



# SOFTWARE DRIVERS FOR W19B(L)320ST/B FLASH MEMORIES

## **Table of Contents-**

1.	GENERAL DESCRIPTION .....	2
2.	W19B(L)320ST/B PROGRAMMING MODEL .....	2
2.1	Bus Operations and Commands .....	3
2.2	Operation Status Bits .....	4
2.3	A Detail Example .....	4
3.	HOW TO USE THE SOFTWARE DRIVERS .....	6
3.1	General Considerations .....	6
3.2	Porting the Drivers to the Target System .....	7
3.3	C Library Functions Provided .....	9
3.4	Getting Started.....	12
4.	CONCLUSION .....	13
5.	VERSION HISTORY .....	14

# SOFTWARE DRIVERS FOR W19B(L)320ST/B FLASH MEMORIES



## 1. GENERAL DESCRIPTION

This application note provides library source code in C for the W19B(L)320ST/B Flash memories. The low-level drivers described in this application note have been provided to simplify the process of developing application code in C for Winbond Flash Memory W19B(L)320ST/B. This enables users to concentrate on writing the high level functions required for their particular applications. These high level functions can access Flash Memories by calling the low level drivers, and further keeping the detail special command sequences away from users' high level code: this will result in a simpler and easier way to maintain source code.

The overview of the programming model for the W19B(L)320ST/B is also included in this application note. This will familiarize users 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 it requires the minimal changes by users in order to compile and run. This note explains how the user should modify the source code for individual target hardware. The source code contains comments throughout, explaining how it is used and why it has been written the way it is. For example, since W19B(L)320ST/B supports two kind of bus operation modes, WORD mode and BYTE mode, the SW engineer must re-build the low-level driver for an indicated uP bus width either WORD(x16) or BYTE(x8) for accessing W19B(L)320ST/B Flash memory.

This application note should be read along with W19B(L)320ST/B datasheet to follow some of the explanations. The software, supplied with this application note, has been tested on a target platform (W90N740 ARM7TDMI), and is usable in C environments. It is small in size and can be applied to any target hardware.

The W19B(L)320ST/B provides all the hardware and software functionality compatible with AMD Am29xV32x, and more. Source code written for AMD Am29x320x can easily be modified to be applied on Winbond W19B(L)320ST/B parts.

## 2. W19B(L)320ST/B PROGRAMMING MODEL

The W19B(L)320ST/B is a 32 Mbit (4Mbit x 8 or 2 Mbit x 16) Flash memory, which can be electrically erased at sector level and programmed in-system on a Byte-by-Byte or Word-by-Word basis through special coded command sequences on most standard microprocessor buses. The devices feature an asymmetrical sector architecture. The W19B(L)320ST/B has an array of 71 sectors: 8 Parameter Sectors of 8(4) KBytes(Words) and 63 Main Sectors of 64(32) KBytes(Words). The W19B(L)320ST/B has the Parameter Sectors at the top or bottom of the memory address space that the Parameter Sectors of W19B(L)320SB is at the bottom sector ;while W19B(L)320ST is at the top. Each sector can be erased separately. Erase can be suspended in order to either read from or program to any other sectors, and then resumed. Each sector can be programmed and erased approximately 100,000 cycles.

The 8 Parameter Sectors can be protected individually while the 63 Main Sectors can be protected in groups (4 sectors per group). For Sector Unprotect, the algorithm will unprotect all sectors in parallel and all unprotected sectors must first be protected before the unprotect write command cycle is issued. Please refer to W19B(L)320ST/B datasheet describes In-system Sector Protect/Unprotect Algorithms section for detail information and the source code accompanied in this application note. From the viewpoint of application, the Temporary Sector Unprotect function is much efficient when system software wants to revise data in protected sector. The VID is needed on the #RESET pin in both Sector Protect/Unprotect and Temporary Sector Unprotect function. W19B(L)320ST/B offers an additional hardware protection: when #WP is low, the outermost boot sectors are protected regardless of sector protect status.

