User Guide

Enhanced Flash Driver (EFD) 2.2

Introduction

The Enhanced Flash Driver (EFD) described in this document is intended to simplify the process of developing application code for Micron flash memory devices. The source code has been written to ANSI C standards and is intended to be compiled with any ANSI C compliant compiler/development environment.

All operations with the Micron flash devices are implemented to comply with the datasheet documented functionality relating to the flash device. Please refer to the datasheet for details of each function. The comments in the source code should not serve as a replacement for the detailed documentation provided by the datasheet.

This user guide does not replace the datasheet to flash devices either. It refers to the datasheets throughout, and it is necessary to have copies of the datasheets to follow some of the explanations.

Contents

About this Document

Goals and Objectives

The goal of the Enhanced Flash Driver (EFD) is to provide support for Micron flash memory devices. EFD provides great flexibility in RAM/ROM scalability and supports multiple operating systems. The core product is architected in such a way that it can be used on several operating systems. The operating system (OS) abstraction layer can be implemented for each target operating system as part of the EFD porting effort. However, the core product will remain the same across all target operating systems.

The current driver package is composed of low-level functions for a set of Micron flash memory devices and external Application Programming Interface (API) functions that support the unified interface of functions and operations for all supported flash memory devices. As a result, future device changes will not necessarily lead to the code changes in the application environments.

Users can use low-level functions directly or adapt them. The high-level code accesses the flash memory device by calling the low-level code. This means that users do not have to concern themselves with the details of the special command sequences. The resulting source code is both simpler and easier to maintain.

Additionally, the EFD package contents an external API layer that can be used as is for a task or as reference code that demonstrates the usage of low-level functions.

With the provided software driver interface, users can focus on writing the high-level code required for their particular applications.

Audience

This user guide describes the API and structures associated with the EFD. The application notes explain how to modify the source code for individual target hardware. The source code contains comments throughout, explaining how it is used and why it has been written the way it has.

The target audience includes engineers who develop different tools working with Micron flash memory devices. This document was designed to be used as a reference guide for all operations supported by Micron flash memory devices.

Terminology

Table 1: Glossary of Terms

Term	Definition
API	Application Programming Interface
Block	A group of bits, bytes or words within the flash memory array that erase simultaneously when the Erase command is given to the device.
CFI	The Common Flash Interface is an open standard jointly developed by AMD*, Intel*, Sharp* and Fujitsu*.
Command	Command refers to the implementation of a specific datasheet command.
EFD	Enhanced Flash Driver
HW	Hardware

Table 1: Glossary of Terms (Continued)

Term	Definition
OS	Operating System
OTP	One-Time Programmable. This is the part of memory that can be programmed once and cannot be erased.
OUM	Ovonic Unified Memory (also known as PCM, PRAM, PCRAM and Chalcogenide RAM C-RAM) is a type of non-volatile computer memory.
SnD	Store and Download. In a SnD system, code is copied into RAM and then executed from RAM.
SPI	The Serial Peripheral Interface is a synchronous serial data link standard named by Motorola* that operates in full duplex mode. Devices communicate in master/slave mode where the master device initiates the data frame.
SW-RWE	Software Read While Erase
SW-RWW	Software Read While Write
WORD	WORD in context this document is used to describe the smallest assessable unit of a flash memory device. In configuration with a single 16-bit flash device, a WORD would be 16 bits (two bytes). In configuration with two 16-bit flash devices, a WORD would be 32 bits (two bytes).
XiP	eXecute in Place. XiP systems execute code directly from the flash memory device without first copying the code into RAM.

Related Documents

Table 2: Related Documentation

Document Name	Link
Micron Site	http://www.micron.com
Technical Notes	http://www.micron.com/support/tn-directory
Customer Service Notes	http://www.micron.com/support/csn
JEDEC Standard: Common Flash Interface (CFI)	http://www.jedec.org/download/search/jesd68-01.pdf

Features

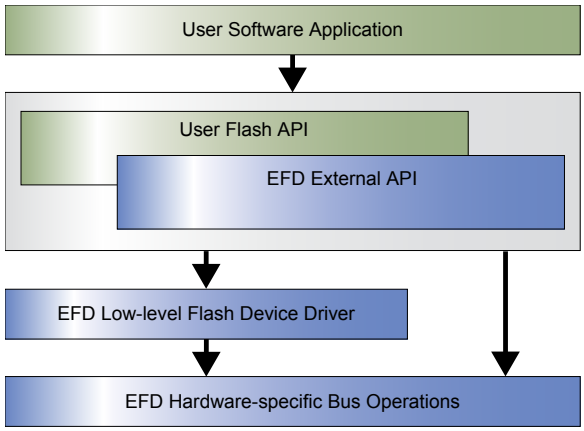
Layered Architecture

The developed code using the provided drivers can be divided into three layers:

1. The hardware-specific bus operations
2. The low-level code
3. External API code, or the high-level code written by the user.

The low-level code requires hardware-specific Bus Read and Bus Write operations to communicate with the flash memory device. This implementation is hardware-platform dependent as it depends on the microprocessor on which the C code runs and on the location of the memory in the microprocessor's address space. The user must write the C functions that are suitable for their hardware platform. See Hardware-specific Bus Operations (page 58). The low-level code takes care of issuing the correct sequences of write operations for each command and of interpreting the information received from the device during programming or erasing. See Low-level Flash Device Driver (page 51). The functionality implemented in this driver is taken directly from the device datasheet. Please read the datasheet for a better understanding of the device's behavior and operation. This should be of special concern if the device is to be used in a non-sequential way. The external API demonstrates the usage of flash operations. See External API Details (page 23).

Figure 1: Layered Architecture



Supported Memory Devices

EFD supports the following flash memory devices:

- P30/33 130nm StrataFlash® Embedded Memory
- P30/33 65nm StrataFlash® Embedded Memory
- J3D Embedded Flash Memory
- J3 65nm Embedded Flash Memory
- C3 Advanced+ Boot Flash Memory
- S33 Serial Flash Memory

Some devices can be supported with limitations, which are described in the section EFD Assumptions (page 66).

Supported Flash Layouts

Flash devices may be organized in a number of different ways. Most flash devices are available in either an x8, x16 or x32 configuration. Additionally, NOR flash often consists of two or four devices, which are interleaved. EFD supports both configurations.

Unified Interface

The current driver package is composed of low-level functions for a set of Micron flash devices and external API functions that support the unified interface of functions and operations for all supported flash memory devices.

Configurability

EFD provides a set of options to enable/disable low-level operations if users are concerned about the code size.

Software Read While Write

Many real-time applications require the capability to interrupt a flash operation write with a higher priority request. For example, while a flash device is writing data, a high priority read request may need to be executed.

The external API introduces Software Read While Write capability. A software read-while-write (SW RWW) appears to execute code from the flash device while a write operation is in process.

The key to the software read-while-write protocol is the capability to:

- Poll interrupts
- Suspend the flash if there is an interrupt pending
- Put the array back into read array
- Enable the interrupt code to be executed directly from flash

Software read-while-write is inefficient due to the amount of time spent polling interrupts and the additional latency associated with the flash suspend.

While Software Read While Write provides excellent flexibility, it typically comes with a performance tax for suspending/un-suspending flash operations.

Software Read While Erase

Many real-time applications require the capability to interrupt a flash operation erase with a higher priority request. For example, while a flash device is erasing block, a high priority read request may need to be executed.

The external API introduces Software Read While Erase capability. A software read-while-erase (SW RWE) appears to execute code from the flash device while an erase operation is in process.

The key to the software read-while-erase protocol is the capability to:

- Poll interrupts
- Suspend the flash if there is an interrupt pending
- Put the array back into read array
- Enable the interrupt code to be executed directly from flash

Software read-while-erase is inefficient due to the amount of time spent polling interrupts and the additional latency associated with the flash suspend.

EFD API Reference

The EFD API is a set of functions common to all Micron flash memory devices.

Structures and Types

Basic Types

Check whether the compiler to be used supports the following basic data types, as described in the source code, and change it if necessary:

```
typedef unsigned char  UINT8;    /* 8-bit  unsigned integer */
typedef char  INT8;    /* 8-bit  signed integer */
typedef unsigned short UINT16;    /* 16-bit unsigned integer */
typedef short INT16;    /* 16-bit signed integer */
typedef unsigned long  UINT32;    /* 32-bit unsigned integer */
typedef long  INT32;    /* 32-bit signed integer */
typedef UINT8 * UINT8_PTR;    /* pointer to 8-bit unsigned inte-
ger */
typedef UINT16 * UINT16_PTR;    /* pointer to 16-bit unsigned in-
teger */
typedef UINT32 * UINT32_PTR;    /* pointer to 32-bit unsigned in-
teger */
```

Device Information

This structure describes the type of flash device.

```
typedef struct
{
   UINT16   VendorId;
   UINT16   DeviceId;
} FLASH_ID;
```

Description

Structure	Member	Description
FLASH_ID	VendorId	Manufacturer code and device ID code.
FLASH_ID	DeviceId	Device ID code.

Interface ID Code

```
typedef enum {
   FLASH_CMDSET_SERIAL,
   FLASH_CMDSET_GENERIC,
   FLASH_CMDSET_AMD
} CMDSET_ID;
```

Description

Interface ID	Description
FLASH_CMDSET_SERIAL	Serial
FLASH_CMDSET_GENERIC	Generic: CFI command set 0x0001,0x0003
FLASH_CMDSET_AMD	Generic: CFI command set 0x0002,0x0004

Erase Block Description

This structure describes an erase block.

```
typedef struct _FLASH_ERASE_BLOCK
{
   UINT16            BlockSize;
   UINT16            BlockCount;
 } FLASH_ERASE_BLOCK, *PFLASH_ERASE_BLOCK;
```

Description

Structure	Member	Description
FLASH_ERASE_BLOCK	BlockSize	Size of the erase blocks (bits 0-15 = z, region erase block(s) size are z x 256 bytes).
FLASH_ERASE_BLOCK	BlockCount	Number of the erase blocks (bits 0-15 = y, y +1 = number of identical-size erase blocks).

Device Description

This structure stores information about device capabilities.

```
typedef struct _FLASH_DESCRIPTION
{
   FLASH_ID                              FlashId;
   UINT8                               FlashType;
   CMDSET_ID                              CommandSetId;
   UINT8                               WriteBufferSize;
   UINT8                               DeviceSizeExponent;
   UINT8                               Capabilities;
   UINT8                               EraseRegionCount;
   FLASH_ERASE_BLOCK       EraseBlockRegionIn-
fo[FLASH_MAX_ERASE_BLOCK_TYPES];
} FLASH_DESCRIPTION, *PFLASH_DESCRIPTION;
```

Description

Structure	Member	Description
FLASH_DESCRIPTION	FlashId	Manufacturer code/device ID code.
FLASH_DESCRIPTION	FlashType	The supported flash device ID.
FLASH_DESCRIPTION	CommandSetId	Interface ID code.
FLASH_DESCRIPTION	WriteBufferSize	This value, when used as the exponent in the equation 2^n, specifies the maximum number of bytes in the write buffer.
FLASH_DESCRIPTION	DeviceSizeExponent	This value, when used as the exponent in the equation 2^n, specifies the total size, in bytes, of the device (all erase block regions combined).
FLASH_DESCRIPTION	EraseRegionCount	Number of the erase block regions (x) within the device: 1. x = 0 means no erase blocking; the device erases in bulk. 2. x specifies the number of device regions with one or more contiguous same-size erase blocks.

Structure	Member	Description
FLASH_DESCRIPTION	EraseBlockRegionInfo	Information about the Erase Block Size and Count.

Flash Operation Object

This structure sets low-level flash device operations.

```
typedef union _FLASH_OPERATION
{
    struct
        {
FLASH_ERROR                         (*DeviceId)( FLASH_ID * );
            FLASH_ERROR                         (*ReadCfi)
( UINT32, UINT32, UINT8 * );
            FLASH_ERROR                          (*ReadCfiEx)
( FLASH_CFI, UINT32 * );
            UINT32                              (*ReadFlash)
( UINT32 );
            FLASH_ERROR                          (*WriteFlash)
( UINT32, UINT32 );
            FLASH_ERROR                          (*BufferProgram)
( UINT32, UINT32, UINT8 * );
            FLASH_ERROR                          (*ProgramSuspend)
( UINT32 );
            FLASH_ERROR                          (*ProgramResume)
( UINT32 );
            FLASH_ERROR                          (*EraseSuspend)
( UINT32 );
            FLASH_ERROR                          (*EraseResume)
( UINT32 );
            FLASH_ERROR                          (*BlockErase)
( UINT32 );
            UINT32                                  (*BlockState)
( UINT32 );
            FLASH_ERROR                          (*BlockUnLock)
( UINT32 );
            FLASH_ERROR                          (*BlockLockDown)
( UINT32 );
            FLASH_ERROR                          (*BlockLock)
( UINT32 );
            UINT32                                  (*ReadStatus)
( UINT32 );
            void
(*ClearStatus)( UINT32 );
            void                                     (*ReadMode)
( UINT32 );
            FLASH_ERROR                          (*ReadProtReg)
( UINT32, UINT32 * );
            FLASH_ERROR                          (*WriteProtReg)
( UINT32, UINT32 );
            FLASH_ERROR                          (*ReadConfReg)
( UINT32, UINT16 * );
            FLASH_ERROR                          (*WriteConfReg)
( UINT32, UINT16 );
            FLASH_ERROR                          (*BitAltWrite-
Flash)( UINT32, UINT32 );
            FLASH_ERROR                          (*BitAltBuffer-
```

```
Program)( UINT32, UINT32, UINT8 * );
            FLASH_ERROR                             (*StreamModeEn-
try)( UINT32 );
            FLASH_ERROR                             (*StreamModeExit)
( UINT32 );
            FLASH_ERROR                             (*BlankCheck)
( UINT32 );
            FLASH_ERROR                             (*ProgramEfi)
( UINT16, UINT32, UINT32, UINT8 * );
    } GenOp;

    struct
    {
            FLASH_ERROR                             (*DeviceId)
( FLASH_ID * );
            UINT32                                     (*ReadStatus)
( void );
            FLASH_ERROR                             (*WriteStatus)
( UINT32 );
            FLASH_ERROR                             (*ClearStatus)
( void );
            FLASH_ERROR                             (*WriteEnable)
( void );
            FLASH_ERROR                             (*WriteDisable)
( void );
            FLASH_ERROR                             (*BulkErase)
( void );
            FLASH_ERROR                             (*SectorErase)
( UINT32 );
            FLASH_ERROR                             (*PageProgram)
( UINT32, UINT32, UINT8 * );
            FLASH_ERROR                             (*ReadFlash)
( UINT32, UINT32, UINT8 * );
    } SpiOp;

} FLASH_OPERATION, *PFLASH_OPERATION;
```

Fields

For an explanation, see Low-level Functions Provided (CFI COMMANDSET) (page 52).

