AN205392

F²MC - 8FX Family, New 8FX Series, EEPROM Library

F²MC-8FX EEPROM library provides EEPROM function based all new 8FX series dual operation flash MCU products. The library runs in the upper bank and could update the data in the lower bank of flash memory to achieve the EEPROM features.

Contents

1 Introduction....1
2 Library Outline....2
2.1 Realization Theory Profile....2
2.2 Target MCU Application....2
2.3 Library Type Description....2
2.4 ROM Usage....3
2.5 Features....3
2.6 Performance....4
3 API Functions....5
3.1 Overview....5
3.2 Read....6
3.3 Write....7
3.4 Define....8
3.5 Restore....9
3.6 Check....10
4 Recommended API Usage....11
4.1 Recommended Initialization Procedure....11
4.2 Recommended Read and Write Procedures....12
4.3 NG Status Handling....15
4.4 PAR ERR Status Handling....16
4.5 Busy Status Handling....16
4.6 Low Power Status Handling....16
4.7 Low Power Mode Transition Procedure....17
4.8 Summarize Exception Handling Procedure....18
5 Project Setting....19
5.1 Overview of Project Setting....19
5.2 Set Code and EEPROM Data Area in Different Flash Bank....19
5.3 Add F²MC-8FX-EEPROM.lib to User Project....22
5.4 Add #include "F²MC_8FX_EEPROM.h" in "main.c"....25
5.5 Flash Interrupt Setting in “vector.c”....26
6 Limitations of EEPROM Library Usage....27
6.1 EEPROM Data Area Updating and Monitor in SOFTUNE....27
6.2 Limitation of Call API by Using "CALL" Command in SOFTUNE....27
6.3 Limitation of Use the Sector Swap Function....27
6.4 Limitation of Flash Interrupt....27
6.5 Limitation of MCU Mode Change....27
6.6 Limitation of Watchdog Timer....27
6.7 Limitation of Variables Definition....28
7 Additional Information....28
Document History....29

1 Introduction

F²MC-8FX EEPROM library provides EEPROM function based all new 8FX series dual operation flash MCU products.

The library runs in the upper bank and could update the data in the lower bank of flash memory to achieve the EEPROM features.

The library provide the below 5 application programming interfaces (API)

1. EEPROM_Check
2. EEPROM_Restore
3. EEPROM_Define
4. EEPROM_B_Write
5. EEPROM_B_Read

2 Library Outline

2.1 Realization Theory Profile

The Library runs in the upper bank flash of target MCU. It control and update EEPROM data in the lower bank of flash memory which comprises two 2K bytes-flash sectors.

In this solution, the lower bank flash is divided into 8 blocks which size is 512 bytes each. EEPROM and flags information is stored in the block. Insides the block, 80 cells are assigned and EEPROM data & address information is stored in it independently.

In the byte-write process, library updates EEPROM data and address in the cell one by one. If all 80 cells of current active block are full, library transfers EEPROM data to the next block in the background. If current active sector are full, library transfers EEPROM data to the next sector and erase the full sector then.

In the byte-read process, library detects the target EEPROM address' cell position from the "position flag table" which is also in current active block. Then, it read out EEPROM data in the detected cell.

All 8 blocks are used in turn and full sector is erased after valid EEPROM data are moved to release the space

Flash erase operation is done in background to achieve the higher write & read performance.

2.2 Target MCU Application

The F^2MC-8FX EEPROM library can be applied for all Cypress new 8FX series dual flash MCU products.

2.3 Library Type Description

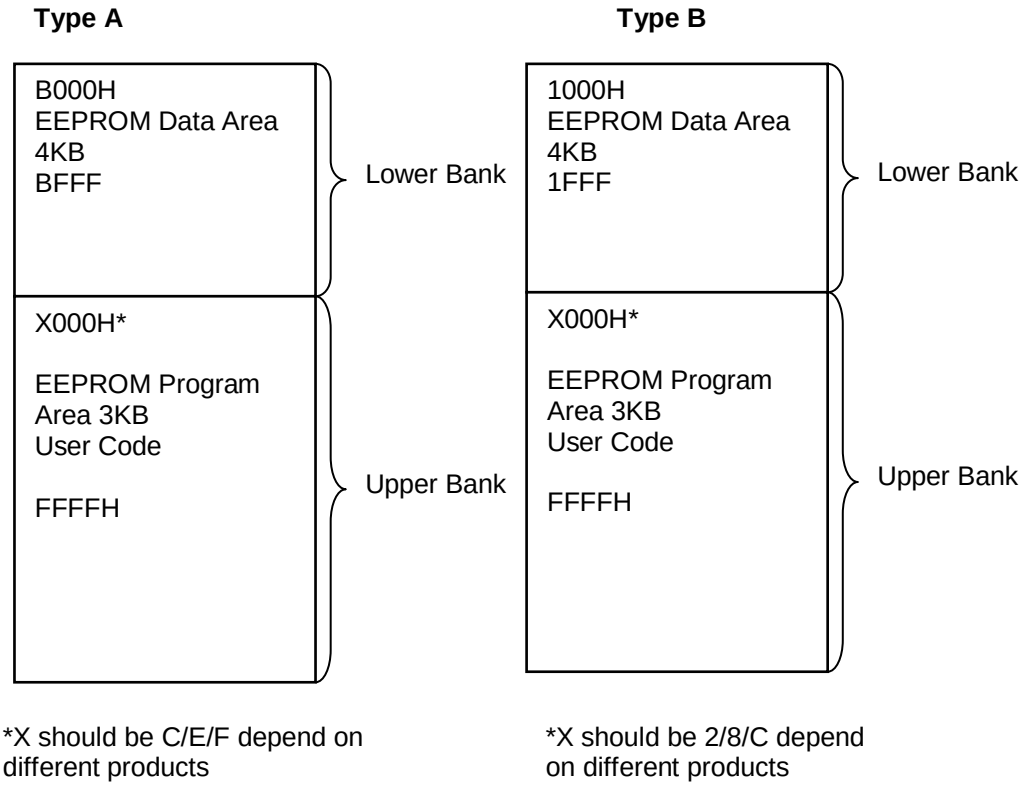
There are five types library for usage in total. Type A and type C are for some products which the flash memory size is 20KB; type B and type E are for some products which the flash memory size is 60KB, type C is for some products which the flash memory size is 36KB. Below table enumerates the products map.

PN	Flash size	Lower bank SA	Upper bank SA	EEPROM
MB95F260	8/12/20KB	$B000_H$	$C/E/F000_H$	TypeA
MB95F350	8/12/20KB	$B000_H$	$C/E/F000_H$	TypeA
MB95F330	8/12/20KB	$B000_H$	$C/E/F000_H$	TypeA
MB95F430	8/12/20KB	$B000_H$	$C/E/F000_H$	TypeA
MB95F390	20/36/60KB	1000_H	$2/8/C000_H$	TypeB
MB95F370	20/36/60KB	1000_H	$2/8/C000_H$	TypeB
MB95F470	20/36/60KB	1000_H	$2/8/C000_H$	TypeB
MB95F310	20/36/60KB	1000_H	$2/8/C000_H$	TypeB
MB95F410	20/36/60KB	1000_H	$2/8/C000_H$	TypeB
MB95F560	8/12/20KB	$B000_H$	$C/E/F000_H$	TypeC
MB95F630	8/12/20/36KB	1000_H	$8/C/E/F000_H$	TypeD
MB95F850	36KB	1000_H	8000_H	TypeD
MB95F860	36KB	1000_H	8000_H	TypeD
MB95F870	36KB	1000_H	8000_H	TypeD
MB95F690	20/36/60KB	1000_H	$2/8/C000_H$	TypeE
MB95F710	20/36/60KB	1000_H	$2/8/C000_H$	TypeE
MB95F770	20/36/60KB	1000_H	$2/8/C000_H$	TypeE
MB95F810	20/36/60KB	1000_H	$2/8/C000_H$	TypeE

2.4 ROM Usage

The ROM usage is described as the following.

Figure 1. Memory Usage Map



2.5 Features

1. EEPROM size: 16, 32, 64, 128, 256, 512 bytes
2. Max EEPROM re-write cycle
   - 16’000’000 times when 512 bytes size defined
   - 32’000’000 times when 256 bytes size defined
   - 64’000’000 times when 128 bytes and lower size defined
3. Retention time: 20 years
4. ROM needed: 7K bytes
   - ROM for EEPROM data: 2*2K bytes
   - ROM for library: 3K bytes
5. RAM: 20B in direct addressing area
6. Register: R0, R1 are used in the library
7. Stack: 44B stack area are used in the library

2.6 Performance

2.6.1 EEPROM Performance Comparison

Below table shows the performance comparison between traditional I²C EEPROM and emulated EEPROM.

