu zatěžovacího úhlu β a nepřímo úměrná podélné reaktanci X_d. Největšího momentu je tedy dosaženo při $\beta = \pi/2$.

1.2.3.2 Řízení v oblasti vysokých otáček

Pokud je požadováno vyšší úhlové rychlosti v situaci, kdy již není možné dále navyšovat napájecí napětí, které je dodáváno zdrojem, navýšení je možné provést metodou odbuzování. Jelikož magnetický tok nelze regulovat přímo, protože motory s permanentními magnety neobsahují budicí vinutí, odbuzování se dosahuje nepřímou metodou. Ta spočívá v zavedení složky proudu I_S v záporném směru osy d. Taková situace je zobrazena na Obr. 1-6. [5] V důsledku zavedení této složky proudu I_S dojde k natočení fázoru statorového proudu $\hat{I}_S$ vůči ose q. Složka I_{Sq} vytváří, stejně jako v oblasti nízkých otáček moment motoru M.



Obr. 1-6 Fázorový diagram pro oblast vysokých otáček $n > n_N$ [5]

Podobně rozložíme i úbytky na odporu R_S a indukčnosti L_S do směru osy d a osy q. Tento způsob řízení je možný, pokud jsou splněny následující podmínky: První podmínkou je, že modul fázoru napájecího napětí $\hat{U}$ musí mít velikost, kterou je schopen napájecí zdroj dodat. Dále musí být dodrženo proudové omezení, dané dimenzováním statorového vinutí. [5] Musí platit rovnice

$$I_S = \sqrt{I_{\mathrm{Sd}}^2 + I_{\mathrm{Sq}}^2} \leq I_{\mathrm{SMAX}}\,, \tag{1-5}$$

kde I_{SMAX} je maximální přípustný statorový proud. Při odbuzování dochází ke snižování momentu na hřídeli, jelikož při dodání složky I_{Sd} musí dojít ke snížení velikosti složky I_{Sq}.

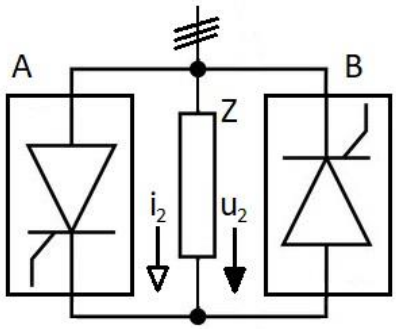
Z výše uvedených kapitol o řízení vyplývá, že při požadavku na zvyšování úhlové rychlosti řídíme velikost a fázi napájecího napětí U tak, aby složka statorového proudu $I_{\mathrm{Sd}} = 0$ a I_{Sq} byla menší než I_{SMAX}, pokud není dosaženo maximálního napájecího napětí. Následné navyšování úhlové rychlosti je možné natáčením fázoru proudu $\hat{I}_S$ při udržování stálého napájecího napětí.

KAPITOLA 2: FREKVENČNÍ MĚNIČE

Nedílnou součástí elektrického pohonu je polovodičový měnič. Frekvenční měnič umožňuje přeměnu střídavé energie se vstupním kmitočtem na střídavou energii jiného kmitočtu. Používají se pro řízení asynchronních i synchronních pohonů, kde díky vlastnostem frekvenčního měniče jsme schopni pohony plynule rozbíhat bez proudového nárazu a řídit v požadovaném pracovním pásmu otáček od velice nízkých otáček, i několika otáček za minutu až po extrémně vysoké rychlosti v řádech desítek tisíc otáček za minutu. Kromě využití v oblasti elektrických pohonů si našly frekvenční měniče své místo také například v energetice, kde dovolují spojovat nesynchronizované střídavé sítě nebo se užívají pro indukční ohřev. Často je zároveň s řízením frekvence požadováno i řízení výstupního napětí tak, aby byl v motoru zachován konstantní magnetický tok. Měniče frekvence dělíme na přímé a nepřímé. [6]

2.1 Přímý měnič frekvence

Přímý měnič frekvence se také často označuje jako cyklokonvertor. Mění vstupní napájecí frekvenci na výstupní frekvenci přímo bez stejnosměrných meziobvodů. Pro jednofázové zapojení se jedná o dva řízené tyristorové usměrňovače, které jsou zapojeny antiparalelně, viz Obr. 2-1. Změnou řídicího úhlu α jsme schopni v tomto zapojení dosáhnout periodicky měnící se střední hodnoty výstupního napětí. [6] Antiparalelně zapojené usměrňovače pracují bez okruhových proudů, tedy nikdy nebudou oba můstky propouštět proud současně a proud se nebude uzavírat mimo zátěž. Cyklokonvertory se většinou vyskytují ve třífázovém zapojení v aplikacích, kde je pro napájení pohonů požadováno nízkého kmitočtu (např. kulové mlýny v cementárnách, lodní šrouby), jelikož výstupní frekvence může dosahovat maximálně pouze 40% vstupní frekvence.



Obr. 2-1 Blokové schéma zapojení jednofázového přímého měniče frekvence

2.2 Nepřímý měnič frekvence

Nepřímý měnič frekvence se skládá ze dvou polovodičových měničů a stejnosměrného meziobvodu. Na vstupu frekvenčního měniče se nachází usměrňovač a na výstupu střídač. Stejnosměrný meziobvod obsahuje filtrační člen, který slouží k oddělení impedancí obou měničů, a může být napěťový nebo proudový. [6]

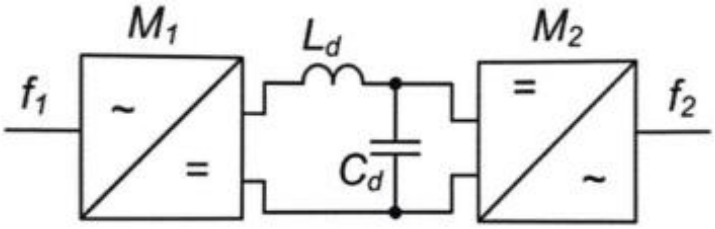
2.2.1 Nepřímý měnič frekvence s napěťovým meziobvodem

Tento typ se nejčastěji používá v kombinaci se synchronním motorem s permanentními magnety. Existuje velké množství zapojení usměrňovačů, které se pro tento typ měniče používají, ale nejpoužívanějším je nereverzační třífázový neřízený můstkový usměrňovač. Střídač musí být napěťového typu. Schéma měniče je znázorněno na Obr. 2-2. Stejnosměrný meziobvod se skládá z indukčnosti L_d, která svou impedancí odděluje výstupní napětí usměrňovače od napětí na kondenzátoru C. [6] Indukčnost vyhlazuje proudy a tím zlepšuje chod usměrňovače. Kondenzátor s vysokou kapacitou pak zajišťuje čistě stejnosměrné napětí na vstupu do střídače.

Časté je také doplnění stejnosměrného meziobvodu o jednoduchý stejnosměrný měnič

napětí s odporovou zátěží v paralelní větvi ke kondenzátoru. V případě zvýšení napětí ve stejnosměrném meziobvodu nad dovolenou hodnotu, například při odstavení motoru dojde k vybití energie do odporu.

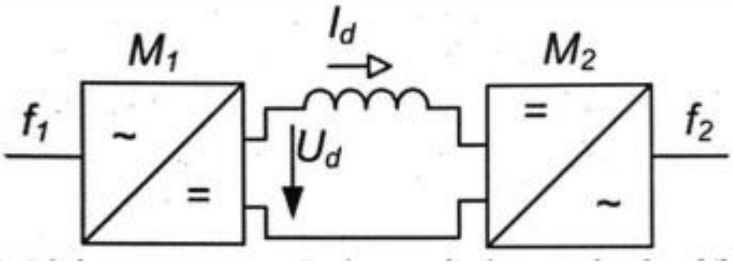
Problémem usměrňovače je, že ze sítě odebírá nesinusové proudy. Proto se užívá pulsně šířkové řízení napěťového střídače, kterým zároveň měníme efektivní hodnotu výstupního napětí. Střídač je řízen tak, že činný výkon protéká ze střídavé strany do strany stejnosměrné a odebíraný proud ze střídavé strany má téměř sinusový průběh. [6]



Obr. 2-2 Nepřímý měnič frekvence s napěťovým meziobvodem [6]

2.2.2 Nepřímý měnič frekvence s proudovým meziobvodem

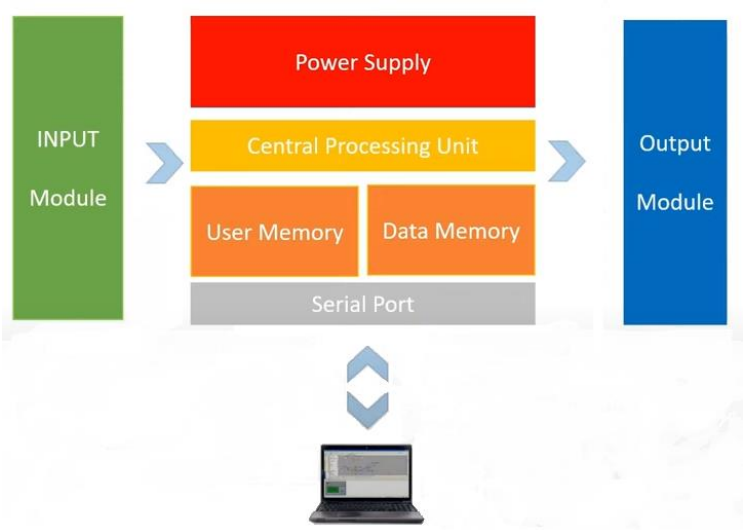
Na rozdíl od měniče s napěťovým meziobvodem, usměrňovač měniče s proudovým meziobvodem musí být řízený, aby byl zajištěn ve stejnosměrném meziobvodu požadovaný proud I_d. Meziobvod je tvořen indukčností L_d, která zajišťuje proudový charakter meziobvodu.



Obr. 2-3. Nepřímý měnič frekvence s proudovým meziobvodem [6]

KAPITOLA 3: PLC

Z angličtiny zkratka pro Programmable Logic Controller neboli programovatelný logický automat. Jedná se o průmyslový počítač, který prostřednictvím logického řízení vstupů a výstupů umožňuje automatizaci strojů a řízení procesů v průmyslu. Přijímá signály ze snímačů (vstupů) a uvádí do činnosti akční členy na základě naprogramované logiky v reálném čase. Řízení pomocí PLC odstavilo řízení za použití jednoúčelových integrovaných obvodů, které se již vyskytují pouze v jednodušších jednoúčelových aplikacích, na vedlejší kolej, jelikož jsou funkce integrovaného obvodu navždy dané a nenabízí potřebnou flexibilitu.



Obr. 3-1 Blokové schéma PLC

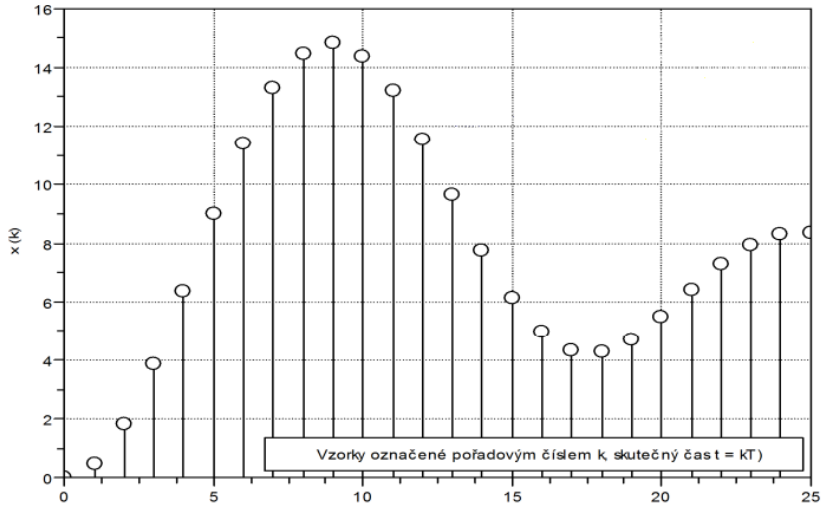
Na obrázku 3.1. vidíme blokové schéma PLC. V pravidelném cyklu v řádech milisekund dochází ke čtení vstupů, dále k jejich zpracování řídicí jednotkou a následnému napájení výstupů, jež mohou ovládat světelnou signalizaci, motor, ventily, nebo spínat ovládací relé. Vstupní a výstupní signály mohou být buď digitální nebo analogové. Digitální signál vychází z binární číselné soustavy a je reprezentován logickou nulou nebo jedničkou. Analogový signál je obvykle vytvářen analogovými snímači, kdy dochází ke spojité změně napětí v závislosti na změnách v provozním prostředí. Typicky se používají při snímání vzdálenosti, tlaku nebo teploty.

3.1 Zpracování analogového signálu

PLC neumí přímo zacházet s analogovými veličinami, a proto je třeba jim přiřadit přibližnou číselnou hodnotu. Aproximace analogové veličiny se skládá ze dvou kroků: vzorkování v čase a kvantování v amplitudě.

3.1.1 Vzorkování v čase

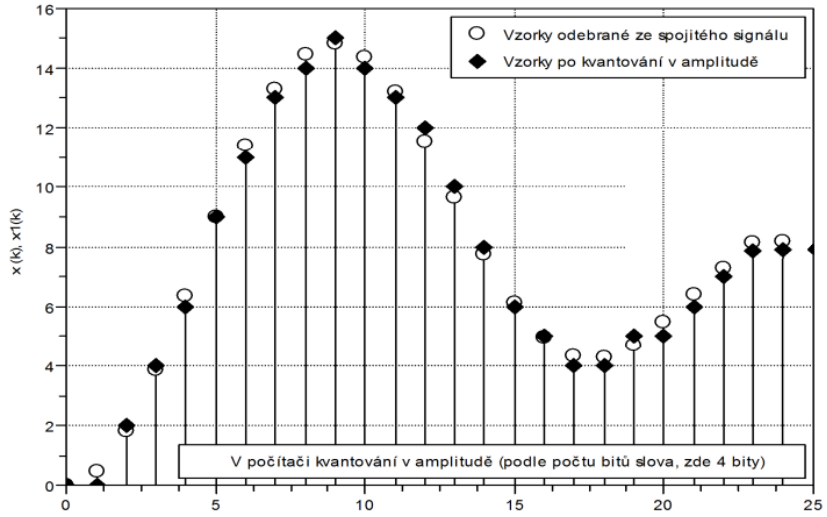
V prvním kroku je třeba analogovou veličinu ovzorkovat. To probíhá s konstantní periodou. Při vzorkování dochází ke ztrátě informace, jelikož neznáme průběh veličiny mezi jednotlivými vzorky. Z toho důvodu je třeba dodržet vzorkovací teorém, který nám říká, že „přesná rekonstrukce spojitého, frekvenčně omezeného signálu z jeho vzorků je možná tehdy, pokud byla vzorkovací frekvence vyšší než dvojnásobek nejvyšší harmonické složky vzorkovaného signálu.“ Při nedodržení vzorkovacího teorému může vzniknout aliasing. [7]



Obr. 3-2 Vzorkování v čase (upraveno) [7]

3.1.2 Kvantování v amplitudě

V druhém kroku je nutné odebraným vzorkům přiřadit hodnotu podle počtu dostupných bitů. Opět dochází ke ztrátě informace vlivem rozlišení analogově číslicového převodníku. [7]



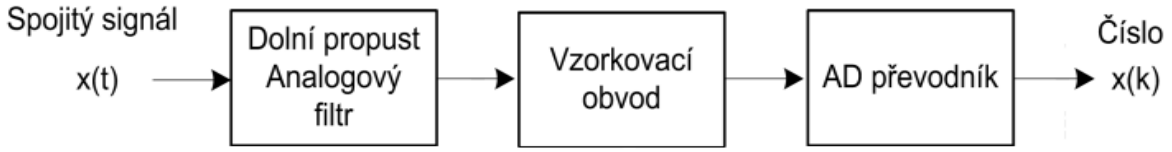
Obr. 3-3 Kvantování v amplitudě pro 4 bity (upraveno) [7]

3.1.3 Analogově-číslicový převod

Proces analogově číslicového převodu se skládá z několika bloků, které jsou zobrazeny na Obr. 3-4. Prvním blokem je antialiasingový filtr, který zajišťuje dodržení vzorkovacího teorému, omezuje tedy horní frekvenci signálu.

Druhým blokem je vzorkovací obvod, který je umístěn na vstup a umožňuje měření vyšších frekvencí, pro které by byla doba převodu příliš dlouhá. Vzorkovací obvod proto odebere z analogového signálu vzorek a drží jeho hodnotu během doby trvání analogově-číslicového převodu.

