VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

SOFTWAROVÝ FRAMEWORK PRO STANDARDIZACI PLC PROGRAMU
SOFTWARE FRAMEWORK FOR PLC PROGRAM STANDARDIZATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
Otto Hrubý

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Miroslav Jirgl, Ph.D.

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Student: Otto Hrubý ***ID:*** 211422

Ročník: 3 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Softwarový framework pro standardizaci PLC programu

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je implementace SW frameworku pro účely standardizace návrhu SW pro zařízení na bázi PLC v rámci firemního použití ve společnosti ALPS Electric Czech, s.r.o.

1. Prostudujte normu IEC61131-3 a její možnosti uplatnění v OOP návrhu PLC systémů.
2. Analyzujte existující programy používané v zařízeních používaných ve firmě a identifikujte často používané kódové vzory a idiomy.
3. Na základě analýzy navrhněte základní strukturu programu pro jednoúčelové zařízení používané v tzv. assembly procesech.
4. Program doplňte o proměnné pro řízení toku programu a aktuálním stavu zařízení pro dotazy z Master PLC.
5. Navrhněte program pro komunikaci mezi zařízeními Master-Slave, který je nezávislý na použité sběrnici a otestujte jej na sběrnici EtherNet/IP a TCP/IP Socket.
6. Z výše uvedených programů navrhněte opakovatelně použitelný framework pro PLC zařízení v roli slave, včetně referenční dokumentace.
7. Řešení ověřte a otestujte na zvoleném hardware/zařízení.

DOPORUČENÁ LITERATURA:

PÁSEK, Jan. Programovatelné automaty v řízení technologických procesů [online]. Brno: FEKT VUT v Brně, 2007 [cit. 2020-09-14].

Termín zadání: 8.2.2021 ***Termín odevzdání:*** 24.5.2021

Vedoucí práce: Ing. Miroslav Jirgl, Ph.D.

doc. Ing. Václav Jirsík, CSc.
předseda rady studijního programu

Abstrakt

Tato práce se zabývá standardizací softwaru pro PLC v rámci firemního využití. Cílem práce bylo navrhnout a implementovat program pro jednoúčelová zařízení používané v assembly procesu a program pro komunikaci umožňující nasazení zařízení v Master-Slave systému v roli slave. Program využívá prvky z normy IEC 61131-3, aby byla zabezpečena snadná přenositelnost programu pro různá vývojová prostředí. Software byl navržen tak, aby se dal lehce používat a měnit pro různá zařízení.

Klíčová slova

PLC, IEC 61131-3, framework, jednoúčelové zařízení, konečný stavový automat

Abstract

This thesis deals with standardization of PLC software used in company. The aim of this thesis was to design program for single-purpose machine used in assembly process and communication program to allow machines to be used in Master-Slave system as Slave. The programs use elements from IEC 61131-3 to make them portable among different integrated development environments. The software was designed to be easily reusable and modifiable for different machines.

Keywords

PLC, IEC 61131-3, framework, single-purpose machine, finite state machine

Bibliografická citace

HRUBÝ, Otto. Softwarový framework pro standardizaci PLC programu. Brno, 2021. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/134754. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Miroslav Jirgl.

Prohlášení autora o původnosti díla

Jméno a příjmení studenta:	Otto Hrubý
VUT ID studenta:	211422
Typ práce:	Bakalářská práce
Akademický rok:	2020/21
Téma závěrečné práce:	Softwarový framework pro standardizaci PLC programu

Prohlašuji, že svou závěrečnou práci na téma Softwarový framework pro standardizaci PLC programu jsem vypracoval samostatně pod vedením vedoucího závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: 24. května 2021 ----------

podpis autora

Poděkování

Rád bych poděkoval panu Ing. Dávidovi Genčanskému za odborné rady a cenné připomínky. Zároveň bych chtěl poděkovat vedoucímu bakalářské práce panu Ing. Miroslavovi Jirglovi, Ph.D. za odborné vedení.

V Brně dne: 24. května 2021 ----------

podpis autora

Obsah

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Semestrální práce byla vytvořena ve spolupráci s firmou ALPS Electric Czech, s.r.o. se sídlem v Sebranicích u Boskovic za účelem standardizace návrhu softwaru pro jednoúčelová zařízení ovládaná logickým programovatelným automatem (PLC) pomocí frameworku, který se skládá z programů, funkčních bloků (FB), funkcí a dokumentace.

Některé části frameworku jsou předem vytvořené a definují pro různá zařízení stejnou softwarovou architekturu. Použitím těchto částí se zrychlí zavádění nových zařízení, neboť nebude potřeba začínat s vývojem softwaru vždy od začátku. Jednotná struktura usnadní také budoucí údržbu softwaru.

Další části jsou pro různá zařízení specifická. Jejich tvorbu lze urychlit pomocí vytvoření šablon s ukázkovými příklady, které jsou obsaženy v referenční dokumentaci.

Cílem této bakalářské práce je nejdříve nastudovat normu IEC 61131-3 pro PLC a její uplatnění v objektově orientovaném návrhu. Dále pak analyzovat programy používané ve firmě a vyhledat v nich často používané prvky. Na základě provedené analýzy bude vytvořen hlavní program pro jednoúčelová zařízení. Od programu se požaduje, aby byl přenositelný pro různé výrobce PLC a aby se skládal z vícero menších celků pro umožnění rychlejších úprav programu.

Následně bude vytvořen program pro komunikaci, od kterého se požaduje, aby umožnil zařízením pracovat v Master-Slave systému jako slave. Program bude umožňovat nasazení nezávisle na použitém komunikačním protokolu, tak aby byl vhodný pro univerzální použití.

Na základě vytvořených programů bude vytvořena referenční dokumentace s ukázkovými příklady použití pro snazší práci s frameworkem. Výsledky práce budou na závěr otestovány na reálném zařízení.

1. NORMA IEC 61131-3

V dnešní době zastává v procesní automatizaci software čím dál důležitější roli. Požaduje se efektivní, odladěný, přenositelný kód, u kterého je možné rychle měnit a vytvářet nové funkcionality v závislosti na aktuální poptávce. Zároveň se ale zvyšují výdaje za software, a proto je snaha jeho vývoj co nejvíce zefektivňovat. [1]

Právě proto byla vytvořena mezinárodní norma IEC 61131-3 pro PLC, která se zabývá standardizací základní softwarové architektury a programovacích jazyků. [2]

1.1 Programovací jazyky

Norma definuje sémantiku a syntaxi 5 programovacích jazyků [2]:

- Ladder Diagram (LD) – grafický, dříve nejpoužívanější [3]
- Sequencial Function Chart (SFC) – grafický
- Function Block Diagram (FBD) – grafický
- Structured Text (ST) – textový, dnes nejpoužívanější [3]
- Instruction List (IL) – textový, nejstarší, v dalším vydání bude odstraněn

1.2 Datové typy

Základní datové typy definované normou se oproti datovým typům používaných v běžných programovacích jazycích příliš neliší. Mezi základní datové typy patří `BOOL`, `BYTE`, `WORD`, `INTEGER`, `REAL`, `STRING`, `DATE`, `TIME` a další. Uživatel si může z těchto typů vytvářet své vlastní datové typy (struktury), tzv. UDT. [2]

1.3 Programové organizační jednotky

Programová organizační jednotka (POU) je společný název pro program, funkční blok, funkci i třídu a slouží jak k modularizaci, tak ke strukturování kódu. Každou POU tvoří její název, typ, deklarace proměnných a tělo. Volba programovacího jazyka, ve kterým je daná POU napsána závisí na uživateli. V rámci projektu lze programovací jazyky různě střídat. [2, 4]

1.3.1 Funkční blok

Funkční blok generuje výstupní hodnoty v závislosti na vstupních hodnotách a jeho paměti. Volání funkčního bloku se dá provést jedině pomocí instancí. V rámci normy jsou implementovány základní funkční bloky pro bistabilní klopné obvody, detekci náběžné či sestupné hrany logické hodnoty, počítadla, časovače a další. [2]

1.3.2 Funkce

Funkce se odlišuje oproti funkčnímu bloku tím, že nemá paměť, a proto výsledek závisí pouze na vstupních hodnotách. Podobně jako u funkčních bloků norma implementuje standardní funkce například pro konverzi mezi různými datovými typy, matematické operace, zpracování textových řetězů, bitové operace a další. [2]

1.3.3 Třída

Třída obsahuje metody a proměnné, které mohou být veřejné nebo privátní. Na rozdíl od funkčního bloku neobsahuje ani vstupní, ani výstupní proměnné. U třídy lze volat pouze jednotlivé metody. [4]

1.3.4 Program

Hlavní úloha programu je zajistit výměnu dat mezi jednotlivými POU a vytvořit mezi nimi logické vazby. Všechny hodnoty jsou uchovávány do dalšího volání programu. [2]

1.4 Norma a objektově orientované programování

V minulosti vypadala struktura programu pro PLC tak, že se skládala z mnoha úloh, například pro čtení, filtrování, zpracování vstupů, zapsání výstupů. Problém nastal, pokud se chtěly do systému přidávat další funkcionality. Musely se projít a modifikovat všechny úlohy. Složitější změny úloh navíc často způsobovaly chyby v jiných částech programu.

Můžeme také zvolit jiný přístup. Program složíme z různých objektů. Pod pojmem objekty si můžeme představit třeba senzory, motory, pneumatické válce či kamery. Přidávání funkcionalit je nyní mnohem jednodušší. Stačí pouze přidávat nové objekty. Tento programovací styl se již dlouho používá v informatice pod názvem objektově orientované programování (OOP). Účelem je, aby byl program lehce modifikovatelný, přehlednější, přenositelnější a lépe se dal testovat. [5]

Právě aktuální třetí vydání normy IEC 61131-3 vzniklo v roce 2013 jakožto doplnění druhého vydání z roku 2003 zejména o prvky pro objektově orientované programování jako jsou dědění, polymorfismus, metody nebo rozhraní. [4]

1.4.1 Zapouzdření

Objekt v sobě obsahuje veškerou potřebnou funkcionalitu. Implementujeme jej například pomocí funkčního bloku, který si můžeme představit jako černou skříňku, jelikož k vnitřním proměnným objektu nemáme zvnějšku přístup. Předcházíme tak vzniku nepředvídatelných, špatně vyhledatelných chyb, poněvadž zabráníme jiným částem programu, aby tato data nedopatřením nepoškodily. Uživatel ani ostatní objekty navíc ani nepotřebují vědět, jak daný objekt uvnitř funguje. Potřebují pouze vědět, jak s objektem zacházet. [5]

1.4.2 Tvoření instancí

Abychom mohli efektivně využít zapouzdření, tak potřebujeme vytvořit více kopií objektu, které spolu nesdílí data. Díky schopnosti tvořit instance lze využívat kopie objektu bez toho, aniž bychom museli znovu tvořit samostatné POU, které by obsahovaly stejný kód, akorát by se jinak jmenovaly. Počet instancí je omezený pamětí. [2, 5]

Objekty navíc mohou vytvářet instance dalších objektů. Můžeme tak vytvořit objekt s hierarchickou strukturou. Pomocí vnořených objektů lze rozdělit program do vícero úrovní. [6]

1.4.3 Metody

Metoda se podobá funkci. Implementace se provádí uvnitř funkčního bloku nebo třídy pomocí klíčového slova `METHOD`. Když v rámci metody použijeme klíčové slovo `THIS`, tak se nám umožní přístup k vnitřním proměnným či metodám objektu. Norma definuje následující typy metod [4]:

- `PUBLIC` – může být volána odkudkoliv
- `PRIVATE` – může být volána pouze uvnitř objektu, ve kterém je definovaná
- `PROTECTED` – může být volána pouze uvnitř objektu, ve kterém je definovaná nebo z odvozeného objektu
- `INTERNAL` – může být volána pouze v rámci stejného jmenného prostoru
- `FINAL` – nemůže být přepsána

1.4.4 Dědičnost

Dědičnost nám dovoluje od základního funkčního bloku (rodiče) vytvořit odvozený funkční blok (potomka), který obsahuje veškeré funkcionality rodiče. Potomkovi se dají navíc implementovat další funkcionality. Dědit mohou také rozhraní a třídy. Ve vývojovém prostředí použijeme dědičnost pomocí klíčového slova `EXTENDS`.

Důvod proč používat dědičnost je ten, že můžeme přejímat již existující funkcionality definované v rodičovi bez zbytečného kopírování kódu. Při změně funkcionality v rodičovi se nám změny ihned projeví i do potomků. [4]

1.4.5 Polymorfismus

Polymorfismus souvisí s dědičností, neboť dovoluje potomkům upravovat zděděné funkcionality. Pokud chceme přepsat nějakou metodu, tak stačí pouze uvnitř potomka vytvořit metodu se stejným názvem, ale s jiným obsahem. Potomek má také možnost přistupovat k proměnným nebo metodám rodiče pomocí klíčového slova `SUPER`. [4]

1.4.6 Rozhraní

Rozhraní obsahuje prototypy metod. Prototypy metod se skládají pouze z definice bez žádné implementace, kterou později provede potomek. Pro použití rozhraní použijeme v potomkovi klíčové slovo `IMPLEMENTS`. Rozhraní nám pomáhá zbavit se některých

nevýhod dědičnosti, poněvadž na rozdíl od funkčního bloku nebo třídy umožňuje vícenásobné dědění. Rozhraním nám také umožňuje jednotně přistupovat a ovládat různé FB se stejným rozhraním. [4, 6]

1.5 Existující knihovny

Organizace PLCopen se zabývá standardizací aplikací pro řízení motorů, komunikaci a bezpečnost pomocí vytvoření knihoven obsahující FB. Zdrojové kódy k nim však nejsou dostupné. Nelze je tak lehce upravovat a přenášet do libovolných vývojových prostředí.

Volně přístupnou knihovnu poskytuje komunita Open Source Community for Automation Technology (OSCAT) obsahující funkce a FB například pro komunikaci, měření, řízení. Zdrojové kódy jsou napsané v jazyku ST. Použití grafických jazyků není efektivní pro tvorbu snadno přenositelného kódu.

Obě knihovny ctí normu IEC 61131-3. Pro FB obsažené v obou knihovnách platí, že obsahují výstupy informující o jejich aktuálním stavu. Vlastní aplikace se vytváří pomocí opakovaného použití již vytvořených a otestovaných bloků s podobným rozhraním. [7, 8]

2. KOMUNIKACE V PRŮMYSLU

Framework je vyvíjen pro zařízení, která se budou ovládat pomocí komunikačního programu. Mezi nejčastěji používané komunikační protokoly v průmyslu na základě průzkumu z roku 2019 uvedeného např. v [9] patří EtherNet/IP, PROFINET, EtherCAT a Modbus TCP Ještě v roce 2008 měl však na základě průzkumu – viz [10] dominantní postavení protokol TCP/IP Socket.

