```
/***************************************************************************
 *  File:   flashpgm.dsp - contains flash programming support routines
 *  Author: Sanjay Gupta
 *  Date:   Tue Jul  9 11:10:56 IST 1996
 *  Bugs:
 *  Change Log:  Changes made for 4mb flash,K.Narayanan 17/5/99
 *          <Changes>
 ***************************************************************************/
.module FlashPgmMod;

/* ------------------------------------------------------------------------ */
/* Include Files */
/* ------------- */
#include <../stdinc/system.h>
#include <../stdinc/bufq.h>
#include <../stdinc/def2181.h>
#include <../stdinc/global.h>
#include <../stdinc/globid.h>
#include <../stdinc/led.h>
#include <../stdinc/spy.h>
#include <../stdinc/timer.h>
#include <../stdinc/bbireg.h>
#include <../iwu/iwuglob.h>

/* ------------------------------------------------------------------------ */
/* Routines provided entry */
/* ---------------------- */
.entry InitFlashPgmPg_;
.entry FlashSchedularPg_;
.entry FlashMsgProcPg_;
.entry FlashPgmTmrExpHdlrPg_;
/* ------------------------------------------------------------------------ */
/* External Functions */
/* ------------------ */
.external Freeze_;
.external MsgRout_;
.external WatchDogTgr_;
.external SlaveSchedular_;
/* ------------------------------------------------------------------------ */
/* External Variables */
/* ------------------ */
.external slaveQ_;
.external slvIwuQ_;
.external phySlaveTxFlag;
.external flashQ_;     /* Q to store the flash msg from RS232 or OMC */
.external flashIwuQ_;  /* Q to store the OAF flash msg to IWU */
.external flashStatus_;      /* flag to have the current flash status */
/* ------------------------------------------------------------------------ */
/* Local Functions */
/* --------------- */

/* ------------------------------------------------------------------------ */
/* Local Variables */
/* --------------- */
.var/dm/ram flashRcvBufIdx;        /* tmp storage of rcv buf idx */
.global          flashRcvBufIdx;

.var/dm/ram flashPgmState;         /* flash pgminng state */
.global          flashPgmState;
```

```
.var/dm/ram flashPgmCmd;           /* temp store: cmd wrd for flash */
.var/dm/ram timerExpIdentity;/* stores the id of running timer*/
.var/dm/ram flashPgmSrcId;         /* programming seq src Id */
.var/dm/ram flashSendMsg;          /* msg sub cmd to be sent */
.var/dm/ram flashTxBufIdx;         /* tmp storage of tx buf idx */
.var/dm/ram flashPgmStateEvent;    /* flash pgminng state */
.var/dm/ram flashPgmAddrHi;        /* Flash addr lo for curr byte access*/
.var/dm/ram flashPgmAddrLo;        /* Flash addr lo for curr byte access*/
.var/dm/ram flashRecLen;           /* no of valid bytes in flashRecData */
.var/dm/ram flashRecPtr;           /* ptr to curr byte in flashRecData */
.var/dm/ram flashRecData[64];/* seq of data bytes read|to-be-pgmed*/

.var/dm/ram flashIwuMsg;           /* store msg to the IWU layer */
.var/dm/ram flashIwuIdx;

.var/dm/ram retryCntInFlashXfer;
.var/dm/ram     blChStSector;
.var/dm/ram flashPgmWaitDelayCnt;
.var/dm/ram rdHiAddr;
.var/dm/ram wrHiAddr;

!.var/dm/ram     maxLoopCntHi;
!.var/dm/ram     maxLoopCntLo;

.var/dm/ram     flashErrReason;
.var/dm/ram abortFromSrc;    /* set if the abort is from the OMC */
#if 0 /* for time caluculation of byte pgming */
.var/dm/ram maxBitCntHi;
.var/dm/ram maxBitCntLo;
.var/dm/ram startBitCnt;
#endif

/* ---------------------------------------------------------------------- */
/* Local Constants */
/* --------------- */

      /* Message Format Offsets */
      /*    0: Control 1: Length        */
      /*    2: Dest Id 3: Source Id           */
      /*    4: Module Id     -> UPDT_MODULE_ID*/
      /*    5: Msg Type -> DOWN_LOAD_BNM */
      /*    6: Sub Cmd  -> BNM_DN_LD_[ERROR|START|READ|WRITE|STOP] */
      /*    7...N-3: BNM Information          */
      /*    N-2: IntraDIU DLC Chksum 0        */
      /*    N-1: IntraDIU DLC Chksum 1        */


      /* BNM Info: for Motorola S Format */
      /*          ----------------- */
      /* 0: format indicator 'S' Motorola S */
      /* 1: recType - 1: 2 byte addr, 2: 3 byte addr, 9: end of file */
      /* 2: length (including address, data & chksum) */
      /* 3..5: 3 byte addr (hi byte first) for S2 records */
      /* 6..N-2: data bytes */
      /* N-1: chksum 1's complement of bin sum of len, addr & data bytes */

      /* BNM Info: for Intel Hex Format */
      /*          ---------------- */
      /* 0: format indicator ':' Intel Hex */
      /* 1: length (including data only) */
      /* 2..3: 2 byte addr (hi byte first) */
      /* 4: recType - 0: data rec, 1: end of file */
      /* 5..N-2: data bytes */
      /* N-1: chksum 2's complement of bin sum of len, addr & data bytes */

/* end of Local Constants */


/*===========================================================================
 *  FlashPgmTmrExpHdlrPg --
 *     Args:     none
 *    Returns:   Nothing
 *    Bugs:
 * --------------------------------------------------------------------------*/
FlashPgmTmrExpHdlrPg_:
!     FLSPY(0x1a02);   ar = dm(upTimeLow_); FLSPY(ar);
!     ar = dm(upTimeHigh_);FLSPY(ar);
      ar = dm (timerExpIdentity);

!     FLSPY(ar);
      ay1 = ONE_SEC_TIMEOUT;
      af = ar xor ay1;
      if eq jump OneSecTimeOutL;

      ay1 =TWO_SEC_TIMEOUT;
      af = ar xor ay1;
      if eq jump TwoSecTimeOutL;

      ay1 =FOUR_SEC_TIMEOUT;
      af = ar xor ay1;
      if eq jump FourSecTimeOutL;

      ay1 =TEN_SEC_TIMEOUT;
      af = ar xor ay1;
      if eq jump TenSecTimeOutL;

      ar = 0xfff4; jump Freeze_;   /* invalid timer expiry */

OneSecTimeOutL:
TwoSecTimeOutL:
FourSecTimeOutL:
TenSecTimeOutL:
