VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2017 František Majchrák

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

MĚŘENÍ VÝPOČETNÍHO VÝKONU PLC PFC200
PLC PFC200 BENCHMARKING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
František Majchrák

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Jakub Arm

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Automatizační a měřicí technika**
Ústav automatizace a měřicí techniky

Student: František Majchrák ***ID:*** 173692
Ročník: 3 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Měření výpočetního výkonu PLC PFC200

POKYNY PRO VYPRACOVÁNÍ:

1) Seznamte se s PLC PFC200 firmy WAGO.
2) Seznamte se s možnostmi implementace vlastních programových bloků v prostředí embedded Linux a jejich propojení s programem v PLC prostředí.
3) Porovnejte PLC a embedded Linux prostředí pomocí benchmarku, který definuje certifikát TC3.
4) Vytvořte demonstrační program v PLC prostředí, který bude zobrazovat všechny důležité informace o vytížení celého systému pomocí vytvořeného programu v embedded Linux prostředí.

DOPORUČENÁ LITERATURA:

Martinásková M., L. Šmejkal. Řízení programovatelnými automaty, vydavatelství ČVUT, Praha, 2004.

Bruce Eckel. Myslíme v jazyku C++. Grada, 2000. ISBN 80-247-9009-2.

Termín zadání: 6.2.2017 ***Termín odevzdání:*** 29.5.2017

Vedoucí práce: Ing. Jakub Arm
Konzultant: Jakub Svoboda

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

ABSTRAKT

V práci sa analyzujú možnosti merania výpočetného výkonu PLC PFC200 od firmy WAGO. Prvú časť práce venujem priblíženiu funkčnosti automatu, jeho využitiu, jednotlivým možnostiam programovania, zoznamujem sa s implementáciou vlastných programových blokov v prostredí embedded Linux a zisťujem jeho prepojenie s PLC prostredím. V druhej časti práce sa sústredím na meranie výkonu, jeho typy, popis certifikovaných meraní a jeho celkové zhodnotenie. V poslednej časti je vysvetlená aplikácia, ktorá zobrazuje vyťaženie celého systému PLC.

KĽÚČOVÉ SLOVÁ

embedded Linux, implementácia programových blokov, meranie výpočetného výkonu, PLC PFC200, vyťaženie systému, WAGO

ABSTRACT

The paper analyzes the possibilities of benchmarking PLC PFC200 from WAGO. The first part of the paper is dedicated to functional approximation of the machine, its possible use, individual programming options, to get acquainted with the possibilities of implementing its own program blocks in embedded Linux environment and determine their interconnection possibilities with PLC environment. In the second part I focus on benchmarking, benchmarking types, a description of certified measurements and assessment of measurement. The last part explains an application that displays the load of the entire PLC system.

KEYWORDS

benchmarking, embedded Linux, implementation of the program blocks, PLC PFC200, system load, WAGO

MAJCHRÁK, František *Měření výpočetního výkonu PLC PFC200*: bakalárska práca. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2017. 55 s. Vedúci práce bol Ing. Jakub Arm

VYHLÁSENIE

Vyhlasujem, že som svoju bakalársku prácu na tému „Měření výpočetního výkonu PLC PFC200“ vypracoval(a) samostatne pod vedením vedúceho bakalárskej práce, využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor(ka) uvedenej bakalárskej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto bakalárskej práce som neporušil(a) autorské práva tretích osôb, najmä som nezasiahol(-la) nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý(-á) následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákoníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora(-ky)

PODAKOVANIE

Rád by som poďakoval vedúcemu práce pánovi Ing.Jakubovi Armovi za odborné vedenie, konzultácie, trpezlivosť a podnetné návrhy k práci.

Brno ....

podpis autora(-ky)

OBSAH

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK

ZOZNAM VÝPISOV

ÚVOD

Táto práca sa spočiatku zaoberá teoretickým popisom procesorového modulu PLC PFC200. Popis je rozdelený do niekoľkých základných častí ako je napríklad popis hardwaru, popis softwaru, programovanie modulu, napájanie, komunikácia s modulom... . Popis je zostavený za účelom zoznámenia sa s prístrojom pre ďalšiu prácu s ním.

Ďalej je v práci popísané meranie výkonu na PLC prístrojoch podľa certifikátu TC3. V tejto časti je popísaná realizácia merania, požiadavky na meranie a zdokumentovanie merania. Meranie výkonu sa delí na aplikačne a jazykovo orientované meranie.

V praktickej časti práce je dopodrobna rozobraté meranie a sú tam uvedené namerané hodnoty ako aj spôsob akým sa k ním dá dopracovať. Ďalej tam je popísaná aplikácia monitorujúca vyťaženie celého systému PLC, napr. vyťaženie CPU, vyťaženie pamäte, čas cyklu PLC a podobne.

1 TEORETICKÁ ČASŤ PRÁCE

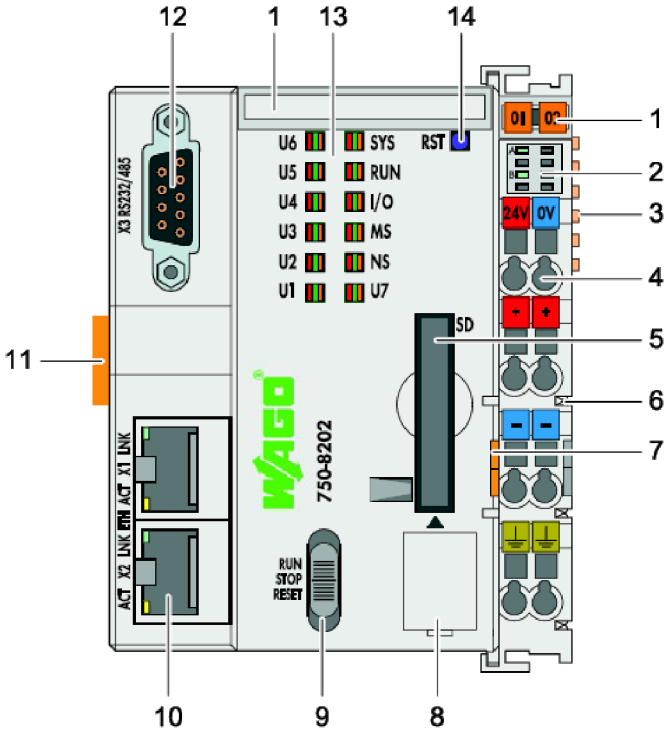
1.1 Zoznámenie s PLC PFC200

Hlavnou výhodou procesorových modulov PFC200 je, že si vzali niečo z priemyselných PC a niečo z klasických procesorových modulov. Vďaka tomu sú tieto moduly univerzálne a použiteľné takmer vo všetkých priemyselných aplikáciách. S modulom je možné pracovať v klasickom PLC prostredí ale aj s využitím vyšších programovacích jazykov. Procesorový modul s Linuxom je možné rozšíriť použitím rôznych balíčkov a zostaviť si tak operačný systém na programovanie embedded aplikácií podľa vlastných požiadaviek. Vďaka tomuto môžeme využívať rôzne programovacie jazyky.

Obr. 1.1: Ukážka procesorového modulu PFC200-8202.[3]

1.1.1 Základný popis procesorového modulu

1. Značenie vývodov (Mini-WSB).
2. LED indikátory - napájanie.
3. Dátové kontakty na pripojenie modulov.
4. Pripojenie napájania pomocou kontaktov "CAGE CLAMP".
5. Slot pre pamäťovú kartu.
6. Napájacie výkonové kontakty pre napájanie I/O modulov.
7. Úchyt na uvoľnenie prídavného modulu.
8. Servisný konektor (Schovaný za klapkou).



Obr. 1.2: Popis procesorového modulu.[3]

9. Prepínač na vyberanie pracovného módu.
10. Konektory pre ETHERNET.
11. Úchyt na pripevnenie modulu na montážnu lištu.
12. Sériové rozhranie.
13. Systémové LED indikátory.
14. Reset tlačidlo (Schované v otvore).

Popis vychádza z literatúry [3].

1.1.2 Komunikácia s modulom

Moduly patria medzi kompaktné riadiace jednotky, ktoré okrem klasických úloh dokážu tiež zaistiť komunikáciu s podriadenými a nadriadenými systémami. V moduloch sa využíva procesor Cortex A8 taktovaný na frekvenciu 600MHz, ktorý dokáže komunikovať so všetkými riadiacimi jednotkami a modulmi zo systému WAGO-I/O-SYSTEM 750.

WAGO-I/O-SYSTEM 750 vďaka svojej modularite a nezávislosti na type priemyselnej zbernice dokáže splniť všetky požiadavky na decentralizované systémy priemyselných zberníc. V závislosti na prípade použitia môžeme teda voliť zo širokého výberu komunikačných a procesorových modulov.

Viac modulov je možné medzi sebou prepojiť pomocou siete ethernet a súčasne môžeme využiť možnosti ďalších rozhraní, ktoré sú k dispozícií. Taktiež môžeme prepojiť tieto jednotky s inými PLC s rozhraním ethernet. Využitím protokolu MODBUS TCP je možná komunikácia aj s riadiacimi jednotkami iných značiek, prípadne iných zariadení.

Na jeden procesorový modul je možné pripojiť maximálne 64 I/O (vstupno/výstupných) modulov, prípadne 250 I/O modulov, ak využijeme rozdelenie vnútornej zbernice K-Bus do viacerých častí.

1.1.3 Komunikačné rozhrania

Rada procesorových modulov PFC200 je vybavená dvoma ethernetovými portmi, ktoré sú základom v komunikácii. Podporujú širokú škálu protokolov, vrátane telemetrie.

Typy procesorových modulov a ich komunikačné rozhrania [3]:

1. Typ 750-8202 má rozhranie ethernet s dvoma portami a jedno konfigurovateľné sériové rozhranie (RS 232 / RS 485).
2. Typ 750-8203 má rozhranie ethernet s dvoma portami a jedno rozhranie CAN konfigurovateľné ako master/slave.
3. Typ 750-8204 má rozhranie ethernet s dvoma portami, jedno konfigurovateľné sériové rozhranie (RS 232/RS 485) a rozhranie CAN konfigurovateľné ako master/slave.
4. Typ 750-8206 má rozhranie ethernet s dvoma portami, jedno konfigurovateľné sériové rozhranie (RS 232/RS 485), rozhranie CAN konfigurovateľné ako master/slave a rozhranie PROFIBUS DP slave.

1.1.4 Pamäť

Velkosť hlavnej pamäte RAM v module je 256MB, vnútorná flash pamäť má tiež 256MB, retain pamäť 128kB. Pamäť je možné rozšíriť použitím pamäťovej SDHC karty do veľkosti až 32GB.

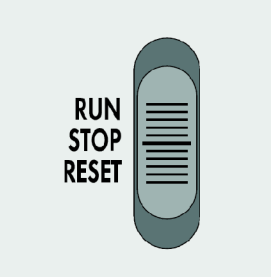
Časť pamäte RAM využíva CoDeSys (16 MB pre program, 64 MB pre dáta, 2 kB pre obraz vstupov, 2 kB pre obraz výstupov, 2 kB pre MODBUS, 244 B pre PROFIBUS a 4 kB pre CAN). Zvyšok pamäte je určený pre operačný systém LINUX a aplikácie fungujúce pod ním. [4]

1.1.5 Ovládanie modulu

Prepínač režimu prevádzky modulu

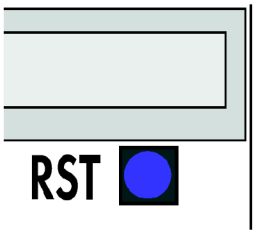
Procesorový modul PFC200 má tri režimy prevádzky: [3]

1. RUN - Normálna prevádzka. Aplikácie, ktoré sú nahrané v pamäti modulu sa vykonávajú. Táto poloha je aretovaná.
2. STOP - Modul nepracuje. Všetky aplikácie sú zastavené. Táto poloha prepínaču je tiež aretovaná.
3. RESET - Táto poloha prepínaču má rôzne funkcie v závislosti na tom ako dlho je prepínač v tejto polohe. Môže to byť Reset warm start alebo Reset cold start. Táto poloha prepínaču nieje aretovaná ale vždy po pustení prepínaču sa prepínač vráti do polohy STOP.



