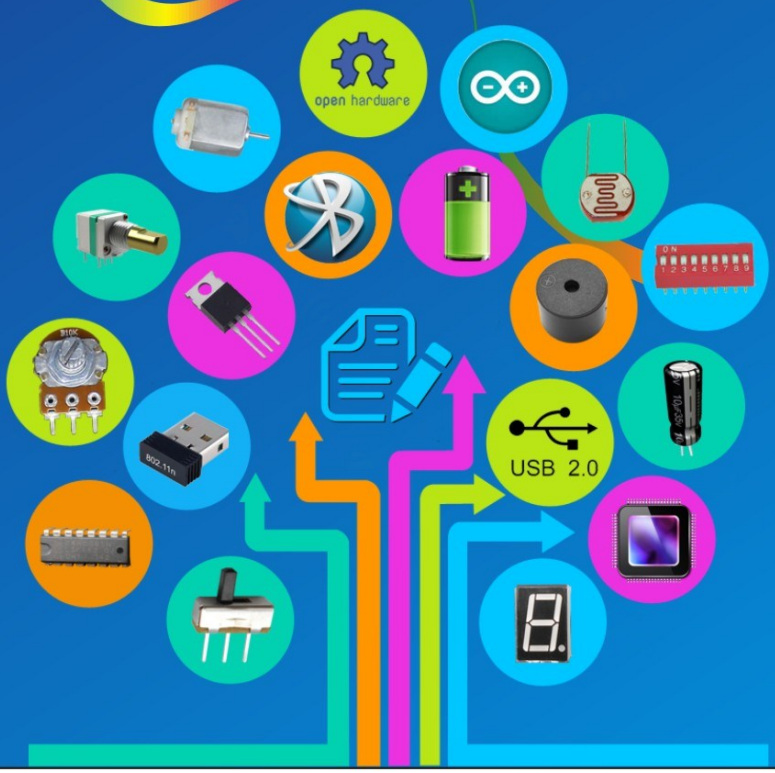
Adeept

Super Starter kit for Arduino

Sharing Perfects Innovation



Preface

Adeept is a technical service team of open source software and hardware. Dedicated to applying the Internet and the latest industrial technology in open source area, we strive to provide best hardware support and software service for general makers and electronic enthusiasts around the world. We aim to create infinite possibilities with sharing. No matter what field you are in, we can lead you into the electronic world and bring your ideas into reality.

This is an entry-level learning kit for Arduino. Some common electronic components and sensors are included. Through the learning, you will get a better understanding of Arduino, and be able to make fascinating works based on Arduino.

If you have any problems for learning, please contact us at support@adeept.com. We will do our best to help you solve the problem.

Component List

1x Adeept UNO Board(Arduino UNO)
1x DC Motor
1x L9110 motor driver
1x LCD1602
1x Dot-matrix Display
1x 7-Segment Display
1x NE555 timer
2x 74HC595
1x Active buzzer
1x Photoresistance
1x Tilt Switch
2x Switch
1x RGB LED
8x Red LED
4x Green LED
4x Yellow LED
4x Blue LED
16x Resistor(220Ω)
10x Resistor(1 kΩ)
5x Resistor(10 kΩ)
5x Capacitor(104)
2x Capacitor(10uF)
4x Button(large)
8x Button(small)
1x Button cap(red)
1x Button cap(white)
2x Button cap(blue)
2x NPN Transistor(8050)
2x PNP Transistor(8550)
2x Potentiometer(10KΩ)
1x A battery holder
1x Breadboard
1x USB Cable
40x Male to Male Jumper Wires
8x Male to Female Jumper Wires
1x Header(40pin)
1x Band Resistor Card
1x Project Box

Content

About Arduino

What is Arduino?

Arduino is an open-source electronics platform based on easy-to-use hardware and software. It's intended for anyone making interactive projects.

ARDUINO BOARD

Arduino senses the environment by receiving inputs from many sensors, and affects its surroundings by controlling lights, motors, and other actuators.

ARDUINO SOFTWARE

You can tell your Arduino what to do by writing code in the Arduino programming language and using the Arduino development environment.