2.1 EtherNet/IP

EtherNet/Industrial Protocol (EtherNet/IP) byl vyvinut společnostmi Open DeviceNet Vendors Association (ODVA) a Rockwell Automation tak, aby byl kompatibilní s klasickým Ethernetem TCP/IP. [11]

V síti EtherNet/IP se používají 2 způsoby komunikaci:

- Explicitní přenos – využívá protokol TCP/IP k přenosu typu žádost-odpověď
- Implicitní přenos – využívá protokol UDP/IP k cyklickému přenosu vstupních a výstupních dat

EtherNet/IP definuje tři typy zařízení:

- Messaging class – určen pro explicitní přenos zpráv
- Scanner class – zajišťuje spojení a přenos dat se zařízením typu adapter
- Adapter class – zpracovává data a čeká na zahájení spojení

2.2 PROFINET

PROFINET je otevřený standard průmyslového Ethernetu vytvořený organizací Profibus International (PI). [12]

V síti Profinet se používají 3 kanály pro různé způsoby komunikace:

- Standardní komunikace – obsahuje základní funkce, používá se pro nekritické procesy
- Komunikace pro reálný čas – používá se pro průmyslovou automatizaci a řízení procesů
- Izochronní časově synchronizovaná komunikace – používá se pro řízení pohybu s požadavkem na nízkou dobu odezvy

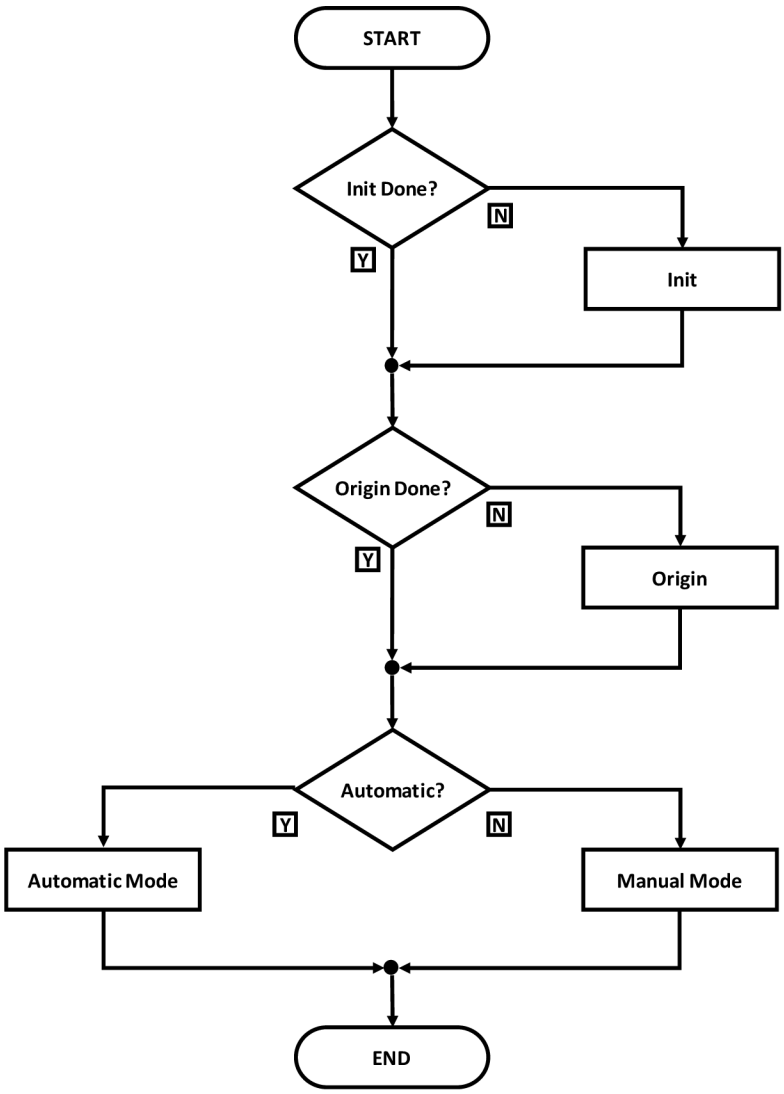
2.3 Modbus TCP

Modbus je otevřený komunikační protokol vytvořen firmou Modicon. Přenos dat je uskutečněn pomocí Ethernetu TCP/IP pracující v Master-Slave režimu. Na jedno master zařízení může připadat více než 1 slave zařízení. Zařízení slave nezahajují komunikaci. Odpovídají pouze, pokud dostanou dotaz od master zařízení. [13]

3. ANALÝZA EXISTUJÍCÍCH PROGRAMŮ

Před začátkem vývoje frameworku byla provedena analýza programů používaných ve firmě. Všechny programy byly napsány v programovacích jazycích ST nebo LD ve vývojovém prostředí Control FPWIN Pro 7 vyvíjené firmou Panasonic Electric Works Europe AG. Výrobce uvádí, že vývojové prostředí ctí normu IEC 61131-3. [14]

Z existujících programů byla zjištěna základní struktura programů (viz obrázek 3.1) a využívané prvky, které jsou okomentovány v následujících kapitolách. V ukázkách kódů budou původní komentáře označeny znaky „(* *)“, vlastní komentáře znaky „//“. Mohou se v nich vyskytovat také různé funkce či FB definované normou IEC 61131-3 nebo výrobcem.



Obrázek 3.1: Struktura programu

3.1 Inicializace

Programy obsahují inicializaci sloužící pro počáteční nastavení proměnných používaných v programu. Provádí se vždy při prvním spuštění zařízení. Po inicializaci se provádí vždy Origin.

```
IF bStart THEN (* Start a program *)
      ; // Program pracuje
ELSE (* bStart *)
      // Zkráceno. Když je program vypnutý, tak proběhne inicializace
      iError := 0; // Reset chyby
      bAuto := 0; // Přepnout do manuálního módu
      bOrigDN := 0; // Reset flagu pro provedení Origin
      uiState := 0; // Reset KSA
END_IF; (* bStart *)
```

3.2 Origin a časový limit

Origin se v programech označuje sekvence příkazů, po jejímž provedení se zařízení uvede do výchozího stavu. Po úspěšném dokončení Origin je přesně definované, v jakém stavu se zařízení nachází.

```
TON1(IN := T1EN, PT := T1TIME, Q => T1DN, ET => T1CTIME); (* Timeout timer *)
T1TIME := T#10s; (* Timeout time *)

IF iError <> 0 AND bOrigDN = False THEN
(* In case of Error and Origin not DN *)
      T1EN := 0;
END_IF;

IF NOT T1DN THEN (* If timeout timer has not timed out *)
      iError := 0; (* No error *)
      T1EN := 1; (* Activate timeout timer *)

      IF xMoveOP AND NOT xMoveRB THEN
      (* If cylinder is at operator's side, set bOriginDN to TRUE *)
            bOrigDN := 1;
            T1EN := 1; (* If Origin was Ok, Turn Off timer *)
      ELSE (* xMoveOP AND NOT xMoveRB *)
      (* If cylinder is at robot's side, move it *)
            yMoveOP := 1;
            yMoveRB := 0;
      END_IF; (* xMoveOP AND NOT xMoverB *)
ELSE (* NOT T1DN *)
      (* Timer timed out, origin not done -> error *)
      iError := 2;
END_IF; (* NOT T1DN *)
```

Origin je v ukázce implementován tak, že se nastaví písty pneumatického válce do určité polohy. Po změně výstupů PLC se v programu pomocí senzorů kontroluje, zdali se zařízení skutečně dostalo do požadovaného stavu. Při poruše senzorů či akčních členů by

program stále dokola neúspěšně čekal na splnění dané kontrolní podmínky. Zařízení by ale celou dobu bylo ve stavu nečinnosti. Navíc by obsluha zařízení nebyla informována o případné poruše.

Právě proto programátor do programu vhodně přidal časový limit pro kontrolu vstupních hodnot. Po jeho vypršení zařízení signalizuje nahlásí chybu s určitým kódem tak, aby se v případě poruchy minimalizovala doba odstávky zařízení.

3.3 Resetování

V programech se vyskytují 2 typy resetů – Retry a Reset.

Při spuštění Retry se vynulují proměnné obsahující informace o tom, že se zařízení nachází v chybovém stavu. Zařízení se tak pokusí znovu provést příkaz, ve kterém nastala chyba.

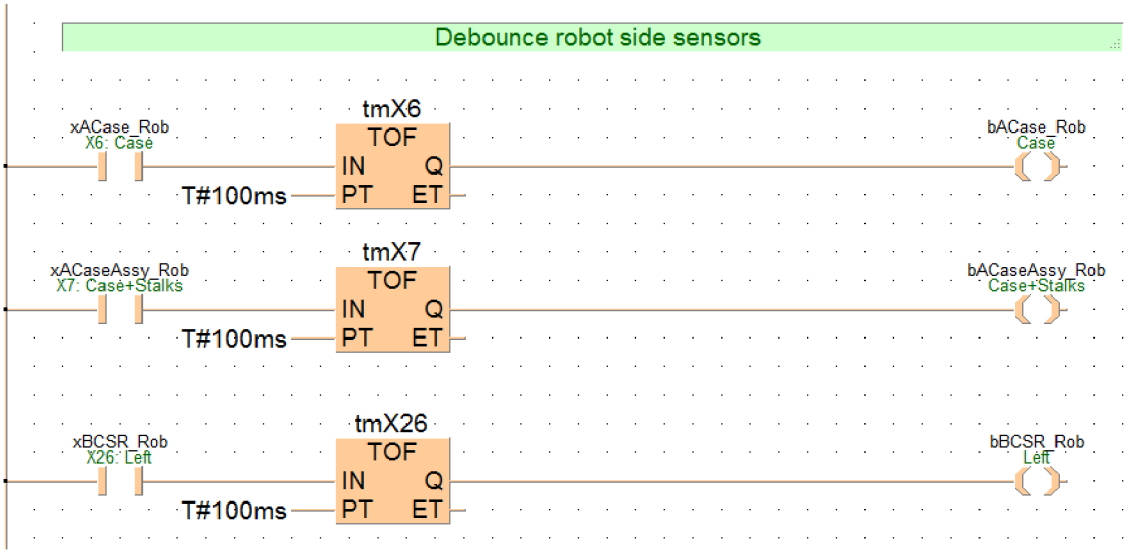
Reset slouží k opětovné inicializaci programu, takže se vynulují všechny proměnné. Zařízení se následně pokusí dostat do počátečního stavu Origin a celý proces musí začít od znovu.

```
IF DF(bReset) THEN
	bRetry := TRUE;
	bOriginDone := FALSE; // Po resetu proběhne znovu Origin
	uiRunningJob := 0; // Resetování procesu
	bReset := FALSE;
	// Zkráceno.
END_IF;
IF DF(bRetry) THEN
	bJigWarning := FALSE; // Reset varování
	iErrorCode := 0; // Reset chybového kódu
	bRetry := FALSE;
	// Zkráceno.
END_IF;
```

V ukázce se používá specifická funkce vývojového prostředí `DF` sloužící pro detekci náběžné hrany logické proměnné typu `BOOL`. Specifické funkce by se měly používat jen tehdy, pokud nelze program realizovat bez jejich použití, jelikož je pravděpodobně u jiných výrobců nenajdeme, v horším případě mohou mít jinou implementaci a kód se tak bude chovat nepředvídatelně. V tomto případě lze funkci `DF` nahradit FB s názvem Rising edge detector (`R_TRIG`) ze standardních FB normy IEC 61131-3.

3.4 Ošetření zákmitů

Vlivem elektrického rušení, či instalací levných, ale méně kvalitních senzorů a tlačítek se mohou na vstupech PLC objevit zákmity. Ošetření zákmitů pomocí softwaru nemusí být vhodné pro použití ve všech aplikacích, neboť dochází ke snížení odezvy zařízení na změnu vstupních hodnot. Zákmity lze ošetřit také hardwarově, například zakoupením kvalitnějších, ale dražších senzorů, instalací vstupní karty obsahující obvody pro filtrování signálů nebo připojením RC článku s vhodnou časovou konstantou na vstup PLC.



Obrázek 3.2: Ukázka části programu v jazyku LD

Ošetření zákmitů se v ukázce na obrázku 3.2 realizuje pomocí časovačů se zpožděným vypnutím (`TOF`) z knihovny standardních FB normy IEC 61131-3. Hodnota z výstupu `Q` příslušného časovače se zapíše do odpovídající proměnné na pravé straně diagramu. Použití časovačů zamezuje příliš časté změně proměnných na pravé straně, do kterých se zapisují ošetřené hodnoty ze senzorů. Doba, po kterou může proměnná na pravé straně ukazovat hodnotu `TRUE` není menší než nastavený časový limit 100 ms.

Ošetření zákmitů pomocí softwaru může být pro danou aplikaci efektivní, poněvadž zákmity vstupních hodnot mohou vést k tomu, že se zařízení dostane do stavu, v němž se ve skutečnosti nemá nacházet. Zařízení se tak může dostat do chybového stavu.

Implementace by však mohla být provedena jinak, protože se v ukázce pouze proměnné na pravé straně přidrží po dobu `PT` na hodnotě `TRUE`, ale není jasné, kdy se ošetření zákmitů dokončilo a zdali se vůbec dokončilo. Může se stát, že senzor má ukazovat hodnotu `FALSE`. Na vstupu PLC se však objeví zákmit, takže se na vstupu PLC objeví na malou chvíli hodnota `TRUE`. Program nám navíc tento zákmit přidrží. Pro danou aplikaci může být uvedená implementace vyhovující, ale není vhodná pro univerzální využití. Navíc by mohly mít časovače pro snadnější orientaci v kódu konkrétní názvy.

Ošetření zákmitů by se dalo implementovat tak, že by se pomocí časovače se zpožděným zapnutím (TON) kontrolovalo, jestli se daná proměnná s požadovanou hodnotou po uplynutí časového úseku PT nezměnila. Pokud by se změnila, tak by se časovač vynuloval a kontrola by proběhla znovu. Pokud by se hodnota za danou dobu nezměnila, tak by se nastavila proměnná signalizující úspěšné dokončení.

3.5 Konečný stavový automat

Zařízení pracují buď v automatickém, nebo v manuálním módu, který je realizován pomocí konečného stavového automatu (KSA) a tvoří hlavní část programu. KSA se skládá ze stavů a přechodů mezi stavy.

```
tonOriginStart(IN := iJState = 20, PT := T#5s);
tonMoveUpTimeOut(IN :=  iJState = 46, PT := T#5s);
CASE iJState OF
// Upraveno. Jsou uvedeny některé stavy KSA.
20: (* Set origin *)
   bCyl_WiperUp := FALSE; (* Move down *)
   IF (NOT xSns_Cyl_Wiper_Up) THEN
      iJState := 46;
   ELSIF (tonOriginStart.Q) THEN
      IF (xSns_Cyl_Wiper_Up) THEN iJ_ErrorCode := 31; END_IF;
   END_IF;
46: (* Press stamp on PWB *)
   bCyl_WiperUp := FALSE; (* Move down *)
   IF (NOT xSns_Cyl_Wiper_Up) THEN
      iJState := 48;
   ELSIF (tonMoveUpTimeOut.Q) THEN
      IF (xSns_Cyl_Wiper_Up) THEN iJ_ErrorCode := 31; END_IF;
   END_IF;
END_CASE;
```

Ukázka pouze znázorňuje, jak se dá realizovat KSA v jazyku ST. Oproti minulé ukázce z kapitoly 3.2 se v této ukázce lépe generují chybová hlášení, která vždy přesně informují, proč nastala chyba. Obsluze zařízení určitě více pomůže hlášení s konkrétní informací jako je třeba: „Nastala chyba, protože sensor xSns_Cyl_Wiper_Up ukazuje hodnotu FALSE, ale měl by ukazovat TRUE“ než „Nastala chyba při provádění Origin“.

V jednotlivých stavech se ve většině případů využívá mnoho operací jako je ovládání výstupů, kontrola vstupů s časovým limitem, s ošetřením zákmitů a s tím související generování chybových kódů. Je výhodnější, když se kód rozdělí na menší části, neboť se lépe opakovatelně používají pro podobná zařízení a lépe se testují v simulátoru.

