AN1204
APPLICATION NOTE

Software Drivers for the M58BF008 Flash Memory

CONTENTS

INTRODUCTION

This application note provides library source code in C for the M58BF008 Flash Memory. The M58BF008 supports both Asynchronous and Synchronous Burst bus interfaces. The C code drivers work on a layer above the hardware and can be used successfully over either bus interface.

Listings of the source code can be found at the end of this document. The source code is also available in file form from the internet site http://www.st.com or from your STMicroelectronics distributor. The c1204_32.c and c1204_32.h files contain libraries for accessing the M58BF008 Flash Memories.

Also included in this application note is an overview of the programming model for the M58BF008. This will familiarize the reader with the operation of the memory devices and provide a basis for understanding and modifying the accompanying source code.

The source code is written to be as platform independent as possible and requires minimal changes by the user in order to compile and run. The application note explains how the user should modify the source code for their individual target hardware. All of the source code is backed up by comments explaining how it is used and why it has been written as it has.

This application note does not replace the M58BF008 datasheet. It refers to the datasheet throughout and it is necessary to have a copy in order to follow some of the explanations.

The software and accompanying documentation has been fully tested on a target platform. It is small in size and can be applied to any target hardware.

THE M58BF008 PROGRAMMING MODEL

The M58BF008 is an 8Mb (256Kb x32) Flash Memory which can be electrically erased and programmed through special coded command sequences on most standard microprocessor buses. The device is broken down into 32 blocks, each 32 Kbytes in size. Each block can be erased individually, or the whole chip can be erased at once, erasing all 8Mb. An additional 32 Kbyte Overlay Block is provided to hold the boot code for the microprocessor; once booted the microprocessor can hide the Overlay Block and access Block 0.

The M58BF008 is a smart voltage device. It differs from first generation dual voltage devices which require a 12V supply to

program or erase. The M58BF008 is therefore easier to use since the hardware does not need to cater for special bus signal levels. The voltages needed to erase the device are generated by charge pumps inside the device. Three power supply pins are used to give the optimal supply voltage conditions. In normal operation the core is supplied with 5V, the I/O buffers are supplied with 3.3V and the V_{PP} Supply pin is supplied with 0V (all blcks protected) or 5V (no blocks protected). The program/erase section can be supplied with 12V through V_{PP} pin to speed up programming in the factory; this mode is not to be used as a permanent solution.

Included in the device is a Program/Erase Controller. With first generation Flash Memory devices the software had to manually program all of the words to 0000h before erasing to FFFFh using special programming sequences. The Program/Erase Controller in the M58BF008 allows a simpler programming model to be used, by taking care of all the necessary steps required to erase and program the memory. This has led to improved reliability so that in excess of 10,000 program/erase cycles are guaranteed per block on the device.

Bus Operations and Commands

Most of the functionality of the M58BF008 is available via the two standard bus operations: read and write. Read operations retrieve data or status information from the device. Write operations are interpreted by the device as commands, which modify the data stored or the behavior of the device. Only certain special sequences of write operations are recognized as commands by the M58BF008. The various commands recognized by the M58BF008 are listed in the Instructions Table of the datasheet; the main commands can be grouped as follows:

1. Read
2. Read Electronic Signature
3. Erase
4. Program
5. Program/Erase Suspend
6. Enable/Disable Overlay Block
7. Asynchronous/Synchronous bus interface toggle.

The Read command returns the M58BF008 to Read mode where it behaves as a ROM. In this state, a read operation outputs onto the data bus the data stored at the specified address of the device.

The Read Electronic Signature command places the device in a mode which allows the user to read the Manufacturer, Device Code and Version Code of the device. In the library of software drivers this mode is known as the Auto Select mode to make the M58BF008 compatible with the M29 series Flash Memories.

The Erase command is used to set all the bits to ‘1’ in every memory location in the selected block. All data previously stored in the erased block will be lost. The erase command takes longer to execute than the other commands, because an entire block is erased at once. Attempts to erase or program a block while the memory is protected (i.e. $V_{PP} = V_{IL}$) generate an error and do not modify the contents of the memory. A special command is used to erase the Overlay Block, providing additional security for the data in the Overlay Block.

The Program command is used to modify the data stored at the specified address of the device. Note that programming can only change bits from ‘1’ to ‘0’. Attempting to change a ‘0’ to a ‘1’ using the Program command will fail, though the Status Register may not indicate the failure. It may therefore be necessary to erase the block before programming to addresses within it. Programming modifies a single word at a time. Programming larger amounts of data must be done one word at a time, by issuing a Program command, waiting for the command to complete, then issuing the next Program command, and so on. A special command is used to program the Overlay Block, providing additional security for the data in the Overlay Block.

Issuing the Program/Erase Suspend command during a Program or Erase operation will temporarily place the M58BF008 in Program/Erase Suspend mode. While an Erase operation is being suspended the blocks not being erased may be read or programmed as if in the reset state of the device. While a Program operation is being suspended the rest of the device may be read. This allows the user to access information stored in the device immediately rather than waiting until the Program or Erase operation completes, typically 10µs for programming and 0.33s for erasing on the M58BF008. The Program or Erase operation is resumed when the device receives the Program/Erase Resume command.

The Overlay Block is read in the same address space as Block 0. The Overlay Block Toggle command is used to switch between reading the Overlay Block and reading Block 0. Special commands are used to program or erase the Overlay Block and are independent of the block selected for Read mode.

The Bus interface on read operations can be toggled between asynchronous and synchronous modes. However write operations will be either asynchronous or synchronous depending on the factory configuration of the device. The C code does not include this function as it would typically be performed as part of the hardware initialisation.

The Status Register

While the M58BF008 is programming or erasing, a read from the device will output the Status Register of the Program/Erase Controller. The Status Register, which can also be accessed by issuing the Read Status Register command, provides valuable information about the most recent Program or Erase command. The Status Register bits are described in the Status Register Bits Table of the M58BF008 datasheet. Their main use is to determine when programming or erasing is complete and whether it is successful or not.

Completion of the Program or Erase operation is indicated by the Program/Erase Controller Status bit (Status Register bit DQ7) becoming ‘1’. Programming or erasing errors are then indicated by one or more of the various error bits (Status Register bits DQ3, DQ4 and DQ5) being ‘1’. If a failure occurs the Status Register error bits will remain set until a Clear Status Register command is issued to the device. This should be done before performing any further operations or it will not be possible to determine whether the following operation resulted in an error or not.

A Detailed Example

The Instructions Table of the M58BF008 datasheet describes the sequences of write operations that will be recognized by the Program/Erase Controller as valid commands. For example programming 3F819465h to the address 03E2h requires the user to write the following sequence (in C):

```
*(unsigned long*)(0x0000) = 0x00000040;
*(unsigned long*)(0x03E2) = 0x3F819465;
```

The first of the two addresses (0000h) is arbitrary, so long as it is inside the Flash address space. This example assumes that address 0000h of the M58BF008 is mapped to address 0000h in the microprocessor address space. In practice it is likely that the Flash will have a base offset which needs to be added to the address.

While the device is programming the specified address, read operations will access the Status Register bits. Status Register bit DQ7 will be ‘0’ while programming is on-going and will become ‘1’ on completion. If Status Register bits DQ3 or DQ4 are set on completion then the Program command will have failed. The Status Register bits do not indicate an error when an attempt to change a ‘0’ to a ‘1’ has been made; it is recommended that the value in the memory after programming is compared to the desired value to trap this error.

