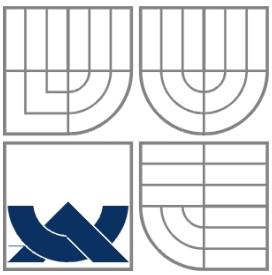


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

VYTVOŘENÍ VÝUKOVÝCH PODKLADŮ PRO PRÁCI VE VÝVOJOVÉM PROSTŘEDÍ STEP 7-MICRO/WIN

TEACHING DOCUMENTS CREATION FOR WORK IN THE DEVELOPMENT SYSTEM STEP 7-MICRO/WIN

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAKUB SVOBODA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. TOMÁŠ MARADA, Ph.D.**
SUPERVISOR

BRNO 2008

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav automatizace a informatiky
Akademický rok: 2007/08

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

student(ka): Svoboda Jakub

který/která studuje v **bakalářském studijním programu**

obor: **Aplikovaná informatika a řízení (3902R001)**

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Vytvoření výukových podkladů pro práci ve vývojovém prostředí STEP 7-Micro/WIN

v anglickém jazyce:

Teaching documents creation for work in the development system STEP 7-Micro/WIN

Stručná charakteristika problematiky úkolu:

Cílem práce je seznámit se s vývojovým prostředím STEP 7-Micro/WIN a vytvořit podklady použitelné pro výuku programování programovatelných automatů v tomto prostředí.

Cíle bakalářské práce:

1. Seznamte se s vývojovým prostředím STEP 7-Micro/WIN.
2. Seznamte se s programovatelnými automaty Simatic S7-200.
3. Proveďte popis vývojového prostředí STEP 7-Micro/WIN tak, aby byl použitelný pro výukové účely.
4. Vytvořte ukázkové úlohy s programovatelnými automaty Simatic S7-200. Tyto úlohy důkladně popište.

Seznam odborné literatury:

[1] Šmejkal, L., Martinásková, M., PLC a automatizace, Praha: BEN, 1999.
[2] Firemní materiály o programovatelných automatech fy Siemens S7-200.

Vedoucí bakalářské práce: Ing. Tomáš Marada, Ph.D.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2007/08.

V Brně, dne 11. 12. 2007

L.S.

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA STROJNÍHO INŽENÝRSTVÍ DĚKANÁT

Šeda

doc. RNDr. Ing. Miloš Šeda, Ph.D.
Ředitel ústavu

v.z. R. Kunz

doc. RNDr. Miroslav Doupovec, CSc.
Děkan fakulty

LICENČNÍ SMLOUVA

(na místo tohoto listu vložte vyplněný a podepsaný list formuláře licenčního ujednání)

ABSTRAKT

Tato bakalářská práce pojednává o programovatelném automatu Simatic S7-200, který je součástí vybavení laboratoře programovatelných automatů na Ústavu automatizace a informatiky Fakulty strojního inženýrství Vysokého učení technického v Brně.

Cílem bylo se nejdříve seznámit se samotným automatem, vývojovým prostředím STEP 7-Micro/WIN a v další fázi vytvořit souhrnné podklady pro podporu výuky. Současně vzniklo i několik vzorových úloh s různou úrovní obtížnosti, na kterých mohou studenti prakticky ověřit teorii.

ABSTRACT

This document deals with the programmable logic controller Simatic S7-200, which is a part of the equipment in the laboratory of programmable controllers at the Brno University of Technology (Faculty of Mechanical Engineering, Institute of Automation and Computer Science).

The purpose was to make self familiar with this controller and development system STEP 7-Micro/WIN at first and create a user guide which could be used as a teaching aid in the next phase. There were also created several sample tasks at different level of difficulty at the same time. Students can use them to prove the theory in practice.

KLÍČOVÁ SLOVA

PLC, Simatic S7-200, STEP 7-Micro/WIN.

KEYWORDS

PLC, Simatic S7-200, STEP 7-Micro/WIN.

PODĚKOVÁNÍ

Touto cestou bych rád poděkoval vedoucímu mé bakalářské práce Ing. Tomášovi Maradovi, Ph.D. za jeho čas, věcné podněty a připomínky při vypracovávání této bakalářské práce.

Obsah:

1 ÚVOD

Programovatelné automaty, neboli PLC (Programmable Logic Controler), mají v dnešní době poměrně pevnou pozici a nezastupitelné místo v řídicích systémech. Jejich uplatnění najdeme nejen v průmyslových aplikacích, ale i v automatizaci budov (osvětlení, vytápění) a dopravní technice. Díky širokému sortimentu a univerzálnosti nahrazují na postu řídicích prvků například reléové řízení. Na tento sektor zaměřuje svoji pozornost stále více firem. Tento fakt spolu s neustálým vývojem dává tušit, že se dočkáme jejich ještě mohutnějšího rozšíření.

Z konstrukčního hlediska můžeme programovatelné automaty rozdělit do dvou skupin, z nichž každá má své typické rysy a zástupce. Jedná se o automaty kompaktní a stavebnicové. Pro stavebnicové PLC je příznačné to, že se skládají z jednotlivých modulů, které jsou navzájem propojeny pomocí lišty v pomyslný jeden celek. Takto lze snadno vytvořit konfiguraci doslova na míru podle konkrétního použití a potřeb. Spadá sem například model S7-300 nebo S7-400. Na druhou stranu charakteristickým rysem kompaktních automatů je jejich celistvá konstrukce. Zvýšená odolnost je ovšem vykoupena nižší variabilitou v podobě menšího množství různých rozšíření.

Typickým představitelem je právě modelová řada Siemens S7-200, která je v několika kusech zastoupena v laboratoři programovatelných automatů v rámci Ústavu automatizace a informatiky Fakulty strojního inženýrství Vysokého učení technického v Brně. Variantě S7-200 v konfiguraci s CPU 224XP bude věnován následující text.

2 SIMATIC S7-200 CPU 224XP

Technické údaje uvedené v této kapitole byly převzaty z několika firemních materiálů firmy Siemens [2].

Programovatelný automat Simatic S7-200 ve variantě s CPU 224XP je výrobkem divize Automatizace a pohony firmy Siemens. Ta v dnešní době spolu s Phoenix Contact nebo Mitsubishi zaujímá jednu z čelních pozic na poli automatizační techniky.

Řada S7-200 spadá do kategorie takzvaných mikrosystémů, která tvoří pomyslný stupeň mezi logickými moduly LOGO! (ideální pro jednoduchá řešení) a robustními řídicími systémy jako například Simatic S7-300/400 s dostatečným výkonem a možnostmi pro realizaci náročnějších úloh. Podle krabicové konstrukce je typickým zástupcem rodiny kompaktních automatů s možností rozšíření o další jednotky (o konkrétních možnostech je pojednáváno v podkapitole 2.4).

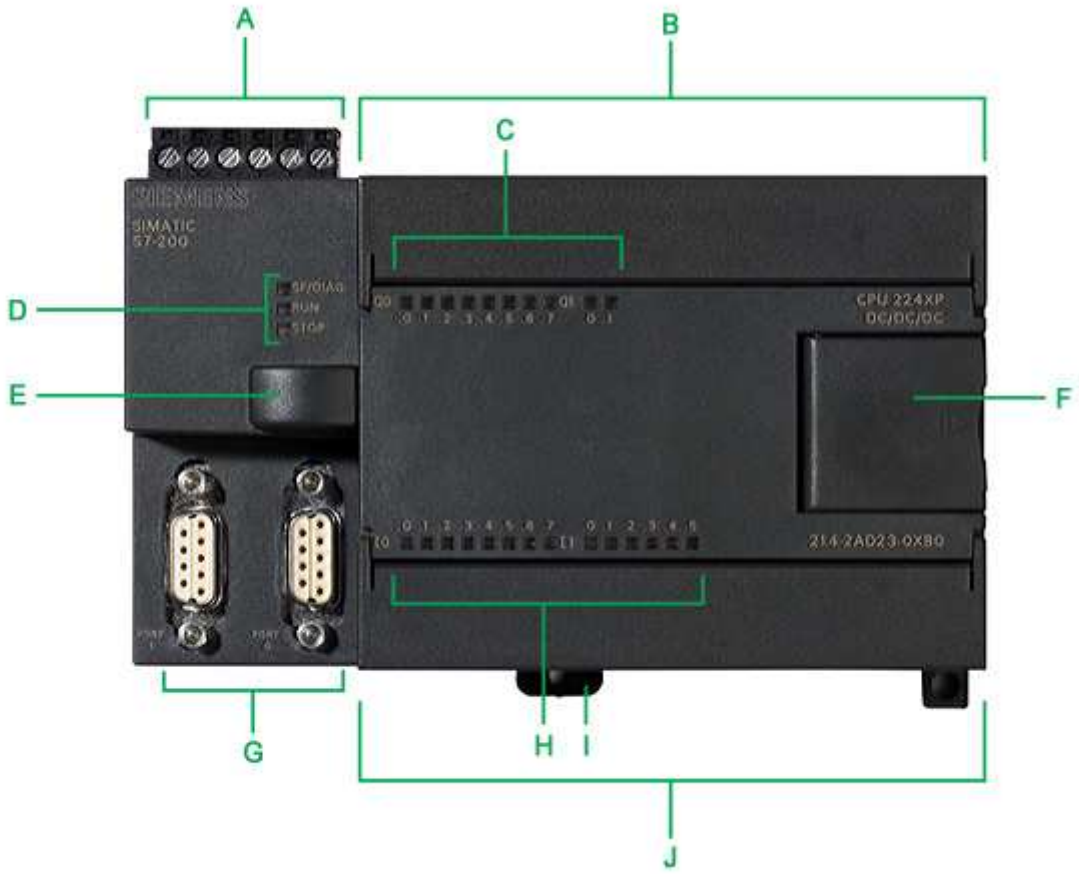
Obr. 2.1 Programovatelný automat Simatic S7-200 CPU 224Xp a příslušný napájecí zdroj Sitop Smart 2,5A / 24V. [1]

2.1 Parametry

- 14 digitálních vstupů
- 10 digitálních výstupů
- 2 analogové vstupy
- 1 analogový výstup
- 2 analogové potenciometry
- 2 komunikační porty RS-485
- programová paměť o velikosti 16kB
- datová paměť o velikost 10kB
- 10 vysokorychlostních čítačů
- podporované protokoly: PPI, MPI, Freeport
- rozměry: 140 x 80 x 62 [mm]

2.2 Popis ovládacích prvků

Na základním modulu (tedy bez jakéhokoliv rozšíření) lze zpravidla nalézt svorkovnice vstupů a výstupů, indikátory jejich stavu, přepínač režimu automatu s indikací, konektor pro komunikační a konfigurační kabel. Přesný popis všech prvků a jejich rozmístění znázorňuje následující obrázek.



Obr. 2.2 Popis jednotlivých prvků automatu.[1]

A analogové vstupy/výstupy
B svorkovnice digitálních výstupů a napájení
C signalizační diody výstupů
D stavové diody (SF/DIAG, RUN, STOP)
E krytka (konektor pro bateriový modul, časový modul, paměťový modul 64/256 kB)
F přístupová dvířka (analogové potenciometry, přepínač režimů CPU, konektor pro rozšiřovací moduly)
G komunikační porty RS-485
H signalizační diody vstupů
I úchyt na standardní lištu (DIN)
J svorkovnice digitálních vstupů

2.3 Pracovní režimy CPU

Programovatelný automat je možno provozovat ve dvou základních režimech – RUN a STOP. Mezi režimy lze volit buď přepínačem na těle automatu, softwarově v prostředí STEP 7-Micro/WIN (viz. kapitola 3.2), anebo pomocí instrukce STOP v samotném programu zastavit vykonávání programu.

RUN – v tomto režimu automat vykonává program, reaguje na signály na vstupech a posílá impulsy na výstupy. Jedná se tedy v podstatě o režim provozní. Program je možné editovat i za běhu, ale vzhledem k absenci prostoru pro otestování a ladění to může být velmi rizikové, protože změny se projeví ihned.

Samotný program je zpracováván v cyklu o několika hlavních krocích.

Obr. 2.3 Cyklický chod programu.

STOP – v režimu STOP se program nevykonává. Je to tedy stav využívaný pro vytváření, editaci a přenos uživatelského programu mezi automatem a počítačem.

TERM – při zvolení režimu TERM se aktuální režim nezmění. Umožňuje ovládat automat za pomoci prostředí STEP 7-Micro/WIN, vytvářet, editovat a monitorovat chod programu.

2.4 Rozšiřující moduly

K základní jednotce je možno připojit až sedm dalších modulů. Tím lze značně zvětšit její možnosti a rozšířit tak oblast použití. Jednotlivé přídavné dílce můžeme rozčlenit do několika kategorií: vstupně výstupní, komunikační, ovládací, monitorovací a speciální jednotky pro specifické úkony.

2.4.1 Vstupně výstupní

Slouží zpravidla k navýšení počtu digitálních, respektive analogových vstupů a výstupů. V aktuálním sortimentu figurují verze s různými vstupními či výstupními parametry a několika kombinacemi počtu konektorů.

EM 221 (vstupní)	- 8 digitálních vstupů 24 V DC - 8 digitálních vstupů 120/230 V AC - 16 digitálních vstupů 24 V DC
EM 222 (výstupní)	- 4 digitální tranzistorové výstupy 24 V DC, 5A - 4 digitální reléové výstupy, 10 A - 8 digitálních výstupů 24 V DC - 8 digitálních reléových výstupů - 8 digitálních výstupů 120/230 V AC
EM 223 (kombinovaný)	- 4 digitální vstupy 24 V DC, 4 digitální tranzistorové výstupy - 4 digitální vstupy 24 V DC, 4 digitální reléové výstupy - 8 digitálních vstupů 24 V DC, 8 digitálních tranzistorových výstupů - 8 digitálních vstupů 24 V DC, 8 digitálních reléových výstupů - 16 digitálních vstupů 24 V DC, 16 digitálních tranzistorových výstupů - 16 digitálních vstupů 24 V DC, 16 digitálních reléových výstupů - 32 digitálních vstupů 24 V DC, 32 digitálních tranzistorových výstupů - 32 digitálních vstupů 24 V DC, 32 digitálních reléových výstupů
EM 231 (vstupní)	- 4 analogové vstupy/12 bitů - 4 analogové vstupy/15 bitů, snímač pro měření odporů

- 4 analogové vstupy/15 bitů, snímač pro termočlánky
- 8 analogové vstupy/12 bitů
- 4 analogové vstupy/15 bitů, snímač pro měření odporů
- 8 analogové vstupy/15 bitů, snímač pro termočlánky

EM 232 (výstupní) - 4 analogové výstupy/12 bitů

EM 235 (kombinovaný) - 4 analogové vstupy/12 bitů, 1 analogový výstup/12 bitů

2.4.2 Komunikační

Umožní dálkovou konfiguraci, správu, údržbu či diagnostiku (EM 241), komunikaci přes FTP, email nebo HTML (CP 243-IT), popřípadě GSM/GPRS (SINAUT MD720-3). Dále připojení k průmyslovým sběrnicím typu PROFIBUS (EM 277) a AS-Interface (CP 243-2).

