Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

AN205488

Fcr4, Flash Programming

FCR4 family devices have two different kinds of Flash. One Flash is used to store the program code, it is called TC Flash (Tightly Coupled Flash) or Instruction Flash. The other Flash is likely to be used as a replacement for an external EEPROM, it is called EE Flash (EEPROM Emulation Flash).

Contents

1 Introduction

The Cypress Cortex-R4(F) (FCR4) family devices have two different kinds of Flash. One Flash is used to store the program code, it is called TC Flash (**T**ightly **C**oupled Flash) or Instruction Flash. The other Flash is likely to be used as a replacement for an external EEPROM, it is called EE Flash (**E**EPROM **E**mulation Flash). The EE Flash is optional and may not necessarily be included in every FCR4 family device.

This document describes the requirements and summarizes the most important points that must be considered in order to program both Flash types. The parallel Flash programming process where the MCU pins are directly mapped to Flash address/data/control bus signals is not in the scope of this document.

2 TC Flash programming

This chapter gives information about the TC Flash and its programming process.

2.1 Overview of TC Flash

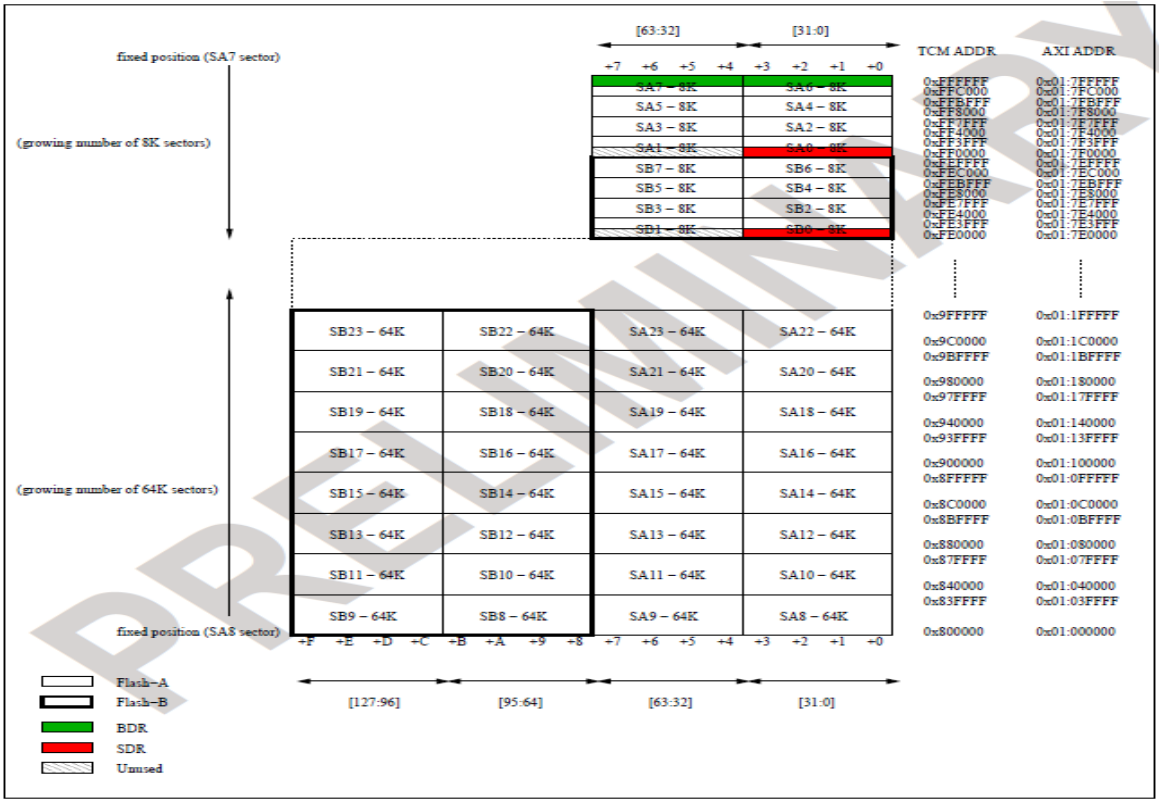
2.1.1 Features

- Up to 8 MB of Flash memory
- Multiple separate Flash macros are used to increase the performance by reducing Flash wait states. The number of Flash macros is device-specific, e.g. MB9DF126 ‘Atlas’ comprises 2 macros whereas MB9DF125 ‘Atlas-L’ comprises only 1 macro
- Accessible via AXI and TCM interface
- ECC support
- Small and big Flash sectors available with different organization
- Protection of sectors is possible
- Programming/Erasing via "Auto Algorithm" sequences (or parallel Flash programming, not described in this document)

2.1.2 Memory map

The following figure shows the memory map of the TC Flash in MCUs with 2 Flash macros and the sector and macro organization; refer to the Hardware Manual and device specific datasheet for the latest version of this figure.

