Spansion® Analog and Microcontroller Products

The following document contains information on Spansion analog and microcontroller products. Although the document is marked with the name “Fujitsu”, the company that originally developed the specification, Spansion will continue to offer these products to new and existing customers.

Continuity of Specifications

There is no change to this document as a result of offering the device as a Spansion product. Any changes that have been made are the result of normal document improvements and are noted in the document revision summary, where supported. Future routine revisions will occur when appropriate, and changes will be noted in a revision summary.

Continuity of Ordering Part Numbers

Spansion continues to support existing part numbers beginning with “MB”. To order these products, please use only the Ordering Part Numbers listed in this document.

For More Information

Please contact your local sales office for additional information about Spansion memory, analog, and microcontroller products and solutions.

Colophon

The products described in this document are designed, developed and manufactured as contemplated for general use, including without limitation, ordinary industrial use, general office use, personal use, and household use, but are not designed, developed and manufactured as contemplated (1) for any use that includes fatal risks or dangers that, unless extremely high safety is secured, could have a serious effect to the public, and could lead directly to death, personal injury, severe physical damage or other loss (i.e., nuclear reaction control in nuclear facility, aircraft flight control, air traffic control, mass transport control, medical life support system, missile launch control in weapon system), or (2) for any use where chance of failure is intolerable (i.e., submersible repeater and artificial satellite). Please note that Spansion will not be liable to you and/or any third party for any claims or damages arising in connection with above-mentioned uses of the products. Any semiconductor devices have an inherent chance of failure. You must protect against injury, damage or loss from such failures by incorporating safety design measures into your facility and equipment such as redundancy, fire protection, and prevention of over-current levels and other abnormal operating conditions. If any products described in this document represent goods or technologies subject to certain restrictions on export under the Foreign Exchange and Foreign Trade Law of Japan, the US Export Administration Regulations or the applicable laws of any other country, the prior authorization by the respective government entity will be required for export of those products.

Trademarks and Notice

The contents of this document are subject to change without notice. This document may contain information on a Spansion product under development by Spansion. Spansion reserves the right to change or discontinue work on any product without notice. The information in this document is provided as is without warranty or guarantee of any kind as to its accuracy, completeness, operability, fitness for particular purpose, merchantability, non-infringement of third-party rights, or any other warranty, express, implied, or statutory. Spansion assumes no liability for any damages of any kind arising out of the use of the information in this document.

Copyright © 2013 Spansion Inc. All rights reserved. Spansion®, the Spansion logo, MirrorBit®, MirrorBit® Eclipse™, ORNAND™ and combinations thereof, are trademarks and registered trademarks of Spansion LLC in the United States and other countries. Other names used are for informational purposes only and may be trademarks of their respective owners.

Fujitsu Semiconductor Europe
Application Note

MCU-AN-300401-E-V12

FM3 FAMILY

32-BIT MICROCONTROLLER

MB9A/BFXXX

FLASH PROGRAMMING

APPLICATION NOTE

Revision History

Date	Issue
2011-02-28	V1.0; MWi; 1st version
2011-12-13	V1.1; MWi; Device type differences added
2012-03-16	V1.2; MWi; Work Flash added

This document contains 28 pages.

 © Fujitsu Semiconductor Europe GmbH

Warranty and Disclaimer

The use of the deliverables (e.g. software, application examples, target boards, evaluation boards, starter kits, schematics, engineering samples of IC's etc.) is subject to the conditions of Fujitsu Semiconductor Europe GmbH ("FSEU") as set out in (i) the terms of the License Agreement and/or the Sale and Purchase Agreement under which agreements the Product has been delivered, (ii) the technical descriptions and (iii) all accompanying written materials.

Please note that the deliverables are intended for and must only be used for reference in an evaluation laboratory environment.

The software deliverables are provided on an as-is basis without charge and are subject to alterations. It is the user's obligation to fully test the software in its environment and to ensure proper functionality, qualification and compliance with component specifications.

Regarding hardware deliverables, FSEU warrants that they will be free from defects in material and workmanship under use and service as specified in the accompanying written materials for a duration of 1 year from the date of receipt by the customer.

Should a hardware deliverable turn out to be defect, FSEU's entire liability and the customer's exclusive remedy shall be, at FSEU's sole discretion, either return of the purchase price and the license fee, or replacement of the hardware deliverable or parts thereof, if the deliverable is returned to FSEU in original packing and without further defects resulting from the customer's use or the transport. However, this warranty is excluded if the defect has resulted from an accident not attributable to FSEU, or abuse or misapplication attributable to the customer or any other third party not relating to FSEU or to unauthorised decompiling and/or reverse engineering and/or disassembling.

FSEU does not warrant that the deliverables do not infringe any third party intellectual property right (IPR). In the event that the deliverables infringe a third party IPR it is the sole responsibility of the customer to obtain necessary licenses to continue the usage of the deliverable.

In the event the software deliverables include the use of open source components, the provisions of the governing open source license agreement shall apply with respect to such software deliverables.

To the maximum extent permitted by applicable law FSEU disclaims all other warranties, whether express or implied, in particular, but not limited to, warranties of merchantability and fitness for a particular purpose for which the deliverables are not designated.

To the maximum extent permitted by applicable law, FSEU's liability is restricted to intention and gross negligence. FSEU is not liable for consequential damages.

Should one of the above stipulations be or become invalid and/or unenforceable, the remaining stipulations shall stay in full effect.

The contents of this document are subject to change without a prior notice, thus contact FSEU about the latest one.

Contents

 © Fujitsu Semiconductor Europe GmbH

1 Introduction

This application note describes how to program the embedded Flash memory while the application is running. It also shows how to adjust the IAR and KEIL compilers for generating RAM code.

 © Fujitsu Semiconductor Europe GmbH

2 Programming Principle

THIS CHAPTER SHOWS THE PRINCIPLE OF FLASH PROGRAMMING

2.1 Programming via RAM code (Main Flash)

Because for Flash programming the Flash interface has to be set to the command sequencer mode, it cannot be used for reading. This means that the CPU is not able to fetch any instruction from it in this mode.

Therefore for any usage of the automatic programming algorithm with its command sequences the code execution must be done outside the Flash memory. This can be done in external memory, but more useful in the Instruction-RAM area of the FM3 starting from address 0x2000.0000.

The user has to take care, that the programming code itself has to be copied from (constant) ROM area to the I-RAM area before executing it (Step1). Normally this can be done by compiler and linker settings in the used project builder IDE, which is explained later.

The following graphic illustrates the mechanism and principle.

