LdrLib

KNIHOVNY PRO TVORBU ZAVADĚČE (LOADERU) ŘÍDICÍCH APLIKACÍ

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 2.20

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: 04.03.2005
Datum posledního uložení dokumentu: 04.03.2005
(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.

Obsah :

1. O dokumentu

1.1. Revize dokumentu

Verze dokumentu	Verze SW	Autor	Datum vydání	Popis změn
1.00	2.XX	Hvo	19.12.2002	První vydání
1.10	2.XX	Hvo	05.02.2003	Přesnější popis strLdrVer a strLdrName. Oprava implicitní adresy _laLdrDesc.
1.11	2.XX	Tum	16.05.2003	Úprava dokumentu dle ISO9000. Doplněna konstanta strLoader.
1.12	3.XX	Hvo	19.08.2003	Úprava interface objektu z důvodu přechodu na rozšířenou standardní knihovnu Crc16. Dřívější verze používaly ExCrc16. Upřesnění popisu práce s konstantou strLdrVer
2.00	4.XX	Hvo	21.01.2004	Rozšíření knihoven LDRLIB pro podporu kopírování bloku umístěných v prostoru spravovaném DiskIO V souvislosti s těmito změnami byl změněn komunikační protokol, od teto verze se používá verze 3. Opravena chyba typu proměnné, která obsahuje čas posledního spuštění aplikace (interní proměnná odpovídající wExecTime) Oprava typu a přejmenování proměnné, která obsahuje počet spuštění aplikace ve sledovaném čase (wExecNo přejmenovaná na bExecNo) Úprava komentářů Přidána podpora pro uživatelské funkce Init, Write, Copy, tzn. podpora knihoven DiskIO Upravena podpora procesoru V40. Musi se pridat knihovna LdrV40Fn a zavolat funkce LdrV40Init. Změna názvu a deklarace struktur tAppRstDesc a tLdrDesc. Struktury byly přesunuty do XtKing_1 Změna implicitních hodnot u proměnných _laAppRstDesc a _laLdrDesc
2.10	5.XX	Wil	26.08.2004	Zavedení jednotného formátu všech verzí struktur a programů (BCD formát definovaný knihovnou LibVer). Při více podmínkách, kdy se nespustí řídicí aplikace a čeká se na komunikaci (např. špatná ModuleTable, bParForLdr=1 apod.) se tyto podmínky kumulovaly a odmazávaly postupně při každém reset, tj. bylo nutno provést tolikrát reset, kolik bylo nakumulovaných podmínek, než došlo ke spuštení řídicí aplikace. Opraveno: Při špatné ModuleTable se vynuluje i případný bParForLdr. Úprava komunikačního rozhranní s vizualizací (protokol verze 4). - Rozšíření protokolu o příkazy pro čtení paměťových bloků. Změna struktury XtKing_1.tHwInfo4 - byla přidána položka obsahující verzi LOADER. Tato hodnota se v projektu bude používat pro omezení řídicí aplikace na min. verzi LOADER. Přidána návratová hodnota eLoaderInvalidGetPar. Modul s příznakem moduleattr_SrvcByApp není kontrolován, v předchozí verzi mohla chyba v tomto modulu způsobit nespuštění řídicí aplikace. Přidána podpora čtení bloku paměti z prostoru DiskIO.

				Přidány a přejmenovány chybová hlášení cWrite_XXXX -> cMem_XXXX. Přidány chybová hlášení ze strany LOADER, tj. eLOADER_XXXX. Změna velikosti LOADER při použití knihovny DiskIO, musí se změnit adresa jeho umístění a velikost modulu ve Flash. Zrušena konstanta xDesc.wLibVer, která se dříve používala ke kontrole verze systémových knihoven. To je ale aplikační záležitost a proto tuto kontrolu musí provést aplikace (tj. LOADER) pomocí funkce LibVer.TestLibVerNo.
2.20	5.XX	Wil	04.03.2005	Umožněn zápis (update) konfigurační tabulky BIOSu. Přidány funkce fnGetOnLinePar a fnPutOnLinePar pro zpracování uživatelských OnLine parametrů.

1.2. Účel dokumentu

Tento dokument slouží jako popis programového balíku knihoven LdrLib používaného pro tvorbu aplikace Loader, která se používá pro spuštění řídicí aplikace a nahrávání jejích nových verzí pomocí vizualizačního programu TheKing na PC. To znamená, pokud daný projekt vyžaduje možnost nahrávání nových verzí řídicích aplikací do systému Kit z vizualizačního programu TheKing, musí se vytvořit pomocí knihoven LdrLib aplikace Loader, která toto umožňuje. Knihovny LdrLib byly vytvořeny za účelem maximálního zjednodušení tvorby konkrétní aplikace Loader.

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

Řídicí aplikace – Aplikace nahraná v paměti Flash řídicího systému zpravidla provádějící měření, řízení, regulace nebo další úkony potřebné pro daný projekt (zakázku).

LDRLIB – Balík knihoven používaný pro tvorbu vlastní aplikace LOADER.

LOADER - Označuje uživatelskou aplikaci používanou jako zavaděč řídicí aplikace, která používá knihovny LDRLIB. Tato aplikace je uložena v paměti Flash řídicího systému.

Vizualizace, THEKING – Vizualizační program na PC, který pomocí komunikační linky je schopen zobrazovat data z řídicího systému a dále (pokud řídicí systém

obsahuje aplikaci LOADER) umožňuje nahrávání. V budoucnu mohou vznikat různé modifikace tohoto vizualizačního programu, které se mohou jmenovat jinak než TheKing.

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

3. Úvod

Pomocí programového balíku knihoven LDRLIB je možné vytvořit program dále nazývaný LOADER, kterým lze nahrát novou verzi řídicí aplikace pomocí programu TheKing. Nahrávání se provádí zpravidla po sériové komunikační lince RS232/RS485, ale lze použít i rozhraní ethernet nebo telefonní modem (vždy však s určitými úpravami knihoven LDRLIB nebo jejich podmíněným překladem).

Vytvořený program LOADER je možné do EEPROM (FLASH) paměti řídicího systému zapsat pomocí programátoru společně s BIOS nebo pomocí programu RTD.

Hlavním úkolem těchto knihoven je zajistit za všech okolností buď úspěšné spuštění řídicí 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 jsou schopny detekovat následující chyby aplikace – poškození a padání aplikace krátce po startu (např. zápis do paměti FLASH). Tyto chyby jsou zachyceny, zpracovány a při častém výskytu těchto chyb je možno nahrát novou verzi řídicí aplikace.

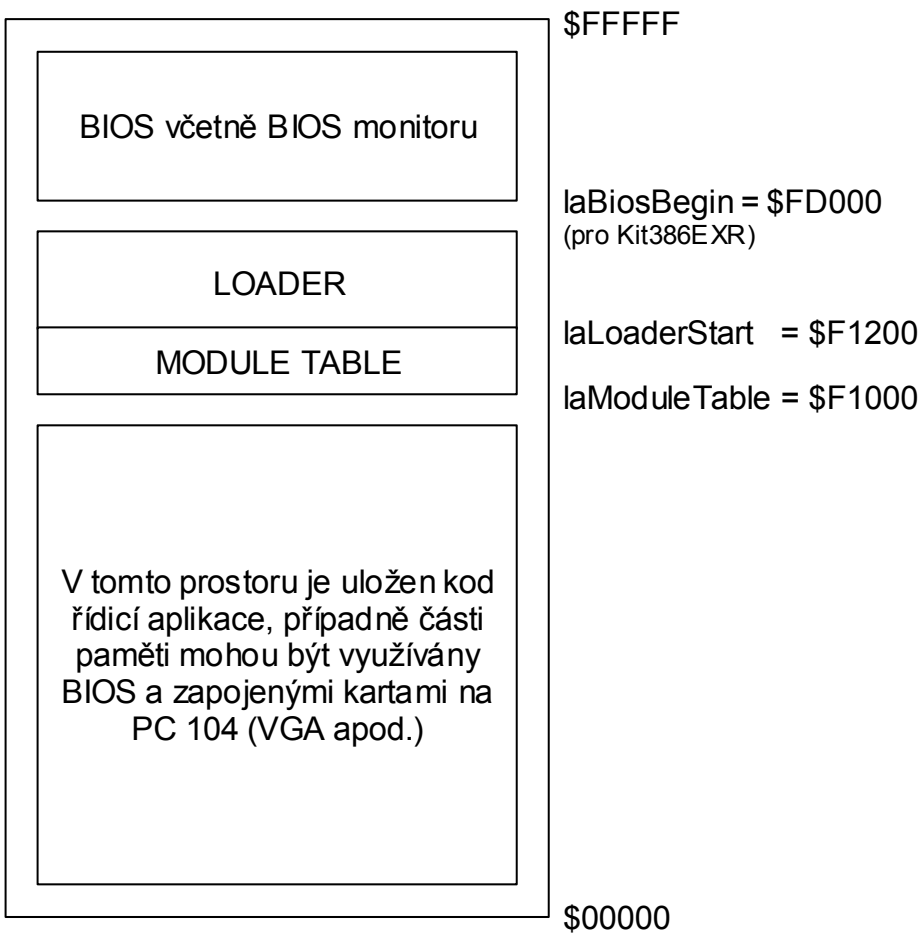
4. Rozložení paměti FLASH

Paměť Flash je rozdělena na několik částí viz „obrázek 1: Příklad umístění programu LOADER v paměti Flash řídicího systému“. Protože na začátku programu se provádí kontrola velikosti tabulky modulů (viz dále) a umístění tabulky modulů, je potřeba zachovat zobrazené pořadí oblastí v paměti řídicího systému. Pokud se použije standardní nastavení a vytvořený program LOADER je určen pro řídicí systém s Kit386 s přenosem dat po komunikačním portu COM, může se použít dále uvedené rozložení paměti.