WRITING C CODE FOR THE M58BF008

The low-level functions (drivers) described in this application note have been provided to simplify the process of developing application code in C for the STMicroelectronics Flash Memories (M58BF008). This enables users to concentrate on writing the high level functions required for their particular applications.

These high level functions can access the Flash Memories by calling the low level drivers, hence keeping details of special command sequences away from the users' high level code: this will result in source code both simpler and easier to maintain.

Code developed using the drivers provided can be decomposed into three layers:

1. the hardware specific bus operations
2. the low-level drivers
3. the high level functions written by the user

The implementation in C of the hardware specific read and write bus operations is required by the low-level drivers in order to communicate with the M58BF008. This implementation is hardware platform dependent as it is affected by which microprocessor the C code runs on and by where in the microprocessor's address space the memory device is located. The user will have to write the C functions appropriate to his hardware platform (see **`FlashRead()`** and **`FlashWrite()`** in the next section).

The low-level drivers take care of issuing the correct sequences of write operations for each command and of interpreting the information received from the device during programming or erasing. These drivers encode all the specific details of how to issue commands and how to interpret the Status Register bits.

The high level functions written by the user will access the memory device by calling the low-level functions. By keeping the specific details of how to access the M58BF008 away from the high level functions, the user is left with code which is simple and easier to maintain. It also makes the user's high level functions easier to apply to other STMicroelectronics Flash Memories.

When developing an application, the user is advised to proceed as follows:

– first write a simple program to test the low level drivers provided and verify that these operate as expected on the user's target hardware and software environments.
– then the user should write the high level code for his application, which will access the Flash Memories by calling the low level drivers provided.
– finally test the complete application source code thoroughly.

C LIBRARY FUNCTIONS PROVIDED

The software library provided with this application note provides the user with source code for the following functions:

`FlashReadReset()` is used to reset the device into the Read Array mode. Note that there should be no need to call this function under normal operation as all of the other software library functions leave the device in this mode.

`FlashAutoSelect()` is used to identify the Manufacturer Code, Device Code and Version Code of the device. The function uses the Read Electronic Signature mode of the device. The function is called **`FlashAutoSelect()`** to make it compatible with the M29 series Flash Memories.

`FlashBlockErase()` is used to erase a block in the device. The blocks cannot be erased when V_{PP} is invalid: attempting to do so generates an error and leaves the Flash in an indeterminate state.

`FlashChipErase()` is used to erase the entire chip. The chip cannot be erased when V_{PP} is invalid: attempting to do so generates an error and leaves the Flash in an indeterminate state.

`FlashProgram()` is used to program data arrays into the Flash. Only previously erased double words can be programmed reliably. Again, programming cannot take place when V_{PP} is invalid.

`FlashOverlayBlockEnable()` is used to enable the Overlay Block and disable Block 0 for future calls to **`FlashRead()`**.

`FlashOverlayBlockDisable()` is used to disable the Overlay Block and enable Block 0 for future calls to **`FlashRead()`**.

FlashOverlayBlockRead() is used to read a double word from the Overlay Block and then disable the Overlay Block. It is useful to retrieve single values, for multiple reads enable the Overlay Block, read the values required and disable it again.

FlashOverlayBlockErase() is used to erase the Overlay Block. The Overlay Block cannot be erased when V_{PP} is invalid: attempting to do so generates an error and leaves the Flash in an indeterminate state.

FlashOverlayProgram() is used to program data arrays into the Flash. Only previously erased words can be programmed reliably. Again, the Overlay Block cannot be programmed when V_{PP} is invalid.

The functions provided in the software library rely on the user implementing the hardware specific bus operations. This is to be done by writing two functions as follows:

– **FlashRead()** must be written to read a value from the Flash.

– **FlashWrite()** must be written to write a value to the Flash.

An example of these functions is provided in the source code.

In many instances these functions can be written as macros and therefore will not incur the function call time overhead. The two functions which perform the basic I/O to the device have been provided for users who have awkward systems. For example where the addressing system is peculiar or the data bus has D0..D15 of the device on D16..D31 of the microprocessor. They allow any user to quickly adapt the code to virtually any target system.

Throughout the functions assumptions have been made on the data types. These are:

A **char** is 8 bits (1 byte). This is not the case in all microcontrollers. Where it is not it will be necessary to mask the unused bits of the word.

An **int** is 16 bits (2 bytes). Again, like the **char**, if this is not the case it will be necessary to use a variable type which is 16 bits or longer and mask bits above 16 bits (particularly in the user's **FlashRead()** function).

A **long** is 32 bits (4 bytes). It is necessary to have arithmetic greater than 16 bits in order to address the entire device.

Two approaches to the addressing are available: the desired address in the Flash can be specified by a 32 bit linear pointer or a 32 bit offset into the device could be provided by the user. The **FlashRead()** functions in each case would declared as:

```
unsigned long FlashRead( unsigned long *Addr);
unsigned long FlashRead( unsigned long ulOff);
```

The pointer option has the advantage that it runs faster. The 32 bit offset needs to be changed to an address for each access and this involves 32 bit arithmetic. Using a 32 bit offset is, however, more portable since the resulting software can easily be changed to run on microprocessors with segmented memory spaces (such as the 8086). For maximum portability all the functions in this application note use a 32 bit unsigned long offset, rather than a pointer.

PORTING THE DRIVERS TO THE TARGET SYSTEM

Before using the software in the Target System the user needs to write **FlashRead()** and **FlashWrite()** functions appropriate to the Target Hardware. The example **FlashRead()** and **FlashWrite()** functions provided in the source code should give the user a good idea of what is required and can be used in many instances without much modification.

To test the source code in the Target System start by simply reading from the M58BF008. If it is erased then only FFFFFFFFh data should be read. Next read the Manufacturer and Device codes and check they are correct. If these functions work then it is likely that all of the functions will work but they should all be tested thoroughly.

The programmer needs to take extra care when the device is accessed during an interrupt service routine. Two situations exist which must be considered:

When the device is in Read mode interrupts can freely read from the device.

Interrupts which do not access the device may be used during all the functions.

The programmer should also take care when a Reset is applied during Program or Erase operations. The Flash will be left in an indeterminate state and data could be lost.

LIMITATIONS OF THE SOFTWARE

The software provided does not implement a full set of the M58BF008's functionality. It is left to the user to implement the Program/Erase Suspend command of the device. The Standby mode is a hardware feature of the device and cannot be controlled through software. It is left to the user to implement to Asynchronous/Synchronous Read Toggle command, this is considered a hardware feature that is outside the scope of these C code drivers.

Care should be taken in some of the **`while()`** loops. No time-outs have been implemented. Software execution may stop in one of the loops due to a hardware error. A **`/* TimeOut! */`** comment has been put at these places and the user can add a timer to them to prevent the software failing.

The software only caters for one device in the system. To add software for more devices a mechanism for selecting the devices will be required.

When an error occurs the software simply returns the error message. It is left to the user to decide what to do. Either the command can be tried again or, if necessary the device may need to be replaced.

