PM0054
Programming manual

How to program STM8L Flash program memory and data EEPROM

Introduction

This manual describes how to program Flash program memory and data EEPROM on STM8 microcontrollers. It applies to low density STM8L15x/STM8L101x, medium and medium + density STM8L15x, and high density STM8L15x/16xdevices. It is intended to provide information to the programming tool manufacturers and to the customers who want to implement programming by themselves on their production line.

The in-circuit programming (ICP) method is used to update the content of Flash program memory and data EEPROM while the user software is not running. It uses the Single wire interface module (SWIM) to communicate between the programming tool and the device.

In contrast to the ICP method, in-application programming (IAP) can use any communication interface supported by the microcontroller (I/Os, SPI, USART, I^{2}C, USB, CAN...). IAP has been implemented for users who want their application software to update itself by re-programming the Flash program memory during program execution. The main advantage of IAP is its ability to re-program Flash program memory and data EEPROM when the chip has already been soldered on the application board and while the user software is running. Nevertheless, part of the Flash program memory has to be previously programmed using ICP.

Some devices also contain a bootloader embedded in a ROM memory. Through this firmware the device memory can be re-programmed using a standard communication interface. This programming method is not described in this document.

For details on memory implementation and features, registers or stack top addresses, refer to the product datasheets.

Related documents

- STM8 SWIM communication protocol and debug module (UM0470)
- STM8 bootloader user manual (UM0560)l
- Low density STM8L101x microcontroller family reference manual (RM0013)
- Low density, medium density, medium + density and high density STM8L15x/16x microcontroller family reference manual (RM0031)
- Basic in-application programming example using the STM8 I^{2}C and SPI peripherals (AN2737)
- Low density STM8L101x datasheet
- Medium density STM8L15xxx datasheet
- Medium + and high density STM8L15x8/R6 datasheet
- High density STM8L162x8 datasheet

Contents

List of tables

1 Glossary

This section gives a brief definition of acronyms and terms used in this document:

● **Block**

A block is a set of bytes that can be programmed or erased in one single programming operation. Operations that are available on a block are fast programming, erase only, and standard programming (which includes an erase operation). Refer to *Section 2: Memory organization* for details on block size according to the device.

● **Bootloader**

The bootloader is an IAP application embedded in the system memory of the device. It is used to erase and program the device using a standard serial communication port. The bootloader is not available on small devices, and is not described in the present document. Refer to STM8 bootloader user manual (UM0500) for more details.

● **Driver**

A driver is a control program defined by the application developer. It is used to manage the allocation of system resources to start application programs. In this document two drivers are described, ICP and IAP drivers.

● **In-application programming (IAP)**

IAP is the ability to re-program the Flash program memory and data EEPROM (DATA) of a microcontroller while the device is already plugged-in to the application and the application is running.

● **In-circuit programming (ICP)**

ICP is the ability to program the Flash program memory and data EEPROM of a microcontroller using the SWIM protocol while the device is plugged-in to the application.

● **In-circuit debugging (ICD)**

ICD is the ability to debug the user software using the SWIM protocol. The user has the ability to connect the device to a debugger and insert breakpoints in his firmware. Debugging may be intrusive (application patched to allow debugging) or non intrusive (using a debug module).

● **Memory access security system (MASS) keys**

The Memory access security system (MASS) consists of a memory write protection lock designed to prevent unwanted memory modifications due to EMS or program counter loss. To unlock the memory protection, one or more keys must be written in a dedicated register and in a specific order. When the operation (write or erase) is completed, the MASS must be activated again to provide good memory security.

● **Page**

A page is a set of blocks. The number of blocks in a page may differ from one device to another. Refer to *Section 2: Memory organization* for details on page size according to the device.

Dedicated option bytes can be used to configure by increments of one page the size of the user boot code, the proprietary code, and the data EEPROM. Refer to *Section 5: Flash program memory and data EEPROM comparison* for a description of available areas and option bytes according to the devices.

● **Proprietary code area (PCODE)**

The proprietary code area (PCODE) can be used to protect proprietary software libraries used to drive peripherals. Refer to *Section 3.2: Proprietary code area protection* for details.

● **Read-while-write (RWW)**

The RWW feature provides the ability for the software to perform write operation on data EEPROM while reading and executing the program memory. Execution time is therefore optimized. The opposite operation is not allowed: the software cannot read data memory while writing program memory.

The RWW feature is not available on all STM8 devices. Refer to *Table 10* for information on devices with RWW capability.

● **Single wire interface module (SWIM)**

The SWIM is a communication protocol managed by hardware in the STM8 microcontrollers. The SWIM main purpose is to provide non intrusive debug capability. It can also be 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 SWIM protocol is used for ICP. It is accessed by providing a specific sequence on the SWIM pin either during the reset phase or when the device is running (if allowed by the application).

● **System memory**

The STM8 system memory is a small ROM accessible when the user software is executed. It contains the bootloader. The system ROM and the bootloader are not available on all STM8 devices.

● **User boot code area (UBC)**

The user boot code area is a write-protected area which contains reset vector, interrupt vectors, and IAP routine for the device to be able to recover from interrupted or erroneous IAP programming.

● **User mode**

The user mode is the standard user software running mode in the STM8. It is entered either by performing a power-on-reset on the device or by issuing the SWIM SRST command from a development tool.

● **Word**

A word is a set of 4 bytes and corresponds to the memory granularity.

