ChnV40P

JEDNOTKA SÉRIOVÉ KOMUNIKACE RS232 / RS485 S OBVODEM I8251 PROCESORU V40 A PROTOKOLEM PRT

Příručka uživatele a programátora

***SofCon*®spol. s r.o.**
Střešovická 49
162 00 Praha 6
tel/fax: +420 220 180 454
E-mail: sofcon@sofcon.cz
www: http://www.sofcon.cz

Verze dokumentu 1.20

Informace v tomto dokumentu byly pečlivě zkontrolovány a SofCon věří, že jsou spolehlivé, přesto SofCon nenese odpovědnost za případné nepřesnosti nebo nesprávnosti zde uvedených informací.
SofCon negarantuje bezchybnost tohoto dokumentu ani programového vybavení, které je v tomto dokumentu popsané. Uživatel přebírá informace z tohoto dokumentu a odpovídající programové vybavení ve stavu, jak byly vytvořeny a sám je povinen provést validaci bezchybnosti produktu, který s použitím zde popsaného programového vybavení vytvořil.
SofCon si vyhrazuje právo změny obsahu tohoto dokumentu bez předchozího oznámení a nenese žádnou odpovědnost za důsledky, které z toho mohou vyplynout pro uživatele.

Datum vydání: 06.08.2004
Datum posledního uložení dokumentu: 06.08.2004
(Datum vydání a posledního uložení dokumentu musí být stejné)

Upozornění:
V dokumentu použité názvy výrobků, firem apod. mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

Copyright © 2004, SofCon® spol. s r.o., Adam Wild

Obsah :

1. O dokumentu

1.1. Revize dokumentu

Verze dokumentu	Verze SW	Autor	Datum vydání	Popis změn
1.00	1.XX	Wi		První vydání
1.10	4.XX	Tu	16.05.2003	Úprava dokumentu dle ISO9000 Doplněné CH_SBuff a CH_SMSuff Zrušené konstanty SOH, DLE, EXT
1.20	4.XX	Wil	06.08.2004	Přidána automatická kompenzace zrychlování systémového časovače metod ChConnect a ChDisconnect a úprava jejich popisu.

1.2. Účel dokumentu

Tento dokument slouží jako popis jednotky sériové komunikace RS232 / RS485 s obvodem i8251 v procesoru V40 a protokolem PRT (tento protokol je také definován v knihovně ChnPrt). Ke své činnosti využívá služeb MCP BIOSu.

1.3. Rozsah platnosti

Určen pro programátory a uživatele programového vybavení SofCon.

1.4. Související dokumenty

Pro čtení tohoto dokumentu je potřeba seznámit se s manuálem „ChnVirt“ popisujícím jednotné rozhraní všech komunikačních objektů, dále knihovna používá některé typy z knihovny „ChTypes“ a z knihovny „uComM“ používá procedury na volání obsluhy komunikačního kanálu V40.

Popis formátu verze knihovny a souvisejících funkcí je popsán v manuálu „LibVer“.

2. Termíny a definice

Používané termíny a definice jsou popsány v samostatném dokumentu „Termíny a definice“.

3. Úvod

Knihovna ChnV40P definuje objekt **tChnV40P**, jehož instance vytváří nižší a zároveň i vyšší vrstvu v komunikačním kanálu tvořenou protokolem a sériovým rozhranním RS232 nebo RS485 s obvodem i8251 obsaženým v procesoru V40. Ke své činnosti využívá přerušovacího systému počítače prostřednictvím služeb obsažených v BIOSu MCP, kde je definován také formát protokolu. Přenášený protokol je binární a obsahuje adresu adresáta i odesílatele. Umožňuje zaslat zprávu všem připojeným stanicím najednou. Výsledný protokol je shodný s protokolem, který je naimplementován v komunikačním objektu tChnPrt v knihovně ChnPrt. Komunikační objekt tChnV40P je v podstatě sloučením komunikačního objektu nižší fyzické vrstvy **tChnV40** z knihovny ChnV40 a komunikačního objektu vyšší protokolové vrstvy **tChnPrt** z knihovny ChnPrt.

Knihovna rovněž definuje objekt **tAddChnV40P**, který je dědicem od rodičovského objektu tAddChnVirt. Objekt tAddChnV40P zajistí, aby daný komunikační objekt (objekt tChnV40P) byl k aplikaci připojen a popřípadě zajistí vytvoření instance tohoto objektu. Po přilinkování této jednotky do aplikace (příkazem "uses ChnV40P"), se jméno objektu tChnV40P automaticky vloží do seznamu správců komunikačních objektů pro případné použití.

Protože je objekt **tChnV40P** dědicem rodičovského komunikačního objektu **tChnVirt**, jsou v této příručce popsány jen odlišnosti a speciality pro tento druh sériové komunikace. Ostatní naleznete v příručce **ChnVirt**. Některé použité konstanty a typy jsou předdefinované v jednotce **ChnTypes**.

4. Popis konstant a typů

```
cVerNo = např. $0251; { BCD formát }
cVer   = např. '02.51,07.08.2003';
```

Číslo verze jednotky v BCD tvaru a v textové podobě včetně datumu změny.

```
cName  = 'V40P';
```

Konstanta **cName** definuje jméno komunikačního objektu **tChnV40P**.

5. Definice přenášeného protokolu

1	1	1	2	LEN	2	1	velikost v byte
SOH	DNODE	NODE	LEN	DATA	CRC	ETX	

Význam položek:

SOH - začátek zprávy (sekvence znaků DLE, SOH)
DNODE - adresa cílové stanice (destination node)
NODE - adresa zdrojové stanice (node)
LEN - délka vysílaného pole DATA
DATA - pole dat
CRC - zbytek po dělení cyklickým polynomem CRC16, $x^{16}+x^{15}+x^2+1$, který je generován z položek zprávy SOH až poslední byte pole DATA
ETX - konec zprávy (sekvence znaků DLE, ETX)

U položek **LEN** a **CRC** je zasílán nižší byte jako první.

Velikost přenášeného pole dat **DATA** je omezena na 32734 byte.

Pokud některý z bytů položek **DNODE** až poslední byte **CRC** je roven hodnotě znaku DLE, je tento znak vyslán dvakrát (DLE, DLE). Tím je zajištěno rozlišení, jedná-li se o řídící znak protokolu (SOH nebo ETX) či nikoliv.

6. Objekty

6.1. tChnV40P

6.1.1. Položky

`Ch_Rate : tRate;`
Položka **Ch_Rate** obsahuje požadovanou přenosovou rychlost v bitech za sekundu.

`Ch_Parity : tParity;`
Položka **Ch_Parity** obsahuje požadovanou paritu přenášeného znaku.

`Ch_Stop : tStop;`
Položka **Ch_Stop** obsahuje počet stop bitů u přenášeného znaku.

`Ch_Length : tLength;`
Položka **Ch_Length** obsahuje počet datových bitů u přenášeného znaku.

`CH_RSDelay1 : Longint;`
Položka **CH_RSDelay1** definuje minimální časovou prodlevu v ms před vysíláním.

`CH_RSDelay2 : Longint;`
Položka **CH_RSDelay2** definuje minimální časovou prodlevu v ms po vysílání.

`CH_SBuff : Pointer;`
Položka **CH_SBuff** definuje ukazatel na vysílací buffer.

`CH_MSBuff : Word;`
Položka **CH_MSBuff** definuje velikost vysílacího bufferu.

`CH_SMess : Pointer;`
Položka **CH_SMess** definuje ukazatel na vysílanou zpravu.

`CH_LSMess : Word;`
Položka **CH_LSMess** definuje délku vysílané zprávy.

`CH_RecOn : Boolean;`
Položka **CH_RecOn** určuje, zda se má během vysílání ponechat povolený příjem znaků či zda ho zakázat.

`CH_STime : tTimer;`
Položka **CH_STime** je určena pro vnitřní použití, pro odměřování časových intervalů vysílače.

`CH_STick : Boolean;`
Položka **CH_STick** je určena pro vnitřní použití, pro určení, zda je vykonávána činnost vysílacího automatu.

6.1.2. Metody

6.1.2.1. Init konstruktor

```
constructor Init;
```

Konstruktor **Init** slouží k vytvoření a inicializaci instance komunikačního objektu. Ve svém těle volá nejprve zděděný konstruktor **Init** z rodičovského objektu **tChnVirt** a poté inicializuje položky objektu. Tělo konstruktoru vypadá následovně:

```
inherited Init;
CH_Type     := cName;
CH_Name     := CH_Type
CH_NumName  := ChNumName(CH_Type);
CH_Rate     := 4800;
CH_Parity   := ParOdd;
CH_Stop     := Stop1;
CH_Length   := Bits8;
CH_RSDelay1 := 0;
CH_RSDelay2 := 0;
CH_SBuff    := nil;
CH_MSBuff   := 0;
CH_SMess    := nil;
CH_LSMess   := 0;
CH_RecOn    := true;
CH_STick    := false;
```

6.1.2.2. ChInitParam konstruktor

```
constructor ChInitParam(const S: TParamStr);
```

Konstruktor **ChInitParam** je sloučením konstruktoru **Init** a metody **ChSetparam**. Slouží ke zkrácenému vytvoření instance komunikačního objektu s nastavením parametrů komunikace.

6.1.2.3. Done destruktor

```
destructor Done;
```

Destruktor **Done** slouží ke zrušení instance komunikačního objektu. Pokud je alokován vysílací buffer, je odstraněn z paměti. Na konci destruktoru je volána zděděná metoda **Done** od přímého rodičovského objektu pro uzavření podřízené komunikační vrstvy.

6.1.2.4. ChSetOneParam funkce

```
procedure ChSetParam(S: TParamStr);
function  ChSetOneParam(const S: tWordString; var CmdL: tCmd)
          : tChResult;
```

Metoda **ChSetOneParam** slouží k dekódování a nastavení jednoho konkrétního parametru, který je zadán v parametru S. Tato metoda se volá v aplikaci prostřednictvím metody **ChSetParam**. Metoda **ChSetOneParam** komunikačního objektu tChnPrt dekóduje tyto parametry:

LSB=Size

Parametrem **LSB** ("Length of Send Buffer") je alokován nový vysílací buffer **CH_MSBuff** dané velikosti Size. Do tohoto bufferu je transformována vysílaná zpráva, která je předána objektu nižší komunikační vrstvy k odeslání. Size může nabývat hodnot 17 až 32750 byte.

NOD=aaa

Parametr definuje node stanice na komunikační síti. aaa může nabývat hodnot 0 až 255. Tento parametr je možné nastavit jen před voláním metody **ChConnect**.

DNO=bbb

Parametr definuje node adresáta, kterému budou zprávy určeny. Node adresáta je možné také definovat voláním metody **ChDestNode**. bbb může nabývat hodnot 0 až 255.

BD=aaa

Parametr **BD** ("BaudRate") určuje přenosovou rychlost požadované sériové komunikace. aaa může nabývat hodnot 25, 50, 75, 100, 110, 150, 300, 600, 1200, 2400, 4800, 9600 nebo 19200 Bd. Při rychlostech 9600 a 19200 Bd je automaticky nastaven jiný poměr v předděliči vstupních hodin, což ovlivní i všechny k němu připojená zařízení. Při rychlosti 9600 Bd bude systémový časovač zrychlen dvakrát, při rychlosti 19200 Bd čtyřikrát.

BIT=bbb

Parametr **BIT** ("Number of Data Bits") určuje počet datových bitů v přenášeném znaku. bbb může nabývat hodnot 5 až 8.

PAR=ccc

Parametr **PAR** ("Parity") určuje paritu přenášeného znaku. ccc může nabývat hodnot O "Odd" pro lichou paritu, E "Even" pro sudou paritu a N "None" pro znak bez parity.

STO=ddd

Parametr **STO** ("Number of Stop Bits") určuje počet stop-bitů v přenášeném znaku. ddd může nabývat hodnot 1 nebo 2.

LRB=eee

Parametr **LRB** ("Length of Receive Buffer") určuje velikost vstupního kruhového vyrovnávacího bufferu. Buffer je alokován na heapu a každá jeho položka zaujímá v paměti prostor o velikosti 2 byte (přijatý znak a status). Platí, čím větší je vstupní buffer, tím více znaků dokáže udržet, aniž by byly znaky metodami **ChReceive** a **ChReceiveChar** odebírány. Velikost bufferu je shora omezena na 32750. Je proto nutno volit kompromis mezi velikostí bufferu a periodou, kterou přijaté znaky zpracováváme.

RS1=fff

Parametr **RS1** určuje zpoždění v ms před vysíláním.

RS2=ggg

Parametr **RS2** určuje zpoždění v ms po vysílání.

REC=hhh

Parametr **REC** ("Receive While Sending") určuje, zda má být pří vysílání povolen příjem znaků. hhh může nabývat hodnot ON nebo OFF.

Příklad:

Příklad ukazuje, jak je možné nastavit v komunikačním objektu se jménem V40P parametry NODE na hodnotu 20, DNODE na hodnotu 30, rychlost přenosu na hodnotu 9600, sudou paritu, velikost vstupního a výstupního vyrovnávacího bufferu na 1000 položek.

```
ChSetParam('NAM=V40P NOD=20 DNO=30 BD=9600 PAR=E LRB=1000 LSB=1000');
```

Pozn.: Všimněte si, že není volána metoda ChSetOneParam, ale metoda ChSetParam.

6.1.2.5. ChGetParam funkce

```
function  ChGetParam(const S: TParamStr): TParamStr;
```

Metoda **ChGetParam** navrací nastavené hodnoty parametrů komunikačního objektu. Nejprve vrátí nastavení parametrů rodičovského komunikačního objektu tChnVirt a poté k nim připojí seznam svých parametrů. Seznam parametrů je uveden výše u popisu metody **ChSetOneParam**.

6.1.2.6. ChConnect procedura

```
procedure ChConnect;
```

Metoda **ChConnect** zavolá zděděnou metodu **ChConnect** od přímého rodičovského objektu a pokud nenastala žádná chyba, nastaví automat přijímače **CH_RCtrl** do počátečního stavu pro příjem zprávy.

Při komunikační rychlosti 9600Bd a vyšší může dojít ke zrychlení systémového časovače. Proto je zavolána automatická korekce, která se pokusí toto zrychlení kompenzovat systémovými prostředky knihovny **Tick**. Více podrobností o této korekci se dozvíte v dokumentu knihovny Tick.

6.1.2.7. ChDisConnect procedura

```
procedure ChDisConnect;
```

Metoda **ChDisConnect** zavolá zděděnou metodu **ChDisConnect** od přímého rodičovského objektu a pokud nenastala žádná chyba, nastaví automat přijímače **CH_RCtrl** do neaktivního stavu, aby se nepřijímaly žádné zprávy.

Při komunikační rychlosti 9600Bd a vyšší může dojít naopak ke zpomalení systémového časovače. Proto je opět zavolána automatická korekce, která se pokusí toto zpomalení kompenzovat systémovými prstředky knihovny **Tick**. Více podrobností o této korekci se dozvíte v dokumentu knihovny Tick.

6.1.2.8. ChSend procedura

```
procedure ChSend(Buff : Pointer; Len : Word);
```

Metoda **ChSend** způsobí započetí vysílání zprávy s datovým polem, na které ukazuje parametr Buff, podle výše definovaného protokolu. Parametr Len udává délku vysílacího bufferu pro vysílání (pro tento protokol délku vysílaných dat).

6.1.2.9. ChSendReady funkce

```
function  ChSendReady: TChState;
```

Metoda **ChSendReady** způsobí provedení kroku vysílacího automatu na základě volání metody **ChSendTick**. Jako svoji funkční hodnotu vrátí aktuální stav automatu vysílače komunikačního kanálu, který je uložen v položce **CH_SCtrl**. Pokud kanál není ve stavu **CHS_Connect**, vrací metoda stav **CHS_SendNoReady**.

6.1.2.10. ChReceiveReady funkce

```
function  ChReceiveReady: tChState;
```

Metoda **ChReceiveReady** způsobí provedení kroku přijímacího automatu na základě volání metody **ChReceiveTick**. Jako svoji funkční hodnotu vrací aktuální

stav automatu přijímače komunikačního kanálu, který je uložen v položce **CH_RCtrl**. Zpravidla se provádí test pouze na stabilní stav **CHS_ReceiveReady** (který znamená, že byla přijata nějaká zpráva), protože ostatní stavy jsou stavy probíhajícího příjmu.

6.1.2.11. ChReceive procedura

```
procedure ChReceive(var Len: Word);
```

Metoda **ChReceive** provede přijmutí celé zprávy a její uložení do přijímacího bufferu, který byl definován metodou **ChReceiveBuffer**. Metoda naplní buffer pouze patřičnými daty, úvodní a zakončovací řídicí znaky a kontrolní součet ze zprávy metoda vyhodnotí a pro uživatele odstraní. Odpověď na zprávu, ať došla v pořádku nebo porušená, generuje uživatel sám pomocí metody **ChSend**.

6.1.2.12. ChReceiveFlush procedura

```
procedure ChReceiveFlush;
```

Metoda **ChReceiveFlush** způsobí vyprázdnění přijímacích bufferů a nastavení stavu automatu přijímače na počátek příjmu zpráv.

6.1.2.13. ChGetNode procedura

```
procedure ChGetNode(var SNode, DNode: TNode);
```

Po volání metody **ChGetNode** je do proměnné **SNode** uloženo číslo (adresa) stanice, která zprávu odeslala, a do proměnné **DNode** číslo (adresa) stanice, pro kterou byla zpráva určena. Tuto metodu má smysl volat po přijetí zprávy metodou **ChReceive**.

6.1.2.14. ChSendTick procedura

```
procedure ChSendTick;
```

Metoda **ChSendTick** způsobí provedení jednoho či více kroků vysílacího automatu. Je nutné ji periodicky volat během vysílání. Metoda **ChSendTick** je rovněž automaticky volána v metodě **ChSendReady**.

6.2. tAddChnV40P

Typ **tAddChnV40P** je typem objektu, který slouží k definování prvku v seznamu správců komunikačních objektů (tzv. správce komunikačního objektu tChnV40P v seznamu správců). Objekt tAddChnV40P je dědicem od rodičovského objektu **tAddChnVirt**.

6.2.1. Metody

6.2.1.1. ChInit funkce

```
function ChInit: pChnVirt;
```

Metoda **ChInit** slouží k vytvoření instance komunikačního objektu tChnV40P a ukazatel na instanci tohoto objektu vrací jako svoji funkční hodnotu.

7. Příklad

Příklad ukazuje použití komunikační jednotky ChnV40P. Je vytvořen komunikační kanál definovaných vlastností, po kterém je zasílána zpráva a z kterého je poté očekáván příjem zpráv.

```
uses
  uString,
  ChnVirt,
  ChnV40P,
  ...
const
  ParamStr : tParamStr =
             'NAM=V40P NOD=1 DNOD=2 '+
             'BD=1200 BIT=8 PAR=E STOP=2 LRB=1000 LSB=1000';
  LMess    = 40;
type
  tMess    = array [0..LMess+10] of Byte;

var
  Chn      : pChnVirt;
  SMess    : ^tMess;
  RMess    : ^tMess;
  LRMess   : Word;

begin
  ...
  New(SMess);
  New(RMess);
  ...

  { vytvoření instance Chn }
  Chn:=ChnCollection^.ChNewInit(ChnV40P.cName);
  with Chn^ do
  begin
    { nastavení parametrů komunikace }
    ChSetParam(ParamStr);
    if ChResult<>res_Ok then WriteLn('Chyba');
    ChOpen;
    repeat
      if ChResult<>res_Ok then WriteLn('Chyba');
    until ChReady=CHS_Open;
    if ChResult<>res_Ok then WriteLn('Chyba');
    { definování místa, kam se má přjatá zpráva uložit }
    ChReceiveBuffer(RMess,SizeOf(RMess^));
    if ChReceiveResult<>res_Ok then WriteLn('Chyba');
    ChConnect;
    repeat
      if ChResult<>res_Ok then WriteLn('Chyba');
    until ChReady=CHS_Connect;
    if ChResult<>res_Ok then WriteLn('Chyba');
    ...
    { naplnění zprávy daty a naplnění délky vysílané zprávy LSMess }
    ...
    { vyslání zprávy }
    if ChSendReady=CHS_SendReady then
    begin
      ChSend(SMess, LSMess);
      { čekání na odvysílání zprávy }
      repeat
        if ChSendResult<>res_Ok then Writeln('Chyba');
      until ChSendReady=CHS_SendReady;
      if ChSendResult<>res_Ok then WriteLn('Chyba');
      ...
```

```
    end;
    ...
    { čekání na příjem zprávy }
    while not ChReceiveReady=CHS_ReceiveReady do
    begin
      if ChReceiveResult<>res_Ok then Writeln('Chyba');
    end;
    { příjem zprávy }
    ChReceive(LRMess);
    if ChReceiveResult<>res_Ok then WriteLn('Chyba')
    else
      { zpráva se přijala }
    ...
    { ukončení }
    ChDisConnect;
    repeat
      if ChResult<>res_Ok then WriteLn('Chyba');
    until ChReady=CHS_DisConnect;
    if ChResult<>res_Ok then WriteLn('Chyba');
    ChClose;
    repeat
      if ChResult<>res_Ok then WriteLn('Chyba');
    until ChReady=CHS_Close;
    if ChResult<>res_Ok then WriteLn('Chyba');
  end;
  { zrušení instance Chn }
  Dispose(Chn,Done);
  ...
end.
```