Univerzita Pardubice

Dopravní fakulta Jana Pernera

Ovládání modelu soustavy ”Kulička na rameni”

Radek Ambrus

Diplomová práce

2017

Univerzita Pardubice
Dopravní fakulta Jana Pernera
Akademický rok: **2016/2017**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Radek Ambrus**
Osobní číslo:	**D15449**
Studijní program:	**N3708 Dopravní inženýrství a spoje**
Studijní obor:	**Elektrotechnické a elektronické systémy v dopravě**
Název tématu:	**Ovládání modelu soustavy "Kulička na rameni"**
Zadávající katedra:	**Katedra elektrotechniky, elektroniky a zabezpečovací techniky v dopravě**

Zásady pro vypracování:

Dokončete rozpracovaný laboratorní přípravek "Kulička na nakloněném rameni", který bude využíván ve výuce základů teorie řízení na KEEZ.
Doporučený postup:
1. Seznámení se s aktuálním stavem rozpracovanosti zařízení.
2. Úprava firmware řídící jednotky zařízení (Arduino Mega2560).
a. Úprava stávajícího kódu (zpřehlednění a oprava chyb).
b. Doplnění komunikačního protokolu do kódu, aby bylo možné zařízení ovládat z PC přes USB.
3. Vytvoření ovládacího SW v NI LabView, požadované funkce:
a. Vizualizace regulačního pochodu.
b. Logování dat do souboru.
c. Zpětnovazební řízení polohy kuličky nebo ruční zadávání polohy ramene.
d. Nastavení regulátoru polohy kuličky.
4. Ověření funkce zařízení, porovnání výstupů simulačního modelu s reálnou soustavou.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování diplomové práce: **tištěná**

Seznam odborné literatury:

BALÁTĚ, Jaroslav. Automatické řízení. 2. vydání. BEN - technická literatura, 2004. ISBN 978-80-7300-148-3.
OGATA, Katsuhiko. Modern control engineering. 5th ed. Boston: Prentice-Hall, c2010, x, 894 p. ISBN 0136156738.
BRESS, Thomas. Effective LabVIEW Programming. NTS Press, 2013. ISBN 978-1-934891-08-7.
VÁŇA, Vladimír. Mikrokontroléry ATMEL AVR - Programování v jazyce C: popis a práce ve vývojovém prostředí CodeVisionAVR C. Praha: BEN - technická literatura, 2003. ISBN 8073001020.
BUREŠ, Zdeněk. Návrh a realizace modelu Kulička na rameni. Pardubice, 2015. Diplomová práce. Univerzita Pardubice.
Datasheety výrobců komponent.

Vedoucí diplomové práce: **Ing. Zdeněk Mašek, Ph.D.**
Katedra elektrotechniky, elektroniky a zabezpečovací techniky v dopravě

Datum zadání diplomové práce: **28. listopadu 2016**

Termín odevzdání diplomové práce: **26. května 2017**

doc. Ing. Libor Švadlenka, Ph.D.
děkan

L.S.

Ing. Dušan Čermák, Ph.D.
vedoucí katedry

V Pardubicích dne 15. března 2017

Prohlášení autora

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury. Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 26. 5. 2017 Radek Ambrus

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Zdeňkovi Maškovi, Ph.D., za jeho odborné vedení, cenné rady, připomínky a návrhy v celém průběhu práce. Dále bych chtěl poděkovat všem kamarádům, kteří mi byli nápomocni po celou dobu studia. Závěrem bych chtěl poděkovat celé své rodině, která mě vždy ve studiu podporovala.

ANOTACE

Diplomová práce se zabývá ovládáním modelu soustavy ”Kulička na rameni”. Jedná se o navázání na diplomovou práci s názvem Návrh a realizace modelu ”Kulička na rameni”. Autorem byl Bc.Zdeněk Bureš. Práce je rozdělena na několik částí. Prvním bodem bylo analyzovat aktuální stav zařízení. Dále byl upraven firmware řídící jednotky. A nakonec byl vytvořen ovládací software, pomocí kterého lze laboratorní přípravek ovládat z počítače.

KLÍČOVÁ SLOVA

Kulička, rameno, Arduino, firmware, ovládací software

TITLE

Control "Ball on beam" model system.

ANNOATTION

Master thesis is focused on the control "Ball on beam" model system. This is continuation of the Master thesis Design and implementation of "Ball on beam" model from the author Bc. Zdeněk Bureš. The thesis is divided into several parts. An analyzing the current status of the device was the first part. Then the firmware of the control unit was modified. In the last part was created the control software, which can be used for an easy controlling of the device from the computer.

KEYWORDS

Ball, beam, Arduino, firmware, control software

Obsah

ÚVOD

Tato práce se zabývá úpravou modelu zařízení ”Kulička na rameni”. Jedná se o zařízení, které bude využíváno pro výuku na KEEZ. Laboratorní přípravek byl zhotoven v rámci Diplomové práce, jejíž autorem byl Bc. Zdeněk Bureš.

Jeho práce se zabývala Analýzou možných řešení, výběrem jednotlivých komponent pro zařízení, simulační částí, ve které se zabýval matematickým popisem, návrhem regulátoru a nakonec vytvořením simulačního modelu. Druhá část práce se zabývala vlastní konstrukcí zařízení, realizací číslicového regulátoru a na závěr došlo k ověření funkce celkového zařízení.

Reálný stav zařízení je takový, že po hardwarové stránce se jedná o plně funkční výrobek a s jeho aktuálním stavem se seznámíme v jedné z následujících částí. Z hlediska softwarového je přípravek oživen a byla ověřena jeho funkce jako celku. Software plní pouze základní funkci.

Z hardwarové stránky nebudou provedeny žádné úpravy. Zásahy se budou týkat pouze softwarové části. Tuto úpravu lze rozčlenit na dva hlavní úkoly. Prvním úkolem je úprava firmwaru řídící jednotky, aby byl přehlednější a jeho správa jednodušší. A druhým, by mělo být rozšíření programu o komunikační protokol, který bude sloužit pro komunikaci mezi přípravkem a počítačem. A bude tak možné přípravek ovládat pomocí aplikace přes PC.

Vytvoření ovládací aplikace pro přípravek byla jedním z dalších bodů této práce. Ovládací aplikace by měla sloužit k jednoduchému a rychlému ovládání laboratorního přípravku bez nutnosti znalosti komunikačního protokolu. Pro vytvoření této aplikace bude využito vývojové prostředí LabVIEW. Výsledná aplikace by měla plnit několik základních funkcí. Mezi ně patří vizualizace regulačního pochodu, logování přijatých dat do souboru, zpětnovazební řízení polohy kuličky nebo zadání úhlu natočení ramene a nastavení konstant regulátoru polohy.

1. AKTUÁLNÍ STAV ZAŘÍZENÍ „KULIČKA NA RAMENI“

V této části práce se seznámíme s aktuálním stavem přípravku „Kulička na rameni“. Aktuálně stavu se jedná o plně funkční model po hardwarové stránce. Z hlediska softwarové části je zařízení oživené a plní pouze základní funkce. Aby byl přípravek plně funkční, musí být připojen síťovým kabelem na napětí 230V 50Hz a zároveň musí být připojen pomocí kabelu k USB portu PC.

1.1 HW část

Všechny komponenty zařízení jsou připevněné na dřevotřískové desce o rozměrech 120x30 cm. Zařízení je rozděleno na část silnoproudou a slaboproudou.

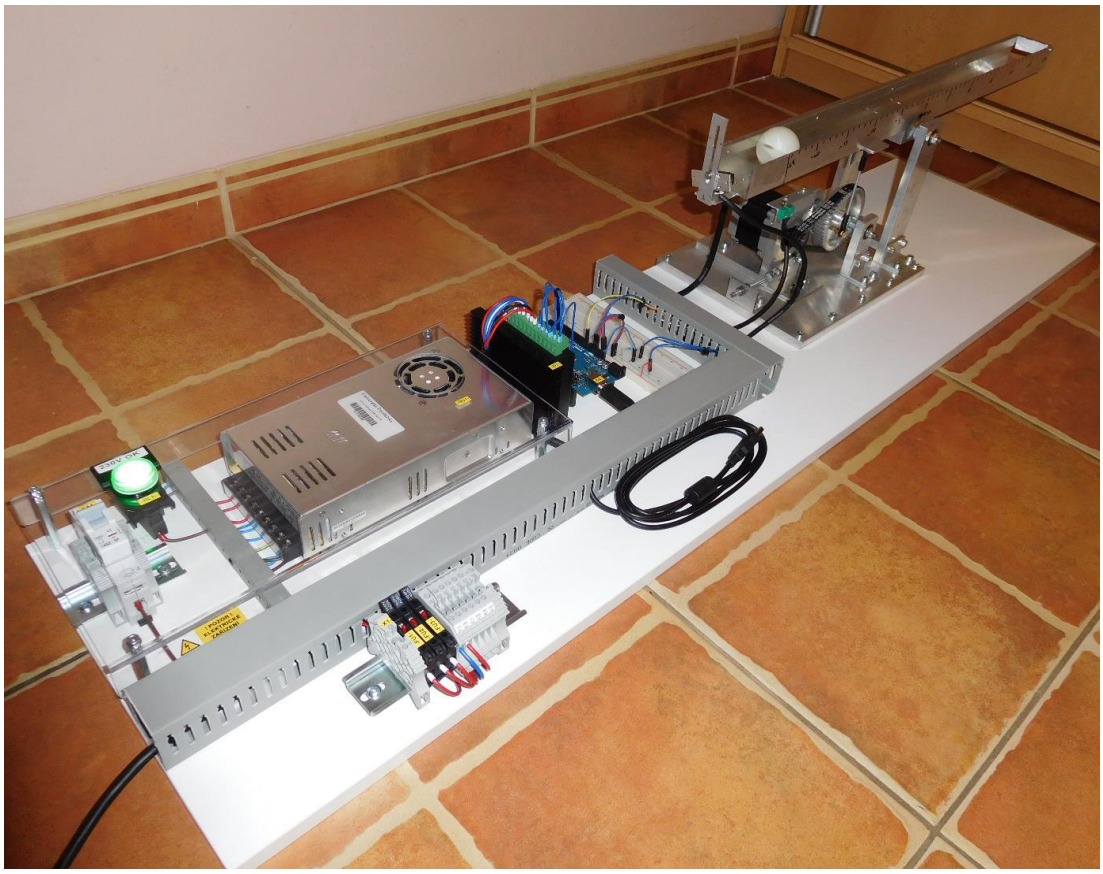
Silnoproudá část, tedy jistič a spínaný napájecí zdroj jsou kryty čirým plastem. Ten slouží jako ochrana uživatelů před případným zásahem elektrického proudu. Zároveň je na této ochranné plastové části připevněna signální kontrolka, která svítí, pokud je zařízení připojeno k napájení. Jistič zde plní funkci prvku, který slouží pro připojení, respektive odpojení zařízení od napájení.

Další část je možné označit jako slaboproudou, kterou tvoří Driver, který slouží pro napájení a řízení krokového motorku a řídící deska Arduino MEGA2560. Do této části spadá také v neposlední řadě krokový motor, pomocí kterého je ovládáno rameno, optický snímač polohy kuličky, který je připevněn na konci ramene a koncový spínač. Veškeré propojovací kabely jsou ukryty v lištách, které jsou pevně připevněny na desku.

Poslední je část mechanická. Jedná je o bytelnou hliníkovou konstrukci, ke které je připevněno rameno. Přenos momentu motoru na rameno je řešen pomocí ozubených kol a řemenu. Větší z ozubených kol je poté spojeno táhlem s ramenem.

Jako kulička, jejíž poloha bude regulována, byl vybrán pingpongový míček. Ten je dostatečně velký, takže není problém snímat jeho polohu po celé délce ramene.

Pod kabelovým žlabem jsou připevněny pojistková pouzdra a svorky, které nemají žádnou souvislost s funkci celého zařízení. Na tyto svorky je pouze vyvedeno stejnosměrné napětí 24V, které může být využito pro napájení různých externích zařízení. [1]



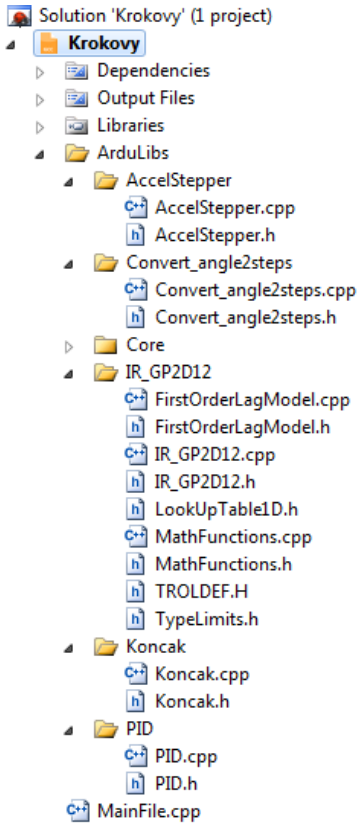
Obrázek 1 Celkový pohled na laboratorní přípravek [1]

Hlavní využité komponenty:

- Spínaný napájecí zdroj Mean Well SP-320-24
- Driver typ HY-DIV268N-5A
- Řídící deska Arduino MEGA 2560 – s mikrokontrolérem ATmega2560
- Krokový motor SX23-1414d
- Infračervený snímač vzdálenosti analogový - Sharp GP2Y0A21YK0F
- Rameno hliníkový U profil 50x30x2mm, délka 600mm
- Kulička - pingpongový míček - průměr 40mm, hmotnost cca 2g
- Řemen – šířka 9mm rozteč zubů 5mm
- Řemenice 12 a 48 zubů – převodový poměr tedy 1:4

1.2 SW část

Současný řídící program pro ovládání regulace kuličky na rameni se skládá z několika samostatných modulů. Většina jich je tvořena C++ třídami. Každá třída má za úkol určitou funkci v programu jako například PID regulátor, měření aktuální polohy kuličky, ovládání krokového motoru a další. Tyto moduly obsahují další knihovny a podprogramy, které se podílejí na funkci hlavního programu s názvem "Krokovy". [1]



Obrázek 2 Struktura projektu [1]

1.2.1 Jednotlivé využité knihovny [1]

- **AccelStepper**
  Jedná se o knihovnu, která slouží pro ovládání krokového motoru.

- **Knihovna Convert_angle2steps**
  Obsahuje funkce, které slouží k převodu žádaného úhlu na počet kroků pro krokový motor. K převodu je využívána převodní tabulka úhel - kroky. Velikost žádaného úhlu se pohybuje v rozmezí od -15 ° do 15°.

- **IR_GP2D12**
  Slouží k vyčítání hodnot z optického snímače. Součástí je i několik dalších knihoven, které jsou potřebné pro správné určení polohy kuličky. Jedná se například o filtraci snímaných hodnot.

- **Koncak**
  Jednoduchá knihovna, která slouží pouze ke snímání stavu koncového spínače. Využívána je pouze po spuštění, kdy rameno sjíždí na koncový spínač.

- **PID**
  Jedná se o PID regulátor. Regulace je nastavena zadanými parametry Kp, Ki a Kd. Vstupní proměnné pro regulátor jsou hodnoty žádané a aktuální polohy kuličky. Výpočtem získáváme žádaný úhel natočení ramene, který se převádí na počet kroků pro krokový motor.

1.2.2 MainFile.cpp

Jedná se o hlavní zdrojový soubor aplikace. Na začátku zdrojového kódu jsou připojeny patřičné hlavičkové soubory. Následuje vytváření instancí tříd a poté jsou definovány využívané proměnné. MainFile je vždy složen minimálně z dvou funkci a to setup() a loop(). Kde funkce setup() proběhne pouze jednou po startu zařízení a je použita především k inicializování. Funkce loop() se vykonává neustále dokola, avšak kód v této funkci je naspaný tak, že se vykoná pouze jednou. Nakonec jsou zapsány rutiny pro obsloužení přerušení.

1.2.3 Funkce přípravku jako celku

Po propojení zařízení a PC pomocí USB kabelu typu A-B a zapnutí pomocí jističe se ihned začne vykonávat program. Po připojení hlavičkových souborů, definování proměnných, vytvoření instancí tříd se začne vykonávat funkce setup().

Ve funkci setup() dochází k nastavení sériové linky, tedy její přenosové rychlosti, parametrů krokového motoru, PID regulátoru a časovačů. Kdy po vykonání funkce setup(), se začne vykonávat funkce loop(). V tomto případě jsou příkazy ve smyčce loop() vykonány pouze po spuštění, kdy je nutné zjistit polohu ramene. S ramenem se sjíždí na koncový spínač, po sepnutí se rameno nastaví do vodorovné polohy. Po vykonání této sekvence úkonů se teprve mohou vykonávat příkazy v rutině pro obsloužení přerušení od časovače 3.

V této obsluze přerušení se načítá aktuální pozice kuličky, která spolu s žádanou polohou jsou vstupní parametry pro výpočet žádaného úhlu natočení ramene. Tato hodnota je poté pomocí převodní tabulky převedena na počet kroků pro krokový motor, který přechází do žádané polohy, a tím dochází k natočení ramene. Žádaná poloha kuličky je nastavena na 300 mm, což odpovídá středu ramene.

V kódu je ještě zapsána obsluha přerušení časovače 2. Při tomto přerušení je obsluhován krokový motor, který se na žádanou pozici.

1.3 Spojení s přípravkem

Řídící deska laboratorního přípravku musí být připojena k PC pomocí USB portu. Toto spojení slouží zároveň pro vlastní napájení desky a pro komunikaci. Přenos může být teoreticky oboustranný. Což znamená, že můžeme přenášet hodnoty do PC nebo naopak z PC posílat příkazy do řídící desky. Zařízení v současném stavu však nepodporuje příkazy z PC, a pouze po USB vypisuje hodnoty a řetězce, které jsou definovány ve firmwaru řídící desky. Jedná se o

informační zprávy pro uživatele a stavy, ve kterých se zařízení aktuálně nachází. Postupně dochází k výpisu následujících zpráv:

Rameno sjizdi na koncak

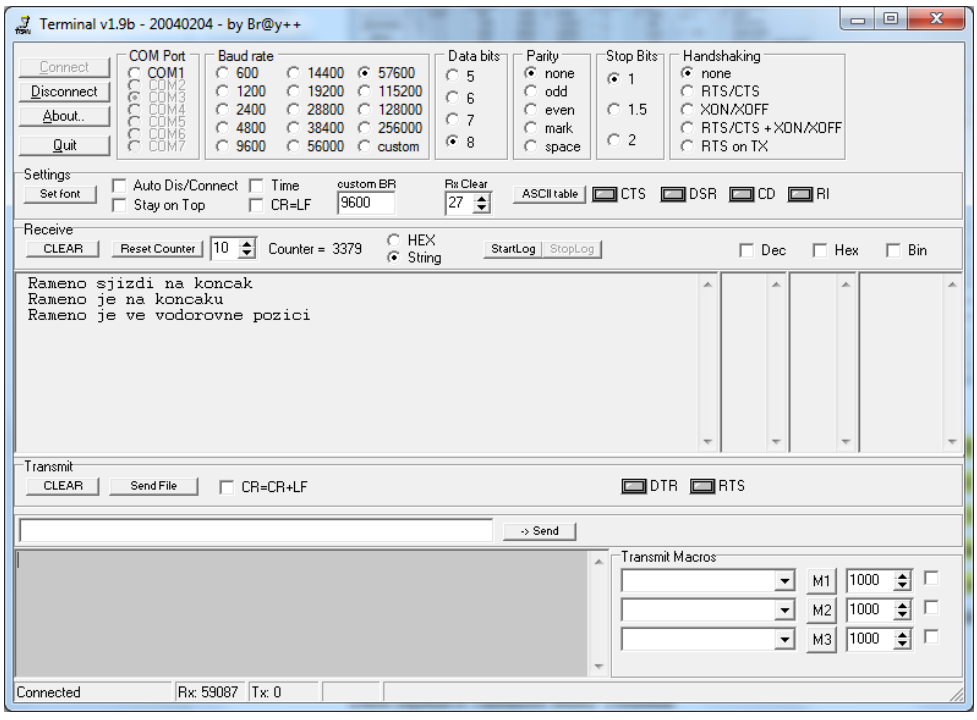
Rameno je na koncaku

Nastavuji vychozi polohu motoru na: -550

Rameno je ve vodorovne pozici

Aktualni poloha motoru = 0

Pro zobrazení vypisovaných hodnot v PC byla využita aplikace „Terminal“. Jedna se o jednoduchý program, který nevyžaduje instalaci a umožňuje snadnou obsluhu komunikace po sériové lince. Na výběr jsou porty COM1 až COM10 a velké množství přenosových rychlostí i s možností vlastní volby přenosové rychlosti. Přijímaná data mohou být zobrazována ve formátu HEX nebo ASCII a zároveň mohou být logována do souboru. Program podporuje odesílání textových příkazů nebo celých souborů. Možné je i využití maker pro periodické odesílání, kde se pouze zadá příkaz a perioda odesílání. [1]



Obrázek 3 Okno aplikace Terminal v1.9b [1]

2. NÁVRH KOMUNIKAČNÍHO PROTOKOLU PRO SPOJENÍ PŘÍPRAVKU S PC

Původní verze firmwaru přípravku nedisponovala obousměrnou komunikací s PC. V programu pro řídící desku tedy bylo pouze možné zapsat příkazy pro výpis aktuálních hodnot a textu. Tyto informace poté mohly být zobrazeny na PC například pomocí již zmíněné aplikace Terminal.

U původní zařízení byly všechny parametry pevně nastaveny v programu. Pokud vznikl požadavek na změnu některého z těchto parametrů, muselo dojít k přepsání hodnot v programu a opětovně ho nahrát do řídící desky. Což je nekomfortní a časově zdlouhavé řešení.

Hlavním cílem této práce je tedy upravit firmware výrobku tak, aby ho bylo možné ovládat pomocí aplikace, která bude spuštěna na PC.

Jedním z prvních kroků tedy bylo navrhnout a implementovat komunikační protokol pro spojení přípravku s PC. Jedná se o soubor příkazů, pomocí kterých bude přípravek ovládán. Kdy komunikace by měla být obousměrná, tedy podporující zápis a čtení. Zápis spočívá v tom, že pomocí příkazu odeslaného z PC, který má přesně definovaný formát a obsah, je možné změnit hodnoty vybraných parametrů i při spuštěném přípravku. Naopak při odeslání příkazu ke čtení dojde k vyčtení aktuální hodnoty některého z podporovaných parametrů. Kdy parametr, který chceme změnit nebo vyčíst, je dán obsahem příkazu.

Celý přípravek by mělo být možné plně ovládat z PC pomocí příkazů po USB.

Návrh komunikačního protokolu mi byl poskytnut vedoucím práce.

2.1 Fyzická a linková vrstva

Tabulka 2.1 Specifikace komunikačního protokolu [2]

Rozhraní	UART/USB
BaudRate	115200 Bd
Stop bit	1
Počet datových bajtů	8
Parita	Žádná

2.2 Dostupné funkce [2]

- Nastavení jednotlivých parametrů PID regulátoru polohy kuličky: Kp, Ti, Td, N

- Vyčtení jednotlivých parametrů PID regulátoru polohy kuličky z přípravku: Kp, Ti, Td, N
- Nastavení žádané polohy kuličky [mm]
- Vyčtení skutečné polohy kuličky [mm]
- Nastavení módu regulace:
    - Nastavování úhlu lavice na ruku
    - Automatická zpětnovazební regulace polohy kuličky
- Nastavení žádaného úhlu lavice [°]
- Vyčtení skutečného úhlu lavice [°]
- Spuštění/zastavení regulace
- Navázání/ukončení komunikace s přípravkem

2.3 Seznam podporovaných příkazů

Komunikace mezi přípravkem a PC je textová, odesílají se přesně definované textové řetězce. Řetězec musí být vždy zakončen zakončovacím znakem, jinak nedojde k rozpoznání příkazu. Jako zakončovací znak byl zvolen podtržítka ” _ ”. Procesní hodnoty odesílané z přípravku jsou odděleny středníkem.

Každý příkaz lze pomyslně rozdělit na 3 základní části, v případě některých příkazů pro zápis i na 4 části.

Příkaz složený z 3 částí:

Typ příkazu | výběr podporované funkce | zakončovací znak

Příkaz složený ze 4 částí:

Typ příkazu | výběr podporované funkce | nová hodnota | zakončovací znak

Typ příkazu – jedná se o výběr mezi čtením – symbol “ r ” a zápisem – symbol “w”

Výběr podporované funkce – text, který definuje, o jakou z podporovaných funkcí se jedná. Jednotlivé příkazy jsou uvedeny v tabulkách 3.2 a 3.3.

Nová hodnota - Některé příkazy pro zápis vyžadují zadání nové hodnoty. Nová hodnota je vyžadována v jednom ze dvou tvarů (tvar závisí na příkazu) **:**

xxx.yyy – textový řetězec, který představuje reálné číslo

xxx – textový řetězec, který představuje celé číslo

Zakončovací znak – Reprezentuje konec příkazu.

Tabulka 2.2 Podporované příkazy pro zápis [2]

Příkaz	**PC → přípravek**	**přípravek → PC**	**Poznámka**
Navázání komunikace s přípravkem	wCONNECT_	CONNECTED_	
Ukončení komunikace s přípravkem	wDISCONNECT_	DISCONNECTED_	
Nastav Kp reg. polohy kuličky	wKPxxx.yyy_	Žádná odpověď	
Nastav Ti reg. polohy kuličky	wTIxxx.yyy_	Žádná odpověď	
Nastav Td reg. polohy kuličky	wTDxxx.yyy_	Žádná odpověď	
Nastav N reg. polohy kuličky	wNxxx_	Žádná odpověď	
Nastav žádanou polohu kuličky [mm]	wPZADxxx_	Žádná odpověď	
Nastav žádaný úhel lavice [°]	wUZADxxx.yyy_	Žádná odpověď	
Nastavení módu regulace	wMODxxx_	Žádná odpověď	xxx může nabývat těchto hodnot: 0 - řízení úhlu lavice na ruku 1- zpětnovazební regulace polohy kuličky
Spuštění regulace	wRUN_	Žádná odpověď	Po spuštění regulace přípravek začne automaticky do PC periodicky posílat procesní hodnoty.

Zastavení regulace	wSTOP_	Žádná odpověď	Po zastavení regulace přípravek přestane do PC posílat procesní hodnoty.

Procení hodnoty jsou po spuštění regulace odesílány z přípravku do PC ve formátu CSV (textové hodnoty oddělené středníkem) v následujícím tvaru: PeriodaVzorkování;ŽádanáPolohaKuličky;SkutečnáPolohaKuličky;ŽadanýÚhelRamene;SkutečnýÚhelRamene_ [2]

Tabulka 2.3 Podporované příkazy pro čtení [2]

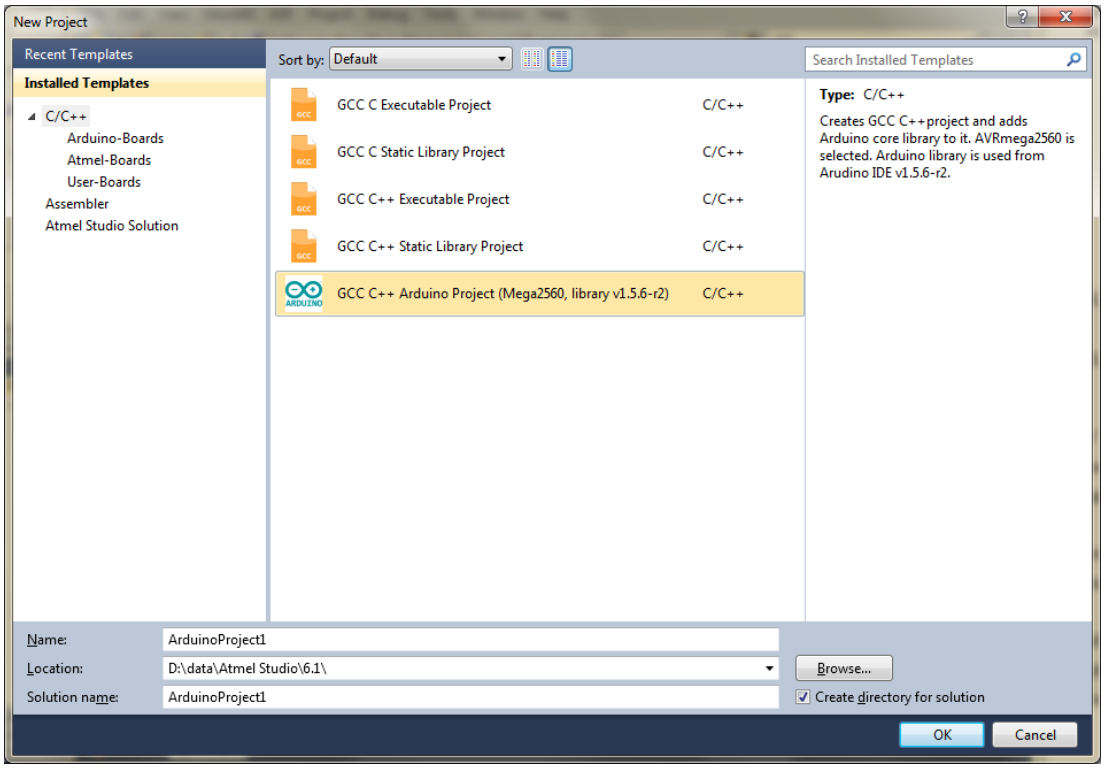
Příkaz	**PC → přípravek**	**přípravek → PC**	**Poznámka**
Vyčti Kp reg. polohy kuličky	rKP_	xxx.yyy_	
Vyčti Ti reg. polohy kuličky	rTI_	xxx.yyy_	
Vyčti Td reg. polohy kuličky	rTD_	xxx.yyy_	
Vyčti N reg. polohy kuličky	rN_	xxx_	
Vyčti žádanou polohu kuličky [mm]	rPZAD_	xxx_	
Vyčti skutečnou polohu kuličky [mm]	rPSKUT_	xxx_	
Vyčti žádaný úhel lavice [°]	rUZAD_	xxx.yyy_	
Vyčti skutečný úhel lavice [°]	rUSKUT_	xxx.yyy_	
Vyčtení módu regulace	rMOD_	xxx_	xxx může nabývat těchto hodnot: 0 - řízení úhlu lavice na ruku 1- zpětnovazební regulace polohy kuličky
Vyčti periodu vzorkování	rTS_	xxx_	xxx je v [μs]

3. VYUŽITÍ ATMEL STUDIA PRO ÚPRAVU FIRMWARU

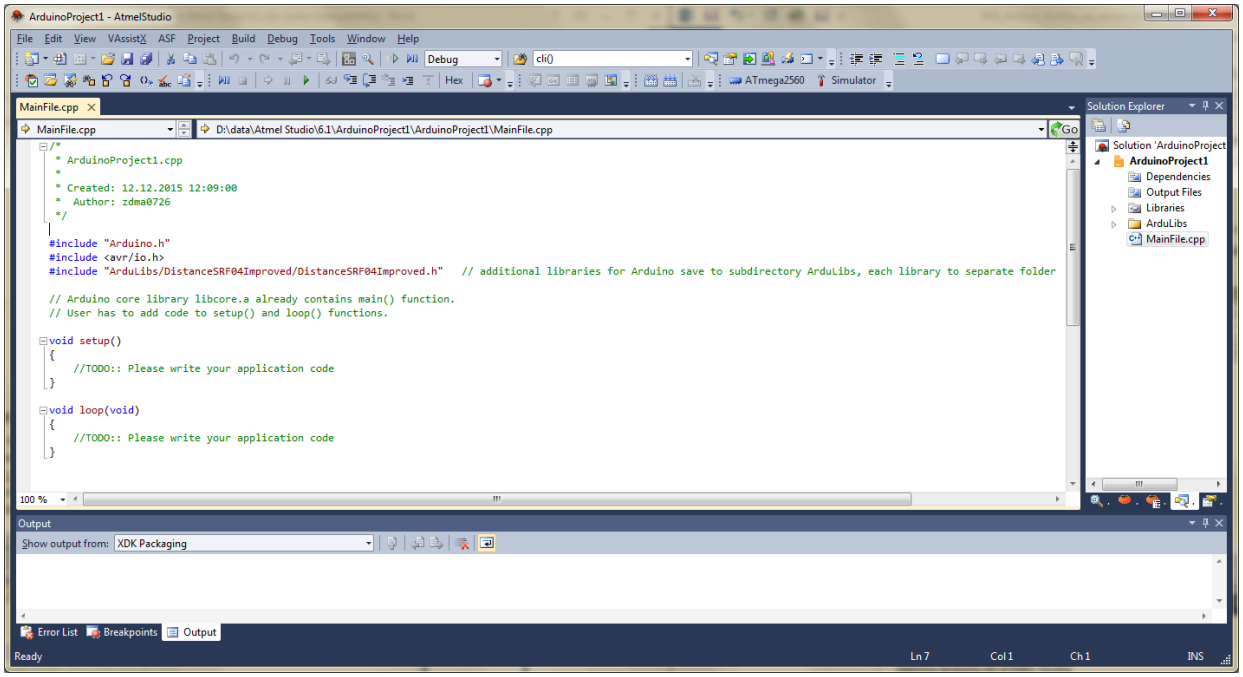
Původní firmware, který byl vytvořen pro tento laboratorní přípravek, nebyl vytvořen za použití vývojového prostředí Arduino. Jelikož bylo zřejmé, že program bude rozsáhlejší, bylo k vytvoření programu použito Atmel Studio 6.1 s využitím programovacího jazyka C++. Atmel studio umožňuje lepší správu projektu. Celý projekt je poté přehlednější a práce tak uživatelsky přívětivější.

