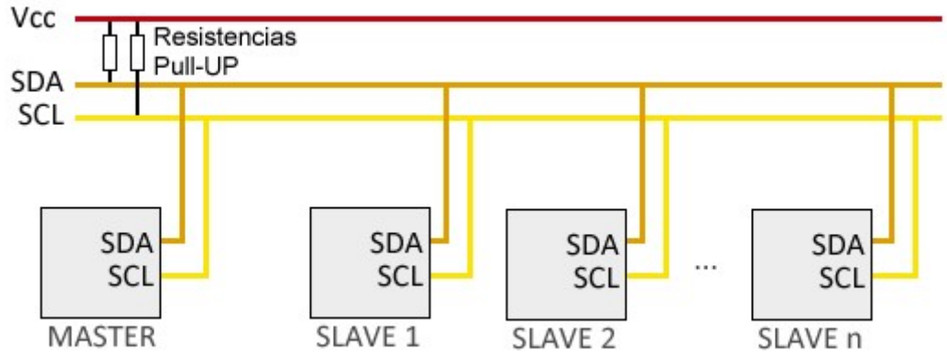
EL BUS I2C EN ARDUINO

18 MAYO, 2016

En esta entrada veremos el bus I2C, uno de los sistemas de comunicación disponible en Arduino. En entradas anteriores ya hemos visto el puerto serie y el bus SPI que, junto al bus I2C, integran los principales sistemas de comunicación.
El bus I2C tiene interés porque, de forma similar a lo que pasaba con el bus SPI, una gran cantidad de dispositivos disponen conexión mediante I2C, como acelerómetros, brújulas, displays, etc.

EL BUS I2C

El estándar I2C (Inter-Integrated Circuit) fue desarrollado por Philips en 1982 para la comunicación interna de dispositivos electrónicos en sus artículos. Posteriormente fue adoptado progresivamente por otros fabricantes hasta convertirse en un estándar del mercado.
I2C también se denomina TWI (Two Wired Interface) únicamente por motivos de licencia. No obstante, la patente caducó en 2006, por lo que actualmente no hay restricción sobre el uso del término I2C.
El bus I2C requiere únicamente dos cables para su funcionamiento, uno para la señal de reloj (CLK) y otro para el envío de datos (SDA), lo cual es una ventaja frente al bus SPI. Por contra, su funcionamiento es un poco más complejo, así como la electrónica necesaria para implementarla.



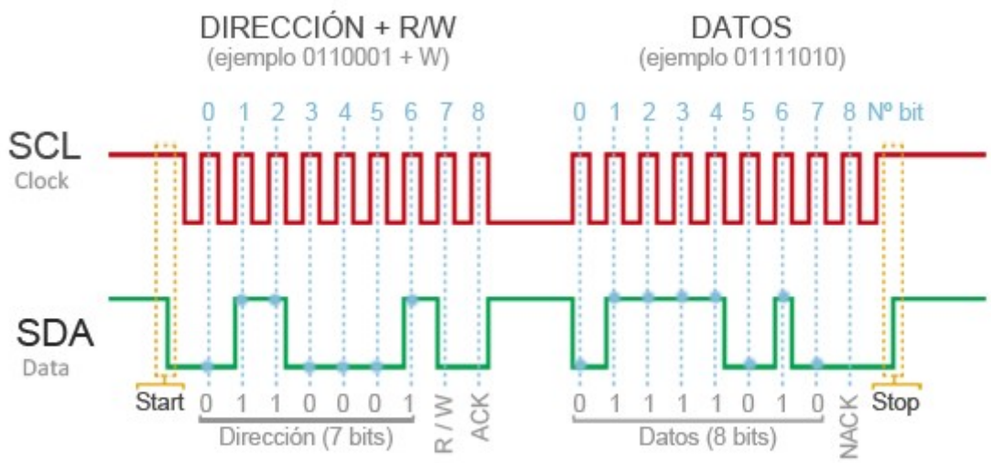
En el bus Cada dispositivo dispone de una dirección, que se emplea para acceder a los dispositivo de forma individual. Esta dirección puede ser fijada por hardware (en cuyo caso, frecuentemente, se pueden modificar los últimos 3 bits mediante jumpers o interruptores) o totalmente por software.
En general, cada dispositivo conectado al bus debe tener una dirección única. Si tenemos varios dispositivos similares tendremos que cambiar la dirección o, en caso de no ser posible, implementar un bus secundario.
El bus I2C tiene una arquitectura de tipo maestro-esclavo. El dispositivo maestro inicia la comunicación con los esclavos, y puede mandar o recibir datos de los esclavos. Los esclavos no pueden iniciar la comunicación (el maestro tiene que preguntarles), ni hablar entre si directamente.
Es posible disponer de más de un maestro, pero solo uno puede ser el maestro cada vez. El cambio de maestro supone una alta complejidad, por lo que no es algo frecuente.
El bus I2C es síncrono. El maestro proporciona una señal de reloj, que mantiene sincronizados a todos los dispositivos del bus. De esta forma, se elimina la necesidad de

que cada dispositivo tenga su propio reloj, de tener que acordar una velocidad de transmisión y mecanismos para mantener la transmisión sincronizada (como en UART)
El protocolo I2C prevé resistencias de Pull-UP de las líneas a Vcc. En Arduino veréis que frecuentemente no se instalan estas resistencias, ya que la librería Wire activa las resistencias internas de Pull-UP. Sin embargo las resistencias internas tienen un valor de entre 20-30kOhmnios, por lo que son unas resistencias de Pull-UP muy blandas.
Usar unas resistencias blandas implica que los flancos de subida de la señal serán menos rápidas, lo que implica que podremos usar velocidades menores y distancias de comunicación inferiores. Si queremos emplear velocidades o distancias de transmisión superiores, deberemos poner físicamente resistencias de Pull-UP de entre 1k a 4K7.

FUNCIONAMIENTO DEL BUS I2C

Para poder realizar la comunicación con solo un cable de datos, el bus I2C emplea una trama (el formato de los datos enviados) amplia. La comunicación costa de:

- 7 bits a la dirección del dispositivo esclavo con el que queremos comunicar.
- Un bit restante indica si queremos enviar o recibir información.
- Un bit de validación
- Uno o más bytes son los datos enviados o recibidos del esclavo.
- Un bit de validación



Con estos 7 bits de dirección es posible acceder a 112 dispositivos en un mismo bus (16 direcciones de las 128 direcciones posibles se reservan para usos especiales).
Este incremento de los datos enviados (18bits por cada 8bits de datos) supone que, en general, la velocidad del bus I2C es reducida. La velocidad estándar de transmisión es de 100Mhz, con un modo de alta velocidad de 400Mhz.
El estándar I2C define otros modos de funcionamiento, como un envío de dirección de 8,10 y 12bits, o velocidades de transmisión de 1Mbit/s, 3.4Mbit/s y 5Mbit/s. No suelen ser empleados en Arduino.

VENTAJAS Y DESVENTAJAS DEL I2C

VENTAJAS

- Requiere pocos cables
- Dispone de mecanismos para verificar que la señal hay llegado

DESVENTAJAS

- Su velocidad es media-baja
- No es full duplex
- No hay verificación de que el contenido del mensaje es correcto

EL BUS I2C EN ARDUINO

Arduino dispone de soporte I2C por hardware vinculado físicamente a ciertos pines. También es posible emplear cualquier otro grupo de pines como bus I2C a través de software, pero en ese caso la velocidad será mucho menor.

Los pines a los que está asociado varían de un modelo a otro. La siguiente tabla muestra la disposición en alguno de los principales modelos. Para otros modelos, consultar el esquema de patillaje correspondiente.

MODELO	SDA	SCK
Uno	A4	A5
Nano	A4	A5
Mini Pro	A4	A5
Mega	20	21

Para usar el bus I2C en Arduino, el IDE Standard proporciona la librería “Wire.h”, que contiene las funciones necesarias para controlar el hardware integrado.

Algunas de las funciones básicas son las siguientes

Wire.begin() // Inicializa el hardware del bus

Wire.beginTransmission(address); //Comienza la transmisión

Wire.endTransmission(); // Finaliza la transmisión

Wire.requestFrom(address,nBytes); //solicita un numero de bytes al esclavo en la dirección address

Wire.available(); // Detecta si hay datos pendientes por ser leídos

Wire.write(); // Envía un byte

Wire.read(); // Recibe un byte

Wire.onReceive(handler); // Registra una función de callback al recibir un dato

Wire.onRequest(handler); // Registra una función de callback al solicitar un dato

Existen otras librerías más avanzadas que Wire.h para manejar el bus I2C, como por ejemplo I2Cdevlib o I2C library.

ESCÁNER DE I2C

En un mundo ideal sabríamos la dirección de dispositivo que compramos. Pero en algunas ocasiones, sobre todo al comprar en vendedores chinos, el fabricante no nos facilita la dirección del dispositivo o incluso lo proporciona de forma incorrecta.

Esta es una circunstancia común y nada preocupante. Para eso disponemos de un sketch llamado “Scanner I2C” que realiza un barrido por todas las posibles direcciones del bus, y muestra el resultado en caso de encontrar un dispositivo en la dirección.

De esta forma podemos determinar cómodamente las direcciones de los dispositivos desconocidos.

El sketch scanner I2C está disponible en este enlace, o podéis usar la siguiente versión reducida y traducida.

```
#include "Wire.h"

extern "C" {
    #include "utility/twi.h"
}

void scanI2CBus(byte from_addr, byte to_addr, void(*callback)(byte address, byte result) )
{
  byte rc;
  byte data = 0;
  for( byte addr = from_addr; addr <= to_addr; addr++ ) {
    rc = twi_writeTo(addr, &data, 0, 1, 0);
    callback( addr, rc );
  }
}

void scanFunc( byte addr, byte result ) {
  Serial.print("addr: ");
  Serial.print(addr,DEC);
  Serial.print( (result==0) ? " Encontrado!":"       ");
  Serial.print( (addr%4) ? "\t":"\n");
}


const byte start_address = 8;
const byte end_address = 119;

void setup()
{
    Wire.begin();

    Serial.begin(9600);
    Serial.print("Escaneando bus I2C...");
    scanI2CBus( start_address, end_address, scanFunc );
    Serial.println("\nTerminado");
}

void loop()
{
    delay(1000);
}
```