Tabulka modulů ***ModuleTable*** je struktura ukládaná v paměti Flash s popisem jednotlivých binárních modulů (jejich umístění, velikost, atributy atd.), ze kterých se skládá řídicí aplikace. Tuto strukturu využívá LOADER zejména k tomu, aby určil startovací adresu řídicí aplikace a zjistil její platnost a konzistentnost v paměti Flash. Řídicí aplikace nesmí do této tabulky zapisovat.

Pozn. Při popisu začátku jednotlivých bloků paměti se používají lineární adresy.

Pozn. V případě použití přenosu dat po Ethernet s protokolem UDP nebo použití knihoven DiskIO je potřeba změnit rozložení paměti, protože program LOADER má vyšší nároky na paměť.



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

Při používání knihoven LdrLib je doporučeno nastavit podle typu procesoru a komunikačního kanálu následující hodnoty:

	KITV40	**KIT386EXR**	**KIT188ER**
V40_	laBiosBegin =$FE000 laLoaderStart=$F2000 laModuleTable=$F2200	nelze	nelze
COM188	Nelze	nelze	Dosud neimplementováno
COM	LaBiosBegin =$FE000 laLoaderStart=$F1000 laModuleTable=$F1200	laBiosBegin =$FD000 laLoaderStart=$F1000 laModuleTable=$F1200	Dosud neimplementováno
UDP	laBiosBegin =$FE000 laLoaderStart=$E0000 laModuleTable=$E0200	LaBiosBegin =$FD000 laLoaderTable=$E0000 laModuleTable=$E0200	Dosud neimplementováno
MODEM	Dosud neimplementováno	Dosud neimplementováno	Dosud neimplementováno

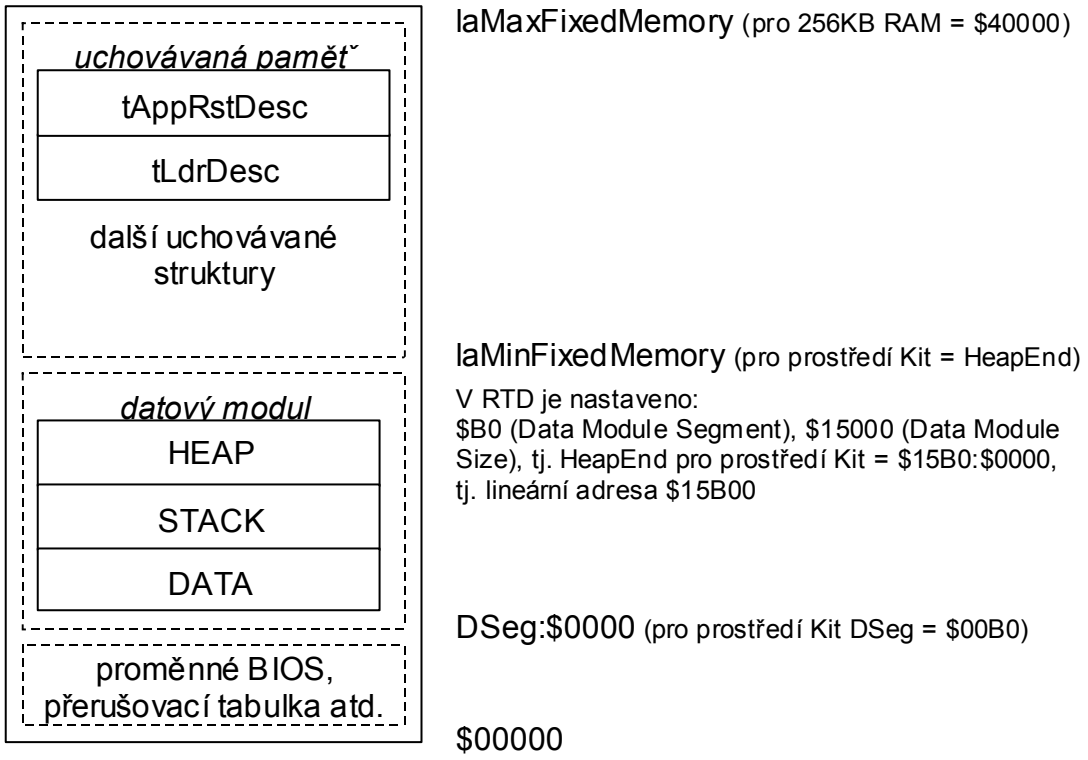
tabulka 1: Příklady funkčního nastavení konstant popisující rozložení paměti Flash

Tyto hodnoty (zejména **laLoaderStart** a tím i **laModuleTable**) jsou samozřejmě závislé na velikosti konkrétní Vaší aplikace LOADER, tj. pokud do této aplikace naprogramujete další vlastní algoritmy, velikost LOADERu se tím zvětší a tím musíte správně nastavit proměnné **laLoaderStart** a **laModuleTable**. Dále se doporučuje ponechat ve Flash paměti jistou rezervu (zpravidla 10%) pro případné budoucí verze LOADERu.

5. Rozložení paměti RAM

Paměť RAM je rozdělena na několik čístí viz „obrázek 2: Příklad rozložení struktur v paměti RAM“. *Datový modul* slouží pro pracovní proměnné aplikace LOADER nebo řídicí aplikace. Z toho plyne, že obsah této části paměti není uchován po reset nebo vypnutí napájení. *Uchovávaná paměť* se používá pro zálohované struktury i po reset nebo vypnutí napájení. Jsou v ní umístěny zejména dvě struktury: ***AppRstDesc*** a ***LdrDesc***, které jsou sdílené jak aplikací LOADER tak řídicí aplikací. Zbývající část *uchovávané paměti* je určena pro zálohovaná data řídicí aplikace (např. technologické parametry, archivy apod.).

Z důvodu kontrol překrytí rozmístěných struktur v programu LOADER musí být tyto struktury v paměti RAM rozloženy způsobem uvedeným na obrázku níže. Zobrazené rozložení paměti předpokládá řídicí systém s minimálně 256kB (=$40000) paměti RAM a datový modul o velikosti $15000B s umístěním na lineární adrese $B00 (segmentově $B0). Parametry tohoto modulu se zadávají v RTD.



obrázek 2: Příklad rozložení struktur v paměti RAM

Proměnná **laMaxFixedMemory** se nastavuje zpravidla na velikost používané RAM (tj. pro příklad na obrázku na $40000). Proměnná **laMinFixedMemory** se pro prostředí Kit zpravidla nastavuje na stejnou adresu jako je ukazatel **HeapEnd**, např. příkazem `laMinFixedMemory:=PtrToAbsAddr(HeapEnd);` Pro prostředí DOS na PC se musí nastavit ručně (tj. pro příklad na obrázku na $15B00) podle nastavení v RTD, jelikož proměnná HeapEnd má na PC pokaždé (při každém spuštění) jinou hodnotu poplatnou umístění aplikace v paměti spravované DOSem.

6. Popis spouštění a činnosti programu LOADER

V následující části je popsán průběh spouštění (dále stavy) a činnost programu LOADER po inicializaci řídicího systému (tzn. zapnutí napájení případně reset).

Některé stavy jsou poplatné doporučované konfiguraci a rozložení paměti, viz dodávané příklady.

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

Pozn. V jednotlivých stavech může být voláno více funkcí.

Na závěr této kapitoly bude uveden vývojový diagram popisující činnost programu LOADER.

6.1. Přizpůsobení programu

V tomto stavu se volají funkce **LdrAdapt** a **fnAdaptToUserApp**.

Funkce **LdrAdapt** slouží pro nastavení parametrů LOADER – timeout, parametry komunikačního kanálu, identifikace LOADER (včetně verze), implicitní nastavení parametrů paměti ATMEL Flash (viz dodávané příklady a popis knihovny ATMFlash) a případně signalizace běhu LOADER pomocí LED.

Proměnná **fnAdaptToUserApp** odkazuje na funkci, která provede nastavení adres obsahujících umístění sdílených struktur v paměti. Tato funkce by měla být umístěna v extra jednotce, aby mohla být také volána v řídicí aplikací.

