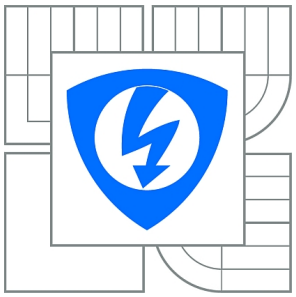
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF CONTROL AND INSTRUMENTATION

ACYKLICKÁ KOMUNIKACE PO PROFINETU MEZI MĚNIČI SINAMICS A PLC ŘADY SIMATIC S7-1200

ACYCLIC PROFINET COMMUNICATION BETWEEN SINAMICS CONVERTERS AND SIMATIC S7-1200 PLCS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. JIŘÍ DOSTÁL

AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN PÁSEK, CSc.

SUPERVISOR

BRNO 2013

VYSOKÉ UČENÍ
TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

Ústav automatizace a měřicí techniky

Diplomová práce

magisterský navazující studijní obor
Kybernetika, automatizace a měření

Student: Bc. Jiří Dostál ***ID:*** 106411
Ročník: 2 ***Akademický rok:*** 2012/13

NÁZEV TÉMATU:

Acyklická komunikace po Profinetu mezi měniči Sinamics a PLC řady Simatic S7-1200

POKYNY PRO VYPRACOVÁNÍ:

Vytvořte univerzální funkční blok pro vyčítání a zápis parametrů z a do měničů řady Sinamics po komunikační sběrnici Profinet. Využije se acyklický způsob komunikace, který trvale nealokuje prostor na sběrnici. Funkční blok by měl být univerzální a to jak pro jednoduché měniče řady Sinamics G tak pro víceosé měniče řady Sinamics S120.
Proveďte analýzu a případný vývoj univerzálního algoritmu, který by mohl strukturovat data pro systémové funkční bloky používané pro cyklickou komunikaci (Proxy FB) a umožnil tak implementovat obecné Proxy FB pro acyklickou komunikaci s měniči s jednoduchým rozhraním.
Vypracujte příkladovou aplikaci s použitím systémů SIMATIC S7-1200 a SINAMICS zaměřené na řízení typizované úlohy řízení rychlosti a polohy pohonu.

DOPORUČENÁ LITERATURA:

[1] Communication Function Blocks on PROFIBUS DP and PROFINET IO [online]. PROFIBUS Nutzerorganisation e.V., Listopad 2005 [cit. 2012-12-30] Dostupné z: <http://www.profibus.com.
[2] Siemens Industry Automation & Drive Technologies [online]. Siemens AG, 2012. Dostupné z: <https://www.cee.siemens.com

Termín zadání: 11.2.2013 ***Termín odevzdání:*** 20.5.2013

Vedoucí práce: Ing. Jan Pásek, CSc.
Konzultanti diplomové práce:

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Tato práce se zabývá acyklickou komunikací PLC Simatic S7-1200 a frekvenčních měničů Sinamics od společnosti Siemens. Práce popisuje možnosti propojení a způsoby komunikace těchto systémů po průmyslové sběrnici Profinet a vývoj univerzálních Proxy FB pro acyklické čtení a zápis parametrů měniče. Spolu s dalšími dvěma FB, které využívají acyklické komunikace pro čtení chybového zásobníku a pro nastavení absolutního snímače polohy, je na ukázkovém demokufru uvedena příkladová aplikace s vizualizací pro jednoduché polohové řízení a pro demonstraci použití vytvořených Proxy FB.

KLÍČOVÁ SLOVA

Acyklická komunikace, Profinet, Proxy FB, Simatic S7-1200, Sinamics

ABSTRACT

This paper deals with acyclic communication of Siemens Simatic S7-1200 PLCs and Sinamics drives on the Profinet industrial bus. Various possibilites of communication of the devices in accordance to the PROFIdrive profile is presented. General Proxy FBs for acyclic read/write of the drive's parameters are implemented. Together with other two FBs, that make use of acyclic communication to read the drive's fault buffer and to adjust an absolute encoder, an application with visualization for a simple positioning and for a demonstration of the Proxy FBs functionality is presented using a demo box.

KEYWORDS

Acyclic communication, Profinet, Proxy FB, Simatic S7-1200, Sinamics

DOSTÁL, Jiří. *Acyklická komunikace po Profinetu mezi měniči Sinamics a PLC řady Simatic S7-1200*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2013. 62 s. Vedoucí práce byl Ing. Jan Pásek, CSc.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Acyklická komunikace po Profinetu mezi měniči Sinamics a PLC řady Simatic S7-1200“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

(podpis autora)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplmové práce panu Ing. Janu Páskovi, CSc. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Vzájemná komunikace komponent průmyslové automatizace je nezbytná pro dobré fungování průmyslových systému. Důležitá je zejména interoperabilita systémů a podpora jednotných komunikačních standardů mezi různými platformami. Zákazníci u produktů průmyslové automatizace často vyžadují podporu co největšího množství komunikačních standardů a zároveň co nejjednodušší programování všech komponent, které zkracuje dobu jejich zprovoznění a tím snižuje náklady. Výrobci zabývající se průmyslovou automatizaci tak musí svoje produkty vybavovat co možná nejširším spektrem komunikačních systémů a zároveň poskytovat dostatečnou softwarovou podporu, aby bylo zprovoznění komunikace mezi jednotlivými zařízeními co nejjednodušší.

Softwarová podpora ve formě knihoven s tzv. *Proxy funkčními bloky* je jedním ze způsobů, jak usnadnit programování komunikace s daným zařízením a zvýšit tak jeho konkurenceschopnost.

Tato práce se zabývá implementací acyklické komunikace mezi frekvenčními měniči Sinamics a PLC Simatic S7-1200. Implementace acyklického zápisu/čtení jediného parametru měniče v současnosti vyžaduje poměrně značné úsilí, přestože by v principu mělo jít o velmi jednoduchou operaci. To je motivací k hledání nějakého řešení, které by tuto operaci co možná nejvíce zjednodušilo. Snahou je vytvořit obecné funkce, resp. funkční bloky, které bude možné jednoduchým způsobem zavolat z uživatelského programu, a které budou schopné samostatně provést veškeré nutné operace pro zapsání a vrácení požadovaných parametrů.

Teoretická část práce nejprve stručně představuje jeden z nejpoužívanějších systémů pro komunikaci v průmyslu – standard Profinet a dále popisuje velmi rozšířený komunikační profil pro průmyslové systémy s pohony PROFIdrive. Důraz je kladen na specifikace určené pro acyklickou komunikaci v rámci těchto standardů a na jejich implementaci v malých modulárních PLC Siemens S7-1200.

Praktická část práce se poté zabývá implementací univerzálních Proxy funkčních bloků pro acyklický zápis a čtení libovolných parametrů z a do měniče, které mají jednoduché vstupně/výstupní rozhraní a zároveň jsou dostatečně robustní.

K univerzálním funkčním blokům budou také implementovány dva bloky, které řeší konkrétní aplikaci založenou na acyklické komunikaci. K demonstraci výsledků má být vytvořena příkladová aplikace pro řízení typizované úlohy rychlosti a polohy pohonu. Tato aplikace bude implementována na poskytnutém ukázkovém *demokufru* s PLC Simatic S7-1200, frekvenčním měničem Sinamics S110 a malým dotykovým panelem.

1 POPIS POUŽITÝCH ZAŘÍZENÍ

1.1 Simatic S7-1212C

Simatic S7-1200 je označení pro řadu programovatelných logických automatů (Programmable Logic Controller – PLC), které se hodí především pro aplikace menšího rozsahu. Simatic PLC S7-1200 jsou kompaktní, modulární a poměrně levné PLC. Kompaktní jsou pro jejich malé rozměry a modulární pro jejich snadnou rozšířitelnost pomocí množství přídavných modulů, což je dělá vhodné pro použití v širokém spektru aplikací [5, 9].

PLC Simatic S7-1200 disponují především:

- integrovaným rozhraním Profinet pro programování, připojení HMI (Human–Machine Interface) či jiných zařízení;
- integrovanými technologickými funkcemi jako jsou čítače, časovače, PID řízení, a další;
- integrovanými binárními a analogovými vstupy a výstupy.

V tab. 1.1 jsou potom uvedeny vybrané parametry modelu z řady 1200 a to konkrétně S7-1212C DC/DC/DC, které je použito v praktické části tohoto projektu.

Tab. 1.1: Vybrané parametry PLC Simatic S7-1212C DC/DC/DC [9]

Parametr	Hodnota
Integrované binární I/O	8 vstupů 24 VDC / 6 výstupů 24 VDC
Integrované analogové I/O	2 vstupy 0-10 VDC
Max. počet binárních I/O	82
Max. počet analogových I/O	19
Integrovaná pracovní paměť	25 kB
Integrovaná pevná paměť	1 MB
Rychlost	0,1 µs / instrukci
Napájecí napětí	24 VDC

1.2 Sinamics S110

Sinamics S110 je nízkonapěťový, modulární frekvenční měnič pro jednoduché, standardní polohové úlohy. Jsou v něm integrovány všechny polohové funkce, takže s ním lze snadno řídit synchronní servomotory, ale i asynchronní motory s výkonem od

0,12 do 90 kW. V závislosti na použité řídicí jednotce disponuje vestavěnými komunikačními rozhraními s protokolem USS, Profibus a především Profinet. Lze ho tak snadno připojit k nadřazenému řídicímu systému ve formě PLC, jako je například výše zmíněný Simatic S7-1200. Krom toho má integrované binární vstupy a výstupy a analogové rozhraní ±10 V. Důležitou součástí jsou také integrované bezpečnostní funkce přes rozhraní Profibus či Profinet ve formě bezpečnostního profilu PROFIsafe, či bezpečnostních svorek [7].

Frekvenční měnič Sinamics S110 se skládá ze dvou základních komponent:

- výkonový modul PM340 v několika velkostech podle výkonu;
- řídicí jednotka CU305 ve třech provedeních podle komunikačního rozhraní (Profibus, Profinet nebo CAN) s volitelným ovládacím panelem.

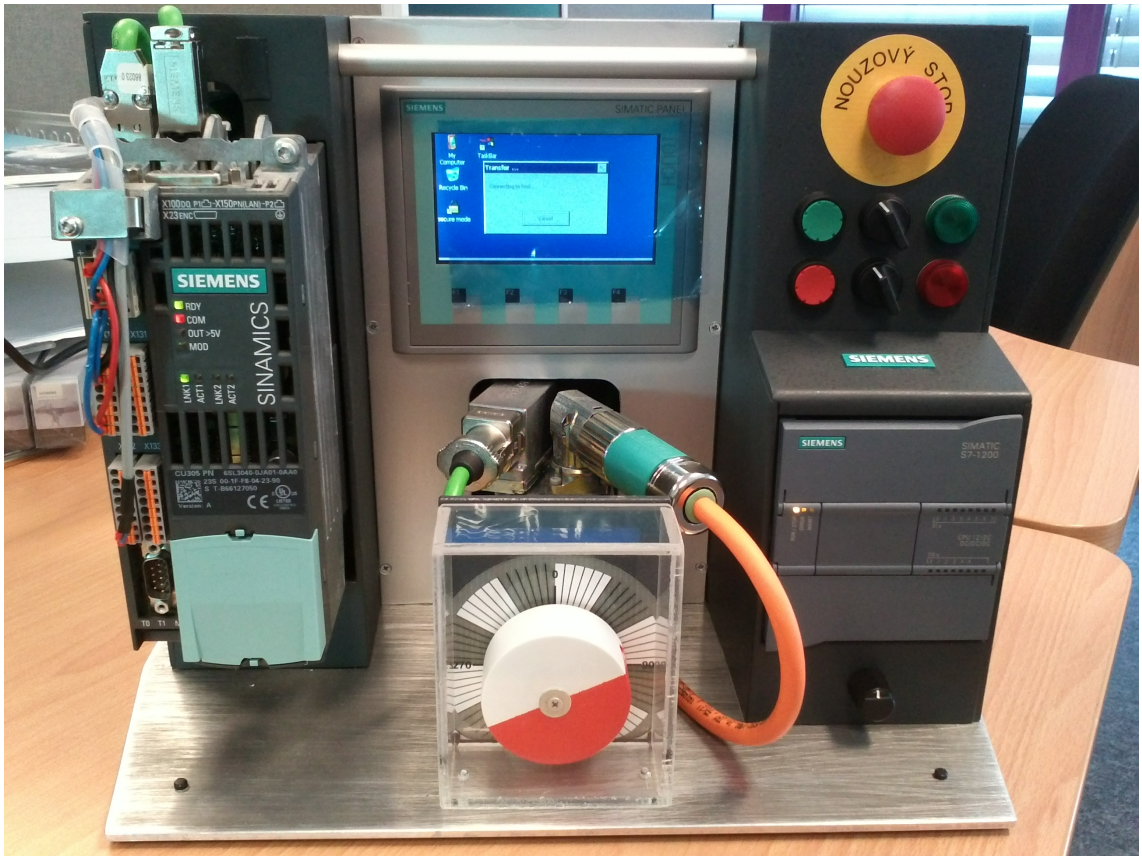
Pro připojení motoru a enkodéru lze použít rozhraní Drive-CLiQ, což je speciální otevřené rozhraní pro komunikaci mezi díly měničů Sinamics.

1.3 Demokufr Sinamics S110 + Simatic S7-1212C

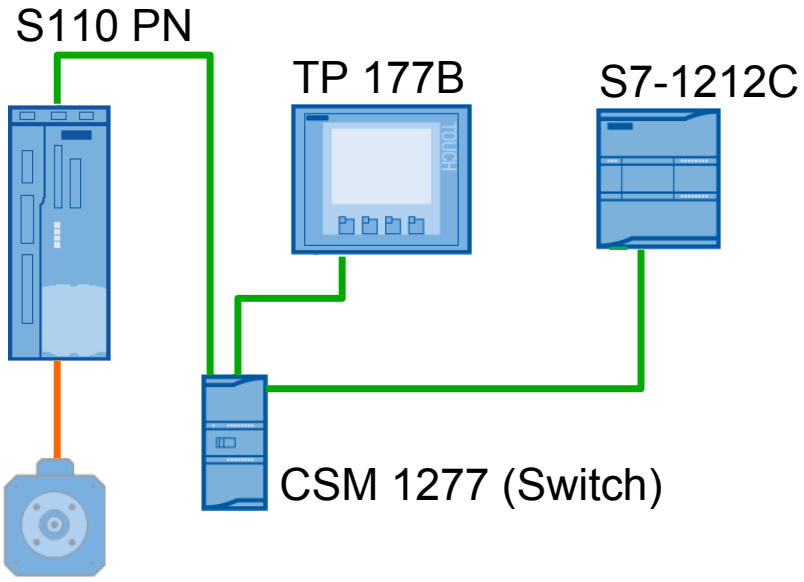
Ověření funkce pohonu Sinamics S110 ve spojení s PLC Simatic S7-1212 bylo provedeno na poskytnutém ukázkovém *demokufru*, který je na obr. 1.1. Součástí demokufru jsou následující komponenty:

- frekvenční měnič Sinamics S110:
  - výkonový modul PM340 (0,37 kW / 2,5 A);
  - synchronní servomotor 1FK7 s rozhraním DRIVE-CLiQ a enkodérem;
  - řídicí jednotka CU305 PN;
- PLC Simatic S7-1212C DC/DC/DC;
- dotykový panel SIMATIC TP177B 4” COLOR PN/DP;
- Profinet Switch CSM 1277;
- ovládací tlačítka, přepínače a indikátory.

Komponenty demokufru jsou propojeny sítí Profinet pomocí switche CSM 1277 viz schéma na obr. 1.2. Měnič s motorem je propojen rozhraním Drive-CLiQ.



Obr. 1.1: Demokufr Sinamics S110 + Simatic S7-1212C



Obr. 1.2: Schéma zapojení demokufru

2 KOMUNIKAČNÍ STANDARDY

2.1 Profinet

Termín Průmyslový Ethernet vyjadřuje použití Ethernetové komunikační sítě v průmyslovém prostředí pro řízení procesů v automatizaci. Vzhledem k použité technologii CSMA/CD (*Carrier Sense with Multiple Access and Collision Detection*) nedokáže Ethernet sám o sobě zaručit deterministickou Real-time (RT) odezvu, která je nezbytná pro průmyslové sběrnice. Z toho důvodu vzniklo množství standardů průmyslového Ethernetu, které pomocí různých mechanismů zaručují RT chování sběrnice založené na Ethernetu a specifikují modifikace vhodné pro průmyslové prostředí. Výhody použití průmyslové sběrnice založené na Ethernetu jsou zejména:

- velká přenosová rychlost až 1 Gbit/s;
- použití stejného média pro všechny vrstvy komunikace od nejnižší – procesní až po nejvyšší – kancelářskou;
- nízká cena zařízení vzhledem k rozšířenosti na poli informačních technologiích (internet).

Jedním z nejrozšířenějších standardů tohoto typu je právě Profinet. Profinet je otevřený standard vyvíjený mezinárodní organizací PROFIBUS & PROFINET International (PI) a je součástí standardů IEC 61158 a IEC 61784. Profinet umožňuje komunikaci použitím klasických protokolů TCP/IP a zároveň umožňuje komunikaci v reálném čase, která je nezbytná pro aplikace v průmyslové automatizaci, zejména pak pro řízení pohybu. Standard PROFINET definuje dva základní koncepty: Profinet CBA (*Component Based Automation*) a Profinet IO (*Input/Output*) [14].

