HIL simulátor školících úloh na PLC Mitsubishi

Bakalářská práce

Vedoucí práce: Ing. Karel Kubíček
Konzultant: Ing. Jan Opálka
Vypracoval: Marek Siřiště

Plzeň, 2022

Titulní list

Zadání

- **1.** Seznamte se s problematikou vývoje SW pro PLC a jejich stavbou.
- **2.** Prostudujte si pojmy MIL, SIL, PIL a HIL. Co znamenají, jaký mají význam a k čemu se používají? Zasaďte je do procesu vývoje SW.
- **3.** Seznamte se s nástroji PLC Mitsubishi a HMI Mitsubishi a příslušnými vývojovými prostředími GX Works 2 a GX Works 3 pro PLC a GT Designer 2 a GT Designer 3 pro HMI.
- **4.** Vytvořte simulace tří školících úloh pro výuku programování PLC Mitsubishi. Simulace budou naprogramovány ve vývojovém prostředí GX Works 2 nebo GX Works 3. Simulace úloh bude vytvořena ve smyslu Hardware-in-the-loop (HIL).
- **5.** Ke všem úlohám vytvořte vizualizaci na panelu HMI v prostředí GT Designer 2 nebo GT Designer 3.
- **6.** Všechny úlohy zdokumentujte a vytvořte k nim manuál a metodické pokyny pro účely školení. Proveďte diskuzi, ve které slovně zhodnotíte přínos Vámi zpracovaných příkladů a problémů, se kterými jste se při řešení setkal.
- **7.** Analyzujte možnost modularity simulátoru, aby bylo možné z modelů jednotlivých menších technologických celků skládat modely větších technologických celků.

Assignment

- **1.** Get to know the issue of SW development for PLCs and their construction.
- **2.** Study the terms MIL, SIL, PIL and HIL. What do they mean, their meaning and what are they used for? Set them in the process of developing SW.
- **3.** Meet the PLC Mitsubishi and HMI Mitsubishi tools and relevant development environments of GX Works 2 and GX Works 3 for PLC and GT Designer 2 and GT Designer 3 for HMI.
- **4.** Create simulations of three training tasks to teach PLC Mitsubishi programming. The simulations will be programmed in the development environment of GX Works 2 or GX Works 3. Simulation of tasks will be created as Hardware-in-the-loop (HIL).
- **5.** For all tasks, create a visualization on the HMI panel in GT Designer 2 or GT Designer 3.
- **6.** Document all tasks and create a manual and guidance for training purposes. Have a discussion in which you evaluate the benefits of the examples and problems you have encountered in solving.
- **7.** Analyse the possibility of a simulator modularity so that models of larger technological units can be composed from models of individual smaller technological units.

Čestné prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne .....

Abstrakt

Cílem této práce je tvorba školících úloh pro programování programovatelných logických automatů (PLC). Práce obsahuje obecný popis PLC společně se seznámením s nástroji pro tvorbu programů a vizualizace. Praktická část práce je věnována tvorbě modelů, které byly navrženy ve smyslu Hardware in the loop (HIL). Hlavním cílem práce bylo sestavení modelu linky z dílčích navržených modelů dopravníku a pístu. Tento model slouží jako simulace reálného systému v rámci HIL simulátoru. Na tomto simulátoru lze navrhnout a ověřit řízení bez nutnosti připojení reálného fyzického zařízení. Jedna školící úloha byla věnována simulaci chování regulační smyčky. Pro každou školící úlohu byla vytvořena příslušná vizualizační část.

Klíčová slova: FBD, GT Designer3, GX Works2, GX Works3, Hardware in the loop, HMI, Model Based Design, PLC

Abstract

The primary goal of this work is to design training tasks for programming programmable logic controllers (PLC). The thesis contains a general description of PLC with an introduction to the tools for creating programs and visualization. The practical part of the work is devoted to the creation of models that have been designed as Hardware in the loop (HIL). The main goal of this thesis was to build a model of the line from partially designed models of conveyor and piston. This model serves as a simulation of the real system within the HIL simulator. On this simulator, the control system can be designed and verified without the need to connect a real physical device. One training task was devoted to simulating the behaviour of the control loop. An appropriate visualization section has been created for each training task.

Key words: FBD, GT Designer3, GX Works2, GX Works3, Hardware in the loop, HMI, Model Based Design, PLC

Poděkování

V první řadě bych rád poděkoval vedoucímu bakalářské práce Ing. Karlu Kubíčkovi za veškerou pomoc ohledně práce. Především za jeho odborné rady, které mi pomohly dovést bakalářskou práci do finální podoby a taktéž za jeho veškerý čas, který mi věnoval. Dále bych chtěl poděkovat konzultantovi Ing. Janu Opálkovi za odborné konzultace, které mi pomohly hlouběji pochopit principy programování PLC a také za pomoc při sestavování modelů pro praktickou část práce. V neposlední řadě bych rád poděkoval rodině za veškerou podporu v průběhu studia.

Obsah

1 Úvod

1.1 Obecný úvod

Tato bakalářská práce se zabývá tvorbou softwaru (SW) pro účely školení na PLC typu Mitsubishi. Bakalářská práce vznikla ve spolupráci s firmou **ATS aplikované technické systémy s.r.o**, která je pobočkou firmy **ATS Global**.

Společnost **ATS Global** byla založena v roce 1986 dvěma studenty v Holandsku. Dnes má tato firma zastoupení po celém světě, především v odvětví automatizace a řízení. Společnost se zabývá návrhy a kompletací velkých komplexních linek, ale zároveň i výrobou jednoúčelových strojů a zařízení. Mezi další předměty práce této společnosti patří i odborné školení a semináře, což je hlavním důvodem vzniku této práce.

Cílem práce je sestavit školící úlohy, na kterých si může začínající programátor PLC vyzkoušet základní prvky programování a osvojit si tak standardy při návrhu řízení systémů. Obrovskou výhodou simulace samotného řízeného systému v PLC je možnost plnohodnotného návrhu řízení s absencí fyzického systému, tím se školící stanice stane lépe přenositelná s možnostmi různých modulací a adaptací systému dle požadované potřeby školení.

Úlohy mohou sloužit též jako inspirace při návrhu řízení složitějších systému. V praxi se často programátor setkává s problémem absencí reálného systému při sestavování řízení. Linka, kterou je potřebné řídit, nemusí být v době vyhrazené pro programování řídícího systému sestavena, a tak má návrhář omezené možnosti návrhu. Nejjednodušším řešením je práce s určitou abstrakcí systému, čímž je možné včas odhalit chyby, které by mohly v průběhu implementace programu nastat. Práce se simulovaným systémem je tedy velice přínosná i z časového a ekonomického hlediska.

Teoretická část práce se věnuje popisu PLC, vysvětlení základních pojmů programování a seznámení s jednotlivými jazyky, se kterými je možné se v praxi setkat. Dále je popsán způsob návrhu Model-Based Design (MBD), zejména jeho jednotlivé části Software in the loop (SIL), Processor in the loop (PIL) a Hardware in the loop (HIL). A na konec tato část obsahuje seznámení s nástroji a popis práce v PLC typu Mitsubishi, především s nástroji **GX Works3** a **GT Designer 3**.

Praktická část práce spočívá ve třech školících úlohách v programovém prostředí **GX Works3**. Cílem první úlohy bylo vytvoření detailního modelu pístu, který se následně využije v druhé hlavní úloze. Píst je navržen tak, že je schopen simulovat problémy, které by mohly v praxi nastat (zaseknutí, nefunkčnost čidla) a zároveň je možné píst různě modulovat (jednočinný/dvojčinný píst, nastavení délky, rychlosti, stanovení polohy čidel) a upravovat, aby mohl být adaptován na jakýkoliv typ implementace. Pro demonstraci funkce pístu a představu jeho ovládání je následně sestavena jednoduchá sekvence řízení tří pístů. Pro potřebu další úlohy byla vytvořena ještě navíc simulace dopravníku. Výsledkem druhé úlohy je linka vytvořená na míru dle předlohy reálného modelu. Linka je sestavena z předešlých modelů, což ilustruje jistou část modularity a znovupoužitelnost již vytvořených celků. Poslední úloha je věnována regulaci jednoduchého systému druhého řádu. Systém i řízení může uživatel libovolně modifikovat a pochopit tak základy

jednoduché regulační smyčky. Součástí modelu je i simulace vstupních i výstupních chyb a jejich dopad na celkové chování systému.

Ke všem výše zmíněným úlohám je vytvořena vizualizace v programovém prostředí **GT Designer 3**, která napomáhá lepší představě o chování systému. Uživatel tak přímo může vidět změny chování systému na základě navrženého řízení.

1.2 Motivace

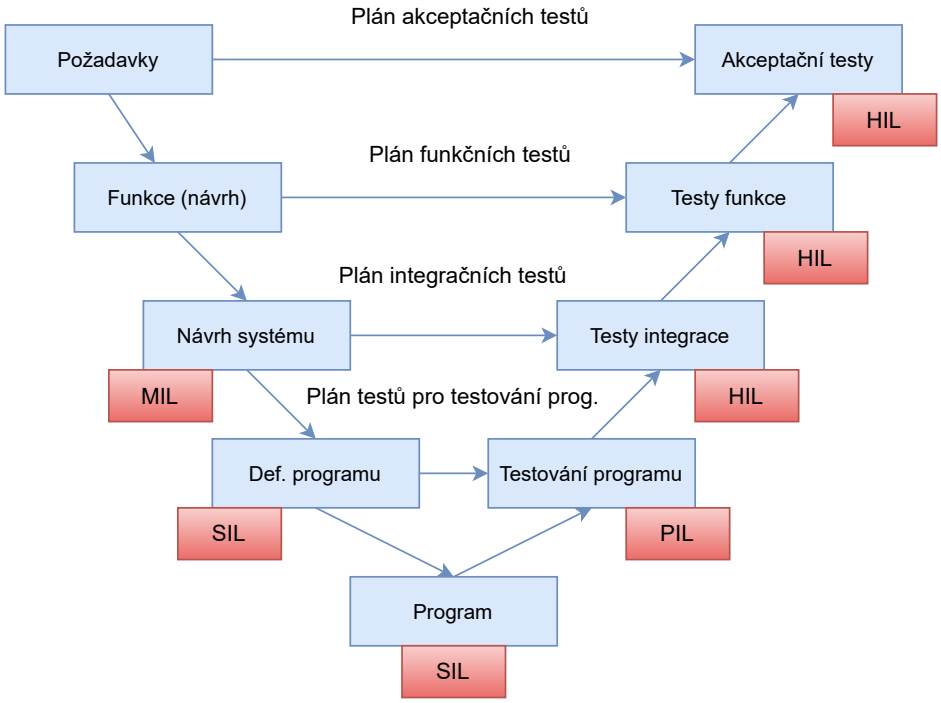
Hlavním tématem této bakalářské práce je simulace **HIL**, tedy simulace a testování modelu na zařízení PLC. Tento způsob návrhu vychází z návrhové metody **Model-Based Design**. Ke každému simulovanému modelu je vytvořen odpovídající řídící systém, jež má za úkol demonstrovat funkci jednotlivých školících úloh.

Vlivem rozvoje výpočetní techniky v uplynulých letech se v praxi mnohem častěji uplatňuje automatizace jednotlivých funkčních celků a tím dochází k optimalizaci výroby. Díky dekompozici je možné popsat složitý systém pomocí dílčích částí. Pro popis dílčích částí se hojně využívají funkční blokové diagramy (FBD) [1]. V tomto ohledu přináší velkou výhodu metoda návrhu **Model-Based Design**, která umožňuje rychle a efektivně navrhovat daný systém. Tento vývoj bývá doplněn o několik přístupů testování. První z těchto přístupů **Model in the loop** (MIL) spočívá v modelaci samotného systému ve vhodném modelovacím nástroji (Matlab/Simulink, Labview atd.), kde je následně systém testován. V této fázi je nutné správně namodelovat systém a správně vyhodnotit, jaké skutečnosti je možné z modelu vypustit a vhodně jej tak zjednodušit. V další fázi testování **SIL** dochází k naprogramování kódu v daném programovacím jazyce, ten je následně zkompilován, spuštěn v požadovaném softwaru a podroben testování. Ručnímu programování toho, co je již zahrnuto v modelu, se lze vyhnout pomocí automatického generování C kódu, v programovém prostředí Matlab tuto funkci provádí **MATLAB Coder** [2]. Kód je následně přenesen a testován na požadovaném hardwaru, tento krok se nazývá **PIL**. Za zmínku stojí i nástroj TargetLink od firmy dSpace, který je užíván především v automobilovém průmyslu. Tento nástroj umožňuje generovat vysoce efektivní C++ kód, který je možné nahrát do Electronic Control Unit (ECU), tím odpadá nutnost části kódu psát ručně. Poslední fází před nasazením systému je **HIL**, ve které je řídící systém připojen k reálnému systému. V této fázi řídící systém komunikuje s reálnými vstupy a výstupy [3, 2].

Další velkou výhodou využití FBD je jednodušší testování a kontrola kvality navrženého systému. Testování systému ve fázích vývoje vede ke zvýšení bezpečnosti a spolehlivosti navrhovaných systémů. Návrhář může sledovat chování systému v nežádoucích stavech bez rizika poškození systému a je schopen navrhnout algoritmus, který těmto stavům dokáže mnohem lépe předejít. Tento přístup velice dobře pasuje na vývoj softwaru pro PLC, kde je žádoucí využít tzv. **V-diagram** [4].

Způsob návrhu systému pomocí V-diagramu se využívá v kombinaci s MBD. Obecný postup návrhu při použití V-diagramu je znázorněn na obrázku 1. Tento přístup napomáhá neustále kontrolovat navrhovaný model s původními požadavky kladenými na systém [4]. Pokud není určitý požadavek naplněn, je nutné se vrátit k jednomu z předchozích kroků a model upravit. Model je tak vyladěn ještě před samotnou implementací. Důležitým

aspektem návrhu je testování jednotlivých fází, bez těch by nemohly být chyby odhaleny, systém je tedy ve všech fázích důkladně otestován. Z každého testování by se měly vytvořit zápisy, které následně slouží jako kontrola jakosti celého procesu. Tyto zápisy bývají často využívány kontrolory kvality, aby ohodnotili kvalitu produktu, nástroje nebo softwaru. Manuální testování je velice časově a finančně náročné, a proto se uplatňuje automatické generování testů [5]. Další rozšíření V-diagramu může představovat implementace metody **Model Checking**. Tato metoda má za úkol kontrolovat, zda byl model vytvořen přesně dle požadavků zákazníka a nedošlo tak k chybám interpretace [4, 1, 5].



Obrázek 1: *V-diagram*

V-diagram znázorňuje vývoj kompletně nového systému, pokud je ale systém už vytvořený a je nutné modifikovat jeho dílčí komponenty, jedná se o **W-diagram**. W-model se od klasického V-diagramu liší zejména v testování. Testování se liší hlavně tím, že probíhá na každé úrovni, u V-modelu k testování docházelo zejména v pravé části diagramu. Pokud je tedy systém rozdělen po částech mezi více vývojářů, tak testování provádí každý pro svou danou část zvlášť [6].

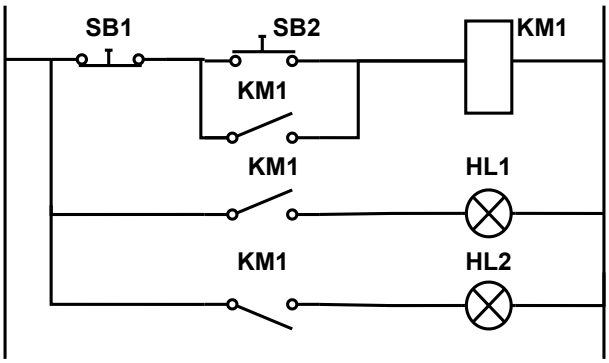
Veškerý SW byl navrhován pro Mitsubishi PLC typu **FX5**, vývoj byl prováděn v souladu s výše zmíněnými metodikami. Hlavní program reprezentující linku byl navrhován z jednotlivých funkčních bloků, z nichž každý blok byl testován. Výsledný otestovaný model složený z jednotlivých funkčních bloků byl následně přenesen na PLC. K danému systému bylo tak možné plně navrhnout řízení a ošetřit chyby, které simulovaný model pokrýval. Finální řídící systém je následně možné aplikovat na reálný školící model linky a ověřit tak funkčnost a spolehlivost návrhu.

2 PLC

2.1 Obecný popis PLC

PLC nebo-li programovatelný logický automat (z anglického Programmable Logic Controller) je průmyslový počítač menších rozměrů, jež je vybaven potřebným hardwarem a softwarem, pomocí kterého je automat schopen pracovat s procesy podle požadované funkce [7].

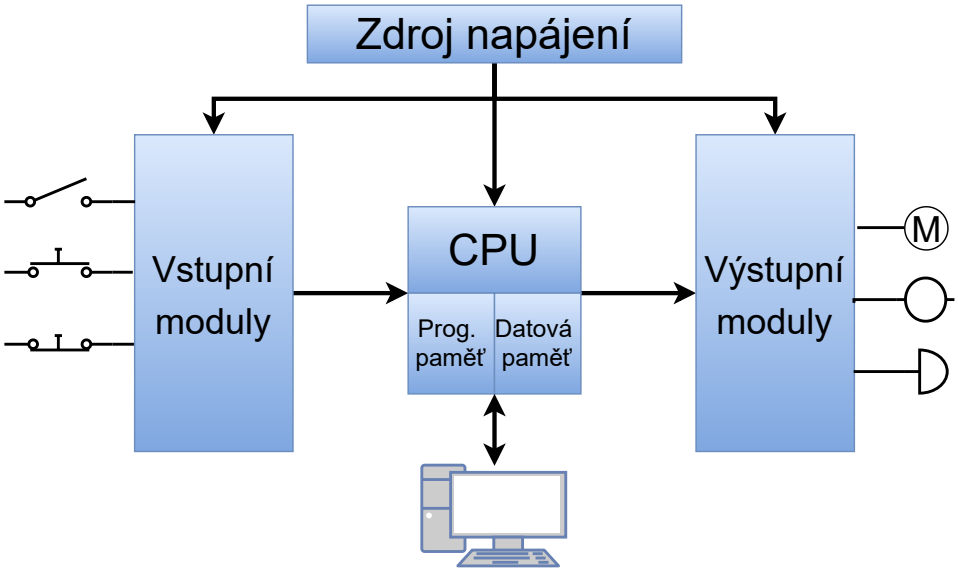
Prvotním záměrem PLC bylo nahrazení složitých reléových obvodů, jejichž úprava a případná změna byla velice časově náročná. PLC jsou oproti reléovým obvodům více flexibilní, rychlejší, jednodušší na zapojení, levnější a modulární. PLC odstraňuje potřebu složitého drátování a umožňuje vytvářet programy, které jsou kompatibilní s jakýmkoliv jiným zařízením PLC. Změny v programu je možné provést kdykoliv bez jakékoliv změny hardwarové části. Uvádí se, že pokud aplikace potřebuje více než šest relé, je mnohem levnější využít PLC [7]. Další velké výhody jsou komunikační schopnost s ostatními zařízeními a možnost ladit chyby v reálném čase, a tak odhalit případné problémy v implementaci [8, 7].



Obrázek 2: *Reléové schéma*

Každé PLC zařízení je složeno ze dvou hlavních komponentů: CPU (centrální procesorová jednotka) a vstupně/výstupní moduly. Procesorová jednotka slouží ke spuštění řídícího programu. Jednotka načte data ze vstupních modulů a na jejich základě vyšle příkazy výstupnímu modulu. Architektura PLC je znázorněna na obrázku 3 [7].

PLC lze řadit do dvou základních skupin podle množství modularity na **Kompaktní** a **Modulární**. Zatímco kompaktní PLC obsahuje všechny komponenty v jednom zařízení a tím pádem máme velmi omezené možnosti řešení, tak modulární PLC se skládá ze samostatných modulů, které můžeme různě kombinovat a dává nám takřka neomezené možnosti [7]. Typickým představitelem modulárních PLC jsou například PLC řady **QCPU** (obrázek 4). Jádro modulárního PLC se skládá ze základové jednotky, zdroje energie a alespoň jedné CPU jednotky. K základové jednotce je možno dále připojovat jednotlivé moduly jako například digitální nebo analogové vstupy a výstupy. Možnost modulace se pak následně odvíjí od velikosti racku a paměti PLC [9, 10].



Obrázek 3: *Architektura PLC*

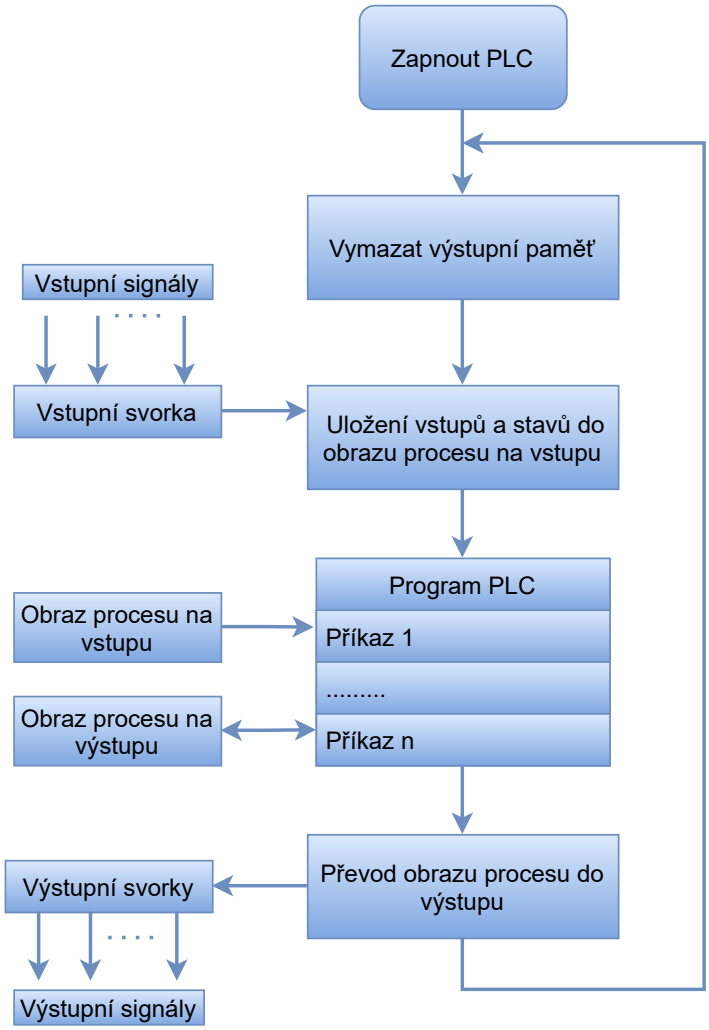
Obrázek 4: *Mitsubishi Q Series PLC CPU*

2.2 Zpracování programu v PLC

PLC program je složen ze sekvencí příkazů, které se vykonávají po sobě. Tato sekvence příkazů je cyklická a vykonává se nepřetržitě ve smyčce. Jeden programový cyklus PLC je nazýván jako **scan** a udává nám rychlost odezvy PLC [7, 8].

Program není prováděn přímo na zapojených vstupech a výstupech, ale na **obrazu procesu** vstupu a výstupu. Automat tedy na začátku každého cyklu zjistí stavy vstupů a uloží je do přechodové paměti, kterou při vykonávání sekvence příkazů využívá. Výstupy jsou následně též uloženy do přechodové paměti, odkud jsou nadále převedeny na skutečné výstupy. Zpracování programu v PLC je znázorněno na obrázku 5 [8].

Po spuštění programu tedy PLC pracuje se stavy, jež byly uloženy na začátku cyklu do obrazu procesu. Pokud dojde ke změně na vstupu, PLC tuto změnu stavu vyhodnotí až v následujícím cyklu/scanu. Sekvence příkazů je prováděna od shora dolů v pořadí, v jakém byly příkazy napsány, výsledky jednotlivých kroků jsou ukládány a program s nimi může v průběhu cyklu pracovat. Výsledky operací jsou ukládány do přechodné paměti, ze které jsou na konci cyklu zapsány do výstupu [8].



Obrázek 5: *Zpracování programu v PLC*

Nevýhoda PLC může být už výše naznačený problém, kdy se nám vstupní veličina mění s vysokou frekvencí a PLC je schopno tuto změnu detekovat až po uplynutí programového cyklu. Tato nevýhoda je z velké části vyrovnána velmi krátkou dobou programového cyklu. Je nutné zmínit, že doba programového cyklu se odvíjí od počtu a typu prováděných příkazů. Pokud je potřeba řídit veličinu, jejíž charakter se mění s velmi malou periodou, je nezbytné využít **programové přerušení** (interrupt). Aplikace reálného času mívají zpravidla periodu vykonání 1ms [8].

3 Programování PLC

Tato kapitola bude věnována především programovacím jazykům PLC. Na začátku je však potřebné zmínit základy, které platí pro všechny programovací jazyky.

3.1 Základní prvky programování PLC

3.1.1 Proměnné

Proměnné lze třídit na globální a lokální. Ke globálním proměnným má přístup program v jakékoliv vrstvě, k lokálním má přístup pouze vrstva, ke které jsou proměnné vázané. Lokální proměnné u funkčních bloků se následně dělí na vstupní, výstupní a vstupně-výstupní [11].