2   Memory organization

This section describes the memory organization corresponding to:

● **low density** STM8L101x

● **low density** STM8L151C3/K3/K2/G3/G2/F3/F2F

● **medium density** STM8L151C6/C4/K6/K4/G6/G4 and STM8L152C6/C4/K6/K4

● **medium+ density** STM8L151R6 and STM8L152R6

● **high density** STM8L151x8, STM8L152x8, STM8L162x8

Low density STM8L101x microcontrollers feature up to 8 Kbytes of Flash program memory including up to 2 Kbytes of data EEPROM.

Low density STM8L15x microcontrollers feature from 4 to 8 Kbytes of Flash memory, plus 256 bytes of data EEPROM.

Medium and medium+ density STM8L15x microcontrollers feature from 16 to 32 Kbytes of Flash memory, plus 1 Kbyte of data EEPROM.

High density STM8L15x/16x microcontrollers feature 64 Kbytes of Flash memory and 2 Kbytes of data EEPROM.

The memory organization differs from one STM8 family to another. Refer to *Section 2.1*, *Section 2.2* and *Section 2.5*,for a description of the memory organization according to microcontroller family and memory density.

A memory accelerator takes advantage of the parallel 4-byte storage, which corresponds to a word. The Flash program memory and data EEPROM can be erased and programmed at byte level, word level or block level. In word programming mode, 4 bytes are programmed/erased during the same cycle, while in block programming mode, a whole block is programmed/erase during the same cycle. Refer to *Section 2.1*, *Section 2.2* and *Section 2.5*, for information on block size according to the devices.

2.1   Low density STM8L101x microcontrollers

The low density STM8L101x memory features:

● Up to 8 Kbytes of Flash program memory including up to 2 Kbytes of data EEPROM. The whole memory array is divided into 128 pages of one block (64 bytes) each.

  The Flash program memory is divided into 3 areas:

  –   The user boot code area (UBC)

  –   A configurable data EEPROM area (DATA)

  –   The main program area

  The DATA and main program areas can be write protected independently by using the memory access security mechanism (MASS).

  The size of UBC, and DATA areas can be configured through option bytes.

● One block (64 bytes) of option bytes of which 5 bytes are already used to configure device hardware features. The option bytes can be programmed only in ICP/SWIM mode.

Refer to *Table 1: Low density STM8L101x memory partition* for a detailed description of low density STM8L101x memory partition.

Table 1.   Low density STM8L101x memory partition[1]

Area			Page/block number (1 page = 1 block)	Address
Option Byte			0	0x00 4800-0x00 483F
Flash program memory	UBC and main program memory		0	0x00 8000-0x00 803F
			1	0x00 8040-0x00 807F
			2	0x00 8080-0x00 80BF
			3	0x00 80C0-0x00 80FF
			…	…
			95	…
	Configurable data EEPROM[2]		96	0x00 9800-0x00 987F
			-	…
				0x00 9F80-0x00 9FBF
			Up to 127	0x00 9FC0-0x00 9FFF

1. The memory mapping is given for the devices featuring 8 Kbytes of Flash program memory including up to 2 Kbytes of data EEPROM.

2. The size of the data EEPROM area is configurable from 0 to 32 pages starting from the last page of the Flash program memory down to page 96.

2.2   Low density STM8L15x microcontrollers

The low density STM8L15x memory features:

● From 4 to 8 Kbytes of embedded Flash program divided into 128 pages of 64 bytes each. The memory array is organized in 32-bit words (4 bytes). It includes two areas:

The Flash program memory is divided into 2 areas:

– The user boot code area (UBC)

– The main program area

The size of the UBC area can be configured through option bytes.

● 256 bytes of data EEPROM located in a distinct array from Flash program memory

● Up to 64 option bytes (one block) of which 5 bytes are already used to configure device hardware features. The option bytes can be programmed in user, IAP and ICP/SWIM modes except for ROP and UBC option bytes (refer to *Section 4.5.2*).

Refer to *Table 2* for a detailed description of the memory partition for medium density STM8L15x microcontrollers.

Table 2.   Low density STM8L15x memory partition[1]

Area	Page number (1 page=1 block)	Address
Data EEPROM	0	0x00 1000-0x00 103F
	1	0x00 1040-0x00 107F
	2	0x00 1080-0x00 10BF
	3	0x00 10C0- 0x00 10FF

Table 2. **Low density STM8L15x memory partition[(1)] (continued)**

Area			Page number (1 page=1 block)	Address
Option bytes			0	0x00 4800-0x00 483F
Flash program memory	Main program memory	UBC , PCODE	0	0x00 8000-0x00 003F
			1	0x00 8040-0x00 807F
			2	0x00 8080-0x00 80BF
			3	0x00 80C0-0x00 80FF
			4	0x00 8100-0x00 813F
			5	0x00 8140-0x00 817F
			...	...
		UBC	127	0x00 9FC0-0x00 9FFF

1. The memory mapping is given for the devices featuring 8 Kbytes of Flash program memory and 1 Kbyte of data EEPROM.

2.3 Medium density STM8L15x microcontrollers

The medium density STM8L15x memory features:

● From16 to 32 Kbytes of embedded Flash program divided into 256 pages of one block (128 bytes) each. The memory array is organized in 32-bit words (4bytes). It includes two areas:

The Flash program memory is divided into 2 areas:

  – The user boot code area (UBC)

  – The main program area

The size of the UBC area can be configured through option bytes.

● 1 Kbyte of data EEPROM located in a distinct array from Flash program memory

● Up to 128 option bytes (one block) of which 5 bytes are already used to configure device hardware features. The option bytes can be programmed in user, IAP and ICP/SWIM modes except for ROP and UBC option bytes (refer to *Section 4.5.2*).

Refer to *Table 3* for a detailed description of the memory partition for medium density STM8L15x microcontrollers.