      dm (flashPgmStateEvent) = ay1;

      jump FlashSchedularPg_;

/*===========================================================================
 *  InitFlashPgmPg_ -- Initialises Maintenance/Debug Module
 *     Args:     none
 *    Returns:   Nothing
 *    Bugs:
 * --------------------------------------------------------------------------*/
InitFlashPgmPg_:
      ar = BUF_NULL;
      dm(flashRcvBufIdx)  = ar;
      dm(flashTxBufIdx)   = ar;

      ar = ^flashQ_;
      call InitQ_;
      ar = 0xFFFF;
      dm(flashPgmStateEvent) = ar;
      dm(flashErrReason) = ar;
      ar = FLASH_IDLE; dm(flashPgmState) = ar;
      dm (abortFromSrc) = ar;

      dm (timerExpIdentity) = l4;

      dm (flashPgmAddrLo) = l4;

      dm (flashPgmAddrHi) = l4;
      dm (flashCheckSum_) = l4;
      dm (wrHiAddr)=l4;
!     dm (maxLoopCntHi)= l4;
!     dm (maxLoopCntLo)= l4;
!     dm (maxBitCntHi)= l4;
!     dm (maxBitCntLo)= l4;
      ar = 0x0004;
      dm (rdHiAddr)=ar;
      ar = ID_UNKNOWN;
      dm (flashPgmSrcId) = ar;
      ar = FLASH_IDLE; dm (flashStatus_) = ar;

InitCheckSumLpL:
      /* before chaging this state set flashCheckSum = flashPgmAddr = 0 */
      /* read from lower flash and compute cksum */
      si  = ^flashRecData;         my1 = %flashRecData;
      mr1 = dm(flashPgmAddrHi);    mr0 = dm (flashPgmAddrLo);
      call ReadBytesFromFlash;

      /* compute checksum of this record */
      i1   = ^flashRecData;
      ar   = dm (flashCheckSum_);
      ay1  = dm (i1,m1);
      cntr = %flashRecData;
      do ComputeCksumL until ce;
ComputeCksumL:   ar = ar + ay1, ay1 = dm (i1,m1);
      dm (flashCheckSum_) = ar;

      call WatchDogTgr_;           /* watchdog control */

      ar = dm (flashPgmAddrLo);    /* inc Pgm Addr by 64 bytes */
      ay1 = %flashRecData;         /* chk if pgming is over */
      ar = ar + ay1;
      dm (flashPgmAddrLo) = ar;
      if NOT ac jump InitCheckSumLpL;
      ar=dm(flashPgmAddrHi);ar=ar+1;dm(flashPgmAddrHi)=ar;
      ay1=0x0004;
      ar=ar xor ay1;
      if ne jump InitCheckSumLpL;

      rts;

/*=========================================================================
 *  FlashSchedularPg_ --
 *     Args:     none
 *    Returns:   Nothing
 *    Bugs:
 * ---------------------------------------------------------------------------*/
FlashSchedularPg_:
```

```
!     FLSPY(0x1a04);ar = dm(upTimeLow_); FLSPY(ar);
!     ar = dm(upTimeHigh_);FLSPY(ar);

      ar = dm (flashPgmState);

      ay1 = MAX_FLASH_STATES;           /* chk if state exceeds max */
      af  = ar - ay1;
      if lt jump FlashSchedCont1L;
      ar = 0x7071;     jump Freeze_;    /* invlaid flashPgmState */

FlashSchedCont1L:/* sr0 = dm (flashPgmState) */
      i6 = ^FlashJumpTable;
      m5 = ar;
      modify (i6,m5);
      jump (i6);

FlashJumpTable:
      jump FlashIdleL;       /* FLASH_IDLE                 */
      jump FlashEraseWaitHiL;           /* FLASH_ERASE_WAIT_HI        */
      jump FlashWaitBnmLineL;           /* FLASH_WAIT_FOR_BNM_LINE    */
      jump FlashWaitByteWriteL;   /* FLASH_WAIT_FOR_BYTE_WRITE  */


/* ---------------------------------------------------------------------------*/
/*          FlashPgmState Machine starts here             */
/* ---------------------------------------------------------------------------*/
FlashIdleL:                 /* FLASH_IDLE                 */
      ar = dm(flashPgmStateEvent);
      ay1 = ABORT_FLASH_PGM;
      af = ar xor ay1;
      if eq jump GoToAbortHdlrL;

      ar = dm(flashPgmStateEvent);
      ay1 = START_FLASH_PGM;
      af = ar xor ay1;
      if ne jump EndOfFlashSchedularL;

      call WriteEraseSeqHi;
      ar = FLASH_PGM_TIMER;
      ay1 = FOUR_SEC_TIMEOUT;dm(timerExpIdentity)= ay1;
      call StartTimer_;

      ar = FLASH_ERASE_WAIT_HI; dm (flashPgmState) = ar;
      jump EndOfFlashSchedularL;


/*-------------------------------------------*/
FlashEraseWaitHiL:     /* FLASH_ERASE_WAIT */
      ar = dm(flashPgmStateEvent);
      ay1 = ABORT_FLASH_PGM;
      af = ar xor ay1;
      if eq jump GoToAbortHdlrL;

      ay1 = ONE_SEC_TIMEOUT;
      af = ar xor ay1;
      if eq jump CheckBlankCheckHiL;

      ay1 = TWO_SEC_TIMEOUT;
      af = ar xor ay1;
      if eq jump CheckBlankCheckHiL;
      ay1 = FOUR_SEC_TIMEOUT;
      af = ar xor ay1;
      if eq jump CheckBlankCheckHiL;

      ay1 = TEN_SEC_TIMEOUT;
      af = ar xor ay1;
      if eq jump CheckBlankCheckHiL;

      jump EndOfFlashSchedularL;

CheckBlankCheckHiL:
        mr1 = 0x0004;           /* starting sector number            */
        call BlankCheck;       /* do Blank check for next 4 sectors */

      ar = pass ar;
      if ne jump StartTimerL;

      call SendDnLdRead;
      FLSPY(0x2a2a);

      ar  = OAF_INITIATED;
      ay1 = dm (flashStatus_);
      af  = ar xor ay1;
      if eq jump Skip10SecTimerL;

      ar = FLASH_PGM_TIMER;
      ay1 = TEN_SEC_TIMEOUT;dm(timerExpIdentity) = ay1;
      call StartTimer_;
!     FLSPY(0x1a01);   ar = dm(upTimeLow_); FLSPY(ar);
!     ar = dm(upTimeHigh_);FLSPY(ar);

Skip10SecTimerL:
      ar = FLASH_WAIT_FOR_BNM_LINE; dm (flashPgmState) = ar;
      jump EndOfFlashSchedularL;

StartTimerL:
      ar = dm(flashPgmStateEvent);
      ay1 = ONE_SEC_TIMEOUT;
      af = ar xor ay1;
      if eq jump Start2SecTimeOutL;

      ay1 = TWO_SEC_TIMEOUT;
      af = ar xor ay1;
      if eq jump Start10SecTimeOutL;

      ay1 = FOUR_SEC_TIMEOUT;
      af = ar xor ay1;
      if eq jump Start1SecTimeOutL;

      ay1 = TEN_SEC_TIMEOUT;
      af = ar xor ay1; ar = 0xFFF0;
      if ne jump Freeze_;
      /* erase timed out */
      ar = FLASH_ERR_ERASE_TIMEOUT;dm(flashErrReason)= ar;
      jump AbortDownLdL;

Start1SecTimeOutL:
      ar = FLASH_PGM_TIMER;