Profinet CBA rozděluje celý automatizovaný systém na nezávislé pracující komponenty. Tyto komponenty jsou uloženy jako databáze souborů PCD, které je popisují pomocí jazyka XML. Projektování systému poté spočívá pouze v konfiguraci spojení jednotlivých komponent na základě databáze PCD a není tedy nutné komunikaci programovat. Profinet CBA je určen zejména pro Non real-time (NRT) komunikaci na vyšší úrovni automatizovaného systému [14].

Profinet IO je naopak uzpůsoben pro komunikaci v nejnižší procesní vrstvě systému. Profinet IO je realizací vlastního průmyslového Ethernetu tak, aby byl deterministický. Je zaměřen na rychlou komunikaci mezi distribuovanými IO zařízeními. Standard kompletně specifikuje způsob výměny dat mezi řídicími členy – tzv. Master (většinou se jedná o PLC) a periferiemi – tzv. Slave. Profinet IO definuje tři skupiny zařízení [12]:

- Profinet IO Controller – funkce Mastera, který řídí automatizační proces a komunikaci adresováním připojených zařízení.

- Profinet IO Device – periferní zařízení ve funkci Slave.
- Profinet IO Supervisor – Zařízení, které většinou slouží ke sledování procesu, diagnostice a nastavování parametrů. Jsou to např. zobrazovací panely, nebo PC.

Komunikace na síti Profinet probíhá třemi definovanými způsoby [12, 14]:

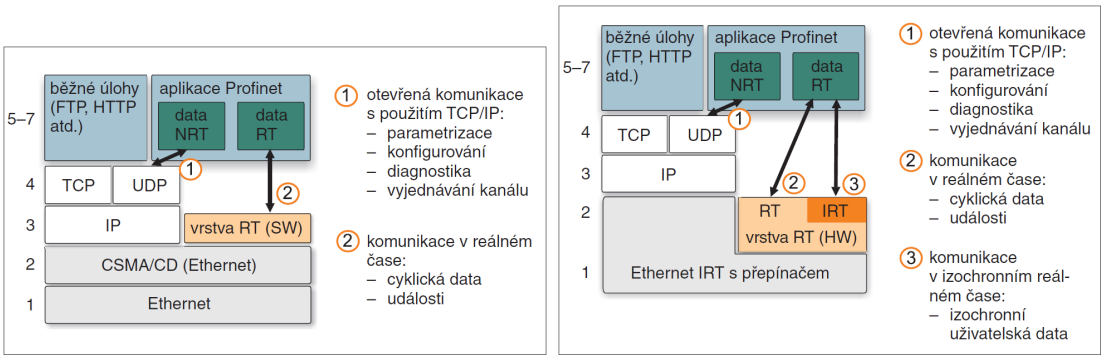
- Profinet NRT,
- Profinet RT,
- Profinet IRT.

2.1.1 Profinet NRT

Používá se pro acyklické, časové nekritické přenosy dat, jako jsou data pro parametrizaci, konfiguraci, vizualizaci a standardní „kancelářská“ data. Síť Profinet umožňuje využít část svého pásma pro přenos těchto dat pomocí standardních Ethernetových mechanismů použitím protokolů TCP/IP nebo UDP/IP. Přístroje tak mohou být propojeny s jakýmikoliv účastníky v prostředí standardní výpočetní techniky, což je jedna z hlavních výhod sítě Profinet [12, 14, 2]. Právě Profinet NRT je využíván navrženými bloky pro acyklickou komunikaci mezi měničem a PLC.

2.1.2 Profinet RT

Real-time komunikace s dobou odezvy v jednotkách ms, která zajišťuje, že přenos časově kritických dat je proveden v garantovaném časovém intervalu. RT odezva je zde zajištěna obejitím některých částí protokolů TCP/IP tak, jak je zobrazeno na obr. 2.1a, přičemž dvě nejnižší vrstvy ISO/OSI modelu zůstávají nezměněny, takže lze použít standardní Ethernetová zařízení.



(a) Profinet RT (b) Profinet IRT

Obr. 2.1: Komunikační model Profinet IO [14]

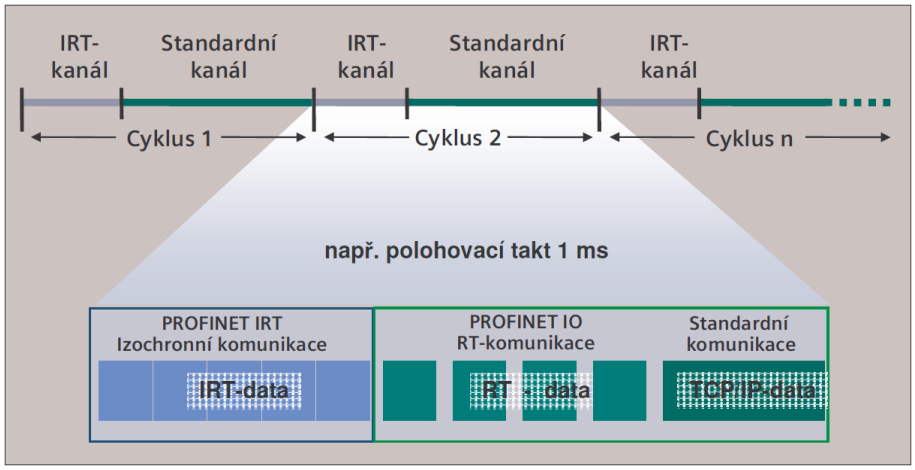
Profinet RT pokrývá kompletní řešení komunikace v Ethernetové síti [12, 8]:

- RT Komunikace v rámci jedné sítě, kde se nepoužívá adresa IP ani informace TCP, UDP.
- RT Komunikace mezi sítěmi s použitím protokolu UDP/IP a profilu „RT over UDP“.
- RT multicasting s použitím protokolu UDP/IP a profilu „RT over UDP“, kde se vysílá do většího počtu uzlů najednou.

2.1.3 Profinet IRT

Obzvlášť v oblasti řízení pohonů, kde je nutná velmi rychlá komunikace např. pro synchronizaci několika polohovacích systémů, implementuje Profinet IO řešení pomocí tzv. Isochronous real-time (IRT), jehož struktura je znázorněna na obr. 2.1b. Jak je vidět z obrázku, tento způsob komunikace již vyžaduje speciální hardware [2, 12].

Princip IRT spočívá v rozdělení komunikačního cyklu na deterministickou a otevřenou část viz obr. 2.2. Časově kritická data jsou přenášena v deterministickém kanálu, který je časově synchronizován s použitím speciálního hardware jednotlivých účastníků sítě. Tímto lze dosáhnout doby cyklu v řádu stovek µs a jitteru do 1 µs [2, 8].



Obr. 2.2: Rozdělení komunikačního cyklu Profinet [2]

Velkou výhodou Profinetu je to, že lze společně na jednom médiu paralelně přenášet jak RT (IRT) tak i jakákoliv kancelářská NRT data, aniž by se vzájemně ovlivňovaly, nebo vylučovaly.

2.2 PROFIdrive

Technologie pohonů je jedna z nejdůležitějších součástí průmyslové automatizace. Důkazem je například to, že pohony v průmyslu spotřebovávají v průměru 60% elektrické energie. Technologie PROFIdrive je standardizovaný profil (standard IEC 61800-7) pro systémy pohonů, který je implementován pro průmyslové sběrnice Profinet a Profibus a taktéž je vyvíjen organizací PI. [3]

2.2.1 Aplikační model

Aplikace pro řízení pohonů dle profilu PROFIdrive se skládá z těchto základních procesů [3]:

- Řídicí proces v měniči, což je obvykle proudové řízení motoru a řízení rychlosti, případně polohy.
- Řídicí proces v PLC, což je obvykle výpočet žádané hodnoty rychlosti, či polohy.
- Komunikační systém mezi PLC a měničem, který přenáší potřebná data pro oba řídicí procesy v závislosti na zvoleném způsobu řízení.

2.2.2 Třídy zařízení

V podstatě stejně jako standard Profinet rozděluje PROFIdrive zařízení do třech tříd: Controller – typicky PLC, Supervisor – zařízení pro konfiguraci, monitoring, operátorské řízení a tzv. P-Device – Periferní zařízení, typicky frekvenční měnič s jednou nebo více pohonnými jednotkami, které se označují DO (Drive Object), viz obr. 2.3 [3].

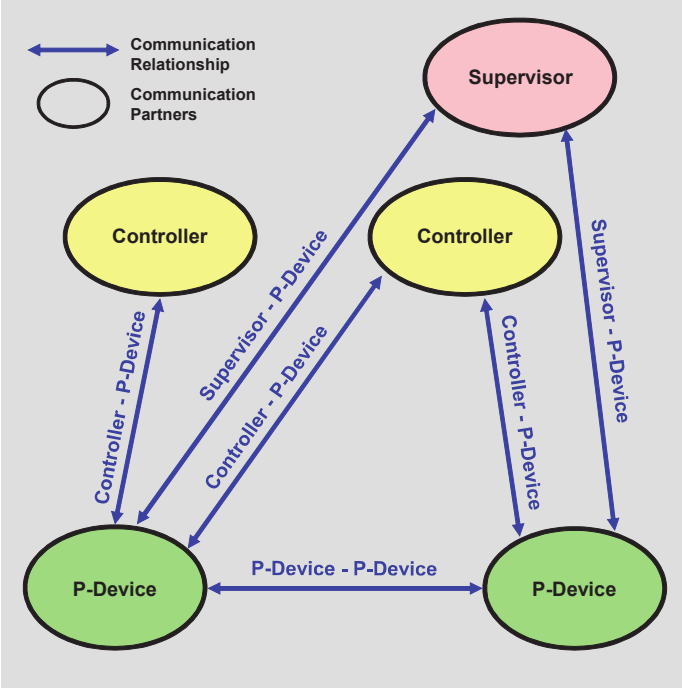
2.2.3 Komunikace dle PROFIdrive

Cyklická komunikace

Během řízení pohonů v uzavřené či otevřené smyčce musí řídicí systém pracovat kontinuálně. Nezávisle na způsobu řízení, data jako žádaná hodnota, akční veličina či aktuální měřené výstupní veličiny musí být přenášeny cyklicky. Podle způsobu řízení a náročnosti aplikace lze použít pro cyklickou komunikaci buď neizochronní RT kanál nebo izochronní IRT kanál [3].

Acyklická komunikace

Kromě výše zmíněných procesních dat, které musí být přenášeny cyklicky, je častým požadavkem přenos dalších dat pro parametrizaci měniče a čtení různých hodnot



Obr. 2.3: Třídy zařízení a vztahy mezi nimi [3]

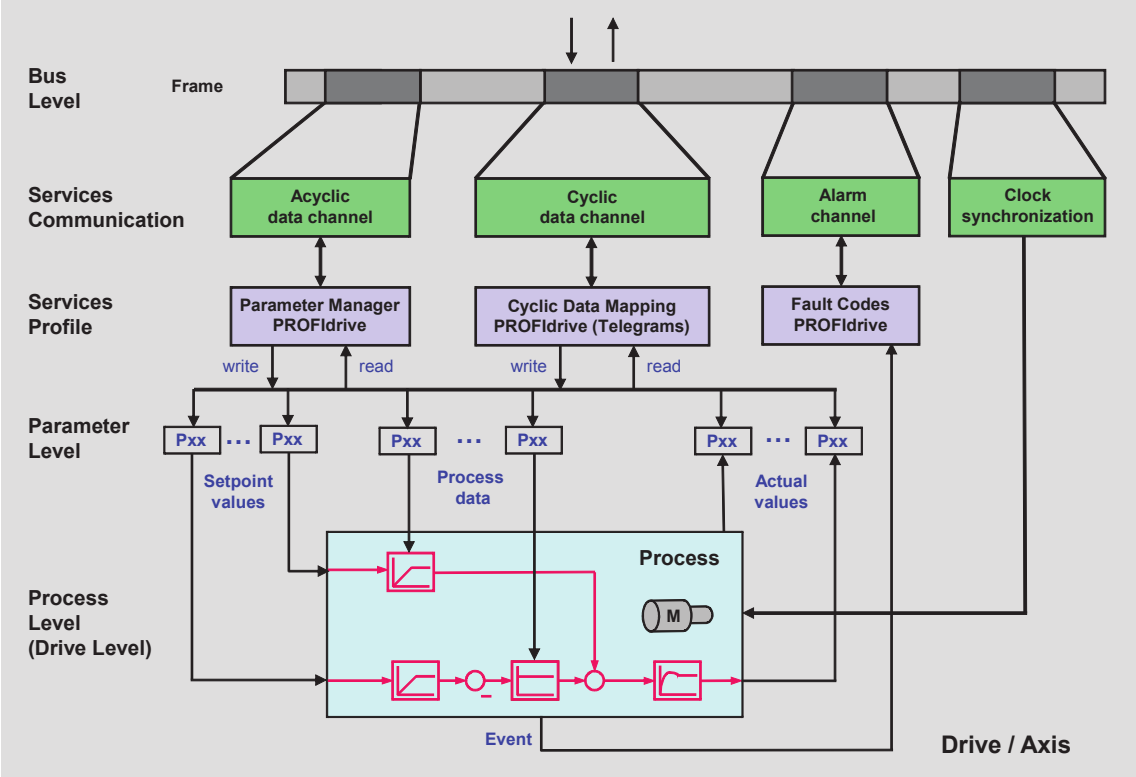
nepřímo souvisejících s řídicí aplikací. Přenos těchto dat není časově kritický a často se také přenáší pouze jedenkrát například při spuštění procesu [3].

Dále standard specifikuje kanál alarmů, který je řízen událostmi a používá se k diagnostice chybových stavů a k jejich kvitování a kanál pro izochronní data a hodinový signál pro synchronizaci při nutnosti použití izochronního přenosu dat. Zjednodušené schéma všech komunikačních kanálů vystihuje obr. 2.4 [3].

2.2.4 Stavový diagram a aplikační třídy měniče

Moderní frekvenční měniče disponují řadou pokročilých funkcí, které mohou vykonávat nezávisle na nadřazeném řídicím systému. Standard PROFIdrive definuje šest následujících aplikačních tříd podle různých funkcí DO měniče [3, 8]:

- *Aplikační třída 1 – Standardní pohon.* Nejjednodušší případ, kdy Controller zadává žádanou hodnotu otáček a otáčkové řízení je implementováno pohonem.
- *Aplikační třída 2 – Standardní pohon s přídavnou funkcí.* Kromě otáčkového řízení zajišťují samotné pohony dílčí automatizační funkce.
- *Aplikační třída 3 – Polohovací zařízení.* Pohon implementuje navíc polohové řízení. Controller zadává žádanou hodnotu polohy.
- *Aplikační třída 4 – Centrální polohové řízení.* Pohon implementuje otáčkové řízení a Controller polohové řízení. Regulační smyčka je uzavřená pomocí komunikační sběrnice, což vyžaduje IRT komunikaci.



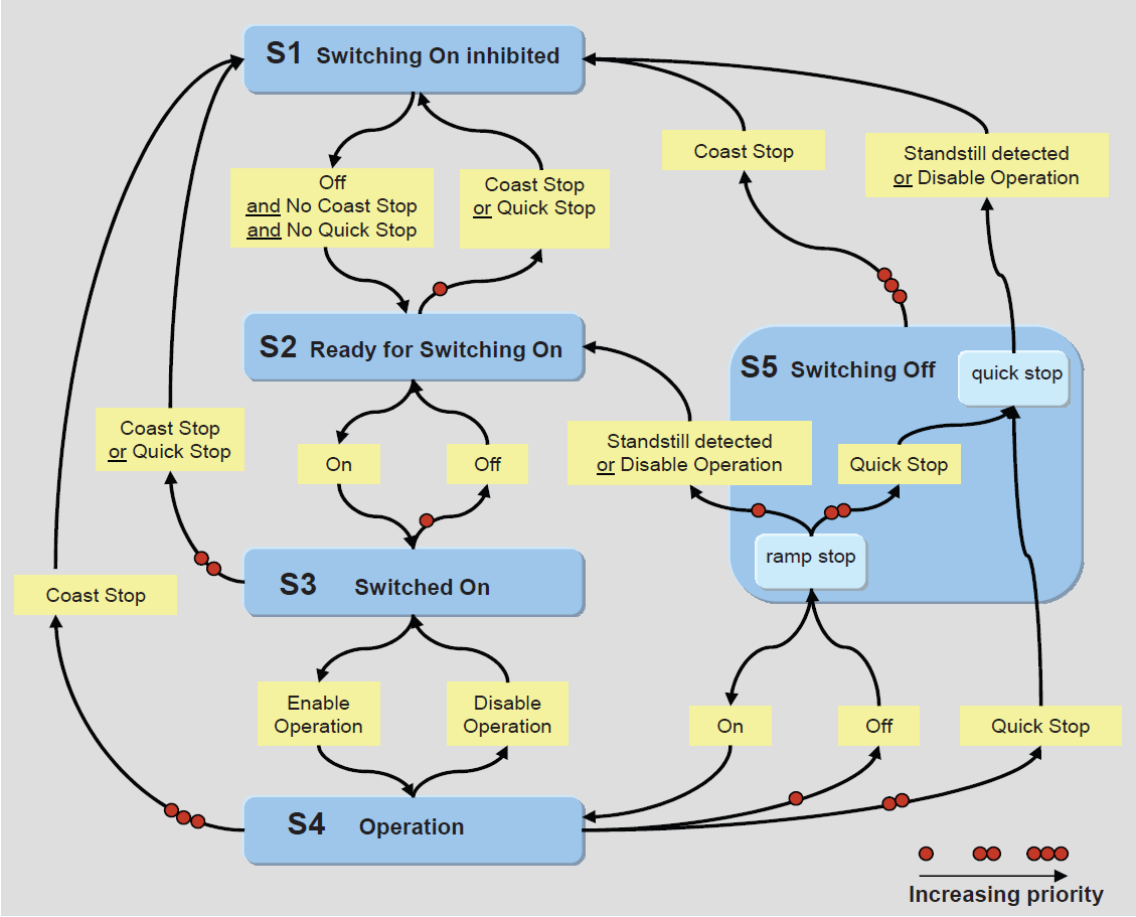
Obr. 2.4: Komunikační model pohonné jednotky [3]

- *Aplikační třída 5 – Centrální otáčkové řízení.* Stejné jako třída 4, ale místo polohové smyčky je přes sběrnici uzavřena smyčka otáčková.
- *Aplikační třída 6 – Decentrální řízení.* Komunikace probíhá také přímo mezi jednotlivými pohony. Uplatnění např. pro elektronické vačky a podobné aplikace.