CP 243-1
- pro spojení s průmyslovým ethernetem
- umožňuje komunikovat s jinými automaty řady S7-200/300/400 nebo PC s OPC serverem

CP 243-2
- pro připojení na sběrnici AS-Interface
- umožňuje připojit stanici S7-200 jako master ke stanicím z nižších řad (LOGO!), které pracují jako slave

EM 241
- modem pro dálkovou komunikaci a správu
- umožňuje například sledovat stav automatu, přehrát uživatelský program, popřípadě jej ladit

EM 277
- pro připojení na sběrnici PROFIBUS-DP
- umožňuje připojit stanici S7-200 jako slave ke stanicím z vyšších řad (S7-300/400)

SINAUT MD 720-3
- umožňuje komunikovat přes GPRS a ovládat tak vzdálené stanice rozmístěné v oblastech pokrytých signálem GSM
- příkazy lze zadávat i prostřednictvím zpráv SMS

2.4.3 Ovládací a monitorovací

Sem spadají zejména zobrazovací displeje, které většinou zahrnují i rozličný počet ovládacích prvků. Jejich primární funkcí je přístup k aktuálním informacím a v omezené míře i konfigurace.

Text Display TD 100C
- čtyřřádkový textový displej s nastavitelným kontrastem
- pro zobrazení textů, nastavování vstupů a výstupů
- přímé připojení k rozhraní CPU bez nutnosti zvláštního napájení
- až 14 programovatelných tlačítek
- odolné zpracování; přizpůsobitelné rozhraní s použitím šablon a nástroje Keypad Designer integrovaného v STEP 7-Micro/WIN

Text Display TD 200
- dvouřádkový textový displej s podsvícením a nastavitelným kontrastem
- pro zobrazení textů, nastavování vstupů a výstupů
- přímé připojení k rozhranní CPU bez nutnosti zvláštního napájení
- odolné zpracování; 8 programovatelných tlačítek

Text Display TD 200C - vychází z TD 200

- až 20 programovatelných tlačítek
- přizpůsobitelné rozhraní s použitím šablon a nástroje Keypad Designer integrovaného v STEP 7-Micro/WIN

SIMATIC TP 177micro
- dotykový panel s plně grafickým displejem; čtyři úrovně podsvícení
- podpora 32 jazyků; knihovna grafických ikon
- široké možnosti přizpůsobení prostředí
- odolné zpracování; možnost horizontální i vertikální montáže

SIMATIC OP 73micro
- třípalcový monochromatický displej
- podpora 32 jazyků; knihovna grafických ikon
- 4 programovatelná a 8 systémových tlačítek
- odolné zpracování; rychlá montáž a konfigurace

2.4.4 Speciální

Například moduly pro měření teploty (RTD a TC), hmotnosti (SIWAREX MS), anebo pro ovládání servopohonů a krokových motorů (EM 253).

EM 253
- výkonný polohovací modul pro ovládání krokových motorů a servopohonů
- 4 analogové vstupy
- 4 pulzní výstupy s frekvencí pulzů od 12 Hz do 200 kHz
- polohovací úkony řeší samostatně - nezatěžuje CPU
- konfigurace prostřednictvím integrovaného rozhraní v STEP 7-Micro/WIN

SIWAREX MS
- všestranný váhový modul s rozlišením 16b a přesností 0,05%
- 30 nebo 50 měření za sekundu
- digitální filtrace signálu
- nastavitelné limity (min/max) – při překročení informuje CPU
- možnost připojení externího displeje
- možnost použití ve výbušném prostředí

2.5 Příklady použití

Níže jsou uvedeny některé aplikace automatu Simatic S7-200 v tuzemském prostředí.

Kostelecké uzeniny, a. s. [5]
- zařízení pro přepravu, rozřazování, vyskladňování a třídění prázdných přepravek
- Simatic S7-200 CPU 224, rozšiřující moduly EM 223

Keramika Horní Bříza, a. s [5]
- semimobilní třídicí linka pro třídění kaolinové suroviny
- Simatic S7-200 CPU 224, rozšiřující moduly EM 221 a EM 223

Mlékárna Olešnice [6]
- úpravy a balení mléka
- Simatic S7-200, Simatic S7-300, PC (InTouch), AS-Interface

PZP Komplet Semechnice [4]
- zařízení na solenoidové ventily
- Simatic S7-200

Alco Controls Kolín [4]
- zařízení na plnění ventilů, zařízení na testování ventilů

- Simatic S7-200

Brose CZ s. r. o. [3]
- životnostní testy zámků dveří automobilů
- Simatic S7 200, ovládací panel TP170

Cembrit Beroun [7]
- elektročást a SKŘ pro 2ks komorových podavačů
- Simatic S7 200, ovládací panel

Elektrárna Mělník I [7]
- distribuční centrum popílku - vyprazdňování sil
- Simatic S7 200, ovládací panel

Cinergetika Ústí nad Labem [7]
- pneudoprava komorovým podavačem a míchání stabilizátu
- 2x Simatic S7 200, ovládací panel

Škoda Mladá Boleslav [7]
- elektročást a řízení pro dva kusy komorových podavačů
- 2x Simatic S7 200, 2x operátorský panel

3 STEP 7-MICRO/WIN

STEP 7-Micro/WIN je prostředí určené pro programování a konfiguraci programovatelných automatů SIMATIC S7-200. Umožňuje programovat v jazyce mnemokódů (který je v rámci editoru označován jako STL, ale spíše odpovídá IL), kontaktních schémat (Ladder) anebo funkčních bloků (FBD). Mezi jednotlivými jazyky je možné libovolně přepínat.

Přímo v editoru je zakomponováno několik průvodců pro usnadnění konfigurace několika přídavných modulů, operátorských panelů a PID regulátorů.

Při tvorbě kapitol 3.2 a 3.3 bylo částečně použito terminologie z materiálů firmy Siemens [9].

3.1 Připojení a konfigurace automatu

Celá procedura připojení je záležitostí několika okamžiků. Předpokladem je funkční automat, konfigurační kabel a nainstalované prostředí STEP 7-Micro/WIN.

Fyzické připojení:

1. Připojit USB/PPI kabel ke konektoru na těle automatu.
2. Připojit USB/PPI kabel k portu USB na počítači.

Konfigurace v prostředí STEP 7-Micro/WIN:

3. Přepnout automat do režimu TERM.
4. Spustit prostředí STEP 7-Micro/WIN.
5. Přes nabídku *PLC → Type → Communications → Set PG/PC Interface* nastavit *PC/PPI cable(PPI)*
6. *PLC → Type → Communications → Set PG/PC Interface → Properties → Local Connection* zvolit *USB* a potvrdit stiskem *OK*
7. *PLC → Type → Communications → Double-Click to Refresh* a po detekování automatu potvrdit stiskem *OK*
8. *PLC → Type → Communications → Read PLC* a po načtení konkrétního typu potvrdit stiskem *OK*

3.2 Popis prostředí

Následující řádky přibližují význam jednotlivých položek, které jsou popsaný na snímku základní obrazovky editoru v příloze č. 2: Popis prostředí STEP 7-Micro/WIN.

3.2.1 Program Block (programový blok)

Slouží k samotnému vytváření a editaci programů. To je možné provádět buď pomocí jazyka kontaktních schémat (Ladder), jazyka funkčních bloků (FBD), nebo jazyka mnemokódů (STL). Mezi nimi se lze snadno přepínat přes nabídku *View → STL/Ladder/FBD*. Při přepnutí z jazyka mnemokódů (STL) do Ladder, případně FBD, mohou nastat potíže způsobené nevhodným členěním do dílčích networků. Řešením je tedy stávající program ještě více rozdělit.

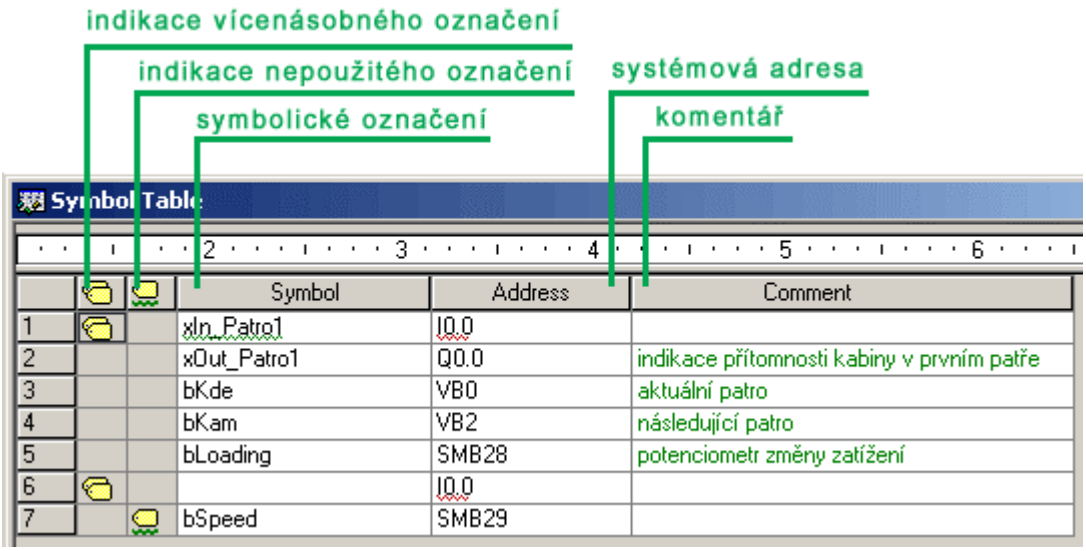
View → Component → Program Block

3.2.2 Symbol Table (tabulka symbolů)

Umožňuje přiřadit a upravovat symbolické názvy vstupů, výstupů a proměnných. Za pomocí přiřazeného názvu se na ně pak lze odkazovat v celém programu. Mezi symbolickým a systémovým označením je možné přepínat pomocí klávesové zkratky CTRL+Y anebo přes menu View → Symbolic Addressing. Jednoznačnou výhodou je tedy možnost původní označení podle potřeby přizpůsobit. Duplicitní a nepoužité symboly v programu jsou označeny ikonou

v příslušném sloupci.

Při zavádění symbolických názvů je kvůli přehlednosti vhodné držet se následujícího pravidla: před samotný název vložíme předponu v závislosti na zvoleném datovém typu (x – boolean, b – byte, w – word, d – double word, r – real, s – string). Je tedy zřejmé, že například `xIn_Patro1` je typu boolean a `bKde` typu byte.



Obr. 3.1 Tabulka symbolů.

Na obrázku je vidět, že výstupu s adresou `Q0.0` je přiřazeno symbolické označení `xOut_Patro1`. To znamená, že v celém programu se na příslušný výstup můžeme odkazovat právě pomocí tohoto názvu. Dále pak vstup s adresou `I0.0` je znovu nadefinován a potenciometr `SMB29` je sice regulérně označen, ale v celém programu nepoužit.

Kromě tabulky editovatelné uživatelem je automaticky vytvářena další tabulka, ve které figurují údaje o podprogramech (subroutines) a přerušeních (interrupts). Ty je možné změnit pouze přejmenováním příslušného podprogramu nebo přerušení.

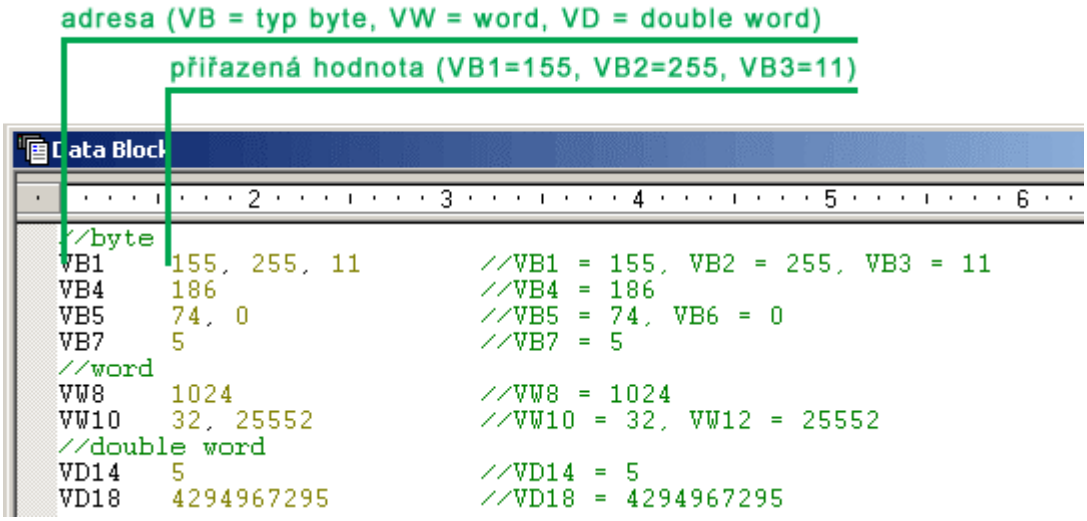
View → Component → Symbol Table

3.2.3 Data Block (datový blok)

Uchovává hodnoty proměnných, lze jej také použít pro definování počátečních hodnot. Data, respektive proměnné, mohou být libovolného typu, potažmo velikosti – byte (VB), word (VW), doubleword (VD). Jedinou podmínkou je přiřazování do části paměti označované V.

Při odřádkování pomocí Ctrl+Enter dojde k automatickému navýšení adresy na následující volnou hodnotu. Na jednom řádku lze také uvést více hodnot stejného datového typu, oddělené čárkami. Je však nutné, aby první údaj byl správně adresován. Ostatní jsou opět automaticky přiřazeny nejbližším volným adresám.

