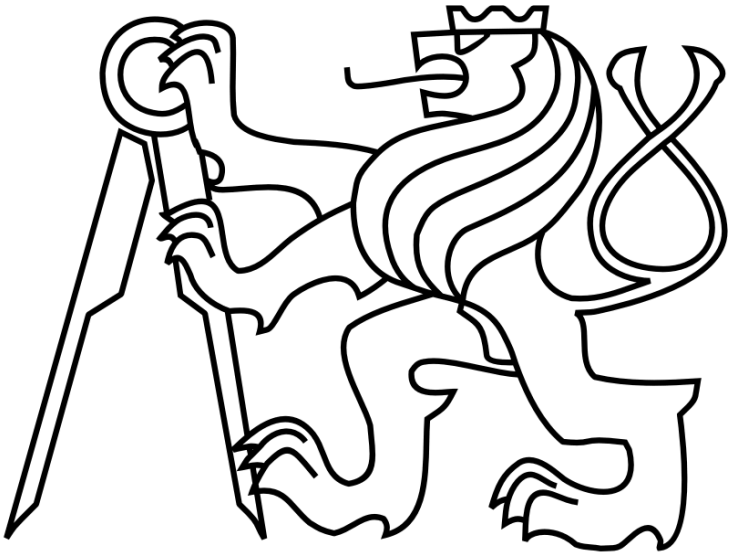
České Vysoké Učení Technické v Praze

Fakulta Elektrotechnická

Katedra kybernetiky



Monitorovací a servisní rozhraní pro novou generaci řídicího a regulačního systému

BAKALÁŘSKÁ PRÁCE

Sebastian Skoupý

Vedoucí : Ing. Michal Sojka, PhD.

Studijní program: Kybernetika a Robotika

Obor: Robotika

2013

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Sebastian S k o u p ý

Studijní program: Kybernetika a robotika (bakalářský)

Obor: Robotika

Název tématu: Monitorovací a servisní rozhraní pro novou generaci řídicího a regulačního systému

Pokyny pro vypracování:

1. Nastudujte základy architektury procesoru TI OMAP-L137 a HW zapojení řídicího regulátoru Emadyn-F.
2. Nastudujte základy Code Comoser Studio (CCS) a SW RTSC, SYS/BIOS 6, PSP Drivers a NDK.
3. Nastudujte protokol EtherCAT, zejména EoE (Ethernet over EtherCAT) a CoE (CAN application layer over EtherCAT) a protokol Modbus.
4. Navrhněte rozšíření protokolu Modbus pro přímý 32bitový přístup do paměti regulátoru Emadyn-F tak, aby byl v souladu s normou Modbus Application Protocol v1.1b implementujte SW driver Modbus TCP a Modbus RTU v jazyce C/C++ nebo Java.
5. V jazyce C/C++ nebo Java vytvořte grafický SW pro testování komunikace pomocí Modbus protokolu.
6. V jazyce C/C++ nebo Java vytvořte grafické rozhraní pro sledování veličin, nastavování parametrů a diagnostické nástroje v regulátoru Emadyn-F, který by byl použitelný při oživování zařízení na zkušebně ČKD a u zákazníka.
7. Ověřte tento SW a to při použití Ethernet, resp. EtherCAT rozhraní.
8. Výsledky své práce zdokumentujte.

Seznam odborné literatury:

[1] Libor Dostálek, Alena Kabelová: Velký průvodce protokoly TCP/IP a systémem DNS. Aktualizované druhé vydání - Computer Press, Praha, 2000.
[2] Herbert Schildt: Java 7 Výukový kurz. Computer Press, Brno, 2012.

Vedoucí bakalářské práce: Ing. Michal Sojka, Ph.D.

Platnost zadání: do konce zimního semestru 2013/2014

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

L.S.

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 10. 1. 2013

Prohlášení:

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškeré informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 23.5.2013 ..... S. Skoupý .....

Sebastian Skoupý

Poděkování

Děkuji především firmě ČKD Elektrotechnika, a.s., ve které jsem tuto bakalářskou práci zpracovával a také mému vedoucímu, Ing. Michalu Sojkovi, PhD., že se mě ujal a v průběhu práce mi vždy pomáhal vyřešit vše, co jsem potřeboval.

Abstrakt

Tato práce se zabývá vytvořením grafické aplikace pro počítač na podporu vývoje nové generace regulačního systému Emadyn-F, vyvíjeného firmou ČKD Elektrotechnika, a.s., testováním správnosti vývoje systému. Poté se zabývá vytvořením grafické aplikace pro počítač pro sledování veličin regulačního systému Emadyn-F, změnu jeho parametrů a jeho servis v případě nefunkčnosti. Obě aplikace jsou vytvořeny v jazyce Java kvůli jeho multiplatformnost a dobré podpoře grafiky. Těžiště této bakalářské práce je tedy komunikace po síti po sériové lince a Ethernetovu spolu s vytvořením grafického rozhraní pro snadné ovládání externího zařízení – regulátoru. Na závěr se tato práce zabývá vytvořením aplikací pro ověření vytížení přenosových linek, základní nastavení regulátoru a aktualizaci jeho softwaru.

Abstract

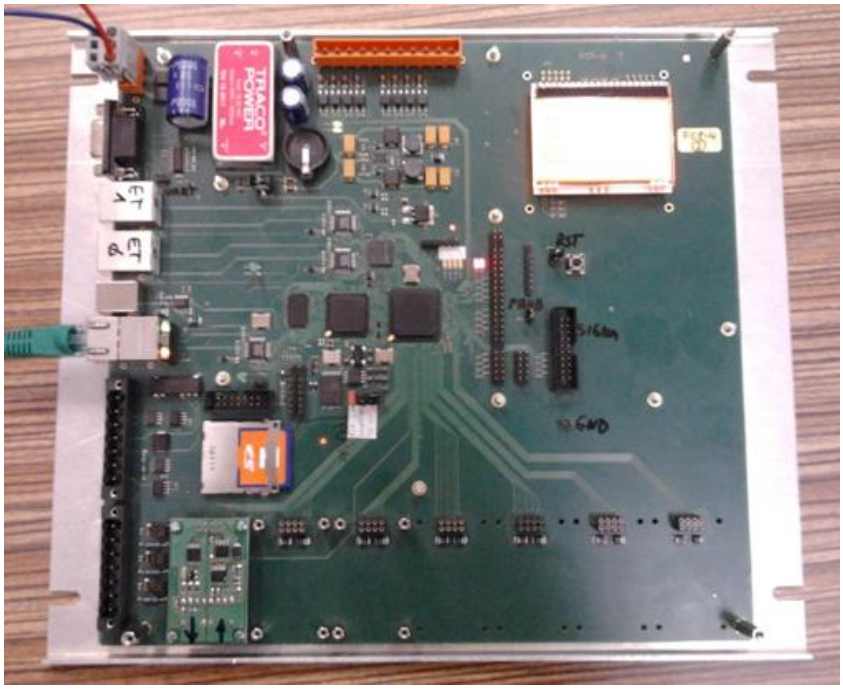
This thesis focuses on creating a graphic application to support development of a new generation of regulation system Emadyn-F, developed by ČKD Elektrotechnika, a.s. and testing the way of Emadyn-F‘s development is progressing. Then it focuses on creating another graphic application to watch variables of this regulation system, change its parameters and service in case of failure. Both application are created in Java due to its multiplatform support and its good support of graphics. The main focus of this paper is the used concept of networking via serial link and Ethernet protokol, creating a graphic user interface to provide an easy and elegant way to control the external device - the regulation system. As an add-on this paper discusses applications for verifying the extent of the usage of the

communication lines, basic configuration of the regulation system and update of its software.

Obsah

1 Úvod

Při vývoji elektronických řídích jednotek je zapotřebí vše testovat, abychom věděli, jakým směrem se náš vývoj posouvá. Asi nikdo si nedovede představit, že by vývojář připojil rozdělaný kód k letadlu a zkusil, co že vlastně kód dělá a nakonec zjistil, že ulomil křídlo kvůli chybě ve svém kódu. Testovat je možné jak hardware, tak software a právě testování se týká má práce. Cílem této práce je vývoj několika aplikací pro firmu ČKD Elektrotechnika, a.s. Všechny tyto aplikace mají za cíl sloužit hlavně pro účely vývoje nového regulačního systému Emadyn-F, který tato firma vyvíjí. Systém Emadyn se používá na řízení elektromotorů velmi vysokého napětí (až do 22kV), generátorů a filtračně kompenzačních zařízení. Používá se například v důlních zařízeních či v továrnách na kompenzaci jalové složky proudu vznikající při výrobním procesu. Moje práce se zabývá vývojem aplikací na kontrolu a nastavení softwarové (SW) části regulátoru. Hardware (HW) se ve firmě kupuje a jeho testování není pro tuto práci důležité. Aplikace popsané v této bakalářské práci jsou vytvořené v jazyce Java a mohou běžet na libovolném zařízení s Java Virtual Machine (JVM) 1.7.0_21. Deska regulátoru Emadyn-F je na obrázku 1.



Obr. 1: Deska regulátoru Emadyn-F.

Veškerý vývoj SW v této práci vycházel hlavně z minulých programů, které vytvořili jiní pracovníci firmy ČKD Elektrotechnika. Koncept aplikací popsaných v této práci zůstal podobný jejich předchůdcům, ale byl rozšířen. Tato rozšíření byla diskutována s jejich budoucími uživateli, aby splňovala jejich požadavky. Nejdůležitější bylo rozšíření o komunikaci po Ethernetu. Předchozí aplikace ji neumožňovaly, existovala u nich jen komunikace po sériové lince. Tyto předchozí aplikace jsem dostal kompletně k dispozici včetně jejich zdrojových kódů, ale protože byly psané v jazyce C++, tak jsem kód nemohl přímo použít. Hlavní pro mě bylo pochopit fungování programu.

Prvním bodem vývoje bylo vytvořit aplikaci nazvanou ModbusTest-F. ModbusTest proto, že se s ním testuje komunikace protokolem Modbus s externím zařízením – regulačním systémem Emadyn-F. F značí verzi systému. Tato aplikace se skládá z několika oken (grafické části) a komunikačních vláken, která implementují komunikaci zvoleným protokolem. Na výběr je Modbus-RTU a Modbus-TCP, ty se liší použitým přenosovým médiem.

Během vývoje ModbusTestu-F přišel ze strany zadavatele požadavek na vytvoření dvou aplikací, které nejsou v zadání bakalářské práce, ale souvisí s jejím tématem, a proto je také zmíním. První aplikace testuje, nakolik procent jsme schopni využít fyzickou vrstvu sítě. Druhá aplikace slouží k aktualizaci softwaru na regulátoru Emadyn-F. Zároveň umí nastavit paměťové a síťové parametry regulátoru, jako je například její IP či MAC adresa. V obou aplikacích jsem vytvořil základní grafické rozhraní a komunikační vlákna obsluhující provoz po síti, tentokrát po proprietárním protokolu fungujícím nad TCP.

Po dokončení těchto dvou aplikací a dokončení aplikace ModbusTest-F jsem vytvořil závěrečnou aplikaci Emvis-F. Ta má za úkol vizualizovat fungování regulačního systému Emadyn-F. Zatím je používána pouze pro simulace, protože kompletní systém Emadyn-F ještě nevznikl. Ve výsledku se bude používána pro rozběh motoru na zkušebně firmy ČKD Elektrotechnika a sledování, jestli je regulátor dobře naprogramován a chová se jak má. Tato aplikace bude sloužit k interním potřebám firmy ČKD. Jediná situace, kdy je použita mimo budovu ČKD, je, když vyrazí technik firmy ČKD do terénu kvůli řešení poruchy. V tom případě mu je aplikace nápomocna tím, že si může snadno zjistit hodnoty veličin

z regulačního systému online přes síť a nemusí si měřit jednotlivé veličiny ručně, popřípadě měřit přenášené zprávy osciloskopem a z nich zjišťovat, jaké se přenáší a zprávy proč. Aplikace se skládá z grafického rozhraní a obslužných komunikačních vláken využívajících tentokrát pouze protokol Modbus-TCP.

Práce má následující strukturu. Ve druhé kapitole popíšu systém Emadyn-F, jeho hardware, programy použité při vývoji jeho softwaru a komunikační protokoly, které využívá. V třetí kapitole specifikuji zadání aplikací popsaných v této práci a jejich implementaci. V čtvrté kapitole shrnu výsledky své práce.

2 Teoretická část

V této kapitole popíšu architekturu použitého procesoru v systému Emadyn-F. Poté popíšu jeho HW zapojení. Následovně popíšu Code Composer Studio (CCS), jakožto program, v němž je Emadyn-F programovaný, Real-Time Software Components (SW RTSC), Platform Support Products (PSP), Network Develloper's Kit(NDK) jakožto nástroje použité při vývoji SW procesoru Emadynu-F a SYS/BIOS 6 jakožto operační systém procesoru použitého na desce Emadyn-F.

2.1 Hardware systému Emadyn-F

2.1.1 Architektura procesoru TI OMAP-L137

Procesor OMAP-L137 je dvoujádrový procesor založený na ARM926EJ-S a C674xDSP jádře. Dvoujádrová architektura OMAP-L137 mu přináší výhody jak DSP, tak RISC technologií.