CONCLUSION

The M58BF008 single voltage Flash Memory is an ideal product for 32 bit embedded and other computer systems, able to be easily interfaced to microprocessors and driven with simple software drivers written in the C language.

```
/**** c1204_32.h Header File for Flash Memory *********************************

   Filename:    c1204_32.h
   Description: Header file for c1204_32.c V1.02. Consult the C file for details

   Copyright (c) 2000 STMicroelectronics.

   THIS PROGRAM IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND,EITHER
   EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTY
   OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK
   AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE
   PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,
   REPAIR OR CORRECTION.
******************************************************************************/

/******************************************************************************
Commands for the various functions
******************************************************************************/
/* Read Signature values */
#define FLASH_READ_MANUFACTURER    (-3)
#define FLASH_READ_DEVICE_CODE     (-2)
#define FLASH_READ_VERSION_CODE    (-1)

/******************************************************************************
Error Conditions and return values.

See end of C file for explanations and help
******************************************************************************/
#define FLASH_SUCCESS                (-1)
#define FLASH_BLOCK_INVALID          (-5)
#define FLASH_PROGRAM_FAIL           (-6)
#define FLASH_OFFSET_OUT_OF_RANGE    (-7)
#define FLASH_WRONG_TYPE             (-8)
#define FLASH_BLOCK_FAILED_ERASE     (-9)
#define FLASH_UNPROTECTED            (-10)
#define FLASH_PROTECTED              (-11)
#define FLASH_FUNCTION_NOT_SUPPORTED (-12)
#define FLASH_VPP_INVALID            (-13)
#define FLASH_ERASE_FAIL             (-14)
#define FLASH_UNPROTECT_FAIL         (-16)
#define FLASH_PROTECT_FAIL           (-18)
#define FLASH_OVERLAY_FAIL           (-19)
#define FLASH_OVERLAY_DISABLED       (-20)
#define FLASH_OVERLAY_ENABLED        (-21)

/******************************************************************************
Function Prototypes
******************************************************************************/
extern unsigned long FlashRead( unsigned long ulOff );
extern void FlashReadReset( void );
extern int FlashAutoSelect( int iFunc );
extern int FlashBlockErase( unsigned char ucBlock );
extern int FlashChipErase( int *iResults );
extern int FlashProgram( unsigned long ulOff, size_t NumWords, void *Array );
extern int FlashOverlayBlockDisable( void );
extern int FlashOverlayBlockEnable( void );
extern unsigned long FlashOverlayBlockRead( unsigned long ulOff, int *iRetVal );
extern int FlashOverlayBlockErase( void );
extern int FlashOverlayBlockProgram( unsigned long ulOff
```

```
   , size_t NumWords, void *Array );
extern char *FlashErrorStr( int iErrNum );
```

```
/**** c1204_32.c  M58BFxxx Flash Memory Driver ********************************

   Filename:    c1204_32.c
   Description: Library routines for the M58BFxxx Flash Memory.

   Version:     1.02 Initial release.
   Date:        08/11/2000
   Author:      Tim Webster, Oxford Technical Solutions (www.ots.ndirect.co.uk)

   Copyright (c) 2000 STMicroelectronics.

   THIS PROGRAM IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER
   EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTY
   OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK
   AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE
   PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,
   REPAIR OR CORRECTION.
******************************************************************************

   Version History.

   Ver.   Date        Comments

   1.00   03/10/2000  Initial creation
   1.01   07/11/2000  Add programming failure on attempt to program a ‘0’ to a
                      ‘1’ in functions FlashProgram and FlashOverlayProgram
   1.02   08/11/2000  Add Program Verification to functions FlashProgram and
                      FlashOverlayBlockProgram
   1.03   11/12/2000  Changed BASE_ADDR to be “long*” not “int*”

******************************************************************************

   This source file provides library C code for using the M28WxxxC devices.
   The following devices are supported in the code:
      M58BF008

   The following functions are available in this library:

      FlashRead()               to read from the flash
      FlashReadReset()          to reset the flash for normal memory access
      FlashAutoSelect()         to get information about the device
      FlashBlockErase()         to erase a single block
      FlashChipErase()          to erase the whole chip
      FlashProgram()            to program a double-word or an array
      FlashOverlayBlockDisable() to enable main block reads
      FlashOverlayBlockEnable()  to enable overlay block reads
      FlashOverlayBlockRead()    to read a double-word from the overlay block
      FlashOverlayBlockErase()   to erase the entire overlay block
      FlashOverlayBlockProgram() to program a double-word/array to the overlay
                                 block
      FlashErrorStr()           to return the error string of an error

   For further information consult the Data Sheet and the Application Note.
   The Application Note gives information about how to modify this code for
   a specific application.

   The hardware specific functions which need to be modified by the user are:

      FlashWrite() for writing a double-word to the flash
```

```
      FlashRead()  for reading a double-word from the flash

   A list of the error conditions is at the end of the code.

   There are no timeouts implemented in the loops in the code. At each point
   where an infinite loop is implemented a comment /# TimeOut! #/ has been
   placed. It is up to the user to implement these to avoid the code hanging
   instead of timing out.

   The source code assumes that the compiler implements the numerical types as

   unsigned char     8 bits
   unsigned int     16 bits
   unsigned long    32 bits

   Additional changes to the code will be necessary if these are not correct.

*******************************************************************************/

#include <stdlib.h>

#include “c1204_32.h”   /* Header file with global prototypes */

#define USE_M58BF008

/*******************************************************************************
Constants
*******************************************************************************/
#define MANUFACTURER_ST (0x00000020L)  /* ST RSIG for manufacturer is 0x20 */
#define BASE_ADDR ((volatile unsigned long*)0x00000000L)
   /* BASE_ADDR is the base address of the flash, see the functions FlashRead
      and FlashWrite(). Some applications which require a more complicated
      FlashRead() or FlashWrite() may not use BASE_ADDR */

#define CMD_ADDR (0x00000000L) /* Address to write to P/EC */

#ifdef  USE_M58BF008
#define EXPECTED_DEVICE (0x000000F0L) /* Device code for the M58BF008 */
#define FLASH_SIZE (0x00040000L)      /* Total device size */
#define OVERLAY_SIZE (0x00002000L)    /* Overlay block size */

static const unsigned long BlockOffset[] =
{
   0x00000000L,  /* Start offset of block 0 */
   0x00002000L,  /* Start offset of block 1 */
   0x00004000L,  /* Start offset of block 2 */
   0x00006000L,  /* Start offset of block 3 */
   0x00008000L,  /* Start offset of block 4 */
   0x0000A000L,  /* Start offset of block 5 */
   0x0000C000L,  /* Start offset of block 6 */
   0x0000E000L,  /* Start offset of block 7 */
   0x00010000L,  /* Start offset of block 8 */
   0x00012000L,  /* Start offset of block 9 */
   0x00014000L,  /* Start offset of block 10 */
   0x00016000L,  /* Start offset of block 11 */
   0x00018000L,  /* Start offset of block 12 */
