České vysoké učení technické v Praze
Fakulta elektrotechnická

DIPLOMOVÁ PRÁCE

Testování pohonů na síti Ethernet/IP

Praha, 2013 Autor: Petr Procházka

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 _____ _____

podpis

Poděkování

Tímto bych chtěl poděkovat vedoucímu diplomové práce Ing. Pavlu Burgetovi PhD. za rady a přípomínky, společnosti Siemens, s.r.o., jakožto zadavateli této práce. Dále bych chtěl poděkovat svým rodičům a přátelům za jejich podporu a také bych chtěl hlavně poděkovat členům testlabu společnosti Siemens, s.r.o, kteří mě mnohému naučili. Děkuji.

Abstrakt

Ve společnosti Siemens, s.r.o. se implementuje možnost komunikace na síti Ethernet/IP a aplikační profil protokolu CIP pro nízkonapěťový frekvenční měnič. V průběhu vývoje je nezbytné pravidelně testovat dané zařízení. Testování, který probíhá současně s vývojem je jednou z prvních zpětných vazeb pro vývojáře. Práce se na začátku zabývá protokolem CIP a aplikačním profilem AC Drive, který je implementován do zařízení. Následně je popsáno objektové prostředí PyTeMat (Python Test Automat) pro psaní automatických testů, které je rozšířeno o nezbytné moduly pro testování frekvenčního měniče. Na konci jsou popsané automatické testy, které byly navrženy a vytvořeny pro toto zařízení v testovacím prostředí PyTeMat. Klíčová slova: *Ethernet/IP, CIP, Python Test Automat, testování, frekvenční měnič*

Abstract

In Siemens Ltd. the possibility of communication via the Ethernet/IP network and the application profile of the CIP protocol for low-voltage frequency converter is implemented. During the development, it is necessary to regularly test this device. Integration test, which is running simultaniously with the developement, is one of the first feedbacks for the developer. The thesis deals with the CIP protocol and application profile AC Drive which is implemented into device. Later, it describes the object environment PyTeMat (Python Test Automat) for creation of automatic tests which is widened by necessary modules for frequency converter testing. In the end, the automatic tests created for this device in testing environment PyTeMat are described. Keywords: *Ethernet/IP, CIP, Python test automat, testing, frequency converter*

vložit originální zadáníííííííííííííííííí !!!!!

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Důležitou součástí vývoje, ať už softwaru nebo hardwaru, je testování. Testování se skládá z několika fází, nejčastěji z integračního, systémového a akceptačního testování. Integrační test probíhá současně s vývojem a je jednou z prvních zpětných vazeb pro vývojáře. Systémový test začíná na konci vývoje a jedná se o závěrečné testování před předáním produktu zákazníkovi. Zákazník si následně zařízení otestuje, zda-li odpovídá jeho požadavkům. Tato část se nazývá akceptační test. S postupem vývoje a neustálou potřebou testovat je důležitá automatizace testů, které automaticky otestují konkrétní funkčnost nebo vlastnost daného zařízení/softwaru. Pro usnadnění psaní testů je dobré si navrhnout a naprogramovat testovací prostředí.

Tato diplomová práce vznikla v průběhu integračního testu pro nízkofrekvenční napěťový měnič[1]. Frekvenční měnič, který podporuje PROFIdrive profil nad technologiemi PROFINET a PROFIBUS, je rozšířen o komunikační rozhraní Ethernet/IP, podporu CIP (Common Industrial Protocol) protokolu a aplikační profil AC Drive. Náplní této práce je rozšíření objektové struktury PyTeMat (Python Test Automat) o potřebné moduly a vytvoření sady automatických testů, které napomůžou odhalit chyby při vývoji a implementaci AC Drive profilu.

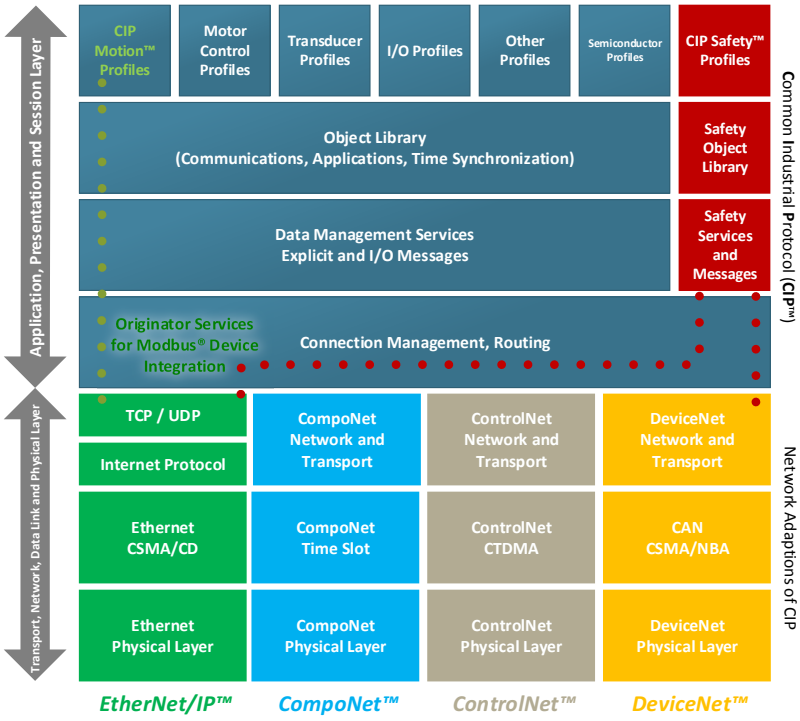
V první a druhé kapitole je popsán objektový model protokolu CIP, komunikace na síti Ethernet/IP, objektový model zařízení a profily, které jsou v tomto zařízení implementované nad protokolem CIP. V třetí kapitole je představeno testovací prostředí PyTeMat (Python Test Automat), které je vyvinuto (a nadále se vyvíjí), nejen pro testování našeho konkrétního zařízení, ale také pro ostatní projekty testlabu společnosti Siemens, s.r.o. Součástí PyTeMatu jsou moduly, objektové struktury a funkce, které byly vyvinuty pro potřeby testování našeho zařízení. V poslední kapitole jsou popsané automatické testy pro testované zařízení. Testuje se převážně funkcionalita a chování z hlediska specifikace protokolu CIP [1] a dalších vlastností zařízení na základě interní dokumentace zvané *Feature Description*.

[1] Z důvodu compliance politky firmy Siemens, s.r.o není uveden konkrétní typ zařízení.

Kapitola 2

Protokol CIP a Ethernet/IP

Ethernet/IP je otevřený síťový standard pro průmyslovou automatizaci, vyvinutý firmou Rockwell Automation a spravovaný organizací ODVA (Open DeviceNet Vendor Association). Ethernet/IP využívá standardního Ethernetu (IEE802.3) a sady protokolů TCP/IP balíku. Poslední tři vrstvy ISO/OSI modelu - relační, prezentační a aplikační realizuje protokol CIP *(Common Industrial Protocol)*. Protokol CIP je nezávislý na prvních čtyřech vrstvách síťového ISO/OSI modelu. Jak můžeme vidět na obrázku 2.1, tak kromě Ethernetu/IP tento protokol v současné době využívají také sítě CompoNet, ControlNet a DeviceNet. Tato kapitola čerpá z literatury [1] až [10].



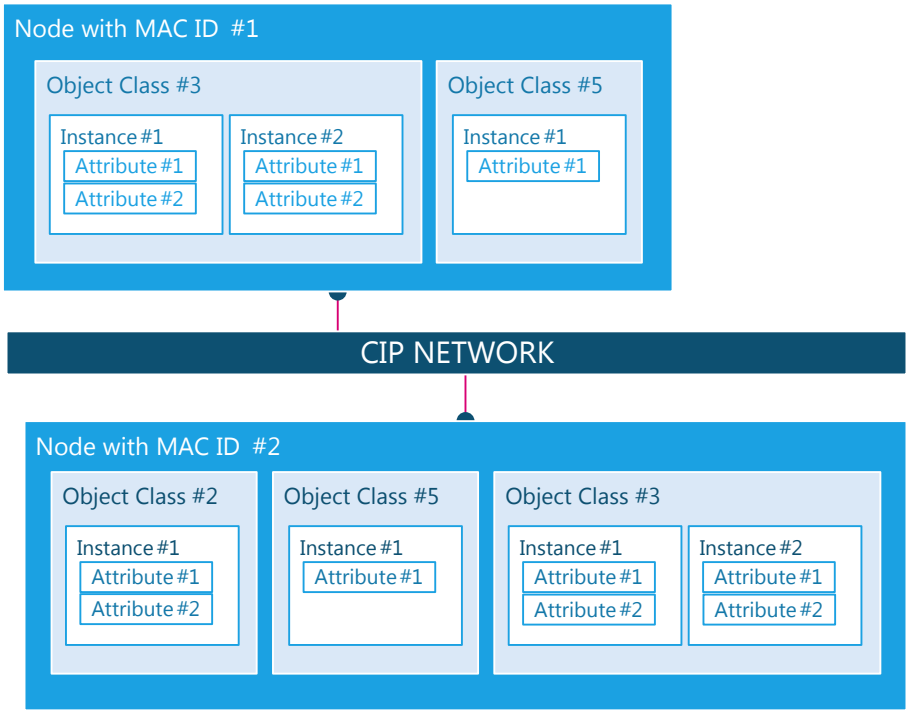
Obrázek 2.1: Protokoly rodiny CIP

2.1 CIP protokol

CIP protokol je objektově orientovaný protokol poskytující spojení mezi průmyslovými zařízeními. Hlavním účelem protokolu CIP je přenos řídicích dat a přenos dalších dat, které mohou sloužit k popisu, konfiguraci nebo k diagnostice zařízení.

2.1.1 Objektový model

V CIP síti je každé zařízení abstraktně nazýváno uzlem a každý uzel v této síti má unikátní adresu - MAC ID (Media Access Control Identifier). V případě zařízení v síti Ethernet/IP je adresa uzlu (MAC ID) jeho IP adresa. Každý uzel lze popsat kolekcí objektů. Tyto objekty jsou instancemi tříd. Třída obsahuje kolekci stejných objektů a je pouze zobecněním objektu. Objekt abstraktně reprezentuje určitou část uvnitř zařízení. Každý objekt může mít až n instancí dané třídy. Instance charakterizuje již reálnou část zařízení. Každá instance objektu obsahuje atributy, které popisují konkrétní vlastnost nebo stav objektu. Atributy jsou pro všechny instance jedné třídy stejné, různé jsou pouze jejich hodnoty. Obrázek 2.2 názorně vyjadřuje objektovou strukturu a její hierarchii.



Obrázek 2.2: Příklad objektové struktury uzlů v CIP síti

Tabulka 2.1 znázorňuje objektovou strukturu na příkladu auta.

Class	**Instance**	**Attribute**	**Value**
Auto	Škoda Fabia	Rok výroby	2001
		Najeto	120000
	Ford Focus	Rok výroby	2003
		Najeto	89000

Tabulka 2.1: Objektový model

Každý objekt navíc poskytuje sadu služeb pro operace s atributy a jiné služby. Základní seznam služeb, které může objekt poskytovat je následující :

- Get_Attributes_All
- Set_Attribute_All
- Get_Attribute_Single
- Set_Attribute_Single
- Get_Attribute_List
- Set_Attribute_List
- No_Operation
- Get_Member
- Set_Member
- Insert_Member
- Remove_Member

Výše uvedené služby nejsou povinné, jsou volitelné, není-li ve specifikaci uvedeno jinak. Abychom mohli vyčíst hodnotu atributu, například pomocí služby Get_Attribute_Single, musíme atribut umět adresovat. To znamená, že musíme mít adresy pro třídu, instanci objektu a atribut.

Adresa třídy, tzv. Class ID, je reprezentována číslem v hexadecimálním formátu. Třídy, které specifikuje ODVA mají již přidělené konkrétní Class ID. Pro třídy, které si chce výrobce zařízení specifikovat sám, smí Class ID zvolit pouze ve vymezeném rozsahu, který je pro tento účel vymezen. Další část rozsahu adres má ODVA rezervováno pro budoucí použití.

Dále musíme mít konkrétní adresu instance objektu, což je celé nezáporné číslo. Každý objekt může mít nepovinně instanci s adresou 0 a tato instance obsahuje volitelné atributy týkající se celé třídy, jako například :

- Revize
- Maximální počet možných instancí
- Aktuální počet instancí třídy
- List atributů a jejich počet

Poslední adresa je adresa atributu - Attribute ID. Jedná se stejně jako v případě Instance ID o celé nezáporné číslo. Všechny objekty a jejich atributy jsou popsány v kapitole 5: Object Library specifikace [1].

2.1.2 Komunikace na síti Ethernet/IP

Každé zařízení musí podporovat UCMM (Unconnected Message Manager), jedná se o třídu bez Class ID a skrze tento objekt se vytváří všechna připojení za pomocí služby UCMM Forward_Open. Žádost UCMM Forward_Open obsahuje všechny důležité informace pro vytvoření spojení, jako například :

- Time-out informaci pro vytvářené spojení
- CID (Connection ID) spojení ve směru zakladatel → cíl
- CID spojení ve směru cíl → zakladatel
- Informace o identitě zařízení vytvářející spojení
- RPI (Request Packet Interval)
- Informace o datech, která se budou vyměňovat
- a další

Všechna připojení, která se vytváří na síti Ethernet/IP lze rozdělit do dvou kategorií, a to na implicitní a explicitní. Implicitní komunikace je vždy spojovaná (connected). Explicitní komunikace může být jak spojovaná (connected), tak i nespojovaná (unconnected).

2.1.2.1 Implicitní komunikace

Implicitní komunikace se nejčastěji využívá pro cyklickou výměnu I/O dat, synchronizaci nebo pro řízení pohybu. Na transportní vrstvě ISO/OSI modelu využívá protokolu UDP a v datové části není žádný jiný zapouzdřený protokol, nýbrž jen data. Data, která se vyměňují, jsou definovaná ve Forward_Open dotazu, který vytváří tuto komunikaci. Implicitní komunikace je typu producer/consumer, přičemž zařízení může produkovat pouze jedna data, která však může poskytovat více spotřebitelům. Data se tedy mohou posílat unicastem nebo multicastem. Komunikace probíhá v pravidelném intervalu na základě RPI (Request Packet Interval). Na základě CID (Connection ID) obě zařízení ví, o které připojení se jedná a jaká data obsahuje.

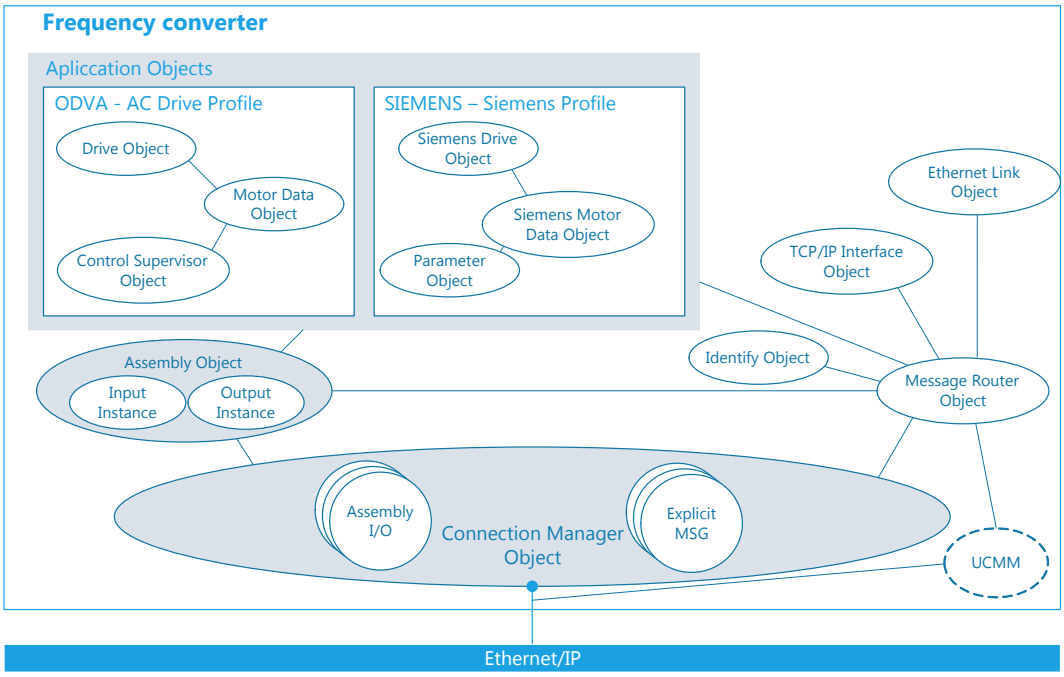
2.1.2.2 Explicitní komunikace

Explicitní komunikace je acyklickou komunikací typu request/response a používá se nejčastěji pro konfiguraci zařízení, diagnostiku nebo sběr dat. Na transportní vrstvě ISO/OSI modelu využívá protokolu TCP. Jak již bylo řečeno, explicitní komunikace může být spojovaná (connnected) nebo nespojovaná (unconnected). Vysíláme-li spojovanou (connected) explicitní zprávu, předchází výměně dat Forward_Open dotaz a každá zpráva je potvrzovaná. Vysíláme-li nespojovanou (unconnected) explicitní zprávu, tvůrce této zprávy vysílá rovnou požadavek, v zařízení je zpracován objektem UCMM a zařízením je rovnou vyslána odpověď.

Kapitola 3

Frekvenční měnič a protokol CIP

Stejně jako PROFIdrive i protokol CIP definuje aplikační vrstvu různých zařízení pro řízení motorů. V oddělení CT DC společnosti Siemens, s.r.o. se vyvíjí pro jeden z frekvenčních měničů možnost komunikace na síti Ethernet/IP a hlavně aplikační úroveň protokolu CIP. V následující části si popíšeme aplikační profil tohoto protokolu pro řízení motorů, data v cyklické komunikaci a objekty, které z hlediska specifikace musí zařízení podporovat. Na obrázku 3.1 vidíme, jak vypadá měnič z hlediska protokolu CIP.



Obrázek 3.1: Objektový model frekvenčního měniče

Objekty v typickém zařízení se dají rozdělit do tří skupin, a to na objekty :

- Základní objekty
- Aplikační objekty
- Objekty specifické pro danou síť

V následujících podkapitolách jsou popsány objekty a jejich atributy, které podporuje náš frekvenční měnič. Podrobněji jsou popsány objekty, jenž jsou předmětem následného testování.

3.1 Základní objekty

3.1.1 01_{HEX} Identify Object

Tento objekt je povinný pro všechna zařízení nehledě na jeho typ. Class ID tohoto objektu je 01_{HEX}. Tento objekt poskytuje základní informace o daném zařízení. Atributy v následující tabulce 3.1 jsou povinné pro všechna zařízení. Tabulka obsahuje adresu atributu - Attribute ID, přístup, jméno atributu a datový typ. Přístupem se myslí, zda-li hodnota atributu je určena pouze pro čtení (GET) nebo i pro zápis (SET). Objekt, na základě pravidel přístupu, poskytuje nad těmito atributy služby Get_Attribute_Single a Set_Attribute_Single. Dále je dobré při vyčítání atributů znát i datový typ, kvůli konverzi, se kterou je třeba se vypořádat při psaní testů.

Attr. ID	Access Rule	Attribute Name	Data Type
1	GET	Vendor ID	UINT
2	GET	Device Type	UINT
3	GET	Product Code	UINT
4	GET	Revision	STRUCT OF :
		Major Revision	USINT
		Minor Revision	USINT
5	GET	Status	WORD
6	GET	Serial Number	UDINT
7	GET	Product Name	SHORT_STRING

Tabulka 3.1: Identify Object

Atribut **Vendor ID** poskytuje informaci o výrobci daného zařízení. Každý výrobce má přidělené své ID a tato ID spravuje ODVA. Pro Siemens je toto ID rovno hodnotě 1251.

