ChnSBus

JEDNOTKA DEFINUJÍCÍ KOMUNIKAČNÍ PROTOKOL SBUS

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: 07.04.2004
Datum posledního uložení dokumentu: 07.04.2004
(Datum vydání a posledního uložení dokumentu musí být stejné)

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	22.05.2003	Úprava dokumentu dle ISO9000.
1.20	4.XX	Wil	07.04.2004	Změna číslování chybových konstant res_ErrXxx dle standardu.

1.2. Účel dokumentu

Tento dokument slouží jako popis jednotky definující komunikační protokol SBUS.

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 základní rodičovský prvek pro tvorbu komunikačních objektů a s manuálem „ChnTypes“.

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 definuje formát protokolu SBUS používaného při komunikaci se zařízeními firmy SAIA. Obstarává zabezpečení dat, vkládání a vyjímání nadbytečností, tak jak to tento protokol předepisuje. Fyzický přenos dat je zajištěn prostřednictvím nižší komunikační vrstvy.

Knihovna ChnSBus definuje komunikační objekt **tChnSBus**, který je dědicem od rodičovského komunikačního objektu tChnVirt. Instance objektu tChnSBus representuje vyšší komunikační vrstvu v komunikačním kanálu. Transformuje předávaná data mezi komunikačními objekty nižších vrstev, které provádějí fyzický přenos, a aplikací popřípadě další vyšší komunikační vrstvou. Určení, přes jakou fyzickou komunikační vrstvu bude komunikace probíhat je voleno až parametrem nastavovací metody ChSetParam.

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

Protože je objekt **tChnSBus** 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  = 'SBUS';
```

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

```
cmd_GetAddrByte = $0202;
```

Konstanta **cmd_GetAddrByte** definuje hodnotu Code pro metodu **ChGetBinParam** komunikačního objektu nižší vrstvy. Slouží pro určení stavu přijetí speciálního znaku, který označuje začátek zprávy. Tato konstanta musí mít stejnou hodnotu jako příslušné konstanty z jednotek ChnComPB a ChnComBR.

4.1. Konstanty chybových kódů

Následující chyby mohou vracet metody ChReceiveResult, ChSendResult a případně i ChResult.

```
Res_ErrFrame    = $20;
```

Chybný formát zprávy - přijatou zprávu nelze akceptovat.