Flash device object

This structure describes the flash device.

```
typedef struct _FLASH_DEVICE_OBJECT
{
   FLASH_DESCRIPTION                                    Desc;
   FLASH_OPERATION                                     Ops;

UINT32
  StartingAddress;

UINT32
  Size;

UINT32
  BufferSize;

UINT8
```

```
 DataWidth;

UINT8
 Interleave;
} FLASH_DEVICE_OBJECT, *PFLASH_DEVICE_OBJECT;
```

Description

Structure	Member	Description
FLASH_DEVICE_OBJECT	Desc	This field keeps information about the device.
FLASH_DEVICE_OBJECT	Ops	A list of all low-level functions.
FLASH_DEVICE_OBJECT	StartingAddress	Start address of the flash memory device (valid system address).
FLASH_DEVICE_OBJECT	Size	The density of the flash device in bytes.
FLASH_DEVICE_OBJECT	BufferSize	The on-chip write buffer size in bytes.
FLASH_DEVICE_OBJECT	DataWidth	Minimum size, in bytes, of data used to operate with the flash memory device.
FLASH_DEVICE_OBJECT	Interleave	Number of physical flash devices interleaved across the data bus. Flash devices may be organized in a number of different ways. Most flash devices are available in either an x8, x16 or x32 configuration. Additionally, NOR flash devices often consist of two or four devices, which are interleaved.

Commands for IOCTL Operations

```
typedef enum {
   FLASH_CMD_ID
   FLASH_CMD_CFI
   FLASH_CMD_CFI_EX
   FLASH_CMD_READ
   FLASH_CMD_WRITE
   FLASH_CMD_BUFFERED_WRITE
   FLASH_CMD_SUSPEND
   FLASH_CMD_RESUME
   FLASH_CMD_BLOCK_SUSPEND
   FLASH_CMD_BLOCK_RESUME
   FLASH_CMD_BLOCK_ERASE
            FLASH_CMD_BLOCK_STATE
   FLASH_CMD_BLOCK_UNLOCK
   FLASH_CMD_BLOCK_LOCKDOWN
   FLASH_CMD_BLOCK_LOCK
   FLASH_CMD_READ_STATUS
   FLASH_CMD_CLEAR_STATUS
   FLASH_CMD_READ_PROTECTION_REG
   FLASH_CMD_WRITE_PROTECTION_REG
   FLASH_CMD_READ_CONFIGURATION_REG
   FLASH_CMD_WRITE_CONFIGURATION_REG
   FLASH_CMD_ALTERABLE_WRITE
   FLASH_CMD_ALTERABLE_BUFFERED_WRITE
   FLASH_CMD_STREAM_ENTRY
   FLASH_CMD_STREAM_EXIT
   FLASH_CMD_BLANK_CHECK
   FLASH_CMD_EFI
```

```
   FLASH_CMD_SPI_ID
   FLASH_CMD_SPI_READ_STATUS
   FLASH_CMD_SPI_WRITE_STATUS
   FLASH_CMD_SPI_CLEAR_STATUS
   FLASH_CMD_SPI_WRITE_ENABLE
   FLASH_CMD_SPI_WRITE_DISABLE
   FLASH_CMD_SPI_BULK_ERASE
   FLASH_CMD_SPI_SECTOR_ERASE
   FLASH_CMD_SPI_PAGE_PROGRAM
   FLASH_CMD_SPI_READ_FLASH
} FLASH_COMMAND;
```

Description

FLASH COMMAND, INTEL CFI COMMAND SET	Description
FLASH_CMD_ID	Retrieve the device ID information from flash.
FLASH_CMD_CFI	Read CFI Information.
FLASH_CMD_CFI_EX	Get data from predefined CFI offsets.
FLASH_CMD_READ	Read WORD from the flash memory device.
FLASH_CMD_WRITE	Write WORD into the flash memory device.
FLASH_CMD_BUFFERED_WRITE	Use a buffered write operation.
FLASH_CMD_SUSPEND	Launch the program suspend command.
FLASH_CMD_RESUME	Launch the program resume command.
FLASH_CMD_BLOCK_SUSPEND	Launch the erase suspend command.
FLASH_CMD_BLOCK_ERASE	Launch the erase operation.
FLASH_CMD_BLOCK_STATE	Read the current block state.
FLASH_CMD_BLOCK_UNLOCK	Execute the block unlock operation.
FLASH_CMD_BLOCK_LOCKDOWN	Execute the block lockdown operation.
FLASH_CMD_BLOCK_LOCK	Execute the block lock operation.
FLASH_CMD_READ_STATUS	Read the Status register.
FLASH_CMD_CLEAR_STATUS	Clear the Status register.
FLASH_CMD_READ_PROTECTION_REG	Read data from protection registers.
FLASH_CMD_WRITE_PROTECTION_REG	Write data to registers.
FLASH_CMD_READ_CONFIGUARTION_REG	Read data from the RCR register.
FLASH_CMD_WRITE_CONFIGURATION_REG	Write data to the RCR register.
FLASH_CMD_ALTERABLE_WRITE	Write WORD from the PCM Memory.
FLASH_CMD_ALTERABLE_BUFFERED_WRITE	Write buffer into the PCM Memory.
FLASH_CMD_STREAM_ENTRY	Enable Streaming Mode for Program Region.
FLASH_CMD_STREAM_EXIT	Disable Streaming Mode for Program Region.
FLASH_CMD_BLANK_CHECK	Confirm whether or not a main-array block is completely erased.
FLASH_CMD_EFI	Execute subcommand operations.

FLASH COMMAND, SPI	Description
FLASH_CMD_SPI_ID	The Read ID command reads three bytes of data describing the flash memory device.
FLASH_CMD_SPI_READ_STATUS	Continuously polls the SPI Status Register.
FLASH_CMD_SPI_WRITE_STATUS	Allows the user to write to the writable Status Register bits.
FLASH_CMD_SPI_CLEAR_STATUS	Resets bit SR5 (Erase Fail Flag) and bit SR6 (Program Fail Flag).
FLASH_CMD_SPI_WRITE_ENABLE	The Write Enable command sets the WEL bit.
FLASH_CMD_SPI_WRITE_DISABLE	The Write Disable command clears the WEL bit.
FLASH_CMD_SPI_BULK_ERASE	The Bulk Erase command serially erases the entire main array, including the parameter blocks.
FLASH_CMD_SPI_SECTOR_ERASE	The Sector Erase command is used to erase a memory sector.
FLASH_CMD_SPI_PAGE_PROGRAM	The Page Program command programs 1 bit to 256 bytes of data within a 256-Byte-Aligned memory segment.
FLASH_CMD_SPI_READ_FLASH	Read data from the flash memory device.

Parameters for IOCTL Operations

This structure is provided as a uniform way for passing addresses and data.

```
typedef union {
   FLASH_ID FlashId;

   struct {
      UINT32 Length;
      UINT32 Offset;
                        UINT8 *Data;
   } Cfi;

   struct {
                        FLASH_CFI Code;
      UINT32 Data;
   } CfiEx;

   struct {
      UINT32 Addr;
      UINT32 Value;
   } Read;

   struct {
      UINT32 Addr;
      UINT32 Value;
   } Write;

   struct {
      UINT32 Length;
      UINT32 Offset;
      UINT8 *Data;
   } BufWrite;

    struct {
```

```
      UINT32 Addr;
    } ProgramOp;

    struct {
      UINT32 Addr;
    } BlockOp;

    struct {
      UINT32 Addr;
      UINT32 State;
    } BlockState;

   struct {
      UINT16 OpCode;
      UINT32 Addr;
      UINT32 Length;
      UINT8 *Data;
   } Efi;

} FLASH_PARAMETER;
```

Description

Structure	Member	Description	Usage
FlashId	-	Manufacturer code and device ID code.	FLASH_CMD_ID
Cfi	Length	Size of the buffer, in bytes, pointed to by the data measured.	FLASH_CMD_CFI
Cfi	Offset	Offset in the Common Flash Interface (CFI) Query structure or database.	FLASH_CMD_CFI
Cfi	Data	Pointer to a buffer to place read data.	FLASH_CMD_CFI
CfiEx	Code	Code of place in the Common Flash Interface (CFI) Query structure or database.	FLASH_CMD_CFI_EX
CfiEx	Data	Storage for read data (valued data read by the predefined offset can be 8, 16 or 32 bytes in size).	FLASH_CMD_CFI_EX
Read	Addr	Offset in the flash device that is a measure of the distance in bytes from BASE_ADDR.	FLASH_CMD_READ FLASH_CMD_READ_STATUS FLASH_CMD_READ_PROTECTION_REG FLASH_CMD_READ_CONFIGURATION_REG
Read	Value	Data read by the set address offset. It can be 8, 16 or 32 bytes in size, depending on the flash device bus width and configuration.	FLASH_CMD_READ FLASH_CMD_READ_STATUS FLASH_CMD_READ_PROTECTION_REG FLASH_CMD_READ_CONFIGURATION_REG

Structure	Member	Description	Usage
Write	Addr	Offset in the flash device that is a measure of the distance in bytes from the BASE_ADDR.	FLASH_CMD_WRITE FLASH_CMD_ALTERABLE_WRITE FLASH_CMD_WRITE_PROTECTION_REG FLASH_CMD_WRITE_CONFIGURATION_REG
Write	Value	Data written by the set address offset. It can be 8, 16 or 32 bytes size depending on the flash device bus width and configuration.	FLASH_CMD_WRITE FLASH_CMD_ALTERABLE_WRITE FLASH_CMD_WRITE_PROTECTION_REG FLASH_CMD_WRITE_CONFIGURATION_REG
BufWrite	Length	The size, in bytes, of the buffer pointed to by the data.	FLASH_CMD_BUFFERED_WRITE FLASH_CMD_ALTERABLE_BUFFERED_WRITE
BufWrite	Offset	Offset in the flash device that is a measure of distance in bytes from the BASE_ADDR.	FLASH_CMD_BUFFERED_WRITE FLASH_CMD_ALTERABLE_BUFFERED_WRITE
BufWrite	Data	A pointer to a buffer with written data.	FLASH_CMD_BUFFERED_WRITE FLASH_CMD_ALTERABLE_BUFFERED_WRITE
ProgramOp	Addr	Offset in the flash device that is a measure of the distance in bytes from the BASE_ADDR.	FLASH_CMD_CLEAR_STATUS FLASH_CMD_SUSPEND FLASH_CMD_RESUME FLASH_CMD_STREAM_ENTRY FLASH_CMD_STREAM_EXIT
BlockOp	Addr	Offset in the flash device that is a measure of the distance in bytes from the BASE_ADDR.	FLASH_CMD_BLOCK_SUSPEND FLASH_CMD_BLOCK_RESUME FLASH_CMD_BLOCK_ERASE FLASH_CMD_BLOCK_UNLOCK FLASH_CMD_BLOCK_LOCKDOWN FLASH_CMD_BLOCK_LOCK
BlockState	Addr	Offset in the flash device that is a measure of the distance in bytes from the BASE_ADDR.	FLASH_CMD_BLOCK_STATE
BlockState	State	Bit-field information about the state of the directed block.	FLASH_CMD_BLOCK_STATE
Efi	OpCode	Sub-Op-Code.	FLASH_CMD_EFI
Efi	Addr	Offset in the flash device that is a measure of the distance in bytes from the BASE_ADDR.	FLASH_CMD_EFI
Efi	Length	Size of the buffer, in bytes, pointed to by the data.	FLASH_CMD_EFI
Efi	Data	Pointer to a buffer consisting of the address-data pair to place the program.	FLASH_CMD_EFI