6.2. Kontrola aplikace

V tomto stavu se volají funkce **LdrInit**, **ReadModuleTable**, **CheckExecTime, CopyBinModules a CheckAplication.**

6.2.1. LdrInit

Funkce **LdrInit** provede nastavení proměnných obsahujících adresy sdílených struktur mezi řídicí aplikací a programem LOADER. Jedná se o struktury ***AppRstDesc*** a ***LdrDesc***. Při jejich nastavování se použijí uživatelsky nastavitelné proměnné **_laAppRstDesc** a **_laLdrDesc**. Dále se nastaví implicitní adresa pro spouštění aplikace **laAppStart,** která se použije v případě chybějícího modulu s atributem ***Start*** v tabulce modulů.

Po jejich nastavení se provede kontrola verze struktury ***AppRstDesc*** typu ***tAppRstDesc1*** (struktura využívaná pro ladění řídicí aplikace, jsou do ní ukládána případné RunTime chyby). Výsledkem této kontroly je nastavení příznaku, zda se smí s touto strukturou pracovat. Pokud je verze struktury v položce **wVer** různá od hodnoty ***cAppRstDescVer***, postupuje se podle následujícího pravidla.

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 (jejich popis začíná prefixem „L „) nejsou do ní zapisovány.
- **spodní byte** vyjadřuje pouze změnu používání položky Dummy řídicí aplikací. Protože knihovny LDRLIB toto pole nepoužívají, jsou případné detekované chyby (jejich popis začíná prefixem „L „) do této struktury zapisovány dále.

Z tohoto popisu vyplývá, že struktura ***AppRstDesc*** přísluší zejména řídicí aplikaci. LOADER do ní může ukládat pouze v případě, pokud používá její stejnou nebo kompatibilní strukturu.

Struktura ***AppRstDesc*** je typu ***tAppRstDesc1***, který je definován v knihovně **XTKing_1** (není součástí balíku LdrLib), pro úplnost je zde ale uvedena.

```
PAppRstDesc1 = ^TAppRstDesc1;
TAppRstDesc1 = 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 se stejným kódem 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.

Pozn. Ve verzi s VGA (direktiva Deb_Crt) jsou do **ErrDescArray** také ukládány některé chyby s prefixem **ldrerr_XXX** s offsetem 20.

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

```
if  ((_laAppRstDesc + sizeof(tAppRstDesc1)) <= laMaxFixedMemory) and
    ((_laLdrDesc + sizeof(tLdrDesc3)) <= _laAppRstDesc) and
    (laMinFixedMemory <= _laLdrDesc)
   {$ifdef MCP}
    and
   (PtrToAbsAddr(HeapEnd) <= laMinFixedMemory)
   {$endif}
   then
```

Pokud se zjistí překryv, zapíše se do položky **wLastError** ve struktuře **LdrDesc** 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 do stavu komunikace s nadřízeným počítačem. V tomto stavu se může nahrát nová verze programu **LOADER**, případně aplikace kompatibilní se strukturou **LdrDesc**.

Struktura ***LdrDesc*** je typu ***tLdrDesc3***, který je definován v knihovně **XTKing_1** (není součástí balíku LdrLib), pro úplnost je zde ale uvedena.

```
PLdrDesc3 = ^TLdrDesc3;
TLdrDesc3 = record
  wVer            : Word;      { verze této 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 }
  wLdrVer         : Word;      { verze programu LOADER, ktery je
                               spusten )
                               { pridano od verze knihoven 4.XX }
  wLibVer         : Word;      { pridano od verze struktury 3.00
                               (knihovny 4.XX)
                               verze systemovych knihoven, s kterymi
                               byl LOADER prelozen}
  wCrc            : Word;      { CRC cele struktury }
end;
```

V této struktuře jsou uloženy statické datové položky knihoven LDRLIB. Tato deklarace struktury je používána od verze knihoven 4.XX.

Položka **wVer** je nastavena při každé inicializaci a mazání této struktury.

Položka **wLdrVer** je kontrolována při každém spuštění programu LOADER. Při rozdílu verzí LOADER se provede nastavení aktuální verze programu LOADER.

Pozn. Aplikace musí správně vyhodnocovat verze v položkách **wVer** a **wLdrVer**.

Pozn. Při poškození této struktury se nastaví položka **bParForLdr** na 1, což způsobí později přechod do stavu komunikace s nadřízeným počítačem.

Po volání funkce **LdrInit** se zavolá funkce **fnUserInit**, která má za úkol provést inicializaci uživatelské knihovny obsluhující uživatelskou paměť. Jestliže při jejím volání dojde k chybě, tato funkce nastaví chybový kód **fnSetLastError** a přejde se do stavu spuštění aplikace.

6.2.2. Detekce pamětí FLASH

Funkce **ATMIdent** se pokusí detekovat typ paměti ATMEL Flash. Pokud detekce skončí neúspěchem, nastaví typ paměti na hodnotu proměnné **wFlashManDev** a velikost sektoru na hodnotu proměnné **wFlashLenSect**. Dále se do položky ***wLastError*** ve struktuře **LdrDesc** uloží hodnota ***ldrerr_FlashId***.

Pozn. Pokud tato chyba nemá být ošetřována, musí se nastavit vlastní obsluha chyb v proměnné **fnSetLastError**.

6.2.3. ReadModuleTable

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

Po ukončení funkce **ReadModuleTable** se při návratové hodnotě FALSE přejde do stavu komunikace s nadřízeným počítačem. Při vrácení TRUE se otestuje položka **bParForLdr** ve struktuře **LdrDesc**. Pokud je její hodnota **1**, přechází se do stavu komunikace s nadřízeným počítačem. Pokud je její hodnota různá od **1**, pokračuje se v kontrole aplikace voláním funkcí **CheckExecTime, CopyBinModules a CheckApplication**.

6.2.4. CheckExecTime

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 **gpAppRstDesc^.ErrDescArray** a přejde se do stavu komunikace s nadřízeným počítačem. V opačném případě se po návratu z funkce přechází do stavu spuštění aplikace.

6.2.5. CopyBinModules

Funkce **CopyBinModules** volá funkci odkazovanou proměnnou ***fnCopyBinModules***, která umožní zavedení aplikace, její kontrolu a spuštění z uživatelsky obsluhované paměti.

Pozn. Pro práci s rozšířenou pamětí se musí k programu LOADER přidat jednotka **LdrDUAFn** a provést její inicializace voláním funkce **LdrExtDiskUnitInit**.

6.2.6. CheckApplication

Funkce **CheckApplication** provede kontrolu aplikace podle tabulky modulů.
Pozn. Podle verze protokolu (tj. knihoven LDRLIB) se tabulka modulů tModuleDesc generuje v LOADER (verze 2.XX a 3.XX) nebo ve vizualizaci (verze 4.XX a novější).

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 *„check sum“* modulů s příznakem ***CheckSum*** je 0. Pokud je *„check sum“* neplatný, přeruší se další kontroly a přechází se na stavu komunikace s nadřízeným počítačem. Pokud je *„check sum“* platný, tak se u spustitelných souborů s příznakem ***Executable*** zkontroluje počáteční hodnota modulu na $AA55. Jestliže tato hodnota na začátku modulu není, přeruší se další kontroly a přechází se na stavu komunikace s nadřízeným počítačem. U modulu s příznak ***Start*** se nastaví do proměnné **laAppStart** adresa startu aplikace. Tzn. přepíše se implicitní nastavení startovací adresy aplikace. V případě že se při kontrole aplikace zjistí chyba, *„check sum“* nebo počáteční hodnota modulu, zapíše se do pole **gpAppRstDesc^.ErrDescArray** chyba **„L SUM“**.
Pozn. Při nahrávání aplikace do řídicího systému pomocí programu **TheKing** je možné příznaky modulů změnit. Podrobný popis nastavování těchto příznaků je uveden v manuálu programu **TheKing**.

6.3. Spuštění řídicí aplikace

Ještě před spuštěním aplikace se vyvolá uživatelská procedura **fnBeforeApp**.

V tomto stavu se provede spuštění aplikace podle hodnoty v proměnné **laAppStart**. Pokud je tato proměnná různá od NIL, provede se spuštění aplikace bez ohledu na předchozí stav.

Jestliže je proměnná **laAppStart** nastavena na NIL a v předchozím stavu probíhala komunikace s nadřízeným počítačem, uloží se chyba ***ldrerr_RunApp*** a řídicí systém se inicializuje. Jestliže v předchozím stavu komunikace neprobíhala pokračuje se do stavu komunikace s nadřízeným počítačem.

6.4. Komunikace s nadřízeným počítačem

Tento stav je časově omezena hodnotou v proměnné **liTimeoutToAppRun**. Pokud v této době nedojde k navázání komunikace s vizualizací na nadřízeném počítači, přechází se do stavu spuštění aplikace. V opačném případě se provádí následující algoritmus.

Nejdříve se otevře komunikační kanál se zadanými parametry. Pokud se kanál nepodaří otevřít, mohou se uložit do položky **wLastError** ve struktuře **LdrDesc** následující hodnoty ***ldrerr_COMPar***, ***ldrerr_Open***, ***ldrerr_RecBuff*** a ***ldrerr_Connect*** a poté se přejde do stavu spuštění aplikace.
Pozn. Bližší popis požadavků a parametrů je uveden v samostatném manuálu příslušného použitého komunikačního kanálu (ChnCom, ChnV40_, Chn188 apod.).

Po otevření komunikačního kanálu se volá funkce odkazovaná proměnnou **fnBeforeSetTick**, která v případě procesoru KitV40 umožní provést zjištění potřebné korekce časovačů. Vlastní korekce se provede při následném volání funkce odkazované proměnnou **fnSetTick**, která se volá v zakázaném přerušení.

Po provedení korekce časovače se nastaví obsluha Watch-Dog, která hlídá komunikaci s nadřízeným počítačem. Pokud dojde k zastavení komunikace delší než čas uvedený v proměnné **wWinComWD**, uloží se do položky **wLastError** ve struktuře **LdrDesc** hodnota ***ldrerr_Freeze*** a provede se přechod do stavu spuštění aplikace.

Nakonec tohoto stavu se spustí čekání na komunikaci s nadřízeným počítačem. Jestliže k navázání komunikace v nastaveném čase (viz výše) nedošlo, přechází se do

stavu spuštění aplikace. V případě navázání komunikace lze z tohoto stavu přejít do dále uvedených stavů pomocí stisku tlačítek v programu **TheKing**.

- Po stisku tlačítka *„Restart“* nebo pomocí **reset** lze provést inicializaci řídicí systému a nové spuštění programu LOADER.
- Po stisku tlačítka *„Prg Download“* nebo *„Automatic program download“* se přechází do stavu nahrávání nové aplikace.

Při probíhající komunikaci s nadřízeným počítačem, na kterém je spuštěn vizualizační program TheKing, se obsluhuje následující přenos dat:

Archivy – Aplikace LOADER neobsluhuje žádné archivy, tj. na dotaz počtu archivů od TheKing odpovídá číslem 0.

Binární soubory (File) - Aplikace LOADER neobsluhuje žádné soubory, tj. na dotaz počtu souborů od TheKing odpovídá číslem 0.

OnLine parametry – Aplikace LOADER obsluhuje následující systémové OnLine parametry:

LA_RdMemoryInfo,
LA_MemoryInfoRq – čtení obecného bloku paměti
LA_RstStruct – čtení Restart struktury

Aplikace LOADER je dále schopna obsluhovat uživatelské OnLine parametry (viz následující kapitola).

6.4.1. Čtení a zápis uživatelských OnLine parametrů

Při čtení uživatelských OnLine parametrů se volá funkce **fnGetOnLinePar**. Při zápisu uživatelských OnLine parametrů se volá funkce **fnPutOnLinePar**. Parametry obou funkcí jsou stejné. Parametr **aLA** udává kód (logickou adresu) OnLine parametru. Parametr **aData** udává ukazatel na čtená/zapisovaná data.

Definice funkcí fnGetOnLinePar a fnPutOnLinePar je zde pro úplnost uvedena:

```
type
 { uživatelské procedury pro čtení a zápis OnLine parametrů }
  tUserGetOnLinePar = function(aLA:word; var aData:pointer):
                              tLoaderRetCode;
  tUserPutOnLinePar = function(aLA:word; var aData:pointer):
                              tLoaderRetCode;
  {tyto funkce vrací jako výsledek jednu z hodnot:
     eLoaderOK           - OnLine parametr byl v pořádku
                           přečten/zapsán
     eLoaderNoSupport    - neznámý OnLine parametr
     eLoaderShowResetDlg - výzva k zobrazení Reset dialogu
  }

function EmptyOnLinePar(aLA:word; var aData:pointer):tLoaderRetCode;
         far;

const
  fnGetOnLinePar : tUserGetOnLinePar = EmptyOnLinePar;
  fnPutOnLinePar : tUserPutOnLinePar = EmptyOnLinePar;
```

Funkce fnGetOnLinePar a fnPutOnLinePar implicitně neznají žádný uživatelský OnLine parametr a vrací hodnotu eLoaderNoSupport.

Nyní si na příkladu ukážeme, jak v aplikaci LOADER obsloužit některé OnLine parametry.

```
{ Funkce pro obsluhu cteni OnLine parametru specifickych pro dany
  Loader }
function MyGetOnLineParam(aLA:word; var aData:pointer):
                          tLoaderRetCode; far;
var pB:^Byte;
begin
  MyGetOnLineParam:=eLoaderOk;
  pB:=aData;
  case aLA of
    110 : {čtení parametru typu word}
      begin
        word(aData^):=$1234;
        Inc(pB,SizeOf(word));
      end;
    120 : {čtení parametru typu real}
      begin
        real(aData^):=35.5;
        Inc(pB,SizeOf(real));
      end;
    else {case aLA}
      MyGetOnLineParam:=eLoaderNoSupport;
  end;{case aLA}
  aData:=pB;
end;

{ Funkce pro obsluhu zapisu OnLine parametru specifickych pro dany
  Loader }
function MyPutOnLineParam(aLA:word; var aData:pointer):
                          tLoaderRetCode; far;
var pB:^Byte;
begin
  MyPutOnLineParam:=eLoaderOk;
  pB:=aData;
  case aLA of
    110 : {zápis parametru typu word}
      begin
        GlbWord:=word(aData^);
        Inc(pB,SizeOf(word));
      end;
    else {case aLA}
      MyPutOnLineParam:=eLoaderNoSupport;
  end;{case aLA}
  aData:=pB;
end;
```

Poté v hlavním těle aplikace LOADER, před volání funkce **fnRunLoader**, nastavíme námi nadefinované funkce pro čtení a zápis OnLine parametru specifických pro daný Loader:

```
fnGetOnLinePar := MyGetOnLineParam;
fnPutOnLinePar := MyPutOnLineParam;
```

Tím je zajištěno, že při příchodu požadavku na čtení/zápis OnLine parametru se nejprve otestuje, jestli se nejedná o systémový OnLine parametr a poté se zavolá příslušná funkce fnGetOnLinePar/fnPutOnLinePar.

Pozn: Tyto funkce se používají (volají) nejen při čtení/zápisu jednotlivých OnLine parametrů, ale i při čtení/zápisu blokových (tj. skupiny) OnLine parametrů.

6.5. Nahrávání nové aplikace

V tomto stavu se pomocí vizualizačního programu **TheKing** zapisuje nová aplikace. Do paměti řídicího systému se zapisují jednotlivé datové BIN bloky dle příchozích požadavků.
Pozn. Na rozdíl od knihoven verze 2.XX a 3.XX, kde tabulka modulů **ModuleTable** se vytvářela po zápisu všech datových bloků, je ve verzi 4.XX a výše tabulka modulů generována v programu **TheKing** a zapisována jako běžný datový blok.

Pokud během nahrávání dojde k přerušení (výpadek napájení, reset nebo po stisku tlačítka *„Abort“*), musí se celá aplikace nahrát znova, protože tabulka modulů a aplikace jsou v nedefinovaném stavu.

Loader při ukládání příchozích datových BIN bloků provádí následující kontroly:

- Je-li příchozí datový blok určen pro použitý řídicí systém (KitV40, Kit386 apod.).
- Nezasahuje-li příchozí datový blok do oblasti vyhrazenou pro BIOS. Výjimkou je konfigurační tabulka BIOSu.
- Nezasahuje-li příchozí datový blok do oblasti vyhrazenou pro LOADER.

Při nesplnění jakékoliv podmínky se zápis datového bloku do paměti neprovede a LOADER vrátí do vizualizačního programu některý z chybových kódů uvedených v kapitole „12 Chyby zobrazované ve vizualizaci“

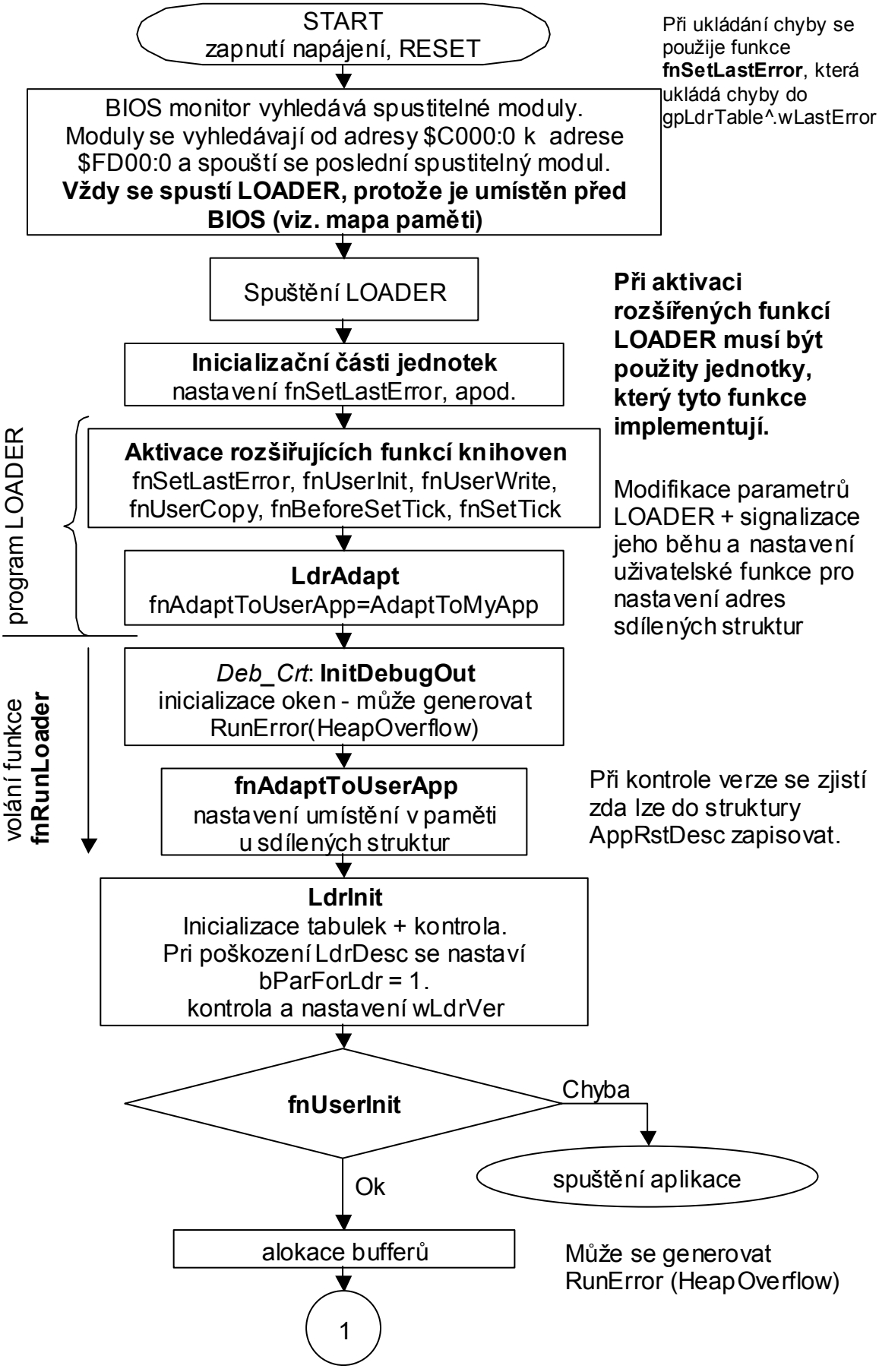
6.6. Vývojový diagram

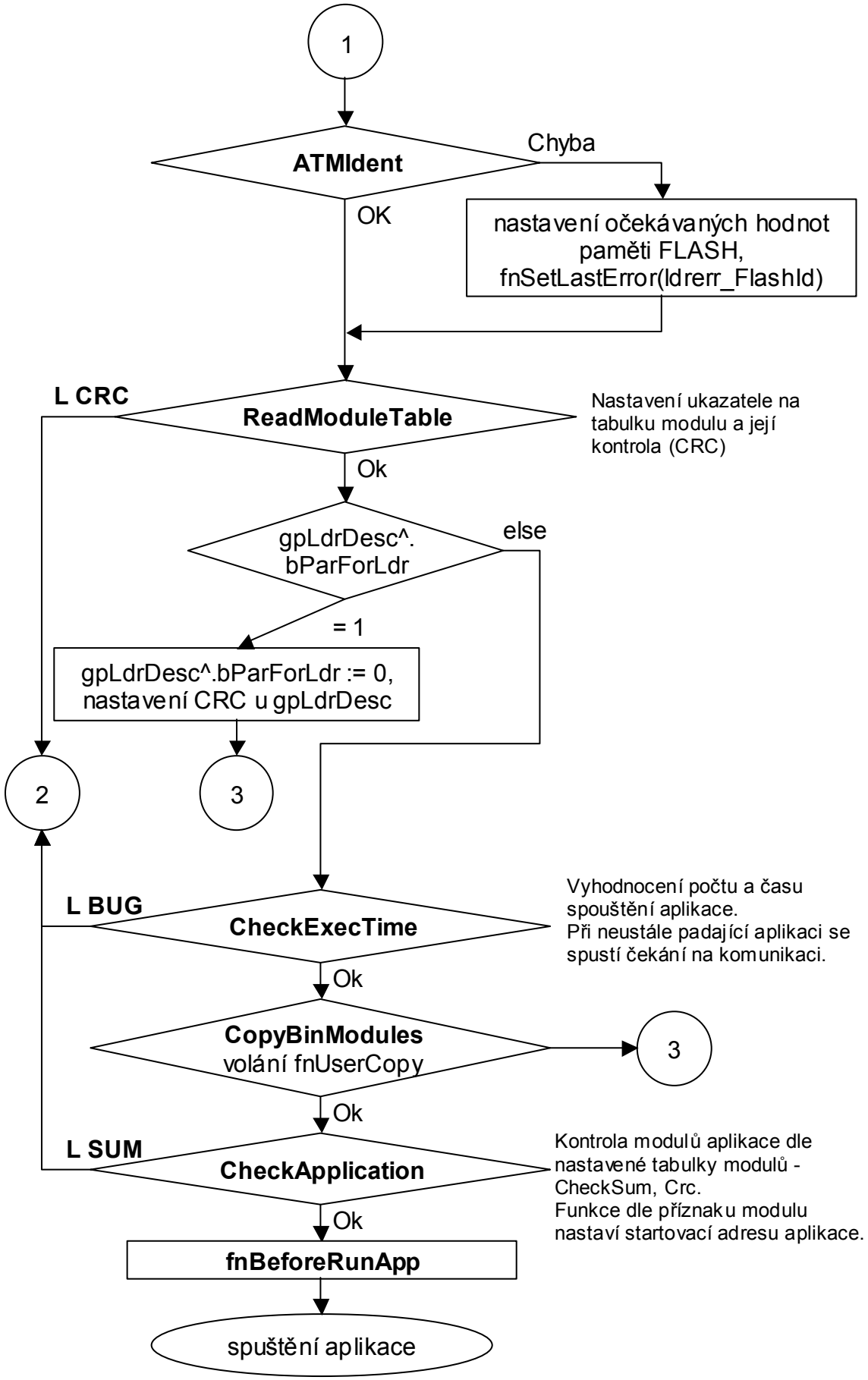
Pozn. Protože knihovny LDRLIB umožňují pracovat v ladicím režimu na PC s obrazovkou (direktiva Deb_Crt), volá se na PC funkce **InitDebugOut**.

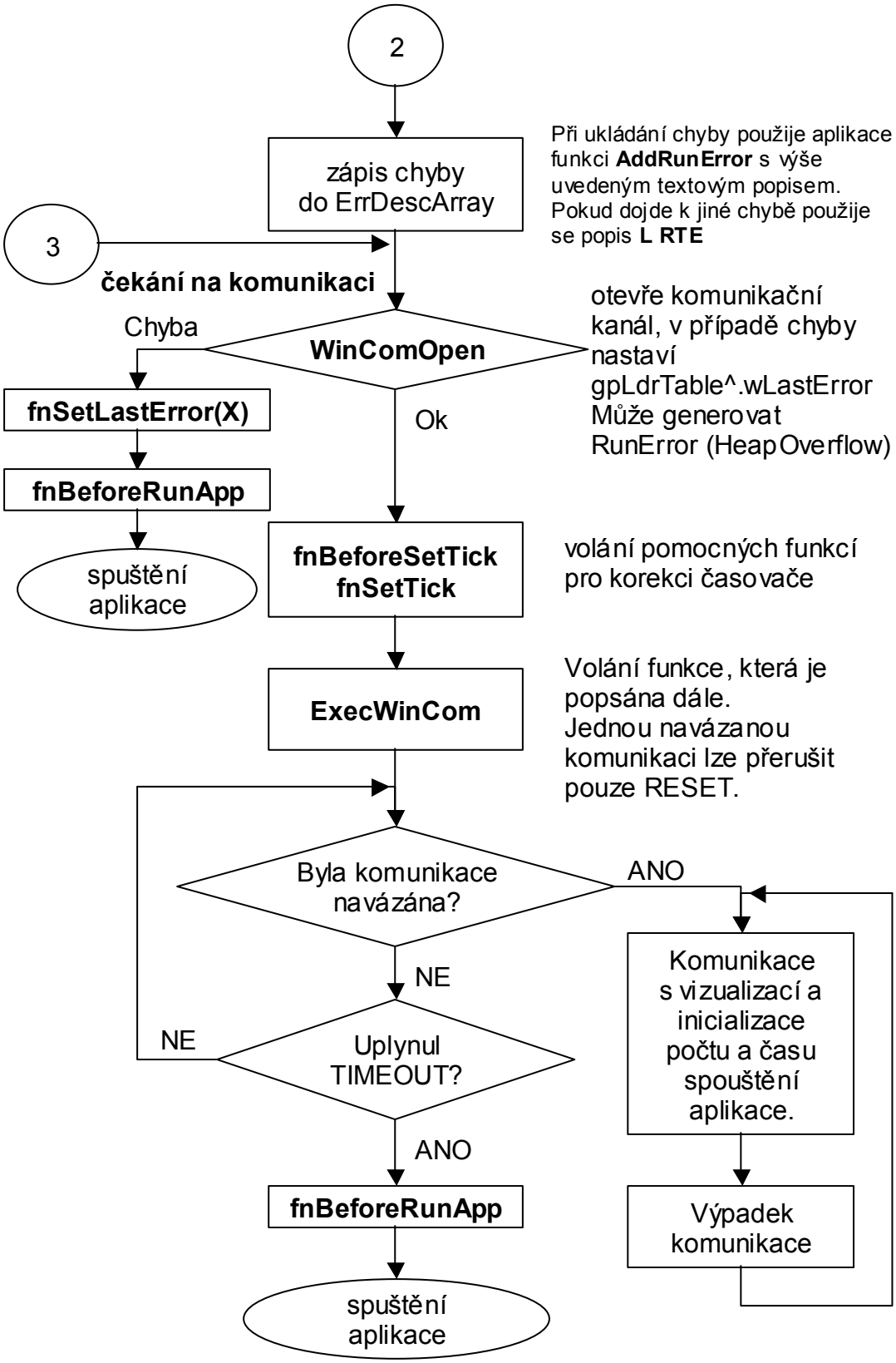
Pozn. Při spuštění programu LOADER na PC v prostředí DOS se sdílené (zálohované) struktury vytváří na HEAPu a jsou automaticky zapisovány případně čteny ze souboru při spuštění nebo ukončení LOADERu. Proto při jejich vytváření může dojít k RunTime chybě 203 (**Heap_Overflow**).

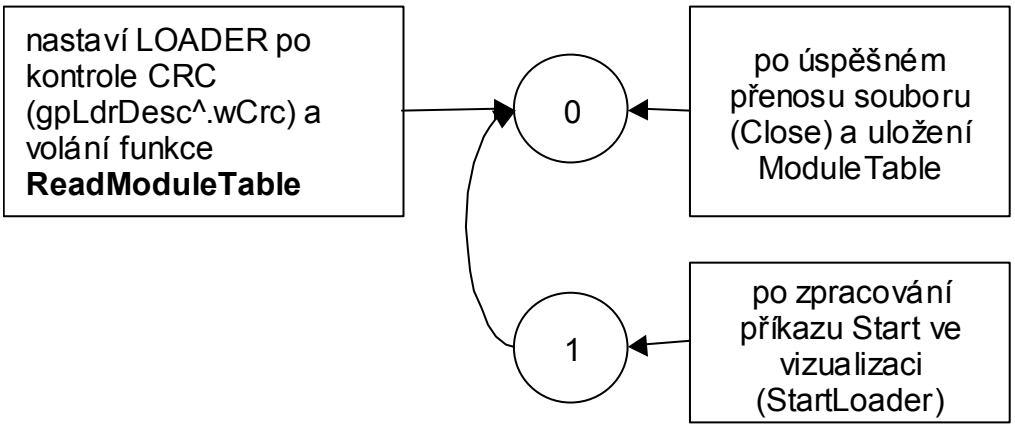
Pozn. Pokud při běhu LOADER dojde k vygenerování RunTime chyby, je tato chyba uložena do **ErrDescArray** jako **„L RTE“**. Číslo kódu odpovídá poslední vygenerované chybě. Počítadlo těchto chyb je použito pro všechny runtime chyby.

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.

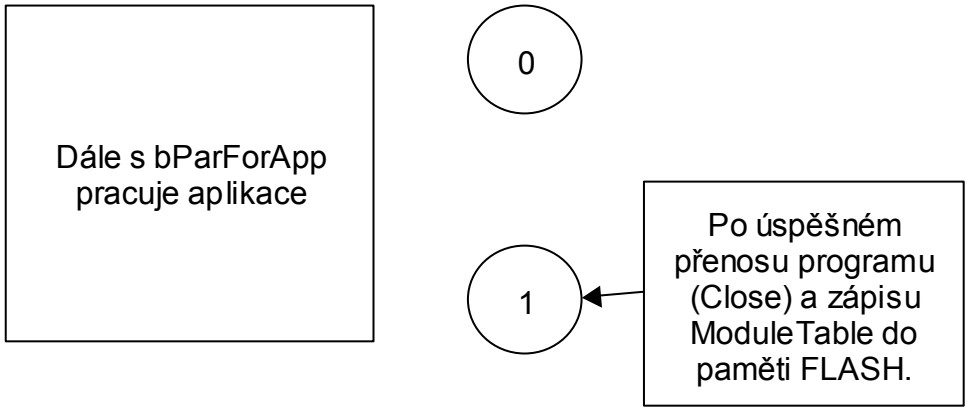

START
zapnutí napájení, RESET
BIOS monitor vyhledává spustitelné moduly.
Moduly se vyhledávají od adresy $C000:0 k adrese $FD00:0 a spouští se poslední spustitelný modul.
Vždy se spustí LOADER, protože je umístěn před BIOS (viz. mapa paměti)
Při ukládání chyby se použije funkce **fnSetLastError**, která ukládá chyby do gpLdrTable^.wLastError
Spuštění LOADER
Inicializační části jednotek
nastavení fnSetLastError, apod.
Při aktivaci rozšířených funkcí LOADER musí být použity jednotky, který tyto funkce implementují.
program LOADER
Aktivace rozšiřujících funkcí knihoven