# SOFTWARE DRIVERS FOR W19B(L)320ST/B FLASH MEMORIES



Program and Erase commands are written to the Command Interface of the device. The end of a program or erase operation can be detected and any error conditions will be identified. The Command Set required to control the device is consistent with JEDEC standards.

The W19B(L)320ST/B devices offer three program modes to improve the programming throughput: **Normal Program, Unlock Bypass Program and ACC Program (V<sub>HH</sub> on the #WP/ACC pin )**

## 2.1 Bus Operations and Commands

Most W19B(L)320ST/B functionalities are 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 stored data or the device's behavior. Only certain write operation sequences can be recognized as commands by W19B(L)320ST/B. The various commands recognized by the devices are listed in the Commands Tables provided in the corresponding datasheets. The main commands can be classified as follows:

1. Read Array Data, Security Sector And Common Flash Interface(CFI) Query
2. Erase
3. Normal Program, Unlock Bypass Program & ACC Program
4. Erase Suspend And Resume.
5. Sector Protect/Unprotect, Temporary Sector Unprotect

The Read command returns the W19B(L)320ST/B to Read Mode where it acts as ROMs. In this state, a read operation will output the data, stored at the specified device address, onto the data bus.

Enter Autoselect mode command places the devices in a mode which allows the user to read the manufacture ID, Device ID, check Security Sector Factory Protect and Sector Protect Verify. These are accessed by reading different offset addresses.

The Erase command is used to set all the bits to '1' at every memory location in the selected sector. The 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.

The Program command is used to modify the data stored at the specified device address. Note that programming can only change bits from '1' to '0'. If an attempt is made to change a bit from '0' to '1' using the Program command, the command will be executed and no error will be signaled but the bit will remain unchanged. It may therefore be necessary to erase the sector before programming to addresses within it. Programming modifies a single Byte or Word at a time.

Issuing the Erase Suspend command during an erase operation will temporarily place the W19B(L)320ST/B in Erase Suspend Mode. When in Erase Suspend Mode, the interruption to a sector erase operation and then read data from or program data to any sector not selected for erasure is allowed. This allows users to access information stored in the W19B(L)320ST/B immediately without waiting for the completion of erase operation. The erase operation is resumed when the device receives the Erase Resume command.

The Read Common Flash Interface (CFI) Query command allows users to identify the information relating to the typical and maximum Program, Erase times. This allows users to implement software timeouts and prevent waiting indefinitely for a defective Flash memory to finish programming or erasing. This CFI interface also allows software to identify the physical memory layout and command sets of the Flash memory. For further information about the CFI, please refer to the CFI specification available from the internet site.  
([http:// www.jedec.org](http://www.jedec.org)).

# SOFTWARE DRIVERS FOR W19B(L)320ST/B FLASH MEMORIES



## 2.2 Operation Status Bits

During program or erase operations, a bus read operation will output the contents of the Status Bits, i.e., DQ2, DQ3, DQ5, DQ6 and DQ7. The Operation Status Bits are described in the Write Operation Status Bits Tables provided in the W19B(L)320ST/B datasheets. They are mainly used to determine when programming or erasing is complete and whether the operation was successful or not. Please note the Chip/Sector Erase Command can only be executed on unprotected sectors. After writing an erase command sequence, if not all selected sectors are protected, the Embedded Erase algorithm will only erase the unprotected sectors and ignore the protected sectors. However, if the system reads DQ7 at an address within a protected sector, the status may not be valid. DQ6 Toggle bit therefore is a better method to check the erasure status.

In addition, the RY/#BY is a dedicated pin to indicate whether an Embedded Algorithm is in progress or complete. This RY/#BY pin offers another operation status checking by HW instead of SW polling. By using this RY/#BY pin, system can get a good performance and more efficiency in multi-tasks OS since system SW does not need to keep on polling operation status bits.