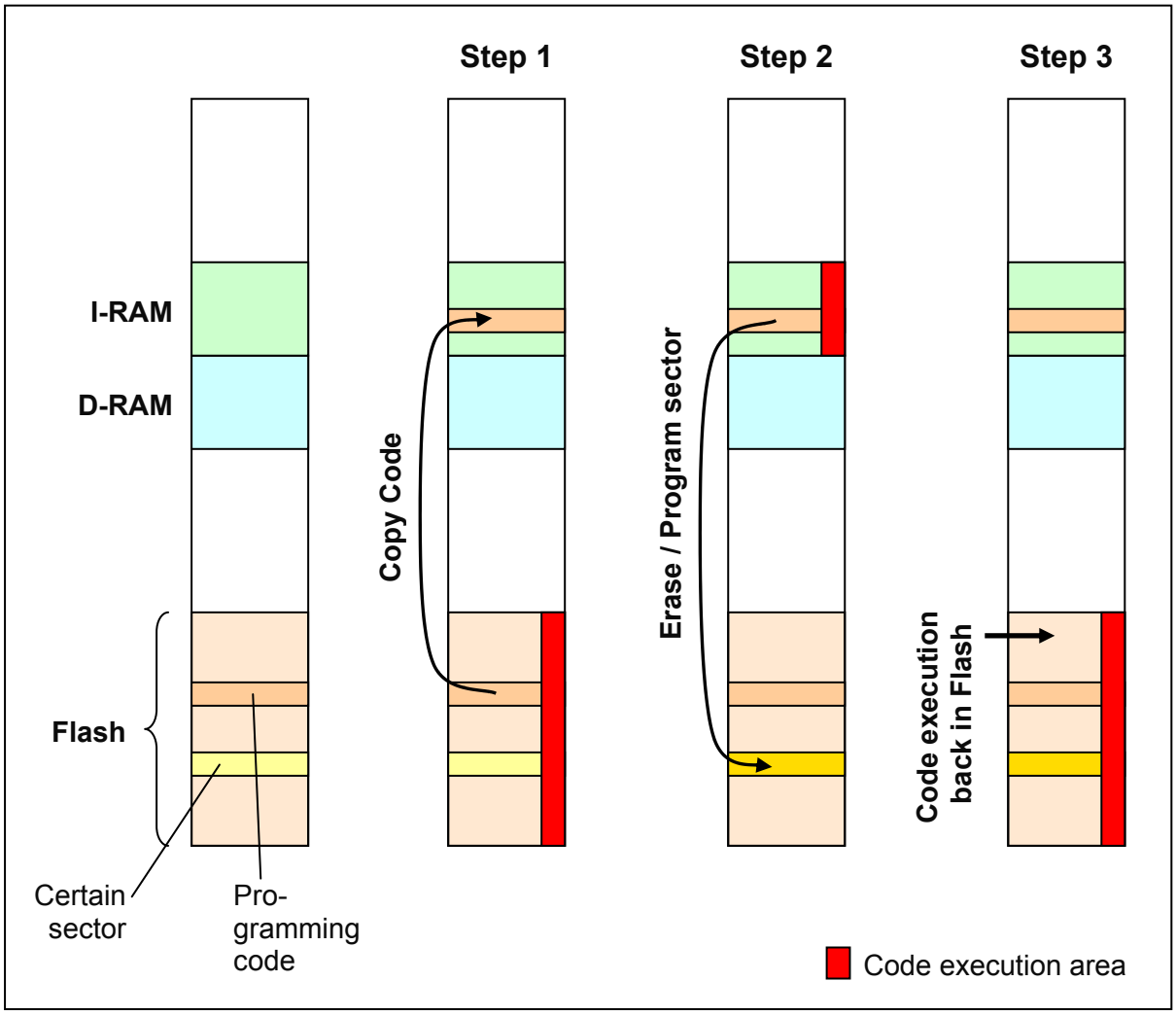

Figure 2-1: (Main) Flash programming principle

After copying the code in Step 1, the application has to jump to the copied RAM code. Here it is allowed to set the Flash memory to erase/programming mode. In the step 2 in illustration above a certain sector is erased and programmed. After successful Flash content change the application can jump back to the Flash area for normal code execution (Step 3).

2.2 Programming the Work Flash

If a device supports Work Flash no RAM code is needed for programming. The Work Flash is a second independent Flash memory, which can be erased and programmed from the Main Flash. This is also possible vice versa.

The following graphic illustrates this:

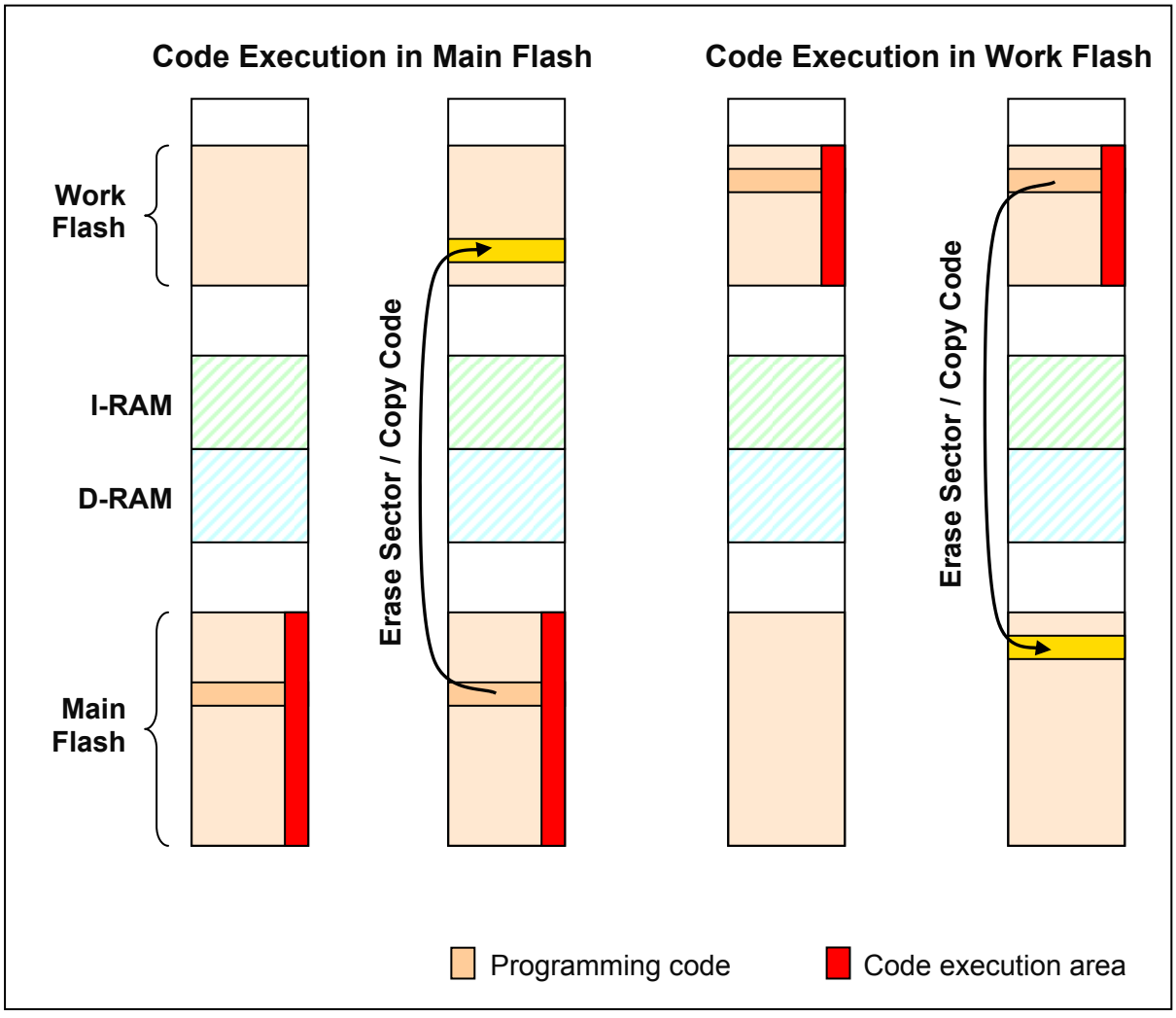

Figure 2-2: Work Flash programming principle

 © Fujitsu Semiconductor Europe GmbH

2.3 Main Flash Memory Organization

Note that the topology of the Flash sectors is organized by lower- and upper-32-bit word sectors, which results in a 64-bit-wide Flash memory. The following graphic shows this topology:

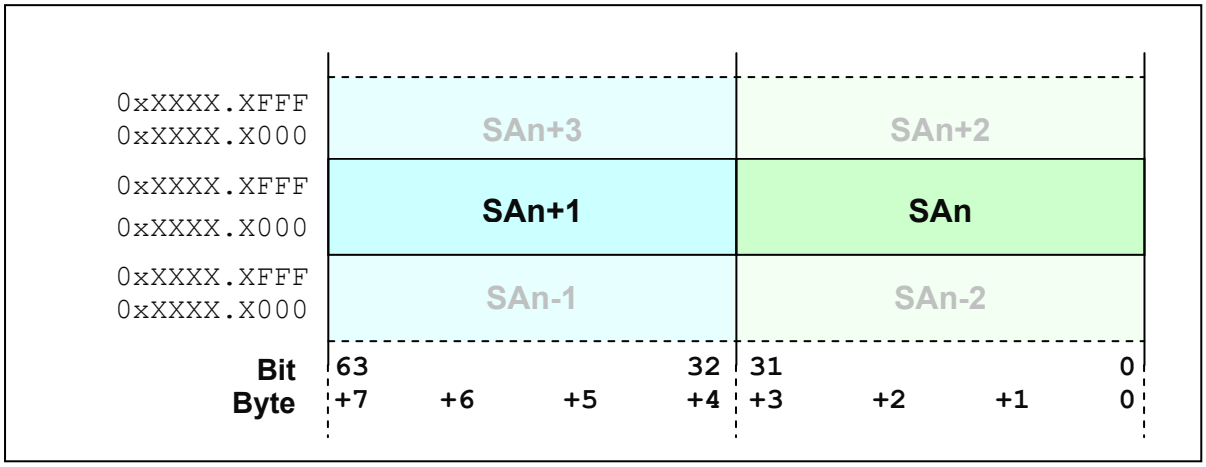

Figure 2-3: Main Flash Sector Topology

This sector topology results in the following sector/memory address organization within two upper- and lower-32-bit-word sectors (SAn and SAn+1):

