The following document contains information on Cypress products. Although the document is marked with the name “Spansion” and “Fujitsu”, the company that originally developed the specification, Cypress will continue to offer these products to new and existing customers.

Continuity of Specifications
There is no change to this document as a result of offering the device as a Cypress product. Any changes that have been made are the result of normal document improvements and are noted in the document history page, where supported. Future revisions will occur when appropriate, and changes will be noted in a document history page.

Continuity of Ordering Part Numbers
Cypress continues to support existing part numbers. To order these products, please use only the Ordering Part Numbers listed in this document.

Worldwide Sales and Design Support
We look forward to partnering with you to provide solutions that will help make your systems successful. Cypress maintains a worldwide network of offices, solution centers, manufacturer’s representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

About Cypress
Cypress (NASDAQ: CY) delivers high-performance, high-quality solutions at the heart of today’s most advanced embedded systems, from automotive, industrial and networking platforms to highly interactive consumer and mobile devices. With a broad, differentiated product portfolio that includes NOR flash memories, F-RAM™ and SRAM, Traveo™ microcontrollers, the industry’s only PSoC® programmable system-on-chip solutions, analog and PMIC Power Management ICs, CapSense® capacitive touch-sensing controllers, and Wireless BLE Bluetooth® Low-Energy and USB connectivity solutions, Cypress is committed to providing its customers worldwide with consistent innovation, best-in-class support and exceptional system value.

FUJITSU

FMX
MICROCONTROLLER
FMX Series

SD CARD LIBRARY
APPLICATION NOTE

For more information for the FM3 microcontroller, visit the web site at:
http://www.fujitsu.com/global/services/microelectronics/product/micom/roadmap/industrial/fm3/

ALL RIGHTS RESERVED

The contents of this document are subject to change without notice.

Customers are advised to consult with sales representatives before ordering.

The information, such as descriptions of function and application circuit examples, in this document are presented solely for the purpose of reference to show examples of operations and uses of FUJITSU SEMICONDUCTOR device; FUJITSU SEMICONDUCTOR does not warrant proper operation of the device with respect to use based on such information. When you develop equipment incorporating the device based on such information, you must assume any responsibility arising out of such use of the information. FUJITSU SEMICONDUCTOR assumes no liability for any damages whatsoever arising out of the use of the information.

Any information in this document, including descriptions of function and schematic diagrams, shall not be construed as license of the use or exercise of any intellectual property right, such as patent right or copyright, or any other right of FUJITSU SEMICONDUCTOR or any third party or does FUJITSU SEMICONDUCTOR warrant non-infringement of any third-party's intellectual property right or other right by using such information. FUJITSU SEMICONDUCTOR assumes no liability for any infringement of the intellectual property rights or other rights of third parties which would result from the use of information contained herein.

The products described in this document are designed, developed and manufactured as contemplated for general use, including without limitation, ordinary industrial use, general office use, personal use, and household use, but are not designed, developed and manufactured as contemplated (1) for use accompanying fatal risks or dangers that, unless extremely high safety is secured, could have a serious effect to the public, and could lead directly to death, personal injury, severe physical damage or other loss (i.e., nuclear reaction control in nuclear facility, aircraft flight control, air traffic control, mass transport control, medical life support system, missile launch control in weapon system), or (2) for use requiring extremely high reliability (i.e., submersible repeater and artificial satellite).
Please note that FUJITSU SEMICONDUCTOR will not be liable against you and/or any third party for any claims or damages arising in connection with above-mentioned uses of the products.

Please note that FUJITSU SEMICONDUCTOR will not be liable against you and/or any third party for any claims or damages arising in connection with above-mentioned uses of the products.

Any semiconductor devices have an inherent chance of failure. You must protect against injury, damage or loss from such failures by incorporating safety design measures into your facility and equipment such as redundancy, fire protection, and prevention of over-current levels and other abnormal operating conditions.

Exportation/release of any products described in this document may require necessary procedures in accordance with the regulations of the Foreign Exchange and Foreign Trade Control Law of Japan and/or US export control laws.

The company names and brand names herein are the trademarks or registered trademarks of their respective owners.

Revision History

Date	Author	Version
2013-03-13	Robin Qian	V1.0

Table of Contents

1 Introduction

1.1 About FMx MCU

FMx MCU means the CM0+, CM3 and CM4 MCU of Fujitsu (FM0+, FM3, FM4). It is a series of 32-bit ARMv7 architecture MCU. There is CSIO I/F in all FMx series MCU. And for CSIO I/F, it contains SIO and SPI mode communication method.

1.2 About SD Card Library (Secure Digital)

For SD, it is a non-volatile memory card format for use in portable devices, and SD has four families: SDSC/SDHC/SDXC/SDIO. SD has three different forms: original size/mini size/micro size. For communication methods, it supports: SPI & SD I/F.

In this SD Card Library, it will support:

- For families, it supports: SDSC/SDHC.
- For forms, it supports all.
- For communications, it supports SPI.
- For SD API, it provides:
  - SD Card Initialization
  - SD Card sector R/W
  - SD Card status get

1.3 About Document

This application note will mainly introduce:

1) SD Card Library API introduction
2) SD Card Library sample demonstration
3) SD Card Library adaption steps

2 SD Card Library API introduction

2.1 SD Card Library

It mainly includes:

- **Low level API:**
  - static void SdcardPowerOn(void)
  - static void SdcardChipSelect(stc_sdcardinfo_t *pstcSdcardInfo, uint8_t u8Select)
  - static uint8_t SdcardPresent(void)
  - static uint8_t SdcardWriteProtect(void)
  - static void SdcardPowerOff(void)
  - void Sdcard_TimeWait(uint32_t u32cnt)
  - uint8_t Sdcard_IsCardExist(void)
  - void Sdcard_CardSelect(stc_sdcardinfo_t *pstcSdcardInfo)
  - void Sdcard_CardDeSelect(stc_sdcardinfo_t *pstcSdcardInfo)
  - void Sdcard_Start(void)
  - void Sdcard_PowerOn(void)
  - void Sdcard_WaitPowerup(void)
  - void Sdcard_Stop(void)
  - void Sdcard_PowerOff(void)
  - uint8_t Sdcard_IsWrProtect(void)
  - void Sdcard_ConfigFifo(stc_sdcardinfo_t *pstcSdcardInfo)
  - void Sdcard_SpiModeConfig(stc_sdcardinfo_t *pstcSdcardInfo)
  - static void SdcardSpiStopTX(void)
  - static void SdcardSpiTXHandler(void)
  - uint32_t Sdcard_Write(stc_sdcardinfo_t *pstcSdcardInfo, uint8_t *pu8Buf, uint16_t u16Length)
  - static void SdcardSpiStopRX(void)
  - static void SdcardSpiRXHandler(void)
  - uint32_t Sdcard_Read(stc_sdcardinfo_t *pstcSdcardInfo, uint8_t *pu8Buf, uint16_t u16Length)
- **SD CMD basic API:**
  - static uint8_t SdcardGetCRC7(uint8_t *pu8buf, int32_t i32len)
  - static uint16_t SdcardGetCRC16(const uint8_t *pu8buf, int32_t i32len)
  - static void WriteCmd(stc_sdcardinfo_t *pstcSdcardInfo, uint8_t u8idx, uint32_t u32arg)
  - static int32_t WaitResponse(stc_sdcardinfo_t *pstcSdcardInfo, uint8_t *pu8c, uint32_t u32timeout)

- static int32_t ReadCmdResponse(stc_sdcardinfo_t *pstcSdcardInfo, uint8_t *pu8buf, uint32_t u32len)
- static int32_t ReadDataBlock(stc_sdcardinfo_t *pstcSdcardInfo, uint8_t *u8buf, int32_t i32len, int32_t i32timeout)
- static int32_t ReadDataResponse(stc_sdcardinfo_t *pstcSdcardInfo)
- static int32_t WaitBusyOff(stc_sdcardinfo_t *pstcSdcardInfo)
- static int32_t WaitBusyOff2(stc_sdcardinfo_t *pstcSdcardInfo)
- static void Shutdown(stc_sdcardinfo_t *pstcSdcardInfo)