Table 1. F²MC-8FX EEPROM Library Performance

EEPROM Performance Comparison											
		I²C	Library V2.0			Library V2.0			Library V2.0		
Max transport speed		100KHz	MCU @ 12.5MHz			MCU @8MHz			MCU @ 1MHz		
			512B	256B	128/64/32/16B	512B	256B	128/64/32/16B	512B	256B	128/64/32/16B
Single byte write	Normal	100us	180us	180us	180us	220us	220us	220us	1320us	1320us	1320us
	Max	5ms	50ms	25ms	12.5ms	70ms	35ms	17.5ms	530ms	265ms	132.5ms
Continue write 81B	Normal	13ms	14.58ms	14.58ms	26.9ms	17.82ms	17.82ms	35.1ms	106.92ms	106.92ms	238.1ms
	Max		64.4ms	39.4ms	26.9ms	87.6ms	52.6ms	35.1ms	635.6ms	370.6ms	238.1ms
Continue write 161B	Normal	21ms	28.98ms	28.98ms	53.62ms	35.2ms	35.2ms	69.98ms	211.2ms	211.2ms	474.88ms
	Max		578.8ms	78.62ms	53.62ms	605.2ms	70.2ms	69.98ms	1241.2ms	739.88ms	474.88ms
Single byte read	Average	200us	72us	72us	72us	122us	122us	122us	900us	900us	900us
	Max		206us	206us	206us	266us	266us	266us	1210us	1210us	1210us

2.6.2 Explanation of Emulated EEPROM Performance

This performance is measured and calculated in the normal condition. Such as the flash completion time is 500mS*, this time may shorter/longer depend on the environment (voltage, temperature…)

Normal case means write data in the average distribution of EEPROM address.

Max means continually write into the same EEPROM address, in this case, data copy and erase action should be trigger.

Average means in read process, 0 ~ 10 cell need to check, and checking the 5th cell is average.

* This value is the maximum erase completion time which refers from datasheet.

2.6.3 Conclusion

The single byte write/read time of emulated EEPROM is longer/shorter than the I²C EEPROM.

3 API Functions

3.1 Overview

API	Function
Read	Reads one data from the emulated EEPROM.
Write	Writes one data in the emulated EEPROM and takes care of copying data from old block/sector to new block/sector and erasing old sector.
Define	Initializes the emulated EEPROM in one of predefined sizes.
Check	Checks consistency of the data stored in the emulated EEPROM and the information* stored in RAM. It informs whether it is blank, consistent, broken but it can be restored or broken and it cannot be restored. This function is intended to be performed every time after power on. Therefore it reconstructs the EEMS stored in RAM that is volatile by its nature.
Restore	Restores the emulated EEPROM including the EEMS in RAM. This function is intended to solve data inconsistency that is caused by incomplete execution of the API software.

* To emulate EEPROM using flash memory, some information need to be defined in RAM area, it is called EEMS (emulated EEPROM Management Structure) hereafter.

3.2 Read

API function		Read (Address, Data)	
Return value	Data	Read value	
	Status	OK	Read successful
		NG	Data cannot be read /Flash operation Timeout by preceding API
		PAR ERR	Input parameter error
		BUSY	Already executing Define, Restore or Check API
		LOW POWER	N/A
Multiple access*	Read	Possible	
	Write	Possible	
	Define	Not possible	
	Restore	Not possible	
	Check	Not possible	
Background Erase operation		Suspends Erase operation	

* Multiple access: Please refer to section 4.2.1 for details.

3.2.1 Interface Description

Name: unsigned char EEPROM_B_Read (unsigned int uiAdr, unsigned char *ucData)

Function: Read data (one byte) from emulated EEPROM input address

Input:

uiAdr → The address data read from

*ucData→ Store and return the read data

Output:

0 → OK (Read successfully)

1 → NG (Data cannot be read/ flash operation timeout by preceding API)

2 → BUSY (Already executing Define, Restore or Check API)

4 → PAR_ERR (Input parameter out of range)

Note:

The return values indicate function execution status, and ucData is a pointer to return the read data according input address.

3.3 Write

API function		Write(Address, Data, Flag)	
Return value	Flag	NIL/OVERFLOW	
	Status	OK	Write successful
		NG	Data cannot be written /Write operation Timeout /Flash operation Timeout by preceding API
		PAR ERR	Input parameter error
		BUSY	Already executing other API /Write access overflow
		LOW POWER	In Low Power Mode
Multiple access	Read	Not possible	
	Write	Not possible	
	Define	Not possible	
	Restore	Not possible	
	Check	Not possible	
Background Erase operation		Suspends Erase operation /Returns BUSY	

3.3.1 Interface Description

Name: unsigned char EEPROM_B_Write (unsigned int uiAdr, unsigned char ucData, unsigned char *OverFlowF)

Input:

uiAdr → The address data written into

ucData → The written data

*OverFlowF → Indicate overflow status, please refer to section 4.2.2 for details.

Output:

0 → OK (Write successfully)

1 → NG (Data cannot be written/ Write operation timeout/ Flash operation timeout by preceding API)

2 → BUSY (Already executing other API/ Write access overflow)

3 → LOWPOWER (Indicate MCU in low power mode)

4 → PAR_ERR (Input parameter out of range)

Note

The return value indicates the function execution status, and OverFlowF is a pointer to indicate overflow situation occurs.

3.4 Define

API function		Define(Size)	
Return value	Data	N/A	
	Status	OK	Define successful
		NG	/Flash operation Timeout /Flash operation Timeout by preceding API
		PAR ERR	Input size error
		BUSY	Already executing other API /Background Erase operation
		LOW POWER	In Low Power Mode
Multiple access	Read	Not possible	
	Write	Not possible	
	Define	Not possible	
	Restore	Not possible	
	Check	Not possible	
Background Erase operation		Returns BUSY	

3.4.1 Interface Description

Name: unsigned char EEPROM_Define (unsigned char ucSize)

Function: Define an emulated EERPOM, initial EEMS and ROM for further usage.

Input:

ucSize → Emulated EEPROM size

Output:

0 → OK (Define successful)

1 → NG (Flash operation timeout/ flash operation timeout by preceding API)

2 → BUSY (Already executing other API/ Background erase operation)

3 →LOWPOWER (Indicate MCU in low power mode)

4 → PAR_ERR (Input parameter error)

Note

Input parameter (ucSize) should equal predefine values (16B, 32B, 64B, 128B, 256B, 512B)

This function is intended to define an emulate EEPROM. The 16B, 32B, 64B, 128B, 256B, 512B can be supported. Firstly, API will erase sector 1, and then, write emulate EEPROM information into block configuration area, including block flag, ID, size. EEMS should be initialized too. In older to ensure sector 2 is blank, an erase operation is necessary (Background erase).

3.5 Restore

API function		Restore()	
Return value	Data	N/A	
	Status	OK	Restore successful
		NG	Data cannot be restored /Flash operation Timeout /Flash operation Timeout by preceding API
		PAR ERR	N/A
		BUSY	Already executing other API /Background Erase operation
		LOW POWER	In Low Power Mode
Multiple access	Read	Not possible	
	Write	Not possible	
	Define	Not possible	
	Restore	Not possible	
	Check	Not possible	
Background Erase operation		Returns BUSY	

3.5.1 Interface Description

Name: unsigned char EEPROM_Restore (void)

Function: Restore emulated EEPROM data to new sector if the old sector is damaged.

Input:

None

Output:

0 → OK (Restore successful)

1 → NG (Data cannot be restored/ Flash operation timeout/ Flash operation timeout by preceding API)

2 → BUSY (Already executing other API/ Background erase operation)

3 → LOWPOWER (Indicate MCU in low power mode)

The function is intended to restore the emulated EEPROM data that has become inconsistent because of incomplete execution of the API software. This inconsistency is conceived to occur in case of power failure during execution of the Write, Define or Restore API functions. Other data inconsistencies that do not show match with the incomplete execution of these API functions cannot be restored. For example, unintended Write/Erase operation (Unintended manipulation of the emulated EEPROM) performed by user software cannot be covered.

3.6 Check

API function		Check (Size, Data)	
Return value	Data	DEFINE/REDEFINE/RESTORE/CONSISTENT	
	Status	OK	Check successful
		NG	Flash operation Timeout by preceding API
		PAR ERR	Input size error
		BUSY	Already executing other API /Background Erase operation
		LOW POWER	N/A
Multiple access	Read	Not Possible	
	Write	Not possible	
	Define	Not possible	
	Restore	Not possible	
	Check	Not possible	
Background Erase operation		Returns BUSY	

3.6.1 Interface Description

Name: unsigned char EEPROM_Check (unsigned char ucSize, unsigned char *ucData)

Function: Check emulated EEPROM if is ready to use.

Input:

ucSize → The emulated EEPROM size

*ucData→ To store check result

0 → CONSISTENT

1 → DEFINE

2 → RESTORE

3 → REDEFINE

Output:

0 → OK (Check successful)

1 → NG (Flash operation timeout by preceding API)

2 → BUSY (Already executing other API/ Background erase operation)

4 → PAR_ERR (Input parameter error)

Note:

The return value only tells check function status to user, and input parameter is a pointer (*ucData) will store the check result as a reference.

This function is intended to be performed every time after power on. Therefore it reconstructs the EEMS stored in RAM that is volatile by its nature.

The block configuration information, erase status, unused data area and update flag would be checked by check API. According to the return value, user code could determine the next action.

4 Recommended API Usage

4.1 Recommended Initialization Procedure

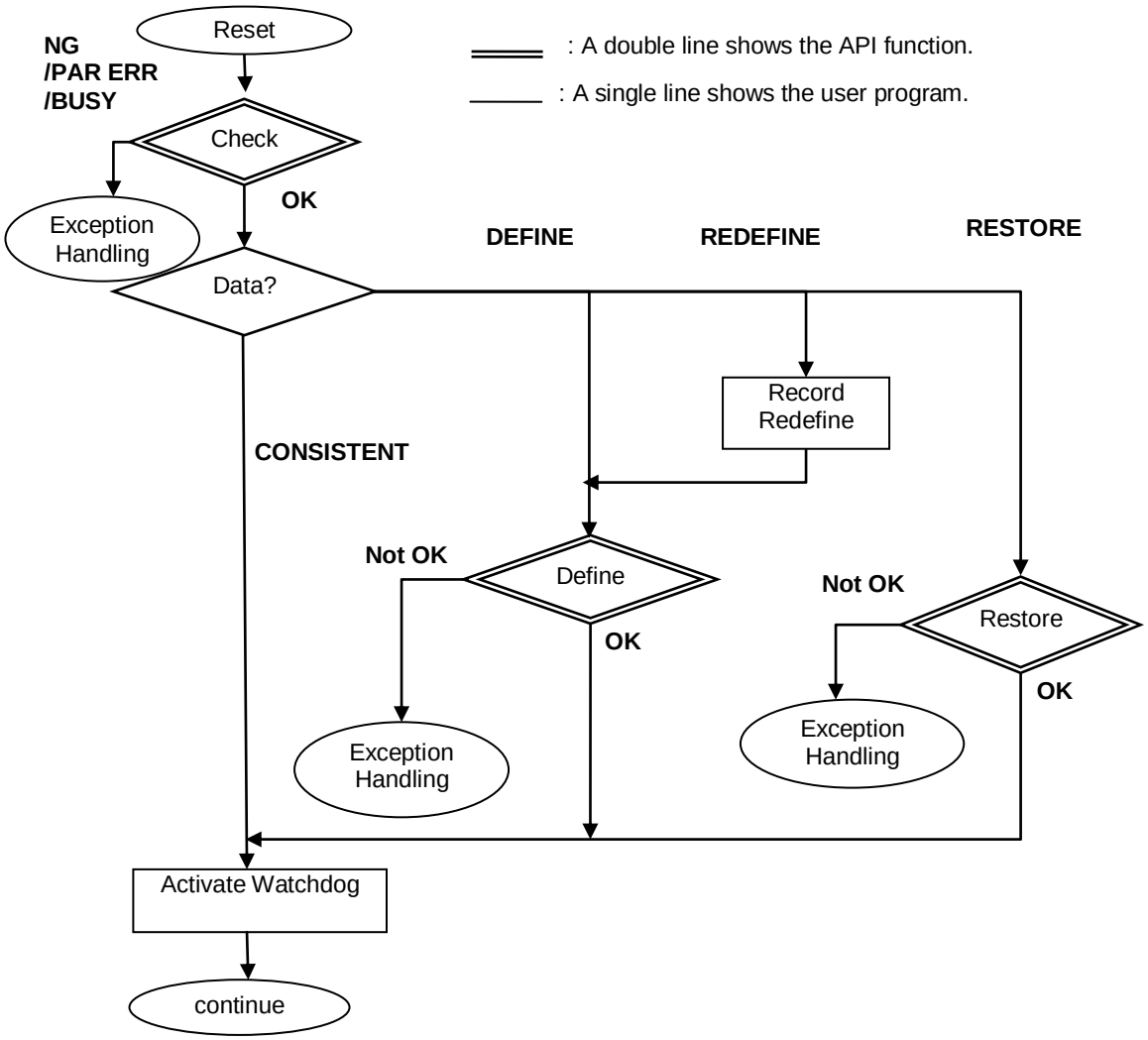
Every time after power on or reset, the user software must perform the Check API function. This is the only way to start accessing the emulated EEPROM by the API software.

Depending on the result of the Check API function, if the check result is not “CONSISTENT”, the user software should perform Define or Restore API function.

This initialization procedure must be completed before the first read/write access to the emulated EEPROM after power on or reset.

It is recommended to activate the watchdog timer after this initialization procedure in order to prevent unexpected watchdog reset that may be caused by longer execution time of the Define and Restore functions involving the Erase operation of the Flash memory.

The figure below illustrates the recommended initialization procedure.



4.2 Recommended Read and Write Procedures

4.2.1 Competition between Read and Write Accesses

The Read API function can be successfully performed during the execution of the Write API function. On contrary, the Write API function returns BUSY when the Read API function is performed.

This effectively means that the Write API function should not be performed in an interrupt service routine unless both the Write and Read API functions are performed in the same interrupt service routine.

The following table shows possible combinations for locating the Write and Read API functions.

Read API function	Write API function	Recommendation
Main routine	Main routine	Recommended
Main routine	Interrupt service routine	Not recommended
Interrupt service routine	Main routine	Possible
Interrupt service routine	Different interrupt service routine	Not recommended
Interrupt service routine	Same interrupt service routine	Recommended

If the user software requires performing the write access to the emulated EEPROM in an interrupt service routine, it is recommended to write the data in user-defined buffer RAM in the interrupt service routine and the buffered data should be written in the emulated EEPROM by the Write API function in the main routine.

Besides, multiple Write API functions should be performed sequentially. Therefore all the Write API functions should reside within the main routine or within the same interrupt service routine.

4.2.2 EEPROM Write Access Overflow

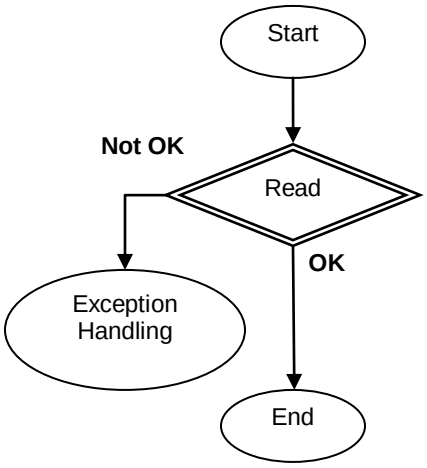
The Write API function may return BUSY as status and OVERFLOW as data if the frequency of write accesses is so high that one whole Flash sector is used up within one Erase operation time.

If it occurs, the only way with the user software is to discard the write data which resulted in BUSY and OVERFLOW then to record/signal that EEPROM write access overflow has occurred.

Otherwise, the user software should construct a write buffer in RAM to absorb the excess write accesses then it should write the buffered data to the emulated EEPROM when the write access frequency is lower.

4.2.3 Recommended Read Procedure

The figure below illustrates the recommended Read procedure.

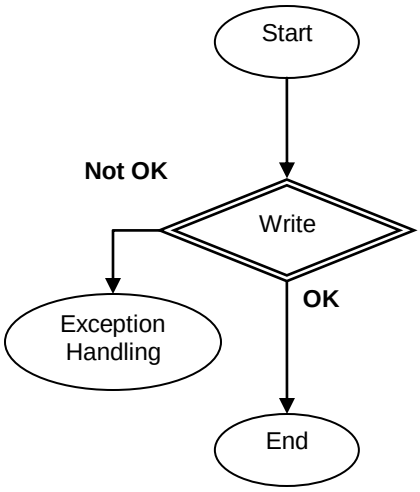


: A double line shows the API function.

: A single line shows the program.

4.2.4 Recommended Write Procedure without Buffer RAM

The figure below illustrates the recommended Write procedure without buffer RAM. In this recommended procedure, the write access overflow results in discarding data.