Tyto aplikační třídy vlastně nepřímo definují komunikaci mezi PLC a frekvenčním měničem.

Pro všechny uvedené aplikační třídy platí jednotný stavový diagram, který je znázorněn na obr. 2.5 a který definuje všechny stavy DO a přechody mezi nimi. Tento stavový diagram je důležitý pro správné navržení řízení měniče zejména při spuštění aplikace a při kvitování alarmů s následným opětovným spuštěním pohonu [3].

Žlutě jsou zde vyznačeny řídicí signály z nadřazeného zařízení, obvykle PLC, kterými lze ovládat přechody mezi stavy S1–S5. Červené tečky na přechodech diagramu vyznačují prioritu přechodu v případě, že jsou v jeden okamžik splněny podmínky přechodu do dvou různých stavů.



Obr. 2.5: Základní stavový diagram DO [3]

2.2.5 Parametrový model PROFIdrive

Velice důležitou součástí pro pochopení principu řízení frekvenčních měničů dle standardu PROFIdrive je parametrový model. Každý frekvenční měnič je logicky uspořádán do několika funkčních modulů, jejichž vstupy a výstupy jsou vnitřně reprezentovány parametry. Parametrem je doslova každá proměnná i statická hodnota, se kterou měnič pracuje a která ho definuje. Je jím aktuální napětí motoru, žádaná hodnota, parametrem je také číslo aktuální poruchy měniče i identifikační číslo výrobce měniče či IP adresa měniče. Každý parametr v rámci celého frekvenčního měniče má své jedinečné číslo, které ho identifikuje. Parametry číslo 900 – 999 jsou tzv. PROFIdrive profilem specifikované parametry, které mají stejný význam pro všechny měniče nezávisle na aplikační třídě. Ostatní jsou výrobcem specifikované a mohou se u jednotlivých výrobců a zařízení lišit. Některé parametry jsou určeny pouze ke čtení (zpravidla se označují `rXXXX`), ostatní lze číst i zapisovat (označení `pXXXX`). Kromě toho mohou mít různý datový typ a délku a mohou obsahovat i pole hodnot [8] [3].

Parametrový model vyjadřuje to, že frekvenční měnič je pouze komplexní soustava parametrů, které mají jasně definovaný význam a funkce měniče je tak zcela definována hodnotami a vzájemným propojením těchto parametrů.

Na obrázku 2.4 je vidět, že veškerá komunikace s frekvenčním měničem dle PROFIdrive probíhá prostřednictvím zápisu nebo čtení vybraných parametrů. Při použití cyklické komunikace, jsou definovány tzv. standardní telegramy, které určují, jaké parametry jsou předmětem výměny, resp. přiřazují parametry v měniči jednotlivým hodnotám v dané zprávě. Pomocí telegramů lze tedy přistupovat pouze k omezenému množství parametrů měniče. Pro většinu aplikací je ovšem tato množina dostačující, neboť obsahuje všechny parametry, které je nutné přenášet cyklicky v jednotlivých aplikačních třídách zmíněných v kap. 2.2.4. Požadujeme-li čtení či zápis dalších parametrů nespecifikovaných v použitém telegramu, můžeme ojediněle využít buď volných bitů a bytů v rámci daného telegramu, nebo definovat vlastní volný telegram, kde při konfiguraci měniče přiřadíme jednotlivá slova ve zprávě zvoleným parametrům. Většinou se ale jedná o parametry, které není nutno přenášet nepřetržitě cyklicky. Proto se toto řešení příliš často nepoužívá a mnohem vhodnější je použít řešení pomocí **acyklické** komunikace [3].

Obvykle pro procesní data, jako je žádaná a aktuální hodnota, použijeme některý ze standardních nebo výrobcem definovaných telegramů a pro čtení či zápis dalších parametrů použijeme funkce pro acyklickou komunikaci, která trvale nealokuje prostor na komunikační sběrnici.

3 KOMUNIKACE FREKVENČNÍCH MĚNIČŮ SINAMICS A PLC SIMATIC S7-1200

Při programování PLC Simatic S7-1200 lze využít následujících způsobů komunikace s frekvenčním měničem Sinamics [1]:

- Typické řešení je cyklická komunikace, kde využijeme tzv. obraz procesu (process image). Správnou konfigurací PLC lze dosáhnout toho, že vzdálené parametry měniče jsou mapovány na zvolené místo v obrazu procesu a v programu k nim potom přistupujeme stejně jako k běžným vstupním a výstupním operandům v systémové paměti. Výměna dat přes Profinet probíhá cyklicky se zvolenou periodou nezávisle na vlastním programu PLC.
- Alternativní řešení je použití systémových funkčních bloků přímo ve vlastním programu. Standardní knihovna bloků PLC Simatic S7-1200 obsahuje systémové funkční bloky pro acyklickou výměnu procesních dat, které jsou definovány v [1] jako univerzální komunikační funkční bloky pro komunikaci mezi programovatelnými logickými automaty a periferními zařízeními v sítích PROFIBUS DP a PROFINET IO.

Cyklický i acyklický způsob komunikace probíhá paralelně.

3.1 Cyklická komunikace s použitím standardních telegramů

Ke komunikaci s měniči Sinamics podle standardu PROFIdrive se používají tzv. telegramy. Telegramy jednoznačně definují význam jednotlivých bytů, případně bitů v souboru dat přenášených po sběrnici, a tak jsou tato data pro obě komunikující strany srozumitelná. Standard PROFIdrive definuje vlastní tzv. standardní telegramy, které lze použít ke komunikaci s jakýmkoliv zařízením implementujícím tento profil a tudíž jsou multiplatformní. Kromě těchto standardních telegramů, které lze s měniči Sinamics bez problémů použít, jsou také definovány vlastní, výrobcem specifikované telegramy, které navíc obsahují některé specifické parametry [3].

Komunikace využitím telegramů je výhradně **cyklická komunikace** pro přenášení základních dat dle aplikačních tříd PROFIdrive.

V tab. 3.1 je přehled všech telegramů, které lze ve spojení s měničem Sinamics S110 použít. Telegramy s označením menším než 100 jsou standardní telegramy, ostatní telegramy jsou specifikované výrobcem. Poslední sloupec tabulky ukazuje, pro jakou konfiguraci měniče vzhledem k aplikační třídě dle standardu PROFIdrive jsou dané telegramy určeny [8].

Tab. 3.1: Přehled telegramů pro měnič Sinamics S110 [8]

		Aplikační třída			
Telegram	Význam	1	2	3	4
1	Žádaná hodnota rychlosti 16-bit	×	×		
2	Žádaná hodnota rychlosti 32-bit	×	×		
3	Žádaná hodnota rychlosti 32-bit s polohovým enkodérem		×		×
4	Žádaná hodnota rychlosti 32-bit s 2 polohovými enkodéry				×
7	Základní polohování			×	
9	Základní polohování s přímým vstupem			×	
102	Základní polohování s polohovým enkodérem a redukcí momentu				×
103	Základní polohování s 2 polohovými enkodéry a redukcí momentu				×
110	Základní polohování s MDI, přepsáním hodnot a XIST_A			×	
111	Základní polohování v MDI módu			×	
390	Řídicí jednotka s DI/DO	×	×	×	×
391	Řídicí jednotka s DI/DO a 2 měřicími sondami	×	×	×	×
392	Řídicí jednotka s DI/DO a 4 měřicími sondami	×	×	×	×
393	Řídicí jednotka s DI/DO, AI a 4 měřicími sondami	×	×	×	×
394	Řídicí jednotka s DI/DO	×	×	×	×
999	Volný telegram	×	×	×	×

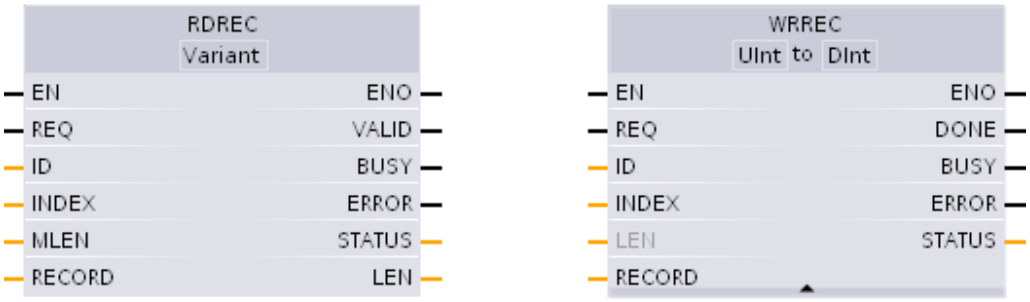
3.2 Acyklická komunikace použitím systémových funkčních bloků

Systémové funkční bloky SFB jsou funkční bloky integrované do operačního systému PLC. V prostředí Step7 je lze najít na kartě instrukcí a volat je stejným způsobem jako běžné FB.

Pro acyklickou výměnu dat mezi PLC a periferními zařízeními slouží systémové funkční bloky RDREC (Read Process Data Record) a WRREC (Write Process Data Record). Na obr. 3.1 jsou znázorněny tyto bloky a jejich rozhraní (vstupy a výstupy) [13] [10].

Datový typ a význam vstupů a výstupů bloků WRREC a RDREC udává tab. 3.2 [13] [10] [1]

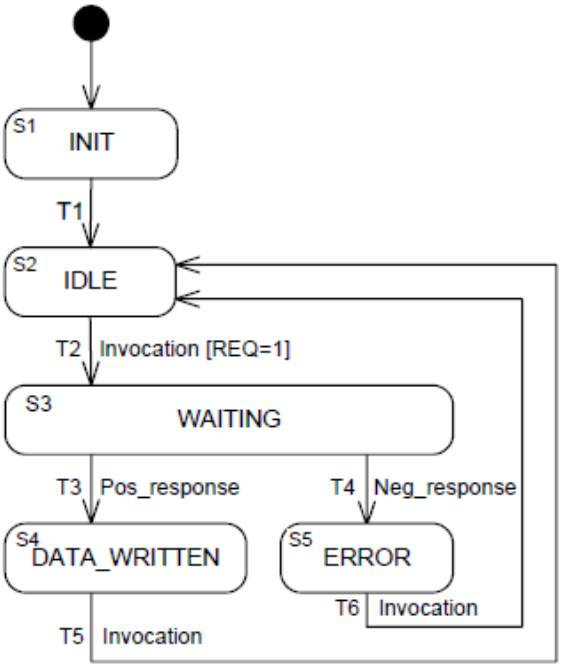
Oba funkční bloky WRREC i RDREC pracují asynchronně, tzn. že jejich volání a vykonání může trvat několik programových cyklů. Volání bloků nelze řetězit a k jejich použití je nutné vytvořit algoritmus dle stavového diagramu na obr. 3.2 a 3.3. Je nutné sledovat výstupní hodnoty BUSY, DONE a ERROR, které definují přechody mezi jednotlivými stavy funkčních bloků. Na obr. 3.4 a 3.5 jsou znázorněny



Obr. 3.1: Systémové funkční bloky RDREC a WRREC

možné způsoby volání daných funkčních bloků pomocí signálů REQ, BUSY, DONE a ERROR [1]:

1. Signál REQ zůstane v log. 1 dokud funkce není úspěšně provedena, neboli je resetován až když se signál DONE (VALID) změní na log. 1.
2. Funkce je volána pulsem signálu REQ. Resetování signálu REQ nezpůsobí zrušení volání funkce, ovšem dokud vykonání funkce není dokončeno, nemá další změna signálu REQ žádný vliv (funkce nebude znovu zavolána).
3. Stejné jako v případě 1, ale místo úspěšného provedení funkce je výsledek chyba.



Obr. 3.2: Stavový diagram funkčního bloku WRREC [1].

Tab. 3.2: Vstupy a výstupy WRREC a RDREC [13] [10].

Název	Datový typ	Popis
REQ	BOOL	Funkce je zavolána pokud REQ = 1.
ID	HW_IO (WORD)	Logická adresa periferního zařízení.
INDEX	INT	Číslo souboru dat.
MLEN	UINT	Maximální délka dat čtení v bytech.
LEN	UINT	Délka zapisovaných či přečtených dat v bytech.
DONE	BOOL	DONE = 1 indikuje úspěšné provedení zápisu dat.
VALID	BOOL	VALID = 1 indikuje, že data byla přijata a jsou platná.
BUSY	BOOL	BUSY = 1 indikuje probíhající zápis nebo čtení dat.
ERROR	BOOL	ERROR = 1 indikuje chybu při zápisu nebo čtení dat.
STATUS	DWORD	V případě chyby obsahuje informace o chybě.
RECORD	VARIANT	Data pro zápis nebo cílová paměť pro přijatá data při čtení. Jedná se o ukazatel na souvislý blok dat, jehož struktura je popsána v 3.2.3.

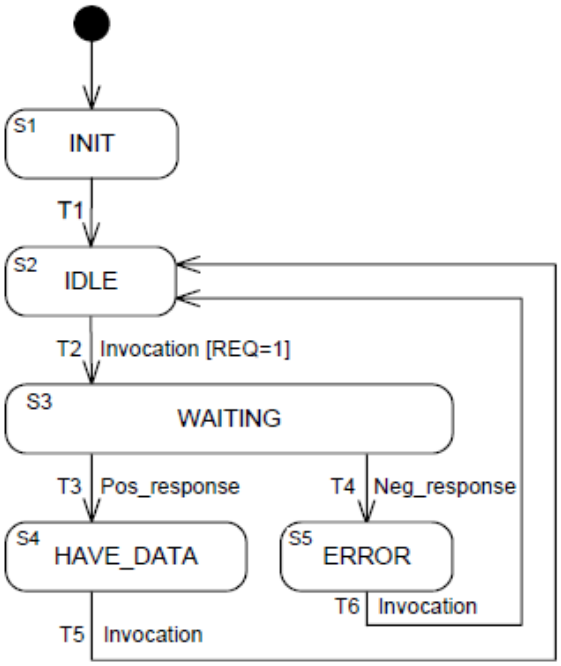
3.2.1 Zápis parametrů

Abychom mohli použít funkční blok WRREC pro acyklický zápis parametrů do měniče, musíme nejprve vytvořit datovou strukturu, která bude zápis definovat. To uděláme vytvořením datového bloku pro zápis parametrů, jehož struktura je definována na obr. 3.6. Poté se na tento datový blok odkážeme ukazatelem typu VARIANT při volání SFB WRREC. Je možné zapisovat více parametrů zároveň, a to maximálně tolik, aby celková velikost datového bloku pro zápis včetně hlavičky nepřesáhla 240 bytů [8].

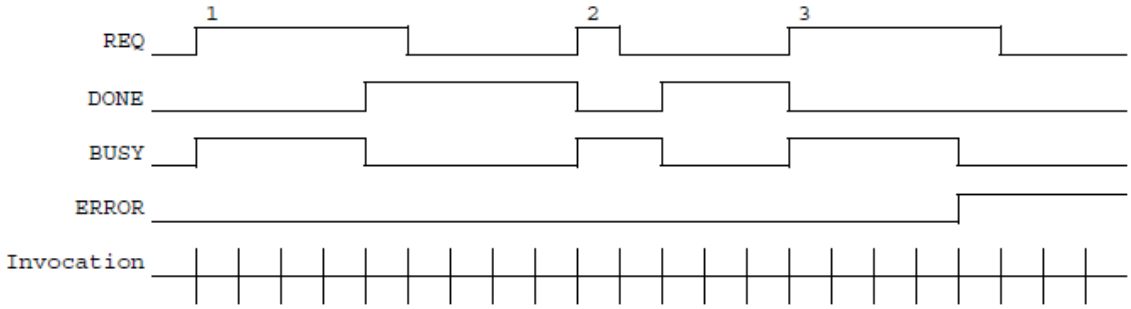
3.2.2 Čtení parametrů

Čtení parametrů z měniče se skládá ze dvou po sobě jdoucích kroků:

1. Volání funkčního bloku WRREC s datovým blokem definujícím požadavek na čtení parametrů, viz obr. 3.6.



Obr. 3.3: Stavový diagram funkčního bloku RDREC [1].

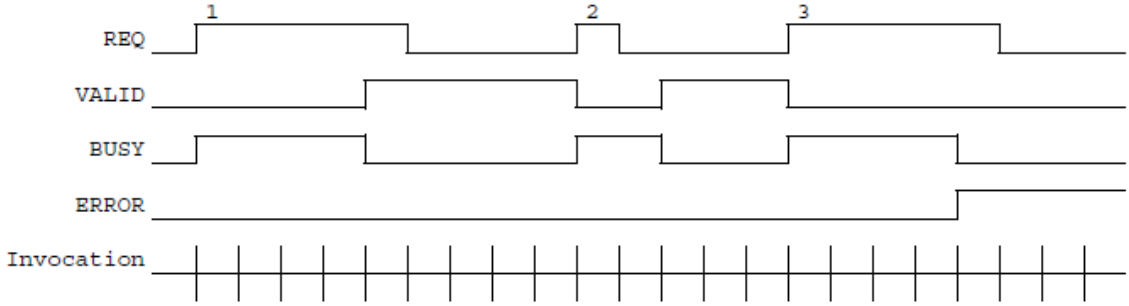


Obr. 3.4: Časový diagram funkčního bloku WRREC [1].

2. Volání funkčního bloku RDREC, který vrátí data ve struktuře dle obr. 3.7.

Pro čtení parametrů tedy definujeme dva datové bloky. První je podobný datovému bloku pro zápis parametrů a je použit při volání WRREC, druhý potom slouží jako paměť pro uložení odpovědi vrácené blokem RDREC a pro rozklíčování odpovědi je nutné jeho strukturu definovat způsobem odpovídajícím požadavku čtení v prvním datovém bloku.

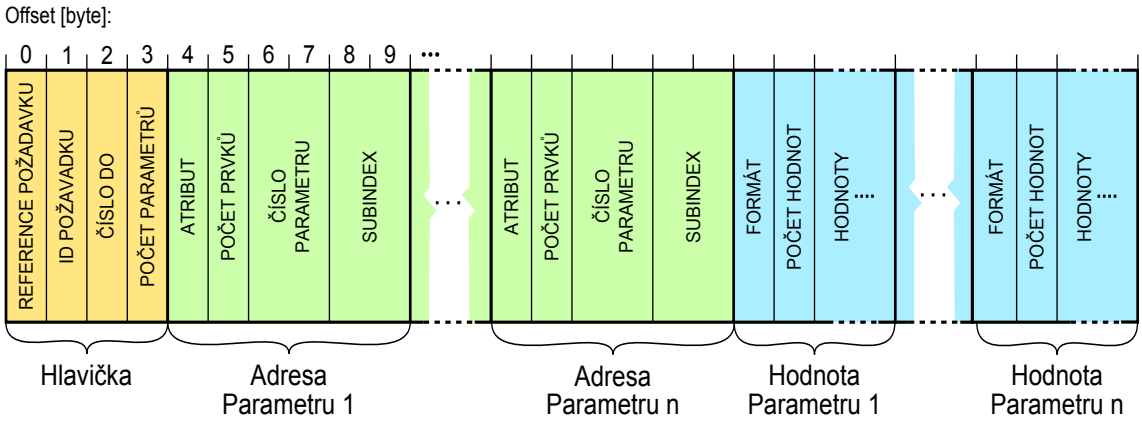
Celý proces čtení parametrů opět nelze řetězit, tzn. že nový požadavek na čtení je možné volat až poté, co měnič vyřídí předchozí požadavek. Jinými slovy jsou postupně dokončeny oba kroky čtení dle výše uvedeného výčtu (každý je dokončen dle obr. 3.4 a 3.5), případně je čtení ukončeno chybou [1].



Obr. 3.5: Časový diagram funkčního bloku RDREC [1].

3.2.3 Struktura datových bloků pro WRREC a RDREC

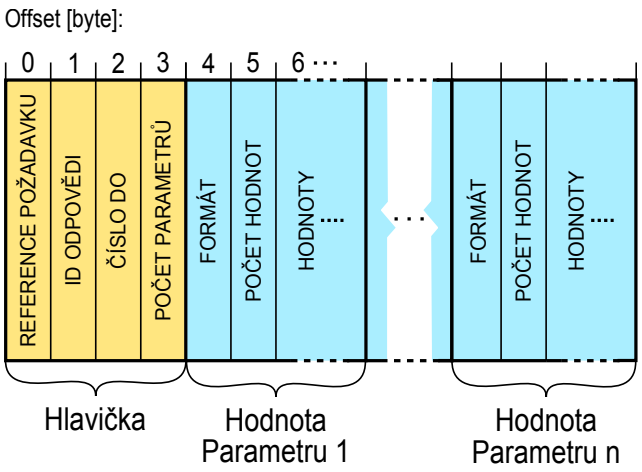
Systémové funkční bloky WRREC a RDREC jsou pouze univerzální komunikační funkce, které zajišťují asynchronní zápis a čtení dat z periferních zařízení po sběrnicích PROFIBUS DP nebo PROFINET IO. Vlastní informaci a definici, co daná zpráva znamená, určují samotná data, která jsou přenášena (data daná parametrem RECORD funkčních boků WRREC a RDREC). Tento blok dat má přesně daný formát, který je specifikován v dokumentaci periferního zařízení, v našem případě v dokumentaci frekvenčního měniče.



Obr. 3.6: Struktura dat pro vstup RECORD bloku WRREC.

Struktura dat přenášených funkčním blokem WRREC do frekvenčních měničů Sinamics je schematicky znázorněna na obr. 3.6. Struktura dat přenášených z měniče pomocí RDREC je znázorněna na obr. 3.7. Význam jednotlivých položek v těchto schématech je uveden dále v tab. 3.3. Nejdůležitější vlastnosti přenášených dat jsou [8]:

- Data jsou strukturována do hlavičky a těla.
- Parametry a indexy jsou adresovány 16-bitovou hodnotou.



Obr. 3.7: Struktura dat výstupu RECORD bloku RDREC.

- V jednom požadavku je možné přenášet více parametrů zároveň (limitováno maximální délkou zprávy 240 bytů).
- U parametrů typu pole je možný přenos buď celého pole, nebo pouze zvolené části pole.
- Je možný pouze jeden požadavek na čtení nebo zápis v daném čase (požadavky nelze řetězit).
- Délka zprávy včetně hlavičky nesmí přesáhnout 240 bytů.

Tab. 3.3: Význam položek v datových blocích pro WRREC a RDREC [8]

Položka	Datový typ	Hodnoty [hex]
Reference požadavku	unsigned8	0x01 . . . 0xFF
	Referenční číslo požadavku a odpovědi, které je jedinečné pro každý požadavek/odpověď a tím je identifikuje. PLC toto číslo mění s každým požadavkem a měnič jej kopíruje do odpovědi odeslané zpět, aby bylo dohledatelné, která odpověď patří kterému požadavku.	
ID požadavku	unsigned8	0x01 – Požadavek na čtení 0x02 – Požadavek na zápis
	Určuje typ požadavku	
ID Odpovědi	unsigned8	0x01 – Odpověď na úspěšné čtení 0x02 – Odpověď na úspěšný zápis 0x81 – Odpověď na neúspěšné čtení 0x82 – Odpověď na neúspěšný zápis

	Odpověď měniče kopíruje ID požadavku nebo udává výskyt chyby při zápisu nebo čtení parametrů. V případě chyby jsou v oblasti s hodnotami přenášeny chybové hodnoty.	
Číslo DO	unsigned8	0x00 . . . 0xFF Udává číslo DO (Objekt pohonu), na který je směřován daný požadavek.
Počet parametrů	unsigned8	0x01 . . . 0x27 – Omezeno maximální délkou zprávy. Udává počet parametrů pro zápis nebo čtení.
Atribut	unsigned8	0x10 – Hodnota 0x20 – Popis 0x30 – Text (není implementováno) Udává jaká složka parametru je předmětem požadavku.
Počet prvků	unsigned8	0x00 . . . 0x75 – Omezeno maximální délkou zprávy. Požadovaný počet prvků parametru typu pole.
Číslo parametru	unsigned16	0x0001 . . . 0xFFFF – 1 . . . 65535
Subindex	unsigned16	0x0000 . . . 0xFFFF – 0 . . . 65535 Index prvního požadovaného prvku parametru typu pole.
Formát	unsigned8	0x02 – integer8 0x03 – integer16 0x04 – integer32 0x05 – unsigned8 0x06 – unsigned16 0x07 – unsigned32 0x08 – floating point Jiné – viz. standard PROFIdrive v3.1 0x40 – Potvrzení úspěšného zápisu. 0x41 – Byte 0x42 – Word 0x43 – Double Word 0x44 – Error Datový typ hodnoty parametru. Na velikosti datového typu závisí velikost položky v daném datovém bloku.

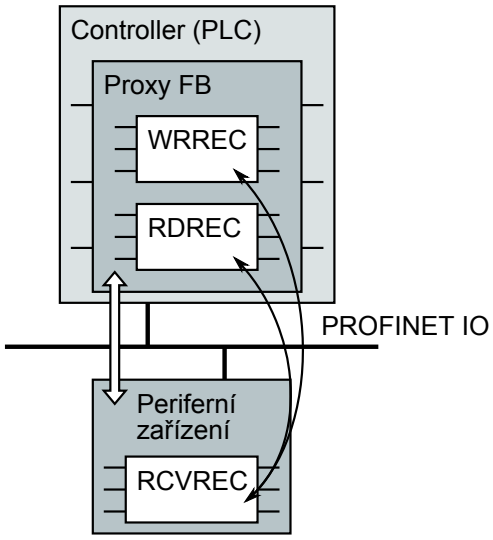
Počet hodnot	unsigned8	0x00 . . . 0xEA – Omezeno maximální délkou zprávy. Udává skutečný počet indexovaných hodnot v položce Hodnoty, příp. počet chybových hodnot.
Chybové hodnoty	unsigned16	0x0000 . . . 0x00FF V případě chyby při zápisu nebo čtení hodnoty některého z parametrů tato položka udává číslo chyby. Tabulka s významem jednotlivých chyb je uvedena v dokumentaci frekvenčního měniče [8].
Hodnoty	Dle typu	- Hodnota požadovaného parametru. V případě, že konečná délka zprávy v bytech je lichá, je na konec zprávy připojen byte 0x00, aby měla zpráva délku v jednotkách wordů.

V případě čtení parametrů je datový blok pro WRREC zkrácen o část s hodnotami parametrů. V případě úspěšného zápisu parametrů je datový blok odpovědi RDREC tvořen pouze hlavičkou (také zkrácen o část s hodnotami).

3.3 Proxy FB

Programovací model pro komunikační funkční bloky [1] definuje pojem Proxy FB jako funkční blok, který obaluje univerzální systémové komunikační bloky jako jsou např. WRREC a RDREC a vytváří tak rozhraní pro konkrétní periferní zařízení či přímo pro konkrétní aplikaci, viz obr. 3.8. V aplikačním programu tento blok reprezentuje periferní zařízení nebo jeho funkční část. Výrobci periferních zařízení mohou takové Proxy FB poskytovat v knihovnách ke svým zařízením. Jejich výhoda je zřejmá – uživatel se nemusí zabývat implementací komunikace se zařízením a studiem jeho funkčnosti, do svého programu pouze vloží požadovaný Proxy FB jako model celého zařízení či jeho vybrané funkce.

Knihovna popsaná v kap. 4.2.2 je příkladem takového souboru Proxy FB, které lze použít pro cyklickou komunikaci s frekvenčními měniči Sinamics. Knihovna obsahuje funkční bloky pro zjednodušení komunikace s použitím vybraných standardních telegramů. Na základě této knihovny, která se již běžně používá v koncových aplikacích vznikl požadavek na vytvoření podobné knihovny s Proxy FB pro acyklickou komunikaci s měniči Sinamics. Kap. 5 se zabývá způsobem implementace těchto Proxy FB.



Obr. 3.8: Proxy FB a systémové komunikační bloky

PROFIBUS and PROFINET International (PI) uvádí v [4] tzv. aplikační profily pro zařízení komunikující na průmyslové síti. Zde je definována hierarchie těchto profilů, která souvisí s úrovní standardizace periferních zařízení vzhledem k jejich interoperabilitě. Proxy FB dle tohoto standardu tvoří třetí úroveň aplikačních profilů, kterou mohou výrobci implementovat do svých produktů. Snahou výrobců je vytvářet takové Proxy FB, které jsou mimo jiné univerzální pro co nejširší spektrum výrobků a dokonce i mezi jednotlivými výrobci může vzniknout dohoda o jednotných Proxy FB pro podobná zařízení. Výhodou je potom možnost záměny takových zařízení za jiné bez nutnosti změny programu PLC. Proxy FB, které byly vytvořeny v rámci této práce, jsou univerzální pro všechny měniče řady Sinamics, neboť mají společný komunikační model, který lze k tomuto účelu použít.

4 PROGRAMOVÉ PROSTŘEDKY A KONFIGURACE

4.1 Starter

Program Starter je nástroj pro kompletní konfiguraci a monitoring pohonů Siemens. Používá se při uvádění pohonů do provozu a prvotní odzkoušení jejich funkce. Většinu parametrů pohonu lze nastavit v uživatelsky přívětivé grafické podobě. Program zvládá i takové funkce, jako je nastavení parametrů regulace pomocí automatické identifikace regulované soustavy [6].

4.1.1 Konfigurace Sinamics S110

Prvním krokem k vytvoření aplikace pro vyzkoušení a ověření komunikace měniče s PLC na síti Profinet je konfigurace měniče pomocí programu Starter. Důležité kroky konfigurace jsou zjednodušeně popsány takto:

1. Dle schématu na Obr. 1.2 propojíme počítač s řídicí jednotkou měniče. Pro funkční komunikaci s počítačem je nutné nastavit ve vlastnostech počítače pevnou adresu IP a masku podsítě tak, aby byl počítač s řídicí jednotkou ve stejné síti z hlediska komunikace protokolem TCP/IP.
   Pomocí funkce *Accessible nodes* ve Starteru vyhledáme řídicí jednotku měniče a taktéž adekvátně nastavíme IP adresu a masku podsítě. Zároveň nastavíme jméno zařízení pro síť Profinet.
2. Volbou Load CPU / drive unit to PG načteme aktuální konfiguraci měniče do PC. Můžeme předtím v online módu resetovat přístroj do továrního nastavení. Starter automaticky vyhledá konfiguraci přístroje a vloží do projektu dva hlavní objekty: řídicí jednotku a servo.
3. Dalším krokem je konfigurace samotného pohonu, nejlépe pomocí konfiguračního průvodce. Většina nastavení týkající se hardwaru je již provedena díky předchozí volbě. Co je nutné nastavit, je zejména způsob řízení – otáčková regulace, základní polohování a další. Tato volba reflektuje aplikační třídy profilu PROFIdrive, které jsou popsány v kap. 2.2. Podle zvoleného modelu řízení se nastavují další parametry. V našem případě bylo vybráno základní polohování, a tak v následujících krocích průvodce nastavíme použitou převodovku, počet tzv. LU[1] na otáčku, modulo korekci a další.

[1] Length Unit – minimální délka odměřování. Vhodné nastavit tak, aby odpovídala požadovaným fyzikálním jednotkám v dané aplikaci (např. mm, °). Měnič pracuje výhradně v jednotkách LU!

4. Nastavíme způsob komunikace pro ovládání měniče, neboli propojení vstupů a výstupů jako je žádaná hodnota, aktuální poloha a další. V našem případě vybereme komunikaci na základě zpráv PROFIdrive. Z hlediska aplikace je měnič tvořen dvěma DO, řídicí jednotkou se svými binárními vstupy a výkonovou jednotkou, která řídí pohyb motoru. Pro oba tyto objekty existují komunikační telegramy na základě standardu PROFIdrive, jejichž seznam je uveden v tab. 3.1.
5. Po nahrání konfigurace do měniče (Download) můžeme nastavit parametry regulátoru buď ručně, nebo lze využít automatického nastavení pomocí identifikace měřením. Automatické nastavení je velmi užitečná funkce, servo pomocí náhodného signálu proměří parametry soustavy svázané s motorem a navrhne optimální regulaci. Toto automatické nastavení funguje dobře, chceme-li nastavit regulaci ručně, lze ho použít alespoň k prvotnímu odhadu parametrů. Na závěr lze nastavit limitní hodnoty rychlosti, zrychlení, polohy a další velké množství parametrů, jimiž měnič disponuje.
6. Posledním volitelným krokem může být vyzkoušení nastaveného systému pomocí funkce *Commissioning*. Touto funkcí lze přímo zadávat žádané hodnoty a vyzkoušet, zda se motor pohybuje požadovaným způsobem.