Využití běžné verze Atmel studia nedovoluje vytvářet projekty, ve kterých lze psát programy pro platformu Arduino. Aby bylo možné tyto projekty vytvářet je nutné do Atmel studia nainstalovat šablonu Arduino. Po nainstalování tato šablona umožňuje jednoduché založení GCC C++ projektu s využitím hlavních knihoven Arduina. Poté je již v tomto projektu možné psát programy pro Arudino v Atmel studiu. [2]

Atlmel studio 6.1, Arduino IDE ve verzi 1.5.6-r2 a Arduino šablonu, která je určena pouze pro tuto verzi Arduino IDE a využitou desku Mega2560 mi byly poskytnuty vedoucím práce.



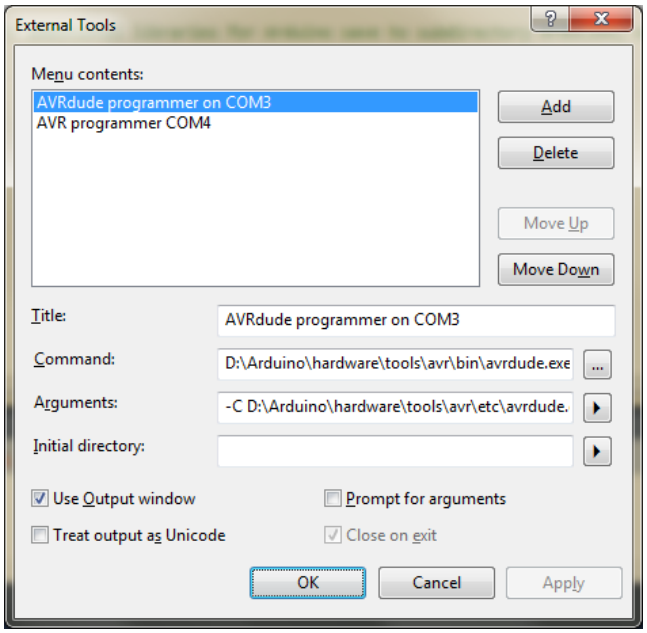
Obrázek 4 Založení projektu s využitím hlavních knihoven Arduino



Obrázek 5 Atmel studio 6.1 struktura nově založeného projektu s využitím šablony Arduino

Založením nového projektu dochází v jeho složce k vytvoření zdrojového souboru MainFile.cpp. V tomto souboru jsou ihned předpřipraveny dvě prázdné funkce, setup() a loop(), pro zapsání kódu.

K programování desek Arduino se využívá aplikace s názvem avrdude.exe, která je součástí Arduino IDE. Pro nahrání programu na desku využívá avrdude.exe USB (virtual COM port). Aby nebylo nahrávání programu složité a zdlouhavé, přidáme aplikaci avrdude.exe jako externí nástroj do Atmel studia.



Obrázek 6 Nastavení avrdude.exe jako External tool

V okně External Tools musíme vyplnit následující části a přidáme pomocí tlačítka Add.

- **Title**

Název, pod kterým se bude nástroj zobrazovat v nabídce

- **Command**

Zde se musí vybrat celá cesta k aplikaci avrdude.exe, nachází se ve složce Arduino IDE (…\Arduino\hardware\tools\avr\bin\avrdude.exe)

- **Arguments**

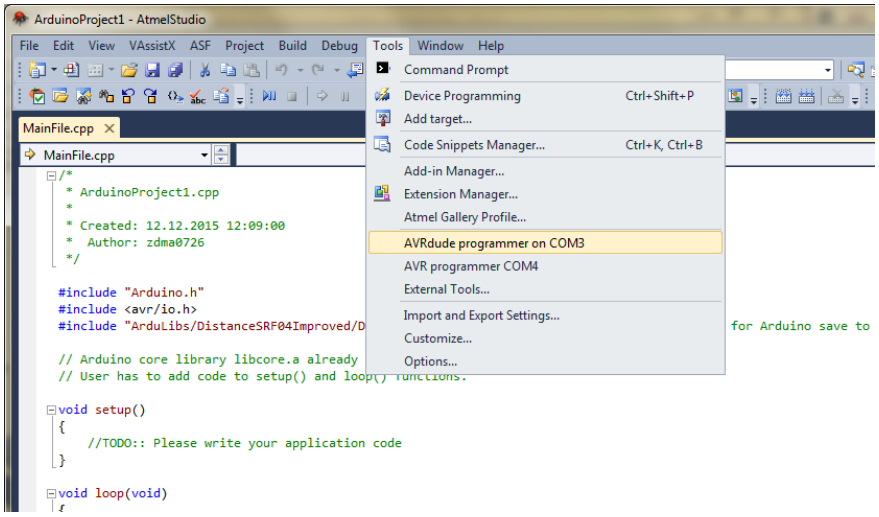
Přidání parametrů, se kterými se bude spouštět avrdude.exe:

-C D:\Arduino\hardware\tools\avr\etc\avrdude.conf -v -v -p atmega2560 -c wiring -P\\.\COM3 -b115200 -D -Uflash:w:"$(ProjectDir)\Debug\$(TargetName).hex":i

V této položce dochází k nastavení COM portu, ke kterému deska bude deska přihlášena. Pokud bude deska přihlášena na jiném portu, je nejjednodušší řešení přepsat číslo COM portu v Arguments.

Další velice důležitý parametr je *-c wiring*. Kdy slovo *wiring* zajistí, že aplikace avrdude.exe automaticky resetuje desku Arduino před programováním, čímž dojde k aktivaci bootloaderu. V případě nezapsání této části by nedošlo ke zresetování desky automaticky a byly by notné desku resetovat manuálně před spuštěním avrdude.exe. [2]

Do desky se nahrává HEX soubor. Před spuštěním programování nejdříve napsaný program přeložíme a poté vybereme nově vytvořený nástroj z nabídky, čímž spustíme programování.



Obrázek 7 Výběr nástroje pro spuštění programování

4. VLASTNÍ ÚPRAVA FIRMWARU PRO PŘÍPRAVEK

Vedoucím práce mi byl poskytnut vzorový firmware pro ovládání kuličky na rameni, který ale nebyl určen pro tento konkrétní laboratorní přípravek. Dostal jsem svolení, že tuto aplikaci mohu využít jako základní kostru pro budoucí program.

Tento program musel být upraven, aby byl plně použitelný a funkční na laboratorním přípravku, kterým se tato práce zabývá. Některé části programu, například převodní tabulky, byly využity z původního programu, který byl součástí diplomové práce s názvem: Návrh a realizace modelu „Kulička na rameni“ od autora Z. Bureše.

V programu bude využíváno několik přidaných knihoven, které plní určitou funkci důležitou pro správný chod programu.

4.1 Využité piny řídící desky Arduino

Využitá řídící deska Arduino je vybavena mikrokontrolérem ATmega2560, který má 100 pinů. Deska Arduino MEGA2560 umožňuje připojení velkého množství periférií. K připojení může být využito 54 digitálních vstupně-výstupních pinů nebo 16 vstupních analogových pinů.

Před úpravou firmwaru bylo nutné analyzovat, na které piny desky Arduino jsou připojeny jednotlivé součásti zařízení. Toto zařízení využívá pouze malé množství dostupných pinů.

Využité piny jsou sepsány v následující tabulce.

Tabulka 4.1 Využité piny na desce Arduino

Číslo / Název pinu	Využití
POWER - 5V	Vyvedeno na nepájivé kontaktní pole Využito k napájení: • Optického snímače • Koncového spínače
POWER – GND	Vyvedeno na nepájivé kontaktní pole Připojeno: • Optický snímač • Koncový spínač – přes rezistror 10 kΩ
Analogový PIN A0	Připojen optický snímač
Digitální PIN 8	Nastaven jako vstupní Připojen koncový spínač
GND	Připojené k driveru krokového motoru, na svorky

	• DIR – • PUL– • EN–
PWM PIN 5	Připojené k driveru na svorku EN+ (neboli ENABLE) • Log.1 – s motorem je možné pohybovat • Log.0 – motor je odpojen od napájení V aplikaci není využito
PWM PIN 6	Připojené k driveru na svorku PUL+ (PULSE) • Přivedením pulsu s hodnotou log.1 se motor pootočí o jeden krok
PWM PIN 7	Připojené k driveru na svorku DIR+ (DIRECTION) Nastavení směru otáčení krokového motoru • Log.1 – po směru hodinových ručiček • Log.0 – proti směru hodinových ručiček

4.2 Hlavní využívané knihovny

4.2.1 AccelStepper

Jedná se o volně dostupnou knihovnu, která slouží k ovládání krokových motorů. Autorem této knihovny je Mike McCauley. Jedná se o knihovnu, která vychází z knihovny Arduino Stepper, která byla vhodná pro využití v jednodušších aplikacích s krokovým motorem. Knihovna AccelStepper byla rozšířena o nové užitečné funkce a přináší další zlepšení oproti původní knihovně.

Jako například:

- Podpora akcelerace a decelerace
- Podpora využití více na sobě nezávislých krokových motorů najednou
- Řízení motorů pomocí driveru i bez
- Podpora malých rychlostí

4.2.2 IR_GP2D12

Tato knihovna mi byla poskytnuta vedoucím práce jakou součást vzorového příkladu. Knihovna obsahující funkce, které jsou nezbytné pro získání skutečné hodnoty polohy kuličky pomocí optického snímače SHARP GP2D12. Knihovna též obsahuje defaultní převodní charakteristiku snímače, která je využita, pokud není definována externí převodní charakteristika. Součástí jsou též další knihovny, které slouží například pro filtrování hodnot.

4.2.3 BallPositionGP2D12

Tato knihovna mi byla poskytnuta vedoucím práce jakou součást vzorového příkladu. Knihovna byla vytvořena pro získávání pozice kuličky v případě, kdy je pozice snímána jedním snímačem GP2D12, který je umístěn na konci ramene. V této knihovně se nastavují parametry snímače a to konkrétně číslo analogového pinu, ke kterému je snímač připojen a referenční napětí A/D převodníku. Dále je zde také definována externí převodní charakteristika optického snímače, která byla změřena přímo pro využití na tomto konkrétním přípravku.

4.2.4 PID

Tato knihovna mi byla poskytnuta vedoucím práce jakou součást vzorového příkladu. Jedná se o softwarový PID regulátor polohy kuličky, který vrací hodnotu žádaného natočení ramene, na základě skutečné a požadované polohy kuličky. Při vytváření objektu třídy PID regulátoru se do konstruktoru předává hodnota, která definuje periodu vykonávání regulace. Dále je nutné knihovně předávat další parametry a to rozsah akční veličiny a jednotlivé konstanty Kp, Ti a Td.

4.2.5 BeamControlStepper

Tato knihovna mi byla poskytnuta vedoucím práce jakou součást vzorového příkladu. V této knihovně jsem provedl výrazné změny, aby mohla být využita po laboratorní přípravek, kterým se tato práce zabývá. Změny vychází především ze skutečnosti, že je využit jiný krokový motor, než pro který byla původní knihovna určena. Knihovna obsahuje veškeré funkce, které souvisí s ovládáním pohybu ramene a některé funkce, které jsou na pohybu závislé.

V této knihovně jsou inicializovány dvě převodní tabulky. První slouží pro zjištění počtu kroků pro krokový motor v závislosti žádaném úhlu natočení ramene a druhá pro zjištění skutečného úhlu natočení v závislosti na krocích motoru. Důvodem zavedení přepočtových tabulek je nelineární závislost mezi počtem kroků a úhlovým natočením ramene.

Dále zde dochází k nastavení následujících počátečních parametrů krokového motoru: Rychlost, Maximální rychlost a Akcelerace.

Důležité funkce v knihovně:

- bool TestBeamControl (void)

  Tato funkce využívá získanou informaci z koncového spínače, který je připojen k pinu číslo 8. Pokud není rameno na koncovém spínači, tak motor dostává příkaz k pohybu o jeden krok, směrem ke koncovému spínači, vzhledem k jeho aktuální pozici. Pokud dojde k sepnutí koncového spínače, nastaví se aktuální pozice krokového motoru na -550 kroků, což

odpovídá úhlu natočení ramene -18°. Poté motor dostává příkaz k pohybu na opačnou stranu. Jakmile je pozice krokového motoru 0 kroků, což znamená, že je rameno ve vodorovné poloze, motor zastaví. Tím získáme aktuální a přesnou informaci o natočení ramene. Nakonec funkce vrací proměnnou typu bool, která reprezentuje, zdali došlo ke správnému vykonání celé funkce.

- int AngleToSteps(float angle)

  Funkce, která slouží k převodu úhlu na počet kroků pro krokový motor a zalimitování úhlu na platný rozsah. Vstupní proměnnou funkce je hodnota úhlu. Natočení ramene je možné přibližně v rozsahu od -18° do 18°. Z důvodu ochrany před mechanickým poškozením, dochází k omezení rozsahu žádaného úhlu. Po omezení může žádaný úhel nabývat hodnot z intervalu od -15° do 15°. Na základě hodnoty omezeného žádaného úhlu získáme pomocí převodní tabulky žádaný počet kroků pro krokový motor, který odpovídá tomuto úhlu. Funkce vrací počet kroků.

- void SetBeamAngle(float angle)

  Jednoduchá funkce, jejíž vstupním parametrem je úhel. Je zadán příkaz na pohyb motoru na absolutní pozici, která odpovídá počtu kroků získaných zavoláním funkce AngleToSteps. Zavoláním funkce AngleToSteps získáme počet kroků pro krokový motor, aby bylo dosaženo požadovaného úhlu.

- float StepsToAngle(int steps_act)

  Jedná se o funkci, která se využívá pro převod počtu kroků krokového motoru na úhel. Vstupním parametrem jsou kroky krokového motoru. Primárně tato funkce slouží ke zjištění skutečného úhlu natočení ramene. Funkce vrací hodnotu aktuálního úhlu.

- float actual_angle (void)

  Jednoduchá funkce, bez vstupního parametru. Pouze voláme funkci StepsToAngle a předáváme jí pozici motoru v krocích. Funkce vrací hodnotu skutečného úhlu natočení ramene.

4.2.6 Communication

Tato knihovna mi byla poskytnuta vedoucím práce jakou součást vzorového příkladu. Jedná se o knihovnu, jejíž úkol spočívá ve zpracování přijatých dat a zavolání obslužné funkce

v závislosti na přijatém příkazu. Příkaz odeslaný z PC se ukládá to přijímacího bufferu. Jakmile jsou v přijímacím bufferu nějaká data, dochází k jejich vyčítání. Pokud dojde k přetečení přijímacího bufferu, tak je řetězec ignorován. Data jsou vyčítána a ukládána do jednoho řetězce, dokud není načten zakončovací znak, což značí, že byl příkaz přijat celý a může být dále zpracováván. Následuje parsování přijatých dat, neboli rozdělení přijatého řetězce na jednotlivé části příkazu. Nejdříve se oddělí první znak z řetězce, který určuje, jestli se jedná o příkaz čtení nebo zápisu a uloží se do řetězce *rw* . Následuje parsování zbytku řetězce, kdy se zpracovává část, která může být složena z malých nebo velkých písmen a ukládá se do nového řetězce s názvem *command*. Nyní mohou nastat dvě situace, narazí se na poslední znak a parsování je dokončeno nebo se narazí na číselnou část, která následně zpracovává a ukládá do nového řetězce pojmenovaného *data*. Zpracováním posledního znaku je parsování dokončeno. Dále je nutné identifikovat přijatý příkaz. To je provedeno tak, že porovnáváme řetězec *command* s příkazy z definovaného seznamu příkazů. Pokud nastane úplná shoda, získáme index příkazu, na základě kterého je volána obslužná funkce pro daný příkaz. Řetězce *rw* a případně *data* jsou vstupní parametry obslužné funkce.

V této knihovně je také definována funkce pro odesílání procesních dat. Voláním této funkce dochází k odeslání aktuálních hodnot definovaných parametrů ve formě textových hodnot oddělených středníkem.

4.2.7 CommandsProcessing

Tato knihovna mi byla poskytnuta vedoucím práce jakou součást vzorového příkladu. Tato knihovna obsahuje veškeré obslužné funkce pro komunikační příkazy. Každá obslužná funkce se skládá ze dvou částí: čtení a zápis. Využití jednotlivých částí závisí na dané funkci. Obslužné funkce pro některé příkazy vykonávají určitou činnost jak při čtení tak i při zápisu. Naopak některé reagují pouze na čtení nebo zápis.

Pokud přijde některý z příkazů pro zápis nové hodnoty jedné z konstant regulátoru, dojde k jejímu nastavení a poté se nová přijatá hodnota uloží do paměti EEPROM.

4.2.8 EEPROM_SETINGS.h

Soubor, ve kterém pouze jsou definovány adresy, pro uložení hodnot konstant regulátoru polohy a příznaku platnosti dat do paměti EEPROM. Příznak platnosti dat je velký 1 bajt. Jednotlivé adresy od sebe musí být dostatečně vzdáleny, protože hodnoty Kp, Ti a Td mohou být desetinná čísla a jsou tak ukládána. Pro uložení datového typu float do paměti EEPROM jsou využity čtyři po sobě jdoucí adresy po 8 bitech. Konstanta N je vždy celé číslo s maximální velikostí 16bitů.

4.2.9 BallOnBeam

Jedná se o hlavní zdrojový soubor celého programu. Při založení nového projektu vždy dojde k jeho vytvoření s názvem MainFile.cpp. V něm jsou vždy předpřipraveny dvě prázdné funkce setup() a loop(). Tyto dvě funkce musí být přítomny vždy, i když by byly prázdné a nevyužívaly se. Většinou jsou však funkce využity a je v nich zapsán kód, který se vykonává. Kdy funkce setup() se vykoná pouze jednou. Funkce loop() je nekonečná smyčka aplikace, která se vykonává neustále dokola.

Části kódu BallOnBeam.cpp:

- Připojení hlavičkových souborů využitých knihoven pomocí direktivy #include a zadefinování jednotlivých proměnných.

- Funkce void setup()

Jelikož se tato funkce vykoná pouze jednou, je výhodné sem zapsat tu část kódu, kterou není nutné opakovaně vykonávat. V tomto případě dochází v této funkci k inicializaci sériové linky a komunikace, nastavení filtrace měřené polohy kuličky, nastavení časovačů a nakonec volání funkce TestBeamControl(), kterou nastavíme rameno do výchozí polohy.

Po inicializace sériové linky dochází k vyčítání konstant regulátoru a příznaku platnosti dat z paměti EEPROM. Pokud příznak platnosti dat bude nabývat jiné hodnoty než, která je dána proměnnou PlatnyPriznak, dochází k nastavení defaultních hodnot regulátoru a uložení těchto hodnot do paměti. V opačném případě se nastaví hodnoty konstant regulátoru, které byly vyčteny z paměti EEPROM.

- Funkce void loop()

Nekonečná smyčka aplikace, která se vykonává neustále dokola, byla využita pouze pro obsluhu komunikace mezi laboratorním přípravkem a PC.

- Funkce void SetNewGainsForBallPositionController(PIDctrl::tPIDGains* pGains)

Funkce, která se vykonává pouze v případě, že přijatý příkaz z PC se týká změny hodnoty jednoho z parametrů PID regulátoru. Tato funkce zajistí změnu parametru za běhu programu.

- ISR(TIMER2_COMPA_vect)

Jedná se o obsluhu přerušení, které bylo vyvoláno přetečením časovače 2. Perioda přerušení je 502 μs. V tomto přerušení dochází k obsluze krokového motoru, který přejde na žádanou cílovou pozici.

- ISR(TIMER3_COMPA_vect)

Obsluha přerušení, vyvolaná přetečením časovače 3. Perioda přerušení je 4 ms. V obsluze přerušení se vykonává několik činností. Zjišťuje aktuální úhel natočení ramene, dochází k omezení žádané polohy kuličky, zjišťuje se skutečná poloha kuličky, probíhá regulace polohy kuličky a nakonec ještě dochází k odeslání procesních dat.

4.3 Funkce přípravku jako celku

Nejdůležitější je přípravek připojit přívodním kabelem k napájení 230V 50Hz a zároveň propojit řídící desku Arduino s PC pomocí USB kabelu typu A-B. Po vykonání těchto úkonů rameno postupně začne sjíždět až na koncový spínač. Po sepnutí koncového spínače rameno přejde do vodorovné polohy. Série těchto dvou akcí se provádí pouze po spuštění, protože nevíme jaké je aktuální natočení ramene a získáme tím informaci o počáteční pozici. Následovně mohou být vykonávány další funkce přípravku.

V tomto okamžiku může nastat více situací a to podle počátečních parametrů programu.

Regulace může být zastavena. V tomto stavu nedochází k odesílání procesních dat do PC a rameno je ve vodorovné tedy výchozí pozici. Komunikace je funkční. Lze tedy pomocí odesílaných příkazů vyčítat aktuální hodnoty a zapisovat nové hodnoty podporovaných parametrů. Dále je možné pomocí příkazu *RUN* regulaci zapnout.

Druhá možnost nastává, pokud je regulace povolena. V tomto stavu přípravek do PC odesílá procesní data. Následná funkce přípravku pak závisí na zvoleném módu. Pokud je zvolen MOD1, což je první možnost, tak je aktivní zpětnovazební regulace polohy kuličky. Při níž dochází k výpočtu žádaného úhlu natočení ramene, kde vstupními hodnotami jsou žádaná a skutečná poloha kuličky. Krokovým motorem je poté nastavován žádaný úhel natočení ramene. Hodnota žádaného úhlu natočení se mění tak, aby skutečná poloha kuličky byla rovna poloze žádané. V tomto stavu je komunikace plně funkční a lze zadávat novou žádanou hodnotu polohy kuličky pomocí příkazu wPZADxxx_. Druhou možností je MOD0, při kterém je

zpětnovazební regulace vypnuta a pomocí příkazů z PC lze nastavovat úhel natočení ramene. Regulaci je možné zastavit příkazem wSTOP_.

4.4 Využité zdroje mikrokontroléru ATmega2560

Program, který je nahraný do řídící desky Arduino, využívá několik zdrojů mikrokontroléru ATmega2560. Ty jsou využity pro různé funkce, které jsou nezbytné pro správnou funkci přípravku. V následující tabulce jsou vypsány ty nejdůležitější.

Tabulka 4.2 Využité zdroje ATmega2560

Zdroj mikrokontroléru ATmega2560	Funkce

Časovač 0 (Timer0)	• 8 bitový časovač
	• Využitý k časování vlastním Arduinem
	• Umožnuje využití funkcí:
	  • delay() - slouží k pozastavení programu, vstupním parametrem je hodnota v milisekundách
	  • millis() – funkce vrací časovou hodnotu od spuštění programu v milisekundách, k přetečení dojde přibližně za 50 dní
	  • micros() - funkce vrací časovou hodnotu od spuštění programu v mikrosekundách, k přetečení dojde přibližně za 70 minut
	  • delayMicroseconds() - slouží k pozastavení programu, vstupním parametrem je hodnota v mikrosekundách

Časovač2 (Timer2)	• 8 bitový časovač • Využitý k časování pro obsluhu krokového motoru
Časovač3 (Timer3)	• 16 bitový časovač • Využitý k časování pro obsluhu regulační smyčky
A/D převodník	• Optický snímač polohy je analogový • Využit pro převod z analogových hodnot na digitální
UART0	• Využit ke komunikaci s PC
Přerušení od Časovače2	• Při přetečení Časovače2 – časuje do 80H neboli do 128 dekadicky • Perioda přetečení 502 μs

	• Obsluha pohybu krokového motoru – perioda udává maximální rychlost krokování motoru
Přerušení od Časovače3	• Při přetečení Časovače3 • Perioda vykonávání 4 ms
Přerušení od sériové linky UART0	

4.5 Časování aplikace

Obsluha krokového motoru je prováděna při přerušení, které nastává vlivem přetečení časovače číslo 2. Motor je ovládán pomocí driveru, který podporuje mikrokrokování. V plném kroku má motor rozlišení 200 kroků na 360°.

Tomu odpovídá délka plného kroku:

$$délka\ plného\ kroku = \frac{360°}{200 kroků} = 1{,}8° \quad (1)$$

Tento krok by byl však velice hrubý a nevhodný pro tuto aplikaci. Proto bylo využito mikrokrokování. Kdy je délka kroku rozdělena na menší části (1/2, 1/4, 1/8, 1/16, 1/64, 1/128, 1/256). Bylo zvoleno mikrokrokování 1/4.

Regulace polohy kuličky se vykonává každé 4 ms v přerušení časovače číslo 3. Zároveň dochází k odesílání procesních dat, a to při každém druhém přerušení. Takže k odesílání dat dochází každých 8 ms.

V neposlední řadě závisí dynamika regulace na parametrech nastavených v programu. Jedná se o parametry rychlost, maximální rychlost a akcelerace. Kdy těmito parametry jsme schopni ovlivnit rychlost motoru v krocích za sekundu.

V současném stavu jsou nastaveny parametry následovně:

- Akcelerace 15 000
- MAX_SPEED 3000

4.5.1 Změna hodnot, které ovlivňují dynamiku regulace

Jednotlivé hodnoty, které mají vliv na dynamiku regulace, je možné změnit přepsáním hodnot v programu.

Hodnoty, které lze změnit:

- Rychlost krokování lze nastavit změnou periody přetečení časovače 2. Nastavení těchto registrů se provádí ve funkci setup() v hlavním zdrojovém souboru s názvem BallOnBeam.cpp.

```
//Nastaveni 8bit casovace Timer2 - krokovy motor
        TCCR2A = 0x02;
        TCCR2B = 0x04;
        OCR2A = 0x80/K_STEPPING;       // 80H neboli 128 dekadicky
        TIMSK2 = 0x02;
```

- Perioda vykonávání regulační smyčky je dána konstantou CONTROL_PERIOD. Tuto hodnotu lze změnit v souboru BallOnBeam.h. Aktuální hodnota je 4 ms.

```
#define CONTROL_PERIOD          4  //perioda vykonavani regulatoru [ms], max. 268 ms
```

- Případně je možné změnit i periodu odesílání procesních dat. Změnu lze provést v obsloužení přerušení od časovače 3 - ISR(TIMER3_COMPA_vect). Kde je nutné změnit děličku pro odesílání hodnot, ve kterém se volá funkce pro odesílání procesních dat. Perioda odesílání procesních dat po USB je odvozena od periody vykonávání regulátoru (CONTROL_PERIOD) podělením děličkou.

  Aktuální hodnota děličky je 10.

```
if (!(loopCounter % 10))
{
   SendProcessDataAsCSV();
}
```

- Možné je změnit i parametry ovládání motoru a zvýšit tak rychlost krokování motoru. Tyto parametry jsou zapsány v souboru BeamControlStepper.h

```
const float akcelerace = 15000;
const float MAX_SPEED = 3000;
```

4.6 Testování rychlosti krokového motoru

Po vytvoření firmwaru došlo k jeho otestování na přípravku. Kdy probíhaly různé pokusy s nastavováním parametrů akcelerace a maximální rychlosti. Vznikl tak požadavek na určení vzájemného vztahu mezi nastavenou a skutečnou hodnotou akcelerace a maximální rychlosti.

Před praktickým měřením bylo nejdříve nutné provést rozbor výpočtu rychlosti pohybu krokového motoru, který se provádí v knihovně AccelStepper.

4.6.1 Krokový motor rychlost pohybu – teoretický rozbor

Při prvním spuštění dochází k určení polohy ramene definovanou sekvencí úkonů. Nejdříve rameno sjíždí na koncový spínač s konstantní rychlostí. Poté je přenastaveno do vodorovné polohy nerovnoměrnou rychlostí, tedy s využitím akcelerace a decelerace.

Pokud je zvoleno pouze ovládání úhlu natočení tak se rameno pohybuje stále konstantní rychlostí. Protože funkce pro výpočet rychlosti je vykonána pouze jednou při zadání žádosti na změnu úhlu natočení ramene.

Pokud je zapnuta zpětnovazební regulace, tak je při každém přerušení od časovače 3 vykonávána funkce pro změnu cílové polohy krokového motoru. Při vykonávání této funkce je volána i funkce pro přepočet rychlosti pro následující krok. Motor tedy využívá akceleraci respektive deceleraci.

Při inicializaci krokového motoru voláme funkce pro nastavení maximální rychlosti a akcelerace. V těchto dvou funkcích jsou počítány hodnoty proměnných, které jsou důležité pro přepočet rychlosti.

Nastavení maximální rychlosti

K nastavení maximální rychlosti se využívá funkce setMaxSpeed(float speed). Kdy hodnota maximální rychlosti slouží k výpočtu minimální délky kroku v mikrosekundách.

$$\_c_{min} = \frac{1000000}{maxSpeed}\ [\mu s] \qquad (2)$$

kde

$\_c_{min}$ minimální délka kroku [μs]

Následuje přepočet počítadla kroků pro výpočet rychlosti podle následujícího vztahu, avšak pouze v případě, že hodnota tohoto počítadla je větší než nula. Poté je volána funkce pro přepočet rychlosti.

$$\_n = \frac{speed^2}{2 \cdot acceleration} \qquad (3)$$

kde

_n počítadlo kroků pro výpočet rychlosti

Jelikož je funkce volána při inicializaci, tak je hodnota počítadla pro výpočet rychlosti nula a takže nedochází k jeho přepočtu ani volání funkce pro přepočet rychlosti.

Nastavení akcelerace

K nastavení se využívá funkce setAcceleration(float acceleration). Funkce slouží pro nastavení hodnoty akcelerace, respektive decelerace, kterou bude motor využívat. V této funkci dochází k výpočtu s odmocninou a trvá tak nezanedbatelnou dobu. Nedoporučuje se tedy časté volání funkce.

Zároveň znovu dochází k přepočtu hodnoty počítadla kroků pro výpočet rychlosti.

$$\_n_i = \_n_{i-1} \cdot \left(\frac{acceleration_{i-1}}{acceleration_i}\right) \tag{4}$$

Jelikož je funkce pro nastavení hodnoty akcelerace volána pouze jednou a to při inicializaci, je původní hodnota akcelerace i počítadla kroků pro výpočet rychlosti nulová. Proto je i výsledná hodnota počítadla kroků pro výpočet rychlosti rovna nule.

Dále následuje výpočet počáteční velikosti kroku v mikrosekundách.

$$\_c_0 = 0{,}676 \cdot \sqrt{\frac{2}{acceleration}} \cdot 1000000 \ \ [\mu s] \tag{5}$$

kde

$\_c_0$ počáteční velikost kroku [μs]

Na závěr je volána funkce na přepočet rychlosti.

Výpočet nové rychlosti

K výpočtu rychlosti pro následující krok slouží funkce computeNewSpeed(). Tato funkce musí být využita, pokud chceme, aby se motor využíval akceleraci respektive deceleraci.

Nejdříve se zjistí hodnota distanceToGo, neboli vzdálenost mezi současnou cílovou polohou v krocích. Poté následuje výpočet hodnoty stepsToStop dle následujícího vztahu.

$$stepsToStop = \frac{speed^2}{2 \cdot acceleration} \tag{6}$$

Na základě aktuálního stavu počítadla kroků pro výpočet rychlosti, hodnot distanceToGo, stepsToStop a směru otáčení se určí nová hodnota počítadla kroků pro výpočet rychlosti _n. Kdy nová hodnota bude nabývat jedné z hodnot -_n nebo – stepsToStop.

Před výpočtem rychlosti je nutné provést výpočet délky minulého kroku v mikrosekundách.

$$\_c_n = \_c_{n-1} - \left(\frac{2*\_c_{n-1}}{4\cdot\_n+1}\right) \text{ [µs]} \quad (7)$$

kde

$\_c_n$ délka minulého kroku [µs]

Následně dochází k porovnání, jestli není tato hodnota menší jak minimální délka kroku $\_c_{min}$. Pokud by byla menší, tak by došlo ke změně:

$$\_c_n = \_c_{min} \quad (8)$$

V případě, že by motor stál, tedy že hodnota počítadla pro výpočet rychlosti by byla nulová, a byl přijat požadavek na pohyb tak platí:

$$\_c_n = \_c_0 \quad (9)$$

Pro výpočet rychlosti pak platí následující vztah

$$\_speed = \frac{1000000}{\_c_n} \ [krok\text{ů}/s] \quad (10)$$

Zároveň se výší hodnota počítadla kroků pro výpočet rychlosti o 1.