Table 3. **Medium density STM8L15x memory partition[(1)]**

Area	Page number (1 page=1 block)	Address
Data EEPROM	0	0x00 1000-0x00 107F
	1	0x00 1080-0x00 10FF
	2	0x00 1100-0x00 117F
	..	0x00 1180-0x00 11FF
	7	0x00 1380-0x00 13FF
Option bytes	0	0x00 4800-0x00 487F

Table 3. **Medium density STM8L15x memory partition[1] (continued)**

Area			Page number (1 page=1 block)	Address
Flash program memory	Main program memory	UBC	0	0x00 8000-0x00 807F
			1	0x00 8080-0x00 80FF
			2	0x00 8100-0x00 817F
			3	0x00 8180-0x00 81FF
			4	0x00 8200-0x00 827F
			5	0x00 8280-0x00 82FF
			...	...
			255	0x00 FF80-0x00 FFFF

1. The memory mapping is given for the devices featuring 32 Kbytes of Flash program memory and 1 Kbyte of data EEPROM.

2.4 Medium+ density STM8L15x microcontrollers

The medium+ density STM8L15x memory features:

● 32 Kbytes of embedded Flash program divided into 128 pages of 256 bytes each. The memory array is organized in 32-bit words (4 bytes). It includes three areas.

The Flash program memory is divided into 3 areas:

– The proprietary code area (PCODE)
– The user boot code area (UBC)
– The main program area

The size of the PCODE and UBC areas can be configured through option bytes.

● 1 Kbyte of data EEPROM located in a distinct array from Flash program memory

● Up to 128 option bytes (one block) of which 5 bytes are already used to configure device hardware features. The option bytes can be programmed in user, IAP and ICP/SWIM modes except for ROP and UBC option bytes (refer to *Section 4.5.2*).

Refer to the following table for a detailed description of the memory partition for medium+ density STM8L15x microcontrollers.

Table 4. **Medium+ density STM8L15x/16x memory partition**

Area	Page number (1 page = 2 blocks)	Address
Data EEPROM	0	0x00 1000-0x00 10FF
	1	0x00 1100-0x00 11FF
	2	0x00 1200-0x00 12FF
	..	...
	7	0x00 1700-0x00 17FF
Option bytes	0	0x00 4800-0x00 487F

Table 4.   Medium+ density STM8L15x/16x memory partition (continued)

Area			Page number (1 page = 2 blocks)	Address
Flash program memory	Main program memory	PCODE, UBC	0	0x00 8000-0x00 80FF
			1	0x00 8100-0x00 81FF
			2	0x00 8200-0x00 82FF
			3	0x00 8300-0x00 83FF
			4	0x00 8400-0x00 84FF
			...	...
			127	0x00 FF00-0x00 FFFF

2.5   High density STM8L15x/16x microcontrollers

The high density STM8L15x/16x memory features:

● 64 Kbytes of embedded Flash program divided into 256 pages of two blocks (2x128 bytes) each. The memory array is organized in 32-bit words (4 bytes). It includes three areas:

The Flash program memory is divided into 3 areas:

– The proprietary code area (PCODE)

– The user boot code area (UBC)

– The main program area

The size of PCODE and UBC areas can be configured through option bytes.

● Up to 2 Kbytes of data EEPROM located in a distinct array from Flash program memory

● Up to 128 option bytes (one block) of which 5 bytes are already used to configure device hardware features. The option bytes can be programmed in user, IAP and ICP/SWIM modes except for ROP, UBC and PCODESIZE option bytes (refer to *Section 4.5.2*).

Refer to *Table 5* for a detailed description of the memory partition for high density STM8L15x/16x microcontrollers.

Table 5.   High density STM8L15x/16x memory partition

Area	Page number (1 page = 2 blocks)	Address
Data EEPROM	0	0x00 1000-0x00 10FF
	1	0x00 1100-0x00 11FF
	2	0x00 1200-0x00 12FF
	..	...
	7	0x00 1700-0x00 17FF
Option bytes	0	0x00 4800-0x00 487F

Table 5.   High density STM8L15x/16x memory partition (continued)

Area			Page number (1 page = 2 blocks)	Address
Flash program memory	Main program memory	PCODE, UBC	0	0x00 8000-0x00 80FF
			1	0x00 8100-0x00 81FF
			2	0x00 8200-0x00 82FF
			3	0x00 8300-0x00 83FF
			4	0x00 8400-0x00 84FF
			...	...
			255	0x01 7F00-0x01 7FFF

3 Memory protection strategy

The STM8 devices feature several mechanisms allowing to protect the content of the Flash program and data EEPROM areas:

● **Readout protection**

The software can prevent application code and data stored in the Flash program memory and data EEPROM from being read and modified in ICP/SWIM mode. The readout protection is enabled and disabled by programming an option byte in ICP/SWIM mode. Refer to *Section 3.1* for details.

● **Proprietary code area (PCODE)**

To protect proprietary peripheral software driver libraries, some STM8 devices features a permanently readout protected area, the proprietary code area (PCODE). This area is part of the Flash program memory. Its content cannot be modified and can only be read/executed in user privileged mode.

The size of the PCODE area can be configured in ICP/SWIM mode through an option byte by increments of one page.

The PCODE area is not available on all devices. Refer to *Table 10: Comparison between STM8L devices* for the devices featuring PCODE area, and to *Section 3.2* for details on PCODE area.

● User boot code area (UBC)

In order to guaranty the capability to recover from an interrupted or erroneous IAP programming, all STM8 devices provide a write-protected area called user boot code (UBC). This area is a part of the Flash program memory which cannot be modified in user mode (that is protected against modification by the user software). The content of the UBC area can be modified only in ICP/SWIM mode after clearing the UBC option byte.