   0x0001A000L,  /* Start offset of block 13 */
   0x0001C000L,  /* Start offset of block 14 */
   0x0001E000L,  /* Start offset of block 15 */
```

```
   0x00020000L,  /* Start offset of block 16 */
   0x00022000L,  /* Start offset of block 17 */
   0x00024000L,  /* Start offset of block 18 */
   0x00026000L,  /* Start offset of block 19 */
   0x00028000L,  /* Start offset of block 20 */
   0x0002A000L,  /* Start offset of block 21 */
   0x0002C000L,  /* Start offset of block 22 */
   0x0002E000L,  /* Start offset of block 23 */
   0x00030000L,  /* Start offset of block 24 */
   0x00032000L,  /* Start offset of block 25 */
   0x00034000L,  /* Start offset of block 26 */
   0x00036000L,  /* Start offset of block 27 */
   0x00038000L,  /* Start offset of block 28 */
   0x0003A000L,  /* Start offset of block 29 */
   0x0003C000L,  /* Start offset of block 30 */
   0x0003E000L   /* Start offset of block 31 */
};
#endif /* USE_M58BF008 */

#define NUM_BLOCKS (sizeof(BlockOffset)/sizeof(BlockOffset[0]))

/*****************************************************************************
Static Prototypes

The following function is only needed in this module.
*****************************************************************************/
static unsigned long FlashWrite( unsigned long ulOff, unsigned long uVal );

/*****************************************************************************
Function:    unsigned int FlashWrite( unsigned long ulOff, unsigned int uVal )
Arguments:   ulOff is double-word offset in the flash to write to.
   uVal is the value to be written
Returns:     uVal
Description: This function is used to write a double-word to the flash. On many
   microprocessor systems a macro can be used instead, increasing the speed of
   the flash routines. For example:

#define FlashWrite( ulOff, uVal ) ( BASE_ADDR[ulOff] = (unsigned int) uVal )

   A function is used here instead to allow the user to expand it if necessary.
   The function is made to return uVal so that it is compatible with the macro.

Pseudo Code:
   Step 1: Write uVal to the double-word offset in the flash
   Step 2: return uVal
*****************************************************************************/
static unsigned long FlashWrite( unsigned long ulOff, unsigned long uVal )
{
   /* Step1, 2: Write uVal to the double-word offset in flash and return it */
   return BASE_ADDR[ulOff] = uVal;
}

/*****************************************************************************
Function:    unsigned int FlashRead( unsigned long ulOff )
Arguments:   ulOff is the double-word offset into the flash to read from.
Returns:     The unsigned int at the double-word offset
Description: This function is used to read a double-word from the flash. On many
   microprocessor systems a macro can be used instead, increasing the speed of
   the flash routines. For example:
```

```
#define FlashRead( ulOff ) ( BASE_ADDR[ulOff] )

   A function is used here instead to allow the user to expand it if necessary.

Pseudo Code:
   Step 1: Return the value at double-word offset ulOff
*******************************************************************************/
unsigned long FlashRead( unsigned long ulOff )
{
   /* Step 1 Return the value at double-word offset ulOff */
   return BASE_ADDR[ulOff];
}

/*******************************************************************************
Function:      void FlashReadReset( void )
Arguments:     none
Return Value:  none
Description:   This function places the flash in the Read Array mode described
   in the Data Sheet. In this mode the flash can be read as normal memory.

   All of the other functions leave the flash in the Read Array mode so this is
   not strictly necessary. It is provided for completeness.

Pseudo Code:
   Step 1: write command sequence (see Instructions Table of the Data Sheet)
*******************************************************************************/
void FlashReadReset( void )
{
   /* Step 1: write command sequence */
   FlashWrite( CMD_ADDR, 0xFFFFFFFFL );
}

/*******************************************************************************
Function:      int FlashAutoSelect( int iFunc )
Arguments:     iFunc should be set to one of the Read Signature values. The
   header file defines the values for reading the Signature.
Return Value:  When iFunc is FLASH_READ_MANUFACTURER (-3) the function returns
   the manufacturer's code. The Manufacturer code for ST is 00000020h.
   When iFunc is FLASH_READ_DEVICE_CODE (-2) the function returns the Device
   Code.  The device codes for the parts are:

      M58BF008   0x000000F0

   If iFunc is FLASH_READ_VERSION_CODE (-1) a value from 0 to 7 is reported to
   indicate the version of the flash.

   When iFunc is invalid the function returns FLASH_FUNCTION_NOT_SUPPORTED (-12)
Description:   This function can be used to read the electronic signature of the
   device or the manufacturer code.

Pseudo Code:
   Step 1:  Send the Auto Select Instruction to the device or RSIG instruction
   Step 2:  Read the required function from the device
   Step 3:  Return the device to Read Array mode
*******************************************************************************/
int FlashAutoSelect( int iFunc )
{
   int iRetVal;   /* Holds the return value */
```

```
   /* Step 1: Send the Read Electronic Signature instruction */
   FlashWrite( CMD_ADDR, 0x00000090L );

   /* Step 2: Read the required function */

   /* Note Only A0 and A1 are valid read addresses A2-17 Don't care */

   if( iFunc == FLASH_READ_MANUFACTURER )
      iRetVal = FlashRead( 0x00000000L ); /* A0 = A1 = 0 */

   else if( iFunc == FLASH_READ_DEVICE_CODE )
      iRetVal = FlashRead( 0x00000001L ); /* A0 = 1, A1 = 0 */

   else if( iFunc == FLASH_READ_VERSION_CODE )
      iRetVal = FlashRead( 0x00000002L ); /* A0 = 0, A1 = 1 */

   else
      iRetVal = FLASH_FUNCTION_NOT_SUPPORTED;

   /* Step 3: Return to Read Array mode */
   FlashWrite( CMD_ADDR, 0x000000FFL );

   return iRetVal;
}

/*****************************************************************************
Function:     int FlashBlockErase( unsigned char ucBlock )
Arguments:    ucBlock is the number of the Block to be erased.
Return Value: The function returns the following conditions:
   FLASH_SUCCESS              (-1)
   FLASH_WRONG_TYPE           (-8)
   FLASH_BLOCK_INVALID        (-5)
   FLASH_VPP_INVALID          (-13)
   FLASH_BLOCK_FAILED_ERASE   (-9)
Description:  This function can be used to erase the Block specified in ucBlock.
   The function checks that the block is valid before issuing the erase
   command. Once the erase has completed the function checks the Status
   Register for errors. Any errors are returned, otherwise FLASH_SUCCESS
   is returned.

Pseudo Code:
   Step 1:  Check for correct flash type
   Step 2:  Check that the block is valid
   Step 3:  Issue Erase Command
   Step 4:  Wait until Program/Erase Controller is ready
   Step 5:  Check for any errors
   Step 6:  Return to Read Array mode
   Step 7:  Return error condition
*****************************************************************************/
int FlashBlockErase( unsigned char ucBlock )
{
   int iRetVal = FLASH_SUCCESS; /* Holds return value: optimistic initially! */
   unsigned long ulStatus;      /* Holds the Status Register reads */

   /* Step 1: Check for correct flash type */
   if( !(FlashAutoSelect( FLASH_READ_MANUFACTURER ) == MANUFACTURER_ST)
   ||  !(FlashAutoSelect( FLASH_READ_DEVICE_CODE ) == EXPECTED_DEVICE ) )
      return FLASH_WRONG_TYPE;
```

```
   /* Step 2: Check that the block is valid */
   if( ucBlock >= NUM_BLOCKS )
      return FLASH_BLOCK_INVALID;

   /* Step 3: Issue Erase Command */
   FlashWrite( CMD_ADDR, 0x00000050L ); /* Clear Status Register */

   /* NOTE ! CSR also clears b1 BPS as well as b3,4 and 5 */

   FlashWrite( CMD_ADDR, 0x00000020L );             /* 1st cycle */