```
Res_ErrCrc      = $21;
```

Chyba kontrolního součtu CRC16 - přijatou zprávu nelze akceptovat.

`Res_ErrLen = $22;`
Chyba délky zprávy - přijatou zprávu nelze akceptovat.

`Res_ErrVal = $23;`
Chyba při převodu čísla na textový řetězec a naopak.

`Res_ErrUnknownCode = $25;`
Chyba indikující neznámý Code.

4.2. Kódy služeb protokolu

`R_Counter = 00;`
Konstanta **R_Counter** definuje kód zprávy pro čtení čítače.

`R_DispReg = 01;`
Konstanta **R_DispReg** definuje kód zprávy pro čtení display registru.

`R_Flag = 02;`
Konstanta **R_Flag** definuje kód zprávy pro čtení příznaku.

`R_Input = 03;`
Konstanta **R_Input** definuje kód zprávy pro čtení vstupu.

`R_RTC = 04;`
Konstanta **R_RTC** definuje kód zprávy pro čtení hodin reálného času.

`R_Output = 05;`
Konstanta **R_Output** definuje kód zprávy pro čtení výstupu.

`R_Register = 06;`
Konstanta **R_Register** definuje kód zprávy pro čtení registru.

`R_Timer = 07;`
Konstanta **R_Timer** definuje kód zprávy pro čtení časovače.

`W_Counter = 10;`
Konstanta **W_Counter** definuje kód zprávy pro zápis čítače.

`W_Flag = 11;`
Konstanta **W_Flag** definuje kód zprávy pro zápis příznaku.

`W_RTC = 12;`
Konstanta **W_RTC** definuje kód zprávy pro zápis hodin reálného času.

`W_Output = 13;`
Konstanta **W_Output** definuje kód zprávy pro zápis výstupu.

`W_Register = 14;`
Konstanta **W_Register** definuje kód zprávy pro zápis registru.

`W_Timer = 15;`
Konstanta **W_Timer** definuje kód zprávy pro zápis časovače.

`R_StsCpu0 = 20;`
Konstanta **R_StsCpu0** definuje kód zprávy pro čtení statusu 0. procesoru.

`R_StsCpu1 = 21;`
Konstanta **R_StsCpu1** definuje kód zprávy pro čtení statusu 1. procesoru.

`R_StsCpu2 = 22;`
Konstanta **R_StsCpu2** definuje kód zprávy pro čtení statusu 2. procesoru.

`R_StsCpu3 = 23;`
Konstanta **R_StsCpu3** definuje kód zprávy pro čtení statusu 3. procesoru.

`R_StsCpu4 = 24;`
Konstanta **R_StsCpu4** definuje kód zprávy pro čtení statusu 4. procesoru.

`R_StsCpu5 = 25;`
Konstanta **R_StsCpu5** definuje kód zprávy pro čtení statusu 5. procesoru.

`R_StsCpu6 = 26;`
Konstanta **R_StsCpu6** definuje kód zprávy pro čtení statusu 6. procesoru.

`R_StsCpu7 = 27;`
Konstanta **R_StsCpu7** definuje kód zprávy pro čtení statusu vlastního procesoru.

4.3. Řídící znaky protokolu

`ACK = $06;`
Konstanta **ACK** definuje kód znaku pro krátké pozitivní potvrzení od stanice Slave na úspěšný zápis parametru.

`NAK = $15;`
Konstanta **NAK** definuje kód znaku pro krátké negativní potvrzení od stanice Slave na neúspěšný zápis parametru.

`FS = $B5;`
Konstanta **FS** (Frame Synchronization) definuje hodnotu prvního znak zprávy pro datový režim.

`DLE = $C5;`
Konstanta **DLE** (Data Link Escape) definuje hodnotu pomocného znaku pro vysílání řídících znaků pro datový režim.

4.4. Intervalové typy datových a časových údajů

`tWeekYear = 1..59;`
Intervalový typ **tWeelYear** vymezuje číslo týdne v roce.

`tDayWeek = 1.. 7;`
Intervalový typ **tDayWeek** vymezuje číslo dne v týdnu.

`tYear = 0..99;`
Intervalový typ **tYear** vymezuje poslední dvojčíslí roku.

`tMonth = 1..12;`
Intervalový typ **tMonth** vymezuje číslo měsíce.

`tDay = 1..31;`
Intervalový typ **tDay** vymezuje číselnou hodnotu dne v měsíci.

`tHour = 0..23;`
Intervalový typ **tHour** vymezuje údaj hodin.

`tMinute = 0..59;`
Intervalový typ **tMinute** vymezuje údaj minut.

`tSecond = 0..59;`
Intervalový typ **tSecond** vymezuje údaj sekund.

4.5. Struktury přijímacích a vysílacích bufferů

Poznámka: V této kapitole a i v dalších kapitolách jsou uvedeny některé zkratky, které jsou zpravidla součástí symbolického názvu položky bufferu. Tyto zkratky

mají následující významy:
RTC - jedná se o Register, Timer či Counter nebo se jedná o Real Time Clock (rozdíl vyplývá z kontextu)
IOF - jedná se Input, Output či Flag