The size of the user boot code area can be configured through an option byte by increments of one page.

Refer to *Section 3.3* for details on user boot code area.

● **Unwanted memory access protection**

All STM8 devices offer unwanted memory access protection, which purpose is to prevent unintentional modification of program memory and data EEPROM (for example due to a firmware bug or EMC disturbance).

This protection consists of authorizing write access to the memory only through a specific software sequence which is unlikely to happen randomly or by mistake. Access to Flash program and data EEPROM areas is enabled by writing MASS keys into key registers.

Refer to *Section 3.4* for details on unwanted memory access protection.

3.1 Readout protection

On low density STM8L101x microcontrollers, the readout protection is enabled by writing 0xAA in the ROP option byte. It is disabled by reprogramming the ROP option byte with any value except for 0xAA and resetting the device.

On low density STM8L15x, medium density STM8L15x and high density STM8L15x/16x microcontrollers, the readout protection is set by writing any value except for 0xAA in the

ROP option byte. It is disabled by reprogramming the ROP option byte with 0xAA, and resetting the device. To unprotect the device, the ROP must be written 2 times: the first writing, with any value, launches a global erase which includes the option byte; the second writing, 0xAA, unprotects the device, and then the 0xAA value remains in the ROP byte. Please note the EOP should be checked each time.

The readout protection can only be disabled in ICP/SWIM mode.

When the readout protection is selected, reading or modifying the Flash program memory in ICP mode (using the SWIM interface) is forbidden. When available, the data EEPROM memory is also protected against read and write access through ICP.

Erasing the ROP option byte to disable the readout protection causes the Flash program memory, the DATA area and the option bytes to be erased.

Even though no protection can be considered as totally unbreakable, the readout protection feature provides a very high level of protection for general purpose microcontrollers. Of course, a software that allows the user to dump the Flash program memory content make this readout protection useless. *Table 9* describes possible accesses to each memory areas versus the different modes and readout protection settings.

3.2 Proprietary code area protection

The memory pages containing peripherals software libraries must be located in the proprietary code area (PCODE).

The size of the PCODE area can be configured through the PCODE option byte (PCODESIZE) in ICP/SWIM mode. Refer to *Table 6* for details on PCODE minimum and maximum size. Once programmed, the PCODE option byte cannot be erased, the size of the PCODE area remains fixed and its content protected from read and write operations whatever the mode.

Table 6.   PCODE size

STM8 microcontroller family	Minimum size	Maximum size
Low density STM8L15x	3 pages ((192 bytes)	127 pages (last page always left free for main program)
Medium+ density STM8L15x	1 page (256 bytes)	128 pages
High density STM8L15x/16x	1 page (256 bytes)	255 pages (last page always left free for main program)

Except for the interrupt vectors which can be read directly, the PCODE area can be read only through TRAP or TLI interrupt in ICP/SWIM (with readout protection disabled), user and IAP mode.

When the PCODE option byte is set, the PCODE area and both TLI and TRAP vectors are write protected to prevent a malicious user/program from inserting a "dump" routine inside the protected code.

When accessing the PCODE area, no other program can be executed (all interrupt and debug access are disabled). The programmer must consequently ensure that the protected

code embedded in the PCODE area gives back control to the main/end user software at reasonable periods of time.

3.3 User Boot Code area protection

Whatever the memory content, it is always possible to restart an ICP session after a critical error by applying a reset and restarting the SWIM communication.

On the contrary, during IAP sessions, the programming software driver must always be write protected to be able to recover from any critical failure that might happen during programming (such as power failure).

The pages where the IAP driver is implemented must be located in the write-protected boot code area (UBC). The application reset and interrupt vectors and the reset routine must also be stored in the UBC. These conditions allow the user software to manage the recovery from potential critical failure by applying a reset and restarting the IAP routine from the protected boot area.

The UBC size is defined by the user boot code (UBC) area option byte. This option byte may slightly differ from one product to another. See the following table for the minimum and maximum size of the UBC area.

Table 7. **Recommended minimum and maximum sizes of the UBC area**

STM8 microcontroller family	Recommended minimum size of the UBC area	Maximum size of the UBC area
Low density STM8L101x	3 pages = 192 bytes	127 pages
Low density STM8L15x	3 pages = 192 bytes	128 pages
Medium density STM8L15x	2 pages = 256 bytes	255 pages
Medium+ density STM8L15x and	1 page = 256 bytes	128 pages
High density STM8L16x	1 page = 256 bytes	255 pages

3.4 Unwanted memory access protection

The unwanted memory access protection consists of writing two 8-bit keys in the right order into dedicated MASS key registers.

Writing the correct sequence of keys in the program memory MASS key register (FLASH_PUKR) enables the programming of the program memory area excluding the UBC and the PCODE area (when available). If wrong keys are provided, a reset must to be generated to be able to reprogram the right keys.

Once the write memory protection has been removed, it is possible to reactivate the protection of the area by resetting the PUL bit in FLASH_IAPSR.

To enable write access to the data EEPROM area, another specific MASS key register (FLASH_DUKR) and a different key sequence must be used. Once the data EEPROM/option byte area is unlocked, it is possible to reactivate the protection of the area by resetting the DUL bit in FLASH_IAPSR.

If wrong keys have been provided to the FLASH_PUKR register, the device must be reset before performing a new key program sequence. However, when wrong keys are provided to

the FLASH_DUKR register, new keys can be entered without the device being previously reset.

On low density STM8L devices, the size of the DATA area can be configured through the DATASIZE option byte for a given product.