   FlashWrite( BlockOffset[ucBlock], 0x000000D0L ); /* 2nd cycle */

   /* Step 4:    Wait until Program/Erase Controller is ready */
   /* TimeOut! */
   do
      ulStatus = FlashRead(CMD_ADDR);
   while( (ulStatus&0x00000080L) == 0x00000000L );

   /* Step 5: Check for any errors */
   if( ulStatus&0x00000008L )
      iRetVal = FLASH_VPP_INVALID;
   else if( ulStatus&0x00000020L )
      iRetVal = FLASH_BLOCK_FAILED_ERASE;

   /* Step 6:  Return to Read Array mode */
   FlashWrite( CMD_ADDR, 0x00000050L );   /* Clear Status Register */

   /* NOTE ! CSR also clears b1 BPS as well as b3,4 and 5 */

   FlashWrite( CMD_ADDR, 0x000000FFL );   /* Read Array Command */

   /* Step 7: Return error condition */
   return iRetVal;
}

/*****************************************************************************
Function:     int FlashChipErase( int *iResults )
Arguments:    iResults is a pointer to an array where the results will be
   stored. If iResults == NULL then no results are stored.
Return Value: The function returns the following conditions:
   FLASH_SUCCESS      (-1)
   FLASH_WRONG_TYPE   (-8)
   FLASH_ERASE_FAIL   (-14)
   If FLASH_SUCCESS is returned then Results is left unchanged.
   If FLASH_ERASE_FAIL is returned then Results will be filled with the error
   conditions for each block. The possible error conditions are:
   FLASH_SUCCESS      (-1)
   FLASH_VPP_INVALID  (-13)
   FLASH_ERASE_FAIL   (-14)
Description: The function can be used to erase the whole flash chip. Each Block
   is erased in turn. The function only returns when all of the Blocks have
   been erased or have generated an error, except if the FLASH_VPP_INVALID is
   encountered, in which case the function aborts and reports all remaining
   blocks as having FLASH_VPP_INVALID. If Vpp is invalid for one block then it
   follows that it will be invalid for subsequent blocks (battery failure?).

Pseudo Code:
   Step 1: Check for correct flash type
```

```
   Step 2: For each block
   Step 3: Send Block Erase Command
   Step 4: Register the errors in the array
   Step 5: If FLASH_VPP_INVALID returned fill rest of results array & abort
   Step 6: Return error condition
*****************************************************************************/
int FlashChipErase( int *iResults )
{
   unsigned char ucCurBlock;    /* Used to track the current block in a range */
   int iRetVal = FLASH_SUCCESS; /* Return value: Initially optimistic */
   int iError;                  /* Holds the latest error */

   /* Step 1: Check for correct flash type */
   if( !(FlashAutoSelect( FLASH_READ_MANUFACTURER ) == MANUFACTURER_ST)
   ||  !(FlashAutoSelect( FLASH_READ_DEVICE_CODE ) == EXPECTED_DEVICE ) )
      return FLASH_WRONG_TYPE;

   /* Step 2: For each block */
   for( ucCurBlock = 0; ucCurBlock < NUM_BLOCKS; ucCurBlock++ )
   {
      /* Step 3: Send Block Erase Command */
      iError = FlashBlockErase( ucCurBlock );
      if( iError != FLASH_SUCCESS )
         iRetVal = FLASH_ERASE_FAIL;

      /* Step 4: Register the errors in the array */
      if( iResults != NULL )
         iResults[ucCurBlock] = iError;

      /* Step 5: If FLASH_VPP_INVALID returned fill rest of results array
                 & abort */
      if( iError == FLASH_VPP_INVALID )
      {
         if( iResults != NULL )
            while( ++ucCurBlock < NUM_BLOCKS ) /* on remaining blocks */
               iResults[ucCurBlock] = iError;  /* fill in Vpp error */
         /* the for() loop will now terminate since ucCurBlock == NUM_BLOCKS */
      }
   }

   /* Step 6: Return error condition */
   return iRetVal;
}

/*****************************************************************************
Function:     int FlashProgram( unsigned long ulOff, size_t NumWords,
   void *Array )
Arguments:    ulOff is the double-word offset into the flash to be programmed
   NumWords holds the number of double-words in the array.
   Array is a pointer to the array to be programmed.
Return Value:  On success the function returns FLASH_SUCCESS (-1).
   If Vpp is invalid then the function returns FLASH_VPP_INVALID (-13)
   On failure the function returns FLASH_PROGRAM_FAIL (-6).
   If the address range to be programmed exceeds the address range of the Flash
   Device the function returns FLASH_ADDRESS_OUT_OF_RANGE (-7) and nothing is
   programmed.
   If the wrong type of flash is detected then FLASH_WRONG_TYPE (-8) is
   returned and nothing is programmed.
Description: This function is used to program an array into the flash. It does
```

```
   not erase the flash first and will fail if the block(s) are not erased first.

Pseudo Code:
   Step 1: Check for correct flash type
   Step 2: Check the offset range is valid
   Step 3: While there is more to be programmed
   Step 4: Check for changes from '0' to '1'
   Step 5: Program the next double-word
   Step 6: Wait until the Program/Erase Controller is ready
   Step 7: Check for any errors
   Step 8: Verify program operation
   Step 9: Update pointers
   Step 10:Clear status register and return to read array mode
   Step 11:Return the error condition
*************************************************************************/
int FlashProgram( unsigned long ulOff, size_t NumWords, void *Array )
{
   unsigned long *ulArrayPointer; /* Use an unsigned long to access the array */
   unsigned long ulLastOff;       /* Holds the last offset to be programmed */
   unsigned long ulStatus;        /* Holds the Status Register reads */
   int iRetVal = FLASH_SUCCESS;   /* Return Value: Initially optimistic */

   /* Step 1: Check that the flash is of the correct type */
   if( !(FlashAutoSelect( FLASH_READ_MANUFACTURER ) == MANUFACTURER_ST)
   ||  !(FlashAutoSelect( FLASH_READ_DEVICE_CODE ) == EXPECTED_DEVICE ) )
      return FLASH_WRONG_TYPE;

   /* Step 2: Check the offset range is valid */
   ulLastOff = ulOff + NumWords - 1;
   if( ulLastOff >= FLASH_SIZE )
      return FLASH_OFFSET_OUT_OF_RANGE;

   /* Step 3: While there is more to be programmed */
   ulArrayPointer = (unsigned long *)Array;
   while( ulOff <= ulLastOff && iRetVal == FLASH_SUCCESS )
   {
      /* Step 4: Check for changes from '0' to '1' */
      if( ~FlashRead( ulOff ) & *ulArrayPointer )
         return FLASH_PROGRAM_FAIL;

      /* Step 5: Program the next double-word */
      FlashWrite( CMD_ADDR, 0x00000050L );    /* Clear Status Register */
      FlashWrite( CMD_ADDR, 0x00000040L );    /* Program Set-up */
      FlashWrite( ulOff, *ulArrayPointer );   /* Program value */

      /* Step 6: Wait until Program/Erase Controller is ready */
      /* TimeOut! */
      do
         ulStatus = FlashRead(CMD_ADDR);
      while( (ulStatus&0x00000080L) == 0x00000000L );

      /* Step 7: Check for any errors */
      if( ulStatus&0x00000008L )
         iRetVal = FLASH_VPP_INVALID;
      else if( ulStatus&0x00000010L )
         iRetVal = FLASH_PROGRAM_FAIL;

      FlashWrite( CMD_ADDR, 0x000000FFL ); /* Read Array Command */
```

```
      /* Step 8: Verify program operation */
      if ( FlashRead( ulOff ) != *ulArrayPointer )
      {
         iRetVal=FLASH_PROGRAM_FAIL;
         break;
      }

      /* Step 9: Update pointers */
      ulOff++;           /* next double-word offset */
      ulArrayPointer++;  /* next double-word in array */
   }

   /* Step 10: Clear status register and return to read array mode */
   FlashWrite( CMD_ADDR, 0x00000050L ); /* Clear Status Register */
   FlashWrite( CMD_ADDR, 0x000000FFL ); /* Read Array Command */

   /* Step 11: return the error condition */
   return iRetVal;
}

/*****************************************************************************
Function:     int FlashOverlayBlockDisable( void );
Arguments:    none
Return Value: On success this function returns FLASH_SUCCESS (-1) or if the
   flash overlay block is already disabled it returns FLASH_OVERLAY_DISABLED.
   If the toggle command was unsuccessful and the flash overlay block will not