View → Component → Data Block

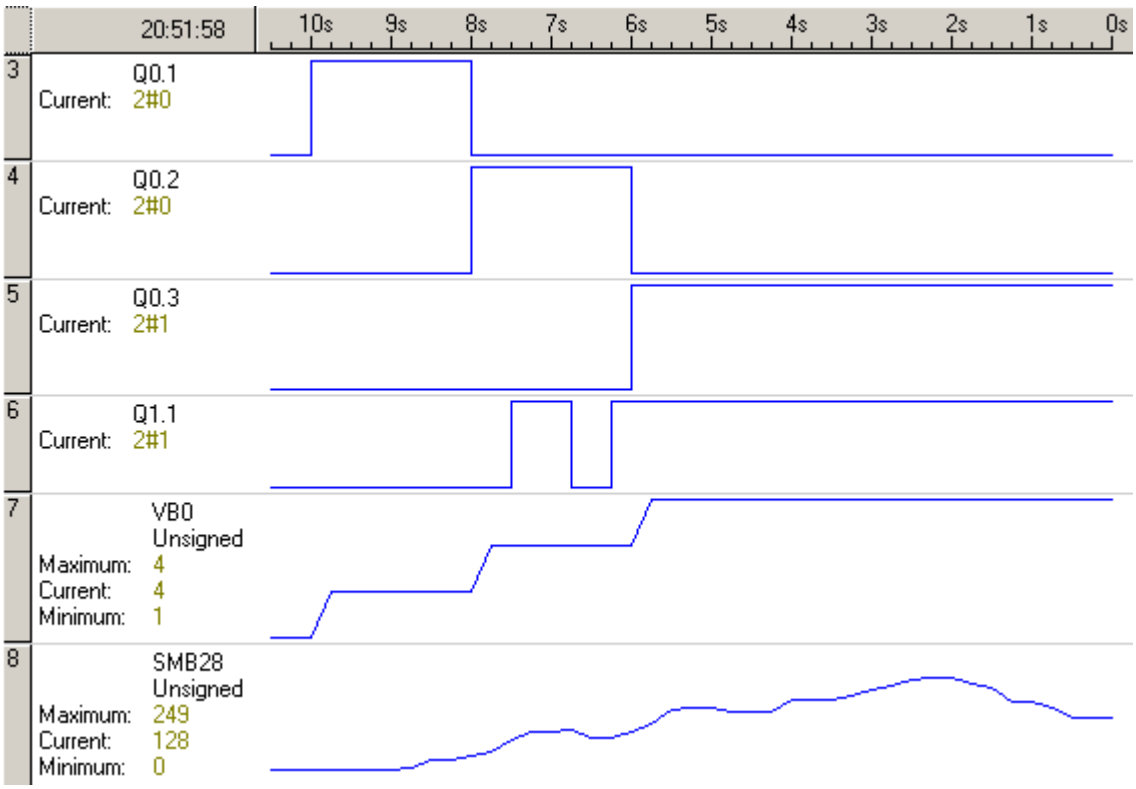


Obr. 3.2 Datový blok.

3.2.4 Status Chart (stavový diagram)

Dovoluje monitorovat a upravovat hodnoty vstupů, výstupů, časovačů, proměnných a dalších prvků za běhu programu. Hodnoty mohou být zobrazeny v grafickém diagramu se závislostí na čase (viz obr. 3.3), v tabulce (viz obr. 3.4), nebo přímo ve schématu FBD nebo Ladder (viz obr. 3.5). Pokud je zvolena poslední varianta je nutné, aby oba programy (v paměti automatu i v prostředí STEP 7-Micro/WIN) byly shodné.

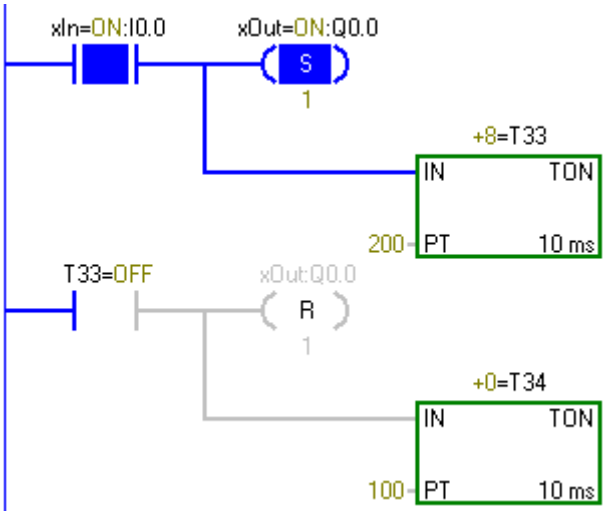
View → Component → Status Chart



Obr. 3.3 Stavový diagram – grafické znázornění.

	Address	Format	Current Value	New Value
1	I1.1	Bit	2#0	
2	Q0.0	Bit	2#0	
3	Q0.1	Bit	2#0	
4	Q0.2	Bit	2#0	
5	Q0.3	Bit	2#1	
6	Q1.1	Bit	2#1	
7	VB0	Unsigned	4	
8	SMB28	Unsigned	129	

Obr. 3.4 Stavový diagram – přehled v tabulce.



Obr. 3.5 Monitorování běhu programu v jazyce Ladder.

3.2.5 System block (systémový blok)

Umožňuje nastavit hodnoty digitálních a analogových výstupů při přechodu do režimu STOP; nadefinovat oblasti paměti, které budou zálohovány při výpadku napájení; nastavit filtraci signálu digitálních a analogových vstupů; nastavit zachytávání krátkých pulsů na vstupech; konfigurovat LED kontrolku SF/DIAG; nastavit velikost paměti pro uživatelský program; nastavit ochranu heslem pro různé úkony.

View → Component → System block

3.2.6 Cross Reference (tabulka křížových odkazů)

Poskytuje bližší informace o jednotlivých proměnných. Najdeme zde přesný popis místa použití (blok, network, řádek) a také v jakém kontextu figurovaly. Záložky Bit Usage a Byte Usage obsahují bližší statistiku o využití paměti.

View → Component → Cross Reference

3.2.7 Communications (komunikace)

Slouží k nastavení a ověření komunikace mezi počítačem a automatem. Zobrazuje také informace o aktuálně nainstalovaném a ovladači a jeho parametrech, který se vybírá v dialogu Set PG/PC Interface.

View → Component → Communications

3.2.8 Set PG/PC Interface (nastavit rozhraní PG/PC)

Umožňuje přidávání, odebírání, konfiguraci komunikačních ovladačů a také výběr konkrétního komunikačního rozhraní (PPI Multi-master kabel, CP komunikační karta, ethernetová komunikační karta). U zvoleného ovladače je dále možno měnit adresu, přenosovou rychlost, typ portu (USB/COM) a podobně.

View → Component → Set PG/PC Interface

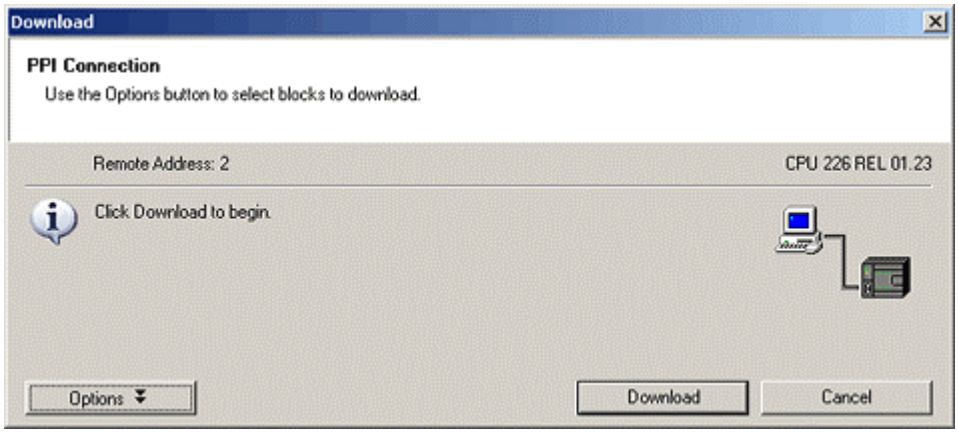
3.2.9 Upload a download programu

Upload – zkopíruje program z paměti programovatelného automatu do stávajícího projektu v prostředí STEP 7-Micro/WIN. Z trvalé paměti je možné přesunout obsah Program Block, Data Block a System Block. V případě připojeného paměťového modulu pak navíc Recepies a Data Log Configuration. Předpokladem pro úspěšný přenos je automat v režimu STOP a fungující spojení s počítačem.

File → Upload

Download – slouží pro přesun aktuálně otevřeného programu do paměti programovatelného automatu. Do trvalé paměti je možné přesunout obsah Program Block, Data Block a System Block. V případě připojeného paměťového modulu pak ještě Recepies a Data Log Configuration. Předpokladem pro úspěšný přenos je automat v režimu STOP a fungující spojení s počítačem. Předcházející obsah je přepsán.

File → Download



Obr. 3.6 Dialog downloadu programu do automatu.

3.2.10 Kompilování programu

Compile (zkompilovat) – zkompiluje kód v aktuálním okně (Program Block nebo Data Block). Výsledek se zobrazí v Output Window.

PLC → Compile

Compile All (zkompilovat vše) – provede kompilaci celého projektu. Výsledek se zobrazí v Output Window.

PLC → Compile All

Obr. 3.7 Výsledek kompilace zobrazený v Output Window.

3.2.11 Softwarová změna režimu automatu

Run (spustit) – přepne automat do režimu RUN. Podmínkou je přepínač režimů v poloze RUN nebo TERM a fungující spojení s počítačem.

PLC → RUN

Stop (zastavit) – přepne automat do režimu STOP. Podmínkou je přepínač režimů v poloze RUN nebo TERM a fungující spojení s počítačem.

PLC → STOP

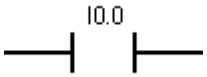
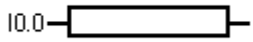
3.3 Přehled základních příkazů

Vzhledem k prostorové náročnosti byla z celé instrukční sady vybrána pouze část. Jde jednak o základní a nejčastěji používané příkazy a také ty bezprostředně nutné pro řešení úloh v kapitole 4.

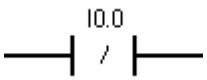
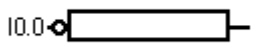
Příkazy lze do těla programu vkládat přetažením z Instruction Tree (stromu instrukcí), pomocí tlačítek Contact, Coil, Box při programování v zobrazení FBD a Ladder, anebo vepsáním v případě STL.

3.3.1 Contacts (kontakty)

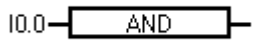
LD (načíst) – přesune parametr příkazu na vrchol zásobníku, v důsledku čehož poslední hodnota v zásobníku zanikne.
parametr: typu bit, např. adresa vstupu/výstupu

STL	Ladder	FBD
LD I0.0		

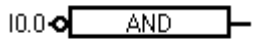
LDN (načíst negované) – parametr příkazu zneguje a přesune jej na vrchol zásobníku, v důsledku čehož poslední hodnota v zásobníku zanikne.
parametr: typu bit, např. adresa vstupu/výstupu

STL	Ladder	FBD
LDN I0.0		

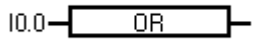
A (AND) – provede logický součin parametru příkazu a bitu na vrcholu zásobníku.
parametr: typu bit, např. adresa vstupu/výstupu

STL	Ladder	FBD
A I0.0	prvky v sérii	

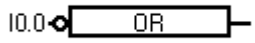
AN () – provede logický součin znegovaného parametru příkazu a bitu na vrcholu zásobníku.
parametr: typu bit, např. adresa vstupu/výstupu

STL	Ladder	FBD
AN I0.0	prvky v sérii	

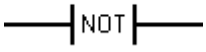
O (OR) – provede logický součet parametru příkazu a bitu na vrcholu zásobníku.
parametr: typu bit, např. adresa vstupu/výstupu

STL	Ladder	FBD
O I0.0	prvky paralelně	

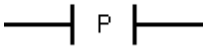
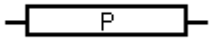
ON () – provede logický součet znegovaného parametru příkazu a bitu na vrcholu zásobníku.
parametr: typu bit, např. adresa vstupu/výstupu

STL	Ladder	FBD
O I0.0	prvky paralelně	

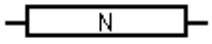
NOT – invertuje logickou hodnotu bitu na vrcholu zásobníku.

STL	Ladder	FBD
NOT		

EU (vzestupná hrana) – pokud je detekována vzestupná hrana (0 → 1), změní hodnotu na vrcholu zásobníku na log. 1.

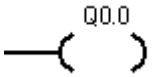
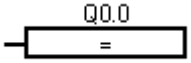
STL	Ladder	FBD
EU		

ED (sestupná hrana) – pokud je detekována sestupná hrana (1 → 0), změní hodnotu na vrcholu zásobníku na log. 1.

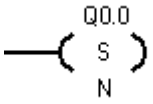
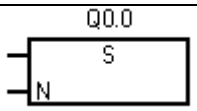
STL	Ladder	FBD
ED		

3.3.2 Coils (cívky)

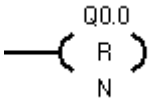
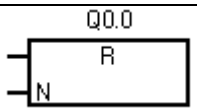
= (výstup) – zapíše hodnotu bitu na vrcholu zásobníku na určený výstup.
parametr: adresa výstupu

STL	Ladder	FBD
= Q0.0		

SET (nastavení) – přiřadí log. 1 specifikovanému výstupu a N-1 dalším. V případě, že N=1 nastaví pouze specifikovaný výstup.
parametr: adresa výstupu, počet výstupů

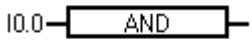
STL	Ladder	FBD
S Q0.0, N		

RESET (vynulování) - přiřadí log. 0 specifikovanému výstupu a N-1 dalším. V případě, že N=1 vynuluje pouze specifikovaný výstup.
parametr: adresa výstupu, počet výstupů

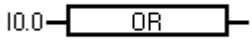
STL	Ladder	FBD
R Q0.0, N		

3.3.3 Logic Stack Instructions (operace se zásobníkem)

ALD (logický součin) – provede logický součin prvních dvou hodnot v zásobníku a výsledek uloží zpět na jeho vrchol, zároveň se využití zásobníku sníží o jeden prvek.

STL	Ladder	FBD
ALD	prvky v sérii	

OLD (logický součet) – provede logický součet prvních dvou hodnot v zásobníku a výsledek uloží zpět na jeho vrchol, zároveň se využití zásobníku sníží o jeden prvek.

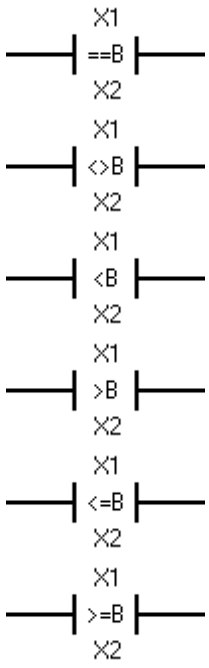
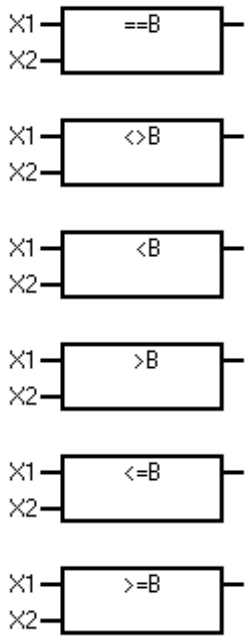
STL	Ladder	FBD
OLD	prvky paralelně	

3.3.4 Compare Instructions (instrukce porovnávání)

Provede porovnání dvou hodnot stejného datového typu. Na základě výsledku přiřadí hodnotu log. 1 zásobníku (STL), kontaktu (Ladder) anebo výstupu (FBD).

pozn.: Porovnávat lze hodnoty typu byte, word, double word a real. Příkazy se liší příznakem (B – byte, W – word, D – double word, R – real). Použití je vysvětleno na porovnávání hodnot typu byte (B).