Pokud je žádaný směr otáčení po směru hodinových ručiček tak platí:

$$\_speed = -\_speed \quad (11)$$

Z předchozích vztahů tedy vyplývá, že hodnotou akcelerace, nastavujeme rychlost pro první krok motoru po zastavení.

4.6.2 Měření - Testování rychlosti

Jedním z důležitých úkolů pro správné pochopení funkce laboratorního přípravku bylo nutné zjistit jaký je vzájemný vztah mezi zadávanými hodnotami akcelerace a maximální rychlostí ve firmwaru a skutečnými hodnotami, se kterými se rameno přípravku pohybuje.

Tento vzájemný vztah byl zjištěn provedením praktického měření. Měření bylo provedeno pomocí osciloskopu, kdy byly sledovány průběhy PUL+ a DIR+ na svorkách Driveru. Tedy pulsní elektrický signál (PUL+), kde úroveň logické jedničky způsobí pohyb motoru o jeden krok a signál směru otáčení krokového motoru.

Pohyb konstantní rychlostí

Jak je již známo po spuštění přípravku nejdříve rameno zařízení sjíždí konstantní rychlostí na koncový spínač. Kdy je z řídící jednoty odesílán příkaz driveru, aby odeslal motoru puls, tedy povel k provedení jednoho kroku dokud řídící jednotka neobdrží informaci, že krokový motor sepnul koncový spínač.

Teoretický výpočet

Výpočet bude proveden podle vztahů, které jsou uvedeny v kapitole Krokový motor rychlost pohybu – teoretický rozbor.

Hodnoty zadané ve firmwaru

Akcelerace = 5000

Max_speed = 500

Pro délku počátečního kroku v mikrosekundách platí vztah (5)

$$\_c_0 = 0{,}676 \cdot \sqrt{\frac{2}{acceleration}} \cdot 1000000 = 0{,}676 \cdot \sqrt{\frac{2}{5000}} \cdot 1000000 = 13520\mu s$$

Motor dostává příkaz k vykonání jednoho kroku a poté zastaví, dokud není sepnut koncový spínač. Platí tedy vztah (9)

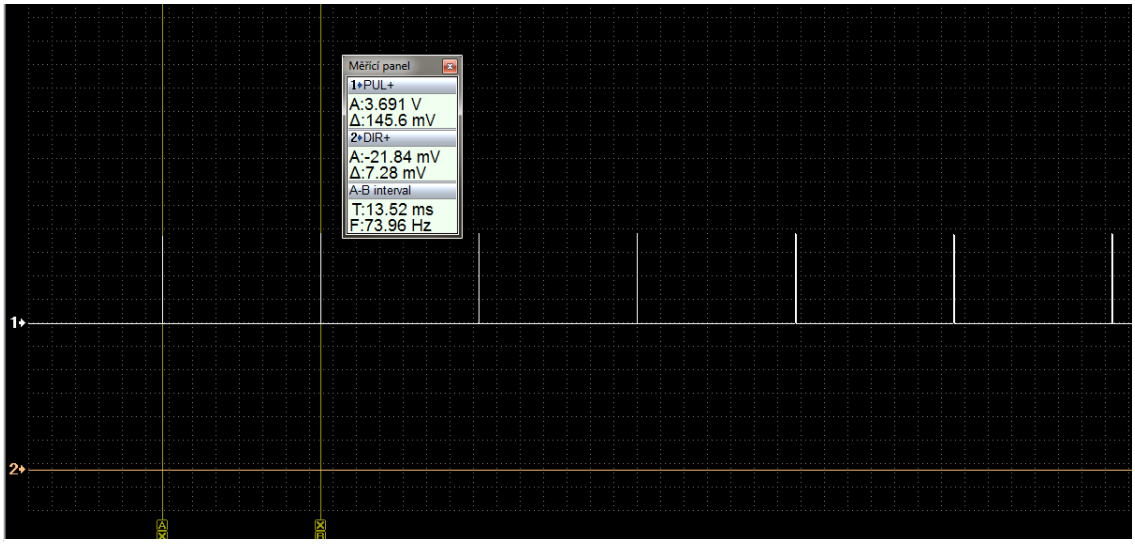
$\_c_n = \_c_0$

Kde $\_c_0$ je délka následujícího kroku v mikrosekundách.

Pro výslednou rychlost krokování tedy platí vztah (11)

$$\_speed = \frac{1000000}{c_n} = \frac{1000000}{13520} = 73{,}96 krok\text{ů}/s$$

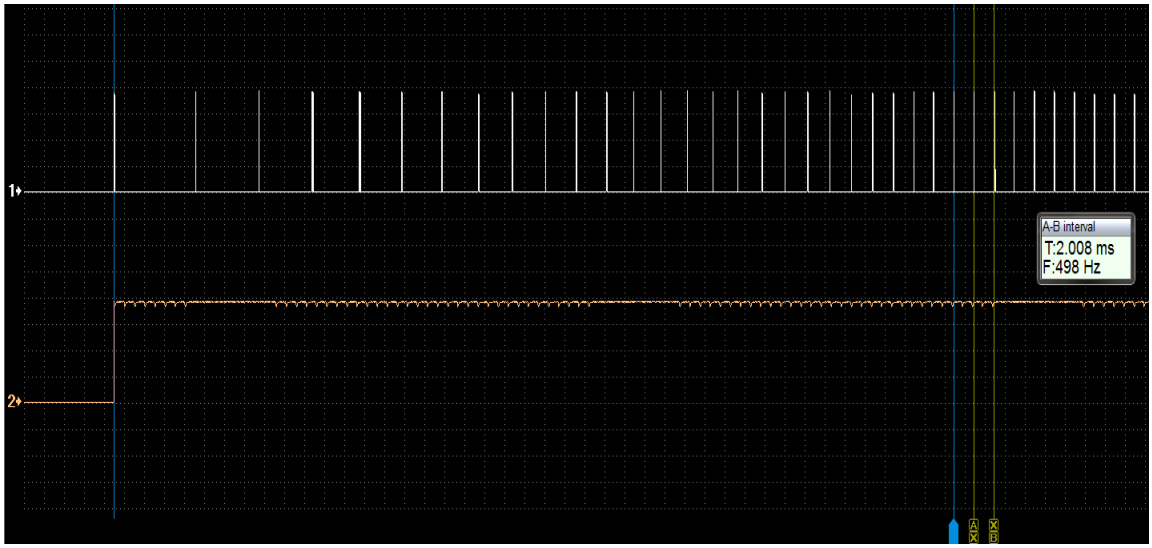
Rozlišení os naměřených průběhů (Obrázek 8 a Obrázek 9)
Osa x - 2 ms/dílek
Osa y - 1 V/dílek (platí pro kanál 1 i 2)



Obrázek 8 Průběh impulsů na svorce PUL+ (CH1) a průběh na svorce DIR+ (CH2)

Na Obrázku 8 je naměřený průběh, kdy se krokový motor otáčel konstantní rychlostí krokování na koncový spínač. Na kanálu 1 je pulsní průběh, na svorce PUL+, s frekvencí 73,96Hz což odpovídá vypočítané rychlosti krokování.

4.7 Pohyb se zrychlením



Obrázek 9 Průběh zrychlení, PUL+ (CH1), DIR+ (CH2)

Než rameno sepne koncový spínač, frekvence impulsů (CH1) je konstantní. Po sepnutí koncového spínače se rameno přenastaví do vodorovné polohy zrychleným pohybem. A tedy

se interval mezi impulsy zkracuje. Po sepnutí koncového spínače se tedy změní směr otáčení motoru, což dokazuje i změna úrovně napětí na svorce DIR+ (CH2).

Modré záložky, které jsou v průběhu (Obrázek 9), vymezují interval, ve kterém dochází ke zvyšování rychlosti. Za tímto intervalem je frekvence impulsů konstantní a to 498Hz. Z důvodu omezení nastavenou hodnotou maximální rychlosti, která byla 500ot/s.

Výpočet zrychlení

Počáteční frekvence: 73,96 Hz

Koncová frekvence: 498 Hz

Délka akcelerace: 83,94ms

$$akcelerace = \frac{koncová\ frekvence - počáteční\ frekvence}{délka\ akcelerace} = \frac{498 - 73{,}96}{83{,}94 \cdot 10^{-3}} = 5051\ krok\text{ů}/s^{-2}$$

Toto praktické měření bylo provedeno s různým nastavením hodnot akcelerace a maximální rychlosti. Pokud se motor pohyboval konstantní rychlostí, odpovídaly teoreticky vypočtené hodnoty naměřeným. V případě zrychleného pohybu při vyšších hodnotách akcelerace bylo složitější přesné určení koncové frekvence a doby akcelerace. Vypočítané hodnoty skutečné akcelerace se tedy od zadaných lišily v řádu stovek kroků/s^{-2} .

5. VYUŽITÍ VÝVOJOVÉHO PROSTŘEDÍ LABVIEW

Zpočátku byla k ovládání přípravku využívána jednoduchá aplikace Terminal. Pomocí této aplikace bylo možné přípravek plně ovládat, ale jednalo se o komplikované řešení, kdy uživatel musel jednotlivé příkazy napsat a odeslat pomocí nějaké aplikace, která slouží pro komunikaci. Zároveň musel uživatel znát nebo mít k dispozici jednotlivé příkazy komunikačního protokolu.

Řešením tedy bylo vytvořit ovládací software, což je druhým hlavním bodem této práce, po úpravě firmwaru laboratorního přípravku. Kdy hlavní myšlenkou byla rychlá a pohodlná obsluha zařízení. Bez nutnosti znalosti jednotlivých komunikačních příkazů.

Nová ovládací aplikace pro laboratorní přípravek, kterým se tato práce zabývá, bude vytvořena pomocí vývojového prostředí LabVIEW.

5.1 Vývojové prostředí LabVIEW

Název vývojového prostředí LabVIEW vznikl jako zkratka pro Laboratory Virtual Instrument Engineering WorkBench. Jedná se o profesionální produkt americké firmy National Instrument. K programování, se v tomto vývojovém prostředí využívá grafického jazyka G. Zdrojový kód, tedy není psaný v textové podobě, ale je vytvářen pomocí grafických bloků, které lze mezi sebou vzájemně datově propojit. Vzniká tak blokový diagram.

Tento nástroj má široké uplatnění v různých oborech a lze využívat k řešení komplexních problémů. Uplatnění nachází například v oblasti zpracování signálů, automatizace a řízení. Vzniknout tak může rozsáhlá aplikace tvořená systémy a podsystémy, které mezi sebou komunikují. Zároveň lze zajistit spojení a komunikaci s různým hardwarem.

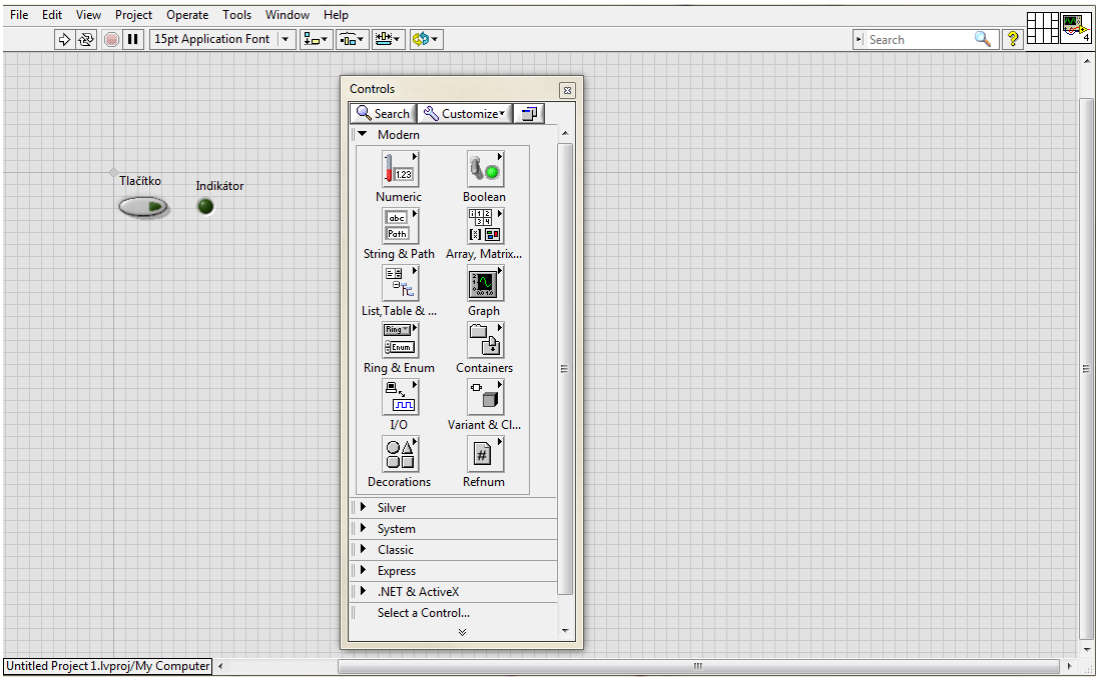
Soubory, které obsahují nějaký kód mají koncovku *.VI. Jedná se o zkratku z anglického názvu pro tento program – Virtual Instrument neboli virtuální přístroj. Aplikace lze rozdělit na dvě základní části a to Čelní panel (Front Panel) a Blokový diagram (Block Diagram).

5.1.1 Čelní panel (Front panel)

Čelní panel slouží jako interface mezi uživatelem a vlastním programem. Na čelním panelu mohou být umístěny různé prvky. Tyto prvky lze rozdělit z hlediska funkce na vstupní neboli ovládací (tlačítka, různé prvky pro nastavení hodnoty) a výstupní neboli zobrazovací (tabulky, grafy). Nabídka dostupných prvků se zobrazí na paletě Controls po kliknutí pravým tlačítkem myši v oblasti čelního panelu. Veškeré prvky, které jsou umístěny na čelní panel, mají svůj ekvivalent v blokovém diagramu, se kterým svou pevně spojeny. Tedy smazáním prvku v jedné části dojde ke smazání jeho ekvivalentu ve druhé.

Jednotlivé prvky mohou být libovolně uspořádány v prostoru čelního panelu. Výsledná podoba tak čistě závisí na programátorovi.

Ukázka čelního panelu a palety Controls je uvedena na Obrázku 10.



Obrázek 10 Čelní panel, paleta dostupných prvků - Controls

5.1.2 Blokový diagram (Block diagram)

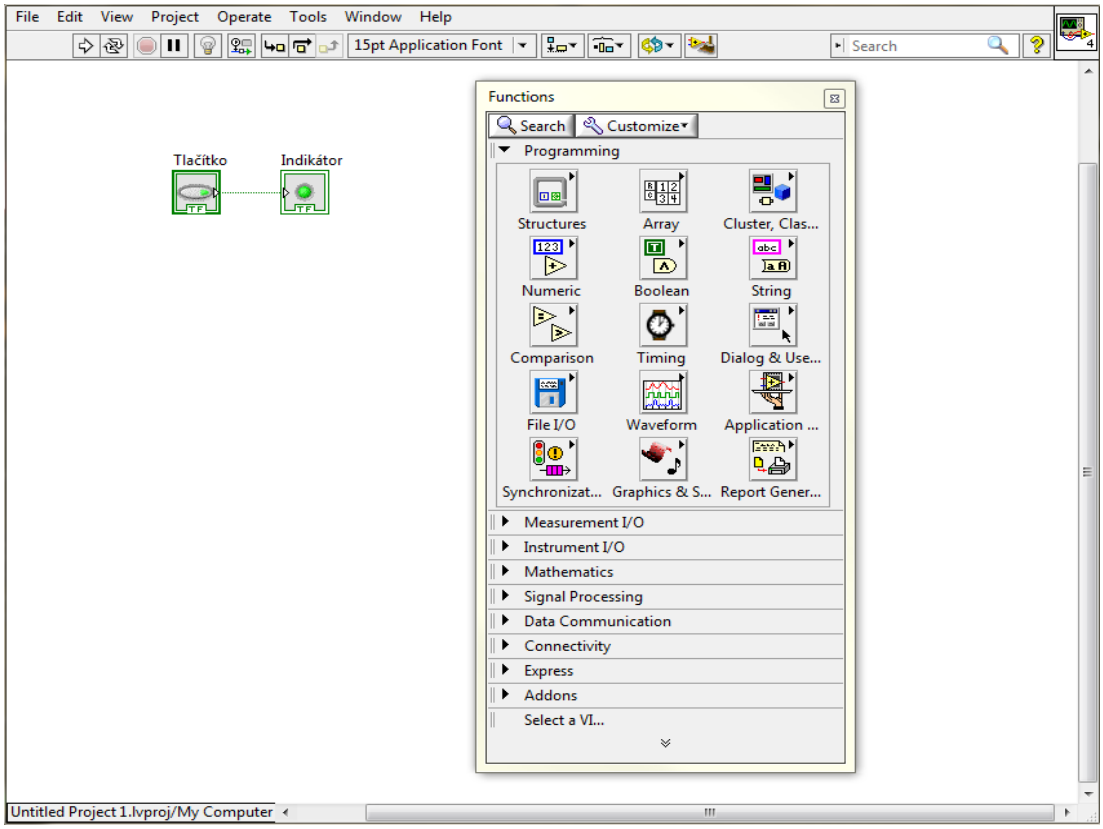
Druhou částí aplikace je blokový diagram, ve které programátor vytváří vlastní algoritmus programu. Vlastní tvorba programu spočívá ve spojování jednotlivých funkčních bloků. Dostupné funkční bloky lze vybrat z palety Function, která se zobrazí po kliknutí pravým tlačítkem myši kdekoli v prostoru blokového diagramu. Spoji mezi bloky se přenáší informace. Barva spoje závisí na datovém typu informace.

Tabulka 5.1 Základní typy datových spojů a jejich barevné označení

Datový typ	Barva spoje
Binární	Zelená
Celočíselný	modrá
Číselný s pohyblivou čárkou	oranžová
Znakový řetězec	růžová

Blokový diagram je přímo spustitelný kód, který je překládán již během tvorby, což umožňuje okamžitou indikaci některých chyb. Například spojení neslučitelných datových typů.

Na Obrázku 11 je ukázka blokového diagramu a palety dostupných funkčních bloků. Bloky Tlačítko a Indikátor vznikly při vytvoření prvků na čelním panelu (Obrázek 10).



Obrázek 11 Blokový diagram a paleta dostupných funkčních bloků

V blokovém diagramu programu se často využívají podprogramy neboli SubVI. Podprogramy se vytvoří sdružením například prostorově rozlehlých nebo často se opakujících částí programu. Získáme tak blok s danými vstupy a výstupy, který lze využít i vícekrát, což vede ke zpřehlednění celého programu. Každé SubVI má poté vlastní čelní panel a blokový diagram. Pro vytvoření SubVI se označí určitá část programu a na záložce Edit stačí vybrat Create SubVI.

Ikonu SubVI lze následně upravovat a zobrazit například název, což může pomoci s následnou identifikací v rozlehlejších programech.

Obrázek 12 Automatická podoba bloku SubVI (vlevo) a upravená (vpravo)

5.2 Využité struktury a VI

Aplikace pro ovládání přípravku musí zvládat několik hlavních funkcí, mezi které patří vizualizace regulačního pochodu, logování přijatých dat do souboru, nastavení konstant regulátoru a zpětnovazební řízení polohy kuličky nebo úhlu natočení ramene. Jedná se tedy o složitější aplikaci, která vyžaduje využití složitější programové architektury. Zároveň bude nutné uchovávat větší množství dat, ke kterým by měl být snadný přístup.

5.2.1 Funkční globální proměnná – FGV

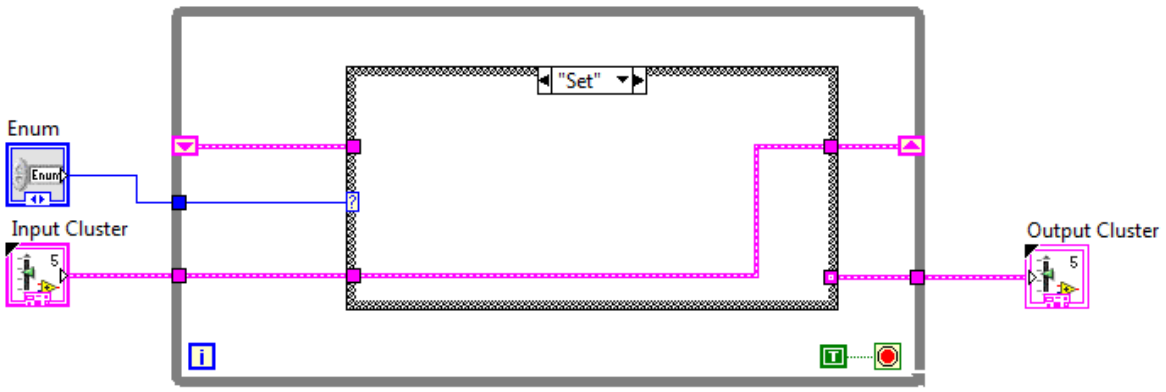
Jedna se o VI, které se využívá k ukládání dat. Využívá se neinicializovaného shift registru. Využití FGV přináší řadu výhod. Mezi hlavní patří možnost přístupu k uloženým datům kdekoliv v aplikaci bez nutnosti přivedení datového vodiče. Pokud dojde k volání tohoto VI, vykoná se pouze jednou a poté se ukončí.

V případě je-li nutné uchovávat velké množství dat, je výhodné využít cluster.

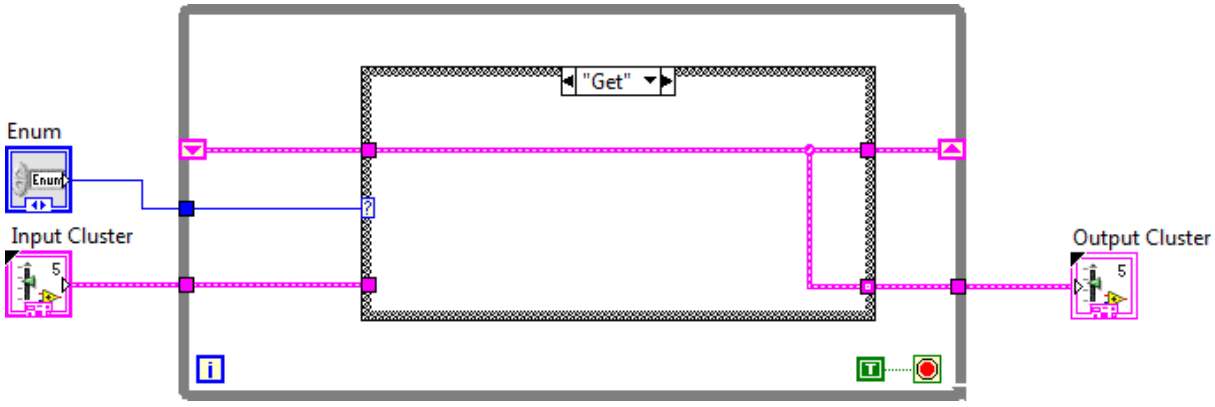
Prvky FGV:

- Smyčka While
- Neinicializovaný shift registr
- Case struktura
- Enum Control
- Vstupní a výstupní cluster

Enum Control slouží k určení stavu. V toto případě se bude jednat o dva stavy Set – nastav data a Get – získej data. Zapojení Blokového diagramu FGV pro jednotlivé stavy je na Obrazcích 15 a 16.

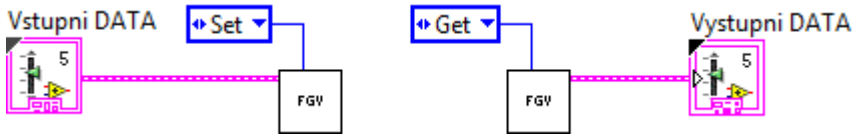


Obrázek 13 FGV - Uložení nových hodnot



Obrázek 14 FGV – Získání uložených hodnot

Využití funkční globální proměnné v aplikaci pak vypadá následovně.



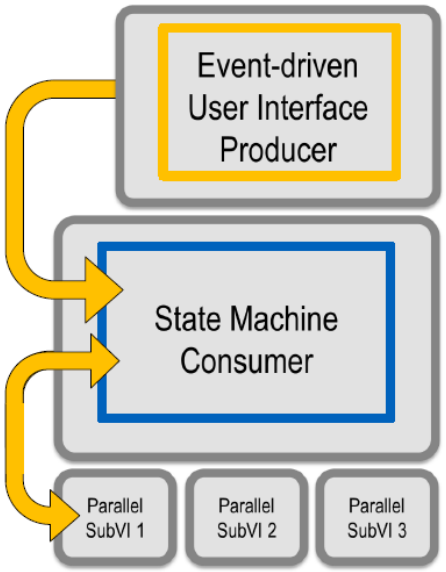
Obrázek 15 Využití FGV – nastavení hodnot (vlevo), získání hodnot (vpravo)

Toto VI má dva vstupy a jeden výstup (Obrázek 15). Prvním vstupem je Enum, který musí být připojen vždy, protože s jeho pomocí vybíráme, jestli chceme data uložit nebo získat A druhým je Cluster (Vstupni DATA), který musí obsahovat všechny prvky ve stejném pořadí jako Input Cluster v FGV (Obrázek 13 a 14). Pokud chceme uložená data získat, využívá se výstupní terminál, který je na pravé straně tohoto VI.

5.2.2 Architektura Queued State Machine – Producer Consumer - QSM-PC

Jedná se o jednu ze základních architektur pro střední nebo velké, pokročilé aplikace. Producer/Consumer se využívá v případech, kdy je nutné vykonávat dva procesy paralelně v jednom čase, a které mezi sebou mají komunikovat. Kdy ke komunikaci se využívají fronty. QMS-PC se využívá, pokud je nutné pomocí události od uživatele (smyčka Producer) ovládat sled událostí ve smyčce Consumer.

Ukázka základní architektury QSM-PC je zobrazena na Obrázku 16.



Obrázek 16 Architektura QSM-PC [5]

Funkce bloků:

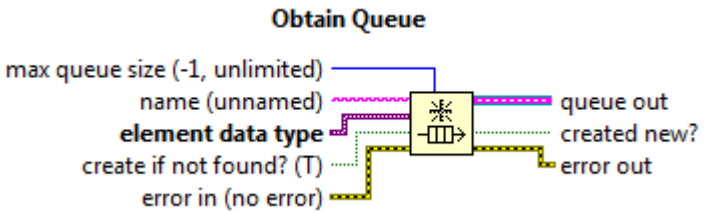
- Event struktura (Producer) přijímá události od uživatele
- Producer přidává příkazy a data do fronty
- State Machine (Consumer) zpracovává příkazy a data z fronty a v závislosti na nich vykonává určitou činnost
- Paralelní SubVIs komunikují s Consumer pomocí referencí na fronty

5.3 Základní bloky pro práci s frontami

Pro komunikace se tedy využívá front. V blokovém diagramu je možné využít několik základních bloků pro práci s frontami.

5.3.1 Obtain Queue

Blok, který vrací referenci na frontu.



Obrázek 17 Obtain Queue

Tabulka 5.2 Obtain Queue Vstupy

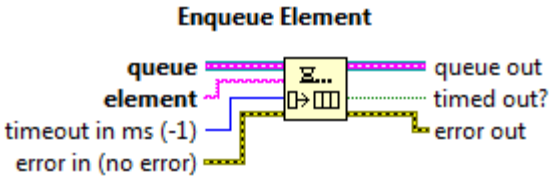
Vstupy	
Max queue size	Maximální velikost fronty, defaultně -1 tedy neomenená velikost
Name	Název fronty, kterou chceme získat nebo vytvořit
Element data type	Povinné, Datový typ vkládaných prvků do fronty
Create if not found (T)	Vytvořit frontu pokud neexistuje – defaultně True tedy vytvořit
Error in	

Tabulka 5.3 Obtain Queue Výstupy

Výstupy	
Queue out	Vrací referenci na získanou nebo nově vytvořenou frontu
Create new?	Pokud byla vytvořena nová fronta, vrací true jinak false
Error out	

5.3.2 Enqueue Element

Přidání elementu na konec fronty.



Obrázek 18 Enqueue Element

Tabulka 5.4 Enqueue Element Vstupy

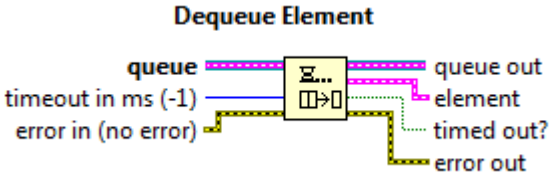
Vstupy	
queue	Povinné, Reference na frontu, do které chceme přidat element
element	Povinné, Element, který chceme přidat na konec fronty
Timeout in ms	Timeout, který se čeká, pokud ve frontě není místo, defaultně -1 tj. nenastane nikdy
Error in	

Tabulka 5.5 Enqueue Element Výstupy

Výstupy	
Queue out	Vrací nezměněnou referenci frontu
Timed out	TRUE - Pokud nevzniklo místo ve frontě před vypršením timeoutu
Error out	

5.3.3 Dequeue Element

Vyjme a vrací element z fronty.



Obrázek 19 Dequeue Element

Tabulka 5.6 Dequeue Element Vstupy

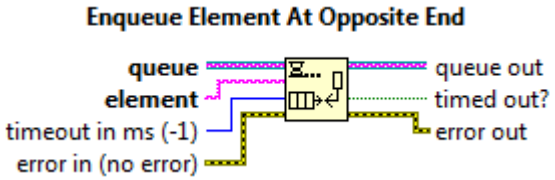
Vstupy	
queue	Povinné, Reference na frontu, ze které chceme vyjmout element
Timeout in ms	Timeout, který se čeká, na element pokud je fronta zatím prázdná, defaultně -1 tj. nenastane nikdy
Error in	

Tabulka 5.7 Dequeue Element Výstupy

Výstupy	
Queue out	Vrací nezměněnou referenci frontu
Timed out	TRUE - Pokud byla fronta prázdná a nebyl tak vyjmut element před vypršením timeoutu
Error out	

5.3.4 Enqueue Element At Opposite End

Přidání elementu na začátek fronty.



Obrázek 20 Enqueue element at opposite end

Tabulka 5.8 Enqueue element at opposite end Vstupy

Vstupy	
queue	Povinné, Reference na frontu, do které chceme přidat element
element	Povinné, Element, který chceme přidat na začátek fronty
Timeout in ms	Timeout, který se čeká, pokud ve frontě není místo, defaultně -1 tj. nenastane nikdy
Error in	

Tabulka 5.9 Enqueue element at opposite end Výstupy

Výstupy	
Queue out	Vrací nezměněnou referenci frontu
Timed out	TRUE - Pokud nevzniklo místo ve frontě před vypršením timeoutu
Error out	

5.4 Využitá šablona QSM-PC

Pro tuto aplikaci byla využita šablona QSM template, která mi byla poskytnuta vedoucím práce. Struktura projektu je na Obrázku 21. Základem jsou dvě složky Master a Slave.

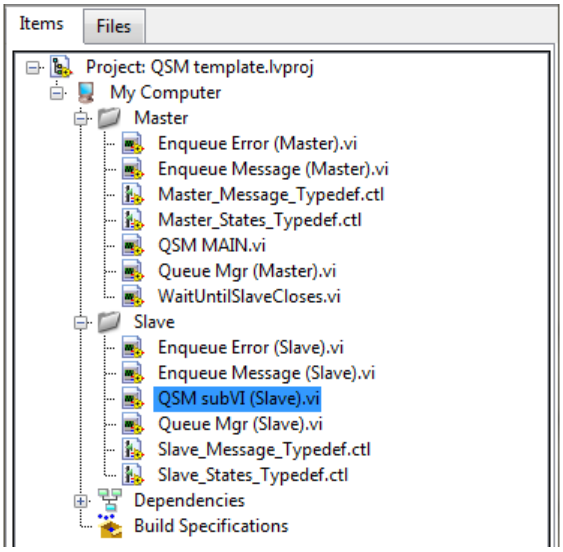
Ve složce Master je hlavní zdrojový soubor aplikace s názvem QSM MAIN.vi a několik dalších souborů důležitých pro správnou funkci. Popis hlavních souborů je v jedné z následujících podkapitol. Blokový diagram QSM MAIN.vi je na Obrázku 22. Data jsou uložena v clusteru, kdy datový vodič musí procházet přes stavový automat. Což bylo následně předěláno a byla využita funkční globální proměnná - FGV. Důvod zavedení FGV byl ten, aby měla smyčka s event strukturou přístup k aktuálním datům ve smyčce Consumer. Čímž je možné na jednoduché události, které vyžadují sdílená data, reagovat přímo v event struktuře. Není tak nutné přeposílat tyto požadavky do Consumera, kde by byly dále zpracovány.

