

**F<sup>2</sup>MC-16LX FAMILY**  
16-BIT MICROCONTROLLER  
**MB90F897**

---

**DUAL OPERATION FLASH**

APPLICATION NOTE

## Revision History

Date	Issue
2003-05-29	First version
2003-08-27	V1.1; MWi, Hard-wired Reset Vector Note added
2004-05-05	V1.2: MWi, Sequence in chap. 1.2 corrected

This document contains 12 pages.

## Warranty and Disclaimer

To the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH restricts its warranties and its liability for **all products delivered free of charge** (eg. software include or header files, application examples, target boards, evaluation boards, engineering samples of IC's etc.), its performance and any consequential damages, on the use of the Product in accordance with (i) the terms of the License Agreement and the Sale and Purchase Agreement under which agreements the Product has been delivered, (ii) the technical descriptions and (iii) all accompanying written materials. In addition, to the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH disclaims all warranties and liabilities for the performance of the Product and any consequential damages in cases of unauthorised decompiling and/or reverse engineering and/or disassembling. **Note, all these products are intended and must only be used in an evaluation laboratory environment.**

1. Fujitsu Microelectronics Europe GmbH warrants that the Product will perform substantially in accordance with the accompanying written materials for a period of 90 days from the date of receipt by the customer. Concerning the hardware components of the Product, Fujitsu Microelectronics Europe GmbH warrants that the Product 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.
2. Should a Product turn out to be defect, Fujitsu Microelectronics Europe GmbH's entire liability and the customer's exclusive remedy shall be, at Fujitsu Microelectronics Europe GmbH's sole discretion, either return of the purchase price and the license fee, or replacement of the Product or parts thereof, if the Product is returned to Fujitsu Microelectronics Europe GmbH 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 Fujitsu Microelectronics Europe GmbH, or abuse or misapplication attributable to the customer or any other third party not relating to Fujitsu Microelectronics Europe GmbH.
3. To the maximum extent permitted by applicable law Fujitsu Microelectronics Europe GmbH disclaims all other warranties, whether expressed or implied, in particular, but not limited to, warranties of merchantability and fitness for a particular purpose for which the Product is not designated.
4. To the maximum extent permitted by applicable law, Fujitsu Microelectronics Europe GmbH's and its suppliers' liability is restricted to intention and gross negligence.

### **NO LIABILITY FOR CONSEQUENTIAL DAMAGES**

**To the maximum extent permitted by applicable law, in no event shall Fujitsu Microelectronics Europe GmbH and its suppliers be liable for any damages whatsoever (including but without limitation, consequential and/or indirect damages for personal injury, assets of substantial value, loss of profits, interruption of business operation, loss of information, or any other monetary or pecuniary loss) arising from the use of the Product.**

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

## Contents

<b>REVISION HISTORY</b> .....	<b>2</b>
<b>WARRANTY AND DISCLAIMER</b> .....	<b>3</b>
<b>CONTENTS</b> .....	<b>4</b>
<b>0 INTRODUCTION</b> .....	<b>5</b>
<b>1 DUAL OPERATION FLASH</b> .....	<b>6</b>
1.1 General .....	6
1.1.1 Flash Memory .....	6
1.1.2 Flash Control Registers .....	6
1.1.2.1 Flash Memory Control Status Register (FMCS) .....	6
1.1.2.2 Flash Memory Write Control Register (FWR0/1) .....	7
1.2 Erasing / Programming .....	7
1.2.1 Erase sequence .....	7
1.2.2 Programming (Writing) sequence .....	8
1.2.3 Other sequences .....	8
<b>2 PROGRAMMING EXAMPLE</b> .....	<b>9</b>
2.1 Description .....	9
2.2 Example code .....	9
2.2.1 Definitions .....	9
2.2.2 Sector Erase .....	9
2.2.3 Sector Write .....	10
2.2.4 Main Program .....	11
2.3 Linker Options .....	11

## 0 Introduction

This application note describes how to use the Dual Operation Flash in the MB90F897.

# 1 DUAL OPERATION FLASH

## SHORT DESCRIPTION

### 1.1 General

With the Dual Operation Flash it is possible to erase and program flash sectors during code execution in the other flash bank.