Obr. 1.3: Prepínač prevádzkového režimu.[3]

Reset tlačidlo



Obr. 1.4: Tlačidlo reset.[3]

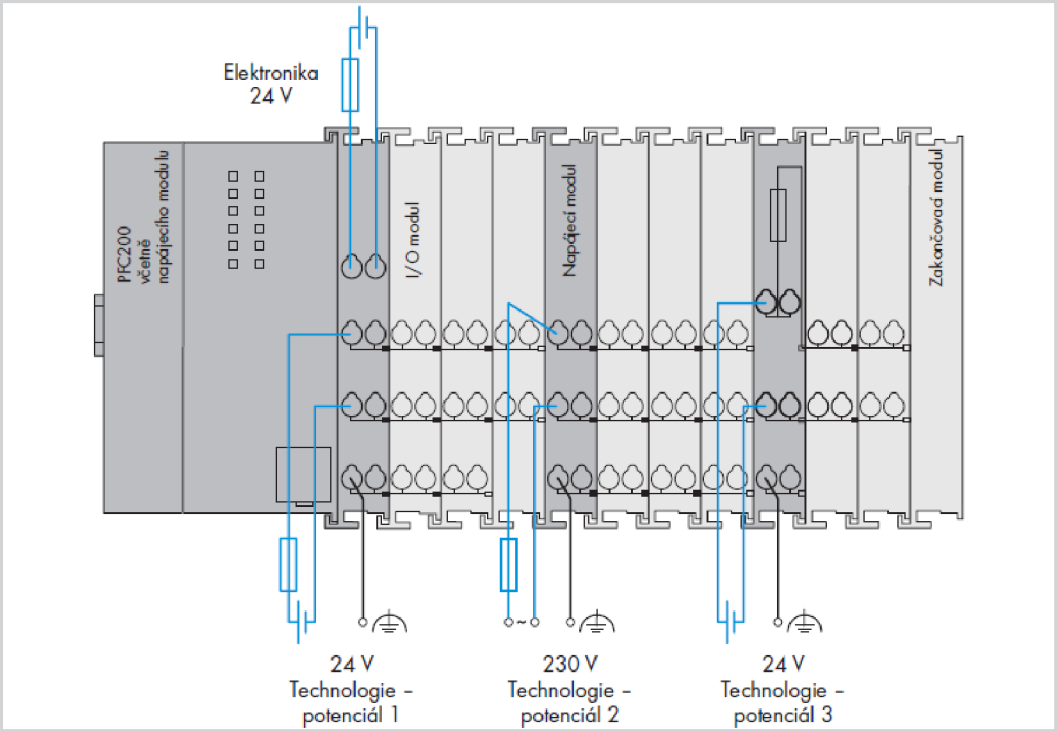
Tlačidlo je schované za otvorom aby sa predišlo nechcenému stlačeniu pri obsluhe prístroja. Tlačidlo je možné ovládať pomocou vhodného predmetu (napríklad pero). Toto tlačidlo má rôzne funkcie v závislosti na polohe prepínaču režimov. Môžeme s ním napríklad: [3]

- Dočasne nastaviť pevnú IP adresu.

- Vykonať softvérový reset (reštart modulu).
- Obnoviť továrenské nastavenia.

1.1.6 Napájanie modulu

Na napájanie procesorového modulu sa používa napájací zdroj 24V DC (s maximálnou povolenou odchýlkou napätia -25% až +30%) musí mať kapacitu min. 550 mA, čo je maximálna spotreba prúdu, ktorú dokáže procesorový modul vyvinúť. Kapacita interného zdroja napájania zbernice K-Bus je 1700 mA. Jednotky sú vybavené napájaním silových pomocných zberníc vstupno/výstupných modulov. Zbernica je dimenzovaná pre napätie 24 V DC a maximálne prúdové zaťaženie 10 A. Toto napájanie je oddelené od systémového napájania, ale nie je zakázané tieto napájacie body navzájom spojiť a potom použiť len jeden napájací zdroj.



Obr. 1.5: Hierarchia napájania jednotlivých modulov.[1]

1.1.7 ADI/DAL rozhranie pre procesorový modul PFC200

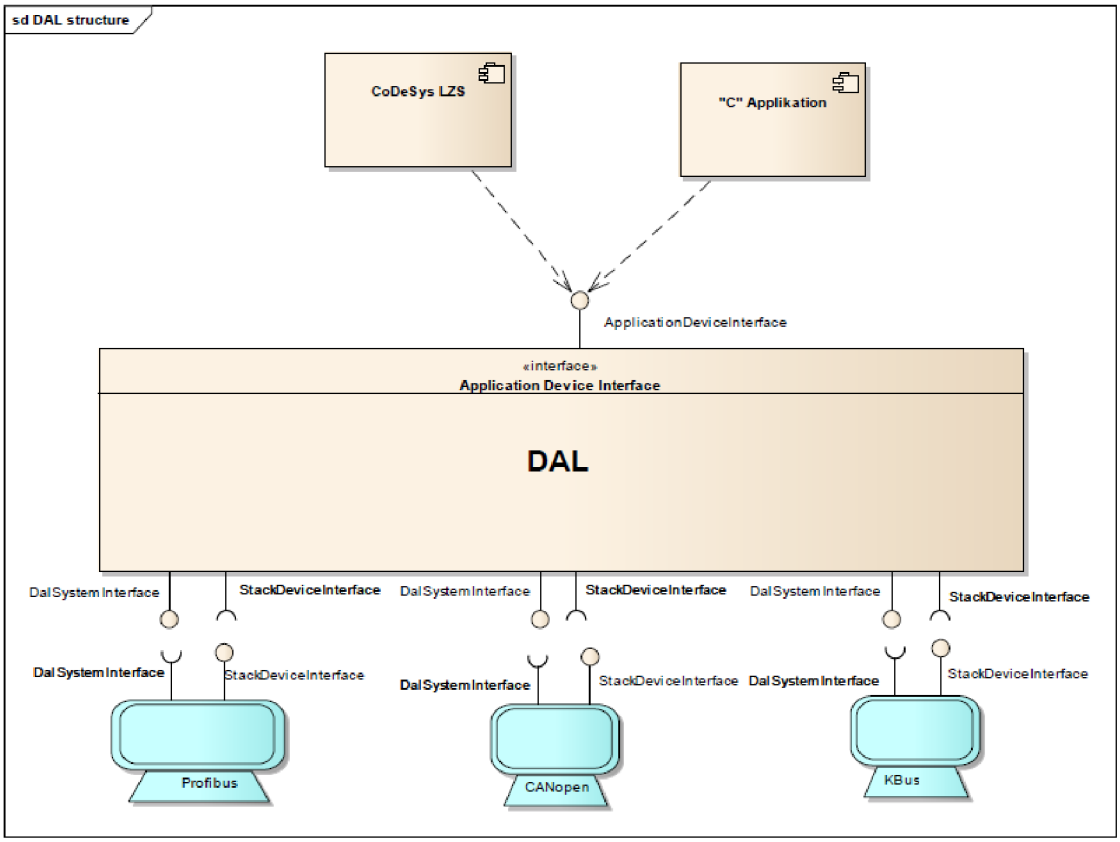
Hlavnou úlohou DAL $Device - Abstraction - Layer$ a ADI $Application - Device - Interface$ je:

- Zjednotenie prístupu k rôznym zberniciam zariadenia.

- Otvorenie rozhraní pre programovanie aplikácií bez autorských práv alebo licenčných obmedzení.
- Zjednodušenie CoDeSys-I/O-Driver rozhrania, prenesie iba funkcie, ktoré požadujeme.

ADI/DAL v súčasnosti podporuje abstrakciu a jednotný prístup k nasledujúcim normám:

- PROFIBUS (Slave).
- CANopen.
- Modbus TCP a ModbusRTU.
- KBus.



Obr. 1.6: ADI/DAL štruktúra.[8]

ADI rozhranie môže byť použité z CoDeSys-Runtime alebo C/C++ aplikácie ale nikdy nie v rovnakom čase. ADI definuje povinnú sadu funkcií, ktorú každé zariadenie musí poskytovať. Tiež poskytuje proxy mechanizmus na podporu zariadení so špecifickými funkciami.
Hlavnou úlohou rozhrania DAL je:

- Administrácia I/O (vstupno/výstupných) knižníc ovládačov (zoznam, otvorenie, zatvorenie).
- Funkcia presmerovania volania na jednotlivé I/O ovládače.
- Prenosová funkcia pre všetky I/O ovládače

Pred zavedením ADI/DAL rozhrania komunikovali všetky zariadenia priamo s PLC Runtime, toto pri použití prístrojov v rámci inej aplikácie robilo komunikáciu ťažkú až nemožnú.

ADI/DAL pôsobí ako administratívna jednotka medzi zariadeniami a akoukoľvek aplikáciou, ktorá zariadenia využíva.

V prípade PLC-Runtime (RTS), ako CoDeSys2.3 alebo e!Cockpit sú pôvodné RTS rozšírené o IO-Driver-Interface, ktorý konvertuje 3S-Softwarové špecifické požiadavky do ADI/DAL žiadosti a naopak. Zariadenie by nikdy nedokázalo priamo použiť RTS funkciu, je nutné použiť ADI/DAL, ktoré upravuje alebo transformuje požiadavky.

Viac o rozhraní je popísané v dokumentácii [8].

1.1.8 Programovanie modulu

Jednotka PFC200 používa operačný systém LINUX 3.6 s podporou prevádzky real time. Konfigurácia modulu je možná pomocou integrovaného webového serveru a webového administračného rozhrania, pomocou programovacieho prostredia CoDeSys 2.3 podľa IEC 61131-3, prípadne v novo vyvinutom prostredí WAGO e!COCKPIT, na báze CoDeSys 3 a taktiež pomocou konfiguračného dialógu priamo v konzole systému Linux. Vďaka tomu nepotrebujeme špecifické znalosti rôznych operačných systémov.

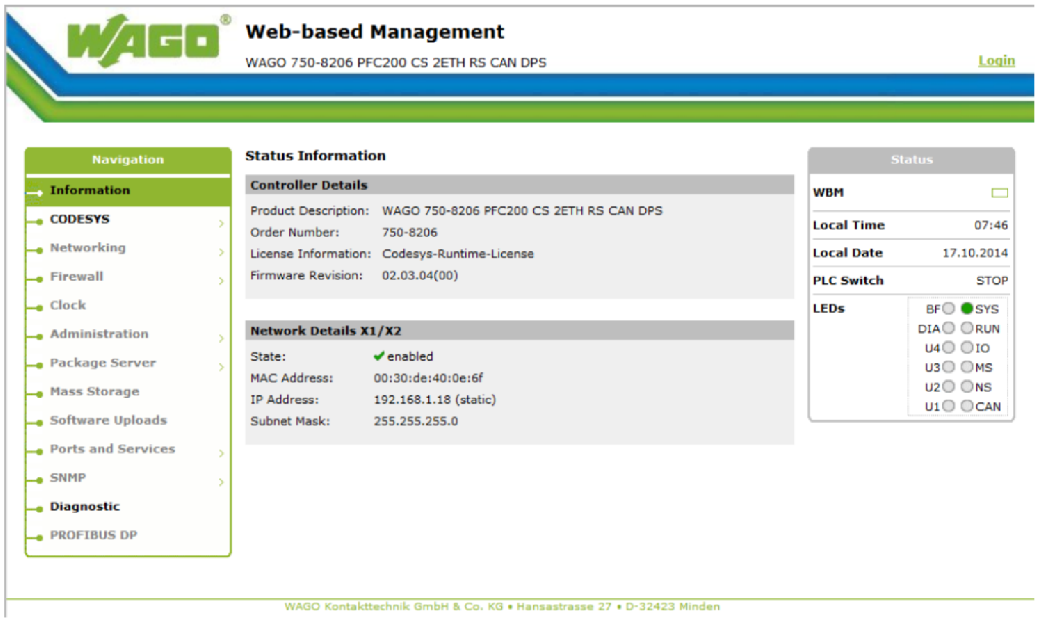
WBM - WEB-based Management

Existencia WEB servera a ethernetového rozhrania v procesorovom module nám umožňuje naprogramovanie, parametrizáciu, prípadne riadenie počas prevádzky na diaľku.

Pomocou WBM je napríklad možné nastavovať typ runtime systému, ktorý má bežať na PLC, nahrávať aktualizácie softwaru, prekopírovať software z SD karty do internej pamäte, konfigurovať možnosti pripojenia k PLC, sledovať stav pomocou Led indikátorov, a podobne.

Podrobný popis WBM serveru:

Názov zariadenia je zobrazený v hlavičke okna na obrázku 1.7. Keď je užívateľ odhlásený v pravom hornom rohu je zobrazené *Login* a keď prihlásený *Logout* tlačidlo.



Obr. 1.7: Ukážka webového serveru pre prácu s modulom.

Navigačný panel je v ľavej časti okna. Niektoré stránky z tohto okna sa dajú použiť až po príhlasení do zariadenia, ktoré je možné vykonať pomocou tlačidla *Login*.
Panel na pravej strane zobrazuje:

- WBM status: Tento indikátor zobrazuje či WBM komunikuje so zariadením na pozadí. Inými slovami, zobrazuje sa tu či WBM poslal požiadavku na zariadenie a čaká od neho odpoveď.
- Local Time: Čas v zariadení.
- Local Date: Dátum v zariadení.
- PLC Switch: Zobrazuje v akej polohe je prepínač režimov.
- LEDs: Indikátory zobrazujú stav lediek na zariadení. Šedá farba znamená, že ledka je vypnutá. Plná farba znamená, že ledka svieti danou farbou. Polovičná farba znamená, že ledka bliká danou farbou.