Atribut **Device Type** - typ zařízení je určeno jeho profilem. Náš frekvenční měnič má (viz. obrázek 3.1) dva profily, a to AC Drive profil s ID 02_{HEX} , který specifikuje CIP [1]. Druhý profil Siemens[1], který je vyvinutý výrobcem a řadí se do kategorie Vendor specific.

Atribut **Product Code** je určen výrobcem zařízení a určuje produktovou řadu. Tento produktový kód se musí shodovat s produktovým kódem v příslušném EDS (Electronic Data Sheet) souboru.

Atribut **Revision** se skládá ze dvou krátkých neznaménkových integerů. Pokud se změní zařízení a není třeba jej změnit v EDS souboru, změna se projeví v minor revision. Pokud se zařízení změní natolik, že je třeba provést i změnu v EDS (Electronic Data Sheet) souboru, inkrementuje se major revision.

Atribut **Status** poskytuje informaci o zařízení, zda-li například nebyla provedena změna v konfiguraci nebo zda zařízení nehlásí chybu či varování. Každý bit tohoto atributu má svůj vlastní význam a je detailně popsán v popisu Indentify object ve specifikaci [1].

3.1.2 04_{HEX} Assembly Object

Tento objekt seskupuje atributy z různých objektů do jednoho atributu Data s Attribute ID 3. Assembly objekt je využíván pro implicitní zprávy, tedy v cyklické komunikaci, kde máme tzv. Input Assembly a Output Assembly. Input Assembly i Output Assembly jsou instancí Assembly objektu. Každý profil, kde je třeba cyklické komunikace, má předdefinované již některé instance a je na výrobci zařízení, které z nich bude podporovat. Taktéž je určen rozsah instancí pro výrobce, které si může nadefinovat libovolně. Rozsahy instancí a jejich přidělení se nachází v tabulce 3.2.

Rozsah	**Přidělení**
01_{HEX} - 63_{HEX}	Přiděleno profilům zařízení definované v CIP
64_{HEX} - $C7_{HEX}$	Přiděleno pro volné použití výrobcům zařízení
$C8_{HEX}$ - $2FF_{HEX}$	Přiděleno profilům zařízení definované v CIP
300_{HEX} - $4FF_{HEX}$	Přiděleno pro volné použití výrobcům zařízení
500_{HEX} - $FFFFF_{HEX}$	Přiděleno profilům zařízení definované v CIP
100000_{HEX} - $FFFFFFFF_{HEX}$	Rezervovaný rozsah pro budoucí využití v CIP

Tabulka 3.2: Rozsah instancí Assembly objektu

[1]Název profilu je pozměněn z důvodu compliance politiky firmy Siemens, s.r.o.

Instance tohoto objektu mohou být dvojího typu, a to :

- **Statické** - Atributy seskupené do atributu 3 jsou již nadefinované a nelze je měnit.
- **Dynamické** - Atributy svázané do atributu 3 lze volně přidávat i odebírat. Objekt pak musí podporovat příslušné služby jako Insert_Member a Remove_Member. Instance ID vytvořeného assembly lze přidělit z rozsahu instancí určené pro výrobce 3.2.

3.1.3 02_{HEX} Message Router Object

Skrze objekt Message Router s Class ID 02_{HEX} klient adresuje požadavky (služby) na příslušné objekty. Message Router objekt nebude předmětem testování.

3.1.4 06_{HEX} Connection Manager Object

Objekt Connection Manager s Class ID 06_{HEX} alokuje a spravuje interní zdroje asociované s implicitním i excplicitním připojením. Connection Manager objekt nebude předmětem testování.

3.2 Objekty specifické pro Ethernet/IP

Každá síť využívající protokolu CIP má své specifické objekty, které je povinné implementovat. Pro Ethernet/IP jsou tyto objekty definované ve specifikaci [2].

3.2.1 $F6_{HEX}$ Ethernet Link Object

Ethernet Link objekt s Class ID $F6_{HEX}$ obsahuje specifické čítače a informace IEEE 802.3 komunikačního rozhraní, jako například fyzickou adresu. Pro každé rozhraní IEEE 802.3 je přesně jedna instance tohoto objektu. Ethernet Link objekt nebude předmětem testování.

3.2.2 $F5_{HEX}$ TCP/IP Interface Object

Objekt TCP/IP Interface s Class ID $F5_{HEX}$ poskytuje možnost konfigurace TCP/IP síťového rozhraní daného zařízení, jako například IP adresu, síťovou masku a gateway. Tabulka 3.3 obsahuje všechny atributy, které naše zařízení podporuje.

Attr. ID	Access Rule	Attribute Name	Data Type
1	GET	Status	DWORD
2	GET	Configuration Capability	DWORD
3	GET/SET	Configuration Control	DWORD
4	GET	Physical Link Object	STRUCT OF :
		Path Size	UINT
		Path	Padded EPATH
5	GET/SET	Interface Configuration	STRUCT OF :
		IP Address	UDINT
		Network Mask	UDINT
		GateWay Address	UDINT
		Name Server	UDINT
		Name Server 2	UDINT
		Domain Name Length	UDINT
		Domain Name	STRING
6	GET/SET	Host Name	STRUCT OF :
		Host Name Length	UINT
		Host Name	STRING
10	GET	Select ACD	BOOL
11	GET/SET	Last Conflict Detected	STRUCT OF :
		ACD Activity	USINT
		Remote MAC	6xUSINT
		ARP PDU	28xUSINT

Tabulka 3.3: TCP/IP Interface Object

Atribut **Status** poskytuje informaci o stavu TCP/IP síťového rozhraní. Prvních sedm bitů tohoto atributu má svůj význam. Například první čtyři bity udávají informaci, zda-li konfigurace tohoto rozhraní je získana pomocí BOOTP, DHCP, pevně uložená v zařízení, nebo že zařízení nemá zatím žádné nastavení. Pátý bit[1] například indikuje, zda-li atribut 5 Interface Configuration byl nastaven a zařízení vyžaduje restart. Šestý bit[1] indikuje, zda-li nastala kolize adres.

Atribut **Configuration Capability** dává informaci o tom, zda-li zařízení je například schopno získat nastavení TCP/IP rozhraní pomocí BOOTP, DHCP, zda-li podporuje ACD a mnoho dalšího. Více informací lze zjistit v [1].

[1] počítáno od nuly

3.3 Profil AC Drive

Specifikace CIP [1] v kapitole 6 definuje profily různých zařízení. Každý profil obsahuje informace o třídách a jejich atributech, I/O datech a chování daného zařízení. V našem případě se nás týká profil AC Drive s hexadecimálním identifikátorem 02_{HEX}.

3.3.1 Cyklická komunikace

Jak již bylo naznačeno v kapitole [3.1.2], Assembly objekt obsahuje dvě instance této třídy. Jedna z těchto instancí slouží pro data, která jsou cyklicky produkována zařízením do sítě a tvoří Input Assembly. Druhá z instancí je pro data, která jsou ze sítě konzumována a tvoří Output Assembly. Na první pohled se zdá, že pojmy input a output jsou zaměněné, ale není tomu tak. Označení input a output je z pohledu sítě. Tedy síť od zařízení dostává Input Assembly a ze sítě zařízení dostává Output Assembly.

AC Drive profil má předdefinované Input i Output Assemblies. Z tabulky 3.4 je vidět, že tento profil má předdefinovaných šest Input Assemblies a šest Output Assemblies a je vyžadováno podporovat minimálně jednu dvojici těchto sestav, a to sestavy s Instance ID 20 a 70. Dohromady tvoří základní sestavu dat pro řízení rychlosti pomocí cyklických dat.

Instance ID	Required/Optional	Type	Name
20	Required	Output	Basic Speed Control Output
21	Optional	Output	Extended Speed Control Output
22	Optional	Output	Speed and Torque Control Output
23	Optional	Output	Extended Speed and Torque Control Output
24	Optional	Output	Process Control Output
25	Optional	Output	Extended Process Control Output
70	Required	Input	Basic Speed Control Input
71	Optional	Input	Extended Speed Control Input
72	Optional	Input	Speed and Torque Control Input
73	Optional	Input	Extended Speed and Torque Control Input
74	Optional	Input	Process Control Input
75	Optional	Input	Extended Process Control Input

Tabulka 3.4: Instance třídy Assembly

Naše zařízení zatím podporuje jednu dvojici těchto instancí, a to dvojici 20-70 pro základní řízení rychlosti. Na následujícím obrázku 3.2 jsou ukázána data, která se v těchto sestavách posílají.

Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
20	0						Fault Reset		Run Fwd
	1								
	2	Speed Reference – Low Byte							
	3	Speed Reference – High Byte							
70	0						Running1		Faulted
	1								
	2	Speed Actual – Low Byte							
	3	Speed Actual – High Byte							

Obrázek 3.2: Data v Output a Input Assembly

Vysvětlení dat obsahující Assemblies na obrázku 3.2 je následující :

- **RunFwd** - řídicí bit pro roztočení motoru
- **FaultReset** - řídicí bit pro reset chyby v případě, že nastala
- **SpeedReference** - dva bajty obsahující referenci rychlosti
- **Faulted** - bit indikující chybu
- **Running1** - bit indikující roztočení motoru v jednom směru
- **SpeedActual** - dva bajty obsahující aktuální otáčky motoru

Všechna tato data, jsou atributy z aplikačních objektů daného profilu sjednocené do atributu Data v instancích objektu Assembly. Tabulka 3.5 mapuje data v Assemblies na atributy v aplikačních objektech, které budou popsány v následujících podkapitolách.

Data Component Name	Class Name	Class ID	Attribute Name	Attr. ID
Run Fwd	Control Supervisor	29_{HEX}	Run1	3
Fault Reset	Control Supervisor	29_{HEX}	FaultRst	12
Faulted	Control Supervisor	29_{HEX}	Faulted	10
Running1	Control Supervisor	29_{HEX}	Running1	7
Speed Reference	AC/DC Drive	$2A_{HEX}$	SpeedReference	8
Speed Actual	AC/DC Drive	$2A_{HEX}$	SpeedActual	7

Tabulka 3.5: Mapování dat v I/O Assembly na atributy objektů

3.3.2 Aplikační objekty profilu

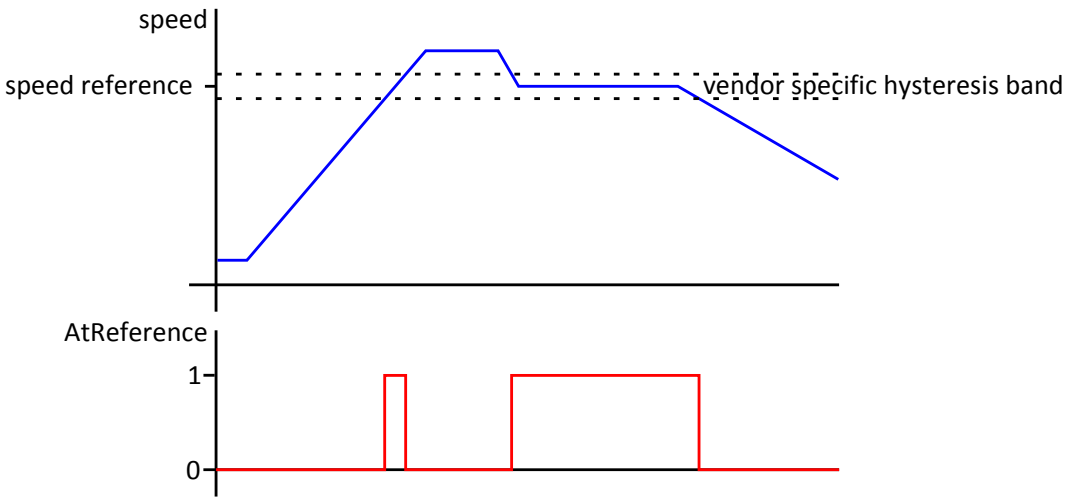
3.3.2.1 $2A_{HEX}$ AC/DC Drive object

AC/DC Drive objekt s Class ID $2A_{HEX}$ modeluje některé funkce motoru, jako například dobu náběhu a doběhu, řízenou veličinu a další. Tabulka 3.6 obsahuje přehled atributů této třídy.

Attr. ID	Access Rule	Attribute Name	Data Type	Units
3	GET	AtReference	BOOL	
4	GET/SET	NetRef	BOOL	
6	GET/SET	Drive Mode	USINT	
7	GET	SpeedActual	INT	[RPM / $2^{SpeedScale}$]
8	GET/SET	SpeedRef	INT	[RPM / $2^{SpeedScale}$]
9	GET	CurrentActual	INT	[100mA]
10	GET/SET	CurrentLimit	INT	[100mA]
15	GET	PowerActual	INT	[W]
16	GET	InputVoltage	INT	[V]
17	GET	OutputVoltage	INT	[V]
18	GET/SET	AccelTime	UINT	[ms]
19	GET/SET	DecelTime	UINT	[ms]
20	GET/SET	LowSpdLimit	INT	[RPM / $2^{SpeedScale}$]
21	GET/SET	HighSpdLimit	INT	[RPM / $2^{SpeedScale}$]
21	GET/SET	HighSpdLimit	INT	[RPM / $2^{SpeedScale}$]
22	GET/SET	SpeedScale	SINT	
29	GET	RefFromNet	BOOL	

Tabulka 3.6: AC/DC Drive Object

Atribut **AtReference** indikuje, kdy hodnota aktuální řízené veličiny (rychlost, moment nebo jiná), dosáhla reference nebo se dostala do hysterezního pásma. Hysterezní pásmo může být fixně určené výrobcem, programovatelné nebo nemusí být žádné, záleží na výrobci. Obrázek 3.3 názorně ukazuje, kdy je hodnota tohoto atributu v logické jedničce.



Obrázek 3.3: Atribut AtReference

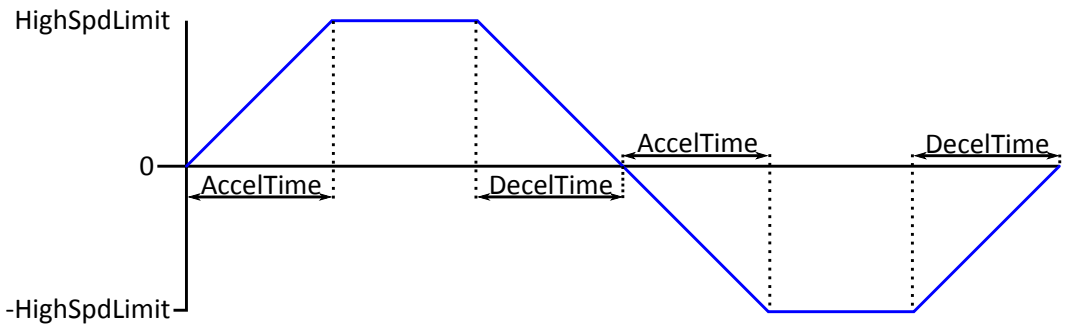
Atribut **NetRef** nastavuje, zda-li je motor řízen pomocí cyklických dat nebo se dá řídit atributy v aplikačních objektech pomocí explicitních zpráv a služeb Get/Set_Attribute_Single. V praxi to vypadá tak, že pokud pomocí explicitní zprávy nastavíme tento bit do logické jedničky, přepne se zdroj referenční hodnoty rychlosti z cyklických dat na hodnotu atributu SpeedRef v tomto objektu. Následně motor můžeme pomocí dalších atributů v Control Supervisor objektu řídit.

Atribut **DriveMode** specifikuje řízenou veličinu a způsob regulace. V tabulce 3.7 je popsáno, jaká hodnota atributu znamená jaký mód řízení.

Attribute Value	**Drive Mode**
0	Vendor specific mode
1	Open loop speed (Frequency)
2	Closed loop speed control
3	Torque control
4	Process control (e.g. PID)
5	Position control

Tabulka 3.7: Hodnoty atributu Drive Mode a jejich význam

Atributy **AccelTime** a **DecelTime** definují dobu náběhu a dobu doběhu motoru, jak ukazuje obrázek 3.4.



Obrázek 3.4: Atributy AccelTime a DecelTime

Hodnoty atributů **LowSpdLimit** a **HighSpdLimit** udávají dolní a horní mez rychlosti motoru. Rychlost motoru nepřekročí větší rychlost, než je hodnota atributu HighSpdLimit, i když je reference rychlosti vyšší.

Podíváme-li se na jednotky všech atributů mající co dočinění s rychlostí, můžeme si všimnout, že jejich hodnoty lze přeškálovat pomocí atributu **SpeedScale** za pomocí rovnice :

$$ScaledValue = RPM/2^{SpeedScale} \tag{3.1}$$

Je to dáno tím, že žádný atribut v CIP specifikaci nemá datový typ umožňující desetinná čísla. Všechny číselné datové typy, se kterými CIP specifikace pracuje, jsou pouze celočíselného formátu. Kdyby nebyla zavedena možnost přeškálovat hodnotu rychlosti a změnit tak jednotky, nebylo by možné točit motorem přesněji než na jednotky otáček za minutu. V tabulce 3.8 lze vidět, jaké jsou jednotky přeškálované rychlosti při různé hodnotě atributu SpeedScale. Například pro hodnotu atributu $SpeedScale = 2$ jsou jednotky rychlosti 0,25 [RPM]. To znamená, že při hodnotě atributu $SpeedScale = 2$ a $SpeedActual = 10$ je rychlost v jednotkách [RPM] rovna 2,5 otáčkám za minutu.

SpeedScale	**Unit**
3	0,125 [RPM]
2	0,25 [RPM]
1	0,5 [RPM]
0	1 [RPM]
-1	2 [RPM]
-2	4 [RPM]
-3	8 [RPM]

Tabulka 3.8: Přeškálované jednotky rychlosti

Schopnost škálování rychlosti je pro výrobce volitelná záležitost. Náš frekvenční měnič podporuje pouze škálování rychlosti, avšak specifikace uvádí další volitelné atributy pro škálování atributů s veličinami jako jsou čas, moment, napětí, proud a výkon.

Atribut **RefFromNet** je zrcadlem atributu NetRef a odráží jeho hodnotu. Narozdíl od atributu NetRef je atribut RefFromNet určen pouze pro čtení.

3.3.2.2 28_{HEX} Motor Data Object

Motor Data objekt s Class ID 28_{HEX} obsahuje informace o parametrech motoru.

Attr. ID	Access Rule	Attribute Name	Data Type	Units
3	GET	MotorType	USINT	
6	GET/SET	RatedCurrent	UINT	[100mA]
7	GET/SET	RatedVoltage	UINT	[V]
8	GET/SET	RatedPower	UDINT	[W]
9	GET/SET	RatedFreq	UINT	[Hz]
10	GET/SET	RatedTemp	UINT	[°C]
11	GET/SET	MaxSpeed	UINT	[RPM]
12	GET/SET	PoleCount	UINT	
13	GET/SET	TorqConstant	UDINT	[0,001 Nm/A]
14	GET/SET	Inertia	UDINT	[10^{-6} kg m^2]
15	GET/SET	BaseSpeed	UINT	[RPM]

Tabulka 3.9: Motor Data Object

Atribut **Motor Type** udává typ motoru. Běžně tento atribut podporuje i službu SET. Nastavení zařízení se provádí pomocí programu Simatic SCOUT, kde je jiné rozlišení motorů, a proto je tento atribut pouze ke čtení. Po nastavení motoru v Simatic SCOUT a nastavení AC Drive profilu se do atributu MotorType namapuje hodnota odpovídající specifikaci CIP.

3.3.2.3 29_{HEX} Control Supervisor Object

Control Supervisor objekt s Class ID 29_{HEX} je nejzajímavější z aplikačních objektů tohoto profilu. Tento objekt modeluje všechny potřebné funkce pro řízení motoru. Taktéž tento objekt předepisuje stavový diagram chování řízeného motoru 3.5. V tabulce 3.10 jsou uvedeny atributy tohoto objektu.