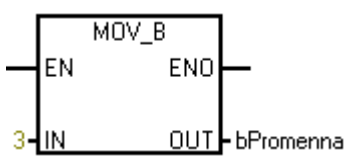
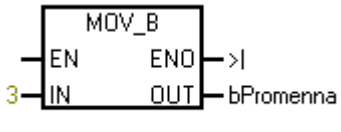
STL LDB – porovná dvě hodnoty X1 a X2 typu byte. Je-li splněna podmínka (výsledek porovnání je pravdivý), umístí na vrchol zásobníku log. 1.
AB – porovná dvě hodnoty X1 a X2 typu byte. Je-li splněna podmínka (výsledek porovnán je pravdivý), provede se logický součin hodnoty na vrcholu zásobníku a log. 1.
OB – porovná dvě hodnoty X1 a X2 typu byte. Je-li splněna podmínka (výsledek porovnání je pravdivý), provede se logický součet hodnoty na vrcholu zásobníku a log. 1.
Ladder – porovná dvě hodnoty X1 a X2 typu byte. Je-li splněna podmínka (výsledek porovnání je pravdivý), přiřadí kontaktu hodnotu log. 1.
FBD – porovná dvě hodnoty X1 a X2 typu byte. Je-li splněna podmínka (výsledek porovnání je pravdivý), přiřadí výstupu hodnotu log. 1.
parametr: dvě hodnoty (proměnné) příslušného datového typu

STL	Ladder	FBD
LDB= X1, X2 LDB<> X1, X2 LDB< X1, X2 LDB> X1, X2 LDB<= X1, X2 LDB>= X1, X2 AB= X1, X2 AB<> X1, X2 AB< X1, X2 AB> X1, X2 AB<= X1, X2 AB>= X1, X2 OB= X1, X2 OB<> X1, X2 OB< X1, X2 OB> X1, X2 OB<= X1, X2 OB>= X1, X2		

3.3.5 Move Instructions (instrukce přesunu)

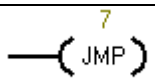
Přesune hodnotu z paměťového místa IN do paměťového místa OUT.

pozn.: Přesouvat lze hodnoty typu byte, word, double word a real. Příkazy se liší příznakem (B – byte, W – word, D – double word, R – real). Použití je vysvětleno na přesouvání hodnot typu byte (B).
parametr: hodnota a proměnná příslušného paměťového typu

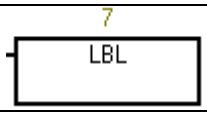
STL	Ladder	FBD
MOVB IN, OUT		

3.3.6 Program Control Instructions (instrukce pro řízení programu)

JMP (skok) – provede skok na místo v programu označené návěštím LBL.
parametr: adresa návěští

STL	Ladder	FBD
JMP 7		

LBL (návěští) – označuje místo pro doskok. Pro správnou funkčnost je nutné, aby se instrukce skoku (JMP) i samotné návěští (LBL) nacházely ve stejné části programu.
parametr: adresa návěští

STL	Ladder	FBD
LBL 7		

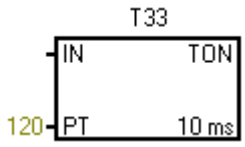
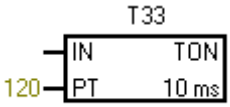
3.3.7 Timer Instructions (instrukce časovače)

TON (časovač zapnutí) – začne odpočítávat čas od okamžiku sepnutí vstupu (IN). Instrukční sada automatu S7-200 CPU 214XP obsahuje 3 typy časovačů. Jedná se o 1ms, 10ms a 100ms časovače, které se liší tzv. časovou základnou. Ta vyjadřuje, s jakou přesností bude čas měřen. Před použitím časovače je nutné jej vynulovat pomocí instrukce RESET.

Typ časovače (časová základna)	Označení časovačů	Max. měřitelný úsek [ms]
1ms	T32, T96	1*2^15 = 32768
10ms	T33 až T36, T97 až T100	10*2^15 = 327680
100ms	T37 až T63, T101 až T255	100*2^15 = 3276800

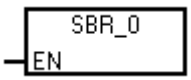
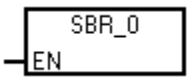
Tab. 3.8 Přehled časovačů.[9]

Například pro odměření 5s lze použít 1ms časovač s parametrem 5000, 10ms s parametrem 500 anebo 100ms s parametrem 50.
parametr: typ časovače (časová základna), měřený čas

STL	Ladder	FBD
TON T33, 120		

3.3.8 Subroutine Instructions (instrukce podprogramů)

CALL (volání podprogramu) – po zadání instrukce CALL pokračuje běh programu prováděním příkazů v podprogramu. Jakmile je vykonána poslední instrukce podprogramu, pokračuje se v původním programu.
parametr: název podprogramu

STL	Ladder	FBD
CALL SBR_0		

4 NAVRŽENÉ ÚLOHY

Pět řešených úloh je seřazeno vzestupně podle náročnosti. V každé z nich je demonstrováno použití několika příkazů probraných v kapitole 3.3. U všech lze nalézt krátký úvod, zadání problému, teoretické řešení, postup naprogramování v prostředí STEP 7-Micro/WIN a výsledná schémata spolu s popisem použitých vstupů a výstupů.

4.1 Realizace logické funkce

Úvodní úloha má za cíl zejména přiblížit použití základních příkazů a seznámit se samotným prostředím.

4.1.1 Zadání

Realizujte logickou funkci se třemi vstupy, která je dána následující pravdivostní tabulkou.

x_1	x_2	x_3	y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Tab. 4.1 Realizace logické funkce - pravdivostní tabulka.

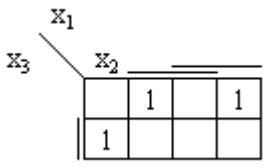
4.1.2 Řešení

Nejdříve si funkci vyjádříme například pomocí logického součtu logických součinů neboli ve tvaru disjunktivní normální formy (DNF).

$$y = \overline{x_1} \cdot \overline{x_2} \cdot x_3 + \overline{x_1} \cdot x_2 \cdot \overline{x_3} + x_1 \cdot \overline{x_2} \cdot \overline{x_3}$$

Obr. 4.2 Realizace logické funkce - funkce ve tvaru DNF.

Při pokusu o minimalizaci zjistíme, že nelze vytvořit žádné dvojice ani čtveřice. Funkce je tedy již v minimálním tvaru a v této podobě ji lze i snadno zapsat v prostředí Step-7 MicroWin.



Obr. 4.3 Realizace logické funkce - Karnaughova mapa.

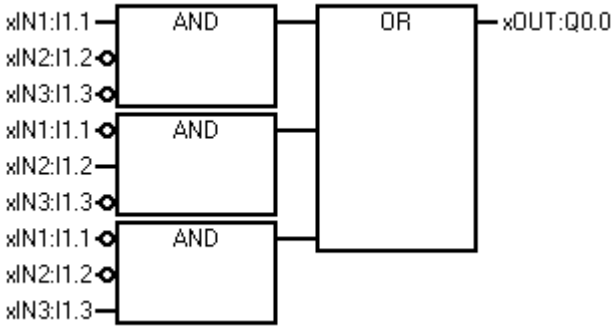
4.1.3 Realizace v prostředí STEP 7-Micro/WIN

Definice vstupů a výstupů v tabulce symbolů (zahrnuje pouze tři vstupy a jeden výstup) by mohla vypadat následovně:

Symbol	Address	Comment
xIN1	I1.1	
xIN2	I1.2	
xIN3	I1.3	
xOUT	Q0.0	

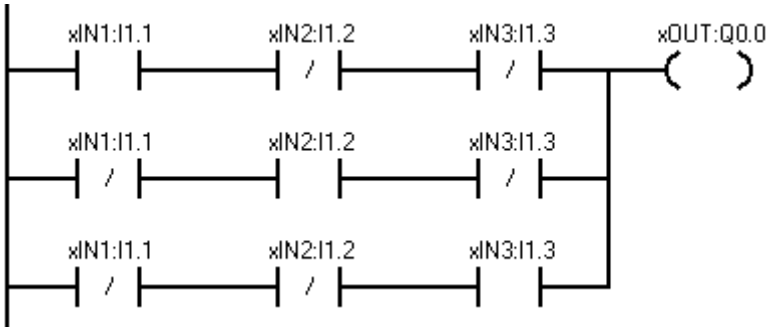
Obr. 4.4 Realizace logické funkce – tabulka symbolů.

Vzhledem k poměrně jednoduchému zadání se zdá být zápis pomocí funkčních bloků nejnázornějším. Je zde vidět přímá analogie s výchozí rovnicí a to zejména v podobě použití tří vícevstupových hradel typu AND, jednoho typu OR a také ve znegování potřebných vstupů.



Obr. 4.5 Realizace logické funkce – schéma FBD.

Výsledné schéma v jazyce Ladder se skládá ze tří paralelních větví (reprezentujících logický součet), tvořených jednotlivými prvky v sérii (reprezentující logické součiny).

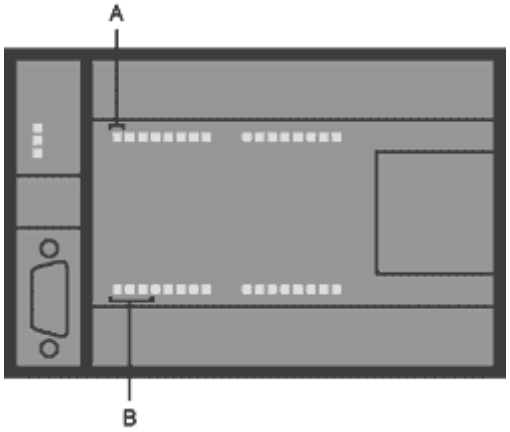


Obr. 4.6 Realizace logické funkce – schéma Ladder.

Oproti zápisu pomocí funkčních bloků (FBD), kde můžeme volit počet vstupů, mají v jazyce STL instrukce typu `AND` a `OLD` pouze dva parametry. Z tohoto důvodu budeme muset logický součet provést postupně nadvakrát. Je také vhodné dbát na průběžné provádění logického součtu s ohledem na hospodárné využívání zásobníku a jeho omezenou kapacitu. V našem příkladu by obě instrukce `OLD` mohly být vykonány na konci naráz. V případě rozsáhlejšího projektu by ovšem další pozice v zásobníku mohla chybět.

```
Network 1
LD    xIN1   //přesun hodnoty vstupu xIN1 na vrchol zásobníku
AN    xIN2   //součin hodnoty na vrcholu zásobníku a negované xIN2,
             uloží na vrchol zásobníku
AN    xIN3   //součin hodnoty na vrcholu zásobníku a negované xIN2,
```

```
                uloží na vrchol zásobníku

LDN   xIN1      //přesun negované hodnoty vstupu xIN1 na vrchol zásobníku
A     xIN2      //součin hodnoty na vrcholu zásobníku a xIN2, uloží na
                vrchol zásobníku
AN    xIN3      //součin hodnoty na vrcholu zásobníku a negované xIN2,
                uloží na vrchol zásobníku

OLD             //logický součet prvních dvou položek v zásobníku

LDN   xIN1      //přesun negované hodnoty vstupu xIN1 na vrchol zásobníku
AN    xIN2      //součin hodnoty na vrcholu zásobníku a negované xIN2,
                uloží na vrchol zásobníku
A     xIN3      //součin hodnoty na vrcholu zásobníku a xIN2, uloží na
                vrchol zásobníku

OLD             //logický součet prvních dvou položek v zásobníku

=     xOUT      //přiřazení výsledné hodnoty výstupu xOUT
```



A Stav výstupu – y – (svítí = log 1)
B Stav vstupů – x_1, x_2, x_3 – (svítí = log 1)

Obr. 4.7 Realizace logické funkce – vstupy a výstupy.

4.2 Uzávěrka diferenciálu

Cílem této úlohy je více přiblížit postup zpracování reálného problému. Ve výsledku se sice podobá předešlému příkladu, ale hlavním záměrem bylo zahrnout všechny fáze při řešení úkolu. Tedy propracovat se od obecného zadání, přes pochopení principu a rozvržení úlohy až k samotnému naprogramování a následnému odzkoušení.

4.2.1 Zadání

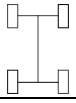
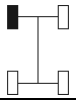
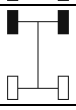
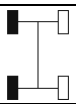
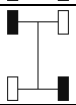
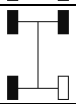
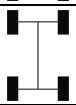
Realizujte program, jehož výstupem bude impuls pro uzamčení nápravového diferenciálu v závislosti na počtu protáčejících se kol.

4.2.2 Řešení

Diferenciál je zařízení umožňující různou frekvenci otáčení kol na jedné nápravě. I přes nespočet výhod (lepší ovladatelnost, nižší namáhání karosérie), se lze setkat s podmínkami, kdy je jeho použití nevýhodné a omezující. Jedná se zejména o případy prokluzu některého z poháněných kol v důsledku provozu na nepříznivém povrchu. Za těchto okolností je ideálním řešením uzamknout diferenciál, což může být v reálu provedeno buď manuálně, anebo automaticky. [10]

Předpokládejme, že automobil má stálý pohon čtyř kol a na každém z nich je snímač,

který poskytuje informace o případném prokluzování. Aby bylo možné adekvátně reagovat, je nutné znát modelové situace, které mohou v reálu nastat. Ty popisuje následující tabulka.

Situace	Schéma	Popis	Uzamknout diferenciál
A		neprokluzuje žádné kolo	ne
B		prokluzuje pouze jedno kolo	ne
C		prokluzují obě kola na stejné nápravě	ne
D		prokluzují obě kola na stejné straně	ano
E		prokluzují dvě kola úhlopříčně	ano
F		prokluzují tři kola	ano
G		prokluzují všechna čtyři kola	ne

Tab. 4.8 Modelové situace prokluzu kol.

Jednotlivé stavy lze vyjádřit i pomocí pravdivostní tabulky. Logická jednička na vstupech x_1 až x_4 (tedy výstup příslušného snímače) značí situace, kdy dochází k protáčení daného kola. Hodnota logické jedničky na výstupu y_1 je pak impulsem pro uzamčení diferenciálu. Sloupec „Situace“ poskytuje odkaz na předchozí tabulku.

x_1	x_2	x_3	x_4	y_1	Situace
0	0	0	0	0	A
0	0	0	1	0	B
0	0	1	0	0	B
0	0	1	1	0	C
0	1	0	0	0	B
0	1	0	1	1	D
0	1	1	0	1	E
0	1	1	1	1	F
1	0	0	0	0	B
1	0	0	1	1	E
1	0	1	0	1	D
1	0	1	1	1	F
1	1	0	0	0	C
1	1	0	1	1	F
1	1	1	0	1	F
1	1	1	1	0	G

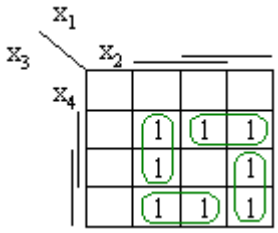
Tab. 4.9 Uzávěrka diferenciálu - pravdivostní tabulka.