4.2 TIA Portal

Totally Integrated Automation Portal (TIA Portal) je nové vývojové prostředí společností Siemens pro programování kompletních automatizačních systémů. Svým názvem dobře vystihuje hlavní myšlenku tohoto nového prostředí, to jest integrace všech dílčích součástí projektování automatizačních systémů do jednoho softwarového nástroje s jednotným ovládáním. V současné době (verze 11) je dokončena integrace systému pro vývoj aplikací z řídící techniky a decentralizovaných periferií (současná Step 7 V11), dále pro vývoj vizualizací v blízkosti stroje (ovládací panely) a systémů SCADA (WinCC V11). Připravuje se integrace systému pro práci s pohony Sinamics (Sinamics StartDrive), který ve výsledku nahradí výše uvedený software Starter [6].

Prostředí TIA Portal je rozděleno do dvou základních zobrazení [10]:

- *Portal View* – Zobrazení ve formě úkolů odpovídajících dílčím součástem projektu, které jsou seřazeny postupně podle běžného způsobu vytváření projektu. Je to úkolově orientované zobrazení.
- *Project View* – Zobrazuje celý projekt a umožňuje tak přístup ke všem dílčím součástem projektu najednou. Je to objektově orientované zobrazení.

4.2.1 Konfigurace Simatic S7-1200

Konfiguraci projektu v prostředí TIA Portal lze taktéž zjednodušeně popsat v několika krocích:

1. Vytvoříme nový projekt a přidáme požadované PLC. U PLC je nutno nastavit zejména IP adresu sítě včetně masky podsítě a jméno zařízení v síti Profinet.
2. Z katalogu přidáme požadovaný měnič Sinamics S110 a vložíme ho do projektu v Network View. Pokud nemáme požadovaný měnič v katalogu, stáhneme si příslušný GSDML (General Station Description Markup Language) soubor z webu Siemens a nainstalujeme jej. Propojíme měnič s PLC a ve vlastnostech zařízení taktéž nastavíme IP adresu a jméno v síti Profinet (tentokrát se musí shodovat s údaji nastavenými při konfiguraci měniče programem Starter).
3. V konfiguraci měniče vložíme z katalogu objekty řídicí jednotky a serva a k nim zvolené komunikační telegramy (musí samozřejmě odpovídat těm, které jsme zvolili při konfiguraci měniče). U telegramů vybereme oblast vstupních a výstupních operandů PLC pro mapování dat telegramu. Ve vlastnostech měniče můžeme nastavit periodu výměny dat po síti Profinet.
4. V programu poté přistupujeme k parametrům měniče přímo pomocí vstupních a výstupních operandů ve zvolené oblasti systémové paměti. Přiřazení operandů k parametrům měniče je dáno zvoleným telegramem. Můžeme také použít funkčních bloků z níže popsané knihovny pro cyklickou komunikaci.

4.2.2 Knihovna funkcí pro cyklickou komunikaci

Pro ovládání měničů řady Sinamics byla v brněnské pobočce Siemens vytvořena knihovna *S7-1200-Sinamics-Lib-v7a* pro prostředí TIA Portal, která obsahuje funkční bloky na odesílaní a příjímání procesních dat pomocí některých telegramů, tedy pro cyklickou komunikaci. Pro měniče Sinamics jsou implementovány funkční bloky pro telegramy 1, 2, 9, 111, 352, 390 a 394. Tyto bloky zjednodušují řízení měniče v základních aplikacích. Na rozdíl od přímého přístupu k mapovaným operandům není nutné znát přesně strukturu telegramu, měnič lze ovládat pouze pomocí vstupů a výstupů daného bloku, jejichž význam je srozumitelně popsán.

4.3 Programovací jazyk SCL

K programování PLC v prostředí TIA Portal STEP 7 je možné použít různé programovací jazyky. PLC řady S7-1200 lze programovat v jazyce LAD (Ladder Logic), FBD (Function Block Diagram) a SCL (Structured Control Language). SCL je z těchto jazyků nejpokročilejším programovacím jazykem, a proto byl také zvolen

pro programování všech bloků v této práci. Na rozdíl od grafických programovacích jazyků LAD a FBD je to jazyk textový, založený na jazyce PASCAL. Dle standardu IEC 61131-3, který definuje syntaxi a strukturu programovacích jazyků pro PLC, spadá do kategorie programovacího jazyka Structured Text (ST) a je tedy tedy plně standardizován [4] [13].

Výhodou jazyka SCL je zejména podpora typu instrukcí známých z vyšších programovacích jazyků, jako jsou např. konstrukce `IF..THEN..ELSE`, `FOR` cykly, `WHILE` cykly a další. Samozřejmě podporuje i blokový koncept STEP 7 a lze jej tak libovolně kombinovat s bloky naprogramovanými v ostatních podporovaných jazycích, jinak řečeno, uvnitř bloku psaném v SCL lze volat bloky psané v ostatních jazycích a naopak [4].

Nevýhodou programování PLC S7-1200 je absence tzv. *debuggeru*, který umožňuje zastavení běhu programu v požadovaném místě pomocí *breakpointů*. Pokračování programu lze potom ručně krokovat po jednotlivých řádcích a detailně tak sledovat tak průběh programu s hodnotami veškerých proměnných. To je obrovská výhoda u rozsáhlejších aplikací, neboť lze snadno odhalit logické chyby v kódu. Naproti tomu bez debuggeru je hledání chyb složité a hlavně velmi zdlouhavé. Je nutné vytvořit řadu pomocných proměnných a kódu navíc, který nám umožní sledovat chod programu a odhalit tak chyby.

5 IMPLEMENTACE

5.1 Proxy FB pro acyklický zápis a čtení parametrů

Výsledkem implementace acyklického čtení a zápisu z a do měničů Sinamics jsou dva Proxy FB, `Read_Params` pro čtení parametrů z měniče a `Write_Params` pro zápis parametrů do měniče. Jádro implementace obou funkčních bloků je poměrně podobné, liší se zejména vstupním a výstupním rozhraním.

5.1.1 Implementace pomocí stavového diagramu

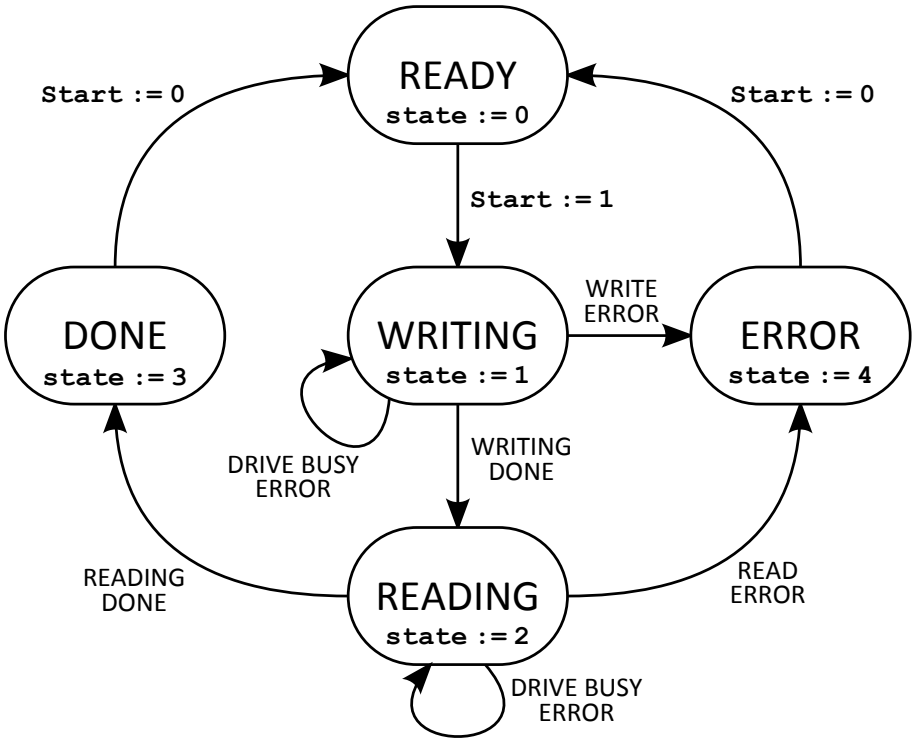
Acyklické čtení a zápis parametrů jsou z hlediska volání systémových funkčních bloků WRREC a RDREC téměř shodné úlohy, kde nejprve voláme blok WRREC pro přenos dat z PLC do měniče a po jeho úspěšném dokončení voláme blok RDREC pro čtení odpovědi z měniče. Oba funkční bloky jsou vzhledem k cyklickému průběhu vykonávání programu PLC asynchronní, tzn. že jejich volání trvá několik programových cyklů.

V případě běžného synchronního volání by byla úloha velice jednoduchá – program by se skládal pouze z volání jednoho a následně druhého SFB. Pro asynchronní volání je ovšem nutné vytvořit algoritmus, který sleduje výstupní hodnoty těchto funkčních bloků a na jejich základě rozhoduje o dalším postupu. Na základě toho jsem se rozhodl vytvořit schéma v podobě stavového automatu, které je znázorněno na obr. 5.1. Význam jednotlivých stavů ve schématu je uveden v tab. 5.1.

Tab. 5.1: Význam stavů FB pro acyklickou komunikaci.

Stav		Popis
0	READY	FB čeká na spuštění signálem Start.
1	WRITING	Probíhá zápis SFB WRREC.
2	READING	Probíhá čtení SFB RDREC.
3	DONE	Všechny operace byly úspěšně dokončeny.
4	ERROR	Došlo k chybě v během zpracování požadavku.

Každý z uvedených stavů obsahuje část kódu implementující přechod do jiných stavů. Možné přechody jsou jasně dány schématem 5.1. Přechod do následujících stavů je vždy dán kombinací vstupu bloku a výstupu SFB WRREC a RDREC, které jsou volány na konci tohoto FB. Blok je naprogramován tak, že reaguje vždy



Obr. 5.1: Schéma stavového automatu FB pro acyklickou komunikaci.

na kladnou hranu vstupního signálu `Start`. Následující kapitola popisuje celkové řešení Proxy FB pro acyklickou komunikaci.

Stavový automat, který je uveden na obr. 5.1, je implementován pomocí konstrukce `CASE..OF`. Následující odstavec uvádí pouze kostru zdrojového kódu této implementace.

```
CASE #state OF
  0: //stav READY (zacatek)
    IF #Start THEN
      #state := 1;
      #wr.start := true;
    END_IF;
... (*Inicializace vstupních a výstupních proměnných*)
  1: //Stav WRITE
    #wr.start := false;
    IF NOT #wr.busy THEN
      IF #wr.error THEN
        #stat := #wr.status;
        CASE #stat OF
         //Chyby pri spatne zadanem HW_ID
          16#C080_9000, 16#C080_B600, 16#C080_B200:
            #state := 4; //+popis chyby
          //Chyba udava ze menic neni pripraven -> opakovani pokusu
          16#DF80_B500:
            #wr.repeat_count := #wr.repeat_count + 1;
            IF #wr.repeat_count < 100 THEN
              #wr.start := true;
              #state := 1;
```

```
            ELSE
              #state := 4; //+popis chyby
            END_IF;
          ELSE
            #state := 4; //+popis chyby
        END_CASE;
      ELSIF #wr.done THEN //Zapis hotov -> cteni odpovedi
        #state := 2;
        #rd.start := true;
      ELSE
        #state := 4; //+popis chyby
      END_IF;
    END_IF;
  2: //read state
    #rd.start := false;
    IF NOT #rd.busy THEN
      IF #rd.valid THEN
...(***DEKOMPOZICE BLOKU PRECTENYCH DAT***)
        //Kontrola response ID zda odpoved obsahuje chyby
        IF #rd.data.header.response_id = 16#81 THEN
          #state := 4; //+popis chyby
        ELSIF #rd.data.header.response_id = 16#01 THEN
          #state := 3;
        ELSE
          #Error_Description := 'Neplatne Response ID.';
          #state := 4; //+popis chyby
        END_IF;
      ELSIF #rd.error THEN
        #stat := #rd.status;
        CASE #stat OF
          //Chyby pri spatne zadanem HW_ID
          16#C080_9000, 16#C080_B600, 16#C080_B200:
            #state := 4; //+popis chyby
          //Chyba udava ze menic neni pripraven -> opakovani pokusu
          16#DE80_B500:
            #rd.repeat_count := #rd.repeat_count + 1;
            IF #rd.repeat_count < 100 THEN
              #rd.start := true;
              #state := 2;
            ELSE
              #state := 4; //+popis chyby
            END_IF;
          ELSE
            #state := 4; //+popis chyby
        END_CASE;
      ELSE
        #state := 4; //+popis chyby
      END_IF;
    END_IF;
  3://Stav DONE
    #DONE := true;
    #BUSY := false;
    IF NOT #Start THEN
      #state := 0;
    END_IF;
  4: //Stav ERROR
    #ERROR := true;
    #BUSY := false;
```

```
    IF NOT #Start THEN
      #state := 0;
    END_IF;
  ELSE
    #ERROR := true;
    #BUSY := false;
    IF NOT #Start THEN
      #state := 0;
    END_IF;
END_CASE;
```

Pomocí statické proměnné `state` se na začátku rozhodne, která část kódu se bude vykonávat. V každé části jsou postupně testovány relevantní proměnné a na základě toho je do proměnné `state` přiřazen následující stav. V dalším cyklu PLC je tedy vykonán kód tohoto stavu.

Stav READY

Pokud `Start = true`, nastaví se nejprve `wr.start := true`, což je proměnná spouštějící blok WRREC a následující stav se nastaví na 1 (stav WRITE). Dále se provede kompletní inicializace, která ve výtahu z kódu není uvedena a která obsahuje:

- Vynulování všech výstupních proměnných.
- Nastavení výstupu BUSY.
- Přepis hodnot vstupů do pole pro pozdější možnost indexace.
- Inkrementování referenčního čísla požadavku.
- Zjištění počtu parametrů – spočívá v nalezení prvního parametru ze vstupů, u kterého je číslo parametru 0.
- Pro případ zápisu (funkce `Write_Params`) algoritmus pro generování hodnotové části struktury s daty pro SFB WRREC (viz obr. 3.6).
- Výpočet délky struktury s daty pro blok WRREC v bytech.

Délka struktury s daty pro blok WRREC závisí na počtu parametrů, které jsou přenášeny a také na tom, zda je požadavek na čtení nebo na zápis. V případě požadavku na čtení se skládá pouze s části s hlavičkou (4 B) a s části s adresami (6 B na parametr). Celková délka je tedy $4+6n$ B, kde n je počet čtených parametrů. V případě požadavku na zápis přibývá ještě část s hodnotami, jejíž délka je závislá jak na počtu parametrů, tak na datovém typu jednotlivých parametrů. Celková délka se tedy určí až po vygenerování hodnotové části dat v rámci jeho algoritmu, který je popsán v kap. 5.1.2.

Stav WRITE

Zápis je spuštěn už ve stavu READY při náběžné hraně vstupu `Start`. Hned v dalším cyklu je aktivován nový stav WRITE, `wr.start` je resetováno a je tedy v log. 1

jenom během jednoho cyklu CPU. Dle časového diagramu na obr. 3.4 je toto validní spuštění asynchronního SFB WRREC.

Dále jsou v tomto stavu testovány výstupy SFB WRREC, proměnné `wr.busy`, `wr.done` a `wr.error` a v případě chyby bloku je testován status kód chyby `wr.status`. Jeho význam lze nalézt v manuálu PLC [10]. Jestliže proměnná `wr.status` udává chybu 0xDF80B500 – měnič není připraven, je pokus čtení opakován nastavením `wr.start := true`. Maximální počet pokusů opakování je omezen pomocí proměnné `wr.repeat_count`. V případě jiné chyby je její status kód a možný popis její příčiny předán na výstup bloku a následující stav je ERROR. V případě úspěšného zápisu (`wr.done = true`) se nastaví proměnná pro spuštění SFB čtení RDREC (`rd.start := true`) a následující stav je READ.

Stav READY

V tomto stavu probíhá vše podobně jako ve stavu WRITE, navíc je ale při úspěšném přečtení blokem RDREC provedena dekompozice čtené zprávy. Zde je velmi důležité si uvědomit, že úspěšné provedení SFB WRREC a RDREC ještě nemusí znamenat úspěšně provedený zápis nebo čtení požadovaných parametrů. Tyto funkční bloky pouze zprostředkovávají posílání a čtení zpráv v acyklickém pásmu Profinetu na zařízení dané vstupem `HW_ID`. To, že výstupy bloku udávají úspěšné čtení nebo zápis pouze znamená, že zpráva byla přijata daným periferním zařízením nebo z něj byla přečtena. Vyskytne-li se chyba ve formátu této zprávy, měnič tuto chybu indikuje v těle zprávy s odpovědí. I v případě zápisu parametrů je tedy nutné ji přečíst a přesvědčit se, že všechny parametry byly úspěšně zapsány a v opačném případě analyzovat chyby při zápisu všech parametrů zvlášť. Algoritmus pro dekompozici odpovědi frekvenčního měniče je popsán v kap. 5.1.2.

