# Loader

## KNIHOVNY ZAVADĚČE APLIKACE

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 1.12

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: 19.08.2003
Datum posledního uložení dokumentu: 19.08.2003
(Datum vydání a posledního uložení dokumentu musí být stejné)

# Obsah :

1.O dokumentu 5
1.1. Revize dokumentu 5
1.2. Účel dokumentu 5
1.3. Rozsah platnosti 5
1.4. Související dokumenty 5
2.Termíny a definice 5
3.Úvod 6
4.Rozložení paměti 6
5.Popis spouštění a činnosti LOADER 8
5.1. Přizpůsobení programu 9
5.2. Kontrola aplikace 9
5.3. Spuštění aplikace 13
5.4. Komunikace s nadřízeným počítačem 13
5.5. Nahrávání nové aplikace 13
5.6. Vývojový diagram 14
6.Přizpůsobení knihoven LOADER Vaší aplikaci 18
7.Nastavení sdílených paměťových oblastí 20
8.Popis chybových kódů 21
9.Nahrání nové verze LOADER 22
9.1. Přizpůsobení požadavkům aplikace 22
9.2. Nahrání nové verze LOADER pomocí programu TheKing 23

# 1. O dokumentu

## 1.1. Revize dokumentu

| Verze dokumentu | Verze SW | Autor | Datum vydání | Popis změn |
|---|---|---|---|---|
| 1.00 | 2.XX | Hv | 19.12.2002 | První vydání |
| 1.10 | 2.XX | Hv | 05.02.2003 | Přesnější popis strLdrVer a strLdrName.<br>Oprava implicitní adresy _laLdrDesc. |
| 1.11 | 2.XX | Tu | 16.05.2003 | Úprava dokumentu dle ISO9000.<br>Doplněna konstanta strLoader. |
| 1.12 | 3.XX | Hv | 19.08.2003 | Úprava interface objektu z důvodu přechodu na rozšířenou standardní knihovnu Crc16. Dřívější verze používaly ExCrc16.<br>Upřesnění popisu práce s konstantou strLdrVer |
| | | | | |

## 1.2. Účel dokumentu

Tento dokument slouží jako popis programového balíku knihoven Loader používaného pro nahrávání nové verze aplikace pomocí programu TheKing.

## 1.3. Rozsah platnosti

Určen pro programátory a uživatele programového vybavení SofCon.

## 1.4. Související dokumenty

Pro čtení tohoto dokumentu není potřeba číst žádný další manuál, ale je potřeba orientovat se v používání programového vybavení SofCon.

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

Pomocí těchto knihoven můžete vytvořit program, s kterým můžete nahrát novou verzi aplikace do řídicího systému s Kit386 nebo KitV40 pomocí programu TheKing.

Vytvořený program můžete do paměti zapsat pomocí programátoru společně s BIOS nebo pomocí programu RTD.

Hlavním úkolem těchto knihoven je zajistit za všech okolností buď spuštění aplikace nebo v případě chyby spuštění časově omezeného navazování komunikace s vizualizačním programem. Pokud v nastaveném čase nedojde k navázání komunikace, spustí se aplikace. V opačném případě je možné nahrát novou verzi aplikace nebo zjistit typ chyby pomocí restart struktury.

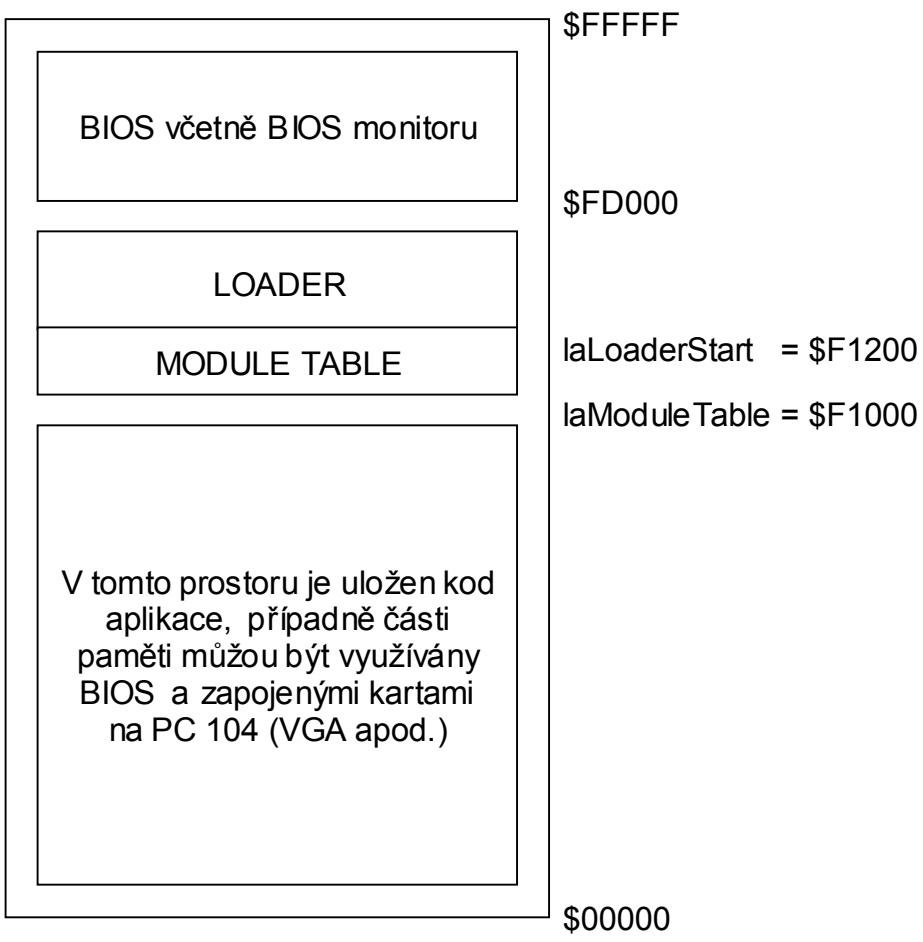
Dodávané knihovny LOADER, dále program LOADER, jsou schopny detekovat následující chyby aplikace – poškozená aplikace (přepis paměti CRC) a padání aplikace krátce po startu (např. zápis do paměti FLASH). Tyto chyby LOADER zachytí, zpracuje a při častém výskytu těchto chyb umožní nahrát novou verzi firmware.

# 4. Rozložení paměti

Pokud použijete standardní nastavení a program LOADER je určen pro řídicí systém s Kit386 s přenosem dat po komunikačním portu COM, můžete použít následující rozložení paměti. Protože na začátku programu se provádí kontrola velikosti tabulky modulů a umístění tabulky modulů, je potřeba zachovat pořadí oblastí v paměti řídicího systému, viz obrázek 1: Přiklad umístění programu LOADER v paměti řídicího systému.

Pozn. Při popisu začátku jednotlivých bloků paměti se používají lineární adresy.
Z těchto adres získáte segmentové adresy pomocí následujícího výpočtu:
Segment = (lineární adresa) shr 4
Offset = (lineární adresa ) and 4

Pozn. V případě použití přenosu dat po Ethernet s protokolem UDP je potřeba změnit rozložení paměti, protože komunikační knihovny pro Ethernet mají vyšší nároky na paměť.




