

V řadě aplikací se stávají osobní počítače řídicími jednotkami systémů sběru dat, případně systémů řízení a regulace procesů. To ve spolupráci se speciálními jednoúčelovými přístroji přes často velmi složité interface definované hardwarovými a softwarovými axiomy. Protokoly přenosu, dané těmito axiomy, jsou často tak složité, že užití komunikace mimo aplikační vrstvu je neekonomické, ne-li nemožné. V komerční sféře se stává dobrým artiklem obchodu tvorba knihoven zjednodušujících užití komunikačních interface.

Jihočeská univerzita v Českých Budějovicích
Pedagogická fakulta
katedra informatiky

Interface v .NET (Compact) Framework

BAKALÁŘSKÁ PRÁCE

vedoucí bakalářské práce
Ing. Václav Novák, CSc.

diplomant
David Strnad

České Budějovice 2005

Knihovna JU - PF



3 1 1 5 1 7 1 7 3 2

Rád bych touto cestou poděkoval vedoucímu diplomové práce Ing. Václavu Novákovi, CSc. za pomoc, rady a připomínky k obsahu této práce.

Obsah

UVOD

1 KOMUNIKAČNÍ ROZHRANÍ

1.1 Úvodní kapitola

1.1.1 Úvodní kapitola

1.1.2 Úvodní kapitola

1.1.3 Úvodní kapitola

1.1.4 Úvodní kapitola

1.1.5 Úvodní kapitola

1.1.6 Úvodní kapitola

1.1.7 Úvodní kapitola

1.1.8 Úvodní kapitola

1.1.9 Úvodní kapitola

1.1.10 Úvodní kapitola

1.1.11 Úvodní kapitola

1.1.12 Úvodní kapitola

1.1.13 Úvodní kapitola

1.1.14 Úvodní kapitola

1.1.15 Úvodní kapitola

1.2 Úvodní kapitola

1.2.1 Úvodní kapitola

1.2.2 Úvodní kapitola

1.2.3 Úvodní kapitola

1.2.4 Úvodní kapitola

1.2.5 Úvodní kapitola

1.2.6 Úvodní kapitola

1.2.7 Úvodní kapitola

1.2.8 Úvodní kapitola

1.2.9 Úvodní kapitola

1.2.10 Úvodní kapitola

1.2.11 Úvodní kapitola

1.2.12 Úvodní kapitola

1.2.13 Úvodní kapitola

1.2.14 Úvodní kapitola

1.2.15 Úvodní kapitola

1.2.16 Úvodní kapitola

1.2.17 Úvodní kapitola

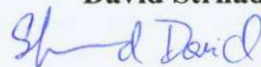
1.2.18 Úvodní kapitola

1.2.19 Úvodní kapitola

1.2.20 Úvodní kapitola

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a že jsem uvedl veškerou použitou literaturu a internetové zdroje, ze kterých jsem čerpal.

David Strnad



Obsah:

ÚVOD	9
1 KOMUNIKAČNÍ ROZHRANÍ	10
1.1 Sériová komunikace	10
1.1.1 RS-232 (COM)	10
1.1.2 PS/2	11
1.1.3 USB	11
1.1.3.1 Základní parametry rozhraní USB	11
1.1.3.2 Používané standardy	11
1.1.3.3 Přenos dat	12
1.1.4 IrDA	12
1.1.5 Fire Wire (IEEE 1394)	13
1.1.6 Bluetooth	13
1.1.7 Wi-Fi (Wireless Fidelity)	14
1.2 Paralelní komunikace	14
1.2.1 Paralelní port LPT	15
1.2.1.1 SPP Mode	15
1.2.1.2 Nibble Mode	15
1.2.1.3 Byte Mode	16
1.2.1.4 EPP Mode	16
1.2.1.5 ECP Mode	16
2 SÉRIOVÉ ROZHRANÍ RS-232	18
2.1 Popis standardu RS-232	19
2.1.1 Elektrické vlastnosti	19
2.1.2 Přenosová rychlost a délka vedení	20
2.1.3 Definice signálů	21
2.1.3.1 Zapojení konektorů pro RS-232	23
2.1.3.2 Popis signálů	25
2.2 Parametry datového přenosu	27
2.2.1 Baud Rate	27
2.2.2 Parita	27
2.2.3 Stop bity	28
2.2.4 Data bity	28
2.3 Řízení toku dat (Handshaking)	29
2.3.1 Co je handshaking	29
2.3.2 Hardwarový handshaking	29

2.3.3 Softwarový handshaking	29
2.4 Synchronní a asynchronní přenos dat	30
2.4.1 Synchronní přenos.....	30
Základní vlastnosti synchronního přenosu:	30
2.4.2 Asynchronní přenos	30
Základní vlastnosti asynchronního přenosu:.....	30
2.4.3 Synchronizace RS-232	31
2.5 Používané kabely	31
2.6 Řadiče portů.....	32
2.7 Přenos dat na větší vzdálenost.....	32
2.7.1 Rozhraní RS-422.....	33
2.7.2 Rozhraní RS-485	33
3 .NET FRAMEWORK.....	35
3.1 Popis prostředí .NET Framework.....	35
3.1.1 Common Language Runtime	36
3.1.2 Common Language Specification	36
3.1.3 .NET Framework Class Library	37
3.1.4 Common Type System	37
Hodnoty a objekty.....	38
Typy a assembly	38
Typy a jmenné prostory	38
3.2 Popis prostředí .NET Compact Framework.....	39
3.2.1 Common Language Runtime	39
3.2.2 Common Language Specification	39
3.2.3 Assembly a formáty souborů.....	39
3.2.4 Threading model	40
3.2.5 Podpora standardů.....	40
3.2.6 ASP.NET.....	40
3.2.7 Knihovna tříd	40
3.2.8 Události (Events).....	41
3.2.9 Regulární výrazy	41
3.2.10 Síťové prostředí.....	41
3.2.11 COM a callback funkce.....	41
3.2.12 Datové typy a přesnost ve výpočtech s pohyblivou čárkou	41
3.2.13 Jména souborů a cesty.....	42
3.2.14 Windows Forms a grafika	42
3.2.15 Arrays.....	42
3.2.16 Common Type System	42
3.3 .NET Framework 1.1.....	42
3.3.1 Nové a vylepšené znaky	43

3.3.2	Nová dokumentace	43
3.3.3	Kompatibilita verzí	44
3.4	.NET Framework 2.0	44
3.4.1	Generické typy	44
3.4.2	Podpora 64bitových platforem	45
3.4.3	Podpora Edit and Continue	45
3.4.4	Detekce změn v síťovém připojení	45
3.4.5	Vylepšení I/O	45
3.4.6	Podpora FTP	46
3.4.7	Globalizace	46
4	TŘÍDY PRO KOMUNIKACI V .NET (COMPACT) FRAMEWORK	47
4.1	Třídy pro ovládání IrDA	47
4.1.1	Odesílání dat přes IrDA	48
4.1.2	Přijímání dat přes IrDA	50
5.	PRAKTICKÁ ČÁST – TVORBA KNIHOVNY PRO SÉRIOVOU KOMUNIKACI	53
5.1	Struktura knihovny	53
5.2	Využívané API funkce	54
5.3	Konstruktory	55
5.3.1	Vytvoření objektu s implicitními hodnotami	55
5.3.2	Vytvoření objektu s definovanými hodnotami	55
5.4	Metody	56
5.4.1	Odesílání dat na COM port	57
5.4.2	Čtení dat z COM portu	57
5.5	Vlastnosti	58
5.5.1	Ukázka použití vlastností	58
6	MOŽNOSTI SÉRIOVÉ KOMUNIKACE V .NET FRAMEWORK 2.0	59
6.1	System.IO.Ports	59
6.1.1	Třída SerialPort	60
6.2	Použití třídy SerialPort	63
6.2.1	Vytvoření objektu SerialPort	63
6.2.1.1	Vytvoření objektu SerialPort s implicitními hodnotami	63

6.2.1.2 Vytvoření objektu SerialPort s definovanou hodnotou.....	63
6.2.1.2 Vytvoření objektu SerialPort s definovanými hodnotami.....	63
6.2.2 Odesílání dat na COM port	63
6.2.2 Čtení dat z COM portu	64

ZÁVĚR.....	66
-------------------	-----------

POUŽITÁ LITERATURA	67
---------------------------------	-----------

POUŽITÉ INTERNETOVÉ ZDROJE	68
---	-----------

PŘÍLOHY	69
----------------------	-----------

Úvod

Cílem této bakalářské práce je seznámit čtenáře s použitím sériové komunikace a na vybraných příkladech ukázat její možnosti a použití na platformách .NET Framework a .NET Compact Framework.

Práce je rozdělena do šesti kapitol. V první jsou popsány druhy komunikace a různá, dnes používaná rozhraní. Dále je vybráno a podrobněji popsáno sériové rozhraní RS-232. Následující kapitola se zabývá platformou .NET Framework včetně její verze Compact pro mobilní zařízení. Čtvrtá kapitola pojednává o komunikaci pomocí infračerveného rozhraní pro .NET Compact Framework. Praktická část bakalářské práce je popsána v následující kapitole spolu s krátkými ukázkami programového kódu. Poslední kapitola ukazuje použití profesionálně vytvořených tříd pro sériovou komunikaci, která je součástí nové verze .NET Frameworku verze 2.0.

Kompletní zdrojové kódy ukázkových aplikací a knihovny (včetně dokumentace) jsou vzhledem k rozsahu na přiloženém kompaktním disku.

1 Komunikační rozhraní

Komunikačním rozhraním se rozumí jakákoli zásuvka v počítači, s jejíž pomocí může počítač komunikovat s dalšími zařízeními.

Poslední dobou se stále více využívá takových připojení, kde není třeba žádného mechanického propojení. Takovým komunikačním rozhraním je například Infračervený port nebo Bluetooth.

Komunikace může probíhat buďto sériově nebo paralelně. Mezi nejběžnější používaná rozhraní v počítači patří sériové porty PS/2, COM (RS-232), modernější USB a paralelní port LPT.

1.1 Sériová komunikace

Sériový přenos znamená, že pro přenos informace od odesílatele k příjemci je jen jedna dvojice drátů (jeden drát a zemnění), takže jednotlivé bity každého znaku se musí přenášet postupně za sebou, tj. sériově.

Sériová komunikace může probíhat synchronně nebo asynchronně.

1.1.1 RS-232 (COM)

RS-232 je sériové rozhraní pro přenos informací mezi dvěma zařízeními do vzdálenosti několika desítek metrů maximální rychlostí 115,2 kb/s.

V osobních počítačích se používají obvykle 2 porty, COM1 a COM2, pro připojení modemů, polohovacích a kreslicích zařízení. Nejtypičtějším použitím sériového portu je připojení myši nebo modemu.

Pro sériový port COM se používají 9 nebo 25pinové konektory.

1.1.2 PS/2

Toto sériové rozhraní dnes slouží pro připojení myši (nahradilo COM) a klávesnice k osobnímu počítači nebo notebooku.

1.1.3 USB

Rozhraní USB (Universal Serial Bus) představuje nejnovější standard pro sériový přenos dat, který během posledních pár let pomalu vytlačuje klasické sériové porty počítače založené na RS-232 a v řadě případů nahrazuje také paralelní port a SCSI rozhraní pro připojení periférií.

1.1.3.1 Základní parametry rozhraní USB

Oproti RS-232 má USB řadu výhod, podporuje rychlost od 1,5 Mbit/s do 480 Mbit/s, umožňuje připojení až 127 zařízení na jediný kořenový rozbočovač. Připojená zařízení mohou být napájena přímo z USB sběrnice, pokud jejich odběr nepřekročí 100 mA , případně 500 mA, pokud je na celé sběrnici maximálně jedno zařízení. Napájecí napětí je 5 V. Rozhraní USB odpovídá plně standardu Plug and Play.

1.1.3.2 Používané standardy

V současné době se používají dvě verze USB rozhraní, které se liší maximální přenosovou rychlostí. USB 1.1 používající rychlosti 1,5 Mbps (Low Speed) nebo 12 Mbps (Full Speed) a USB 2.0 dosahující maximálně 480 Mbps (High Speed).

Jednotlivé standardy jsou vzájemně kompatibilní, lze tedy bez problémů připojovat zařízení s rozhraním USB 1.1 k rozbočovači USB 2.0 a naopak. Přenosová rychlost pochopitelně odpovídá pomalejšímu řadiči.

1.1.3.3 Přenos dat

Pro veškerou komunikaci mezi počítačem a funkční jednotkou jsou k dispozici tři typy paketů. Každá výměna dat začíná tím, že počítač vyšle tak zvaný token packet obsahující popis typu a směru výměny dat, adresu USB zařízení a číslo koncové jednotky (endpoint number). Pak zařízení, které má vysílat data, vyšle datový paket nebo indikuje, že žádná data nejsou k dispozici. Příjemcí strana nakonec vyšle handshake packet, kterým informuje, zda přenos proběhl úspěšně.

Existují zde dva typy přenosového modelu. Prvním je tok dat (stream) využívající izochronní přenos dat v reálném čase. Nemá přesně definovanou strukturu. Druhý typ přenosového modelu je zpráva (message) využívající asynchronní přenos. Zpráva má přesnou strukturu.

Pro kódování je použit kód NRZI (not-return-to-zero recording). Zabezpečení přenosu zajišťuje CRC (Cyclic Redundancy Check).

1.1.4 IrDA

Infračervené záření se používá již poměrně dlouhou dobu jako přenosové medium pro dálkové ovladače, počítače a tiskárny. Donedávna však neexistoval žádný standard, popisující vrstvy přenosu. Standard nazvaný IrDA (Infrared Data Association), byl představen až v roce 1993 firmami HP, IBM a Sharp. Hlavními výhodami IrDA jsou snadný přenos mezi různými aplikacemi i různým hardware, jednoduchá a poměrně levná implementace, nízký příkon umožňující použití v mobilních zařízeních a efektivní a spolehlivý datový přenos.

IrDA standart pro přenos dat infrazářením se úspěšně vyvinul od IrDA 1.0 s přenosovou rychlostí 115,2 Kbps po IrDA 1.1 s přenosem až 4 Mbps během posledních několika let. Pro popis struktury komunikačního systému se často používá OSI model (Open Systems Interconnection), přijatý

mezinárodním normalizačním úřadem ISO, rozděluje komunikační systém do sedmi vrstev. Jednotlivé vrstvy na sebe hierarchicky navazují. V praxi se však nepoužívá všech sedm vrstev a jejich počet je redukován. Model IrDA komunikačního systému vychází ze třech základních vrstev, fyzické, IrLAP a IrLMP.

1.1.5 Fire Wire (IEEE 1394)

Sériová sběrnice Fire Wire byla navržena v roce 1995 firmou Apple. Její původní určení bylo především pro přenos audia a videa zejména z digitálních kamer. Dnes ji lze použít i pro připojení externích disků a optických CD i DVD mechanik.

K jednomu portu lze připojit až 63 zařízení uspořádaných do stromové nebo hvězdicové struktury. Podporuje připojení a odpojení za chodu (Hot Swap) a technologii Plug and Play.

Podle normy z roku 1995 byla sběrnice schopna přenést data rychlostí 50 Mb/s. Její modifikace IEEE 1394a z roku 1999 již 100, 200 a 400 Mb/s a poslední verze IEEE 1394b dokáže komunikovat rychlostí 800 Mb/s a 3,2 Gb/s.

1.1.6 Bluetooth

V roce 1998 byl za společného úsilí firem 3Com, Ericsson, IBM, Intel, Lucent, Microsoft, Motorola, Nokia a Toshiba vydán popis otevřeného standardu pro bezdrátové připojení zařízení na krátké vzdálenosti označeného jako Bluetooth. Standard byl navržen pro jednoduché a stálé propojení mezi různými komunikačními zařízeními, jako jsou telefony, počítače, počítačové periferie, PDA apod.

Standard Bluetooth byl navržen jako bezdrátový komunikační standard, který využívá komunikační frekvenční pásmo ISM (Industrial Scientific

Medical), jež je volně k použití (v tak zvaném bezlicenčním pásmu 2,4 GHz). Standard je koncipován tak, aby podporoval typy přenosů point-to-point i point-to-multipoint.

V červenci roku 1999 byla zveřejněna první specifikace standardu Bluetooth 1.0. V současné době je platná specifikace Bluetooth verze 1.1.

Bluetooth se vyznačuje nízkými nároky na napájení, dosahem do 10 metrů a zabezpečovacím protokolem se 128bitovým klíčem. Data lze ve fyzické vrstvě přenášet rychlostí až 1 Mbit/s, skutečná propustnost je přibližně 720 kbit/s.

1.1.7 Wi-Fi (Wireless Fidelity)

Wi-Fi (Wireless Fidelity) je bezdrátová technologie používaná v bezlicenčním pásmu 2,4 GHz, založenou na protokolu 802.11b. Používá se především pro připojení k internetu nebo v menších WLAN sítích.

Specifikace 802.11b operuje v nelicencovaném pásmu 2,4 GHz a nabízí přenosovou rychlost 11 Mbit/s. Standard 802.11a umožňuje komunikaci rychlostí až 54 Mbit/s v licenčním pásmu 5 GHz. Další specifikace 802.11g umožňuje navýšení rychlosti v pásmu 2,4 GHz až na 20 Mbit/s se zpětnou kompatibilitou s 802.11b.

1.2 Paralelní komunikace

Paralelní porty dovolují posílat současně všechny bity daného znaku přes osm (nebo násobek osmi) souběžných vedení. Na rozdíl od sériového portu přenáší najednou celý byte, tj. paralelně.

Paralelní přenos se používá zejména uvnitř počítače na jeho sběrnicích, ale i třeba pro komunikaci s paralelní tiskárnou.

1.2.1 Paralelní port LPT

Paralelní port LPT je standardní součástí běžných počítačů PC pro paralelní komunikaci s periferiemi pomocí 17 digitálních linek, které lze rozdělit na 8 datových signálů a 9 signálů pro řízení komunikace (handshaking). PC s periferií lze spolehlivě propojit na vzdálenost 2 m. V praxi lze komunikovat na vzdálenost maximálně 5 m, při používání stíněných datových vodičů komunikačního kabelu.

Původně byl paralelní port pro PC vytvořen pro komunikaci s tiskárnou, tedy jednosměrný přenos dat od PC do tiskárny. Později však byl dalšími módy uzpůsoben i pro obousměrný přenos dat rychlostí až několika MB/s. Paralelní port byl v roce 1994 standardizován pod IEEE 1284. Tento standard definuje 5 módů činnosti: SPP Mode - Compatibility Mode (Centronics Mode), Nibble Mode, Byte Mode, EPP (Enhanced Parallel Port) Mode a ECP (Extended Capability Port) Mode.

1.2.1.1 SPP Mode

Tento mód je taky často označován jako Centronics Mode. Je to základní mód pro standardní paralelní port, který byl původně určen pouze pro komunikaci PC s tiskárnou. Definován je jen přenos od PC k periférii a rychlost přenosu se pohybuje jen do 150 KB/s. Byl vyvinut firmou Centronics v roce 1981.

1.2.1.2 Nibble Mode

Nibble Mode je cesta, jak získat zpětný kanál, neboli obousměrnou komunikaci mezi periferií a PC, přímo z klasického SPP módu bez hardwarových úprav. Nutná je tedy jen podpora softwarová. Proto je limit v rychlosti přenosu na 100 KB/s. Výhodou je schopnost používat tuto obousměrnou komunikaci na všech PC s LPT.

K přenosu se využívá 5 vstupních linek portu do PC primárně určených pro handshake. Čtyři z nich se však využívají pro přenos dat do PC. Využitím těchto linek může periferie vyslat byte jako posloupnost dvou tzv. nibblů (4 bity) ve dvou po sobě následujících datových cyklech.

1.2.1.3 Byte Mode

Po implementaci rozhraní paralelního portu, někteří výrobci v čele s IBM zvýšili kapacitu přenosu vypuštěním budiče datových linek a umožnili tak obousměrnou 8-bitovou komunikaci přes datové linky. Přenosová rychlost je 100 až 200 KB/s.

1.2.1.4 EPP Mode

EPP (Enhanced Parallel Port) protokol byl původně vyvinut firmami Intel, Xircom a Zenith Data Systems, aby poskytnul výkonné propojení přes paralelní port a přitom byl kompatibilní se standardním paralelním portem. Poté byl přijat jako standardu IEEE 1284 jako pokročilý (advanced) mód.

Efekt tohoto módu je rychlost přenosu mezi 500 KB/s a 2 MB/s. Přenos se však vždy uskuteční přenosovou rychlostí toho nejpomalejšího komunikujícího zařízení.

EPP protokol poskytuje 4 přenosové cykly: Data Write Cycle, Data Read Cycle, Address Write Cycle a Address Read Cycle. Data cykly jsou určeny pro přenos dat mezi PC a periferií. Adresové cykly jsou určeny k přenosu adresy kanálu, nebo příkazu a řídící informace. Oba je nutné zobrazit jako dva rozdílné přenosové cykly.

1.2.1.5 ECP Mode

ECP (Extended Capability Port) protocol byl stvořen firmami Hewlett Packard a Microsoft jako pokročilý mód pro obousměrnou komunikaci PC s dalšími periferiemi jako je například scanner. Přenosová rychlost v tomto

módu je až 800 KB/s bez využití DMA (Direct Memory Access) a až 2 MB/s při použití DMA. To platí při implementaci na ISA sběrnici. Dnes však při implementaci na PCI sběrnici PC lze dosáhnout 3 až 5 MB/s. Někdy se udává až 8 MB/s.

ECP protokol poskytuje dva komunikační cykly. Jsou to datové cykly (Data cycles) a příkazové cykly (Command cycles).

Dokument "The IEEE 1284 Extended Capabilities Port Protocol and ISA Interface Standard" od společnosti Microsoft definuje vlastnosti a implementaci, které nepostihuje standard IEEE 1284. Například RLE (Run Length Encoding) datovou kompresi, FIFO buffery, DMA a programování.

RLE umožňuje real-time datovou kompresi s poměrem až 64:1, ale musí být povolena na obou komunikujících zařízeních. To je výhodné například při přenosu velkého množství dat obrázku ze scanneru. Adresování kanálů je také odlišné od EPP módu. Pokud je připojeno více zařízení k portu, lze užitím ECP Channel Addressing přistoupit ke každému zařízení. Lze například přijímat data z modemu během doby, kdy datový kanál je busy vlivem tisku tiskárny.

2 Sériové rozhraní RS-232

Standard RS-232 (Recommended Standard no. 232) pro komunikaci přes společné sériové rozhraní byl v šedesátých letech dvacátého století navržen americkou standardizační organizací EIA (Electronic Industries Association). EIA sdružuje především výrobce zařízení pro telekomunikaci. Od roku 1967 byl vydán upravený standard s názvem RS-232-C, jehož název se používá i dnes. Standard se někdy označuje pouze jako RS-232.

Více jak třicet let po uvedení prvního standardu RS-232 byly provedeny některé další modifikace. V roce 1987 změnil označení na RS-232-D a brzy nato, roku 1991 došlo k přejmenování standardu na EIA-232-E Standard. Byly přejmenované některé signálové linky a další nové definované. Poslední verze standardu byla vydána v roce 1997 pod názvem TIA (Telecommunications Industry Association) RS-232-F.

Standard RS-232-C byl s minimálními úpravami převzat i jako standard CCITT (Comité Consultatif International de Télégraphique et Téléphonique) pod označením V.24. Členy CCITT jsou především spojové organizace, které v jednotlivých členských zemích odpovídají za oblast telekomunikací. Standardy, které organizace CCITT vydává, mají formu doporučení, nikoli norem.

Datová komunikace probíhá mezi dvěma zařízeními, mezi počítačem a vzdáleným terminálem nebo mezi dvěma terminály bez použití počítače. Tato zařízení byla spojena telefonní linkou a proto vyžadovala na každém konci modem pro převod signálu. Standard specifikuje napěťové úrovně signálů, časování signálu, funkčnost, protokol pro výměnu informací a konektory pro připojení.

Sériové rozhraní RS-232 je nejužívanějším sériovým rozhraním především díky masovému rozšíření osobních počítačů PC, kde je tento typ rozhraní použit. Stejně rozhraní mají i počítače Challenge DM/L/XL, Onyx, O2, Origin200, Origin2000, Onyx2 a Octane.

Rozhraní RS-232 bylo původně vytvořené pro komunikaci dvou zařízení do vzdálenosti 20 m. Pro větší odolnost proti rušení je informace po propojovacích vodičích přenášena větším napětím, než je standardních 5 V. Přenos informací probíhá asynchronně, pomocí pevně nastavené přenosové rychlosti a synchronizace sestupnou hranou startovacího impulsu.

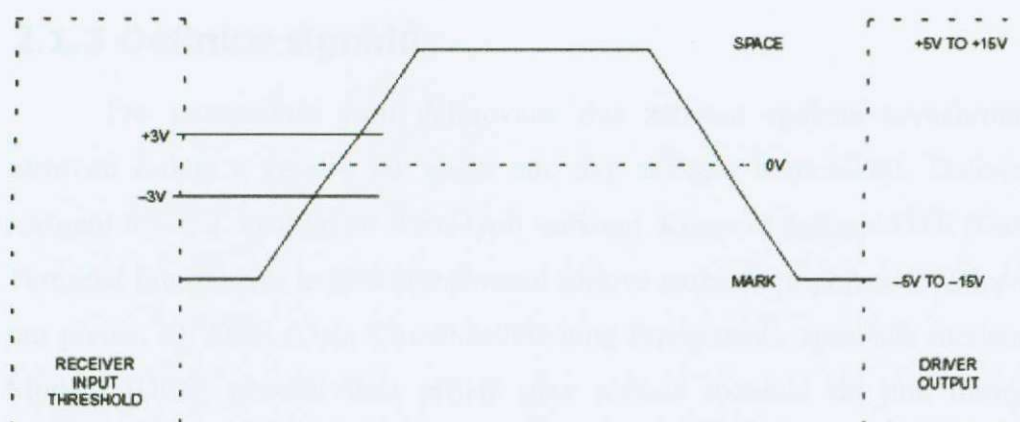
2.1 Popis standardu RS-232

2.1.1 Elektrické vlastnosti

Část standardu RS-232 zabývající se elektrickými vlastnostmi rozhraní specifikuje napěťové úrovně, rychlost změn úrovní signálu a odpor linky.

RS-232 používá dvě napěťové úrovně. Logickou jedničku (1) a logickou nulu (0). Logická 1 je někdy označována jako marking state nebo také klidový stav. Logické 0 se říká space state.

Logická 1 je přenášena zápornou úrovní, zatímco logická 0 kladnou úrovní výstupních vodičů. Nejběžněji se pro generování napětí používá napěťový zdvojovač z 5 V a invertor. Logické úrovně jsou potom přenášeny napětím od +3 do +15 V pro logickou 0 a od -3 do -15 V pro logickou 1 pro vstupní signál. Pro výstupní signál je využíváno napětí od +5 do +15V pro logickou 0 a od -5 do -15V pro logickou 1 .



Obr. 2.1 Specifikace logických úrovní RS-232

2.1.2 Přenosová rychlost a délka vedení

Standard RS-232 definuje odpor rozhraní od $3k\Omega$ do $7k\Omega$. Kabel mezi zařízeními může mít maximální délku 15 metrů, nebo délku vodiče o kapacitě 2500 pF. To znamená, že při použití kvalitních vodičů lze dodržet standard a při zachování jmenovité kapacity prodloužit vzdálenost až na přibližně 50 metrů.

Kabel lze též prodlužovat při snížení přenosové rychlosti, protože potom bude přenos odolnější vůči velké kapacitě vedení. Uvedené parametry počítají s přenosovou rychlostí 19200 Bd.

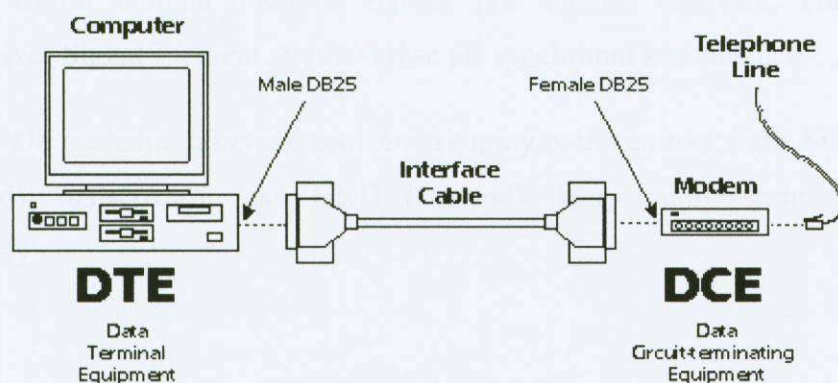
Baud rate [Bd]	Stíněný kabel	Nestíněný kabel
19 200	15	6
9 600	76	30
4 800	152	76
2 400	610	152
1 200	915	152
300	1 220	305
110	1 525	305

Tab. 2.1 Maximální délka kabelu v metrech při dané Baud Rate

Pro přenos dat na větší vzdálenosti je výhodnější používat rozhraní RS-422 nebo RS-485.

2.1.3 Definice signálů

Pro komunikaci jsou definována dvě zařízení spojená asynchronní sériovou linkou a způsob jak spolu tato dvě zařízení komunikují. Definice rozhraní RS-232 vychází ze dvou typů zařízení. Koncové zařízení DTE (Data Terminal Equipment) je přes asynchronní sériové rozhraní připojeno k zařízení pro přenos dat DCE (Data Circuit-terminating Equipment), zpravidla modem. Modem (DCE) převádí data přijatá přes sériové rozhraní do jiné formy. Například analogové a přes jiný typ rozhraní, například telefonní linku, zasílá data jinému modemu (DTE), který přijatá data předává přes svoje asynchronní sériové rozhraní druhému koncovému zařízení (DTE). Stejným způsobem se předávají data opačným směrem.



Obr. 2.2 Propojení DTE a DCE

Mezi koncovým zařízením DTE a komunikačním zařízením DCE definuje norma RS-232 čtyři skupiny signálů.

První skupinou jsou datové signály. Data z DTE do DCE přenáší vodič označený TxD (Transmit Data). Data opačným směrem z DCE do DTE přenáší vodič RxD (Receive Data). Data se přenášejí prostřednictvím tzv. asynchronního protokolu.

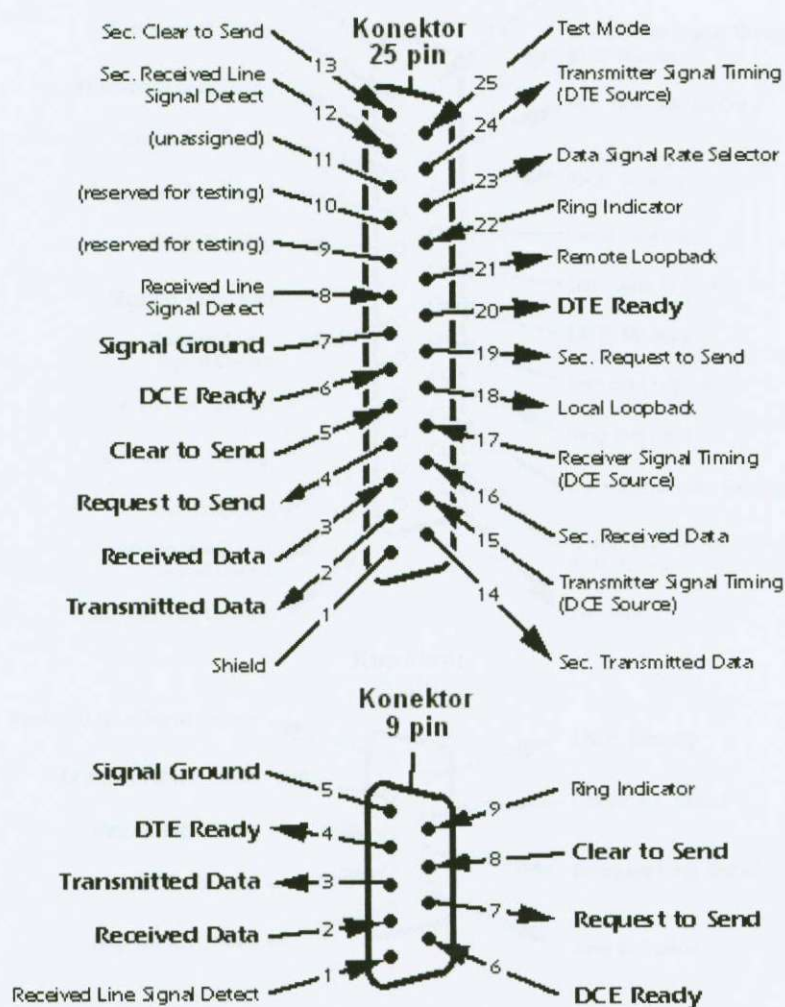
Druhou skupinu tvoří řídicí signály, které se mohou využívat jen z části nebo dokonce nejsou vůbec použity. Dříve plnily tyto řídicí signály funkci

jakýchsi semaforů pro řízení poloduplexní komunikace s modemy. V plně duplexních komunikačních zařízeních ztrácejí řídicí signály částečně svůj původní význam. Programy je využívají způsobem zařízení DTE, které hlásí, že je momentálně připraveno (nepřipraveno) přijmout data. K tomu může programátor využít jak signál DTR, tak signál RTS. Právě tak lze pro zařízení DCE použít signál DSR nebo CTS. Proto některé programy umožňují zvolit, které handshakové signály se pro řízení toku dat použijí. V nastavení takových programů můžeme proto někdy najít více variant hardwarového řízení toku. Jedná se zejména o hardware (RTS/CTS), hardware (DTR/DSR) a podobně. Takto použitý řídicí signál se pak podle nových definic nazývá Ready For Receiving.

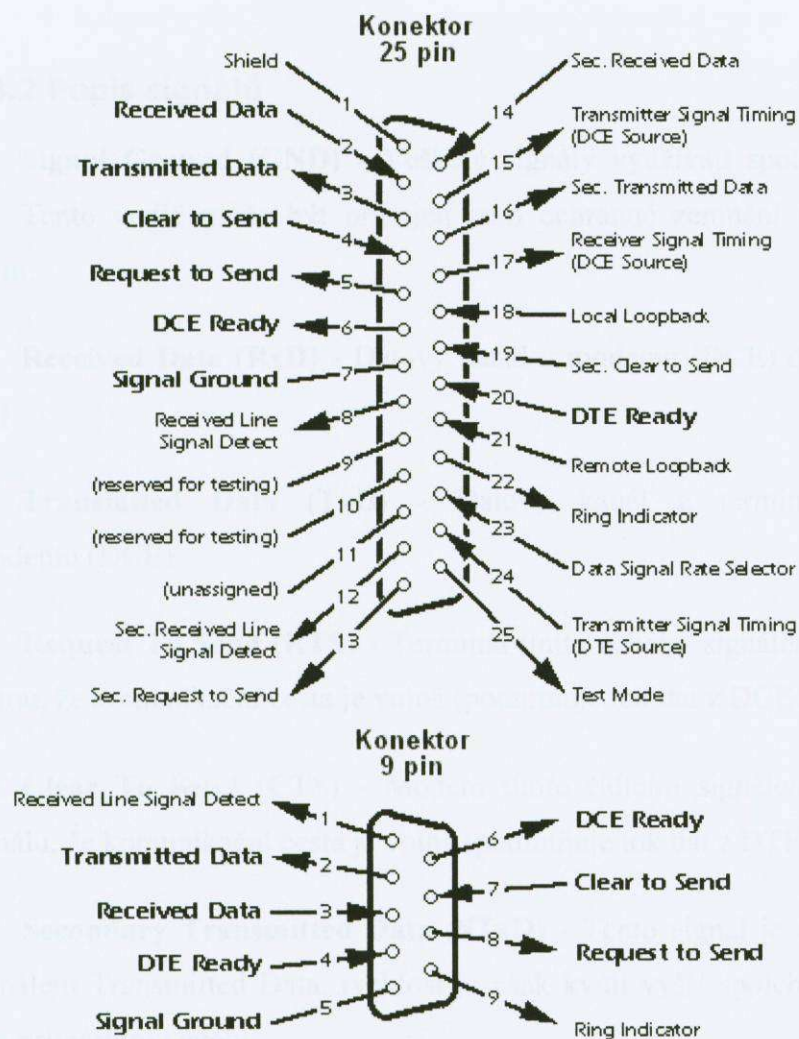
Čtvrtá skupina obsahuje signály pro regulaci časování, Transmitter a Receiver Signal Element. Využívají se při synchronní komunikaci.

Do poslední, takzvané společné skupiny patří zemnicí vodič SG (Signal Ground), označovaný také GND (Ground), který mají všechny signály společný.

2.1.3.1 Zapojení konektorů pro RS-232



Obr. 2.3 Zapojení konektorů DTE zařízení pro RS-232



Obr. 2.4 Zapojení konektorů DCE zařízení pro RS-232

Podle normy je DTE a DCE zařízeními využito pro signál a zemnění 22 z 25 dostupných pinů. Pokud používáme pro komunikaci devítipinový konektor je využita pouze část všech možných signálů.

2.1.3.2 Popis signálů

Signal Ground (GND) - Veškeré signály využívají společný zemní vodič. Tento vodič může být připojen jako ochranné zemnění uvnitř DCE zařízení.

Received Data (RxD) - Datový kanál z modemu (DCE) do terminálu (DTE).

Transmitted Data (TxD) - Datový kanál z terminálu (DTE) do modemu (DCE).

Request To Send (RTS) - Terminál tímto řídicím signálem oznamuje modemu, že komunikační cesta je volná (podmiňuje tok dat z DCE do DTE).

Clear To Send (CTS) - Modem tímto řídicím signálem oznamuje terminálu, že komunikační cesta je volná (podmiňuje tok dat z DTE do DCE).

Secondary Transmitted Data (STxD) - Tento signál je ekvivalentní se signálem Transmitted Data, rychlost je však kvůli vyšší spolehlivosti nižší než na primárním kanálu.

Secondary Received Data (SRxD) - Tento signál je ekvivalentní se signálem Received Data, rychlost je však kvůli vyšší spolehlivosti nižší než na primárním kanálu.

Secondary Request To Send (SRTS) - Tento signál je ekvivalentní se signálem Request To Send.

Secondary Clear To Send (SCTS) - Tento signál je ekvivalentní se signálem Clear To Send.

DCE Ready (DSR) - Podmiňuje otevření sériového propojení. Modem tímto signálem oznamuje terminálu, že je připraven komunikovat. Při jeho absenci funkci přebírá signál DCD.

DTE Ready (DTR) - Řídicí signál, kterým terminál otevírá sériové spojení. Oznamuje modemu, že je připraven komunikovat a přeje si otevřít komunikační kanál..

Received Line Signal Detector (Carrier Detect, CD) - Tento signál je používán, pokud DCE zařízení je modem. Při ztrátě spojení je přes tento signál dán pokyn k „zavěšení“.

Secondary Received Line Signal Detector (Secondary Carrier Detect, SCD) - Tento signál je ekvivalentní s Received Line Signal Detektorem, ale odkazuje se na druhý kanál.

Ring Indicator (RI) - Tento signál je využíván, pokud DCE zařízení je modem. Modem oznamuje terminálu, že na telefonní lince detekoval signál „zvonění“.

Data Signal Rate Selektor - Tento signál může pocházet buď z DTE nebo DCE zařízení a je používán pro výběr jednoho ze dvou předem určených Baud Rate.

Transmitter Signal Element Timing (Transmitter Clock, TC) - Tento signál se využívá pouze když DCE zařízení je modem a je používán synchronní komunikační protokol. Modem generuje časový signál pro kontrolu přesné rychlosti kterou jsou data odesílána z DTE na DCE zařízení.

Receiver Signal Element Timing (Receiver Clock, RC) - Signál RC je využíván podobně jako signál TC s výjimkou toho, že poskytuje informace pro zařízení DTE.

Transmitter Signal Element Timing (External Transmitter Clock, ETC) - Signál ETC je poskytován pouze pro použití modemu a to pokud nejsou využívány signály TC a RC.

Local Loopback (LL) - Tento signál je generován DTE zařízením a je používán pro přepnutí modemu do testovacího stavu.

Remote Loopback (RL) - Signál Remote Loopback je generován DTE zařízením a je využíván podobně jako signál Local Loopback pro přepnutí vzdáleného modemu do testovacího stavu.

Test Mode (TM) - Tento signál se využívá pouze pokud DCE zařízení je modem. Signalizuje, že modem je ve stavu Local Loopback nebo Remote Loopback.

2.2 Parametry datového přenosu

2.2.1 Baud Rate

Přenosová rychlost definuje rychlost přenosu dat z datového média na jiné datové médium. Baud Rate udává počet změn signálu za sekundu. Počet změn se pak vyjadřuje v Baudech.

Baud je jednotka používaná pro měření rychlosti přenosu dat. Jednotka baud je pojmenována po Jean-Maurice-Émile Baudotovi (1845 - 1903).

2.2.2 Parita

Parita je nejjednodušší způsob jak bez nároků na výpočetní výkon zabezpečit přenos dat. Ve vysílacím zařízení se sečte počet jedničkových bitů a doplní se paritním bitem tak, aby byla zachována předem dohodnutá podmínka sudého nebo lichého počtu jedničkových bitů.

Sudá parita je pokud součet jedničkových bitů a paritního bitu je roven sudému číslu.

Lichá parita je pokud součet jedničkových bitů a paritního bitu je roven lichému číslu.

Při použití Space parity neboli nulové parity má paritní bit vždy hodnotu logická 0. Používá se například při komunikaci 7bitového zařízení s 8bitovým, kdy paritní bit nahrazuje logickou 0 poslední bit v byte a tím je zachována kompatibilita s 8bitovým přenosem.

Mark parita je, pokud bude paritní bit nastaven na logickou 1. Při kompenzaci 7bitového provozu je třeba jej na přijímací straně nulovat, jinak není kompatibilní s ASCII.

2.2.3 Stop bity

Stop bit definuje ukončení rámce. Zároveň zajišťuje určitou prodlevu pro přijímač. Právě v době příjmu Stop bitu většina zařízení zpracovává přijatý Byte.

Zdvojený Stop bit se používá u pomalejších zařízení pro dobůh zpracování přijatého znaku. Jedná se o standard na 110 Bd .

2.2.4 Data bity

Při komunikaci nemusíme používat standardní osmibitovou abecedu, ale pomocí data bitů lze nastavit velikost abecedy. Pokud chceme využívat pouze 32 znaků ($32 \text{ je } 2^5$), stačí nám jen abeceda pětibitová.

2.3 Řízení toku dat (Handshaking)

2.3.1 Co je handshaking

DTE vysílá data, která DCE zapisuje do vyrovnávací paměti. Odtud je vybírá a předává dále. Jestliže se vyrovnávací paměť naplní, musí DCE vynutit zastavení vysílání dat z DTE. Naopak po uvolnění vyrovnávací paměti musí DCE opět povolit vysílání dat z DTE. Tento mechanismus se nazývá řízení toku dat (flow control, handshaking). Existují dva různé způsoby řízení toku dat - softwarový a hardwarový.

2.3.2 Hardwarový handshaking

Hardwarové řízení toku dat se zajišťuje zvláštními řídicími signály RTS a CTS. Výhodou oproti softwarovému handshakingu je, že data se mohou přenášet oběma směry zároveň, tj. spojení je duplexní.

2.3.3 Softwarový handshaking

Probíhá na úrovni komunikačních protokolů (ZMODEM, KERMIT...). Jestliže na přijímající straně dojde k naplnění vyrovnávací paměti, vyšle se opačným směrem dohodnutý znak, tj. přijímač vysílači, který pro vysílače znamená pozastavení vysílání. Po uvolnění vyrovnávací paměti pošle přijímač vysílači jiný dohodnutý znak, že je opět připraven přijímat data a vysílač může pokračovat v zasílání dat. V počítačích PC se používají pro softwarový handshaking znaky v Ascii tabulce XON/XOF. Je-li však potřeba v toku dat znaky XON/XOF vyslat, je nutné použít speciální sekvenci znaků, což přenos dat zpomalí.

2.4 Synchronní a asynchronní přenos dat

2.4.1 Synchronní přenos

Synchronní přenos informací znamená, že na nějakém vodiči nebo vodičích se nastaví určitá úroveň, která přenáší informaci a validita informace se potvrdí impulzem nebo změnou úrovně synchronizačního signálu. Synchronizačním signálem se tedy informace kvantují.

Základní vlastnosti synchronního přenosu:

- Je výhodný pro velké objemy dat, přenášené po více vodičích.
- Nutno jednoznačně určit, kdo vysílá synchronizační impulzy.
- Možno použít spojitě proměnnou rychlost přenosu, například podle poměru chybovosti.
- Nutnost synchronizačního vodiče navíc, který v podstatě nepřenáší žádnou užitečnou informaci.
- Na straně zařízení nepotřebuje nijak složitou elektroniku.

2.4.2 Asynchronní přenos

Asynchronní přenos dat přenáší data v určitých sekvencích. Data jsou přenášena přesně danou rychlostí a uvozena startovací sekvencí, na kterou se synchronizují všechna přijímací zařízení. Všechny strany obsahují vlastní přesný oscilátor, díky kterému odečítají data v přesně definovaných intervalech. Po ukončení sekvence je další příjem opět synchronizován startovní sekvencí.

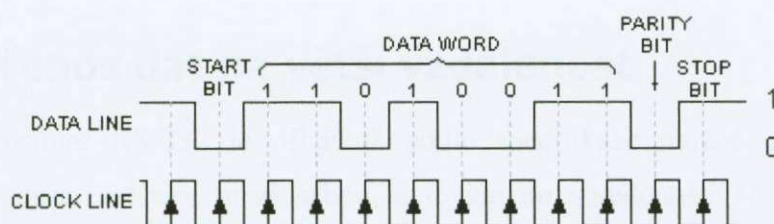
Základní vlastnosti asynchronního přenosu:

- Nevýhodné pro velké objemy dat, ale vhodné pro dlouhá vedení, na nichž by synchronizační vodič činil nezanedbatelné finanční náklady.

- Lze použít pro komunikaci mezi mnoha zařízeními.
- Nutno definovat jednoznačně přenosové rychlosti, změnu rychlosti je třeba ošetřit softwarovou sekvencí, která přiměje počítač změnit hardwarově přenosovou rychlost.
- Celkem složitá a drahá elektronika, nutno použít krystalové oscilátory.
- Až o 20% menší přenosová rychlost užitečných dat při stejné rychlosti komunikace, vzhledem k nutnosti startovacích a paritních bitů

2.4.3 Synchronizace RS-232

RS-232 používá asynchronní přenos informací. Každý přenesený byte konstantní rychlostí je proto třeba synchronizovat. K synchronizaci se používá sestupná hrana tak zvaného Start bitu. Za ní již následují posílaná data.



Obr. 2.5 Synchronizace RS-232

2.5 Používané kabely

Všechny DTE-DCE kabely jsou přímé a vývody jsou propojeny 1:1. DTE-DTE a DCE-DCE kabely patří mezi křížené. DTE - DCE se nazývá 'Straight Cable' (Přímý), DTE - DTE se nazývá 'Null-Modem' a DCE - DCE se nazývá 'Tail Circuit'.

2.6 Řadiče portů

Seznam v BIOSu obsahuje až čtyři základní adresy COM portů. Během POST BIOS testuje a inicializuje COM1 a COM2. Tyto porty jsou tvořeny obvody 8250 u PC/XT nebo 16450 u PC/AT, což jsou zcela programově i vývodově kompatibilní verze. Nevýhodou těchto obvodů je, že jsou schopny si zapamatovat pouze jeden přijatý znak, takže může, hlavně při vyšších přenosových rychlostech, dojít ke ztrátě dat (overrun), když počítač nestihne včas odebrat přijatý znak před příchodem dalšího. Z tohoto důvodu byl vyvinut obvod 16550, opět vývodově i programově kompatibilní, který však má 14ti znakovou přijímací vyrovnávací paměť. Tato paměť musí být programově zapnuta, jinak se obvod chová jako zcela standardní 16450. Je tedy možno jej v komunikačních kartách vyměnit, standardně je použit v systémech PS/2 a v portech osazených na základní desce.

2.7 Přenos dat na větší vzdálenost

Rozhraní RS-232 je oficiálně podle specifikace možné použít pro propojení dvou zařízení mezi sebou, a to jen do vzdálenosti 15 metrů při přenosové rychlosti do 20 kb/s, což vyplývá z povolené kapacity kabelu 2500 pF.

V praxi jsou dosahovány výsledky mnohem lepší, 115200 kb/s při vzdálenosti až 50 metrů, díky použití kabelů s kapacitou pod 1000 pF.

Pro propojení dvou zařízení s rozhraním RS-232 v minimální konfiguraci stačí tři vodiče (RxD, TxD, GND), tak zvaný null modem kabel. Toho se využívá při přenosu dat na větší vzdálenosti přes rozhraní RS-422, RS-485 nebo proudovou smyčku.

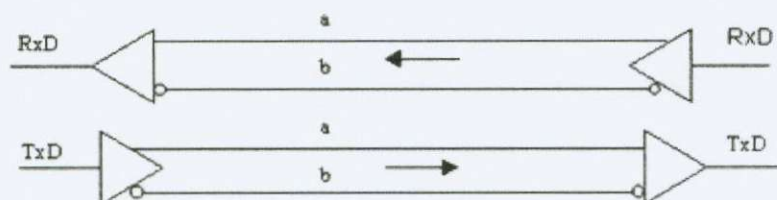
Podle specifikace je dosah těchto rozhraní 1200 metrů a přenosová rychlost 10 Mb/s. Toho lze dosáhnout pokud použijeme stíněnou kroucenou dvojlinku.

2.7.1 Rozhraní RS-422

Linka RS-422 používá jeden pár vodičů pro signál RxD a druhý pro signál TxD. Z toho vyplývá, že použijeme-li linku RS-422 k prodloužení přenosové vzdálenosti místo třídrátové RS-232 (RxD, TxD, GND), nic se nemusí na způsobu komunikace měnit a není tedy třeba ani zásah do software.

Každý ze signálů linky je přenášen po dvojici vodičů. Vodiče označované „a“ a „b“ jsou vysílačem buzeny v protifázi a přijímač vyhodnocuje jejich napěťový rozdíl. Tímto principem se odstraní součtové (aditivní) rušení.

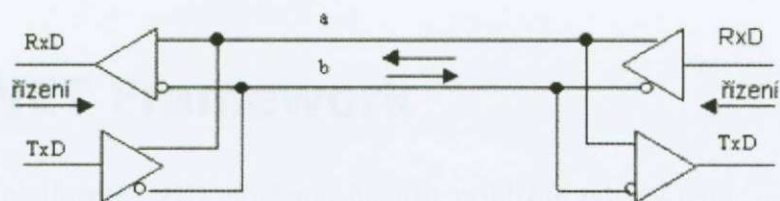
Linky mohou být vedeny až na vzdálenost 1600m (vodiče s kapacitou do 65pF/m) a lze je větvit.



Obr. 2.6 Schéma zapojení linky RS-422

2.7.2 Rozhraní RS-485

Narozdíl od linky RS-422 používá rozhraní RS-485 jen jeden pár vodičů pro oba směry toku dat. Je tedy třeba směr komunikace přepínat a to může být problém zvláště v případech, kdy s touto možností software nepočítá.



Obr. 2.7 Schéma zapojení linky RS-485

2.1 Popis prostředí .NET Framework

Prostředí .NET Framework je platforma pro vývoj aplikací, která umožňuje vývoj aplikací v různých jazycích, jako jsou C#, Visual Basic, J#, a další. Prostředí .NET Framework je založeno na Common Language Runtime (CLR) a .NET Framework Class Library (FCL). CLR je runtime prostředí, které umožňuje vývoj aplikací v různých jazycích, jako jsou C#, Visual Basic, J#, a další. FCL je knihovna tříd, která obsahuje různé třídy, jako jsou třídy pro práci s daty, soubory, sítě, a další.

Prostředí .NET Framework je založeno na Common Language Runtime (CLR) a .NET Framework Class Library (FCL). CLR je runtime prostředí, které umožňuje vývoj aplikací v různých jazycích, jako jsou C#, Visual Basic, J#, a další. FCL je knihovna tříd, která obsahuje různé třídy, jako jsou třídy pro práci s daty, soubory, sítě, a další.

Prostředí .NET Framework je založeno na Common Language Runtime (CLR) a .NET Framework Class Library (FCL). CLR je runtime prostředí, které umožňuje vývoj aplikací v různých jazycích, jako jsou C#, Visual Basic, J#, a další. FCL je knihovna tříd, která obsahuje různé třídy, jako jsou třídy pro práci s daty, soubory, sítě, a další.

3 .NET Framework

O platformě .NET můžeme hovořit přibližně od poloviny roku 2000, kdy byl představen Microsoft .NET SDK (Software Development Kit) spolu s první beta verzí Visual Studio .NET.

3.1 Popis prostředí .NET Framework

Systém Microsoft .NET Framework je rozsáhlou kolekcí tříd určenou pro programátory operačního systému Windows. .NET Framework, stejně jako rozhraní MFC (Microsoft Fundamentation Classes) zastřešuje funkce Windows API. Umožňuje tedy programátorům provádět určité úlohy pomocí méně příkazů než při použití API. Zároveň však dovoluje API funkce používat přímo.

Platforma .NET Framework se skládá ze dvou částí, ze společného jazykového prostředí pro běh aplikací (Common Language Runtime) a knihovny Framework Class Library. Asociace evropských výrobců počítačů (ECMA) přijala v prosinci roku 2001 jako standard programovací jazyk C#, část prostředí Common Language Runtime a knihovny Framework Class Library.

Platforma .NET vylepšila některé nedostatky aktuální platformy Windows. Všechny aplikační služby jsou zpřístupněny přes společný objektově orientovaný model programování. Komponenty aplikace se oddělují, aby aplikace načetla vždy přesně ty komponenty, se kterými byla sestavena a vyzkoušena. Během překladu zdrojového kódu pro .NET Framework kompilátor generuje místo tradičních instrukcí procesoru mezijazyk (Intermediate Language). Po spuštění programu Common Language Runtime překládá mezijazyk do konkrétních instrukcí procesoru. Díky tomu můžeme

aplikace .NET Framework distribuovat na všechny počítače, na kterých běží Common Language Runtime a Framework Class Library. .NET Framework používá ověřování typové kompatibility a nelze použít ani odkaz na libovolné místo paměti. V rámci Common Language Runtime se všechny chyby hlásí pomocí výjimek. Common Language Runtime automaticky sleduje využívání zdrojů a zaručuje, že budou využívány efektivně.

3.1.1 Common Language Runtime

Pro Platformu .NET bylo vytvořeno zcela nové běhové prostředí, které je základní infrastrukturou nad níž je celý Framework vybudován. Common Language Runtime je robustní a bezpečné prostředí pro běh aplikací. Obsahuje vestavěnou funkci Garbage-collection, která na systémové úrovni řídí životnost objektů alokovaných v paměti. Dále překladač z Microsoft Intermediate Language na nativní kód procesoru. Microsoft Intermediate Language je procesorově nezávislý jazyk, který je při spuštění aplikace přeložen Just in Time kompilátorem do kódu procesoru. Aplikace je tedy nezávislá na platformě, na které je spuštěna.

3.1.2 Common Language Specification

Common Language Runtime integruje všechny jazyky a dovoluje, aby objekty vytvořené v jednom jazyku vystupovaly zcela rovnocenně i v kódu napsaném ve zcela jiném jazyku.

Common Language Specification definuje pravidla, kterých se musí držet pouze navenek viditelné typy a metody, mají-li být dosažitelné z libovolného programovacího jazyka, jež vyhovuje standardu Společné jazykové specifikace.

Díky Společné jazykové specifikaci je možné používat knihovny pro různé jazyky a ve Visual Studiu tedy programovat v téměř jakémkoli objektově orientovaném programovacím jazyku.

3.1.3 .NET Framework Class Library

.NET Framework obsahuje třídy, rozhraní a datové typy, které urychlují a optimalizují proces vývoje a poskytování přístupu k funkcím systému.

Typy jsou základy, na kterých jsou vystavěné jednotlivé aplikace a komponenty. .NET Framework obsahuje typy, které provádějí následující úlohy:

- reprezentují základní datové typy a výjimky,
- zahrnují datové struktury,
- provádějí vstupní a výstupní operace,
- zpřístupňují informace o použitých typech,
- vyvolávají bezpečnostní kontroly
- poskytují přístup k datům klientů s grafickým rozhraním.

.NET Framework poskytuje bohatou sadu rozhraní, abstraktních i konkrétních (neabstraktních) tříd. Lze využívat již připravené konkrétní třídy, vytvářet vlastní třídy a rozhraní nebo odvozovat nové od již existujících tříd nebo rozhraní.

3.1.4 Common Type System

Společný typový systém podporuje dvě základní kategorie typů. Jsou to hodnotové typy a typy referenční. Všechny typy jsou odvozené od základního typu `System.Object`. Tento objekt je tedy kořenem všech ostatních typů a zaručuje, že každá instance typu bude obsahovat minimální množinu chování,

testování dvou instancí na rovnost, výpočet hashovacího kódu instance, zjištění typu instance, provedení mělké kopie a reprezentaci ve formě textového řetězce.

Hodnotové typy přímo obsahují svá data. Instance hodnotového typu je alokována v zásobníku nebo inline ve struktuře. Hodnotové typy mohou být vestavěné, definované uživatelem nebo enumerické.

Referenční typy odkazují na adresu položky v paměti a jsou alokovány na heapu. Referenční typy mohou být samopopisné, pointery nebo interface typy. Samopopisné typy jsou dále rozděleny na arrays a class typy. Class typy jsou třídy definované uživatelem, zapouzdřené typy a delegáti.

Hodnoty a objekty

Hodnoty jsou binární reprezentací dat, cestu pro interpretaci těchto dat poskytují typy. Každá hodnota má přesný typ, který definuje její reprezentaci a operace, které jsou pro ni definovány. Hodnoty samopopisných typů jsou nazývány objekty. Vždy je možné určit přesný typ objektu pomocí jeho hodnoty.

Typy a assembly

Běhové prostředí využívá assembly pro nalezení a nahrání typu. Manifest obsažený v assembly obsahuje informace, které využívá prostředí pro rozeznání všech typů referencí vytvořených v assembly.

Typy a jmenné prostory

Z hlediska běhového prostředí je jmenný prostor kolekce jmen typů. Jmenné prostory jsou hierarchicky uspořádány s určitými vztahy mezi sebou. Při práci s textovými řetězci lze využít jmenný prostor System.String, při práci s formuláři prostor System.Windows.Forms apod.

3.2 Popis prostředí .NET Compact Framework

.NET Compact Framework je prostředí vytvořené pro vývoj aplikací pro mobilní zařízení.

.NET Compact Framework je hardwarově nezávislé prostředí pro běh programů na zdrojově omezených zařízeních jako jsou PDA (Personal Data Assistants), Pocket PC, mobilní telefony a různá speciálně navržená zařízení s vestavěným operačním systémem Windows CE .NET.

.NET Compact Framework je podmnožina knihovny tříd .NET Framework, dědí tedy architekturu Common Language Runtime a Manager Code Execution z .NET Framework.

.NET Compact Framework má mnoho společného s .NET Framework, ale jsou zde i rozdíly.

3.2.1 Common Language Runtime

Common Language Runtime na obou platformách využívá řízeného provádění kódu, just-in-time kompilace kódu a garbage collectoru. Oba podporují Common Language Specification. Velikost Common Language Runtime .NET Compact Frameworku je však přibližně dvanáctiprocentní oproti kompletnímu Common Language Runtime .NET Frameworku.

3.2.2 Common Language Specification

Prolínání jazyků je vestavěno i do .NET Compact Frameworku. Lze využívat jazyky Visual C#, Visual Basic .NET a některé další.

3.2.3 Assembly a formáty souborů

Aplikace vytvořené v .NET Framework i .NET Compact Framework používají Assembly. Aplikace přistupují k přenosně vykonatelným PE

(Portable Executable) souborům, které obsahují Microsoft Intermediate Language a metadata jenž definují tuto .NET (Compact) Framework aplikaci. PE soubor může odkazovat na programové jmenné prostory definované a sdílené v jiných souborech s assembly.

Ačkoli formáty souborů obou Frameworků jsou identické, nelze binární komponenty kompilované v .NET Compact Frameworku spouštět v .NET Frameworku, ani naopak. Příčinou jsou přísně závazná pravidla. Lze však rekompilovat aplikaci do druhého Frameworku.

3.2.4 Threading model

Oba Frameworky poskytují programování vícevláknového modelu, takže používají plánovací mechanismus hostitelského systému.

3.2.5 Podpora standardů

Jádro technologie implementované do obou Frameworků vyhovuje standardu European Compute Manufactures Association Common Language Infrastructure.

3.2.6 ASP.NET

.NET Compact Framework je především bohatá klientská platforma, ale neposkytuje podporu ASP.NET. Pro vývoj webových stránek pro mobilní zařízení lze použít ASP.NET Mobile Web Controls.

3.2.7 Knihovna tříd

.NET Compact Framework podporuje podmnožinu .NET Framework Class Library. Tato podmnožina je vhodná pro aplikace navržené pro běh na zdrojově omezených zařízeních a je kompatibilní se stejně pojmenovanými třídami v .NET Frameworku.

3.2.8 Události (Events)

.NET Compact Framework podporuje události GotFocus a LostFocus, ale nepodporuje události Activated a Deactivated.

3.2.9 Regulární výrazy

Aplikace používající regulární výrazy v .NET Compact Frameworku nejsou binárně kompatibilní s aplikacemi používajícími regulární výrazy v .NET Frameworku, jsou však kompatibilní z hlediska zdrojového kódu.

3.2.10 Síťové prostředí

.NET Compact Framework poskytuje třídy pro infračervené spojení (IrDA) a třídy pro HTTP požadavky mobilních zařízení. Tyto třídy jsou dostupné pouze v .NET Compact Framework.

3.2.11 COM a callback funkce

.NET Compact Framework nepodporuje vnitřní operace s COM objekty, ale lze použít platformu PInvoke, odvolávající se na DLL funkce, které mohou COM objekty volat. ActiveX, Windows Scripting Host a callback funkce využívající COM rozhraní nejsou podporovány.

3.2.12 Datové typy a přesnost ve výpočtech s pohyblivou čárkou

Podporována je jen metoda Math.Round(double a), nikoli však Math.Round(double a, double b).

Při dělení, pokud je dělitel hodně velký (resp. malý) nebo nabývá maxima (resp. minima) hodnoty pohyblivé čárky, výpočet namísto správného výsledku vrací nulu.

Platforma MIPS nepodporuje kompletní specifikaci standardu pro výpočty s binární pohyblivou čárkou a může způsobit nepředvídatelné výsledky. .NET Compact Framework neposkytuje schopnosti emulace pohyblivé čárky.

3.2.13 Jména souborů a cesty

Windows CE .NET nepoužívá upřesňující informace o cestě souborů jakmile jsou v kořenovém adresáři zařízení. .NET Compact Framework používá URI řetězců podle prefixového schématu file:// n rozdíl od .NET Frameworku.

3.2.14 Windows Forms a grafika

Pro .NET Compact Framework jsou speciálně vyvinuté ovladače Windows.Forms. Oproti .NET Framework jsou zde tedy rozdíly.

3.2.15 Arrays

Ačkoli některé jazyky podporují indexování od jiného prvku, než je nula, Common Language Runtime nikoli. Pokud první člen je jiný než nula, vyhodí výjimku `MissingMethodException`.

3.2.16 Common Type System

Oba Frameworky mají vestavěné primitivní typy tak jako ostatní typy, od kterých lze odvozovat nové a které lze pro odvození použít při vytváření aplikace.

3.3 .NET Framework 1.1

.NET Framework verze 1.1 rozšiřuje možnosti verze 1.0, přináší některé nové rysy, zlepšení starších a vylepšenou dokumentaci.

3.3.1 Nové a vylepšené znaky

ASP.NET Mobile Controls rozšiřuje možnosti .NET Frameworku a Visual Studio .NET pro programování aplikací s podporou mobilních bezdrátových zařízení jakou jsou telefony a PDA. Protože ovládání mobilních zařízení je nyní částí .NET Frameworku, byla tomu přizpůsobena i dokumentace, která je nyní součástí dokumentace .NET Frameworku jako ASP.NET Mobile Controls.

Změnami prošlo i ADO.NET. .NET Framework Data Provider pro ODBC, který byl dříve dostupný pouze jako download z webu, je nyní součástí .NET Frameworku pod jmenným prostorem System.Data.Odbc. Nově byly do objektů DataReader a Connection přidány některé vlastnosti a metody.

.NET Framework verze 1.1 podporuje tzv. side-by-side spouštění, což je schopnost spouštět více verzí aplikace nebo komponenty na jednom počítači. Nově nainstalované verze .NET Frameworku tedy nemají vliv na již nainstalované aplikace. Aplikace si vybere s jakou verzí knihovny má být spuštěna a kterou bude používat.

Novinek se dočkalo i zabezpečení. Aplikace, která obdrží menší než úplnou důvěru systému pro zabezpečení přístupu ke kódu nemůže volat sdílené knihovny. Byla změněna dosavadní politika bezpečnosti aplikací, které přistupují do zóny internetu.

Nově byla do .NET Frameworku verze 1.1 přidána podpora vznikajícího protokolu IP verze 6 (IPv6).

3.3.2 Nová dokumentace

S novou verzí prostředí byla vydána i nová dokumentace, aktualizovaná a doplněná o některé nové informace o .NET Compact Frameworku, podpoře mobilních zařízení a Windows Forms.

3.3.3 Kompatibilita verzí

.NET Framework podporuje, jak zpětnou, tak i dopřednou kompatibilitu verzí. Aplikace vytvořené za použití dřívějších verzí lze spouštět na novějších verzích a naopak i později vytvořené aplikace lze spouštět na dřívějších verzích. Většina aplikací používajících verzi 1.0 tedy poběží i ve verzi 1.1 a rovněž opačně.

3.4 .NET Framework 2.0

Koncem roku 2004 vydal Microsoft první betaverzi Visual Studia 2005 a nový Framework ve verzi 2.0, který opět přináší spoustu novinek a změn. Jsou to webové služby, transakce, vyhledávání a filtrování dat, podpora SMTP, výjimky Security Exception a mnoho dalších.

3.4.1 Generické typy

.NET Framework 2.0 Beta zavádí generické typy, známé z C++ při použití šablon. Je to jedna z nejsložitějších a také nejdůležitějších vlastností nového frameworku. Generickými se mohou stát třídy, struktury, rozhraní i jednotlivé metody.

Hlavním cílem použití generických typů je umožnit vyšší úroveň znovupoužití kódu. Své uplatnění najdou všude tam, kde se opakuje stejný kód několikrát pro různé datové typy.

S generickými typy přichází do programů i silnější typová kontrola.

Pro práci s generickými typy lze použít nový jmenný prostor `System.Collections.Generic`. Nové třídy byly také přidány do `System.Type` a `System.Reflection.MethodInfo`.

3.4.2 Podpora 64bitových platforem

Nová generace 64bitových počítačů umožňuje vytvářet aplikace, které běží rychleji mohou adresovat více paměti než dosavadní 32bitové. V nové verzi .NET Frameworku je již zahrnuta jejich podpora a lze snadno vytvářet 64bitové aplikace.

3.4.3 Podpora Edit and Continue

Betaverze .NET Frameworku 2.0 zavádí podporu Edit and Continue, která pomáhá programátorovi při odladování chyb v aplikacích vytvářených ve Visual Studiu. Může měnit zdrojový kód, zatímco je spuštěn tzv. Break mód. Potom, co aplikuje změny ve zdrojovém kódu, může se znovu spustit kód a pozorovat efekt. Edit and Continue je dostupný v jakémkoli programovacím jazyku, který podporuje Visual Studio.

3.4.4 Detekce změn v síťovém připojení

Třída NetworkChange poskytuje aplikacím oznámení o změnách síťové IP adresy nebo síťové karty. Příčin takovýchto oznámení může být několik. Například odpojení síťového kabelu, pokud se dostane dané zařízení z dosahu, při bezdrátovém připojení nebo hardwarová chyba. Třída NetworkChange poskytuje událost, pokud je detekována změna.

3.4.5 Vylepšení I/O

Vylepšena byla také použitelnost a funkčnost různých tříd pro vstupní a výstupní operace. Nyní lze jednodušeji provádět některé operace, například čtení a zápis textových souborů a získávání informací o jednotkách.

Lze využívat nové třídy v prostoru System.IO.Compression pro čtení a zápis dat s GZIP kompresním a dekompresním standardem. Dále byly přidány třídy pro podporu komunikace přes sériový port v prostoru System.IO.Ports.

3.4.6 Podpora FTP

Aplikace nyní mohou přistupovat k File Transfer Protokolu (FTP), pomocí tříd `WebRequest`, `WebResponse`, `WebClient` a přenášet tímto protokolem v internetu i lokální sítě soubory.

3.4.7 Globalizace

.NET Framework 2.0 poskytuje větší podporu pro vývoj aplikací použitelných pro různé jazyky a kultury.

Pro podporu různých kultur, jejich úpravu a vytváření lze využít třídu `System.Globalization.CultureAndRegionInfoBuilder`. .NET Framework nyní podporuje nejnovější standardy definované konsorciem Unicode. Třída `UTF8Encoding` obsahuje kódování používané pro převod znaků Unicode do bytů.

4 Třídy pro komunikaci v .NET (Compact) Framework

.NET Compact Framework poskytuje třídy pro vývoj aplikací s infračerveným připojením určeným pro mobilní zařízení a osobní počítače. Tyto třídy jsou součástí jmenného prostoru System.Net.Sockets a implementují programování infračerveného portu vyhovující standardu daného specifikací IrDA (Infrared Data Association).

4.1 Třídy pro ovládání IrDA

.NET Framework neobsahuje třídy pro ovládání IrDA. Pro navázání spojení mezi osobním počítačem a mobilním zařízením je třeba na straně počítače použít platformu Windows socket.

IrDA definuje protokol navržený pro podporu point-to-point infračerveného datového přenosu mezi dvěma zařízeními na krátkou vzdálenost. Více zařízení může navázat spojení se stejným hostitelem.

Implementována je funkce serveru a klienta. Klient je zařízení, které zahajuje spojení. Po navázání spojení mohou být vyměňována data. Protože server vyžaduje dodatečný požadavek na zásobník, bývá to obvykle osobní počítač.

Třídy IrDA obsluhují vrstvu Information Access Service (IAS). Vrstva IAS může mít jednu nebo dvě komponenty a to server nebo klient. Server obsahuje kolekci objektů, která popisuje služby a aplikace dostupné pro přichodí spojení na server. Klient posílá dotazy na komponentu serveru pro získání seznamu dostupných spojení. Třída IrDAClient poskytuje přístup

ke klientovi, zjištění požadovaného spojení, jeho otevření a posílání a přijímání dat.

Třída	Popis třídy
IrDAClient	Poskytuje popis dostupného připojení a vytváří objekt připojení pro otevírání a uzavírání připojení k serveru.
IrDADeviceInfo	Poskytuje informace o dostupných připojeních na serveru.
IrDAEndPoint	Zajišťuje vytváření připojení serveru a získávání informací infračerveného portu.
IrDAListener	Přepíná soket do stavu poslechu pro sledování infračerveného spojení ze specifikované služby nebo síťové adresy.

Tab. 4.1 Třídy prostoru System.Net.Sockets pro IrDA

Enumerický typ	Popis
IrDACharacterSet	Popisuje znakovou sadu podporovanou IrDA zařízením, které bylo nalezeno.
IrDAHints	Obsahuje hodnoty označující typ zařízení nebo připojení jako je například fax.

Tab. 4.2 Enumerické datové typy prostoru System.Net.Sockets pro IrDA

4.1.1 Odesílání dat přes IrDA

```
// Vytvoření nové instance IrDA klienta
IrDAClient irClient = new IrDAClient();

// Soubor, který se má odeslat
fileToSend = ".\\Dokumenty\\send.txt";

// Jméno IrDA zařízení
irName = "IrDaprenos";

// Nastaví maximální velikost bufferu pro přenos
// souboru
bufferSize = 256;

// Otevře soubor pro čtení a vrací jeho stream.
Stream fileStream;

try{
    fileStream = new FileStream(fileToSend,
        FileMode.Open);
}
```

```

catch (Exception e) {
    MessageBox.Show("Nelze otevřít " +
        e.ToString());
    return;
}

// Vytvoří IrDA klienta se jménem, které se musí
// shodovat se jménem přijímacího zařízení
try {
    irClient = new IrDACLient(irName);
}
catch (SocketException e) {
    MessageBox.Show("Chyba při vytváření
        socketu: " + e.Message);
    return;
}

// Vrátí základní stream klienta
Stream baseStream = irClient.GetStream();

// Vrací velikost souboru, který se má poslat a
// zapisuje ji do streamu
int fileLength = (int)fileStream.Length;
byte[] length = BitConverter.GetBytes(fileLength);
baseStream.Write(length, 0, length.Length);

// Vytváří buffer pro čtení souboru
byte[] buffer = new byte[bufferSize];

// Čte stream ze souboru a zapisuje na výstupní
// stream
while (fileLength > 0){
    int numRead = fileStream.Read(buffer, 0,
        buffer.Length);
    baseStream.Write(buffer, 0, numRead);
    fileLength -= numRead;
}

// Uzavře streamy a IrDA klienta
fileStream.Close();
baseStream.Close();
irClient.Close();

```

4.1.2 Přijímání dat přes IrDA

```
IrDAListener irListen;  
IrDAClient irClient;  
IrDAEndPoint irEndP;  
IrDADeviceInfo[] irDevices;  
  
// Vytvoří kolekci nalezených zařízení  
irDevices = irClient.DiscoverDevices(2);  
if(irDevices.Length == 0){  
    MessageBox.Show("Nenalezeno žádné infračervené  
        zařízení!");  
    return;  
}  
  
// Použijeme ID prvního nalezené zařízení  
int ID = BitConverter.ToInt32(irDevices[0].  
    DeviceID, 0);  
  
// Soubor, do kterého mají zapsat přijatá data  
fileReceive = ".\\Dokumenty\\receive.txt";  
  
// Jméno IrDA zařízení  
irName = "IrDAprenos";  
  
// Nastaví maximální velikost bufferu pro přenos  
// souboru  
bufferSize = 256;  
  
// Vytvoří stream pro zápis souboru.  
Stream writeStream;  
try{  
    writeStream = new FileStream(fileReceive,  
        FileMode.OpenOrCreate);  
}  
catch(Exception) {  
    MessageBox.Show("Nelze otevřít " + fileReceive  
        + " pro zápis.");  
    return;  
}
```

```

// Vytvoří spojení pomocí třídy IrDAEndPoint a čeká
// na příjem dat
try{
    irEndP = new IrDAEndPoint(irDevices[0].DeviceID,
        irServiceName);
    irListen = new IrDAListener(irEndP);
    irListen.Start();
}
catch(SocketException e){
    MessageBox.Show("Chyba při komunikaci zařízení "
        + irServiceName + ": " + exSoc.ErrorCode);
}

// Vytvoří klientské spojení pro zařízení detekované
// posluchačem
try{
    irClient = irListen.AcceptIrDAClient();
}
catch(SocketException e){
    MessageBox.Show("Nelze přijímat " +
        e.ErrorCode);
    return;
}

// Vrátí základní stream klienta
Stream baseStream = irClient.GetStream();

int numToRead;

// Vytváří buffer pro čtení souboru
byte[] buffer = new byte[bufferSize];

// Čte stream který obsahuje data ze vzdáleného
// zařízení do té doby až nejsou dostupné žádné
// bajty
numToRead = 4;
while(numToRead > 0){
    int numRead = baseStream.Read(buffer, 0,
        numToRead);
    numToRead -= numRead;
}

// Vrací velikost bufferu, který má být zapsán
numToRead = BitConverter.ToInt32(buffer, 0);
// Zapiše stream do souboru

```

```
while (numToRead > 0){  
    int numRead = baseStream.Read(buffer, 0,  
        buffer.Length);  
    numToRead -= numRead;  
    writeStream.Write(buffer, 0, numRead);  
}  
  
// Uzavře streamy, přestane naslouchat a zavře IrDA  
// klienta  
baseStream.Close();  
writeStream.Close();  
irListen.Stop();  
irClient.Close();
```

5. Praktická část – Tvorba knihovny pro sériovou komunikaci

Praktická část popisuje tvorbu knihovny pro komunikaci přes sériový port COM. Hlavním cílem je zjednodušit uživateli komunikaci tím, že místo složitého volání API funkcí může používat mnohem jednodušší rozhraní knihovny.

Knihovna obsahuje metody pro nejčastější operace se sériovým portem COM, což jsou odesílání a přijímání dat v různých reprezentacích (byte, znak, řetězec). Nechybí ani metody a vlastnosti pro různá nastavení sériového portu.

Pro vývoj jsem použil software Microsoft Visual Studio 2003 spolu s prostředím .NET Framework. Zdrojový kód je psán v jazyce Microsoft C#, zkompilevanou knihovnu lze však využít v jakékoli aplikaci napsané pro .NET (Compact) Framework verze 1.0, 1.1 i 2.0.

5.1 Struktura knihovny

Knihovna je postavena na funkcích Windows API, jejichž funkčnost je dále využívána v jednotlivých metodách a vlastnostech. Pro správnou funkci API jsou v knihovně vytvořeny struktury (vnitřní třídy a enumerické typy).

Dále jsou v knihovně obsaženy privátní metody, které lze využívat pouze v dané třídě a uživateli nejsou přímo dostupné. Tyto provádějí základní nastavení sériového portu a jsou využívány ostatními veřejnými metodami, jenž jsou určené pro uživatele knihovny. Pro další nastavení nebo informace o stavu sériového portu slouží vlastnosti třídy.

5.2 Využívané API funkce

Celá knihovna čerpá a využívá některé API funkce, určené pro komunikaci. Jsou to funkce pro otevření (CreateFileW) a uzavření (CloseHandle) portu, funkce pro nastavení (SetupComm, SetCommTimeouts, atd.), funkce pro přenos dat (WriteFile, ReadFile) a funkce pro události vzniklé při komunikaci (EventModify, CreateEvent, atd.).

Název funkce	Popis funkce
APICreateFileW	Funkce vytváří nebo otevírá objekty(porty, adresáře, soubory,...) a vrací jejich handle přes který je možno k objektu přistoupit.
APISetupComm	Funkce inicializuje parametry komunikace a specifikuje komunikační zařízení.
APIWriteFile	Funkce zapíše data do souboru, začíná na pozici, kterou určuje ukazatel.
APIReadFile	Funkce čte data ze souboru.
APISetCommMask	Funkce specifikuje množinu událostí sledovaných pro komunikační zařízení.
APISetCommState	Funkce konfiguruje komunikační zařízení na základě informací z DCB struktury. Funkce reinitializuje veškerá hardwarová a ovládací nastavení.
APISetCommTimeouts	Funkce nastavuje parametry pro operace čtení a zápisu komunikačního zařízení.
APIWaitCommEvent	Funkce čeká na událost vzniklou na komunikačním zařízení.
APIGetCommModemStatus	Funkce získává control-register hodnoty modemu.
APICloseHandle	Uzavře handle otevřeného objektu.
APIWaitForSingleObject	Funkce čeká po určitou dobu na objekt.
APIEventModify	Funkce modifikuje objekt události.
APICreateEvent	Funkce otevírá objekt události.

Tab. 5.1 API funkce použité v knihovně

Některé API funkce pro svou správnou činnost potřebují datové struktury, jako například Device Control Block, které slouží pro uchování parametrů a různých nastavení komunikace.

Název struktury	Popis struktury
EventFlags	Komunikační události pro funkci APISetCommMask používané pro sledování událostí.
DeviceControlBlock	Struktura DCB (Device control block) definuje ovládací nastavení pro sériovou komunikaci.
CommTimeout	Struktura CommTimeout je využívána funkcemi SetCommTimeouts a GetCommTimeouts pro nastavení a získání parametrů komunikačních zařízení.
Flags	Struktura Flags je využívána ve funkci APIEventModify pro modifikaci objektu události.

Tab. 5.2 Pomocné struktury pro API funkce

5.3 Konstruktory

Třída Knihovna obsahuje dva konstruktory. Jeden bez parametrů s implicitními hodnotami a druhý s parametry pomocí níž lze již při vytváření objektu nastavit některé hodnoty používané při komunikaci.

Název konstrukturu	Popis konstrukturu
Knihovna()	Konstruktor Knihovna() inicializuje novou instanci třídy s implicitními hodnotami. Pro komunikaci bude využit port COM1 s hodnotami Baud Rate 9600, 8 data bitů, nebude používán paritní bit a stop bit bude 0.
Knihovna(String, UInt, Byte, Byte, Byte)	Tento konstruktor inicializuje novou instanci třídy s hodnotami jméno portu, Baud Rate, paritní bit, stop bit a datový bit převzatými z parametrů.

Tab. 5.3 Konstruktory třídy

5.3.1 Vytvoření objektu s implicitními hodnotami

```
// Vytvoření objektu Knihovna

Knihovna sp = New Knihovna ();
```

5.3.2 Vytvoření objektu s definovanými hodnotami

```
// Vytvoření objektu Knihovna pro COM3
// s Baud Rate 19 200

static String portName = "COM3";
static UInt baudRate = 19200;
```

```
static Byte parity = 0;
static Byte stopBit = 0;
static Byte dataBit = 8;
```

```
Knihovna sp = new
    Knihovna(portName, baudRate, parity, stopBit,
        dataBit);
```

5.4 Metody

V knihovně jsou obsažené základní metody pro komunikaci, tzn. metody pro otevření (OpenPort) a uzavření portu (ClosePort), odesílání (Send) a příjem dat (ReadString, ReadChar a ReadByte) a nastavení komunikace (SetupComm, SetDCB, atd.).

Název metody	Popis metody
SetupComm()	Metoda pro nastavení komunikace.
SetConnectionTimeout()	Metoda pro nastavení TimeOut.
SetDCB()	Metoda pro nastavení DCB.
CreateCommThread()	Metoda vytváří vlákno.
Init()	Veřejná metoda pro inicializaci objektu a otevření portu.
OpenPort()	Metoda otevírá COM port.
ClosePort()	Metoda uzavírá COM port.
Send(String)	Metoda pošle řetězec na COM port.
Send(Char)	Metoda pošle znak na COM port.
Send(Byte)	Metoda pošle byte na COM port.
ReadString()	Metoda čte řetězec z COM portu.
ReadChar()	Metoda čte znak z COM portu.
ReadByte()	Metoda čte byte z COM portu.
ToString()	Přetížená metoda ToString(), vrací informace o portu.
Dispose()	Metoda Dispose() uzavře port a uklidí po objektu.
CommThread()	Metoda vlákna.

Tab. 5.4 Metody třídy

5.4.1 Odesílání dat na COM port

Po otevření portu pomocí metody Init můžeme poslat data na COM pomocí metody Send s parametry typu řetězec, znak nebo byte.

```
// Otevření portu  
  
sp.Init();  
  
// Odeslání textu na port  
  
String text = "Ahoj";  
sp.Send(text);  
text = "";  
  
// Uzavření portu;  
  
sp.ClosePort();
```

5.4.2 Čtení dat z COM portu

Po otevření portu pomocí metody Init můžeme data z COM portu přijímat pomocí různých metod. Metoda ReadString čte data do řetězce, ReadChar provádí čtení jednoho znaku a ReadByte jednoho bytu ze sériového portu.

```
// Otevření portu  
  
sp.Init();  
  
// Přečtení dat z portu a vypsání do Message Boxu  
  
String text = "";  
text = sp.ReadString();  
MessageBox.Show(text);  
  
// Uzavření portu;  
  
sp.ClosePort();
```

5.5 Vlastnosti

Vlastnosti slouží pro zjišťování stavu portu (IsOpen) a nastavování některých hodnot, jako například jména portu (PortName) nebo Stop Bitů (StopBits) atd.

Název vlastnosti	Popis vlastnosti
IsOpen	Vlastnost vrací true, pokud je port otevřen, v opačném případě false.
PortName	Vlastnost vrací nebo nastavuje jméno portu.
BaudRate	Vlastnost vrací nebo nastavuje BaudRate.
Parity	Vlastnost vrací nebo nastavuje Paritu.
StopBits	Vlastnost vrací nebo nastavuje počet StopBitů na byte.
Size	Vlastnost vrací nebo nastavuje počet DataBitů.

Tab. 5.5 Vlastnosti třídy

5.5.1 Ukázka použití vlastností

```
// Nastavení jména portu na COM2

String portName = "COM2";
sp.PortName = portName;

// Vypsání Baud Rate do Message Boxu

String text = "Baud Rate: ";
baudRate = sp.BaudRate;
text += Convert.ToString(baudRate);
MessageBox.Show(text);
```

6 Možnosti sériové komunikace v .NET Framework 2.0

Nová verze platformy .NET Framework 2.0 Beta obsahuje již vestavěné prostředky pro sériovou komunikaci. Tyto třídy jsou obsaženy ve jmenném prostoru System.IO.Ports a umožňují komunikaci se sériovými vstupními a výstupními zařízeními.

6.1 System.IO.Ports

Jmenný prostor System.IO.Ports obsahuje třídy pro práci se sériovými porty. Nejdůležitější třída SerialPort poskytuje metody pro práci s asynchronními událostmi řízenými vstupními a výstupními zařízeními.

Systém.IO.Ports obsahuje také enumerické datové typy, které zjednodušují ovládání sériových portů. Jsou to Handshake, Parity, SerialPinChanges a StopBits.

Třída	Popis třídy
SerialEventArgs	Specifikuje argumenty zaslané události EventArgs.
SerialPinChangedEventArgs	Specifikuje argumenty zaslané události PinChangedEventArgs.
SerialPort	Reprezentuje zdroje sériového portu.
SerialReceivedEventArgs	Specifikuje argumenty zaslané události ReceivedEventArgs.

Tab. 6.1 Třídy prostoru System.IO.Ports

Delegát	Popis
SerialEventHandler	Zastupuje metodu, která ukazuje na událost EventArgs objektu SerialPort.
SerialPinChangedEventHandler	Zastupuje metodu, která ukazuje na událost PinChangedEventArgs objektu SerialPort.

Delegát	Popis
SerialReceivedEventHandler	Zastupuje metodu, která ukazuje na událost ReceivedEvent objektu SerialPort.

Tab. 6.2 Delegáti prostoru System.IO.Ports

Enumerický typ	Popis
Handshake	Specifikuje kontrolní protokol využívaný při navázání spojení při komunikaci přes sériový port pro objekt SerialPort.
Parity	Specifikuje paritní bit pro objekt SerialPort.
SerialErrors	Specifikuje chyby v komunikaci, které mohou nastat při použití objektu SerialPort.
SerialPinChanges	Specifikuje typ změny, která může nastat u objektu SerialPort.
SerialReceived	Specifikuje typ znaku, který byl přijatý na sériový port pro objekt SerialPort.
StopBits	Specifikuje stop bity použité v objektu Serial Port.

Tab. 6.3 Enumerické datové typy prostoru System.IO.Ports

6.1.1 Třída SerialPort

Třída SerialPort je určena pro řízení sériového portu COM. Poskytuje synchronní a událostmi řízený vstup a výstup pro sériový port a přístup k jeho vlastnostem. Krom toho může být funkčnost třídy zapouzdřena do vnitřních Stream objektů, přístupných skrze BaseStream vlastnosti.

Konstruktor	Popis
SerialPort()	Inicializuje novou instanci třídy SerialPort s implicitními hodnotami. Výchozí hodnota DataBits je 8, Parita je nastavena na hodnotu None a StopBit na 1.
SerialPort(IContainer)	Inicializuje novou instanci třídy SerialPort, využívá objekt IContainer. Kontejner může uchovávat další objekty.
SerialPort(String)	Inicializuje novou instanci třídy SerialPort. Z parametru typu String přebírá název portu, například COM2.
SerialPort(String, Int32)	Inicializuje novou instanci třídy SerialPort. Z parametrů přebírá název portu a Baud Rate.

Konstruktor	Popis
SerialPort(String, Int32, Parity)	Inicializuje novou instanci třídy SerialPort. Z parametrů přebírá název portu, Baud Rate a paritu.
SerialPort(String, Int32, Parity, Int32)	Inicializuje novou instanci třídy SerialPort. Z parametrů přebírá název portu, Baud Rate, paritní bit a data bity.
SerialPort (String, Int32, Parity, Int32, StopBits)	Inicializuje novou instanci třídy SerialPort. Z parametrů přebírá název portu, Baud Rate, paritní bit, data bity a stop bit.

Tab. 6.4 Konstruktory třídy SerialPort

Název metody	Popis
Close	Metoda uzavře spojení s portem, nastaví vlastnost IsOpen na false a zavolá metodu Dispose.
DiscardInBuffer	Metoda vyprázdní vstupní datový buffer sériového portu.
DiscardOutBuffer	Metoda vyprázdní výstupní datový buffer sériového portu.
Dispose	Metoda uvolní zdroje používané komponentou.
Open	Metoda otevře nové připojení na sériový port.
Read	Metoda čte data do pole bytů nebo znaků ze vstupního bufferu sériového portu.
ReadByte	Metoda provádí synchronní čtení jednoho bytu ze vstupního bufferu sériového portu.
ReadChar	Metoda provádí synchronní čtení jednoho znaku ze vstupního bufferu sériového portu.
ReadExisting	Metoda čte všechny okamžitě dostupné znaky ze vstupního bufferu sériového portu.
ReadLine	Metoda čte ze vstupního bufferu sériového portu, dokud nenarazí na znak nové řádky.
ReadTo	Metoda čte ze vstupního bufferu sériového portu dokud nenarazí na hodnotu specifikovanou v parametru.
Write	Metoda zapíše řetězec, pole bytů nebo pole znaků do výstupního bufferu sériového portu.
WriteLine	Metoda zapíše řetězec na konci se znakem nové řádky do výstupního bufferu sériového portu.

Tab. 6.5 Vybrané metody třídy SerialPort

Název vlastnosti	Popis
BaudRate	Vlastnost vrací nebo nastavuje baud rate.
BreakState	Vlastnost vrací nebo nastavuje signál stavu přerušení.
BytesToRead	Vlastnost vrací počet datových bytů ve vstupním bufferu.
BytesToWrite	Vlastnost vrací počet datových bytů ve výstupním bufferu.
DataBits	Vlastnost vrací nebo nastavuje standardní délku data bitu na byte.
DsrHolding	Vlastnost vrací stav DSR (Data Set Ready) signálu.
DtrEnable	Vlastnost vrací nebo nastavuje povolení DTR (Data Terminal Ready) signálu při sériové komunikaci.
Handshake	Vlastnost vrací nebo nastavuje protokol pro přenos dat přes sériový port.
IsOpen	Vlastnost vrací status objektu SerialPort, zda je otevřený nebo není.
NewLine	Vlastnost vrací nebo nastavuje hodnotu používanou pro ukončení řádku, jež využívají metody ReadLine a WriteLine.
Parity	Vlastnost vrací nebo nastavuje paritu.
PortName	Vrací nebo nastavuje jméno portu pro sériovou komunikaci.
ReadBufferSize	Vlastnost vrací nebo nastavuje velikost vstupního bufferu objektu SerialPort.
ReadTimeout	Vlastnost nastavuje nebo vrací počet milisekund, jak dlouho má operace read čekat na data.
RtsEnable	Vlastnost vrací nebo nastavuje povolení RTS (Request to Transmit) signálu během sériové komunikace.
StopBits	Vlastnost vrací nebo nastavuje počet stop bitů na byte.
WriteBufferSize	Vlastnost vrací nebo nastavuje velikost výstupního bufferu sériového portu.
WriteTimeout	Vlastnost vrací nebo nastavuje počet milisekund, jak dlouho má operace write čekat než se zapíše data.

Tab. 6.6 Vybrané vlastnosti třídy SerialPort

6.2 Použití třídy SerialPort

6.2.1 Vytvoření objektu SerialPort

Při vytváření nového objektu třídy SerialPort můžeme využít konstruktor s implicitními hodnotami nebo některý z dalších šesti, kde lze přímo nadefinovat některé hodnoty jako například jméno portu, paritu, stop bit, baud rate nebo data bity. Výchozí jméno portu je COM1, stop bit 1 a počet data bitů je 8.

6.2.1.1 Vytvoření objektu SerialPort s implicitními hodnotami

```
// Vytvoření objektu SerialPort  
  
SerialPort sp = New SerialPort();
```

6.2.1.2 Vytvoření objektu SerialPort s definovanou hodnotou

```
// Vytvoření objektu SerialPort pro COM2  
  
String portName = "COM2";  
  
SerialPort spCOM2 = New SerialPort(portName);
```

6.2.1.2 Vytvoření objektu SerialPort s definovanými hodnotami

```
// Vytvoření objektu SerialPort pro COM3  
// s baud rate 19 200 a paritou "Space"  
  
static String portName = "COM3";  
static Parity parity = Parity.Space;  
static Int32 baudRate = 19200;  
  
SerialPort sp = new  
    SerialPort(portName, baudRate, parity);
```

6.2.2 Odesílání dat na COM port

Po otevření portu pomocí metody Open můžeme poslat data na COM přes metodu Write s parametrem typu řetězec, pole znaků nebo pole bytů.

Druhou možností je použití metody `WriteLine`, která pošle zadaný řetězec spolu se znakem ukončení řádku.

```
// Otevření portu  
sp.Open();  
  
// Odeslání textu na port  
String text = "Ahoj";  
sp.WriteLine(text);  
text = "";  
  
// Uzavření portu  
sp.Close();
```

6.2.2 Čtení dat z COM portu

Po otevření portu pomocí metody `Open` a nastavení `read time out` můžeme data z COM portu přijímat pomocí různých metod. Metoda `Read` čte data do pole bytů nebo znaků. Metody `ReadByte` a `ReadChar` provádí synchronní čtení jednoho bytu nebo znaku z portu. `ReadLine` čte data ze sériového portu do té doby, než narazí na znak nové řádky. Pro čtení dat, dokud se nenarazí na určitou, předem specifikovanou hodnotu, se používá metoda `ReadTo`.

```
// Otevření portu  
sp.Open();  
  
// Nastavení read time out na 500 ms  
sp.ReadTimeout = 500;
```

```
// Přečtení dat ze sériového portu a vypsání do  
// Message Boxu
```

```
String text = "";  
text = sp.ReadLine();  
MessageBox.Show(text);
```

```
// Uzavření portu;
```

```
sp.Close();
```

Závěr

Sériový port (neboli rozhraní RS-232) je jednoduchá univerzální sběrnice určená pro komunikaci mezi dvěma zařízeními. Mohlo by se zdát tak trochu zpátečnické, v dnešní době, která je poznamenána bouřlivým rozmachem USB, se zabývat sériovým portem. Ale sériový port ještě dlouhou dobu zůstane používaným standardem zejména pro svoji rozšířenost a jednoduchost.

Mnohá zařízení, která lze připojit k počítači, jako jsou mobilní telefony a PDA využívají stále sériový port, případně jiná, ať už drátová (USB), či bezdrátová (IrDA, Bluetooth) rozhraní. Sériová komunikace začíná nahrazovat paralelní komunikaci, nejen v oblasti tiskáren apod., ale v poslední době i uvnitř počítače pro připojení pevných disků přes rozhraní Serial ATA.

Příložená knihovna pro komunikaci přes sériový port zjednodušuje používání jeho rozhraní a umožňuje rychlejší a komfortnější práci s portem. Programátor se tak může plně soustředit na problémy vytvářené aplikace a nemusí se zabývat zdoluhavým a složitým používáním API funkcí.

Použitá literatura

- [1] **Gorgon, Peter W.:** *Sériová komunikace*. První vydání, Praha, GRADA Publishing, 1995. Autorizovaný překlad anglického vydání. ISBN 80-7169-131-3.
- [2] **Kačmář, Dalibor:** *Programujeme .NET aplikace ve Visual Studiu .NET*. Praha, Computer Press. ISBN 80-7226-569-5.
- [3] **Sharp, John a Jagger, Jon:** *Microsoft visual C# .NET, krok za krokem*. První vydání, Praha, Mobil Media a. s., 2002. Autorizovaný překlad anglického vydání. ISBN 80-86593-27-4.
- [4] **Kepřt, Aleš:** Horké novinky v jazyce C# 2.0, *Computerworld*, ročník XV/2004, číslo 42 – 43.

Použité internetové zdroje

- [1] **Microsoft:** *Microsoft Developer Network*, (20. listopadu 2004).
URL: <http://msdn.microsoft.com>
- [2] **Microsoft:** *Microsoft Developer Network, Visual Studio 2005 Library*, (21. prosince 2004).
URL: <http://msdn2.microsoft.com/library/default.aspx>
- [3] **Christopher E. Strangio:** *The RS232 Standard*, (27. prosince 2004).
URL:
http://www.camiresearch.com/Data_Com_Basics/RS232_standard.html
- [4] **HW server:** *HW server představuje RS-232*, (27. prosince 2004).
URL: <http://rs232.hw.cz/>
- [5] **The Code Project**, (27. prosince 2004).
URL: <http://www.codeproject.com/>
- [6] **www.automa.cz:** *Bezdrátové komunikace v automatizační praxi II: standard Bluetooth*, (7. ledna 2005).
URL: <http://www.automa.cz/automa/2003/au070338.htm>
- [7] **HW server:** *HW server představuje LPT (IEEE 1284)*, (21. ledna 2005).
URL: <http://lpt.hw.cz>
- [8] **InetDaemon.Com:** *RS232, EIA232, EIA/TIA232*, (7. března 2005).
URL: <http://www.inetdaemon.com/tutorials/wan/serial/eia/eia232.html>
- [9] **Craig Peacock:** *Interfacing the Serial / RS-232 Port*, (7. března 2005).
URL: <http://www.beyondlogic.org/serial/serial1.htm>
- [10] **Maxim:** *Fundamentals of RS-232 Serial Communications*, (7. března 2005).
URL: http://www.maxim-ic.com/appnotes.cfm/appnote_number/527

Přílohy

Přílohou této bakalářské práce je kompaktní disk se zdrojovými kódy i zkompilovanou knihovnou pro sériovou komunikaci, dokumentací ke knihovně a ukázkovou aplikací pro použití této knihovny.