UnoLib documentation

tone.pas

version 11/10/2025

The tone.pas module contains square wave tone routines.

Routines

```
procedure ToneNoInt(Pin: UInt8; Freq: UInt16; WaveTime:
UInt32);
```

Generates a square wave with the specified frequency using a timer on the specified digital pin, blocking execution of other routines in the program during *WaveTime*.

Parameters
Pin – digital pin on which to generate the tone.
Freq – the frequency of the tone in Hertz (Hz) (cycles per second). Range is typically from 31 Hz up to 65,535 Hz (though human hearing is limited).
WaveTime – time of the wave in microseconds

Note: extra procedure, not present in Arduino sources.

```
procedure _tone(_pin: UInt8; frequency: UInt16; duration:
UInt32 = 0);
```

Generates square wave with specified frequency using timer on the specified digital pin. Once called, the tone generation runs in the background, so the procedure is non-blocking.

Parameters
_pin – digital pin on which to generate the tone.
frequency – the frequency of the tone in Hertz (Hz) (cycles per second). Range is typically from 31 Hz up to 65,535 Hz (though human hearing is limited).
duration - the duration of the tone in milliseconds (ms). If omitted, the tone plays indefinitely until *noTone* is called.

```
procedure noTone(_pin: UInt8);
```

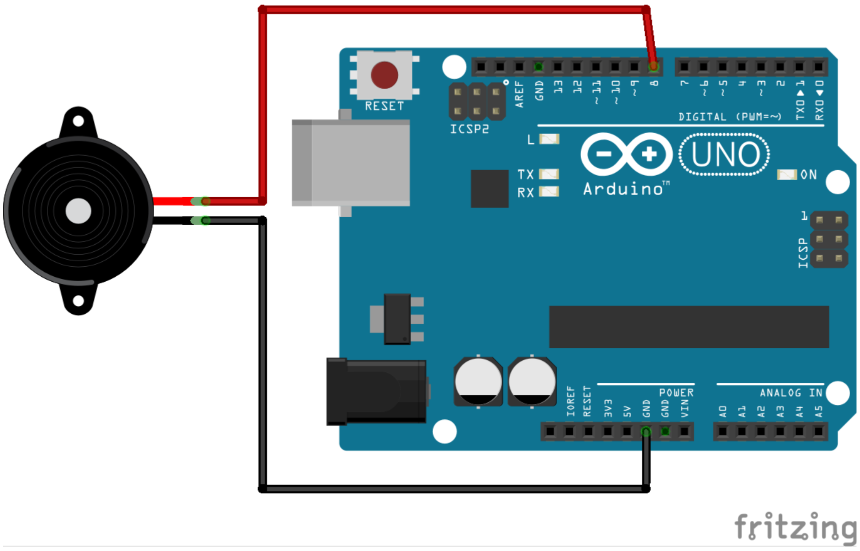
Used to stop the square wave generation that was started by the *_tone* procedure on a specific digital pin.

Parameters
_pin – digital pin on which the tone is currently playing. It is the same pin used in the preceding *_tone* call.

Example

This example is designed to play a pre-defined melody through an 8-ohm speaker or buzzer connected to Digital Pin 8 of Arduino.

