velleman®

VMA501

DIY STARTER KIT FOR ARDUINO®

USER MANUAL

USER MANUAL

1. Introduction

To all residents of the European Union

Important environmental information about this product

This symbol on the device or the package indicates that disposal of the device after its lifecycle could harm the environment. Do not dispose of the unit (or batteries) as unsorted municipal waste; it should be taken to a specialized company for recycling. This device should be returned to your distributor or to a local recycling service. Respect the local environmental rules.

If in doubt, contact your local waste disposal authorities.

Thank you for choosing Velleman®! Please read the manual thoroughly before bringing this device into service. If the device was damaged in transit, do not install or use it and contact your dealer.

2. Safety Instructions

- This device can be used by children aged from 8 years and above, and persons with reduced physical, sensory or mental capabilities or lack of experience and knowledge if they have been given supervision or instruction concerning the use of the device in a safe way and understand the hazards involved. Children shall not play with the device. Cleaning and user maintenance shall not be made by children without supervision.

- Indoor use only.
  Keep away from rain, moisture, splashing and dripping liquids.

3. General Guidelines

- Refer to the Velleman® Service and Quality Warranty on the last pages of this manual.
- Familiarise yourself with the functions of the device before actually using it.
- All modifications of the device are forbidden for safety reasons. Damage caused by user modifications to the device is not covered by the warranty.
- Only use the device for its intended purpose. Using the device in an unauthorised way will void the warranty.
- Damage caused by disregard of certain guidelines in this manual is not covered by the warranty and the dealer will not accept responsibility for any ensuing defects or problems.
- Nor Velleman nv nor its dealers can be held responsible for any damage (extraordinary, incidental or indirect) – of any nature (financial, physical…) arising from the possession, use or failure of this product.
- Due to constant product improvements, the actual product appearance might differ from the shown images.
- Product images are for illustrative purposes only.
- Do not switch the device on immediately after it has been exposed to changes in temperature. Protect the device against damage by leaving it switched off until it has reached room temperature.
- Keep this manual for future reference.

4. What is Arduino®

Arduino® is an open-source prototyping platform based in easy-to-use hardware and software. Arduino® boards are able to read inputs – light-on sensor, a finger on a button or a Twitter message – and turn it into an output – activating of a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so, you use the Arduino programming language (based on Wiring) and the Arduino® software IDE (based on Processing).

Surf to www.arduino.cc and www.arduino.org for more information.

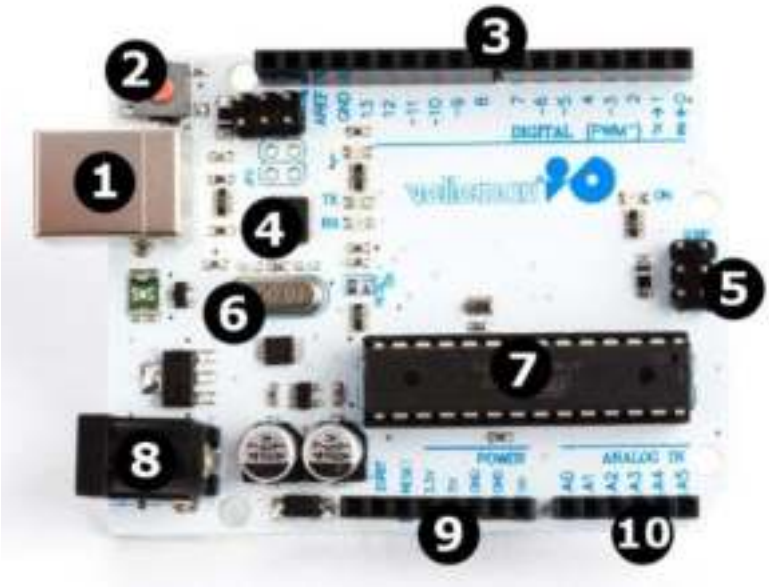
5. Contents

- 1 x ATmega328 UNO DEVELOPMENT BOARD (VMA100)
- 15 x LED (different colors)
- 8 x 220 Ω resistor (RA220E0)
- 5 x 1k resistor (RA1K0)
- 5 x 10k resistor (RA10K0)
- 1 x 830 hole breadboard
- 1 x RGB LED module (VMA318)
- 4 x 4-pin Key switch
- 1 x active buzzer (VMA319)
- 1 x passive buzzer
- 1 x 1838 IR infrared 37.9 kHz receiver (VMA317)
- 1 x infrared remote control
- 1 x infrared sensor diode
- 1 x LM35 temperature sensor (LM35DZ)
- 2 x ball tilt switch (similar to MERS4 & MERS5)
- 3 x photosensitive transistor
- 1 x 74HC595 shift register (HC595)
- 1 x battery holder for 6 AA batteries (similar to BH363B)
- 1 x 8*8 LED matrix display
- 1 x single-digit 7-segment LED display
- 1 x 4-digit 7-segment LED display
- 30 x breadboard jumper wire
- 1 x USB cable

6. The ATmega328

VMA100

The VMA100 (Arduino® Uno compatible) is a microcontroller board based on the ATmega328. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analogue inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller. Connect it to a computer with a USB cable or power it with an AC-to-DC adapter or battery to get started.



1	USB interface
2	reset button
3	digital I/O
4	Atmel mega16U2
5	ICSP

6	16 MHz clock
7	Atmel mega328p (DIL)
8	7-12 VDC power input
9	power and ground pins
10	analogue input pins

microcontroller ATmega328
operating voltage ..... 5 VDC
input voltage (recommended) ..... 7-12 VDC
input voltage (limits) ..... 6-20 VDC
digital I/O pins ..... 14 (of which 6 provide PWM output)
analogue input pins 6
DC current per I/O pin ..... 40 mA
DC current for 3.3 V pin ..... 50 mA
flash memory ..... 32 kB (ATmega328) of which 0.5 kB used by bootloader
SRAM ..... 2 kB (ATmega328)
EEPROM ..... 1 kB (ATmega328)
clock speed ..... 16 MHz
dimensions
length ..... 68.6 mm
width ..... 53.4 mm
weight ..... 25 g

7. Operation

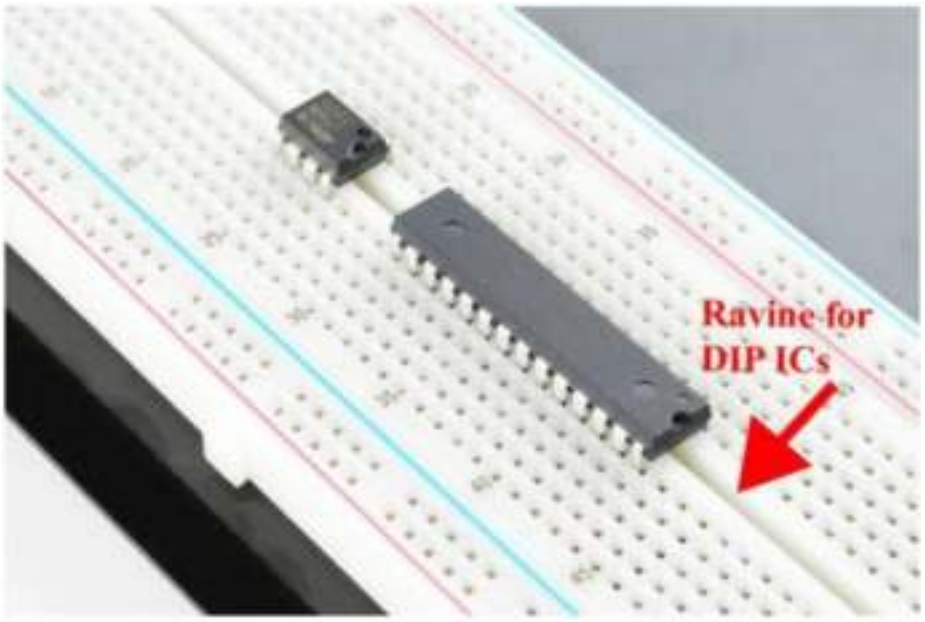
7.1 The Breadboard

Breadboards are one of the most fundamental pieces when learning how to build circuits. In this tutorial, we will introduce you to what breadboards are and how they work.

Let us look at a larger, more typical breadboard. Aside from the horizontal rows, breadboards have what are called **power rails** that run vertically along the sides.

Power rails.

Chips have legs that come out of both sides and fit perfectly over the **ravine**. Since each leg on the IC is unique, we do not want both sides to be connected to each other. That is where the separation in the middle of the board comes in handy. Thus, we can connect components to each side of the IC without interfering with the functionality of the leg on the opposite side.



Ravine.