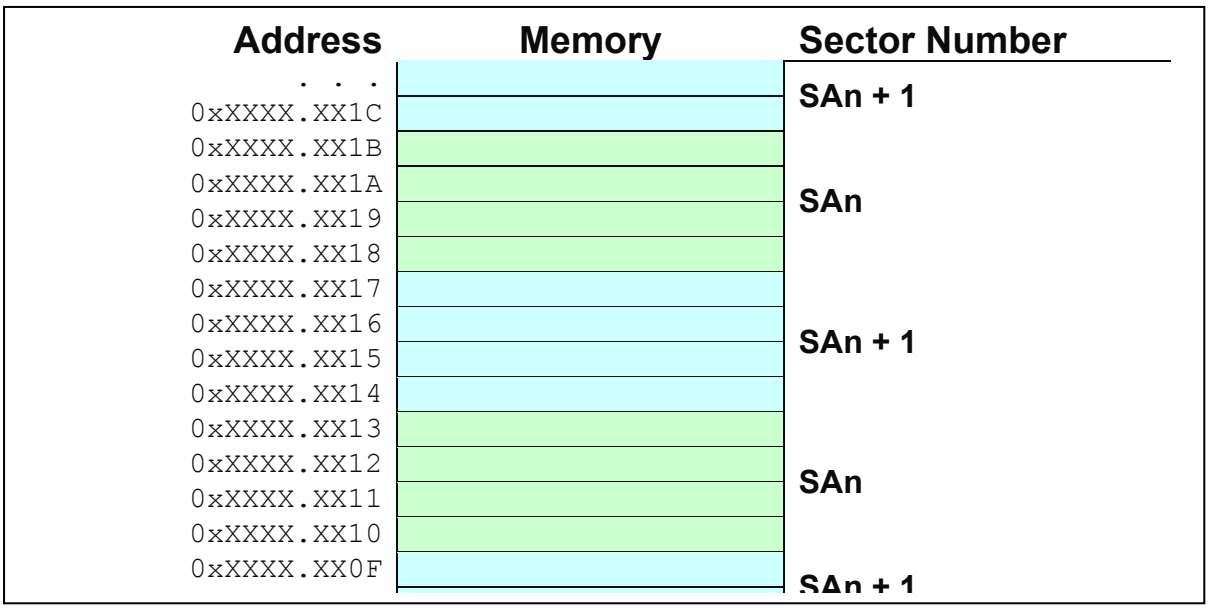

Figure 2-4: Main Flash Memory Organization

2.4 Work Flash Memory Organization

The Work Flash is organized in consecutive 32-Bit Words without interlace:

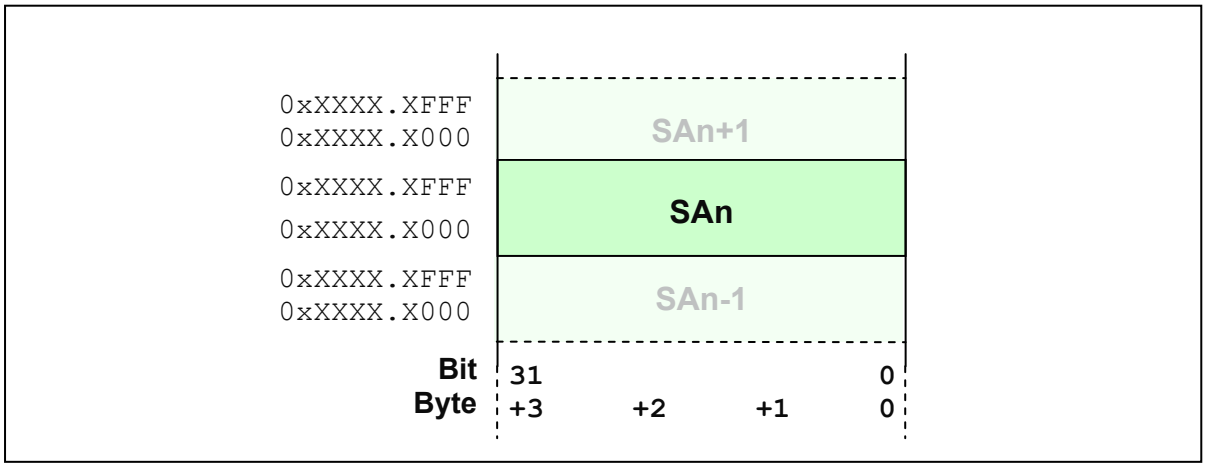

Figure 2-5: Work Flash Sector Topology

 © Fujitsu Semiconductor Europe GmbH

3 Flash Programming Sequences and Registers

THIS CHAPTER SHOWS THE FLASH SEQUENCES AND RELEVANT REGISTERS

3.1 Flash Interface

The FM3 embedded Flash memory provides a Flash interface with an automatic programming algorithm. For unlocking this algorithm certain address/16-bit-data sequences have to be written to the Flash area (where the upper 8 data bits are “don’t care”).

Any write access to the Flash memory area triggers the sequencer, but only certain addresses with certain data unlock a Flash memory command.

3.1.1 Main Flash (Chip) Erase Command Sequence

The Main Flash (chip) erase command sequence consist of the following address/data write accesses:

Address Device Type 0 and 2	Address Device Type 1	Data	Comment
0x0000[1].1550	0x0000[1].0AA8	0xXXAA	1st sequence write
0x0000[1].0AA8	0x0000[1].0554	0xXX55	2nd sequence write
0x0000[1].1550	0x0000[1].0AA8	0xXX80	3rd sequence write
0x0000[1].1550	0x0000[1].0AA8	0xXXAA	4th sequence write
0x0000[1].0AA8	0x0000[1].0554	0xXX55	5th sequence write
0x0000[1].1550	0x0000[1].0AA8	0xXX10	6th sequence write

Table 3-1: Main Flash (Chip) Erase Command Sequence

Note that *this* sequence *only* erases the whole Flash, if the device does *not* support a Work Flash area. If Work Flash is provided, it has to be erased separately (see below).

3.1.2 Main Flash Sector Erase Command Sequence

The Main Flash sector erase command sequence consist of the following address/data write accesses:

Address Device Type 0 and 2	Address Device Type 1	Data	Comment
0x0000[1].1550	0x0000[1].0AA8	0xXXAA	1st sequence write
0x0000[1].0AA8	0x0000[1].0554	0xXX55	2nd sequence write
0x0000[1].1550	0x0000[1].0AA8	0xXX80	3rd sequence write
0x0000[1].1550	0x0000[1].0AA8	0xXXAA	4th sequence write
0x0000[1].0AA8	0x0000[1].0554	0xXX55	5th sequence write
SAn	SAn	0xXX30	6th sequence write with SAn address within sector to be erased

Table 3-2: Main Flash Sector Erase Command Sequence

[1] Note that the sequence addresses for the unlock of the erase command can be any address within the (Main) Flash area. Because the (Main) Flash memory starts from address 0x0000.0000 in the FM3 architecture, the upper 16-bit can be left as 0x0000.