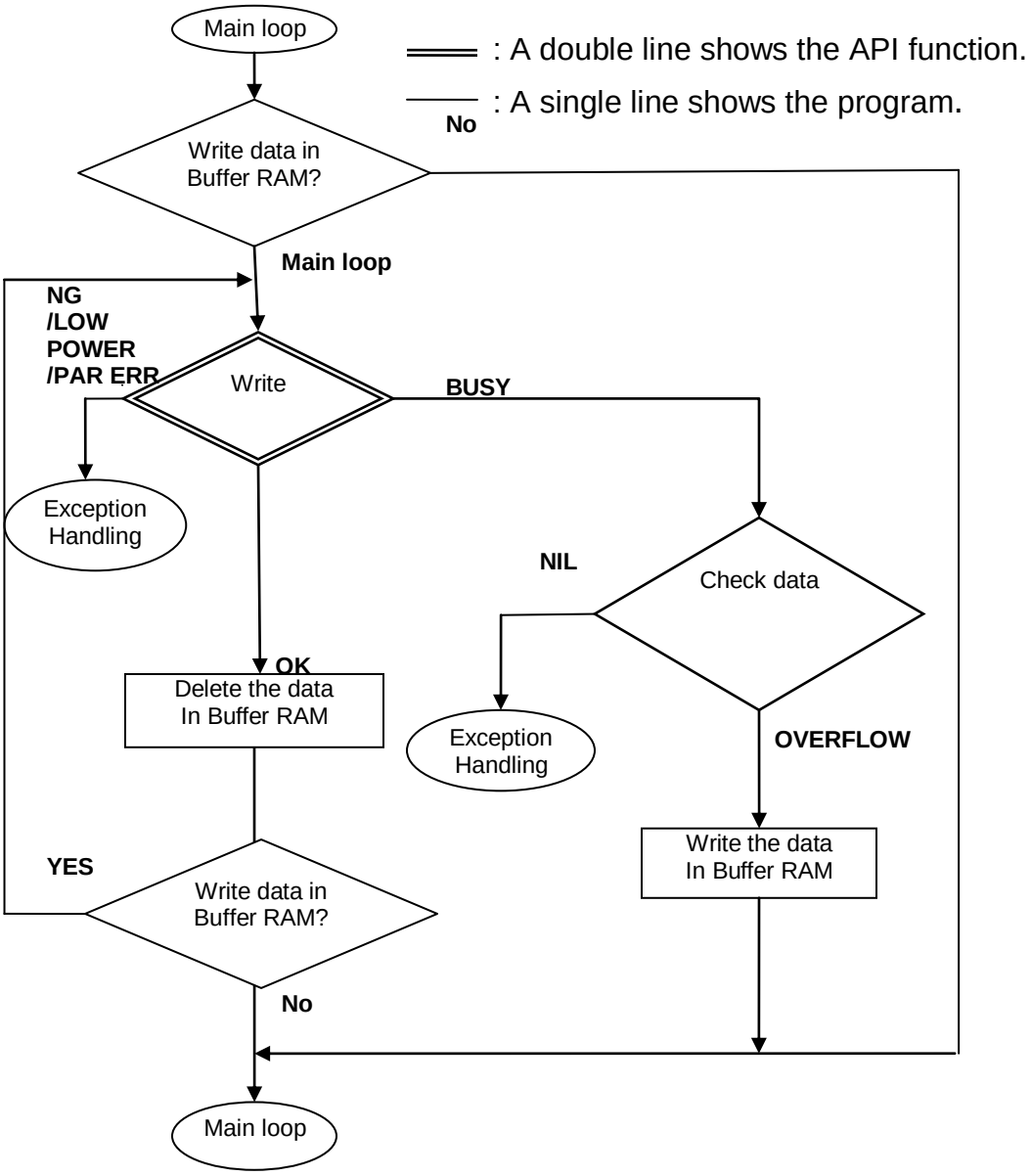


: A double line shows the API operation.

: A single line shows the program.

4.2.5 Recommended Write Procedure with Buffer RAM

The figure below illustrates the recommended Write procedure with buffer RAM. In this recommended procedure, the write access overflow is handled by buffer RAM. The user software performs the write access to the emulated EEPROM via user-defined buffer RAM.



4.3 NG Status Handling

The hardware has physically failed unless it is caused by the Timeout with the Write/Erase operation in the Flash memory.

In order to identify the cause, the user software can examine the Timeout status by reading the status flag in the Flash memory (flash timeout status) or by performing the Check API function.

In this case, it is recommended to perform exception handling and/or the microcontroller has to be initialized.

The Timeout status indicates that the Write/Erase operation has not been completed.

This situation is similar to that of power failure during the Write/Erase operation.

If Timeout occurs, the user software may issue the Reset command to the Flash memory then it may perform the Check API function followed by the Restore or Define API function. In this way, Write/Read operations in the Flash memory are re-performed.

However, instead of making attempt to restore the data, the user software may simply initialize the microcontroller. In this case, the initialization procedure will take care of the rest.

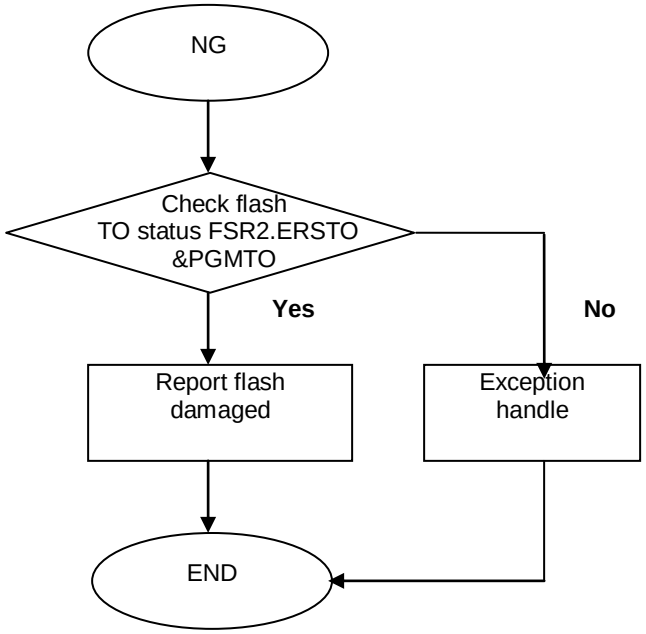
Thus, in the event of Timeout, the user software basically has two options.

1. To make attempt to restore. But, the last written data may be lost when the Restore API function is performed. Restore API also returns NG when the data cannot be restore
2. To simply initialize microcontroller

In order to simplify the matter, it can be generally recommended to initialize if there is no specific requirement to avoid initialization or to achieve higher safety level.

In case of the user application requiring higher level of safety, it is recommended to introduce redundant used of the emulated EEPROM, for example, mirroring of data in the emulated EEPROM.

For NG status, checking flash program/erase time-out is recommended for user code because this indicates flash has been damaged.



4.4 PAR ERR Status Handling

The return value of PAR_ERR means that the user software has failed with calculation of logical address or size of the emulated EEPROM. The whole system reset is recommended.

4.5 Busy Status Handling

If the recommended use of the API functions in this chapter is followed, the BUSY state suggests that there is a software bug or physical failure in the hardware except for the BUSY state caused by the write access overflow.

The write access overflow can be identified by examining the data return value of the Write API function.

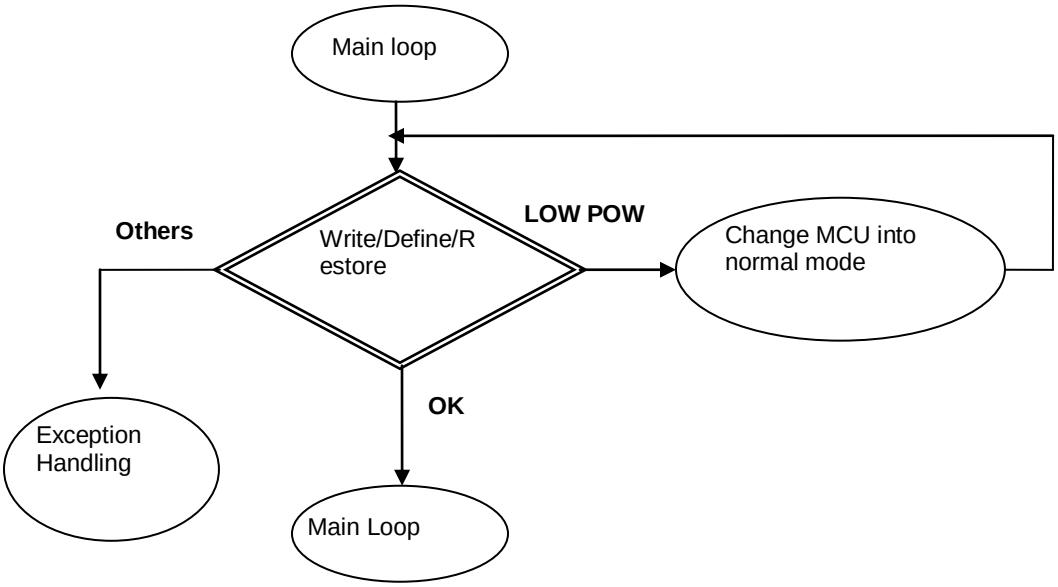
If it is write access overflow, the user software should follow the recommendation describe in Section 4.2.5.

Otherwise, the user software has to perform exception handling and/or the microcontroller has to be initialized.

4.6 Low Power Status Handling

The Write, Define and Restore functions should not be performed when the microcontroller is in the low power mode. The LOW POWER state indicates this error and stop flash write and erase operation because MCU is in sub-clock mode.

