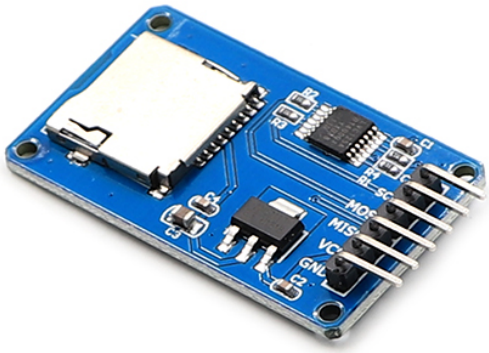
ARDUINO MODULE MICRO SD CARD

User Manual



Overview:

The communication between the microcontroller and the SD card uses SPI, which takes place on digital pins 11, 12, and 13 (on most Arduino boards) or 50, 51, and 52 (Arduino Mega). Additionally, another pin must be used to select the SD card. This can be the hardware SS pin - pin 10 (on most Arduino boards) or pin 53 (on the Mega) - or another pin specified in the call to SD.begin(). Note that even if you don't use the hardware SS pin, it must be left as an output or the SD library won't work. Different boards use different pins for this functionality, so be sure you've selected the correct pin in SD.begin(). Not all the functions are listed on the main SD library page, because they are part of the library's utility functions.

Formatting/Preparing the card:

(NB : whenever referring to the SD card, it means SD and microSD sizes, as well as SD and SDHD formats)
Most SD cards work right out of the box, but it's possible you have one that was used in a computer or camera and it cannot be read by the SD library. Formatting the card will create a file system that the Arduino can read and write to.
It's not desirable to format SD cards frequently, as it shortens their life span.
You'll need a SD reader and computer to format your card. The library supports the FAT16 and FAT32 filesystems, but useFAT16 when possible. The process to format is fairly straightforward.
Windows : right click on your card's directory and choose "Format" from the drop down. Make sure you choose FAT as the filesystem.
OSX : Open Disk Utility (located in Applications>Utilities). Choose the Card, click on the erase tab, select MS-DOS(FAT) as the Format, and click Erase. *NB: OSX places a number of "hidden" files on the device when it formats a drive. To format a SD car without the extra files on OSX, follow these notes on Ladyada's site.*
Linux: With a SD card inserted, open a terminal window. At the prompt, type df, and press enter. The windows will report the device name of your SD card, it should look something like /dev/sdb1. Unmount the SD card, but leave it in the computer. Type sudo mkdosfs -F 16 /dev/sdb1, replacing the device name with yours. Remove the SD card and replace it to verify it works.

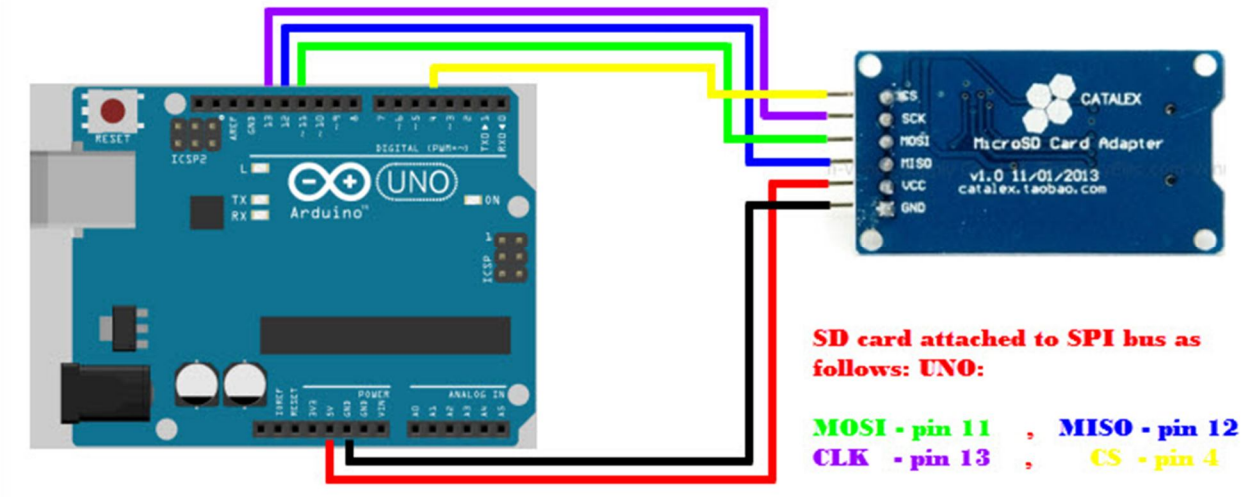
File Naming:

FAT file systems have a limitation when it comes to naming conventions. You must use the 8.3 format, so that file names look like “NAME001.EXT”, where “NAME001” is an 8 character or fewer string, and “EXT” is a 3 character extension. People commonly use the extensions .TXT and .LOG. It is possible to have a shorter file name (for example, mydata.txt, or time.log), but you cannot use longer file names.

Opening/Closing files:

When you use file.write(), it doesn't write to the card until you flush() or close(). Whenever you open a file, be sure to close it to save your data. As of version 1.0, it is possible to have multiple files open.

Pin Configuration:



Example Code: Datalogger:

```
/*
 SD card datalogger
 This example shows how to log data from three analog sensors
 to an SD card using the SD library.

 The circuit:
 * SD card attached to SPI bus as follows:
 ** UNO:  MOSI - pin 11, MISO - pin 12, CLK - pin 13, CS - pin 4 (CS pin can be changed)
  and pin #10 (SS) must be an output
 ** Mega:  MOSI - pin 51, MISO - pin 50, CLK - pin 52, CS - pin 4 (CS pin can be changed)
  and pin #52 (SS) must be an output
 ** Leonardo: Connect to hardware SPI via the ICSP header
                 Pin 4 used here for consistency with other Arduino examples

 created  24 Nov 2010
 modified 9 Apr 2012 by Tom Igoe

 This example code is in the public domain.

 */

#include <SPI.h>
#include <SD.h>
// On the Ethernet Shield, CS is pin 4. Note that even if it's not
// used as the CS pin, the hardware CS pin (10 on most Arduino boards,
// 53 on the Mega) must be left as an output or the SD library
// functions will not work.
const int chipSelect = 10;
```

```
File dataFile;

void setup()
{
 // Open serial communications and wait for port to open:
  Serial.begin(9600);
   while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo only
  }


  Serial.print("Initializing SD card...");
  // make sure that the default chip select pin is set to
  // output, even if you don't use it:
  pinMode(SS, OUTPUT);

  // see if the card is present and can be initialized:
  if (!SD.begin(chipSelect)) {
    Serial.println("Card failed, or not present");
    // don't do anything more:
    while (1) ;
  }
  Serial.println("card initialized.");

  // Open up the file we're going to log to!
  dataFile = SD.open("datalog.txt", FILE_WRITE);
  if (! dataFile) {
    Serial.println("error opening datalog.txt");
    // Wait forever since we cant write data
    while (1) ;
  }
}

void loop()
{
  // make a string for assembling the data to log:
  String dataString = "Test";
  dataFile.println(dataString);

  // print to the serial port too:
  Serial.println(dataString);

  // The following line will 'save' the file to the SD card after every
  // line of data - this will use more power and slow down how much data
  // you can read but it's safer!
  // If you want to speed up the system, remove the call to flush() and it
  // will save the file only every 512 bytes - every time a sector on the
 // SD card is filled with data.
 dataFile.flush();

 // Take 1 measurement every 500 milliseconds
 delay(500);
}
```