

Knihovna FlashLib

TXV 003 55.01
první vydání
listopad 2009
změny vyhrazeny

Historie změn

Datum	Vydání	Popis změn
Listopad 2009	1	První vydání, popis odpovídá FlashLib_v12

OBSAH

1 Úvod.....	3
2 Datové typy.....	4
3 Konstanty.....	5
4 Globální proměnné.....	6
5 Funkce	6
5.1 Funkce FlashInfo.....	7
5.2 Funkce FlashRead.....	8
5.3 Funkce FlashWrite.....	10
5.4 Funkce GetLastFlashErr.....	12
5.5 Funkce GetLastFlashErrTxt.....	13
6 Funkční bloky.....	14
6.1 Funkční blok fbFlashErase.....	15
6.2 Funkční blok fbFlashSave.....	17
7 Příklad použití.....	18

1 ÚVOD

Knihovna FlashLib je standardně dodávána jako součást programovacího prostředí Mosaic. Knihovna obsahuje funkce a funkční bloky umožňující práci s pamětí flash, která je standardní součástí procesorových jednotek řady K.

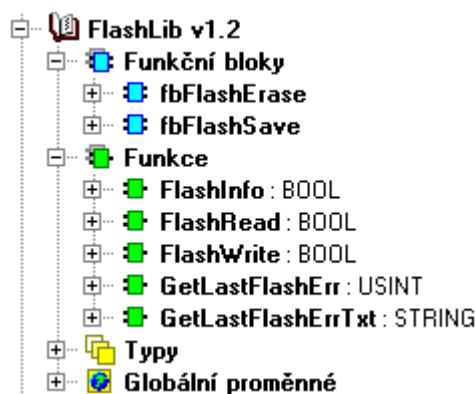
Paměť Flash je nevolatilní (semipermanentní) elektricky programovatelná (zapisovatelná) paměť s libovolným přístupem. Paměť je vnitřně organizována po blocích (segmentech) a na rozdíl od paměti typu EEPROM, lze programovat každý blok samostatně (obsah ostatních bloků je zachován). Paměť se používá jako paměť typu ROM např. pro uložení firmware. Výhodou této paměti je, že ji lze znovu naprogramovat (např. přeprogramování novější verze firmware) bez vyjmutí ze zařízení. Část paměti flash na procesorové jednotce je přístupná pro čtení a zápis z aplikačního programu PLC prostřednictvím knihovny FlashLib.

Výhodou této paměti je její energetická nezávislost, takže data uložená v této paměti nejsou závislá na stavu zálohovací baterie. Paměť flash je navíc integrální součástí procesorové jednotky, což znamená, že ji nelze ze systému odebrat (na rozdíl např. od SD/MMC karty). Obsah této paměti se při nahrávání kódu programu z prostředí Mosaic nemění.

Nevýhodou paměti flash je omezený počet zápisů do paměti, který se typicky pohybuje v řádu 100 000 zápisů. Při zápisu nových dat do paměti je nutné nejprve smazat celý segment paměti, mazání po jednotlivých buňkách paměti není možné.

Paměť flash je tedy vhodná pro uložení takových dat, která se během předpokládané životnosti zařízení mění jen výjimečně. Výrobce neručí za chyby vzniklé překročením povoleného počtu zápisů do paměti flash. Typickým příkladem pro použití paměti flash je uložení strojových konstant stroje, hodnoty nastavení regulačních smyček apod.

Následující obrázek ukazuje strukturu knihovny FlashLib v prostředí Mosaic



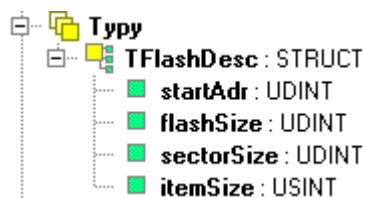
Pokud chceme funkce z knihovny FlashLib použít v aplikačním programu PLC, je třeba nejprve přidat tuto knihovnu do projektu. Knihovna je dodávána jako součást instalace prostředí Mosaic od verze v2.0.21. Simulátor PLC v prostředí Mosaic podporuje simulaci flash paměti od verze v2.2.0.3.

Knihovna FlashLib není podporovaná na systémech TC-650, u systému TC700 nelze knihovnu použít s procesorovými moduly CP-7002, CP-7003 a CP-7005.

Funkce z knihovny FlashLib jsou podporovány v centrálních jednotkách řady K (TC700 CP-7000 a CP-7004, všechny varianty systému Foxtrot) od verze v4.6.

2 DATOVÉ TYPY

V knihovně FlashLib jsou definovány následující datové typy:

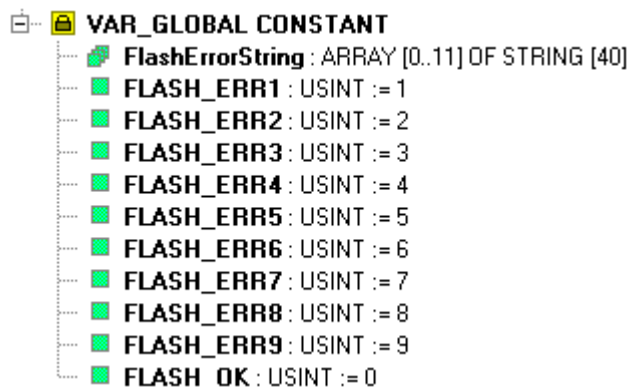


Datový typ *TFlashDesc* je struktura obsahující informace o paměti flash, kterou vrací funkce *FlashInfo* (viz Funkce FlashInfo). Význam jednotlivých položek je následující:

Identifikátor	Typ	Význam
<i>TFlashDesc</i>	STRUCT	Popis paměti flash
<i>.startAdr</i>	UDINT	Adresa začátku paměti flash
<i>.flashSize</i>	UDINT	Celková velikost paměti (počet bytů)
<i>.sectorSize</i>	UDINT	Velikost jednoho sektoru paměti (počet bytů)
<i>.itemSize</i>	USINT	Velikost jedné paměťové buňky (počet bytů)

3 KONSTANTY

V knihovně FlashLib jsou definovány následující konstanty:



Konstanty FLASH_ERR1 až FLASH_ERR9 jsou chybové kódy, které vrátí funkce GetLastFlashErr v případě chyby vzniklé při práci s pamětí flash. Význam těchto kódů je následující:

<i>Identifikátor</i>	<i>Typ</i>	<i>Hodnota</i>	<i>Význam</i>
<i>FLASH_OK</i>	USINT	0	Žádná chyba
<i>FLASH_ERR1</i>	USINT	1	Chybná adresa paměti flash
<i>FLASH_ERR2</i>	USINT	2	Chybná velikost
<i>FLASH_ERR3</i>	USINT	3	Chybný parametr
<i>FLASH_ERR4</i>	USINT	4	Chyba při zápisu do paměti
<i>FLASH_ERR5</i>	USINT	5	Chyba při mazání paměti
<i>FLASH_ERR6</i>	USINT	6	Nedostatek paměti při simulaci flash (simulátor PLC)
<i>FLASH_ERR7</i>	USINT	7	rezerva
<i>FLASH_ERR8</i>	USINT	8	rezerva
<i>FLASH_ERR9</i>	USINT	9	rezerva

4 GLOBÁLNÍ PROMĚNNÉ

V knihovně FlashLib nejsou definovány žádné globální proměnné.

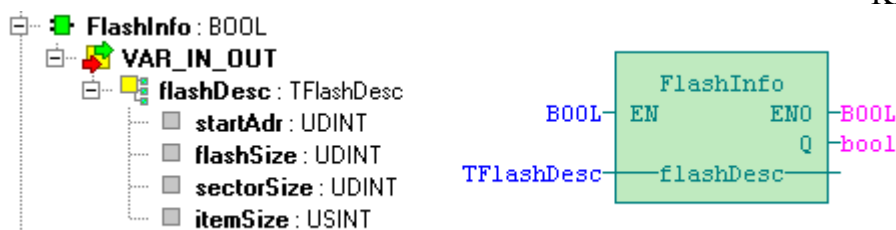
5 FUNKCE

Knihovna FlashLib obsahuje následující funkce:



<i>Funkce</i>	<i>Popis</i>
<i>FlashInfo</i>	Vrátí informace o paměti flash
<i>FlashRead</i>	Načte data z paměti flash a uloží je do proměnné PLC
<i>FlashWrite</i>	Zapíše obsah proměnné do paměti flash
<i>GetLastFlashErr</i>	Vrátí výsledek poslední operace s pamětí flash
<i>GetLastFlashErrMsg</i>	Převede chybový kód na textové hlášení

5.1 Funkce FlashInfo

Knihovna : *FlashLib*

Funkce *FlashInfo* vrátí informace o paměti flash do proměnné uvedené v parametru *flashDesc*. Tato proměnná musí být povinně typu *TflashDesc* (viz Datové typy).

Tato funkce je podporovaná na centrálních jednotkách řady K (TC700 CP-7004, Foxtrot) od verze v4.6. Do knihovny FlashLib je funkce zařazena od verze FlashLib_v10.

Popis proměnných :

	Proměnná	Typ	Význam
VAR_IN_OUT			
	<i>flashDesc</i>	TFlashDesc	Proměnná, do které jsou uloženy následující informace o paměti flash: - Adresa začátku paměti flash - Celková velikost paměti (počet bytů) - Velikost jednoho sektoru paměti (počet bytů) - Velikost jedné paměťové buňky (počet bytů)
FlashInfo			
	Návratová hodnota	BOOL	TRUE pokud se podaří získat informace o flash paměti, jinak FALSE

Příklad programu s voláním funkce *FlashInfo* :

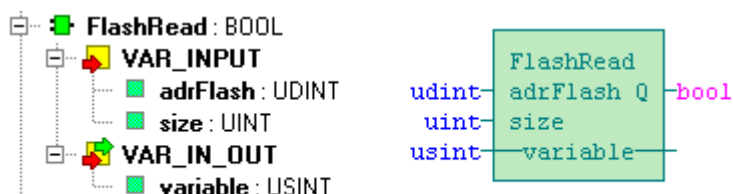
```

PROGRAM FlashInfoExample
VAR
    flashDesc      : TFlashDesc;
    tmp            : BOOL;
    numFlashSectors : UDINT;
END_VAR

tmp := FlashInfo(flashDesc); // get flash info
IF tmp THEN
    numFlashSectors := flashDesc.flashSize / flashDesc.sectorSize;
END_IF;
END_PROGRAM

```

5.2 Funkce FlashRead

Knihovna : *FlashLib*

Funkce *FlashRead* načte data z paměti flash do proměnné, jejíž jméno je uvedeno v parametru *variable*. Velikost načítaných dat musí odpovídat velikosti proměnné.

Tato funkce je podporovaná na centrálních jednotkách řady K (TC700 CP-7004, Foxtrot) od verze v4.6. Do knihovny FlashLib je funkce zařazena od verze FlashLib_v10.