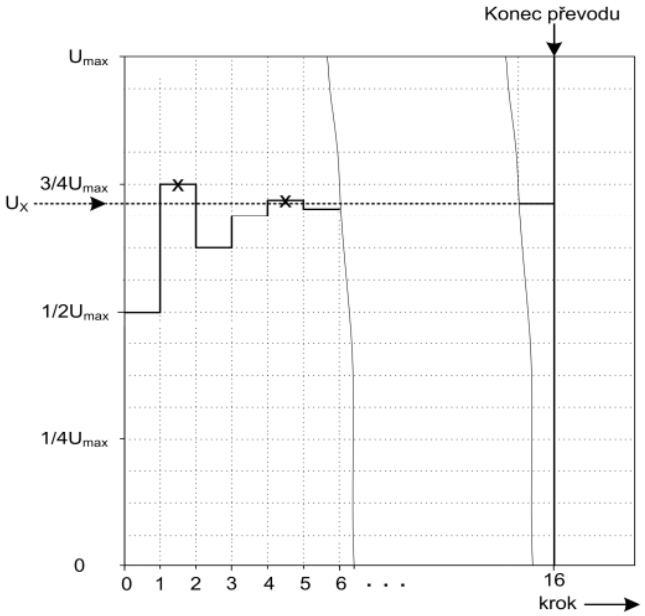
Posledním blokem je ADC převodník, přičemž existuje několik metod převodu, Nejpoužívanější metoda využívá postupné aproximace. [7]



Obr. 3-4 Typické schéma analogově číslicového převodu (upraveno) [7]

3.1.4 Metoda postupných aproximací

Pro použití metody postupných aproximací je třeba vybavit ADC převodník zpětným DAC převodníkem a komparátorem. Princip metody popisuje Obr. 3-5. V prvním kroku DAC převodník generuje poloviční napětí rozsahu převodníku a je v komparátoru porovnáno s ovzorkovaným napětím U_X. Pokud je generované napětí menší než U_X, v dalším kroku se napětí zvýší o polovinu zbývajícího rozsahu oproti minulému kroku a opět se porovná s U_X. V opačném případě se generované napětí o polovinu snižuje. Tento proces se opakuje do konce převodu. Doba převodu závisí na počtu převáděných bitů. [7]



Obr. 3-5 Princip metody postupných aproximací [7]

3.2 TIA Portal

Celým názvem Totally Integrated Automation Portal je software vyvinutý firmou Siemens. Jedná se o univerzální nástroj, který má sloužit k projektování, provozu a údržbě automatizačních systémů. Program TIA Portal si stáhneme do našeho počítače, ze kterého můžeme externě programovat PLC a následně program do PLC uploadovat. Rozšířené funkce dále nabízejí například možnost simulace programu na virtuálním PLC, čehož jsem hojně využíval pro odladění programu.

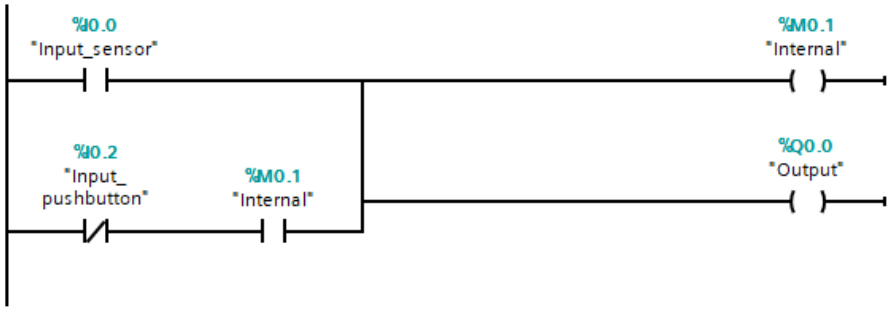
3.3 Programovací jazyky

TIA Portal podporuje několik programovacích jazyků pro konfiguraci PLC. Dle normy IEC 61131-3 existuje 5 PLC programovacích jazyků, které jsou podrobněji popsané níže.

3.3.1 Žebříčkové diagramy (LAD)

Jedná se o velmi běžně používaný programovací jazyk, který přehledně v řádcích zobrazuje jednotlivé logické operace. Schéma žebříčku se typicky skládá ze dvou svislých čar, které představují výkonové kolejnice (V programu TIA Portal není pravá kolejnice). Kolejnice jsou propojeny příčnými čarami, kam zapisujeme logické operace. Jednotlivé řádky jsou vzestupně

očíslovány, přičemž jeden řádek představuje jeden příkaz. V momentě spuštění PLC jsou výkonové kolejnice napájeny a procesor začne provádět instrukce řádek po řádku. Vstupu v žebříčkovém diagramu rozumíme NO, nebo NC kontakt. Výstup je pak reprezentován cívkou, která může představovat fyzický výstup PLC, nebo bit interní paměti pro použití jinde v programu.



Obr. 3-6 Ukázka práce s žebříčkovým diagramem

Nástroj TIA portal umožňuje kromě těchto základních operací i kombinovat programovací jazyky a do jednotlivých řádků vkládat funkční bloky.

3.3.2 Funkční bloky (FBD)

Funkční bloky můžeme rozdělit do několika skupin podle systémového využití. Existuje velké množství funkčních bloků. Můžou to být matematické funkce, časovače, porovnávací bloky, ale i bloky, které ovládají pohon nebo bloky, které si programátor sám vytvoří, aby zastupovaly požadovanou funkci. Základním funkčním blokem je hlavní organizační blok (OB1), do kterého je následně zapisován náš program a je spouštěn cyklicky.

Obr. 3-7 Ukázka práce s funkčními bloky

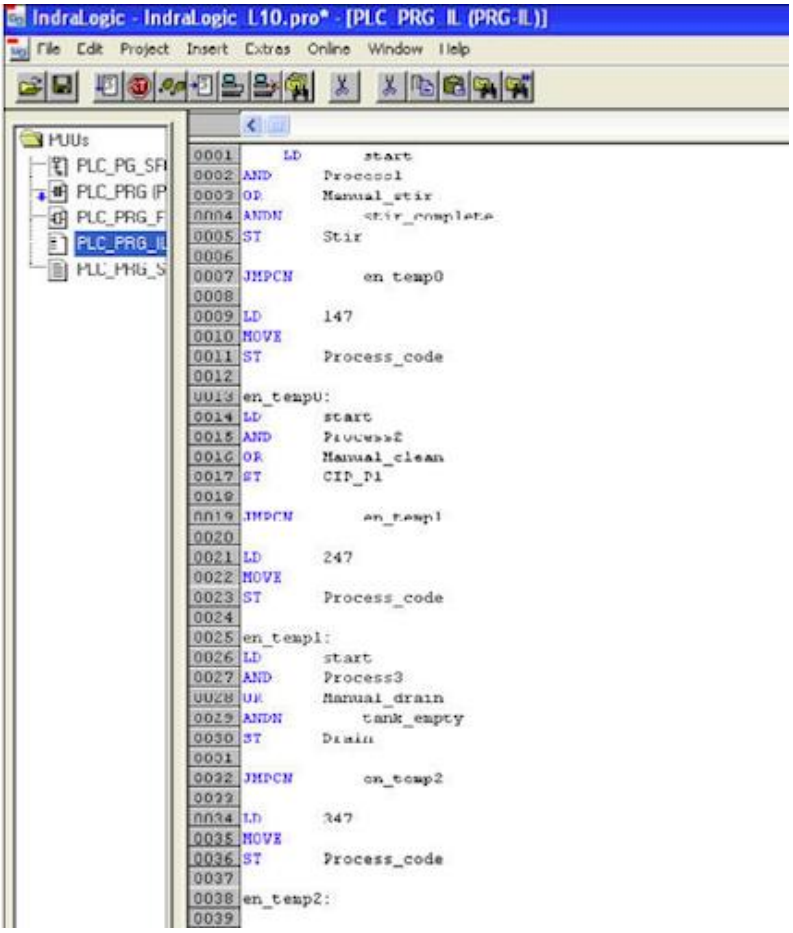
3.3.3 Strukturovaný text (SCL)

Strukturovaný text slouží k definování komplexních funkčních bloků a funkcí, které by byly velmi složitě realizovatelné v reléové logice. Tento programovací jazyk je vystavěn na základech jazyka PASCAL. Obsahuje širokou škálu prvků, kterou bychom od programovacího jazyka očekávali, jako například větvení (IF, THEN, ELSE, CASE OF) nebo cykly (FOR, WHILE, REPEAT). Jednotlivé prvky je možné vnořovat, viz ukázka níže:

```
WHILE #counter <= 10 DO
      IF sensor_1 THEN
            #counter += 1;
      ELSE
            #error := TRUE;
      END_IF;
END_WHILE;
```

3.3.4 Seznam instrukcí (IL)

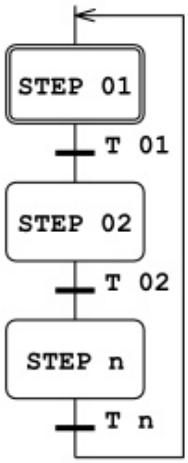
Jak už název napovídá, program se skládá z řady instrukcí a může připomínat programovací jazyk Assembler. Velkou výhodou tohoto jazyka je rychlost vykonávání programu v porovnání s grafickými jazyky. Seznam instrukcí je paměťově méně náročný, což je jistá výhoda u PLC s omezenou pamětí. Naopak nevýhodou je, že se nejedná o příliš rozšířený programovací jazyk hlavně z důvodu, že uživatelé upřednostňují přehlednější grafické jazyky. [8]



Obr. 3-8 Ukázka jazyku seznamu instrukcí [8]

3.3.5 Sekvenční funkční diagram (SFC)

I když SFC řadíme mezi programovací jazyk, spíše se jedná o pomocný nástroj pro přehledné strukturování programu. Sekvenční funkční diagram se skládá z kroků a přechodů, které přesně určují návaznost jednotlivých akcí. Kroky mají k sobě přiřazené soubory akcí a mohou být napsané v libovolném jazyce dle normy IEC. Například se může jednat o akci „Rozběh motoru“. Přechody se vážou k podmínkám, které musí být splněny, aby došlo k ukončení aktivního kroku a aktivaci kroku následujícího.



Obr. 3-9 Ukázka sekvenčního funkčního diagramu [12]

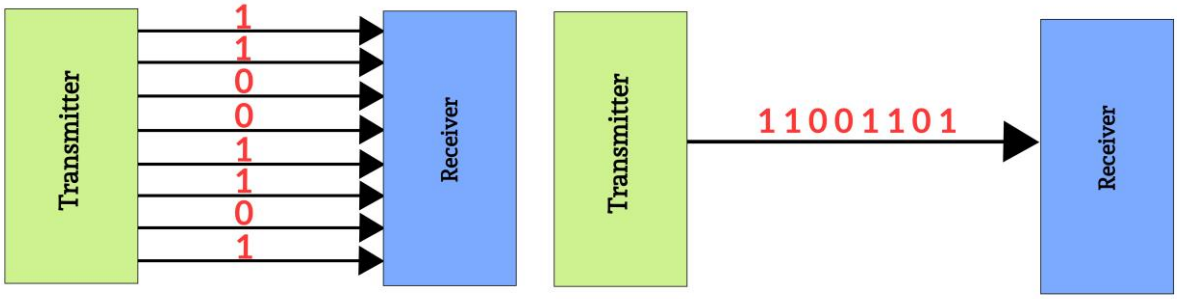
3.4 HMI

HMI představuje uživatelské rozhraní mezi zařízením a člověkem, které umožňuje člověku zařízení ovládat a zároveň poskytuje jistou vizualizaci systému. HMI komunikuje s PLC za účelem informovat uživatele o stavech prostřednictvím obrazovky. Zároveň poskytuje možnost ovládání zařízení a zadávaní hodnot.

Technologie urazila od počátku své existence dlouhou cestu. Dnešní HMI disponují grafickým zobrazením o vysokém rozlišení, množstvím ovládacích objektů, dotykovým displejem, knihovnou animací a objektů, možnostmi chybových hlášení, archivací hodnot a mnohými dalšími funkcemi. HMI, jeho funkce a provedení se podstatně liší podle účelu zařízení a místu aplikace. Převládá používání v průmyslových aplikacích, kde zastávají hlavně funkce sledování procesů, diagnostiky problémů a vizualizace dat.

KAPITOLA 4: KOMUNIKAČNÍ STANDARDY

Digitální přenos informace je možný provádět v principu dvěma způsoby. Jednou z možností je paralelní přenos, který umožňuje přenášet více bitů současně, tím pádem by měl být rychlejší než sériový přenos. Jeho nevýhodou je, že pro přenos je třeba větší množství vodičů mezi vysílačem a přijímačem a také není možné na větší vzdálenosti efektivně tlumit rušení včetně vzájemného, a je proto nutné omezit přenosovou rychlost. Po rozšíření sériových sběrnic jako USB se stal paralelní přenos záležitostí minulosti a v dnešních zařízeních se už v podstatě nevyskytuje. U starých tiskáren bychom se mohli setkat s rozhraním LPT nebo s rozhraním IEEE 488 u osciloskopů a jiných měřicích přístrojů. V této kapitole se dále budeme věnovat pouze sériové komunikaci.



Obr. 4-1 Paralelní a sériová komunikace

4.1 Sériová komunikace

Dnes nejpoužívanější typ komunikace umožňuje přenášet v jednom časovém okamžiku pouze jeden bit. Vysílač (Transmitter) a přijímač (Receiver) musí být synchronizované, aby nedocházelo ke ztrátě při přenosu. Existuje několik způsobů synchronizace vysílače a přijímače.

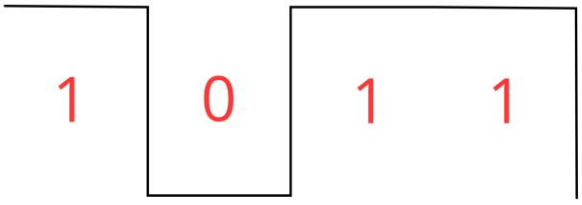
Přenos může být také asynchronní. V takovém případě dochází k synchronizaci prvním bitem tzv. „start bitem“. Synchronní přenos může být zajištěn samostatným vedením hodin, nebo mohou být hodiny přímo přenášeny současně s daty a na přijímači od dat odděleny. [9]

4.2 Kódování bitů

Existuje několik způsobů kódování sériově přenášených bitů. Jednotlivé metody mají různou odolnost vůči rušení, synchronizaci a složitost. [9]

4.2.1 NRZ (Non Return to Zero)

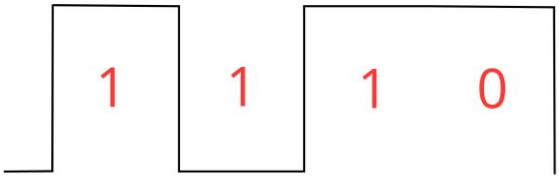
Tento typ kódování obsahuje pouze dvě hladiny, vyšší a nižší. Vyšší hladina reprezentuje 1 a nižší hladina reprezentuje 0. Neexistuje třetí hladina, jako tomu je např. u kódování s návratem k nule. Pro tento typ kódování je obtížná synchronizace na straně přijímače v případě, kdy je vysíláno větší množství bitů stejné hladiny, jelikož signál neobsahuje žádné hrany. NRZ kódování se používá ve vysílačích typu UART. [9]



Obr. 4-2 Ukázka NRZ kódování

4.2.2 NRZI (Non Return to Zero Inverted)

Vysílač používající kódování NRZI v případě přenášení 1 generuje hranu a při přenášení 0 nedochází ke změně hladiny signálu. Díky tomu dochází k omezení komplikací se synchronizací pouze na posloupnost nul, kdy se v signálu nevyskytují hrany.



