Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.



THIS SPEC IS OBSOLETE

Spec No: 002-05213

Spec Title: AN205213 - FR30 EXAMPLE INTERFACE TO EXTERNAL FLASH MEMORY

Replaced By: NONE

AN205213

FR30 Example Interface to External Flash Memory

This application note shows how to interface from an MB91101 (FR30 device) to an external flash memory. As target hardware, the FR30 evaluation board is used in this example. On this board, the MB91101 is connected to 29F400 flash memory.

Contents

1 Background 1
2 Bit Assignment, Even/Odd Address, Byte/Word Accesses 2
3 Example Program.....2
Document History.....15

OBSOLETE

This application note shows how to interface from an MB91101 (FR30 device) to an external flash memory. As target hardware, the FR30 evaluation board is used in this example. On this board, the MB91101 is connected to 29F400 flash memory.

1 Background

FLASH memory can be directly connected to any microcotroller bus in a similar manner to ROM or EPROM devices. Flash memories can also be programmed on dedicated programmers similar to EPROMs, but the real advantage is, that they can be re-programmed in-circuit under the appropriate software control. One could assume that a flash would require some additional and special control signals, but that is not really the case. Actually, compared with a standard EPROM, there are some special signals like /WE, RY/BY, /RESET, and BYTE but their functions are quite trivial:

The /BYTE input is used to configure the device to operate in either 8-bit or 16-bit wide mode. If it is configured as an 8-bit memory, the additional address line “A -1” is needed to address the complete memory area. As mentioned above, for this application note we assume the 16-bit mode, so “A -1” is not used. “RY/BY” is an output status signal which could be used to check if the flash memory is busy (during “embedded algorithm operations”), but since this status information can also be determined by software, it’s not really necessary to connect this signal to a dedicated controller input.

“/WE” is a simple write enable input and is used to enable writing data to the flash memory. But, do not assume that writing to a flash memory location would program this location with the written data. This would be too simple - and much more important, too dangerous. A “corrupted” program could then overwrite and destroy the memory contents if it writes something into the flash memory space. To write to a flash memory location, a special command has to be sent to the flash device internal state machine to put it into the “embedded command” mode. In this mode, you can then execute a write operation or one of the other embedded commands, such as erasing. “/RESET” could be used to get the flash memory out of the “embedded command” mode, but as above, this can be done by software as well, so this input could be simply tied to Vcc. In fact, the /RESET input could be useful in case the flash is “unlocked” and then an MCU-reset will occur.

2 Bit Assignment, Even/Odd Address, Byte/Word Accesses

To understand the software procedures which can activate the “embedded commands”, there are some more hardware related issues to discuss, since they have some impact on the software. The main reason for this is because we have chosen the 16-bit wide example. The typical connection scheme between a Fujitsu microcontroller and a 16-bit wide flash memory is shown in the following figure. Other Architectures might look slightly different.

The address line A1 of the controller is connected to A0 of the flash memory. The controller would need its A0 line only to address single bytes in an external memory, but since we configured our system to be 16-bit wide (“half-word”), the smallest unit the microcontroller can read is a 16-bit word.

3 Example Program

An example Softune-project **FLASDEMO.PRJ** was developed which is able to demonstrate the embedded functions. The project is structured into two source code modules. **FLA.C** provides the “embedded command functions”, **FLASDEMO.C** is a user interface or monitor type program, which calls these functions to display some information about the flash memory (Vendor-, Device-Code), and which allows to do some basic thinks like chip-erase, individual sector erase, program and display memory locations. To invoke the program, load it into the emulator, press reset and go (or download it to the FR30-board and press the user reset button). If a terminal is connected to the external UART- port (right-hand RS232-plug), the flash info will apear and a menu will be displayed.

