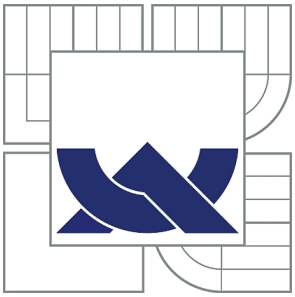


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

NÁVRH A REALIZACE CANOPEN KOMUNIKAČNÍ JEDNOTKY

DESIGN AND IMPLEMENTATION OF A CANOPEN COMMUNICATION UNIT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. TOMÁŠ ŘEHÁK
AUTHOR

VEDOUCÍ PRÁCE — Ing. PAVEL HOUŠKA, Ph.D.
SUPERVISOR

BRNO 2013

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav automatizace a informatiky
Akademický rok: 2012/2013

ZADÁNÍ DIPLOMOVÉ PRÁCE

student(ka): Bc. Tomáš Řehák

který/která studuje v **magisterském navazujícím studijním programu**

obor: **Aplikovaná informatika a řízení (3902T001)**

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

Návrh a realizace CANopen komunikační jednotky

v anglickém jazyce:

Design and implementation of a CANopen communication unit

Stručná charakteristika problematiky úkolu:

Práce se zabývá realizací komunikátoru pro diagnostiku a ovládání řídicích jednotek elektrických motorů, komunikujících prostřednictvím protokolu CANopen. Zvolenou platformou pro realizaci komunikátoru je .NET Micro Framework na vývojovém kitu GHI EMX. Pro vývoj budou primárně použity řídicí jednotky Faulhaber MCDC, MCBL, MCLM, dále pak Maxon EPOS2.

Cíle diplomové práce:

1. Seznamte se platformou .NET Micro Framework a komunikačním protokolem CANopen.
2. Navrhněte a realizujte komunikační rozhraní kompatibilní s CANopen na vývojovém kitu GHI EMX.
3. Definujte parametry propojení komunikátoru s řídicími jednotkami Faulhaber a realizujte jejich připojování.
4. Navrhněte a realizujte vhodné uživatelské rozhraní pro ovládání řídicích jednotek.
5. Realizovaný komunikátor ověřte a zhodnoťte.

Seznam odborné literatury:

[1] Pfeiffer O., Ayre A.,Keydel Ch.: Embedded Networking with CAN and CANopen, Copperhill Technologies Corporation, ISBN 978-0-9765116-2-5
[2] Faulhaber: Communication and function manual (CANopen), 1st edition, 23.02.2012, dostupné z www.faulhaber.com
[3] Microsoft .NET Micro Framework, stránky projektu, dostupné z http://netmf.codeplex.com
[4] Issa, G.: Beginners Guide to C# and .NET Micro Framework, GHI e-book, dostupné z http://wiki.tinyclr.com/
[5] Beginner Porting Guide ebook, GHI e-book, dostupné z http://wiki.tinyclr.com/

Vedoucí diplomové práce: Ing. Pavel Houška, Ph.D.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2012/2013.

V Brně, dne 21.11.2012

L.S.

Ing. Jan Roupec, Ph.D. Ředitel ústavu	prof. RNDr. Miroslav Doupovec, CSc., dr. h. c. Děkan fakulty

ABSTRAKT

Tato diplomová práce se zabývá návrhem a realizací průmyslového komunikátoru pro synchronizaci a ovládání řídicích jednotek elektrických pohonů s rozhraním RS232 nebo CANopen. Hardware komunikátoru je postaven na platformě vývojového systému EMX od společnosti GHI Electronics. Software komunikátoru je realizován v jazyce C# na platformě .NET Micro Framework.

Komunikátor umožňuje nastavení a kontrolu až čtyř motorových pohonů. Po úspěšném navázání komunikace s jednotlivými jednotkami lze řídit natočení, které je zadáváno počtem otáček s přesností na tisícinu otáčky. Komunikátor také periodicky zjišťuje a zobrazuje aktuální hodnotu natočení připojených jednotek. Uživatel obsluhuje komunikátor v grafickém prostředí, pokyny aplikaci zadává pomocí dotykového displeje nebo tlačítek.

ABSTRACT

This diploma thesis describes the design and implementation of an industrial communicator for synchronization and control of electrical drive controllers with the interface of RS232 or CANopen. Hardware of communicator uses the EMX development system by GHI Electronics. Communicator software is implemented in C# by the .NET Micro Framework.

Communicator allows user to adjust and control up to four electric motor drives. After a successful connection with each unit, communicator can control its position, which specify the number of revolutions to the nearest thousandth of a turn. Communicator also periodically finds out and displays the current position of the connected units. User operates the communicator with GUI, application instructions are entered via the touch screen or buttons.

KLÍČOVÁ SLOVA

.NET Micro Framework, CANopen, průmyslový komunikátor, řízení pohybu, GHI-EMX vývojový systém.

KEYWORDS

.NET Micro Framework, CANopen, Industrial Communicator, GHI-EMX Development System, Motion Control.

PROHLÁŠENÍ O ORIGINALITĚ

Tímto prohlašuji, že předkládaná diplomová práce je mojí původní autorskou prací, kterou jsem vypracoval pod vedením vedoucího diplomové práce a s použitím uvedené odborné literatury.

Bc. Tomáš Řehák, Brno, 2013

BIBLIOGRAFICKÁ CITACE

ŘEHÁK, T. *Návrh a realizace CANopen komunikační jednotky*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2013. 64 s. Vedoucí diplomové práce Ing. Pavel Houška, Ph.D..

PODĚKOVÁNÍ

Touto cestou bych chtěl poděkovat Ing. Pavlu Houškovi, Ph.D. za odborné vedení, cenné rady, připomínky a celkovou pomoc poskytnutou při zpracování diplomové práce.

OBSAH

1 ÚVOD

Microsoft .NET a jeho odlehčená verze .NET Micro Framework (NETMF) se zaměřuje na specifické požadavky omezených zdrojů vestavných systémů. Díky použití vyššího programovacího jazyka pomáhá uživatele zcela odstínit od nízko úrovňových konstrukčních detailů hardwarové platformy a umožňuje relativně rychlý vývoj aplikací v jazyce C#, který využívá výhody objektově orientovaného programování. Micro Framework je dostupný zdarma včetně všech jeho součástí.

Existují dva způsoby práce s NETMF. Obtížné portování systému provádí výrobce hardwaru, používání naportovaného systému uživatelem je již snadné. Z ekonomických důvodů není možné portovat NETMF na nový hardware v případě, že se zaměřuje na střední nebo nízké množství aplikací. Tato nevýhoda je kompenzována modularitou a robustností systému díky dostupným OEM modulům, které mají veškeré nutné vybavení po hardwarové i softwarové stránce [1].

NETMF lze rozdělit do dvou hlavních částí. Jádro CLR (*Common Language Runtime*) a na vrstvu HAL (*Hardware Access Layer*). Knihovny jádra jsou implementovány tak, aby byly nezávislé na konkrétní hardwarové platformě, obvykle nejsou v knihovnách jádra nutné žádné dodatečné uživatelské úpravy. Vlastní portování NETMF pro specifickou hardwarovou platformu závisí pouze na zvládnutí specifické HAL vrstvy, tzn. propojení a zprovoznění hardwaru, ovládání jeho horních vrstev [1].

NETMF implementuje emulátor, který může být rozšířen nebo modifikován dle potřeb uživatele, většinu vývoje a ladění aplikace lze provést před vlastním nahráním programu na hardware. Emulátor obsahuje funkcionalitu směrových tlačítek a LCD displej skutečného rozlišení, nejsou podporovány vyšší externí funkce jako PWM, USB Host, atd. Ladění aplikace je možné také přímo na vývojovém kitu, v obou případech však probíhá ve spolupráci s Microsoft Visual Studiem [1].

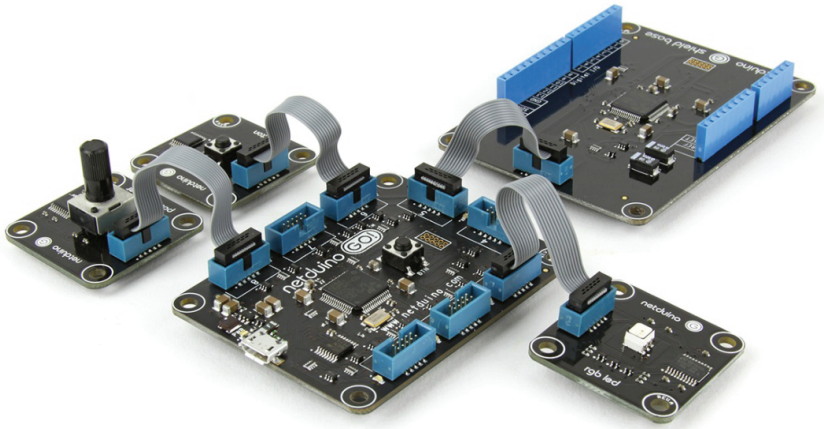
2 MIKROŘADIČE A VÝVOJOVÉ KITY PODPOROVANÉ NETMF

V současné době je NETMF podporováno na procesorech architektury ARM (ARM7, ARM9 a Cortex-M). Na trhu jsou k dispozici vývojové kity spolu se zdrojovým kódy, které jsou volně ke stažení na webové stránce Microsoft Download Center pod licencí Apache 2.0. Vývojové systémy produkuje několik hlavních dodavatelů, kteří vyrábějí čipy, vývojové kity a ostatní přídavná zařízení pro provoz pod platformou NETMF. Jedná se o společnosti Secret Labs (produkt Netduino), GHI Electronics (produkty EMX Module, ChipworkX Module, Embedded Master, USBizi144, USBizi100) nebo netmfdevices (platforma využívající FEZHacker) [2].

Zajímavým vývojovým systémem je hardwarová platforma .NET Gadgeteer, která je definována společností Microsoft. Platforma byla vytvořena k rychlé realizaci aplikací díky unifikaci designu komunikace mezi jednotlivými moduly (10 pinový konektor) a základní deskou. Na trhu je nabízeno několik variací zařízení typu Gadgeteer, například od společnosti GHI Electronics (produkty FEZ Spider, FEZ Hydra, FEZ Cerberus, FEZ Cerbuino Bee), od Sytech Designs systém NANO, Love Electronics vyrábí Argon R1 nebo produkty společnosti Mountaineer Group (Mountaineer Ethernet, Mountaineer USB) [2].

2.1 Netduino

Netduino je otevřená elektronická vývojová platforma založená na .NET Micro Frameworku. Používá 32 bitové mikrokontroléry architektury ARM. Základní desky systému Netduino (kromě typů Mini a Go) jsou navrženy tak, aby byly kompatibilní s většinou modulů vývojového systému Arduino. Aktuální model Netduino Go umožňuje připojit rozšiřující doplňkové moduly k základní desce pomocí konektorů. Každý modul má mikročip, který zprostředkovává spolupráci se základní deskou [3].

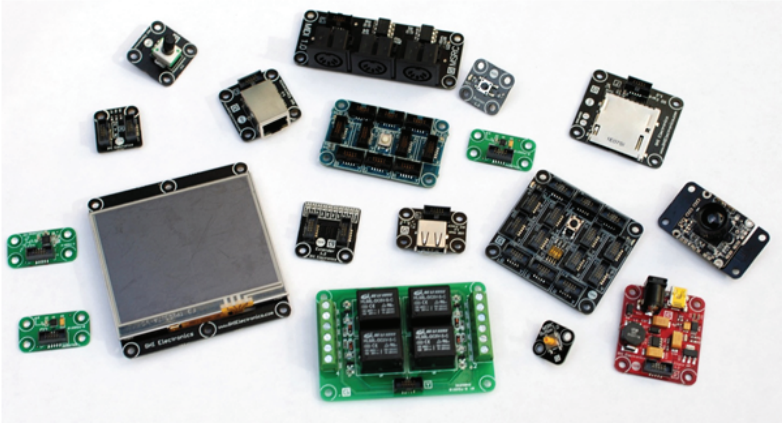


Obr. 1 Vývojový systém Netduino Go [4].

2.2 .NET Gadgeteer

Jednotky systému .NET Gadgeteer jsou programovány v jazyce C# pomocí intuitivního grafického průvodce v prostředí .NET Micro Frameworku. Celý systém se skládá ze základní desky obsahující procesor a různých modulů, které se připojují k základní desce prostřednictvím jednoduchého *„plug-and-play“* rozhraní. Řada výrobců produkuje širokou škálu hardwarových modulů, které slouží jako jednotlivé komponenty pro stavbu výsledných modulárních zařízení [5].

Existuje velké množství typů modulů, k dispozici jsou například jednotky jako displej, fotoaparát, síťové rozhraní, úložné jednotky a různé senzory. Konektory základní desky jsou očíslovány a označeny jedním nebo více písmeny, která značí, které moduly lze pomocí konektoru připojit. Každý modul také obsahuje konektor s informací, jaký druh základní desky vyžaduje [5].



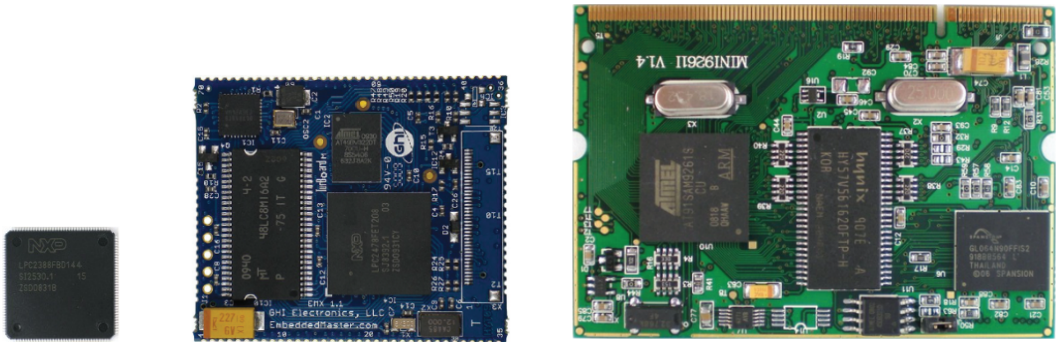
Obr. 2 Vývojový systém .NET Gadgeteer [5].

2.3 EMX Development System

Z výše popsaných zařízení je k realizaci diplomové práce použit právě EMX Development Systém, oficiální vývojový nástroj od společnosti GHI Electronics, který je určený pro mikroprocesorový modul EMX. Nabízí vysoký výkon a integruje v sobě množství využitelných periferií a rozhraní, které jej činí zajímavým pro všechny vývojáře i uživatele. Systém sestává z EMX modulu, desky vývojového kitu, 3,5“ TFT displeje a USB kabelu [1].

Výhodou pro uživatele je neustálá podpora od výrobce, průběžné doplňování nových knihovních funkcí a opravy knihoven zastaralých. Pomocí poskytovaných knihoven je usnadněna práce s pokročilými funkcemi. Jedná se například o síťové rozhraní Ethernetu, složitější práci s grafikou či programovou obsluhu dotykového rozhraní nejen u výrobcem dodávaných TFT displejů [1].

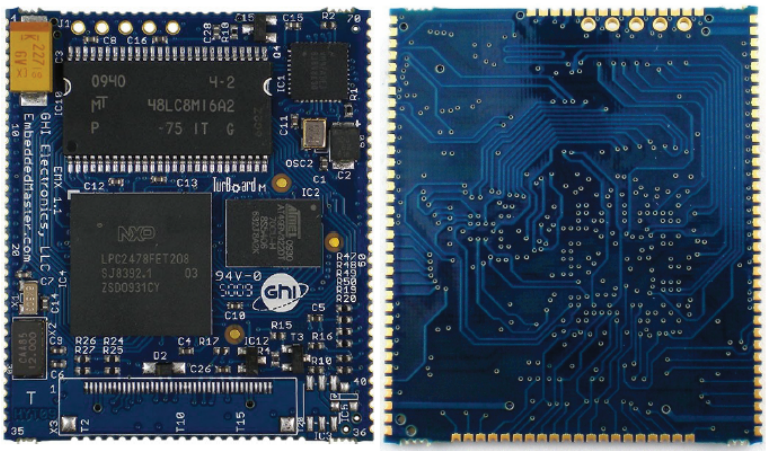
Uživatel potřebuje k zahájení vývoje vlastní aplikace EMX Development System, Microsoft Visual Studio 2010 s nejnovějšími aktualizacemi, Microsoft NETMF SDK verze 4.1 plus aktuální verzi GHI NETMF SDK přístupnou z webových stránek GHI Electronics. Pokud uživatel vlastní nový EMX Development System, je doporučeno aktualizovat TinyBooter a firmware [1].



Obr. 3 Vývojová zařízení GHI Electronics: USBizi, EMX, ChipworkX [1].

3 EMX MODUL

EMX modul sestává z kombinace hardwaru a softwaru (ARM procesoru, flash paměti, RAM paměti, fyzických vrstev periferií) na velmi malé (39 x 45 mm) osmivrstvé desce plošného spoje, která hostí NETMF s různými PAL/HAL ovladači. Dále obsahuje veškerý nutný software a hardware k provozování jednotlivých vyšších funkcí, jako jsou Ethernet, podpora USB hostitele, síťový protokol PPP atd. EMX modul je velmi komplexní, jeho složitost v důsledku poskytuje koncovému uživateli prostor pro jednoduchou implementaci aplikací. Stačí připojit napájení a lze začít se sofistikovaným vývojem [1].



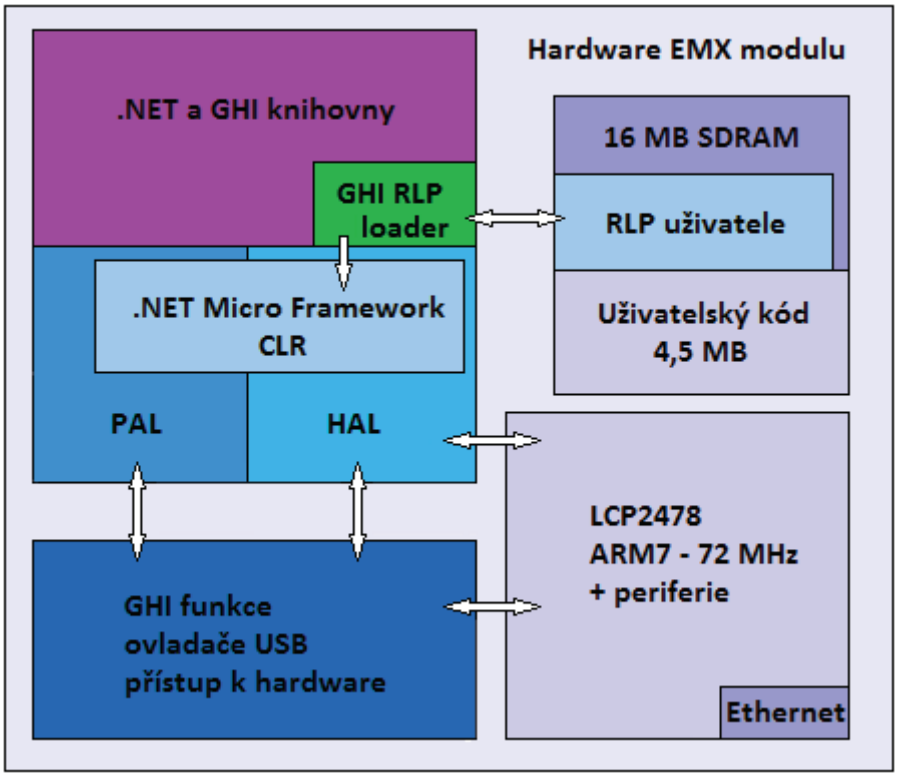
Obr. 4 EMX modul [1].

Jádrem modulu je LPC2478, což je 32 bitový mikroprocesor architektury ARM7 s taktem 72 MHz. Modul obsahuje rozhraní paměťové akcelerace, což umožňuje jádru procesoru pracovat bez zbytečných stavů čekání a spouštět aplikaci z externí paměti rychleji. Vnitřní flash paměť má velikost 0,5 MB a je plně využita k efektivnímu spuštění kompletního jádra NETMF. Mikropočítač obsahuje obvod RTC, který pracuje i při vypnutém procesoru, dále v sobě integruje širokou škálu periferií (PWM, GPIO, řadiče LCD, USB a další) [1].

Modul obsahuje 16 MB externí paměti typu SDRAM. K dispozici jsou 4 MB externí flash paměti, ve které již není zahrnuto výše zmíněných 0,5 MB. 1 MB externí paměti je využit pro zavaděč systémových souborů a dalších vnitřních zdrojů. Zbývající 3 MB jsou vyhrazeny pro uživatelskou aplikaci včetně všech jejích zdrojů. V paměti jsou vyhrazeny dva bloky o velikosti 128 kb, první pro dva sektory EWR, druhý je vyhrazen pro použití CLR. Dále 2 kB této paměti lze zálohovat baterií, data jsou tedy uchována i při ztrátě externího napájení [1].

Vestavěn je také řadič LCD displeje a řadič pro WiFi modul (v tomto případě pro ZG2100). K dispozici je 76 vstupně/výstupních pinů a 39 vstupů pro přerušení. Samozřejmostí je možnost připojení paměťové karty typu SD/MMC. Modul obsahuje šest PWM generátorů. Většina signálů na EMX modulu je multiplexována, aby bylo možné poskytnout více než jednu funkci pro každý z jeho pinů. Výběr funkce, kterou u daného pinu použít, je pouze na uživateli. GHI ovladače a NETFM kontrolují, aby systém nemohl využít dvě funkce na jednom pinu současně. Například analogový vstup (ADC3) a analogový výstup (AOUT) jsou na stejném pinu IO7, tudíž obě funkce nesmí být použity současně [1].

Maximální proudová spotřeba modulu činí 160 mA (jsou-li všechny periferie v provozu), spotřeba ve stavu hibernace je 40 mA. Modul lze provozovat v rozpětí teplot -40 až +85 °C. Příkladem využití modulu jsou aplikace časově řízených rutin, prodejní automaty, měřicí a testovací jednotky, síťová zařízení, robotika, GPS zařízení, zdravotnická výbava, bezpečnostní systémy, SMART zařízení, průmyslové automaty či jednotky Windows SideShow [1].



Obr. 5 Architektura EMX modulu [1].

3.1 Vstupy a výstupy

Všechny digitální I/O piny jsou 3,3 a 5 V tolerantní, což znamená, že signály z jiných obvodů (např. jiných mikropočítačů) mohou být až 5 V. Všechny I/O piny dále podporují použití „*pull-up*“ a „*pull-down*“ rezistorů pro vstup i výstup signálů. Většina digitálních I/O pinů také může sloužit k vyvolání externího přerušení, a tak asynchronně vyvolat v provozované aplikaci různé funkce. Přerušení může být aktivováno pomocí náběžné či sestupné hrany [1].

K dispozici jsou výstupní PWM generátory, které mají veškerý nutný hardware pro generování PWM signálů. Integrovány jsou dva nezávislé PWM časovače, které jednotlivé generátory sdílejí. Změna hodnoty frekvence jednoho generátoru má tedy vliv i na ostatní generátory. Jedná se o piny PWM0, PWM2 a PWM1, PWM3, PWM4, PWM5, které sdílejí stejný časovač, na což musí vývojář pamatovat. Pomocí funkce „*Output Compare*“ mohou uživatelé generovat signály různých průběhů. Tato funkce je k dispozici na každém výstupním digitálním pinu [1].

Analogové vstupy mohou číst napětí velikosti od 0 do 3,3 V s 10 bitovým rozlišením. Na analogové výstupy lze obdobně zapsat napětí v rozsahu od 0 do 3,3 V se stejným rozlišením. Čtené i generované napětí je napětí stejnosměrné. Analogové piny sice jsou 5 V kompatibilní, multiplexery jejich ADC převodníků však nejsou [1].

3.2 Sériové periferie

Nejjednodušší systémem podporovanou sériovou komunikací je „*One-wire*“. Prostřednictvím jediného vodiče může vedoucí zařízení komunikovat s více podřízenými zařízeními. „*One-wire*“ lze aktivovat na libovolném digitálním I/O pinu EMX modulu [1].

Dalším komunikačním protokolem je UART (USART). EMX modul pracuje se čtyřmi nezávislými sériovými porty (COM1 - UART0, COM2 - UART1, COM3 - UART2, COM4 - UART3). Hardwarový „*handshaking*“ je podporován pouze na portu COM2. Vývody sériového portu mají napěťovou úroveň 3,3 V (úroveň TTL logiky). Pro správnou funkci této komunikace s ostatními zařízeními je nutné zařadit ještě převodník napěťových úrovní (zde použit MAX232) [1].

Modul dále podporuje dvě SPI rozhraní, tedy SPI1 a SPI2, která jsou navržena tak, aby tvořila rozhraní od EMX směrem k několika SPI podřízeným zařízením. Aktivní zařízení je voleno pomocí vodiče „*chip select*“. Další podporovaná sériová komunikace je sběrnice I2C, což je dvouvodičové adresovatelné rozhraní. EMX podporuje pouze jeden I2C port [1].

EMX modul obsahuje dvě nezávislá CAN rozhraní. Ke komunikaci jsou zapotřebí pouze dva piny (TD - přenos dat a RD - příjem dat). Tyto piny pracují s digitálními signály, které se převádí na analogové signály pro přenos po médiu. Jednotlivá zařízení integrují různé CAN řadiče, nejběžnější z nich je dvouvodičové provedení schopné přenosu dat rychlostí až 1 Mb/s [1].

3.3 Grafický výstup a dotyková vrstva

EMX modul podporuje TFT displeje o barevné hloubce 16 bitů. Výchozí rozlišení je 320x240 pixelů, což odpovídá 3,5 palcům TFT displeje PT0353224T-A802, který je k dispozici přímo na vývojovém EMX kitu. Uživatelé mohou použít téměř libovolný TFT displej a připojit jej pomocí příslušných signálů z EMX modulu (jedná se o signály HSync, Vsync, CLK, Enable a 16 bitovou linku barev) [1].

Pro uživatele, kteří chtějí používat VGA monitory, EMX podporuje rozlišení 640x480 pixelů. V tomto případě je nutný jednoduchý rekodér pro převedení digitálních signálů na analogové RGB signály. Pokud je požadováno vyšší rozlišení než 640x480 pixelů, lze využít snímkových generátorů (například Chrontel CH7025). Maximální podporované rozlišení je 800x600 pixelů [1].

EMX Modul podporuje použití rezistivních dotykových vrstev. Čtení dotykových signálů probíhá pomocí čtyř vodičů (YU, YD, XL, XR). Uživatelé mohou pracovat mimo integrované dotykové vrstvy také s dotykovými vrstvami externích displejů a jejich dotykovými řadiči. V tomto případě je nutné naprogramování jednoduchého ovladače, který stanoví poziční parametry externího zařízení [1].

3.4 USB hostitel a klient

Modul EMX obsahuje USB klienta i USB hostitele, oba mohou pracovat současně. USB Klienti a hostitelé jsou zcela odlišná zařízení. Hostitelé pracují jako řídicí zařízení sběrnice. Klientská zařízení jsou ve srovnání s hostitelskými jednoduchá a mohou se pouze připojit a komunikovat se svým hostitelem [1].

USB hostitel umožňuje použití rozbočovačů, paměťových zařízení, joysticků, klávesnic, počítačových myší, tiskáren a dalších zařízení. S EMX ovladači se vývojář nemusí starat o vnitřní funkci komunikace. Pro USB zařízení, která nejsou standardní, je podporován i přístupu ke sběrnici na nízké úrovni [1].

USB klientské rozhraní je obvykle používáno jako vlastní rozhraní EMX pro napájení, přístup k ladění a nahrávání aplikace prostřednictvím Microsoft Visual Studia. Uživatel má však plnou kontrolu nad klientským rozhraním, například jej lze využít pro simulaci klávesnice či externích úložných zařízení. GHI Electronics dále nabízí pro usnadnění vývoje a poskytnutí přímé podpory pro vybraná zařízení knihovnu „*USB Client*“. Knihovna je schopna vytvořit klienta, který je složen z více rozhraní [1].

3.5 Souborový systém

Implementovaný souborový systém umožňuje modulu na připojené SD kartě nebo USB paměťovém zařízení pracovat se soubory a složkami. NETMF přímo podporuje souborové systémy FAT16 a FAT32. Před použitím paměťových zařízení je nutné jejich naformátování. SD karty a USB paměťová zařízení se nepřipojují a neodpojují automaticky. Modul podporuje paměťové karty typu SD (SD, mini SD a mikro SD), MMC, SDHC a všechna USB paměťová zařízení [1].

3.6 Síťové rozhraní

Klíčovou devizou dnešních vestavných zařízení je použití konvenčního síťového rozhraní. NETMF v sobě zahrnuje plnou podporou protokolu TCP/IP s kompletním vybavením pro obsluhu řízených aplikací. Rozhraní implementuje síťové protokoly PPP, SSL, HTTP, profil zařízení pro webové služby a standardy Ethernet a WiFi. Uživatelé mohou využít nástroj MFDeploy a aktualizovat MAC adresu či SSL „*seed*“ modulu ještě před jeho připojením do informační sítě. Nastavení síťové komunikace lze měnit dynamicky pomocí aplikačního kódu [1].

Modul v sobě integruje fyzickou vrstvu průmyslového Ethernetu spolu s veškerými potřebnými obvody. Oscilátor Ethernetu je řízen procesorem, což umožňuje uživateli řídit spotřebu energie modulu. Uživatel pouze propojí příslušné piny modulu ke konektoru sítě Ethernet. GHI Electronics dodává vyhrazenou MAC adresu pro každý EMX modul [1].

EMX dále podporuje WiFi přes externí modul, umožňuje uživateli připojení pomocí socketů. Lze použít až 127 TCP/UDP zásuvek s protokolem zabezpečení SSL, což klade vysoké nároky na použitý sériový most. WiFi modul splňuje normu IEEE 802.11bgn, obsahuje SPI rozhraní k hostitelskému procesoru či jinému zdroji dat. Dále v sobě integruje „*baseband*“ procesor, transceiver se zesilovačem, frekvenční reference, anténu, všechny potřebné WLAN protokoly, příslušné konfigurační funkce a obsahuje MAC adresu [1].

Při použití obou výše zmíněných standardů jsou podporovány protokoly DHCP a protokol IP (statický i dynamický). Pomocí protokolu PPP mohou uživatelé vytvářet sockety a komunikovat i přes linky, které nejsou přímo Ethernetové a zahrnují PPP klienta s protokolem ověřování PAP. Tato funkce umožňuje uživateli připojit se k Internetu nebo Extranetu přes sériový modem (V.90/GPRS/3G). V tomto případě se nastavení sítě získá z terminálového serveru hostitele [1].

3.7 Další funkce

EMX modul dokáže využívat různé režimy běhu procesoru s nízkým příkonem, je podporován stav hibernace (procesor je uspán). K jeho probuzení dochází v závislosti na specifické události v systému. Procesor LPC2478 dále obsahuje obvod hodin reálného času, který procesoru poskytuje různé alarmovací funkce [1].

Funkce zvaná „*In-Field Update*" umožňuje systémům automaticky aktualizovat jejich software. Této vlastnosti lze využít pro vzdálená zařízení u koncových uživatelů. Tímto způsobem lze aktualizovat celý firmware nebo pouze uživatelskou aplikaci. Velmi užitečná je také funkce RLP, která dovoluje uživatelům nahrát vlastní zkompilovaný nativní kód a spustit jej přímo přes aplikační kód. Tato funkce je obdobná použití DLL knihoven na klasickém PC. RLP mohou být použity ke zpracování časově kritických rutin [1].

Dále lze použít funkci „*watchdog*", která je nutná k resetování systému, pokud se tento dostane do chybného stavu. V neposlední řadě lze pomocí dodávané knihovny zakázat čtení nasazené uživatelské aplikace. Tato možnost je užitečná tehdy, potřebuje-li uživatel zajistit ochranu aplikace proti kopírování, nežádoucí manipulaci nebo zabránit jejímu odstranění [1].

3.8 Zavádění systému EMX modulu

Modul obsahuje tři hlavní části vestavěného software. Jedná se o GHI Boot Loader, EMX TinyBooter a EMX Firmware. Při startu systému Boot Loader inicializuje paměti flash a RAM. Poté je ve flash paměti vyhledán platný TinyBooter, který se posléze přesune a spouští z paměti RAM. Jakmile TinyBooter převezme kontrolu nad hardwarem, připraví veškeré nutné prostředky pro zavedení firmwaru. Firmware je tudíž hlavním softwarem, pod kterým pracuje celé jádro NETMF, a tedy i uživatelem řízená aplikace [1].

Přístupové rozhraní je definováno jako hardwarové rozhraní EMX modulu, které je používáno uživatelem pro přístup k Boot Loaderu, TinyBooter nebo firmwaru (TinyCLR). U EMX modulu může být tímto rozhraním USB nebo sériový port (COM1), zvoleno je hardwarově nastavením příslušného pinu [1].

Tab. 1 Základní rozdělení vestavěného software [1].

	GHI Boot Loader	**EMX TinyBooter**	**EMX Firmware**
Použití	Aktualizace TinyBooteru, nízko úrovňová údržba flash paměti	Aktualizace firmwaru, konfigurace údržba kódu aplikace	Nasazení, provádění a ladění aplikace, virtuální stroj
Aktualizace	Nelze aktualizovat	Lze aktualizovat	Lze aktualizovat
Rozhraní	USB, UART	USB, UART	USB, UART, Ethernet
Přístup	Příkazová řádka	MFDeploy	MS Visual Studio
Složitost	Jednoduchý, zvládá pouze funkce údržby	Kompaktní, zvládne i vyšší přiřazené funkce	Sofistikovaný, vyžaduje HAL a PAL ovladače

Během zavádění systému může uživatel zaváděcí sekvenci přerušit a zůstat na konfigurační úrovni Boot Loaderu, TinyBooteru nebo Firmwaru. K zastavení této sekvence dojde na základě stisku tlačítek na vývojovém kitu, dle požadavků na systém se jedná o tlačítka „*Up*“, „*Down*“ a „*Select*“ [1].

3.8.1 GHI Boot Loader

Boot Loader je součástí všech EMX modulů. Je používán k aktualizaci TinyBooteru nebo pro nízko úrovňovou údržbu paměti typu flash. Přijímá jednoduché příkazy v ASCII znacích, které mu byly poslány pomocí softwarového terminálu (TeraTerm). EMX modul při zavádění svého systému poskytuje na displeji veškeré potřebné informace o tom, jak si vybrat vstupní rozhraní pro přístup ke spuštění Boot Loaderu. Mezi příkazy patří například „*E*“, který smaže paměť vyjma Boot Loaderu, příkaz „*X*“, který načte nový TinyBooter nebo „*R*“ pro spuštění firmware [1].

Při spuštění systému Boot Loader převezme procesor a ověřuje platnost TinyBooteru uloženého ve flash paměti. Pokud byl nalezen TinyBooter a je platný, spouštění je převedeno na TinyBooter. Za obvyklých podmínek by uživatel neměl vyžadovat aktualizaci TinyBooteru, protože není používán ve finální aplikaci a neovlivňuje její běh. Využití aktualizace je pouze ve výjimečných případech, například při přechodu na jinou verzi NETMF [1].

3.8.2 EMX TinyBooter

Další potřebnou částí softwarové výbavy je TinyBooter od společnosti Microsoft, který může být použit k aktualizaci EMX firmwaru, k údržbě paměťového regionu určeného pro uživatelskou aplikaci, získání potřebných informací o systému nebo aktualizaci systémové konfigurace (například nastavení síťového rozhraní) [1].

EMX modul při zavádění svého systému poskytuje na displeji veškeré potřebné informace o tom, jak si vybrat vstupní rozhraní a jak přistoupit k TinyBooteru. Taktéž není nutné ve většině případů aktualizovat TinyBooter, není použit při běhu finální aplikace. Aktualizace je nutná pouze při přechodu na jinou verzi NETMF. Softwarové rozhraní pro komunikaci s koncovým uživatelem se nazývá MFDeploy [1].

3.8.3 EMX Firmware

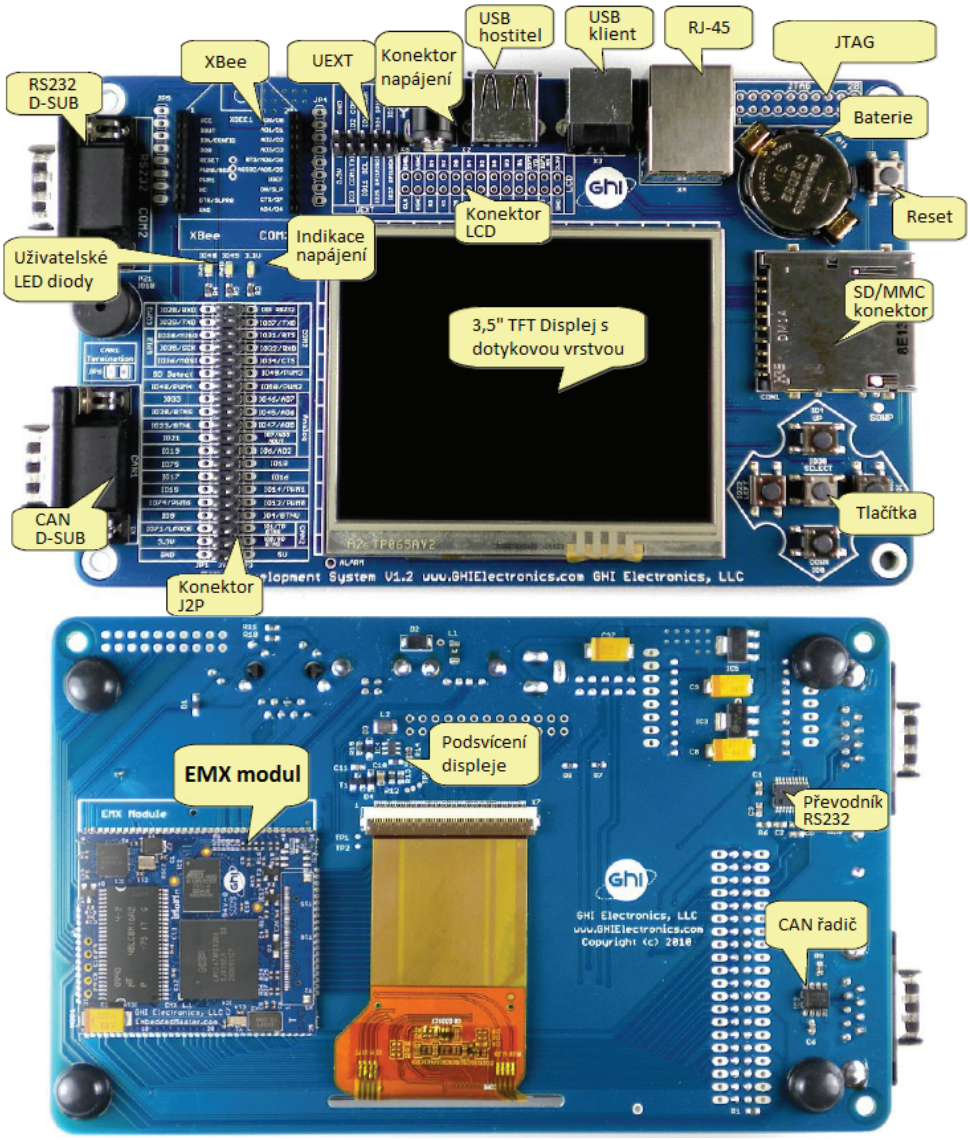
Z celé sady dodávaného software je firmware tím nejdůležitějším. Je hostitelem jádra NETMF s potřebnými ovladači vrstvy HAL, poskytuje modulu nejrůznější funkce, které může uživatel využít. Uživatel také nasazuje a ladí řídicí kód aplikace přímo na EMX modulu pomocí Visual Studia. Ladicí rozhraní je definováno mezi firmwarem a aplikačním kódem terminálu (Visual C# debugger). Ladění aplikace lze provádět pomocí USB, sériového portu nebo Ethernetu. Výchozím komunikačním rozhraním je USB. Pomocí příslušného software je možné aktivovat i jiná přístupová rozhraní, pokud je to dle charakteru aplikace nezbytné [1].

EMX modul při zavádění svého systému poskytuje veškeré potřebné informace o tom, jak si vybrat vstupní rozhraní pro přístup ke spuštění firmwaru. Uživatelé mohou dle potřeby aktualizovat firmware pomocí TinyBooteru. Koncový uživatel komunikuje s firmwarem přes grafické softwarové rozhraní zvané MFDeploy, které je dodáváno společně s NETMF SDK od společnosti Microsoft [1].

3.9 EMX Development Kit

EMX Modul je výhodné provozovat spolu s oficiálně distribuovaným vývojovým kitem, který rozvíjí jeho periferie a rozhraní, čímž tvoří výchozí bod pro práci vývojáře. Pro usnadnění použití periferií EMX Modulu je většina signálů (například GPIO, SPI a UART) vyvedena přes konektory k rychlému vytváření prototypů funkčních systémů a jejich následnému ladění [1].

Vývojový kit se skládá ze základní desky, která obsahuje EMX Modul, TFT displej s dotykovou vrstvou, Ethernetový konektor RJ-45, vyvedení signálů pro WiFi modul, JTAG konektor, vyvedení TFT signálů, vyvedení GPIO signálů i s přerušeními, vyvedení 4 UART linek (jedna linka RS232 s hardwarovým „*handshakingem*"), 7 analogových vstupů pro ADC (2 jsou použity pro dotykovou vrstvou displeje), 1 analogový výstup (DAC), CAN konektor, 6 PWM generátorů, slot pro SD/MMC karty, USB Device port, USB Host port, socket pro modul XBee, rozhraní UEXT pro snadné rozšíření kitu o moduly jako GPS, MP3 dekodér nebo tříosý akcelerometr, baterii, LED diody a tlačítka, piezokrystal a 2,1 mm vstup pro stejnosměrné napětí 6 V [1].



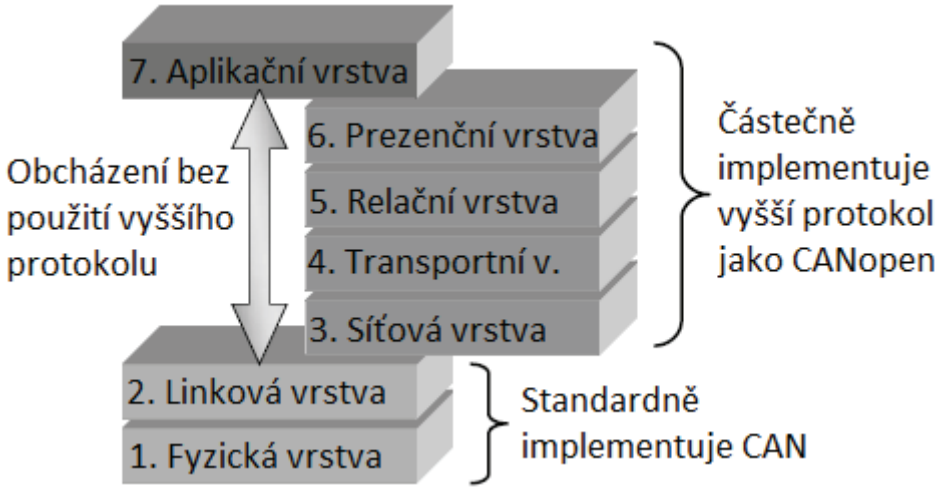
Obr. 6 Vývojový kit EMX (přední a zadní strana) [1].

4 SBĚRNICE CAN

CAN je velmi flexibilní komunikační sběrnice, která může být fyzicky implementována na různých přenosových médiích, mezi které patří například kroucená dvojlinka, silové vedení nebo i optická dráha. Sběrnice je typu „*multimaster*“, každý komunikační uzel může začít zasílat svá data kdykoli. Původním určením této sběrnice je vnitřní automobilová komunikace, kde mnoho senzorů potřebuje svá data zasílat poměrně často. V důsledku toho CAN komunikuje pomocí malých datových rámců, které obsahují maximálně 8 datových bajtů. Datovou režii na každou zprávu tvoří 11 bitový identifikátor a 15 bitový kontrolní součet (CRC). Na nejnižší úrovni jsou všechny datové rámce zasílány jako „*broadcast*“, každý uzel na sběrnici obdrží každou zprávu. Teprve poté uzel vyhodnotí, zda je zpráva určena pro něj či nikoliv. Na vrcholu komunikační vrstvy jsou distribuovány vyšší protokoly různých norem, také je vytvářen nespočet interních firemních standardů [6].

4.1 Upravený model ISO/OSI

Síťový model dle standardu ISO/OSI definuje 7 vrstev, počínaje fyzickými přenosovými médii až po aplikační rozhraní. Tato tradiční implementace vyžaduje rozhraní mezi jakýmikoliv dvěma sousedními vrstvami, což je pro vestavné systémy režijně nepřijatelné. Dále je třeba poznamenat, že ne všechny vrstvy modelu ISO/OSI jsou pro sběrnici CAN implementovány [6].



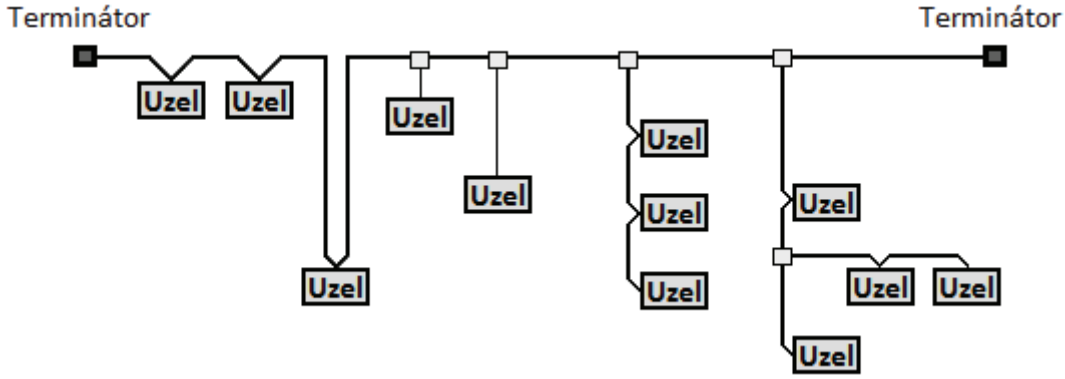
Obr. 7 Upravený model ISO/OSI pro sběrnici CAN [6].

Většina „*on-chip*“ komunikačních rozhraní obvykle implementuje pouze funkčnost první vrstvy (vrstvy fyzické). CAN částečně implementuje také funkce druhé vrstvy (vrstvy linkové). K dispozici jsou funkce pro zasílání a příjem dat, ale není definováno, kdy a jak zprávy cestují po sběrnici a jaký druh informací obsahují. Funkčnost vyšších vrstev je obvykle realizována softwarovou cestou a implementuje pouze vybrané funkce vyšších vrstev, čímž se snižuje zatížení komunikace [6].

Neexistuje standardizované softwarové rozhraní. Standardy nebo jejich části, které toto rozhraní implementují, se nazývají protokoly vyšší vrstvy. Tyto protokoly určují datové typy, identifikátory zpráv pro specifické služby, způsob reprezentace jednotlivých datových hodnot atd. Výhodou tohoto přístupu je zamezení připojení jakýchkoliv modulů třetích stran [6].

4.2 Topologie sběrnice a fyzická vrstva

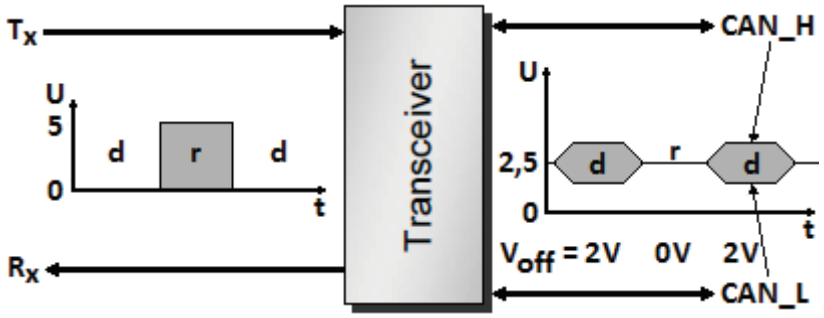
Fyzické rozložení sběrnice CAN je lineární. Hlavní páteř se skládá ze dvou vodičů, signálů CAN_L a CAN_H. Vedení musí mít na každém konci terminační odpory. Doporučeno je použití 120 Ω pro přenosovou rychlost 1Mbps, pomalejší a delší sběrnice vyžadují terminační odpory v rozmezí 150 až 300 Ω [6].



Obr. 8 Topologie sběrnice CAN [6].

Maximální délka sběrnice je závislá na zvolené rychlosti. Pro přenosovou rychlost 1 Mb/s je maximální délka kolem 40 m. Větší vzdálenosti jsou dosažitelné při nižších rychlostech, například 500 m při přenosové rychlosti 125 kb/s. Při výpočtu maximální délky sběrnice však existuje více faktorů. Kromě vodivostních vlastností kabeláže a konektorů se také jedná o počet připojených uzlů nebo zpoždění vysílačů jednotlivých uzlů. Čas vysílání jednoho bitu nemůže být kratší než doba, kterou vyžaduje signál pro cestu přes řadič na sběrnici, přesun na nejvzdálenější konec sběrnice, opětovnou cestu přes řadič a celou cestu zpět [6].

Transceiver z hlediska fyzické vrstvy přijímá TTL signál přicházející na vstupní komunikační pin a převádí jej na diferenciální signál mezi dvěma vodiči informačního vedení (vodiče CAN_L a CAN_H). Obdobně jsou diferenciální signály převedeny zpět do úrovně TTL a vráceny na výstupní pin řadiče k jejich odběru [6].



Obr. 9 CAN transceiver [6].

4.3 Kabeláž a konektory

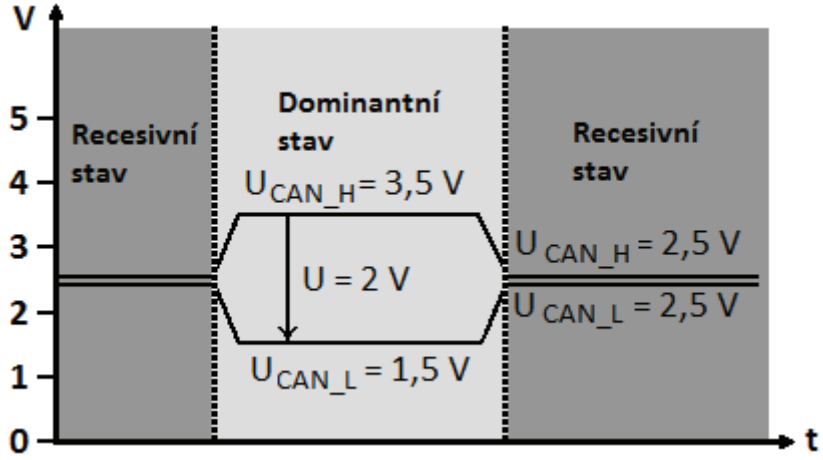
Typickým nosičem CAN signálů je zkroucená dvojlinka. Kromě mnoha variant vedení pomocí kroucených dvojlinek existují i aplikace, které používají ploché kabely, telefonní kabely, sériové kabely od PC, kabeláž Ethernetu či Firewire a další. Diferenciální signál dává sběrnici CAN dobrou úroveň ochrany proti EMI. Pro elektromagneticky rušená prostředí je doporučeno stínění kabelů [6].

Společným prvkem všech rozvodů je fakt, že CAN je sběrnice dvouvodičová. Pro spolehlivý provoz je vhodné přidat ještě třetí vodič společné země. Mnoho aplikací používá sběrnicové vedení i pro rozvod napájecího napětí jednotlivých zařízení. Takto vzniká nejvíce populární, tedy čtyřvodičová varianta. Skládá se z jedné stíněné kroucené dvojlinky pro CAN signály a dalšího páru vodičů pro společný rozvod země a kladného napětí [6].

Mezi nejčastěji používané konektory se řadí D-Sub, Muti-Pole, Dual Header Row, nebo konektory typu RJ10 a RJ45. Signály stanovené pro obecný konektor jsou následující: CAN_L, CAN_H, CAN GND, CAN stínění (volitelné), kladné napětí (volitelné) a GND (volitelné) [6].

4.4 Elektrické signály

Logická hodnota 1 na sběrnici tvoří recesivní stav, který je reprezentován nulovým rozdílem mezi vodiči CAN_L a CAN_H. Oba vodiče se v tu chvíli nachází na shodné napěťové úrovni okolo 2,5 V. Logická hodnota 0 tvoří dominantní stav. Ten je reprezentován 2 V napěťovým rozdílem mezi vodiči. Napětí na CAN_L se tedy o 1 V sníží, kdežto napětí na CAN_H se o 1 V zvýší. Není definován žádný třetí stav, který by indikoval sběrnici v nečinnosti, používají se rozsáhlé periody stavu recesivního [6].



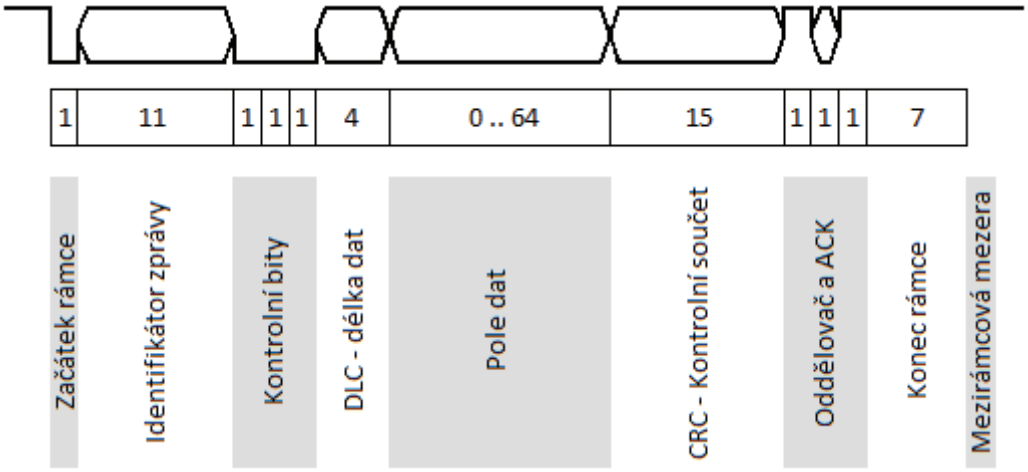
Obr. 10 Elektrické signály vodičů sběrnice CAN [6].

Dominantního signál na sběrnici přepíše signál recesivní. Pokud alespoň jeden uzel zapíše dominantní stav, celá sběrnice se nachází v dominantním stavu. Tento mechanismus se používá k detekci kolizí. Komunikační uzel zapisující recesivní stav a odčítající zpět dominantní stav tak zjistí, že došlo ke kolizi [6].

4.5 Formát datového rámce

Jak je výše zmíněno, v nečinnosti sběrnice zaujímá stav recesivní. Děje se tak z důvodu, aby byl každý přechod do dominantního stavu považován za začátek vysílání datového rámce libovolným uzlem. Datový rámec začíná dominantním start bitem. Následuje 11 bitový identifikátor zprávy, další tři bity jsou rezervovány a nastaveny na nulovou hodnotu. Dále se odesílá DLC, což je 4 bitová hodnota určující počet datových bajtů. Pro DLC jsou povoleny hodnoty 0 až 8 (0, 8, 16, 24, 32, 40, 48, 56 nebo 64 datových bitů). Následuje datový blok specifikované délky a 15 bitů cyklické redundantní kontroly [6].

Zbývající tři bity jsou CRC oddělovač, ACK (potvrzení) a ACK oddělovač. Oddělovače se používají z toho důvodu, aby všechny uzly měly čas na zpracování příchozích bitů a vytvoření odpovídající reakce. Celý datový rámec končí sledem sedmi po sobě jdoucích recesivních bitů [6].



Obr. 11 Formát datového rámce sběrnice CAN [6].

4.6 Kolize a jejich řešení

Jedním z ústředních prvků sběrnice CAN je způsob řešení kolizního stavu. Postup je velmi podobný jako u Ethernetu (CSMA/CD), každý uzel neustále odposlouchává veškeré dění na sběrnici. Jelikož může mít současný přístup k nosnému médiu více uzlů najednou, kolize nastane právě tehdy, pokud současně začne vysílat více uzlů. Na rozdíl od Ethernetu, kde by byla vyprodukována rušící sekvence bitů, jsou kolize na sběrnici CAN řešeny okamžitě [6].

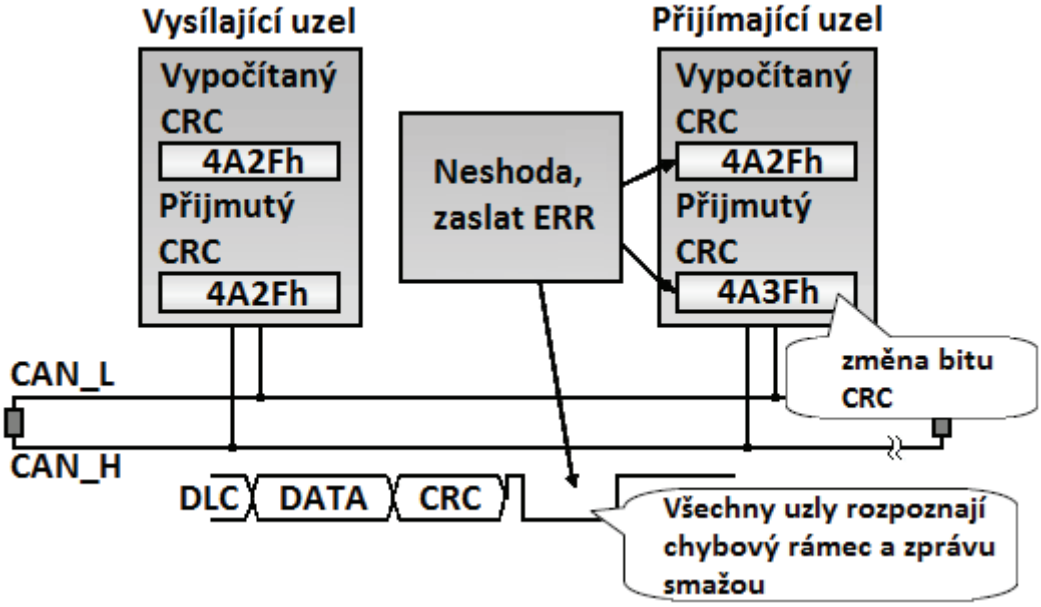
Existují čtyři klíčové body řešení kolizí. Identifikátory zpráv musí být v celé komunikační síti jedinečné. Identifikátor zpráv je přiřazen konkrétnímu uzlu a pouze ten jej může zprávě předat. Zapsání dominantního stavu na sběrnici přepíše stav recesivní. Každý uzel zapisující na sběrnici zároveň i odečítá kvůli zjištění informace, zda byl jeho zápis úspěšný nebo byl někým přepsán [6].

Pokud se pokusí vysílat více uzlů současně, kolize nastane na některé pozici 11 bitového identifikátoru zprávy. Uzly, které zapisují recesivní stav ze svého identifikátoru a přečtou zpět stav dominantní tak zjistí, že jiný uzel přepsal jejich bit stavem recesivním. Tímto zjištěním uzel okamžitě prohrává vzniklý spor. Odstoupí od vysílání a po odeslání kolizního rámce s vyšší prioritou začne přijímat zprávu od uzlu, který tuto kolizi vyhrál. Celý proces je okamžitě opakován se všemi zprávami, které předchozí spor prohrály. Opět vyhraje další uzel s nejvyšší prioritou a tak dále. Tímto mechanismem se kolize zúčastněných uzlů postupně rozřeší [6].

4.7 Detekční mechanismus chyb

Součástí protokolu CAN je implementace velmi propracovaného systému zachytávání chyb. Většina kroků je prováděna hardwarově přímo v řadiči, není ovlivněna či řízena aplikačním programem. Každý uzel sběrnice aktivně monitoruje její činnost a výpočtem kontroluje CRC každé zprávy. Dokonce i uzel, který zprávu odesílá, při jejím zápisu vytváří kontrolní součet. CRC pokrývá všechny bity od prvního bitu identifikátoru zprávy až po poslední bit datového bloku [6].

Jakýkoliv uzel na sběrnici, který obdrží libovolný datový rámec, vypočítá jeho kontrolní součet a porovná jej s tím, který obdržel prostřednictvím zprávy. Pokud se tyto dva kontrolní součty shodují, zapíše na sběrnici dominantní stav jako potvrzení, že proces příjmu zprávy byl úspěšný. Pokud alespoň jeden uzel zaznamená nesoulad kontrolních součtů, zapíše na sběrnici stav recesivní, čímž neguje všechna případná pozitivní potvrzení od ostatních uzlů. V důsledku toho nesouladu se generuje chybový rámec, který je odeslán všem uzlům sběrnice. Každý uzel uzná nebezpečí vzniklé chyby a aktuální zprávu smaže. Po vypršení časové mezery mezi rámci se aktivní uzel automaticky pokusí předat svou zprávu znova [6].



Obr. 12 Kontrola CRC každé zprávy na sběrnici CAN [6].

V každém řadiči se nachází dva chybové čítače. Jedná se o čítač chyb v příjmu zpráv (REC) a čítač chyb v jejich zasílání (TEC). Tyto se s každým detekovaným chybným stavem daného uzlu inkrementují o určitou hodnotu. Čím je chyba vážnější, tím je hodnota přírůstku vyšší. Na druhou stranu lze tyto čítače také dekrementovat, k čemuž dochází s každou úspěšně přijatou či odeslanou zprávou [6].

Každý CAN řadič se v návaznosti na práci chybových čítačů může nacházet ve třech režimech. Jedná se o režimy „*Error Active*“, „*Error Passive*“ a „*Bus Off*“. Výchozím nastavením každého komunikačního uzlu, získaným po zapnutí nebo restartu CAN řadiče, je režim „*Error Active*“. V tomto režimu uzel aktivně vykonává všechny výše popsané CRC výpočty a srovnání [6].

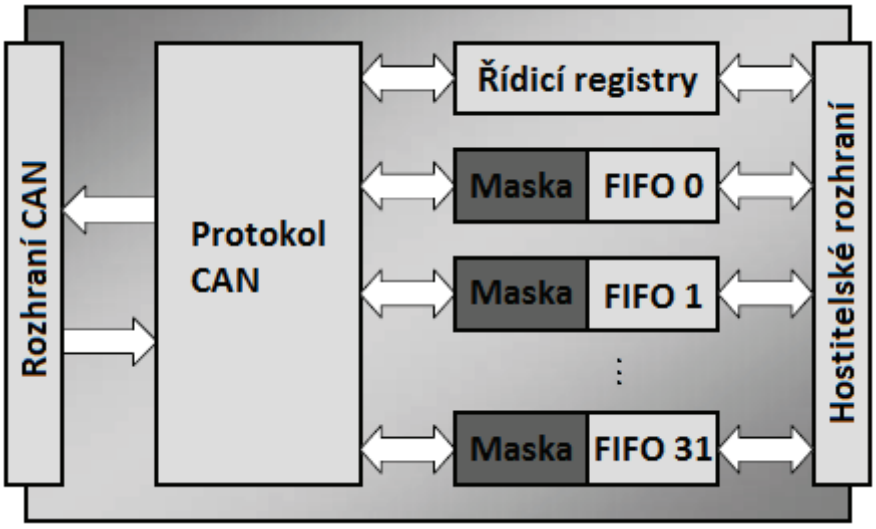
Pokud kterýkoliv ze dvou čítačů libovolného uzlu dosáhne hodnoty 127, řadič přejde do režimu „*Error Passive*“. V tomto režimu může uzel nadále odesílat a přijímat zprávy, omezením je pouze to, že na sběrnici aktivně nezasahuje do ověřování správnosti příjmu datových rámců. Vzhledem k možnosti dekrementace čítačů se mohou uzly z dočasné síťové poruchy bezezbytku zotavit. Pokud se hodnoty chybových čítačů nadále zvyšují a některý čítač přeteče (hodnota větší než 255), uzel přejde do stavu „*Bus Off*“, CAN řadič se kompletně odstaví. Z tohoto stavu lze řadič opět aktivovat pouze jeho reinicializací [6].

4.8 CAN řadiče

Rozhodujícím kritériem pro výběr CAN řadiče je požadovaný komunikační výkon, který závisí na konkrétní aplikaci a její realizaci. Pro řadič je výpočetně nejhorší možnou variantou následující případ. Přenosová rychlost sběrnice je nastavena na 1 Mb/s, nejdelší zpráva obsahuje 8 datových bajtů. Nejkratší zpráva (obsahující nula datových bajtů) trvá na sběrnici 50 bitových časových jednotek, což při 1 Mb/s odpovídá právě 50 μs. Jelikož cílem každé aplikace je reagovat na komunikační události v reálném čase, je nutné, aby zpracování nejdelší příchozí zprávy s 8 datovými bajty proběhlo za méně než výše zmíněných 50 μs [6].

Moderní CAN řadič obsahuje hardwarové filtry a buffery. Díky ignoraci nechtěných zpráv a ukládání přijatých zpráv do vyrovnávací paměti dochází ke snížení zatížení interní komunikace řadiče. Při příjmu zprávy se identifikátor (někdy i nesená data) porovnávají s konfigurací filtru. Pouze v případě, že příchozí zpráva odpovídá filtru, se tato uloží do vyrovnávací paměti. Hlavními rozdíly v hardwarových filtrech jsou datová šířka (standardní CAN vyžaduje 11 bitů) a zda je filtr nastavený pouze na shodu nebo také umožňuje použití masky. Maska nastavuje bity identifikátoru uzlů, které řadič zajímají, na požadovanou hodnotu. K výběru rozsahu požadovaných adres lze použít vhodnou kombinaci filtru a masky [6].

Nejčastěji používaným typem CAN řadiče je typ „*Full CAN*“ s FIFO buffery pro každou příchozí zprávu. Ačkoli je toto řešení velmi robustní, je také nejnáročnější na konfiguraci, zejména pokud je každý vstupní FIFO buffer náhodně umístěn v paměti RAM a může nabývat různé délky [6].



Obr. 13 Blokové schéma Full CAN řadiče s FIFO buffery [6].

5 VYŠŠÍ PROTOKOLY SBĚRNICE CAN

Sběrnice CAN definuje komunikační hardware (fyzickou vrstvu) a pracuje s daty pouze na jejich základní úrovni (vrstva linková). CAN sám o sobě pouze určuje, jakým způsobem pomocí sdíleného komunikačního média dopravit datové rámce z bodu A do bodu B. Neobsahuje žádné informace o nesených datech a jejich správě. Neřeší řízení datového toku, dopravu větších zpráv než standardních 8 bajtových, nezná adresy uzlů, nestará se o navázání komunikace a tak dále [7].

Pro plnou funkčnost komunikace je nutné implementovat vyšší komunikační protokol. Neexistují žádná vyšší standardizovaná softwarová rozhraní. Za vyšší protokoly je možné považovat jednotlivé standardy nebo jejich části, které toto rozhraní tvoří. Úkolem protokolu je určovat datové typy, identifikátory zpráv pro specifické služby, způsob reprezentace jednotlivých datových hodnot atd. Mnoho průmyslových aplikací používá soukromé firemní standardy [6].

Mezi vyšší komunikační standardy se řadí například CanKingdom, CANopen, CCP/XCP, DeviceNet, OSEK/VDX nebo standard SDS. Z důvodu použití motorů Faulhaber, které ke komunikaci využívají z výše jmenovaných protokol CANopen, následující část práce přibližuje právě jeho funkci.

5.1 CANopen pro řízení pohybu

CANopen byl navržen jako vyšší vrstva protokolu CAN a pracuje pomocí jeho hardware. Mezinárodní organizace CiA (CAN in Automation) ve standardu DS301 definuje komunikační profily (komunikační strukturu, metody pro parametrický přístup, kontrolu a monitorovací funkce). Tyto profily jsou určeny pro různá zařízení, například norma DSP402 je určena pro pohony, norma DS401 pro I/O zařízení [8].

Veřejná data protokolu jsou spravována pomocí objektového slovníku. Slovník obsahuje tabulku parametrů, přístup k jednotlivým položkám je realizován pomocí indexů a subindexů. Protokol má k dispozici dva datové komunikační objekty, tedy PDO (řízení a monitoring sběrnice) a SDO (definované pro přístup k jednotlivým objektům slovníku). Jsou definovány i další speciální objekty, které jsou určeny pro správu a údržbu komunikační sítě, monitorování činnosti jednotlivých uzlů či objekty synchronizační. Vlastní komunikace je založena na zasílání zpráv. Každý komunikační objekt má přiřazen vlastní 11 bitový identifikátor. Protokol podporuje až 127 uzlů na segment sítě s přenosovou rychlostí až 1 Mb/s [8].

5.1.1 Objektový slovník

Objektový slovník (OD) je záznamová tabulka, která obsahuje všechna dostupná data. Každý uzel na sběrnici musí implementovat vlastní objektový slovník, který obsahuje popis konfigurace sběrnice. Obsažené údaje mohou být čteny a přepisovány do jiných uzlů. Slovník obsahuje specifické aplikační informace daného uzlu. Ostatní uzly na sběrnici mohou přečtením položky ze slovníku určitého uzlu zjistit informace o jeho stavu nebo činnosti. Jednotlivé položky ve slovníku se mohou lišit dle výsledné aplikace a jejího využití, některé položky jsou však povinné a musí být přítomny. Zapisování dat do položek slovníku uzlu může znamenat také externí příkaz k provedení určité operace, například změření aktuální hodnoty a její aktualizaci ve slovníku [6].

Každá položka slovníku má 16 bitový index a může obsahovat až 256 vnořených subpoložek, na které lze odkazovat pomocí 8 bitového subindexu. Každá položka povinně obsahuje alespoň jednu subpoložku. Ne všechny položky slovníku jsou implementovány nebo používány, v tabulce dochází k vytvoření prázdných záznamů. Běžnou praxí je použití čísel šestnáctkové soustavy pro adresaci jednotlivých záznamů (indexy i subindexy) [6].

V protokolu CANopen společnosti Faulhaber je slovník rozdělen do tří hlavních oblastí. Jedná se o parametry komunikační (indexy 0x1000 až 0x1FFF), oblast volně specifikovatelnou (indexy 0x2000 až 0x5FFF) a standardizované profily jednotlivých zařízení (indexy 0x6000 až 0x9FFF).

První oblast obsahuje objekty dle normy DS301, druhá oblast je vyhrazena pro specifické objekty výrobce a poslední oblast obsahuje objekty dle normy DSP402 (specifické objekty zabývající se jednotkami řízení pohybu značky Faulhaber). Níže je uvedena ukázka objektů OD dle DS301 a řídicího profilu OD dle DSP402 [8].

Tab. 2 Výběr z komunikačních objektů normy DS301 [8].

Index	Objekt	Jméno	Typ	Atribut
0x1000	VAR	Typ jednotky	UNSIGNED32	RO
0x1001	VAR	Registr chyby	UNSIGNED8	RO
0x1008	VAR	Jméno jednotky	STRING	KONST
0x1010	ARRAY	Skladovací parametry	UNSIGNED32	RW

Tab. 3 Výběr z objektů řídicího profilu normy DSP402 [8].

Index	Jméno	Typ	Atribut	Význam
0x6040	Kontrola	Unsigned16	RW	Řízení stavu
0x6041	Status	Unsigned16	RO	Zobrazení stavu
0x6061	Pracovní režimy	Integer8	RO	Nastavení režimu
0x606B	Požadovaná rychlost	Integer32	RO	Požadovaná rychlost
0x606C	Aktuální rychlost	Integer32	RO	Aktuální rychlost
0x606D	Okno rychlosti	Unsigned16	RW	Ukončení okna
0x606F	Práh rychlosti	Unsigned16	RW	Hodnota prahu
0x607F	Maximální rychlost	Unsigned32	RW	Hodnota rychlosti
0x6096	Faktor rychlosti	ARRAY Unsigned32	RW	Faktor rychlosti
0x60F9	Parametry rychlosti	ARRAY Unsigned16	RW	Parametry

5.1.2 Objekty procesních dat

Objekty procesních dat (PDO) také odpovídají datovému rámci komunikace CAN. Zprávy mohou obsahovat až 8 bajtů a jsou používány pro přenos procesních dat, tedy k řízení a monitorování činnosti jednotlivých uzlů. Přijatá PDO obsahují řídicí údaje, odesílaná PDO obsahují údaje povahy monitorovací. Existují tři typy komunikace pomocí těchto objektů. PDO řízené událostí, kdy jsou data automaticky zaslána po změně stavu zařízení. Dále vyžádaná data, která jsou zaslána jako odpověď na příslušný rámec. Poslední způsob je synchronní, kdy jsou data odeslána po obdržení synchronizačního objektu [8].

Objekty typu PDO jsou následující [8]:

- *„Kontrola“* obsahuje 16 bitů, řídí stavový automat jednotky.
- „*Status*“ obsahuje 16 bitů, zjišťuje stav automatu jednotky.
- *„Faulhaber příkaz“* předává výrobcem specifikované příkazy. Rámce jsou dlouhé vždy 5 bajtů, přičemž první bajt obsahuje příkaz a následující 4 bajty předávají argument ve formě čísla.
- *„Faulhaber data“* zasílají dotazovací příkazy. Rámce jsou dlouhé vždy 6 bajtů, přičemž první bajt obsahuje příkaz a následující 4 bajty udávají požadovanou hodnotu ve formě čísla.
- *„Konfigurace trasování“* se používá k nastavení trasovacího režimu, pomocí kterého lze rychle přečíst vnitřní parametry jednotky. Obsahuje informaci o počtu paketů, které mají být na základě požadavku předány, také časový interval mezi těmito pakety.
- „*Trasovací data*“ vrací trasovací údaje

Tab. 4 Objekty typu PDO [8].

Název	Identifikátor	Bajty zprávy
Kontrola	0x200(512D) + ID uzlu	LB, HB
Status	0x180(384D) + ID uzlu	LB, HB
FAULHABER příkaz	0x300(768D) + ID uzlu	CMD, LLB, LHB, HLB, HHB
FAULHABER data	0x280(640D) + ID uzlu	CMD, LLB, LHB, HLB, HHB, ERR
Konfigurace trasování	0x400(1024D) + ID uzlu	Mód1, Mód2, TC, Pakety, Čas
Trasovací data	0x380(896D) + ID uzlu	Data0, Data1, Data2 … Data7

5.1.3 Objekty servisních dat

Servisní objekty (SDO) mohou být použity pro čtení nebo popis parametrů jednotlivých objektů umístěných v objektovém slovníku daného uzlu. Jednotlivé položky slovníku jsou přístupné pomocí 16 bitového indexu a 8 bitového subindexu. Řídicí jednotka se chová jako server, na žádost klienta vyžádané položky zasílá („*upload*“), nebo je může od klienta obdržet („*download*“). Rozlišuje se mezi dvěma typy přenosu objektů SDO. První možností je urychlený přenos (maximálně 4 bajty) nebo segmentovaný přenos, který využívá přenos více než 4 bajtů [8].

Tab. 5 Objekty typu SDO [8].

Název	Identifikátor	Bajty zprávy
Požadavek na „*upload*“	0x600(1536D) + ID uzlu	0x40, index LB, index HB, subindex, 0, 0, 0, 0
Odpověď na „*upload*“	0x580(1408D) + ID uzlu	0x4x, index LB, index HB, subindex LLB, LHB, HLB, HHB
Požadavek na „*download*“	0x600(1536D) + ID uzlu	0x2x, index LB, index HB, subindex LLB, LHB, HLB, HHB
Odpověď na „*download*“	0x580(1407D) + ID uzlu	0x60, index LB, index HB, subindex, 0, 0, 0, 0
Chybová událost	0x600(1536D) + ID uzlu	0x80, index LB, index HB, subindex, err0, err1, err2, err3

5.1.4 Krizové objekty

Krizové objekty informují ostatní zařízení na sběrnici o chybách, které se během komunikace vyskytly. V 11 bitovém identifikátoru zprávy je umístěna hodnota 0x80(128D) + číslo uzlu, velikost datové části objektu je vždy stejná (8 bajtů). První dva bajty obsahují kód chyby, třetí bajt obsahuje registr chyby. Následujících 5 bajtů je přiděleno výrobci pro bližší specifikaci chybového stavu. Tyto rámce identifikuje takzvaný registr chyb.

Chyby, které mohou při komunikaci nastat, jsou popsány v objektovém slovníku pod indexem 0x1001. Následující bity rámce blíže specifikují chybový stav. Pro ukázku je uvedena tabulka základního rozdělení chyb [8].

Tab. 6 Základní rozdělení krizových objektů [8].

Kód chyby	**Význam**
0x0000	Žádná chyba
0x1000	Generická chyba
0x2000	Chyba proudu jednotky
0x3000	Chyba napětí jednotky
0x4000	Chyba teploty jednotky
0x5000	Chyba hardware jednotky
0x6000	Chyba software jednotky
0x8000	Chyba monitoringu jednotky

Toto základní rozdělení lze rozvést na konkrétní chybové stavy určené pomocí masky a bitového registru. Níže je rozvedena tabulka chyb monitoringu zařízení.

Tab. 7 Základní rozdělení krizových objektů [8].

Kód chyby	**Význam**	**Maska**	**Bitový registr**
0x8100	Chyba komunikace		
0x8130	Chyba funkce zařízení	0x0100	4
0x8140	Navrácení z režimu „*Bus Off*“	0x0200	4
0x8400	Odchylka rychlosti	0x0002	5
0x8600	Odchylka pozice		
0x8611	Sledovací odchylka	0x0002	5

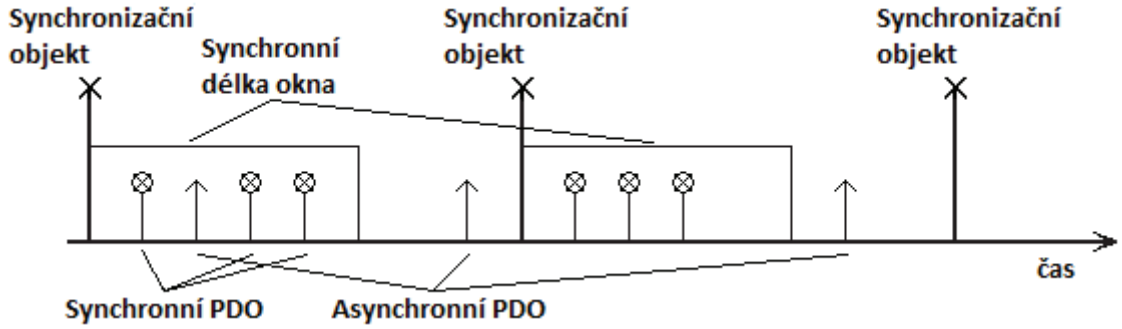
5.1.5 Synchronizační objekty

Synchronizační objekt je krátký rámec bez obsahu jakýchkoliv dat, který se používá ke spuštění objektů typu PDO. Této funkce lze využít například k pseudo současnému spouštění několika různých procesů na zcela odlišných zařízeních. Identifikátor synchronizačního objektu může být vyhledán pomocí objektového slovníku podle indexu 0x1005. Přenosy objektů typu PDO lze rozdělit dle následující tabulky [8].

Tab. 8 Rozdělení synchronizačních objektů [8].

Typ přenosu	**Význam**
255	Asynchronní zaslání PDO
253	Asynchronní zaslání PDO, pouze na požadavek
252	Synchronní zaslání PDO, pouze na požadavek
1–240	Synchronní cyklické zasílání PDO
0	Synchronní acyklické zaslání PDO

Přenos typu 1–240 může být také využit pro skupinu uzlů. Uzlům je umožněn synchronní příjem objektů PDO. Příkaz, který byl již dříve předán pomocí objektu typu PDO, není spuštěn do té doby, dokud není obdržen odpovídající synchronizační objekt. Tímto způsobem lze například synchronizovat pohyb stroje ve více osách. Také je možné využít synchronního zasílání objektů PDO. Po obdržení synchronizačního objektu je PDO s aktuálními údaji odesláno co možná nejrychleji [8].



Obr. 14 Synchronní zasílání objektů PDO [8].

5.1.6 Sběrnicový management

Jednotky se po zapnutí a provedení inicializace automaticky vyskytují v předoperačním režimu. V tomto režimu lze s jednotkou komunikovat pouze pomocí zpráv sběrnicového managementu (NMT) nebo pomocí datových objektů (SDO), které slouží výhradně k nastavení či odečtu parametrů jednotky [8].

V protokolu CANopen jsou implementovány pouze tři zprávy sběrnicového managementu. Připojené uzly lze spouštět jednotlivě pomocí rámce s určením jejich adresy nebo lze použít plošné spuštění všech uzlů. Zpráva typu „*Boot-Up*“ u nově aktivovaného modulu signalizuje ukončení fáze jeho inicializace, modul lze konfigurovat nebo spustit. Konfigurace je uložena ve flash paměti jednotlivých jednotek a po startu systému je okamžitě k dispozici [8].

Tab. 9 Zprávy sběrnicového managementu [8].

Název	Identifikátor	Bajty zprávy
Spustit vzdálený uzel	0x000	0x01, ID uzlu
Spustit všechny uzly	0x000	0x01, 0x00
Startovací zpráva	0x700 (1792D) + ID uzlu	0x00

5.2 Řízení polohy modulů

Jednotky řízení pohybu mohou být nakonfigurovány pro různé provozní režimy. Výchozím nastavením jednotky je režim polohovací. Pro řízení pomocí Faulhaber kanálu je nutné tuto konfiguraci změnit, což lze udělat pomocí odpovídajících příkazů. Aktuální nastavení jednotky je poté uloženo v paměti flash, odkud je také při jejím spuštění načítáno [8].

„*Faulhaber Motion Manager*“, volně přístupný program pro PC, umožňuje nastavení veškerých parametrů a provozních režimů prostřednictvím grafických dialogových oken. Příkazy lze zadávat ve formátu prostého textu nebo je vybírat z příslušného menu. Taktéž je možné kontrolovat stavové automaty sběrnice CANopen, aktuální stav se zobrazuje ve stavovém řádku [8].

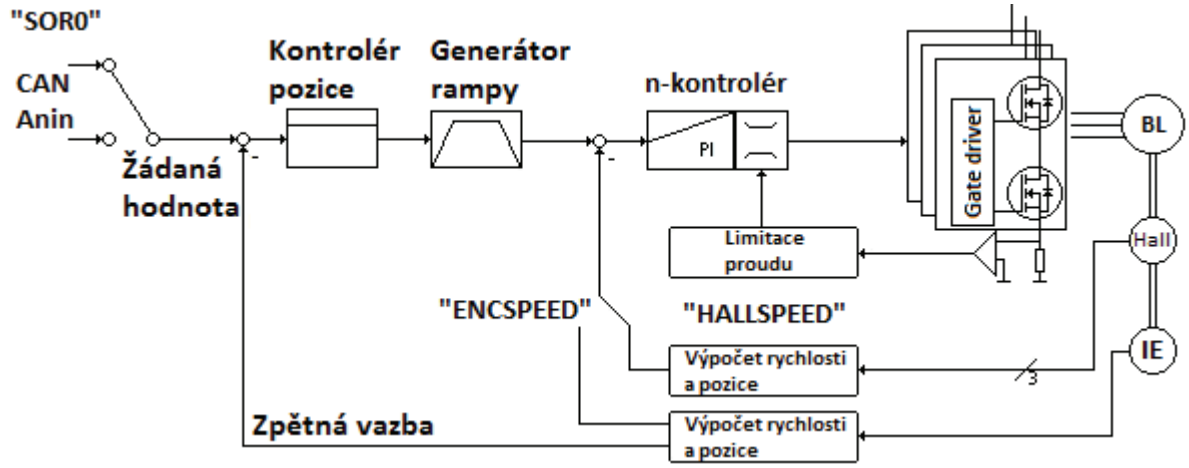
Faulhaber příkazy je možné přijímat pouze v aktivním stavu jednotky. Každý komunikační uzel je nutno nejdříve inicializovat a poté spustit. Níže je uveden přehled jednotlivých provozních režimů jednotek a příslušných příkazů k jejich nastavení [8].

Tab. 10 Příkazy pro změnu provozních režimů [8].

Příkaz	Argument	Funkce
SOR	0 – 4	Způsob řízení rychlosti
CONTMOD	-	Spojitý režim
STEPMOD	-	Režim krokového motoru
APCMOD	-	Režim analogového polohování
ENCMOD	-	Režim s externím enkodérem
GEARMOD	-	Režim pohonu
VOLTMOD	-	Režim napětí
IXRMOD	-	Režim IxR

5.3 Režim provozu jednotky s externím enkodérem

Pro vysoce přesné aplikace jsou hodnoty skutečné polohy pohonu měřeny pomocí externího enkodéru (režim „*ENCMOD*“). Snímač lze namontovat přímo na hřídel motoru. Je možné použít složitější profily řízení, které obsahují nastavení cílových hodnot (maximální rychlost, zrychlení, koncová poloha) pro polohování. Dále je možnost překročit rozsah enkodéru (±1,8.10^9) a plynule pokračovat v čítání bez ztráty dat. Pro zahájení pohybu motoru je nutné zaslat povel ke startu „*M*“. Dosažení cílové polohy je signalizováno zprávou „*OK*“ ve statusu jednotky [8].



Obr. 15 Struktura jednotky s externím enkodérem [8].

Mezi základní nastavení jednotky lze zahrnout příkazy udávající provozní režim „*ENCMOD*“ a „*SOR0*“. Dále lze pomocí níže uvedených příkazů nastavit rozsah polohovacích limitů (příkaz „*LL*“) a tyto limity aktivovat pomocí „*APL*“, dále lze nastavit proporcionální zesílení „*PP*“ a časový limit „*PD*“ [8].

Tab. 11 Základní příkazy režimu ENCMOD [8].

Příkaz	Argument	Popis
PP	1 - 255	Načte do jednotky zesílení
PD	1 - 255	Načte do jednotky časový limit
LL	±1,8.10^9	Kladné hodnoty určují horní limit, záporné spodní
APL	0 - 1	Platí pro všechny provozní režimy kromě VOLTMOD

Po započetí komunikace s jednotkou, určení jejího režimu práce a případném nastavení dodatečných informací, je nutné navázat komunikaci také s externím snímačem a provést jeho konfiguraci. Příkazy pro práci s externím enkodérem jsou následující [8].

Tab. 12 Příkazy pro práci s enkodérem [8].

Příkaz	**Argument**	**popis**
ENCMOD	-	Změna režimu snímače jako vysílače polohy
ENCSPEED	-	Měření rychlosti pomocí enkodéru
HALLSPEED	-	Měření rychlosti pomocí hallova senzoru
ENCRES	8 - 65 535	Načtení rozlišení externího snímače (4 krát puls/otáčka)

Po nastavení snímače lze zasílat další příkazy, které provádí samotné polohování. Do této kategorie patří příkazy jako uvedení jednotky do provozu nebo její odpojení, zadání požadované pozice motoru nebo nulování pozice enkodéru. Z vyšších funkcí může jednotka využívat generátoru rampy. Maximální dosažitelnou rychlost lze definovat pomocí příkazů „*AC*“, „*DEC*“ a „*SP*“. Dále lze nastavit omezení proudu pomocí hodnot „*LPC*“ a „*LCC*“, čehož lze využít k ochraně motoru před proudovým přetížením [8].

Tab. 13 Příkazy pro práci s enkodérem [8].

Příkaz	**Argument**	**Funkce**
EN	-	Aktivuje jednotku
DI	-	Deaktivuje jednotku
LA	Hodnota	Načte absolutní pozici
LR	Hodnota	Načte relativní pozici
M	-	Začne provádět příkaz ke změně pozice
HO	-/Hodnota	Definuje základní pozici motoru
POS	-	Vrací hodnotu aktuální pozice

6 NÁVRH APLIKACE

Úkolem komunikátoru je poskytnout uživateli možnost připojení a ovládání řídicích jednotek elektrických pohonů. Jednotky mohou ke komunikaci využívat sériového rozhraní nebo pokročilého rozhraní průmyslové sběrnice CAN. Vývojový kit disponuje 3,5“ TFT displejem s rezistivní technologií dotykové vrstvy, díky kterému je schopen nabídnout uživateli příjemné grafické rozhraní a moderní ovládání aplikace pomocí dotykové vrstvy a hardwarových tlačítek.

Těžištěm aplikace je práce komunikátoru v reálném čase. Kladen je velký důraz na jednoduchost funkcí grafického rozhraní, minimalizaci vytížení procesoru režijními a komunikačními výpočty. Komunikátor je současně schopen obsluhovat až čtyři řídicí jednotky elektrických pohonů, což při plném zatížení komunikačních rozhraní klade vysoké nároky na výkon procesoru.

Z výše zmíněných důvodů je nutné, aby po inicializaci aplikace byly vyvolávány jednotlivé části programu pouze na vyžádání. Nežádoucí je využívání aktivního čekání procesoru či nekonečných smyček. Ke splnění tohoto požadavku je výhodné využití možností externího přerušení (vstupy hardwarových tlačítek, dotykové vrstvy, komunikačních rozhraní) i interního přerušení procesoru (rutinní, interním časovačem řízená výměna dat s připojenými jednotkami).

6.1 Princip funkce aplikace

Řídicí aplikace průmyslového komunikátoru se logicky i graficky skládá z jednotlivých ploch. Plochy (formuláře) se svým jednoduchým a funkčním designem podobají formulářům platformy .NET pro stolní počítače. Každá plocha obsahuje libovolné množství řídicích prvků, pomocí kterých je možné aplikaci zasílat požadavky uživatele nebo naopak zobrazovat aktuální stav aplikace. Každý generovaný formulář musí pro správnou funkci obsahovat jméno, seznam vložených řídicích prvků a aktivní prvek. Aktivní prvek formuláře je graficky označen, aktivace prvku slouží pro následné ovládání prvku pomocí tlačítek. Z hlediska funkcionality formuláře by měl tento implementovat základní metody chodu plochy (vlastní zavedení a spuštění), dále implementovat metody pro zpracování vstupů od uživatele (dotyková vrstva a tlačítka), zvládat veškerou práci se svými prvky (vkládání, odstraňování, vyhledávání následujících a předchozích prvků k aktivaci) a přidávat pomocné grafické a funkční prvky okna.

Každý funkční prvek formuláře obsahuje jméno, souřadnici umístění na ploše, velikost a informaci, zdali může být označen a aktivizován. Všechny prvky zaujímají základní tvar čtverce, bod umístění na ploše určuje horní levý roh. Zadání velikosti prvku nachází uplatnění při zpracování vstupu od dotykové vrstvy.

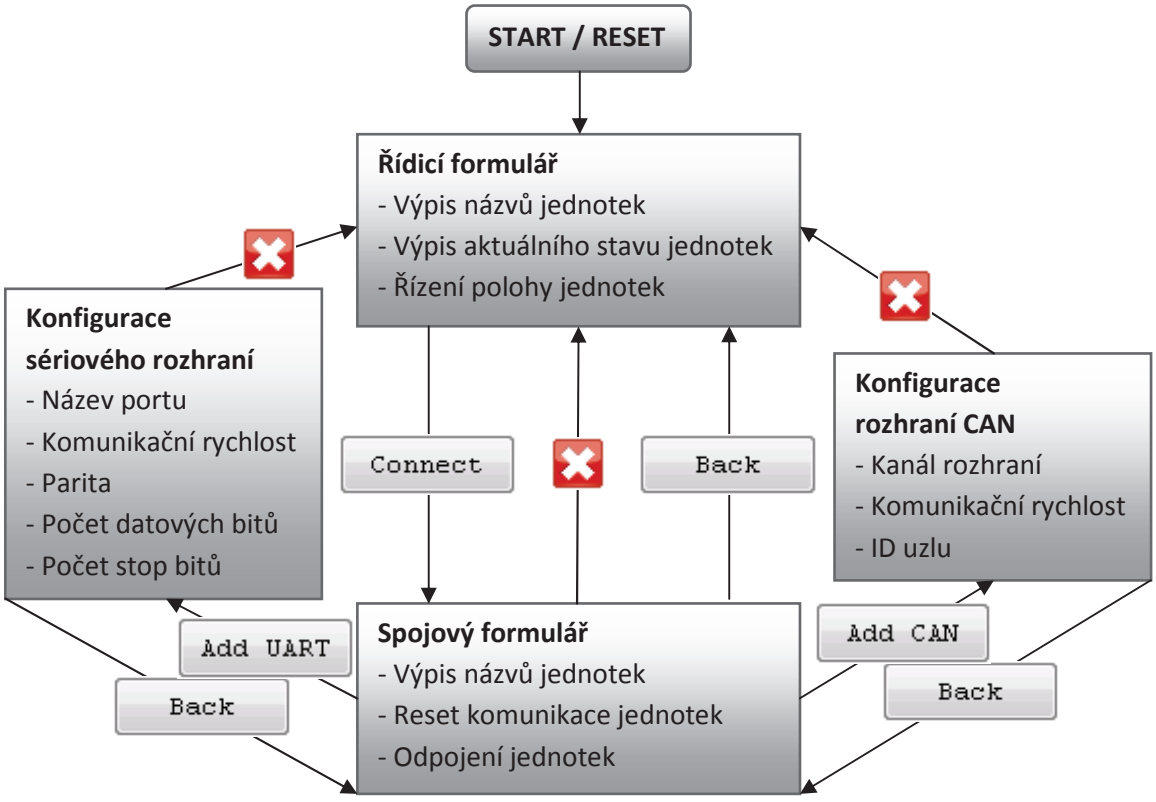
Z důvodu úspory paměti systému je řídicí aplikace dynamická, stará se o chod celého komunikátoru a spravuje veškeré dostupné zdroje. Mezi zdroje hardwarové se řadí funkcionalita displeje, SD karty, dotykové vrstvy, tlačítek a komunikačních rozhraní. Do softwarových zdrojů lze zařadit držení instance aktuálně používaného formuláře a správu funkcí připojených řídicích jednotek elektrických pohonů. Pro zvýšení komfortu práce s komunikátorem lze konfiguraci kontrolérů ukládat na vloženou paměťovou kartu, odkud se kontroléry při restartu aplikace zpětně načítají, čímž šetří čas uživatele s opětovnou konfigurací jednotek.

6.2 Způsob chodu aplikace

Po spuštění aplikace se uživateli zobrazí hlavní plocha, která obsahuje dynamický výpis veškerých připojených řídicích jednotek. Jednotky jsou periodicky dotazovány na aktuální natočení řízeného pohonu, jehož hodnota je ve formuláři v případě nutnosti aktualizována. Každé zobrazené jednotce lze zadat cílovou polohu natočení, která je přes odpovídající komunikační rozhraní jednotce odeslána.

Všechny formuláře (mimo hlavní) obsahují odkaz na svého předka (ve formě tlačítka „*Back*“). Pro urychlení přesunu uživatele v aplikaci je přidáván i odkaz na formulář hlavní (ve formě tlačítka zavření okna „*Cross*“). Aplikace se skládá z následujících formulářů.

- Řídicí formulář, který se stará o komunikaci s aktivními jednotkami, výpis jejich názvu, aktuálního natočení a vstupy uživatele pro zadání žádaného natočení. Obsahuje také odkaz na formulář spojový.
- Spojový formulář zobrazuje seznam aktivních jednotek s možností restartu komunikace a jejich úplného odpojení. Obsahuje odkazy na konfigurační formuláře pro rozhraní sériové linky a rozhraní CAN.
- Formulář konfigurace sériového rozhraní se stará o uživatelskou konfiguraci připojení nové jednotky pomocí sériové linky. Je nutné nastavit parametry jako název portu, komunikační rychlost, paritu, počet datových a stop bitů.
- Formulář CAN rozhraní zařizuje uživatelskou konfiguraci připojení nové jednotky pomocí sběrnice CAN. Lze nastavit parametry jako komunikační kanál, komunikační rychlost a ID uzlu.



Obr. 16 Vývojový diagram chodu aplikace.

7 REALIZACE APLIKACE

Aplikace je programově realizována pomocí několika tříd, které se dají rozdělit dle své funkce do následujících skupin: správa grafického displeje, obsluha vstupů od uživatele (dotyková vrstva a tlačítka), funkcionalita formulářů, funkcionalita prvků formulářů, pomocné datové struktury, komunikační rozhraní a jejich konfigurace, vlastní kontroléry a třída pro práci s ukládáním a načítáním konfiguračních dat pomocí SD karty.

Aplikace využívá reference na externí knihovny, které komunikátoru poskytují vyšší funkce pro práci s interními daty nebo nízko úrovňovou funkcionalitu hardwarových prvků komunikátoru. Do této kategorie se řadí použití periferií komunikátoru jako sériového portu, řadiče sběrnice CAN, rezistivní dotykové vrstvy, displeje, tlačítek atd.

Software komunikátoru dále pracuje s možností importu externích zdrojů, do aplikace je možné vkládat datové soubory jako holý text, textové fonty, obrázky, ikony či zvukové stopy. Této možnosti je využito především pro vložení několika typů externě vygenerovaných fontů a obrázků pro grafické rozhraní komunikátoru.

7.1 Ovládání aplikace

Uživatel má možnost s komunikátorem pracovat dvěma způsoby. Moderním a příjemným způsobem ovládání komunikátoru je použití integrované rezistivní dotykové vrstvy displeje. Druhým (spíše standardním) způsobem je použití hardwarových tlačítek. Většina aplikace je nezávislá na použitém způsobu vstupu od uživatele, některé funkce jsou přístupné pouze za použití dotykové vrstvy.

Vstupy uživatele pomocí hardwarových tlačítek mají vždy stejnou šablonu využití. Funkce jsou namapovány následujícím způsobem. Směrová tlačítka „*Up*“ a „*Down*“ mění aktuálnímu formuláři aktivní prvek. Aktivní prvek je graficky označen orámováním. Pořadí změny prvků udává pořadí jejich vložení do formuláře. Směrová tlačítka „*Left*“ a „*Right*“ jsou určena výhradně pro změnu aktuální hodnoty aktivního prvku formuláře. Středové tlačítko „*Select*“ slouží k provedení akce aktivního prvku.

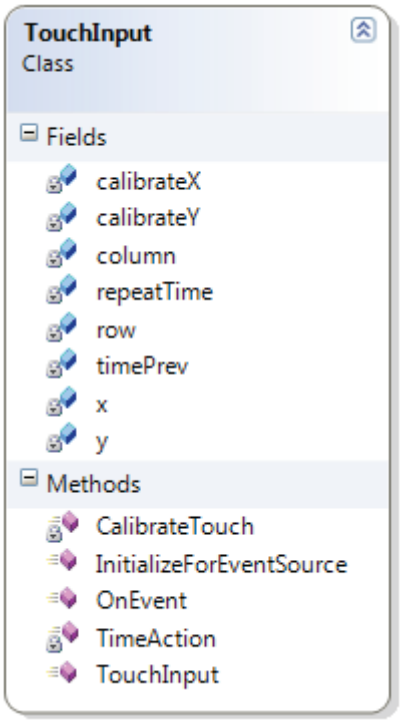
Vstupy uživatele od dotykové vrstvy jsou možné pouze ve formě jednoduchého dotyku, není možné použití gesta tahu, vícenásobného dotyku (například dvojklik) či dotyku několikanásobného.

7.1.1 Rezistivní dotyková vrstva displeje

Čtení signálů rezistivní dotykové vrstvy probíhá pomocí čtyř vodičů, které odečítají pozici stisku v souřadnicích X (XL, XR) a Y (YU, YD). Po převodu analogových signálů pomocí ADC převodníků a jejich zpracování dotykovým řadičem je vyvoláno externí přerušení procesoru. Tuto funkcionalitu uživateli zprostředkovává třída *Microsoft.SPOT.Touch*.

Výchozí rozlišení dotykového displeje o úhlopříčce 3,5 palce je 320x240 pixelů. Obdobně, s použitím výše zmíněného principu, je dotyková vrstva schopna pracovat se stejným rozlišením, tedy 320x240 bodů s vysokou přesností. Jelikož je použita rezistivní technologie odečtu pozice stisku, je vhodné k přesné navigaci používat stylus.

Třída *TouchInput*, která se stará o dotykovou komunikaci aplikace, implementuje pět metod. Jelikož při stisku dotykové plochy tato neustále vyvolává přerušení procesoru, je nutné po každém jeho vyvolání nejdříve časově určit, není-li stisk v zakázaném časovém intervalu od posledního stisku. Tolerance je 200 ms, časový interval je počítán pomocí tiků procesoru. Jakékoliv rychleji se opakující přerušení od dotykové vrstvy zbytečně zatěžuje procesor režijními výpočty, jelikož každý vstup je nutné dodatečně kalibrovat. Je-li vstup od dotykové vrstvy časově validní, začíná kalibrační přepočet souřadnice. Jak je uvedeno níže, přepočet sestává z přičtení kalibračních konstant k oběma souřadnicím dotyku.



Obr. 17 Diagram třídy TouchInput.

Po přesném určení pozice vstupu je souřadnice odeslána aplikaci, která ji odešle aktuálnímu formuláři. Formulář vyhodnotí, jestli se na pozici stisku vyskytuje nějaký prvek. Pokud je podmínka splněna, vyvolá se akce označeného prvku.

7.1.2 Kalibrace dotykové vrstvy

Displej pracuje s pixely o souřadnicích 0 až 319 v horizontálním směru a 0 až 239 ve směru vertikálním. Jelikož souřadnice stisku dotykové vrstvy neodpovídá skutečnosti, je nutný dodatečný kalibrační přepočet. Z důvodu neúnosného zatížení procesoru sofistikovaným přepočtem každého jednotlivého vstupu (polární přepočet souřadnic, nelinearita velikosti chyby v obou směrech, souřadný systém dotykové vrstvy je oproti souřadnému systému displeje natočen) je dotyková plocha rozdělena na dvanáct čtverců o velikosti 80x80 pixelů. Každá z dvanácti částí displeje má v obou směrech jiné kalibrační konstanty.

Kalibrační konstanty pro jednotlivé sektory displeje lze zjistit experimentálně. Na plochu displeje je vykresleno dvanáct čtverců shodných rozměrů, v jejichž středu se nachází hlavní kalibrační bod. Poté je odečítána shoda reálné souřadnice jednotlivých kalibračních bodů s hodnotou nekalibrované souřadnice zaslané dotykovým řadičem.

Tab. 14 Reálné souřadnice kalibračních bodů displeje.

Sektor	1	2	3	4
1	A[40;40]	A[120;40]	A[200;40]	A[280;40]
2	A[40;120]	A[120; 120]	A[200; 120]	A[280; 120]
3	A[40;200]	A[120; 200]	A[200; 200]	A[280; 200]

Tab. 15 Výstup souřadnic z dotykového řadiče.

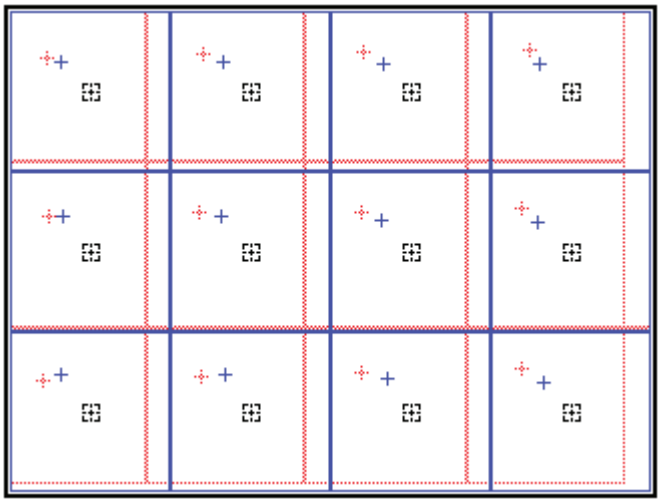
Sektor	1	2	3	4
1	B[31;38]	B[109;36]	B[190;34]	B[274;32]
2	B[32;120]	B[108;118]	B[188;115]	B[270;113]
3	B[31;202]	B[107;201]	B[185;197]	B[268;194]

Po drobné úpravě experimentálně zjištěných kalibračních parametrů jednotlivých částí plochy displeje vznikla tabulka pro finální kalibrační přepočet. Z tabulky je patná relativně podstatná odlišnost parametrů jednotlivých sektorů.

Tab. 16 Kalibrační přepočet dotykové vrstvy.

Sektor	1	2	3	4
1	C = B + [7;2]	C = B + [10;4]	C = B + [10;6]	C = B + [5;7]
2	C = B + [7;0]	C = B + [11;2]	C = B + [10; 4]	C = B + [8;7]
3	C = B + [9;-3]	C = B + [12;-1]	C = B + [13;3]	C = B + [11;7]

Pro zjednodušení procesorového výpočtu bylo nutné zanedbat chybu natočení souřadného systému dotykové vrstvy. Z níže uvedené ilustrace plyne vyšší rozlišení dotykové vrstvy než displeje a nelinearita chyby odečtu souřadnice. Souvislou modrou čarou je zobrazen výstup displeje, červenou nesouvislou čarou je zobrazen odpovídající výstup dotykové vrstvy bez kalibrace (s ukázkou velikosti chyby souřadnic a hlavních kalibračních bodů jednotlivých sektorů).

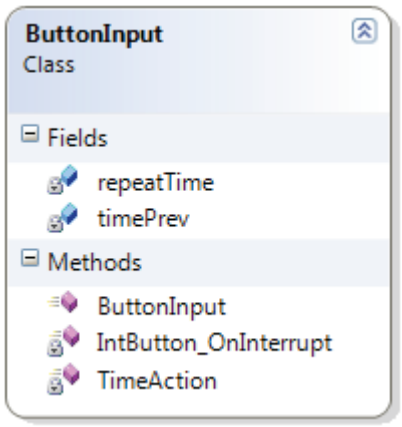


Obr. 18 Kalibrace dotykové vrstvy displeje.

7.1.3 Hardwarová tlačítka

Vývojový systém, tedy i komunikátor, disponuje šesti hardwarovými tlačítky, která pracují v režimu „*pull-up*“ a reagují na sestupnou hranu signálu. Jedná se o jedno resetovací tlačítko, čtyři směrová (IO0, IO1, IO4 a IO23) a střední (IO30). Tlačítka jsou připojena na digitální vstupy procesoru a vyvolávají jeho přerušení. Funkcionalitu obstarávají knihovny *Microsoft.SPOT.Hardware* a *GHI.Premium.Hardware*.

Třída *ButtonInput*, která se stará o komunikaci s aplikací pomocí tlačítek, implementuje tři metody. Opět je nutné po každém vyvolání přerušení určit, není-li stisk tlačítka v zakázaném časovém intervalu. Tolerance je nyní určena na časový úsek 150 ms. Jakýkoliv rychleji se opakující vstup působí negativně na funkci aplikace. Je-li stisk tlačítka časově validní, je odesláno číslo jeho portu ke zpracování aplikaci, která jej odešle aktuálnímu formuláři.



Obr. 19 Diagram třídy ButtonInput.

7.2 Grafické rozhraní aplikace

Uživatel pracuje s komunikátorem pomocí grafického rozhraní (GUI), které je zobrazováno na integrovaném displeji o úhlopříčce 3,5 palce (rozlišení 320x240 pixelů). Aplikace obsahuje několik předem nadefinovaných formulářů (tedy ploch, nelze pracovat s více okny zároveň), mezi kterými je možno plynule přecházet a využívat jejich funkcí. Každý formulář může obsahovat libovolné množství funkčních prvků, které ve svých metodách implementují veškerou funkcionalitu jednotlivých formulářů aplikace.

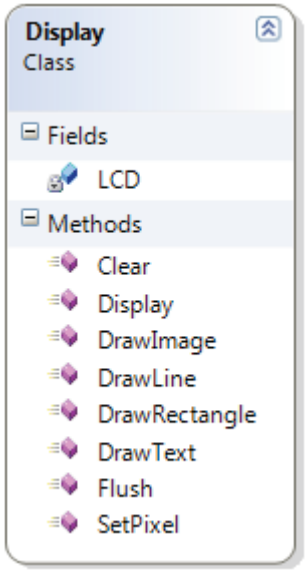
7.2.1 Integrovaný displej

Veškerý nízko úrovňový přístup systému k displeji zprostředkovává jmenný prostor *Microsoft.SPOT*. Komunikátor plně využívá možností integrované zobrazovací jednotky pomocí uživatelské třídy *Display*, která implementuje grafické funkce.

Systém pracuje s plochou displeje jako s maticí bodů, pro upřesnění se jedná o postupné budování bitmapové grafiky. Z hlediska funkčnosti zobrazení plochy existují dvě základní metody. Metoda *Clear* vyprázdní datový buffer pro tvoření aktuálního snímku, metoda *Flush* naopak snímek z bufferu převede na LCD displej.

Vytváření jednotlivých snímku je implementováno pomocí metod jako *SetPixel*, *DrawText*, *DrawImage*, *DrawLine* nebo *DrawRectangle*.

Třída *Bitmap* jmenného prostoru *Microsoft.SPOT* dále pokročilé různé pokročilé funkce pro práci s grafikou (na příklad průhlednost pixelů), jejich využití je však podstatně limitováno výkonem procesoru komunikátoru.



Obr. 20 Diagram třídy Display.

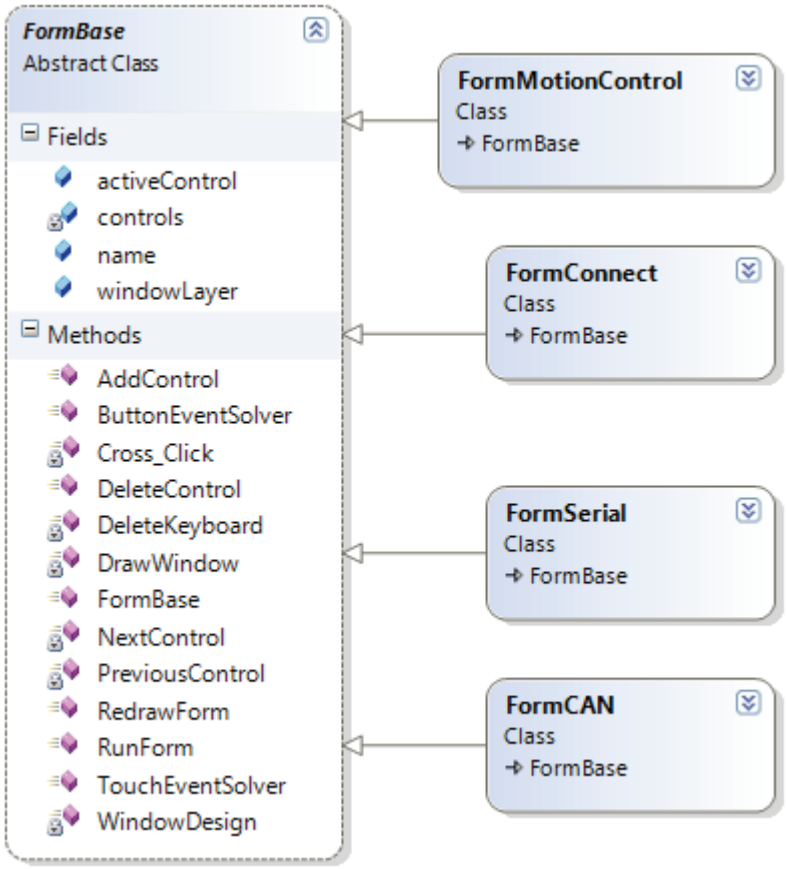
7.2.2 Formuláře aplikace

Veškerou funkcionalitu obsahuje abstraktní mateřská třída, ze které všechny použité formuláře dědí atributy a metody. Mezi hlavní metody patří zavedení, spuštění a překreslení formuláře, přidání či odebrání prvků formuláře nebo řešitelé vstupů od uživatele.

Práci se zobrazením a funkcionalitou grafické šablony okna formuláře zajišťují metody *DrawWindow* a *WindowDesign*, které se řídí hodnotou atributu *WindowLayer*. Dle dvoustavové hodnoty dochází buď k vykreslení prostého ohraničení plochy, nebo k vykreslení panelu okna s názvem formuláře a přidáním dodatečné funkcionality tlačítka „*Cross*“, které uživatele navede na základní formulář aplikace.

Vedlejší metody se starají především o práci s jednotlivými prvky formuláře. Důležitými atributy každého formuláře jsou jméno, seznam prvků a jeho aktivní prvek. Z důvodu úspory paměti systému se při zavedení a spuštění libovolného formuláře vždy volá jeho konstruktor, interní proměnné komunikátoru je tedy nutno ukládat jako statické na úrovni aplikace.

Níže na obrázku je uveden diagram abstraktní třídy *FormBase*, ze které je odvozena funkcionalita veškerých formulářů komunikátoru. Jak je z ilustrace patrné, aplikace pracuje celkem se čtyřmi formuláři. Hlavní funkční plochou komunikátoru je *FormMotionControl*. Pomocí formuláře *FormConnect* lze zobrazit a pracovat s připojenými kontroléry. Zbývající dva formuláře *FormSerial* a *FormCAN* se starají o navázání komunikace s novými řídicími jednotkami.



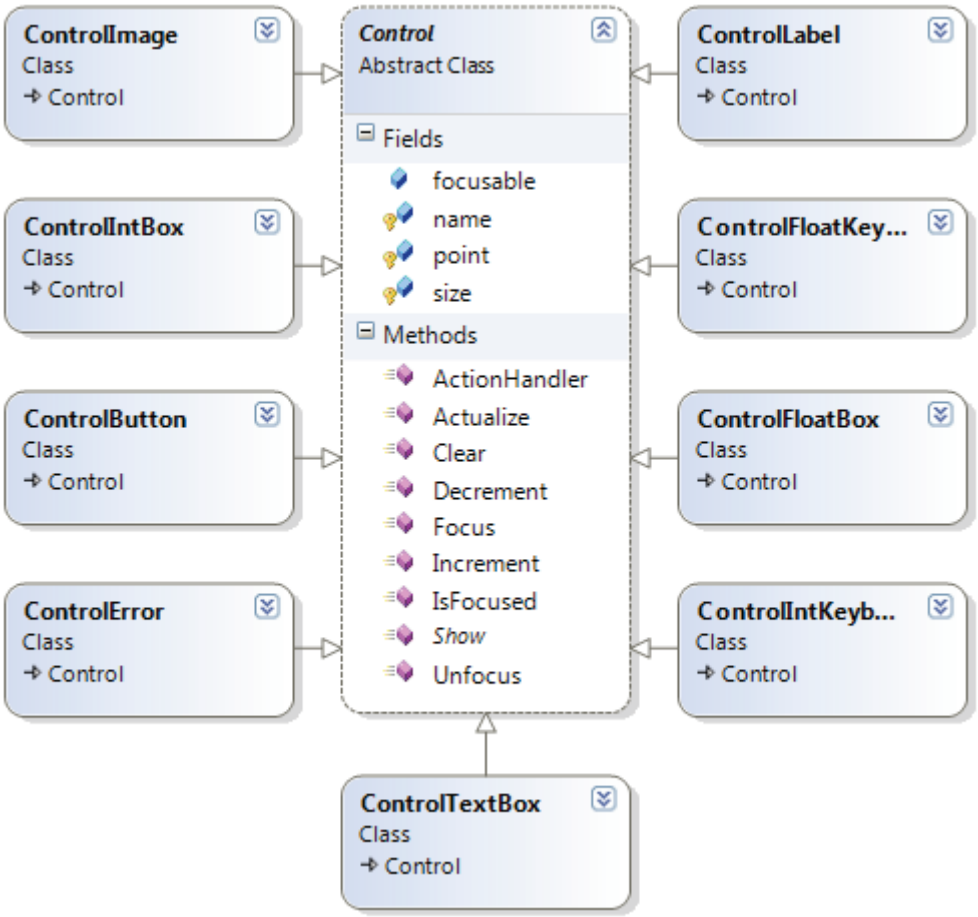
Obr. 21 Diagram třídy FormBase s odvozenými třídami formulářů.

7.2.3 Prvky formulářů

Každý formulář může obsahovat libovolné množství funkčních prvků, které jsou v každém formuláři předem nadefinované v konstruktoru. Společným předkem všech funkčních prvků je abstraktní třída *Controls*, která definuje sdílené atributy a metody. Každý prvek musí být pojmenován, musí znát své přesné umístění a velikost ve formuláři. Část metod je všem prvkům společná (smazání prvku, označení a ztráta označení, indikace zvolení prvku dotykem), část metod přetěžují jednotlivé prvky dle svých specifických potřeb (akce, aktualizace hodnoty, inkrementace či dekrementace hodnoty).

Jednotlivé prvky, které lze pro budování aplikace použít, jsou následující. Základ tvoří výpis textu *ControlLabel*, tlačítko *ControlButton*, výběr z předdefinovaných hodnot *ControlTextBox*, vstup celého čísla *ContolIntBox* pomocí klávesnice *ControlIntKeyboard* nebo vstup desetinného čísla *ContolFloatBox* pomocí klávesnice *ControlFloatKeyboard*. Z pokročilejších prvků lze zmínit *ControlError*, který informuje uživatele o neproveditelném požadavku.

Níže na obrázku je uvedena struktura, metody a atributy abstraktní třídy *Control*, ze které je odvozena funkcionalita veškerých devíti uživatelsky nadefinovaných typů funkčních prvků formulářů.



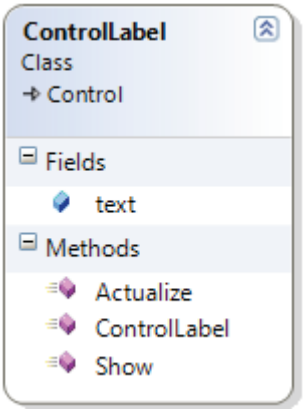
Obr. 22 Diagram třídy Control s odvozenými třídami funkčních prvků.

7.2.4 Funkce prvků formulářů

Jednotlivé prvky lze rozdělit do čtyř základních kategorií. Do první kategorie prvků informačních lze zařadit pouze výpis textu *ControlLabel*. Druhou kategorii tvoří prvky *ControlButton*, *ControlImage*, *ControlIntKeyboard* a *ControlFloatKeyboard* s funkcí tlačítka. Další kategorii tvoří prvky umožňující vstup celočíselné, neceločíselné či výběr textové hodnoty uživatelem. Sem lze zařadit zbývající prvky *ControlIntBox*, *ControlFloatBox* a *ControlTextBox*.

Funkci zcela odlišnou zastává *ControlError*, který pracuje s ošetřenými chybovými stavy a výjimkami v aplikaci. Níže následuje výčet prvků a jejich unikátních funkcí.

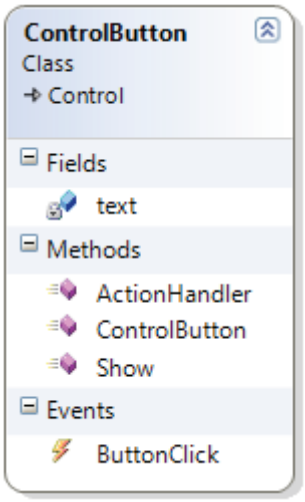
Prvek *ControlLabel* slouží k tvoření popisků nebo pro výpis libovolných, i časově proměnných informací. Popisek není možné označit či vyvolat jeho akci, je možné pouze aktualizovat jeho text. Prvek využívá pouze jediného atributu, který určuje právě text.



Obr. 23 Diagram třídy ControlLabel.

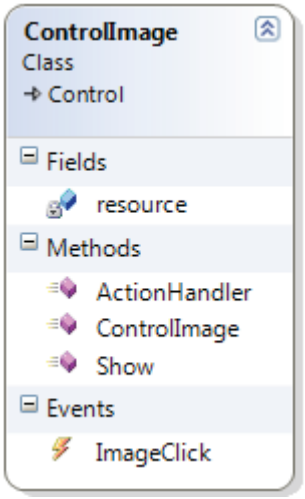
Prvek *ControlButton* se chová jako tlačítko. Je možné jej označit a poté provést akci (funkční kód akce je definován v metodě příslušného formuláře). Tlačítko také využívá jediného soukromého atributu, který určuje popisek.

Grafickým podkladem tlačítka je obrázek vždy shodné velikosti, který pochází z externích zdrojů aplikace. Design tlačítka je velmi podobný s designem tlačítek frameworku .NET pro stolní počítače.



Obr. 24 Diagram třídy ControlButton.

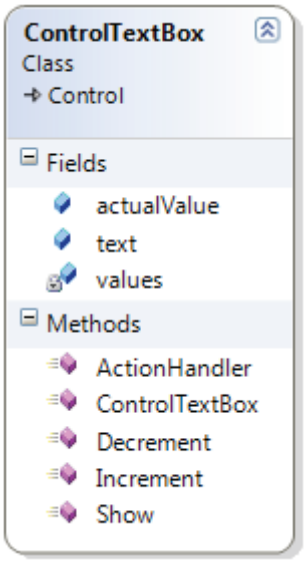
Prvek *ControlImage* umožňuje do formulářů vkládat obrázky (přípustné formáty obrázků v NETMF jsou *PNG*, *BMP*, *GIF*, *JPEG* a *TIFF*). Obrázek může být dle potřeb uživatele označitelný a lze jej případně využít jako tlačítko (vyvolání kódu akce). Obdobně jako u tlačítka je kód akce specifikován v příslusném formuláři, ve kterém se prvek nachází. Vložený grafický objekt má pouze jeden soukromý atribut, tedy odkaz na svůj externí zdroj.



Obr. 25 Diagram třídy ControlImage.

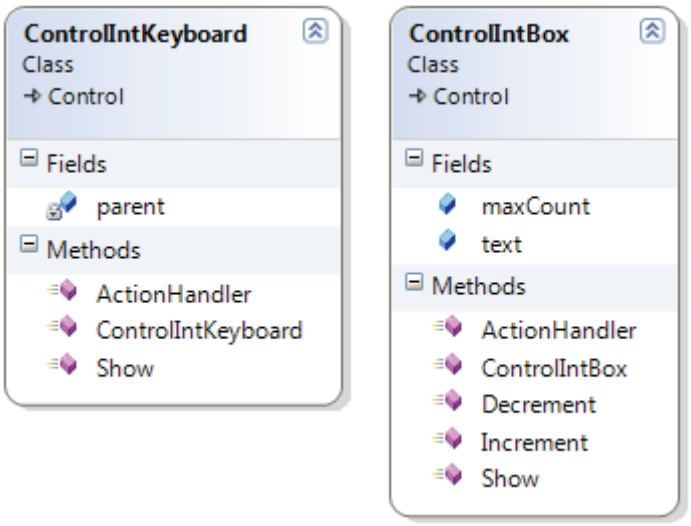
Prvek *ControlTextBox* slouží k výběru požadované hodnoty z hodnot uživatelem přednastavených. Prvek je možné označit, jeho dotykovou akcí je změna aktuální hodnoty na další možnost v pořadí (rotace hodnot jedním směrem). Pomocí hardwarových tlačítek lze aktuální hodnotu rotovat směry oběma.

ControlTextBox obsahuje atributy jako číslo aktuální hodnoty (výhodné pro programové zpracování zvolené možnosti), text aktuální hodnoty (pro výpis ve formuláři) a seznam přednastavených textových hodnot. Jako poslední parametr lze v konstruktoru prvku zadat požadovanou výchozí hodnotu prvku.



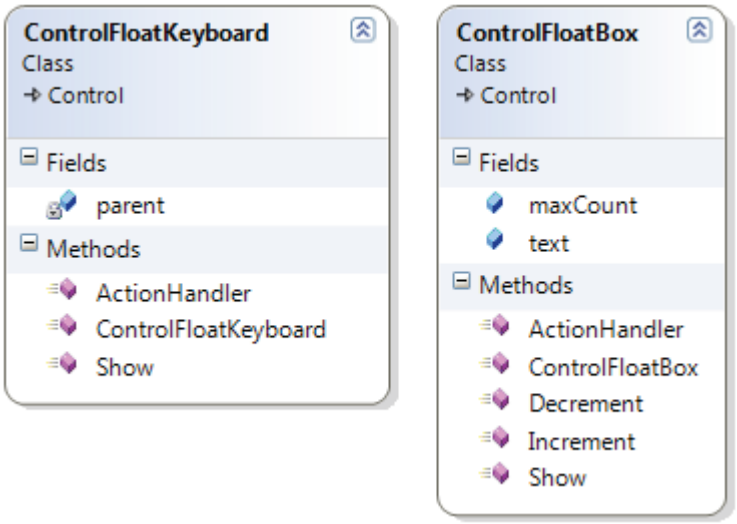
Obr. 26 Diagram třídy ControlTextBox.

Další prvek *ControlIntBox* umožňuje uživateli zadat do aplikace celočíselnou hodnotu. Lze jej označit, akcí je vynulování aktuální hodnoty a rozbalení softwarové klávesnice. Pomocí hardwarových tlačítek lze aktuální hodnotu snižovat či zvyšovat o jednotku. Prvek obsahuje dva soukromé atributy, přednastavenou číselnou hodnotu a maximální počet řádů čísla. Úzce souvisejícím prvkem je klávesnice *ControlIntKeyboard*, pomocí které lze hodnotu zadat.



Obr. 27 Diagramy tříd ControlIntKeyboard a ControlIntBox.

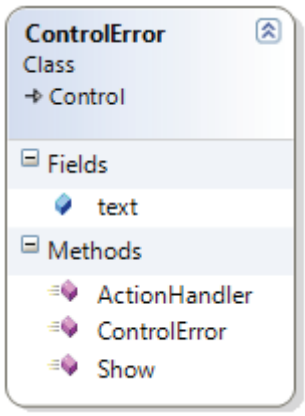
Obdobný prvek je *ControlFloatBox*, umožňuje zápis hodnoty s plovoucí desetinnou čárkou. Je možné jej označit, jeho akcí je vynulování aktuální hodnoty a rozbalení softwarové klávesnice *ControlFloatKeyboard*. Pomocí hardwarových tlačítek lze aktuální hodnotu snižovat či zvyšovat o jednotku. Prvek obsahuje dva soukromé atributy, tedy přednastavenou číselnou hodnotu a maximální počet znaků.



Obr. 28 Diagramy tříd ControlFloatKeyboard a ControlFloatBox.

Poslední implementovaný prvek je *ControlError*, který je vyvolán, nastane-li v systému ošetřený chybový stav. Prvek graficky mění formulář (vizuální ztmavení, přebarvení 75 procent pixelů aktuální plochy na černou barvu) a textově informuje o způsobu řešení chybového stavu.

Chybové hlášení není označitelné, jakákoliv akce ze strany uživatele vrátí systém o krok zpět. Prvek má jediný soukromý atribut, textovou informaci s popisem chyby.



Obr. 29 Diagram třídy ControlError.

7.3 Komunikační rozhraní komunikátoru

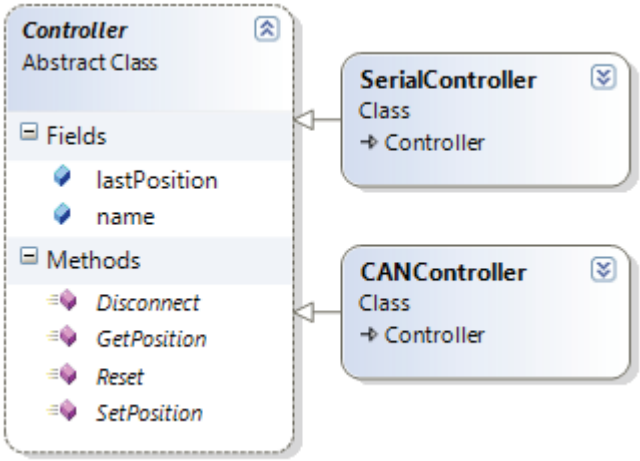
Komunikátor využívá ke komunikaci s řadiči elektrických pohonů dvou rozhraní. První možností je použití sériového rozhraní, druhou možností je připojení jednotky pomocí řadiče sběrnice CAN.

Sériové rozhraní umožňuje připojení až čtyř jednotek (porty s názvy *COM1*, *COM2*, *COM3* a *COM4*). Rozhraní sběrnice CAN umožňuje na svých dvou kanálech (s názvy *CH1* a *CH2*) teoreticky připojit až 127 + 127 zařízení s různou adresou. Aplikace je limitována současným použitím čtyř kontrolérů, které mohou využívat různých kombinací dostupných komunikačních rozhraní.

Základní předpis funkcionality pro komunikaci s jednotkami obsahuje abstraktní třída *Controller*. Každý kontrolér obsahuje dva atributy, jméno pro rozlišení a výpis názvu jednotky *name* a poslední hodnotu natočení *lastPosition*, která je načítána a zobrazena při přechodu mezi formuláři aplikace. Obě komunikační rozhraní sdílí metody pro práci s kontroléry. Připojení jednotek je realizováno pomocí statické metody *Connect*, kterou implementuje každé rozhraní odlišně.

K základní komunikaci s jednotkami slouží metoda *GetPosition*, která vrací hodnotu aktuálního natočení pohonu dotazované jednotky. Jejím opakem je metoda *SetPosition*, která jednotce zasílá žádanou hodnotu natočení.

Mezi další sdílené funkcionality patří metoda *Reset*, která slouží k resetování komunikace s jednotkou a metoda *Disconnect*, která jednotku trvale odpojí.

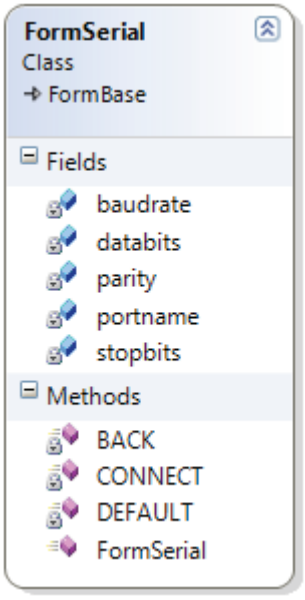


Obr. 30 Diagram třídy Controller s odvozenými třídami.

7.3.1 Sériové komunikační rozhraní

Práce kontroléru se sériovým rozhraním začíná vstupem požadovaných parametrů sériové komunikace pomocí kontaktního formuláře *FormSerial*. Jedná se o standardní parametry sériové komunikace, tedy o název sériového portu *portname*, komunikační rychlost *baudrate*, paritu *parity*, počet datových bitů *databits* a počet stop bitů *stopbits*. Zadání hodnot uživatelem probíhá rolováním přednastavených standardizovaných hodnot.

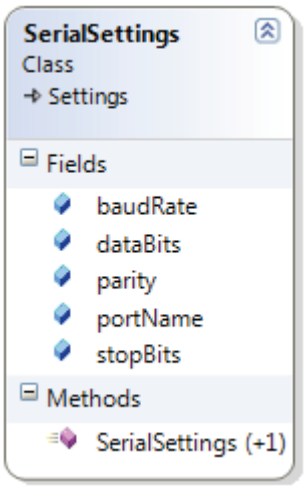
Z funkčních prvků dále formulář obsahuje tři tlačítka. První dvě „*Connect*“ a „*Back*“ zvolenou konfiguraci buď uloží, nebo ji zruší, aplikace uživatele vrátí zpět na spojový formulář *FormConnect*. Třetí tlačítko „*Default*“ zobrazí výchozí parametry komunikace.



Obr. 31 Diagram třídy FormSerial.

Zadáním komunikačních parametrů je vytvořena třída *SerialSettings*, která obsahuje výše zmíněné parametry sériové komunikace. Třída *SerialSettings* stejně jako třída *CANSettings* dědí z prázdné abstraktní třídy *Settings*.

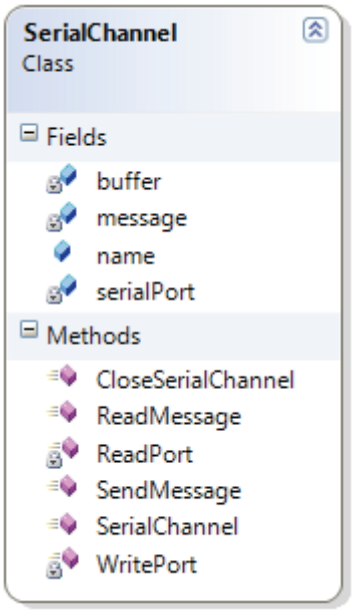
- *portName:* COM1, COM2, COM3 a COM4
- *baudRate:* 75 až 115200 (standard)
- *parity:* lichá, sudá a bez parity
- *dataBits:* 5, 6, 7, 8 a 9
- *stopBits:* 0, 1, 1.5 a 2



Obr. 32 Diagram třídy SerialSettings.

Pomocí parametrů sériové komunikace lze otevřít sériový komunikační kanál *SerialChannel*. Komunikační kanál obsahuje atributy jako své jméno *name*, otevřený sériový port *serialPort*, proměnnou *buffer* pro postupné vytváření zprávy a proměnnou *message* pro zprávu hotovou.

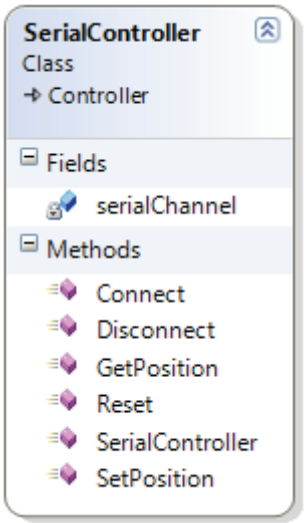
Metody komunikačního kanálu zprostředkovávají nejnižší úroveň samotné komunikace. Soukromé metody *ReadPort* a *WritePort* se starají o čtení a zápis datových řetězců na sériovou linku. Zbývající metody jsou již veřejně přístupné a zprostředkovávají funkcionalitu pro externí třídu kontroléru. Jedná se o zaslání datové zprávy *SendMessage*, přečtení přijaté zprávy *ReadMessage* a ukončení práce kanálu, tedy metodu *CloseSerialChannel*.



Obr. 33 Diagram třídy SerialChannel.

S otevřeným komunikačním kanálem již pracuje samotný sériový kontrolér. Kontrolér si ve formě atributu uchovává instanci komunikačního kanálu *serialChannel*. Kontrolér implementuje pro svou práci šest metod, jak je již výše zmíněno, své veřejné metody dědí z abstraktní třídy *Controller* a přidává vlastní statickou metodu *Connect*.

- *Connect:* zajišťuje připojení k jednotce
- *Disconnect:* zajišťuje odpojení jednotky
- *GetPosition:* dotazuje aktuální natočení
- *SetPosition*: zadává požadované natočení
- *Reset:* restartuje komunikaci s jednotkou

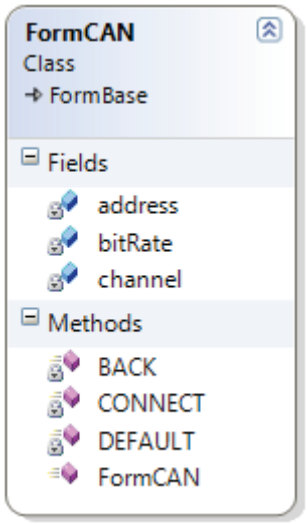


Obr. 34 Diagram třídy SerialController.

7.3.2 Komunikační rozhraní CAN

Uživatelská práce s kontrolérem využívajícím rozhraní CAN začíná vstupem požadovaných parametrů komunikace pomocí konfiguračního formuláře *FormCAN*. Jedná se o parametry *address* (ID uzlu na daném komunikačním kanálu), *bitRate* (určuje výchozí komunikační rychlost) a parametr *channel*, pomocí kterého lze vybrat mezi dvěma nezávislými CAN kanály.

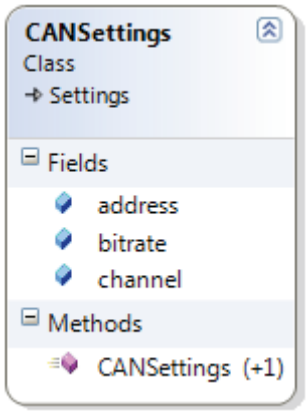
Z funkčních prvků formulář obsahuje popisky a tři tlačítka. První dvě „*Connect*“ a „*Back*“ zvolenou konfiguraci buď uloží, nebo ji zruší, aplikace uživatele vrátí zpět na spojový formulář *FormConnect*. Třetí tlačítko „*Default*“ zobrazí výchozí parametry komunikace.



Obr. 35 Diagram třídy FormCAN.

Zadáním komunikačních parametrů je vytvořena konfigurační třída *CANSettings*, která obsahuje výše zmíněné komunikační parametry. Třída *CANSettings* stejně jako třída *SerialSettings* dědí z prázdné abstraktní třídy *Settings*, obsahuje přetížení konstruktoru.

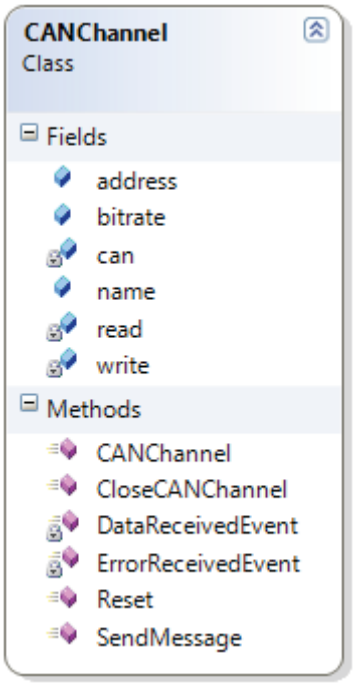
- *address:* přípustné jsou hodnoty 0 až 127
- *bitRate:* výběr z hodnot 125, 250, 500 a 1000 kb/s
- *channel:* integrována jsou dvě rozhraní CH1 a CH2



Obr. 36 Diagram třídy CANSettings.

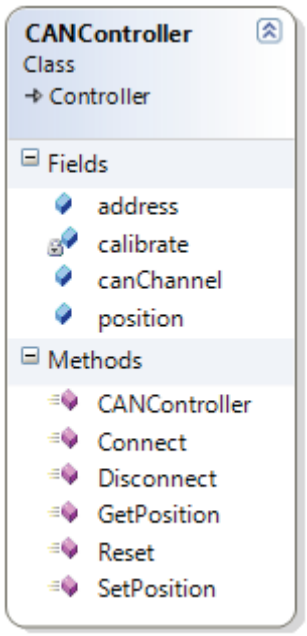
Pomocí parametrů rozhraní CAN lze otevřít komunikační kanál *CANChannel*, který pracuje s veřejnými atributy jako komunikační rychlost *bitrate*, instance otevřeného kanálu *can* a jménem *name*. Pomocná proměnná *address* slouží k inicializaci kontrolérů. Na jeden komunikační kanál lze připojit až 127 různých jednotek. Kanál využívá proměnných *read* a *write*, které aplikaci zajišťují bezztrátový příjem a zápis zpráv.

Metody kanálu *CANChannel* zprostředkovávají nízkou úroveň komunikace. Soukromé metody *DataReceivedEvent* a *ErrorReceivedEvent* se starají o příjem zpráv a chybových hlášení. Ostatní metody jsou již veřejně přístupné a zprostředkovávají funkcionalitu kanálu pro externí třídu CAN kontrolérů. Jedná se o metody pro zaslání zprávy *SendMessage*, restartování CAN řadiče komunikátoru *Reset* a ukončení činnosti kanálu *CloseCANChannel*.



Obr. 37 Diagram třídy CANChannel.

S otevřeným komunikačním kanálem již pracuje vlastní kontrolér rozhraní CAN, který si ve formě atributu uchovává instanci otevřeného komunikačního kanálu *can-Channel*, unikátní adresu řídicí jednotky *address* a proměnnou pro příjem zpráv *message*. Hodnota calibrate slouží ke korekci výstupu enkodéru. Kontrolér implementuje šest metod. Jak je již výše zmíněno, své veřejné metody dědí z abstraktní třídy *Controller* a přidává vlastní statickou metodu *Connect*. Jeden komunikační kanál může sdílet více instancí *CANController*.



- *Connect:* zajišťuje připojení k jednotce
- *Disconnect:* zajišťuje odpojení jednotky
- *GetPosition:* dotazuje aktuální natočení
- *SetPosition*: zadává požadované natočení
- *Reset:* restartuje komunikaci s jednotkou

Obr. 38 Diagram třídy CANController.

7.4 Ukládání konfigurace kontrolérů

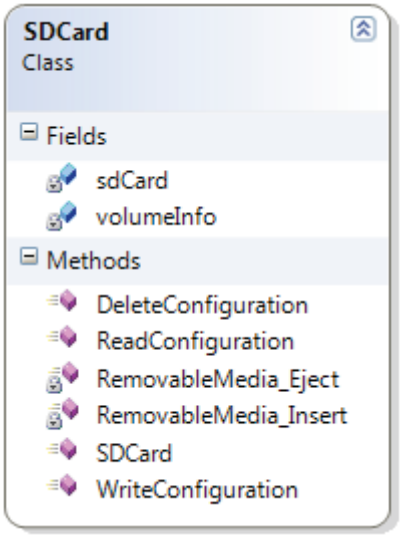
Pro zvýšení uživatelského komfortu komunikátoru lze navázaná spojení s řídícími jednotkami pohonů průběžně ukládat na vloženou SD kartu. Funkcionalitu zprostředkovává třída *SDCard*. Soukromými atributy jsou instance trvalého úložiště *sdCard* a instance informací o datového svazku *volumeInfo*.

Třída implementuje přerušení na základě vložení média *Removable-Media_Insert* a jeho odpojení *RemovableMedia_Eject*. Třída dále pracuje pomocí tří metod, které implementují zápis, čtení a mazaní konfiguračních souborů.

Metoda *WriteConfiguration* je volána při úspěšném navázání komunikace s řídicí jednotkou, ukládá konfigurační instanci příslušného potomka třídy *Settings* do samostatného souboru. Soubor nese unikátní název dle jména kontroléru a je zakončen příponou „**.set*“.

Metoda *ReadConfiguration* je spuštěna pouze při startu nebo restartu aplikace. Je-li vložena SD karta, je přečten její obsah a jsou vyhledány veškeré konfigurační soubory. Pomocí konfiguračních souborů jsou následně zrekonstruovány konfigurační třídy jednotlivých kontrolérů. Poté se komunikátor snaží s řídicími jednotkami navázat komunikaci.

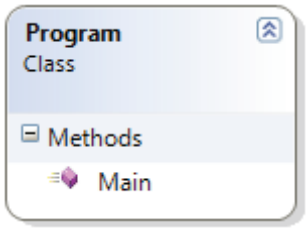
Poslední implementovanou metodou je *Delete-Configuration*, které je volána tehdy, pokud uživatel ve formuláři *FormConnect* stiskne tlačítko „*Disconnect*“ s cílem jednotku korektně odpojit, dále pokud s jednotkou při restartu aplikace nelze navázat spojení. V obou případech je na SD kartě vyhledán a smazán odpovídající konfigurační soubor.



Obr 39. Diagram třídy SDCard.

7.5 Spuštění aplikace

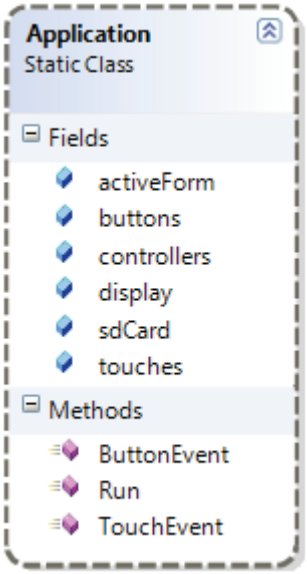
Vstupním bodem celého programu je standardní třída *Program* a její jediná metoda *Main*, jejímž posláním je zavolání statické metody *Run* ze statické třídy *Application*. Tím dochází ke spuštění aplikace.



Obr. 40 Diagram třídy Program.

Statická třída *Application* reprezentuje vlastní aplikaci a její funkcionalitu. Pomocí šesti statických atributů uchovává veškeré nutné informace pro svůj bezproblémový chod. Atribut *activeForm* obsahuje instanci aktivního formuláře, proměnná *buttons* obsahuje instanci pro práci se vstupy od hardwarových tlačítek, seznam *controllers* obsahuje veškeré aktivní kontroléry, proměnná *display* spravuje instanci třídy *Display* pro funkcionalitu displeje, atribut *sdCard* spravuje externí úložné zařízení a atribut *touches* obsahuje instanci třídy *TouchInput*, která se stará o zpracování dotykových signálů.

Metoda *Run* inicializuje veškeré atributy, zavádí a spouští hlavní formulář. Statické metody *ButtonEvent* a *TouchEvent* se starají o zpracování vstupů od uživatele do aktivního formuláře.

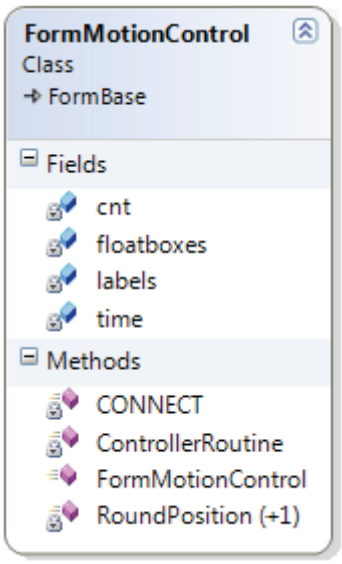


Obr. 41 Diagram třídy Application.

Po inicializaci všech potřebných zdrojů aplikace komunikátoru a případného načtení dříve uložených konfigurací kontrolérů z vložené SD karty se bod běhu aplikace přesouvá do grafického rozhraní. Je zaveden a spuštěn první formulář, formulář hlavní s názvem *FormMotionControl*.

Mezi soukromé atributy formuláře se řadí pomocná proměnná kontrolérů *cnt* a seznam zobrazených funkčních prvků pro zadání požadované hodnoty natočení aktivních kontrolérů *floatboxes*. Počtu aktivních kontrolérů také odpovídá seznam neustále aktualizovaných popisků *labels*, které slouží k informování uživatele o aktuálních hodnotách natočení připojených jednotek.

Atribut *time* slouží pro nastavený časového intervalu kontrolní rutiny komunikátorů *ControllerRoutine*, která se periodicky dotazuje na stav připojených řídicích jednotek, popřípadě přeposílá požadavky uživatele. Pomocné metody *RoundPosition* stylisticky zpracovávají číselné hodnoty natočení k výpisu ve formuláři, který obsahuje pouze jediné tlačítko s názvem „*Connect*“, odkaz na formulář spojový.

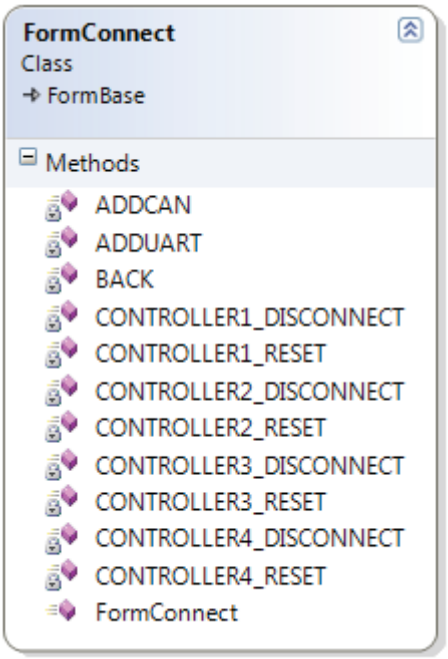


Obr. 42 Diagram třídy FormMotionControl.

Spojový formulář *FormConnect* obsahuje funkcionalitu pro základní práci s již připojenými řídicími jednotkami.

Formulář zobrazuje dynamický výpis názvů až čtyř aktivních kontrolérů. Pomocí tlačítek „*Reset*“ a „*Disconnect*“ na odpovídajícím řádku výpisu lze danému kontroléru restartovat s jednotkou komunikaci nebo ji od komunikátoru zcela odpojit a smazat její konfigurační údaje.

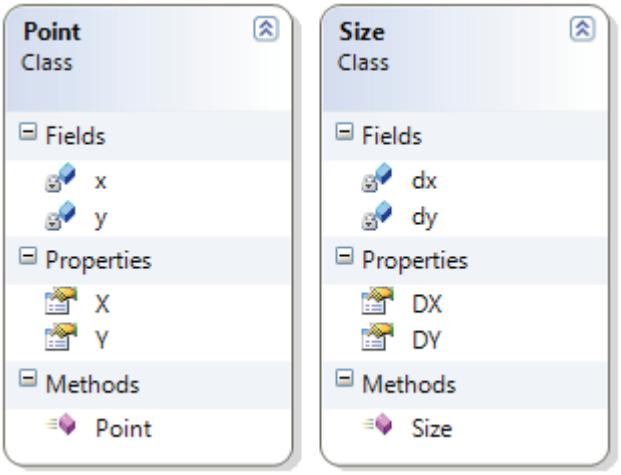
Formulář dále obsahuje tři tlačítka. První dvě „*Add UART*“ a „*Add CAN*“ odkazují na výše zmíněné konfigurační formuláře pro sériové a CAN komunikační rozhraní. Tlačítko „*Back*“ odkazuje na přímého předka spojového formuláře, na formulář hlavní.



Obr. 43 Diagram třídy FormConnect.

7.6 Pomocné třídy

Pomocné třídy slouží aplikaci ke kompaktnímu uchování informací o souřadnicovém umístění funkčních prvků na ploše formulářů a rozměrech aktivní plochy prvků. Obě třídy obsahují odpovídající soukromé atributy a vlastnosti pro přístup. Třída *Point* uchovává dvě celočíselné souřadnice prvků (0 až 319 horizontálně, 0 až 239 vertikálně). Třída *Size* uchovává také dvě celočíselné hodnoty, tentokrát se jedná o délky stran obdélníku ohraničení prvku.



Obr. 44 Diagramy tříd Point a Size.

8 FUNKCE APLIKACE

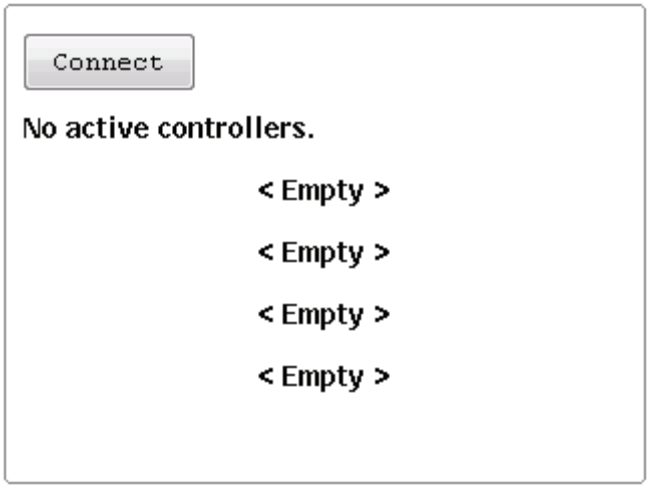
Aplikace průmyslového komunikátoru je určena ke správě řídicích jednotek elektrických pohonů, které lze současně spravovat až čtyři. Ke komunikaci s jednotkami lze využít dvou implementovaných komunikačních rozhraní, tedy sériové linky (integrovány jsou čtyři porty) nebo rozhraní sběrnice CAN (integrovány jsou dva nezávislé kanály). V rámci připojení čtyř řídicích jednotek ke komunikátoru je možné využít všech dostupných kombinací komunikačních rozhraní.

Těžiště aplikace je implementováno v řídicím formuláři, který obsahuje časově řízenou rutinní metodu *ControllerRoutine*. Metoda je volána časovačem každých 500 ms, dotazuje se na aktuální natočení a zasílá požadované hodnoty natočení všem připojeným jednotkám. V případě potřeby rutina aktualizuje hodnoty formuláře.

Grafické prostředí komunikátoru se skládá z jednotlivých formulářů, které obsahují funkční prvky. Při spuštění libovolného formuláře zpočátku není označen žádný prvek, lze jej označit dotykem nebo listováním prvky (hardwarová tlačítka „*Up*“ a „*Down*“). Označený prvek lze aktivovat (dotyk nebo tlačítko „*Center*“) a měnit jeho aktuální hodnotu pomocí hardwarových tlačítek „*Left*“ a „*Right*“. Softwarové klávesnice lze obsluhovat pouze pomocí dotykové vrstvy. Dotykem na prázdnou plochu označení aktivního prvku mizí.

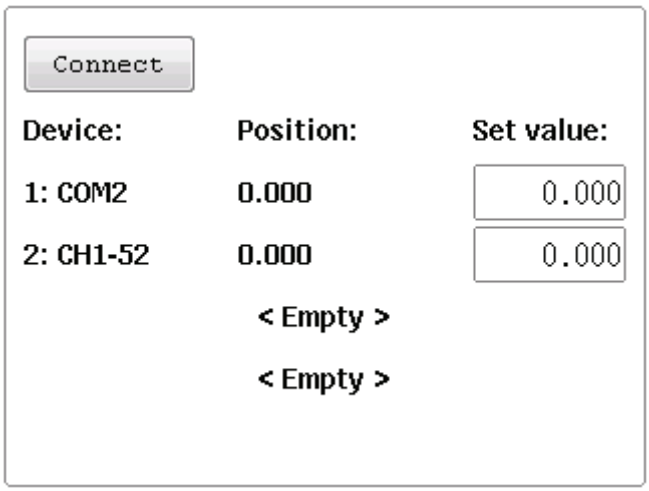
8.1 Řídicí formulář

Hlavní řídicí formulář slouží uživateli k výpisu aktuálního stavu připojených jednotek a k zadávání žádané hodnoty jejich natočení. Formulář neobsahuje grafický panel okna, je pouze jednoduše ohraničen obdélníkem. Tlačítko „*Connect*“ odkazuje na spojový formulář. Následuje výpis aktivních kontrolérů, v případě potřeby je ve spodní části displeje vykreslena klávesnice pro uživatelský vstup neceločíselné hodnoty požadovaného natočení dané jednotky. Po spuštění či restartu aplikace je načten prázdný hlavní formulář, ke komunikátoru nejsou připojeny žádné řídicí jednotky. Jsou zobrazeny pouze čtyři volné sloty. Jsou-li na SD kartě k dispozici soubory uložených konfigurací, jsou načteny a komunikátor se s řídicími jednotkami pokusí spojit. Povede-li se danou jednotku zkontaktovat, je přidána a v seznamu zobrazena. V opačném případě se konfigurace zahodí a soubor z SD karty smaže.



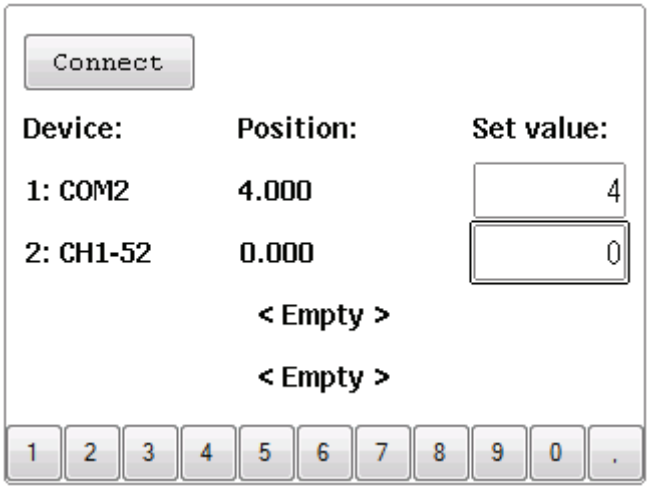
Obr. 45 Prázdný řídicí formulář.

V níže ilustrovaném případě jsou ke komunikátoru připojeny dva kontroléry. Je zobrazeno jejich jméno, aktuální natočení a požadované natočení. Jméno kontroléru určuje typ použitého komunikačního rozhraní („*COM*“ pro sériovou linku plus číslo portu, „*CH*“ pro sběrnici CAN plus číslo kanálu plus ID uzlu). Natočení je zobrazováno i zadáváno počtem otáček.



Obr. 46 Řídicí formulář se dvěma připojenými jednotkami.

V poslední uvedené situaci jsou opět připojeny dva kontroléry, je označen a aktivován editační prvek druhé jednotky. Zobrazena je softwarová klávesnice pro vstup požadované neceločíselné hodnoty. Hodnotu lze zadat pomocí klávesnice nebo inkrementovat či dekrementovat o jednotku pomocí hardwarových tlačítek.

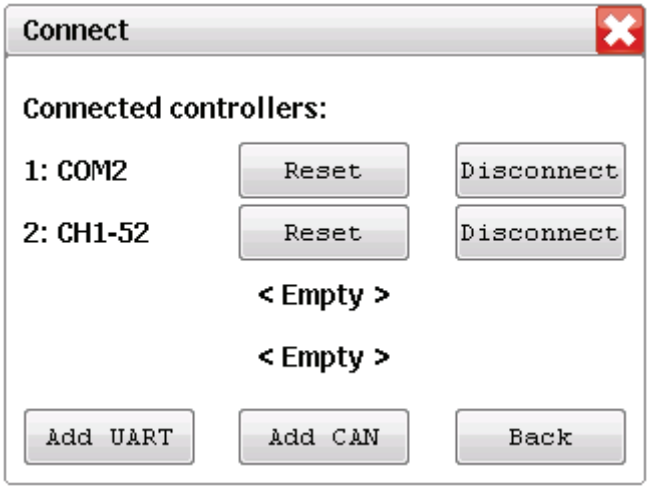


Obr. 47 Řídicí formulář se vstupem požadované hodnoty.

8.2 Spojový formulář

Po stisku tlačítka „*Connect*“ na hlavním formuláři je načten a spuštěn formulář spojový, který uživateli slouží k základní správě řízených jednotek. Formulář obsahuje grafický panel s funkcionalitou okna. Červené tlačítko „*Cross*“ uživatele vrátí zpět na formulář řídicí, obdobně jako tlačítko „*Back*“ v dolní části plochy. Zbývající tlačítka „*Add UART*“ a „*Add CAN*“ odkazují na konfigurační formuláře, pomocí kterých lze přidat do aplikace nové kontroléry. Je-li kapacita komunikátoru naplněna, při pokusu přidat další jednotku se zobrazí příslušné chybové hlášení.

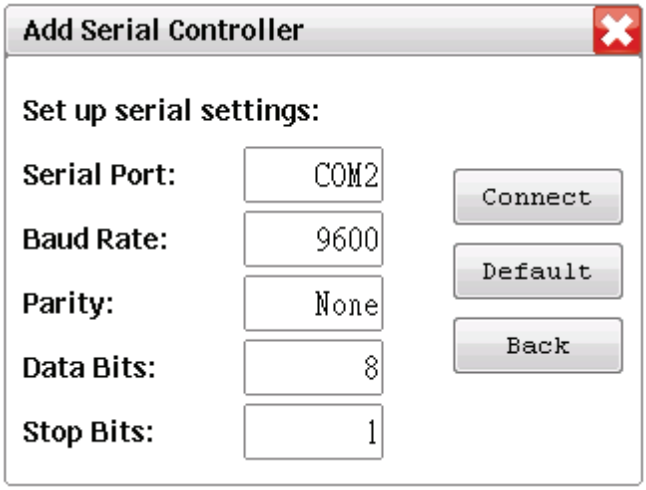
Na formuláři níže je zobrazen příklad se dvěma připojenými kontroléry. Vypsán je jmenný seznam aktivních jednotek spolu s příslušnými tlačítky, která jednotlivým kontrolérům zprostředkovávají funkci restartu nebo odpojení od komunikátoru. Při odpojení jednotky je také smazán příslušný konfigurační soubor z vložené SD karty.



Obr. 48 Spojový formulář se dvěma připojenými jednotkami.

8.3 Konfigurační formulář sériového rozhraní

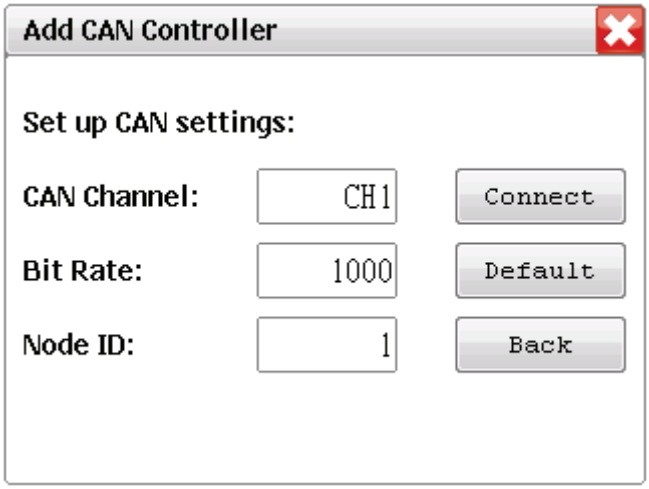
Po stisku tlačítka spojového formuláře „*Add UART*“ je uživatel odkázán na formulář sloužící ke konfiguraci nového sériového kontroléru. Formulář také obsahuje grafický panel s funkcionalitou okna, tlačítko „*Cross*“ uživatele přesune na formulář řídicí. Tlačítko „*Back*“ uživatele bez uložení zvolené konfigurace vrací na formulář spojový, tlačítko „*Default*“ obnovuje níže zobrazené výchozí hodnoty parametrů komunikace. Poslední tlačítko „*Connect*“ uloží provedenou konfiguraci, otevře komunikační kanál a pokusí se spojit s připojenou jednotkou pomocí příslušných příkazů. Odpoví-li jednotka korektně, zvolená konfigurace je uložena na SD kartu a kontrolér do aplikace. Nepodaří-li se komunikaci navázat, je zobrazeno odpovídající chybové hlášení. Odlišný chybový stav nastane, pokud se uživatel pokusí otevřít již otevřený komunikační port.



Obr. 49 Konfigurační formulář sériového rozhraní.

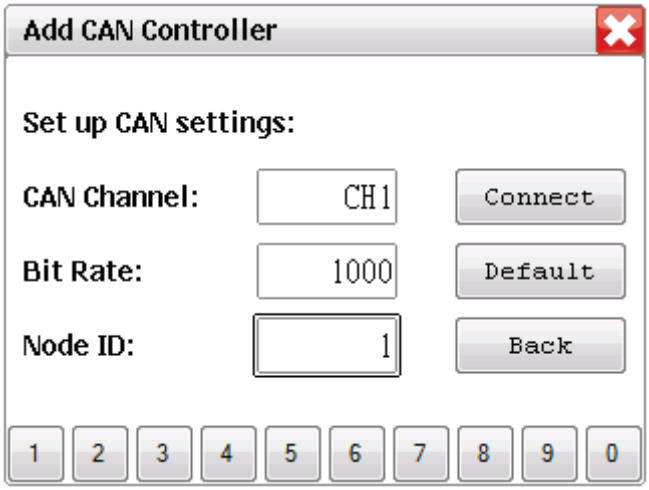
8.4 Konfigurační formulář CAN rozhraní

Po stisku tlačítka „*Add CAN*“ spojového formuláře je uživatel přesměrován na formulář sloužící ke konfiguraci nového kontroléru, který komunikuje přes rozhraní CAN. Formulář obsahuje grafický panel s funkcionalitou okna, tlačítko „*Cross*“ uživatele přesune na řídicí formulář. Tlačítka „*Connect*“, „*Back*“ a „*Default*“ mají obdobnou funkci jako u konfiguračního formuláře rozhraní sériového.



Obr. 50 Konfigurační formulář rozhraní CAN.

První dvě konfigurační hodnoty uživatel volí z hodnot přednastavených. K dispozici jsou dva nezávislé kanály komunikačního CAN rozhraní (CH1 a CH2). Komunikační rychlost lze zvolit z hodnot 125, 250, 500 a 1000 kb/s. Poslední hodnota udávající adresu uzlu na sběrnici je celočíselná, přípustný rozsah je 0 až 127 a její vstup je realizován pomocí softwarové klávesnice.



Obr. 51 Konfigurační formulář rozhraní CAN s editací hodnoty adresy uzlu.

Jelikož lze na jeden kanál připojit více zařízení, může při konfiguraci nastat několik chybových stavů. Pokusí-li se uživatel přidat novou jednotku na komunikační kanál již otevřený s odlišnou komunikační rychlostí, je zobrazeno chybové hlášení. Uživatel je také informován o pokusu přidání jednotky již využité adresy na otevřeném kanálu či adresy, která není validní.

9 Závěr

Diplomová práce řeší návrh a realizaci komunikátoru pro ovládání a synchronizaci řídicích jednotek elektrických pohonů prostřednictvím sériových komunikačních rozhraní UART (RS-232, RS-485) a CAN. Komunikátor je navržen a realizován na softwarové platformě .NET Micro Framework v jazyce C#. První část práce obsahuje seznámení s platformou .NET Micro Framework a vývojovým kitem EMX od společnosti GHI Electronics, na kterém je komunikátor realizován. V následující části práce je provedeno představení sběrnice CAN a protokolu CANopen. Ve zbytku práce je popsán návrh, koncept a následná realizace aplikace komunikátoru.

Aplikace průmyslového komunikátoru reaguje na podněty uživatele a komunikuje s připojenými jednotkami v reálném čase. Pro použitý vývojový kit, jsem vytvořil chybějící třídy nutné pro realizaci grafického uživatelského rozhraní, které umožňuje vytváření formulářů a práci s okny na dotykovém displeji. Dále byla vytvořena podpora pro kalibraci dotykové vrstvy použitého displeje.

Pro práci s řídicími jednotkami byla navržena a implementována základní třída Controller, která definuje potřebné rozhraní pro připojení a následnou práci s řídicími jednotkami. Od této třídy jsou poté odvozeny třídy SerialController a CANController, které implementují podporu pro jednotky s podporou UART a CAN. S použitím rozhraní třídy Controller je aplikace schopná realizovat nakonfigurované připojení jednotky přes zvolené rozhraní a ovládat jednotku v polohovém režimu.

Funkce komunikátoru byla prakticky ověřena pomocí připojení řídicích jednotek a následného řízení polohy elektrických pohonů obou zmíněných implementovaných komunikačních rozhraní.

Zrealizovanou aplikaci je možné dále rozšířit například o plánovač pohybů, který by umožnil řízení jednoduchých strojů s jedním nebo více pohony ovládanými podporovanými řídicími jednotkami.

SEZNAM POUŽITÉ LITERATURY

[1] EMX User Manual. GHI ELECTRONICS. *Catalog: EMX Module* [online]. Rev.1.3. 2013-09-24 [cit. 2013-04-26]. Dostupné z: http://www.ghielectronics.com/downloads/EMX/EMX_User_Manual.pdf

[2] .NET Micro Framework. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2013, 2013-03-07 [cit. 2013-04-26]. Dostupné z: http://en.wikipedia.org/wiki/.NET_Micro_Framework

[3] Netduino. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2013, 2013-04-18 [cit. 2013-05-11]. Dostupné z: http://en.wikipedia.org/wiki/Netduino

[4] Netduino Go Starter Kit. SK PANG ELECTRONICS LTD. *SK Pang Electronics: Netduino* [online]. Harlow. Essex, 2013 [cit. 2013-05-11]. Dostupné z: http://www.skpang.co.uk/catalog/netduino-go-starter-kit-p-1087.html

[5] .NET Gadgeteer: What is Gadgeteer?. MICROSOFT CORPORATION. *Microsoft .NET* [online]. 2011 [cit. 2013-05-11]. Dostupné z: http://www.netmf.com/gadgeteer/what-is-gadgeteer.aspx

[6] PFEIFFER, Olaf, Andrew AYRE a Christian KEYDEL. COPPERHILL TECHNOLOGIES CORPORATION. *Embedded Networking with CAN and CANopen*. United States of America: RTC Books, San Clemente, CA, 2008. Revised First Edition. ISBN 978-0-9765116-2-5. Dostupné z: http://www.copperhillmedia.com

[7] About CAN: Higher Layer Protocols. KVAZER. *Kvazer: Advanced CAN Solutions* [online]. 2013 [cit. 2013-04-26]. Dostupné z: http://www.kvaser.com/en/about-can/higher-layer-protocols.html

[8] FAULHABER. *CANopen with FAULHABER CAN: Communication / Function Manual*. 2nd edition. Schönaich · Germany, 2012. Dostupné z: http://www.micromo.com/manuals/pdf/schnittstelle/can_bl_dc/EN_7000_05030.pdf

SEZNAM POUŽITÝCH ZKRATEK

ACK	Acknowledgement
ADC	Analog to Digital Converter
ARM	Advanced RISC Machine
ASCII	American Standard Code for Information Interchange
BMP	Bitmap
CAN	Controller Area Network
CCP	CAN Calibration Protocol
CiA	CAN in Automation
CLK	Clock
CLR	Common Language Runtime
CMD	Command
CRC	Cyclic Redundancy Check
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
DAC	Digital to Analog Converter
DHCP	Dynamic Host Configuration Protocol
DLC	Data Length Code
DLL	Dynamic Link Library
ERR	Error
EWR	Extended Week References
FAT	File Allocation Table
FIFO	First In, First Out
GIF	Graphics Interchange Format
GND	Ground
GPIO	General Purpose Input/Output
GPS	Global Positioning Systém
GUI	Graphical User Interface
HAL	Hardware Access Layer
HB	High Byte
HHB	High High Byte
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ISO/OSI	International Standards Organization/Open Systems Interconnection
I/O	Input/Output
JPEG	Joint Photographic Experts Group
JTAG	Joint Test Action Group
LB	Low Byte
LCD	Liquid Crystal Display
LED	Light Emitting Diode

LLB	Low Low Byte
MAC	Media Access Control
NETMF	.NET Micro Framework
NMT	Network Management
OD	Object Dictionary
OEM	Original Equipment Manufacturer
PAP	Password Authentication Protocol
PC	Personal Computer
PDO	Process Data Object
PNG	Portable Network Graphics
PPP	Point to Point Protocol
PWM	Pulse Width Modulation
RAM	Random Access Memory
REC	Receive Error Counter
RGB	Red, Green, Blue
RLP	Runtime Loadable Procedure
RTC	Real Time Clock
SDK	Software Development Kit
SDO	Service Data Object
SDHC	Secure Digital High Capacity
SDRAM	Synchronous Dynamic Random Access Memory
SDS	Smart Distributed System
SD/MMC	Secure Digital/Multi Media Card
SPI	Serial Peripheral Interface
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol/Internet Protocol
TEC	Transmit Error Counter
TFT	Thin Film Transistor
TIFF	Tag Image File Format
TTL	Transistor Transistor Logic
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
USART	Universal Synchronous/Asynchronous Receiver and Transmitter
USB	Universal Serial Bus
VGA	Video Graphics Array
WLAN	Wireless Local Area Network