Obr. 4-3 Ukázka NRZI kódování

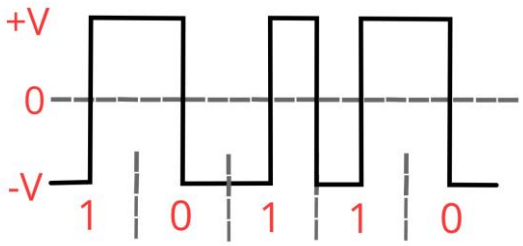
Používá se i kódování MLT3, což není nic jiného než tříhladinové kódování NRZI. Princip kódování je totožný s NRZI a využívá ho například 100 Mbps Ethernet. Důvodem používání tohoto kódování je snížit požadavek na propustné kmitočtové pásmo. Naopak nevýhodou je, že přijímač musí mít schopnost rozlišit 3 hladiny signálu. [9]

4.2.3 Bit stuffing/destuffing

Je metoda používaná u NRZ a NRZI kódování za účelem synchronizace přijímače. V případě, kdy se v kódu vyskytuje větší množství stejných za sebou jdoucích hodnot, se vkládá umělá opačná hodnota, kterou přijímač následně odebere. [9]

4.2.4 Fázové kódování

Fázové kódování se také nazývá Manchester. Pokud chceme vytvořit jedničku, musí být do poloviny bitového intervalu původního signálu vložena náběžná hrana. Naopak sestupná hrana kóduje nulu. Velkou výhodou tohoto kódování je dostatečné množství hran pro synchronizaci hodin. Fázové kódování vyžaduje ale oproti kódování NRZ dvojnásobné kmitočtové propustné pásmo, protože pokud budeme chtít vytvořit signál pro dvě jedničky/nuly za sebou, je potřeba během bitového intervalu vytvořit dvě náběžné/seběžné hrany. [9]



Obr. 4-4 Ukázka fázového kódování (Manchester)

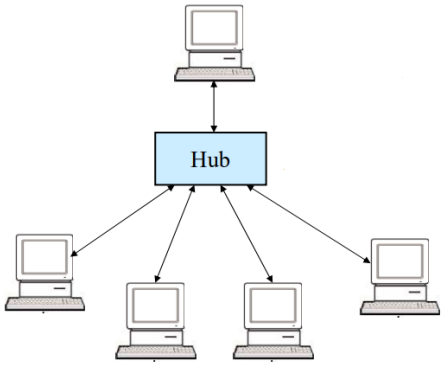
4.3 Topologie sítí

Topologie sítí udává zapojení dvou nebo více zařízení v rámci jedné sítě. Kromě fyzického zapojení zařízení v síti rozlišujeme, jestli je zapojení časově sdílené (Half-Duplex). V časově sdíleném zapojení je umožněno v jednom časovém okamžiku vysílat pouze jednomu připojenému zařízení a ostatní zařízení přijímají. Příkladem takového zapojení je společná sběrnice. Existuje i zapojení Full-Duplex, u kterého mohou zařízení vysílat a přijímat současně, neboť každé zařízení má svojí vlastní linku. [9]

4.3.1 Zapojení hvězda

Dnes nejpoužívanější topologie sítě je zapojení hvězda. V současnosti ho využívá například USB nebo Ethernet. V tomto zapojení jsou všechna zařízení v síti připojena k přepínači (Hubu). Výhodou tohoto zapojení je, že pokud dojde k přerušení komunikace na jednom spojení, nedojde

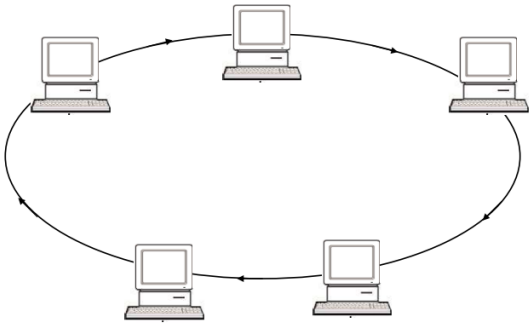
ke kolapsu celé sítě. Naopak nevýhodami jsou náročnost na kabeláž a přítomnost dalšího hardwaru, který v případě selhání odstaví celou síť.



Obr. 4-5 Zapojení hvězda (upraveno) [9]

4.3.2 Zapojení kruh

V kruhové topologii jsou všechna zařízení v síti zapojena v sérii, kdy každé zařízení má port pro sériový vstup a výstup. Výstup prvního zařízení je propojen se vstupem druhého a tak dále. Přenos v tomto zapojení je poměrně jednoduchý, jelikož je přenos pouze jednosměrný a nedochází ke kolizi. Oproti topologii hvězda je méně náročné na kabeláž, ale v případě přerušení komunikace na jednom uzlu přestávají komunikovat i další zařízení.



Obr. 4-6 Zapojení kruh [9]

4.4 Přístup na společnou sběrnici

Za účelem zamezení kolizí na společné sběrnici se začalo využívat modelu Master/Slave, kde Master je zařízení, které řídí přístup na společnou sběrnici. Postupně vyzývá zařízení typu Slave, aby začala vysílat. V některých sítích může být v záloze tzv. Weak-master, který v případě ztráty komunikace s jednotkou Master začne vykonávat její funkci.

Existuje i model Multi-master, ve kterém si jsou jednotky rovnocenné. Toto propojení je možné provozovat v režimu Half-Duplex nebo pomocí metody CSMA/CD a CSMA/CR. [9]

4.4.1 CSMA/CD (Carried Sense Multi Access/Collision Detection)

Metoda CSMA/CD je způsob řízení signálu na společné sběrnici. Všechny zařízení sledují, jestli na sběrnici někdo vysílá. Pokud na sběrnici není signál vysílán, libovolné zařízení může začít vysílat. Jestliže se jednotka rozhodne vysílat, paralelně detekuje kolize snímáním vysílání ostatních stanic. Pokud začalo ve stejnou chvíli vysílat více jednotek a došlo ke zjištění kolize, jednotka ukončí vysílání a je vygenerován pseudonáhodný časový interval, po jehož skončení se jednotka může pokusit opět vysílat. Existuje i metoda CSMA/CR (Collision Resolution), které využívá například protokol CAN. [9]

4.5 Protokoly používané pro sériovou komunikaci

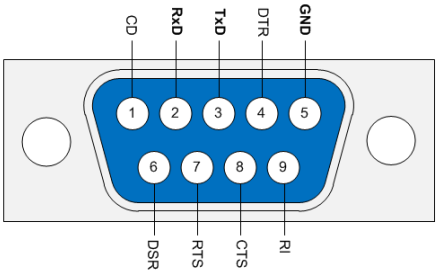
Protokolů pro sériovou komunikaci existuje je velké množství. Převážně se tyto protokoly liší v přenosové rychlosti, což je v dnešní době jeden z nejdůležitějších parametrů. Zatímco některé protokoly nedisponují vysokou přenosovou rychlostí, mohou ale vynikat například svou odolností vůči rušení nebo maximální přenosovou vzdáleností. Mohou mít rozdílnou metodu přístupu na sběrnici, způsob kódování, přenosové médium a liší se i v dalších parametrech. [9]

4.5.1 UART

Je hardwarové zařízení, které slouží k asynchronní sériové komunikaci. Lze u něj konfigurovat formát dat a přenosovou rychlost. Používá kódování typu NRZ a přenášené zprávy neboli znaky jsou orámované start bity a stop bity. UART má oddělené vysílání a příjímání a může být konfigurován jako Full-Duplex. Protože pracuje na napěťové hladině počítače (0-3,3 V), není tato komunikace uzpůsobená přenosům na delší vzdálenost a používá se v kombinaci s konvertory různé úrovně. [9]

4.5.2 RS232

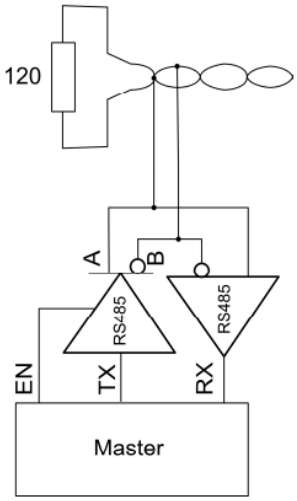
Jeden z historicky nejoblíbenějších komunikačních standardů je RS232, který se používal jako komunikační rozhraní osobních počítačů, kde se k připojení používá devíti pinový konektor D-Sub. Ten kromě datového vedení zahrnuje i vodiče pro řídicí signály. Nejčastější napěťové úrovně byly 12 V, což odpovídá logické 0 a logické 1 při –12 V. Přenosová rychlost standardu je 250kbps do vzdálenosti 20 metrů. [9]



Obr. 4-7 Konektor DE-9 pro standard RS232 [10]

4.5.3 RS422 a RS485

Z komunikačního standardu RS232 byly následně odvozeny RS422 a RS485. RS485 na rozdíl od svého předchůdce podporuje společnou sběrnici v časovém sdílení. Přenosová rychlost těchto konvertorů se zvýšila na 100 Mbps. Zároveň dosahují vyšší odolnosti proti rušení a umožňují přenosy na stovky metrů. Důvodem vyšší odolnosti vůči rušení je, že se jedná o dvouvodičové vedení. Vyhodnocení logického stavu závisí na napětí mezi oběma vodiči na rozdíl od RS232, kde se vyhodnocuje vůči zemi a je náchylnější vůči rušivým signálům. Dvoulinka je zakončena paralelně vloženou impedancí, která zabraňuje odrazům signálu. Díky těmto vlastnostem si našly široké uplatnění pro průmyslovou komunikaci. [9]



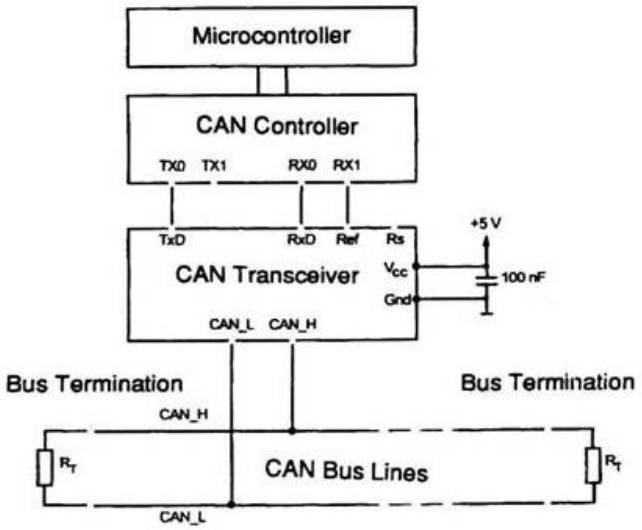
Obr. 4-8 Zapojení RS485 (upraveno) [9]

4.5.4 CAN (Controll Area Network)

Se sběrnicovým standardem CAN přišla firma Bosch a původně byl určen do automobilů a obtížných podmínek z hlediska rušení. Následně se rozšířil i do oblasti průmyslové automatizace. CAN řadíme do rodiny průmyslových protokolů FIELDBUS.

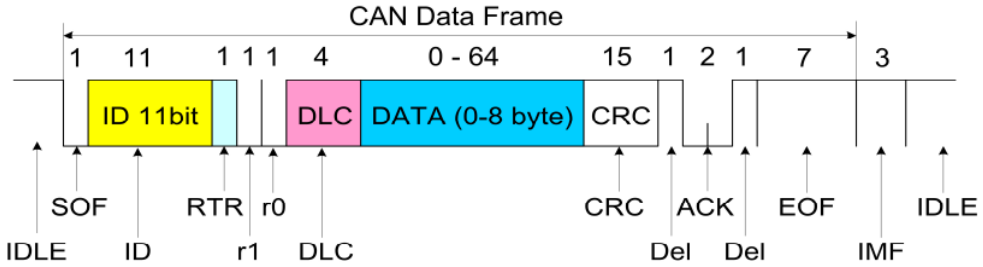
Protokol umožňuje komunikaci v reálném čase díky využití metody CSMA/CR a nedestruktivního arbitrážního řízení. Vysílaná data nemají adresu, ale obsah zprávy definuje unikátní identifikátor (ID), který zároveň určuje i prioritu zprávy. Jedná se o sběrnici typu Multi-Master. Všechna zařízení připojená ke sběrnici na ní sledují provoz. V momentě, kdy nedetekují zprávy, může jednotka vyslat start bit, za kterým následují bity ID. V případě, kdy začne vysílat více zařízení najednou a dojde ke kolizi, prioritu dostává ID s nejnižším binárním kódem, tedy nejdůležitější zpráva. Ta získá sběrnici a začíná vysílat. Každé zařízení připojené ke sběrnici CAN má na přijímací straně filtr, který propouští pouze zprávy s určitým ID. Pokud byla zpráva jednotkou přijata, přenos je potvrzen pomocí ACK bitů. Jestliže zařízení nezískalo sběrnici, opětovně se pokusí získat sběrnici, jakmile se sběrnice uvolní.

Protokol využívá kódování typu NRZ a bit stuffingu. Kromě identifikátoru zpráv má CAN i důslednou kontrolou chyb během přenosu zpráv, kdy rozlišuje náhodné rušení a trvalé chyby. Přenos zprostředkovává kroucená dvoulinka se zakončovacími odpory. Maximální přenosová rychlost je 1 Mbps do vzdálenosti 40 metrů. Pro větší vzdálenosti se rychlost přenosu snižuje. [11]



Obr. 4-9 Zjednodušené schéma zařízení s CAN rozhraním [11]

Obr. 4-10 popisuje standardní formát protokolu CAN s 11 bity pro ID, kde SOF značí start bit, následuje ID, DLC indikuje délku zprávy, CRC slouží pro kontrolu správného odesílání zpráv a detekci chyb, do bloku ACK se potvrzuje úspěšné přijetí zprávy, EOF značí konec zprávy. [9]

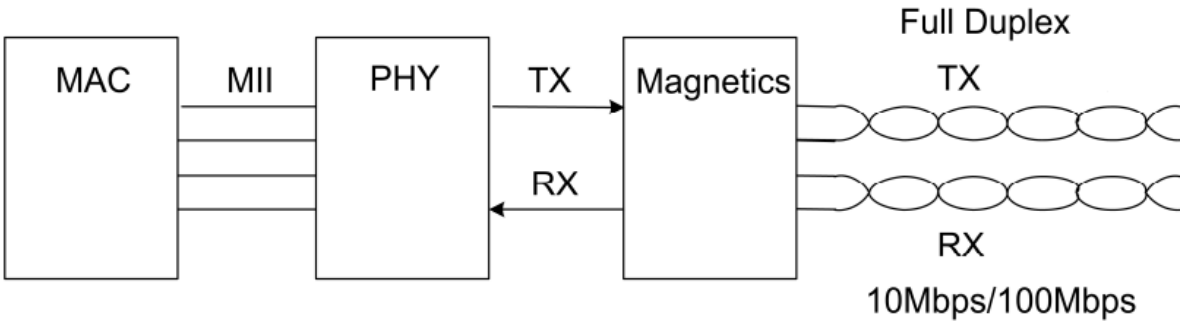


Obr. 4-10 Standardní formát protokolu CAN s 11 bity pro ID (upraveno) [9]

4.5.5 Ethernet

Ethernet je dnes nejrozšířenější komunikační protokol. Existuje několik verzí Ethernetu s různými vlastnostmi, ale nejpoužívanější je Ethernet s přenosovou rychlostí 100 a 1000 Mbps. Jako přenosové médium lze využít kroucené dvoulinky, optických vláken nebo ve starších verzích koaxiálního kabelu. Protokol realizuje dvě nejnižší vrstvy OSI modelu, který představuje příklad řešení sériové komunikace, složeného z jednotlivých vrstev. Ty jsou navzájem nezávislé a nahraditelné.

První vrstva definovaná ethernetem určuje parametry fyzického propojení. Na Obr. 4-11 je realizována blokem PHY (Physical Layer), který kóduje bity, řídí přístup na médium a dekóduje příchozí signál. Spojová vrstva je realizována blokem MAC (Medum Access Control). Blok MAC sestavuje jednotlivé rámce pro vysílání, rozděluje rámce přijaté a detekuje chyby. Typicky je blok MAC součástí mikrokontoleru a blok PHY je samostatná součástka. Bloky jsou propojené rozhraním MII (Medim Independent Interface). Výstup bloku PHY je vyveden přes transformátor při použití kroucené dvoulinky na standardní konektor RJ45.



Obr. 4-11 Zapojení ethernetového protokolu (upraveno) [9]

Nejrozšířenější ethernetový protokol je 100BASE-TX s přenosovou rychlostí 100 Mbps. Je typu Multi-Master, Full-Duplex a využívá topologie hvězda. Na rozdíl od protokolu CAN, kde bylo vysílání veřejné, zde protokol obsahuje adresu odesílatele i příjemce. Jako přenosového médium je využívána kroucená dvoulinka kategorie CAT-5, pro kterou je určena přenosová vzdálenost 100 metrů. Kódování je tříhladinové typu MLT3. Protokol za účelem synchronizace přijímače překódovává, kdy je každá čtveřice bitů doplněna o pátý znak tak, aby obsahovala alespoň dvě jedničky (tedy hrany u MLT3 kódování). Ethernet není vhodný na dlouhé vzdálenosti a pro aplikace reálného času. [9]

4.5.6 EtherCAT

EtherCAT je modifikovaný protokol Ethernetu do podoby Master-Slave v kruhové topologii. Byl vyvinut společností Beckhoff Automation pro řídicí účely a automatizaci procesů v reálném čase. Funkce práce v reálném čase je zajištěna následující metodikou. Master jednotka vysílá data, která procházejí všemi uzly. Každé Slave zařízení čte data, která mu jsou adresována „za běhu“, a vkládá do rámce svá data. Rámec je díky tomuto principu zpožděn pouze o dobu šíření. Poslední uzel v segmentu detekuje otevřený port a odesílá zprávu zpět do Master jednotky za využití Full-Duplex zapojení.