3.1.2 Základní datové typy

Základním datovým typem pro tvorbu kombinační logiky je **bit**, který má pouze dva stavy - 1 nebo 0. V praxi tento datový typ může reprezentovat například stav čidla (aktivní/neaktivní). Dalšími základními datovými typy jsou **WORD** (integer) nebo **FLOAT** (reálné číslo), od těchto typů jsou odvozené další datové typy. Mezi další datové typy, které je nezbytné zmínit a které se objeví v praktické části, jsou časovače **TIMER**. Časovačů je v PLC celá řada, nejpoužívanější jsou časovače typu TON a TOFF. TON časovače fungují na bázi pozdržení aktivního signálu na vstupu po určitou zadanou dobu, časovače typu TOF fungují analogicky s neaktivním signálem. Rozsahy datových typů jsou znázorněny v tabulce 1, V tabulce se vyskytují datové typy **signed** (znaménkové) a **unsigned** (bezznaménkové), dle rozsahů je zřejmé, že rozdíl je v tom, že znaménkové datové typy počítají se zápornými hodnotami, bezznaménkové nikoliv [11].

Datový typ	Rozsah min	Rozsah max
Word [Signed]	-32768	32767
Double Word [Signed]	-2147483648	2147483647
Word [Unsigned]	0	65535
Double Word [Unsigned]	0	4294967295
FLOAT	-2^{128}	2^{128}

Tabulka 1: *Tabulka datových typů*

3.1.3 Pole

Ve složitějších implementacích je potřeba někdy využít odvozeného datového typu pole. Pole v PLC musí obsahovat prvky stejného datového typu a je možné pracovat i s více dimenzionálními poli. K jednotlivým prvkům pole se přistupuje pomocí indexů v hranatých závorkách. Pole je indexováno od nuly [11].

3.1.4 Struktury

Struktura je datový typ, který nám umožňuje seskupit více proměnných různých datových typů. K jednotlivým proměnným se přistupuje přes tečkovou notaci [11].

3.1.5 Funkce (FUN)

PLC programovací jazyky, stejně jako jiné jazyky podporují tvorbu funkcí. Každá funkce může mít několik vstupních a výstupních hodnot. Dále je možné využívat vestavěné funkce, ale i vytvářet nové, vlastní. Funkce mohou být volané v programu nebo i v jiné funkci [11].

3.1.6 Funkční blok (FB)

Funkční blok je základní stavební jednotkou programovacího jazyka FBD. Funkční blok popisuje relaci mezi vstupem a výstupem. Každý blok může mít libovolný počet vstupů a výstupů. Hlavním rozdílem oproti funkci (FUN) je vnitřní paměť, kterou si funkční blok udržuje. Oproti funkci je tedy možné dostat různé hodnoty výstupů při stejných vstupních hodnotách [11, 12].

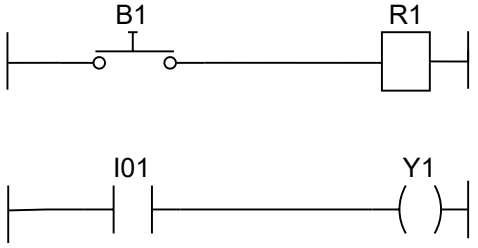
3.1.7 Paměťové zařízení

PLC má určitý počet vstupů a výstupů, který je možné rozšířit přes moduly, čemuž je věnována kapitola 2.1. V programu se na vstupní zařízení odkazuje písmenem **I** s adresou daného vstupu, analogicky se takto odkazuje na výstupní zařízení písmenem **Y** a adresou. PLC je dále vybaveno paměťovými bity **M**, které slouží pro uchování bitové hodnoty, a datovými registry **D**, ve kterých se ukládají složitější datové typy. Existuje celá řada dalších pamětí, kterými je PLC vybaveno, mnohé z nich jsou ale užívané ve velmi omezeném měřítku. Za zmínku stojí redundantní paměti, jejichž obsah se uchovává i po vypnutí PLC. Každý typ paměti má v PLC vymezený adresový prostor, který je možné omezit či rozšířit na úkor jiného typu paměti [11].

3.2 Programovací jazyky

Programovacích jazyků PLC je celá řada, v této kapitole budou zmíněné tři nejpoužívanější typy. Z těchto tří programovacích jazyků byl využit jazyk **FBD/LD** pro sestavení školících úloh hlavně díky možnosti modularity.

3.2.1 Ladder diagram



Obrázek 6: *Kontaktní logika v porovnání s Ladder diagramem*

Programovací jazyk Ladder diagram vychází z logiky zapojení reléových schémat, podobnostem PLC a reléových obvodů je věnována kapitola 2.1. Demonstrace jisté analogie mezi tímto programovacím jazykem a kontaktní logikou je znázorněna na obrázku 6.

Reálné fyzické periferie (tlačítko **B1** a relé **R1**) byly nahrazeny kontakty a výstupními buňkami, které odkazují na vstupní/výstupní svorku [8].

Ladder diagram je tedy programovací jazyk, který umožňuje řídit sekvenci řízení pomocí kombinační logiky. Základním stavebním kamenem tohoto programovacího jazyka jsou **kontakty** a **cívky** (resp. výstupní příkaz). Kontakty se dále dělí na kontakty spínací **NO** (normal open), rozpínací **NC** (normal closed) a kontakty reagující na sestupnou či vzestupnou hranu. Dále existují negované kontakty, které reagují na sestupné/vzestupné hrany, ty se však v praxi moc nepoužívají [8].

Obrázek 7: *Základní příkazy Ladder diagramu*

3.2.2 Strukturovaný jazyk

Strukturovaný jazyk (zkratka **ST**) je narozdíl od Ladder diagramu programovacím jazykem psaným v textové formě se strukturou podobnou jazyku C. Tento jazyk se využívá hlavně ve složitějších implementacích, které nemohou být dobře popsány Ladder diagramem. Jazyk se často kombinuje s funkčními bloky [11, 12].

Příkazy pro větvení		Iterační příkazy	

IF.....ELSE	IF Var1 THEN
	Var2 := TRUE;
	ELSE
	Var2 := FALSE;
	END_IF;
		FOR	FOR intV1:=0
			TO 30;
			BY 1 DO
			Var1 := Var1+2;
			END_FOR;

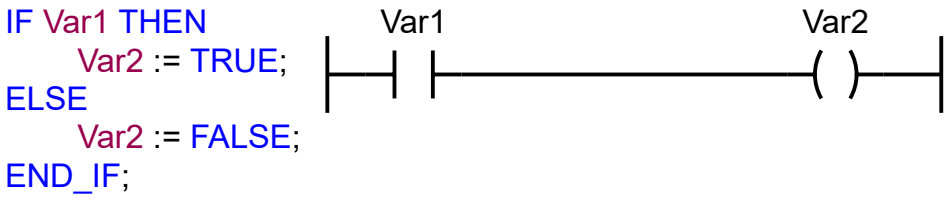
IF.....ELSIF	IF Var1 THEN
	intV1:=intV1+2;
	ELSEIF Var2 THEN
	intV2:=intV2*2;
	ELSEIF Var3 THEN
	intV3:=intV1-2 ;
	END_IF;
		WHILE	WHILE intV1=10 DO
			intV1:=intV1+2;
			END_WHILE;

CASE	CASE A1 THEN
	1:
	intV1:=intV1+2;
	2:
	intV1:=intV1*2;
	ELSE
	intV1:=intV1+5;
	END_CASE;
		REPEAT	REPEAT
			intV1:=intV1+1;
			UNTIL intV1=10;
			END_REPEAT;

Tabulka 2: *Syntaxe jazyka ST*

Jak je výše zmíněno, syntaxe je velice podobná jazyku C. Každý příkaz je tedy ukončen středníkem. Lze využívat proměnné, do kterých se hodnota deklaruje pomocí dvojice znaků :=. Stejně tak jako Ladder diagram, tak jazyk ST obsahuje operátory pro sčítání, násobení, dělení, porovnání hodnot, logické operace a mnoho dalšího. Jeho největší výhoda přichází při používání podmínek pro větvení (if, else, case) nebo při psaní cyklů (while), tyto příkazy se mnohem snáz používají ve formě textu. Ukázka syntaxe některých příkazů je znázorněna v tabulce 2 [11].

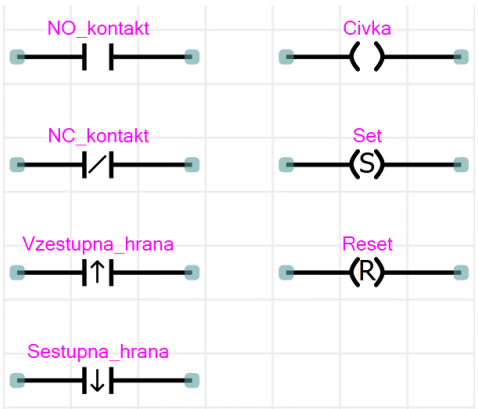
Tuto variantu je tak vhodnější používat v momentě, kdy se v programu objevuje více smyček a větvení. Ty by při použití Ladder diagramu mohly vést na složité a nepřehledné schémata. Porovnání Ladder diagramu a strukturovaného textu je znázorněno na obrázku 8 [11].



Obrázek 8: *Porovnání ST a LD*

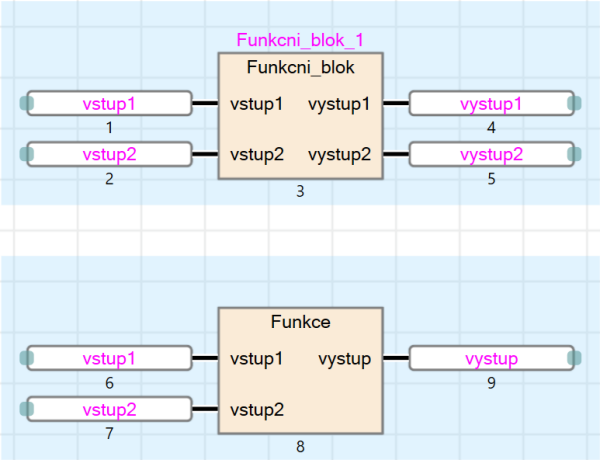
3.2.3 FBD/LD jazyk

Jazyk FBD/LD jak už z názvu napovídá slouží jako kombinace funkčních blokových diagramů a Ladder diagramu. Využití funkčních bloků umožňuje usnadnit a zpřehlednit kód, jelikož do funkčních bloků lze zabalit část kódu, jež se v programu opakovaně objevuje. Vývoj programu pomocí funkčních bloků je též přínosný z hlediska testování, zejména při automatickém generování testů. Díky těmto aspektům se pro tento programovací jazyk dobře uplatní metoda návrhu **Model-Based Design** [11, 13, 1].



Obrázek 9: *LD elementy*

V programu se tedy vyskytují elementy funkčních bloků (FBD) a Ladder Diagramu (LD), kterým je věnována kapitola 3.2.1. Základní elementy z části LD jsou znázorněny na obrázku 9, tyto elementy tvoří samotné jádro programu, které může být následně zaobaleno do funkcí či funkčních bloků. FBD část se následně skládá z funkcí (FUN), funkčních bloků (FBD) a proměnných, které jsou vyobrazeny na obrázku 10 [11, 13].



Obrázek 10: *FBD elementy*

Funkční blok či funkce mohou být psané v různých jazycích (LD, FBD/LD, ST) bez ohledu na jazyku, ve kterým je psán hlavní program. Tento aspekt vede na obrovskou výhodu, kdy programátor může složitější části kódu řešit v rámci funkčního bloku pomocí jazyka strukturovaného textu, přičemž hlavní část kódu je psaná v jazyce LD. Program se tak stává lépe rozšířitelný a modulární, než tomu bylo u předchozích jazyků [11, 13].

Funkční bloky a funkce se liší pouze v užívání vnitřní paměti. Funkční bloky je možné tedy využívat i na složitější funkce, při kterých je nutné udržovat hodnoty proměnných uvnitř bloku. Příkladem aplikace funkčního bloku je píst z první školící úlohy, kde bylo nutné uchovávat aktuální polohu pístu jako vnitřní proměnnou. V tomto případě tedy záviselo na stavu vnitřních hodnot bloku, a tak blok poskytuje pro stejné hodnoty vstupu různé hodnoty výstupu [11].

4 Seznámení s nástroji pro tvorbu programů PLC

Tato kapitola se zabývá nástroji pro tvorbu programů PLC a vizualizace panelů PLC. Budou vysvětleny základní kroky při tvorbě programů a užitečné standardy při sestavování algoritmů. Pro tuto baklářskou práci jsou důležité především prostředí **GX works3** a **GT Designer3**, jelikož byly využity pro sestavení školících úloh.

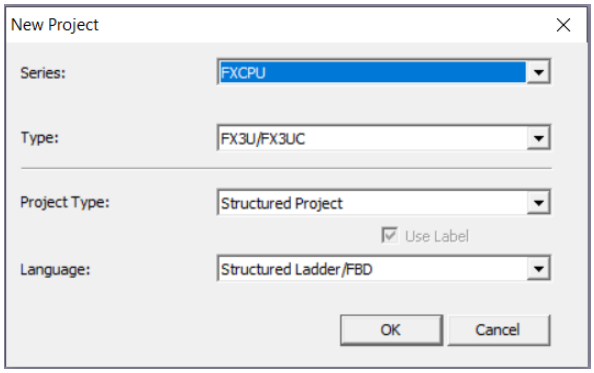
4.1 GX Works2

Prostředí GX Works2 je nástupcem vývojového prostředí **GX Developer**. Oproti svému předchůdci přineslo řadu novinek jako využití programovacího jazyka FBD/LD nebo uživatelsky přívětivější vývojové prostředí.

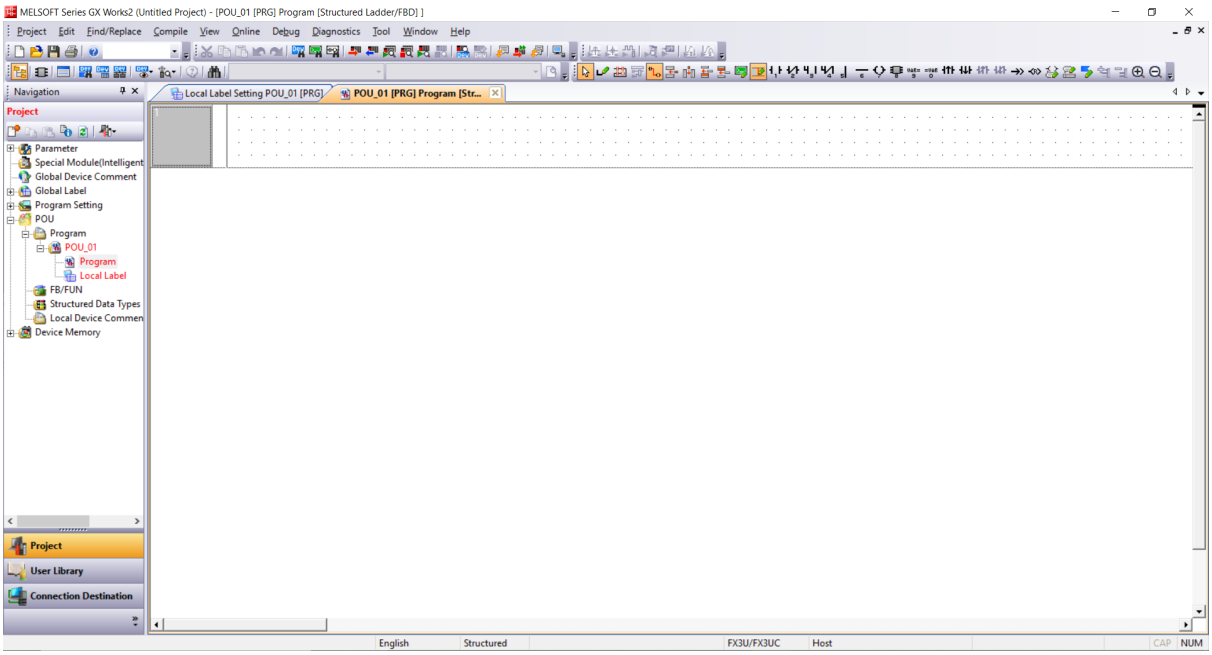
Vývojové prostředí nabízí tvorbu dvou typů projektů - **jednoduchý projekt** a **strukturovaný projekt**. Jednoduchý projekt nabízí tvorbu pouze jednoduché sekvence příkazů, stejně jako tomu bylo ve vývojovém prostředí GX Developer. Tento typ projektu podporuje pouze dva grafické jazyky (Ladder a SFC) a jeden textový (ST). Strukturovaný projekt, jak již název napovídá, nabízí tvorbu strukturovaného programu. Program je tak možné rozdělit do menších celků, čímž se stává lépe pochopitelným a modulárním. Oproti jednoduchému projektu je podporován jeden grafický jazyk navíc a to je FBD/LD [14].

4.1.1 Vytvoření nového projektu

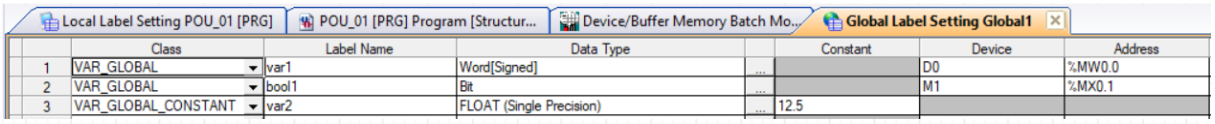
Po provedení volby vytvoření nového projektu se před uživatelem objeví tabulka vyobrazena na obrázku 11. V kolonce **Series** si uživatel vybírá mezi řadu PLC a v kolonce **Module Type** je možné následně zvolit přesný typ PLC. V tomto případě bylo zvoleno PLC typu FX3U. Následně je potřebné zvolit typ projektu, zde jsou dvě možnosti, které byly popsané v kapitole 4.1. Typ projektu byl tedy zvolený jako strukturovaný projekt z důvodu demonstrace důležitých aspektů využitých v úlohách z praktické části. V poslední kolonce se následně volí programovací jazyk, v tomto případě jazyk FBD/LD. Důležité je zmínit, že volba typu projektu a programovacího jazyka se odvíjí od zvoleného HW. Jak je výše zmíněno GX Works2 je do jisté míry kompatibilní s GX Developer. Některé PLC tedy nepodporují tvorbu strukturovaných projektů nebo třeba jazyk FBD/LD. [14, 15].



Obrázek 11: *Vytvoření nového projektu*



Obrázek 12: *Vývojové prostředí GX Works2*



Obrázek 13: *Nastavení globálních proměnných*

Po vytvoření nového projektu je před uživatelem vývojové prostředí zobrazené na obrázku 12. Na horní liště jsou umístěny příkazy, nástroje pro debugging, připojení k PLC či pro lazení programu. V levé liště lze nalézt konkrétní parametry daného projektu:

- **Parameter**: Parametry PLC - název PLC, paměťová zařízení a jejich rozpoložení v adresním prostoru PLC, nastavení ethernetu a mnoho dalších.
- **Global labels**: Nastavení globálních proměnných. Uživatel může deklarovat několik sad globálních proměnných. Na obrázku 13 je znázorněna sada *Global1*, ve které jsou vytvořené příklady demonstrující možnosti nastavení. Nastavení každé proměnné obsahuje:
  - Class: Nastavení, zda se bude jednat o proměnnou či konstantní proměnnou.
  - Label Name: Nastavení názvu dané proměnné.
  - Data Type: Nastavení datového typu, podle čehož se odvíjí i nastavení paměťového zařízení, viz kapitola 3.1.
  - Constant: Nastavení konstantní hodnoty. Políčko je dostupné pouze v případě, že bylo v kolonce *Class* zvoleno, že se jedná o konstantu.

  - Device: Nastavení paměťového zařízení.
  - Adress: Adresa paměťového zařízení.
  - Comment: Políčko pro programátorský komentář. Psaní komentářů k proměnným je doporučeno hlavně z hlediska přehlednosti programu.

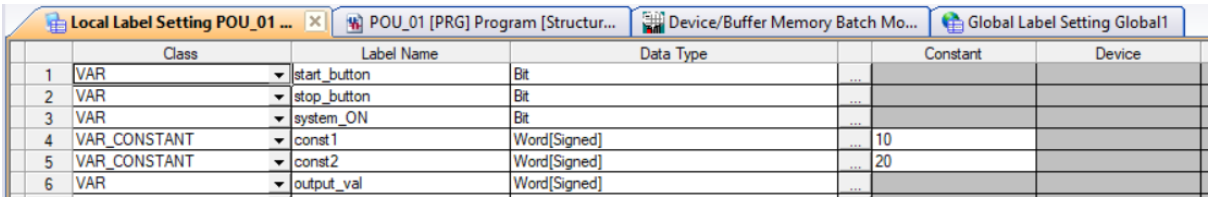
- **Program Setting**: Slouží k nastavení pořadí vykonávání jednotlivých programových bloků.
- **POU**: Programová část. Do programové části lze vkládat programové bloky. Každý programový blok obsahuje svou vlastni sadu lokálních proměnných (*Local Label*) a program (*Program*).
- **FB/FUN**: Funkce a funkční bloky deklarované uživatelem.
- **Structured Data Types**: Struktury definované uživatelem.

Před začátkem psaní každého programu je nutné deklarovat hardwarovou část - přídavné analogové či digitální I/O moduly, zdroj energie a popřípadě další CPU. [14, 15].

4.1.2 První program

Pro sestavení prvního programu byl využit výchozí programový blok POU_1. Nejprve budou deklarovány proměnné, které budou použity v programové části. Pro demonstraci funkce bude vytvořen jednoduchý start-stop obvod, který po spuštění aktivuje součet dvou konstant a výsledek uloží do proměnné. V této kapitole budou tak demonstrovány základní bitové a celočíselné operace.

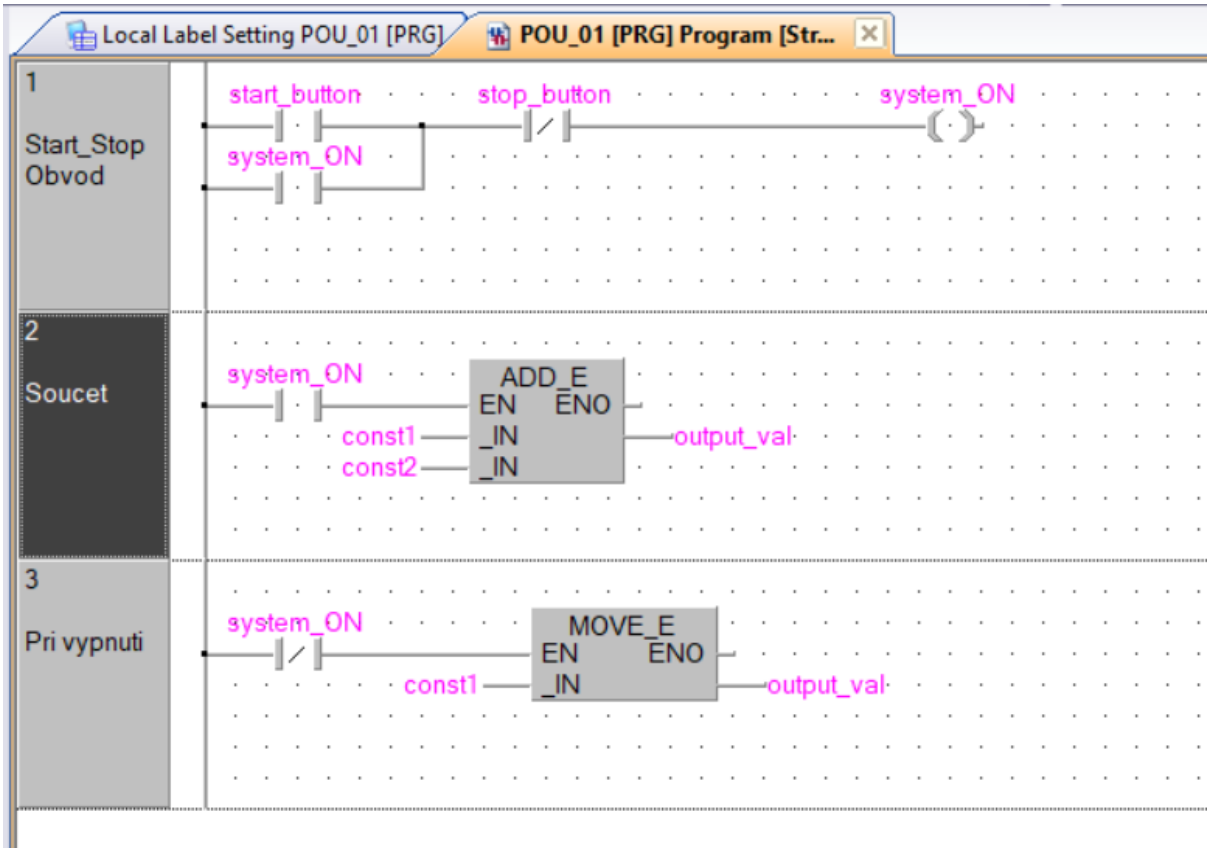
Deklarace lokálních proměnných se provádí v kolonce *Local Label*, viz kapitola 4.1.1. Pro tento program postačí deklarovat start a stop tlačítko, následně proměnnou, která detekuje aktivitu systému, konstanty a výstupní proměnnou, do které bude načten součet. Jak je na obrázku 14 možné pozorovat, k lokálním proměnným nelze přiřadit paměťové zařízení, to lze pouze u globálních proměnných [14, 15].