Attr. ID	Access Rule	Attribute Name	Data Type
3	SET	Run1	BOOL
4	SET	Run2	BOOL
5	SET	NetCtrl	BOOL
6	GET	State	USINT
7	GET	Running1	BOOL
8	GET	Running2	BOOL
9	GET	Ready	BOOL
10	GET	Faulted	BOOL
11	GET	Warning	BOOL
12	SET	FaultRst	BOOL
13	GET	FaultCode	UINT
14	GET	WarnCode	UINT
15	GET	CtrlFromNet	UINT

Tabulka 3.10: Control Supervisor Object

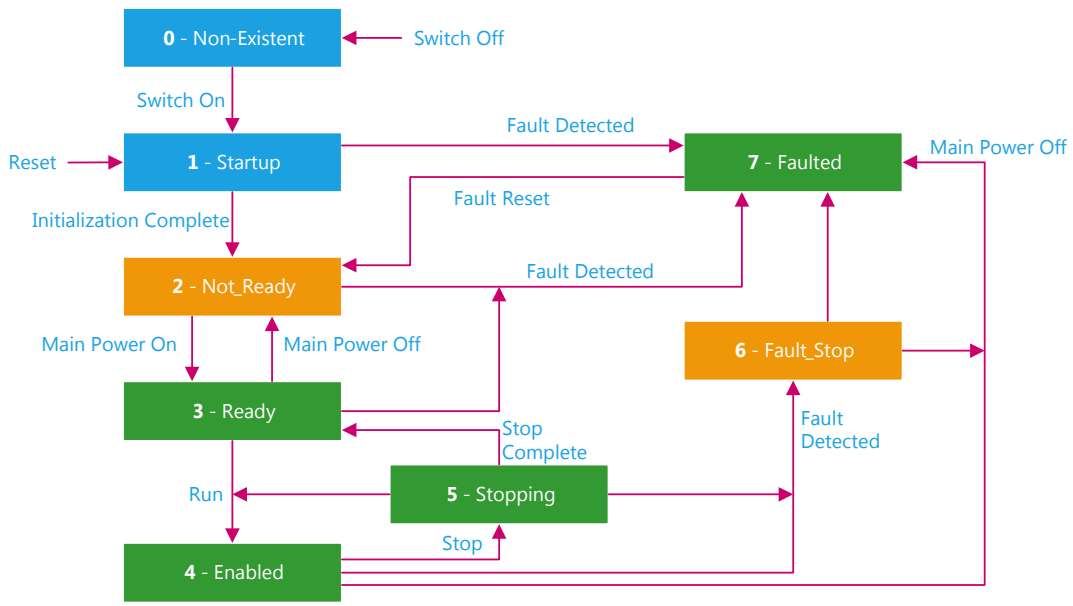
Atributy **Run1** a **Run2** spouští běh motoru a jak je uvedeno v kapitole [3.3.1] atribut Run1 je také součástí dat implicitních zpráv v Input Assembly. Run1 spouští běh motoru v směru dopředu a Run2 spouští běh motoru v směru dozadu. Podíváme-li se na data v Output Assembly 3.2, zjistíme, že první bajt obsahuje pouze RunFwd, což je atribut Run1, ale neobsahuje atribut Run2, pro zpětný běh motoru. Atribut Run2 například obsahuje až instance Output Assembly 21 pod názvem RunRev. I když instance Output Assembly 20 neobsahuje RunRev, je možné točit hřídelí motoru opačným směrem. Reference rychlosti v Output Assembly je spjat s atributem SpeedReference v Drive objektu s Class ID $2A_{HEX}$ a ten je datového typu INT, tedy je možné zapsat zápornou hodnotu rychlosti. Motor se tak točí směrem dopředu zápornými otáčkami. Tabulka 3.11 ukazuje, který typ běhu motoru je aktivní při nastavení Run1 nebo Run2 atributu.

Run1	**Run2**	**Run Type**
0	0	NA
$0 \rightarrow 1$	0	Run1
0	$0 \rightarrow 1$	Run2
$0 \rightarrow 1$	$0 \rightarrow 1$	NA
1	1	NA
$1 \rightarrow 0$	1	Run2
1	$1 \rightarrow 0$	Run1

Tabulka 3.11: Run1 and Run2

Nastavením atributu **NetCtrl** nebo atributu NetRef z Drive objektu s Class ID $2A_{HEX}$ na logickou jedničku můžeme řídit motor explicitními zprávami. Nastavením tohoto atributu, zařízení přestane reagovat na Output Assembly v implicitní komunikaci, hodnoty v Input Assembly jsou však vždy aktuální. Dále jsou nám atributy Run1, Run2, FaultRst a atribut SpeedRef z Drive objektu zpřístupněny pro zápis. Nyní můžeme pomocí explicitních zpráv nastavit referenci rychlosti a roztočit motorem jedním nebo druhým směrem. Po nastavení atributu NetCtrl na logickou nulu bude motor opět řízen pomocí implicitní komunikace.

Atribut **State** nám poskytuje informaci o stavu motoru, zda-li je připraven ke startu, či se točí nebo zastavuje. Všechny stavy, přechody a události vyvolávající přechod do druhého stavu zobrazuje obrázek 3.5. Po zapnutí napájení, se zařízení dostane přes stavy 1-Startup a 2-Not_Ready do stavu 3-Ready a je připraven pro spuštění motoru. Po jeho spuštění se stav dostane na 4-Enabled. Během brždění, či doběhu motoru, je ve stavu 5-Stopping. Nastane-li chyba během 5-Stopping stavu nebo během stavu 4-Enabled, motor je téměř okamžitě zastaven a během tohoto krátkého okamžiku brždění je ve stavu 6-Fault_Stop a poté spadne do stavu 7-Faulted. Po resetu chyby se motor opět dostane přes stav 2-Not_Ready do stavu 3-Ready.



Obrázek 3.5: Stavový diagram v AC Drive profilu

Atributy **Running1** a **Running2** indikují, zda je motor v pohybu. Running1 je v logické jedničce, v případě, kdy platí výrok :

$$(Enabled \wedge Run1) \vee (Stopping \wedge Running1) \vee (Fault\_Stop \wedge Running1). \tag{3.2}$$

Atribut Running2 je v logické jedničce, pokud platí výrok :

$$(Enabled \wedge Run2) \vee (Stopping \wedge Running2) \vee (Fault\_Stop \wedge Running2). \tag{3.3}$$

Atribut **Ready** má hodnotu logickou jedničku, pokud hodnota atributu State je 3-Ready, 4-Enabled nebo 5-Stopping. V logické nule se nachází hodnota tohoto atributu v případě jiného stavu.

Atribut **Warning** má hodnotu logickou jedničku, pokud zařízení hlásí varování. Číslo tohoto varování zjistíme z atributu **WarnCode**.

Atribut **Faulted** má hodnotu logickou jedničku, v případě, zda nastala chyba a atribut State má hodnotu 7-Fauled. Číslo této chyby můžeme vyčíst z atributu **FaultCode**. Přechodem z logické nuly do logické jedničky v atributu **FaultRst** vymažeme chybu a stav opět přejde do stavu 3-Ready.

V případě, že nastalo více chyb nebo zařízení hlásí více varovných zpráv, je na výrobci zařízení, která zpráva se v atributech WarnCode a FaultCode objeví.

Atribut **CtrlFromNet** pouze zrcadlí hodnotu atributu NetCtrl.

3.4 Profil Siemens

Vedle AC Drive profilu, který definuje ODVA, je v zařízení nadefinován vlastní profil, navržený výrobcem, a to Siemens profil. V tomto profilu jsou tři aplikační objekty. Stavový diagram zařízení v tomto profilu a cyklická komunikace koresponduje s profilem PROFIdrive [11].

3.4.1 Cyklická komunikace

V Siemens profilu Input a Output Assembly korespondují s telegramy, které definuje PROFIdrive. Výběr telegramu se provádí v programu Simatic SCOUT nebo lze telegram vybrat i za pomocí objektu Parameter. Pro účely testů postačí pouze telegram 1 a telegram 999. Pokud použijeme předem připravené EDS pro Siemens profil, můžeme počítat pouze s telegramem 1, jelikož EDS file je připraven pouze pro přenos čtyř bajtů. Na obrázku 3.6 je vidět, jaká data se v daných čtyřech bajtech přenáší.

	Instance	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
OUTPUT	101	0	STW1 – Low Byte							
		1	STW1 – High Byte							
		2	NSOLL_A – Low Byte							
		3	NSOLL_A – High Byte							
INPUT	102	0	ZSW1 – Low Byte							
		1	ZSW1 – High Byte							
		2	NIST_A – Low Byte							
		3	NIST_A – High Byte							

Obrázek 3.6: Input a Output Assembly v Siemens profilu

V PROFIdrive se v cyklické komunikaci přenáší tzv. PZD (**p**ro**z**ess**d**atenbereich), kde 1 PZD je o velikosti dvou bajtů. Některá PZD jsou předdefinovaná, jako STW1 (**st**euerwor**t**). STW1 je řídicí slovo a pomocí tohoto slova řídíme motor. Nazpátek dostáváme informaci ZSW1 (**zu**stands**w**ort), stavové slovo obsahující informace o stavu motoru. NSOLL_A je setpoint, tedy reference a NIST_A je aktuální hodnota. Tyto hodnoty jsou normalizované dle normy N2. Abychom dostali hodnotu rychlosti v jednotkách RPM, musíme tyto dvě hodnoty přepočítat dle vzorců :

$$SpeedReference = \frac{NSOLL\_A * MaxSpeed}{2^{14}}, \tag{3.4}$$

$$ActualSpeed = \frac{NIST\_A * MaxSpeed}{2^{14}}, \tag{3.5}$$

kde MaxSpeed je maximální rychlost motoru na štítku. Tato hodnota je taktéž v příslušném parametru v expertlistu nebo v Siemens Motor objektu.

Pokud chceme zvolit jiné assemblies, máme možnost v PLC přidat zařízení jako CIP Generic Device. Nakonfigurujeme IP adresu, instance assemblies a jejich příslušnou délku. Délka se udává v počtu PZD. Například pro telegam 999 Free BICO, nastavíme délku PZD na 12. První dvě PZD jsou stejná jako v telegramu 1. Zbytek můžeme nakonfigurovat libovolně pomocí BICO propojek v aplikaci Simatic SCOUT.

3.4.2 Aplikační objekty profilu

3.4.2.1 $32D_{HEX}$ Siemens Motor Data Object

Siemens Motor Data objekt s Class ID $32D_{HEX}$ obsahuje stejné atributy jako Motor Data objekt [3.3.2.2] v AC Drive profilu. K těmto atributům tento objekt obsahuje navíc jeden atribut, a to atribut Commisioning state.

Attr. ID	Access Rule	Attribute Name	Data Type
2	GET/SET	Commisioning state	UINT16

Tabulka 3.12: Atribut Commisioning state

Obecně pokud chceme v expertlistu v programu Simatic SCOUT měnit parametry motoru, je nutné přepnout parametr p10 Commissioning state do Quick commisioning módu. Jelikož Siemens profil je velice podobný PROFIdrive profilu, byl tento parametr zaveden jako atribut do tohoto objektu. Některé užitečné možnosti nastavení tohoto atributu jsou v následující tabulce.

Value	Commisioning state
0	Ready
1	Quick commissioning
2	Power unit commissioning
3	Motor commissioning

Tabulka 3.13: Vybrané hodnoty atributu Commisioning state

3.4.2.2 401_{HEX} Parameter Object

Objekt Parameter s Class ID 401_{HEX} umožňuje dostat se na jakýkoliv parametr v expertlistu, které jsou běžně dostupné uživateli skrz program Simatic SCOUT. Parametry zařízení v expertlistu mají svou adresu a subindex. Proto, abychom mohli vyčíst hodnotu parametru skrze tento objekt, zadáváme adresu parametru jako Instance ID a subindex parametru jako Attribut ID. Chceme-li například vyčíst hodnotu z parametru p300 se subindexem 3, budeme adresovat Parameter objekt s Class ID 401_{HEX}, instanci s ID 300 a atribut s ID 3. Datový typ takto získaného parametru je takový, jak jej definuje PROFIdrive [11].

3.4.2.3 $32C_{HEX}$ Siemens Drive Object

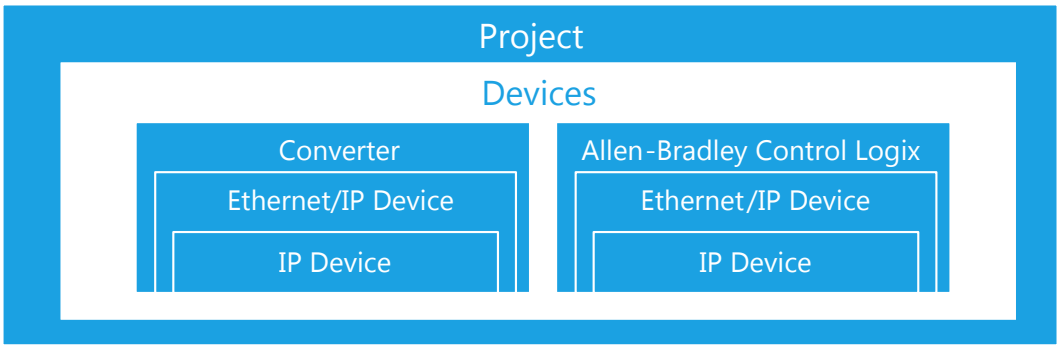
Siemens Drive Object obsahuje mnoho atributů týkající se nastavení zařízení, výstupy ze senzorů apod. Počet atributů je 83, z tohoto důvodu tabulku s atributy přikládám do přílohy [B].

Kapitola 4

PyTeMat

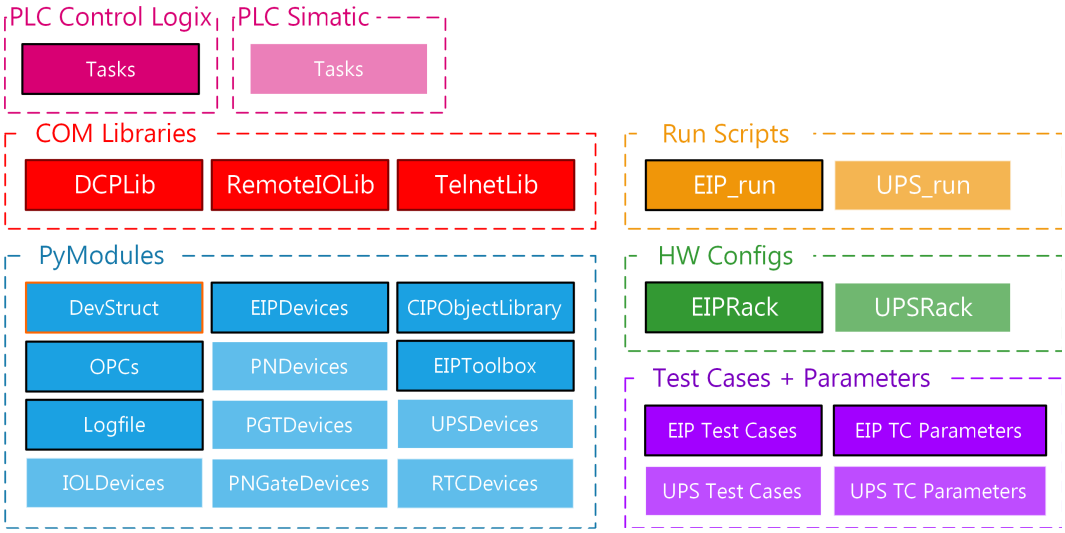
Python Test Automat, zkráceně PyTeMat, je vyvíjen v testlabu společnosti Siemens, s.r.o. a jedná se o strukturu tříd, funkcí a datových struktur, které napomáhají snadnému psaní testů. Právě snadné psaní testů je velkou motivací, jelikož po testerovi již nikdo netestuje a je velmi důležité, aby se tester nedopouštěl chyb. Programovací jazyk Python ve verzi 2.7.3 byl zvolen z důvodů, že se jedná o objektově orientovaný skriptovací jazyk, který je jednoduchý, přehledný, s rozsáhlými možnostmi a tím splňuje požadavky pro tvorbu testovacího prostředí.

Každé zařízení se dá rozvrstvit do několika částí, které mohou být společné napříč všemi zařízeními. Máme-li pak více zařízení se stejnými vlastnostmi či funkcemi, můžeme si tak snadno vytvořit operace pracující s těmito společnými vlastnostmi a funkcemi. Základní myšlenku této objektové struktury znázorňuje obrázek 4.1. Navrchu celé struktury je projekt, který obsahuje kolekci zařízení. Tato kolekce obsahuje zařízení a jejich připojení. Každé zařízení můžeme rozvrstvit, kdy jsme jako nejnižší vrstvu zvolili IP zařízení. Nad touto vrstvou je vrstva komunikačního protokolu, na kterém dané zařízení komunikuje. A navrchu je již konkrétní zařízení. Tato základní struktura a mnohé další jsou naimplementovány v kolekci modulů, které dohromady tvoří PyModules.



Obrázek 4.1: Model a myšlenka PyTeMatu

PyModules, které poskytují objektové struktury a funkce, jsou součástí většího celku PyTeMatu. Další části PyTeMatu jsou COM knihovny, úlohy v kontroléru, spouštěcí skripty a samotné testy (viz. obrázek 4.2). Moduly, které jsou výraznější než ostatní, jsou využity pro testy našeho frekvenčního měniče. Moduly jenž mají černý rámeček byly vyvinuty v rámci této práce a na modulu Dev_Struct.py se tato práce výrazně podílela.



Obrázek 4.2: Model a myšlenka PyTeMatu

4.1 Spouštěcí skript

Každý projekt má svůj spouštěcí skript, v našem případě EIP_run.py. Spouštěcí skript vykonává postupně následující :

1. Naimportuje potřebné moduly, které poskytuje Python (sys, time, ctypes, atd.) a moduly z PyModules.
2. Vytvoří logovací soubor.
3. Spustí skript pro vytvoření projektu.
4. Spustí skript s parametry testu.
5. Spustí test.

Spouštěcí skript obsahuje příkazy pro výkon všech testů. Konkrétní test a jeho parametry, který chceme spustit vybíráme formou zrušení komentovacího znaku před příkazem pro spuštění testu. Odkomentujeme-li více testů, vykonají se v pořadí za sebou a spouštíme tak celou dávku.

4.2 HW configs

Jedná se o skripty, které realizují obrázek 4.1, tedy vytváří instanci Projektu obsahující kolekci Devices a do této kolekce vkládáme instance tříd konkrétních zařízení, které vytváříme v těchto skriptech. Nakonec ukládáme do kolekce Devices informace o topologii. Pro testy našeho zařízení je skript pojmenován EIP_rack.py. Kde se nachází třída Project a třídy realizující zařízení bude popsáno v kapitole PyModules.

4.3 COM knihovny

Součástí PyTeMatu jsou i COM knihovny, které byly vyvinuty dříve a používají se v testech napříč různými projekty v testlabu. Poskytují obecnou funkcionalitu a jsou nezbytné pro základní testy. Funkce těchto knihoven jsou zapouzdřeny do tříd a funkcí v Pythonu.

4.3.1 RemoteIOLib.dll

Pomocí této knihovny ovládáme vzdálené I/O zařízení, které je připojeno do testovací sítě. Zařízení má několik vstupů a výstupů pro čtení či zapisování logické jedničky nebo nuly. Pomocí výstupů tak můžeme ovládat Ethernet Breaker zařízení a spínat či rozepínat relátka, která umožňují rozpínat napájení testovaných zařízení pro účely testování.

4.3.2 DCPlib.dll

DCP (**D**iscovery and basic **C**onfiguration **P**rotocol) je protokol definovaný uvnitř PROFINETu. Skrze tento protokol můžeme objevovat zařízení na síti a konfigurovat jejich síťové nastavení. Tato COM knihovna nám umožňuje vysílat na síť rámce definované tímto protokolem.

4.3.3 TelnetLib.dll

Telnet je aplikační protokol typu klient-server pracující nad TCP/IP vrstvou. Naše zařízení tento protokol též podporuje a dají se skrze něj vyčítat a zapisovat hodnoty parametrů. Tato funkcionalita je určena pouze pro účely vývoje a testování a není uživateli běžně dostupná. Knihovna nám poskytuje funkce pro vyčtení nebo zápis parametrů z expertlistu a pro zjištění, zda-li zařízení má navázán aplikační vztah s kontrolérem.

4.4 Úlohy v PLC

V každé HW konfiguraci pro testování je zapotřebí kontroléru. V našem případě je kontrolérem PLC Allen-Bradley ControlLogix. Kontrolér nám umožňuje řídit motor za pomocí cyklických dat a po implementaci následujících úloh, také poskytuje různé funkce nejen pro řízení motoru, ale také pro vysílání explicitních zpráv se službami Get_Attribute_Single nebo Set_Attribute_Single a další. Tyto úlohy jsou spouštěny skrze OPC v objektové struktuře PyTeMat. Všechny úlohy v PLC jsou psané v jazyce STL a jsou vytvořené tak, aby byly uživatelsky přívětivé i pro manuální užívání přímo z PLC. Soupis všech úloh je v následující tabulce 4.1.

Task	**Type**
ODVA_MotionTask	Periodic
Siemens_MotionTask	Periodic
MSG_GetSingleAttribute	Continuous
MSG_SetSingleAttribute	Continuous
WaitForStateODVA	Continuous
InputData	Continuous
InputAssembly	Continuous

Tabulka 4.1: Úlohy v PLC

4.4.1 ODVA_MotionTask

Jedná se o periodickou úlohu mající za úkol ovládat motor pomocí cyklických dat, která si uživatel snadno předem nakonfiguruje. Pro přehlednost a dobrou konfigurovatelnost byl vytvořen tag s datovou strukturou Assembly (tabulka 4.2), obsahující všechny atributy z AC Drive profilem definovaných instancí Assembly objektu. Tyto atributy se na základě InstanceNo nakopírují do výstupního bufferu kontroléru a posílají se cyklicky v rámci aplikačního vztahu cílovému zařízení. Pro spuštění zápisu do výstupního bufferu je třeba nastavit tag RunTask do logické jedničky. Po tomto startu jsou taktéž periodicky aktualizována data v Input Assembly struktuře ze vstupního bufferu.

Assembly.	INPUT.	InstanceNo
		Faulted
		Warning
		Running1
		Running2
		Ready
		CtrlFromNet
		RefFromNet
		AtReference
		DriveState
		LB_SpeedActual
		HB_SpeedActual
		LB_ProcessActual
		HB_ProcessActual
		LB_TorqueActual
		HB_TorqueActual
	OUTPUT.	InstanceNo
		RunFwd
		RunRev
		FaultReset
		NetCtrl
		NetRef
		NetProc
		Mode
		LB_SpeedReference
		HB_SpeedReference
		LB_ProcessReference
		HB_ProcessReference
		LB_TorqueReference
		HB_TorqueReference

Tabulka 4.2: Struktura Asembly Tagu

Náš frekvenční měnič sice zatím podporuje jen instance Assembly objektu 20 a 70, ale v budoucnu se to může změnit a tato úloha je na to připravena.

Chceme-li například roztočit motor otáčkami o rychlosti 100 [RPM], musíme do následujících tagů nastavit hodnoty :

Tag	Value
Assembly.INPUT.InstanceNo	70
Assembly.OUTPUT.InstanceNo	20
Assembly.OUTPUT.RunFwd	True
Assembly.OUTPUT.LB_SpeedReference	64_{HEX}
RunTask	True

Tabulka 4.3: Příklad roztočení motoru

Vyplněním InstanceNo, tento task bude vědět, která data kopírovat na výstup pro cyklický přenos. RunFwd spouští motor a bude se točit rychlostí hodnoty SpeedReference = 64_{HEX}, což je námi požadovaných 100 [RPM].

4.4.2 Siemens_MotionTask

Tento task umožňuje spuštění motoru v Siemens profilu. Nejprve je nutné nastavit tagy STW1 a NSOLL_A. Hodnota STW1 je pro rozběh motoru rovna $47f_{HEX}$ a pro zastavení $4fe_{HEX}$.Hodnoty řídicího slova STW1 vychází ze stavového diagramu, jak jej definuje PROFIdrive [11]. NSOLL_A je reference rychlosti, normovaná dle normy N2 (vzorec 3.4). Pro spuštění úlohy je třeba nastavit RunTask do logické jedničky. Nastavením RunTask tagu na logickou nulu, zastavíme motor (v Output Assembly se zapíše do STW1 hodnota $4fe_{HEX}$).

4.4.3 MSG_GetSingleAttribute a MSG_SetSingleAttribute

Tyto dvě úlohy jsou si velmi podobné. Obě využívají tagy datového typu MESSAGE a základního STL příkazu MSG(MESSAGE) pro vyslání explicitní zprávy. Struktura datového typu MESSAGE obsahuje mnoho informací, nejdůležitější jsou však :

- IP adresa cílového zařízení
- Maska podsítě cílového zařízení
- Protokol - V našem případě CIP
- Služba - V našem případě Get_Attribute_Single nebo Set_Attribute_Single
- Buffer - Pokud je služba nastavena na Get_Attribute_Single, tak se jedná o buffer (pole bajtů), kam přijmeme data atributu na která jsme se dotázali a v případě služby Set_Attribute_Single se jedná o buffer dat, s novou hodnotou atributu do kterého budeme chtít zapsat.

- Class ID cílového objektu
- Instance ID cílového objektu
- Attribute ID cílového atributu
- Informace o chybě - Nastala-li během přenosu (vypršení časového limitu) nebo pokud posílá zařízení chybu jako odezvu (například kdy cílový atribut neexistuje nebo objekt neposkytuje danou službu nad cílovým atributem).

Vytvořené tagy s datovým typem MESSAGE mohou býti umístěné pouze v tabulce tagů kontroléru a jedná se tedy o globální tagy. Vytvořené tagy s tímto datovým typem jsou dva. Jeden se službou Get_Attribute_Single a druhý se službou Set_Attribute_Single. V tabulce tagů obou úloh je vytvořen tag 4.4 se strukturou pro snadné zasílání explicitních zpráv s příslušnou službou. Po vyplnění hodnot této struktury a nastavením tagu RunTask na logickou jedničku, odešleme příslušnou explicitní zprávu.

Attribute.	Class ID
	Instance ID
	Attribute ID
	Connected
	DataLength
	Data
	Error
	Error No.
	OtherIPAddress
	Delay
	Done

Tabulka 4.4: Struktura tagu Attribute

Tagem Connected nastavujeme, zda-li se bude jednat o explicitní zprávu spojovanou či nespojovanou [2.1.2.2]. Data jsou přijímána nebo vysílána jako bajtové pole. Datovou konverzi zajišťjeme v příslušných funkcí tříd PyTeMatu. Nastane-li error, je detekován ve stejnojmenném tagu struktury Attributue. Také je vypsáno číslo chyby. Význam této chyby je dohledatelný v helpu programu RSLogix. Potřebujeme-li vyslat explicitní zprávu na zařízení s jinou IP adresu, je třeba vyplnit tag OtherIPAddress. Pokud tento atribut není vyplněn, jsou explicitní zprávy posílané na předem nastavenou IP adresu zařízení. Po přijetí zprávy je pro informaci vypočteno zpoždění v počtech cyklů PLC.

4.4.4 WaitForStateODVA

Po nastavení tagu RunTask do logické jedničky, úloha cyklicky vysílá explicitní zprávu se službou Get_Attribute_Single na atribut State v Control Supervisor objektu a porovnává jí se zadanou hodnotou stavu (tag State). Nastane-li očekávaný stav, úloha se přestane na tento atribut dotazovat a nastaví tag Done na logickou jedničku.

4.4.5 InputData

Tato úloha se používá pouze v případě, je-li zařízení v Siemens profilu. Po nastavení tagu RunTask na logickou jedničku, se do struktury InputData (tab. 4.5) nahrají aktuální data z Input Assembly.

InputData.	LB_ZSW1
	HB_ZSW1
	LB_NSOLL_A
	HB_NSOLL_A

Tabulka 4.5: Struktura tagu InputData

4.4.6 InputAssembly

Tento task se používá jen pokud je zařízení v AC drive profilu. Po nastavení tagů InstanceNo na příslušné Instance ID objektu Input Assembly a RunTask na logickou jedničku, se do struktury InputAssembly nahrají všechna data, která daná insance Input Assembly obsahuje. Tag do kterého se nahrávají data má stejnou strukturu jako je InputAssembly v tabulce 4.2.

4.5 PyModules

PyModules poskytuje objektové struktury všech možných zařízení a kontrolérů, které se při testech používají. Dále například obsahuje modul pro dokumentaci průběhu testu a další užitečné objekty, strukury a funkce. V následujících kapitolách si představíme základní objektovou strukturu a důležité moduly PyTeMatu, jejichž implementace byla jednou z hlavních částí praktické části.

4.5.1 DevStruct.py

Hlavním modulem a srdcem PyModules je DevStruct.py. Obsahuje třídy pro vytvoření testovacího prostředí a základní třídy zařízení, které následně dědí konkrétní typy zařízení.

IPDevice je základní třídou, jelikož každé zařízení podporuje IP protokol. Vlastnosti této třídy jsou následující :

- *Name* [STRING]
- *MAC* [STRING]
- *IpAddress* [STRING]
- *SubNetMask* [STRING]
- *GateWay* [STRING]
- *PowerSwitchAddress* [STRING] - Adresa I/O zařízení pro vypnutí napájení. Adresa je ve tvaru 'IPadresa:Port'.
- *PcConnect* [BOOL] - Tato promměnná indikuje, zda-li nekterý z portů je připojen k PC.
- *Cotnroller* [BOOL] - Tato proměnná indikuje, zda-li zařízení je kontrolér.
- *Ports* [Ports] - Tato proměnná obsahuje instanci třídy Ports a jedná se o jednotlivé fyzické síťové porty, které zařízení obsahuje. Vlastnosti této třídy budou popsány později.

Funkce třídy IPDevice jsou následující :

- *Ping(repetition=3)* - Tato funkce vyšle příkaz ping na IP adresu daného zařízení a vrátí True nebo False. Je důležité, aby vlastnost IpAddress tohoto zařízení byla vyplněna.
- *PowerOFF()* - Tato funkce vypne napájení zařízení. Metoda využívá COM knihovny RemoteIOLib.dll a je nutné aby vlastnost PowerSwitchAddress byla vyplněna.
- *PowerON()* - Funkce opět zapne napájení zařízení.

Instance třídy **Ports** je vlastností třídy IPDevice. Tato třída je slovníkem obsahující navíc vlastnosti jako :

- *Neighbour* [Ports] - Port dalšího zařízení, se kterým je tento port propojen kabelem.
- *EthernetBreakerAddress* [STRING] - Obsahuje adresu ve tvaru 'IPadresa:Port', kde IP adresa je adresa I/O zařízení a číslo portu, na kterém je připojeno zařízení Ethernet Breaker pro rozpojení kabelu mezi porty.

- *Activity* [BOOL] - Tato vlastnost indikuje, zda-li kabel je mezi porty rozpojen (Activity=False) nebo spojen (Activity=True)

Potomky třídy IPDevice jsou třídy PNDevice a EIPDevice. PNDevice modeluje obecné zařízení komunikující na síti PROFINET a obsahuje dle toho příslušné vlastnosti a funkce, jako například DCP. Nás ale více zajímá třída EIPDevice.

Třída **EIPDevice** je potomkem třídy IPDevice a rozšiřuje tuto třídu pouze o vlastnosti DCP. Může se zdát, že je nesmyslné, aby zařízení komunikující na síti Ethernet/IP podporovalo protokol DCP, jelikož je to jeden z protokolů PROFINETu, ale musíme si uvědomit, že naše zařízení umí též komunikovat na síti PROFINET a podpora protokolu DCP na síti Ethernet/IP je rozšiřující vlastností tohoto zařízení.

Třída **DCPProperties** využívá knihovny DCPLib.dll a zapouzdřuje její funkce do této třídy. Funkce této třídy jsou následující :

- *ResetToFactorySettings()* - Tato funcke nastaví síťové nastavení do výrobního nastavení, tedy smaže device name zařízení a nastaví IP adresu, masku podsítě a bránu na samé nuly.
- *SetDHCP()* - Tato funkce řekne zařízení, aby IP adresu převzal z DHCP serveru.
- *SetIP(ipAddr=None, mask=None, gate=None, permanent=True)* - Tato funkce nastaví pomocí DCP protokolu nové síťové nastavení zařízení (IP adresu, masku podsítě a bránu). Nepředáme-li této funkci žádné nové parametry síťového nastavení, funkce nastaví v zařízení síťové nastavení takové, jaké má uložené ve vlastnostech zařízení. Pokud nenastavíme vstupní parametr permanent=True, nové síťové nastavení bude v zařízení nastaveno permanentně, tzn. že tato IP adresa bude nastavena v zařízení i po jeho restartu.
- *SetDeviceName(devName, permanent=True)* - Funkce nastaví nové jméno zařízení permanentně nebo dočasně na základě vstupního parametru permanent.

EthernetSwitch je další třída popisující základní zařízení. Tato třída má pouze základní vlastnosti jako jméno (*Name*) a porty (*Ports*). Při vytváření instance této třídy je vstupním parametrem počet portů.

Třída **Devices** dědí datový typ slovník a do instance této třídy budeme přidávat veškerá zařízení, která si vytvoříme. Tato třída obsahuje základní funkce slovníku (Add, Remove, Contains, atd.), dále je rozšířena o funkce, které dovolují pracovat se všemi obsahujícími zařízeními. Navíc tento objekt vytváří a uchovává propojení mezi zařízeními. Na základě těchto propojení pak tato třída ví, která zařízení jsou dostupná a která nejsou a má celkový přehled o topologii. Funkce této třídy jsou :

- *Add(device)* - Přidává zařízení do slovníku. Jméno zařízení musí být unikátní.

- *Remove(device)* - Odebírá konkrétní zařízení ze slovníku.
- *Ping()* - Funkce prověří, zda-li všechna zařízení, která obsahuje, jsou na síti dostupná. Funkce vrací list nedostupných zařízení.
- *PowerOFF()* - Vypne všechna zařízení.
- *PowerON()* - Zapne všechna zařízení.
- *GetActive()* - Vrátí list obsahující všechna zapnutá zařízení.
- *GetInActive()* - Vrátí list obsahující všechna vypnutá zařízení.
- *AddConnection(name_port1, name_port2, eth_break_adrress)* - Vytvoří připojení mezi dvěma zařízeními na konkrétních portech. Připojení též obsahuje adresu a port I/O zařízení pro rozpojení tohoto připojení. Vstupní parameter name_port je pole obsahující jméno zařízení a číslo portu.
- *GetReachable()* - Vrátí list zařízení, která nemají mezi sebou rozpojená připojení a mají odezvu na ping.
- *GetUnReachable()* - Vrátí list zařízení, která mají mezi sebou rozpojená připojení a nemají odezvu na ping.

Instanci třídy Devices vytváří třída **Project**, která kromě slovníku Devices, kam přidáváme veškeré instance zařízení v konfiguraci testu, obsahuje i instanci třídy **PGPC**, kterou objekt Devices obsahují. Tato třída PGPC reprezentuje počítač, který je též připojen do HW konfigurace. PGPC obsahuje pouze instanci třídy DCPProperties, čili skrze připojený počítač můžeme nastavit pomocí DCP protokolu, jakémukoliv zařízení v konfiguraci, jeho síťové nastavení. Třída Project obsahuje jednu funkci, a to funkci *CheckConfiguration()*, která zkontroluje HW zapojení projektu s reálným zapojením. K tomu využívá funkce Ping, PowerOn, PowerOFF a další.

4.5.2 EIP_Devices.py

Tento modul obsahuje třídy již konkrétních zařízení, které komunikují na síti Ethernet/IP a tyto třídy jsou potomkem třídy EIPDevice z modulu Dev_Struct.py.

Třída **Converter** představuje obecné zařízení typu frekvenční měnič, komunikující na síti Ethernet/IP. Tuto třídu rozšiřujeme o instanci třídy Telnet. Třída Telnet se nachází v modulu PN_Devices.py a využívá COM knihovnu TelnetLib.dll. Třída Telnet, o jejíž funkce roširujeme tuto třídu, obsahuje následující funkce :

- *GetArConnect()* - Funkce se přes Telnet zeptá zařízení, zda-li má navázaný aplikační vztah s kontrolérem. Návratová hodnota je pak True nebo False.
- *ReadParameterAsDecimal(deviceID, paramID, subindex, length)* - Funkce vyčte konkrétní číselnou hodnotu nebo hodnoty parametru v expertlistu zařízení. Vstupními parametry je deviceID (ID zařízení), paramID (ID konkrétního parametru v expertlistu), subindex (subindex daného parametru) a length (počet hodnot v subindexech parametru, které od zadaného subindexu má funkce vrátit). Například pro vyčtení hodnot v parametru p300[2 až 4] je paramID=300, subindex=2 a length=3. Pokud length je větší jak 1, tak návratová hodnota je pole hodnot parametru.
- *WriteParameterAsDecimal(deviceID, paramID, subindex, value)* - Funkce zapíše číselnou hodnotu nebo hodnoty (pak předávaný parametr value je pole číselných hodnot), které se mají do konkrétního parametru zapsat.
- *ReadParameterAsString(deviceID, paramID, subindex, length)* - Funkce slouží pro vyčítání parametrů s datovým typem STRING.
- *WriteParameterAsString(deviceID, paramID, subindex, value)* - Funkce zapíše zadaný textový řetězec do parametru.
- *ReadAttribute(atributte)* - Některé hodnoty atributů v objektech definovaných specifikací [1], se dají vyčíst i z konkrétních parametrů expertlistu. Datový typ parametru se však velmi často liší oproti datovému typu atributu. Vstupní parametr této funkce je atribut z modulu CIPObjectLibrary.py. Atribut z tohoto modulu obsahuje informace, s kterým parametrem v expertlistu souvisí a jak jej překonvertovat na hodnotu či datový typ specifikovaný tímto atributem. Pomocí této funkce se můžeme například přesvědčit, zda-li hodnota atributu vyčtená skrze explicitní zprávu, odpovídá hodnotě souvisejícího parametru v expertlistu.

Další důležitou třídou, kterou modul EIP_Devices.py obsahuje je **ABControlLogix**. Tato třída reprezentuje PLC, které je použito jako kontrolér v našich testech. Tato třída obsahuje instanci třídy ABControlLogixOPC z modulu OPCs.py. Funkce této třídy jsou popsány v další kapitole.

4.5.3 OPCs.py

Tento modul obsahuje dvě třídy, které zpřístupňují PyTeMatu funkce a tagy kontrolérů. První třída je SimaticOPC a druhá, které se budeme věnovat je ABControlLogixOPC. Obě třídy dědí volně dostupný modul OpenOPC [12]. OpenOPC modul poskytuje základní funkce

k připojení k OPC serveru a vyčítání či zapisování hodnot. Třída **ABControlLogix** obsahuje následující parametry, které je nutno zadat.

- *OpcServerName* [STRING] - Jméno serveru, ke kterému se budeme připojovat.
- *OpcTopicName* [STRING] - Jméno topicu, který je na serveru vytvořen a je součástí cesty k připojení klienta k OPC.

Jelikož se nyní můžeme připojit k danému PLC přes OPC, zpřístupnili se nám naše funkce naprogramované v PLC, jak je popsáno v kapitole [4.4]. Třída tedy obsahuje tyto funkce :

- *GoOnline()* - Funkce vytvoří připojení k OPC serveru a zkusí vyčíst testovací hodnota.
- *GoOffline()* - Zavře vytvořené připojení k OPC serveru.
- *GetInputData()* - Funkce spustí úlohu InputData [4.4.5] v PLC a vrátí strukturu popsanou v tabulce 4.6, kde ActualSpeed je již přepočtená hodnota dle vzorce 3.5.

InputData.	ZSW1
	NIST_A
	ActualSpeed

Tabulka 4.6: Návratová struktura funkce GetInputData()

- *GetInputAssembly()* - Funkce spustí úlohu InputAssembly [4.4.6] a vrátí strukturu popsanou v tabulce 4.7. Na základě InstanceNo jsou vyplněna příslušná data.