      ay1 = ONE_SEC_TIMEOUT; dm(timerExpIdentity) = ay1;
      call StartTimer_;
      jump EndOfFlashSchedularL;

Start2SecTimeOutL:
      ar = FLASH_PGM_TIMER;
      ay1 = TWO_SEC_TIMEOUT; dm(timerExpIdentity) = ay1;
      call StartTimer_;
      jump EndOfFlashSchedularL;

Start4SecTimeOutL:
      ar = FLASH_PGM_TIMER;
      ay1 = FOUR_SEC_TIMEOUT; dm(timerExpIdentity) = ay1;
      call StartTimer_;
      jump EndOfFlashSchedularL;

Start10SecTimeOutL:
      ar = FLASH_PGM_TIMER;
      ay1 = TEN_SEC_TIMEOUT; dm(timerExpIdentity) = ay1;
      call StartTimer_;
      jump EndOfFlashSchedularL;

                  /*---------------------------------------------*/
FlashWaitBnmLineL:           /* FLASH_WAIT_FOR_BNM_LINE */
      /* transition out of this state when a rec is rcvd */
      /* only a time out is maintained over here */
      /* if a msg is held, process that msg */
!     FLSPY(0x1b1b);

      ar = dm (flashPgmStateEvent);
      ay1= TEN_SEC_TIMEOUT ;
      af = ar xor ay1;
      if eq jump GoTo10SecTimeOutL;

      ay1 = WRITE_FIRST_BYTE_OF_BNM_LINE;
      af = ar xor ay1;
      if eq jump ChgStateWaitByteWriteL;

      ay1 = HI_FLASH_PGM_OVER;
      af = ar xor ay1;
      if eq jump ChgStateToEraseLoL;

      ay1 = ABORT_FLASH_PGM;
      af = ar xor ay1;
      if eq jump GoToAbortHdlrL;
      jump EndOfFlashSchedularL;

GoTo10SecTimeOutL:
         /* timeout in READ_WAIT state */
      ar = FLASH_ERR_READ_WAIT_TIMEOUT;dm(flashErrReason) = ar;
      jump AbortDownLdL;


ChgStateWaitByteWriteL:
      call WriteByteSeq;
!     ar = 1;    call DelayMulOfOneusec;           /* wait 5u sec b4 readback*/
!     ar = IO(bcr);dm(startBitCnt) = ar;

      call SendDnLdRead;

      ar  = OAF_INITIATED;   /* no timer if in the OAF flashing */
      ay1 = dm (flashStatus_);
      af  = ar xor ay1;
      if eq jump Skip10SecTimerAgainL;

      ar = FLASH_PGM_TIMER;
      ay1 = TEN_SEC_TIMEOUT;dm(timerExpIdentity)= ay1;
      call StartTimer_;

Skip10SecTimerAgainL:
      ar = 0; dm(flashPgmWaitDelayCnt)= ar;

      ar = FLASH_WAIT_FOR_BYTE_WRITE; dm(flashPgmState) = ar;
      jump EndOfFlashSchedularL;


/*--------------------------------------------*/
FlashWaitByteWriteL:         /* FLASH_WAIT_FOR_BYTE_WRITE */
      /* when over, if more bytes to be pgmed, beg pgming of next byte */
      /* when pgmming of curr rec is done, send read msg */
      /* and goto read wait */
      ar  = dm(flashPgmStateEvent);
      ay1 = ABORT_FLASH_PGM;
      af  = ar xor ay1;
      if eq jump GoToAbortHdlrL;

      ar = dm(flashPgmWaitDelayCnt);
      ar = ar+1;dm(flashPgmWaitDelayCnt) = ar;

      si  = ^flashPgmCmd;    my1 = 1;    /* src ptr, cnt */
      mr1 = dm (flashPgmAddrHi);         /* hi & lo addr for data */
      mr0 = dm (flashPgmAddrLo);
      call ReadBytesFromFlash;           /* alters ar, af, ay1, sr0,1 */

      i1 = dm (flashRecPtr);             /* src ptr */
      ay1 = dm (i1,m1);            /* get data byte written */
      ar = dm (flashPgmCmd);
      af = ar xor ay1;             /* cmp the two bytes */
      if eq jump BytePgmOverL;           /* if b7 same, pgm is over */

      ar = dm(flashPgmWaitDelayCnt);
      ay1 = 20;                    /* abort after 15u timeout */
      af = ar xor ay1;
      if eq jump AbortingL;

!     ar = 1;    call DelayMulOfOneusec;           /* 5usec delay b4 next check*/
      jump EndOfFlashSchedularL;
AbortingL:
            /* Flash Byte Pgm Time out */
      ar = FLASH_ERR_BYTE_PGM_TIMEOUT;dm(flashErrReason) = ar;
      jump AbortDownLdL;                 /* Flash Byte Pgm Time out */

BytePgmOverL:
#if 0 /* for timing calc of one byte write */

      ay1 = dm(startBitCnt);
      ar = IO(bcr);
      ar = ar - ay1;
      if le jump SkipBitCntHiL;

      ay1 = dm(maxBitCntHi);
      af = ar - ay1;
      if gt jump StoreBitCntHiL;
      jump SkipBitCntHiL;
StoreBitCntHiL:
      dm (maxBitCntHi) = ar;
SkipBitCntHiL:
      ar = dm(flashPgmWaitDelayCnt);
      ay1 = dm(maxLoopCntHi);
      af = ar - ay1;

      if gt jump StoreMaxLoopCntHiL;
      jump SkipMaxLoopCntHiL;

StoreMaxLoopCntHiL:
      dm(maxLoopCntHi) = ar;
SkipMaxLoopCntHiL:
#endif
!     FLSPY(0x1d1d);
      ar = dm (flashRecLen);             /* check if pgming the */
      ar = ar - 1;                       /* record is over */
      dm (flashRecLen) = ar;
      if eq jump RecPgmOverL;

      ar = dm (flashPgmAddrLo);          /* inc pgmAddr */
      ar = ar + 1;
      dm (flashPgmAddrLo) = ar;

      ar = dm (flashRecPtr);             /* inc flashRecPtr */
      ar = ar + 1;
      dm (flashRecPtr) = ar;
      call WriteByteSeq;
!     ar = IO(bcr); dm(startBitCnt) = ar;
!     ar = 1;    call DelayMulOfOneusec;            /* wait 5u sec b4 readback*/
      ar = 0; dm(flashPgmWaitDelayCnt)= ar;

      jump EndOfFlashSchedularL;

RecPgmOverL:
      FLSPY(0x1c1c);
      ar = FLASH_WAIT_FOR_BNM_LINE;           dm (flashPgmState) = ar;

      jump EndOfFlashSchedularL;

      /* ----------------------------------------------------------- */

ChgStateToEraseLoL:
      /* BNM file down loaded into High sectors of flash. Transfer  */
      /* from Hi to Lo Sectors needs to be started. Hence all other */
      /* Foreground tasks are stopped */
      imask = 0x000;

      ar = dm(Sys_Ctrl_Reg);
      ar = clrbit 12 of ar;              /* disable SPORT0 */
      ar = clrbit 11 of ar;              /* disable SPORT1 */
      dm (Sys_Ctrl_Reg) = ar;
      dis timer;            /* disable timer */
      ar = FLASH_PGM_TIMER; call StopTimer_;

      dm (flashPgmAddrLo) = l1;    /* reset the addr for xfer operation*/
      call WriteEraseSeqLo;
      FLSPY(0x1f1f);
      dm (retryCntInFlashXfer) = l4;
CheckBlankCheckLoL:

      /* Wait for 1 sec. On expiry, do a blank check */
      ar = 1; call DelayMulOfOnesec;

      ar  = dm (retryCntInFlashXfer);
      ar  = ar + 1;
      ay1 = 16;
      af  = ar - ay1;
      if gt jump FlashEraseFailedInXferL;
      dm (retryCntInFlashXfer) = ar;

      mr1 = 0x0000;           /* starting sector number            */
        call BlankCheck;       /* do Blank check for next 4 sectors */
      ar = pass ar;
      if eq jump StartHiToLoXferL;
      jump CheckBlankCheckLoL;

/*--------------------------------------------*/
      /* Start the HI to LO Sector transfer after erase */
StartHiToLoXferL:
      call WatchDogTgr_;           /* watchdog control */

      /* stop msg rcvd, start transfer of data from upper flash to lower */
      /* read 64 bytes of data into flashRecData */
      si  = ^flashRecData;   my1 = %flashRecData;
      mr1 = dm(rdHiAddr);    mr0 = dm (flashPgmAddrLo);
      call ReadBytesFromFlash;

      i1 = ^flashRecData;    /* i1 => ptr to data to be pgmed */
      cntr = %flashRecData;
      do FlashBytePgmLpL until ce;
              call WatchDogTgr_;         /* watchdog control */
            call WriteByteToFlashDuringTransfer;
!           ar = IO(bcr); dm(startBitCnt)= ar;
            dm (retryCntInFlashXfer) = l4;
WaitForByteWriteL:
!           ar = 1; call DelayMulOfOneusec;


            ar  = dm (retryCntInFlashXfer);

            ar  = ar + 1;
            ay1 = 40;
            af  = ar - ay1;
            if gt jump FlashWriteFailedInXferL;
            dm (retryCntInFlashXfer) = ar;

            si  = ^flashPgmCmd; my1 = 1; /* src ptr, cnt */
            mr1 = dm(wrHiAddr);mr0 = dm (flashPgmAddrLo);
            call ReadBytesFromFlash;     /* alters ar, af, ay1, sr0,1 */

            ay1 = dm (i1,m2);      /* get data byte written */
            ar = dm (flashPgmCmd);
            af = ar xor ay1;       /* cmp the two bytes */
            if eq jump FlashBytePgmOverL;     /* if b7 same, pgm is over */
            jump WaitForByteWriteL;
```

```
FlashBytePgmOverL:
#if 0       /* for timing calculation of byte pgmin */
            ay1 = dm(startBitCnt);
            ar = IO(bcr);
            ar = ar - ay1;
            if le jump SkipStoreBitCntLoL;
            ay1 = dm(maxBitCntLo);
            af = ar - ay1;
            if gt jump StoreBitCntLoL;
            jump SkipStoreBitCntLoL;
StoreBitCntLoL:
            dm(maxBitCntLo)= ar;
SkipStoreBitCntLoL:
            ar  = dm (retryCntInFlashXfer);
            ay1 = dm(maxLoopCntLo);
            af = ar - ay1;
            if gt jump StoreMaxLoopCntL;
            jump SkipMaxLoopCntL;
StoreMaxLoopCntL:
            dm(maxLoopCntLo) = ar;
SkipMaxLoopCntL:
#endif
             ar = dm (flashPgmAddrLo);   /* inc pgmAddrLo */
             ar = ar + 1;
             dm (flashPgmAddrLo) = ar;
FlashBytePgmLpL: modify (i1,m1);        /* inc data ptr */

      call WatchDogTgr_;           /* watchdog control */

      ar = dm (flashPgmAddrLo);    /* inc Pgm Addr by 64 bytes */
      ar = pass ar;                /* chk if pgming is over */
      if ne jump StartHiToLoXferL ;

      ar=dm(wrHiAddr);ar=ar+1;dm(wrHiAddr)=ar;
      ar=dm(rdHiAddr);ar=ar+1;dm(rdHiAddr)=ar;

      ay1=0x0008;
      ar=ar xor ay1;
      if ne jump StartHiToLoXferL;
      call WatchDogTgr_;

      /* Send Stop message on completion of Transfer only for RS232 */
      ar  = dm (flashStatus_);
      ay1 = OAF_CMPLTD;
      ar  = ar xor ay1;
      if eq jump SkipSendDnLdStopL;

      call SendDnLdStop;
      FLSPY(0xfa00);

WaitTillQIsEmptyL:
      call WatchDogTgr_;
      call SlaveSchedular_;
      ar = ^slaveQ_;
      call IsQEmpty_;
      ar = pass ar;
      if eq jump WaitTillQIsEmptyL;
!     FLSPY(0xfa01);
```

```
WaitTillSlvTxL:
      call WatchDogTgr_;
      call SlaveSchedular_;
      ar = dm (phySlaveTxFlag);
      ar = pass ar;
      if ne jump WaitTillSlvTxL;
      FLSPY(0xfa02);
SkipSendDnLdStopL:

      idle;                  /* wait for watchDogReset */

      /* ---------------------------------------------*/
GoToAbortHdlrL:
AbortDownLdL:                      /* --------------------- */
      FLSPY(0x2b2b);
      ar =  dm (flashErrReason);
      FLSPY(ar);

      call SendDnLdErr;      /* if valid old src id, send err msg */

      ar  = dm (flashErrReason);
      ay1 = FLASH_ALREADY_IN_PROGRESS;
      af = ar xor ay1;
      if eq jump EndOfFlashSchedularL;

      ar = FLASH_IDLE; dm (flashPgmState) = ar;

      dm (timerExpIdentity) = l4;
      dm (flashPgmAddrLo) = l4;

      dm (flashPgmAddrHi) = l4;
      dm (wrHiAddr)=l4;
!     dm (maxLoopCntHi)= l4;
!     dm (maxLoopCntLo)= l4;
!     dm (maxBitCntHi) = l4;
!     dm (maxBitCntLo) = l4;

      ar = 0x0004;
      dm (rdHiAddr)=ar;
      ar = FLASH_PGM_TIMER; call StopTimer_;
      ar = BUF_NULL;
      dm(flashRcvBufIdx)  = ar;
      dm(flashTxBufIdx)   = ar;

      ar = ^flashQ_;
      call InitQ_;

      ar = FLASH_IDLE; dm (flashStatus_) = ar;
      dm (abortFromSrc) = ar;

      ar = ID_UNKNOWN;
      dm (flashPgmSrcId) = ar;
      /*reset the flash to read byte array mode*/
      ar = 0xf0; dm (flashPgmCmd)= ar;
      call WriteBytesToFlash;           /* alters ar, af, ay1, sr0,1 */
      ar = 1;call DelayMulOfOneusec;

EndOfFlashSchedularL:
      ar = 0xffff;dm(flashPgmStateEvent)= ar;
      dm(flashErrReason) = ar;
      rts;

FlashWriteFailedInXferL:
      ar = 0x5432;     jump Freeze_;
FlashEraseFailedInXferL:
      ar = 0x5433;     jump Freeze_;


/***************************************************************************
 *  Function:    WriteEraseSeqLo:
 *  args:  none
 *  returns:     none
 *  comments:    Gives a sequence of commands to erase the 0,1,2,3 sectors of
            the flash.