Obrázek 14: *Nastavení lokálních proměnných*

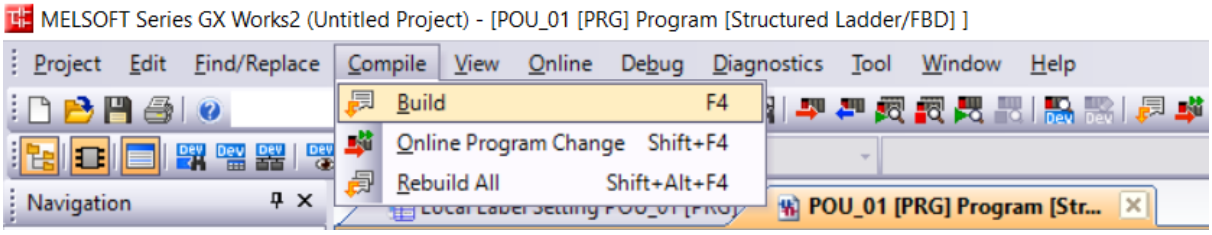
Následuje sestavení programu, což začíná start-stop obvodem, což je úplný základ většiny programů psaných pro PLC. Start-stop obvod funguje na bázi toho, že pokud je stisknuto tlačítko start a není stisknuto tlačítko stop, proměnná **system_ON** je aktivní. Aby byla proměnná aktivní i po stisknutí tlačítka zařizuje přídržný kontakt po start tlačítkem. Stisknutí stop tlačítka přeruší obvod, čímž se výstupní proměnná deaktivuje. Start-stop obvod je znázorněn v prvním **Ladder bloku** na obrázku 15 [14].

Pro přidání dalšího Ladder bloku je potřeba stisknout pravým tlačítkem aktuální blok a zvolit možnost přidání bloku. Blok může být přidán buďto pod nebo nad aktuální blok. Druhý a třetí blok slouží k vyhodnocení výstupní proměnné. Pokud je proměnná *system_On* aktivní, do výstupní hodnoty se načte součet konstant, pokud je proměnná neaktivní, do výstupní proměnné je uložen obsah proměnné *const*1. Funkční bloky lze nalézt přes **Element section** v kolonce s příkazy nebo stačí příkaz napsat, což spustí vyhledávání funkcí. Obě funkce obsahují tzv. **Enable**, jsou tedy provedeny pouze, pokud je přivedena hodnota na vstupu **EN** aktivní. Celý program je vyobrazen na obrázku 15.



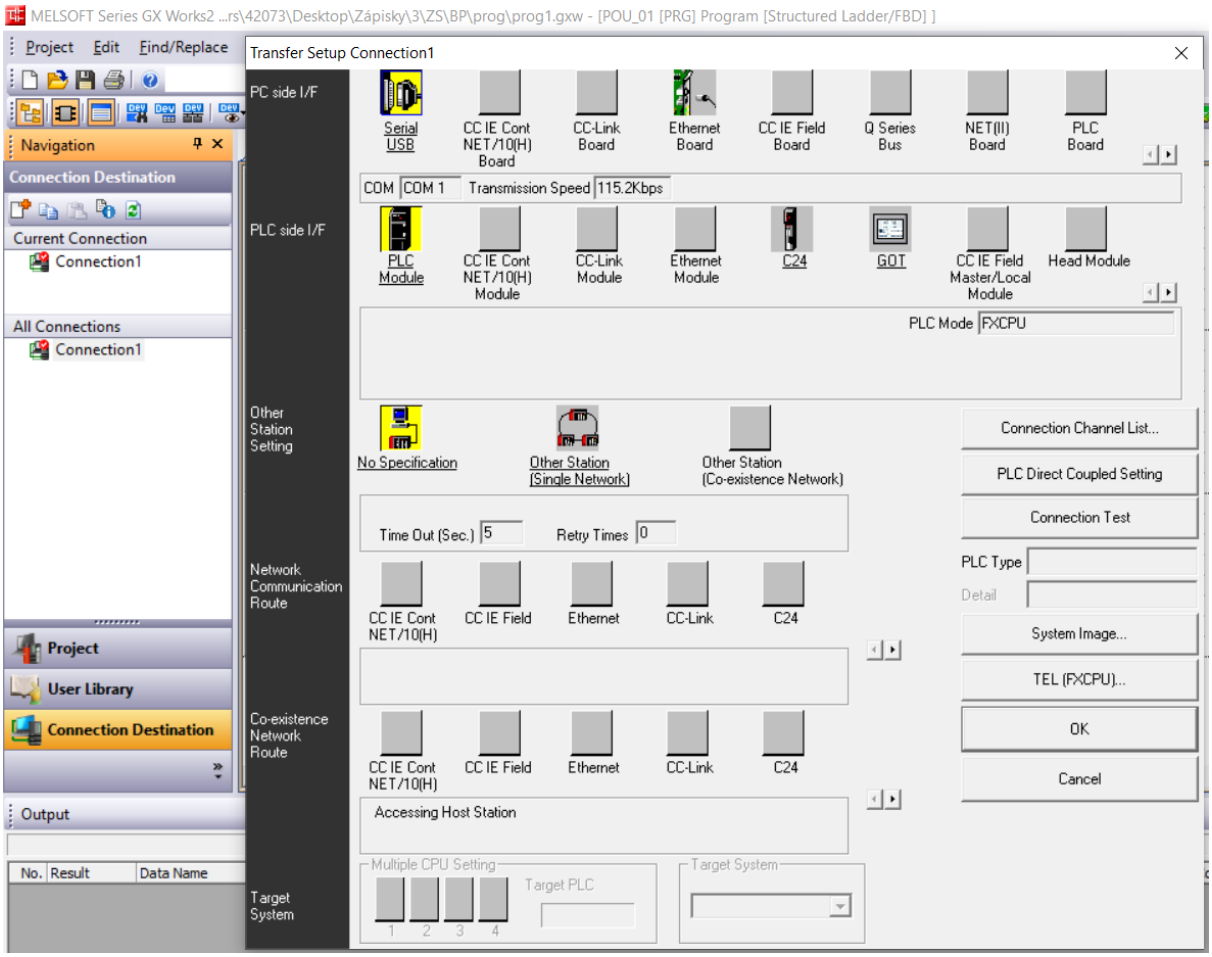
Obrázek 15: *První program*

Před nahráním nebo spuštěním simulace musí být program zkompilován. To je provedeno buďto přes klávesovou zkratku **F4** nebo přes kolonku **compile** a příkaz **Build**, viz obrázek 16.



Obrázek 16: *Kompilace*

Dalším krokem je připojení k PLC. To se provádí přes **Connection Destination**, který se nachází v levém dolním rohu vývojového prostředí, které je znázorněno na obrázku 12. Volba otevře seznam všech připojení, do kterého může uživatel přidávat nová připojení. Každé připojení je nutné konfigurovat. V konfiguraci se volí komunikační kanály jak ze strany PC, tak ze strany PLC. Způsobů komunikací s PLC je celá řada, PC může s PLC komunikovat přímo přes sériovou linku, USB či ethernet. Komunikace ale nemusí být i přímá, PC může například komunikovat s panelem, který je propojený s PLC. Možnosti připojení jsou na obrázku 17. Úspěšné připojení lze ověřit pomocí tlačítka **Connection Test**.

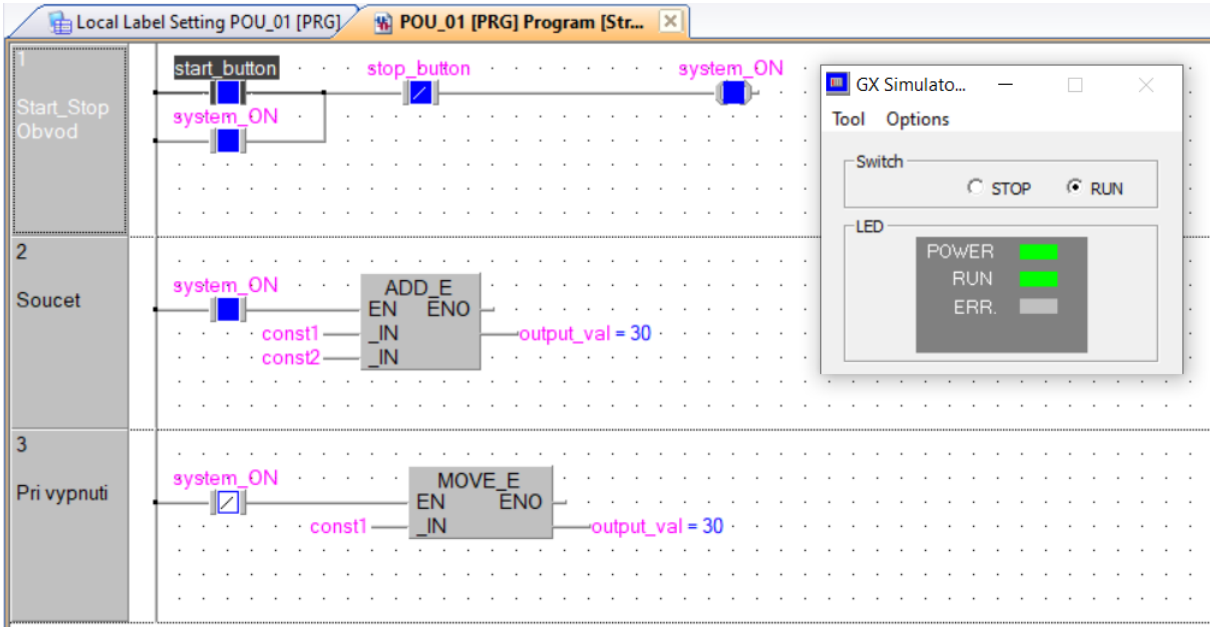


Obrázek 17: *Konfigurace připojení*

Zkompilovaný program lze po připojení nahrát do PLC přes kolonku **Online**. Program je možné do PLC nahrát, ale taky jej z něj stahovat, což se v praxi využívá především při aktualizaci programu. Programátor musí program z PLC nejprve stáhnout, následně upravit a nakonec znovu do PLC nahrát [15].

Pokud není dostupný hardware, tak prostředí umožňuje přes kolonku **Debug** spustit simulaci, ve které si uživatel může vyzkoušet funkce daného programu v tzv **Monitorovacím módu PLC**. V tomto módu prostředí ukazuje aktuální hodnoty proměnných

při běhu programu v PLC (v tomto případě se jedná pouze o simulovaný běh programu). Monitorovací mód při běhu programu je demonstrován na obrázku 18. Pro účely testování je v simulaci možné i hodnoty vstupů/výstupů modifikovat, konkrétně pomocí Online/-Monitor/Device/Buffer Memory batch.



Obrázek 18: *Monitorovací mód*

4.2 GX Works3

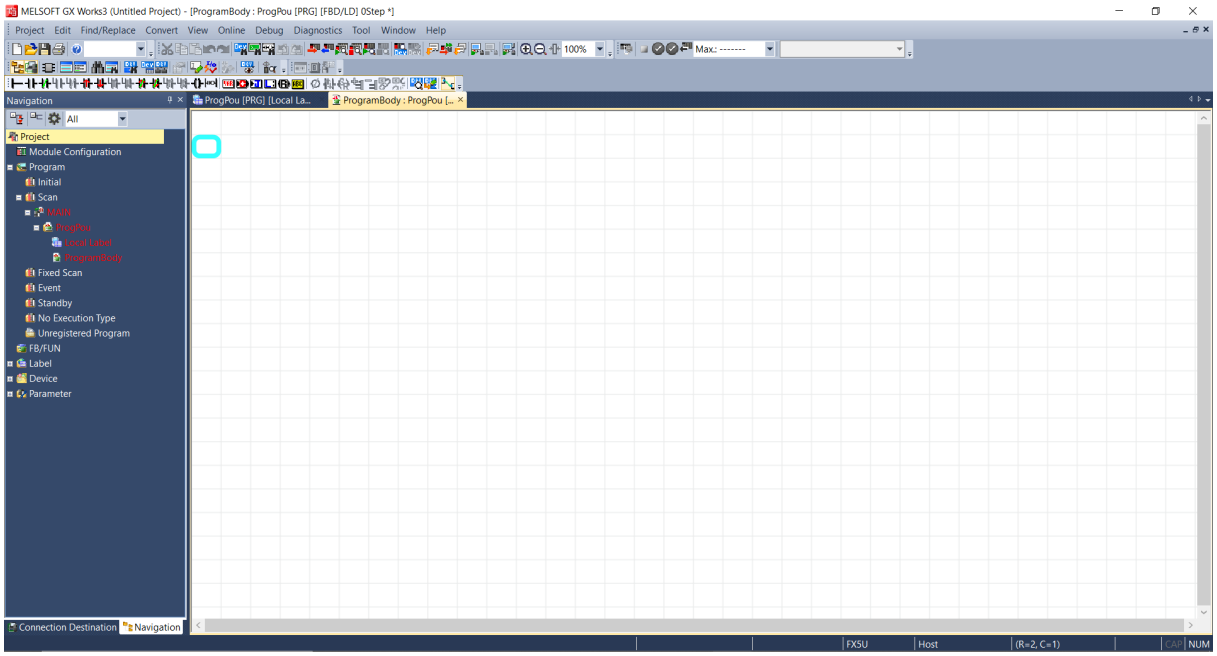
Vývojové prostředí **GX Works3** je nástupcem GX Works2, oproti svému předchůdci už však nenabízí žádnou míru kompatibilnosti se starším SW GX Developer. Neumožňuje tedy tvorbu jednoduchých projektů, které jsou popsány v kapitole 4.1. Software umožňuje vytvářet programy ve stejných jazycích jako GX Works2, záleží ovšem na zvoleném HW. Například PLC typu **FX5** nepodporuje grafický jazyk SFC [13].

Hlavním rozdílem všech tří zmíněných SW pro tvorbu PLC programů je podpora zařízení. GX Works2 podporuje především PLC typu QCPU (Q mode) a typu FX (FX0 až FX3), tyto typy též i se staršími jako například LCPU a QCPU (A mode) podporuje i vývojové prostředí GX Developer. Narozdíl tomu v prostředí GX Works3 lze programovat užší výběr, příklad PLC typu RCPU či nejnovější PLC řady FX - **FX5**. V praxi je tedy možné se setkat se všemi z uvedených programů [15, 13].

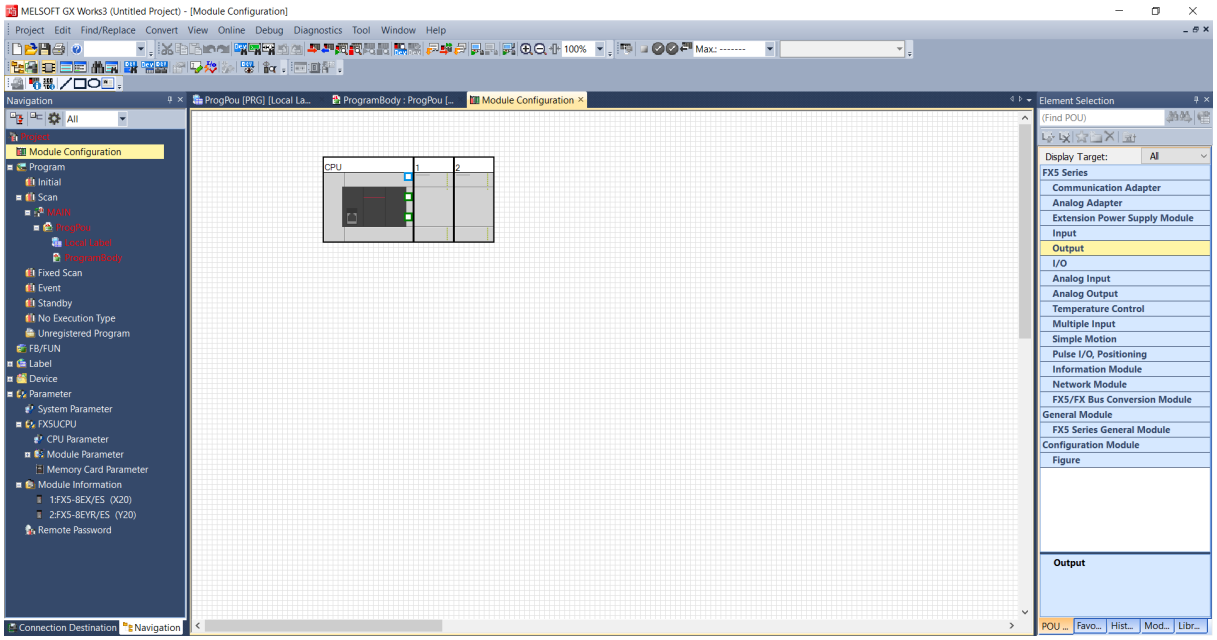
4.2.1 Vytvoření funkčního bloku

Tvorba nového programu v GX Works3 se od svého předchůdce výrazně neliší. V levé liště se dají naleznout jisté rozdíly oproti předchozí verzi. Hlavním rozdílem je Hardwarová část **Module Configuration**, ve které je před tvorbou každého programu nutné deklarovat, jaké různé moduly PLC využívá. Další novinkou je volba scanu, ve kterém bude program probíhat. Většina programu bude spadat pod větev **Scan**. Délka scanu se

v této části odvíjí od složitosti programu. V úlohách je dále využita větev **Fixed Scan**, ve které si uživatel může nastavit libovolnou periodu spouštění scanu. Tento typ se hodí pro měření veličin, které se mění rychle v čase. Dále může být fixovaný scan využitý pro operace, které musí být provedeny s pevnou periodou. Mezi tyto operace spadá například integrace použitá ve školících úlohách. Vývojové prostředí je znázorněno na obrázku 19.



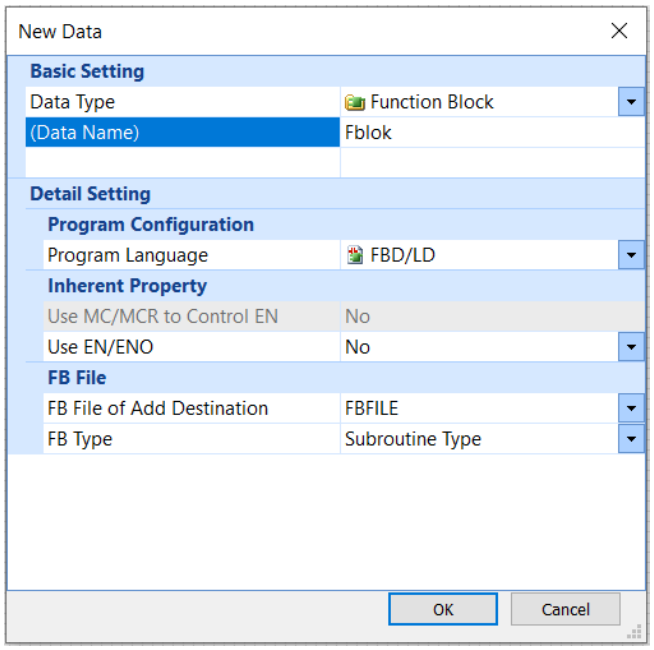
Obrázek 19: *Vývojové prostředí GX Works3*



Obrázek 20: *Hardwarová část*

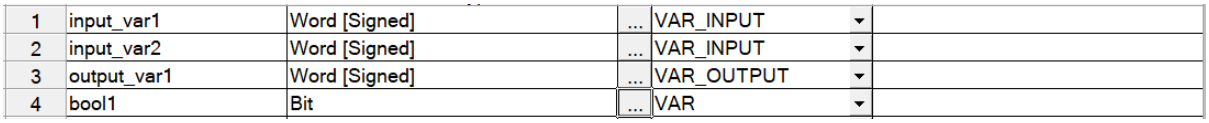
Pro tento projekt bylo zvolené PLC typu FX5. Toto PLC je kompaktní (viz kapitola 3.1), což znamená, že jej lze rozšířit jen o omezený počet modulů. Pro ukázku byly přidány dva moduly, jeden vstupní a druhý výstupní. Po uložení hardwarové části lze informace o modulech nalézt v **Module Information**, které jsou součástí větve **Parameter** v levé liště. Ukázka tvorby hardwarové části je vyobrazena na obrázku 20 [13].

Vytvoření funkčního bloku se provádí ve větvi FB/FUN, ve které se pravým tlačítkem myši navolí možnost vytvoření nových dat. Po této volbě se v tabulce zvolí datový typ (FB, FUN, STD a mnoho dalších), programovací jazyk a další různá nastavení. Pro tento případ stačí navolit datový typ **function block** a jako programovací jazyk FBD/LD. To je znázorněno na obrázku 21.



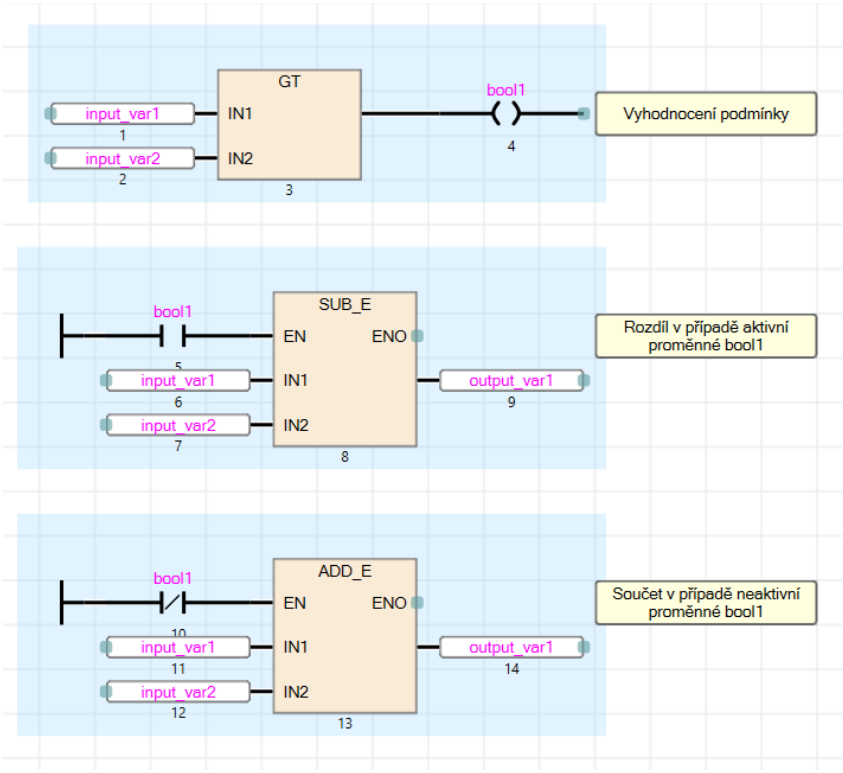
Obrázek 21: *Vytvoření funkčního bloku*

Nově vytvořený blok má svoji sadu lokálních proměnných. V tomto případě je při jejich nastavování nutné deklarovat, zda se jedná o vstupní, výstupní nebo vnitřní proměnnou. Tento funkční blok bude mít na vstupu dvě proměnné, které porovná. Pokud je první proměnná větší, na výstupu bude rozdíl těchto dvou hodnot. V obráceném případě bude na výstupu součet vstupních hodnot. V lokálních proměnných bude třeba tedy deklarovat čtyři proměnné - dvě vstupní hodnoty, jednu výstupní a jeden vnitřní stav pro vyhodnocení porovnávací podmínky (obrázek 22).



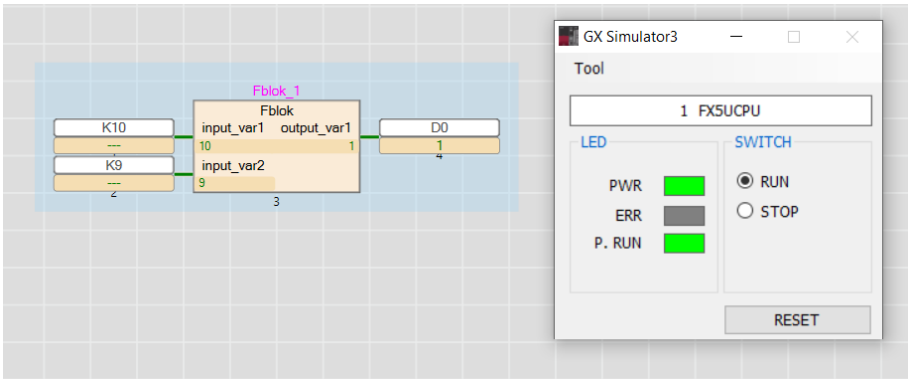
Obrázek 22: *Proměnné funkčního bloku*

Vnitřní logika bloku se následně skládá z vyhodnocovací funkce **GT** (*Greater Than*), funkce součtu **ADD** a rozdílu **SUB**. Funkce GT aktivuje výstup *bool*1, pokud je první přivedená vstupní proměnná větší než druhá. Podle proměnné *bool*1 se následně aktivuje příslušný blok pro matematickou operaci a zapíše výsledek na výstupní hodnotu (obrázek 23).



Obrázek 23: *Program uvnitř funkčního bloku*

Nově vytvořený funkční blok lze následně využít v jakékoliv části programu mimo funkční blok. Pro ověření funkce byl funkční blok využit v hlavní části programu. Jako vstupy byly přivedeny dvě konstanty a výstupní hodnota bloku bude zapsána do datového registru D0. Z obrázku 24 je zřejmé, že funkce v simulačním běhu funguje správně.