3.1.3 Main Flash 16-/32-Bit Word Write Command Sequence

For writing a 16-bit word to an erased Flash cell, the following command sequence has to be used:

Address Device Type 0 and 2	Address Device Type 1	Data	Comment
0x0000[1].1550	0x0000[1].0AA8	0xXXAA	1st sequence write
0x0000[1].0AA8	0x0000[1].0554	0xXX55	2nd sequence write
0x0000[1].1550	0x0000[1].0AA8	0xXXA0	3rd sequence write
PAddr	PAddr	PData	4th actual write access to PAddr and PData. The address at PAddr will contain PData after successful programming.

Table 3-3: Main Flash Write Data Command Sequence

For writing a 32-Bit word including automatic ECC calculation (Type 2 devices) program the 1st lower 16-Bit word to the desired Flash address and the 2nd upper 16-Bit word the Flash address + 2 afterwards. The second programming step programs also the ECC cells.

[1] Note that the sequence addresses for the unlock of the erase command can be any address within the (Main) Flash area. Because the (Main) Flash memory starts from address 0x0000.0000 in the FM3 architecture, the upper 16-bit can be left as 0x0000.

3.1.4 Work Flash Erase Command Sequence

The Work Flash erase command sequence consist of the following address/data write accesses:

Address	Data	Comment
0x200C[2].0AA8	0xXXAA	1st sequence write
0x200C[2].0554	0xXX55	2nd sequence write
0x200C[2].0AA8	0xXX80	3rd sequence write
0x200C[2].0AA8	0xXXAA	4th sequence write
0x200C[2].0554	0xXX55	5th sequence write
0x200C[2].0AA8	0xXX10	6th sequence write

Table 3-4: Work Flash Erase Command Sequence

[2] Note that the sequence addresses for the unlock of the Work Flash erase command can be any address within the Work Flash area. Assume the Work Flash memory starts from address 0x200C.0000, the upper 16-bit then have to be 0x200C.

 © Fujitsu Semiconductor Europe GmbH

3.1.5 Work Flash Sector Erase Command Sequence

The Work Flash sector erase command sequence consist of the following address/data write accesses:

Address	Data	Comment
0x200C[2].0AA8	0xXXAA	1st sequence write
0x200C[2].0554	0xXX55	2nd sequence write
0x200C[2].0AA8	0xXX80	3rd sequence write
0x200C[2].0AA8	0xXXAA	4th sequence write
0x200C[2].0554	0xXX55	5th sequence write
SAn	0xXX30	6th sequence write with SAn address within sector to be erased

Table 3-5: Work Flash Sector Erase Command Sequence

3.1.6 Main Flash 16-/32-Bit Word Write Command Sequence

For writing a 16-bit word to an erased Flash cell, the following command sequence has to be used:

Address	Data	Comment
0x200C[2].0AA8	0xXXAA	1st sequence write
0x200C[2].0554	0xXX55	2nd sequence write
0x200C[2].0AA8	0xXXA0	3rd sequence write
PAddr	PData	4th actual write access to PAddr and PData. The address at PAddr will contain PData after successful programming.

Table 3-6: Work Flash Write Data Command Sequence

[2] Note that the sequence addresses for the unlock of the Work Flash commands can be any address within the Work Flash area. Assume the Work Flash memory starts from address 0x200C.0000, the upper 16-bit then have to be 0x200C.

3.1.7 Read/Reset

This command can be used to abort any Flash command and reset it to the default read state. Be careful when using it together with chip or sector erase. Incomplete erase states may result.

The read/reset command is issued when writing a **0xXXF0** to any of the Flash memory addresses.

3.1.8 Sector Erase Suspend and Restart

When starting a sector erase by its command sequence, it can be halted by the suspend command. To issue a suspend command, write **0xXXB0** to any of the sector's addresses.

When the sector erase is suspended and in the halt state, it is allowed to write new data to another sector. This is useful, if urgent data have to be written to a non-volatile memory area, but a sector erase was started.

After any write command to another sector, the erase can be resumed by writing **0xXX30** to any on the sector's addresses.

3.1.9 Automatic Programming Algorithm Run States

By reading any Flash address after issuing a command sequence, the lower 8 bits correspond to certain state flags of the Flash interface.

Bit Number	DQ7	DQ6	DQ5	DQ4	DQ3	DQ2	DQ1	DQ0
Name	DPOL	TOGG	TLOV	-	SETI	TOGG2	-	-

Table 3-7: Flags of Automatic Programming Algorithm State

After any command a small user code should check these bits by polling to determine if the command was finished successful or a time-out has occurred.

There a two different ways for this: Data polling and data toggle algorithm.

The most important bits are DPOL, TLOV and SETI, if the data polling algorithm is used. This method is described in this application note. Refer to the Flash programming manual for details of the data toggle algorithm.

Another method is to use the Flash Status Register FSTR.

Bit Number	7	6	5	4	3	2	1	0
Name	-	-	-	-	-	EER[1]	HNG	**RDY**

The RDY bit shows the current state of erase or program. RDY == 1 shows a command finished state.

3.2 Main Flash Access Size

Special attention has to be paid to the Flash Access Size Register (FASZR). For normal code and data fetch the Flash interface is 32-bit wide, but for accessing the Flash interface by write sequence and flag polling it has to be set to 16-bit data width.

The following table shows the two different allowed settings.

Bit Number	7	6	5	4	3	2	1	0
Name	*reserved*						ASZ	
Not allowed	0	0	0	0	0	0	0	0
16-Bit read/write (Erase/Program)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
32-Bit read (ROM mode)	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
Not allowed	0	0	0	0	0	0	1	1
Not allowed	0	0	0	0	0	1	0	0
	. . .							
	1	1	1	1	1	1	1	1

Table 3-8: ASZ Bit Configuration of Main Flash Access Size Register (FASZR)

Note: This register shall only be written, when code is *not* executed in the Flash area!

3.3 Work Flash Access Size

Special attention has to be paid to the Work Flash Access Size Register (WFASZR), if Work Flash is provided. For normal code and data fetch the Flash interface is 32-bit wide, but for accessing the Flash interface by write sequence and flag polling it has to be set to 16-bit data width.

[1]Only available at ECC Flash support.

 © Fujitsu Semiconductor Europe GmbH

The following table shows the two different allowed settings.

Bit Number	7	6	5	4	3	2	1	0
Name	***reserved***							**ASZ**
16-Bit read/write (Erase/Program)	0	0	0	0	0	0	0	**0**
32-Bit read (ROM mode)	0	0	0	0	0	0	0	**1**
Not allowed	0	0	0	0	0	0	1	1
	0	0	0	0	0	1	0	0
	. . .							
	1	1	1	1	1	1	1	1

Table 3-9: ASZ Bit Configuration of Work Flash Access Size Register (WFASZR)

Note: This register shall only be written, when code is *not* executed in the Flash area!

4 Flash Programming Software Example

HOW TO PERFORM A SECTOR ERASE AND HOW TO PROGRAM NEW DATA

4.1 Main Flash Sector Erase – Type 0 and 2 Devices

The following flowchart shows how to program a RAM code section for performing a Flash sector erase.