Programovanie cez e!COCKPIT

Software e!COCKPIT je komplexný nástroj pre návrh, konfiguráciu, naprogramovanie a tiež pre diagnostiku celého riadiaceho systému. Software je projektovo orientovaný, umožňuje spravovať naraz niekoľko samostatných riadiacich modulov navzájom prepojených v sieti.

Prehľadným grafickým spôsobom rieši konfiguráciu a parametrizáciu komunikačnej siete, hardvérovú konfiguráciu každého riadiaceho systému, následné vytvorenie

riadiaceho programu pre jednotlivé moduly a tiež prevádzku a diagnostiku celého riadiaceho systému.

Vizualizačný editor obsiahnutý v tomto programe má možnosť priameho prístupu k programovým premenným. Podporuje jazyky HTML 5 alebo CSS.

Činnosť procesorovej jednotky pri neštandardných stavoch si môžeme určiť v prostredí nastavením výstupov do troch možných definovaných stavov: všetky výstupy = 0, všetky výstupy = 1, alebo všetky výstupy ostanú v stave v akom boli pred vznikom neštandardného stavu. Najčastejší zdroj vzniku neštandardnej situácie je strata napájacieho napätia.

Vlastné programovanie sa vykonáva v prostredí CoDeSys 3 podľa protokolu IEC 61131-3.

Ponúka možnosť spracovania programu vo viacerých jazykoch:

1. ST (Structured Text) je jednoduchý na programovanie v jazyku Pascal alebo C.

Výpis 1.1: Jednoduchý príklad programu v jazyku ST.

```
TxtState := STATES[StateMachine];

CASE StateMachine OF
   1: ClosingValve();
      StateMachine := 2;
   2: OpeningValve();
ELSE
    BadCase();
END_CASE;
```

2. LD (Ladder Diagram) umožňuje programátorovi prakticky kombinovať reléové kontakty a cievky.

Výpis 1.2: Jednoduchý príklad programu v jazyku LD.

```
Logical AND
 ------[ ]--------------[ ]----------------( )
   Key Switch 1       Key Switch 2        Door Motor

Logical OR
 --+-------[ ]-------+-----------------( )
   | Exterior Unlock |                  Unlock
   |                 |
   +-------[ ]-------+
     Interior Unlock
```

3. FBD (Function Block Diagram) umožňuje užívateľovi rýchlo programovať lo-

gické aj analógové výrazy.

Výpis 1.3: Jednoduchý príklad programu v jazyku FBD.

```
Comparator                                                    1
     +----------+                                             2
     |  CMP>R   |                                             3
IN  -| IN1      |     +----------+                            4
     |          |     |   MOVE   |                            5
MAX -| IN2      |-----| EN       |                            6
     +----------+     |          |- MAX                       7
                 IN -| IN       |                            8
                     |          |                            9
                     +----------+                            10
```

4. IL (Instruction List) je ako programovací jazyk assembler.

Výpis 1.4: Jednoduchý príklad programu v jazyku IL.

```
          LD      Speed                                       1
          GT      2000                                        2
                                                              3
          JMPCN   VOLTS_OK                                    4
          LD      Volts                                       5
VOLTS_OK  LD      1                                           6
          ST      %Q75                                        7
```

5. SFC (Sequential Function Chart) je vhodný pre programovanie sekvenčných procesov a tokov.

Výpis 1.5: Jednoduchý príklad programu v jazyku SFC.

```
     |                                                    1
-----+---Start condition                                  2
     |                                                    3
+----------+                                              4
| Step 1   |          +-+----------------------+-+        5
| Extend   |----------| | Energise Solenoid A | |        6
| Piston A |          +-+----------------------+-+        7
+----------+                                              8
     |                                                    9
-----+---Piston A Extended                                10
     |                                                    11
+----------+                                              12
| Step 2   |          +-+----------------------+-+        13
| Extend   |----------| | Energise Solenoid B | |        14
| Piston B |          +-+----------------------+-+        15
+----------+                                              16
     |                                                    17
-----+---Piston B Extended                                18
     |                                                    19
```

6. CFC (Continuous Function Chart) je akousi úpravou FBD editora. Rozdielne ako v sieťovo orientovanom FBD editore kde sú prepojenia medzi vstupmi, operátormi a výstupmi nastavené automaticky je tu možnosť ich vypracovania programátorom. Všetky boxy možno ľubovoľne umiestniť, čo umožňuje naprogramovanie spätnej väzby bez priebežných premenných. Tento jazyk nieje definovaný v protokole IEC.

Programovanie embedded-Linux aplikácií

Využívaním tejto možnosti programovania máme plný prístup k systému procesorového modulu. Môžeme si jeho operačný systém ľubovoľne upravovať, konfigurovať a vytvárať v ňom vlastné aplikácie pomocou rôznych balíčkov, ktoré máme k dispozícii. Oproti programu e!COCKPIT je v tomto prípade obmedzená podpora.

Pre programovanie je výhodné využiť predpripravené programové balíčky od výrobcu, kde je program vysvetlený aj s postupom nahrávania do modulu. Balíčky je možné nájsť priamo na stránkach výrobcu, napríklad: [5].

Možnosti spolupáce s embedded prostredím

1. Spustenie Shell-Skriptu v rámci e!COCKPIT prostredia. Toto je možné pomocou nasledujúceho postupu:

(a) Napíšeme svoj vlastný shell skript.
(b) Skopírujeme tento skript do zložky PFC modulu "/etc/config-tools/".
(c) Skontrolujeme či je tento skript spustiteľný (chmod +x).
(d) Napíšeme e!COKCPIT program pomocou "WagoAppConfigTool.lib".
(e) Spustíme PLC program.

2. Vytvoriť externú e!COKCPIT knižnicu kódovanú v jazyku C. Knižnicu je možné vytvoriť nasledovne:
   (a) Otvoríme si nový projekt v e!COCKPIT.
   (b) Definujeme si rozhranie, cez ktoré to chceme realizovať.
   (c) Súbor uložíme ako externú knižnicu, z toho nám vzniknú súbory: libExternal.lib, libExternal.c, libExternal.h.
   (d) Pre tieto súbory napíšeme implementáciu.
   (e) Pridáme „WAGO-wide“ rozhranie.
   (f) Skompilujeme kód ako zdieľaný objekt. Vznikne nám súbor libExternal.so.
   (g) Skopírujeme zdieľaný objekt do súboru PFC modulu "/usr/lib/wide/".
   (h) Následne reštartujeme PFC modul.
   (i) Napíšeme PLC program s využitím „libExternal.lib".
   (j) Spustíme PLC program.
3. Rozšíriť „WAGO-Basicimage“ s dodatočnými „Open-Source-Packages“.
   (a) Vyberieme si balíček s ktorým budeme pracovať v položke "PTXdist menuconfig".
   (b) Následne skompilujeme všetky báličky pomocou nástroju PTXdist.
   (c) Vytvorí sa nám image "hd.img"v položke "PTXdist images".
   (d) Nahráme "hd.img"na SD-kartu napríklad pomocou programu DiskImager.
   (e) Vložíme kartu SD do modulu PFC200.
   (f) Reštartujeme PFC200.
4. Napísať svoju vlastnú C/C++ aplikáciu využívajúcu KBUS, CAN a/alebo PROFIBUS.
   (a) Vytvoríme nový PTXdist balík.
   (b) Upravíme si súbory názov_balíku.in, názov_balíku.make.
   (c) Vyberieme si balík na vytvorenie.
   (d) Vytvoríme si projekt ECLIPSE-CDT, cez ktorý môžeme kód debugovať.
   (e) Nakonfigurujeme si modul PFC ako "remote system".
   (f) Pridáme DAL rozhranie (KBUS, CAN, ...) kvôli tomu aby sme mohli pracovať so vstupmi a výstupmi PLC.
   (g) Editujeme zdrojové súbory a makefile, pre ukážku môžeme použiť súbory dodávané výrobcom s názvom WAGO-PFC-BSP-howtos.

(h) Vzdialene odladíme svoju aplikáciu.

5. Prípadne môžeme použiť pokročilé ladiace mechanizmy.

1.2 Meranie výkonu - Benchmarking

V tejto kapitole je popísané meranie výkonu podľa certifikátu TC3. Meranie výkonu na PLC sa začalo v roku 2005 s cieľom vytvoriť klasifikačný štandard pre výkonnosť rôznych PLC. Viac o meraní je možné zistiť v literatúre [9] a [10].
Cieľom merania výkonu je:

- Odhad výkonu aplikácií na konkrétnej platforme.
- Objektívne porovnanie výkonu na rôznych platformách automatov.

Meranie výkonu pozostáva z dvoch častí. Prvá časť, nazývaná **Aplikačne orientované meranie výkonu**, definuje 5 typických PLC aplikácií:

1. Spracovanie digitálnych vstupov a výstupov.
2. Spracovanie sekvencií s digitálnymi vstupmi a výstupmi v SFC editore.
3. Ovládanie pohybu (využíva sa pri ovládaní viac osích zariadení, tlačiarní).
4. Spracovanie dát (aplikácie, ktoré pracujú s dátami).
5. Spätnoväzobné aplikácie (aplikácie s rôznymi regulátormi).

Druhá časť sa nazýva **Jazykovo orientované meranie výkonu**, je navrhnutá tak aby sa uľahčilo meranie jednotlivých jazykov protokolu IEC 61131-3. Cieľom tohto merania je zabrániť nežiadúcim efektom, ako sú napríklad časy prerušenia alebo prístup k vstupom alebo výstupom, čo najviac ako to je možné. Testované funkcie sú:

- Operácie so štandardnými typmi dát: BOOL, WORD, INT, REAL, STRING.
- Variabilný prístup: lokálne a globálne premenné, vstupné a výstupné premenné, priame premenné, konfiguračné premenné.
- Inštrukcie v ST editore: IF, FOR, CASE, WHILE, REPEAT.
- Volanie blokov: volanie funkcií a funkčných blokov.
- SFC editor: inicializácia, kroky, prechody a rozbočky.
- Štandardná knižnica: časovač, čítač, spúšťač.

1.2.1 Realizácia

Osobitný dôraz pri meraní sa kladie na požiadavky, ktoré majú zaručiť flexibilné a predovšetkým použiteľné meranie výkonu. Napríklad:

- Meranie musí obsahovať jasnú definíciu toho čo bolo merané a ako to bolo merané.
- Merací prístroj musí byť jasne definovaný aby bolo možné porovnávať výsledky meraní.

- Tiež je dôležité definovať meracie metódy, ktoré boli použité.

1.2.2 Obecné požiadavky na meranie

- Doba cyklu sa meria v mikrosekundách aby bolo meranie dostatočne presné.
- Skúška by mala byť vykonaná minimálne 10000 krát z čoho sa vypočíta priemerný čas, je to kvôli tomu aby sa eliminovali časy prerušenia, ktoré sú spôsobené operačným systémom.
- Do výsledku je potrebné napísať minimálnu, maximálnu a priemernú dobu cyklu.
- Popis výsledkov musí tiež obsahovať podmienky merania. Medzi ne patrí kompilátor, ktorý sa používal a hardware PLC, t.j. pamäť, procesor a operačný systém PLC.

1.2.3 Meranie času

Namerané cykly sú založené na systémovom čase PLC kde si definujeme začiatok a koniec merania času. Po vykonaní aplikácie vypočítame rozdiel týchto časov a to je doba cyklu. Táto meracia metóda samozrejme zahŕňa aj prerušenia operačného systému, ktoré môžu byť len ťažko vylúčené a samozrejme majú vplyv na meraný čas.

1.2.4 Aplikačne orientované meranie

Táto meracia kategória v sebe zahŕňa 5 typov aplikácií, ktoré sú popísané vyššie. Väčšine aplikácií, bežne používaných v automatoch je možné priradiť niektorý z týchto typov, prípadne sú ich kombináciou. Typické testovanie výkonu je definované pre všetky typy týchto aplikácií. Musia spĺňať určité požiadavky:

- Pokiaľ ide o prístup k vstupom a výstupom a veľkosť aplikácie, mala by zodpovedať reálnej PLC aplikácii.
- Testy musia byť škálovateľné tak, aby bolo možné vyhodnotiť výkon dvakrát väčšej alebo polovične veľkej aplikácie.
- Meranie výkonu má byť implementované v IEC editore ST.

Charakteristické pre toto meranie je veľa zapisovania a čítania z variabilných polí. Aplikácie obsahujú rôzne typy inštrukcií: FOR, IF, CASE, WHILE, ktoré sú naimplementované v štyroch funkčných blokoch a v jednej funkcii. Keď je spustený program, bloky sú volané a sú vykonávané rôzne inštrukcie v závislosti na podmienkach. Medzi týmito operáciami je zapisovanie do polí, porovnávanie, sčítanie a násobenie. Zložitosť skutočnej aplikácie je simulovaná vnorenými slučkami FOR, ktoré obsahujú ďalšie inštrukcie.