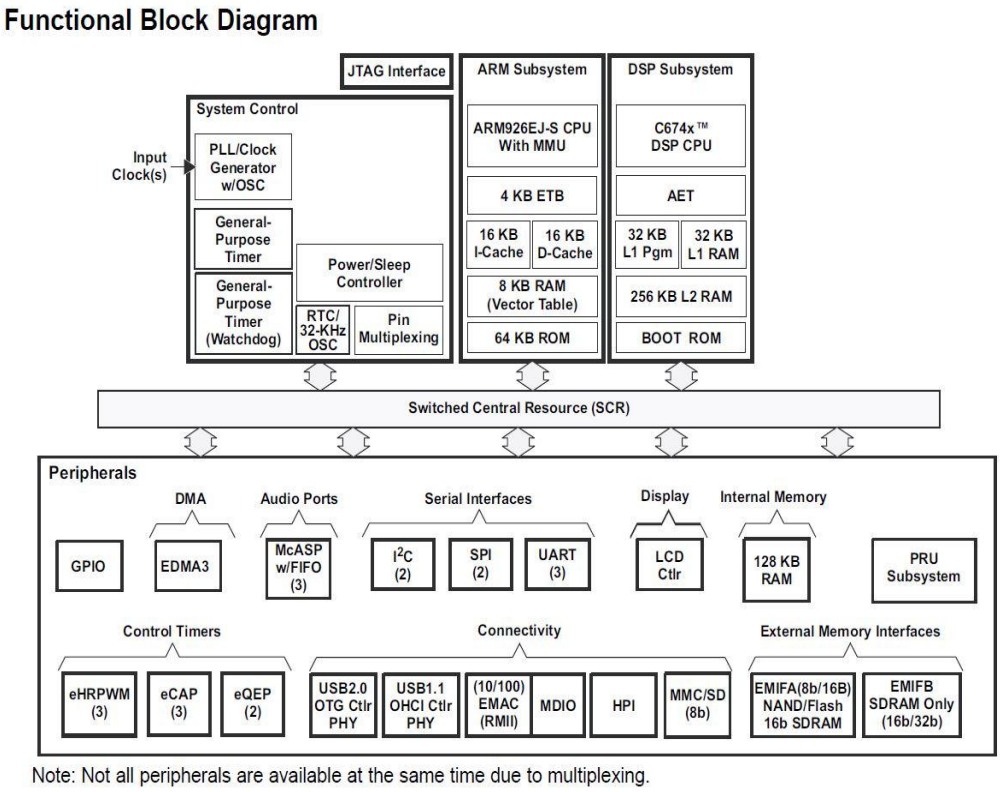
ARMové jádro je 32bitové RISC jádro. Umí provádět 32bitové nebo 16bitové operace a pracuje s 32, 16 či 8bitovými daty. Má 16kB data cache. Využívá čtyř-cestně asociativní VIVT - Virtual index virtual tag – metodu cachování, která umožňuje velmi rychlé vyhledávání obsahu cache, protože index i tag jsou virtuální a není potřeba zjišťovat z jednotky správy paměti (Memory management unit – MMU) fyzickou adresu pro danou virtuální adresu.

ARMové jádro má dále 8kB RAM a 64kB Flash.

DSP jádro využívá dvouúrovňové cachování. Programová cache první úrovně (L1P) je 32kB přímo mapovaná cache a datová cache první úrovně (L1D) je 32kB dvou -cestně asociativní cache. Programová cache druhé úrovně (L2P) je tvořena 256kB pamětí, která je sdílena mezi programem a prostorem pro data. Paměť L2 může být nakonfigurovaná jako mapovaná paměť, cache, či kombinace obojího. DSP L2 je přístupná ARMem a dalšími členy v systému, přídavná 128kB

sdílená RAM je dostupná k použití dalšími členy systému bez ovlivnění výkonu DSP.

Procesor obsahuje následující periferie: 10/100 Mb/s Ethernet MAC (EMAC) s Management Data Input/Output modulem (MDIO), tři multichannel audio serialové porty (McASP) s FIFO buffery, dva 64bitové čítače k obecnému použití, z nichž jeden lze použít jako watchdog, konfigurovatelné 16bitové host port interface (HPI), osm sad 16 pinových vstupně/výstpuních konektorů k obecnému použití (general-purpose input/output – GPIO) multiplexovaných s ostatními periferiemi, tři UART rozhraní, tři pulzně šířkové modulátory s vysokým rozlišením (enhanced High Resolution Pulse Width Modulator - eHRPWM), tři 32bitové moduly s rozšířeným záznamem (enhanced Capture Periferal - eCAP), dvě externí paměťové rozhraní - asynchronní a SDRAM - externí paměťové rozhraní (External Memory InterFace - EMIFA) pro pomalejší periferie a vysoko rychlostní paměťové rozhraní (EMIFB) pro SDRAM.



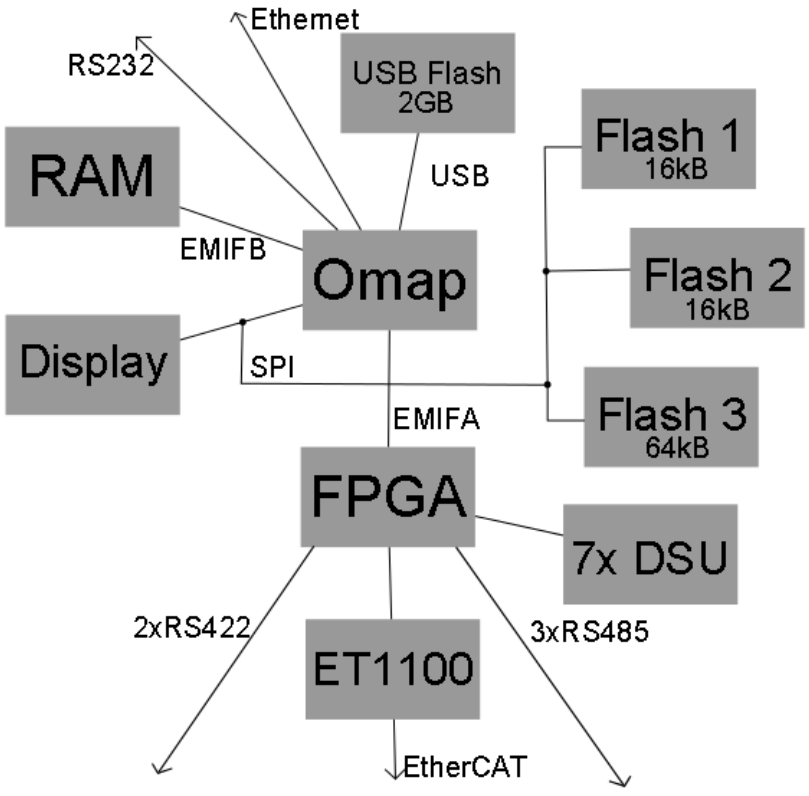
Obr. 2: Schéma procesoru OMAP-L137. Zdroj [3]

EMAC zajišťuje účinné rozhraní mezi OMAPem a sítí ve výše popsaných rychlostech ať už v half-duplex či full-duplex režimu. HPI, I2C, SPI a USB porty umožňují snadné ovládání periferií a komunikaci s dalšími procesory.

OMAP-L137 má kompletní sadu vývojových nástrojů jak pro ARM, tak pro DSP. Obsahují například C překladače, pomůcky pro DSP optimalizaci pro zjednodušení programování a úkolování či Windows ladící rozhraní pro lepší interakci vykonávání kódu.[3]

2.1.2 Hardwarové zapojení řídicího regulátoru Emadyn-F

Hardwarové zapojení regulátoru Emadyn-F jsem znázornil na obr. 3.



Obr. 3: HW zapojení řídicího regluátoru Emadyn-F

Uprostřed je procesor OMAP-L137. Po sběrnicích EMIFA a EMIFB je připojen k RAM, resp. k FPGA (hradlovému poli). Po sběrnici SPI je připojen ke třem Flash pamětím a k displeji. Přímo je napojen na Ethernet a RS232. FPGA má dva

konektory na RS422, tři konektory na RS485 a je spojeno ještě s modulem ET1100. To je EtherCAT slave komunikační modul.

2.2 Software použitý pro vývoj systému Emadyn-F

2.2.1 Code Composer Studio

Code Composer Studio (CCS) je integrované vývojové prostředí (integrated development environment - IDE) pro rodinu TI vestavěných procesorů. CCS přináší sadu nástrojů pro vývoj a laděni vestavěných aplikací. Zahrnuje kompilátory pro každou z rodin procesorů firmy TI, editor zdrojového kódu, prostředí pro tvorbu celého projektu, debugger, profiler, simulátor, real-time operační systém a mnoho dalších výhod. Prostředí je vyvinuto tak, aby uživatele navedlo k postupnému pochopení vývoje aplikací, a vychází z open source frameworku Eclipse.

Eclipse byl původně vytvořen jako otevřený framework pro vývoj dalších vývojových nástrojů. Nyní se vyvinul do podoby, kdy nabízí výjimečné prostředí pro tvorbu softwarových vývojových prostředí jako je právě CCS. Je standardem, že právě Eclipse je framework použitý většinou prodejců vestavěných softwarů. CCS rozšířil Eclipse hlavně o Texas Instruments pokročilé ladící schopnosti.

CCS běží jak na operačním systému Windows, tak na Linux, avšak některé jeho vlastnosti nejsou na Linuxu dostupné. [4]

2.2.2 Real-Time Software Components

Real-Time Software Components (RTSC) je projekt fungující pod záštitou Eclipse. RTSC je sada komponent, která poskytuje základní nástroje a nízko úrovňové runtime prostředí k vývoji programů pro tzv. Component-based programming (koncept programování založený na separaci programů do jednotlivých komponent, každá je nezávislá na ostatních). Je založen na jazyce C a je možné ho použít pro všechny druhy vestavěných zařízení. Hlavní výhodou RTSC

oproti podobným programům je možnost jeho využití i pro prostředkově omezené systémy jako je například DSP či 16bitový mikrokontroler. Hlavní vlastnost programů vyvinutých pomocí RTSC je vysoce optimalizovaný C/C++ kód. Výhodou je, že prakticky není potřeba nahrávat žádné prostředky na zařízení, aby RTSC programy fungovaly. Každá komponenta RTSC navíc obsahuje JavaScriptový kód, který běží jak v dané komponentě, tak i na počítači, abychom mohli sledovat průběh vykonávání instrukcí v aplikaci. [5]

2.2.3 SYS/BIOS 6

SYS/BIOS je pokročilý, operační systém pracující v reálném čase. Je používaný v mnoha DSP, ARM mikroprocesorech a mikrokontrolerech firmy TI. Je také použit v procesoru OMAP-L137. Byl vyvinut pro použití ve vestavěných aplikací. Nabízí preemptivní multitasking, hardwarovou abstrakci (rozhraní pro jednotné ovládání různě fungujícího hardware) a správu paměti (memory management). [6]

2.2.4 Platform Support Products

Platform Support Products je sada nástrojů vyvinutá firmou Texas Instruments. PSP je vyvíjeno za účelem poskytnutí připojení k periferiím a driverů pro zařízení operačnímu systému. Obsahuje ukázkové aplikace na nastartování vývoje tzv. TI SOC (System on a Chip) platforem jejich zákazníky. Díky němu zákazníci TI nemusí vytvářet všechna základní rozhraní, ale již mají porty a drivery připravené k použití a mohou se soustředit na samotnou aplikaci než na vývoj obecných věcí, které by si jinak každý vývojář musel vytvářet sám. Tím se samozřejmě urychlí každý vývoj. [7]

2.2.5 Network Developer's Kit

NDK je další vývojový program od firmy TI. Je to platforma sloužící pro vývoj a demonstraci síťových aplikací ve vestavěných zařízeních v procesorech TI.

Momentálně je kompatibilní s ARM procesory a jen s některými DSP procesory. Kód v NDK je generický kód v jazyce C. NDK slouží jako rychlá prototypová platforma pro vývoj aplikací zpracovávajících sítě a pakety. Může být použit pro přidání připojení k síti do existujících aplikací, a tím jim přidat možnost nastavitelnosti a ovládání. Díky využití modulů NDK může vývojář rychle přejít od vývojového návrhu k pracující implementaci kódu. NDK pracuje na SYS/BIOS Real-Time Operating Systém (RTOS) a může být přenesen na mnoho druhů hardwaru od TI, je vyvíjen jako dobře oddělitelná nástavba k SYS/BIOS a vývojových nástrojům CCS. [8]

2.3 Komunikační protokoly používané při komunikaci se systémem Emadyn-F

2.3.1 EtherCAT

Ethernet for Control Automation Technology je vysokorychlostní sběrnicový systém založený na Ethernetu. EtherCAT a Ethernet protokol specifikují linkovou vrstvu. Cílem vývoje EtherCATu bylo umožnit použití Ethernetu pro řídící aplikace, které potřebovaly rychlé aktualizace dat s malým komunikačním jitterem při nízké ceně hardwaru (jitter je nežádoucí odchylka od periodického signálu v elektrických zařízeních).