SD CMD API

- int32_t SDDSendCMD0_GO_IDLE_STATE(stc_sdcardinfo_t *pstcSdcardInfo)
- int32_t SDDSendCMD8_CHECK_VERSION(stc_sdcardinfo_t *pstcSdcardInfo)
- int32_t SDDSendCMD9_SEND_CSD(stc_sdcardinfo_t *pstcSdcardInfo, uint8_t *pu8csd)
- int32_t SDDSendCMD13_SEND_STAUS(stc_sdcardinfo_t *pstcSdcardInfo)
- int32_t SDDSendCMD16_SET_BLOCK_LEN(stc_sdcardinfo_t *pstcSdcardInfo)
- int32_t SDDSendCMD17_READ_SINGLE_BLOCK(stc_sdcardinfo_t *pstcSdcardInfo, uint32_t u32addr, uint8_t *pu8buf)
- int32_t SDDSendCMD24_WRITE_BLOCK(stc_sdcardinfo_t *pstcSdcardInfo, uint32_t u32addr, const uint8_t *pu8buf)
- static int32_t SendCMD55_APP_CMD(stc_sdcardinfo_t *pstcSdcardInfo)
- int32_t SDDSendACMD41_SEND_OP_CODE(stc_sdcardinfo_t *pstcSdcardInfo)
- int32_t SDDSendCMD58_READ_OCR(stc_sdcardinfo_t *pstcSdcardInfo, uint8_t *pu8buf)
- int32_t SDDSendCMD59_CRC_ON_OFF(stc_sdcardinfo_t *pCSIOInfo, int32_t i32on_off)

SD API

- int32_t Sdcard_Init(stc_sdcardinfo_t *pstcSdcardInfo)
- int32_t Sdcard_ReadSector(stc_sdcardinfo_t *pstcSdcardInfo, uint32_t u32start, uint16_t u16count, uint8_t *pu8buf)
- int32_t Sdcard_WriteSector(stc_sdcardinfo_t *pstcSdcardInfo, uint32_t u32start, uint16_t u16count, const uint8_t *pu8buf)
- int32_t Sdcard_GetCardSize(uint32_t *pu32sectors)

2.2 Global API Functions

2.2.1 Sdcard_TimeWait

Wait for counted number cycles

Prototype:

void

Sdcard_TimeWait(uint32_t u32cnt)

Parameter:

u32cnt: Counted numbers

Return:

None

Description:

This function is used during the SD card command/initialization, for time wait usage.

The parameter *u32cnt* can be:

- **unsigned int number**

Remark:

None.

2.2.2 Sdcard_IsCardExist

Check whether the SD Card is in the socket

Prototype:

uint8_t

Sdcard_IsCardExist(void)

Parameter:

None

Return:

SD Card exists status

Description:

This function is used to check the CD pin to find out whether the SD Card is in the socket.

The return value can be one of following values:

0: Not exists

1: Exists

Remark:

None.

2.2.3 Sdcard_CardSelect

SD card selection on.

Prototype:

void

Sdcard_CardSelect(stc_sdcardinfo_t *pstcSdcardInfo)

Parameter:

***pstcSdcardInfo*:** Pointer of SD card information

Return:

None

Description:

This function is used to make CS of SD Card effective.

Remark:

None.

2.2.4 Sdcard_CardDeSelect

SD card selection off.

Prototype:

void

Sdcard_CardDeSelect(stc_sdcardinfo_t *pstcSdcardInfo)

Parameter:

***pstcSdcardInfo*:** Pointer of SD card information

Return:

None

Description:

This function is used to make CS of SD Card un-effective.

Remark:

None.

2.2.5 Sdcard_IsWrProtect

Check whether the SD Card is written protected

Prototype:

uint8_t

Sdcard_IsWrProtect(void)

Parameter:

None

Return:

SD Card written protected status

Description:

This function is used to check the WP pin to find out whether the SD Card is written protected.

The return value can be one of following values:

0: Not protected

1: Protected

Remark:

None.

2.2.6 Sdcard_ConfigFifo

Configure FIFO for SPI channel which used for SD Card.

Prototype:

void

Sdcard_ConfigFifo(stc_sdcardinfo_t *pstcSdcardInfo)

Parameter:

***pstcSdcardInfo*: Pointer of SD card information**

Return:

None

Description:

This function is used to configure FIFO for SPI channel which used for SD Card.

Remark:

None.

2.2.7 Sdcard_SpiModeConfig

Configure SPI channel which used for SD Card.

Prototype:

void

Sdcard_SpiModeConfig(stc_sdcardinfo_t *pstcSdcardInfo)

Parameter:

***pstcSdcardInfo*:** Pointer of SD card information

Return:

None

Description:

This function is used to configure working mode for SPI channel which used for SD Card.

Remark:

None.

2.2.8 Sdcard_Write

Transmit data to the SD Card.