V případě používání stejných operací v mezi různými stavy, jako je například ovládání stejného pneumatického válce včetně provádění kontrol a generování chybových kódu, není vhodné tyto operace mezi stavy kopírovat. Pokud je potřebujeme změnit je nutné procházet větší část kódu, aby bylo zajištěno, že změny byly provedeny ve všech kopiích.

Tento problém bychom mohli vyřešit rozdělením kódu do vícero POU, z nichž by každá zapouzdřovala odlišnou funkcionalitu a byly by lépe přenositelnější. Také by se usnadnilo testování, protože můžeme jednotlivé části testovat samostatně.

4. NÁVRH STRUKTURY HLAVNÍHO PROGRAMU

Po analýze programů bude dalším krokem ve vývoji frameworku návrh základní struktury programu pro jednoúčelové zařízení řízené PLC používané v assembly procesech.

Jednoúčelová zařízení se používají ve velkosériové výrobě a slouží pro opakované vykonávání jedné činnosti jako je například kontrola kvality výrobku, obrábění, sváření, slisování, montáž či různé manipulační úkony s výrobkem. Pracují buď automaticky, poloautomaticky, nebo manuálně.

Cílem assembly procesu je vytvořit výrobek, který vznikne složením různých dílů přesně danou sekvencí pohybů.

4.1 Vývojové prostředí

Před samotným návrhem byl proveden průzkum vývojových prostředí od různých výrobců PLC (viz tabulka 4.1). Průzkum ukázal, že některá vývojová prostředí, která podle výrobce ctí normu IEC 61131-3, neobsahují rozšíření pro OOP, takže nepodporují využívání tříd, metod, rozhraní, dědičnosti a polymorfismu. Jelikož se od frameworku požaduje, aby byl dobře přenositelný mezi různými vývojovými prostředími, tak software nebude tyto prvky obsahovat. Z první kapitoly tak může být využito pouze zapouzdření a tvoření instancí.

Pro vývoj softwaru bylo vybráno vývojové prostředí CODESYS vyvíjené firmou CODESYS GmbH v souladu s normou IEC 61131-3. Prostředí není závislé na konkrétním výrobci PLC, neboť podporuje spoustu modelů PLC od různých výrobců. Pro potřeby testování vytvořených aplikací je možné spustit na počítači SoftPLC nainstalované společně s CODESYS. Bez zakoupené licence je však nutné po 2 hodinách systém obnovit. [15]

Tabulka 4.1: Průzkum vývojových prostředí pro programování PLC

Výrobce	Vývojové prostředí	Podpora IEC 61131-3	Rozšíření pro OOP
Siemens [16]	TIA Portal	ANO	NE
Rockwell Automation [17]	Studio 5000	ANO	NE
Mitsubishi Electric [18]	GX Works3	ANO	NE
Omron [19]	Sysmac Studio	ANO	NE
Keyence [20]	KV STUDIO	NE	NE
ABB [21]	Automation Builder	ANO	ANO
Panasonic [14]	Control FPWIN PRO 7	ANO	NE
Beckhoff [22]	TwinCAT 3	ANO	ANO
CODESYS GmbH [18]	CODESYS	ANO	ANO

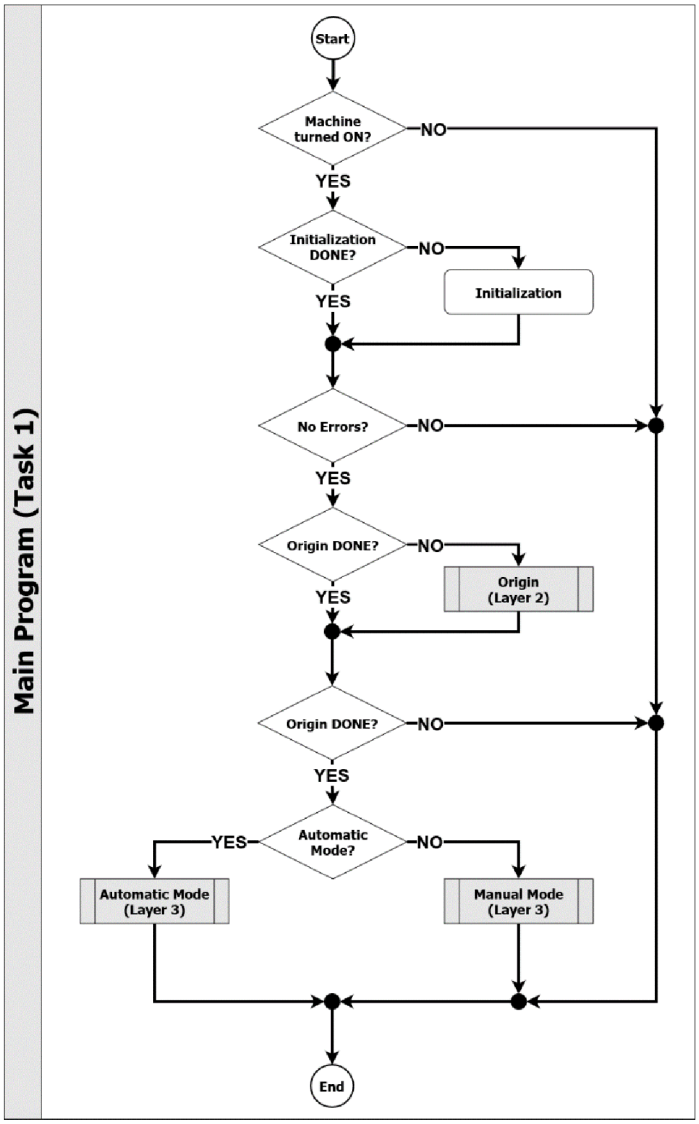
4.2 Struktura hlavního programu

Od programu se požaduje, aby byl dobře přenositelný mezi různými výrobci a vývojovými prostředími podporující normu IEC 61131-3 a aby se skládal z vícero menších částí, což umožní snadnější testování v simulátoru a rychlejší provádění změn.

Na základě analýzy a požadavků byla navržena architektura hlavního programu obsahující 3 vrstvy, z nichž každá zapouzdřuje odlišnou funkcionalitu (viz obrázek 4.1).

V hlavním programu bude probíhat inicializace, Origin, kontrola, jestli může zařízení bezpečně pracovat, tzn. že se nevyskytla žádná chyba, a výběr mezi manuálním a automatickým módem.

Nevýhodou zvolené softwarové architektury je, že není zrovna nejefektivnější z hlediska paměťové a časové náročnosti, protože program prochází všemi vrstvami. V případě jednoúčelových zařízení by se však uvedené nedostatky neměly projevit. [23]



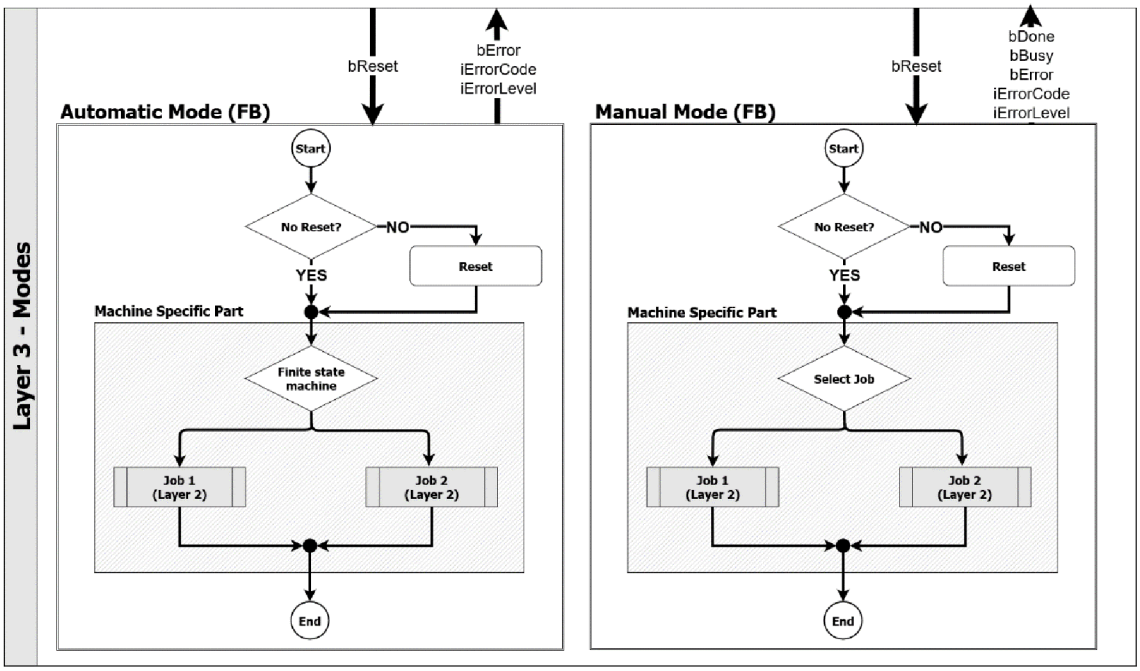
Obrázek 4.1: Blokový diagram hlavního programu

4.2.1 Manuální a automatický mód

Manuální mód (viz obrázek 4.2) bude obsahovat příkazy pro manuální obsluhu zařízení zadávaných přes operátorský panel. Jelikož uživatel může vybrat jakýkoliv implementovaný příkaz, tak by se v programu měla vyskytovat kontrola, jestli může být daný příkaz přijat, tak aby se zabránilo poškození zařízení či výrobku.

Automatický mód bude realizován pomocí KSA. V každém stavu se bude volat vybraný úkol. Když se úkol nepovede provést, tak se nahlásí chyba. Po resetu chyby se pokusí znovu provést poslední metoda.

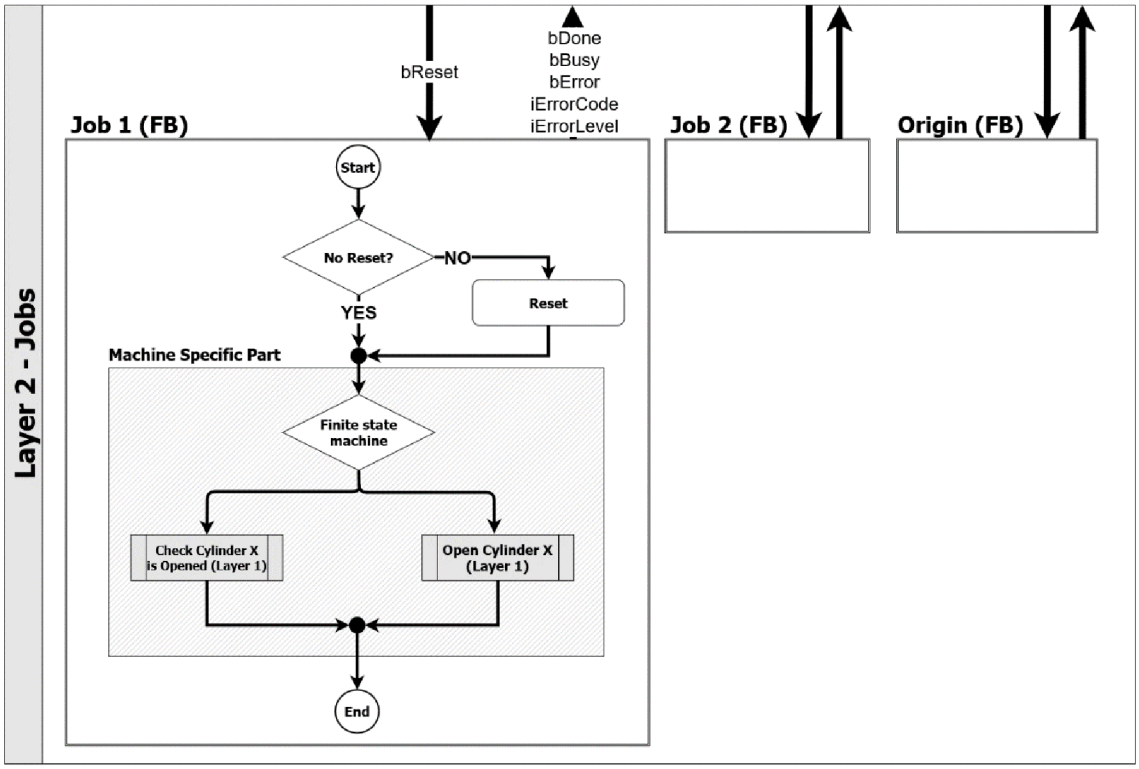
KSA bude obsahovat stav, v němž bude zařízení čekat na další příkaz. Příkazem může být, že má zařízení buď pokračovat dál ve vykonávání automatického módu, nebo že se má přepnout do manuálního módu. Implementace módů se bude nacházet ve FB.



Obrázek 4.2: Blokový diagram vrstvy 3 - Módy

4.2.2 Úkoly

Úkoly (viz obrázek 4.3) budou sloužit k rozdělení KSA na vícero menších částí. V případě pneumatického lisu mohou být úkoly například proveď Origin, počkej na vložení výrobku do přepravníku a přesuň ho pod pneumatický lis, slisuj výrobek, dovez zpátky výrobek a počkej na jeho vyjmutí. Úkoly se vykonávají pomocí postupného volání metod. Každý úkol se bude tvořit podle šablony a nacházet se ve zvláštním FB.



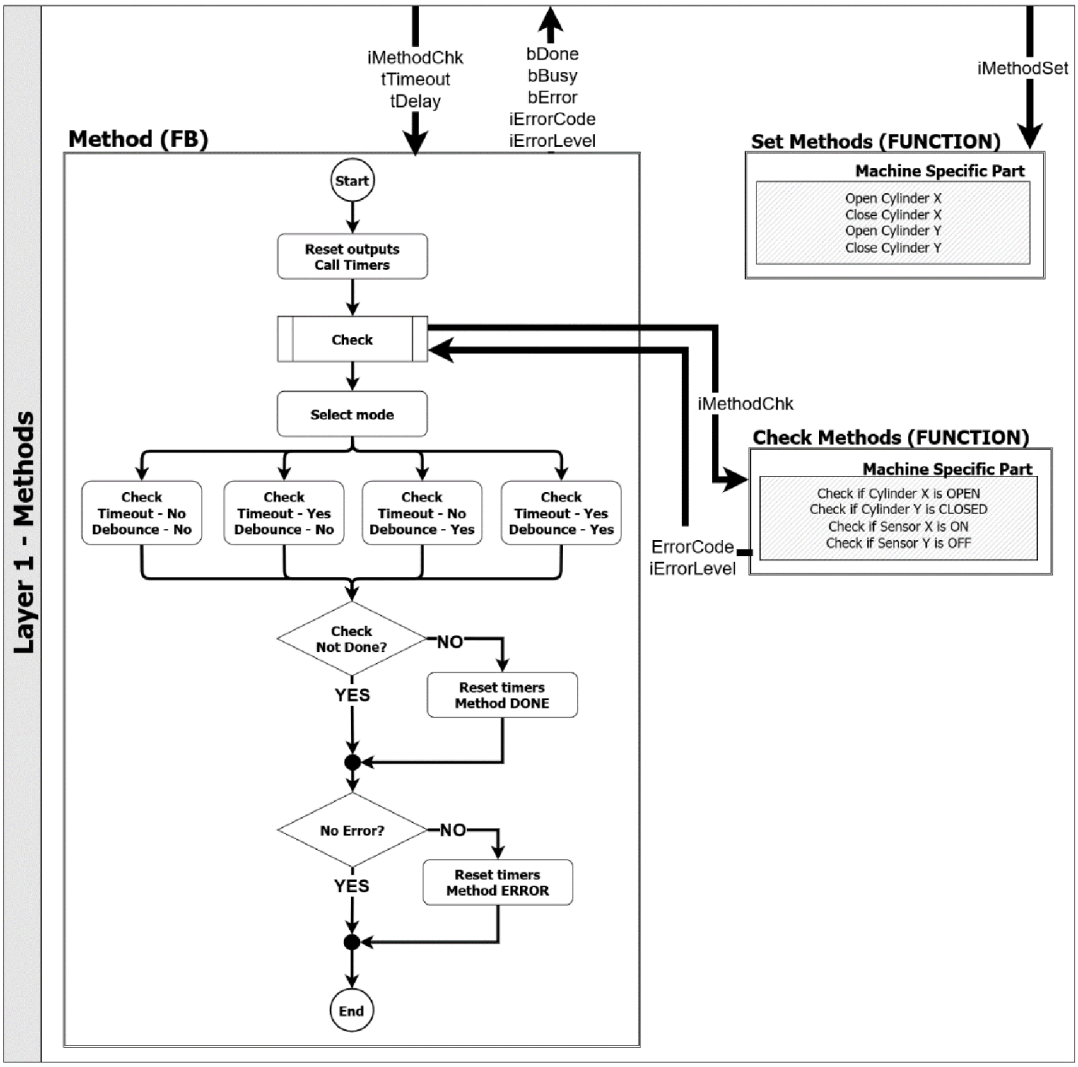
Obrázek 4.3: Blokový diagram vrstvy 2 - Úkoly

4.2.3 Metody

Pro snadnější kontrolu a nastavování proměnných byly vytvořeny 2 uživatelské funkce, ze kterých jsou tyto instrukce volány (viz obrázek 4.4). Změny tak stačí provádět pouze v těchto funkcích.

