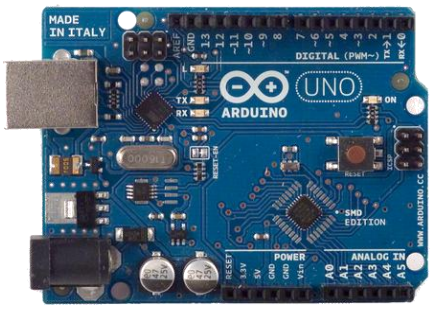
1

Manejo y Aplicaciones del Bus I2C de Arduino

Ver. 1.0



José Manuel Ruiz Gutiérrez

Serie Monografías: Aplicaciones de ARDUINO

INDICE

Agosto de 2012 Versión de Documento: Versión. (Beta en Revisión)
Blog de referencia: http://josemanuelruizgutierrez.blogspot.com

Introducción

Con este nuevo manual se inicia una serie de Monografías dedicadas a Arduino en la que se abordarán de manera monográfica temas que revistan especial interés en el campo de las aplicaciones de la Plataforma Open Hardware Arduino. En esta primera entrega se pretende recoger la información más relevante acerca del protocolo del bus I2C que implementa Arduino y de la librería que acompaña al IDE Arduino (wire) con la que podemos realizar la conexión de dos o más tarjetas de Arduino y/o dispositivos que sean compatibles con el bus I2C.

Ejemplos.

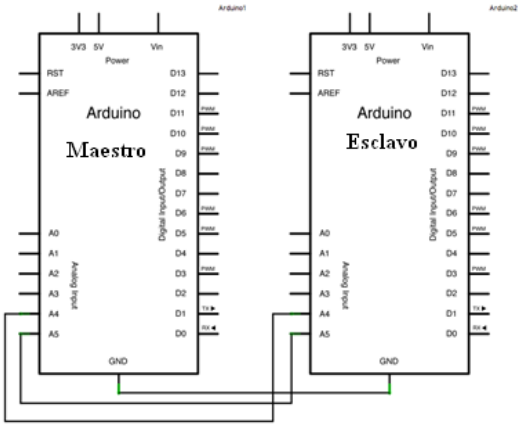
Ejemplo 1

Maestro escribe en Esclavo

En este ejemplo vemos como se comunican dos Arduinos a través del bus I2C

Un Arduino actúa como Maestro y el otro como Esclavo.

El maestro envía datos al esclavo, en nuestro ejemplo envía el valor de “x” incrementándose y el esclavo lo escribe en el puerto serie pudiendo visualizar los envíos en la ventana de monitorización del IDE Arduino.



En la figura vemos los conexionados

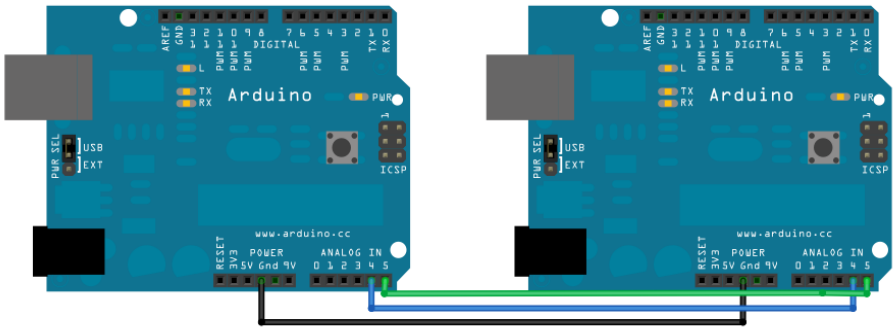


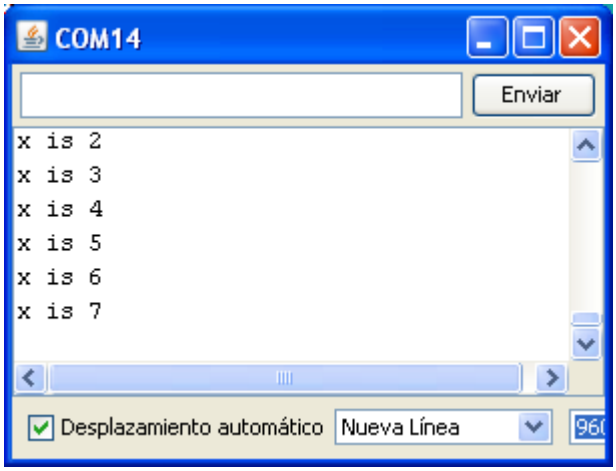
Tabla de conexionado

Maestro	Esclavo
GND	GND
A4	A4
A5	A5

Modo de operación:

Se cargan cada uno de los sketch en la tarjeta Arduino que actúe de Maestro y la de Esclavo y después se alimenta la tarjeta maestro con alimentación independiente (batería) y conectamos el puerto USB de la tarjeta Esclavo y desde el IDE Arduino seleccionamos la opción de monitorizar el puerto y veremos cómo se escriben los valores.

Si desconectamos uno de los terminales de unión entre las tarjetas deja de realizarse el envío de datos.



Sketch Maestro

```
#include <Wire.h>
//
byte x = 0;
void setup()
{
 Wire.begin();
}
//
void loop()
{
 Wire.beginTransmission(4); // transmit to device #4
 Wire.write("x is "); // sends five bytes
 Wire.write(x); // sends one byte
```

```
 Wire.endTransmission(); // stop transmitting
 x++;
 delay(500);
}
```

Sketch Esclavo

```
#include <Wire.h>
//
void setup()
{
  Wire.begin(4); // join I2C bus with address #4
  Wire.onReceive(receiveEvent); // register event
  Serial.begin(9600); // start serial for output
}
//
void loop()
{
}
//
void receiveEvent(int howMany)
{
   while(1 < Wire.available()) // loop through all but the last
   {
      char c = Wire.read(); // receive byte as a character
      Serial.print(c); // print the character
   }
   int x = Wire.read(); // receive byte as an integer
   Serial.println(x); // print the integer
}
```

Ejemplo 2

Maestro recibe del Esclavo

En este ejemplo vemos como se comunican dos Arduinos a través del bus I2C

Un Arduino actúa como Maestro y el otro como Esclavo.

El maestro envía datos al esclavo, en nuestro ejemplo envía el valor de “x” incrementándose y el esclavo lo escribe en el puerto serie pudiendo visualizar los envíos en la ventana de monitorización del IDE Arduino.

En la figura vemos los conexionados

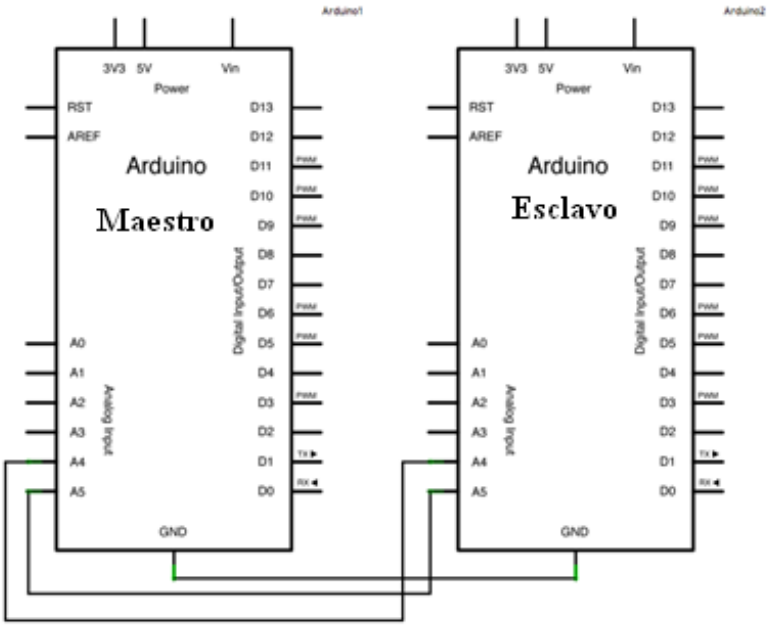


Tabla de conexionado

Maestro	Esclavo
GND	GND
A4	A4
A5	A5

Sketch Maestro

```
// Wire Master Reader
// by Nicholas Zambetti <http://www.zambetti.com>
// Demonstrates use of the Wire library
// Reads data from an I2C/TWI slave device
// Refer to the "Wire Slave Sender" example for use with this
// Created 29 March 2006
// This example code is in the public domain.


#include <Wire.h>
void setup()
{
  Wire.begin();        // join i2c bus (address optional for master)
  Serial.begin(9600);  // start serial for output
}

void loop()
{
  Wire.requestFrom(2, 6);    // request 6 bytes from slave device #2
  while(Wire.available())    // slave may send less than requested
  {
    char c = Wire.read(); // receive a byte as character
    Serial.print(c);         // print the character
  }

  delay(500);

}
```

Sketch Esclavo

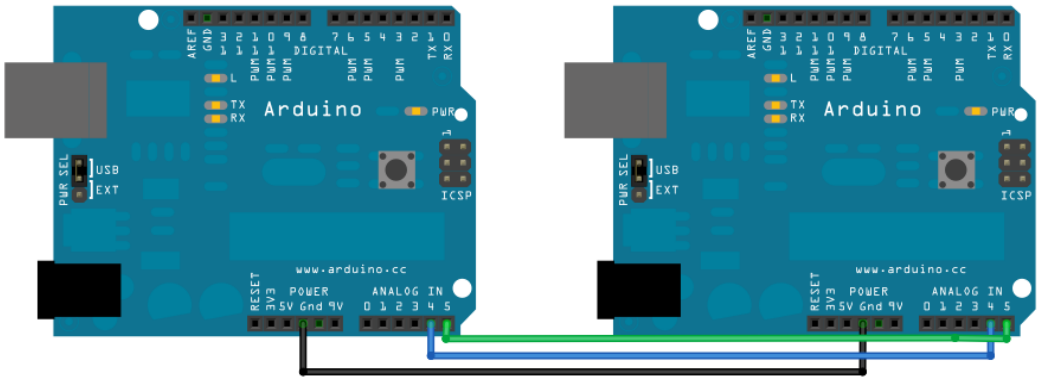
```
// Wire Slave Sender
// by Nicholas Zambetti <http://www.zambetti.com>
// Demonstrates use of the Wire library
// Sends data as an I2C/TWI slave device
// Refer to the "Wire Master Reader" example for use with this
// Created 29 March 2006
// This example code is in the public domain.


#include <Wire.h>
void setup()
{
  Wire.begin(2);                // join i2c bus with address #2
  Wire.onRequest(requestEvent); // register event
}
```

```
void loop()
{
  delay(100);
}
// function that executes whenever data is requested by master
// this function is registered as an event, see setup()
void requestEvent()
{
  Wire.write("hello "); // respond with message of 6 bytes
                        // as expected by master
}
```

Ejemplo 3

Gobierno de un Led

Gobierno de un Led conectado en la tarjeta Esclavo desde un pulsador conectado en la tarjeta maestro.



Sketchs

// Maestro modo receptor

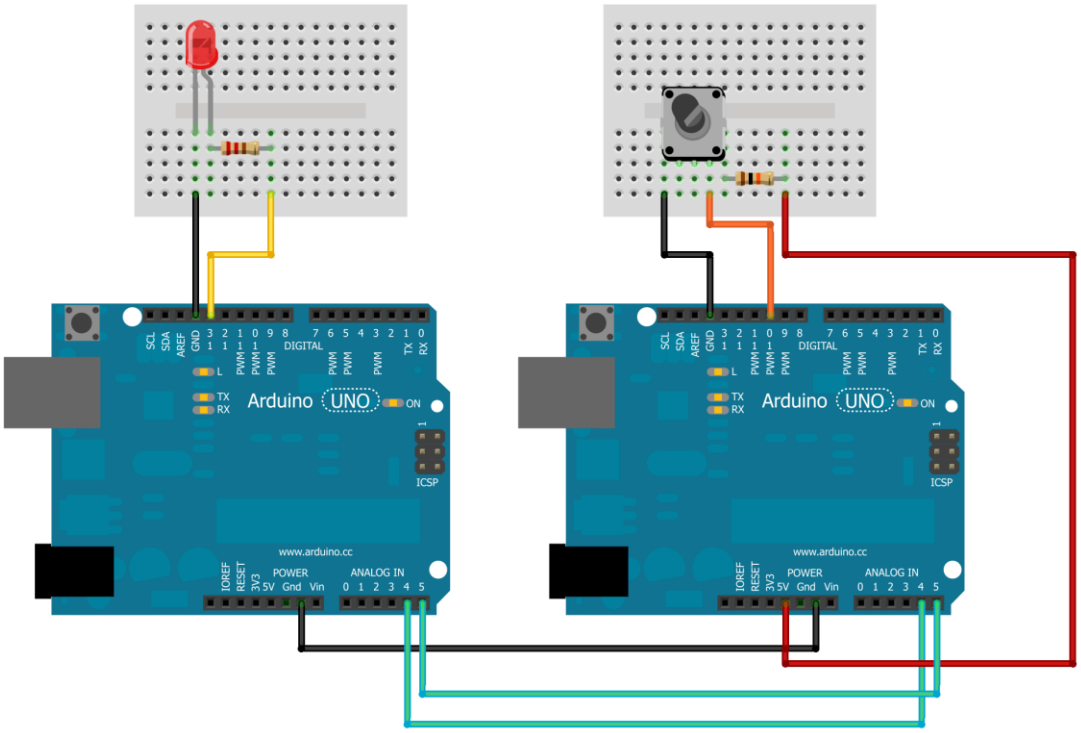
```
#include <Wire.h>
int led = 13;
void setup()
{
  pinMode(led, OUTPUT);
  Wire.begin();        // join i2c bus (address optional for master)
  Serial.begin(9600);  // start serial for output
}
void loop()
{
  Wire.requestFrom(2, 1);    // request 6 bytes from slave device #2

  while(Wire.available())    // slave may send less than requested
  {
    char c = Wire.read(); // receive a byte as character

    if (c=='s')  {
      digitalWrite(led, HIGH);
       Serial.println("si");
```

```
   }
   if (c=='n') {
     digitalWrite(led, LOW);
      Serial.println("no");
   }
  }

  delay(100);
}
```

Sketch Esclavo en modo emisor

```
// Esclavo emisor
#include <Wire.h>
const int buttonPin = 10;
int buttonState = 0;

void setup()
{
  pinMode(buttonPin, INPUT);
  Wire.begin(2);                // join i2c bus with address #2
  Wire.onRequest(requestEvent); // register event
}

void loop()
{
  delay(100);
}

// function that executes whenever data is requested by master
// this function is registered as an event, see setup()
void requestEvent()
{
  buttonState = digitalRead(buttonPin);
  if (buttonState == HIGH) {
    // turn LED on:
    Wire.write("s");
  }
  else {
    // turn LED off:
    Wire.write("n");
  }
```

```
  Wire.write("hello "); // respond with message of 6 bytes
                        // as expected by master
}
```



Otra solución actuando el Maestro como emisor y el Esclavo como receptor

// Esclavo Receptor

```
#include <Wire.h>

int led = 13;
void setup()
{
  pinMode(led, OUTPUT);
  Wire.begin(2);                // Define direccion i2c del bus #2
  Wire.onReceive(receiveEvent); // registra evento
  Serial.begin(9600);           // Inicia el puerto serie
}

void loop()
{
  delay(100);
}

// función que se ejecuta cada vez que se reciben los datos del maestro
// Esta función está registrada como un evento, ver setup ()
void receiveEvent(int howMany)
{
  while(1 < Wire.available()) // recorre todos mientas se prodcue la detecion de dato en el bus
  {
    char c = Wire.read(); // recibe como un caracter
    Serial.print(c);         // imprime el caracter
   }
  char x = Wire.read();    // recibe un byte como un entero
  Serial.println(x);         // imprime el entero

  if (x=='s')  {            // Si el byte es s se activa la salida PIN 13
      digitalWrite(led, HIGH);
       Serial.println("si");
    }
    if (x=='n') {
      digitalWrite(led, LOW); // Si el byte es s se desactiva la salida PIN 13
       Serial.println("no");
    }
 }
```

// Maestro emisor

```
#include <Wire.h>
const int buttonPin = 10;     // numero de PIN del Boton
int buttonState = 0;         // Varoable que recoge el valor del esatdo del botón


void setup()
{
  Serial.begin(9600);
  pinMode(buttonPin, INPUT);
  Wire.begin(); // configura el bus i2c(direccion opcional para el maestro)
}

void loop()
{
  Wire.beginTransmission(2); // transmite al elemento #2

  buttonState= digitalRead(buttonPin);
    if (buttonState == HIGH) {
      Wire.write('s');        // envia un byte
      Serial.println("s");
      Wire.endTransmission();    // detoene la transmisión
  }
  if (buttonState==LOW){
       Wire.write('n');        // envia un byte
        Serial.println("n");
        Wire.endTransmission();    // detiene la trasmisión
  }

    //
    delay(500);
    //Wire.endTransmission();    // aqui podriamos poner el fin de transmisión
                              //tambien y quitar los anteriores

}
```

Ejemplo 4

Gobierno de dos salidas con dos entradas

Se trata de gobernar dos diodos led, Rojo, Verde, que se encuentran en una unidad i2c Esclava de dirección 4, mediante dos pulsadores BotonA y BotonB que se encuentran en la unidad maestro de acuerdo a la siguiente tabla de funcionamiento.