7.2 A Blinking LED

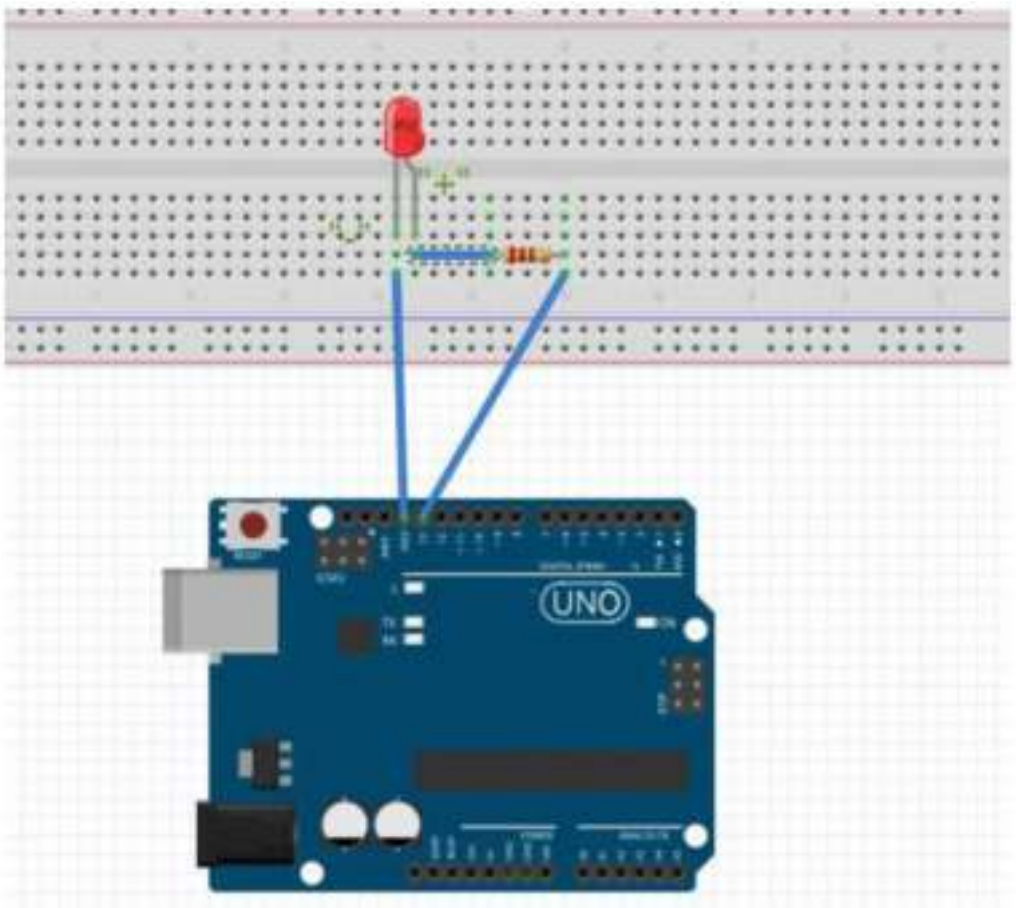
Let's start with a simple experiment. We are going to connect an LED to one of the digital pins rather than using LED13, which is soldered to the board.

Required Hardware

- 1 x red M5 LED
- 1 x 220 Ω resistor
- 1 x breadboard
- jumper wires as needed

Follow the diagram below. We are using digital pin 10, and connecting the LED to a 220 Ω resistor to avoid high-current damaging the LED.

Connection



Programming Code

```
*******code begin*******
int ledPin = 10; // define digital pin 10.
void setup()
{
pinMode(ledPin, OUTPUT);// define pin with LED connected as output.
}
void loop()
{
digitalWrite(ledPin, HIGH); // set the LED on.
delay(1000); // wait for a second.
digitalWrite(ledPin, LOW); // set the LED off.
delay(1000); // wait for a second
}
*******code end*******
```

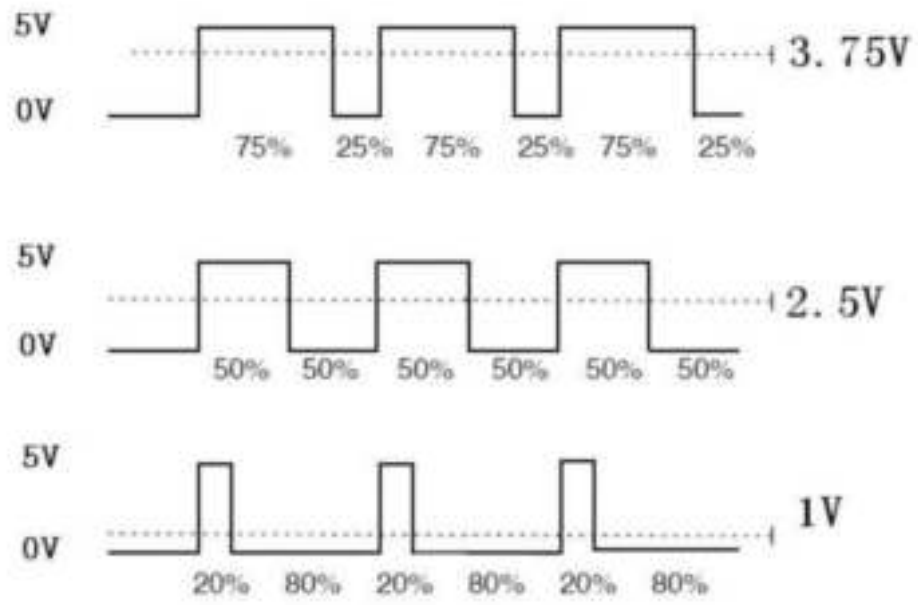
Result

After programming, you will see the LED connected to pin 10 blinking, with an interval of approximately one second. Congratulations, the experiment is now successfully completed!

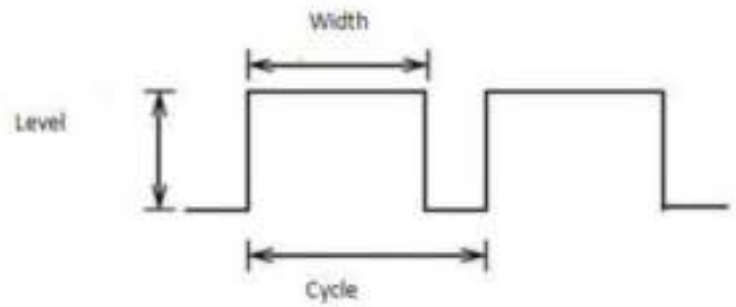
7.3 PWM Gradational LED

PWM (Pulse Width Modulation) is a technique used to encode analogue signal levels into digital ones. A computer cannot output analogue voltage but only digital voltage values. So, we will be using a high-resolution counter to encode a specific analogue signal level by modulating the duty cycle of PWM. The PWM signal is also digitalized because in any given moment, fully on DC power is either 5 V (on) of 0 V (off). The voltage or current is fed to the analogue load (the device using the power) by repeated pulse sequence being on or off. Being on, the current is fed to the load; being off, it is not. With the adequate bandwidth, any analogue value can be encoded using PWM. The output voltage value is calculated via the on and off time.

output voltage = (turn on time/pulse time) * maximum voltage value



PWM has many applications: lamp brightness regulation, motor speed regulation, sound making, etc. The following are the basic parameters of PWM:

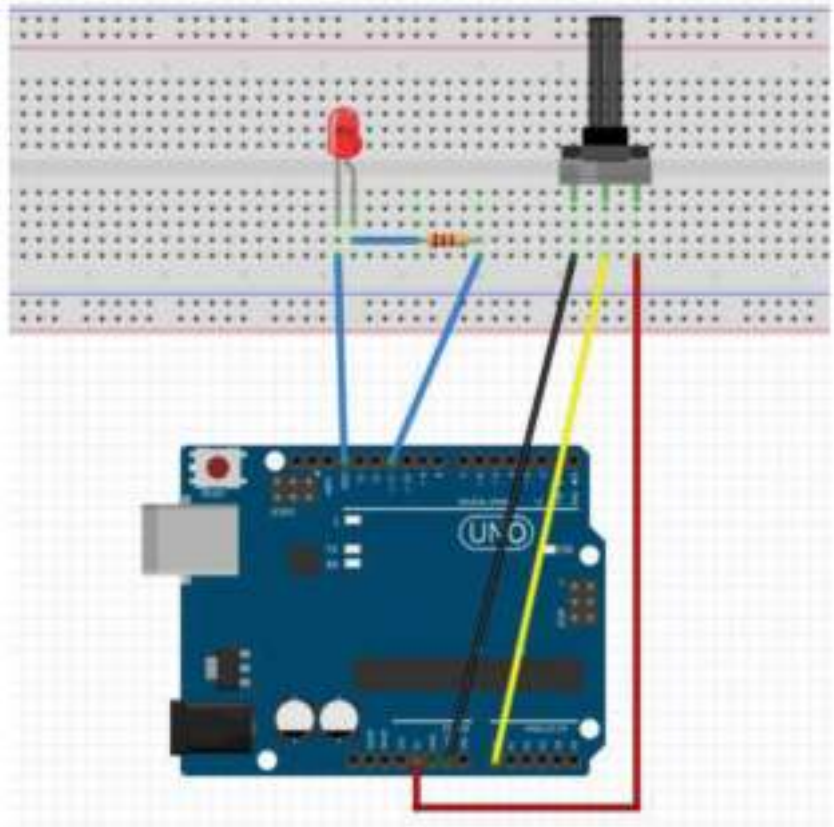


There are six PQM interfaces on Arduino®, namely digital pin, 3, 5, 6, 9, 10 and 11. In this experiment, we will be using a potentiometer to control the LED brightness.

