SONIDO CON ARDUINO

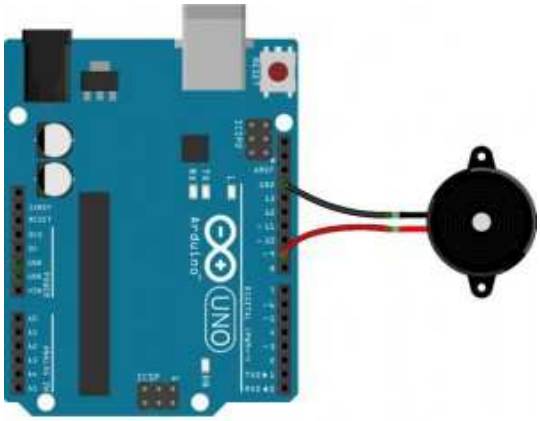
Podemos generar sonidos fácilmente con Arduino con un simple zumbador (buzzer) piezoeléctrico. El sonido es de bastante mala calidad pero nos resulta suficiente para introducir sonidos de alarmas o avisos en nuestros programas.

Los materiales piezoeléctricos tienen la propiedad de vibrar al ser sometidos a una tensión eléctrica variable. Dependiendo de la frecuencia con la que varíe la tensión, variará la frecuencia de vibración y por tanto el sonido que se produzca.

También podemos utilizar pequeños altavoces. En este caso, las vibraciones de la membrana se consiguen sometiendo una bobina a una tensión variable, creando un campo magnético variable que provoca el movimiento del núcleo de la bobina.

Podemos obtener sonidos simples de varias formas:

1.- Conectando el zumbador a un pin PWM



Los pines PWM de Arduino (los que vienen marcados con virgulilla (~) en la placa) proporcionan, cuando se les utiliza con la función analogWrite(pin, valor), una señal cuadrada donde se alternan el LOW y el HIGH a una frecuencia fija.

Por tanto, tan solo tenemos que conectar la patilla negativa del zumbador a GND y la positiva a un pin PWM de Arduino (en la tarjeta Arduino 1 son PWM los pines 3, 5, 6, 9, 10 y 11) y utilizar analogWrite().

Ejemplo: el siguiente programa hará que el zumbador emita pitidos intermitentes indefinidamente. El valor 20 (sobre 255) utilizado en analogWrite() podría ser otro pero esto no variará mucho el tono, pues la frecuencia de la señal viene fijada por Arduino.

```
const int pinbuzzer=9;
void setup(){
  pinMode(pinbuzzer,OUTPUT);
}
void loop(){
  beep(200);  //el pitido durará 200 milisegundos
}
void beep(int pausa){
  analogWrite(pinbuzzer, 20);//empieza a pitar
  delay(pausa); // se mantiene pitando un tiempo igual a pausa
  analogWrite(pinbuzzer, 0); //deja de pitar
  delay(pausa); //se mantiene en silencia un tiempo igual a pausa
}
```

2.- Conectando el zumbador a un pin digital cualquiera y creando nosotros manualmente la señal cuadrada.

Creamos una señal cuadrada a base de alternar valores HIGH y LOW en un pin digital cualquiera, utilizando en este caso la función digitalWrite().

Ejemplo: este programa emite pitidos intermitentes de forma indefinida, aunque la frecuencia puede variar y escucharse un tono diferente.

```
const int pinbuzzer=9;
void setup(){
  pinMode(pinbuzzer,OUTPUT);
}
void loop(){
  beep(200);  //el pitido durará 200 milisegundos
}
void beep(int pausa){
  for(int i=0;i<(2*pausa);i++){
    digitalWrite(pinbuzzer, HIGH);
    delayMicroseconds(300);
    digitalWrite(pinbuzzer, LOW);
    delayMicroseconds(200);
  }
  delay(pausa);
}
```

3.- Utilizando las funciones tone() y noTone() de Arduino

Esta es la mejor forma si quiero generar señales de tono variable.

La función **tone(***pin, frecuencia, duración***)** genera en el pin que se le indica como primer parámetro una señal de la frecuencia (en Hz) que se le introduce como segundo parámetro durante el tiempo (en ms) que se le indica como tercer parámetro. Si antes de que pase el tiempo marcado por el parámetro *duración* se ejecuta la función **noTone**(pin) el sonido se detiene. Igualmente, si antes de que pase el tiempo marcado por *duración* se ejecuta una nueva instrucción **tone**() en el mismo pin la orden **tone**() anterior queda anulada.