V první funkci se budou definovat metody[1] pro nastavování proměnných (například: vysunou/zasunout píst válce). Druhá funkce bude obsahovat metody pro kontrolu proměnných (například: zkontroluj, zdali se píst válce vysunul/zasunul). Při neúspěchu vrátí funkce chybu obsahující informaci, proč kontrola neproběhla úspěšně.

U kontroly je však potřeba používat časovače pro časový limit a pro ošetření zákmitů, což funkce neumožňuje, protože nemá paměť, proto bude funkce volána z předdefinovaného FB. Pro univerzální použití bude FB umožňovat výběr, které časovače mají být spuštěny.



Obrázek 4.4: Blokový diagram vrstvy 1 - Metody

[1] Pojmenování bylo stanoveno na základě funkcionality vrstvy. Nejedná se o metody definované normou IEC 61131-3.

5. IMPLEMENTACE A TOK HLAVNÍHO PROGRAMU

Po návrhu softwarové architektury byly jednotlivé POU programu doplněny o proměnné `bDone`, `bBusy`, `bError`, `iErrorCode`, `iErrorLevel` potřebné pro komunikaci mezi jednotlivými vrstvami. Vývojové diagramy se nachází v příloze A.
Pro pojmenování proměnných platí následující pravidla:

- Slova nejsou oddělena žádnými znaky a začínají velkým písmenem
- Proměnné typu `BOOL` mají před názvem předponu „`b`“
- Proměnné typu `INT` mají před názvem předponu „`i`“
- Proměnné typu `TIME` mají před názvem předponu „`t`“
- Proměnné pro vstupy PLC mají před názvem předponu „`x`“
- Proměnné pro výstupy PLC mají před názvem předponu „y“
- Globální proměnné obsahují předponu „`G_`“

V programu se vyskytují 2 úrovně chyb:

- Málo závažná chyba – Vznikne například pokud dojde zařízení materiál. Po jeho doplnění a po resetu chyby (Retry) pokračuje zařízení dál v činnosti.
- Závažná chyba – Vznikne například při detekci nesprávné hodnoty tlaku. Po opravě zařízení nemůže pokračovat dál v činnosti, a proto se musí vyresetovat všechny proměnné (Reset) a začít znovu.

5.1 Vrstva 1 – Metody

K vrstvě se přistupuje pomocí předdefinovaného FB s názvem `METHOD` a slouží ke kontrole proměnných na základě vybrané metody. Seznam a popis vstupů a výstupů jsou uvedeny v tabulce 5.1. Před tím, než může uživatel předdefinovaný FB použít, potřebuje vytvořit metody podle následující kapitoly.

Tabulka 5.1: Vstupy a výstupy METHOD (FB)

Typ	Název	Datový typ	Komentář
Vstup	iMethodChk	INT	Číslo vybrané metody
Vstup	tTimeout	TIME	T#0s: Bez časového limitu
Vstup	tDelay	TIME	T#0s: Bez ošetření zákmitů
Výstup	bDone	BOOL	TRUE: Metoda úspěšně dokončena
Výstup	bBusy	BOOL	TRUE: Metoda právě probíhá
Výstup	bError	BOOL	TRUE: Nastala chyba
Výstup	iErrorCode	INT	0: Žádná chyba
Výstup	iErrorLevel	INT	0: Žádná chyba 1: Málo závažná chyba 2: Závažná chyba

Pro zajištění univerzálního použití bude FB pracovat v různých režimech. Výběr režimu závisí na hodnotách vstupních proměnných `tTimeout` a `tDelay` (viz tabulka 5.2 a tabulka 5.3). První proměnná určuje časový limit. Druhá proměnná určuje časovou konstantu pro ošetření zákmitů.

Tabulka 5.2: Výběr režimu METHOD (FB)

	tDelay = 0 s	**tDelay > 0 s**
tTimeout = 0 s	Režim 1	Režim 2
tTimeout > 0 s	Režim 3	Režim 4

Tabulka 5.3: Funkce režimů METHOD (FB)

Režim	**Časový limit**	**Ošetření zákmitů**
1	NE	NE
2	NE	ANO
3	ANO	NE
4	ANO	ANO

5.1.1 Metody pro nastavování proměnných

Funkce `METHOD_SET` je uživatelská. Seznam a popis vstupů a výstupů jsou uvedeny v tabulce 5.4 a slouží k nastavování proměnných na základě vybrané metody. Pokud není zadaná metoda definována, tak funkce nic nenastaví. Funkce nemá návratovou hodnotu.

Tabulka 5.4: Vstupy a výstupy METHOD_SET (FUN)

Typ	**Název**	**Datový typ**	**Komentář**
Vstup	iMethodSet	INT	Číslo vybrané metody

Pro lepší pochopení, jak definovat tuto funkci, byla na ukázku vytvořena metoda s číslem 1 pro vysunutí pístu dvojčinného pneumatického válce.

```
// Příklad, jak se vytváří metoda pro vysunutí pístu válce
IF iMethodSet = 1 THEN
   GVL.yCylOff := FALSE;
   GVL.yCylOn := TRUE;
END_IF;
```

5.1.2 Metody pro kontrolu proměnných

Funkce `METHOD_CHK` se skládá z uživatelské a předdefinované části. Seznam a popis vstupů a výstupů jsou uvedeny v tabulce 5.5 a slouží ke kontrole proměnných na základě vybrané metody. Pokud byly proměnné úspěšně zkontrolovány, tak se vrátí `iErrorCode` s hodnotou 0. V případě, že kontrola neproběhla úspěšně, tak se vrátí chyba s příslušným kódem a závažností. Výchozí hodnota `iErrorCode` je nastavena na 999 (nedefinovaná chyba) a hodnota `iErrorLevel` na 2 (závažná chyba). Pokud není zadaná metoda definována, tak funkce vždy vrátí, že kontrola proběhla úspěšně.

Tabulka 5.5: Vstupy a výstupy METHOD_CHK (FUN)

Typ	Název	Datový typ	Komentář
Vstup	iMethodChk	INT	Číslo vybrané metody
Výstup	iErrorCode	INT	0: Žádná chyba
Výstup	iErrorLevel	INT	0: Žádná chyba 1: Málo závažná chyba 2: Závažná chyba

K metodě pro vysunutí pístu válce byla vytvořena metoda, která zkontroluje, zdali se píst válce vysunul. Kontrolu stavu pístu válce zajišťují 2 koncové snímače, z nichž první signalizuje zasunutý píst, druhý vysunutý píst.

V případě neúspěchu kontroly hodnot se vyžaduje, aby zařízení vracelo chybu s kódem obsahující informaci o přesném místě vzniku chyby. Výpočet kódu je vyřešen tak, že se kontrolované hodnoty (maximum je 15) postupně přiřadí do proměnných. Každá proměnná odpovídá 1 bitu typu INT (`bIn0_OK` odpovídá bitu 0, `bIn1_OK` bitu 1, ...).

Počet možných chyb metody se dá vypočítat pomocí vztahu:

$$x = 2^N - 1\,, \tag{5.1}$$

kde x představuje počet možných chyb a N je počet kontrolovaných proměnných.

Pomocí proměnné `iErrorOffset` se nastaví číslo, od jakého se chyba pro danou metodu počítá. V případě kontroly 2 vstupů mohou na základě rovnice 5.1 nastat 3 různé chyby (chyby 10 – 12). Například chyba s kódem 12 říká, že oba koncové senzory ukazují špatnou hodnotu. V případě potřeby lze nastavit závažnost chyby pomocí proměnné `iErrorLevel`.

```
// Příklad, jak se vytváří metoda pro kontrolu vysunutí pístu válce
IF iMethodChk = 1 THEN
   bIn0_OK := NOT GVL.xCylClosed; // Musí mít hodnotu FALSE pro splnění
   bIn1_OK := GVL.xCylOpen; // Musí mít hodnotu TRUE pro splnění
   iErrorOffset := 10; // Při neuvedení a nesplnění kontroly má chyba kód 999
   iErrolLevel := 2; // Při neuvedení vrací funkce implicitně úroveň chyby 2
END_IF;
```

5.2 Vrstva 2 – Úkoly

Úkoly se tvoří podle šablony `JOB` a pro každý úkol se vytváří zvláštní FB. Pro zajištění univerzálního použití budou mít FB stejnou strukturu a stejné vstupy a výstupy popsané v tabulce 5.6. Úkoly jsou vykonávány voláním jednotlivých metod, a proto je před použitím úkolů nutné, aby uživatel nejdříve vytvořil jednotlivé metody podle předchozí kapitoly.

Tabulka 5.6: Vstupy a výstupy JOB (FB)

Typ	Název	Datový typ	Komentář
Vstup	bReset	BOOL	TRUE: Vyresetuj úkol
Výstup	bDone	BOOL	TRUE: Úkol úspěšně dokončen
Výstup	bBusy	BOOL	TRUE: Úkol právě probíhá
Výstup	bError	BOOL	TRUE: Nastala chyba
Výstup	iErrorCode	INT	0: Žádná chyba
Výstup	iErrorLevel	INT	0: Žádná chyba 1: Málo závažná chyba 2: Závažná chyba

Na ukázku byl vytvořen úkol s číslem 1, v němž se nejdříve vysune píst válce (stav 0), pak se píst válce zasune a úkol se při úspěšné kontrole dokončí (stav 1). Při tvorbě úkolu se použijí již dříve vytvořené metody, k nimž se podobným způsobem vytvoří metody pro zasunutí pístu válce. Vytvořené metody pro nastavování zavoláme pomocí `METHOD_SET` a metody pro kontrolu zavoláme `METHOD_CHK` pomocí vytvořené instance `CallMethod` s nastaveným časovým limitem 10 s a s ošetřením zákmitů s časovou konstantou 100 ms. V případě potřeby je možné časové údaje pro metody lehce upravit a není nutné, aby měly ve všech stavech stejnou hodnotu.

Při vypršení časového limitu metoda signalizuje její neúspěšné provedení nastavením proměnné `bError`. Po dokončení metody se signalizuje úspěch nastavením proměnné `CallMethod.bDone` a dojde k přechodu KSA do dalšího stavu. V případě stavu 1 je tato proměnná předána do výstupní proměnné úkolu `bDone` a úkol po splnění metody signalizuje dokončení. Proměnná `bReset` bude nastavována vyšší vrstvou a slouží k resetu KSA do stavu 0.

```
// Příklad, jak se vytváří úkol 1

// Nastavení výchozích hodnot časovačů
CallMethod.tTimeout := T#10s;
CallMethod.tDelay := T#100ms;

IF iState = 0 THEN
   // Vysuň píst válce
   METHOD_SET(iMethodSet := 1);

   // Zkontroluj, zdali se píst vysunul
   CallMethod(iMethodChk := 1, bError => bError);

   // Při úspěšné kontrole, přejdi do dalšího stavu
   IF CallMethod.bDone THEN iState := iState + 1; END_IF;
ELSIF iState = 1 THEN
   // Zasuň píst válce
   METHOD_SET(iMethodSet := 2);

   // Zkontroluj, zdali se píst zasunul
   CallMethod(iMethodChk := 2);

   // Při úspěšné kontrole byl úkol dokončen
   // Nastav bDone a vyresetuj KSA
   IF CallMethod.bDone THEN bDone := True; iState := 0; END_IF;
END_IF;
```

5.3 Vrstva 3 – Manuální a automatický mód

Vrstva 3 slouží pro výběr a vykonávání vytvořených úkolů.

5.3.1 Manuální mód

K manuálnímu módu se přistupuje pomocí FB s názvem `MANUAL`. Seznam a popis vstupů a výstupů jsou uvedeny v tabulce 5.7.

Tabulka 5.7: Vstupy a výstupy MANUAL (FB)

Typ	Název	Datový typ	Komentář
Vstup	bReset	BOOL	TRUE: Vyresetuj manuální mód
Výstup	bDone	BOOL	TRUE: Úkol úspěšně dokončen
Výstup	bBusy	BOOL	TRUE: Úkol právě probíhá
Výstup	bError	BOOL	TRUE: Nastala chyba
Výstup	iErrorCode	INT	0: Žádná chyba
Výstup	iErrorLevel	INT	0: Žádná chyba 1: Málo závažná chyba 2: Závažná chyba

Ve FB v manuálním módu si uživatel pouze vytvoří, jaký úkol se má pro dané číslo příkazu obsažené v proměnné `G_iJobManual` vykonat. Po dokončení je proměnná s příkazem ve vyšší vrstvě vynulována a zařízení čeká na další příkaz. Pro ukázku se přiřadí úkol vytvořený v přechozí ukázce k příkazu s číslem 1. Úkoly se resetují vždy při jejich prvním spuštění nebo po Resetu.

```
// Příklad, jak se vytváří manuální mód

IF G_iJobManual = 1 THEN
   YOUR_JOB(bReset := bDone, bDone => bDone, bBusy => bBusy,
   bError => bError, iErrorCode => iErrorCode, iErrorLevel => iErrorLevel);
END_IF;
```

5.3.2 Automatický mód

K automatickému módu se přistupuje pomocí FB s názvem `AUTO`. Seznam a popis vstupů a výstupů jsou uvedeny v tabulce 5.8.

Automatický mód oproti ostatním FB neobsahuje výstupní proměnné `bDone` a `bBusy`, jelikož se automatický mód vykonává neustále dokola, takže nikdy nedojde k jeho dokončení, a proto jsou uvedené proměnné nepotřebné.