Firma také nabízí doplněk k výše popsanému protokolu EtherCAT P, který kromě přenosu komunikačních dat zajišťuje i napájení prostřednictvím standardního čtyřvodičového ethernetového kabelu. [13]

4.5.7 AS-i

Rozhraní AS-i využívá dvojice plochých vodičů a umožňuje snadnou instalaci díky piercing-technologii, během které jehlou propíchneme vodič a vytvoříme vodivé spojení. Systém má sloužit ke snadnému a ekonomickému připojení zařízení nejnižší úrovně automatizace, tedy snímačů a akčních členů a jejich připojení k vyšší úrovni řízení. Protokol AS-i zajišťuje snadné rozšíření systému, jelikož každé zařízení může být libovolně adresováno a může být připojeno ke sběrnicovému systému v kterémkoliv bodě do sítě s jednou Master jednotkou. Protokol podporuje libovolnou topologii sítě. Kabel slouží nejen k přenosu dat, ale zároveň i napájení. [14]

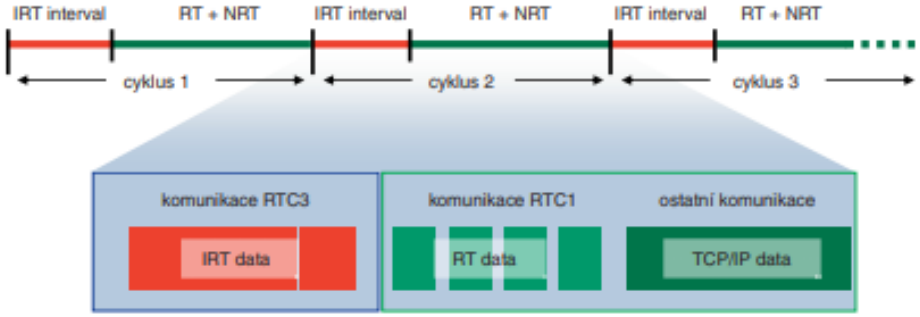
4.5.8 Profinet

Profinet je jeden z nejrozšířenějších komunikačních systémů pro automatizační techniku, které jsou založené na bázi průmyslového Ethernetu. Systém Profinet využívá Ethernet jako přenosovou vrstvu. Důležitým prvkem Profinetu jsou přepínače, které pracují na úrovni druhé vrstvy komunikačního modelu ISO. Ty umožňují bezkolizní komunikaci v režimu Multi-master.

Každý přepínač předává zprávy ze vstupního portu na výstupní, kde se nachází cílové zařízení. Přepínač ví, přes který port jsou připojená jednotlivá zařízení, jelikož zařízení mají přiřazené jednoznačné MAC adresy a při uvádění do provozu probíhá sekvence identifikace zařízení v síti.

Komplikace nastává při současném příchodu dvou zpráv určených pro stejný výstupní port. V takovém případě musí přepínač jednu zprávu pozdržet. K určení kterou, využívá Profinet dvou

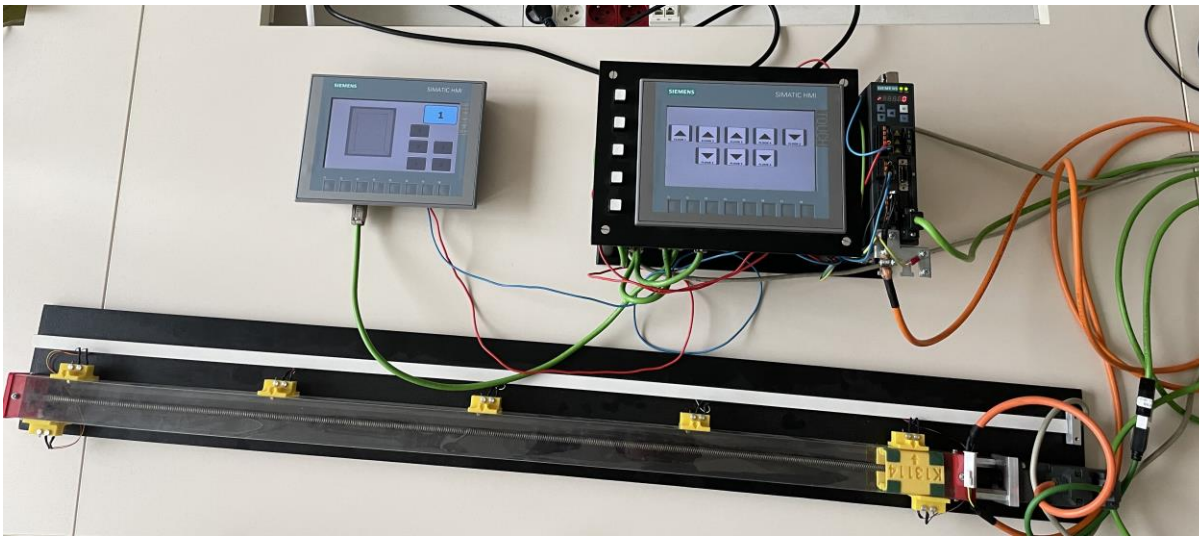
principů. Přednost mají v první řadě zprávy s vyšší prioritou. Při rovnosti priorit má přednost zpráva, která dorazila dříve. Vyšší prioritu mají například provozní data. Zprávy s vyšší prioritou se označují jako RTC1 a zprávy s nižší prioritou jako NRT. Pro úlohy, které vyžadují čistě deterministickou komunikaci nabízí Profinet typ komunikace RTC3. Pro tyto zprávy je v rámci komunikačního cyklu vyhrazeno samostatné pásmo, ve kterém není zařízením umožněno vysílat zprávy typu RTC1 a NRT. Veškerá komunikace typu RTC3 je dopředu naplánovaná. Strukturu komunikačního cyklu popisuje Obr. 4-12. Velkou výhodou systému Profinet je využití síťové infrastruktury několika protokolů současně se stejnou spojovací strukturou – Ethernet. [16]



Obr. 4-12 Struktura komunikačního cyklu [16]

KAPITOLA 5: POUŽITÁ SESTAVA

K realizaci tohoto projektu mám dispozici následující zařízení. PLC S1215 dc/dc/dc, které řízením logických vstupů a výstupů bude dávat signály měniči SINAMICS V90. Měnič ovládá PMSM servomotor SINAMICS S-1FL6. Komunikace probíhá po sběrnici Profinet a další zařízení připojená do sítě je dvojice HMI panelů KPT700 Basic, které slouží k vizualizaci. Na Obr. 5-1 můžeme vidět model výtahu, kde se v jednotlivých podlažích nacházejí spínací kontakty. Kromě kontaktů v podlažích je model vybaven dvojicí koncových stop kontaktů pro případ nouze, které okamžitě zastaví výtah. Pokud kabina projíždí podlažím, kontakt je sepnut a vysílá signál, který vstupuje do PLC.



Obr. 5-1 Fotografie použité sestavy (PLC se nachází pod HMI panelem)

5.1 PLC S1215 dc/dc/dc

Jednotlivé typy CPU z řady PLC 1200 se od sebe liší napájecím napětím, velikostí paměti, počtem vstupů a výstupů a dalšími parametry. Pro tento projekt využijeme PLC S1215 dc/dc/dc s rozšířením o kompaktní switch modul pro zapojení všech komponentů do sítě PROFINET. První člen dc v názvu značí způsob napájení, tedy 24 V stejnosměrně. Druhý a třetí označují provozní napětí digitálních vstupů a výstupů (opět 24 V DC). [15]

5.2 Servomotor Simotics S-1FL6

Pohon výtahu zajišťuje třífázový servomotor s permanentními magnety na rotoru. Pro potřeby projektu je zcela dostačující model s jmenovitým momentem 0,32 Nm a jmenovitým výkonem 100 W. Servomotor je vybaven inkrementálním enkodérem s rozlišením 2500 pulsů na otáčku, který poskytuje relativní informaci o změně polohy.

5.3 Sinamics V90

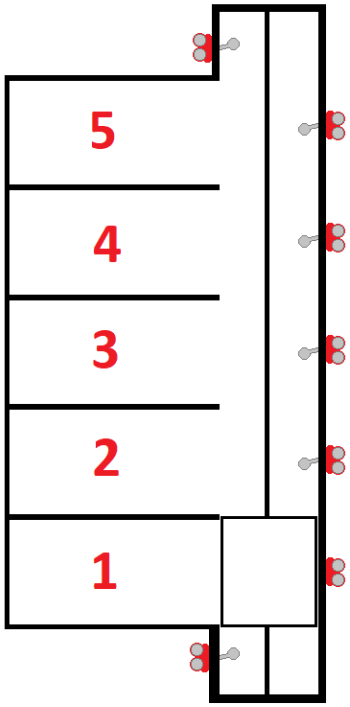
Jedním z prvků pro optimalizované řešení servopohonu je servoměnič Sinamics V90, který zajišťuje řízení pohybových funkcí pro servomotor S-1FL6.

5.4 HMI KTP700 Basic

KTP700 je označení pro HMI key touch panel se 7palcovým dotykovým displejem. Dále disponuje HMI panel i zabudovanými tlačítky. Jedná se o HMI řady Basic určených pro jednodušší aplikace.

KAPITOLA 6: VLASTNÍ PROJEKT

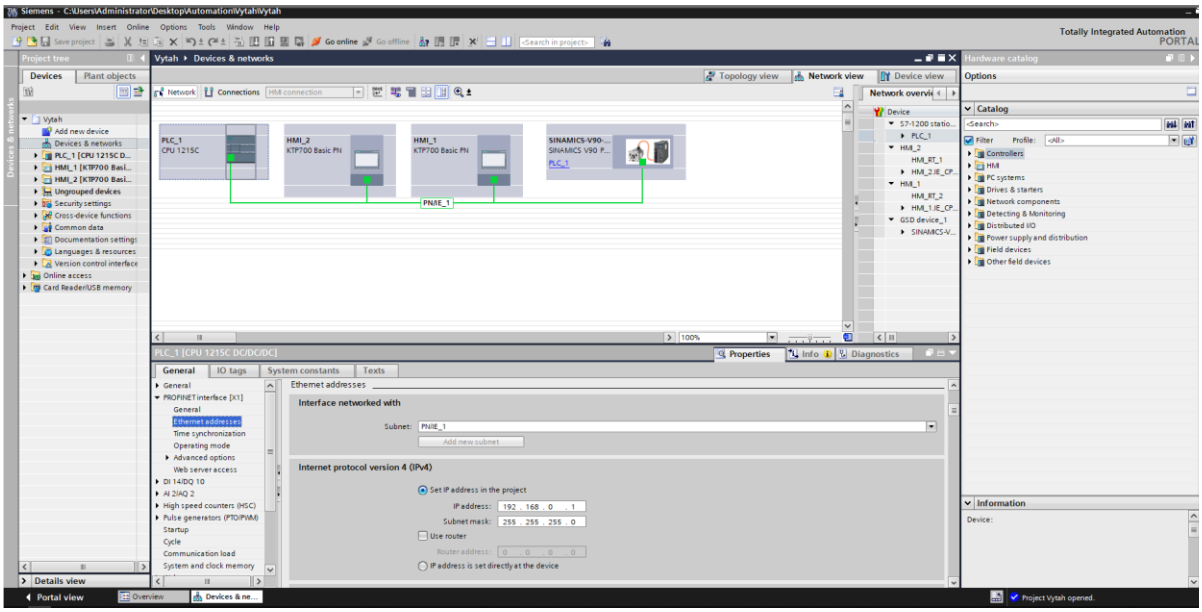
Součástí pokynů pro vypracování této bakalářské práce je v programu TIA Portal navrhnout logiku řízení výtahu pro pětipodlažní budovu. Logika řízení výtahu by měla být vybavena funkcí paměti zvolených pater a zastavování podle směru jízdy. Výsledný projekt řízení výtahu jsem měl implementovat na model výtahu s PLC Simatic S1200.



Obr. 6-1 Model výtahu

6.1 Přidání zařízení

Po založení projektu a vybrání správného PLC je třeba projekt doplnit o další prvky, kterých budu využívat. Servomotor S-1FL6 a měnič Sinamics V90 jsou zde zastoupeny jako jednotný prvek. Dále budu využívat dvojici HMI panelů. Všem zařízením je třeba nastavit odpovídající IP adresy. Následně na záložce Devices & Networks v okně Network view (viz Obr. 6-2) propojím všechna zařízení do jednotné sítě za využití komunikačního systému Profinet. U měniče je třeba vybrat komunikační telegram z hardwarového katalogu. Pomocí PROFIdrive telegramů jsou data cyklicky převáděna. Pro tento projekt jsem vybral standardní telegram 3 pro řízení otáček v uzavřené smyčce.



Obr. 6-2 TIA Portal – Network view

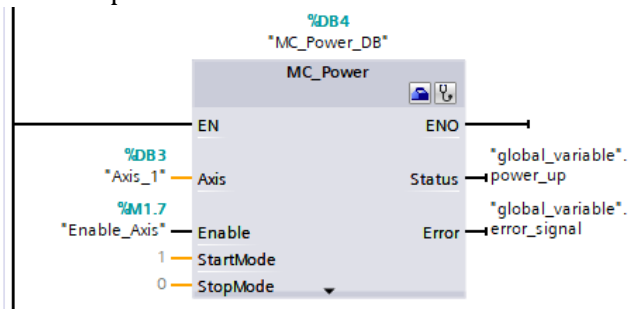
6.2 Přidání osy

V záložce Technology objects přidám novou technologickou osu. Osu je třeba parametrizovat. Nejprve zvolím používaný pohon a enkodér. Velice podstatné je pro tento projekt správně zvolit rychlost, zrychlení a zpomalení pohonu. Na jednu stranu chceme, aby se výtah plynule rozjížděl a zastavoval jako v reálné aplikaci, ale také zároveň musí být vždy dostatečně rychle zastaven v krajních patrech, kde se pár centimetrů od kontaktů signalizujících první a páté podlaží nachází koncové kontakty, které by odpojili celý pohon. Proto jsem zvolili maximální rychlost 250 mm/s, zrychlení 50 mm/s^2 a zpomalení 1500 mm/s^2.

Následně jsem nastavil funkci aktivního homingu přes digitální vstup kontaktu v prvním podlaží. Při zapnutí PLC dochází k homingu na tuto polohu a teprve následně je možné výtah ovládat.

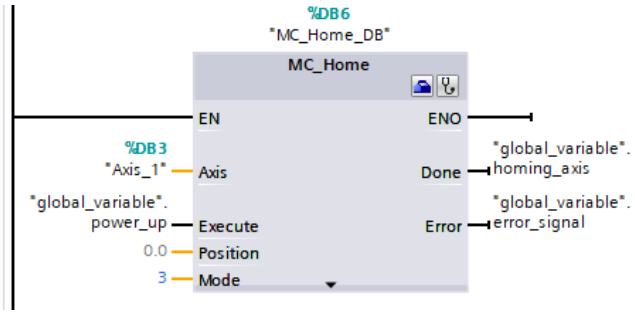
6.2.1 Bloky Motion Control

Funkční bloky Motion Control (MC) jsou k dispozici po přidání technologické osy. Jedná se o předdefinované bloky, které zastávají pohybové funkce. První blok, který jsem přidal je blok MC Power. Pokud je vstup bloku napájen, začne se napájet i pohon. Při napájení pohonu servomotor drží svou polohu.



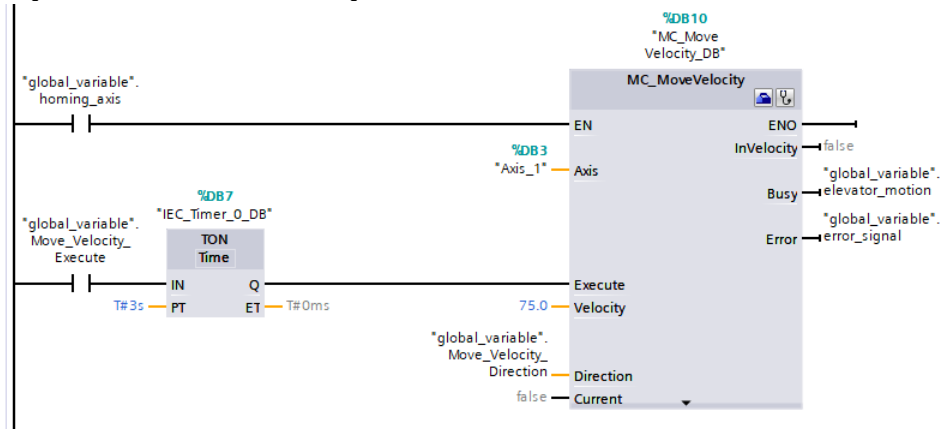
Obr. 6-3 Zapojení bloku MC Power

Napájený blok MC Power dává signál dalšímu bloku, který jsem přidal. Blok MC Home slouží k synchronizaci kabiny výtahu na výchozí pozici, což je v našem případě kontakt v prvním podlaží.



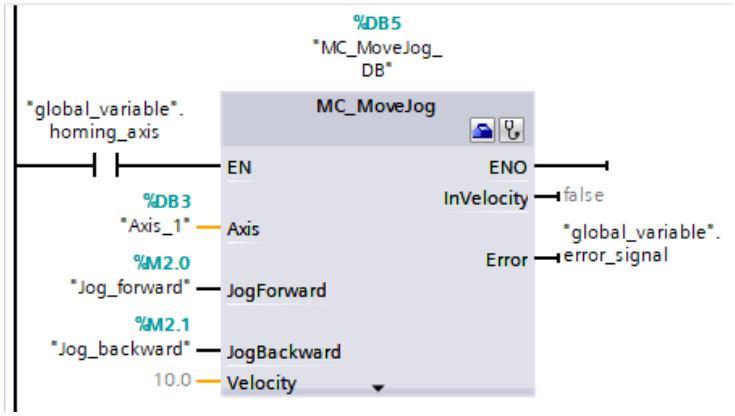
Obr. 6-4 Zapojení bloku MC Home

Po provedení synchronizace je výtah připraven k vykonávání pohybových funkcí. Existuje několik bloků, které toto umožňují. Pro tento projekt jsem vybral blok MC Move Velocity. Abych přivedl signál na vstup, využiju časovače, čímž dojde ke zpoždění signálu. Toho se využívá, pokud se v poli nachází více požadavků, aby výtah v cílových patrech zastavil po definovanou dobu. Blok MC Move Velocity jsem zvolil, jelikož dobře funguje v kombinaci s předem připravenou logickou částí programu a modelem výtahu. Blok MC Move Absolute, který by lépe umožňoval řízení z hlediska rozjezdů a zastavení nebyl optimální, jelikož kontakty v jednotlivých podlažích nejsou ekvidistanční a docházelo by k situacím, kdy výtah nedojede do požadované hodnoty a nedošlo by k sepnutí kontaktu v cílovém podlaží.



