AN205556

F²MC-16FX Family, Flash Memory

This application note describes the functionality and operation of the Flash memory and gives some software examples.

Contents

1 Introduction....1
1.1 Key Features....1
2 Flash Memory....2
2.1 Outline....2
2.2 Sector Configuration of Main Flash memory ...2
2.3 Sector configuration of Data Flash memory 3
2.4 Registers....3
3 Flash Write and Erase Access....10
3.1 Flash Access....10
3.2 Write Access Command Sequence....11
3.3 Sector Erase Command Sequence....11
3.4 Chip Erase Command Sequence....12
3.5 Sector Erase Suspend Command....12
3.6 Sector Erase Resume Command....12
3.7 Read/Reset Command and Command Sequence....13
3.8 Fast Programming Mode Command Sequences....13
3.9 Hardware Sequence Flags....13
3.10 Notes on using Sequence Flags 16
4 Flash Memory Examples....16
4.1 Flash RAM Routines 16
4.2 Flash Memory Write 17
4.3 Flash Sector Erase....19
4.4 Flash Read Access Timing Configuration 21
4.5 Manual RAM Copy 22
4.6 Writing by Command Sequencer 24
5 Additional Information....24
6 Document History....25

1 Introduction

This application note describes the functionality and operation of the Flash memory and gives some software examples.

1.1 Key Features

- Non-volatile reprogrammable Memory
- 10000 Program and Erase Cycles
- Up to 20 years Data Retention Time
- Flash Security (prevents Read-Out)
- Sector-oriented Memory (64K Bytes / 16K Bytes / 8K Bytes)
- Flash Memory B (Dual Bank, Device dependent)
- Data Flash Memory A, B (EEPROM emulation, Device dependent)
- Flash Timing (Wait Cycles) selectable
- Sector Write/Erase Protection selectable during runtime
- Fast Mode programming function

2 Flash Memory

The Basic Functionality of the Flash Memory

2.1 Outline

The Flash memory contains the user application and data. It can be programmed word-wise and erased per sector or entire chip.

The Flash memory can be programmed via CPU mode (during run time or serial programming mode) or via parallel programming (Main Flash only). There are up to four Flash memories available on-chip: Main Flash A, Main Flash B, Data Flash A, and Data Flash B. During programming or erasing, the same Flash memory cannot be read out. That means if Main Flash A is getting erased/programmed then only Main Flash B, Data Flash A, or Data Flash B can be read out and vice a versa. Hence the erase or program routine has to be available in the memory other than the memory which is erased or programmed. Otherwise the programming in CPU mode has to be done from RAM routines.

An internal Embedded Algorithm[1] is built in, which takes care of the correct programming or erasing sequences. The user program has only to check, if the operation has finished successfully, failed or timed out.

After erase operation, all bit cells of the corresponding sector contain a logical “1”. These cells can be programmed to “0”. Reprogramming to “1” is not possible, but a new sector erase does this. A minimum of 10,000 programming/erase cycles are guaranteed.

A sector-wise programming/erasing protection can be set in CPU mode.

For detailed information about Flash Security please refer to application note: *mcu-an-300213-e-16fx_flash_security*.

2.2 Sector Configuration of Main Flash memory

The following table shows the general sector configuration for the Main Flash memory:

Address	General Sector Name and Size	Flash Memory Type
0xFFFFFF 0xFF0000	S39 – 64 KByte	Main Flash 64 KByte-Sectors
0xFEFFFF 0xFE0000	S38 – 64 KByte	
. . .	. . .	
0xE1FFFF 0xE10000	S9 – 64 KByte	
0xE0FFFF 0xE00000	S8 – 64 KByte	
0xDF7FFF 0xDF6000	SA3 – 8 KByte	Main Flash A 8 KByte-Sectors
0xDF5FFF 0xDF4000	SA2 – 8 KByte	
0xDF3FFF 0xDF2000	SA1 – 8 KByte	
0xDF1FFF 0xDF0000	SA0 – 8 KByte	

[1] Embedded Algorithm is a trademark of Advanced Micro Devices, Inc.

Address	General Sector Name and Size	Flash Memory Type
0xDE7FFF 0xDE6000	SB3 – 8 KByte	Main Flash B 8 KByte-Sectors
0xDE5FFF 0xDE4000	SB2 – 8 KByte	
0xDE3FFF 0xDE2000	SB1 – 8 KByte	
0xDE1FFF 0xDE0000	SB0 – 8 KByte	

Please note, that the address space between 0xDF0000 and 0xDF007F (beginning of SA0) and 0xDE0000 and 0xDE002F (beginning of SB0) is used for Flash memory and Boot-ROM settings. These address spaces are called ROM Configuration Blocks.

2.3 Sector configuration of Data Flash memory

The following table shows the general sector configuration for the Data Flash memory:

Address	General Sector Name and Size	Flash Memory Type
0x0EFFFF 0x0EFF00	SDA0 – 256 Byte	Data Flash A Configuration block
0x0EFEFF 0x0EFE00	SDB1 – 256 Byte	Data Flash B Configuration block
0x0EFDFF 0x0E0000	Reserved	
0x0DFFFF 0x0DC000	SDA4 – 16 KByte	Data Flash A 16 KByte-Sectors
0x0DBFFF 0x0D8000	SDA3 – 16 KByte	
0x0D7FFF 0x0D4000	SDA2 – 16 KByte	
0x0D3FFF 0x0D0000	SDA1 – 16 KByte	
0x0CFFFF 0x0CC000	SDB4 – 16 KByte	Data Flash B 16 KByte-Sectors
0x0CBFFF 0x0C8000	SDB3 – 16 KByte	
0x0C7FFF 0x0C4000	SDB2 – 16 KByte	
0x0C3FFF 0x0C0000	SDB1 – 16 KByte	

2.4 Registers

2.4.1 Hardware Sequence Flags

These flags indicate the state of the Flash memory. They are not located in a special register, but in the Flash memory itself. Depending on the operation state of the Flash memory each address will show either these flags or the memory content. During write or erase operation all even addresses will show these flags, only during suspend state the sectors not being erased will show their memory contents. After any successful write or erase operation all addresses will return to show their memory contents.

Note that the recognition of toggling bits does not depend on any timing restrictions, but on the read sequence only.

Bit No.	Name	Explanation	Operation	Flag Status
7	DQ7	Data Polling Flag	Write	DQ7x[2]
			Chip/Sector Erase	0
			Write → Completed	DQ7x[2] → Data[7]
			Chip/Sector Erase → Completed	0 → 1
			Sector Erase Wait → Started	1 → 0
			Sector Erase → Suspend	0 → 1
			Sector Erase Suspend → Restarted	1 → 0
			Other Sectors during Erase Suspended	Data[7]
6	DQ6	Toggle Bit Flag	Write	Toggle
			Chip/Sector Erase	Toggle
			Write → Completed	Toggle → Data[6]
			Chip/Sector Erase → Completed	Toggle → 1
			Sector Erase Wait → Started	Toggle
			Sector Erase → Suspend	Toggle → 1
			Sector Erase Suspend → Restarted	1 → Toggle
			Other Sectors during Erase Suspended	Data[6]
5	DQ5	Timing Limit Exceeded Flag	Write	0
			Chip/Sector Erase	0
			Write → Completed	0 → Data[5]
			Chip/Sector Erase → Completed	0 → 1
			Sector Erase Wait → Started	0
			Sector Erase → Suspend	0
			Sector Erase Suspend → Restarted	0
			Other Sectors during Erase Suspended	Data[5]
4	–	-	-	-
3	DQ3	Sector Erase Timer Flag	Write	0
			Chip/Sector Erase	1
			Write → Completed	0 → Data[3]
			Chip/Sector Erase → Completed	1
			Sector Erase Wait → Started	0 → 1
			Sector Erase → Suspend	1 → 0
			Sector Erase Suspend → Restarted	0 → 1
			Other Sectors during Erase Suspended	Data[3]
2	DQ2	Toggle Bit-2 Flag	Write	1
			Chip/Sector Erase	Toggle
			Write → Completed	1 → Data[2]
			Chip/Sector Erase → Completed	Toggle → 1
			Sector Erase Wait → Started	Toggle
			Sector Erase → Suspend	Toggle
			Sector Erase Suspend → Restarted	Toggle
			Other Sectors during Erase Suspended	Data[2]
1	–	-	-	–
0	–	-	-	–

[2] DQ7x is the inverted data bit #7 of the data to be written during Flash Write.

Operation state	DQ7	DQ6	DQ5	DQ3	DQ2
Idle	D7[3]	D6[3]	D5[3]	D3[3]	D2[3]
Erase Wait	1	Toggle	0	0	Toggle
Erase	0	Toggle	0	1	Toggle
Suspended	1	1	0	0	Toggle
Write	DQ7x[2]	Toggle	0	0	1
Write Error	DQ7x[2]	Toggle	1	0	1
Erase Error	0	Toggle	1	1	0 or 1

Please see chapter 3.9 for detailed sequences.

2.4.2 Control Status Register Main Flash (MCSRA, MCSRB)

Bit No.	Name	Explanation	Initial Value	Value	Operation
7	–	-	X	–	Reserved; always write “0”
6	RD19V	1.9V Operation Mode Select bit	0	0	1.8V Operation Mode Selected
				1	1.9V Operation Mode Selected
5	DRBE	Data Read Buffer Enable Bit	1	0	Disable data read buffer
				1	Enable data read buffer
4	CRBE	Code Read Buffer Enable Bit	1	0	Disable code read buffer
				1	Enable code read buffer
3	WE	Write Enable Bit[4]	0	0	Disable Flash write
				1	Enable Flash write
2	INTE	Interrupt Enable Bit	0	0	Disable interrupt function
				1	Enable interrupt function
1	RDYINT	RDY Interrupt Flag	0	0	No write/erase command termination
				1	Write/erase command has been terminated
0	RDY	Flash RDY Status	X	0	Write/erase ongoing
				1	Flash ready

[3] D7, D6, D5, D3 and D2 are the corresponding bits of the value stored at that address.

[4] Setting this bit to 1 does not disable data read nor does it explicitly enable the sequence flags. Setting this bit to 0 simply inhibits Flash from recognizing command sequences.

2.4.3 Memory Timing Configuration Register Main Flash (MTCRA, MTCRB)

The Memory Timing Configuration Register configures the wait states for Flash memory access as well as the setup of the internal control signals. The settings below have been evaluated for proper function. Only the wait state setting is relevant for application performance. Please select a setting that corresponds to the selected clock setting and core voltage. Note that some settings require a minimum frequency of CLKS1 and/or a certain Bus Clock Division.

MTCRA, MTCRB	Permitted frequency range for CLKS1		Permitted CLKB divider setting	Resulting CLKB frequency range		Purpose
	1.8V operation	1.9V operation		1.8V operation	1.9V operation	
6E3Dh	$f_{CLKS1} \le 100$ MHz		1 … 16	$f_{CLKB} \le 100$ MHz		Read with 5 wait states
6E3Fh						Read and Write with 7 wait states
0239h (init value)	$f_{CLKS1} \le 25$ MHz		1 … 16	$f_{CLKB} \le 25$ MHz		Read with 1 wait state
2129h	$f_{CLKS1} \le 30$ MHz	$f_{CLKS1} \le 32$ MHz	1 … 16	$f_{CLKB} \le 30$ MHz	$f_{CLKB} \le 32$ MHz	Read with 1 wait state
233Ah	$f_{CLKS1} \le 50$ MHz	$f_{CLKS1} \le 56$ MHz	1 … 16	$f_{CLKB} \le 50$ MHz	$f_{CLKB} \le 56$ MHz	Read with 2 wait states
	$f_{CLKS1} \le 20$ MHz			$f_{CLKB} \le 20$ MHz		Read and Write with 2 wait states
4B3Bh	$f_{CLKS1} \le 60$ MHz	$f_{CLKS1} \le 64$ MHz	1 … 16	$f_{CLKB} \le 60$ MHz	$f_{CLKB} \le 64$ MHz	Read with 3 wait states
4B3Dh						Read and Write with 5 wait states
2128h	$f_{CLKS1} \le 30$ MHz	$f_{CLKS1} \le 32$ MHz	2 … 16	$f_{CLKB} \le 15$ MHz	$f_{CLKB} \le 16$ MHz	Read with 0 wait states
2208h	20 MHz ≤ $f_{CLKS1} \le 50$ MHz	20 MHz ≤ $f_{CLKS1} \le 56$ MHz	2	10 MHz ≤ $f_{CLKB} \le 25$ MHz	10 MHz ≤ $f_{CLKB} \le 28$ MHz	Read with 0 wait states
4C09h	$f_{CLKS1} \le 76$ MHz	$f_{CLKS1} \le 84$ MHz	2	$f_{CLKB} \le 38$ MHz	$f_{CLKB} \le 42$ MHz	Read with 1 wait state
6B09h	20 MHz ≤ $f_{CLKS1} \le 92$ MHz	20 MHz ≤ $f_{CLKS1} \le 100$ MHz	2	10 MHz ≤ $f_{CLKB} \le 46$ MHz	10 MHz ≤ $f_{CLKB} \le 50$ MHz	Read with 1 wait state
2279h	$f_{CLKS1} \le 40$ MHz		2 … 16	$f_{CLKB} \le 20$ MHz		Read and Write with 1 wait state

A correct setting for application execution is commonly selected in the start.asm file. The setting is selected implicitly by selection of the clock setup. This selection does not consider writing into Flash memory. Hence, it is necessary to select a Flash memory timing setting that allows writing. The setting that can always be used is the 0x6E3F. This selects 7 wait states. Although this seems to be a quite slow setting, please consider that it needs to be used only for the actual erasing or programming algorithm. The respective function is usually executed either from RAM, which always works without wait states, or a different Flash memory with separate setting. Before returning to execute code from Flash memory, the initial Flash memory timing configuration should be restored.

2.4.4 Write Control Registers 0-5 (FMWC0–5)

These 8 bit registers control the write/erase access to the Flash memory sectors.

FMWC0 controls the sectors of the Main Flash B, where Bit 7 (WCB7) corresponds to sector SB7 and Bit 0 (WCB0) to SB0.

FMWC1 controls the sectors of the Main Flash A, where FMWC1_WCA0 controls sector SA0 till FMWC1_WCA7, which controls sector SA7.

FMWC2-5 control the sectors of the Main Flash A or B (depending on available memory), where FMWC2_WC8 controls sector S8 till FMWC5_WC39, which controls the highest addressed sector S39.

Writing “0” to these bits disables the corresponding sector write/erase, writing “1” enables it.

If the used device does not provide all sectors, please write “0” to not provided sector bits.

Please note, if a sector was protected by writing “0”, it can be enabled again for write/erase only after a reset. Till then it is not possible to write a value of “1”.

Also note that the initial value is “0”, so that for the first erase/write access the corresponding sector has to be write-enabled.

There is no write protection for Data Flash memory.

2.4.5 Write Protection Activation Markers (FWPAMA0/1, FWPAMB0/1)

The FMWCn registers (2.4.4) can be set automatically by the Flash Write Protection Sector Markers after reset, if the Activation Markers (FWPAMA1-0 or FWPAMB1-0) contain the ‘magic word’ 0x292D3A7B. For the Main Flash Memory A the markers are FWPAMA0/1 and for the Main Flash Memory B the markers are FWPAMB0/1, if available.

These markers are located in the ROM Configuration Block as follows:

Address	Marker	Memory
0xDE001C	FWPAMB0	Main Flash B
0xDE001E	FWPAMB1	
0xDF001C	FWPAMA0	Main Flash A
0xDF001E	FWPAMA1	

There is no write protection for the Data Flash memory.

2.4.6 Write Protection Sector Markers (FWPSMA0-4, FWPSMB0-4)

These markers are located in the ROM configuration blocks. If the magic word is located at the Flash Memory Write Protection Activation Markers (2.4.5), their contents are copied to the FMWCn registers after reset. There are five Markers for the Main Flash Memory A (FWPSMA0-4) and five for the Main Flash Memory B (FWPSMB0-4), if provided.

The organization is similar to the FMWCn registers. Bit 0 of the FWPSMA0 correlates to sector SA0 and bit 7 of the FWPSMA4 or FWPSMB4 to sector S39. Writing “0” to these bits disables the corresponding sector write/erase, writing “1” enables it.

These markers are located in the ROM Configuration Block as follows:

Address	Marker	Memory
0xDE0020	FWPSMB0	Main Flash B Sector 0 to 7
0xDE0021	FWPSMB1	Main Flash B Sector 8 to 15
0xDE0022	FWPSMB2	Main Flash B Sector 16 to 23
0xDE0023	FWPSMB3	Main Flash B Sector 24 to 31
0xDE0024	FWPSMB4	Main Flash B Sector 32 to 39
0xDF0020	FWPSMA0	Main Flash A Sector 0 to 7
0xDF0021	FWPSMA1	Main Flash A Sector 8 to 15
0xDF0022	FWPSMA2	Main Flash A Sector 16 to 23
0xDF0023	FWPSMA3	Main Flash A Sector 24 to 31
0xDF0024	FWPSMA4	Main Flash A Sector 32 to 39

2.4.7 Data Flash Control Status register (DFCSA, DFCSB)

Bit No.	Name	Explanation	Initial Value	Value	Operation
7	–	-	X	–	Reserved; always write “0”
6	TMG2	Timing configuration bits	1	See table below	
5	TMG1		1		
4	TMG0		1		
3	–	-	X	–	Reserved; always write “0”
2	INTE	Interrupt Enable Bit	0	0	Disable interrupt function
				1	Enable interrupt function
1	RDYINT	RDY Interrupt Flag	0	0	No write/erase command termination
				1	Write/erase command has been terminated
0	RDY	Flash RDY Status	X	0	Write/erase ongoing
				1	Flash ready

TMG2	TMG1	TMG0	Maximum CLKB frequency	CLKB wait cycles	
				Read	Write
0	0	0	7.5 MHz	2	2
0	0	1	16 MHz	3	2
0	1	0	33 MHz	4	2
0	1	1	50 MHz	5	2
1	0	0	66 MHz	7	3
1	0	1	83 MHz	8	4
1	1	0	100 MHz	9	5
1	1	1	100 MHz	10	5

The Data Flash Control Status register has 2 purposes. First, it controls the access timing. Secondly, it provides the Ready status and according interrupt function during erasing and writing to Flash memory.

The default timing is selected such that it works for all operating conditions of the MCU. It is safe to leave this value unchanged. In case maximum performance is desired, the setting may be changed to match the maximum CLKB (CPU) frequency used in the application.

2.4.8 Data Flash Write command sequencer Control register (DFWCA, DFWCB)

Bit No.	Name	Explanation	Initial Value	Value	Operation
15	–	-	X	–	Reserved; always write “0”
14	–	-	X	–	Reserved; always write “0”
13	–	-	X	–	Reserved; always write “0”
12	ERINTE	Command Sequencer Error Interrupt Enable	0	0	Disable the interrupt of the error flag ERINT
				1	Enable the interrupt of the error flag ERINT
11	FININTE	Command Sequencer Finish Interrupt Enable	0	0	Disable the interrupt of the finish flag FININT
				1	Enable the interrupt of the finish flag FININT

Bit No.	Name	Explanation	Initial Value	Value	Operation
10	IDLINTE	Command Sequencer Idle Interrupt Enable	0	0	Disable the interrupt of the idle flag IDLINT
				1	Enable the interrupt of the idle flag IDLINT
9	-	-	X	-	Reserved; always write “0”
8	WE	Write Enable	0	0	Command Sequencer is disabled
				1	Command Sequencer is enabled

The Data Flash Write command sequencer Control register provide function to enable status interrupts of the command sequencer. It also serves to enable the command sequencer.

2.4.9 Data Flash Write command sequencer Status register (DFWSA, DFWSB)

Bit No.	Name	Explanation	Initial Value	Value	Operation
7	PAERF	Prohibited Access Error flag	0	0	No prohibited access detected
				1	CPU tried to access Flash during command sequencer programming (prohibited access)
6	WIERINT	Command Sequencer Write Incomplete Error Interrupt flag	0	0	Command sequencer was not disabled during writing
				1	Command sequencer was disabled during writing
5	WERINT	Command Sequencer Write Error Interrupt flag	0	0	No write error detected
				1	A write operation ended with incorrect data
4	TOERINT	Command Sequencer Timeout Error Interrupt flag	0	0	No timeout error was detected
				1	A write operation ended with timeout error
3	FININT	Command Sequencer Finish Interrupt flag	0	0	Write command was not (yet) finished successfully
				1	Write command was finished successfully
2	IDLINT	Command Sequencer Idle Interrupt flag	0	0	Command sequencer is disabled or not in Idle state
				1	Command sequencer entered idle state
1	ST1	Command sequencer status	0	See table below	
0	ST0		0		

ST1	ST0	Write command sequencer status
0	0	Idle (sequencer accepts new write command)
0	1	Sequencer is submitting a write command
1	0	Sequencer is waiting for Flash program finish
1	1	Illegal State

The Data Flash Write command sequencer Status register contains the interrupt flags for command sequencer operation. In addition, status flags are provided for polling operation. In case the command sequencer status flags report constantly “illegal State”, a reset must be issued.

3 Flash Write and Erase Access

How to Write and Erase the Flash Memory

3.1 Flash Access

The Main Flash memory contents cannot be changed by a simple write instruction. The access is done via a command sequence, which consists of several write accesses with certain data to certain addresses. Afterwards the Automatic Algorithm of the Flash memory is started. The user has then to wait until the command has finished or timed out. For this the user has to poll special hardware sequence bits, which are described in 2.4.1 or use interrupts.

The Data Flash memory can be programmed in a similar way as the Main Flash memory. In addition, the Data Flash memory can be programmed using the Command Sequencer. This allows to program data without generating the command sequence in software. Instead, the sequence is generated in hardware by the Command Sequencer.

In general Flash bit cells can be programmed to “0” individually. An individual reprogramming to “1” is not possible. To do this, a whole sector has to be erased. After this, all bit cells of the sector contain a “1”.

Please check or set the corresponding Flash Memory Write Control Bits (2.4.4) before accessing to the Flash memory.

Note, that `sss` in the following examples stands for the upper 3 nibbles of either the Flash memory address that is written to or the Flash memory sector base address that is being erased or suspended.

3.1.1 Important Note for High Speed Applications

Writing to the Main Flash memory area is allowed under the following conditions:

FMTCH/L	Permitted Frequency Range for CLKS1		Permitted CLKB Divider Setting	Resulting CLKB Frequency Range		Access Type
	1.8V Operation	1.9V Operation		1.8V Operation	1.9V Operation	
0x6E3F	CLKS1 ≤ 100 MHz		1 … 16	CLKB ≤ 100 MHz		Synchronous Read/Write, 7 Wait States
0x233A	CLKS1 ≤ 20 MHz		1 … 16	CLKB ≤ 20 MHz		Synchronous Read/Write, 2 Wait States
0x4B3D	CLKS1 ≤ 60 MHz	CLKS1 ≤ 64 MHz	1 … 16	CLKB ≤ 60 MHz	CLKS1 ≤ 64 MHz	Synchronous Read/Write, 5 Wait States
0x2279	CLKS1 ≤ 40 MHz		2 … 16	CLKB ≤ 20 MHz		Synchronous Read/Write, 1 Wait State
0x0231	CLKS1 ≤ 7 MHz		1 … 16	CLKB ≤ 7 MHz		Asynchronous Read/Write, 1 Wait State

The Data Flash memory can be written with any timing configuration that fits the `CLKB`.

Reading from the Main Flash memory is faster than writing. Write or erase is possible under the above discussed conditions and the corresponding values of timing configuration registers only (where as read is possible in all the conditions discussed in section 2.4.3). Hence before writing or erasing, one needs to make sure that the corresponding timing configurations are done depending upon the CLKS1 and CLKB frequency. Depending upon the timing configurations the wait states while write or erase will vary between 1 and 7.

Example:

An application runs with high speed (CLKS1 = 96 MHz), then a Main Flash memory Write access has to be done as follows:

```
unsigned int MTCRA_save;
       . . .

MTCRA_save = MTCRA; // Temporary save Flash access settings
MTCRA = 0x6E3F;     // Set MTCR temporary to sync. R/W, 7 Wait States

       . . .        // Flash Sequence here

MTCRA = MTCRA_save; // Restore Flash access settings

       . . .        // Further Code (e.g. Data Polling, etc.)
```

To write to the Main Flash memory for the sequences (Write, Erase, etc.) temporary set Flash Memory access to 7 wait states. After writing the old setting can be restored. No further clock or PLL setting has to be done for this.

This mechanism ensures safe Main Flash memory access with almost no performance lost regarding the application speed.

3.2 Write Access Command Sequence

The following sequence has to be used to write to the Main Flash memory. It needs to be done 16-bit-wise.

```
sssAAA              := 0xXXAA
sss554              := 0xXX55
sssAAA              := 0xXXA0
Write_Address_Even  := Write_word
```

The following sequence has to be used to write to the Data Flash memory. It needs to be done 8-bit-wise.

```
sssAAA              := 0xXXAA
sss554              := 0xXX55
sssAAA              := 0xXXA0
Write_Address       := Write_char
```

3.3 Sector Erase Command Sequence

The following sequence has to be used to erase a sector:

```
sssAAA              := 0xXXAA
sss554              := 0xXX55
sssAAA              := 0xXX80
sssAAA              := 0xXXAA
sss554              := 0xXX55
Sector_Address_Even := 0xXX30
```

Important Note:

The DQ6 and DQ5 polling algorithms have to be started after the sector erase timer time of 50 µs. This wait time can be polled by DQ3 flag. Before this time the contents of DQ6 and DQ5 may be undefined.

Alternatively use the toggle bit algorithm (DQ2) or poll the RDYINT flag after writing the erase sequence.

Example for DQ3 polling:

```
. . .   // Erase Sequence

while ((*Sector_Address_Even & DQ3) != DQ3); //sector erase timer ready?

   . . .   // Data polling or Toggle Bit polling
```

3.4 Chip Erase Command Sequence

The following sequence has to be used to erase all sectors at once.

```
sssAAA := 0xXXAA
sss554 := 0xXX55
sssAAA := 0xXX80
sssAAA := 0xXXAA
sss554 := 0xXX55
sssAAA := 0xXX10
```

Please note that chip erase does either erase the Main Flash memory A or B or Data Flash memory A or B depending upon the sector address (sss) chosen. Erasing more than one memory with just one command sequence is not possible.

3.5 Sector Erase Suspend Command

It is allowed to suspend a sector erase temporally to write to a different sector or to set the Flash memory back to read mode. In this case only those sectors will return to read mode that were not target to the ongoing erase operation. The other sectors will continue to be in flag mode. The erase command can be later resumed, so that the user does not need to restart the sector erase. The sum of such discrete erase times (erase + suspend + resume) is approximately the time of a not suspended erase.

```
sssXXX := 0xXXB0
```

Please note that suspend command shall not be stated before 1.2 ms of erasing time. After setting the resume command Flash read access is possible after about 50 µs (sector erase suspend). Both conditions can be polled by reading DQ3 or DQ6 and DQ7.

Also, use the sector erase suspend command only up to 10 times during an ongoing erase operation of a sector.

This command is only available, if a sector erase command was set before. Otherwise it is ignored.

3.6 Sector Erase Resume Command

With the following command a suspended sector erase is resumed. Please note that a further suspend command should not be set within 1.2 ms. Poll DQ3 for sector erase wait for this wait time.

```
sssXXX := 0xXX30
```

This command is only available, if a sector erase suspend command was set before. Otherwise it is ignored.

3.7 Read/Reset Command and Command Sequence

The following command and command sequence resets the Flash algorithm and allows the Flash read access. Both command sequences are equivalent. The 3 cycle sequence is available for compatibility with old JEDEC specification.

```
sssXXX := 0xXXF0
```

```
sssAAA := 0xXXAA
sss554 := 0xXX55
sssAAA := 0xXXF0
```

3.8 Fast Programming Mode Command Sequences

For Flash write it is possible to shorten the write sequence described in 3.2 by setting Fast Mode. After programming, this mode has to be reset. No other command but Flash fast writing is allowed in this mode.

3.8.1 Set to Flash Fast Mode Command Sequence

The following command sequence set the Flash Fast Mode:

```
sssAAA := 0xXXAA
sss554 := 0xXX55
sssAAA := 0xXX20
```

3.8.2 Fast Write Command Sequence

The following command sequence has to be used for Main Flash writing. Note, that this sequence is only available in Fast Mode.

```
sssXXX              := 0xXXA0
Write_Address_Even  := Write_word
```

The following command sequence has to be used for Data Flash writing. Note, that this sequence is only available in Fast Mode.

```
sssXXX              := 0xXXA0
Write_Address       := Write_char
```

3.8.3 Reset from Flash Fast Mode Command Sequence

The following command sequence has to be used after Fast writing to reset the Fast Mode.

```
sssXXX := 0xXX90
sssXXX := 0xXXF0 or sssXXX := 0xXX00
```

3.9 Hardware Sequence Flags

The following tables show the hardware sequence flags after writing a command sequence to the Flash memory. Each light-grey line represents a read access to the Flash memory.

3.9.1 Flash Programming

The following table shows the read-out behaviour of the hardware sequence flags after writing the write command sequence.

Please note that the `DQ7x` flag shows the inverted data bit #7 (`D7`) during write.

DQ7	DQ6	DQ5	DQ3	DQ2	Remark
DQ7x	1	0	0	1	DQ6 toggling
DQ7x	0	0	0	1	
DQ7x	1	0	0	1	
DQ7x	0	0	0	1	
. . .					
DQ7x	1	0	0	1	DQ6 toggling
D7	D6	D5	D3	D2	Finished: Read-out is Flash content (0xXXXX)

When switching from flag mode to data mode (final step) the data read can be corrupted for a single read out. Therefore if DQ5 gets "1" during read-out (time-out error), please check immediately DQ7 by reading the sequence flags again. If DQ7 still equals to DQ7x, then there was error in the write.

3.9.2 Sector Erase

The following table shows the read-out behaviour of the hardware sequence flags after writing the sector erase command sequence.

DQ7	DQ6	DQ5	DQ3	DQ2	Remark
1	1	0	0	1	Sector erase wait time (DQ3 = 0); DQ6, DQ2 toggling
1	0	0	0	0	DQ6, DQ2 toggling
1	1	0	0	1	
1	0	0	0	0	
. . .					
1	0	0	0	0	DQ6, DQ2 toggling
0	1	0	1	1	Sector erase starts (DQ3 = 1); DQ6, DQ2 toggling
0	0	0	1	0	DQ6, DQ2 toggling
0	1	0	1	1	
. . .					
0	1	0	1	1	DQ6, DQ2 toggling
1	1	1	1	1	Finished: Read-out erased Flash content (0xFFFF)

When switching from flag mode to data mode (final step) the data read can be corrupted for a single read out. Therefore if DQ5 gets "1" during read-out (time-out error), please check immediately DQ7 by reading the sequence flags again. If DQ7 still equals to DQ7x, then there was error in the write.

3.9.3 Sector Erase Suspend

The Sector Erase Suspend command suspends the sector erase operation being executed and enables data to be read from or written to sectors that are not being erased. Entering the Sector Erase Suspend command during the sector erase wait period will immediately terminate sector erase wait, cancel the erase operation, and set the erase stop state. Entering the Erase Suspend command during the erase operation after the sector erase wait period has terminated will set the erase suspend state after a maximum period of 20µs has elapsed.

The following table shows the read-out behaviour of the hardware sequence flags with addresses within one of the sectors that have been erased after writing the sector erase suspend command sequence.

DQ7	DQ6	DQ5	DQ3	DQ2	Remark
0	1	0	1	1	Sector being erased; DQ6, DQ2 toggling
0	0	0	1	0	DQ6, DQ2 toggling
0	1	0	1	1	
. . .					
0	1	0	1	1	DQ6, DQ2 toggling
1	1	0	0	0	Sector Erase suspended; DQ2 toggling
1	1	0	0	1	DQ2 toggling
1	1	0	0	0	
. . .					

3.9.4 Sector Erase Resume

The following table shows the read-out behaviour of the hardware sequence flags after writing the sector erase resume command sequence.

DQ7	DQ6	DQ5	DQ3	DQ2	Remark
1	1	0	0	1	Sector in suspend state; DQ2 toggling
1	1	0	0	0	DQ2 toggling
1	1	0	0	1	
. . .					
1	1	0	0	0	DQ2 toggling
0	0	0	1	1	Sector Erase resumed; DQ6, DQ2 toggling
0	1	0	1	0	DQ6, DQ2 toggling
0	0	0	1	1	
. . .					

3.9.5 Chip Erase

The following table shows the read-out behaviour of the hardware sequence flags after writing the chip erase command sequence.

DQ7	DQ6	DQ5	DQ3	DQ2	Remark
0	1	0	1	1	DQ6, DQ2 toggling
0	0	0	1	0	
0	1	0	1	1	
0	0	0	1	0	
. . .					
0	1	0	1	1	DQ6, DQ2 toggling
1	1	1	1	1	Finished: Read-out erased Flash content (0xFFFF)

If `DQ5` gets "`1`" during read-out (time-out error), please check immediately `DQ7` by reading the sequence flags again. In the case `DQ7 = 1` no error occurred.

3.10 Notes on using Sequence Flags

The sequence flag word may contain invalid data at the transition to finished state. This has to be considered when using toggle flags. In this case a non-toggling state between two reads does not indicate a time-out error.

4 Flash Memory Examples

Examples for Flash Memory Access

4.1 Flash RAM Routines

MCUs without Main Flash Memory B or Data Flash memory need RAM code for programming the Main Flash Memory A. If the Cypress *Start.asm* file is used, please configure the settings as follows, to activate the automatic RAM copying of the `RAMCODE` section during start-up.

```
;=====================================================================
; 4.13  Enable RAMCODE Copying
;=====================================================================

#set      COPY_RAMCODE      ON      ; <<< enable RAMCODE section to
                                     ; be copied from ROM to RAM
```

The section `RAMCODE` has to be set with the linker options:

`-sc RAMCODE/BYTE=`*`RAM_NAME`*
`-sc @RAMCODE/BYTE=`*`ROM_NAME`*

This can be done in the linker settings as shown below:

Project -> Setup Project -> Linker -> Disposition/Connection → Set Section → Specify in Address.

Please note, the *`RAM_NAME`* and *`ROM_NAME`* are placeholders for the names of RAM and ROM sections in the predefined linker settings of our template project.

In the C module, which contains the RAM code, the section `RAMCODE` is opened with the following `#pragma` directive:

```
#pragma segment FAR_CODE=RAMCODE, attr=CODE, locate=0x2220
```

Please note, that `attr=` and `locate=` are optional.

To close the section, please use:

```
#pragma segment FAR_CODE
```

Please note, that the default section name with "`FAR_CODE`" only has to be used in medium or large memory model. In small or compact memory model "`CODE`" must be used instead.

4.2 Flash Memory Write

The following sample code shows, how to write a word to the Main Flash Memory A (Sector `SA0`). It should be noted that before writing the corresponding sector should be erased first.

```
#include "mb96348rs.h"

#define seq_AAA ((__far volatile unsigned int*)0xDF0AAA)    // sequence address
#define seq_554 ((__far volatile unsigned int*)0xDF0554)

#define SA0 ((__far unsigned int*)0xDF0000)  // Main Flash A sector SA0

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

#pragma segment FAR_CODE=RAMCODE

/*****************************************************************************/
// Write word data to Flash
//
// Input: address, data
// Output: 1 = successful write, 2 = time-out

unsigned char Main_Flash_write(volatile __far unsigned int *wr_adr,
                               volatile unsigned int wdata)
{
  unsigned char flag = 0;
  unsigned char MCSRA_save;
  unsigned int  MTCRA_save;

  // preparations
  MCSRA_save = MCSRA;  // save Flash settings
  MTCRA_save = MTCRA;

  MTCRA = 0x6E3F;      // slow down Flash access to 7 wait states

  MCSRA_WE = 1;        // set write enable flag

  // start Flash programming sequence
  *seq_AAA = 0x00AA;   // sends program command to the pointed address
  *seq_554 = 0x0055;
  *seq_AAA = 0x00A0;
  *wr_adr  = wdata;    // send data to the pointed address

  while(flag == 0)
  {
    if((*wr_adr & DQ7) == (wdata & DQ7))   // data polling
    {
      flag = 1;                            // successful programmed!
    }

    if((*wr_adr & DQ5) == DQ5)             // time out?
    {
      if((*wr_adr & DQ7) == (wdata & DQ7)) // successful anyway?
      {
        flag = 1;                          // successful programmed!
      }
      else
      {
        flag = 2;                          // time out error!
      }
    }
  }
```

```
  MTCRA = MTCRA_save;  // restore Flash settings
  MCSRA = MCSRA_save;

  MCSRA_WE = 0;        // reset write enable flag

  return(flag);
}

#pragma segment FAR_CODE


//---------------- Main Application -------------------------

void main(void)
{
  unsigned char result;

  . . .

  __DI();                                          // disable Interrupts globally
  FMWC1 = 0xFF;                                    // unlock SA0 - SA7
  result |= Main_Flash_write(SA0 + 2, 0xAA55);     // write 0xAA55 to 0xDF0004
  __EI();                                          // enable Interrupts globally


  . . .
}
```

Please note, that the section `RAMCODE` has to be defined in the linker settings as described in 4.1.

The sequence pointers and the word address pointer should be declared with the `volatile` qualifier, because the compiler may interpret them otherwise as redundant code when using compiler optimizations.

The sector which is written to must be unlocked by the `FMWCn` (2.4.4) register.

Please also note that during RAM execution no interrupts are allowed, if the interrupt vectors point to the Flash memory being accessed by erase or write. If interrupts are mandatory, the interrupt vector table and the service routines have to be mapped to the RAM area temporary. Please refer to application note *AN205548* for relocation details.

4.3 Flash Sector Erase

The following sample code shows, how to erase the Flash memory sector `SA0`

```
#include "mb96348rs.h"

#define seq_AAA ((__far volatile unsigned int*)0xDF0AAA)    // sequence address
#define seq_554 ((__far volatile unsigned int*)0xDF0554)

#define SA0 ((__far unsigned int*)0xDF0000)  // Main Flash A sector SA0

#define DQ7 0x0080            // data polling flag
#define DQ5 0x0020            // time limit exceeded flag
#define DQ3 0x0008            // sector erase timer flag

#pragma segment FAR_CODE=RAMCODE

/*****************************************************************************/
// Erase Flash Sector
//
// Input: sector address
// Output: 1 = successful write, 2 = time-out

unsigned char Main_Flash_sector_erase(volatile __far unsigned int *sec_adr)
{
  unsigned char flag = 0;
  unsigned char MCSRA_save;
  unsigned int  MTCRA_save;

  // preparations
  MCSRA_save = MCSRA;    // save Flash settings
  MTCRA_save = MTCRA;

  MTCRA = 0x6E3F;        // slow down Flash access to 7 wait states

  MCSRA_WE = 1;          // set write enable flag

  // start with Flash sector erase sequence
  *seq_AAA = 0x00AA;     // sends erase command to the pointed address
  *seq_554 = 0x0055;
  *seq_AAA = 0x0080;
  *seq_AAA = 0x00AA;
  *seq_554 = 0x0055;
  *sec_adr = 0x0030;

  while ((*sec_adr & DQ3) != DQ3);  // sector erase timer ready?
```

Please note that the section `RAMCODE` has to be defined in the linker settings as described in 4.1.

The sequence pointers and the sector address pointer should be declared with the `volatile` qualifier, because the compiler may interpret them otherwise as redundant code when using compiler optimizations.

The sector which is erased must be unlocked by the `FMWCn` (2.4.4) register.

```
while(flag == 0)
  {
    if((*sec_adr & DQ7) == DQ7)      // data polling
    {
      flag = 0x10;                   // successful erased!
    }

    if((*sec_adr & DQ5) == DQ5)      // time out?
    {
      if((*sec_adr & DQ7) == DQ7)    // successful anyway?
      {
        flag = 0x20;                 // successful erased!
      }
      else
      {
        flag = 0x40;                 // time out error!
      }
    }
  }

  MCSRA_WE = 0;           // reset write enable flag

  MTCRA = MTCRA_save;     // restore Flash settings
  MCSRA = MCSRA_save;

  return(flag);
}

#pragma segment FAR_CODE


//---------------- Main Application --------------------------

void main(void)
{
  unsigned char result;

  . . .

  __DI();                                     // disable Interrupts globally
  FMWC1 = 0xFF;                               // unlock SA0 - SA7
  result |= Main_Flash_sector_erase(SA0);     // erase SA0
  __EI();                                     // enable Interrupts globally

  . . .
}
```

Please note that during RAM execution no interrupts are allowed, if the interrupt vectors point to the Flash memory being accessed by erase or write. If interrupts are mandatory, the interrupt vector table and the service routines have to be mapped to the RAM area temporary. Please refer to application note *AN205548* for relocation details.

4.4 Flash Read Access Timing Configuration

The following sample code example demonstrates to configure the Flash timing registers for the optimum read access time after power-on at the desired clock frequencies.

```
void Flash_Time_Config(void)
{
  /* considering the fact that this function is directly called after power-on
   * reset, all the clocks are fed by CLKRC, which is 2 MHz. Also it is assumed that
   * the CLKMC is 4 MHz
   */

  /* The PLL clock is configured at 48 MHz, i.e. 4 MHz x 12 */
  PLLCR = 0x000B;

  /* CLKP1 and CLKB are divided by 2 and CLKP2 is divided by 4 */
  CKFCR = 0x3111;

  /* Set Flash timing as synchronous read with 3 wait states. Kindly note that it is
   * possible to switch to this setting from the default power-on settings of 0x0239
   * (CLKS1 < 25 MHz and CLKB < 25 MHz) since the CLKS1 and CLKB both are less than
   * 60 MHz, even if they are sourced from CLKRC (if the switch had not happened) or
   * CLKPLL (if the switch had happened)
   */

  MTCRA = 0x4B3B;

  /* CLKS1 and CLKS2 both set as CLKPLL 48 MHz, hence CLKP1 and CLKB both as 24 MHz and
   * CLKP2 as 12 MHz.
   */
  CKSR = 0xFA;

  /* Wait till PLL is stabilized */
  while (1 != CKMR_PCM);

  /* Set the synchronous read with 0 wait states that is the optimum setting
   * possible at CLKS1 of 48 MHz and CLKB of 24 MHz
   */
  MTCRA = 0x2208;
}
```

The above example aims to configure the Flash memory timing from the power-on default setting of synchronous read with 1 wait state (0x0239) to synchronous read with 0 wait states at the target CLKS1 of 48 MHz and CLKB of 24 MHz. The same is achieved in two steps.

In the first step the clocks are configured at the desired frequencies from their power-on default and the Flash timing is set to synchronous read with 3 wait state (0x4B3B). This is possible because the permitted frequency for CLKS1 and CLKB is less than 60 MHz and both the frequencies before and after the switching to CLKPLL is within this limit.

In the second step the Flash memory timing is set to synchronous read with 0 wait state (0x2208) after CLKS1 is switched to 48 MHz and CLKB is switched to 24 MHz. This is possible because the permitted frequency for CLKS1 is between 20 MHz and 50 MHz and also the permitted frequency for CLKB is between 10 MHz and 25 MHz.