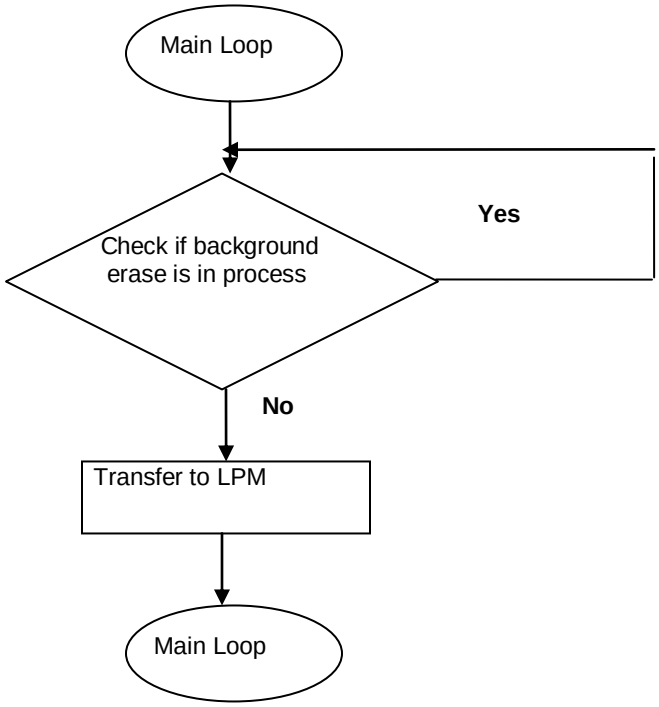
In this case, user code changing MCU mode into normal mode is recommended



4.7 Low Power Mode Transition Procedure

In order to avoid any unexpected execution of the API software in Low Power mode, certain precautions have to be taken.

The figure below illustrates the recommended low power mode transition procedure.



The following table shows possible combinations for locating the low power mode transition and write API function.

Low power mode transition	Write API function	Recommendation
Main routine	Main routine	Recommended
Main routine	Interrupt service routine	Not recommended
Interrupt service routine	Main routine	Not recommended
Interrupt service routine	Different interrupt routine	Not recommended
Interrupt service routine	Same interrupt service routine	Recommended

4.8 Summarize Exception Handling Procedure

If there is a possibility that software bug or hardware failure caused any of NG, BUSY, PAR ERR and LOW POWER statuses, the microcontroller has to be initialized or the user software has to set the microcontroller into a safe mode.

However, possibilities of Timeout and EEPROM write access overflow may require different handling.

Therefore, the status value has to be carefully evaluated in an exception handling routine in the user software.

The following tables are the recommended evaluation cases.

NG	BUSY	LOW POWER	PAR ERR	Recommended action
X	X	1	X	Initialize
		X	1	
X	X	0	0	Check Overflow & Timeout

It is suggested that the microcontroller should be initialized if the status indicates LOW POWER or PAR ERR regardless of other state because the LOW POWER or PAR ERR state can be considered as a fatal error.

If it is not LOW POWER or PAR ERR, the user software should examine if there was EEPROM access overflow or Timeout. The table below suggests the next evaluation cases.

Overflow	Timeout	Recommended action
0	0	Initialize
X	1	Timeout handling
1	0	Overflow handling

In this table, it is suggested that the microcontroller should be initialized if there was Timeout regardless of EEPROM access overflow because Timeout can be considered as more serious error than EEPROM access overflow and EEPROM access overflow may be an outcome of Timeout.

Timeout handing may be simple initialization of the microcontroller or it may be an attempt to restore the EEPROM data then it may return to the originating routine.

Overflow handling may be just returning to the originating routine or it should be treated as an error of certain significance and the initialization may be performed if the user software does not allow the EEPROM access overflow.

In any case, it is generally recommended to store the evaluation result in RAM for later use of failure interpretation.

5 Project Setting

5.1 Overview of Project Setting

To correctly use EEPROM library, take care of the following settings.

1. Set the code and EEPROM data area in different flash bank
2. Add F2MC_8FX_EEPROM.lib to user project
3. Add #include "F2MC_8FX_EEPROM.h" in " main.c"

5.2 Set Code and EEPROM Data Area in Different Flash Bank

For dual operation flash, EEPROM area is in lower bank and the code area is in upper bank. The start address and the end address can be set to an arbitrary address of upper bank address area.

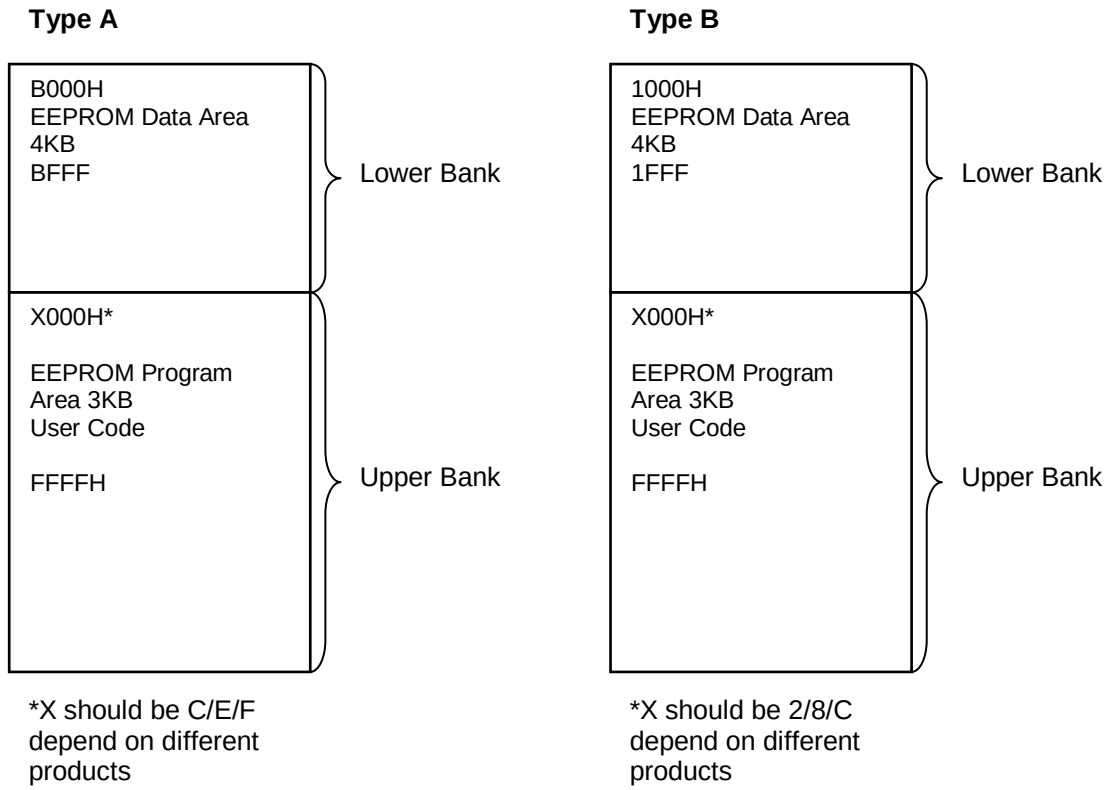
The start address should be larger than or equal to start address of upper bank, and should be less than the end address, the end address should be less than or equal to 0XFFFF.

For example, the start address is 0XC000 and the end address is 0XDFFF, and so on.

The code area is in the upper bank, and the code will be compiled to the code area by the compiler, so the code is in upper bank.

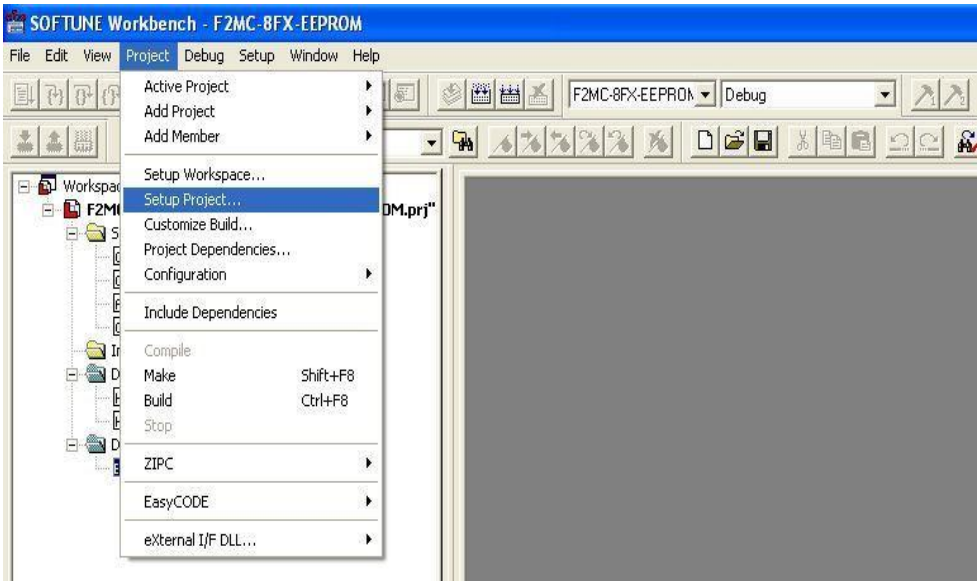
The memory usage map is as below.