## 2.3 A Detail Example

The Command Tables provided in the W19B(L)320ST/B datasheets describe the Bus Write sequences recognized as valid commands by device.

Accessing the Manufacture & Device ID codes and then checking if they are correct or not can be as an example. If the data is correct then it is likely that all of the functions can work normal since this process includes "Write" & "Read" command cycles sequence for memory device in system.

```
// Write the Read Identifier Codes Command into the Flash
*(DWIDE*) (WflashBase + CMD_1stCYCLE_ADDR) = CMD_1stCYCLE_DATA ;
*(DWIDE*) (WflashBase + CMD_2stCYCLE_ADDR) = CMD_2stCYCLE_DATA ;
*(DWIDE*) (WflashBase + CMD_3stCYCLE_ADDR) = CMD_3stCYCLE_DATA ;

// Read & Store Manufacturer code
VendorID = *(DWIDE*) ( WflashBase + ID_MAKER_OFFSET );

// Write the Read Identifier Codes Command into the Flash
*(DWIDE*) (WflashBase + CMD_1stCYCLE_ADDR) = CMD_1stCYCLE_DATA ;
*(DWIDE*) (WflashBase + CMD_2stCYCLE_ADDR) = CMD_2stCYCLE_DATA ;
*(DWIDE*) (WflashBase + CMD_3stCYCLE_ADDR) = CMD_3stCYCLE_DATA ;

// Read & Store Device code
DeviceID =*(DWIDE*) (base_add + ID_DEVICE_OFFSET);
```

where

(1) DWIDE is defined as the following 16-bit value

```
#define DWIDE          volatile unsigned short
```

or defined as the following 8-bit value

```
#define DWIDE          volatile unsigned char
```

# SOFTWARE DRIVERS FOR W19B(L)320ST/B FLASH MEMORIES



(2) WflashBase is a flash device base address in microprocessor address space.

CMD\_XstCYCLE\_ADDR, X=1,2,3 is a special coded command sequences on address pins,

CMD\_XstCYCLE\_DATA, X=1,2,3 is a special coded command sequences on data bus lines.

When device bus operation is in WORD mode, CMD\_XstCYCLE\_ADDR are 0x555,0x2AA and 0x555 as well as in BYTE mode CMD\_XstCYCLE\_ADDR are 0xAAA,0x555 and 0xAAA.

The first of the three addresses (WflashBase + CMD\_XstCYCLE\_ADDR) is arbitrary; however, they must be inside the Flash memory address space. We could assume that the address 0000h in the W19B(L)320ST/B is mapped to the address 0000h in the microprocessor address space. In practice, the Flash memories are likely to have a base offset that has to be added to the address.

Another example, consider the programming of the value 56h to the address 0639h. The required C language sequence is as following:

```
// Write 1st ~ 3st the Word/Byte Command into the Flash
*(DWIDE*)(WFlashBase + CMD_1stCYCLE_ADDR) = CMD_1stCYCLE_DATA ;
*(DWIDE*)(WFlashBase + CMD_2stCYCLE_ADDR) = CMD_2stCYCLE_DATA ;
*(DWIDE*)(WFlashBase + CMD_3stCYCLE_ADDR) = CMD_3stWRITE_DATA ;
// Write the Data into the Flash address
*(DWIDE*)(0x0639) = 0x56;
```

While the device is programming to the specified address, read operations will access the Operation Status Bits. The device outputs on DQ7 the complement of datum programmed to DQ7 if system software uses the DQ7 to check the programming is in progress or complete as well as DQ6 Toggle Bit is another method. Both DQ7 and DQ6 are able to be combined with DQ5 to indicate whether the program time has exceeded a specified internal pulse count limit. Please refer to the following source code and the flowcharts in W19B(L)320ST/B datasheet.

```
#if defined(DQ7_POLLING)
// Wait until the DQ bit7 is a complement of the datum programming data, to check whether
// DQ5 is Timeout or not and then to read DQ7 again
```

```
while( (*address & DQ7_PROGE_READY) != (*data & DQ7_PROGE_READY) )
{
    if( *address & DQ5_EXCEED_TIME )
    {
        if( (*address&DQ7_PROGE_READY) == (*data & DQ7_PROGE_READY) ){ break;}
        else {errcode=ERR_DQ5_TIMEOUT; break;}
    }
};
#elif defined(DQ6_TOGGLEBIT) // Using DQ6_TOGGLEBIT
```