Zařízení v sítích pro průmyslovou automatizaci jsou charakteristická velmi krátkou délkou dat, které si uzly vyměňují. Ta je často menší, než je minimální délka Ethernetového rámce. Pokud by se tedy neudělala žádná změna, tak by po síti bylo vysíláno mnoho neužitečného obsahu, jak by byly rámce data doplňovány bity postrádajícími významu, aby dosáhly minimální délky rámce pro Ethernet. Proto po EtherCATu již není vysílán klasický Ethernetový rámec. Uzle jsou řazeny do topologie kruhu. Každý uzel čte jemu adresovaná data během toho, co jím rámec prochází. Během toho též stihne nahrát svá data do tohoto rámce. Díky tomu je rámec procházející uzlem zpožděn jen o mikrosekundy (zpoždění je velmi důležitá

veličina v řídicích systémech) a rovnou pokračuje k dalšímu uzlu. Velmi často je takto celá síť adresována jediným rámcem. [9]

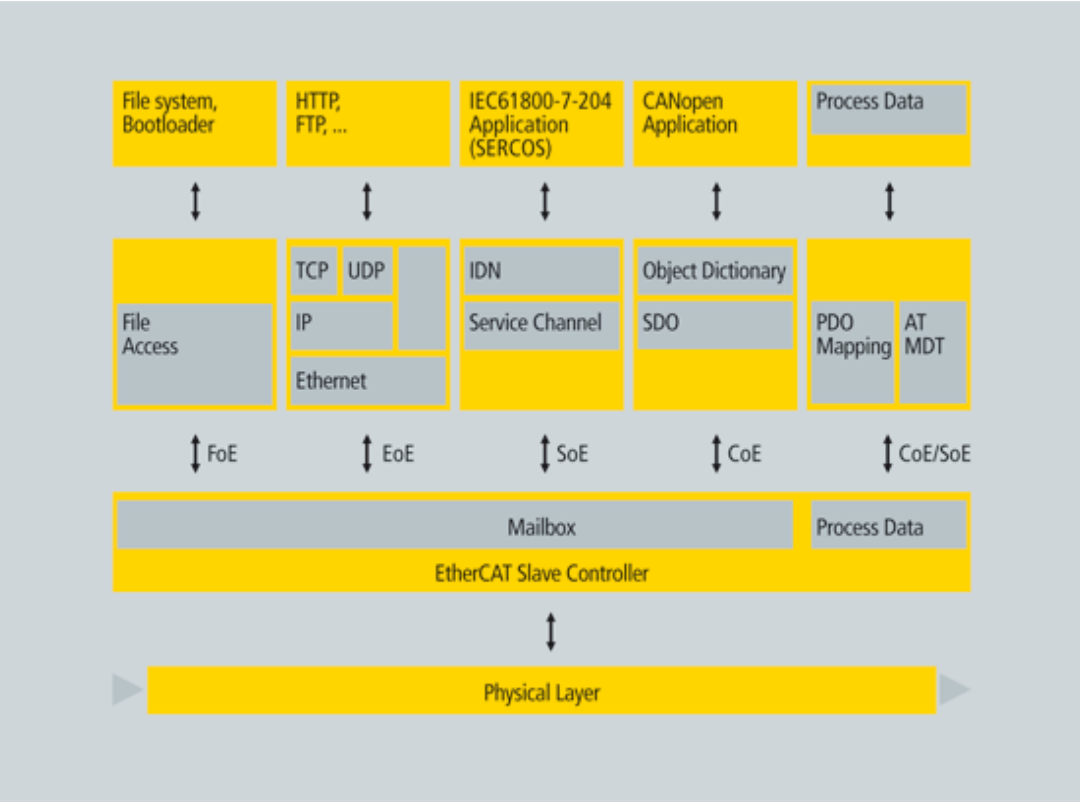
2.3.2 Ethernet over EtherCAT (EoE)

Technologie EtherCAT je kompatibilní s technologií Ethernet a navíc je hodně otevřená k jejich propojení. Umožňuje použití mnoha služeb a protokolů založených na Ethernetu, aniž by se to nějak projevilo na jeho výkonu. Nejsou žádná omezení na druh Ethernetového zařízení, které může být připojeno do EtherCATu přes port přepínače (switch). Ethernetové rámce jsou tunelované skrz EtherCATový protokol. EtherCATová síť je plně transparentní pro Ethernetová zařízení. EtherCATové zařízení může navíc implementovat další Ethernetové protokoly, a tím se chovat jako klasické Ethernetové zařízení. Síť funguje principu master-slave. Master je řídící uzel, slave jsou všechny ostatní. Řídící uzel dle dat získaných od ostatních uzlů rozhoduje, co mají jednotlivé uzly vykonávat. Tyto uzly vykonávají jeho příkazy a posílají mu informace o svém stavu. Master se může chovat jako dvouvrstvý přepínač a může přesměrovávat rámce patřičnému zařízení na základě adresy. Tím pádem všechny internetové technologie, jako jsou např. e-mail, či FTP, mohou být používány v EtherCATové síti. [9]

2.3.3 CANopen over EtherCAT (CoE)

CANopen je protokol aplikační vrstvy vyvinutý původně pro sběrnicí CAN. Je používán v zařízeních a aplikacích dostupných v nejrůznějších oblastech, od vstupně/výstupních zařízení, přes enkodéry, ventily, ovladače hydrauliky, až po další výrobní zařízení. EtherCAT může poskytovat stejné komunikační mechanismy jako CANopen: Object Dictionary, PDO (process data objects) a SDO (service data objects) za porovnatelných nákladů na provoz. EtherCAT tudíž může být implementovaný s minimálním úsilím do zařízení vybavených CANopenem. Navíc velká část CANopen firmware může být při této implementaci použita, např. objekty mohou být rozšířeny za účelem využití větší šířky pásma (bandwidth), kterou EtherCAT nabízí. [9]

Na obr. 4 je ukázka protokolů fungujících po EtherCATu.



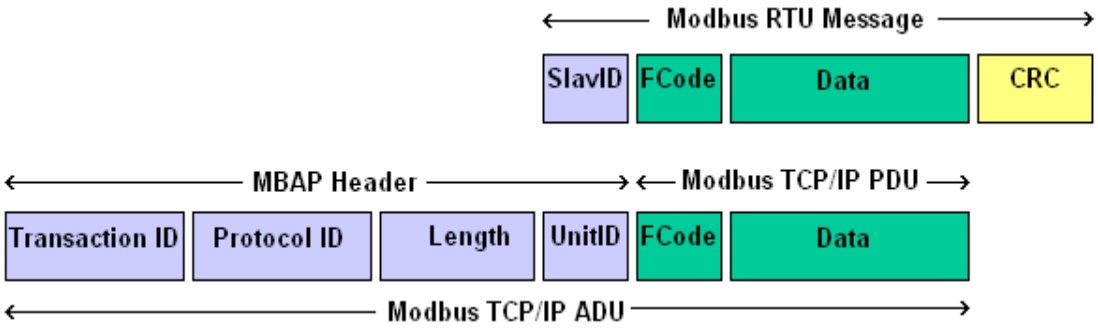
Obr. 4: Ukázka protokolů fungujících po EtherCATu. Zdroj: [9]

2.3.4 Modbus

Protokol Modbus je pro tuto práci nejdůležitější a proto se o něm rozepíšu nejvíce. Je to protokol linkové vrstvy. Modbus je otevřený protokol vyvinutý firmou Modicon v roce 1979 pro komunikaci s PLC (programmable logic controller). Díky jeho jednoduchosti a robustnosti se stal standardním komunikačním protokolem a lze ho dnes běžně najít vedle protokolů jako je InterBus či ProfiBus. Hlavní důvod pro jeho použití je, že byl vyvíjen s cílem na použití v průmyslových aplikacích a je pro ně přizpůsoben, je otevřený, je bez poplatků za použití a je velmi snadný na spuštění. Dnes se o vývoj protokolu stará Modbus Organization.

Protokol se dělí na tři druhy podle použité fyzické vrstvy – Modbus-TCP pro Ethernetový kabel, Modbus pro sériovou linku (RS-485 apod.) a Modbus PLUS. O Modbusu PLUS se nebudu zmiňovat, protože se u zadavatele nepoužívá a není tématem mé práce. Obecně je na sběrnici jeden master a více slave zařízení, pro Modbus-TCP může být na síti teoreticky více masterů. [10]

Modbus protokol je definován typem uspořádání zpráv. Zprávy mají hlavičku a data. Na obrázku 5 je v prvním řádku zobrazena zpráva v Modbusu-RTU a na druhém řádku lze vidět, jak vypadá zpráva v Modbusu-TCP.



Obr. 5: Zpráva protokolu Modbus. Zdroj: [11]

Jak je vidět, tak jsou samozřejmě podobné. Modbus-RTU je tzv. headless - má výrazně menší hlavičku a je následován kontrolním součtem (CRC), Modbus-TCP má hlavičku delší a CRC nemá, jelikož využívá všechny výhody TCP protokolu, mezi které patří jistota, že pokud zpráva dorazí, tak dorazila správně. Proto není CRC potřeba posílat. Při zkoumání paketu v TCP komunikaci bychom viděli nejdříve TCP hlavičku, za kterou by následovali data. První data by byla právě hlavička Modbusu-TCP. Celá zpráva v protokolu Modbus má následující posloupnost a význam:

Transaction ID je 2bytová hodnota, označující číslo zprávy. Podle toho si lze držet přehled o tom, který dotaz a která odpověď k sobě patří. V protokolu TCP totiž není zaručeno stejné pořadí příchodu odpovědí ve kterém jsme zaslali dotazy.

Protocol ID je 2bytová hodnota nastavena na 0 pro Modbus-TCP.

Length je 2bytová hodnota a označuje, kolik za ní následuje ještě bitů - tj. součet UnitID, FCode a Data.

SlavID nahoře resp. UnitID dole je jedna a ta samá věc označující adresu stanice na sběrnici. Je to jednobytová hodnota a určité hodnoty mají speciální smysl. Hodnota 0 určuje multicast a jednotlivá zařízení na ni nemají odpovídat. Hodnoty 1 až 247 jsou unicastové a po zpracování na ně zařízení odpovídají. Hodnoty 248 až 255 jsou rezervované, hodnota 255 označuje, že se tento byte nepoužívá.

FCode, zkratka pro Function Code, je jednobytová hodnota označující o jakou instrukci se jedná. Existují instrukce definované normou protokolu Modbus, instrukce volné (ty si každý může nadefinovat, jak potřebuje) a instrukce rezervované, které by nikdo používat neměl.

Na konec jsou zařazena Data a CRC.

Pro Modbus-RTU je jako pro každý jiný přenos po sériové lince synchronizační okno. To znamená, že před každou a po každé zprávě je určitý počet povinných bitů rovných nule, aby se před každou zprávou zesynchronizovaly slave a master a zprávy nebyly slavem špatně interpretovány.

Modbus protokol využívá Big-Endian pořadí, tj. MSB (most significant byte) je první.

Ve mnou vyvinutých programech pracuji jen s Modbus protokolem ve formě Modbus-TCP a Modbus-RTU. Z počítače jsem připojen buď přes COM port, nebo přes Ethernetový port. Jestli zprávy poté jdou přes nějaký převodník, či jestli je protokol tunelovaný přes jiný protokol, není pro můj program důležité, a proto se detekcí ani nezabývá.

V ČKD Elektrotechnika se v protokolu Modbus kromě klasických funkčních kódů používají i vlastní funkční kódy. V předchozí verzi regulačního systému (Emadyn-D) se používal funkční kód pro přímý 32bitový přístup do paměti regulátoru.

3 Vyvinuté aplikace

V této kapitole specifikuji zadání vyvinutých aplikací, zvolený programovací jazyk, principy použité při tvorbě zadaných aplikací a jejich základy ovládání.

3.1 Specifikace zadání vyvinutých aplikací

V jazyce C/C++ nebo Java vytvořte grafickou aplikaci pro testování komunikace s regulátorem Emadyn-F pomocí Modbus protokolu.

V jazyce C/C++ nebo Java vytvořte grafickou aplikaci pro sledování veličin, nastavování parametrů a diagnostické nástroje v regulátoru Emadyn-F, která by byla použitelná při oživování zařízení na zkušebně ČKD a u zákazníka.

3.2 Rozbor vyvinutých aplikací

3.2.1 Výběr programovacího jazyka

Jako první věc jsem musel zvolit, v jakém jazyce budu programy vytvářet. Z mého hlediska se nabízely dvě možnosti – jazyk C++ a jazyk Java. Pro C++ hovořilo to, že jsem měl k dispozici předchozí programy s jejich zdrojovými kódy a mohl bych z nich vycházet, ale proti hovořilo to, že jsem grafické uživatelské rozhraní v C++ nikdy nedělal. Proto jsem vybral jazyk Java. Java je multiplatformní a programy v ní napsané lze spustit na jakémkoliv zařízení s dostatečně novou Java Virtual Machine (JVM) bez další kompilace, v době vývoje to byla verze 1.7.0_21 firmy Oracle. Jazyky jako C# by se sice možná vyjímaly hezky na Windows, ale protože je v plánu přenést všechny tyto aplikace i na Linux, tak nepřicházely v úvahu.

3.2.2 Rozšíření protokolu Modbus pro přímý 32bitový přístup do pamětí regulátoru

Původně bylo v plánu implementovat 32bitový přístup do paměti regulátoru, protože tato funkce byla podporována u předchozího regulátoru. Tento bod jsem nerealizoval, protože od této myšlenky se ve firmě upustilo. To je dáno rozdíly v architektuře procesoru v Emadynu-D a Emadynu-F. Toto rozšíření není zapotřebí a navíc se vedení vývoje dohodlo, že by to v této verzi regulačního systému ani nebylo vhodné. Místo toho vzniklo jiné rozšíření, které přistupuje jen k vybraným oblastem paměti. Detailně ho popíšu při popisu programu Emvis-F, který ho využívá.