 *  Assumptions: None
 *
 ***************************************************************************/
WriteEraseSeqLo:
/* begin erase of flash pages 0, 1, 2 & 3 */
      /* goto FLASH_ERASE_WAIT for polling the completion */
      /* Sector Erase Seq:   0xAA @ 0x5555    */
      /*                 0x55 @ 0x2AAA    */
      /*                 0x80 @ 0x5555    */
      /*                 0xAA @ 0x5555    */
      /*                 0x55 @ 0x2AAA    */
      /*                 0x30 @   SA      (SA = any addr in sec) */
      /* total time for erase seq down load = app 10 usec */
      si  = ^flashPgmCmd;    my1 = 1;    /* src ptr, cnt */
      mr1 = 0x0000;    mr0 = 0x5555;          /* hi & lo addr */

/*reset the flash to read byte array mode*/
      ar = 0xf0; dm (flashPgmCmd)= ar;
      call WriteBytesToFlash;           /* alters ar, af, ay1, sr0,1 */
      ar = 1;call DelayMulOfOneusec;

      ar  = 0xAA; dm (flashPgmCmd) = ar;/* store data to Tx */
      call WriteBytesToFlash;           /* alters ar, af, ay1, sr0,1 */

                 mr0 = 0x2AAA;
      ar  = 0x55; dm (flashPgmCmd) = ar;/* 2nd unlock byte */
      call WriteBytesToFlash;           /* si, my1, mr1 = unaltered */

                 mr0 = 0x5555;
      ar  = 0x80; dm (flashPgmCmd) = ar;/* Erase Set-up cmd */
      call WriteBytesToFlash;           /* si, my1, mr1 = unaltered */

                 mr0 = 0x5555;
      ar  = 0xAA; dm (flashPgmCmd) = ar;/* first unlock byte again */
      call WriteBytesToFlash;           /* si, my1, mr1 = unaltered */

                 mr0 = 0x2AAA;
      ar  = 0x55; dm (flashPgmCmd) = ar;/* 2nd unlock byte again */
      call WriteBytesToFlash;           /* si, my1, mr1 = unaltered */

                 mr0 = 0x0000;    /* hi & lo addr of sector 0 */
      ar  = 0x30; dm (flashPgmCmd) = ar;/* write sector erase code */
      call WriteBytesToFlash;           /* si, my1 = unaltered */

/*Bottom boot 4mb flash*/
```

```
                    mr0 = 0x4000;
      call WriteBytesToFlash;           /* si, my1 = unaltered */

                    mr0 = 0x6000;
      call WriteBytesToFlash;           /* si, my1 = unaltered */

                    mr0 = 0x8000;
      call WriteBytesToFlash;           /* si, my1 = unaltered */


      mr1=0x0001; mr0 = 0x0000;   /* hi & lo addr of sector 1 */
      call WriteBytesToFlash;     /* data, si, my1, mr1 unaltered */

      mr1=0x0002; mr0 = 0x0000;   /* hi & lo addr of sector 2 */
      call WriteBytesToFlash;     /* data, si, my1, mr1 unaltered */

      mr1=0x0003; mr0 = 0x0000;   /* hi & lo addr of sector 3 */
      call WriteBytesToFlash;     /* data, si, my1, mr1 unaltered */


      rts;
/***************************************************************************
 *  Function:    WriteEraseSeqHi:
 *  args:  none
 *  returns:     none
 *  comments:    Gives a sequence of commands to erase the 4,5,6,7 sectors of
          the flash.
 *  Assumptions: None
 *
 ***************************************************************************/
WriteEraseSeqHi:
      /* begin erase of flash pages 4, 5, 6 & 7 */
      /* goto FLASH_ERASE_WAIT for polling the completion */
      /* Sector Erase Seq:   0xAA @ 0x5555    */
      /*                0x55 @ 0x2AAA    */
      /*                0x80 @ 0x5555    */
      /*                0xAA @ 0x5555    */
      /*                0x55 @ 0x2AAA    */
      /*                0x30 @   SA      (SA = any addr in sec) */


                /* if any int delays successive write by more */
                /* than 80 usec, the erase seq gets aborted */
      push sts;
      imask = 0x000;
      FLSPY(0x1a1a);
      /* total time for erase seq down load = app 10 usec */

      si  = ^flashPgmCmd;    my1 = 1;   /* src ptr, cnt */
      mr1 = 0x0000;    mr0 = 0x5555;          /* hi & lo addr */

/*reset the flash to read byte array mode*/
      ar = 0xf0; dm (flashPgmCmd)= ar;
      call WriteBytesToFlash;           /* alters ar, af, ay1, sr0,1 */
      ar = 1;call DelayMulOfOneusec;

      ar  = 0xAA; dm (flashPgmCmd) = ar;/* store data to Tx */
      call WriteBytesToFlash;                 /* alters ar, af, ay1, sr0,1 */

                    mr0 = 0x2AAA;
      ar  = 0x55; dm (flashPgmCmd) = ar;/* 2nd unlock byte */
      call WriteBytesToFlash;                 /* si, my1, mr1 = unaltered */

                       mr0 = 0x5555;
      ar  = 0x80; dm (flashPgmCmd) = ar;/* Erase Set-up cmd */
      call WriteBytesToFlash;                 /* si, my1, mr1 = unaltered */

                       mr0 = 0x5555;
      ar  = 0xAA; dm (flashPgmCmd) = ar;/* first unlock byte again */
      call WriteBytesToFlash;                 /* si, my1, mr1 = unaltered */

                       mr0 = 0x2AAA;
      ar  = 0x55; dm (flashPgmCmd) = ar;/* 2nd unlock byte again */
      call WriteBytesToFlash;                 /* si, my1, mr1 = unaltered */

      ar  = 0x30; dm (flashPgmCmd) = ar;/* write sector erase code */
      mr1 = 0x0004;    mr0 = 0x0000;          /* hi & lo addr of sector 4 */
      call WriteBytesToFlash;                 /* si, my1 = unaltered */

      mr1 = 0x0005;    mr0 = 0x0000;          /* hi & lo addr of sector 5 */
      call WriteBytesToFlash;           /* data, si, my1, mr1 unaltered */

      mr1 = 0x0006;    mr0 = 0x0000;          /* hi & lo addr of sector 6 */
      call WriteBytesToFlash;           /* data, si, my1, mr1 unaltered */

      mr1 = 0x0007;    mr0 = 0x0000;    /* hi & lo addr of sector 7 */
      call WriteBytesToFlash;           /* data, si, my1, mr1 unaltered */

                 mr0 = 0x8000;    /* hi & lo addr of sector 7 */
      call WriteBytesToFlash;           /* data, si, my1, mr1 unaltered */

                 mr0 = 0xa000;    /* hi & lo addr of sector 7 */
      call WriteBytesToFlash;           /* data, si, my1, mr1 unaltered */

                 mr0 = 0xc000;    /* hi & lo addr of sector 7 */
      call WriteBytesToFlash;           /* data, si, my1, mr1 unaltered */

      pop sts;                    /* restore imask */


      rts;

/***************************************************************************
 *  Function:    WriteByteToFlashDuringTransfer:
 *  args:  none
 *  returns:     none
 *  comments:    Gives a sequence of commands for writing a byte into flash.
 *  Assumptions:Ptrs point to the byte to be written.
 *
 ***************************************************************************/
WriteByteToFlashDuringTransfer:
      /* total time for pgm seq down load = app 4 usec */
            si  = ^flashPgmCmd;    my1 = 1;/* src ptr, cnt */
            mr1 = dm(wrHiAddr);    mr0 = 0x5555;    /* hi & lo addr */
            ar  = 0xAA; dm (flashPgmCmd) = ar;/* store data to Tx */
            call WriteBytesToFlash;     /* alters ar, af, ay1, sr0,1 */

                             mr0 = 0x2AAA;
            ar  = 0x55; dm (flashPgmCmd) = ar;/* 2nd unlock byte */
            call WriteBytesToFlash;     /* si, my1, mr1 = unaltered */
```

```
                          mr0 = 0x5555;
          ar  = 0xA0; dm (flashPgmCmd) = ar;/* Pgm Set-up cmd */
          call WriteBytesToFlash;      /* si, my1, mr1 = unaltered */

          si  = i1;  mr0 = dm (flashPgmAddrLo);
          call WriteBytesToFlash;      /* write actual byte */
          rts;

/***************************************************************************
 *  Function:    WriteByteSeq
 *  args:  none
 *  returns:     none
 *  comments:    Gives a sequence of commands for writing a byte into flash.
