AN2311
Application note

Software drivers for the M29W640FT and M29W640FB Flash memories

Introduction

This application note describes library source code in C for the M29W640FT and M29W640FB Flash memories using the new STFL-I software driver interface V2.

The M29W640FT and M29W640FB memories will be referred to as M29W640F throughout the document unless otherwise specified.

The M29W640F can be configured to operate in 8-bit or 16-bit bus mode.

The beta release of the source code is available from the internet site *http://www.st.com* or from your STMicroelectronics distributor. The 2311.c and 2311.h files contain libraries for accessing the M29W640F Flash memories.

This application note also includes an overview of the programming model for the M29W640F. 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 individual target hardware. The source code contains comments throughout, explaining how it is used and why it has been written the way it has.

This application note does not replace the M29W640F datasheet. It refers to it throughout, and it is necessary to have a copy of the datasheet to follow some of the explanations. The software, supplied with this documentation, has been tested on a target platform, and is usable in C and C++ environments. It is small in size and can be applied to any target hardware.

Contents

1 Programming model

The M29W640F is a 64 Mbit (8 Mbit x8 or 4 Mbit x16, Boot Block) Flash memory. It can be electrically erased at block level and programmed in-system through special coded command sequences on most standard microprocessor buses. The device features an asymmetric block architecture. The M29W640F has an array of 135 blocks: 8 parameter blocks of 8 Kbytes and 127 main blocks of 64 KBytes. The M29W640FT memory has the Parameter Blocks at the top of the memory address space while the M29W640FB locates them at the bottom. Each block can be erased separately. Erase can be suspended in order to either read from or program to any other block, and then resumed. Each block can be programmed and erased over 100,000 cycles.

The blocks can be protected in groups, preventing any accidental programming or erasure. The memories offer a Temporary Unprotect function controlled by the Reset/Block Temporary Unprotect pin $\overline{RP}$. Refer to the datasheet for more details on the protection/unprotection mechanism.

Program and Erase commands are written to the Command Interface of the memories. An on-chip Program/Erase Controller (P/E.C.) takes care of the timings necessary for program and erase operations. The end of a program or erase operation can be detected and any error conditions identified. The command set required to control the memory is consistent with JEDEC standards.

The M29W640F offers five features to improve the programming throughput:

- The Double Word Program command - used to write two adjacent Words in parallel (16 bit mode),
- The Quadruple Word Program command - used to write a page of four adjacent Words in parallel (16 bit mode),
- The Double Byte Program command - used to write two adjacent Bytes in parallel (8 bit mode)
- The Quadruple Byte Program Command used to write a page of four adjacent Bytes in parallel (8 bit mode)
- The Octuple Byte Program Command used to write a page of eight adjacent Bytes in parallel (8 bit mode).

Note: 1 Data with the current CPU data bus width will be referred to as elements throughout the document unless otherwise specified. Due to the flexibility of the STFL-I software driver, the size of an element depends on the current configuration (see Section 2.2 for a description of the User Change Area).

1.1 Bus operations and commands

Most M29W640F 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 data stored or the device behavior. Only some special write operation sequences are recognized as commands by the M29W640F. The various commands recognized by the devices are listed in the Command Tables provided in the corresponding datasheet. The main commands can be classified as follows:

1. Read/Reset
2. Auto Select
3. Block Erase and Chip Erase
4. Program
5. Fast Program: Double Word Program (16 bit mode), Quadruple Word Program (16 bit mode), Double Byte Program (8 bit mode), Quadruple Byte Program (8 bit mode) and Octuple Byte Program (8 bit mode)
6. Program/Erase Suspend and Resume
7. Read CFI Query
8. Block Protect and Chip Unprotect
9. Enter/Exit Extended Block
10. Unlock Bypass, Unlock Bypass Program, Unlock Bypass Reset

1.1.1 Read/Reset command

The Read/Reset command returns the M29W640F to Read mode where the device behaves as a ROM. In this mode, a read operation outputs the data stored at the specified device address onto the data bus.

1.1.2 Auto Select command

The Auto Select command places the device in a mode that allows the user to read the Manufacturer Code, the Device Code, the Block Protection Status and the Extended Memory Block Verify Code. These information are accessed by reading at different addresses while the device is in Auto Select mode.

1.1.3 Block Erase and Chip Erase commands

The Block Erase is used to set to ‘1’ all the bits contained in one block, while the Chip Erase command erases the entire memory. Previously stored data will be lost.

The erase commands take longer to execute than the other commands because an entire block (Block Erase) or the whole memory (Chip Erase) is erased at once.

Attempts to erase a protected block will not generate any error but the contents of the block will remain unchanged.

1.1.4 Program commands

The program commands are used to modify the data stored at specified memory addresses.

The M29W640F features Program command that allows to program a single value to one address in the memory array, and Fast Program commands which improve the programming throughput by writing several adjacent Bytes or Words in parallel.

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 commands, the command will be executed and no error will be signalled but the bit will remain unchanged. It may therefore be necessary to erase the block before programming to addresses within it.

Programming modifies a single element at a time. There are two solutions to program larger amounts of data:

- Program one element at a time by issuing a Program command, wait for the command to complete, issue the next Program command, and so on.
- Issue a Fast Program command: Double Byte Program (only in 8 bit mode), Quadruple Byte Program (only in 8 bit mode), Octuple Byte Program (only in 8 bit mode), Quadruple Word Program (only in 16 bit mode), or Double Word Program (only in 16 bit mode).

1.1.5 Program/Erase Suspend and Resume commands

Issuing the Erase Suspend command during an erase operation will temporarily place the M29W640F in Erase Suspend mode. While an erase operation is suspended, the blocks not being erased can be read or programmed as if in the reset state of the device. This allows the user to access information stored in the M29W640F immediately without having to wait for the erase operation to complete. The erase operation is resumed when the device receives the Erase Resume command.

Issuing the Program Suspend command during a program operation will temporarily place the M29W640F in Program Suspend mode. While a program operation is suspended the blocks not being programmed can be read as if in the reset state of the devices. This allows the user to access information stored in the M29W640F immediately without having to wait for the program operation to complete. The program operation is resumed when the device receives the Program Resume command.

1.1.6 Read CFI Query command

The Read CFI (Common Flash Interface) Query command allows the user to identify the number of blocks in the Flash memory and the block addresses. The interface also contains information related to the typical and maximum Program and Erase times. This allows the user to implement software timeouts and prevents waiting indefinitely for a defective Flash memory to finish programming or erasing. For further information about the CFI, please refer to the CFI specification available from the internet site (*http://www.jedec.org*) or from your STMicroelectronics distributor.

1.1.7 Block Protect and Chip Unprotect commands

Blocks can be protected against accidental program and erase operations that could alter their contents. Each block belongs to a group of blocks (see the device datasheet for details). The user can protect or unprotect the blocks only by groups. To perform a group of blocks Protect/Unprotect command, a high voltage level on the Reset/Block Temporary Unprotect pin, $\overline{RP}$, is required.

The whole chip can be unprotected to allow the data inside the blocks to be changed.

1.1.8 Enter/Exit Extended Block command

The M29W640F has an extra block, the Extended Memory Block, of 256 Bytes. The Extended Memory Block can only be accessed using the Enter Extended Block command. In Extended Block mode, bus read and write operations to the Boot Block addresses access the Extended Memory Block, and it is not possible to access the Boot Blocks.

To access the Boot Blocks, the Extended Block mode must be exited by issuing the Exit Extended Block command.

The Extended Memory Block cannot be erased: in that, it is comparable to a one-time programmable (OTP) memory.

1.1.9 Unlock Bypass command

The Unlock Bypass command is used in conjunction with the Unlock Bypass Program command to program the memory faster than with the standard program commands. Once the Unlock Bypass command has been issued, the memory accepts the Unlock Bypass Program command and the Unlock Bypass Reset command. The memory can be read as if in Read mode.

When V_{PP} is applied to the V_{PP}/Write Protect pin, the memory automatically enters the Unlock Bypass mode and the Unlock Bypass Program command can be issued immediately.

1.1.10 Unlock Bypass Program command

The Unlock Bypass Program command programs one address in the memory array at a time. The Program operation using the Unlock Bypass Program command behaves identically to the Program operation using the Program command.

An Unlock Bypass command must be issued previously, to place the device in unlock Bypass mode.

1.1.11 Unlock Bypass Reset command

The Unlock Bypass Reset command can be used to return to Read/Reset mode from Unlock Bypass mode. Read/Reset command does not exit from Unlock Bypass mode.

1.2 Status Register

During program or erase operations a bus read operation will output the contents of the Status Register. The Status Register provides valuable information about the latest program or erase operation. The Status Register bits are described in the Status Register Bits Tables provided in the M29W640F datasheet. They are mainly used to determine when programming or erasing is complete and whether the operation was successful or not.

The completion of the program or erase operation is indicated either by the Data Polling bit (Status Register bit DQ7) or by the Toggle bit (Status Register bit DQ6). The software driver functions use the Data Toggle Flowchart (see the Data Polling and Data Toggle flowcharts in the datasheet for details).

Programming or erasing errors are indicated by the Error bit (Status Register bit DQ5) going High. In the case of a failure, the command will not complete and subsequent read operations will output the Status Register bits until a Read/Reset command is issued.

1.3 A detailed example

The Command Tables provided in the M29W640F datasheet describe the Bus Write sequences recognized as valid commands by the Program/Erase Controller.

As an example, consider the programming of the value 9465h to the address 03E2h for the device in 16 bit mode. The required C language sequence is the following:

```
*(uword*)(0x0555) = 0x00AA;
*(uword*)(0x02AA) = 0x0055;
*(uword*)(0x0555) = 0x00A0;
*(uword*)(0x03E2) = 0x9465;
```

where uword is defined as the following 16-bit value:

```
typedef unsigned short uword
```

The example assumes that the address 0000h in the M29W640F 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.

While the device is programming to the specified address, read operations will access the Status Register. Status Register bit DQ5 going High (set to ’1’) means that an error has occurred and the operation has failed. Status Register bit DQ6 toggles during programming and DQ7 is the complement of the bit being programmed.

Once programmed, the address 03E2h cannot be reprogrammed reliably until an erase operation is issued to erase the entire block.

2 How to use the software driver

2.1 General considerations

The software drivers described in this application note are intended to simplify the process of developing application code in C for the STMicroelectronics M29W640F Flash devices.

This software driver supports the new STFL-I interface that is implemented in all software drivers. As a result, future device changes will not necessarily lead to code changes in application environments.

Note that, to meet compatibility requirements, the Standard Software Driver STFL-I interface allocates numbers to each block in a Flash memory starting from 0 (block 0 always has address offset 0) and up to the highest-address block in the device. Block numbers may be described differently in the datasheets. For example, in a Flash device containing 64 blocks, the STFL-I interface will always refer to the block with address offset 0 as block number 0, and to the last block, as block number 63.

This application note gives a summary description of the new STFL-I interface; a complete description is available from your STMicroelectronics distributor.

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 resulting source code is both simpler and easier to maintain.

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

- Hardware-specific bus operations
- Low-level code
- High-level code written by the user

The low-level code requires hardware-specific Read and Write Bus operations in C in order to communicate with the M29W640F. 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. Another consequence is that the user's high-level code is easier to apply to other STMicroelectronics Flash memories.

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

1. 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.
2. Then write the high-level code for the desired application. The application will access the Flash memories by calling the low-level code.
3. Finally thoroughly test the complete source code of the application.

2.2 Porting the drivers to the target system (user change area)

All sensible changes to the software driver that the user has to consider can be found in the header file. A designated area called the "User Change Area" contains the following items required to port the software driver to new hardware:

2.2.1 Basic data types

Check whether the compiler to be used supports the following basic data types, as described in the source code, and change it where necessary.

```
typedef unsigned char ubyte; (8 bits)
typedef char byte; (8 bits)
typedef unsigned short uword; (16 bits)
typedef short word; (16 bits)
typedef unsigned int  udword; (32 bits)
typedef int dword; (32 bits)
```

2.2.2 Device type

Choose the right device by using the appropriate define statement:

```
#define USE_M29W640FT_8
#define USE_M29W640FT_16
#define USE_M29W640FB_8
#define USE_M29W640FB_16
```

2.2.3 Flash memory location

BASE_ADDR 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 the FlashRead() and FlashWrite() functions. The default value is set to zero, and needs to be adjusted appropriately:

```
#define BASE_ADDR ((volatile uCPUBusType*)0x00000000)
```

2.2.4 Flash configuration

Choose the right Flash memory configuration:

```
#define USE_16BIT_CPU_ACCESSING_2_8BIT_FLASH
```

This define statement supports a board configuration containing a CPU with an external 16-bit memory bus with two 8-bit Flash memories connected to it.

```
#define USE_16BIT_CPU_ACCESSING_1_16BIT_FLASH
```

This define statement supports a board configuration containing a CPU with an external 16-bit memory bus with a single 16-bit Flash memory connected to it.

```
#define USE_32BIT_CPU_ACCESSING_4_8BIT_FLASH
```

This define statement supports a board configuration containing a CPU with an external 32-bit memory bus with four 8-bit Flash memories connected to it.

```
#define USE_32BIT_CPU_ACCESSING_2_16BIT_FLASH
```

This define statement supports a board configuration containing a CPU with an external 32-bit memory bus with two 16-bit Flash memories connected to it.

2.2.5 Timeout

Timeouts are implemented in the loops of the code to provide an exit for operations that would otherwise never terminate. There are two possibilities:

1. The ANSI library functions declared in "time.h" exist

   If the current compiler supports "time.h" the define statement TIME_H_EXISTS should be activated. This will prevent any change in timeout settings due to the performance of the current evaluation hardware.

   ```
   #define TIME_H_EXISTS
   ```

2. The Option COUNT_FOR_A_SECOND

   If the current compiler does not support "time.h", the define statement TIME_H_EXISTS cannot be used. In this case the COUNT_FOR_A_SECOND value has to be defined so as to create a one-second delay. For example, if 100,000 repetitions of a loop are needed to give a time delay of one second, then COUNT_FOR_A_SECOND should have the value 100,000.

   ```
   #define COUNT_FOR_A_SECOND (chosen value)
   ```

Note that this delay depends on the hardware performance and should therefore be updated every time the hardware is changed.

This driver has been tested with a certain configuration and other target platforms may have other performance data. It may therefore be necessary to change the COUNT_FOR_A_SECOND value. It is up to the user to implement the correct value to prevent the code from timing out too early and allow correct completion.

2.2.6 Additional subroutines `#define VERBOSE`

In the software driver, the VERBOSE define statement is used to activate the FlashErrStr() function, in order to generate a text string describing the return code from the Flash memory.

2.2.7 Additional considerations

The access timing data for the Flash memories can sometimes be problematic. It may be necessary to change the FlashRead() and FlashWrite() functions if they are not compatible with the timings of the target hardware. These problems can be solved with a logic state analyzer.

The programmer needs to take extra care when the device is accessed during an interrupt service routine. When the device is in Read mode, interrupts can freely read from the device. Interrupts that do not access the device may be used during all functions.

2.3 C library functions provided

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

Flash() is used to access all device functions, and acts as the main Flash memory interface. This function is available on all software drivers written in the new STFL-I format and should be used exclusively. Any functionality unsupported by the Flash memories can be detected and malfunctions can thus be avoided.

Note that the other functions are listed to offer a second-level interface when enhanced performance is required. Within the STFL-I the functions are always used in the same way, which means that the function interface (names, return codes, parameters, data types) remains unchanged regardless of the Flash memory.

FlashBlockErase() is used to erase one block in the device. A block cannot be erased when it is protected. Attempting to do so generates no error. During this operation only the Erase Suspend and the Read/Reset commands are not ignored.

FlashCheckBlockProtection() is used to check whether a block is protected.

FlashCheckCompatibility() is used to check the Flash memory for compatibility.

FlashChipErase() is used to erase the entire device. Protected blocks will be not erased and no error will occur.

FlashChipUnprotect() is used to unprotect all the blocks in the Flash memory. Once all the blocks are unprotected, the data contained in all the blocks can be entirely erased or new data can be programmed. This function uses the In System Technique to unprotect the memory and requires a high voltage level on the Reset/Blocks Temporary Unprotect pin, $\overline{RP}$.

FlashDoubleProgram() (available in both 8 bit and 16 bit mode) is used to program two consecutive elements to the Flash memory. Only previously erased elements can be programmed reliably. For the function to operate properly $V_{PP} = V_{PPH}$ is required.

FlashEnterExtendedBlock() is used to access the additional 256-Byte Extended Memory Block) instead of accessing the Boot Blocks. In Extended Block mode all bus Read and Write operations to the Boot Blocks addresses access the Extended Memory Block, and the Boot Blocks are not accessible.

FlashErrorStr() is used to generate a text string describing the detected error.

FlashExitExtendedBlock() is used to exit from the Extended Block mode and return the device to Read mode. Once the command has been issued, the Extended Memory Block is no longer accessible.

FlashGroupProtect() is used to protect a selection of blocks called a group. This function uses the 'In System Technique' to protect the block and requires a high voltage level on the Reset/Blocks Temporary Unprotect pin, $\overline{RP}$.

FlashMultipleBlockErase() can be used to erase a list of one or more blocks. Only unprotected blocks can be erased. Attempting to erase protected blocks will leave their contents unchanged and produce no error. During the erase operation, all commands will be ignored except for the Erase Suspend and Read/Reset commands.

FlashProgram() is used to program arrays of elements to the Flash memories. Only previously erased elements can be programmed reliably because it is not allowed to set a bit from '0' to '1'. Protected blocks cannot be programmed.

FlashQuadProgram() (available in both 8 bit and 16 bit mode) is used to program four consecutive elements to the Flash memory. Only previously erased elements can be programmed reliably. For the function to work properly $V_{PP} = V_{PPH}$ is required.

FlashOctProgram() (available only in 8-bit mode) is used to program eight consecutive elements to the Flash memory. This procedure automatically aligns address to 8-Bytes boundary. For the function to work properly $V_{PP} = V_{PPH}$ is required.

FlashReadCfi() is used to check that the Common Flash Interface is supported and then to read the CFI data at the specified offset.

FlashReadDeviceId() is used to read the Device Code of the Flash memory.

FlashReadExtendedBlockVerifyCode() is used to read the lock/unlock status of the Extended Memory Block.

FlashReadManufacturerCode() is used to read the Manufacturer Code of the Flash memory.

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

FlashResume() is used to resume the erase operation being suspended.

FlashSingleProgram() is used to program a single element. Only a previously erased element can be programmed reliably because it is not allowed to set a bit from ‘0’ to ’1’.

FlashSuspend() is used to suspend the erase operation in progress.

FlashUnlockBypassProgram() is used to program the memory faster than with the standard program commands. This function can be used only if $V_{PP} = V_{PPH}$.

FlashUnlockBypassReset() is used to return to Read/Reset mode from the Unlock Bypass mode. The functions provided in the software library rely on the user implementing the hardware-specific bus operations, and on access timings, to communicate properly with the Flash memory. If changes to the software driver are necessary, only the two following functions need to be changed:

- **FlashRead()** is used to read a value from the Flash memories.
- **FlashWrite()** is used to write a value to the Flash memories.

2.4 Getting started (example quicktest)

To test the source code in the target system, simply start by reading from the M29W640F device. If it is erased, only FFFFh data should be read. Then read the Manufacturer and Device codes and check that they are correct. If these functions work, the other functions are very likely to work too. However, all the functions should be tested thoroughly.

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 (Device Code, Manufacturer Code) and a simple Block Erase command.

```
#include "c2311.c"
void main(void) {
ParameterType fp;    /* Contains all Flash Parameters */
ReturnType rRetVal; /* Return Type Enum */
Flash(ReadManufacturerCode, &fp);
printf("Manufacturer Code: %08Xh\r\n",
fp.ReadManufacturerCode.ucManufacturerCode);
Flash(ReadDeviceId, &fp);
printf("Device Code: %08Xh\r\n",
fp.ReadDeviceId.ucDeviceId);
fp.BlockErase.ublBlockNr = 10;    /* block number 10 will be erased
*/
rRetVal = Flash(BlockErase, &fp); /* function execution */
} /* EndFunction Main */
```

3 Software limitations

The described software does not implement the full set of the M29W640F functionalities. When an error occurs the software simply returns the error message. It is up to the user to decide what to do. He can either try the command again or, if necessary, replace the device.

4 Conclusion

The M29W640F 3V supply Flash memories are ideal products for embedded and other computer systems. They can be easily interfaced to microprocessors and driven with simple software drivers written in the C language.

The new STFL-I interface allows changeable Flash memory configurations, compiler-independent data types and a unique access mode for a broad range of Flash memory devices.

Moreover, applications supporting the new STFL-I standard can implement any Flash memory device with the same interface without any code change. A simple re-compiling with a new software driver is all that is needed to control a new device.

5 Revision history

Table 1. Document revision history

Date	Revision	Changes
20-Mar-2006	1	Initial release.