The code iterates through an array of musical notes and their durations. For each note, it calculates its duration in milliseconds and uses the *_tone* procedure to start generating the required frequency (pitch) on pin 8. It calculates a brief pause (130% of the note's duration) to separate the notes, uses the delay function to wait for that period, and then calls *noTone*(8) to stop the current sound before playing the next one.



```
program TestTone;

{
  Plays a melody

  circuit:
  - 8 ohm speaker/buzzer (+) on digital pin 8 and (-) on GND

  ported 2025 to Pascal by @ackarwow
  based on melody sketch by Tom Igoe
(https://www.arduino.cc/en/Tutorial/Tone)
}

{$IFDEF AVRPascal}
  {$IF NOT (DEFINED(atmega328p) or DEFINED(arduinouno) or
DEFINED(arduinonano))}
    {$Fatal Invalid controller type, expected: atmega328p, arduinouno, or
arduinonano}
  {$ENDIF}
{$ELSE}
  {$IF NOT (DEFINED(fpc_mcu_atmega328p) or DEFINED(fpc_mcu_arduinouno) or
DEFINED(fpc_mcu_arduinonano))}
    {$Fatal Invalid controller type, expected: atmega328p, arduinouno, or
arduinonano}
```

```
  {$ENDIF}
{$ENDIF}

uses
  tone, timer, float32;

const
  NOTE_B0  = 31;   NOTE_C1 = 33;    NOTE_CS1 = 35;   NOTE_D1  = 37;
  NOTE_DS1 = 39;   NOTE_E1  = 41;   NOTE_F1  = 44;   NOTE_FS1 = 46;
  NOTE_G1  = 49;   NOTE_GS1 = 52;   NOTE_A1  = 55;   NOTE_AS1 = 58;
  NOTE_B1  = 62;   NOTE_C2  = 65;   NOTE_CS2 = 69;   NOTE_D2  = 73;
  NOTE_DS2 = 78;   NOTE_E2  = 82;   NOTE_F2  = 87;   NOTE_FS2 = 93;
  NOTE_G2  = 98;   NOTE_GS2 = 104;  NOTE_A2  = 110;  NOTE_AS2 = 117;
  NOTE_B2  = 123;  NOTE_C3  = 131;  NOTE_CS3 = 139;  NOTE_D3  = 147;
  NOTE_DS3 = 156;  NOTE_E3  = 165;  NOTE_F3  = 175;  NOTE_FS3 = 185;
  NOTE_G3  = 196;  NOTE_GS3 = 208;  NOTE_A3  = 220;  NOTE_AS3 = 233;
  NOTE_B3  = 247;  NOTE_C4  = 262;  NOTE_CS4 = 277;  NOTE_D4  = 294;
  NOTE_DS4 = 311;  NOTE_E4  = 330;  NOTE_F4  = 349;  NOTE_FS4 = 370;
  NOTE_G4  = 392;  NOTE_GS4 = 415;  NOTE_A4  = 440;  NOTE_AS4 = 466;
  NOTE_B4  = 494;  NOTE_C5  = 523;  NOTE_CS5 = 554;  NOTE_D5  = 587;
  NOTE_DS5 = 622;  NOTE_E5  = 659;  NOTE_F5  = 698;  NOTE_FS5 = 740;
  NOTE_G5  = 784;  NOTE_GS5 = 831;  NOTE_A5  = 880;  NOTE_AS5 = 932;
  NOTE_B5  = 988;  NOTE_C6  = 1047; NOTE_CS6 = 1109; NOTE_D6  = 1175;
  NOTE_DS6 = 1245; NOTE_E6  = 1319; NOTE_F6  = 1397; NOTE_FS6 = 1480;
  NOTE_G6  = 1568; NOTE_GS6 = 1661; NOTE_A6  = 1760; NOTE_AS6 = 1865;
  NOTE_B6  = 1976; NOTE_C7  = 2093; NOTE_CS7 = 2217; NOTE_D7  = 2349;
  NOTE_DS7 = 2489; NOTE_E7  = 2637; NOTE_F7  = 2794; NOTE_FS7 = 2960;
  NOTE_G7  = 3136; NOTE_GS7 = 3322; NOTE_A7  = 3520; NOTE_AS7 = 3729;
  NOTE_B7  = 3951; NOTE_C8  = 4186; NOTE_CS8 = 4435; NOTE_D8  = 4699;
  NOTE_DS8 = 4978;

const
  melody: array[0..7] of UInt16 = (NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3,
NOTE_G3, 0, NOTE_B3, NOTE_C4);
  noteDurations: array[0..7] of UInt32 = (4, 8, 8, 4, 4, 4, 4, 4);

const
  Raw1_3 = $3FA66666; // = 1.30
var
  thisNote: UInt8;
  noteDuration, pauseBetweenNotes: UInt16;
  fpause: TRawFloat32;
begin
  // iterate over the notes of the melody:

  for thisNote:=0 to 7 do
  begin
    // to calculate the note duration, take one second divided
    // by the note type.
    // e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.

    noteDuration:= 1000 div noteDurations[thisNote];
    _tone(8, melody[thisNote], noteDuration);

    // to distinguish the notes, set a minimum time between them.
    // the note's duration + 30% seems to work well:

    //pauseBetweenNotes:= noteDuration * 1.30;
    fpause:=Float32Mul(IntToFloat32(noteDuration), Raw1_3);
    pauseBetweenNotes:= Float32ToInt(fpause);

    delay(pauseBetweenNotes);

    // stop the tone playing:
    noTone(8);
  end;

  while true do;
  // no need to repeat the melody.

end.
```