Obr. 6-5 Zapojení bloku MC Move Velocity

Pohyb výtahu je přerušen nadřazeným blokem jako například MC Halt. Program jsem ještě vybavil dalšími funkčními bloky MC jako je MC Reset a MC Move Jog. Tento blok zajišťuje pohyb osy v definovaném směru za podmínky, kdy je na vstup přiváděn signál z tlačítka z HMI panelu. Tuto možnost jsem přidal pro případ potřeby manuálně pohybovat kabinou výtahu při údržbě.



Obr. 6-6 Zapojení bloku MC Move Jog

6.3 PLC tagy

V dalším kroku jsem vytvořil PLC tagy, které představují vstupy, flagy a výstupy z PLC. Vstupy jsou v našem případě pouze vyvedené kontakty z jednotlivých podlaží a koncových kontaktů (písmeno I). Tlačítka z HMI panelu, které nejsou vyvedena na vstup PLC, ale do PLC se dostávají prostřednictvím Profinetu, označíme jako flagy (písmenem M). Výstupy označujeme jako Q. Tabulka použitých PLC tagů se nachází v příloze. Tagy Axis_1 si vytvořil program sám pro komunikaci pomocí telegramů.

6.4 Programovací bloky

V dalším kroku po vytvoření PLC tagů naprogramuji logiku výtahu. Vzhledem ke složitosti problému by bylo příliš komplikované psát celý program v žebříčkovém diagramu. Místo toho vytvořím funkční bloky, jejichž funkce popíšu SCL jazykem.

Hlavní program je zapsán do hlavního programovacího bloku Main OB1. Sem jsou vkládány napsané funkční bloky a následně i bloky řízení pohybu osy MC.

6.4.1 Datový blok global variable

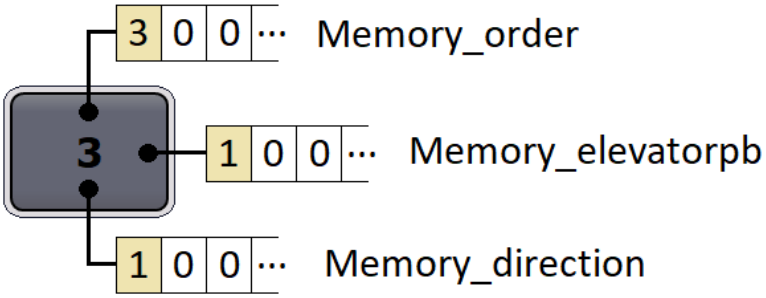
Založil jsem datový blok global variable, ve kterém si vytvořím potřebné globální proměnné. Celý seznam globálních proměnných se nachází v příloze, ale nejpodstatnější je proměnná Position_elevator, která udává aktuální polohu výtahu, trojice polí, kam jsou zapisována stisknutá tlačítka a proměnné number_of_orders a number_of_elevator_orders, jež informují o počtu požadavků ve frontě. Ke globálním proměnným přistupujeme zápisem:

```
"global.variable".Position_elevator
```

V rámci projektu jsou rozeznávány dva typy tlačítek. Tlačítka z výtahu a tlačítka z patra. Údaj o typu tlačítka zachycuje pole Memory_elevatorpb typu Bool. Pokud signál přišel z výtahu, přepíše se na příslušné pozici v poli logická 0 na 1.

Druhé pole Memory_orders je typu Int a ukládá údaj o zvoleném patře. Pokud tedy někdo v pátém podlaží zavolá výtah, zapíše se na příslušnou pozici pětka.

V každém podlaží (s výjimkou prvního a posledního) jsou dvě tlačítka, které člověk zmáčkne podle zvoleného směru jízdy, a tato informace se následně propíše do třetího pole Memory_direction typu Bool logickou nulou/jedničkou. Na obrázku níže můžeme vidět příklad, kdy se výtah nacházel v prvním podlaží a pasažér zmáčkl ve výtahu třetí podlaží.



Obr. 6-7 Ukázka zápisu při stisknutí tlačítka

6.4.2 Funkční blok Elevator

Blok Elevator zastává několik funkcí. Jednou z funkcí je přepisování aktuální polohy výtahu v proměnné Position elevator. Kdykoliv dojde k sepnutí kontaktu pohybující se kabinou výtahu, je přepsána poloha a proběhne funkce Order done, která je popsána v kapitole 6.4.6.

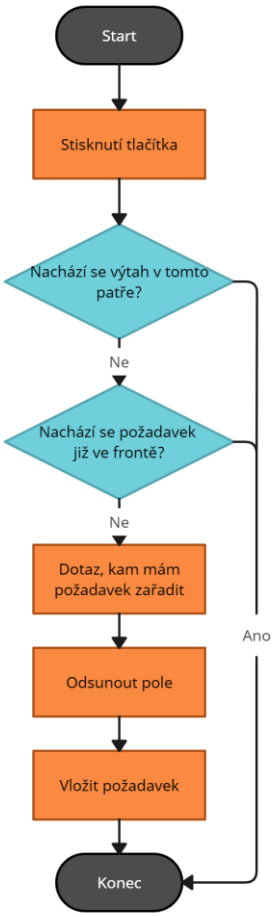
Při stisknutí tlačítka dochází k zadání nového příkazu k zápisu. Na kterou pozici v poli má být zápis proveden, definuje funkce určená této problematice. Funkční blok Elevator také zajišťuje, aby při stisknutí tlačítka nedošlo k jeho načtení do paměti několikrát. Podmínkou pro zápis je, že stisknuté tlačítko nesmí označovat patro, ve kterém se výtah zrovna nachází. Tento odstavec popsaný kódem pro jedno tlačítko vypadá následovně:

```
IF #p1_pb_up THEN
   IF #aretace_p1_up = FALSE THEN
         IF NOT("global_variable".Position_elevator = 1) THEN
               "global_variable".button_p1_up := TRUE;
         END_IF;
         "New_Order"(floor := 1,direction_up := TRUE, elevator_signal :=
         FALSE, elevator_motion := "global_variable".elevator_motion);
         #aretace_p1_up := TRUE;
   END_IF;
ELSE
   #aretace_p1_up := FALSE;
END_IF;
```

Zastává také ochrannou funkci, při sepnutí koncových stop kontaktů zajišťuje okamžité zastavení osy a vyšle chybové hlášení. V neposlední řadě dává při splnění podmínek pokyn k vykonání pohybu pro blok MC Move Velocity.

6.4.3 Funkce New Order

Funkce New Order se skládá z několika dílčích funkcí, které ve výsledku zapíší požadavek na příslušnou pozici v poli, pokud požadavek splňuje podmínky. V první řadě se výtah nesmí nacházet ve stejném patře jako stisknuté tlačítko. Funkce Find same value projde pole, jestli již pole tuto hodnotu neobsahuje. Pokud ne, prostor dostává nejsložitější funkce programu Where to shift, jejíž výstup je pozice, kam má být požadavek zařazen. Tato pozice následně vstupuje do funkce Shift array, kde je pole posunuto, aby se vytvořilo místo pro nový požadavek, který se vloží funkcí Set order. Funkci New Order popisuje vývojový diagram přiložený níže.



Obr. 6-8 Vývojový diagram funkce New Order

6.4.4 Funkce Where to shift

Účelem funkce Where to shift je určit a v podobě výstupu předat informaci o tom, na jakou pozici má být nový požadavek zařazen. Na vstupu do funkce jsou rozlišovány požadavky z jednotlivých podlaží a požadavky z výtahu. Asi nejlépe funkci vystihuje vývojový diagram přiložený níže. Nejprve funkce zjišťuje, jestli se ve frontě již nachází nějaké požadavky. Pokud se ve frontě žádné požadavky nevyskytují, na výstup funkce je zapsána nula a funkce končí. V případě, kdy se ve frontě již vyskytují požadavky, které čekají na vyřízení, je třeba určit, na jakou pozici má být nový požadavek zapsán.

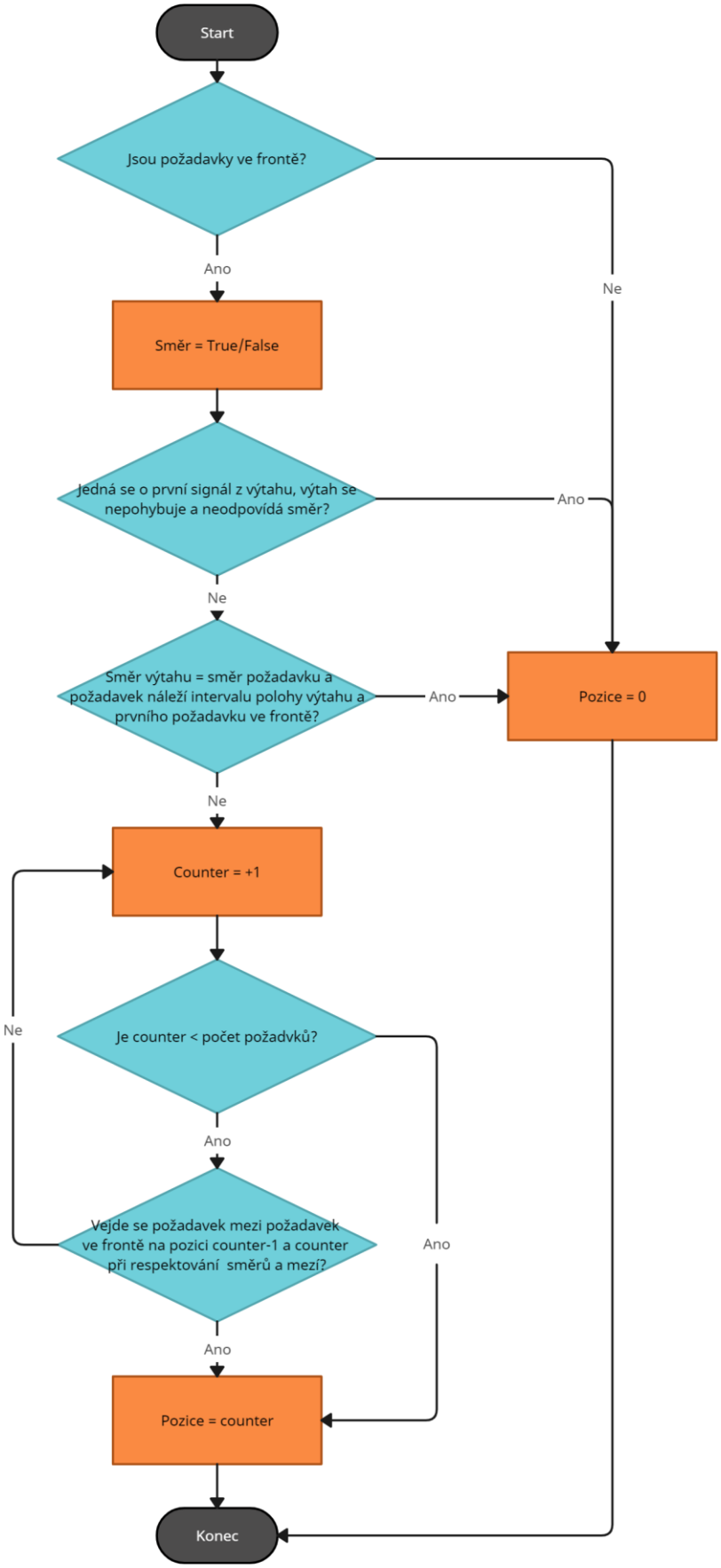
V druhém kroku funkce porovnává požadavek na prvním místě ve frontě s aktuální polohou výtahu. Tím je určen směr výtahu potřebný pro zařazení nových požadavků.

Požadavky z výtahu mají prioritu oproti požadavkům z jednotlivých pater v tom smyslu, že pokud člověk nastoupí do prázdného výtahu (ve frontě není požadavek z výtahu) a stiskne tlačítko ve výtahu, výtah se vždy rozjede směrem k patru, které bylo stisknuté ve výtahu.

Tento případ názorněji popisuje následující příklad. Osoba nastupuje do výtahu ve třetím podlaží, ale než stihne zmáčknout tlačítko s cílovým pátým podlažím, někdo jiný stiskne tlačítko v prvním podlaží a na nultou pozici ve frontě se zařadí první podlaží. Jakmile ale stiskne tlačítko osoba ve výtahu, na nultou pozici ve frontě se předřadí požadavek z výtahu a první podlaží je odsunuté na první pozici. Výtah se po zavření dveří rozjede směrem nahoru.

Pokud nový požadavek nesplňuje podmínky pro zařazení na pozici nula, counter zvýší hodnotu o jedna a začne probíhat WHILE cyklus. V první fázi cyklu je porovnáván counter s počtem požadavků (number of orders). Jakmile počet požadavků není větší číslo než counter, znamená to, že jsme již na konci fronty a tento požadavek zapisujeme na první volnou pozici a funkce končí. Při splnění této podmínky dochází k porovnávání požadavku na pozici counter – 1 a pozici counter s právě stisknutým požadavkem, který čeká na zapsání. Zároveň musí být respektován směr výtahu a nového požadavku.

Představme si situaci, kdy se výtah nachází v prvním patře a ve frontě jsou druhé a čtvrté podlaží se směrem nahoru. Směr výtahu je tedy nahoru. Stiskneme-li tlačítko v podlaží se směrem odpovídajícím směru jízdy výtahu, které náleží intervalu dva (požadavek na pozici counter - 1) a čtyři (požadavek na pozici counter), nový požadavek je zapsán mezi tyto hodnoty. Pokud by směr výtahu neodpovídal směru stisknutému tlačítku v podlaží, podmínky nejsou splněny, dojde k inkrementaci proměnné counter a požadavek bude v tomto případě zapsán za požadavky s vyhovujícím směrem, jelikož se counter rovná počtu požadavků. Analogicky funkce pracuje i v opačném směru jízdy. Tlačítka stisknutá ve výtahu vždy vyhovují směru jízdy. Specifický případ, který je třeba ošetřit, je jízda do krajních podlaží, jelikož v těchto patrech je požadovaný směr jízdy vždy opačný oproti směru jízdy výtahu, aby se do krajního podlaží dostal.



Obr. 6-9 Vývojový diagram funkce Where to shift

6.4.5 Funkce Shift Array

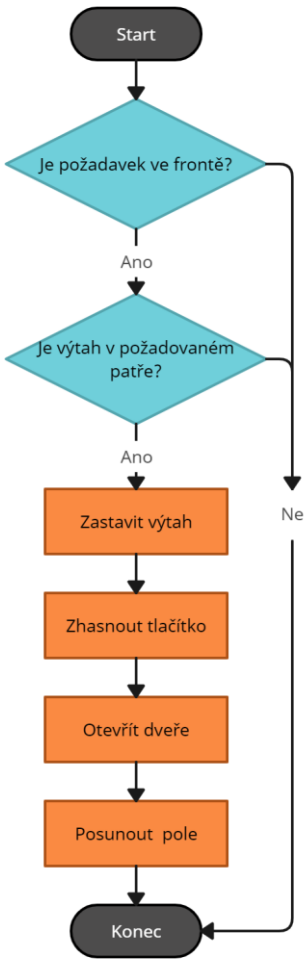
Tato funkce má za úkol odsunout pole od určité pozice, která je jí předána jako parametr. Funkci voláme ve dvou případech. Zaprvé pokud chceme pole odsunout směrem doprava, abychom mohli vložit novou hodnotu tak, aby nedošlo k přepsání hodnoty již obsažené v poli.

Druhý případ volání této funkce nastává v případě, kdy výtah dojel do požadované podlaží a je třeba pole posunout doleva. Požadavek na první pozici se posune na nultou, druhý na první atd.

Vstupními parametry této funkce je pozice, od které má dojít k posunu, a informace, jestli jde o posun směrem doprava, či doleva. Funkce posouvá zároveň všechny tři pole Memory order, Memory elevatorpb a Memory direction, ve kterých jsou uloženy informace o stisknutých tlačítkách.

6.4.6 Funkce Order done

Vždy, když dojde ke změně polohy výtahu ve smyslu sepnutí kontaktu v jednotlivých podlažích, je volána tato funkce. Kontroluje, jestli se výtah změnou polohy nedostal do cílového patra. Pokud ano, dochází k zastavení výtahu, zhasnutí tlačítka v příslušném patře nebo ve výtahu, otevření výtahových dveří a posunutí všech tří polí směrem doleva.



Obr. 6-10. Vývojový diagram funkce Order done

6.4.7 Funkce Reset

Funkce Reset je napojena na tlačítko RESET. Po jeho zmáčknutí dochází k vymazání pole požadavků a nastavení pole na nuly. S tím souvisí i globální proměnné informující o počtu

požadavků v poli, které jsou také nastavené na nulu. Vynulování se týká i chybových signálů a podsvícení všech tlačítek.

6.5 HMI panely

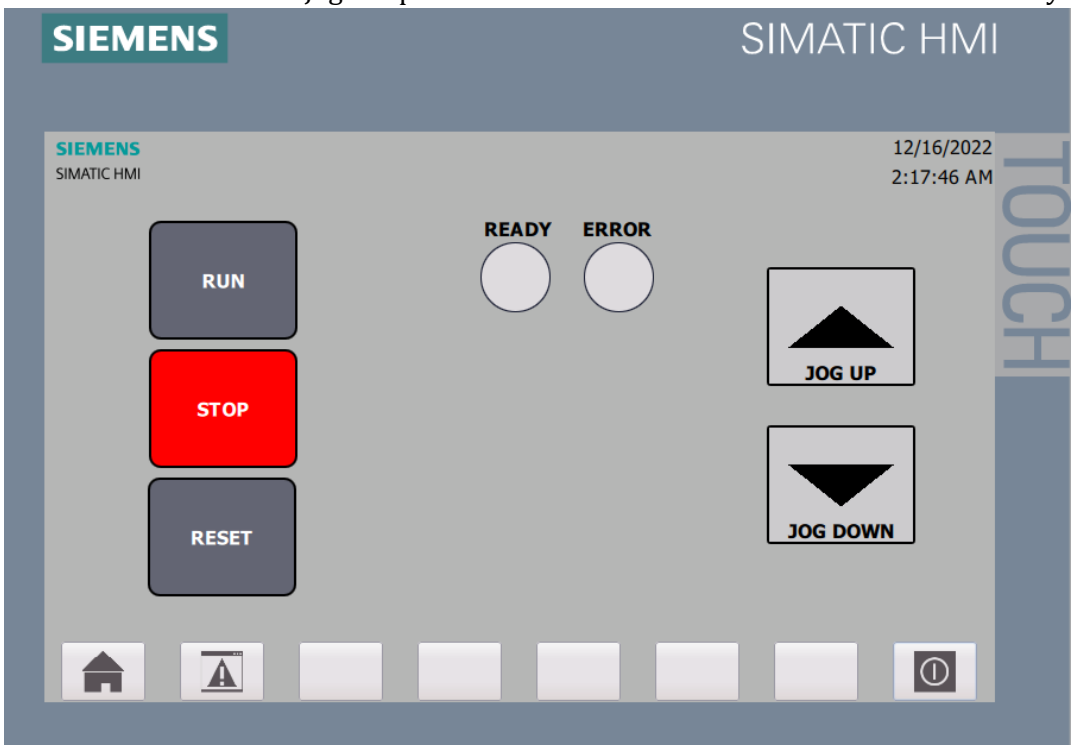
K realizaci našeho projektu jsem využil dvojici HMI panelů. Každému panelu byla vytvořena příslušná obrazovka. Jeden HMI panel reprezentuje kabinu výtahu společně s ovládací obrazovkou a druhý panel je vybaven tlačítky v jednotlivých podlažích.

6.5.1 Ovládací obrazovka

Součástí prvního HMI panelu je úvodní ovládací obrazovka, kterou můžeme vidět na Obr. 6-11. Ta se zobrazí po oživení sestavy. Stisknutím tlačítka RUN dojde k odblokování funkčního bloku MC Power a pohon začne napájet. Následně dojde k synchronizaci osy na kontakt v prvním podlaží. Po dokončení synchronizace se rozsvítí signálka READY a pohon je tedy připraven. Všechny MC bloky mají vyvedený chybový signál na společnou globální proměnnou. Při chybové signalizaci se rozsvítí signálka ERROR. Další důvod rozsvícení chybové signálky může být ten, že kabina výtahu najela na koncový kontakt.

Tlačítko STOP ihned odstaví pohon odpojením funkčního bloku MC Power. Posledním tlačítkem v levém sloupci je RESET. Po stisknutí tlačítka bude proveden funkční blok MC Reset a zároveň funkce Reset, která byla popsána v kapitole 6.4.7.

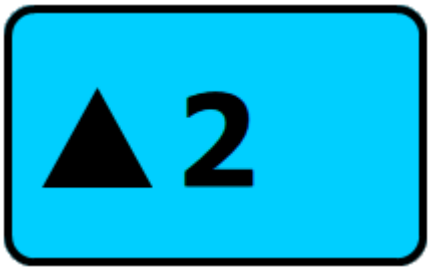
V pravé části obrazovky se nachází tlačítka JOG UP a JOG DOWN. Tato tlačítka jsou napojena na funkční blok MC MoveJog. Při podržení tlačítka se servomotor začne otáčet zvoleným směrem.



Obr. 6-11. Ovládací obrazovka panelu HMI 1

6.5.2 Obrazovka výtahu

Mezi ovládací obrazovkou a obrazovkou výtahu přepínáme tlačítky F1 a F3. V pravém horním rohu obrazovky výtahu se nachází modrý displej, který ukazuje aktuální polohu výtahu. Vedle polohy výtahu jsem vložil šipku ukazující směr jízdy výtahu.

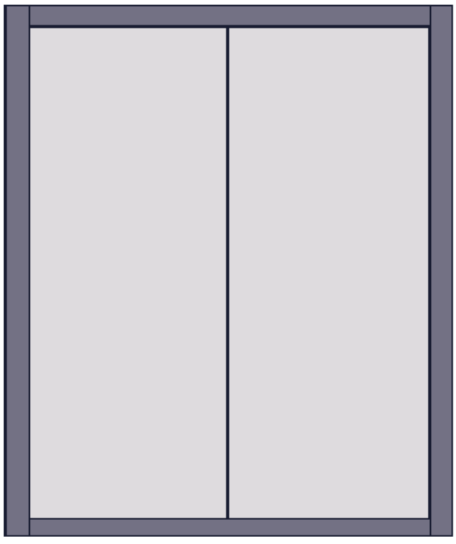


Obr. 6-12 Displej aktuální polohy a směru výtahu

Dále jsem pravou část obrazovky vybavil pěti tlačítky jednotlivých pater. Po stisknutí tlačítka se okraj tlačítka a číslice rozsvítí červeně a zhasne po dokončení požadavku.

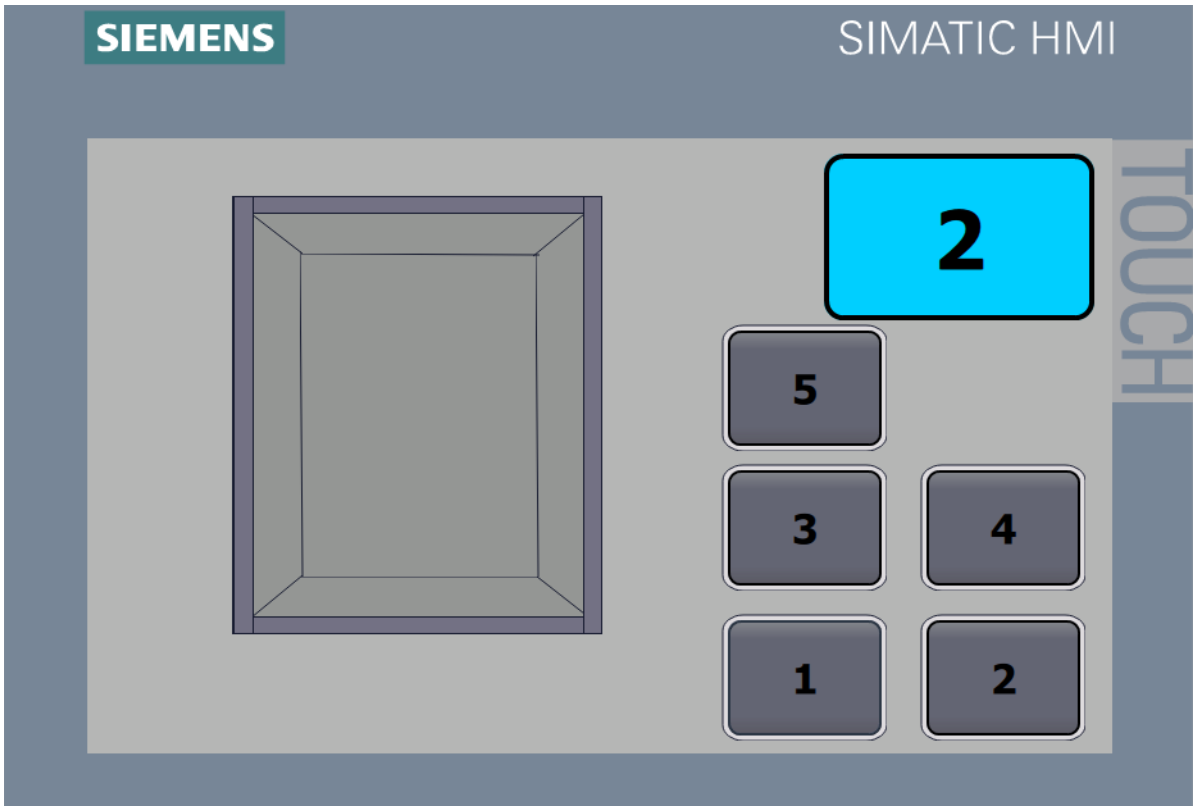
Obr. 6-13 Ukázka nestisknutého a stisknutého tlačítka ve výtahu

V levé části obrazovky je vizualizace kabiny výtahu. Pokud se má výtah rozjet, nejprve se dveře výtahu zavřou a následně po dvou sekundové prodlevě se výtah rozjede. Jakmile se výtah nachází v cílovém podlaží, výtah se zastaví a dveře se otevřou.



Obr. 6-14 Zavřené dveře výtahu

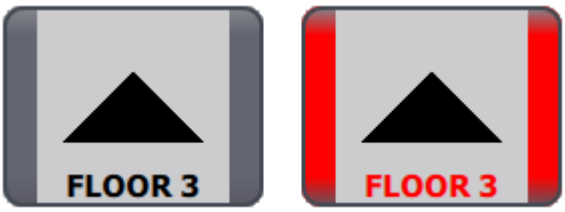
Všechny objekty dohromady pro případ, kdy výtah nemá žádné požadavky ve frontě, můžeme vidět níže.



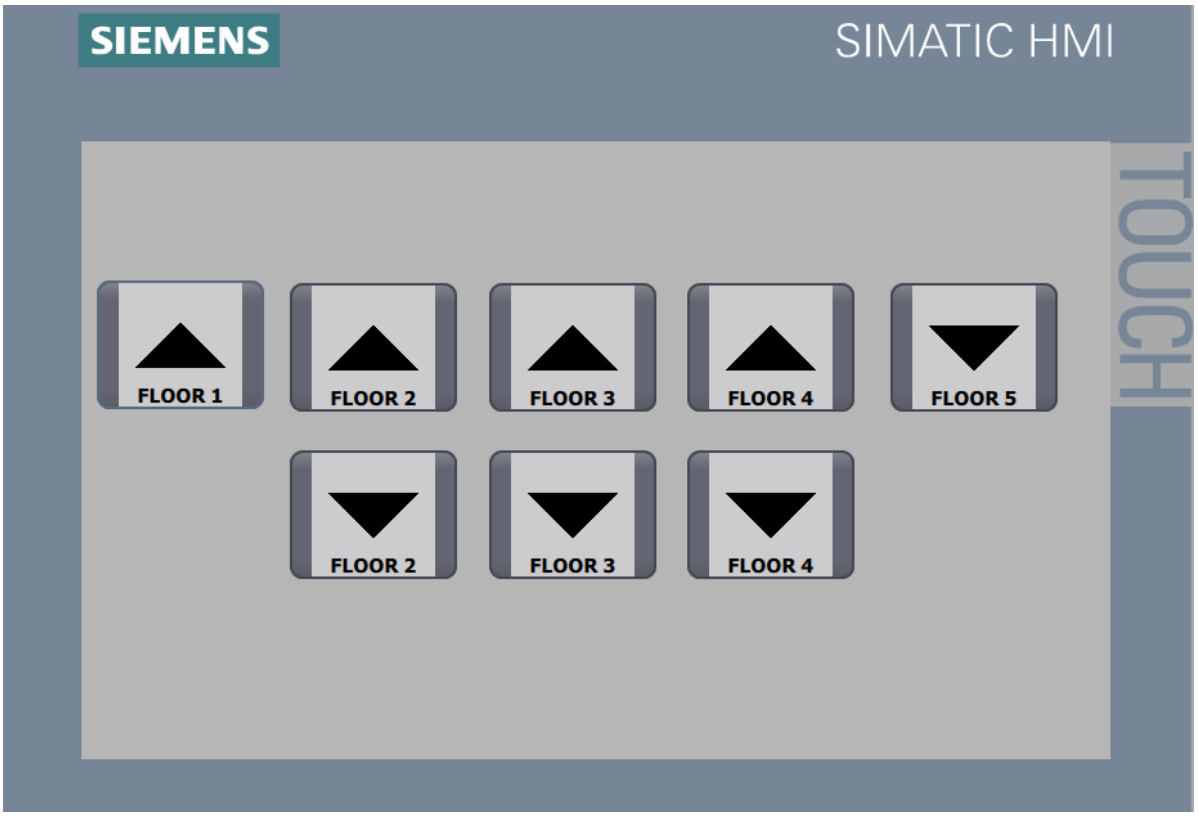
Obr. 6-15 Obrazovka výtahu panelu HMI 1

6.5.3 Obrazovka pro podlaží

Na panelu HMI 2 se po napájení PLC zobrazí obrazovka reprezentující jednotlivá podlaží. Skládá se z osmice tlačítek, které je možné zmáčknout v různých podlažích. V prvním podlaží se nachází tlačítko pouze nahoru a v posledním pátém podlaží se nachází tlačítko pouze dolů. V ostatních podlažích je může pasažér stisknout podle zvoleného směru jízdy. Po stisknutí tlačítka se tlačítko rozsvítí, čímž indikuje zápis požadavku do paměti obdobně, jako tomu bylo u tlačítek ve výtahu.



Obr. 6-16 Ukázka nestisknutého a stisknutého tlačítka v patře



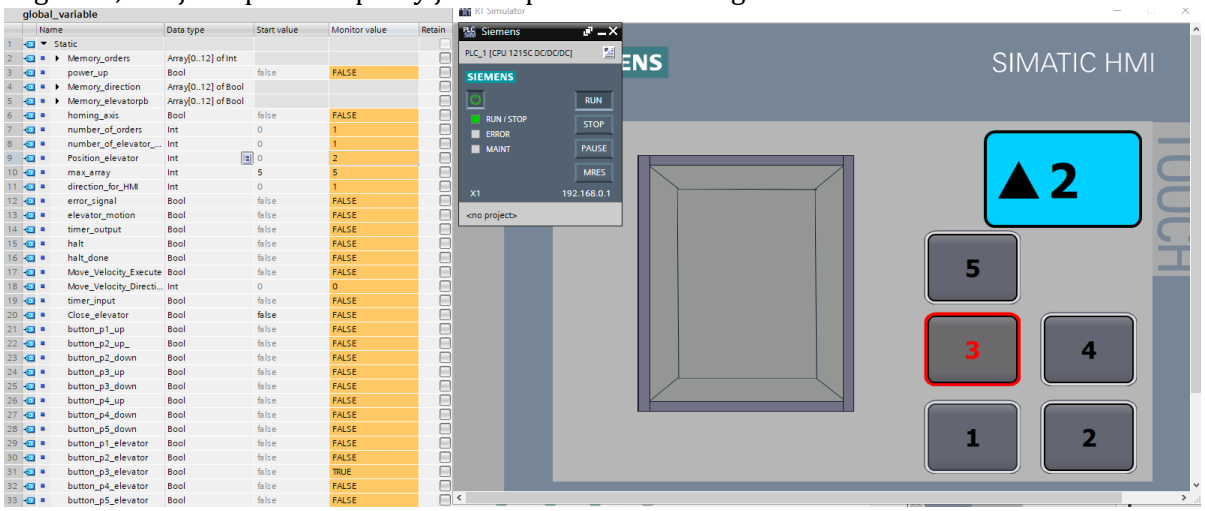
Obr. 6-17 Obrazovka pro podlaží na panelu HMI 2

KAPITOLA 7: SIMULACE

Velice užitečnou funkcí, kterou program TIA Portál nabízí, byla při vzniku tohoto projektu simulace. Hojně jsem ji využíval pro odladění správné funkčnosti logické části programu bez přítomnosti skutečného hardwaru. Simulace umožňuje nahradit PLC, ale i jednotlivé HMI panely. Simulace jednotlivých prvků spolu dokážou a komunikovat, čímž jsem dokázal replikovat reálnou situaci.

Nejprve je třeba spustit simulaci našeho PLC, následně program do virtuálního PLC nahrát a simulaci spustit. Během simulace můžeme pozorovat průběh hodnot při běhu programu. Bohužel použitá licence nepodporuje používání breakpointů, které by nám umožnily program v jednotlivých bodech zastavit a tím ulehčit odladění programu.

Vzhledem k hardwarové náročnosti simulace jsem nejprve vytvořil a ozkoušel logickou část programu, než jsem přidával prvky jako například technologickou osu.



Obr. 7-1. Ukázka simulace PLC a HMI panelu

ZÁVĚR

V teoretické části práce jsem se nejprve snažil stručně popsat a vymezit typ motoru zvolený pro tento projekt. Servomotory jsem rozdělil na jednotlivé skupiny a popsal jejich výhody a nevýhody, přičemž jsem se zaměřil na PMSM a jejich řízení. Nezbytnou součástí potřebnou pro řízení PMSM jsou frekvenční měniče, které jsem popsal v druhé kapitole. V následujících kapitolách jsem popsal PLC a udělal krátký úvod do PLC programování a softwaru firmy Siemens TIA portál. Všechny komponenty sestavy spolu musí komunikovat, a proto jsem se rozhodl do práce zahrnout kapitolu o komunikačních standardech.

Poté jsem se již věnoval samotnému vytváření programu v softwaru TIA Portál. Jednalo se o mojí první interakci s programováním logických automatů, a proto jsem se musel nejprve zorientovat v prostředí. V první řadě jsem vytvořil logickou část programu. Při programování bylo mojí hlavní myšlenkou, aby se model výtahu co nejvíce podobal reálné aplikaci. Bohužel z důvodu zjednodušení a nepřesností v použitém modelu nebylo možné všem reálným požadavkům vyhovět.

Například nebylo možné plynule zrychlovat a zpomalovat do požadovaných podlaží, jelikož kontakty v jednotlivých podlažích nejsou stejně daleko od sebe a mohlo by se stát, že by výtah nedojel přesně na určené místo. Dále by bylo vhodné koncové kontakty umístit o trochu dále od kontaktů v prvním a pátém podlaží, aby výtah nemusel tak prudce brzdit. Optimálním řešením by bylo, aby se v každém podlaží nacházela dvojice kontaktů, která by byla umístěna v krajních bodech. Tím by bylo dosaženo delší dráhy, která by umožňovala plynulejší zpomalování do podlaží.

Do logiky výtahu jsem zahrnul prioritizaci prvního tlačítka z výtahu. Důvod je ten, aby člověk, který nastoupí do prázdného výtahu, nebyl zmatený tím, že se s ním prázdný výtah rozjede opačným směrem, než který požaduje. Zároveň je tímto způsobem dosaženo menší obsazenosti výtahu. Výše uvedený scénář, kdy se směr výtahu neshoduje s požadovaným směrem pasažéra, je jediný případ, kdy dochází k vynucené změně směru jízdy. V ostatních případech výtah vždy seřadí frontu požadavků tak, že vykoná všechny požadavky v jednom směru a následně vykoná požadavky ve směru opačném. Výtah proto pracuje na principu nejkratší cesty s výjimkou tlačítka z výtahu.

Drobnou komplikaci při zhotovování mého řešení byly extrémy, tedy první a páté podlaží, jelikož požadovaný směr pasažéra je opačný než směr, který musí výtah vykonat, aby se do daného podlaží dostal.

Zvolený způsob řešení přináší jistá úskalí pro složitější situace, které by bylo optimální řešit složitějším algoritmem. Vzhledem k náročnosti simulací doporučuji nejprve otestovat funkčnost logické části programu před přidáním technologických objektů.

Především navrhování logiky řízení výtahu mě při vypracovávání této bakalářské práce bavilo a rád bych nabyté vědomosti zužitkoval v navazujícím studiu.

LITERATURA

[1] S.K. Sahdev, Electrical Machines. Cambridge University Press, 2018. ISBN 978-1-108-43106-4

[2] Hughes, Austin. Electric motors and drives: fundamentals, types and applications. 3rd ed. Newnes, 2006. ISBN 0-7506-4718-3.

[3] Kobrle, Pavel, Pavelka, Jiří. Elektrické pohony a jejich řízení. 3. přepracované vydání. České vysoké učení technické v Praze, 2019. ISBN 978-80-01-06007-0.

[4] Bill, Drury. The Control Techniques Drives and Controls Handbook. 2nd ed. The Institution of Engineering and Technology, 2009. ISBN 978-1-84919-01308.

[5] Motory s permanentními magnety. Motor.feld.cvut.cz [online]. Praha FEL, [cit. 2022-12-27]. Dostupné z: https://motor.feld.cvut.cz/sites/default/files/predmety/A1M14PO2/Prednaska_6_Spec_pohony.pdf.

[6] Lettl, Jiří, Jiří Pavelka a Jan Bauer. Výkonová elektronika. 4. přepracované vydání. Praha: České vysoké učení technické v Praze, 2019. ISBN 978-80-01-06514-3.

[7] Speciální obvody a jejich programování v C [online]. Praha FEL,2021 [cit. 2022-12-27]. Dostupné z: https://moodle.fel.cvut.cz/pluginfile.php/322430/mod_resource/content/1/B1B14MIS-2021-08z-spec-obvody-a-C.pdf

[8] What are Instruction Lists (ILs) for PLC programming?. Motioncontroltips.com [online]. 2017 [cit. 2022-12-27]. Dostupné z: https://www.motioncontroltips.com/instruction-lists-ils-plc-programming

[9] Sériový přenos informace. *Moodle.fel.cvut.cz* [online]. Praha FEL, 2021 [cit. 2022-12-27]. Dostupné z: https://moodle.fel.cvut.cz/pluginfile.php/324381/mod_resource/content/1/B1B14MIS-2021-15z-seriova-komunikace.pdf

[10] Serial Communication. Wikipedia [online]. 2022 [cit. 2022-12-27]. Dostupné z: https://en.wikibooks.org/wiki/Applied_Robotics/Microcontrollers/Serial_Communication

[11] CAN sběrnice. Canlab.cz [online]. [cit. 2022-12-27]. Dostupné z: https://www.canlab.cz/cs/can_bus?fbclid=IwAR3txO6wbSFX2GIMJLCRNif0fmZ4TfsmHrUMPmat9CnDAG0dBH7Mo4wRj_A

[12] SFC. Plc-automatizace.cz [online]. [cit. 2022-12-27]. Dostupné z: http://plc-automatizace.cz/knihovna/program/zobrazeni/AS-SFC.htm

[13] EtherCAT. Etherecat.org [online]. [cit. 2022-12-27]. Dostupné z: https://www.ethercat.org/en/technology.html

[14] AS-i. As-interface.net [online]. [cit. 2022-12-27]. Dostupné z: https://www.as-interface.net/en/technology

[15] System manual PLC S1200. Siemens.com [online]. 2018 [cit. 2022-12-27]. Dostupné z: https://support.industry.siemens.com/cs/document/109759862/simatic-s7-s7-1200-programmable-controller?dti=0&lc=en-CZ

[16] Principy komunikace a diagnostika sítí Profinet. Automa [online]. Praha: FEL, 2013 [cit. 2023-01-10]. Dostupné z: http://www.automa.cz/aton/filerepository/pdf_articles/10377.pdf

PŘÍLOHA A: SEZNAM SYMBOLŮ A ZKRATEK

A.1 Seznam symbolů

U (V)	napětí zdroje
R_a (Ω)	odpor kotvy
I_a (A)	proud kotvou
U_i (V)	indukované napětí
μ (−)	poměrná permeabilita
L_d (H)	indukčnost ve směru osy d
L_q (H)	indukčnost ve směru osy q
R_S (Ω)	odpor jedné fáze statorového vinutí
L_S (H)	náhradní indukčnost reakce kotvy
Φ_f (Wb)	magnetický tok permanentních magnetů
E (V)	napětí indukované magnetickým tokem Φ_f
U_S (V)	fázové napájecí napětí
ω (rad. s^{-1})	elektrická úhlová rychlost
$\hat{\phi}_f$ (Wb)	fázor magnetického toku
$\hat{U}_S$ (V)	fázor napájecího napětí
I_S (A)	statorový proud
$\hat{I}_S$ (A)	fázor statorového proudu
n (ot. $\min^{-1}$)	mechanická otáčivá rychlost rotoru
n_N (ot. $\min^{-1}$)	jmenovitá mechanická otáčivá rychlost rotoru
M (N. m)	moment synchronního stroje
p_p (–)	počet pólpárů
U_{ief} (V)	vnitřní napětí
X_d (Ω)	podélná reaktance
X_q (Ω)	příčná reaktance
β (rad)	zatěžovací úhel
I_{Sq} (A)	složka statorového proudu ve směru osy q
I_{Sd} (A)	složka statorového proudu ve směru osy d
I_{SMAX} (A)	maximální přípustný statorový proud
α (rad)	řídicí úhel
I_d (A)	proud stejnosměrným meziobvodem

A.1.1 Seznam zkratek

DC	stejnosměrný proud (Direct Current)
AC	střídavý proud (Alternating Current)
PM	permanentní magnet
PMSM	synchronní motor s permanentními magnety
SM	synchronní motor
PLC	programovatelný logický automat (Programmable Logic Controller)
ADC	analogově-digitální převodník (Analog-Digital Convertor)
DAC	digitálně-analogový převodník (Digital-Analog Convertor)
TIA	plně integrovaná automatizace (Totally integrated Automation)
LAD	žebříčkový diagram (Ladder Diagram)
NO	normálně otevřené (Normally open)
NC	normálně zavřené (Normally closed)
FBD	funkční blok (Function Block Diagram)
SCL	strukturovaný text (Structured Control Language)

IL	seznam instrukcí (Instruction List)
SFC	sekvenční funkční diagram (Sequential Function Chart)
HMI	rozhraní stroj-člověk (Human Machine Interface)
NRZ	bez návratu k nule (Non Return to Zero)
NRZI	bez návratu k nule s inversí (Non Return to Zero Inverted)
USB	universální sériová sběrnice (Universal Serial Bus)
CSMA/CD	(Carried Sense Multi Access/Collision Detection)
CSMA/CR	(Carried Sense Multi Access/Collision Resolution)
UART	(Universal asynchronous receiver-transmitter)
CAN	(Control Area Network)
PHY	fyzická vrstva (Physical Layer)
MAC	(Medium Access Control)
MII	(Medium Independent Interface)
AS-i	(Actuator Sensor Interface)
MC	(Motion Control)
RTC1	(Real Time Class 1)
RTC3	(Real Time Class 3)
NRT	(Non Real Time)

PŘÍLOHA B: PROGRAM V TIA PORTÁL

B.1 Seznam PLC Tagů

PLC tags

	Name	Tag table	Data type	Address	Retain	Acces...	Writa...	Visibl...
1	contact_stop_down	Default tag table	Bool	%I0.0	☐	☑	☑	☑
2	contact_stop_up	Default tag table	Bool	%I0.6	☐	☑	☑	☑
3	contact_p1	Default tag table	Bool	%I0.1	☐	☑	☑	☑
4	contact_p2	Default tag table	Bool	%I0.2	☐	☑	☑	☑
5	contact_p3	Default tag table	Bool	%I0.3	☐	☑	☑	☑
6	contact_p4	Default tag table	Bool	%I0.4	☐	☑	☑	☑
7	contact_p5	Default tag table	Bool	%I0.5	☐	☑	☑	☑
8	p1_pushbutton_up	Default tag table	Bool	%M0.1	☐	☑	☑	☑
9	p2_pushbutton_down	Default tag table	Bool	%M0.2	☐	☑	☑	☑
10	p2_pushbutton_up	Default tag table	Bool	%M0.3	☐	☑	☑	☑
11	p3_pushbutton_down	Default tag table	Bool	%M0.4	☐	☑	☑	☑
12	p3_pushbutton_up	Default tag table	Bool	%M0.5	☐	☑	☑	☑
13	Restart	Default tag table	Bool	%M0.6	☐	☑	☑	☑
14	p4_pushbutton_down	Default tag table	Bool	%M0.7	☐	☑	☑	☑
15	p4_pushbutton_up	Default tag table	Bool	%M1.0	☐	☑	☑	☑
16	p5_pushbutton_down	Default tag table	Bool	%M1.1	☐	☑	☑	☑
17	elevator_pb1	Default tag table	Bool	%M1.2	☐	☑	☑	☑
18	elevator_pb2	Default tag table	Bool	%M1.3	☐	☑	☑	☑
19	elevator_pb3	Default tag table	Bool	%M1.4	☐	☑	☑	☑
20	elevator_pb4	Default tag table	Bool	%M1.5	☐	☑	☑	☑
21	elevator_pb5	Default tag table	Bool	%M1.6	☐	☑	☑	☑
22	Axis_1_Drive_IN	Default tag table	"PD_TEL3_IN"	%I68.0	☐	☑	☑	☑
23	Axis_1_Drive_OUT	Default tag table	"PD_TEL3_OUT"	%Q68.0	☐	☑	☑	☑
24	Enable_Axis	Default tag table	Bool	%M1.7	☐	☑	☑	☑
25	Jog_forward	Default tag table	Bool	%M2.0	☐	☑	☑	☑
26	Jog_backward	Default tag table	Bool	%M2.1	☐	☑	☑	☑

Obr. B-1. Seznam PLC Tagů

B.2 Datový blok global variable

global_variable

	Name	Data type	Start value	Retain	Accessible f...	Writa...	Visible in ...
1	Static			☐	☐	☐	☐
2	Memory_orders	Array[0..12] of Int		☐	☑	☑	☑
3	power_up	Bool	false	☐	☑	☑	☑
4	Memory_direction	Array[0..12] of Bool		☐	☑	☑	☑
5	Memory_elevatorpb	Array[0..12] of Bool		☐	☑	☑	☑
6	homing_axis	Bool	false	☐	☑	☑	☑
7	number_of_orders	Int	0	☐	☑	☑	☑
8	number_of_elevator_...	Int	0	☐	☑	☑	☑
9	Position_elevator	Int	0	☐	☑	☑	☑
10	max_array	Int	5	☐	☑	☑	☑
11	direction_for_HMI	Int	0	☐	☑	☑	☑
12	error_signal	Bool	false	☐	☑	☑	☑
13	elevator_motion	Bool	false	☐	☑	☑	☑
14	timer_output	Bool	false	☐	☑	☑	☑
15	halt	Bool	false	☐	☑	☑	☑
16	halt_done	Bool	false	☐	☑	☑	☑
17	Move_Velocity_Execute	Bool	false	☐	☑	☑	☑
18	Move_Velocity_Directi...	Int	0	☐	☑	☑	☑
19	timer_input	Bool	false	☐	☑	☑	☑
20	Close_elevator	Bool	false	☐	☑	☑	☑
21	button_p1_up	Bool	false	☐	☑	☑	☑
22	button_p2_up_	Bool	false	☐	☑	☑	☑
23	button_p2_down	Bool	false	☐	☑	☑	☑
24	button_p3_up	Bool	false	☐	☑	☑	☑
25	button_p3_down	Bool	false	☐	☑	☑	☑
26	button_p4_up	Bool	false	☐	☑	☑	☑
27	button_p4_down	Bool	false	☐	☑	☑	☑
28	button_p5_down	Bool	false	☐	☑	☑	☑
29	button_p1_elevator	Bool	false	☐	☑	☑	☑
30	button_p2_elevator	Bool	false	☐	☑	☑	☑
31	button_p3_elevator	Bool	false	☐	☑	☑	☑
32	button_p4_elevator	Bool	false	☐	☑	☑	☑
33	button_p5_elevator	Bool	false	☐	☑	☑	☑