# SOFTWARE DRIVERS FOR W19B(L)320ST/B FLASH MEMORIES



```
// Wait until the DQ bit6 no more Toggle, check whether DQ5 Timeout or not & then Check DQ6 //  
Toggle status again
```

```
do{  
    BitDQ6a>(*address & DQ6_TOGGLE_HIGH );  
    BitDQ6b>(*address & DQ5_6TOGGLE_TIMEOUT );  
    if(BitDQ6a == (BitDQ6b & DQ6_TOGGLE_HIGH) ) return errcode;  
}while(!(BitDQ6b & DQ5_EXCEED_TIME));  
  
BitDQ6a>(*address & DQ6_TOGGLE_HIGH );  
BitDQ6b>(*address & DQ6_TOGGLE_HIGH );  
if(BitDQ6a == BitDQ6b ) return errcode;  
else errcode= ERR_DQ6_TOGGLE ;  
  
#else  
#error "Please define either DQ7_POLLING or DQ6_TOGGLEBIT!!!"  
#endif
```

Once programmed, the address 0639h cannot be reprogrammed reliably until an Erase operation is issued to erase the entire sector.

## 3. HOW TO USE THE SOFTWARE DRIVERS

### 3.1 General Considerations

The software drivers (Low-Level Driver) described in this application note are intended to simplify the process of developing an application code in C for the Winbond W19B(L)320ST/B Flash devices. With the software driver interface, users can focus on writing the high-level code required for their particular applications. The high-level code accesses the Flash memory by calling the low-level code, so users do not have to concern themselves with the details of the special command sequences.

The low-level code requires hardware-specific Read and Write Bus operations in C in order to communicate with the W19B(L)320ST/B. The implementation of these operations is hardware-platform dependent as it depends on the microprocessor on which the C code runs and on the location of the memory in the microprocessor's address space.

The user will have to write the C drivers that are suitable for the current hardware platform. The low-level code takes care of issuing the correct write operation sequence for each command, and of interpreting the information received from the devices during programming and erasing.

The high-level code written by the user accesses the memory devices by calling the low-level code. In this way, the code used is simple and easier to maintain. When developing an application, the user is advised to proceed as follows:

- First, write a simple program to test the low-level code provided, and verify that it operates as expected in the user's target hardware and software environments;

# SOFTWARE DRIVERS FOR W19B(L)320ST/B FLASH MEMORIES



- Then, write a high-level code for the desired application. The application will access the Flash memories by calling the low-level code;
- Finally, thoroughly tests the complete source code of the application.

## 3.2 Porting the Drivers to the Target System

All sensible changes to the software driver that the user has to consider can be found in the header file (wbflash.h). This header file contains the following items required to port the software driver to a new hardware:

**Basic Data Types & Device Type.** Check whether the compiler to be used supports the following basic data types as well as choose the right device by using the appropriate define statement, as described in the header file, and change it where necessary.

```
// If you use a device with "WORD_MODE", please validate following one line.
#define WORD_MODE

// If you use a device with "BYTE_MODE", please validate following one line.
#define BYTE_MODE

// Define using either DQ7 polling bit or DQ6 Toggle bit
// If you use DQ7, please validate DQ7_POLLING line, otherwise DQ6_TOGGLEBIT.
#define DQ7_POLLING
#define DQ6_TOGGLEBIT

#if defined(WORD_MODE)
#define DWIDE    volatile unsigned short
#elif defined(BYTE_MODE)
#define DWIDE    volatile unsigned char
#else
#error "Please define either BYTE__MODE or WORD_MODE !!!"
#endif /* WORD_MODE or BYTE_MODE */
```