Po převedení na tvar logického součtu logických součinů (DNF) bude funkce vypadat takto.

$$y_1 = \overline{x_1} \cdot x_2 \cdot \overline{x_3} \cdot x_4 + \overline{x_1} \cdot x_2 \cdot x_3 \cdot \overline{x_4} + \overline{x_1} \cdot x_2 \cdot x_3 \cdot x_4 + x_1 \cdot \overline{x_2} \cdot \overline{x_3} \cdot x_4 +$$
$$+ x_1 \cdot \overline{x_2} \cdot x_3 \cdot \overline{x_4} + x_1 \cdot \overline{x_2} \cdot x_3 \cdot x_4 + x_1 \cdot x_2 \cdot \overline{x_3} \cdot x_4 + x_1 \cdot x_2 \cdot x_3 \cdot \overline{x_4}$$

Obr. 4.10 Uzávěrka diferenciálu – funkce ve tvaru DNF.

Již na první pohled je zřejmé, že tento tvar je poměrně nehospodárný, a proto bude vhodné provést minimalizaci. Opět můžeme využít názorného zápisu do Karnaughovy mapy.



Obr. 4.11 Uzávěrka diferenciálu – Karnaughova mapa.

Celkem lze vytvořit čtyři dvojice a minimalizací tak dostaneme následující tvar algebraické rovnice.

$$y_1 = \overline{x_1} \cdot x_2 \cdot x_4 + x_1 \cdot \overline{x_2} \cdot x_3 + x_1 \cdot \overline{x_3} \cdot x_4 + x_2 \cdot x_3 \cdot \overline{x_4}$$

Obr. 4.12 Uzávěrka diferenciálu – minimalizovaný tvar.

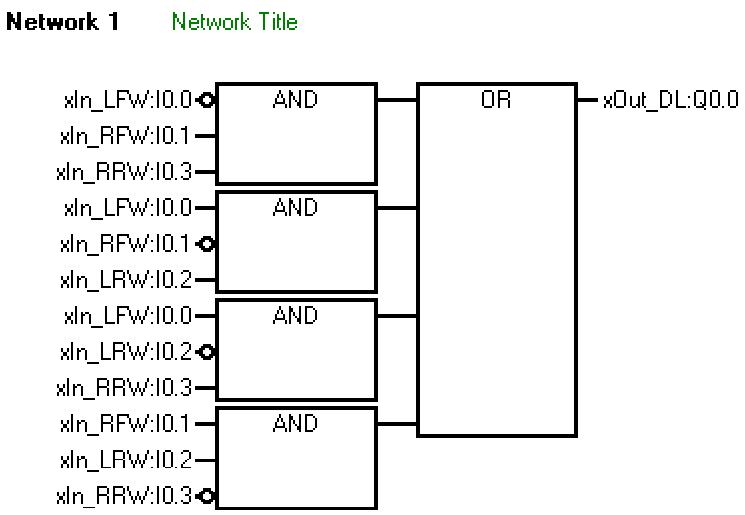
4.2.3 Realizace v prostředí STEP 7-Micro/WIN

Nejdříve je nutné v tabulce symbolů nadefinovat jednotlivé vstupy a výstupy. Prokluz je v našem případě vyvolán manuálně sepnutím jednoho či více vypínačů. Jejich logické hodnoty tedy reprezentují impulsy, které by vyslaly jednotlivé snímače. Postačí proto čtyři vstupy (jeden pro každé kolo) a jediný výstup, který nese informaci, zda uzamknout diferenciál.

Symbol	Address	Comment
xIn_LFW	I0.0	leve predni kolo (x1)
xIn_LRW	I0.2	leve zadni kolo (x3)
xIn_RFW	I0.1	prave predni kolo (x2)
xIn_RRW	I0.3	prave zadni kolo (x4)
xOut_DL	Q0.0	uzamceni diferencialu (y1)

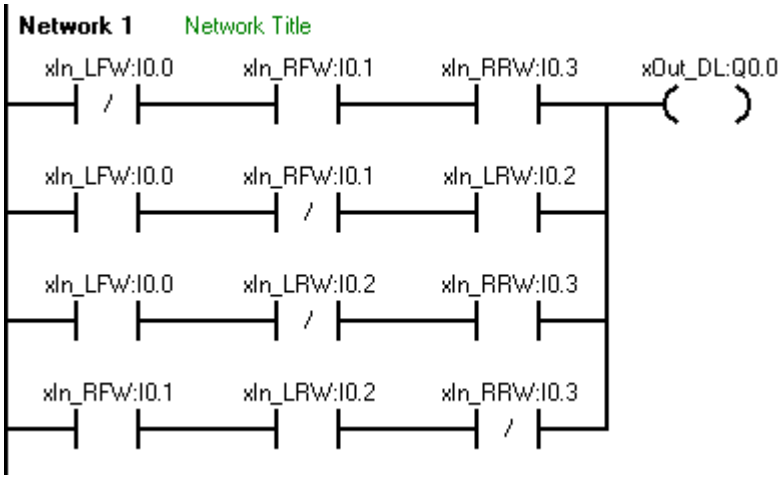
Obr. 4.13 Uzávěrka diferenciálu – tabulka symbolů.

Z minimalizovaného tvaru je vidět, že pro zapojení je třeba čtyř hradel typu AND a jednoho hradla typu OR. Při programování stačí postupovat dle získané algebraické rovnice. To znamená přivádět jednotlivé proměnné na vstupy hradel AND a jejich výstupy následně přivést na vstupy hradla OR. V samotném závěru pak stačí výsledek, který leží na vrcholu zásobníku, přiřadit vybranému výstupu.



Obr. 4.14 Uzávěrka diferenciálu – schéma FBD.

Při srovnání výsledku v podobě Ladder a FBD je vidět, že obě řešení jsou si v podstatě velmi podobná. V případě Ladder je logický součin reprezentován paralelním zapojením jednotlivých větví série logických součinů.



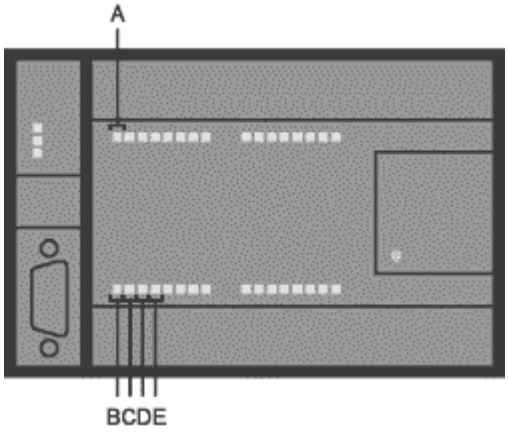
Obr. 4.15 Uzávěrka diferenciálu – schéma Ladder.

Také v zápisu pomocí STL je poměrně dobře vidět použití čtyř logických součinů a tří logických součtů, které jsou prováděny průběžně s ohledem na omezenou kapacitu zásobníku.

Nejdříve načteme proměnnou na vrchol zásobníku (`LDN`), provedeme logické součiny s dalšími proměnnými (`A`). Výsledek se opět uloží na vrchol zásobníku. V následujícím kroku načteme další proměnnou na vrchol zásobníku (`LD`), která se předřadí před původní. Poté se opakuje procedura logických součinů (`A, AN`). Následuje instrukce pro logický součet (`OLD`), která sečte dvě hodnoty na vrcholu zásobníku a výsledek logického součtu opět uloží na vrchol. Tento postup se opakuje ještě dvakrát. V závěru nakonec výstupu `xOut` přiřadíme hodnotu z vrcholu zásobníku.

```
Network 1
LDN  xIn_LFW   //přesun negované hodnoty vstupu xIn_LFW na vrchol
               zásobníku
A    xIn_RFW   //součin hodnoty na vrcholu zásobníku a xIn_RFW; uloží
               na vrchol zásobníku
```

```
A    xIn_RBW   //součin hodnoty na vrcholu zásobníku a xIn_RBW; uloží
               na vrchol zásobníku

LD   xIn_LFW   //přesun hodnoty vstupu xIn_LFW na vrchol zásobníku
AN   xIn_RFW   //součin hodnoty na vrcholu zásobníku a negované
               xIn_RFW; uloží na vrchol zásobníku
A    xIn_LBW   //součin hodnoty na vrcholu zásobníku a xIn_LBW; uloží
               na vrchol zásobníku

OLD            //provede logický součet prvních dvou položek v
               zásobníku

LD   xIn_LFW   //přesun hodnoty vstupu xIn_LFW na vrchol zásobníku
AN   xIn_LBW   //součin hodnoty na vrcholu zásobníku a negované
               xIn_LBW; uloží na vrchol zásobníku
A    xIn_RBW   //součin hodnoty na vrcholu zásobníku a xIn_RBW; uloží
               na vrchol zásobníku

OLD            //provede logický součet prvních dvou položek v
               zásobníku

LD   xIn_RFW   //přesun hodnoty vstupu xIn_RFW na vrchol zásobníku
A    xIn_LBW   //součin hodnoty na vrcholu zásobníku a xIn_LBW; uloží
               na vrchol zásobníku
AN   xIn_RBW   //součin hodnoty na vrcholu zásobníku a negované
               xIn_RBW; uloží na vrchol zásobníku

OLD            //provede logický součet prvních dvou položek v
               zásobníku

=    xOut_     //přiřadí výslednou hodnotu výstupu xOut_
```



A stav diferenciálu (svítí = uzamčen)
B levé přední kolo – LFW – (svítí = prokluzuje)
C pravé přední kolo – RFW – (svítí = prokluzuje)
D levé zadní kolo – LBW – (svítí = prokluzuje)
E pravé zadní kolo – RBW – (svítí = prokluzuje)

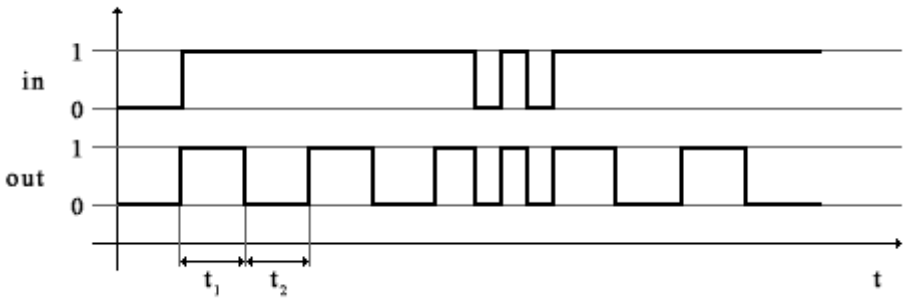
Obr. 4.16 Uzávěrka diferenciálu – vstupy a výstupy.

4.3 Clock Cascade

Tato úloha objasňuje funkci a použití časovačů a také speciálního paměťového bitu SM0.1, který je využíván při provádění inicializace (počáteční nastavení hodnot).

4.3.1 Zadání

Realizujte program popsaný časovým průběhem na obr. 4.17.



Obr. 4.17 Clock Cascade – finální diagram.

4.3.2 Řešení

Stavebním kamenem celé úlohy jsou dva časovače, které odměřují stanovené časové úseky t_1 a t_2. Impulsem pro spuštění časovače je doběh předchozího (kromě prvního cyklu, kdy je časovač spuštěn příchodem log 1 na vstup in). Úloha by šla zřejmě řešit i s použitím pouze jednoho časovače, který by se průběžně resetoval a poté volal střídavě s různými parametry (odpočítávaný čas). Ovšem s ohledem na přehlednost a dostatek prostředků není nutné k tomuto kroku přistupovat.

4.3.3 Realizace v prostředí STEP 7-Micro/WIN

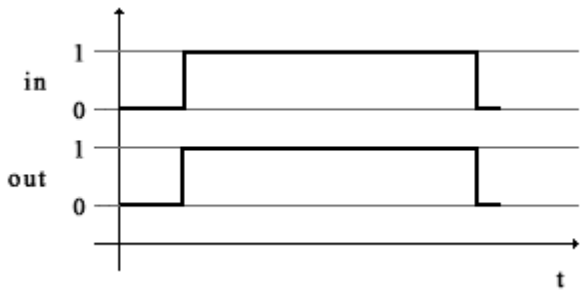
V tabulce symbolů postačí nadefinovat jeden vstup, kterým se bude aktivovat celý proces a jeden výstup, který bude indikovat aktuální stav výstupu.

Symbol	Address	Comment
xIn	I0.0	
xOut	Q0.0	

Obr. 4.18 Clock Cascade – tabulka symbolů.

Řešení této úlohy lze rozdělit do několika kroků, ve kterých se postupně propracujeme k požadované funkci.

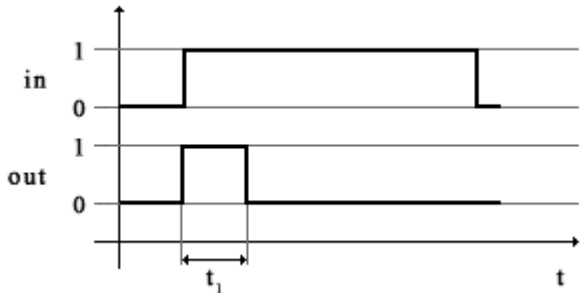
V první fázi výstup (`xOut`) přesně kopíruje stav vstupu (`xIn`). Pokud je na vstupu (`xIn`) logická jednička, bude i na výstupu (`xOut`) logická jednička.



Obr. 4.19 Clock Cascade – první fáze.

```
Network 1
LD      xIn   //přesun hodnoty vstupu xIn na vrchol zásobníku
=       xOut  //přiřazení výsledné hodnoty výstupu xOut
```

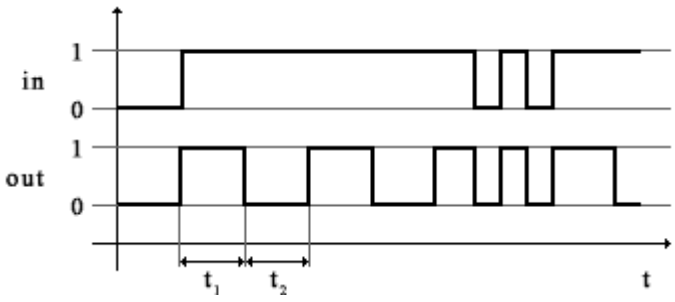
Následuje zařazení prvního časovače tak, aby se spustil s příchodem logické jedničky na vstup (`xIn`). Po dobu t_1 bude na výstupu (`xOut`) logická jednička. Jakmile čas vyprší, výstup (`xOut`) se nastaví na hodnotu logická nula. Je vhodné připomenout, že časovač je nutné před každým použitím vyresetovat.



Obr. 4.20 Clock Cascade – druhá fáze.

```
Network 1
R       T33, 1     //reset časovače T33

LD      xIn        //přesun hodnoty vstupu xIn na vrchol zásobníku
S       xOut       //set – přivedení log.1 na výstup xOut
TON     T33, 100   //spuštění časovače T33 s parametrem 1000ms

LD      T33        //testování uplynutí 1000ms (T33)
R       xOut, 1    //reset – přivedení log.0 na výstup xOut
```

Nyní zbývá přidat druhý časovač, který se spustí za účelem odměření doby t_2, po kterou bude hodnota výstupu (`xOut`) rovna logické nule. Impulsem pro jeho spuštění bude doběh prvního časovače, respektive resetování výstupu (`xOut`). Pro větší přehlednost při odlaďování je lepší volit časy t_1 a t_2 odlišné. Další změnou je umístění inicializační podmínky (`LD SM0.1`) do `Network 1`. Ta zajistí, že se následující příkazy provedou pouze při prvním programovém cyklu. V našem případě jsme provedli pouze počáteční reset časovačů.



