ChnItch1

JEDNOTKA IMPLEMENTUJÍCÍ KOMUNIKAČNÍ PROTOKOL ČTEČKY IDENTIFIKAČNÍCH KLÍČŮ DALLAS

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 1.00

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: 14.04.2004
Datum posledního uložení dokumentu: 14.04.2004
(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	01.1X	We	14.04.2004	Vytvoření dokumentu.

1.2. Účel dokumentu

Tento dokument je popisem jednotky definující komunikační protokol ChnItch a komunikační automat, který je použit pro komunikaci s čtečkou Itch01 čipových identifikačních klíčů fi. Dallas.

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, s manuálem „ChnTypes“ a „ChnCom“.

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 ChnItch definuje formát protokolu používaného při komunikaci se čtečkou Itch01, čipových identifikačních klíčů fi. Dallas. Čtečka Itch01 (produkt fi. SofCon) obsahuje jednočipový procesor, který z klíče samostatně vyčítá jeho „family" kód a identifikační kód. Pokud čtečka načte platná data z klíče, zabalí je do protokolu ChnItch a pošle je po sériové komunikaci nadřízenému. Dokud je klíč ke čtečce přiložen, čtečka periodicky vyčítá kód klíče a posílá jej po komunikaci. Perioda děje je asi 1 sekunda. Přenos dat je směřován ze čtečky do nadřízeného procesoru, je pouze jednosměrný, bez vyžádání a potvrzení. Knihovna ChnItch implementuje pouze příjem dat ze čtečky. Knihovna obstarává zabezpečení dat a vyjímání nadbytečností, tak jak to tento protokol předepisuje. Fyzický přenos dat je zajištěn prostřednictvím nižší komunikační vrstvy, zpravidla knihovnou ChnCom.

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

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

Protože je objekt **tChnItch** 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. Konstanty a typy

```
cVerNo = např. $0101; { BCD formát }
cVer   = např. '01.01,17.11.2003';
```

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

```
cName = 'ITCH';
```

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

4.1. Řídící znaky protokolu

```
DLE   = $10;
SOH   = $01;
ETX   = $03;
```

Konstanta **DLE** definuje kód znaku pro identifikaci začátku nebo konce zprávy.
Konstanta **SOH** definuje kód znaku pro začátek zprávy.
Konstanta **ETX** definuje kód znaku pro konec zprávy.
Umístění znaků ve zprávě je popsáno dále v kapitole Struktura zpráv protokolu ChnItch.

4.2. Konstanty výsledků přijímacího automatu

```
res_ErrSum     = $21; { chybny CRC }
```

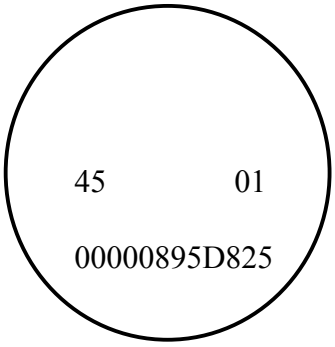
Knihovna zavádí nový chybový výsledek, který vrací metoda **ChReceiveResult** když nevyjde CRC kód přijaté zprávy ze čtečky čipových klíčů. Ostatní kódy jsou definovány v ChnVirtu.

4.3. Struktura přijímacího bufferu

```
tRecBuff  = array [0..6]of Byte;
```

tRecBuff [0] obsahuje "family" kód, zbytek bufferu obsahuje identifikační kód v opačném pořadí, než jak jej vysílá čipový klíč. Toto obrácené pořadí odpovídá evropskému způsobu čtení identifikačního kódu vyraženého na pouzdře klíče. Pokud přijatý identifikační kód v bufferu hexadecimálně vypíšeme od indexu = 0 do 6, dostaneme identický zápis kódu jako na pouzdře.