Popis proměnných :

	<i>Proměnná</i>	<i>Typ</i>	<i>Význam</i>
VAR_INPUT			
	<i>adrFlash</i>	UDINT	Počáteční adresa v paměti flash, ze které se bude číst
	<i>size</i>	UINT	Počet načítaných bytů
VAR_IN_OUT			
	<i>variable</i>	USINT	Proměnná, do které budou uložena data přečtená z paměti flash
FlashRead			
	<i>Návratová hodnota</i>	BOOL	TRUE pokud se podaří načíst požadovaná data, jinak FALSE

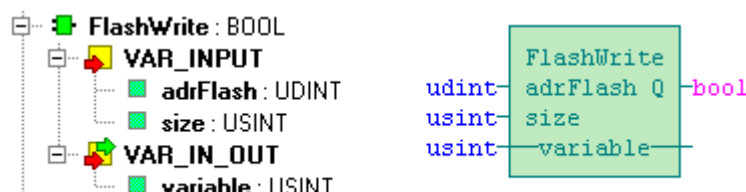
Příklad programu s voláním funkce *FlashRead* :

```
PROGRAM FlashReadExample
VAR CONSTANT
    SIZE_FLASH_REGION : INT := 32;
END_VAR
VAR
    flashDesc    : TFlashDesc;
    tmp          : BOOL;
    flashMirror   : ARRAY[0..SIZE_FLASH_REGION-1] OF BYTE;
    result       : STRING;
END_VAR

tmp := FlashInfo(flashDesc);           // get flash info
IF flashDesc.flashSize <> 0 THEN
    // read 32 bytes from flash
    tmp := FlashRead( adrFlash := flashDesc.startAdr,
                      size      := sizeof(flashMirror),
                      variable := void(flashMirror));
    result := 'Flash is available';
ELSE
    result := 'Flash is not available';
END_IF;
END_PROGRAM
```

Viz také Funkce FlashInfo

5.3 Funkce FlashWrite

Knihovna : *FlashLib*

Funkce *FlashWrite* zapíše data uložená v proměnné *variable* do paměti flash. Velikost zapisovaných dat musí odpovídat velikosti proměnné. Na jedno zavolání funkce *FlashWrite* lze do paměti flash zapsat maximálně 255 bytů dat. Pokud chceme uložit větší množství dat, je nutné použít funkční blok *fbFlashSave*. Nutnou podmínkou pro úspěšný zápis je smazaná paměť flash.

Tato funkce je podporovaná na centrálních jednotkách řady K (TC700 CP-7004, Foxtrot) od verze v4.6. Do knihovny FlashLib je funkce zařazena od verze FlashLib_v10.

Popis proměnných :

	Proměnná	Typ	Význam
VAR_INPUT			
	<i>adrFlash</i>	UDINT	Počáteční adresa v paměti flash, do které se bude zapisovat
	<i>size</i>	USINT	Počet zapisovaných bytů
VAR_IN_OUT			
	<i>variable</i>	USINT	Proměnná, ze které se budou číst data zapisovaná do paměti flash
FlashWrite			
	<i>Návratová hodnota</i>	BOOL	TRUE pokud se podaří zapsat požadovaná data, jinak FALSE

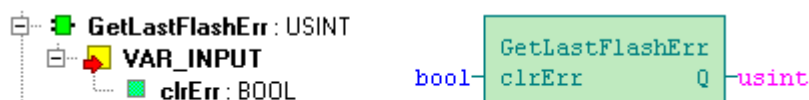
Příklad programu s voláním funkce *FlashWrite* :

```
PROGRAM FlashWriteExample
VAR
  flashDesc    : TFlashDesc;
  EraseSector  : fbFlashErase;
  flashConst   : ARRAY[0..31] OF BYTE :=
    [ 0,1,2,3,4,5,6,7,8,9,
      10,11,12,13,14,15,16,17,18,19,
      20,21,22,23,24,25,26,27,28,29,
      30,31 ];
  chk          : BYTE;
END_VAR

FlashInfo(flashDesc);           // get flash info
IF flashDesc.flashSize <> 0 THEN
  FlashRead( adrFlash := 0, size := 1, variable := void(chk));
  IF chk <> flashConst[0] THEN
    // erase first sector
    EraseSector( exec := TRUE, adrFlash := 0);
    IF EraseSector.done THEN
      // write 32 bytes to flash
      FlashWrite( adrFlash := 0, size := 32, variable := void(flashConst));
    END_IF;
  END_IF;
END_IF;
END_PROGRAM
```

Viz také Funkce FlashInfo, Funkce FlashRead, Funkční blok fbFlashErase

5.4 Funkce *GetLastFlashErr*

Knihovna : *FlashLib*

Funkce *GetLastFlashErr* vrací hodnotu vnitřní globální proměnné, která je nastavena vždy, když v některé z funkcí v knihovně *FlashLib* dojde k chybě. Pokud byla funkce vykonaná bez chyby, funkce *GetLastFlashErr* vrátí hodnotu 0 (konstanta *FLASH_OK*).

Tato funkce je podporovaná na centrálních jednotkách řady K (TC700 CP-7004, Foxtrot) od verze v4.6. Do knihovny *FlashLib* je funkce zařazena od verze *FlashLib_v10*.