   disable the function returns FLASH_OVERLAY_FAIL.
Description:  This function gets the flash into normal block read mode.

Pseudo Code:
   Step 1: Check for correct flash type
   Step 2: Read Status Register
   Step 3: If the Overlay block is already disabled do nothing and inform the
           user of that fact
   Step 4: Disable the Overlay Block
   Step 5: Read Status Register
   Step 6: Test to see that the Disable Overlay toggle was successful
   Step 7: Return success
*****************************************************************************/
int FlashOverlayBlockDisable( void )
{
   unsigned long ulStatus;   /* Used to store status register value */

   /* Step 1: Check that the flash is of the correct type */
   if( !(FlashAutoSelect( FLASH_READ_MANUFACTURER ) == MANUFACTURER_ST)
   ||  !(FlashAutoSelect( FLASH_READ_DEVICE_CODE ) == EXPECTED_DEVICE ) )
      return FLASH_WRONG_TYPE;

   /* Step 2: Read Status Register */
   FlashWrite(CMD_ADDR,0x00000070L);
   ulStatus = FlashRead(CMD_ADDR);

   /* Step 3: Test to see is its already disabled */
   if (!(ulStatus & 0x00000001L))
   {
      FlashWrite( CMD_ADDR, 0x000000FFL );
      return FLASH_OVERLAY_DISABLED;
   }
```

```
   /* Step 4: Disable Overlay Block */
   FlashWrite(CMD_ADDR,0x00000006L);

   /* Step 5: Read Status Register */
   FlashWrite(CMD_ADDR,0x00000070L);
   ulStatus = FlashRead(CMD_ADDR);

   FlashWrite( CMD_ADDR, 0x000000FFL ); /* Read Array Command */

   /* Step 6: Test to see if the operation failed */
   if (ulStatus & 0x00000001L) return FLASH_OVERLAY_FAIL;

   /* Step 7: Return Success */
   return FLASH_SUCCESS;
}
/*****************************************************************************
Function:     int FlashOverlayBlockEnable( void );
Arguments:    none
Return Value: On success the function returns FLASH_SUCCESS (-1) or if the flash
   overlay block is already enabled it returns FLASH_OVERLAY_ENABLED. If the
   toggle command is unsuccessful and the flash overlay block will not enable
   the function returns FLASH_OVERLAY_FAIL.
Description:  This function gets the flash into overlay block read mode.

Pseudo Code:
   Step 1: Ensure that we have the correct flash
   Step 2: Read Status Register
   Step 3: If the Overlay block is already enabled do nothing and inform the
   user of that fact
   Step 4: Enable the Overlay Block
   Step 5: Read Status Register
   Step 6: Test to see that the Enable Overlay toggle was successful
   Step 7: Return success
*****************************************************************************/
int FlashOverlayBlockEnable( void )
{
   unsigned long ulStatus;   /* Used to store status register value */

   /* Step 1: Check that the flash is of the correct type */
   if( !(FlashAutoSelect( FLASH_READ_MANUFACTURER ) == MANUFACTURER_ST)
   ||  !(FlashAutoSelect( FLASH_READ_DEVICE_CODE ) == EXPECTED_DEVICE ) )
      return FLASH_WRONG_TYPE;

   /* Step 2: Read Status Register */
   FlashWrite(CMD_ADDR,0x00000070L);
   ulStatus = FlashRead(CMD_ADDR);

   /* Step 3: Test to see is its already enabled */
   if (ulStatus & 0x00000001L)
   {
      FlashWrite( CMD_ADDR, 0x000000FFL );
      return FLASH_OVERLAY_ENABLED;
   }

   /* Step 4: Enable Overlay Block */
   FlashWrite(CMD_ADDR,0x00000006L);

   /* Step 5: Read Status Register */
   FlashWrite(CMD_ADDR,0x00000070L);
```

```
   ulStatus = FlashRead(CMD_ADDR);

   FlashWrite( CMD_ADDR, 0x000000FFL ); /* Read Array Command */
   /* Step 6: Test to see if the operation failed */
   if (!(ulStatus & 0x00000001L)) return FLASH_OVERLAY_FAIL;

   /* Step 7: Return Success */
   return FLASH_SUCCESS;
}

/*****************************************************************************
Function:    unsigned long FlashOverlayBlockRead( unsigned long ulOff,
   int *iRetVal);
Arguments:   ulOff stores the address offset to read. iRetVal points to the
   return value integer.
Return Value: (iRetVal) A successful read will return FLASH_SUCCESS (-1). Error
   conditions possible are:
   FLASH_OFFSET_OUT_OF_RANGE (-7) when the address passed to this function is
      outside the address space of the overlay block.
   FLASH_WRONG_TYPE (-8) if the part Autoselected was not as expected.
Description:  This function automatically enables the overlay block then reads
   the value at address offset ulOff, then resets the flash into normal block
   read mode.
Pseudo Code:
   Step 1: Ensure that the address is a valid overlay block address
   Step 2: Enable the overlay block so that it can be read
   Step 3: Read the value at the address offset ulOff
   Step 4: Disable the overlay block return value read
*****************************************************************************/
unsigned long FlashOverlayBlockRead( unsigned long ulOff, int *iRetVal )
{
   unsigned long ulTmp;   /* Used to store read value from overlay block */

   /* Step 1: Is the address valid ? */
   if (ulOff >= OVERLAY_SIZE) *iRetVal=FLASH_OFFSET_OUT_OF_RANGE;
   else
   {
      /* Step 2: If the overlay block read enable was not successful
         return error*/
      if (((*iRetVal=FlashOverlayBlockEnable()) == FLASH_SUCCESS) ||
         ( *iRetVal == FLASH_OVERLAY_ENABLED))
         {
            /* Step 3: Read the value from the overlay block */
            ulTmp = FlashRead( ulOff );
         }
      else return 0;

      /* Step 4: Return the value read from the overlay block */
      *iRetVal=FlashOverlayBlockDisable();
      return ulTmp;
   }
   return 0;
}

/*****************************************************************************
Function:     int FlashOverlayBlockErase( void );
Arguments:    none
Return Value: This function returns FLASH_SUCCESS (-1) if the overlay block has
   been successfully erased or FLASH_BLOCK_FAILED_ERASE (-9) if the operation
```

```
   failed. Failure return codes are:
   FLASH_WRONG_TYPE   (-8)
   FLASH_VPP_INVALID  (-13)
Description:  This function erases the overlay block.
Pseudo Code:
   Step 1: Check for correct flash type
   Step 2: Issue the overlay block erase command
   Step 3: Wait for the P/E C to complete
   Step 4: Check for errors
   Step 5: Reset the device in read array mode
   Step 6: Return iRetVal
*****************************************************************************/
int FlashOverlayBlockErase( void )
{
   int iRetVal = FLASH_SUCCESS; /* Holds return value: optimistic initially! */
   unsigned long ulStatus;        /* Holds the Status Register reads */

   /* Step 1: Check for correct flash type */
   if( !(FlashAutoSelect( FLASH_READ_MANUFACTURER ) == MANUFACTURER_ST)
   ||  !(FlashAutoSelect( FLASH_READ_DEVICE_CODE ) == EXPECTED_DEVICE ) )
      return FLASH_WRONG_TYPE;

   /* Step 2: Issue Erase Command */
   FlashWrite( CMD_ADDR, 0x00000050L ); /* Clear Status Register */

   FlashWrite( CMD_ADDR, 0x00000002L );       /* 1st cycle */
   FlashWrite( BlockOffset[0], 0x0000000DL ); /* 2nd cycle */

   /* Step 3:    Wait until Program/Erase Controller is ready */
   /* TimeOut! */
   do
      ulStatus = FlashRead(CMD_ADDR);
   while( (ulStatus&0x00000080L) == 0x00000000L );

   /* Step 4: Check for any errors */
   if( ulStatus&0x00000008L )
      iRetVal = FLASH_VPP_INVALID;
   else if( ulStatus&0x00000020L )
      iRetVal = FLASH_BLOCK_FAILED_ERASE;

   /* Step 5:  Return to Read Array mode */
   FlashWrite( CMD_ADDR, 0x00000050L );   /* Clear Status Register */

   FlashWrite( CMD_ADDR, 0x000000FFL );   /* Read Array Command */

   /* Step 6: Return error condition */
   return iRetVal;
}

/*****************************************************************************
Function:     int FlashOverlayBlockProgram( unsigned long ulOff,
   size_t NumWords, void *Array )
Arguments:    ulOff is the double-word offset into the flash to be programmed
   NumWords holds the number of double-words in the array.
   Array is a pointer to the array to be programmed.
Return Value:  On success the function returns FLASH_SUCCESS (-1).