Obr. 4.21 Clock Cascade – třetí fáze.

```
Network 1
LD        SM0.1     //inicializace při spuštění
R         T33, 1    //reset časovače T33
R         T34, 1    //reset časovače T34

Network 2
LD        xIn       //přesun hodnoty vstupu xIn na vrchol zásobníku
S         XOut, 1   //set – přivedení log.1 na výstup xOut

TON       T33, 200  //spuštění časovače T33 s parametrem 2000ms
```

```
LD        T33       //testování uplynutí 2000ms
R         xOut, 1   //reset – přivedení log.0 na výstup xOut

TON       T34, 100  //spuštění časovače T34 s parametrem 1000ms
LD        T34       //testování uplynutí 1000ms
S         xOut, 1   //set – přivedení log.1 na výstup xOut

R         T33, 1    //reset časovače T33
R         T34, 1    //reset časovače T34
```

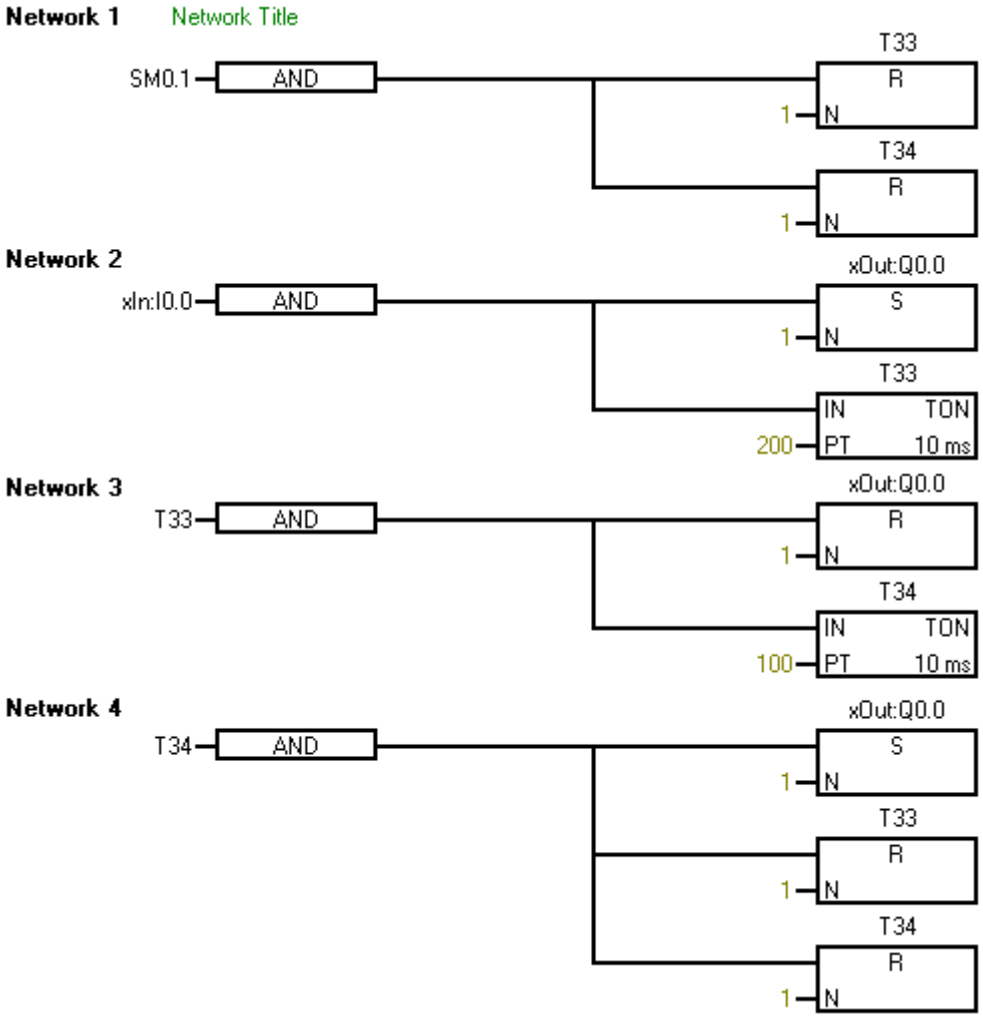
V zápisu pomocí jazyka funkčních bloků (FBD) a jazyka Ladder je nutné program rozdělit do čtyř oddílů:

`Network 1:` S využitím inicializační podmínky `SM0.1` jsou pouze v prvním cyklu vyresetovány oba používané časovače `T33` a `T34`.

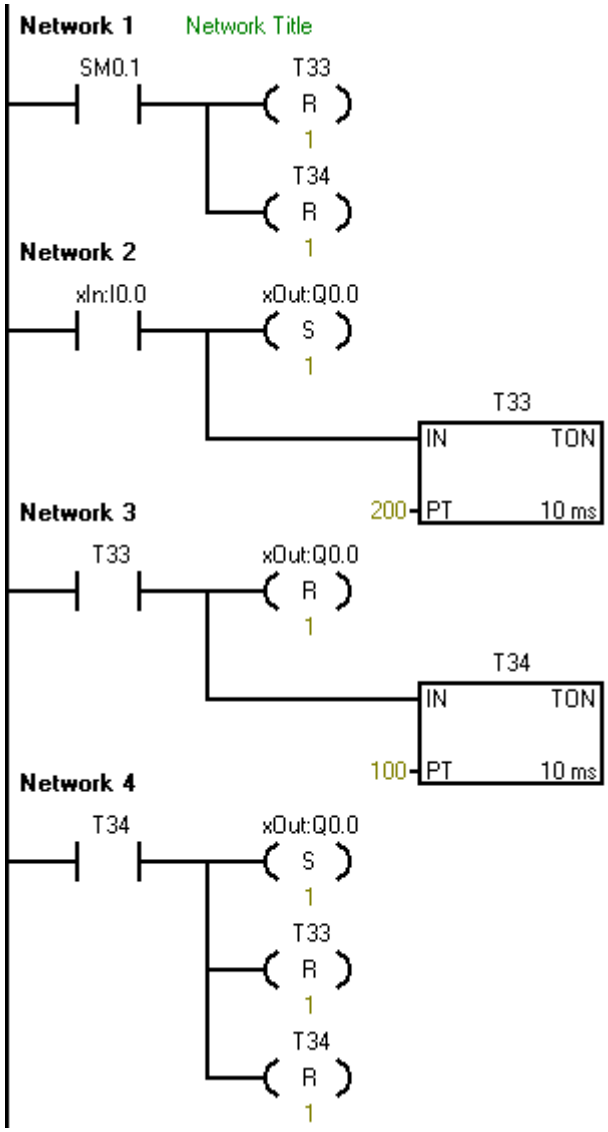
`Network 2:` Při splněné podmínce zapnutého vstupu `xIn` je spuštěn první časovač `T33` a zároveň sepnut výstup `xOut`.

`Network 3:` Jakmile je čas odměřen, výstup `xOut` se vyresetuje a je aktivován druhý z časovačů `T34`.

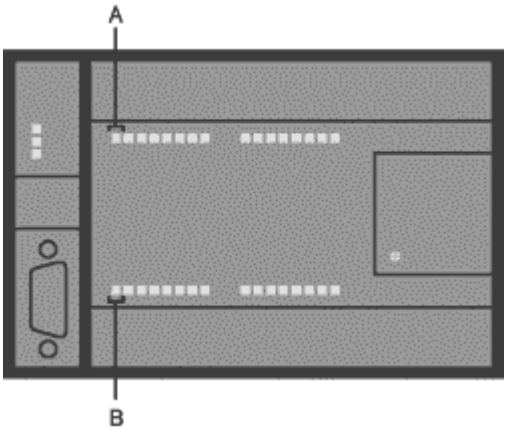
`Network 4:` Poté co doběhne i on, výstup `xOut` je znovu sepnut a oba časovače vyresetovány pro opětovné použití.



Obr. 4.22 Clock Cascade – jazyk FBD.



Obr. 4.23 Clock Cascade – jazyk Ladder.



A stav výstupu (svítí = log 1)

B spuštění cyklování

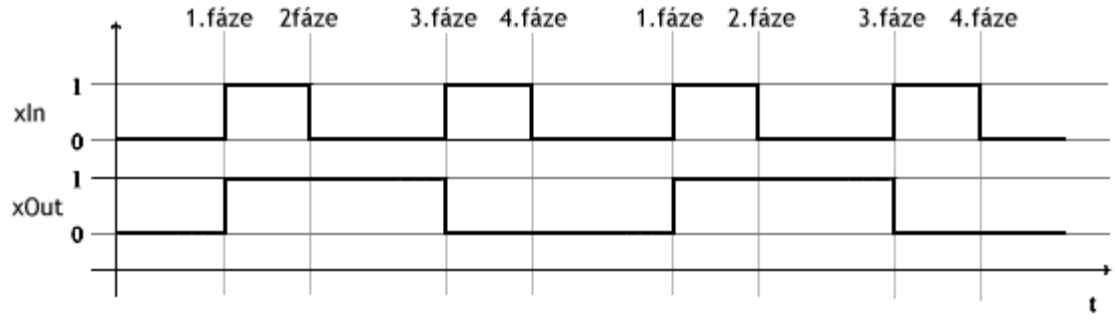
Obr. 4.24 Clock Cascade – vstupy a výstupy.

4.4 Flip-flop

Tato úloha demonstruje detekci vzestupných hran signálu a použití příkazů pro porovnávání a přiřazování hodnot do proměnných.

4.4.1 Zadání

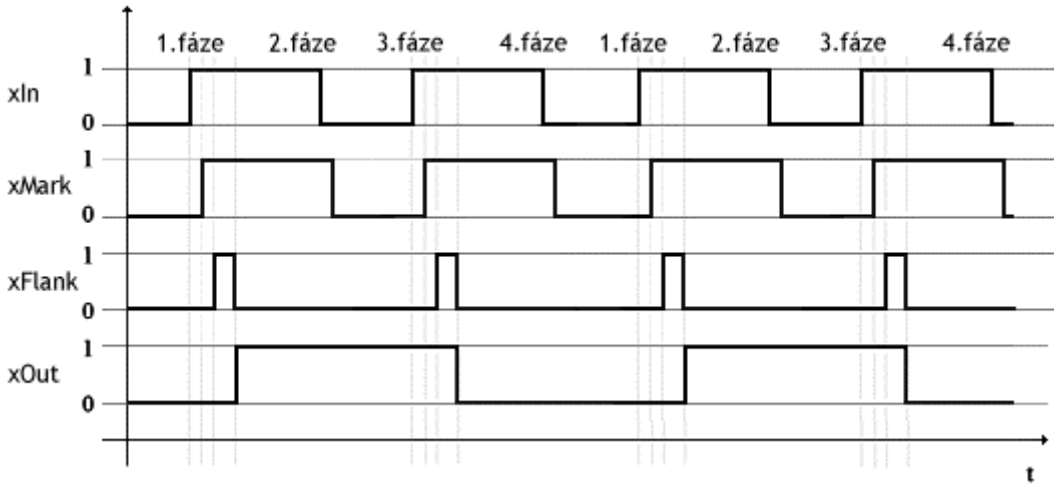
Realizujte program, který invertuje hodnotu výstupu, jestliže bude detekována vzestupná hrana signálu na vstupu.



Obr. 4.25 Flip-flop – základní diagram průběhu.

4.4.2 Řešení

Z předchozího obrázku je vidět, že výstup `xOut` reaguje pouze na přítomnost vzestupné hrany signálu na vstupu `xIn`. Při sestupné hraně na vstupu `xIn` se hodnota výstupu `xOut` nijak nemění.



Obr. 4.26 Flip-flop – rozšířený diagram průběhu.

Hlavním vodítkem v této úloze je tedy změna stavu vstupu `xIn`. V závislosti na tom, zda se jedná o vzestupnou či sestupnou hranu, změní se hodnota pomocných proměnných `xMark` a `xFlank`, které v kombinaci s dosavadní hodnotou určují následující stav `xOut`.

4.4.3 Realizace v prostředí STEP 7-Micro/WIN

Program byl postupně proveden ve dvou obdobných verzích „A“ a „B“. V první z nich jsou jako pomocná úložiště hodnot použity výstupy. Těch je však k dispozici poměrně nízký počet, a tudíž by při zpracování obsáhlejší úlohy mohly chybět. Druhá a mírně obtížnější varianta využívá proměnných, kterých je nespočetně více a jsou pro tento účel vhodnější. Jejich

použití si ovšem vyžádalo několik úprav ve struktuře schématu.

Prvního rozdílu si lze povšimnout při srovnání obou následujících tabulek symbolů. Jedná se o již zmíněné použití výstupů, respektive proměnných typu byte.

	Symbol	Address	Comment
1	xIn	I0.0	
2	xFlank	Q0.7	
3	xOut	Q0.0	
4	xMark	Q0.6	

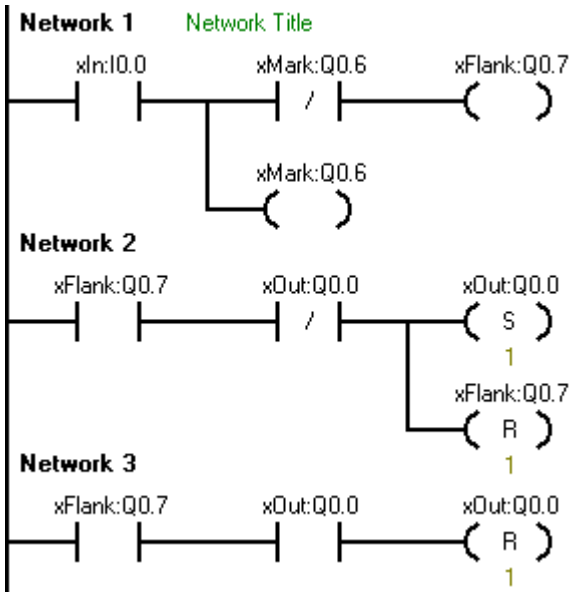
Obr. 4.27 Flip-flop – tabulka symbolů (varianta A).

	Symbol	Address	Comment
1	xIn	I0.0	
2	bFlank	VB0	
3	bOut	VB2	
4	bMark	VB4	

Obr. 4.28 Flip-flop – tabulka symbolů (varianta B).

Blok příkazů v Newtvork 1 slouží jako náhrada za instrukci EU a detekuje vzestupnou hranu signálu. Následně je za pomoci proměnných `xFlank` a `xMark` a v závislosti na minulé hodnotě změněn výstup `xOut`

V případě že se jedná o hranu sestupnou, nedojde k aktivaci pomocné proměnné `xFlank` a neprovede se ani zbytek kódu v částech Network 2 a Network 3, které zajišťují změnu hodnoty výstupu `xOut`.