In order to be as effective as possible, the application software must lock again the unwanted memory access protection as soon as the programming is completed. Otherwise, the protection level of the MASS is significantly reduced. To activate the MASS protection again, the user must reset the corresponding bits in the FLASH_IAPSR register (DUL bit for data EEPROM or PUL bit for Flash program memory).

Note: *1   The mechanism to lock and unlock unwanted memory access protection is identical for option bytes and data EEPROM (see Table 8: MASS).*

            2   Before starting programming program memory or data EEPROM, the software must verify that the area is not write protected by checking that the PUL or DUL bit is effectively set.

4 Programming STM8 Flash microcontrollers

This section describes how to program STM8 single-voltage Flash microcontrollers.

4.1 Unlocking the Memory Access Security System (MASS)

The memory must be unlocked before attempting to perform any erase or write operation. To unlock it, follow the procedure described in *Section 3.4: Unwanted memory access protection*, and *Table 8*.

The software must poll the PUL and DUL bit, before attempting to write to program memory and data EEPROM, respectively.

Table 8.   MASS

Microcontroller family	Data EEPROM and option bytes		Program memory	
	Unlock	**Lock**	**Unlock**	**Lock**
STM8L	Write 0xAE then 56h in FLASH_DUKR (0x00 5053)[1][2]	Reset bit 3 (DUL) in FLASH_IAPSR (0x00 5054)	Write 0x56 then 0xAE in FLASH_PUKR (0x00 5052)[3]	Reset bit 1 (PUL) in FLASH_IAPSR (0x00 5054)

1. In the low density STM8L101x devices the option bytes are not accessible in user/IAP mode.

2. The OPT bit of the FLASH_CR2 register must be set/cleared to enable access to the option bytes.

3. If wrong keys have been entered, a reset must to be generated to be able to reprogram the right keys.

4.2 Block programming

Block write operations allow to program an entire block in one shot, thus minimizing the programming time.

There are three possible block programming modes: erase, write only (also called fast programming) and combined erase/write cycle (also called standard block programming). The programming mode is selected through FLASH_CR2 register.

The memory must be unlocked before performing any of these operations.

Block program operations can be performed both to main program memory and DATA area:.

● Programming a block of main program memory:

The block program operation has to be executed totally from RAM.

The program execution continues from RAM. If the program goes back to main program memory, it is stalled until the block program operation is complete. On medium and medium+ density STM8L15x and high density STM8L15x/16x, the DMA controller can

be programmed to perform a block transfer to Flash program memory, and put the CPU in Wait mode.

● Programming a block of data EEPROM with RWW capability:

The block program operation can be executed from main program memory. However the data loading phase has to be executed from RAM (see below).

Normal program execution can continue from main program memory. The HVOFF bit of the FLASH_IAPSR register can be polled to check if the memory is ready for RWW.

● Programming a block of data EEPROM without RWW capability:

The block program operation must be executed totally from RAM.

The programming can also be performed directly through the SWIM interface. In this case, it is recommended to stall the device in order to prevent the core from accessing the Flash program memory during the block program or erase operation. This can be done by setting the STALL bit in the DM_CSR2 debug module register. Refer to the STM8 SWIM communication protocol and debug module (UM0470) for more information.

Caution: During a block program or erase operation, it is recommended to avoid executing instructions performing a read access to program memory.

Caution: If the number of written memory locations is higher than what is required in the block program/erase sequence, the additional locations are handled as redundant byte write operations.

If the number of written memory locations is lower than what is specified in the block program/erase sequence, the block program/erase process does not start and the CPU stalls waiting for the remaining operations to be performed.

Caution: EOP and WR_PG_DIS bits of FLASH_IAPSR register are automatically cleared when a program/erase operation starts.

Caution: If a block program or erase sequence is interrupted by a reset, the data programmed in the memory may be corrupted.

Standard block programming

The following sequence is required to perform a standard block program sequence (block erased and programmed):

1. Unlock the memory if not already done.

    The UBC option byte can be read to check if the block to program is not in the UBC area. If necessary, reprogram it to allow erasing and programming the targeted block.

2. Write 0x01 in FLASH_CR2 (PRG bit active).

3. Write all the data bytes of the block you want to program starting with the very first address of the block. No read or write access to the program memory is allowed during these load operations as they might corrupt the values to be programmed.

    The programming cycle starts automatically when all the data in the block have been written.

4. Check the WR_PG_DIS bit in FLASH_IAPSR to verify if the block you attempted to program was not write protected (optional)

5. To check if the program operation is complete, poll the EOP bit in FLASH_IAPSR from program memory. EOP is set to '1' when the standard block program operation has completed. To avoid polling the EOP bit, an interrupt can be generated when EOP is set.

Note: It is mandatory to execute steps 2 to 4 from RAM.

Caution: EOP and WR_PG_DIS bits are cleared by reading the FLASH_IAPSR register. It is consequently strongly recommended to perform one single read operation to the FLASH_IAPSR register to check the values of these bits.

Fast block programming operation

The following sequence is required to perform a fast block program sequence (block programmed without previous erase):

1. Unlock the memory if not already done.

   The UBC option byte can be read to check if the block to program is not in the UBC area. If necessary, reprogram it to allow programming the targeted block.

2. Write 0x10 in FLASH_CR2 (FPRG bit active).

3. Write all the data bytes of the block you want to program starting with the very first address of the block. No read or write access to the program memory is allowed during these load operations as they might corrupt the values to be programmed.

   The programming cycle starts automatically when the complete block has been written.

4. Check the WR_PG_DIS bit in FLASH_IAPSR to verify if the block you attempted to program was not write protected (optional).