InputAssembly.	InstanceNo
	Faulted
	Warning
	Running1
	Running2
	Ready
	CtrlFromNet
	AtReference
	DriveState
	LB_SpeedActual
	HB_SpeedActual
	SpeedActual
	LB_ProcessActual
	HB_ProcessActual
	ProcessActual
	LB_TorqueActual
	HB_TorqueActual
	TorqueActual

Tabulka 4.7: Návratová struktura funkce GetInputAssembly()

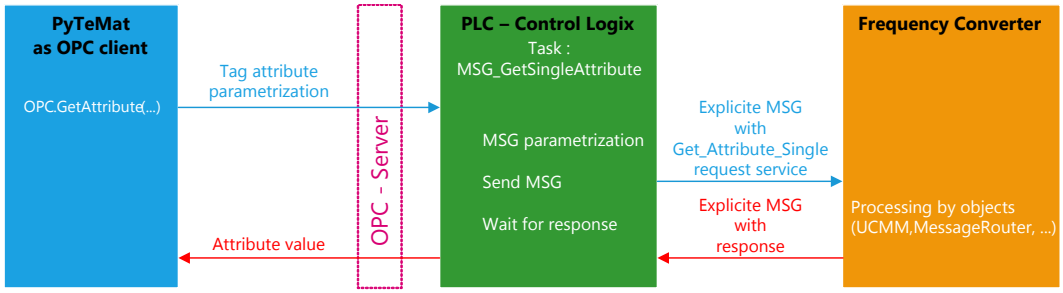
- *WaitForState(State, Timeout=5, Exception=False)* - Funkce spustí úlohu WaitForStateODVA [4.4.4] v PLC a čeká po dobu Timeout [s], zda-li nastane v zařízení daný stav definovaný atributem State v Control Supervisor objektu [3.3.2.3]. Nenastane-li a parametr Exception=True, funkce vyhodí vyjímku a zastaví běh programu.

- *StartStopDrive(Run, SpeedReference=0, Profile='ODVA', InputInstanceNo=70, OutputInstanceNo=20)* - Funkce na základě parametru Run (True/False), zastaví nebo roztočí motor příslušnými otáčkami. Je-li frekvenční měnič v AC Drive profilu, vstupní hodnotou Profile může být 'ODVA' nebo číselná hodnota 1. V tomto případě funkce využije úlohy ODVA_MotionTask [4.4.1]. Je-li však frekvenční měnič v profilu 'Siemens' (nebo Profile=0), funkce využije Siemens_MotionTask [4.4.2] v PLC k roztočení motoru. V tomto případě také není nutné zadávat čísla vstupní a výstupní instance.

- *ResetFaults(Profile)* - Nastala-li chyba v zařízení, pro opětovné roztočení motoru je třeba tuto chybu potvrdit. Je-li frekvenční měnič v profilu AC Drive (Profile='ODVA' nebo Profile=1), nastaví se příslušný atribut FaultReset v Output Assembly na True. Poté, co kontrolér dostane nazpátek v Input Assembly atribut Faulted=False,

shodí ResetFault opět na False. Je-li však frekvenční měnič v profilu Siemens (Profile='Siemens' nebo Profile=0), zapíše se postupně do PZD STW1 cyklických dat hodnoty STW1=4fe$_{HEX}$, poté STW1=47f$_{HEX}$. První hodnota nejprve nastaví spouštěcí bit kontrolního slova na logickou nulu a bit pro reset chyby na logickou jedničku. Po potvrzení chyby, se opět zapíše hodnota kontrolního slova pro spuštění motoru.

- *GetAttribute(Attribute, Connected=True , OtherIPAddress=", Exception=True)* - Tato funkce na základě parametrů Attribute (jedná se o datový typ atribut z modulu CIPObjectLibrary.py), Connected a OtherIPAddress, nastaví příslušný tag typu MESSAGE v PLC funkci MSG_GetSingleAttribue [4.4.3], odstartuje tuto funkci, jenž vyšle danou explicitní zprávu a čeká na odpověď. Nastala-li chyba a parametr Exception=True, bude vyvolána vyjímka a zastaví se běh programu. Jelikož přijatá data v PLC je pole bajtů, je třeba překonvertovat toto bajtové pole do příslušného datového typu atributu. K tomuto účelu je využíván modul ctypes [13], který obsahuje a umí základní datové typy, jak jsou nadefinované v jazyce C. Na obrázku 4.3 lze vidět, jak probíhá vyčítání hodnoty atributu. Jsme-li připojeni k OPC serveru, tato funkce naparametrizuje tag Attribue (4.4) a dá pokyn k vykonání úlohy MSG_GetSingleAttribute [4.4.3] v PLC. Úloha naparametrizuje strukturu MESSAGE a vyšle explicitní zprávu se službou Get_Attribute_Single. Zpráva je zpracována příslušnými objekty v zařízení (UCMM, MessageRouter, atd.) a vyšle nazpátek odpověď s hodnotou atributu, na který jsme se dotazovali. Hodnota je uložena v tagu Attribute a taktéž je nastaven tag Done indikující získanou odpověď. Hodnota atributu je následně vyčtena skrze OPC touto funkcí a předána jako návratová hodnota.



Obrázek 4.3: Průběh funkce GetAttribute

- *SetAttribute(Attribute, Value, Connected=True, OtherIPAddress=", Exception=True)* - Funkce na základě vstupních parametrů nastaví příslušný tag a pomocí funkce MSG_SetSingleAttribue [4.4.3] v PLC vyšle explicitní zprávu s příslušnou službou pro nastavení hodnoty atributu. Vstupní parametr Attribute je datového typu atribut z CIPObjectLibrary.py. Návratová hodnota je True nebo False v případě chyby.

4.5.4 Logfile.py

Jednou z nejdůležitějších částí testování je vytváření si záznamu o průběhu testu do textového či jiného souboru. V případě chyby tak snadno odhalíme, v kterém kroku se chyba vyskytla. Dále nám tento záznam, tzv. *log*, pomůže při vytváření reportu chyby. K tomuto účelu slouží modul **Logfile.py**. Modul obsahuje pouze jednu třídu stejného názvu jako modul - *Logfile*. Instance této třídy se vytváří na počátku každého testu a platí interní konvence, že tato instance má název **LOG**. Tuto konvenci je nutné dodržovat, jelikož téměř všechny funkce napříč všemi moduly potřebují zaznamenávat svůj průběh či stav a ktomu využívají funkce této instance. Tato instance LOG se nepředává jako parametr žádné funkci, ale na začátku každého modulu se importuje, jakožto globální proměnná. Třída Logfile má jediný vstupní parametr při vytváření instance, a to cestu a název textového souboru, který vytvoří a do nějž bude zaznamenávat pomocí funkcí, které poskytuje, průběh celého testu. Třída logfile poskytuje následující funkce :

- *InfoMessage(*logTexts)* - Funkce může přijmout pole textových řetězců, nebo jediný textový řetězec. Zprávě přidá časovou známku, zapíše do souboru a vytiskne na obrazovku.
- *FaultMessage(*logTexts, Exception=True)* - Tato funkce stejně jako InfoMessage zapíše záznam s časovou známkou do souboru, avšak tuto zprávu zvýrazní. Druhým parametrem je Exception, který je v základu nastaven na hodnotu True. Pokud hodnota tohoto parametru je True, funkce po zápisu do souboru vyhodí vyjímku a zastaví běh programu. Zastavení testu v době chyby je dobré v případech, kdy potřebujeme mít zařízení v chybovém stavu pro účely další analýzy.
- *Evaluate()* - Tato funkce je vždy na konci testu, kde se vypíšou veškeré chyby, které nastaly v průběhu testu.

4.5.5 CIPObjectLibrary.py

CIPObjectLibrary je knihovna všech objektů a jejich atributů, které podporuje náš frekvenční měnič. Modul obsahuje dvě privátní[1] třídy. První třída charakterizuje *Objekt*, jak jej známe ze specifikace CIP a druhá třída charakterizuje konkrétní *Atribut*. V modulu jsou již vytvořeny všechny objekty s jejich atributy z kapitoly [3]. Vytvořený objekt má pak zobecněnou strukturu znázorněnou v tabulce 4.8. Vytvořené objekty v tomto modulu nemají názvy Object, jak je uvedeno v tabulce, ale již konkrétní svůj název. Taktéž neobsahují jeden atribut, ale všechny atributy, které náš frekvenční měnič podporuje a jsou v této struktuře pod svými názvy.

[1] Python jako jazyk nemá modifikátory přístupu, jako jiné objektové jazyky. Třídy, vlastnosti nebo funkce, jejichž název začíná podtržítkm se tváří jako privátní.

Object.	Properties.	ID	
		Name	
	Attribute.	Class.	ID
			Name
		ID	
		Name	
		DataType	
		AccessRule.	Get
			Set
		Parameter.	ID
			SubIndex
			Length
			Bit
			Scaling
			ConvertTable

Tabulka 4.8: Obecná struktura objektu v CIPObjectLibrary.py

Každý objekt v parametru *Properties* má uloženo své Class ID a jméno. Dále objekt obsahuje atributy, v nichž je uložena informace o objektu, z kterého pochází (Attribute.Class.ID/Name), dále obsahuje informaci, zda-li se dá do atributu zapisovat nebo jen číst (Attribute.AccessRule.Get/Set - obsahují boolovou hodnotu) a také obsahuje informaci o parametru z expertlistu, pokud tedy s nějakým souvisí. Vlastnost *Parameter* tedy neobsahují všechny atributy. Vlastnosti *ID*, *SubIndex* a *Length* parametru jsou povinné, zbylé jako *Bit*, *Scaling* a *ConvertTable* jsou nepovinné a parametr je obsauje pouze v případě nutnosti. Pokud máme atribut boolového charakteru a je taktéž obsažen v parametru expertlistu, tak vlastnost *Bit* je číslo daného bitu v parametru. Například atribut Running1 z Control Supervisor objektu indikuje zda se motor točí a má souvislost s druhým bitem parametru r52, kde sídlí status word ZSW1. Vlastnost *Scaling* je konstanta, o kterou se hodnota parametru musí vynásobit, abychom dostali hodnotu atributu. Tato vlastnost je tu pro atributy, kde se liší jednotky atributů s jednotkama parametru. Například atribut RatedCurrent má jednotky v [100mA], ale jednotky hodnoty parametru v expertlistu, kde sídlí hodnota tohoto atributu, je v jednotkách [A]. Proto je třeba hodnotu parametru vynásobit parametrem *Scaling*, v tomto případě *Scaling=10*, abychom získali hodnotu atributu. *ConvertTable* je slovník, který překládá hodnotu parametru na hodnotu atributu. Například tato tabulka je v případě atributu MotorType v MotorData objektu.

Jak jsem se již zmiňoval, v modulu OPCs.py [4.5.3] má třída ABControlLogixOPC funkce

GetAttribute a SetAttribute, kde jeden vstupních paramtetrů je právě atribut z tohoto modulu. Kdybychom tedy například chtěli vyčíst aktuální rychlost motoru z Drive objektu, pomocí funkce GetAttribute, předali bychom jí atribut z tohoto modulu následovně : *GetAttribute(CIPObjectLibrary.DriveObject.SpeedActual)*, kde tedy z modulu vybíráme Drive objekt a z něj atribut SpeedActual.

Tato knihovna nám usnadňuje práci s atributy, kdy není zapotřebí si pamatovat ID třídy, atributu a dalších vlastností. Další výhodou je, že při změně specifikace nebo parametru, se kterým je atribut spjat, se provede změna pouze v tomto modulu.

4.5.6 EIPToolbox.py

V průběhu návrhu testů jsme si všimli, že mnoho částí se neustále opakuje a to dalo základ myšlence vytvořit Toolbox s funkcemi, které velmi usnadní psaní samotných testů. Jelikož je zde využit jazyk Python, který je dynamicky interpretovaný, je snadné pomocí tohoto modulu zreprodukovat nalezené chyby a dále je analyzovat.

Po importu tohoto modulu je třeba jej naparametrizovat a následně můžeme využívat jeho funkce. Funkce se dají rozdělit do třech skupin. První skupina jsou **GET/SET funkce**, které mají za úkol vyčíst či nastavit určitý atribut. Druhou skupinou jsou **Check funkce**, které mají za úkol zkontrolovat stav zařízení a poslední skupinou jsou **Ostatní funkce**.

4.5.6.1 Properties

Modul obsahuje třídu *Properties*, kterou je třeba po importu naparametrizovat. Parametry *Properties* jsou *Controller*, *Device* a *PGPC*. Atributu Controller se předá vytvořená instance ABControlLogix z EIP_Devices.py, atributu Device se předá vytvořená instance třídy Converter z EIP_Devices.py a PGPC se předá PGPC, které je obsaženo v struktuře Project. Tato parametrizace je důležitá, neboť funkce EIPToolboxu pracují s funkcemi, které tato zařízení poskytují.

4.5.6.2 GET/SET funkce

Get/Set funkce mají za úkol vyčíst hodnotu daného atributu nebo do atributu zapsat konkrétní hodnotu. Je důležité si uvědomit, že ne vždy můžeme dané atributy vyčíst či do nich zapsat. Vše závisí na mnoha okolnostech. Například nemůžeme vyčítat či zapisovat hodnoty atributů, které jsou v jiném profilu, než se zařízení nachází. Taktéž nelze zapisovat do některých atributů, točí-li se motor nebo není nastavena možnost řízení pomocí Control Supervisor objektu.

- *getState()* - Funkce vrátí hodnotu atributu State z Control Supervisor objektu.

- *getAtReference()* - Funkce vrátí hodnotu atributu AtReference z Drive objektu zařízení.
- *getAccelTime()* - Funkce vrátí hodnotu atributu AccelTime z Drive objektu zařízení.
- *getDecelTime()* - Funkce vrátí hodnotu atributu DecelTime z Drive objektu zařízení.
- *getSpeedScale()* - Funkce vrátí hodnotu atributu SpeedScale z Drive objektu zařízení.
- *getHighSpdLimit()* - Funkce vrátí hodnotu atributu HighSpdLimit z Drive objektu zařízení.
- *getLowSpdLimit()* - Funkce vrátí hodnotu atributu LowSpdLimit z Drive objektu zařízení.
- *getSpeedRef()* - Funkce vrátí hodnotu atributu SpeedRef z Drive objektu zařízení.
- *getIPAddress(Via, OtherIPAddress=")* - Funkce vrací nastavenou IP adresu zařízení. Vstupní parametr Via určuje, zda-li IP adresu vyčte z TCP/IPInterface objektu (Via='MSG') nebo z příslušného parametru pomocí Telnetu (Via='Telnet'). Pokud zjišťujeme IP adresu z TCP/IP Interface objektu, je třeba se zařízení dotázat pomocí explicitní zprávy, kterou vysílá kotnrolér. V kontroléru je standardně naparametrizovaná IP adresa dotazovaného zařízení. Pokud tedy cílové zařízení má nastavenou IP adresu jinou než výchozí, je nutné nastavit novou cílovou adresu explicitní zprávy. K tomuto účelu slouží vstupní parametr *OtherIPAddress*, který přejímá IP adresu jako textový řetězec.
- *getSubNetMask(Via)* - Funkce vrátí nastavenou masku podsítě v zařízení. Opět tuto informaci můžeme vyčíst z TCP/IPInterface objektu (Via='MSG') nebo z parametru v expertlistu (Via='Telnet').
- *getGateWay(Via)* - Funkce vrátí nastavenou bránu v síťovém nastavení zařízení. Tuto informaci můžeme vyčíst z TCP/IPInterface objektu (Via='MSG') nebo z parametru v expertlistu (Via='Telnet').
- *getDeviceName(Via)* - Funkce vrátí jméno zařízení nastavené v síťovém nastavení zařízení. Tuto informaci můžeme vyčíst z TCP/IPInterface objektu (Via='MSG') nebo z parametru v expertlistu (Via='Telnet').
- *setNetCtrl()* - Funkce nastaví atribut NetCtrl v Control Supervisor objektu na hodnotu True a umožní tak řídit motor pomocí tohoto objektu.
- *clearNetCtrl()* - Funkce nastaví atribut NetCtrl v Control Supervisor objektu na hodnotu False, zařízení bude opět řízeno pomocí dat v cyklické komunikaci.
- *setNetRef()* - Funkce nastaví atribut NetRef v Drive objektu na hodnotu True a umožní tak řídit motor pomocí Control Supervisor objektu.

- *clearNetRefl()* - Funkce nastaví atribut NetRef v Drive objektu na hodnotu False, motor bude opět řízen pomocí dat v cyklické komunikaci.
- *setRun1()* - Funkce nastaví atribut Run1 v Control Supervisor objektu na hodnotu True a spustí motor, za podmínek, že je povoleno řízení pomocí Control Supervisor objektu (atribut NetCtrl nebo NetRef musí mít hodnotu True).
- *clearRun1()* - Funkce nastaví atribut Run1 v Control Supervisor objektu na hodnotu False a zastaví tak motor, za podmínek, že je povoleno řízení pomocí Control Supervisor objektu (atribut NetCtrl nebo NetRef musí mít hodnotu True).
- *setRun2()* - Funkce nastaví atribut Run2 v Control Supervisor objektu na hodnotu True a spustí tak motor, za podmínek, že je povoleno řízení pomocí Control Supervisor objektu (atribut NetCtrl nebo NetRef musí mít hodnotu True).
- *clearRun2()* - Funkce nastaví atribut Run2 v Control Supervisor objektu na hodnotu False a zastaví tak motor, za podmínek, že je povoleno řízení pomocí Control Supervisor objektu (atribut NetCtrl nebo NetRef musí mít hodnotu True).
- *setSpeedRef(Value)* - Nastaví atribut SpeedRef v Drive objektu na požadovanou hodnotu.
- *setDecelTime(Value)* - Nastaví atribut DecelTime v Drive objektu na požadovanou hodnotu.
- *setAccelTime(Value)* - Nastaví atribut AccelTime v Drive objektu na požadovanou hodnotu.
- *setSpeedScale(Value)* - Nastaví atribut SpeedScale v Drive objektu na požadovanou hodnotu.
- *setHighSpdLimit(Value)* - Nastaví atribut HighSpdLimit v Drive objektu na požadovanou hodnotu.
- *setLowSpdLimit(Value)* - Nastaví atribut LowSpdLimit v Drive objektu na požadovanou hodnotu.

4.5.6.3 Check funkce

Check funkce mají za úkol zkontrolovat hodnotu atributu, zda-li odpovídá předpokládané hodnotě. V některých případech je pro jistotu ověřeno více spolu souvisejících atributů. Návratová hodnota všech těchto funkcí je datového typu bool.

- *checkSpeed(Value=None, Telnet=False Tolerance=10, Profile=None, Exception=False)* - Funkce je závislá na profilu, jelikož v každém profilu kontroluje jiné atributy. V profilu AC Drive funkce přečte hodnoty atributů SpeedRef a SpeedActual v Drive objektu, aktuální hodnotu rychlosti z Input Assembly, tedy z cyklických dat a následně tyto hodnoty porovná v rozashu tolerance s očekávanou hodnotou (Value). V profilu Siemens se přečtou atributy MainSetpoint, ActualSpeed z Siemens Drive objektu a aktuální rychlost z cyklických dat. Tyto hodnoty se porovnají v rozsahu tolerance s očekávanou hodnotou (Value). Je-li vstupní parametr Telnet=True, vyčítá se navíc aktuální rychlost z příslušného parametru v expertlistu k porovnání skrze Telnet. Pokud není zadána očekávaná vstupní hodnota Value, porovnávají se vyčtené aktuální rychlosti s referencí. Není-li zadán profil, funkce si sama zjistí, v jakém profilu se zařízení nachází.

- *checkRunning1(Value, Exception=False)* - Funkce přečte hodnotu atributu Running1 v Control Supervisor objektu a porovná s očekávanou hodnotou Value.

- *checkRunning2(Value, Exception=False)* - Funkce přečte hodnotu atributu Running2 v Control Supervisor objektu a porovná s očekávanou hodnotou Value.