Obr. 4.29 Flip-flop – jazyk Ladder (varianta A).

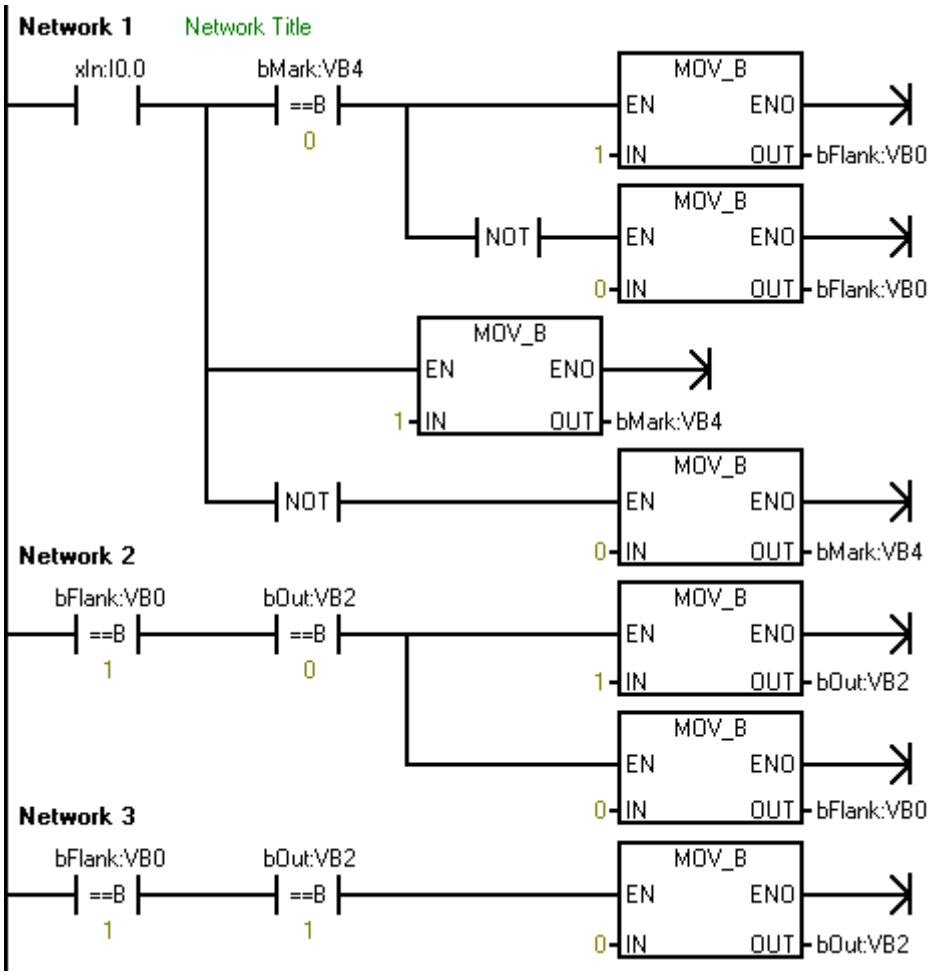
1.fáze - výsledkem Network 1 je log. 1 na výstupu `xFlank`
- výsledkem Network 2 je log. 1 na výstupu `xOut` a reset výstupu `xFlank`
- obsah Network 3 se neprovede, protože nejsou splněny obě podmínky
- závěr: hodnota výstupu `xOut` se změnila na log. 1

2.fáze - obsah Network 1 se neprovede, protože není splněna podmínka
- obsah Network 2 se neprovede, protože nejsou splněny obě podmínky
- obsah Network 3 se neprovede, protože nejsou splněny obě podmínky
- závěr: hodnota výstupu `xOut` zůstává rovna log. 1

3.fáze - výsledkem Network 1 je log. 1 na výstupu `xFlank`
- obsah Network 2 se neprovede, protože nejsou splněny obě podmínky
- výsledkem Network 3 je reset výstupu `xOut`
- závěr: hodnota výstupu `xOut` se změnila na log. 0

4.fáze - obsah Network 1 se neprovede, protože není splněna podmínka
- obsah Network 2 se neprovede, protože nejsou splněny obě podmínky
- obsah Network 3 se neprovede, protože nejsou splněny obě podmínky
- závěr: hodnota výstupu `xOut` zůstává rovna log. 0

Kromě tabulky symbolů je nejmarkantnějším rozdílem varianty B použití instrukcí pro přiřazovaní hodnot do proměnných (`MOV_B`), které nahrazují původních `Set` a `Reset`. Dále pak nutnost za pomoci příkazů `NOT` rozdělit nastavení proměnných tam, kde byly použity příkazy „výstup“ (Network 1).



Obr. 4.30 Flip-flop – jazyk Ladder (varianta B).

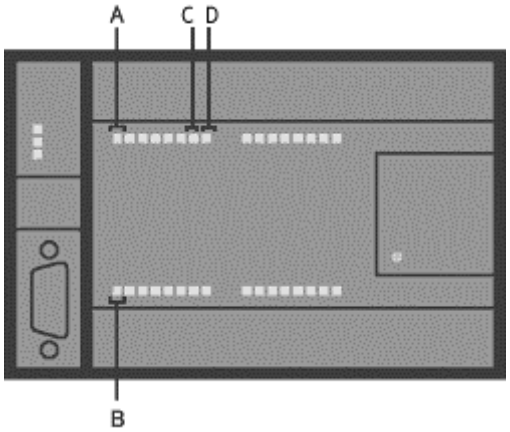
Z úsporných důvodů je dále uvedeno pouze řešení obou variant v jazyce STL, které se vygenerovalo přepnutím jazyka v popsaném v kapitole 3.2.1.

Varianta A:

```
Network 1
LD      xIn         //přesun hodnoty vstupu xIn na vrchol zásobníku
LPS                 //duplikuje horní hodnotu zásobníku
AN      xMark       //součin hodnoty na vrcholu zásobníku a negované
                    hodnoty xMark, uloží se na vrchol zásobníku
=       xFlank      //přiřazení výsledné hodnoty výstupu xFlank
LPP                 //přesunutí druhé hodnoty v zásobníku na jeho vrchol
=       xMark       //přiřazení výsledné hodnoty výstupu xMark

Network 2
LD      xFlank      //přesun hodnoty vstupu xFlank na vrchol zásobníku
AN      xOut        //součin hodnoty na vrcholu zásobníku a negované
                    hodnoty xOut, uloží se na vrchol zásobníku
S       xOut        //set – přivedení log.1 na výstup xOut
R       xFlank      //reset – přivedení log.0 na výstup xFlank

Network 3
LD      xFlank      //přesun hodnoty vstupu xFlank na vrchol zásobníku
A       xOut        //součin hodnoty na vrcholu zásobníku a hodnoty
                    xOut, uloží se na vrchol zásobníku
R       xOut        //reset – přivedení log.0 na výstup xOut
```

Varianta B:

```
Network 1
LD     xIn          //přesun hodnoty vstupu xIn na vrchol zásobníku
LPS                 //duplikuje horní hodnotu zásobníku
AB=    bMark, 0     //porovnání hodnoty proměnné bMark s 0 a log. Součin
                    s hodnotou na vrcholu zásobníku
MOVB   1, bFlank    //přiřazení hodnoty 1 proměnné bFlank
NOT                 //negace hodnoty na vrcholu zásobníku
MOVB   0, bFlank    //přiřazení hodnoty 0 proměnné bFlank
LPP                 //přesunutí druhé hodnoty v zásobníku na jeho vrchol
MOVB   1, bMark     //přiřazení hodnoty 1 proměnné bMark
NOT                 //negace hodnoty na vrcholu zásobníku
MOVB   0, bMark     //přiřazení hodnoty 0 proměnné bMark

Network 2
LDB=   bFlank, 1    //porovnání hodnoty proměnné bFlank s 1
AB=    bOut, 0      //porovnání hodnoty proměnné bOut s 0 a log. Součin
                    s hodnotou na vrcholu zásobníku
MOVB=  1, bOut      //přiřazení hodnoty 1 proměnné bOut
MOVB=  0, bFlank    //přiřazení hodnoty 0 proměnné bFlank

Network 3
LDB=   bFlank       //porovnání hodnoty proměnné bFlank s 1
AB=    bOut, 1      //porovnání hodnoty proměnné bOut s 1 a log. Součin
                    s hodnotou na vrcholu zásobníku
MOVB   0, bOut      //přiřazení hodnoty 0 proměnné bOut
```



A stav výstupu xOut (svítí = log 1) – pouze u varianty A

B stav vstupu xIn (svítí = log 1)

C stav výstupu `bMark` (svítí = log 1) – pouze u varianty A

D stav výstupu `bFlank` (svítí = log 1) – pouze u varianty A

Obr. 4.31 Flip-flop – vstupy a výstupy.

4.5 Obsluha výtahu čtyřpatrové budovy

Závěrečná úloha s cílem shrnout doposud probrané příkazy a postupy dále vysvětluje použití podprogramů a příkazů pro inkrementaci a dekrementaci proměnných.

4.5.1 Zadání

Realizujte program, který bude napodobovat chování výtahu ve čtyřpatrové budově. Do řešení zahrňte následující funkce:

- pojezd nahoru a dolů v rámci čtyř pater
- možnost změny zatížení
- detekci přetížení kabiny
- detekci průchodu dveřmi
- indikaci aktuálního patra
- indikaci otevřených dveří

4.5.2 Řešení

Zvolený počet čtyř obsluhovaných pater již dostatečně ilustruje možnosti a zároveň je stále uskutečnitelný s ohledem na počet použitých vstupů a výstupů. Samotný přejezd je realizován univerzálním způsobem, takže celý program by po drobných úpravách byl schopen obsluhovat i vyšší počet pater.

Změna zatížení kabiny je uskutečněna pomocí potenciometru na těle automatu, který umožňuje plynulé navýšení, či snížení. Druhou zvažovanou možností jak dosáhnout různého zatížení, je současným sepnutím kombinace několika vstupů. Těm by byly přiřazeny různé hodnoty, které by ve výsledku dávaly několik pevných úrovní naložení. S ohledem na eleganci řešení a možnost demonstrovat použití dalšího prvku, byla zvolena první varianta.

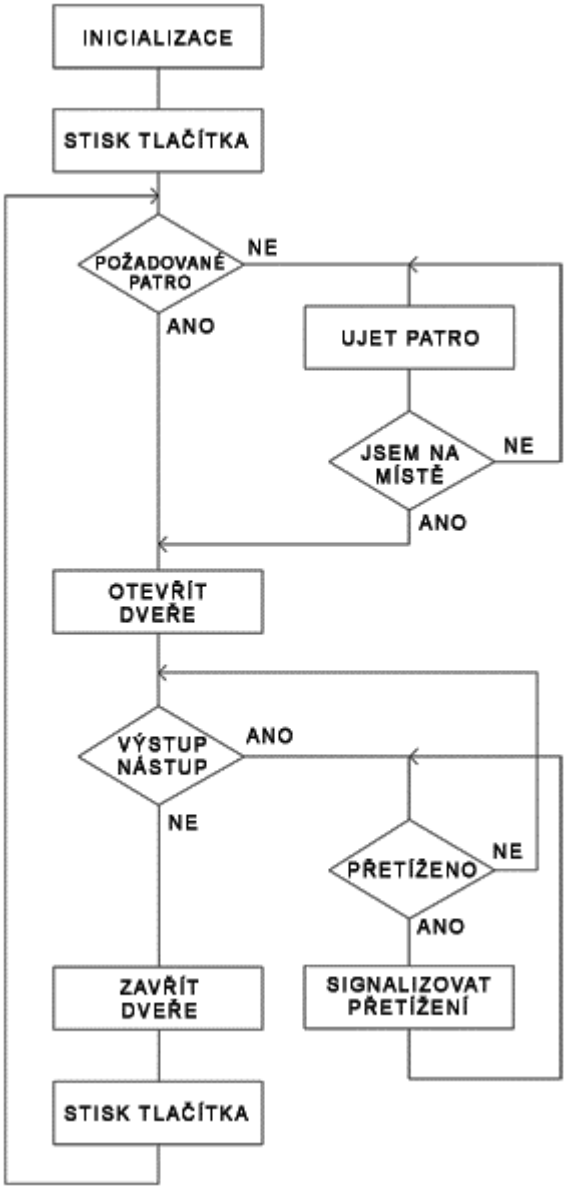
Detekci přetížení kabiny lze provést prostým porovnáním aktuální hodnoty zmíněného potenciometru s pevnou hranicí, kterou prohlásíme za maximální naložení. Jestliže bude překročena, znemožníme zavření dveří a signalizujeme stav například sepnutím vyhrazeného výstupu.

Detekce průchodu dveřmi má zajistit jejich znovuotevření na základě impulsu od snímače ve dveřním prostoru. Pro napodobení tohoto procesu je použit jeden ze vstupů. Jeho sepnutím vyvoláme pomyslný průchod a zabráníme tak dovření dveří, respektive znovu spustíme odpočet času do jejich zavření.

Patro, ve kterém se aktuálně nachází kabina výtahu, je indikováno čtyřmi výstupy, a to na základě porovnání proměnné s údajem o pozici výtahu a číslem (označením) patra.

Pro signalizaci otevřených dveří je možné opět použít jeden výstup.

Jelikož se jedná o rozsáhlejší úlohu, je vhodné postupovat systematicky po jednotlivých krocích. Pro tento účel může posloužit například vývojový diagram, který by mohl vypadat následovně:



Obr. 4.32 Vývojový diagram.

4.5.3 Realizace v prostředí STEP 7-Micro/WIN

Vzhledem k prostorové náročnosti bude uvedeno pouze řešení v jazyce STL.

Pro základní funkce pojezdu je potřeba zavést proměnnou, nesoucí informaci o aktuální pozici výtahu a další s údajem o požadovaném (cílovém) patře. Jelikož se jedná o výtah obsluhující čtyři patra, budeme potřebovat čtyři vstupy pro volání výtahu a čtyři výstupy pro indikaci pozice kabiny. Signalizace přetížení a otevřených dveří si vyžádá další dva výstupy. Pro vyvolání procesu nastupování či vystupování bude potřeba ještě jednoho vstupu. Možnost změny zatížení je možné realizovat například jedním z integrovaných potenciometrů.