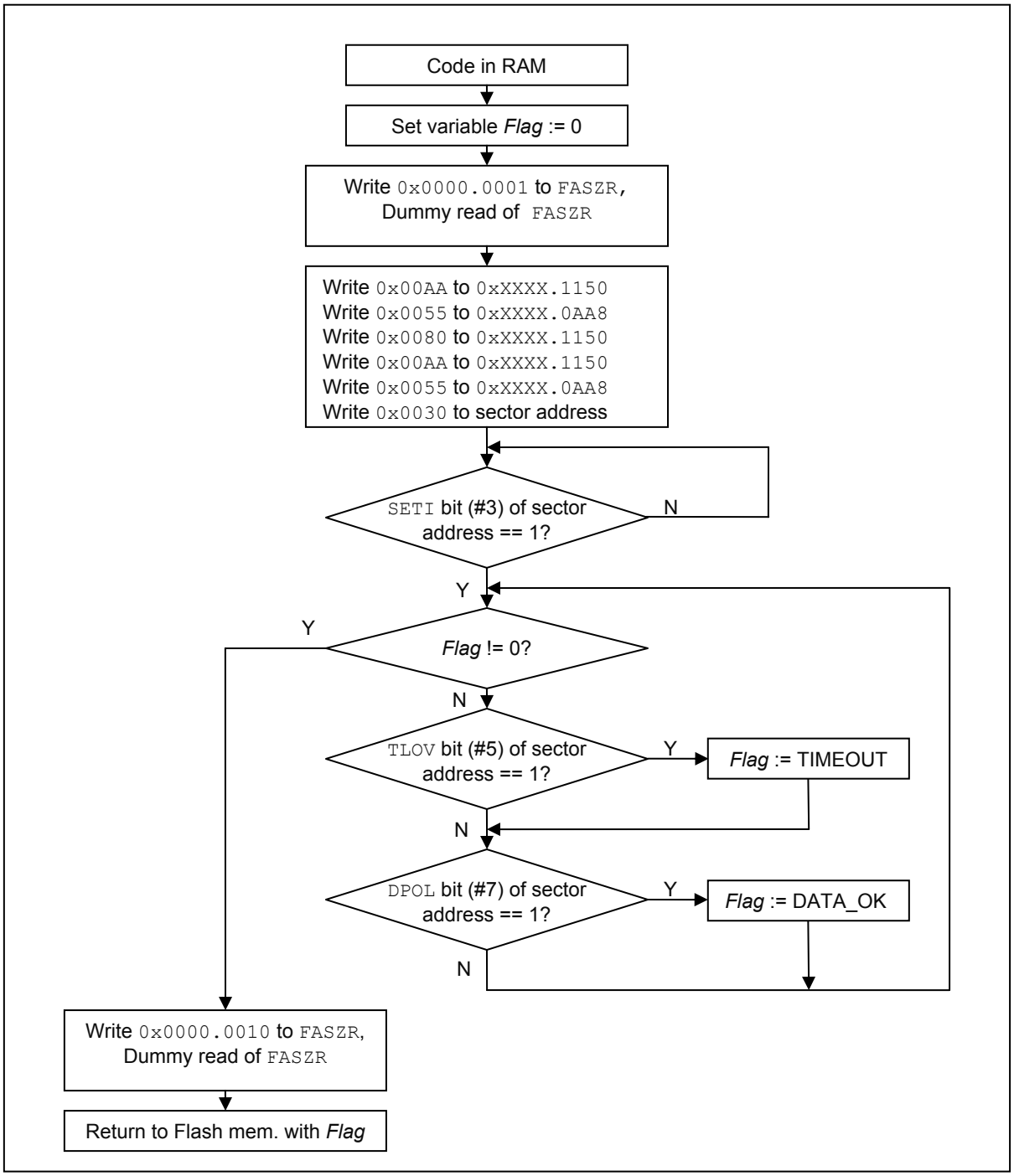

Figure 4-1: Flash Sector Erase Type 0 and 2 Devices Flow Chart

Note, with Work Flash sequence addresses this flow diagram is also valid for Work Flash except that its execution does not need to be in RAM area.

The program code of the sector erase RAM function may look like the following example:

```
#include "mb9bfxxx.h"

#define FLASH_SEQ_1550 ((volatile uint16_t*) 0x00001550) // sequence address 1
#define FLASH_SEQ_0AA8 ((volatile uint16_t*) 0x00000AA8) // sequence address 2

#define FLASH_SECTOR_ERASE_1 0x00AA   // sector erase commands
#define FLASH_SECTOR_ERASE_2 0x0055
#define FLASH_SECTOR_ERASE_3 0x0080
#define FLASH_SECTOR_ERASE_4 0x00AA
#define FLASH_SECTOR_ERASE_5 0x0055
#define FLASH_SECTOR_ERASE_6 0x0030

#define FLASH_DQ7 0x0080     // data polling flag bit (DPOL) position
#define FLASH_DQ5 0x0020     // time limit exceeding flag bit (TLOV) position
#define FLASH_DQ3 0x0008     // sector erase timer flag bit (SETI) position

#define FLASH_TIMEOUT_ERROR  -1
#define FLASH_OK             1

#ifdef __ICCARM__
  #pragma section = ".flash_ram_code"
#endif

#ifdef __ICCARM__
  __ramfunc
#elif  __CC_ARM
  __attribute__ ((section (".ramfunc")))
#else
  #error Please check compiler and linker settings for RAM code
#endif
int32_t FlashRomEraseSector(uint32_t u32SectorEraseAddress)
{
  volatile int32_t  i32FlashFlag = 0;
  volatile uint32_t u32DummyRead;

  FM3_FLASH_IF->FASZR &= 0xFFFD;      // ASZ[1:0] = 2'b01
  FM3_FLASH_IF->FASZR |= 1;
  u32DummyRead = FM3_FLASH_IF->FASZR; // dummy read of FASZR

  *(FLASH_SEQ_1550) = FLASH_SECTOR_ERASE_1;
  *(FLASH_SEQ_0AA8) = FLASH_SECTOR_ERASE_2;
  *(FLASH_SEQ_1550) = FLASH_SECTOR_ERASE_3;
  *(FLASH_SEQ_1550) = FLASH_SECTOR_ERASE_4;
  *(FLASH_SEQ_0AA8) = FLASH_SECTOR_ERASE_5;
  *(volatile uint32_t*)u32SectorEraseAddress = FLASH_SECTOR_ERASE_6;

  // sector erase timer ready?
  while ((*(volatile uint16_t *)u32SectorEraseAddress & FLASH_DQ3) != FLASH_DQ3);

  while (0 == i32FlashFlag)
  {
    // Flash timeout?
    if((*(volatile uint16_t *)u32SectorEraseAddress & FLASH_DQ5) == FLASH_DQ5)
    {
      i32FlashFlag = FLASH_TIMEOUT_ERROR;
    }

    // Data correct?
    if((*(volatile uint16_t *)u32SectorEraseAddress & FLASH_DQ7) == FLASH_DQ7)
    {
      i32FlashFlag = FLASH_OK;
    }
  }

  FM3_FLASH_IF->FASZR &= 0xFFFE;      // ASZ[1:0] = 2'b10
  FM3_FLASH_IF->FASZR |= 0x2;
  u32DummyRead = FM3_FLASH_IF->FASZR; // dummy read of FASZR

  return (i32FlashFlag);
}
```

Figure 4-2: Flash Sector Erase Type 0 and 2 Devices Software Example

4.2 Main Flash Programming – Type 0 Devices

The following flowchart shows how to program a RAM code section for performing Flash 16-bit word programming.