OBSOLETE

```
/*-----------------------------------------------------------------------------*/
/*                    F U J I T S U                                            */
/*          M i k r o e l e k t r o n i k   G m b H                            */
/*                                                                                    */
/*  Filename:          FLASDEMO.C                                                    */
/*  Function:          Demoprogram for simple ext. FLASH interface                   */
/*                                                                                    */
/*  Series: FR30                                                                  */
/*  Version:           V01.00                                                        */
/*  Design:            E. Bendels                                                    */
/*  Change:            M. Mierse Dec-97                                              */
/*-----------------------------------------------------------------------------*/


#include <string.h>

#include "fla.h"                    /* Flash Functions headers */

#define nSectors 11
#define MaxInpStrLen 10
#define pLED     (*(volatile unsigned short *)(0x0108000L))
#define UARTdata (*(volatile unsigned short *)(0x0100000L))
#define UARTcnt  (*(volatile unsigned short *)(0x0100002L))

#define SRAMstart 0xFFFC0000
#define SRAMwsize 0x100

char xxx = 55;
char yy;

CSTR COMPANY[]  = "(C) Fujitsu Mikroelektronik GmbH, FLASH-Demo";
CSTR    date[]  = __DATE__;
CSTR    time[]  = __TIME__;

CSTR InitMsg[]     = "\n\n** Fujitsu MBM29F400xA FLASH-Demonstration Program **";
CSTR InfoMsg[]          = "\n\n== Current Flash Status ==";
CSTR ManuMsg[]     = "\n Manufacturer Code : ";
CSTR DeviMsg[]     = "\n Device Code       : ";
CSTR SecInfoMsg[]  = "\n Sector Information ";
CSTR CrLfMsg[]     = "\n";
CSTR SecNLMsg[]    = "  Sector";
CSTR BaseAdrMsg[]  = "\n BaseAddress : ";
CSTR NotBeginMsg[] = " (WARNING: No Physical Start Address !)";
CSTR SecToMsg[]    = " - ";
CSTR FujiMsg[]     = " FUJITSU";
CSTR UnknMsg[]     = " Unknown !";
CSTR F400TAMsg[]   = " MBM29F400TA";
CSTR F400BAMsg[]   = " MBM29F400BA";
CSTR ProtectMsg[]  = " (PROTECTED)";
CSTR NoProtectMsg[]= " (NOT Protected)";
CSTR ProtErrMsg[]  = " (ProtectCheckError!)";
CSTR EmptyMsg[]    = " (EMPTY)";
CSTR NotEmptyMsg[] = " (NOT Empty)";
CSTR CommandMsg[]= "\n\n MENUE:     'R':ReadMemoroyWord, 'P':ProgramMemoryWord"
                        "\n 'I':Info   'S':SectorErase,     'E':CompleteErase"
                        "\n COMMAND -> ";
CSTR ReadAdrMsg[]   = "\nRead from Adr : *0x";
CSTR ProgAdrMsg[]   = "\nProgram to Adr: *0x";
CSTR ReadDataMsg[]  = " => 0x";
CSTR WriteDataMsg[]  = " <= 0x";
CSTR BackSpaceSeq[]  = "\10 \10";
CSTR OkMsg[]            = " OK.";
CSTR NotOkMsg[]   = " ERROR";
CSTR LocNotEmptMsg[] = "(Location not Empty, or Odd Address !)";
CSTR SecEraMsg[]     = "\n Sector Erase, Setctor 0..A : ";
CSTR CompEraMsg[]    = "\n Chip Erase, BaseAddress 0x ";
CSTR ErasingMsg[]    = "\n Erase in Process, please be patient ... ";
```

OBSOLETE

```
DWORD FBaseAdrMsk   = 0x00FFFF;              /* 512 KByte Address Range Mask */
DWORD FlashBaseAddr = 0x180000;
WORD ManCode, DevCode;
BYTE InpStr[MaxInpStrLen];

typedef const struct {
  const BYTE SecAdrIdx;
  const DWORD SecSAdr;
  const DWORD SecEAdr;
} SegmentInfoType;

SegmentInfoType SecInfoTA[] = {
 { 0x00, 0x00000, 0x0FFFF },

{ 0x08, 0x10000, 0x1FFFF },
 { 0x10, 0x20000, 0x2FFFF },
 { 0x18, 0x30000, 0x3FFFF },
 { 0x20, 0x40000, 0x4FFFF },
 { 0x28, 0x50000, 0x5FFFF },
 { 0x30, 0x60000, 0x6FFFF },
 { 0x38, 0x70000, 0x77FFF },
 { 0x3C, 0x78000, 0x79FFF },
 { 0x3D, 0x7A000, 0x7BFFF },
 { 0x3E, 0x7C000, 0x7FFFF }
};

SegmentInfoType SecInfoBA[] = {
 { 0x00, 0x00000, 0x03FFF },
 { 0x02, 0x04000, 0x05FFF },
 { 0x03, 0x06000, 0x07FFF },
 { 0x04, 0x08000, 0x0FFFF },
 { 0x08, 0x10000, 0x1FFFF },
 { 0x10, 0x20000, 0x2FFFF },
 { 0x18, 0x30000, 0x3FFFF },
 { 0x20, 0x40000, 0x4FFFF },
 { 0x28, 0x50000, 0x5FFFF },
 { 0x30, 0x60000, 0x6FFFF },
 { 0x38, 0x70000, 0x7FFFF }
};

SegmentInfoType *pAcSecInfo;


unsigned char RxUART();
void TxUART(unsigned char Data);
int StUART();
int CheckTpUart();
void TxString(const char *pStr);


BYTE RdStr(BYTE __far *pStr,BYTE MaxLen)
{
  BYTE Ch;
  WORD Ix;
       Ix = 0;
            while (((Ch=RxUART())!=13)&&(Ix<MaxLen)) {
              if ((Ch==8)&&(Ix)) {
                Ix--;
                TxString(BackSpaceSeq); }
              else if (Ch>' ') {
                pStr[Ix++] = Ch;
                TxUART(Ch); }
            }
            pStr[Ix] = 0;                    /* Replace Last Char by 0-Termination */
            return Ch;
}
```

OBSOLETE

```
void PrHexB(BYTE Dat)
{
  BYTE i, Nib;
     for (i=0;i<2;i++) {
               Nib = Dat >> 4;
               Dat = Dat << 4;
               if (Nib < 10) Nib += '0';
               else          Nib += ('A'-10);
               TxUART(Nib);
     }
}
void PrHexW(WORD Dat)
{
  BYTE i, Nib;
     for (i=0;i<4;i++) {
               Nib = Dat >> 12;
               Dat = Dat << 4;
               if (Nib < 10) Nib += '0';
               else          Nib += ('A'-10);
               TxUART(Nib);
     }
}
void PrHexD(DWORD Dat)
{
  BYTE i, Nib;
  DWORD TDat;
     for (i=0;i<8;i++) {
               Nib = (TDat = Dat >> 28);
               Dat = Dat << 4;
               Nib &= 0x0F;
               if (Nib < 10) Nib += '0';
               else          Nib += ('A'-10);
TxUART(Nib);
     }
}

void PrHexAd(DWORD Dat)
{
  BYTE i, Nib;
  DWORD TDat;
     for (i=0;i<6;i++) {
               Nib = (TDat = Dat >> 20);
               Dat = Dat << 4;
               Nib &= 0x0F;
               if (Nib < 10) Nib += '0';
               else          Nib += ('A'-10);
               TxUART(Nib);
     }
}
```

OBSOLETE

```
BYTE HexToBinD(BYTE __far *pStr, DWORD *pDat)
{
  BYTE Ch, ix, OkF;
  DWORD TDat;
     ix   = 0;
     TDat = 0;
     OkF  = 1;
     while (Ch=pStr[ix++]) {
              Ch = toupper(Ch);
              TDat = TDat << 4;
              if (Ch >= 'A') Ch -= ('A'-10);
              else           Ch -= '0';
              if (Ch < 16) {
                TDat |= Ch; }
              else {
                OkF = 0; }
     } /* while */
     if (OkF) *pDat = TDat;
     return OkF;
}

/*****************************************************/
/**          Info routine                           **/
/*****************************************************/
```

OBSOLETE

```
void FlashInfo()
{
  WORD i;
  WORD SecWordLen;
  DWORD SectAdr;

            TxString(InfoMsg);
            FL_FastReset(FlashBaseAddr);

            FL_AutoSelect(FlashBaseAddr, &ManCode, &DevCode);
            TxString(ManuMsg);
            PrHexW(ManCode);
            if (ManCode == 0x0004) TxString(FujiMsg);
            else                   TxString(UnknMsg);
            TxString(DeviMsg);
            PrHexW(DevCode);
            if      (DevCode == 0x2223) {
              TxString(F400TAMsg);
              pAcSecInfo = SecInfoTA; }
            else if (DevCode == 0x22AB) {
              TxString(F400BAMsg);
              pAcSecInfo = SecInfoBA; }
            else {
              TxString(UnknMsg);
              return; }

            TxString(SecInfoMsg);
            TxString(BaseAdrMsg);
            PrHexD(FlashBaseAddr);
            if (FlashBaseAddr & FBaseAdrMsk) TxString(NotBeginMsg);

            for (i=0;i<nSectors;i++) {

              SecWordLen = (pAcSecInfo[i].SecEAdr - pAcSecInfo[i].SecSAdr + 1) / 2;
              SectAdr = pAcSecInfo[i].SecAdrIdx;      /* Get Sector OffsetValue */
              SectAdr = SectAdr << 13;                /* adjust to required position*/
              SectAdr |= FlashBaseAddr;
                        TxString(CrLfMsg);
                        PrHexAd(SectAdr);
                        TxString(SecNLMsg);
                        PrHexB((BYTE)i);                        /* Actual Sector Index */
                        TxUART(' ');
            PrHexAd(pAcSecInfo[i].SecSAdr);
                        TxString(SecToMsg);
                        PrHexAd(pAcSecInfo[i].SecEAdr);
                        if (FL_GetSectProt(SectAdr)==0) {
                          TxString(NoProtectMsg); }
                        else if (FL_GetSectProt(SectAdr)==1) {
                          TxString(ProtectMsg); }
                        else {
                          TxString(ProtErrMsg); }

                        if (FL_CheckEmpty(SectAdr,SecWordLen)) {
                          TxString(EmptyMsg); }
                        else {
                          TxString(NotEmptyMsg); }
                      }
            }
```

OBSOLETE

```
    /*---------------------------------*/
    /*--  Routines for external UART -*/
    /*---------------------------------*/
int CheckTpUart()
{
  unsigned short Stat;
            UARTcnt  = 0x0080;                  /* Reset TP Uart */
            UARTcnt  = 0x0000;                  /* DeActivate Reset again */
            UARTdata = 0x00C1;                  /* Mode: 8-bit, 2 Stop, No Parity */
            UARTdata = 0x0000;                  /* No Interrupt Enable */
            UARTdata = 0x000D;                  /* Baud Rate 9600 */
            Stat = UARTcnt;                     /* read Status */
            Stat = Stat & 0xFC;           /* Check Bits 7..3 */
            if (Stat!=0x00) return 0;     /* not 0, thenn error */
            UARTcnt  = 0x0064;                  /* write control bits */
            Stat = UARTcnt;                     /* read Status again */
            Stat = Stat & 0xFC;           /* Check Bits 7..3 */
            if (Stat!=0x40) return 0;     /* not 0x40, thenn error */
            return 1;                     /* else return OK */
}
/*---------------------------------*/
void TxString(const char *pStr)
{
  unsigned char Ch;

            while (Ch = *pStr++) {
              if (Ch==10) TxUART(13);
              TxUART(Ch);
            }
}


/*---------------------------------*/
void TxUART(unsigned char Data)
{
  unsigned short Stat;

            do {
              Stat = UARTcnt;                   /* read Status */
              Stat = Stat & 0x40;
            } while (!Stat);
            UARTdata = Data;                    /* No Interrupt Enable */
}

/*---------------------------------*/
unsigned char RxUART()
{
            while (!(UARTcnt&0x80));            /* wait until character received */
            return UARTdata;
}


/*---------------------------------*/
int StUART()
{
            if (!(UARTcnt&0x80)) return 0;      /* if no character pending */
            else return 1;
}
```

OBSOLETE

```
        /*-----------------------------------------------*/
        /* Main Unit */
        /*-----------------------------------------------*/

void main ()
{

BYTE Key;
 DWORD MemAdr;
 WORD Data;
 DWORD TDat;

        while (!CheckTpUart()) ;        /* Initialize Uart */

        TxString(InitMsg);              /* dump message */

        FlashInfo();                            /* show info first */

        do {

         TxString(CommandMsg);                  /* display menu */
         Key = RxUART();                /* reveice key code */

         if (Key=='r') {                        /*---- Read Command ----*/
           TxString(ReadAdrMsg);
           RdStr(InpStr,MaxInpStrLen);
           if (HexToBinD(InpStr,&MemAdr)) {
             Data = FL_ReadWord(MemAdr);
             TxString(ReadDataMsg);
             PrHexW(Data);
           }
         }

         else if (Key=='p') {                   /*---- Program Command ----*/
           TxString(ProgAdrMsg);
           RdStr(InpStr,MaxInpStrLen);
           if (HexToBinD(InpStr,&MemAdr)) {
             if ((!(MemAdr&1))&&(FL_ReadWord(MemAdr)==0xFFFF)) {
               TxString(WriteDataMsg);
               RdStr(InpStr,MaxInpStrLen);
               if (HexToBinD(InpStr,&TDat)) {
                       Data = TDat;
                 if (FL_WriteWord(MemAdr,Data)) TxString(OkMsg);
                 else TxString(NotOkMsg);
               }
             }
             else {
               TxString(LocNotEmptMsg); }
           }
         }

         else if (Key=='s') {                   /*-- Sector Erase Command --*/
           TxString(SecEraMsg);
           RdStr(InpStr,MaxInpStrLen);
           if (HexToBinD(InpStr,&MemAdr)) {
             if ((Data=MemAdr) < nSectors) {
               TxString(ErasingMsg);
                     MemAdr = pAcSecInfo[Data].SecAdrIdx; /* Get Sector OffsetValue */
                     MemAdr = MemAdr << 13;             /* adjust to required position*/
                     MemAdr |= FlashBaseAddr;
               if (FL_SectorErase(MemAdr)) TxString(OkMsg);
               else                        TxString(NotOkMsg);
             }
           }
         }
```

OBSOLETE

```
    else if (Key=='e') {                    /*-- Chip Erase Command --*/
      TxString(CompEraMsg);
      RdStr(InpStr,MaxInpStrLen);
      if (HexToBinD(InpStr,&MemAdr)) {
        TxString(ErasingMsg);
        if (FL_ChipErase(MemAdr)) TxString(OkMsg);
        else                      TxString(NotOkMsg);
      }
    }

    else if (Key=='i') {                    /*---- Read Command ----*/
      FlashInfo(); }

  } while (1);
}
/*--------------------------------*/
```

OBSOLETE

```
/*------------------------------------------------------------------------*/
/*                      F U J I T S U                                     */
/*            M i k r o e l e k t r o n i k   G m b H                     */
/*                                                                        */
/*  Filename:          FLA.C                                              */
/*  Function:          Basic Routines for FLASH operations                */
/*                                                                        */
/*  Series: FR30                                                          */
/*  Version:           V01.00                                             */
/*  Design:            E. Bendels                                         */
/*  Change:            M. Mierse Dec-97                                   */
/*------------------------------------------------------------------------*/

#include "fla.h"                        /* Flash Functions */

#define AutoSelCmd 0x9090
#define ProgramCmd 0xA0A0
#define SecEraCmd1 0x8080
#define SecEraCmd2 0x3030
#define ChpEraCmd2 0x1010
#define LoAdr5555 0xAAAA                /* Address 5555 is AAAA physically ! */
#define LoAdrAAAA 0x5554                /* Address AAAA is 5554 physically ! */
#define LoAdrxx00 0x0000                /* Address xx00 is xx00 physically ! */
#define LoAdrxx01 0x0002                /* Address xx01 is xx02 physically ! */
#define LoAdrxx02 0x0004                /* Address xx02 is xx04 physically ! */
                                        /* since A1 of Micro => A0 of Flash */
/* Note: OddAddress would lead to a 2 cycle Access ! */

/*-------------------------------------------------*/
/* send "unlock"-sequence */
/*-------------------------------------------------*/
void GenerateSequence(DWORD FAdr, WORD Cmd)
{
  WORD __far *pAAAA;                    /* some Pointers to Flash Memory */
  WORD __far *p5555;                    /* some Pointers to Flash Memory */

        p5555 = (WORD __far*) (( FAdr & 0xFFFE0000) | LoAdr5555);
        pAAAA = (WORD __far*) (( FAdr & 0xFFFE0000) | LoAdrAAAA);

        *p5555 = 0xAAAA;                /* Data = AA */
        *pAAAA = 0x5555;                /* Data = 55 */
        *p5555 = Cmd;                       /* Data=Command Byte (Word)*/
}

/*-------------------------------------------------*/
/*-------------------------------------------------*/
WORD FL_AutoSelect(DWORD FAdr, WORD *MCode, WORD *DCode)
{
  WORD __far *pAdr;

        GenerateSequence( FAdr, AutoSelCmd);
        pAdr = (WORD __far*) (( FAdr & 0xFFFE0000) | LoAdrxx00);
        *MCode = *pAdr;                 /* read Device Code */

        pAdr = (WORD __far*) (( FAdr & 0xFFFE0000) | LoAdrxx01);
        *DCode = *pAdr;                         /* read Manufacture Code */

        FL_FastReset(FAdr);                     /* Terminate AutoMode */
        return 1;
}
```

OBSOLETE

```
/*-----------------------------------------------*/
/*-----------------------------------------------*/
WORD FL_GetSectProt(DWORD FAdr)
{
  WORD __far *pAdr;
  WORD Flag;

        GenerateSequence( FAdr, AutoSelCmd);
        pAdr = (WORD __far*) (( FAdr & 0xFFFE0000) | LoAdrxx02);
        Flag = *pAdr;                                   /* Protect Bit Status */
        FL_FastReset(FAdr);                             /* Terminate AutoMode */
        return Flag;
}

/*-----------------------------------------------*/
/*-----------------------------------------------*/
void FL_FastReset(DWORD FAddr)
{
  WORD __far *pAdr;

        pAdr = FAddr & 0xFFFE0000;
        *pAdr = 0xF0F0;
}

/*-----------------------------------------------*/
/*-----------------------------------------------*/
WORD FL_ReadWord(DWORD FAddr)
{
  WORD __far *pAdr;
  WORD Dat;

        pAdr =  (WORD __far*) FAddr;
        Dat = *pAdr;
        return Dat;
}

/*-----------------------------------------------*/
/*
/*-----------------------------------------------*/
WORD FL_WriteWord(DWORD FAdr, WORD Data)
{
  WORD __far *pAdr;
  WORD TDat;

        pAdr =  (WORD __far*) FAdr;
        GenerateSequence( FAdr, ProgramCmd);
        *pAdr = Data;
        do {
          TDat = *pAdr;
          if (TDat==Data) return 1;
          else if ((TDat&0x0028)==0x0028) return 0;      /* Failure Case */
        } while (1);
}
```

OBSOLETE

```
/*-----------------------------------------------*/
/* erase one sector */
/*-----------------------------------------------*/
WORD FL_SectorErase(DWORD FAdr)
{
  WORD __far *pAAAA;                    /* some Pointers to Flash Memory */
  WORD __far *p5555;                    /* some Pointers to Flash Memory */
  WORD __far *pAdr;

  WORD TDat;

            pAdr =  (WORD __far*) FAdr;
            p5555 = (WORD __far*) (( FAdr & 0xFFFE0000) | LoAdr5555);
            pAAAA = (WORD __far*) (( FAdr & 0xFFFE0000) | LoAdrAAAA);

            *p5555 = 0xAAAA;            /* Data = AA */
            *pAAAA = 0x5555;            /* Data = 55 */
            *p5555 = SecEraCmd1;                /* Erase Command 1 */

            *p5555 = 0xAAAA;            /* Data = AA */
            *pAAAA = 0x5555;            /* Data = 55 */
            *pAdr  = SecEraCmd2;        /* Erase Command 2 */

            do {
              TDat = *pAdr;                                 /* poll bit */
              if (TDat&0x0080) return 1;                    /* if finished OK */
              else if ((TDat&0x0028)==0x0028) return 0;     /* else if failure */
            } while (1);
}

/*-----------------------------------------------*/
/* erase every sector */
/*-----------------------------------------------*/
WORD FL_ChipErase(DWORD FAdr)
{

  WORD __far *pAdr;
  WORD TDat;

            FAdr &= 0xFFFE0000;                             /* Make sure EvenAddress */
            pAdr =  (WORD __far*) FAdr;

            GenerateSequence( FAdr, SecEraCmd1);
            GenerateSequence( FAdr, ChpEraCmd2);

            do {
              TDat = *pAdr;                                 /* poll bit */
              if (TDat&0x0080) return 1;                /* if finished OK */
              else if ((TDat&0x0028)==0x0028) return 0;     /* else if failure */
            } while (1);
}
```

OBSOLETE

```
/*-------------------------------------------------*/
/* checks if a sector is empty (every byte = FF)  */
/*-------------------------------------------------*/

WORD FL_CheckEmpty(DWORD FAdr,WORD SecWordLen)
{
  WORD __far *pAdr;
  WORD ix, Flag, Dat;

            pAdr =  (WORD __far*) FAdr;
            Flag = 1;

            for (ix=0;ix<SecWordLen;ix++) {
              if ((Dat=*pAdr++)!=0xFFFF) Flag = 0; }
            return Flag;
}
```

OBSOLETE

Document History

Document Title: AN205213 - FR30 example interface to external Flash Memory

Document Number:002-05213

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	MM	10/13/1999	V1.0, New Format, new updated version
		MEN	07/04/2000	V1.1 Updated Application
*A	5080479	MKEA	01/13/2016	Converted Spansion Application Note “MCU-AN-391015-E-V11” to Cypress format
*B	5560875	WOFR	20/12/2016	Obsoleted

OBSOLETE

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Automotive	cypress.com/go/automotive
Clocks & Buffers	cypress.com/go/clocks
Interface	cypress.com/go/interface
Lighting & Power Control	cypress.com/go/powerpsoc
Memory	cypress.com/go/memory
PSoC	cypress.com/go/psoc
Touch Sensing	cypress.com/go/touch
USB Controllers	cypress.com/go/usb
Wireless/RF	cypress.com/go/wireless
Spansion Products	cypress.com/spansionproducts

PSoC® Solutions

psoc.cypress.com/solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP

Cypress Developer Community

Community | Forums |Blogs | Video |Training

Technical Support

cypress.com/go/support

OBSOLETE

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com

© Cypress Semiconductor Corporation, 1999-2016. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

This Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.