   If VPP is invalid then the function returns FLASH_VPP_INVALID (-13)
   On failure the function returns FLASH_PROGRAM_FAIL (-6).
   If the address range to be programmed exceeds the address range of the Flash
```

```
   Device the function returns FLASH_ADDRESS_OUT_OF_RANGE (-7) and nothing is
   programmed.
   If the wrong type of flash is detected then FLASH_WRONG_TYPE (-8) is
   returned and nothing is programmed.
Description: This function is used to program an array into the flash. It does
   not erase the flash first and will fail if the block(s) are not erased first.

Pseudo Code:
   Step 1: Check for correct flash type
   Step 2: Check the offset range is valid
   Step 3: Enable the overlay block
   Step 4: While there is more to be programmed
   Step 5: Check for changes from '0' to '1'
   Step 6: Program the next double-word
   Step 7: Wait until the Program/Erase Controller is ready
   Step 8: Check for any errors
   Step 9: Verify program operation
   Step 10:Update pointers
   Step 11:Disable the overlay block
   Step 12:Clear status register and return to read array mode
   Step 13:Return the error condition
*****************************************************************************/
int FlashOverlayBlockProgram( unsigned long ulOff, size_t NumWords,
   void *Array )
{
   unsigned long *ulArrayPointer; /* Use an unsigned int to access the array */
   unsigned long ulLastOff;       /* Holds the last offset to be programmed */
   unsigned long ulStatus;        /* Holds the Status Register reads */
   int iRetVal = FLASH_SUCCESS;   /* Return Value: Initially optimistic */

   /* Step 1: Check that the flash is of the correct type */
   if( !(FlashAutoSelect( FLASH_READ_MANUFACTURER ) == MANUFACTURER_ST)
   ||  !(FlashAutoSelect( FLASH_READ_DEVICE_CODE ) == EXPECTED_DEVICE ) )
      return FLASH_WRONG_TYPE;

   /* Step 2: Check the offset range is valid */
   ulLastOff = ulOff + NumWords - 1;
   if( ulLastOff >= OVERLAY_SIZE )
      return FLASH_OFFSET_OUT_OF_RANGE;

   /* Step 3: Enable the Overlay Block */
   if ((FlashOverlayBlockEnable()) == FLASH_OVERLAY_FAIL)
      return FLASH_OVERLAY_FAIL;

   /* Step 4: While there is more to be programmed */
   ulArrayPointer = (unsigned long *)Array;
   while( ulOff <= ulLastOff && iRetVal == FLASH_SUCCESS )
   {
      /* Step 5: Check for changes from '0' to '1' */
      if( ~FlashRead( ulOff ) & *ulArrayPointer )
      {
         FlashOverlayBlockDisable();
         return FLASH_PROGRAM_FAIL;
      }

      /* Step 6: Program the next double-word */
      FlashWrite( CMD_ADDR, 0x00000050L );        /* Clear Status Register */
      FlashWrite( CMD_ADDR, 0x00000004L );        /* Program Set-up */
      FlashWrite( ulOff, *ulArrayPointer );       /* Program value */
```

```
      /* Step 7: Wait until Program/Erase Controller is ready */
      /* TimeOut! */
      do
         ulStatus = FlashRead(CMD_ADDR);
      while( (ulStatus&0x00000080L) == 0x00000000L );

      /* Step 8: Check for any errors */
      if( ulStatus&0x00000008L )
         iRetVal = FLASH_VPP_INVALID;
      else if( ulStatus&0x00000010L )
         iRetVal = FLASH_PROGRAM_FAIL;

      FlashWrite( CMD_ADDR, 0x000000FFL ); /* Read Array Command */

      /* Step 9: Verify program operation */
      if ( FlashRead( ulOff ) != *ulArrayPointer )
      {
         iRetVal=FLASH_PROGRAM_FAIL;
         break;
      }

      /* Step 10: Update pointers */
      ulOff++;           /* next double-word offset */
      ulArrayPointer++;  /* next double-word in array */
   }

   /* Step 11: Disable the Overlay Block */
   if ((FlashOverlayBlockDisable()) == FLASH_OVERLAY_FAIL)
      iRetVal = FLASH_OVERLAY_FAIL;

   /* Step 12: Clear status register and return to read array mode */
   FlashWrite( CMD_ADDR, 0x00000050L ); /* Clear Status Register */
   FlashWrite( CMD_ADDR, 0x000000FFL ); /* Read Array Command */

   /* Step 13: return the error condition */
   return iRetVal;
}

/*****************************************************************************
Function:     char *FlashErrorStr( int iErrNum );
Arguments:    iErrNum is the error number returned from another Flash Routine
Return Value: A pointer to a string with the error message
Description:  This function is used to generate a text string describing the
   error from the flash. Call with the return value from another flash routine.

Pseudo Code:
   Step 1: Check the error message range.
   Step 2: Return the correct string.
*****************************************************************************/
char *FlashErrorStr( int iErrNum )
{
   static char *str[] = { “Flash Success”,
                          “Flash Poll Failure”,
                          “Flash Too Many Blocks”,
                          “MPU is too slow to erase all the blocks”,
                          “Flash Block selected is invalid”,
                          “Flash Program Failure”,
                          “Flash Address Offset Out Of Range”,
```

```
                        "Flash is Wrong Type",
                        "Flash Block Failed Erase",
                        "Flash is Unprotected",
                        "Flash is Protected",
                        "Flash function not supported",
                        "Flash Vpp Invalid",
                        "Flash Erase Fail",
                        "Flash Toggle Flow Chart Failure",
                        "Flash Unprotect failed",
                        "Flash Bank Invalid",
                        "Flash Protect Failed",
                        "Flash Overlay Command Failed",
                        "Flash Overlay Disabled already",
                        "Flash Overlay Enabled already"
                       };

   /* Step 1: Check the error message range */
   iErrNum = -iErrNum - 1;   /* All errors are negative: make +ve & adjust */

   if( iErrNum < 0 || iErrNum >= sizeof(str)/sizeof(str[0])) /* Check range */
      return "Unknown Error\n";

   /* Step 2: Return the correct string */
   else
      return str[iErrNum];
}

/*****************************************************************************
List of Errors and Return values, Explanations and Help.
*****************************************************************************

Return Name:  FLASH_SUCCESS
Return Value: -1
Description:  This value indicates that the flash command has executed
   correctly.
*****************************************************************************

Error Name:   FLASH_POLL_FAIL
Notes:        Applies to M29 series FLASH only. This error condition should not
   occur when using this library.
Return Value: -2
Description:  The Program/Erase Controller algorithm has not managed to complete
   the command operation successfully. This may be because the device is damaged
Solution:     Try the command again. If it fails a second time then it is
   likely that the device will need to be replaced.
*****************************************************************************

Error Name:   FLASH_TOO_MANY_BLOCKS
Notes:        Applies to M29 series FLASH only. This error condition should not
   occur when using this library.
Return Value: -3
Description:  The user has chosen to erase more blocks than the device has.
   This may be because the array of blocks to erase contains the same block
   more than once.
Solutions:    Check that the program is trying to erase valid blocks. The device
   will only have NUM_BLOCKS blocks (defined at the top of the file). Also check
   that the same block has not been added twice or more to the array.
*****************************************************************************
```

```
Error Name:   FLASH_MPU_TOO_SLOW
Notes:        Applies to M29 series FLASH only. This error condition should not
   occur when using this library.
Return Value: -4
Description:  The MPU has not managed to write all of the selected blocks to the