Obrázek 24: *Simulace programu*

4.3 GT Designer3

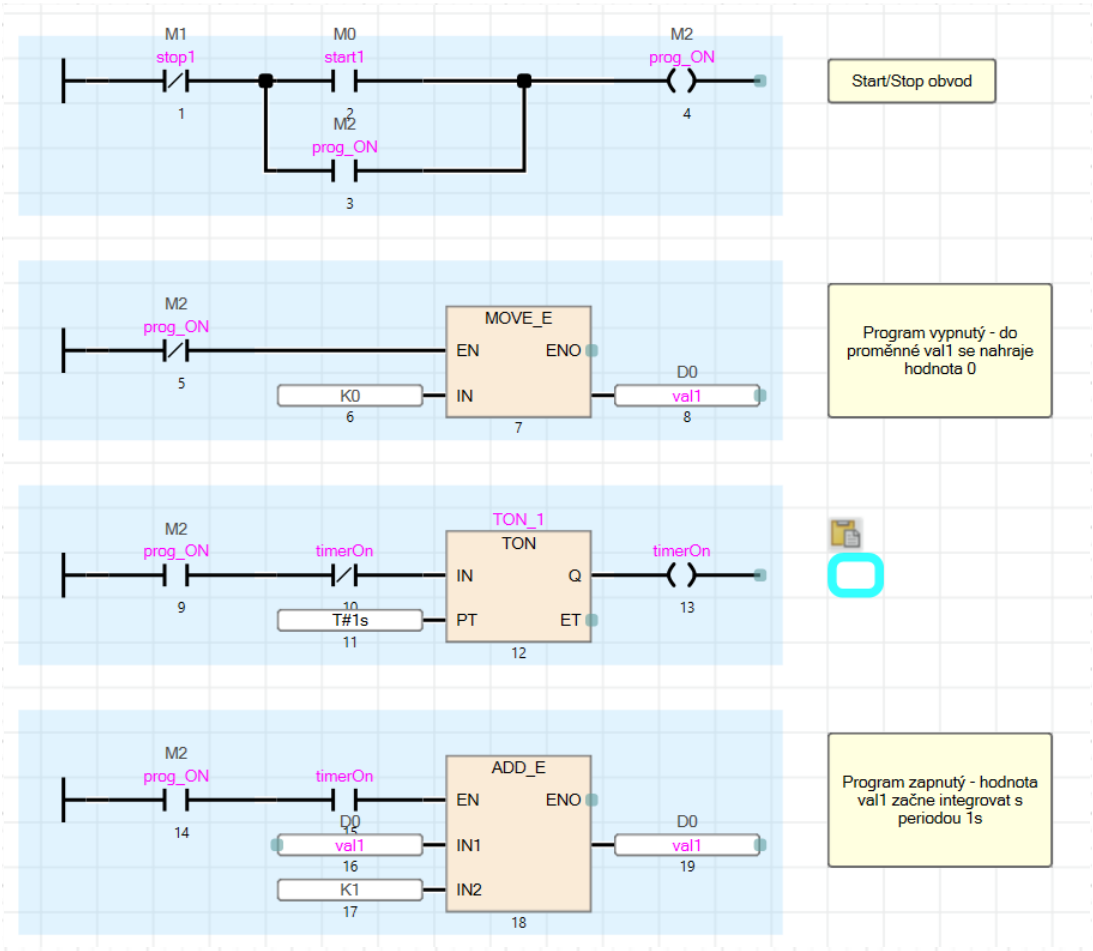
Tato kapitola bude věnována SW pro tvorbu tzv. **Human Machine Interface** (HMI), tedy vizualizace programu. Vizualizace je neméně významnou částí při tvorbě projektu. Slouží především k obsluze programu či znázornění činnosti, kterou program vykonává. Uživatel si tak může snáz představit funkci systému, který obsluhuje [16].

GT Designer slouží pro tvorbu vizualizace pro Mitsubishi panely, tzv. **Graphic Operation Terminal** (GOT). V této kapitole bude popsán pouze SW GT Designer3, jelikož byl použit pro vytvoření vizualizace všech školících úloh. Rozdíl mezi použitou verzí a jejím předchůdcem GT Designer2 je pouze v HW, na který je možné HMI vytvářet [16].

V této kapitole bude demonstrován pouze jednoduchý příklad. Všechny podrobné funkce SW jsou popsány v dokumentaci, na kterou bude v kapitole odkazováno [16].

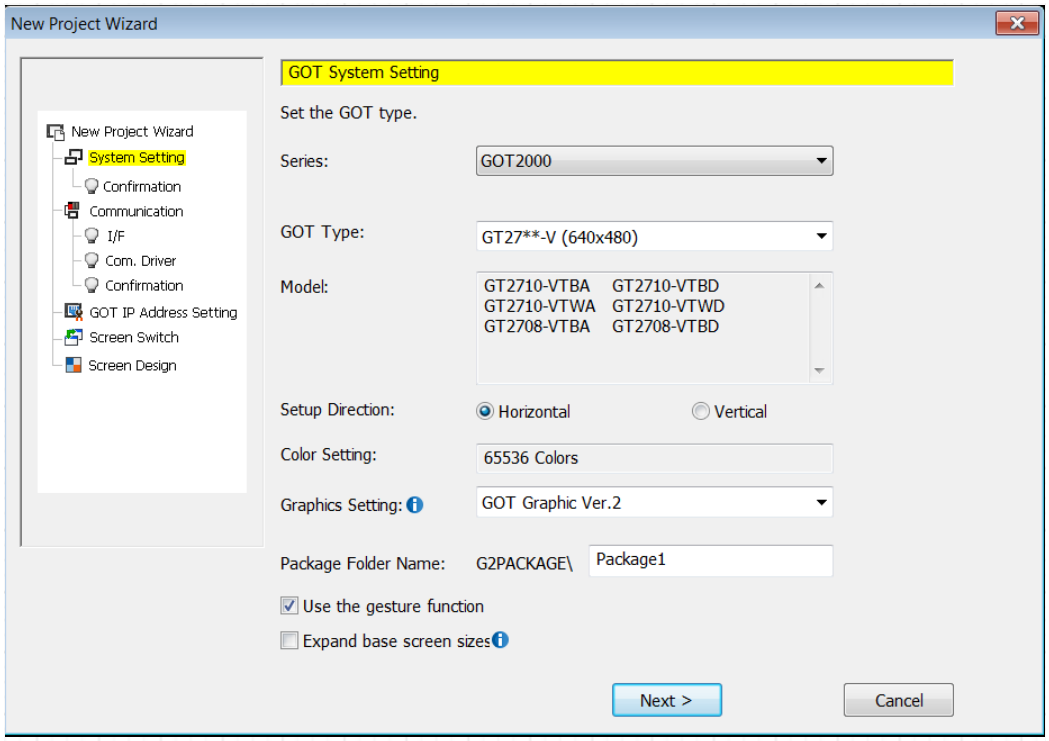
4.3.1 První vizualizace

Před samotnou vizualizací je třeba vytvořit program, který chceme vizualizovat. Pro demonstraci základních funkcí byl vytvořen krátký program, který obsahuje start-stop obvod, časovač a funkce pro matematické operace.



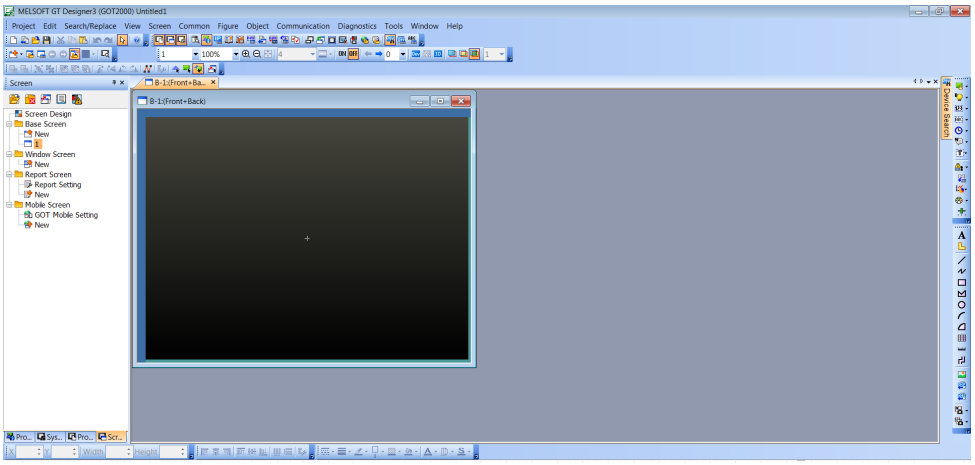
Obrázek 25: *Program pro úlohu vizualizace*

Celý program je znázorněn na obrázku 25. Proměnné *stop*1, *start*1, *prog_ON* a *val*1 musí v tomto případě být globální proměnné, jelikož jsou napojené na paměťové zařízení. Konkrétní přiřazené paměťové zařízení je napsané nad názvem proměnné. Funkce programu je taková, že při stisknutém tlačítku *stop*1 či nestisknutém tlačítku *start*1 je do hodnoty *val*1 nahrána konstantní hodnota, tedy nula. Při spuštění je hodnota *val*1 integrována o jedničku s periodou jedné sekundy. Perioda je dána časovačem *TON_*1, který při spuštěném obvodu aktivuje svůj výstup po uplynutí jedné sekundy. Výstupní hodnota *timerOn* po aktivaci následně deaktivuje časovač. V tomto případě je důležité uspořádání programu. Z kapitoly 2.2 je známo, jak PLC zpracovává program. Po uplynutí jedné sekundy je výstup časovače aktivovaný, tedy je aktivní *timerON* a tedy v následujícím kroku dojde k inkrementaci proměnné *var*1. V následujícím cyklu bude časovač opět neaktivní kvůli NC kontaktu *timerOn*.



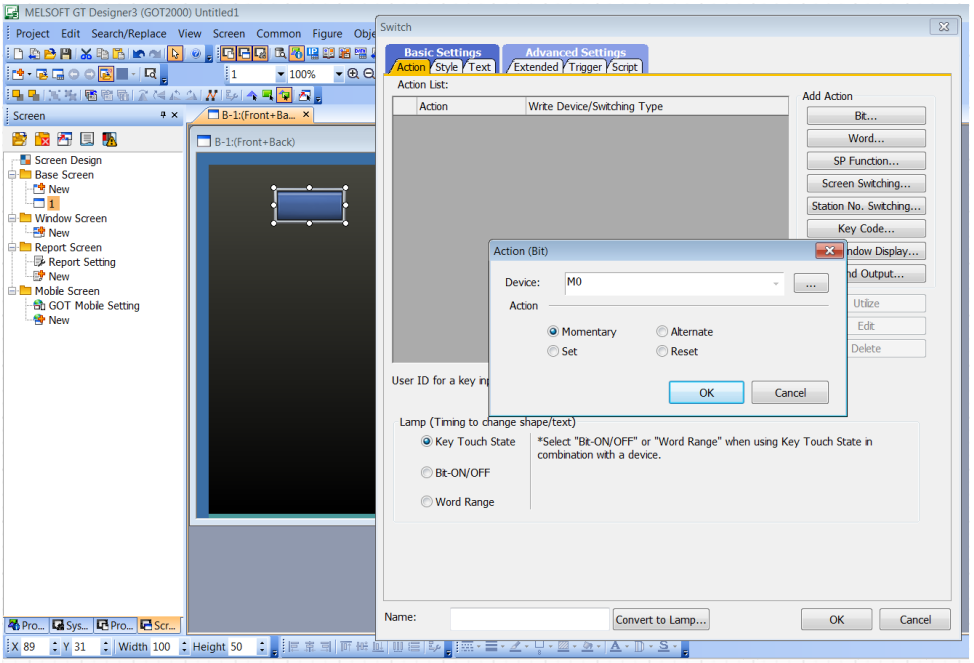
Obrázek 26: *Volba panelu*

Následuje vytvoření projektu ve vývojovém prostředí **GT Designer3**. Tvorba nového projektu se skládá z několika kroků. Po proklikání volby vytvoření nového programu se před uživatelem objeví volba několika možností nastavení projektu. Pro účely práce jsou důležité nastavení zobrazeny na obrázku 26. První z voleb **Series** slouží k volbě řady panelu, konkrétně z řady **GOT1000** či **GOT2000**. V následující kolonce **GOT Type** je možné zvolit konkrétní typ panelu. Panely se liší převážně rozlišením, je tedy nezbytné zvolit správný typ panelu, který bude ve finálním projektu použit. V tomto případě je vizualizace vytvořená pouze za účelem demonstrace, tudíž volba panelu je libovolná. Další nastavení se týkají převážně komunikace, která není předmětem této práce, podrobnější možnosti nastavení lze naleznout v citované dokumentaci [16].



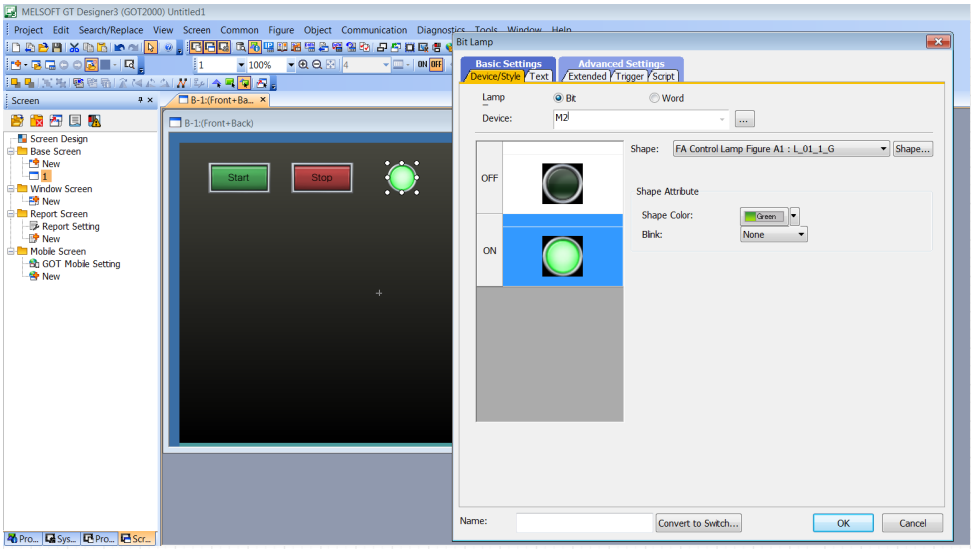
Obrázek 27: *Vývojové prostředí GT Designer3*

Vývojové prostředí, které se před uživatelem objeví po vytvoření nového programu je vyobrazeno na obrázku 27. Horní lišta obsahuje nástroje pro základní úpravy projektu, dále nástroje pro komunikaci či simulaci. V kolonce **Figures** může uživatel nalézt základní obrazce na vykreslení, jedná se především o matematické obrazce jako obdélník, trojúhelník a mnoho dalších. Kolonka **Object** zahrnuje složitější objekty jako třeba tlačítka, grafy, lampy a další. Levá lišta ukrývá tři kolonky, první z nich **Screens** je zobrazená na obrázku a slouží k nastavení obrazovek (přidání, odebrání a další). Druhá z nich **System** je věnována hlavně nastavení prostředí, komunikačním kanálům a jiný nastavením, které lze nalézt v dokumentaci [16]. V příkladu bude využita i poslední kolonka **Project**, ve které je možné vytvářet vlastní obrazce (kolonka **Parts**) či nastavení alarmů, komentářů, logování a dalších. Uprostřed je následně vyobrazena první obrazovka, na kterou lze vkládat objekty.



Obrázek 28: *Nastavení objektu Switch*

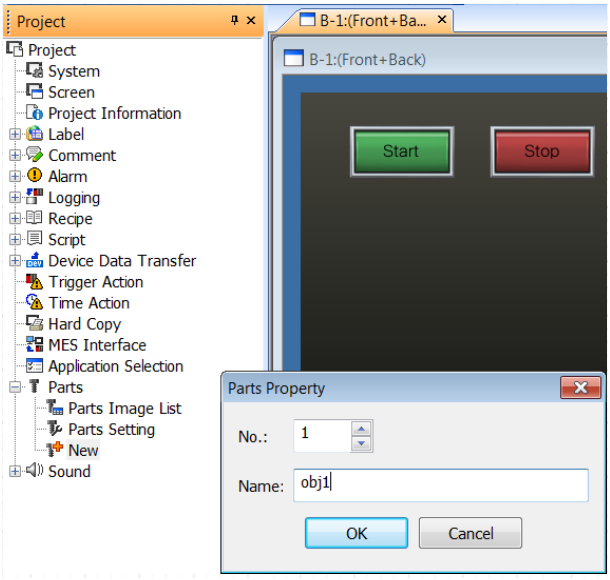
Prvním krokem je vytvoření tlačítek pro start-stop obvod. Tlačítko lze tedy nalézt v kolonce **Object**, jedná se konkrétně o objekt **Switch**. Po přenesení tlačítka na obrazovku je nutné nastavit jeho parametry. První karta **Basic Settings** zahrnuje kolonky **Action**, **Style** a **Text**. První kolonka slouží k volbě akce, která se provede po stisku. V tomto případě chceme, aby se po stisku aktivoval bit, který je vázaný k proměnné *start*1. Za zmínku stojí dále akce **Screen Switching**, která umožňuje vytvářet tlačítka, pomocí kterých se přepínají jednotlivá okna. Po volbě bitové akce je třeba navázat konkrétní paměťové zařízení, v tomto případě $M0$, viz obrázek 25. Přesná akce bude **Momentary**, tudíž tlačítko bude aktivní pouze v případě, že ho uživatel drží stisknuté. V dalších dvou kartách lze nastavit text a grafické parametry tlačítka. Analytickým způsobem bylo vytvořeno tlačítko pro vypnutí programu. Všechny parametry jsou vyznačené na obrázku 28.



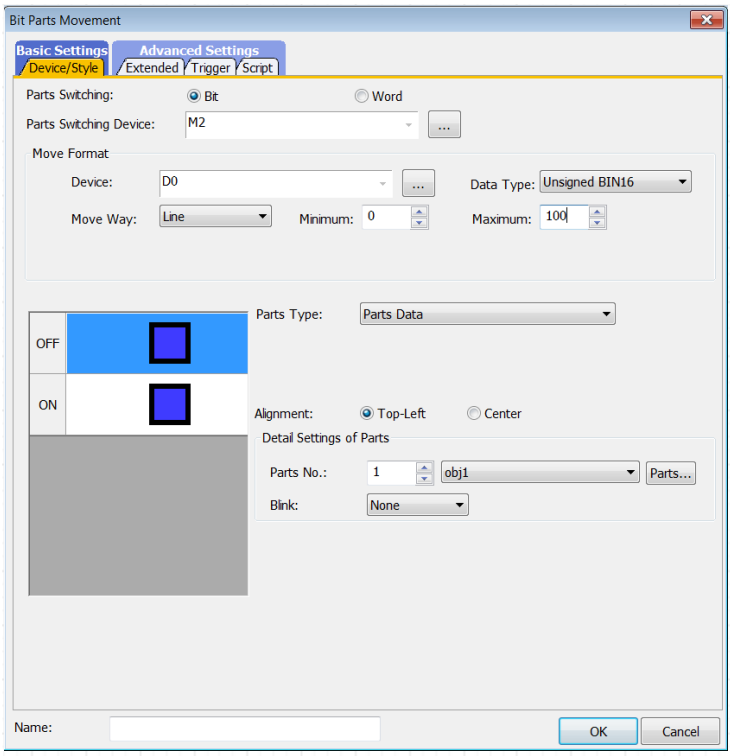
Obrázek 29: *Nastavení objektu Lamp*

Dalším krokem bude nastavení objektu **Lamp**, který se nachází ve stejné kolonce jako tlačítko. Jedná se přesně o typ **bit Lamp**, objekt tedy reaguje na změnu bitové hodnoty. Žárovka bude tedy svítit v případě aktivního bitu. V tomto případě se jedná o bit navázaný na proměnnou *prog_ON*, který detekuje chod programu. Stejně jako u tlačítka je možné upravit i grafické parametry. Všechny nastavované parametry jsou zobrazeny na obrázku 29.

Poslední částí vizualizace bude pohyb po přímce vytvořeného objektu. První je tedy nutné vytvořit si objekt, k tomu slouží kolonka **Project** v levé liště, ve které se nachází větev **Parts**, ve které bude deklarován nový objekt, viz obrázek 30. Do nového objektu lze nakreslit jakýkoliv obrazec, který může být následně využit v animaci. Důležité je zmínit, že souřadný systém má počátek v levém horním rohu. Při tvorbě obrazce je s tím potřebné počítat, veškeré animace vychází právě z tohoto počátečního bodu. Pro účely demonstrace byl vytvořen čtverec o délce strany 20 pixelů umístěný do levého horního rohu [16]. Animaci pohybu lze následně vytvořit přes kolonku **Object**, objekt **Parts Movement**, konkrétně **bit Parts**. Uživatel dále nastaví, který **Part** chce využít, zde je zvolený vytvořený objekt modrého čtverce. Následně je nutné zvolit aktivační bit, v tomto

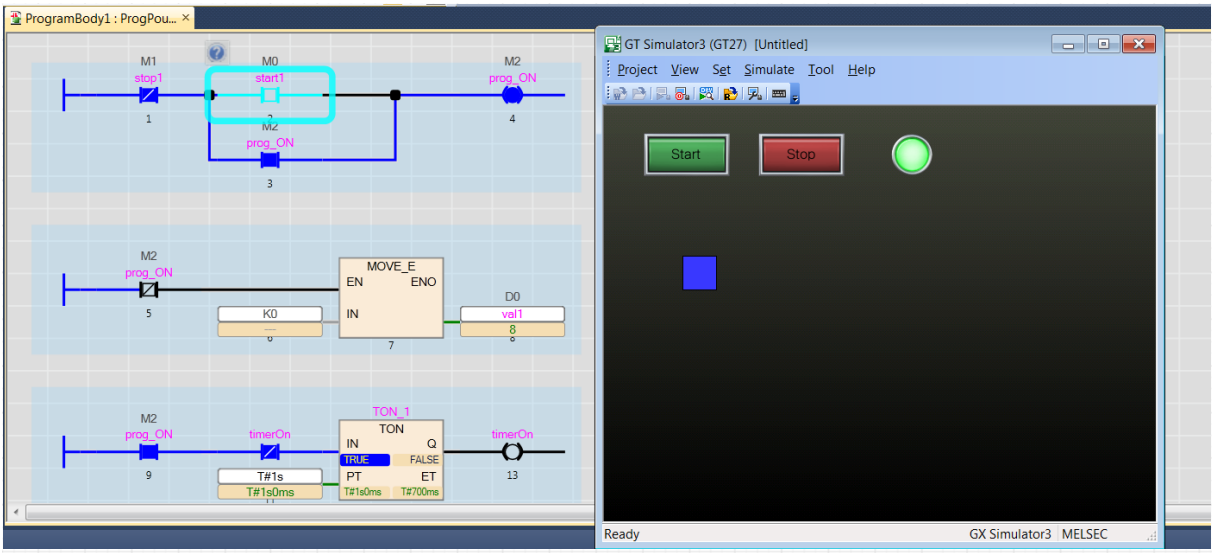


Obrázek 30: *Vytvoření nového objektu*



Obrázek 31: *Posun objektu*

případě bit vázaný na proměnnou *prog_ON*, tedy zařízení $M2$. Poté je třeba připojit zařízení, které deklaruje pohyb, v tomto případě se jedná o datový registr $D0$, ve kterém se integruje hodnota. Způsob pohybu je zvolen jako jednoduchý pohyb po přímce, který je omezen minimem a maximem. Celkové nastavení i se zmíněnými úpravami je na obrázku 31. Přesné souřadnice, ve kterých se na obrazovce má objekt pohybovat, se dále nastaví kurzorem myši v prostředí.



Obrázek 32: *Simulace vizualizace a programu*

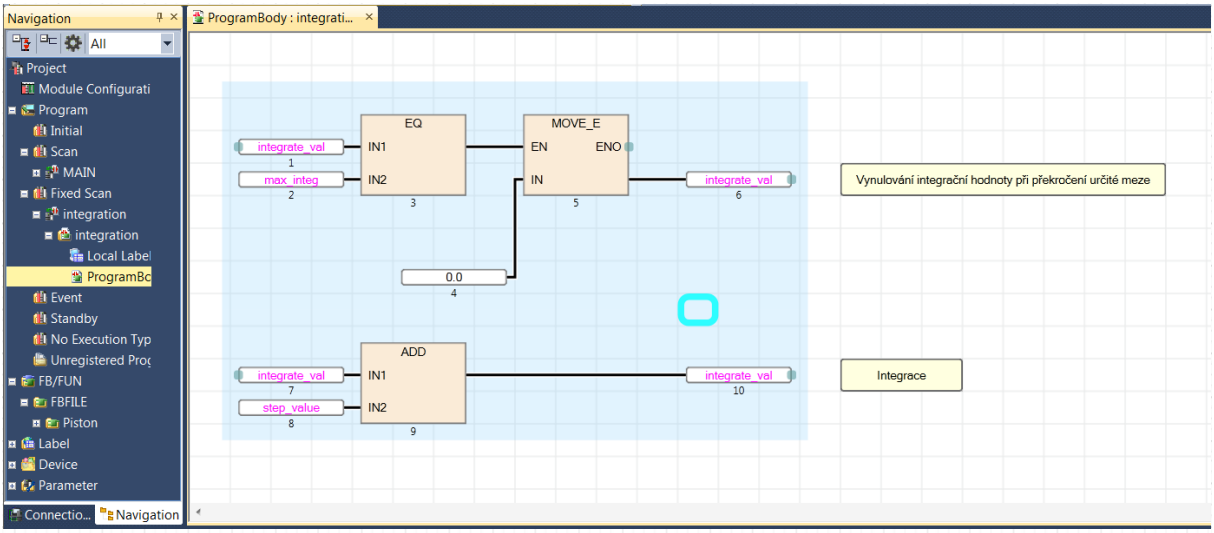
Pro ověření funkčnosti pomocí simulace je potřebné nejprve spustit simulaci programu v prostředí GX Works3. Po spuštění simulace programu, se v prostředí GT Designer zvolí přes kolonku **Tools** volba **Simulator Active**. Zkouška simulace je následně vyobrazena na obrázku 32.