Složka Slave obsahuje soubory, důležité pro chod paralelního SubVI. Popis jednotlivých souborů je v jedné z následujících podkapitol. Hlavní soubor paralelního SubVI je pojmenován jako QSM subVI (Slave). Paralelní SubVI je stavový automat, blokový diagram je na Obrázku 23.

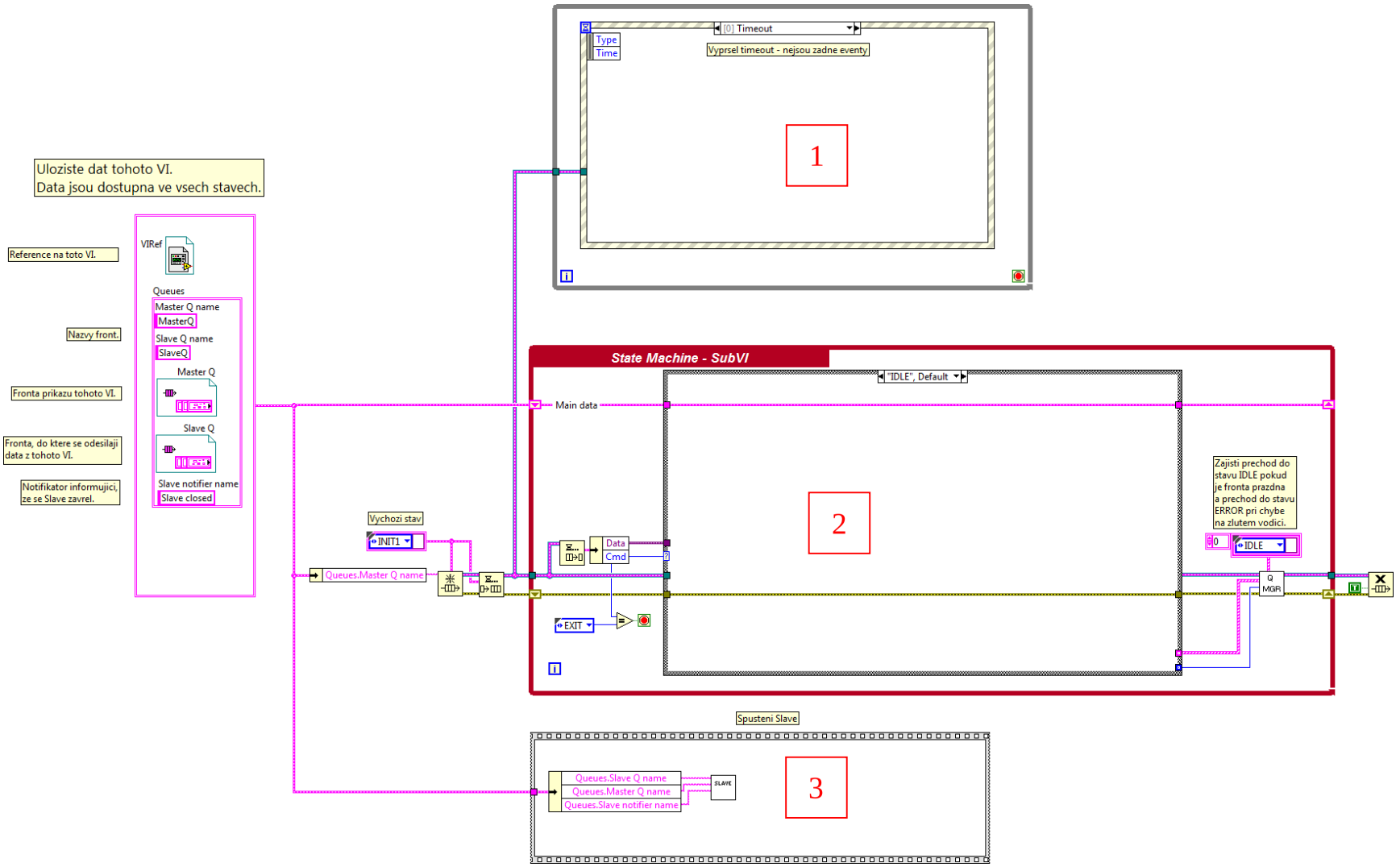
Master i Slave mají svou vlastní frontu, která je využívána pro komunikaci. Kdy názvy těchto front jsou definovány pouze v QSM MAIN.vi a musí být do paralelního SubVI předány spolu s názvem notifikátoru. Notifikátor s tímto názvem se vytváří při ukončení QSM subVI (Slave).vi a kontroluje se při ukončování celé aplikace.

Číslicemi ve čtvercích na Obrázku 22 jsou označeny základní části QSM – PC.

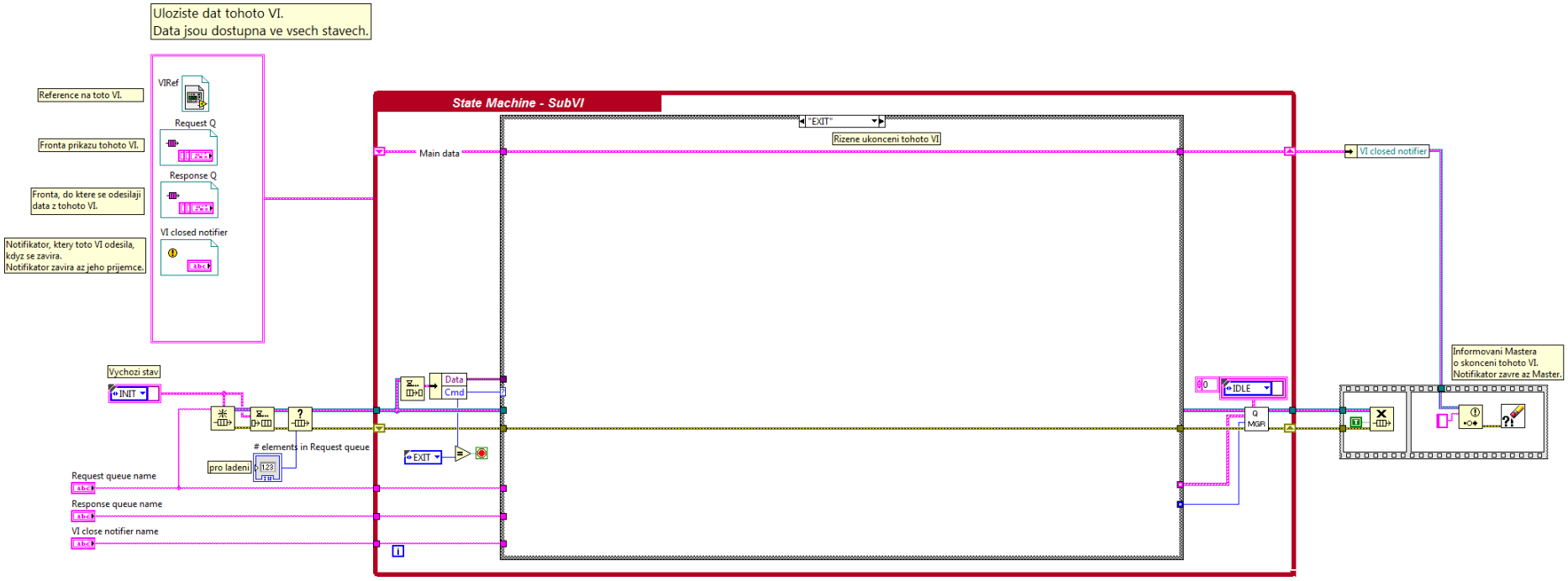
1 Event struktura – Producer
2 State Machine – Consumer
3 Paralelní SubVI



Obrázek 21 Struktura projektu QMS template



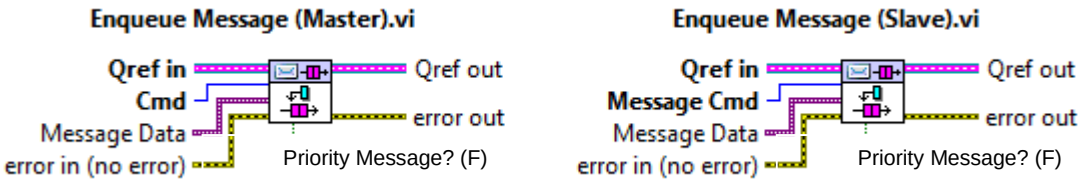
Obrázek 22 QSM –PC hlavní soubor (Master)



Obrázek 23 Blokový diagram QSM subVi (Slave)

5.4.1 Soubory šablony QSM template

- **Enqueue Message**

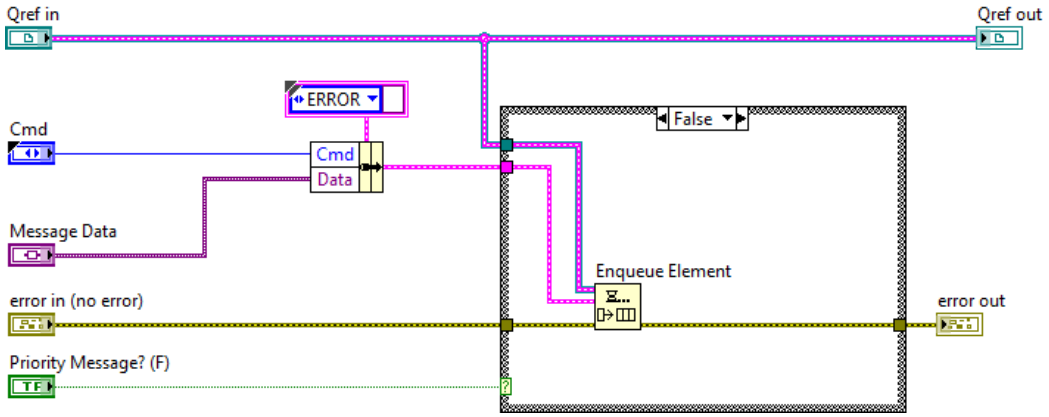


Obrázek 24 Enqueue Message (Master, Slave)

Aby byla při tvorbě programu ušetřena plocha byly využívány pro přidání zpráv do front těchto SubVI. Toto SubVI je vnitřním zapojením stejné pro přidání zpráv do Master i Slave fronty. Rozdílné jsou vstupy, kdy každý očekává svou frontu (Qref in) a příkaz příslušného stavového automatu (Cmd).

Jelikož se často zprávy přidávané do front skládají z příkazu a dat je sloučení provedeno uvnitř SubVI, čím se šetří prostor.

Zároveň je zde vstup Priority Message? (F), který je defaultně nastaven na False čímž se zpráva přidá na konec fronty. Pokud by bylo True změní se stav struktury Case, kde místo bloku Enqueue Elemet, který přidává zprávu na konec fronty, bude blok Enqueue Element At Opposite End a zpráva se přidá na začátek fronty.



Obrázek 25 Blokový diagram Enqueue Message

Tabulka 5.10 Enqueue Message Vstupy

Vstupy	
Qref in	Povinné - Reference na frontu, do které chceme přidat zprávu
Cmd (Message Cmd)	Povinné, Příkaz zprávy, který chceme přidat do fronty
Message Data	Data zprávy, která chceme přidat do fronty
Error in	

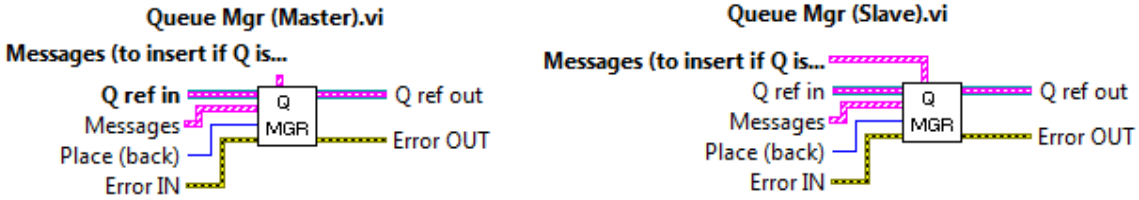
Tabulka 5.11 Enqueue Message Výstupy

Výstupy	
Queue out	Vrací nezměněnou referenci frontu
Error out	

- **Queue Mgr**

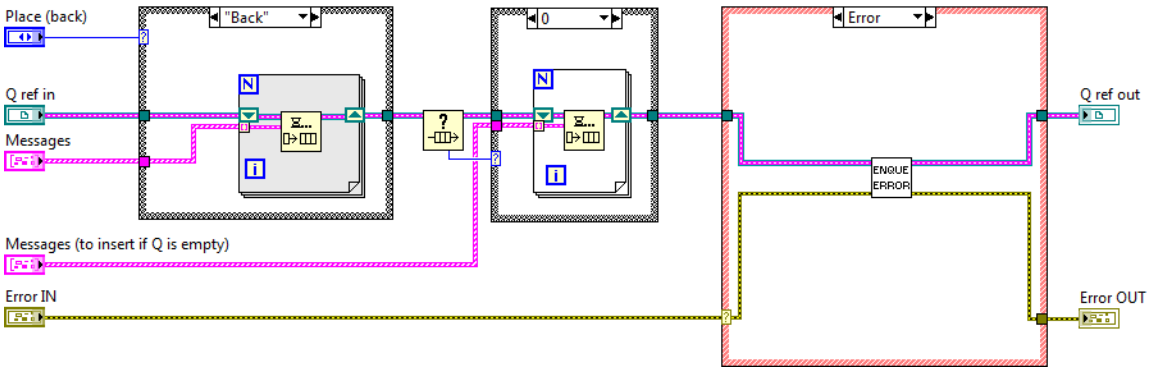
SubVI, které zajišťuje několik důležitých funkcí:

- Zajišťuju přechod do dalšího stavu Case struktury State Machine. Kdy přidá do fronty zprávu, která je tvořena příkazem a prázdnými daty. Pomocí prvku enum lze vybrat prioritu. Na výběr jsou tři stavy Back – přidání zprávy na konec fronty, Front – přidání zprávy na začátek fronty nebo Replace – kdy dojde k uvolnění všech zpráv z fronty a vloží se daná zpráva
- Zajišťuje přidání zprávy do fronty, pokud je fronta prázdná.
- Zajištuje pomocí SubVI ENQUEUE ERROR přechod do stavu ERROR při chybě na žlutém vodiči.



Obrázek 26 Queue Mgr

Toto SubVI je vnitřním zapojením stejné pro Master i Slave. Rozdílné jsou vstupy, kdy každý očekává svou frontu (Qref in), jinou zprávu (Messages), která je přidána do fronty a zajistí tak přechod do dalšího stavu příslušného stavového automatu a jinou zprávu, která se přidá do fronty, pokud je fronta prázdná.



Obrázek 27 Blokový diagram Queue Mgr

Tabulka 5.12 Queue Mgr Vstupy

Vstupy	
Messages (to insert if Q is empty)	Povinné, Zpráva, která se vloží do fronty, pokud je fronta prázdná
Q ref in	Povinné, Reference na frontu, do které chceme přidat zprávu
Messages	Zpráva s příkazem následujícího stavu
Place (back)	Priorita zprávy
Error in	

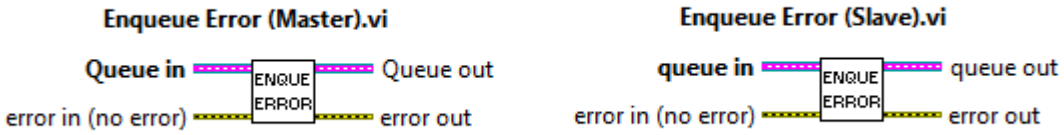
Tabulka 5.13 Queue Mgr Výstpy

Výstupy	
Q ref out	Vrací nezměněnou referenci frontu
Error out	

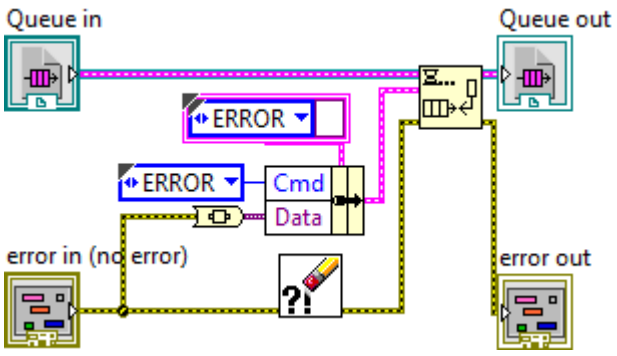
- **ENQUEUE ERROR**

SubVI, které slouží ke zpracování chyby na žlutém vodiči. Vnitřní zapojení je stejné pro hlavní (Master Q) i SubVI frontu (Slave Q). Rozdílné jsou vstupy. Kdy každý očekává jinou frontu (Queue in) a jinou zprávu pro přechod do stavu ERROR.

Pokud nastane chyba na chybovém vodiči, tak při zpracování v tomto bloku se vytvoří prioritní zpráva, která se přidá na začátek fronty, s příkazem (Cmd) ERROR a do Dat se uloží kód chyby. Zároveň dojde ke smazání chyby. Takže na výstupu není chyba. Je však zajištěno, že při další obrátce nekonečné smyčky dojde k vyčtení této zprávy z fronty a stavový automat přechází do stavu, ve kterém je kód pro zpracování chyby.



Obrázek 28 ENQUEUE ERROR



Obrázek 29 Blokový diagram ENQUEUE ERROR

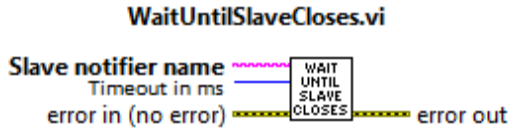
Tabulka 5.14 ENQUEUE ERROR Vstupy

Vstupy	
Queue in	Povinné, Reference na frontu
Error in	

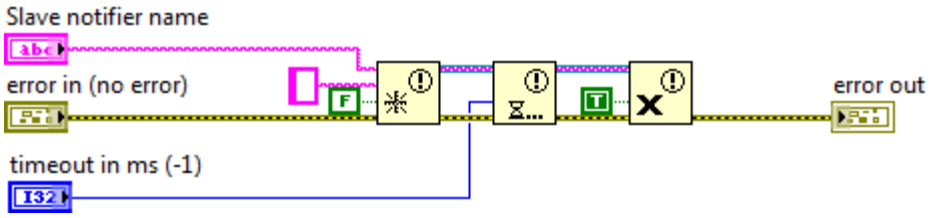
Tabulka 5.15 ENQUEUE ERROR Výstupy

Výstupy	
Q ref out	Vrací nezměněnou referenci na frontu
Error out	

- **WaitUntilSlaveCloses**
  Jednoduché SubVI, které čeká na notifikátor, který vzniká při uzavření Slave SubVI.



Obrázek 30 WaitUntilSlaveCloses



Obrázek 31 Blokový diagram WaitUntilSlaveCloses

Tabulka 5.16 WaitUntilSlaveCloses Vstupy

Vstupy	
Slave notifier name	Název notifikátoru
Timeout in ms (-1)	Čas, který se čeká na notifikátor, -1 nikdy nevyprší
Error in	

Tabulka 5.17 WaitUntilSlaveCloses Výstupy

Výstupy	
Error out	

- **Master_States_Typedef.ctl, Slave_States_Typedef.ctl**

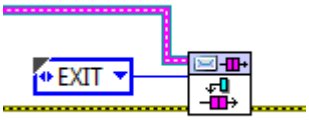
Control prvek tvořený pouze prvkem Enum, který obsahuje všechny stavy stavového automatu hlavního souboru (Master) nebo paralelního SubVI (Slave).

- **Master_Message_Typedef.ctl, Slave_Message_Typedef.ctl**

Cluster složený ze dvou prvků. Enum, který obsahuje všechny stavy stavového automatu hlavní smyčky (Master) nebo paralelního SubVI(Slave). A druhý prvek jsou Data typu Variant. Zprávy, které se ukládají do front jsou tohoto typu.

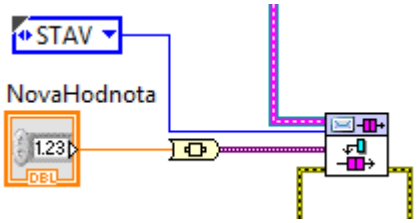
5.4.2 Vložení zprávy do fronty

Novou zprávu do fronty je možné přidat více způsoby, kdy závisí především na jejím obsahu. Respektive jestli do zprávy přidáváme data nebo ne.



Obrázek 32 Vložení zprávy do fronty 1

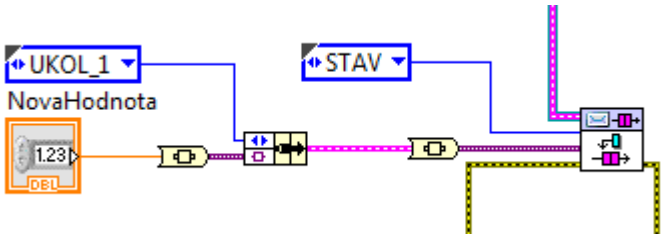
Na Obrázku 32 je zobrazen praktický příklad přidání zprávy do fronty. Zpráva je tvořena pouze příkazem EXIT, data zůstávají prázdná. Po výběru této zprávy příslušný stavový automat přejde do stavu EXIT.



Obrázek 33 Vložení zprávy do fronty 2

Na Obrázku 33 je příklad přidání zprávy, která bude složena z příkazu STAV a dat NovaHodnota. Po výběru této zprávy přejde stavový automat do tohoto nového stavu, ve kterém se data typu variant převedou zpět na číslo a mohou být dále zpracovávána.

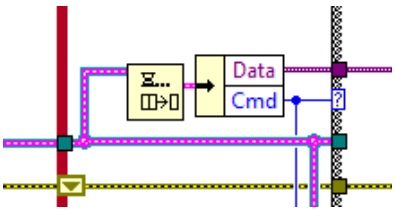
Data ve zprávě mohou být tvořena další zprávou, která obsahuje příkaz a data. A to v případě, že v daném stavu stavového automatu je struktura Case (Obrázek 34).



Obrázek 34 Vložení zprávy do fronty 3

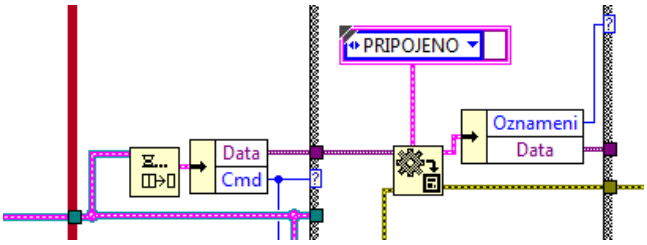
5.4.3 Výběr zprávy z fronty

Reference na frontu je přivedena na blok, který vyjme a vrátí zprávu z fronty. Zpráva je rozložena na Data a příkaz (Cmd). Kde příkaz je připojen na selektor Case struktury stavového automatu, jelikož se jedená o jeden z jeho stavů. Případná Data mohou být následovně zpracována z daném stavu stavového automatu.



Obrázek 35 Výběr zprávy z fronty

Data zprávy mohou být tvořeny další příkazem a daty (Obrázek 36). V tomto případě příkaz zprávy vybrané z fronty vybere stav stavového automatu, data typu variant se převedou zpět původní datový typ, v tomto případě cluster s dvěma prvky Oznamení a Data. Kdy Oznámení určí stav struktury Case a data, pokud nějaká jsou, mohou být dále využita.



Obrázek 36 Výběr zprávy z fronty 2

6. VÝSLEDNÁ OVLÁDACÍ APLIKACE

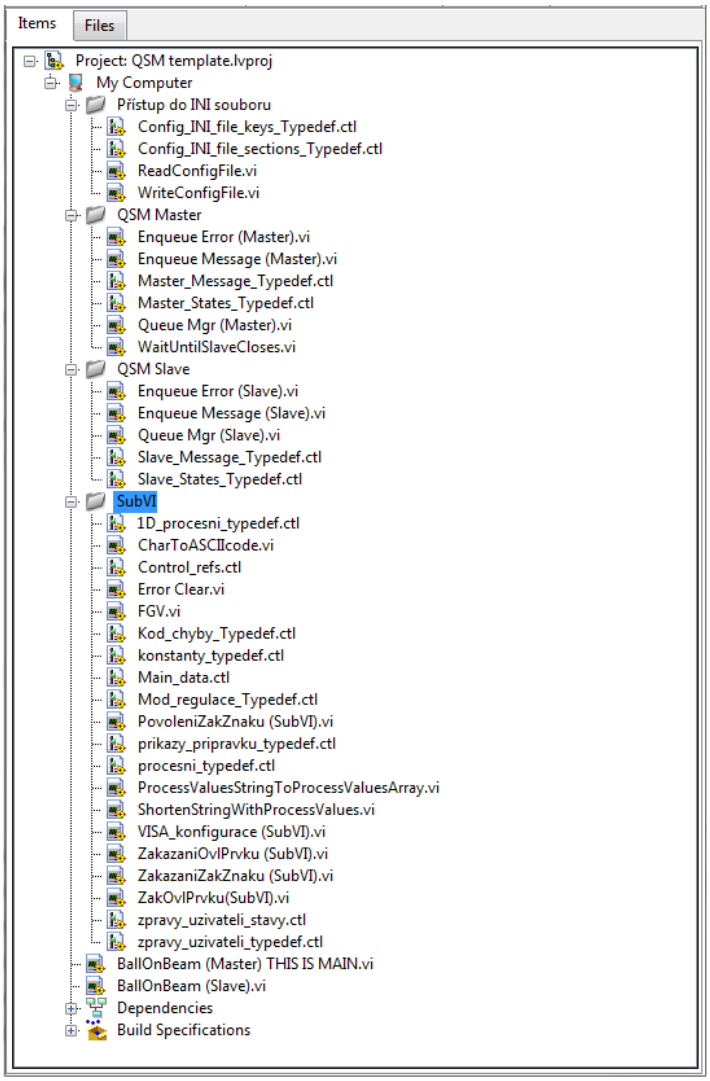
Ovládací aplikace tedy vychází ze šablony QSM-PC, která byla popsána v předchozí kapitole. Pomocí této aplikace bude možné přípravek plně ovládat a sledovat regulační proces.

Funkce ovládací aplikace:

- Výběr COM portu, ke kterému je zařízení připojeno
- Navázání nebo ukončení spojení s přípravkem
- Spuštění nebo zastavení regulace
- Výběr módu – zpětnovazební regulace polohy kuličky nebo nastavení úhlu ramene
- Zobrazení aktuálně nastavených konstant regulátoru polohy kuličky
- Nastavení nových hodnot konstant regulátoru polohy kuličky
- Zobrazení skutečné polohy kuličky na ukazateli
- Zobrazení skutečného úhlu natočení ramene na ukazateli
- Nastavení žádané polohy kuličky
- Nastavení žádaného úhlu natočení ramene
- Zobrazení procesních hodnot v tabulce
- Uložení procesních hodnot do souboru *.csv
- Zobrazení přijatých procesních dat do grafu – skutečná poloha kuličky, skutečný úhel natočení ramene, žádaná poloha kuličky, žádaný úhel natočení ramene

V aplikaci je využito tedy velké množství ovládacích prvků, pomocí kterých uživatel může ovládat laboratorní přípravek. Většina z těchto prvků, ale není trvale dostupná. Kdy v jednotlivých stavech aplikace jsou vždy stanovené určité prvky, které mohou být využity. Například dokud není navázáno spojení s přípravkem lze pouze vybrat COM port a ovládat tlačítko pro připojení.

Na Obrázku 37 je struktura projektu výsledného ovládacího softwaru. Projekt obsahuje všechny součásti využité šablony, zároveň bylo přidáno několik dalších souborů důležitých pro správnou funkci programu.



Obrázek 37 Struktura projektu výsledné aplikace

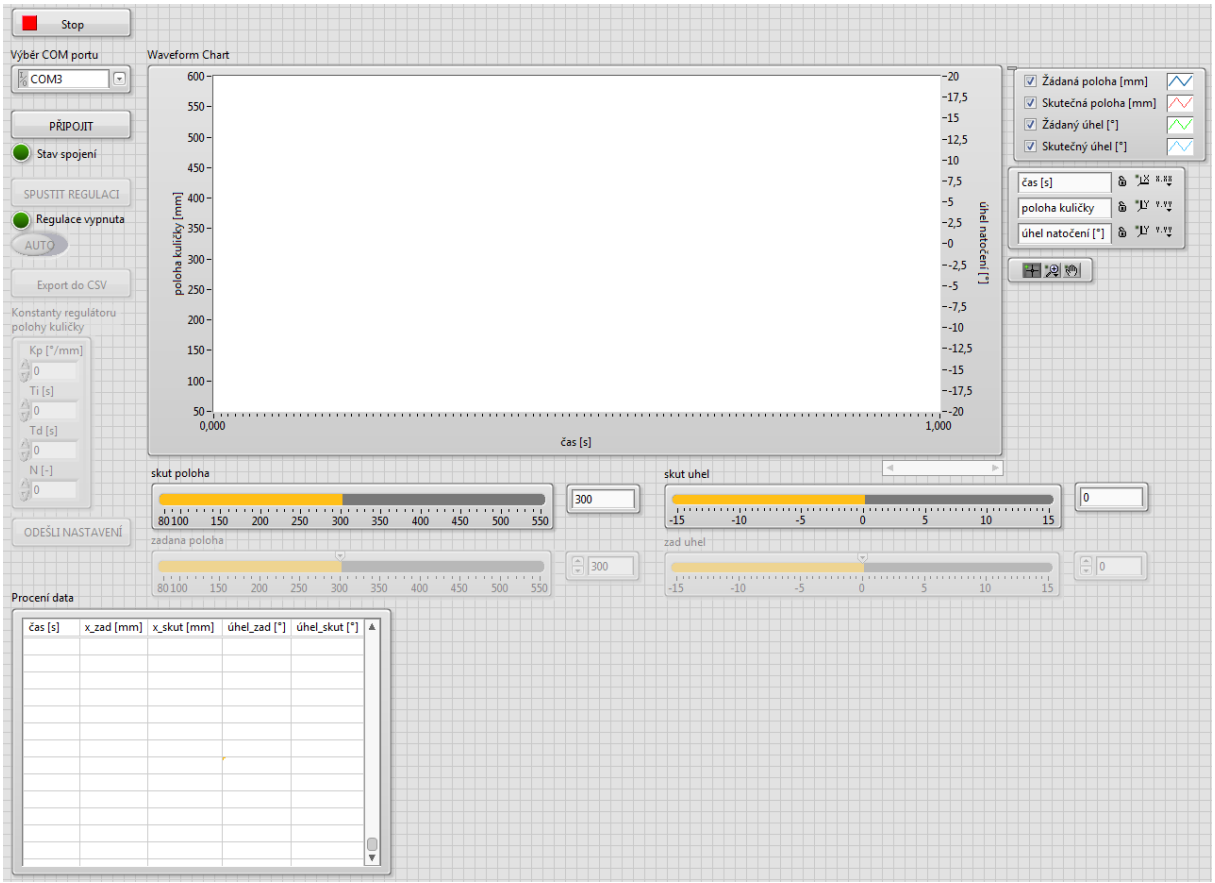
6.1 Čelní panel ovládací aplikace

Na Obrázku 38 je zobrazen čelní panel výsledné aplikace, která nebyla spuštěná a některé ovládací prvky jsou tak zakázány.

Čelnímu panelu dominuje graf, který slouží pro vykreslování přijatých hodnot. Je tedy snadné sledovat průběh regulačního procesu. V pravé části u grafu lze nastavit veličiny, které mohou být vykreslovány.

V levé části jsou umístěny ovládací prvky, které slouží k výběru COM portu, připojení nebo odpojení od přípravku, spuštění nebo zastavení regulace, uložení přijatých procesních dat do souboru *.CSV, nastavení a odeslání konstant regulátoru. Zároveň jsou zde umístěny dva indikátory, které zobrazují stav spojení a regulace.

Pod grafem jsou dva ukazatele skutečné polohy (levý ukazatel) a úhlu natočení ramene (pravý ukazatel). Pod nimi jsou dva příslušné ovládací posuvné prvky, pomocí kterých lze nastavit žádanou polohu kuličky nebo žádaný úhel natočení ramene.



Obrázek 38 Čelní panel aplikace

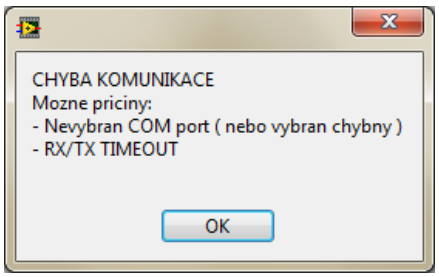
6.2 Popis funkce a ovládání aplikace

Nejdříve musíme připojit laboratorní přípravek pomocí kabelu k počítači. V aplikaci je ovládací prvek s názvem Výběr COM portu, ve kterém je předvolen COM port, ke kterému bylo zařízení připojeno naposledy. A to z důvodu, že aplikace po spuštění načte inicializační soubor, ve kterém je tato informace uložena. Po výběru správného COM portu stačí stisknout tlačítko PŘIPOJIT. Jelikož příkaz pro připojení je od přípravku potvrzovaný kontroluje se tato odpověď.

