UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Uživatelské rozhraní sériového portu v Excelu

Jan Veselý

Bakalářská práce
2011

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2010/2011**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jan VESELÝ**
Osobní číslo:	**I07829**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Uživatelské rozhraní sériového portu v Excelu**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cíl: Cílem je vytvořit funkce a uživatelské rozhraní pro ovládání sériového portu z prostředí MS Excel.
Obsah teoretické části: Sériová komunikace a sériový port počítače - popis hardwarového a softwarového řešení.
Obsah implementační části: Vytvoření aplikace v prostředí Microsoft Office Visual Basic, která umožní plně ovládat sériový port počítače - nastavovat a číst stavové signály a posílat a přijímat data. Jednoduchá aplikace se zařízením komunikujícím přes sériový port.

Podmínkou je vlastní počítač se sériovým portem nebo USB-RS232 převodníkem.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

***dokumentace k Windows a Microsoft Office Visual Basic**

***B. Kainka, Měření, řízení a regulace pomocí PC. BEN - technická literatura, Praha 2003**

***D. Matoušek, Udělejte si z PC - generátor, čítač, převodník, programátor. BEN - technická literatura, Praha 2001**

Vedoucí bakalářské práce: **Ing. Daniel Honc, Ph.D.**
Katedra řízení procesů

Datum zadání bakalářské práce: **17. prosince 2010**

Termín odevzdání bakalářské práce: **13. května 2011**

Univerzita Pardubice
Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2011

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

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

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 4. 5. 2011 Jan Veselý

Poděkování

Závěrem bych chtěl poděkovat vedoucímu mé bakalářské práce, panu Ing. Danielu Honcovi, Ph.D., za cenné rady, důležité připomínky a zapůjčení měřicího přístroje a digitálního teplotního čidla, což mi usnadnilo vypracování bakalářské práce.

Anotace

Práce je zaměřena na vytvoření funkcí a uživatelského rozhraní pro ovládání sériového portu z prostředí MS Excel. Ovládání portu je řešeno pomocí API (Application Programming Interface). Ukázkové aplikace jsou vytvořeny pomocí VBA (Visual Basic for Applications). První umožňuje plně ovládat sériový port počítače, další dvě jsou určeny přímo pro komunikaci PC s teplotním čidlem a měřicím přístrojem.

Klíčová slova

sériový port, DLL, API, VBA, MS Excel

Title

User interface of serial port in Excel

Annotation

The work is aimed to creation of functions and user interface for serial port control from MS Excel environment. The control of the port is solved with help of API (Application Programming Inteface). Illustrative applications are developped with the assistance of VBA (Visual Basic for Applications). The first one allows us to control fully the serial port of the computer, the next two are determined directly for the communication of PC with temperature sensor and the multimeter.

Keywords

serial port, DLL, API, VBA, MS Excel

Obsah

Seznam zkratek

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
DCB	Device Control Block
DLL	Dynamik link library
VBA	Visual Basic for Applications
PC	Personal Computer
OS	Operating System

Seznam obrázků

Seznam tabulek

1 Úvodní informace

Bakalářská práce se zabývá tvorbou uživatelského rozhraní pro ovládání sériového portu z prostředí MS Excel. V poslední době ustupuje sériový port z oblasti počítačů do pozadí, protože byl nahrazen novým univerzálnějším a rychlejším rozhraním USB. Sériový port RS-232 i jeho modifikace se přesto ještě budou několik let dále používat, a to hlavně v průmyslu, v souvislosti s technikou řízení, regulace a měření.

Většina aplikací, které pracují se sériovým portem, využívá knihovnu RSCOM.DLL. V této knihovně se nacházejí hotové funkce pro ovládání portu. Tzn., že pokud budeme chtít používat aplikaci, musíme mít v PC uloženou i danou knihovnu. V případě, že chceme vytvořit a následně používat aplikaci ve VBA v Excelu bez použití knihovny RSCOM.DLL, musíme aplikaci naprogramovat na úrovni API. Funkce API se nacházejí ve standardních knihovnách OS. Použitím funkcí API docílíme, že naše aplikace bude funkční v PC s nainstalovanými MS Windows s MS Excel bez nutnosti instalace dodatečné knihovny.

V teoretické části bakalářské práce je vysvětlen princip sériové komunikace, popsán sériový port, řešení komunikace pomocí dynamicky linkované knihovny DLL a pomocí funkcí WIN32 API a je popsán programovací jazyk Visual Basic for Applications (VBA). V praktické části jsou uvedeny tři ukázkové aplikace - messenger, aplikace pro multimetr METEX a pro teplotní čidlo.

2 Teoretická část

2.1 Sériová komunikace

2.1.1 Vysvětlení pojmu

Princip sériové komunikace narozdíl od paralelní je založen na přenášení dat jen po jednom vodiči. Tzn. že jednotlivé bity se přenášejí postupně za sebou. Z toho vyplývají určité výhody i nevýhody. Při paralelním přenosu můžeme dosahovat vyšších rychlostí, ale za vyšší cenu kabelů. Pokud ale potřebujeme přenos dat na větší vzdálenost, je paralelní přenos nevýhodný, protože během přenosu dochází k velkému rušení. Je vhodnější použít sériový přenos, který nám umožní snížení přenosové rychlosti a tím je zajištěna dostatečná odolnost proti rušení.

Na vysílací straně je přenos jednotlivých bitů realizován pomocí nastavování logických úrovní „0" nebo „1" (to je dáno určitým rozsahem velikosti napětí). V komunikujících zařízeních, která mají nastavenou stejnou přenosovou rychlost, musí proběhnout synchronizace, aby přijímač začal přijímat ve správnou dobu. Aby dokázal přijímač jednotlivé bity správně poskládat i v případech, kdy nedochází ke střídání úrovní, jsou v praxi používány dvě metody - synchronní a asynchronní.

[1]

2.1.2 Metoda synchronní

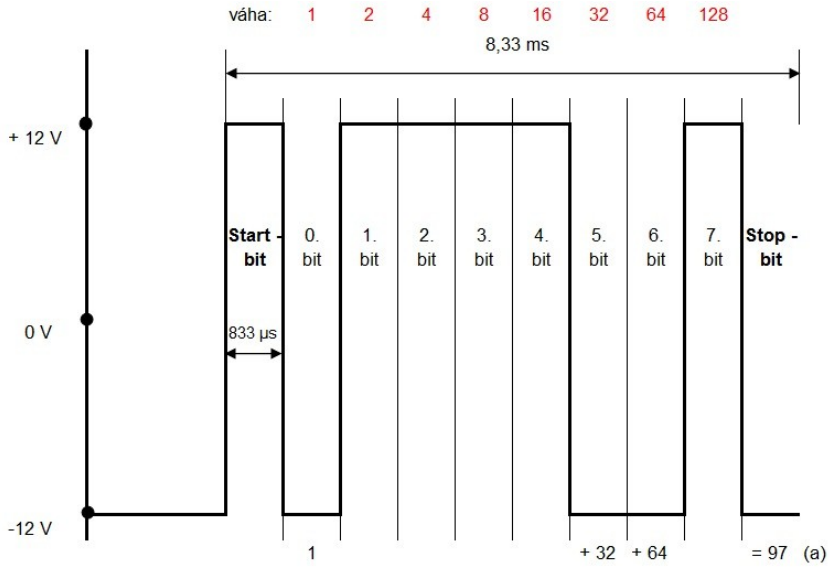
Abychom mohli využívat synchronního přenosu, musíme použít kromě datového ještě jeden vodič. Po novém vodiči je vysílán signál hodin, který nám zajistí synchronizaci vysílače a přijímače. Pro každý datový bit je vyslán impulz označující platnost datového bitu. Díky impulzu přijímač pozná, že má bit přijmout.

[1]

2.1.3 Metoda asynchronní

Při asynchronním přenosu není třeba dalšího vodiče, po kterém by se vysílal hodinový signál. Jelikož se data vysílají po blocích (po související skupině bitů), musí být při vysílání vždy označen začátek a konec bloku (v RS-232) tzv. startbitem a stopbitem. Při každém přenosu, než se začne vysílat první datový bit z bloku, musí být vyslán nejdříve startbit. Startbit pozná přijímač dle změny úrovní. Po úspěšném dokončení přenosu bude vyslán stopbit, jejž lze v pauzách mezi přenosy libovolně prodlužovat. Jelikož je přesně definován časový rastr bloku (viz následující obrázek 1), který určuje množství přiděleného času pro každý bit z bloku, přijímač po přečtení startbitu začne vybírat jednotlivé bity dle rastru.

[1]



Obrázek 1 – Sériově vyslaný znak „a" (ASCII 97) s přenos. rychl. 1200 baudů

2.2 Sériový port RS-232

Jako komunikační rozhraní osobních počítačů a jiné elektroniky se používá sériový port RS-232. Je určen k propojování a vzájemné sériové komunikaci dvou zařízení. V roce 1969 vyšla jeho poslední varianta, která navíc přibírá do názvu C, tzn. RS-232C. V oblasti osobních počítačů byl tento standard nahrazen modernějším a hlavně výkonnějším USB (univerzálním sériovým rozhraním). V oblasti měření, řízení a regulace je a bude standard RS-232 používán nadále. RS-232 jen definuje přenos určité sekvence bitů, tzn. jen na nejnižší úrovni a o ostatní úrovně se nestará. Rozhraní USB se narozdíl od RS-232 vyššími vrstvami zabývá. Na PC bývá tento port vyveden pomocí konektoru D-Sub typu DE-9 M (Male). Na propojovacím kabelu je stejný konektor opačného typu DE-9 F (Female). V případě, že se v našem počítači port COM nenachází, můžeme si zakoupit rozšiřující kartu (řadič), popř. převodník z USB/RS-232, který má obvykle delší odezvu než RS-232 bez převodníku. To způsobuje někdy v některých aplikacích problémy s fungováním.

[2]

2.2.1 Technický popis portu RS-232

Napájecí napětí je typické pro RS-232 -12 V a +12 V. To má výhody i u dlouhých kabelů a vyšších přenosových rychlostí, kdy je dosaženo větší odolnosti proti rušení. Pro přenosovou rychlost 19,2 kBd je možné použít i kabel dlouhý 15m. Napěťové úrovně dle normy rozlišují logické úrovně, kde logická jednička odpovídá napětí -12 V a logická nula +12 V. RS-232 určuje pro přenos dat asynchronní sériovou komunikaci. Přenos je proveden od nejméně významného bitu po nejvýznamnější. Sériový port má dva druhy konektorů, a to buď 9 pólový či 25 pólový. Pro základní komunikaci nám stačí vysílací linka TXD, zemnicí linka GND, v případě obousměrné komunikace potřebujeme ještě přijímací linku RXD. Kromě nich lze používat ještě dalších 6 řídicích linek (viz. tabulka).

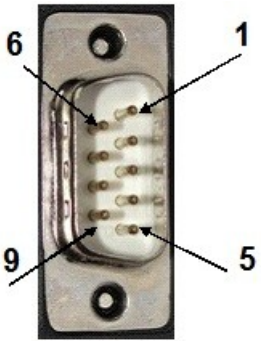
Některé z nich jsou vstupní, jiné výstupní. V případě, že nevyužíváme některé řídicí linky, lze nevyužité vývody použít např. pro napájení elektronických zařízení.

[1]

Tabulka 1 – Řídicí signály s čísly vývodů

			Označení vývodů	
Řídicí signál	**Směr**	**Funkce**	**9 vývodů**	**25 vývodů**
TXD (Transmit Data)	výstup	vysílaná data	3	2
RXD (Receive Data)	vstup	přichozí data	2	3
RTS (Request To Send)	výstup	zapnout vysílací zařízení	7	4
CTS (Clear To Send)	vstup	připravenost k vysílání	8	5
DSR (Data Set Ready)	vstup	připravenost k provozu	6	6
GND (Ground)	-	pracovní zem	5	7
DCD (Data Carrier Detect)	vstup	úroveň přijímaného signálu	1	8
DTR (Data Terminal Ready)	výstup	koncové zařízení připraveno	4	20
RI (Ring Indicator)	vstup	zvonek (příchozí volání)	9	22

[1]



Obrázek 2 – Označení vývodů RS-232 u 9 pinového konektoru

2.2.2 Propojovací kabel pro RS-232

Kabel může být zapojen různě:

a) 1:1, tzn. že na obou koncích propojuje stejné kolíky

b) překřížené zapojení (lze použít např. k propojení dvou PC)

c) univerzální zapojení

[1]

2.2.3 Parametry portu pro nastavení:

- přenosová rychlost – 50 - 38400 baudů
- počet datových bitů – nejčastěji 7 nebo 8 bitů, některé přístroje mohou pracovat s 5 nebo 6 bity (pro zachování kompatibility s dálnopisem)

- počet stopbitů – lze volit 1 nebo 2, v případě použití paritního bitu v kombinaci s 8 datovými bity je povolen pouze 1 stopbit

- paritní bit – přenos datových bitů může být zabezpečen pomocí paritního bitu, který může nabývat hodnoty sudé, liché, konstantní space či mark (nechceme-li přenos zabezpečit, není použita žádná hodnota), paritní bit se nachází za posledním datovým bitem

[1]

2.2.4 FlowControl (řízení toku)

Jedná se o mechanismus, který dokáže komunikaci pozastavit, dokud je některé zařízení vytíženo, nebo nemůže pracovat z jiných důvodů. Existují dva typy řízení – hardwarové a softwarové. Hardwarové je řešeno použitím řídicích signálů a nedochází ke snížení rychlosti. Softwarové probíhá jen po datovém vodiči s vysíláním dvou speciálních znaků XON a XOFF. Tím dochází ke snížení rychlosti.

[3]

2.3 Přehled vybraných typů sběrnic

Aby bylo možné si představit, jak hodně se odlišuje přenosová rychlost RS-232 od jiných počítačových sběrnic, je uveden přehled přenosových rychlostí v následující tabulce.

Tabulka 2 – Porovnání přenosů dle typů sběrnice

Název	Typ sběrnice	Max. teoretická rychlost
RS-232	sériová	Až 19200 Bd
LPT	paralelní	Až 12000 Kb/s
USB 1.1	sériová	12 Mb/s
USB 2.0	sériová	480 Mb/s
USB 3.0	sériová	5 Gbit/s
FireWire 400	sériová	400 Mb/s
FireWire 800	sériová	800 Mb/s
PATA	paralelní	133 MB/s
SATA	sériová	1.5 Gb/s (150 MB/s)
SATA II	sériová	3 Gb/s
SATA III	sériová	6 Gb/s
PCI	paralelní	132 – 533 MB/s
PCI-Express 2.0	sériová	8 GB/s (32x)

2.4 DLL (Dynamic-link library, dynamicky linkovaná knihovna)

DLL je označení pro knihovnu, která obsahuje kód a data. Tyto se mohou používat najednou ve více programech. Tzn. v jakémkoliv našem programu lze použít funkce z knihoven DLL.

Výhodou toho je, že můžeme kdykoli opakovaně použít hotový kód a neztrácet pokaždé čas s novým vytvářením funkce a tím nenastane duplikace kódu a dochází tak k efektivnímu využívání paměti. To ovlivní výkon programů běžících v popředí i dalších programů probíhajících ve Windows. Knihovny DLL jsou založeny na modulární architektuře Promotes, která nám dále umožní vyvíjet složitější programy skládající se z několika modulů. Jednotlivé moduly lze při spuštění hlavního programu dynamicky načítat. Další výhodou je, že pro opravy chyb v DLL stačí DLL aktualizovat a tím nemusíme zasahovat do veškerých programů jednotlivě.

[4]

2.4.1 Soubory knihoven DLL implementovaných v OS Windows

- Soubory prvky ActiveX (OCX)
- Soubory ovládacích panelů (CPL)
- Soubory ovladače (drv) zařízení

[4]

2.4.2 Řešení komunikace pomocí DLL (ve VBA)

Abychom mohli v našich aplikacích využívat funkce z DLL knihoven, musíme na začátku modulu deklarovat funkce, které chceme z knihovny používat. V případě, že tedy chceme používat knihovnu RSCOM.DLL, bude např. deklarace funkce pro otevření portu vypadat následovně:

```
Declare Function OPENCOM Lib "RSCOM" (ByVal OpenString$) As
Integer
```

pro zavření portu:

```
Declare Sub CLOSECOM Lib "RSCOM" ()
```

Deklarace vždy začíná slovem Declare, pak následují typ funkce (Function/Sub), název funkce (použité v DLL) a slovo (Lib), které nám určuje, že funkce chceme načítat z knihovny. Do uvozovek uvedeme název knihovny s příponou (.dll) či bez přípony, do závorek nadeklarujeme parametry, které se dané funkci mají předávat, a v případě funkce (Function) nadeklarujeme typ návratové hodnoty.

Po deklaraci funkcí z DLL použijeme funkce tak, že je zavoláme s příslušnými parametry. Pro použití portu COM3 s přenosovou rychlostí 9600 baudů, 7 datovými bity,

1 stopbitem a bez použití parity zavoláme funkci OPENCOM a nemusíme se o nic dál starat. V případě, že nenastane během připojování portu nějaká chyba, dojde k otevření portu a můžeme začít rovnou komunikovat.

```
OPENCOM (COM3:9600,N,7,1)
```

2.5 Win32 API

Win32 API (Application Programming Interface) je programovací rozhraní vyvinuté Microsoftem, používá se v OS Windows pro přímé volání funkcí Windows. Operační systém Windows obsahuje velké množství knihoven DLL. Velmi důležitou knihovnou je KERNEL32.DLL. API zpřístupňuje datové typy, struktury a funkce operačního systému. Funkce z DLL mohou být vyvolány jakýmkoliv programem. API obsahuje obrovské množství funkcí (uvádějí se tisíce) a čím více použijeme knihoven DLL, tím se množství funkcí navyšuje. Tyto funkce mohou být různě rozděleny, např.: funkce pro síť, zvuk, grafiku aj. Počátky API sahají do Windows verze 1.0, tehdy obsahovalo přibližně 450 funkcí. Od té doby se neustále rozrůstá.

[5]

2.6 Řešení sériové komunikace pomocí Win32 API

Prostřednictvím pouze několika volání API orientujících se na správu dat probíhají přístupy na sériový port. Pokud chceme pracovat se sériovým portem, pracuje se s ním jako se souborem. Tzn., že pro otevření portu použijeme funkci *CreateFile*, která se běžně používá pro otevření souboru či zařízení. Pro zavření portu nebo zařízení použijeme *CloseHandle*. Potřebujeme-li odeslat data na port či provést zápis, využijeme *WriteFile*, pro čtení a příjem dat *ReadFile*. Obdobným způsobem se pracuje s porty USB aj.

Sériový port může používat dva režimy přenosu – overlapped (překrývaný) a nonoverlapped (nepřekrývaný).

- ***Nonoverlapped (nepřekrývaný)***

Tento režim je jednodušší, ale dokud není dokončena operace read či write, nemůže proces (thread) pracovat dále, na programy nesouvisející s naší aplikací nemá pozastavení vliv. Tzn. např., že dokud nedokončí přijímač přijímání dat, nemůže vysílač data odesílat. Za předpokladu, že přijímači dorazí méně bajtů, než přijímač čekal, nastane problém – program zůstane viset. Aby se tomuto případu zamezilo, je nutné nastavit timeout.

- ***Overlapped (překrývaný)***

Tento režim je více efektivní, protože může přijímat a vysílat data současně, ale je oproti předchozímu mnohem složitější. Je vhodnější použít ho při velkých přenosech dat. Proces (thread) běží i za předpokladu, že přenos není dokončen. Čas, ve kterém přenos probíhá, může být využit i pro jiné procesy – to je výhodou oproti nepřekrývanému režimu.

To ale navíc vyžaduje synchronizaci formou zpráv a stavu akce. Abychom zjistili, zda přenos úspěšně proběhl, použijeme funkci *GetOverlappedResult*.

[1]

2.6.1 Otevření portu

Při volání funkce *CreateFile* musíme zadat její parametry, které specifikují, o jaký přenos se bude jednat. V případě úspěšného otevření portu vrací funkce handle na port, v opačném případě je již port obsazen nebo neexistuje a vrací hodnotu -1 (INVALID_HANDLE_VALUE). V následující ukázce je příklad funkce *CreateFile* s nastavením režimu overlapped.

```
HANDLE hComm;
hComm = CreateFile( gszPort,
      GENERIC_READ | GENERIC_WRITE,
      0,
      0,
      OPEN_EXISTING,
      FILE_FLAG_OVERLAPPED,
      0);

if (hComm == INVALID_HANDLE_VALUE)
//Port je již otevřen, špatně nastavené parametry, nebo
neexistuje, proto se ho nepodařilo otevřít.
```

[3]

Do prvního parametru se zadává název portu, který chceme otevřít. Např. COM2. Následující položka nám určuje přístup, zda chceme z portu číst, zapisovat nebo obojí. Čtení a současně zápis nastavíme tak, že použijeme konstanty s logickým součtem (OR) – GENERIC_READ | GENERIC_WRITE. Další nastavení se týká módu sdílení, pro COM porty se zadává vždy 0, neboť nemohou být sdíleny jako soubory s ostatními aplikacemi. Čtvrtý parametr se týká ukazatele na strukturu zabezpečovacích atributů, taktéž nastavíme 0, protože dané atributy nejsou pro sériové porty podporovány. Jako další se pro COM nastavuje vždy OPEN_EXISTING, tento parametr rozlišuje, zda soubory mají být jen otevřeny, nebo znovu založeny. Předposlední, co musíme nastavit, je, zda se bude jednat o režim overlapped (nastavíme konstantou FILE_FLAG_OVERLAPPED) či režim nonoverlapped (nastavíme nulou). Poslední parametr předává handle na soubor s rozšířenými atributy, pro COM vždy 0.

Po otevření portu se dají nastavovat další parametry přenosu. V případě, že máme již port otevřený, měli bychom specifikovat přenosovou rychlost, počet datových bitů, počet stopbitů, paritu aj. Jak nastavit některé parametry je uvedeno v následujících kapitolách.

2.6.2 DCB struktura

DCB (Device-Control Block) – kontrolní blok zařízení. Pomocí této struktury nastavujeme mnoho vlastností sériového portu. Pokaždé, když otevíráme port, je struktura

DCB naplněna nastavením, které bylo předtím použito jako poslední. Poslední nastavení nám ale většinou nevyhovuje, proto je nutné pokaždé po připojení portu nastavit ve struktuře parametry, které potřebujeme pro naši komunikaci. V případě, že nám komunikace z nějakých důvodů nefunguje, příčinou bývá špatná inicializace či nastavení DCB. Pro získání parametrů z aktuální struktury slouží funkce *GetCommState*, kterou je dobré použít v případě, že potřebujeme změnit jen několik parametrů. Pro nastavení nové struktury slouží pak dvě funkce – *BuildCommDCB* či *SetCommState*.

[1]

```
DCB dcb;
FillMemory(&dcb, sizeof(dcb), 0);
if (!GetCommState(hComm, &dcb))  //získá aktuálně použitou
DCB do struktury dcb
     //chyba při získávání použité DCB
return FALSE;

//aktualizace přenosové rychlosti DCB
dcb.BaudRate = CBR_9600;

//změna nastavení DCB
if (!SetCommState(hComm, &dcb))
//chyba při nastavování DCB
//jedná se o chybu handlu portu, nebo chybné nastavení DCB
```

[3]

Existuje velké množství možností pro nastavení DCB. V tabulce uvádím pouze možnosti nejčastěji používané.

Tabulka 3 – Výběr z možností pro nastavení DCB

Vlastnost	**Popis**
DCBlength	délka struktury DCB (v bajtech)
BaudRate	nastavení přenosové rychlosti např.: 300, 9600, apod. (v baudech)
XonLim	udává počet bajtů v přijímacím bufferu, při kterém se vysílá XON
XoffLim	udává počet bajtů v přijímacím bufferu, při kterém se vysílá XOFF
ByteSize	počet datových bitů (délka znaku) – nejčastěji 7, 8
Parity	paritní bit (no, odd, even, mark, space)
StopBits	počet stopbitů (1, 1.5, 2)
XonChar	použitý znak XON
XoffChar	použitý znak XOFF
ErrorChar	náhradní znak při chybě
EofChar	znak konce souboru
EvtChar	při příchodu zde nastaveného znaku je vyvolána událost (event)

2.6.3 Timeouty

Při přijímání nonoverlapped operací může nastat stav, kdy očekáváme určitý počet bajtů a ne všechny dorazí z důvodu nějaké poruchy při komunikaci. V tomto případě by vlákno čekalo nekonečně dlouho na bajty, které už ani nemusí dorazit a tím dojde k uvíznutí programu. Abychom předešli zastavení komunikace, je nutné nastavit časy (timeout) hlavně pro přijímač, případně pro vysílač. Pro zápis je timeout důležitý v případě, že máme plný výstupní buffer a potřebujeme přenést dlouhý řetězec. Pro nastavení timeoutů existuje také struktura – COMMTIMEOUTS. Pro zjištění aktuálně používaných timeoutů se používá funkce *GetCommTimeouts*, pro nové nastavení *SetCommTimeouts*.

Struktura obsahuje 5 možností pro nastavení časů:

ReadIntervalTimeout – tímto nastavíme max. čas v ms, který bude port čekat mezi dvěma přijímanými bajty po celou dobu trvání funkce *ReadFile*. Časová perioda začne běžet až po přijetí prvního bajtu. Při překročení nastaveného času je čtení ukončeno a v případě, že jsou ve vstupním bufferu nějaké přijaté bajty, vrátí je. Pokud tento timeout nechceme používat, nastavíme hodnotu na 0.

ReadTotalTimeoutMultiplier – používá se pro výpočet celkové doby trvání funkce *ReadFile*. Celková doba se vypočítá ze zadané hodnoty v ms vynásobené počtem bajtů, které chceme přijmout. Při nastavení 0 nebude timeout použit.

ReadTotalTimeoutConstant – používá se také k výpočtu celkové doby trvání funkce *ReadFile* a to tak, že nastavená hodnota se poté přičte k výsledku násobení ReadTotalTimeoutMultiplier a počtu bajtů, které chceme načíst. Interval zakážeme opět hodnotou 0.

WriteTotalTimeoutMultiplier hodnota – je určena pro výpočet celkové doby trvání každého zápisu bajtů do bufferu. Při vynásobení zadané hodnoty v ms s počtem požadovaných bajtů k odeslání získáme celkovou dobu trvání.

WriteTotalTimeoutConstant – určena pro výpočet celkové doby trvání každého zápisu bajtů do bufferu. Celková doba se vypočítá přičtením zadané hodnoty v ms k výsledku násobení WriteTotalTimeoutMultiplier a počtu bajtů, které chceme zapsat.

[1]

2.6.4 Buffery

Sériový port využívá pod OS Windows jeden vstupní buffer a jeden výstupní buffer. Veškeré vysílání čí příjem se děje pomocí nich. Pokud tedy odesíláme nějaká data funkcí *WriteFile*, jsou nejdříve uložena do výstupního bufferu a následně z bufferu odvysílána. V případě příjmu dat je princip opačný – příchozí data jsou uložena do vstupního bufferu a z něho jsou vyjmuta funkcí *ReadFile*.

Velikosti obou bufferů lze po otevření portu měnit ve velkém rozsahu, ale ne vždy platí, že čím větší velikost bufferu nastavíme, je to pro nás výhodou. V případě pomalého

zpracování po jednom bajtu se bude nacházet velké množství starých dat. Nastavení nových velikostí obou bufferů se provede současně pomocí funkce *SetupComm*. Vyprázdnění bufferu se provádí funkcí *PurgeComm*, stačí uvést handle portu a typ bufferu. Pokud potřebujeme zjistit, kolik bajtů se nachází v každém bufferu, stačí zavolat funkci *ClearCommError*.

[1]

2.6.5 Události

V případě, že potřebujeme provádět nějaké operace jen za určitých okolností, je možné nastavit pomocí masky události, které nás budou o změně stavu informovat. K nastavení událostí se používá funkce *SetupCommMask*. V momentě, kdy nastane námi požadovaná změna stavu, je vyvolána událost, díky které zjistíme, že došlo ke změně stavu, a můžeme provést požadované operace. Např.: chceme-li, aby došlo ke spuštění funkce Read jen v momentě, když přijde do bufferu bajt, musíme funkcí *SetupCommMask* nastavit událost, jež nás bude informovat o příchodu bajtu. Když potřebujeme zjistit, jaké události jsou povoleny a jaké zakázány, použijeme funkci *GetCommMask*. Pokud nechceme využívat události, nastavíme do masky hodnotu 0. Zda událost nastala, zjišťujeme např. funkcí *WaitCommEvent*.

[3]

Události mohou být různé, zmíním pouze některé z nich s náhodnými příklady použití:

a) Na vstupní rozhraní přišel bajt. Po příchodu jednoho bajtu je vyvolána událost, díky které se nám spustí čtení. Tuto událost lze použít při malém objemu dat v případě, že není dohodnutý bajt označující konec zprávy nebo počet bajtů ve zprávě.

b) Přišel námi definovaný znak, např. Enter CR (Carriage Return), který má v ASCII kódu hodnotu 13. Při příchodu takového znaku nastane událost, po níž se může v případě naší potřeby spustit čtení. Tento typ lze např. použít u teplotního čidla.

c) Objeví-li se v bufferu námi definovaný určitý počet bajtů, nastane opět událost. Musíme vyčkat, až dorazí všechny bajty, s neúplnými daty nemůžeme dále pracovat. Lze použít např. u zařízení METEX, které posílá hodnoty na 4 řádcích, tj. celkem 56 bajtů.

Většina nastavení – např. definování znaku, počet bajtů aj. se provede pomocí DCB.

2.6.6 Nastavení řídicích signálů

Funkce *EscapeCommFunction* nám umožní nahození či shození výstupních linek. Pro zjištění aktuálních stavů vstupních linek se používá funkce *GetCommModemStatus*.

2.6.7 Zápis na port (odesílání dat)

Pro zápis na sériový port je určena funkce *WriteFile*. Následující příklad ukazuje, jak se provede zápis při nonoverlapped operaci.

```
WriteFile(hComm, lpBuf, dwToWrite, &dwWritten, NULL)
```

[3]

První parametr je handle na portu, na který chceme zapisovat. Dále se určuje ukazatel na data, která se mají zapsat. Třetí proměnná se týká předešlých dat, určuje nám, kolik bajtů se má odeslat. Z předposlední části zjistíme, kolik bajtů bylo ve skutečnosti odesláno (při úspěšném přenosu se musí dwWritten rovnat s počtem bajtů v dwToWrite, který se měl zapsat). Do posledního parametru se zadává ukazatel na overlapped strukturu v případě otevření portu s režimem overlapped (FILE_FLAG_OVERLAPPED). V ostatních případech se zadává NULL.

2.6.8 Čtení z portu (přijímání dat)

Provádí se velmi podobným způsobem jako zápis. Funkce pro čtení se jmenuje *ReadFile*.

```
ReadFile(hComm, lpBuf, READ_BUF_SIZE, &dwRead, NULL)
```

[3]

Prvním parametrem je opět handle portu, ze kterého chceme číst data, další je ukazatel, kam se budou přijaté bajty ukládat. Třetí část (READ_BUF_SIZE) určuje, kolik bajtů se má číst, následující část (dwRead) značí, kolik bajtů bylo úspěšně přijato. Nakonec je NULL v případě použití nonoverlapped režimu, v opačném případě je to ukazatel na overlapped strukturu.

Pokud chceme například číst bajty, jen když nastane nějaká událost, provedeme to následujícím způsobem:

```
DWORD dwCommEvent;
DWORD dwRead;
char  chRead;

if (!SetCommMask(hComm, EV_RXCHAR))
//Nastala chyba při nastavování událostní masky
for ( ; ; ) {
     if (WaitCommEvent(hComm, &dwCommEvent, NULL)) {
     if (ReadFile(hComm, &chRead, 1, &dwRead, NULL))
     // bajt byl načten
     else
     //chyba při volání ReadFile
     break;
     }
     else
     // chyba při volání funkce WaitCommEvent.
     break;
}
```

[3]

2.6.9 Zavření portu

V okamžiku, kdy už námi otevřený port nepotřebujeme, je nutné ho uvolnit, aby ho mohly používat i jiné aplikace. Dokud není uvolněn aplikací, která ho otevřela, není ho možné použít v jiné aplikaci. Uvolnění provedeme pomocí funkce *CloseHandle*, do které vložíme za parametr handle portu (číselná adresa otevřeného portu).

Např.: `CloseHandle(hComm)`

[1]

2.7 VBA

VBA (Visual Basic for Applications) je programovacím jazykem, který vychází z klasického VB (Visual Basicu), je ochuzen o další možnosti, tzv. okleštěn . VBA lze obohacovat o další nestandardní funkce a možnosti pomocí API (vysvětleno viz výše). Zjednodušeně řečeno, API je programovým rozhraním, které nám umožní pracovat s různými funkcemi, jež jsou obsaženy v dynamických knihovnách DLL systému Windows.

VBA nenalezneme jen v několika aplikacích z balíku MS Office, jako je např. ve Wordu, Excelu, Powerpointu, Outlooku, ale také v aplikacích jiných výrobců, jako např. AutoCAD, Corel Draw aj. Vlastní rozšíření v balíku MS Office lze provádět nejen pomocí VBA, ale i pomocí VBScript (Visual Basic Script) a případně JavaScript. Spíše se zabývají vytvářením kódů pro HTML. Od verze Excelu 97 je základ VBA pro všechny následující verze stejný, tím je dosaženo zpětné kompatibility.

[6]

Abychom mohli používat VBA v různých aplikacích, musíme pochopit jeho objektové modely. VBA totiž s objekty pouze manipuluje, takže např. Word, Excel, Access, PowerPoint aj. mají svůj vlastní objektový model. Ve struktuře objektového modelu funguje vztah nadřízenosti a podřízenosti modelů. S objekty se dá ve VBA pracovat. Aplikace se může programovat prostřednictvím objektů, jež vystavuje. Např. objektový model Excelu obsahuje navíc objekty pro rozbor dat. Používají se různé objekty např. grafy, listy, kontingenční tabulky a všelijaké funkce (matematické, ekonomické, finanční apod.). Předkem všech objektů je objekt Application. Pracujeme-li v Excelu, je objekt Application jedinečnou spuštěnou aplikací Excel, kde se nachází kód. Zpravidla se název objektu Application neuvádí.

VBA se v budoucnu bude ještě používat, jelikož z něho vyplývá obrovský počet řešení, která se dají snáze naučit než jeho alternativy. Je to jazyk, který pomáhá programátorům, jelikož dokáže při práci s daty všechny interní detaily automaticky spravovat. V některých programovacích jazycích musí totiž programátoři povinně datový typ pro každou proměnnou explicitně definovat.

[7]

2.7.1 Definice datových typů

Jak jsou uložena data v paměti (zda jako čísla reálná, celá, řetězce, aj.), nám řekne datový typ proměnné. V případě, že chceme o datových typech proměnných nechat rozhodnout VBA, je pro VBA výchozím datovým typem Variant (mění typ podle práce s proměnnou). Výhoda – při vytváření kódu šetří čas uživatele (nemusí deklarovat datový typ proměnné). Nevýhody: více spotřebované paměti než je nutné, pomalejší provádění příkazů a během kompilace kódu se špatně odhadují přídavné kontroly možných chyb.

[7]

2.7.2 Použití VBA:

Pokaždé nám nestačí používat standardní funkce Excelu, a proto jsme nuceni vytvořit si své vlastní funkce, které jsou šité na míru našim potřebám. VBA nám usnadní používat opakovaný sled stejných úkonů, dále chceme-li analyzovat data či provádíme-li uživatele v určitém procesu.

[6]

Chceme-li vytvářet či upravovat kód VBA, musíme použít editor jazyka VBE (Visual Basic Editor). Editor je samostatná aplikace, která je propojena např. s Excelem, tzn. že bez Excelu nemůžeme samostatně VBE spustit. Jako první musí naběhnout Excel a teprve z něho se spustí editor VBE.

[7]

2.7.3 Základní pojmy VBA

Kód (Code) – vytvářen posloupností instrukcí dle pravidel jazyka VBA a je uložen v modulu. Kód můžeme vytvářet ručně (pomocí klávesnice), kopírováním např. z jiného modulu anebo záznamem makra, který pro nás rychle vygeneruje kód, ale ne vždy splňuje všechny naše požadavky, k tomu ještě obsahuje řádky navíc, které zmenšují rychlost programu. Z toho vyplývá, že vygenerovaný kód je nutné dále upravit, popř dopsat ručně.

Modul (Module) – nachází se v sešitech Excelu, obsahuje programový kód (složen z procedur a deklarací). Existují dva typy – standardní moduly a moduly tříd. K editaci a úpravě se používá editor VBE.

Objekt (Object) – je základní nejmenší částí objektového modelu. Příkladem objektů mohou být – pracovní list, sešit, graf, buňka, tlačítko... Uspořádání objektů je dáno určitou hiearchií – při odkazu na členský objekt určíme jeho pozici tak, že za název kontejneru, ve kterém se nachází požadovaný objekt, uvedeme tečku a za ni název členu (tečka představuje oddělovač kontejneru od vlastního členu).

Formulář (UserForm) – lze ho považovat za určitý druh modulu (obsahuje programový kód související s ovládajícími prvky formuláře), vytváří uživatelské okno, jež slouží k jednoduchému dorozumívání s uživatelem. Pomocí něho získá program vstupy, od kterých se bude odvíjet výstup.

Procedura (Procedure) – taktéž nazývaná makro. Jedná se vlastně o část programového kódu, provádějícího některé činnosti. Vyvolat proceduru můžeme, kdykoli ji potřebujeme. VBA rozlišuje dva typy procedur – *Sub* a *Function*. Funkce má jednu návratovou hodnotu, popř. pole proměnných. *Sub* se skládá z řady příkazů a nemá žádnou návratovou hodnotu.

Projekt (Project) – nachází se v každém sešitě. Je tvořen moduly, objekty a formuláři patřícími do daného sešitu. Za projekt je považován každý otevřený sešit – v editoru vidíme tolik projektů, kolik máme spuštěných sešitů.

[6]

2.7.4 Přehled datových typů používaných ve VBA

Tabulka 4 – Datové typy VBA

Datový typ	Počet použitých bytů
Byte	1 bajt
Boolean	2 bajty
Integer	2 bajty
Long	4 bajty
Single	4 bajty
Double	8 bajtů
Currency	8 bajtů
Decimal	12 bajtů
Date	8 bajtů
Object	4 bajty
String (s prom. délkou)	10 bajtů + délka řetězce
String (s pevnou délkou)	délka řetězce
Variant (s čísly)	16 bajtů
Variant (se znaky)	22 bajtů + délka řetězce
Uživatelsky definovaný	různý

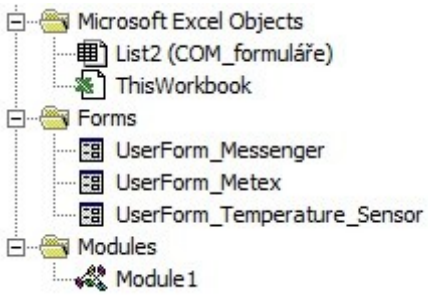
[7]

3 Praktická část

Znalosti a zkušenosti z teoretické části jsem použil pro vytvoření tří ukázkových aplikací pracujících se sériovým portem PC. První aplikací je terminál pro posílání textových zpráv mezi dvěma PC spojenými sériovým kabelem. Další dvě aplikace komunikují s měřicím přístrojem METEX a měřicím čidlem teploty. Jsou použity funkce OS Application Programming Interface a aplikace jsou vytvořeny ve Visual Basic for Applications.

3.1 Obsah objektů v projektu

Projekt se skládá z listu List2, ThisWorkbook, tří formulářů (aplikací) – UserForm_Messenger, UserForm_Metex, UserForm_Temperature_Sensor a jednoho modulu (Modul1).



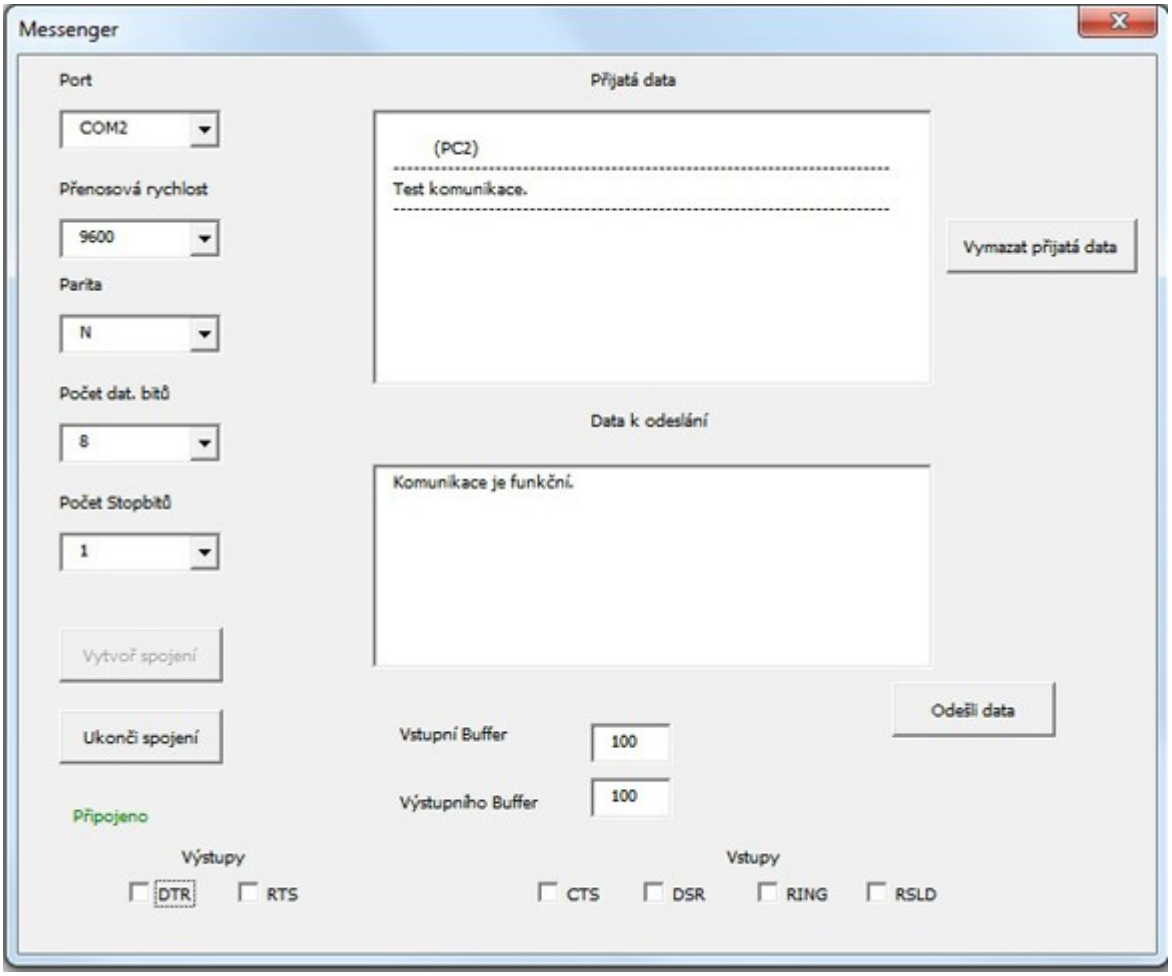
Obrázek 3 – Struktura projektu.

3.2 Aplikace č. 1 (UserForm_Messenger)

Tato aplikace umožňuje nastavit parametry přenosu jako je např. označení portu, přenosová rychlost, nastavení parity, počet datových bitů, počet stopbitů, nastavení velikostí vstupního a výstupního bufferu (od Windows NT a jeho vyšších verzí jsou podporovaná jen sudá čísla v bajtech, v mém případě od 10 do 4096 bajtů). Dále je možné posílat libovolné textové zprávy na druhé zařízení, nastavovat řídicí signály na výstupu a sledovat příchozí signály na vstupu.

Aby nám komunikace mezi dvěma zařízeními fungovala, musíme na obou zařízeních nastavit stejné přenosové parametry. Tyto parametry nemusíme psát, stačí u každého parametru rozkliknout `ComboBox` a vybrat si z příslušných nabídek. Dále pak klikneme na tlačítko `Vytvoř spojení`, které nám zajistí vytvoření komunikace. Po vytvoření komunikace je tlačítko `Vytvoř spojení` zakázáno do doby, dokud není komunikace ukončena. V případě, že došlo k úspěšnému připojení, je již povoleno tlačítko `Odešli data`, které odešle napsaný text z `TextBoxu` (s nadpisem `Data k odeslání`). Poté je v aplikaci vlevo dole napsáno zeleně `Připojeno`, v opačném případě je zde nápis napsán červeně, a to buď `Odpojeno`, nebo `Error`. Pokud máme funkční komunikaci a chceme pracovat se zařízením, které má jiné přenosové parametry

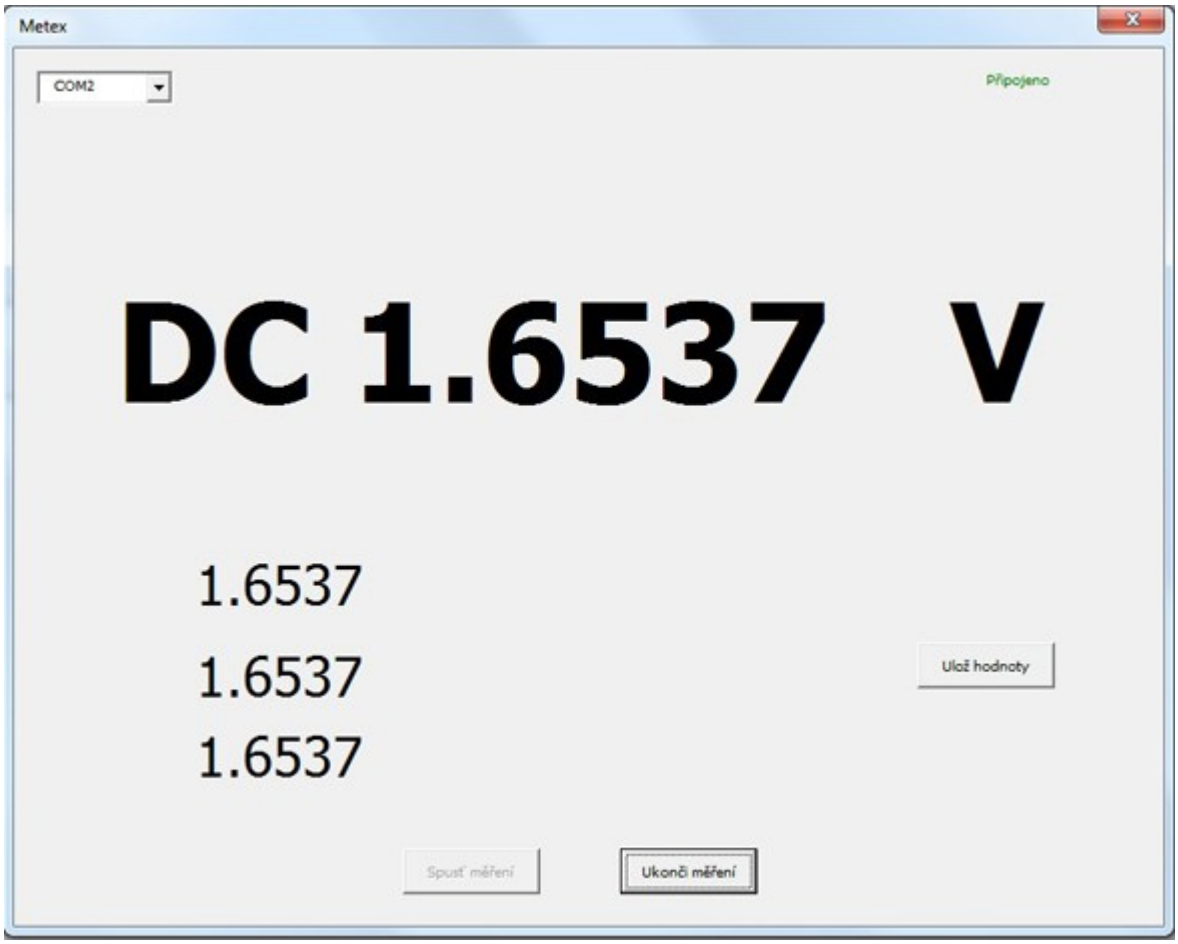
než zařízení, které jsme používali dosud, musíme nejdříve ukončit komunikaci a po nastavení požadovaných parametrů spustit opět tlačítko `Vytvoř spojení`.



Obrázek 4 – Okno aplikace UserForm_Messenger

3.3 Aplikace č. 2 (UserForm_Metex)

Jedná se aplikaci určenou k získávání hodnot z měřicího přístroje METEX do PC. Aktuální naměřené hodnoty je možno uložit do textového souboru, stačí kliknout na tlačítko vpravo `Ulož hodnoty`. Opět připomínám, že první aplikace dokáže také komunikovat s METEXem (méně přehledně), ale za předpokladu nastavení správných přenosových parametrů, řídicích signálů a zadáním správných příkazů. V této aplikaci nám stačí opět jen nastavit port připojeného METEXu a spustit měření tlačítkem `Spusť měření`. Po vytvoření komunikace je tlačítko `Spusť měření` zakázáno do doby, dokud není ukončeno měření. Zda je port otevřen, vidíme v pravém rohu nahoře (`Připojeno`, `Odpojeno`, `Error`). Hlavní údaj je zvýrazněn velkým písmem, pod ním se nachází dodatečné veličiny. Dodatečné veličiny jsou pokaždé jiné, podle nastaveného rozsahu měřicího přístroje. V případě měření střídavého napětí jsou dodatečnými veličinami frekvence (FR) a útlum (db).



Obrázek 5 – Okno aplikace pracující se zařízením METEX (měření stejnosměr. napětí)

3.3.1 Parametry komunikace METEXu:

Pro připojení METEXu jsem použil dodávaný kabel, který má na jedné straně 25 pinový konektor, takže ve většině případů musíme použít redukci na 9 pinový. Druhá strana kabelu je přizpůsobena tak, aby šla připojit k METEXu (je vyvedeno 5 kontaktů). Přenosová rychlost je 9600 Bd, 7 datových bitů, 2 Stopbity a žádná parita. Jsou použity signály RD (RXD), TD (TXD), GND, DTR a RTS. Data jsou vždy vysílána v počtu 56 bajtů, které se skládají ze 4 řádků po 14 bajtech. Každý 14. bajt obsahuje vždy znak Enter. Hodnoty v jednotlivých řádcích jsou zobrazeny dle nastaveného rozsahu přístroje, tzn. že např. při měření stejnosměrného napětí jsou ve všech řádcích hodnoty změřeného napětí, přičemž okamžitá měřená veličina je uvedena pouze v prvním řádku. V případě měření střídavého rozsahu jsou měřeny najednou tři veličiny (napětí, frekvence a útlum). V každém řádku, kromě druhého, jsou ke konci uvedeny jednotky. Druhý řádek je odeslán prázdný. Měření nastane po nastavení signálu DTR (signál RTS musí být shozen) a k tomu musíme poslat znak bez uvozovek „D”. V případě požadavku na událost, která nám ohlásí, že dorazil daný počet bajtů (v našem případě 56 bajtů), by se vyvolání události muselo nastavit podobně, jako je zmiňováno v případě teplotního čidla, tzn. počet bajtů nastavit v DCB a pak nastavit masku pro danou událost apod. Jelikož ve všech mých aplikacích je použit timer, byla cyklicky prováděna kontrola, zda se nachází v bufferu daný počet bajtů. V případě, že je v bufferu daný počet bajtů, pak teprve je použita funkce *ReadFile*.

Obrázek 6 – Měřicí přístroj METEX (M-4660A)

3.4 Aplikace č. 3 (UserForm_Temperature_Sensor)

Tato aplikace slouží pro měření teploty okolí. Pro měření teploty bylo použito digitální teplotní čidlo od výrobce PaPouch. Umožňuje měřit teploty v rozsahu -55 °C až +125 °C. Klikneme-li na tlačítko `Spusť měření`, je měření teploty zahájeno, tlačítko je zakázáno do momentu, kdy ukončíme měření. První měření máme zajištěno do 1 s, ostatní měření je prováděno opakovaně každých 10 s, dokud nestiskneme tlačítko `Ukonči měření`. Ačkoli je možné provádět měření teploty i přes první aplikaci, je tato aplikace vytvořena jako zjednodušená forma pro uživatele, kteří neznají parametry teplotního čidla nebo nechtějí při každém připojení digitálního teplotního čidla zadávat přenosové parametry a nastavovat či shazovat příslušné řídicí signály. Pro změření teploty nám tedy stačí nastavit port, na kterém máme teplotní čidlo připojeno, a spustit tlačítko `Spusť měření`. V pravém rohu nahoře se při správném otevření portu objeví nadpis `Připojeno`, při uzavření portu `Odpojeno`. `Odpojeno` je napsáno i před zahájením měření. Není-li port připojen ani odpojen, zobrazí se nápis `Error`.

Obrázek 7 – Ukázka aplikace pro měření teploty

3.4.1 Parametry komunikace teplotního čidla:

Čidlo používá přenosovou rychlost 9600 Bd s 8 datovými bity, 1 stopbitem a nepoužívá paritu. Připojení probíhá pomocí linky RS-232 (zjednodušená), 9 pinového konektoru CANNON. V případě, že máme jen 25 pin. konektor, musíme použít redukci. To byl i můj případ. Teplotní čidlo pro komunikaci využívá jen tří signálů, a to: RXD, GND a DTR. Pokud potřebujeme tedy změřit teplotu, stačí nahodit signál DTR a přijímaná data nám dorazí na RXD. Dokud budeme mít signál DTR nahozený, po tu dobu nám čidlo bude posílat změřenou teplotu každých 10 s, a to až do momentu, dokud signál neshodíme. Data nám dorazí v ASCII formátu jako testový řetězec včetně stupňů Celsia. Jelikož je každý vyslaný řetězec zakončen znakem Enter, je možné v aplikacích načítat data z bufferu, až když dorazí konkrétní znak (v případě teplotního čidla ASCII 13 = Enter). Znak, při kterém má nastat událost, se nastavuje ve struktuře DCB. Dále bychom museli nastavit pomocí masky, že událost má nastat při příchodu určitého znaku. Pak už nám zbývá jen kontrolovat, zda nastala událost. Jakmile nastane, použije se funkce *ReadFile* pro čtení z bufferu. V naší aplikaci jsme se rozhodli, že místo událostí pro čtení vytvoříme timer, který po uběhnutí nastavené doby cyklicky zkontroluje, zda jsou v bufferu nějaké bajty. V případě, že se zde nachází nejméně jeden bajt, je použita funkce *ReadFile*.

Obrázek 8 – Digitální teplotní čidlo

3.5 Vytvořené funkce

Veškeré funkce týkající se ovládání sériového portu se nacházejí v hlavním modulu Modul1. Tento modul pak dále využívají všechny tři aplikace (Messenger, METEX a Teplotní čidlo).

3.5.1 Funkce a procedura pro otevření či zavření komunikace

- **Function OpenPort**

Tato funkce zajišťuje vytvoření komunikace v nonoverlapped režimu. Stará se o otevření portu, nastavení jeho parametrů pomocí struktury DCB a nastavení timeoutů. Při volání funkce OpenPort se zadávají parametry funkce. Předává se název portu, přenosová rychlost, nastavení parity, počet datových bitů a počet Stopbitů. V případě úspěšného provedení funkce je návratovou hodnotou ID portu (handle), v opačném případě je navrácena celá číselná záporná hodnota, a to jedna ze třech možných v rozsahu -1 až -3. Pokud nelze port otevřít, je navrácena hodnota -1. Hodnota -2 značí, že nastala chyba při získávání aktuální struktury DCB. Poslední možnost je -3 a znamená, že nelze nastavit pro port nové nastavení DCB. Využito je funkce *SetCommTimeout* (vysvětlené dále) a API funkcí *CreateFile*, *GetCommState* a *SetCommState*.

Př.

```
OpenPort("COM2", 9600, "N", 7, 2)
```

- **Sub ClosePort**

Jak již název napovídá, jedná se o proceduru, která nám ukončí jen otevřenou komunikaci. Využito funkce API *CloseHandle*.

Př.

```
ClosePort
```

3.5.2 Procedura pro nastavení timeoutů

- **Sub SetCommTimeout**

Procedura nastavuje timeouty pro vysílání i příjem. Obsahuje strukturu CommTimeouts, v níž se provede nastavení jednotlivých timeoutů, které se pak nastaví pomocí API funkce *SetCommTimeouts*. Vstupní parametr *SetCommTimeout* je čas v ms, který se nastaví do ReadTotalTimeoutConstant a WriteTotalTimeoutConstant. ReadIntervalTimeout a ReadTotalTimeoutMultiplier jsou nastaveny na 1 ms. WriteTotalTimeoutMultiplier je nastaven na 10 ms.

Př.

```
SetCommTimeout (10)
```

3.5.3 Funkce pro odesílání a příjem dat

- **Function SendString**

Funkce odesílá naše data po sériové lince. Vstupním parametrem je tedy String, který je předán API funkci *WriteFile*. Návratovou hodnotou je při úspěšném provedení 1, při neúspěšném 0.

Př.

```
SendString("Test")
```

- **Function ReadByte**

Funkce zajišťuje načtení přijatého bajtu. Funkce nemá žádné parametry. Návratovou hodnotou je ASCII přijatý znak v případě úspěšného načtení. Pokud došlo k chybě, je návratovou hodnotou -1. Využito je funkce API *ReadFile*.

Př.

```
Dim Code As Integer
Code = ReadByte
```

3.5.4 Funkce pro zjišťování, nastavování řídicích signálů

- **Sub Set_DTR**

Procedura pro nahození či shození výstupu DTR. Vstupním parametrem je hodnota Boolean, tzn. True nebo False dle požadavku nahození či shození DTR. Využito je funkce API *EscapeCommFunction*.

Př.

```
Set_DTR(True)
```

- **Sub Set_RTS**

  Procedura funguje stejně jako Set_DTR, pouze se týká výstupu RTS a ne DTR.

  Př.

```
Set_RTS(False)
```

- **Function Get_CTS**

Funkce pro zjištění řídicího signálu CTS na vstupu. Pomocí masky se zjistí nahození či shození signálu CTS. Vstupní parametr nezadáváme. Návratovou hodnotou jsou Boolean hodnoty (True/False) dle aktuálního nastavení stavu CTS. Zjištění stavu nám zajistí funkce API *GetCommModemStatus*.

Př.

```
Get_CTS
```

- **Function Get_DSR**

Funkce pro zjištění vstupního řídicího signálu DSR. Funkce funguje jako *Get_CTS* pouze pro signál DSR.

Př.

```
Get_DSR
```

- **Function Get_RING**

Tato funkce je předposlední funkcí ke sledování aktuálního stavu vstupního řídicího signálu RING. Funguje opět jako *Get_CTS*.

Př.

```
Get_RING
```

- **Function Get_RSLD**

Get_RSLD je poslední funkcí na zjištění aktuálního stavu vstupního řídicího signálu RSLD. Také funguje jako Get_CTS.

Př.

```
Get_RSLD
```

3.5.5 Funkce pro práci s buffery

- **Function Buffer_Not_Empty**

Funkce pro zjištění počtu bajtů ve vstupním bufferu. Návratovou hodnotou je počet bajtů v přijímacím bufferu nebo hodnota 0 (prázdný buffer). Využívá API funkce *ClearCommError* pro zjištění naplněnosti bufferu.

Př. (dokud vstupní buffer obsahuje bajty, jsou po jednom odebírány)

```
Do While (Buffer_Not_Empty > 0)
ReadByte
Loop
```

- **Function Set_Buffer**

Funkce nastavuje velikosti vstupního a výstupního bufferu. Parametry funkce jsou jednotlivé velikosti pro každý vstupní a výstupní buffer. Prvním parametrem je hodnota pro vstupní buffer, druhým je velikost výstupního bufferu. Návratová hodnota je nenulová v případě úspěchu, v opačném případě je to nula. K nastavení bufferů se využívá API funkce *SetupComm*.

Př.

```
Set_Buffer(100,100)
```

- **Sub Clear_Buffer**

Procedura zajišťuje vyprázdnění vstupního a výstupního bufferu. Parametry nemá. Pro vyprázdnění bufferu je určena funkce *PurgeComm*, kterou zde využívám.

Př.

```
Clear_Buffer
```

3.5.6 Procedury pro spuštění, ukončení timeru

- **Sub Timer_ON**

Procedura pro nastavení timeru (časovače), který nám umožní cyklické načítání příchozích dat a zjišťování aktuálních stavů vstupních řídicích signálů. Timer kontroluje stav každých 500 ms. Jelikož timer využívají všechny tři aplikace (Messenger, METEX, Teplotní čidlo) musí být timer rozlišen, pro kterou aplikaci bude následně používán. To se nastaví jen dle jediného vstupního parametru. API funkce, kterou zde používám, se jmenuje *SetTimer*.

Př.

```
Timer_ON(2)
```

- **Sub Timer_OFF**

Procedura nám umožní ukončit spuštěný časovač (timer). Parametr neobsahuje. Ukončení funkce timeru je zajištěno API funkcí *KillTimer*.

Př.

```
Timer_OFF
```

3.5.7 Procedury spouštěné pro jednotlivé aplikace

- **Sub TimerProc**

Po uběhnutí nastaveného času časovače je zavolána tato procedura, která je použita pro Messenger. Časovač si ji vždy volá sám. Dochází ke zjišťování aktuálních řídicích stavů na vstupu a dokud jsou v bufferu data, jsou po jednom bajtu vybírána. Využívá funkcí *Get_CTS*, *Get_DSR*, *Get_RING*, *Get_RSLD*, *Buffer_Not_Empty* a *ReadByte*.

- **Sub TimerProc2**

Tato procedura funguje obdobně jako předchozí *TimerProc* také s časovačem, ale nedochází ke zjišťování vstupních řídicích signálů a bajty jsou z bufferu vybírány až poté, co se jich ve vstupním bufferu objeví 56. Je tedy použita při měření METEXem. Využívá funkce *Buffer_Not_Empty*, *ReadByte* a *SendString*.

- **Sub TimerProc3**

Procedura je využívána za použití časovače v případě teplotního čidla. Funguje obdobně jako *TimerProc*. Používá funkce *Buffer_Not_Empty* a *ReadByte*.

4 Závěr

Jak jsem již v úvodu uvedl, cílem mé bakalářské práce bylo vytvořit funkce a uživatelské rozhraní pro ovládání sériového portu z prostředí MS Excel. Nejdříve jsem se seznámil s informacemi o sériové komunikaci a sériovém portu. Poté jsem prostudoval programování ve VBA a následně zjistil jak pomocí API ovládat sériový port. Na závěr jsem vytvořil aplikace komunikující po sériovém portu.

Po vypracování praktické části jsem došel k závěru, že z pohledu programátora záleží na tom, co všechno chce programátor nastavovat či kontrolovat. V případě, že mu stačí jen jednoduše např. otevřít port (bez nastavování dalších parametrů) a rovnou komunikovat, pak může využít např.: knihovnu RSCOM.DLL. Využije-li tuto knihovnu, je ale omezen jejími funkcemi. Chceme-li tuto knihovnu použít, musíme ji do PC nakopírovat. V opačném případě, kdy nám nestačí již vytvořené funkce z DLL pro sériovou komunikaci a my chceme plně ovládat sériový port, musíme využít funkce API. Díky API se nám otevírají široké možnosti při práci se sériovým portem, máme věci více pod kontrolou a ve své režii, ale u programátorů vyžaduje používání API větší znalosti Windows a programování. Při volání funkcí API se dále využívají knihovny DLL, které jsou součástí OS, a tudíž je nemusíme nikde shánět a kopírovat do PC. Tím pádem mají naše aplikace velkou přenositelnost mezi počítači s Windows. Rozhodli jsme se tyto funkce využít a na ukázku jsme vytvořili 3 aplikace (aplikaci „messenger" po sériovém portu, aplikaci pro multimetr METEX a pro teplotní čidlo). Tyto aplikace jsou uloženy na CD, které je součástí této bakalářské práce.

Literatura

[1] KAINKA, Burkhard. 2005. *Měření, řízení a regulace pomocí PC. Vývoj hw a sw pro praxi.* Praha : BEN, 2005. 271 s. ISBN 80-7300-089-X.

[2] *RS-232.* [online]. Naposledy editována 31. 3. 2011 [cit. 4. 4. 2011]. Dostupné na: <http://cs.wikipedia.org/wiki/RS-232>.

[3] DENVER, Allen. 1995. *Serial Communications in Win32.* [Online] 11.12.1995. [Citace: 6. 4 2011.] Dostupné na: <http://msdn.microsoft.com/en-us/library/ms810467.aspx>.

[4] *Co je knihovna DLL?* [online]. ID článku: 815065 4. 12. 2007 [cit. 21. 3. 2011]. Dostupné na: <http://support.microsoft.com/kb/815065/cs>.

[5] SKÁLA, Kamil. 2006. *Win32 API – Úvod.* [Online] 1.1 2006. [cit. 16. 3 2011.] Dostupné na: <http://programujte.com/?akce=clanek&cl=2005122808-win32-api-uvod>.

[6] PECHÁČEK, Petr. *Excelentně v Excelu I. a II.* [Online] 29. 3. 2010 [cit. 4. 4. 2011]. Dostupné na: <http://www.instaluj.cz/excelentne-v-excelu-i-a-ii->.

[7] WALKENBACH, John. 2008. *Microsoft Office Excel 2007. Programování ve VBA.* Brno : Computer Press, 2008. 893 s. ISBN 978-80-251-2011-8.