fnSetLastError, fnUserInit, fnUserWrite, fnUserCopy, fnBeforeSetTick, fnSetTick
Modifikace parametrů LOADER + signalizace jeho běhu a nastavení uživatelské funkce pro nastavení adres sdílených struktur
LdrAdapt
fnAdaptToUserApp=AdaptToMyApp
volání funkce **fnRunLoader**
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 AppRstDesc zapisovat.
LdrInit
Inicializace tabulek + kontrola.
Pri poškození LdrDesc se nastaví bParForLdr = 1.
kontrola a nastavení wLdrVer
fnUserInit
Chyba
spuštění aplikace
Ok
alokace bufferů
Může se generovat RunError (HeapOverflow)
1


1
ATMIdent
Chyba
OK
nastavení očekávaných hodnot paměti FLASH, fnSetLastError(ldrerr_FlashId)
L CRC
ReadModuleTable
Nastavení ukazatele na tabulku modulu a její kontrola (CRC)
Ok
gpLdrDesc^. bParForLdr
else
= 1
gpLdrDesc^.bParForLdr := 0, nastavení CRC u gpLdrDesc
2
3
L BUG
CheckExecTime
Vyhodnocení počtu a času spouštění aplikace.
Při neustále padající aplikaci se spustí čekání na komunikaci.
Ok
CopyBinModules
volání fnUserCopy
3
Ok
L SUM
CheckApplication
Kontrola modulů aplikace dle nastavené tabulky modulů - CheckSum, Crc.
Funkce dle příznaku modulu nastaví startovací adresu aplikace.
Ok
fnBeforeRunApp
spuštění aplikace


2
zápis chyby do ErrDescArray
Při ukládání chyby použije aplikace funkci **AddRunError** s výše uvedeným textovým popisem. Pokud dojde k jiné chybě použije se popis **L RTE**
3
čekání na komunikaci
Chyba
WinComOpen
otevře komunikační kanál, v případě chyby nastaví gpLdrTable^.wLastError Může generovat RunError (HeapOverflow)
fnSetLastError(X)
Ok
fnBeforeRunApp
spuštění aplikace
fnBeforeSetTick fnSetTick
volání pomocných funkcí pro korekci časovače
ExecWinCom
Volání funkce, která je popsána dále. Jednou navázanou komunikaci lze přerušit pouze RESET.
Byla komunikace navázána?
ANO
NE
Komunikace s vizualizací a inicializace počtu a času spouštění aplikace.
NE
Uplynul TIMEOUT?
ANO
fnBeforeRunApp
Výpadek komunikace
spuštění aplikace



obrázek 3: Automat popisující nastavování položky gpLdrDesc^.bParForLdr



obrázek 4: Automat popisující nastavování položky gpLdrDesc^.bParForApp

7. Podpora rozšířené paměti

Pro podporu práce s rozšířenou pamětí implementovanou v knihovnách DiskIO se musí k programu LOADER přidat jednotka **LdrDUAFn** a provést její inicializaci voláním funkce **LdrExtDiskUnitInit**. Tuto inicializaci je vhodné provést před voláním funkce **fnRunLoader**.

8. Podpora procesoru KitV40

Pro podporu práce s procesorem KitV40 používajícím jeho interní sériový komunikační port se musí k programu LOADER přidat jednotka **ChnV40_**.

9. Podpora procesoru Kit188ER

Pro podporu práce s procesorem Kit188ER používajícím jeho interní sériový komunikační port na konektoru X2 se musí k programu LOADER přidat jednotka **Chn188**.

10. Přizpůsobení programu LOADER Vaší řídicí aplikaci

Program LOADER lze přizpůsobit požadavkům řídicí aplikace pomocí následujících proměnných při dodržení pořadí a rozmístění oblastí a struktur uvedených na předchozích obrázcích, viz „obrázek 1: Příklad umístění programu LOADER v paměti Flash řídicího systému“ a „obrázek 2: Příklad rozložení struktur v paměti RAM“.

10.1. Proměnné pro nastavení chodu LOADERu

- Následující globální proměnné jsou definovány v jednotce **LdrDesc**.