 *  Assumptions:Ptrs point to the byte to be written.
 *
 ***************************************************************************/
WriteByteSeq:
      /* start the next rec programming, goto FLASH_PGM_WAIT for polling */
      /* assumption is that flashRecLen is at least 1 */
      /* Byte Program Seq:   0xAA @ 0x5555    */
      /*               0x55 @ 0x2AAA    */
      /*               0xA0 @ 0x5555    */
      /*                PD  @   PA      */
                  /* if any int delays successive write by more */
                  /* than XX usec, the pgm seq may get aborted */
                  /* data sheet does not give any value for XX */
      push sts;
      imask = 0x000;

      /* total time for pgm seq down load = app 4 usec */

      si  = ^flashPgmCmd;    my1 = 1;    /* src ptr, cnt */
      mr1 = 0x0000;    mr0 = 0x5555;          /* hi & lo addr */
      ar  = 0xAA; dm (flashPgmCmd) = ar;/* store data to Tx */
      call WriteBytesToFlash;                 /* alters ar, af, ay1, sr0,1 */

                  mr0 = 0x2AAA;
      ar  = 0x55; dm (flashPgmCmd) = ar;/* 2nd unlock byte */
      call WriteBytesToFlash;                 /* si, my1, mr1 = unaltered */

                  mr0 = 0x5555;
      ar  = 0xA0; dm (flashPgmCmd) = ar;/* Pgm Set-up cmd */
      call WriteBytesToFlash;                 /* si, my1, mr1 = unaltered */

      si  = dm (flashRecPtr);      my1 = 1;    /* src ptr, cnt */
      mr1 = dm (flashPgmAddrHi);         /* hi & lo addr for data */
      mr0 = dm (flashPgmAddrLo);
      call WriteBytesToFlash;                 /* write actual byte */

      pop sts;                    /* restore imask */
      rts;

/***************************************************************************
 *  Function:    BlankCheck
 *  args:  mr1 = sector from which blank check commences (0,4)
 *  returns:     ar = 0 if success ff if failure
 *
 ***************************************************************************/
BlankCheck:
        dm(blChStSector) = mr1;                  /* hi addr - sector no     */
        si  = ^flashPgmCmd;     my1 = 1;         /* src ptr, cnt            */
        mr0 = 0x0000;                            /* lo addr(start from max.)*/

Read1ByteL:
        call ReadBytesFromFlash;                 /* alters ar, af, ay1, sr0 */
        ay1 = dm(flashPgmCmd);
        ar = 0xff;
        af = ar xor ay1;
        if ne rts;                              /* blank check failed      */
        ar = mr0 + 1;
        mr0 = ar;
        ar = pass ar;
        if ne jump Read1ByteL;                   /* one sector not complete */
!        call WatchDogTgr_;
        ar = mr1 + 1;
        mr1 = ar;
        ay1 = dm(blChStSector);
        ar = ar - ay1;
        ar = ar - 4;
        if lt jump Read1ByteL;                   /* 4 sectors not complete  */
ExitBlankCheckL:
        rts;
/*------------------------------------------------*/

/*======================================================================
 * DelayMulOfOnesec:
 *    Args: ar = no of 1 sec
 *    Ret:  Nothing
 *    Bugs:
 *----------------------------------------------------------------------*/
DelayMulOfOnesec:
      cntr = ar;
      do NoOfOneSecDelayL until ce;
      call WatchDogTgr_;
      ax0 = 1000;      { 1000 ms delay }

OuterLoopL:
      ar = 26000;      { 52 mips CPU }

InnerLoopL:            { delay of 1 ms }
      ar = ar - 1;
      if ne jump InnerLoopL;

      ar = ax0;
      ar = ar - 1;
      ax0 = ar;
      if ne jump OuterLoopL;
NoOfOneSecDelayL: nop;

      rts;
/*======================================================================
 * DelayMulOfOneusec:
 *    Args: ar = no of 1 microsec
 *    Ret:  Nothing
 *    Bugs:
 *----------------------------------------------------------------------*/
DelayMulOfOneusec:
      af = pass ar;
MulOf1usec:
      ar=26;
OneusL:
      ar = ar - 1;
      if gt jump OneusL;

      af = af -1;
      if gt jump MulOf1usec;
rts;

/*============================================================================
 * WriteBytesToFlash - Write bytes to flash prom thru BDMA
 *    Args: si: data ptr, my1: cnt, mr0: Addr Lo, mr1: Addr Hi.
 *    Ret:  Nothing
 *    Bugs:
 *---------------------------------------------------------------------------*/
WriteBytesToFlash:
      dm (BDMA_Internal_Address) = si; /* store int mem data addr */

      dm (BDMA_External_Address) = mr0; /* only lower 14 bits valid */

      sr  = lshift mr0 by -6 (lo); /* b15..14 as b9..8 of page num */
      ay1 = 0x0300;
      af  = sr0 and ay1;
                             /* mr1 = Ext Addr Hi */
      sr  = lshift mr1 by 10 (lo); /* b15..10 of page num */
      ar  = sr0 or af;       /* combine b15..10 & b9..8 */
      ay1 = 0x0007;                /* to store LSB into boot mem */
      ar  = ar or ay1;
      dm (BDMA_Control) = ar;

      dm (BDMA_Word_Count) = my1;  /* write xfer cnt and init BDMA xfer */

WriteBytesWaitL:
!     idle;                  /* no forground activity */
      ar = dm (BDMA_Word_Count);
      ar = pass ar;
      if ne jump WriteBytesWaitL;
      rts;                   /* byte xfer over */

/*============================================================================
 * ReadBytesFromFlash - Read bytes from flash prom thru BDMA
 *    Args: si: data ptr, my1: cnt, mr0: Addr Lo, mr1: Addr Hi.
 *    Ret:  Nothing
 *    Bugs:
 *---------------------------------------------------------------------------*/
ReadBytesFromFlash:
      dm (BDMA_Internal_Address) = si; /* store int mem data addr */

      dm (BDMA_External_Address) = mr0; /* only lower 14 bits valid */

      sr  = lshift mr0 by -6 (lo); /* b15..14 as b9..8 of page num */
      ay1 = 0x0300;
      af  = sr0 and ay1;
                             /* mr1 = Ext Addr Hi */
      sr  = lshift mr1 by 10 (lo); /* b15..10 of page num */
      ar  = sr0 or af;       /* combine b15..10 & b9..8 */
      ay1 = 0x0003;                /* to read LSB from boot mem */
      ar  = ar or ay1;
      dm (BDMA_Control) = ar;
      dm (BDMA_Word_Count) = my1;  /* write xfer cnt and init BDMA xfer */

ReadBytesWaitL:
!     idle;                  /* no forground activity */
      ar = dm (BDMA_Word_Count);
      ar = pass ar;
      if ne jump ReadBytesWaitL;
      rts;                   /* byte xfer over */


/*============================================================================
 *  FlashMsgProcPg_ -- Process BNM Download related msgs.