5. To check if the program operation is complete, poll the EOP bit in FLASH_IAPSR from program memory. EOP is set to '1' when the block program operation has completed. To avoid polling the EOP bit, an interrupt can be generated when EOP is set.

Note: It is mandatory to execute steps 2 to 4 from RAM.

Caution: EOP and WR_PG_DIS bits are cleared by reading the FLASH_IAPSR register. It is consequently strongly recommended to perform one single read operation to the FLASH_IAPSR register to check the values of these bits.

Caution: The memory block must be empty when performing a fast block programming operation.

Block erase operation

The following sequence is required to perform a block erase sequence:

1. Unlock the memory if not already done.

   The UBC option byte can be read to check if the block to erase is not in the UBC area. If necessary, reprogram it to allow erasing the targeted block.

2. Write 0x20 in FLASH_CR2 (ERASE bit active).

3. Write '0x00 00 00 00' to any word inside the block to be erased using a LOAD instruction.

4. Check the WR_PG_DIS bit in FLASH_IAPSR to verify if the block you attempted to erase was not write protected (optional).

5. To check if the erase operation is complete, poll the EOP bit in FLASH_IAPSR from program memory. EOP is set to '1' when the block erase operation has completed. To avoid polling the EOP bit, an interrupt can be generated when EOP is set.

Note: It is mandatory to execute steps 2 to 4 from RAM.

Caution: EOP and WR_PG_DIS bits are cleared by reading the FLASH_IAPSR register. It is consequently strongly recommended to perform one single read operation to the FLASH_IAPSR register to check the values of these bits.

4.3 Word programming

Both main program memory and data EEPROM can be programmed and erased at word level. Word operations are performed in the same way as block operations. They can be executed either from program memory or from RAM.

When a new word program operation starts, EOP and WR_PG_DIS bits of FLASH_IAPSR register are automatically cleared.

Contrary to word programming of the Flash program memory, the word programming of data EEPROM with RWW feature (when available) does not stop program execution. The EOP bit can then be used to know if the previous operation has completed. This bit is automatically reset when reading FLASH_IAPSR.

The following sequence is required to perform a word program operation:

1. Unlock the memory if not already done.

   The UBC option byte can be read to check if the word you want to program is not in the UBC area. If necessary, reprogram it to allow programming the targeted word.

2. Write 0x40 in FLASH_CR2 (WP bit active).

3. Write the 4 data bytes to the memory starting with the very first address of the word to be programmed.

   The programming cycle starts automatically when the 4 bytes have been written.

4. Check the WR_PG_DIS bit in FLASH_IAPSR to verify if the word you attempted to program was not write-protected (optional).

5. To check if the program operation is complete, poll the EOP bit in FLASH_IAPSR register for the end of operations. EOP is set to '1' when the word program operation has completed. To avoid polling the EOP bit, an interrupt can be generated when EOP is set.

Caution: EOP and WR_PG_DIS bits are cleared by reading the FLASH_IAPSR register. It is consequently strongly recommended to perform one single read operation to the FLASH_IAPSR register to check the values of these bits.

Caution: If a word program operation is interrupted by a reset, the data programmed in the memory may be corrupted.

4.4 Byte programming

Both main program memory and data EEPROM can be programmed and erased at byte level.

Byte programming is performed by executing a write instruction (ld, mov...) to an address in main program memory when the memory is unlocked. The write instruction initiates the erase/program cycle and any core access to the memory is blocked until the cycle has completed. This means that program execution from the Flash program memory is stopped until the end of the erase/program cycle.

When a new byte program operation starts, EOP and WR_PG_DIS bits of FLASH_IAPSR register are automatically cleared. At the end of the program operation, the EOP bit in the FLASH_IAPSR register is set and the program execution restarts from the instruction following the write/erase instruction.

Contrary to byte programming of the Flash program memory, the byte programming of data EEPROM with RWW feature (when available) does not stop program execution. The EOP

bit can then be used in order to know if the previous operation has completed. To avoid polling the EOP bit, an interrupt can be generated when EOP is set. This bit is automatically reset when reading FLASH_IAPSR.

The erase/program cycle lasts longer if the whole word containing the byte to be programmed is not empty because in this case an erase operation is performed automatically. If the word is empty, the erase operation is not performed. However, if a defined programming time is wanted, the FIX bit in the FLASH_CR1 register forces the programming operation to always erase first whatever the contents of the memory. Therefore a fixed programming time is guaranteed (erase time + write time).

To erase a byte location, just write '0x00' to the byte location.

Caution: A byte programming operation performs a word (4-byte) access to the Flash program memory. If a byte program operation is interrupted by a reset, the 4 bytes programmed in the memory may be corrupted.

4.5   Programming the option bytes

Option bytes are used to configure the device hardware features as well as the memory protection. They are stored in a dedicated memory block.

4.5.1   Summary of memory dedicated option bytes

The Flash program memory includes several option bytes dedicated to memory protection:

● ROP

   The ROP option byte is used to prevent the Flash program memory from being read and modified in ICP/SWIM mode. Refer to *Section 3.1: Readout protection* for a detailed description of readout protection.

● PCODESIZE (not available on low density STM8L101x and medium density STM8L15x devices)

   The PCODESIZE option byte allows to configure the size of the proprietary code area (PCODE) which can be used to store proprietary software libraries. The minimum size of proprietary code area is of 1 page (256 bytes) and the maximum size of 255 pages. The PCODESIZE option byte can be modified only in ICP/SWIM mode.

   Refer to *Section 3.2: Proprietary code area protection* for a detailed description of the PCODE area, and to *Table 10: Comparison between STM8L devices* for the devices featuring PCODE area and PCODESIZE option byte.