- *checkNetworkControl(Value, Exception=False)* - Tato funkce kontroluje hodnoty atributů NetCtrl, CtrlFromNet, NetRef a RefFromNet. Atributy NetCtrl a NetRef mají stejnou funkci, pouze se liší objektem, ve kterém se nacházejí. Hodnoty všech těchto atributů musí být stejné a funkce je porovnává s očekávanou hodnotou Value.

- *checkNetworkControl(Value, Exception=False)* - Tato funkce kontroluje hodnoty atributů NetCtrl, CtrlFromNet, NetRef a RefFromNet. Atributy NetCtrl a NetRef mají stejnou funkci, pouze se liší objektem, ve kterém se nacházejí. Hodnoty všech těchto atributů musí být stejné a funkce je porovnává s očekávanou hodnotou Value.

- *checkReady(Value, Exception=False)* - Funkce přečte hodnotu atributu Ready v Control Supervisor objektu a porovná s očekávanou hodnotou.

- *checkAtReference(Value, Exception=False)* - Funkce přečte hodnotu atributu AtReference v Drive objektu a porovná s očekávanou hodnotou.

- *checkFaults(Value, Profile=None, Exception=False)* - Funkce zkontroluje, zda-li zařízení hlásí nebo nehlásí chybu a porovná s očekávanou hodnotou Value. Pokud jsme v AC Drive profilu, je kontrolován nultý bit Faulted v Input Assembly a atribut Faulted v Control Supervisor objektu. V Siemens profilu je kontrolován třetí bit[1] status wordu (ZSW1), který dostáváme v cyklických datech od zařízení a atribut Fault present v Siemens Drive

[1] Počítáno od nuly.

objektu. Pokud není zadán profil, funkce si zkontroluje sama, v jakém profilu se zařízení nachází.

- *checkDrive(Value, Telnet=True, Profile=None, Exception=False)* - Funkce kontroluje, zda-li je motor v pohybu či není a porovná s očekávanou hodnotou Value. V AC Drive profilu jsou kontrolovány atributy State, Running1, Running2, ActualSpeed a obsah Input Assembly. V Siemens profilu jsou kontrolovány atributy OperationEnabled, ActualSpeed a dále obsah InputAssembly. Pokud je vstupní hodnota Telnet=True, kontrolují se přes telnet parametry se status wordem (ZSW1) a aktuální rychlost. Funkce nám tak prověří všechny dostupné atributy a parametry, z kterých lze zjistit, zda-li se motor točí či nikoliv.
- *checkState(Value, Exception=False)* - Funkce zjistí hodnotu atributu State v Control Supervisor objektu a porovná s očekávanou hodnotou Value.
- *checkIPAddress(Value, Exception=False)* - Funkce nejprve zkontroluje odpověď na dotaz ping na zadanou IP adresu (Value), poté zkontroluje nastavenou IP adresu v TCP/IP Interface objektu zařízení a nakonec otestuje IP adresu zařízení pomocí DCP.
- *checkSubNetMask(Value, Exception=False)* - Funkce zkontroluje nastavenou masku podsítě v TCP/IP Interface objektu zařízení a poté otestuje masku podsítě zařízení pomocí DCP.
- *checkGateWay(Value, Exception=False)* - Funkce zkontroluje nastavenou bránu v TCP/IP Interface objektu zařízení a poté otestuje bránu zařízení pomocí DCP.
- *checkDeviceName(Value, Exception=False)* - Funkce zkontroluje nastavené jméno zařízení v TCP/IP Interface objektu a poté otestuje jméno zařízení pomocí DCP.

4.5.6.4 Ostatní funkce

Do této kategorie patří následující funkce :

- resetFaults(Via, Profile=None) - Funkce potvrdí chyby v zařízení. V profilu AC Drive pomocí bitu FaultReset v Output Assembly (Via='CyclicData') nebo pomocí explicitní zprávy se službou Set_Attribute_Single a hodnotou True adresované na atribut FaultReset v Control Supervisor objektu (Via='MSG'). V profilu Siemens můžeme potvrdit chyby bitem 7[1] control wordu STW1 v Output Assembly (Via='CyclicData'). Nebo pomocí atributu AcknowledgeFault v Siemens Drive objektu (Via='MSG').
- *restartDevices()* - Funkce restartuje zařízení a kontrolér, které má EIPToolbox uložené v Properties.

[1] Počítáno od nuly.

- *pingDevices(Timeout, Exception=False)* - Funkce čeká po dobu Timeout na ping odezvu zařízení a kontroléru, které má EIPToolbox uložené v Properties.
- *ping(IPAddress)* - Funkce vyšle příkaz ping na zadanou IP adresu a čeká na odpověď. Návratová hodnota je datového typu boolean.
- *findDeviceViaDCP()* - Funkce se snaží najít pomocí DCP zařízení uložené v Properties na základě jeho MAC adresy. Pokud je zařízení nalezeno, vrátí se dané zařízení jako datový typ IPDevice z Dev_Struct.py se síťovým nastavením, které bylo v odezvě na DCP.

4.6 Příklad skriptování

Na obrázku 4.4 vidíme ukázku skriptování pomocí funkcí z EIPToolbox.py a jejich zápisů do logovacího souborou. Na začátu po spuštění spouštěcího skriptu, který naimportuje potřebné knihovny, vytvoří projekt s příslušnou HW konfigurací a založí nový soubor pro logování, je třeba naparametrizovat vlastnosti EIPToolbxu. Do EIPToolbox.Properties je potřeba předat kontrolér a zařízení. V této ukázce spouštíme motor pomocí cyklických dat v profilu AC Drive s referencí 100[RPM]. Zkontrolujeme, zda-li rychlost odpovídá naší referenci. Poté zkusíme, zda-li stav zařízení je ve stavu 3-Ready. Z kapitoly [3.3.2.3] víme, že pokud se motor točí, stav tohoto zařízení bude 4-Enabled. Zde můžeme vidět chybu, která se viditelně zaznamenává. Následně pomocí funkce setNetCtrl() sebereme kontroléru kontrolu nad zařízením a můžeme jej ovládat pomocí Control Supervisor a Drive objektu. Příkazem setSpeedRef(Value=280) nastavíme vyšší rychlost otáček a zkontrolujeme znovu rychlost. Zastavíme motor nastavením atributu Run1 do logické nuly a přesdvědčíme se, zda-li se zařízení nachází ve stavu 3-Ready. Dále nastavením atributu NetCtrl do logické nuly předáme kontrolu nad zařízením zpět kontroléru. Ten stále vysílá v Output Assembly, aby se motor točil rychlostí 100[RPM]. Proto zkontrolujeme rychlost a nakonec zastavíme motor.

```
>>> EIPToolbox.Properties.Controller = Project.Devices['ABControlLogix']
>>> EIPToolbox.Properties.Device = Project.Devices[´FrequencyConverter´]
>>> controller.OPC.GoOnline()
28.04.2013 12:31:32: OPC client: going online...
28.04.2013 12:31:34: Connection to OPC server established
>>> controller.OPC.StartStopDrive(Run=True,SpeedReference=100,Profile='ODVA')
>>> EIPToolbox.checkSpeed(Value=280,Telnet=False,Tolerance=10,Profile='ODVA')
28.04.2013 12:39:49: Check speed :
28.04.2013 12:39:49:    Actual speed expected : 280
28.04.2013 12:39:50:    Speed reference in Drive Object : 280
28.04.2013 12:39:52:    Actual speed in Drive Object : 281
28.04.2013 12:39:53:    Actual speed in Input Assembly : 280
28.04.2013 12:39:53:    ---OK---
True
>>> EIPToolbox.clearRun1()
28.04.2013 12:40:47: Clear Run1 ... OK
>>> device.Restart()
>>> EIPToolbox.checkState(Value=3,Exception=False)
28.04.2013 12:51:34: Check ODVA state :
28.04.2013 12:51:34:    Expected ODVA state is 3
------------------------------------
28.04.2013 12:51:37: FAULT :   Actual ODVA state is 4 ... FAIL
------------------------------------
False
>>> EIPToolbox.setNetCtrl()
28.04.2013 12:53:22: Set NetCtrl ... OK
>>> EIPToolbox.clearRun1()
28.04.2013 12:53:49: Clear Run1 ... OK
>>> EIPToolbox.checkState(Value=3)
28.04.2013 12:54:32: Check ODVA state :
28.04.2013 12:54:32:    Expected ODVA state is 3
28.04.2013 12:54:34:    Actual ODVA state is 3 ... OK
True
>>> EIPToolbox.clearNetCtrl()
28.04.2013 12:54:58: Clear NetCtrl ... OK
>>> EIPToolbox.checkSpeed(Value=100,Telnet=False,Tolerance=10,Profile='ODVA')
28.04.2013 12:57:54: Check speed :
28.04.2013 12:57:54:    Actual speed expected : 100
28.04.2013 12:57:56:    Speed reference in Drive Object : 100
28.04.2013 12:57:58:    Actual speed in Drive Object : 97
28.04.2013 12:58:00:    Actual speed in Input Assembly : 97
28.04.2013 12:58:00:    ---OK---
True
>>> controller.OPC.StartStopDrive(Run=False,Profile='ODVA')
```

Obrázek 4.4: Ukázka skriptování pomocí funkcí EIPToolboxu

Kapitola 5

Testy

V této kapitole si popíšeme testy, které byly navrženy pro testování převážně AC Drive profilu našeho zařízení a dalších vlastností. Chování jednotlivých atributů musí korespondovat s chováním popsaným ve specifikaci protokolu CIP [1]. Dále se testují vlastnosti, které popisuje interní dokument zvaný *Feature Description*. Některé testy jsou společné pro oba profily testovaného frekvenčního měniče a liší se pouze v parametru Profile, který se nachází v skriptu pro parametrizaci testu. Tyto testy jsem nazval jako společné. Další testy jsou vyvinuté přímo pro profil AC Drive. Každý test se skládá ze tří částí :

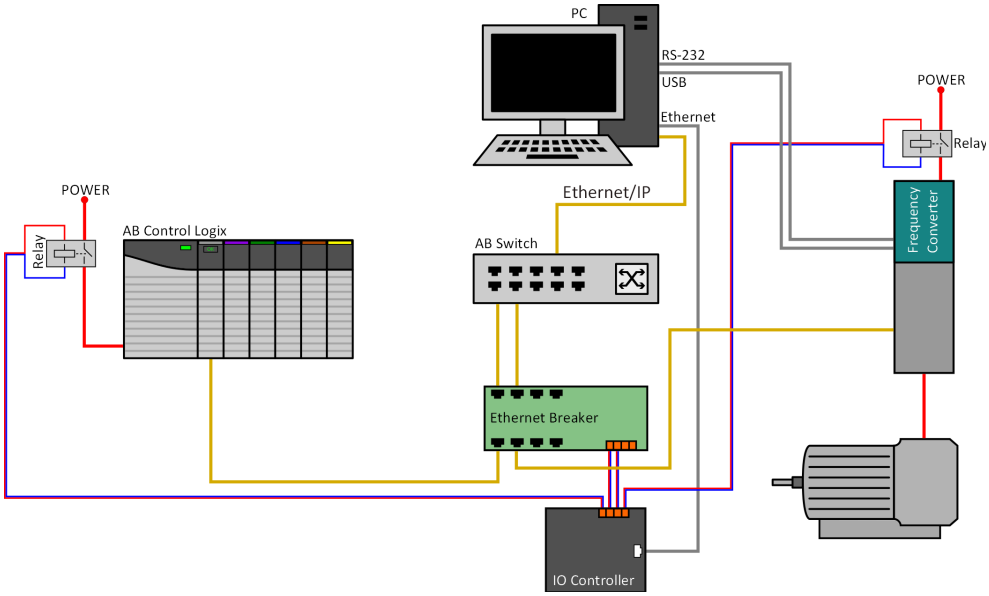
- Příprava na test
- Test
- Konec testu

V části pro přípravu testu většinou inicializujeme HW, připravujeme EIPToolbox a vytváříme připojení k OPC serveru. V této části také připravujeme zařízení do výchozího stavu pro test. Po přípravě se provádí test samotný. Po skončení testu zrušíme připojení k OPC serveru a uvedeme zařízení do výchozího stavu. Dále jsou vypsány veškeré chyby, pokud se tedy jednalo o chyby, které nezpůsobují vyjímku a nezastavují běh testu. Všechny testy využívají hlavně EIPToolboxu a jsou tak snadno čitelné a prostor pro vytvoření chyby je tak minimalizován. Vývojové diagramy testů jsou v příloze C.

5.1 Zapojení racku

Dříve než se pustíme do popisu samotných testů, popíšeme si reálné zapojení racku, které je znázorněno na obrázku 5.1, kde vidíme PLC Allen-Bradley jako kontrolér, zapojený do zařízení Ethernet Breaker, který pokud dostane signál od IO Controller, rozpojí spojení mezi

PLC a switchem. Testovaný frekvenční měnič je také zapojen do zařízení Ethernet Breaker. IOController je ovládán počítačem díky knihovně RemoteIOLib.dll a komunikují spolu skrze ethernet. IO Controller též ovládá relé, která při odpadnutí odpojí napájení měniče. Switch, který vidíme na obrázku plní pouze úlohu switche a nepoužíváme žádné jeho speciální možnosti. Frekvenční měnič je k PC připojen nejen skrze ethernet, ale také sériovým kabelem a USB kabelem. Program Simatic SCOUT, který používáme ke konfiguraci měniče, využívá připojení skrze USB. Výhoda tohoto připojení je v době, kdy přerušíme síťový kabel mezi měničem a switchem, jelikož program Simatic SCOUT je nadále připojen a můžeme tak pozorovat vnitřní stavy testovaného zařízení. Připojení skrze sériový kabel využívá nástroj TCI pro sběr debugovacích logů přímo ze zařízení. Štítkové údaje použitého motoru jsou v tabulce 5.1



Obrázek 5.1: Zapojení racku

SIEMENS 3 Mot IEA7096-4AA16
UD 0702/1062838-001-2
IP55 90L IM B35
50Hz 230/400V Δ/Y
1,5kW 5,9/3,4A
cos φ=0,81 1420/min
220-240/380-420V Δ/Y
6,1-6,1/3,5-3,5A

Tabulka 5.1: Štítkové údaje motoru

5.2 Společné testy

V této části jsou popsány testy, které jsou společné pro oba profily. Kód testu je pro oba profily stejný, pouze se liší v parametru Profile a případně dalších. Parametry testu se nachází ve skriptech pro parametrizaci testů. Každý test může mít více parametrizačních skriptů a je pak na nás s jakými parametry chceme testy spouštět.

5.2.1 Ethernet cable break test

Ethernet cable break test je jeden ze základních a velmi důležitých testů. V tomto testu se přerušuje síťové spojení mezi kontrolérem a měničem. frekvenční měnič po ztrátě aplikačního vztahu s kontrolérem musí okamžitě vypnout motor a vyhlásit chybu. Po obnovení spojení a komunikace, je třeba chybu potvrdit pro znovuroztočení motoru. Test probíhá neustále v cyklu a jakákoliv neočekávaná chyba zastaví celý běh, abychom mohli provést analýzu chyby. Počet cyklů a v jakém profilu se zařízení nachází se nastavuje v skriptu pro parametrizaci testu. Diagram průběhu testu je na obrázku C.1.

5.2.2 Power cable break test

Power cable break test je taktéž jeden ze základních testů. Tento test cyklicky vypíná a zapíná testované zařízení. Testuje se, zda-li po zapnutí napájení frekvenčního měniče naběhne zpátky komunikace a motor se opět roztočí. Diagram průbhu tohoto testu je na obrázku C.2

5.2.3 DCP test

Tento test je nezávislý na profilu, jelikož reakce na DCP protokol nesouvisí s profilem. V tomto testu zkoušíme, zda-li zařízení správně zareaguje na změnu síťového nastavení, které se projeví v TCP/IP Interface objektu. Důležité je zkusit všechna možná nastavení jak dočasně, tak i permanentně. Vždy pomocí restartu zařízení se přesvědčíme, zda-li nové síťové nastavení zůstalo či nezůstalo uložené. V skriptu pro parametrizaci testu volíme, jaké testovací síťové nastavení se má nastavit. V kontroléru máme přidaný navíc další měnič s tímto testovacím síťovým nastavením. Změníme-li IP adresu testovaného zařízení, kontrolér ztratí s tímto zařízením AR, ale naváže nové AR se zařízením skrze druhé nastavené zařízení v HW konfiguraci. Diagram průběhu testu je na obrázku C.3.

5.2.4 GET test a SET test

Jedná se o dva testy, které jsou si velmi podobné svým průběhem. Některé atributy jsou spjaté s parametry v expertlistu a právě tyto atributy jsou předmětem našeho testování. Atributy pochází z následujících objektů : Motor Data, Drive, Siemens Drive, Siemens Motor Data a Parameter. Pro každý objekt je vytvořen skript pro parametrizaci testu. V tomto skriptu využíváme modulu CIPObjectLibrary.py, kde jsou vytvořeny struktury s objekty a jejich atributy. Z tohoto modulu si importujeme konkrétní objekt, z něhož vybíráme atributy, jenž budeme testovat a vkládáme je do iterovatelného pole.

V GET testu procházíme toto pole s atributy a postupně je testujeme. Na začátku zkontrolujeme, zda-li je atribut určen pouze pro čtení nebo i pro zápis. Je-li atribut určen pouze pro zápis, přečteme hodnotu parametru expertlistu pomocí Telnetu a hodnotu atributu skrze explicitní zprávu. Obě hodnoty porovnáme. Po porovnání vezmeme další atribut z pole. Je-li tento atribut určen nejen pro čtení, ale i pro zápis, zkusíme zapsat testovací hodnotu do parametru skrze Telnet. Pole testovacích hodnot atributu definujeme v skriptu pro parametrizaci testu. Po zapsání testovací hodnoty, hodnotu opět vyčteme pomocí Telnetu z parametru a pomocí explicitní zprávy z atributu. Tímto otestujeme, zda-li je atribut opravdu spjat se správným parametrem[1]. Taktéž tímto vyzkoušíme, zda-li konverze mezi hodnotou parametru a atributu je správná. Diagram GET testu je na obrázku C.4.

SET test je velmi podobný, avšak v parametrizačním skriptu vybíráme pouze atributy, které jsou určené i pro zápis skrze explicitní zprávu. V GET testu jsme zkoušeli zapisovat do parametru skrze Telnet a vyčítat hodnotu skrze explicitní zprávu, poté porovnali. V případě SET testu nastavíme hodnotu atributu skrze explicitní zprávu a vyčteme hodnotu parametru skrze Telnet k porovnání. Tím otestujeme, zda-li atributy jsou skutečně určené pro zápis a zda-li jsou spjaté se správným parametrem. Diagram SET testu je na obrázku C.5.

Na hodnoty atributů se dotazujeme skrze explicitní zprávu, kterou za nás vždy vysílá kontrolér. Přijmutá data nebo odeslaná data, jsou vždy pole bajtů o délce datového typu atributu. Konverzi mezi datovým typem atributu a polem bajtů nám zajišťují funkce pro vysílání či zapisování atributu v PyTeMatu. Při této konverzi je vždy kontrolována délka přijatých nebo odesílajících dat. Tímto je nepřímo zkontrolováno, zda-li atributy jsou naimplementované ve správném datovém formátu. Další kontrolou datového typu je právě porovnávání hodnot atributů s hodnotami parametrů v expertlistu.

5.2.5 Change Interface Config test

V tomto testu zkoušíme nastavit nové síťové nastavení pomocí explicitních zpráv se službou Set_Attribute_Single v objektu TCP/IP Interface [3.2.2]. Postupně zkoušíme změnit atributy IP

[1] Jaký je atribut spjatý s parametrem je definováno v interní neveřejné dokumentaci zvané Feature Description.

adresy, masky podsítě, brány a jména zařízení. Důležité je po každé změně restartovat zařízení a vyzkoušet, zda-li dané nastavení se uložilo permanentně.

5.3 Testy pro ODVA profil

Soubor těchto testů souvisí hlavně s atributy v Supervisor Control objektu a Drive objektu. Testy byly napsány na základě informací z specifikace protokolu CIP [1].