Botón A	Botón B	Led Rojo	Led Verde	Texto que envía
ON	ON	APAGADO	APAGADO	nada
ON	OFF	ENCENDIDO	APAGADO	rojo
OFF	ON	APAGADO	ENCENDIDO	verde
OFF	OFF	APAGADO	APAGADO	nada

Designación de E/S

En el lado Maestro		En el lado Esclavo	
BotonA	PIN 2	LedRojo	PIN 2
BotonB	PIN 3	LedVerde	PIN3

Códigos

Codigo del Maestro (actúa como emisor de datos)

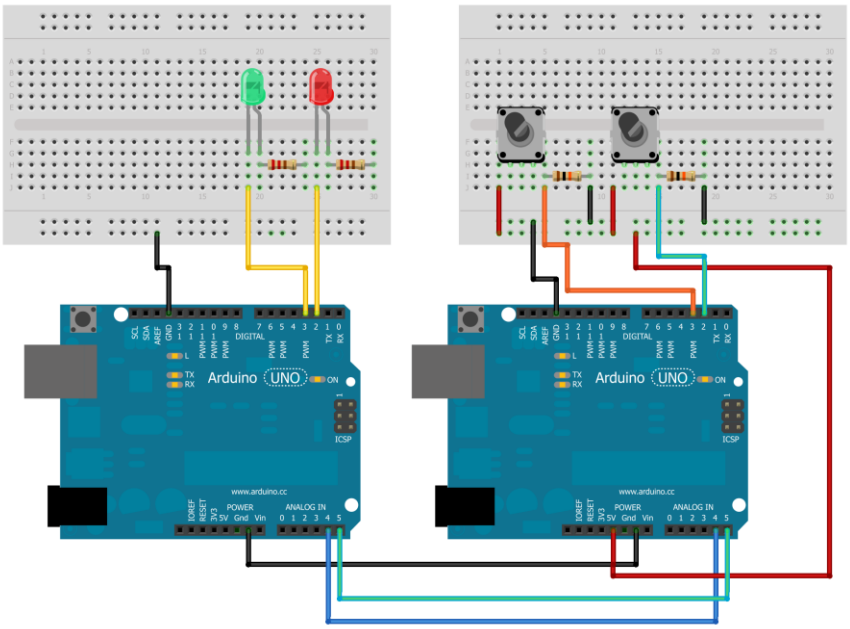
```
// Maestro actuando como emisor de datos

#include <Wire.h>

int pinBotonA = 2;
int pinBotonB = 3;

int rojo = 1;
int verde = 2;


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

  pinMode( pinBotonA , INPUT );
```

```
  pinMode( pinBotonB , INPUT );
}//setup

void loop(){

  Wire.beginTransmission(4);

  int valorBotonA = digitalRead ( pinBotonA );
  int valorBotonB = digitalRead ( pinBotonB );

  if ( (valorBotonA == HIGH) && (valorBotonB == LOW) ){
    Wire.write(rojo);
    Serial.println("Envio -> rojo");
  }
  if ( (valorBotonA == LOW) && (valorBotonB == HIGH) ){
    Wire.write(verde);
    Serial.println("Envio -> verde");
  }


  Wire.endTransmission();

  delay(500);
}//loop
```

Código del Esclavo (actúa como receptor de datos)

```
// Esclavo actua como receptor de datos
#include <Wire.h>



int pinLedRojo = 2;
int pinLedVerde = 3;

int rojo = 1;
int verde = 2;

void setup() {
  Wire.begin(4);
  Wire.onReceive(receiveEvent);
  Serial.begin(9600);
```

```
  pinMode( pinLedRojo , OUTPUT );
  pinMode( pinLedVerde , OUTPUT );
}

void loop(){}

void receiveEvent(int howMany) {
  int recibo = Wire.read();
  Serial.print("Recibo : ");
  Serial.println(recibo);
  Serial.print("howMany : ");
  Serial.println(howMany);

  if( recibo == 1 ){
    digitalWrite( pinLedRojo , HIGH );
    digitalWrite( pinLedVerde , LOW );
  }
  if ( recibo == 2 ){
    digitalWrite( pinLedRojo , LOW );
    digitalWrite( pinLedVerde , HIGH );
  }

  delay(500);

 }//receiveEvent
```



ANEXO I

Descripción de las funciones de la Librería Wire

(Referencia http://arduino.cc/es/Reference/Wire)

La librería viene incluida en el IDE Arduino. Esta librería te permite comunicar con dispositivos I2C / TWI. En la mayoría de las placas Arduino, SDA (línea de datos) está en el pin analógico 4, y SCL (línea de reloj) está en el pin analógico 5. En Arduino Mega, SDA esta en el pin digital 20 y SCL en el 21.

Funciones

- begin()
- begin(address)
- requestFrom(address, count)
- beginTransmission(address)
- endTransmission()
- send()
- byte available()
- byte receive()
- onReceive(handler)
- onRequest(handler)

Nota

Hay dos versiones de direccionamiento en I2C 7 y 8 bits. 7 bits identifican al dispositivo, y el octavo bit determina si se está leyendo o escribiendo. La librería Wire usa 7 bits de direccionamiento siempre. Si tienes un datasheet o un código que usa 8 bits de direccionamiento, tendrás que renunciar al bit más bajo (además debes desplazar el valor un bit a la derecha), cediendo una dirección entre 0 y 127.

Wire.begin()

Wire.begin(address)

Descripción

Inicializa la librería Wire y configura el bus I2C como maestro o esclavo.

Parámetros

address: La dirección de 7 bits de esclavo (opcional); si no se especifica, se configura como maestro.

Devuelve

Nada

Wire.requestFrom(address, quantity)

Descripción

Solicita bytes de otro dispositivo. Los bytes pueden ser recibidos con las funciones available() y receive().

Parámetros

address: la dirección de 7 bits del dispositivo al que pedir los bytes

quantity: el número de bytes a pedir

Devuelve

Nada

Wire.beginTransmission(address)

Descripcion

Comienza una transmisión a un dispositivo I2C esclavo con la dirección dada (address). Posteriormente, prepara los bytes a transmitir con la función send() y los transmite llamando a la función endTransmission().

Parámetros

address: la dirección de 7 bits del dispositivo a transmitir

Devuelve

Nada

Wire.endTransmission()

Descripción

Finaliza una transmisión a un esclavo que fue empezada por beginTransmission() y realmente transmite los bytes que fueron preparados por send().

Parámetros

Nada

Returns

Nada

write()

Description

Writes data from a slave device in response to a request from a master, or queues bytes for transmission from a master to slave device (in-between calls to beginTransmission() and endTransmission()).

Syntax

Wire.write(value)
Wire.write(string)
Wire.write(data, length)

Parameters

value: a value to send as a single byte

string: a string to send as a series of bytes

data: an array of data to send as bytes

length: the number of bytes to transmit

Returns

byte
write() will return the number of bytes written, though reading that number is optional

Example

```
#include <Wire.h>

byte val = 0;

void setup()
{
  Wire.begin(); // join i2c bus
}

void loop()
{
  Wire.beginTransmission(44); // transmit to device #44 (0x2c)
                              // device address is specified in datasheet
  Wire.write(val);            // sends value byte
  Wire.endTransmission();     // stop transmitting

  val++;        // increment value
  if(val == 64) // if reached 64th position (max)
  {
    val = 0;    // start over from lowest value
  }
  delay(500);
}
```

[Get Code]

See also

- WireRead()
- WireBeginTransmission()
- WireEndTransmission()
- Serial.write()

Wire.send(value)

Wire.send(string)

Wire.send(data, quantity)

Descripción

Envía datos desde un esclavo en respuesta a una petición de un maestro, o prepara los bytes para transmitir de un maestro a un esclavo (entre llamadas a beginTransmission() y endTransmission()).

Parámetros

value: un byte para envíar (*byte*)

string: una cadena para envíar (*char* *)

data: un vector de datos para envíar (*byte* *)

quantity: el número de bytes de datos para transmitir (*byte*)

Devuelve

Nada

read()

Description

Reads a byte that was transmitted from a slave device to a master after a call to requestFrom() or was transmitted from a master to a slave. read() inherits from the Stream utility class.

Syntax

Wire.read()

Parameters

none

Returns

The next byte received

Example

```
#include <Wire.h>

void setup()
{
  Wire.begin();        // join i2c bus (address optional for master)
  Serial.begin(9600);  // start serial for output
}

void loop()
{
  Wire.requestFrom(2, 6);    // request 6 bytes from slave device #2
```

```
  while(Wire.available())    // slave may send less than requested
  {
    char c = Wire.read();    // receive a byte as character
    Serial.print(c);         // print the character
  }

  delay(500);
}
```

Wire.available()

Descripción

Devuelve el numero de bytes disponibles para recuperar con receive(). Debería ser llamada por un maestro después de llamar a requestFrom() o por un esclavo dentro de la función a ejecutar por onReceive()

Parámetros

Nada

Devuelve

El número de bytes disponibles para leer

byte Wire.receive()

Descripción

Recupera un byte que fue transmitido desde un dispositivo esclavo a un maestro después de una llamada a requestFrom o que fue transmitido de un maestro a un esclavo.

Parámetros

Nada

Devuelve

El byte recibido.

byte Wire.receive()

Descripción

Recupera un byte que fue transmitido desde un dispositivo esclavo a un maestro después de una llamada a requestFrom o que fue transmitido de un maestro a un esclavo.

Parámetros

Nada

Devuelve

El byte recibido.

Wire.onReceive(handler)

Descripción

Registra una función que será lllamada cuando un dispositivo esclavo reciba una transmisión desde un maestro.

Parametros

handler: la función que será llamada cuando el esclavo recibe datos; esta función debería coger un único parámetro entero (el número de bytes recibidos desde un maestro) y no devolver nada, por ejemplo: `void myHandler(int numBytes)`

Devuelve

Nada

Wire.onRequest(handler)

Descripción

Registra una función que será llamada por el dispositivo esclavo cuando un maestro solicite datos.

Parámetros

handler: la función a ser llamada, no coge parámetros y no devuelve nada, por ejemplo: `void myHandler()`

Devuelve

Nada

ANEXO 2

Protocolo I2C / TWI

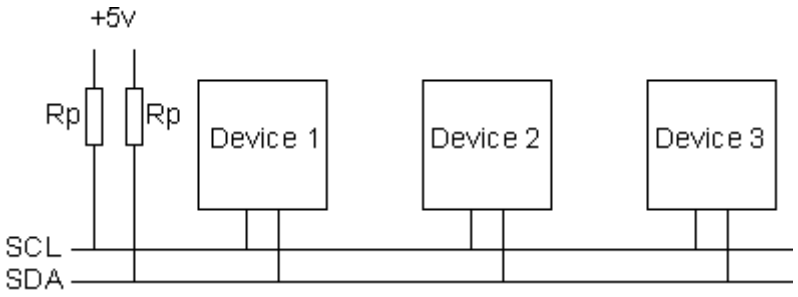
Fuentes de información: http://jmnlab.com/i2c/i2c.html
http://www.hispavila.com/3ds/atmega/intro_i2c.html

QUÉ ES EL I²C-BUS.

El **I²C bus**, no tan sólo son dos cables, usados para realizar una transmisión bidireccional de datos entre distintos sistemas gobernados por microcontroladores de forma eficaz. Veremos, cómo podemos controlar un importante número de dispositivos con nuestro Arduino, aunque, no es fácil de dominar. Se trata de, un bus bidireccional que utiliza dos líneas, una de datos serie (SDA) y otra de reloj serie (SCL), que requiere resistencias de polarización a positivo (RPA). SCL es la línea de reloj, se utiliza para sincronizar todos los datos SDA de las transferencias durante **I²C bus**. SDA es la línea de datos.

Las líneas SCL y SDA están conectadas a todos los dispositivos en el **I²C bus**. Ambas líneas SCL y SDA son del tipo drenador abierto asociados a un transistor de efecto de campo (o FET), es decir, un estado similar al de colector abierto. Esto significa que el chip puede manejar su salida a BAJO, pero no puede manejar a ALTO. Para que la línea pueda ir a ALTO, se deben proporcionar resistencias de polarización a 5V. Necesita de una resistencia de la línea SCL a la línea de 5V y otra de la línea SDA a la línea de 5V. Sólo necesita un conjunto de resistencias de RPA (pull-up) para todo el **I²C bus**, no son necesarias para cada dispositivo. La alimentación del sistema, debe tener una masa común, también puede haber una alimentación compartida que, se distribuye entre los distintos dispositivos.

Los dispositivos en el I²C bus son maestros o esclavos. **El maestro, es siempre el dispositivo que maneja la línea de reloj SCL**. Los esclavos, son los dispositivos que responden al maestro. Un esclavo no puede iniciar una transferencia a través del I²C bus, sólo un maestro puede hacer esa función. Generalmente son, varios esclavos en el I²C bus, sin embargo, normalmente hay un solo maestro. Es posible tener varios maestros, pero es inusual y no se comentará aquí. **Los esclavos, nunca inician una transferencia**. Tanto el maestro, como el esclavo puede transferir datos a través del I²C bus, pero la transferencia siempre es controlada por el maestro.



Todas las direcciones **I²C bus** son de 7 bits o 10 bits. Esto significa que, se pueden tener hasta 128 dispositivos en el bus I²C, ya que un número de 7bit puede estar de 0 a 127. El I²C tiene un diseño de espacio de referencia de 7 bits de direcciones, reservado con 16 direcciones, de modo que finalmente, pueden comunicarse en el mismo bus un máximo de 112 nodos. El número máximo de nodos está limitado por el espacio de direcciones y también por la capacidad total de los buses de 400 pF, lo que restringe la práctica de comunicación, a distancias de unos pocos metros.

Cuando se envía la dirección de 7 bits, siempre seguimos enviando 8 bits. El **bit** extra (bit 8) se usa para informar al esclavo si el maestro está escribiendo o leyendo de él. Si el **bit** 8º es 0, el maestro está escribiendo al esclavo. Si el **bit** 8º es 1, el maestro esta en la lectura del esclavo. Existen disposiciones en la norma de I²C para entornos multi-master, así como de 10 bits frente a la más simple y la más comúnmente usada, es la configuración de un solo maestro, de 7 bits de direccionamiento.

Yo se que, la localización de la dirección de 7 bits en la parte superior del byte, es una fuente de confusión para los recién llegados, pero intentaré hacer un esfuerzo para dejar lo más claro posible este punto. No debería haber ningún problema en que, los tres bits A0, A1 y A2 correspondientes a los pines 1, 2 y 3 seleccionan la dirección del dispositivo, del P0 al P7 son los puertos de E/S e INT, es una salida de interrupción que no lo usaremos. De modo que, para poner en servicio un PDF8574, son necesarias dos cosas, la dirección del dispositivo y un byte de datos para el pin de salida que se necesita.

BYTE	BIT							
	7 (MSB)	6	5	4	3	2	1	0 (LSB)
I^2C slave address	L	H	L	L	A2	A1	A0	R/$\overline{W}$
I/O data bus	P7	P6	P5	P4	P3	P2	P1	P0

De modo que, si ponemos los pines A0 ~ A2 a masa o GND, la dirección de nuestro dispositivo en el sistema binario será 0100000, o 0x20 en formato hexadecimal. Y, para establecer los pines de salida, de nuevo, haremos lo mismo, por ejemplo, si queremos poner todos a Alto (H), enviamos 0 en binario que, en hexadecimal es 0, o sea que, para poner los cuatro primeros a **H** y los segundos cuatro a **L**. Se utiliza 00001111 en binario que, es 0x0F en hexadecimal.

¿Se ha dado cuenta? Porqué razón estos datos parecen estar al revés de lo que se ha comentado. El motivo es, que las E/S del PCH8574 son sumideros de corriente, esto quiere decir que, las corrientes circulan desde los +5V, a través de las cargas hacia los

pines de E/S. Además, en la instrucción, no hay que olvidar el **bit0** (R/W), que le indica al bus I2C según su valor (para leer 0 y para escribir 1), la función del maestro.

¿Qué es TWI?
Aunque las patentes de I^2C ya han expirado, algunos vendedores utilizan los nombres TWI y TWSI para referirse a I^2C. Es exactamente lo mismo.

Características del protocolo:

- Velocidad standard de **100Kbit/s** (100kbaudios). Se puede cambiar al modo de alta velocidad (400Kbit/s)
- Configuración maestro/esclavo. La direccion del esclavo se configura con software
- Solo se necesitan **dos** lineas:
    - **SDA (Serial Data Line):** Linea de datos.
    - **SCL/CLK (Serial Clock Line):** Linea de reloj, será el que marque el tiempo de RW (Lectura/Escritura)
    - *Nota: Suponemos que todos los dispositivos tienen masa común, si no fuera así hay que uncluir una linea de masa.*
- Los comunicación siempre tiene la estructura siguiente:
    - Transmisor: Byte de datos (8 Bits)
    - Receptor: Bit llamado ACK de confirmación.

¿Cómo se realizan las conexiones?
SDA y SCL van a su pin correspondiente en cada dispositivo, de manera que todos quedan en paralelo.

Las lineas SDA y SCL estan independientemente conectadas a dos resistores Pull-Up que se encargaran de que el valor logico siempre sea alto a no ser que un dispositivo lo ponga a valor lógico bajo.

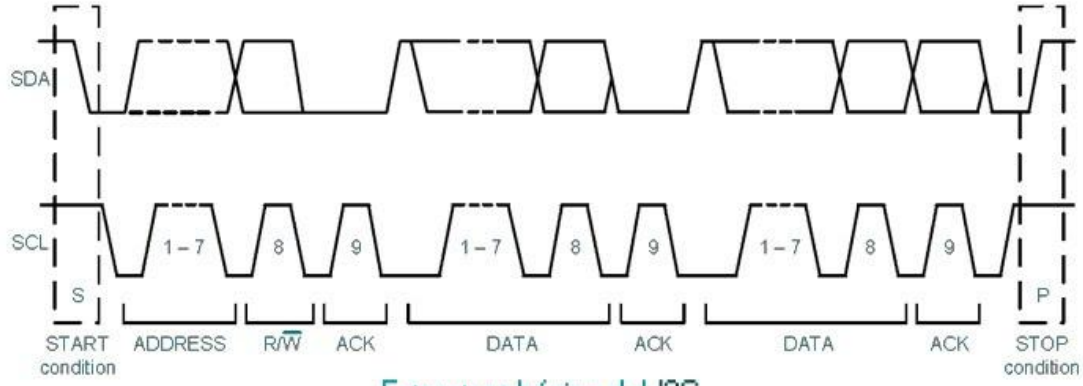
¿Qué tipo de comunicación es?
Es una comunicación de tipo **half duplex**. Comunicación bidireccional por la misma linea pero no simultáneamente bidireccional.

¿Cual es la estructura de la comunicación?
La estructura de la comunicación básica es la siguiente:

1. START condition *(Master)*
2. 7 Bits de direccion de esclavo *(Master)*
3. 1 Bit de RW, 0 es Leer y 1 Escribir. *(Master)*
4. 1 Bit de Acknowledge *(Slave)*
5. Byte de dirección de memoria *(Master)*
6. 1 Bit de Acknowledge *(Slave)*
7. Byte de datos *(Master/Slave (Escritura/Lectura))*

8. 1 Bit de Acknowledge *(Slave/Master (Escritura/Lectura))*
9. STOP condition *(Master)*

Esta es la base de la comunicación pero para leer o escribir, según el dispositivo con el que se comunica el Master la comunicación tendrá una estructura específica.



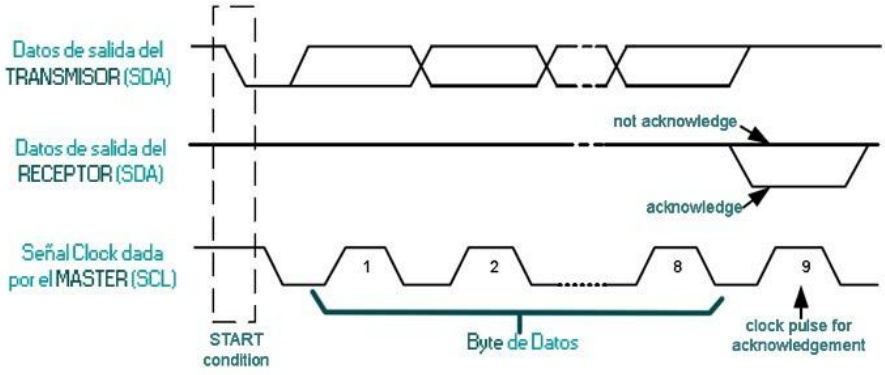
Estructura básica del I2C

¿Qué es el bit de Acknowledge (ACK)?

Este bit es una respuesta del receptor al transmisor. Es una parte básica de la comunicación y tiene diferentes sentidos según el contexto.

Se utiliza principalmente con dos propósitos:

1. Conocer si el transmisor ha sido escuchado
2. **Lecturas multidatos:**
   Cuando se esta leyendo de un esclavo, si el master realiza un ACK, es decir, que responde; el esclavo pasa al siguiente valor de registro y lo envía también, hasta recibir un NACK (Not Acknowledge), es decir, ninguna respuesta del master. Esto sirve para hacer múltiples lecturas. Por ejemplo, nuestro acelerómetro o giroscopio, que tienen valores de X, Y y Z.



ACKnowledge en protocolo I2C

En esencia este es el funcionamiento del protocolo de comunicación con el que nos comunicaremos con múltiples sensores

El I²C-bus con Arduino.

Actualmente existen muchas aplicaciones, para aplicar con la placa Arduino; como un reloj en tiempo real, potenciómetros digitales, sensores de temperatura, brújulas digitales, chips de memoria, servos, circuitos de radio FM, E/S expansores, controladores de LCD, amplificadores, etc. Y además usted puede tener más de una idea para aplicar este bus, en cualquier momento, como se ha dicho, el número máximo de dispositivos I²C utiliza en un momento dado es de 112.

Debe tenerse muy en cuenta el Arduino que está utilizando, si es un Arduino Mega, el pin SDA, es pin 20 y el SCL, es el pin 21, así que, tenga en cuenta que en los escudos con I²C, necesitan ser específicamente para el Mega. Si usted tiene otro tipo de tarjeta, revise la hoja de datos o consulte la página web de pruebas de hardware de Arduino. Y por último, si usted está usando un microcontrolador ATMega328 base DIP-PU, deberá a utilizar para SDA el pin 27 y SCL el pin 28.

Si sólo está utilizando un dispositivo I²C, las resistencias RPA, no son necesarias ya que el microcontrolador ATMega328 en nuestro Arduino las ha incorporado Sin embargo, si está empleando una serie de dispositivos utilice dos resistencias de 10k ohmios. Como todo algunas pruebas en un circuito protoboard o prototipo determinará su necesidad. La longitud máxima de un bus I²C es de alrededor de un metro y es una función de la capacidad del bus. Esta distancia se puede ampliar con el uso de un circuito especial que no examinaremos en el presente trabajo.

Cada dispositivo se puede conectar al bus en cualquier orden y, como ya se ha mencionado, los dispositivos pueden ser maestros o esclavos. En nuestra aplicación el IDE Arduino, es el maestro y los dispositivos que "colguemos" en el bus I²C son los esclavos. Podemos escribir datos en un dispositivo o leer datos de un dispositivo. A estas alturas ya debe estar pensando "¿cómo podemos diferenciar cada dispositivo en el bus?" Más arriba, de modo general, ya se ha documentado. Decíamos que, cada dispositivo tiene una dirección única. Usaremos esa dirección, en las funciones descritas más adelante para dirigir nuestras peticiones de lectura o escritura al dispositivo correcto.

Al igual que en la mayoría de dispositivos, haremos uso de una librería para Arduino, en este caso **<wire.h>**. A continuación, utilizamos la function **Wire.begin();** dentro de la configuración del **void setup()** y estaremos listos para empezar.

Envío de datos (sending), desde nuestro Arduino a los dispositivos I²C depende de dos cosas: la dirección única del dispositivo (que necesitamos esté en hexadecimal) y al menos un byte de datos a enviar. (Como ya se ha descrito arriba).

Por ejemplo; la dirección del dispositivo en el ejemplo (de abajo) es 01101000 (binario), que es 0x68 en hexadecimal.

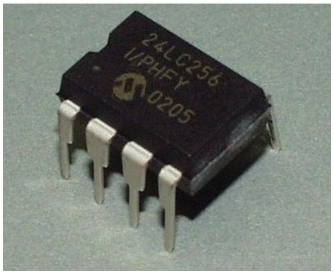
Entonces, debemos establecer el valor del puntero, que es un valor entre 0 y 127, o sea 0×00 y 0x7F en hexadecimal. Así que para establecer el puntero a cero, usaríamos las funciones siguientes:

ANEXO 3

DISPOSITIVOS I2C en el mercado

Memoria EEPROM I2C 24LC256

http://www.bricogeek.com/shop/componentes/209-memoria-eeprom-i2c-24lc256.html



Precio 2.05 €

Documentacion Memoria EEPROM de 256kbits imprescindible en cualquier proyecto con microcontroladores donde se necesite almacenar datos.

Características:

- Tecnología CMOS de bajo consumo
- Corriente máxima para escritura: 3 mA (5V)
- Corriente máxima de lectura: 400 uA (5V)
- Corriente en reposo: 100 nA
- Interfaz de dos cables I2C
- Cascada de hasta 8 memorias
- Control interno automático del ciclo de lectura/escritura
- Paginación de 64-byte para paginas
- Velocidad de escritua: 5 ms max
- Protección de escritura por hardware
- Ciclos de borrado/escritura: 1.000.000
- Retención de datos: > 200 años
- Encapsulado: 8-pin PDIP
- Pb-fre y RoHS

Documentación: Datasheet

BlinkM - I2C Controlled RGB LED

https://www.sparkfun.com/products/8579

Precio:$ 12.95

Descripción: BlinkM es un "LED inteligente", conectable en red y programable, a todo color del tipo RGB LED. Está diseñado para permitir la fácil adición de indicadores dinámicos, pantallas, y la iluminación para proyectos nuevos o existentes. Si usted ha utilizado todo su microcontrolador PWM control de los canales RGB LED, y todavía quiere más, BlinkM es para usted. BlinkM utiliza una alta calidad y de alta potencia LED RGB y un microcontrolador AVR pequeño como para permitir a un usuario para controlar digitalmente un LED RGB sobre una interfaz I2C sencilla. BlinkMs múltiples pueden ser atrapados juntos en un bus I2C que permite algunas pantallas de luz increíbles.

BlinkMs también son también programables con el software secuenciador ThingMs, puede crear una mezcla de colores, con intervalos de tiempo diferentes, cargar esa secuencia que el y luego ejecutarla.

Expansor de Canales

http://www.bricogeek.com/shop/componentes/198-expansor-i2c-pcf8575.html

Precios 9.60 €

Estas escaso de pines de entrada/salida en tu proyecto? Con éste expansor pordrás obtener 16 entradas o salidas más utilizando tan sólo el bus I2C!

Características:

- 16 pines individualmente configurables
- Cada pin puede ser configurado como entrada o salida
- Pin de interrupción
- Dimensiones: 2,5x2,5cm

Documentos:

- Información general PCF8575 (Texas Instruments)
- PCF8575C Datasheet

Compás HMC6352

http://www.bricogeek.com/shop/sensores/275-compas-hmc6352.html

Precio 28.50€

Ésta pequeña pero interesante placa hace exactamente lo que su nombre indica. Es un compás (o brújula) digital que indica los grados, de 0 a 360, con respecto al norte. Mediante su bus I2C podremos averiguar la orientación con un refresco configurable de hasta 20Hz.

Es ideal para proyecto de geolocalización, robótica, arduino o cualquier proyecto que necesite de una orientación precisa en el espacio!

Características:

- Interfaz I2C
- Alimentación 2.7 - 5.2V
- Refresco ajustable de 1 a 20Hz
- Resolución 0.5 grados
- Consumo: 1mA (3V)
- Dimensiones: 15x15mm

Documentación:

- Esquema
- Datasheet HMC6352
- Página del fabricante Honeywell
- Ejemplo con Wiring
- Ejemplo con Arduino

Sensor de humedad y temperatura SHT15

http://www.bricogeek.com/shop/sensores/36-sensor-de-humedad-y-temperatura-sht15.html

Precio 32,90 €

Este estupendo sensor "2 en 1" proporciona a la vez una medición de temperatura como de humedad relativa en un solo chip. Además ofrece la ventaja de que los datos se recuperan mediante el bus I2C con lo que podemos usarlo facilmente con cualquier microcontrolador como PIC, AVR, BasicStamp, Propeller etc.
Al disponer de dos sensores en un mismo encapsulado, avitaremos usar otras patillas de nuestro microcontrolador, las cuales podemos usar ahora para otros fines.

Documentación:

- Datasheet SHT1X (PDF)
- Código de ejemplo en C para PIC16F88 (zip)
- Esquema electrico de la placa (PDF)