 *     Args:     none
 *    Returns:   Nothing
 *    Bugs:
 * ---------------------------------------------------------------------------*/
FlashMsgProcPg_:
      ar = ^flashQ_;   /* poll the flashQ to receive msg for flash module */
      call IsQEmpty_;
      af = pass ar;
      if ne rts;

!     FLSPY(0xFF55);

      ar = ^flashQ_;
      call RdFromQ_;
      dm(flashRcvBufIdx) = ar;
      call SetBufAccess_;          /* ar has buf idx */
      i6  = ar;
      m5  = SOURCE_ID_OFFSET;
      modify(i6,m5);
      m5  = 1;
      af = pass 0xFF;
      ar  = dm(i6,m5);       /* get source Id */
      ar  = ar and af;       /* mask off the higher order bits */
      dm (flashPgmSrcId) = ar;     /* copy srcId for later use */

      ar  = dm (flashStatus_);     /* allow first message frm src*/
      ay1 = FLASH_IDLE;
      af  = ar xor ay1;
      if eq jump AllowFlashL;

      /* Simultaneous flashing check */
      ar  = dm (flashPgmSrcId);
      ay1 = OMC;
      af  = ar xor ay1;
      if ne jump ChkRS232InitiatedL;

      ar  = dm (flashStatus_);
      ay1 = RS232_INITIATED;
      af  = ar xor ay1;
      if eq jump FlashInProgressL;
      jump AllowFlashL;


ChkRS232InitiatedL:
      ar  = dm (flashPgmSrcId);
      ay1 = ID_DWS_SLAVE;
      af  = ar xor ay1;
      if ne jump FreeFlashMsgL;    /* invalid src for flashing freemsg*/
      ar  = dm (flashStatus_);
      ay1 = OAF_INITIATED;
      af  = ar xor ay1;
      if eq jump FlashInProgressL;

AllowFlashL:

      FLSPY(0xFF66);
      ay1 = 0x00FF;
      ar  = dm(i6,m5);       /* get module Id */
      ar  = ar and ay1;      /* mask off the higher order bits */
      ay1 = UPDT_MODULE_ID;        /* chk if correct module Id */
      ar  = ar xor ay1;
      if ne jump FreeFlashMsgL;    /* free all non-down_load msgs */

      ay1 = 0x00FF;
      ar  = dm(i6,m5);       /* get the msg type   */
      ar  = ar and ay1;      /* mask off the higher order bits */
      ay1 = DOWN_LOAD_BNM;         /* chk if down load msg only */
      ar  = ar xor ay1;
      if ne jump FreeFlashMsgL;    /* free all non-down_load msgs */


      ar  = dm(i6,m5);       /* get the sub cmd */
      ay1 = 0x00FF;
      ar  = ar and ay1;      /* mask off the higher order bits */


      ay1 = BNM_DN_LD_START;
      af  = ar xor ay1;
      if eq jump DownLdStartL;     /* BNM_DN_LD_START */

      ay1 = BNM_DN_LD_WRITE;
      af = ar xor ay1;
      if eq jump DownLdWriteL;     /* BNM_DN_LD_WRITE */

      ay1 = BNM_DN_LD_STOP;
      af = ar xor ay1;
      if eq jump DownLdStopL;            /* BNM_DN_LD_STOP  */

      ay1 = BNM_DN_LD_ERROR;
      af = ar xor ay1;
      if eq jump DownLdErrorL;     /* BNM_DN_LD_ERROR */

FreeFlashMsgL:
      ar = dm(flashRcvBufIdx);
      call FreeMsgBuf_;

      rts;

DownLdStartL:                      /* ---------------------- */

      /* start cmd rcvd in idle state, start the bnm downloading seq */

      FLSPY(0x0011);

      ar  = dm (flashPgmSrcId);
      ay1 = OMC;
      af  = ar xor ay1;
      if eq jump SetOmcInitL;      /* set flashStatus according to the source */

      ar  = RS232_INITIATED;
      dm (flashStatus_) = ar;
      jump SkipVersionChkL;

SetOmcInitL:
      ay1 = OAF_INITIATED;
      dm (flashStatus_) = ay1;

      /* check for force update flag */
      ar = dm(flashRcvBufIdx);
      call SetBufAccess_;          /* ar has buf idx */
      m5  = ar;              /* ar has buf ptr */
      i6  = FORCE_CHK_OFFSET;
      modify (i6,m5);

      ar  = dm (i6,m5);
      ar  = pass ar;
      if ne jump SkipVersionChkL;

      i6  = FLASH_VER_OFFSET;      /* points to the first digit of version */
      modify (i6,m5);
      m5 = 2;

      i1  = ^versionNo_;
      m3  = 1;
      ar = 0xff;
      af = pass ar;
      FLSPY(0x0AF9);
      cntr = 4;  /* to check version type 'ipcp','ipep','wsep' */
      do ContinueVersionChkL until ce; /* version type check */
        ar  = dm (i6,m5);
        FLSPY(ar);
        ar = ar and af;
        ay1 = dm (i1,m3);
        FLSPY(ay1);
        ar  = ar xor ay1;
ContinueVersionChkL: if ne jump InvalidBnmTypeL;

SkipVersionChkL:
      ar = START_FLASH_PGM; dm (flashPgmStateEvent) = ar;
      jump FreeFlashMsgL;

DownLdStopL:                       /* ---------------------- */

      FLSPY(0x2233);

      ar = dm (flashStatus_);
      ay1 = OAF_INITIATED;
      ar = ar xor ay1;
      if ne jump ChkRS232L;
      ar = OAF_CMPLTD;
      dm (flashStatus_) = ar;
      jump EndDownLdStopL;
ChkRS232L:
      ar = dm (flashStatus_);
      ay1 = RS232_INITIATED;
      ar = ar xor ay1;
      if ne rts;
      ar = RS232_CMPLTD;
      dm (flashStatus_) = ar;
EndDownLdStopL:
      ar = HI_FLASH_PGM_OVER;
      dm(flashPgmStateEvent)= ar;
      jump FreeFlashMsgL;

DownLdErrorL:                     /* --------------------- */
      FLSPY(0x3344);

      ar = dm (i6,m5); /* store the abort reason */
      dm (flashErrReason) = ar;

      dm (abortFromSrc) = m1;

      ar = ABORT_FLASH_PGM; dm (flashPgmStateEvent) = ar;
      jump FreeFlashMsgL;

DownLdWriteL:                     /* --------------------- */
ProcessWriteMsgL:/* i6 pointing to bnm info, m5 = 1 */

      FLSPY(0x1122);

!     ar = 0x9898;     jump Freeze_;

      ay1 = dm (i6,m5);
      ar  = 0x0053;               /* 0x53 = 'S' */
      ar  = ar xor ay1;
      if ne jump NonMotorolaFomatL;   /* if not motorola S, abort */

      sr0 = dm (i6,m5);/* save record type */
      sr1 = dm (i6,m5);/* save record length */

      ar = sr1 - 1;          /* compute chksum excluding chksum byte */
      if le jump RecLenNOKL;  /* error in record length received */
      ar = sr1 - 1;
      cntr = ar;
      i1 = i6;         /* i6: ptr to addr, data, chksum */
      ar = i1;
!     FLSPY(0x2c2c);FLSPY(ar);
      ar = sr1;
      ay1 = dm (i1,m1);
      do CalcSChkSumL until ce;
CalcSChkSumL:    ar = ar + ay1, ay1 = dm (i1,m1);
      ar = NOT ar;           /* 1's complement */
      ar = ar xor ay1; /* cmp rcvd chksum */
      ay1 = 0xFF;      /* mask out MSByte of comparison */
      ar = ar and ay1;

      if ne jump CKSumNOKL;   /* if chksum not OK, abort */