Stavy DONE a ERROR

Tyto stavy slouží k indikaci úspěšného či chybového provedení celého funkčního bloku. Oba stavy by bylo teoreticky možné vypustit. Nastavení výstupů ERROR, DONE a BUSY by mohlo být provedeno již ve stavech WRITE a READ během zjištění chyby nebo úspěšného provedení a mohl by tak následovat pouze stav READY. Použití těchto stavů ale zjednodušuje kód v stavech READ a WRITE, kde se tak nemusí mnohokrát opakovat nastavování výstupů. Navíc jsou tyto stavy použité tak, že zaručují funkčnost spouštění bloku na náběžnou hranu vstupu `Start`. Přechod do stavu READY se totiž provede až po vynulování tohoto vstupu. Výhodou je také o něco lepší čitelnost kódu.

5.1.2 Kompozice datových struktur pro SFB WRREC a RDREC

Datový typ VARIANT je ukazatel, který může ukazovat na data s různými datovými typy. Na rozdíl od ukazatelů typu POINTER a ANY nezabírá žádné místo v paměti. Vstup typu VARIANT lze použít dvěma následujícími způsoby:

- Absolutní adresování – ukazatel na souvislý blok dat uvedeme např. ve tvaru `P#DB10.DBX10.0 INT 12`, kde se přímo odkazujeme na posloupnost 12-ti integerů v datovém bloku DB10 počínající 10. bytem bloku.
- Symbolické adresování – lze dosadit Tag jakéhokoliv typu včetně polí a struktur.

Aby byly Proxy FB použitelné v jakémkoliv uživatelském programu, nelze používat absolutní adresování. Absolutní adresy v programu budou vždy záviset na jeho tvůrci, jak je nadefinuje. Z toho vyplývá, že musíme použít symbolické adresování.

V dokumentaci [10], [13] jsou uvedeny pouze velmi stručné informace k použití symbolického adresování vstupu typu VARIANT. Při absolutním adresování je vždy předána počáteční adresa a délka dat v Bytech. Experimentováním bylo zjištěno, že délka dat u symbolického adresování se odvodí z typu použité proměnné. Dosadíme-li Tag ukazující na jednoduchou proměnou jako Int, Byte, Real apod., odpovídá délka pouze jedné proměnné. To je v našem případě nezajímavé, potřebujeme se symbolicky odkázat na určitou definovanou posloupnost dat obsahující data strukturovaná do požadavku zápisu nebo čtení dle obr. 3.6 a 3.7. Toho lze dosáhnout dvěma způsoby. Dosadíme-li za vstup typu VARIANT pole hodnot (typ Array), je jeho délka odvozená z délky tohoto pole. Dosadíme-li strukturu (typ Struct), je délka odvozená z celkové délky struktury.

Ideálním řešením se jeví vytvoření struktury podle požadovaných parametrů, která je z hlediska jednotlivých položek velmi přehledná. To ale vzhledem k variabilitě typů parametrů a jejich různé délce nelze. Struktura musí být pevně definována na začátku bloku v definici proměnných a to se všemi proměnnými, které obsahuje. Některé části dat pro zápis a čtení přesto mají statickou strukturu a délku, např. hlavička. Pro část s hodnotami parametrů, která se mění, je ale jediným možným řešením dynamicky vytvářet pole hodnot, které naplníme postupně pomocí níže uvedeného algoritmu.

Za vstup RECORD SFB WRREC či RDREC potom dosadíme proměnou typu Struct, která se skládá z vnitřní struktury s hlavičkou a s polem Wordů s adresami a hodnotami parametrů. Konečné řešení tak v podstatě kombinuje obojí, předaná hodnota je typu Struct, ovšem část jejích dat je uložena v poli Wordů.

Struktura zapisovaných dat

Algoritmus pro kompozici datové struktury pro zápis je tvořen dvěma cykly `FOR`. První slouží k vytvoření části s adresami parametrů, druhý k části s hodnotami. K indexování používáme dvě lokální proměnné: `i`, pro indexování pomocného struktury s parametry vytvořené během inicializace, která je inkrementována v rámci `FOR` cyklu a `j`, která je inkrementována v kódu uvnitř cyklu v závislosti na velikosti přiřazených dat. Druhý cyklus obsahuje instrukci `CASE..OF`, která na základě typu parametru přetypuje vstupní proměnnou a zapíše její hodnotu do daného počtu bytů na správnou pozici v rámci pole.

Pro typ parametru unsigned8 a integer8 je dodrženo zarovnání struktury dat na jednotky wordů, tzn. že v prvním bytu slova je hodnota a zapisujeme-li lichý počet indexovaných hodnot, je konec doplněn nulovým bytem.

```
//vytvoreni casti adres parametru
#j := 0;
FOR #i := 0 TO #n - 1 DO
  #wValue.B1 := #wr.params[#i].attribute;
  #wValue.B0 := #wr.params[#i].elements;
  #wr.data.values[#j] := #wValue;
  #j := #j + 1;
  #wr.data.values[#j] := INT_TO_WORD(#wr.params[#i].number);
  #j := #j + 1;
  #wr.data.values[#j] := INT_TO_WORD(#wr.params[#i].start_index);
  #j := #j + 1;
END_FOR;
//vytvoreni casti hodnot parametru
FOR #i := 0 TO #n - 1 DO
  #wValue.B1 := #wr.params[#i].type;
  #wValue.B0 := #wr.params[#i].elements;
  #wr.data.values[#j] := #wValue;
  #j := #j + 1;
  #type := #wr.params[#i].type;
  CASE #type OF
    2, 5: //integer8, unsigned8
      #wValue := 0;
      #wValue.B1 := DINT_TO_BYTE(#wr.params[#i].value);
      #wr.data.values[#j] := #wValue;
      #j := #j + 1;
    3, 6: //integer16, unsigned16
      #wr.data.values[#j] := DINT_TO_WORD(#wr.params[#i].value);
      #j := #j + 1;
    4, 7: //integer32, unisgned32
      #dwValue := DINT_TO_DWORD(#wr.params[#i].value);
      #wr.data.values[#j] := #dwValue.W1;
      #wr.data.values[#j + 1] := #dwValue.W0;
      #j := #j + 2;
    8: //real
      #dwValue := REAL_TO_DWORD(#wr.params[#i].value_real);
      #wr.data.values[#j] := #dwValue.W1;
      #wr.data.values[#j+1] := #dwValue.W0;
      #j := #j + 2;
    ELSE
```

```
      #state := 4;
      #wr.start := false;
  END_CASE;
END_FOR;
//delka zapisu v~bytech
#wr.len := SINT_TO_UINT(4 + 2*#j);
```

Dekompozice návratových dat

Algoritmus pro dekompozici je tvořen jedním cyklem `FOR` s instrukcí `CASE..OF`. Indexujeme opět dvěma proměnnými, `j` slouží k indexaci výstupního pole s hodnotami parametrů a `i` slouží indexaci pole vráceného blokem RDREC a je opět inkrementována v kódu uvnitř cyklu v závislosti na právě přečteném datovém typu vráceného parametru. Hodnoty parametrů jsou přiřazeny do dvou dočasných polí typu DInt a Real, podle přečteného typu parametru. V případě čtení parametru typu unsigned32 je provedena kontrola přetečení.

V případě, že vrácená data obsahují chybové hodnoty, jejich číselná reprezentace je předána na chybový výstup daného parametru `Par[0-9]_Error` a popis chyby je předán na výstup `Error_Description`.

```
#dwValue := 0;
#i := 0;
FOR #j := 0 TO BYTE_TO_SINT(#rd.data.header.number_of_parameters - 1) DO
  #wValue := #rd.data.values[#i];
  #type := BYTE_TO_USINT(#wValue.B1);
  #i := #i + 1;
  CASE #type OF
    2: //integer8
      #wValue := #rd.data.values[#i];
      #Out[#j] := SINT_TO_DINT(BYTE_TO_SINT(#wValue.B1));
      #Out_Real[#j] := DINT_TO_REAL(#Out[#j]);
      #i := #i + 1;
    3: //integer16
      #wValue := #rd.data.values[#i];
      #Out[#j] := INT_TO_DINT(WORD_TO_INT(#wValue));
      #Out_Real[#j] := DINT_TO_REAL(#Out[#j]);
      #i := #i + 1;
    4: //integer32
      #dwValue.W1 := #rd.data.values[#i];
      #dwValue.W0 := #rd.data.values[#i+1];
      #Out[#j] := DWORD_TO_DINT(#dwValue);
      #Out_Real[#j] := DINT_TO_REAL(#Out[#j]);
      #i := #i + 2;
    5: //unsigned8
      #wValue := #rd.data.values[#i];
      #Out[#j] := USINT_TO_DINT(BYTE_TO_USINT(#wValue.B1));
      #Out_Real[#j] := DINT_TO_REAL(#Out[#j]);
      #i := #i + 1;
    6: //unsigned16
      #wValue := #rd.data.values[#i];
      #Out[#j] := UINT_TO_DINT(WORD_TO_UINT(#wValue));
      #Out_Real[#j] := DINT_TO_REAL(#Out[#j]);
```

```
      #i := #i + 1;
    7: //unsigned32    #dwValue.W1 := #rd.data.values[#i];
      #dwValue.W0 := #rd.data.values[#i+1];¨
      //Kontrola preteceni
      IF #dwValue  <= 16#7FFF_FFFF THEN
        #Out[#j] := UDINT_TO_DINT(DWORD_TO_UDINT(#dwValue));
      ELSE
        #Out[#j] := 16#7FFF_FFFF;
        #state := 4;
      END_IF;
      #Out_Real[#j] := DINT_TO_REAL(#Out[#j]);
      #i := #i + 2;
    8: //real
      #dwValue.W1 := #rd.data.values[#i];
      #dwValue.W0 := #rd.data.values[#i+1];
      #Out_Real[#j] := DWORD_TO_REAL(#dwValue);
      #Out[#j] := ROUND(#Out_Real[#j]);
      #i := #i + 2;
    68: //error
      #Out[#j] := 0;
      #Out_Real[#j] := 0.0;
      #wValue := #rd.data.values[#i-1];
      #k :=#wValue.B0;
      #wValue := #rd.data.values[#i];
      #hta_error := HTA(IN:=#wValue, N:=2, OUT=>#error_code);
      #Error_Description := CONCAT(IN1 := #Error_Description, ...
      #i := #i + USINT_TO_SINT(#k);
    ELSE
      #Out[#j] := 0;
      #Out_Real[#j] := 0.0;
      #Error_Description := CONCAT(IN1 := #Error_Description, ...
      #state := 4;
      #i := #i + 1;
  END_CASE;
END_FOR;
```

5.1.3 Zhodnocení způsobu řešení

Zvolené řešení Proxy FB je pouze jedním z možných a má svoje výhody i nevýhody. Během implementace bloků byly zamýšleny různé varianty řešení, některé z nich jsou naznačeny v následujících podkapitolách.

Rozklad na sub-funkce

Řadu částí zdrojového kódu by bylo běžně lepší strukturovat do sub-funkcí, které by mohly být volány z hlavního FB, a v některých případech by mohly být společné pro oba bloky čtení i zápisu. Rozpad zdrojového kódu do více menších funkcí by zcela určitě pomohl čitelnosti kódu a je zajisté jedním ze základních pravidel správného programování.

Prostředí STEP7 V11 bohužel není dostatečně uzpůsobené k vytváření složitějších knihoven funkcí a nedovoluje deklarovat sub-funkce či funkční bloky v rámci

jiných funkcí či funkčních bloků. Chceme-li tedy v našem případě vytvořit knihovnu s funkčním blokem, který půjde jednoduše a nezávisle vložit z knihovny do jakéhokoliv uživatelského programu, nelze v bloku volat žádné další uživatelské funkce či FB. Pokud bychom v rámci bloku volali jiné uživatelské funkce či FB, při jeho vložení z knihovny do jiného programu by při kompilaci vznikaly chyby z důvodu neexistujících volaných funkcí, které by uživatel musel všechny zvlášť vkládat do svého programu. Z tohoto důvodu jsou nakonec Proxy FB na úkor přehlednosti naprogramovány v celku bez použití sub-funkcí.

Jeden blok pro zápis i čtení

Oba funkční bloky zápisu i čtení by bylo možné sjednotit do jednoho bloku. Parametry pro zápis by mohly být oddělené od parametrů pro čtení, případně by mohlo být u každého parametru uveden další binární přepínač zápis/čtení, který by udával požadovanou operaci daného parametru.

Rozhraní bloků je už i tak jak jsou nyní navrženy poměrně zdlouhavé a toto řešení by seznam vstupních a výstupních parametrů ještě více prodloužilo. Problém v použití dvou oddělených bloků může nastat při současném spuštění obou bloků najednou použijeme-li společnou proměnnou pro vstup `Start`. Zápis a čtení není volat souběžně, pokud tak učiníme, druhý volaný blok skončí chybou při čtení z měniče, která je indikována na výstupu bloku.

Sjednocením obou bloků do jednoho by bylo teoreticky možné tento problém eliminovat, ale pouze za cenu zmenšení přehlednosti, která už i tak není nejlepší. Navíc pokud by uživatel vytvořil více instancí takových funkčních bloků a volal by je současně, vznikl by stejná chyba, kterou v takovém případě nelze nijak eliminovat.

Vstup typu `Struct` nebo `Array`

Oba Proxy FB mají poměrně dlouhé vstupně/výstupní rozhraní, které by bylo možné zkrátit použitím vstupů typu pole (`Array`) nebo struktury (`Struct`). Použití takových funkčních bloků není ale zdaleka tak přímočaré jako definování každého vstupu zvlášť.

Použitím rozvětvené struktury by bylo možné redukovat počet vstupů na jeden vstup pro každý parametr, použitím pole jako vstupu bychom mohli odstranit opakování všech proměnných jako je číslo, index, typ a hodnota parametru. Chceme-li ale volat blok se vstupními proměnnými typu `Struct`, musíme nejprve zjistit, jak je daná struktura nadefinována, poté deklarovat pomocné proměnné stejné struktury a ty potom dosadit na vstup.

Použitím polí na vstupech bloku bychom mohli odstranit opakování všech proměnných jako je číslo, index, typ a hodnota parametru. Opět by bylo nutné nejprve deklarovat pomocná pole, naplnit je a poté je dosadit na vstupy bloku.

Tento přístup je vzhledem k jednoduchosti použití bloků nevýhodný, a proto bylo také od použití struktury nebo pole na místě vstupního rozhraní Proxy FB upuštěno.

5.1.4 Vstupně/výstupní rozhraní

Vzhledem k tomu, že parametry měniče mají různé datové typy, je nutné nějakým způsobem zobecnit typ vstupu či výstupu reprezentující hodnotu parametru.

Prvotní návrh spočíval v použití vstupů a výstupů typu VARIANT, což by umožnilo dosadit proměnou jakéhokoliv typu. V dokumentaci k PLC [10] je uvedeno, že proměnná tohoto typu nezabírá žádné místo v paměti. Není zde uveden žádný možný způsob, jak přistoupit k datům, na které taková proměnná ukazuje. Přestože lze u rozhraní vstupů a výstupu FB deklarovat Tag tohoto typu, nejsou zde uvedeny žádné informace o takovém použití a jak s takovou vstupní proměnnou pracovat ve vlastním FB.

Experimentováním se vstupní proměnnou typu VARIANT jsem nenalezl způsob, jak přistoupit k původní proměnné nebo oblasti dat, na které ukazuje. Na rozdíl od ukazatele typu ANY pro PLC S7-300/400, jehož je obdobou, jej nelze rozložit na strukturu obsahující informace o adrese uložených dat např. pomocí „AT“ konstrukce, kterou normálně používáme pro deklaraci dvou a více tagů ukazujících na stejná data. Nelze jej ani dosadit za vstup žádné z operací typu „MOVE“ pro přesun bloku dat. Kompilace takového kódu skončí chybou, přestože některé z těchto základních instrukcí mají vstup typu VARIANT. Jako vstup uživatelského bloku je tak téměř bezvýznamný. Jediný možný způsob použití je předat jej jako parametr dál do systémových funkčních bloků a jiných operací, které tento typ používají. Zdá se tedy, že typ VARIANT byl vytvořen pouze z důvodu potřeby některých SFB, které díky jejich podstatě vyžadují vstupní či výstupní proměnnou libovolného typu resp. oblast dat nezávisle na typu.

Protože typ VARIANT pro vstupní hodnoty nelze použít, bylo jako řešení problému variability datového typu parametrů měniče zvoleno řešení se dvěma vstupy pro jeden parametr. První slouží jako vstup pro všechny použité celočíselné datové typy (integer8, integer16, integer32, unsigned8, unsigned16 a unsigned32) a je typu DInt (integer32). Tomuto vstupu lze přiřadit jakoukoliv proměnnou jiných celočíselného typu a konverze je provedena implicitně, vyjma proměnné typu UDInt, kde hrozí přetečení a konverzi je nutné uvést explicitně. Parametr měniče s hodnotou větší než $2^{31} - 1$ je ovšem velmi nepravděpodobný a případ přetečení lze snadno

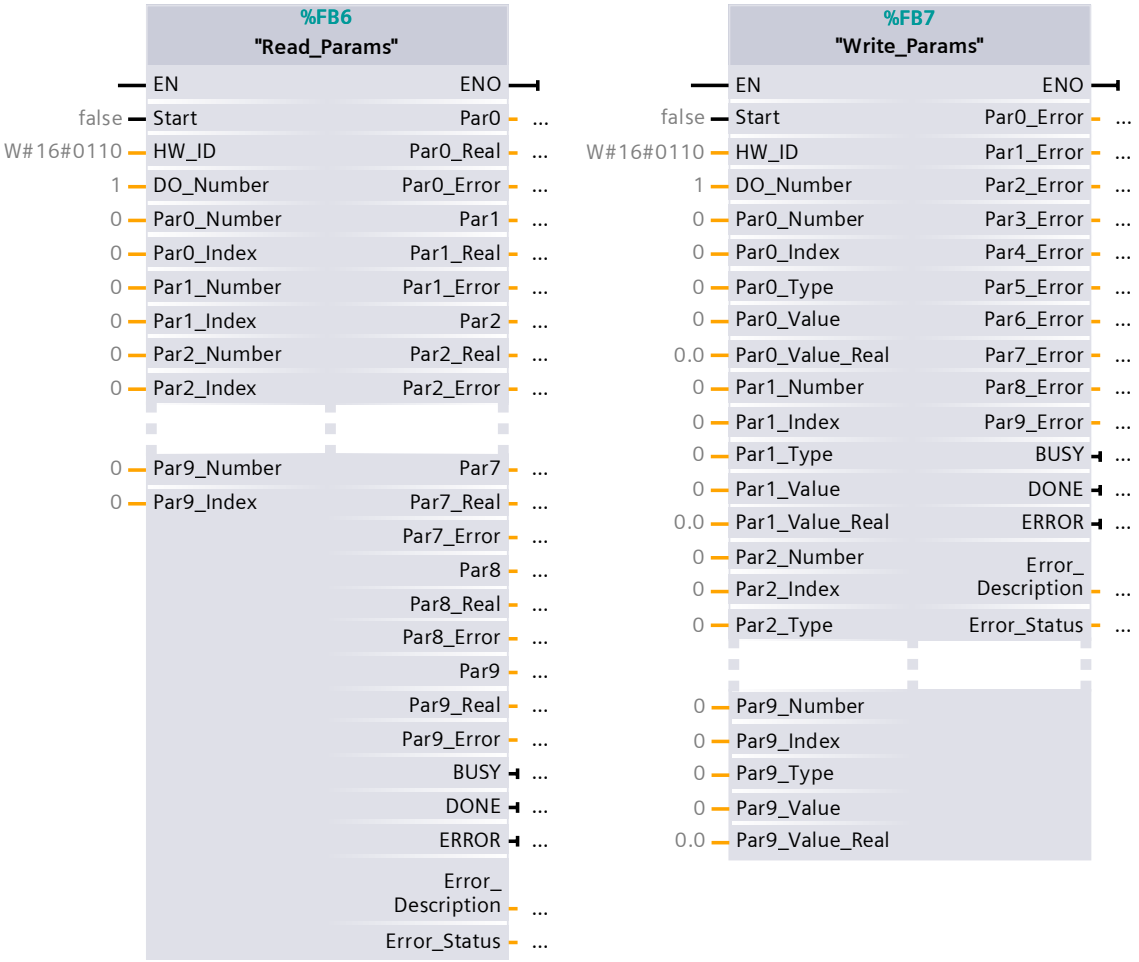
ošetřit podmínkou. Druhý slouží jako vstup pro parametry s reálnými čísly a je tedy typu Real.

V případě zápisu musí uživatel bloku znát datový typ parametru, který zapisuje. Ten lze snadno zjistit v dokumentaci měniče. Na základě uvedeného datového typu vstupem `Par[0-9]_Type` se předá k zapsání hodnota buď celočíselného nebo reálného vstupu. U čtení parametrů není nutné typ parametru znát, Proxy FB z odpovědi měniče automaticky rozezná typ parametru a jeho hodnotu převede na oba výstupy bloku, reálný i celočíselný s tím, že celočíselný výstup je zaokrouhlen.

Nejpodstatnějším rozdílem mezi Proxy FB pro zápis a pro čtení je různé vstupní a výstupní rozhraní bloků. Následující část popisuje detailně vstupně/výstupní rozhraní bloků `Read_Params` a `Write_Params`.

Rozhraní bloku `Read_Params`

Na obr. 5.2 je znázorněn blok `Read_Params` volaný z funkce programované v LAD. Význam jednotlivých vstupů a výstupů bloku je vždy uveden v jejich komentáři,



Obr. 5.2: Vstupně/výstupní rozhraní bloku `Read_Params` a `Write_Params`

takže po najetí na ně ukazatelem myši se zobrazí popis, který je drobnou nápovědou jak je použít. V tab. 5.2 je potom uveden jejich detailnější popis.

Rozhraní bloku `Write_Params`

Obr. 5.2 opět ukazuje blok `Write_Params` volaný z funkce programované v LAD. Tab. 5.2 detailně popisuje význam jednotlivých vstupů a výstupů. Důležitá je správná definice vstupu `Par[0-9]_Type`, která definuje typ příslušného parametru a bez které by nebylo možné správně vytvořit strukturu dat pro zápis. Legenda typů je účelně zachována ve stejném formátu jako u definice typu v datové struktuře pro zápis, viz tab. 3.3. Rozdíl je jen v datovém typu samotného vstupu, který je USInt, což vede uživatele na používání běžné desítkové soustavu namísto šestnáctkového zápisu ve tvaru `16#XY`, který je na první pohled o něco méně přehledný a pro laika méně srozumitelný.

Tab. 5.2: V/V rozhraní Proxy FB `Read_Params` a `Write_Params`

Název	Popis
Vstupy	
`Start`	Zápis parametrů je spuštěn s náběžnou hranou.
`HW_ID`	Identifikátor rozhraní Profinet pro zápis. V „Properties“ frekvenčního měniče v TIA Portálu je to jméno „rozhraní PROFINET“ (PROFINET interface [X1]) – výchozí hodnota, která je dána GSDML souborem měniče, je u většiny měničů „PN-IO“ a ta je také přednastavena jako výchozí hodnota tohoto vstupu. U některých měničů se ale liší a je vždy potřeba zkontrolovat a správně nastavit!
`DO_Number`	Číslo objektu pohonu (DO), jehož parametry čteme. Seznam DO měniče lze najít např. na záložce „Communication“ v konfiguraci frekvenčního měniče v programu Starter.
`Par[0-9]_Number`	Čísla jednotlivých parametrů, které chceme číst/zapisovat.
`Par[0-9]_Index`	Index v rámci daného parametru, který chceme číst/zapisovat.

`Par[0-9]_Type`	Datový typ parametru. Pouze u čtení `Write_Params`. Legenda: 2 . . . integer8 3 . . . integer16 4 . . . integer32 5 . . . unsigned8 6 . . . unsigned16 7 . . . unsigned32 8 . . . floating point (Real)
`Par[0-9]_Value`	Hodnota zapisovaného parametru, pokud je celočíselného typu. Pouze u čtení `Write_Params`.
`Par[0-9]_Value_Real`	Hodnota zapisovaného parametru, pokud je typu Real (floating point). Pouze u čtení `Write_Params`.
Výstupy	
`Par[0-9]`	Hodnota přečteného parametru, pokud je celočíselného typu. (Jinak je hodnota zaokrouhlena)
`Par[0-9]_Real`	Hodnota přečteného parametru jako reálné číslo (floating point)
`BUSY`	Čtení nebylo dokončeno dokud BUSY = 1
`DONE`	Čtení bylo úspěšně dokončeno, výstupní hodnoty jsou platné
`ERROR`	Čtení bylo dokončeno s chybou, detaily chyby jsou na výstupu Error_Description a Error_Status
`Error_Description`	V případě chyby bloku udává textovou vysvětlivku chyby.
`Error_Status`	Může obsahovat číslo doplňující význam chyby.

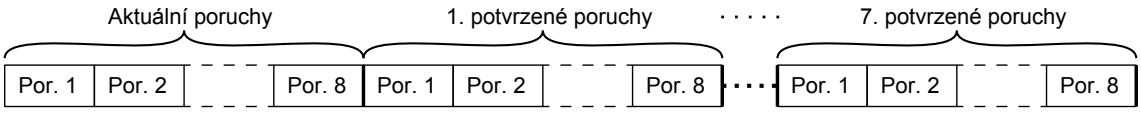
5.2 Funkční bloky pro specifické aplikace

Kromě obecných Proxy FB, které umožňují zápis nebo čtení až deseti libovolných parametrů, byly vytvořeny dva funkční bloky pro specifické aplikace s jednoduchým rozhraním:

- FB pro čtení zásobníku poruch a výstrah;
- FB pro nastavení absolutního snímače.

5.2.1 Funkční blok pro čtení poruch a výstrah

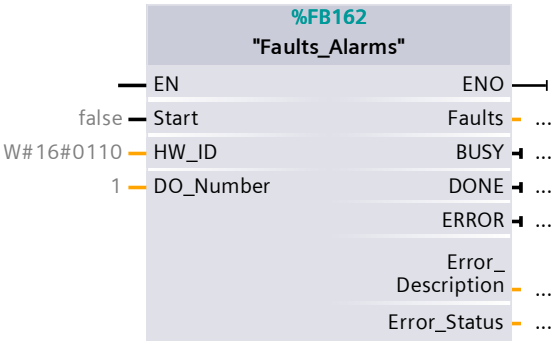
Frekvenční měniče Sinamics disponují pamětí se zásobníkem poruch a výstrah. Tento zásobník uchovává až 64 identifikátorů poruch a výstrah a jeho struktura je znázorněna na obr. 5.3. Zásobník obsahuje čísla poruch pro aktuální poruchy a také pro historii potvrzených poruch.



Obr. 5.3: Zásobník poruch a výstrah.

Poruchový stav měniče lze snadno identifikovat pomocí cyklické komunikace standardními telegramy. Jeden z bitů stavového slova ZSW1, které je součástí všech dostupných standardních telegramů, signalizuje poruchu v měniči. Použitím telegramu 111 lze číst aktuální číslo poruchy a výstrahy z měniče. Často ovšem jedna konkrétní událost způsobí hlášení několika poruch najednou. Chceme-li vyčíst ze zásobníku poruch všechny aktuální poruchy anebo minulé, již potvrzené poruchy, musíme použít parametrický přístup pomocí acyklické komunikace.

Zásobník poruch je uložen v parametru p945[0..63]. Pro čtení tohoto zásobníku acyklickým kanálem byl vytvořen nezávislý funkční blok s jednoduchým rozhraním. Vstupně/výstupní rozhraní bloku je vidět na obr. 5.4. Význam vstupů a výstupů je stejný jako u obecných Proxy FB a lze jej nalézt v tab. 5.2. Navíc má blok výstup `Faults`, který je polem UInt (0..63) a jeho hodnoty odpovídají jak jinak než číslům poruch a výstrah přečteným ze zásobníku.



Obr. 5.4: Vstupně/výstupní rozhraní bloku `Faults_Alarms`

Struktura FB je velmi podobná Proxy FB pro čtení parametrů, opět využíváme stavový diagram jako na obr. 5.1, rozdílné jsou především datové struktury pro čtení

a zápis bloky WRREC a RDREC. Ty nejsou v tomto případě vytvářeny dynamicky, ale jsou statické, protože se jedná vždy o čtení stejného parametru.

V obecných Proxy FB nepoužíváme čtení ani zápisu více indexovaných hodnot jednoho parametru pomocí atributů „Počet prvků“ a „Počet hodnot“ ve struktuře dat pro SFB WRREC a RDREC (tab. 3.3). Chceme-li zapisovat nebo číst více indexů stejného parametru, můžeme na vstupech obecných Proxy FB uvést stejné číslo parametru vícekrát a pokaždé jiný index. U bloku pro čtení zásobníku poruch a výstrah se naopak použití těchto atributů nabízí, neboť čteme jeden parametr s 64-mi indexovanými hodnotami.

5.2.2 Funkční blok pro nastavení absolutního snímače

Další funkční blok, který byl v rámci této práce naprogramován, je blok pro nastavení absolutního snímače polohy.

Absolutní snímače polohy jsou schopné okamžitě po zapnutí měniče, např. po výpadku napájení, udat skutečnou polohu motoru. To je jejich hlavní výhoda oproti inkrementálním snímačům, u kterých je po výpadku napájení nutné nejprve provést tzv. *referencování*, pomocí kterého je zjištěna skutečná poloha snímače. Referencování je možné provést různými způsoby, obvykle spočívá v nalezení koncových snímačů pohybem motoru do krajní polohy. To je ovšem v některých aplikacích vyloučeno, a tak je nutné použít snímače absolutní.

Tyto snímače disponují jednoznačným kódováním polohy v celém jejich rozsahu měření. Pro potřeby polohové regulace je nutné při uvedení do provozu nastavit vztah mezi jeho interním kódováním a skutečnou absolutní polohou v dané aplikaci ve zvolených jednotkách. Toto nastavení lze poté uložit do paměti ROM měniče, takže nastavení snímače není ztraceno po výpadku napájení.

Nastavení snímače je úloha spadající do kategorie parametrizace, je to časově nekritická úloha prováděná jednorázově pouze při nutnosti změny nastavení snímače, a proto je nanejvýš vhodné ji řešit pomocí acyklické komunikace.

Nastavení lze samozřejmě provést např. při úvodním uvedení do provozu pomocí systému Starter. Vytvoření funkčního bloku, který umožní jednoduše nastavit absolutní snímač z programu PLC, má tu výhodu, že nastavení lze provádět např. přímo z operátorského panelu. To je vyžadováno tam, kde hrozí časté výměny mechanických součástí strojů, kdy je nutné mechanicky rozpojit hřídel motoru s hnaným aparátem, nebo třeba v případě výměny motoru.

FB pro nastavení absolutního snímače je mimo jiné i příkladem použití obecných Proxy FB pro acyklickou komunikaci.

Úloha nastavení je zajímavá tím, že je nutné provést několik po sobě jdoucích kroků, zápisů a čtení různých parametrů. Celé nastavení snímače se skládá z těchto

kroků:

1. zápis aktuálního offsetu;
2. iniciace nastavení snímače;
3. čekání na úspěšné dokončení nastavení;
4. uložení nového nastavení do ROM.

Nastavení snímače provádíme v libovolném pracovním bodě. V prvním kroku zapíšeme aktuální polohu do parametru p2599. Druhý krok spočívá v zápisu parametru p2507, který udává status nastavení snímače. Přípustné hodnoty tohoto parametru jsou:

0: Nastala chyba při nastavování snímače.
1: Snímač není nastaven.
2: Snímač není nastaven a je iniciováno nastavení.
3: Snímač je nastaven.

Nastavení snímače tedy iniciujeme zápisem p2507 = 2. Po zapsání frekvenční měnič provádí nastavení, které není okamžité, ale může nějakou dobu trvat. V třetím kroku tedy opakujeme čtení parametru p2507 tak dlouho, dokud parametr není změněn na p2507 = 3 a poté pokračujeme posledním krokem, tedy uložením nastavení snímače do paměti ROM pomocí zápisu parametru p971 = 1. Přečteme-li v kroku 3 jinou hodnotu p2507 (0 nebo 1), ukončíme blok chybovým hlášením. Ukázka zdrojového kódu implementující tento algoritmus je uvedena níže.

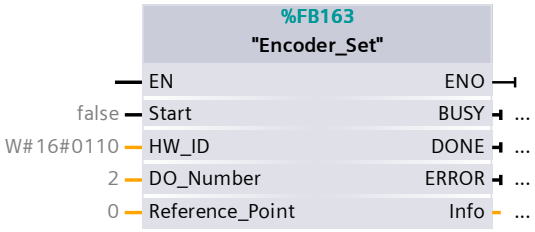
```
CASE #state OF
  0:
    IF #Start THEN
      #start_p2599 := true;
      #state := 1;
    END_IF;
  1:
    #start_p2599 := false;
    IF NOT #Write_p2599.BUSY THEN
      IF #Write_p2599.ERROR THEN
        #state := 6;
      ELSIF #Write_p2599.DONE THEN
        #start_p2507_wr := true;
        #state := 2;
      END_IF;
    END_IF;
  2:
    #start_p2507_wr := false;
    IF NOT #Write_p2507.BUSY THEN
      IF #Write_p2599.ERROR THEN
        #state := 6;
      ELSIF #Write_p2507.DONE THEN
        #start_p2507_rd := true;
        #state := 3;
      END_IF;
    END_IF;
  3:
```

```
    #start_p2507_rd := false;
    IF NOT #Read_p2507.BUSY THEN
      IF #Read_p2507.ERROR THEN
        #state := 6;
      ELSIF #Read_p2507.DONE THEN
        CASE #p2507 OF
          0: #Info := 'Error occured while adjusting.';
          1: #Info := 'Absolute encoder not adjusted.';
          2: #start_p2507_rd := true;
          3: #start_p971 := true;
             #state := 4;
        END_CASE;
      END_IF;
    END_IF;
  4:
    #start_p971 := false;
    IF NOT #Write_p971.ERROR THEN
      IF #Write_p971.ERROR THEN
        #state := 6;
      ELSIF #Write_p971.DONE THEN
        #state := 5;
      END_IF;
    END_IF;
  5: //done
    #BUSY := false;
    #ERROR := false;
    #DONE := true;
    IF NOT #Start THEN
      #state := 0;
    END_IF;
  6: //error
    #BUSY := false;
    #DONE := false;
    #ERROR := true;
    IF NOT #Start THEN
      #state := 0;
    END_IF;
END_CASE;
```

Stejně jako v obecných Proxy FB pro acyklickou komunikaci jsem k implementaci použil stavový automat. Od počátečního pokusu naprogramovat blok bez použití stavového automatu tak, aby byl jednodušší a mohl snadno demonstrovat použití FB `Write_Params` a `Read_Params`, bylo upuštěno. Algoritmus se zdál naopak nepřehledný a náchylný k chybám typu špatného ošetření spouštěcího vstupu `Start` proti několikanásobnému spuštění ještě před dokončením bloku a podobným. Naopak použití stavového automatu považuji za přímočaré řešení, které je poměrně dobře čitelné, je jasně strukturováno a především si lze snadno představit, jak bude výsledný program probíhat a zda někde nehrozí logická chyba.