			Symbol	Address	Comment
1			xIn_Floor1	I0.0	volání výtahu - první patro
2			xIn_Floor2	I0.1	volání výtahu - druhé patro
3			xIn_Floor3	I0.2	volání výtahu - třetí patro
4			xIn_Floor4	I0.3	volání výtahu - čtvrté patro
5			xIn_Move	I1.1	vyvolání nástupu / výstupu
6			xOut_Floor1	Q0.0	indikace přítomnosti kabiny - první patro
7			xOut_Floor2	Q0.1	indikace přítomnosti kabiny - druhé patro
8			xOut_Floor3	Q0.2	indikace přítomnosti kabiny - třetí patro
9			xOut_Floor4	Q0.3	indikace přítomnosti kabiny - čtvrté patro
10			xDoors	Q1.0	1 = dveře otevřené
11			xOverload	Q1.1	1 = kabina přetížena
12			bCurrent_floor	VB0	aktuální patro
13			bTarget_floor	VB2	následující patro
14			bLoading	SMB28	potenciometr změny zatížení

Obr. 4.33 Obsluha výtahu – tabulka symbolů.

hlavní blok programu:

- Vytvoříme inicializační blok. K tomu využijeme již zmíněný paměťový bit SM0.1. Ten zajistí, že následující příkazy se provedou pouze jedenkrát. Chceme, aby po spuštění byl výtah v prvním patře a otevřely se dveře. To znamená, že bude nutné nastavit proměnným `bCurrent_floor` a `bTarget_floor` hodnotu 1. Všechny používané časovače T32, T33 a T34 je vhodné preventivně vyresetovat.
- Zajistíme přiřazení odpovídající hodnoty do proměnné `bTarget_floor` v závislosti na stisknutém tlačítku (zvoleném patře).
- Porovnáme proměnné `bCurrent_floor` a `bTarget_floor`. V závislosti na výsledku nastavíme volání odpovídajícího podprogramu pro pohyb výtahu nahoru, dolů nebo otevření dveří.

```
Network 1
LD      SM0.1                          //inicializační special marker
MOVB    1, bCurrent_floor              //přiřazení 1 proměnné bKde
MOVB    1, bTarget_floor               //přiřazení 1 proměnné bKam

R       T32, 1                         //reset časovačů T32, T33, T34
R       T33, 1
R       T34, 1

Network 2
LD      xIn_Floor1                     //první spínač -> bKam = 1
MOVB    1, bTarget_floor

LD      xIn_Floor2                     //druhý spínač -> bKam = 2
MOVB    2, bTarget_floor

LD      xIn_Floor3                     //třetí spínač -> bKam = 3
MOVB    3, bTarget_floor

LD      xIn_Floor4                     //čtvrtý spínač -> bKam = 4
MOVB    4, bTarget_floor

Network 3
LDB=    bTarget_floor, bCurrent_floor  //zavolán ze stejného patra
CALL    SBR_0                          //volat podprogram pro otevření
                                       dveří
```

```
LDB>    bTarget_floor, bCurrent_floor //zavolán z vyššího patra
CALL    SBR_1                         //volat podprogram pro pohyb
                                      nahoru

LDB<    bTarget_floor, bCurrent_floor //zavolán z nižšího patra
CALL    SBR_2                         //volat podprogram pro pohyb
                                      dolů
```

podprogram pro otevření dveří:

- Zavoláme podprogram pro detekci aktuálního patra.
- Pokud jsou dveře kabiny zavřené (tzn. proměnná `xDoors` = 0), spustíme časovač za účelem odměření doby před otevřením dveří.
- Jakmile uběhne zmíněný okamžik, dáme pokyn pro otevření dveří.
- Vyresetujeme časovače T32 a T34.

```
Network 1
CALL    SBR_3           //volání podprogramu pro detekci aktuálního
                        patra

LDN     xDoors          //pokud jsou dveře zavřené
TON     T32, 2000       //spustit časovač (2s)

LD      T32             //pokud časovač doběhl
S       xDoors, 1       //tak otevřít dveře

R       T34, 1          //reset časovačů T32, T34
R       T32, 1
```

podprogram pro pohyb nahoru:

- Zavoláme podprogram pro nástup a detekci přetížení kabiny.
- Pokud jsou dveře kabiny zavřené, spustíme časovač, který bude měřit čas na přejezd mezi patry.
- Po uběhnutí času zvýšíme hodnotu proměnné `bCurrent_floor` o 1.
- Zavoláme podprogram pro detekci aktuálního patra.

```
Network 1
CALL    SBR_4           //volání podprogramu pro nástup a detekci
                        přetížení

LDN     xDoors          //pokud jsou dveře zavřené
TON     T33, 200        //spustit časovač (2s - doba na přejezd mezi
                        patry)

LD      T33             //pokud časovač doběhl
R       T33, 1          //reset časovače T33
INCB    bCurrent_floor  //přejet o patro nahoru
CALL    SBR_3           //volání podprogramu pro detekci aktuálního
                        patra
```

podprogram pro pohyb dolů:

- Shodný jako pro pohyb nahoru. Jediným rozdílem je snižování hodnoty proměnné `bCurrent_floor`.

```
Network 1
CALL    SBR_4           //volání podprogramu pro nástup a detekci
                        přetížení
```

```
LDN     xDoors          //pokud jsou dveře zavřené
TON     T33, 200        //spustit časovač (2s - doba na přejezd mezi
                        patry)

LD      T33             //pokud časovač doběhl
R       T33, 1          //reset časovače T33
DECB    bCurrent_floor  //přejet o patro dolů
CALL    SBR_3           //volání podprogramu pro detekci aktuálního
                        patra
```

podprogram detekce aktuálního patra:

- Postupně porovnáváme hodnoty proměnné `bCurrent_floor` s čísly jednotlivých pater (1 až 4) a nastavíme výstup odpovídajícího patra a resetujeme výstupy ostatní.

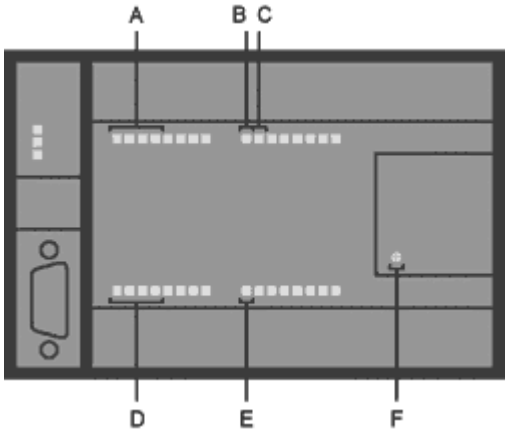
```
Network 1
LDB=    bCurrent_floor, 1   //pokud aktuální patro odpovídá 1

S       xOut_Floor1, 1      //tak signalizovat polohu v prvním patře
R       xOut_Floor2, 1
R       xOut_Floor3, 1
R       xOut_Floor4, 1

LDB=    bCurrent_floor, 2   //pokud aktuální patro odpovídá 2

R       xOut_Floor1, 1
S       xOut_Floor2, 1      //tak signalizovat polohu ve druhém patře
R       xOut_Floor3, 1
R       xOut_Floor4, 1

LDB=    bCurrent_floor, 3   //pokud aktuální patro odpovídá 3

R       xOut_Floor1, 1
R       xOut_Floor2, 1
S       xOut_Floor3, 1      //tak signalizovat polohu ve třetím patře
R       xOut_Floor4, 1

LDB=    bCurrent_floor, 4   //pokud aktuální patro odpovídá 4

R       xOut_Floor1, 1
R       xOut_Floor2, 1
R       xOut_Floor3, 1
S       xOut_Floor4, 1      //tak signalizovat polohu ve čtvrtém patře
```

podprogram pro nástup a detekci přetížení:

- Jestliže jsou dveře kabiny otevřené, spustíme časovač, který odměří dobu než se pokusí zavřít dveře.
- Pokud během této doby snímač ve dveřích zachytí pohyb, vyresetujeme časovač a ten začne odměřovat znova.
- Porovnáme aktuální hodnotu potenciometru (s rozsahem 0 až 255), který nese informaci o zatížení kabiny se zvolenou hodnotou reprezentující maximální zatížení (např. 100). Pakliže je zatížení vyšší než povolená mez, signalizujeme přetížení za pomoci zvoleného výstupu.
- V případě, že časovač doběhl a kabina není přetížena, dáme pokyn pro zavření dveří.

```
Network 1
LD      xDoors          //pokud jsou dveře otevřené

TON     T34, 500        //spustit časovač
```

```
LD      xIn_Move        //pokud snímač zachytí pohyb (nástup/výstup)
S       xDoors, 1       //tak otevřít dveře
R       T34, 1          //a resetovat časovač

LDB<=   bLoading, 100   //pokud je hodnota potenciometru (zatížení) <=
                        100
R       xOverload, 1    //tak nesignalizovat přetížení

LDB>    bloading, 100   //pokud je hodnota potenciometru (zatížení) >
                        100
S       xOverload, 1    //tak signalizovat přetížení

LD      T34             //pokud časovač doběhl
AN      xOverload       //a výtah není přetížen
R       xDoors, 1       //tak zavřít dveře
```



A aktuální patro (1. - 4.patro)
B stav dveří (svítí = otevřené)
C signalizace přetížení kabiny (svítí = přetížena)
D volání výtahu (1. - 4.patro)
E vyvolání nástupu/výstupu
F potenciometr změny zatížení kabiny

Obr. 4.34 Obsluha výtahu – vstupy a výstupy.

5 ZÁVĚR

Hlavním cílem této práce bylo seznámit se s programovatelným automatem Simatic S7-200 CPU 224XP a vývojovým prostředím STEP 7-Micro/WIN. Provést popis tohoto prostředí tak, aby byl vytvořený materiál použitelný pro výukové účely. Následně připravit několik vzorových úloh.

Úvodní řádky poskytují základní informace o významu a použití programovatelných automatů a také stručný popis jednotlivých konstrukčních variant spolu s jejich typickými představiteli.

Druhá kapitola se věnuje pouze charakteristice zmíněného Simatic S7-200 CPU 224XP. Kromě technických parametrů samotného automatu jsou zde popsány i konkrétní dostupné moduly, které poměrně značně rozšiřují teoretické pole nasazení. Popis zevnějšku, rozmístění hlavních ovládacích prvků a provozních režimů, by měl pomoci usnadnit manipulaci s automatem a základní úkony při jeho použití. To vše uzavírá přehled praktických aplikací v tuzemských firmách.

K efektivní práci v jakémkoliv vývojovém prostředí je dobré mít alespoň základní přehled o jeho možnostech. Tomu se věnuje třetí kapitola, která má za cíl urychlit prvotní orientaci v editoru STEP 7-Micro/WIN a to prostřednictvím popisu jednotlivých panelů a nabídek (printscreen pracovní obrazovky s popisky je umístěn v příloze). Najdeme zde přehled a význam nejdůležitějších a nejčastěji používaných příkazů, vybraných z instrukční sady automatu a také proces připojení a konfigurace automatu.

Poslední kapitola obsahuje celkem pět praktických úloh s různou obtížností, které vznikly za účelem ověření a procvičení teoretických postupů. Každá z nich svým způsobem demonstruje použití některých metod a příkazů z třetí kapitoly. Jednotlivé úkoly pojí společná šablona skládající se z několika kroků: od zadání, přes teoretické řešení až k samotnému naprogramování v prostředí STEP 7-Micro/WIN. Pro umocnění názornosti je vždy přiloženo schéma automatu s významem použitých vstupů a výstupů. Okomentované zdrojové kódy se nacházejí na přiloženém nosiči.

Úvodní úloha Realizace logické funkce je zaměřena na ověření elementárních postupů. Funkce je zadána pravdivostní tabulkou, kombinaci hodnot na vstupech tedy odpovídá hodnota výstupu.

Cílem následující úlohy Uzávěrka diferenciálu je vyslat impuls pro uzamčení nápravového diferenciálu a to v závislosti na počtu protáčejících se kol. Z obecného zadání je nejdříve vytvořen seznam všech možných kombinací prokluzu a následné reakce, které lze v dalším kroku vyjádřit pravdivostní tabulkou. Ta je následně minimalizována pomocí Karnaughovy mapy a výsledný tvar realizován v prostředí STEP 7-Micro/WIN.

Třetí úloha Clock cascade je postavena především na aplikaci dvou časovačů s různými časovými základnami. Mimo jiné je vysvětleno využití speciálního paměťového bitu, který umožňuje provést počáteční nastavení proměnných. Samotné řešení je rozděleno do několika kroků s využitím názorných diagramů.

Výsledkem čtvrtého úkolu s názvem Obvod typu flip-flop je program, invertující hodnotu výstupu při detekování vzestupné hrany signálu na vstupu. Postup, rozčleněný do čtyř fází, je naznačen prostřednictvím tabulky, diagramu a slovního popisu.

Závěrečná úloha s cílem shrnout doposud probrané příkazy napodobuje chování výtahu ve čtyřpatrové budově. Kromě samotného pojezdu je zahrnuta funkce plynulé změny zatížení kabiny za pomoci potenciometru, detekce přetížení s následným zablokováním pohybu, indikace aktuálního patra prostřednictvím diod a signalizace otevřených dveří.

SEZNAM POUŽITÉ LITERATURY

[1] SIEMENS. *Image Database* [online]. [cit. 16.května 2008]. Dostupné z: <https://www.automation.siemens.com/bilddb/guiwelcome.asp?lang=en>
[2] SIEMENS . *Micro Automation SIMATIC S7-200* [online]. [cit. 28. dubna 2008]. Dostupné z: <http://www.automation.siemens.com/_en/s7-200/index.htm>.
[3] SIMPO - Computer & Control Systéms. *Reference* [online]. 2007, [cit. 15.dubna 2008]. Dostupné z: <http://www.simpo.cz/automatizace.php?category=reference>.
[4] ATE spol. s.r.o. *Reference* [online]. [cit. 15.dubna 2008]. Dostupné z: <http://www.ate.cz/reference.html>.
[5] HLINOVSKÝ, Martin. *Příklady použití programovatelného automatu Simatic S7-200*. Automa [online]. 2004, č. 01 [cit 15. dubna 2008]. Dostupné z: <http://www.odbornecasopisy.cz/index.php?id_document=32151>.
[6] ŠTOHL, Radek. *Možnosti jediného kabelu*. Automa [online]. 2006, č. 07 [cit 15. dubna 2008]. Dostupné z: <http://www.odbornecasopisy.cz/index.php?id_document=31233>.
[7] MICROCOMP s.r.o. *Reference*. [online] [cit. 15.dubna 2008]. Dostupné z: <http://microcomp.sro.cz/ref/pneu.html>.
[8] VOJÁČEK, Antonín. *Co se skrývá pod označením PLC?*. [online]. 6. srpen 2006 [cit 6. dubna 2008]. Dostupné z: <http://automatizace.hw.cz/co-se-skryva-pod-oznacenim-plc>.
[9] SIMATIC. *Programovatelný automat S7-200*. Systémový manuál. [dokument pdf]. Siemens. 26. červenec 2004 [cit 18. dubna 2008]. Dostupné z: <http://www1.siemens.cz/ad/current/content/data_files/automatizacni_systemy/mikrosystemy/simatic_s7200/manual_s7_200_2004_cz.pdf>.
[9] *Diferenciál (mechanika)*. [online]. poslední úprava 9. května 2008 [cit 3.4.2008]. Dostupné z: <http://cs.wikipedia.org/wiki/Diferenciál_(mechanika)>.

SEZNAM PŘÍLOH

Příloha č. 1 – Instalace STEP 7-Micro/WIN
Příloha č. 2 – Popis prostředí STEP 7-Micro/WIN
Příloha č. 3 – Přiložené medium CD-R obsahující:

- Tento dokument v elektronické podobě
- Zdrojové kódy jednotlivých řešených úloh