5 Školící úlohy

Přínosem této bakalářské práce jsou především tři školící úlohy, kterým budou následující podkapitoly věnovány. Tyto úlohy byly vytvořeny takovým způsobem, aby napodobovaly reálné aplikace, ve kterých se PLC používají. Uživatel si může na úlohách vyzkoušet realizace různých algoritmů řízení. Úlohy byly navrženy modulárně, což znamená, že jejich části lze v programu využít opakovatelně. Cílem úloh je propojení dvou PLC, které mezi sebou budou přes digitální a analogové vstupy/výstupy komunikovat. Jedno PLC bude obsahovat simulaci úlohy a druhé řízení. Jedná se tak o HIL simulaci. Úlohy jsou psané tak, že je simulaci včetně řízení možné provádět i v rámci jediného PLC. Programové bloky simulace a řízení jsou propojeny přes vnitřní proměnné PLC.

5.1 Centrální diskrétní integrátor

Vytvořené modely jsou časově závislé dynamické systémy. Vytvořené funkční bloky tedy vyžadují na svém vstupu čas. Čas je realizován externě pomocí **centrálního diskrétního integrátoru**. Integrátor pracuje s konstantní vzorkovací periodou, a proto je řešený v programové části **Fixed scan**.

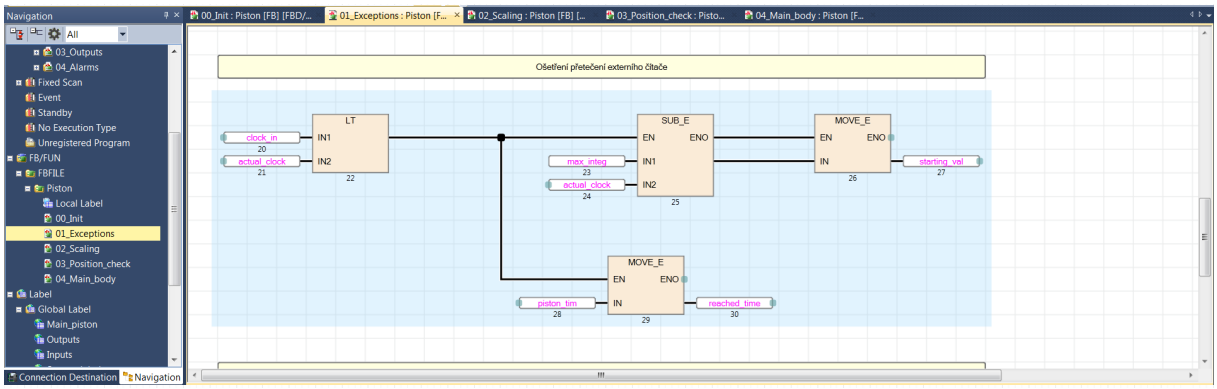


Obrázek 33: *Centrální čítač v části **Fixed Scan***

Z obrázku 33 je zřejmé, že se jedná o jednoduchý algoritmus čítání. Algoritmus inkrementuje proměnnou **integrate_val** o jedničku při každém scanu. Perioda spouštění je zde nastavena na 100ms, což tedy udává rychlost 10 jednotek za sekundu.

Integrační hodnota **integrate_val** je globální proměnná, která je vyvedena na vstup všech dynamických systémů. Tato hodnota má pevně danou maximální hodnotu, do které čítá. Po dosažení, je hodnota resetována na nulu. Každý funkční blok, na jehož vstup byla přivedena tato hodnota, musí obsahovat algoritmus ošetřující přetečení integrační hodnoty.

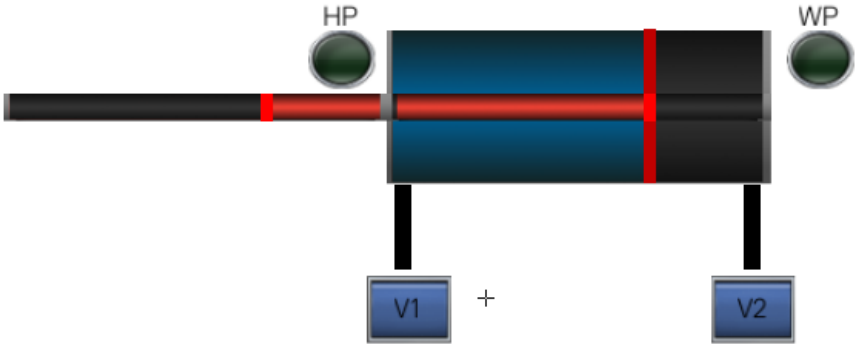
Ošetření je ve všech funkčních blocích realizováno analyticky jako na obrázku 34. Pro ukázku byl využit kód z funkčního bloku pístu, jehož aktuální poloha je dopočítávaná právě pomocí času. Program funguje tak, že porovná hodnotu integrátoru na vstupu s jeho předešlou hodnotou, která je uchována v proměnné **actual_clock**. Pokud je hodnota integrátoru menší než poslední dochovaná, je zaznamenáno přetečení. V případě přetečení je uložen dosažený časový okamžik pístu do pomocné proměnné a následně je dopočtená nová výchozí hodnota pístu. Výchozí hodnota v proměnné **starting_val** deklaruje počáteční hodnotu času, od které algoritmus počítá. Počáteční hodnotu času nelze pouze vynulovat, jelikož blok pracuje s větší periodou scanu než integrátor. Hodnota musí být dopočtena jako rozdíl maximální hodnoty integrátoru a poslední dochované hodnoty.



Obrázek 34: *Ošetření přetečení čítače*

5.2 Simulace pístu

Cílem této školící úlohy byla demonstrace základních bitových operací při práci s pístem. Samotný píst je navržen jako funkční blok. Detailní návrh tedy zahrnuje možnosti modifikace základních vlastností pístu jakožto jeho délku, rychlost a rozmístění senzorů. Návrh taktéž obsahuje možnosti zavedení chyb do simulace v podobě poruchy čidel nebo zaseknutého pístu. Pro představu je píst znázorněn na obrázku 35.



Obrázek 35: *Vizualizace pístu*

Funkční blok pístu obsahuje několik vstupů a výstupů. Mezi vstupy patří:

- **input_unit1_in**: Bitová hodnota, která definuje první z případných dvou vstupních ventilů. V případě aktivního bitu je na tento vstup přiveden vzduch.
- **input_unit2_in**: Bit definující druhý z vstupních ventilů. Tento vstup je funkční pouze v případě, kdy je zvolen dvojčinný píst.
- **DA_piston_in**: Tento vstup deklaruje, zda se jedná o dvojčinný či jednočinný píst. Jedná se opět o bitovou hodnotu, jednička udává dvojčinný píst, nula jednočinný.
- **piston_length_in**: Udává délku pístu, jedná se o proměnnou typu float.
- **piston_tempo_in**: Rychlost posunu pístu, která je taktéž datového typu float.
- **HP_sensor_error_in** a **WP_sensor_error_in**: Bity, pomocí kterých se aktivuje chyba senzoru v domovské resp. pracovní pozici. Chyba senzoru značí, že nedetekuje aktivitu při dosažení požadované polohy pístu.
- **piston_error_in**: Zaseknutí pístu v případě aktivní hodnoty na tomto vstupu.
- **pos_force_in**: Nastavení aktuální pozice pístu. Tato vstupní float hodnota je funkční pouze v případě zaseknutého pístu.
- **clock_in**: Na tento vstup musí být přiveden centrální diskrétní integrátor. Vstup slouží pro počítání časových okamžiků. Jedná se float hodnotu, jelikož je součástí několika matematických úprav v rámci funkčního bloku.

Mezi výstupy dále patří:

- **HP_reached_out** a **WP_reached_out**: Bity indikující aktivní čidla v domovské resp. pracovní pozici. Výstupy jsou aktivní, pokud se poloha pístu kryje s polohou čidel. Pokud jsou aktivovány chyby senzorů, výstupy jsou vždy neaktivní.
- **scaled_piston_position_out**: Float proměnná definující aktuální polohu pístu. Tato poloha je v pístu dopočtena pomocí času a zadané rychlosti.
- **max_reached_out** a **min_reached_out**: Bity, které indikují dorazy. Tyto výstupy jsou používané pouze pro testování.

Vstupní a výstupní hodnoty jsou deklarovány přímo ve funkčním bloku pouze v případě první úlohy. Pro druhou úlohu byly vytvořeny struktury, které zahrnují všechny potřebné vstupní a výstupní proměnné.

Program je hierarchicky rozčleněn do pěti programových částí:

1. **Inicializační část**: Zahrnuje nastavení výchozích hodnot při prvním scanu programu. Tento krok zaručuje, že program vždy začne za stejných počátečních podmínek.
2. **Ošetření vyjímek**: Slouží především k ošetření stavů, kdy dochází ke konfiguračním změnám (přepnutí na dvojčinný/jednočinný píst, zaseknutí pístu). Část dále zahrnuje ošetření přetečení centrálního integrátoru.

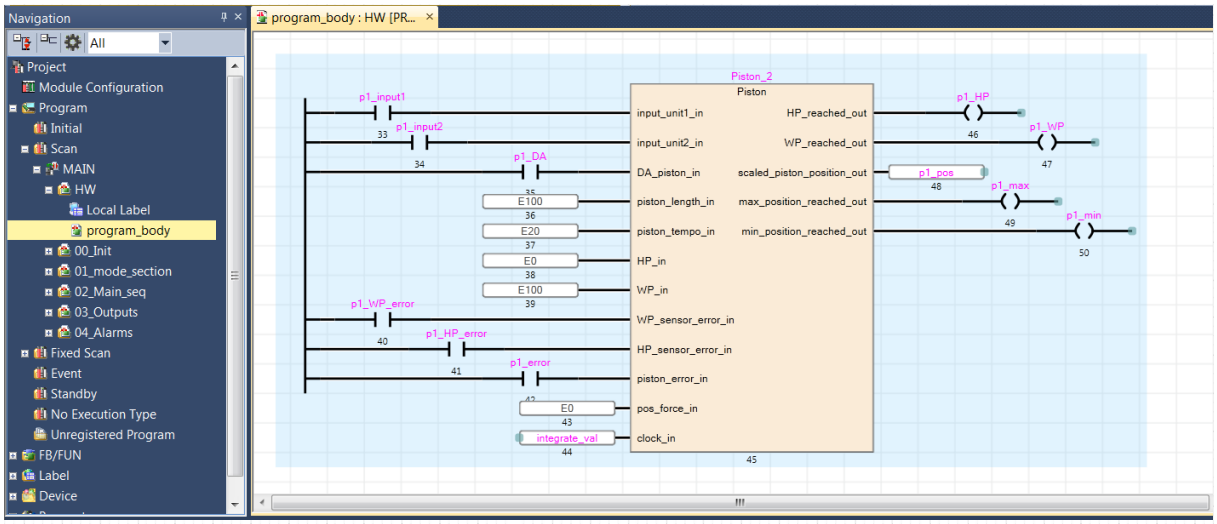
3. **Škálování**: Vypočte z dosaženého časového úseku a rychlosti aktuální polohu a provede patřičné škálování.

4. **Kontrola mezních poloh**: Slouží k detekci aktivity čidel. Pokud se poloha pístu překrývá s pásmem polohy čidla, je čidlo detekováno jako aktivní. Pásmo polohy čidla je definováno jako vstupní poloha čidla s dvou procentní tolerancí (oboustrannou).

5. **Hlavní programová část**: Obsluhuje výpočet časových úseků. Algoritmus funguje tak, že při detekci náběžné hrany na jakémkoliv vstupním ventilu pístu ukládá aktuální hodnotu času do proměnné. Uplynulý časový úsek je následně z aktuální a výchozí hodnoty času dopočten. Ze získaného časového úseku je ve třetí programové části vypočtena poloha pístu. Část programu též zahrnuje pevné nastavení polohy pístu v případě zaseknutí.

5.2.1 Návrh řízení

Tím, že je simulace napsaná do funkčního bloku, je možné tento blok použít v PLC vícekrát a sestavit tak řízení pro libovolný počet pístu. Pro demonstraci byla vytvořena sekvence pro tři písty. Parametry všech bloků byly nastaveny na stejné hodnoty a jsou znázorněny na obrázku 36. Parametry jsou vyvedené na jednotlivé vstupy pístu.



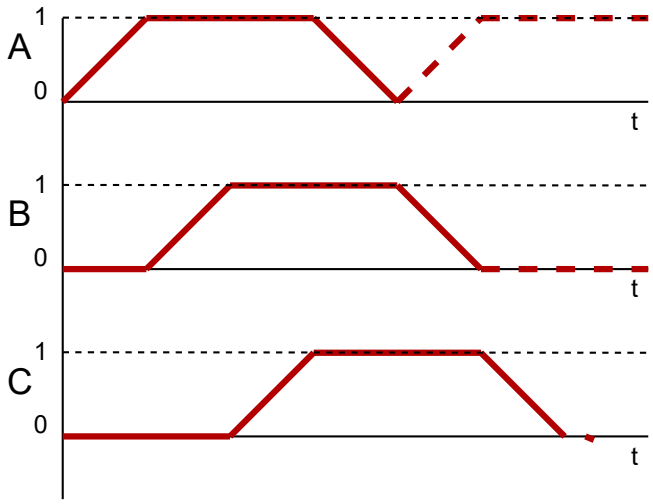
Obrázek 36: *Funkční blok pístu s parametry*

Řízení se skládá ze čtyř částí:

- **Přepínání mezi manuálem a automatem**: Algoritmus založený na principu start-stop obvodu, který přepíná mezi dvěma módy.
- **Hlavní sekvence**: Obsahuje hlavní sekvenci automatického módu.
- **Vyvedení výstupů**: Obsluha výstupů programu. Aktivace výstupů je podmíněna určitou kombinací. Kombinace deklaruje stav, ve kterém má být výstup aktivní. Příkladem může být aktivace ventilu prvního pístu za podmínky volby automatického módu a aktivního druhého kroku sekvence.

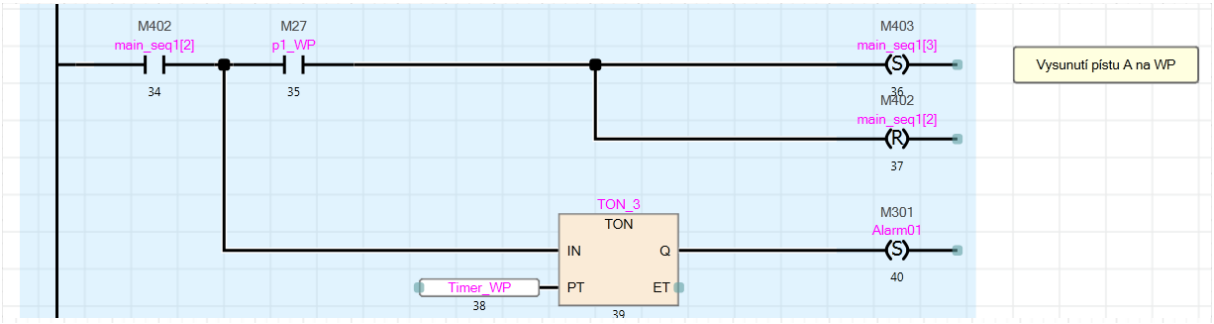
- **Alarmy**: Obsluha alarmů a alarmních stavů. Alarm v tomto případě může detekovat nefunkční čidlo nebo ventil.

Sekvence je realizována pomocí bitového pole. Každý bit pole představuje jeden krok. V daný moment může být aktivní pouze jeden krok sekvence. Pro demonstraci jednoduché sekvence bylo navrženo řízení tří pístů znázorněné na schématu 37.



Obrázek 37: *Sekvence řízení*

Ze schématu lze vyčíst chování 3 pístů (A, B, C) v závislosti mezi sebou. Nula značí zasunutý píst, jednička naopak píst vysunutý. Ze schématu je zřejmé, že písty se budou jeden po druhém vysouvat a následně zasouvat v nekonečném cyklu. Schéma lze přeložit do jednotlivých kroků sekvence. Tedy vysunutí pístu A bude prvním krokem sekvence, vysunutí pístu B druhým krokem a tak dále.



Obrázek 38: *Ukázka zapsané sekvence*

Příklad zápisu jednoho kroku sekvence je na obrázku 38. V tomto kroku dochází k vysunutí pístu A. Jakmile píst A dosáhne své pracovní pozice, je aktivován následující krok sekvence a aktuální krok je deaktivován. Program dále obsahuje časovač vyvedený před čidlem. Pokud není detekováno dosažení pracovní pozice pístu ve vymezeném času, je aktivován alarm. Tento alarm představuje chybu čidla nebo ventilu.

5.2.2 Vizualizace

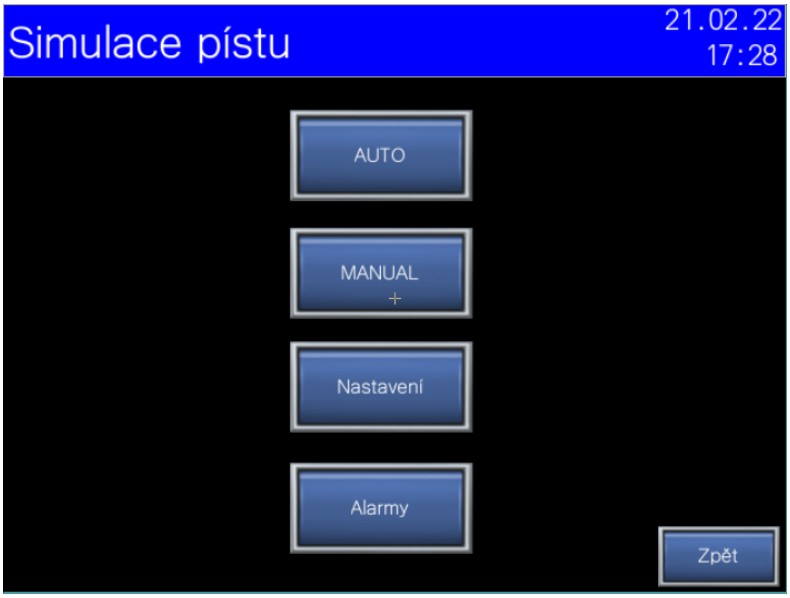
Pro demonstraci všech funkcí pístu byla vytvořena vizualizace znázorněna na obrázku 39. Vizualizace umožňuje uživateli si vyzkoušet základní funkce pístu.

Obrázek 39: *Vizualizace funkce pístu*

Dále je umožněno nastavit jednotlivé parametry pístu, to je znázorněno na obrázku 40. Uživateli je povoleno zanést poruchy do systému a sledovat, jak se bude měnit jeho chování.

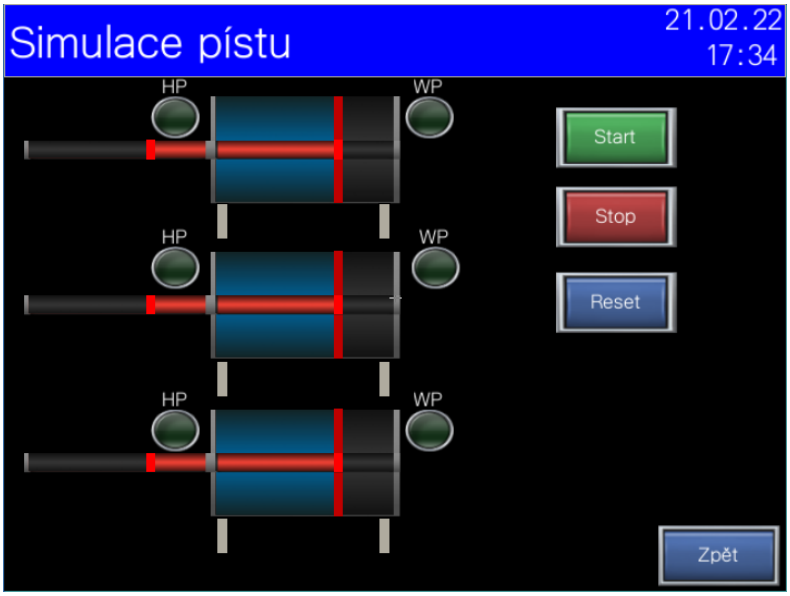
Obrázek 40: *Nastavení parametrů pístu*

Dále byla vytvořena vizualizace pro řízení. Na obrázku 41 je menu obsahující všechny možnosti vizualizace. Uživatel může přepínat mezi automatickým a manuálním módem v rámci panelu. Manuální mód slouží především k ověření funkce jednotlivých pístů a čidel. Menu dále nabízí možnost nastavení, ve kterém uživatel nalezne nastavení časovačů a poruch. Poslední část zahrnuje detailní informace ohledně alarmů, které v průběhu



Obrázek 41: *Hlavní menu vizualizace pístu*

programu mohou nastat. Vizualizace automatického módu na obrázku 42 umožňuje sledovat chod hlavní sekvence. Uživatel získává přehled, ve kterém kroku sekvence se program nachází.



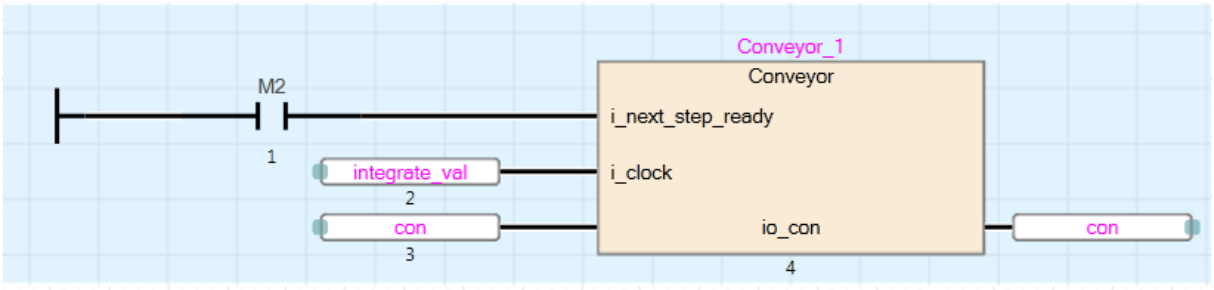
Obrázek 42: *Vizualizace hlavní sekvence*

5.3 Simulace dopravníku

Tato kapitola bude věnována modelu dopravníku, který byl vytvořen pro účely druhé školící úlohy. Je uvažován dopravník, po kterém může být přepraven určitý počet produktů naráz. Dopravník obsahuje několik čidel, které slouží k vyhodnocení polohy jednotlivých produktů na páse. Produkt je uvažován jako krychle o předem definované velikosti.

Model dopravníku vychází z modelu pístu, který byl rozšířen o několik nových funkcí. Dopravník umožňuje přesun jednotlivých objektů, čímž se původní model stává mnohem komplexnější, jelikož zahrnuje výpočet několika poloh zároveň.

Simulace je rozložena do dvou funkčních bloků. Hlavní funkční blok obsluhuje chování pásu jako celku, přičemž vnořený funkční blok simuluje chování individuálních produktů na páse. Model si předává mezi bloky velké množství proměnných, a proto byly vytvořeny struktury. Ty jsou na bloky vyvedené jako vstupně/výstupní proměnné. Ukázka funkčního bloku je na obrázku 43, kde je na vstup a výstup přivedena struktura **con**.



Obrázek 43: *Funkční blok dopravníku*

Blok obsahuje dvě vstupní proměnné, které nebyly zahrnuty v rámci struktury:

- **i_next_step_ready**: V případě aktivní bitové hodnoty na tomto vstupu bude resetována poloha produktu při dosažení koncové polohy dopravníku.
- **i_clock**: Vstup slouží pro centrální diskrétní integrátor.

Mezi vstupy zahrnuté dále ve struktuře **con** patří:

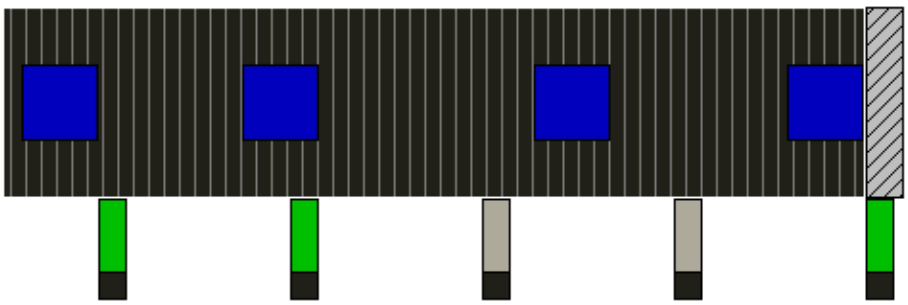
- **input_on**: Bit, který spouští pohyb dopravníku.
- **new_product_pulse**: Při detekci náběžné hrany na tomto bitu je přidán nový prvek na pás.
- **con_length**: Délka dopravníku (float).
- **con_speed**: Rychlost pohybu dopravníku (float).
- **product_length**: Délka produktů na páse (float). V simulaci se předpokládá, že produkty mají tvar krychle.
- **max_sensors**: Počet čidel na páse (word), který je omezen na deset čidel.

- **max_products**: Maximální počet produktů, které mohou být na páse (word). Toto číslo je též omezeno na deset. Obě dvě hodnoty jsou datového typu word (integer), jelikož jsou využity pro indexaci polí.
- **sensors_pos**: Pole float hodnot, ve kterém jsou uloženy polohy jednotlivých čidel. Délku pole nelze deklarovat v PLC dynamicky, proto je stanovená na výchozí hodnotu deset. Maximálně tak může být na páse deset senzorů polohy. Pokud je čidel méně než maximální počet, program využívá jen část pole.

Mezi výstupy ve struktuře **con** se řadí:

- **active_sensors**: Bitové pole, které je též omezeno maximálním počtem prvků. Pole může obsahovat až deset bitových hodnot. V těch je uchovaná informace o aktivitě jednotlivých čidel. Čidlo je aktivní v případě, že se jeho poloha kryje s polohou některého produktu.
- **products_positions**: Float pole, ve kterém je zaznamenána poloha jednotlivých produktů. Platí stejná omezení jako u pole čidel. Tato hodnota je výstupní jen kvůli vizualizaci.
- **output_ready**: Bitová hodnota, která dává informaci o tom, že některý z produktů na páse dosáhl konečné polohy. Hodnota je na výstupu především pro další využití při sestavování linky z jednotlivých funkčních bloků.
- **products_active**: Bitové pole, které zaznamenává aktivitu produktů. Pokud je produkt s konkrétním indexem na páse, v poli bude na stejném indexu aktivní bitová hodnota. Tato proměnná je vyvedená na výstup jen pro vizualizační potřeby.

Pro pochopení jednotlivých parametrů a lepší představu ohledně funkce je dopravník vykreslen na obrázku 44. Tato realizace umožňuje maximálně pět produktů na páse a obsahuje pět čidel. Aktivní čidlo je znázorněno zelenou barvou, neaktivní šedou. Zábrana na konci značí, že vstup **i_next_step_ready** není aktivní. Produkt tak není resetován po dosažení koncové polohy, je pouze zastaven.



Obrázek 44: *Vizualizace dopravníku*

Vnitřní logika funkčního bloku je podstatně složitější než tomu bylo u pístu. V rámci funkčního bloku dopravníku je zahrnut ještě blok produktu. Každý produkt na dopravníku má svou vlastní strukturu. Ta musí obsahovat i část proměnných převzatých ze

struktury dopravníku. Všechny struktury produktů jsou zahrnuty v rámci jednoho pole struktur. Ke konkrétnímu produktu se přistupuje přes index, který mu byl přiřazen. Mezi významné vstupy/výstupy ve struktuře se řadí:

- **index**: Index konkrétního produktu.
- **active**: Bit, který dává informaci o tom, zda je produkt na páse.
- **done**: Indikace dosažení maximální hodnoty produktu.
- **reset_product**: Bitová hodnota, která definuje, zda může být produkt resetován při dosažení koncové polohy.
- **product_pos**: Aktuální poloha produktu.

Nejsou uvedeny všechny proměnné uvnitř struktury, jelikož většina z nich jsou převzata od struktury dopravníku. Jedná se například o rychlost pásu, polohu čidel, délku dopravníku apod. Program bloku produktu vychází z programu pístu, tudíž obsahuje dále stejné proměnné pro uchování časových okamžiků jako píst.

Program dopravníku je složen ze čtyř programových částí:

- **Inicializační část**: Reset důležitých proměnných při startu programu, nulování indexů a nastavení parametrů.
- **Přidávání produktů na dopravník**: Při náběžné hraně na vstupu je produkt na příslušném indexu aktivován. Toto se provede pouze v případě, pokud není produkt již aktivní.
- **Hlavní část**: Obsahuje dvě hlavní smyčky. V první smyčce jsou všechny struktury produktů z pole předány jako vstupně/výstupní parametr funkčnímu bloku produktu. Do struktur jsou následně uloženy informace o vzájemných polohách produktů. Jednotlivé polohy jsou načteny ze struktur produktů do celkové struktury dopravníku. Dále je vyhodnoceno, zda není nutné provést resetování poloh některého produktu. Druhá smyčka řeší vzájemné dorazy bloků. Tato smyčka zařídí, že se jeden blok zastaví o druhý.
- **Vyhodnocení čidel**: Každá struktura produktu obsahuje informaci o aktivních čidlech. Tento programový blok obsahuje smyčku, která tyto informace sjednotí do jednoho pole. Toto pole je pak vyvedeno na výstup (**active_sensors**).

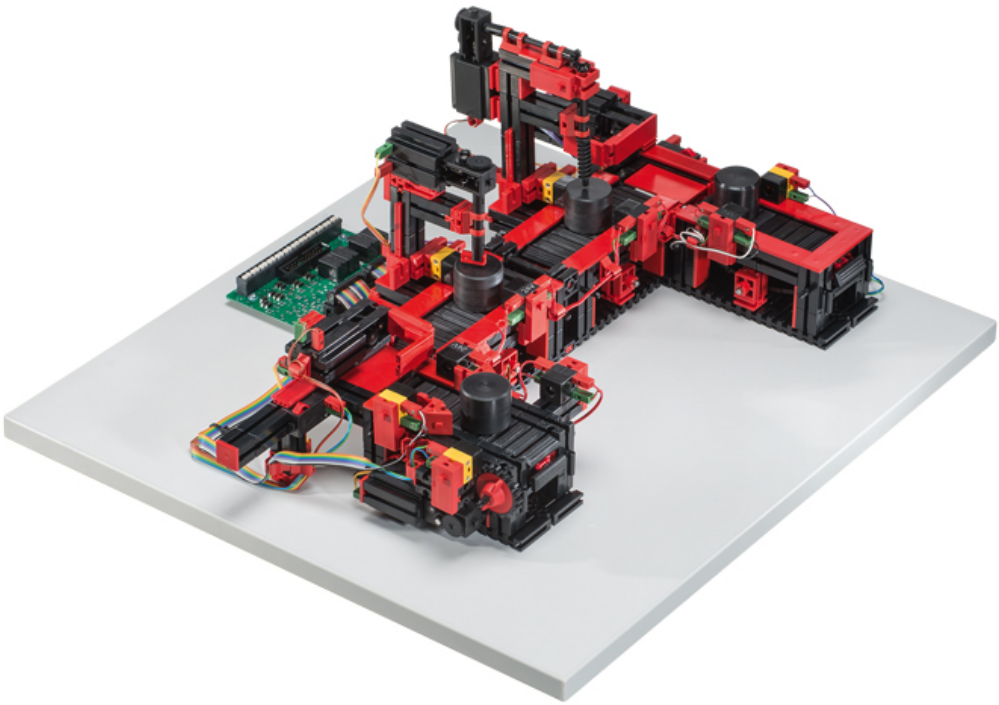
Program funkčního bloku produktu se skládá z následujících částí:

- **Ošetření přetečení**: Program vychází z programu pístu, algoritmus je opět postaven na počítání časových okamžiků. Program tedy musí obsahovat i ošetření přetečení.
- **Kontrola pozice**: Část sloužící ke kontrole polohy aktuálního produktu. Pokud objekt dosáhl maximální povolené vzdálenosti, jsou aktivovány příslušné výstupy. Dále je v této části prováděno resetování polohy výrobku.

- **Hlavní část**: Funkční blok produktu si neuchovává informace ohledně poloh ostatních produktů. V programu slouží jen jako funkce, na jejíž vstup a výstup je přivedena struktura požadovaného produktu. První tedy program v hlavní části musí načíst poslední dosažený čas. Následně dochází k počítání časových okamžiků stejně tak, jako tomu je u pístu.
- **Vyhodnocení čidel**: V této části dochází k testům, zda se poloha konkrétního produktu nekryje s polohou některého čidla. Pokud dochází k překrývání, jsou aktivované příslušné výstupy. Vyhodnocení čidel každého produktu je následně sjednoceno v programu dopravníku.

5.4 Simulace Linky

Tato kapitola bude věnována druhé školící úloze. V rámci této úlohy byla vytvořena simulace reálného modelu, který je na obrázku 45. Model se skládá ze čtyř dopravníků a dvou pístů. Pro tuto simulaci byly tedy použity již vytvořené modely dopravníku a pístu. Ty bylo nutné rozšířit tak, aby bylo možné propojit jednotlivé prvky a zajistit všechny požadované funkce linky. Tento fakt dokumentuje modularitu a znovupoužitelnost již vytvořených programových částí. Je nutné dodat, že reálný model linky byl již vytvořen a sloužil pouze jako předloha pro simulaci.

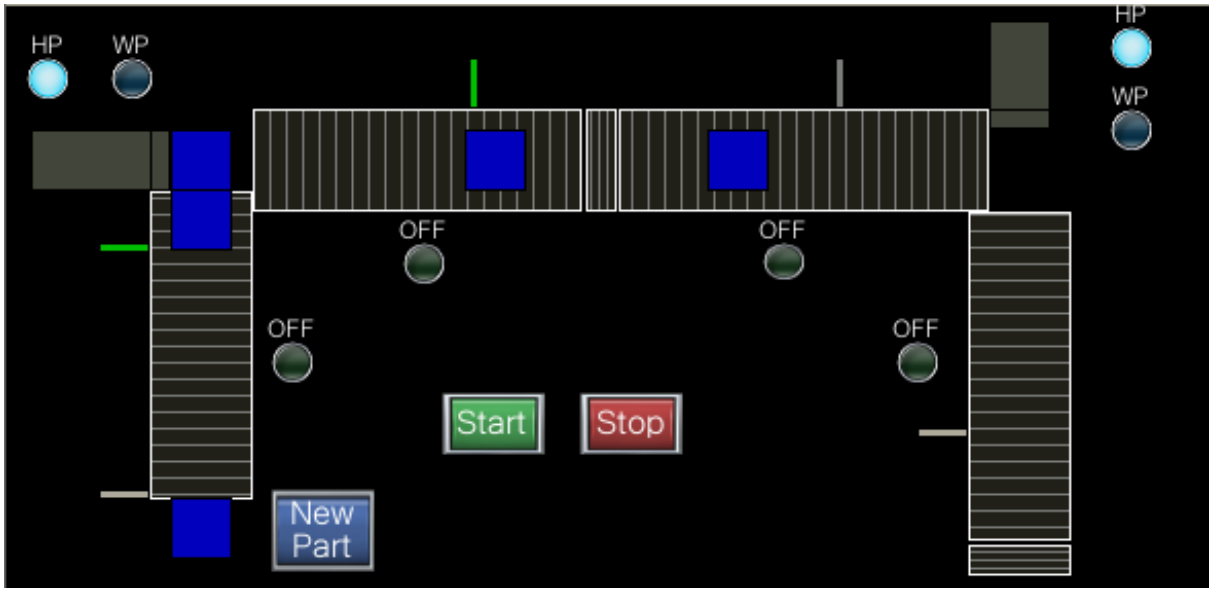


Obrázek 45: *Reálný model linky*

Funkce linky spočívá v tom, že je na první dopravník vložen produkt. Ten je převezen

před první píst, který jej nasune na druhý dopravník. Na druhém a třetím dopravníku jsou umístěny nástroje, kterými je možné produkty upravit (navrtání, otočení). Na těchto pásech jsou umístěny čidla u nástrojů. Po dokončení úprav a přesunu výrobku před druhý píst, je produkt pístem dopraven na poslední dopravník.

Vytvořená vizualizace modelu linky na obrázku 46 byla vytvořena tedy dle reálné předlohy. Model obsahuje oproti reálnému modelu určitá zjednodušení. Ty se týkají přechodů mezi částmi a omezení počtu prvků na pásech.



Obrázek 46: *Model linky*

Jednotlivé modely pístu a dopravníku musely být rozšířeny z důvodu návaznosti na sebe. Vstupy a výstupy pístu byly po vzoru dopravníku přesunuty do struktury. Ta byla dále rozšířena o dvě proměnné:

- **product_in**: Bitová hodnota, která detekuje, že se produkt nachází před pístem.
- **reset**: Proměnná, při jejíž aktivaci je resetován celý systém.

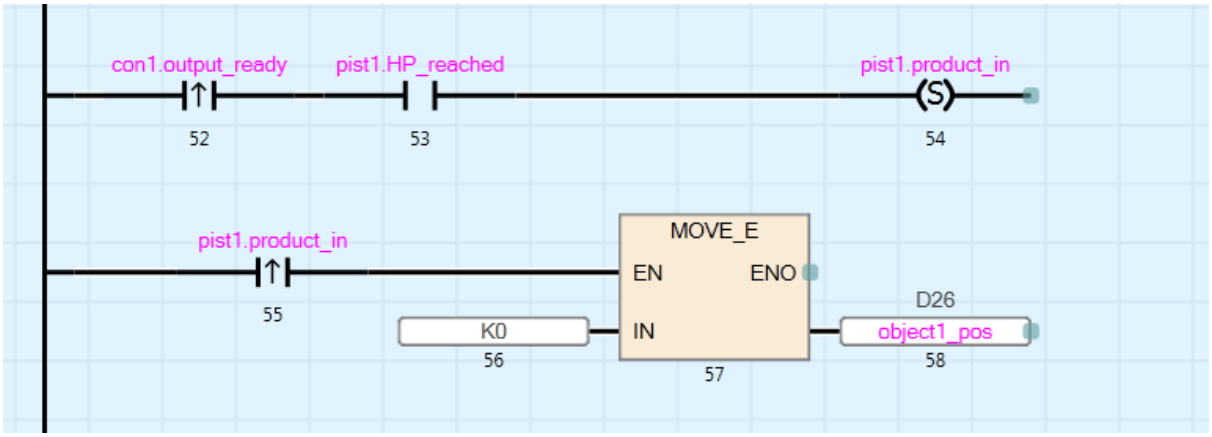
Rozšířena byla i struktura dopravníku:

- **product_done**: Bit, který detekuje, že jeden z produktů opustil dopravník.
- **product_at_start**: Hodnota, která dává informaci o tom, že je produkt na začátku dopravníku. Tato bitová hodnota je aktivní do doby, dokud není produkt vzdálen od startu tak, aby se na pás vešel další produkt.
- **all_products_active**: Signalizace, že na páse je maximální počet povolených produktů.
- **reset**: Proměnná pro resetování celého systému.
- **adjust_product**: Bitový vstup pro spuštění úpravy produktu pomocí příslušného nástroje.

Programy obou modelů jsou pozměněny pouze danými vstupy a výstupy. Do programů byla přidána funkce resetování modelu. Toto rozšíření slouží především k ladění a zachycení chyb. Dále byly přidány indikace polohy produktu pro plynulý přechod mezi jednotlivými prvky simulace. V případě dopravníku byl ještě přidán vstup pro spuštění nástroje.

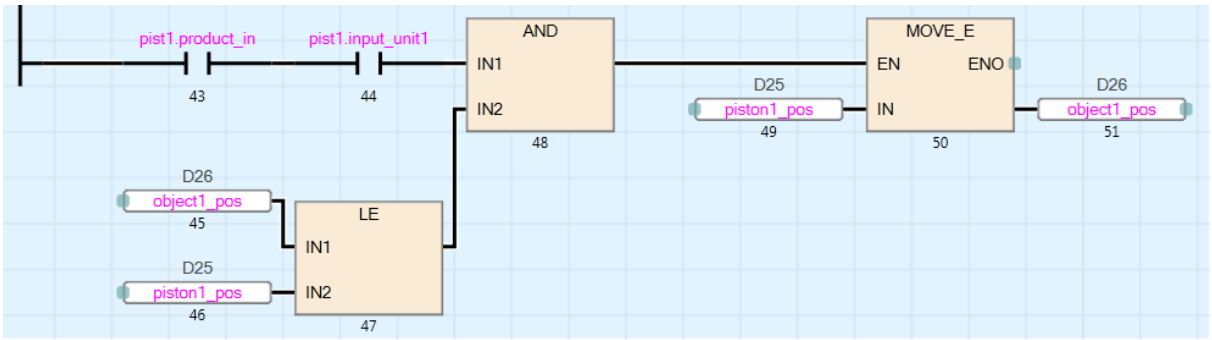
Simulace je složena z jednotlivých modelů, které jsou vzájemně propojeny. V případě přechodů mezi částmi nestačí pouhé propojení. Ty musí být vyřešeny vně funkčních bloků. To se týká pouze přechodů mezi dopravníky a písty. Přesun mezi dvěma pásy je realizován pomocí propojení vstupů a výstupů funkčních bloků.

Pro přechod mezi dopravníkem a pístem musela být zavedena nová proměnná, která zaznamenává aktuální polohu produktu. Na obrázku 47 je první část kódu pro přechod. Produkt je před pístem detekován, pokud produkt opustil dopravník (**output_ready**) a píst je ve své domovské pozici. Pokud došlo k detekci produktu před pístem, je jeho poloha nulována.



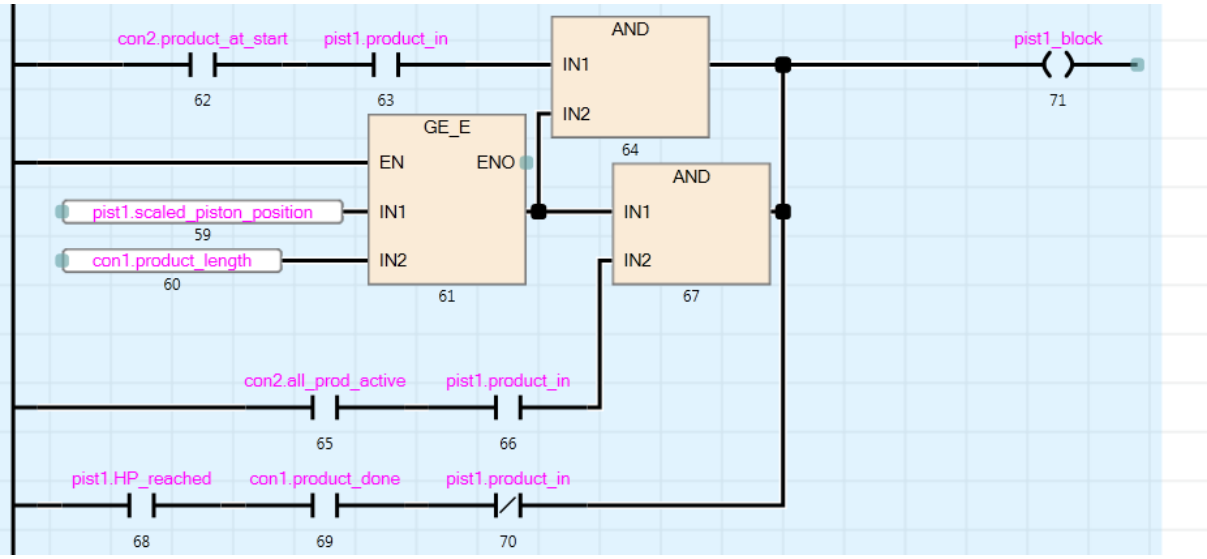
Obrázek 47: *Detekce produktu před pístem a reset polohy*

Z obrázku 48 plyne, že pohyb produktu kopíruje pohyb pístu. Pokud je detekován produkt před pístem (**product_in**) a je spuštěn přívod vzduchu, tak dochází k sunutí produktu. K posunu nedochází v případě, pokud není aktuální poloha produktu a pístu stejná.



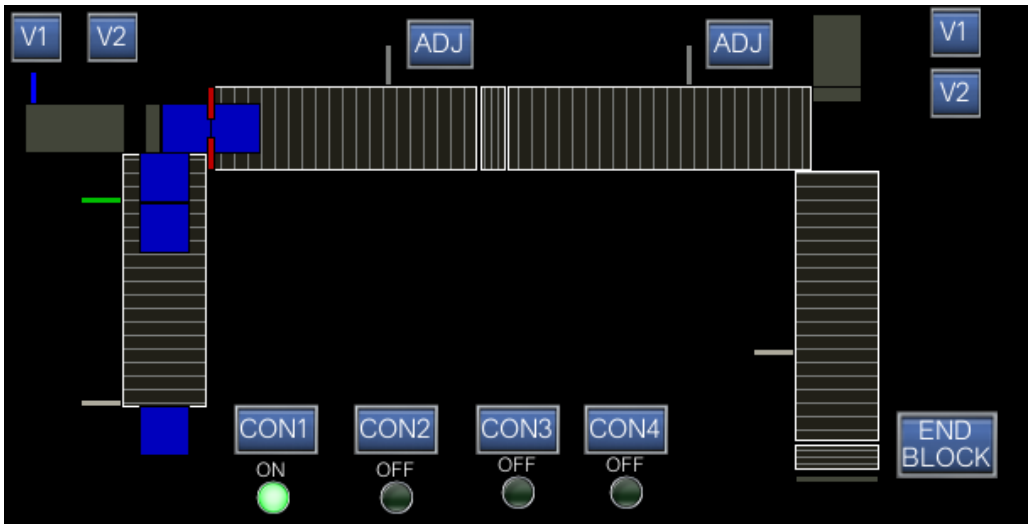
Obrázek 48: *Pohyb produktu před pístem*

Problém nastává v momentě, kdy je na následujícím dopravníku produkt. Zde bylo provedeno první zjednodušení simulace oproti reálnému systému. Pokud je na následujícím dopravníku produkt na startu a dochází k dorazu, aktivuje se závora. Tuto funkci zajišťují první dvě větve na obrázku 49. Třetí větev aktivuje závoru, pokud je následující dopravník plný. Poslední větev je dalším zjednodušením pro simulaci. Kód ošetřuje moment, kdy produkt opustil první dopravník, ale ještě není přesně před pístem. V tento moment není možné hnout s pístem.



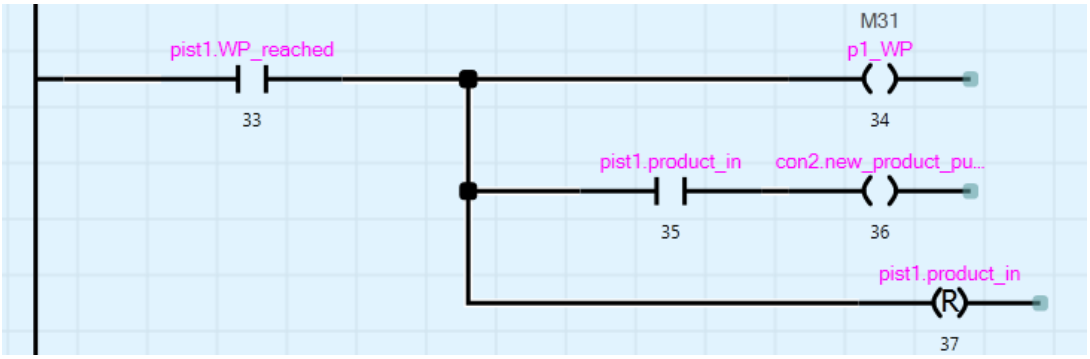
Obrázek 49: *Blokování pístu*

Pro lepší pochopení je na obrázku 50 znázorněn stav, kdy je aktivována závora na druhém dopravníku. V tomto případě se jedná o stav, kdy je na následujícím dopravníku produkt na počáteční pozici, viz první dvě větve na obrázku 49. Zjednodušení spočívá v tom, že místo toho, aby byl produkt na začátku dopravníku posunut produktem před pístem, je aktivována závora. Ta zapříčiní zamezení pohybu pístu do doby, dokud druhý produkt neopustí počáteční pozici na dopravníku.



Obrázek 50: *Stav aktivace závory*

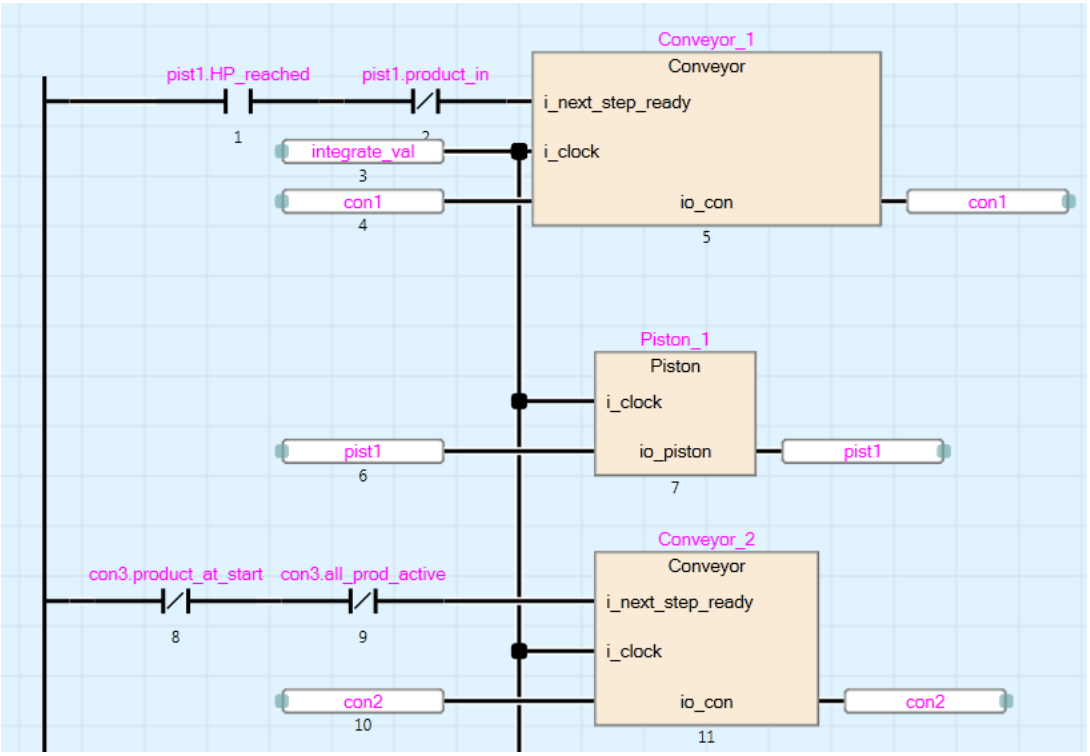
Předání produktu následujícímu dopravníku je realizováno na obrázku 51. Pokud píst dosáhne své pracovní pozice, dochází k přesunu produktu na následující prvek. Zároveň dochází k nulování proměnné, která detekuje produkt před pístem.