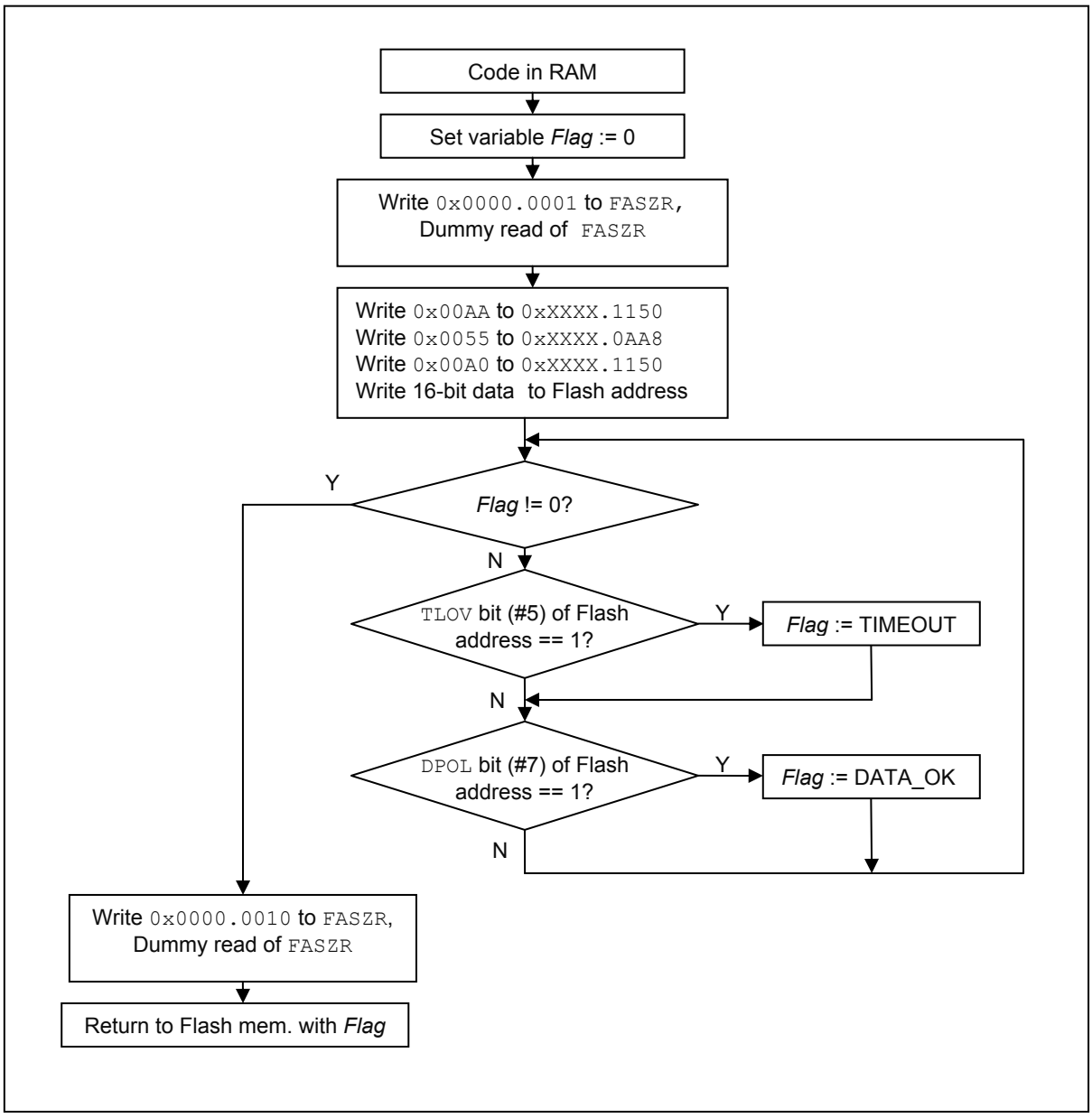

Figure 4-3: Flash Sector Erase Type 0 Devices Flow Chart

Note, with Work Flash sequence addresses this flow diagram is also valid for Work Flash except that its execution does not need to be in RAM area.

 © Fujitsu Semiconductor Europe GmbH

The program code of the Flash programming RAM function may look like the following example:

```
#include "mb9bfxxx.h"

#define FLASH_SEQ_1550 ((volatile uint16_t*) 0x00001550) // sequence address 1
#define FLASH_SEQ_0AA8 ((volatile uint16_t*) 0x00000AA8) // sequence address 2

#define FLASH_WRITE_1 0x00AA   // flash write commands
#define FLASH_WRITE_2 0x0055
#define FLASH_WRITE_3 0x00A0

#define FLASH_DQ7 0x0080       // data polling flag bit (DPOL) position
#define FLASH_DQ5 0x0020       // time limit exceeding flag bit (TLOV) position

#define FLASH_TIMEOUT_ERROR -1
#define FLASH_OK             1

#ifdef __ICCARM__
  #pragma section = ".flash_ram_code"
#endif

#ifdef __ICCARM__
  __ramfunc
#elif __CC_ARM
  __attribute__ ((section (".ramfunc")))
#else
  #error Please check compiler and linker settings for RAM code
#endif
int32_t FlashRomProgram(uint32_t u32ProgramAddress, uint16_t u16ProgamData)
{
  volatile int32_t  i32FlashFlag = 0;
  volatile uint32_t u32DummyRead;

  FM3_FLASH_IF->FASZR &= 0xFFFD;      // ASZ[1:0] = 2'b01
  FM3_FLASH_IF->FASZR |= 1;
  u32DummyRead = FM3_FLASH_IF->FASZR; // dummy read of FASZR

  *(FLASH_SEQ_1550) = FLASH_WRITE_1;
  *(FLASH_SEQ_0AA8) = FLASH_WRITE_2;
  *(FLASH_SEQ_1550) = FLASH_WRITE_3;
  *(volatile uint16_t*)u32ProgramAddress = u16ProgamData;

  while (0 == i32FlashFlag)
  {
    // Flash timeout?
    if((*(volatile uint16_t *)u32ProgramAddress & FLASH_DQ5) == FLASH_DQ5)
    {
      i32FlashFlag = FLASH_TIMEOUT_ERROR;
    }

    // Data correct?
    if((*(volatile uint16_t *)u32ProgramAddress & FLASH_DQ7) == FLASH_DQ7)
    {
      i32FlashFlag = FLASH_OK;
    }
  }

  FM3_FLASH_IF->FASZR &= 0xFFFE;      // ASZ[1:0] = 2'b10
  FM3_FLASH_IF->FASZR |= 0x2;
  u32DummyRead = FM3_FLASH_IF->FASZR; // dummy read of FASZR

  return (i32FlashFlag);
}
```

Figure 4-4: Flash Sector Erase Type 0 Devices Software Example

Note that the `#pragma section` directive is only needed once in a module for the IAR compiler.

4.3 Type 1 Devices

For Type 1 devices the same algorithms can be used. The user only has to modify the sequence addresses according the tables shown in chapter 3.1.

4.4 Main Flash Programming – Type 2 Devices

The following flowchart shows how to program a RAM code section for performing Flash 32-bit word programming.