Prototype:

uint32_t

Sdcard_Write(stc_sdcardinfo_t *pstcSdcardInfo, uint8_t *pu8Buf, uint16_t u16Length)

Parameter:

***pstcSdcardInfo*:** Pointer of SD card information

pu8Buf : Pointer of transmit data buffer

u16Length : Number of transmit data

Return:

None

Description:

This function is used to write the specified length data to the SD Card.

The return value can be one of following values:

SpiCommEND: Successful end

SpiCommERR: Error during communication

Remark:

None.

2.2.9 Sdcard_Read

Receive data from the SD Card.

Prototype:

uint32_t

Sdcard_Read(stc_sdcardinfo_t *pstcSdcardInfo, uint8_t *pu8Buf, uint16_t u16Length)

Parameter:

***pstcSdcardInfo*:** Pointer of SD card information

pu8Buf : Pointer of received data buffer

u16Length : Number of received data

Return:

None

Description:

This function is used to receive the specified length data to the SD Card.

The return value can be one of following values:

SpiCommEND: Successful end

SpiCommERR: Error during communication

Remark:

None.

2.2.10 Sdcard_Cmd0

Command 0 of the SD Card.

Prototype:

int32_t

Sdcard_Cmd0(stc_sdcardinfo_t *pstcSdcardInfo)

Parameter:

***pstcSdcardInfo*:** Pointer of SD card information

Return:

Command 0 result

Description:

This function is used to send command 0 to the SD Card, GO_IDLE_STATE, used to change from SD to SPI mode.

The return value can be one of following values:

E_SDD_CMD_OK: Success

E_SDD_CMD_ACCEPT: Command accepted

E_SDD_CMD_ERR: Command error

E_SDD_CMD_TIMEOUT: Timeout error

E_SDD_CMD_CRC_ERR: CRC error

E_SDD_CMD_WR_ERR: Write error

E_SDD_CMD_IDLE: Command idle

E_SDD_CMD_CARD_ERR: Card error

Remark:
None.

2.2.11 Sdcard_Cmd8

Command 8 of the SD Card.

Prototype:
int32_t

Sdcard_Cmd8(stc_sdcardinfo_t *pstcSdcardInfo)

Parameter:
***pstcSdcardInfo*:** Pointer of SD card information

Return:
Command 8 result

Description:
This function is used to send command 8 to the SD Card, required to support High Capacity Memory.

The return value can be one of following values:

E_SDD_CMD_OK: Success

E_SDD_CMD_ACCEPT: Command accepted

E_SDD_CMD_ERR: Command error

E_SDD_CMD_TIMEOUT: Timeout error

E_SDD_CMD_CRC_ERR: CRC error

E_SDD_CMD_WR_ERR: Write error

E_SDD_CMD_IDLE: Command idle

E_SDD_CMD_CARD_ERR: Card error

Remark:
None.

2.2.12 Sdcard_Cmd9

Command 9 of the SD Card.

Prototype:
int32_t

Sdcard_Cmd9(stc_sdcardinfo_t *pstcSdcardInfo)

Parameter:

***pstcSdcardInfo*:** Pointer of SD card information

Return:

Command 9 result

Description:

This function is used to send command 9 to the SD Card, SEND_CSD.

The return value can be one of following values:

E_SDD_CMD_OK: Success

E_SDD_CMD_ACCEPT: Command accepted

E_SDD_CMD_ERR: Command error

E_SDD_CMD_TIMEOUT: Timeout error

E_SDD_CMD_CRC_ERR: CRC error

E_SDD_CMD_WR_ERR: Write error

E_SDD_CMD_IDLE: Command idle

E_SDD_CMD_CARD_ERR: Card error

Remark:

None.

2.2.13 Sdcard_Cmd13

Command 13 of the SD Card.

Prototype:

int32_t

Sdcard_Cmd13(stc_sdcardinfo_t *pstcSdcardInfo)

Parameter:

***pstcSdcardInfo*:** Pointer of SD card information

Return:

Command 13 result

Description:

This function is used to send command 13 to the SD Card, SEND_STATUS.

The return value can be one of following values:

E_SDD_CMD_OK: Success

E_SDD_CMD_ACCEPT: Command accepted

E_SDD_CMD_ERR: Command error

E_SDD_CMD_TIMEOUT: Timeout error

E_SDD_CMD_CRC_ERR: CRC error

E_SDD_CMD_WR_ERR: Write error

E_SDD_CMD_IDLE: Command idle

E_SDD_CMD_CARD_ERR: Card error

Remark:
None.

2.2.14 Sdcard_Cmd16

Command 16 of the SD Card.

Prototype:
int32_t

Sdcard_Cmd16(stc_sdcardinfo_t *pstcSdcardInfo)

Parameter:
***pstcSdcardInfo*:** Pointer of SD card information

Return:
Command 16 result

Description:
This function is used to send command 16 to the SD Card, SET_BLOCKLEN.

The return value can be one of following values:

E_SDD_CMD_OK: Success

E_SDD_CMD_ACCEPT: Command accepted

E_SDD_CMD_ERR: Command error

E_SDD_CMD_TIMEOUT: Timeout error

E_SDD_CMD_CRC_ERR: CRC error

E_SDD_CMD_WR_ERR: Write error

E_SDD_CMD_IDLE: Command idle

E_SDD_CMD_CARD_ERR: Card error

Remark:
None.

2.2.15 Sdcard_Cmd17

Command 17 of the SD Card.

Prototype:

int32_t

Sdcard_Cmd17(stc_sdcardinfo_t *pstcSdcardInfo)

Parameter:

***pstcSdcardInfo*:** Pointer of SD card information

Return:

Command 17 result

Description:

This function is used to send command 17 to the SD Card, READ_SINGLE_BLOCK.

The return value can be one of following values:

E_SDD_CMD_OK: Success

E_SDD_CMD_ACCEPT: Command accepted

E_SDD_CMD_ERR: Command error

E_SDD_CMD_TIMEOUT: Timeout error

E_SDD_CMD_CRC_ERR: CRC error

E_SDD_CMD_WR_ERR: Write error

E_SDD_CMD_IDLE: Command idle

E_SDD_CMD_CARD_ERR: Card error

Remark:

None.

2.2.16 Sdcard_Cmd24

Command 24 of the SD Card.

Prototype:

int32_t

Sdcard_Cmd24(stc_sdcardinfo_t *pstcSdcardInfo)

Parameter:

***pstcSdcardInfo*:** Pointer of SD card information

Return:

Command 24 result

Description:

This function is used to send command 24 to the SD Card, WRITE_BLOCK.

The return value can be one of following values:

E_SDD_CMD_OK: Success

E_SDD_CMD_ACCEPT: Command accepted

E_SDD_CMD_ERR: Command error

E_SDD_CMD_TIMEOUT: Timeout error

E_SDD_CMD_CRC_ERR: CRC error

E_SDD_CMD_WR_ERR: Write error

E_SDD_CMD_IDLE: Command idle

E_SDD_CMD_CARD_ERR: Card error

Remark:

None.

2.2.17 Sdcard_AppCmd41

Application Command 41 of the SD Card.

Prototype:

int32_t

Sdcard_AppCmd41(stc_sdcardinfo_t *pstcSdcardInfo)

Parameter:

***pstcSdcardInfo*:** Pointer of SD card information

Return:

Application Command 41 result

Description:

This function is used to send application command 41 to the SD Card, SD_APP_OP_COND.

The return value can be one of following values:

E_SDD_CMD_OK: Success

E_SDD_CMD_ACCEPT: Command accepted

E_SDD_CMD_ERR: Command error

E_SDD_CMD_TIMEOUT: Timeout error

E_SDD_CMD_CRC_ERR: CRC error

E_SDD_CMD_WR_ERR: Write error

E_SDD_CMD_IDLE: Command idle

E_SDD_CMD_CARD_ERR: Card error

Remark:
None.

2.2.18 Sdcard_AppCmd55

Application Command 55 of the SD Card.

Prototype:
int32_t

Sdcard_AppCmd55(stc_sdcardinfo_t *pstcSdcardInfo)

Parameter:
***pstcSdcardInfo*:** Pointer of SD card information

Return:
Application Command 55 result

Description:
This function is used to send application command 55 to the SD Card, APP_CMD.

The return value can be one of following values:

E_SDD_CMD_OK: Success

E_SDD_CMD_ACCEPT: Command accepted

E_SDD_CMD_ERR: Command error

E_SDD_CMD_TIMEOUT: Timeout error

E_SDD_CMD_CRC_ERR: CRC error

E_SDD_CMD_WR_ERR: Write error

E_SDD_CMD_IDLE: Command idle

E_SDD_CMD_CARD_ERR: Card error

Remark:
None.

2.2.19 Sdcard_Cmd58

Command 58 of the SD Card.

Prototype:
int32_t

Sdcard_Cmd58(stc_sdcardinfo_t *pstcSdcardInfo)

Parameter:

***pstcSdcardInfo*:** Pointer of SD card information

Return:

Command 58 result

Description:

This function is used to send command 58 to the SD Card, READ_OCR.

The return value can be one of following values:

E_SDD_CMD_OK: Success

E_SDD_CMD_ACCEPT: Command accepted

E_SDD_CMD_ERR: Command error

E_SDD_CMD_TIMEOUT: Timeout error

E_SDD_CMD_CRC_ERR: CRC error

E_SDD_CMD_WR_ERR: Write error

E_SDD_CMD_IDLE: Command idle

E_SDD_CMD_CARD_ERR: Card error

Remark:

None.

2.2.20 Sdcard_Cmd59

Command 59 of the SD Card.

Prototype:

int32_t

Sdcard_Cmd59(stc_sdcardinfo_t *pstcSdcardInfo)

Parameter:

***pstcSdcardInfo*:** Pointer of SD card information

Return:

Command 59 result

Description:

This function is used to send command 59 to the SD Card, CRC_ON_OFF.

The return value can be one of following values:

E_SDD_CMD_OK: Success

E_SDD_CMD_ACCEPT: Command accepted

E_SDD_CMD_ERR: Command error

E_SDD_CMD_TIMEOUT: Timeout error

E_SDD_CMD_CRC_ERR: CRC error

E_SDD_CMD_WR_ERR: Write error

E_SDD_CMD_IDLE: Command idle

E_SDD_CMD_CARD_ERR: Card error

Remark:
None.

2.2.21 Sdcard_Init

SD card initialization.

Prototype:
int32_t

Sdcard_Init(stc_sdcardinfo_t *pstcSdcardInfo)

Parameter:
***pstcSdcardInfo*:** Pointer of SD card information

Return:
Initialization result

Description:
This function is used to initialize the SD Card.

The return value can be one of following values:

E_SDD_OK: Success

E_SDD_NO_CARD: No card error

E_SDD_INACTIVE: SD Card is not activated

E_SDD_PARAMETER: Parameter error

E_SDD_PROTECT: SD Card is written protected

E_SDD_DEVICE_TYPE: Not support

E_SDD_ACTIVE: SD Card is already activated

Remark:
None.

2.2.22 Sdcard_ReadSector

SD card read sector.

Prototype:

int32_t

Sdcard_ReadSector(stc_sdcardinfo_t *pstcSdcardInfo, uint32_t u32start, uint16_t u16count, uint8_t *pu8buf)

Parameter:

***pstcSdcardInfo*:** Pointer of SD card information

***u32start*:** Start sector number

***u16count*:** Sector count

***pu8buf*:** Received buffer

Return:

Read result

Description:

This function is used to read data from the SD Card.

The return value can be one of following values:

E_SDD_OK: Success

E_SDD_NO_CARD: No card error

E_SDD_INACTIVE: SD Card is not activated

E_SDD_PARAMETER: Parameter error

E_SDD_PROTECT: SD Card is written protected

E_SDD_DEVICE_TYPE: Not support

E_SDD_ACTIVE: SD Card is already activated

Remark:

None.

2.2.23 Sdcard_WriteSector

SD card write sector.

Prototype:

int32_t

Sdcard_WriteSector(stc_sdcardinfo_t *pstcSdcardInfo, uint32_t u32start, uint16_t u16count, uint8_t *pu8buf)

Parameter:

***pstcSdcardInfo*:** Pointer of SD card information

***u32start*:** Start sector number

***u16count*:** Sector count

***pu8buf*:** Transmitted buffer

Return:

Write result

Description:

This function is used to write data to the SD Card.

The return value can be one of following values:

E_SDD_OK: Success

E_SDD_NO_CARD: No card error

E_SDD_INACTIVE: SD Card is not activated

E_SDD_PARAMETER: Parameter error

E_SDD_PROTECT: SD Card is written protected

E_SDD_DEVICE_TYPE: Not support

E_SDD_ACTIVE: SD Card is already activated

Remark:

None.

3 SD Card Library sample demonstration

3.1 SD Card basic API demo

3.1.1 Code

IO initialization code

```
    Sdcard_Io_Init();
```

SD Card information setting code

```
        /* Config structure init */
        Sdcard_stcSdSpiInfo.u8Channel = SD_MFS_CH;
        Sdcard_stcSdSpiInfo.u8HwFifoFlag = NOHWFIFO;
        Sdcard_stcSdSpiInfo.u8TransState = SpiCommIDLE;
        Sdcard_stcSdSpiInfo.pstcModeConfig = &m_stcSdSpiModeConfig;
        Sdcard_stcSdSpiInfo.pstcFIFOConfig = NULL;

        Sdcard_SpiModeConfig(&Sdcard_stcSdSpiInfo);

        if(Sdcard_stcSdSpiInfo.u8HwFifoFlag == HWFIFO)
        {
            Sdcard_ConfigFifo(&Sdcard_stcSdSpiInfo);
        }
```

SD Card initialization code

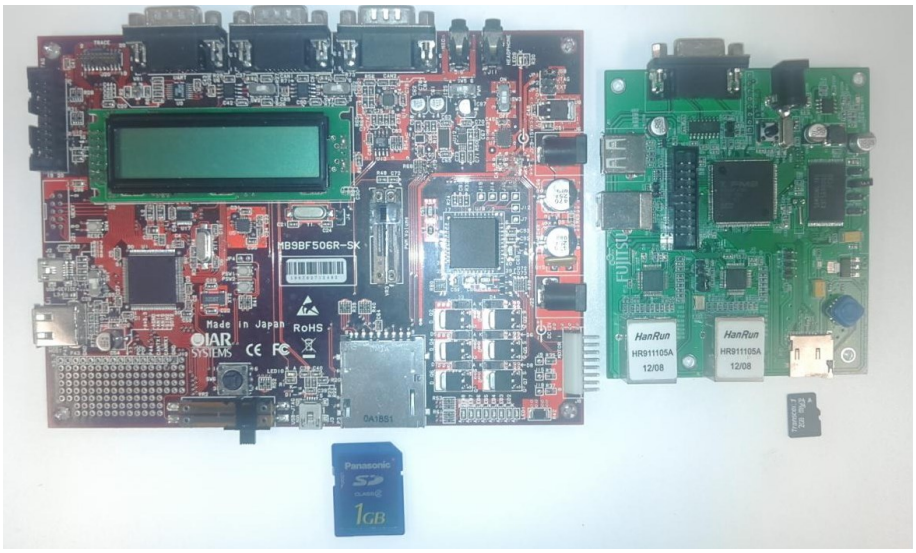
```
        i32RetValue = SdcardInit(&Sdcard_stcSdSpiInfo);

        if(0 == i32RetValue)
        {
         ...
```

SD Card get sector number/write/read sector test code

```
        if(0 == i32RetValue)
        {
            //Initialization success!
            i32RetValue = Sdcard_GetCardSize(&m_u32SdcardSectorNum);
            if(0 != i32RetValue)
            {
                //Fail
                while(1);
            }
            //Write sector test
            i32RetValue = SdcardWriteSector(&Sdcard_stcSdSpiInfo);
            if(0 != i32RetValue)
            {
                //Fail
                while(1);
            }
            //Read sector test
```

```
            i32RetValue = SdcardReadSector(&Sdcard_stcSdSpiInfo);
            if(0 != i32RetValue)
            {
                //Fail
                while(1);
            }
        }
```

3.1.2 Environment



MB9BF506R-SK (IAR)

FSSDC-9B618-EVB-V1.0

3.1.3 Demo procedure

1) Open IAR
2) Start the corresponding PRJ (506/618)
3) Insert the SD Card/micro SD Card into corresponding card socket
4) Run the test code and check the result (set the BPs and check)

```
    Sdcard_SpiModeConfig(&Sdcard_stcSdSpiInfo);

    if(Sdcard_stcSdSpiInfo.u8HwFifoFlag == HWFIFO)
    {
        Sdcard_ConfigFifo(&Sdcard_stcSdSpiInfo);
    }
#if 1
    /* SD Card RAW API test code */
    i32RetValue = SdcardInit(&Sdcard_stcSdSpiInfo);

    if(0 == i32RetValue)
    {
        //Initialization success!
        i32RetValue = Sdcard_GetCardSize(&m_u32SdcardSectorNum);
        if(0 != i32RetValue)
        {
            //Fail
            while(1);
        }
        //Write sector test
        i32RetValue = SdcardWriteSector(&Sdcard_stcSdSpiInfo);
        if(0 != i32RetValue)
        {
            //Fail
            while(1);
        }
        //Read sector test
        i32RetValue = SdcardReadSector(&Sdcard_stcSdSpiInfo);
        if(0 != i32RetValue)
        {
            //Fail
            while(1);
        }
    }
```

3.2 SD Card fatFS demo

3.2.1 Code

IO initialization code

```
    Sdcard_Io_Init();
```

SD Card information setting code

```
        /* Config structure init */
        Sdcard_stcSdSpiInfo.u8Channel = SD_MFS_CH;
        Sdcard_stcSdSpiInfo.u8HwFifoFlag = NOHWFIFO;
        Sdcard_stcSdSpiInfo.u8TransState = SpiCommIDLE;
        Sdcard_stcSdSpiInfo.pstcModeConfig = &m_stcSdSpiModeConfig;
        Sdcard_stcSdSpiInfo.pstcFIFOConfig = NULL;

        Sdcard_SpiModeConfig(&Sdcard_stcSdSpiInfo);
```

```
        if(Sdcard_stcSdSpiInfo.u8HwFifoFlag == HWFIFO)
        {
            Sdcard_ConfigFifo(&Sdcard_stcSdSpiInfo);
        }
```

SD Card initialization code

```
        i32RetValue = SdcardInit(&Sdcard_stcSdSpiInfo);

        if(0 == i32RetValue)
        {
         ...
```

fatFS test code

```
        #if 1
        /* fatFS API test */
        /* Call the Test API */

    //    FatFsfmount();
    //    FatFsfopenclose();
    //    FatFsfread();
    //    FatFsfwrite();
    //    FatFsflseek();
    //    FatFsfgetfree();
    //    FatFsunlink();
    //    FatFschmod();
    //    FatFsrename();
    //    FatFsgets();
    //    FatFsputc();
    //    FatFsputs();
        FatFssize();
```

3.2.2 Environment

Same as above.

3.2.3 Demo procedure

1) Open IAR
2) Start the corresponding PRJ (506/618)
3) Insert the SD Card/micro SD Card into corresponding card socket
4) Run the test code and check the result (set the BPs and check)

```
    }
#endif

#if 1
    /* fatFS API test */
    /* Call the Test API */

//    FatFsfmount();
//    FatFsfopenclose();
//    FatFsfread();
//    FatFsfwrite();
//    FatFsflseek();
//    FatFsfgetfree();
//    FatFsunlink();
//    FatFschmod();
//    FatFsrename();
//    FatFsgets();
//    FatFsputc();
//    FatFsputs();
    FatFssize();

    /* End */

#endif
```

4 SD Card Library adaption steps

This SD Card Library can be adapted to any FMx MCU with SPI I/F. Please use the following steps:

1) Define the clock and correlative pin in board.h
2) Check the SdcardChipSelect/SdcardPresent/SdcardWriteProtect in sdcard_hw.c

5 Annex

Pin assignment

board.h		
	506R	618S
CLKHC	(4MHZ)	(4MHZ)
CLKLC	(100KHZ)	(100KHZ)
CLKMO	(4MHZ)	(4MHZ)
CLKSO	(32768UL)	(32768UL)
SD_CS_IO_PORT	IO_PORT4	IO_PORT5
SD_CS_IO_PIN	IO_PINxA	IO_PINx3
SD_CD_IO_PORT	IO_PORT4	IO_PORTO
SD_CD_IO_PIN	IO_PINxB	IO_PINx9
SD_WP_IO_PORT	IO_PORT5	-
SD_WP_IO_PIN	IO_PINxB	-
USER_MFS_CH	MFS_Ch1	MFS_Ch3
SD_MFS_PORT	IO_PORT5	IO_PORT5
SD_MFS_SCK_PIN	IO_PINx8	IO_PINx2
SD_MFS_SOT_PIN	IO_PINx7	IO_PINx1
SD_MFS_SIN_PIN	IO_PINx6	IO_PINx0
SD_MFS_SCK_PIN_LOC	IO_MFS_SCKx_SCKx_0	IO_MFS_SCKx_SCKx_1
SD_MFS_SOT_PIN_LOC	IO_MFS_SOTx_SOTx_0	IO_MFS_SOTx_SOTx_1
SD_MFS_SIN_PIN_LOC	IO_MFS_SINx_SINx_0	IO_MFS_SINx_SINx_1