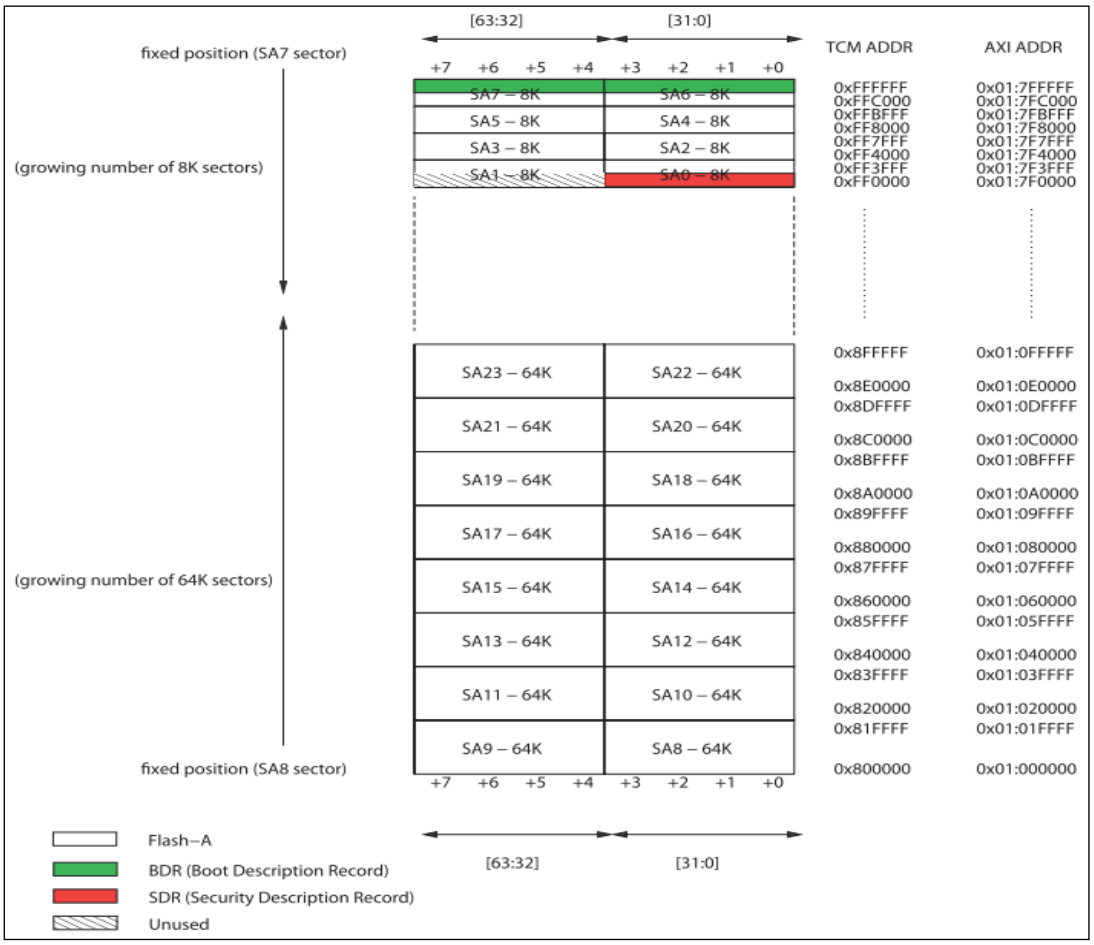
Figure 1:TC Flash memory map and organization (MCUs with 2 Flash macros)



The following figure shows the memory map of the TC Flash in MCUs with 1 Flash macro and the sector and macro organization, refer to the Hardware Manual and device specific datasheet for the latest version of this figure.

PRELIMINARY

Figure 2: TC Flash Memory Map and Organization (MCUs with 1 Flash Macro)



2.2 TC Flash Organization

Every Flash macro provides 8 small Flash sectors (S0 - S7) of 8 KB size each and 16 big sectors (S8 - S23) of 64 KB size each. Besides general Flash status and control registers there is also a certain set of Flash status control registers for every macro that is implemented in an MCU. Refer to chapter 2.1.2 Memory map for a better understanding of the available sectors and their organization that is described in the following sub-chapters.

2.2.1 Sector Interleaving

The sectors are organized in an "interleaved" manner, where two sectors are used to build up a 64-bit word. This means that subsequent 32-bit accesses to increasing addresses are made to different sectors.

2.2.2 Macro Interleaving

The TC Flash in MB9DF126 ‘Atlas’ is built up from two separate Flash macros 'A' and 'B'. The sectors in these Flash macros are called SA0, SA1, SA2, … and SB0, SB1, SB2, … respectively.

The small sector areas of these 2 Flash macros are placed consecutively in the address map, but for the big sectors macro interleaving will be applied. This means that a 128-bit word is built up of 2 64-bit words from every Flash macro. These 64-bit words are again built up of 2 sectors, and as a result of this the 128-bit word is placed across 4 different sectors (2 of each Flash macro).

2.2.3 Conclusion

Due to sector and macro interleaving one always has to consider which macros and sectors are affected by a desired action.

For example in case a certain part of code shall be erased by using the sector erase command, one has to obey the following procedure:

1. Determine whether the code is located in the big or small sector area
2. Identify the occupied sectors of the code that shall be erased. Most likely at least 2 sectors for small sector code and at least 4 sectors for big sector code must be erased.
3. Issue the sector erase command on every affected macro (most likely 1 macro for small sector code and 2 macros for big sector code). The adding of sectors to the sector erase command ("multiple sector erasing") is only possible for sectors on the same Flash macro
4. Check the status feedback of every affected Flash macro by reading its related status register or the HW sequence flags.

2.3 Special TC Flash areas

There are various so called "description records" that are located at pre-determined and fixed addresses in the small sector areas of the Flash macros. Their existence must be considered when programming / erasing the Flash. For the occupied addresses and organization of these description records refer to Hardware Manual chapter 14a “Boot ROM Software Interface”

2.3.1 Boot Description Record

This description record defines the start-up / boot behavior of the MCU. There is only one such description record that is located in Flash macro A

2.3.2 Main Security Description Record

This description record defines some global security settings for the MCU, e.g., a 128-bit device security key, a 128-bit field failure analysis key and other control bits. There is only one such description record that is located in Flash macro A.

If a wrong configuration is written to the Main Security Description Record it may no longer be possible to access the MCU. In order to recover from this state the MCU must be put in a Flash parallel programming tool and a Flash macro erase must be triggered.

2.3.3 TC Flash Security Description Record

Every Flash macro contains a TC Flash Security Description Record, where two permission sets, a key to change the active permission set and a so-called link key for "inter-macro security" can be configured. In the permission sets the read, write, and execute permissions can be configured for every Flash sector individually

2.4 Preconditions for TC Flash Programming / Erasing

Following preconditions must be met before any command (e.g. program, erase, ...) can be executed on the TC Flash:

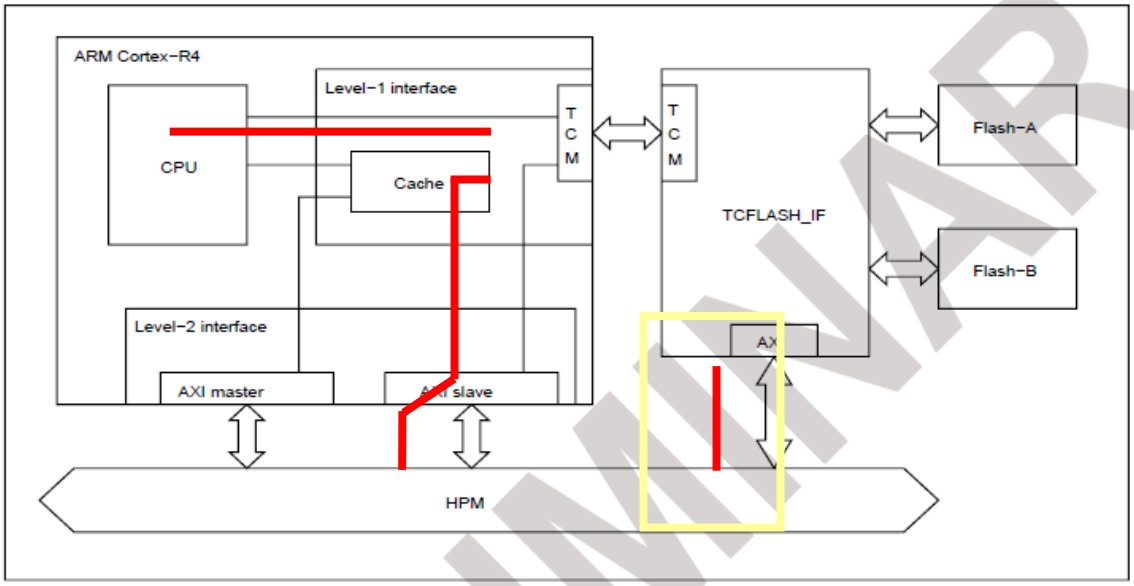
- Code must be run in privileged mode. Privileged are all ARM modes besides User (USR) mode.
- Code must not be executed from Flash macros that are affected by the Flash commands. For most cases, it might be necessary to download or copy the Flash routines to a RAM in the target MCU. For some use cases, it might be possible to run the routines from Flash (e.g. programming of Flash macro A with Flash routines that are located in small sectors of Flash macro B).
- Writing to Flash must be enabled by setting TCFCFG_FCFGR_WE (because also the command sequences are actually write accesses to Flash). The TCFCFG_FCFGR register is a protected register. Before executing the write access, the register must be unlocked by writing a protection key to the TCFCFG_FCPROTKEY register.
- A device security setting may have already been configured. If for example an external software tool wants to access the MCU via JTAG in order to upload a Flash programming kernel, the MCU must be unsecured first. Refer to the device security related chapter in the Hardware Manual.
- Flash security settings for one or more sectors may have been configured. There may exist different Flash permission sets that can be switched by providing the correct key. One could be used for normal program run mode whereas the other could allow Flash programming to enable application updates. This must be regarded in order to avoid errors during the programming process.
- Data cache must be disabled temporarily before the Flash command sequences are executed, otherwise they may not be recognized as command sequences because the write accesses do not reach the Flash interface, but are kept in cache. It may be sufficient to just set the AXI address area of the TCFLASH as non-cacheable using the Cortex-R4 MPU
- Any exception that could possibly occur while Flash content is modified (i.e. erased or written) like a Data Abort exception, any NMI exception or IRQs if not disabled, must have a corresponding exception handler that is placed in RAM.
- Code that handles the watchdog will also be not executable if not placed in RAM.

2.5 TC Flash Access Methods

2.5.1 Address Space

The TC Flash has two bus interfaces and therefore it can be accessed using different address ranges. One of these interfaces connects to the ARM TCM interface and the other one is connected via AXI to the high-performance bus matrix (refer the figure below).

Figure 3: TC Flash System Connectivity



Following address ranges can be used for accessing the TC Flash, refer to the device specific datasheet for the actual addresses:

- via TC Flash AXI (must be used for any Flash command sequences and reading hardware sequence flags!)
- via TCM interface directly from CPU core
- via Arm® Cortex®-R4 AXI slave port routed to TCM interface

If code is linked to the TCM interface address space the programming software must be responsible for adding the required offset to the addresses, so that the TC Flash AXI address space is used for programming.

The possibility to link code and data to TCM interface address space depends on whether the actual used device is included in the list of affected devices of Customer Information document CI707-00003!

2.5.2 Arm Memory Types

The whole address space in any Cortex-R4 controller (and others) is divided in various areas. Each of these areas is assigned a default ARM memory type attribute. One of the following three memory type attributes is possible for any area (with some sub-attributes like shared, non-shared, cacheable, non-cacheable):

- Strongly Ordered
- Device
- Normal

An important difference between 'Normal' type memory and 'Device'/'Strongly Ordered' type memory is that the Cortex-R4 core is allowed to merge store instructions that are addressed to 'Normal' type memory. The above-mentioned TC Flash AXI address space starts at address 0x01000000 which belongs to a 'Normal' type memory area of the default memory map. Flash commands consist of multiple store instructions, hence there is the very likely chance that these write instructions are merged to a lower number of actual executed Flash accesses. As a result, the TC Flash interface will not recognize these instructions as valid Flash commands.

Example:

The following Flash command sequence

1. Write 0xAA to cmd_addr0

2. Write 0x55 to cmd_addr1
3. Write 0xA0 to cmd_addr0
4. Write programming data to programming address

may result in the following actually executed accesses:

1. Write 0xA0 to cmd_addr0
2. Write 0x55 to cmd_addr1
3. Write programming data to programming address

In the example above the core has merged the two write accesses to cmd_addr0 and only writes the latest data value because there is no awareness that these writes actually have an effect at the target as long as the target is assigned the ‘Normal’ memory type.

Merging happens in a store buffer that is included in the CPU core.

There are three possibilities to overcome these implications:

- By using a DSB instruction after any instruction of the Flash command sequence. The DSB instruction drains the Cortex-R4 internal store buffer so that merging is not possible any longer
- Changing the memory type attribute of the TC Flash AXI address space to ‘Device’ or ‘Strongly Ordered’ memory type. This re-assignment can be achieved by using the ARM Cortex-R4 memory protection unit (MPU).
- While the core is in debug state every instruction that is executed via Instruction Transfer Register drains the store buffer, so that merging cannot happen

2.5.3 Access Width

Command sequences can be written with any access width and only the least significant byte will serve as the "command byte", but for the actual programming access (writing programming data to address that shall be programmed) the possible access width is dependent on the TC Flash ECC setting:

- ECC enabled (TCFCFG_FECCCTRL_ECCOFF == 0):
  Use 32-bit access!
- ECC disabled (TCFCFG_FECCCTRL_ECCOFF == 1):
  8-, 16-, 32-bit access can be used

The Flash macros actually support only 16-bit accesses. When using 32-bit accesses one has to execute the programming sequence to the same address with the same data twice. It is automatically taken care by the TC Flash interface to program the least significant 16-bit of programming data on the first access and the most significant 16-bit of programming data on the second access. In that way the TC Flash can also correctly calculate and program the ECC data for the whole 32-bit word on the second programming access. Refer also to descriptions of bits TCFCFG_FSTATn_WR32F and TCFCFG_FICTRLn_WR32FC.

The ECC enable/disable bit is writable only once after reset. Data that had been programmed with ECC disabled should not be accessed when ECC is enabled (initial state after reset) because ECC bits will not be programmed if ECC is disabled.

2.6 TC Flash Command Sequences

A command sequence consists of 1 to 6 "cycles" where certain data is written to certain addresses in the Flash address space. There are 4 possible address types that are used to form the command sequence:

- PA (programming address): Address where data shall be programmed
- SA (sector address): Any address inside the sector that is the target for the command (e.g. sector erase)
- cmd_addr0 (command address 0): A certain address inside the sector that is the target for the command. It is used to build up a command sequence. Some parts of the actual value are also dependent on whether the target sector is a small or big sector and whether it is located in Flash macro A or B
- cmd_addr1 (command address 1): Another address similar to cmd_addr0. Refer description of cmd_addr0

There are 2 possible types of data values that must be written to the addresses mentioned above:

- PD (programming data): The data that actually shall be programmed

- certain command data: A fixed value (depending on the type of command sequence and the cycle in the sequence), e.g. 0xF0, 0xAA, 0x55, …

The command address offsets can be found in the table below (but always check and compare with latest official documents like Hardware Manual and Datasheets). The offsets must be applied to masked target addresses to get the full command addresses as can be seen in an example further below:

MCU type	Cmd address type	Small sector*	Big sector (macro A)	Big sector (macro B)
2 macros	cmd_addr0	0xs550	0x2AA0	0x2AA8
	cmd_addr1	0xsAA8	0x1550	0x1558
1 macro	cmd_addr0	0xs550	0x1550	
	cmd_addr1	0xsAA8	0x0AA8	

*nibble ‘s’:

- s[3:1]: keep target sector address value
- s[0]:
  - ‘1’ for cmd_addr0
  - ‘0’ for cmd_addr1

Example:

Target address for programming shall be 0x017FC014 (which is in a small sector)

→cmd_addr0 = (0x017FC014 & 0xFFFFE000) + 0x1550 = 0x017FD550

→cmd_addr1 = (0x017FC014 & 0xFFFFE000) + 0x0AA8 = 0x017FCAA8

For example the sequence to program the data "PD" to the address "PA" is as follows:

1. Write 0xAA to cmd_addr0
2. Write 0x55 to cmd_addr1
3. Write 0xA0 to cmd_addr0
4. Write PD to PA

The correct way of programming data if ECC is enabled is the following:

1. Optionally check TCFCFG_FSTATn_WR32F for the correct state. This flag is toggled automatically by the TC Flash interface on subsequent programming accesses and controls which part of the 32-bit data + ECC bits are programmed
2. Write 0xAA to cmd_addr0 (any access width)
3. Write 0x55 to cmd_addr1 (any access width)
4. Write 0xA0 to cmd_addr0 (any access width)
5. Write 32-bit PD to PA (use 32-bit access width, 32-bit aligned!)
6. Check programming status
7. Repeat steps 1 - 6

The following figure shows the Flash command sequences of the TC Flash, refer to the Hardware Manual and device specific datasheet for the latest version of this figure.

Figure 4: Overview of TC Flash Command Sequences

Command sequence	Bus write access	1st bus write cycle		2nd bus write cycle		3rd bus write cycle		4th bus write cycle		5th bus write cycle		6th bus write cycle	
		Address	Data	Address	Data	Address	Data	Address	Data	Address	Data	Address	Data
Read/reset	1	cmd_addr0	0xF0	-	-	-	-	-	-	-	-	-	-
Write/ program	4	cmd_addr0	0xAA	cmd_addr1	0x55	cmd_addr0	0xA0	PA	PD	-	-	-	-
Macro erase	6	cmd_addr0	0xAA	cmd_addr1	0x55	cmd_addr0	0x80	cmd_addr0	0xAA	cmd_addr1	0x55	cmd_addr0	0x10
Sector erase	6	cmd_addr0	0xAA	cmd_addr1	0x55	cmd_addr0	0x80	cmd_addr0	0xAA	cmd_addr1	0x55	SA	0x30
Sector erase suspend	1	SA	0xB0	-	-	-	-	-	-	-	-	-	-
Sector erase resume	1	SA	0x30	-	-	-	-	-	-	-	-	-	-

PRELIM

2.7 Retrieving Status Information from TC Flash

Before and / or after executing a command sequence one probably wants to be informed about the current status of the Flash macros or the command. There are two possibilities to retrieve status information from the Flash macros, one is to read the status bits from the Flash interface registers and the other one is to read the so called hardware sequence flags. Both methods can also be used in conjunction, of course.

2.7.1 Status Register

Every Flash macro provides its own status register TCFCFG_FSTATn. In this register a RDY and HANG bit can be found together with their corresponding RDYINT and HANGINT interrupt flags that are set on the transition from 0 to 1 of the corresponding status bit. These flags can be cleared by using their clear bits in the TCFCFG_FICTRLn register. The interrupts can be enabled there as well if this is desired.

2.7.2 Hardware Sequence Flags

More detailed information about the current Flash macro state can be retrieved by reading the Flash macro hardware sequence flags.

The Flash macro interface modifies the data that is read from the Flash while a command is in progress or the Flash macro is in a special state. It depends on the command whether the hardware sequence flags are read from any Flash address or just from addresses in a specific sector. For example when the Sector Erase Suspend command has been executed successfully, a read access to the sectors that have been marked for erasure returns the hardware sequence flags that will indicate that these sectors are in Sector Erase Suspend state. Reading other sectors will return the "real" data that had been programmed to the Flash.

Toggle bits are included in the hardware sequence flags so that reading flags can be distinguished from reading real data, because the toggle bits will change their state ('0' to '1' and vice versa) on every read access.

The following figure shows the hardware sequence flags of the TC Flash, refer to the Hardware Manual and device specific datasheet for the latest version of this figure.

Figure 5: Overview of Hardware Sequence Flags

State		DQ [7,15]	DQ [6,14]	DQ [5,13]	DQ [3,11]	DQ [2,10]	HANG[4]
Hardware reset		DATA[1] [7,15]	DATA [6,14]	DATA [5,13]	DATA [3,11]	DATA [2,10]	0
Normal command state		DATA [7,15]	DATA [6,14]	DATA [5,13]	DATA [3,11]	DATA [2,10]	0
Program state		/DATA[2]	T[3]	0	0	0	0
Macro erase state		0	T	0	1	T	0
Command timeout state		0	T	0	0	T	0
Sector erase state		0	T	0	1	T	0
Erase suspend state[5]	Read on erase sector	0	0	0	1	T	0
	Read on non erase sector	DATA [7,15]	DATA [6,14]	DATA [5,13]	DATA [3,11]	DATA [2,10]	0
Hangup-1 state[6]	Program	/DATA	T	1	0	0	1
	Macro erase	0	T	1	1	T	1
	Sector erase	0	T	1	1	T	1

PRELIMINARY

2.8 Differences of MCUs with 1 Flash Macro and 2 Flash Macros

Following list shall summarize the points which need to be considered when developing software that shall handle MCUs with 1 Flash macro and MCUs with 2 Flash macros.

Item	Difference
Command addresses	The command addresses for the big sectors differ (see chapter 2.6)
Control & Status registers	In 1 macro devices the SW does no longer need to distinguish the target macro to access the correct pair of Flash Control and Status registers (TCFCFG_FICTRLn and TCFCFG_FSTATn)
Flash size	Available Flash address space is different, e.g. if driver functions include parameter checks, the range must be adapted (see chapter 2.1.2)
Big sector organization	The address increment to get from one pair of interleaved big sectors of a Flash macro to the next pair of big sectors of the same macro is different (see chapter 2.1.2)

3 EE Flash Programming

This chapter gives information about the EE Flash and its programming process.

3.1 Overview of EE Flash

Differences in the feature set of EEFLASH are mainly caused by the availability of SHE (Secure Hardware Extension) in the actual used device.

For a detailed list of differences refer to the following Hardware Manual chapters:

- Chapter 10 “EEPROM Emulation Flash”
- Chapter 10a “EEPROM Emulation Flash and SHE”

3.1.1 Features

- Up to 8 sectors of 8 KB Flash memory each
- Programming/Erasing via "Auto Algorithm" sequences (or parallel Flash programming, not described in this document)
- Optional Command Sequencer Mode that handles the Flash programming command sequence. Can be combined with DMA operation
- ECC support

- Protection of sectors is possible
- Support of two address mirror areas

3.1.2 Memory Map

The following figure shows the memory map of the EE Flash on MB9DF126 ‘Atlas’ and the address mirror and sector organization, refer to the Hardware Manual and device specific datasheet for the latest version of this figure and for the absolute addresses.

Figure 6: EE Flash Address Mirror Organization

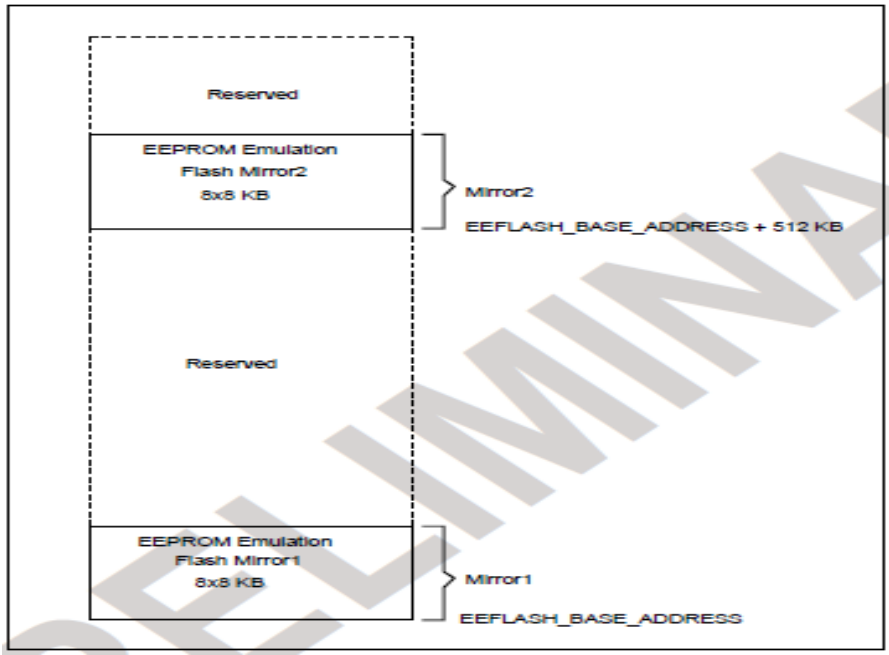
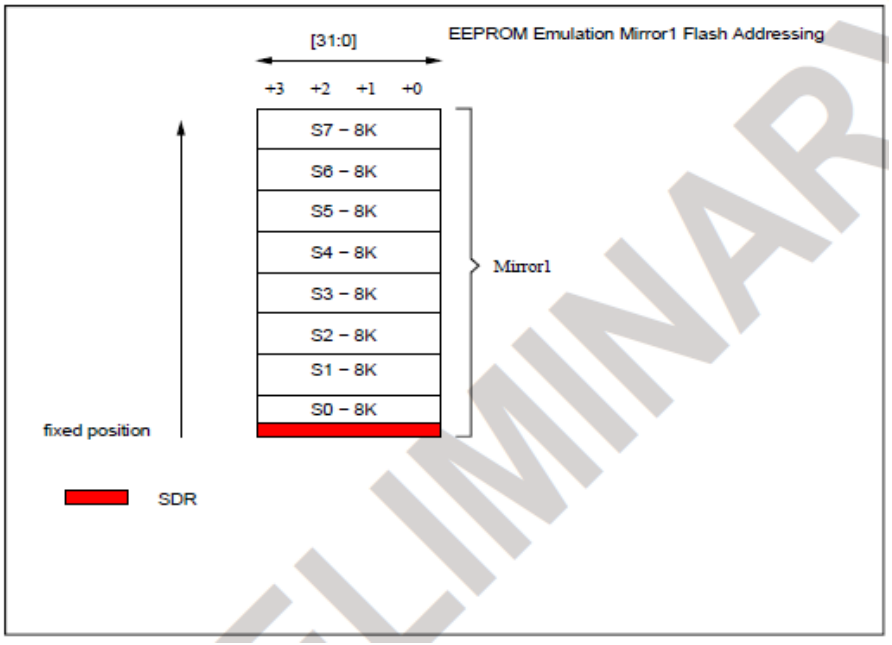


Figure 7: EE Flash Sector Organization



3.2 EE Flash Organization

In contrast to the TC Flash the organization of EE Flash sectors is rather simple. The EE Flash macro provides up to 8 Flash sectors (S0 - S7) of 8 KB size each, that are consecutively placed in the memory map without using any sector interleaving.

3.3 Special EE Flash Areas

There is a so called Security Description Record that is located at a pre-determined and fixed address in the EE Flash. Its existence must be considered when programming / erasing the Flash. For the occupied addresses and organization of this description record refer to Hardware Manual chapter 14a “Boot ROM Software Interface”

3.3.1 EE Flash Security Description Record

The EE Flash contains an EE Flash Security Description Record, where two permission sets, a key to change the active permission set and a so called link key for "inter-macro security" can be configured. In the permission sets for every Flash sector the read, write, and execute permissions can be configured separately

3.4 Preconditions for EE Flash Programming / Erasing

Following preconditions must be met before any command (e.g. program, erase, ...) can be executed on the EE Flash:

- Writing to Flash must be enabled by setting EEFCFG_CR_WE (because also the command sequences are actually write accesses to Flash). The EEFCFG_CR register is a protected register. Before executing the write access, the register must be unlocked by writing a protection key to the EEFCFG_CPR register. This configuration must be executed in privileged mode. Privileged are all ARM modes besides User (USR) mode. In contrast to the TC Flash all command sequences to EE Flash can also be executed in non-privileged mode.
- A device security setting may have already been configured. If for example an external software tool wants to access the MCU via JTAG in order to upload a Flash programming kernel, the MCU must be unsecured first. Refer to the device security related chapter in the Hardware Manual.
- Flash security settings for one or more sectors may have been configured. There may exist different Flash permission sets that can be switched by providing the correct key. One could be used for normal program run mode whereas the other could allow Flash programming to enable application updates. This must be regarded in order to avoid errors during the programming process.

3.5 EE Flash Access Methods

3.5.1 Address Space

The EE Flash is accessible via two different address mirror areas. Refer chapter 3.1.2

- Mirror 1: This mirror should be used for programming. The current state of EEFCFG_ECR_ECCOFF is regarded on read and write accesses
- Mirror 2: This mirror allows reading and writing the EE Flash without ECC functionality independent of the current EEFCFG_ECR_ECCOFF state

3.5.2 ARM Memory Types

The problem of the default ARM memory type for TC Flash address space that is described in chapter 2.5.2 does not apply to the EE Flash because it is located in an area with the ‘Device’ memory type attribute. Hence, store merging does not occur.

3.5.3 Access Width

Command sequences can be written with any access width and only the least significant byte will serve as the "command byte", but for the actual programming access (writing programming data to address that shall be programmed) the possible access width is dependent on the EE Flash ECC setting:

- ECC enabled (EEFCFG_ECR_ECCOFF == 0):
  Use 32-bit access!
- ECC disabled (EEFCFG_ECR_ECCOFF == 1):
  8-, 16-, 32-bit access can be used

The Flash macro actually supports only 16-bit accesses. When using 32-bit accesses one has to execute the programming sequence to the same address with the same data twice. It is automatically taken care by the EE Flash interface to program the least significant 16-bit of programming data on the first access and the most significant 16-bit of programming data on the second access if the Automatic Mode (EEFCFG_WMER_AME == 1) is enabled beforehand. In that way the EE Flash can also correctly calculate and program the ECC data for the whole 32-bit word on the second programming access.

The ECC enable/disable bit is writable only once after reset. Data that had been programmed with ECC disabled should not be accessed when ECC is enabled (initial state after reset) because ECC bits will not be programmed if ECC is disabled.

3.6 EE Flash Command Sequences

A command sequence consists of 1 to 6 "cycles" where certain data is written to certain addresses in the Flash address space. There are 4 possible address types that are used to form the command sequence:

- PA (programming address): Address where data shall be programmed
- SA (sector address): Any address inside the sector that is the target for the command (e.g. sector erase)
- cmd_addr0 (command address 0): A certain address inside the sector that is the target for the command. It is used to build up a command sequence.
- cmd_addr1 (command address 1): Another address similar to cmd_addr0. Refer description of cmd_addr0

There are 2 possible types of data values that must be written to the addresses mentioned above:

- PD (programming data): The data that actually shall be programmed
- certain command data: A fixed value (depending on the type of command sequence and the cycle in the sequence), e.g. 0xF0, 0xAA, 0x55, …

For example the sequence to program the data "PD" to the address "PA" is as follows:

1. Write 0xAA to cmd_addr0
2. Write 0x55 to cmd_addr1
3. Write 0xA0 to cmd_addr0
4. Write PD to PA

The correct way of programming data if ECC is enabled and the Automatic Write Mode shall be used is the following:

1. Set EEFCFG_WMER_AME to ‘1’
2. Write 0xAA to cmd_addr0 (any access width)
3. Write 0x55 to cmd_addr1 (any access width)
4. Write 0xA0 to cmd_addr0 (any access width)
5. Write 32-bit PD to PA (use 32-bit access width, 32-bit aligned!)
6. Check programming status
7. Repeat steps 2 - 6

The correct way of programming data if ECC is enabled and the Manual Write Mode shall be used is the following:

1. Set EEFCFG_WMER_MME to ‘0’ (and ensure that EEFCFG_WMER_AME is set to ‘0’)
2. Write 0xAA to cmd_addr0 (any access width)
3. Write 0x55 to cmd_addr1 (any access width)
4. Write 0xA0 to cmd_addr0 (any access width)
5. Write 32-bit PD to PA (use 32-bit access width, 32-bit aligned!)
6. Check programming status

7. Set EEFCFG_WMER_MME to ‘1’
8. Write 0xAA to cmd_addr0 (any access width)
9. Write 0x55 to cmd_addr1 (any access width)
10. Write 0xA0 to cmd_addr0 (any access width)
11. Write 32-bit PD to PA (use 32-bit access width, 32-bit aligned!)
12. Check programming status

The following figure shows the Flash command sequences of the EE Flash on MB9DF126 ‘Atlas’, refer to the Hardware Manual and device specific datasheet for the latest version of this figure.

Figure 8: Overview of EE Flash Command Sequences

Command sequence	Bus write access	1st bus write cycle		2nd bus write cycle		3rd bus write cycle		4th bus write cycle		5th bus write cycle		6th bus write cycle	
		Address	Data	Address	Data	Address	Data	Address	Data	Address	Data	Address	Data
Read/reset	1	mmmm XXXX	F0	-	-	-	-	-	-	-	-	-	-
Write/ program	4	mmmm XAA8	AA	mmmm X554	55	mmmm XAA8	A0	PA	PD	-	-	-	-
Macro erase	6	mmmm XAA8	AA	mmmm X554	55	mmmm XAA8	80	mmmm XAA8	AA	mmmm X554	55	mmmm XAA8	10
Sector erase	6	mmmm XAA8	AA	mmmm X554	55	mmmm XAA8	80	mmmm XAA8	AA	mmmm X554	55	SA	30
Sector erase suspend	1	mmmm XXXX	B0	-	-	-	-	-	-	-	-	-	-
Sector erase resume	1	mmmm XXXX	30	-	-	-	-	-	-	-	-	-	-
Unlock bypass set	3	mmmm XAA8	AA	mmmm X554	55	mmmm XAA8	20	-	-	-	-	-	-
Unlock bypass program (*1)	2	mmmm XXXX	A0	PA	PD	-	-	-	-	-	-	-	-
Unlock bypass reset (*1)	2	mmmm XXXX	90	mmmm XXXX	XX	-	-	-	-	-	-	-	-

mmmm: Flash memory address, X: Arbitrary value (0/1), PA: Program Address, PD: Program Data

SA: Sector Address

PRELIM

3.7 Retrieving Status Information from EE Flash

Before and / or after executing a command sequence one probably wants to be informed about the current status of the Flash macro or the command. There are two possibilities to retrieve status information from the Flash macro, one is to read the status bits from the Flash interface registers and the other one is to read the so called hardware sequence flags. Both methods can also be used in conjunction, of course.