Obr. B-2. Datový blok global variable

B.3 Funkční blok Elevator

```
//Pozice výtahu
IF #contact_p1 THEN
    "global_variable".Position_elevator := 1;
    "Order_done"();
ELSIF #contact_p2 THEN
    "global_variable".Position_elevator := 2;
    "Order_done"();
ELSIF #contact_p3 THEN
    "global_variable".Position_elevator := 3;
    "Order_done"();
ELSIF #contact_p4 THEN
    "global_variable".Position_elevator := 4;
    "Order_done"();
ELSIF #contact_p5 THEN
    "global_variable".Position_elevator := 5;
    "Order_done"();
END_IF;

//tlačítko 1.patro nahoru
IF #p1_pb_up THEN
    IF #aretace_p1_up = FALSE THEN
```

```
            IF NOT("global_variable".Position_elevator = 1) THEN
                "global_variable".button_p1_up := TRUE;
            END_IF;
            "New_Order"(floor := 1,direction_up := TRUE, elevator_signal :=
FALSE, elevator_motion := "global_variable".elevator_motion);
            #aretace_p1_up := TRUE;
        END_IF;
    ELSE
        #aretace_p1_up := FALSE;
    END_IF;

    //tlačítko 2.patro nahoru
    IF #p2_pb_up THEN
        IF #aretace_p2_up = FALSE THEN
            IF NOT("global_variable".Position_elevator = 2) THEN
                "global_variable"."button_p2_up_" := TRUE;
            END_IF;
            "New_Order"(floor := 2,direction_up := TRUE, elevator_signal:=
FALSE, elevator_motion := "global_variable".elevator_motion);
            #aretace_p2_up := TRUE;
        END_IF;
    ELSE
        #aretace_p2_up := FALSE;
    END_IF;

    //tlačítko 2.patro dolu
    IF #p2_pb_down THEN
        IF #aretace_p2_down = FALSE THEN
            IF NOT("global_variable".Position_elevator = 2) THEN
                "global_variable".button_p2_down := TRUE;
            END_IF;
            "New_Order"(floor := 2,direction_up := FALSE, elevator_signal:=
FALSE, elevator_motion := "global_variable".elevator_motion);
            #aretace_p2_down := TRUE;
        END_IF;
    ELSE
        #aretace_p2_down := FALSE;
    END_IF;

    //tlačítko 3.patro nahoru
    IF #p3_pb_up THEN
        IF #aretace_p3_up = FALSE THEN
            IF NOT("global_variable".Position_elevator = 3) THEN
                "global_variable".button_p3_up := TRUE;
            END_IF;
            "New_Order"(floor := 3,direction_up := TRUE, elevator_signal:=
FALSE, elevator_motion := "global_variable".elevator_motion);
            #aretace_p3_up := TRUE;
        END_IF;
    ELSE
        #aretace_p3_up := FALSE;
    END_IF;

    //tlačítko 3.patro dolu
    IF #p3_pb_down THEN
        IF #aretace_p3_down = FALSE THEN
            IF NOT("global_variable".Position_elevator = 3) THEN
```

```
                "global_variable".button_p3_down := TRUE;
            END_IF;
            "New_Order"(floor := 3,direction_up := FALSE, elevator_signal:=
FALSE, elevator_motion := "global_variable".elevator_motion);
           #aretace_p3_down := TRUE;
        END_IF;
    ELSE
        #aretace_p3_down := FALSE;
    END_IF;

    //tlačítko 4.patro nahoru
    IF #p4_pb_up THEN
        IF #aretace_p4_up = FALSE THEN
            IF NOT("global_variable".Position_elevator = 4) THEN
                "global_variable".button_p4_up := TRUE;
            END_IF;
            "New_Order"(floor := 4,direction_up := TRUE, elevator_signal:=
FALSE, elevator_motion := "global_variable".elevator_motion);
           #aretace_p4_up := TRUE;
        END_IF;
    ELSE
        #aretace_p4_up := FALSE;
    END_IF;

    //tlačítko 4.patro dolu
    IF #p4_pb_down THEN
        IF #aretace_p4_down = FALSE THEN
            IF NOT("global_variable".Position_elevator = 4) THEN
                "global_variable".button_p4_down := TRUE;
            END_IF;
            "New_Order"(floor := 4,direction_up := FALSE, elevator_signal:=
FALSE, elevator_motion := "global_variable".elevator_motion);
            #aretace_p4_down := TRUE;
        END_IF;
    ELSE
        #aretace_p4_down := FALSE;
    END_IF;

    //tlačítko 5.patro dolu
    IF #p5_pb_down THEN
        IF #aretace_p5_down = FALSE THEN
            IF NOT("global_variable".Position_elevator = 5) THEN
                "global_variable".button_p5_down := TRUE;
            END_IF;
            "New_Order"(floor := 5,direction_up := FALSE, elevator_signal:=
FALSE, elevator_motion := "global_variable".elevator_motion);
            #aretace_p5_down := TRUE;
        END_IF;
    ELSE
        #aretace_p5_down := FALSE;
    END_IF;

    //tlačítko výtah 1.patro
    IF #elevator_pb1 THEN
        IF #aretace_elevator_p1 = FALSE THEN
            IF NOT("global_variable".Position_elevator = 1) THEN
                "global_variable".button_p1_elevator := TRUE;
```

```
            END_IF;
            "New_Order"(floor := 1,
                        direction_up := FALSE,
                        elevator_signal := TRUE,
                        elevator_motion :=
"global_variable".elevator_motion);
                #aretace_elevator_p1 := TRUE;
        END_IF;
    ELSE
        #aretace_elevator_p1 := FALSE;
    END_IF;

    //tlačítko výtah 2.patro
    IF #elevator_pb2 THEN
        IF #aretace_elevator_p2 = FALSE THEN
            IF NOT("global_variable".Position_elevator = 2) THEN
                "global_variable".button_p2_elevator := TRUE;
            END_IF;
            "New_Order"(floor := 2,
                        direction_up :=
("global_variable".Position_elevator < 2),
                        elevator_signal := TRUE,
                        elevator_motion :=
"global_variable".elevator_motion);
            #aretace_elevator_p2 := TRUE;
        END_IF;
    ELSE
        #aretace_elevator_p2 := FALSE;
    END_IF;

    //tlačítko výtah 3.patro
    IF #elevator_pb3 THEN
        IF #aretace_elevator_p3 = FALSE THEN
            IF NOT("global_variable".Position_elevator = 3) THEN
                "global_variable".button_p3_elevator := TRUE;
            END_IF;
            "New_Order"(floor := 3,
                        direction_up :=
("global_variable".Position_elevator < 3),
                        elevator_signal := TRUE,
                        elevator_motion :=
"global_variable".elevator_motion);
            #aretace_elevator_p3 := TRUE;
        END_IF;
    ELSE
        #aretace_elevator_p3 := FALSE;
    END_IF;

    //tlačítko výtah 4.patro
    IF #elevator_pb4 THEN
        IF #aretace_elevator_p4 = FALSE THEN
            IF NOT ("global_variable".Position_elevator = 4) THEN
                "global_variable".button_p4_elevator := TRUE;
            END_IF;
            "New_Order"(floor := 4,
                        direction_up :=
("global_variable".Position_elevator < 4),
```

```
                        elevator_signal := TRUE,
                        elevator_motion :=
"global_variable".elevator_motion);
            #aretace_elevator_p4 := TRUE;
        END_IF;
    ELSE
        #aretace_elevator_p4 := FALSE;
    END_IF;

    //tlačítko výtah 5.patro
    IF #elevator_pb5 THEN
        IF #aretace_elevator_p5 = FALSE THEN
            IF NOT ("global_variable".Position_elevator = 5) THEN
                "global_variable".button_p5_elevator := TRUE;
            END_IF;
            "New_Order"(floor := 5,
                        direction_up := TRUE,
                        elevator_signal := TRUE,
                        elevator_motion :=
"global_variable".elevator_motion);
            #aretace_elevator_p5 := TRUE;
        END_IF;
    ELSE
        #aretace_elevator_p5 := FALSE;
    END_IF;

    //tlačítko restart
    IF #Restart_pb THEN
        "Reset"();
    END_IF;

    //error - zablokování osy
    IF #contact_stop_down OR #contact_stop_up THEN
        "global_variable".error_signal := TRUE;
    END_IF;

    IF "global_variable".error_signal THEN
        "global_variable".halt := TRUE;
    END_IF;

    //Pohyb výtahu
        IF "global_variable".number_of_orders > 0 AND
"global_variable".elevator_motion = FALSE THEN
            IF "global_variable".Memory_orders[0] >
"global_variable".Position_elevator THEN
                "global_variable".Move_Velocity_Direction := 1;
                "global_variable".Move_Velocity_Execute := TRUE;
                "global_variable".halt := FALSE;
                "global_variable".direction_for_HMI := 1;
            END_IF;

            IF "global_variable".Memory_orders[0] <
"global_variable".Position_elevator THEN
                "global_variable".Move_Velocity_Direction := 2;
                "global_variable".Move_Velocity_Execute := TRUE;
                "global_variable".halt := FALSE;
```

```
                "global_variable".direction_for_HMI := 2;
            END_IF;
        END_IF;
        IF "global_variable".elevator_motion  THEN
            "global_variable".Move_Velocity_Execute := FALSE;
        END_IF;


    IF "global_variable".number_of_orders = 0 THEN
        "global_variable".direction_for_HMI := 0;
    END_IF;

    //Halt Done
    IF "global_variable".halt_done THEN
        "global_variable".halt := FALSE;
    END_IF;
```

B.4 Funkce New Order

```
    IF NOT ("global_variable".Position_elevator = #floor) THEN
        "Find_same_value"(Floor := #floor,
                          direction_up := #direction_up,
                          elevator_signal := #elevator_signal,
                          Exist => #Existing_value);
        IF #Existing_value THEN
            RETURN;
        END_IF;
            "Where_to_shift"(floor := #floor,
                             direction_up := #direction_up,
                             preferred_signal := #elevator_signal,
                             elevator_motion := #elevator_motion,
                             Position => #Position_Where);
            "Shift_array"(shift_pos := #Position_Where,
                          shift_right := TRUE);
            "Set_Order"(Position := #Position_Where,
                        floor := #floor,
                        direction_up := #direction_up,
                        elevator_signal := #elevator_signal);
    END_IF;
```

B.5 Funkce Where to shift

```
IF "global_variable".number_of_orders = 0 THEN
        #Position := 0;
        RETURN;
END_IF;

    IF "global_variable".Position_elevator <
"global_variable".Memory_orders[0] THEN
        #logic_direction_up := TRUE;
    ELSE
        #logic_direction_up := FALSE;
    END_IF;

    IF (#preferred_signal AND ("global_variable".number_of_elevator_orders
= 0)) THEN
        IF #elevator_motion = FALSE THEN
            IF NOT ("global_variable".Memory_direction[0] = #direction_up)
THEN
```

```
                #Position := 0;
                RETURN;
            END_IF;
        END_IF;
    END_IF;

    #counter := 0;

    IF #logic_direction_up THEN
        IF #direction_up THEN
            IF "global_variable".Position_elevator < #floor AND #floor <=
"global_variable".Memory_orders[0] THEN
                #Position := 0;
                RETURN;
            END_IF;
        END_IF;
    ELSE
        IF #direction_up = FALSE THEN
            IF "global_variable".Position_elevator > #floor AND #floor >=
"global_variable".Memory_orders[0] THEN
                #Position := 0;
                RETURN;
            END_IF;
        END_IF;
    END_IF;
    #counter += 1;
    WHILE (#counter < "global_variable".number_of_orders) DO
        IF "global_variable".Memory_direction[#counter - 1] THEN
            IF "global_variable".Memory_direction[#counter] THEN
                IF #direction_up AND
                    "global_variable".Memory_orders[#counter - 1] < #floor
AND
                    "global_variable".Memory_orders[#counter] >= #floor
THEN
                    #Position := #counter;
                    RETURN;
                END_IF;
            ELSE
                IF #direction_up THEN
                    IF "global_variable".Memory_orders[#counter - 1] <
#floor AND "global_variable".max_array >= #floor THEN
                        #Position := #counter;
                        RETURN;
                    END_IF;
                ELSE
                    IF "global_variable".Memory_orders[#counter] <= #floor
AND "global_variable".max_array >= #floor THEN
                        #Position := #counter;
                        RETURN;
                    END_IF;
                END_IF;
            END_IF;
        ELSE
            IF "global_variable".Memory_direction[#counter] = FALSE THEN
                IF #direction_up = FALSE AND
                    "global_variable".Memory_orders[#counter - 1] > #floor
AND
```

```
                    "global_variable".Memory_orders[#counter] <= #floor
THEN
                    #Position := #counter;
                    RETURN;
                END_IF;
            ELSE
                IF #direction_up THEN
                    IF 1 <= #floor AND
"global_variable".Memory_orders[#counter] >= #floor THEN
                        #Position := #counter;
                        RETURN;
                    END_IF;
                ELSE
                    IF "global_variable".Memory_orders[#counter - 1] >
#floor AND 1 <= #floor THEN
                        #Position := #counter;
                        RETURN;
                    END_IF;
                END_IF;
            END_IF;
        END_IF;
        #counter += 1;
    END_WHILE;

    #Position := #counter;
```

B.6 Funkce Shift Array

```
    IF #shift_right THEN
        FOR #i := 0 TO "global_variable".number_of_orders - #shift_pos - 1
DO

"global_variable".Memory_orders["global_variable".number_of_orders - #i] :=
"global_variable".Memory_orders["global_variable".number_of_orders - #i -
1];

"global_variable".Memory_direction["global_variable".number_of_orders - #i]
:= "global_variable".Memory_direction["global_variable".number_of_orders -
#i - 1];

"global_variable".Memory_elevatorpb["global_variable".number_of_orders -
#i] :=
"global_variable".Memory_elevatorpb["global_variable".number_of_orders - #i
- 1];
        END_FOR;

    ELSE
        IF "global_variable".number_of_orders > 1 THEN
            FOR #i := #shift_pos TO "global_variable".number_of_orders - 1
DO
                "global_variable".Memory_orders[#i] :=
"global_variable".Memory_orders[#i + 1];
                "global_variable".Memory_direction[#i] :=
"global_variable".Memory_direction[#i + 1];
                "global_variable".Memory_elevatorpb[#i] :=
"global_variable".Memory_elevatorpb[#i + 1];
            END_FOR;
        END_IF;
```

```
        "global_variable".Memory_orders["global_variable".number_of_orders-
1] := 0;

"global_variable".Memory_direction["global_variable".number_of_orders-1] :=
FALSE;

"global_variable".Memory_elevatorpb["global_variable".number_of_orders-1]
:= FALSE;
    END_IF;
```

B.7 Funkce Find same value

```
    FOR #i := 0 TO "global_variable".number_of_orders-1 DO
        IF "global_variable".Memory_orders[#i] = #Floor AND
            "global_variable".Memory_direction[#i] = #direction_up AND
            "global_variable".Memory_elevatorpb[#i] = #elevator_signal THEN
            #Exist := TRUE;
            RETURN;
        END_IF;
    END_FOR;

    #Exist := FALSE;
```

B.8 Funkce Set Order

```
    "global_variable".Memory_orders[#Position] := #floor;
    "global_variable".Memory_direction[#Position] := #direction_up;
    "global_variable".Memory_elevatorpb[#Position] := #elevator_signal;
    "global_variable".number_of_orders += 1;
    IF #elevator_signal THEN
        "global_variable".number_of_elevator_orders += 1;
    END_IF;
```

B.9 Funkce Order done

```
    WHILE "global_variable".number_of_orders > 0 AND
"global_variable".Position_elevator = "global_variable".Memory_orders[0] DO
        IF "global_variable".Memory_elevatorpb[0] = TRUE THEN
            "global_variable".number_of_elevator_orders -= 1;
        END_IF;
        "Light_off"();
        "global_variable".halt := TRUE;
        "global_variable".Close_elevator := FALSE;
        "Shift_array"(shift_pos := 0,
                      shift_right := FALSE);
            "global_variable".number_of_orders -= 1;

    END_WHILE;
```

B.10 Funkce Reset

```
    FOR #i := 0 TO 12 DO
        "global_variable".Memory_orders[#i] := 0;
        "global_variable".Memory_direction[#i] := FALSE;
        "global_variable".Memory_elevatorpb[#i] := FALSE;

    END_FOR;

    "global_variable".number_of_orders := 0;
    "global_variable".number_of_elevator_orders := 0;
    "global_variable".direction_for_HMI := 0;
    "global_variable".error_signal := 0;
```

```
"global_variable".halt := FALSE;
"global_variable".Move_Velocity_Execute := FALSE;
"global_variable".button_p1_up := FALSE;
"global_variable"."button_p2_up_" := FALSE;
"global_variable".button_p3_up := FALSE;
"global_variable".button_p4_up := FALSE;
"global_variable".button_p2_down := FALSE;
"global_variable".button_p3_down := FALSE;
"global_variable".button_p4_down := FALSE;
"global_variable".button_p5_down := FALSE;
"global_variable".button_p1_elevator := FALSE;
"global_variable".button_p2_elevator := FALSE;
"global_variable".button_p3_elevator := FALSE;
"global_variable".button_p4_elevator := FALSE;
"global_variable".button_p5_elevator := FALSE;
```

B.11 Funkce Light off

```
IF "global_variable".Memory_orders[0] = 1 THEN
    "global_variable".button_p1_elevator := FALSE;
    "global_variable".button_p1_up := FALSE;
END_IF;

IF "global_variable".Memory_orders[0] = 2 THEN
    IF "global_variable".Memory_elevatorpb[0] THEN
        "global_variable".button_p2_elevator := FALSE;
    ELSE
        IF "global_variable".Memory_direction[0] THEN
            "global_variable"."button_p2_up_" := FALSE;
        ELSE
            "global_variable".button_p2_down := FALSE;
        END_IF;
    END_IF;
END_IF;

IF "global_variable".Memory_orders[0] = 3 THEN
    IF "global_variable".Memory_elevatorpb[0] THEN
        "global_variable".button_p3_elevator := FALSE;
    ELSE
        IF "global_variable".Memory_direction[0] THEN
            "global_variable".button_p3_up := FALSE;
        ELSE
            "global_variable".button_p3_down := FALSE;
        END_IF;
    END_IF;
END_IF;

IF "global_variable".Memory_orders[0] = 4 THEN
    IF "global_variable".Memory_elevatorpb[0] THEN
        "global_variable".button_p4_elevator := FALSE;
    ELSE
        IF "global_variable".Memory_direction[0] THEN
            "global_variable".button_p4_up := FALSE;
        ELSE
            "global_variable".button_p4_down := FALSE;
        END_IF;
    END_IF;
END_IF;
```

```
IF "global_variable".Memory_orders[0] = 5 THEN
    "global_variable".button_p5_elevator := FALSE;
    "global_variable".button_p5_down := FALSE;

END_IF;
```