AN1473
APPLICATION NOTE

How to use the Write to Buffer feature of the M58LW064 family of Flash Memories

CONTENTS

INTRODUCTION

The M58LW064, a 64 Mbit (4Mb x16 or 2Mb x32) non-volatile memory, is the first in a new family of leading-edge Flash memories from ST Microelectronics that uses multilevel cell technology. It can be read, erased and reprogrammed using a single low voltage (2.7V to 3.6V) core supply. Two version are available, the M58LW064A with a x16 bus and the M58LW064B with a x16/x32 bus. The memory is offered in various packages. The M58LW064A is available in TSOP56 (14 x 20 mm) and TBGA64 (1mm pitch). The M58LW064B is available in TBGA80 (1mm pitch).

The device has many advanced features: x16/x32 bus width, synchronous burst read, individual block protection, a One Time Programmable area and a Write to Buffer and Program command to speed up the programming of the device.

This Application Note explains how to use the Write to Buffer feature, emphasizing the Software Design considerations.

DESCRIPTION

The Write to Buffer and Program command is used to program the memory array. The memory is divided into Blocks that can be erased independently. When using the Write to Buffer command it is also useful to think of the memory as divided into Buffers and Pages, where:

- a Page is 4 Words (or 2 Double Words), with A22-A3 fixed, and each Word within the Page is selected using A1 and A2.
- a Buffer is 4 Pages (total of 16 Words or 8 Double Words) with A22-A5 fixed, and each Page within the Buffer is selected with addresses A3 and A4.

The Write Buffer allows the microprocessor to program from 4 to 16 Words (or from 2 to 8 Double Words) in parallel, both speeding up the programming and freeing up the microprocessor to perform other work. Therefore the minimum buffer size for a program operation is a 4 Word (or 2 Double Word) Page. Any attempt to program less than 4 Words (or 2 Double Words) inside the Page of a previously erased Block will result in the correct programming of the Word, however all other Words inside the same Buffer will be set to FFFFh.

Any attempt to change the Buffer (change one of the addresses A22-A5) within the same Write to Buffer command will be ignored. Only the first A22-A5 given for the first Program Address will be considered (see Example 2).

Reprogramming

Once a Buffer (1 to 4 Pages) has been programmed it cannot be reprogrammed until the Block containing that Buffer has been erased (regardless of the current data and the data to be programmed).

If a program operation is attempted in a Buffer which was not previously erased, the operation will abort and the Program/Erase Controller will set the error bits in the Status Register. When the Status Register error bits are set under these circumstances they cannot be reset by a Clear Status Register command. In this case they can only be reset by a Hardware Reset or Power-Up sequence. No further write operations are possible until the Status Register has been reset.

To avoid the inconvenience of a Hardware Reset (which is generally not permitted on a board), it is suggested to modify the write to buffer algorithm in the Software drivers to check if the Buffer has been previously erased. A read cycle should be added to check that the content of the Buffer is FFFFh, followed by a Block Erase cycle if necessary, before starting the write to buffer sequence. This does not affect the performance of the device as the delay introduced by this check is negligible compared to the programming time. Refer to Figure 1 for the recommended flowchart and Appendix A for the recommended C-code.

If the application does not require to reprogram the device then the standard flowchart shown in the M58LW064 datasheet can be used.

USING THE WRITE TO BUFFER AND PROGRAM COMMAND

The Write to Buffer and Program command consists of (N+4) bus cycles, where (N+1) is the number of Words to be programmed. The cycles must be asserted according to the timings described in the datasheet.

As mentioned in the Reprogramming section above, it is recommended to check that the Buffer has been pre-erased before issuing the Write to Buffer and Program command. Figure 1, Write to Buffer and Program Flowchart, shows the suggested flowchart for using the Write to Buffer and Program command. It includes a read cycle to check if the Buffer has been pre-erased and a Block Erase cycle to erase the Block if necessary.

Once the check has been done four successive steps are required to issue the command.