1.2.5 Jazykovo orientované meranie

Tieto testovacie projekty musia mať určitú veľkosť aby sa pri meraní zabránilo nameraniu nesprávnych výsledkov, kvôli rýchlejšiemu zapisovaniu do vyrovnávacej pamäte. Špecifické operácie pre testovanie, ako napríklad štartovanie úlohy, ktorá obsahuje test výkonu, nesmie byť súčasťou merania času. Napriek tomu úloha prepínania časov musí byť uvažovaná v meraní. Táto kategória merania zahŕňa testy, ktoré umožňujú detekciu špecifických PLC výhod a nevýhod spôsobených rôznymi konštrukciami normy IEC 61131-3.

Projekt pre meranie výkonu obsahuje napríklad niekoľko blokov, v ktorých sú vykonávané logické operácie a algebraické výpočty s dátovými typmi normy IEC.

Každý blok obsahuje dátový typ a vykonáva určitú operáciu. Používané dátové typy sú BOOL, WORD, INT, REAL a STRING. Celkom 10000 súčtov, odčítaní a rovnaký počet AND a OR operácií sa vykonáva s týmito dátovými typmi - v prípade, že je operácia podporovaná. Ďalších 1000 operácií násobenia a delenia sa vykonáva pre každý typ dát. Konverzia dát nieje v tomto prípade povolená.

1.2.6 Zdokumentovanie výsledkov merania

Výsledky merania výkonu sa dokumentujú na osobitnom formulári. Tento formulár musí obsahovať nasledujúce údaje:

- Výrobca a typ PLC.
- Programovacie prostredie a jeho verzia.
- Operačný systém.
- Charakteristika hardvéru - procesor, veľkosť RAM, veľkosť vyrovnávacej pamäte.
- Meno osoby, ktorá vykonávala meranie.
- Dátum testu.

Tieto údaje uľahčujú porovnávanie odlišných PLC platform. Všetky kritériá spolu s hodnotami pre maximálnu, minimálnu a priemernú dobu cyklu sú zaznamenané v tabuľke s výsledkami. Je tu zaznamenaná aj veľkosť vytvoreného kódu.

PLC Benchmarking Result

Smart Software Solutions

PowerPC MPC5200			CoDeSys 3.0 SP2 & 2.3	
CPU MHz:	Memory:	Cache:	Board:	Operating system:
396 MHz	64 MB	16 MB	phyCORE-MPC5200B tiny	Linux V. 2.6.17-mpc5200-1
Tested by:	3S GmbH		Test date:	9.11.2006

Results Table

	CoDeSys 3.0				CoDeSys 2.3			
Execution time [µs] / Benchmark	Min.	Max.	Average	Size [bytes]				
Application oriented								
Processing of digital I/Os	30	96	**30**	55.889	39	236	**40**	60.692
SFC and processing of digital I/Os	89	174	**96**	84.775	91	304	**101**	64.208
Motion Control Application	-	-	-	-	-	-	-	-
Data Processing Application	261	402	**263**	70.659	502	698	**504**	69.284
Closed Loop Control Application	68	147	**72**	49.796	41	239	**42**	40.524
Language oriented								
SFC	67	151	**72**	97.068	91	296	**95**	48.068
POU calls								
Function block	146	261	**146**	45.837	199	403	**200**	37.388
Function	143	258	**143**	45.615	164	331	**164**	36.316
ST Control Statements								
CASE	128	196	**128**	47.899	257	462	**258**	77.688
FOR	214	283	**214**	52.327	337	544	**338**	77.608
IF	109	248	**110**	60.535	239	455	**239**	73.524
WHILE	147	278	**148**	60.615	271	462	**272**	77.516
REPEAT	1268	1384	**1270**	61.966	2059	2299	**2063**	82.936
Standard Library								
Trigger	154	223	**154**	49.495	137	337	**137**	67.228
Counter	371	531	**372**	55.582	346	550	**347**	138.432
Timer	260	382	**261**	52.534	241	436	**242**	81.612
Flip Flop	168	236	**168**	49.510	148	353	**149**	69.896
Operations with standard Data Types								
BOOL	1322	1499	**1325**	88.173	1324	1578	**1327**	607.964
INT	1458	1637	**1461**	95.688	1458	1719	**1462**	655.208
DINT	1458	1635	**1462**	95.096	1457	1678	**1460**	719.804
WORD	1458	1629	**1461**	95.712	1457	1689	**1462**	655.196
DWORD	1458	1638	**1461**	95.723	1457	1763	**1460**	721.244
REAL	1464	1667	**1467**	95.175	1459	1718	**1464**	654.028
LREAL	1496	1669	**1498**	95.182	1499	1748	**1502**	720.036
SRTING	-	-	-	-	-	-	-	-
Variable Access								
VAR_GLOBAL	64	136	**71**	57.200	65	266	**71**	76.184
VAR_LOCAL	63	137	**71**	56.121	64	230	**69**	83.572
VAR_IN_OUT	123	192	**124**	58.379	121	314	**122**	92.736
VAR_AT	66	138	**70**	58.062	71	259	**72**	84.256
VAR_CONFIG (AT)	-	-	-	-	-	-	-	-

Obr. 1.8: Ukážka formuláru s nameranými hodnotami.[9]

2 PRAKTICKÁ ČASŤ PRÁCE

2.1 Meranie výkonu

Meranie výkonu sa pochopiteľne delí na dve časti, ktoré medzi sebou porovnávame. Dôležité pri meraní bolo aby boli obe merané časti v rámci možností identické. Teda aby bol dodržaný počet a typ operácií, ktoré boli v rôznych meraniach vykonávané. Prvá časť merania je v programovom prostredí e!COCKPIT písaná jazykom ST. Druhá časť je vytváraná pomocou balíčkovacieho nástroju PTXdist a kód je písaný v jazyku C a C++. Tieto jazyky boli zvolené preto, lebo sú zo všetkých najviac identické a tým pádom môžeme s najväčšou presnosťou určiť, ktoré prostredie v čom vyniká. Obe prostredia samozrejme ponúkajú programovanie aplikácií aj v iných jazykoch. Najväčší rozdiel medzi týmito programovacími jazykmi bol v type a veľkosti premenných, ktoré používajú, preto bolo nutné vyberať premenné, ktoré sú rovnakého charakteru a zaberajú rovnaké miesto v pamäti. Zasa to je kvôli tomu aby sme mohli namerané výsledky porovnávať.

Meranie bolo v oboch prípadoch založené na rovnakom princípe, ktorý popisuje certifikát TC3. Údaje o certifikáte sú získané z literatúry [10]. Tento certifikát je však stavaný pre meranie výkonu v prostrediach ako je napríklad e!COCKPIT, CODESYS a podobne, preto muselo byť meranie prispôsobené tak, aby sme mohli porovnávať dva odlišné prostredia. Meraný program musel byť dosť zložitý aby sa vyrovnal skutočnej aplikácii. Meranie muselo prebehnúť aspoň 10000 krát z čoho sa potom určil maximálny, minimálny a priemerný čas, za ktorý sa meraný program vykonal. Spracovávanie časov nemohlo byť zahrnuté do merania, preto sa časy spracovávali až po odčítaní koncového času. Princíp merania je popísaný na obrázku A.1 pomocou diagramu v prílohách k meraniu.

Meranie bolo v každom prostredí rozdelené na dve hlavné časti. Bolo to aplikačne a jazykovo orientované meranie. **Aplikačne orientované meranie** sa zameriava na meranie vykonania času rôznych typov aplikácií, ktoré sa používajú v reálnych podmienkach. Z týchto typov aplikácií som vybral dve, ktoré je možné naprogramovať v oboch prostrediach. Sú to nasledovné:

- Spracovanie vstupov a výstupov PLC.
  Toto meranie musí obsahovať nasledujúce položky:
  - pracuje sa tu s 5 vstupmi a 5 výstupmi, ktoré sa postupne znásobujú vnorenými slučkami FOR až po 500 vstupov a 500 výstupov.
  - program je rozdelený do 4 funkčných blokov a 1 funkcie. Tieto bloky sú volané z hlavného bloku.
  - 15 premenných na ukladanie medzivýsledkov.

  - 55 AND, 35 OR, 50 NOT operácií a 75 priradení.
- Spracovanie dát.

  Toto meranie musí obsahovať nasledujúce položky:
  - program je rozdelený do 4 funkčných blokov a 1 funkcie. Tieto bloky sú volané z hlavného bloku.
  - 15 polí - 5 jednorozmerných, 5 dvojrozmerných a 5 trojrozmerných.
  - datové typy BOOL, INT/int16_t, WORD/uint16_t, REAL/float a STRING(1)/char.
  - 30 vnorených FOR pracujúcich s poliami.
  - 30 CASE - 15 s 5 možnosťami a 15 s 25 možnosťami.
  - 15 IF-THEN-ELSE a 15 WHILE-DO slučiek.

Jazykovo orientované meranie sa potom orientuje na jednotlivé časti programovacieho prostredia a meria časy každej časti zvlášť. Medzi tieto merania patria:

- Operácie s jednotlivými typmi dát. S týmito typmi sa vykoná 10000 krát operáci AND/OR, prípadne plus a mínus.
  - BOOL - používa sa v oboch prostrediach.
  - INT/int16_t - e!COCKPIT používa 2 bytový int, preto bol použitý aj v embedded prostredí.
  - WORD/uint16_t - tieto premenné sú ekvivalentné, preto boli použité.
  - REAL/float - takisto ekvivalentné premenné.
  - STRING(1)/char - e!COCKPIT používa dátový typ STRING, preto bol použitý a obmedzený na veľkosť jedného bytu čím sa vyrovnal premennej char v C++ programe.
- Operácie s odvodenými typmi dát.
  - Polia - 3 polia s dvadsiatimi prvkami typu int. 100 krát zápis do každého prvku poľa.
  - Štruktúry - 3 štruktúry s dvadsiatimi prvkami typu int. 100 krát zápis do každého prvku štruktúry.
- Prístup k premenným.
  - Lokálne premenné - 5 polí s tristo prvkami. Každé pole iného dátového typu (vyššie uvedené). Meria sa priradenie do všetkých prvkov polí.
  - Globálne premenné - rovnaké meranie ako pri lokálnych premenných akurát polia sú globálne.
- Funkcie na ovládanie toku programu.
  - IF - 1000 vykonaní čo najjednoduchšej IF funkcie.
  - CASE - 1000 vykonaní čo najjednoduchšej CASE funkcie.
  - FOR - 1000 vykonaní čo najjednoduchšej FOR funkcie.
  - WHILE - 1000 vykonaní čo najjednoduchšej WHILE funkcie.
- Volanie programových blokov.

– Funkcia - 1000 zavolaní prázdnej funkcie s 5 vstupmi a 1 výstupom.
– Funkčný blok - 1000 zavolaní prázdneho funkčného bloku s 5 vstupmi a 5 výstupmi.

2.1.1 Meranie v prostredí e!COCKPIT

Pred prácou v prostredí e!COCKPIT je dôležité najprv nastaviť runtime, ktorý tomuto programu prislúcha. To je možné pomocou Web based Management v záložke *Networking - General Configuration*. Tu je potrebné nastaviť *eRuntime* a potvrdiť nastavenie.

Prevedenie merania

Odčítavanie počiatočného a koncového času bolo v tomto prostredí riešené pomocou funkčného bloku z knižnice *WagoAppTime*, verzia 1.7.2.3. Funkčný blok sa nachádza v zložke *Program Organization Units* a následne zložka *Intervals*. Názov funkčného bloku je *FbElapsedTime*. Štartovanie merania bolo pomocou metódy *StartNow()* a následné odčítanie ubehnutého času pomocou *GetElapsedLTime()*. Táto funkcia meria čas v nanosekundách takže bolo potrebné následne čas prerátať na mikrosekundy. Bohužiaľ sa v dokumentácii neuvádza presnosť tejto funkcie ani som ju nedokázal dohľadať.

Ukážka meracieho programu, ktorý sa používal pri každom meraní v tomto prostredí, je v prílohe vo výpise A.1.

Aplikačne orientované meranie

1. Spracovanie vstupov a výstupov PLC.

Tab. 2.1: Namerané hodnoty pri spracovaní vstupov a výstupov v e!COCKPIT.

Počet I/O	Maximálny čas [us]	Priemerný čas [us]	Minimálny čas [us]
50	338	62	30
100	400	115	92
200	584	220	184
300	861	345	276
400	1015	429	338
500	1384	531	430

Lepšie je vidno závislosť času na počte vstupov a výstupov na grafe, ktorý je umiestnený v prílohe ako obrázok A.2.

2. Spracovanie dát.
   Pri spracovaní dát v tomto prostredí boli použité dáta typu BOOL, INT, WORD, REAL a STRING(1). Program je napísaný tak aby spĺňal požiadavky, ktoré sú popísané vyššie.

Tab. 2.2: Namerané hodnoty pri spracovaní dát v e!COCKPIT.

Maximálny čas [us]	Priemerný čas [us]	Minimálny čas [us]
1107	169	123

Jazykovo orientované meranie

1. Operácie s jednotlivými typmi dát.
   Pri tomto meraní boli zasa použité dátové typy ako pri meraní spracovania dát ale každý dátový typ bol meraný osobitne, nie ako komplexná aplikácia.

Tab. 2.3: Namerané hodnoty pri spracovaní dátových typov v e!COCKPIT.

Dátový typ	Maximálny čas [us]	Priemerný čas [us]	Minimálny čas [us]
BOOL	1384	479	400
INT	1353	480	400
WORD	1600	481	400
REAL	2400	1569	1415
STRING(1)	3784	2777	2554

2. Operácie s odvodenými typmi dát.
   Toto meranie zobrazuje ako rýchlo dokáže program pristupovať k premenným, ktoré sú uložené v poli alebo štruktúre.

Tab. 2.4: Namerané hodnoty pri spracovaní odvodených typov dát v e!COCKPIT.

Typ	Maximálny čas [us]	Priemerný čas [us]	Minimálny čas [us]
Pole	1077	171	123
Štruktúra	307	63	30

3. Prístup k premenným.
   Pri tomto meraní zisťujeme či program dokáže rýchlejšie pristupovať k lokálnym alebo globálnym premenným.

Tab. 2.5: Namerané hodnoty pri práci s premennými v e!COCKPIT.

Typ premenných	Maximálny čas [us]	Priemerný čas [us]	Minimálny čas [us]
Lokálne	369	74	61
Globálne	492	79	61

4. Funkcie na ovládanie toku programu.
   Toto meranie ukazuje, ktorú z funkcií dokáže program spracovať rýchlejšie.

Tab. 2.6: Namerané hodnoty funkcií na ovládanie toku v e!COCKPIT.

Typ	Maximálny čas [us]	Priemerný čas [us]	Minimálny čas [us]
CASE	307	49	30
FOR	338	80	61
WHILE	369	61	30
IF	276	47	30

5. Volanie programových blokov.
   V tomto meraní sa meria čas, za ktorý sa dokážu vykonať prázdne programové bloky, respektíve za koľko ich dokáže hlavný program zavolať.

Tab. 2.7: Namerané hodnoty volaní programových blokov v e!COCKPIT.

Typ	Maximálny čas [us]	Priemerný čas [us]	Minimálny čas [us]
F	307	49	30
FB	338	80	61

2.1.2 Meranie v prostredí Embedded

Vytváranie programov v tomto prostredí nieje také jednoduché ako v e!COCKPIT prostredí. Program ovládania vstupov a výstupov je napísaný v programovacom jazyku C a zvyšné programy v C++, je to hlavne kvôli tomu aby bolo možné pracovať s premennou bool, ktorá v jazyku C ako takom nieje. Pre prácu s Embedded prostredím je potrebné mať v počítači nainštalovaný operačný systém Ubuntu minimálne verzie 14.02, prípadne virtuálny operačný systém. Ja som pre prácu využíval virtuálny operačný systém Ubuntu verzie 16.04. Ďalej je potrebné nainštalovať do tohto operačného systému balíčkovací nástroj PTXdist, ktorý dokáže nami vytvorené programy skompilovať a vytvoriť z nich balíčky pre Embedded prostredie v PLC. Balíčkovací nástroj slúži tiež na vytvorenie softwaru na ktorom beží PLC a do ktorého následne môžeme nahrávať nami vytvorené balíčky s programami. Keďže v PLC už je software, nebolo potrebné vytvárať image a nahrávať ho do PLC, stačilo aby som si zo svojich programov spravil balíčky a tie uploadoval do pôvodného softwaru PLC.

Inštaláciu nástroju PTXdist popisuje podrobne výrobca PLC vo svojich súboroch s názvom Board support packages. Pre túto prácu bola použitá najnovšia verzia WAGO-PFC-BSP-2017.3.1. Táto inštalácia bola dosť zložitá a zdĺhavá, obsahuje veľké množstvo krokov a prebieha celá v termináli. Viac o tomto nástroji, jeho inštalácii a práce s ním som si naštudoval aj na stránke výrobcu Pengutronix, ktorá je uvedená v literatúre [12].

Po inštalácii, je potrebné vytvoriť balíčky, ktoré sa nahrávajú do PLC. Pri vytváraní aplikácie, ktorá pracuje so vstupmi a výstupmi PLC som postupoval podľa návodu, ktorý dodáva výrobca v súbore WAGO-PFC-BSP-2016.2.1-howtos v zložke How_TO-ADI-MyKbusApplikation. Tu je podrobný návod ako vytvoriť balíček aj ukážkové kódy na ovládanie vstupov a výstupov. Aplikácia pristupuje k vstupom a výstupom cez ADI/DAL rozhranie pomocou protokolu KBus, toto rozhranie je popísané v kapitole 1.1.7. Tieto kódy som nahradil svojim kódom, aplikáciu skompiloval a vytvoril balíček, ktorý som potom nahral do PLC cez WebBasedManagement, ktorý je popísaný v teoretickej časti práce. Tu sa stačí prihlásiť a nahrať balíček cez funkciu *Software Upload*.

Podobne som postupoval aj pri vytváraní všetkých ostatných aplikácií, ktoré boli napísané v jazyku C++. Jediný rozdiel bol v tom, že som využíval návod od výrobcu, ktorý je dodávaný v súbore WAGO-PFC-BSP-2014.10.3-howtos v zložke How_To-Cplusplus.

Všetky tieto súbory aj so zdrojovými kódmi sú uložené na CD disku, ktorý je dodaný k práci.

Prevedenie merania

O prácu s časom sa v tomto prostredí starala funkcia z knižnice *time.h*. Je to funkcia s názvom *clock_gettime()*. Funkciu som volal s parametrom *CLOCK_PROCESS_CPUTIME_ID*, v tomto prípade pracuje funkcia s časovačom s najvyšším rozlíšením, čo zaručuje vysokú presnosť pri meraní. Funkcia vracia štruktúru typu *struct timespec*, ktorá obsahuje premenné *tv_sec a tv_nsec*, do ktorých sa ukladá čas od spustenia PLC v sekundách a nanosekundách, takže treba časy následne prerátať na mikrosekundy. Túto funkciu som vždy volal na začiatku a na konci merania, tak som získal dva časy, ktoré som po meraní od seba odčítal a tak získal čas vykonania mojej aplikácie. Po vypočítaní času som spracovával minimálny, maximálny a priemerný čas, za ktorý sa aplikácia vykonala.

Výpis kódu, ktorý vykonával meranie sa nachádza v prílohách ako výpis A.2.

Aplikačne orientované meranie

1. Spracovanie vstupov a výstupov PLC.

Tab. 2.8: Namerané hodnoty pri spracovaní vstupov a výstupov v Embedded.

Počet I/O	Maximálny čas [us]	Priemerný čas [us]	Minimálny čas [us]
50	1476	1373	1353
100	2953	2860	2830
200	5630	5543	5508
300	8400	8259	8215
400	11108	10947	10923
500	13877	13681	13661

   Lepšie je vidno závislosť času na počte vstupov a výstupov na grafe, ktorý je umiestnený v prílohe ako obrázok A.3.
2. Spracovanie dát.
   Pri spracovaní dát v tomto prostredí boli použité dáta typu bool, int16_t, uint16_t, float a char. Program je napísaný tak aby spĺňal požiadavky, ktoré sú popísané vyššie.

Tab. 2.9: Namerané hodnoty pri spracovaní dát v Embedded.

Maximálny čas [us]	Priemerný čas [us]	Minimálny čas [us]
215	85	62

Jazykovo orientované meranie

1. Operácie s jednotlivými typmi dát.
   Pri tomto meraní boli zasa použité dátové typy ako pri meraní spracovania dát ale každý dátový typ bol meraný osobitne.

Tab. 2.10: Namerané hodnoty pri spracovaní dátových typov v Embedded.

Dátový typ	Maximálny čas [us]	Priemerný čas [us]	Minimálny čas [us]
bool	523	386	369
int16_t	585	388	369
uint16_t	431	316	277
float	1846	1663	1600
char	369	247	215

2. Operácie s odvodenými typmi dát.
   Toto meranie zobrazuje ako rýchlo dokáže program pristupovať k premenným, ktoré sú uložené v poli alebo štruktúre.

Tab. 2.11: Namerané hodnoty pri spracovaní odvodených typov dát v Embedded.

Typ	Maximálny čas [us]	Priemerný čas [us]	Minimálny čas [us]
Pole	215	128	92
Štruktúra	123	26	20

3. Prístup k premenným.
   Pri tomto meraní zisťujeme či program dokáže rýchlejšie pristupovať k lokálnym alebo globálnym premenným.

Tab. 2.12: Namerané hodnoty pri práci s premennými v Embedded.

Typ premenných	Maximálny čas [us]	Priemerný čas [us]	Minimálny čas [us]
Lokálne	123	49	31
Globálne	154	50	31

4. Funkcie na ovládanie toku programu.
   Toto meranie ukazuje, ktorú z funkcií dokáže program spracovať rýchlejšie.

Tab. 2.13: Namerané hodnoty funkcií na ovládanie toku v Embedded.

Typ	Maximálny čas [us]	Priemerný čas [us]	Minimálny čas [us]
CASE	123	40	30
FOR	185	55	31
WHILE	154	53	31
IF	154	40	31

5. Volanie funkcie.
   V tomto meraní sa meria čas, za ktorý sa dokáže vykonať prázdna funkcia, respektíve za koľko ju dokáže hlavný program zavolať. Funkcia má 5 vstupov rôznych typov premenných a jeden výstup.

Tab. 2.14: Namerané hodnoty volaní programových blokov v Embedded.

Typ	Maximálny čas [us]	Priemerný čas [us]	Minimálny čas [us]
Funkcia	154	54	31

2.1.3 Vplyv priority na meranie

V oboch prostrediach som skúšal meniť prioritu vykonávaného programu ale nemalo to na meranie žiaden vplyv. Vplyv priority by sa prejavil až pri vykonávaní viacerých programov súčasne a vtedy by bolo potrebné uvažovať nad zvolením vyššej alebo nižšej priority pre daný program.

V e!COCKPIT prostredí sa priorita daného programu nastavuje spolu s časom cyklu vykonávania a to v položke *Task Configuration*.

V Embedded prostredí by podľa predbežných úvah mala zmena priority programu zasahovať do nameraného času ale ani v tomto prípade tomu tak nebolo. Zrejme je to spôsobené tým, že náš program beží v užívateľskom priestore ako klasické linux vlákno a zasahujú doňho real-time vlákna. Čas vykonania nášho programu by sa teda začal meniť až v prípade, že by sme nahrali do PLC viac programov a týmto programom by sme upravovali prioritu. Zmena priority je v tomto prípade možná pomocou príkazu *nice priorita názov_programu*. Priorita je číslo od -20 do 19, pričom 19 je najnižšia priorita a -20 najväčšia.

2.1.4 Čas obnovenia časovaču PLC

Po ukončení merania prišlo na rad zisťovanie presnosti merania. Presnosť sa priamo odvíja od toho, ako často sa dokáže obnovovať hodnota časovaču v PLC. Táto hodnota bola meraná v e!COCKPIT prostredí ale aj Embedded prostredí.

K tejto hodnote sa dá dopracovať nasledovne. Stačí ak mnohokrát po sebe zavoláme funkciu, ktorá nám dáva hodnotu času a tieto hodnoty si niekde uložíme, prípadne vypíšeme. V e!COCKPIT prostredí to bola funkcia *GetElapsedLTime ()*, ktorej 100 hodnôt som si uložil do poľa. V Embedded prostredí to zasa bola funkcia *clock_gettime()*, ktorej 100 hodnôt som si nechal vypísať do terminálu.

V týchto nameraných hodnotách som následne hľadal čísla, ktoré sa rovnajú, respektíve je medzi nimi nulový rozdiel. Rozdelil som si namerané hodnoty na takéto skupinky a hľadal časový rozdiel medzi týmito skupinkami. Inak povedané, časovač sa nejakú dobu neobnovuje a preto nameriame rovnaké hodnoty a keď sa obnoví zasa meriame hodnoty iné. Stačí nájsť bod obnovenia a odrátať tieto dva čísla. Teda od novej hodnoty odrátame tú starú. Pri nameraní povedzme tých sto hodnôt je možné nájsť niekoľko takýchto prechodov, takže si môžeme vypočítaný čas niekoľko krát overiť.

Pri tomto meraní som zistil, že sa časovač sa v oboch prostrediach obnovuje každých 30772ns. Z toho vyplýva, že pri každej nameranej hodnote vznikla chyba až **+31us**. Pri meraní v mikrosekundách je to veľká nepresnosť. Z toho vyplýva, že napríklad nameraná hodnota 585us môže byť v skutočnosti v rozmedzí 585-616us, pretože sa časovač po dobu 30,772us neobnovil a systém tak pracoval so starou hodnotou.

2.1.5 Zhodnotenie merania

Celé meranie je spracované na jeden výstupný formulár, ktorý musí obsahovať všetky náležitosti, ktoré popisuje certifikát TC3 v teoretickom úvode v kapitole 1.2. Tento

formulár aj s informáciami o PLC a použitom software sa nachádza v prílohe A.3. Meranie jasne ukazuje v čom je jedno a druhé prostredie výhodnejšie.

e!COCKPIT prostredie je omnoho rýchlejšie pri spracovávaní vstupov a výstupov. Zrejme to bude spôsobené tým, že e!COKPIT prostredie nezapisuje zakaždým na výstupy a nečíta neustále zo vstupov. Toto ale program v Embedded prostredí vykonával pri každej operácii čím sa kód mnohonásobne predĺžil.

Embedded prostredie je zasa rýchlejšie pri aplikácii, ktorá spracováva dáta. Pri jednotlivých dátach je Embedded prostredie o čosi pomalšie pri práci s typom float, inak je celkovo rýchlejšie.

Funkcie pre riadenie toku programu sú v oboch prostrediach približne na rovnakej úrovni, Embedded je však o čosi rýchlejšie.

Pri práci s odvodenými typmi dát ako sú polia a štruktúry je Embedded prostredie zasa omnoho rýchlejšie.

Embedded prostredie dokáže rýchlejšie volať funkcie a na záver dokáže aj rýchlejšie pracovať s lokálnymi a globálnymi premennými.

Síce vo väčšine vykonaných programov Embedded prostredie prevyšuje druhé prostredie, je za to omnoho zložitejšie pre prácu v ňom. Je stavané skôr pre náročnejšieho programátora, ktorý má s týmto prostredím väčšie skúsenosti a takisto mu ponúka aj väčší prístup k systému PLC. e!COCKPIT prostredie zasa neponúka taký rýchly čas spracovania ale je to na úkor jeho intuitívnosti a jednoduchosti. Taktiež nedokážeme pomocou e!COCKPIT prostredia pracovať so systémom ako to je pri Embedded prostredí, zasa tu máme väčšiu podporu od výrobcu v podobe rôznych knižníc.

2.2 Aplikácia zobrazujúca vyťaženie celého systému

Táto aplikácia je vytvorená za účelom informovania užívateľa, prípadne programátora o vyťažení celého PLC. Keďže sú údaje ukladané do premenných v e!COCKPIT prostredí je jednoduché ich zobraziť napríklad vo vizualizácii.

Aplikácia sa delí na program v e!COCKPIT prostredí písaný v jazyku ST a program v Embedded prostredí, ktorý je v tvare shell skriptu.

Užívateľovi sú zobrazované informácie o priemernom vyťažení procesoru (od zapnutia PLC), aktuálnom vyťažení procesoru, vyťažení pamäte PLC, počte aktuálne bežiacich procesov a o čase cyklu PLC.

2.2.1 Program v embedded prostredí

Program v tomto prostredí pozostáva z jedného shell skriptu, ktorý vyčíta potrebné informácie zo systémových súborov prípadne ich získava pomocou rôznych funkcií.

Na začiatku je potrebné vytvoriť skript v pamäti PLC v zložke etc/config=tools, pretože iba v tejto zložke môže e!COCKPIT spúšťať skripty. Po vytvorení skriptu je buď možné upravovať tento skript pomocou nástroju *nano*, ktorý je možné spustiť priamo cez terminál alebo pohodlnejšie cez obyčajný textový editor, v Linuxe to môže byť napríklad program *gedit*.

Aby sme toto všetko boli schopný vykonať, je najprv potrebné pripojiť sa na PLC. Pre prácu v termináli je potrebná komunikácia cez ssh a pre prácu so súbormi ftp. Skúsenejším programátorom stačí práca v termináli no pohodlnejšie je aj vidieť všetky zložky a súbory v PLC a na to je potrebná aj ftp. Na začiatok je vhodné, skontrolovať si či je vôbec táto komunikácia povolená, to je možné cez Web based Management v záložke *Ports and Services*.

Keď máme túto konfiguráciu hotovú, môžeme sa pripojiť na PLC a to príkazmi v termináli *ssh root@IP_adresa* alebo cez prehliadač súborov *ftp://root@IP_adresa* v oboch prípadoch bude vyžadované heslo, základné heslo pre užívateľa root je *wago*. Po pripojení môžeme začať s písaním skriptu.

Kompletný výpis programu je umiestnený v prílohe B.1. Jeho časti sú vysvetlené ďalej v texte.

Po dopísaní skriptu je nutné skontrolovať či má správne nastavené prístupové práva a prípadne ich nastaviť. V termináli na to stačí použiť príkaz *chmod +x nazov_skriptu*.

Vyťaženie procesoru

Aplikácia na vyťaženie procesoru využíva systémové informácie, ktoré si systém ukladá do súboru *stat*. Tento súbor je umiestnený v zložke *proc*. Výpis z tohto súboru môžeme získať príkazom v termináli *cat /proc/stat*. Nás však zaujíma len riadok, ktorý sa nazýva *cpu*. Na tomto riadku sú všetky časy, ktoré procesor strávil pri vykonávaní nejakých prerušení, procesov a takisto, ktoré strávil dalo by sa povedať nič nerobením, tzv. idle time. Časy v tomto súbore sú vo formáte stotín sekundy. Aby sme získali tieto časy je potrebné použiť príkaz
Cpu=(‘cat /proc/stat | grep ’cpu ’‘)
Táto funkcia vráti pole ktorého prvý prvok je práve text *cpu*, ktorý funkcia grep našla v súbore, zvyšné prvky sú časy ktoré sú potrebné na výpočet vyťaženia procesoru, tie program pošle do prostredia e!COCKPIT, kde budú prerátané.

Vyťaženie pamäte

Informácie o vyťažení pamäte získame podobne ako pri získavaní informácií o procesore. Tieto informácie sú akurát uložené v súbore */proc/meminfo*. Sú však uložené na dvoch riadkoch tohto súboru a to na *MemTotal* a *MemFree*, takže som použil príkazy
MemTotal=(‘cat /proc/meminfo | grep 'MemTotal:'‘)
MemFree=(‘cat /proc/meminfo | grep 'MemFree:'‘)
Tieto funkcie zasa vracajú prvky z riadkov, ktoré našli pomocou funkcie *grep* a uložia ich do príslušných polí. Takto získané údaje o veľkosti pamäte a voľnej pamäti stačí už len prerátať a poslať do e!COCKPIT prostredia pomocou funkcie *echo*.

Počet bežiacich procesov

Výpis všetkých procesov je možné získať v termináli pomocou funkcie *ps -A*. My však potrebujeme počet týchto procesov, na tom nám poslúži funkcia *wc -l*, ktorá zráta počet riadkov predchádzajúcej funkcie. Po spustení týchto funkcií sa však zvýši počet bežiacich procesov a takisto sa k bežiacim procesom pripočíta hlavička funkcie, ktorá vypisuje bežiace procesy, takže toto číslo treba v konečnom dôsledku odrátať pomocou ďalšej funkcie, ktorú treba takisto odrátať, čiže výsledný kód na zistenie počtu bežiacich funkcií vyzerá takto:
$(ps -A|tail -n +5|wc -l)
Zasa stačí výsledok poslať do e!COCKPIT prostredia.

2.2.2 Program v e!COCKPIT prostredí

Výpis celého programu v tomto prostredí sa nachádza v prílohách vo výpise B.2. Podrobné vysvetlenie sa nachádza v nasledujúcom texte.

Volanie skriptu

Základom celého tohto programu je volanie skriptu, ktorý dodáva takmer všetky potrebné informácie o vyťažení PLC. Volanie je zabezpečené pomocou knižnice *WagoAppConfigTool*. Volanie zabezpečuje konkrétne funkčný blok *FbConfigTool*. V tejto práci bola využívaná verzia knižnice 1.0.1.1. Funkčný blok obsahuje rôzne vstupy a výstupy, ktoré je možné nastavovať, prípadne z nich čítať. V mojej práci som využil len vstupy *xExecute*-typu bool, pri nástupnej hrane zavolá shell skript. *sCallString*-typu STRING(255), ukladá sa tu názov skriptu, prípadne parametry s ktorými chceme skript volať ak nejaké sú, v tomto prípade to bolo *./get_info*. A výstup *sResultString*-typu STRING(255), návratová hodnota zo skriptu, v ktorej

sú uložené premenné, ktoré využívame na určenie vyťaženia. Tvar stringu prichádzajúceho zo skriptu je *"cpu_vyuzity_cas ; cpu_celkovy_cas ; vytazenie_pamate ; beziace_procesy"*.

Nedostatkom knižnice, ktorá volá skript v Embedded prostredí je, že funkčný blok, ktorý volá skript je nutné raz volať so vstupom v stave TRUE a po vykonaní so vstupom v stave FALSE, potom ju znovu môžeme volať so vstupom v stave TRUE aby sa následne skript znovu vykonal. Keby sme toto nespravili, skript by sa vykonal len raz a následne sa už viac nevykonal. Tento problém spôsobuje spomalenie pri odovzdávaní údajov medzi shell skriptom a e!COCKPIT programom, pretože program musí prebehnúť minimálne tri krát zatiaľ čo získa údaje len raz. Je to spôsobené tým, že shell skript sa vykonáva asynchrónne s programom, ktorý ho volá, čiže program zavolá shell skript a pokračuje ďalej bez toho aby počkal na výstup zo shell skriptu.

Ukladanie do premenných

Výsledný string, ktorý prišiel zo skriptu je potrebné samozrejme rozdeliť na jednotlivé premenné. Na to som využíval v e!COCKPIT knižnicu *WagoAppString*. Verzia knižnice, ktorú som využíval bola 1.6.1.2. Z tejto knižnice bol pre mňa najvýhodnejší pre túto činnosť funkčný blok *StrExtractToken*. Tento funkčný blok vytiahne zo stringu jednu položku až po oddeľovač, ktorý určíme, túto položku môžeme spracovať a pokračovať vo vyťahovaní ďalšej položky, pretože funkcia vracia polohu v stringu v ktorej prestala a túto polohu stačí predať ďalšej funkcii a tá vytiahne ďalšiu položku.

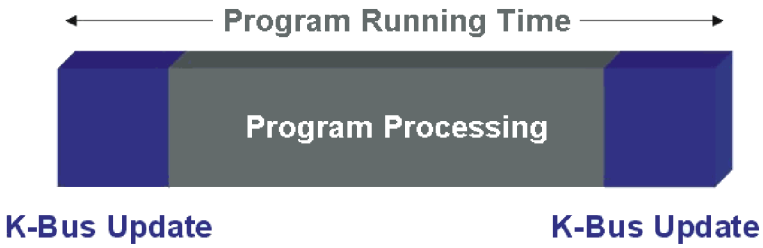
Po oddelení jednotlivých stringov som ich postupne spracovával. Najprv bolo potrebné konvertovať string na číselnú premennú. Pre túto úlohu poslúžila funkcia *STRING_TO_REAL()*. Samozrejme bolo treba niektoré čísla prerátať na percentá, toto spracovanie je vidieť vo výpise programu.

Čas cyklu PLC

Sledovanie času cyklu umožňuje program e!COCKPIT pri diagnostike modulu cez tento program. Vďaka tejto funkcii môžeme sledovať ako rýchlo dokáže PLC vykonať cyklus. Samozrejme môžeme sledovať priemerný, maximálny a minimálny čas cyklu.

Čas cyklu nám určuje frekvenciu opakovania programu. Je to vlastne čas od kedy program prečíta vstupy, vykoná náš kód a zapíše na výstupy ako to je vidieť na nasledujúcom obrázku.

Tento čas je pri každom programe rozdielny a závisí na veľkosti programu. Je to veľmi dôležitá položka pri nastavovaní času opakovania programu v položke *Task*



Obr. 2.1: Čas cyklu PLC.[13]

Configuration. Pred týmto nastavením je potrebné zmerať si priemerný čas cyklu nášho programu a k tomu prirátať minimálne 20 až 50%, pretože v čase kedy sa neobsluhuje náš program pracuje procesor na spracovávaní požiadavok Ethernetu a sériového rozhrania a ak by mal na toto málo času mohla by sa prerušiť komunikácia s PLC a nedalo by sa ovládať. Viac informácii o Cykle PLC je napríklad v literatúre [13].

Maximálny, priemerný a minimálny čas cyklu je uložený v systémových premenných PLC. Je jednoduché ich sledovať pri diagnostike systému ale zložitejšie dostať do programu. Dá sa k nim dopracovať cez knižnicu *AC_DeviceDiagnosis*. V tomto programe bola využitá verzia knižnice 3.5.9.0. Premenné sú ukryté v zložke */AC_DD/AC/CmpIecTask/* a následne v štruktúre s názvom *Task_Info2*. Názvy premenných sú *dwAverageCycleTime, dwMaxCycleTime, dwMinCycleTime*. Tieto premenné ale nieje možné získať priamou cestou k štruktúre *Task_Info2*, pretože premenné štruktúre nepatria. Je ich možné získať jedine pomocou funkcie *IecTaskGetInfo3*, tak ako je to naprogramované vo výpise B.2. Tento program vychádza z literatúry [11].

3 ZÁVER

V teoretickej časti práce som sa snažil čo najpodrobnejšie popísať procesorový modul PLC PFC200, s ktorým som pracoval. Popísal som možnosti komunikácie, ovládania, napájania, jeho rozdelenie pamäte a v neposlednom rade programovanie tohto modulu a prepojenie jednotlivých programovacích prostredí.

Následne som popisoval meranie výkonu podľa certifikátu TC3. Pri popisovaní som sa snažil nevynechať žiadnu položku tohto certifikátu a dodržať tak postup pri meraní.

V prvej polovici praktickej časti som vykonával meranie výkonu na procesorovom module podľa spomínaného certifikátu. Nedostatkom tohto merania bolo to, že spomínaný certifikát je určený pre meranie výkonu v programovacích prostrediach, ktoré sú postavené na norme IEC 61131-3, čiže napríklad e!COCKPIT a CoDeSys. Preto bolo potrebné z merania výkonu vyradiť tie časti, ktoré nieje možné naprogramovať v Embedded prostredí.

Po ukončení a vyhodnotení merania bola dodatočne zistená skutočnosť, že sa časovač v PLC obnovuje každých 30,772us čo predstavuje pri meraní veľkú nepresnosť. Každá z nameraných hodnôt môže byť teda s chybou až +31us.

Meraním bolo aj tak preukázané, že Embedded prostredie je pri väčšine operácií rýchlejšie. Toto je však na úkor jednoduchosti a intuitívnosti tohto prostredia voči prostrediu e!COCKPIT. Teda je na užívateľovi či si vyberie jednoduchosť a väčšiu podporu od výrobcu v prostredí e!COCKPIT alebo výkonnejšie, zložitejšie ale za to voľnejšie prostredie Embedded.

V druhej polovici praktickej časti som sa zameriaval na tvorbu aplikácie, ktorá zobrazuje aktuálne vyťaženie celého systému. Za nedostatok tejto aplikácie by som označil e!COKPIT knižnicu, ktorá volá skript v Embedded prostredí, pretože jej funkčný blok je nutné raz volať so vstupom v stave TRUE a po vykonaní so vstupom v stave FALSE aby sa následne skript znovu vykonal. Keby sme toto nespravili, skript by sa vykonal len raz a následne sa už viac nevykonal. Tento problém spôsobuje spomalenie pri odovzdávaní údajov medzi shell skriptom a e!COCKPIT programom, pretože program musí prebehnúť minimálne tri krát zatiaľ čo získa údaje len raz.

LITERATÚRA

[1] WAGO: *Nová výkonová třída: řada procesorových modulů PFC200* [online]. 2013, posledná aktualizácia 18. 03. 2013 [cit. 22. 12. 2016]. Dostupné z URL: <http://www.wago.cz/aktuality/tiskove-zpravy/press-releases-dateils-1600.jsp>.

[2] WAGO: *Pro vývojáře pracující s embedded Linuxem: PFC200 jako linuxový procesorový modul* [online]. 2014, posledná aktualizácia 18. 03. 2014 [cit. 22. 12. 2016]. Dostupné z URL: <http://www.wago.cz/aktuality/tiskove-zpravy/press-releases-dateils-8839.jsp>.

[3] WAGO: *WAGO-I/O-SYSTEM 750 MANUAL, 750-8202, PFC200 CS 2ETH RS* [online]. Version 3.1.0, posledná aktualizácia 14. 11. 2016 [cit. 20. 12. 2016]. Dostupné z URL: <http://www.wago.cz/download.esm?file=\download\00375232_0.pdf&name=m07508202_xxxxxxxx_0_en.pdf>.

[4] ATP journal: *10/2015* [online]. strana 26,27 [cit. 10. 12. 2016]. Dostupné z URL: <http://www.atpjournal.sk/buxus/docs/casopisy_cele/ATP%20journal%2010%202015.pdf>.

[5] WAGO: *WAGO-PFC-BSP-howtos* [online]. Version 2016.2.1, [cit. 20. 12. 2016]. Dostupné z URL: <http://www.wago.com/public-download/WAGO-PFC-BSP-2016.2.1-howtos.zip>.

[6] WAGO: *WAGO-PFC-BSP*. Version 2017.3.1, [cit. 10. 5. 2017].

[7] WAGO: *WAGO-PFC-BSP-howtos*. Version 2014.10.3, [cit. 10. 5. 2017].

[8] WAGO: *Application note: ADI/DAL for PFC*. Version 1.0.0, [cit. 15. 12. 2016].

[9] atp International: *The PLC Benchmark*. Version 1/2007, [cit. 10. 12. 2016].

[10] PLCopen: *Task Force - Benchmarking*. Version 0.4, [cit. 10. 12. 2016].

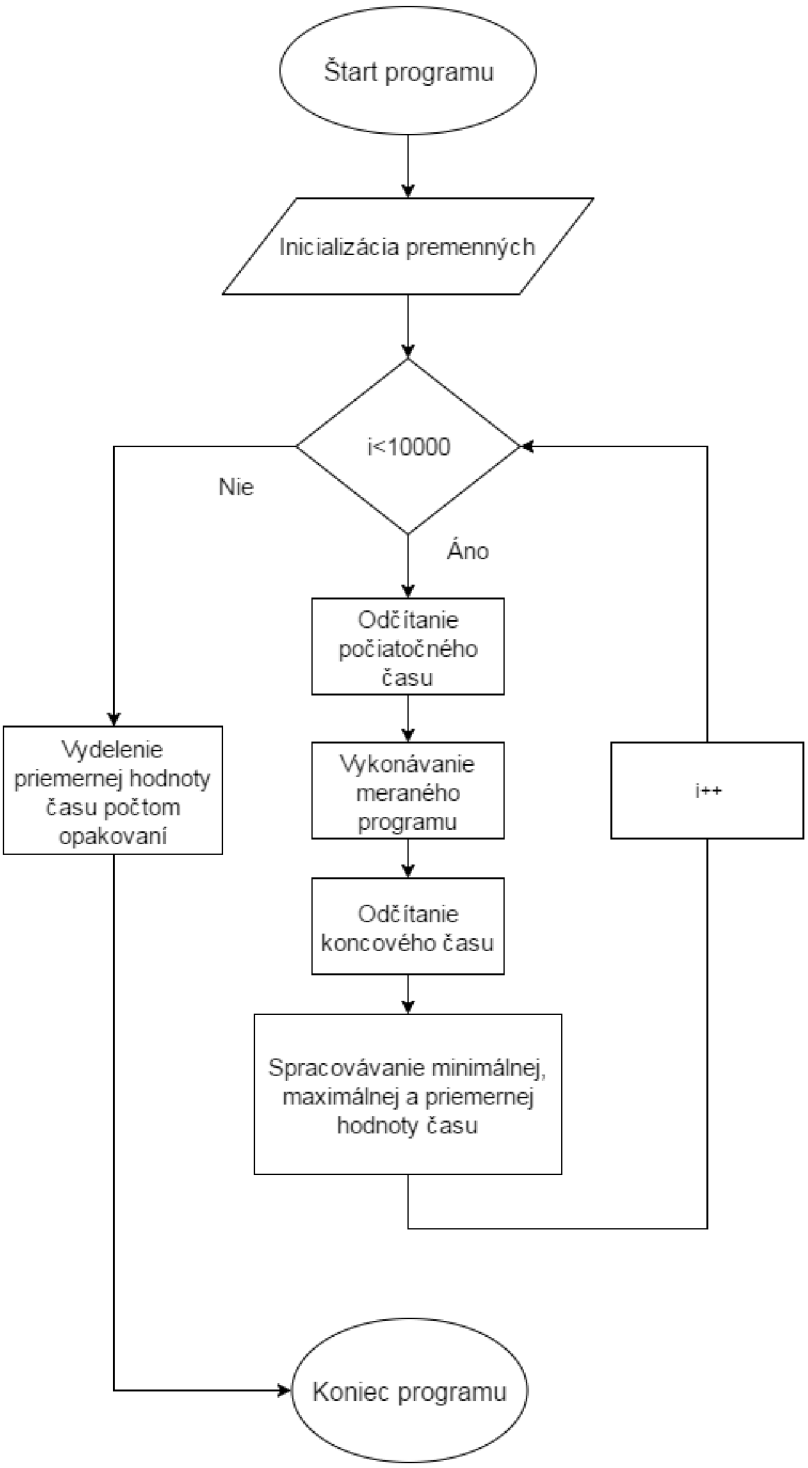
[11] DEIF: *Advanced Wind turbine Controller AWC 500 - IEC61131-3 programming* [online]. Revision D, [cit. 16. 5. 2017]. Dostupné z URL: <https://deif-cdn.azureedge.net/v-ch16bm8ilcv2/-/media/files/documents/awc-500/4189340738_awc_500_iec61131-3_programming.pdf>.

[12] Pengutronix: *PTXdist* [online]. [cit. 18. 5. 2017]. Dostupné z URL: <http://www.pengutronix.de/software/ptxdist/appnotes_en.html>.

[13] Beckhoff: *PLC Cycle Time* [online]. [cit. 20. 5. 2017]. Dostupné z URL: <https://infosys.beckhoff.com/italiano.php?content=../content/1040/bc4000/html/bt_bc%20task%20time.htm&id=>.

ZOZNAM PRÍLOH

A PRÍLOHY K MERANIU VÝKONU



Obr. A.1: Diagram princípu meracieho programu.

A.1 e!COCKPIT prostredie

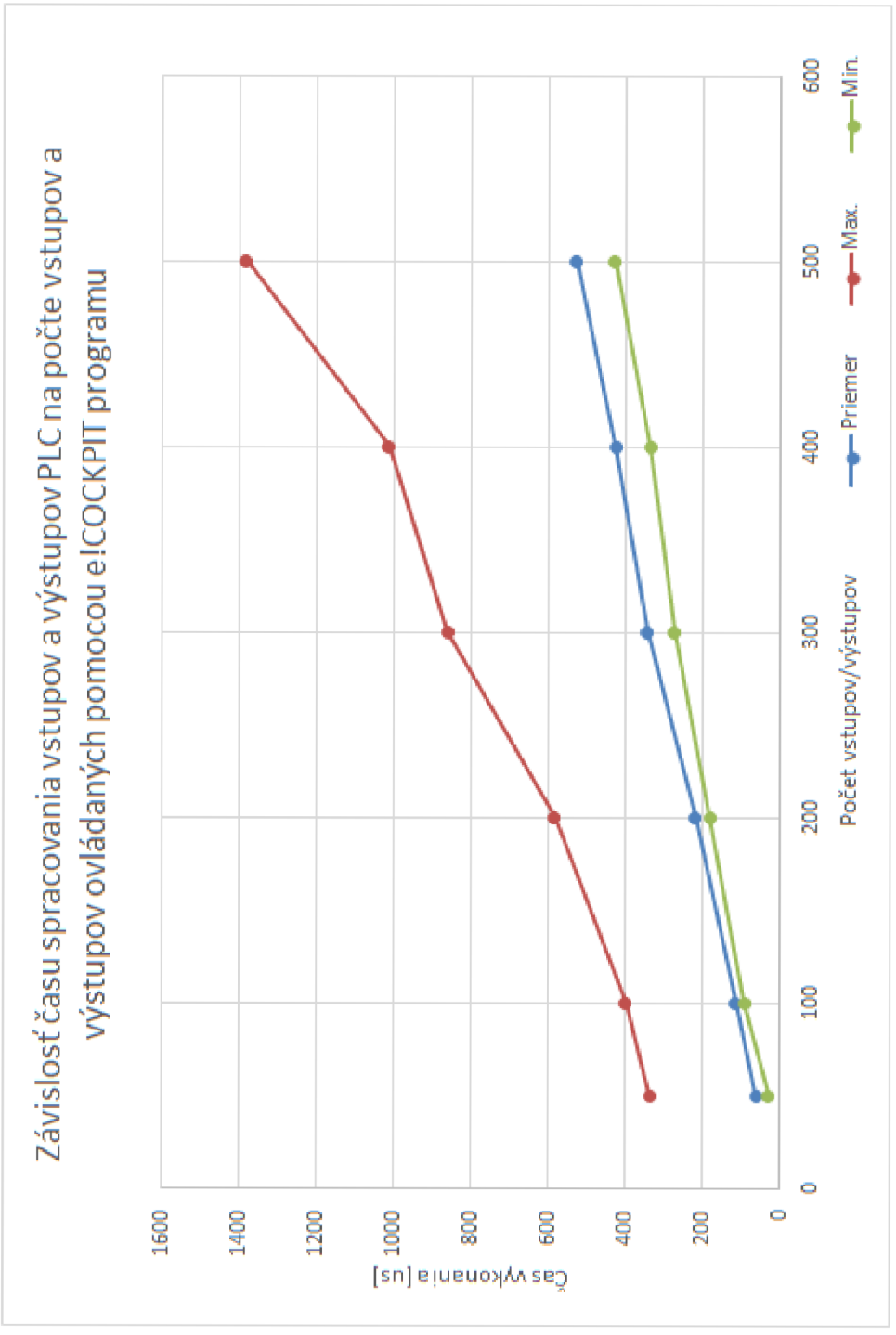
Výpis A.1: Ukážka meracieho programu v e!COCKPIT prostredí.

```
PROGRAM Main
VAR
  Present: DINT;
  Maximal: DINT;
  Minimal: DINT;
  Average: DINT;
  i: INT;
  Measure: FbElapsedTime;
END_VAR
-------------------------------------------------------------

Present := 0;
Maximal := 0;
Minimal := 10000;
Average := 0;
FOR i:= 0 TO 10000 DO
  Measure.StartNow();

  //Tu je umiestnený meraný program, prípadne
  //volanie blokov s meraným programom

  Present:= (LTIME_TO_DINT(Measure.GetElapsedLTime()/1000));
  Average := (Average+Present);
  IF Minimal > Present THEN
    Minimal := Present;
  END_IF
  IF Maximal < Present THEN
    Maximal := Present;
  END_IF
END_FOR
Average := Average/10000;
```



Obr. A.2: Graf č.1

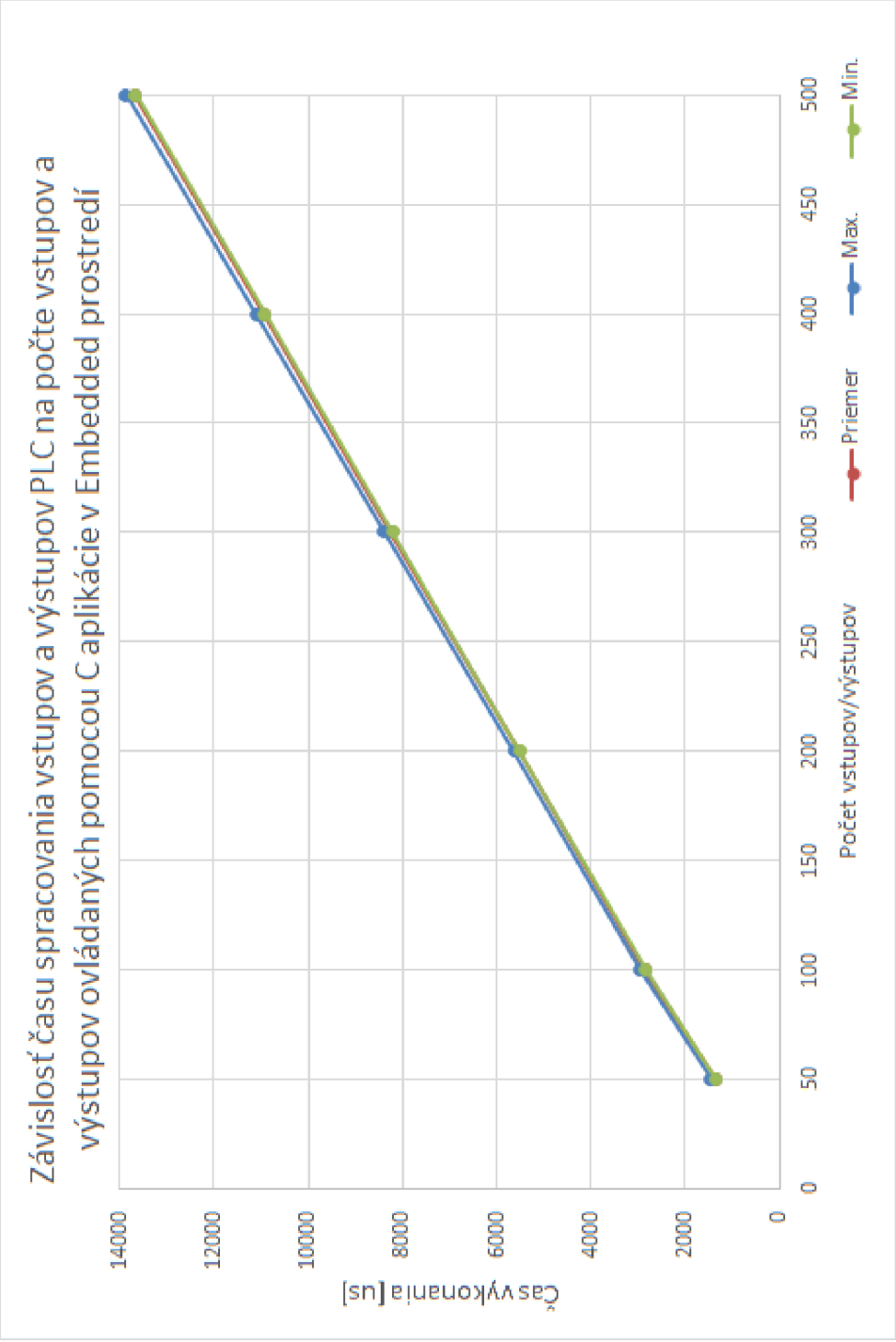
A.2 Embedded prostredie

Výpis A.2: Ukážka meracieho programu v Embedded prostredí.

```
int main(void)
{
  struct timespec start, end;
  double present = 0, max = 0, min = 1000000, avg = 0;
  int i;
  for (i = 0; i < 10000; i++)
  {
    //odcitanie pociatocneho casu
    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &start);

    //------------------MERANY PROGRAM----------------

    //odcitanie koncoveho casu
    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &end);
    //vypocet trvania programu v us
    present = ((double)(end.tv_sec - start.tv_sec) * 1000000)
    + ((double)(end.tv_nsec - start.tv_nsec) / 1000);
    if (max < present)
      max = present;
    if (min > present)
      min = present;
    avg = avg + present;
  }
  //vypocet priemerneho casu
  avg = avg / 10000;
  //vypis nameranych hodnot
  cout << "Meranie casu zavolania funkcie." << '\n';
  cout << "MAX:" << max <<'\n';
  cout << "AVERAGE:" << avg << '\n';
  cout << "MIN:" << min << '\n';
  return 0;
}
```



Obr. A.3: Graf č.2

A.3 Formulár s nameranými hodnotami

Meranie výkonu PLC PFC200

Meral	František Majchrák	Dňa	10.5.2017
Výrobca PLC	Wago	Operačný systém	Linux 3.6
Typ PLC	750-8202	Pamäť RAM	256 MB
Pripojené moduly	750-430 DI 750-530 DO 750-456 AI 750-556 AO 750-600	Vnútorná pamäť	256 MB
		Retain pamäť	128kB
		Procesor	CORTEX A8
		Frekvencia CPU	600 MHz
		e!COCKPIT	3.5.9.10
Napájací zdroj	787-1002	PTXdist	2013.03.0

		e!COCKPIT prostredie				Embedded prostredie			
	Typ merania	Podpor.	Nameraný čas [µs]			Podpor.	Nameraný čas [µs]		
			Max.	Priemerný	Min.		Max.	Priemerný	Min.
Aplikačne orientované	Spracovanie I/O								
	50	x	338	62	30	x	1476	1373	1353
	100	x	400	115	92	x	2953	2860	2830
	200	x	584	220	184	x	5630	5543	5508
	300	x	861	345	276	x	8400	8259	8215
	400	x	1015	429	338	x	11108	10947	10923
	500	x	1384	531	430	x	13877	13681	13661
	Spracovanie dát	x	1107	169	123	x	215	85	62
Jazykovo orientované	Funkcie na ovládanie toku programu								
	CASE	x	307	49	30	x	123	40	30
	FOR	x	338	80	61	x	185	55	31
	WHILE	x	369	61	30	x	154	53	31
	IF	x	276	47	30	x	154	40	31
	Operácie s dátami								
	BOOL	x	1384	479	400	x	523	386	369
	INT	x	1353	480	400	int16_t	585	388	369
	REAL	x	2400	1569	1415	float	1846	1663	1600
	WORD	x	1600	481	400	uint16_t	431	316	277
	STRING(1)	x	3784	2777	2554	char	369	247	215
	Odvodené typy dát								
	Polia	x	1077	171	123	x	215	128	92
	Štruktúry	x	307	63	30	x	123	26	20
	Volanie programových blokov								
	Funkcia	x	1200	379	307	x	154	54	31
	Funkčný Blok	x	1507	721	615	-	-	-	-
	Prístup k premenným								
	Lokálne	x	369	74	61	x	123	49	31
	Globálne	x	492	79	61	x	154	50	31

B PRÍLOHY K APLIKÁCII ZOBRAZUJÚCEJ VYŤAŽENIE SYSTÉMU

Výpis B.1: Shell skript, ktorý posiela informácie do e!COCKPIT prostredia.

```
#!/bin/bash
Cpu=(`cat /proc/stat | grep '^cpu␣'`)
MemTotal=(`cat /proc/meminfo | grep '^MemTotal:'`)
MemFree=(`cat /proc/meminfo | grep '^MemFree:'`)
MemUsage=$((${MemTotal[1]}/100))
UseMem=$((${MemTotal[1]}-${MemFree[1]}))
echo '␣'$((${Cpu[1]}+${Cpu[3]}))'␣;␣'
        $((${Cpu[1]}+${Cpu[3]}+${Cpu[4]}+${Cpu[7]}))'␣;␣'
        $(($UseMem/$MemUsage))'␣;␣'
        $(ps -A|tail -n +5|wc -l)
exit 0
```

Výpis B.2: Výpis programu z e!COCKPIT, ktorý zobrazuje vyťaženie PLC.

```
PROGRAM Inform
VAR
  Get_Info_Skript: FbConfigTool;
  CPU_Average_Usage: REAL;  //% since plc boot
  CPU_Actual_Usage: REAL;   //%
  Memory_Usage: REAL;     //%
  Running_Proc: REAL;       //number of running processes
  PLCMinCycleTime : DWORD;  //microsecond
  PLCMaxCycleTime : DWORD;  //microsecond
  PLCAvgCycleTime : DWORD;  //microsecond
  Information: STRING(255) := '';
  Info_old: STRING(255) := '';
  Extract: STRING(255) := '';
  Temp: BOOL := FALSE;
  Index: UDINT;
  CPU_Use: REAL;
  CPU_All: REAL;
  CPU_Use_old: REAL := 0;
  CPU_All_old: REAL := 0;
  Result : RTS_IEC_RESULT ;
  hTask : RTS_IEC_HANDLE;
  pTaskInfo2 : POINTER TO AC_DD.AC.CmpIecTask.Task_Info2;
END_VAR
```

```
------------------------------------------------------------
Get_Info_Skript(xExecute:= Temp,
        sCallString := './get_info',
        sResultString => Information
        );
Temp := TRUE;
IF (Information <> Info_old) THEN
  Temp := FALSE;
  Index := 1;
  Extract := strExtractToken(Information,SIZEOF(Information)
  ,Semicolon,Index,udiNext=>Index);
  CPU_Use := STRING_TO_REAL(Extract);
  Extract := strExtractToken(Information,SIZEOF(Information)
  ,Semicolon,Index,udiNext=>Index);
  CPU_All := STRING_TO_REAL(Extract);
  Extract := strExtractToken(Information,SIZEOF(Information)
  ,Semicolon,Index,udiNext=>Index);
  Memory_Usage := STRING_TO_REAL(Extract);
  Extract := strExtractToken(Information,SIZEOF(Information)
  ,Semicolon,Index,udiNext=>Index);
  Running_Proc := STRING_TO_REAL(Extract);
  Info_old := Information;
  CPU_Average_Usage := (CPU_Use*100)/CPU_All;
  CPU_Actual_Usage := ((CPU_Use-CPU_Use_old)*100)/
  (CPU_All-CPU_All_old);
  CPU_All_old := CPU_All;
  CPU_Use_old := CPU_Use;
  hTask :=
          AC_DD.AC.CmpIecTask.IecTaskGetCurrent
                                  (pResult:=ADR(Result));
  IF hTask <> RTS_INVALID_HANDLE THEN
    pTaskInfo2 :=
            AC_DD.AC.CmpIecTask.IecTaSkGetInfo3
               (hIecTask := hTask, pResult:=ADR(Result));
    PLCMinCycleTime := pTaskInfo2^.dwMinCycleTime ;
    PLCMaxCycleTime := pTaskInfo2^.dwMaxCycleTime ;
    PLCAvgCycleTime := pTaskInfo2^.dwAverageCycleTime ;
  END_IF
END_IF
```

C OBSAH PŘILOŽENÉHO CD

- `/` ..... koreňový adresár priloženého CD
  - `Bakalárska práca.pdf`
  - `eCOCKPIT`
    - `Benchmark.ecp` ..... projekt obsahujúci všetky programy
  - `Embedded` ..... programy z Embedded prostredia
    - `Application`
      - `DataProcessing.cpp` ..... program spracovania dát
      - `IOProcessing.c` ..... program spracovania I/O
    - `Language`
      - `ControlStatements.cpp` ..... program funkcií na riadenie toku
      - `DataOperation.cpp` ..... program spracovanie jednotlivých dát
      - `DerivedData.cpp` ..... program zapisovania do odvodených typov dát
      - `FunctionCall.cpp` ..... program volania funkcie
      - `VarAccess.cpp` ..... program prístupu k premenným
    - `Shell Skript`
      - `get_info` ..... skript odovzdávajúci informácie o vyťažení