Obrázek 51: *Předání produktu*

Dalším rozšířením je následně indikace jednotlivých úprav na produktech. Ty jsou realizovány bitovými poli, které zaznamenávají úpravy. Po tom, co produkt opustí linku, jsou vypsány všechny provedené úpravy ve vizualizaci.

Všechny další funkce linky jsou zahrnuty v jednotlivých modelech, které jsou mezi sebou propojeny skrz proměnné. Ukázka propojení několika funkčních bloků je na obrázku 52. Pro správné fungování simulace musí být vyveden centrální diskrétní integrátor do všech funkčních bloků.



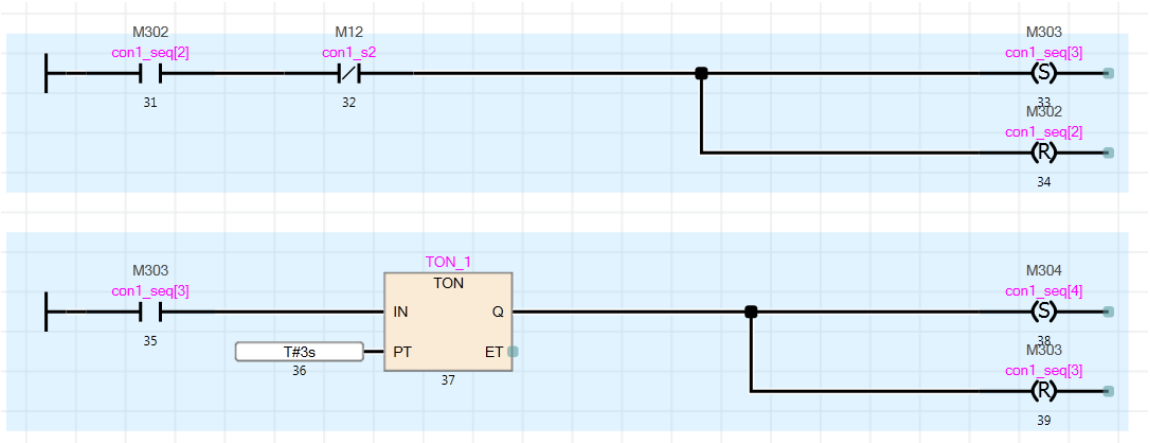
Obrázek 52: *Propojení funkčních bloků*

5.4.1 Návrh řízení

Navržené řízení modelu zahrnuje ovládání linky jako celku. Na lince se v jeden okamžik pohybuje několik produktů zároveň. Program řízení se skládá opět ze čtyř hlavních částí:

- **Přepínání manuál/automat**: Manuální mód byl v tomto případě vytvořen především pro ladění chyb, které mohly během chodu nastat. Přepínání funguje na stejném principu jako tomu bylo u pístů, viz kapitola 5.2.1. Program byl doplněn o resetování systému v případě přepnutí. Obnovení výchozího stavu bylo potřebné pro algoritmus automatického módu, který je založen na počítání produktů.
- **Hlavní sekvence programu**: Program pro hlavní sekvenci automatického módu.
- **Zapojení výstupů**: Aktivace jednotlivých výstupů. Jednotlivé výstupy jsou aktivovány například při automatickém módu při dosažení požadovaného kroku hlavní sekvence.
- **Alarmy**: Obsluha alarmů. Ty v případě linky značí poruchu dopravníku, čidel nebo pístů. Poruchy je možné do modelu zanést opět uměle.

Algoritmus hlavní sekvence je založen na počítání produktů. Pro každý dopravník je deklarována proměnná, která uchovává aktuální počet produktů vyskytujících se na páse. Na každém páse nesmí být překročen maximální počet produktů, v tomto případě se jedná o maximálně tři produkty. Druhý a třetí dopravník je řízen současně pro zjednodušení programu řízení. Na nich jsou zároveň prováděny sekvence úprav produktů (navrtání, otočení). Kromě produktů je v algoritmu počítán i čas. Konkrétní příklad je uveden na obrázku 53. Jakmile je produkt za druhým čidlem na prvním dopravníku je aktivován následující krok sekvence. V něm program čeká vymezený čas, než produkt opustí dopravník a přesune se před píst. Na analytickém principu je založen celý program řízení.



Obrázek 53: *Ukázkové kroky hlavní sekvence řízení linky*

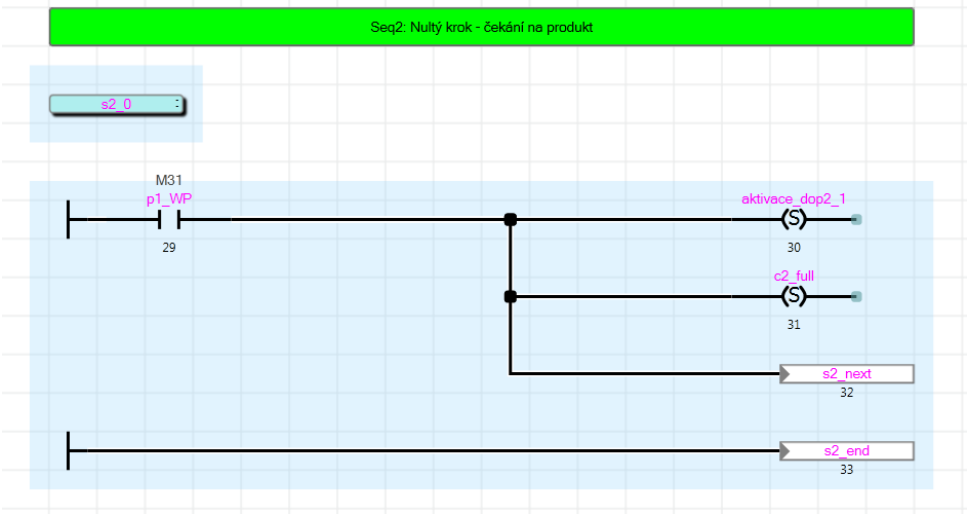
Z prvního řádku programu na obrázku 53 je zřejmé, že čidlo je aktivní v případě nulové hodnoty (**NO** kontakt). Čidla v reálném modelu jsou totiž realizována pomocí fotorezistoru, který je osvětlován protější světelnou diodou (LED). Pokud se mezi LED a fotorezistor dostane produkt, rezistor není osvícen a bitová hodnota signálu je nula.

Ačkoliv je řízení založené na počítání produktů funkční, není příliš časově efektivní. Problém s neefektivitou nastává zejména tehdy, když je na jednom dopravníku více produktů. To je způsobeno absencí více čidel, které by umožnily sledovat podrobněji aktuální stav polohy jednotlivých produktů. Tento problém s nedostupností informace lze vyřešit zjednodušením řízení. To spočívá v tom, že je omezen maximální počet produktů na jednom páse na jeden. Díky tomu je možné využít odlišný přístup návrhu řízení než pomocí bitové sekvence. Při tomto zjednodušení lze využít sekvenci pomocí tzv. **podmíněných skoků**.

Podmíněné skoky fungují tak, že na začátku programové části je vyhodnoceno, v jakém stavu se sekvence nachází. Poté program přeskočí do části, která tento stav ošetřuje. V případě tohoto přístupu se tak program nevykonává celý v každém cyklu, ale vykonávají se jen jeho příslušné části. Výhodou je tedy rychlejší čas scanu PLC. Oproti bitovým sekvencím také není problém přidat libovolný mezikrok. Nevýhodou avšak může být složitější zápis a nutnost řešit časovače a náběžné hrany mimo hlavní sekvenci.

Ukázky sekvence jsou na obrázcích 54 a 55. Jak bylo výše popsáno, na začátku programu je nejprve vyhodnocen stav. Podle stavu se pomocí funkce **JMP** skočí na programovou část začínající příslušným návěstím. Přechody mezi jednotlivými kroky jsou realizovány pomocí inkrementace proměnné v případě splnění přechodové podmínky. Pokud podmínka není splněna, je přes funkci JMP přeskočeno na konec programové části.

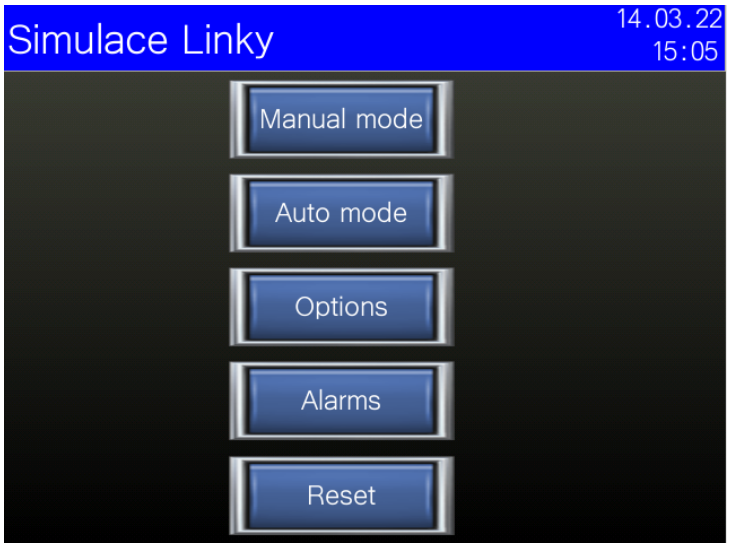
Obrázek 54: *Sekvence pomocí podmíněných kroků - vyhodnocení stavů*



Obrázek 55: *Sekvence pomocí podmíněných kroků - konkrétní krok sekvence*

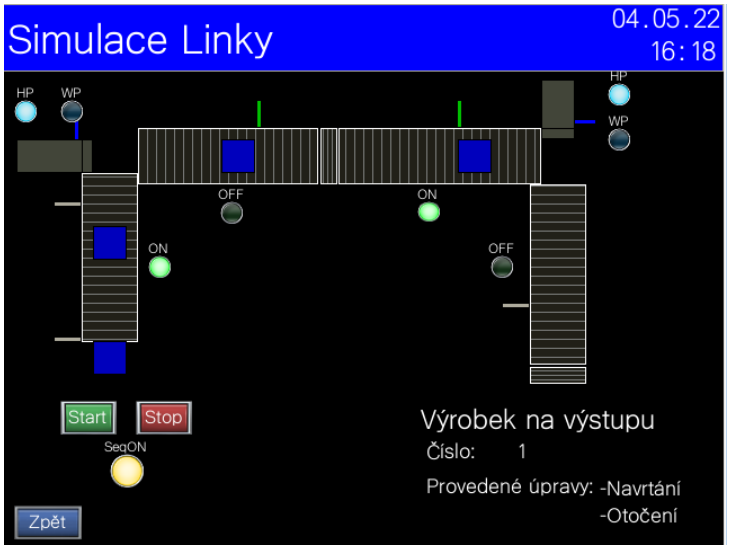
5.4.2 Vizualizace

Nezbytnou součástí vytvořeného modelu je vizualizace celé linky. V rámci vizualizace bylo vytvořeno několik funkcí, které jsou na obrázku 56. Je umožněno ovládat linku pomocí manuálního či automatického módu. Dále je pomocí nastavení možné zanést do modelu chyby (chyby čidel, zaseknutí pístu apod.). Uživatel si následně může prohlédnout detailní informace ohledně alarmů. Poslední tlačítko umožňuje resetovat celý systém a uvést jej do výchozí pozice. Tato možnost je přidána především pro testování.



Obrázek 56: *Vizualizace linky - hlavní stránka*

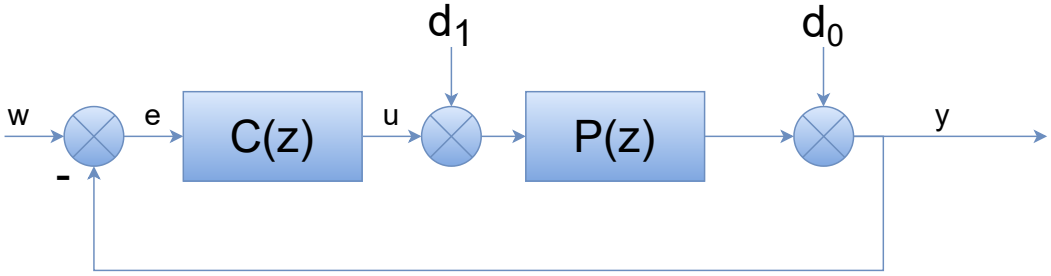
Vizualizace linky v automatickém módu je následně zobrazena na obrázku 57. Po přesunu výrobku na konec, je v pravém dolním rohu zaznamenáno, jaké úpravy na něm byly provedeny.



Obrázek 57: *Vizualizace linky - automatický mód*

5.5 Simulace regulace

Poslední školící úloha byla zaměřena na diskrétní regulaci dynamického systému. V rámci simulace byla vytvořena regulační smyčka, která se skládá z diskrétního systému **P(z)** a regulátoru **C(z)**. Do smyčky lze zanést vstupní (d_1) i výstupní (d_0) poruchu, viz obrázek 58. Uživatel může zvolit libovolný systém druhého řádu, který lze řídit P, I, PI nebo PID regulátorem. Pro účely školení byl zvolen systém a k němu byly vytvořené jednotlivé regulátory pro demonstraci funkčnosti regulační smyčky. Vývojové prostředí GX Works3 a GT Designer3 nedisponují nástroji pro simulaci jako například Matlab/Simulink, a proto bylo nutné zvolit dostatečně pomalý systém pro sledování vývoje systému v čase.



Obrázek 58: *Regulační smyčka*

Simulace předpokládá diskrétní systém druhého řádu ve tvaru přenosové funkce:

$$P(z) = \frac{d_1 z + d_0}{z^2 + a_1 z + a_0},$$

kde parametry d_1, d_0, a_1, a_0 jsou libovolné a mohou být nastaveny dle potřeby uživatele. Pro demonstraci regulací byl zvolen dostatečně pomalý systém ve tvaru:

$$P(z) = \frac{0.01244z + 0.01113}{z^2 - 1.693z + 0.7165}.$$

Systém musí být následně pro účely výpočtů převeden do Frobeniovy stavové reprezentace:

$$A_F = \begin{bmatrix} 0 & 1 \\ -a_0 & -a_1 \end{bmatrix}, B_F = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, C_F = \begin{bmatrix} d_0 & d_1 \end{bmatrix},$$

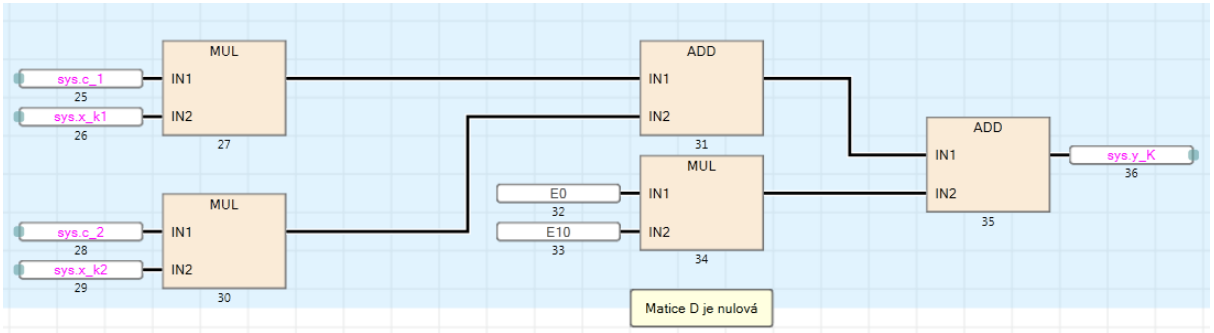
s přímým působením vstupu na výstup není v simulaci počítáno, matice D je tedy nulová [17]. Systém ve stavové reprezentaci má následující tvar:

$$x(k+1) = \begin{bmatrix} 0 & 1 \\ -a_0 & -a_1 \end{bmatrix} x(k) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(k),$$

$$y(k) = \begin{bmatrix} d_0 & d_1 \end{bmatrix} x(k+1).$$

Systém je převeden do stavové reprezentace, aby bylo možné v PLC vypočítat v každém kroku stav a výstupní hodnotu. Regulační smyčka vyžaduje konstantní vzorkovací periodu, proto byl celý program realizovaný v programové části **Fixed scan**.

Vzorkovací perioda je nastavena na $T = 0.5s$. Ukázka výpočtu výstupu systému pomocí výstupní rovnice stavové reprezentace je na obrázku 59.

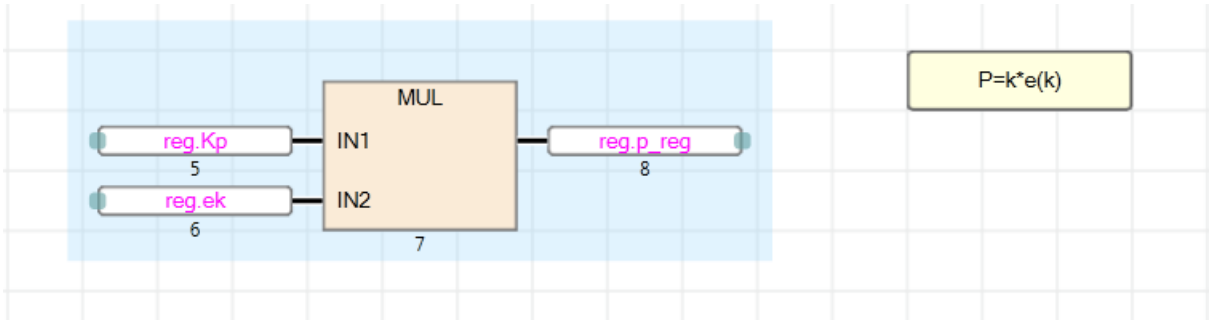


Obrázek 59: *Výstupní rovnice stavové reprezentace v PLC*

Regulátor je v simulaci složen z proporcionální (P), integrační (I) a derivační (D) složky. Podle volby typu regulátoru se následně sečtou příslušné složky. Proporcionální složka je počítaná dle vzorce:

$$u_p(k) = Ke_p(k),$$

kde K je proporcionální zesílení a $e(k) = y(k) - w(k)$ je regulační odchylka. Zvyšováním zesílení K je možné zlepšovat přesnost regulace a rychlost odezvy. Na druhou stranu se zvětšuje přeregulování a snižuje se robustnost ve stabilitě. Výpočet proporcionální složky v PLC je znázorněna na obrázku 60.



Obrázek 60: *Realizace proporcionální složky regulátoru v PLC*

Integrační složka je získaná ze vzorce:

$$u_i(k) = u_i(k-1) + K_i\frac{T}{2} \cdot [e_i(k-1) + e_i(k)],$$

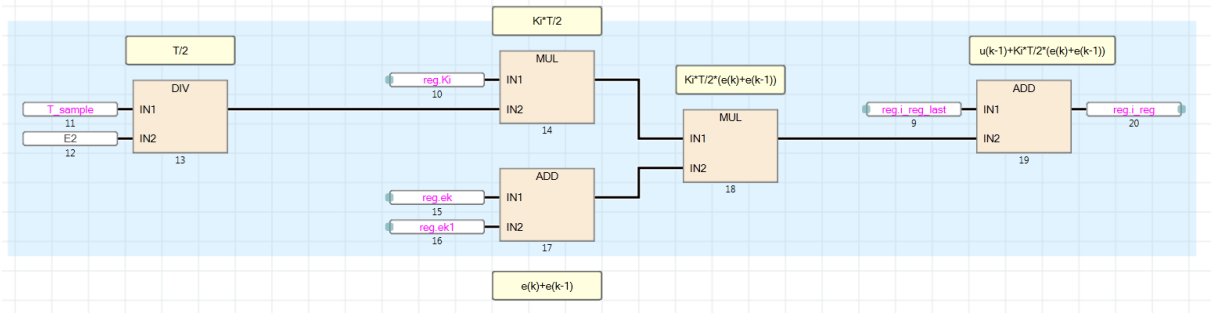
kde $K_i = \frac{K}{T_i}$ je integrační zesílení a T je vzorkovací perioda systému. Integrační složka je potřebná pro dosažení přesnosti regulace. To znamená, že $e(k)$ se blíží k nule v ustáleném stavu. Na druhou stranu zanáší do smyčky fázové zpoždění, snižuje robustnost ve stabilitě a zpomaluje rychlost odezvy.

Derivační složka je doplněna o filtrovanou (aproximativní) derivaci:

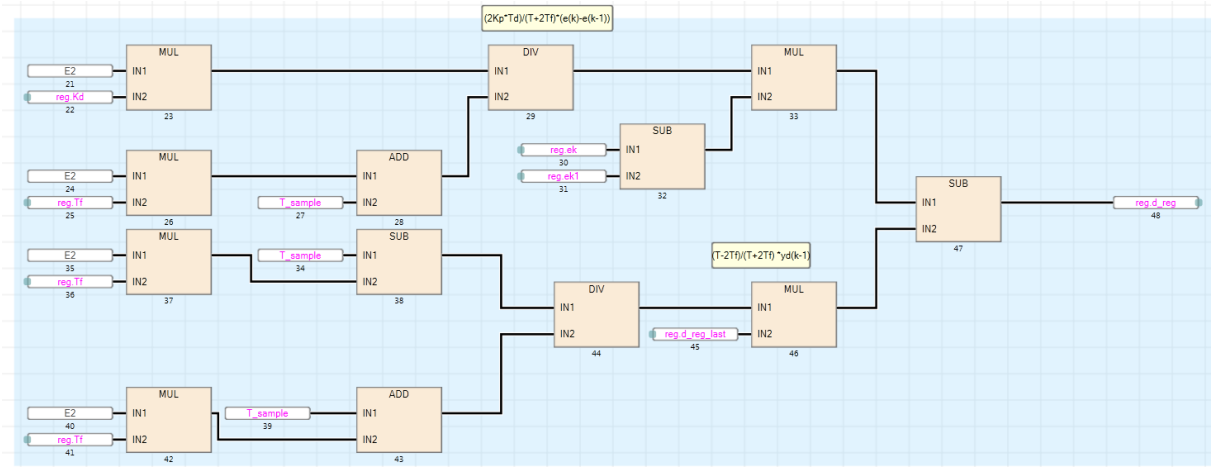
$$u_d(k) = -\frac{T - 2T_f}{T + 2T_f} \cdot u_d(k-1) + \frac{2KT_d}{T + 2T_f}[e_d(k) - e_d(k-1)],$$

kde T_d je derivační časová konstanta a T_f je filtrace derivace. Zvětšováním derivační časové konstanty zmenšuje přeregulování a může zvýšit robustnost ve stabilitě [18].

Realizace integrační a derivační složky v PLC jsou mnohem složitější především kvůli využití programovacího jazyka FBD/LD. Ukázky programů jsou na obrázcích 61 a 62.