**obrázek 1: Příklad umístění programu LOADER v paměti řídicího systému**

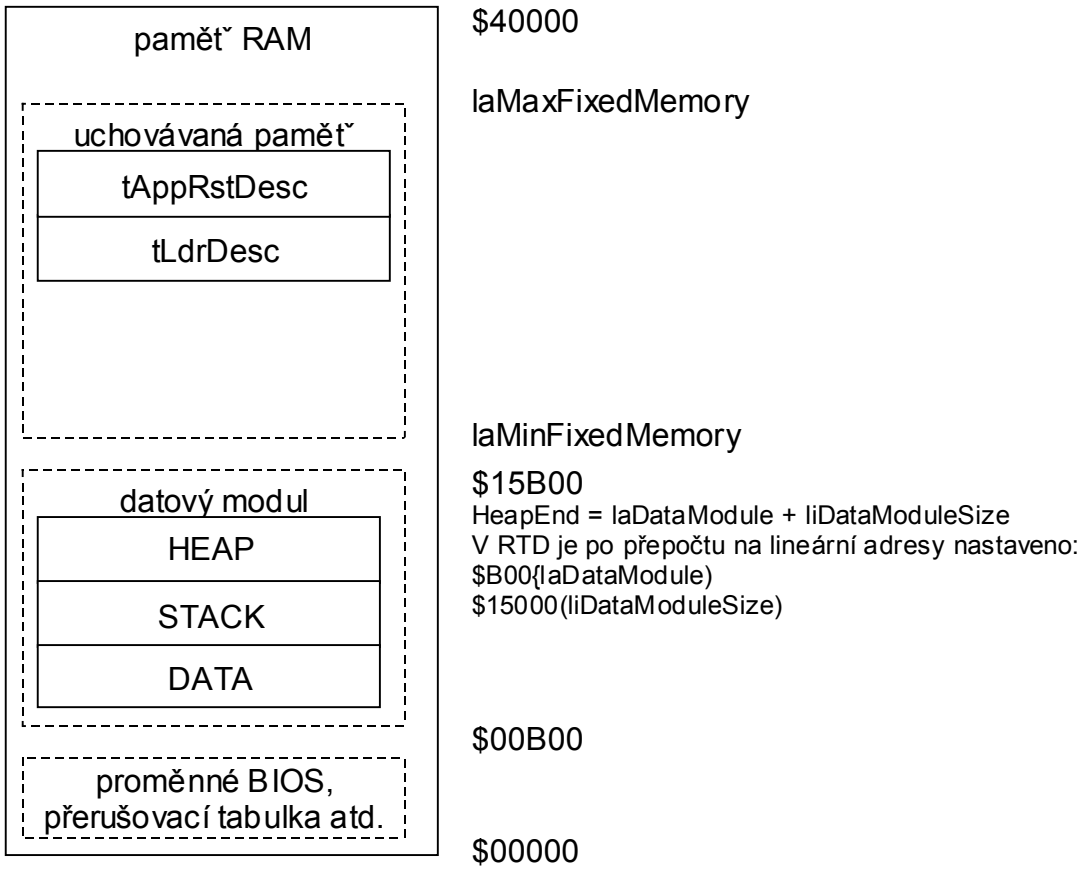
Pokud byste používali řídicí systém s KitV40, doporučujeme změnit případně se mění následující hodnoty:

začátek BIOS z $FD000 na **$FE000**

začátek programu LOADER z $F1200 na **$F2200 (proměnná laLoaderStart)**

začátek tabulky modulů z $F1000 na **$F2000 (proměnná laModuleTable)**

Protože program LOADER provádí kontroly rozmístění struktur, aby zabránil jejich vzájemnému překrytí musí být v paměti RAM rozloženy sdílené struktury následujícím způsobem. Zobrazené rozložení paměti předpokládá řídicí systém s minimálně 256kB paměti RAM a datový modul o velikosti $15000B s umístěním na lineární adrese $B00 (segmentově $B0).




**obrázek 2: Příklad rozložení sdílených struktur v paměti RAM**

# 5. Popis spouštění a činnosti LOADER

V následující části je popsán průběh spouštění (fáze) a činnosti programu LOADER po inicializaci řídicího systému Kit (tzn. zapnutí napájení případně reset). Některé fáze jsou poplatné doporučované konfiguraci a rozložení paměti, viz dodávaný příklad.

V dále popisovaných fází budou uváděny funkce, které odpovídají jejich volání ve vývojovém diagramu.

Průběh spouštění a činnosti programu lze rozdělit do těchto fází – přizpůsobení programu, kontrola aplikace, komunikace s nadřízeným systémem, zápis nové aplikace a spuštění aplikace.

## 5.1. Přizpůsobení programu

Do této fáze patří volání funkcí **LdrAdapt** a **fnAdaptToUserApp**.

Funkce **LdrAdapt** slouží pro nastavení parametrů LOADER – timeout, parametry komunikačního kanálu, identifikace LOADER (včetně verze) a případně signalizace běhu LOADER pomocí LED.

Funkce **fnAdaptToUserApp** je proměnná procedurálního typu, jejímž prostřednictvím se volá funkce nastavující umístění sdílených struktur v paměti. Tato funkce by měla být umístěna v jednotce, která je sdílena s aplikací, aby mohla být volána také v aplikaci.

## 5.2. Kontrola aplikace

Do této fáze patří volání funkcí **LdrInit**, **ReadModuleTable**, **CheckAplication** a **CheckExecTime.**

Funkce **LdrInit** provede nastavení proměnných obsahujících adresy sdílených struktur mezi aplikací a LOADER. Při jejich nastavování se použijí uživatelsky nastavitelné proměnné **_laAppRstDesc** a **_laLdrDesc**. Dále se nastaví defaultní adresa pro spouštění aplikace podle uživatelsky nastavitelné proměnné **laAppStart.**

Po jejich nastavení se provede kontrola verze struktury typu tAppRstDesc. Výsledkem této kontroly je nastavení příznaku, zda se smí s touto strukturou v programu LOADER pracovat. Tato struktura totiž obsahuje číslo verze v položce **wVer** a to je kontrolováno s nastavenou verzí, s kterou LOADER ještě smí pracovat. Rozhodnutí zda LOADER smí strukturu používat probíhá podle následujících pravidel.

Položka wVer je rozdělena na horní (HIGH) a spodní (LOW) byte.

- **horní byte** vyjadřuje verzi celé struktury a pokud se toto číslo liší, došlo k porušení kompatibility – změna velikosti nebo používání struktury, indexu případně počítadla chyb. V důsledku této změny přestane LOADER tuto strukturu používat, tzn. jím detekované chyby (začínají prefixem „L „) nejsou do ní zapisovány.
- **spodní byte** vyjadřuje pouze změnu používání pole Dummy vlastní aplikací. Protože LOADER toto pole nepoužívá, zapisuje do této struktury všechny detekované chyby.

