UnoLib documentation

pulse.pas

version 11/10/2025

The pulse.pas module contains routines for reading a pulse on a pin.

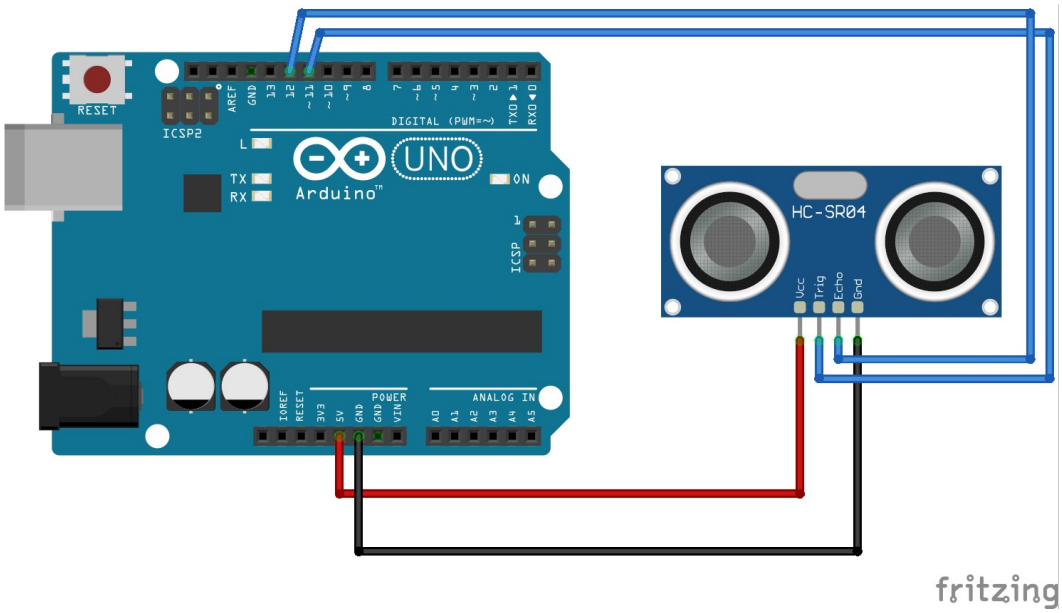
Routines
`function` **`pulseInMilli`**`(pout, State: UINT8; TimeOut: UInt32): UInt32;` Blocking function used to measure the duration of a digital pulse (either HIGH or LOW) on a specified pin. It returns value in milliseconds. *Parameters* *pin* – digital pin number on which to read the pulse. This pin must be set to INPUT via PinMode. *state* – the type of pulse to read: either HIGH or LOW. *TimeOut* - the maximum number of milliseconds to wait for the pulse to start Note: extra procedure, not present in Arduino sources.
`function` **`pulseIn`**`(pin: UInt8; state: UInt8; timeout: UInt32 = 1000000): UInt32;` Blocking function used to measure the duration of a digital pulse (either HIGH or LOW) on a specified pin. It returns value in microseconds. It generally works well for pulses from 10 microseconds up to about 3 minutes in length, though accuracy can decrease with very long pulses. *Parameters* *pin* – digital pin number on which to read the pulse. This pin must be set to INPUT via PinMode. *state* – the type of pulse to read: either HIGH or LOW. *timeout* - the maximum number of microseconds to wait for the pulse to start. Defaults to one second (1,000,000 microseconds).
`function` **`pulseInLong`**`(pin: UInt8; state: UInt8; timeout: UInt32 = 1000000): UInt32;` Advanced, blocking function in Arduino that is an alternative to *pulseIn*. It is specifically designed to provide better accuracy when measuring long-duration digital pulses (either HIGH or LOW) on a specified pin, particularly in scenarios where interrupts are active. *Parameters* *pin* – digital pin number on which to read the pulse. This pin must be set to INPUT via PinMode.

state – the type of pulse to read: either HIGH or LOW. *timeout* - the maximum number of microseconds to wait for the pulse to start. Defaults to one second (1,000,000 microseconds).

Example

This example is designed to measure the distance to an object using an HC-SR04 ultrasonic sensor and output the result in centimeters to the Serial Monitor. The program uses a low-level implementation, including manual pin control and raw floating-point arithmetic (*TRawFloat32*).

The code initializes an HC-SR04 ultrasonic sensor and the Serial port. In a loop, it triggers the sensor, measures the time-of-flight of the sound pulse using the *pulseIn* function, and then calculates the distance in centimeters using single-precision floating-point arithmetic (*Float32Div*) before printing the result to the Serial Monitor.




```
program TestHCSR04;

{
  Test Ultrasonic Sensor HC-SR04

  Ultrasonic sensor Pins:
    VCC: +5VDC
    Trig : Trigger (INPUT) - Pin11
    Echo: Echo (OUTPUT) - Pin 12
    GND: GND

  ported 2025 to Pascal by @ackarwow
  based on sketch by Rui Santos (https://randomnerdtutorials.com)
}

{$IFDEF AVRPascal}
  {$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}

{$mode objfpc}

uses
  timer, defs, digital, hardwareserial, strings, pulse, float32;

Const
  TrigPin=11; // Trigger
  EchoPin=12; // Echo

  RawTwo: TRawFloat32 = $40000000;
  Raw_29_1: TRawFloat32 = $41E8CCCD; //29.1

var
   Res: UInt32;
   B: Uint8;
   Buff: array[0..30] of char;
   BuffPtr: PChar;
   duration, cm: TRawFloat32;
begin
  //Define inputs and outputs
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  //Serial Port begin
  Serial.Start(9600);

  //Main loop
  while True do
  begin
    // The sensor is triggered by a HIGH pulse of 10 or more microseconds.
    // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
    digitalWrite(trigPin, LOW);
    delayMicroseconds(5);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    // Read the signal from the sensor: a HIGH pulse whose
    // duration is the time (in microseconds) from the sending
    // of the ping to the reception of its echo off of an object.
    Res:=pulseIn(echoPin, HIGH);

    FillChar(Buff, SizeOf(Buff), #0);
    BuffPtr:=UInt32ToStr(Res, Buff,10);
    Serial.Write(' Res: ');
    Serial.Writeln(BuffPtr);

    duration := IntToFloat32(Res);

    // Convert the time [microseconds] into a distance [cm]
    // cm = (duration/2) / 29.1
```

```
    cm:=Float32Div(Float32Div(duration, RAWTwo), Raw_29_1);

    FillChar(Buff, SizeOf(Buff), #0);
    b:=Float32ToStr(Buff,30,2,cm);
    if b>0 then
    begin
      Serial.Write(' Cm: ');
      Serial.Writeln(Buff);
    end;

    delay(250);
  end;
end.
```