```
pMaSendRecord = ^tMaSendRecord;
tMaSendRecord = record
  case Code : Byte of
    R_StsCpu0 ,
    R_StsCpu1 ,
    R_StsCpu2 ,
    R_StsCpu3 ,
    R_StsCpu4 ,
    R_StsCpu5 ,
    R_StsCpu6 ,
    R_StsCpu7 :
      ( );
    R_Register,
    R_Timer   ,
    R_Counter ,
    W_Register,
    W_Timer   ,
    W_Counter :
      (CountRTC  : 0..32;
       AddressRTC: word;
       DataRTC   : array[1..32]of longint;);
    R_Input   ,
    R_Output  ,
    R_Flag    ,
    W_Output  ,
    W_Flag    :
      (CountIOF  : 0..128;
       AddressIOF: word;
       DataIOF   : array[1..16]of byte;);
    R_DispReg :
      ( );
    R_RTC     :
      ( );
    W_RTC     :
      (WeekYear  : tWeekYear;
       DayWeek   : tDayWeek;
       Year      : tYear;
       Month     : tMonth;
       Day       : tDay;
       Hour      : tHour;
       Minute    : tMinute;
       Second    : tSecond;);
end;
```

TMaSendRecord je typ variantního záznamu, který svou strukturou odpovídá datům protokolu SBus posílaným Master stanicí do stanice Slave, přičemž je zbaven nadbytečností, které jsou při vysílání doplněny. Položka **Code** určuje kód (typ) zasílané zprávy. Při čtení nebo zápisu registrů (Registers), časovačů (Timers) nebo čítačů (Counters) se používá položka **CountRTC**, která udává počet čtených či zapisovaných údajů (longintů), dále položka **AddressRTC**, která určuje absolutní adresu pro čtení či zápis parametrů, a pokud se jedná o zápis parametrů, používá se dále položka **DataRTC**, která obsahuje hodnoty zapisovaných dat. Při čtení nebo zápisu vstupů (Inputs), výstupů (Outputs) nebo příznaků (Flags) se používá položka **CountIOF**, která udává počet čtených údajů (bitů), dále položka **AddressIOF**, která určuje absolutní adresu pro čtení či zápis parametrů, a pokud se jedná o zápis parametrů, používá se

dále položka **DataIOF**, která obsahuje hodnoty zapisovaných dat. Při zápisu reálného času (Real Time Clock) se používají položky **WeekYear** (týden v roce), **DayWeek** (den v týdnu), **Year** (poslední dvojčíslí roku), **Month** (měsíc v roce), **Day** (den v měsíci), **Hour** (hodiny), **Minute** (minuty) a **Second** (sekundy).

```
pMaRecRecord = ^tMaRecRecord;
tMaRecRecord = record
  case Code     : Byte of
    R_StsCpu0 ,
    R_StsCpu1 ,
    R_StsCpu2 ,
    R_StsCpu3 ,
    R_StsCpu4 ,
    R_StsCpu5 ,
    R_StsCpu6 ,
    R_StsCpu7 :
      (Status   : Char;);
    W_Register,
    W_Timer   ,
    W_Counter ,
    W_Output  ,
    W_Flag    ,
    W_RTC     :
      (AckNack  : byte;);
    R_Register,
    R_Timer   ,
    R_Counter :
      (CountRTC : 0..32;
       Dummy1_2 : array[1..2]of byte;
       DataRTC  : array[1..32]of longint;);
    R_Input   ,
    R_Output  ,
    R_Flag    :
      (CountIOF : 0..128;
       DataIOF  : array[1..16] of byte;);
    R_DispReg :
      (Dummy2_3 : array[1..3]of byte;
       DispReg  : longint;);
    R_RTC     :
      (WeekYear : tWeekYear;
       DayWeek  : tDayWeek;
       Year     : tYear;
       Month    : tMonth;
       Day      : tDay;
       Hour     : tHour;
       Minute   : tMinute;
       Second   : tSecond;);
end;
```

TMaRecRecord je typ variantního záznamu, který svou strukturou odpovídá datům protokolu SBus přijímaným Master stanicí ze stanice Slave, přičemž je zbaven nadbytečností, které jsou při příjmu odstraněny. Položka **Code** určuje kód (typ) přijaté zprávy. Při čtení statusu některého z procesorů se používá položka **Status**, která udává stav daného procesoru ("S" - stopped, "H" - halted, "R" - running, "C" - conditional running, "D" - disconnected). Při zápisu registrů (Registers), časovačů (Timers), čítačů (Counters), výstupů (Outputs), příznaků (Flags) nebo reálného času (Real Time Clock) se používá položka **AckOrNack**, která udává hodnotu přijatého znaku, jako odpovědi na zápis (ACK - zápis proběhl v pořádku, NAK - zápis nelze provést, jiná hodnota - neidentifikovatelná chyba). Při čtení registrů (Registers), časovačů

(Timers) nebo čítačů (Counters) se používá položka **CountRTC**, která udává počet čtených údajů (longintů) a dále položka **DataRTC**, která obsahuje hodnoty čtených dat. Položka **Dummy1_2** nemá žádný zvláštní význam, v záznamu je definována pouze pro zarovnání pole DataRTC na adresu dělitelnou čtyřma. Při čtení vstupů (Inputs), výstupů (Outputs) nebo příznaků (Flags) se používá položka **CountIOF**, která udává počet čtených údajů (bitů) a dále položka **DataIOF**, která obsahuje hodnoty čtených dat. Při čtení registru displeje (Display Register) se používá položka **DispReg**. Položka **Dummy2_3** nemá žádný zvláštní význam, v záznamu je definována pouze pro zarovnání položky DispReg na adresu dělitelnou čtyřma. Při čtení reálného času (Real Time Clock) se používají položky **WeekYear** (týden v roce), **DayWeek** (den v týdnu), **Year** (poslední dvojčíslí roku), **Month** (měsíc v roce), **Day** (den v měsíci), **Hour** (hodiny), **Minute** (minuty) a **Second** (sekundy).

```
pSlSendRecord = ^tSlSendRecord;
tSlSendRecord = tMaRecRecord;
```

TSlSendRecord je typ, který svou strukturou odpovídá datům protokolu SBus vysílaným Slave stanicí do nadřízeného systému (Master stanice). Svou strukturou je shodný s typem pro strukturu přijímacího bufferu stanice Master.

```
pSlRecRecord  = ^tSlRecRecord;
tSlRecRecord  = tMaSendRecord;
```

TSlRecRecord je typ, který svou strukturou odpovídá datům protokolu SBus přijímaným Slave stanicí z nadřízeného systému (Master stanice). Svou strukturou je shodný s typem pro strukturu vysílacího bufferu stanice Master.

5. Objekty

5.1. tChnSBus

5.1.1. Položky

```
CH_RTick   : Boolean;
```

Položka **CH_RTick** označuje, že je vykonávaná činnost přijímacího automatu. Tato poloožka se používá pro ladění.

```
CH_SBuff   : Pointer;
```

Položka **CH_SBuff** definuje ukazatel na vysílací buffer.

```
CH_MSBuff  : Word;
```

Položka **CH_MSBuff** definuje délku vysílacího bufferu.

```
CH_LRMess    : Word;
```

Položka **CH_LRMess** definuje délku přijímané zprávy.

```
CH_RCcitt    : tccittW41;
```

Položka **CH_RCcitt** se používá pro počítání kontrolního CRC přijímače.

```
CH_SCcitt    : tccittW41;
```

Položka **CH_SCcitt** se používá pro počítání kontrolního CRC vysílače.

```
CH_Master  : Boolean;
```
Položka **CH_Master** definuje, je-li stanice zapojena v síti jako nadřazená jednotka (Master) nebo jako podřízená jednotka (Slave).

```
CH_DataMode: Boolean;
```
Položka **CH_DataMode** určuje používání datového režimu protokolu.

```
CH_DLE       : Boolean;
```
Položka **CH_DLE** určuje přijetí prvního řídícího znaku DLE protokolu. Je využívána pouze v datovém režimu.

5.1.2. Metody

5.1.2.1. Init konstruktor

```
constructor Init;
```
Konstruktor **Init** slouží k vytvoření a inicializaci instance komunikačního objektu. Ve svém těle nejprve zavolá zděděný konstruktor **Init** (inherited 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_RTick    := false;
CH_SBuff    := nil;
CH_MSBuff   := 0;
CH_LRMess   := 0;
CH_Master   := true;
CH_DataMode := false;
CH_DLE      := false;
```

5.1.2.2. ChInitParam konstruktor

```
constructor ChInitParam(const S: tParamStr);
```
Konstruktor **ChInitParam** slouží ke zkrácenému vytvoření instance komunikačního objektu s definovaným nastavením parametrů kanálu. Ve svém těle nejprve volá konstruktor **Init** a poté metodu **ChSetParam**.

5.1.2.3. Done destruktor

```
destructor  Done;
```
Destruktor **Done** slouží ke zrušení instance komunikačního objektu. Pokud je v paměti alokován vysílací buffer, bude odstraněn a poté je zavolán zděděný destruktor **Done** z rodičovského objektu tChnVirt (inherited Done).

5.1.2.4. ChSetOneParam funkce

```
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 tChnSBus dekóduje tyto parametry:

MAS=MASTER / SLAVE

Parametrem **MAS** ("Master or Slave") se určuje, zda je jednotka v komunikační síti zapojena jako Master (nadřízená) nebo jako Slave (podřízená).

DAT=ON / OFF

Parametrem **DAT** ("Using of Data Mode") se určí příznak používání datového režimu protokolu (On - používat datový režim, Off - nepoužívat datový režim).

LSB=Size

Parametrem **LSB** ("Length of Send Buffer") je alokován vysílací buffer **CH_MSBuff** dané velikosti Size. Size může nabývat hodnot <1..32750>..

NOD=Node

Parametrem **NOD** ("Node") se určuje číslo (adresa) stanice **CH_Node** v komunikační síti. Node může nabývat hodnot <0..255>.

DNO=DNode

Parametrem **DNO** ("Destination Node") se určuje číslo (adresa) stanice **CH_DNode** v komunikační síti, které budou zprávy určeny. Tuto položku je možno také definovat prostřednictvím metody **ChDestNode**. DNode může nabývat hodnot <0..255>.

5.1.2.5. 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**.

5.1.2.6. 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 v protokolu SBus.

5.1.2.7. 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.

5.1.2.8. ChSend procedura

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

Metoda **ChSend** způsobí započetí vysílání zprávy podle protokolu SBus na podkladu záznamu typu tMaSendRecord nebo tSlSendRecord, na který ukazuje parametr Buff. Parametr Len udává délku vysílacího bufferu pro vysílání. Tento parametr je nutno správně zadat v případě zprávy s daty parametrů (registrů, časovačů, čítačů, výstupů a příznaků) (viz níže). V ostatních případech se tento parametr ignoruje, protože metoda je schopna si ho dopočíst sama podle nastavených položek

záznamu vysílacího bufferu. Před voláním této metody se musí záznam správně naplnit daty (viz níže).

5.1.2.9. 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.

5.1.2.10. ChReceive procedura

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

Metoda **ChReceive** provede přijmutí celé zprávy a její uložení do přijímacího bufferu, který svou vnitřní strukturou odpovídá datům záznamu typu tMaRecRecord nebo tSlRecRecord a byl definován metodou **ChReceiveBuffer**. Metoda naplní buffer pouze patřičnými položkami typu tMaRecRecord nebo tSlRecRecord, úvodní a zakončovací řídicí znaky a kontrolní CRC ze zprávy metoda vyhodnotí a pro uživatele odstraní. Odpověď na zprávu, ať došla v pořádku nebo porušená, generuje uživatel sám pomocí metody **ChSend**.

5.1.2.11. ChReceiveFlush procedura

```
procedure ChReceiveFlush;
```

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

5.1.2.12. 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**.

5.1.2.13. ChReceiveTick procedura

```
procedure ChReceiveTick;
```

Metoda **ChReceiveTick** způsobí provedení jednoho či více kroků automatu přijímače. Je nutné ji periodicky volat během přijímání. Metoda **ChReceiveTick** je rovněž automaticky volána v metodě **ChReceiveReady**.

5.2. tAddChnSBus

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

5.2.1. Metody

5.2.1.1. ChInit funkce

```
function ChInit: pChnVirt;
```

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

6. Popis protokolu SBus

Jednotka aplikuje S-BUS Protokol podle DIN 66019, ISO 8073. Tento protokol používá firma SAIA pro komunikaci se svými řídícími systémy. Protokol SBus komunikuje ve třech režimech: datový režim (Data Mode), režim s paritním bitem (Parity Bit) a režim s vysíláním Break Interruptu (Break Interrupt).

6.1. Režimy protokolu SBus

6.1.1. Režim s vysíláním paritního bitu

Master stanice vysílá první znak zprávy s nastaveným paritním bitem a ostatní znaky vysílá s nulovým paritním bitem. Slave stanice vysílá všechny znaky zprávy s nulovým paritním bitem. Způsob tohoto vysílání nejlépe zajistí komunikační knihovna fyzické vrstvy ChnComPB definující objekt tChnComPB, jehož instanci zřetězíme v komunikačním kanálu za instanci objektu tChnSBus.

6.1.2. Režim s vysíláním Break Interruptu

Master stanice vysílá před prvním znakem zprávy tzv.: Break Interrupt (reset linky do logické úrovně 0 po stanovenou dobu). Slave stanice Break Interrupt před prvním znakem zprávy nevysílá. Způsob tohoto vysílání nejlépe zajistí komunikační knihovna fyzické vrstvy ChnComBR definující objekt tChnComBR, jehož instanci zřetězíme v komunikačním kanálu za instanci objektu tChnSBus.

6.1.3. Datový režim

Master i Slave stanice vysílají ke zjištění prvního znaku zprávy odlišný formát protokolu. Jako první se vyšle znak FS (Frame Synchronization Char) a poté se ostatní znaky vysílají následovně: 1. Pokud některý ze znaků má stejnou hodnotu jako FS, vyšlou se znaky DLE a $00. 2. Pokud některý ze znaků má stejnou hodnotu jako DLE, vyšlou se znaky DLE a $01.

6.2. Obezná struktura zpráv protokolu SBus

Master stanice vysílá zprávu pro čtení či zápis parametrů formátu:

DNode	Code	Data ...	CRC16

Slave stanice vysílá na žádost o čtení parametrů následující zprávu formátu:

Data ...	CRC16

Slave stanice vysílá na žádost o zápis parametrů jednu z následujících zpráv:

Zápis parametrů proběhl v pořádku:

ACK

Zápis parametrů nemohl být proveden:

NAK

Význam položek:
DNode - adresa Slave stanice v komunikační síti
Code - kód (typ) zprávy
Data - pole dat v závislosti na Code
CRC16 - kontrolní CRC zprávy CcittW41
ACK - krátké pozitivní potvrzení na zápis dat - pevná hodnota 06h
NAK - krátké negativní potvrzení na zápis dat - pevná hodnota 15h

6.3. Struktura konkrétních zpráv

V následujících odstavcích jsou popsány způsoby nastavování jednotlivých položek ve strukturách vysílacích bufferů pro konkrétní příklady zpráv protokolu SBus.

6.3.1. Zpráva pro čtení registrů, časovačů nebo čítačů

Master stanice vysílá zprávu s nastavenými položkami:

```
Code       := R_Register, R_Timer nebo R_Counter;
CountRTC   := ...
AddressRTC := ...
délka zprávy = 4
```

Slave stanice odpovídá zprávou s nastavenými položkami:

```
Code       := R_Register, R_Timer nebo R_Counter;
CountRTC   := ...
DataRTC    := ...
délka zprávy = 4+4*CountRTC
```

6.3.2. Zpráva pro čtení vstupů, výstupů nebo příznaků

Master stanice vysílá zprávu s nastavenými položkami:

```
Code       := R_Input, R_Output nebo R_Flag;
CountIOF   := ...
AddressIOF := ...
délka zprávy = 4
```

Slave stanice odpovídá zprávou s nastavenými položkami:

```
Code       := R_Input, R_Output nebo R_Flag;
CountIOF   := ...
DataIOF    := ...
```

```
délka zprávy = 2+počet bytů pro obsažení CountIOF bitů
```

6.3.3. Zpráva pro čtení Display registru

Master stanice vysílá zprávu s nastavenými položkami:

```
Code       := R_DispReg;
délka zprávy = 1
```

Slave stanice odpovídá zprávou s nastavenými položkami:

```
Code       := R_DispReg;
DispReg    := ...
délka zprávy = 8
```

6.3.4. Zpráva pro čtení reálného času

Master stanice vysílá zprávu s nastavenými položkami:

```
Code       := R_RTC;
délka zprávy = 1
```

Slave stanice odpovídá zprávou s nastavenými položkami:

```
Code       := R_RTC;
WeekYear   := ... např.  2;
DayWeek    := ... např.  5; {pátek}
Year       := ... např. 99;
Month      := ... např.  1;
Day        := ... např.  8;
Hour       := ... např. 12;
Minute     := ... např. 51;
Second     := ... např. 00;
délka zprávy = 9
```

6.3.5. Zpráva pro čtení statusu procesoru

Master stanice vysílá zprávu s nastavenými položkami:

```
Code       := R_StsCpu0 až R_StsCpu7;
délka zprávy = 1
```

Slave stanice odpovídá zprávou s nastavenými položkami:

```
Code       := R_StsCpu0 až R_StsCpu7;
Status     := ... např. 'R';
délka zprávy = 2
```

6.3.6. Zpráva pro zápis registrů, časovačů nebo čítačů

Master stanice vysílá zprávu s nastavenými položkami:

```
Code       := W_Register, W_Timer nebo W_Counter;
CountRTC   := ...
AddressRTC := ...
DataRTC    := ...
délka zprávy = 4+4*CountRTC
```

Slave stanice odpovídá zprávou s nastavenými položkami:

```
Code       := W_Register, W_Timer nebo W_Counter;
AckNack    := ACK nebo NAK;
délka zprávy = 2
```

6.3.7. Zpráva pro zápis výstupů nebo příznaků

Master stanice vysílá zprávu s nastavenými položkami:

```
Code       := W_Output nebo W_Flag;
CountIOF   := ...
AddressIOF := ...
DataIOF    := ...
délka zprávy = 4+počet bytů pro obsažení CountIOF bitů
```

Slave stanice odpovídá zprávou s nastavenými položkami:

```
Code       := W_Output nebo W_Flag;
AckNack    := ACK nebo NAK;
délka zprávy = 2
```

6.3.8. Zpráva pro zápis reálného času

Master stanice vysílá zprávu s nastavenými položkami:

```
Code       := W_RTC;
WeekYear   := ... např.  2;
DayWeek    := ... např.  5; {pátek}
Year       := ... např. 99;
Month      := ... např.  1;
Day        := ... např.  8;
Hour       := ... např. 12;
Minute     := ... např. 51;
Second     := ... např. 00;
délka zprávy = 9
```

Slave stanice odpovídá zprávou s nastavenými položkami:

```
Code       := W_RTC;
AckNack    := ACK nebo NAK;
délka zprávy = 2
```

7. Příklad

Následující příklad ukazuje způsob vyslání zprávy pro zápis deseti výstupů do zařízení PCD SAIA. Fyzický přenos se realizuje prostřednictvím knihovny ChnComPB.

```
uses
  ChnVirt,
  ChnCom,
  ChnSBus;

const
  ParamStr : tParamStr =
             'NAM=SBUS LSB=500 NOD=1 DNO=2 ' +
             'NAM=COMPB COM=1 IRQ=4 BD=9600 BIT=8 STOP=1 LRB=1000';
```

```
var
  Chn   : pChnVirt;
  SMess : pMaSendRecord;
  RMess : pMaRecRecord;
  LSMess: word;
  LRMess: word;

begin
  ...
  New(SMess);
  New(RMess);
  ...
  { vytvoření instance Chn }
  Chn:=ChnCollection^.ChNewInit(ChnSBus.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 (např. zpráva pro zápis 10-ti výstupů
      na hodnotu logická 1)}
    with SMess^ do
    begin
      Code       :=W_Output;
      CountIOF   := 10;
      AddressIOF := 0;
      DataIOF[1] := $ff;
      DataIOF[2] := $02;
      LSMess     := 6;
    end;
    { 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
      { dekódování správné odpovědi (např. odpověď ACK)}
      with RMess^ do
```

```
      begin
        if (Code    = W_Output) and
           (AckNack = ACK     ) then
          writeln('Ok')
        else
          writeln('Chyba');
      end;
    ...
    { 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.
```