Required Hardware

- 1 x variable resistor
- 1 x red M5 LED
- 1 x 220 Ω resistor
- 1 x breadboard
- jumper wires as needed

Connection



Programming Code

```
*******code begin*******
int potpin=0;// initialize analog pin 0
int ledpin=11;//initialize digital pin 11（PWM output）
int val=0;// Temporarily store variables' value from the sensor
void setup()
{
pinMode(ledpin,OUTPUT);// define digital pin 11 as "output"
Serial.begin(9600);// set baud rate at 9600
// attention: for analog ports, they are automatically set up as "input"
}
void loop()
{
val=analogRead(potpin);// read the analog value from the sensor and assign it to val
Serial.println(val);// display value of val
analogWrite(ledpin,val/4);// turn on LED and set up brightness（maximum output of
PWM is 255）
delay(10);// wait for 0.01 second
}
*******code end*******
```

In this code, we are using the analogWrite (PWM interface, analogue value) function. We will read the analogue value of the potentiometer and assign the value to PWM port, so there will be corresponding change to the brightness of the LED. One final part will be displaying the analogue value on the screen. You can consider this as the **analogue value reading** project adding the PWM analogue value assigning part.

Result

After programming, rotate the potentiometer knob to see changes of the displaying value. Also, note the obvious change of brightness on the breadboard.

7.4 RGB LED Module

The RGB LED module is a full-colour LED allows for cool lighting effects.

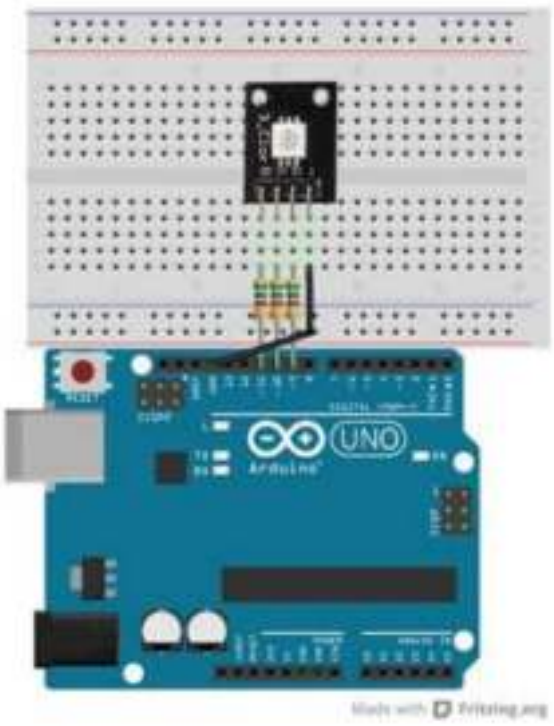
Specifications

red Vf ..... 1.8 to 2.1 V
green Vf ..... 3.0 to 3.2 V
blue Vf ..... 3.0 to 3.2 V
red colour ..... 620-625 nm
green colour ..... 520-525 nm
blue colour ..... 465-470 nm
red brightness @ 20 mA ..... 600-800 mcd
green brightness @ 20 mA ..... 800-1000 mcd
blue brightness @ 20 mA ..... 1500-2000 mcd

Pin Layout

Pin Name	Description
R	red
G	green
B	blue
-	GND

Connection



Programming Code

```
********Code begin********
//RGB LED pins
int ledDigitalOne[] = {10, 11, 9}; //the three digital pins of the digital LED
                                   //10 = redPin, 11 = greenPin, 9 = bluePin

const boolean ON = HIGH;        //Define on as LOW (this is because we use a common
                                  //Anode RGB LED (common pin is connected to +5
volts)
const boolean OFF = LOW;     //Define off as HIGH

//Predefined Colors
const boolean RED[] = {ON, OFF, OFF};
const boolean GREEN[] = {OFF, ON, OFF};
const boolean BLUE[] = {OFF, OFF, ON};
const boolean YELLOW[] = {ON, ON, OFF};
const boolean CYAN[] = {OFF, ON, ON};
const boolean MAGENTA[] = {ON, OFF, ON};
const boolean WHITE[] = {ON, ON, ON};
const boolean BLACK[] = {OFF, OFF, OFF};

//An Array that stores the predefined colors (allows us to later randomly display a color)
const boolean* COLORS[] = {RED, GREEN, BLUE, YELLOW, CYAN, MAGENTA, WHITE,
BLACK};

void setup(){
  for(int i = 0; i < 3; i++){
    pinMode(ledDigitalOne[i], OUTPUT);     //Set the three LED pins as outputs
  }
}

void loop(){

/* Example - 1 Set a color
    Set the three LEDs to any predefined color
*/
    setColor(ledDigitalOne, YELLOW);       //Set the color of LED one

/* Example - 2 Go through Random Colors
  Set the LEDs to a random color
*/
    //randomColor();

}
```

```
void randomColor(){
   int rand = random(0, sizeof(COLORS) / 2);    //get a random number within the range
of colors
   setColor(ledDigitalOne, COLORS[rand]);    //Set the color of led one to a random
color
   delay(1000);
}

/* Sets an led to any color
     led - a three element array defining the three color pins (led[0] = redPin, led[1] =
greenPin, led[2] = bluePin)
     color - a three element boolean array (color[0] = red value (LOW = on, HIGH = off),
color[1] = green value, color[2] =blue value)
*/
void setColor(int* led, boolean* color){
 for(int i = 0; i < 3; i++){
    digitalWrite(led[i], color[i]);
 }
}

/* A version of setColor that allows for using const boolean colors
*/
void setColor(int* led, const boolean* color){
   boolean tempColor[] = {color[0], color[1], color[2]};

   setColor(led, tempColor);
}
********Code End********
```

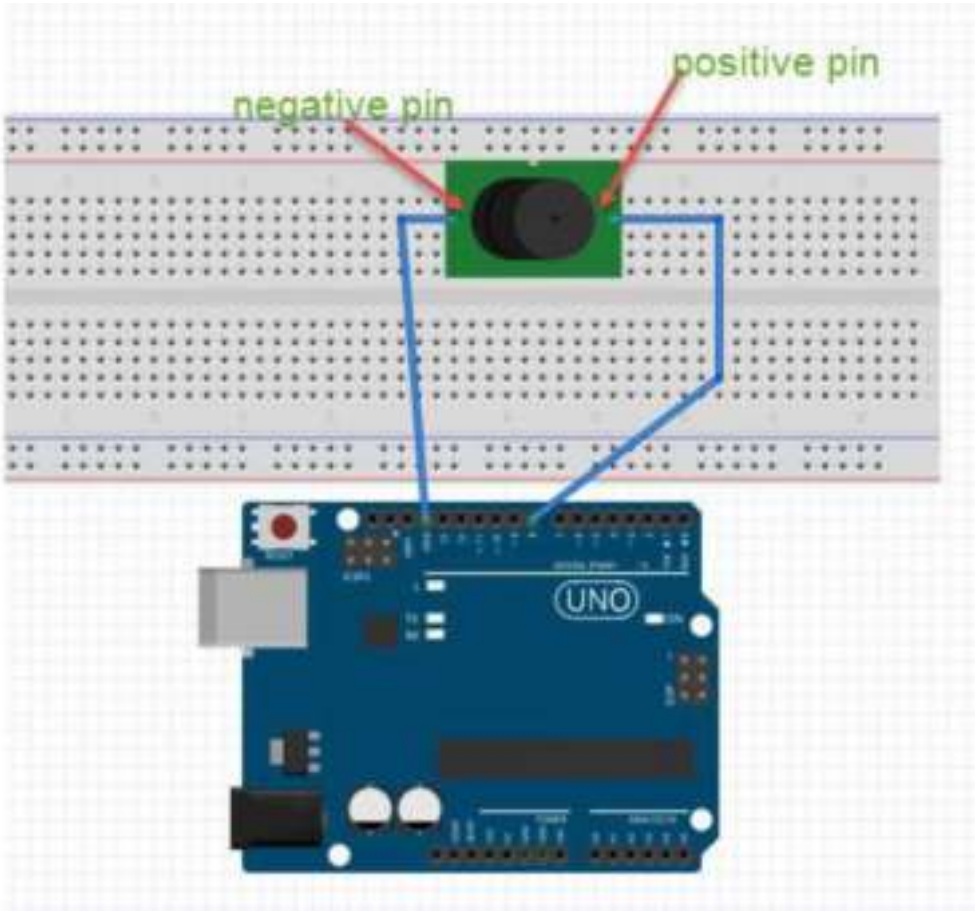
7.5 The Active Buzzer

An active buzzer is widely used on computers, printers, alarms, etc. as a sound-making element. It has an inner vibration source. Simply connect it with a 5 V power supply to make it buzz constantly.

Required Hardware

- 1 x buzzer
- 1 x key
- 1 x breadboard
- jumper wires as needed

Connection



Programming Code

```
*******code begin*******
//////////////////////////////////////////////////////////////////
int buzzer=8;// initialize digital IO pin that controls the buzzer
void setup()
{
  pinMode(buzzer,OUTPUT);// set pin mode as "output"
}
void loop()
{
digitalWrite(buzzer, HIGH); // produce sound
}
//////////////////////////////////////////////////////////////////
*******code End*******
```

Result

After programming, the buzzer should ring.

7.6 The Photosensitive Transistor

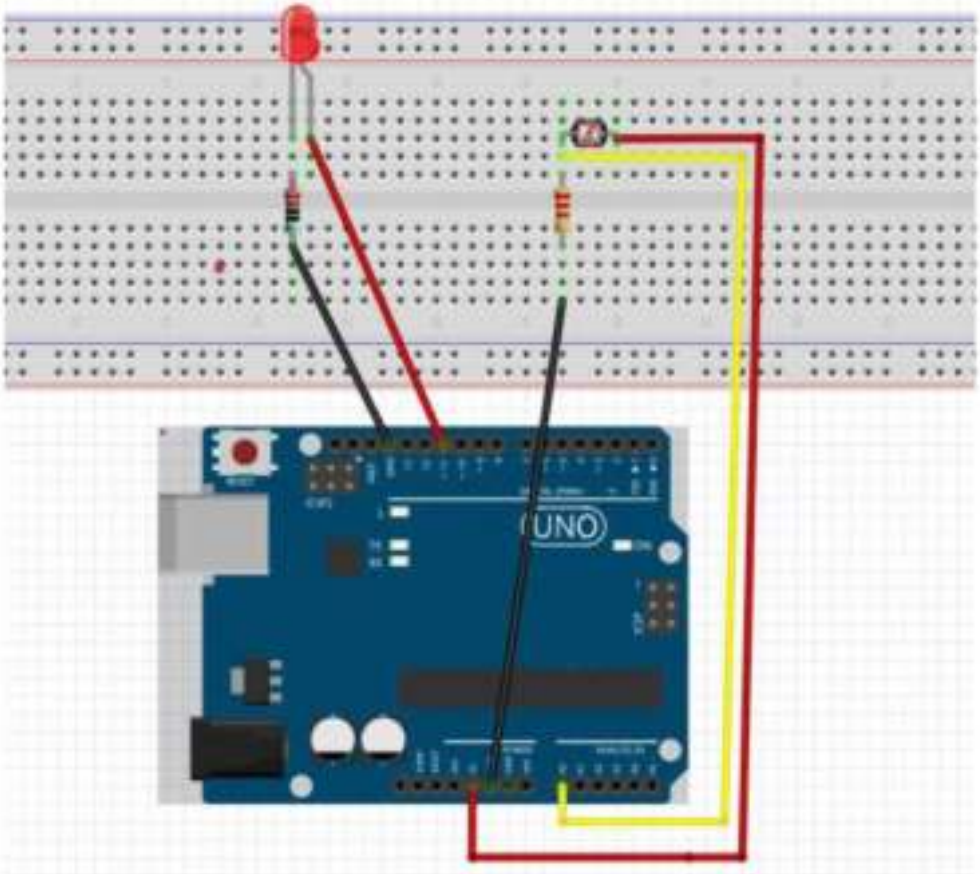
A phototransistor is a transistor whose resistance varies according to different light strengths. It is based on the photoelectric effect of a semiconductor. If the incident light is intense, the resistance reduces; if the incident light is weak, the resistant increases. A phototransistor is commonly applied in the measurement of light, light control and photovoltaic conversion.

Let's start with a relative simple experiment. The phototransistor is an element that changes its resistance as light strength changes. Refer to the PWM experiment, replacing the potentiometer with a phototransistor. When there is a change in light strength, there will be a corresponding change on the LED.

Required Hardware

- 1 x phototransistor
- 1 x red M5 LED
- 1 x 1 KΩ resistor
- 1 x 220 Ω resistor
- 1 x breadboard
- jumper wires as needed

Connection



Programming Code

```
*******code begin*******

//////////////////////////////////////////////////////////
int potpin=0;// initialize analog pin 0, connected with photovaristor
int ledpin=11;// initialize digital pin 11, output regulating the brightness of LED
int val=0;// initialize variable va
void setup()
{
pinMode(ledpin,OUTPUT);// set digital pin 11 as "output"
Serial.begin(9600);// set baud rate at "9600"
}
void loop()
{
val=analogRead(potpin);// read the analog value of the sensor and assign it to val
Serial.println(val);// display the value of val
analogWrite(ledpin,val);// turn on the LED and set up brightness（maximum output
value 255）
delay(10);// wait for 0.01
}
//////////////////////////////////////////////////////////
*******code End*******
```

Result

After programming, change the light strength around the phototransistor and observe the LED changing!

7.7 The Flame Sensor

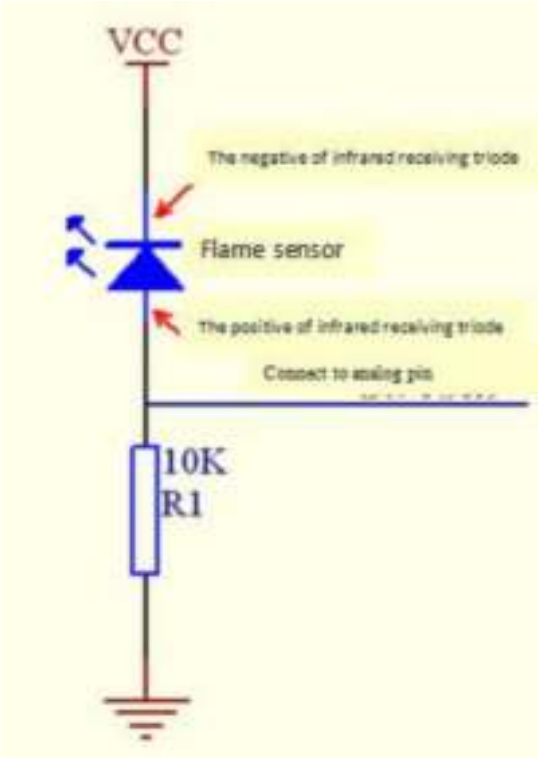
A flame sensor (IR receiving diode) is specifically used on robots to find the fire source. This sensor is highly sensitive to flames.

A flame sensor has a specifically designed IR tube to detect fire. The brightness of the flames will then be converted to a fluctuating level signal. The signals are the input into the central processor.

Required Hardware

- 1 x flame sensor
- 1 x buzzer
- 1 x 10KΩ resistor
- 1 x breadboard
- jumper wires as needed

Connection

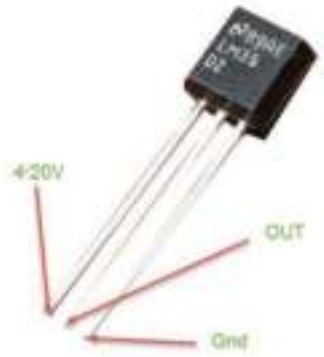


Connect the negative to the 5 V pin and the positive to the resistor. Connect the other end of the resistor to GND. Connect one end of a jumper wire to a clip, which is electrically connected to sensor positive, the other end to the analogue pin.

Programming Code

```
*******code End*******
int flame=0;// select analog pin 0 for the sensor
int Beep=9;// select digital pin 9 for the buzzer
int val=0;// initialize variable
void setup()
{
  pinMode(Beep,OUTPUT);// set LED pin as "output"
pinMode(flame,INPUT);// set buzzer pin as "input"
Serial.begin(9600);// set baud rate at "9600"
}
void loop()
{
 val=analogRead(flame);// read the analog value of the sensor
 Serial.println(val);// output and display the analog value
 if(val>=600)// when the analog value is larger than 600, the buzzer will buzz
 {
   digitalWrite(Beep,HIGH);
   }else
   {
      digitalWrite(Beep,LOW);
    }
   delay(500);
}
*******code End*******
```

7.8 The LM35 Temperature Sensor

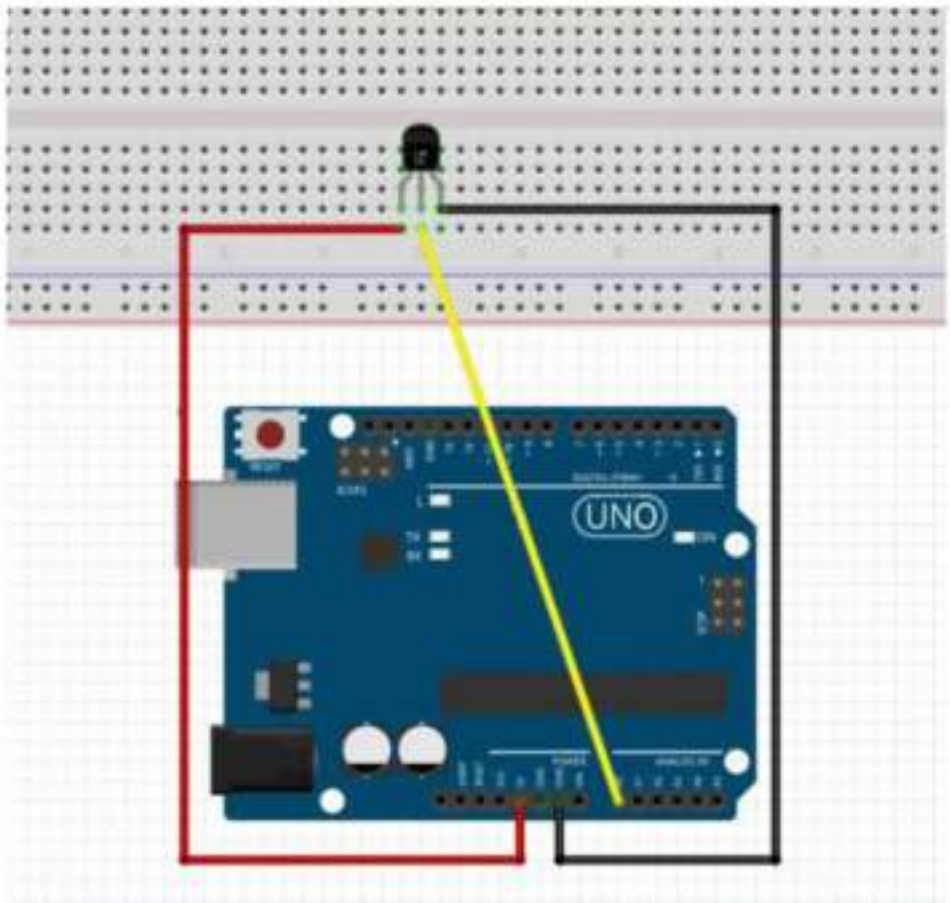


The LM35 is a common and easy-to-use temperature sensor. It does not require other hardware, you just need an analogue port to make it work. The difficulty lies in compiling the code to convert the analogue value it reads to Celsius temperature.

Required Hardware

- 1 x LM35 sensor
- 1 x breadboard
- jumper wires as needed

Connection



Programming Code

```
*******code begin*******
int potPin = 0; // initialize analog pin 0 for LM35 temperature sensor
void setup()
{
Serial.begin(9600);// set baud rate at"9600"
}
void loop()
{
int val;// define variable
int dat;// define variable
val=analogRead(0);// read the analog value of the sensor and assign it to val
dat=(125*val)>>8;// temperature calculation formula
Serial.print("Tep:");// output and display characters beginning with Tep
Serial.print(dat);// output and display value of dat
Serial.println("C");// display "C" characters
delay(500);// wait for 0.5 second

 }
 *******code End*******
```

Result

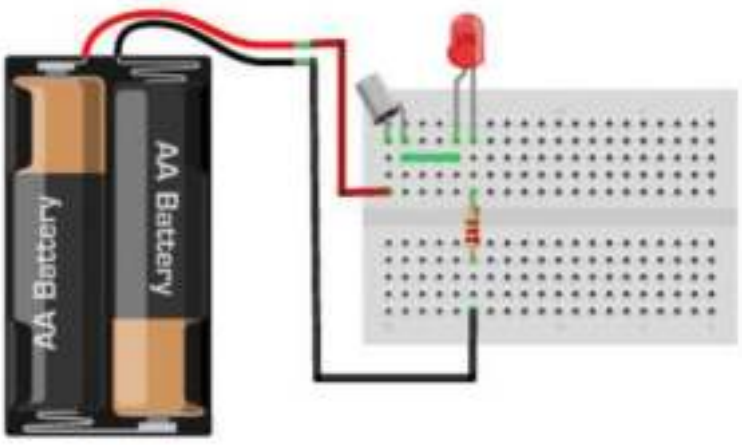
After programming, open the monitoring window to see the current temperature.

7.9 The Tilt Sensor Switch

A tilt sensor will detect orientation and inclination. They are small, low power and easy-to-use. If used properly, they will not wear out. Their simplicity makes them popular for toys, gadgets and other appliances. They are referred to as **mercury**, **tilt** or **rolling ball** switches.

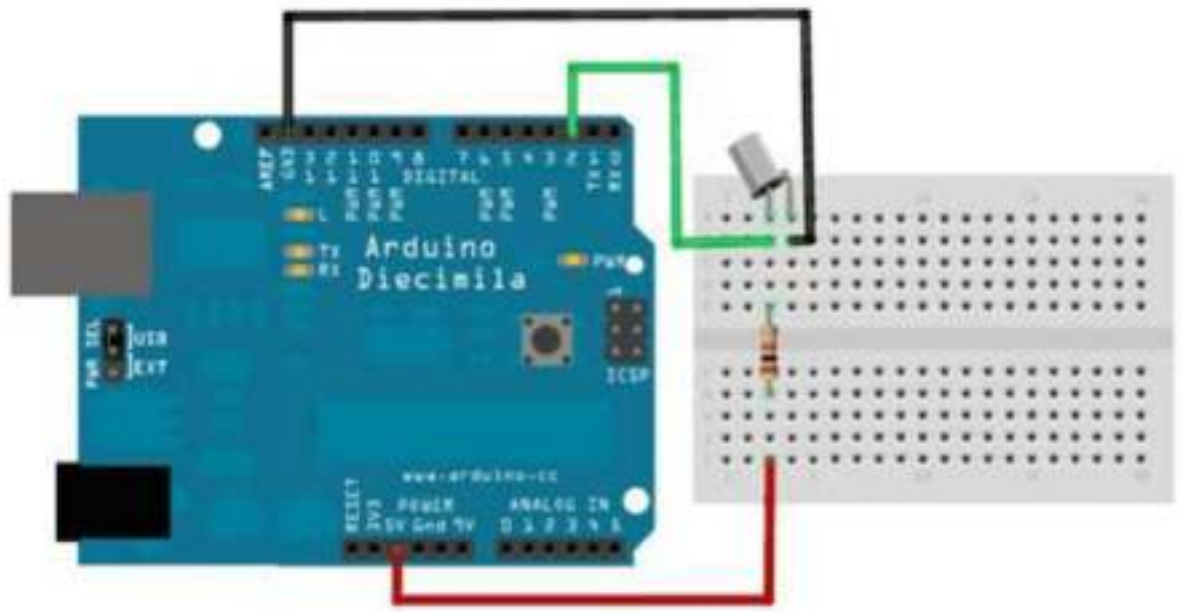
The Simple Tilt-Activated LED

This is the most basic connection of a tilt switch, but can be a handy while one is learning about them. Simply connect in series with an LED, resistor and battery.



Reading the Switch State with a Microcontroller

The layout below shows a 10K pull-up resistor. The code states the built-in pull-up resistor that you can turn on by setting an input pin to the high output. If you use the internal pull-up you can skip the external one.



Programming Code

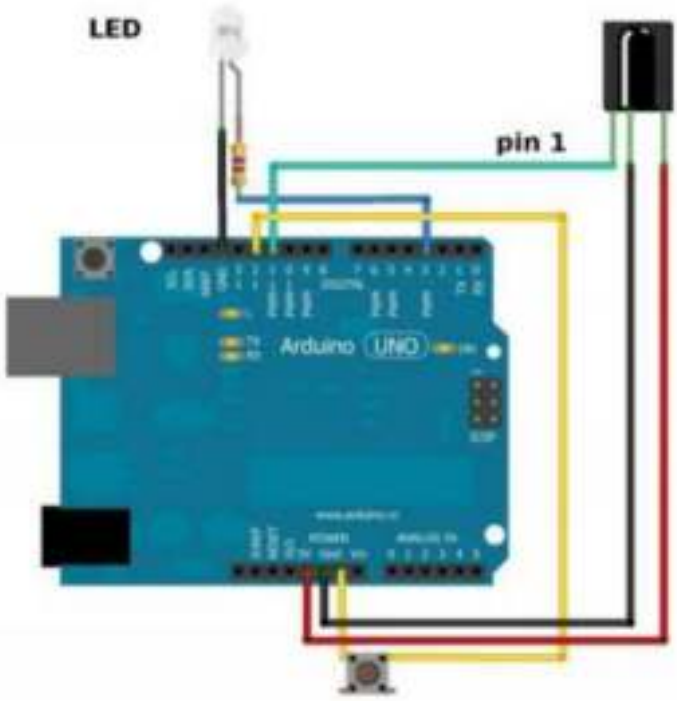
```
*******code begin*******
int inPin = 2;              // the number of the input pin
int outPin = 13;            // the number of the output pin

int LEDstate = HIGH;        // the current state of the output pin

int reading;                // the current reading from the input pin
int previous = LOW;         // the previous reading from the input pin

// the follow variables are long's because the time, measured in milliseconds,
// will quickly become a bigger number than can be stored in an int.
long time = 0;              // the last time the output pin was toggled
long debounce = 50;         // the debounce time, increase if the output flickers

void setup()
{
  pinMode(inPin, INPUT);
  digitalWrite(inPin, HIGH);     // turn on the built in pull-up resistor
  pinMode(outPin, OUTPUT);
}

void loop()
{
  int switchstate;

  reading = digitalRead(inPin);
```

```
  // If the switch changed, due to bounce or pressing...
  if (reading != previous) {
    // reset the debouncing timer
    time = millis();
  }

  if ((millis() - time) > debounce) {
     // whatever the switch is at, its been there for a long time
     // so lets settle on it!
     switchstate = reading;

     // Now invert the output on the pin13 LED
    if (switchstate == HIGH)
      LEDstate = LOW;
    else
      LEDstate = HIGH;
  }
  digitalWrite(outPin, LEDstate);

  // Save the last reading so we keep a running tally
  previous = reading;
}
*******code End*******
```

7.10 The IR Receiver

Infrared communication is a common, inexpensive and easy-to-use wireless communication technology. IR light is very similar to visible light, except that it has a slightly longer wavelength. This means that IR is undetectable to the human eye – perfect for wireless communication. Example: When you hit a button on your TV remote, an IR LED repeatedly turns on and off, 38000 times a second, to transmit information to an IR photo sensor on your TV.

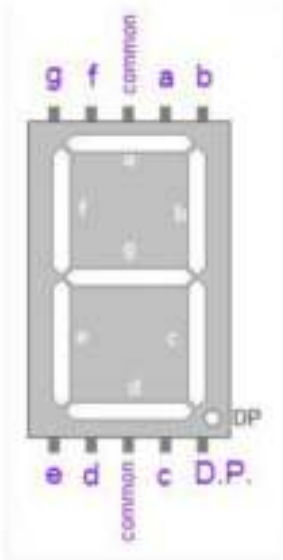
Connection



Programming Code

```
*******code begin*******
#include <IRremote.h>

int RECV_PIN = 11;

IRrecv irrecv(RECV_PIN);

decode_results results;
void setup()
{
  Serial.begin(9600);
  irrecv.enableIRIn(); // Start the receiver
}
void loop() {
  if (irrecv.decode(&results)) {
    Serial.println(results.value, HEX);
    irrecv.resume(); // Receive the next value
  }
}

*******code End*******
```

7.11 One-Digit Seven-Segment Display

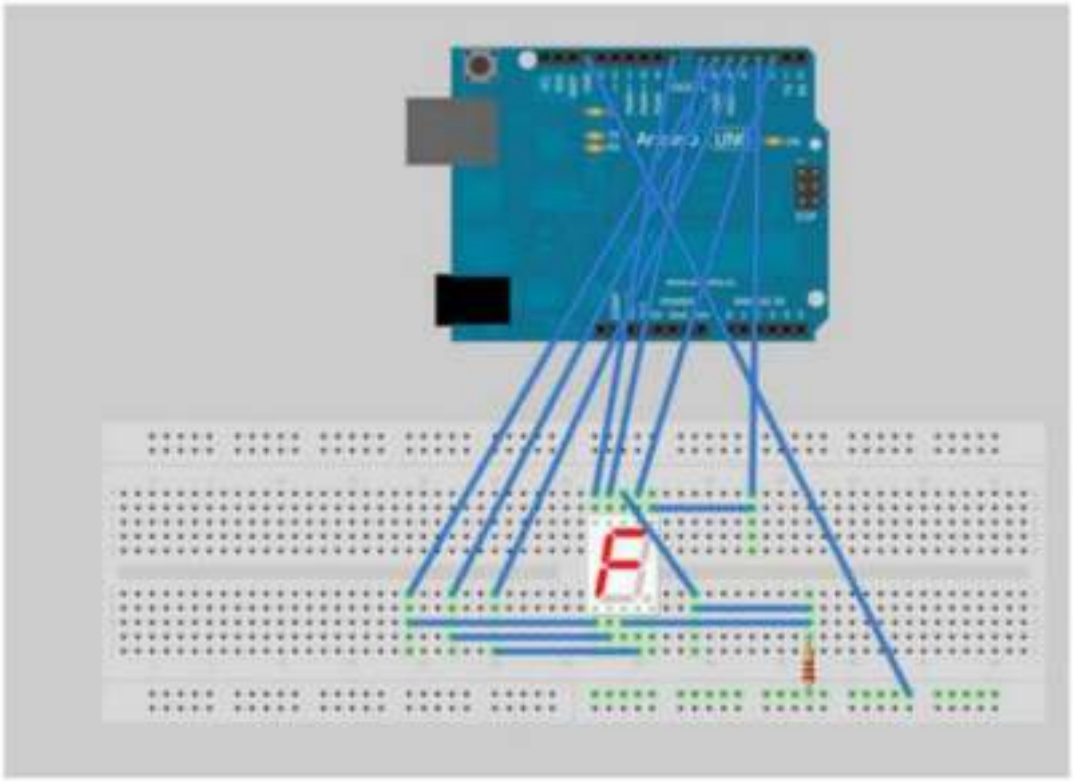


LED segment displays are common for displaying numerical information. They are widely applied on displays of ovens, washing machines, etc. the LED segment display is a semiconductor light-emitting device. Its basic unit is an LED (light-emitting diode). Segment displays can be divided into 7-segment and 8-segment displays.

According to the wiring method, LED segment displays can be divided into displays with common anode and displays with common cathode. Common anode displays refer to displays that combine all the anodes of the LED units into one common anode (COM).

For the common anode display, connect the common anode (COM) to +5 V. When the cathode level of a certain segment is low, the segment is on; when the cathode level of a certain segment is high, the segment is off. For the common cathode display, connect the common cathode (COM) to GND. When the anode level of a certain segment is high, the segment is on; when the anode level of a certain segment is low, the segment is off.

Connection

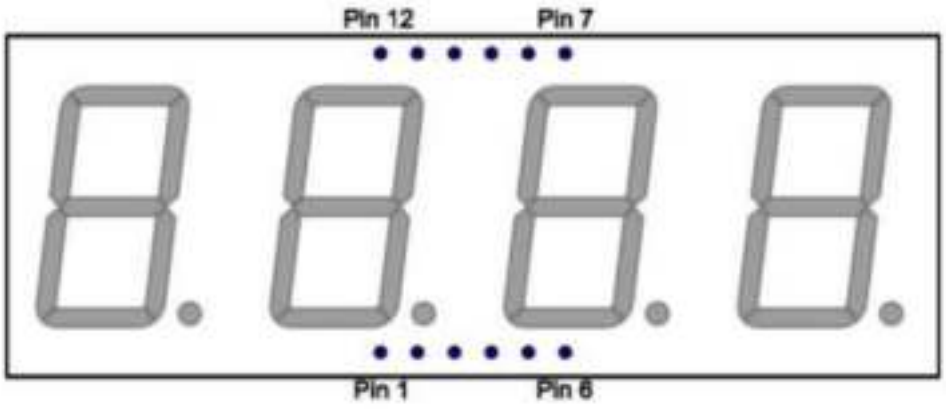


Programming Code

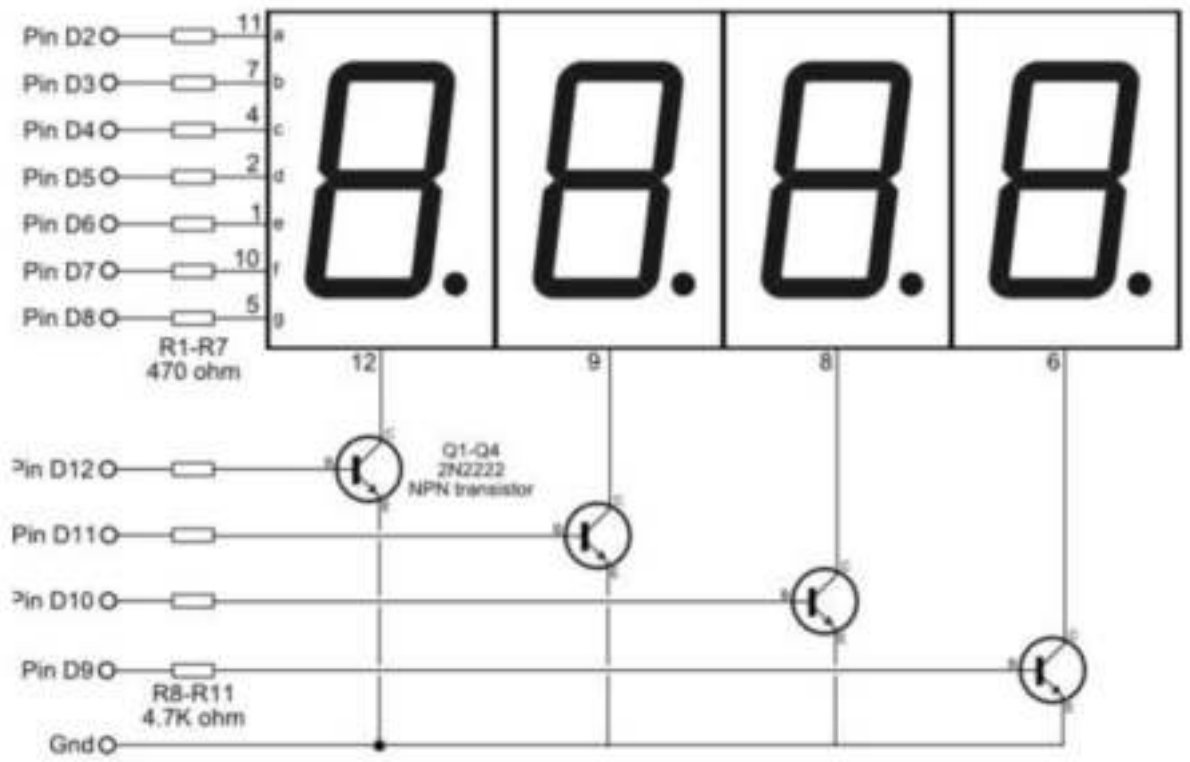
```
*******code begin*******
// Define the LED digit patters, from 0 - 9
// Note that these patterns are for common cathode displays
// For common anode displays, change the 1's to 0's and 0's to 1's
// 1 = LED on, 0 = LED off, in this order:
//                                                  Arduino pin: 2,3,4,5,6,7,8
byte seven_seg_digits[10][7] = { { 1,1,1,1,1,1,0 },   // = 0

{ 0,1,1,0,0,0,0 },   // = 1

{ 1,1,0,1,1,0,1 },   // = 2

{ 1,1,1,1,0,0,1 },   // = 3

{ 0,1,1,0,0,1,1 },   // = 4

{ 1,0,1,1,0,1,1 },   // = 5
```

```
{ 1,0,1,1,1,1,1 },   // = 6

{ 1,1,1,0,0,0,0 },   // = 7

{ 1,1,1,1,1,1,1 },   // = 8

{ 1,1,1,0,0,1,1 }    // = 9
                                                    };

void setup() {
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  writeDot(0);   // start with the "dot" off
}

void writeDot(byte dot) {
  digitalWrite(9, dot);
}
void sevenSegWrite(byte digit) {
  byte pin = 2;
  for (byte segCount = 0; segCount < 7; ++segCount) {
    digitalWrite(pin, seven_seg_digits[digit][segCount]);
    ++pin;
  }
}

void loop() {
  for (byte count = 10; count > 0; --count) {
   delay(1000);
   sevenSegWrite(count - 1);
  }
  delay(4000);
}
*******code End*******
```

7.12 Four-Digit Seven-Segment Display



Connection



Programming Code

This is an example of how to drive a 7-segment LED display without the use of current limiting resistors.

This technique is very common but requires some knowledge of electronics – you do run the risk of dumping too much current through the segments and burning out parts of the display. If you use the stock code you should be ok, but be careful editing the brightness values.

This code should work with all colours (red, blue, yellow, green) but the brightness will vary from one colour to the next because the forward voltage drop of each colour is different. This code was written and calibrated for the red colour.

```
*******code Begin*******

int digit1 = 11; //PWM Display pin 1
int digit2 = 10; //PWM Display pin 2
int digit3 = 9; //PWM Display pin 6
int digit4 = 6; //PWM Display pin 8
```

```
//Pin mapping from Arduino to the ATmega DIP28 if you need it
//http://www.arduino.cc/en/Hacking/PinMapping
int segA = A1; //Display pin 14
int segB = 3; //Display pin 16
int segC = 4; //Display pin 13
int segD = 5; //Display pin 3
int segE = A0; //Display pin 5
int segF = 7; //Display pin 11
int segG = 8; //Display pin 15

void setup() {
  pinMode(segA, OUTPUT);
  pinMode(segB, OUTPUT);
  pinMode(segC, OUTPUT);
  pinMode(segD, OUTPUT);
  pinMode(segE, OUTPUT);
  pinMode(segF, OUTPUT);
  pinMode(segG, OUTPUT);

  pinMode(digit1, OUTPUT);
  pinMode(digit2, OUTPUT);
  pinMode(digit3, OUTPUT);
  pinMode(digit4, OUTPUT);

  pinMode(13, OUTPUT);
}

void loop() {

  //long startTime = millis();

  displayNumber(millis()/1000);

  //while( (millis() - startTime) < 2000) {
  //displayNumber(1217);
  //}
  //delay(1000);
}

//Given a number, we display 10:22
//After running through the 4 numbers, the display is left turned off

//Display brightness
```

```
//Each digit is on for a certain amount of microseconds
//Then it is off until we have reached a total of 20ms for the function call
//Let's assume each digit is on for 1000us
//If each digit is on for 1ms, there are 4 digits, so the display is off for 16ms.
//That's a ratio of 1ms to 16ms or 6.25% on time (PWM).
//Let's define a variable called brightness that varies from:
//5000 blindingly bright (15.7mA current draw per digit)
//2000 shockingly bright (11.4mA current draw per digit)
//1000 pretty bright (5.9mA)
//500 normal (3mA)
//200 dim but readable (1.4mA)
//50 dim but readable (0.56mA)
//5 dim but readable (0.31mA)
//1 dim but readable in dark (0.28mA)

void displayNumber(int toDisplay) {
#define DISPLAY_BRIGHTNESS  500

#define DIGIT_ON  HIGH
#define DIGIT_OFF  LOW

  long beginTime = millis();

  for(int digit = 4 ; digit > 0 ; digit--) {

    //Turn on a digit for a short amount of time
    switch(digit) {
    case 1:
      digitalWrite(digit1, DIGIT_ON);
      break;
    case 2:
      digitalWrite(digit2, DIGIT_ON);
      break;
    case 3:
      digitalWrite(digit3, DIGIT_ON);
      break;
    case 4:
      digitalWrite(digit4, DIGIT_ON);
      break;
    }

    //Turn on the right segments for this digit
    lightNumber(toDisplay % 10);
    toDisplay /= 10;
```

```
      delayMicroseconds(DISPLAY_BRIGHTNESS); //Display this digit for a fraction of a
second (between 1us and 5000us, 500 is pretty good)

      //Turn off all segments
      lightNumber(10);

      //Turn off all digits
      digitalWrite(digit1, DIGIT_OFF);
      digitalWrite(digit2, DIGIT_OFF);
      digitalWrite(digit3, DIGIT_OFF);
      digitalWrite(digit4, DIGIT_OFF);
   }

   while( (millis() - beginTime) < 10) ; //Wait for 20ms to pass before we paint the
display again
}

//Given a number, turns on those segments
//If number == 10, then turn off number
void lightNumber(int numberToDisplay) {

#define SEGMENT_ON    LOW
#define SEGMENT_OFF HIGH

  switch (numberToDisplay){

  case 0:
     digitalWrite(segA, SEGMENT_ON);
     digitalWrite(segB, SEGMENT_ON);
     digitalWrite(segC, SEGMENT_ON);
     digitalWrite(segD, SEGMENT_ON);
     digitalWrite(segE, SEGMENT_ON);
     digitalWrite(segF, SEGMENT_ON);
     digitalWrite(segG, SEGMENT_OFF);
     break;

  case 1:
     digitalWrite(segA, SEGMENT_OFF);
     digitalWrite(segB, SEGMENT_ON);
     digitalWrite(segC, SEGMENT_ON);
     digitalWrite(segD, SEGMENT_OFF);
     digitalWrite(segE, SEGMENT_OFF);
     digitalWrite(segF, SEGMENT_OFF);
```

```
    digitalWrite(segG, SEGMENT_OFF);
    break;

  case 2:
    digitalWrite(segA, SEGMENT_ON);
    digitalWrite(segB, SEGMENT_ON);
    digitalWrite(segC, SEGMENT_OFF);
    digitalWrite(segD, SEGMENT_ON);
    digitalWrite(segE, SEGMENT_ON);
    digitalWrite(segF, SEGMENT_OFF);
    digitalWrite(segG, SEGMENT_ON);
    break;

  case 3:
    digitalWrite(segA, SEGMENT_ON);
    digitalWrite(segB, SEGMENT_ON);
    digitalWrite(segC, SEGMENT_ON);
    digitalWrite(segD, SEGMENT_ON);
    digitalWrite(segE, SEGMENT_OFF);
    digitalWrite(segF, SEGMENT_OFF);
    digitalWrite(segG, SEGMENT_ON);
    break;

  case 4:
    digitalWrite(segA, SEGMENT_OFF);
    digitalWrite(segB, SEGMENT_ON);
    digitalWrite(segC, SEGMENT_ON);
    digitalWrite(segD, SEGMENT_OFF);
    digitalWrite(segE, SEGMENT_OFF);
    digitalWrite(segF, SEGMENT_ON);
    digitalWrite(segG, SEGMENT_ON);
    break;

  case 5:
    digitalWrite(segA, SEGMENT_ON);
    digitalWrite(segB, SEGMENT_OFF);
    digitalWrite(segC, SEGMENT_ON);
    digitalWrite(segD, SEGMENT_ON);
    digitalWrite(segE, SEGMENT_OFF);
    digitalWrite(segF, SEGMENT_ON);
    digitalWrite(segG, SEGMENT_ON);
    break;
```

```
case 6:
  digitalWrite(segA, SEGMENT_ON);
  digitalWrite(segB, SEGMENT_OFF);
  digitalWrite(segC, SEGMENT_ON);
  digitalWrite(segD, SEGMENT_ON);
  digitalWrite(segE, SEGMENT_ON);
  digitalWrite(segF, SEGMENT_ON);
  digitalWrite(segG, SEGMENT_ON);
  break;

case 7:
  digitalWrite(segA, SEGMENT_ON);
  digitalWrite(segB, SEGMENT_ON);
  digitalWrite(segC, SEGMENT_ON);
  digitalWrite(segD, SEGMENT_OFF);
  digitalWrite(segE, SEGMENT_OFF);
  digitalWrite(segF, SEGMENT_OFF);
  digitalWrite(segG, SEGMENT_OFF);
  break;

case 8:
  digitalWrite(segA, SEGMENT_ON);
  digitalWrite(segB, SEGMENT_ON);
  digitalWrite(segC, SEGMENT_ON);
  digitalWrite(segD, SEGMENT_ON);
  digitalWrite(segE, SEGMENT_ON);
  digitalWrite(segF, SEGMENT_ON);
  digitalWrite(segG, SEGMENT_ON);
  break;
case 9:
  digitalWrite(segA, SEGMENT_ON);
  digitalWrite(segB, SEGMENT_ON);
  digitalWrite(segC, SEGMENT_ON);
  digitalWrite(segD, SEGMENT_ON);
  digitalWrite(segE, SEGMENT_OFF);
  digitalWrite(segF, SEGMENT_ON);
  digitalWrite(segG, SEGMENT_ON);
  break;

case 10:
  digitalWrite(segA, SEGMENT_OFF);
  digitalWrite(segB, SEGMENT_OFF);
  digitalWrite(segC, SEGMENT_OFF);
  digitalWrite(segD, SEGMENT_OFF);
```

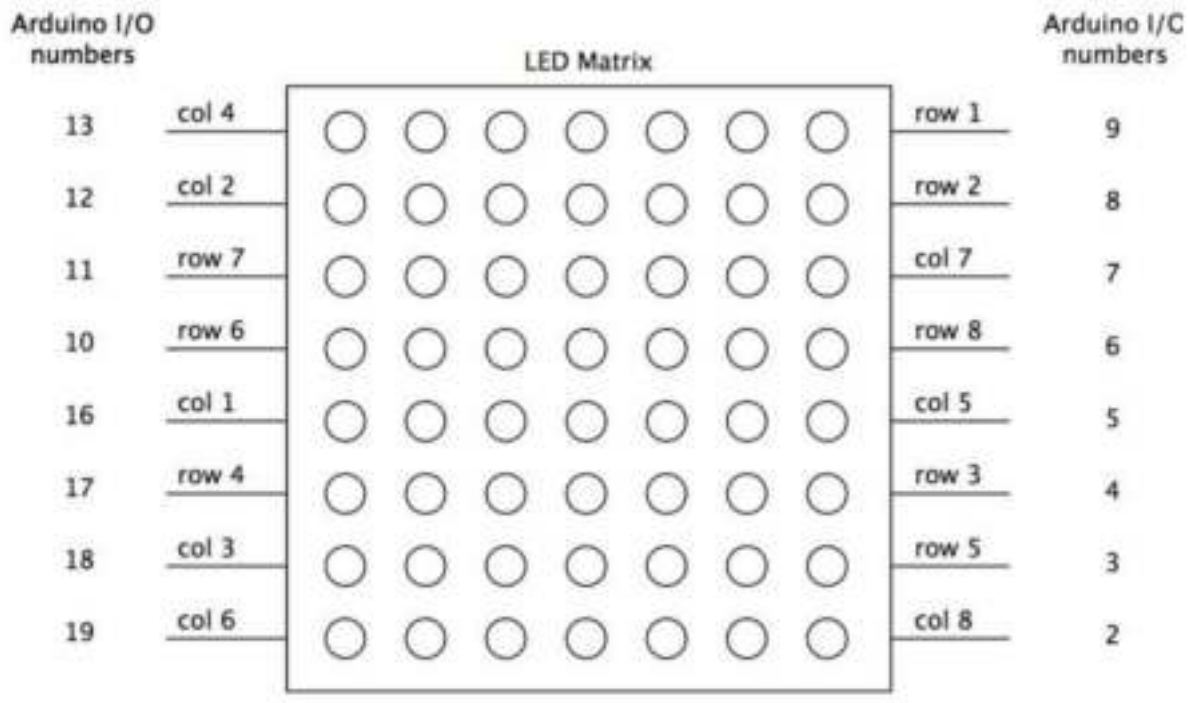
```
        digitalWrite(segE, SEGMENT_OFF);
        digitalWrite(segF, SEGMENT_OFF);
        digitalWrite(segG, SEGMENT_OFF);
        break;
    }
}
*******code End*******
```

7.13 8x8 LED Matrix

With low-voltage scanning, LED dot-matrix displays have advantages such as power saving, long service life, low cost, high brightness, wide angle of view, long visual range, being waterproof… LED dot-matrix displays can meet the needs of different applications and thus have a broad development prospect.

The 8*8 dot-matrix is made up of sixty-four LEDs, and each LED is placed at the cross point of a row and a column. When the electrical level of a certain row is 1 and the electrical level of a certain column is 0, the corresponding LED will light up. If you want to light the LED on the first dot, you should set pin 9 to high level and pin 13 to low level. If you want to light LEDs on the first row, you should set pin 9 to high level and pins 13, 3, 4, 10, 6, 11, 15 and 16 to low level. If you want to light the LEDs on the first column, set pin 13 to low level and pins 9, 14, 8, 12, 1, 7, 2 and 5 to high level.

Connection



Programming Code

```
*******code Begin*******
// set an array to store character of "0"
unsigned char Text[]={0x00,0x1c,0x22,0x22,0x22,0x22,0x22,0x1c};
void Draw_point(unsigned char x,unsigned char y)// point drawing function
{ clear_();
   digitalWrite(x+2, HIGH);
   digitalWrite(y+10, LOW);
   delay(1);
}
void show_num(void)// display function, call point drawing function

{
  unsigned char i,j,data;
  for(i=0;i<8;i++)
  {
     data=Text[i];
     for(j=0;j<8;j++)
     {
        if(data & 0x01)Draw_point(j,i);
        data>>=1;
     }
  }
}
void setup(){
int i = 0 ;
for(i=2;i<18;i++)
{
   pinMode(i, OUTPUT);
  }
  clear_();
}
void loop()
{ show_num();
}
void clear_(void)// clear screen
{for(int i=2;i<10;i++)
  digitalWrite(i, LOW);
  for(int i=0;i<8;i++)
  digitalWrite(i+10, HIGH);
}
*******code End*******
```

Use this device with original accessories only. Velleman nv cannot be held responsible in the event of damage or injury resulting from (incorrect) use of this device. For more info concerning this product and the latest version of this manual, please visit our website www.velleman.eu. The information in this manual is subject to change without prior notice.

© COPYRIGHT NOTICE

The copyright to this manual is owned by Velleman nv. All worldwide rights reserved. No part of this manual may be copied, reproduced, translated or reduced to any electronic medium or otherwise without the prior written consent of the copyright holder.

Velleman® Service and Quality Warranty

Since its foundation in 1972, Velleman® acquired extensive experience in the electronics world and currently distributes its products in over 85 countries.

All our products fulfil strict quality requirements and legal stipulations in the EU. In order to ensure the quality, our products regularly go through an extra quality check, both by an internal quality department and by specialized external organisations. If, all precautionary measures notwithstanding, problems should occur, please make appeal to our warranty (see guarantee conditions).

General Warranty Conditions Concerning Consumer Products (for EU):

• All consumer products are subject to a 24-month warranty on production flaws and defective material as from the original date of purchase.

• Velleman® can decide to replace an article with an equivalent article, or to refund the retail value totally or partially when the complaint is valid and a free repair or replacement of the article is impossible, or if the expenses are out of proportion.

You will be delivered a replacing article or a refund at the value of 100% of the purchase price in case of a flaw occurred in the first year after the date of purchase and delivery, or a replacing article at 50% of the purchase price or a refund at the value of 50% of the retail value in case of a flaw occurred in the second year after the date of purchase and delivery.

• Not covered by warranty:

- all direct or indirect damage caused after delivery to the article (e.g. by oxidation, shocks, falls, dust, dirt, humidity...), and by the article, as well as its contents (e.g. data loss), compensation for loss of profits;

- consumable goods, parts or accessories that are subject to an aging process during normal use, such as batteries (rechargeable, non-rechargeable, built-in or replaceable), lamps, rubber parts, drive belts... (unlimited list);

- flaws resulting from fire, water damage, lightning, accident, natural disaster, etc.…;

- flaws caused deliberately, negligently or resulting from improper handling, negligent maintenance, abusive use or use contrary to the manufacturer's instructions;

- damage caused by a commercial, professional or collective use of the article (the warranty validity will be reduced to six (6) months when the article is used professionally);

- damage resulting from an inappropriate packing and shipping of the article;

- all damage caused by modification, repair or alteration performed by a third party without written permission by Velleman®.

• Articles to be repaired must be delivered to your Velleman® dealer, solidly packed (preferably in the original packaging), and be completed with the original receipt of purchase and a clear flaw description.

• Hint: In order to save on cost and time, please reread the manual and check if the flaw is caused by obvious causes prior to presenting the article for repair. Note that returning a non-defective article can also involve handling costs.

• Repairs occurring after warranty expiration are subject to shipping costs.

• The above conditions are without prejudice to all commercial warranties.

The above enumeration is subject to modification according to the article (see article's manual).

Made in PRC
Imported by Velleman nv
Legen Heirweg 33, 9890 Gavere, Belgium
www.velleman.eu