● UBC

   The UBC option byte is used to program the size of the write protected user boot code area. The boot area always includes the reset and interrupt vectors and can go up to the full program memory size. The boot area size granularity is of one page.

   Refer to *Section 3.3: User Boot Code area protection* for a detailed description of the UBC area.

● DATASIZE (available only on low density STM8L101x devices)

   The DATASIZE option byte is used to configure the size of the data EEPROM area.

   This option byte specifies the number of pages starting from the end of the memory. The maximum size of the data EEPROM area is of 2 Kbytes.

   Refer to *Table 10: Comparison between STM8L devices* for the devices featuring DATASIZE option byte.

4.5.2 How to program the option bytes

The option bytes are stored only once.

On low density STM8L101x devices, the option bytes can be modified only in ICP/SWIM mode with OPT bit of the FLASH_CR2 register set to '1' (refer to the description of FLASH_CR2 register in the reference manual).

On medium and medium+ density STM8L15x and high density STM8L15x/16x devices, the option bytes can be modified both in user/IAP mode and ICP mode (using the SWIM), except for the readout protection (ROP), UBC and PCODESIZE (when available) which can only be removed in ICP mode. In ICP mode, the OPT bit in the FLASH_CR2 register must be set to '1' before attempting to modify the option bytes.

4.6 Memory access versus programming method

Table 9 gives a description of possible accesses from the core to memory areas according to the programming method.

Table 9. Memory access versus programming method[1]

Mode	ROP	Memory Area	Low density STM8L101x	Low and medium density STM8L15x / High density STM8L15x/16x
			Access from core	
User, IAP, and Bootloader (if available)	Readout protection enabled	Interrupt vectors except for TRAP and TLI (if available)	R/W[2]/E	
		TRAP and TLI (if available)	R/W[3]/E	
		Proprietary code area (PCODE) (if available)	R/E	
		User boot code area (UBC)	R/E	
		Main program	R/W/E[4]	
		Data EEPROM area (DATA)	R/W[5]	
		Option bytes	R	
	Readout protection disabled	Interrupt vectors except for TRAP and TLI (if available)	R/W[2]/E	
		TRAP and TLI (if available)	R/W[3]/E	
		Proprietary code area (PCODE) (if available)	R/E[6]	
		User boot code area (UBC)	R/E[7]	
		Main program	R/W/E[4]	
		Data EEPROM area (DATA)	R/W[5]	
		Option bytes	R	R/W[8]

Table 9.   Memory access versus programming method[1] (continued)

Mode	ROP	Memory Area	Low density STM8L101x	Low and medium density STM8L15x / High density STM8L15x/16x
			Access from core	
SWIM active (ICP mode)	Readout protection enabled	Interrupt vectors except for TRAP and TLI (if available)	P	
		TRAP and TLI (if available)	P	
		Proprietary code area (PCODE) (if available)	P[6]	
		User boot code area (UBC)	P	
		Main program	P	
		Data EEPROM area (DATA)	P	
		Option bytes	P/W$_{ROP}$[9]	
	Readout protection disabled	Interrupt vectors except for TRAP and TLI (if available)	R/W[3]/E	
		TRAP and TLI (if available)	R/W/E	
		Proprietary code area (PCODE) (if available)	R/E[6]	
		User boot code area (UBC)	R/E[7]	
		Main program	R/W/E[4]	
		Data EEPROM area (DATA)	R/W[5]	
		Option bytes	R/W	

1. R/W/E = Read; Write and Execute;
   R/E = Read and Execute (write operation forbidden);
   R = Read (write and execute operations forbidden);
   P = the area cannot be accessed (read, execute and write operations forbidden);
   P/W$_{ROP}$ = Protected, write forbidden except for ROP option byte.

2. When no UBC area has been defined, the interrupt vectors except for TRAP and TLI, can be modified in user /IAP mode.

3. If a PCODE area has been defined, then TRAP and TLI vectors cannot be modified in user and IAP mode, otherwise TRAP and TLI follow the same rules as other interrupt vectors.

4. The Flash program memory is write protected (locked) until the correct MASS key is written in the FLASH_PUKR. It is possible to lock the memory again by resetting the PUL bit in the FLASH_IAPSR register. If wrong keys are provided, the device must be reset and new keys programmed.

5. The data memory is write protected (locked) until the correct MASS key is written in the FLASH_DUKR. It is possible to lock the memory again by resetting the DUL bit in the IAPSR register. If wrong keys are provided, another key program sequence can be performed without resetting the device.

6. When available, the PCODE area can be read and executed only in privileged mode through TRAP and TLI vectors. The PCODE cannot be directly accessed through the SWIM.

7. To program the UBC area the application must first clear the UBC option byte.

8. The option bytes are write protected (locked) until the correct MASS key is written in the FLASH_DUKR (with OPT set to '1'). It is possible to lock the memory again by resetting the DUL bit in the FLASH_IAPSR register. If wrong keys are provided, another key program sequence can be performed without resetting the device.

9. When ROP is removed, the whole memory is erased, including option bytes.

4.6.1 ICP methods

The in-circuit programming (ICP) method is used to update the content of Flash program memory and data EEPROM.

The programming interface for STM8 devices is the SWIM (Single Wire Interface Module). It is used to communicate with an external programming device connected via a cable.

See STM8 SWIM communication protocol and debug module user manual (UM0470) for more details on the SWIM mode entry and SWIM protocol.