Popis proměnných :

	Proměnná	Typ	Význam
VAR_INPUT			
	<i>clrErr</i>	BOOL	Požadavek na vynulování případné chyby. Je-li TRUE, chyba je po přečtení vynulovaná
GetLastFlashErr			
	Návratová hodnota	USINT	Chybový kód (viz Konstanty)

Příklad programu s voláním funkce *GetLastFlashErr* :

```

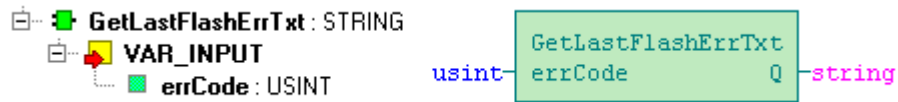
PROGRAM GetFlashErrExample
VAR
  flashDesc      : TFlashDesc;
  flashData      : ARRAY[0..31] OF BYTE;
  errCode        : USINT;
  result         : STRING;
END_VAR

IF FlashInfo(flashDesc) THEN
  // read 32 bytes from flash
  FlashRead( adrFlash := 0, size := 32, variable := void(flashData));
END_IF;
errCode := GetLastFlashErr();
result := GetLastFlashErrTxt( errCode);
END_PROGRAM

```

Viz také Funkce *FlashInfo*, Funkce *FlashRead*, Funkce *GetLastFlashErrTxt*

5.5 Funkce *GetLastFlashErrTxt*

Knihovna : *FlashLib*

Funkce *GetLastFlashErrTxt* převede chybový kód získaný funkcí *GetLastFlashErr* na textové hlášení.

Tato funkce je podporovaná na centrálních jednotkách řady K (TC700 CP-7004, Foxtrot) od verze v4.6. Do knihovny FlashLib je funkce zařazena od verze FlashLib_v10.

Popis proměnných :

	Proměnná	Typ	Význam
VAR_INPUT			
	<i>errCode</i>	USINT	Chybový kód získaný funkcí <i>GetLastFlashErr</i>
GetLastFlashErrTxt			
	<i>Návratová hodnota</i>	STRING	Textový popis chyby

Příklad programu s voláním funkce *GetLastFlashErrTxt* :

```

PROGRAM GetFlashErrTxtExample
VAR
  flashDesc      : TFlashDesc;
  flashData      : ARRAY[0..31] OF BYTE;
  result         : STRING;
END_VAR

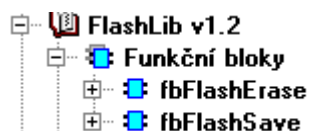
IF FlashInfo(flashDesc) THEN
  // read 32 bytes from flash
  FlashRead( adrFlash := 0, size := 32, variable := void(flashData));
END_IF;
result := GetLastFlashErrTxt( errCode := GetLastFlashErr());
END_PROGRAM

```

Viz také Funkce *FlashInfo*, Funkce *FlashRead*, Funkce *GetLastFlashErr*

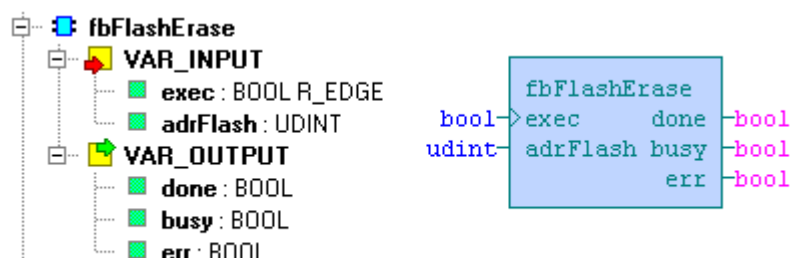
6 FUNKČNÍ BLOKY

V knihovně FlashLib jsou definovány následující funkční bloky:



<i>Funkční blok</i>	<i>Popis</i>
<i>fbFlashErase</i>	Vymaže jeden sektor paměti flash
<i>fbFlashSave</i>	Zapíše blok dat do paměti flash

6.1 Funkční blok fbFlashErase

Knihovna : *FlashLib*

Funkční blok *fbFlashErase* slouží k vymazání jednoho segmentu paměti flash. To je podmínkou pro následný zápis dat do paměti flash. Tato operace může trvat několik cyklů PLC. Po vymazání budou buňky paměti obsahovat hodnoty 16#FF.

Tento funkční blok je podporován na centrálních jednotkách řady K (TC700 CP-7004, Foxtrot) od verze v4.6. Do knihovny FlashLib je blok zařazen od verze FlashLib_v12.

Popis proměnných :

	Proměnná	Typ	Význam
VAR_INPUT			
	<i>exec</i>	BOOL R_EDGE	Náběžná hrana této proměnné odstartuje mazání segmentu paměti flash
	<i>adrFlash</i>	UDINT	Adresa segmentu paměti flash, který bude vymazán
VAR_OUTPUT			
	<i>done</i>	BOOL	TRUE znamená, že segment paměti flash byl úspěšně vymazán
	<i>busy</i>	BOOL	TRUE znamená, že probíhá mazání segmentu
	<i>err</i>	BOOL	TRUE znamená, že vymazání segmentu se nepodařilo a bylo ukončeno

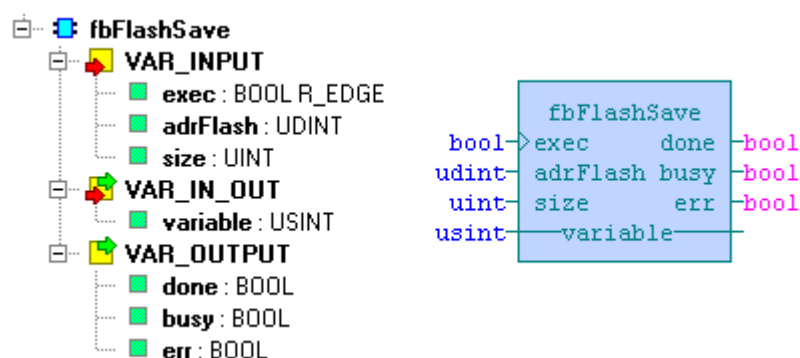
Příklad programu s funkčním blokem *fbFlashErase* :

```
PROGRAM FlashEraseExample
VAR
    flashDesc      : TFlashDesc;
    EraseSector    : fbFlashErase;
    flashData      : ARRAY[0..31] OF USINT;
    i              : INT;
    erased         : BOOL;
END_VAR

FlashInfo(flashDesc);           // get flash info
IF flashDesc.flashSize <> 0 THEN
    IF NOT EraseSector.busy THEN
        FlashRead( adrFlash := 0, size := 32, variable := void(flashData));
        FOR i := 0 TO 31 DO
            IF flashData[i] <> 16#FF THEN exit; END_IF;
        END_FOR;
        erased := i = 32;
    END_IF;
    IF not erased THEN
        // erase first sector
        EraseSector( exec := TRUE, adrFlash := 0);
        IF EraseSector.done THEN
            erased := TRUE;
        END_IF;
    END_IF;
END_IF;
END_PROGRAM
```

Viz také Funkce FlashInfo, Funkce FlashRead

6.2 Funkční blok *fbFlashSave*

Knihovna : *FlashLib*

Funkční blok *fbFlashSave* slouží k zápisu bloku dat do paměti flash. Segment paměti, do kterého budou data uložena, musí být před zápisem dat vymazán. Operace zápisu dat do paměti může trvat několik cyklů PLC.

Tento funkční blok je podporován na centrálních jednotkách řady K (TC700 CP-7004, Foxtrot) od verze v4.6. Do knihovny FlashLib je blok zařazen od verze FlashLib_v12.

Popis proměnných :

	Proměnná	Typ	Význam
VAR_INPUT			
	<i>exec</i>	BOOL R_EDGE	Náběžná hrana této proměnné odstartuje zápis dat do paměti flash
	<i>adrFlash</i>	UDINT	Adresa paměti flash, od které budou data zapsána
	<i>size</i>	UINT	Délka zapisovaných dat (počet bytů)
VAR_IN_OUT			
	<i>variable</i>	USINT	Proměnná, ve které jsou připravena zapisovaná data
VAR_OUTPUT			
	<i>done</i>	BOOL	TRUE znamená, že data byla úspěšně uložena do paměti flash
	<i>busy</i>	BOOL	TRUE znamená, že probíhá zápis dat do paměti
	<i>err</i>	BOOL	TRUE znamená, že zápis dat do paměti se nepodařil a byl ukončen

Příklad programu s funkčním blokem *fbFlashSave* viz následující kapitola.

Viz také Funkce FlashInfo

7 PŘÍKLAD POUŽITÍ

Předpokládejme následující zadání: chceme napsat program PLC systému tak, aby bylo možné sledovat stavy binárních vstupů a hodnoty analogových vstupů prostřednictvím webového prohlížeče. Program má být připraven tak, aby se každý vstupní signál dal pojmenovat podle konkrétního projektu, u binárních signálů musí jít zadat texty zobrazované ve stavu 0 a 1, zobrazení musí jít podmíněně vypnout, analogové hodnoty musí být možné před zobrazením přepočítat ($y = kx + q$) a musí jít zadat počet cifer a počet desetinných míst, které budou zobrazeny. Uvedené nastavení musí být možné beze změny aplikačního programu PLC. To znamená, že informace o zobrazení musí být uloženy v proměnných PLC. Lze to samozřejmě řešit zálohovanými proměnnými VAR_GLOBAL RETAIN, ale v případě studeného restartu PLC nebo při výpadku baterie se nastavené hodnoty nahradí inicializačními hodnotami. Nabízí se tedy uložit informace o zobrazení do paměti flash, která je energeticky nezávislá.

V systémech Foxtrot jsou k dispozici dva segmenty v paměti flash, každý s velikostí 64 Kbytů. Do jednoho segmentu se tedy s jistotou vejdu všechny potřebné proměnné, které obsahují informace o způsobu zobrazení. Pokud se způsob zobrazení změní, uloží se nové informace do toho segmentu, který obsahuje starší vzorek dat. Aby bylo možné rozhodnout, který ze segmentů obsahuje starší data, je třeba kromě vlastních dat ukládat do paměti flash jakousi hlavičku, která má následující strukturu:

```

TYPE
  THeaderPermanent : STRUCT
    length          : UINT;
    version         : UINT;
    dateTime        : DT;
    modification    : UDINT;
    crc             : UINT;
  END_STRUCT;
END_TYPE

```

Význam položek ve struktuře *THeaderPermanent* je následující:

Identifikátor	Typ	Význam
<i>THeaderPermanent</i>	STRUCT	Hlavička uložených dat
<i>.length</i>	UINT	Počet uložených bytů
<i>.version</i>	UINT	Verze uložených dat (informace o struktuře uložených dat)
<i>.dateTime</i>	DT	Datum a čas uložení dat
<i>.modification</i>	UDINT	Číslo modifikace
<i>.crc</i>	UINT	Zabezpečovací znak

Do flash paměti se budou ukládat informace, které budou řídit způsob zobrazení binárních a analogových vstupů PLC systému ve webové stránce.