Může tak nastat několik stavů:

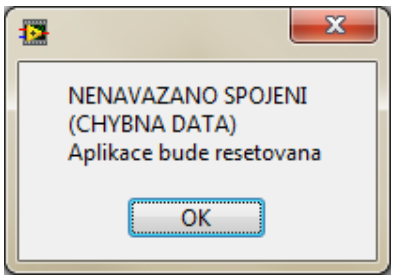
- Ihned po stisku se zobrazí okno CHYBA KOMUNIKACE (Obrázek 39). Příčinou je chybně vybraný COM port, nebo není možné se k vybranému portu připojit.

• Rameno sjíždí na koncový spínač, sepne ho a vrací se do vodorovné polohy, poté se zobrazí okno CHYBA KOMUNIKACE (Obrázek 39). Příčinou je, že vypršel Rx nebo Tx timeout. Nedochází tak k navázání spojení s přípravkem.



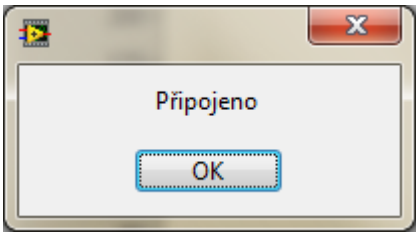
Obrázek 39 Okno CHYBA KOMUNIKACE

• Rameno sjíždí na koncový spínač, sepne ho a vrací se do vodorovné polohy, poté se zobrazí okno NENAVAZANO SPOJENI (Obrázek 40). Příčinou je, že přijatá odpověď od přípravku není shodná s očekávanou odpovědí. Nedochází tak k navázání spojení s přípravkem.



Obrázek 40 Okno NENAVAZANO SPOJENI

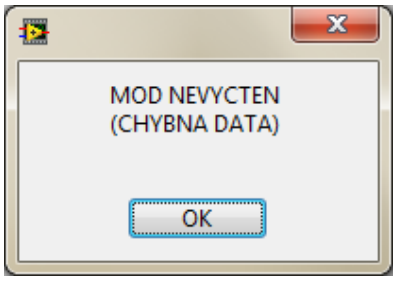
• Rameno sjíždí na koncový spínač, sepne ho a vrací se do vodorovné polohy, poté se zobrazí okno Připojeno (Obrázek 41). Jedná se o jediný korektní stav, kdy byla přijata správná odpověď od přípravku a je s ním navázáno spojení. Zároveň dojde k uvolnění několika ovládacích prvků.



Obrázek 41 Okno Připojeno

Po úspěšném navázání spojení s přípravkem se vyčítají konstanty regulátoru polohy kuličky a mód regulace. Vyčítané hodnoty konstant regulátoru nejsou kontrolovány, ale může dojít k chybě, kdy vyprší Tx nebo Rx timeout a zobrazí se okno CHYBA KOMUNIKACE (Obrázek 39). K této chybě může dojít i při vyčítání módu regulace, zároveň je ale kontrolována

přijatá hodnota, která může nabývat pouze 0 nebo 1. Pokud je přijato cokoli jiného, zobrazí se okno MOD NEVYCTEN (Obrázek 42). Pokud se zobrazí okno CHYBA KOMUNIKACE (Obrázek 39), dochází k restartu aplikace. V ostatních případech je možné dále ovládat aplikaci.



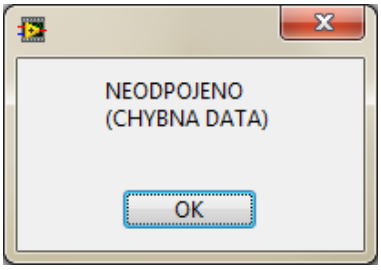
Obrázek 42 Okno MOD NEVYCTEN

Pokud došlo k připojení, rozsvítí se indikátor Stav spojení, původní tlačítko pro připojení nyní slouží pro odpojení. Zároveň jsou uvolněny další ovládací prvky a je možné provádět různé úkony.

Ukončení spojení aplikace s přípravkem se provede stiskem tlačítka ODPOJIT. Odpojení je možné pouze v případě, když není spuštěna regulace. Jedná se znovu o potvrzovaný příkaz. Kontroluje se tedy odpověď od přípravku.

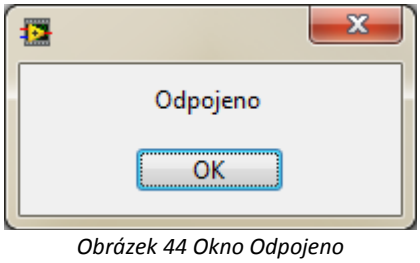
Může nastat několik stavů:

- Vyprší RX nebo TX timeout není tak získána potřebná odpověď a zobrazí se okno CHYBA KOMUNIKACE (Obrázek 39).
- Přijatá odpověď neodpovídá očekávané. Zobrazí se okno NEODPOJENO (Obrázek 43)



Obrázek 43 Okno NEODPOJENO

- Přijatá odpověď od přípravku se shoduje s očekávanou odpovědí. V tomto případě se ukončí navázané spojení s přípravkem. A zobrazí se potvrzovací okno Odpojeno. (Obrázek 44).



Obrázek 44 Okno Odpojeno

Pokud je zařízení připojeno je dále možné například nastavit nové hodnoty konstant regulátoru polohy kuličky. Nové hodnoty se odešlou do přípravku stiskem tlačítka ODEŠLI NASTAVENÍ. Kdy odesílané hodnoty nejsou nijak kontrolované.

Dále je možné nastavit mód regulace pomocí přepínacího tlačítka AUTO/MANUAL. V případě nastavení na MANUAL je aktivní řízení úhlu natočení ramene. AUTO reprezentuje zpětnovazební regulaci polohy kuličky.

Další možností je nastavení žádané polohy nebo úhlu natočení ramene. Pokud není zapnuta regulace, hodnoty se neodesílají.

Spuštění regulace se provede stiskem tlačítka SPUSTIT REGULACI. Nejdříve dochází k odesílání aktuálně nastavených hodnot žádané polohy a žádaného uhlu poté se odesílá příkaz do přípravku pro spuštění regulace a rozsvítí se indikátor Regulace zapnuta. Podle zvoleného módu se rameno buď natáčí na žádaný úhel, nebo je aktivní zpětnovazební regulace kuličky na žádanou polohu. Mód, žádanou polohu a žádaný úhel je možné měnit při spuštěné regulaci. Po dobu, kdy je regulace spuštěná, jsou přijímána procesní data od přípravku, která jsou zpracována a poté zobrazena do grafu a uložena do tabulky. Do grafu mohou být vykreslovány průběhy Žádané polohy, Skutečné polohy, Žádaného a skutečného úhlu natočení ramene. Je možné si zvolit, jakou z těchto čtyř veličin chceme nechat vykreslovat. Vypnutí regulace se provede stiskem stejného tlačítka jako pro spuštění regulace, avšak toto tlačítko plní, při spuštěné regulaci, funkci Zastavení regulace.

V případě, že jsou v tabulce s názvem Procesní data hodnoty, které je chceme uložit, stiskneme tlačítko Export do CSV. Zobrazí se dialogové okno pro výběr souboru *.csv, pokud nedojde k výběru souboru, vytvoří se nový soubor *.csv. Uložení dat je možné pouze v případě, když není aktivní regulace. Uložená data v souboru jsou číselné hodnoty, se kterými lze bez dalších úprav ihned pracovat. Jednotlivé veličiny jsou v samostatných sloupcích (Obrázek 45).

	A	B	C	D	E
1	čas [s]	x_zad [mm]	x_skut [mm]	úhel_zad [°]	úhel_skut [°]
2	0,002	250	350	5	11
3	0,004	250	350	5	11
4	0,006	250	350	5	11
5	0,008	250	350	5	11
6	0,01	250	350	5	11

Obrázek 45 Ukázaka uložených dat z CSV

6.3 Popis jednotlivých částí hlavního souboru programu

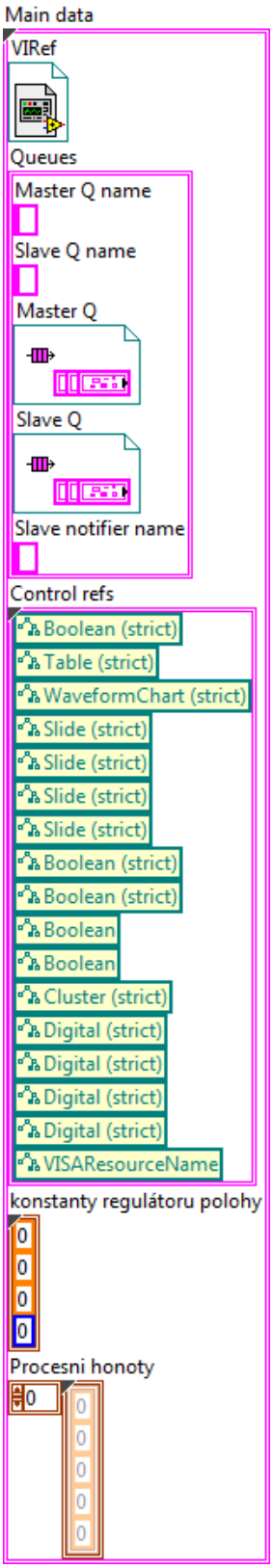
Využitá šablona QSM-PC, která byla popsána v jedné z předchozích kapitol, obsahovala pouze základní prvky. Nejdůležitější částí, kterou šablona obsahovala, byla vytvořená komunikace mezi jednotlivými smyčkami a paralelními SubVI. Tato skutečnost výrazně urychlila vytvoření výsledné aplikace. I tak byla tvorba aplikace velice časově náročná, protože se jedná o relativně rozsáhlou aplikaci.

V hlavním souboru aplikace jsou především zpracovávány události od uživatele, jako například stisk tlačítka, a jsou ovládány zobrazovací a ovládací prvky. Paralelní SubVI zajišťuje především komunikaci s přípravkem a prvotní zpracování přijatých dat.

6.3.1 Data dostupná v hlavním souboru

V hlavní souboru aplikace je několik smyček, ve kterých je nutné mít k dispozici data. K ukládání dat je využita Funkční Globální Proměnná neboli FGV, která byla popsána v jedné z předchozích kapitol. Při každém spuštění aplikace nejprve proběhne inicializace clusteru FGV, kdy se nastaví názvy front a notifikátoru.

Jelikož je nutné uchovávat větší množství dat je využit cluster s názvem Main data. Tento cluster je tvořen pěti základními prvky, kde některé jsou typu cluster a obsahují další prvky.



Obrázek 46 Cluster Main data

Obsah Clusteru Main Data:

- VIRef – reference na Hlavní VI
- Queues – cluster tvořený dalšími prvky
    - Master Q name – název fronty hlavního souboru (Master VI)
    - Slave Q name – název smyčky, paralelního SubVI (Slave VI)
    - Master Q – reference na frontu hlavního souboru (Master VI)
    - Slave Q – reference na frontu paralelního SubVI (Slave VI)
    - Slave notifier name – název, se kterým se vytvoří notifikátor při ukončení Slave VI
- Control refs – cluster referencí na ovládací prvky, aby je bylo pomocí referencí možné ovládat z jakékoliv části aplikace. Jedná se postupně o: Export do CSV, Procesni data, Waveform Chart, Zadany uhel, Skutecny uhel, Skutecna poloha, Zadana poloha, OdesliNasRegPolohy, MOD, RUN, CONNECT, Konstanty regulátoru polohy kuličky, N, Td, Ti, Kp, VISAName Refnum
- Konstanty regulátoru polohy – cluster se čtyřmi prvky
    - Kp
    - Ti
    - Td
    - N
- Procesni hodnoty – 1D pole clusteru s pěti prvky
    - Ts
    - ZadPoloha

- SkutPoloha
- ZadUhel
- SkutUhel

6.3.2 Event struktura

Event struktura, která se nachází v hlavním souboru aplikace. Slouží k obsluze událostí, která byly vytvořeny uživatelem, jedná se například o stisknutí jednoho z tlačítek. Vlastní Event struktura je umístěna ve smyčce While.

Event struktura v tomto případě může obsluhovat jedenu z osmi událostí. Obsluha jednotlivých událostí z hlediska náročnosti je různá. Pokud se jedná o jednoduchý úkon, může být vykonán ihned, ve většině případů obsluha spočívá v přidání zprávy do Master nebo Slave fronty. Další obsluha se tedy provede v příslušném stavu daného stavového automatu.

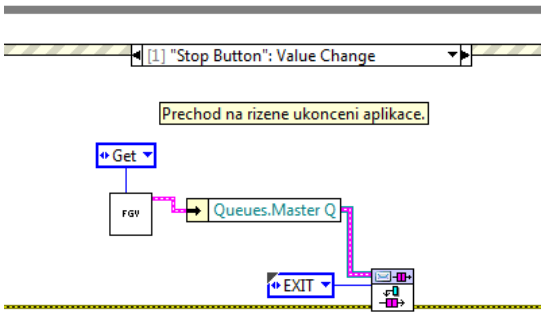
Jednotlivé stavy

• Timeout

Pokud nejsou žádné jiné události. Nevykonává se žádná činnost.

• Stop Button

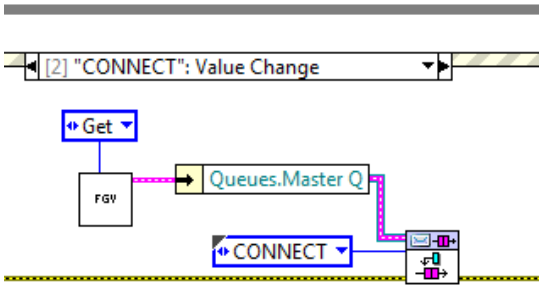
Obsluha po stisku tlačítka Stop. Z FGV získáme referenci na frontu Master a přidáme zprávu, která se skládá pouze z příkazu EXIT. Obsluha nastává při změně hodnoty tlačítka.



Obrázek 47 Event struktura Stop button

• CONNECT

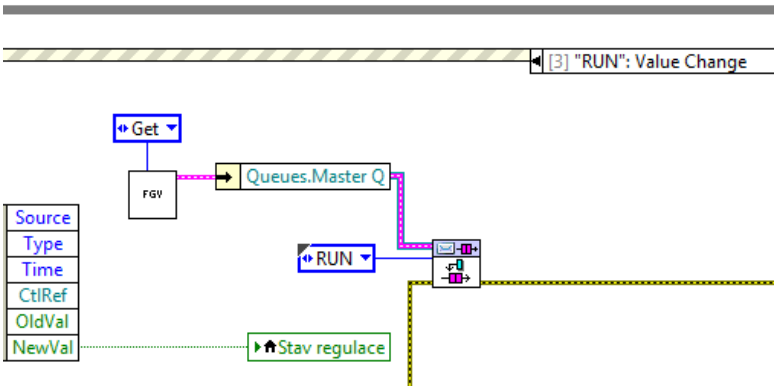
Obsluha při změně hodnoty tlačítka sloužící pro připojení respektive odpojení od přípravku. Přidává se pouze příkaz do zprávy, která je přidána do fronty Master a další zpracování probíhá ve stavu CONNECT stavového automatu hlavní smyčky.



Obrázek 48 Event struktura CONNECT

- **RUN**

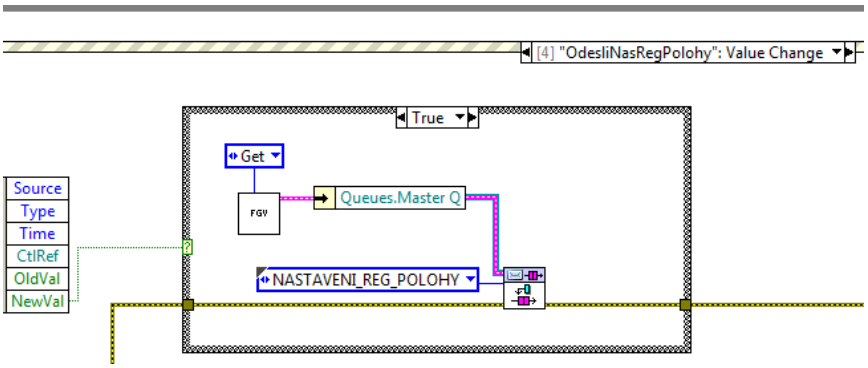
Reakce na změnu hodnoty tlačítka pro spuštění nebo zastavení regulace. Do Master fronty se přidává zpráva, u které je nastaven pouze příkaz. Zároveň se podle nové hodnoty nastaví hodnota indikátoru s názvem Stav regulace



Obrázek 49 Event struktura RUN

- **OdesliNasRegPolohy**

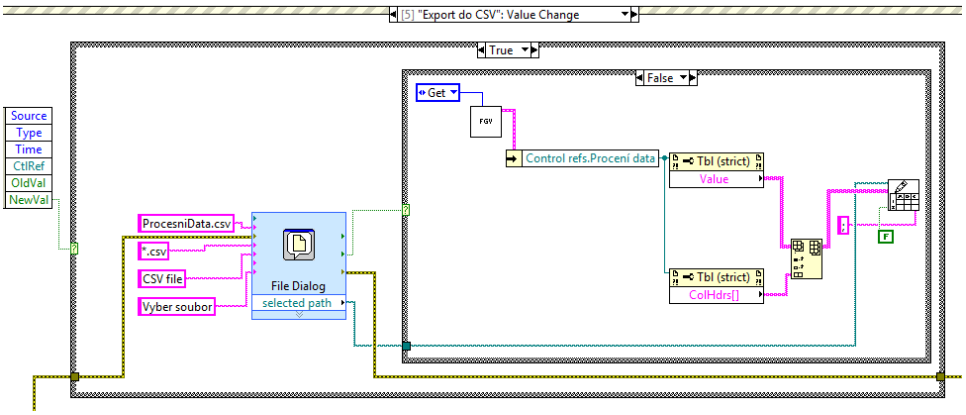
Pokud dojde ke změně stavu tlačítka a je jeho nová hodnota true, přidá se do Master fronty zpráva, u které je zadán pouze příkaz. V případě false se nic nevykonává.



Obrázek 50 Event struktura OdesliNasRegPolohy

• **Export do CSV**

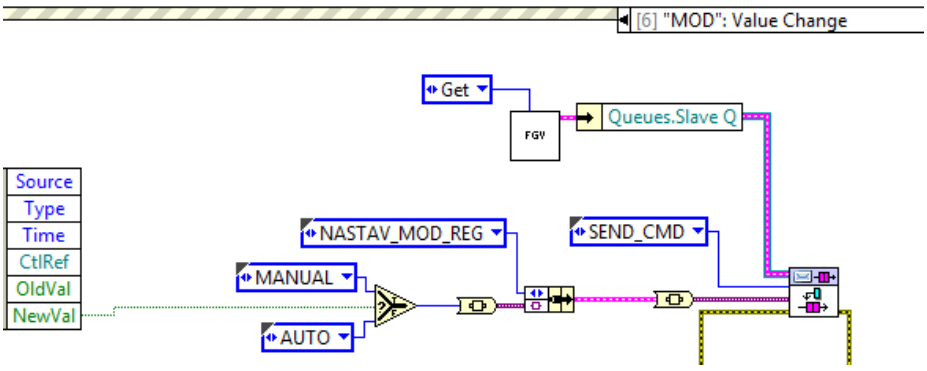
Reakce na změnu hodnoty tlačítka. Pokud je hodnota true, zobrazí se dialogové okno pro vybrání souboru, do kterého se uloží přijatá procesní data. Pokud není vybrán soubor, vytvoří se nový s názvem ProcesniData.csv. V případě, že je okno uzavřeno nedojde k uložení Procesních dat. První řádek v souboru je vždy hlavička.



Obrázek 51 Event struktura Export do CSV))

• **MOD**

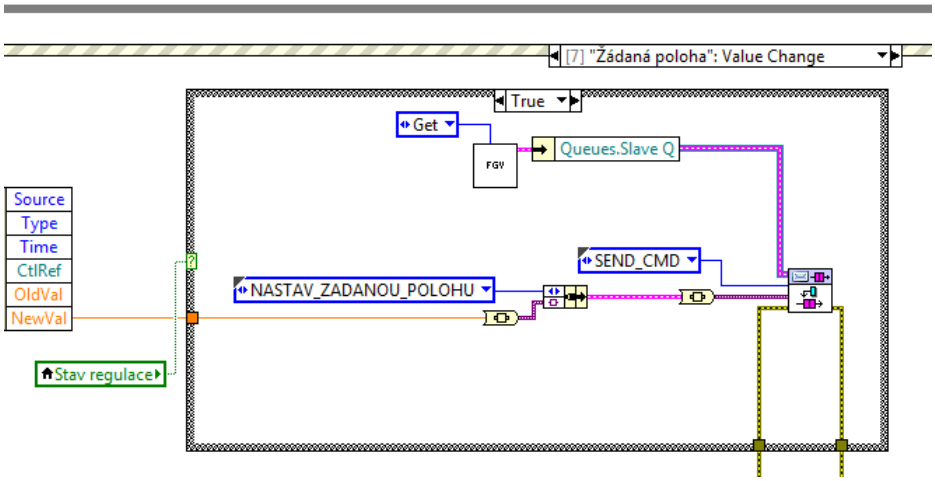
Reakce na změnu hodnoty přepínače. V tomto případě se přidává zpráva do fronty Slave. Zpráva je tvořena příkazem SEND_CMD a daty typu variant, které se skládají z dalšího příkazu NASTAV_MOD_REGULACE a dat typu variant. Tyto data mohou nabývat jedné ze dvou hodnot a to podle toho, zdali je hodnota přepínače true nebo false.



Obrázek 52 Event struktura MOD

• **Žádaná poloha**

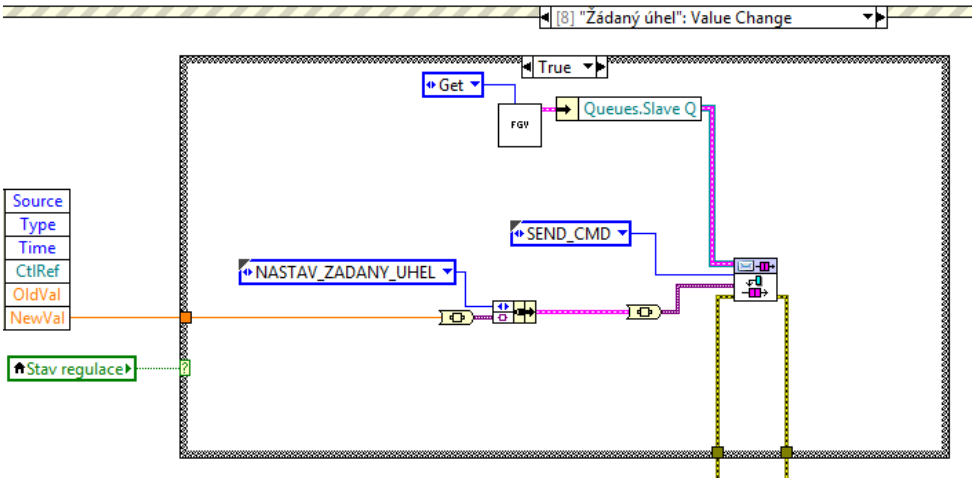
Reakce na změnu hodnoty posuvníku Žádané polohy. V případě, že je regulace aktivní přidává se do fronty zpráva při každé změně hodnoty. Zpráva je tvořená příkazem SEND_CMD a daty typu variant, která jsou tvořena dalším příkazem NASTAV_ZADANOU_POLOHU a daty typu variant. Data jsou dána hodnotou navolenou posuvníkem. Pokud regulace není aktivní, nevykonává se žádná činnost.



Obrázek 53 Event struktura Žádaná poloha

• **Žádaný úhel**

Totožné jako nastavení žádané polohy. Příkaz zprávy je stejný, pouze data jsou tvořena příkazem NASTAV_ZADANY_UHEL a hodnotou příslušného posuvníku. Pokud není regulace spuštěna, nevykonává se žádná činnost.



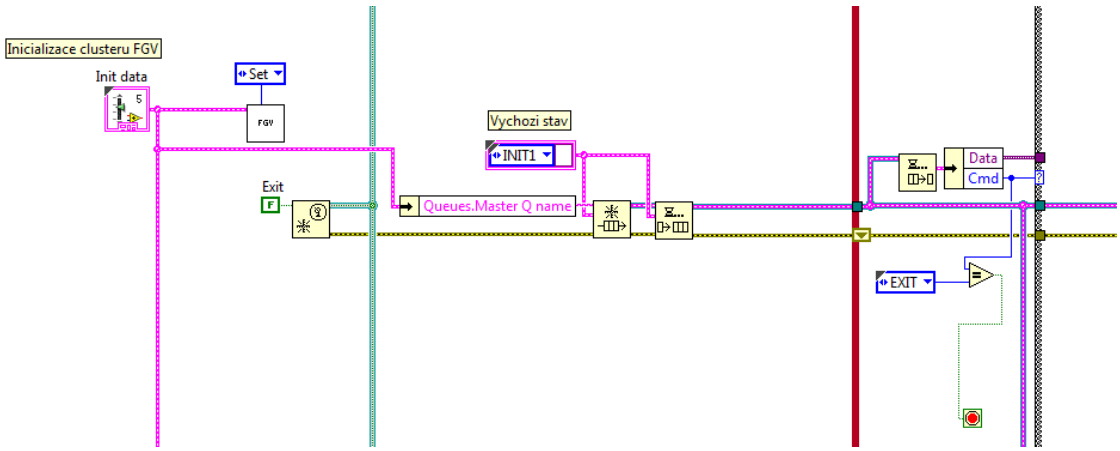
Obrázek 54 Event struktura Žádaný úhel

• **EXIT**

Reakce na vygenerovanou událost. Tento stav způsobí ukončení While smyčky přivedením hodnoty true na podmínku smyčky, která je nastavena na Stop if true.

6.3.3 State Machine

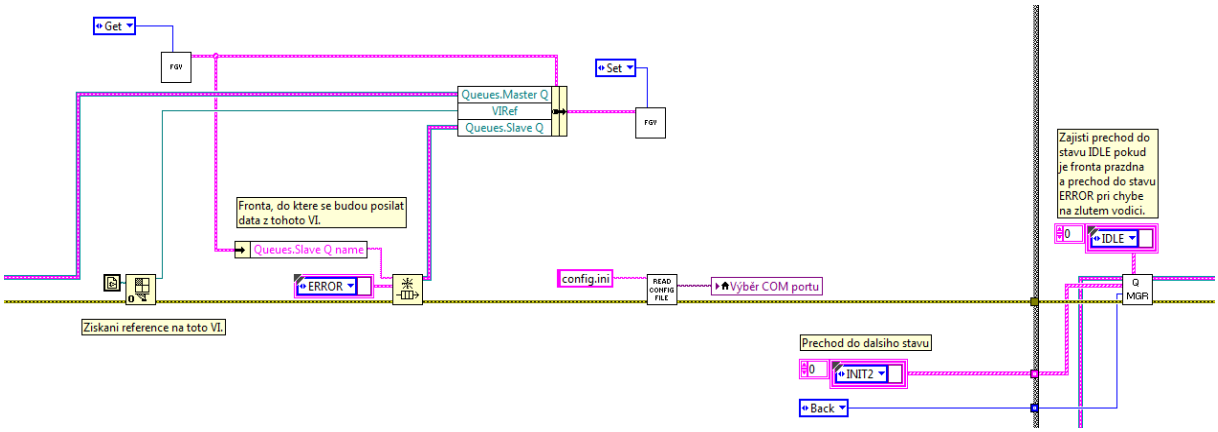
Zde se zpracovávají zprávy z fronty Master Q. Při každém spuštění aplikace je nutné inicializovat cluster FGV, vytvořit referenci na uživatelskou událost, která slouží pro generování události pro ukončení Event smyčky. Dále je nutné získání reference na Master Q a přidání první zprávy do fronty, která představuje výchozí stav stavového automatu. (Obrázek 55). Uvnitř stavového automatu poté dochází k vyzvednutí prvku z fronty. Pokud je příkaz zprávy EXIT, následuje ukončení smyčky.



Obrázek 55 Stavový automat inicializace po spuštění

• **INIT1**

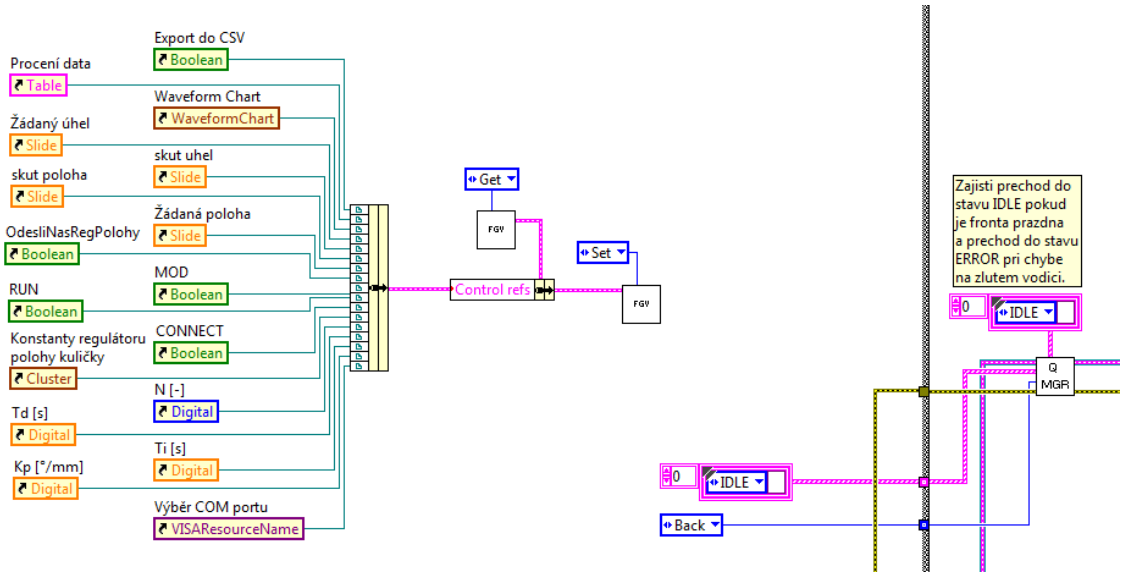
První stav stavového automatu po spuštění. Ve stavu INIT1 se získává reference na toto hlavní VI, frontu SlaveQ. Reference na Master Q je získána po spuštění aplikace (Obrázek 55). Tyto tři reference jsou uloženy do FGV. Následuje čtení konfiguračního souboru. Aby bylo možné konfigurační soubor přečíst, musí mít daný název a musí se nacházet ve stejném adresáři s ovládacím softwarem. V současné verzi aplikace se z tohoto souboru vyčítá pouze naposledy použitý COM port. Nakonec se předávají informace pro přechod do dalšího stavu stavového automatu.



Obrázek 56 Stavový automat – INIT1

- **INIT2**

Následující stav po INIT1. V tomto stavu dochází k inicializaci referencí na ovládací prvky a uložení do FGV. Po tomto stavu přechází stavový automat do stavu IDLE. V případě přidání dalšího prvku na front panel je nutné vytvořit referenci na tento ovládací prvek a tuto referenci přidat do clusteru na obrázku 57 a následně i do Control_refs.ctl a uložit, čímž se aktualizuje i cluster Main_data.ctl.



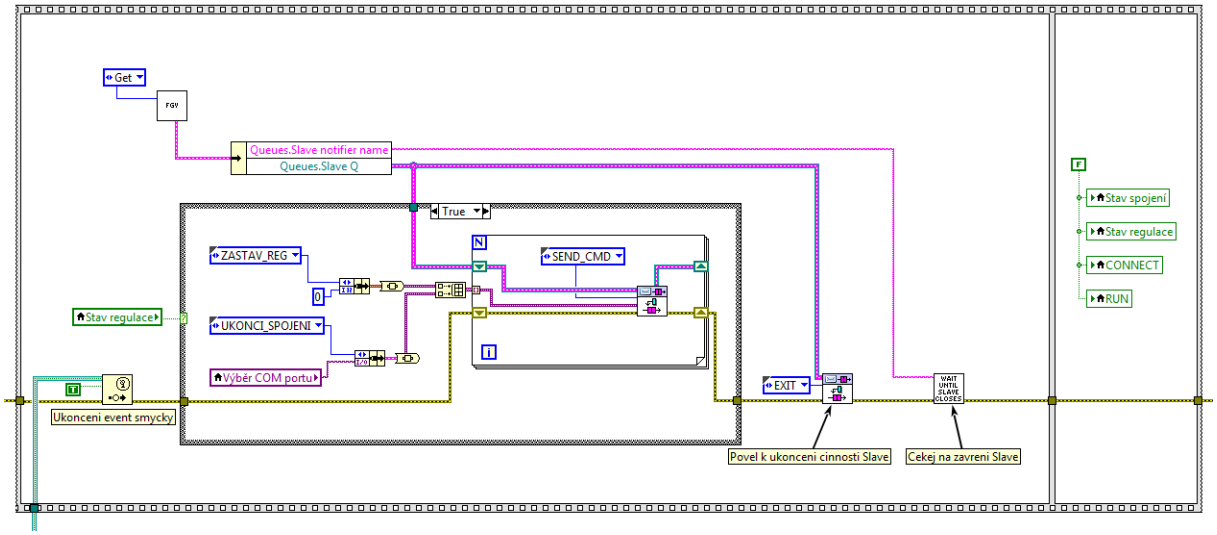
Obrázek 57 Stavový automat - INIT2

- **IDLE**

V tomto stavu se nic nevykonává, ani není nastavován přechod do dalšího stavu. Pokud je po vykonání tohoto stavu fronta Master Q prázdná přidá Q MGR zprávu s příkazem IDLE a vykonání tohoto stavu se opakuje.

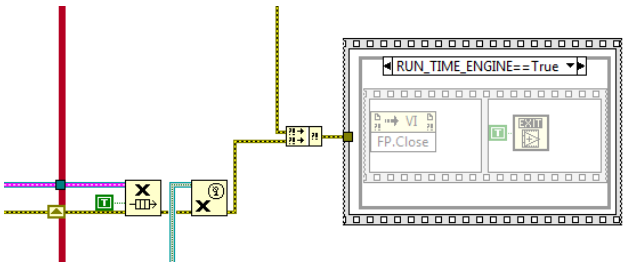
• **EXIT**

Jedná se o stav, do kterého aplikace přechází po stisku tlačítka Stop. Nejprve dojde k vygenerování události pro ukončení Event smyčky. Následně záleží na stavu regulace. Pokud je regulace aktivní tak se přidávají dvě zprávy do fronty Slave Q. Příkaz je pro obě stejný a to SEND_CMD. Liší se data, která se opět skládají z příkazu a dat. Nejdříve jsou data tvořena příkazem pro zastavení regulace a žádnými daty a poté se přidává zpráva, kde data tvoří příkaz pro ukončení komunikace a COM port ke kterému je zařízení připojeno. Pokud regulace není aktivní, přechází struktura Case do stavu False, ve které se nic nevykonává. Následuje přidání zprávy do Slave Q, u které je zadán pouze příkaz EXIT. Poté se čeká na notifikátor, který je generován při ukončení paralelního SubVI. Jako poslední dojde k nastavení hodnoty False indikátorů Stav spojení, Stav regulace, ovládacích prvků sloužících pro připojení respektive odpojení aplikace k přípravku a ke spuštění nebo zastavení regulace. Pokud je příkaz vyjmuté zprávy EXIT dojde k ukončení While smyčky stavového automatu (Obrázek 55).



Obrázek 58 Stavový automat - EXIT

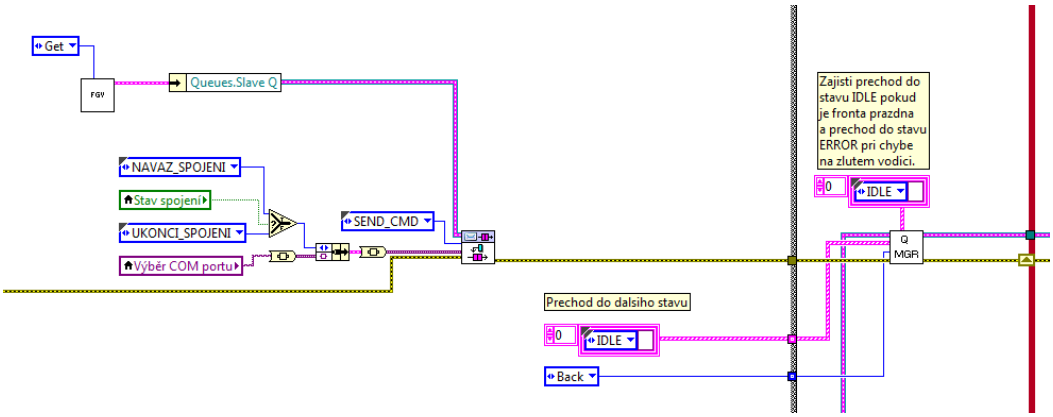
Poté následuje uvolnění fronty, zničení reference na uživatelskou událost a ukončení celé aplikace (Obrázek 59).



Obrázek 59 Ukončení aplikace

- **CONNECT**

Přechod do toho stavu je reakcí na stisk tlačítka pro připojení respektive odpojení aplikace od přípravku. Do zprávy, která se vkládá do fronty Slave Q, se předává příkaz SEND_CMD a data typu Variant. Data byla před převedením na Variant tvořena clusterem se dvěma prvky. Jedním prvkem je příkaz pro navázání nebo ukončení spojení. Výběr jednoho z těchto příkazů závisí na hodnotě Stavu spojení. Druhým prvkem clusteru je název COM portu, ke kterému je přípravek připojen. Po přidání této zprávy do fronty stavový automat přechází do stavu IDLE.

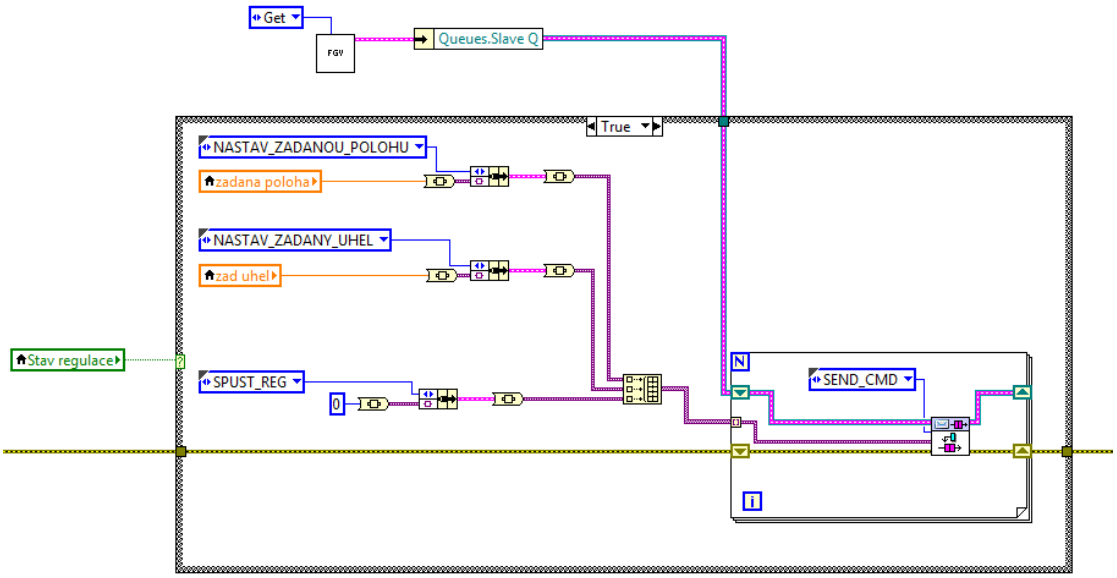


Obrázek 60 Stavový automat CONNECT

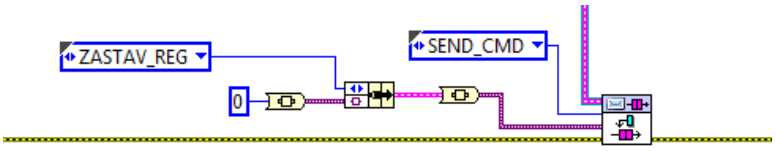
- **RUN**

Po stisku tlačítka pro připojení nebo odpojení se v Event struktuře přidá do fronty Master Q zpráva s příkazem pro přechod stavového automatu do tohoto stavu. Kdy podle stavu regulace se rozhodne, jaké zprávy budou odeslány. Pokud je hodnota true, jsou do Slave Q fronty nejdříve přidány zprávy pro nastavení žádané hodnoty polohy kuličky a úhlu natočení ramene. Až poté je přidána zpráva pro spuštění regulace (Obrázek 61). Naopak, pokud je hodnota false dojde k odeslání pouze jedné zprávy k zastavení regulace (Obrázek 62).

Zároveň v tomto stavu dojde k povolení a zakázání některých ovládacích prvků v závislosti na Stavu regulace. Po tomto stavu následuje přechod do stavu IDLE.



Obrázek 61 Stavový automat RUN spuštění regulace



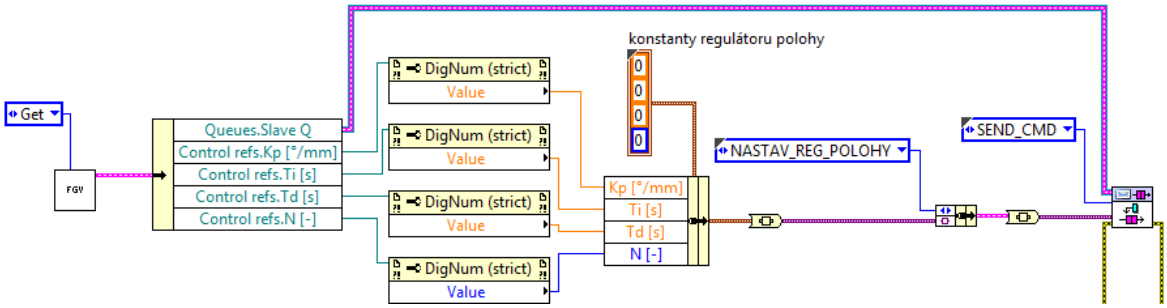
Obrázek 62 Stavový automat zastavení regulace

- **NASTAVENI_REG_POLOHY**

Po úspěšném navázání spojení s přípravkem je možné nastavit nové konstanty regulátoru polohy kuličky. K nastavení nových hodnot slouží tlačítko ODEŠLI NASTAVENÍ. Do tohoto stavu se přejde na základě příkazu ze zprávy, která byla do fronty přidána v Event struktuře.

V tomto stavu se přidá do fronty Slave Q zpráva s příkazem SEND_CMD a daty. Data jsou cluster se dvěma prvky, který byl převeden na datový typ variant. Prvním prvkem clusteru je příkaz NASTAV_REG_POLOHY a druhý prvek jsou data typu Variant, před převedením se jednalo o cluster se čtyřmi prvky. Kde tyto čtyři prvky jsou hodnoty jednotlivých konstant regulátoru polohy kuličky.

Po vykonání tohoto stavu přechází stavový automat do stavu IDLE.

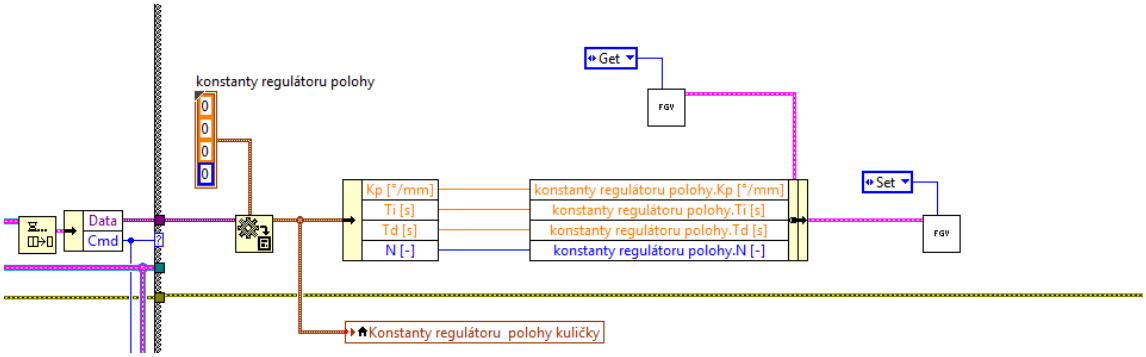


Obrázek 63 Stavový automat NASTAVENI_REG_POLOHY

- **PRIJATE_NAST_REG_POLOHY**

Po navázání spojení s přípravkem jsou vyčteny z přípravku konstanty regulátoru polohy kuličky. Přijetí těchto hodnot se provádí v paralelním SubVI s názvem BallOnBeam (Slave). Toto SubVI přepošle přijatá data do hlavního programu tím, že zapíše zprávu do fronty Master Q, data této zprávy jsou hodnoty jednotlivých konstant.

V hlavním programu dojde k vyzvednutí zprávy z fronty Master Q a vyjmutí dat ze zprávy. Data typu variant se převedou na cluster se čtyřmi hodnotami a to Kp, Ti, Td a N. Tyto aktuální hodnoty uložíme do FGV a aktualizujeme příslušné ovládací prvky na čelním panelu. Poté stavový automat přechází do stavu IDLE.

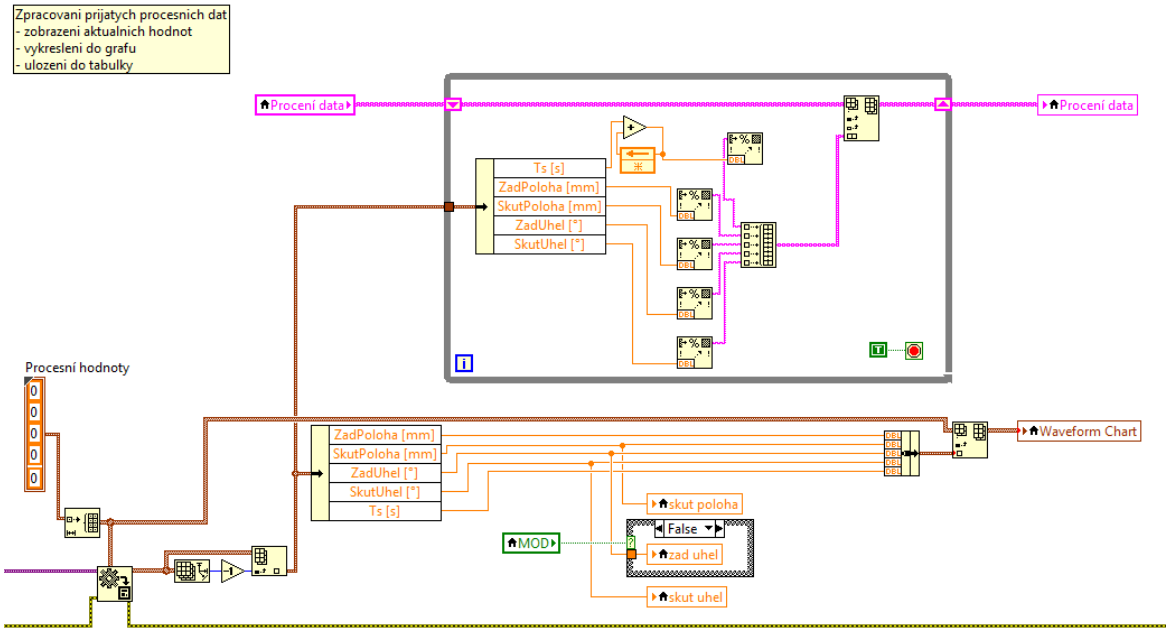


Obrázek 64 Stavový automat PRIJATE_NAST_REG_POLOHY

- **PRIJATA_PROCESNI_DATA**

Pokud je regulace aktivní, odesílá přípravek neustále procesní data, která jsou přijímána a zpracovávána v paralelním SubVI. Zpracovaná data přidá SubVI do zprávy s příkazem pro přechod do tohoto stavu ve stavovém automatu hlavního programu. Tato zpráva je následně přidána do fronty Master Q.

Přijaté procesní hodnoty jsou uloženy do tabulky, vykresleny do grafu a zároveň dochází k aktualizaci prvků na čelním panelu. Stavový automat poté přechází do stavu IDLE.

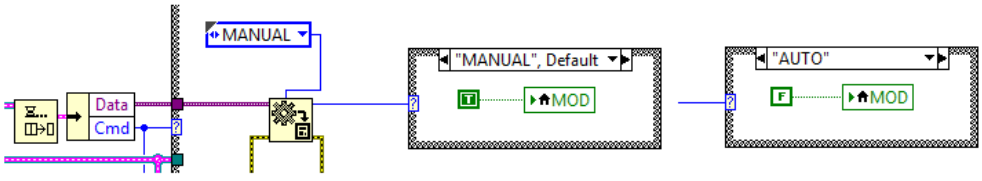


Obrázek 65 Stavový automat PRIJATA_PROCESNIi_DATA

- **PRIJAT_MOD_REGULACE**

Po úspěšném spojení s přípravkem je vyčten také nastavený mód regulace. Informace o módu je přijata v paralelním SubVI, kde je zpracována, přidána ve formě dat spolu s příkazem pro přechod do tohoto stavu do zprávy. Zpráva je následně přidána do fronty Master Q.

Po výběru zprávy, stavový automat přechází na základě příkazu do tohoto stavu. Data jsou převedena z datového typu Variant a následně přivedena na selektor Case struktury, v níž pouze dojde k nastavení přepínače do polohy odpovídající přijaté informaci. Po vykonání přechází stavový automat do stavu IDLE.

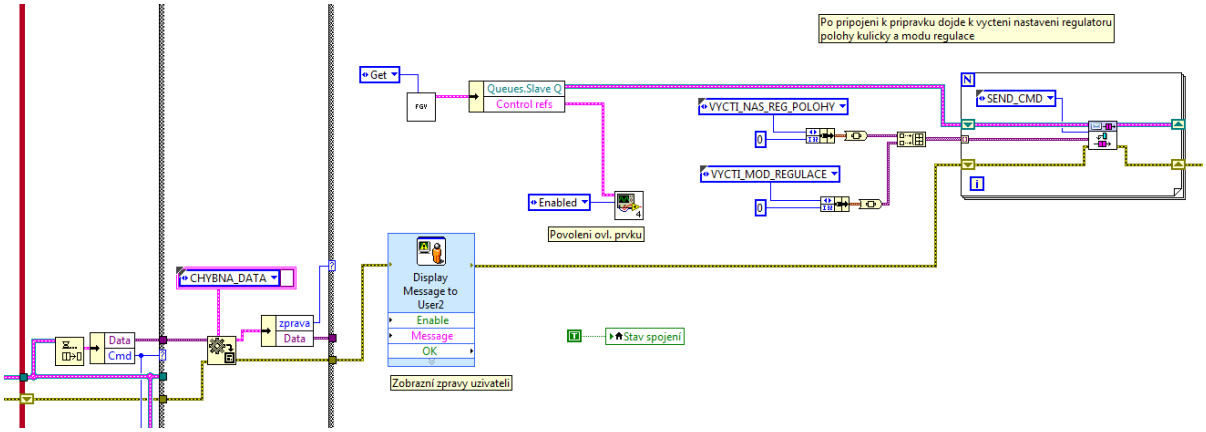


Obrázek 66 Stavový automat přijat mód regulace, MANUAL (vlevo), AUTO (vpravo)

• **OZNAMENI_UZIVATELI**

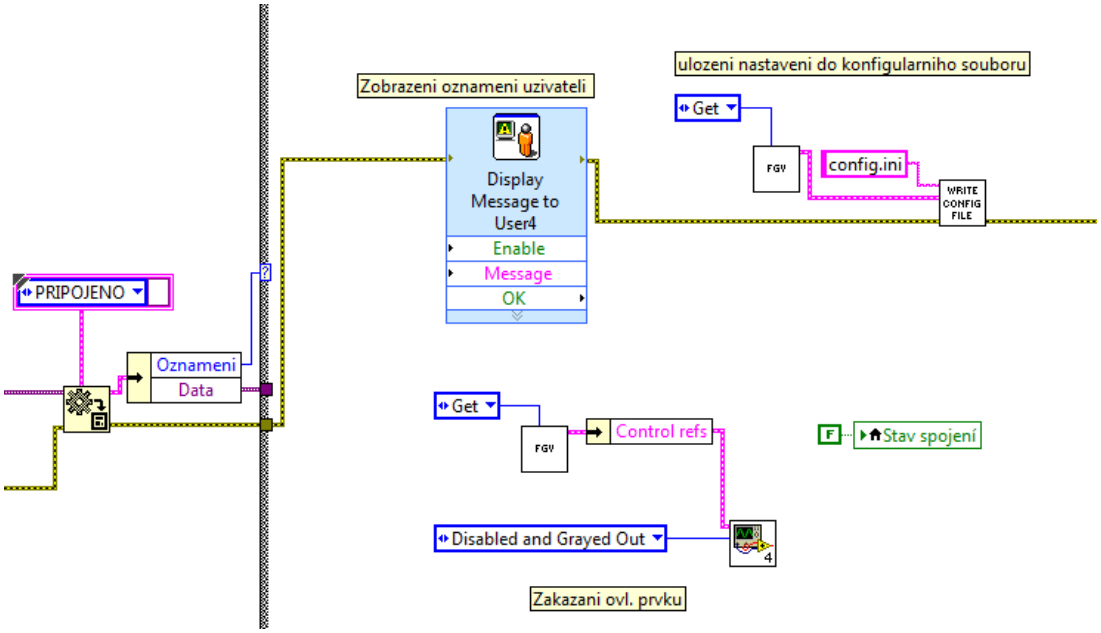
V některých okamžicích nastává situace, kdy je nutné zobrazit nějaké oznámení uživateli. Uživateli se může zobrazit jedno ze čtyř možných oznámení. K realizaci byla využita další struktura case, která se nachází v tomto stavu stavového automatu. Takže Data zprávy, která byla vybrána z fronty Master Q jsou tedy tvořena oznámením a daty. Nevýhodou je, že dokud uživatel nepotvrdí oznámení, je celá aplikace blokována. Po vykonání tohoto stavu přechází stavový automat do stavu IDLE.

Oznámení PRIPOJENO znamená, že je navázáno spojení mezi přípravkem a aplikací. Zobrazí se okno Připojeno (Obrázek 41), nastaví se hodnota indikátoru Stav spojení a uvolní se některé ovládací prvky. Zároveň se přidají dvě zprávy do fronty Slave Q. Jedná se o zprávy pro přechod stavového automatu paralelního SubVI do stavu, který zajistí vyčtení nastavení regulátoru polohy kuličky a módu regulace.



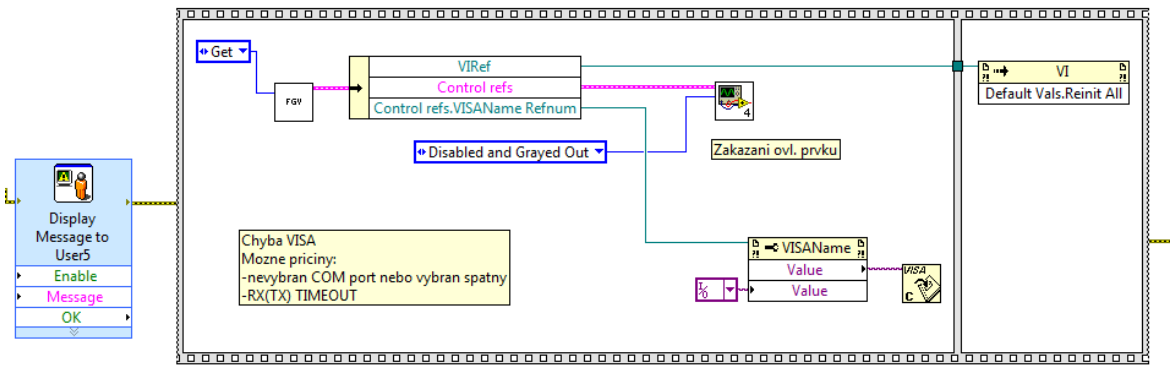
Obrázek 67 Stavový automat Oznámení uživateli PRIPOJENO

Oznámení ODPOJENO. Spojení mezi aplikací a přípravkem bylo ukončeno. Uživateli se zobrazí okno Odpojeno (Obrázek 44), dojde k zakázání ovládacích prvků, nastaví hodnota Stavu spojení a zapíše se nastavení do konfiguračního souboru.



Obrázek 68 Stavový automat Oznámení uživateli ODPOJENO

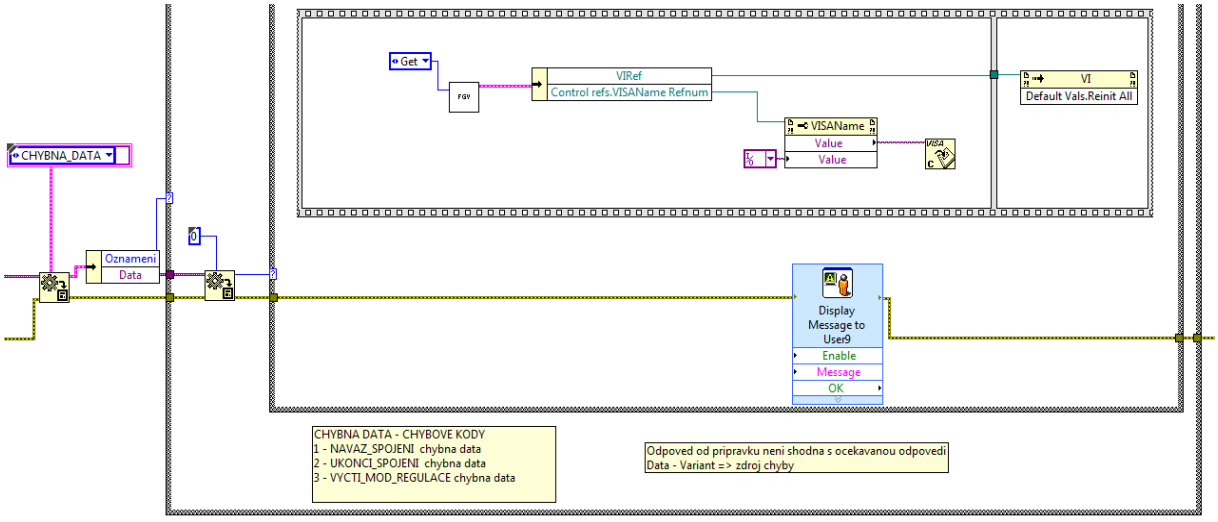
Oznámení CHYBA_VISA. Uživateli se zobrazí okno CHYBA KOMUNIKACE (Obrázek 39). Možné příčiny jsou vypršení timeoutu příjmu nebo vysílání, nevybrán COM port nebo je nedostupný. Dále dojde k zakázání ovládacích prvků, uzavření COM portu a vymazání názvu COM portu z prvku pro jeho výběr. Nakonec se nastaví defaultní hodnoty všech prvků, čím se uvede aplikace do stavu, v jakém byla po spuštění.



Obrázek 69 Stavový automat Oznámení uživateli CHYBA_VISA

Oznámení CHYBNA_DATA. Uživateli se může zobrazit jedno ze tří oznámení. Kdy záleží na zdroji chyby. Je možné obdržet chybnou odpověď od přípravku při připojování (kód chyby = 1), odpojování (kód chyby = 2) nebo může být chybně vyčten mód (kód chyby = 3). Kód chyby je získán z dat zprávy.

Na Obrázku 70, je stav, kdy byla obdržena chybná data při pokusu o spojení s přípravkem. Uživateli se zobrazí okno s chybou (Obrázek 40). Poté se vykonají stejné úkoly jako při CHYBA_VISA, pouze s rozdílem, že nedojde k zakázání ovládacích prvků.



Obrázek 70 Stavový automat Oznámení uživateli Chybná data - Nepřipojeno

Při příjmu chybných dat při odpojování se uvažuje, že k odpojení nedošlo. Uživateli se zobrazí chybové okno (Obrázek 43), tlačítko zůstává ve státu true tedy připojeno. Je tedy možné zkusit odpojit přípravek znovu.

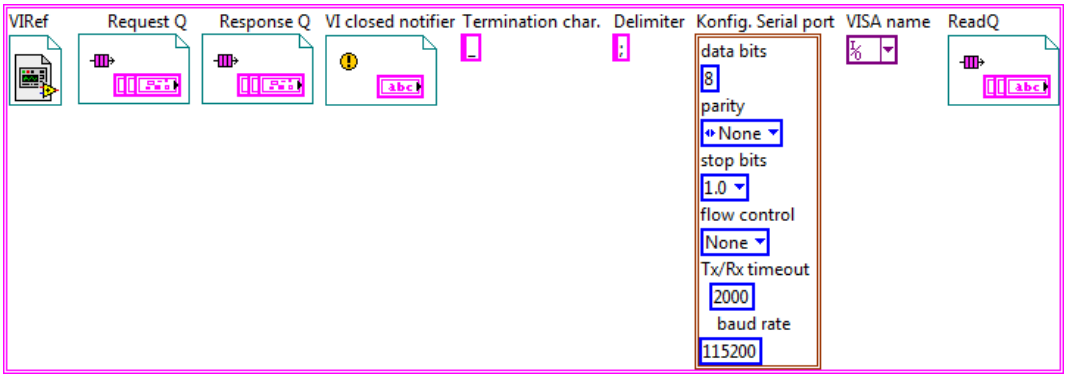
V případě, že došlo k chybnému vyčtení módu, uživateli se pouze zobrazí chybové okno (Obrázek 42). Nic dalšího se nevykoná.

6.4 Popis jednotlivých částí paralelního SubVi

Paralelní SubVI vykonává funkce, které se týkají komunikace s přípravkem a zpracováním přijatých dat. Komunikace se smyčkami v hlavním souboru je zajištěna pomocí front. SubVI je tvořeno stavovým automatem a paralelní smyčkou, která běží současně a slouží k periodickému čtení procesních dat od přípravku, pokud je regulace aktivní.

Toto SubVI má tři vstupy. Jedná se o název fronty tohoto VI, název fronty hlavního VI a název notifikátoru, který se vytvoří při uzavření tohoto VI. Předání názvů front je důležité pro zajištění vzájemné komunikace.

Data tohoto VI jsou uložena v clusteru, kdy datový vodič musí procházet pře stavový automat. Datový cluster, který se v tomto SubVI využívá je na Obrázku 71.



Obrázek 71 Datový cluster Paralelního SubVI

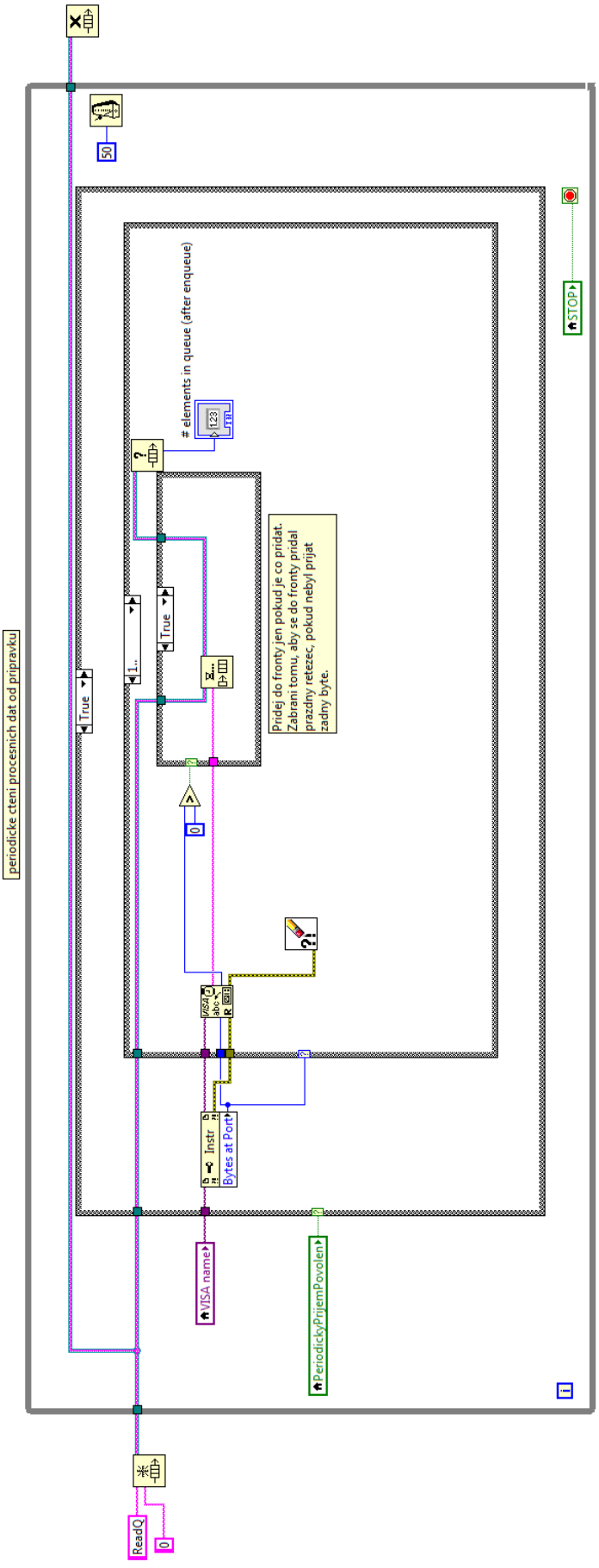
Jedná se tedy o cluster, který je tvořen následujícími devíti prvky:

- VIRef – reference na toto VI
- Request Q – reference na frontu příkazů tohoto VI
- Response Q – reference na frontu, do které se přidávají data z tohoto VI – fronta pro stavový automat v hlavním souboru
- VI closed notifier – reference na notifikátor, který se odesílá při ukončení tohoto VI
- Termination char. – zakončovací znak pro komunikaci s přípravkem
- Delimiter – oddělovač procesních hodnot v komunikaci s přípravkem
- Konfig. Serial port – nastavení sériové linky
- VISA name – COM port, ke kterému je připojen přípravek
- ReadQ – reference na frontu, do které se ukládají přijaté procesní hodnoty od přípravku

6.4.1 Paralelní smyčka periodického čtení dat

Pokud je regulace aktivní odesílá neustále přípravek po sériové lince procesní hodnoty do PC. Pro jejich příjem je využita paralelní smyčka While. Před spuštěním smyčky se získá reference na frontu Read Q, do které se budou přijatá data ukládat (Obrázek 72).

Dále záleží jestli je povolen periodický příjem dat. V případě, že není tak se v této smyčce nic nevykonává. Pokud povolen je, zjisťuje se jestli na daném seriovém portu jsou data ke čtení. Pokud jsou, dojde k jejich vyčtení jinak se nic nevykonává. Pokud je počet vyčtených bajtů větší jak nula, tedy došlo k úspěšnému vyčtení, přidají se data fronty, v opačném případě se nic nevykonává. Přijatá procesní data se poté zpracovávají ve stavu IDLE stavového automatu.

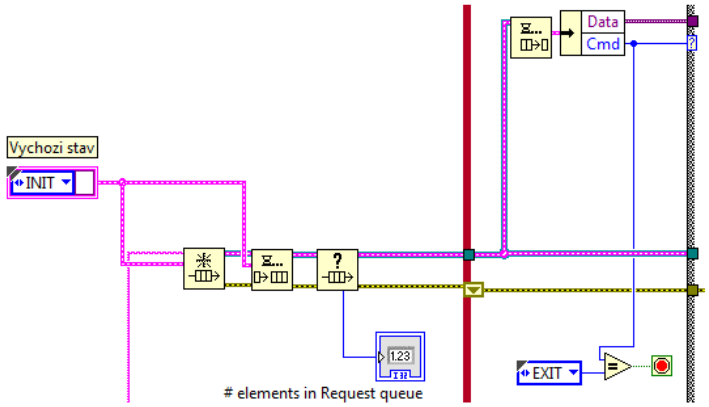


Obrázek 72 Smyčka periodického čtení procesních dat

6.4.2 Stavový automat

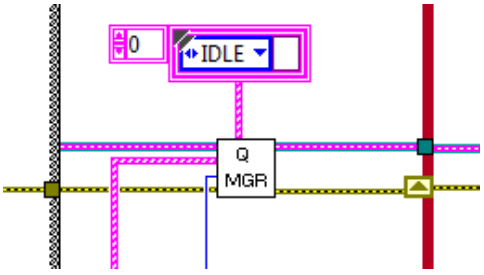
Zpracování přijatých dat a veškerá komunikace mezi aplikací a přípravkem, je realizována právě ve stavovém automatu tohoto SubVI. Kdy se vždy vykonává jeden z pěti stavů.

Před vstupem do stavového automatu je nejdříve získána reference na frontu tohoto VI a následuje přidání první zprávy do fronty, kterou je definován výchozí stav stavového automatu.



Obrázek 73 Získání reference fronty paralelního SubVI, vybrání prvku z fronty

Pokud je fronta stavového automatu prázdná vloží se do ní zpráva, jejíž příkaz (Cmd) zajistí přechod do stavu IDLE. Přidání zajistí Q MGR (Obrázek 74).



Obrázek 74 Přidání zprávy pokud je fronta prázdná

- **Stav INIT**

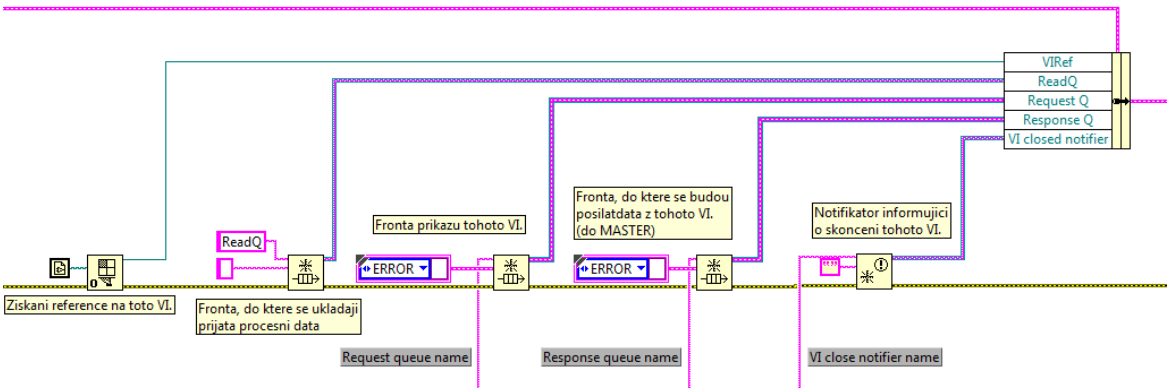
Po spuštění stavový automat přechází do výchozího stavu, v tomto případě se jedná o INIT. V tomto stavu je získáno několik referencí, které jsou uloženy do datového clusteru, aby mohly být dále využívány. Jedná se o následující reference: tohoto VI (VIRef), front ReadQ, RequestQ, ResponseQ a VI closed notifier. Kdy názvy dvou front a notifikátoru jsou definovány v hlavním souboru a jsou předávány do tohoto SubVI. Po vykonání stavový automat přechází do stavu IDLE.

Názvy front a notifikátoru (prvek – název):

Fronta RequestQ – SlaveQ

Fronta ResponseQ – MasterQ

Notifikátor VI closed notifier – Slave closed

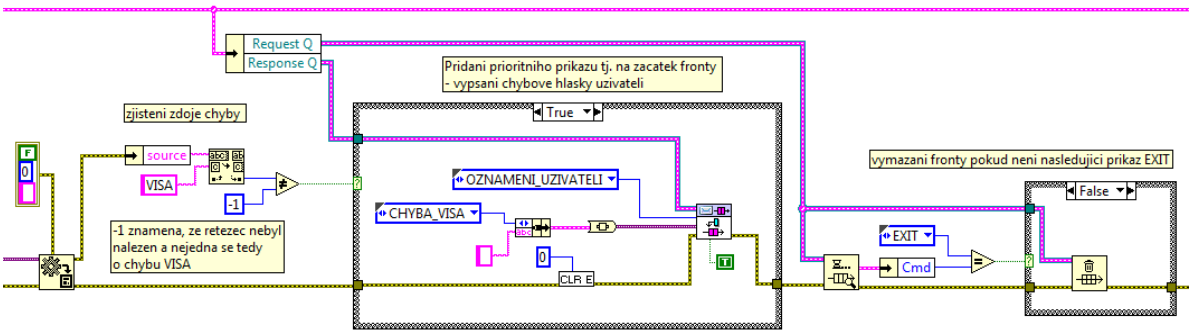


Obrázek 75 Slave stavový automat - IDLE

- **Stav ERROR**

Pokud dojde k jakékoliv chybě, která se projeví na žlutém vodiči, dochází k jejímu zpracování v SubVI Q MGR respektive v ENQUE ERROR. A přidáním zprávy se zajistí přechod do stavu ERROR, error cluster je uložen v datech této zprávy. V této aplikaci jsou ošetřeny pouze chyby, které se týkají komunikace s přípravkem.

Data zprávy vyjmuté z fronty se převedou zpět na původní error cluster. Následně vyhledáváme v názvu zdroje chyby řetězec VISA, protože chceme reagovat pouze na chyby v komunikaci. Pokud nebyl řetězec nalezen je na výstupu bloku s názvem offset of match hodnota -1 a nejedná se tak o chybu VISA. V case struktuře se tak nic nevykoná. V opačném případě, kdy byl řetězec nalezen, dochází k přidání prioritní zprávy do fronty Response Q, tedy do fronty hlavního souboru. Příkaz zprávy zajistí přechod stavového automatu, v hlavním souboru, do stavu OZNAMENI_UZIVATELI. Data dále obsahují oznámení, které určí, do jakého stavu se přepne Case struktura ve stavovém automatu.



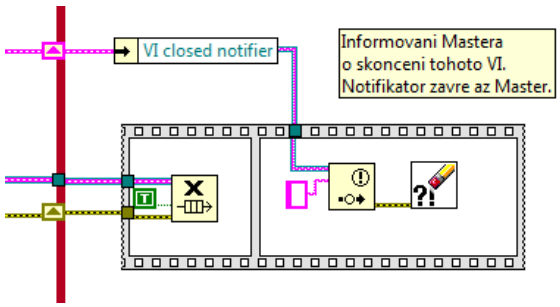
Obrázek 76 Slave stavový automat - ERROR

Bez jakékoliv vazby na vzniklou chybu se vždy kontroluje příkaz další zprávy ve frontě. Pokud je následující příkaz EXIT nic se nevykoná. V opačném případě dojde k vymazání všech prvků z fronty. Tím se zajistí, že Slave nebude po chybě komunikace reagovat na další příkazy ve frontě.

- **Stav EXIT**

V případě, že je příkaz zprávy vyjmuté z fronty EXIT je splněna podmínka pro ukončení smyčky While stavového automatu (Obrázek 73). Ten přechází do stavu EXIT, kde dojde pouze k nastavení hodnoty pomocného indikátoru STOP na true, který slouží pro zastavení smyčky periodického vyčítání procesních dat.

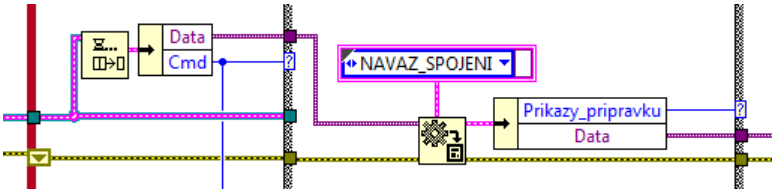
Po ukončení smyčky je nutné odeslat notifikátor, který oznamuje, že došlo k ukončení tohoto VI. Před odesláním notifikátoru je ještě uvolněna jeho fronta (Obrázek 77). Notifikátor si odchytí hlavní program, kterému je tímto signalizováno, že paralelní SubVI se ukončilo, následně se zavře celá aplikace.



Obrázek 77 Oznámení o ukončení Slave

- **Stav SEND_CMD -**

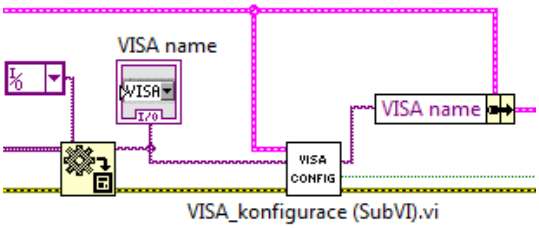
Veškerá komunikace s přípravkem kromě periodického čtení procesních hodnot se provádí v tomto stavu. Jednotlivým komunikačním příkazům odpovídá vždy jeden ze stavů vložené Case struktury. Pro odeslání nějakého příkazu přípravku se musí zpráva skládat z příkazu, který zvolí stav stavového automatu, a dat typu Variant, která byla původně cluster tvořený dvěma prvky Prikazy_pripravku a daty typu variant. Prikazy_pripravku mohou nabývat jedné z deseti hodnot. Po vykonání stavu se přechází do stavu IDLE



Obrázek 78 Slave stavový automat SEND_CMD výběr příkazu přípravku

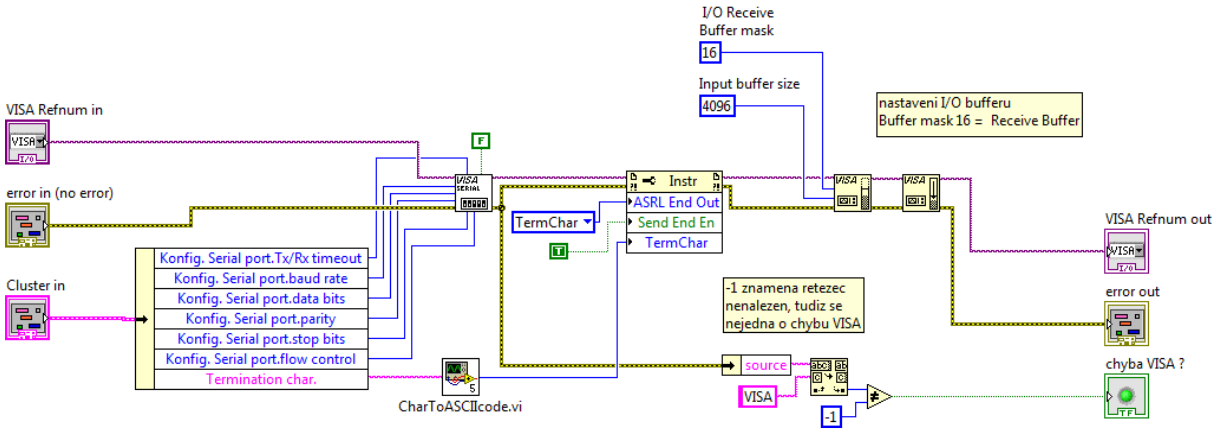
• **Stav SEND_CMD - NAVAZ_SPOJENI**

Jeden z nejdůležitějších příkazů sloužící ke spojení s přípravkem. Příkaz ke spojení přichází z hlavního VI. V datech je přenášen název COM portu, ke kterému je zařízení připojeno.



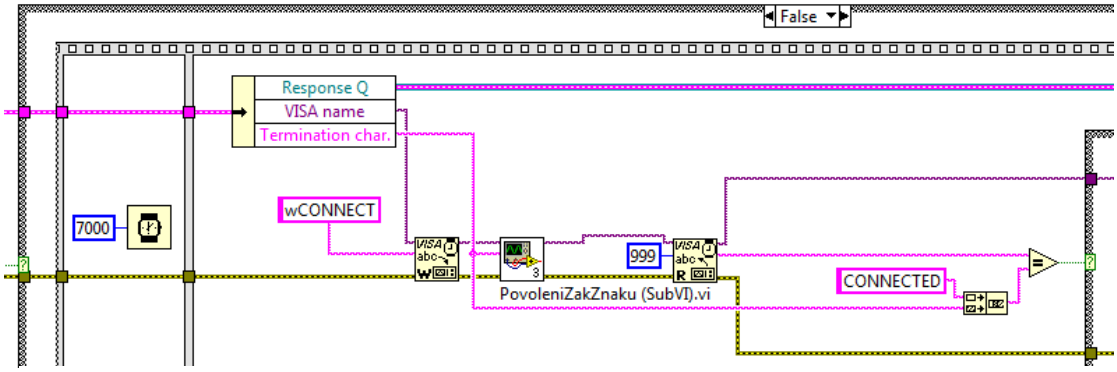
Obrázek 79 Navaž spojení 1

Po získání názvu COM portu proběhne nastavení sériové linky. Z prostorového hlediska bylo vytvořeno SubVI VISA CONFIG. SubVI má tři vstupy: VISA Refnum, Error in a Datový cluster. K vlastnímu nastavení sériové linky je využit blok VISA Configure Serial Port VI, kdy všechny nastavované parametry jsou uloženy v Datovém clusteru. Tím jsou usnadněny případné změny parametrů, které jsou všechny na jednom místě a snadno dohledatelné. Následuje nastavení zakončovacího znaku a bufferu pro příjem. Zároveň se kontroluje, zdali nedošlo k chybě spojené s VISA.



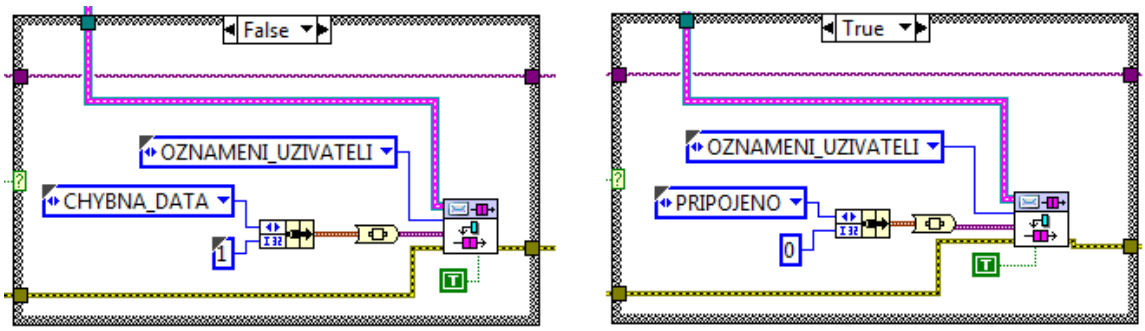
Obrázek 80 SubVI VISA CONFIG

Pokud proběhla konfigurace bez chyby, pokračuje se v navázání spojení. Jelikož při konfiguraci se otevře COM port, dojde k resetu řídící desky Arduino a rameno sjíždí na koncový spínač. Poté se rameno přenastaví do vodorovné polohy. Přípravek během této fáze nekomunikuje, proto je nastaveno čekání 7000ms. Poté se po sériové lince do přípravku odesílá příkaz pro připojení a povolí se zakončovací znak při příjmu. Následuje blok čtení, pro vyčítání dat dokud není přijat zakončovací znak. Přijatý řetězec se kontroluje s očekávanou odpovědí.



Obrázek 81 Navaž spojení 2

Na základě výsledku porovnání přijatého a očekávaného řetězce se vybere jeden ze stavů Case struktury. V obou případech se přidává se do fronty Master Q prioritní zpráva tvořená příkazem OZNAMENI_UZIVATELI a daty. Pokud jsou řetězce shodné, data jsou složená s oznámení PRIPOJENO a nulových dat. V případě, že by řetězce nebyly stejné, jsou data tvořena oznámením CHYBNA_DATA a chybovým kódem 1 (Obrázek 82).

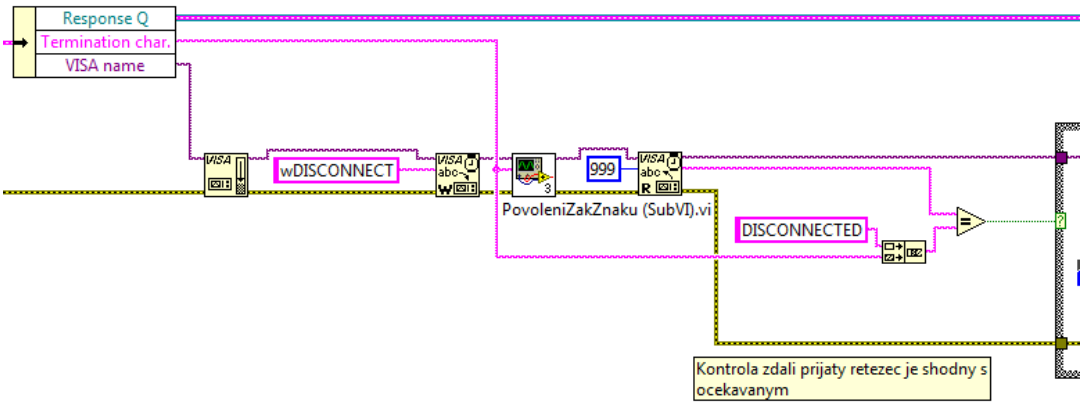


Obrázek 82 Přidání zprávy, řetězce nejsou shodné (vlevo), řetězce jsou shodné (vpravo)

Na závěr dojde k zakázání zakončovacího znaku, aby ve smyčce periodického vyčítání dat byla vždy vyčtena všechna data.

• **Stav SEND_CMD - UKONCI_SPOJENI**

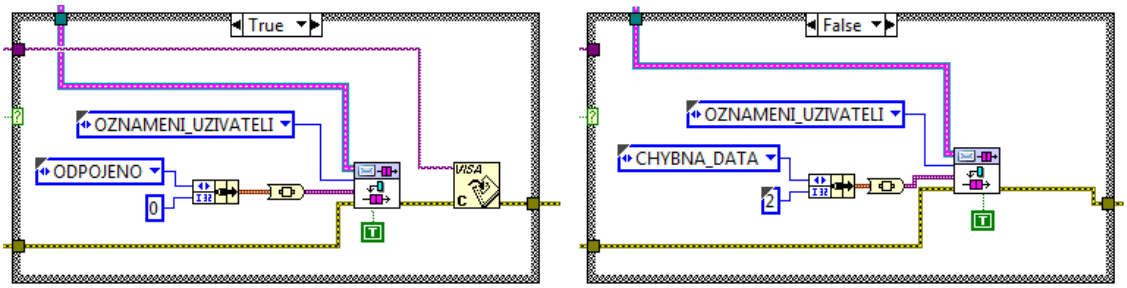
Stav, ve kterém se provede ukončení spojení s přípravkem. Jelikož mohou být v přijímacím bufferu procesní data, která by mohla být vyčtena místo odpovědi od přípravku, je nejdříve přijímací buffer vyprázdněn. Až poté je odeslán příkaz k ukončení spojení a povolen zakončovací znak při příjmu. Následně dochází ke čtení dat, které končí v okamžiku příjmu zakončovacího znaku. Přijatý řetězec porovnáváme s očekávaným.



Obrázek 83 Ukončení spojení

Ať už jsou řetězce shodné nebo rozdílné vždy se přidá prioritní zpráva do fronty Master Q. Pro oba případy má zpráva stejný příkaz a to OZNAMENI_UZIVATELI. Rozdíl je v datech, která jsou tvořená jiným oznámením a daty. V případě shody řetězců je oznámení ODPOJENO a nulová data. Pokud jsou řetězce rozdílené, je oznámení CHYBNA_DATA a data tvoří chybový kód 2.

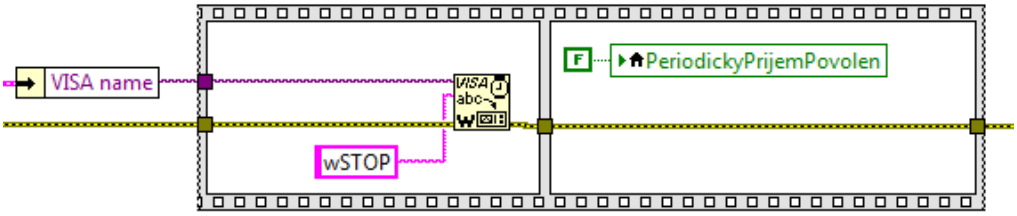
Pokud jsou řetězce shodné, znamená to, že odpojení proběhlo úspěšně a dojde k uzavření COM portu.



Obrázek 84 ODPOJENO přidání zprávy, řetězce jsou shodné (vlevo), nejsou shodné (vpravo)

• **Stav SEND_CMD - ZASTAV_REG**

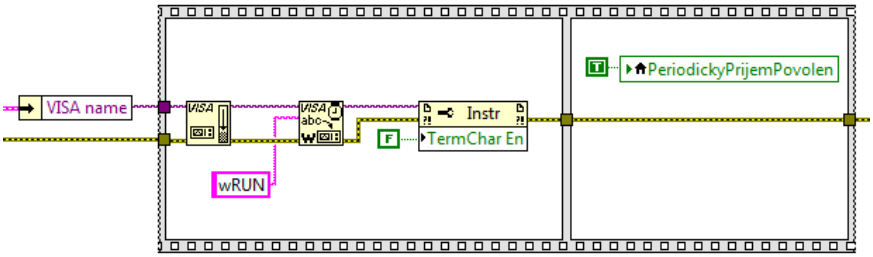
Dojde pouze odeslání příkazu k zastavení regulace a nastavení hodnoty pomocného indikátoru PeriodickyPrijemPovolen na false. Tím dojde k zastavení periodického vyčítání dat. Nejedná se o potvrzovaný komunikační příkaz, proto nedochází k žádné kontrole odpovědi.



Obrázek 85 Zastavení regulace

- **Stav SEND_CMD - SPUST_REG**

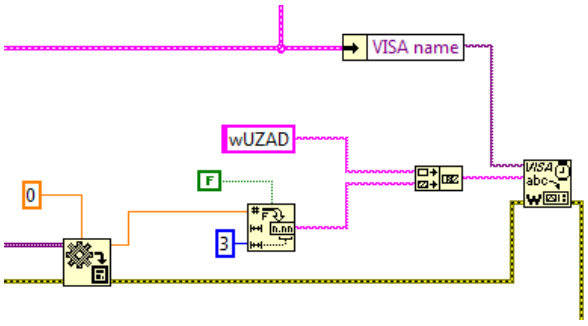
Před odesláním příkazu ke spuštění regulace se vyprázdní přijímací buffer. Po odeslání příkazu se zakáže příjem zakončovacího znaku, aby ve smyčce periodického čtení procesních dat docházelo k vyčítání všech dat. Poté se nastaví hodnota indikátoru PeriodickyPrijemPovolen na true, čímž se povolí periodické čtení.



Obrázek 86 Spuštění regulace

- **Stav SEND_CMD - NASTAV_ZADANY_UHEL**

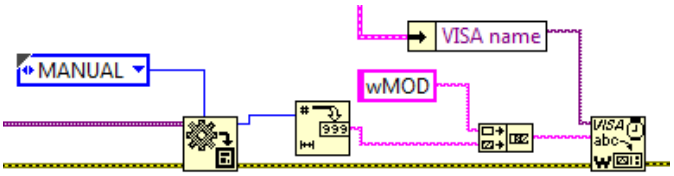
Odeslání příkazu k nastavení žádaného úhlu natočení ramene. V datech zprávy je žádaná hodnota úhlu, kterou převedeme na string. Poté ho spojíme se řetězcem wUZAD, čímž získáme celý komunikační příkaz, který je následně odeslán.



Obrázek 87 Odeslání žádaného úhlu

• **Stav SEND_CMD - NASTAV_MOD_REG**

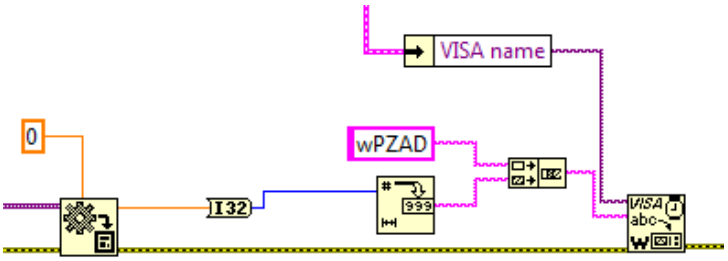
Tento stav slouží odeslání příkazu nastavení módu regulace. Požadovaný mód je v datech zprávy. Číslo převedeme na string a spojíme se stringem wMOD. Tento komunikační příkaz poté odesíláme po sériové lince.



Obrázek 88 Odeslání nastaveného módu regulace

• **Stav SEND_CMD - NASTAV_ZADANOU_POLOHU**

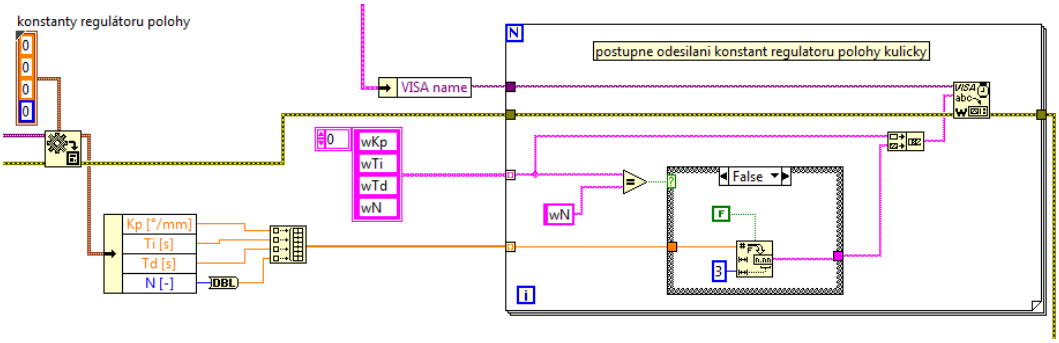
Odeslání žádané polohy kuličky. Žádanou polohy získáme z dat zprávy, převedeme na string, spojíme se stringem wPZAD a tento složený string odesíláme po sériové lince.



Obrázek 89 Odeslání žádané polohy kuličky

• **Stav SEND_CMD - NASTAV_REG_POLOHY**

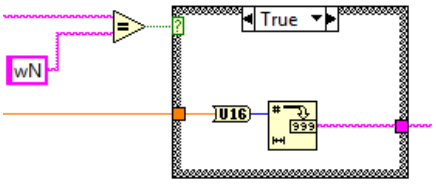
Odeslání nastavených konstant regulátoru polohy kuličky. Jednotlivé hodnoty získané z dat zprávy se postupně spojují s příslušnou textovou částí komunikačního příkazu a odesílají ve smyčce For.



Obrázek 90 Odeslání nastavených konstant regulátoru

Pokud se odesílá jeden z příkazů pro nastavení Kp, Ti nebo Td převádí se číselná hodnota na string s přesností na 3 desetinná místa (Obrázek 90).

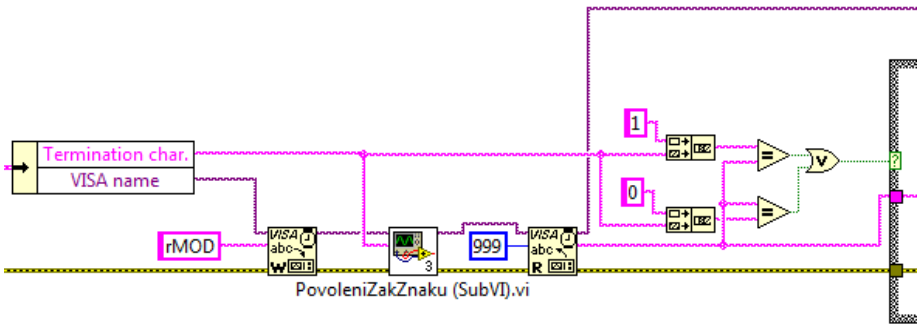
V případě odesílání konstanty N, což je vždy celé číslo, probíhá převod na string jiným způsobem (Obrázek 91).



Obrázek 91 Převod celého čísla N na string

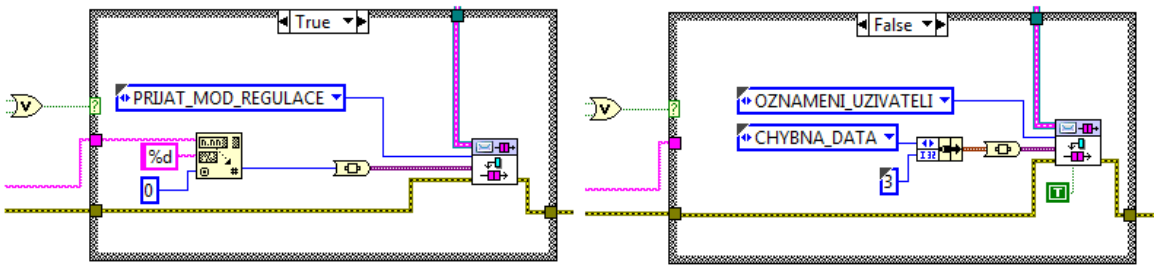
- **Stav SEND_CMD - VYCTI_MOD_REGULACE**

Odešleme příkaz pro vyčtení nastaveného módu regulace přípravku a povolíme zakončovací znak při příjmu. Data jsou vyčítána, dokud se nenačte zakončovací znak. Jelikož mód regulace může nabývat pouze jedné z hodnot 1 nebo 0 je tedy snadné provést kontrolu, zdali je přijatý řetězec správný.