V dalším kroku se provede kontrola rozmístění sdílených struktur v paměti. Při kontrole se použije následující podmínka, která předpokládá rozložení paměti, viz obrázek 2: Příklad rozložení sdílených struktur v paměti RAM.

```
if  ((_laAppRstDesc + sizeof(tAppRstDesc)) <= laMaxFixedMemory) and
    ((_laLdrDesc + sizeof(tLdrDesc)) <= _laAppRstDesc) and
    (laMinFixedMemory <= _laLdrDesc) and

    { kontrola nastaveni RTD - !!! lze pouze pro MCP !!!
      v pripade teto chyby se musi zkontrolovat laDataModule,
    liDataModuleSize a nastaveni RTD u datoveho modulu }
{$ifNdef MCP}
    (TRUE) then
{$else}
    (((LongInt(Seg(HeapEnd^)) shl 4) + (Ofs(HeapEnd^) and $F)) <=
```

```
   laMinFixedMemory) then
{$endif}
```

Pokud se zjistí překryv, zapíše se do položky wLastError ve struktuře typu tLdrDesc hodnota err_Alloc. V opačném případě se provede kontrola CRC této struktury. Jeli zjištěna chyba, struktura se zinicializuje a nastaví se bParForLdr na 1. Toto nastavení dále způsobí přechod fáze komunikace s nadřízeným počítačem, abychom se mohla nahrát nová verze LOADER případně aplikace kompatibilní se strukturou tLdrDesc.

Deklarace sdílených struktur ze souboru xDesc.pas.

```
pAppRstDesc = ^tAppRstDesc;
tAppRstDesc = record
  wVer            : Word;      { verze struktury (tato polozka MUSI
                                 byt prvni) }
  bIxWrittenAt    : Byte;      { index na posledni zapsanou chybu v
                                 poli ErrDescArray }
  ErrDescArray    : tArrRst;   { pole se strukturou popisujici chyby
                                 - definovana v XTking_1 }
  wAppVer         : Word;      { verze aplikace, ktera naposled
                                 zapisovala do teto struktury }
  wRunErrCnt      : Word;      { citac obslouzenych Runtime Error
                                 (kompatibilita) }
  dwPfiCnt        : LongInt;   { citac vypadku napajeni }
  dwAppStartCnt   : LongInt;   { citac startu aplikace }
  Dummy           : array[1..64] of byte;
                                 { rezervovano pro budouci pouziti }
end;
```

Struktura slouží pro ukládání chybových stavů při ukončování a startu aplikace. LOADER při zápisu nové chyby provádí hledání předchozí chyby v ErrDescArray (BUG, CRC, SUM případně RTE bez kontroly kódu chyby) a pokud tuto chybu nalezne, provede aktualizaci všech položek, tzn. adresu a kód chyby, datum a čas. Dále v textovém popisu zvýší počítadlo. V opačném případě tuto chybu ve struktuře vytvoří.

```
pLdrDesc = ^tLdrDesc;
TLdrDesc = record
  wVer            : Word;      { verze struktury (tato polozka MUSI
                                 byt prvni) }
  wLastError      : Word;      { kod posledni chyby v LOADER }
                               { hodnota se necha precist pomoci
                               cteni pameti v RTD, viz laModuleTable
                               }
  dwLastExecTime  : LongInt;   { cas posledniho spusteni aplikace }
  bLastExecNo     : Byte;      { pocet spusteni aplikace }
                               { Citac se vynuluje pri navazani
                               komunikace nebo pri spusteni aplikace
                               po zadanem case, wExecTime}
  bParForApp      : Byte;      { parametr pro aplikaci
                                  0       - bezny start aplikace
                                  1       - aplikace je poprve
                                     spustena po jejim zapisu do
                                     pameti  (muze byt stejna verze
                                     jako predchozi)
                                  ostatni - rezervovano }
  bParForLdr      : Byte;      { parametr pro LOADER
                                  0       - bezny start LOADER
                                  1       - cekani na komunikaci
                                  ostatni - rezervovano }
  wCheckSumCnt    : Word;      { pocet spusteni aplikace s nejmene
                               jednim poskozenym modulem, tj. pokud
                               je poskozen nejaky modul aplikace,
                               citac se zvysi o 1 }
  wBugCnt         : Word;      { pocet detekovanych restartu aplikace
                               pomoci sledovani poctu restartu v
                               zadanem case (nejcastejsi pricinou
                               teto chyby bude zapis do pameti FLASH)
                               }
  wTableCrcCnt    : Word;      { pocet spusteni aplikace s poskozenou
                               tabulkou popisujici moduly aplikace.
                               (Tabulka se vytvori pri zapisu nove
                               aplikace.) }
  wCrc            : Word;      { CRC cele struktury }
end;
```

Struktura LOADER. Verze LOADER je nastavena při každé inicializaci a mazání této struktury. S číslem této verze pracuje aplikace, která ho musí správně vyhodnocovat.

Funkce **ReadModuleTable** nejdříve nastaví proměnnou obsahující adresu tabulky modulů pomocí uživatelsky nastavitelné proměnné **laModuleTable**. Poté se provede kontrola překrytí tabulky modulů a programu LOADER. Pokud je zjištěna chyba, uloží se do položky wLastError ve struktuře tLdrDesc hodnota ldrerr_ModuleTable. Pak se zkontroluje CRC tabulky modulů a v případě chyby se do pole ErrDescArray zapíše chyba „**L CRC**“ a vrátí se FALSE. V opačném případě se vrací TRUE.

Po ukončení funkce **ReadModuleTable** se při návratové hodnotě FALSE přejde do fáze komunikace s nadřízeným počítačem. V opačném případě se otestuje položka **bParForLdr** ve struktuře tLdrDesc. Pokud je její hodnota **1**, přechází se do fáze komunikace s nadřízeným počítačem. Pokud je její hodnota různá od **1**, pokračuje se v kontrole aplikace voláním funkcí **CheckApplication** a **CheckExecTime**.

Funkce **CheckApplication** provede kontrolu aplikace podle tabulky modulů. (Pozn. Tabulka modulů se zapíše do paměti FLASH při ukončení nahrávání aplikace pomocí programu TheKing. Pokud nahrávání nebylo ukončeno, je v paměti uložena předchozí podoba tabulky, což může, ale nemusí vést k signalizaci chyby aplikace.) Při kontrole aplikace se zkontroluje, zda CheckSum u modulů s příznakem moduleattr_CheckSum je 0. Pokud je CheckSum neplatný, přeruší se další kontroly a přechází se na fázi komunikace s nadřízeným počítačem. Pokud je CheckSum platný, tak se u spustitelných souborů zkontroluje počáteční hodnota modulu na $AA55. Pokud tato hodnota na začátku modulu není, přeruší se další kontroly a přechází se na fázi komunikace s nadřízeným počítačem. Pokud je u modulu zjištěn příznak moduleattr_Start, nastaví se tato adresa do proměnné obsahující adresu startu aplikace. Tzn. přepíše se defaultní nastavení startovací adresy aplikace. Po bezchybné kontrole všech modulů se volá **CheckExecTime**. Pokud se při kontrole aplikace zjistí chyba, CheckSum nebo počáteční hodnota modulu, zapíše se do pole ErrDescArray chyba **„L SUM“**.

Pozn. Při nahrávání aplikace do řídicího systému pomocí programu TheKing je možné příznaky modulů změnit. Podrobný popis nastavování těchto příznaků najdete v manuálu programu TheKing.

Deklarace struktur popisující aplikaci ze souboru xDesc.pas.

```
pModuleDesc = ^tModuleDesc;
tModuleDesc = record
   wBegAddr        : Word;      { segmentova adresa zacatku modulu }
   dwLen           : LongInt;   { velikost modulu - pocet BYTE }
   setFlags        : TModuleAttrBits;
                                { priznaky modulu }
end;
```

Struktura popisující jeden modul.

```
pModuleTable = ^tModuleTable;
tModuleTable = record
   bCount          : Byte;      { pocet platnych polozek v poli
                                ModuleDesc }
   ModuleDesc      : array [0..cMaxModuleTable-1] of tModuleDesc;
                                { pole modulu }
   wCrc            : Word;      { CRC }
end;
```

Struktura tabulky modulů popisující aplikaci a skládající se z několika modulů.

Funkce **CheckExecTime** provede kontrolu spouštění aplikace. Pokud je aplikace znovu spuštěna v uživatelsky definovatelném čase (**wExecTime**), zvýší se počítadlo spouštění aplikace. Pokud toto počítadlo překročí uživatelsky definovatelný počet (**wExecNo**), zapíše se chyba **„L BUG“** do pole ErrDescArray. Poté se přechází se do fáze komunikace s nadřízeným počítačem. V opačném případě se po návratu z funkce přechází do fáze spuštění aplikace.

## 5.3. Spuštění aplikace

Pokud je nastavena proměnná se spouštěcí adresou aplikace různou od NIL, provede se spuštění aplikace. V opačném případě se pokračuje buď do fáze komunikace s nadřízeným počítačem nebo se řídicí systém inicializuje, pokud této fázi předcházela komunikace s nadřízeným počítačem.

## 5.4. Komunikace s nadřízeným počítačem

Tato fáze je časově omezena dle uživatelsky nastavitelné proměnné **liTimeoutToAppRun**. Pokud do této doby nedojde k navázání komunikace s nadřízeným počítačem přechází se do fáze spuštění aplikace.

V této fázi se nejdříve provede detekce pamětí FLASH. Pokud dojde k chybě, uloží se do položky wLastError ve struktuře tLdrDesc hodnota ldrerr_FlashId. Dále se pokračuje otevřením komunikačního kanálu. Pokud se kanál nepodaří otevřít, mohou se uložit do položky wLastError ve struktuře tLdrDesc následující hodnoty ldrerr_COMPar, ldrerr_Open, ldrerr_RecBuff a ldrerr_Connect. Pokud se komunikační kanál nepodaří otevřít, přechází se do fáze spuštění aplikace. Pokud je LOADER spuštěn na KitV40, tak se po otevření komunikačního kanálu provede korekce časovače. Dále se nastaví vlastní obsluha Watch-Dog, která hlídá komunikaci s nadřízeným počítačem. Pokud dojde k zastavení komunikace delší než uživatelsky definovatelný čas v proměnné **wWinComWD**, uloží se do položky wLastError ve struktuře tLdrDesc hodnota ldrerr_Freeze a provede se přechod do fáze spuštění aplikace. Nakonec této fáze se spustí čekání na komunikaci s nadřízeným počítačem. Pokud dojde k navázání komunikace, lze tuto fázi ukončit buď pomocí příkazu Restart v programu TheKing nebo pomocí reset. Z této fáze lze pomocí příkazu *„Prg Download“* nebo *„Automatic program download“* v programu TheKing přejít do fáze nahrávání nové aplikace. Pokud k navázání komunikace v nastaveném čase nedošlo, přechází se do fáze spuštění aplikace.

## 5.5. Nahrávání nové aplikace

V této fázi se pomocí programu TheKing zapisuje nová aplikace. Do paměti řídicího systému se nejdříve zapíše celá aplikace, případně všechny moduly nastavené ve vizualizaci, a poté se zapíše tabulka modulů. Pokud během nahrávání dojde k přerušení (výpadek napájení, reset nebo po stisku tlačítka ABORT), musí se celá aplikace nahrát znova, protože v paměti řídicího systému je neplatná tabulka modulů a aplikace je v nedefinovaném stavu.

## 5.6. Vývojový diagram

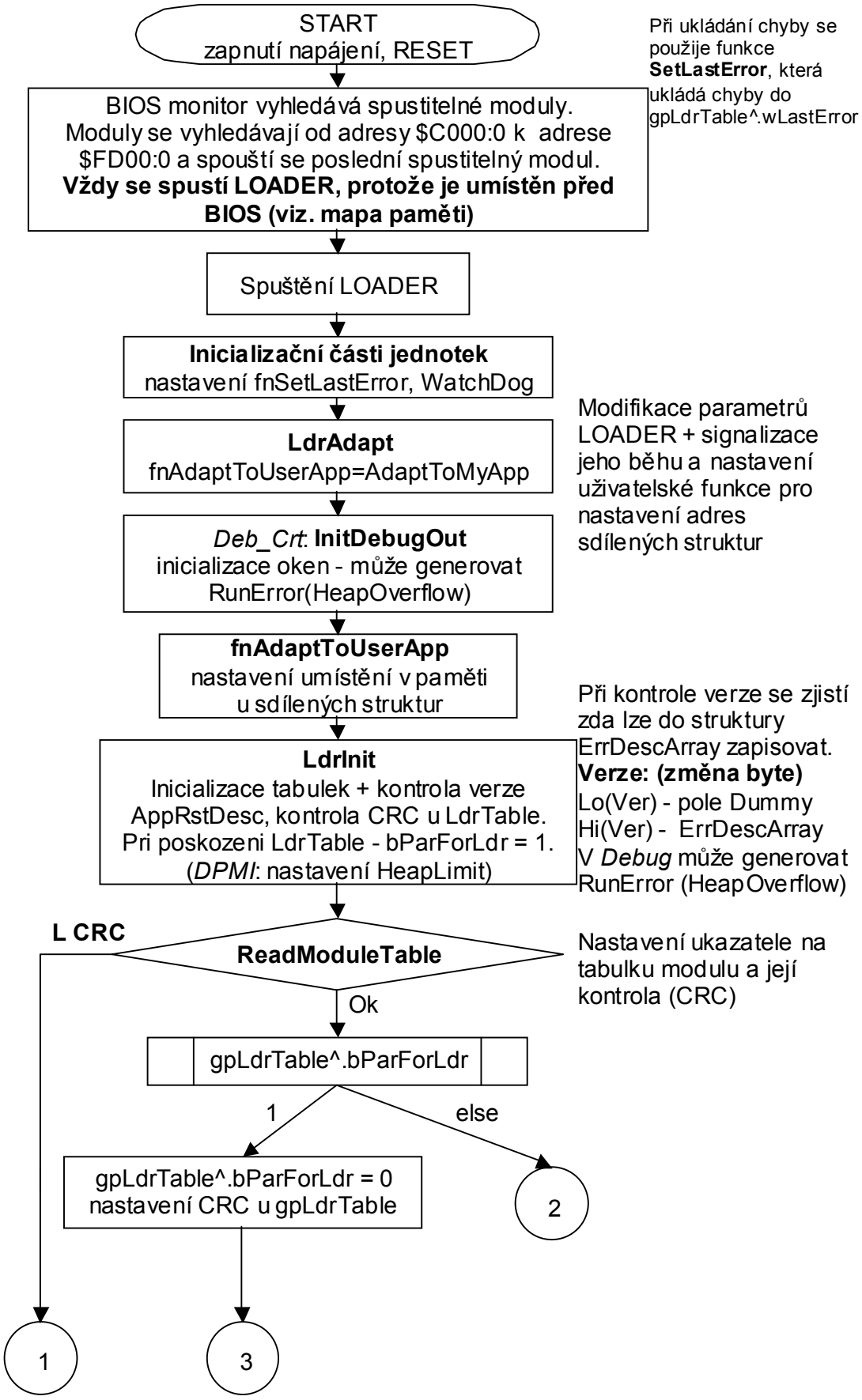
Pozn. Protože LOADER umožňuje pracovat v ladicím režimu s obrazovkou (direktiva Deb_Crt) funkce InitDebugOut.