1. One Bus Write operation is required to set up the Write to Buffer and Program Command. Issue the set up command (E8h) with the selected memory Block Address (BA) where the program operation should occur (any address in the block where the data will be programmed can be used). Any Bus Read operations will start to output the Status Register after the 1st cycle. Bit 7 of the Status Register can be checked to verify if the Program/Erase Controller (P/E.C) is ready to receive the next cycle (Bit7 = 1).
2. Use one Bus Write operation to write the same block address (BA) along with the value N on the Data Inputs/Output, where N+1 is the number of Words (x16 Bus Width) or Double Words (x32 Bus Width) to be programmed.
3. Use N+1 Bus Write operations to load the address and data for each Word or Double Word into the Write Buffer. See the constraints on the address combinations listed below. The addresses must have the same A5-A22.
4. Finally, use one Bus Write operation to issue the final cycle to confirm the command (D0h) and start the Program operation.

After the program operation has started the Status Register can be read by toggling Output Enable ($\overline{G}$) to follow the progress of the operation. When Program/Erase Controller Bit 7 of the Status Register goes to High the operation is finished.

Invalid address combinations or failing to follow the correct sequence of Bus Write cycles will set an error in the Status Register and abort the operation without affecting the data in the memory array. The Status Register should be cleared before re-issuing the command.

If the block being programmed is protected an error will be set in the Status Register and the operation will abort without affecting the data in the memory array. The block must be unprotected using the Blocks Unprotect command or by using the Blocks Temporary Unprotect feature of the Reset/Power-Down pin, $\overline{RP}$. See datasheet for more details.

Programming Time

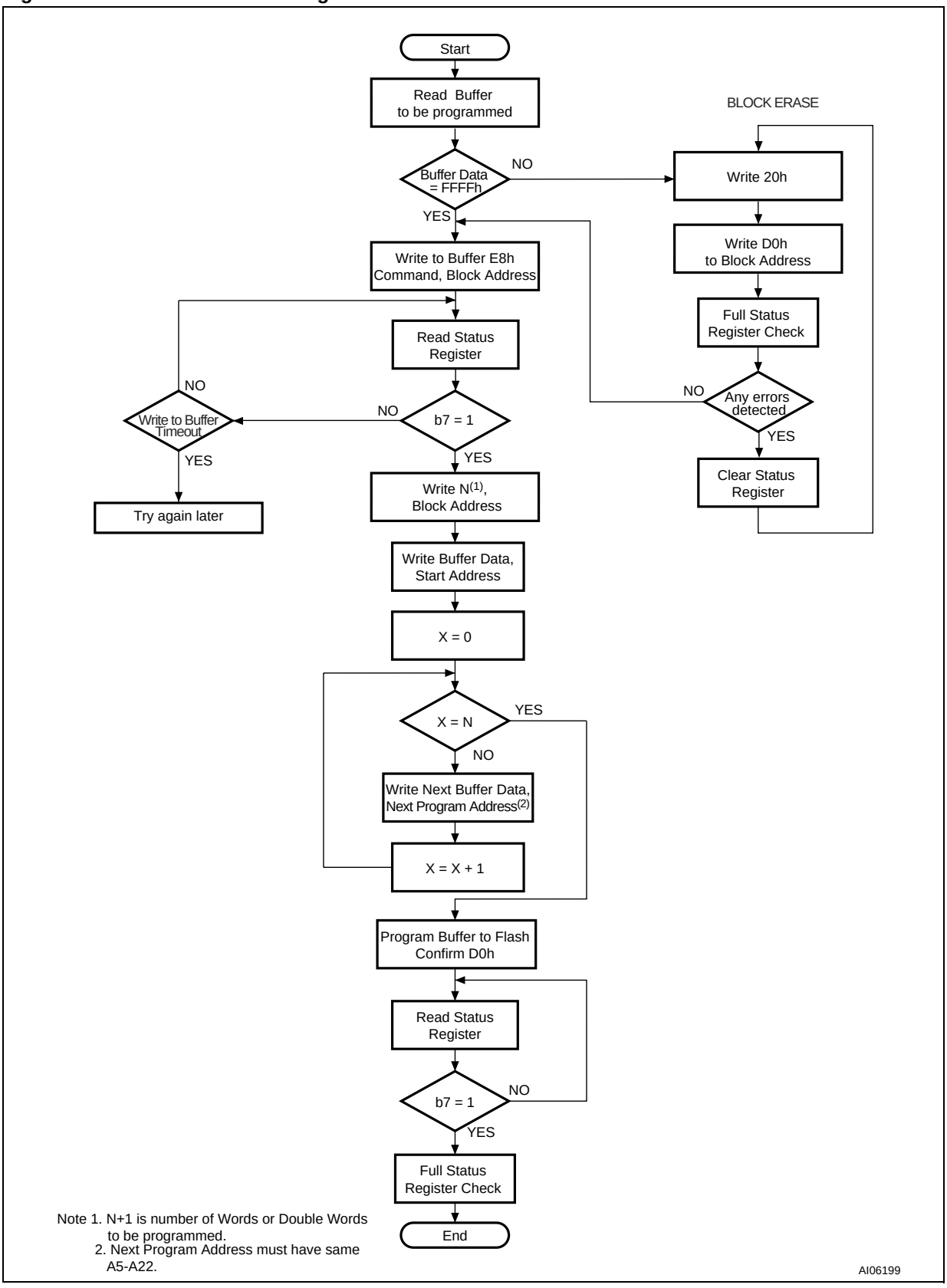
The equivalent programming time for a number of Words N (N= 4 to 16) programmed within the same Write to Buffer Command is:

(Buffer Program Time)/N

So if you program the whole buffer within the same command you will obtain an equivalent program time per word of (Buffer Program Time)/16. To optimize the programming time using the Write to Buffer feature the whole buffer should be programmed at once.

Please refer to the “Program, Erase Times and Program Erase Endurance Cycles” Table of the datasheet for the buffer programming time.

Figure 1. Write to Buffer and Program Flowchart



SUSPENDING A WRITE TO BUFFER AND PROGRAM COMMAND

The M58LW064 features a Program/Erase Suspend command which can be used to pause a Write to Buffer and Program operation to allow a Read operation in another block or pause an Erase operation to perform a Program or Read operation in another block.

One Bus Write cycle is required to issue the Program/Erase Suspend command (B0h, Address Don't Care) and pause the Program/Erase Controller. To resume the operation the Erase Resume command can be issued (D0h, Address Don't Care). Please refer to the datasheet for further details.

STATUS REGISTER BITS

The Status Register provides information on the current or previous Program, Erase, Block Protect or Blocks Unprotect operation. The various bits in the Status Register convey information and errors on the operation. They are output on DQ7-DQ0.

Table 1 provides a summary of the Status Register bits associated with the Write to Buffer and Program command.

Program/Erase Controller Status (Bit 7). The Program/Erase Controller Status bit indicates whether the Program/Erase Controller is active or inactive. When the Program/Erase Controller Status bit is Low, the Program/Erase Controller is active and all other Status Register bits are High Impedance; when the bit is High, the Program/Erase Controller is inactive, and the operation has ended or is suspended. The contents of the entire Status Register should be checked for errors.

Program Status (Bit 4). The Program Status bit is used to identify a Program or Block Protect failure.

When the Program Status bit is Low, the memory has successfully verified that the Write Buffer has programmed correctly or the block is protected. When the Program Status bit is High, the Program or Block Protect operation has failed. Depending on the cause of the failure other Status Register bits may also be set to High.

- If only the Program Status bit (bit 4) is set High, V_{OH}, then the Program/Erase Controller has applied the maximum number of pulses to the byte and still failed to verify that the Write Buffer has programmed correctly or that the Block is protected.
- If the failure is due to a program or block protect with V_{PP} low, then V_{PP} Status bit (bit 3) is also set High.
- If the failure is due to a program on a protected block then Block Protection Status bit (bit 1) is also set High.
- If the failure is due to a program or erase incorrect command sequence then Erase Status bit (bit 5) is also set High.

V_{PP} Status (Bit 3). The V_{PP} Status bit can be used to identify if a Program, Erase, Block Protection or Block Unprotection operation has been attempted when V_{PP} is Low. The V_{PP} pin is only sampled at the beginning of a Program or Erase operation.

When the V_{PP} Status bit is Low, no Program, Erase, Block Protect or Blocks Unprotect operations have been attempted with V_{PP} Low, since the last Clear Status Register command, or hardware reset. When the V_{PP} Status bit is High, a Program, Erase, Block Protection or Block Unprotection operation has been attempted with V_{PP} Low.

Program Suspend Status (Bit 2). The Program Suspend Status bit indicates that a Program operation has been suspended and is waiting to be resumed. The Program Suspend Status should only be considered valid when the Program/Erase Controller Status bit is High.

When the Program Suspend Status bit is Low, the Program/Erase Controller is active or has completed its operation; when the bit is High, a Program/Erase Suspend command has been issued and the memory is waiting for a Program/Erase Resume command.

When a Program/Erase Resume command is issued the Program Suspend Status bit returns Low.

Block Protection Status (Bit 1). The Block Protection Status bit can be used to identify if a Program or Erase operation has tried to modify the contents of a protected block.

When the Block Protection Status bit is Low, no Program or Erase operations have been attempted to protected blocks since the last Clear Status Register command or hardware reset; when the Block Protection Status bit is High, a Program or Erase operation has been attempted on a protected block.

Once bits 4, 3 and 1 are set High, they can only be reset by a Clear Status Register command or a hardware reset. Refer to the datasheet for the other bits in the Status Register.

Table 1. Summary of Status Register Bits Associated with Write to Buffer Command

Status Register bits	meaning
Bit 7=0	The internal P/E.C. is busy All other bits are High Impedance
Bit 7=1	The internal P/E.C. is ready
Bit 4=1	Program error
Bit4=1 Bit1=1	Program Block protect failure
Bit4=1 Bit3=1	Program failure due to V_{PP} error
Bit 7=1 Bit 2=1	Program suspended Reading possible on another block
Bit 7=1 Bit 5=1	Erase suspended Programming/Reading possible on another block

COMMON FLASH INTERFACE

The M58LW064 supports the Common Flash Interface (CFI). The CFI is a JEDEC approved, standardized data structure that can be read from the Flash memory device. It allows a system software to query the device to determine various electrical and timing parameters, density information and functions supported by the memory. The CFI contains information on the size of the Buffer and information on the typical timeout for the Write to Buffer command. Refer to the datasheet for further information.

EXAMPLES

Example 1: Programming Four Words

Table 2, shows the series of bus cycles required to program the data: 0101h (0000000100000001 in binary), 0A0Ah (0000101000001010), B1B1h (1011000110110001), CCCCh (1100110011001100) at the first four addresses of a previously erased device, where the initial Status Register value is 80h (bit 7 = 1, P/E.C inactive). Table 3 shows the results, all data are programmed correctly and no errors are signalled in the Status Register. The equivalent programming time per word is: (Buffer programming time)/4.

Table 2. Example 1, Command Flow

Cycle No.	Bus cycle type	Address	Data	Comment
1	Write	0000h	E8h	Write to Buffer set up command
2	Read	0000h	Status Register	Toggle $\overline{E}$ to check SR. Bit7 must be '1' to proceed
3	Write	0000h	3h	Number of words to be programmed minus one
4	Write	0000h	0101h	Address, Content
5	Write	0001h	0A0Ah	Address, Content
6	Write	0002h	B1B1h	Address, Content
7	Write	0003h	CCCCh	Address, Content
8	Write	X	D0h	Confirm command
9	Read	0000h	Status Register	Toggle $\overline{E}$ to check SR. If Bit7is '1' the operation is finished
10	Write	X	FFh	Command to reset the memory to the Read Array mode
11	Write Read	X X	70h Status Register	Read SR to verify if the operation completed successfully (SR=80h)

Note: X = Don't Care, SR = 80h = 10000000 i.e. bit 7 = 1, bits6-0 = 0.

Table 3. Example 1, Results

Address	Initial Data	Data to Program	Final Data
0000h	FFFFh	0101h	0101h
0001h		0A0Ah	0A0Ah
0002h		B1B1h	B1B1h
0003h		CCCCh	CCCCh
Status Register	80h	-	80h

Note: SR = 80h = 10000000 i.e. bit 7 = 1, bits6-0 = 0.

Example 2: Programming Sixteen Words

Table 4, shows the series of bus cycles required to program the data: 0000h, 0001h, 0002h, …, 000Fh starting from address 0008h of a previously erased device, where the initial Status Register value is 80h (bit 7 = 1, P/E.C inactive). Table 5 shows the results. At the end of the operation the Status Register will output 80h, and so it appears as thought the program operation was successful. However this is not entirely true as in this case, there is a change of buffer (change of address A5) which is ignored and so addresses 0010h to 0017h remain unchanged (FFFFh) while the programmed data from 0008h to 000Fh will appear in the first addresses of the buffer.

The equivalent programming time per word is: (Buffer programming time)/16.

Table 4. Example 3, Command Flow

Cycle No.	Bus cycle type	Address	Data	Comment
1	Write	0000h	E8h	Write to Buffer set up command
2	Read	0000h	Status Register	Toggle $\overline{E}$ to check SR. Bit7 must be '1' to proceed
3	Write	0000h	3h	Number of words to be programmed minus one
4	Write	0008h	0000h	Address, Content
5	Write	0009h	0001h	Address, Content
6	Write	000Ah	0002h	Address, Content
7	Write	000Bh	0003h	Address, Content
8	Write	000Ch	0004h	Address, Content
9	Write	000Dh	0005h	Address, Content
10	Write	000Eh	0006h	Address, Content
11	Write	000Fh	0007h	Address, Content
12	Write	0010h	0008h	Address, Content
13	Write	0011h	0009h	Address, Content
14	Write	0012h	000Ah	Address, Content
15	Write	0013h	000Bh	Address, Content
16	Write	0014h	000Ch	Address, Content
17	Write	0015h	000Dh	Address, Content
18	Write	0016h	000Eh	Address, Content
19	Write	0017h	000Fh	Address, Content
20	Write	X	D0h	Confirm command
21	Read	0000h	Status Register	Toggle $\overline{E}$ to check SR. If Bit7is '1' the operation is finished
22	Write	X	FFh	Command to reset the memory to the Read Array mode
23	Write Read	X X	70h Status Register	Read SR to verify if the operation completed successfully (SR=80h)

Note: X = Don't Care, SR = 80h = 10000000 i.e. bit 7 = 1, bits6-0 = 0.

Table 5. Example 3, Results

Address	Initial Data	Data to Program	Final Data
0000h	FFFFh	-	0008h
0001h		-	0009h
0002h		-	000Ah
0003h		-	000Bh
0004h		-	000Ch
0005h		-	000Dh
0006h		-	000Eh
0007h		-	000Fh
0008h		0000h	0000h
0009h		0001h	0001h
000Ah		0002h	0002h
000Bh		0003h	0003h
000Ch		0004h	0004h
000Dh		0005h	0005h
000Eh		0006h	0006h
000Fh		0007h	0007h
0010h		0008h	FFFFh
0011h		0009h	FFFFh
0012h		000Ah	FFFFh
0013h		000Bh	FFFFh
0014h		000Ch	FFFFh
0015h		000Dh	FFFFh
0016h		000Eh	FFFFh
0017h		000Fh	FFFFh
Status Register	80h	-	80h

Note:SR = 80h = 10000000 i.e. bit 7 = 1, bits6-0 = 0.