#### 1.1.1 Flash Memory

The flash memory in the MB90F897 is divided into two banks, which are divided into sectors as follows:

	Flash memory	CPU address
Upper bank	SA9 (4 KB)	FFFFFF <sub>H</sub> FFF000 <sub>H</sub>
	SA8 (4 KB)	FEFFFF <sub>H</sub> FEE000 <sub>H</sub>
	SA7 (4 KB)	FDFFFF <sub>H</sub> FFD000 <sub>H</sub>
	SA6 (4 KB)	FCFFFF <sub>H</sub> FFC000 <sub>H</sub>
	SA5 (16 KB)	FBFFFF <sub>H</sub> FF8000 <sub>H</sub>
	SA4 (16 KB)	F7FFFF <sub>H</sub> FF4000 <sub>H</sub>
Lower bank	SA3 (4 KB)	F3FFFF <sub>H</sub> FF3000 <sub>H</sub>
	SA2 (4 KB)	F2FFFF <sub>H</sub> FF2000 <sub>H</sub>
	SA1 (4 KB)	F1FFFF <sub>H</sub> FF1000 <sub>H</sub>
	SA0 (4 KB)	F0FFFF <sub>H</sub> FF0000 <sub>H</sub>

#### 1.1.2 Flash Control Registers

##### 1.1.2.1 Flash Memory Control Status Register (FMCS)

The following table gives a short description of the FMCS register:

Bit	Bit name	Function
0 to 3	Reserved	Always write 0 to it.
4	RDY	Flash programming/erasing status bit (read only); 1 = p/e completed
5	WE	Flash programming/erasing enable; 1 = enable
6	RDYINT	Flash operation flag bit; read: 0 = p/e, 1 = p/e terminated; write: 0 = clear bit, 1 = no effect
7	INTE	Flash p/e interrupt enable; 1 = IRQ after p/e

### 1.1.2.2 Flash Memory Write Control Register (FWR0/1)

This register is intended to protect flash memory sectors from accidental write (or erase). Each sector is represented in a specified bit position:

FWR0:

Bit	Bit name	Function
0	SA0E	Accidental write prevention enable for sector SA0
1	SA1E	Accidental write prevention enable for sector SA1
2	SA2E	Accidental write prevention enable for sector SA2
3	SA3E	Accidental write prevention enable for sector SA3
4	SA4E	Accidental write prevention enable for sector SA4
5	SA5E	Accidental write prevention enable for sector SA5
6	SA6E	Accidental write prevention enable for sector SA6
7	SA7E	Accidental write prevention enable for sector SA7

FWR1:

Bit	Bit name	Function
8	SA8E	Accidental write prevention enable for sector SA8
9	SA9E	Accidental write prevention enable for sector SA9
10	reserved	Always write 0 to this bit
11	reserved	Always write 0 to this bit
12	reserved	Always write 0 to this bit
13	reserved	Always write 0 to this bit
14	reserved	Always write 0 to this bit
15	reserved	Always write 0 to this bit

Once one of these bits is set to “0”, only a power-on or a reset can “unlock” the corresponding sector for programming or erasing.

**Important note:** Be careful with this register. If you want to “lock” a sector for protection, don’t use bit manipulation instructions (SETB, CLR). Mask the corresponding bit with a “0”, set all other bits to “1”, and use a MOV instruction.

## 1.2 Erasing / Programming

The sequences for programming and erasing are much the same as for standard “single operation” flash memories.

### 1.2.1 Erase sequence

Following sequence is needed for erasing a sector:

FFnAAA<sub>H</sub> ← xxAA<sub>H</sub>

FFn554<sub>H</sub> ← xx55<sub>H</sub>

FFnAAA<sub>H</sub> ← xx80<sub>H</sub>

FFnAAA<sub>H</sub> ← xxAA<sub>H</sub>

FFnAAA<sub>H</sub> ← xx55<sub>H</sub>

FFn000<sub>H</sub> ← xx30<sub>H</sub>

( n = SA<sub>nH</sub>, x = don’t care )

### 1.2.2 Programming (Writing) sequence

Following sequence is needed for writing data to a sector:

$FFnAAA_H \leftarrow xxAA_H$

$FFn554_H \leftarrow xx55_H$

$FFnAAA_H \leftarrow xxA0_H$

Sector-Address  $\leftarrow$  Word-Data

( n = SAn<sub>H</sub>, x = don't care )

### 1.2.3 Other sequences

For Read/Reset, Chip Erase, Sector Erase Suspend, Sector Erase Resume, and Auto Select sequences please refer to the hardware manual for details.