Pokud je kód na pouzdře vyražen v tomto uspořádání:



45 je CRC kód zprávy z čipového klíče. Tento kód kontroluje čtečka čipových klíčů a uživatel se k němu nedostane. 01 je „family" kód a je uložen v Ch_RecBuff [0]. Identifikační kód 00000895D825 je uložen od **Ch_RecBuff** [1] do Ch_RecBuff [6]. Obsah **Ch_RecBuff**, který je typu **tRecBuf** tedy bude tento:

```
Ch_RecBuff [0] = 01h
Ch_RecBuff [1] = 00h
Ch_RecBuff [2] = 00h
Ch_RecBuff [3] = 08h
Ch_RecBuff [4] = 95h
Ch_RecBuff [5] = D8h
Ch_RecBuff [6] = 25h
```

Příkazem

```
for i:=0 to 6
doWrite(ByteStrHex(Ch_RecBuff[i]))
```

dostaneme text: „0100000895D825".

5. Objekty

5.1. tChnITch

5.1.1. Položky

`CH_RCrc : tCrc8;`

Položka **CH_RCrc** slouží pro výpočet a následnou kontrolu zbytku z dělení zabezpečovacím polynomem Crc8 při příjmu dat od čtečky čipových klíčů.

`CH_RTick : Boolean;`

Položka **CH_RTick** signalizuje, že se knihovna ChnItch právě analyzuje přijatá data ze čtečky čipových klíčů. Je právě vykonávaná činnost přijímacího automatu.

`CH_RecBuff : tRecBuff;`

Položka **CH_RecBuff** je přijímací buffer knihovny pro zprávu od čtečky čipových klíčů. Metoda **ChReceive** jej pak přemístí do bufferu nastaveného metodou **ChReceiveBuffer**.

`CH_RecCount: Integer;`

Položka **CH_RecCount** je pracovní index do pole **CH_RecBuff**.

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 zavolá zděděný konstruktor **Init** (inherited Init) od rodičovského objektu tChnVirt a inicializuje položky objektu.

5.1.2.2. 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. Vzhledem k tomu, že u knihovny ChnItch není co nastavovat, volá tato metoda pouze **tChnItch.Init** a metodu **ChsetParam** svého přímého rodičovského objektu pro nastavení podřízené komunikační vrstvy.

5.1.2.3. Done destruktor

`destructor Done;`

Destruktor **Done** slouží ke zrušení instance komunikačního objektu a je volána zděděná metoda **Done** od přímého rodičovského objektu pro uzavření podřízené komunikační vrstvy.

5.1.2.4. ChConnect procedura

`procedure ChConnect; virtual;`

Metoda **ChConnect** zavolá zděděnou metodu od přímého rodičovského objektu pro provedení **ChConnect** od nižší vrstvy. Pokud nenastala žádná chyba, nastaví

automat přijímače **CH_RCtrl** do počátečního stavu pro příjem zprávy v protokolu ChnItch.

5.1.2.5. ChDisConnect procedura

```
procedure ChDisConnect; virtual;
```

Metoda **ChDisConnect** zavolá zděděnou metodu od přímého rodičovského objektu pro provedení **ChDisConnect** od nižší vrstvy. Pokud nenastala žádná chyba, nastaví automat přijímače **CH_RCtrl** do neaktivního stavu, aby se nepřijímaly žádné zprávy.

5.1.2.6. 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.

5.1.2.7. 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 **tRecRecord** a byl definován metodou **ChReceiveBuffer**. Metoda naplní buffer pouze patřičnými položkami typu tRecBuff, úvodní a zakončovací řídicí znaky a kontrolní součet ze zprávy metoda vyhodnotí a pro uživatele odstraní. V parametru Len vrátí velikost přijímacího bufferu.

5.1.2.8. ChReceiveFlush procedura

```
procedure ChReceiveFlush;
```

Metoda **ChReceiveFlush** způsobí volání předka a nastavení stavu automatu přijímače na počátek příjmu zpráv v protokolu ChnItch.

5.1.2.9. ChReceiveTick procedura

```
procedure ChReceiveTick;
```

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

5.2. tAddChnItch

Typ **tAddChnItch** je typem objektu, který slouží k definování prvku v seznamu správců komunikačních objektů (tzv. správce komunikačního objektu tChnICS v seznamu správců). Objekt **tAddChnIItch** 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 **tChnItch** a ukazatel na instanci tohoto objektu vrací jako svoji funkční hodnotu.