V automatickém módu uživatel vytvoří KSA pro vykonávání úkolů. V případě, že má zařízení signalizovat, že čeká na další příkazy, tak se nastaví proměnná `G_bReady`. Potom se v hlavním programu čeká na její vynulování, které bude prováděné v komunikačním programu, nebo může být přepnuto do manuálního módu.

Tabulka 5.8: Vstupy a výstupy AUTO (FB)

Typ	Název	Datový typ	Komentář
Vstup	bReset	BOOL	TRUE: Vyresetuj automatický mód
Výstup	bError	BOOL	TRUE: Nastala chyba
Výstup	iErrorCode	INT	0: Žádná chyba
Výstup	iErrorLevel	INT	0: Žádná chyba 1: Málo závažná chyba 2: Závažná chyba

```
// Příklad, jak vytvořit automatický mód

IF iState = 0 THEN
   // Zavolej vytvořený úkol
   YOUR_JOB(bReset := bDone, bDone => bDone
   bError => bError, iErrorCode => iErrorCode, iErrorLevel => iErrorLevel);

   // Úkol byl dokončen => přejdi do dalšího stavu
   IF YOUR_JOB.bDone THEN
       iState := 50;
       // Nastav G_bready pro informování master, že je zařízení připraveno
       gvl.G_bReady := True;
   END_IF;
END_IF;

IF iState = 50 THEN
   // Pro pokračování v automatickém módu počkej na příkaz od master
   IF gvl.G_bReady = False THEN iState := 0; END_IF;
END_IF;
```

5.4 Globální proměnné hlavního programu

Proměnné hlavního programu, které se budou zpracovávat v komunikačním programu a budou sloužit pro informování master PLC o aktuálním stavu zařízení, se nachází v tabulce 5.9 a jsou definovány jako globální. Hlavní program bude předdefinovaný.

Tabulka 5.9: Globální proměnné hlavního programu

Název	Datový typ	Komentář
G_bStart	BOOL	TRUE: Zařízení je spuštěno
G_bInitDone	BOOL	TRUE: Inicializace proběhla
G_bOriginDone	BOOL	TRUE: Origin byl proveden
G_bAuto	BOOL	TRUE: Je vybrán automatický mód FALSE: Je vybrán manuální mód
G_bReady	BOOL	TRUE: Zařízení čeká na příkaz od master
G_iJobManual	INT	Nastavené číslo úkolu pro manuální mód
G_bError	BOOL	TRUE: Nastala chyba
G_iErrorCode	INT	0: Žádná chyba
G_iErrorLevel	INT	0: Žádná chyba 1: Málo závažná chyba 2: Závažná chyba

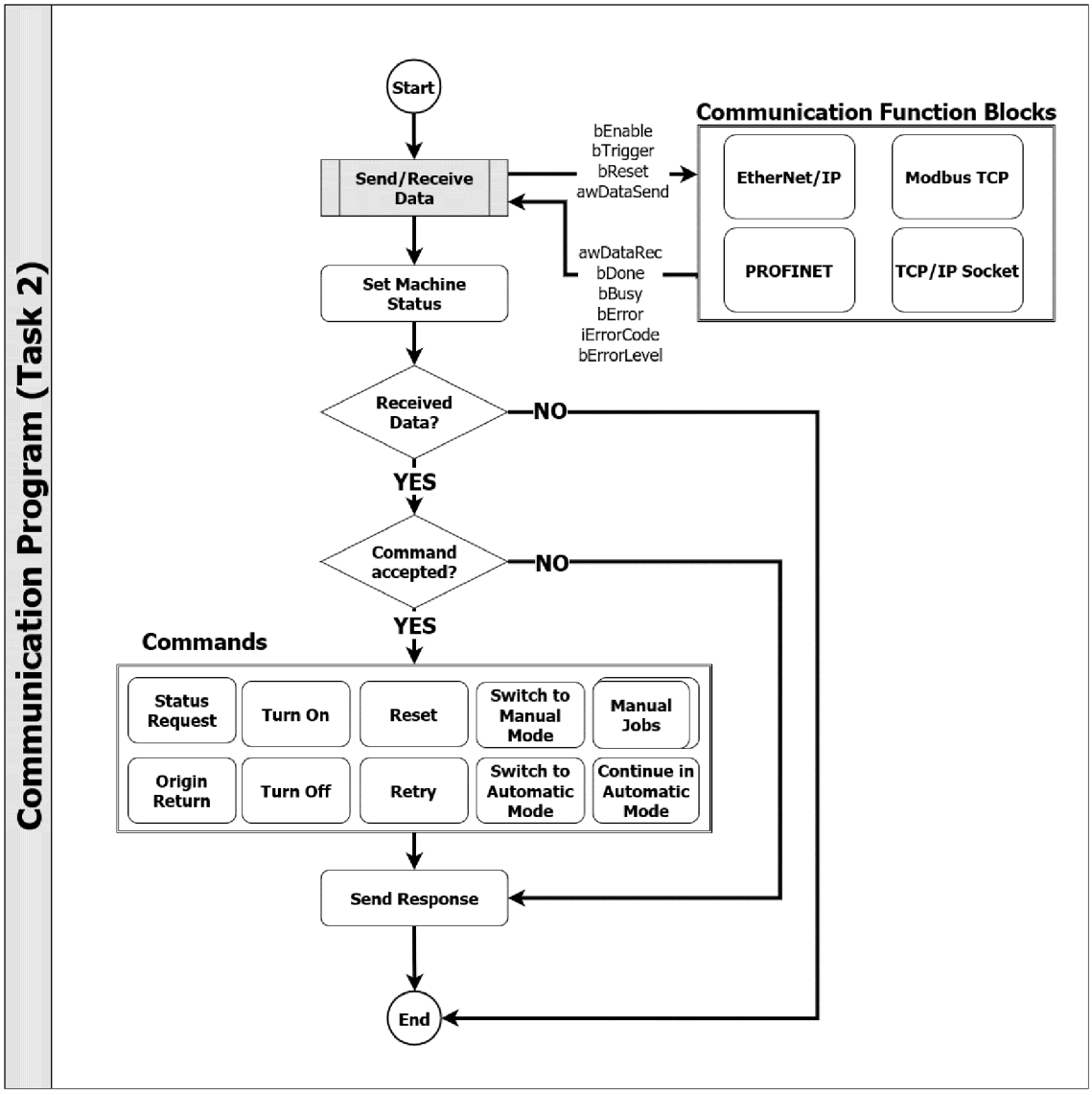
6. KOMUNIKACE

Jednoúčelová zařízení nemusí vždy pracovat samostatně. Ve firmě se často integrují do automatizovaných výrobních systémů, ve kterých jich může být větší počet. Zařízení budou pracovat v Master-Slave režimu (viz obrázek 6.1), což byl další z požadavků na framework. Role „slave“ je přidělena jednoúčelovým zařízením, protože pouze přijímají příkazy, po jejichž obdržení odesílají odpověď, od nadřazeného zařízení označovaného jako „master“. Master může být PLC, počítač, robot.

Obrázek 6.1: Master-Slave režim

6.1 Program pro komunikaci

Po hlavním programu byl vytvořen program pro komunikaci s master PLC pracující paralelně s hlavním programem. Od programu se požaduje, aby pracoval nezávisle na použitém komunikačním protokolu pro možnost univerzálního využití. V programu jsou implementovány příkazy pro ovládání zařízení (viz obrázek 6.2). Příjem a odeslání dat zajišťuje FB tvořící nižší vrstvu komunikačního programu.

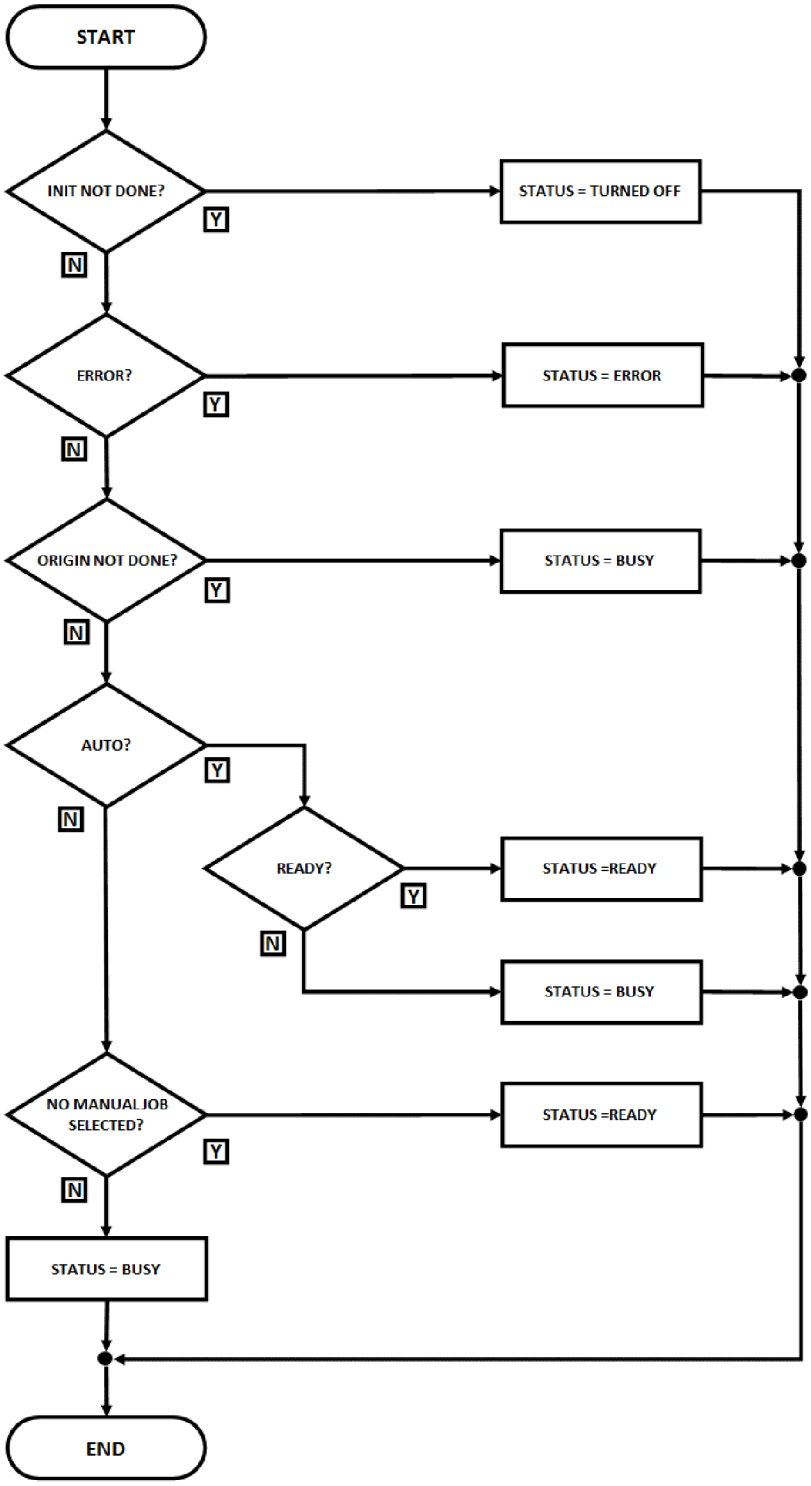


Obrázek 6.2: Blokový diagram komunikačního programu

6.1.1 Aktuální stav zařízení

Proměnná `iStatus` (viz obrázek 6.3) slouží pro informování master PLC o aktuálním stavu zařízení. V případě, že zařízení čeká na první spuštění, tak zařízení při obdržení příkazu „Zjistit status“ informuje, že je „Vypnuto (Turned off)“. Po obdržení příkazu po zapnutí se provede inicializace a Origin. Po jeho dokončení zařízení při dalším obdržení tohoto příkazu vrací „Připraveno (Ready)“. Nyní zařízení čeká na zadání manuálního

příkazu nebo na přepnutí do automatického módu. Pokud zařízení právě pracuje, pak `iStatus` obsahuje hodnotu reprezentující „Pracuje (Busy)“, při chybě pak „Chyba (Error)“.



Obrázek 6.3: Zjištění aktuálního stavu zařízení

6.1.2 Příkazy

Na základě aktuální stavu zařízení závisí, jestli může být právě obdržený příkaz přijat a vykonán, anebo jestli nemůže být přijat, protože to aktuální stav zařízení neumožňuje. Například pokud by zařízení právě bylo „Připraveno“, tak nemohou být přijaty příkazy pro resetování. Seznam a popis příkazů se nachází v tabulce 6.1.

Pro většinu příkazů je jasně definovaná odpověď závisející pouze na aktuálním stavu zařízení. Pro následují příkazy však musí být splněny další podmínky:

- „Pokračovat“ – Přijat pouze, když je zařízení „Připraveno“ a současně je přepnuto v automatickém módu.
- „Úkoly“ – Zařízení musí být přepnuto v manuálním módu. Pokud není daný „Úkol“ pro použité zařízení definován, tak se vrátí odpověď „Neznámý příkaz“.
- „Retry“ – Přijat pouze při vzniku málo závažné chyby.

Tabulka 6.1: Seznam příkazů

	Status			
Příkaz	**1: Vypnuto**	**2: Připraveno**	**3: Pracuje**	**4: Chyba**
(1) Zjistit status	1: Vypnuto	2: Připraveno Data1: 1- Automat Data1: 2- Manuál	3: Pracuje	4: Chyba Data1: Číslo Data2: Závažnost
(2) Auto. mód	2: Nepřijato	1: Přijato	2: Nepřijato	2: Nepřijato
(3) Manu. mód	2: Nepřijato	1: Přijato	2: Nepřijato	2: Nepřijato
(4) Origin	2: Nepřijato	1: Přijato	2: Nepřijato	2: Nepřijato
(5) Reset	2: Nepřijato	2: Nepřijato	2: Nepřijato	1: Přijato
(6) Retry	2: Nepřijato	2: Nepřijato	2: Nepřijato	1: Přijato 2: Nepřijato
(7) Zapnout	1: Přijato	2: Nepřijato	2: Nepřijato	2: Nepřijato
(8) Vypnout	2: Nepřijato	1: Přijato	2: Nepřijato	2: Nepřijato
(50) Pokračovat	2: Nepřijato	1: Přijato 2: Nepřijato	2: Nepřijato	2: Nepřijato
(101-199) Úkoly	2: Nepřijato	1: Přijato 2: Nepřijato 99: Neznámý	2: Nepřijato	2: Nepřijato
(???) Neznámý příkaz	99: Neznámý	99: Neznámý	99: Neznámý	99: Neznámý

Příkazy jsou posílány 3 proměnnými typu `WORD` ve tvaru <Command; Param1; Param2> . Param1 a Param2 jsou zatím nevyužity. Jsou připraveny pro budoucí použití.

Odpověď se odesílá 3 proměnnými typu `WORD` ve tvaru <Response; Data1; Data2>. Data1 a Data2 jsou použita pouze při odpovědi na „Zjistit status“, jinak neobsahují žádnou informaci.

Pro ostatní příkazy mohou podle tabulky 6.1. nastat tři případy s následujícími odpověďmi:

- Přijato – odpověď: <1; 0; 0>
- Nepřijato – odpověď: <2; 0; 0>
- Neznámý příkaz – odpověď: <99; 0; 0>

Odpověď „Neznámý příkaz“ by však neměla být nikdy odeslána. Znamená to, že přijatá data jsou buď poškozená, nebo se snaží master zavolat příkaz, který slave nezná.

6.1.3 Příjem a odesílání dat

Pro příjem a odesílání dat slouží FB tvořící nižší vrstvu komunikačního programu s tabulkou vstupů a výstupů uvedenými v tabulce 6.2. Implementace závisí na použitém komunikačním protokolu či modelu PLC, protože se v ní většinou používají specifické funkce pro dané PLC. Návrh tak není možné příliš dobře standardizovat. Výhodou tohoto řešení je, že při změně protokolu stačí pouze měnit FB se stejným rozhraním. V programu se tak nemusí již nic upravovat.

Tabulka 6.2: Vstupy a výstupy FB pro příjem a odesílání dat

Typ	Název	Datový typ	Komentář
Vstup	bEnable	BOOL	TRUE: Povolit komunikaci
Vstup	bTrigger	BOOL	TRUE: Pošli data
Vstup	bReset	BOOL	TRUE: Vyresetuj komunikaci
Vstup	awDataSend	ARRAY OF 3 WORDS	Poslaná data
Výstup	awDataRec	ARRAY OF 3 WORDS	Přijatá data
Výstup	bDone	BOOL	TRUE: Příkaz byl obdržen
Výstup	bBusy	BOOL	TRUE: Úkol právě probíhá
Výstup	bError	BOOL	TRUE: Nastala chyba
Výstup	iErrorCode	INT	0: Žádná chyba
Výstup	iErrorLevel	INT	0: Žádná chyba 1: Málo závažná chyba 2: Závažná chyba

6.1.4 Test komunikačního programu

Pro kontrolu, zdali komunikační program pracuje správně, byl proveden test na 2 z těchto protokolů, a to na TCP/IP Socket a EtherNet/IP.

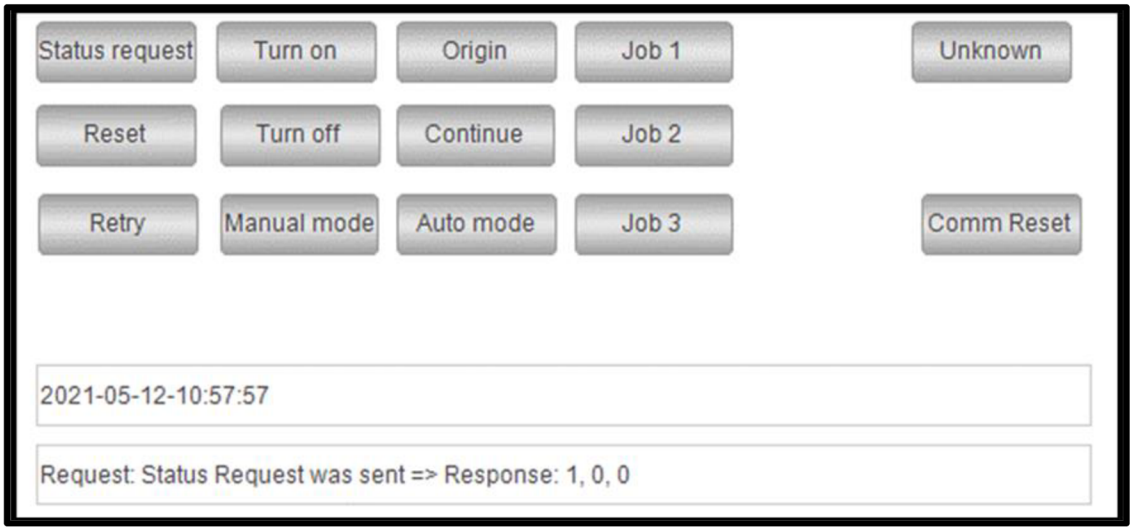
Sledováno bylo, zdali se přijatá data správně zpracují, jestli se příkaz správně přijme či nepřijme a odešle se správná odpověď na základě aktuálního stavu zařízení. Aplikace pro PLC byly vytvořeny v CODESYS verze V3.5 SP16 Patch 3 + (64-bit). Jako zařízení slave bylo použito Raspberry Pi 4 se spuštěným SoftPLC CODESYS Control for Raspberry Pi SL verze 4.0.1.0. Jednoduché testování master proběhlo na osobním počítači. Před testem byly nejdříve vytvořeny FB pro příjem a odesílání dat.

Test na protokolu EtherNet/IP

Pro testování na protokolu TCP/IP Socket byl slave nakonfigurován jako EtherNet/IP adapter, master jako EtherNet/IP scanner.

Master běžel na počítači se spuštěným SoftPLC CODESYS Control Win SL verze 3.5.16.30. Jednotlivé příkazy byly posílány přes vytvořený testovací panel ve vizualizaci a sledovala se odpověď (viz obrázek 6.4).

První testování ukázalo, že některé příkazy nebyly naprogramovány správně, protože master zaznamenával neočekávané odpovědi. Nalezené chyby byly opraveny. Po provedení dalších testů již nebyly zaznamenány další chyby.



Obrázek 6.4: Test komunikace na protokolu EtherNet/IP

Test na protokolu TCP/IP Socket

U testu na protokolu TCP/IP Socket byl slave nakonfigurován jako TCP/IP server, který čeká na připojení od klienta (master).

Pro klienta byl vytvořen skript v programovacím jazyku Python verze 3.8.3 pomocí modulu socket využívající socket API. Tento skript byl spuštěn na počítači. Požadované příkazy byly posílány manuálně zadáním do konzoly a sledovala se odpověď (viz obrázek 6.5). Více informací ohledně modulu socket a socket API lze nalézt na [24, 25].

Pro implementaci TCP/IP serveru v CODESYS byla využita knihovna CAA Net Base Services, 3.5.15.0 (CAA Technical Workgroup). Využitím uvedené knihovny se vyhneme přímé práci se sockety. Většina vývojových prostředí však nemá veřejně k dispozici podobné knihovny, pokud si je uživatel sám nevytvoří.

```
Waiting for connection
1: Status request
2: Switch to automatic mode
3: Switch to manual mode
4: Return to Origin
5: Reset
6: Retry
7: Turn on
8: Turn off
50: Continue in automatic mode
101: Job 1
102: Job 2
103: Job 3
Enter next command: 1
        Request: Status request was sent => Status: Turned off
Enter next command: 7
        Request: Turn on was sent => Response: Accepted
Enter next command: 1
        Request: Status request was sent => Status: Busy
Enter next command: 50
        Request: Continue in automatic mode was sent => Response: Not accepted
Enter next command: 101
        Request: Job 1 was sent => Response: Accepted
Enter next command: 1
        Request: Status request was sent => Status: Busy
Enter next command: 1
        Request: Status request was sent => Status: Ready, Manual mode
Enter next command: 5
        Request: Reset was sent => Response: Not accepted
Enter next command: 999
        Request: Undefined command was sent => Response: Unknown command
Enter next command: 8
        Request: Turn off was sent => Response: Accepted
Enter next command: 1
        Request: Status request was sent => Status: Turned off
Enter next command:
```

Obrázek 6.5: Test komunikace na protokolu TCP/IP Socket

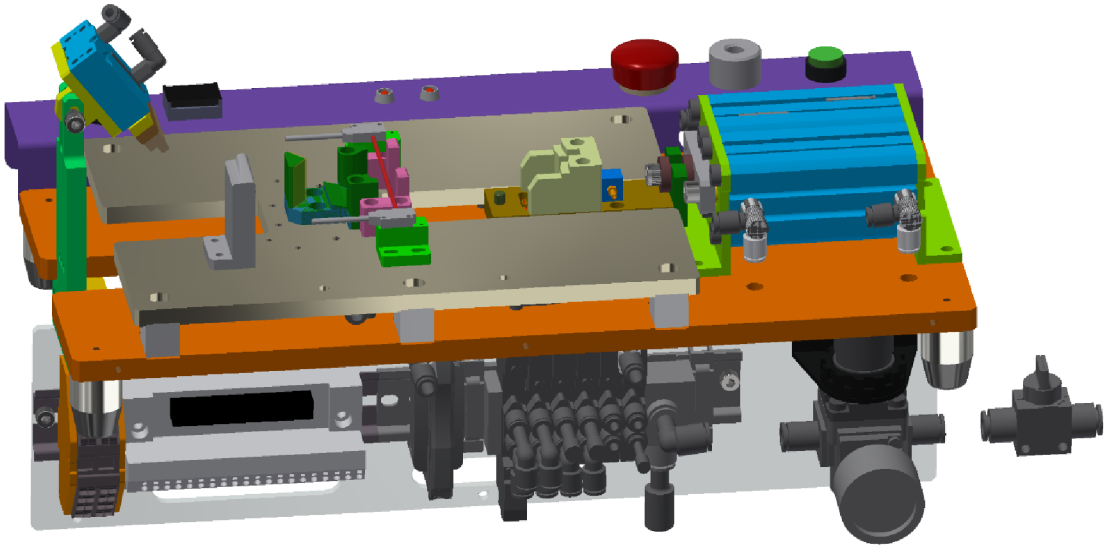
7. DOKUMENTACE

Na základě vytvořených programů byl vytvořen framework pro jednoúčelová zařízení. Pro správnou práci s frameworkem byla vytvořena referenční dokumentace přiložená k této práci jako příloha A.

Použité POU obsahují vždy krátký popis, tabulku s obrázkem použitých vstupů a výstupů, vývojový diagram, včetně ukázkových příkladů použití a různých poznámek a varování, čemu se při jejich tvoření vyvarovat, aby se předcházelo problémům. V případě nalezení nových poznámek, tipů, varování nebo chyb bude dokumentace aktualizována. Při aktualizaci budou případné změny uvedeny na titulní stránce a bude navýšeno číslo obsahující aktuální vydání dokumentace.

8. TEST ŘEŠENÍ NA PNEUMATICKÉM LISU

Test funkcionality byl ověřen na pneumatickém lisu (viz obrázek 8.1) ovládaném PLC FP0H C32ET/EP od firmy Panasonic Electric Works Europe AG. PLC se programuje v již zmíněném vývojovém prostředí Control FPWIN Pro 7 v souladu s normou IEC 61131-3. 3D model lisu se včetně popisu senzorů, akčních členů a chybových kódů nachází v příloze B. Tlačítka pro manuální ovládání lisu nebudou použita. Místo nich bude použit komunikační program.



Obrázek 8.1: 3D model lisu

8.1 Metody pro lis

Prvním krokem bylo vytvoření metod pro lis.

Nejdříve byly vytvořeny následující metody pro nastavování výstupů:

1. Uzamkni díl Lever
2. Odemkni díl Lever
3. Posuň stojan pro uzamknutí dílu Lever doprava
4. Posuň stojan pro uzamknutí dílu Lever doleva
5. Uzamkni držák s dílem Holder
6. Odemkni držák s dílem Holder
7. Vysuň písty válců pro slisování dílů
8. Zasuň písty válců pro slisování dílů
9. Uvolni písty válců pro slisování dílů

Následně byly vytvořeny metody pro kontrolu vstupů:

1. Zkontroluj, zdali je díl Lever uzamknut
2. Zkontroluj, zdali je díl Lever odemknut
3. Zkontroluj, zdali je stojan pro uzamknutí dílu Lever vpravo
4. Zkontroluj, zdali je stojan pro uzamknutí dílu Lever vlevo
7. Zkontroluj, zdali jsou písty válců pro slisování dílů vysunuty
8. Zkontroluj, zdali jsou písty válců pro slisování dílů zasunuty
10. Zkontroluj, zdali nejsou vloženy díly v lisu
11. Zkontroluj, zdali jsou vloženy oba dílu v lisu

Z popisu zařízení si lze všimnout, že válci `Lever Cyl` chybí koncový snímač pro detekci zasunutého pístu. Kdyby se v budoucnu snímač přidal stačilo by upravit pouze odpovídající metodu následujícím způsobem. Není tak nutné procházet další části kódu.

```
// Zkontroluj, zdali je součástka Lever uzamknuta – předtím
IF iMethodChk = 1 THEN
   bIn0_OK := GVL.xLeverCylDown;
   iErrorOffset := 10; (* 10 *)
END_IF;
```

```
// Zkontroluj, zdali je součástka Lever uzamknuta – potom
IF iMethodChk = 1 THEN
   bIn0_OK := GVL.xLeverCylDown;
   bIn1_OK := NOT GVL.xLeverCylUp;
   iErrorOffset := 10; (* 10 – 12 *)
END_IF;
```

8.2 Popis činnosti lisu

Dalším krokem bylo vytvoření stavových diagramů úkolů, podle kterých byly jednotlivé úkoly naprogramovány. Nakonec byl vytvořen stavový diagram automatického módu. Diagramy se nachází v příloze B.

Při zapnutí zařízení se nejdříve provede Origin, který nastaví zařízení do stejného stavu jako je znázorněno na 3D modelu lisu v příloze. Zařízení je tak připraveno pro vložení dílů. Po vykonání Origin se čeká na vložení dílů. Pokud jsou vloženy díly v lisu díly při provedení Origin, tak jsou označeny za špatné.

Při zvolení automatického módu po vložení dílů master PLC pošle příkaz lisu pro pokračování v činnosti. Spustí se tak úkol 1, ve kterém se slisují vložené díly. Potom se spustí úkol 2, který připraví zařízení do stavu, který umožní bezpečné odebrání vzniklého dílu. Zařízení pak čeká na příkaz od master PLC pro potvrzení, že díl byl odebrán. Nakonec se vykoná úkol 3, který slouží k zasunutí válců pro slisování a pro kontrolu, zda byly díly skutečně odebrány. Celý cyklus pak začne znovu.

V úkolech se používá proměnná `G_iResult`, která uchovává aktuální stav dílů jako je „Žádné informace“, „Slisováno“, „Připraveno pro odebrání“ a „Díly nejsou v pořádku“. Tato proměnná se bude posílat při obdržení příkazu na „Zjistit status“, pokud je zařízení „Připraveno“ jako Data2.

V manuálním módu je možné spouštět úkoly zvlášť, pokud to dovoluje aktuální stav zařízení.

ZÁVĚR

V teoretické části této bakalářské práce byla nastudována norma IEC 61131-3 pro PLC. Bylo zjištěno, že od posledního vydání norma podporuje kromě zapouzdření a tvoření instancí, což se dá realizovat pomocí FB, také použití prvků z rozšíření pro OOP jako jsou třídy, metody, dědičnost, polymorfismus a rozhraní. Po průzkumu vývojových prostředích v kapitole 4 bylo rozhodnuto, že prvky z rozšíření nebudou použity, protože většina vývojových prostředí, ačkoliv ctí normu IEC 61131-3, tak rozšíření pro OOP neobsahuje. Software by se tak nedal příliš dobře přenášet mezi PLC od různých výrobců.

Další část teoretické části se zabývá analýzou programů jednoúčelových zařízeních používaných ve firmě. Bylo zjištěno, že zařízení obsahují následující prvky: inicializace, Origin, časové limity, resetování, softwarové ošetření zákmitů vstupů PLC a konečný stavový automat.

V praktické části práce byl na základě získaných poznatků a požadavků vytvořen hlavní program pro jednoúčelová zařízení. Program umožňuje rychlejší zavádění nových zařízení a rychlejší provádění změn programu.

Dále byl vytvořen komunikační program pracující nezávisle na použitém komunikačním protokolu. Tento program byl otestován a odladěn na protokolu EtherNet/IP a TCP/IP Socket ve vývojovém prostředí CODESYS.

K programům byla vytvořena poslední část frameworku, a to referenční dokumentace, která je přiložena v příloze.

Navržené řešení bylo ověřeno na pneumatickém lisu. Po ověření návrhu v simulátoru v CODESYS byly zdrojové kódy přesunuty do vývojového prostředí Control FPWIN PRO 7 a nahrány do PLC od firmy Panasonic. Při přesunu se vyskytla jedna překážka. V obou vývojových prostředí se globální proměnné deklarují ve zvláštní sekci. K těmto proměnných se v CODESYS přistupuje pomocí předpony například „GVL.“.

V Control FPWIN PRO 7 se k těmto proměnným přistupuje pouze pomocí jména bez předpony. Stačilo tak odstranit předpony před jmény globálních proměnných. Tímto bylo ověřeno, že navržené řešení ctí normu IEC 61131-3 a může být použito pro další vývojová prostředí, která jsou navržena v souladu s normou.

LITERATURA

[1] What is PLCopen | PLCopen. PLCopen | for efficiency in automation [online]. Gorinchem: PLCOpen [cit. 2020-12-22]. Dostupné z: https://www.plcopen.org/what-plcopen

[2] IEC 61131-3: Programming Industrial Automation Systems: Concepts and Programming Languages, Requirements for Programming Systems, Decision-Making Aids. 2nd ed. Berlin: Springer, 2014. ISBN 3642120148.

[3] PLC Programming Preference Survey — Insights and User Comments. Automation.com [online]. Gorinchem: Automation.com [cit. 2020-10-01]. Dostupné z: https://www.automation.com/en-us/articles/2019/plc-programming-preference-survey-insights-user-co

[4] Object-Oriented Programming with SIMOTION: Fundamentals, Program Examples and Software Concepts According to IEC 61131-3. Erlangen: Publicis, 2017. ISBN 3895784567.

[5] Leverage object-oriented industrial programming. Control Engineering. 2019, 2019(7), 17-21.

[6] Leveraging OOIP, Part 2: Abstraction, nesting and interfaces. Control Engineering. 2019, 2019(8), 32-34.

[7] Creating Reusable, Hardware Independent Motion Control Applications via IEC 61131-3 and PLCopen Motion Control Profile. In: IEEE/ASME (AIM) International Conference on Advanced Intelligent Mechatronics [online]. Como, Italy: IEEE, 2001, s. 6 [cit. 2021-03-27]. ISBN 0-7803-6736-7. Dostupné z: https://ieeexplore.ieee.org/document/936763

[8] OSCAT — Neuigkeiten [online]. Wörth Donau: OSCAT, c2015 [cit. 2021-03-27]. Dostupné z: http://www.oscat.de/

[9] What is the most popular industrial network protocol? Engineering 360 - Engineerging Search & Industrial Supplier Catalogs [online]. 2019 [cit. 2021-04-15]. Dostupné z: https://insights.globalspec.com/article/11936/what-is-the-most-popular-industrial-network-protocol

[10] Research results: Protocols for Industrial Ethernet. Control Engineering [online]. Control Engineering, 2008 [cit. 2021-04-15]. Dostupné z: https://www.controleng.com/articles/research-results-protocols-for-industrial-ethernet/

[11] Průmyslový Ethernet IX: EtherNet/IP, EtherCAT. AUTOMA. 2008, 2008(10), 60-64.

[12] Profinet Entry Archive – PI North America. *The worlds leading Industrial Ethernet & Fieldbus | PI North America* [online]. PI North America, c2006-2021 [cit. 2021-5-2]. Dostupné z: https://us.profinet.com/technology/profinet/

[13] Jak na Modbus. AUTOMA. 2008, 2009(02), 46-47.
[14] Control FPWIN Pro | Panasonic. Panasonic Electric Works Czech s.r.o., Sales Office Brno | Panasonic [online].
Panasonic Electric Works Europe [cit. 2020-12-22].
Dostupné z: https://www.panasonic-electric-works.com/cz/control-fpwin-pro.htm
[15] CODESYS [online]. Kempten: CODESYS, c2020 [cit. 2020-12-22].
Dostupné z: https://www.codesys.com/
[16] PLC Programming with SIMATIC STEP 7 (Tia Portal) | Software in Tia Portal | Siemens Global. Siemens [online].
Mnichov: Panasonic Electric Works Europe [cit. 2020-12-22].
Dostupné z: https://new.siemens.com/global/en/products/automation/industry-software/automation-software/tia-portal/software/step7-tia-portal.html
[17] Studio 5000 Logic Designer | Rockwell Automation [online]. Milwaukee: Rockwell Automation, c2020 [cit. 2020-12-22].
Dostupné z: https://www.rockwellautomation.com/en-au/products/software/factorytalk/designsuite/studio-5000/studio-5000-logix-designer.html
[18] Programming GX Works3 Features of the software Programmable Controllers MELSEC | MITSUBISHI ELECTRIC FA. MITSUBISHI ELECTRIC Global website [online].
Tokio: Mitsubishi Motors Corporation [cit. 2020-12-22]. Dostupné z: https://www.mitsubishielectric.com/fa/products/cnt/plceng/smerit/gx_works3/programming.html
[19] Sysmac Studio | OMRON, Česká republika. Průmyslová automatizace | OMRON, Česká republika [online]. c2021 [cit. 2021-03-09].
Dostupné z: https://industrial.omron.cz/cs/products/sysmac-studio
[20] KV STUDIO Ver.9: Global version — KV-H9G | KEYENCE America. Sensors and Machine Vision Systems for Factory Automation | KEYENCE America [online]. Keyence Corporation, c2020 [cit. 2020-12-22]. Dostupné z: https://www.keyence.com/products/controls/plc-package/kv_nano/models/kv-h9g/
[21] Drive application programming — Automation Builder | ABB. ABB Group. Leading digital technologies for industry — ABB Group [online]. ABB Group, c2020 [cit. 2020-12-22].
Dostupné z: https://new.abb.com/plc/automationbuilder/drives
[22] TwinCAT | Automation software Beckhoff | Beckhoff Česká republika. Beckhoff | New Automation Technology | Beckhoff Česká republika [online].
Verl: Beckhoff Automation, c2020 [cit. 2020-12-22].
Dostupné z: https://www.beckhoff.com/cs-cz/products/automation/twincat/
[23] RICHARDS, Mark. Software Architecture Patterns. Sebastopol (California): O'Reilly, 2015. ISBN 9781491924242.

[24] Socket Programming in Python (Guide) - Real Python. Python Tutorials — Real Python [online]. [cit. 2021-03-16].
Dostupné z: https://realpython.com/python-sockets/
[25] Socket Programming HOWTO — Python 3.9.2 documentation. Python Tutorials — Real Python [online]. Python Software Foundation, 2021 [cit. 2021-03-16].
Dostupné z: https://docs.python.org/3/howto/sockets.html

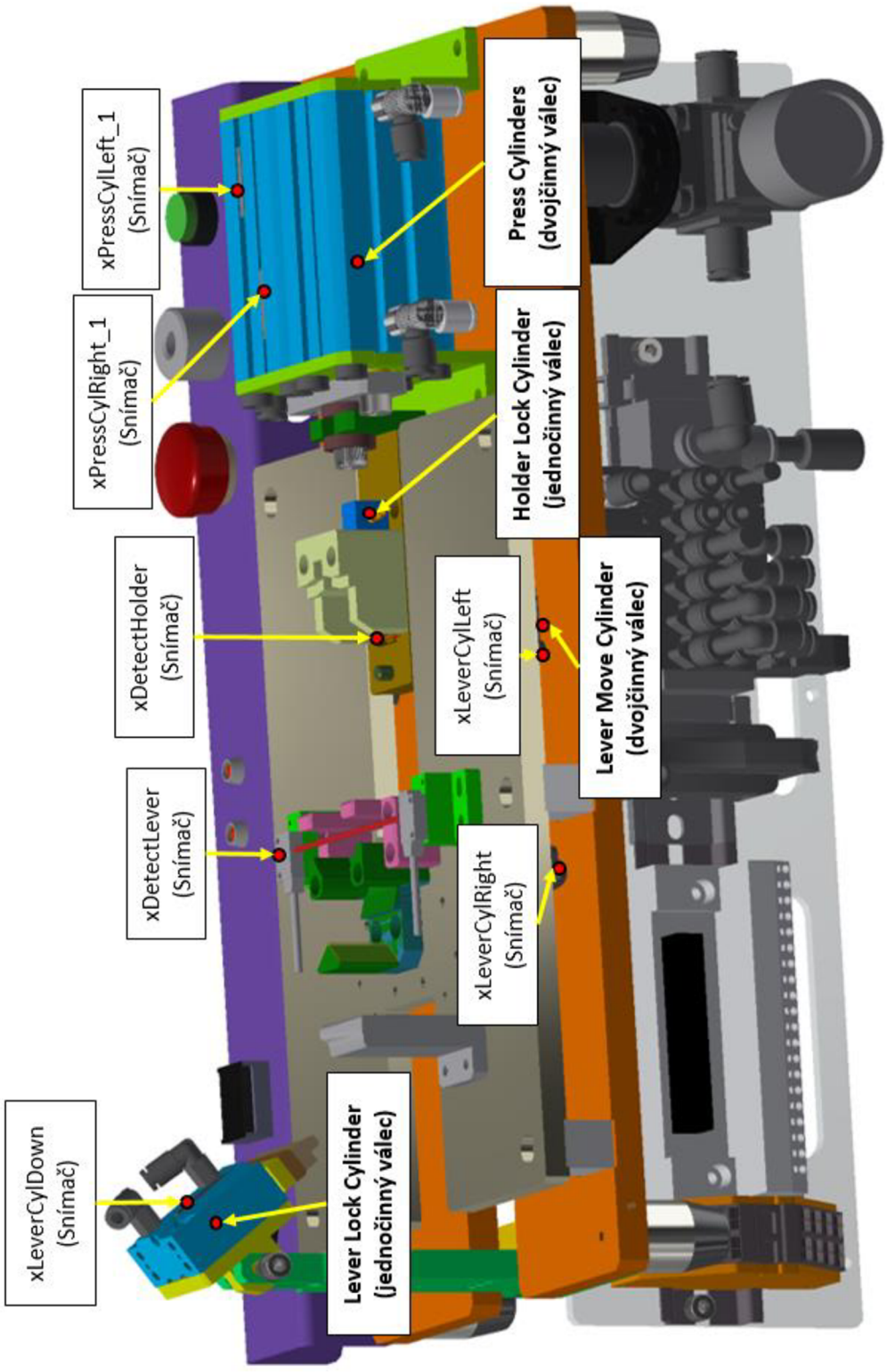
SEZNAM PŘÍLOH

Příloha A - Referenční dokumentace k frameworku

Nachází se v příloze v souboru PLC_manual.pdf

Příloha B - Pneumatický lis

B.1 3D model



B.2 Seznam vstupů a výstupů

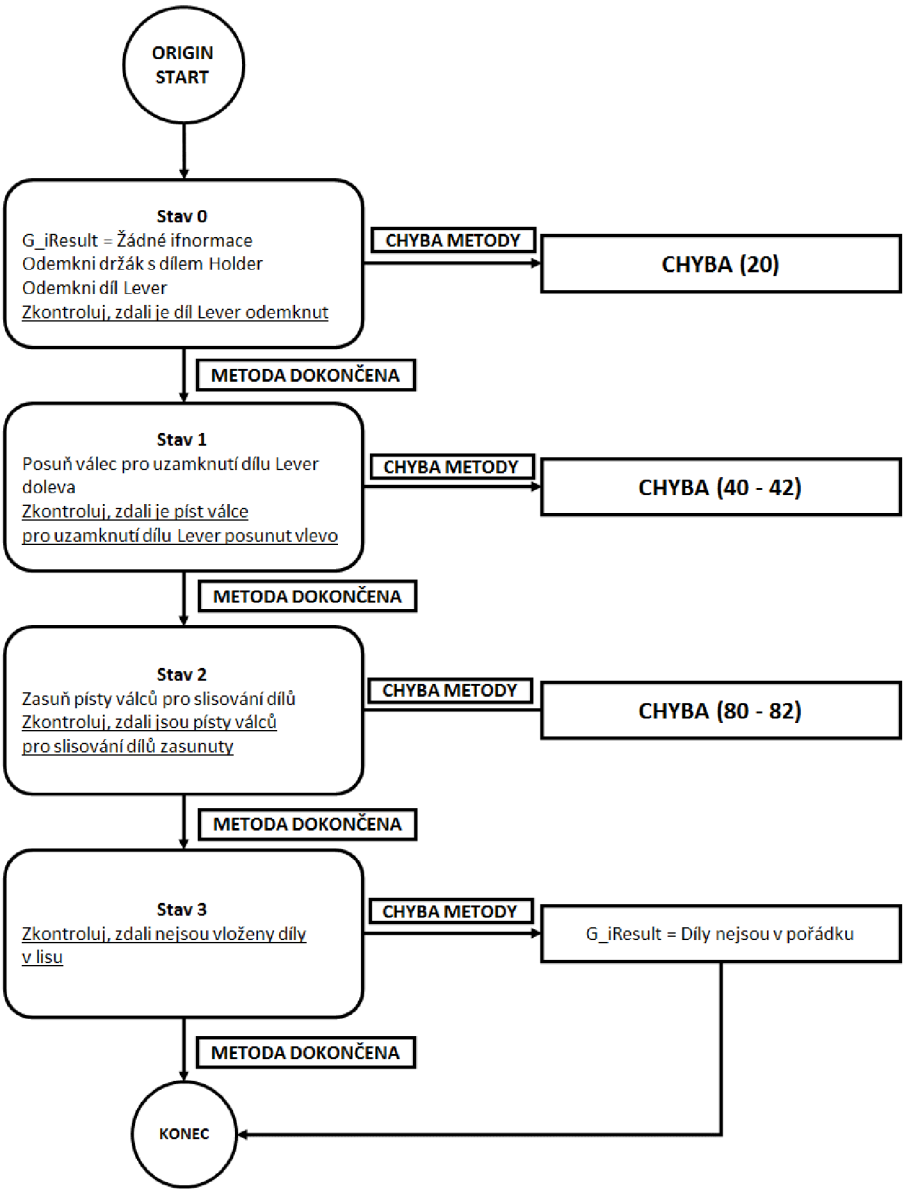
Název	Typ	Popis
xLeverCylDown	DI	Snímač pro detekci vysunutí pístu válce Lever Lock Cylinder
xLeverCylRight	DI	Snímač pro detekci vysunutí pístu válce Lever Move Cylinder
xLeverCylLeft	DI	Snímač pro detekci zasunutí pístu válce Lever Move Cylinder
xPressCylLeft_1	DI	Snímač pro detekci zasunutí pístů válců Press Cylinder
xPressCylRight_1	DI	Snímač pro detekci vysunutí pístů válců Press Cylinder
xDetectHolder	DI	Snímač pro detekci dílu Holder
xDetectLever	DI	Snímač pro detekci dílu Lever
xAirPressure	DI	Snímač tlaku
yLeverMove	DO	Ovládání ventilu válce Lever Lock Cylinder
yLeverLock	DO	Ovládání ventilu válce Lock Cylinder
yPressRight	DO	Ovládání ventilů válců Press Cylinders
yPressLeft	DO	Ovládání ventilů válců Press Cylinders
yHolderLock	DO	Ovládání ventilu válce Holder Lock Cylinder

Poznámky: DI (Digital Input), DO (Digital Output)