**Flash Memory Location.** WFlashBase is the start address of the Flash memories. It must be set according to the target system, in order to access the Flash memories at the right address. This value is used by many functions. The default value is set to 0x50000000 in current target platform (W90N740), and needs to be adjusted appropriately:

```
#define WFlashBase ((DWIDE *) (0x50000000))
```

**Flash Configuration.** Choose the right Flash memory configuration:

```
// If you use a "TOP PARAMETER" device W19B(L)320ST, please validate following one line.
#define TOP_MODE
```

# SOFTWARE DRIVERS FOR W19B(L)320ST/B FLASH MEMORIES



// If you use a "BOTTOM PARAMETER" device W19B(L)320SB, please validate following one line. \*/

```
#define BOTTOM_MODE
```

```
#define Flash32M 33554432 /* 32Mbit */
```

```
#if defined(WORD_MODE)
```

```
#define EXTEND_DATA(x) ((DWIDE)(x))
```

```
#define EXTEND_CMD(x) (((DWIDE)(x)<<8)+(DWIDE)(x))
```

```
#define ADDRESS_OFFSET(x) ((x))
```

```
#define FlashDensity Flash32M/16
```

```
#elif defined(BYTE_MODE)
```

```
#define EXTEND_DATA(x) ((DWIDE)(x))
```

```
#define EXTEND_CMD(x) ((DWIDE)(x))
```

```
#define ADDRESS_OFFSET(x) ((x)<<1 )
```

```
#define FlashDensity Flash32M/8
```

```
#endif
```

```
#if defined(WORD_MODE)
```

```
#define CMD_1stCYCLE_ADDR 0x555
```

```
#define CMD_2stCYCLE_ADDR 0x2AA
```

```
#define CMD_3stCYCLE_ADDR 0x555
```

```
#define CMD_4stCYCLE_ADDR 0x555
```

```
#define CMD_5stCYCLE_ADDR 0x2AA
```

```
#define CMD_6stERASE_CHIP_ADDR 0x555
```

```
#define CMD_CFIQRY_ADDR 0x55
```

```
#elif defined(BYTE_MODE)
```

```
#define CMD_1stCYCLE_ADDR 0xAAA
```

```
#define CMD_2stCYCLE_ADDR 0x555
```

```
#define CMD_3stCYCLE_ADDR 0xAAA
```

```
#define CMD_4stCYCLE_ADDR 0xAAA
```

```
#define CMD_5stCYCLE_ADDR 0x555
```

```
#define CMD_6stERASE_CHIP_ADDR 0xAAA
```

```
#define CMD_CFIQRY_ADDR 0xAA
```

```
#endif
```

The above statement is to define WORD (16-bit) memory bus or BYTE (8-bit) Flash memory connected to system and the related command cycle offset address, density value and so on.

# SOFTWARE DRIVERS FOR W19B(L)320ST/B FLASH MEMORIES



## 3.3 C Library Functions Provided

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

```
/* Change Flash device to Read Array Mode */
void ReadArrayCmd(          /* OUT  : None          */
    DWIDE *address         /* IN   : Flash Address */
);
```

```
/* W19xxx Flash Write with Word/Byte mode */
int W19FlashWrite(         /* OUT  : errcode      */
    DWIDE *address,        /* IN   : Flash Address */
    DWIDE *data            /* IN   : Data          */
);
```

```
/* W19xxx Flash Sector Erase */
int W19SectorErase(       /* OUT  : errcode      */
    DWIDE *address        /* IN   : Flash Address */
);
```

```
/* W19xxx Flash Chip Erase */
int W19ChipErase(         /* OUT  : errcode      */
    DWIDE *address        /* IN   : Flash Address */
);
```

```
/* W19xxxx Flash Read Identifier Code */
void W19ReadIDCodes(      /* OUT  : None          */
    DWIDE *base_add,      /* IN   : Flash base address */
    DWIDE *VendorID,      /* IN   : Pointer of Manufacturer code */
    DWIDE *DeviceID       /* IN   : Pointer of Device code */
);
```

```
/* W19xxx Flash Unlock ByPass Write Cmd */
void W19UnLockByPassCMD( /* OUT  : None          */
    void                 /* IN   : None          */
);
```