Figure 2. Memory Usage Map



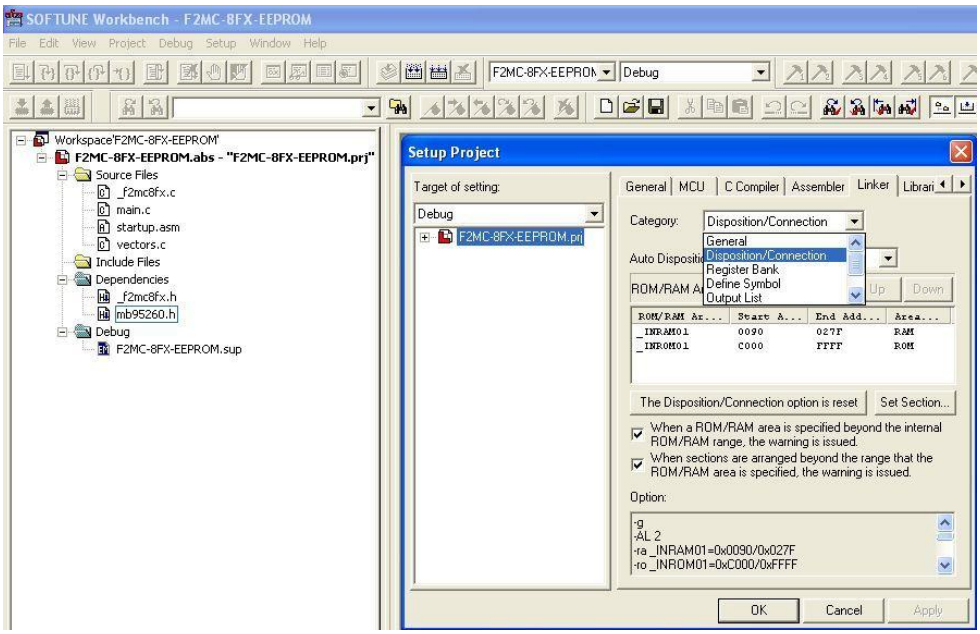
In SOFTUNE, select **Project→Setup Project**.

Figure 3. Open Setup Project Window



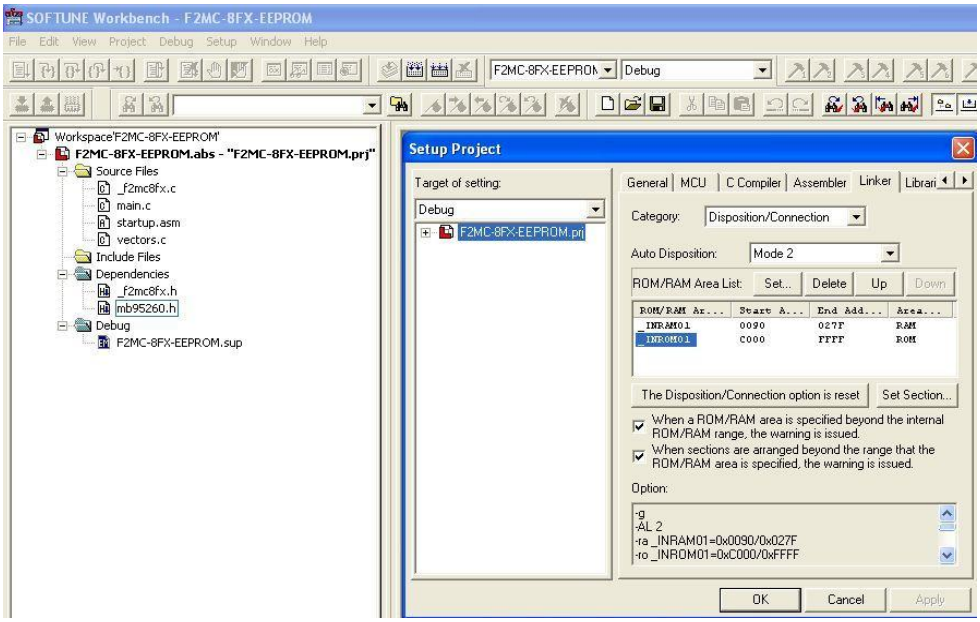
In Setup Project window, select Linker→Disposition/Connection.

Figure 4. Select Disposition/Connection



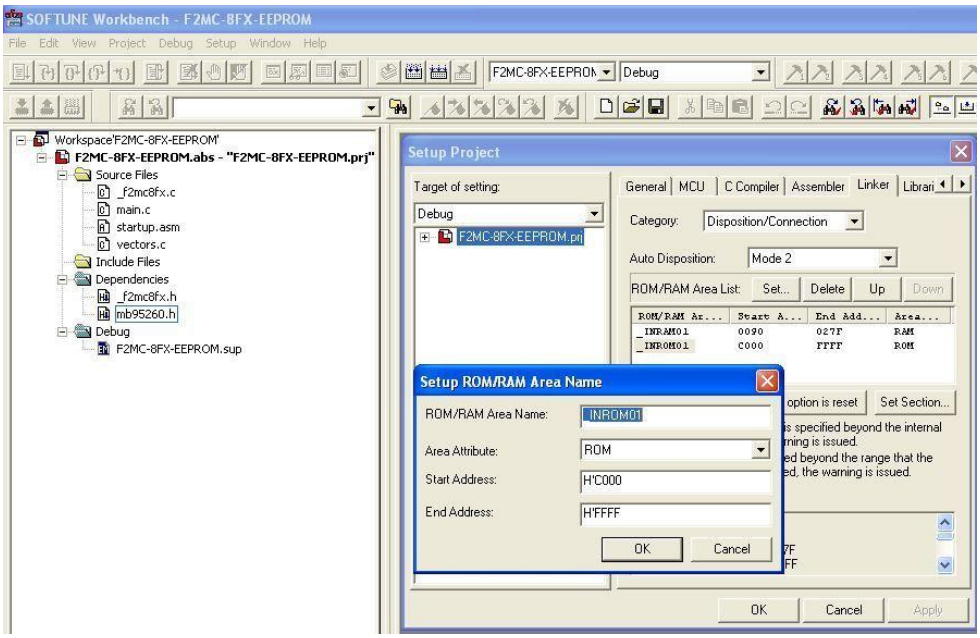
Select **_INROM01** and click **Set**.

Figure 5. Open Setup ROM/RAM Area Name Window



In **Setup ROM/RAM Area Name**, set **Start Address**, such as 0XC000 (between 0XC000 and 0XFFFF), set **End Address**, such as 0XDFFF (between 0XC000 and 0XFFFF), and click **OK** to finish the setting.

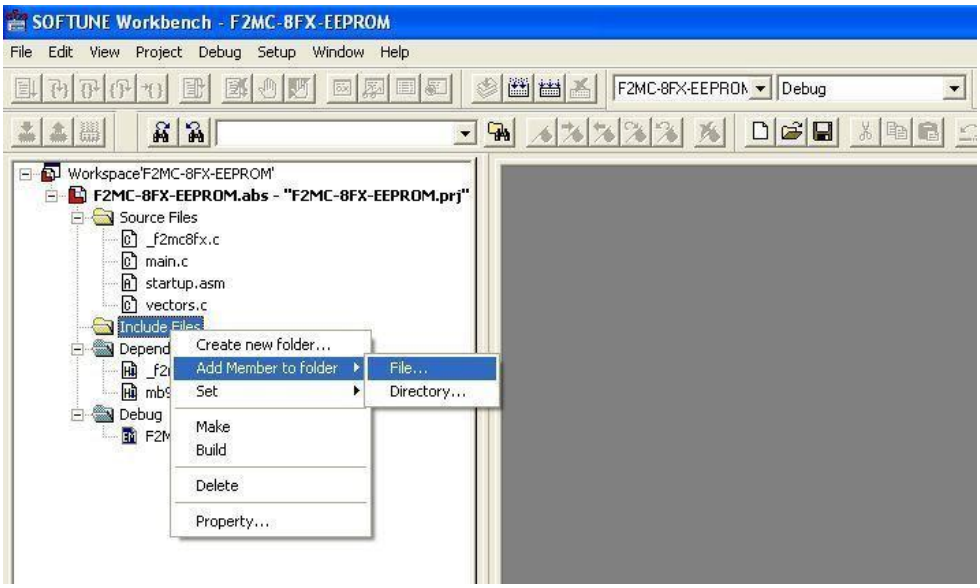
Figure 6. Set ROM



5.3 Add F²MC-8FX-EEPROM.lib to User Project

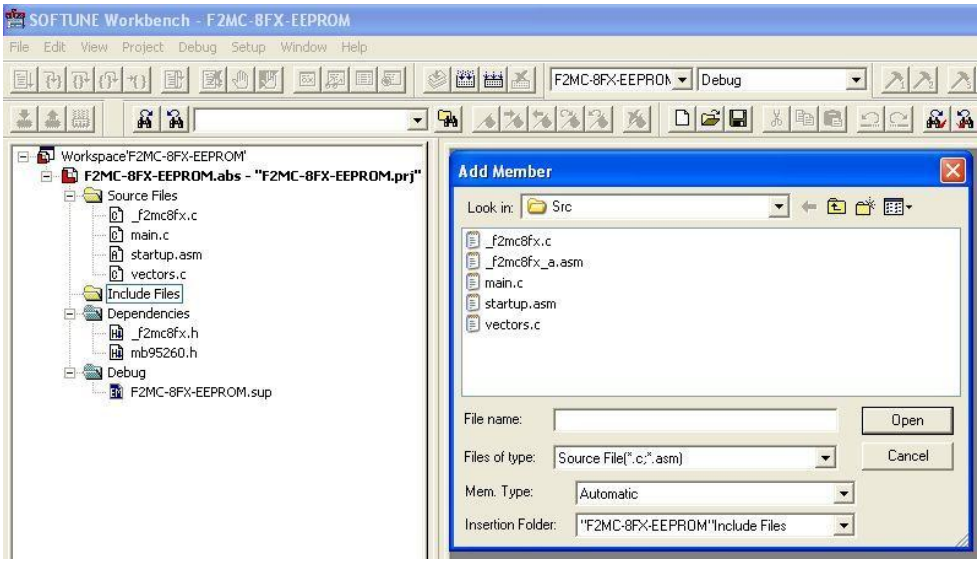
In SOFTUNE, right click **Include Files**, and select **Add Member to Folder→ File** from the shortcut menu.

Figure 7. Select File



In this case, The F²MC-8FX-EEPROM.lib cannot be found.

Figure 8. Open Add Member Window



In **Add Member** window, select **All Files** from the **Files of type**, then you will find the F²MC-8FX-EEPROM.lib.

Figure 9. Select File of Type

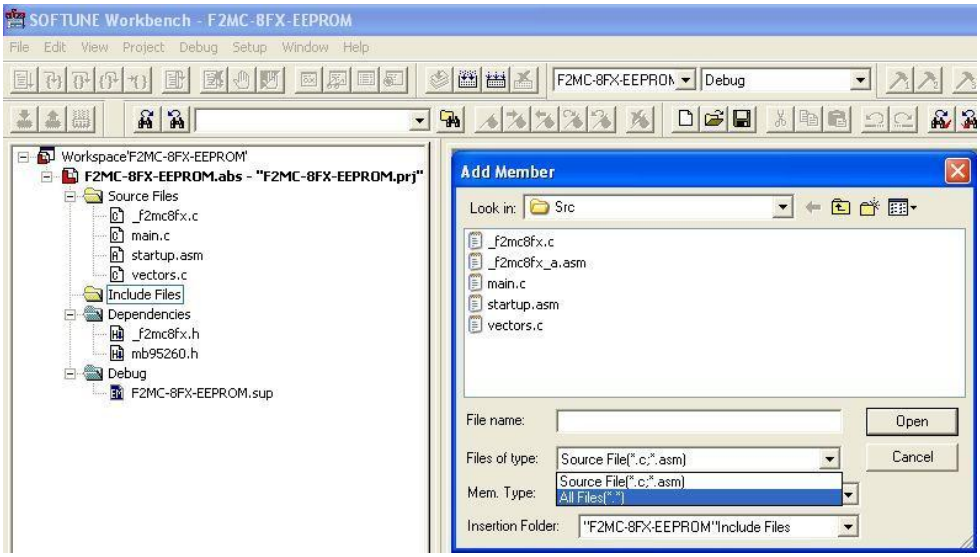
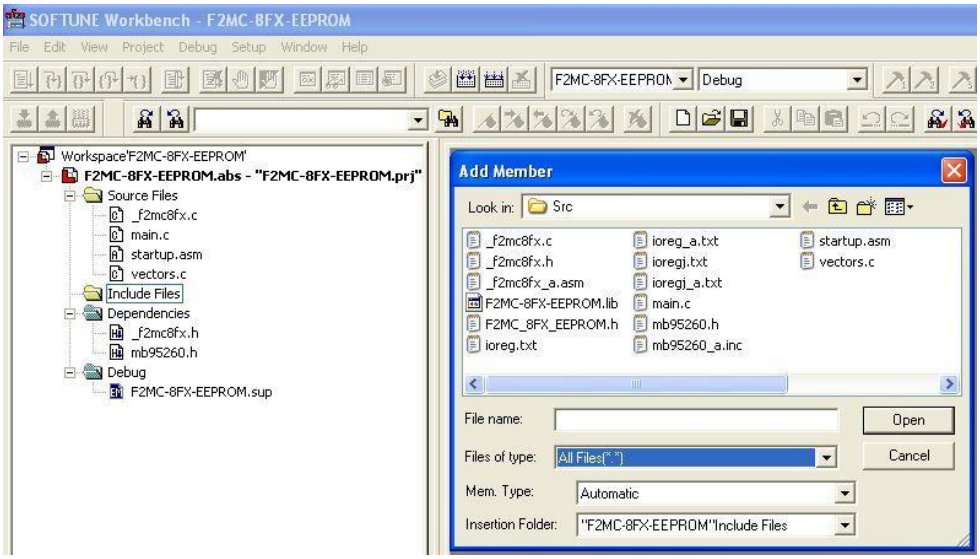
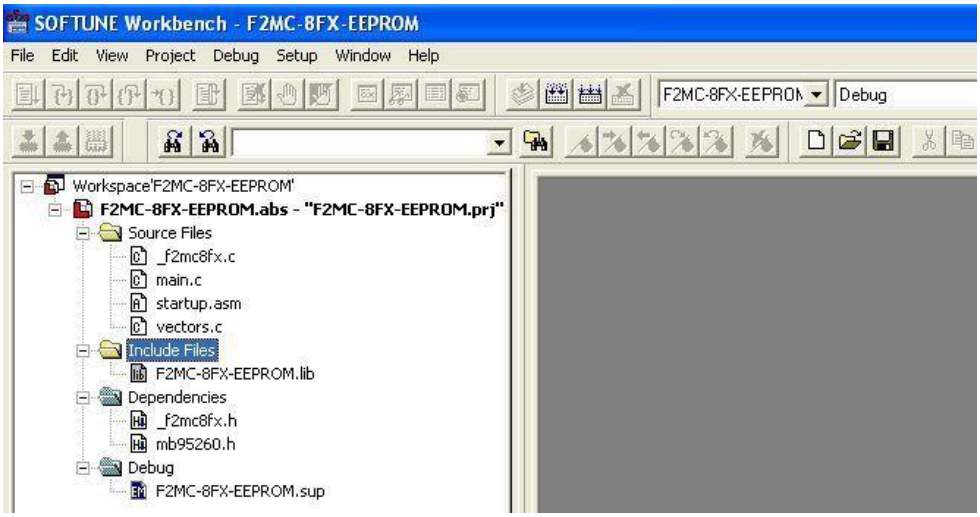


Figure 10. Show F²MC-8FX-EEPROM.lib



Double click the **F²MC-8FX-EEPROM.lib** and it will be added to Source Files.

Figure 11. Add F²MC-8FX-EEPROM.lib



5.4 Add #include "F²MC_8FX_EEPROM.h" in "main.c"

Add #include "F²MC_8FX_EEPROM.h" in "main.c ". Compiling "F²MC_8FX_EEPROM.h" will link F²MC-8FX-EEPROM.lib to main.c, and then, user program can use API provided by F²MC-8FX-EEPROM.lib.