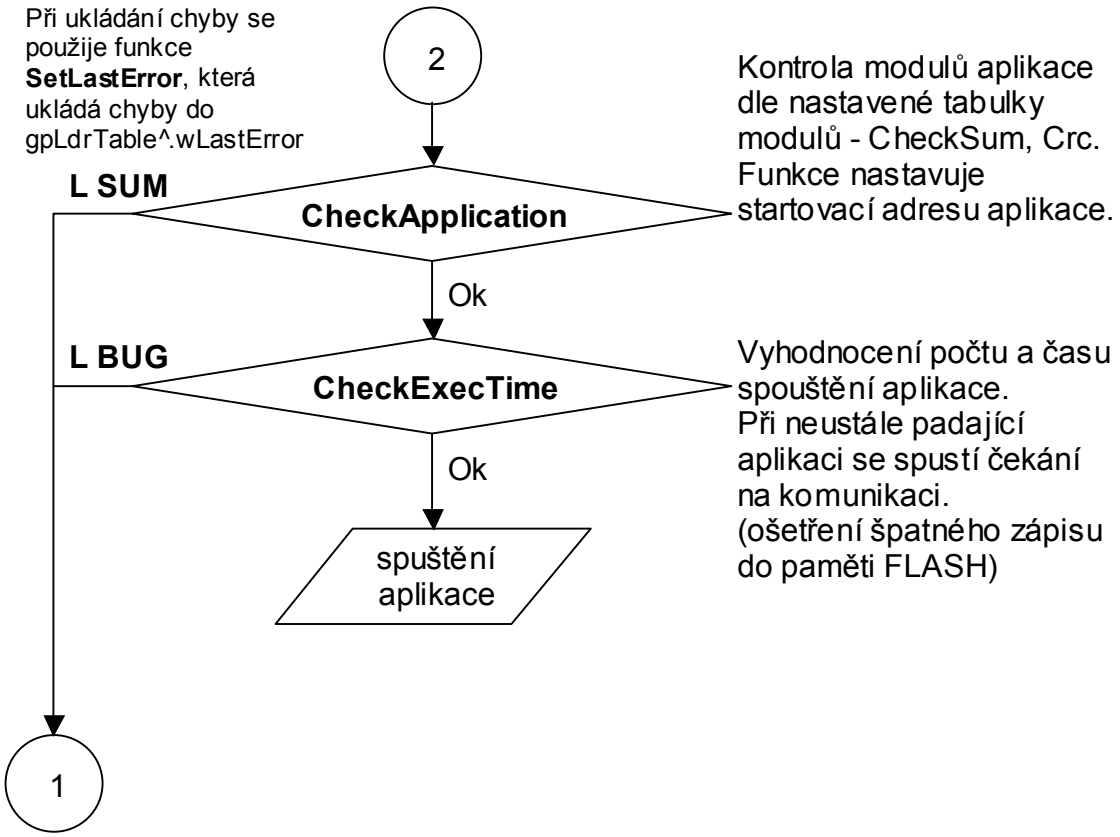
Pozn. Při běhu LOADER na PC a překladu pro chráněný režim se musí nastavit HeapLimit na 0.

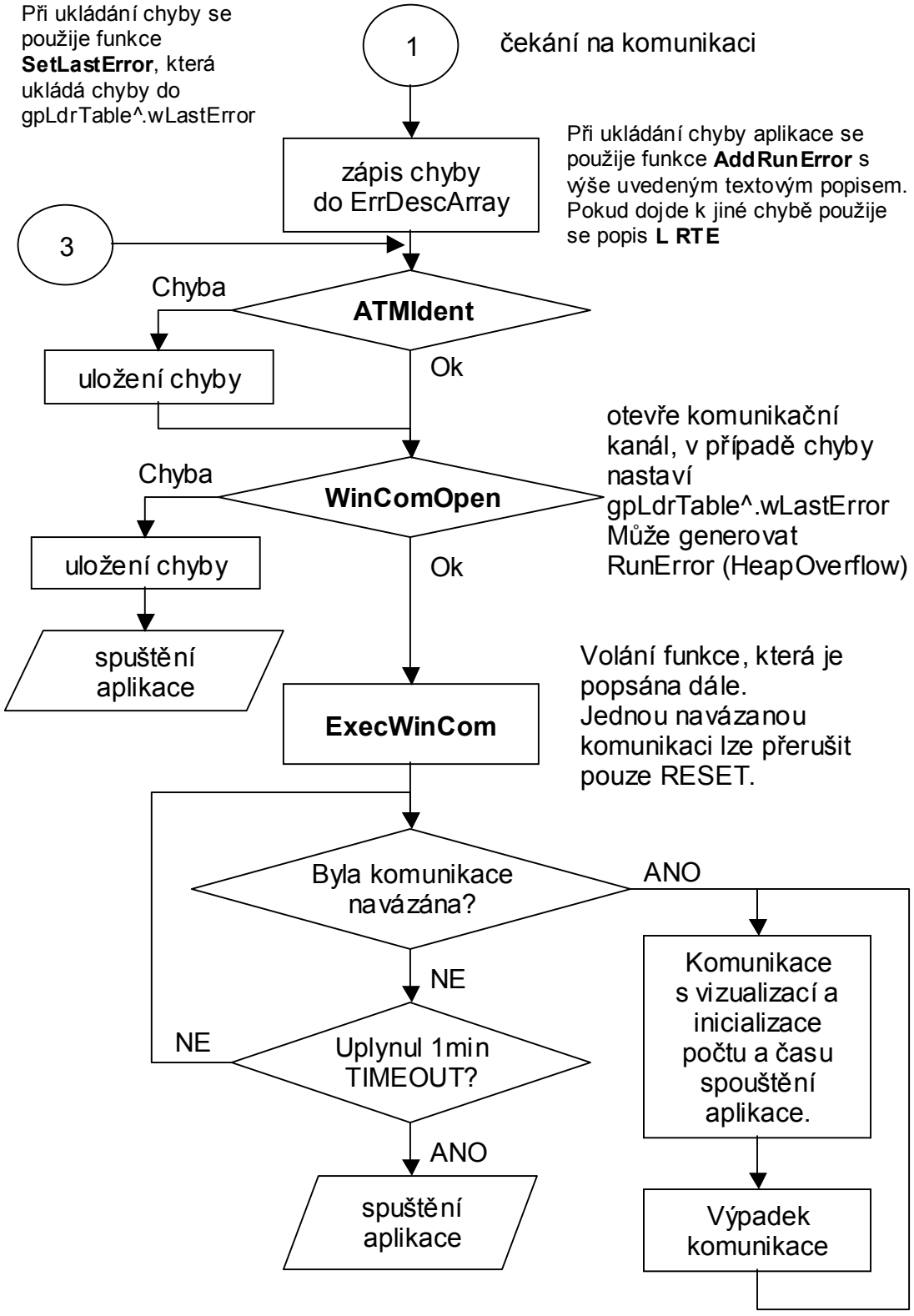
Pozn. Při spuštění LOADER na PC se sdílené struktury vytváří na HEAP a jsou automaticky zapisovány případně čteny ze souboru při jeho spuštění nebo ukončení. Proto při jejich vytváření může dojít k chybě RunError(Heap_Overflow).