When using the SWIM protocol, two methods can be used:

First method

The first method consists of writing directly into the Flash registers and memory locations through the write memory command of the SWIM protocol. To make sure that the CPU is not accessing the memory during block Flash programming, the core must be stalled by setting the STALL bit in the DM_CSR2 debug module register.

The following sequence is required:

1. Apply a RESET
2. Activate the SWIM by sending the entry sequence on the SWIM pin
3. Activate the SWIM_CSR register by writing 1 to the DM bit in SWIM_CSR
4. Disable interrupts by setting the SAFE_MASK bit in SWIM_CSR
5. Release RESET
6. Verify the DeviceID by reading it using ROTF command
7. Send the SWIM SRST command
8. Unlock the memory by writing the MASS keys
9. Program the Flash program memory using the SWIM WOTF command

Second method

The second method uses the same sequence of operations as the first method except that the ICP driver firmware must be downloaded in RAM before being launched:

1. Apply a RESET
2. Activate the SWIM by sending the entry sequence on the SWIM pin
3. Activate the SWIM_CSR register by writing 1 to the DM bit in SWIM_CSR
4. Disable interrupts by setting the SAFE_MASK bit in SWIM_CSR
5. Release RESET
6. Verify the DeviceID by reading it using ROTF command
7. Send the SWIM SRST command
8. Unlock the memory by writing the MASS keys
9. Download the ICP driver firmware into the device RAM using the SWIM WOTF command
10. Execute the ICP driver:
    a) Modify the CPU registers (new PC, X, Y, CC...) using the WOTF commands
    b) Set the FLASH bit in the DM_CSR2 register
    c) Clear the STALL bit in the DM_CSR2 register

4.6.2 IAP method

Refer to application note AN2737- Basic in-application programming example using the STM8 I^{2}C and SPI peripherals.

5 Flash program memory and data EEPROM comparison

Table 10 shows the comparison between STM8L microcontroller devices.

Table 10. Comparison between STM8L devices

Feature	Low density STM8L101x	Low density STM8L15x	Medium density STM8L15x	Medium+ density STM8L15x	High density STM8L15x/16x
Memory					
Block size	64 bytes	128 bytes	128 bytes	128 bytes	128 bytes
Page size	1 block (64 bytes)	1 block (128 bytes)	1 block (128 bytes)	2 blocks (256 bytes)	2 blocks (256 bytes)
Flash program memory	Up to 8 Kbytes	From 4 to 8 Kbytes	From 16 to 32 Kbytes	32 Kbytes	64 Kbytes
Data EEPROM	Up to 2 Kbytes included in Flash program memory Size configurable by option byte	256 bytes located in separate memory array Fixed size	1 Kbyte located in separate memory array Fixed size	1 Kbyte located in separate memory array Fixed size	2 Kbytes located in separate memory array Fixed size
Proprietary code area (PCODE)	NO	YES	NO	YES	YES
User boot code (UBC)	YES - size configurable by option byte				
Bootloader	NO	YES	YES	YES	YES
Option bytes	Programmable in ICP/SWIM - 3 memory dedicated option bytes (ROP, UBC, DATASIZE)	Programmable in ICP/SWIM, IAP - 2 memory dedicated option bytes (ROP, UBC)	Programmable in ICP/SWIM, IAP - 2 memory dedicated option bytes (ROP, UBC)	Programmable in ICP/SWIM, IAP - 3 memory dedicated option bytes (ROP, UBC, PCODESIZE)	Programmable in ICP/SWIM, IAP - 3 memory dedicated option bytes (ROP, UBC, PCODESIZE)
Programming/erasing features					
Read-while-write (RWW)	NO	NO	YES	YES	YES
Programming methods [1]	Block programming (fast and standard) Word programming / Byte programming Block Erase[2]				
Flash control registers	FLASH_CR2				

1. Block program/erase sequence must be executed from RAM.

2. Any word in the block programmed to 0.

6 Revision history

Table 11. Document revision history

Date	Revision	Changes
07-April-2009	1	Initial release.
10-Sep-2009	2	Medium density STM8L devices added. NMI replaced by TLI in the whole document. Updated low density SMT8L memory description in *Section 2.1: Low density STM8L101x microcontrollers*. Updated *Section 3.1: Readout protection* for medium density devices. Updated *Section 4.2: Block programming*. Updated HVOFF bit polling when programming a block of data EEPROM with RWW capability. Updated *Section 4.3: Word programming* in particular for RWW feature and word programming sequence. Added EOP interrupt *Section 4.4: Byte programming*. Updated memory access table (*Table 10*).
10-Dec-2009	3	Removed proprietary code area for Medium density STM8L devices. Removed TLI interrupt vector. Suppressed distinction between TRAP and other interrupt vectors in *Table 10: Comparison between STM8L devices*.
16-Jun-2010	4	Added high density STM8L15x/16x devices. Updated last page address in *Table 3: Medium density STM8L15x memory partition*. Specified that the last page is reserved for the main program memory in *Table 3: Medium density STM8L15x memory partition* and *Table 5: High density STM8L15x/16x memory partition*. Updated *Section 3.3: User Boot Code area protection* Updated *Note 1* below *Table 8: MASS*. For medium density STM8L15x and high density STM8L15x/16x, specified that ROP, UBC and PCODESIZE can be programmed only in ICP mode/SWIM mode.
14-Sep-2010	5	Corrected readout protection enabling/disabling conditions in *Section 3.1: Readout protection*. Added medium+ density devices information
18-Feb-2011	6	Updated *Table 4: Medium+ density STM8L15x/16x memory partition*. Added information about disabling device protection in *Section 3.1: Readout protection*.
05-Jul-2011	7	Added information on low density STM8L15x devices.

Please Read Carefully: