



EBook Gratis

APRENDIZAJE arduino

Free unaffiliated eBook created from
Stack Overflow contributors.

#arduino

Tabla de contenido

Acerca de.....	1
Capítulo 1: Empezando con el arduino.....	2
Observaciones.....	2
¿Qué es Arduino?.....	2
¿Por qué usar Arduino?.....	2
Versiones.....	2
Examples.....	2
Mínimo.....	2
Parpadeo.....	3
Configuración por primera vez.....	4
Preparar.....	5
Subir.....	7
Monitor de serie.....	7
LED - Con control de botón.....	7
Capítulo 2: Almacenamiento de datos.....	9
Examples.....	9
cardInfo.....	9
Registrador de datos de tarjeta SD.....	11
Volcado de archivo de tarjeta SD.....	12
Ejemplo de archivo básico de tarjeta SD.....	13
Listfiles.....	14
Tarjeta SD de lectura / escritura.....	16
Capítulo 3: Arduino IDE.....	18
Examples.....	18
Instalación en Windows.....	18
Aplicación portátil en Windows.....	18
Instalación en Fedora.....	18
Instalando en Ubuntu.....	18
Instalación en macOS.....	18
Capítulo 4: Biblioteca de cristal líquido.....	19

Introducción.....	19
Sintaxis.....	19
Parámetros.....	19
Examples.....	19
Uso básico.....	19
Capítulo 5: Bibliotecas.....	21
Introducción.....	21
Examples.....	21
Instalar bibliotecas con el administrador de bibliotecas.....	21
Incluyendo bibliotecas en tu croquis.....	22
Capítulo 6: Bucles.....	24
Sintaxis.....	24
Observaciones.....	24
Examples.....	24
Mientras.....	24
por.....	25
Hacer ... mientras.....	25
Control de flujo.....	26
Capítulo 7: Cómo almacenar variables en EEPROM y usarlas para almacenamiento permanente	27
Sintaxis.....	27
Parámetros.....	27
Observaciones.....	27
Examples.....	27
Almacene una variable en EEPROM y luego recupérela e imprima en la pantalla.....	27
Capítulo 8: Cómo Python se integra con Arduino Uno.....	29
Sintaxis.....	29
Parámetros.....	29
Observaciones.....	29
Examples.....	29
Primera comunicación serial entre Arduino y Python.....	29
Capítulo 9: Comunicación bluetooth.....	31

Parámetros.....	31
Observaciones.....	32
Examples.....	32
Bluetooth básico hola mundo.....	32
Capítulo 10: Comunicación I2C.....	33
Introducción.....	33
Examples.....	33
Esclavos múltiples.....	33
Capítulo 11: Comunicación MIDI.....	36
Introducción.....	36
Examples.....	36
Ejemplo de MIDI THRU.....	36
MIDI a través de la cola.....	36
Generación de reloj MIDI.....	38
Mensajes MIDI definidos.....	39
Capítulo 12: Comunicación serial.....	44
Sintaxis.....	44
Parámetros.....	44
Observaciones.....	44
Examples.....	45
Simple leer y escribir.....	45
Filtrado Base64 para datos de entrada serie.....	45
Manejo de comandos sobre serie.....	45
Comunicación serial con Python.....	46
Arduino:.....	46
Pitón:.....	47
Capítulo 13: Comunicación SPI.....	48
Observaciones.....	48
Señales de selección de chip.....	48
Actas.....	48
Usando el SPI en las rutinas de servicio de interrupción.....	49

Examples.....	49
Conceptos básicos: inicialice el SPI y un pin de selección de chip, y realice una transfer.....	49
Capítulo 14: Entradas analógicas.....	51
Sintaxis.....	51
Observaciones.....	51
Examples.....	51
Imprima un valor analógico.....	51
Obtener voltaje de pin analógico.....	51
Capítulo 15: Entradas digitales.....	53
Sintaxis.....	53
Parámetros.....	53
Observaciones.....	53
Examples.....	53
Pulsador de lectura.....	53
Capítulo 16: Funciones.....	55
Observaciones.....	55
Examples.....	55
Crear una función simple.....	55
Llamar a una función.....	55
Capítulo 17: Gestión del tiempo.....	57
Sintaxis.....	57
Observaciones.....	57
Bloqueo contra código no bloqueante.....	57
Detalles de implementacion.....	57
Examples.....	58
bloqueando blinky con retraso ().....	58
Blinky sin bloqueo con la librería elapsedMillis (y clase).....	58
Blinky sin bloqueo con millis ().....	59
Mida cuánto tiempo tomó algo, utilizando elapsedMillis y elapsedMicros.....	60
Más de 1 tarea sin demora ().....	60
Capítulo 18: Interrupciones.....	62

Sintaxis.....	62
Parámetros.....	62
Observaciones.....	62
Examples.....	62
Interrupción al presionar un botón.....	62
Capítulo 19: Números al azar.....	64
Sintaxis.....	64
Parámetros.....	64
Observaciones.....	64
Examples.....	64
Generar un número aleatorio.....	64
Poniendo una semilla.....	65
Capítulo 20: Pines de hardware.....	66
Examples.....	66
Arduino Uno R3.....	66
Capítulo 21: PWM - Modulación de ancho de pulso.....	69
Examples.....	69
Controlar un motor de CC a través del puerto serie utilizando PWM.....	69
Los basicos.....	69
Lista de materiales: ¿qué necesitas para construir este ejemplo?.....	70
La construcción.....	70
El código.....	70
PWM con un TLC5940.....	71
Capítulo 22: Salida de audio.....	73
Parámetros.....	73
Examples.....	73
Salidas de notas básicas.....	73
Capítulo 23: Salida digital.....	74
Sintaxis.....	74
Examples.....	74
Escribir a pin.....	74

Capítulo 24: Servo	75
Introducción.....	75
Sintaxis.....	75
Examples.....	75
Mover el servo de ida y vuelta.....	75
Capítulo 25: Usando Arduino con Atmel Studio 7	76
Observaciones.....	76
Preparar	76
Conexiones	76
Consideraciones de depuración	78
Configuración de software	80
Incluir bibliotecas en tu croquis	81
Para agregar la ventana de terminal	81
Beneficios	81
Examples.....	82
Ejemplo de boceto importado de Atmel Studio 7.....	82
Capítulo 26: Variables y tipos de datos	83
Examples.....	83
Crear variable.....	83
Asignar valor a una variable.....	83
Tipos de variables.....	83
Creditos	85

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [arduino](#)

It is an unofficial and free arduino ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official arduino.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con el arduino

Observaciones

¿Qué es Arduino?

Arduino es una plataforma electrónica de código abierto basada en hardware y software fáciles de usar.

¿Por qué usar Arduino?

- Barato. También puedes comprar clones que son incluso más baratos.
- Fácil de usar y comenzar con
- Comunidad enorme
- Completamente Open Source

Versiones

Versión	Fecha de lanzamiento
1.0.0	2016-05-08

Examples

Mínimo

Aquí está el boceto de Arduino 'mínimo'. Esto se puede cargar en el IDE de Arduino seleccionando `File > Examples > 01. Basics > Bare Minimum`.

```
void setup() {
  // put your setup code here, to run once
}

void loop() {
  // put your main code here, to run repeatedly
}
```

El código en la función de `setup()` se ejecutará una vez cuando se inicie el programa. Esto es útil para configurar pines de E / S, inicializar variables, etc. El código en la función `loop()` se ejecutará repetidamente hasta que Arduino esté apagado o se cargue un nuevo programa. Efectivamente, el código anterior se ve así en la biblioteca de tiempo de ejecución de Arduino:

```
setup();
while(1) {
  loop();
}
```

A diferencia de los programas que se ejecutan en su computadora, el código Arduino nunca puede cerrarse. Esto se debe a que el microcontrolador solo tiene un programa cargado. Si este programa se cierra, no hay nada que le diga al microcontrolador qué hacer.

Parpadeo

Aquí hay un breve ejemplo que muestra las funciones `setup()` y `loop()`. Esto se puede cargar en el IDE de Arduino seleccionando `File > Examples > 01. Basics > Blink`. (*Nota: la mayoría de las placas Arduino ya tienen un LED conectado al pin 13, pero es posible que necesite agregar un LED externo para ver los efectos de este boceto*).

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);           // wait for a second
  digitalWrite(13, LOW); // turn the LED off by making the voltage LOW
  delay(1000);           // wait for a second
}
```

El fragmento anterior:

1. Define la función `setup()`. La función `setup()` se llama primero al ejecutarse en cada programa Arduino.

1. Establece el pin 13 como salida.

Sin esto, podría configurarse en una entrada, lo que haría que el LED no funcionara; sin embargo, una vez que se establece como salida, seguirá siendo así, por lo que solo debe hacerse una vez que se inicie el programa.

2. Define la función `loop()`. La función `loop()` se llama repetidamente mientras el programa se está ejecutando.

1. `digitalWrite(13, HIGH)`; enciende el LED.
 2. `delay(1000)`; Espera un segundo (1000 milisegundos).
 3. `digitalWrite(13, LOW)`; apaga el LED.
 4. `delay(1000)`; Espera un segundo (1000 milisegundos).

Debido a que `loop()` se ejecuta repetidamente durante el tiempo que se ejecuta el programa, el LED se encenderá y apagará con un período de 2 segundos (1 segundo encendido, 1 segundo

apagado). Este ejemplo se basa en el Arduino Uno y cualquier otra placa que ya tenga un LED conectado al Pin 13. Si la placa que se está utilizando no tiene un LED integrado en ese pin, se puede conectar una externa.

Más información sobre el tiempo (por ejemplo, retrasos y tiempo de medición): [Gestión del tiempo](#)

Configuración por primera vez

Software necesario: [Arduino IDE](#)



smartbox

DataPacket.cpp

DataPacket.h

EnCoPacket.cpp

EnCoPacket.h

InstrumentationPacket.cpp

Instrumen

```
1 #include "keys.h"
2 #include "device.h"
3 #include "LowPower.h"
4 #include "instrumentationParamEnum.h"
5 #include "sensor.h"
6 // #include "Sensor.h"
7
8 #include <SoftwareSerial.h>
9 #include <avr/wdt.h>
10 #include <avr/sleep.h>
11
12 // Console
13 #define SERIAL_BAUD 9600
14 #define debugSerial Serial
15
16 // Button to send msg on which pin ?? 2 OR 3
17 // Pin change interrupt possible on other pins if needed ....
18 #define BTN_SEND_PIN 2
19 // PIN 2 => IRQ0 // 3 => IRQ1
20 #define IRQ 0
21
22 #define PIN_TX_RN2483 8
23 #define PIN_RX_RN2483 9
24
25 // Arduino's
26 #if defined (__AVR_ATmega328P__)
27     // Serial setup to connect Modem
28     #define PIN_PWR_RN2483 12
```

"DataPacket.h" contains unrecognized characters. If this code was created with an older version of the IDE, you may need to update the file.

"LoRaModem.h" contains unrecognized characters. If this code was created with an older version of the IDE, you may need to update the file.

empty setup() y loop() . Esto es suficiente para subir a un tablero Arduino, pero no hará nada en absoluto. El boceto de ejemplo "Blink" funciona como una prueba simple cuando se usa por primera vez una placa Arduino. Vaya a Archivo → Ejemplos → 01. Básicas → Parpadeo. Esto abrirá una nueva ventana con el boceto de Blink.

Seleccione su tablero. Vaya a Herramientas → Tablero → [nombre de su tablero Arduino].

The screenshot shows the Arduino IDE interface. The 'Tools' menu is open, and the 'Board' option is selected. The 'Board Manager' window is open, displaying a list of available boards. The 'SODAQ Mbili 1284p 8MHz using Optiboot at 57600 baud' board is selected in the Board Manager.

The code in the background is as follows:

```

IAL_BAUD 9600
ugSerial Serial

o send msg on which pin ?? 2 OR 3
ge interrupt possible on other pins if needed ....
_SEND_PIN 2
_IRQ0 // 3 => IRQ1
0

_TX_RN2483 8
_RX_RN2483 9

s
(__AVR_ATmega328P__)
al setup to connect Modem
PIN_PWR_RN2483 12

PIN_Q_MT PD3
PIN_Q_FULL PD5
PIN_Q_IN_BETWEEN PD4
MODEM_SERIAL modemSerial
ed (__AVR_ATmega1284P__)
Q Mbili
PIN_TX_RN2483 3
PIN_RX_RN2483 NULL
PIN_PWR_RN2483 23
    
```

Seleccione el puerto COM para su tablero. La mayoría de las tarjetas compatibles con Arduino

crearán un puerto COM falso, que se utiliza para la comunicación en serie (depuración) y para la programación de la tarjeta. COM 1 es *por lo general* ya presente, y su junta directiva creará uno nuevo, por ejemplo COM 4. Seleccione esto desde Herramientas → → Puerto COM 4 (u otro número de COM).

Algunas placas tienen configuraciones adicionales en el menú Herramientas, como la velocidad del reloj. Estos varían de una placa a otra, pero generalmente ya se ha seleccionado un conjunto aceptable de valores predeterminados.

Subir

Ya estás listo para subir Blink. Haga clic en el botón Cargar o seleccione Boceto → Cargar. El boceto se compilará, luego, cárguelo en su tablero Arduino. Si todo funcionó, el LED de a bordo comenzará a parpadear cada segundo.



Monitor de serie

En Arduino IDE you have un monitor serie. Para abrirlo use el botón *monitor serie* en el lado derecho de la ventana.



Asegúrese de que el código esté cargado antes de abrir el monitor. ¡La carga y el monitor no se ejecutarán al mismo tiempo!

LED - Con control de botón

También puede usar este código para configurar un LED con un interruptor de botón con una resistencia de extracción, esto podría ser preferiblemente con el siguiente paso después de configurar el controlador de LED inicial

```
int buttonState = 0; // variable for reading the pushbutton status

void setup()
{
  // initialize the LED pin as an output:
  pinMode(13, OUTPUT); // You can set it just using its number
  // initialize the pushbutton pin as an input:
  pinMode(2, INPUT);
}

void loop()
{
  // read the state of the pushbutton value:
  buttonState = DigitalRead(2);
```

```
// check if the pushbutton is pressed.  
// If it's not, the buttonState is HIGH : if (buttonState == HIGH)  
{  
    // turn LED off:  
    digitalWrite(13, LOW);  
}  
else  
{  
    // turn LED off:  
    digitalWrite(13, HIGH);  
}  
}
```

Lea Empezando con el arduino en línea: <https://riptutorial.com/es/arduino/topic/610/empezando-con-el-arduino>

Capítulo 2: Almacenamiento de datos

Examples

cardInfo

```
/*
SD card test

This example shows how use the utility libraries on which the
SD library is based in order to get info about your SD card.
Very useful for testing a card when you're not sure whether its working or not.

The circuit:
 * SD card attached to SPI bus as follows:
 ** MOSI - pin 11 on Arduino Uno/Duemilanove/Diecimila
 ** MISO - pin 12 on Arduino Uno/Duemilanove/Diecimila
 ** CLK - pin 13 on Arduino Uno/Duemilanove/Diecimila
 ** CS - depends on your SD card shield or module.
           Pin 4 used here for consistency with other Arduino examples

created 28 Mar 2011
by Limor Fried
modified 9 Apr 2012
by Tom Igoe
*/

// include the SD library:
#include <SPI.h>
#include <SD.h>

// set up variables using the SD utility library functions:
Sd2Card card;
SdVolume volume;
SdFile root;

// change this to match your SD shield or module;
// Arduino Ethernet shield: pin 4
// Adafruit SD shields and modules: pin 10
// Sparkfun SD shield: pin 8
const int chipSelect = 4;

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("\nInitializing SD card...");

  // we'll use the initialization code from the utility libraries
  // since we're just testing if the card is working!
  if (!card.init(SPI_HALF_SPEED, chipSelect)) {
    Serial.println("initialization failed. Things to check:");
    Serial.println("* is a card inserted?");
  }
}
```



```

    Serial.println("* is your wiring correct?");
    Serial.println("* did you change the chipSelect pin to match your shield or module?");
    return;
} else {
    Serial.println("Wiring is correct and a card is present.");
}

// print the type of card
Serial.print("\nCard type: ");
switch (card.type()) {
    case SD_CARD_TYPE_SD1:
        Serial.println("SD1");
        break;
    case SD_CARD_TYPE_SD2:
        Serial.println("SD2");
        break;
    case SD_CARD_TYPE_SDHC:
        Serial.println("SDHC");
        break;
    default:
        Serial.println("Unknown");
}

// Now we will try to open the 'volume'/'partition' - it should be FAT16 or FAT32
if (!volume.init(card)) {
    Serial.println("Could not find FAT16/FAT32 partition.\nMake sure you've formatted the
card");
    return;
}

// print the type and size of the first FAT-type volume
uint32_t volumesize;
Serial.print("\nVolume type is FAT");
Serial.println(volume.fatType(), DEC);
Serial.println();

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("\nFiles found on the card (name, date and size in bytes): ");
root.openRoot(volume);

// list all files in the card with date and size
root.ls(LS_R | LS_DATE | LS_SIZE);
}

void loop(void) {
}

```

Registrador de datos de tarjeta SD

```
/*
  SD card datalogger

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

  The circuit:
  * analog sensors on analog ins 0, 1, and 2
  * SD card attached to SPI bus as follows:
  ** MOSI - pin 11
  ** MISO - pin 12
  ** CLK - pin 13
  ** CS - pin 4

  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>

const int chipSelect = 4;

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...");

  // 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:
    return;
  }
  Serial.println("card initialized.");
}

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

  // read three sensors and append to the string:
  for (int analogPin = 0; analogPin < 3; analogPin++) {
    int sensor = analogRead(analogPin);
    dataString += String(sensor);
    if (analogPin < 2) {
      dataString += ",";
    }
  }
}
```

```

}

// open the file. note that only one file can be open at a time,
// so you have to close this one before opening another.
File dataFile = SD.open("datalog.txt", FILE_WRITE);

// if the file is available, write to it:
if (dataFile) {
  dataFile.println(dataString);
  dataFile.close();
  // print to the serial port too:
  Serial.println(dataString);
}
// if the file isn't open, pop up an error:
else {
  Serial.println("error opening datalog.txt");
}
}

```

Volcado de archivo de tarjeta SD

```

/*
  SD card file dump

  This example shows how to read a file from the SD card using the
  SD library and send it over the serial port.

  The circuit:
  * SD card attached to SPI bus as follows:
  ** MOSI - pin 11
  ** MISO - pin 12
  ** CLK - pin 13
  ** CS - pin 4

  created 22 December 2010
  by Limor Fried
  modified 9 Apr 2012
  by Tom Igoe

  This example code is in the public domain.

  */

#include <SPI.h>
#include <SD.h>

const int chipSelect = 4;

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...");

```

```

// 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:
  return;
}
Serial.println("card initialized.");

// open the file. note that only one file can be open at a time,
// so you have to close this one before opening another.
File dataFile = SD.open("datalog.txt");

// if the file is available, write to it:
if (dataFile) {
  while (dataFile.available()) {
    Serial.write(dataFile.read());
  }
  dataFile.close();
}
// if the file isn't open, pop up an error:
else {
  Serial.println("error opening datalog.txt");
}
}

void loop()
{
}

```

Ejemplo de archivo básico de tarjeta SD

```

/*
  SD card basic file example

  This example shows how to create and destroy an SD card file
  The circuit:
  * SD card attached to SPI bus as follows:
  ** MOSI - pin 11
  ** MISO - pin 12
  ** CLK - pin 13
  ** CS - pin 4

  created   Nov 2010
  by David A. Mellis
  modified  9 Apr 2012
  by Tom Igoe

  This example code is in the public domain.

  */
#include <SPI.h>
#include <SD.h>

File myFile;

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...");

if (!SD.begin(4)) {
    Serial.println("initialization failed!");
    return;
}
Serial.println("initialization done.");

if (SD.exists("example.txt")) {
    Serial.println("example.txt exists.");
}
else {
    Serial.println("example.txt doesn't exist.");
}

// open a new file and immediately close it:
Serial.println("Creating example.txt...");
myFile = SD.open("example.txt", FILE_WRITE);
myFile.close();

// Check to see if the file exists:
if (SD.exists("example.txt")) {
    Serial.println("example.txt exists.");
}
else {
    Serial.println("example.txt doesn't exist.");
}

// delete the file:
Serial.println("Removing example.txt...");
SD.remove("example.txt");

if (SD.exists("example.txt")) {
    Serial.println("example.txt exists.");
}
else {
    Serial.println("example.txt doesn't exist.");
}
}

void loop()
{
    // nothing happens after setup finishes.
}

```

Listfiles

```

/*
Listfiles

This example shows how print out the files in a
directory on a SD card

The circuit:

```

```

* SD card attached to SPI bus as follows:
** MOSI - pin 11
** MISO - pin 12
** CLK - pin 13
** CS - pin 4

created   Nov 2010
by David A. Mellis
modified  9 Apr 2012
by Tom Igoe
modified  2 Feb 2014
by Scott Fitzgerald

This example code is in the public domain.
*/

#include <SPI.h>
#include <SD.h>

File root;

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...");

  if (!SD.begin(4)) {
    Serial.println("initialization failed!");
    return;
  }
  Serial.println("initialization done.");

  root = SD.open("/");

  printDirectory(root, 0);

  Serial.println("done!");
}

void loop()
{
  // nothing happens after setup finishes.
}

void printDirectory(File dir, int numTabs) {
  while(true) {

    File entry = dir.openNextFile();
    if (! entry) {
      // no more files
      break;
    }
    for (uint8_t i=0; i<numTabs; i++) {
      Serial.print('\t');
    }
    Serial.print(entry.name());
  }
}

```

```

    if (entry.isDirectory()) {
        Serial.println("/");
        printDirectory(entry, numTabs+1);
    } else {
        // files have sizes, directories do not
        Serial.print("\t\t");
        Serial.println(entry.size(), DEC);
    }
    entry.close();
}
}
}

```

Tarjeta SD de lectura / escritura

```

/*
  SD card read/write

  This example shows how to read and write data to and from an SD card file
  The circuit:
  * SD card attached to SPI bus as follows:
  ** MOSI - pin 11
  ** MISO - pin 12
  ** CLK - pin 13
  ** CS - pin 4

  created   Nov 2010
  by David A. Mellis
  modified  9 Apr 2012
  by Tom Igoe

  This example code is in the public domain.

  */

#include <SPI.h>
#include <SD.h>

File myFile;

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...");

  if (!SD.begin(4)) {
    Serial.println("initialization failed!");
    return;
  }
  Serial.println("initialization done.");

  // open the file. note that only one file can be open at a time,
  // so you have to close this one before opening another.
  myFile = SD.open("test.txt", FILE_WRITE);

```

```
// if the file opened okay, write to it:
if (myFile) {
  Serial.print("Writing to test.txt...");
  myFile.println("testing 1, 2, 3.");
  // close the file:
  myFile.close();
  Serial.println("done.");
} else {
  // if the file didn't open, print an error:
  Serial.println("error opening test.txt");
}

// re-open the file for reading:
myFile = SD.open("test.txt");
if (myFile) {
  Serial.println("test.txt:");

  // read from the file until there's nothing else in it:
  while (myFile.available()) {
    Serial.write(myFile.read());
  }
  // close the file:
  myFile.close();
} else {
  // if the file didn't open, print an error:
  Serial.println("error opening test.txt");
}
}

void loop()
{
  // nothing happens after setup
}
```

Lea Almacenamiento de datos en línea:

<https://riptutorial.com/es/arduino/topic/6584/almacenamiento-de-datos>

Capítulo 3: Arduino IDE

Examples

Instalación en Windows

1. Vaya a <https://www.arduino.cc/en/Main/Software>
2. Haga clic en el enlace "Windows Installer"
3. Sigue las instrucciones

Aplicación portátil en Windows

Para usar el IDE de Arduino en Windows sin necesidad de instalarlo:

1. Vaya a <https://www.arduino.cc/en/Main/Software>
2. Haga clic en el enlace "Windows ZIP for no admin install"
3. Extraer el archivo a una carpeta
4. Abra la carpeta, y haga doble clic en `Arduino.exe`

Instalación en Fedora

1. Abre un terminal y ejecuta: `sudo dnf install arduino`
2. Abra la aplicación Arduino, o escriba `arduino` en el terminal

Instalando en Ubuntu

1. Abre un terminal y ejecuta: `sudo apt-get install arduino`
2. Abra la aplicación Arduino, o escriba `arduino` en el terminal

Instalación en macOS

1. Vaya a <https://www.arduino.cc/en/Main/Software>
2. Haga clic en el enlace de `Mac OS X`
3. Descomprima el archivo `.zip`.
4. Mueva la aplicación `Arduino` a `Applications`.

Lea Arduino IDE en línea: <https://riptutorial.com/es/arduino/topic/3790/arduino-ide>

Capítulo 4: Biblioteca de cristal líquido

Introducción

La `LiquidCrystal Library` de Arduino es una biblioteca para controlar pantallas LCD compatibles con el controlador Hitachi HD44780, que se caracteriza por su interfaz de 16 pines. Los 16 pines pueden estar conectados a través de una interfaz I2C. Estas pantallas contienen una matriz de bloques de 5x7 píxeles utilizados para mostrar caracteres o pequeñas imágenes monocromáticas. Las pantallas suelen tener nombres de acuerdo con la cantidad de filas y columnas que tienen, por ejemplo, 16x2 o 1602 para 16 columnas y 2 filas, y 20x4 o 2004 para 20 columnas y 4 filas.

Sintaxis

- `#include <LiquidCrystal.h> // Incluye la biblioteca`
- `LiquidCrystal (rs, enable, d4, d5, d6, d7) //`
- `LiquidCrystal (rs, rw, enable, d4, d5, d6, d7)`
- `LiquidCrystal (rs, enable, d0, d1, d2, d3, d4, d5, d6, d7)`
- `LiquidCrystal (rs, rw, enable, d0, d1, d2, d3, d4, d5, d6, d7)`

Parámetros

Parámetro LiquidCrystal	Detalles
rs	el número del pin Arduino que está conectado al pin RS en la pantalla LCD
rw	el número del pin Arduino que está conectado al pin RW en la pantalla LCD (opcional)
habilitar	el número del pin Arduino que está conectado al pin habilitado en la pantalla LCD
d0 - d7	Los números de los pines Arduino que están conectados a los pines de datos correspondientes en la pantalla LCD. d0, d1, d2 y d3 son opcionales; si se omite, la pantalla LCD se controlará utilizando solo las cuatro líneas de datos (d4, d5, d6, d7).

Examples

Uso básico

```
/*
```

```

Wiring:
LCD pin 1 (VSS) -> Arduino Ground
LCD pin 2 (VDD) -> Arduino 5V
LCD pin 3 (VO) -> Arduino Ground
LCD pin 4 (RS) -> Arduino digital pin 12
LCD pin 5 (RW) -> Arduino Ground
LCD pin 6 (E) -> Arduino digital pin 11
LCD pin 11 (D4) -> Arduino digital pin 5
LCD pin 12 (D5) -> Arduino digital pin 4
LCD pin 13 (D6) -> Arduino digital pin 3
LCD pin 14 (D7) -> Arduino digital pin 2
*/

#include <LiquidCrystal.h> // include the library

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // start writing on the first row and first column.
  lcd.setCursor(0, 0);
  // Print a message to the LCD.
  lcd.print("hello, world!");
}

void loop() {
  // No need to do anything to keep the text on the display
}

```

Lea Biblioteca de cristal líquido en línea: <https://riptutorial.com/es/arduino/topic/9395/biblioteca-de-cristal-liquido>

Capítulo 5: Bibliotecas

Introducción

Aquí encontrarás documentación sobre:

-Instalación de bibliotecas en el IDE de Arduino.

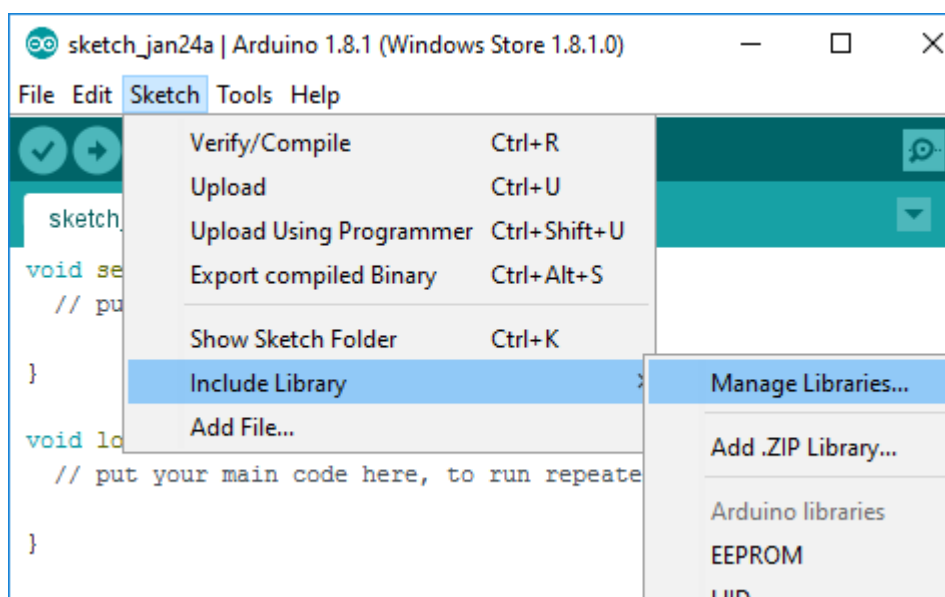
-Incluyendo bibliotecas en un Sketch

Examples

Instalar bibliotecas con el administrador de bibliotecas

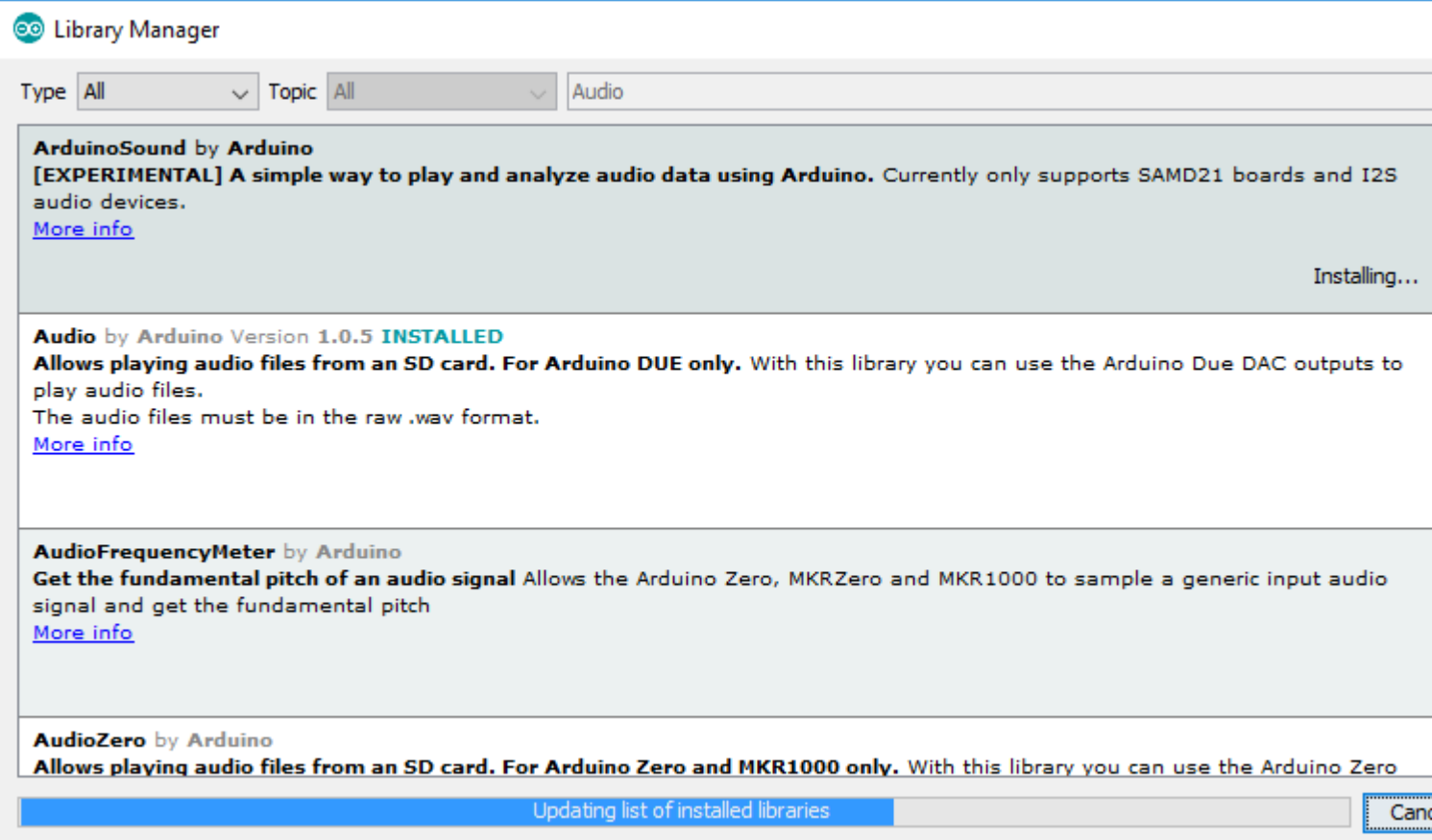
Para instalar una nueva biblioteca en el IDE de Arduino:

- **Abra el menú de croquis> Incluir biblioteca> Administrar bibliotecas.**



Una vez que haya abierto el Administrador de bibliotecas, puede usar el menú en la parte superior para filtrar los resultados.

- **Haga clic en la biblioteca que desee, seleccione una versión en el menú desplegable y haga clic en instalar.**

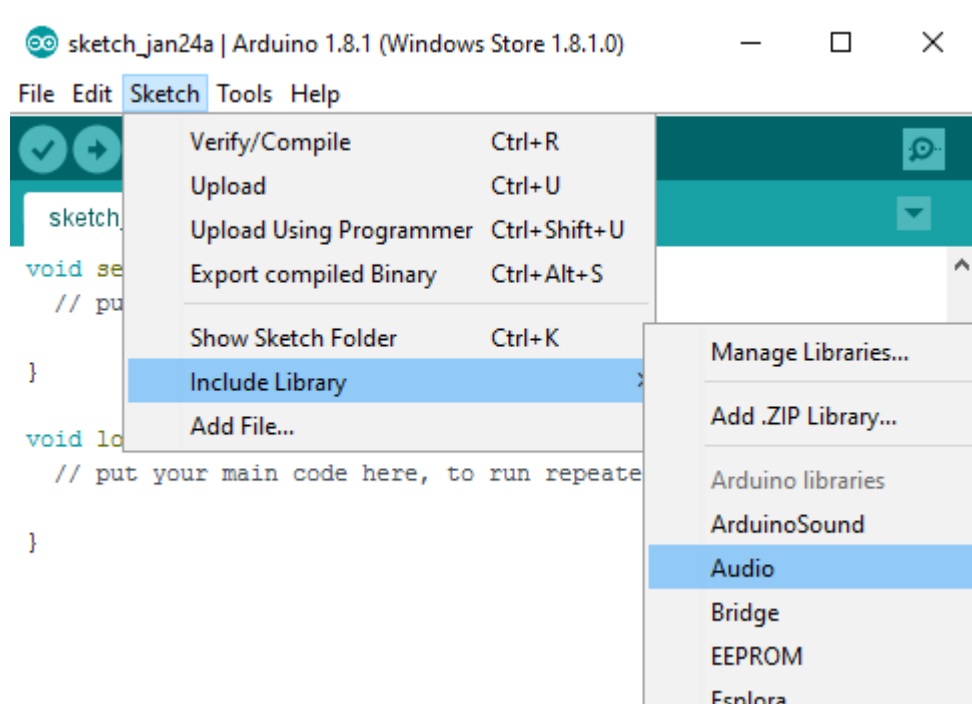


Ahora su biblioteca está instalada. Para usarlo, debes incluirlo en tu boceto.

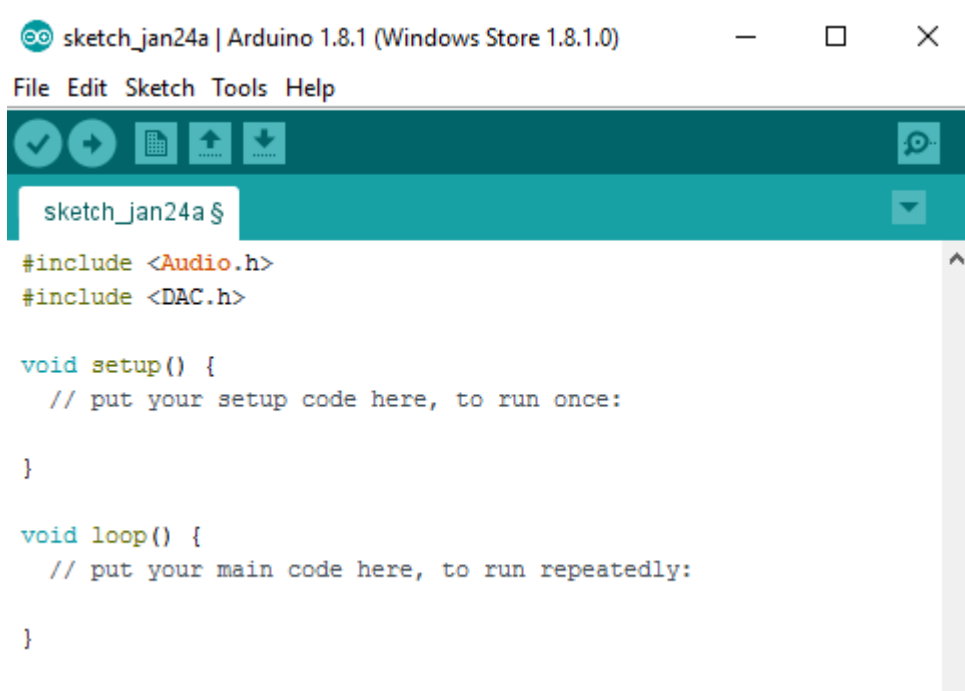
Incluyendo bibliotecas en tu croquis.

Una vez que haya instalado una biblioteca, debe incluirla en su boceto para poder usarla.

- **Abra el menú Boceto > Incluir biblioteca y haga clic en la biblioteca que desea incluir.**



- Ahora, el IDE ha generado las etiquetas de inclusión requeridas en su código.



The screenshot shows the Arduino IDE interface. The title bar reads "sketch_jan24a | Arduino 1.8.1 (Windows Store 1.8.1.0)". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". The toolbar contains icons for saving, running, and other IDE functions. The code editor displays the following code:

```
sketch_jan24a $
#include <Audio.h>
#include <DAC.h>

void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

Ahora la biblioteca está incluida en su boceto, y puede usarla en su código.

Lea Bibliotecas en línea: <https://riptutorial.com/es/arduino/topic/8896/bibliotecas>

Capítulo 6: Bucles

Sintaxis

- para (declaración, condición, iteración) {}
- while (condición) {}
- do {} while (condicion)

Observaciones

General Remark Si pretende crear un bucle para esperar a que suceda algo, es probable que esté en el camino equivocado aquí. Más bien, recuerde que todo el código después de la configuración () se ejecuta desde un método llamado bucle (). Entonces, si necesita esperar por algo, es más fácil no hacer nada (o solo otras cosas independientes) y volver para verificar la condición de espera la próxima vez.

`do { } while(condition)` no evaluará la declaración de condición hasta después de la primera iteración. Es importante tener esto en cuenta si la declaración de condición tiene efectos secundarios.

Examples

Mientras

Un `while` de bucle evaluará su condición, y si `true`, se ejecutará el código dentro y empezará de nuevo. Es decir, siempre y cuando su condición se evalúa como `true`, el `while` de bucle se ejecutará una y otra vez.

Este bucle se ejecutará 100 veces, cada vez que se agregue 1 a la variable `num`:

```
int num = 0;
while (num < 100) {
    // do something
    num++;
}
```

El bucle anterior es equivalente a un bucle `for`:

```
for (int i = 0; i < 100; i++) {
    // do something
}
```

Este bucle se ejecutará para siempre:

```
while (true) {
    // do something
}
```

El bucle anterior es equivalente a un bucle `for` :

```
for (;;) {
    // do something
}
```

por

`for` bucles son sintaxis simplificada para un patrón de bucle muy común, lo que podría llevarse a cabo en más líneas con un `while` de bucle.

El siguiente es un ejemplo común de un bucle `for` , que se ejecutará 100 veces y luego se detendrá.

```
for (int i = 0; i < 100; i++) {
    // do something
}
```

Esto es equivalente a un `while` de bucle:

```
int num = 0;
while (num < 100) {
    // do something
    num++;
}
```

Puede crear un bucle sin fin omitiendo la condición.

```
for (;;) {
    // do something
}
```

Esto es equivalente a un `while` de bucle:

```
while (true) {
    // do something
}
```

Hacer ... mientras

Un `do while` bucle es lo mismo que un `while` de bucle, excepto que está garantizado para ejecutar al menos una vez.

El siguiente bucle se ejecutará 100 veces.

```
int i = 0;
```



```
do {
    i++;
} while (i < 100);
```

Un bucle similar, pero con una condición diferente, se ejecutará 1 vez.

```
int i = 0;
do {
    i++;
} while (i < 0);
```

Si el bucle de arriba eran meramente un `while` bucle, se ejecutaría 0 veces, porque la condición evaluaría a `false` antes de la primera iteración. Pero como es un bucle `do while` while, se ejecuta una vez, luego verifica su condición antes de ejecutarse nuevamente.

Control de flujo

Hay algunas formas de romper o cambiar el flujo de un bucle.

`break;` saldrá del bucle actual y no ejecutará más líneas dentro de ese bucle.

`continue;` no ejecutará ningún código más dentro de la iteración actual del bucle, pero permanecerá en el bucle.

El siguiente bucle se ejecutará 101 veces ($i = 0, 1, \dots, 100$) en lugar de 1000, debido a la instrucción `break` :

```
for (int i = 0; i < 1000; i++) {
    // execute this repeatedly with i = 0, 1, 2, ...
    if (i >= 100) {
        break;
    }
}
```

El siguiente bucle dará como resultado que el valor de `j` sea 50 en lugar de 100, debido a la instrucción `continue` :

```
int j=0;
for (int i = 0; i < 100; i++) {
    if (i % 2 == 0) { // if `i` is even
        continue;
    }
    j++;
}
// j has the value 50 now.
```

Lea Bucles en línea: <https://riptutorial.com/es/arduino/topic/2802/bucles>

Capítulo 7: Cómo almacenar variables en EEPROM y usarlas para almacenamiento permanente

Sintaxis

- `EEPROM.write (dirección, valor); //` (Almacenar variables en EEPROM en una dirección particular)
- `EEPROM.read (dirección); //` (Recuperar valores de EEPROM y leer datos almacenados en EEPROM)

Parámetros

Parámetros de EEPROM.write	Detalle
dirección	La dirección donde se almacena el valor en EEPROM
valor	Variable principal para almacenar en EEPROM. Tenga en cuenta que este es un <code>uint_8</code> (byte único): usted mismo debe dividir los tipos de datos de múltiples bytes en bytes individuales. O puede utilizar <code>EEPROM.put</code> para almacenar datos flotantes u otros tipos de datos.
Parámetros de EEPROM.Leer	Detalle
dirección	La dirección desde la que se va a leer la variable.

Observaciones

Las direcciones permitidas varían según el hardware.

- ATmega328 (Uno, Pro Mini, etc.): 0–1023
- ATmega168: 0-511
- ATmega1280: 0-4095
- ATmega2560: 0-4095

[fuente](#)

Examples

Almacene una variable en EEPROM y luego recupérela e imprima en la

pantalla

Primero, agregue una referencia a `<EEPROM.h>` al comienzo de su boceto:

```
#include <EEPROM.h>
```

Entonces su otro código:

```
// Stores value in a particular address in EEPROM. There are almost 512 addresses present.  
  
// Store value 24 to Address 0 in EEPROM  
int addr = 0;  
int val = 24;  
EEPROM.write(addr, val);    // Writes 24 to address 0  
  
// -----  
// Retrieves value from a particular address in EEPROM  
// Retrieve value from address 0 in EEPROM  
int retrievedVal = EEPROM.read(0);    // Retrieves value stored in 0 address in  
                                       // EEPROM  
  
// *[NOTE: put Serial.begin(9600); at void setup()]*  
Serial.println(retrievedVal);    // Prints value stored in EEPROM Address 0 to  
                                   // Serial (screen)
```

Lea [Cómo almacenar variables en EEPROM y usarlas para almacenamiento permanente en línea](https://riptutorial.com/es/arduino/topic/5987/como-almacenar-variables-en-eprom-y-usarlas-para-almacenamiento-permanente): <https://riptutorial.com/es/arduino/topic/5987/como-almacenar-variables-en-eprom-y-usarlas-para-almacenamiento-permanente>

Capítulo 8: Cómo Python se integra con Arduino Uno

Sintaxis

- `Serial.begin(baudrate)` // Set baud rate (bits per second) for serial data transmission
- `Serial.println(value)` // Print data to serial port followed by Carriage Return `\r` and Newline character `\n`
- `serial.Serial((port=None, baudrate=9600, bytesize=EIGHTBITS, parity=PARITY_NONE, stopbits=STOPBITS_ONE, timeout=None, xonxoff=False, rtscts=False, write_timeout=None, dsrdtr=False, inter_byte_timeout=None))` // Initialize serial port with all parameters
- `serial.readline()` // Read serial data which contains Carriage Return `\r` and Newline character `\n`

Parámetros

Parámetro	Detalles
de serie	El paquete Python contiene clases y métodos para acceder al puerto serie
hora	El paquete Python incluye funciones relacionadas con el tiempo.

Observaciones

Utilizo un Arduino Uno con Arduino IDE 1.6.9 y Python 2.7.12 que se ejecuta en Windows 10.

Examples

Primera comunicación serial entre Arduino y Python.

En este primer ejemplo, se inicia una operación de escritura en serie básica desde un dispositivo Arduino.

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  Serial.println("Hello World!");
  delay(100);
}
```

En la `setup()`, la función `Serial.begin(9600)` configura la velocidad en baudios para la comunicación de datos en serie. En este ejemplo, se utiliza una velocidad en baudios de 9600.

Otros valores se pueden leer aquí: [Arduino Serial.begin \(\) function](#)

En `loop()` , el primer mensaje que nos gustaría enviar es "Hello World!". Este mensaje se transmite mediante `Serial.println("Hello World!")` , Ya que enviará esta cadena al puerto serie en formato ASCII. Al final del mensaje, hay retorno de carro (`CR`, `\r`) y carácter de nueva línea (`\n`) . Además, se utiliza un retraso de 100 milisegundos cada vez que el programa se imprime en el puerto serie.

A continuación, cargue este boceto de Arduino a través del puerto COM (recuerde este número de puerto COM, ya que se usará en el programa Python).

El programa Python que lee los datos en serie enviados desde el dispositivo Arduino se muestra a continuación:

```
import serial
import time

ser = serial.Serial('COM8', 9600)
while (1):
    print ser.readline()
    time.sleep(0.1)
```

Primero, el paquete `pyserial` debe ser importado. Para obtener más información sobre la instalación de `pyserial` en el entorno de Windows, consulte esta instrucción: [Instalación de Python y pyserial](#) . Luego, inicializamos el puerto serie con el número de puerto COM y la velocidad de transmisión. La velocidad de transmisión debe ser la misma que la utilizada en el croquis de Arduino.

El mensaje recibido se imprimirá en bucle `while` utilizando la función `readline()` . También se usa un retraso de 100 milisegundos aquí como en el croquis de Arduino. Tenga en cuenta que la función `readline()` requiere un tiempo de espera al abrir un puerto serie (documentación de `pyserial` : [PySerial ReadLine](#)).

Lea [Cómo Python se integra con Arduino Uno en línea](#):

<https://riptutorial.com/es/arduino/topic/6722/como-python-se-integra-con-arduino-uno>

Capítulo 9: Comunicación bluetooth

Parámetros

método	detalles
SoftwareSerial.h	Documentación
SoftwareSerial (rxPin, txPin, inverse_logic)	Constructor. rxPin : Pin de entrada (recepción) de datos, por defecto es 0. txPin : Pin de salida de datos (transmisión), por defecto es 1. inverse_logic : Si es verdadero, trata BAJO como si fuera ALTO y ALTO como BAJO al determinar los valores de bit. por defecto es falso.
comenzar (velocidad)	Establece la velocidad en baudios para la comunicación en serie. Las velocidades de transmisión admitidas son 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 31250, 38400, 57600 y 115200.
disponible()	Compruebe si hay algunos datos en serie
leer()	Lee una cadena de serie
esta escuchando()	Comprueba si el puerto serie del software solicitado está escuchando activamente.
rebosar()	Comprueba si se ha producido un desbordamiento de búfer serie de software. Al llamar a esta función, se borra el indicador de desbordamiento, lo que significa que las llamadas posteriores devolverán el valor falso a menos que se haya recibido y descartado otro byte de datos mientras tanto. El búfer serial del software puede contener 64 bytes.
ojeada()	Devuelve un carácter que se recibió en el pin RX del puerto serie del software. Sin embargo, a diferencia de read (), las llamadas posteriores a esta función devolverán el mismo carácter. Tenga en cuenta que solo una instancia de SoftwareSerial puede recibir datos entrantes a la vez (seleccione cuál con la función <code>listen()</code>).
imprimir (datos)	Imprime datos en el pin de transmisión del puerto serie del software. Funciona igual que la función <code>Serial.print()</code> .
println (datos)	Imprime datos en el pin de transmisión del puerto serie del software, seguido de un retorno de carro y avance de línea. Funciona igual que la función <code>Serial.println()</code> .
escucha()	Habilita el puerto serie del software seleccionado para escuchar. Solo un puerto serie de software puede escuchar a la vez; Los datos que

método	detalles
	lleguen a otros puertos serán descartados. Cualquier dato ya recibido se descarta durante la llamada a <code>listen()</code> (a menos que la instancia dada ya esté escuchando).
escribir (datos)	Imprime datos en el pin de transmisión del puerto serie del software como bytes sin procesar. Funciona igual que la función <code>Serial.write()</code> .

Observaciones

Error común: si mantiene los pines rx y tx en los valores predeterminados (0 y 1), no puede cargar código nuevo hasta que lo elimine, a menos que así sea, por lo que casi siempre es mejor cambiar los pines tx y rx en el constructor `SoftwareSerial`.

Examples

Bluetooth básico hola mundo

```
#include <SoftwareSerial.h>
// its always better to change the default tx and rx as the may interfere with other process
// in future.