CFI Offsets

The list of CFI offsets that can be used to get data.

```
#define CFI_QRY_OFFSET    0x0
#define CFI_PRI_OFFSET    0x1
typedef enum {
   FLASH_CFI_QRY                =CFI_CODE( CFI_QRY_OFFSET,
0x10 ),
   FLASH_CFI_CMD_SET_1            =CFI_CODE( CFI_QRY_OFFSET,
0x13 ),
   FLASH_CFI_EXT_QRY_ADDR_1        =CFI_CODE( CFI_QRY_OFFSET,
0x15 ),
   FLASH_CFI_CMD_SET_2            =CFI_CODE( CFI_QRY_OFFSET,
0x17 ),
   FLASH_CFI_EXT_QRY_ADDR_2        =CFI_CODE( CFI_QRY_OFFSET,
0x19 ),
   FLASH_CFI_DEVICE_SIZE           =CFI_CODE( CFI_QRY_OFFSET,
0x27 ),
   FLASH_CFI_BUFFER_SIZE           =CFI_CODE( CFI_QRY_OFFSET,
0x2A ),
   FLASH_CFI_NUM_ERASE_REGION_X    =CFI_CODE( CFI_QRY_OFFSET,
0x2C ),
   FLASH_CFI_NUM_ERASE_REGION_1    =CFI_CODE( CFI_QRY_OFFSET,
0x2D ),
   FLASH_CFI_SIZE_ERASE_REGION_1    =CFI_CODE( CFI_QRY_OFFSET,
0x2F ),
   FLASH_CFI_NUM_ERASE_REGION_2    =CFI_CODE( CFI_QRY_OFFSET,
0x31 ),
   FLASH_CFI_SIZE_ERASE_REGION_2    =CFI_CODE( CFI_QRY_OFFSET,
0x33 ),
   FLASH_CFI_PRI                =CFI_CODE( CFI_PRI_OFFSET,
0x00 ),
   FLASH_CFI_MJ_VERSION            =CFI_CODE( CFI_PRI_OFFSET,
0x03 ),
   FLASH_CFI_MN_VERSION            =CFI_CODE( CFI_PRI_OFFSET,
0x04 )
} FLASH_CFI;
```

Description

Parameter	Description
FLASH_CFI_QRY	Query-unique ASCII string “QRY”.
FLASH_CFI_CMD_SET_1	Primary vendor command set and control interface ID code. The ID code for vendor-specified algorithms.
FLASH_CFI_EXT_QRY_ADDR_1	Extended Query Table primary algorithm address.
FLASH_CFI_CMD_SET_2	Alternate vendor command set and control interface ID code. “0000h” indicates that no second vendor-specified algorithm exists.
FLASH_CFI_EXT_QRY_ADDR_2	Secondary algorithm Extended Query Table address. “0000h” indicates that none exist.
FLASH_CFI_DEVICE_SIZE	“n” such that the device size = 2n in number of bytes.
FLASH_CFI_BUFFER_SIZE	“n” such that the maximum number of bytes in the write buffer = 2n.

Parameter	Description
FLASH_CFI_NUM_ERASE_REGION_X	Number of erase block regions within the device: 1. x = 0 means no erase blocking; the device erases in "bulk". 2. x specifies the number of device or partition regions with one or more contiguous same-size erase blocks. 3. Symmetrically blocked partitions have one blocking region. 4. Partition size = (total blocks) x (individual block size).
FLASH_CFI_NUM_ERASE_REGION_1	Erase Block Region 1 Information. Bits 0-15 = y, y+1 = number of identical-size erase blocks.
FLASH_CFI_SIZE_ERASE_REGION_1	Erase Block Region 1 Information. Bits 0-15 = z, region erase block(s) size are z x 256 bytes.
FLASH_CFI_NUM_ERASE_REGION_2	Erase Block Region 2 Information. Bits 0-15 = y, y+1 = number of identical-size erase blocks.
FLASH_CFI_SIZE_ERASE_REGION_2	Erase Block Region 2 Information. Bits 0-15 = z, region erase block(s) size are z x 256 bytes.
FLASH_CFI_PRI	Primary extended query table. Unique ASCII string "PRI".
FLASH_CFI_MJ_VERSION	Major version number, ASCII.
FLASH_CFI_MN_VERSION	Minor version number, ASCII.

Error Codes

The EFD provides detailed error codes to describe the various errors that may occur during the operation. Some functions return an error number directly as the function value. These functions return a value of zero (FLASH_ERR_NONE) to indicate a success. If more than one error occurs when processing a function call, any one of the possible errors may be returned, as the order of detection is undefined. The following table lists all possible error codes that may be returned by the EFD and provides a description of each error code.

```
typedef enum
{
   FLASH_ERR_NONE = 0x0,
   FLASH_ERR_VPP,
   FLASH_ERR_SEQUENCE,
   FLASH_ERR_LOCKED,
   FLASH_ERR_ERASE,
   FLASH_ERR_PROGRAM,
   FLASH_ERR_SUSPEND,
   FLASH_ERR_TIMEOUT,
   FLASH_ERR_BAD_BLOCK,
   FLASH_ERR_BAD_ADDRESS,
   FLASH_ERR_CFI,
   FLASH_ERR_BAD_ARGUMENT,
   FLASH_ERR_UNSUPPORTED,
   FLASH_ERR_BUSY,
   FLASH_ERR_UNKNOWN
} FLASH_ERROR;
```

Description

Error Code	Description
FLASH_ERR_NONE	The function completed successfully.
FLASH_ERR_VPP	Vpp failure.
FLASH_ERR_SEQUENCE	A command sequence error.
FLASH_ERR_LOCKED	Locked.
FLASH_ERR_ERASE	Erase fails for block.
FLASH_ERR_PROGRAM	Operation programming failure.
FLASH_ERR_SUSPEND	Erase/program operation completed.
FLASH_ERR_TIMEOUT	Access to the device timed out.
FLASH_ERR_BAD_BLOCK	Block number not valid for this device.
FLASH_ERR_BAD_ADDRESS	Address not valid for this device.
FLASH_ERR_CFI	Failure experienced during CFI operations.
FLASH_ERR_BAD_ARGUMENT	Incorrect parameter.
FLASH_ERR_UNSUPPORTED	Memory device does not support operation or this operation is turned off in the configuration file.
FLASH_ERR_BUSY	The device is busy.
FLASH_ERR_UNKNOWN	General error.

EFD Organization

EFD is implemented to support three layers of abstractions, such as:

- External API
- Low-level flash device driver
- Hardware-specific bus operations

This chapter describes each layer of abstraction.

External API Details

The external API consists of several methods that allow the user to perform operations with a flash device in a unified way. These functions are more complex than low-level functions. They are able to check function arguments, address boundary, etc. EFD_Write() implements the SW-RWW algorithm. EFD_Erase() implements the SW-RWE algorithm. A software read-while-write/erase appears to execute code from the flash device while a write/erase operation is in process. The key to the software read-while-write/erase protocol is the capability to poll interrupts, suspend the flash if there is a interrupt pending, put the array back into read array and enable the interrupt code to be executed directly from flash. Software read-while-write/erase is inefficient due to the amount of time spent polling interrupts and the additional latency associated with the flash suspend. A user is responsible for locking/unlocking flash devices that are writing data to the flash device or erasing a block.

EFD_Init Function

The EFD_Init() function is used to a recognize a flash device and initialize the internal object referred to by fdo that should be FLASH_DEVICE_OBJECT. This function should be called at least once before using any other function.

Prototype

```
FLASH_ERROR EFD_Init(
        void              *fdo);
```

Parameters

Parameter	Description
fdo	(OUT) Pointer to the internal object allocated/deallocated by the user. It describes the flash device and stores a set of low-level operations.

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter was detected.	Failure
FLASH_ERR_UNSUPPORTED	The flash device is not supported or the function cannot be executed correctly because configuration options, such as EFD_CONF_DEVICE_ID, EFD_CONF_CFI and EFD_CONF_CFI_EX are set as FALSE.	Failure
FLASH_ERR_CFI	An error occurred during operations with CFI.	Failure

EFD_Exit Function

The EFD_Exit() function closes all internal objects.

Prototype

```
FLASH_ERROR EFD_Exit(
        void              *fdo);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describe the flash device and stores a set of low-level operations.

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid paramenter was detected.	Failure

EFD_Read Function

The EFD_Read() function attempts to read the length in bytes from the flash device, starting from a position associated with addr_offset into the buffer pointed to by buffer. The addr_offset argument can be unaligned with a WORD. Upon successful completion, the EFD_Read() function returns the amount of bytes actually read and places read data in ret_length, which can be less than or equal to the length argument.

For more information, refer to the EFD_Read API function usage example in the section "Read data from flash memory using a user-defined address and store one inside buffer".

Prototype

```
FLASH_ERROR EFD_Read(
void              *fdo,
        UINT32             addr_offset,
        UINT32             length,
        UINT8             *buffer,
        UINT32            *ret_length );
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
addr_offset	(IN) Offset in the flash device that is a measure of the distance in bytes from the beginning of the device.
length	(IN) Size of the buffer pointedto by the buffer measured in bytes.
buffer	(OUT) Pointer to a buffer with data to be read.
ret_length	(OUT) Number of bytes actually read.

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter is detected.	Failure
FLASH_ERR_BAD_ADDRESS	Address is not valid for this device.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure

EFD_Write Function

The EFD_Write() function attempts to write the length in bytes from the buffer pointed to by buffer to the flash device, starting from a position associated with addr_offset. The addr_offset argument can be unaligned with a WORD. Upon successful completion, the EFD_Write() function returns the amount of bytes actually written in ret_length, which can be less than or equal to the length argument. EFD_Write() implements the SW-RWW algorithm inside. For example, a record of data on a flash device can be suspended if somebody tries to read data from the flash device. After the completion of the read operation, the record procedure is the resumed value (for a detailed description, see the sectin "Software Read While Write").
Note: A user is responsible for lock/unlock operations when writing data to the flash device.

Refer the EFD_Write API function usage example in the section "Write data from buffer and write one into flash memory using user-defined address".

Prototype

```
FLASH_ERROR EFD_Write(
        void              *fdo,
        UINT32             addr_offset,
        UINT32             length,
        UINT8             *buffer,
        UINT32             *ret_length );
```

Parameters

Parameter	Description
fdo	(IN) Pointer to an internal object that describes the flash device and stores a set of low-level operations.
addr_offset	(IN) Offset in the flash device that is a measure of the distance in bytes from the beginning of the device.
length	(IN) The size, in bytes, of the buffer pointed to by the buffer.
buffer	(IN) The pointer to a buffer with data to be written.
ret_length	(OUT) The number of bytes actually written.

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter is detected.	Failure
FLASH_ERR_BAD_ADDRESS	The address not valid for this device.	Failure

Value	Description	Success/Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure
FLASH_ERR_TIMEOUT	An error due to a time-out lapse.	Failure
FLASH_ERR_VPP	A voltage range error.	Failure
FLASH_ERR_SEQUENCE	A sequence program error.	Failure
FLASH_ERR_PROGRAM	An operation programming failure.	Failure
FLASH_ERR_LOCKED	An operation was performed in a protected area.	Failure

EFD_Erase Function

The EFD_Erase() function allows users to erase a block within the flash hardware. The addr_offset should direct any address inside block to be erased. The addr_offset argument can be unaligned with WORD. The EFD_Erase() implements the SW RWE algorithm. For example, block erasing can be suspended if somebody tries to read data from the flash device. After the read operation completes, the erase procedure is resumed (for a detailed description, see the section "Software Read While Write").
Note: A user is responsible for lock/unlock operations when erasing a block.

Prototype

```
FLASH_ERROR EFD_Erase(
        void              *fdo,
        UINT32              addr_offset );
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
addr_offset	(IN) Offset in the flash device that is a measure of the distance in bytes from the beginning of the device.

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter was detected.	Failure
FLASH_ERR_BAD_ADDRESS	The address is not valid for this device.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure
FLASH_ERR_TIMEOUT	An error due to a time-out lapse.	Failure
FLASH_ERR_VPP	A voltage range error.	Failure
FLASH_ERR_SEQUENCE	A sequence program error.	Failure
FLASH_ERR_ERASE	An operation erasing failure.	Failure
FLASH_ERR_LOCKED	An operation was performed in a protected area.	Failure

EFD_Ioctl Function

The EFD_Ioctl() function provide the user with a common mechanism to execute different atomic operations on the flash device. The call is used as a catch-all for operations. The type of arg depends upon the particular control request. All operations are blocked, which means that they wait for the completion of the operation, check the Status Register and clear it in the case of an error. A summary of the existing parameters of EFD_Ioctl() can be found in the "IOCTL Operations" table.

Prototype

```
FLASH_ERROR EFD_Ioctl(
        void                    *fdo,
        FLASH_COMMAND                   cmd,
        FLASH_PARAMETER                   *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to an internal object that describes the flash device and tores a set of low-level operations.
cmd	(IN) Flash command. (For a detailed description, see the item commands for IOCTL operations in the section "Structures and types").
arg	(IN/OUT) Parameter value. (For a detailed description, see the section "Structures and types".)

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter was detected.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure
OTHER ERROR CODES	Other error codes are listed in the "IOCTL Operations" table.	Failure

EFD_Ioctl Function (FLASH_CMD_ID)

The EFD_Ioctl() with FLASH_CMD_ID command is used to retrieve the device identification that is unique among the flash devices of one vendor. After completion, the device is returned to the Read Array Mode. The parameter arg should not be null. The result is retrieved inside arg.FlashId.

Prototype

```
FLASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
cmd	(IN) FLASH_CMD_ID flash command.

Parameter	Description
arg	(OUT) The FlashId parameter value. For a detailed description, see Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter was detected.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure

EFD_Ioctl Function (FLASH_CMD_CFI)

The EFD_Ioctl() with FLASH_CMD_CFI command allows the user to read information from the CFI area. After completion, the device is returned to the Read Array Mode. The parameter arg should be a type Cfi. Cfi.Offset should direct a place in the CFI table for reading. Cfi.Length sets the number of bytes to read. Cfi.Data should be allocated for Cfi.Length size. Read data is placed in Cfi.Data.

Prototype

```
FLASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
cmd	(IN) The FLASH_CMD_CFI flash command.
arg	(IN/OUT) The Cfi parameter value. For a detailed description, see Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter was detected.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure

EFD_Ioctl Function (FLASH_CMD_CFI_EX)

The EFD_Ioctl() with FLASH_CMD_CFI_EX command allows the user to read one, two or four bytes of information from the CFI area by the predefined codes (see the section "Structures and types"). After completion, the device is returned to the Read Array Mode. The parameter arg should be a type CfiEx. CfiEx.Code should be one of the predefined offsets. Read data is placed in Cfi.Data.

Prototype

```
FLASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
cmd	(IN) The FLASH_CMD_CF_EX flash command.
arg	(IN/OUT) The CfiEx parameter value. For a detailed description, see Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter was detected	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure

EFD_Ioctl Function (FLASH_CMD_READ)

The EFD_Ioctl() with FLASH_CMD_READ command allows the user to read one WORD from a flash device by the set offset. The parameter arg should have a type Read. Read.Addr should be aligned with WORD and be inside the flash memory boundary. The result is stored in Read.Value.

Prototype

```
FLASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
cmd	(IN) The FLASH_CMD_READ flash command.
arg	(IN/OUT) The read parameter value. For a detailed description, see Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter was detected.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure
FLASH_ERR_BAD_ADDRESS	An address is out of the flash boundary or unaligned with a WORD.	Failure

EFD_Ioctl Function (FLASH_CMD_WRITE)

The EFD_Ioctl() with FLASH_CMD_WRITE command allows the user to write one WORD to the flash device by the set offset. After completion, the device is returned to the Read Array Mode. The parameter arg should have a type Write. Write.Addr should be aligned with a WORD and be inside the flash memory boundary.

Prototype

```
FLASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
cmd	(IN) The FLASH_CMD_WRITE flash command.
arg	(IN) The Write parameter value. For a detailed description, see Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter was detected.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure
FLASH_ERR_BAD_ADDRESS	An address is out of the flash boundary or unaligned with a WORD.	Failure
FLASH_ERR_TIMEOUT	An error due to a time-out lapse.	Failure
FLASH_ERR_VPP	A voltage range error.	Failure
FLASH_ERR_PROGRAM	An operation programming failure.	Failure
FLASH_ERR_LOCKED	An operation was performed in a protected area.	Failure

EFD_Ioctl Function (FLASH_CMD_BUFFERED_WRITE)

The EFD_Ioctl() with FLASH_CMD_BUFFERED_WRITE command allows the user to write a chunk of bytes by the set offset in one erase block. After completion, the device is returned to the Read Array Mode. The parameter arg should have a type BufWrite. BufWrite.Offset should be aligned with a WORD and must be inside the flash memory boundary. BufWrite.Length should be divisible by a WORD. BufWrite.Data should be allocated for the BufWrite.Length size.

Prototype

```
FLASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
cmd	(IN) The FLASH_CMD_BUFFERED_WRITE flash command.
arg	(IN) The BufWrite parameter value. For a detailed description, see Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter was detected.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure
FLASH_ERR_BAD_ADDRESS	An address is out of the flash boundary or unaligned with a WORD.	Failure
FLASH_ERR_TIMEOUT	An error due to a time-out lapse.	Failure
FLASH_ERR_VPP	A voltage range error.	Failure
FLASH_ERR_SEQUENCE	A sequence program error.	Failure
FLASH_ERR_PROGRAM	An operation programming failure.	Failure
FLASH_ERR_LOCKED	An operation was performed in a protected area.	Failure

EFD_Ioctl Function (FLASH_CMD_SUSPEND)

The EFD_Ioctl() with FLASH_CMD_SUSPEND command temporarily pauses the ongoing Program operations and read operations are allowed on the rest of the device. This allows the user to access the information stored in the device immediately, rather than waiting until the Program operation is completed. After completion, the device is returned to the Read Array Mode. The parameter arg should have a type ProgramOp. ProgramOp.Addr can be unaligned with the flash bus width and must be inside the flash memory boundary.

Prototype

```
FLASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
cmd	(IN) The FLASH_CMD_SUSPEND flash command.
arg	(IN) The ProgramOp parameter value. For a detailed description, see Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter was detected.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure
FLASH_ERR_BAD_ADDRESS	An address is out of the flash boundary.	Failure
FLASH_ERR_TIMEOUT	An error due to a time-out lapse.	Failure
FLASH_ERR_SUSPEND	The erase/program operation completed.	Failure

EFD_Ioctl Function (FLASH_CMD_RESUME)

The EFD_Ioctl() with FLASH_CMD_RESUME instructs the device to continue programming operation, and automatically clears Status Register bits SR[7,2]. This command can be written to any address. If error bits are set, the Status Register should be cleared before issuing the next instruction. Parameter arg should have a type ProgramOp. ProgramOp.Addr can be unaligned with the flash bus width and must be inside the flash memory boundary.

Prototype

```
LASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
cmd	(IN) The FLASH_CMD_RESUME flash command.
arg	(IN) The ProgramOp parameter value. For a detailed description, see Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter is detected.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure
FLASH_ERR_BAD_ADDRESS	An address is out of the flash boundary.	Failure

EFD_Ioctl Function (FLASH_CMD_BLOCK_SUSPEND)

The EFD_Ioctl() with FLASH_CMD_BLOCK_SUSPEND gives the Erase Suspend command while erasing suspends the block erase operation. This allows data to be accessed from memory locations other than the one being erased. The Erase Suspend command can be given to any device address. A block erase operation can be suspended to perform a WORD or buffer program operation, or a read operation within any block except the block that is erase suspended. After completion, the device is returned to the Read

Array Mode. The parameter arg should have a type BlockOp. BlockOp.Addr can be unaligned with the flash bus width and must be inside the flash memory boundary.

Prototype

```
FLASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
cmd	(IN) The FLASH_CMD_BLOCK_SUSPEND flash command.
arg	(IN) The BlockOp parameter value. For a detailed description, see Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter was detected.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure
FLASH_ERR_BAD_ADDRESS	An address is out of the flash boundary.	Failure
FLASH_ERR_TIMEOUT	An error due to a time-out lapse.	Failure
FLASH_ERR_SUSPEND	The erase/program operation completed.	Failure

EFD_Ioctl Function (FLASH_CMD_BLOCK_RESUME)

The EFD_Ioctl() with FLASH_CMD_BLOCK_RESUME instructs the device to continue erasing, and automatically clears status register bits SR[7,6]. This command can be written to any address. If status register error bits are set, the Status Register should be cleared before issuing the next instruction. The parameter arg should have a type BlockOp. BlockOp.Addr can be unaligned with the flash bus width and must be inside the flash memory boundary.

Prototype

```
FLASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
cmd	(IN) The FLASH_CMD_BLOCK_RESUME flash command.

Parameter	Description
arg	(IN) The BlockOp parameter value. For a detailed description, see Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter was detected.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure
FLASH_ERR_BAD_ADDRESS	An address out of the flash boundary.	Failure

EFD_Ioctl Function (FLASH_CMD_BLOCK_ERASE)

The EFD_Ioctl() with FLASH_CMD_BLOCK_ERASE initiates an erase operation for the block directed to by the address. This command can be written to any address inside a block. After completion, the device is returned to Read Array Mode. The parameter arg should have a type BlockOp. BlockOp.Addr can be unaligned with the flash bus width and must be inside the flash memory boundary.

Prototype

```
FLASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
cmd	(IN) The FLASH_CMD_BLOCK_ERASE flash command.
arg	(IN) The BlockOp parameter value. For a detailed description, see Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter was detected.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure
FLASH_ERR_BAD_ADDRESS	An address not valid for this device.	Failure
FLASH_ERR_TIMEOUT	An error due to a time-out lapse.	Failure
FLASH_ERR_VPP	A voltage range error.	Failure
FLASH_ERR_SEQUENCE	A sequence program error.	Failure
FLASH_ERR_ERASE	On operation erasing failure.	Failure
FLASH_ERR_LOCKED	An operation was performed in a protected area.	Failure

EFD_Ioctl Function(FLASH_CMD_BLOCK_STATE)

The EFD_Ioctl() with FLASH_CMD_BLOCK_STATE is used to determine a block's lock status. This command can be written to any address inside the block. After completion, the device is returned to Read Array Mode. The parameter arg should have a type BlockOp. BlockOp.Addr can be unaligned with the flash bus width and must be inside the flash memory boundary. The result is stored in BlockOp.State. Refer to the datasheet for the flash memory device for a description of possible states.

Prototype

```
FLASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
cmd	(IN) The FLASH_CMD_BLOCK_STATE flash command.
arg	(IN/OUT) The BlockOp parameter value. For a detailed description, see Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter was detected.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure
FLASH_ERR_BAD_ADDRESS	An address is not valid for this device.	Failure

EFD_Ioctl Function (FLASH_CMD_BLOCK_UNLOCK)

The EFD_Ioctl() with FLASH_CMD_BLOCK_UNLOCK is used to unlock the blocks. The unlocked blocks can be read, programmed and erased. The unlocked blocks return to a locked state when the device is reset or powered down. This command can be written to any address inside a block. After completion, the device is returned to the Read Array Mode. The parameter arg should have a type BlockOp. BlockOp.Addr can be unaligned with the flash bus width and must be inside the flash memory boundary.

Prototype

```
LASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores set of low-level operations.
cmd	(IN) The FLASH_CMD_BLOCK_UNLOCK flash command.

Parameter	Description
arg	(IN) The BlockOp parameter value. For a detailed description, see Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter is detected.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure
FLASH_ERR_BAD_ADDRESS	An address is not valid for this device.	Failure
FLASH_ERR_TIMEOUT	An error to a due time-out lapse.	Failure

EFD_Ioctl Function (FLASH_CMD_BLOCK_LOCKDOWN)

The EFD_Ioctl() with FLASH_CMD_BLOCK_LOCKDOWN is used to lock down blocks. The blocks in a lock-down state cannot be programmed or erased. They can only be read. Unlike locked blocks, their locked state cannot be changed by software commands only. This command can be written to any address inside a block. After completion, the device is returned to the Read Array Mode. The parameter arg should have a type BlockOp. BlockOp.Addr can be unaligned with the flash bus width and must be inside the flash memory boundary.

Prototype

```
FLASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
cmd	(IN) The FLASH_CMD_BLOCK_LOCKDOWN flash command.
arg	(IN) The BlockOp parameter value. For a detailed description, see Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter is detected.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure
FLASH_ERR_BAD_ADDRESS	An address not valid for this device.	Failure

Value	Description	Success/Failure
FLASH_ERR_TIMEOUT	An error due to a time-out lapse.	Failure

EFD_Ioctl Function (FLASH_CMD_BLOCK_LOCK)

The EFD_Ioctl() with FLASH_CMD_BLOCK_LOCK is used to lock blocks. The locked blocks can be read, programmed and erased. This command can be written to any address inside a block. After completion, the device is returned to the Read Array Mode. The parameter arg should have a type BlockOp. BlockOp.Addr can be unaligned with the flash bus width and must be inside the flash memory boundary.

Prototype

```
FLASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
cmd	(IN) The FLASH_CMD_BLOCK_LOCK flash command.
arg	(IN) The BlockOp parameter value. For a detailed description, see Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter was detected.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure
FLASH_ERR_BAD_ADDRESS	An address not valid for this device.	Failure
FLASH_ERR_TIMEOUT	An error due to a time-out lapse.	Failure

EFD_Ioctl Function (FLASH_CMD_READ_STATUS)

The EFD_Ioctl() with FLASH_CMD_READ_STATUS is used to retrieve information about the error status of the flash device. Status register bits present the status and error information about the program, erase, suspend, VPP and blocklocked operations. The parameter arg should have a type Read. Read.Addr should be unaligned with a WORD and must be inside the flash memory boundary. Read Status Register information is placed in Read.Value.

Prototype

```
FLASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
cmd	(IN) The FLASH_CMD_READ_STATUS flash command.
arg	(OUT) The status parameter value. For a detailed description, see Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter was detected.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure
FLASH_ERR_BAD_ADDRESS	An address not valid for this device.	Failure

EFD_Ioctl Function (FLASH_CMD_CLEAR_STATUS)

The EFD_Ioctl() with FLASH_CMD_CLEAR_STATUS clears the Status Register.

Prototype

```
FLASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
cmd	(IN) The FLASH_CMD_CLEAR_STATUS flash command.
arg	The ProgramOp parameter value. For a detailed description, see Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure

EFD_Ioctl Function (FLASH_CMD_READ_PROTECTION_REG)

The EFD_Ioctl() with FLASH_CMD_READ_PROTECTION_REG command allows the user to read one WORD from the flash device Protection Register area. After completion, the device is returned to the Read Array Mode. The parameter arg should have a type Read. Read.Addr should be aligned with a WORD and must be inside the flash memory boundary. The result is stored in Read.Value.

Prototype

```
FLASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
cmd	(IN) The FLASH_CMD_READ_PROTECTION_REG flash command.
arg	(IN/OUT) The read parameter value. For a detailed description, see Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter was detected.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure

EFD_Ioctl Function (FLASH_CMD_WRITE_CONFIGURATION_REG)

The EFD_Ioctl() with FLASH_CMD_WRITE_CONFIGURATION_REG command allows the user to write to the 16-bit Read Configuration Register (RCR) (for detailed information, refer to the datasheet for the flash device). After completion, the device is returned to the Read Array Mode. The parameter arg should have a type Write. Write.Addr should be aligned with a WORD.

Prototype

```
LASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
cmd	(IN) The FLASH_CMD_WRITE_CONFIGURATION_REG flash command.
arg	(IN) The Write parameter value. For a detailed description, see Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter was detected.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure

EFD_Ioctl Function (FLASH_CMD_READ_CONFIGURATION_REG)

The EFD_Ioctl() with FLASH_CMD_READ_CONFIGURATION_REG command allows the user to read the 16-bit Read Configuration Register (RCR) (for detailed information,

refer to the datasheet for the flash device). After completion, the device is returned to the Read Array Mode. The parameter arg should have a type Read. Read.Addr should be aligned with a WORD and must be inside the flash memory boundary. The result is stored in Read.Value.

Prototype

```
FLASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
cmd	(IN) The FLASH_CMD_READ_CONFIGURATION_REG flash command.
arg	(IN/OUT) The read parameter value. For a detailed description, see Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter was detected.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure

EFD_Ioctl Function (FLASH_CMD_WRITE_CONFIGURATION_REG)

The EFD_Ioctl() with FLASH_CMD_WRITE_CONFIGURATION_REG command allows the user to write to the 16-bit Read Configuration Register (RCR) (for detailed information, refer to the datasheet for the flash device). After completion, the device is returned to the Read Array Mode. The parameter arg should have a type Write. Write.Addr should be aligned with a WORD.

Prototype

```
LASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
cmd	(IN) The FLASH_CMD_WRITE_CONFIGURATION_REG flash command.
arg	(IN) The Write parameter value. For a detailed description, see Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter was detected.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure

EFD_Ioctl Function (FLASH_CMD_ALTERABLE_WRITE)

The EFD_Ioctl() with FLASH_CMD_ALTERABLE_WRITE command allows the user to write directly into the PCM memory, unlike flash memory, which can only go from 1 to 0 before an erase of the entire block by the set offset. After completion, the device is returned to the Read Array Mode. The parameter arg should have a type Write. Write.Addr should be aligned with a WORD and be inside the flash memory boundary.

Prototype

```
FLASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores set of low-level operations.
cmd	(IN) The FLASH_CMD_WRITE flash command.
arg	(IN) The Write parameter value. For a detailed description, see Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter was detected.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure
FLASH_ERR_BAD_ADDRESS	An address is out of the flash boundary or unaligned with a WORD.	Failure
FLASH_ERR_TIMEOUT	An error due to a time-out lapse.	Failure
FLASH_ERR_VPP	A voltage range error.	Failure
FLASH_ERR_PROGRAM	An operation programming failure.	Failure
FLASH_ERR_LOCKED	An operation was performed in a protected area.	Failure

EFD_Ioctl Function (FLASH_CMD_ALTERABLE_BUFFERED_WRITE)

The EFD_Ioctl() with FLASH_CMD_ALTERABLE_BUFFERED_WRITE command allows the user to write directly into the PCM memory, unlike flash memory, which can only go from 1 to 0 before an erase of the entire block, a chunk of bytes by set offset in one erase block. After completion, the device is returned to the Read Array Mode. The parameter arg should have a type BufWrite. BufWrite.Offset should be aligned with a WORD and

must be inside the flash memory boundary. BufWrite.Length should be divisible by a WORD. BufWrite.Data should be allocated for the BufWrite.Length size.

Prototype

```
FLASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
cmd	(IN) The FLASH_CMD_ALTERABLE_BUFFERED_WRITE flash command.
arg	(IN) The BufWrite parameter value. For a detailed description, see the Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter was detected.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure
FLASH_ERR_BAD_ADDRESS	An address is out of the flash boundary or unaligned with a WORD.	Failure
FLASH_ERR_TIMEOUT	An error due to a time-out lapse.	Failure
FLASH_ERR_VPP	A voltage range error.	Failure
FLASH_ERR_SEQUENCE	A sequence program error.	Failure
FLASH_ERR_PROGRAM	An operation programming failure.	Failure
FLASH_ERR_LOCKED	An operation was performed in a protected area.	Failure

EFD_Ioctl Function (FLASH_CMD_STREAM_ENTRY)

The EFD_Ioctl() with FLASH_CMD_STREAM_ENTRY operation initiates a microcode algorithm to only turn on the high voltage program pumps for an addressed Program Region (a Program Region size is defined as a 16 Mbit region of PCM memory cells). After completion, the device is returned to the Read Array Mode. The parameter arg should have a type ProgramOp. ProgramOp.Addr can be unaligned with the flash bus width and must be inside the flash memory boundary.

Prototype

```
FLASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
cmd	(IN) The FLASH_CMD_SUSPEND flash command.
arg	(IN) The ProgramOp parameter value. For a detailed description, see Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter was detected.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure
FLASH_ERR_BAD_ADDRESS	An address is out of the flash boundary.	Failure
FLASH_ERR_TIMEOUT	An error due to a time-out lapse.	Failure
FLASH_ERR_PROGRAM	An operation programming failure.	Failure

EFD_Ioctl Function (FLASH_CMD_STREAM_EXIT)

The EFD_Ioctl() with FLASH_CMD_STREAM_EXIT operation initiates a microcode algorithm to only power down the high voltage program pumps for an addressed Program Region (a Program Region size is defined as a 16 Mbit region of PCM memory cells). After completion, the device is returned to the Read Array Mode. The parameter arg should have a type ProgramOp. ProgramOp.Addr can be unaligned with the flash bus width and must be inside the flash memory boundary.

Prototype

```
FLASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
cmd	(IN) The FLASH_CMD_SUSPEND flash command.
arg	(IN) The ProgramOp parameter value. For a detailed description, see Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter was detected.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure
FLASH_ERR_BAD_ADDRESS	An address is out of the flash boundary.	Failure
FLASH_ERR_TIMEOUT	An error due to a time-out lapse.	Failure

Value	Description	Success/Failure
FLASH_ERR_PROGRAM	An operation programming failure.	Failure

EFD_Ioctl Function (FLASH_CMD_BLANK_CHECK)

The EFD_Ioctl() with FLASH_CMD_BLANK_CHECK operation is used to confirm whether or not a main-array block is completely erased. A Blank Check operation is performed one block at a time, and cannot be used during Program Suspend or Erase Suspend. After completion, the device is returned to the Read Array Mode. The parameter arg should have a type BlockOp. BlockOp.Addr can be unaligned with the flash bus width and must be inside the flash memory boundary.

Prototype

```
FLASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
cmd	(IN) The FLASH_CMD_BLANK_CHECK flash command.
arg	(IN) The BlockOp parameter value. For a detailed description, see Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter was detected.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure
FLASH_ERR_BAD_ADDRESS	An address not valid for this device.	Failure
FLASH_ERR_TIMEOUT	An error due to a time-out lapse.	Failure
FLASH_ERR_VPP	A voltage range error.	Failure
FLASH_ERR_SEQUENCE	A sequence program error.	Failure
FLASH_ERR_ERASE	An operation erasing failure.	Failure

EFD_Ioctl Function (FLASH_CMD_EFI)

The EFD_Ioctl() with FLASH_CMD_EFI command allows the user to program the Extended Function Interface command. After completion, the device is returned to the Read Array Mode. The parameter arg should have a type Efi. Efi.Addr should be aligned with a WORD and must be inside the flash memory boundary. Efi.Length should be divisible by 2*sizeof(UINT32). Efi.Data should be allocated for the Efi.Length size and filled by the address-data pair. Address and data should be UINT32 sizes.

Prototype

```
FLASH_ERROR EFD_Ioctl( void *fdo,
                                    FLASH_COMMAND cmd,
                                    FLASH_PARAMETER *arg);
```

Parameters

Parameter	Description
fdo	(IN) Pointer to the internal object that describes the flash device and stores a set of low-level operations.
cmd	(IN) The FLASH_CMD_EFI flash command.
arg	(IN) The Efi parameter value. For a detailed description, see Structures and Types (page 11).

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_BAD_ARGUMENT	An invalid parameter was detected.	Failure
FLASH_ERR_UNSUPPORTED	An operation that is needed for the function execution was turned off.	Failure
FLASH_ERR_BAD_ADDRESS	An address is out of the flash boundary or unaligned with a WORD.	Failure
FLASH_ERR_TIMEOUT	An error due time-out lapse.	Failure
FLASH_ERR_VPP	A voltage range error.	Failure
FLASH_ERR_SEQUENCE	A sequence program error.	Failure
FLASH_ERR_PROGRAM	An operation programming failure.	Failure
FLASH_ERR_LOCKED	An operation was performed in a protected area.	Failure

Table 3: IOCTL Operations

Command	Parameter	In	Out	Description
FLASH_CMD_ID	FlashId	-	FlashId	FLASH_ERR_NONE FLASH_ERR_UNSUPPORTED FLASH_ERR_BAD_ARGUMENT
FLASH_CMD_CFI	Cfi	Length Offset	Data	FLASH_ERR_NONE FLASH_ERR_UNSUPPORTED FLASH_ERR_BAD_ARGUMENT
FLASH_CMD_CFI_EX	CfiEx	Code	Data	FLASH_ERR_NONE FLASH_ERR_UNSUPPORTED FLASH_ERR_BAD_ARGUMENT
FLASH_CMD_READ	Read	Addr	Value	FLASH_ERR_NONE FLASH_ERR_UNSUPPORTED FLASH_ERR_BAD_ARGUMENT FLASH_ERR_BAD_ADDRESS

Table 3: IOCTL Operations (Continued)

Command	Parameter	In	Out	Description
FLASH_CMD_WRITE	Write	Addr Value	-	FLASH_ERR_NONE FLASH_ERR_UNSUPPORTED FLASH_ERR_BAD_ARGUMENT FLASH_ERR_BAD_ADDRESS FLASH_ERR_TIMEOUT FLASH_ERR_VPP FLASH_ERR_PROGRAM FLASH_ERR_LOCKED
FLASH_CMD_BUF-FERED_WRITE	BufWrite	Length Offset Data	-	FLASH_ERR_NONE FLASH_ERR_UNSUPPORTED FLASH_ERR_BAD_ARGUMENT FLASH_ERR_BAD_ADDRESS FLASH_ERR_TIMEOUT FLASH_ERR_VPP FLASH_ERR_SEQUENCE FLASH_ERR_PROGRAM FLASH_ERR_LOCKED
FLASH_CMD_SUSPEND	ProgramOp	Addr	-	FLASH_ERR_NONE FLASH_ERR_UNSUPPORTED FLASH_ERR_BAD_ARGUMENT FLASH_ERR_BAD_ADDRESS FLASH_ERR_TIMEOUT FLASH_ERR_SUSPEND
FLASH_CMD_RESUME	ProgramOp	Addr	-	FLASH_ERR_NONE FLASH_ERR_UNSUPPORTED FLASH_ERR_BAD_ARGUMENT FLASH_ERR_BAD_ADDRESS
FLASH_CMD_BLOCK_S USPEND	BlockOp	Addr	-	FLASH_ERR_NONE FLASH_ERR_UNSUPPORTED FLASH_ERR_BAD_ARGUMENT FLASH_ERR_BAD_ADDRESS FLASH_ERR_TIMEOUT FLASH_ERR_SUSPEND

Table 3: IOCTL Operations (Continued)

Command	Parameter	In	Out	Description
FLASH_CMD_BLOCK_RESUME	BlockOp	Addr	-	FLASH_ERR_NONE FLASH_ERR_UNSUPPORTED FLASH_ERR_BAD_ARGUMENT FLASH_ERR_BAD_ADDRESS
FLASH_CMD_BLOCK_ERASE	BlockOp	Addr	-	FLASH_ERR_NONE FLASH_ERR_UNSUPPORTED FLASH_ERR_BAD_ARGUMENT FLASH_ERR_BAD_ADDRESS FLASH_ERR_TIMEOUT FLASH_ERR_VPP FLASH_ERR_SEQUENCE FLASH_ERR_ERASE FLASH_ERR_LOCKED
FLASH_CMD_BLOCK_STATE	BlockState	Addr	State	FLASH_ERR_NONE FLASH_ERR_UNSUPPORTED FLASH_ERR_BAD_ARGUMENT FLASH_ERR_BAD_ADDRESS
FLASH_CMD_BLOCK_UNLOCK	BlockOp	Addr	-	FLASH_ERR_NONE FLASH_ERR_UNSUPPORTED FLASH_ERR_BAD_ARGUMENT FLASH_ERR_BAD_ADDRESS FLASH_ERR_TIMEOUT
FLASH_CMD_BLOCK_LOCKDOWN	BlockOp	Addr	-	FLASH_ERR_NONE FLASH_ERR_UNSUPPORTED FLASH_ERR_BAD_ARGUMENT FLASH_ERR_BAD_ADDRESS
FLASH_CMD_BLOCK_LOCK	BlockOp	Addr	-	FLASH_ERR_NONE FLASH_ERR_UNSUPPORTED FLASH_ERR_BAD_ARGUMENT FLASH_ERR_BAD_ADDRESS FLASH_ERR_TIMEOUT

Table 3: IOCTL Operations (Continued)