3.7.1 Status Register

The EE Flash macro provides the status register EEFCFG_SR. In this register a RDY and HANG bit can be found together with their corresponding RDYINT and HANGINT interrupt flags that are set on the transition from 0 to 1 of the corresponding status bit. These flags can be cleared by using their clear bits in the EEFCFG_ICR register. The interrupts can be enabled there as well if this is desired.

3.7.2 Hardware Sequence Flags

More detailed information about the current Flash macro state can be retrieved by reading the Flash macro hardware sequence flags.

The Flash macro interface modifies the data that is read from the Flash while a command is in progress or the Flash macro is in a special state. It depends on the command whether the hardware sequence flags are read from any Flash address or just from addresses in a specific sector. For example when the Sector Erase Suspend command has been executed successfully, a read access to the sectors that have been marked for erasure returns the hardware sequence flags that will indicate that these sectors are in Sector Erase Suspend state. Reading other sectors will return the "real" data that had been programmed to the Flash.

Toggle bits are included in the hardware sequence flags so that reading flags can be distinguished from reading real data, because the toggle bits will change their state ('0' to '1' and vice versa) on every read access.

The following figure shows the hardware sequence flags of the EE Flash on MB9DF126 ‘Atlas’, refer to the Hardware Manual and device specific datasheet for the latest version of this figure.

Figure 9: Overview of Hardware Sequence Flags

State		DQ [7,15]	DQ [6,14]	DQ [5,13]	DQ [3,11]	DQ [2,10]	HANG[4]
Hardware reset		DATA[1] [7,15]	DATA [6,14]	DATA [5,13]	DATA [3,11]	DATA [2,10]	0
Normal command state		DATA [7,15]	DATA [6,14]	DATA [5,13]	DATA [3,11]	DATA [2,10]	0
Program state		/DATA[2]	T[3]	0	0	0	0
Macro erase state		0	T	0	1	T	0
Command timeout state		0	T	0	0	T	0
Sector erase state		0	T	0	1	T	0
Erase suspend state[5]	Read on erase sector	0	0	0	1	T	0
	Read on non erase sector	DATA [7,15]	DATA [6,14]	DATA [5,13]	DATA [3,11]	DATA [2,10]	0
Hangup-1 state[6]	Program	/DATA	T	1	0	0	1
	Macro erase	0	T	1	1	T	1
	Sector erase	0	T	1	1	T	1

PRELIMINARY

3.8 Command Sequencer Mode and DMA Mode

The Command Sequencer Mode supports the programming of the EE Flash. If this mode is enabled the Flash programming command sequence is handled by hardware inside the EE Flash interface.

- Using Command Sequencer Mode is only possible if ECC is enabled (EEFCFG_ECR_ECCOFF == 0), hence only 32-bit write accesses are allowed
- As long as Command Sequencer Mode is enabled, no other Flash commands are recognized because they are treated as normal writes that will program the respective command addresses.
- Before writing a word it must be waited for the Command Sequencer to become idle (EEFCFG_WSR_ST == 0)
- If the DMA Mode (EEFCFG_WCR_DMAEN == 1) together with Command Sequencer Mode is enabled, the EE Flash interface will generate a DMA request on the transition of the Command Sequencer to the idle state. If a DMA transfer is configured appropriately, large amounts of data can be programmed to EE Flash autonomously.

4 Appendix

4.1 References

Related documents for further information are listed in the table below:

Ref. #	Document file name	Description
1	FCR4_Cluster_Series_Hardware_Manual.pdf	FCR4 Cluster Series Hardware Manual
2	mb9df126-ds707-00002-e.pdf	FCR4 Cluster Series MB9DF126 - ATLAS Datasheet
3	DDI0363D_cortexr4_r1p3_trm.pdf	ARM Cortex-R4 and Cortex-R4F Technical Reference Manual
4	DDI0406B_arm_architecture_reference_manual_errata_markup_4_0.pdf	ARM® Architecture Reference Manual ARM®v7-A and ARM®v7-R edition

Document History

Document Title: AN205488 - FCR4, Flash Programming

Document Number: 002-05488

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	MKEA	03/10/2011	V1.0, CEy First draft
			06/16/2011	V1.1, CEy Added note about mandatory use of TC Flash AXI address space to chapter 1 and 2.5.1
			04/16/2012	V1.2, CEy Removed FPGA chapter which is obsolete, added further items to chapter 2.4, minor corrections and changes throughout the whole document
			07/26/2012	V1.3, CEy Checked whole chapter 2 in order to make adaptations reflecting differences of MCUs with 1 TCFLASH macro and those with 2 TCFLASH macros. (Changes in chapters 2.1.2, 2.6, added chapter 2.8)
			03/18/2013	V1.4, CEy Removed "CONFIDENTIAL" footer, some minor fixes, updated EEFLASH chapter (SHE case)
*A	5074293	MKEA	12/31/2015	Converted Spansion Application Note “MCU-AN-300128-E-V14” to Cypress format
*B	5873173	AESATMP8	09/05/2017	Updated logo and Copyright.
*C	6054547	NOFL	02/12/2018	Updated links Updated template

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 MCU

Cypress Developer Community

Community | 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, 2011-2018. 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. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress does not assume any liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. In addition, the products described in these materials may contain design defects or errors known as errata which may cause the product to deviate from published specifications. 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.