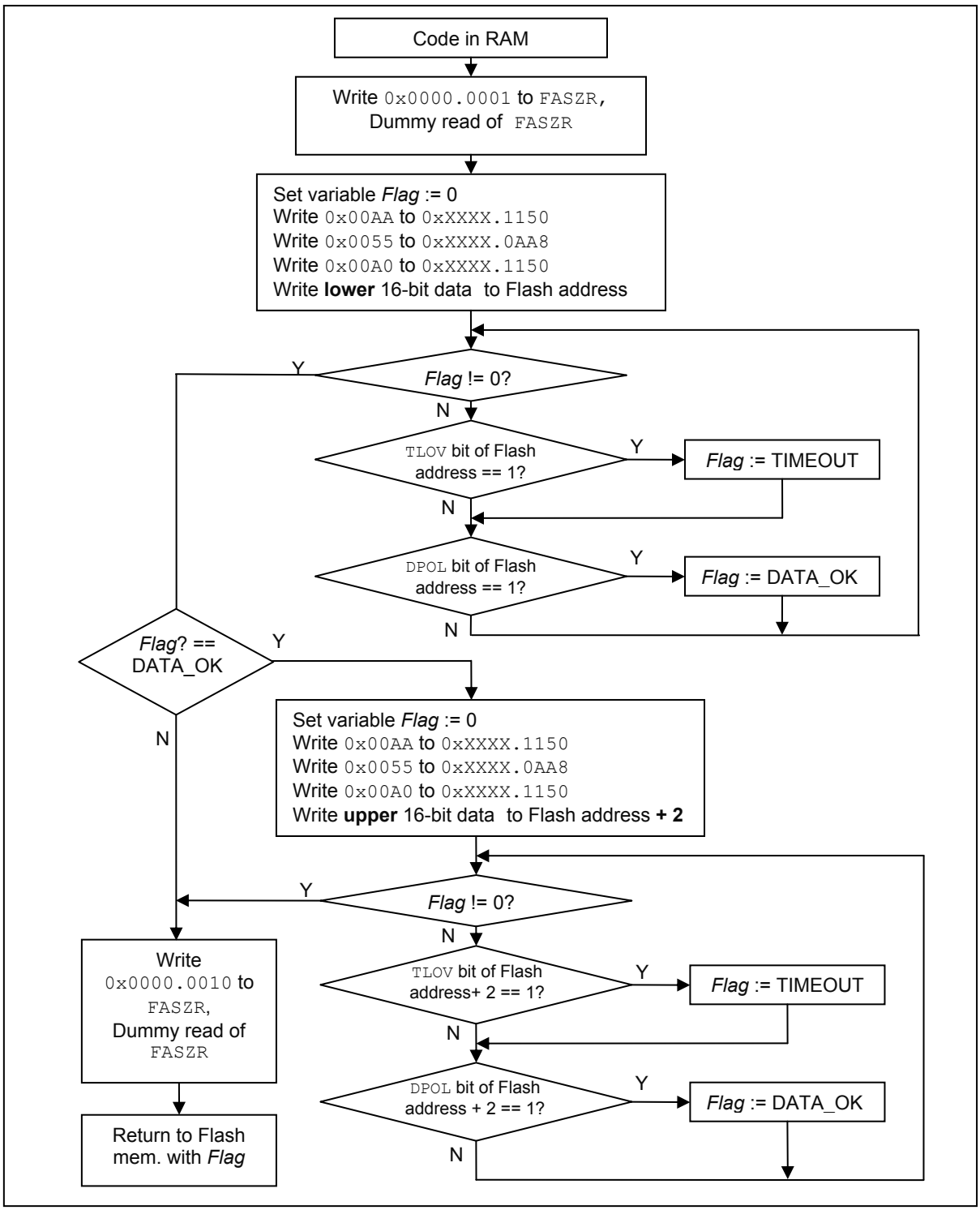

Figure 4-5: Flash Programming Type 2 Device Flow Chart

The program code of the Flash programming RAM function may look like the following example:

```
#include "mb9bfxxx.h"

#define FLASH_SEQ_1550 ((volatile uint16_t*) 0x00001550) // sequence address 1
#define FLASH_SEQ_0AA8 ((volatile uint16_t*) 0x00000AA8) // sequence address 2

#define FLASH_WRITE_1 0x00AA   // flash write commands
#define FLASH_WRITE_2 0x0055
#define FLASH_WRITE_3 0x00A0

#define FLASH_DQ7 0x0080       // data polling flag bit (DPOL) position
#define FLASH_DQ5 0x0020       // time limit exceeding flag bit (TLOV) position

#define FLASH_TIMEOUT_ERROR -1
#define FLASH_OK            1

#ifdef __ICCARM__
  #pragma section = ".flash_ram_code"
#endif

#ifdef __ICCARM__
  __ramfunc
#elif __CC_ARM
  __attribute__ ((section (".ramfunc")))
#else
  #error Please check compiler and linker settings for RAM code
#endif
int32_t FlashDataPolling (uint32_t u32PollAddress, uint16_t u16PollData)
{
  volatile int32_t  i32FlashFlag = 0;
  volatile uint16_t u16DummyRead;

  u16DummyRead = *(volatile uint16_t *)u32PollAddress;
  while(0 == i32FlashFlag)
  {
    // Flash timeout?
    if((*(volatile uint16_t *)u32PollAddress & FLASH_DQ5) == FLASH_DQ5)
    {
      i32FlashFlag = FLASH_TIMEOUT_ERROR;
    }

    // Data correct?
    if((*(volatile uint16_t *)u32PollAddress & FLASH_DQ7) == (u16PollData & FLASH_DQ7))
    {
      i32FlashFlag = FLASH_OK;
    }
  }

  return i32FlashFlag;
}

#ifdef __ICCARM__
  __ramfunc
#elif __CC_ARM
  __attribute__ ((section (".ramfunc")))
#else
  #error Please check compiler and linker settings for RAM code
#endif
int32_t FlashRomProgram(uint32_t u32ProgramAddress, uint32_t u32ProgamData)
{
  volatile int32_t  i32FlashFlag = 0;
  volatile uint32_t u32DummyRead;
  volatile uint16_t u16HalfData;

  FM3_FLASH_IF->FASZR &= 0xFFFD;      // ASZ[1:0] = 2'b01
  FM3_FLASH_IF->FASZR |= 1;
  u32DummyRead = FM3_FLASH_IF->FASZR; // dummy read of FASZR
```

```
  // Data [0:15]
  u16HalfData = (uint16_t) (u32ProgamData & 0x0000FFFF);
  *(FLASH_SEQ_1550) = FLASH_WRITE_1;
  *(FLASH_SEQ_0AA8) = FLASH_WRITE_2;
  *(FLASH_SEQ_1550) = FLASH_WRITE_3;
  *(volatile uint16_t*)u32ProgramAddress = u16HalfData;
  i32FlashFlag = FlashDataPolling(u32ProgramAddress, u16HalfData);

  if (FLASH_OK == i32FlashFlag)
  {
    // Data [16:31] (Set ECC Flash cells)
    u16HalfData = (uint16_t) ((u32ProgamData >> 16) & 0x0000FFFF);
    *(FLASH_SEQ_1550) = FLASH_WRITE_1;
    *(FLASH_SEQ_0AA8) = FLASH_WRITE_2;
    *(FLASH_SEQ_1550) = FLASH_WRITE_3;
    *(volatile uint16_t*)(u32ProgramAddress + 2) = u16HalfData;
    i32FlashFlag = FlashDataPolling((u32ProgramAddress + 2), u16HalfData);
  }

  FM3_FLASH_IF->FASZR &= 0xFFFE;       // ASZ[1:0] = 2'b10
  FM3_FLASH_IF->FASZR |= 0x2;
  u32DummyRead = FM3_FLASH_IF->FASZR; // dummy read of FASZR

  return (i32FlashFlag);
}
```

Figure 4-6: Flash Programming Type 2 Devices Software Example

Note that because the same data polling algorithm has to be used for the 1st lower 16-Bit word and the 2nd upper 16-Bit word, it was rolled out to an own C function (`FlashDataPolling`).

4.5 Project Adjustments for generating RAM Code and automatically Copying at Start-Up Phase

The following paragraphs will explain how to set up RAM code (for Main Flash only devices), which is copied from ROM to RAM at start-up phase, for the IAR and KEIL compilers.

4.5.1 IAR project settings

For compiling a RAM code section the user has to state a `#pragma section` directive with a section name, like *`.flash_ram_code`*. If different compilers should be able to compile the code, this `#pragama` directive has to be set in a `#ifdef __ICCARM__/#endif` pre-processor condition, which uses the predefined macro `__ICCRAM__` of the IAR compiler.

```
#ifdef __ICCARM__
  #pragma section = ".flash_ram_code"
#endif
```

Additionally the RAM functions itself shall get a special qualifier called `__ramfunc`. Also here place it in a pre-processor condition to avoid conflicts with other compilers.

```
#ifdef __ICCARM__
  __ramfunc
#endif
(Function declaration)
```

The last step is to adjust the linker file (*<name>.icf*). The following example shows in bold the additional lines for the RAM code linkage and automatically copying at start-up:

```
/*###ICF### Section handled by ICF editor, don't touch! ****/
/*-Editor annotation file-*/
/* IcfEditorFile="$TOOLKIT_DIR$\config\ide\IcfEditor\cortex_v1_0.xml" */
/*-Specials-*/
define symbol __ICFEDIT_intvec_start__ = 0x00000000;
/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__ = 0x00000000;
define symbol __ICFEDIT_region_ROM_end__   = 0x0007FFFF;
define symbol __ICFEDIT_region_RAM_start__ = 0x1FFF8000;
define symbol __ICFEDIT_region_RAM_end__   = 0x20007FFF;
/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__ = 0x400;
define symbol __ICFEDIT_size_heap__   = 0x800;
/**** End of ICF editor section. ###ICF###*/

define memory mem with size = 4G;
define region ROM_region   = mem:[from __ICFEDIT_region_ROM_start__   to
    __ICFEDIT_region_ROM_end__];
define region RAM_region   = mem:[from __ICFEDIT_region_RAM_start__   to
    __ICFEDIT_region_RAM_end__];

define symbol __RAM_func_start__   = 0x20000000;
define symbol __RAM_func_end__     = 0x20007FFF;
define region RAM_func_region =   mem:[from __RAM_func_start__ to __RAM_func_end__];

define block CSTACK    with alignment = 8, size = __ICFEDIT_size_cstack__   { };
define block HEAP      with alignment = 8, size = __ICFEDIT_size_heap__     { };

initialize by copy { readwrite };
do not initialize  { section .noinit };

place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };
define block RamCode { section .flash_ram_code };
place in RAM_func_region  { block RamCode };

place in ROM_region   { readonly };
place in RAM_region   { readwrite,
                        block CSTACK, block HEAP };
```

Figure 4-7: IAR Linker File

Note that the blue highlighted name for the RAM code section must be the same stated in the #pragma section directive name attribute above.