5.3.1 AtReference test

AtReference je atribut z Drive objektu, který jsme si popsali v kapitole [3.3.2.1]. Po importu a inicializaci EIPToolboxu, připojení k OPC serveru, přebereme kontrolu nad zařízením nastavením atributu NetCtrl do logické jedničky. Přednastavíme si v atributu SpeedRef velké otáčky (1200[RPM]) a také si nastavíme dlouhou dobu náběhu a doběhu (60[s]). V této chvíli se motor netočí a je vklidu. Zkusíme zkontrolovat atribut AtReference, zd-li je v logické nule. Dalším krokem je roztočení motoru pomocí atributu Run1 a jelikož máme nastavenou dlouhou dobu náběhu, můžeme v průběhu najíždění motoru kontrolovat atribut AtReference, že je stále v logické nule. Současně kontrolujeme rychlost, zda-li nedosáhla reference. Po dosáhnutí požadované rychlosti necháme motor chvíli běžet a stále kontrolujeme atribut AtReference, tentokrát však požadujeme, aby byl v logické jedničce. Na konec nastavíme atribut Run1 na logickou nulu. Motor vlivem nastaveného dlouhého decelarčního času se začne pomalu zastavovat. Atribut State je v 5-Stopping stavu a během tohoto stavu a následného 3-Ready stavu je atribut AtReference v logické nule. Na konci testu předáme kontrolu nad zařízením kontroléru a zrušíme spojení s OPC. Diagram testu je v příloze na obrázku C.6.

5.3.2 Negative speed test

Jedná se o krátký test, kdy nejprve zkusíme roztočit motor pomocí kontroléru a cyklických dat. Avšak do přenášených bajtů rychlosti, zapíšeme zápornou hodnotu. Jelikož dle CIP specifikace je SpeedRef datového typu INT, je možné zapsat zápornou hodnotu. Dále zkontrolujeme, zda-li se motor opravdu točí zápornými otáčkami, tedy v druhém směru. Zastavíme motor a přebereme nad ním kontrolu pomocí atributu NetCtrl. Zapíšeme do atributu SpeedRef zápornou hodnotu rychlosti a pomocí atributu Run1 motor roztočíme. Zkontrolujeme, zda-li se motor opět točí zápornými otáčkami. Nakonec zastavíme motor a vrátíme kontrolu nad zařízením zpět kontroléru. Diagram testu je na obrázku C.7.

5.3.3 ODVA State test

Test se zaměřuje na stavový diagram, který definuje AC Drive profil atributem State v Control Supervisor objektu [3.3.2.3]. Na obrázku stavového diagramu 3.5 jsou stavy rozlišeny barevně :

- Modré - 0-Non-Existent, 1-Startup
- Oranžové - 2-Not_Ready, 6-Fault_Stop
- Zelené - 3-Ready, 4-Enabled, 7-Faulted

Modře označené stavy nejsme schopni zaznamenat, jelikož probíhají v době, kdy zařízení nemá ještě navázanou komunikaci s kontrolérem. Oranžové stavy nejsme opět schopni zaznamenat, protože mají krátkou dobu trvání, řádově jednotky μ sekund. Zelené jsou stavy, které jako uživatel jsme schopni zaznamenat v atributu State.

Během celého testu ovládáme frekvenční měnič pomocí explicitních zpráv v režimu, kdy jsme sebrali kontroléru kontrolu nad zařízením (NetCtrl=True).

Stav 3-Ready je výchozí stav zařízení a do stavu 4-Enabled se dostaneme roztočením motoru (Run1=True). Na začátku testu ve fázi přípravy jsme nastavili dlouhý decelerační čas a vysoké otáčky reference. To nám dá dostatek času během zastavování motoru (Run1=False), na zaznamenání 5-Stopping stavu, během kterého v testu zkusíme znovu roztočit motor (Run1=True), čímž se nám opět dostane do stavu 4-Enabled. V průběhu 5-Stopping stavu také zkusíme vyhodit chybu rychlým rozpojením a zapojením síťového kabelu, čímž dostaneme zařízení do stavu 7-Faulted. Abychom se dostali ze stavu 7-Faulted je třeba potvrzení chyby (FaultReset=True). Dále také zkusíme vyvolat chybu během stavů 3-Ready a 4-Enabled. Tímto pokryjeme všechny možné přechody mezi stavy, kterých jsme schopni dosáhnout. Nakonci testu uvedeme zařízení do výchozího stavu.

Diagram průběhu testu je na obrázku C.8.

5.3.4 Run1 & Run2 test

V tomto testu ověřujeme, zda-li nastavování atributů Run1 a Run2 koresponduje s tabulkou 3.11. Ověřování probíhá pomocí hodnot atributů Running1 a Running2. Je-li aktivní RunType=Run1, hodnota atributu Running1 je v logické jedničce a naopak. Důležité je hlavně otestovat chování v případě, kdy například máme nastaven atribut Run1=True a chceme nastavit atribut Run2 do hodnoty True. V takovém případě zařízení dle tabulky 3.11 nebude reagovat a zůstává nastaven původní RunType=Run1.

5.3.5 Running & Ready test

Hodnoty atributů Running1 a Running2 nejsou podmíněné pouze atributy Run1 a Run2, ale také stavem ve kterém se zařízení nachází, jak je uvedeno v kapitole 3.3.2.3. Kdy jsou atributy Running1 a Running2 v logické jedničce je dáno výroky 3.2 a 3.3. Také atribut Ready je podmíněn stavem zařízení, tedy atributem State. Hodnota atributu Ready je v logické jedničce v případě že stav zařízení je 3-Ready, 4-Enabled nebo 5-Stopping. Do potřebných stavů se dostáváme pomocí atributu Run1, rozpojením-zapojením síťového kabelu a u stavu 5-Stopping využíváme vysokých otáček s dlouhým časem decelerace. Diagram testu je na obrázku C.10.

5.3.6 Set-Clear network control test

Tento test kontroluje atributy NetCtrl, CtrlFromNet, NetRef a RefFromNet. Atributy NetCtrl a NetRef mají totožnou funkci a to převzít kontrolu nad zařízením. Proto nastavím-li jeden atribut, musí se tato změna projevit i v atributu druhém a naopak. Diagram tohoto testu je na obrázku C.11.

5.3.7 Speed scaling test

Test je zacílen na atribut SpeedScale, který je popsán v kapitole [3.3.2.1] a atributy, které ovlivňuje (LowSpdLimit, HighSpdLimit, SpeedActual, SpeedRef). Test se dá rozdělit do dvou částí. Vždy probíhá stejně, pouze testuje jiné atributy.

Nejdříve testujeme atributy LowSpdLimit a HighSpdLimit, které souvisí s omezením rychlosti zdola a shora. Změníme atribut SpeedScale a vygenerujeme nové náhodné nepřeškálované hodnoty testovaných atributů, samozřejmě v omezeném rozsahu. Tyto hodnoty přeškálujeme dle vzorce 3.1. Dále hodnoty z atributů vyčteme a porovnáme. Toto zkusíme několikrát a poté opět změníme atribut SpeedScale. Takto testujeme dokud nezkusíme hodnoty atributů v celém rozsahu možností hodnot atributu SpeedScale. Rozsah atributu SpeedScale je $\langle -5, 5 \rangle$.

Po otestování těchto atributů, přejdeme k otestování atributů SpeedActual a SpeedRef. Tyto atributy testujeme obdobně. Nastavíme atribut SpeedScale na novou hodnotu, vygenerujeme novou náhodnou hodnotu reference rychlosti do maximální hodnoty rychlosti, kterou se může motor točit. Dále hodnotu přeškálujeme a zapíšeme do atributu SpeedRef. Roztočíme motor, a zkontrolujeme rychlost, kterou se motor točí. Dříve než nastavíme novou hodnotu atributu SpeedScale, zkusíme několikrát různé rychlosti. Na konci testu uvedeme zařízení do výchozího stavu.

Diagram tohoto testu je na obrázku C.12

Kapitola 6

Závěr

Práce na začátku stručně popisuje objektový model protokolu CIP a komunikaci na síti Ethernet/IP. Testovaný frekvenční měnič má nově možnost komunikace na síti Ethernet/IP a podporuje profil AC Drive definovaný ve specifikaci CIP [1]. Tento profil je po studii specifikace velmi podrobně popsán, jelikož jeho správná funkcionalita byla převážně předmětem našeho testování. Vedle tohoto profilu frekvenční měnič podporuje i výrobcem specifický profil Siemens, který převzal z PROFIdrive profilu obsah cyklické komunikace a stavovou mašinu. Tento profil je plně funkční a je testován pouze v rámci společných testů pro oba profily.

Jelikož je velmi důležité, aby test se psal snadno a byl co nejjednodušší, je v testlabu společnosti Siemens, s.r.o. vyvíjeno prostředí PyTeMat (Python Test Automat), který výrazným způsobem usnadňuje psaní testů a zabraňuje testerovi v průběhu vývoje tvořit chyby. Prostředí PyTeMat, dle zadání této práce, bylo rozšířeno o třídy, funkce a moduly, které umožnily vývoj následných testů. PyTeMat se neustále vyvíjí a v současné době se rozšiřuje o vlastní preprocesor a sadu nových příkazů, které daleko více zpřehlední test a následně umožní automatickou dokumentaci celého testu. Také je v plánu změnit dokumentování průběhu jednotlivých testů, kdy v současné době zapisujeme do textového souboru. Myšlenka je taková, aby průběh testu se zapisoval do xml souboru. Též se vyvine program pro čtení těchto průběhů a zpřístupní různé filtry pro zobrazení vybraného obsahu.

Na základě specifikací [1], [2] a neveřejného interního dokumentu *Feature Description*, byly v testovacím prostředí PyTeMat vyvinuty automatické testy, které převážně testují správnou funkčnost AC Drive profilu. Pomocí těchto testů byly odhaleny desítky chyb, některé i závažného charakteru. V průběhu této práce právě samotné testování zabralo nejvíce času.

Literatura

[1] Open DeviceNet Vendor Association : *The CIP Networks Library Volume 1: Common Industrial Protocol*. Specification, www.odva.org, 2011

[2] Open DeviceNet Vendor Association : *The CIP Networks Library Volume 2: EtherNet/IP Adaptation of CIP*. Specification, www.odva.org, 2011

[3] Open DeviceNet Vendor Association : *EtherNet/IP Quick Start for Vendors*. Handbook, www.odva.org, 2008

[4] Open DeviceNet Vendor Association : *EtherNet/IP^{TM} - CIP on Ethernet Technology*. Overview, www.odva.org, 2008

[5] Open DeviceNet Vendor Association : *Network Infrastructure for EtherNet/IP^{TM} - Intrduction and Consideration*. www.odva.org, 2008

[6] Open DeviceNet Vendor Association, *Recommended Functionality for EtherNet/IP Devices*. www.odva.org, 2012

[7] Open DeviceNet Vendor Association, *The Common Industrial Protocol (CIP^{TM}) and the Family of CIP Networks*. www.odva.org, 2007

[8] Brooks, P. : *Ethernet/IP – Industrial Protocol*. white paper, www.rockwellautomation.com, ENET-WP001A-EN-P, 2001

[9] Kožnar, J. : *EtherNet/IP a ControlNet*. Diplomová práce, Praha, 2003.

[10] Prokůpek, Z. : *EtherNet/IP pro průmyslové řízení*. Diplomová práce, Praha, 2007.

[11] Profibus International : *Profile Drive Technology PROFIdrive*. Technical specification, www.profibus.com, 2006.

[12] Internetové stránky Python modulu OpenOPC : *http://openopc.sourceforge.net/*

[13] Dokumentace Python modulu ctypes : *http://docs.python.org/2/library/ctypes.html#module-ctypes*

Příloha A

Obsah přiloženého CD

Přílohou je CD obsahující následující soubory:

- Adresář **Dokumenty** - text diplomové práce, podepsané prohlášení a zadání diplomové práce, vše ve formátu .pdf

Příloha B

$32C_{HEX}$ Siemens Drive Object

Attr. ID	Access Rule	Attribute Name	Data Type
2	GET/SET	Commissioning state	UINT
3	GET	STW1 Bit0 ON/OFF1	BOOL
4	SET	STW1 Bit1 OC/OFF2	BOOL
5	GET	STW1 Bit2 OC/OFF3	BOOL
6	GET	STW1 Bit3 Operation Enable	BOOL
7	GET	STW1 Bit4 Ramp Function Generator Enable	BOOL
8	GET	STW1 Bit5 Continue Ramp Function Generator Enable	BOOL
9	GET	STW1 Bit6 Speed Setpoint Enable	BOOL
10	GET	STW1 Bit7 Acknowledge Fault	BOOL
11	SET	STW1 Bit8 Jog Bit0	BOOL
12	GET	STW1 Bit9 Jog Bit1	BOOL
13	GET	STW1 Bit10 Master ctrl by PLC	BOOL
14	GET	STW1 Bit11 Direction Reversal Setpoint	BOOL
15	GET	STW1 Bit12	BOOL
16	GET	STW1 Bit13 Motorized Potentiometer Raise	BOOL
17	GET	STW1 Bit14 Motorized Potentiometer Lower	BOOL
18	GET	STW1 Bit15 CDS Bit0	BOOL
19	GET	Main Setpoint	REAL
20	GET	ZSW1 Bit0 Rdy For Switch On	BOOL
21	GET	ZSW1 Bit1 Ready	BOOL
22	GET	ZSW1 Bit2 Operation Enabled	BOOL
23	GET	ZSW1 Bit3 Fault Present	BOOL

Tabulka B.1: Siemens Drive Object část 1.

Attr. ID	Access Rule	Attribute Name	Data Type
24	GET	ZSW1 Bit4 Coast Down Active (OFF2)	BOOL
25	GET	ZSW1 Bit5 Quick Stop Active (OFF3)	BOOL
26	GET	ZSW1 Bit6 Switching On Inhibited Active	BOOL
27	GET	ZSW1 Bit7 Alarm Present	BOOL
28	GET	ZSW1 Bit8 Deviation, Setpoint/Actual Speed	BOOL
29	GET	ZSW1 Bit9 Control Request	BOOL
30	GET	ZSW1 Bit10 Maximum Speed Reached	BOOL
31	GET	ZSW1 Bit11 I,M,P Limit Reached	BOOL
32	GET	ZSW1 Bit12 Motor Holding Break Open	BOOL
33	GET	ZSW1 Bit13 Alarm Motor Overtemperature	BOOL
34	GET	ZSW1 Bit14 Motor Rotates Forward	BOOL
35	GET	ZSW1 Bit15 Alarm Drive Converter Overload	BOOL
36	GET	Actal Speed	REAL
37	GET/SET	Ramp Up Time	REAL
38	GET/SET	Ramp Down Time	REAL
39	GET/SET	Current Limit	REAL
40	GET/SET	Maximum Speed	REAL
41	GET/SET	Minimum Speed	REAL
42	GET/SET	OFF3 Ramp Down Time	REAL
43	GET/SET	PID Enable	UDINT
44	GET/SET	PID Filter Time Constant	REAL
45	GET/SET	PID D Gaind	REAL
46	GET/SET	PID P Gain	REAL
47	GET/SET	PID I Gain	REAL
48	GET/SET	PID Up Limit	REAL
49	GET/SET	PID Down Limit	REAL
50	GET	Speed Setpoint	REAL
51	GET	Output Frequency	REAL
52	GET	Output Voltage	REAL
53	GET	DC Link Voltage	REAL
54	GET	Actual Current	REAL
55	GET	Actual Torque	REAL
56	GET	Output Power	REAL

Tabulka B.2: Siemens Drive Object část 2.

Attr. ID	Access Rule	Attribute Name	Data Type
57	GET	Motor Temperature	REAL
58	GET	Power Unit Temperature	REAL
59	GET	Energy kWh	REAL
60	GET	CDS Eff(Local Mode)	UINT
61	GET	Status Word 2	UINT
62	GET	Control Word 1	UINT
63	GET	Motor Speed (Encoder)	REAL
64	GET	Digital Inputs	UDINT
65	GET	Digital Outputs	UDINT
66	GET	Analog Input 1	REAL
67	GET	Analog Input 2	REAL
68	GET	Analog Output 1	REAL
69	GET	Analog Output 2	REAL
70	GET	Fault Code 1	UINT
71	GET	Fault Code 2	UINT
72	GET	Fault Code 3	UINT
73	GET	Fault Code 4	UINT
74	GET	Fault Code 5	UINT
75	GET	Fault Code 6	UINT
76	GET	Fault Code 7	UINT
77	GET	Fault Code 8	UINT
78	GET	Pulse Frequency	REAL
79	GET	Alarm Code 1	UINT
80	GET	Alarm Code 2	UINT
81	GET	Alarm Code 3	UINT
82	GET	Alarm Code 4	UINT
83	GET	PID Setpoint Output	REAL
84	GET	PID Feedback	REAL

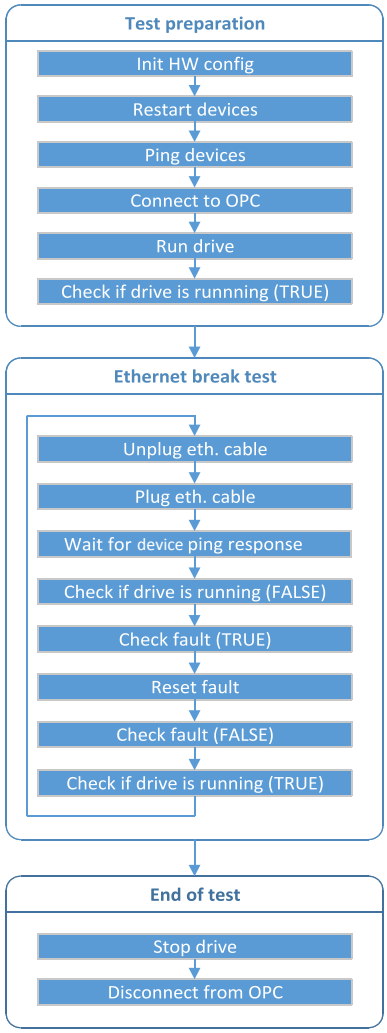
Tabulka B.3: Siemens Drive Object část 3.

Příloha C

Vývojové diagramy testů

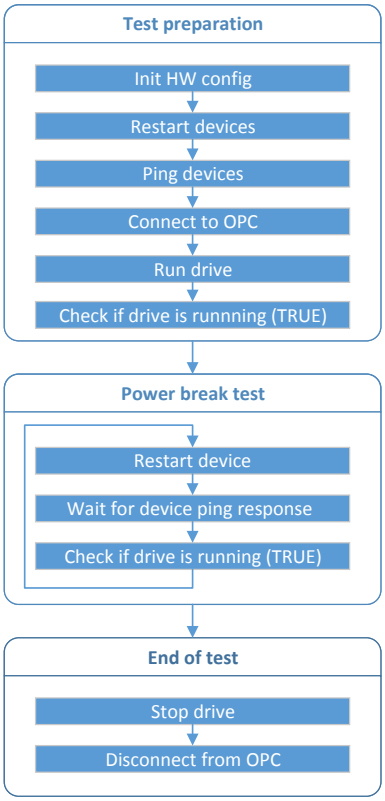
Hodnoty v závorkách u kroků jsou očekávané hodnoty.

C.1 Ethernet cable break test



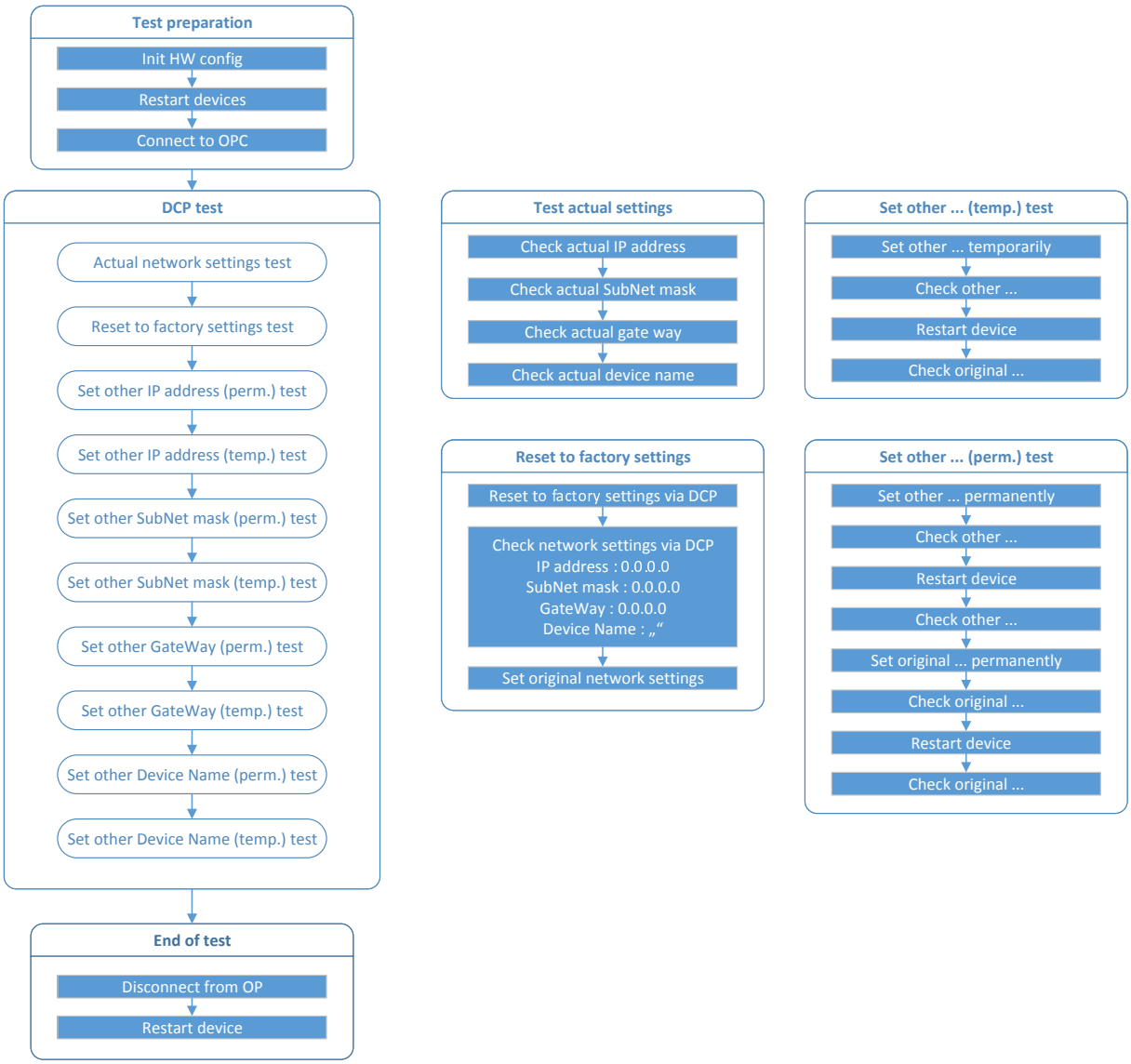
Obrázek C.1: Vývojový diagram ethernet cable break testu

C.2 Power cable break test



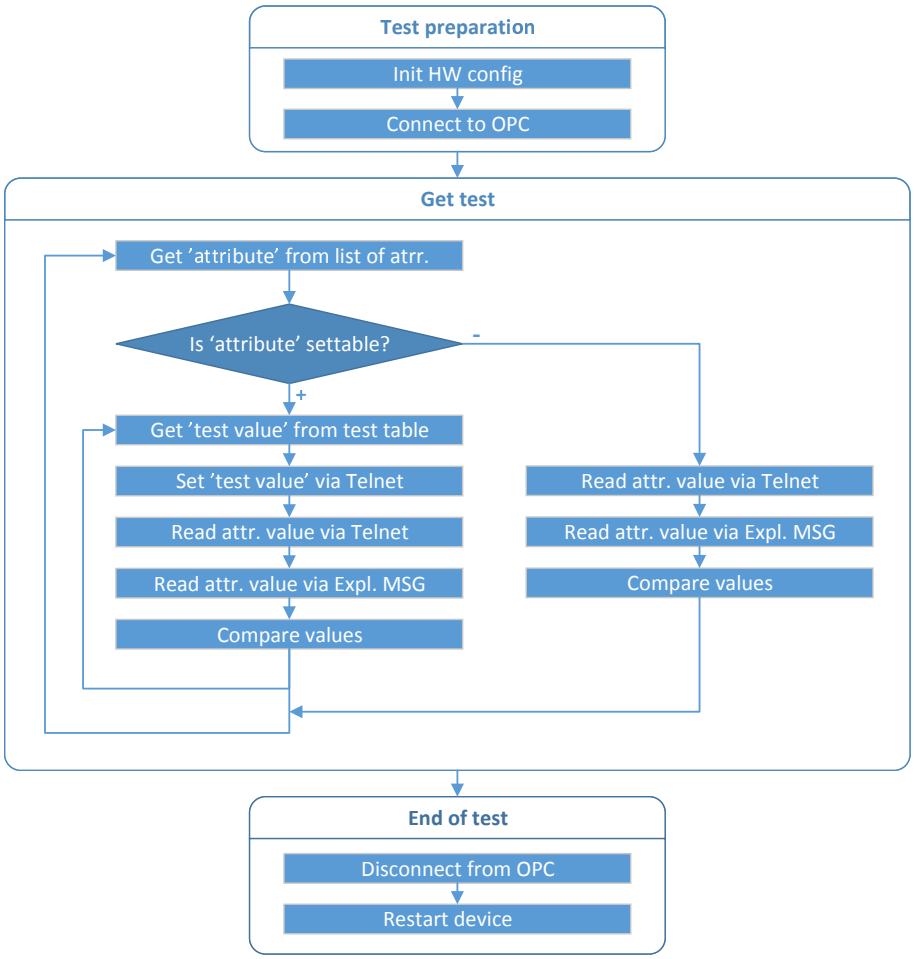
Obrázek C.2: Vývojový diagram power cable break testu

C.3 DCP test



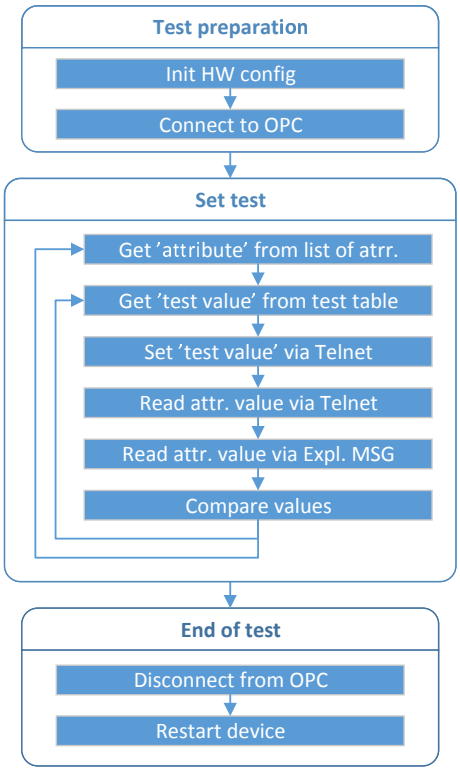
Obrázek C.3: Vývojový diagram DCP testu

C.4 GET test



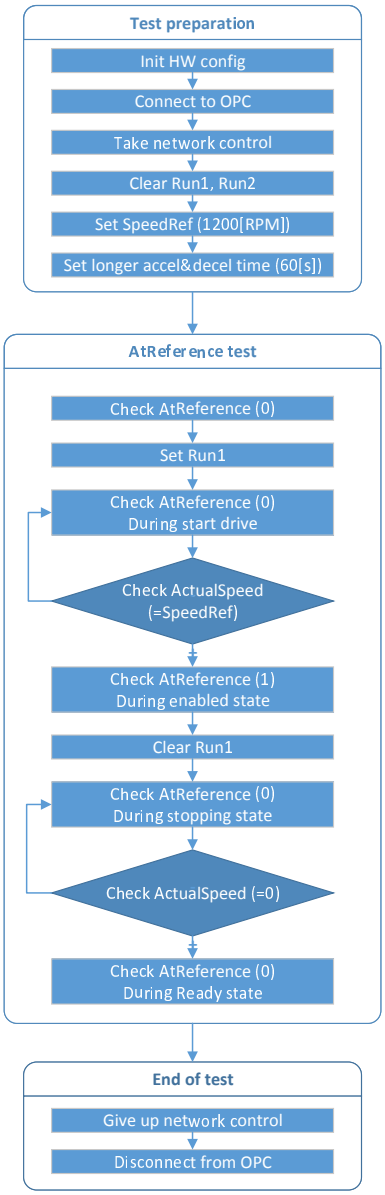
Obrázek C.4: Vývojový diagram GET testu

C.5 SET test



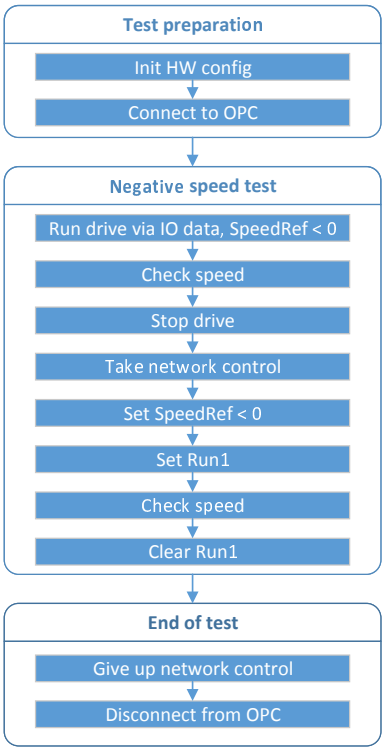
Obrázek C.5: Vývojový diagram SET testu

C.6 AtReference test



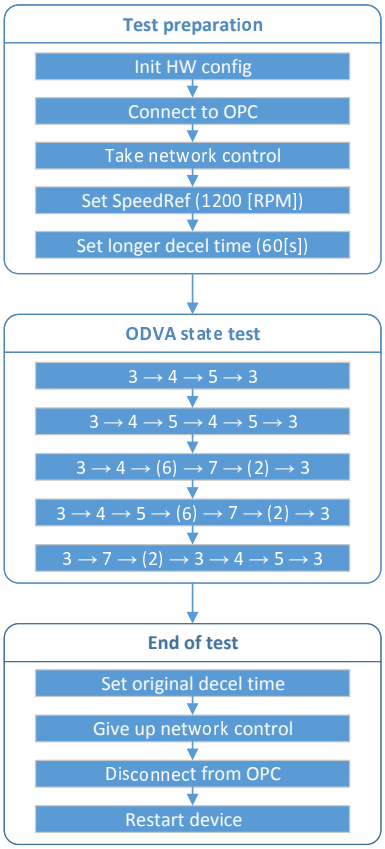
Obrázek C.6: Vývojový diagram testu atributu AtReference

C.7 Negative speed test



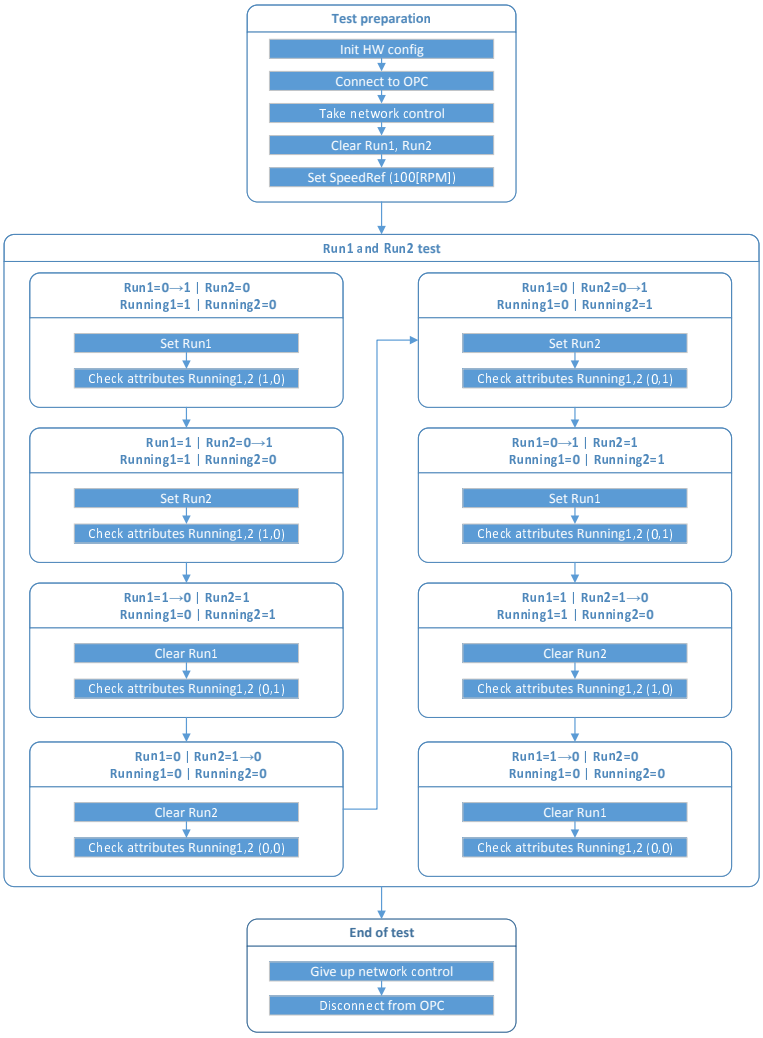
Obrázek C.7: Vývojový diagram testu záporné rychlosti

C.8 ODVA State test



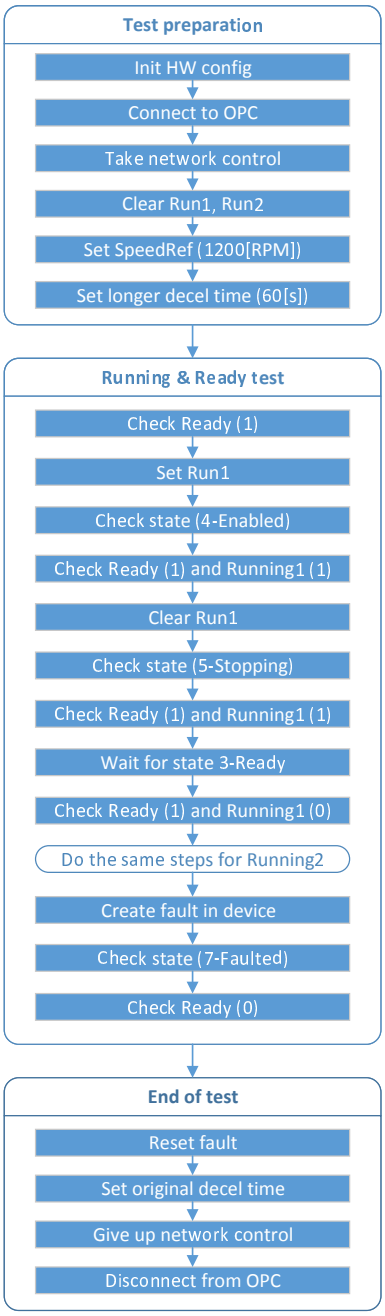
Obrázek C.8: Vývojový diagram ODVA stavové mašiny

C.9 Run1 & Run2 test



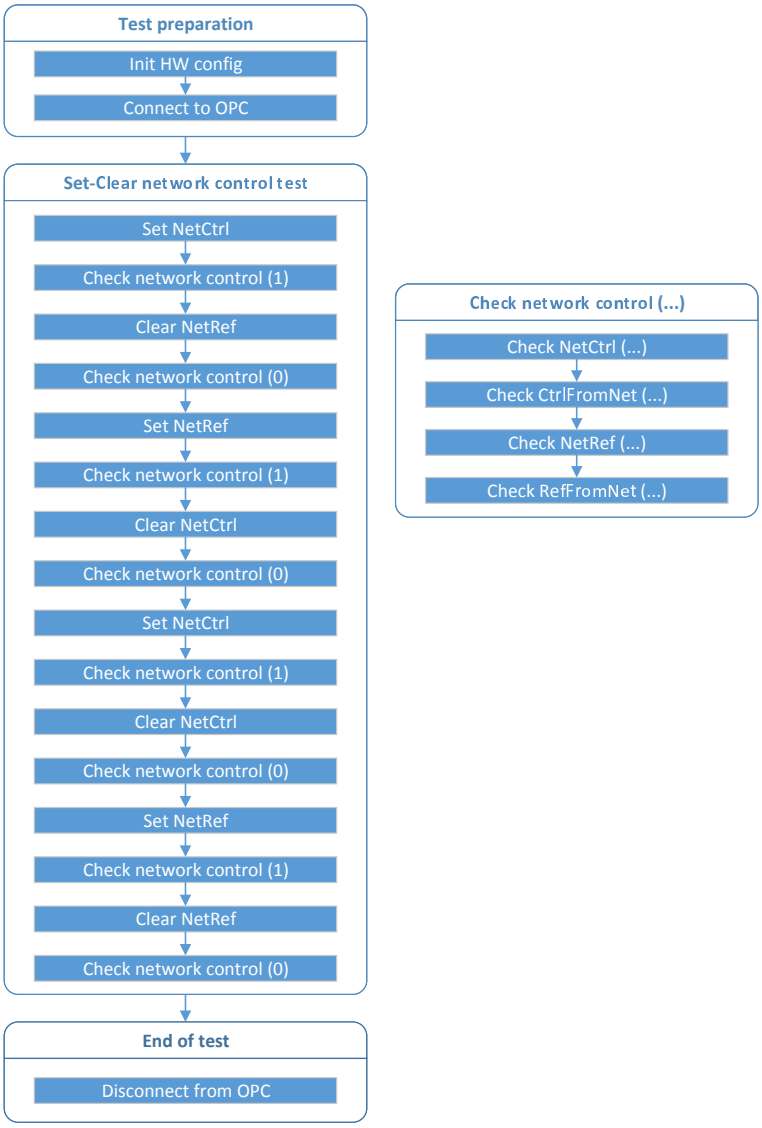
Obrázek C.9: Vývojový diagram testu attributů Run1 a Run2

C.10 Running & Ready test



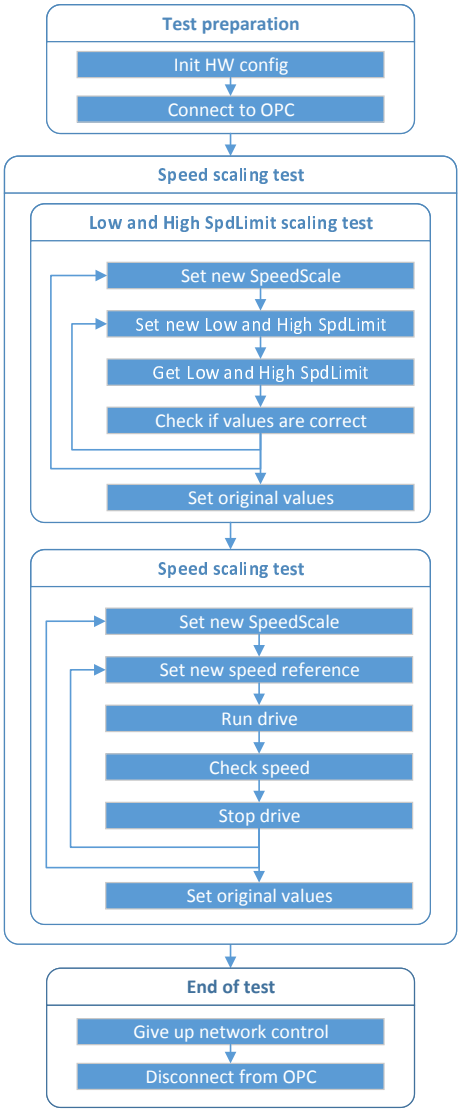
Obrázek C.10: Vývojový diagram testu attributů Running1, Running2 a Ready

C.11 Set-Clear network control test



Obrázek C.11: Vývojový diagram testu take and give up net. control

C.12 Speed scaling test



Obrázek C.12: Vývojový diagram testu speed scale test