AN1673

Using the PIC16F1XXX High-Endurance Flash (HEF) Block

Author: *Lucio Di Jasio*
Microchip Technology Inc.

INTRODUCTION

The PIC16F1XXX family of general purpose Flash microcontrollers features the 8-bit PIC® MCU enhanced mid-range core. Carefully trading functionality versus cost, several members of this family, including the PIC16F14XX, PIC16F15XX and PIC16F17XX, have made a departure from the usual set of peripherals found in previous models to achieve a lower price point while still offering a compelling new set of features. Among the several new peripherals introduced, it is worth noting:

- Configurable Logic Cell – a small set of logic blocks (unlike a small PLD) that can help directly interconnect various peripherals inputs/outputs without CPU intervention.
- Complementary Output Generator – the front end of a traditional PWM module, now decoupled from the timing generator and, therefore, available for direct connection to other modules (including analog comparators).
- Numerically Controlled Oscillator – a frequency synthesizer that allows linear control of the frequency output.
- High-Endurance Flash (HEF) Block – replacing the data EEPROM present on previous models with a block of Flash memory that is ensured to provide the same high endurance (100,000 erase/write cycles).

This application note illustrates the benefits offered by the high-endurance Flash block and describes a small C library which allows a simple and safe use of its capabilities.

FLASH VS. HIGH-ENDURANCE FLASH

Like most other PIC microcontrollers in Flash technology, the PIC16F1XXX series features a single-voltage self-write Flash program memory array.

This means that, without additional external hardware support, these devices can modify the contents of their Flash memory at runtime, under firmware control.

As an example, this capability is conveniently used to implement boot loaders, enabling embedded application that can be reprogrammed in the field via a simple serial connection (UART, SPI, I^{2}C™, USB, etc.) and without requiring the use of a dedicated in-circuit programmer/debugger device.

This capability can also be used to store and/or update calibration data in program memory (obtained at the end of a production line or after product installation).

However, the main limitation of the self-write Flash program memory array lies in the relatively small number of possible erase/write cycles.

For example, the PIC16(L)F1508/9 data sheet (DS40001609) specifies a minimum of 10K cycles in the industrial temperature range (-40˚C to +85˚C) (see http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en553471). This is an order of magnitude smaller than the traditional value offered by most data EEPROM modules and stand-alone devices.

The high-endurance block is a block of 128 memory locations, found at the top of the Flash program memory that is ensured to provide a superior endurance, equal to that of a traditional data EEPROM memory within a given temperature range (0˚C to 60˚C) (see Table 1 below from the PIC16(L)F1508/9 data sheet).

TABLE 1: MEMORY PROGRAMMING SPECIFICATIONS

Standard Operating Conditions (unless otherwise stated)							
Param. No.	**Sym.**	**Characteristic**	**Min.**	**Typ.†**	**Max.**	**Units**	**Conditions**
		Program Memory Programming Specifications					
D110	VIHH	Voltage on $\overline{MCLR}$/VPP pin	8.0	—	9.0	V	**(Note 2)**
D111	IDDP	Supply Current during Programming	—	—	10	mA	
D112	VBE	VDD for Bulk Erase	2.7	—	VDDMAX	V	
D113	VPEW	VDD for Write or Row Erase	VDDMIN	—	VDDMAX	V	
D114	IPPPGM	Current on $\overline{MCLR}$/VPP during Erase/Write	—	1.0	—	mA	
D115	IDDPGM	Current on VDD during Erase/ Write	—	5.0	—	mA	
		Program Flash Memory					
D121	EP	Cell Endurance	10K	—	—	E/W	-40°C ≤ TA ≤ +85°C **(Note 1)**
D122	VPRW	VDD for Read/Write	VDDMIN	—	VDDMAX	V	
D123	TIW	Self-timed Write Cycle Time	—	2	2.5	ms	
D124	TRETD	Characteristic Retention	—	40	—	Year	Provided no other specifications are violated
D125	EHEFC	High-Endurance Flash Cell	100K	—	—	E/W	0°C ≤ TA ≤ +60°C, lower byte last 128 addresses

† Data in “Typ” column is at 3.0V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

Note 1: Self-write and Block Erase.

2: Required only if single-supply programming is disabled.

Compare parameter D121, EP, Cell Endurance of a regular Flash program memory cell, to parameter D125, EHEFC, High-Endurance Flash Cell.

It is also worth noting that each high-endurance cell can be used only to hold an 8-bit value, whereas the standard Flash memory will hold 14 bits of information as per the traditional PIC MCU mid-range program memory array design.

FLASH VS. EEPROM

Both the high-endurance Flash and the regular Flash memory arrays differ from a data EEPROM module in two important ways:

a) Data must be manually erased before a write and this can be performed only in blocks (referred to as *rows*) of a fixed size determined by the Flash array inner design.

b) Writing to Flash stalls the MCU for a few milliseconds (see parameter D123 in Table 1).

By contrast, true data EEPROMs are designed to allow byte-by-byte erase and do not stall the MCU execution during a write cycle, although, for simplicity reasons, most applications will include a delay loop to ensure a (byte) write has been completed before the next one can be initiated.

Note:	To speed up the writing process, external serial EEPROM devices will often include a *page* mechanism that is not too dissimilar in principle to the rows of a Flash array.

The differences noted above have implications for the design of the embedded application, which need to be fully understood whether considering the use of regular or high-endurance Flash in lieu of a true EEPROM.

Row Erase/Write

Notice that no restrictions exist on the amount of data fetched during a read operation or its alignment with row boundaries.

When writing a block of data to a Flash memory, it is important to ensure alignment with the array row boundaries. When the data block does not entirely fit inside a single row, multiple write operations can be required to store sequential rows, which explains why it is essential to know the size of the Flash memory array (row size). The user needs to ensure that the data is aligned with the row size, especially if data spans more than one row.

Stall During Write

In contrast with the behavior of a data EEPROM, the MCU is unable to fetch new instructions from the Flash memory array during a self-write cycle and it is, therefore, stalled. This means that no other work can be performed in parallel during an erase or write operation, nor can the application respond to interrupts. In fact, these are delayed until the MCU code execution resumes. The designer must, therefore, take into account the potentially-introduced additional jitter or the momentary application freeze during the nonvolatile memory updates.

Note that the delay duration is independent of the amount of data actually written in a row, be it a single byte or the entire row.

Finally, note that no delays or penalties are incurred by reading the content of the Flash memory arrays.

A FLASH SUPPORT LIBRARY

The `flash.c` module has been written to simplify the use of the self-write Flash memory in embedded applications written in C (see Listing B-2). It can be used to operate independently on the normal Flash array as well as on the high-endurance Flash block.

Five basic functions are prototyped and documented in `flash.h` (see Listing B-1).

The first three functions provide read capability (see Function 1 through Function 3).

FUNCTION 1: FLASH_read

```
/**
 * Read a word from program Flash memory
 *
 * @param address source address (absolute
                  Flash memory address)
 * @return        word retrieved from
                  Flash memory
 */
unsigned FLASH_read (unsigned address);
```

The `FLASH_read` function reads a 14-bit word from a given address in the device program Flash memory. It can be used to read very large tables of data from memory.

FUNCTION 2: FLASH_readConfig

```
/**
 * Read a word from configuration Flash memory
 *
 * @param address   source address (absolute
                    Flash memory address)
 * @return          word retrieved from
                    Flash memory
 */
unsigned FLASH_readConfig (unsigned address);
```

The `FLASH_readConfig` function is a variant of the previous function that allows access to addresses above the 0x8000 threshold where the IDLOC information, processor Configuration bits and calibration data are found.

FUNCTION 3: FLASH_readBlock

```
/**
 * Read a block of words from program Flash memory
 *
 * @param buffer    destination buffer (must be
                    sufficiently large)
 * @param address   source address (absolute
                    Flash memory address)
 * @param count     number of words to be
                    retrieved
 */
void FLASH_readBlock (unsigned* buffer, unsigned
address, char count);
```

The `FLASH_readBlock` function will perform a read operation from the main program memory array of the device transferring a number of 14-bit words (up to 255) into a buffer of 16-bit unsigned integers.

The last two functions provide the write and erase capability (see Function 4 and Function 5).

FUNCTION 4: FLASH_write

```
/**
 * Write a word of data to Flash memory (latches)
 * if parameter latch = 1, data is only latched
   and no actual write cycle
 * is initiated until a subsequent call with
   latch = 0
 *
 * @param address   destination address (absolute
                    Flash memory)
 * @param data      word of data to be written
                    (latched)
 * @param latch
 */
void FLASH_write (unsigned address, unsigned data,
byte latch);
```

The `FLASH_write` function offers the smallest atomic unit required to perform an actual write operation. Depending on the value passed as the latch parameter, it can simply load a latch or initiate a write cycle during which all values previously loaded in a row get written at once. Refer to **Section 10.0, Flash Program Memory Control**, of the device data sheet for more details on the low-level sequence of operation required (see http://ww1.microchip.com/downloads/en/DeviceDoc/40001609C.pdf).

> **Note:** When latching, the function performs a short unlock sequence which includes the temporary disabling of interrupts, but no delay (MCU stall) is incurred. During an actual write operation, the function performs the unlock sequence, but it also stalls the MCU for the self-write cycle time. Any interrupt event occurring during such time will be served after execution is resumed.

FUNCTION 5: FLASH_erase

```
/**
 * Erase a row of Flash memory
 *
 * @param address     absolute address in Flash
                      contained in selected row
 */
void FLASH_erase (unsigned address);
```

The `FLASH_erase` function performs the erase of a single row of the program memory Flash array. The address provided can be relative to any of the locations contained in the row. This function performs a brief unlock sequence and then proceeds to stall the MCU for the self-write cycle time. Any interrupt event occurring during such time will be served after execution is resumed.

Note: It is not necessary to precede each write operation with a corresponding erase operation. In fact an erase is necessary only when a memory location content update requires a bit presently set to a logic '0', to return to a logic value of '1'. After an erase, every location contained in the given memory row will be set to the value of 0x3FFF (14 bits set to '1').

Write operations can be performed sequentially or simultaneously on one or more locations within a row (using the latching mechanism) without any specific order. Each memory location can be written to several times, as long as each time, only bit-clear operations are requested ('1'→'0').

EEPROM-LIKE USE OF THE HIGH-ENDURANCE FLASH

Utilizing the five functions presented in the previous section, it is possible to implement any number of strategies for the implementation and management of a nonvolatile storage system on Flash.

Several application notes and articles have already presented solutions to help emulate a high-endurance block using multiple locations of (lower endurance) Flash memory. See AN1095 for use with PIC18, PIC24 and dsPIC33FJ devices (http://ww1.microchip.com/downloads/en/AppNotes/01095D.pdf).

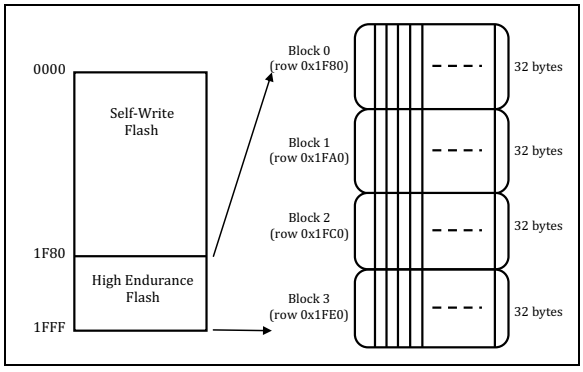
The "*High-Endurance Flash Block"* application note will instead take advantage of the superior characteristics of the high-endurance Flash array, available on the current PIC16F1XXX family and its future derivatives.

Since high endurance is already ensured by design, the approach presented will require no additional Flash memory space outside of the high-endurance Flash region and no additional resources.

A simple library module (`HEFlash.c`, `HEFlash.h`) has been prepared to provide the simplest and most intuitive use of the high-endurance Flash region for data EEPROM-like nonvolatile storage (admittedly more similar to a paged data EEPROM use).

For maximum efficiency, the high-endurance Flash library handles the high-endurance Flash as a small array of data blocks, mapping each block to a row, adding up to 128 bytes of total capacity (see Figure 1).

FIGURE 1: HIGH-ENDURANCE MEMORY USED AS AN ARRAY OF FOUR BLOCKS ON PIC16F1509



The actual capacity of each data block is dictated by the dimensions of the high-endurance Flash memory row in the given microcontroller model:

- PIC16F1XXX models with 4K words of Flash (7K bytes), or more, have a row size of 32 bytes.
- Smaller devices use a row size of 16 bytes.

This means that 4K words and larger models will offer only four independent data blocks (32 bytes each) of nonvolatile storage, while smaller devices will offer eight independent data blocks (16 bytes each).

Each data block can be modified any number of times up to the maximum specified endurance of the high-endurance Flash memory (100,000 times) without influencing any of the neighboring data blocks.

FUNCTION 6: writeBlock()

```
/**
 * Write a block of data to High Endurance Flash
 * the entire block must fit within a row
 *
 * @param radd      HE Flash block number
                    (0 - MAXROWS-1)
 * @param buffer    source buffer
 * @param count     number of bytes to write to
                    block (< ROWSIZE)
 * @return          0 if successful, -1 if
                    parameter error
 */
char    HEFLASH_writeBlock (char radd,
        char* buffer, char count);
```

The `writeBlock()` function takes the contents of a data structure pointed to by *buffer* and writes it into the selected high-endurance Flash block. Blocks are atomically updated. All the contents of a block are erased first and then written to, even if only a portion of the block capacity (`count<ROWSIZE`) is used.

The function will return a parameter error (‘`-1`’) if any of the input parameters are out of bounds, such as data too large or invalid block number. The function will return a write error (‘`1`’) if the write sequence failed for any reason.

The value zero (‘`0`’) will be returned in case of success (see Function 6).

FUNCTION 7: readBlock()

```
 /**
 * Read a block of data from HE Flash memory
 *
 * @param buffer  destination buffer (must be
                  sufficiently large)
 * @param radd    source block of HE Flash memory
                  (0 - MAXROWS-1)
 * @param count   number of bytes to be retrieved
                  (< ROWSZE)
 * @return        0 if successful, -1 if parameter
                  error
 */
char    HEFLASH_readBlock (char* buffer, char radd,
        char count);
```

The `readBlock()` function matches the functionality of the `writeBlock()` function by extracting a block of data from high-endurance Flash and placing the contents into a structure pointed to by *buffer*.

This function will return a parameter error (‘`-1`’) if any of the input parameters is out of bounds, such as data too large or invalid block number (see Function 7).

FUNCTION 8: readByte()

```
/**
 * Read a byte of data from HE Flash memory
 *
 * @param radd    source block of HE Flash memory
                  (0 - MAXROWS-1)
 * @param offset  offset within the HE block
                  (0 - ROWSIZE-1)
 * @return        byte of data retrieved
 */
char      HEFLASH_readByte (char radd,
          char offset);
```

For further convenience, the `readByte()` function provides the ability to read any individual byte contained in a high-endurance Flash memory by passing a block number and an offset (see Function 8).

RESERVING HIGH-ENDURANCE FLASH MEMORY

It is important to note that the high-endurance Flash memory region is normally assumed to be available to the C compiler for the application code storage. In order to avoid any possible conflict (overlapping code and data usage), it is important to reserve the device-specific memory range by using the `--ROM` linker switch in the project configuration.

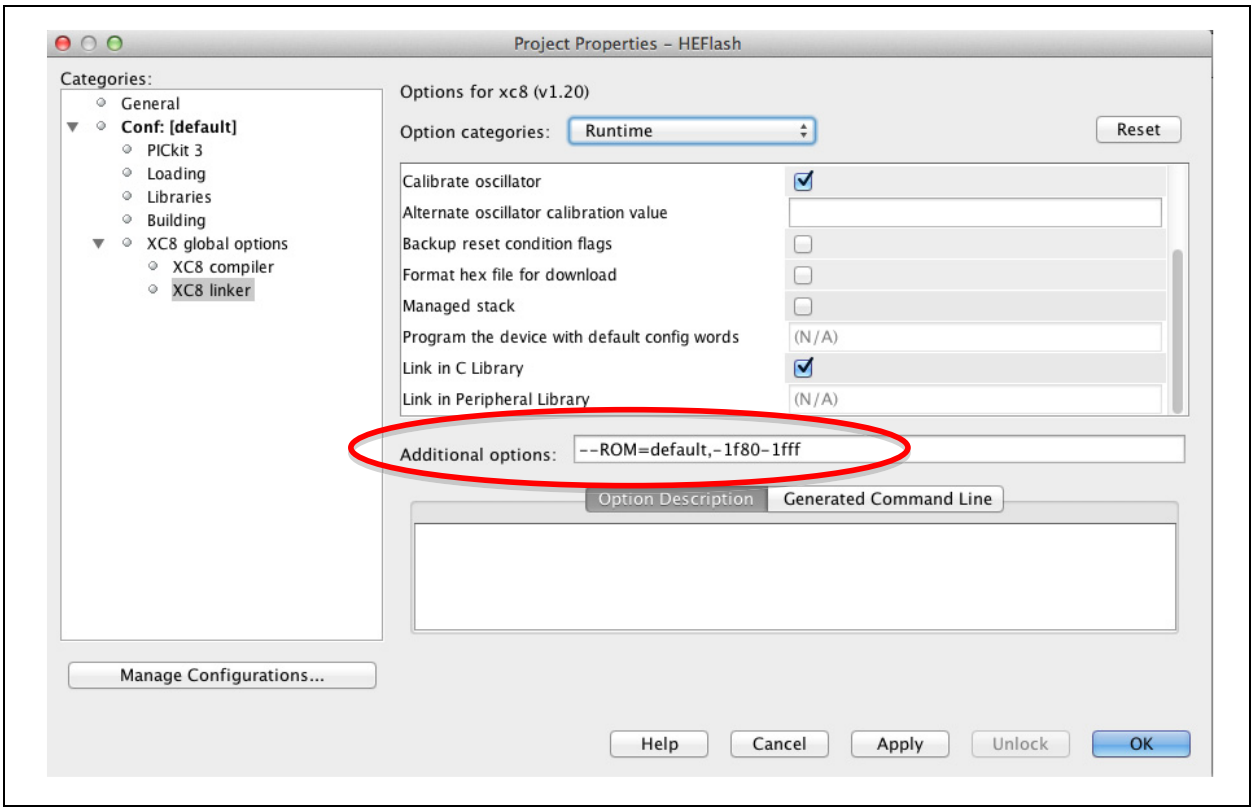
In the example in Figure 2, the region 0x1F80 to 0x1FFF (high-endurance Flash block for a PIC16F1509 microcontroller) has been removed from the default space available for code storage using the notation:

```
--ROM=default,-1f80-1fff
```

This is documented in **Section 4.8.50, --ROM: Adjust ROM Ranges,** of the "*MPLAB® XC8 C Compiler User's Guide"* (DS50002053) (see http://ww1.microchip.com/downloads/en/DeviceDoc/xc8-v1.21-manual.pdf).

Note:	Make sure to use a double dash before the `ROM` keyword and a comma immediately following `default`.

FIGURE 2: SETTING THE LINKER OPTIONS TO RESERVE THE HIGH-ENDURANCE FLASH MEMORY FOR DATA STORAGE



Device Configuration

The device Configuration bits WRT<1:0>, contained in Configuration Word CONFIG2, control the Flash memory self-write protection. Such protection should be set to off (‘11’) or limited to the lower portion of the Flash memory array (‘10’, ‘01’).

An Example of Use

Listing A-1 provides a simple test program that can be used to validate the library operation using a PICkit™ Low Pin Count Explorer Board (DM164130-9) in conjunction to a PIC16F1509 device and a PICkit 3 debugger/programmer.

The PIC16F1509 has 8K words of program memory and a row size of 32 bytes.

The high-endurance Flash memory is managed as four blocks (0-3) capable of 32 bytes of data each.

Example 1 includes the definition of a simple data structure called data of type Record.

EXAMPLE 1:

```
     typedef struct{
       unsigned ID ;     //     2 bytes
       char   Name[20];  //     20 bytes
       long   Amount;    //     4 bytes
     } Record;           //     26 bytes total
Record data = {0x1234, "HE Flash", 42};
```

This can be saved to a block (‘1’) of the high-endurance Flash (see Example 2).

EXAMPLE 2:

```
 r = HEFLASH_writeBlock(1,(void*)&data, sizeof
     (data));
```

Note the use of the `void*` cast to pass the record-type contents to the write function.

The returned value indicates whether the `writeBlock()` function accepted and completed the requested operation.

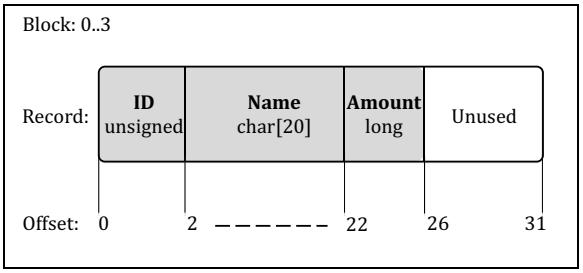
The contents of the high-endurance Flash block (‘1’) can be read back into a record-type variable using the command in Example 3.

EXAMPLE 3:

```
 r = HEFLASH_readBlock((void*)&data, 1, sizeof
     (data));
```

Once more, the returned value indicates whether the function accepted the parameters and completed the requested operation (see Figure 3).

FIGURE 3: FITTING A DATA RECORD IN A HIGH-ENDURANCE FLASH BLOCK



SUMMARY

Members of the PIC16F1XXX family of microcontrollers offer a high-endurance Flash block that is capable of 100,000 erase/write cycles. The high-endurance Flash can be effectively used to provide nonvolatile storage, with EEPROM-like endurance and simplicity, to low-cost embedded control applications.

The code provided supports and simplifies the management of the high-endurance Flash block, as well as the access to the standard Flash memory array.

APPENDIX A: EXAMPLE PROJECT

Software License Agreement

The software supplied herewith by Microchip Technology Incorporated (the “Company”) is intended and supplied to you, the Company’s customer, for use solely and exclusively with products manufactured by the Company.

The software is owned by the Company and/or its supplier, and is protected under applicable copyright laws. All rights are reserved. Any use in violation of the foregoing restrictions may subject the user to criminal sanctions under applicable laws, as well as to civil liability for the breach of the terms and conditions of this license.

THIS SOFTWARE IS PROVIDED IN AN “AS IS” CONDITION. NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

LISTING A-1: Main.c

```
/*
 * File:   main.c
 * Author: Lucio Di Jasio
 *
 * Created on August 28, 2013
 */
#include "Flash.h"
#include "HEFlash.h"
#include <assert.h>
#include <string.h>

//  Configuration Bit Settings

__IDLOC( 4D8A);

// CONFIG1
#pragma config FOSC = INTOSC    // Oscillator Selection Bits (INTOSC oscillator: I/O function on CLKIN pin)
#pragma config WDTE = OFF       // Watchdog Timer Enable (WDT disabled)
#pragma config PWRTE = OFF      // Power-up Timer Enable (PWRT disabled)
#pragma config MCLRE = ON       // MCLR Pin Function Select (MCLR/VPP pin function is MCLR)
#pragma config CP = OFF         // Flash Program Memory Code Protection
                                   (Program memory code protection is disabled)
#pragma config BOREN = ON       // Brown-out Reset Enable (Brown-out Reset enabled)
#pragma config CLKOUTEN = OFF   // Clock Out Enable
                                   (CLKOUT function is disabled. I/O or oscillator function on the CLKOUT pin)
//# pragma config IESO = ON     // Internal/External Switchover Mode
                                   (Internal/External Switchover mode is enabled)
//# pragma config FCMEN = ON    // Fail-Safe Clock Monitor Enable (Fail-Safe Clock Monitor is enabled)

//CONFIG2
#pragma config WRT = OFF        // Flash Memory Self-Write Protection (Write protection off)
#pragma config STVREN = ON      // Stack Overflow/Underflow Reset Enable
                                   (Stack Overflow or Underflow will cause a Reset)
#pragma config BORV = LO        // Brown-out Reset Voltage Selection
                                   (Brown-out Reset Voltage (VBOR), low trip point selected.)
#pragma config LPBOR = OFF      // Low-Power Brown Out Reset (Low-Power BOR is disabled)
#pragma config LVP = OFF        // Low-Voltage Programming Enable
                                   (High-voltage on MCLR/VPP must be used for programming)

void _fassert (int line, const char *file, const char *expr)
{
    TRISC = 0xf0;
    PORTC = PORTC+1;
} //_fassert

void main(void)
{
    unsigned r, i;
    unsigned wbuffer[4];
    char buffer [FLASH_ROWSIZE];
    typedef struct {
        unsigned ID;
        char     Name[20];
        long     Amount;
    } Record;
    Record data = {0x1234, "HE FLASH", 42};

    //testing the FLASH read (word in program memory)
    r = FLASH_read(0x0001);
    assert (r == 0x3180);

    //read the first IDLOC
    r = FLASH_readConfig(0x8000);
    assert (r == 4);

    //testing the HE FLASH read block (bytes in HE Flash)
    r = HEFLASH_readBlock (buffer, 1, FLASH_ROWSIZE);
    assert (r == 0);
```

LISTING A-1: `Main.c` (continued)

```
      //write data to HE row 1
    r = HEFLASH_writeBlock(1, (void*)&data, sizeof(data));
    assert (r == 0);

    //empty the buffer
    memset(&data, 0, sizeof(data));
    //data.ID = 0;
    //data.Name[0] = '\0';
    //data.Amount = 0L;

    // read back its contents
    r = HEFLASH_readBlock((void*)&data, 1, sizeof(data));
    assert (r == 0);

    //read a single byte
    r = HEFLASH_readByte(1, 5);
    assert (r == 'F');
    while(1)
    {
         // stop here
    }
}
```

APPENDIX B: FLASH SUPPORT LIBRARY LISTINGS

LISTING B-1: Flash.h

```
/*
 * Flash.h
 *
 */
#include<xc.h>
#if defined(_PIC16F1501_H_)
//1K
    #define FLASH_ROWSIZE 16          //size of a row
    #define HEFLASH_START     0x0380   //first address in HE Flash memory
    #define HEFLASH_END       0x03FF   //last address in HE Flash memory
#elif defined(_PIC16F1503_H_)||defined(_PIC16F1507_H_)||defined(_PIC16F1512_H_)||\
    defined(_PIC16F1703_H_)||defined(_PIC16F1707_H_)
//2K
    #define FLASH_ROWSIZE 16          //size of a row
    #define HEFLASH_START     0x0780   //first address in HE Flash memory
    #define HEFLASH_END       0x07FF   //last address in HE Flash memory
#elif defined(_PIC16F1508_H_)||defined(_PIC16F1513_H_)||\
    defined(_PIC16F1704_H_)||defined(_PIC16F1708_H_)||defined(_PIC16F1713_H_)
//4K
    #define FLASH_ROWSIZE 32          //size of a row
    #define HEFLASH_START     0x0F80   //first address in HE Flash memory
    #define HEFLASH_END       0x0FFF   //last address in HE Flash memory
#elif defined(_PIC16F1509_H_)||defined(_PIC16F1526_H_)||\
    defined(_PIC16F1454_H_)||defined(_PIC16F1455_H_)||defined(_PIC16F1459_H_)||\
    defined(_PIC16F1705_H_)||defined(_PIC16F1709_H_)||\
    defined(_PIC16F1716_H_)||defined(_PIC16F1717_H_)
//8K
    #define FLASH_ROWSIZE 32          //size of a row
    #define HEFLASH_START     0x1F80   //first address in HE Flash memory
    #define HEFLASH_END       0x1FFF   //last address in HE Flash memory
#elif defined(_PIC16F1518_H)||defined(_PIC16F1519_H)||defined(_PIC16F1527_H_)||\
    defined(_PIC16F1718_H_)||defined(_PIC16F1719_H_)
//16K
    #define FLASH_ROWSIZE 32          //size of a row
    #define HEFLASH_START     0x3F80   //first address in HE Flash memory
    #define HEFLASH_END       0x3FFF   //last address in HE Flash memory
#endif

#define FLASH_ROWMASK        FLASH_ROWSIZE-1

/*****************************************************************************
 * Generic Flash functions
 */

/**
 * Read a word from program Flash memory
 *
 * @param address   source address (absolute Flash memory address)
 * @return          word retrieved from Flash memory
 */
unsigned FLASH_read (unsigned address);

/**
 * Read a word from configuration Flash memory
 *
 * @param address   source address (absolute Flash memory address)
 * @return          word retrieved from Flash memory
 */
 unsigned FLASH_readConfig (unsigned address);

/**
 * Read a block of words from program Flash memory
 *
 * @param buffer    destination buffer (must be sufficiently large)
 * @param address   source address (absolute Flash memory address)
 * @param count     number of words to be retrieved
 */
void    FLASH_readBlock (unsigned* buffer, unsigned address, char count);
```

LISTING B-1: `Flash.h` (continued)

```
/**
 * Write a word of data to Flash memory (latches)
 * an actual write is performed only if LWLO = 0, data is latched if LWLO = 1
 *
 * @param address   destination address (absolute Flash memory)
 * @param data      word of data to be written (latched)
 * @param latch     1 = latch, 0 = write
 */
void    FLASH_write (unsigned address, unsigned data, char latch);

/**
 * Erase a row of Flash memory
 *
 * @param address   absolute address in Flash contained in selected row
 */
void    FLASH_erase (unsigned address);
```

LISTING B-2: `Flash.c`

```
/*
 * File: Flash.c
 *
 * Self-Write Flash support functions
 *
 * Author: Lucio Di Jasio
 *
 * Created on August 28, 2013
 */
#include "Flash.h"

/******************************************************************************
 * Generic Flash functions
 */

unsigned FLASH_readConfig (unsigned address)
{
    //  1.  load the address pointers
    PMADR = address;
    PMCON1bits.CFGS = 1;      //select the configuration Flash address space
    PMCON1bits.RD = 1;        //next operation will be a read
    NOP();
    NOP();
    //  2.  return value read
    return PMDAT;
}//FLASH_config

unsigned FLASH_read (unsigned address)
{
    //  1.  load the address pointers
    PMADR = address;
    PMCON1bits.CFGS = 0;      //select the Flash address space
    PMCON1bits.RD = 1;        //next operation will be a read
    NOP();
    NOP();
    //  2.  return value read
    return PMDAT;
}//FLASH_read
void FLASH_readBlock (unsigned *buffer, unsigned address, char count)
{
    while (count > 0)
    {
        *buffer++ = FLASH_read (address++);
        count--;
    }
}//FLASH_readBLock
```

LISTING B-2: `Flash.c` (continued)

```
/**
 *  unlock Flash Sequence
 */
void _unlock (void)
{
   #asm
       BANKSEL      PMCON2
       MOVLW        0x55
       MOVWF        PMCON2 & 0x7F
       MOVLW        0xAA
       MOVWF        PMCON2 & 0x7F
       BSF          PMCON1 & 0x7F,1   ; set WR bit
       NOP
       NOP
   #endasm
}//unlock

void FLASH_write (unsigned address, unsigned data, char latch)
{
   //  1.  disable interrupts (remember setting)
   char temp = INTCONbits.GIE;
   INTCONbits.GIE = 0;

   //  2.  load the address pointers
   PMADR = address;
   PMDAT = data;
   PMCON1bits.LWLO = latch;  //   1 = latch, 0 = write row
   PMCON1bits.CFGS = 0;      //   select the Flash address space
   PMCON1bits.FREE = 0;      //   next operation will be a write
   PMCON1bits.WREN = 1;      //   enable Flash memory write/erase

   //  3.  perform unlock sequence
   _unlock();

   //  4.  restore interrupts
   if (temp)
       INTCONbits.GIE = 1;

}//FLASH_write

void FLASH_erase (unsigned address)
{
   //  1.  disable interrupts (remember setting)
   char temp = INTCONbits.GIE;
   INTCONbits.GIE = 0;

   //  2.  load the address pointers
   PMADR = address;
   PMCON1bits.CFGS = 0;      //   select the Flash address space
   PMCON1bits.FREE = 1;      //   next operation will be an erase
   PMCON1bits.WREN = 1;      //   enable Flash memory write/erase

   //  3.  perform unlock sequence and erase
   _unlock();

   //  4.  disable writes and restore interrupts
   PMCON1bits.WREN = 0;      //   disable Flash memory write/erase
   if (temp)
       INTCONbits.GIE = 1;
}//FLASH_erase
```

APPENDIX C: HIGH-ENDURANCE FLASH SUPPORT LIBRARY LISTINGS

LISTING C-1: HEFlash.h

```
/*
 * HEFlash.h
 *
 */
#include "FLash.h"

#define HEFLASH_MAXROWS    (HEFLASH_END-HEFLASH_START+1)/FLASH_ROWSIZE

/*****************************************************************************
 * High Endurance Flash functions
 */

/**
 * Write a block of data to High Endurance Flash
 * the entire block must fit within a row
 *
 * @param radd             HE Flash block number(0 - MAXROWS-1)
 * @param buffer           source buffer
 * @param count            number of bytes to write to block (< ROWSIZE)
 * @return                 0 if successful, -1 if parameter error, 1 if write error
 */
char      HEFLASH_writeBlock (char radd, char* buffer, char count);


/**
 * Read a block of data from HE Flash memory
 *
 * @param buffer           destination buffer (must be sufficiently large)
 * @param radd             source block of HE Flash memory (0 - MAXROWS-1)
 * @param count            number of bytes to be retrieved (< ROWSZE)
 * @return                 0 if successful, -1 if parameter error
 */
char      HEFLASH_readBlock (char* buffer, char radd, char count);


/**
 * Read a byte of data from HE Flash memory
 *
 * @param radd             source block of HE Flash memory (0 - MAXROWS-1)
 * @param offset           offset within the HE block (0 - ROWSIZE-1)
 * @return                 byte of data retrieved
 */
char      HEFLASH_readByte (char radd, char offset);
```

LISTING C-2: HEFlash.c

```
/*
 * File: HEFlash.c
 *
 * High Endurance Flash - EEPROM emulation and support routines
 *
 * Author: Lucio Di Jasio
 *
 * Created on August 28, 2013
 */
#include "HEFlash.h"

/******************************************************************************
 * High Endurance Flash functions
 */

char  HEFLASH_writeBlock (char radd, char* data, char count)
{
      //  1.  obtain absolute address in HE FLASH row
      unsigned add = radd * FLASH_ROWSIZE + HEFLASH_START;

      //  2.  check input parameters
      if ((count > FLASH_ROWSIZE)||(radd >= HEFLASH_MAXROWS))
          return -1;//return parameter error

      //  3.  erase the entire row
      FLASH_erase (add);

      //  4.  fill the latches with data
      while (count > 1)
      {
        //load data in latches without writing
        FLASH_write (add++, (unsigned) *data++, 1);
        count--;
      }
      //  no delay here!!!

      //  5.  last byte of data -> write
      FLASH_write (add, (unsigned) *data, 0);
      //  NOTE: 2ms typ. delay here!!!

      //  6.  return success
      return PMCON1bits.WRERR;    //0 success, 1 = write error

} //HEFLASH_writeBlock

char  HEFLASH_readBlock (char *buffer, char radd, char count)
{
      //  1.  obtain absolute address in HE FLASH row
      unsigned add = radd * FLASH_ROWSIZE + HEFLASH_START;

      //  2.  check input parameters
      if ((count > FLASH_ROWSIZE)||(radd >= HEFLASH_MAXROWS))
          return -1;

      //  3.  read content
      while (count > 0)
      {
        *buffer++ = (char) FLASH_read (add++);
        count--;
      }

      //  4.  success
      return 0;
} //HEFLASH_readBlock
char HEFLASH_readByte (char radd, char offset)
{
      //  1.  add offset into HE Flash memory
      unsigned add = radd * FLASH_ROWSIZE + HEFLASH_START + offset;
      //  2.  read content
      return (char) FLASH_read (add);
} //HEFLASH_read
```

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Trademarks

Printed on recycled paper.

ISBN: 9781620779033

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Cleveland
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110

Canada - Toronto
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8569-7000
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

China - Hangzhou
Tel: 86-571-2819-3187
Fax: 86-571-2819-3189

China - Hong Kong SAR
Tel: 852-2943-5100
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8864-2200
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-3019-1500

Japan - Osaka
Tel: 81-6-6152-7160
Fax: 81-6-6152-9310

Japan - Tokyo
Tel: 81-3-6880- 3770
Fax: 81-3-6880-3771

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-5778-366
Fax: 886-3-5770-955

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Dusseldorf
Tel: 49-2129-3766400

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Pforzheim
Tel: 49-7231-424750

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Venice
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Poland - Warsaw
Tel: 48-22-3325737

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820

10/28/13