REVISION HISTORY

Table 6. Document Revision History

Date	Version	Revision Details
01-Feb-2002	-01	First Issue

APPENDIX A. WRITE TO BUFFER AND PROGRAM C-CODE EXAMPLE

Below is an example for the function FlashProgram, which can be implemented in Software, in order to reproduce the flowchart shown in Figure 1.

```
/*****************************************************************************
Function:     ReturnType FlashProgram( udword udMode, udword udAddrOff,
   udword udNrOfElementsInArray, void *pArray )
Arguments:    udMode changes between programming modes
   udAddrOff is the address offset into the flash to be programmed
   udNrOfElementsInArray holds the number of (double)words in the array. pArray
   is a void pointer to the array with the contents to be programmed.
Return Value: The function returns the following conditions:
   FLASH_SUCCESS                  successful operation
   FLASH_ADDRESS_INVALID          program range outside device
   FLASH_BLOCK_PROTECTED          block to program is protected
   FLASH_DOUBLE_PROGRAMM_ATTEMPT  the area to be programmed contains already
                                  programmed (double)words
   FLASH_PROGRAM_FAILED           failure not covered below
   FLASH_VPP_INVALID              Vpp is not valid
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.
   Note that the function always programs all addresses within the same page by
   issuing a single program command, as is required by the device architecture.
   Once the program command has completed the function checks the Status
   Register for errors. Any errors are returned without any further attempts to
   program other addresses of the device. The function returns FLASH_SUCCESS
   when all addresses have successfully been programmed.

   Note: Two program modes are available:
   - udMode = 0, Normal Programming Mode
   The number of elements (udNumberOfElementsInArray) contained in pArray
   are programmed directly to the udAddrOff within the flash
   - udMode > 0, Repeated Values Program Mode
   udMode is now a number of Values, which is contained in pArray. These
   numbers are repeatedly programmed according to the area defined by
   udAddrOff and udNumberOfElementsInArray.

   For Example: Values in pArray: 11,3,32  |1
                         udMode: 3        |1
                      udAddrOff: 100      |100
       udNumberOfElementsInArray: 7        |7

                Results in Memory: Addr:100: Content: 11 | 1
                                   Addr:101: Content: 3  | 1
                                   Addr:102: Content: 32 | 1

                                   Addr:103: Content: 11 | 1
                                   Addr:104: Content: 3  | 1
                                   Addr:105: Content: 32 | 1

                                   Addr:106: Content: 11 | 1
```

```
Pseudo Code:
   Step 1:  Check whether the data to be programmed are are within the
            Flash memory
   Step 1a: Check whether the intended location is ready to program
   Step 2:  While there is more to be programmed
   Step 3:  Determine limits of current set of 4/2 pages (word/dword)
   Step 4:  Program within the next set of 4/2 pages (word/dword)
   Step 5:  Decision between direct and modulo programming
   Step 6:  Wait until the Program/Erase Controller is ready
   Step 7:  Check for any errors
   Step 8:  Clear Status Register and return to Read Array mode
   Step 9:  Return the error condition
*****************************************************************************/
ReturnType FlashProgram(udword udMode, udword udAddrOff, udword udNrOfElementsIn
Array, void *pArray ) {
   ReturnType rRetVal = FLASH_SUCCESS; /* Return Value: Initially optimistic */