Obrázek 61: *Realizace integrační složky regulátoru v PLC*



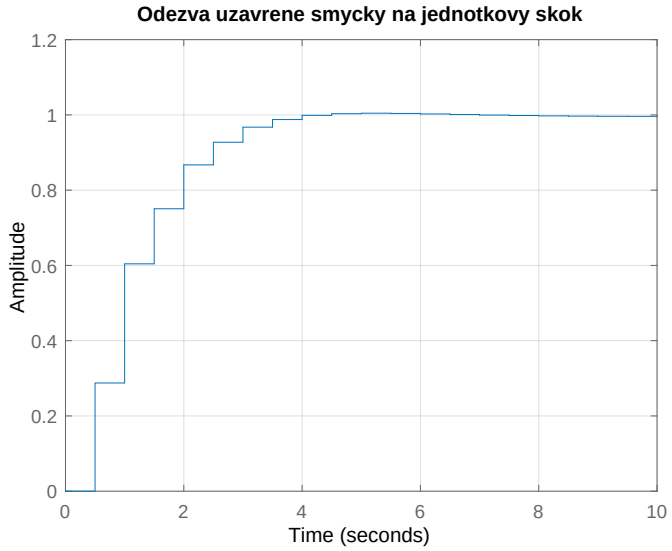
Obrázek 62: *Realizace derivační složky regulátoru v PLC*

5.5.1 Návrh regulátoru

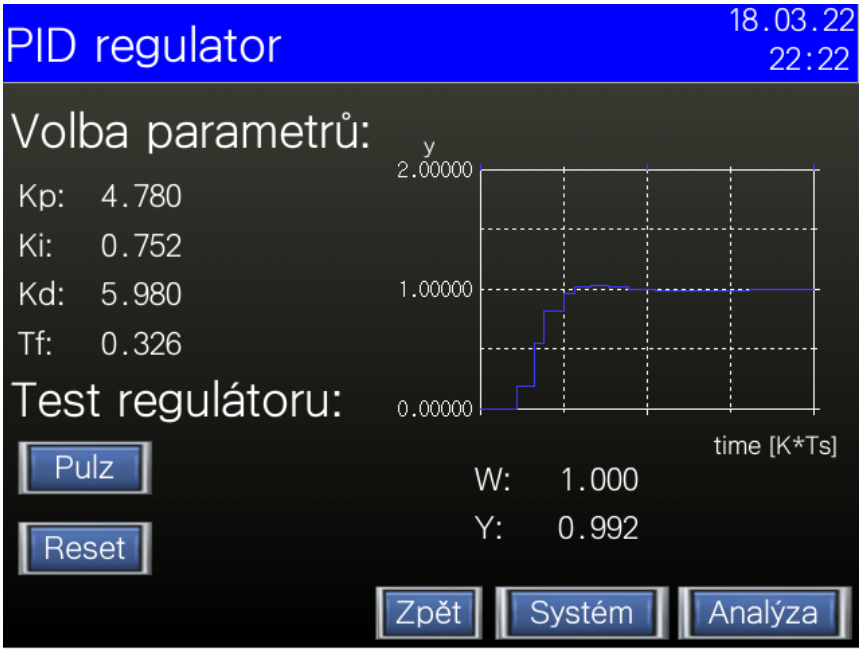
Účelem této školící úlohy je, aby si uživatel sám navrhl regulátor. Uživatel může sledovat, jak změny jednotlivých složek ovlivňují průběh regulace. Nicméně pro ověření funkčnosti regulační smyčky byl navržen PID regulátor ve vývojovém prostředí Matlab. Regulátor byl navržen pro výše konkrétně uvedený systém. Pro nalezení správných parametrů

regulátoru byly využita metoda loopshaping pomocí toolboxu sisotool [19]. Požadavky na regulátor byly stanoveny pouze dva: rychlá odezva výstupu systému a co nejmenší akční zásah.

Parametry nalezeného regulátoru byly přeneseny do PLC programu a výsledné odezvy uzavřené smyčky na jednotkový skok jsou porovnány na obrázcích 63 a 64.



Obrázek 63: *Odezva uzavřeného systému ve vývojovém prostředí Matlab*

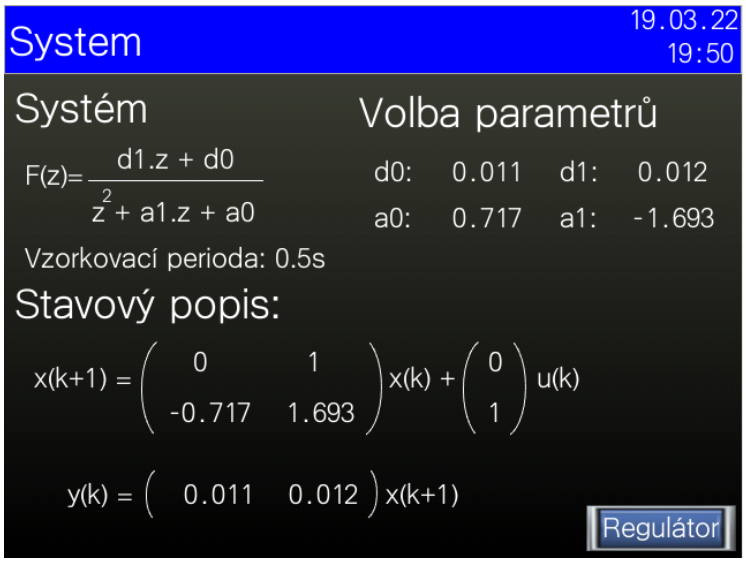


Obrázek 64: *Odezva uzavřeného systému ve vývojovém prostředí GT Designer3*

Na grafech lze vidět, že je v nich nepatrný rozdíl. Tento rozdíl může být zapříčiněn hlavně způsobem vykreslovaní grafů ve vývojovém prostředí GT Designer3. Zároveň toto vývojové prostředí nedisponuje tolika nástroji pro vykreslování signálů tak jako Matlab/Simulink.

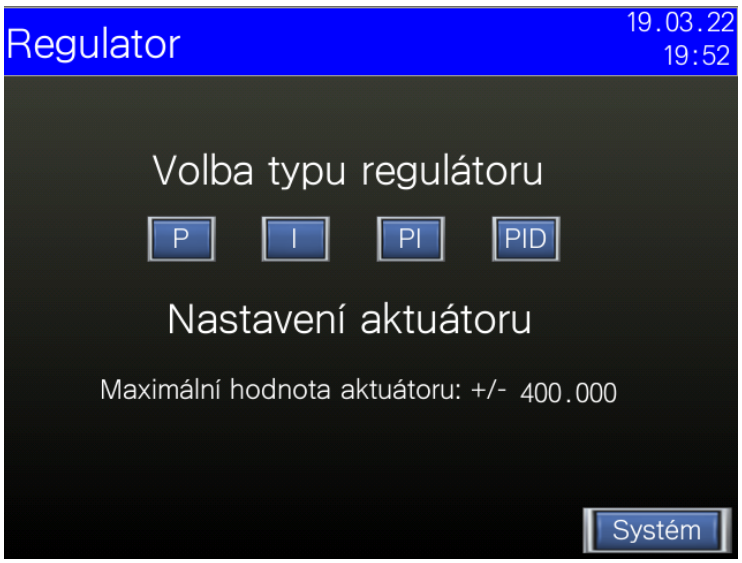
5.5.2 Vizualizace

S programem úzce souvisí vizualizace, pomocí které může uživatel nastavit systém a ladit regulátory. Při spuštění může uživatel navolit jednotlivé parametry systému druhého řádu do tvaru přenosové funkce. Systém je automaticky přepočten a vypsán ve stavové reprezentaci, viz obrázek 65.



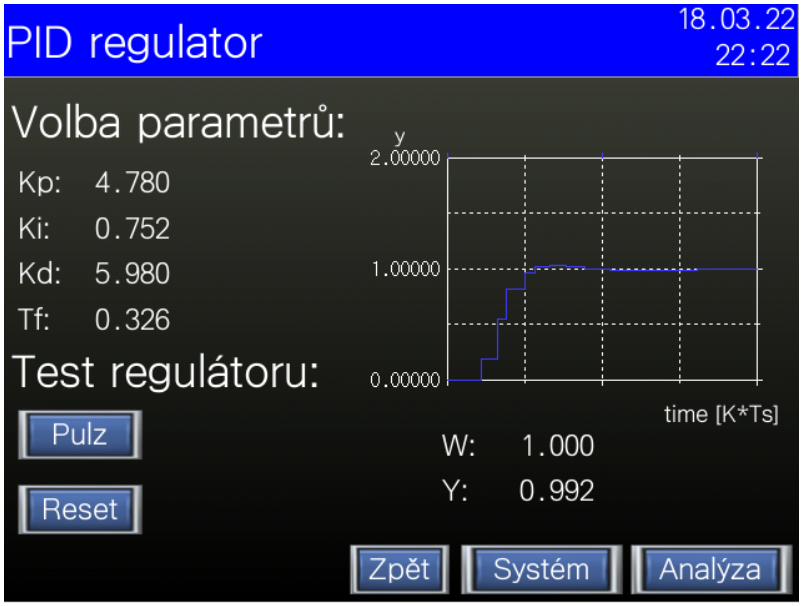
Obrázek 65: *Úvodní stránka vizualizace regulační smyčky*

Dalším krokem je nastavení regulátoru. Je možné vybrat celkem ze čtyř typů regulátoru a to P, I, PI a PID. Kromě typu regulátoru je možné taktéž nastavit hodnotu saturace regulátoru. Saturace akčního členu je taktéž velmi důležitý údaj, který je nutné zohlednit v návrhu řízení.



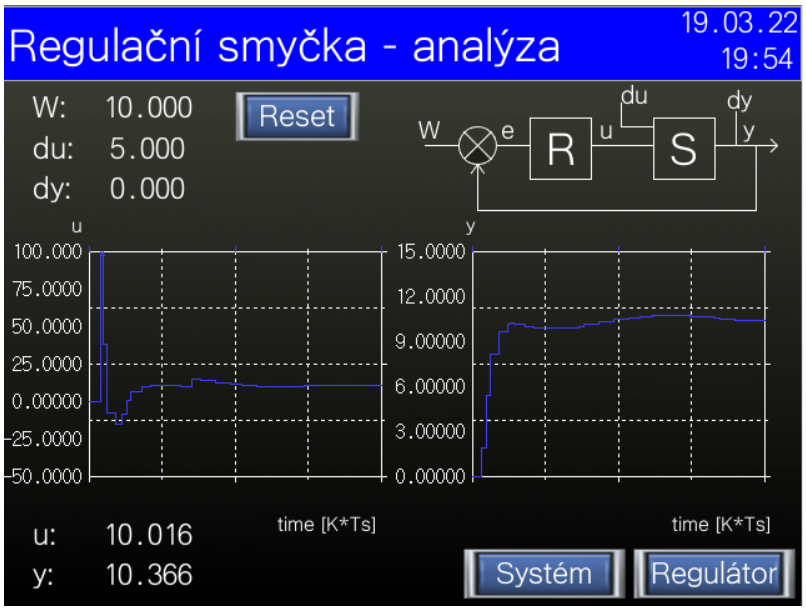
Obrázek 66: *Volba regulátoru*

Při volbě jednoho z regulátorů si uživatel může vyzkoušet nastavení jednotlivých parametrů a sledovat, jak uzavřená smyčka reaguje na jednotkový skok, viz obrázek 67.



Obrázek 67: *Návrh regulátoru*

Po nalezení parametrů regulátoru je možné provést podrobnou analýzu uzavřeného systému. Na obrázku 68 je tato analýza znázorněna. Uživatel může navolit libovolnou požadovanou hodnotu a vstupní resp. výstupní poruchy a sledovat, jak se mění průběhy grafů. Na levém grafu je znázorněn průběh akčního zásahu regulátoru ($u(k)$) a na pravém grafu je odezva uzavřeného systému ($y(k)$).



Obrázek 68: *Analýza uzavřené smyčky*

5.6 Shrnutí školících úloh a jejich možné vylepšení

5.6.1 Model pístu

První model pístu byl vytvořen především za účelem pochopení základních bitových operací v PLC. Díky modelu si uživatel může lépe představit ovládání jednodušších prvků, které mohou být mnohdy součástí komplexnějších systémů. Kromě vysouvání, zasouvání a aktivaci koncových čidel je do modelu možné zanést poruchy, které přibližují simulaci blíže k reálnému systému. Mezi poruchy se řadí nefunkční čidla či zaseknutí pístu. Uživatel tak může ověřit robustnost svého navrženého algoritmu. Model dále umožňuje měnit jednotlivé parametry pístu jako je délka, rychlost nebo typ (dvojčinný/jednočinný). Tato vlastnost implikuje znovupoužitelnost modelu a tedy modularitu. V simulaci je tak možné použít celou řadu pístů s různými parametry bez nutnosti jakékoliv programové změny modelu. K pístu byla vytvořena vizualizace, která umožňuje detailně zobrazit funkci pístu. Pro demonstraci funkčnosti modelu byl vytvořen jednoduchý systém tří pístů a k nim byla navržena vhodná sekvence řízení.

Mezi nedostatky, které by mohly být v budoucnu napraveny patří hlavně centrální diskrétní integrátor, který slouží jako vnější čítač. Při návrhu nebyl nalezen jiný efektivní způsob, který by umožnil lepší způsob počítání času.

5.6.2 Model dopravníku

Funkční blok pístu byl navržen co nejobecněji, což umožnilo blok využít při sestavování simulace dopravníku. Hlavní funkcí modelu pístu bylo počítání aktuální polohy z času a rychlosti posunu. Pro dopravník byla tato myšlenka pouze rozšířena pro počítání více poloh najednou. Funkční blok dopravníku tedy uvnitř zahrnuje upravený model pístu, pomocí kterého jsou dopočítávány jednotlivé polohy produktů na páse. Tento systém byl tak rozšířen o potřebné funkce, které musí dopravník obsahovat. Mezi nejdůležitější přidanou funkci patři především vzájemné chování produktů mezi sebou. Produkty se musí jeden o druhý zastavit v případě dorazů. Uživatel může opět volit libovolné parametry dopravníku. Kromě rychlosti a délky dopravníku byly též přidány možnosti nastavení počtu produktů, délky produktů, počet čidel a jejich poloha. K simulaci byla rovněž vytvořena vizualizace, na které je možné sledovat aktuální polohu produktů na dopravníku. Stejně jako u modelu pístu je u dopravníku zaručena do jisté míry modularita. Ta byla využita při sestavování druhé školící úlohy, která se skládá z dvou předešlých modelů.

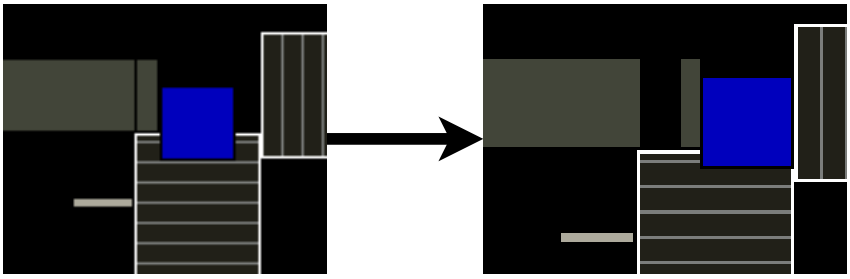
Nedostatkem v modelu dopravníku je opět vnější čítač. Dopravník dále neumožňuje oboustranný pohyb. Produkty se mohou pohybovat pouze jedním směrem, pokud tedy dosáhnou nějaké nežádoucí polohy, nelze je vrátit. Tato funkce by mohla být časem doplněna do modelu.

5.6.3 Model Linky

Druhou školící úlohou byla simulace linky. Linka byla vytvořena na míru dle předlohy reálného modelu. Na simulaci je možné vyzkoušet řízení komplexnější soustavy, která se skládá z více prvků. Konkrétně je složena z dvou pístů a čtyř dopravníků, které musí být řízeny současně. Modularita u předchozích modelů byla žádoucí hlavně z důvodů této

školící úlohy. Již vytvořené funkční bloky byly poskládány a navzájem propojeny tak, aby dohromady tvořily jednu soustavu. Problém nastal v částech přechodů mezi různými prvky. Přechod mezi pístem a dopravníkem musel být vyřešen mimo funkční bloky. Do simulace bylo nutné taktéž zařadit logiku objektu, který je sunut pístem. Úlohu je možné řešit několika způsoby, čímž si uživatel může vyzkoušet různé přístupy řízení a osvojit si tak základy programování pro PLC. Pro úlohu bylo vytvořené současně i řízení pro plnou demonstraci struktury a základů programů řízení.

Linka je vytvořena jako programový celek a ne jako funkční blok. Je to zapříčiněné tím, že v tomto případě se nepovedlo dosáhnout modularity. Ta je možná pouze u jednotlivých prvků modelu. Rozložení a počet prvků měnit nelze. Dalším nedostatkem linky je přechod mezi pístem a dopravníkem. V moment, kdy produkt opustil dopravník a sune se před píst, je zavedeno zjednodušení modelu. To spočívá v tom, že v daný moment nelze hýbat pístem, dokud produkt není přesně před pístem. Toto bylo zavedeno především kvůli spojitým přechodům ve vizualizaci. Zjednodušení by mohlo být v budoucnu nahrazeno šikmým pohybem produktu, pokud by jej píst sunul dřív, než bude produkt kolmo před ním. Zjednodušená situace a její možné nahrazení je znázorněno na obrázku 69.



Obrázek 69: *Zjednodušení modelu a možné budoucí vylepšení*

5.6.4 Model regulátoru

Poslední školící úloha nabízí možnost simulace uzavřené regulační smyčky. Uživatel může zadat libovolný systém druhého řádu, ke kterému může navrhnout příslušný regulátor. Typy možných regulátorů byly pevně stanoveny na **P**, **I**, **PI** a **PID**. Kromě návrhů jednotlivých parametrů regulátoru může uživatel stanovit, v jakých hodnotách dochází k saturaci aktuátoru. K modelu byla vytvořena vizualizace, pomocí které lze sledovat, jak změny jednotlivých parametrů ovlivňují výstup uzavřené smyčky. Kromě jednoduchého ověření funkce regulátoru nabízí simulace i komplexnější analýzu uzavřené smyčky. V té si uživatel může vyzkoušet, jak se smyčka chová při vstupních či výstupních poruchách. Dále lze sledovat průběh akční veličiny a je možné tak ověřit, že nedochází k saturaci akční veličiny.

Ačkoliv lze systému druhého řádu libovolně měnit parametry, není to doporučeno. Výchozí systém je zvolen dostatečně pomalý, aby bylo možné sledovat jeho průběh v PLC. Dalším problémem je, že vývojové prostředí pro vizualizaci neumožňuje detailní náhled na jednotlivé charakteristiky uzavřené smyčky. Uživatel se musí spokojit s hrubou vizualizací průběhů. Budoucím plánem je nalezení lepších způsobů, jak vykreslit jednotlivé grafy, ze kterých by bylo možné odečíst přesné hodnoty.

6 Závěr

Cílem této bakalářské práce bylo sestavení školících úloh, které budou sloužit k zaučování nováčků v oboru programování programovatelně logických automatů. První část práce je věnována obecnému popisu PLC. Byla uvedena základní definice společně s obecnou architekturou a rozdělením PLC do skupin dle možnosti modularity. Dalším tématem byly výhody, které tyto řídící jednotky přinesly oproti svým předchůdcům. Posledním tématem bylo zpracování programu v PLC. Tato část byla klíčová zejména pro pochopení základních principů programování, které bylo rozebrané v další části.

Druhá část je tedy věnována samotnému programování PLC. Byly vysvětleny základní prvky programování, které byly následně využity v praktické části. Následně byly uvedeny i základní programovací jazyky. Kromě základního grafického jazyka LD, který je nejrozšířenější v aplikacích PLC, byly popsány jazyky ST a FBD/LD. První z uvedených není oproti ostatním grafickým jazykem a své využití nalezne především ve složitějších aplikacích. Druhý z uvedených jazyků je FBD/LD a kombinuje funkční bloky společně s Ladder diagramem. Právě tato kombinace byla žádoucí při sestavování modelů jednotlivých školících úloh.

Třetí část je věnována seznámení s nástroji pro tvorbu programů PLC. Kromě seznámení s programovacími prostředími GX Works2 a GX Works3 byla demonstrována základní tvorba vizualizace pomocí nástroje GT Designer3. V kapitole je po jednotlivých krocích vysvětlen postup vytváření programů a nováček si tak může vyzkoušet základní principy programování PLC. Kapitola taktéž slouží k demonstraci jednotlivých technik, které byly využity při sestavování praktické části.

V praktické části bakalářské práce byly modelovány tři školící úlohy. První úloha byl model pístu, díky kterému si uživatel může vyzkoušet základní bitové operace v PLC. Píst byl navržen velmi obecně, čímž bylo možné jeho program rozšířit a vytvořit z něj simulaci dopravníku. Model pístu i dopravníku byly navrženy jako funkční bloky, kterým je možné libovolně měnit parametry. Tento aspekt společně s vlastností znovupoužitelnosti funkčního bloku ilustruje možnosti modularity daného přístupu. Ta je využita především u druhé školící úlohy, která se věnuje simulaci linky. Linka je tvořena z funkčních bloků pístů a dopravníků, které jsou mezi sebou propojeny. Díky této úloze je možné si vyzkoušet řízení složitější úlohy, která je blíže praktickým realizacím. Poslední úlohou je simulace regulační smyčky. Model umožňuje návrh regulátorů zadaných typů (P, I, PI nebo PID) na libovolný systém druhého řádu. Všechny úlohy jsou doplněny o vizualizaci, takže uživatel může pozorovat, jak se chová systém pod jeho navrženém řízení.

Konec praktické části je věnován shrnutí školících úloh a jejich možnému vylepšení do budoucna. Vylepšení se hlavně týká druhé úlohy, která obsahuje několik zjednodušení od reality. Budoucím doplňkem je tedy přiblížení modelu více reálné implementaci a tím zvýšit jeho přesnost a věrohodnost. Dále by na základě modularity jednotlivých modelů bylo možné v budoucnu vytvořit další komplexnější školící úlohu.

Seznam všech použitých zkratek

- **CPU** - Centrální procesorová jednotka
- **ECU** - Electronic Control Unit
- **FUN** - Funkce
- **FB** - Funkční blok
- **FBD** - Funkční Blokový Diagram
- **GOT** - Graphic Operation Terminal
- **HW** - Hardware
- **HIL** - Hardware In the Loop
- **HMI** - Human Machine Interface
- **LD** - Ladder Diagram
- **MBD** - Model Based Design
- **MIL** - Model In the Loop
- **NC** - Normal Closed
- **NO** - Normal Open
- **PIL** - Processor In the Loop
- **PLC** - Programovatelný logický automat
- **SW** - Software
- **SIL** - Software In the Loop
- **STD** - Strukturovaný Datový Typ
- **ST** - Strukturtovaný Text

Seznam obrázků

Seznam tabulek

Odkazy

[1] Tomáš Ausberger et al. „Test case generation for Function Block Diagram based on blocks' predefined behaviour“. In: *2021 23rd International Conference on Process Control (PC)*. 2021, s. 206–211. DOI: 10.1109/PC52310.2021.9447525.

[2] Karel Kubíček, Martin Čech a Jan Škach. „Continuous enhancement in model-based software development and recent trends“. In: *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2019, s. 71–78. DOI: 10.1109/ETFA.2019.8869237.

[3] Martin Čech et al. „Novel tools for model-based control system design based on FMI/FMU standard with application in energetics“. In: *2017 21st International Conference on Process Control (PC)*. 2017, s. 416–421. DOI: 10.1109/PC.2017.7976250.

[4] Tomáš Ausberger et al. „Model Checking application on Function Block Diagram model“. In: *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Sv. 1. 2020, s. 1807–1814. DOI: 10.1109/ETFA46521.2020.9212180.

[5] Tomáš Ausberger et al. „Analytic method for automatic test case generation for Function Block Diagram“. In: *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Sv. 1. 2020, s. 1799–1826. DOI: 10.1109/ETFA46521.2020.9212034.

[6] Bc. Karel Kubíček. „Modelově orientovaný vývoj softwaru: řízení spojky automatické převodovky kamionů“. diplomathesis. Západočeská univerzita v Plzni, 2019.

[7] *Introducing to Programmable Logic Controllers (PLC's)*. University Lecture. 2006.

[8] Mitsubishi Electric. *MELSEC FX Family Programmable Logic Controllers Beginner´s Manual*. URL: https://si3a.mitsubishielectric.com/fa/sl/dl/2340/166388.pdf (cit. 20. 01. 2022).

[9] Industrial Text & Video Company. *Introducing to PLC Programming and Implementation - from Relay Logic to PLC Logic*. Industrial Text & Video Company, 1999.

[10] Mitsubishi Electric. *Mitsubishi Programmable Controler Q-series advanced course (for GX works2)*. URL: https://cz3a.mitsubishielectric.com/fa/cs/dl/12448/SH(NA)-081124ENG-A.pdf (cit. 20. 01. 2022).

[11] Mitsubishi Electric. *MELSEC iQ-F FX5 Series Programming Manual*. URL: http://www.allied-automation.com/wp-content/uploads/2015/05/MITSUBISHI_manual_plc_fx5_programming.pdf (cit. 20. 01. 2022).

[12] *PLC Academy*. URL: https://www.plcacademy.com (cit. 26. 01. 2022).

[13] Mitsubishi Electric. *GX works3 Operating Manual*. URL: https://eu-assets.contentstack.com/v3/assets/blt5412ff9af9aef77f/bltbe900f675b2b290a/61722894ca617d08ba7646eb/sh081215.pdf (cit. 20. 01. 2022).

[14] Mitsubishi Electric. *GX Works2 Beginner's Manual (Strutured project)*. URL: https://dl.mitsubishielectric.com/dl/fa/document/manual/plc/sh080788eng/sh080788engr.pdf (cit. 27. 01. 2022).

[15] Mitsubishi Electric. *GX Works2 Operating Manual (Common)*. URL: http://www.int76.ru/upload/iblock/5c9/5c951f4448672b4f8143a7f284cd2545.pdf (cit. 27. 01. 2022).

[16] Mitsubishi Electric. *Screen Design Software GT Designer 3*. URL: https://www.mitsubishifa.co.th/files/dl/sh080866engan.pdf (cit. 27. 01. 2022).

[17] Doc. Ing. Jiří Melichar CSc. a Ing. Martin Goubej Ph.D. *Lineární systémy 1*. učební text. 2018.

[18] Doc. Ing. Jiří Melichar CSc. a Ing. Martin Goubej Ph.D. *Lineární systémy 2*. učební text. 2018.

[19] MathWorks. *Control System Designer*. URL: https://www.mathworks.com/help/control/ref/controlsystemdesigner-app.html (cit. 28. 03. 2022).