   device before the timeout period expired. See BLOCK ERASE COMMAND
   section of the Data Sheet for details.
Solutions:    If this occurs occasionally then it may be because an interrupt is
   occuring between writing the blocks to be erased. Search for "DSI!" in
   the code and disable interrupts during the time critical sections.
   If this error condition always occurs then it may be time for a faster
   microprocessor, a better optimising C compiler or, worse still, learn
   assembly. The immediate solution is to only erase one block at a time.
   Disable the test (by #define'ing out the code) and always call the function
   with one block at a time.
*******************************************************************************

Error Name:   FLASH_BLOCK_INVALID
Return Value: -5
Description:  A request for an invalid block has been made. Valid blocks number
   from 0 to NUM_BLOCKS-1.
Solution:     Check that the block is in the valid range.
*******************************************************************************

Error Name:   FLASH_PROGRAM_FAIL
Return Value: -6
Description:  The programmed value has not been programmed correctly.
Solutions:    Make sure that the block containing the value was erased before
   programming. Try erasing the block and re-programming the value. If it fails
   again then the device may have blocks protected. It is also possible for the
   block to be locked but is unlikely if /WP pin is kept high.
   If the program fail still results it is likely that the flash is suspect.
*******************************************************************************

Error Name:   FLASH_OFFSET_OUT_OF_RANGE
Return Value: -7
Description:  The address offset given is out of the range of the device.
Solution:     Check the address offset is in the valid range.
*******************************************************************************

Error Name:   FLASH_WRONG_TYPE
Return Value: -8
Description:  The source code has been used to access the wrong type of flash.
Solutions:    Use a different flash chip with the target hardware or contact
   STMicroelectronics for a different source code library.
*******************************************************************************

Error Name:   FLASH_BLOCK_FAILED_ERASE
Return Value: -9
Description:  The previous erase to this block has not managed to successfully
   erase the block.
Solution:     The block may be protected/locked or the flash is suspect.
*******************************************************************************

Return Name:  FLASH_UNPROTECTED
Return Value: -10
Description:  The user has requested to unprotect a block that is already
   unprotected. This is just a warning to the user that their operation
   did not make any changes and was not necessary.
```

```
*******************************************************************************

Error Name:  FLASH_PROTECTED
Return Value: -11
Description:  The user has attempted to erase, program or protect a block of
   the flash that is protected. The operation failed because the block was
   protected.
Solutions:    Choose another (unprotected) block for erasing or programming.
   Alternatively change the block protection status of the current block.
   (see Datasheet for more details). In the case of the user protecting a block
   that is already protected, this warning notifies the user that the command
   had no effect.
*******************************************************************************

Return Name:  FLASH_FUNCTION_NOT_SUPPORTED
Return Value: -12
Description:  The user has attempted to make use of functionality not
   available on this flash device (and thus not provided by the software
   drivers). This is simply a warning to the user.
*******************************************************************************

Error Name:   FLASH_VPP_INVALID
Notes:        Applies to M28 series Flash only.
Return Value: -13
Description:  A Program or a Block Erase has been attempted with the Vpp supply
   voltage outside the allowed ranges. This command had no effect since an
   invalid Vpp has the effect of protecting the whole of the flash device.
Solution:     The (hardware) configuration of Vpp will need to be modified to
   make programming or erasing the device possible.
*******************************************************************************

Error Name:   FLASH_ERASE_FAIL
Return Value: -14
Description:  This indicates that the previous erasure of the whole device has
   failed.
Solution:     Investigate this failure further by attempting to erase each block
   individually. If erasing a single block still causes failure, then the Flash
   sadly needs replacing.
*******************************************************************************

Error Name:   Flash_TOGGLE_FAIL
Return Value: -15
Notes:        Applies to M29 series Flash only. This error condition should not
   occur when using this library.
Description:  The Program/Erase Controller algorithm has not managed to complete
   the command operation successfully. This may be because the device is
   damaged.
Solution:     Try the command again. If it fails a second time then it is
   likely that the device will need to be replaced.
*******************************************************************************

Error Name:   Flash_UNPROTECT_FAIL
Return Value: -16
Description:  This error return value indicates that a block unprotect command
   was unsuccessful.
Solution:     Try the command again. If it fails a second time then it is
   likely that the device is either locked or will need to be replaced.
   (Part is unlocked but protected on power-up with /WP pin at V_high).
*******************************************************************************
```

```
Error Name:   Flash_BANK_INVALID
Return Value: -17
Notes:        This applies to M59DRxxx series Flash only.
Description:  This error return value indicates that a bank does not exist.
*****************************************************************************

Error Name:   Flash_PROTECT_FAIL
Return Value: -18
Description:  This error return value indicates that a block protect command
   was unsuccessful.
Solution:     Try the command again. If it fails a second time then it is
   likely that the device is either locked or will need to be replaced.
   (Part is unlocked but protected on power-up with /WP pin at V_high).
*****************************************************************************

Error Name:   Flash_OVERLAY_FAIL
Return Value: -19
Description:  This error indicates that any Overlay Block associated function
   has failed.

*****************************************************************************

Error Name:   Flash_OVERLAY_DISABLED
Return Value: -20
Description:  This error informs the user that a OverlayBlockDisable command
   was not neccesary because the Overlay Block was already disabled.

*****************************************************************************

Error Name:   Flash_OVERLAY_ENABLED
Return Value: -21
Description:  This error informs the user that a OverlayBlockEnable command
   was not neccesary because the Overlay Block was already enabled.

*****************************************************************************/
```

If you have any questions or suggestion concerning the matters raised in this document please send them to the following electronic mail address:

ask.memory@st.com (for general enquiries)

Please remember to include your name, company, location, telephone number and fax number.

littlediode
COMPONENTS FOR ALL

Littlediode supplies new, hard to find or obsolete electronic components and semiconductors all over the world.

With over two million different components listed you are sure to find the part you need.

Feel free to visit us today at our online store:

LittleDiode.com

Looking forward to providing you with the best possible service.