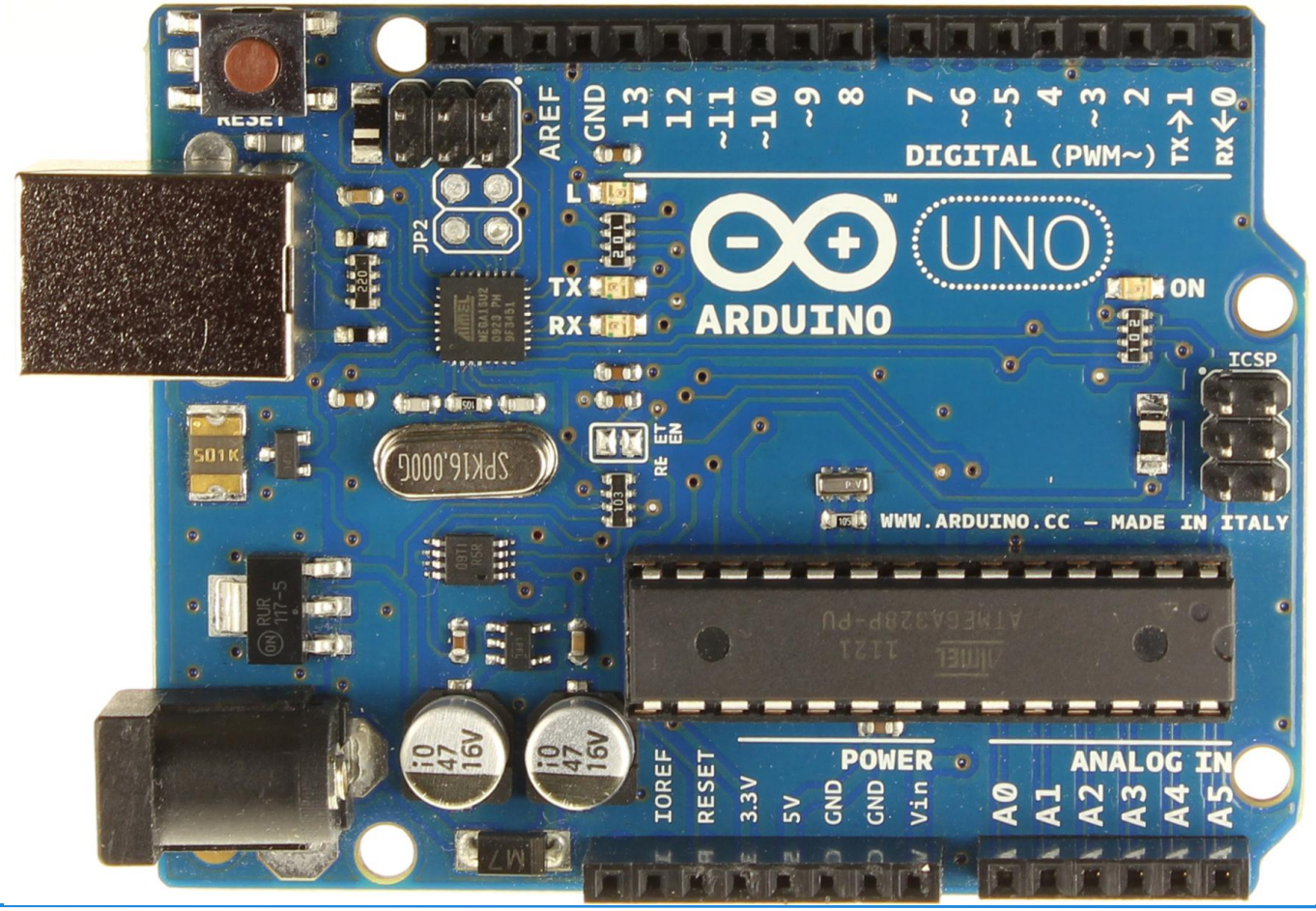
Curso avanzado sobre Arduino: Almacenamiento

ElCacharreo.com

Arduino avanzado: Presente



Arduino avanzado: Presente

José Antonio Vacas Martínez

blog

javacasm@elcacharreo.com

twitter
linkedin

Arduino avanzado: Memoria

¿Cuántos tipos de memoria tiene Arduino y porqué?

Arduino Board	Family	MEMORY		
		SRAM	FLASH	EEPROM
Duemilanove (328)	ATmega328	2K	32k	1kB
Uno	ATmega328	2k	32k	1kB
Arduino Mega 2560	ATmega2560	8k	256k	1kB
Arduino Mega ADK	ATmega2560	8k	256k	1kB
Arduino Ethernet	ATmega328	2k	32k	1kB
Arduino BT	ATmega328	2k	32k	1kB
Arduino Pro Mini 328 5V	ATmega328	2k	32k	1kB
Arduino Nano 3.0	ATmega328	2k	32k	1kB
Arduino Mini	ATmega328	2k	32k	1kB
Arduino Pro 3.3V	ATmega328P	2k	32k	1kB
Arduino Pro 5V	ATmega328P	2k	32k	1kB
Arduino Fio	ATmega328P	2k	32k	1kB
LilyPad Simple Board	ATmega168P	1k	16k	512B
LilyPad 328 Main Board	ATmega328P	2k	32k	1kB

Arduino avanzado: EEPROM

¿Por qué necesitamos almacenar en EEPROM?

¿Qué guardamos?

¿Por qué no guardarlo en el código?

Librería: EEPROM

Guarda datos en la memoria no volátil

- EEPROM.write(address, value) escribe el valor value en address
- EEPROM.read(address): devuelve el valor de la posición address

Ejemplo: Lectura de EEPROM

```
#include <EEPROM.h>
// empieza en el primer byte (dirección 0) de la EEPROM
int direccion = 0;
byte valor;
void setup()
{Serial.begin(9600);}

void loop()
{// Lee un byte de cada direccion de la EEPROM
valor = EEPROM.read(direccion);
Serial.print(direccion);
Serial.print("\t");
Serial.print(valor, DEC);
Serial.println();
// incrementa "direccion" en 1 para avanzar a la siguiente dirección de la EEPROM
direccion = direccion + 1;
// Solamente hay 512 bytes de EEPROM, desde 0 hasta 511
// Si estamos en la direccion 512, volvemos a la dirección 0
if (direccion == 512)
direccion = 0;
delay(500);
}
```

Ejemplo: Escritura de EEPROM

```
#include <EEPROM.h>
// la direccion actual en la EEPROM
// (p.e. cual es el siguiente byte a escribir)
int direccion = 0;
void setup()
{ }

void loop()
{
// Necesita dividir la lectura entre 4 porque el valor analogico estará entre 0 t 1023
// y cada byte de la EEPROM puede contener valores entre 0 y 255
int valor = analogRead(0) / 4;
// Escribe cada valor en la posición apropiada de la EEPROM.
// Estos valores quedarán en la EEPROM cuando la placa se apague.
EEPROM.write(direccion, valor);
// incrementa "direccion" en 1 para avanzar a la siguiente dirección de la EEPROM
direccion = direccion + 1;
// Solamente hay 512 bytes de EEPROM, desde 0 hasta 511
// Si estamos en la direccion 512, volvemos a la dirección 0
if (direccion == 512)
direccion = 0;
delay(100);
}
```

Ejemplo: Otros tipos

Igual que usamos las funciones de EEPROM, existen otro grupo destinado a leer valores más complejos. Para usarlas, hay que saber algo de aritmética de punteros, algo que hemos evitado en este curso. Pongo un ejemplo:

```
#include <EEPROM.h>

void setup()
{
}

void loop()
{
uint8_t valorByte=eeprom_read_byte ((uint8_t *)10) ;
uint16_t valor2Bytes=eeprom_read_word((uint16_t *)10);
uint32_t valor4Bytes=eeprom_read_dword((uint32_t *)10);
}
```

Más sobre EEPROM

http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=38417
Ejemplos: http://playground.arduino.cc/Code/EepromUtil
Guardando estructuras: http://playground.arduino.cc/Code/EEPROMWriteAnything
EEPROMEx: http://playground.arduino.cc/Code/EEPROMex

For reading:

```
uint8_t read(int address);
bool readBit(int address, byte bit)
uint8_t readByte(int address);
uint16_t readInt(int address);
uint32_t readLong(int address);
float readFloat(int address);
double readDouble(int address);
```

For writing:

```
bool write(int address, uint8_t value);
bool writeByte(int address, uint8_t value);
bool writeInt(int address, uint16_t value);
bool writeLong(int address, uint32_t value
bool writeFloat(int address, float value);
bool writeDouble(int address, double value
```

Almacenando datos en la Flash

Podemos alamacenar datos en la memoria Flash PROGMEN tipo variable .

http://arduino.cc/en/Reference/PROGMEM

```
dataType variableName[] PROGMEM = {dataInt0, dataInt1, dataInt3...};

prog_uchar signMessage[] PROGMEM  = {"I AM PREDATOR,  UNSEEN COMBATANT. CREATED BY THE UNITED STATES DEPART"};
```

Almacenamiento: SD

The library supports FAT16 and FAT32 file systems on standard SD cards and SDHC cards. It uses short 8.3 names for files

Comunica con SPI

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.**

http://www.arduino.cc/en/Reference/SD

Almacenamiento: SD

SD class

The SD class provides functions for accessing the SD card and manipulating its files and directories.

- begin()
- exists()
- mkdir()
- open()
- remove()
- rmdir()

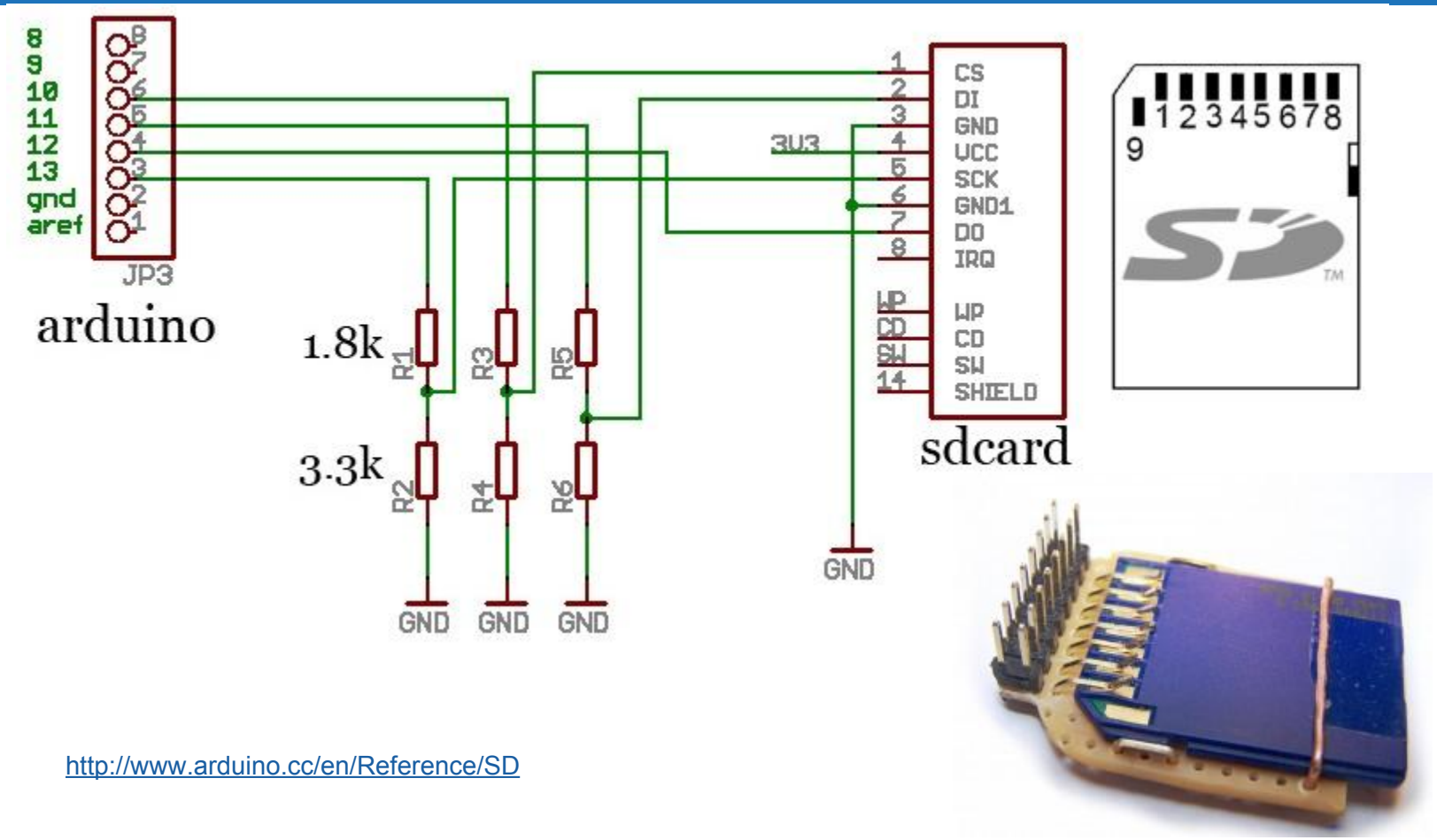
File class

The File class allows for reading from and writing to individual files on the SD card.

- available()
- close()
- flush()
- peek()
- position()
- print()
- println()
- seek()
- size()
- read()
- write()
- isDirectory()
- openNextFile()
- rewindDirectory()

http://www.arduino.cc/en/Reference/SD

Almacenamiento: SD



http://www.arduino.cc/en/Reference/SD

Almacenamiento: SD

```
#include <SD.h>
if (!SD.begin(4))
{
Serial.println("Error inicializando SD");
nFiles=-1;
}
else
{
nFiles=0;
Serial.println("SD initializada.");
}

File file;
if(nFiles>=0 && file)
{
file= SD.open("datalog.txt", FILE_WRITE);
}
if(file)
{
file.print(data);
Serial.print(data);
file.close();
nFilas++;
}
```

Almacenamiento: SD lectura

```
#include <SD.h>
const int chipSelect = 4;
void setup(){
  Serial.begin(9600);
  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(10, OUTPUT);
    // see if the card is present and can be initialized:
  if (!SD.begin(chipSelect)) {
    Serial.println("Card failed, or not present");
    return;  }
    Serial.println("card initialized.");
    File dataFile = SD.open("datalog.txt");  // Sólo un fichero a la vez
  if (dataFile) {
    while (dataFile.available())        Serial.write(dataFile.read());
    dataFile.close();  }
  else     Serial.println("error opening datalog.txt");
}
```

Almacenamiento: SD escritura

```
#include <SD.h>
const int chipSelect = 4;
void setup()  { // Como antes ....}
void loop(){
  String dataString = "";
  for (int analogPin = 0; analogPin < 3; analogPin++) {
    int sensor = analogRead(analogPin);
    dataString += String(sensor);
    if (analogPin < 2)       dataString += ",";   }
  File dataFile = SD.open("datalog.txt", FILE_WRITE);
  if (dataFile) {
    dataFile.println(dataString);
    dataFile.close();
    Serial.println(dataString);  }
  else     Serial.println("error opening datalog.txt");

}
```

Almacenamiento: SD gestión

```
if (!volume.init(card)) {
  Serial.println("Could not find FAT16/FAT32 partition.\nMake sure you've formatted the card");    return;   }
uint32_t volumesize;
Serial.print("\n#Volume type is FAT");   Serial.println(volume.fatType(), DEC);
volumesize = volume.blocksPerCluster();    // clusters are collections of blocks
volumesize *= volume.clusterCount();       // we'll have a lot of clusters
volumesize *= 512;                         // SD card blocks are always 512 bytes
Serial.print("#Volume size (bytes): ");   Serial.println(volumesize);   Serial.print("#Volume size (Kbytes): ");
volumesize /= 1024;
Serial.println(volumesize);   Serial.print("#Volume size (Mbytes): ");
volumesize /= 1024;
Serial.println(volumesize);   Serial.println("\n#Files found on the card (name, date and size in bytes): ");
root.openRoot(volume);
root.ls(LS_R | LS_DATE | LS_SIZE); }
```

Almacenamiento: SD gestión

```
if (!volume.init(card)) {
  Serial.println("Could not find FAT16/FAT32 partition.\nMake sure you've formatted the card");    return;   }
uint32_t volumesize;
Serial.print("\n#Volume type is FAT");   Serial.println(volume.fatType(), DEC);
volumesize = volume.blocksPerCluster();    // clusters are collections of blocks
volumesize *= volume.clusterCount();       // we'll have a lot of clusters
volumesize *= 512;                         // SD card blocks are always 512 bytes
Serial.print("#Volume size (bytes): ");   Serial.println(volumesize);   Serial.print("#Volume size (Kbytes): ");
volumesize /= 1024;
Serial.println(volumesize);   Serial.print("#Volume size (Mbytes): ");
volumesize /= 1024;
Serial.println(volumesize);   Serial.println("\n#Files found on the card (name, date and size in bytes): ");
root.openRoot(volume);
root.ls(LS_R | LS_DATE | LS_SIZE); }
```

Conclusiones

Gracias por vuestra atención