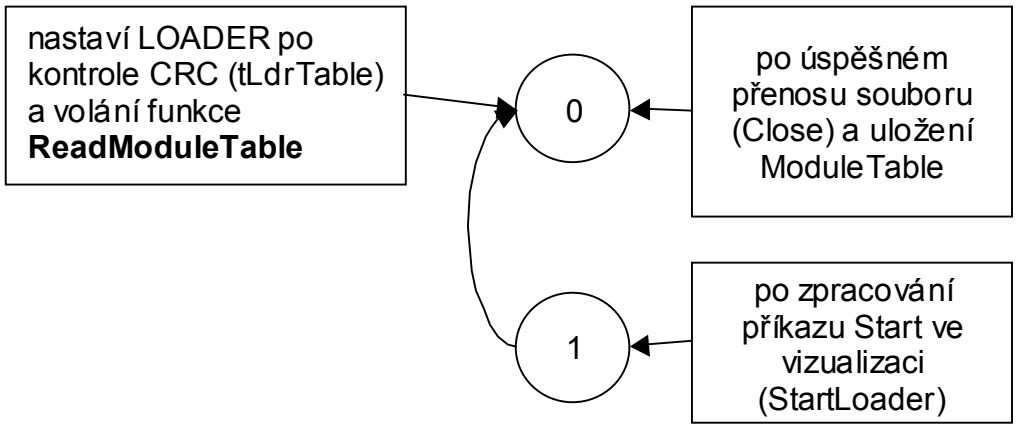
Pozn. Pokud při běhu LOADER dojde k vygenerování runtime chyby, je tato chyba uložena do ErrDescArray jako **„L RTE“**. Číslo kódu odpovídá poslední vygenerované chybě. Počítadlo těchto chyb je použito pro všechny runtime chyby.

Pozn. Za textovým popisem chyby **„L SUM“**, **„L CRC“**, **„L BUG“** a **„L RTE“** následuje počítadlo s max. hodnotou 9999. Po dosažení této hodnoty se přestane zvyšovat.

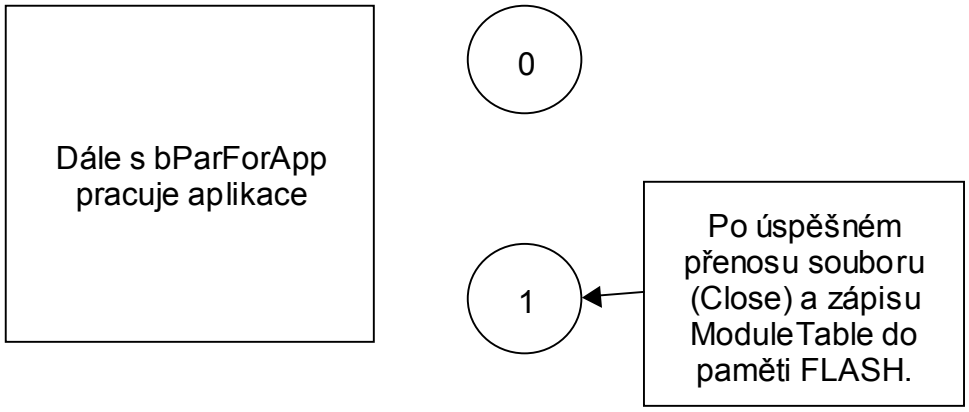

START
zapnutí napájení, RESET
BIOS monitor vyhledává spustitelné moduly.
Moduly se vyhledávají od adresy $C000:0 k adrese $FD00:0 a spouští se poslední spustitelný modul.
**Vždy se spustí LOADER, protože je umístěn před BIOS (viz. mapa paměti)**
Při ukládání chyby se použije funkce **SetLastError**, která ukládá chyby do gpLdrTable^.wLastError
Spuštění LOADER
**Inicializační části jednotek**
nastavení fnSetLastError, WatchDog
**LdrAdapt**
fnAdaptToUserApp=AdaptToMyApp
Modifikace parametrů LOADER + signalizace jeho běhu a nastavení uživatelské funkce pro nastavení adres sdílených struktur
*Deb_Crt*: **InitDebugOut**
inicializace oken - může generovat RunError(HeapOverflow)
**fnAdaptToUserApp**
nastavení umístění v paměti u sdílených struktur
Při kontrole verze se zjistí zda lze do struktury ErrDescArray zapisovat.
**Verze: (změna byte)**
Lo(Ver) - pole Dummy
Hi(Ver) - ErrDescArray
V *Debug* může generovat RunError (HeapOverflow)
**LdrInit**
Inicializace tabulek + kontrola verze AppRstDesc, kontrola CRC u LdrTable.
Pri poskozeni LdrTable - bParForLdr = 1.
(*DPMI*: nastavení HeapLimit)
**L CRC**
**ReadModuleTable**
Nastavení ukazatele na tabulku modulu a její kontrola (CRC)
Ok
gpLdrTable^.bParForLdr
1
else
gpLdrTable^.bParForLdr = 0
nastavení CRC u gpLdrTable
2
1
3


Při ukládání chyby se použije funkce **SetLastError**, která ukládá chyby do gpLdrTable^.wLastError
2
L SUM
CheckApplication
Kontrola modulů aplikace dle nastavené tabulky modulů - CheckSum, Crc. Funkce nastavuje startovací adresu aplikace.
Ok
L BUG
CheckExecTime
Vyhodnocení počtu a času spouštění aplikace. Při neustále padající aplikaci se spustí čekání na komunikaci. (ošetření špatného zápisu do paměti FLASH)
Ok
spuštění aplikace
1


Při ukládání chyby se použije funkce **SetLastError**, která ukládá chyby do gpLdrTable^.wLastError
1
čekání na komunikaci
zápis chyby do ErrDescArray
Při ukládání chyby aplikace se použije funkce **AddRunError** s výše uvedeným textovým popisem. Pokud dojde k jiné chybě použije se popis **L RTE**
3
Chyba
**ATMIdent**
Ok
uložení chyby
otevře komunikační kanál, v případě chyby nastaví gpLdrTable^.wLastError Může generovat RunError (HeapOverflow)
Chyba
**WinComOpen**
uložení chyby
Ok
spuštění aplikace
Volání funkce, která je popsána dále. Jednou navázanou komunikaci lze přerušit pouze RESET.
**ExecWinCom**
Byla komunikace navázána?
ANO
NE
Komunikace s vizualizací a inicializace počtu a času spouštění aplikace.
NE
Uplynul 1min TIMEOUT?
ANO
spuštění aplikace
Výpadek komunikace




**obrázek 3: Automat popisující nastavování položky gpLdrTable^.bParForLdr**




**obrázek 4: Automat popisující nastavování položky gpLdrTable^.bParForApp**

# 6. Přizpůsobení knihoven LOADER Vaší aplikaci

Knihovny LOADER, tzn. vytvořený program LOADER, můžete přizpůsobit požadavkům aplikace pomocí následujících proměnných při dodržení pořadí a rozmístění oblastí a struktur uvedených na předchozích obrázcích, viz obrázek 1: Přiklad umístění programu LOADER v paměti řídicího systému a obrázek 2: Příklad rozložení sdílených struktur v paměti RAM.

```
wExecTime          (defaultně 90s)
wExecNo            (defaultně 5)
```

Hodnoty těchto proměnných udávají, kolikrát se musí aplikace spustit resp. spadnout v zadaném čase, aby se spustila fáze komunikace s nadřízeným počítačem. Tyto hodnoty ošetřují padání aplikace, např. v důsledku zápisu do paměti FLASH.