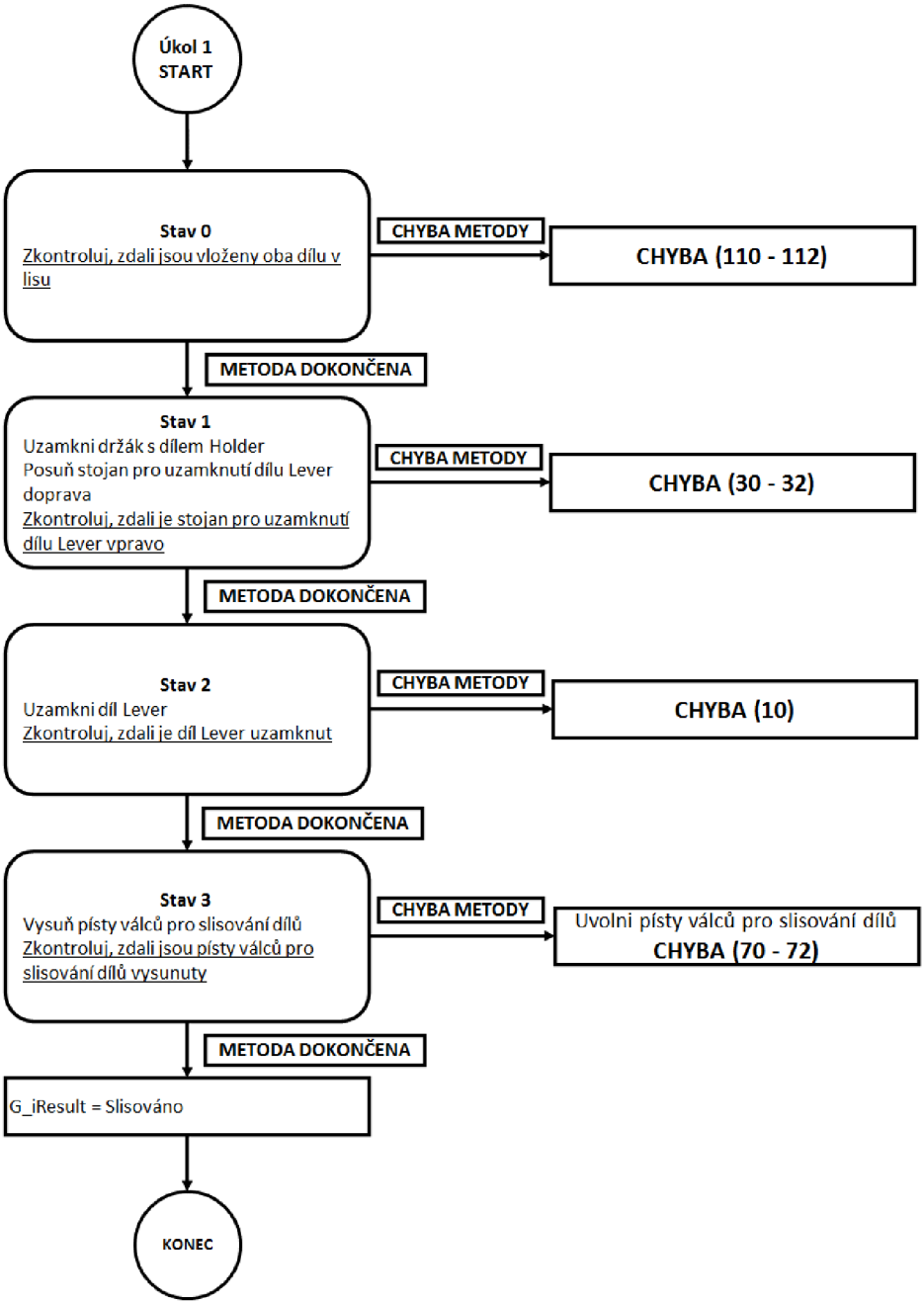
B.3 Seznam chybových hlášení

Press - Error list		
No.	**Level**	**Description**
1	2	Air pressure is not good
10	2	CYLINDER - LEVER LOCK - UNLOCKED when should be LOCKED
20	2	CYLINDER - LEVER LOCK - LOCKED when should be UNLOCKED
30	2	CYLINDER - LEVER MOVE - OPEN sensor ON when should be OFF
31	2	CYLINDER - LEVER MOVE - CLOSED sensor OFF when should be ON
32	2	CYLINDER - LEVER MOVE - OPEN sensor ON when should be OFF, CLOSED sensor OFF when should be ON
40	2	CYLINDER - LEVER MOVE - OPEN sensor OFF when should be ON
41	2	CYLINDER - LEVER MOVE - CLOSED sensor ON when should be OFF
42	2	CYLINDER - LEVER MOVE - OPEN sensor OFF when should be ON, CLOSED sensor ON when should be OFF
70	2	CYLINDER - PRESS - OPEN sensor OFF when should be ON
71	2	CYLINDER - PRESS - CLOSED sensor ON when should be OFF
72	2	CYLINDER - PRESS - OPEN sensor OFF when should be ON, CLOSED sensor ON when should be OFF
80	2	CYLINDER - PRESS - OPEN sensor ON when should be OFF
81	2	CYLINDER - PRESS - CLOSED sensor OFF when should be ON
82	2	CYLINDER - PRESS - OPEN sensor ON when should be OFF, CLOSED sensor OFF when should be ON
100	2	PARTS - HOLDER - Detected when it should not
101	2	PARTS - LEVER - Detected when it should not
102	2	PARTS - HOLDER, LEVER - Detected when it should not
110	2	PARTS - HOLDER - Not detected when it should be inserted
111	2	PARTS - LEVER - Not detected when it should be inserted
112	2	PARTS - HOLDER, LEVER - Not detected when it should be inserted
999	2	Undefined method error

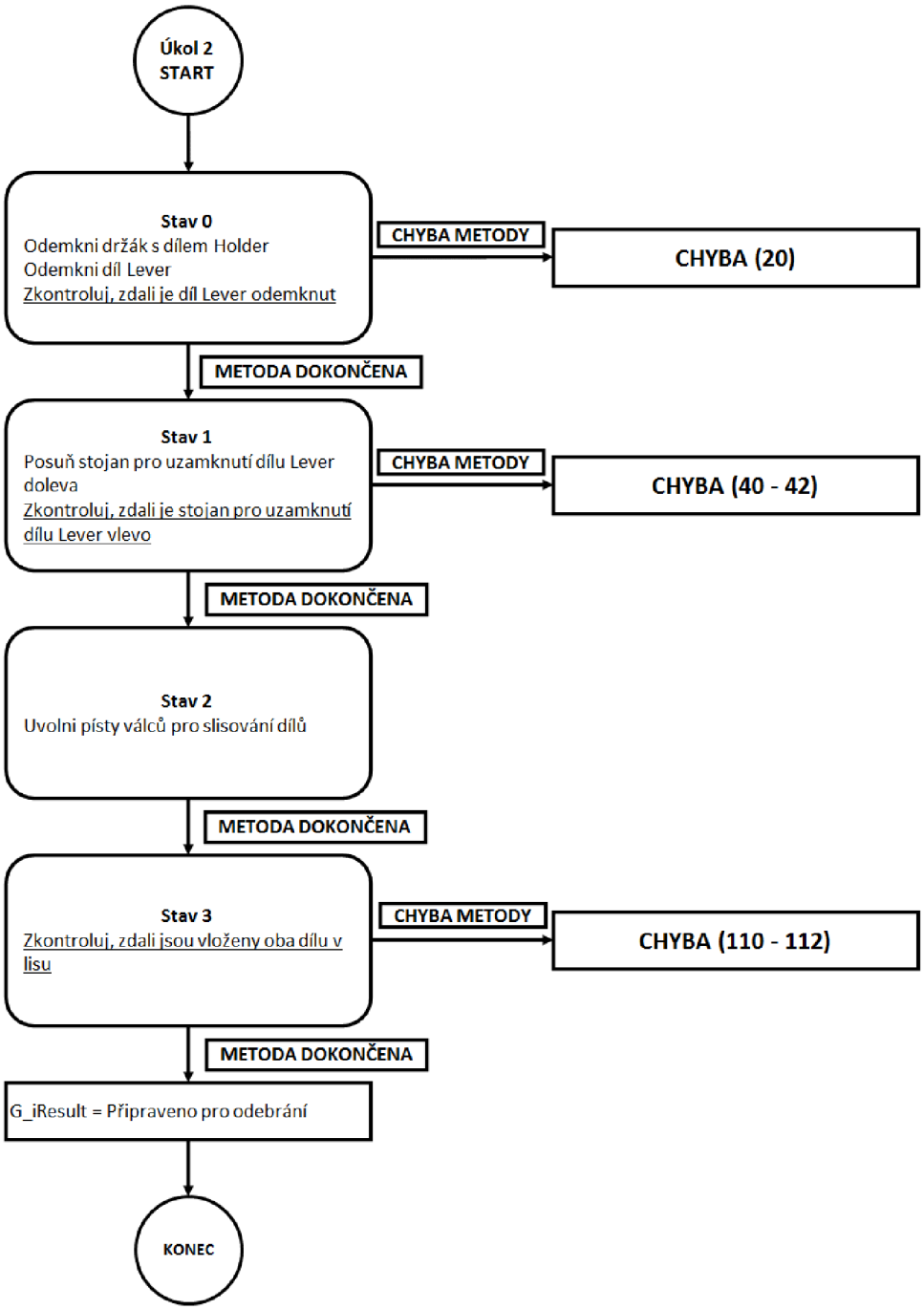
B.4 Stavový diagram – Origin



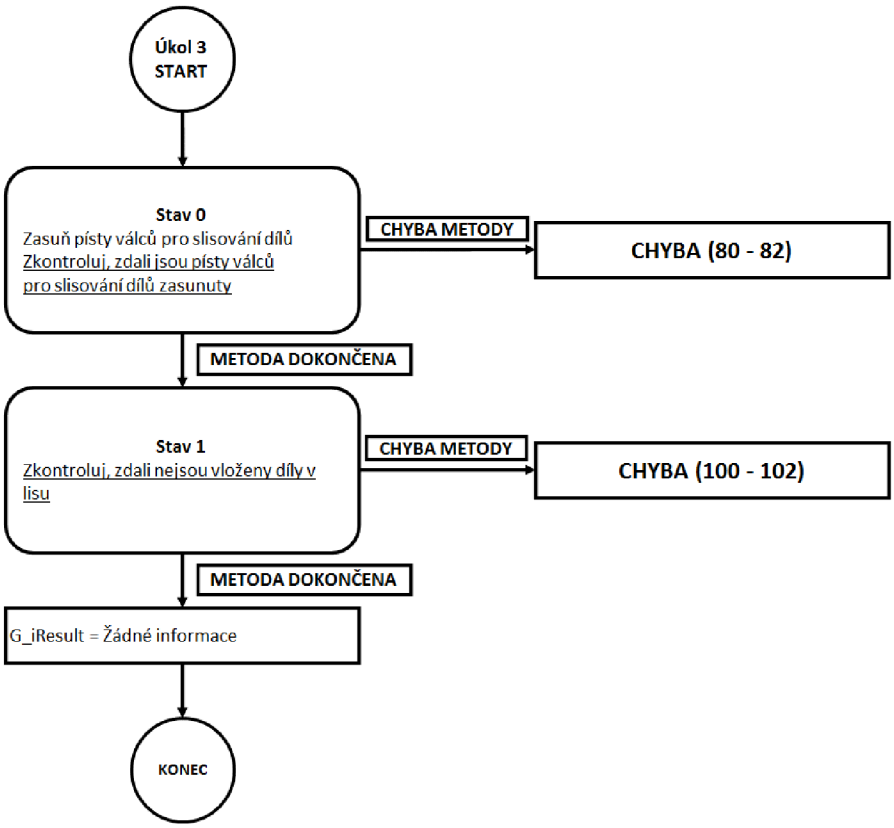
B.5 Stavový diagram – Úkol 1



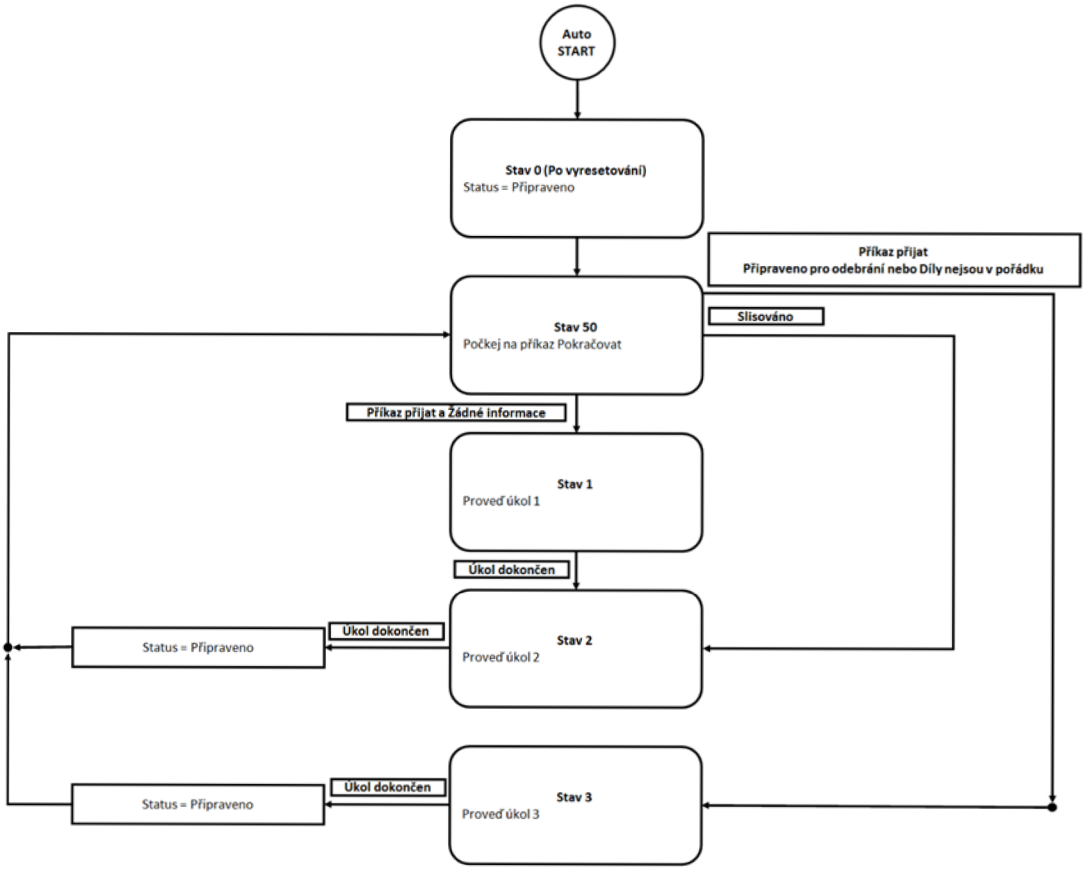
B.6 Stavový diagram – Úkol 2



B.7 Stavový diagram – Úkol 3



B.8 Stavový diagram – Automatický mód



Poznámky: Pro přehlednost nejsou zobrazeny přechody do stavu „Chyba“, které nastanou, pokud vznikne chyba při provádění úkolů.

Příloha C - Obsah CD

- Text bakalářské práce ve formátu PDF
- Zdrojové kódy pro testování komunikace na protokolu TCP/IP Socket
- Zdrojové kódy pro testování komunikace na protokolu EtherNet/IP
- Framework – zdrojové kódy a dokumentace