ST7 FAMILY

FLASH Programming

REFERENCE MANUAL

Rev. 2

May 2005

USE IN LIFE SUPPORT DEVICES OR SYSTEMS MUST BE EXPRESSLY AUTHORIZED.

STMicroelectronics PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF STMicroelectronics. As used herein:

1. Life support devices or systems are those which (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided with the product, can be reasonably expected to result in significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can reasonably be expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

ST7 FAMILY

FLASH PROGRAMMING REFERENCE MANUAL

INTRODUCTION

This manual describes how to program FLASH memory of an ST72Fxxx microcontroller. For implementation details such as register or stack top address, refer to the product datasheet.

Three FLASH technologies are available in the ST7 family with different programming methods. Each ST7 FLASH microcontroller is associated with one of these three technologies as shown in the Flash Technology Types table in the Flash Programming Quick Reference Manual.

- XFlash (Extended): FLASH memory based on a EEPROM technology (see Section 2)
- HDFlash (High Density): FLASH memory based on FLASH technology (see Section 3)
- CFlash (ST72Cxxx): FLASH memory based on a EEPROM technology and not described in this document (see AN1179 for more details)

The **In-Circuit Programming (ICP)** method is used to update the entire contents of the FLASH memory (including option bytes) while the user application is not running. It uses **In-Circuit Communication (ICC)** protocol which enables an ST7 microcontroller to communicate with an external controller such as a PC with only 4 wires including V_{SS} (see “ICC Protocol” reference manual for more details).

For ST72Cxxx devices, the In-Situ programming (ISP) method is used instead of ICP one. ISP is not described in this document (see AN1179 for more details).

In contrast to the ICP method, **In-Application programming (IAP)** uses any communication interface supported by the microcontroller (I/Os, SPI, SCI, USB, CAN...). IAP has been implemented for users who want their application software to update itself by re-programming the FLASH memory during execution. The main advantage of IAP is its ability to re-program the FLASH memory when the chip has already been soldered on the application board and while the user application is running. Nevertheless, part of the FLASH memory has to be previously programmed using ICP.

Related Documentation

- ICC Protocol Reference Manual
- Flash Programming Quick Reference Manual[1]
- Debug Module Reference Manual

[1]The Quick Reference manual provides device-specific tables included in the previous revision of the Flash Progamming and ICC reference manuals.

Rev. 2

Table of Contents

Table of Contents

GLOSSARY

This chapter gives a brief definition of all new acronyms and names linked to the ICP and IAP methods as a quick reference:

- **XFlash (Extended):** Extended FLASH memory is based on EEPROM technology. The XFlash provides extended features such as byte per byte re-programming (by means of the byte erasing) and data EEPROM capability. The XFlash devices are available between 1k and 16kbytes.
- **HDFlash (High Density):** High density FLASH memory is based on FLASH technology. The high density of the HDFlash cell is used for devices with 4k up to 60kbytes FLASH memory. This HDFlash is programmed byte per byte but erased by sector.
- **CFlash:** CFlash memory is based on EEPROM technology and is implemented in ST72Cxx devices. CFlash programming is not described in this manual but in the AN1179 application note.
- **IAP (In-Application Programming)**: The IAP is the ability to re-program the FLASH memory of a microcontroller while the device is already plugged-in to the application and the application is running.
- **ICC Mode/Protocol (In-Circuit Communication):** The ICC is either an ST7 mode or a software protocol stored by ST7 microcontrollers in the System Memory. The ICC software is used to download programs into RAM and execute them. It can also write (registers or RAM) or read any part of the memory space and jump to any memory address. The ICC protocol is used for ICP. It is accessed by entering ICC mode after a dedicated reset sequence.
- **ICP (In-Circuit Programming):** The ICP is the ability to program the FLASH memory of a microcontroller using ICC protocol while the device is plugged-in to the application.
- **Driver:** A driver is a control program defined by the application developer. A driver is used to manage the allocation of system resources to start application programs. In this document two drivers are described: ICP driver and IAP driver.
- **ICC Monitor:** The ICC monitor is the ST7 embedded software which manages the ICC protocol. It is located in System Memory.
- **USER Mode:** The USER mode is the standard user application running mode in the ST7. It is entered by means of the $\overline{\text{RESET}}$ pin and without any specific reset sequence.
- **Embedded Commands:** The Embedded Commands are the software drivers used by the USER application to program or erase HDFlash devices.
- **System Memory:** The System Memory is a write-protected part of the memory which contains the ICC monitor. The HDFlash System Memory also contains the Embedded Command software. The XFlash System Memory is accessible when the USER application is executed however this is not the case for HDFlash System Memory.

1 MEMORY PROTECTION STRATEGY

The ST7 FLASH protection strategy is based on 3 different types of protection:

- **Read-out Protection:** The user is able to protect the program and data stored in the FLASH memory from being read. To do so, the corresponding option bit must be programmed in the option byte using the ICP method.
- **Recovery Protection:** During the programming phase, the device can always recover in a stable and known state by executing a reset sequence. During ICP or IAP, this is guaranteed by the fact that the programming software driver is always write protected.
- **Programming Mode Access Protection:** This feature is used to protect the application against unwanted re-programming of the FLASH memory, disabling the FLASH Control/ Status Register (FCSR). Access to the FCSR is enabled by writing a specific key sequence (see Section 1.3 REGISTER ACCESS SECURITY SYSTEM (RASS)). For HDFlash memories (which are dual voltage), the Programming Mode Access Protection is also guaranteed by the fact that an external V_{PP} supply is needed to write or erase memory.

1.1 READ-OUT PROTECTION

The read-out protection can be removed only if the entire FLASH memory is programmed or erased. Readout protection, when selected provides a protection against program memory content extraction and against write access to Flash memory. Even though no protection can be considered as totally unbreakable, the feature provides a very high level of protection for a general purpose microcontroller. Of course a user application that permits the user to dump the FLASH memory contents will render this read-out protection useless. Table 1 describes the capability of the FLASH memory sectors versus the ST7 modes and read-out protection settings.

Caution: When the read-out protection is set, STMicroelectronics is not able to disable this option without erasing the FLASH content.

Table 1. ST7 FLASH Memory Read-out Protection Summary

ST7 Modes →	USER mode (including IAP)		ICC mode	
FLASH read-out protection →	**Disabled**	**Enabled**	**Disabled**	**Enabled**
FLASH Sector 0	Read / Execute		Read / Write / Execute	Protected (not accessible)
FLASH Sector 1 or 2	Read / Write / Execute			

1.2 RECOVERY PROTECTION

During either ICP or IAP sessions, the programming software driver is always write/erase protected. When entering ICC mode for ICP, the System Memory area, which contains the ICC monitor, cannot be modified. For IAP, sector 0 (where the IAP driver must be implemented) is write/erase protected.

– In ICC mode, this protection guarantees the ability to restart the ICP phase from the beginning by a device reset after an undesired programming or program counter corruption.

– Using IAP, the application RESET vector and the IAP driver software are in the write/erase protected sector 0. Thanks to these conditions, the USER application is able to manage the recovery protection in sector 0.

1.3 REGISTER ACCESS SECURITY SYSTEM (RASS)

The purpose of the RASS is to prevent the application from unintentionally accessing the FLASH control register which could cause involuntary re-programming of the FLASH memory. Even if the ICP or IAP is suddenly activated, no programming can be done unless the RASS procedure is completed.

After a device reset, the FLASH Control/Status Register (FCSR) is available only in Read mode. To be able to write to this register, a sequence of two hardware keys fixed by STMicroelectronics must be sequentially written to the FCSR address. If a write command that is other than the correct hardware key sequence is attempted, the FCSR register cannot be written to until the next device reset.

– When the FCSR register is written to for the first time after a reset, the data bus content is not latched into the FCSR register, but compared to the first hardware key value (56h).

– If the key available on the data bus is incorrect, then the FCSR register will remain locked until the next reset. Any new write commands sent to this address will be discarded.

– If the first hardware key is correct when the FCSR register is written to for the second time, the data bus content is still not latched into the FCSR register, but compared to the second hardware key value (AEh).

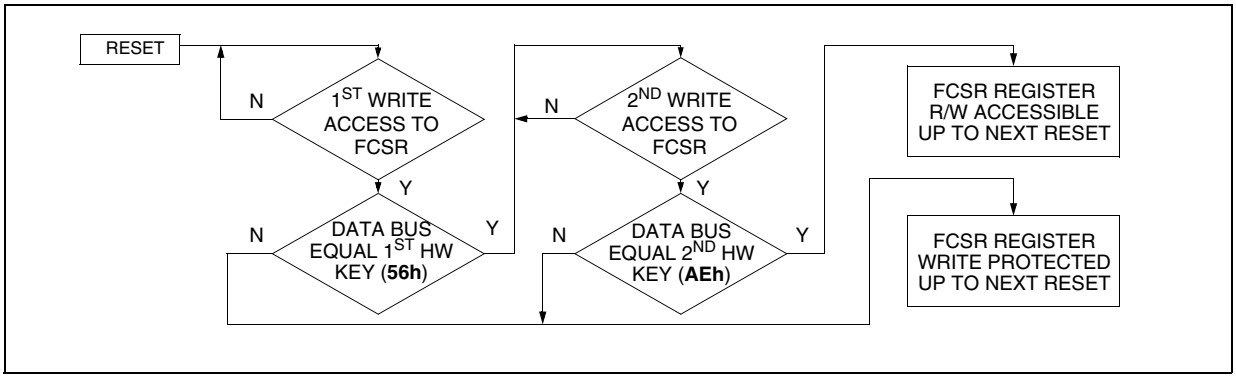
– If the key available on the data bus is incorrect, then the FCSR register will remain locked until the next reset. Any new write commands sent to this address will be discarded.

– If the second hardware key is correct, the FLASH register access is unlocked until the next reset. This means that the next write command sent to the FLASH register will be taken into account.

– There are two different ways of checking if the FCSR register is locked or unlocked, depending on whether the register is associated with XFlash (see Section 2.2.1.4) or HDFlash (see Section 3.2.5.2) memory.

The serial RASS Principle is shown in Figure 1.

Figure 1. RASS Principle



To increase the reliability of the system, the software sequence which writes the hardware keys must not be stored in the program memory. Both hardware keys must always be loaded externally (via I/O ports, SCI, etc...). This security feature prevents any wrongful access after a program counter corruption.

Note: The hardware key values are atypical values of the FCSR register to ensure that the RASS sequence is not in the memory area.

2 PROGRAMMING XFlash MCUS

2.1 INTRODUCTION

This section describes how to program the ST7 Single-Voltage XFlash microcontrollers (MCUs). See ST7 device list in section on page 3.

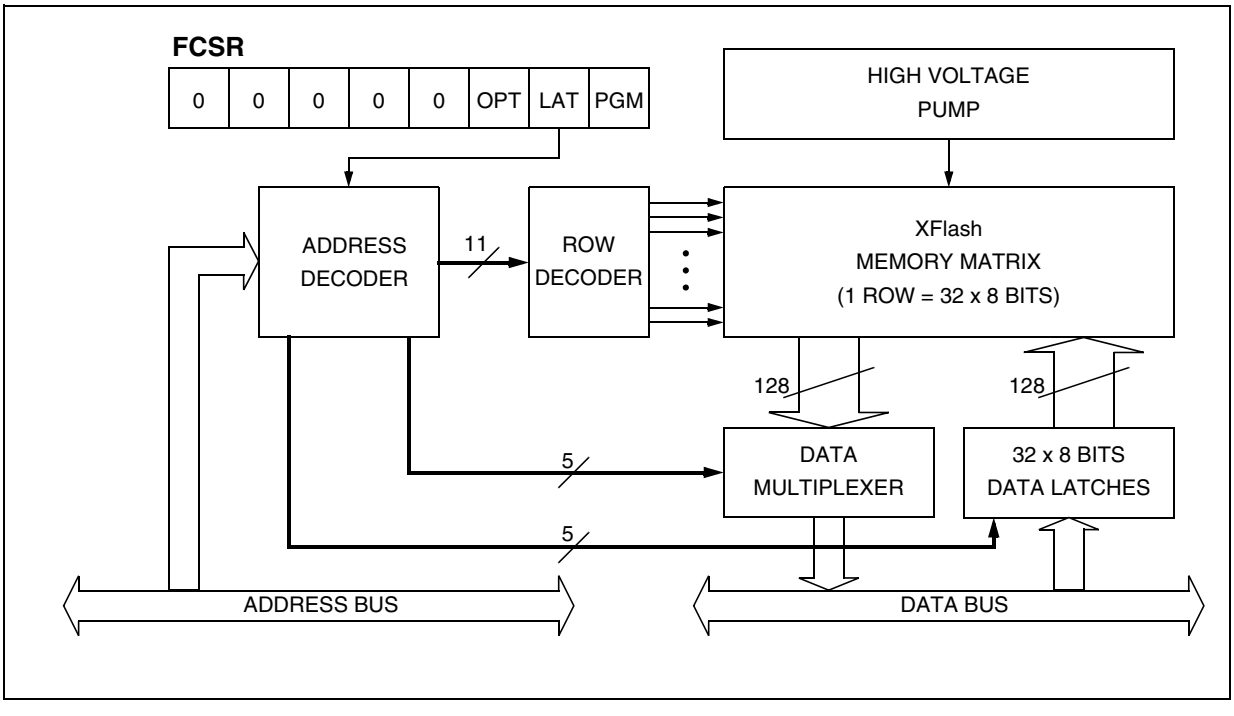
2.2 XFlash PROGRAMMING

2.2.1 XFlash Programming Organization

The XFlash program memory is organized in 8-bit wide memory cells which can be used for storing both code and data constants. It is mapped in the upper part of the ST7 addressing space and includes the reset and user interrupt vector area. The XFlash memory can be erased and programmed byte per byte. Up to 32 bytes can be programmed in the same erase and programming cycle. The Data EEPROM (featured on some devices) can be programmed the same way as the XFlash memory using either ICP or IAP sequences. The XFlash sector 0 and option bytes can be modified using ICP only.

XFlash programming is controlled by a single FLASH Control/Status Register (FCSR) described in Section 2.2.1.5.

Figure 1. XFlash Memory Block Diagram



Note: When the data EEPROM is not available in a ST7 device, it can be emulated by the XFlash memory with some restrictions (see APPENDIX 4 on page 53 for more details).

XFlash memory Read/Write access modes are controlled by the LAT bit in the FCSR register as shown in Figure 2.

- Read Operation (LAT=0): The XFlash memory program can be read as a normal ROM location when the LAT bit of the FCSR register is cleared.
- Write Operation (LAT=1): To enter Write mode, the LAT bit must be set by software (the PGM bit remains cleared). When a Write access to the XFlash area occurs, the value is latched by the 32 data latches.

When the PGM bit is set by software, all the bytes previously written in the data latches (up to 32) are programmed in the XFlash cells. The 11 most significant bits of the address (row) are given by the last data latch write sequence. This means that the selected row is defined by the last latched byte address, previous latched data bytes are taken into account within this row. Therefore, to prevent incorrect programming, the user must verify that all the bytes written between two programming sequences have the same high address: only the five least significant bits (LSBs) of the address can change.

At the end of the programming cycle, the PGM and LAT bits are cleared simultaneously by hardware.

This write operation sequence is illustrated in Figure 3. A source code example is given in APPENDIX 2 on page 44.

Caution: As soon as the LAT bit is set and up to the end of programming, the XFlash contents cannot be executed. So the programming sequence has to be executed from another memory (RAM, Data EEPROM...).

Notes:

1. Care should be taken during the programming cycle. Writing to the same memory location will over-program the memory (logical AND between the two write accesses) because the data latches are only cleared at the end of the programming cycle or when LAT bit is cleared. Latched data cannot be read. This note is illustrated in Figure 2.
2. If the LAT bit is not set, then the PGM bit can not be set (it is kept cleared).

Figure 2. XFlash Programming Flowchart

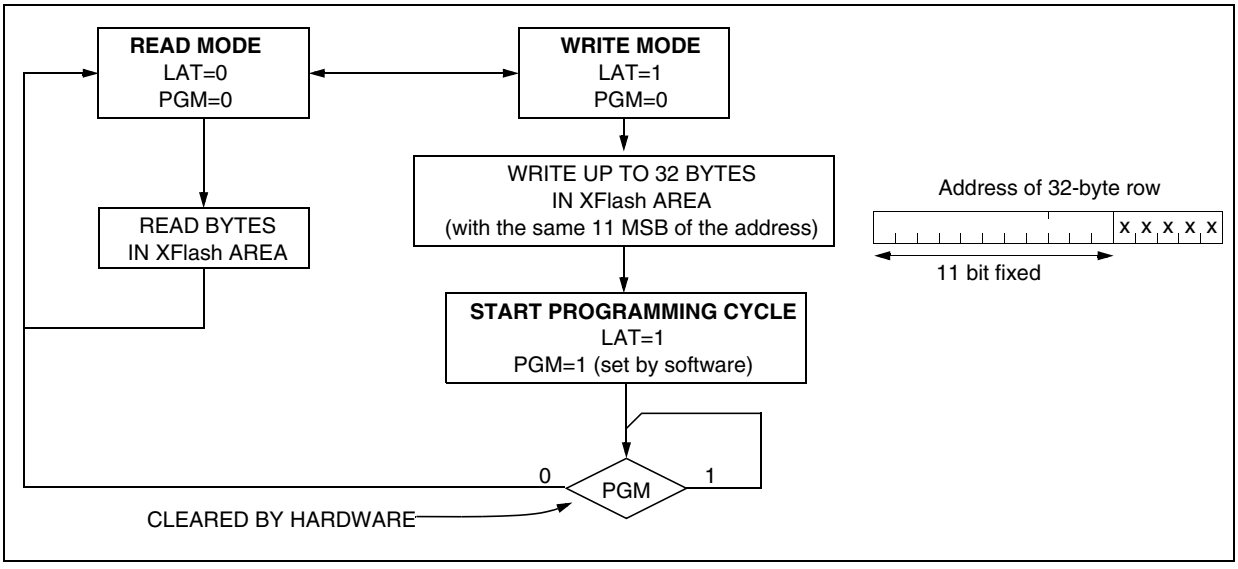
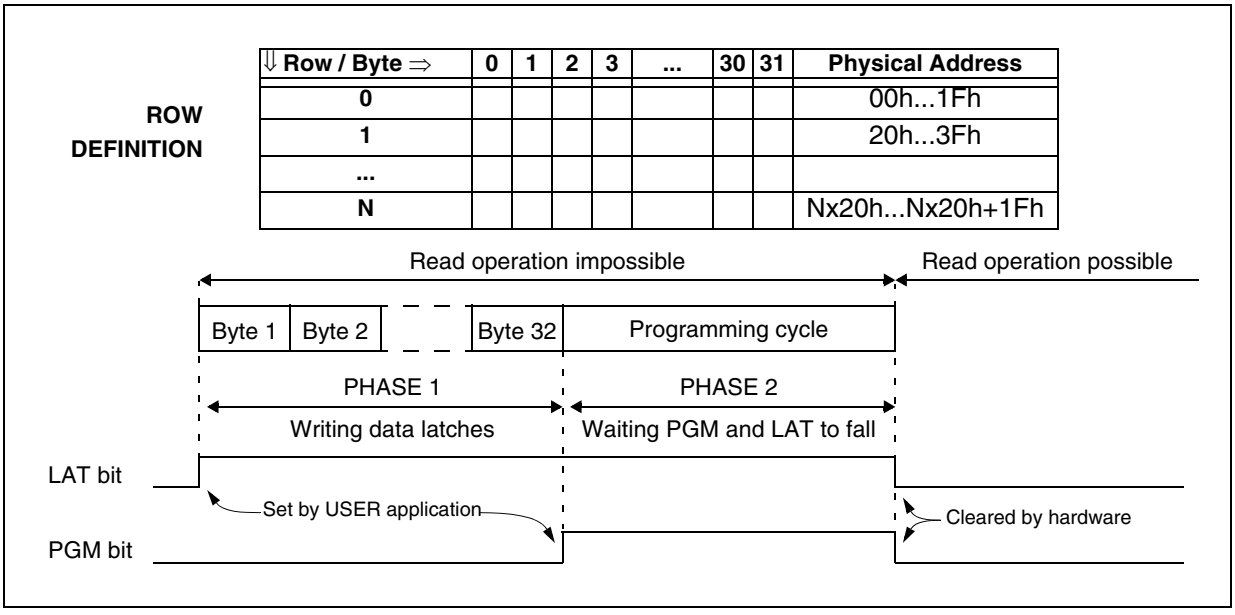


Figure 3. XFlash Write Operation



Note: If a programming cycle is interrupted (by software or a reset action), the integrity of the data in memory is not guaranteed.

2.2.1.1 XFlash Option Byte Programming

For security, each one of the 16 option bits is physically stored as two bytes (where only bit 0 of each byte is significant). Consequently, the 2 option bytes used to configure the MCU are mapped to 32 bytes of FLASH memory known as the Option Byte R21 row (see Figure 4). To access the Option byte row for programming, the OPT bit must be set in the FCSR register.

To program an Option Bit, write bit 0 of the corresponding byte pair to the opposite value you want to program and write bit 8 of the byte pair to the value (see Figure 4). Then, for each byte pair, the value can be set to FFFEh (option is set) or FEFFh (option is cleared).

The option bits are taken into account immediately after they are programmed, and not after the next reset as is the case with HDFlash.

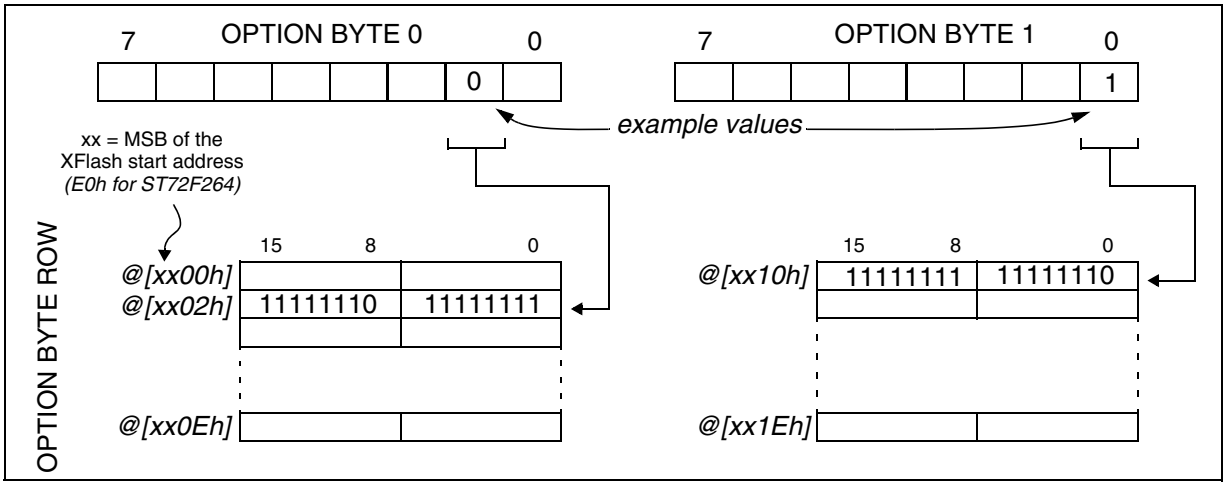
At the end of Option Byte programming, the OPT bit must be cleared by software.

A source code example is given in APPENDIX 2 on page 44.

Important note: To avoid static power consumption, it is mandatory to program ALL THE OPTIONS (and, consequently, all the 32 bytes in the option byte row), respecting the rule that the LSBit of each byte in the byte pair must be programmed with opposite values (all other bits must be kept set). If this rule is not respected, static consumption will occur and the option byte will be undefined. Moreover, a parasitic write protection could be set making the XFlash not reprogrammed.

Caution: As soon as the LAT bit is set and up to the end of programming, the XFlash contents cannot be executed. So the option byte programming sequence has to be executed from another memory (RAM, Data E^2...).

Figure 4. Option Byte Mapping



2.2.1.2 XFlash Protection Option Bits

XFlash systematically includes two option bits dedicated to memory protection. One of these two bits (bit 0) is used to protect the XFlash content from being read (see section 1 on page 8 for more details), while the second one (bit 1) can be used to write protect the device (FLASH ROM capability).

- Read-out protection bit: as soon as this option bit is set, it is not possible to read-out the XFlash content as shown in Table 2. If this option bit is erased in order to re-program the XFlash that has been read-out protected, the whole XFlash memory is erased first. Note that even though no protection can be considered as totally unbreakable, the feature provides a very high level of protection for a general purpose microcontroller.
- Write protection bit: as soon as this option bit is set, it is not possible to re-write the XFlash memory, it behaves like a ROM (see Table 2).

Caution: When the write protection is set, STMicroelectronics is not able to disable this option without erasing the entire XFlash memory.

Table 2. ST7 XFlash Memory Protections

<table>
<tr><td colspan="2" align="right">ST7 Modes →</td><td colspan="2">USER mode (including IAP)</td><td colspan="3">ICC mode</td></tr>
<tr><td colspan="2" align="right">FLASH write protection bit →</td><td>0</td><td>1</td><td>0</td><td>1</td><td>X</td></tr>
<tr><td colspan="2" align="right">FLASH read-out protection bit →</td><td>X</td><td>X</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td colspan="2">DATA EEPROM when available</td><td colspan="2">R / W / E</td><td colspan="2">R / W / E</td><td rowspan="2">Protected (not accessible)</td></tr>
<tr><td rowspan="2">XFlash</td><td>Sector 0</td><td colspan="2">R / E</td><td rowspan="2">R / W / E</td><td rowspan="2">R / E</td></tr>
<tr><td>Sector 1</td><td>R / W / E</td><td>R / E</td></tr>
<tr><td rowspan="3">Option bytes</td><td>Write protection</td><td colspan="2" rowspan="3">Protected (not accessible)</td><td>R / W once</td><td rowspan="3">R</td><td>R / W once</td></tr>
<tr><td>Read-out protection</td><td rowspan="2">R / W</td><td>R / W</td></tr>
<tr><td>Others</td><td>R</td></tr>
</table>

Notes: the following abbreviations are used in the previous table.
R / W / E: read, write and execute capabilities
R / W: read and write capabilities
R / W once: read capability and write once capabilities
R / E: read and execute capabilities
R: read capability

2.2.1.3 Readout Protection Recovery

Removal of readout protection causes loss of RC calibration values in devices supporting this feature. When implementing your own programmer, these values should be restored when readout protection is removed using the following procedure:

1. Enter ICC mode with external clock. In this mode, option bytes configuration is ignored. (Refer Table 1)
2. Remove readout protection.
3. Refer to the RC calibration value table in the Flash Programming Quick Reference Manual and read off the correct RC calibration address values according appropriate device.
4. Rewrite the values in FLASH (0xFFDE and 0xFFDF) and/or EEPROM (0x1000 and 0x1001).

A user may want to either use a particular address as standard memory or store the RC calibration values in it. In order to provide this flexibility, it is suggested to prompt the user for restoration of each RCCRx location in FLASH and EEPROM. In case the user chooses to use these addresses as standard memory, the programmer must never write into these addresses and in case the user wants to use them to store RC calibration values, the programmer must give a warning in case of any attempt of violation.

2.2.1.4 XFlash and RASS Protection

To protect the XFlash memory against involuntary re-programming, before programming the XFlash for the first time after a device reset, the RASS key sequence has to be written in the FLASH Control/Status Register (FCSR). If the access to the FCSR is write-protected by the RASS (wrong keys have been previously written in FCSR), a device reset has to be performed to unlock the RASS protection:

```
.RASS_Disable
    LD    A,#$56    ; 1st hardware key to unlock FCSR
    LD    FCSR,A
    LD    A,#$AE    ; 2nd hardware key to unlock FCSR
    LD    FCSR,A
    RET
```

Caution: This example is intentionally simplified. For reliability reasons, keys should not be stored in the program memory. See Section 1.3 for more details.

To check if the FCSR register is locked or not by the RASS, set the LAT bit and read it again to verify if the bit has been written. If yes, the FCSR is unlocked, otherwise it is locked and a reset sequence has to be generated in order to unlock it.

2.2.1.5 XFlash Control/Status Register (FCSR)

Read/Write
Reset Value: 0000 0000 (00h)
1ST RASS Key: 0101 0110 (56h)
2ND RASS Key: 1010 1110 (AEh)

7							0
0	0	0	0	0	OPT	LAT	PGM

Bits 7:3 = Reserved, must be always kept cleared.

Bit 2 = **OPT**: *Option byte Programming Mode.*
This bit is set and cleared by software to select Program memory or Option byte access in ICP mode. When the LAT bit is set, the OPT bit can not be modified. So the OPT and the LAT bits cannot be set together, this operation must be done in two steps (example: BSET FCSR,#OPT then BSET FCSR,#LAT).
0: Program memory access
1: Option byte access

Bit 1 = **LAT**: *Read/Write bit.*
This bit is set by software. It is cleared by hardware at the end of the programming cycle. It can only be cleared by software if the PGM bit is cleared. While LAT bit is set the XFlash cannot be executed.
0: Read mode (32-byte latches are not accessible and are all set to FFh)
1: Write mode (32-byte latches are accessible)

Bit 0 = **PGM**: *Program Bit*
This bit is set by software to begin the programming cycle. At the end of the programming cycle, this bit is cleared by hardware.
0: Programming finished or not yet started
1: Programming cycle is in progress

Note: If the PGM bit is cleared during the programming cycle, the memory data is not guaranteed

2.3 XFlash PROGRAMMING TIME

Description		Conditions	Max.	Unit
Erase/Programming pulse time		up to 32 bytes at a time	5	ms
Erase/programming time	1 kbyte	by block of 32 bytes	160	
	nb kbyte		nb x 160	

Note: The programming time is proportional to the number of bytes to be programmed but independent from f_{CPU}. The XFlash latch loading and the communication time for getting the data to be programmed depend on the application (including f_{CPU}) and have to be added to this erase/programming time. For more details on programming time refer to APPENDIX 1 on page 41.

2.4 XFlash IN-CIRCUIT PROGRAMMING

2.4.1 ICP Method

ICP uses a protocol called ICC (In-Circuit Communication) to communicate with an external programming device connected via cable. See “ICC Protocol” reference manual for more details on ICC mode entry and the ICC protocol. ICP is performed in three steps:

– Switch the ST7 to ICC mode (In-Circuit Communications). When the ST7 enters ICC mode, it fetches a specific RESET vector which points to the ST7 System Memory containing the ICC protocol routine. This routine enables the ST7 to download the ICP driver into RAM.

– Download the ICP driver software into the RAM of the ST7 microcontroller using ICC.

– Using the ICC <go> command, jump to the ICP driver start address to start the XFlash programming sequence (this software may use the ICC protocol to transfer the data to be written in XFlash memory or another protocol).

The ST7 exits ICC mode by being reset.

Refer to the device datasheet for the hardware interface on the application board.

2.4.2 ICP Example

In this ICP driver example, the programmed device receives the data bytes using the ICC protocol while a programming cycle is in progress. This process can be used to optimize the global programming time. The programming and verify phases have been split to fit in a small RAM memory space. When a large RAM memory is available the programming and verify drivers can be downloaded at the same time. The verify procedure is needed to check if the XFlash has been correctly programmed. The `ICP_Prog_Drv` and `ICP_Verif_Drv` programs are described in Figure 5 and Figure 6 respectively. They are also described in APPENDIX 5 on page 56.

The programming procedure is summarized with the following steps:

– Enter ICC mode using the specific reset sequence (ICC monitor is then executed).

– Using the ICC <Write memory> command, download the programming `ICP_Prog_Drv` program into RAM. The purpose of the `ICP_Prog_Drv` program is to receive data bytes while a programming cycle is in progress and to program these data bytes without any verification. The data bytes are received by calling the `ICC_receive_byte` subroutine stored in the System Memory.

– Using the ICC <Go> command, jump to `ICP_Prog_Drv` to launch the programming phase. At the end of the `ICP_Prog_Drv` program, the program jumps back to the ICC monitor to be ready for the next download to RAM.

– Using the ICC <Write memory> command, download the `ICP_Verif_Drv` program into the RAM. The purpose of the `ICP_Verif_Drv` program is to check that the programming has been correctly completed by reading the entire XFlash memory. This check is done by calling

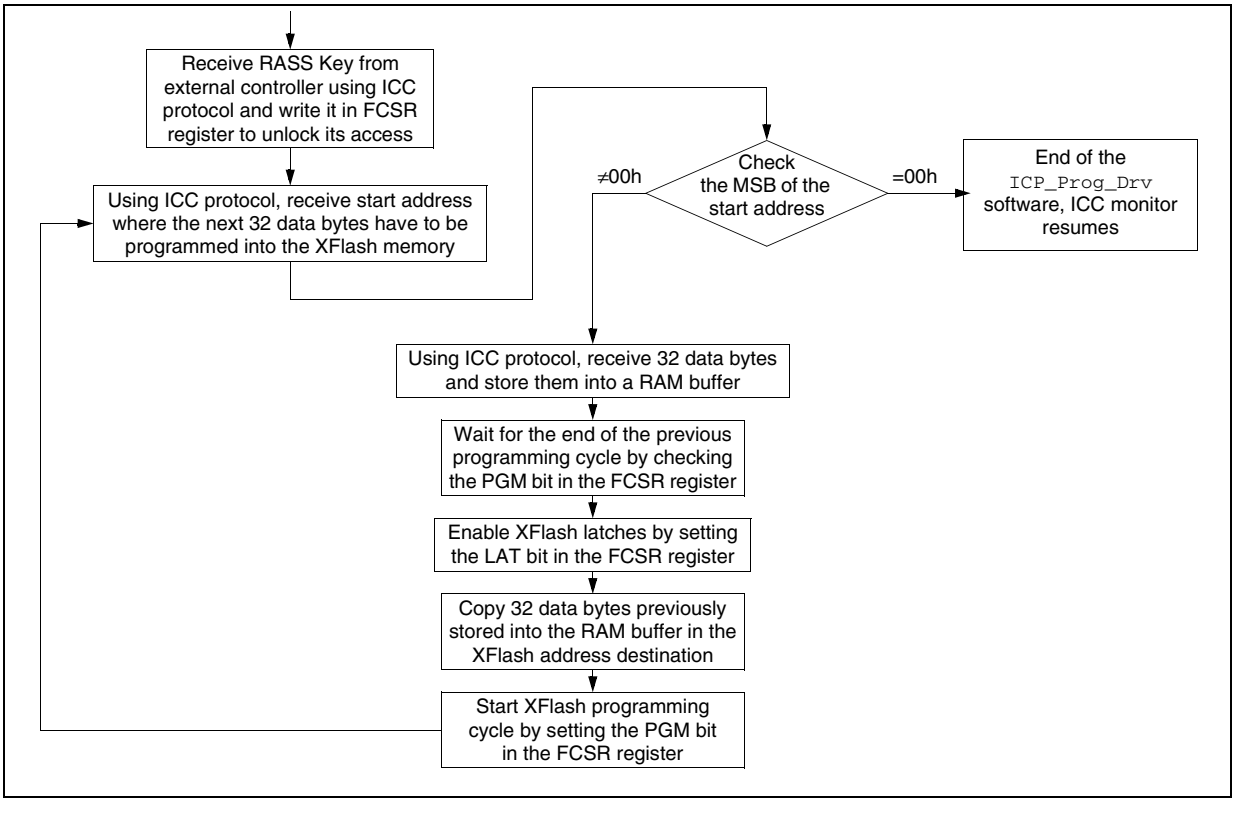
the ICC_receive_byte subroutine stored in the System Memory and the ICC_send_byte subroutine downloaded in the RAM. At the end of the ICP_Verif_Drv program, ICC mode recovers control.

Note: For the address of the ICC_receive_byte subroutine refer to “ICC Protocol” reference manual.

2.4.2.1 ICP_Prog_Drv Example Program

The ICP_Prog_Drv example program is described in Figure 5. The assembler software is shown in APPENDIX 5 on page 56.

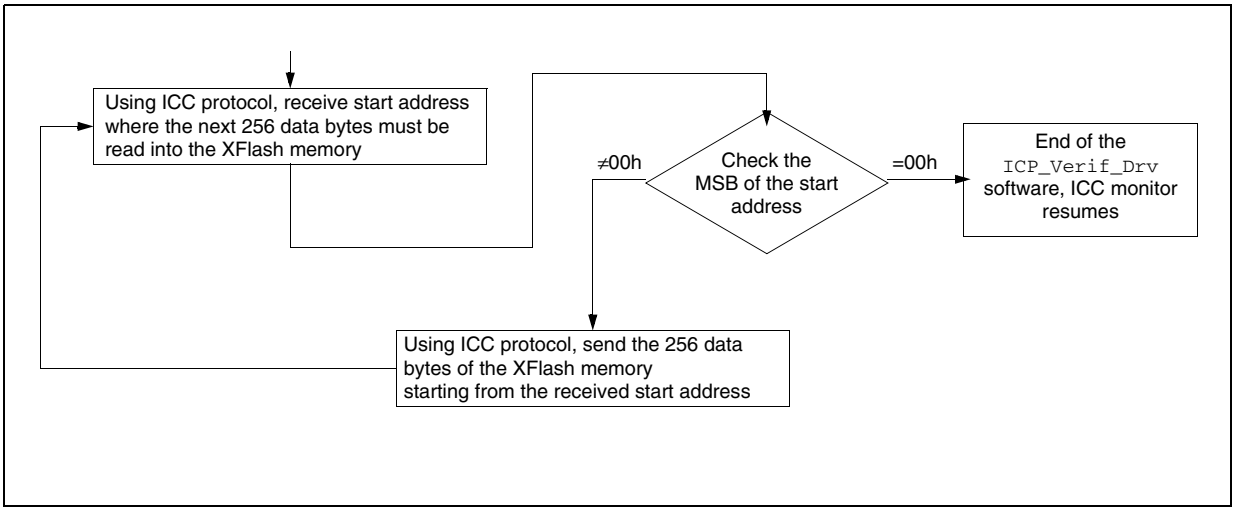
Figure 5. ICP_Prog_Drv Flowchart



2.4.2.2 ICP_Verif_Drv Example Program

The `ICP_Verif_Drv` example program is described in Figure 6. The assembler software is shown in APPENDIX 5 on page 56.

Figure 6. ICP_Verif_Drv Flowchart



Note: To speed-up the verify procedure after the XFlash programming, a simple checksum can be computed by the ST7. In this case, only the checksum result is sent to the controller reducing to the minimum the time spent for communication.

2.5 XFlash IN-APPLICATION PROGRAMMING

2.5.1 IAP Method

In-Application Programming (IAP) enables the user to re-program the contents of the XFlash memory while executing the user application software. IAP has been implemented for users who want their application software to update itself by re-programming the XFlash memory during execution.

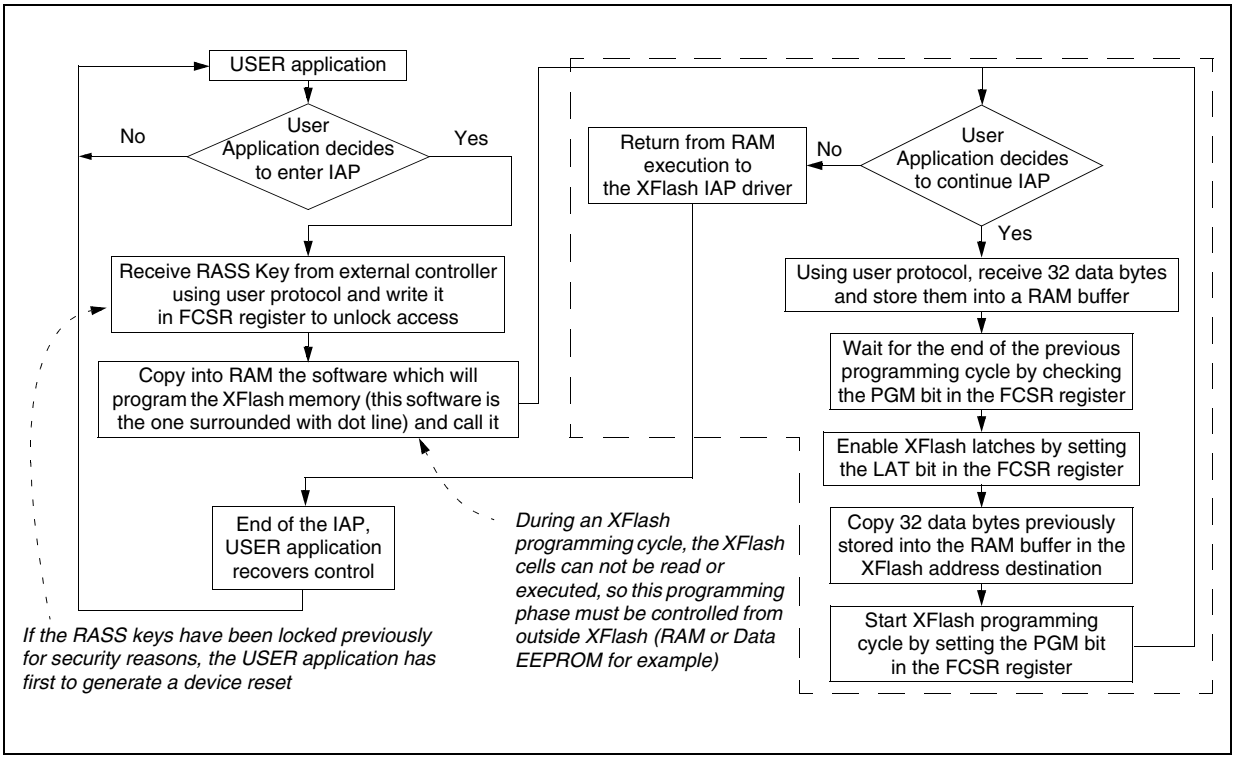
Sector 0 of the XFlash memory is write-protected in USER mode. To use IAP, this sector must contain the software driver (called IAP driver) to be able to re-program XFlash Sector 1. The IAP driver must be programmed by the user in Sector 0 using ICP. Sector 0 size is configurable by option byte so it can be adapted to different applications (see product datasheet for more details).

If a problem occurs during IAP, its execution can be recovered due to the fact that the IAP driver is always write/erase protected. This IAP driver can be programmed using ICP. This gives the user complete flexibility when integrating the IAP function in the application (mode of entry, choice of communication interface used to download the data to be programmed, etc...).

2.5.2 IAP Example

The implementation of IAP depends on the user application software. In the following example (described in Figure 7), the data to be programmed into the XFlash memory is read from an external controller using a communication method called “user protocol”. This user protocol depends on the application and can use, for example, an SPI, SCI, CAN or USB interface.

Figure 7. XFlash IAP Driver Example Flowchart



Caution: During XFlash memory programming cycles, all interrupts MUST be disabled to prevent unexpected Fetch commands from being sent to XFlash memory (not accessible at that time).

3 PROGRAMMING HDFLASH MCUS

3.1 INTRODUCTION

This section describes how to program the ST7 Dual-Voltage HDFlash microcontrollers (MCUs). See ST7 device list on Page 3.

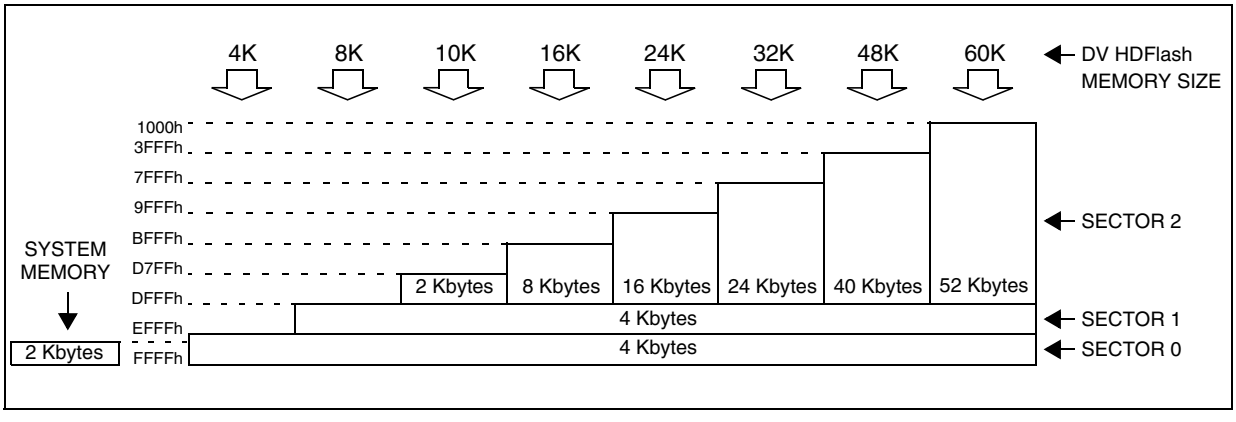
3.2 HDFlash PROGRAMMING

3.2.1 HDFlash Memory Organization

The ST7 DV (Dual Voltage) HDFlash is a non-volatile memory that may be electrically erased (by sector or completely at a time) and programmed in the application board on a byte-by-byte basis using an external V_{PP} supply. Since the external V_{PP} supply must not be permanently supplied, it has to be present only when a write or erase action is in progress. Nevertheless it is still possible to execute the HDFlash memory when the programming voltage is supplied to the V_{PP} pin as V_{PP} is not directly connected to the HDFlash cell but goes through an internal switch controlled by the Embedded Commands. The array matrix organization enables each sector to be erased and reprogrammed without affecting other sectors.

In the ST7 HDFlash memory, each sector can be erased and programmed only a few hundred cycles. It is organized in 8-bit wide sectors that can be used for both code and data storage.

Figure 8. Memory Map and Sector Addresses



The DV HDFlash memory contains one, two or three user sectors depending on its size and one system sector called System Memory (see Figure 8). Each of the user sectors can be erased independently (filled with FFh). This prevents the unnecessary erasing of the entire HDFlash memory when only a partial erasing is required. The two first user sectors are set to 4 Kbytes for all memory sizes. They are mapped in the upper part of the ST7 addressing space and the reset and interrupt vectors are contained in Sector 0 (F000h-FFFFh). The size of the third user sector depends on the size of the total HDFlash memory and is not present if it is lower than or equal to 8 Kbytes.

The System Memory sector is located in a separate memory page that cannot be accessed by the user application. The System Memory contains the ICC monitor and the Embedded Command software that are used for programming and erasing the HDFlash memory.

3.2.2 Embedded Command Use

The HDFlash memory is programmed by executing the Embedded Commands located in System Memory as described in the following procedure.

1. Write the Embedded Command parameters in the specified RAM addresses (see Table 3).
2. Ensure that at least 124 bytes are available in the stack.
3. Unlock access to the FLASH Control/Status Register (FCSR) by writing the two RASS hardware keys.
4. If available, disable the top level interrupt (TLI). This precaution must be taken to avoid a ST7 program counter corruption when a TLI occurs while the ST7 is programming the HDFlash. Consequently, HDFlash programming cannot be started during TLI routine execution.
5. Write a random value in the FLASH Control/Status Register (FCSR). This causes the ST7 to generate a non-maskable interrupt and execute the selected Embedded Command in System Memory. The user application execution is stopped, but all hardware peripherals remain active. All interrupts (except TLI) remain pending until the Embedded Command has finished execution.
6. When the Embedded Command is finished, an IRET instruction is generated and the user application program resumes. A status code is returned in the “ECMD” RAM location (address 00FFh, see Table 4).

3.2.3 RAM Memory Management

3.2.3.1 Parameter Passing

The Embedded Command parameters are exchanged through a RAM area located from 00F8h to 00FFh on page zero (see Figure 9). Table 3 lists the commands and parameters that must be specified by the user.

This RAM area as well as 00F0h to 00F7h is also used by the Embedded Command to store local variables and must not be written by the user application until the Embedded Command has completed execution as indicated by the return status code (see Table 4). When Embedded Commands are not used, the 00F0h..00FFh area is available for the application.

Table 3. Embedded Command Description

Embedded Commands		Embedded Command Parameters						
Name	ECMD (00FFh)	SECT (00FEh)	PTRL (00FDh)	PTRH (00FCh)	ENDL (00FBh)	ENDH (00FAh)	DATA (00F9h)	FREQ (00F8h)
Byte programming	00h		@ Low	@ High			Data	[1..8] 1)
Block programming	01h	Byte Number	Flash Start@L	Flash Start@H	Buffer 2) Start@L	Buffer 2) Start@H		[1..8] 1)
Option byte programming 3)	02h		20h		38h	Opt1	Opt0 4)	[1..8] 1)
Sector and option erase (erased value is FFh)	03h	Flash Sector 5)	03h	Flash Sector 5)				[1..8] 1)
All programming 6)	04h		00h	04h	FFh	FFh	00h	[1..8] 1)
Option byte read	05h					Returned Opt1	Returned Opt0 4)	[1..8] 7)
Checksum computation (256 bytes)	06h		Start@L	Start@H	Returned CSL	Returned CSH		[1..8] 7)

Notes:

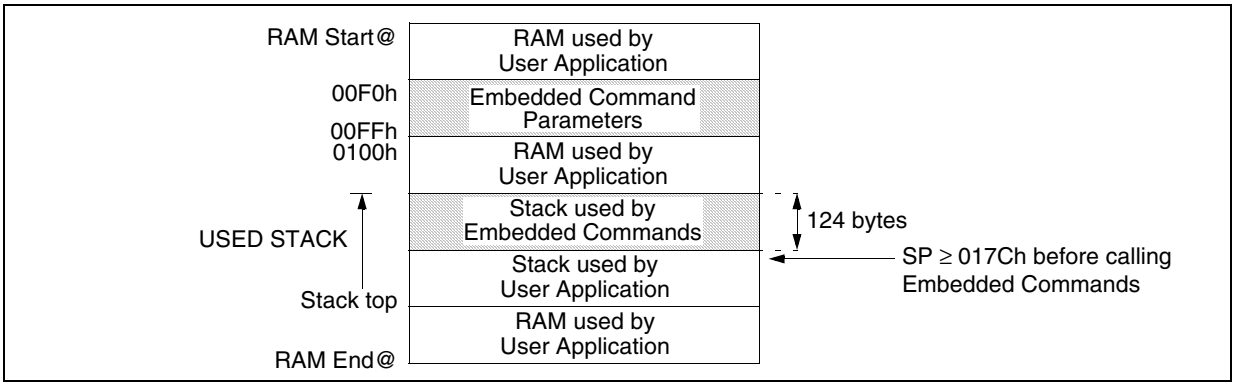
1. For recommendations on using the FREQ parameter, refer to section 3.2.6 on page 33. FREQ must be set for all commands and its value must never exceed f_{CPU}, or 8 if f_{CPU} is greater than 8MHz.
2. The data buffer to be programmed can be either in the RAM area or in the hardware register area.
3. A new option byte configuration becomes valid only after next device RESET. Nevertheless, the <Option byte read> command returns new option byte value. To program option bytes correctly, an erase must always be performed before programming an option byte, even if the value is FFh. Even if the device uses only one option byte for user settings, both option bytes have to be read and the second one reprogrammed with the value previously read. The same applies to reserved option bits, they must be programmed with the same value got from the read operation.
4. The Opt0 option byte includes the read-out protection bit (see Section 3.2.5.1 for more details).
5. To erase option bytes, the <Sector Erase> command must be called with Sector 0Fh as parameter (see Section 3.2.5.6 for more details).
6. This command can only be used to program the entire FLASH memory to 00h. In USER mode, this command must not be used when the watchdog is activated because this command has no timeout (see Section 3.2.4). If this command is used in USER mode, it returns a fail status after programming Sector 2 and 1 due to the fact that Sector 0 is write protected. The <All Programming> command must be called before erasing the option byte when the read out protection is activated.
7. FREQ parameter value is not used for the command computation but must be set between 1 and 8.

Caution: The returned status code, ECMD, MUST ALWAYS be checked after each Embedded Command (see Section 3.2.3.3). After an Embedded Command has been executed, not only is the ECMD location modified but all the reserved area (from 00F0h to 00FFh) can be corrupted by the execution. Consequently, after each call, all command parameters have to be re-initialized.

3.2.3.2 Stack Usage

At least 124 bytes of the stack are required to execute the Embedded Commands (see Figure 9). This is due to the fact that during the programming phase, the DV HDFlash memory cannot be read or executed, so execution is done from RAM.

Figure 9. Required RAM area for Embedded Commands



3.2.3.3 Returned Status Codes

A status code is returned by each Embedded Command in the ECMD RAM location, to report if the operation was performed successfully (see Table 4). This status code is only given in the 4 most significant bits of the ECMD RAM location.

Caution: The returned status code MUST be checked after each Embedded Command call to be sure that the command has been done successfully. To guarantee a correct programming or erasing, the Embedded Command checks the programming or the erasing in the worst conditions and then returns the result in the status code.

Table 4. Returned Status Codes

Status Code (ECMD MSB Bits)	Description
none (0xh)	Operation successful
bit4=1 (1xh)	Operation failed
bit5=1 (2xh)	V_{PP} voltage out of programming range
bit6=1 (4xh)	Incorrect command or parameters or not enough space in stack
bit7=1 (8xh)	Operation not yet completed (“BUSY”)

Caution: Only one status bit is returned at any one time, so one status type may mask another.

3.2.4 Command Timeout

As some Embedded Commands can last a long time (like <Block Programming> and <Sector Erase>), a maximum timeout is implemented (see Table 5) after which the Embedded Commands are temporarily exited (user application resumes).

This pending command is indicated to the user application through a “BUSY” return status code (bit 7 of ECMD set as shown in Table 4). This timeout is implemented to enable the user, for example, to refresh an internal or external watchdog if necessary. Writing to the FLASH Control/Status register (FCSR) causes the temporarily stopped Embedded Command to resume execution.

The detailed returned status code for each Embedded Command is shown in Table 6.

If an Embedded Command is exited without being completed, the 00F0h to 00FFh RAM area must not be modified. This RAM area is used to memorize the status of the Embedded Command state machine. To finish the current Embedded Command without completed it, call the Embedded Command again with a wrong FREQ value (FREQ=FFh for example) to get an error status.

See the examples shown in Section 3.2.5.4 and Section 3.2.5.6.

Table 5. Embedded Command Timeout

Embedded Commands			**Block programming**	**Sector erase**
“BUSY” Maximum Timeout	**Formula** [t_{CPU}]		41808+5120.FREQ	1587+54069.FREQ
	Duration with f_{CPU}=8MHz	**FREQ=1**	~5.9ms	~7.0ms
		FREQ=8	~10.4ms	~54.3ms

Table 6. Embedded Command Return Status Codes

Embedded Command	Operation	Returned status (ECMD: 00FFh)				
Byte programming	**Normal Operation**	**00h** → ↓	→ 00h succeded			
	Failed [1]	X0h				
Block programming	**Normal Operation**	**01h** → ↓	→ 81h [2] → block prog ↓	→ 01h succeded		
	Failed [1]	X1h failed	11h failed			
Option byte programming	**Normal Operation**	**02h** → ↓	→ 02h succeded			
	Failed [1]	X2h				
Sector erasing	**Normal Operation**	**03h** → ↓	→ 87h [2] → sector prog. ↓	→ 8Ah [2] → sector erase ↓	↔ 8Bh [2] → sector verify ↓	→ 03h succeded
	Failed [1]	X3h	17h	1Ah	1Bh	
Option byte erasing	**Normal Operation**	**03h** → ↓	→ 09h succeded			
	Failed [1]	X3h,18h				
All programming	**Normal Operation**	**04h** → ↓	→ 04h succeded			
	Failed [1]	X4h				
Option byte read	**Normal Operation**	**05h** → ↓	→ 05h succeded			
	Failed [1]	x5h				
Checksum computation	**Normal Operation**	**06h** → ↓	→ 06h succeded			
	Failed [1]	x6h				

Notes:

1. In the returned status code, X∈{1,2,4} and x∈{1,4} (see Table 4 for the meaning of these numbers).
2. This returned status means that the command is not yet finalised but the Embedded Command has stopped temporarily after the timeout. The low significant nibble indicates during which phase the Embedded Command has stopped.

3.2.5 Command Descriptions and Examples

3.2.5.1 Commands and HDFlash Protections

HDFlash systematically includes an option bit dedicated to memory protection. This bit (bit 0 of the OPT0 option byte) is used to protect the HDFlash content from being read (see section 1 on page 8 for more details). As soon as this option bit is reset, it is not possible to read-out the HDFlash content as shown in Table 7.

To remove the read-out protection (to allow the re-programming of the HDFlash), the following sequence has to be performed:

- First, the entire memory must be written to 00h using the “All programming” command.
- If the “All programming” command succeeds (see the return status code), then, the option bytes can be erased (to FFh) using the “Sector erase” command.
- Finally, the option bytes can be programmed again using the “Option byte programming” command.

No reset must be performed between the execution of the two first Embedded Commands, otherwise the complete sequence must be done again.

Table 7. ST7 HDFlash Memory Protections

ST7 Modes →		USER mode (including IAP)		ICC mode	
FLASH read-out protection bit →		1	0	1	0
HDFlash	Sector 0	R / E		R / W / E	Protected (not accessible)
	Sector 1 or 2	R / W / E	R / W / E		
Option bytes	Read-out protection	R / W [1]	R [2]	R / W	R [3]
	Others				

Legend: the following abbreviations are used in the previous table.
R / W / E: read, write and execute capabilities
R / W: read and write capabilities
R / E: read and execute capabilities
R: read capability

Notes:

1. Option bytes can be programmed and erased using IAP in USER mode (“Option byte programming” and option byte “Sector erase” commands are available). But they must not be used if the option bytes enable the user to configure critical ST7 system configurations (such as oscillators). Consequently ST7 functions are not guaranteed if these commands are used.
2. The “All programming” command is principally used to remove readout protection. It is accessible in USER mode but it systematically returns a “fail” status when trying to program Sector 0. So, when the read-out protection is set, the “Sector erase” command for option byte will always fail and then the option byte can not be modified. Consequently, the read-out protection cannot be removed using IAP.
3. In ICC mode, when the “All programming” command is called while the read-out protection is set, the option bytes remain unchanged. To remove the protection, the option byte “Sector erase” command must be performed before the next reset.

Depending if the ST7 is in USER mode or ICC mode, the Embedded Command are available in the same conditions as shown in Table 8.

Table 8. Embedded Command Availability

Embedded Commands	HDFlash not protected			HDFlash read-out protected [1]		
	ICC mode	USER mode (including IAP) [2]		ICC mode	USER mode (including IAP) [2]	
	All Sectors	S0	S1 & S2	All Sectors	S0	S1 & S2
Byte programming	X		X			X
Block programming	X		X			X
Option byte programming	X	X [3] [4]			X [3] [4]	
Sector erase	X		X+Options[3]			X+Options[3]
All programming	X	X [4]		X [5]	X [4]	
Option byte read	X	X		X	X	
Checksum computation	X	X		X	X	

Notes:

1. When programming the read-out protection, its activation is effective only after a device RESET.
2. If the ST7 device contains one HDFlash sector (sector 0), the IAP is not available for this device as all the HDFlash memory is write/erase protected while the user application is running.
3. The “Option byte programming” and option byte “Sector erase” commands are accessible in USER mode, but they must not be used if the option bytes enable the user to configure critical ST7 system configurations (such as oscillators). Consequently ST7 functions are not guaranteed if these commands are used.
4. The “All programming” command is accessible in USER mode but it systematically returns a “fail” status when trying to program Sector 0. So, when the read-out protection is set, the “Sector erase” command for option byte will always fail and then the option byte can not be modified. Consequently, the read-out protection cannot be removed using IAP.
5. In ICC mode, when the “All programming” command is called while the read-out protection is set, the option bytes remain unchanged.

3.2.5.2 RASS Key Entry

To protect the HDFlash memory against involuntary re-programming, before accessing the Embedded Commands for the first time after a device reset, the RASS key sequence has to be written in the FLASH Control/Status Register (FCSR). If the access to the FCSR is write-protected by the RASS (wrong keys have been previously written in FCSR), a device reset has to be performed to unlock the RASS protection:

```
.RASS_Disable
    LD    A,#$56    ; 1st hardware key to unlock FCSR
    LD    FCSR,A
    LD    A,#$AE    ; 2nd hardware key to unlock FCSR
    LD    FCSR,A
    RET
```

To check if the FCSR register is locked or not by the RASS, execute a dummy “Checksum Computation” or “Option Byte Read” command with an incorrect parameter, for example

FREQ at 0, and check the returned status in ECMD (00FFh). If the returned status is 4xh (Incorrect command or parameters or not enough space in stack) then the FCSR is unlocked, otherwise it is locked and a reset sequence has to be generated in order to unlock it.

Caution: This example is intentionally simplified. For reliability reason, keys must not be stored in the program memory. See Section 1.3 for more details.

3.2.5.3 Byte Programming (00h)

This command enables the user to program a data byte in the user HDFlash memory area.

Note: Data cannot be transferred (via a communication peripheral, for example) while the byte programming operation is in progress, as no user application software can be executed during programming.

Example:

```
.FLASH_WriteByte
    LD  FLASH_DATA,A    ; A   = Data byte to program
    LD  FLASH_PTRL,X    ; Y:X = HDFlash address to program
    LD  FLASH_PTRH,Y
    LD  A,#$00
    LD  FLASH_ECMD,A    ; 00h = Byte programming command
    LD  A,#$01
    LD  FLASH_FREQ,A    ; Set to 1 for optimum prog. time (see note 1 Page 24)
    LD  FCSR,A          ; Dummy write of FCSR = launch HDFlash command
    LD  A,FLASH_ECMD    ; Return the status code in A reg.
    RET
```

3.2.5.4 Block Programming (01h)

This command enables the user to program a user HDFlash memory area using data previously stored in the ST7 RAM.

Note: The FLASH_SECT parameter defines the number of bytes to be programmed from 1 (01h) to 255 (FFh). To program 256 bytes, this parameter must be set to 0 (00h).

Example:

```
.FLASH_WriteBlock
    LD    FLASH_SECT,A  ; A   = Nb of bytes to program (see above note)
    LD    FLASH_PTRL,X  ; Y:X = HDFlash start address to program
    LD    FLASH_PTRH,Y
    CLR   A             ; RAM buffer address to copy start from 0080h
    LD    FLASH_ENDH,A
    LD    A,#$80
    LD    FLASH_ENDL,A
    LD    A,#$01
    LD    FLASH_ECMD,A  ; 01h = Block programming command
    LD    A,#$01
    LD    FLASH_FREQ,A  ; Set to 1 for optimum prog. time (see note 1 Page 24)
.block
    LD    X,#$7F               ; Watchdog update if used in USER application
    LD    WDGCR,X
    LD    FCSR,A               ; Dummy write of FCSR = launch HDFlash command
    BTJT  FLASH_ECMD,#7,block  ; Block write not yet completed => continue
    LD    A,FLASH_ECMD         ; Return the status code in A reg.
    RET
```

A complete source code example is given in APPENDIX 3 on page 50.

3.2.5.5 Option Byte Programming (02h)

This command enables the user to write the two static option bytes. Unused option bits or byte must be kept with the original factory configured value to avoid ST7 malfunctioning. This original value can be obtained using the “Option byte read” Embedded Command. If the device has only 1 user Option byte, always read Option byte 2 as well and write it back with the same value to keep the original factory configuration. It is mandatory to erase option bytes before programming them. Even if the value being programmed is 0xFF, option bytes must first be erased and then they must be programmed with 0xFF without a reset between the erase and program operations. The user must read the ECMD byte after each operation to check successful completion of the operation.

Example:

```
.FLASH_WriteOptByte
    LD  FLASH_DATA,A    ; A = Option byte 1 value to program
    LD  FLASH_ENDH,X    ; X = Option byte 2 value to program
    LD  A,#$38
    LD  FLASH_ENDL,A
    LD  A,#$20
    LD  FLASH_PTRL,A
    LD  A,#$02
    LD  FLASH_ECMD,A    ; 02h = Option Byte programming command
    LD  A,#$01
    LD  FLASH_FREQ,A    ; Set to 1 for optimum prog. time (see note 1 Page 24)
    LD  FCSR,A          ; Dummy write of FCSR = launch HDFlash command
    LD  A,FLASH_ECMD    ; Return the status code in A reg.
    RET
```

3.2.5.6 Sector Erasing (03h)

This command enables the user to erase a selected user HDFlash sector or the option bytes. This command is done in two main steps: first all the sectors (or option bytes) are programmed to 00h and then an erase pulse is performed to erase the HDFlash cell to FFh.

Example:

```
.FLASH_EraseSector
    LD    FLASH_SECT,A  ; A = Sector to erase or option bytes
    LD    FLASH_PTRH,A
    LD    A,#$03
    LD    FLASH_ECMD,A  ; 03h = Sector erasing command
    LD    FLASH_PTRL,A
    LD    A,#$01
    LD    FLASH_FREQ,A  ; Set to 1 for optimum prog. time (see note 1 Page 24)
.erase
    LD    X,#$7F               ; Watchdog update if used in USER application
    LD    WDGCR,X
    LD    FCSR,A               ; Dummy write of FCSR = launch HDFlash command
    BTJT  FLASH_ECMD,#7,erase  ; Erase not yet completed => continue
    LD    A,FLASH_ECMD         ; Return the status code in A reg.
    RET
```

Caution: In ICC mode, an option byte programming must be done systematically after an option byte erasing, without any device reset in-between. This must be done to avoid wrong FFh option values.

3.2.5.7 All Programming (04h)

This command enables the user to program the entire user HDFlash memory to 00h (except option bytes). The purpose of this command is to enable read-out protected devices to be re-programmed without any risk of piracy.

Example:

```
.FLASH_ProgAll
    CLR FLASH_DATA    ; Data to program = 00h in the whole HDFlash matrix
    LD  A,#$FF
    LD  FLASH_ENDH,A  ; Flash end address is FFFFh
    LD  FLASH_ENDL,A
    LD  A,#$04
    LD  FLASH_ECMD,A  ; 04h = All programming command
    LD  FLASH_PTRH,A
    CLR FLASH_PTRL
    LD  A,#$01
    LD  FLASH_FREQ,A  ; Set to 1 for optimum prog. time (see note 1 Page 24)
    LD  FCSR,A              ; Dummy write of FCSR = launch HDFlash command
    LD  A,FLASH_ECMD        ; Return the status code in A reg.
    RET
```

3.2.5.8 Option Byte Read (05h)

This command enables the user to read the option byte content.

Note: The FLASH_FREQ parameter is not used in the option byte read command computation but it must be set in the correct range (integer between 1 and 8) to avoid an incorrect parameter returned status.

Example:

```
.FLASH_ReadOptByte
    LD  A,#$05
    LD  FLASH_ECMD,A    ; 05h = Option Byte read command
    LD  A,#$01
    LD  FLASH_FREQ,A    ; Set to 1 for example(must be in the [1..8] range)
    LD  FCSR,A          ; Dummy write of FCSR = launch HDFlash command
    LD  A,FLASH_ECMD    ; Return the status code in A reg.
    LD  X,FLASH_DATA    ; X = Option byte 1 read value
    LD  Y,FLASH_ENDH    ; Y = Option byte 2 read value
    RET
```

3.2.5.9 Checksum Computation (06h)

This command enables the user to compute a checksum on the user HDFlash memory area. A checksum result in FLASH_ENDH:FLASH_ENDL is the addition of the CSH:CSL input value (FLASH_ENDH:FLASH_ENDL) and the sum of the 256 bytes starting from FLASH_PTRH:FLASH_PTRL. The ST7 computed algorithm is:

```
    CLR   X
.Checksum_loop
    LD    A,FLASH_ENDL
    ADD   A,([FLASH_PTRH.w],X)
    LD    FLASH_ENDL,A
    LD    A,FLASH_ENDH
    ADC   A,#ZEROS
    LD    FLASH_ENDH,A
    INC   X
    JRNE  Checksum_loop
```

Note: The FLASH_FREQ parameter is not used in the checksum command computation but it must be set in the correct range (integer between 1 and 8) to avoid an incorrect parameter returned status.

Example:

```
.FLASH_Checksum
    LD  A,#$06
    LD  FLASH_ECMD,A    ; 06h = Checksum computation command
    LD  A,#$01
    LD  FLASH_FREQ,A    ; Set to 1 for example(must be in the [1..8] range)
    LD  FCSR,A          ; Dummy write of FCSR = launch HDFlash command
    LD  A,FLASH_ECMD    ; Return the status code in A reg.
    LD  X,FLASH_ENDL    ; X = CSL (Checksum low)
    LD  Y,FLASH_ENDH    ; Y = CSH (Checksum high)
    RET
```

3.2.6 Defining the Programming/Erasing Pulse Length (FREQ Parameter)

The FREQ parameter is used to calculate the programming/erasing pulse length.

When the sector (0,1 or 2) erasing command is run, the following sequence of operations is executed:

– Program all0 + verify all0 + erase + verify all1 + re-program (to recover depleted cells) + verify (after re-programming.)

Therefore two consecutive sector erases are not cumulative.

When the option byte erase command is run, the following sequence of operations is executed:

– Erase + verify all1

Therefore two consecutive option byte erases are cumulative.

Several cases apply which are described below:

3.2.7 Programming the HDFLASH

3.2.7.1 Method when f_{CPU} is unknown

The HDFLASH must have already been erased before programming.

In this case a first trial should be performed with FREQ set to 1: this is the fastest method because only a few short programming pulses are required. Should a fail occur, other trials should be performed with the FREQ parameter incremented up to 8.

This method is true for byte, block, option bytes and all programming and should succeed with an ambient temperature below 60°C.

3.2.7.2 Method when f_{CPU} is known

The HDFLASH must have already been erased before programming.

The method in Section 3.2.7.1 is applied and a first trial is be performed with FREQ set to 1: this is the fastest method because only a few short programming pulses are required. Should a fail occur, other trials should be performed with the FREQ parameter incremented up to f_{CPU}.

This method is true for byte, block, option bytes and all programming and should succeed with an ambient temperature below 60°C.

3.2.7.3 Other method when f_{CPU} is known

The HDFLASH must have already been erased before programming.

FREQ should be set to f_{CPU} value: this is not the fastest method because the programming pulses are longer compared to the methods in Section 3.2.7.1 and Section 3.2.7.2.

This method is true for byte, block, option bytes and all programming and is recommended when the temperature is above 60°C.

3.2.8 Erasing the HDFLASH

3.2.8.1 Method when f_{CPU} is known

In this case FREQ should be set to the same value as f_{CPU}. This is the optimum condition for erasing in terms of time and efficiency, because long erase pulses are applied.

This method is recommended over the whole temperature range and the security timeout should be set to 80s.

3.2.8.2 Method when f_{CPU} is unknown

In this case, a first trial should be performed with FREQ set to 1. Should a fail occur, other trials should be performed with the FREQ parameter incremented up to 8.

Since the erasing is not cumulative, this method is longer and less efficient than the previous one, but it should succeed with an ambient temperature below 60°C.

Since the erasing is not cumulative, this method is longer and less efficient than the previous one, but it should succeed with an ambient temperature below 60°C and the security timeout should be set 170s.

3.2.9 Erasing the option bytes

3.2.9.1 Method when f_{CPU} is unknown

In this case a first trial should be performed with FREQ set to 1. Should a fail occur other trials should be performed with the FREQ parameter incremented up to 8. This method is the most efficient because the trials are cumulative.

This method should succeed over the whole temperature range and the security timeout should be set to 16s.

3.2.9.2 Method when f_{CPU} is known

The method in Section 3.2.9.1 is applied and then a first trial should be performed with FREQ set to 1. Should a fail occur other trials should be performed with the FREQ parameter incremented up to f_{CPU}. This method is the most efficient because the trials are cumulative.

This method should succeed over the whole temperature range and the security timeout should be set to 16s.

3.2.9.3 Other method when f_{CPU} is known

Another method is to perform a first trial with FREQ set to f_{CPU}. Because the maximum number of iterations authorized when erasing the option bytes is only 48 (compared to 1280 for the HDFLASH), a fail may occur. In this case, 5 subsequent attempts are allowed but the response must be checked after each trial in order not to over-erase the option bytes.

This method should succeed over the whole temperature range and the security timeout should be set to 16s.

3.2.10 General Rule

In any case, in terms of reliability, it is better to know the f_{CPU} value, in order to prevent the FREQ parameter exceeding f_{CPU}.

3.2.11 Time-out

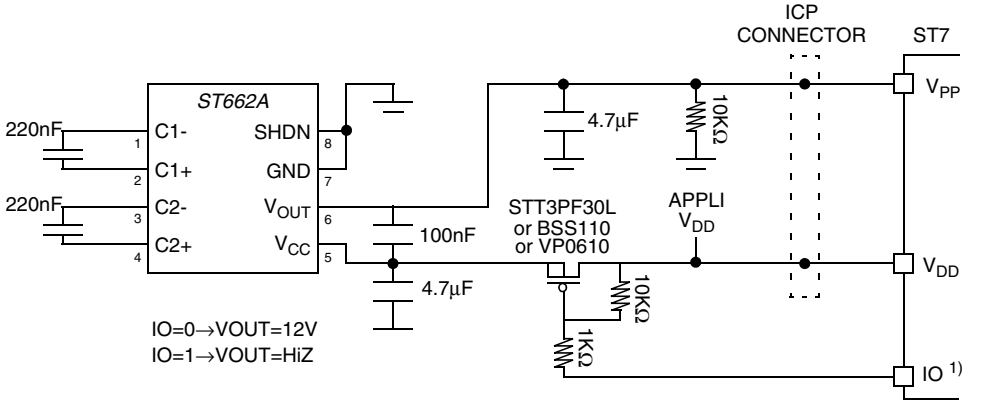
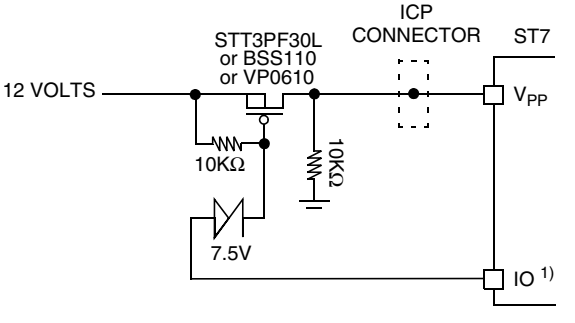
If after the above-mentioned timeouts, the program/erase still fails, the HDFlash should to be replaced (for example this can be due to over-cycling).

3.3 VPP FOR DUAL VOLTAGE HDFlash DEVICES

For Dual-Voltage HDFlash memory devices, an external 12 volt supply has to be provided on the application board for programming and erase operations. Table 9 shows different examples of how to connect the ST7 V_{PP} pin.

Note: For users of programming tools like STICK/EPB/inDART, the 12V programming voltage is supplied by the programming tool and this circuit has to be rebuilt on the application board only if the user application implements IAP.

Table 9. 12 Volt V_{PP} Supply Application Examples

	VPP dedicated circuit for Dual Voltage HDFlash with $4.5V \leq V_{DD} \leq 5.5V$
On Application Board 12Volts Charge Pump for IAP	
On Application Board or External 12Volts for IAP	

Note: Take care that these pins are configured as floating input during the RESET phase (reset value) and are not used by the ICC protocol.

Caution: For reliability reasons, the 12V supply must not be permanently applied on the V_{PP} pin throughout the life of the application. However it can be applied for periods in the order of tens of minutes without any problem.

Warning: The proposed solutions for IAP in Table 9 are not valid if the product has an embedded Low Voltage Detector (LVD) and if it is enabled.

3.4 HDFlash PROGRAMMING TIME

Description	Embedded Command	Formula Approximation for Typical Conditions	Size	Typ. [1]
Programming time	Byte programming	~2600.t_{CPU}+5µs	1b	330µs
	Block programming	~nbblock.[90000.t_{CPU}+1.28ms] (average ~49µs/byte with f_{CPU}=8MHz)	256b	12.5ms
			1kb	50ms
			32kb	1.6s
			60kb	3s
	Option byte programming	~3600.t_{CPU}	2b	450µs

Description	Embedded Command	Sector	Size	Typ. [1] & [2]
Erase time	Sector erase	0	4kb	1.5s ~ 2s
		1	4kb	1.5s ~ 2s
		2	8kb	2s ~ 2.5s
			24kb	4s ~ 4.5s
			52kb	8s ~8.5s
		option bytes	2b	110ms

Notes:

1. Typical values have been measured in the following conditions: Embedded Command, FREQ parameter set to 1, 25°C ambient temperature, f_{CPU}=8 MHz, V_{DD}=5V +-10% and V_{PP}=12V +-5%.
   The maximum value is not specified because Embedded Commands allow up to 126 iterations when programming the HDFlash and up to 1280 iterations when erasing it (only 48 iterations are allowed for the option bytes). The recommended and guaranteed value for the programming voltage (V_{PP}) is 12V.The higher the VPP voltage, the shorter the programming time, subject to the range permitted for V_{PP}.
2. Typical erasing time measurement have been done with the worst flash content (filled with FFh). In this condition, the Embedded Command must first program all the bytes in the memory before erasing the sector, while if the memory is already programmed with 00h, the Embedded Command just checks that it is correctly programmed.
3. For more details on programming times, refer to APPENDIX 1 on page 41.

3.5 HDFlash IN-CIRCUIT PROGRAMMING

3.5.1 ICP Method

In-Circuit Programming (ICP) uses a protocol called ICC (In-Circuit Communication) to communicate with an external programming device connected via cable. See the “ICC Protocol” Reference Manual for more details on ICC mode entry and ICC protocol. ICP is performed in three steps:

– Switch the ST7 to ICC mode (In-Circuit Communications). When the ST7 enters ICC mode, it fetches a specific RESET vector which points to the ST7 System Memory containing the ICC protocol routine. This routine enables the ST7 to download the ICP driver into RAM.
– Download the ICP driver software into the RAM of the ST7 microcontroller using ICC.
– Using ICC <go> command, jump to the ICP driver start address to start the HDFlash programming sequence (this software may use the ICC protocol to transfer the data to be written in HDFlash memory or another protocol).

The ST7 exits ICC mode by being reset.

Refer to the device datasheet for a description of the hardware interface on the application board.

3.5.2 ICP Example Program

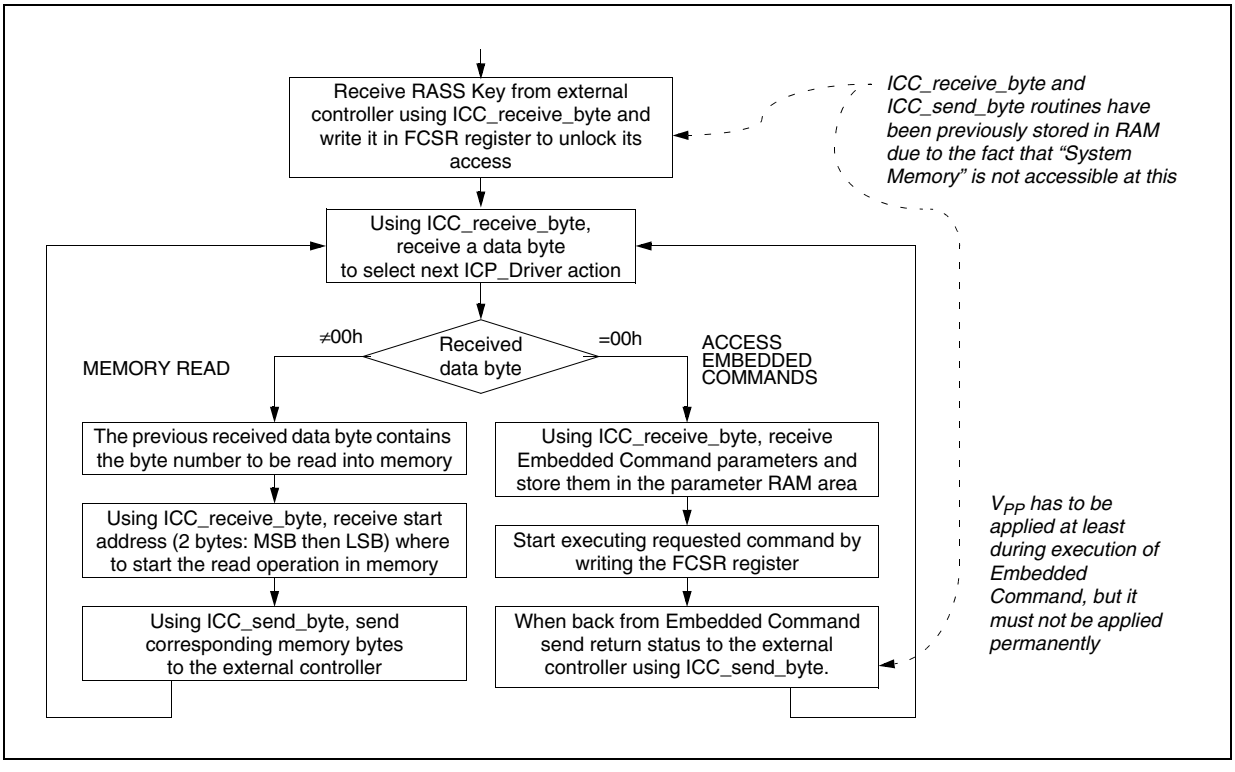
The ICC monitor in the System Memory only includes the ICC <Write Memory> and <Go> commands. The other ICC commands such as the <Read Memory> command must be downloaded into the RAM in order to be executed.

The HDFlash programming procedure is summarized in the following steps:

– Enter ICC mode using the dedicated reset sequence (ICC monitor is then executed).
– Using the ICC <Write memory> command, download the HDFlash programming software (called “ICP Driver”) into the RAM using ICC protocol. The purpose of the ICP Driver is to receive Embedded Commands and then execute them or read any part of the memory for verification. The ICP Driver includes the `ICC_receive_byte` and the `ICC_send_byte` subroutines of the ICC protocol used for data byte communication. These routines are needed in the ICP driver because after the ICC <Go> command execution, the System Memory is no longer accessible.
– Using the ICC <go> command, jump to the ICP driver start address to start the HDFlash programming sequence (this software may use the ICC protocol to transfer the data to be written in HDFlash memory or another protocol).

An example of ICP Driver software is described in Figure 10. To implement the assembler code see the Embedded Command examples given in Section 3.2.5.

Figure 10. HDFlash ICP_Driver Flowchart



After the HDFlash programming, a quick verify procedure can be run to check if the programmed HDFlash content is correct. To do this, the <Checksum computation> Embedded Command can be used and its result sent to the controller using ICC.

APPENDIX 3 on page 50 describes another ASM code example for ICP Driver software.

3.6 HDFlash IN-APPLICATION PROGRAMMING

In-Application Programming (IAP) enables the user to re-program the HDFlash memory contents while executing the user application software. IAP has been implemented for users who want their application software to update itself by re-programming the HDFlash memory during execution.

The HDFlash memory has a sector called Sector 0 that is write-protected in USER mode. This sector must contain the application software driver that is used to re-program all other HDFlash sectors (using a CAN or USB communication protocol, for example). This software driver is called the IAP driver. The IAP driver must be first programmed by the user in the HDFlash Sector 0 using ICP.

If a problem occurs during IAP, its execution can be recovered due to the fact that the IAP driver memory locations are always write/erase protected. This IAP driver is included in the user application software. This gives the user complete flexibility when integrating the IAP function in the application (mode of entry, choice of communication protocol used to retrieve the data to be stored, etc...).

REMINDER: If the Top Level Interrupt (TLI) is available in the device, it has to be disabled before programming the HDFlash to avoid any conflict with HDFlash Embedded Commands (see Section 3.2.2 for more details).

APPENDIX 1 (XFlash AND HDFlash PROGRAMMING TIMES)

This appendix gives timing examples for XFlash, HDFlash and ICC communication. The timings in Table 10 and Table 11 have been measured with the following conditions:

– ST7 f_{CPU}=8MHz and no speed limitation from the programming tool for the communication: ST7 Receive ..... ~ 20.0 kbyte/s
ST7 Transmit ..... ~ 15.0 kbyte/s
– 25°C ambient temperature
– V_{DD}=5V and V_{PP}=12V
– FREQ parameter set to 1 and matrix filled with FFh before erasing for HDFlash memory.

The timings in Table 12 and Table 14 have been measured with the following conditions:

– ST7MDT20J or ST7MDT20M EPB Programming Board
– Parallel I/O port
– ST7 Visual Programmer STVP7 (1.9.0)

The timings in Table 13 and Table 15 have been measured with the following conditions:

– ST7MDT20J or ST7MDT20M EPB Programming Board
– Parallel I/O port
– ST7 Visual Programmer STVP7 (1.9.0) with patch for FREQ=8 algorithm

1.1 XFLASH PROGRAMMING TIME

■ 1 to 32 bytes erasing / programming

– 1~32 byte 5ms
– 1 kbyte ..... 160ms (// by 32)

As the ST7 receives 32 bytes within ~1.6ms and sends 32 bytes within ~2.2ms, the ICC communication (including the verify) can always be done during programming and then at no time cost.

Table 10. XFlash Raw Programming Speed (without software overhead)

Memory Size [kbyte]	1	4	8	16
Prog. by 32byte block	0.16s	0.64s	1.28s	2.56s

1.2 HDFLASH PROGRAMMING TIME

■ Block programming

– n bytes ..... ~ n x 49µs
– 256 bytes ..... ~ 12.5ms
– 16 kbytes..... ~ 800ms 32 kbytes ..... ~ 1.6s 60 kbytes.....~ 3s

■ Sector erasing

– S0 (4 kbytes) .....2s ~ 2.5s
– S1 (4 kbytes) .....2s ~ 2.5s
– S2 (8 kbytes) .....2.5s ~3s (24 kbyte) ..... 4s ~4.5s (52 kbyte) .....7s ~8s

Table 11. HDFlash Raw Programming Speed (without software overhead)

Memory Size [kbyte]	ICC Communication		TOTAL (including communication)					
	Data sending	Bit map verify	Prog. by 256byte block	Erasing	Prog. + Verify		Erasing then Prog + Verify	
					Bit map	**Checksum***	Bit map	Checksum*
16	0.8s	1.2s	1.6s	6.5s ~ 8s	2.8s	**1.7s**	9.3s ~ 10.7s	8.2s ~ 9.7s
32	1.7s	2.3s	3.3s	8s ~ 9.5s	5.6s	**3.4s**	13.6s ~ 15.1s	11.4s ~ 12.9s
60	3.1s	4.1s	6.1s	11s ~ 13s	10.2s	**6.3s**	21.2s ~ 23.2s	17.3s ~ 19.3s

Note: The time spent for the verify can be optimized if a checksum computation is done instead of a bit map verification. This verify method can be done easily with the “Checksum” Embedded Command which requires ~3.75ms/kbytes to be computed.

Caution: As STMicroelectronics provides erased HDFlash devices, on the customer production line, only the programming and the verify time must be considered. Note that the option byte erasing/programming time is negligible.

The following tables give an idea of the device erase/program/verify times at frequency f_{CPU} = 8 MHz including the software overhead incurred using a standard programming tool:

Table 12. 60K Flash ST72521R9 FREQ parameter increasing from 1 to 8

	Flash			Option Bytes		
Operation	**Min.**	**Avg.**	**Max.**	**Min.**	**Avg.**	**Max.**
Erase (previous contents FF	15.33s	16.65s	19.24s	NA	NA	NA
Program (with all 00)	8.51s	8.93s	9.16s	549.3ms	739.8ms	865.3ms
Program (with all 00) + verify	15.94s	16.56s	16.98s	951.3ms	1.13s	1.295s
Erase (previous contents FF) + Program (with all 00) + verify	30.88s	32.66s	34.69s	NA	NA	NA
Read	6.82s	7.09s	7.268s	201.1ms	214.88ms	229.3ms

Table 13. 60K Flash ST72521R9 FREQ parameter = 8

	Flash		
Operation	**Min.**	**Avg.**	**Max.**
Erase (previous contents FF)	16.38s	16.5s	16.68s

Table 14. 32K Flash ST72324J/K6 FREQ parameter increasing from 1 to 8

	Flash			Option Bytes		
Operation	**Min.**	**Avg.**	**Max.**	**Min.**	**Avg.**	**Max.**
Erase (previous contents FF)	14.73s	16.36s	19.91s	NA	NA	NA
Program (with all 00)	5.048s	5.32s	5.59s	0.8377s	0.96s	1.066s
Program (with all 00) + verify	9.488s	9.85s	10.16s	1.106s	1.37s	1.717s
Erase (previous contents FF) + Program (with all 00) + verify	23.93s	25.75s	27.24s	NA	NA	NA
Read	4.094s	4.22s	4.35s	166.7ms	176.69ms	195.9ms

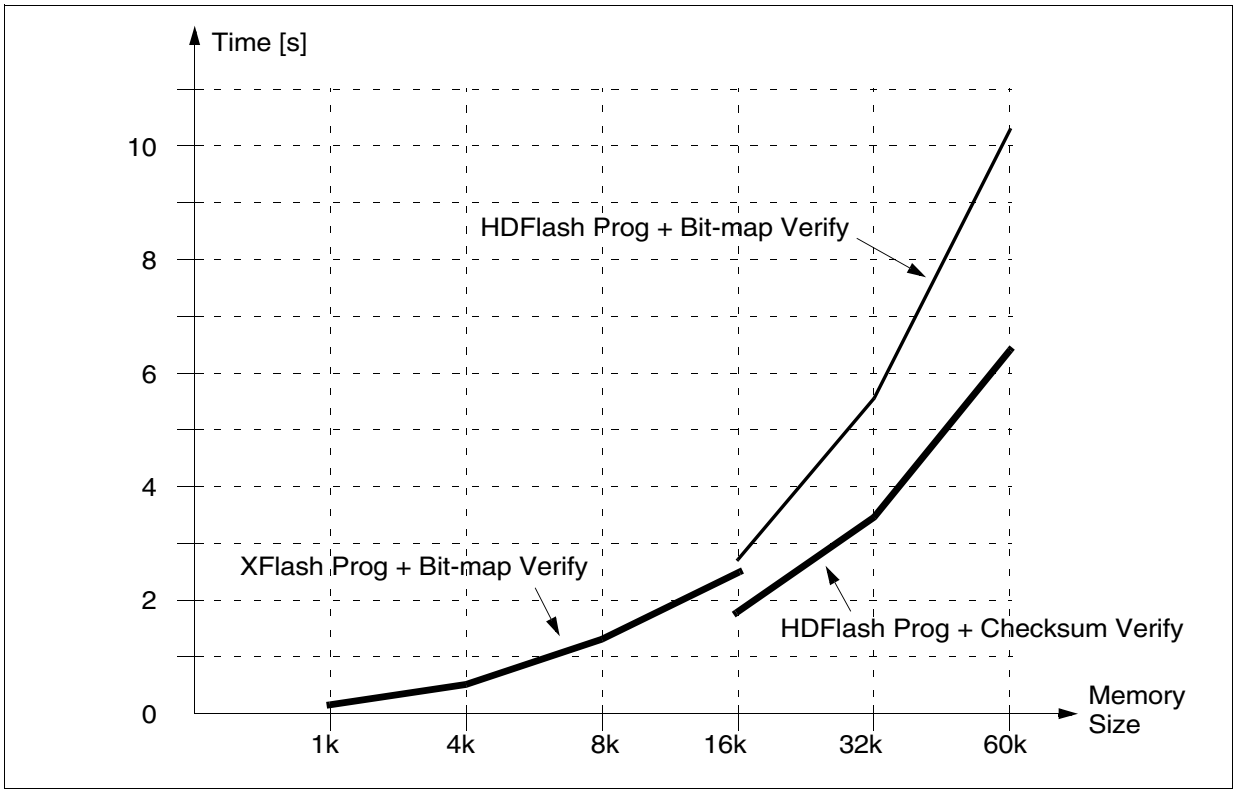
Table 15. 32K Flash ST72324J/K6 FREQ parameter = 8

	Flash		
Operation	**Min.**	**Avg.**	**Max.**
Erase (previous contents FF)	13.34s	14.18s	14.97s

Legend: NA = not available (operation not supported by STVP7 programming interface).

1.3 FLASH PROGRAMMING TIME SUMMARY

Figure 11. Typical ST7 FLASH Programming Times



APPENDIX 2 (XFlash PROGRAMMING SOURCE CODE EXAMPLE)

The purpose of this appendix is to provide an optimized source code example for XFlash programming. These examples are based on the ST72F264 device and use the ICC Protocol to receive or send the data (see “ICC Protocol Reference Manual” for more details).

2.1 FLASH PROGRAM MEMORY AND DATA EEPROM

The following example provides an optimum programming source code using the 32 byte parallelism capability of the XFlash. It illustrates the programming of the program memory space but it can be applied also to the data EEPROM area.

```
st7/
;*******************************************************************************
; MODULE:      pgm_xfpm.asm
; AUTHOR:      CMG_MCD Application Team
; UPDATE:      January 3rd, 2002
; DESCRIPTION: Optimized XFlash program memory space programming
;              using In-Circuit Communication protocol
;              (This example is based on a ST72F264 device)
;*******************************************************************************
; HIGH LEVEL ICC PROTOCOL EXAMPLE:
; 1. The controller sends the start address where to program in the XFlash
; 2. The controller sends the end address where to finish the programming
; 3. The controller sends the N data bytes to program (low address first)
;*******************************************************************************

  TITLE "MEMORY SPACE PROGRAMMING FOR ST72F264"

BYTES

  #DEFINE FCSR      $72     ; FLASH status control register

  #DEFINE STOPH     $80     ; Start Address High location
  #DEFINE STOPL     $81     ; Start Address Low  location
  #DEFINE CURRENTH  $82     ; End   Address High location
  #DEFINE CURRENTL  $83     ; End   Address Low  location

WORDS

  SEGMENT byte at 0082-00F7 'RAM1'

.main

; Enter RASS KEYS to access the XFLASH register ------------------------------

  LD      A,#$56            ; First key
  LD      FCSR,A
  LD      A,#$AE            ; Second key
  LD      FCSR,A

; Get Start and End Address of the XFlash Space to be Programmed -------------

  CALL    ICC_receive_byte  ; Get Start Address High
  LD      CURRENTH,A
  CALL    ICC_receive_byte  ; Get Start Address Low
  LD      CURRENTL,A
  CALL    ICC_receive_byte  ; Get End Address High
```

```
  LD      STOPH,A
  CALL    ICC_receive_byte  ; Get End Address Low
  LD      STOPL,A

; Data Latching loop ------------------------------------------------------

.latchenable
  BSET    FCSR,#1           ; Enable XFlash latches

.recvnextbyte
  CALL    ICC_receive_byte  ; Receive data byte to program and
  LD      [CURRENTH.w],A    ; store it in the corresponding XFlash latch
  LD      A,STOPL           ; Check if end address is reached
  CP      A,CURRENTL
  JRNE    endnotreached
  LD      A,STOPH
  CP      A,CURRENTH
  JRNE    endnotreached
  JRT     endreached

.endnotreached              ; When the end address is not reached
  INC     CURRENTL          ; compute the next latch address
  JRNE    checkprog         ; and then check if a programming pulse
  INC     CURRENTH          ; must be performed
  JRT     checkprog

; End of the programming sequence -----------------------------------------

.endreached
  BSET    FCSR,#0           ; Launch the last programming phase (PGM=1)
.progend                    ; and wait for the programming end (PGM=0)
  BTJT    FCSR,#0,progend

.endpgm                     ; END OF THE EXAMPLE: infinity loop
  JRT     endpgm

; Check if a programming pulse must be performed (end of a row) ---------------

.checkprog
  LD      A,#$1F
  AND     A,CURRENTL
  JRNE    recvnextbyte      ; End of the row not yet reached
                            ; then receive and latch next byte

; XFlash programming loop -------------------------------------------------

  BSET  FCSR,#0             ; Launch the programming phase (PGM=1)
.progloop                   ; and wait for the programming end (PGM=0)
  BTJT  FCSR,#0,progloop

  JRT   latchenable         ; Start the next row programming

;*******************************************************************************
  END
```

2.2 OPTION BYTES

The following source codes provide examples for an option byte read operation and an option byte write operation for an XFlash devices.

- Option byte read operation

```
st7/
;******************************************************************************
; MODULE:      pgm_xfor.asm
; AUTHOR:      CMG_MCD Application Team
; UPDATE:      January 3rd, 2002
; DESCRIPTION: Optimized XFlash option bytes read
;              using In-Circuit Commmunication protocol
;              (This example is based on a ST72F264 device)
;******************************************************************************
; HIGH LEVEL ICC PROTOCOL EXAMPLE:
; 1. The ST7 sends the read option byte 1
; 2. The ST7 sends the read option byte 0
;******************************************************************************

  TITLE "OPTION BYTES PROGRAMMING FOR ST72F264"

BYTES

  #DEFINE FCSR      $72         ; FLASH status control register

  #DEFINE STOPH     $80         ; Start Address High location
  #DEFINE STOPL     $81         ; Start Address Low  location
  #DEFINE CURRENTH  $82         ; End   Address High location
  #DEFINE CURRENTL  $83         ; End   Address Low  location

WORDS

  #DEFINE FLASH_BASE_ADD $E000  ; XFLASH Base Address

  SEGMENT byte at 0082-00F7 'RAM1'

.main

; Enter RASS KEYS to access the XFLASH register ------------------------------

  LD    A,#$56                  ; First key
  LD    FCSR,A
  LD    A,#$AE                  ; Second key
  LD    FCSR,A

  BSET  FCSR,#2                 ; Active option page to acccess options (OPT=1)

; Option byte read: Decoding from 32 bytes ------------------------------------

  LD    X,#$1F                  ; Read the two bytes
.readoptbit                     ; which correspond to an option bit
  LD    A,(FLASH_BASE_ADD,X)    ; and store the value in the carry
  LD    $80,A
  DEC   X
  LD    A,(FLASH_BASE_ADD,X)
  CP    A,#$FF
  JREQ  readone
.readzero
  RCF
  JRT   readcont
```

```
.readone
  SCF
.readcont
  RLC   Y                       ; Store the option bit value in Y register

; Check coherency between the two bytes -------------------------------------

  XOR   A,$80
  CP    A,#$01
  JRNE  readerror               ; Error detected !

; Send option bytes when ready ----------------------------------------------

  DEC   X
  CP    X,#$0F                  ; Check of the end of the first option byte
  JRNE  next                    ; End not reached
  LD    A,Y                     ; End reached...
  CALL  ICC_send_byte           ; => Send the option byte 1 to the controller
  JRT   readoptbit              ; Next bit to read...
.next
  TNZ   X                       ; Check of the end of the second option byte
  JRPL  readoptbit              ; End not reached => next bit to read
  LD    A,Y                     ; End reached...
  CALL  ICC_send_byte           ; => Send the option byte 0 to the controller

; End of the program --------------------------------------------------------

.endpgm                         ; END OF THE EXAMPLE: infinity loop
  JRT   endpgm

; Option Byte Read error management -----------------------------------------

.readerror                      ; Error of coherency detected when reading the
                                ; 2 bytes which correspond to an option bit
  ; < Add here the error management >
  JRT   readerror

;*******************************************************************************
  END
```

- Option byte write operation

```
st7/
;*******************************************************************************
; MODULE:      pgm_xfop.asm
; AUTHOR:      CMG_MCD Application Team
; UPDATE:      January 3rd, 2002
; DESCRIPTION: Optimized XFlash option bytes programming
;              using In-Circuit Commmunication protocol
;              (This example is based on a ST72F264 device)
;*******************************************************************************
; HIGH LEVEL ICC PROTOCOL EXAMPLE:
; 1. The controller sends the option byte 1 to be programmed
; 2. The controller sends the option byte 0 to be programmed
;*******************************************************************************

  TITLE "OPTION BYTES READ FOR ST72F264"

BYTES

  #DEFINE FCSR      $72         ; FLASH status control register
```

```
  #DEFINE STOPH     $80         ; Start Address High location
  #DEFINE STOPL     $81         ; Start Address Low  location
  #DEFINE CURRENTH  $82         ; End   Address High location
  #DEFINE CURRENTL  $83         ; End   Address Low  location

WORDS

  #DEFINE FLASH_BASE_ADD $E000  ; XFLASH Base Address

  SEGMENT byte at 0082-00F7 'RAM1'

.main

; Enter RASS KEYS to access the XFLASH register ------------------------------

  LD    A,#$56                  ; First key
  LD    FCSR,A
  LD    A,#$AE                  ; Second key
  LD    FCSR,A

  BSET  FCSR,#2                 ; Active option page to acccess options (OPT=1)


; Preparing the Option Byte LATCHES ------------------------------------------

  BSET  FCSR,#1                 ; Active option byte latches (LAT=1)

  LD    A,#$FF                  ; First, set all 32 latches to $FF
  LD    X,#$1F
.prepFF
  LD    (FLASH_BASE_ADD,X),A
  DEC   X
  JRPL  prepFF


; Get Option Byte #1 ---------------------------------------------------------

.getopt1
  CALL  ICC_receive_byte
  LD    X,A

; Compute and latch the 2 bytes corresponding to an option bit ----------------

  LD    Y,#$1F                  ; Y register is used as byte latch counter
nextoptbit
  LD    A,#$FE
  RLC   X
  JRNC  itis1
itis0
  LD    (FLASH_BASE_ADD,Y),A
  DEC   Y
  JRT   continue
itis1
  DEC   Y
  LD    (FLASH_BASE_ADD,Y),A

; Check the status of the decoding operation ---------------------------------

continue
  DEC   Y
  CP    Y,#$0F                  ; Check if the option byte 1 has been coded
  JREQ  getopt0                 ; If yes, then get the option byte 0
  TNZ   Y                       ; Check if the option byte 0 has been coded
```

```
  JRPL  nextoptbit              ; If no, then decode next bit

; Launch and wait XFlash programming ----------------------------------------

  BSET  FCSR,#0                 ; Launch programming (PGM=1)
progloop                        ; and wait for the end of the programming pulse
  BTJT  FCSR,#0,progloop

; End of the program --------------------------------------------------------

.endpgm                         ; END OF THE EXAMPLE: infinity loop
  JRT   endpgm

; Get Option Byte #0 --------------------------------------------------------

getopt0
  CALL  ICC_receive_byte
  LD    X,A
  JRT   nextoptbit

;*******************************************************************************
  END
```

APPENDIX 3 (HDFlash PROGRAMMING SOURCE CODE EXAMPLE)

The purpose of this appendix is to provide an optimized source code example for HDFlash programming. This example is based on the ST72F521 device and uses the ICC Protocol to receive or send the data (see “ICC Protocol Reference Manual” and section 3.5 on page 38 for more details).

```
st7/
;******************************************************************************
; MODULE:      pgm_hdfm.asm
; AUTHOR:      CMG_MCD Application Team
; UPDATE:      October 11th, 2002
; DESCRIPTION: Optimized HDFlash program memory space programming
;              using In-Circuit Commmunication protocol
;              (This example is based on a ST72F521 device)
;******************************************************************************
; HIGH LEVEL ICC PROTOCOL EXAMPLE:
; 1. The controller sends the start address where to program in the XFlash
; 2. The controller sends the end address where to finish the programming
; 3. The controller sends the N data bytes to program (low address first)
; 4. The ST7 sends a status ($AA = succeeded)
;******************************************************************************

  TITLE "MEMORY SPACE PROGRAMMING FOR ST72F521"

BYTES

  #DEFINE FCSR        $29   ; FLASH status control register

  #DEFINE FLASH_FREQ  $F8   ; HD FLASH embedded command parameters
  #DEFINE FLASH_DATA  $F9
  #DEFINE FLASH_ENDH  $FA
  #DEFINE FLASH_ENDL  $FB
  #DEFINE FLASH_PTRH  $FC
  #DEFINE FLASH_PTRL  $FD
  #DEFINE FLASH_SECT  $FE
  #DEFINE FLASH_ECMD  $FF

  #DEFINE STOPH       $80   ; Start Address High location
  #DEFINE STOPL       $81   ; Start Address Low  location
  #DEFINE CURRENTH    $82   ; End   Address High location
  #DEFINE CURRENTL    $83   ; End   Address Low  location

  #DEFINE STATUS      $84   ; Returned status location
  #DEFINE ENDPROG     $85   ; End of programming flag location

  #DEFINE BUFF_START  $0200 ; RAM Buff definition for block programming
  #DEFINE BUFF_SIZE   0     ; RAM Buff size (from 1 to 255 and 0 for 256 bytes)

WORDS

  SEGMENT byte at 0082-00F7 'RAM1'

.main

; Enter RASS KEYS to access the HDFlash register -----------------------------

  LD      A,#$56            ; First key
  LD      FCSR,A
```

```
  LD      A,#$AE            ; Second key
  LD      FCSR,A

; Get Start and End Address of the HDFlash Space to be Programmed ------------

  CALL    ICC_receive_byte  ; Get Start Address High
  LD      FLASH_PTRH,A      ; Stored in parameters for first programming
  LD      CURRENTH,A

  CALL    ICC_receive_byte  ; Get Start Address Low
  LD      FLASH_PTRL,A      ; Stored in parameters for first programming
  LD      CURRENTL,A

  CALL    ICC_receive_byte  ; Get End Address High
  LD      STOPH,A

  CALL    ICC_receive_byte  ; Get End Address Low
  LD      STOPL,A

; Set status and end flag -----------------------------------------------------

  LD      A,#$AA            ; Status   =0xAA: Ok / <>0xAA: Error
  LD      STATUS,A
  CLR     ENDPROG           ; End flag =0x00: No / <>0x00: End

; Initialization of a RAM buffer to be programmed -----------------------------

.nextblocprog
  CLR     X

; Fill the RAM buffer to be programmed ----------------------------------------

.nextbyte2recv

  LD      Y,X               ; Rem: X is used in ICC_receive_byte routine
  CALL    ICC_receive_byte  ; Get Data to program
  LD      X,Y
  LD      (BUFF_START,X),A  ; and store it in the RAM buffer

  LD      A,STOPH           ; Check if end address reached
  CP      A,CURRENTH        ; and increment current ptr
  JRNE    inccurrentptr
  LD      A,STOPL
  CP      A,CURRENTL
  JRNE    inccurrentptr
  INC     ENDPROG           ; End of progrogramming reached

.inccurrentptr              ; Increment current pointer for next byte
  INC     CURRENTL
  JRNE    checkend
  INC     CURRENTH

.checkend                   ; Check if the RAM buffer is full
  INC     X
  TNZ     ENDPROG
  JRNE    next              ; RAM buffer full => Programming sequence

.checkbuffsize
  CP      X,#BUFF_SIZE
  JRNE    nextbyte2recv     ; RAM buffer not yet full => next byte to receive

; RAM buffer programming sequence ---------------------------------------------
```

```
.next
  JP      RAM3_start
  SEGMENT byte at 0100-013F 'RAM3'
.RAM3_start
  CLR     FLASH_FREQ
.progseq
  INC     FLASH_FREQ          ; To optimize the programming time the FREQ
  LD      A,FLASH_FREQ        ; parameter is incremented from 1 to 8
  CP      A,#8                ; (see note 1 page 20)
  JRUGT   error
  LD      A,#$01              ; HDFlash Block programming embedded command
  LD      FLASH_ECMD,A
  LD      FLASH_SECT,X        ; * Set number of bytes to program
  LD      A,#BUFF_START.l     ; * Set RAM buffer start address
  LD      FLASH_ENDL,A
  LD      A,#BUFF_START.h
  LD      FLASH_ENDH,A

.progloop
  LD      FCSR,A              ; Launch embalgo
  LD      A,$FF               ; Get status
  BCP     A,#$70              ; Check status
  JRNE    progseq             ; if OK then continue programming
.noerr                        ; if FAIL then trial with next FREQ value
  BTJT    $FF,#7,progloop     ; Loop while prog is not completed
  JRT     nextprog
.error
  LD      A,$FF
  LD      STATUS,A
.nextprog
  TNZ     ENDPROG             ; Test if the programming end is reached
  JRNE    endreached
  LD      A,CURRENTL          ; Prepare next Flash start address to program
  LD      FLASH_PTRL,A
  LD      A,CURRENTH
  LD      FLASH_PTRH,A

  JP      nextblocprog

; End of the HDFlash programming ------------------------------------------

.endreached
  LD      A,STATUS            ; Send the status to the controller
  CALL    ICC_send_byte

.endpgm                       ; END OF THE EXAMPLE: infinity loop
  JRT     endpgm

;*****************************************************************************
  END
```

APPENDIX 4 (EMULATED DATA EEPROM WITH XFlash MEMORY)

When the data EEPROM is not available in a ST7 device, it can be emulated by the XFlash memory with some restrictions. This appendix describes how to emulate this feature with a ST72FLite05 device and the restrictions this emulation implies.

4.1 RESTRICTIONS

- To guarantee that the XFlash program memory is write protected when programming the Emulated Data EEPROM, the whole program memory MUST be located in Sector 0. This implies that:
  - The maximum program memory size is 1kbyte (sector 0 set by option byte to the maximum size which allows Sector 1 availability).
  - IAP is not available for program memory.
- During emulated data EEPROM programming, the XFlash cannot be executed. This implies that:
  - The software which programs the emulated data EEPROM must be located in RAM. This software needs at least 16 bytes of RAM as shown in following program example.
  - The interrupts cannot be served during programming so they have to be masked.

4.2 PROCEDURE

To program 1 byte in the emulated data EEPROM (located in Sector 1) the following steps have to be done:

- Enter the XFlash RASS key to unlock the access to the FCSR register (only once, after reset for example).
- Download the programming driver into RAM (from 0083h to 008Fh for example).
- Write the data and address to be programmed in a RAM buffer (at RAM address 0080h to 0082h for example).
- Call the downloaded RAM driver to program the emulated data EEPROM.

4.3 ASSEMBLER PROGRAM EXAMPLE

The following program example describes a driver routine to be called to emulate data EEPROM with an XFlash ST7 device. This example assumes that all restrictions are taken into account.

```
st7/
;*****************************************************************************
; TITLE:       XdataE2Emul.asm
; AUTHOR:      CMG_MCD Application Team
; DESCRIPTION: Data EEPROM emulation with XFlash memory (ST72FLite05 example)
;*****************************************************************************
```

```
  TITLE "XdataE2Emul.asm"

BYTES

FCSR   EQU $2F ; XFLASH Control/Status register definition
  #DEFINE LAT   1
  #DEFINE PGM   0

E2DATA EQU $80 ; 1 byte:  Emul. EEPROM Data to be programmed
E2ADDR EQU $81 ; 2 bytes: Emul. EEPROM Address to be programmed

WORDS

SEGMENT byte at FA00-FBFF 'XFlash Sect1 -Emul Data EEPROM'

SEGMENT byte at FC00-FFFF 'XFlash Sect0 -Program'

; < RESET >

  LD  A,#$56  ; Enter RASS keys to unlock FCSR register
  LD  FCSR,A
  LD  A,#$AE
  LD  FCSR,A

; < USER APPLICATION PROGRAM >

  CALL XemulE2_ByteProg

; < USER APPLICATION PROGRAM >

; ------------------------------------------------------------------------------
; ROUTINE: XemulE2_ByteProg
; DESCRIPTION: Emulated data EEPROM byte programming driver routine
; BEFORE: A   = data to be programmmed
;         X:Y = address where it has to be programmed [FA00h..FBFFh]
; AFTER:  Interrupt are disabled
;         The requested data byte is programmed
; RESSOURCES:
;         Program size:  40 bytes in sector 0
;         Used RAM area: 16 bytes from 0080h to 008Fh.
; ------------------------------------------------------------------------------

.XemulE2_ByteProg
  LD    E2DATA,A              ; Data         to be programmed (0080h) is in A
  LD    {E2ADDR},X            ; Address high to be programmed (0081h) is in X
  LD    {E2ADDR+1},Y          ; Address low  to be programmed (0082h) is in Y

  LD    X,#$0C                ; Copy programming software driver
.RAM_Copy                     ; into RAM from address 0083h
  LD    A,(RAM_Driver,X)
  LD    ($83,X),A
  DEC   X
  JRPL  RAM_Copy

  SIM                         ; Disable interrupts
  JP $83                      ; Call the programming driver located into RAM

.RAM_Driver
  BSET  FCSR,#LAT             ; Enable Emul. EEPROM latches
  LD    A, E2DATA
  LD    [E2ADDR.w],A          ; Set address/data to be programmed
  BSET  FC  SR,#PGM           ; Launch the Emul. EEPROM programming
.EEPROM_Prog
```

```
  BTJT  FCSR,#PGM,EEPROM_Prog  ; Wait end of programming (~5ms)
  RET

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  END
```

APPENDIX 5 (XFlash ICP DRIVER SOFTWARE EXAMPLE)

ICP Driver: programming phase

```
st7/
;******************************************************************************
; TITLE:        ICP_Prog_Drv.asm
; AUTHOR:       CMG_MCD Application Team
; DESCRIPTION: In-Circuit Programming
;******************************************************************************

  TITLE "icp_prog_drv.asm"

  BYTES

.FCSR     EQU   $26    ; XFlash control/status register (device dependant,
                       ; refer to device specification for address details)
  #DEFINE PGM   0      ; Bit definitions of FCSR
  #DEFINE LAT   1

.STRTADD  EQU   $82    ; XFlash Start Address (if MSB=0 then end of programming)
.RAMBUFF  EQU   $84    ; 32 byte RAM buffer
                       ; ICC protocol used RAM space: 2 bytes (80h and 81h)
  WORDS

;******************************************************************************
;  LOW LEVEL ICC PROTOCOL SUBROUTINES
;******************************************************************************

.ICC_receive_byte EQU $xxxx ; This routine is available in System Memory area
                            ; and does not need to be copied into RAM.
                            ;(refer to "ICC Protocol" reference manual for
                            ; address details)
        ; Subroutine used by the ST7 to receive a byte from ICCDATA pin
        ; before start:
        ;        CC.H (half carry flag) must be cleared
        ; before start and after end of subroutine:
        ;        ICCCLK = 1 as floating input
        ;        ICCDATA as floating input
        ;        bit ICCCLK on Port x Data Register cleared
        ; after end of subroutine:
        ;        Acc A = byte received
        ;        Idx X = 0

;******************************************************************************
;  ICP DRIVER
;******************************************************************************

  SEGMENT byte at 00B0-00FF 'icp_prog_drv'

.ICP_Prog_Drv
  call  ICC_receive_byte      ; Receive 1st RASS key from external controller
  ld    FCSR,A
  call  ICC_receive_byte      ; Receive 2sd RASS key from external controller
  ld    FCSR,A

.ICP_Programming
  call  ICC_receive_byte      ; Receive XFlash MSB @ byte to be programmed
  tnz   A
  jreq  ICP_End               ; If MSB=0 then end of programming procedure
  ld    STRTADD,A
```

```
  call  ICC_receive_byte      ; Receive XFlash LSB @ byte to be programmed
  ld    {STRTADD+1},A

  ld    Y,#32                 ; Receive 32bytes using ICC and store it into RAM
.ICP_RAMLoadLoop              ; (high address byte first)
  call  ICC_receive_byte      ; Receive data byte to be programmed
  ld    (RAMBUFF,Y),A
  dec   Y
  jrne  ICP_RAMLoadLoop

.ICP_WaitPgm                  ; Wait end of previous programming phase
  btjt  FCSR,#PGM,ICP_WaitPgm

  bset  FCSR,#LAT             ; Copy 32 data byte from RAM buffer to EEPROM
latches
  ld    X,#32
.ICP_LatchLoadLoop
  ld    A,(RAMBUFF,X)
  ld    ([STRTADD.w],X),A
  dec   X
  jrne  ICP_LatchLoadLoop

  bset  FCSR,#PGM             ; Launch programming phase
  jrt   ICP_Programming       ; Next programming

.ICP_End
  jp    ICC_monitor ; End of ICP driver return to ICC monitor or infinite loop

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  END
```

ICP Driver: verify phase

```
st7/
;******************************************************************************
; TITLE:       icp_verif_drv.asm
; AUTHOR:      CMG_MCD Application Team
; DESCRIPTION: In-Circuit Programming  Verify Procedure
;******************************************************************************

  TITLE "icp_verif_drv.asm"

  BYTES

.PxDR     EQU $08   ; I/O port definition for ICC management (device dependant,
.PxDDR    EQU $09   ; refer to datasheet for address details)
.PxOR     EQU $0A
  #DEFINE ICCCLK  0   ; Bit definitions for ICC pin allocation
  #DEFINE ICCDATA 1

.STRTADD  EQU $82     ; XFlash Start Address Pointer
                      ; ICC protocol used RAM space: 2 bytes (80h and 81h)
  WORDS

;******************************************************************************
;  LOW LEVEL ICC PROTOCOL SUBROUTINES
;******************************************************************************

.ICC_receive_byte EQU $xxxx ; This routine is available in System Memory area
                            ; and does not need to be copied into RAM.
                            ;(refer to "ICC Protocol" reference manual for
```

```
                              ; address details)
        ; Subroutine used by the ST7 to receive a byte from ICCDATA pin
        ; before start:
        ;       CC.H (half carry flag) must be cleared
        ; before start and after end of subroutine:
        ;       ICCCLK = 1 as floating input
        ;       ICCDATA as floating input
        ;       bit ICCCLK on Port x Data Register cleared
        ; after end of subroutine:
        ;       Acc A = byte received
        ;       Idx X = 0
  SEGMENT byte at 0090-00FF 'icp_prog_drv'

; ---------------------------------------------------------------------------
.ICC_send_byte          ; This routine must be implemented in this driver
                        ; when it is not implemented in the System Memory.
        ; Subroutine used by the ST7 to send a byte on ICCDATA
        ; before start and after end of subroutine:
        ;       ICCCLK = 1 as floating input
        ;       ICCDATA as floating input
        ; before start of subroutine:
        ;       Acc A = byte to send
        ; after end of subroutine:
        ;       Idx X = 0

  LD    X,#$08                ; X = BitCounter
  BSET  PxDDR,#ICCDATA        ; Define ICCDATA as open-drain output

.icc_s1
  BSET  PxDR, #ICCDATA        ; ICCDATA=1 because tying line to 0 is faster
  RLC   A                     ; Copy bit to transmit into carry flag
  JRC   icc_s2                ; Test bit to transmit
  BRES  PxDR, #ICCDATA        ; If "0" set ICCDATA = 0
.icc_s2
  TNZ   A                     ; Dummy instructions to get a 2 x 3 Tcpu delay
  TNZ   A                     ; to insure the ICCDATA level
  BRES  PxDR ,#ICCCLK         ; Because BSET/BRES ICCDATA may set ICCCLK DR
  BSET  PxDDR,#ICCCLK         ; => ICCCLK output = 0
  TNZ   A                     ; Dummy instructions to get a 4 x 3 Tcpu delay
  TNZ   A                     ; for external controller reaction time
  TNZ   A
  TNZ   A

  BRES  PxDDR,#ICCCLK         ; Release ICCCLK after 5 Tcpu

.icc_s3
  BTJF  PxDR ,#ICCCLK,icc_s3  ; Wait until ICCCLK = 1
                              ; Ext. controller already detected 1 (TTL input)
  DEC   X
  JRNE  icc_s1                ; Check if the whole byte is sent

  BRES  PxDDR,#ICCDATA        ; If yes, define ICCDATA as floating input

  RET

;*****************************************************************************
;  ICP VERIFY
;*****************************************************************************

.ICP_Verif_Drv

  call  ICC_receive_byte    ; Receive XFlash MSB start @ byte to be read
  tnz   A
```

```
  jreq  ICP_End             ; If MSB=0 then end of verify procedure
  ld    STRTADD,A
  call  ICC_receive_byte    ; Receive XFlash LSB start @ byte to be read
  ld    {STRTADD+1},A
  ; Verify 256 data byte into the memory
  clr   Y
.ICP_VerifLoop
  ld    A,([STRTADD.w],Y)
  call  ICC_send_byte    ; send data byte to be verify
  inc   Y
  jrne  ICP_VerifLoop
  JRT   ICP_Verif_Drv       ; Next verify
.ICP_End
  jp    ICC_monitor ; End of ICP driver return to ICC monitor or infinite loop

;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  END
```

APPENDIX 6 (EMULATED DATA EEPROM WITH HDFlash MEMORY)

ST7 HDFlash devices do not feature data EEPROM, but they can emulate it with certain restrictions. This appendix describes how to emulate this feature and the restrictions that apply.

As HDFlash is a dual voltage FLASH memory, the 12-volt programming voltage must be provided on the application board (see section 3.3 on page 36 for more details).

6.1 PRINCIPLE

Two implementations can be distinguished:

- Assuming that it is possible to limit the data EEPROM byte values to 00h..FEh (as FFh is the default HDFlash erased value), the principle of this emulation is to reserve n bytes in the HDFlash for each emulated data EEPROM byte which has to be cycled n times. With this solution, the FFh value can not be used.
  As shown in Figure 12, a “ptr” pointer gives the access to the emulated data EEPROM byte.
  – For a read operation, from “ptr” address, read the first data byte not equal to FFh to get the current value.
  – For a write operation, from “ptr” address, look for the last data byte equal to FFh and then write the new value at this location using the HDFlash “Byte programming” Embedded Command (same as IAP method).
- Assuming no limitation for the data EEPROM byte values (00h..FFh), each byte value will need 2 byte locations. The first one will determine if it is the current value (example: 00h = current value, FFh = not yet used value) and the second one will contain the current byte value. So the principle of this emulation is to reserve 2xn bytes in the HDFlash for each emulated data EEPROM byte which has to be cycled n times.
  As shown in Figure 13, a “ptr” pointer gives the access to the emulated data EEPROM byte.
  – For a read operation, from “ptr” address, read the first data byte not equal to FFh (equal to 00h), the current value is the next adjacent byte.
  – For a write operation, from “ptr” address, look for the last data byte equal to FFh and then write the new value at this location and 00h in the previous location using the HDFlash “Byte programming” Embedded Command (same as IAP method).

For both implementations, the programming method is address decreasing. Therefore, the emulated data EEPROM values must be initialized (see Figure 12 and Figure 13). This can be done on the production line (using ICP for example) or calling a dedicated user application routine (label `HDemulE2_ByteProgInit` in the assembler example in "6.4 ASSEMBLER PROGRAM EXAMPLE").

Figure 12. HDFlash Emulated Data EEPROM (value in 00h..FEh)

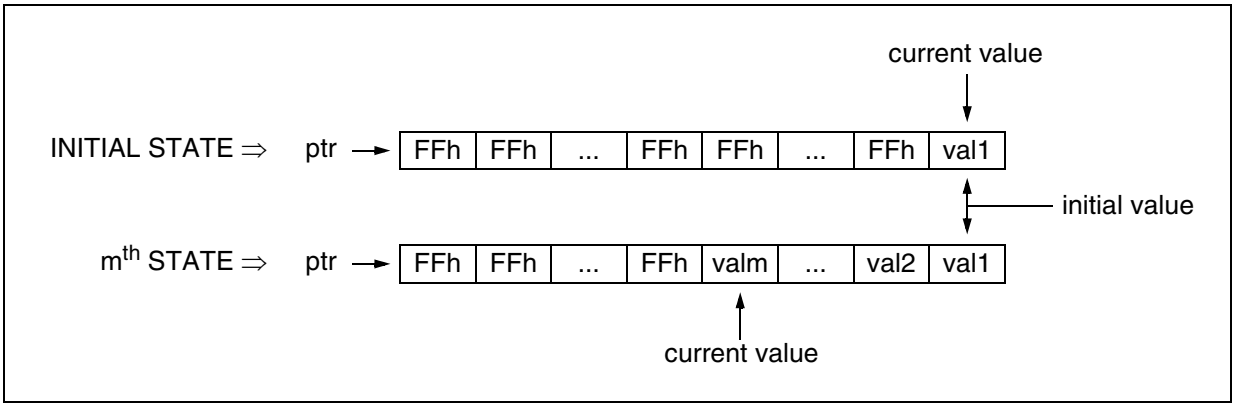
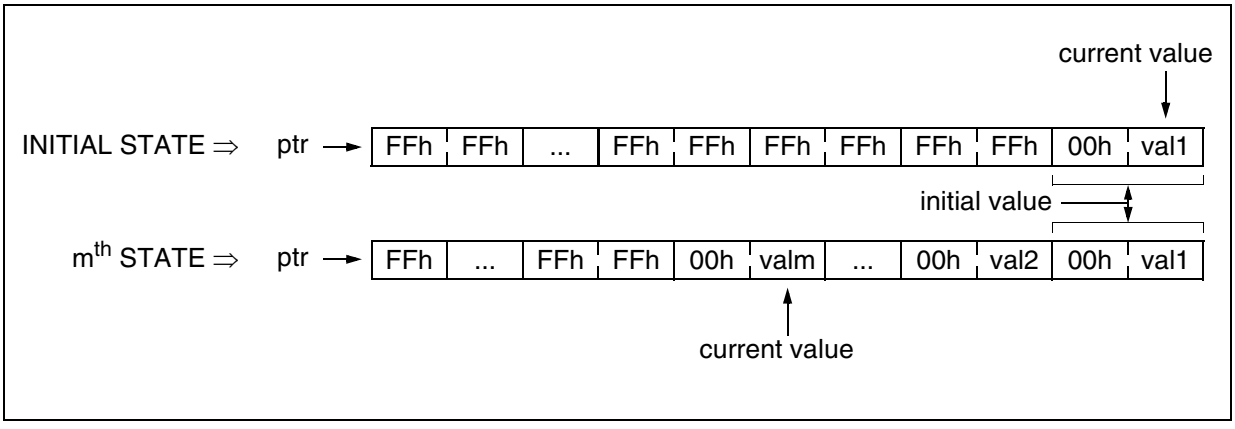


Figure 13. HDFlash Emulated Data EEPROM (value in 00h..FFh)



6.2 RESTRICTIONS

- Only a few data EEPROM bytes with a limited number of write/erase cycles can be emulated (these characteristics are directly linked to the needed memory space in HDFlash).
- Emulated data EEPROM must be located in Sectors 1 or 2 as Sector 0 is write protected in USER mode.
- To use the Embedded Commands the RASS protection must be disabled. So the protection against unintentional access to the HDFlash control register is no longer available. Unintentional programming is only guaranteed by the low probability of accidentally executing an Embedded Command with V_{PP} at 12 volts. See section 1 on page 8 for more details.

6.3 ADVANTAGE

- The HDFlash emulated data EEPROM method can keep the data byte value history.

6.4 ASSEMBLER PROGRAM EXAMPLE

The following program example describes a driver routine to be called to emulate data EEPROM with a HDFlash ST7 device (data value limited to 00h..FEh). This example assumes that all restrictions are taken into account.

```
st7/

;*******************************************************************************
; TITLE:       HDdataE2Emul.asm
; AUTHOR:      CMG_MCD Application Team
; DESCRIPTION: Data EEPROM emulation with HDFlash memory (ST72F521 example)
;*******************************************************************************

  TITLE "HDdataE2Emul.asm"

  BYTES

FCSR        EQU   $29 ; HDFLASH Control register definition

FLASH_CMD   EQU   $FF   ; Reserved RAM area for HDFLASH Embedded Commands
FLASH_SECT  EQU   $FE
FLASH_PTRL  EQU   $FD   ;     PTRH:PTRL is also used for read operation
FLASH_PTRH  EQU   $FC
FLASH_ENDL  EQU   $FB
FLASH_ENDH  EQU   $FA
FLASH_DATA  EQU   $F9
FLASH_FREQ  EQU   $F8

  WORDS

  SEGMENT WORD AT 1000-DFFF 'HDFlash Sect2'

  < MEMORY LOCATION OF THE EMULATED DATA EEPROM : FROM 8000h IN THIS EXAMPLE >

  SEGMENT WORD AT E000-EFFF 'HDFlash Sect1'

  < POTENTIAL MEMORY LOCATION FOR EMULATED DATA EEPROM >


  SEGMENT WORD AT F000-FFFF 'HDFlash Sect0'

; < RESET >

  LD  A,#$56  ; Enter RASS keys to unlock FCSR register
  LD  FCSR,A
  LD  A,#$AE
  LD  FCSR,A

; < USER APPLICATION PROGRAM >

  LD   A, #$80             ; Read emulated data EEPROM value with a pointer
  LD   FLASH_PTRH, A       ; located at address 8000h (sector 2).
  LD   A, #$00
  LD   FLASH_PTRL, A

  CALL HDemulE2_ByteRead   ; Result: current value in A register

; < USER APPLICATION PROGRAM >
```

```
  LD   A, #$80              ; Read emulated data EEPROM value with a pointer
  LD   FLASH_PTRH, A        ; located at address 8000h (sector 2).
  LD   A, #$00
  LD   FLASH_PTRL, A

  LD   A, #$55              ; 55h is the new data to be write in emulated data
  LD   FLASH_DATA, A        ; EEPROM byte located at 8000h address

  CALL HDemulE2_ByteProg    ; Result: status returned in FLASH_CMD high nibble

; < USER APPLICATION PROGRAM >

; ------------------------------------------------------------------------------
; ROUTINE: HDemulE2_ByteRead
; DESCRIPTION: Emulated data EEPROM byte read driver routine
; BEFORE: FLASH_PTRH:L = pointer corresponding to the start address of the
;                        memory table allocated to the emulated data EEPROM
;                        byte to read
; AFTER:  A            = current emulated data EEPROM byte value
;         FLASH_PTRH:L = address of the current emulated data EEPROM byte value
; RESSOURCES:
;         Program size:  15 bytes
;         Used RAM area: 2 bytes for PTRH:PTRL.
; ------------------------------------------------------------------------------

.HDemulE2_ByteRead
  LD    A,[FLASH_PTRH.w]          ; Read emulated data EEPROM table content
  CP    A, #$FF                   ; If the value is FFh check next location
  JRNE  HDemulE2_ByteRead_end     ; Else A = current value.
  INC   FLASH_PTRL
  JRNE  HDemulE2_ByteRead
  INC   FLASH_PTRH
  JRT   HDemulE2_ByteRead
.HDemulE2_ByteRead_end
  RET

; ------------------------------------------------------------------------------
; ROUTINE: HDemulE2_ByteProg
; DESCRIPTION: Emulated data EEPROM byte programming driver routine
; BEFORE: FLASH_PTRH:L = pointer corresponding to the start address of the
;                        memory table allocated to the emulated data EEPROM
;                        byte to update
;         FLASH_DATA = data to be programmmed
; AFTER:  FLASH_CMD  = status result of the programming (see embedded status
;                      return code definition of the Embedded Command)
; RESSOURCES:
;         Program size:  24 bytes
;         Used RAM area: same as HDFlash Embedded Command
;                        - 16 bytes from 00F0h to 00FFh
;                        - 124 byte stack
;                        (see dedicated chapter for more details)
; ------------------------------------------------------------------------------

.HDemulE2_ByteProg
  CALL  HDemulE2_ByteRead         ; To look for the current value address
  CP    A, FLASH_DATA             ; If the data to be written is the same as
  JREQ  HDemulE2_ByteProg_end     ; current one, skip programming
  TNZ   FLASH_PTRL                ; Set the new current value address
  JRNE  HDemulE2_ByteProg_next    ; => previous current - 1
  DEC   FLASH_PTRH
.HDemulE2_ByteProg_next
  DEC   FLASH_PTRL
```

```
  CLR   FLASH_CMD                 ; Set Embedded Command to "Byte programming"
  CLR   FLASH_FREQ                ; FREQ parameter trial for an optimum
.freqtrial                        ; programming time
  INC   FLASH_FREQ
  LD    A,FLASH_FREQ
  CP    A,#8
  JRUGT HDemulE2_ByteProg_end
  LD    FCSR,A                     ; Launch the HDFlash Embedded Command

.HDemulE2_ByteProg_end
  RET

;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  END
```

Index

SUMMARY OF CHANGES

Date	Rev	Description
Feb-2001	1.0	First version of the "ICC & Flash Programming" reference manual.
Sept-2001	1.1	Document reduced to "Flash Programming" reference manual and creation of the "ICC Protocol" reference manual.
Mar-2002	1.2	Page 1 Formating of the first page modified and date / revision added. Page 1 ST72FLite00, ST72F561 and ST7FHUB products added. Page 14 In Figure 4, coding of 0 and 1 has been reversed and ST72F264 base address example corrected from F0h to E0h. Page 17 Description added for the OPT bit (OPT and LAT bits can not be set together, this operation must be done in two steps) Page 24 In Table 3, FREQ column has be changed with [1..8] and the note 1 updated accordingly. CAUTION 2 added to mention the return status check. "Command Timeout" chapter re-ordered for beter legibility. Page 24 CAUTION added to indicates that returned status code must be checked after each Embedded Command execution. ... FREQ parameter changed to 1 for all Embedded Command examples. Page 27 In Table 6, "Returned status" changed for "Byte programming", "Block programming" and "Sector erasing". Page 27 Note 2 clarified. Page 29 Note 4 clarified. New note describing how to program 256 bytes with block programming command. Page 32 CAUTION added for the option byte erasing. Page 32 Note added for option byte read and checksum Embedded Commands. Page 36 In Table 9, STT3PF30L transistor reference added and Vout reference versus IO value. Page 37 Updated timings with FREQ parameter set to 1. New notes describing the optimum conditions to get the typical timing specified. Page 39 Add a cross reference to APPENDIX 3 for ICP code example. Page 41 Updated timings with optimized programming methods Page 43 Flash programming time curves added Page 44 Appendix "XFlash programming source code example" added Page 50 Appendix "HDFlash programming source code example" added Page 53 Address of FCSR register changed from $09 to $2F. Page 53 New chapter to describe the emulated data Eeprom initialization. Page 47 RIM instruction replaced by SIM instruction. Page 58... In code example: - Change FCR register name to FCSR. - FLASH_PTRH changed to FLASH_PTRL. - `HDemulE2_ByteProgInit` label added. Page 66 Add of the "SUMMARY OF CHANGES" table.
Jun-2002	1.3	Page 11 Add one sentence at the end of the "Important note" Page 20 In table 3, FREQ column notes have been modified. Note 1: added information for FREQ parameter when over 60°C. Note 7: added for FREQ parameter with erase command. Note 8: added for FREQ parameter with read option byte and checksum commands. Page 21 Last sentence of the "Caution" modified for better legibility. Page 23 In table 6, return status 19h is removed for option byte erasing command. Page 27 In sector erasing code example, the FREQ parameter value has been modified.

Date	Rev	Description
Aug-2002	1.4	Page 21 Stack size needed for HDFlash Embedded Commands execution modified to 124bytes. Page 22 Updated Embedded Command timeouts table 5. Page 30 Added CAUTION for the duration of the 12V applied on Vpp pin.
Oct-2002	1.5	Page 11 Figure 4 modified for a better legibility and to be in-line with datasheets and development tools description of the option bytes. Page 20 Note 7 removed and note 1 modified: f_{CPU} parameter can also be set to 1 for erase command. In note 1, the case of f_{CPU}>8MHz as been mentioned. In table 3, fcpu removed from FREQ parameter command. Page 22 In table 5, "NA" has been removed and replaced by the value for erasing with FREQ parameter set to 1. Page 27 In "Sector erasing" example, FREQ has been modified to 1. Page 35 HDFlash erasing time values updated with FREQ parameter set to 1.Page 44 In Appendix 3, FREQ parameter management has been updated according to the note 1 page 20. Page 55 In Appendix 6, FREQ parameter management has been updated according to note 1 page 20.
Mar-2003	1.6	Page 1 Changed ST7FLITE00 to ST7Superlite, added ST7FSCR Page 11 Added 'At the end....OPT bit must be cleared by S/W' and 'Option bits are taken into account...' Page 22 Added caution 'Only one status bit ....'. Page 25 Changed "As soon as this option bit is set" to "As soon as this option bit is reset" Swapped bit values in Table 7, row 2 Page 27 Modified first paragraph 'to check if FCSR is locked...'. Page 29 Removed watchdog management in All Programming example. Page 28 Added sentence "If the device has only 1 user option byte..." Page 31 Added warning "IAP proposed solutions not valid if LVD is enabled". Page 34 Added missing text in flowchart.
Nov-2003	1.7	Page 15 Added Section 2.2.1.3 Readout Protection Recovery Page 22 Added Caution 'The returned status code... Page 23 Modified note 1 Page 26 Added paragraph 'It is mandatory to erase option bytes.. ' Page 31 Changed "including option byes" to "except option bytes" in Section 3.2.5.7 Page 33 Added note "For users of programming tools like STICK/EPB/inDART" Page 34 Changed note one and added note 4 Page 40 Added Table 12 and following.
May-2005	2	Revision number incremented from 1.7 to 2 due to Internal Document Management System change Removed device-specific tables to move into the "Flash Programming Quick Reference Manual" Removed any references to Debug Module to place into the "Debug Module Reference Manual" Page 8 Rephrased description of Read-out Protection (also Page 15, Page 32) Page 33 Added Section 3.2.6 to 3.2.11 on FREQ parameter (replacing note 4 on page 37) Page 34 Added timeout infomation and other clarifications in Section 3.2.7 thru Section 3.2.11.

"THE PRESENT NOTE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS WITH INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE CONTENT OF SUCH A NOTE AND/OR THE USE MADE BY CUSTOMERS OF THE INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS."