AVR-SD Card Module Guide

Joshua Fain – 9/28/2020

Table of Contents

***Table of Contents*..1-2**

***1. INTRODUCTION*..1-4**

1.1 Overview 1-4

Getting Started – Initialization..1-4

***2. LISTS of OBJECTS, MACROS, and FUNCTIONS*..2-6**

2.1 FUNCTIONS 2-6

SD_SPI_BASE 2-6

SD_SPI_DATA_ACCESS 2-6

SD_SPI_MISC..2-7

2.2 OBJECTS 2-8

SD_SPI_BASE 2-8

2.3 MACROS..2-8

SD_SPI_BASE 2-8

SD_SPI_DATA_ACCESS 2-12

***3. FUNCTION DETAILS*..3-13**

3.1 SD_SPI_BASE..3-13

uint32_t SD_InitializeSPImode(CardTypeVersion *ctv) 3-13

void SD_SendByteSPI(uint8_t byte) 3-14

uint8_t SD_ReceiveByteSPI(void) 3-14

void SD_SendCommand(uint8_t cmd, uint32_t arg) 3-15

uint8_t SD_GetR1(void)..3-15

void SD_PrintR1(uint8_t r1)..3-16

void SD_PrintInitError(uint32_t err) 3-16

3.2 SD_SPI_DATA_ACCESS 3-18

uint16_t SD_ReadSingleBlock(uint32_t blockAddress, uint8_t *block)..3-18

void SD_PrintSingleBlock(uint8_t *block)..3-18

uint16_t WriteSingleBlock(uint32_t blockAddress, uint8_t *data) 3-19

uint16_t SD_EraseBlocks(uint32_t startBlockAdress, uint32_t endBlockAddress)..3-20

uint16_t SD_PrintMultipleBlocks(uint32_t startBlockAddress, uint32_t numberOfBlocks) 3-20

uint16_t SD_WriteMultipleBlocks(uint32_t startbBlockAddress, uint32_t numberOfBlocks, uint8_t *data)...3-
21

uint16_t SD_GetNumberOfWellWrittenBlocks(uint32_t *wellWrittenBlocks)..3-22

void SD_PrintReadError(uint16_t err) 3-22

void SD_PrintWriteError(uint16_t err)..3-23

void SD_PrintEraseError(uint16_t err)..3-23

WARNINGS and DISCLAIMERS

1. **Use at your own risk**. This SD card module was developed for fun and so it is offered as is to anyone that wants to use it, look at it, or modify it for their needs. There is no guarantee of operability under any circumstance and it is possible to erase or overwrite data, lockout the SD card, and/or get the SD card into a bad logical or physical state. I take no responsibility for the loss of data or property through the use of this SD card module.
2. **Backup Data!** See 1.
3. This module has only been tested on an ATmega1280 microcontroller (µC). It is expected to be easily portable to other AVR µCs through simple port (e.g. SPI, USART) reassignments, provided the resources exist, but also see 1.
4. This module has only been tested against version 2.x, 2GB and 4GB micro-SD cards of type SDSC (standard capacity) and SDHC (high capacity). It is unknown how it will behave running against other SD card types, versions, and capacities. Again, see 1.

1. INTRODUCTION

The AVR-SDCard module has been developed to provide the ability to execute SPI mode SD card commands from an AVR target/host. The module can either be implemented as a standalone raw data access module, though it is originally intended as a physical disk layer for a file system layer. This document is intended to provide a brief overview of the structure of the module and provide the specific functions, flags, and objects available for use. This document should be up-to-date with the current functionality of the module, but refer to the source code if any discrepancies are found.

1.1 Overview

AVR-SD Card Module

This SD Card module is divided into 3 separate source/header files. These are listed below in order of importance/role:

1) SD_SPI_BASE – basic functions required for interaction with the SD card in SPI mode.
2) SD_SPI_DATA_ACCESS – special functions to can handle data access (e.g. R/W/E).
3) SD_SPI_MISC – some miscellaneous functions.

Only SD_SPI_BASE is required for interaction with the SD card. It includes the SD card initialization routine as well as the basic send command and receive byte functions. The functions defined in SD_SPI_DATA_ACCESS are special functions related to data access, e.g. reading, writing, and erasing data blocks and requires SD_SPI_BASE. SD_SPI_MISC is intended to be a catch-all for some miscellaneous functions.

Additional Requirements

The following files are required in order for the module to function, and have been included in the repository, however, they are not considered part of the module and so will not be discussed here. These are maintained in the AVR-General repository.

1) SPI.C / SPI.H – required to interface with the AVR's SPI port used for physical transmission and reception of byte-sized packets to/from the SD Card.
2) USART.C / USART.H – required to interface with the AVR's USART port used to print messages and data to a terminal/screen.
3) PRINTS.C / PRINTS.H – required to print integers (decimal, hex, binary) and strings to the screen via the USART.

Initializing the SD Card

Any implementation of this module must first initialize the USART and SPI ports. Once these have been initialized, a new CardTypeVersion struct object must be created and passed as a pointer to the SD card initialization routine [*SD_InitializeSPImode(CardTypeVersion *ctv)*] which will initialize the card into SPI mode. The initialization function will return an error response to indicate whether initialization was successful or failed. This error response can then be read with a 'Print Error' function to indicate if and where the routine failed. If the initialization was successful it will be in the OUT_OF_IDLE in the R1 response and INIT_SUCCESS in the initialization error response. The application can then proceed with calling other commands. The SD card initialization routine will also set the *CardTypeVersion* struct argument's *type* and *version* members which are used to handle how some other functions are called, particularly whether byte or block addressing should be used when calling a 'data access' function.

NOTES:

1. The initialization response is separated into an *Initialization Error Response* and *R1 response* (more details later). These can be printed using the *SD_PrintInitError()* and *SD_PrintR1()* functions, respectively.
2. A standard capacity card (SDSC) is byte addressable. A high capacity card (SDHC / SDXC) is block addressable. This means that when calling a block access function, such as SD_ReadSingleBlock(addr, *block), the address passed as argument 1 for type SDHC should be the block number, whereas for type SDSC the address should be the block number multiplied by the BLOCK_LEN (i.e. the number of bytes in a block which should be 512).

Example: Initialization and Read/Print Block

The following is an example of how to use the *SD_InitializeSPImode()* function to initialize the SD card into SPI mode. It then proceeds to check if the initialization was successful or not. If successful, it proceeds to execute a 'read block' and 'print block' function.

```
int main(void)
{
   USART_Init();
   SPI_MasterInit();

   CardTypeVersion ctv;

   uint32 initResponse;

   initResponse = SD_InitializeSPImode(&ctv);

   if (((initResponse & 0xFF)!= OUT_OF_IDLE) &&
         (initResponse&0xFFF00) != INIT_SUCCESS)
   {
      //FAILED initialization
      SD_PrintR1(initResponse);
      SD_PrintInitError(initResponse);

      //handle failed initialization
   }
   else
   {
     // SD Card successfully initialized.
     // Read Block
     uint32_t blockNumber = 5; // block to read
     uint8_t block[512]; //array to hold contents of blockNumber
     uint16_t err;

     // Read block 5 of SD card into block[] array.
     if(ctv.type == SDHC) // block addressable
        err = SD_ReadSingleBlock(blockNumber, block);
     else // ctv.type == SDSC) // byte addressable
        err = SD_ReadSingleBlock(blockNumber * BLOCK_LEN, block);

     //Handle any read errors here. Left out of example.

     // Print contents on block[] array which now holds the data
     // from block 5 on the SD card.
     SD_PrintSingleBlock(block);
   }
}
```

2. LISTS of OBJECTS, MACROS, and FUNCTIONS

2.1 FUNCTIONS

SD_SPI_BASE

RETURN TYPE	**FUNCTION**
uint32_t	SD_InitializeSPImode(CardTypeVersion *ctv)
void	SD_SendByteSPI(uint8_t byte)
uint8_t	SD_ReceiveByteSPI(void)
void	SD_SendCommand(uint8_t cmd, uint32_t arg)
uint8_t	SD_GetR1(void)
void	SD_PrintR1(uint8_t r1)
void	SD_PrintInitError(uint32_t err)

SD_SPI_DATA_ACCESS

RETURN TYPE	**FUNCTION**
uint16_t	SD_ReadSingleBlock(uint32_t blockAddress, uint8_t *block)
void	SD_PrintSingleBlock(uint8_t *block)

uint16_t	SD_WriteSingleBlock(uint32_t blockAddress, uint8_t *data)
uint16_t	SD_EraseBlocks(uint32_t startBlockAddress, uint32_t endBlockAddress)
uint16_t	SD_PrintMultipleBlocks(uint32_t startBlockAddress, uint32_t numberOfBlocks)
uint16_t	SD_WriteMultipleBlocks(uint32_t startBlockAddress, uint32_t numberOfBlocks, uint8_t *data)
uint16_t	SD_GetNumberOfWellWrittenBlocks(uint32_t *wellWrittenBlocks)
void	SD_PrintReadError(uint16_t err)
void	SD_PrintWriteErrors(uint16_t err)
void	SD_PrintEraseError(uint16_t err)

SD_SPI_MISC

RETURN TYPE	**FUNCTION**
uint32_t	SD_GetMemoryCapacity(void)
void	SD_PrintNonZeroBlockNumbers(uint32_t startBlock, uint32_t endBlock)

2.2 OBJECTS

SD_SPI_BASE

TYPE	**KIND**
CardTypeVersion	struct

2.3 MACROS

SD_SPI_BASE

SPI_Mode SD Card Commands

COMMAND	**VALUE**
GO_IDLE_STATE	0
SEND_OP_COND	1
SWITCH_FUNC	6
SEND_IF_COND	8
SEND_CSD	9
SEND_CID	10
STOP_TRANSMISSION	12
SEND_STATUS	13
SET_BLOCKLEN	16
READ_SINGLE_BLOCK	17
READ_MULTIPLE_BLOCK	18
WRITE_BLOCK	24
WRITE_MULTIPLE_BLOCK	25
PROGRAM_CSD	27
SET_WRITE_PROT	28
CLR_WRITE_PROT	29
SEND_WRITE_PROT	30
ERASE_WR_BLK_START_ADDR	32
ERASE_WR_BLK_END_ADDR	33
ERASE	38
LOCK_UNLOCK	42
APP_CMD	55
GEN_CMD	56
READ_OCR	58
CRC_ON_OFF	59

Application Specific Command

Must call command APP_CMD prior to calling any of these App Commands

APP COMMAND	**VALUE**
SD_STATUS	13
SEND_NUM_WR_BLOCKS	22
SET_WR_BLK_ERASE_COUNT	23
SD_SEND_OP_COND	41
SET_CLR_CARD_DETECT	42
SEND_SCR	51

Card Types – Standard or High Capacity

Card Type	**VALUE**
SDSC	0
SDHC	1

Assert / De-assert CS

CS is the SD card's pin connected to the SS pin of the SPI port. When CS is asserted it signals to the card to prepare for a command. De-asserting stops communication with the SD card.

CS STATE	**ACTION**
CS_LOW	Asserts CS
CS_HIGH	De-asserts CS

R1 Response Flags

An R1 response is the first byte returned by an SD card in response to any command. With the exception of OUT_OF_IDLE and R1_TIMEOUT, the values of these flags are the SD card's R1 responses. Use *SD_GetR1()* to get the R1 response after sending a command, and SD_PrintR1() to read response.

- OUT_OF_IDLE is defined here to be the state when no flags are set in the SD card's R1 response (i.e. No errors are returned in R1 and the card is not IN_IDLE_STATE).
- R1_TIMEOUT: bit 7 of an SD card's R1 response is reserved (always 0). The *SD_GetR1()* makes use of this bit by indicating a timeout waiting for the R1 response.

R1 FLAGS	**VALUE**
OUT_OF_IDLE	0x00
IN_IDLE_STATE	0x01
ERASE_RESET	0x02
ILLEGAL_COMMAND	0x04
COM_CRC_ERROR	0x08
ERASE_SEQUENCE_ERROR	0x10
ADDRESS_ERROR	0x20
PARAMETER_ERROR	0x40
R1_TIMEOUT	0x80

Initialization Error Flags

Initialization error flags are returned in bits 8 to 16 of SD_InitializationSPImode() ' s returned value the lowest byte is set to zero to accommodate the R1 portion of the response. Use SD_PrintInitError() to read the initialization error flags.

INITIALIZATION ERROR FLAGS	**VALUE**
INIT_SUCCESS	0x00000
FAILED_GO_IDLE_STATE	0x00100
FAILED_SEND_IF_COND	0x00200
UNSUPPORTED_CARD_TYPE	0x00400
FAILED_CRC_ON_OFF	0x00800
FAILED_APP_CMD	0x01000
FAILED_SD_SEND_OP_COND	0x02000
OUT_OF_IDLE_TIMEOUT	0x04000
FAILED_READ_OCR	0x08000
POWER_UP_NOT_COMPLETE	0x10000

High Capacity Support

Set to 1 if the host should support high capacity cards (SDHC / SDXC). Set to 0 if host should only support standard capacity cards (SDSC).

MACRO	**DEFAULT VALUE**
HCS	1

Block Length

Defines the expected length, in bytes, of an SD card block. This value should always be 512. No other values for this have been tested, and changing this will likely make the module unusable.

MACRO	**DEFAULT VALUE**
BLOCK_LEN	512

SD_SPI_DATA_ACCESS

Read Error Flags
Flags returned by *Read Block* commands in SD_SPI_DATA_ACCESS. The lowest byte is 0 to accommodate the R1 portion of the response.

READ ERROR FLAGS	**VALUE**
START_TOKEN_TIMEOUT	0x0200
READ_SUCCESS	0x0400

Write Error Flags
Flags returned by *Write Block* commands in SD_SPI_DATA_ACCESS. The lowest byte is 0 to accommodate the R1 portion of the response.

WRITE ERROR FLAGS	**VALUE**
DATA_ACCEPTED_TOKEN	0x00100
CRC_ERROR_TOKEN	0x00200
WRITE_ERROR_TOKEN	0x00400
INVALID_DATA_RESPONSE	0x00800
DATA_RESPONSE_TIMEOUT	0x01000
CARD_BUSY_TIMEOUT	0x02000

Erase Error Flags
Flags returned by *Write Block* commands in SD_SPI_DATA_ACCESS. The lowest byte is 0 to accommodate the R1 portion of the response.

ERASE ERROR FLAGS	**VALUE**
ERASE_SUCCESSFUL	0x00100
SET_ERASE_START_ADDR_ERROR	0x00200
SET_ERASE_END_ADDR_ERROR	0x00400
ERROR_ERASE	0x00800
ERASE_BUSY_TIMEOUT	0x01000

R1 Error Flag

This flag is used by function in SD_SPI_DATA_ACCESS to indicate the returned value is an R1 an R1 response.

Error Flag	VALUE
R1_ERROR	0x8000

3. FUNCTION DETAILS

3.1 SD_SPI_BASE

uint32_t SD_InitializeSPImode(CardTypeVersion *ctv)

DESCRIPTION

Initializes an SD card into SPI mode. It also determines the SD card's type and version and sets the CardTypeVersion stuct object's members accordingly. Only this function should set the CardTypeVersion members and once set they should not be changed. The SD card is successfully initialized if INIT_SUCCESS and OUT_OF_IDLE are the initialization error and R1 portions of the initialization reponse, respectively.

ARGUMENT

Pointer to CardTypeVersion struct

The struct object's members will be set by this function according to the card's type and version.

RETURNS

uint32_t

The returned value is the initialization response and is divided into two portions:

Initialization Error Response [8:16]

This portion of the response occupies bits 8 to 16 of the returned value, and corresponds to the *Initialization Error Flags.* Pass the returned value to *SD_PrintInitError()* to read this portion of the response

R1 Response [0:7]

This portion of the response occupies the lowest byte (bits 0 to 7) of the returned value, and correspond to the R1 Response Flags. This represents the most recently returned R1 response from the SD card retrieved using the SD_GetR1() during the initialization function. Use SD_PrintR1() to print the R1 portion of the response.

NOTE: bits 17 to 31 in the returned value are currently not used and should be 0.

EXAMPLE
```
CardTypeVersion ctv;
uint32_t err = SD_InitializeSPImode(CardTypeVersion *cvt);
SD_PrintR1(err);
SD_PrintInitError(err);
```

---

void SD_SendByteSPI(uint8_t byte)

DESCRIPTION
Sends a single byte to the SD card via SPI. This function, along with SD_ReceiveByteSPI() are the SPI interfacing functions. This function is used by SD_SendCommand() to send the command / argument in byte sized packets to the SD card.

ARGUMENTS
uint8_t

The next byte to be sent to the SD card.

RETURNS
none

---

uint8_t SD_ReceiveByteSPI(void)

DESCRIPTION
Gets the next byte returned by the SD card via SPI. This function, along with SD_SendByteSPI() are the SPI interfacing functions.

ARGUMENT
none

RETURN
uint8_t

The next byte returned by the SD card.

---

void SD_SendCommand(uint8_t cmd, uint32_t arg)

DESCRIPTION
Sends an SD card command, argument to the SD card.

ARGUMENTS
(1) *uint8_t*

A valid SPI mode SD card command (cmd) from the list.

(2) *uint32_t*

The argument (arg) to be sent with the command.

RETURNS
none

NOTE
This function also sends a valid CRC7 token to the SD card. The CRC7 token is generated by calling a "private" function, however, in this current version, the CRC value does not matter (except for CMD0), as the CRC check is turned off (default). It should not be turned on because handling of other CRC values (e.g. on write) is currently not supported.

---

uint8_t SD_GetR1(void)

DESCRIPTION
Gets the R1 response returned by an SD card for a given command / argument. This function should be called immediately after SD_SendCommand() to get the R1 response.

ARGUMENT
none

RETURNS
uint8_t

The R1 response byte returned by the SD card to a given command. This function will only return a valid response if it is called immediately after a command has been sent to the SD card. If any other function queries the SPI Data Register (SPDR) after sending a command, then the R1 response returned by this function will be may not correspond to the actual R1 response. Call SD_PrintR1() to print the returned value.

---

void SD_PrintR1(uint8_t r1)

DESCRIPTION:
Prints the R1 response returned by SD_GetR1()

ARGUMENT:
uint8_t

The R1 byte response returned by SD_GetR1().

RETURN
none

NOTES
The actual SD card's R1 response occupies bits 0 to 6 of the first byte returned by an SD card in SPI mode; bit 7 is reserved and should be 0 when returned by the SD card. The function, SD_GetR1(), however, uses bit 7 as the R1_TIMEOUT flag to indicate the SD card did not return a response within an acceptable amount of time or number of attempts. If R1 response is 0, then the card is not in idle and no errors were returned in the R1 response. In this case OUT_OF_IDLE state will be the 'value' of the R1 response. During initialization, the card should be in the idle state and no error means only the IN_IDLE_STATE flag is set.

---

void SD_PrintInitError(uint32_t err)

DESCRIPTION
Prints the initialization error response flags returned by SD_InitializeSPImode(). This function does not print the R1 response portion of the initialization response.

ARGUMENT
uint32_t

The Initialization Error Response portion of the initialization routine's [SD_InitializeSPImode()] returned value. This function will filter out the R1 portion of the response, so passing the entire 32-bit returned value of the initialization routine is valid.

RETURNS
none

NOTES
The R1 portion SD_InitializeSPImode() portion of the response should be passed to SD_PrintR1() in order to read it's values.

3.2 SD_SPI_DATA_ACCESS

uint16_t SD_ReadSingleBlock(uint32_t blockAddress, uint8_t *block)

DESCRIPTION
Reads in a single 512-byte block from the SD card from location ‘blockAddress’, and stores the contents of the block in the array pointed at by ‘*block’.

ARGUMENTS
(1) *uint32_t blockAddress*

The address on the SD card of the block to be read. A single block in this implementation is always 512 bytes (value of BLOCK_LEN). If the card is SDSC then the card is byte addressable. If the card is SDHC/SDXC then the card is block addressable.

(2) *uint8_t *block*

A pointer to a *uint8_t* array of length 512 (BLOCK_LEN). The function will update the contents of the array with the values of the single byte block at the address specified by argument 1.

RETURNS
uint16_t

Read Block Error Response. This response is composed of the *Read Block Error Flags* in the upper byte, and the R1 response returned after sending the READ_SINGLE_BLOCK command in the lower byte. Read the *Read Block Error Flags* returned using SD_PrintReadError(err). If the R1_ERROR flag is set in the upper portion of the response, then an R1 error occurred. The R1 portion can be read by using SD_PrintR1(R1).

---

void SD_PrintSingleBlock(uint8_t *block)

DESCRIPTION
This function should be used in conjunction with SD_ReadSingleBlock() to print the contents of the block that was read in by that function. The contents must be stored in the 512-byte length array pointed at by the argument.

This function prints the hexadecimal and ASCII characters of the block's contents in rows of 16-bytes. The beginning of each row is the byte offset of the first byte of the row.

ARGUMENT
*uint8_t *block*

A pointer to a *uint8_t array* of length 512 (BLOCK_LEN) that contains the contents of an SD card block stored in the array by the SD_ReadSingleBlock() function.

RETURNS
none

---

uint16_t WriteSingleBlock(uint32_t blockAddress, uint8_t *data)

DESCRIPTION
Writes data to a single block on the SD card.

ARGUMENTS
(1) *uint32_t blockAddress*

The address on the SD card of the block to write the data to. A single block in this implementation is always 512 bytes (value of BLOCK_LEN). If the card is SDSC then the card is byte addressable. If the card is SDHC/SDXC then the card is block addressable.

(2) *uint8_t *data*

A pointer to a *uint8_t* array of length 512 (BLOCK_LEN) that contains the contents of the data to write to the SD card.

RETURNS
uint16_t

Write Block Error Response. This response is composed of the *Write Block Error Flags* in the upper byte, and the R1 response returned after sending the WRITE_BLOCK command in the lower byte. Read the *Write Block Error Flags* returned by using SD_PrintWriteError(err). If the R1_ERROR flag is set in the upper portion of the response, then an R1 error occurred. The R1 portion can be read by using SD_PrintR1(R1).

---

uint16_t SD_EraseBlocks(uint32_t startBlockAdress, uint32_t endBlockAddress)

DESCRIPTION
Erases the blocks between, and including, startBlockAdress and endBlockAddress.

ARGUMENTS
(1) *uint32_t startBlockAddress*

The address of the first block that will be erased. If the card is SDSC then the card is byte addressable. If the card is SDHC/SDXC then the card is block addressable.

(2) *uint32_t endBlockAddress*

The address of the last block that will be erased, and must be greater than the start address. If the card is SDSC then the card is byte addressable. If the card is SDHC/SDXC then the card is block addressable.

RETURNS
uint16_t

Erase Error Response. This response is composed of the *Erase Error Flags* in the upper byte, and the R1 response returned after sending on of the required commands (ERASE_WR_BLK_START_ADDR, ERASE_WR_BLK_END_ADDR, or ERASE) in the lower byte. Read the *Erase Error Flags* returned by using SD_PrintEraseError(err). If the R1_ERROR flag is set in the upper portion of the response, then an R1 error occurred. The R1 portion can be read by using SD_PrintR1(R1).

---

uint16_t SD_PrintMultipleBlocks(uint32_t startBlockAddress, uint32_t numberOfBlocks)

DESCRIPTION
Prints the contents of a specified number of consecutive blocks from the SD card beginning at start block address. This function operates by sending the READ_MULTIPLE_BLOCK command and then calling the SD_PrintSingleBlock() function to print a block's contents every time a new block is returned by the SD card, until all the specified blocks have been read in.

ARGUMENTS
(1) *uint32_t startBlockAddress*

The address of the first block to print. If the card is SDSC then the card is byte addressable. If the card is SDHC/SDXC then the card is block addressable.

(2) *uint32_t numberOfBlocks*

The number of blocks to print to the screen, beginning at startBlockAddress.

RETURNS

Read Block Error Response. This response is composed of the *Read Block Error Flags* in the upper byte, and the R1 response returned after sending the READ_MULTIPLE_BLOCK command in the lower byte. Read the *Read Block Error Flags* returned using SD_PrintReadError(err). If the R1_ERROR flag is set in the upper portion of the response, then an R1 error occurred. The R1 portion can be read by using SD_PrintR1(R1).

---

uint16_t SD_WriteMultipleBlocks(uint32_t startbBlockAddress, uint32_t numberOfBlocks, uint8_t *data)

DESCRIPTION
Writes the contents of the array pointed at by *data to multiple consecutive blocks. This function is not that useful as it writes the same data to multiple blocks, but it is mainly intended to be a demo of the WRITE_MULTIPLE_BLOCK command.

ARGUMENTS
(1) *uint32_t startBlockAddress*

The address of the first block that will be written to. If the card is SDSC then the card is byte addressable. If the card is SDHC/SDXC then the card is block addressable.

(2) *uint32_t numberOfBlocks*

Specifies the total number of blocks that will be written to.

(3) *uint8_t *data*

Pointer to a uint8_t array of length 512 (BLOCK_LEN) that contains the data that will be written to the blocks.

RETURNS

Write Block Error Response. This response is composed of the *Write Block Error Flags* in the upper byte, and the R1 response returned after sending the WRITE_MULTIPLE_BLOCK command in the lower byte. Read the *Write Block Error Flags* returned by using SD_PrintWriteError(err). If the R1_ERROR flag is set in the upper portion of the response, then an R1 error occurred. The R1 portion can be read by using SD_PrintR1(R1).

---

uint16_t SD_GetNumberOfWellWrittenBlocks(uint32_t *wellWrittenBlocks)

DESCRIPTION
This function should be called if the WRITE_ERROR_TOKEN was returned by the SD card while carrying out the WRITE_MULTIPLE_BLOCK command. If this token was returned by the card when calling the function SD_WriteMultipleBlocks() then the WRITE_ERROR_TOKEN_FLAG will be set in its return response, in which case this function should be called.

ARGUMENT
*uint32_t *wellWrittenBlocks*

Pointer to a value of type uint32_t that will be updated with the value of the number of well written blocks returned by the SD card after upon sending the SEND_NUM_WR_BLOCKS (ACMD22).

RETURNS
This function executes similarly to a read block function so it will return a Read Block Error Response. This response is composed of the *Read Block Error Flags* in the upper byte, and the R1 response returned after sending APP_CMD or SEND_NUM_WR_BLOCKS commands in the lower byte. Read the *Read Block Error Flags* returned using SD_PrintReadError(err). If the R1_ERROR flag is set in the upper portion of the response, then an R1 error occurred. The R1 portion can be read by using SD_PrintR1(R1).

---

void SD_PrintReadError(uint16_t err)

DESCRIPTION
Prints *Read Block Error Flags*

ARGUMENT
uint16_t err

Read Block Error Response

RETURNS
none

---

void SD_PrintWriteError(uint16_t err)

DESCRIPTION
Prints *Write Block Error Flags*

ARGUMENT
uint16_t err

Write Block Error Response

RETURNS
none

---

void SD_PrintEraseError(uint16_t err)

DESCRIPTION
Prints *Erase Error Flags*

ARGUMENT
uint16_t err

Erase Error Response

RETURNS
none