```
wExecTime     : word; (defaultně 90s)
bExecNo       : byte; (defaultně 5)
```

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

```
wWinComWD     : word; (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 : longint; (defaultně 60000 = 60s)
```

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

```
liTimeoutToReady : longint; (defaultně 10000 = 10s)
```

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

```
liTimeoutToImmediateSend : longint; (defaultně 2500 = 2.5s)
```

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, např. ***ChnCom***, ***ChnV40_***, ***UDPPrt***. Popis práce s komunikačními knihovnami odpovídá zvyklostem při práci s komunikací, viz samostatné manuály komunikačních knihoven.

```
strLdrVer           (defaultně nastaveno $0000)
```

Verze programu LOADER v BCD formátu. Horní byte znamená BCD číslo před desetinnou tečkou a spodní byte znamená BCD číslo za desetinnou tečkou. Např. $0210 znamená 02.10. **Programátor musí tuto proměnnou správně nastavit v programu LOADER!** Pokud tak neučiní, tj. proměnná zůstane nastavena na default hodnotu ($0000), **bude se generovat RunError(255)**.

`strLdrName` (defaultně 'Ldr NotInitiated')

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 '+VerNoToStr(strLdrVer);
```

- Následující proměnné jsou rovněž definovány v jednotce **LdrDesc**, ale jejich nastavení se provádí automaticky, tj. programátor aplikace LOADER je nesmí měnit. Může je však použít pro přístup paměťovým oblastem, na které ukazují.

`upAppStart : Pointer;`

Ukazatel na spouštěcí blok řídicí aplikace. Nastaví se automaticky podle nalezeného spouštěcího modulu řídicí aplikace v tabulce modulů ModuleTable nebo (při jejím poškození) na adresu danou proměnnou laAppStart (viz níže).

`gpAppRstDesc : pAppRstDesc1;`

Ukazatel na prostor **AppRstDesc**. Tento ukazatel se nastaví automaticky v prostředí Kit podle proměnné _laAppRstDesc (viz níže), v prostředí DOS na PC podle alokace struktury na HEAPu.

`gpLdrDesc : pLdrDesc3;`

Ukazatel na prostor **LdrDesc**. Tento ukazatel se nastaví automaticky v prostředí Kit podle proměnné _laLdrDesc (viz níže), v prostředí DOS na PC podle alokace struktury na HEAPu.

10.2. Proměnné pro nastavení sdílených paměťových oblastí

Protože vytvořený program LOADER, sdílí s aplikací některé struktury, musí se tyto struktury umísťovat na stejnou adresu v paměti. 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 uxDesc, 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      : Longint; (defaultně $80000, případně se volá
                             funkce GetBaseAddr z jednotky HwSyst)
```

Proměnná obsahuje bázovou adresu umístění pamětí FLASH.

`laAppStart : LongInt; (defaultně $E0000)`

Implicitní (očekávaná) počáteční adresa spouštěcího modulu řídicí aplikace, tj. modul obsahující unitu „--LOADER—„ (POZOR zde dochází k používání stejného klíčového slova „LOADER“, ale ve zcela jiném významu. Jednotku „- - LOADER - -„ obsahuje každá aplikace určená pro řídicí systémy Kit. Jedná se v podstatě o obdobu hlavičky EXE souboru pro PC). U takto zadané adresy se ignorují spodní 4bity. V případě poškození ModuleTable (tzn. nenalezení spouštěcí adresy řídicí aplikace) se použije tato adresa jako defaultní adresa pro start řídicí aplikace.

`laModuleTable : LongInt; (defaultně $F1000)`

Lineární adresa začátku tabulky modulů ModuleTable. Při jejím zadávání je potřeba respektovat pořadí umístění ModuleTable vůči aplikaci LOADER

v paměti FLASH, viz „obrázek 1: Příklad umístění programu LOADER v paměti Flash řídicího systému“.

```
laLoaderStart     : LongInt; (defaultně $F1200)
```

Lineární adresa začátku programu LOADER (musí odpovídat nastavení v RTD). Při jejím zadávání je potřeba respektovat pořadí umístění aplikace LOADER vůči ModuleTable v paměti FLASH„ viz „obrázek 1: Příklad umístění programu LOADER v paměti Flash řídicího systému“.

```
_laAppRstDesc     : LongInt; (defaultně $3FE00, což je
                              laMaxFixedMemory - sizeof(tAppRstDesc1)
                              - Reserve(13))
```

Lineární adresa začátku struktury **AppRstDesc** pro prostředí Kit. Při jejím zadávání je potřeba respektovat pořadí vůči struktuře LdrDesc, viz „obrázek 2: Příklad rozložení struktur v paměti RAM“.

```
_laLdrDesc        : LongInt; (defaultně $3FDE0, což je _laAppRstDesc -
                              sizeof(tLdrDesc3) - Reserve(9))
```

Lineární adresa začátku struktury **LdrDesc** pro prostředí Kit. Při jejím zadávání je potřeba respektovat pořadí vůči struktuře AppRstDesc, viz „obrázek 2: Příklad rozložení struktur v paměti RAM“.

```
laMaxFixedMemory : LongInt; (defaultně $40000 = 256KB)
```

Maximální lineární adresa uchovávané paměti RAM – zpravidla velikost osazené RAM.

```
laMinFixedMemory : LongInt;
```

Lineární adresa počátku uchovávané paměti RAM. Pro prostředí Kit se nastavuje automaticky na hodnotu podle ukazatele HeapEnd.

```
fUseErrDescArray : Boolean;
```

Příznak určující zda LOADER může zapisovat do struktury AppRstDesc, tj. zda používá stejnou verzi této struktury jako řídicí aplikace. Tento příznak se nastavuje automaticky.

10.3. Funkce

```
fnSetLastError : tSetLastErrorProc;
```

Funkce odkazovaná touto proměnnou je volána při interní chybě LOADER. Této funkci jsou předávány konstanty **ldrerr_XXX**.

Pozn. Ve verzi s VGA (direktiva Deb_Crt) jsou tyto chyby také zapisovány do **ErrDescArray** s offsetem 20.

`fnAdaptToUserApp : procedure;`

Funkce odkazovaná touto proměnnou je volána po provedení základních inicializací LOADER a je určena pro nastavení aplikačně závislých proměnných:

- laAppStart
- laModuleTable
- laLoaderStart
- _laAppRstDesc
- _laLdrDesc
- laMaxFixedMemory
- laMinFixedMemory

`fnUserInit : tUserInit;`

Funkce odkazovaná touto proměnnou je volána po provedení základních inicializací LOADER a je určena pro inicializaci knihovny spravující uživatelskou paměť.

Pozn. při práci s knihovnou DiskIO se musí přidat jednotka **LdrDUAFn** a provést její inicializaci volání funkce **LdrExtDiskUnitInit**.

`fnUserWrite : tUserWrite;`

Funkce odkazovaná touto proměnnou je volána při komunikaci s nadřízeným počítačem při zápisech přijatých datových bloků do paměti spravované uživatelskou knihovnou.

Pozn. při práci s knihovnou DiskIO se musí přidat jednotka **LdrDUAFn** a provést její inicializaci volání funkce **LdrExtDiskUnitInit**.

`fnUserCopy : tUserCopy;`

Funkce odkazovaná touto proměnnou je volána v těle funkce **CopyBinModules** a je určena pro připravení aplikace ke spuštění. Tzn. kopíruje část aplikace z uživatelské paměti do paměti, ve které aplikace bude spuštěna.

Pozn. při práci s knihovnou DiskIO se musí přidat jednotka **LdrDUAFn** a provést její inicializaci volání funkce **LdrExtDiskUnitInit**.

`fnBeforeSetTick : tEmptyBeforeSetTick;`

Funkce odkazovaná touto proměnnou je volána po inicializaci komunikačního kanálu a slouží ke zjištění požadavků na korekci časovače.

`fnSetTick : tEmptyProc;`

Funkce odkazovaná touto proměnnou je volána po inicializaci komunikačního kanálu a slouží k nastavení korekce časovače. Funkce je volána v zakázaném přerušení, proto by měla být co nejrychlejší.

`fnBeforeRunApp : tEmptyProc = EmptyProc;`

Funkce odkazovaná touto proměnnou je volána těsně před spuštěním řídicí aplikace.

11. Popis chybových kódů

Chybové kódy jsou definovány v souboru **xDesc.pas**.

Chyby s prefixem **err_** jsou používány jak programem LOADER tak řídicí aplikací. Kódy lze zjistit při přečtení struktury **LdrDesc** pomocí čtení paměti v RTD v položce **wLastError**.