Správné provedení jednotlivých kroků je zajištěno použitím obecných Proxy FB, které v sobě vždy zahrnují kontrolu odpovědi měniče, vypovídající o bezchybném provedení zápisu/čtení. Vstupně výstupní rozhraní bloku je na obr. 5.5 a význam většiny vstupů je opět stejný jako u vstupů obecných Proxy FB, viz tab. 5.2. Navíc

je použit vstup `Reference_Point`, kterým je zadána aktuální poloha pro nastavení snímače v jednotkách LU. Výstup `Info` dává informaci o úspěšném či neúspěšném nastavení snímače a popisuje možné příčiny chyby.



Obr. 5.5: Vstupně/výstupní rozhraní bloku `Encoder_Set`

5.3 Příkladová aplikace řízení pohonu

Dle zadání práce byla vytvořena také příkladová aplikace na řízení rychlosti a polohy pohonu. Aplikace kombinuje cyklický i acyklický způsob komunikace s frekvenčním měničem. Cyklická komunikace je implementována pomocí funkčního bloku `FB_for_ST111_MDI_simpl` pro komunikaci standardním telegramem 111 z knihovny *S7-1200-Sinamics-Lib-v7a*, viz kap. 4.2.2. Acyklická komunikace je implementována voláním vytvořených Proxy FB `Write_Params` a `Read_Params`.

V hlavním organizačním bloku jsou volány všechny funkční bloky se specifikací jen některých potřebných vstupních a výstupních parametrů:

```
"Read_Params_DB"(HW_ID:="PN-IO");
"Write_Params_DB"(HW_ID:="PN-IO");
"Write_Jog"(HW_ID:="PN-IO",
            DO_Number:=2,
            Par0_Number:=2585,
            Par0_Index:=0,
            Par0_Type:=4,
            Par1_Number:=2586,
            Par1_Index:=0,
            Par1_Type:=4);
"FB_for_ST111_MDI_simpl_DB"(HW_id:=WORD_TO_INT("telegram111[AI/AO]"),
                            ONOFF1:="Switch1",
                            EPOS_jog_1:="Button_Green",
                            EPOS_jog_2:="Button_Red",
                            Operation_enabled=>"Light_Green",
                            Fault_present=>"Light_Red");
```

K ovládání některých funkcí cyklického FB jsou použita hardwarová ovládací tlačítka umístěná vpravo nahoře na ukázkovém demokufru. Na rozdíl od FB pro acyklickou komunikaci je u tohoto bloku za vstup `HW_id` nutné dosadit identifikátor telegramu vloženého do objektu měniče při jeho konfiguraci.

Další vstupní a výstupní hodnoty volaných bloků jsou ukládány aplikací pro vizualizaci přímo do příslušných instančních datových bloků vytvořených při volání těchto FB.

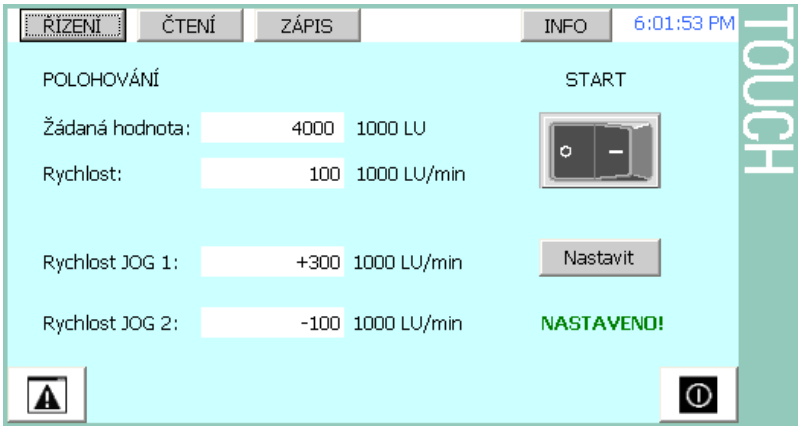
5.3.1 Vizualizace

Příkladová aplikace byla doplněna pouze jednoduchou vizualizací. Cílem není uvést komplexní projekt s propracovanou vizualizací, která pozbývá smysluplného použití, ale spíše demonstrovat, jak snadno lze pracovat s Proxy FB pro acyklickou komunikaci a zapisovat a číst libovolné parametry měniče. Vizualizace se skládá ze tří obrazovek popsaných níže.

Demonstrace nastavení absolutního snímače není ve vizualizaci zahrnuta, neboť použitý demokufr nedisponuje motorem s absolutním snímačem polohy. FB pro nastavení absolutního snímače byl ověřen připojením jiného motoru k frekvenčnímu měniči.

Obrazovka s řízením polohy

Na obr. 5.6 je obrazovka pro ovládání příkladové aplikace k řízení polohy a rychlosti pohonu. Editační pole, která slouží k zadání žádané polohy a rychlosti, a tlačítko pro start polohování zapisují přímo do oblasti vstupů instančního datového bloku `FB_for_S111_MDI_simpl_DB`. Tyto hodnoty jsou tedy přenášeny do měniče využitím cyklické komunikace standardním telegramem 111. Žádaná hodnota polohy je zadávána relativně.



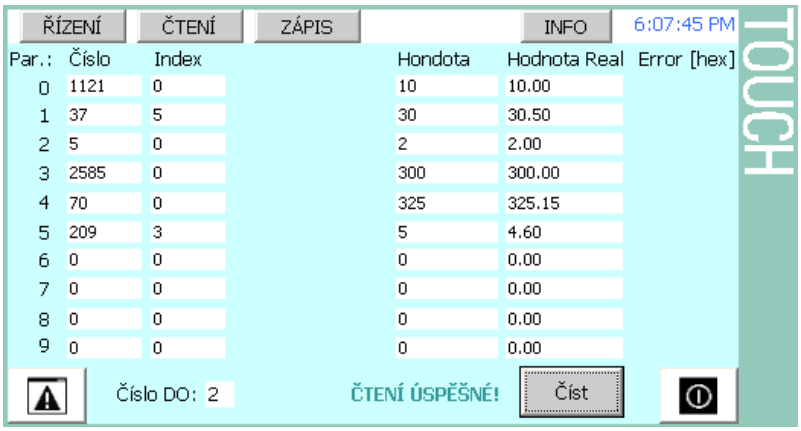
Obr. 5.6: Vizualizace pro polohové řízení

Jako ukázka použití Proxy FB pro acyklickou komunikaci zde slouží editace rychlosti tzv. *joggingu*, což je funkce frekvenčního měniče v režimu polohování, která slouží k pomalému pohybu motoru oběma směry pevně stanovenou rychlostí. Je

to funkce užitečná např. k ručnímu polohování zařízení. Jogging je ovládán vstupy `EPOS_jog_1` a `EPOS_jog_2` funkčního bloku `FB_for_S111_MDI_simpl`, kterým jsou přiřazeny signály ze zeleného a červeného tlačítka demokufru. Obvykle je jeho rychlost neměnná, stanovená při konfiguraci polohování v systému Starter. Využitím Proxy FB pro acyklickou komunikaci lze snadno implementovat změnu parametrů rychlosti joggingu (parametry p2585, p2586). Tlačítko *Nastavit* spustí zápis těchto parametrů vstupem `Start` bloku `Write_Params`, hodnoty rychlosti z editačních polí jsou zapsány přímo do oblasti vstupů instančního DB `Write_Jog` (Vstupy `Par[0,1]_Value`). Typ a čísla parametrů jsou specifikovány při volání bloku v OB1.

Obrazovky pro čtení a zápis libovolných parametrů

Pro celkovou demonstraci použití bloků `Write_Params` a `Read_Params` byly vypracovány vizualizační obrazovky uvedené na obr. 5.7 a 5.8.



Obr. 5.7: Vizualizace pro čtení libovolných parametrů

Obr. 5.8: Vizualizace pro zápis libovolných parametrů

S pomocí této vizualizace lze číst nebo zapisovat až 10 libovolných parametrů, které snadno specifikujeme zadáním čísla a indexu a v případě zápisu zadáme také typ a hodnotu požadovaných parametrů. Sloupec s chybou (demonstrováno na obr. 5.8 zadáním špatného typu parametru) udává číslo chyby při zápisu/čtení příslušného parametru. Význam chybové hodnoty potom najdeme v manuálu použitého frekvenčního měniče (pro Sinamics S110 v tab. 9-33 manuálu [8]).

ZÁVĚR

Tato práce se zabývá komunikací frekvenčních měničů Sinamics a programovatelných logických automatů Simatic S7-1200 po průmyslové sběrnici Profinet. Důraz je kladen na acyklickou komunikaci, která se používá pro přenos časově nekritických dat, určených např. pro parametrizaci frekvenčního měniče. Práce seznamuje s detaily acyklické komunikace po Profinetu a také s detaily komunikačního profilu PROFIdrive. Tento profil definuje komunikační model pro průmyslová zařízení s pohony, je podporován v produktech Siemens a je tedy ideálním modelem pro řešení cyklické i acyklické komunikace mezi danými zařízeními.

Praktická část práce řeší problematiku implementace tzv. *Proxy FB* pro acyklický zápis a čtení parametrů frekvenčního měniče. Základním požadavkem na výsledné Proxy FB je to, aby co možná nejvíce zjednodušovaly v současné době poměrně komplikované čtení a zápis parametrů měniče acyklickým kanálem Profinetu. Měly by tedy mít zejména jednoduché vstupně/výstupní rozhraní, měly by fungovat nezávisle, s libovolnými parametry a měly by být použitelné pro zápis nebo čtení více různých parametrů měniče najednou.

Ke splnění těchto požadavků bylo nutné navrhnout algoritmus, který z jednoduchého vstupního rozhraní bloku dynamicky sestaví komplexní strukturu dat požadovanou frekvenčním měničem pro zápis parametrů a také dokáže podobnou, měničem vrácenou strukturu dat, dekomponovat do jednoduchého výstupního rozhraní. Kromě toho musí zajistit správné asynchronní volání systémových funkčních bloků pro acyklický zápis a čtení z periferních zařízení na sběrnici Profinet a ošetřit co možná nejvíce chybových stavů, které při komunikaci mohou nastat. Tato implementace je nejpodstatnější součástí práce.

Proxy FB, které byly vytvořeny, splňují požadavky uvedeného zadání. Jejich funkčnost byla ověřena na ukázkovém demokufru s PLC Simatic S7-1200, frekvenčním měničem Sinamics S110 a malým dotykovým panelem. Přiložený projekt s jednoduchou vizualizací názorně ukazuje, že bloky fungují dobře. Je možné s nimi číst a zapisovat libovolné parametry měniče a vzhledem k ošetření chybových stavů spolu s chybovými výstupy, které tyto stavy indikují, jsou i poměrně robustní. Jejich vstupně/výstupní rozhraní je poměrně snadno pochopitelné, ovšem vzhledem k omezeným možnostem programování bloků v prostředí Step7 V11 poněkud dlouhé. V průběhu implementace byly zvažovány možné alternativy řešení, např. zda by nebylo lepší použít vstupů/výstupů typu pole nebo struktury. To by sice zkrátilo rozhraní bloků, ale neumožnilo jejich přímočaré použití např. v základním programovacím jazyce LAD.

Kromě obecných Proxy FB pro acyklickou komunikaci byly také navrženy další dva funkční bloky, které implementují některé specifické úlohy související s acyk-

lickou komunikací daných systémů. Jsou to bloky pro vyčítání zásobníku poruch a varování frekvenčního měniče a pro nastavení absolutního snímače servomotoru. Tyto bloky staví na poznatcích získaných během zpracování práce a částečně demonstrují využití obecných Proxy FB. Bloky bude možné použít při programování konkrétních aplikací s frekvenčními měniči Sinamics.

Na závěr je uvedena jednoduchá příkladová aplikace s polohovým řízením motoru. Aplikace mimo jiné demonstruje použití Proxy FB ke změně vybraných parametrů měniče, které nejsou běžně dostupné při cyklické komunikaci standardními telegramy.

LITERATURA

[1] *Communication Function Blocks on PROFIBUS DP and PROFINET IO* [online]. PROFIBUS Nutzerorganisation e.V., Listopad 2005 [cit. 2012-12-30] Dostupné z: <http://www.profibus.com/nc/downloads/downloads/communication-function-blocks-for-profibus-and-profinet/download/141/>.

[2] Fürsattel, M – Kosek, R. Profinet IRT – Řešení společnosti Siemens pro bezpečné ovládání rychlých procesů. *Automatizace*. 2009, roč. 52, č. 3, s. 160-161.

[3] *PROFIdrive System Description: Technolgy and Application* [online]. PROFIBUS Nutzerorganisation e.V., Květen 2011 [cit. 2012-12-29]. Dostupné z: <http://www.profibus.com/nc/downloads/downloads/profidrive-technology-and-application-system-description/download/11644/>.

[4] *PROFILE System Description: Technolgy and Application* [online]. PROFIBUS Nutzerorganisation e.V., Listopad 2010 [cit. 2013-03-28]. Dostupné z: <http://www.profibus.com/nc/download/technical-descriptions-books/downloads/profile-technology-and-application-system-description/download/9478/>.

[5] RAKUŠAN, O. *TIA na dosah* [online]. Siemens, 2011 [cit. 2012-05-16]. Dostupné z: <http://www1.siemens.cz/ad/current/content/data_files/automatizacni_systemy/mikrosystemy/simatic_s71200/prez_s7-1200-step7-basic-fw2_2011_cz.pdf>.

[6] *Siemens Industry Automation & Drive Technologies* [online]. Siemens AG, 2012 [cit. 2012-05-16]. Dostupné z: <https://www.cee.siemens.com/web/cz/cz/corporate/portal/home/industry/IADT/Pages/IADT.aspx>.

[7] *SINAMICS drives: Answers for industry* [online]. Siemens s. r. o., 2011 [cit. 2012-05-16]. Dostupné z: <http://www1.siemens.cz/ad/current/content/data_files/technika_pohonu/menice/_prospekty/brochure_sinamics_drives_2011_cz.pdf>.

[8] *SINAMICS S110 Function Manual* [online]. Siemens AG, 2011 [cit. 2012-05-16]. Dostupné z: <http://www1.siemens.cz/ad/current/content/data_files/technika_pohonu/menice/stridave_menice/nizkonapetove_menice/komdok_sinamics-s110-s120_v45_2012-04_ende.zip>.

[9] *SIMATIC S7-1200 — Micro Controller for Totally Integrated Automation* [online]. Siemens AG, 2009 [cit. 2012-05-16]. Dostupné z: <http://www1.siemens.cz/ad/current/content/data_files/katalogy/st70/news/cat_st-70-n_s7-1200_2009-04_en.pdf>.

[10] *SIMATIC S7-1200 Programmable controller: System Manual* [online]. Siemens AG, Duben 2012 [cit. 2013-01-01]. Dostupné z: <http://sie.ag/VKfxD3>.

[11] *SIMATIC, TIA Portal, STEP 7 Basic V10.5: Getting Started* [online]. Siemens AG, 2009 [cit. 2012-05-16]. Dostupné z: <http://www1.siemens.cz/ad/current/content/data_files/automatizacni_systemy/mikrosystemy/simatic_s71200/manualy/gsg_step7-basic-v10-5_2009-12_en.pdf>.

[12] TELELABOR FHD. *Learning Profinet* [online]. © 2006 [cit. 2012-05-16]. Dostupné z: <http://193.23.168.123:8888/profinetwbt_en/index.html>.

[13] *Totally Integrated Automation Portal* [software]. Verze V11 SP2 Update 5. Siemens AG, 2012 [cit. 2013-01-03].

[14] ZEZULKA, F. – HYNČICA, O. Průmyslový Ethernet VIII: Ethernet Powerlink, Profinet. *Automa*. 2008, č. 5, s. 62-66.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

CBA Component Based Automation

CSMA/CD Carrier Sense with Multiple Access and Collision Detection

DB Datový blok

DO Drive Object

FB Funkční blok

FBD Function Block Diagram

GSDML General Station Description Markup Language

HMI Human–Machine Interface

IO Input/Output

IP Internet Protocol

IRT Isochronous real-time

LAD Ladder Logic

LU Length Unit

NRT Non real-time

OB Organizační blok

PI PROFIBUS & PROFINET International

PLC Programmable Logic Controller

RT Real-time

SCL Structured Control Language

SFB Systémový funkční blok

ST Structured Text

TCP Transmission Control Protocol

TIA Totally Integrated Automation

UDP User Datagram Protocol