## 2 PROGRAMMING EXAMPLE

### HOW TO PROGRAM / ERASE A DUAL FLASH MEMORY SECTOR

#### 2.1 Description

The following example is executed in the upper flash memory bank. During execution it erases sector SA1 of the lower flash memory bank and writes then a data word to the lower address of SA1.

The example code uses medium memory model.

#### 2.2 Example code

##### 2.2.1 Definitions

The following definitions are necessary:

```
//sequence address for SA1:
#define seq_AAAA ((__far volatile unsigned int*)0xFF1AAA)
#define seq_5554 ((__far volatile unsigned int*)0xFF1554)

// sector SA1 start address:
#define SA1      ((__far unsigned int*)0xFF1000)

#define DQ7 0x0080      // data polling flag
#define DQ5 0x0020      // time limit exceeding flag
```

##### 2.2.2 Sector Erase

The following code shows how to erase sector SA1. The function returns 1, if erasing was successful, and 2 if not.

```
int eraseSA1(void)                // Erases sector SA1
{
    unsigned char flag;

    flag = 0;

    FMCS_WE    = 1;                // programming enable
    FWRO       = 0xFF;            // write enable SA1 et al.
    *seq_AAAA  = 0x00AA;          // erase command sequence
    *seq_5554  = 0x0055;
    *seq_AAAA  = 0x0080;
    *seq_AAAA  = 0x00AA;
    *seq_5554  = 0x0055;
    *SA1       = 0x0030;

    while(flag == 0)
    {
        if ((*SA1 & DQ7) == DQ7) // Toggle bit
        {
            flag = 1;            // successful erased
        }
    }
}
```

```

        if ((*SA1 & DQ5) == DQ5) // time out
            if ((*SA1 & DQ7) == DQ7)
            {
                flag = 1;    // successful erased
            }
            else
            {
                flag = 2;    // timeout error
            }
        }
    FMCS_WE = 0;            // disable write enable

    return flag;
}

```

### 2.2.3 Sector Write

The following code shows how to write data to sector SA1. The function returns 1, if writing was successful, and 2 if not.

Note, that only one word is written to the lower address of SA1 (0xFF1000).

```

int writeSA1(unsigned int data)    // Writes a word to lower add. of SA1
{
    unsigned char flag;

    flag = 0;

    FMCS_WE    = 1;                // programming enable
    FWR0       = 0xFF;            // write enable SA1 et al.
    *seq_AAAA  = 0x00AA;          // write command sequence
    *seq_5554  = 0x0055;
    *seq_AAAA  = 0x00A0;
    *SA1       = data;            // send data to the pointed address

    while(flag == 0)
    {
        if ((*SA1 & DQ7) == (data & DQ7))
        {
            flag = 1;
        }
        if ((*SA1 & DQ5) == DQ5)
            if ((*SA1 & DQ7) == (data & DQ7))
            {
                flag = 1;
            }
            else
            {
                flag = 2;
            }
        }
    FMCS_WE = 0;                // reset write enable flag

    return(flag);
}

```

If you want to write more than one word, then change the function header to (e. g.):

```
int writeSA1(volatile __far unsigned int *adr, unsigned int data)
```

Replace all \*SA1 entries in the function block with \*adr.

Note, if FWR0 is not defined in the header file you use, please define it manually with:

```
#define FWR0      ((__far volatile unsigned int*)0x00390A)
. . .
*FWR0 = 0xFF;           // write enable SA1 et al.
. . .
```

## 2.2.4 Main Program

The following code shows an example how to implement the functions above in a program:

```
void Main(void)
{
    unsigned int data;

    . . .

    data = 0xAA;           // data word to be written to flash memory

    . . .

    if (eraseSA1() == 2) error(1);           // erase sector SA1
        else if (writeSA1(data) == 2) error(2); // write data to SA1
            else if (*SA1 != data) error(3); // check written data

    . . .
}
```

Note, `error()` stands for an error handler, which is not described here.

## 2.3 Linker Options

Because the MB90F897 uses a **hard-wired reset vector** pointing to FFE000<sub>H</sub>, the program code has to be linked to this address.

For that use *Project*→*Setup* *Project*→*Linker*→*Category*: Disposition/Connection: ROM:  
Start addr.: **FFE000**, End addr.: FFFFFFFF.