6. Struktura zpráv protokolu ChnItch

formát zprávy:

```
-----------------------------------------------------------------
| DLE | SOH | DeviceFamily | 6x IdentBytes | Crc8 | DLE | ETX |
-----------------------------------------------------------------
```

Význam jednotlivých položek je vysvětlen výše.

7. Příklad

Následující příklad ukazuje periodický příjem identifikačního kódu z čipového klíče. Fyzický přenos se realizuje prostřednictvím knihovny ChnCom.

```
program PokKlic;
{$define VerPC}  {verse pro pocitac PC s PCKitem}
uses
  ChnVirt,
  ChnCom,
  ChnItch1,
  NumToStr,
  Crt,
  uString;

var
  Chn     : PChnVirt; {ukazatel na komunikacni objekt}
  KlicBuff: tRecBuff; {prijimaci buffer}
  PomStr  : string;   {pomocny retezec pro precteni parametru kanalu}
  Delka   : word;     {delka prijate zpravy}
  i       : byte;     {pomocna pro FOR cykly}

const
  {parametry komunikacniho kanalu, jako fyzicka vrstva je zvolen
   standardni COM napr. na desce IOPCOM s bazovou adresou $300 ($2300
   na KitV40/386) } {$ifdef VerPC}
  ParamStr:tParamStr=
   'NAM=ITCH NAM=COM ADD=$310 IRQ=5 BD=4800 BIT=8 PAR=E STO=1 RB=300';
 {$else}
  ParamStr:tParamStr=
'NAM=ITCH NAM=COM ADD=$2310 IRQ=3 BD=4800 BIT=8 PAR=E STO=1 LRB=300';
 {$endif}

begin
      {vytvoreni instance Chn  a nastaveni parametru komunikace}
  Chn:=new(pChnITch,ChInitParam(ParamStr));
  with Chn^ do
  begin
    if ChResult<>res_Ok
    then WriteLn('inicializace chybna')
    else WriteLn('inicializace OK');
```

```
    PomStr:=ChGetParam('');
    WriteLn(PomStr);

    ChOpen;
    repeat
      if ChResult<>res_Ok then WriteLn('Chyba open');
    until ChReady=CHS_Open;
    if ChResult<>res_Ok then WriteLn('Chyba open');

    ChConnect;
    repeat
      if ChResult<>res_Ok then WriteLn('Chyba connect');
    until ChReady=CHS_Connect;
    if ChResult<>res_Ok then WriteLn('Chyba connect');

      {definovani mista, kam se ma prijata zprava ulozit}
    ChReceiveBuffer(@KlicBuff,SizeOf(KlicBuff));
    if ChReceiveResult<>res_Ok then WriteLn('Chyba nastaveni bufferu');

    repeat
      if ChReceiveReady=CHS_ReceiveReady {cekani na prijem zpravy}
      then
        begin
          ChReceive(Delka); {prijem zpravy}

          if ChReceiveResult<>res_Ok
          then WriteLn('Chyba prijmu')
          else
            begin
              Write('Family code: '+ByteStrHex(KlicBuff[0])+
                    ', Serial number: ');
              for i:=1 to 6
              do  Write(ByteStrHex(KlicBuff[i]));
              WriteLn;
            end;
        end
      else
        begin
          if ChReceiveResult<>res_Ok  {test na chybu prijmu}
          then WriteLn('Chyba prijmu');
        end;
    until KeyPressed;


    { ukonceni }
    ChDisConnect;
    repeat
      if ChResult<>res_Ok then WriteLn('Chyba disconnect');
    until ChReady=CHS_DisConnect;
    if ChResult<>res_Ok then WriteLn('Chyba disconnect');
    ChClose;
    repeat
      if ChResult<>res_Ok then WriteLn('Chyba close');
    until ChReady=CHS_Close;
    if ChResult<>res_Ok then WriteLn('Chyba close');
    { zruseni instance Chn }
    Dispose(Chn,Done);
  end;
end.
```