      ay1 = 0x0032;               /* chk for rec type '2' */
      ar = sr0 xor ay1;
      if ne jump InvalidRecTypeL;  /* if rec type != 2 or 9, abort */

ProcessSDataRecL:
      ar = sr1 - 4;               /* compute data bytes length */
      if le jump NoDataInRecL;    /* if no data in rec, abort */
      ar = sr1 - 4;
      ay1 = 64;
      af = ar - ay1;
      if gt jump UnexpectedLenL;   /* if unexpected length, abort */
      ar = sr1 - 4;
      dm (flashRecLen) = ar;       /* len between 1 and 64 */

      ar = ^flashRecData;
      dm (flashRecPtr) = ar;       /* init the ptr to rec */

      ar  = dm (i6,m5);
      ar = ar + 4;                 /* offset to start from sector 4 */
      dm (flashPgmAddrHi) = ar;    /* byte 2 of addr */

      ar = dm (i6,m5);       /* byte 1 of addr */
      sr0 = dm (i6,m5);      /* byte 0 of addr */
      sr = sr or lshift ar by 8 (lo);
      dm (flashPgmAddrLo) = sr0;   /* byte 1,byte0 of addr */

      i1 = ^flashRecData;
      ar = dm (flashRecLen);
      cntr = ar;
      do CopySRecL until ce;
            ay1 = dm (i6,m5);
CopySRecL:  dm (i1,m1) = ay1;

      /* change state of flashPgm here */
      ar = WRITE_FIRST_BYTE_OF_BNM_LINE; dm (flashPgmStateEvent) = ar;

      jump FreeFlashMsgL;

NonMotorolaFomatL:
      ar  = FLASH_ERR_NON_MOTOROLA; dm(flashErrReason) = ar;
      jump CommonAbortL;
RecLenNOKL:
      ar = FLASH_ERR_REC_LEN_NOK; dm(flashErrReason) = ar;
      jump CommonAbortL;
CKSumNOKL:
      ar = FLASH_ERR_CKSUM; dm(flashErrReason)= ar;
      jump CommonAbortL;
InvalidRecTypeL:
      ar = FLASH_ERR_INVALID_REC_TYPE;dm(flashErrReason)= ar;
      jump CommonAbortL;
NoDataInRecL:
      ar = FLASH_ERR_NO_DATA_IN_REC; dm(flashErrReason) = ar;
      jump CommonAbortL;
UnexpectedLenL:
      ar = FLASH_ERR_UNEXPECTED_LEN; dm(flashErrReason) = ar;
      jump CommonAbortL ;
InvalidBnmTypeL:
      pop pc; pop loop; pop cntr;
      ar = WSIP_INVALID_BNM_TYPE; dm(flashErrReason) = ar;
      jump CommonAbortL ;
FlashInProgressL:
      ar  = FLASH_ALREADY_IN_PROGRESS; dm(flashErrReason)= ar;
      jump CommonAbortL ;
CommonAbortL:
      ar = ABORT_FLASH_PGM; dm(flashPgmStateEvent)= ar;
      jump FreeFlashMsgL;


/*============================================================================
 * SendDnLdErr - Send the BNM_DN_LD_ERROR msg to the pgm src
 * SendDnLdRead - Send the BNM_DN_LD_READ msg to the pgm src
 * SendMsgToPgmSrc - Send msg (flashSendMsg) to the pgm src
 *    Args: none
 *    Ret:  none
 *    Bugs:
 *-----------------------------------------------------------------------------*/
SendDnLdErr:
      FLSPY(0x2d2d);

!     ar = 0x2d2e;     jump Freeze_;

      ar = dm (abortFromSrc);
      ar = pass ar;
      if ne rts;

      ar = BNM_DN_LD_ERROR;
      dm (flashSendMsg) = ar;
      FLSPY(ar);
      jump SendMsgToPgmSrc;

SendDnLdRead:
      ar = BNM_DN_LD_READ;
      dm (flashSendMsg) = ar;
      jump SendMsgToPgmSrc;

SendDnLdStop:
      ar  = BNM_DN_LD_STOP;
      dm (flashSendMsg) = ar;

      /*   STOP ONLY TO RS232 based flash */
      ar  = RS232_CMPLTD;
      ay1 = dm (flashStatus_);
      af  = ar xor ay1;
      if eq jump MsgToRS232L;
      ar = 0xaa0d; jump Freeze_;   /* stop msg in invalid state */

SendMsgToPgmSrc:

      ar  = dm (flashStatus_);
      ay1 = RS232_INITIATED;
      af  = ar xor ay1;
      if eq jump MsgToRS232L;

      /* flash already in progress by OAF */
      ar  = dm (flashErrReason);
      ay1 = FLASH_ALREADY_IN_PROGRESS;
      af  = ar xor ay1;
      if eq jump SendMsgToRS232L;
      jump SendMsgToIwu;     /* send error msg to second source */

MsgToRS232L:
      /* flash already in progress check */
      ar  = dm (flashErrReason);
      ay1 = FLASH_ALREADY_IN_PROGRESS;
      af  = ar xor ay1;
      if eq jump SendMsgToIwu;

SendMsgToRS232L:
      call AllocMsgBuf_;     /* allocate transmit mesg buffer */
      ay1 = BUF_NULL;
      af = ar xor ay1;
      if eq rts;       /* if buffer not available, exit */

      dm(flashTxBufIdx) = ar;          /* store allocated buffer index */
      call SetBufAccess_;        /* ar had buf idx */
      i1 = ar;       /* ptr to transmit buffer */

      dm (i1,m1) = 0;                    /* control field */
      dm (i1,m1) = MESSAGE_OVERHEAD_LEN + 1;  /* ovrhd bytes */

      ar = ID_DWS_SLAVE;           /* RS232 */
      dm (i1,m1) = ar;       /* destination id */
      ar = dm (selfId_);
      dm (i1,m1) = ar;             /* source Id */
      dm (i1,m1) = UPDT_MODULE_ID;       /* processId */
      dm (i1,m1) = DOWN_LOAD_BNM;        /* message type */
      ar = dm (flashSendMsg);
      dm (i1,m1) = ar;             /* sub cmd */
                                   /* last two bytes checksun */
      ar = dm (flashErrReason);          /* send error message to pc*/
      dm (i1,m1) = ar;
      ar = dm(flashTxBufIdx);        /* Rout the mesg using MsgRout */
      jump MsgRout_;

/*===========================================================================
 *  SendMsgToIwu -- Sends the OAf response back to the IWU layer
 *             flashSendMsg has the msg to be sent
 *    Args: None
 *    Returns:   Nothing
 *    Bugs:
 * ---------------------------------------------------------------------------*/
SendMsgToIwu:
     call AllocMsgBuf_;      /* allocate transmit mesg buffer */
     ay1 = BUF_NULL;
     af = ar xor ay1;
     if eq rts;        /* if buffer not available, exit */

     dm (flashIwuIdx) = ar;
     ay1 = BUF_NULL;
     af = ar xor ay1;
     if eq rts;        /* if buffer not available, exit */

     call SetBufAccess_;
     i1 = ar;
     FLSPY(0xFF69);
     ar = dm (flashSendMsg);
     FLSPY(ar);
     dm (i1,m1) = ar;
     ay1 = BNM_DN_LD_ERROR;
     ar = ar xor ay1;       /* if err msg fill the error reason */
     if ne jump PutMsgInQL;
     FLSPY(0xFF70);
     ar = dm (flashErrReason);
     dm (i1,m1) = ar;
     FLSPY(ar);

PutMsgInQL:
     ar  = ^flashIwuQ_;
     ay1 = dm (flashIwuIdx);
     call WrInQ_;

     rts;

.endmod;
/***************************** flashpgm.dsp ********************************/
```