Obrázek 92 Vyčtení módu regulace

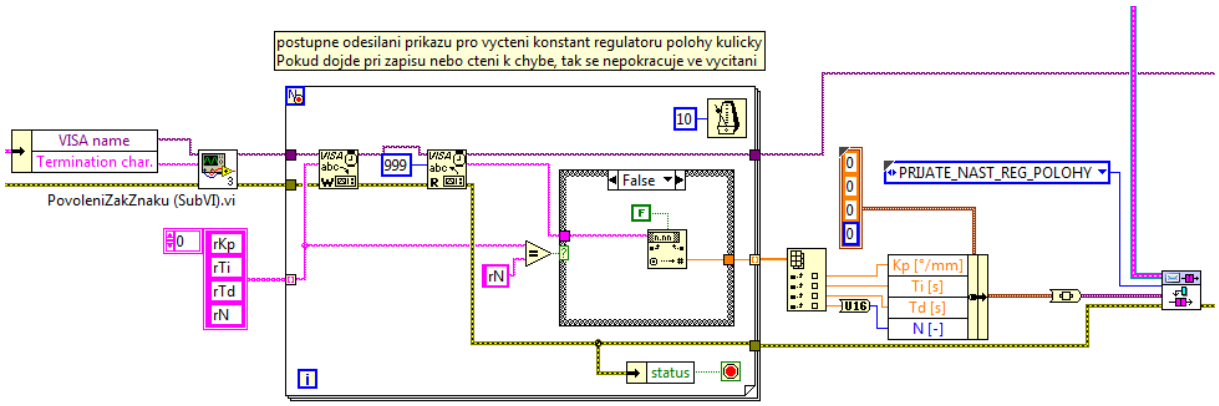
V obou případech tedy ať až je přijatý řetězec správný nebo chybný je přidána do fronty Master Q zpráva. V případě správného řetězce dojde k jeho převedení na číslo a přidání do zprávy jako data, příkaz zprávy je PRIJAT_MOD_REGULACE. V případě nesprávného řetězce, nejsou tato data dále zpracovávána. Pouze se přidá zpráva s příkazem OZNAMENI_UZIVATELI a daty složenými z oznámení CHYBNA_DATA a dat, která jsou reprezentována chybovým kódem 3.



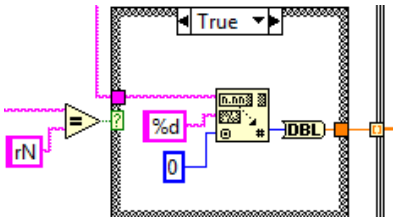
Obrázek 93 Přidání zprávy, přijat mód regulace (vlevo), nepřijat mód - chybná data (vpravo)

- **Stav SEND_CMD - VYCTI_NAS_REG_POLOHY**

Vyčtení nastavených konstant regulátoru polohy kuličky z přípravku. Nejdříve se povolí zakončovací znak při příjmu. Poté postupně dochází k odesílání příkazu pro vyčtení dané konstanty regulátoru a následnému vyčítání dokud není vyčten zakončovací znak. V případě Kp, Ti a Td se uvažuje, že se jedná o desetinná čísla, čemuž odpovídá převod řetězce na číslo (Obrázek 94). Převod řetězce konstanty N, což je vždy celé číslo, je na Obrázku 95.



Obrázek 94 Vyčtení nastavení regulátoru polohy kuličky



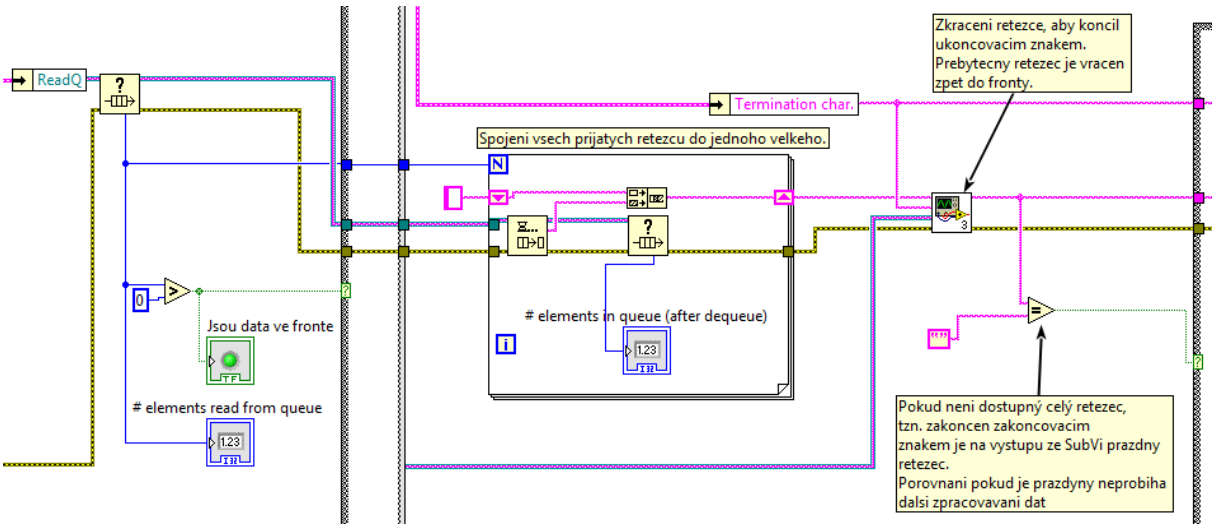
Obrázek 95 Převod přijatého řetězce N na celé číslo

Ze smyčky For získáváme 1D pole prvků, které rozdělíme na jednotlivé prvky, ze kterých následně vytvoříme cluster. Nakonec přidáváme zprávu do fronty Master Q. Kde příkaz zprávy je PRIJATE_NAST_REG_POLOHY a cluster konstant regulátoru převedený na datový typ Variant tvoří data. Poté ještě následuje zakázání zakončovacího znaku.

• **Stav IDLE**

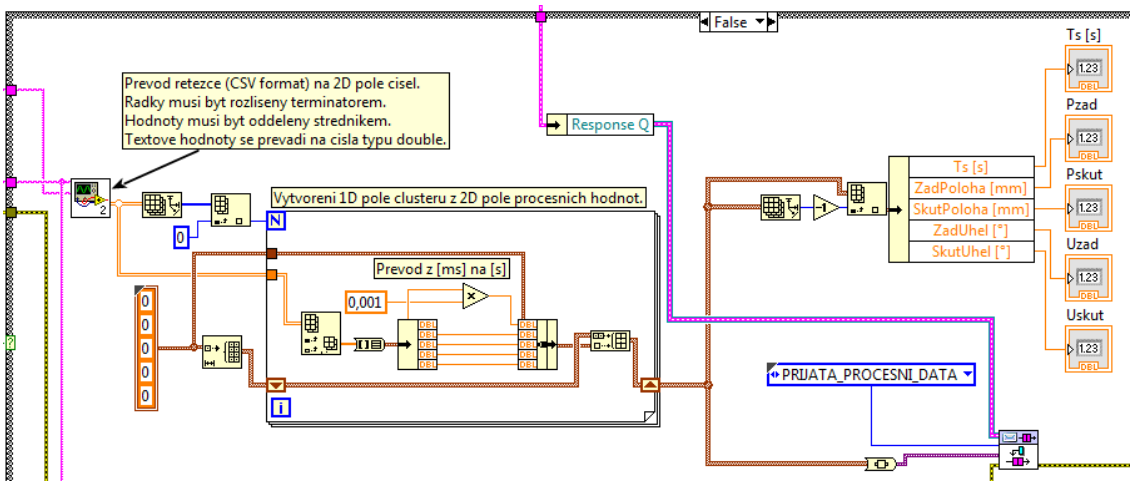
V tomto stavu dochází ke zpracování přijatých procesních dat paralelní smyčkou periodického vyčítání dat. Kdy tato přijatá data jsou vybírána z fronty ReadQ. Pokud jsou nějaká data ve frontě, dochází k jejich zpracování.

První zpracování dat spočívá ve spojení všech přijatých řetězců do jednoho velkého řetězce. Toto je provedeno ve smyčce For. Počet vykonání smyčky je dán počtem prvků ve frontě ReadQ. Získáme tedy jeden dlouhý řetězec, který ale nemusí končit zakončovacím znakem, protože většinu času je zakončovací znak při příjmu zakázán, aby došlo k vyčtení veškerých dat. Je zde tedy využito SubVI, které kontroluje, zdali řetězec končí zakončovacím znakem. V případě, že ano pokračuje se ve zpracování. V ostatních případech zkrátí vstupní řetězec tak, aby končil zakončovacím znakem. Přebytečnou část vrací zpět do fronty ReadQ. Pokud by však při další iteraci této smyčky byla ve frontě pouze vrácená přebytečná část, tedy bez zakončovacího znaku vracelo by toto SubVI prázdný řetězec. Proto je zde zařazena kontrola, jestli není výstupní řetězec prázdný. Pokud je, nedochází k dalšímu zpracování.



Obrázek 96 Stav IDLE 1

Většinou výstupní řetězec není prázdný a je dále zpracováván v následující struktuře Case (Obrázek 97). Pomocí SubVI je řetězec převeden na 2D pole čísel. Toto 2D pole čísel je následně převedeno na 1D pole clusteru. Zároveň dochází k přepočtu hodnoty Periody vzorkování Ts z milisekund na sekundy. Nakonec dochází k přidání zprávy do fronty MasterQ. Zpráva je tvořena příkazem PRIJATA_PROCESNI_DATA a daty - clusterem procesních hodnot, který byl převeden na datový typ Variant.



Obrázek 97 Převod řetězce procesních dat na číslo a vytvoření zprávy s těmito daty

7. TESTOVÁNÍ APLIKACE

Po vytvoření ovládací aplikace mělo následovat vlastní nastavení regulátoru polohy kuličky a porovnání simulačního modelu s reálnou soustavou. Z časových důvodů bylo od těchto bodů upuštěno.

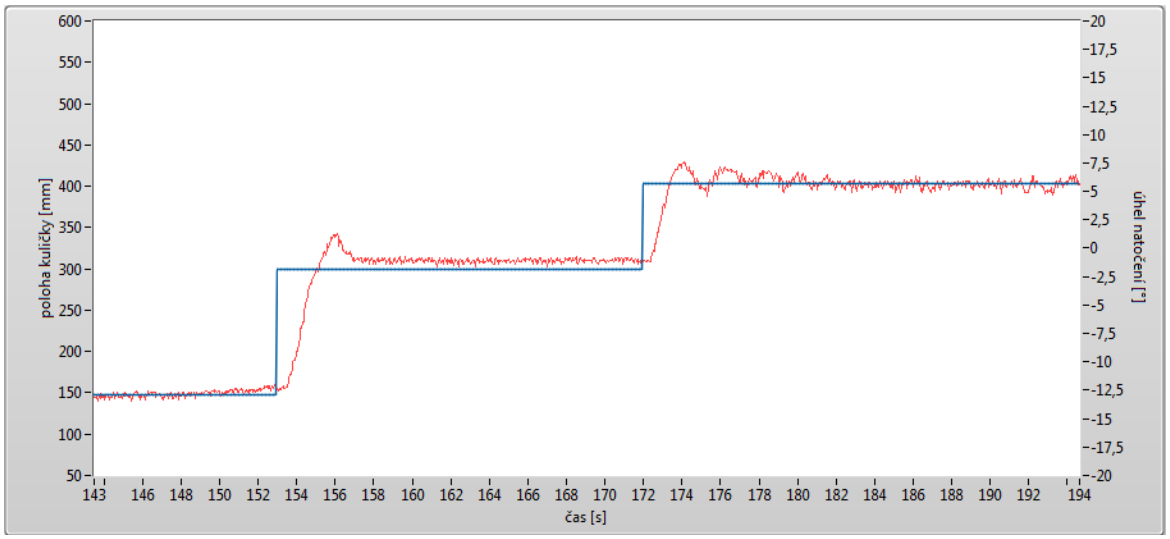
Jelikož nedošlo ke změnám časování ve firmwaru přípravku oproti původnímu, byly jako výchozí hodnoty konstant regulátoru použity hodnoty zjištěné metodou pokus omyl, které byly využity v původním programu.

Původní nastavení PID regulátoru
Kp = 0,05 °/mm
Td = 1 s
Ti = 3 s
N = 3
Kde N je filtrace derivační složky.

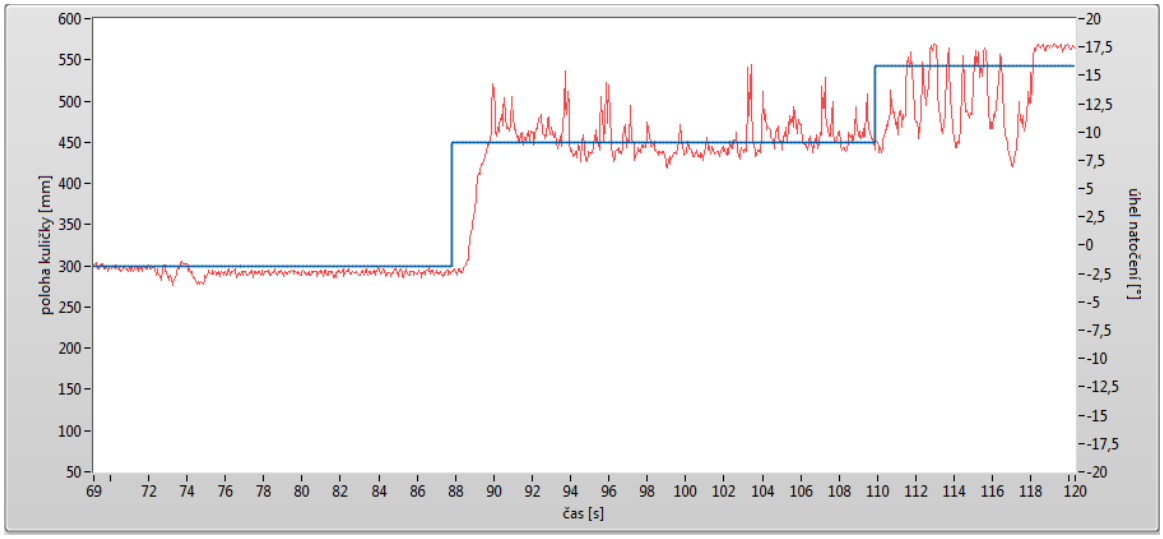
S těmito konstantami bylo provedeno několik pokusných měření. Jedno z nich je na Obrázku 98. Jedná se o průběh žádané (modrá) a skutečné (červená) polohy v případě aktivní zpětnovazební regulace polohy kuličky.

Při měření bylo provedeno několik skokových změn žádané polohy. Po změně žádané polohy došlo i ke změně skutečné polohy, avšak vždy s překmitem. Skutečná hodnota je stále zakmitaná a to z důvodu nedokonalého snímání kuličky. Nastává zde totiž problém, že použitý pingpongový míček je velice lehký a je z vytvořen z materiálu, který se snadno odráží. V kombinaci s hliníkovou lištou dochází k nežádoucímu efektu, kdy míček "skáče", čímž je znemožněno přesné určení polohy pomocí použitého snímače.

Tento nežádoucí efekt se výrazně projeví již při zadání větší hodnoty žádané polohy jak 400 mm. V extrémních případech může nastat i situace, že se míček odrazí úplně mimo lištu. V tomto případě dojde k natočení ramene na +15°, protože snímač polohy indikuje hodnotu 569 mm. Pro nižší hodnoty k odrazům také dochází, ale ve většině případů se výrazně neprojeví. Příklad průběhu, kdy se kulička odráží od lišty je na Obrázku 99. Jedná se strmé překmity v průběhu žádané polohy kuličky.



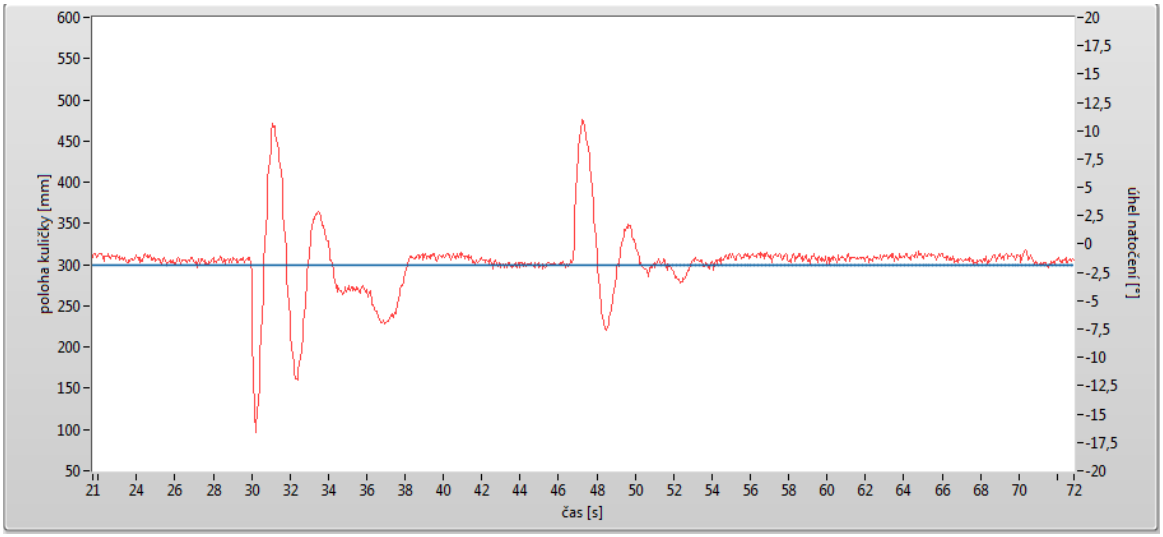
Obrázek 98 Zpětnovazební regulace, žádaná poloha – modrá, skutečná poloha - červená



Obrázek 99 Zpětnovazební regulace - Odrazy, žádaná poloha – modrá, skutečná poloha - červená

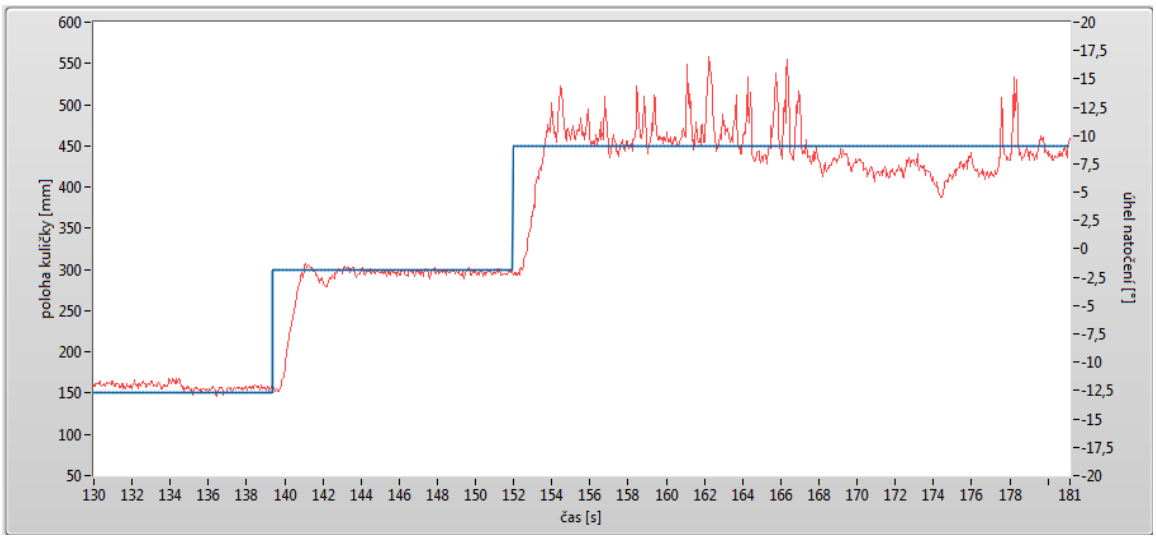
Skutečná poloha byla ustálená na 300 mm, poté došlo ke skokové změně na 450 mm. Vlivem odrazů došlo k výrazným špičatým překmitům. Po chvíli se kulička ustálila, následně ale došlo k dalším odrazům. V případě skokové změny hodnoty na 540 mm se míček neustále odrážel od lišty směrem nahoru, tedy jakoby mimo lištu. Nebylo tak možné přesně snímat jeho skutečnou polohu, což je z průběhu jasně patrné. Na konci míček úplně vyskočil z lišty a snímač zobrazoval hodnotu přibližně 568mm

Na obrázku 100 je průběh žádané a skutečné polohy při aktivní zpětnovazební regulaci polohy kuličky. Žádaná poloha je konstantní. Cvrnknutím prstem do kuličky byla nejdříve vychýlena na minimální a po ustálení i na maximální hodnotu.



Obrázek 100 Zpětnovazební regulace - vychýlení kuličky 1, žádaná poloha – modrá, skutečná poloha - červená

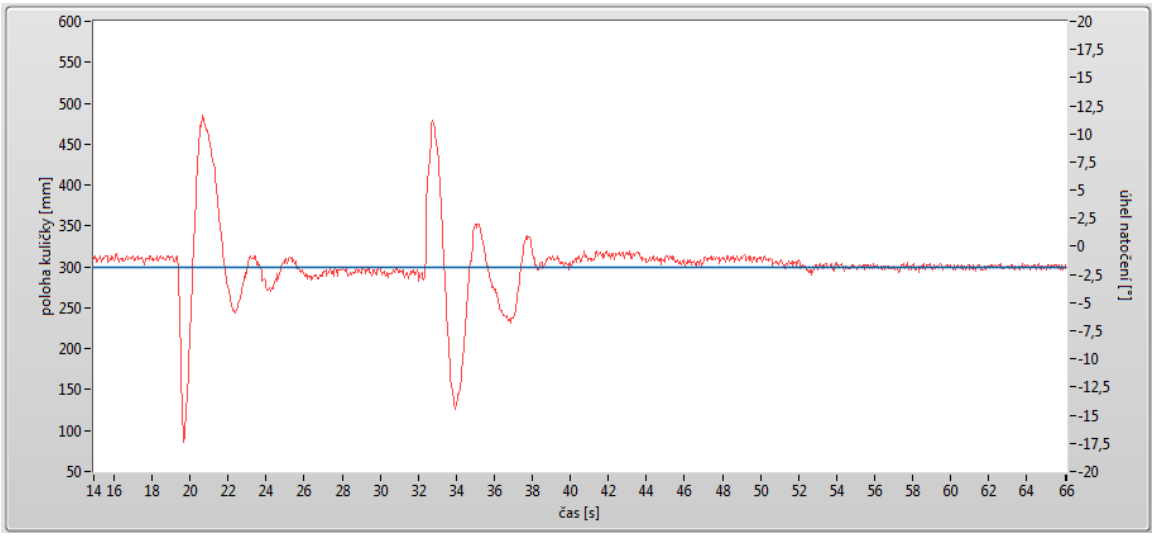
Dále testování probíhalo změnami nastavení konstant regulátoru, aby došlo k omezení překmitu a celkově ke zlepšení regulačního procesu. Omezení překmitu při skokové změně hodnoty žádané hodnoty je možné dosáhnout zvětšením časové integrační konstanty Ti. Hodnota Ti tedy byla zvětšena z 3 s na 10 s. Průběhy odezvy skutečné hodnoty na skokové změny žádané hodnoty s tímto nastavením jsou na Obrázku 101.



Obrázek 101 Zpětnovazební regulace, žádaná poloha – modrá, skutečná poloha - červená

Při skokové změně ze 150 mm na 300 mm nedošlo k výraznému překmitu, při další změně na 400 mm se projevily odrazy míčku od lišty a skutečná poloha je tak velice rozkmitaná.

Na obrázku 102 průběh žádané a skutečné polohy při aktivní zpětnovazební regulaci polohy kuličky. Žádaná poloha je konstantní. Cvrnknutím prstem do kuličky byla nejdříve vychýlena na minimální a po ustálení i na maximální hodnotu.



Obrázek 102 Zpětnovazební regulace - vychýlení kuličky 2, žádaná poloha – modrá, skutečná poloha - červená

Jedno z možných řešení jak eliminovat odrazy od lišty je změna kuličky. Z běžně dostupných kuliček byly dosaženy relativně dobré výsledky s golfovým míčkem. Kde se ale musí brát v úvahu jeho mnohem vyšší hmotnost, která je 46 g. Pingpongový míček váží přibližně 3 g.

Aby byl regulační proces uspokojivý, musely být změněny konstanty regulátoru, což souvisí především s hmotností míčku. Během testování, se ale vyskytlo několik problémů. První byl ve snímání polohy míčku. Ten, i když stál na místě tak byla hodnota ze snímače rozkmitaná. Což bylo pravděpodobně dáno lesklým a nerovnoměrným povrchem, který způsoboval i druhý problém a to složitější uvedení míčku do pohybu. Tyto dva problémy byly výrazně omezeny obroušením povrchu golfového míčku. Na obrázku 103 je podoba původního a obroušeného míčku. Obroušením nedošlo ke snížení hmotnosti.

Obrázek 103 Golfové míčky, původní podoba míčku (vlevo), obroušený míček (vpravo)

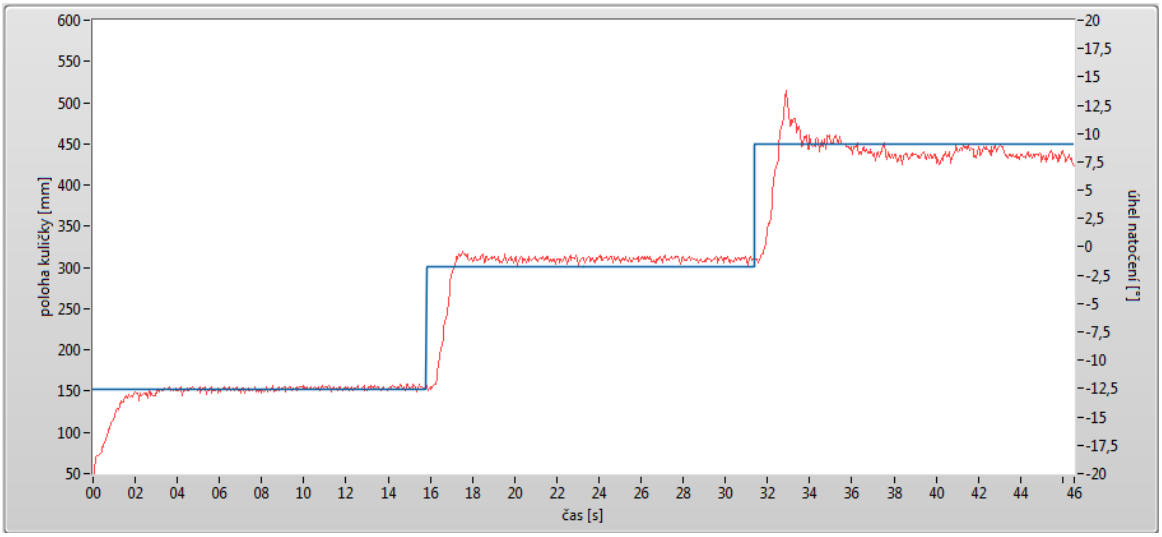
Metodou pokus omyl byly zjištěny následující konstanty regulátoru polohy, při kterých byl regulační proces uspokojivý.

Kp = 0,08 °/mm

Ti = 35 s

Td = 0,5 s

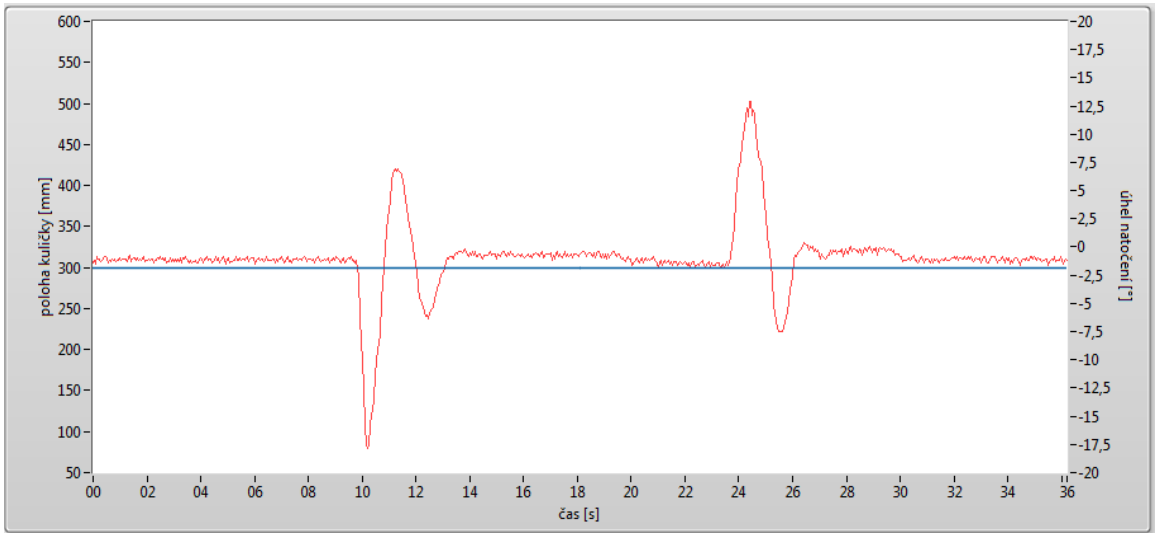
N = 3



Obrázek 104 Zpětnovazební regulace – golfový míček, žádaná poloha – modrá, skutečná poloha - červená

Na obrázku 104 je průběh žádané a skutečné polohy, kdy byl využit golfový míček a výše uvedené nastavení regulátoru. Oproti průběhům, kde byl využit pingpongový míček, je průběh skutečné polohy méně zvlněný, čemuž výrazně pomohlo omezení odrazů od lišty. Ale odrazy se nepovedlo eliminovat úplně. Při větších hodnotách žádané polohy dochází k stále odrazům.

Na obrázku 105 průběh žádané a skutečné polohy při aktivní zpětnovazební regulaci polohy kuličky. Žádaná poloha je konstantní. Cvrnknutím prstem do kuličky byla nejdříve vychýlena na minimální a po ustálení i na maximální hodnotu.



Obrázek 105 Zpětnovazební regulace – golfový míček vychýlení, žádaná poloha – modrá, skutečná poloha - červená

ZÁVĚR

Jedním z cílů této práce bylo upravit původní firmware laboratorní přípravku tak, aby byla jeho správa jednodušší a bylo ho možné rozšířit o další funkce. Původní firmware řídící jednotky umožnoval pouze jednu základní funkci a to zpětnovazební regulaci polohy kuličky. Žádaná poloha byla určena hodnotou proměnné ve firmwaru, který byl nahrán v řídící desce Arduino. Práce je možné rozdělit na několik hlavních částí.

Důležitou částí bylo stanovení komunikačního protokolu, aby byla možná obousměrná komunikace s přípravkem. Což umožní nejen vyčítat aktuální hodnoty parametrů, ale také měnit hodnoty některých parametrů. Implementací obousměrné komunikace je tedy možné ovládat zařízení za chodu pomocí aplikace, která je spuštěna na počítači.

Jednou z hlavních částí byla vlastní úprava firmwaru přípravku. Proběhla úprava vzorového příkladu, který mi byl poskytnut vedoucím práce. Program bylo nutné upravit tak, aby ho bylo možné využít pro tento konkrétní přípravek. Zároveň došlo k rozšíření o několik dalších funkcí. Firmware pro zařízení byl vytvořen v programu Atmel Studio 6.1 s využitím šablony, která umožňuje psát programy pro Arduino. K tvorbě byl využit programovací jazyk C++. Firmware byl při tvorbě průběžně testován na laboratorním přípravku.

Dalším úkolem bylo vytvoření ovládací aplikace, pomocí které bude možné laboratorní přípravek ovládat za chodu. Zároveň musela aplikace splňovat zadané požadavky. K vytvoření softwaru bylo využito vývojového prostředí LabVIEW. Využitá architektura i jednotlivé části programu byly důkladně popsány. Ovládací software byl testován a odladěn na laboratorním přípravku. Ovládací aplikace je plně funkční a splňuje požadavky, které byly stanoveny.

Při závěrečném testování byl objeven hlavní nedostatek laboratorního přípravku, který spočívá v ne úplně vhodném výběru kuličky. Proběhly pokusy s golfovým míčkem, ale nejedná se také o úplně vhodné řešení. Bylo by tedy dobré dát prostor pro otestování funkce s jinými kuličkami s různou hmotností nebo z jiných materiálů a najít tak optimální řešení.

Výsledkem této práce je tedy firmware pro řídící jednotku přípravku a ovládací software, pomocí kterého lze tento přípravek ovládat z počítače. Ve spojení s již vyrobeným zařízením, je tak laboratorní přípravek připraven k použití pro výuku.

LITERATURA

[1] BUREŠ, Zdeněk. Návrh a realizace modelu ”Kulička na rameni”. Pardubice, 2015. Diplomová práce. Univerzita Pardubice.

[2] Podklady od vedoucího práce

[3] Arduino Mega2560. In: Forum.arduino.cc [online]. [cit. 2017-05-24]. Dostupné z: http://forum.arduino.cc/index.php?topic=322175.0

[4] Arduino.cz [online]. [cit. 2017-05-24]. Dostupné z: https://arduino.cz/

[5] Introduction to LabVIEW Design Patterns [online]. In: . s. 60 [cit. 2017-05-24]. Dostupné z: https://www.ieee.li/pdf/viewgraphs/labview_design_patterns.pdf

[6] VLACH, J., HAVLÍČEK, J., VLACH, M. 2008. Začínáme s LabVIEW. Praha: BEN – technická literatura. 247 s. ISBN 978-80-7300-245-9.

SEZNAM OBRÁZKŮ