// configure tx , rx by default they will be 0 and 1 in arduino UNO
SoftwareSerial blue(3,2);
void setup() {
  // preferred baud rate/data transfer rate in general is 38400
  blue.begin(38400);
  // do initialization or put one time executing code here
}

void loop() {

  // put code that you want it to run every time no matter what
  if(blue.available()){
    // put only that code which needsd to run when there is some data
    // This means that the their is some data sent over the bluetooth
    // You can do something with the data

    int n;
    // consider that the data received to be integer, read it by using blue.parseInt();

    n = blue.parseInt();

  }
}
```

Lea Comunicación bluetooth en línea: <https://riptutorial.com/es/arduino/topic/2543/comunicacion-bluetooth>

Capítulo 10: Comunicación I2C

Introducción

I2C es un protocolo de comunicación que puede hacer que dos o más tableros Arduino se comuniquen entre sí. El protocolo utiliza dos pines: SDA (línea de datos) y SCL (línea de reloj). Esos pines son diferentes de un tipo de placa Arduino a otro, así que verifique la especificación de la placa. El protocolo I2C estableció una placa Arduino como maestro y todas las demás como esclavo. Cada esclavo tiene una dirección diferente que el programador configuró de forma rígida. Observación: asegúrese de que todas las tarjetas conectadas a la misma fuente VCC

Examples

Esclavos múltiples

El siguiente ejemplo muestra cómo el maestro puede recibir datos de varios esclavos. En este ejemplo el esclavo envía dos números cortos. El primero es para la temperatura, y el segundo es para la humedad. Tenga en cuenta que la temperatura es un flotador (24.3). Para usar solo dos bytes y no cuatro (el flotador es de cuatro bytes), multiplico la temperatura en 10 y lo guardo como un corto. Así que aquí está el código maestro:

```
#include <Wire.h>

#define BUFFER_SIZE 4
#define MAX_NUMBER_OF_SLAVES 24
#define FIRST_SLAVE_ADDRESS 1
#define READ_CYCLE_DELAY 1000

byte buffer[BUFFER_SIZE];

void setup()
{
  Serial.begin(9600);
  Serial.println("MASTER READER");
  Serial.println("*****");

  Wire.begin();          // Activate I2C link
}

void loop()
{
  for (int slaveAddress = FIRST_SLAVE_ADDRESS;
       slaveAddress <= MAX_NUMBER_OF_SLAVES;
       slaveAddress++)
  {
    Wire.requestFrom(slaveAddress, BUFFER_SIZE);    // request data from the slave
    if(Wire.available() == BUFFER_SIZE)
    { // if the available data size is same as I'm expecting
      // Reads the buffer the slave sent
      for (int i = 0; i < BUFFER_SIZE; i++)
      {
```



```

        buffer[i] = Wire.read(); // gets the data
    }

    // Parse the buffer
    // In order to convert the incoming bytes info short, I use union
    union short_tag {
        byte b[2];
        short val;
    } short_cast;

    // Parse the temperature
    short_cast.b[0] = buffer[0];
    short_cast.b[1] = buffer[1];
    float temperature = ((float)(short_cast.val)) / 10;

    // Parse the moisture
    short_cast.b[0] = buffer[2];
    short_cast.b[1] = buffer[3];
    short moisture = short_cast.val;

    // Prints the income data
    Serial.print("Slave address ");
    Serial.print(slaveAddress);
    Serial.print(": Temprature = ");
    Serial.print(temperature);
    Serial.print("; Moisture = ");
    Serial.println(moisture);
}
}
Serial.println("*****");

delay(READ_CYCLE_DELAY);
}
}

```

Y ahora el código esclavo:

```

#include <Wire.h>
#include <OneWire.h>
#include <DallasTemperature.h>

//=====
// This is the hard-coded address. Change it from one device to another
#define SLAVE_ADDRESS 1
//=====

// I2C Variables
#define BUFFER_SIZE 2
#define READ_CYCLE_DELAY 1000
short data[BUFFER_SIZE];

// Temprature Variables
OneWire oneWire(8);
DallasTemperature temperatureSensors(&oneWire);
float m_temperature;

// Moisture Variables
short m_moisture;

// General Variables

```

```

int m_timestamp;

void setup()
{
  Serial.begin(9600);
  Serial.println("SLAVE SENDER");
  Serial.print("Node address: ");
  Serial.println(SLAVE_ADDRESS);
  Serial.print("Buffer size: ");
  Serial.println(BUFFER_SIZE * sizeof(short));
  Serial.println("*****");

  m_timestamp = millis();
  Wire.begin(NODE_ADDRESS); // Activate I2C network
  Wire.onRequest(requestEvent); // Set the request event handler
  temperatureSensors.begin();
}

void loop()
{
  if(millis() - m_timestamp < READ_CYCLE_DELAY) return;

  // Reads the temperature
  temperatureSensors.requestTemperatures();
  m_temperature = temperatureSensors.getTempCByIndex(0);

  // Reads the moisture
  m_moisture = analogRead(A0);
}

void requestEvent()
{
  data[0] = m_temperature * 10; // In order to use short, I multiple by 10
  data[1] = m_moisture;
  Wire.write((byte*)data, BUFFER_SIZE * sizeof(short));
}

```

Lea Comunicación I2C en línea: <https://riptutorial.com/es/arduino/topic/9092/comunicacion-i2c>

Capítulo 11: Comunicación MIDI

Introducción

La intención de este tema es mostrar algunos programas MIDI básicos que muestran cómo operar con el protocolo y agregan progresivamente funciones útiles que requieren las aplicaciones más complejas.

Examples

Ejemplo de MIDI THRU

El MIDI Thru es simple y fácil de probar. Cuando trabaje correctamente, podrá instalar su proyecto Arduino entre dos dispositivos MIDI, MIDI IN a MIDI OUT y podrá verificar que los dos dispositivos funcionan juntos. Si tiene la capacidad de medir la latencia, verá un aumento debido a las instrucciones de captura y retransmisión del búfer en serie.

```
// This is a simple MIDI THRU.  Everything in, goes right out.
// This has been validate on an Arduino UNO and a Olimex MIDI Shield

boolean byteReady;
unsigned char midiByte;

void setup() {
  // put your setup code here, to run once:
  // Set MIDI baud rate:
  Serial.begin(31250);
  byteReady = false;
  midiByte = 0;
}

// The Loop that always gets called...
void loop() {
  if (byteReady) {
    byteReady = false;
    Serial.write(midiByte);
  }
}

// The little function that gets called each time loop is called.
// This is automated somewhere in the Arduino code.
void serialEvent() {
  if (Serial.available()) {
    // get the new byte:
    midiByte = (unsigned char)Serial.read();
    byteReady = true;
  }
}
```

MIDI a través de la cola

```

// This is a more complex MIDI THRU. This version uses a queue. Queues are important because
some
// MIDI messages can be interrupted for real time events. If you are generating your own
messages,
// you may need to stop your message to let a "real time" message through and then resume your
message.

#define QUEUE_DEPTH 128

// Queue Logic for storing messages
int headQ = 0;
int tailQ = 0;
unsigned char tx_queue[QUEUE_DEPTH];

void setup() {
    // put your setup code here, to run once:
    // Set MIDI baud rate:
    Serial.begin(31250);
}

// getQDepth checks for roll over. Folks have told me this
// is not required. Feel free to experiment.
int getQDepth() {
int depth = 0;
    if (headQ < tailQ) {
        depth = QUEUE_DEPTH - (tailQ - headQ);
    } else {
        depth = headQ - tailQ;
    }
    return depth;
}

void addQueue (unsigned char myByte) {
    int depth = 0;
    depth = getQDepth();

    if (depth < (QUEUE_DEPTH-2)) {
        tx_queue[headQ] = myByte;
        headQ++;
        headQ = headQ % QUEUE_DEPTH; // Always keep the headQ limited between 0 and 127
    }
}

unsigned char deQueue() {
    unsigned char myByte;
    myByte = tx_queue[tailQ];
    tailQ++;
    tailQ = tailQ % QUEUE_DEPTH; // Keep this tailQ contained within a limit
    // Now that we dequeued the byte, it must be sent.
    return myByte;
}

void loop() {
    if (getQDepth>0) {
        Serial.write(deQueue());
    }
}

// The little function that gets called each time loop is called.
// This is automated somewhere in the Arduino code.

```

```

void serialEvent() {
  if (Serial.available()) {
    // get the new byte:
    addQueue((unsigned char)Serial.read());
  }
}

```

Generación de reloj MIDI

```

// This is a MiDI clk generator.  This takes a #defined BPM and
// makes the appropriate clk rate.  The queue is used to let other messages
// through, but allows a clock to go immediately to reduce clock jitter

#define QUEUE_DEPTH 128
#define BPM 121
#define MIDI_SYSRT_CLK 0xF8

// clock tracking and calculation
unsigned long lastClock;
unsigned long captClock;
unsigned long clk_period_us;

// Queue Logic for storing messages
int headQ = 0;
int tailQ = 0;
unsigned char tx_queue[QUEUE_DEPTH];

void setup() {
  // Set MIDI baud rate:
  Serial.begin(31250);
  clk_period_us = 60000000 / (24 * BPM);
  lastClock = micros();
}

// getQDepth checks for roll over.  Folks have told me this
// is not required.  Feel free to experiment.
int getQDepth() {
  int depth = 0;
  if (headQ < tailQ) {
    depth = QUEUE_DEPTH - (tailQ - headQ);
  } else {
    depth = headQ - tailQ;
  }
  return depth;
}

void addQueue (unsigned char myByte) {
  int depth = 0;
  depth = getQDepth();

  if (depth < (QUEUE_DEPTH-2)) {
    tx_queue[headQ] = myByte;
    headQ++;
    headQ = headQ % QUEUE_DEPTH; // Always keep the headQ limited between 0 and 127
  }
}

unsigned char deQueue() {
  unsigned char myByte;

```

```

myByte = tx_queue[tailQ];
tailQ++;
tailQ = tailQ % QUEUE_DEPTH; // Keep this tailQ contained within a limit
// Now that we dequeued the byte, it must be sent.
return myByte;
}

void loop() {
  captClock = micros();

  if (lastClock > captClock) {
    // we have a roll over condition - Again, maybe we don't need to do this.
    if (clk_period_us <= (4294967295 - (lastClock - captClock))) {
      // Add a the ideal clock period for this BPM to the last measurement value
      lastClock = lastClock + clk_period_us;
      // Send a clock, bypassing the transmit queue
      Serial.write(MIDI_SYSRT_CLK);
    }
  } else if (clk_period_us <= captClock-lastClock) {
    // Basically the same two commands above, but not within a roll over check
    lastClock = lastClock + clk_period_us;
    // Send a clock, bypassing the transmit queue
    Serial.write(MIDI_SYSRT_CLK);
  }

  if (getQDepth>0) {
    Serial.write(deQueue());
  }
}

// The little function that gets called each time loop is called.
// This is automated somewhere in the Arduino code.
void serialEvent() {
  if (Serial.available()) {
    // get the new byte:
    addQueue((unsigned char)Serial.read());
  }
}
}

```

Mensajes MIDI definidos

En general, el protocolo MIDI se divide en "mensajes". Hay 4 clases generales de mensajes:

- Canal de voz
- Modo de canal
- Sistema Común
- Mensajes en tiempo real del sistema

Los mensajes comienzan con un valor de byte por encima de 0x80. Cualquier valor por debajo de 0x7F se considera datos. Lo que significa efectivamente que 127 es el valor máximo que se puede codificar en un solo byte de datos MIDI. Para codificar valores más grandes, se requieren dos o más bytes de datos MIDI.

Debe señalarse que los mensajes deben enviarse de principio a fin sin interrupción ... EXCEPTO ... Los mensajes del sistema en tiempo real, que son un solo byte, que se pueden inyectar en medio de cualquier mensaje.

Mensajes de voz del canal

Estado D7..D0	Bytes de datos	Descripción
1000nnnn	0kkkkkkk 0vvvvvvvv	Evento Note Off. Este mensaje se envía cuando se suelta una nota (finaliza). (kkkkkkk) es el número clave (nota). (vvvvvvv) es la velocidad.
1001nnnn	0kkkkkkk 0vvvvvvvv	Nota sobre el evento. Este mensaje se envía cuando se presiona una nota (inicio). (kkkkkkk) es el número clave (nota). (vvvvvvv) es la velocidad.
1010nnnn	0kkkkkkk 0vvvvvvvv	Presión de tecla polifónica (aftertouch). Este mensaje se envía con mayor frecuencia presionando la tecla hacia abajo después de que "toque fondo". (kkkkkkk) es el número clave (nota). (vvvvvvv) es el valor de presión.
1011nnnn	0ccccccc 0vvvvvvvv	Control de cambio. Este mensaje se envía cuando un valor de controlador cambia. Los controladores incluyen dispositivos tales como pedales y palancas. Los números de controlador 120-127 están reservados como "Mensajes de modo de canal" (abajo). (ccccccc) es el número del controlador (0-119). (vvvvvvv) es el valor del controlador (0-127).
1100nnnn	0pppppppp	Cambio de programa. Este mensaje enviado cuando el número de parche cambia. (pppppppp) es el nuevo número de programa.
1101nnnn	0vvvvvvvv	Presión del canal (After-touch). Este mensaje se envía con mayor frecuencia presionando la tecla hacia abajo después de que "toque fondo". Este mensaje es diferente del post-toque polifónico. Use este mensaje para enviar el valor de presión máximo individual (de todas las teclas presionadas actuales). (vvvvvvv) es el valor de presión.
1110nnnn	0lllllll 0mmmmmmm	Cambio de Pitch Bend. Este mensaje se envía para indicar un cambio en el doblador de tono (rueda o palanca, típicamente). El doblador de tono se mide por un valor de catorce bits. El centro (sin cambio de tono) es 2000H. La sensibilidad es una función del receptor, pero se puede configurar utilizando RPN 0. (lllllll) son los 7 bits menos significativos. (mmmmmmm) son los 7 bits más significativos.

Mensajes de modo de canal

Estado D7..D0	Bytes de datos	Descripción
1011nnnn	0ccccccc 0vvvvvvvv	Mensajes de modo de canal. Este es el mismo código que el Cambio de control (arriba), pero implementa el control de Modo y el mensaje especial al usar los números de controlador reservados 120-127. Los comandos son:
		Todo el sonido apagado. Cuando se recibe Todo el sonido apagado, todos los osciladores se apagan y sus envolventes de volumen se ponen a cero tan pronto como sea posible. c = 120, v = 0: Todo el sonido apagado
		Restablecer todos los controladores. Cuando se recibe Restablecer todos los controladores, todos los valores del controlador se restablecen a sus valores predeterminados. (Ver las prácticas recomendadas específicas para los valores predeterminados).
		c = 121, v = x: el valor solo debe ser cero a menos que se permita lo contrario en una práctica recomendada específica.
		Control local. Cuando el control local está desactivado, todos los dispositivos en un canal dado responderán solo a los datos recibidos a través de MIDI. Los datos reproducidos, etc. serán ignorados. Control local activado restaura las funciones de los controladores normales.
		c = 122, v = 0: Control local desactivado
		c = 122, v = 127: Control local activado
		Todas las notas desactivadas. Cuando se reciben todas las notas desactivadas, todos los osciladores se apagarán.
		c = 123, v = 0: todas las notas desactivadas (consulte el texto para obtener una descripción de los comandos del modo real).
		c = 124, v = 0: Modo Omni desactivado
		c = 125, v = 0: Modo Omni activado
		c = 126, v = M: Modo mono activado (Poli apagado) donde M es el número de canales (Omni apagado) o 0 (Omni encendido)
		c = 127, v = 0: Modo poli activado (Mono apagado) (Nota: estos cuatro mensajes también causan todas las notas desactivadas)

Mensajes comunes del sistema

Estado D7..D0	Bytes de datos	Descripción
11110000	0iiiiiii [0iiiiiii 0iiiiiii] 0ddddddd --- --- 0ddddddd 11110111	Exclusivo del sistema. Este tipo de mensaje permite a los fabricantes crear sus propios mensajes (como volcados masivos, parámetros de parches y otros datos no especificados) y proporciona un mecanismo para crear mensajes de Especificación MIDI adicionales. El código de identificación del fabricante (asignado por MMA o AMEI) es 1 byte (0iiiiiii) o 3 bytes (0iiiiiii 0iiiiiii 0iiiiiii). Dos de las ID de 1 byte están reservadas para extensiones denominadas mensajes exclusivos universales, que no son específicos del fabricante. Si un dispositivo reconoce el código de ID como propio (o como un mensaje universal compatible) escuchará el resto del mensaje (0ddddddd). De lo contrario, el mensaje será ignorado. (Nota: solo los mensajes en tiempo real se pueden intercalar con un sistema exclusivo).
11110001	0nnndddd	Código de tiempo MIDI Quarter Frame. nnn = Tipo de mensaje dddd = Valores
11110010	0lllllll 0mmmmmmm	Puntero de posición de la canción. Este es un registro interno de 14 bits que contiene el número de tiempos MIDI (1 tiempo = seis relojes MIDI) desde el inicio de la canción. l es el LSB, m el MSB.
11110011	0sssssss	Seleccione la canción. La Selección de canción especifica qué secuencia o canción se reproducirá.
11110100		Indefinido (Reservado)
11110101		Indefinido (Reservado)
11110110		Solicitud de melodía Al recibir una solicitud de sintonización, todos los sintetizadores analógicos deben sintonizar sus osciladores.
11110111		Fin de Exclusivo. Se utiliza para terminar un volcado exclusivo del sistema (ver arriba).

Mensajes en tiempo real del sistema

Estado D7..D0	Bytes de datos	Descripción
11111000		Reloj de tiempo Se envía una nota de 24 veces por trimestre cuando se requiere sincronización (ver texto).

Estado D7..D0	Bytes de datos	Descripción
11111001		Indefinido (Reservado)
11111010		Comienzo. Inicia la secuencia actual de reproducción. (Este mensaje será seguido con los relojes de tiempo).
11111011		Continuar. Continuar en el punto en que se detuvo la secuencia.
11111100		Detener. Detener la secuencia actual.
11111101		Indefinido (Reservado)
11111110		Detección activa Este mensaje está destinado a enviarse repetidamente para indicar al receptor que una conexión está activa. El uso de este mensaje es opcional. Cuando se recibe inicialmente, el receptor esperará recibir otro mensaje de detección activa cada 300 ms (máx.), Y si no lo hace, asumirá que la conexión ha finalizado. En la terminación, el receptor apagará todas las voces y volverá a la operación normal (detección no activa).
11111111		Reiniciar. Restablecer todos los receptores en el sistema al estado de encendido. Esto debe usarse con moderación, preferiblemente bajo control manual. En particular, no debe enviarse en el encendido.

Lea Comunicación MIDI en línea: <https://riptutorial.com/es/arduino/topic/9406/comunicacion-midi>

Capítulo 12: Comunicación serial

Sintaxis

- `Serial.begin(speed)` // Abre el puerto serie en la velocidad de transmisión dada
- `Serial.begin(speed, config)`
- `Serial[1-3].begin(speed)` // **Arduino Mega solamente!** Al escribir 1-3 significa que puede elegir entre los números 1 a 3 al elegir el puerto serie.
- `Serial[1-3].begin(speed, config)` // ¡ **Arduino Mega solamente!** Al escribir 1-3 significa que puede elegir entre los números 1 a 3 al elegir el puerto serie.
- `Serial.peek()` // Lee el siguiente byte de entrada sin eliminarlo del búfer
- `Serial.available()` // Obtiene el número de bytes en el búfer
- `Serial.print(text)` // Escribe texto en el puerto serie
- `Serial.println(text)` // Igual que `Serial.print()` pero con una nueva línea final

Parámetros

Parámetro	Detalles
Velocidad	La tasa del puerto serie (generalmente 9600)
Texto	El texto para escribir en el puerto serie (cualquier tipo de datos)
Bits de datos	Número de bits de datos en un paquete (de 5 a 8), el valor predeterminado es 8
Paridad	Opciones de paridad para la detección de errores: ninguno (predeterminado), par, impar
Bits de parada	Número de bits de parada en un paquete: uno (predeterminado), dos

Observaciones

El Arduino Mega tiene cuatro puertos serie que se pueden elegir. Se accede de la siguiente manera.

```
Serial.begin(9600);  
Serial1.begin(38400);  
Serial2.begin(19200);  
Serial3.begin(4800);
```

El puerto serie en un Arduino se puede configurar con parámetros adicionales. El parámetro `config` establece bits de datos, paridad y bits de parada. Por ejemplo:

8 bits de datos, paridad par y 1 bit de parada serían - SERIAL_8E1

6 bits de datos, paridad impar y 2 bits de parada serían - SERIAL_6O2

7 bits de datos, sin paridad y 1 bit de parada serían - SERIAL_7N1

Examples

Simple leer y escribir

Este ejemplo escucha la entrada que llega a través de la conexión en serie y luego la repite nuevamente en la misma conexión.

```
byte incomingBytes;

void setup() {
  Serial.begin(9600); // Opens serial port, sets data rate to 9600 bps.
}

void loop() {
  // Send data only when you receive data.
  if (Serial.available() > 0) {
    // Read the incoming bytes.
    incomingBytes = Serial.read();

    // Echo the data.
    Serial.println(incomingBytes);
  }
}
```

Filtrado Base64 para datos de entrada serie

```
String base64="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=";

void setup() {

  Serial.begin(9600); // Turn the serial protocol ON
  Serial.println("Start Typing");
}

void loop() {

  if (Serial.available() > 0) { // Check if data has been sent from the user
    char c = Serial.read(); // Gets one byte/Character from serial buffer
    int result = base64.indexOf(c); // Base64 filtering
    if (result>=0)
      Serial.print(c); // Only print Base64 string
  }
}
```

Manejo de comandos sobre serie

```
byte incoming;
```

```

String inBuffer;

void setup() {
  Serial.begin(9600); // or whatever baud rate you would like
}

void loop(){
  // setup as non-blocking code
  if(Serial.available() > 0) {
    incoming = Serial.read();

    if(incoming == '\n') { // newline, carriage return, both, or custom character

      // handle the incoming command
      handle_command();

      // Clear the string for the next command
      inBuffer = "";
    } else{
      // add the character to the buffer
      inBuffer += incoming;
    }
  }

  // since code is non-blocking, execute something else . . . .
}

void handle_command() {
  // expect something like 'pin 3 high'
  String command = inBuffer.substring(0, inBuffer.indexOf(' '));
  String parameters = inBuffer.substring(inBuffer.indexOf(' ') + 1);

  if(command.equalsIgnoreCase('pin')){
    // parse the rest of the information
    int pin = parameters.substring("0, parameters.indexOf(' ')).toInt();
    String state = parameters.substring(parameters.indexOf(' ') + 1);

    if(state.equalsIgnoreCase('high')){
      digitalWrite(pin, HIGH);
    }else if(state.equalsIgnoreCase('low')){
      digitalWrite(pin, LOW);
    }else{
      Serial.println("did not compute");
    }
  } // add code for more commands
}

```

Comunicación serial con Python

Si tiene un Arduino conectado a una computadora o una Raspberry Pi y desea enviar datos desde el Arduino a la PC, puede hacer lo siguiente:

Arduino:

```

void setup() {
  // Opens serial port, sets data rate to 9600 bps:

```

```
Serial.begin(9600);
}

void loop() {
  // Sends a line over serial:
  Serial.println("Hello, Python!");
  delay(1000);
}
```

Pitón:

```
import serial

ser = serial.Serial('/dev/ttyACM0', 9600) # Start serial communication
while True:
    data = ser.readline() # Wait for line from Arduino and read it
    print("Received: '{}'.format(data)) # Print the line to the console
```

Lea Comunicación serial en línea: <https://riptutorial.com/es/arduino/topic/1674/comunicacion-serial>

Capítulo 13: Comunicación SPI

Observaciones

Señales de selección de chip

La mayoría de los esclavos tienen una entrada activa de selección de chip bajo. El código correcto para inicializar y usar un pin de selección de chip es este:

```
#define CSPIN 1 // or whatever else your CS pin is
// init:
pinMode(CSPIN, OUTPUT);
digitalWrite(CSPIN, 1); // deselect

// use:
digitalWrite(CSPIN, 0); // select
... perform data transfer ...
digitalWrite(CSPIN, 1); // deselect
```

Deseleccionar un esclavo es tan importante como seleccionarlo, porque un esclavo puede manejar la línea MISO mientras está seleccionado. Puede haber muchos esclavos, pero solo uno puede manejar MISO. Si un esclavo no se deselecta correctamente, dos o más esclavos pueden estar manejando MISO, lo que puede provocar cortocircuitos entre sus salidas y dañar los dispositivos.

Actas

Las transacciones tienen dos propósitos:

- avise al SPI cuando queremos comenzar y terminar de usarlo dentro de un contexto particular
- configurar el SPI para un chip específico

La línea del reloj tiene diferentes estados de inactividad en los diferentes modos SPI. Cambiar el modo SPI mientras se selecciona un esclavo puede confundir al esclavo, así que siempre configure el modo SPI antes de seleccionar un esclavo. El modo SPI se puede configurar con un objeto `SPISettings` pasado a `SPI.beginTransaction`:

```
SPI.beginTransaction(SPISettings(1000000, MSBFIRST, SPI_MODE0));
digitalWrite(CSPIN, 0);
... perform data transfer ...
digitalWrite(CSPIN, 1);
SPI.endTransaction();
```

`SPISettings` también puede ser almacenado en otro lugar:

```
SPISettings mySettings(1000000, MSBFIRST, SPI_MODE0);
SPI.beginTransaction(mySettings);
```

Si otra parte del código intenta usar el SPI entre un par de llamadas para `beginTransaction()` y `endTransaction()` , puede surgir un error: la forma en que se hace depende de la implementación.

También vea [Arduino Reference: SPISettings](#)

Usando el SPI en las rutinas de servicio de interrupción

Si se debe usar el SPI dentro de un ISR, no se puede realizar ninguna otra transacción al mismo tiempo. La biblioteca SPI proporciona `usingInterrupt(interrupt_number)` para facilitar esto.

Funciona al deshabilitar la interrupción dada cada vez que se llama a `beginTransaction()` , por lo que la interrupción no se puede disparar entre ese par de llamadas a `beginTransaction()` y `endTransaction()` .

También vea [Arduino Reference: SPI: usingInterrupt](#)

Examples

Conceptos básicos: inicialice el SPI y un pin de selección de chip, y realice una transferencia de 1 byte

```
#include <SPI.h>
#define CSPIN 1

void setup() {
  pinMode(CSPIN, OUTPUT); // init chip select pin as an output
  digitalWrite(CSPIN, 1); // most slaves interpret a high level on CS as "deasserted"

  SPI.begin();

  SPI.beginTransaction(SPISettings(1000000, MSBFIRST, SPI_MODE0));
  digitalWrite(CSPIN, 0);

  unsigned char sent = 0x01;
  unsigned char received = SPI.transfer(sent);
  // more data could be transferred here

  digitalWrite(CSPIN, 1);
  SPI.endTransaction();

  SPI.end();
}

void loop() {
  // we don't need loop code in this example.
}
```


Este ejemplo:

- Inicializa correctamente y utiliza un pin de selección de chip (ver comentarios)
- utiliza correctamente una transacción SPI (ver comentarios)
- Solo usa el SPI para transferir un solo byte. También hay un método para transferir matrices, que no se utiliza aquí.

Lea Comunicación SPI en línea: <https://riptutorial.com/es/arduino/topic/4919/comunicacion-spi>

Capítulo 14: Entradas analógicas

Sintaxis

- `analogRead(pin)` // Leer desde el pin dado.

Observaciones

```
Serial.println(val)
```

Para obtener ayuda con la comunicación en serie, consulte: [Comunicación en serie](#)

Examples

Imprima un valor analógico

```
int val = 0;    // variable used to store the value
                // coming from the sensor

void setup() {
  Serial.begin(9600); //Begin serializer to print out value

  // Note: Analogue pins are
  // automatically set as inputs
}

void loop() {

  val = analogRead(0); // read the value from
                       // the sensor connected to A0.

  Serial.println(val); //Prints the value coming in from the analog sensor

  delay(10); // stop the program for
             // some time
}
```

Obtener voltaje de pin analógico

Las clavijas analógicas se pueden usar para leer voltajes, lo que es útil para monitorear la batería o interactuar con dispositivos analógicos. Por defecto, el pin AREF será el mismo que el voltaje de operación del arduino, pero puede establecerse en otros valores externamente. Si el voltaje a leer es mayor que el voltaje de entrada, se necesitará un desviador potencial para disminuir el voltaje analógico.

```
#define analogPin 14    //A0 (uno)
#define AREFValue 5     //Standard for 5V Arduinos
#define ADCResolution 1023 //Standard for a 10bit ADC
```

```
int ADCValue = 0;
float voltage = 0;

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  readADC();
  Serial.print(voltage); Serial.println("V");
}

void readADC()
{
  ADCValue = analogRead(analogPin);
  float = ( (float)ADCValue/ADCRANGE ) * AREFValue ); //Convert the ADC value to a
float, divide by the ADC resolution and multiply by the AREF voltage
}
```

Lea Entradas analógicas en línea: <https://riptutorial.com/es/arduino/topic/2382/entradas-analogicas>

Capítulo 15: Entradas digitales

Sintaxis

- `pinMode(pin, pinMode)` // Establece el pin en el modo definido.
- `digitalRead(pin);` // Lee el valor de un pin digital especificado,

Parámetros

Parámetro	Detalles
modo de pin	Debe establecerse en <code>INPUT</code> o <code>INPUT_PULLUP</code>

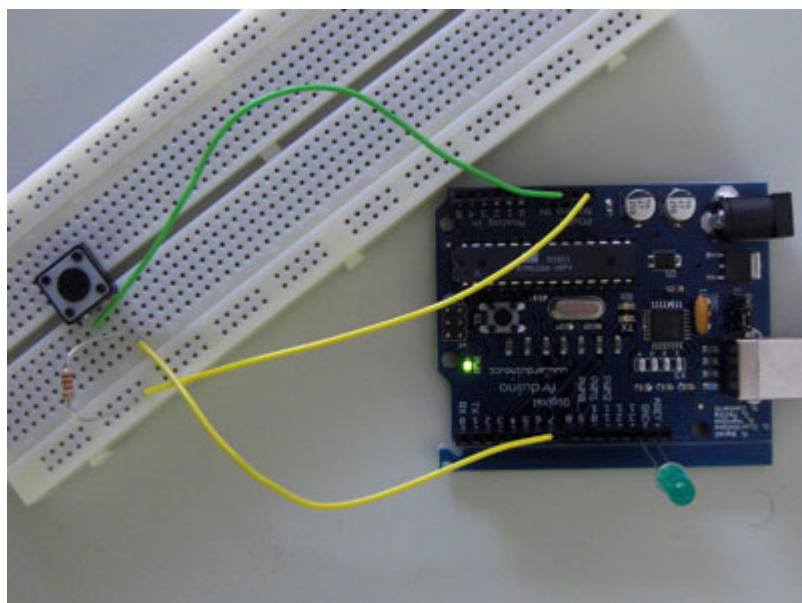
Observaciones

Si el pin de entrada no se presiona LOW o HIGH, el valor flotará. Es decir, no será claramente un 1 o un 0, sino un punto intermedio. Para entrada digital, una resistencia pullup o desplegable es una necesidad.

Examples

Pulsador de lectura

Este es un ejemplo básico sobre cómo cablear y hacer que un LED se encienda / apague cuando se presiona el botón.



```
/* Basic Digital Read
 * -----
```

```

*
* turns on and off a light emitting diode(LED) connected to digital
* pin 13, when pressing a pushbutton attached to pin 7. It illustrates the
* concept of Active-Low, which consists in connecting buttons using a
* 1K to 10K pull-up resistor.
*
* Created 1 December 2005
* copyleft 2005 DojoDave <http://www.0j0.org>
* http://arduino.berlios.de
*
*/

int ledPin = 13; // choose the pin for the LED
int inPin = 7;   // choose the input pin (for a pushbutton)
int val = 0;     // variable for reading the pin status

void setup() {
  pinMode(ledPin, OUTPUT); // declare LED as output
  pinMode(inPin, INPUT);   // declare pushbutton as input
}

void loop(){
  val = digitalRead(inPin); // read input value
  if (val == HIGH) {        // check if the input is HIGH (button released)
    digitalWrite(ledPin, LOW); // turn LED OFF
  } else {
    digitalWrite(ledPin, HIGH); // turn LED ON
  }
}

```

Ejemplo tomado de Arduino.cc .

Lea Entradas digitales en línea: <https://riptutorial.com/es/arduino/topic/1662/entradas-digitales>

Capítulo 16: Funciones

Observaciones

Aparte de en C / C ++ ordinario, el IDE de Arduino permite llamar a una función antes de que se defina.

En los archivos .cpp, debe definir la función, o al menos declarar la función prototipo antes de poder usarla.

En un archivo .ino, el IDE de Arduino crea un prototipo detrás de escena.

[Arduino - declaracion de funcion - oficial](#)

Examples

Crear una función simple.

```
int squareNum (int a) {  
    return a*a;  
}
```

`int` : tipo de retorno

`squareNum` : nombre de la función

`int a` : tipo de parámetro y nombre

`return a*a` : devolver un valor (el mismo tipo que el tipo de retorno definido al principio)

Anatomy of a C function

Datatype of data returned,
any C datatype.

"void" if nothing is returned.

Parameters passed to
function, any C datatype.

Function name

```
int myMultiplyFunction(int x, int y){  
    int result;  
    result = x * y;  
    return result;  
}
```

Return statement,
datatype matches
declaration.

Curly braces required.

Llamar a una función

Si tiene una función declarada, puede llamarla en cualquier otro lugar del código. Aquí hay un

ejemplo de llamar a una función:

```
void setup(){
  Serial.begin(9600);
}

void loop() {
  int i = 2;

  int k = squareNum(i); // k now contains 4
  Serial.println(k);
  delay(500);
}

int squareNum(int a) {
  return a*a;
}
```

Lea Funciones en línea: <https://riptutorial.com/es/arduino/topic/2380/funciones>

Capítulo 17: Gestión del tiempo

Sintaxis

- `millis` largos sin firmar ()
- `micros` largos sin firmar ()
- retraso de vacío (milisegundos largos sin firmar)
- `void delayMicroseconds` (microsegundos largos sin firmar)
- Consulte [el encabezado `elapsedMillis`](#) para los constructores y operadores de esa clase. En breve:
 - `elapsedMillis elapsedMillisObject`; *crea un objeto para realizar un seguimiento del tiempo desde que se creó o desde algún otro punto de tiempo establecido explícitamente*
 - `elapsedMillisObject = 0`; *restablecer el tiempo seguido por el objeto a "desde ahora"*
 - `sin signo long deltaT = elapsedMillisObject`; *nos deja mirar el tiempo seguido*
 - `elapsedMillisObject + = y - =` *funcionan como se esperaba*

Observaciones

Bloqueo contra código no bloqueante

Para bocetos muy simples, escribir código de bloqueo usando `delay()` y `delayMicroseconds()` puede ser apropiado. Cuando las cosas se vuelven más complejas, el uso de estas funciones puede tener algunos inconvenientes. Algunos de estos son:

- Perder el tiempo de la CPU: es posible que los bocetos más complejos necesiten la CPU para otra cosa mientras esperan que termine el período de parpadeo del LED.
- retrasos inesperados: cuando se llama a `delay()` en subrutinas que obviamente no se llaman, por ejemplo, en las bibliotecas que incluye.
- eventos faltantes que ocurren durante el retraso y no son manejados por un manejador de interrupciones, por ejemplo, al presionar un botón de sondeo: un botón puede presionarse por 100 ms, pero esto puede estar sombreado por un `delay(500)`.

Detalles de implementación

`millis()` generalmente se basa en un temporizador de hardware que se ejecuta a una velocidad que es mucho mayor que 1 kHz. Cuando se llama a `millis()`, la implementación devuelve algún valor, pero no se sabe qué edad tiene realmente. Es posible que el milisegundo "actual" haya comenzado, o que termine justo después de esa llamada de función. Eso significa que, al calcular la diferencia entre dos resultados de `millis()`, puede estar desconectado entre casi cero y casi

un milisegundo. Use `micros()` si se necesita mayor precisión.

Al examinar el código fuente de `elapsedMillis` revela que, efectivamente, utiliza `millis()` internamente para comparar dos puntos en el tiempo, por lo que también sufre este efecto. Una vez más, hay la alternativa de `elapsedMicros` para una mayor precisión, desde la misma biblioteca.

Examples

bloqueando blinky con retraso ()

Una de las formas más directas de hacer que parpadee un LED es: enciéndalo, espere un poco, apáguelo, espere de nuevo y repita sin cesar:

```
// set constants for blinking the built-in LED at 1 Hz
#define OUTPIN LED_BUILTIN
#define PERIOD 500

void setup()
{
  pinMode(OUTPIN, OUTPUT);    // sets the digital pin as output
}

void loop()
{
  digitalWrite(OUTPIN, HIGH); // sets the pin on
  delayMicroseconds(PERIOD);  // pauses for 500 microseconds
  digitalWrite(OUTPIN, LOW);  // sets the pin off
  delayMicroseconds(PERIOD);  // pauses for 500 milliseconds

  // doing other time-consuming stuff here will skew the blinking
}
```

Sin embargo, esperar como se hace en el ejemplo anterior desperdicia los ciclos de la CPU, ya que simplemente permanece allí en un bucle esperando que pase un cierto punto en el tiempo. Eso es lo que hacen mejor las formas de no bloqueo, utilizando `millis()` o `elapsedMillis`, en el sentido de que no queman la mayor parte de las capacidades del hardware.

Blinky sin bloqueo con la librería `elapsedMillis` (y clase)

La [biblioteca `elapsedMillis`](#) proporciona una clase con el mismo nombre que realiza un seguimiento del tiempo transcurrido desde que se creó o se estableció en un determinado valor:

```
#include <elapsedMillis.h>

#define OUTPIN LED_BUILTIN
#define PERIOD 500

elapsedMillis ledTime;

bool ledState = false;

void setup()
{
```

```

// initialize the digital pin as an output.
pinMode(OUTPIN, OUTPUT);
}

void loop()
{
  if (ledTime >= PERIOD)
  {
    ledState = !ledState;
    digitalWrite(OUTPIN, ledState);
    ledTime = 0;
  }
  // do other stuff here
}

```

Puede ver en el ejemplo que al objeto `ledTime` se le asigna un cero cuando se `ledTime` el pin LED. Esto puede no ser sorprendente a primera vista, pero tiene un efecto si suceden cosas que consumen más tiempo:

Considere una situación en la que la comparación entre `ledTime` y `PERIOD` se realiza después de 750 milisegundos. Luego, establecer `ledTime` en cero significa que todas las siguientes operaciones de conmutación tendrán un retraso de 250 ms. Si, por el contrario, el `PERIOD` se resta de `ledTime`, el LED verá un período corto y luego continuará parpadeando como si no hubiera pasado nada.

Blinky sin bloqueo con millis ()

Esto está muy cerca de [un ejemplo de la documentación de arduino](#) :

```

// set constants for blinking the built-in LED at 1 Hz
#define OUTPIN LED_BUILTIN
#define PERIOD 500 // this is in milliseconds

int ledState = LOW;

// millis() returns an unsigned long so we'll use that to keep track of time
unsigned long lastTime = 0;

void setup() {
  // set the digital pin as output:
  pinMode(OUTPIN, OUTPUT);
}

void loop() {
  unsigned long now = millis();
  if (now - lastTime >= PERIOD) // this will be true every PERIOD milliseconds
  {
    lastTime = now;
    if (ledState == LOW)
    {
      ledState = HIGH;
    }
    else
    {
      ledState = LOW;
    }
  }
}

```

```
    digitalWrite(OUTPIN, ledState);
}

// now there's lots of time to do other stuff here
}
```

El uso de `millis()` de esta manera (para `elapsedMillis` las operaciones de manera no bloqueante) es algo que se necesita con bastante frecuencia, así que considere usar la biblioteca de `elapsedMillis` para esto.

Mida cuánto tiempo tomó algo, utilizando `elapsedMillis` y `elapsedMicros`

```
#include <elapsedMillis.h>

void setup() {
  Serial.begin(115200);
  elapsedMillis msTimer;
  elapsedMicros usTimer;

  long int dt = 500;
  delay(dt);

  long int us = usTimer;
  long int ms = msTimer;

  Serial.print("delay("); Serial.print(dt); Serial.println(") took");
  Serial.print(us); Serial.println(" us, or");
  Serial.print(ms); Serial.println(" ms");
}

void loop() {
}
```

En este ejemplo, un objeto `elapsedMillis` y un objeto `elapsedMicros` se utilizan para medir cuánto tiempo tardó algo, al crearlos justo antes de que se ejecute la expresión que queremos medir y obtener sus valores posteriormente. Mostrarán resultados ligeramente diferentes, pero el resultado de milisegundos no se desactivará en más de un milisegundo.

Más de 1 tarea sin demora ()

Si tiene más de 1 tarea para ejecutar repetidamente en diferentes intervalos, use este ejemplo como punto de partida:

```
unsigned long intervals[] = {250,2000}; //this defines the interval for each task in
milliseconds
unsigned long last[] = {0,0};           //this records the last executed time for each task

void setup() {
  pinMode(LED_BUILTIN, OUTPUT); //set the built-in led pin as output
  Serial.begin(115200);         //initialize serial
}

void loop() {
  unsigned long now = millis();
```

```

if(now-last[0]>=intervals[0]){ last[0]=now; firstTask(); }
if(now-last[1]>=intervals[1]){ last[1]=now; secondTask(); }

//do other things here
}

void firstTask(){
  //let's toggle the built-in led
  digitalWrite(LED_BUILTIN, digitalRead(LED_BUILTIN)?0:1);
}

void secondTask(){
  //say hello
  Serial.println("hello from secondTask()");
}

```

Para agregar otra tarea para ejecutar cada 15 segundos, extienda los `intervals` las variables y `last` :

```

unsigned long intervals[] = {250,2000,15000};
unsigned long last[] = {0,0,0};

```

Luego agrega una sentencia `if` para ejecutar la nueva tarea. En este ejemplo, lo nombré `thirdTask` .

```

if(now-last[2]>=intervals[2]){ last[2]=now; thirdTask(); }

```

Finalmente declarar la función:

```

void thirdTask(){
  //your code here
}

```

Lea Gestión del tiempo en línea: <https://riptutorial.com/es/arduino/topic/4852/gestion-del-tiempo>

Capítulo 18: Interrupciones

Sintaxis

- `digitalPinToInterrupt (pin); // convierte un id de pin en un id de interrupción, para usar con attachInterrupt () y detachInterrupt () .`
- `attachInterrupt (digitalPinToInterrupt (pin), ISR, modo); // recomendado`
- `attachInterrupt (interrupción, ISR, modo); // no recomendado`
- `detachInterrupt (digitalPinToInterrupt (pin));`
- `detachInterrupt (interrupción);`
- `noInterrupts (); // desactiva las interrupciones`
- `interrupciones (); // volver a habilitar las interrupciones después de que se haya llamado a noInterrupts () .`

Parámetros

Parámetro	Notas
interrumpir	Id de la interrupción. No debe confundirse con el número de pin.
ISR	Rutina de servicio de interrupción. Este es el método que se ejecutará cuando ocurra la interrupción.
modo	Lo que debería hacer que la interrupción se dispare. Uno de LOW, CHANGE, RISING, o FALLING. Las tablas vencidas también permiten ALTA.

Observaciones

Las rutinas de servicio de interrupción (ISR) deben ser lo más cortas posible, ya que pausan la ejecución del programa principal y, por lo tanto, pueden arruinar el código dependiente del tiempo. Por lo general, esto significa que en el ISR establece un indicador y sale, y en el ciclo de programa principal, comprueba el indicador y hace lo que se supone que debe hacer ese indicador.

No puede usar `delay ()` o `millis ()` en un ISR porque esos métodos se basan en interrupciones.

Examples

Interrupción al presionar un botón

Este ejemplo utiliza un botón pulsador (interruptor de tacto) conectado al pin digital 2 y GND, utilizando una resistencia de pull-up interna, de modo que el pin 2 está ALTO cuando no se presiona el botón.

```
const int LED_PIN = 13;
const int INTERRUPT_PIN = 2;
volatile bool ledState = LOW;

void setup() {
  pinMode(LED_PIN, OUTPUT);
  pinMode(INTERRUPT_PIN, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), myISR, FALLING); // trigger when
  button pressed, but not when released.
}

void loop() {
  digitalWrite(LED_PIN, ledState);
}

void myISR() {
  ledState = !ledState;
  // note: LOW == false == 0, HIGH == true == 1, so inverting the boolean is the same as
  switching between LOW and HIGH.
}
```

Un ejemplo simple es que los botones pulsadores tienden a rebotar, lo que significa que al presionar o soltar, el circuito se abre y se cierra más de una vez antes de que se establezca en el estado final cerrado o abierto. Este ejemplo no tiene eso en cuenta. Como resultado, a veces, al presionar el botón se alternará el LED varias veces, en lugar del esperado una vez.

Lea Interrupciones en línea: <https://riptutorial.com/es/arduino/topic/2913/interrupciones>

Capítulo 19: Números al azar

Sintaxis

- `random (max)` // Devuelve un número pseudo-random (long) entre 0 (inclusive) y max (exclusive)
- `random (min, max)` // Devuelve un número pseudo-random (long) entre min (inclusive) y max (exclusive)
- `randomSeed (seed)` // Inicializa el generador de números pseudoaleatorios, lo que hace que se inicie en un punto específico de su secuencia.

Parámetros

Parámetro	Detalles
min	El valor mínimo posible (incluido) que generará la función <code>random()</code> .
max	El valor máximo posible (exclusivo) que generará la función <code>random()</code> .
semilla	La semilla que se utilizará para barajar la función <code>random()</code> .

Observaciones

Si se llama a `randomSeed()` con un valor fijo (por ejemplo, `randomSeed(5)`), la secuencia de números aleatorios generados por el boceto se repetirá cada vez que se ejecute. En la mayoría de los casos, se prefiere una semilla aleatoria, que puede obtenerse leyendo un pin analógico no conectado.

Examples

Generar un número aleatorio

La función `random()` se puede usar para generar números pseudoaleatorios:

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  long randomNumber = random(500); // Generate a random number between 0 and 499
  Serial.println(randomNumber);

  randomNumber = random(100, 1000); // Generate a random number between 100 and 999
  Serial.println(randomNumber);
}
```

```
    delay(100);  
}
```

Poniendo una semilla

Si es importante que una secuencia de números generados por `random()` difiera, es una buena idea especificar una semilla con `randomSeed()` :

```
void setup() {  
    Serial.begin(9600);  
  
    // If analog pin 0 is left unconnected, analogRead(0) will produce a  
    // different random number each time the sketch is run.  
    randomSeed(analogRead(0));  
}  
  
void loop() {  
    long randomNumber = random(500); // Generate a random number between 0 and 499  
    Serial.println(randomNumber);  
  
    delay(100);  
}
```

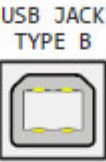
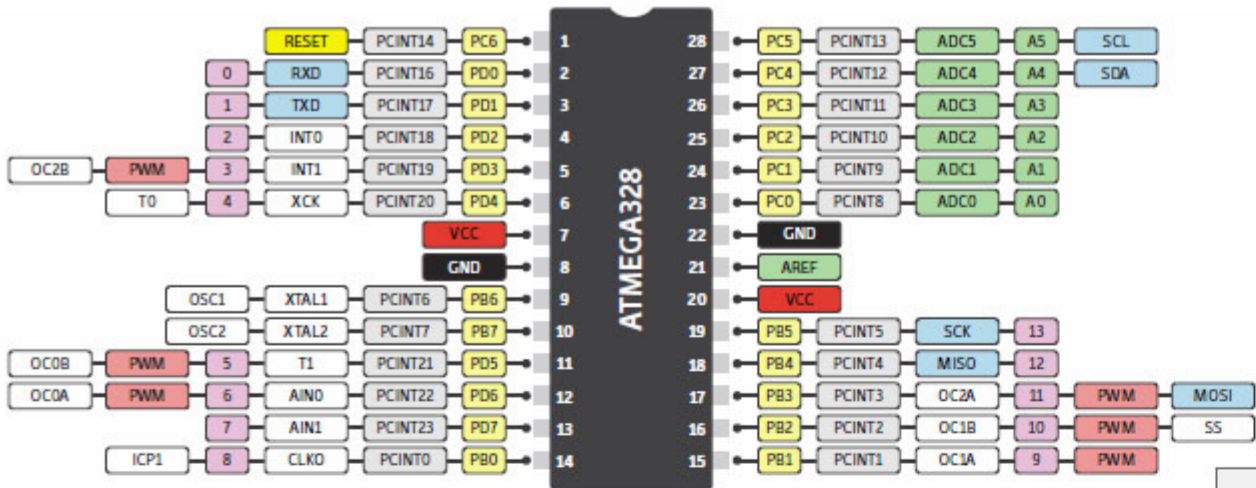
Lea Números al azar en línea: <https://riptutorial.com/es/arduino/topic/2238/numeros-al-azar>

Capítulo 20: Pines de hardware

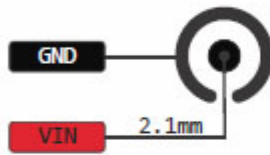
Examples

Arduino Uno R3

Los microcontroladores utilizan pines para interactuar con el resto del circuito. Estos pines normalmente serán uno de los pines de entrada / salida, vin o tierra. Los pines de E / S pueden ser pines de E / S digitales simples, o pueden tener algunas características especiales, como la posibilidad de variar el voltaje de su salida mediante la modulación de ancho de pulso. Aquí hay un esquema del Arduino R3 Uno y sus pines.

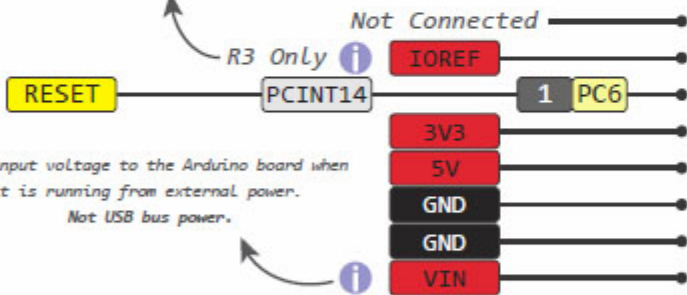


7-12V Depending on current drawn

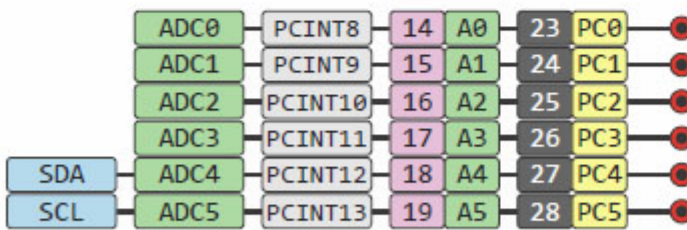


Cut to disable the auto-reset

This provides a Logic reference voltage for shields that use it. It is connected to the 5V bus.



The input voltage to the Arduino board when it is running from external power. Not USB bus power.

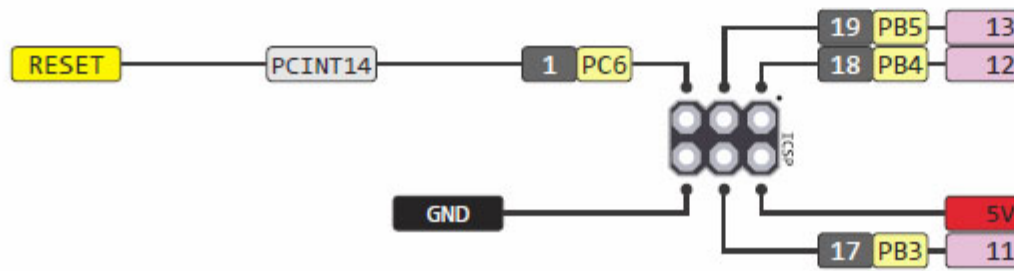


www.pigpio.com



18 FEB 2013

ver 2 rev 2 - 05.03.2013



(fuente)

PWM Pins

PWM le permite controlar el voltaje de la salida cambiando la salida entre alta y baja muy rápidamente. El porcentaje de tiempo que el pin es alto se denomina "ciclo de trabajo".

Pasadores de PWM: 3, 5, 6, 9, 10, 11

Entradas analógicas

Al igual que un pin PWM puede generar un rango de voltajes, los pines analógicos en el Arduino Uno R3 pueden detectar un rango de voltajes de entrada. Puede usar esto para leer la posición de un potenciómetro u otra entrada con una entrada suavemente variable. Tenga en cuenta que los pines analógicos no pueden hacer una salida de WDS analógica, para esto necesita usar pines PWM.

Pines analógicos ADC: A0, A1, A2, A3, A4, A5

Serial, SPI y I2C

Los pines serie en el Arduino Uno R3 también son utilizados por (por ejemplo) el chip USB a Serial cuando se comunica con una computadora a través del puerto USB incorporado. Serie: Tx en 0, Rx en 1

SPI e I2C son protocolos de comunicación que Arduino puede usar para hablar con escudos, sensores, salidas, etc ...:

Pernos SPI: MOSI en 11, MISO en 12, SCLK en 13, SS en 10

Pernos I2C: SCL en A5, SDA en A4

LED de a bordo

El Arduino Uno R3 tiene un LED con su propia resistencia conectada al pin 13. Esto significa que incluso si no conecta ningún LED a su tablero, si configura el pin 13 en una salida y lo establece en un nivel alto, debería ver un LED. en el tablero vamos Use el bosquejo de ejemplo 'Parpadeo' para ubicar su LED incorporado.

De la página de [Arduino Digital Pins](#)

NOTA: El pin digital 13 es más difícil de usar como una entrada digital que los otros pines digitales porque tiene un LED y una resistencia unidos a la placa en la mayoría de las placas. Si habilita su resistencia de pull-up interna de 20k, se colgará a alrededor de 1.7V en lugar de los 5V esperados porque el LED a bordo y la resistencia en serie bajan el nivel de voltaje, lo que significa que siempre retorna BAJO. Si debe usar el pin 13 como entrada digital, configure su pinMode () en ENTRADA y use una resistencia desplegable externa.

Pin LED de a bordo: 13

Lea Pines de hardware en línea: <https://riptutorial.com/es/arduino/topic/4386/pines-de-hardware>

Capítulo 21: PWM - Modulación de ancho de pulso

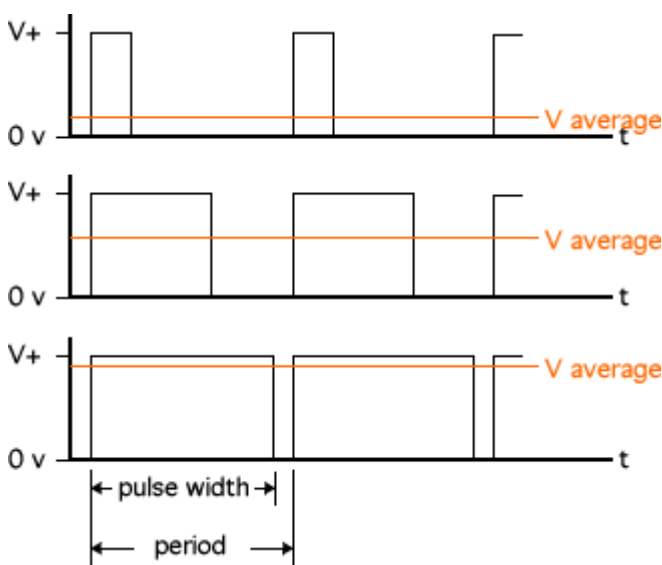
Examples

Controlar un motor de CC a través del puerto serie utilizando PWM

En este ejemplo, nuestro objetivo es lograr una de las tareas más comunes: *tengo un pequeño motor de corriente continua, ¿cómo uso mi Arduino para controlarlo?* Fácil, con PWM y comunicación serial, usando la función `analogWrite()` y la biblioteca `Serial`.

Los basicos

La modulación de ancho de pulso o PWM para abreviar es una técnica para imitar señales analógicas usando una salida digital. ¿Como funciona esto? Usando un tren de impulsos cuya relación D (ciclo de trabajo) entre el tiempo en el nivel alto (digital 1, generalmente 5V) y el tiempo en el nivel bajo (digital 0, 0V) en cada período se puede modificar para producir un voltaje promedio entre estos dos niveles:



Al utilizar la función `analogWrite(pin, value)` Arduino `analogWrite(pin, value)` podemos variar el `value` del ciclo de trabajo de la salida del `pin`. Tenga en cuenta que el `pin` debe ponerse en modo de salida y el `value` debe estar entre 0 (0V) y 255 (5V). Cualquier valor intermedio simulará una salida analógica intermedia proporcional.

Sin embargo, el propósito de las señales analógicas generalmente se relaciona con el control de los sistemas mecánicos que requieren más voltaje y corriente de lo que solo la placa Arduino es capaz de hacer. En este ejemplo, aprenderemos cómo amplificar las capacidades PWM de Arduino.

Para esto se utiliza un diodo MOSFET. En esencia, este diodo actúa como un interruptor. Permite o interrumpe el flujo eléctrico entre su *fuentes* y los terminales de *drenaje*. Pero en lugar de un interruptor mecánico, presenta una tercera terminal llamada *compuerta*. Una corriente muy pequeña (<1mA) "abrirá" esta puerta y permitirá que la corriente fluya. Esto es muy conveniente, ya que podemos enviar la salida PWM de Arduino a esta puerta, creando así otro tren de pulsos PWM con el mismo ciclo de trabajo a través del MOSFET, que permite voltajes y corrientes que destruirían el Arduino.

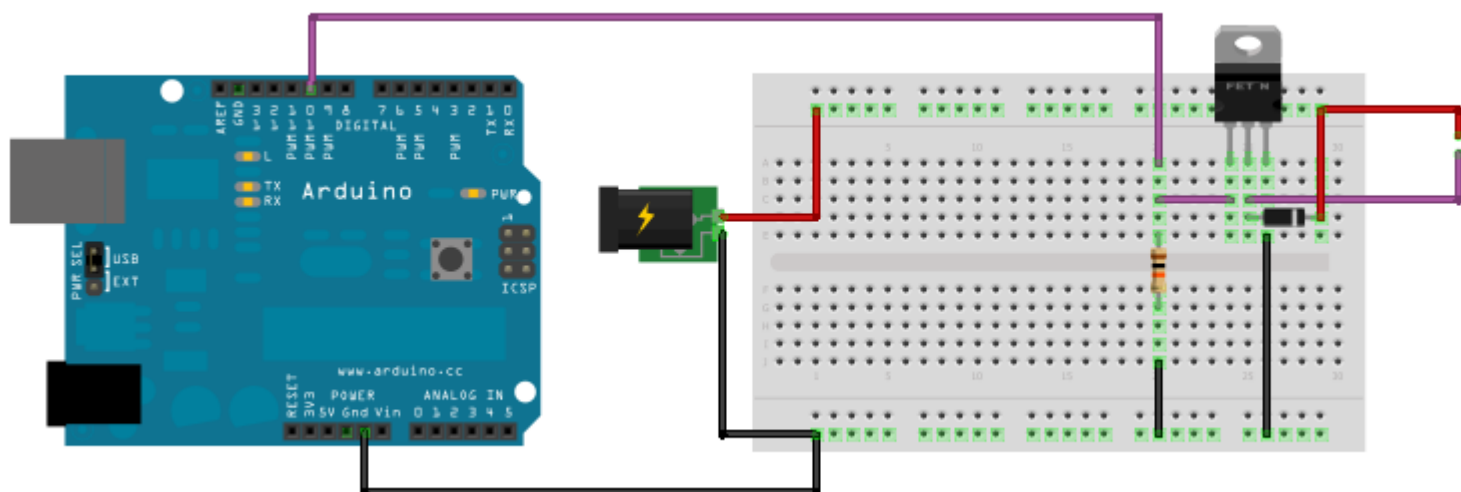
Lista de materiales: ¿qué necesitas para construir este ejemplo?

- Diodo MOSFET: por ejemplo, el popular [BUZ11](#)
- Diodo de protección para el motor: [Schottky SB320](#)
- Resistor: cualquier cosa 10K ~ 1M Ohm
- Motor: un motor pequeño típico (uno típico puede ser de 12 V)
- Una fuente de alimentación compatible con el motor que ha seleccionado.
- Un tablero
- Cables de colores!
- Un Arduino, pero eso ya lo sabías.

La construcción

¡Pon todo junto! Alimente los rieles de la placa de pruebas y coloque el diodo MOSFET en ella. Conecte el motor entre el riel positivo y el drenaje MOSFET. Conecte el diodo de protección de la misma manera: entre el drenaje MOSFET y el riel positivo. Conecte la fuente del MOSFET al riel de tierra común. Finalmente, conecte el pin PWM (estamos usando el pin 10 en este ejemplo) a la compuerta del MOSFET y también a la tierra común a través de la resistencia (necesitamos una corriente muy baja).

Aquí hay un ejemplo de cómo se ve esta compilación. Si prefieres un esquema [aquí hay](#) uno.



El código

Ahora podemos conectar el Arduino a una computadora, cargar el código y controlar el motor, mediante el envío de valores a través de la comunicación en serie. Recuerde que estos valores deben ser enteros entre 0 y 255. El código real de este ejemplo es muy simple. Se proporciona una explicación en cada línea.

```
int in = 0; // Variable to store the desired value
byte pinOut = 10; // PWM output pin

void setup() { // This executes once
  Serial.begin(9600); // Initialize serial port
  pinMode(pinOut, OUTPUT); // Prepare output pin
}

void loop() { // This loops continuously
  if(Serial.available()){ // Check if there's data
    in = Serial.read(); // Read said data into the variable "in"
    analogWrite(pinOut, in); // Pass the value of "in" to the pin
  }
}
```

¡Y eso es! Ahora puede usar las capacidades PWM de Arduino para controlar aplicaciones que requieren señales analógicas, incluso cuando los requisitos de energía superan los límites de la placa.

PWM con un TLC5940

El **TLC5940** es un elemento útil para tener cuando te quedas sin puertos PWM en el Arduino. Tiene 16 canales, cada uno controlable individualmente con 12 bits de resolución (0-4095). Una biblioteca existente está disponible en <http://playground.arduino.cc/Learning/TLC5940> . Es útil para controlar múltiples servos o LEDs RGB. Solo tenga en cuenta que los LED deben ser ánodo común para funcionar. Además, los chips son margaritables, lo que permite incluso más puertos PWM.

Ejemplo:

```
// Include the library
#include <Tlc5940.h>

void setup() {
  // Initialize
  Tlc.init();
  Tlc.clear();
}

unsigned int level = 0;
void loop() {
  // Set all 16 outputs to same value
  for (int i = 0; i < 16; i++) {
    Tlc.set(i, level);
  }
}
```

```
level = (level + 1) % 4096;
// Tell the library to send the values to the chip
Tlc.update();
delay(10);
}
```

Lea PWM - Modulación de ancho de pulso en línea:

<https://riptutorial.com/es/arduino/topic/1658/pwm---modulacion-de-ancho-de-pulso>

Capítulo 22: Salida de audio

Parámetros

Parámetro	Detalles
altavoz	Debe ser una salida a un altavoz analógico.

Examples

Salidas de notas básicas

```
#define NOTE_C4 262 //From pitches.h file defined in [Arduino Tone Tutorial][1]

int Key = 2;
int KeyVal = 0;

byte speaker = 12;

void setup()
{
  pinMode(Key, INPUT); //Declare our key (button) as input
  pinMode(speaker, OUTPUT);
}

void loop()
{
  KeyVal = digitalRead(Key);
  if (KeyVal == HIGH) {
    tone(speaker, NOTE_C4); //Sends middle C tone out through analog speaker
  } else {
    noTone(speaker); //Ceases tone emitting from analog speaker
  }

  delay(100);
}
```

[1]: <https://www.arduino.cc/en/Tutorial/toneMelody>

Lea Salida de audio en línea: <https://riptutorial.com/es/arduino/topic/2384/salida-de-audio>

Capítulo 23: Salida digital

Sintaxis

- `digitalWrite(pin, value)`

Examples

Escribir a pin

```
int ledPin = 13;           // LED connected to digital pin 13

void setup()
{
  pinMode(ledPin, OUTPUT); // sets the digital pin as output
}

void loop()
{
  digitalWrite(ledPin, HIGH); // sets the LED on
  delay(1000);                // waits for a second
  digitalWrite(ledPin, LOW);  // sets the LED off
  delay(1000);                // waits for a second
}
```

Ejemplo en [Arduino.cc](https://www.arduino.cc) .

Lea Salida digital en línea: <https://riptutorial.com/es/arduino/topic/2477/salida-digital>

Capítulo 24: Servo

Introducción

Un Servo es un sistema cerrado que contiene un motor y algunos circuitos de soporte. El eje de un servo se puede girar a un ángulo fijo dentro de un arco utilizando una señal de control. Si la señal de control se mantiene, entonces el servo mantendrá su ángulo. Los servos se pueden controlar fácilmente con la biblioteca Arduino `Servo.h`.

Sintaxis

- `#include <Servo.h>` // Incluir la biblioteca Servo
- `Servo.attach (pin)` // Adjuntar al servo en el pin. Devuelve un objeto Servo
- `Servo.write (grados)` // Grados para moverse a (0 - 180)
- `Servo.read ()` // Obtiene la rotación actual del servo

Examples

Mover el servo de ida y vuelta

```
#include <Servo.h>

Servo srv;

void setup() {
  srv.attach(9); // Attach to the servo on pin 9
}
```

Para usar un servo, primero debe llamar a la función `attach()`. Comienza a generar una señal PWM que controla un servo en un pin específico. En tableros que no sean Arduino Mega, el uso de la biblioteca Servo deshabilita la funcionalidad `analogWrite()` (PWM) en los pines 9 y 10, ya sea que haya un Servo en esos pines o no.

```
void loop() {
  Servo.write(90); // Move the servo to 90 degrees
  delay(1000); // Wait for it to move to it's new position
  Servo.write(0); // Move the servo to 0 degrees
  delay(1000); // Wait for it to move to it's new position
}
```

Tenga en cuenta que no está garantizado que el servo haya alcanzado la posición deseada, ni puede verificarlo desde el programa.

Lea Servo en línea: <https://riptutorial.com/es/arduino/topic/4920/servo>

Capítulo 25: Usando Arduino con Atmel Studio 7

Observaciones

Preparar

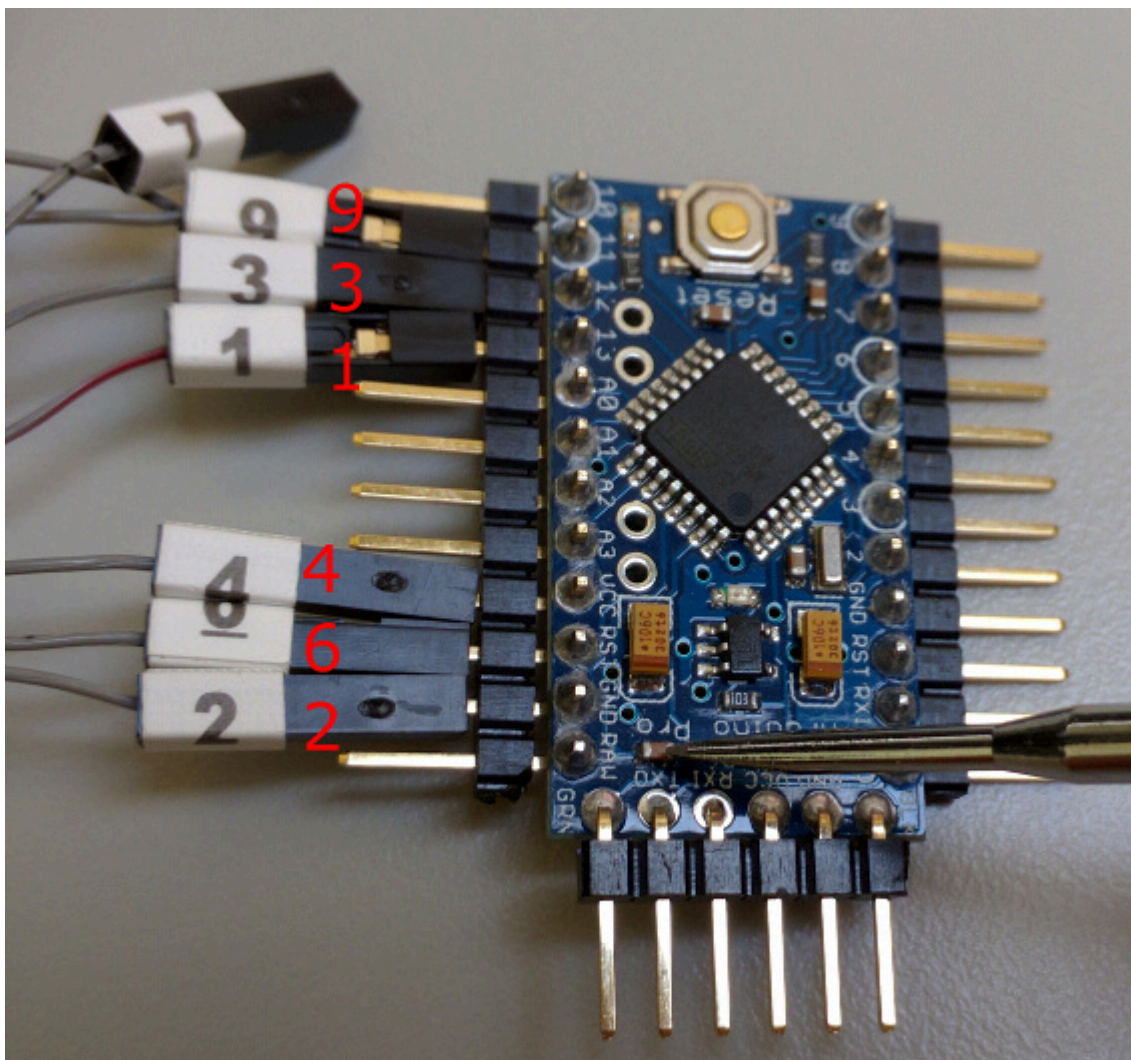
- Descargue e instale Atmel Studio 7 desde [aquí](#) .
- Compra un depurador. Usted puede arreglárselas con un programador de ISP, pero si desea capacidades de depuración, que es una de las grandes ventajas de usar Atmel Studio, querrá un depurador. Recomiendo el [ICE Atmel](#) , ya que proporciona capacidades de depuración para los arduinos basados en AVR (como el Uno, pro mini, etc.) y los Arduinos basados en ARM, como el cero y el vencimiento. Si tiene un presupuesto limitado, puede [obtenerlo](#) sin el estuche de plástico y tener cuidado de no darle una [sacudida](#) eléctrica.

Conexiones

- Para el Uno, use el [cable ICSP de 6 pines](#) . Enchufe un lado en el Uno como se muestra. Conecte el otro lado al puerto AVR del depurador.



Para el Arduino Pro Mini, use el [cable mini squid](#) como se muestra, conectando de nuevo el otro lado del puerto AVR del depurador.

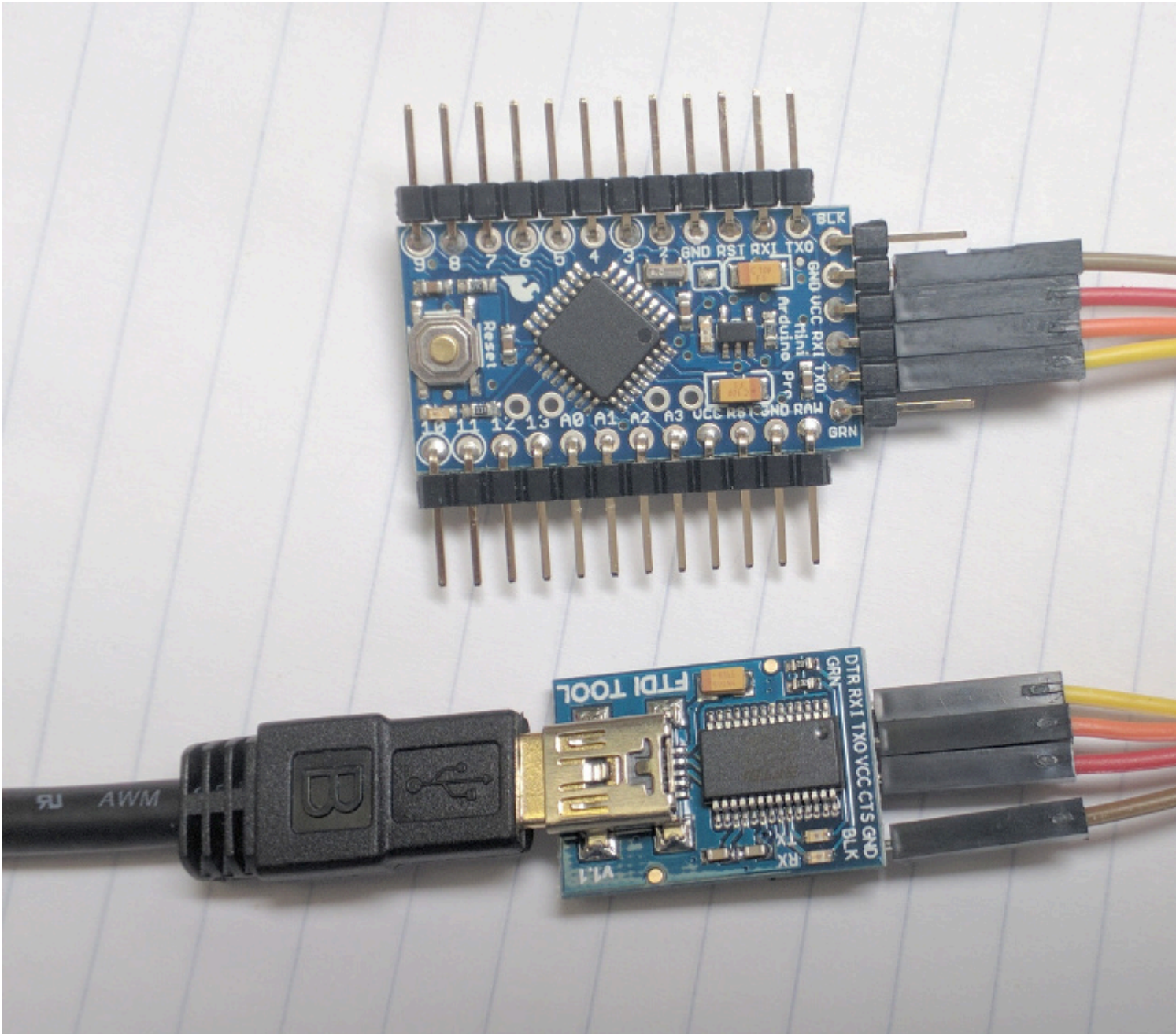


Consideraciones de depuración

Para la depuración con el Uno, deberá cortar la traza Restablecer-habilitar (siempre puede volver a soldar para usar con el IDE de Arduino):



Con Pro Mini, si pretende conectar el puerto serie a su computadora con una tarjeta FTDI, no conecte la línea DTR, ya que interferirá con la interfaz de Debug de cables en serie (SWD) de Atmel. Simplemente conecto la alimentación, tierra, Tx y Rx como se muestra a continuación. Rx y Tx en Arduino van a Tx y Rx, respectivamente, en la placa FTDI. Algunas tarjetas FTDI están etiquetadas de manera diferente, por lo que si el puerto serie no funciona, intercambie Rx y Tx.



Deberá proporcionar alimentación al Arduino por separado, ya que el depurador no lo activará. Esto se puede hacer en el Pro Mini a través de la placa FTDI como se muestra arriba, o con un cable USB o adaptador de CA en el Uno.

Configuración de software

Conecte Atmel ICE a su computadora, inicie Atmel Studio y ahora puede importar un proyecto Arduino existente.

En Atmel Studio, vaya a Archivo -> Nuevo -> Proyecto y seleccione "Crear proyecto desde el boceto de Arduino". Completa las opciones incluyendo menús y menús desplegables de dispositivos.

Vaya a Project -> yourProjectName Properties, haga clic en Tool, seleccione Atmel ICE en

debugger / programmer y debugWire en la interfaz. Ir a depuración -> Iniciar depuración y romper. Debería ver una advertencia y se le preguntará si desea configurar el fusible DWEN. Elija Aceptar, desenchufe el Arduino de la alimentación y vuelva a enchufarlo. Puede detener la depuración haciendo clic en el botón cuadrado rojo y comience haciendo clic en el botón triángulo verde. Para devolver el Arduino a un estado en el que se puede usar en el IDE de Arduino, mientras realiza la depuración, elija Depurar -> deshabilitar debugWIRE y cerrar.

Tenga en cuenta que cualquier función que agregue también debe incluir un prototipo de función (el bucle y la configuración no los necesitan). Puede ver los agregados de Atmel Studio en la parte superior del boceto si hubiera alguna función cuando importó su proyecto a Atmel Studio (consulte el ejemplo de código).

La compatibilidad con C ++ 11 está habilitada de forma predeterminada en Arduino 1.6.6 y superior. Esto proporciona más funciones de lenguaje C ++ y habilitarlo puede aumentar la compatibilidad con el sistema Arduino. Para habilitar C ++ 11 en Atmel Studio 7, haga clic derecho en su archivo de proyecto, seleccione propiedades, haga clic en ToolChain a la izquierda, haga clic en Varios bajo AVR / GNU C ++ Compiler y ponga `-std=c++11` en las otras banderas campo.

Incluir bibliotecas en tu croquis.

Copie el archivo de la biblioteca .cpp en `C:\Users\YourUserName\Documents\Atmel Studio\7.0\YourSolutionName\YourProjectName\ArduinoCore\src\core` , luego, en Atmel Studio, abra la ventana del Explorador de soluciones, haga clic derecho en Arduino Core / src / carpeta central, elija agregar -> elemento existente y elija el archivo que agregó. Haga lo mismo con el archivo de biblioteca .h y la carpeta YourProjectName / Dependancies.

Para agregar la ventana de terminal

Siempre puede tener el IDE de Android abierto y usar esa ventana Serial (solo seleccione el puerto serial correcto), sin embargo, para agregar una ventana Serial incorporada a Atmel Studio, vaya a Herramientas -> Extensiones y actualizaciones, haga clic en Descargas disponibles y busque Terminal de ventana o terminal para Atmel Studio e instalarlo. Una vez instalado, vaya a Ver -> Ventana de terminal.

Beneficios

La programación de Arduino con un IDE moderado como Atmel Studio 7 le brinda numerosas ventajas sobre el IDE de Arduino, que incluyen depuración, autocompletado, salto a definición y declaración, navegación hacia adelante / hacia atrás, marcadores y opciones de refactorización, entre otros.

Puede configurar enlaces de teclas yendo a Herramientas -> Opciones -> Entorno -> Teclado. Algunos que realmente aceleran el desarrollo son:

- Edit.CommentSelection, Edit.UncommentSelection
- View.NavigateForward, View.NavigateBackward
- Edit.MoveSelectedLinesUp, Edit.MoveSelectedLinesDown
- Edit.GoToDefinition

Examples

Ejemplo de boceto importado de Atmel Studio 7

Este es un ejemplo de cómo se ve un simple boceto de Arduino después de ser importado a Atmel Studio. Atmel Studio agregó las secciones generadas automáticamente en la parte superior. El resto es idéntico al código original de Arduino. Si expande el proyecto ArduinoCore que se creó y busca en la carpeta src -> core, encontrará `main.cpp`, el punto de entrada para el programa. Allí puede ver la llamada a la función de configuración de Arduino y un interminable bucle que llama a la función de bucle de Arduino una y otra vez.

```
/* Beginning of Auto generated code by Atmel studio */
#include <Arduino.h>
/* End of auto generated code by Atmel studio */

// Beginning of Auto generated function prototypes by Atmel Studio
void printA();
// End of Auto generated function prototypes by Atmel Studio

void setup() {
  Serial.begin(9600);
}

void loop() {
  printA();
}

void printA() {
  Serial.println("A");
}
```

Lea Usando Arduino con Atmel Studio 7 en línea:

<https://riptutorial.com/es/arduino/topic/2567/usando-arduino-con-atmel-studio-7>

Capítulo 26: Variables y tipos de datos

Examples

Crear variable

Para crear una variable:

```
variableType variableName;
```

Por ejemplo:

```
int a;
```

Para crear una variable e inicializarla:

```
variableType variableName = initialValue;
```

Por ejemplo:

```
int a = 2;
```

Asignar valor a una variable

Si tiene una variable declarada anteriormente, puede asignarle algún valor:

Por ejemplo:

```
int a; // declared previously  
a = 2;
```

O cambiar el valor:

```
int a = 3; // initalized previously  
a = 2;
```

Tipos de variables

- `char` : valor de carácter de 1 byte firmado
- `byte` : entero de 8 bits sin signo
- `int` : con signo de 16 bits (en tarjetas basadas en ATMEGA) o de 32 bits (en Arduino Due) entero
- `unsigned int` : sin signo de 16 bits (en tableros basados en ATMEGA) o de 32 bits (en Arduino Due) entero
- `long` : entero de 32 bits con signo

- `unsigned long` : sin signo entero de 32 bits
- `float` : número de coma flotante de 4 bytes
- `double` : número de coma flotante de 4 bytes (en placas basadas en ATMEGA) o de 8 bytes (en Arduino Due)

Ejemplos:

```
char a = 'A';  
char a = 65;  
  
byte b = B10010;  
  
int c = 2;  
  
unsigned int d = 3;  
  
long e = 186000L;  
  
unsigned long f = millis(); // as an example  
  
float g = 1.117;  
  
double h = 1.117;
```

Lea Variables y tipos de datos en línea: <https://riptutorial.com/es/arduino/topic/2565/variables-y-tipos-de-datos>

Creditos

S. No	Capítulos	Contributors
1	Empezando con el arduino	Abhishek Jain , Christoph , Community , Danny_ds , Doruk , geek1011 , gmuraleekrishna , H. Pauwelyn , jleung513 , Martin Carney , Mizole Ni , Shef , uruloke , Wolfgang
2	Almacenamiento de datos	Danny_ds , robert
3	Arduino IDE	geek1011 , Jeremy , jleung513 , sohnryang , uruloke
4	Biblioteca de cristal líquido	Morgoth
5	Bibliotecas	Oscar Lundberg
6	Bucles	datafiddler , Martin Carney , MikeCAT
7	Cómo almacenar variables en EEPROM y usarlas para almacenamiento permanente	AZ Vcience , Chris Combs , Danny_ds , Jeremy , Peter Mortensen , RamenChef
8	Cómo Python se integra con Arduino Uno	Danny_ds , Peter Mortensen , Stark Nguyen
9	Comunicación bluetooth	Girish , Martin Carney
10	Comunicación I2C	Asaf
11	Comunicación MIDI	Rich Maes
12	Comunicación serial	blainedwards8 , Danny_ds , geek1011 , Leah , Martin Carney , MikeS159 , Morgoth , Nufail Achath , Peter Mortensen , uruloke
13	Comunicación SPI	Christoph
14	Entradas analógicas	Jake Lites , MikeS159 , Ouss4 , uruloke
15	Entradas digitales	Martin Carney , uruloke

16	Funciones	datafiddler , Leah , MikeCAT
17	Gestión del tiempo	Christoph , Rei
18	Interrupciones	DavidJ , Martin Carney
19	Números al azar	Danny_ds , Javier Rizzo Aguirre , MikeCAT
20	Pines de hardware	Jeremy , Martin Carney
21	PWM - Modulación de ancho de pulso	Danny_ds , Johnny Mopp , JorgeGT , Martin Carney
22	Salida de audio	Jake Lites , MikeCAT
23	Salida digital	uruloke
24	Servo	geek1011 , mactro , Morgoth
25	Usando Arduino con Atmel Studio 7	Danny_ds , Nate
26	Variables y tipos de datos	Leah , MikeCAT