4.5 Manual RAM Copy

If the user does not want to use the `COPY_RAMCODE` function of the *Start.asm* file, he can use the following sample code example to copy own sections or segments.

Assume the following module contains RAM routines those are linked to the ROM area and later needs to be copied to the RAM memory for execution.

RAM_Code.c

```
#pragma segment FAR_CODE=FLASH_CODE1

    . . .     // Flash Programming code 1 here

#pragma segment FAR_CODE

    . . .

#pragma segment FAR_CODE=FLASH_CODE2

    . . .     // Flash Programming code 2 here

#pragma segment FAR_CODE

    . . .
```

The following assembly sample code can be used to copy the code from above to the RAM area.

Copy_to_RAM.asm

```
; Copy Flash Routines to RAM
       .PROGRAM COPY_TO_RAM

; dummy sections (needed for SIZEOF-operator)
       .SECTION FLASH_CODE1
       .SECTION FLASH_CODE2

;-------------------------------------------
; ROM->RAM copy table section
       .SECTION COPY_TABLE, CONST, ALIGN=1

; provided by linker
       .IMPORT _RAM_FLASH_CODE1
       .IMPORT _ROM_FLASH_CODE1

       .IMPORT _RAM_FLASH_CODE2
       .IMPORT _ROM_FLASH_CODE2
```

```
; section table data entries
        .DATA.L _ROM_FLASH_CODE1, FLASH_CODE1
        .DATA.H SIZEOF FLASH_CODE1

        .DATA.L _ROM_FLASH_CODE2, FLASH_CODE2
        .DATA.H SIZEOF FLASH_CODE2

; c label export for main application
        .EXPORT _ramcopy

; copy program
        .SECTION COPY_PROGRAM, CODE, ALIGN=1

_ramcopy:   PUSHW (RW0, RW1)         ; save RW0 and RW1

        MOV  A, #BNKSEC COPY_TABLE   ; get bank of table
        MOV  DTB, A                  ; store bank in DTB
        MOVW RW1, #COPY_TABLE        ; get start offset of table
        OR   CCR, #H'20              ; System stack flag set (SSB used)
        BRA  rc2                     ; branch to loop condition
rc1:    MOVW A, @RW1+6               ; get bank of destination
        MOV  SSB, A                  ; save dest bank in SSB
        MOVW A, @RW1+2               ; get source bank
        MOV  ADB, A                  ; save source bank in ADB
        MOVW A, @RW1+4               ; move destination addr in AL
        MOVW A, @RW1                 ; AL ->AH, src addr -> AL
        MOVW RW0, @RW1+8             ; number of bytes to copy -> RW0
        MOVSI SPB, ADB               ; copy data
        MOVN A, #10                  ; length of one table entry is 10
        ADDW RW1, A                  ; set pointer to next table entry
rc2:    MOVW A, RW1                  ; get address of next block
        SUBW A, #COPY_TABLE          ; sub address of first block
        CMPW A, #SIZEOF (COPY_TABLE); all blocks processed ?
        BNE  rc1                     ; if not, branch

        POPW (RW0, RW1)              ; restore RW0 and RW1
        RETP

        .END
```

The copy routine can be called from any C module by simply writing:

```
ramcopy();
```

This function copies all sections which are registered in the table in the section `COPY_TABLE`. This table has the following form:

Table location	Source Start Address (ROM, 4 Bytes)	Target Start Address (RAM, 4 Bytes)	Section Size in Bytes (2 Bytes)
`COPY_TABLE`	`_ROM_FLASH_CODE1`	FLASH_CODE1	Size of `FLASH_CODE1`
`COPY_TABLE + 10`	`_ROM_FLASH_CODE2`	FLASH_CODE2	Size of `FLASH_CODE2`

To tell the linker and compiler that the Flash code should be linked into ROM area but compiled for RAM area, the following linker settings have to be used:

```
-sc @FLASH_CODE1/Code/Byte=_INROM
-sc @FLASH_CODE2/Code/Byte=_INROM
-sc FLASH_CODE1/Code/Byte=_INRAM
-sc FLASH_CODE2/Code/Byte=_INRAM
```

Please note that the names of the RAM and ROM areas (_INRAM, _INROM) depend on the linker settings of the memory definitions -ra and -ro.

4.6 Writing by Command Sequencer

The Data Flash memory offers a simplified means to write to Flash memory. In addition, this simplified method can be combined with DMA transfers and makes it most simply to copy chunks of data without CPU load.

Once the Command Sequencer is enabled, data can be written into Data Flash memory similar to storing data in RAM by copying it to a given location. The Command Sequencer takes care to internally generate the necessary programming command sequence. An interrupt can be generated as shown below in the sample code when the writing has finished.

```
void Data_Flash_write(volatile __far unsigned int *wr_adr,
                              volatile unsigned int wdata)
{
  unsigned char flag = 0;

  // Enable Command Sequencer
  DFWCA = 0x01;

  // clear status flags
  DFWSA = 0x00;

  *wr_adr  = wdata;    // send data to the pointed address

  // Wait until write has finished, i.e. Command Sequencer is in idle state
  while (DFWSA_ST != 0);

  // obtain result
  flag = (DFWSA & 0xF0) >> 4;

  // disable Command Sequencer
  DFWCA = 0x00;

  return(flag);
}
```

5 Additional Information

Information about Cypress Microcontrollers can be found on the following Internet page:

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

The software examples related to this application note is:

96340_flash_data

96340_flash_main

96340_flash_main_fast_mode

96340_flash_main_satellite

96340_flash_satellite

It can be found on the following Internet page:

http://www.cypress.com/16lx

Document History

Document Title: AN205556 - F²MC-16FX Family, Flash Memory

Document Number: 002-05556

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	NOFL	08/24/2006	Initial release
			08/31/2006	Corrected Magic Word ,Add note to disable read cache when reading Toggle Flags and Correct register names for Main Flash
			03/16/2007	Reviewed the document and updated with review findings
			06/29/2007	Updated
			10/26/2007	Sequence flag related corrections
			12/30/2009	Update to new register names, current timing settings; replaced Main Flash, Satellite Flash by Main Flash A and B, added Data Flash
*A	5080684	NOFL	03/07/2016	Migrated Spansion Application Note from MCU-AN-300218-E-V15 to Cypress format
*B	5870360	AESATP12	09/04/2017	Updated logo and copyright.

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
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

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

© Cypress Semiconductor Corporation, 2006-2017. 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, WICED, 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.