Figure 12. Add #include "F²MC_8FX_EEPROM.h" in "main.c"

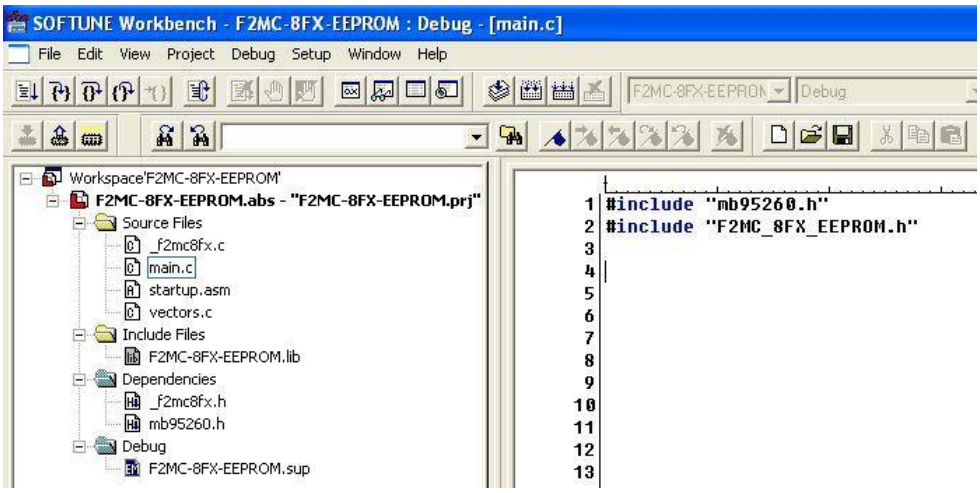
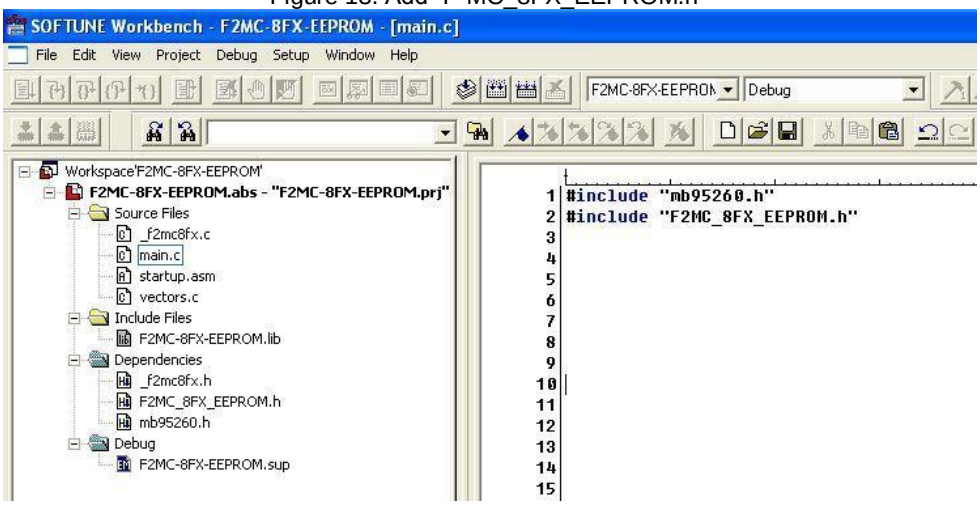


Figure 13. Add "F²MC_8FX_EEPROM.h"



5.5 Flash Interrupt Setting in “vector.c”

To improve performance of EEPROM library, flash erase interrupt is used in program. Please refer below sample code to setting flash interrupt level and vector.

```
…
/*-----------------------------------------------------------------------
----------------------
   Interrupt request level setting
-----------------------------------------------------------------------
----------------------*/
void InitIrqLevels(void)
{
      …
      ILR5 = 0x3F;  // IRQ20: Watch timer / counter
                    // IRQ21: external interrupt ch 8-11
                    // IRQ22: 8/16-bit timer ch1 (lower) | external
            interrupt ch 12-15
                    // IRQ23: Flash | Custom ch1
      …
}
/*-----------------------------------------------------------------------
----------------------
   Prototypes
-----------------------------------------------------------------------
----------------------*/
…
__interrupt void FlashIntHandle(void);
…
/*-----------------------------------------------------------------------
----------------------
   Vector definiton
-----------------------------------------------------------------------
----------------------*/
…
#pragma intvect FlashIntHandle 23  //  IRQ23: Flash | Custom ch1
…
```

6 Limitations of EEPROM Library Usage

6.1 EEPROM Data Area Updating and Monitor in SOFTUNE

The EEPROM data area can be displayed in memory window on SOFTUNE. But users should not modify this area in memory window and the data in this area is also hardly to understand. So using the batch file to read the EEPROM data is recommended. The batch file can detect the EEPROM data and display them directly in command window.

6.1.1 Limitation of Modify the EEPROM Area in Memory Window

If the EEPROM data is modified on memory window on SOFTUNE, SOFTUNE will store the modified data in its PC buffer and also refresh the flash memory data before next execution. Then the data will be written to the EEPROM emulation area. It is may cause the fatal error in EEPROM Library.

So user can not modify the EEPROM data in memory window of SOFTUNE

6.1.2 Limitation of Monitor the EEPROM Data

The EEPROM area has its own structure and it is hard for user to understand the EEPROM data through watching this area in memory window. But the batch file can read it and display the EEPROM data in command window.

So the only way to monitor the EEPROM data is using the provided batch file.

Note:

If set MCU type to dual flash production MCU only, SOFTUNE will set the EEPROM area to ROM attribution. So if user modifies the EEPROM area in memory window, the flash is also modified before next execution operation, it is forbidden. But in this case, SOFTUNE also doesn't upload the EEPROM area onto PC buffer. So the batch file can not read out the EEPROM data correctly because of the batch file can only read the data from PC buffer if the read area's attribution is ROM.

If set MCU type to single flash MCU (200/210 series), SOFTUNE will set the EEPROM area to RAM attribution. SOFTUNE will refresh the RAM area at any time if needed. In this case, the batch file read out the EEOROM data correctly at any time because of the batch file can read the data from MCU directly. And then modify the EEPROM area in memory window cannot change this area because SOFTUNE sends RAM write command, this command cannot write any data into the EEPROM emulation area.

The batch file will be developed to set the attribution of the EEPROM area when starting debug.

6.2 Limitation of Call API by Using "CALL" Command in SOFTUNE

NMI or Break interrupt may occur during flash write/erase operation in debug mode, and then if API is called by using "CALL (write)" command, the flash write operation may not work correctly. (Write cannot be performed or error data is written.). So "CALL" command is forbidden in SOFTUNE if using the EEPROM library.

6.3 Limitation of Use the Sector Swap Function

If customer enable sector swap by their own program, SOFTUNE can no know it. And this operation may break the interrupt vector and the EEPROM data and may cause the system break down in serious case, so the sector swap operation is forbidden.

6.4 Limitation of Flash Interrupt

The Flash interrupt is used by the library, so it cannot be used by user.

6.5 Limitation of MCU Mode Change

Flash operation cannot be performed normally while MCU working at sub-clock, user should check if flash erase/programming operation is completed before change the mode of MCU from main clock to sub clock.

6.6 Limitation of Watchdog Timer

According to the API execute time may exceed the minimal WDT interval time, if WDT enables, clear WDT timer before and after calling API is recommended.

6.7 Limitation of Variables Definition

There are total 20 bytes RAMs are used for the library in direct access area*[1]; the variables which are defined by user code should not exceed this area (0x90 ~ 0xFF, 112 bytes in total).

The 20 bytes for library use are to improving performance. If user code defines too much direct address to out of range, the program may enter infinite status.

To avoid this unexpected problem, less than 92 (112-20) direct address byte definitions are recommended.

*1 unsigned char __direct var1; // var1 can be direct accessed by MCU

User may define direct address using above method, but the total number (library and user code) should not beyond 112.

7 Additional Information

For more information on Cypress Microcontrollers Products, please visit the following websites:

http://www.cypress.com/cypress-microcontrollers

http://www.cypress.com/cypress-mcu-product-softwareexamples

Document History

Document Title: AN205392 – F²MC - 8FX Family, New 8FX Series, EEPROM Library

Document Number: 002-05392

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	Jacky Zhou Levi Zhang	03/30/2008	Create
		Jacob Li	03/31/2009	Add limitation in Chapter 6
		Jacky Zhou	05/24/2009	Update basing on Ext Spec MCU-AN- 500019-E-11 → MCU-AN- 500019-E-20 All EEPROM Lib → F2MC-8FX-EEPROM All pictures Description in Chapter 3, 4, 5
		Levi Zhang	09/25/2009	Update basing on external spec Add recommended usage and status handling
			10/21/2009	Add limitation of variables definition
			12/01/2009	Correct ROM usage description in Section 2.5
			12/03/2009	Add flash interrupt service routine setting description in Chapter 5
			03/05/2010	2V6 Add Type A&B library description in Section 2.3
			08/22/2011	2V7 Correct MB95F390 sector information in section 2.3
			09/09/2011	2V8 Update some unclear descriptions.
			06/19/2012	2V9 Add Type C, D, E description in section 2.3
			02/28/2013	3V0 Revise typo in section 2.3
*A	5265877	HUAL	05/10/2016	Migrated Spansion Application Note “MCU-AN-500019-E-30” to Cypress format.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Lighting & Power Control	cypress.com/powerpsoc
Memory	cypress.com/memory
PSoC	cypress.com/psoc
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless/RF	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP

Cypress Developer Community

Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

PSoC is a registered trademark and PSoC Creator is a trademark of Cypress Semiconductor Corporation. All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com

© Cypress Semiconductor Corporation,2009-2016. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.