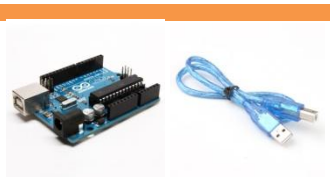
MÓDULO MP3 WTV020-SD-16P Y ARDUINO

OBJETIVOS

- ✓ Integrar sonido en nuestros proyectos electrónicos de una forma sencilla y barata.

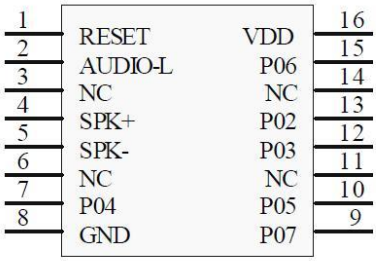
MATERIAL NECESARIO

	Ordenador con el IDE instalado
	Placa de Arduino UNO (u otro modelo), con el cable USB
	Decodificador WTV020SD-16P, y una tarjeta microSD de 2Gb o menos
	3 pulsadores
	Cables para conexiones
	Un altavoz

Decodificador WTV020SD-16P

El decodificador WTV020SD-16P es un dispositivo de bajo costo (unos 2,35€) que nos va a permitir añadir sonido a nuestros proyectos electrónicos con una calidad de sonido superior a como si lo hiciéramos con el propio Arduino.

Vamos a estudiar cómo es y cómo funciona este dispositivo. En las siguientes imágenes podemos ver cuáles son sus pines y la función de cada uno de ellos:

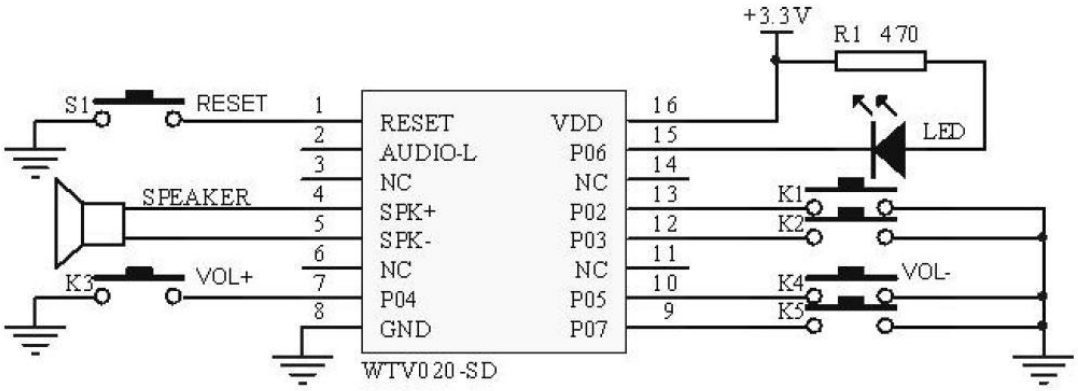


PIN	SYS.	DESCRIPTION	FUNCTION
1	RESET	RESET	Reset pin
2	AUDIO-L	DAC+	DAC audio output(+) to amplifier
3	NC	NC	NC
4	SP+	PWM+	PWM audio output to speaker
5	SP-	PWM-	PWM audio output to speaker
6	NC	NC	NC
7	P04	K3/A2/CLK	Key /CLK in two line serial
8	GND	GND	Address pin
9	P07	K5/A4/SBT	Key
10	P05	K4/A3/DI	Key /DI in two line serial
11	NC	NC	NC
12	P03	K2/A1	Key
13	P02	K1/A0	Key
14	NC	NC	NC
15	P06	BUSY	BUSY pin
16	VDD	VDD	Power input

Con este módulo podemos trabajar de dos modos:

- De forma autónoma (sólo con una pequeña circuitería).
- Conectado a nuestro Arduino y utilizando una librería.

Vamos a ver el primer caso. Para ello deberíamos montar el siguiente esquema:



Básicamente nos valdría con alimentar el dispositivo (3,3V a VDD-> Importante!!!, con más tensión podríais deteriorarlo, y 0V a GND). Luego conectaríamos un pulsador para el RESET y otros dos (P02 y P03) para avanzar/retroceder en las pistas de audio. Por último, conectaríamos un altavoz a los pines SPK+ y SPK-.

El circuito no tiene más dificultad. Ahora sí, debemos tener en cuenta que el dispositivo lee los ficheros de audio en formato .ad4. Para ello debemos convertir nuestros ficheros .mp3 o .wav a dicho formato. Esto lo podemos hacer descargándonos el siguiente programa 4D-SOMO-tool, y viendo los sencillo pasos a seguir en el siguiente vídeo.

Aquí tenéis algunos archivos .ad4 de muestra para hacer alguna prueba si queréis antes de convertir vuestros propios sonidos.

Recomendaciones para no tener problemas con los ficheros de audio:

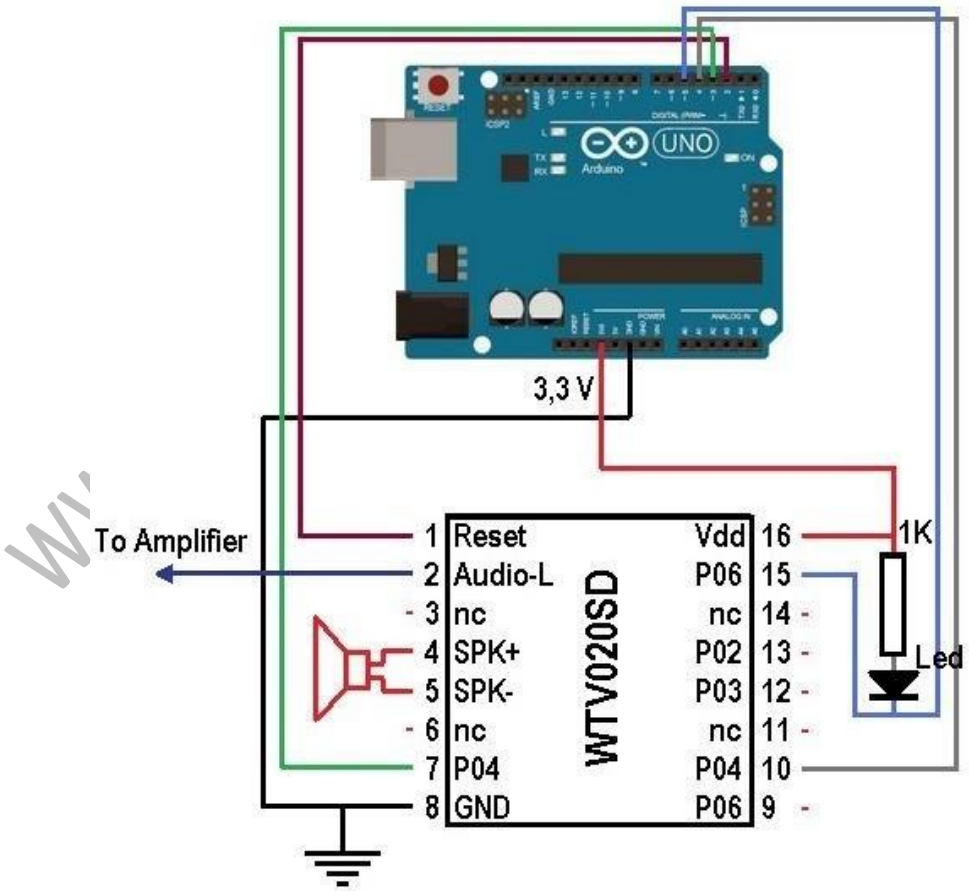
- La tarjeta microSD debe ser de cómo mucho 2 Gb.
- La tarjeta microSD debe estar formateada como FAT32.
- Los archivos de audio deben ser codificados en 4-bit ADPCM.
- Los ficheros de audio deben de tener la extensión .ad4
- Los ficheros de audio deben de estar en el raíz de la tarjeta microSD, y no dentro de carpetas (y preferiblemente solos en el raíz, sin más archivos de otro tipo).
- Los ficheros de audio los debemos renombrar empezando por el 0000.ad4 y hasta el 0511.ad4.

Podéis ver el funcionamiento de este modo en el siguiente vídeo.

Yo, en concreto, en lugar de montar el regulador de tensión para conseguir los 3,3V, lo conecté a la salida de 3,3V de un Arduino y me funciona perfectamente también.

Si optamos por el otro modo (conectado al Arduino), deberemos instalar en nuestro IDE la siguiente librería Wtv020sd16p, y conectar nuestro módulo al Arduino según la siguiente tabla y esquema:

	WTV020-SD-16P	Arduino
VCC	16 (VDD)	3,3V
GND	8 (GND)	GND
RESET	1 (RESET)	D2
CLOCK	7 (P04)	D3
DATA	10 (P05)	D4
BUSY	15 (P06)	D5



En este caso particular, vamos a desarrollar una práctica donde vamos a reproducir los diferentes archivos de audio que hemos grabado en la micro SD en formato .ad4. Por simplicidad, lo vamos a hacer a través del puerto serie:

- Si enviamos un '1', sonará la pista 1.
- Si enviamos un '2', sonará la pista 2.
- Si enviamos un '3', sonará la pista 3.
- Y si enviamos un '4', sonará la pista 4.

El código es el siguiente:

```
#include <Wtv020sd16p.h>

int resetPin = 2;  // The pin number of the reset pin.
int clockPin = 3;  // The pin number of the clock pin.
int dataPin = 4;  // The pin number of the data pin.
int busyPin = 5;  // The pin number of the busy pin.

Wtv020sd16p wtv020sd16p(resetPin,clockPin,dataPin,busyPin);
String orden;

void setup() {
 //Initializes the module.
 wtv020sd16p.reset();
 Serial.begin(9600);
}

void loop() {
 if (Serial.available()>0)
 {
   wtv020sd16p.reset();
   orden=Serial.readString();
   if (orden=="1")
     {
     wtv020sd16p.stopVoice();
     wtv020sd16p.playVoice(1);
     Serial.println(orden);
     }
```

```
    if (orden=="2")
      {
      wtv020sd16p.stopVoice();
      wtv020sd16p.playVoice(2);
      Serial.println(orden);
      }
    if (orden=="3")
      {
      wtv020sd16p.stopVoice();
      wtv020sd16p.playVoice(3);
      Serial.println(orden);
      }
    if (orden=="4")
      {
      wtv020sd16p.stopVoice();
      wtv020sd16p.playVoice(4);
      Serial.println(orden);
      }
  }
}
```

Y aquí tenéis un vídeo del resultado.

Como podéis observar, el esquema es sencillo y el código no tiene gran complicación (no olvidéis incluir la librería en vuestro IDE).

Ahora sólo queda que penséis en vuestra propia aplicación, por ejemplo, un ascensor donde nos vaya diciendo un mensaje de voz según su estado ("subiendo", "bajando", "abriendo puertas", "planta baja", ""planta 1", etc), o un juguete que vaya diciendo mensajes según los estímulos del entorno recibidos.

HEMOS APRENDIDO…

- ✓ El patillaje y funcionamiento del módulo MP3 WTV020-SD-16P.
- ✓ A instalar una librería en nuestro IDE.
- ✓ A convertir ficheros MP3 y WAV a formato .ad4.