El parámetro duración es opcional; si no se especifica, el sonido continúa hasta que se ejecuta la orden **noTone(***pin***)**. En caso de no especificar la duración la función **tone**() se comporta igual que la **digitalWrite**(), que cuando se le da la orden de poner el pin de salida a HIGH, lo mantiene en este estado hasta que se le da la orden de ponerlo a LOW.

Ejemplo: el siguiente programa hace que un buzzer conectado al pin 8 emita un sonido correspondiente a una frecuencia de 440 Hz durante 1 segundo.

```
const int pinbuzzer=8;
int frecuencia=440;
int duracion=1000;
void setup(){
  tone(pinbuzzer, frecuencia, duracion);
}
void loop(){ }
```

Hacer funcionar la instrucción tone() especificando una duración, no quiere decir que hasta que termina el sonido el programa se quede parado (como ocurre con delay() por ejemplo). El programa puede seguir haciendo otras cosas sin tener que preocuparnos de la terminación del sonido. Al cabo del tiempo especificado en *duración*, el sonido terminará, incluso aunque en ese momento el programa esté ejecutando un delay().

Ejemplo: el siguiente programa hará sonar un pitido durante 5 segundos. Al mismo tiempo irá imprimiendo en el monitor serie un contador que se incrementará cada 2 segundos. Si lo ejecutamos observaremos que el sonido se apaga después de haber impreso el 2 y antes de imprimir el 3, es decir, cuando el programa está en un delay().

```
const int pinbuzzer=8;

void setup(){
  tone(pinbuzzer,440,5000);
  Serial.begin(9600);
  for(int i=0;i<10;i++){
    Serial.println(i);
    delay(2000);
  }
}
void loop(){}
```

Reproducción de sonidos cambiantes

A veces nos interesará hacer sonar un sonido que vaya cambiando de tono. Por ejemplo, las típicas sirenas de incendios o de detección de intrusos. Para ello, hay que ir cambiando la frecuencia del sonido.

Ejemplo: Este programa emite el sonido de una sirena. Un bucle for va cambiando la frecuencia en pasos de 1 Hz dentro de un rango (en este caso entre 2000 y 4000 Hz) primero en sentido ascendente y después en sentido descendente. Podríamos seguir cualquier otro patrón matemático que ligue cada frecuencia con la anterior.

```
int fMin=2000; //Frecuencia más baja que queremos emitir
int fMax=4000; //Frecuencia más alta que queremos emitir
int pinbuzzer=7;

void setup(){
   pinMode (pinbuzzer, OUTPUT); //pin configurado como salida
}

void loop(){
  for (int i=fMin;i<=fMax; i++)
     tone(pinbuzzer, i);
  for (int i=fMax;i>=fMin; i--)
     tone(pinbuzzer, i);
}
```

Cuando lo que queremos hacer sonar es una melodía formada por una sucesión de notas que no siguen un patrón matemático, como en el caso anterior, conviene definir una matriz con las frecuencias de las notas a reproducir y leerlas todas sucesivamente con un bucle for().

Ejemplo: este programa emite una sucesión de tonos con una duración de 200 ms cada uno de ellos. Entre sucesión y sucesión se calla durante un segundo.

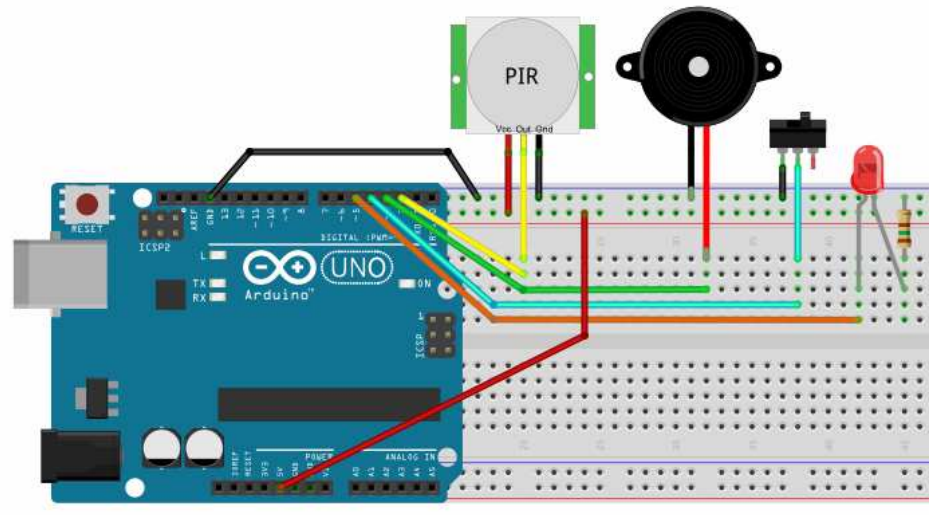
```
const int pinbuzzer=9;
int numerotonos=12;
int tonos[]={261,277,294,311,330,349,370,392,415,440,466,494};

void setup(){}
void loop(){
  for(int i=0;i<numerotonos;i++){
    tone(pinbuzzer,tonos[i]);
    delay(200);
  }
  noTone(pinbuzzer);
  delay(1000);
}
```

Reproducción de sonidos simultáneamente a la ejecución de otras funciones

En los dos ejemplos anteriores, una vez que la ejecución del programa entra en los bucles for y empiezan los sonidos, el programa ya no hace otra cosa hasta que acaban dichos bucles. Esto es poco eficiente. Generalmente necesitaremos que una vez que salte una alarma ésta siga sonando mientras el programa pueda atender a otras órdenes, como la desconexión de dicha alarma, el encendido de luces, la introducción de una clave por un teclado, etc.

Ejemplo: el siguiente programa controla un sistema de alarma. Por simplificar, la alarma se conecta y desconecta mediante un interruptor. Cuando la alarma está conectada o activada, está encendido el LED. El sonido de la alarma lo produce un zumbador. La activación de la alarma es producida por un sensor de movimiento PIR. Estando conectada la alarma, si se activa el sensor PIR suena un sonido breve (para recordar al propietario que tiene que desconectar la alarma y se da un margen de 10 segundos para desconectar la alarma antes de hacer sonar el zumbador con el mismo sonido de sirena de un ejemplo anterior (esto es para dar al propietario tiempo de desconectar la alarma). Observa cómo está construida la función toca_alarma(). Esta función está haciendo cambiar el tono de zumbador y haciendo que suenen dichos tonos mientras que Arduino sigue controlando la desconexión de la alarma.



```
const int pinPIR=2; //detector presencia PIR
const int pinZumb=3; //zumbador
const int pinInt=4;  //interruptor
const int pinLED=5;
int estadoalarma=1;
unsigned long tini;
const int Fmin=2000; //frec. mínima sonido
const int Fmax=4000; //frec. máxima sonido
int tono;
int ascendente=1; //sentido variación tono

void setup() {
  pinMode(pinPIR,INPUT);
  pinMode(pinZumb,OUTPUT);
  pinMode(pinInt,INPUT_PULLUP);
  pinMode(pinLED,OUTPUT);

}

void loop() {
  revisa_alarma();
}

void revisa_alarma(){
  switch (estadoalarma){
    case 1:  //alarma desconectada
      noTone(pinZumb);
      tono=Fmin-1;
      ascendente=1;
      digitalWrite(pinLED,LOW);
      if(digitalRead(pinInt)==LOW)estadoalarma=2;
      break;
    case 2:  //alarma conectada
      digitalWrite(pinLED,HIGH);
      if(digitalRead(pinInt)==HIGH)estadoalarma=1;
      if(digitalRead(pinPIR)==HIGH){
        estadoalarma=3;
        tone(pinZumb,440,200); //pitido de aviso
        tini=millis();
      }
      break;
    case 3:  //en estado de espera desconexión
      if((millis()-tini)>10000)estadoalarma=4;
      if(digitalRead(pinInt)==LOW)estadoalarma=1;
      break;
    case 4:  //alarma activada
      toca_alarma();
      if(digitalRead(pinInt)==LOW)estadoalarma=1;
      break;
  }
}
```

```
void toca_alarma(){
  if(ascendente==1){
    tono++;
    if(tono>=Fmax)ascendente=0;
  }
  else {
    tono--;
    if(tono<=Fmin)ascendente=1;
  }
  tone(pinZumb,tono);
}
```