`wWinComWD` `(defaultně 20, což odpovídá 55ms * 20 = 1100ms)`

Inicializační hodnota Watch-Dog. Tato hodnota znamená na jak dlouho se smí max. zastavit komunikace než se přejde do fáze spuštění aplikace.

`liTimeoutToAppRun` `(defaultně 60000ms)`

Čas, za který se přejde do fáze spuštění aplikace, pokud se nepodaří navázat komunikaci s nadřízeným počítačem.

`liTimeoutToReady` `(defaultně 10000ms)`

Čas, po který se při otevírání komunikačního kanálu čeká na otevření. Pokud se do této doby nepodaří komunikační kanál otevřít, vrací se FALSE a nastaví se chyba do položky wLastError ve struktuře tLdrDesc.

`liTimeoutToImmediateSend` `(defaultně 2500ms)`

Čas, po který se čeká na odvysílání potvrzení příkazů programu TheKing. Pokud se do tohoto času nepodaří potvrzení odeslat, pokračuje se dále.

`strCOM` `(defaultně prázdný)`

Do řetězce se nastaví parametry komunikačního kanálu. Podle zvoleného komunikačního kanálu se musí do části ***uses*** přidat patřičná komunikační knihovna. Popis práce s komunikačními knihovnami odpovídá zvyklostem při práci s komunikací, viz samostatné manuály komunikačních knihoven.

`strLdrVer` `(defaultně nastaveno $0000)`

Verze programu LOADER. Horní byte znamená číslo před desetinnou tečkou a spodní byte znamená číslo za desetinnou tečkou. Např. $0201 znamená 2.01.

Pozn. Tato inicializovaná konstanta je uložena v souboru xDesc a vyjadřuje verzi všech knihoven programového balíku LOADER. Protože tato hodnota sleduje vývoj aplikace, musí být nastavována na požadovanou hodnotu.

`strLdrName` `(defaultně dle verze vydávaných knihoven, „LOADER“)`

Protože tento řetězec je zobrazován v programu TheKing doporučujeme k popisu programu LOADER připojit verzi. K tomuto připojení verze doporučujeme použít automatický způsob generování tohoto řetězce, např.

```
strLdrName := 'LDR ver '+ByteStrDec(Hi(strLdrVer), 0)+'.';
if Lo(strLdrVer) < 10 then
  strLdrName:=strLdrName + '0' + ByteStrDec(Lo(strLdrVer), 0)
else
  strLdrName:=strLdrName + ByteStrDec(Lo(strLdrVer), 0);
```

# 7. Nastavení sdílených paměťových oblastí

Protože knihovny LOADER, tzn. vytvořený program LOADER, sdílí s aplikací některé struktury. Musí být jak v aplikaci tak v programu LOADER stejné umístění těchto struktur v paměti (tj. adresy). Proto doporučujeme provádět toto nastavování prostřednictvím samostatného souboru, např. uxDesc.pas. (Pozn. Prefix **u** jako user a **x** jako exchangeable(shared). Název souboru je zvolen Desc, protože vychází ze souboru xDesc.pas). Dále popisované proměnné jsou definovány v souboru xDesc.pas, který je opět společný pro aplikací i program LOADER.

`laFlashBase` `(defaultně $80000, případně se volá funkce GetBaseAddr z jednotky HwSyst)`

Proměnná obsahuje bázovou adresu umístění pamětí FLASH.

`laAppStart` `(defautlně $E0000)`

Proměnná obsahuje spouštěcí adresu aplikace. Tato adresa je dána lineární adresou modulu aplikace, který obsahuje jednotku - - LOADER - -. U takto zadané adresy se ignorují spodní 4bity. Tato adresa se použije, pokud je poškozena tabulka modulů.

`laModuleTable` `(defaultně $F1000)`

Proměnná určuje začátek tabulky modulů. Při jejím zadávání je potřeba respektovat pořadí, viz obrázek 2: Příklad rozložení sdílených struktur v paměti RAM.

`laLoaderStart` `(defaultně $F1200)`

Proměnná určuje začátek programu LOADER. Při jejím zadávání je potřeba respektovat pořadí, viz obrázek 2: Příklad rozložení sdílených struktur v paměti RAM.

`_laAppRstDesc` `(defaultně $3FE0D)`

Proměnná obsahuje adresu umístění sdílené struktury tAppRstDesc. Při jejím zadávání je potřeba respektovat pořadí, viz obrázek 2: Příklad rozložení sdílených struktur v paměti RAM.

`_laLdrDesc` `(defaultně $3FDFA)`

Proměnná obsahuje adresu umístění sdílené struktury tLdrDesc. Při jejím zadávání je potřeba respektovat pořadí, viz obrázek 2: Příklad rozložení sdílených struktur v paměti RAM.

`laDataModule` `(defaultně $B00)`

Proměnná obsahuje lineární adresu začátku datového modulu definovaného v programu RTD (vychází se ze segmentové adresy). Při jejím zadávání je potřeba respektovat pořadí, viz obrázek 2: Příklad rozložení sdílených struktur v paměti RAM.

`liDataModuleSize` `(defaultně $15000)`

Proměnná obsahuje velikost datového modulu definovaného v programu RTD. Při jejím zadávání je potřeba respektovat pořadí, viz obrázek 2: Příklad rozložení sdílených struktur v paměti RAM.

`laMaxFixedMemory` `(defaultně $40000)`

Proměnná obsahuje max. adresu paměti uchovávající data. Při jejím zadávání je potřeba respektovat pořadí, viz obrázek 2: Příklad rozložení sdílených struktur v paměti RAM.

`laMinFixedMemory` `(defaultně $15B00)`

Proměnná obsahuje min. adresu paměti uchovávající data. Při jejím zadávání je potřeba respektovat pořadí, viz obrázek 2: Příklad rozložení sdílených struktur v paměti RAM.

# 8. Popis chybových kódů

Chybové kódy jsou definovány v souboru xDesc.pas.

Chyby s prefixem err_ jsou používány programem LOADER a mohou být používány aplikací. Kódy lze zjistit při přečtení struktury tLdrDesc pomocí čtení paměti v RTD.

```
err_Ok              = 0;  { OK }
err_DPMI            = 1;  { nenastaven HeapLimit = 0 -> archivy }
err_Alloc           = 2;  { nenastaveny ukazatele - nebo prekryv
                            struktur }
```

Chyby s prefixem ldrerr_ jsou generovány pouze LOADER a mohou být přečteny současně se strukturou tLdrDesc.

```
ldrerr_SaveDesc     = 3;  { chyba pri ukladani struktur aplikace a
                            loader }
ldrerr_LoadDesc     = 4;  { chyba pri nacitani struktur aplikace a
                            loader }
ldrerr_TimeRes      = 5;  { pozadavek na hlidani padu aplikace byl
                            vetsi nez jeden den, tj. cExecTime >
                            86400 }
ldrerr_InvalidTime  = 6;  { neplatny cas v ErrDescArray }
ldrerr_ModuleTable  = 7;  { ModuleTable prekryva LOADER -
                            nastaveni aplikace }
_ldrerr_CheckSize   = 8;  { u ModuleTable se zmenila velikost ->
                            vypocty }
                            { od verze 2.00 - se nepouziva }
ldrerr_Freeze       = 9;  { zacykleni LOADER - watchdog }
ldrerr_FlashID      = 10; { pamet FLASH nelze identifikovat }
ldrerr_ComPar       = 11; { neplatny parametrizacni retezec
                            komunikacniho kanalu - chybi komunikacni
                            knihovna, neplatny parametr }
_ldrerr_ComPar      = 12; { neplatny parametrizacni retezec
                            komunikacniho kanalu - chybi komunikacni
                            knihovna, neplatny parametr }
                            { od verze 2.00 - se nepouziva }
ldrerr_Open         = 13; { nepodarilo se otevrit komunikacni
                            kanal }
ldrerr_RecBuff      = 14; { nepodarilo se pripojit buffer pro
                            prijem }
ldrerr_Connect      = 15; { nepodarilo se pripojit ke
```

```
                              komunikacnimu kanalu }
  ldrerr_RunApp       = 16; { neplatna startovaci adresa - NIL,
                              modul neni spustitelny, doslo k ukonceni
                              aplikace }
                              { pridano od verze 2.00 }
```

