



APPLICATION NOTE 4068

Interfacing SD Cards with the TINI System

Abstract: This application note describes how to use the Secure Digital (SD) media format to expand the DS80C400 microcontroller's nonvolatile data storage. The DS80C400 evaluation (EV) kit and the TINIOS are used as an example development platform.

Maxim's TINI® platform is a microcontroller-based system that executes code for embedded web servers. Combining a broad-based I/O, a full TCP/IP stack, and an extensible Java™ runtime environment, the TINI platform simplifies development of the network-connected equipment.

This application note describes the steps required to connect a Secure Digital (SD) card to the [DS80C400](#) microcontroller's Evaluation (EV) Kit by using the TINI Java runtime libraries to expand the EV kit's nonvolatile data-storage capacity. The techniques described here can easily be applied to other designs that use the DS80C400.

Design Considerations for Using an SD Memory Card

The first design consideration when using an SD memory card is the supply-voltage requirement. SD cards require an operating voltage between 2.0V and 3.6 V. The DS80C400 EV kit supplies a 3.3V connection, which will be adequate for the application.

Second, determine which of the DS80C400's I/O lines will be used to connect the external storage. Ideally the minimum number of I/Os will be used, while still providing sufficient bandwidth for the intended application. SD cards provide two options for communication: first, a proprietary SD bus, which is a six-wire communication channel (clock, command, and four data lines); and second, the SPI™ bus, which uses four wires (clock, data in, data out, and chip select). Since Maxim already provides an SPI library for the DS80C400 and the SPI bus requires fewer I/O lines, the SPI bus is the option used in this application note. There is, however, a disadvantage to the SPI bus: slower communication speeds.

Finally, determine the amount of storage to be added. SD card capacities range from a few megabytes up to a maximum of 4 gigabytes. This wide range of available sizes provides ample external storage for many applications.

Connecting the SD Card to the DS80C400 EV Kit

The EV kit board includes an SPI header (J21) for easy connection. Simply connect the SD card to this header, as described in **Table 1**. The remaining pins are not used and can be left unconnected.

Table 1. SD Card Connections to the DS80C400

SD Card	DS80C400 SPI Header J21
CLK	Pin 1
DI	Pin 3
V _{CC}	Pin 4
DO	Pin 5
GND	Pin 6
CS	Pin 7

SPI Data Format for the SD Card

The SD card requires full-duplex, 8-bit, SPI operation. Data is clocked into the card's DI pin from the DS80C400's MOSI pin, and out of the card's DO line into the MISO pin of the DS80C400. Data is clocked simultaneously in and out of the card on the rising edge of the CLK line. Eight extra clock cycles must be provided at the end of each transaction to permit the SD card to complete any outstanding operations. The data input during these extra clock cycles must be all 1s. The clock rate must be limited to a maximum of 400kHz during the identification phase, but can be increased up to 25MHz once the SD card has been identified.

The DS80C400's SPI library is easily configured to match the requirements of the SD card interface. The default settings for the library set the SPI module to latch data on rising clock edges, set the data length to eight bits, and configure the DS80C400 as the SPI master.

The SD card's SPI protocol is similar to its SD bus protocol. Instead of receiving valid data from the SD card's DO pin at every clock edge, a card with no data to send will hold the DO pin at an idle state of all 1s. When the card has data to send back to the host, specialized tokens with a 0 start bit are sent before the data. All data transmitted from the SD card is sent immediately after these tokens, and is of fixed length. As the receiver knows the number of bytes to expect, no length bytes are contained in the response. Additionally, as the idle state cannot occur until after the start token and data have been sent, all data bytes are transmitted unaltered and unprefixed. Tokens, as with all other traffic on the bus, are aligned on the 8-bit boundaries of the SPI transaction. Commands and data from the host to the card follow a similar format, with all 1s denoting an idle bus. All transactions except status tokens are protected by a cyclic redundancy check (CRC) code appended to the end of data. Two CRC algorithms are provided: CRC-7 for short blocks of data, and CRC-16 for longer blocks of data. The CRC is an optional part of the SD SPI interface, but should be used to guarantee data integrity unless application constraints prevent its use.

Cyclic Redundancy Check

The CRC algorithm is commonly used to detect errors induced by an unreliable communication channel. The selection of a particular CRC is governed by the size of data to be protected. In the case of SD media, CRC-7 and CRC-16 are specified.

The CRC algorithm divides the protected data by a selected divisor and produces a remainder. This division is done without carry logic because of the polynomial math used in the algorithm. As no carries are needed, division can be accomplished with the logical XOR operation. The selected divisor is commonly referred to as the CRC's polynomial. The resulting remainder is then transmitted with the data, and can be used by the receiver to check that the data did corrupt during transmission.

In the case of CRC-7, the remainder can be calculated using a 7-bit shift register in software. This shift register is initialized to all 0s at the start of the calculation. As each bit (MSB first) of the protected data is shifted into the LSB of the shift register, the MSB of the shift register is shifted out and examined. If the bit just shifted out is one, the contents of the shift register are modified by XORing with the CRC-7 polynomial value 0x09. If the bit shifted out of the shift register is 0, no XORing is performed. Once the last bit of protected data is shifted in and the conditional XORing completed, six more 0s must be shifted through in a similar manner. This process is referred to as augmentation, and completes the polynomial division. At this point, the CRC-7 value can be read directly from the shift register.

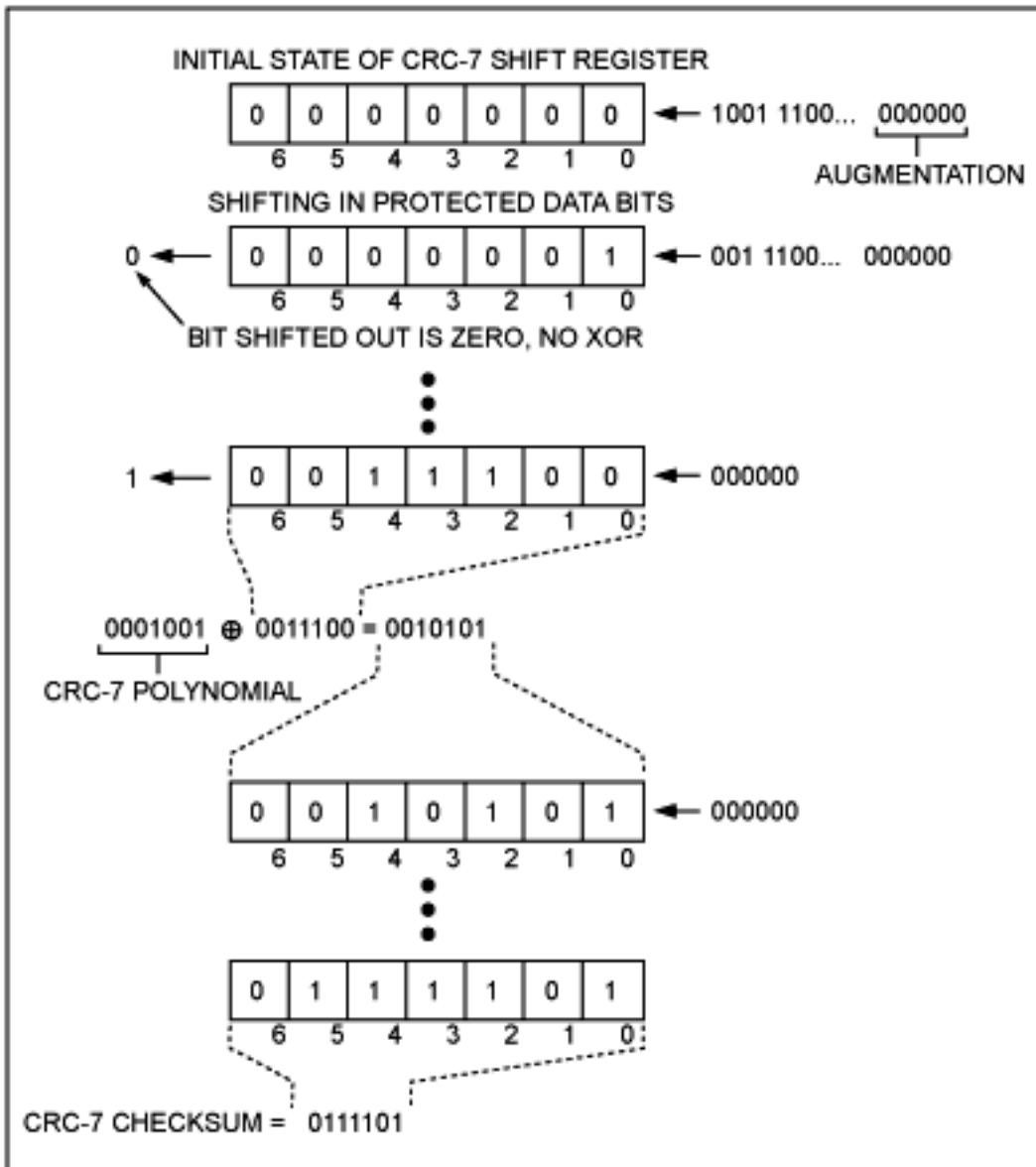


Figure 1. The CRC-7 can be calculated with a shift-register architecture.

When the receiver has all the protected data, the receiver can compute a CRC-7 value over the protected data and compare this with the received CRC-7 value. If the values differ, the receiver knows that the protected data was corrupted during transmission. If the values match, the receiver knows—with a high degree of certainty—that the communications channel did not compromise data integrity.

The CRC-16 algorithm can be constructed in the same manner. In this case, however, the length of the shift register is 16 bits instead of 7, the polynomial value is changed to 0x1021, and the input data is augmented by 16 zero bits.

SD Command Format

Commands are issued to the card in a 6-byte format (**Figure 2**). The first byte of a command can be constructed by ORing the 6-bit command code with hex 0x40. The next four bytes provide a single 32-bit argument, if required by the command. The final byte contains the CRC-7 checksum over bytes 1 through 5.

Table 2 lists important SD commands.

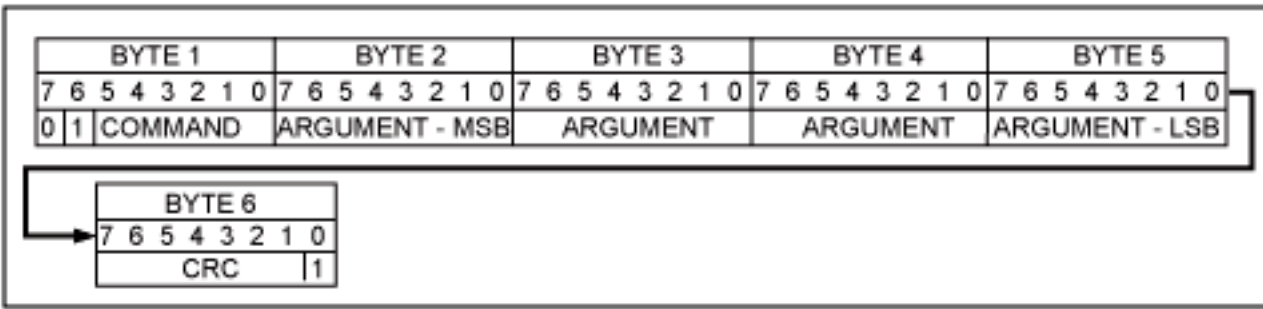


Figure 2. SPI-mode SD commands are issued to the card in a 6-byte format.

Table 2. Selected SD Memory Card Commands

Command	Mnemonic	Argument	Reply	Description
0 (0x00)	GO_IDLE_STATE	None	R1	Resets the SD card.
9 (0x09)	SEND_CSD	None	R1	Sends card-specific data.
10 (0x0A)	SEND_CID	None	R1	Sends card identification.
17 (0x11)	READ_SINGLE_BLOCK	Address	R1	Reads a block at byte address.
24 (0x18)	WRITE_BLOCK	Address	R1	Writes a block at byte address.
55 (0x37)	APP_CMD	None	R1	Prefix for application command.
59 (0x3B)	CRC_ON_OFF	Only Bit 0	R1	Argument sets CRC on (1) or off (0).
41 (0x29)	SEND_OP_COND	None	R1	Starts card initialization.

Initializing the SD Card in SPI Mode

At power-up, the SD card defaults to the proprietary SD bus protocol. To switch the card to SPI mode, the host issues command 0 (GO_IDLE_STATE). The SD card detects SPI mode selection by observing that the active-low card select (active-low CS) pin is held low during the GO_IDLE_STATE command. The card responds with response format R1 (Figure 3). The idle state bit is set high to signify that the card has entered idle state.

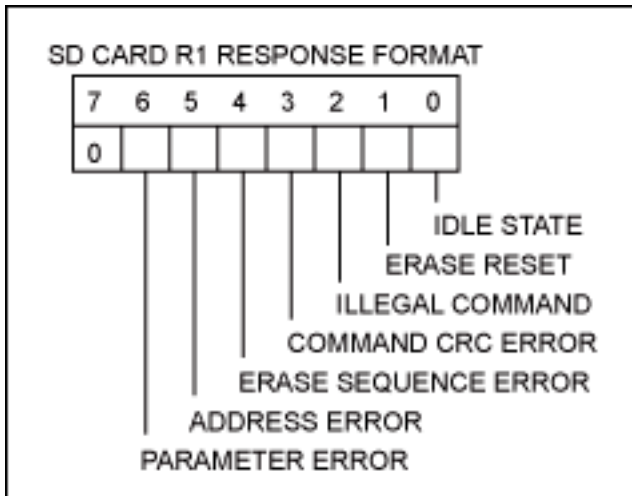


Figure 3. Response format R1 signals the success or failure of the issued command.

Now that the SD card is in SPI mode, the SD specification requires that the host issue an initialization command before any other requests can be processed. Now is also the time to ensure that the connected card is actually an SD card and not a MultiMediaCard (MMC). MMCs have the same form factor as SD cards, but support a different set of commands and have a different feature set. To differentiate between MMC and SD cards, SD cards implement an alternative initialization command to which MMC cards do not respond. Sending command 55 (APP_CMD) followed by application command 41 (SEND_OP_COND) to the card completes this important step. As MMC cards do not respond to command 55, that command can be used to reject MMC cards as invalid media. This command sequence is repeated until all bits in the R1 response from the card are 0 (i.e., the IDLE

bit goes low). For this to work, the SPI clock rate must not exceed 400KHZ at this stage.

The SD card contains several important registers that provide information about the SD card. The most important register is the Card Specific Data register (CSD). For our sample application we are interested in the block size and total size of the memory. We must also pay attention to the Card Identification register (CID), as it contains details about the card's manufacturer and serial number. **Figure 4** shows the layout of the CSD and CID registers.

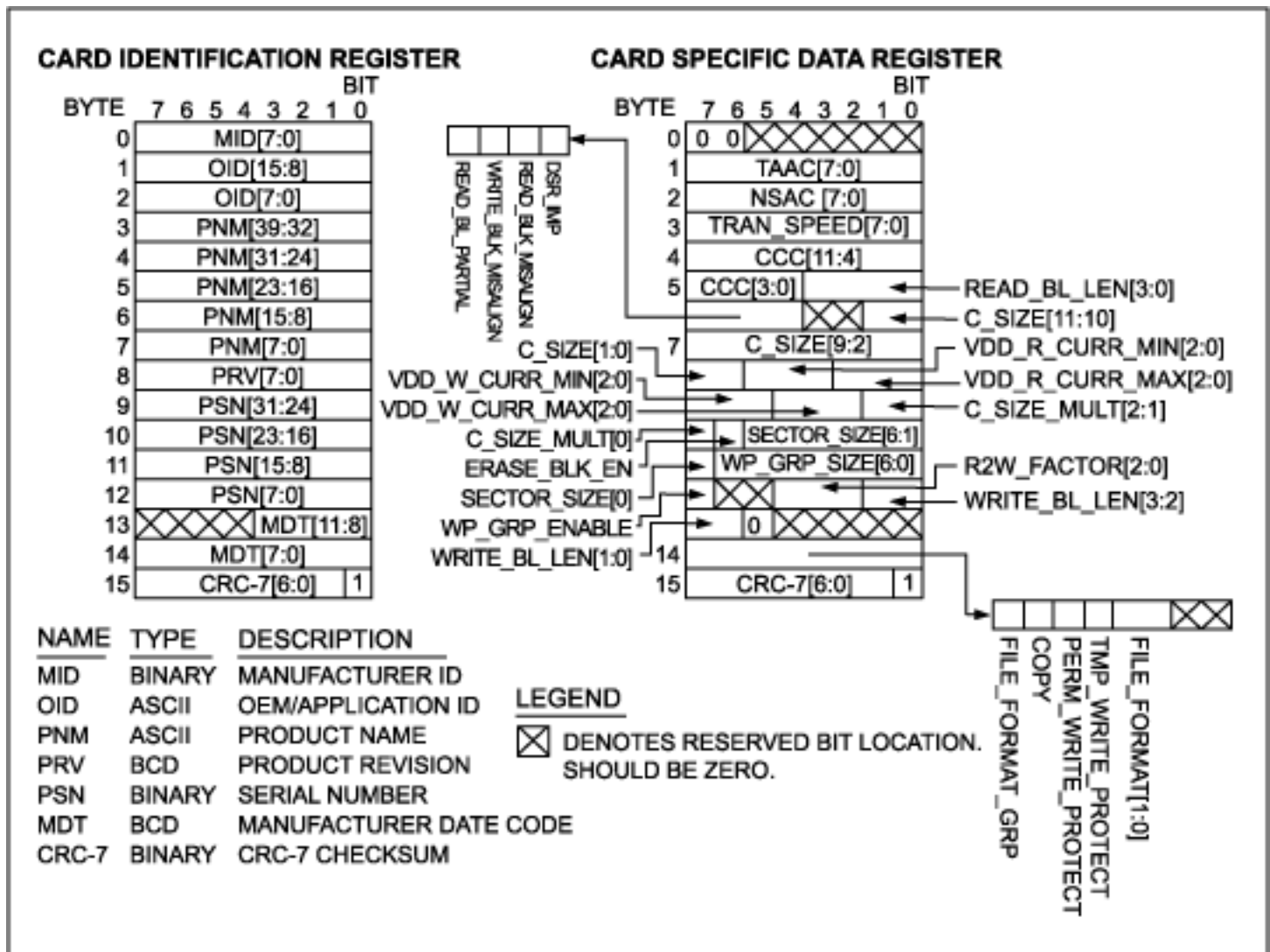


Figure 4. The CSD and CID registers provide information about the SD card.

Examining the SD Card Responses

To read card registers or blocks from the card, we must first understand how the card responds to our inquiries. In SPI mode, the SD card replies to the commands SEND_CSD (9), SEND_CID (10), and READ_SINGLE_BLOCK (17) with an R1 format reply. A start token, the requested data, and finally a CRC-16 checksum over the data follow. We must not assume that the R1 reply and the data start token occur immediately one after the other, as the bus can go to the idle state for some time between these two events. **Figure 5** details the data response.

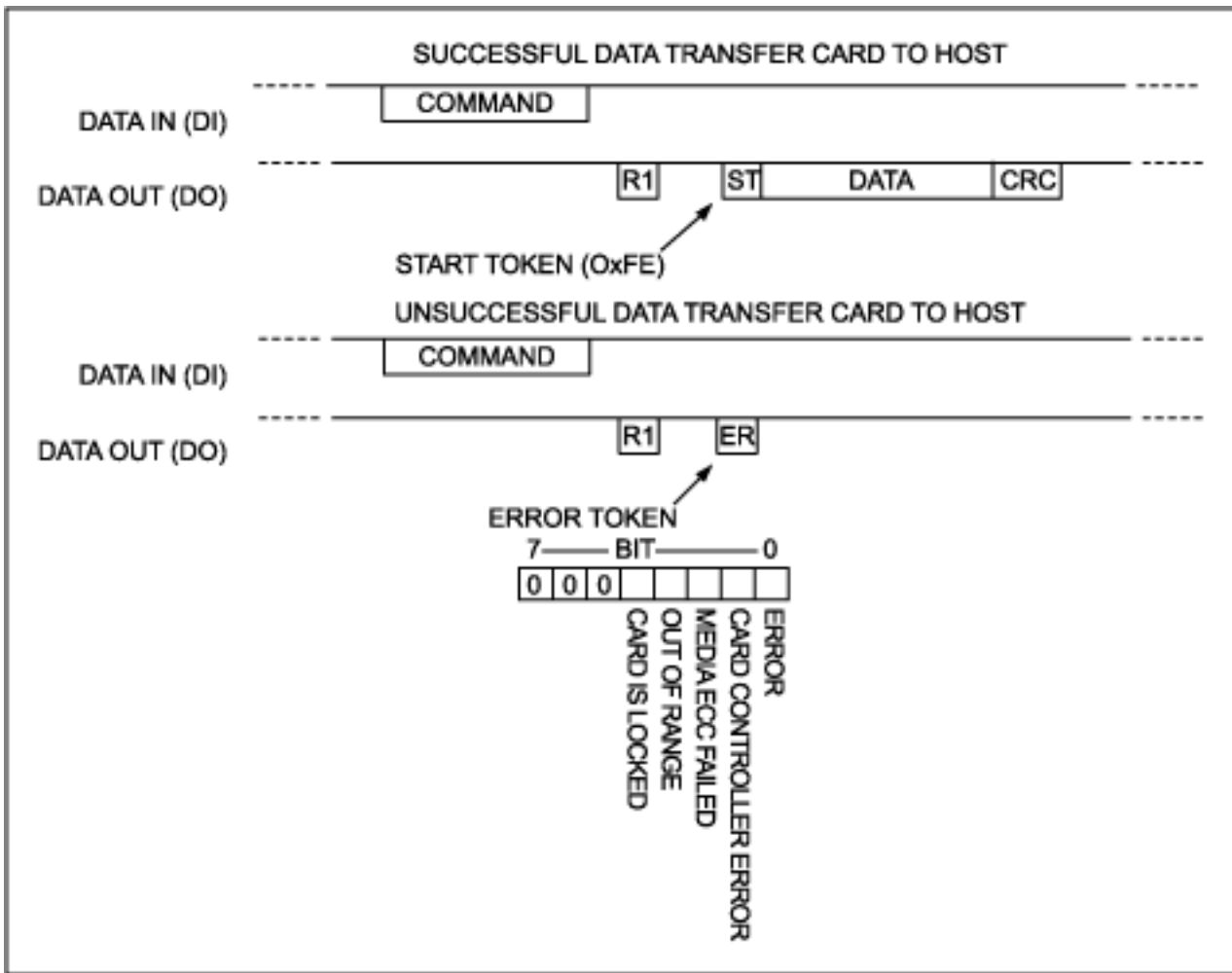


Figure 5. Data transfers from the SD card to the host are prefixed by a start token.

Reading the CSD and CID Registers' Meta-Data

The SEND_CSD and SEND_CID commands send back register contents used to determine the SD card parameters. These commands return a fixed number of bytes, which corresponds to the size of the CSD or CID register, respectively. The argument contained within the command bytes is ignored by the SD card for these SEND commands.

Reading a Block of Data from the SD Card

Reading a block of data from the SD card is quite simple. The host issues the READ_SINGLE_BLOCK command with a starting byte address as the argument. This address must be aligned with the beginning of a block on the media. The SD card then evaluates this byte address and responds back with an R1 command reply. An out-of-range address is indicated in the command reply.

If the read is completed from the SD media without error, a start data token is sent followed by a fixed number of data bytes and two bytes for the CRC-16 checksum. The start data token is not sent if the SD card encounters a hardware failure or media read error. Rather, an error token is sent and the data transfer is aborted.

Writing a Block of Data to the SD Card

Writing a block of data is similar to reading, as the host must supply a byte address that is aligned with the SD card block boundaries. The write block size must equal READ_BL_LEN, which is typically 512 bytes. A write is initiated by issuing the WRITE_BLOCK (24) command, to which the SD card responds with the R1 command

response format. If the command response indicates that the write can proceed, the host transmits the data start token followed by a fixed number of data bytes, and ends with a CRC-16 checksum of the sent data. The SD card returns a data response token that indicates the acceptance or rejection of the data to be written.

If the data is accepted, the SD card holds the DO line low continuously while the card is busy. The host is not obligated to keep the card select low during the busy period, and the SD card releases the DO line if the active-low CS is deasserted. This process is useful when more than one device is connected to the SPI bus. The host can wait for the SD card to release the busy indication, or check the card by periodically asserting the chip select. If the card is still busy, it will pull the DO line low to indicate this state. Otherwise, the card returns the DO line to the idle state (see **Figure 6**).

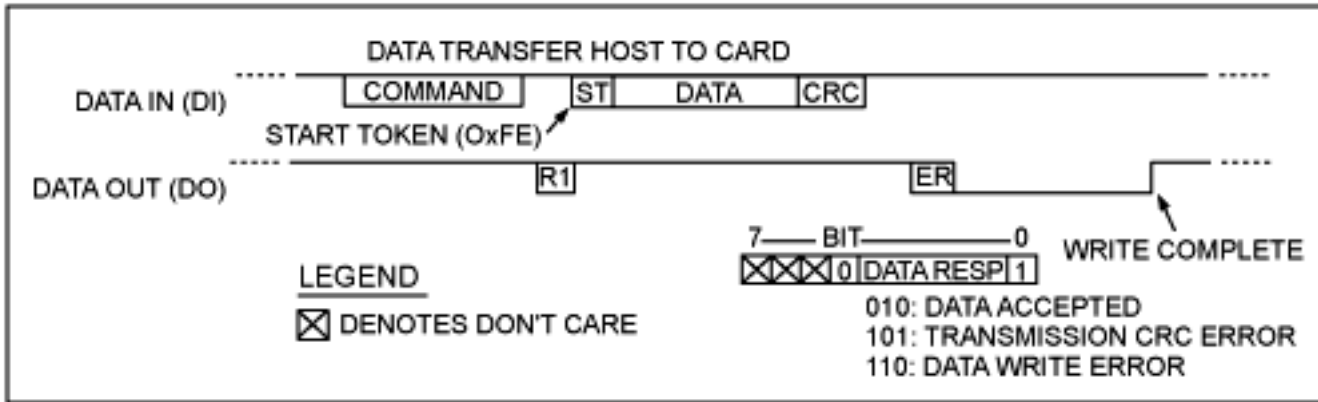


Figure 6. Data transfers from the host to the SD card involve a more complex handshake.

SPI Command and Data Error Detection

The CRC-7 and CRC-16 checksums can be used to detect errors in the communication between the host and SD card. Error detection allows for robust error recovery in the event of physically induced errors such as contact bounce during insertion and removal or nonideal contact-mating situations inherent with detachable media. Therefore, the use of checksums, by issuing the CRC_ON_OFF (59) command with the lowest bit set in the argument, is highly recommended.

Conclusion

The SD media card format represents a compact, low-power, nonvolatile memory solution for embedded systems. By using the SPI support provided by the Maxim DS80C400 microcontroller, SD media cards can be accessed with very little overhead. The [reference software](#) (ZIP) is provided by Maxim. The code requires minimal implementation, which includes the essential operations required to read blocks from and write blocks to an SD card.

References

Code based on the material discussed in the application note can be obtained from Maxim at: http://files.dalsemi.com/microcontroller/app_note_software/an4068_sw.zip.

Further information on the SD media format can be obtained from the [Secure Digital Association](#).

The SD media breakout board used in this project can be ordered from [Spark Fun Electronics](#).

TINI is a registered trademark of Dallas Semiconductor Corp.

Java is a trademark of Sun Microsystems, Inc.

SPI is a trademark of Motorola, Inc.

Dallas Semiconductor is a wholly owned subsidiary of Maxim Integrated Products, Inc.

More Information

For technical questions and support: <http://www.maxim-ic.com/support>

For samples: <http://www.maxim-ic.com/samples>

Other questions and comments: <http://www.maxim-ic.com/contact>

Related Parts

DS80C400: [QuickView](#) -- [Full \(PDF\) Data Sheet](#) -- [Free Samples](#)

DS80C400-KIT: [QuickView](#) -- [Full \(PDF\) Data Sheet](#)

DSTINIM400: [QuickView](#) -- [Full \(PDF\) Data Sheet](#)

DSTINIS400: [QuickView](#) -- [Full \(PDF\) Data Sheet](#)

AN4068, AN 4068, APP4068, Appnote4068, Appnote 4068

Copyright © by Maxim Integrated Products

Additional legal notices: <http://www.maxim-ic.com/legal>