Command	Parameter	In	Out	Description
FLASH_CMD_READ_STATUS	Read	Addr	Value	FLASH_ERR_NONE FLASH_ERR_UNSUPPORTED FLASH_ERR_BAD_ARGUMENT FLASH_ERR_BAD_ADDRESS
FLASH_CMD_CLEAR_STATUS	ProgramOp	Addr	-	FLASH_ERR_NONE FLASH_ERR_UNSUPPORTED
FLASH_CMD_READ_PROTECTION_REG	Read	Addr	Value	FLASH_ERR_NONE FLASH_ERR_UNSUPPORTED FLASH_ERR_BAD_ARGUMENT
FLASH_CMD_WRITE_PROTECTION_REG	Write	Addr Value	-	FLASH_ERR_NONE FLASH_ERR_UNSUPPORTED FLASH_ERR_BAD_ARGUMENT FLASH_ERR_TIMEOUT FLASH_ERR_VPP FLASH_ERR_PROGRAM FLASH_ERR_LOCKED
FLASH_CMD_READ_CONFIGURATION_REG	Read	Addr	Value	FLASH_ERR_NONE FLASH_ERR_UNSUPPORTED FLASH_ERR_BAD_ARGUMENT
FLASH_CMD_WRITE_CONFIGURATION_REG	Write	Addr Value	-	FLASH_ERR_NONE FLASH_ERR_UNSUPPORTED FLASH_ERR_BAD_ARGUMENT
FLASH_CMD_ALTERABLE_WRITE	Write	Addr Value	-	FLASH_ERR_NONE FLASH_ERR_UNSUPPORTED FLASH_ERR_BAD_ARGUMENT FLASH_ERR_BAD_ADDRESS FLASH_ERR_TIMEOUT FLASH_ERR_VPP FLASH_ERR_PROGRAM FLASH_ERR_LOCKED

Table 3: IOCTL Operations (Continued)

Command	Parameter	In	Out	Description
FLASH_CMD_ALTERA-BLE_BUFFERED_WRITE	BufWrite	Length Offset Data	-	FLASH_ERR_NONE FLASH_ERR_UNSUPPORTED FLASH_ERR_BAD_ARGUMENT FLASH_ERR_BAD_ADDRESS FLASH_ERR_TIMEOUT FLASH_ERR_VPP FLASH_ERR_SEQUENCE FLASH_ERR_PROGRAM FLASH_ERR_LOCKED
FLASH_CMD_STREAM_ENTRY	BlockOp	Addr	-	FLASH_ERR_NONE FLASH_ERR_UNSUPPORTED FLASH_ERR_BAD_ARGUMENT FLASH_ERR_BAD_ADDRESS FLASH_ERR_TIMEOUT FLASH_ERR_PROGRAM
FLASH_CMD_STREAM_EXIT	BlockOp	Addr	-	FLASH_ERR_NONE FLASH_ERR_UNSUPPORTED FLASH_ERR_BAD_ARGUMENT FLASH_ERR_BAD_ADDRESS FLASH_ERR_TIMEOUT FLASH_ERR_PROGRAM
FLASH_CMD_BLANK_CHECK	BlockOp	Addr	-	FLASH_ERR_NONE FLASH_ERR_UNSUPPORTED FLASH_ERR_BAD_ARGUMENT FLASH_ERR_BAD_ADDRESS FLASH_ERR_TIMEOUT FLASH_ERR_VPP FLASH_ERR_SEQUENCE FLASH_ERR_ERASE

Table 3: IOCTL Operations (Continued)

Command	Parameter	In	Out	Description
FLASH_CMD_EFI	Efi	OpCode	-	FLASH_ERR_NONE
		Addr Length		FLASH_ERR_UNSUPPORTED
		Data		FLASH_ERR_BAD_ARGUMENT
				FLASH_ERR_BAD_ADDRESS
				FLASH_ERR_TIMEOUT
				FLASH_ERR_VPP
				FLASH_ERR_SEQUENCE
				FLASH_ERR_PROGRAM
				FLASH_ERR_LOCKED

For more information, refer to the EFD_Ioctl API function usage example in the section "IOCTL method usage".

Low-level Flash Device Driver

The low-level driver code implements procedures to program the basic operations described in the datasheets for Micron flash devices. Most operations are available via two standard bus operations: read and write. Read operations retrieve data or status information from the device. Write operations are interpreted by the device as commands that modify the data stored or the device's behavior. Only certain special write operation sequences are recognized as commands by the flash devices. The various commands recognized by the devices are listed in the Commands Tables provided in the corresponding datasheets. All low-level functions are blocked. They wait for the completion of an operation, then check the Status Register and clear it if an error occurs. There is no check in the code to test whether or not the address given as a function argument matches the flash memory location in the address space. It is up to user to implement this if required to avoid unpredictable behavior if the provided address is out of the flash memory address range. The main commands can be classified as described in the following sections.

Read

The Read command returns the flash devices to the Read Mode, where the devices behave as ROMs. In this state, a read operation outputs the data stored at the specified device address onto the data bus.

Program

The Program command is used to modify the data stored at the specified device address. Programming can only change bits from “1” to “0”. If an attempt is made to change a bit from “0” to “1” using the Program command, the command is executed and no error is signaled. However, the bit remains unchanged. It may be necessary to erase the block before programming to addresses within it. Programming modifies a single Word at a time. Programming larger amounts of data must be done one Word at a time by giving a Program command, waiting for the command to complete, giving the next Program command, and so on.

Erase

Flash erase is performed on a block basis. An entire block is erased each time an erase command sequence is given. Only one block is erased at a time. When a block is erased, all bits within that block are read as the logical ones.

Read Common Flash Interface Query

The Read Common Flash Interface Query command allows the user to identify the number of blocks in the flash memory and the block addresses. The interface contains information relating to the typical and maximum Program and Erase times. This allows the user to implement software timeouts and prevents waiting for a defective flash memory device to finish programming or erasing. For additional information about the CFI, please refer to the CFI specification available at http://www.jedec.org.

Block Lock, Block Unlock and Block Lock-Down

Individual instant block locking is used to protect user code and/or data within the flash memory array. All blocks power up in a locked state to protect array data from being altered during power transitions. Locked blocks cannot be programmed or erased. They can only be read. Blocks in a lock-down state cannot be programmed or erased;

they can only be read. However, unlike locked blocks, their locked state cannot be changed by software commands alone. Locked-down blocks revert to the locked state upon reset or when powering up the device.

Status Register

During program or erase operations, a bus read operation outputs the contents of the Status Register. The Status Register, which can also be accessed by issuing the Read Status Register command, provides information about the latest program or erase operation. The Status Register bits are described in the Status Register Bits Tables provided in the datasheet. They are used to determine when programming or erase is complete and whether or not the operation was successful. When an error occurs, the corresponding Status Register bits are set and do not automatically return to “0” when a new command is given. It is consequently essential to clear any error bits in the Status Register before attempting a new Program, Erase or Resume command. This is done by issuing a Clear Status Register command.

Low-level Functions Provided (CFI COMMANDSET)

Whenever any function expects an address offset as an argument or return address as a value, this address is a linear byte address of the target application (microprocessor) byte address space represented by a WORD (8-bit, 16-bit, 32-bit, etc. reference to a specification to a flash memory device) within the flash memory. This address offset can be unaligned by a WORD if it is not specified otherwise. The start (base) address of the flash memory in this address space is defined in the efd_conf.h file as BASE_ADDR, and users may need to change it.

- **FLASH_ERROR DeviceId(FLASH_ID *)**

  This function is used to retrieve information about the Flash Device type.
- **FLASH_ERROR ReadCfi(UINT32, UINT32, UINT8 *)**

  This function is used to read the specified amount of CFI data from the specified offset. After completion, the device is returned to the Read Array Mode.
- **FLASH_ERROR ReadCfiEx(FLASH_CFI, UINT32 *)**

  This function is used to read the CFI data from the predefined code (for more information, see the section Structures and types"). After completion, the device is returned to the Read Array Mode.
- **FLASH_ERROR ReadFlash(UINT32)**

  This function is used to read a single WORD from a flash device. Addr_offset MUST be properly aligned with the flash bus width. No parameter checking is performed.
- **FLASH_ERROR WriteFlash(UINT32, UINT32)**

  This function is used to program a single WORD in a flash device. Program suspend does not work with this function, since this function does not return until the process is finished. Addr_offset MUST be properly aligned with the flash bus width. No parameter checking is performed. After completion, the device is returned to the Read Array Mode.
- **FLASH_ERROR BufferProgram(UINT32, UINT32, UINT8 *)**

  This function is used to program data arrays to the flash device. Program suspend does not work with this function, since this function does not return until the process is finished. Addr_offset MUST be properly aligned with the flash bus width. No pa-

rameter checking is performed. After completion, the device is returned to the Read Array Mode.

- **FLASH_ERROR ProgramSuspend(UINT32)**

  This function is used to suspend the program operation in progress. Giving the Program Suspend command during a Program operation temporarily pauses ongoing Program operations and read operations are allowed on the rest of the device. This allows the user to immediately access the information stored on the device rather than waiting until the Program operation is completed. The addr_offset can be unaligned with the flash bus width. After completion, the device is returned to the Read Array Mode.

- **FLASH_ERROR ProgramResume(UINT32)**

  This function is used to resume the program operation being suspended. The addr_offset can be unaligned with the flash bus width.

- **FLASH_ERROR EraseSuspend(UINT32)**

  This function is used to suspend the erase operation in progress. Issuing the Erase Suspend command during a Erase operation temporarily pauses ongoing Erase operations and the blocks not being erased may be read or programmed as in the default state of the device. This allows the user to immediately access the information stored in the device rather than waiting until the Erase operation is completed. After completion, the device is returned to the Read Array Mode. The addr_offset can be unaligned with the flash bus width.

- **FLASH_ERROR EraseResume(UINT32)**

  This function is used to resume the erase operation being suspended. The addr_offset can be unaligned with the Flash bus width.

- **FLASH_ERROR BlockErase(UINT32)**

  This function is used to erase one block in the device. A block cannot be erased when it is protected. Attempting to do so generates no error. Erase suspend does not work with this function, since this function does not return until the process is finished. After completion, the device is returned to the Read Array Mode. The addr_offset can be unaligned with the Flash bus width.

- **UINT32 BlockState(UINT32)**

  This function is used to check whether a block is locked-down, locked or unlocked. After completion, the device is returned to the Read Array Mode. The addr_offset should direct the Block Base Address.

- **FLASH_ERROR BlockUnlock(UINT32)**

  This function is used to unlock (unprotect) a block in the flash memory device. Once the block is unlocked, the data it contains can be erased or new data can be programmed to it. The addr_offset can be unaligned with the flash bus width. After completion, the device is returned to Read Array Mode.

- **FLASH_ERROR BlockLockDown(UINT32)**

  This function is used to lock-down a block. Once locked-down, the block is locked and the lock status of the block cannot be changed by using the software commands alone. The block will revert to the locked state when the device is reset or powered down. The addr_offset can be unaligned with the flash bus width. After completion, the device is returned to the Read Array Mode.

- **FLASH_ERROR BlockLock(UINT32)**

This function is used to lock (protect) a block in the flash memory device. Once locked (protected), the data in the block cannot be programmed or erased until the block is unlocked (unprotected). The addr_offset can be unaligned with the flash bus width. After completion, the device is returned to the Read Array Mode.

- **UINT32 ReadStatus(UINT32)**

  This function is used to read the Status Register.
- **void ClearStatus(UINT32)**

  This function is used to clear the Status Register.
- **void ReadMode(UINT32)**

  This function places the flash in the Read Array mode described in the datasheet. In this mode, the flash can be read as normal memory.
- **FLASH_ERROR ReadProtReg(UINT32, UINT32 *)**

  This function is used to read a location in the Protection Register. After completion, the device is returned to the Read Array Mode.
- **FLASH_ERROR WriteProtReg(UINT32, UINT32)**

  This function is used to program a Protection Register. After completion, the device is returned to the Read Array Mode.
- **FLASH_ERROR ReadConfReg(UINT32, UINT16 *)**

  This function is used to read the 16-bit Read Configuration Register (RCR) (for detailed information, see the related datasheet). After completion, the device is returned to the Read Array Mode.
- **FLASH_ERROR WriteConfReg(UINT32, UINT16)**

  This is used to program the 16-bit Read Configuration Register (RCR) (for detailed information, see the related datasheet). After completion, the device is returned to the Read Array Mode.
- **FLASH_ERROR BitAltWriteFlash(UINT32, UINT32)**

  This function is used to program a single WORD directly into the PCM memory, unlike flash memory, which can only go from 1 to 0 before an erase of the entire block. Program suspend does not work with this function, since this function does not return until the process is finished. Addr_offset MUST be properly aligned with the flash bus width. No parameter checking is performed. After completion, the device is returned to the Read Array Mode.
  Note: To Design Engineering, a block lock check must be done for any program command to prevent writing to a locked region. OUM technology does not require blocking structure. However, a security issue would occur if the Bit Alterable Programmed allowed a user to write to a locked region.
- **FLASH_ERROR BitAltBufferProgram(UINT32, UINT32, UINT8 *)**

  This function is used to program data arrays into the PCM memory, unlike the flash memory, which can only go from 1 to 0 before an erase of the entire block. Program suspend does not work with this function, since this function does not return until the process is finished. Addr_offset MUST be properly aligned with the flash bus width. No parameter checking is performed. After completion, the device is returned to the Read Array Mode.

  To Design Engineering, a block lock check must be done for any program command to prevent writing to a locked region. OUM technology does not require blocking

structure. However, a security issue would occur if the Bit Alterable Programmed allowed a user to write to a locked region.

- **FLASH_ERROR StreamModeEntry(UINT32)**

  This function is used to initiate a microcode algorithm to only turn on the high voltage program pumps for an addressed Program Region. After completion, the device is returned to the Read Array Mode. The addr_offset can be unaligned with the flash bus width.

- **FLASH_ERROR StreamModeExit(UINT32)**

  This function is used to initiate a microcode algorithm to power down the high voltage program pumps. If programming in another Program Region is required, then the Streaming Mode Exit operation must be used to switch the streaming mode to the other Program region. Otherwise, a program error will occur. After completion, the device is returned to the Read Array Mode. The addr_offset can be unaligned with the flash bus width.

- **FLASH_ERROR BlankCheck(UINT32)**

  This function is used to confirm whether or not a main-array block is completely erased. A Blank Check operation is performed one block at a time, and cannot be used during Program Suspend or Erase Suspend. After completion, the device is returned to the Read Array Mode. The addr_offset can be unaligned with the flash bus width.

- **FLASH_ERROR ProgramEfi(UINT16, UINT32, UINT32, UINT8 *)**

  This function is used to program the Extended Function Interface command. Addr_offset MUST be properly aligned with the flash bus width. A pointer to the buffer that will contain the address (UINT32)-data (UIN32) pair to program. No parameter checking is performed. After completion, the device is returned to the Read Array Mode.

Low-level Functions Across Flash Devices

Table 4: Functions Across Flash Devices

Functions	C3	J3D	P3x
DeviceId	EFD_GEN_DeviceId	EFD_GEN_DeviceId	EFD_GEN_DeviceId
ReadCfi	EFD_GEN_ReadCfi	EFD_GEN_ReadCfi	EFD_GEN_ReadCfi
ReadCfiEx	EFD_GEN_ReadCfiEx	EFD_GEN_ReadCfiEx	EFD_GEN_ReadCfiEx
ReadFlash	EFD_GEN_SingleRead	EFD_GEN_SingleRead	EFD_GEN_SingleRead
WriteFlash	EFD_GEN_SingleProgram	EFD_GEN_SingleProgram	EFD_GEN_SingleProgram
BufferProgram	-	EFD_GEN_BufferProgram	EFD_GEN_BufferProgram
ProgramSuspend	EFD_GEN_ProgramSuspend	EFD_GEN_ProgramSuspend	EFD_GEN_ProgramSuspend
ProgramResume	EFD_GEN_ProgramResume	EFD_GEN_ProgramResume	EFD_GEN_ProgramResume
EraseSuspend	EFD_GEN_EraseSuspend	EFD_GEN_EraseSuspend	EFD_GEN_EraseSuspend
EraseResume	EFD_GEN_EraseResume	EFD_GEN_EraseResume	EFD_GEN_EraseResume
BlockErase	EFD_GEN_BlockErase	EFD_GEN_BlockErase	EFD_GEN_BlockErase
BlockState	EFD_GEN_BlockState	EFD_GEN_BlockState	EFD_GEN_BlockState
BlockUnLock	EFD_GEN_BlockUnLock	EFD_GEN_BlockUnLock	EFD_GEN_BlockUnLock

Table 4: Functions Across Flash Devices (Continued)

Functions	C3	J3D	P3x
BlockLockDown	EFD_GEN_BlockLockDown	-	EFD_GEN_BlockLockDown
BlockLock	EFD_GEN_BlockLock	EFD_GEN_BlockLock	EFD_GEN_BlockLock
ReadStatus	EFD_GEN_ReadStatus	EFD_GEN_ReadStatus	EFD_GEN_ReadStatus
ClearStatus	EFD_GEN_ClearStatus	EFD_GEN_ClearStatus	EFD_GEN_ClearStatus
ReadMode	EFD_GEN_ReadMode	EFD_GEN_ReadMode	EFD_GEN_ReadMode
ReadProtReg	EFD_GEN_ReadProtReg	EFD_GEN_ReadProtReg	EFD_GEN_ReadProtReg
WriteProtReg	EFD_GEN_WriteProtReg	EFD_GEN_WriteProtReg	EFD_GEN_WriteProtReg
ReadConfReg	-	-	EFD_P3X_ReadConfReg
WriteConfReg	-	-	EFD_P3X_WriteConfReg
BitAltWriteFlash	-	-	-
BitAltBufferProgram	-	-	-
StreamModeEntry	-	-	-
StreamModeExit	-	-	-
BlankCheck	-	-	-
ProgramEfi	-	-	-

Functions	J3-65nm	P3x-65nm
DeviceId	EFD_GEN_DeviceId	EFD_GEN_DeviceId
ReadCfi	EFD_GEN_ReadCfi	EFD_GEN_ReadCfi
ReadCfiEx	EFD_GEN_ReadCfiEx	EFD_GEN_ReadCfiEx
ReadFlash	EFD_GEN_SingleRead	EFD_GEN_SingleRead
WriteFlash	EFD_GEN_SingleProgram	EFD_GEN_SingleProgram
BufferProgram	EFD_GEN_BufferProgram	EFD_GEN_BufferProgram
ProgramSuspend	EFD_GEN_ProgramSuspend	EFD_GEN_ProgramSuspend
ProgramResume	EFD_GEN_ProgramResume	EFD_GEN_ProgramResume
EraseSuspend	EFD_GEN_EraseSuspend	EFD_GEN_EraseSuspend
EraseResume	EFD_GEN_EraseResume	EFD_GEN_EraseResume
BlockErase	EFD_GEN_BlockErase	EFD_GEN_BlockErase
BlockState	EFD_GEN_BlockState	EFD_GEN_BlockState
BlockUnLock	EFD_GEN_BlockUnLock	EFD_GEN_BlockUnLock
BlockLockDown	-	EFD_GEN_BlockLockDown
BlockLock	EFD_GEN_BlockLock	EFD_GEN_BlockLock
ReadStatus	EFD_GEN_ReadStatus	EFD_GEN_ReadStatus
ClearStatus	EFD_GEN_ClearStatus	EFD_GEN_ClearStatus
ReadMode	EFD_GEN_ReadMode	EFD_GEN_ReadMode
ReadProtReg	EFD_GEN_ReadProtReg	EFD_GEN_ReadProtReg
WriteProtReg	EFD_GEN_WriteProtReg	EFD_GEN_WriteProtReg
ReadConfReg	-	EFD_P3X_ReadConfReg

Functions	J3-65nm	P3x-65nm
WriteConfReg	-	EFD_P3X_WriteConfReg
BitAltWriteFlash	-	-
BitAltBufferProgram	-	-
StreamModeEntry	-	-
StreamModeExit	-	-
BlankCheck	-	EFD_P3X_BlankCheck
ProgramEfi	EFD_P3X_ProgramEfi	EFD_P3X_ProgramEfi

Low-level Functions Provided (SPI NOR)

- **FLASH_ERROR DeviceId(FLASH_ID *)**

  This function is used to read three bytes of data describing the flash memory device.
- **UINT32 ReadStatus(UINT32)**

  This function is used to read the Status Register.
- **FLASH_ERROR WriteStatus(UINT32)**

  This function is used to allow the user to write to the writable Status Register bits.
- **FLASH_ERROR ClearStatus(void)**

  This function is used to reset bit SR5 (Erase Fail Flag) and bit SR6 (Program Fail Flag).
- **FLASH_ERROR WriteEnable(void)**

  This function is used to set the WEL bit.
- **FLASH_ERROR WriteDisable(void)**

  This function is used to clear the WEL bit.
- **FLASH_ERROR BulkErase(void)**

  This function is used to serially erase the entire main array, including the parameter blocks.
- **FLASH_ERROR SectorErase(UINT32)**

  This function is used to erase a memory sector.
- **FLASH_ERROR PageProgram(UINT32, UINT32, UINT8 *)**

  This function is used to program from 1 byte to 256 bytes of data within a 256-Byte-Aligned memory segment.
- **FLASH_ERROR ReadFlash(UINT32, UINT32, UINT8 *)**

  This function is used to read flash data.

Low-level Functions Across Flash Devices

Table 5: Functions Across Flash Devices

Function	S33
DeviceId	EFD_GEN_DeviceId
ReadStatus	EFD_GEN_ReadCfi
WriteStatus	EFD_GEN_ReadCfiEx
ClearStatus	EFD_GEN_SingleRead

Table 5: Functions Across Flash Devices (Continued)

Function	S33
WriteEnable	EFD_GEN_SingleProgram
WriteDisable	EFD_GEN_BufferProgram
BulkErase	EFD_GEN_ProgramSuspend
SectorErase	EFD_GEN_ProgramResume
PageProgram	EFD_GEN_EraseSuspend
ReadFlash	EFD_GEN_EraseResume

Hardware-specific Bus Operations

The validation test approach focused on testing XiP and SnD on the following configurations: The low-level code requires hardware-specific Read and Write Bus operations to communicate with the flash devices. The implementation of these operations is hardwareplatform dependent as it depends on the microprocessor on which the code runs and on the location of the memory in the microprocessor's address space. These methods should be customized for use on the target hardware. They can be replaced with a macro definition to increase performance when it is possible. A function is used here instead of a macro to allow the user to expand it if necessary.

EFD_HAL_WriteFlash

This function is used to write data to a flash device.

Prototype

```
void EFD_HAL_WriteFlash(
                        UINT32   addr,
                        UINT32   value );
```

Parameters

Parameter	Description
addr	(IN) The address to which data will be written on the flash device.
value	(IN) Data.

Return Values

None.

EFD_HAL_ReadFlash

This function is used to read data from a flash device.

Prototype

```
UINT32 EFD_HAL_ReadFlash( UINT32 addr );
```

Parameters

Parameter	Description
addr	(IN) The address from which data will be read on the flash device.

Return Values

None.

EFD_HAL_TimeOut

This function is used to check the Time-Out value set for different flash operations. Pass msec equal 0 to start time control. Pass msec equal 0xFFFFFFFF to set unlimited time-out.

Prototype

```
FLASH_ERROR EFD_HAL_TimeOut( UINT32 msec );
```

Parameters

Parameter	Description
msec	(IN) The Time-Out value in milliseconds.

Return Values

Value	Description	Success/Failure
FLASH_ERR_NONE	The function executed successfully.	Success
FLASH_ERR_TIMEOUT	An error due to a time-out lapse.	Failure

EFD_HAL_ControlInterrupt

This function is used to set the interrupt level. The system must provide a method for interrupt polling. Interrupt polling is required to handle real-time interrupts that might occur during erase and programming operations on the flash memory device.

Prototype

```
UINT32 EFD_HAL_ControlInterrupt( UINT32 irq_level );
```

Parameters

Parameter	Description
irq_level	(IN) The IRQ Level value.

Return Values

The previous IRQ level value.

EFD_HAL_PendingInterrupt

This function is used to check if there is a pending interrupt.

Prototype

```
UINT32 EFD_HAL_PendingInterrupt( void );
```

Parameters

None.

Return Values

A 0 or 1 value.

EFD_HAL_SerialFlash

This function is used to access a serial flash device.

Prototype

```
void EFD_HAL_SerialFlash( SPI_IO_DATA *io_data );
```

Parameters

Parameter	Description
io_data	(IN/OUT) The serial data content.

Return Values

None.

Configuration

The EFD was designed to support various architectures. Special files, such as efd_conf.h, are required to set the driver code for your target platform. This file is meant to provide all opportunities to customize the EFD according to the requirements of the hardware and flash configuration. It is possible to choose the flash start address, number of flash chips, hardware configuration and performance data.

There are several options in this file to help users, such as:

- The flash chip type
- The start address where the flash memory chips are “visible” within the memory of the CPU
- The bus width in the configuration that defines how to access the user’s hardware
- The timer value
- Options to enable/disable low-level operations if users are concerned about the code size

The EFD package is provided with several efd_conf.h files. They are placed as templates to form the proper user’s configuration.

Flash Device Type

#define USED_DEVICE FLASH_DEV_P3X

Description

These options allow the inclusion of source code for specific flash devices, reducing the code size of the resulting library.

- **FLASH_DEV_P3X** - P30/33-130nm StrataFlash® Embedded Memory
- **FLASH_DEV_P3X_65** - P30/33-65nm StrataFlash® Embedded Memory
- **FLASH_DEV_J3** - J3D-130nm Embedded Flash Memory
- **FLASH_DEV_J3_65** - J3D-65nm Embedded Flash Memory
- **FLASH_DEV_C3** - C3 Advanced+ Boot Flash Memory
- **FLASH_DEV_S33** - S33 Serial Flash Memory

BASE_ADDR

Macro

#define BASE_ADDR 0x80000000

Description

The start address where the flash memory chips are “visible” within the memory of the CPU is called the BASE_ADDR. This address must be set according to the current system. All functions in EFD use the address offset from the BASE_ADDR constant.

CHIP_BUS_WIDTH

Macro

#define USED_CONFIGURATION _16BIT_1_X16_

Description

Flash and Board Configuration.

The driver supports different configurations of the flash chips on the board. In each configuration, a new data Type called “FLASHDATA” is defined to match the current CPU data bus width. This data type is then used for all accesses to the memory. The different options (defines) are:

- _8BIT_1_X8_ - The device configuration is no-interleaving, a single x8 device.
- _16BIT_1_X16_ - No-interleaving, a single x16 device in x16 mode.
- _16BIT_2_X8_ - Two x8 devices interleaved to form x16.
- _32BIT_1_X32_ - No-interleaving, a single x32 device in x32 mode.
- _32BIT_2_X16_ - Two x16 devices interleaved to organize x32.
- _32BIT_4_X8_ - Four x8 devices interleaved to organize x32.

Enable/Disable Low-level Operations

This section provides a configuration list for customer purposes. It is possible to enable/disable individual operations to reduce the code size. If an operation that is not supported by the flash device is enabled, nothing happens. The "Functions across flash devices" table lists the supported and unsupported operations for different flash devices.

Please note that turning off required operations can lead to unworkable external API functions (returning the FLASH_ERR_UNSUPPORTED error code). The following low-level functions are base functions and cannot be disabled:

- ReadFlash()
- WriteFlash()
- BlockErase()
- BlockUnlock()
- BlockLock()

Macro for CFI Devices

#define EFD_CONF_DEVICE_ID TRUE

#define EFD_CONF_CFI TRUE

#define EFD_CONF_CFI_EX TRUE

#define EFD_CONF_BUFFERED_WRITE TRUE

#define EFD_CONF_PROGRAM_SUSPEND TRUE

#define EFD_CONF_ERASE_SUSPEND TRUE

#define EFD_CONF_BLOCK_STATE TRUE

#define EFD_CONF_BLOCK_LOCKDOWN TRUE

#define EFD_CONF_READ_PROTECTION_REG TRUE

#define EFD_CONF_WRITE_PROTECTION_REG TRUE

#define EFD_CONF_READ_CONFIGURATION_REG TRUE

#define EFD_CONF_WRITE_CONFIGURATION _REG TRUE

#define EFD_CONF_ALTERABLE_WRITE FALSE

#define EFD_CONF_ALTERABLE_BUFFERED_WRITE FALSE

#define EFD_CONF_STREAM_MODE FALSE

#define EFD_CONF_BLANK_CHECK FALSE

#define EFD_CONF_EFI FALSE

Description

Option	Description
EFD_CONF_DEVICE_ID	Cut off DeviceId().
EFD_CONF_CFI	Cut off ReadCfi().
EFD_CONF_CFI_EX	Cut off ReadCfiEx().
EFD_CONF_BUFFERED_WRITE	Cut off BufferProgram().
EFD_CONF_PROGRAM_SUSPEND	Cut off ProgramSuspend() and ProgramResume().
EFD_CONF_ERASE_SUSPEND	Cut off EraseSuspend() and EraseResume().
EFD_CONF_BLOCK_STATE	Cut off BlockState().
EFD_CONF_BLOCK_LOCKDOWN	Cut off BlockLockDown().
EFD_CONF_READ_PROTECTION_REG	Cut off ReadProtReg().
EFD_CONF_WRITE_PROTECTION_REG	Cut off WriteProtReg().
EFD_CONF_READ_CONFIGURATION_REG	Cut off ReadConfReg().
EFD_CONF_WRITE_CONFIGURATION_REG	Cut off WriteConfReg().
EFD_CONF_ALTERABLE_WRITE	Cut off BitAltWriteFlash().
EFD_CONF_ALTERABLE_BUFFERED_WRITE	Cut off BitAltBufferProgram().
EFD_CONF_STREAM_MODE	Cut off StreamModeEntry() and StreamModeExit().
EFD_CONF_BLANK_CHECK	Cut off BlankCheck().
EFD_CONF_EFI	Cut off ProgramEfi().

Macro for SPI Devices

#define EFD_CONF_SPI_DEVICE_ID TRUE

#define EFD_CONF_SPI_OTP TRUE

#define EFD_CONF_SPI_DPD TRUE

Description

Option	Description
EFD_CONF_SPI_DEVICE_ID	Cut off DeviceId().
EFD_CONF_SPI_OTP	Cut off ReadOtp() and WriteOtp.
EFD_CONF_SPI_DPD	Cut off InstallDpd() and ReleaseDpd.

EFD Contents

Directory Structure

The EFD source code is grouped in the following structure:

```
\
+---api                          Contents header and source files
with API implementation
|            efd_api.h
|            efd_api.c
|
+---cmn                          Header files for EFD
|            efd_def.h
|            efd_err.h
|            efd_ioctl.h
|
+---conf                     Directories with different configura-
tions (as an example)
|        +----j3
|        |            efd_conf.h
|        |
|        +----c3
|        |            efd_conf.h
|        |
|        +----p3x
|                    efd_conf.h
|
+---flash                     Low-level flash device operations
|    +----gen                              General operations for
all Intel command set CFI flash devices
|        |            gen_cmd.h
|        |            gen_lib.h
|        |            gen_op.h
|        |            gen_op.c
|        |
|        +----spi                    General operations for all
SPI flash devices
|        |            spi_cmd.h
|        |            spi_lib.h
|        |            spi_op.h
|        |            spi_op.c
|        |
|        +----p3x                    Specific operations
|        |            p3x.h
|        |            p3x.c
|        |
|        +----j3
|        |            j3.h
|        |            j3.c
|        |
|        +----c3
|        |            c3.h
|        |            c3.c
|        |
|
+---hw                          Hardware specific code
        +----pxa27x                          HAL functions for
```

```
PXA27x
                              efd_hal.h
                              efd_hal.c
```

EFD_CONF.H

The EFD is adaptable to meet the needs of the diverse types of systems to which it may be ported. Most customization is done by the means of compile options (defines) that configure the source code when compiled. This file contains user-configurable options. The user should define these options. Several configurations are located in this package and can be used as templates. For a description of these options, see the section "Configuration".

EFD_DEF.H

This file defines the standard constants and data type definitions.

EFD_ERR.H

This file defines the error codes.

EFD_IOCTL.H

This file defines FLASH_COMMAND and FLASH_PARAMETER.

EFD_API.H

This file declares the external function prototypes for the static library and basic types.

EFD Assumptions

Low-Level Functions

All low-level functions are blocked, which means that they wait for the completion of an operation, check the Status Register and clear one in the case of an error. When an error occurs, the software returns the error code. It is up to the user to decide what to do.

For an implemented set of operations, see Table 4 (page 55) and Table 5 (page 57).

Performance

The following list provides tips for improving EFD performance:

- Optimum programming performance and lower power usage are obtained by aligning the operation address at the beginning of a word boundary.
- Passing an aligned buffer pointer will give better performance.
- Since the flash memory is usually mapped to memory, you can write/read data without any special operations (write/read directly to/from memory). A macro can be used instead of EFD_HAL_WriteFlash and EFD_HAL_ReadFlash, increasing the speed of the flash routines.
- While Software Read While Write provides excellent flexibility for the entire solution, it usually comes with a performance tax for write/erase procedures suspending/unsuspending flash operations. As a result, SW-RWW and SW-RWE can be tuned by setting the PROGRAM_SUSPEND_THRESHOLD and ERASE_SUSPEND_THRESHOLD macros located in the efd_api.c file.

Building and Compiling

No “makefiles” are provided as this code represents a set of individually usable subroutines and is intended to be integrated into existing software development packages and build environments.

Validating and Testing

The validation test approach focused on testing XiP and SnD on the following configurations:

- StrataFlash® Embedded Memory (P33) x16
- Phase Change Memory (PCM) Memory (A33) x16
- Embedded Flash Memory (J3D) x32
- StrataFlash® Embedded Memory (P30) x32
- Advanced+ Boot Flash Memory (C3) x32

For each release, testing is focused on the external API, low-level operations, multi-thread and stress testing.

Hardware

The EFD software was developed on an Intel® Bulverde DVK Platform with Rodan Processor Card (PXA27x).

Software

The EFD software was compiled with the Intel XScale® compiler (from SDT2.0.1) and built with a Nucleus OS.

Exceptions

The code for the following devices was validated using only basic functionality tests:

- StrataFlash® Embedded Memory (P33-65nm)
- StrataFlash® Embedded Memory (P30-65nm)
- Embedded Flash Memory (J3-65nm)

Code for the following device can be used as reference only:

- Serial Flash Memory (S33)

How to Use the EFD Software

This package is provided without “makefiles” and you must adapt the EFD software to your environment.

Relocate all functions in the following source files, such as p3x.c, c3.c, j3.c, gen_op.c, efd_api.c, efd_hal.c, to RAM if the EFD software and the code are managed on the same device.

Before using the software on the target platform, the user must adapt the hardware-specific bus operations located in the efd_hal.c and efd_hal.h files (see the section "Hardware-specific bus operations") and form the proper efd_conf.h file using the files inside the EFD package as templates. See Configuration (page 60).

When developing an application, it is recommended to proceed as follows:

1. Write a simple program using the external API to test the low-level code provided and verify that it operates as expected on the target system and software environments. To test the source code on the target system, call the EFD_Erase function and EFD_Read if the device is erased. Only 0xFF data should be read. Then, read the Manufacturer and Device Code by issuing EFD_Ioctl with FLASH_CMD_ID i/o code function and verify that they are correct. If these functions work, the other functions also work. All functions should be tested thoroughly.
2. Write the high-level code for the application. The high-level code will access the flash memory device by calling the low-level code provided.
3. Test the complete source code for the application.

EFD API Usage Examples

Read Data from Flash Memory Using a User-defined Address

The following code example illustrates the read operation on a flash device.

```
#include "efd_api.h"

static FLASH_DEVICE_OBJECT fdo;

int efd_read( int argc, const char *argv[] )
{
    int                status = 0;
    UINT32                offset = 0;
    UINT32                length = 0;
    UINT32                ret_length = 0;
    UINT8               buffer[1024];

        if ( sscanf(argv[0], "0x%X", &offset) != 1 &&
                sscanf(argv[0], "0X%X", &offset) != 1 )
        {
            return 1;
        }
        if ( sscanf(argv[1], "0x%X", &length) != 1 &&
                sscanf(argv[1], "0X%X", &length) != 1 )
        {
            return 1;
        }

    }

    status = EFD_Init( &fdo );

    if ( status ) return status;

    status = EFD_Read( &fdo, offset, length, buffer,
&ret_length );
    printf("Read operation has been completed\n");
    printf("Status: 0x%04X", status);
    printf("Address: 0x%08X\n", offset);
    printf("Read length: 0x%X\n", lentgh);

    EFD_Exit( &fdo );

    return status;
}
```

Write Data from Buffer and Into Flash Memory Using User-defined Address

The following code example illustrates the write operation on the flash device.

```
#include "efd_api.h"

static FLASH_DEVICE_OBJECT fdo;

int efd_write( int argc, const char *argv[] )
{
    int   status = 0;
    FLASH_PARAMETER   param;
```

```
    FLASH_COMMAND command;
    UINT32   offset = 0;
    UINT32   length = 0;
    UINT32   ret_length = 0;
    UINT8    buffer[]="1234567890 \
abcdefghijklmnopqrstuvwxyz \
ABCDEFGHIJKLMNOPQRSTUVWXYZ";

        if ( sscanf(argv[0], "0x%X", &offset) != 1 &&
                sscanf(argv[0], "0X%X", &offset) != 1 )
        {
            return 1;
        }
        if ( sscanf(argv[1], "0x%X", &length) != 1 &&
                sscanf(argv[1], "0X%X", &length) != 1 )
        {
            return 1;
        }
        if ( length > (sizeof(buffer) / sizeof(buffer[0]))) re-
turn 1;
    }

    status = EFD_Init( &fdo );

    if ( status ) return status;

    param.BlockOp.Addr = offset;
    command = FLASH_CMD_BLOCK_UNLOCK;
status = EFD_Ioctl( &fdo, command, &param );

    if ( !status )
    {
        status = EFD_Write( &fdo, offset, length, buffer,
&ret_length );
        printf("Write operation has been completed\n");
        printf("Status: 0x%04X", status);
printf("Address: 0x%08X\n", offset);
        printf("Read length: 0x%X\n", lentgh);
    }

    if ( !status )
    {
        command = FLASH_CMD_BLOCK_LOCK;
        status = EFD_Ioctl( &fdo, command, &param );
    }

    EFD_Exit( &fdo );

    return status;
}
```

IOCTL Method Usage

The following code example illustrates a low-level operation on the flash device.

```
#include "efd_api.h"

static FLASH_DEVICE_OBJECT fdo;
```

```
int efd_ioctl( int argc, const char *argv[] )
{
    int   status = 0;
    FLASH_PARAMETER   param;
    FLASH_COMMAND command;
    UINT8 DataCfi[100];

    /* command line arguments processing */
    {
        if ( strncmp( argv[0], "ID", strlen(argv[0]) ) == 0 )
        {
            command = FLASH_CMD_ID;
        }
        else if ( strncmp( argv[0], "CFI", strlen(argv[0]) ) ==
0 )
        {
            if ( sscanf(argv[1], "0x%X", &param.Cfi.Offset) != 1
&&
                    sscanf(argv[1], "0X%X", &param.Cfi.Offset) !
= 1 )
            {
                return 1;
            }
            if ( sscanf(argv[2], "0x%X", &param.Cfi.Length) != 1
&&
                    sscanf(argv[2], "0X%X", &param.Cfi.Length) !
= 1 )
            {
                return 1;
            }
            param.Cfi.Data = DataCfi;
            command = FLASH_CMD_CFI;
        }
        else if ( strncmp( argv[0], "STATE", strlen(argv[0]) )
== 0 )
        {
            if ( sscanf(argv[1], "0x%X", &param.Block-
State.Addr) != 1 &&
                    sscanf(argv[1], "0X%X", &param.Block-
State.Addr) != 1 )
            {
                return 1;
            }
            command = FLASH_CMD_BLOCK_STATE;
        }

    status = EFD_Init( &fdo );

    if ( status ) return status;

    status = EFD_Ioctl( &fdo, command, &param );
    printf("Operation 0x%04X has been completed\n", command);
    printf("Status: 0x%04X", status);

    EFD_Exit( &fdo );

    return status;
}
```

Revision History

Rev. H, 01/12

Updated hyperlinks.

Rev. G, 07/10

Applide branding.

Rev. 6, 11/09

Removed references to A33.

Rev. 5, 06/09

Updated document formatting.

Updated the J3D and J3-65nm columns in the “Functions across flash devices” tables.

Rev. 4, 06/09

Updated document formatting.

Rev. 3, 11/08

Updates for EFD 2.0.

Rev. 2, 03/2008

Applied branding.

Rev. 1, 12/07

Initial public release.

8000 S. Federal Way, P.O. Box 6, Boise, ID 83707-0006, Tel: 208-368-3900
www.micron.com/productsupport Customer Comment Line: 800-932-4992

This data sheet contains minimum and maximum limits specified over the power supply and temperature range set forth herein. Although considered final, these specifications are subject to change, as further product development and data characterization sometimes occur.