# SOFTWARE DRIVERS FOR W19B(L)320ST/B FLASH MEMORIES



```
);

/* W19xxx Flash Unlock ByPass Write Cmd Reset */
void W19UnlockByPassOFF(          /* OUT  : errcode          */
    DWIDE *address                /* IN   : Flash Address   */
);

/* W19xxxx Flash UnLock Bypass Write with Word/Byte mode */
int W19UnlockByPassWrite(        /* OUT  : errcode          */
    DWIDE *address,              /* IN   : Flash Address   */
    DWIDE *data                  /* IN   : Data            */
);

/* W19xxxx Read CFI Query information */
void W19Read_Query (            /* OUT  : None            */
    DWIDE *base_add,            /* IN   : Flash base address */
    CFI_QUERY *query            /* OUT  : Query Information */
);

/* W19xxx Read SecSi Sectory Factory Protect */
DWIDE W19SecSi_CHK (           /* OUT  : None            */
    DWIDE *base_add            /* IN   : Flash base address */
);

/* W19xxx Check Sector Protect status */
DWIDE W19Sector_CHK(           /* OUT  : None            */
    DWIDE *address             /* IN   : Flash Sector address */
);

/* W19xxx Enter SecSi Sectory Region */
void W19SecSi_Entry (          /* OUT  : None            */
    DWIDE *base_add            /* IN   : Flash base address */
);

/* W19xxx Exit SecSi Sectory Region */
```

# SOFTWARE DRIVERS FOR W19B(L)320ST/B FLASH MEMORIES



```
void W19SecSi_Exit (                /* OUT : None                */
    DWIDE *base_add                /* IN  : Flash base address   */
);

/* W19xxx Sector Erase with suspend available as in signal bank device */
void W19SectorEraseSuspend(        /* OUT : errcode             */
    DWIDE *address                /* IN  : Flash Address       */
);

/* W19xxx Sector Erase Suspend Command */
void W19EraseSuspend(             /* OUT : None                */
    void                          /* IN  : None                */
);

/* W19xxx Sector Erase Resume Command & return Erase Status */
void W19EraseResume(             /* OUT : None                */
    void                          /* IN  : None                */
);

/* W19xxx Sector Erase status check after Erase Resume is issued */
int W19SectorEraseCHK(           /* OUT : errcode             */
    DWIDE *address                /* IN  : Flash Address       */
);

/* W19xxx Sector Protect Algorithm */
int W19SectorProtect (           /* OUT : errcode             */
    DWIDE *address                /* IN  : Flash Address       */
);

/* W19xxx Sector UnProtect Algorithm check all sectors */
int W19SectorUnProtectCHKALL (   /* OUT : errcode             */
    unsigned int *arryaddress     /* IN  : Group Address of Erased Block*/
);

/* W19xxx Temporary Sector UnProtect Erase */
```

# SOFTWARE DRIVERS FOR W19B(L)320ST/B FLASH MEMORIES



```
int W19TemUnProtectErase(          /* OUT  : errcode          */
    DWIDE *address                 /* IN   : Flash Address   */
);

/* W19xxx Temporary Sector UnProtect Program */
int W19TemUnProtectWrite(         /* OUT  : errcode          */
    DWIDE *address,               /* IN   : Flash Address   */
    DWIDE *data                   /* IN   : Data             */
);
```

## 3.4 Getting Started

To test the source code in the target system, simply start it by reading from the W19B(L)320ST/B device. To start in a very simple way, write a function main() and include the C file as described below. All the Flash memory functions can be called and executed within the main function. The following example shows a check of the device identifiers (Manufacturer and Device Code), Security Sector Lock/Unlock and a simple Sector Erase command.

```
#include "wbflash.c"
#include "wbflash.h"

int main(int argc, char *argv[])
{
    DWIDE *ptrFlash;
    DWIDE VendorID, DeviceID, Flash_Data;

    ptrFlash= (DWIDE *) WFlashBase;
    W19ReadIDCodes(ptrFlash, &VendorID, &DeviceID);
    printf("\nThe Vendor ID Code is %02x\n", VendorID&0xff);
    printf("The Device Code is %02x\n", DeviceID&0xff);

    Flash_Data=W19SecSi_CHK(ptrFlash);
    printf("The SecSi Factory protect value:%x(19:Un-locked;99:locked)\n", Flash_Data&0xff);
    ReadArrayCmd(ptrFlash);
    return 0;
}
```

# SOFTWARE DRIVERS FOR W19B(L)320ST/B FLASH MEMORIES



Reading the Manufacturer and Device Codes and checking that if they are correct. If this function works correctly, the other functions are very likely to work well too. However, all the functions should be tested thoroughly.

```
#include "wbflash.c"
#include "wbflash.h"

int main(int argc, char *argv[])
{
    DWIDE *ptrFlash;
    Int ErrorCode;
    DWIDE Flash_Data;

    ptrFlash=(DWIDE *) WflashBase ;
    ErrorCode=W19SectorErase(ptrFlash)
    if(ErrorCode) printf("Sector Erase Error. %d...%n", ErrorCode);
    else printf("Sector Erase at %x is done!!!...%n", ptrFlash);

    Flash_Data= *ptrFlash;
    Printf( "Read data is %x%n" , Flash_Data);
    return 0;
}
```

If it is erased, only FFh or FFFFh data should be read. The provided software implement the full set of the W19B(L)320ST/B functionalities. When an error occurs the software simply returns the error message.

## 4. CONCLUSION

The W19B(L)320ST/B 3V(3.3V) Flash memories are ideal products for 8-bits or 16-bits embedded and other computer systems. They can be easily interfaced to microprocessors and driven with simple software drivers written in the C language. A simple recompiling with a new software driver is all that is needed to control a new device.

# SOFTWARE DRIVERS FOR W19B(L)320ST/B FLASH MEMORIES



## 5. VERSION HISTORY

VERSION	DATE	PAGE	DESCRIPTION
A1	Oct. 28, 2003	-	Initial Issued



**Headquarters**  
No. 4, Creation Rd. III,  
Science-Based Industrial Park,  
Hsinchu, Taiwan  
TEL: 886-3-5770066  
FAX: 886-3-5665577  
<http://www.winbond.com.tw/>

**Taipei Office**  
9F, No.480, Rueiguang Rd.,  
Neihu District, Taipei, 114,  
Taiwan, R.O.C.  
TEL: 886-2-8177-7168  
FAX: 886-2-8751-3579

**Winbond Electronics Corporation America**  
2727 North First Street, San Jose,  
CA 95134, U.S.A.  
TEL: 1-408-9436666  
FAX: 1-408-5441798

**Winbond Electronics Corporation Japan**  
7F Daini-ueno BLDG, 3-7-18  
Shinyokohama Kohoku-ku,  
Yokohama, 222-0033  
TEL: 81-45-4781881  
FAX: 81-45-4781800

**Winbond Electronics (Shanghai) Ltd.**  
27F, 2299 Yan An W. Rd. Shanghai,  
200336 China  
TEL: 86-21-62365999  
FAX: 86-21-62365998

**Winbond Electronics (H.K.) Ltd.**  
Unit 9-15, 22F, Millennium City,  
No. 378 Kwun Tong Rd.,  
Kowloon, Hong Kong  
TEL: 852-27513100  
FAX: 852-27552064

*Please note that all data and specifications are subject to change without notice.  
All the trade marks of products and companies mentioned in this data sheet belong to their respective owners.*