Pro každý binární vstup budou mít informace následující strukturu:

```

TYPE
  TDigitalItem : STRUCT
    select      : BOOL;           // channel OFF/ON
    onChange    : BOOL;           // save value if changed
    name        : STRING;         // name of item
    viewAsTxt   : BOOL;           // show VALUE or TEXT
    txtOn       : STRING[16];     // text for value - TRUE
    txtOff      : STRING[16];     // text for value - FALSE
  END_STRUCT;
END_TYPE

```

Význam položek ve struktuře *TDigitalItem* je následující:

Identifikátor	Typ	Význam
<i>TDigitalItem</i>	STRUCT	Popis zobrazení binárního vstupu
<i>.select</i>	BOOL	Informace o vstupu budou zobrazeny
<i>.onChange</i>	BOOL	Vyhodnocovat změnu signálu
<i>.name</i>	STRING	Pojmenování binárního vstupu
<i>.viewAsTxt</i>	BOOL	Zobrazit text místo binární hodnoty
<i>.txtOn</i>	STRING[16]	Text zobrazený při stavu TRUE
<i>.txtOff</i>	STRING[16]	Text zobrazený při stavu FALSE

Pro každý analogový vstup budou mít informace následující strukturu:

```

TYPE
  TAnalogItem : STRUCT
    select      : BOOL;           // channel OFF/ON
    onChange    : BOOL;           // save value if changed
    name        : STRING;         // name of item
    scale       : REAL := 1;      // multiplier
    offset      : REAL;           // val := input * scale + offset
    delta       : REAL;           // sensitivity
    cif         : USINT;          // number of digit
    fract       : USINT;          // decimal places
    format      : STRING[8];      // format string
  END_STRUCT;
END_TYPE

```

Význam položek ve struktuře *TAnalogItem* je následující:

Identifikátor	Typ	Význam
<i>TAnalogItem</i>	STRUCT	Popis zobrazení analogového vstupu
<i>.select</i>	BOOL	Informace o vstupu budou zobrazeny
<i>.onChange</i>	BOOL	Vyhodnocovat změnu signálu

<i>.name</i>	STRING	Pojmenování analogového vstupu
<i>.scale</i>	REAL	Násobící koeficient
<i>.offset</i>	REAL	Posunutí zobrazované hodnoty
<i>.delta</i>	REAL	Necitlivost pro vyhodnocení změny
<i>.cif</i>	USINT	Počet zobrazených cifer
<i>.fract</i>	USINT	Počet zobrazených desetinných míst
<i>.format</i>	STRING[8]	Formátovací řetězec

Pro informace o zobrazení binárních a analogových vstupů jsou v programu PLC založeny pole proměnných *digitalItem* a *analogItem*, kde každý prvek pole obsahuje informace o zobrazení jednoho vstupu. Tyto proměnné budou při každé změně uloženy do paměti flash. Uvedené proměnné jsou globální a mají nastaveny inicializační hodnoty pro případ prvního spuštění programu, kdy je paměť flash prázdná. Pro snazší výpočet velikosti ukládaných dat je na začátku dat ukládaných do paměti flash nadeklarovaná proměnná *beginPermanent* a na konci proměnná *endPermanent*. Odečtením adres těchto proměnných lze snadno získat aktuální velikost ukládaných dat.

```

VAR_GLOBAL CONSTANT
  MAX_DIGITAL_CHAN : USINT := 7;
  MAX_ANALOG_CHAN  : USINT := 3;
  DATA_VERSION    : USINT := 1;
END_VAR

VAR_GLOBAL
  beginPermanent : USINT;
  digitalItem : ARRAY [0..MAX_DIGITAL_CHAN] OF TDigitalItem :=
    [ (name := 'digital channel 0', txtOn := 'ON', txtOff := 'OFF'),
      (name := 'digital channel 1', txtOn := 'ON', txtOff := 'OFF'),
      (name := 'digital channel 2', txtOn := 'ON', txtOff := 'OFF'),
      (name := 'digital channel 3', txtOn := 'ON', txtOff := 'OFF'),
      (name := 'digital channel 4', txtOn := 'ON', txtOff := 'OFF'),
      (name := 'digital channel 5', txtOn := 'ON', txtOff := 'OFF'),
      (name := 'digital channel 6', txtOn := 'ON', txtOff := 'OFF'),
      (name := 'digital channel 7', txtOn := 'ON', txtOff := 'OFF') ];
  analogItem : ARRAY [0..MAX_ANALOG_CHAN] OF TAnalogItem :=
    [ (name := 'analog channel 0', cif := 6, fract := 1, format := '%6.1f'),
      (name := 'analog channel 1', cif := 6, fract := 1, format := '%6.1f'),
      (name := 'analog channel 2', cif := 6, fract := 1, format := '%6.1f'),
      (name := 'analog channel 3', cif := 6, fract := 1, format := '%6.1f') ];
  endPermanent : USINT;
END_VAR

```

Hlavní program, pak po restartu PLC volá funkci *LoadPermanentData()*, která načte naposledy uložené informace o zobrazení z paměti flash do proměnných *digitalItem* a *analogItem*, resp. do všech proměnných deklarovaných mezi *beginPermanent* a *endPermanent*.

Pokud se proměnné *digitalItem* a *analogItem* změní (například z web stránky nebo z operátorského panelu) a nastaví se proměnná *SaveParams*, uloží se nové hodnoty do paměti flash, což zajišťuje volání funkčního bloku *SavePermanentData()*.

Z příkladu programu *prgMain* je vidět, že ukládání a načítání dat není závislé na struktuře ukládaných dat (díky „triku“ s proměnnými *beginPermanent* a *endPermanent*). Uvedené funkční bloky lze tedy použít pro ukládání dat s libovolnou strukturou. Při změně struktury ukládaných dat je třeba změnit konstantu *DATA_VERSION*, která definuje verzi struktury ukládaných dat. Tato konstanta je ukládána do

paměti flash v hlavičce *TheaderPermanent* a při načítání z flash paměti se načtou pouze data s odpovídající verzí.

```

PROGRAM prgMain
VAR
  SavePermanentData : fbSavePermanentData;
  saveParams        : BOOL;
  dataSize          : UINT;
END_VAR

// 1st cycle after power on - load parameters from flash
IF System_S.S2_3 OR System_S.S2_4 THEN
  dataSize := UDINT_TO_UINT( PTR_TO_UDINT( ADR( endPermanent)) -
                             PTR_TO_UDINT( ADR( beginPermanent)));
  LoadPermanentData(length := dataSize, version := DATA_VERSION,
                    data := beginPermanent);
END_IF;

// save new parametrers to flash after change
IF saveParams THEN
  SavePermanentData( exec := 1, length := dataSize, version := DATA_VERSION,
                    data := beginPermanent);
  IF SavePermanentData.done OR SavePermanentData.err THEN
    saveParams := FALSE;
  END_IF;
  RETURN;      // wait for finish of SavePermanentData
END_IF;

// next program
// ...

END_PROGRAM