Textový popis chyb ukládaný do ErrDescArray. Těmto chybám se přidává prefix strLoader definovaný následovně:

```
  strSum:String[2] = 'L ';
  strSum:String[3] = 'SUM';    { CheckSum - modulu aplikace }
  strCrc:String[3] = 'CRC';    { CRC - ModuleTable }
  strBug:String[3] = 'BUG';    { Bug - padání aplikace - např. zápis
                               do FLASH }
  strRTE:String[3] = 'RTE';    { obecné chyby RunError }
```

Chyby strSum, strCrc a strBug mají následující kódy chyb.

```
  rt_SUM          = 40;        { v aplikaci je poškozený modul }
  rt_BUG          = 41;        { aplikace spadla v zadaném časovém
                               intervalu v zadaném počtu }
  rt_CRC          = 42;        { ModuleTable je poškozena (CRC) }
```

# 9. Nahrání nové verze LOADER

Pokud chcete nebo můžete novou verzi LOADER nahrát pomocí programu RTD, nemusíte tento program vůbec používat a tuto kapitolu můžete vynechat. Pokud ale musíte novou verzi LOADER nahrávat pomocí programu TheKing, můžete využít dodávaný program LdrPatch.

Upozornění:

- Při rozmísťování programu LdrPatch v RTD musíte jednotku -- LOADER -- umístit do modulu se jménem LdrPatch. Vzhledem k velikosti programu LdrPatch můžete do tohoto modulu umístit celý program.
- Při rozmísťování programu LOADER v RTD musíte jednotku -- LOADER -- umístit do modulu se jménem ROM. Vzhledem k velikosti programu LOADER při použití standardního komunikačního portu můžete do tohoto modulu umístit celý program.
- Pokud bude program LOADER používat přenos dat pomocí Ethernet protokolem UDP je třeba změnit nastavení direktiv v souboru Sets.inc a upravit rozmístění modulů v paměti.

## 9.1. Přizpůsobení požadavkům aplikace

Protože dodávaná verze programu LdrPatch je naprosto nezávislá na aplikaci a verzi LOADER, musíte nastavit několik proměnných. Pokud měníte tyto proměnné, musíte v některých případech provést upravení souboru LdrCom.mem případně LdrUdp.mem nebo provést nové rozmístění programu LOADER. Pokud tyto úpravy v souboru LdrCom.mem případně LdrUdp.mem neprovedete, nemusí program LdrPatch správně fungovat.

Pozn. Soubor LdrCom.mem slouží pro nahrání nové verze LOADER používající standardní COM případně komunikační port na V40. Soubor LdrUdp.mem slouží pro nahrání nové verze LOADER používající Ethernet s protokolem UDP.
Pozn. Při standardním nastavení LdrCom.mem a LdrUdp.mem by aplikace LdrPatch

měla být umístěna na adresu $E0000(laLdrPatch) do modulu o velikosti $3000. RAM modul by měl mít velikost $10000 a měl by být umístěn na adrese $B00.

Dále uvedené nastavení platí pouze pro variantu s použitím standardního komunikačního portu a souboru LdrCom.mem.

```
laLdrPatch = $E0000;
```

Adresa umístění programu LdrPatch dle nastavení programu.

Pokud bude program LdrPatch umístěn v rozsahu $E0000-$FD000 případně $E0000-$FE000, je nová verze LOADER do řídicího systému zapsána vždy bez ohledu na výpadky systému.

Pokud tuto adresu změníte, musíte upravit hodnotu adresy u modulu LdrPatch v souboru LdrCom.mem. Pokud tuto změnu neprovedete, nemusí program LdrPatch správně fungovat.

```
laLdrUnit = $E3000;
```

Adresa, na kterou se dočasně uloží nová verze programu LOADER. Z této adresy se po spuštění programu LdrPatch provede přepis starší verze LOADER.

Pokud tuto adresu změníte, musíte upravit hodnotu adresy u modulu ROM v souboru LdrCom.mem. Pokud tuto změnu neprovedete, nemusí program LdrPatch správně fungovat.

```
laLdr = $F1200;
cLdrSize = $0BE00;
```

Adresa umístění nové verze programu LOADER podle nastavení v RTD při jejím rozmísťování do paměti řídicího systému KIT. Pokud tuto adresu měníte, musíte znovu rozmístit program LOADER.

## 9.2. Nahrání nové verze LOADER pomocí programu TheKing

Nahrání nové verze provedete pomocí okna **Controller 1 File Transfer** v menu **Views\Communication states overview**. Při nahrávání musíte být přihlášen jako administrátor programu a komunikovat s programem LOADER, jinak je tlačítko **Load *.mem** neaktivní.

Pokud program TheKing komunikuje s programem LOADER nebo s nějakou aplikací, tak se v poli s označením ***Running application*** zobrazí nějaký řetězec. V případě komunikace s programem LOADER se zobrazí ***Loader:*** <libovolný text>. Pokud ale komunikuje s aplikací, zobrazí se ***UserApp:*** <libovolný text>. Přepnutí aplikace do programu LOADER provedete pomocí stisku tlačítka **Start**.

Identifikaci komunikujícího programu vyvoláte pomocí tlačítka **Query.** Pokud se aplikace neidentifikuje, tak tuto funkci nepodporuje a zobrazí se ***Unknown***. Přepnutí aplikace do programu LOADER, můžete provést nastaveným počtem reset po spuštění aplikace (max. 15s).

Stisk tlačítka **Load *.mem** vyvolá dialog pro výběr souboru s koncovkou *.mem. V dialogu vyberte soubor *LdrCom.mem* případně *LdrUdp.mem*. Po jeho přečtení by se měli zobrazit parametry modulů LdrPatch a ROM v dolní části okna **Controller 1 File Transfer** v kolonkách Module, Name, SegAddr, Size, CPU, MEM, Attrib a File path. Pokud při přečtení k žádné chybě,. můžete postoupit k dalšímu kroku.

Nahrání nové verze programu LOADER provedete pomocí tlačítka **Prg Download**. Pokud nedojde k žádné chybě, tak program LdrPatch spustíte stiskem tlačítka **Reset**.

Program LdrPatch po svém spuštění provede přepis starší verze LOADER, poškodí se a provede nové spuštění systému. Proto by se při dalším navázání komunikace měla zobrazit nová verze LOADER v poli s označením ***Running application***.

Tip:

- Při nahrávání nové verze LOADER můžete použít tlačítko **Automatic program Download**, které v najednou provede tyto akce - otevření dialogu pro přečtení souboru *.mem, zápis programu LdrPatch do paměti řídicího systému a reset systému.