```
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 v knihovnách LDRLIB a mohou být přečteny současně se strukturou **LdrDesc** v položce **wLastError**.
Pozn. Ve verzi s VGA (direktiva Deb_Crt) jsou tyto chyby také zapisovány do **ErrDescArrray** s offsetem 20.

```
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;  { watchdog komunikace }
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 }
ldrerr_AppRstVer    = 17; { nekompatibilita verze AppRstDesc,
                            LOADER nesmi pracovat s AppRstDesc }
                          { pridano od verze 4.00 }
ldrerr_User         = 18; { pri praci s uzivatelskou knihovnou pro
                            obsluhu pameti doslo k chybe, horni část
                            obsahuje chybovy kod knihovny }
                          { pridano od verze 4.00 }
ldrerr_ChngLdrVer   = 19; { pri spusteni LOADER se byla zapsana
                            nova (aktualni) verze programu do
                            polozky tLdrDesc.wLdrDesc }
                          { pridano od verze 4.00 }
```

```
ldrerr_RtcTime      = 30; { detekovan neplatny cas v RTC ->
                            nastaveni cMinDateTime -> chyba RTC }
                            { pridano od verze 4.00 }
```

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 knihoven
                               Pascal }
```

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) }
```

12. Chyby zobrazované ve vizualizaci

Číselné kódy zobrazované ve vizualizaci jsou deklarovány ve výčtovém typu **tLoaderRetCode** v jednotce **xDesc** a mají následující význam.

Číselný kód	Jméno konstanty	Význam
0	eLoaderOk	Operace byla úspěšně dokončena.
1	eLoaderNotSupport	Požadovaná operace není podporována, nejspíše nekompatibilní verze protokolu
2	eLoaderBreak	Operace byla předčasně ukončena - rezervováno pro budoucí použití
3	eLoaderFatalErr	Fatální chyba LOADER, nejspíše došlo k poškození paměti.
4	eLoaderCpuType	Nepodporovaný typ procesoru - rezervováno pro budoucí použití.
5	eLoaderMemType	Nepodporovaný typu paměti, nejspíše nekompatibilní verze protokolu.
6	eLoaderMemWrite	Došlo k chybě při zápisu do paměti.
7	eLoaderOverWrite	Data nebyla zapsána, protože by došlo k poškození (přepsání) LOADER. Změnit adresu a velikost zapisovaného bloku.
8	eLoaderUserWrite	Při zápisu do paměti podporované uživatelskou knihovnou došlo k chybě.
9	eLoaderModuleHeader	Nekompatibilní verze protokolu.
10	eLoaderMemInfoSize	Při čtení byl požadován větší datový blok než je podporováno (Memory Dump), nejspíše nekompatibilní verze protokolu.
11	eLoaderInvalidGetPar	Neplatné parametry při otevírání modulu pro čtení.
12	eLoaderUserRead	Při čtení došlo k chybě v uživatelské knihovně.

13. Zjištění informací o datových modulech

```
function GetModuleDescByName(Name:TModuleName;
                             var apModuleDesc:pModuleDesc):Boolean;
```

Funkce podle jména modulu Name nastaví ukazatel na strukturu popisující požadovaný modul. Pokud požadovaný modul není nalezen, vrací funkce FALSE. V opačném případě vrací TRUE.

```
function GetModuleDescByIx(awIndex:Word;
                           var apModuleDesc:pModuleDesc):Boolean;
```

Funkce podle indexu nastaví ukazatel na strukturu popisující požadovaný modul. Pokud požadovaný modul není nalezen, vrací funkce FALSE. V opačném případě vrací TRUE.

14. Zjištění informací o HW

K zjištění informací o HW slouží program nazývaný **HwDetect** v adresáři **Testy** systémových knihoven, který umožní pomocí funkcí v jednotce **HwSyst** a služeb v **BIOS** určit konfiguraci HW. Takto určená konfigurace nemusí být u zákaznických desek a u řídicího systému KitV40 100%, vzhledem k chybějícím informacím v BIOS. V případě standardních sestav s Kit386EXR a vyšším by vypisované informace měly být bezchybné. Tyto HW informace slouží k vyplnění informací o projektu v programu **TheKing**.

Zjištěné informace lze buď zobrazit na obrazovku řídicího systému nebo na obrazovku terminálu v programu **RTD**.

15. Nahrání nové verze aplikace LOADER - LdrPatch

Pokud je zapotřebí provést nahrání nové verze aplikace LOADER do řídicího systému, lze to provést třemi způsoby: 1. Nahráním nové verze do FLASH paměti pomocí programátoru FLASH pamětí. 2. Nahráním nové verze do FLASH paměti pomocí programu **RTD**. 3. Nahráním nové verze do FLASH pamětí pomocí programu **TheKing** (stejně jako se nahrávají nové verze řídicích aplikací). Pro tento třetí případ je potřeba použít program **LdrPatch** a o něm bude tato kapitola.

Upozornění:

- Při rozmísťování programu **LdrPatch** v RTD se musí jednotka -- *LOADER* -- umístit do modulu se jménem **LdrPatch**. Vzhledem k velikosti tohoto programu lze do tohoto modulu umístit celý program.
- Při rozmísťování programu LOADER v RTD se musí jednotka -- *LOADER* -- umístit do modulu se jménem ROM. Vzhledem k velikosti tohoto programu při použití standardního komunikačního portu lze do tohoto modulu umístit celý program.
- Pokud bude program LOADER používat přenos dat pomocí Ethernet protokolem UDP, je třeba změnit rozmístění modulů v paměti. Pro zjednodušení lze použít direktivu v souboru **sets.inc**, která tyto proměnné nastaví dle požadované konfigurace.

15.1. Přizpůsobení požadavkům aplikace

Protože dodávaná verze programu **LdrPatch** v adresáři **Testy** systémových knihoven je naprosto nezávislá na řídicí aplikaci a programu **LOADER**, musí se nastavit několik proměnných. Pokud se mění tyto proměnné, musí se v některých případech provést upravení souboru LdrCom.mem případně LdrUdp.mem nebo provést nové rozmístění programu **LOADER**. Pokud se tyto úpravy v souboru LdrCom.mem případně LdrUdp.mem neprovedou, 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 programu **LOADER** do řídicího systému zapsána vždy bez ohledu na výpadky systému.
Pokud se tato adresa změní, musí se upravit hodnotu adresy u modulu LdrPatch v souboru LdrCom.mem. Pokud se tato změna neprovede, 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 programu **LOADER**.
Pokud se tato adresa změní, musí se upravit hodnota adresy u modulu ROM v souboru LdrCom.mem. Pokud se tato změna neprovede, nemusí program **LdrPatch** správně fungovat.

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

Adresa umístění nové verze programu **LOADER** podle nastavení v **RTD** při jejím rozmísťování do paměti. Pokud se tato adresa změní, musí se znovu rozmístit program **LOADER**.

15.2. Nahrání nové verze LOADER pomocí programu TheKing

Nahrání nové verze programu LOADER se provede pomocí okna **Controller 1 File Transfer** v menu **Views\Communication states overview**. Při nahrávání se musí uživatel přihlásit jako administrátor programu a komunikovat s programem **LOADER**, jinak je tlačítko **Load *.prj** neaktivní.

Pokud program **TheKing** komunikuje s programem LOADER nebo s nějakou aplikací, tak se v poli s označením ***Running application*** zobrazí nějaký řetězec.

- Při komunikaci s LOADER se zobrazí ***Loader:*** <libovolný text>.
- Při komunikaci s aplikací se zobrazí ***UserApp:*** <libovolný text>.

Přepnutí řídicí aplikace do programu LOADER se provede pomocí stisku tlačítka **Start**.

Identifikace komunikujícího programu se vyvolá pomocí tlačítka **Query.** Pokud se aplikace neidentifikuje, tak tuto funkci nepodporuje a zobrazí se ***Unknown***. Přepnutí aplikace do programu **LOADER**, se může mimo jiné provést daným počtem reset v daném čase.

Stisk tlačítka **Load *.prj** vyvolá dialog pro výběr souboru s koncovkou *.prj. V dialogu se vybere soubor *LdrCom.prj* případně *LdrUdp.prj*. Po jeho přečtení by se měli zobrazit parametry modulů LdrPatch a ROM v dolní části okna **Controller 1 File Transfer**. Pokud při přečtení k žádné chybě, může se postoupit k dalšímu kroku.

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

Tento program 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 programu **LOADER** v poli s označením ***Running application***.

Tip:

- Při nahrávání nové verze LOADER se může použít tlačítko **Automatic program Download**, které najednou provede tyto akce - otevře dialog pro přečtení souboru *.prj, zápis programu **LdrPatch** do paměti řídicího systému a reset systému.