```

Funkční blok pro uložení dat do paměti flash *fbSavePermanentData* využívá toho, že v systému Foxtrot jsou k dispozici dva velké segmenty v paměti flash. Na začátku se přečtou hlavičky uložené na začátku těchto segmentů a podle jejich obsahu se zvolí ten segment paměti, který obsahuje starší data. Poté se připraví hlavička pro nově ukládaná data včetně datumu a času uložení dat a výpočtu kontrolního crc znaku, vymaže se zvolený segment paměti, do smazaného segmentu se zapíše data a nakonec se zapíše hlavička na začátek segmentu. Uvedené pořadí je důležité z toho důvodu, že během programování může dojít k výpadku napájení PLC systému. Pokud se nestihne před výpadkem napájení zapsat hlavičku do flash, budou data pokládána za neplatná a po zapnutí napájení se data přečtou z druhého segmentu (pokud je obsahuje).

```

FUNCTION_BLOCK fbSavePermanentData
VAR_INPUT
  exec      : BOOL;
  length    : UINT;      // requested size
  version   : UINT;      // actual version
END_VAR
VAR_IN_OUT
  data      : USINT;     // source variable
END_VAR
VAR_OUTPUT
  done      : BOOL;
  busy      : BOOL;
  err       : BOOL;
END_VAR

```

```

VAR
    execRedge      : R_TRIG;
    descriptor      : TFlashDesc;
    modification    : UDINT;
    numFlashSect    : UDINT;
    dataHeader,
    dataHeader1,
    dataHeader2      : THeaderPermanent;
    startAdr        : UDINT;
    EraseSector      : fbFlashErase;
    SaveToFlash      : fbFlashSave;
    clrSector,
    saveHeader,
    saveData         : BOOL;
END_VAR

execRedge(CLK := exec);
IF execRedge.Q THEN
    done := FALSE; busy := FALSE; err := FALSE;
    clrSector := FALSE; saveHeader := FALSE; saveData := FALSE;
    // try to find sector with last saved data
    IF FlashInfo(flashDesc := descriptor) THEN
        numFlashSect := descriptor.flashSize / descriptor.sectorSize;
        startAdr := descriptor.startAdr;
        IF numFlashSect > 1 THEN
            // read headers from first two sectors
            FlashRead( adrFlash := startAdr, size := sizeof(THeaderPermanent),
                      variable := void(dataHeader1));
            FlashRead( adrFlash := startAdr+descriptor.sectorSize,
                      size := sizeof(THeaderPermanent), variable := void(dataHeader2));
            IF dataHeader1.length = 16#FFFF THEN
                startAdr := descriptor.startAdr;
                modification := dataHeader2.modification;
            ELSE
                IF dataHeader2.length = 16#FFFF THEN
                    startAdr := descriptor.startAdr+descriptor.sectorSize;
                    modification := dataHeader1.modification;
                ELSE
                    IF dataHeader1.dateTime < dataHeader2.dateTime THEN
                        startAdr := descriptor.startAdr;
                        modification := dataHeader2.modification;
                    ELSE
                        startAdr := descriptor.startAdr+descriptor.sectorSize;
                        modification := dataHeader1.modification;
                    END_IF;
                END_IF;
            END_IF;
        END_IF;
        dataHeader.length := length; // prepare new header
        dataHeader.version := version;
        dataHeader.dateTime := GetDateTime();
        dataHeader.modification := modification + 1;
        dataHeader.crc := CRC16(ptr := ADR(data), length := length);
        clrSector := TRUE; busy := TRUE;
    END_IF;
END_IF;
IF busy THEN
    IF clrSector THEN
        EraseSector(exec := 1, adrFlash := startAdr); // erase the older sector
        IF EraseSector.err THEN
            busy := FALSE; err := TRUE; // flash erasing error
        END_IF;
        IF EraseSector.done THEN
            clrSector := FALSE; saveData := TRUE; // now we will program data
        END_IF;
    END_IF;
END_IF;

```

```

IF saveData THEN
  SaveToFlash(exec := 1, adrFlash := startAdr + sizeof(THeaderPermanent),
    size := length, variable := data);
  IF SaveToFlash.err THEN
    busy := FALSE; err := TRUE; // flash programming error
  END_IF;
  IF SaveToFlash.done THEN
    SaveToFlash(exec := 0, variable := void(dataHeader)); // clear exec
    saveData := FALSE; saveHeader := TRUE; // and now we will program header
  END_IF;
END_IF;
IF saveHeader THEN
  SaveToFlash(exec := 1, adrFlash := startAdr,
    size := sizeof(THeaderPermanent), variable := void(dataHeader));
  IF SaveToFlash.err THEN
    busy := FALSE; err := TRUE; // flash programming error
  END_IF;
  IF SaveToFlash.done THEN
    busy := FALSE; done := TRUE; // end of programming
  END_IF;
END_IF;
IF NOT busy THEN
  // end programming of flash
  clrSector := FALSE; saveData := FALSE; saveHeader := FALSE;
  execRedge(CLK := 0);
  EraseSector(exec := 0, adrFlash := 0);
  SaveToFlash(exec := 0, variable := void(dataHeader));
END_IF;
END_FUNCTION_BLOCK

```

Funkce pro načtení dat z paměti flash *LoadPermanentData* předpokládá, že data byla uložena funkčním blokem *fbSavePermanentData* a tudíž na začátku každého segmentu je uložena hlavička s informacemi o množství uložených dat, času a datumu uložení, verzi uložených dat a crc zabezpečení. Funkce na začátku vybere segment s novějšími daty, zkontroluje zda odpovídá požadovaná verze, zkontroluje platnost uložených dat (přepočítá crc znak). Pokud je vše v pořádku, zkopíruje data z paměti flash do proměnných v PLC programu. Pokud nevyjde některá z kontrol (např. při prvním spuštění programu, kdy flash paměť prázdná, nebo pokud neodpovídá verze uložených dat) pak funkce nenačte žádná data a proměnné v PLC programu obsahují hodnoty z inicializace.

```

FUNCTION LoadPermanentData : BOOL
VAR_INPUT
  length      : UINT; // requested size
  version     : UINT; // requested version
END_VAR
VAR_IN_OUT
  data        : USINT; // destination variable
END_VAR
VAR
  descriptor   : TFlashDesc;
  numFlashSect : UDINT;
  dataHeader,
  dataHeader1,
  dataHeader2  : THeaderPermanent;
  startAdr     : UDINT;
  crc          : WORD;
  addr         : UDINT;
  val          : BYTE;
  i, j         : UINT;
END_VAR

```

```

LoadPermanentData := FALSE;
// try to find sector with last saved data
IF FlashInfo(flashDesc := descriptor) THEN
    numFlashSect := descriptor.flashSize / descriptor.sectorSize;
    startAdr := descriptor.startAdr;
    IF numFlashSect > 1 THEN
        // read headers from first two sectors
        FlashRead( adrFlash := startAdr, size := sizeof(THeaderPermanent),
            variable := void(dataHeader1));
        FlashRead( adrFlash := startAdr+descriptor.sectorSize,
            size := sizeof(THeaderPermanent), variable := void(dataHeader2));
        IF dataHeader1.length <> 16#FFFF AND dataHeader2.length <> 16#FFFF THEN
            IF dataHeader1.dateTime < dataHeader2.dateTime THEN
                startAdr := descriptor.startAdr + descriptor.sectorSize;
            END_IF;
        ELSE
            IF dataHeader2.length <> 16#FFFF THEN
                startAdr := descriptor.startAdr + descriptor.sectorSize;
            END_IF;
        END_IF;
    END_IF;
    // read header from selected sector
    FlashRead( adrFlash := startAdr, size := sizeof(THeaderPermanent),
        variable := void(dataHeader));
    // copy data from sector to variable
    IF dataHeader.version = version AND dataHeader.length <> 16#FFFF THEN
        startAdr := startAdr + sizeof(THeaderPermanent);
        // check crc
        crc := 16#FFFF; addr := startAdr;
        FOR i := 1 TO dataHeader.length DO
            FlashRead( adrFlash := addr, size := 1, variable := void(val));
            crc := crc XOR BYTE_TO_WORD(val);
            FOR j := 0 TO 7 DO
                IF (crc AND 16#0001) = 1 THEN
                    crc := SHR( IN := crc, N := 1); crc := crc XOR 16#A001;
                ELSE
                    crc := SHR( IN := crc, N := 1);
                END_IF;
            END_FOR;
            addr := addr + 1;
        END_FOR;
        IF WORD_TO_UINT(crc) = dataHeader.crc THEN
            IF dataHeader.length >= length THEN
                FlashRead( adrFlash := startAdr, size := length,
                    variable := void(data));
            ELSE
                FlashRead( adrFlash := startAdr, size := dataHeader.length,
                    variable := void(data));
            END_IF;
            LoadPermanentData := TRUE;
        END_IF;
    END_IF;
END_IF;
END_FUNCTION

```

Funkce *Crc16* je pomocná funkce pro výpočet zabezpečovacího znaku, kterým se kontroluje integrita dat uložených v paměti flash.

```

FUNCTION Crc16 : UINT
//-----
// Function calculate polynomial (x16 + x15 + x2 + 1) of byte array
// Returns value : UINT polynomial

```

```
//
VAR_INPUT
    ptr      : PTR_TO_USINT;
    length   : UINT;
END_VAR
VAR
    valCRC   : WORD;
    i, j     : UINT;
    ptrMsg   : PTR_TO_BYTE;
END_VAR

valCRC := 16#FFFF; i := length; ptrMsg := ptr;
WHILE i > 0 DO
    valCRC := valCRC XOR BYTE_TO_WORD( ptrMsg^);
    ptrMsg := ptrMsg + 1;
    FOR j := 0 TO 7 DO
        IF (valCRC AND 16#0001) = 1 THEN
            valCRC := SHR( valCRC, 1);
            valCRC := valCRC XOR 16#A001;
        ELSE
            valCRC := SHR( valCRC, 1);
        END_IF;
    END_FOR;
    i := i - 1;
END_WHILE;
CRC16 := WORD_TO_UINT( valCRC);
END_FUNCTION
```

Funkce *LoadPermanentData* a funkční blok *fbSavePermanentData* využívají knihovnu FlashLib a lze je použít pro ukládání a načítání dat, kterými se např. konfiguruje chování PLC při řízení technologie. Výhodou je, že data jsou uložena mimo program PLC a stejný program PLC se v různých aplikacích může chovat různě.

Závěrem je třeba připomenout skutečnost, že paměť flash má garantovaný pouze omezený počet zápisů (cca 100 000) a není tudíž vhodná pro cycklické ukládání dat.