   uCPUBusType *ucpArrayPointer;  /* Use an uCPUBusType to access the array */
   uCPUBusType start,end, a;      /* Holds temp Variables  for Step 1a */
   udword udLastOff;          /* Holds the last offset to be programmed */
   udword udEndSet;           /* Holds the end of the current set of pages */
   uCPUBusType ucStatus;      /* Holds the Status Register reads */
   udword udNrOfElementsToPrg;     /* Number of words/double-words to be
   programmmed */
   udword udModuloCounter = 0;     /* Holds Counter for Repeated Programming
   Mode */

   /* Check whether the data to be programmed are are within the Flash memory
   space*/
   udLastOff = udAddrOff + udNrOfElementsInArray - 1;
   if( udLastOff >= FLASH_SIZE )
        return FLASH_ADDRESS_INVALID;

   /* Step 1a: Check whether the intended location is ready to program */
   start = udAddrOff;
   end   = udLastOff;
   for (a = start; a<=end; a++) {
       if (~FlashRead(a) != 0) { /* Only words/dwords set to 1 can be programmed
       */
          return FLASH_DOUBLE_PROGRAM_ATTEMPT;
       } /* Endif*/
   } /* Next a */

   /* Step 2: While there is more to be programmed */
   ucpArrayPointer = (uCPUBusType *)pArray;

   while( udAddrOff <= udLastOff && rRetVal == FLASH_SUCCESS ) {
      /* Step 3: Determine limits of current set of 4/2 pages (word/dword) */
      udEndSet = udAddrOff | (FLASH_WRITE_BUFFER_SIZE - 1);
      if( udEndSet > udLastOff )
         udEndSet = udLastOff;
      udNrOfElementsToPrg = udEndSet - udAddrOff + 1; /* Number of (double)words
      to beprogrammed */
```

```
      /* Step 4: Program within the next set of 4/2 pages (word/dword) */
      FlashWrite( ANY_ADDR, CMD(0x0050) ); /* Clear Status Register */

      /* NOTE ! CSR also clears bit 1 BPS as well as bits 3, 4 and 5 */
      FlashWrite( udAddrOff, (uCPUBusType)CMD(0x00E8) );  /* Program Setup */
      FlashWrite( udAddrOff, CMD((uCPUBusType)(udNrOfElementsToPrg - 1)) ); /*
      Number of (double)words */

      /* Step 5: Decision between direct and modulo programming */
      if (udMode == 0) {
         while( udAddrOff <= udEndSet )
            FlashWrite( udAddrOff++, *(ucpArrayPointer++) );  /* Direct Value !!
            Program value */
      } else {
         while( udAddrOff <= udEndSet )
            FlashWrite( udAddrOff++, ucpArrayPointer[(udModuloCounter++) %
            udMode] );
      } /* Endif Programming Mode Check */

      FlashWrite( ANY_ADDR, (uCPUBusType)CMD(0x00D0) );  /* Confirm program */

      /* Step 6: Wait until Program/Erase Controller is ready */
      FlashTimeOut(0); /* Initialize TimeOut Counter */
      do {
         ucStatus = FlashRead(ANY_ADDR);
         if (FlashTimeOut(5) == FLASH_OPERATION_TIMEOUT) {
            FlashReset();
            return FLASH_OPERATION_TIMEOUT;
         } /* EndIf */
      } while( (ucStatus & CMDD(0x0080)) != CMDD(0x0080) );
      /* Wait until every Action is finished (StatusRegister Bit7 = 1) */

      /* Step 7: Check for any errors */
      if( ucStatus & CMDD(0x0008) )
         rRetVal = FLASH_VPP_INVALID;
      else if( ucStatus & CMDD(0x0002) )
         rRetVal = FLASH_BLOCK_PROTECTED;
      else if( ucStatus & CMDD(0x0020) )
         rRetVal = FLASH_PROGRAM_FAILED;
      else if( ucStatus & CMDD(0x0010) )
         rRetVal = FLASH_PROGRAM_FAILED;

   } /* EndWhile  Main Program Loop */

   /* Step 8: Clear Status Register and return to Read Array mode */
   FlashWrite( ANY_ADDR, CMD(0x0050) ); /* Clear Status Register */
   /* NOTE ! CSR also clears bit 1 BPS as well as bits 3, 4 and 5 */
   FlashReset(); /* Read Array Command */

   /* Step 9: Return the error condition */
   return rRetVal;
} /* EndFunction FlashProgram */
```

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.