With these settings the RAM code is placed in ROM, but copied automatically to RAM at start-up phase.

4.5.2 KEIL project settings

For compiling a RAM code function it should get the attribute `__attribute__ ((section (".ramfunc")))` before its declaration. The KEIL compiler of the uVision IDE uses the predefined macro `__CC_ARM` for identification. The attribute shall be put within `#ifdef __CC_ARM/#endif` pre-processor condition to stay compatible with other compilers.

```
#ifdef __CC_ARM
  __attribute__ ((section (".ramfunc")))
#endif
```

The linker settings can be done via the workspace tree. Please move the mouse cursor to the module which contain the RAM code functions, click on the right mouse button and choose “*Options for File*”.

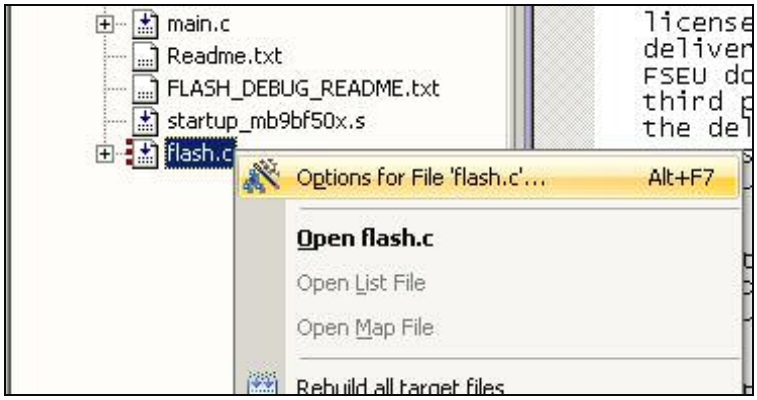

Figure 4-8: KEIL uVision File Options Drop Down Menu

In the options dialog, which follows, choose the tab “Properties” and adjust the *Memory Assignment* for *Code / Const* to an **IRAM** section.

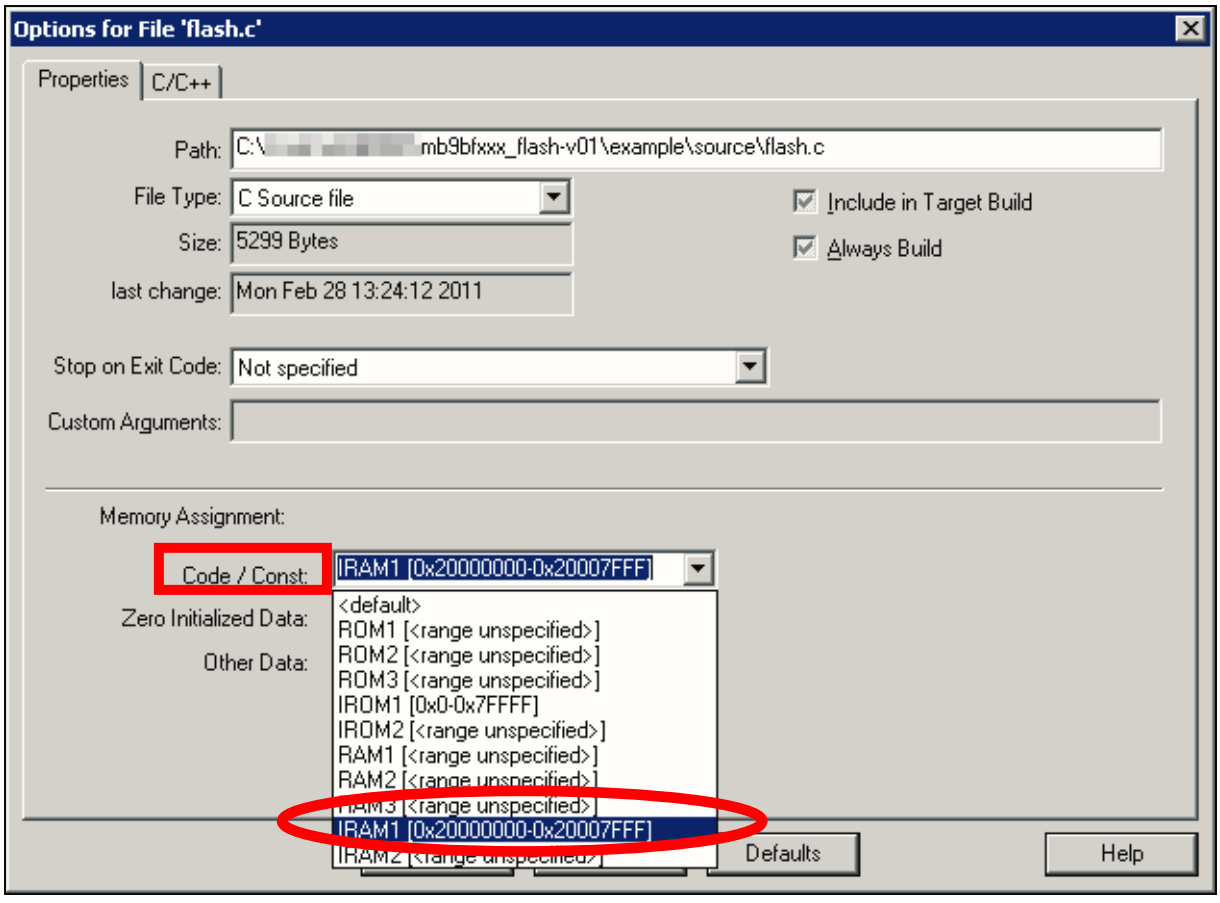

Figure 4-9: uVision File Properties Dialog

Finally click on “*OK*”.

With this adjustment, the RAM code is compiled for RAM but linked to ROM automatically. It is copied at start-up phase from this ROM section to the IRAM section.

4.6 Intercompatibility for different Compilers

For writing RAM code which should be able to be compiled by different compiler the following example can be used. Assume that compiler 1 identifies itself by `__ICC_ARM__` macro, compiler 2 by `__CC_ARM`, compiler 3 by `__YAC_ARM__`, and compiler 4 by `__XYZ_ARM__`.

```
#ifdef __ICCARM__
  // individual setting, qualifier, directive, etc. for compiler 1
#elif __CC_ARM
  // individual setting, qualifier, directive, etc. for compiler 2
#elif __YAC_ARM__
  // individual setting, qualifier, directive, etc. for compiler 3
#elif __XYZ_ARM__
  // individual setting, qualifier, directive, etc. for compiler 4
#else
  #error Please check compiler and linker settings for RAM code
#endif
```

Figure 4-10: Intercompatible Code Example

The `#error` directive is executed, if a compiler is used, which is not identified by the recent compilation process. This shows the user, that he has to take care for compiler-individual RAM code settings.

The linker settings must be done individually in any case by using different linker tools.

5 Additional Information

Information about FUJITSU Semiconductor's Microcontroller can be found on the following Internet page:

http://mcu.emea.fujitsu.com/

The software examples related to this application note are:

mb9bfxxx_flash
mb9af13x_flash
mb9bfd1x_flash
mb9af31x_flash_main_work

It can be found on the following Internet page:

http://mcu.emea.fujitsu.com/mcu_product/mcu_all_software.htm

 © Fujitsu Semiconductor Europe GmbH

List of Figures

List of Tables

 © Fujitsu Semiconductor Europe GmbH