3.3 Vyvinuté aplikace

3.3.1 Aplikace ModbusTest-F

Popis aplikace

Aplikací, kterou budu popisovat, je aplikace na testování komunikace s regulátorem pomocí protokolu Modbus. Tato aplikace se jmenuje ModbusTest-F. Tento program má za úkol testovat funkčnost přenosu dat pomocí protokolu Modbus mezi počítačem a regulačním systémem Emadyn-F. Vesměs jde o program na zapisování a vyčítání hodnot z určitých datových adres na desce systému Emadyn-F. Má otestovat, jestli si systém nechá zapsat hodnoty do Read-only oblastí či jestli dlouhodobé vytížení nezpůsobí nějakou chybu. Zároveň může zjišťovat, jestli po zapsání dané hodnoty je vše správně uloženo a podobně. Program jako takový neřeší chybová hlášení desky Emadyn-F. Předpokládá, že uživatel ví, jaké zařízení chce testovat, a jaká data mu chce poslat. Proto je tato část vynechána.

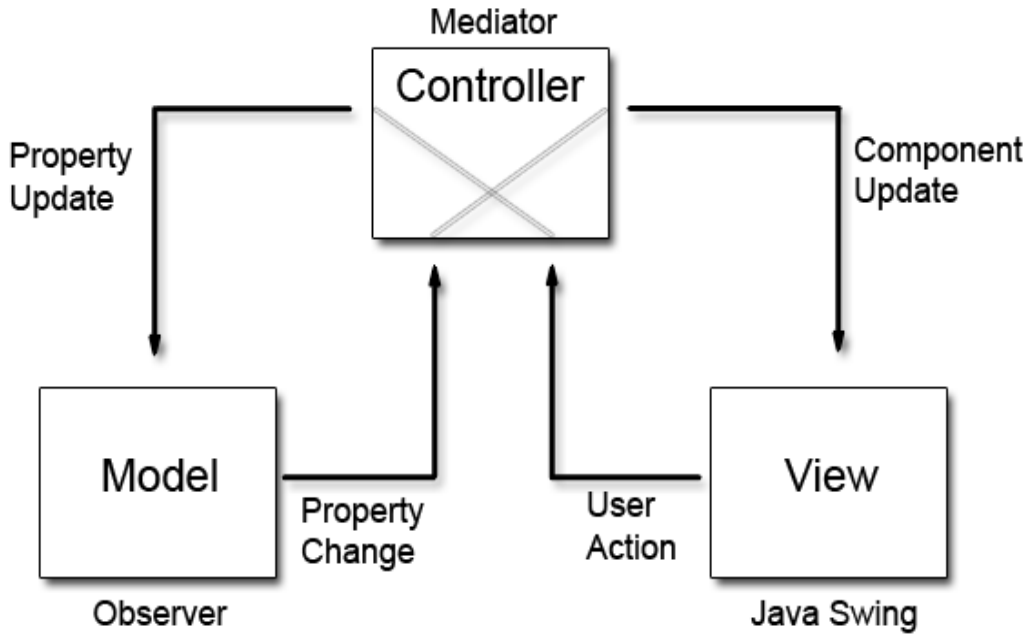
Aplikaci ModbusTest-F jsem vyvíjel na základě jejího předchůdce ModbusTest-D. Inspiroval jsem se jeho grafikou. Je v nich však velký rozdíl, a to že ModbusTest-D nepodporuje testování komunikace po protokolu Modbus-TCP, ale

jen Modbus-RTU. Modbus-TCP funguje po protokolu Ethernet včetně jeho přenosových médii. Modbus-RTU funguje po sériové lince, např. po RS-485.

Model aplikace

Při tvorbě aplikace ModbusTest-F jsem se snažil držet zásad objektového programování (Object Oriented Programming OOP). Jako model programu jsem si vybral návrhový vzor Model-View Controller (MVC).

Model-view-controller je softwarový vzor oddělující reprezentaci informací od uživatelské interakce s nimi. Model se skládá z aplikačních dat, pravidel pro nakládání s nimi, logiky a různých funkcí. View může být jakákoliv reprezentace výstupních dat, např. tabulka či graf. Controller zpracovává vstup – převádí ho na příkazy pro model či pro view. Hlavní myšlenkou použití MVC je oddělení oblastí zájmu v rámci programu a opakovaná využitelnost kódu. Schéma na obrázku 6 naznačuje tuto činnost.



Obr. 6: MVC model. Zdroj: [13]

Existují obdoby toho modelu, nemusíme využít všechny tři jeho složky, ale já jsem se rozhodl držet této struktury.

Typická akce může vypadat následovně: Uživatel vidí otevřené okno, to je složka View pravděpodobně vyplněna nějakými daty z Modelu. Klikne na tlačítko Details. Tlačítko Details je komponenta View a jeho stisknutí zachytí Controller. Controller se zeptá Modelu, jakou hodnotu mají data v Details a až získá odpověď, tak řekne View ať vykreslí nové okno s danými daty. Uživatel vidí jen své kliknutí a nové okno, ale programátorovi to pomáhá utřídit si, kam co patří, a díky tomu má program lepší strukturu, je lépe čitelný pro jiné programátory a snáze se v něm hledají chyby.

Vlastní implementace aplikace

Nyní bych popsal moji implementaci MVC modelu v programu ModbusTest-F.

Část M, Model, zahrnuje dvě datové struktury. Jednu jsem nazval Connection Data a druhou jsem nazval Message Data. Jak napovídají názvy, tak v první struktuře se drží veškerá data týkající se připojení, jako je např. IP adresa, použitý TCP Port a podobně. V druhé struktuře se drží veškerá data o nastavení zpráv, které zasíláme regulátoru. Těch je deset, skládají se z informace co je to za zprávu a jaká data obsahuje. Ke každé zprávě se zároveň drží v paměti poslední odpověď na tuto zprávu. Při spuštění aplikace se ze souboru „config.prop“ uloženého ve složce se spouštěcím souborem načtou celá Connetion Data a část Message Data vyjma odpovědí (ty se při vypnutí programu neukládají). Při zavření aplikace se tento soubor přepíše na nastavení při jeho zavření. Pokud tento soubor neexistuje, tak se vytvoří s obecnými defaultními hodnotami. Je to implementace knihovny Properties. Zároveň je možné tento soubor editovat v jakémkoliv textovém editoru bez spuštění samotného programu. Model M je singleton (v jednu chvíli může existovat pouze jediná instance, pokud uživatel chce vytvořit druhou, tak je návratová hodnota první instance), abych zaručil jeho jedinečnost.

Část V, View, zahrnuje sadu grafických oken. Ty se vykreslí v případě pokynu od Controlleru a data v nich zobrazená odpovídají datům, co jim zaslal Controller. Grafika v jazyce Java může mít určité styly. Já jsem zvolil styl Windows.

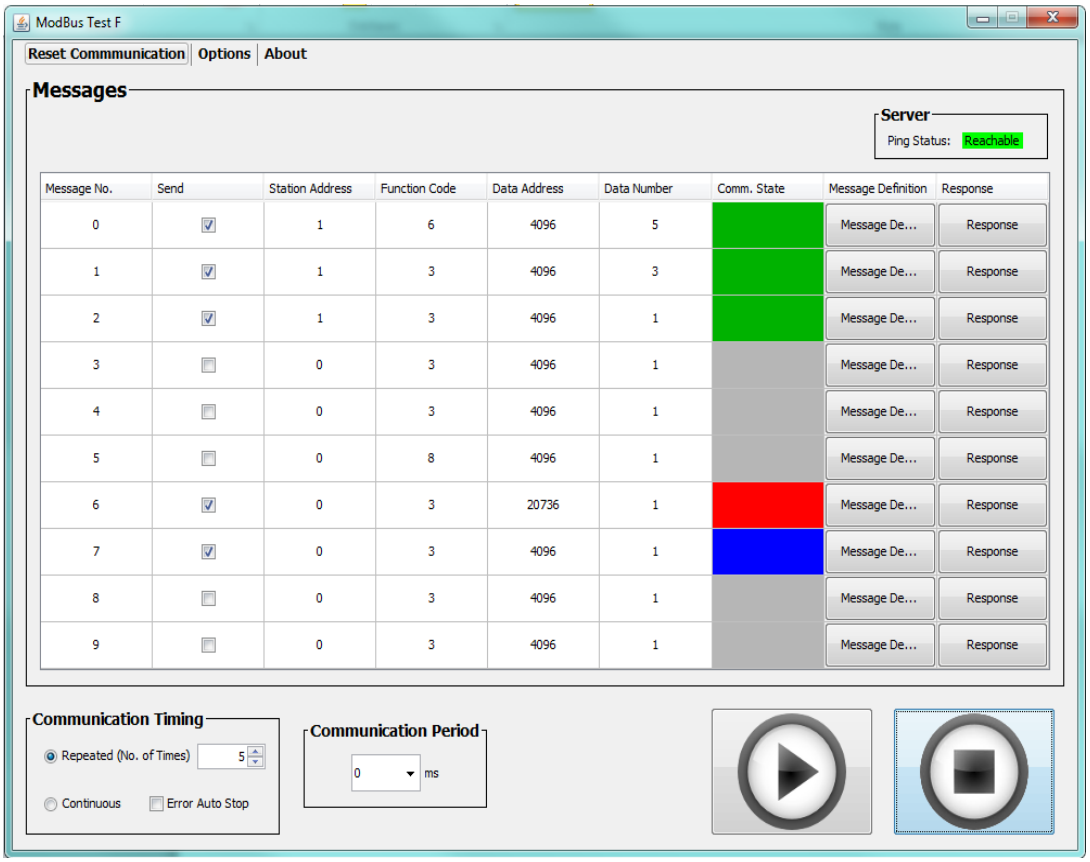
Vybral jsem ho proto, aby vypadal co nejvíce přirozeně na spuštěném počítači (zatím jsou to pouze počítače s Windows). Toto se v Javě nastavuje pomocí tzv. „Look and Feel“.

Část C, Controller, zahrnuje veškerou interakci programu. Vytváří spojení mezi M a V, dále také zahrnuje síťové služby pro komunikaci s deskou Emadyn-F. Zahrnuje také jednu třídu pro vytvoření konfiguračního souboru pro uložení uživatelského nastavení.

ModbusTest-F je samostatně spustitelný program distribuovaný ve formátu jar - java archive. Při poklepu na ModbusTest.jar se v Java Runtime Enviroment (JRE) spustí zkompilovaný program. Tento jar soubor jsem vytvořil v prostředí Netbeans. Program Netbeans kompilaci sám podporuje a nijak jsem ji nestudoval.

Program ModbusTest-F se skládá z několika oken. Okna jsem vytvořil v Netbeans designeru. Designer se spustí, když při vytváření nového souboru místo souboru typu class vybere vytvoření souboru typu form. Designer funguje na tzv. „drag and drop“ stylu (uchop a upusť), kdy na začátku máme jen šedé prázdné okno a vybíráme z java.awt komponent (ty jsou starší a já je osobně raději nepoužívám, protože někdy fungují jinak, než si představuji) a z java.swing komponent. Ty jsou modernější a nabízí více komponent. Mezi ně patří klasické tabulky, tlačítka, comboboxy a další. V tomto grafickém designeru se vytvoří zamčený kód, který nelze upravovat ručně, ale za to ho lze upravovat nepřímo přes vlastnosti v nabídce po kliknutí pravým tlačítkem myši na kteroukoli kompomentu. Respektive kód není zamčený celý, jenom část generující grafiku. Metody operující nad tlačítky (actionlistenery) apod. jsou editovatelné, jinak by designer pořádně nešel použít. Toto rozhraní je velmi dobře vyvinuté a rád ho používám. Všechny komponenty se generují do paralelních a sekvenčních skupin. Generovaný kód je rozdělen na dvě části, v první je vodorovný popis rozmístění všech komponent, v druhé je svislý popis tohoto rozmístění.

První okno, které se na uživatele otevře, je hlavní okno, ze kterého se ovládá celý program. Můžete ho vidět na obrázku 7:



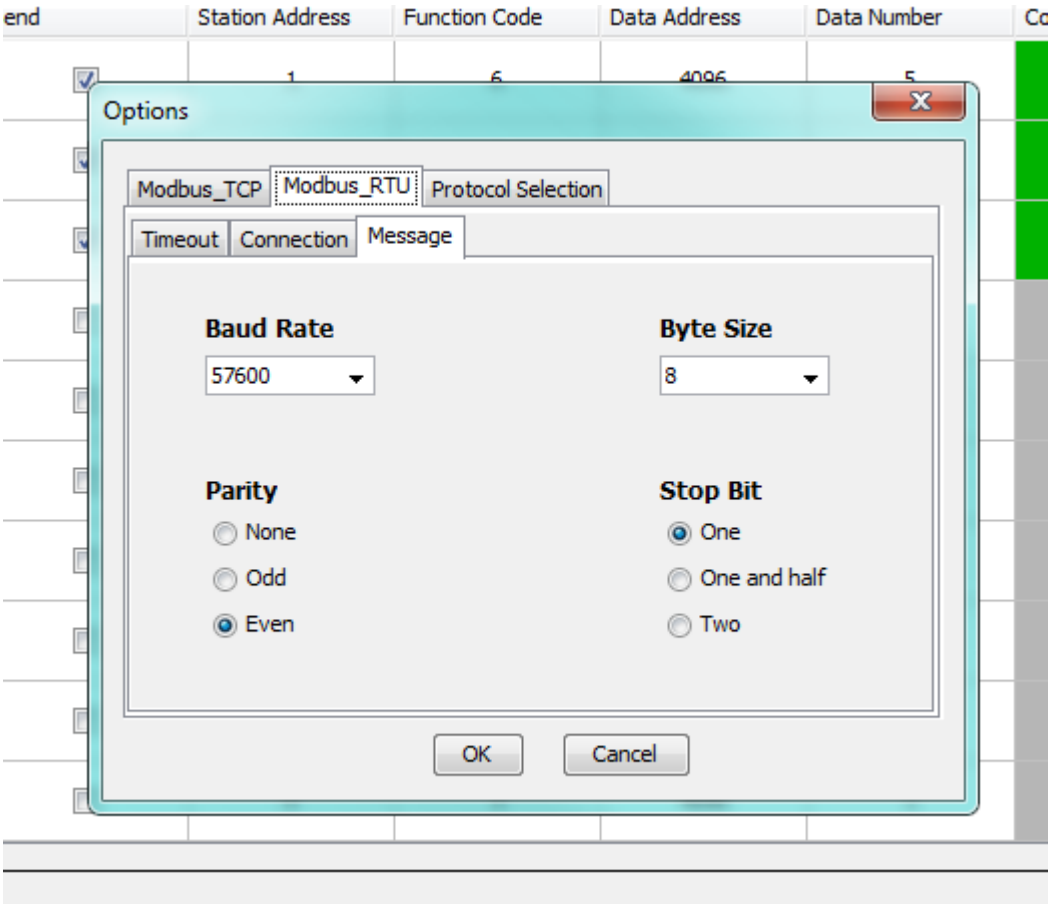
Obr. 7: Hlavní okno programu ModbusTest-F.

Hlavní částí okna je přehled všech deseti zpráv, které si uživatel nastavil na zasílání při testování externího zařízení. K vidění je celá hlavička každé zprávy, data zpráv zobrazena nejsou. Kromě hlavičky jsou zde čtyři sloupce navíc. Druhý sloupec zleva je sloupec Check Boxů. Ty slouží k výběru, jestli tuto zprávu chci přenášet při komunikaci či ne. Tato volba je dobrá k tomu, pokud jsem nastavil nějakou zprávu a momentálně ji nechci posílat, ale zároveň nechci přijít o její nastavení. Sedmý v pořadí je sloupec barevných oknének signalizuji stav přenosu jednotlivých zpráv. Šedá signalizuje, že tato zpráva není přenášena. Zelená barva ukazuje úspěšný přenos zprávy (ten se pozná podle správné odpovědi desky), modrá zpráva ukazuje, že přenos je v procesu a červená zpráva ukazuje neúspěšný přenos. Následují dva sloupce: Sloupec Message Definition – sloupec obsahující tlačítka otevírající nové okno, ve kterém lze nastavit všechny parametry a data zpráv, a sloupec Response – sloupec obsahující také tlačítka, která tentokrát otevírají okna s odpověďmi na jednotlivé zprávy.

Vpravo nahoře jsou tři tlačítka:

Tlačítko Reset Communication přeruší a znovu naváže spojení s deskou regulačního systému. Toto tlačítko vymaže z paměti programu informace o spojení s deskou a vytvoří nové podle stávající konfigurace. Je vhodné ho použít, pokud se deska zachová nějak neočekávaně a chceme začít od začátku.

Tlačítko Options otevírá nové okno s nastavením týkajícím se spojení s externí deskou. Můžete si ho prohlédnout na obrázku 8:



Obr. 8: Okno Options programu ModbusTest-F.

Obsahuje tři hlavní karty: Modbus_TCP, Modbus_RTU a Protocol Selection.

První karta zleva je karta Modbus_TCP. Pro spojení pomocí TCP se nastavuje jen IP adresa, port a timeout.

Druhá karta je karta Modbus_RTU. Ta obsahuje další tři podkarty. Na první se opět nastavují různé timeouty, na druhé se volí COMport použitý pro navázání spojení a komunikaci s deskou a volí se typ použitého kabelu. Na třetí a poslední podkartě (ta

je otevřena na obr. 8) je nastavení parametrů zprávy. Lze nastavit kolik baudů se má přenášet za sekundu, jak velký je byte, jaká parita se používá a jaká musí být odmlka mezi zprávami.

Třetí je karta Protocol Selection. Na té je pouze jeden přepínač, pomocí kterého si uživatel zvolí, jestli bude využívat Modbus_TCP či Modbus_RTU.

Kromě těchto karet okno obsahuje dvě tlačítka – Save a Cancel. Oba zavírají toto okno. Save uloží změny před zavřením a Cancel změny neuloží. Po zavření okna tlačítkem Save se automaticky volá obslužná metoda pro tlačítko Reset Communication popsané dříve, aby se nové změny uvedly rovnou v praxi.

Okno Options je modální. Neumožňuje vícenásobné otevření. To by mohlo způsobit nekonzistenci dat při připojení a to je nebezpečné.

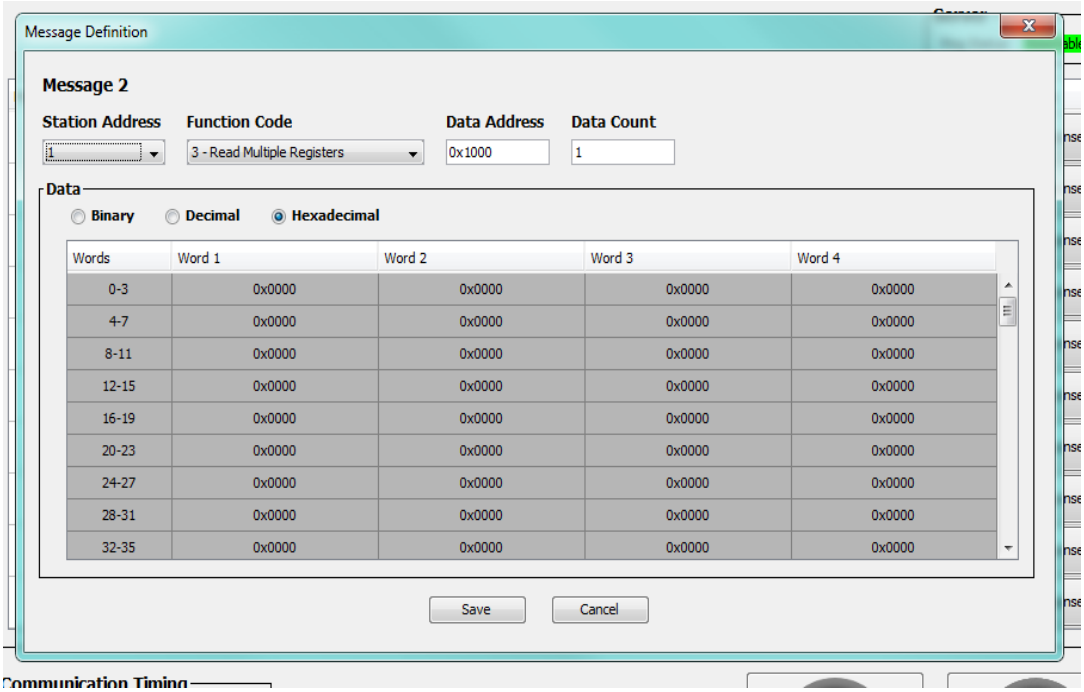
Tlačítko About otevírá malé okno s logem firmy, popisem tohoto programu a zmínkou o autorských právech.

O trochu níže napravo je kolonka značící dostupnost námi vybraného serveru. Za nápisem „Ping Status:“ je buď zeleně napsáno Reachable či červeně napsáno Not Reachable (dostupný/nedostupný). To má smysl jen při komunikaci prostřednictvím TCP, protože služba Echo není definována při použití sériové linky. Zajímavé je, že klasické Java funkce nefungovaly správně. Často hlásily nedostupnost serveru, který jsem byl schopen najít v příkazové řádce. Řešením je volat tento dotaz z programu přes příkazovou řádku. To se dělá vytvořením instance třídy Process, konkrétně takto:

java.lang.Runtime.getRuntime().exec(PŘÍKAZ)

Zde je trochu problém s multiplatformností, časem přidám řešení pro patřičné příkazy v příkazové řádce na základě používaného operačního systému. S tímto řešením jsem strávil i poměrně dost času, protože při klasickém zavolání se ve Windows Echo provede 4x. To stojí zbytečně mnoho času, programu stačí odpověď jednou. Ve Windows existuje parametr n. Použití parametru „-n 1“ (number) zaručí provedení příkazu Echo jen jednou.

Na konci zleva jsou v tabulce dva sloupce tlačítek. První z nich otevírá okno na obrázku 9:



Obr. 9: Okno Message Definition programu ModbusTest-F

Okno obsahuje číslo zprávy, kterou upravujeme, nastavení Station Address, Funcion Code, Data Address, Data Count a samotná Data. U dat ještě můžeme volit druh zobrazení, což působí i na Data Address.

Station Address je JCombobox s hodnotami 0 až 247.

Function Code je JCombobox s několika hodnotami odpovídajícími podporovaným funkčním kódům. Při implementaci tohoto Comboboxu jsem si musel dát pozor. Pro větší uživatelskou přívětivost kromě čísla funkčního kódu zobrazuje i slovní popis tohoto funkčního kódu. Tento slovní popis je ale potřeba oddělit při ukládání, protože do Modelu se ukládá jen číslo zvoleného FC. To jsem udělal pomocí metody String.regex podle mezery, z níž jsem vzal hodnotu pouze prvního pole.

Data Address je instance JTextArea a označuje místo v paměti desky Emadyn-F, se kterým chceme pracovat. Pro robustnost programu jsem implementoval vlastní Document, který jsem nastavil pro tuto JTextAreu. Ten povoluje pouze čísla a písmeno x (možnost decimálního a hexadecimálního zápisu), jiná nedovolí zapsat. Stále lze zapsat nesmysl stylu 0xxx1, ale předpokládám, že uživatel nemá v plánu shodit program. Navíc v takovém případě vyskočí exception viditelná pouze v Netbeans a tato hodnota se neuloží. Ačkoliv jsem to nezmínil výše, tak takto ošetřuji

všechny upravovatelná pole, se kterými může uživatel pracovat. Přišlo mi, že zde je to nejnázornější.

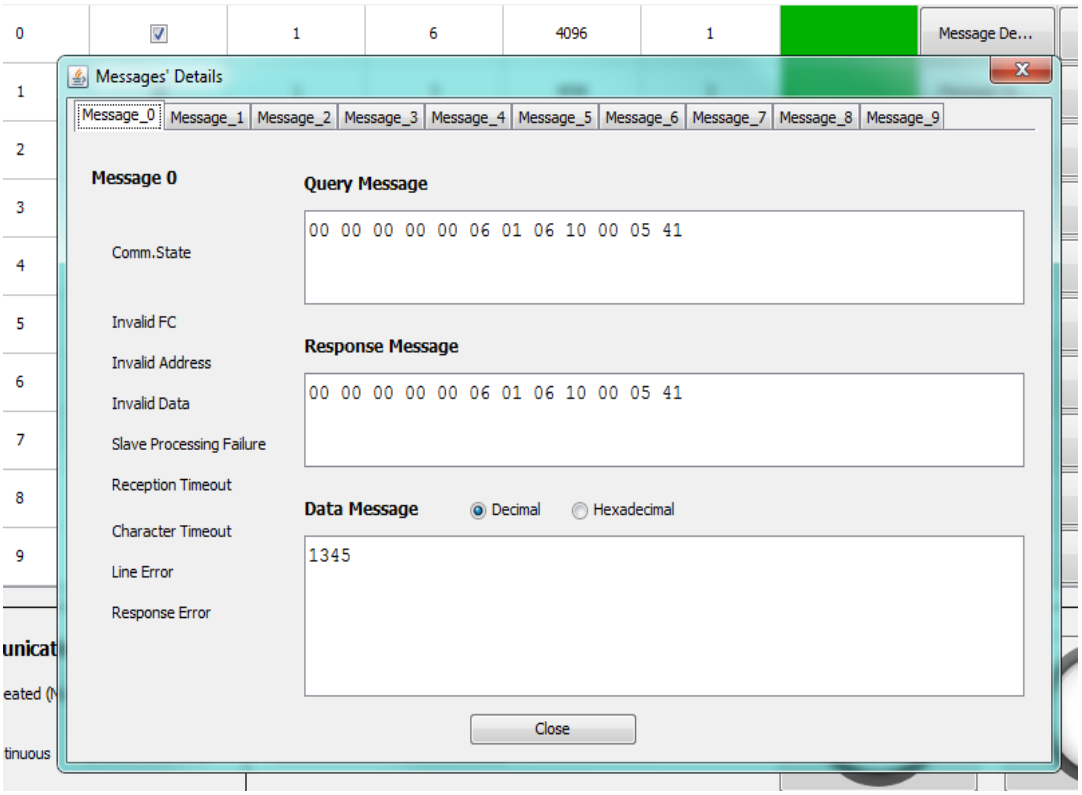
Data Count je počet wordů (16b), se kterými chceme pracovat.

Tabulka s daty se upravovat dá či nedá podle toho, jestli chceme číst, či psát (při čtení jsou data zprávy irelevantní).

Tlačítka Save a Cancel fungují stejně, jako v předchozím okně, jen se nevolá Reset Communication.

Toto okno není modální, pokud by chtěl uživatel jednu zprávu udělat na základě druhé.

Druhý sloupec tlačítek otevírá okno s odpověďmi na zaslané zprávy. Okno s touto odpovědí na zprávu je zobrazeno na obrázku 10:



Obr. 10: Okno Responses/Messages‘ Details programu ModbusTest-F.

V tomto okně je k vidění na deseti kartách deset odpovědí na našich deset zpráv. Každá karta má identický obsah. Je vidět kterou kartu máme otevřenou, a jestli vše proběhlo v pořádku (sloupec slov nalevo). Pokud něco není v pořádku, tak se

odpovídající nápis rozsvítí červeně. Že něco není v pořádku, zjišťuje regulátor. V takovém případě pošle chybovou hlášku místo klasické odpovědi. Podle té zjistím, co je za problém. Chyby typu špatná délka kódu (první tři chyby) a podobně již nenastávají, protože to není programově možné. Chyby typu timeout zjistím sám, pokud deska nedodrží časová kvanta na komunikaci. Takže jediné chyby, které opravdu přichází od regulátoru, jsou Slave Processing Failure, Line Error a Response Error. Všechny popisují chybu práce desky se zprávou, poslední z nich je obecná chyba, která není definována, a zahrnuje vše, co nespadá jinam.

V tomto případě je vše v pořádku (to je i vidět na okně v pozadí, že zpráva 0 má zelený stav), a proto nic nesvítí. Napravo od chybových hlášek jsou tři okna. První okno je okno Query Message. Zde je zobrazen hexadecimální zápis našeho požadavku po bytech. Druhé je okno Response Message. V něm je k vidění celá zpráva, kterou jsme obdrželi od regulátoru. Je zobrazena ve stejném formátu. Třetí okno je okno Data Message. Zde jsou data, která jsme obdrželi od regulátoru, tzn., že obsah je identický s druhým oknem, akorát je ořízntutý o neužitečný obsah. Obě okna tu jsou pro případ poruchy, pokud bych se chtěl podívat na celou zprávu, nebo pro případ, že porucha nenastala, tak abych nemusel složitě počítat, na které pozici začínají data. Formát zobrazení v tomto okně je trochu jiný. Lze zvolit mezi hexadecimálním a decimálním zápisem. V obou případech se data zobrazují po slovech (wordech) se čtyřmi mezerami mezi sebou pro větší přehlednost. Navíc jsou ještě doplněna na čtyři místa nulami pro hexadecimální zápis a na pět pro decimální zápis. Tak je činěno z důvodu přehlednosti, aby jednotlivá slova byla vždy zarovnána pod sebou a netvořily se nám v zobrazení různé „schody“.

Na obrázku je vidět, že jsem zaslal zprávu s FC 6 – zápis jednoho slova do adresy 4096 (0x1000). Zapsaná hodnota byla 0x541. V odpovědi jsou vidět stejné hodnoty, což je správně. V datech je vidět číslo 1345, což je rovno číslu 0x541. Vše funguje.

Zajímavostí podle mě je, že pro zmenšení náročnosti této komponenty jsem negeneroval deset karet s desetkrát stejným obsahem. Místo toho je záložková karta udělaná jen na oko. Funguje, avšak každá záložka je proužek tenký jeden pixel pod nabídkou karet. To se projevuje jako ona šedá čára pod nimi. Funkčnost okna je zaručena actionListenerem, který při změně záložky nadiktuje změnu obsahu oken

podle hodnot v Modelu M. Díky tomu se délka kódu tohoto okna zkrátila z 3500 řádek na 500 řádek.

Poslední částí hlavního okna je nastavení počtu přenosu. Jsou dvě možnosti. Buď mohu zvolit opakovaný přenos a definovat počet jeho opakování a čas, po který se má počkat mezi jednotlivými přenosy, nebo mohu zvolit nepřetržitý přenos. Ten je vhodný, pokud chci zjistit, jestli se s časem nevloudí nějaká chyba do přenosu (při testování se projevilo například špatně nastavená inkrementace proměnné v regulátoru, kdy po určité době došlo k přetečení hodnoty a program kolaboval). V obou případech je možnost automatického přerušení v případě chyby přenosu.

Na závěr jsou v tomto programu už dvě velká tlačítka, která vše spouští. Tlačítko „Play“ a tlačítko „Stop“.

Tlačítko Play se po stisknutí změní na tlačítko „Pause“. Poté se zablokuje tlačítko Options, aby nešlo změnit nastavení připojení v průběhu připojení. Poté se vytvoří nová instance vlákna. Toto vlákno získá z Modelu M potřebná data a vytvoří spojení s externí deskou. Díky tomu, že komunikace probíhá ve vlastním vlákně, tak po dobu komunikace je stále možné používat ostatní části programu. V případě použití jen jednoho vlákna aplikace zamrzla s obrázkem stisknutého tlačítka „Play“ a nešlo nic dělat, dokud celá komunikace nedoběhla. Knihovna na toto spojení v Javě není, ale já jsem ji našel na internetu pod jménem „net.wimpi.modbus“. Autorem knihovny je Dieter Wimberger vystupující jako wimpi na stránce users.sourceforge.net. Tato knihovna je pod licencí Apache v2.0, což znamená, že jsou k volnému použití k libovolnému účelu. Za jeho tvorbu mu děkuji, neboť mi byla velmi nápomocná. Z dat, která vlákno vytáhne z modelu, vytvořím na základě zvoleného protokolu spojení. V této knihovně se pracuje s tzv. Requesty (požadavky). To je posloupnost bytů, která reprezentuje jednu zprávu. Tyto Requesty vytvářím metodou, kdy podle zvoleného FC volím, jestli vytvořím novou instanci ReadMultipleRegisterRequest či WriteMultipleRegisterRequest (klasická ukázka dědění v Javě). Poté všechny zprávy pošlu desce, kontroluji jejich průběh a opakuji podle zvoleného počtu (n-krát či stále dokola). Pro správnou funkčnost komunikace vždy zašlu jeden dotaz a ihned se zeptám na odpověď desky. Pokud není deska připravena (na bufferu s odpovědí není minimální počet znaků pro zprávu, to je devět bytů), tak program čeká až do doby nastavené jako reception

timeout. Pokud bychom poslali více zpráv v jednom paketu do regulátoru, tak on nebude vědět co s nimi a vypíše chybovou hlášku.

V případě stisknutí tlačítka Pause se komunikační vlákno ukončí a uloží se, kde skončilo, tlačítko se opět změní ikonu na Play. Nechtěl jsem, aby mi zůstalo v paměti pozastavené vlákno. Navíc po chvíli přišel Timeout v komunikaci a navázat opětovné spojení již nešlo. Opětovné stisknutí vytvoří nové vlákno, které přeskočí všechny přenosy zpráv až do té, kde předchozí vlákno skončilo. V průběhu komunikace nelze změnit např. IP a tím vytvořit něco nesmyslného, tlačítko Options je stále zakázáno.

V případě stisknutí tlačítka Stop se tlačítko Play vždy nastaví na ikonu Play, ukončí se násilně vlákno a vymaže se z paměti případné informace o tom, kde jsme se zastavili při pauze. Žádné potíže toto násilné zastavení nezpůsobí, v regulátoru nezůstanou připravené případné odpovědi, a i kdyby, tak se vymažou při novém připojení (je velká pravděpodobnost, že se trefíme do průběhu komunikace).

To je v kostce celý program ModbusTest-F. Jeho vývoj mi trval půl roku práce dva dny v týdnu a byl mým odrazovým můstkem pro ostatní programy. S těmi jsem měl práci výrazně lehčí, protože už jsem si ozkoušel spoustu chyb a mohl jsem pro ně používat tento kód. Některé chyby regulátoru Emadyn-F, které testování a používání programu ModbusTest-F odhalilo, bylo například zaměnění Little a Big Endianu - deska má používat Big Endian, ale ve wordech používala Little Endian, či odpovídala na multicast (station address nastaveno na 0).

3.3.2 Emvis-F

Popis aplikace

Emvis-F je aplikace pro sledování veličin a nastavování parametrů a diagnostické nástroje v regulátoru Emadyn-F. Jeho jméno je zkratka pro Emadyn Visualisation, písmeno F je další v pořadí programů třídy Emvis. Opět jsem zde použil návrhového vzoru MVC. Tento program není kompletně hotový, jak bych si ho představoval. Důvodem pro to je nekompletní vývoj desky Emadyn-F ve firmě. Základní funkce

zprovozněné má, na obsluhu případného zařízení už stačí, nefungují pouze některé věci okolo.

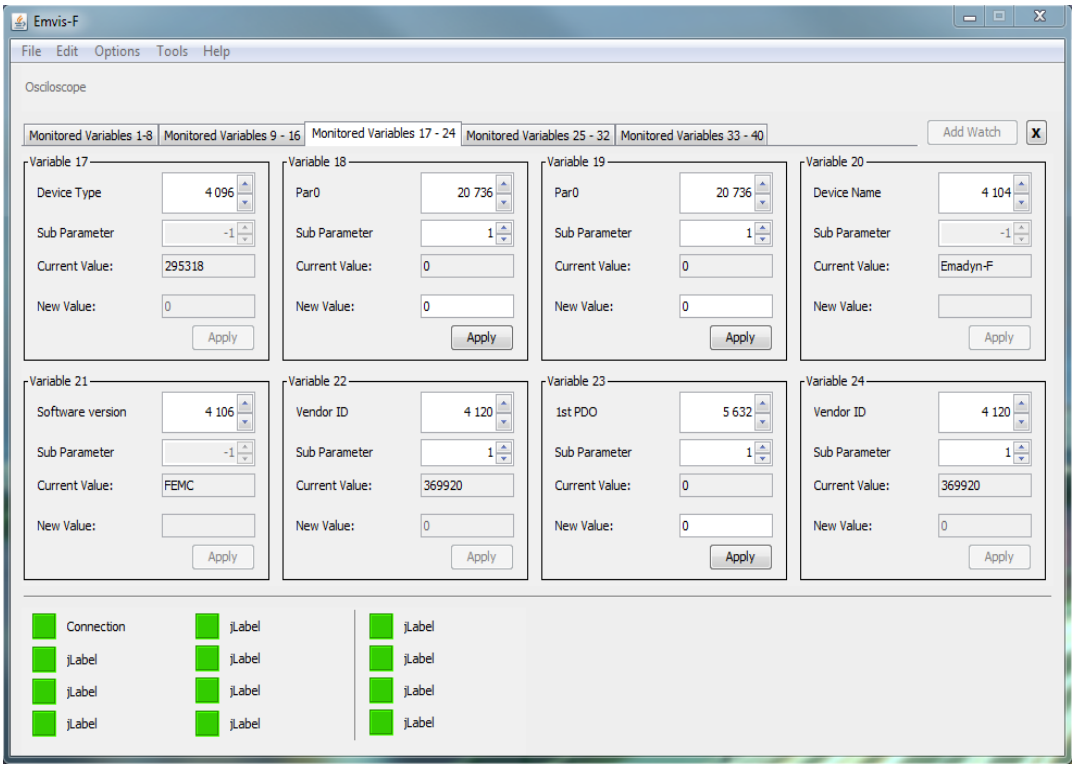
Model aplikace

Program je v určitých aspektech podobný programu ModbusTest-F – opět je zde grafika, nastavují se zprávy a posílají se po síti. Hned v síti je rozdíl. Používaný protokol pro tento program je pouze Modbus-TCP. V tomto programu jsem již striktně dodržel MVC návrhový model.

Model programu je podobný Modelu ModbusTestu-F. Opět jsou zde data o připojení, akorát jsou jednodušší, díky tomu, že se používá jen Modbus-TCP. Opět se vytváří na základě souboru či dle defaultních hodnot. Není zde však již část pro zprávy, protože není potřeba. Všechny informace o zprávách se drží v části jakožto hodnoty objektů ve View. To není narušení MVC modelu, protože ty nějaký obsah potřebují.

Vlastní implementace aplikace

Při spuštění se spustí hlavní okno programu (viz. obrázek 11).

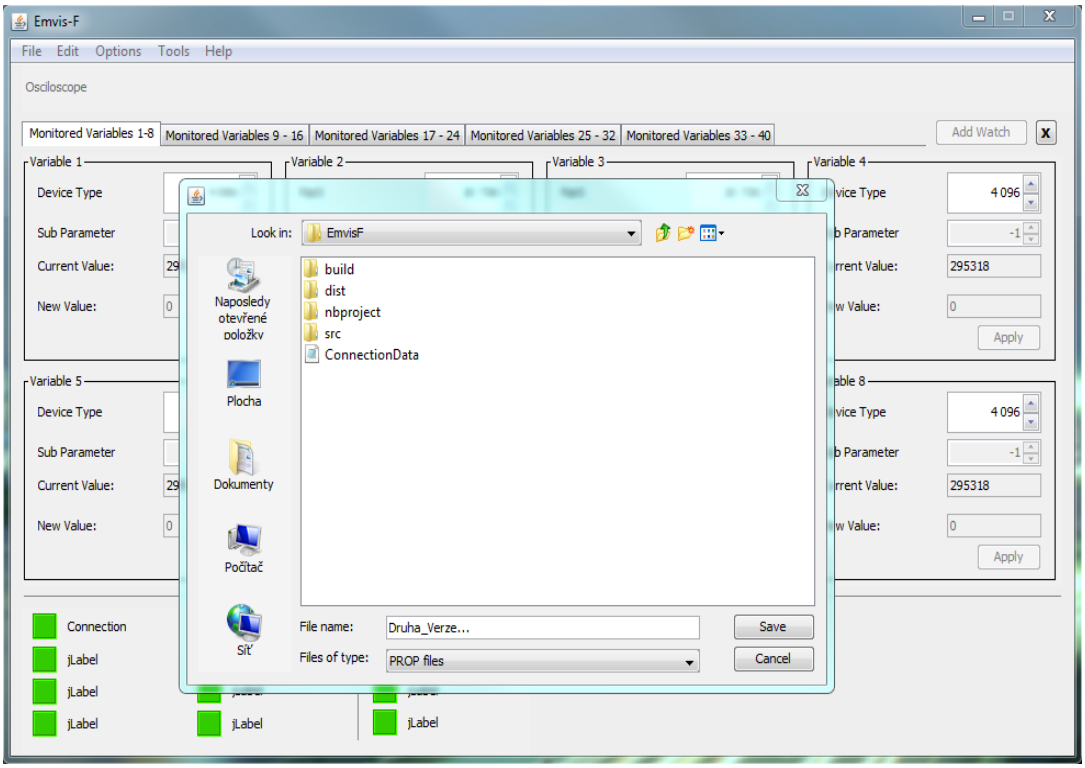


Obr. 11: Hlavní okno programu Emvis-F.

Nahoře se nachází panel s nabídkou, pod ním se nachází panel nástrojů, uprostřed je přehled sledovaných veličin, vpravo nad ním měníme počet sledovaných veličin a vlevo dole se nachází kontrolní „LED diody“. Je vidět, že ne všechny části jsou funkční. Hned na první pohled je uživateli jasné, že se zatím pouze dozví, jestli je připojen či není. Ostatní signalizační LED zatím nemají přiřazenou funkci. To je kvůli tomu, že ještě nevím, co budou signalizovat, ale obecně to bude stav nějakého elektromotoru. Dále je v nástrojích zatím pouze osciloskop. V předchůdci bylo nástrojů více, ale opět z důvodu jiné architektury procesoru, a také z důvodu jiného přístupu k problematice, tam zatím bude jen tento nástroj. Ten není ještě funkční, protože regulátor nemá žádná data, se kterými bych mohl pracovat. Plánuji použít Java knihovny pro kreslení grafů pro tvorbu tohoto nástroje. Deska Emadynu-F zatím jen podává informace typu jméno či verze softwaru. V panelové nabídce je opět záložka Tools. Ta bude obsahovat to samé, co nástrojový panel, takže zatím jen osciloskop.

V panelu s nabídkou je pět záložek. Pod první, File, se nachází uložení a načtení nastavení. Pokud si uživatel chce zachovat stávající nastavení, tak si ho může uložit pod speciálním názvem (jiným, než je defaultní soubor, který se načítá při spuštění a

přepisuje při vypnutí programu), nebo si zde může otevřít nějaké dřívější nastavení. Okno pro načtení a uložení si lze prohlédnout na obrázku 12.

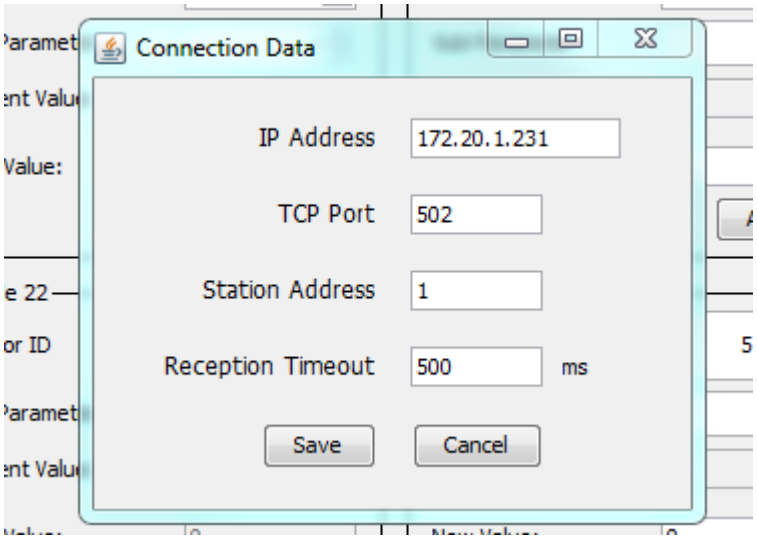


Obr. 12: Okno s načtení a uložením souboru Emvisu-F.

Na základě vybraného tlačítka z nabídky File se vykreslí jedno a to samé okno jen s rozdílem v popisu na potvrzovacím tlačítku (Save/Load). Přidal jsem filtr na povolené koncovky. Toto okno je podporované v Javě a vychází z OS Windows. Opět potvrzovací tlačítko spustí patřičnou akci (popíšu později) a zavře okno, tlačítko Cancel jen okno zavře.

Pod druhou záložkou, Edit, se zatím nachází nic. Přidám tam nastavení volby zobrazení adresy v programu. V nabídce bude zobrazení hexadecimální a decimální. Ostatní funkce budou doplňovány na základě požadavků zadavatele.

Pod třetí záložkou, Options, se nastavují dvě věci. První je okno s nastavením síťového připojení. Toto okno je vidět na obrázku 13.



Obr. 13: Okno s nastavením síťového připojení.

Jak je vidět, tak je opravdu výrazně jednodušší, než stejné okno v programu ModbusTest-F. Opět se při otevření naplní daty z Modelu, při zavření se podle stisknutého tlačítka Model upraví, či nikoliv.

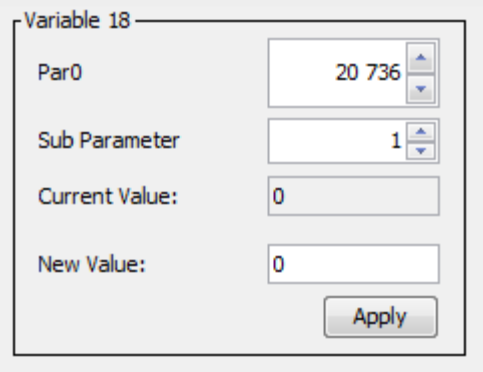
Druhá záložka v nabídce Options zatím není funkční. Je to nastavení PostMortu. PostMort se používá, pokud dojde k nouzovému zastavení regulovaného zařízení. PostMort zaznamená do souboru posledních n hodnot, abychom se mohli podívat, co se dělo, než systém přešel do nouzového stavu. Jelikož ale není Emadyn-F spustitelný, tak jsem nemohl vyvinout PostMort, navíc je to jedna z věcí, které nespěchají (mohl jsem vyvinout nějakou jeho verzi, ale nikdy bych nevěděl, jestli funguje, navíc zatím neznám chování systému Emadyn-F v případě nouzového zastavení).

V poslední záložce, About, je záložka otevírající malé okno, opět s popisem programu, logem firmy a zmínkou o autorských právech.

Nyní se dostáváme k hlavní části programu. To je sada záložkových karet se identickým obsahem (až na hodnoty jejich polí). Karet může být od jedné do pěti (více by se jich nevešlo a bylo by jich zbytečně mnoho), přidávají se tlačítkem Add Watch a odebírají křížkem. Opět jsem nevytvářel pět identických oken, ale jen nabídkovou lištu, která vyvolává překreslování jejich obsahu. Obsahem mám na mysli osm polí se sledovanými veličinami. Každá veličina je definována svou adresou a subadresou. Tímto se dostávám k popisu toho, jak se komunikuje s deskou

v detailu. V programu MudbusTest-F jsem pouze zjišťoval bajtové hodnoty. Nyní mě i zajímá, co znamenají, protože již chci desku ovládat. Deska obsahuje tzv. adresář objektů. Ten se nachází v její paměti a je v něm uložen seznam proměnných a jejich hodnoty. Program Emvis-F tento adresář objektů nezná, protože každé zařízení má jiný. Je však nadefinováno, že všechny systémy Emadyn-F budou mít první objekt v adresáři objektů na adrese 0x1000 (4096). Každý obejkt v adresáři si nese informaci o předchozím a následujícím objektu, přičemž první a poslední ukazují samy na sebe, pokud bychom se chtěli dostat na předchozí, resp. následující. Tudíž postupně jsme schopni zjistit celý obsah adresáře objektů. Adresa a subadresa představují dvourozměrné pole v paměti regulátoru. Díky této znalosti jsme schopni vyčíst celý adresář objektů z desky a program Emvis-F ví, na které adresy má smysl se dotazovat. Celý adresář objektů se ale nevyčítá, protože to by mohlo být zbytečně mnoho dat při inicializaci a zbytečný provoz po síti. Načtou se pouze hodnoty na adresách uložených v inicializačním souboru. Pokud by někdo sáhnul na tento soubor a nastavil hodnoty, které nejsou platné (to se zjistí z odpovědi regulátoru), tak program nastaví místo toho hodnotu 0x1000. V záložce File se při výběru Load také spustí tato inicializace. V modelu se tedy drží pouze osm zobrazených veličin.

Znovu zde vložím část hlavního okna, zvětšenou, aby bylo dobře vidět, která pole se upravují (viz obrázek 14).



Obr. 14: Pole s jednou sledovanou veličinou.

Komunikace probíhá následovně (pakliže probíhá dotaz na objekt na adrese např. 20736):

1) Zjisti, jestli co se skrývá na adrese 20736 za datový typ.

- může to být číslo (Integer)

- může to být řetězec (String)
- může to být záznam (Record)

2) Číslo:

- zeptej se na jméno proměnné a vyplň ho - Par0

- zeptej se na hodnotu proměnné a vyplň ji - 0

- zjisti, jestli je čtení, či i k zápisu - ano. Pokud je k zápisu, proveď

- zeptej se na horní a dolní omezení, nastav to jako hranice pro pole new value, pak se zeptej na default value a nastav ho jako hodnotu do pole new value - 0

jinak

- zablokuj pole new value a tlačítko Apply (to spouští tvorbu nového požadavku na zápis hodnoty na danou adresu v paměti regulátoru)

3) Řetězec

- zeptej se na jméno proměnné a vyplň ho

- zeptej se na hodnotu proměnné a vyplň ji

- zjisti, jestli je čtení, či i k zápisu a povol či zablokuj nastavení nové hodnoty

4) Záznam

- zeptej se na počet subindexů, povol spinner volící subindex (spinner je běžně zablokován s hodnotou -1, která značí kódu, že ho nemá používat) a nastav první jako zvolený. Počet subindexů použij pro nastavení modelu pro tento spinner, aby uživatel nemohl nastavit subindex, který neexistuje.

- opakuj (zjisti, jestli na subindexu 1 je číslo či řetězec, pro záznam vypiš chybovou hlášku)

Pro tuto komunikaci používám proprietární funkční kódy. Mají označení 65 a 66. 65 se používá pro čtení a kód 66 se používá pro zápis. V příloze přikládám Excelový dokument s jejich popisem a popisem používaných standardních kódů.

Uživatel tedy vidí takto zinicializované okno. Pokud chce sledovat jiný parametr regulátoru, stačí se posunout šipkami v jednom ze spinnerů. Pokud provede posun v hlavním spinneru, tak se zavolá metoda, která se zeptá desky, který je sousední index daným směrem (např. po indexu 0x1000 následuje 0x1008, všechny hodnoty nejsou obsazené). Tento index nastaví jako hodnotu spinneru a zjistí vše o něm. Pokud se uživatel posune v subspinneru, tak se aplikace dotazovat desky na sousední index nemusí, ví, že je to přímo ten sousední. Nastavený model subspinneru (spinner označující subindex) zabraňuje vyběhnutí z mezí. Subindex se změní a provede se zjištění všech hodnot na dané adrese. V modelu si nemusíme držet žádné povědomí o adresáři proměnných desky, vždy se na něj ptáme, jen když je to potřeba.

Update této tabulky probíhá v dvouvteřinových cyklech, po kterých se nastaví 8 požadavků na aktuální hodnotu sledované veličiny. Update také proběhne při přepnutí záložky se sledovanými veličinami. V tu chvíli se program zeptá, co je na jiné kartě, nastaví tyto hodnoty do všech spinnerů a provede celý update každé karty. Jeden update trvá 6 dotazů, jeden dotaz při spojení v rámci jedné místnosti trval 100us, přičemž by ho šlo i zkrátit, ale není třeba jít do extrémů. Tudíž při 6 krát 8 dotazech je update tabulky při přepnutí okamžitý, trvá cca 5ms, čehož si nikdo není schopen všimnout.

Controller je udělán trochu odlišně od ModbusTestu-F. Zaprvé spojení je udržováno neustále. Při spuštění se vytvoří spojení, nebo se rozsvítí chybová LED. Spojení se může měnit jen změnou v okně Connection data, nebo off-line přepsáním spouštěcího program, tudíž nehrozí žádný problém s integritou dat a spojení. Opět probíhá ve vlastním vlákně ze stejného důvodu, aby neblokoval běh zbytku programu. Buffer je implementován formou fronty, konkrétně pomocí LinkedBlockedQueue. To je fronta podporující uspání vlákna, které ho obsluhuje. Vlákno tedy stále chce odesílat data, dokud jsou nějaká k dispozici. Program vždy provádí sekvenci: zadej dotaz → zeptej se na odpověď, aby nedošlo k smíchání více zpráv do jedné.

Program Emvis-F je tedy momentálně schopen zjistit hodnoty všech veličin, které se nachází v regulátoru v adresáři objektů. Umí je také měnit, pokud je to povoleno. Nedostane se do žádného stavu, ve kterém by se zasekl a musel by být vypnut.

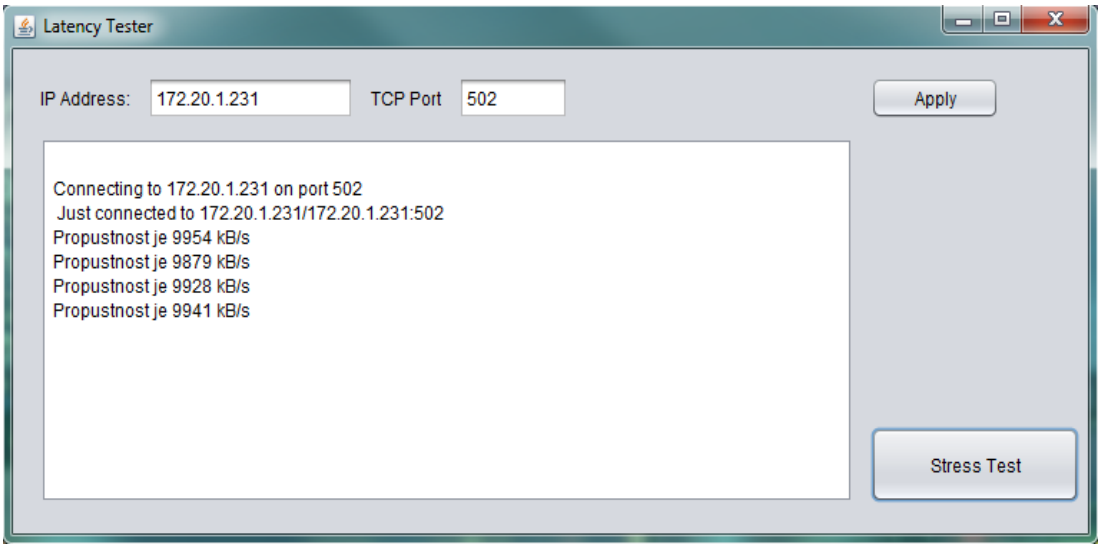
Ostatní funkce nejsou stěžejní, ale jejich vývoj je v plánu. Ověření programu proběhlo po protokolu Ethernet i EtherCAT, jeho výsledky jsou výsledky prezentované výše na obrázcích z jeho chodu. Ačkoliv to není cílem programu, tak se s ním také povedlo najít některé omyly ve vývoji Emadynu-F. Byly to například chybějící zabezpečení proti posunu pod nejnižší a přes nejvyšší index v adresáři objektů, či neprázdné návratové hodnoty na dotaz typu „jaká je nejnižší povolená hodnota řetězce“, což je samozřejmě nesmysl.

3.3.2 Aplikace nezahrnuté v zadání bakalářské práce

Kromě těchto dvou aplikací jsem vyvinul i dvě menší podpůrné aplikace, které jsme použili při vývoji systému Emadyn-F.

Aplikace testování propustnosti - Latency

První aplikace testuje využití fyzické vrstvy sítě. Funguje tak, že naváže TCP/IP spojení, poté se vytvoří zpráva 1218 nulových bajtů. Ta se pošle po tomto spojení. 1218 jsem zvolil proto, že velikost TCP paketu je 1240 bajtů a 32 bajtů je jeho velikost hlavičky. Tím jsem chtěl vyloučit zpoždění či menší využití přenosového média kvůli softwaru. Poté tyto zprávy posílám desce a sleduji, kolik jich stihnu odeslat za vteřinu. Toto číslo vynásobím velikostí paketu a výsledkem je přenosovou rychlost. Aplikace vypadá následovně (viz obrázek 15):



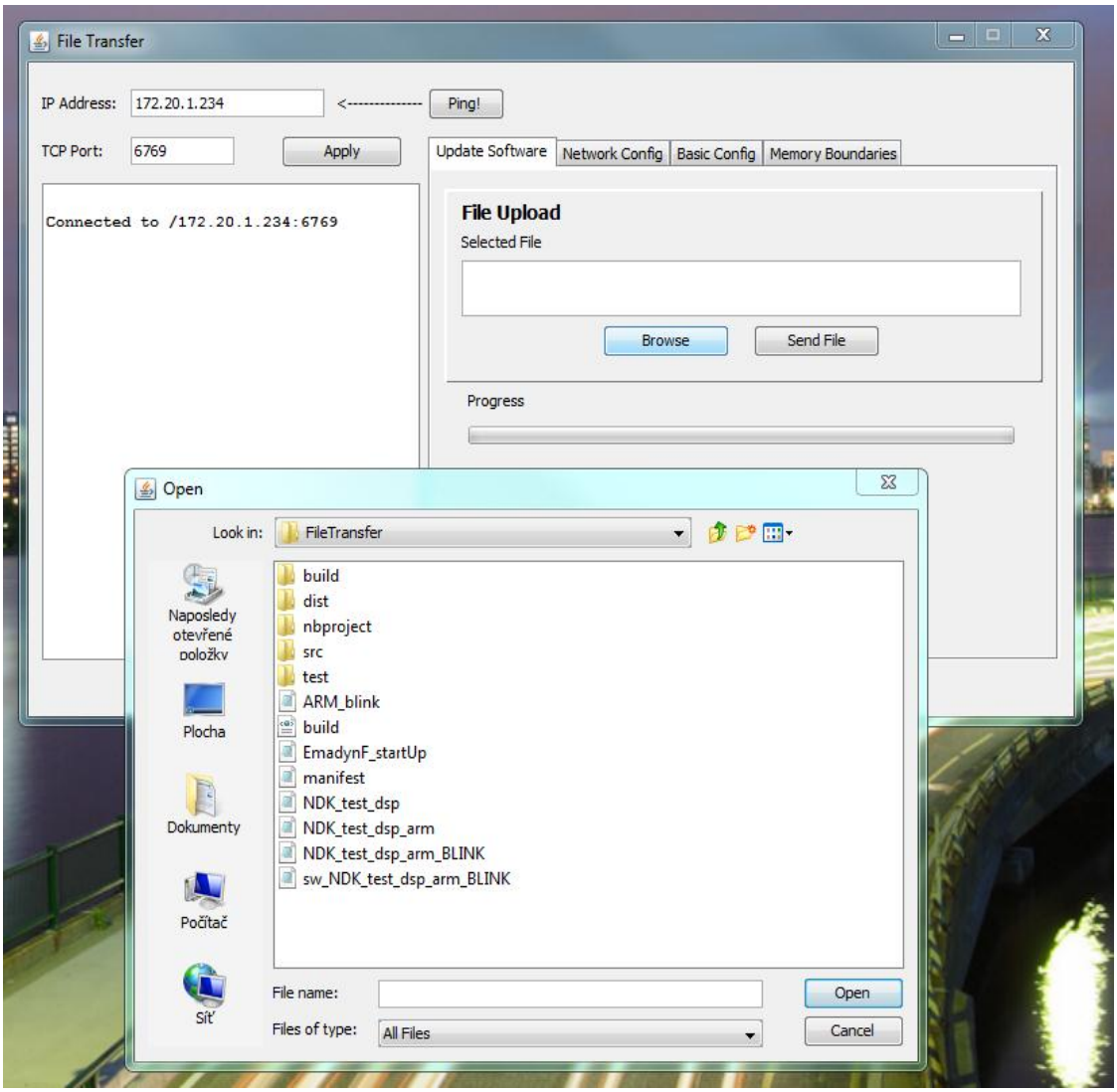
Obr. 15: Aplikace na testování využití komunikačního média.

Testované vlákno bylo 10MB/s, tudíž vytížení se opravdu blíží 100%.

Pozn.: Aplikaci jsem nazval Latency, ačkoliv nyní vím, že tento název označuje zpoždění a nikoliv propustnost dat.

Aplikace File Transfer

Druhá aplikace nahrává nový software do desky Emadynu-F a mění parametry DSP paměti (na začátku jsem zmiňoval, že ji lze do určité míry sdílet mezi ARMem a DPS) a síťové parametry desky (její IP, MAC adresu, Reception Timeout apod.). Funguje po proprietárním protokolu. Každý přenos je zahájen inicializačním paketem, poté následuje paket s daty. Poslední paket s daty je speciální a značí, že to je poslední paket a ať se soubor uloží. Poté existují pakety nastavující DSP a síťové parametry regulátoru Emadyn-F. Ukázka programu je na obrázku 16.



Obr. 16: Aplikace na update softwaru Emadynu-F a jeho základní nastavení.

4 Závěr

Cílem této práce bylo vytvořit novou verzi monitorovacího a testovacího softwaru k systému Emadyn-F vyvíjeného firmou ČKD Elektrotechnika, a.s. Prvním bodem bylo seznámení se s fungováním systému Emadyn-F jako takového. Druhým bodem bylo nastudování komunikačních protokolů, které systém Emadyn-F využívá. Tyto body byly podmínkou toho, abych byl schopen práci vytvořit. Systém Emadyn-F dodržuje protokoly přesně podle norem. Nakonec jsem se zabýval samotnou implementací sady dvou vetších a dvou menších aplikací pro testování a sledování systému Emadyn-F.

Koncepty těchto aplikací byly předmětem diskuze s těmi, kteří je budou používat. Rozhodně mi tento postup velmi rozšířil obzory, protože předtím jsem nedělal žádnou reálnou aplikace (vyjma studijních a výukových). Poznatky uživatelů z praxe takto velmi ovlivnily veškerý vývoj těchto aplikací.

Jako první byla vyvinuta aplikace ModbusTest-F. Tato aplikace slouží k testování komunikace pomocí protokolu Modbus. Aplikace ModbusTest-F pomohla najít pár chyb a dnes u regulátoru Emadyn-F nejsou žádné chyby známy, co se týče komunikačních protokolů.

Druhá vyvinutá aplikace je aplikace Emvis-F. Tato aplikace komunikuje s deskou Emadynu-F a již se jí ptá na konkrétní věci, jako je její jméno či verze jejího softwaru. Zjišťuje, na jakých adresách jsou dané veličiny, jaké mají hodnoty a jakých hodnot mohou nabývat. Časem do ní přibude sada nástrojů pro vykreslování grafů průběhů napětí a jiných veličin.

Menší aplikace slouží k testování využiti fyzické vrstvy sítě, k nahrávání nového softwaru do Emadynu-F a nastavení jeho síťových a paměťových parametrů. Tyto programy nejsou oficiálně součástí zadání, ale byly důležité.

Práci bych označil za úspěšně splněnou. Vyvinuté aplikace fungují a pomáhají ulehčit každodenní práci mým kolegům v ČKD Elektrotechnika. Vývoj na nich bude pokračovat i v budoucnu.

A Seznam použitých zkratek

JVM - Java Virtual Machine

ČKD - Českomoravská Kolben Daněk

TI - Texas Instruments

DSP - Digital Signal Processor

FPGA – Field Programmable Gate Array

RISC - Reduced Instruction Set Computer

CCS - Code Composer Studio

CRC - Cyclic Redundary Check

OOP - Object Oriented Programming

MVC - Model View Controller pattern

JRE - Java Runtime Enviroment

B Použitá literatura

[1] Libor Dostálek, Alena Kabelová - Velký průvodce protokoly TCP/IP a systémem DNS, aktualizované druhé vydání - Computer Press, Praha, 2000

[2] Herbert Schildt - Java 7 Výukový kurz - Computer Press, Brno, 2012

[3] Technical reference to OMAP-L137 - Interní dokumentace ČKD, dostupné z WWW [cit. 21.5.2013]: http://www.ti.com/lit/ds/sprs563f/sprs563f.pdf

[4] Texas Instrument, Code Composer Studio, dostupné z WWW [cit. 21.5.2013]: http://www.ti.com/tool/ccstudio

[5] Eclipse, Real Time Software Components, dostupné z WWW [cit. 21.5.2013]: http://www.eclipse.org/proposals/rtsc/

[6] Texas Instruments, Network Developers Kit FAQ, dostupné z WWW [cit. 21.5.2013]: http://processors.wiki.ti.com/index.php/Network_Developers_Kit_FAQ

[7] Texas Instruments, úvod do Platform Support Products, dostupné z WWW [cit. 21.5.2013]: http://processors.wiki.ti.com/index.php/PSP_Introduction_and_Product_FAQ

[8] Texas Instruments, uživatelský manuál k Network Developer's Kit, dostupné z WWW [cit. 21.5.2013]: http://www.ti.com/lit/ug/spru523h/spru523h.pdf

[9] EtherCAT Technology Group, popis protokolu EtherCAT, dostupné z WWW [cit. 21.5.2013]: http://www.ethercat.org/en/technology.html

[10] Modbus Organization, popis protokolu Modbus verze 1.1b, dostupné z WWW [cit. 21.5.2013]: http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf

[11] Simply Modbus, popis protokolu Modbus, dostupné z WWW [cit. 21.5.2013]: http://www.simplymodbus.ca

[12] Oracle, popis MVC modelu, dostupné z WWW [cit. 21.5.2013]: http://www.oracle.com/technetwork/articles/javase/mvc-136693.html

[13] CodeProject, popis MVC modelu, dostupné z WWW [cit. 21.5.2013]: http://www.codeproject.com/Articles/25057/Simple-Example-of-MVC-Model-View-Controller-Design

C Obsah přiloženého CD

Složky se zkompilovaným programem a se zdrojovými kódy:

- ModbusTest-F
- Emvis-F
- Latency
- FileTransfer

Skoupy_Sebastian_Bakalarska_Prace.pdf - tato práce v PDF

Modbus.xls - popis používaných kódů v Excelu