Before the development of Arduino program, the first thing you have to do is to install Arduino IDE software. The software provides you with the basic development environment that is required for developing Arduino program. You need the following URL to download Arduino IDE:
http://www.arduino.cc/en/Main/Software

For different operating system platforms, the way of using Arduino IDE is different. Please refer to the following links:
Windows User : http://www.arduino.cc/en/Guide/Windows
Mac OS X User : http://www.arduino.cc/en/Guide/MacOSX
Linux User : http://playground.arduino.cc/Learning/Linux

For more detailed information about Arduino IDE, please refer to the following link:
http://www.arduino.cc/en/Guide/HomePage

Lesson 1 Blinking LED

Overview

In this tutorial, we will start the journey of learning Arduino UNO. In the first lesson, we will learn how to make a LED blinking.

Requirement

- 1* Arduino UNO
- 1* USB Cable
- 1* 220Ω Resistor
- 1* LED
- 1* Breadboard
- 2* Jumper Wires

Principle

In this lesson, we will program the Arduino's GPIO output high(+5V) and low level(0V), and then make the LED which is connected to the Arduino's GPIO flicker with a certain frequency.

1. What is the LED?

The LED is the abbreviation of light emitting diode. It is usually made of gallium arsenide, gallium phosphide semiconductor materials. The LED has two electrodes, a positive electrode and a negative electrode, it will light only when a forward current passes, and it can be red, blue, green or yellow light, etc. The color of light depends on the materials it was made.

In general, the drive current for LED is 5-20mA. Therefore, in reality it usually needs an extra resistor for current limitation so as to protect the LED.

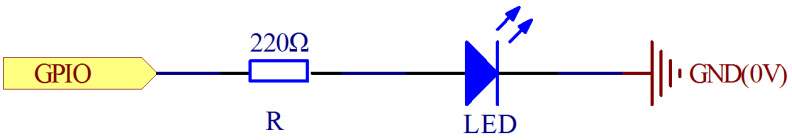
2. What is the resistor?

The main function of the resistor is to limit current. In the circuit, the character 'R' represents resistor, and the unit of resistor is ohm(Ω).

The band resistor is used in this experiment. A band resistor is one whose surface is coated with some particular color through which the resistance can be identified directly.

There are two methods for connecting LED to Arduino's GPIO:

①



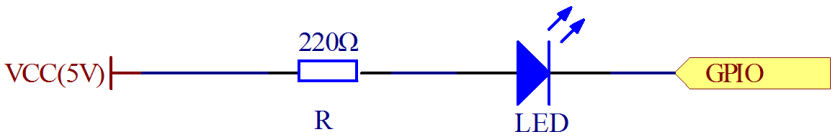
As shown in the schematic diagram above, the anode of LED is connected to Arduino's GPIO via a resistor, and the cathode of LED is connected to the ground(GND). When the GPIO output high level, the LED is on; when the GPIO output low level, the LED is off.

The size of the current-limiting resistor is calculated as follows: 5~20mA current is required to make an LED on, and the out put voltage of the Arduino UNO's GPIO is 5V, so we can get the resistance :

R = U / I = 5V / (5~20mA) = 250Ω~1KΩ

Since the LED has a certain resistance, thus we choose a 220ohm resistor.

②



As shown in the schematic diagram above, the anode of LED is connected to VCC(+5V), and the cathode of LED is connected to the Arduino's GPIO. When the GPIO output low level, the LED is on; when the GPIO output high level, the LED is off.

The experiment is based on method ①, we select Arduino's D8 pin to control the LED. When the Arduino's D8 pin is programmed to output high level, then the LED will be on, next delay for the amount of time, and then programmed the D8 pin to low level to make the LED off. Continue to perform the above process, you can get a blinking LED.

3. Key functions:

- setup()

The setup() function is called when a sketch starts. Use it to initialize variables, pin modes, start using libraries, etc. The setup function will only run once, after each powerup or reset of the Arduino board.

- loop()

After creating a setup() function, which initializes and sets the initial values, the loop() function does precisely what its name suggests, and loops

consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.

●pinMode()

Configures the specified pin to behave either as an input or an output.

As of Arduino 1.0.1, it is possible to enable the internal pullup resistors with the mode INPUT_PULLUP. Additionally, the INPUT mode explicitly disables the internal pullups.

●digitalWrite()

Write a HIGH or a LOW value to a digital pin.

If the pin has been configured as an OUTPUT with pinMode(), its voltage will be set to the corresponding value: 5V (or 3.3V on 3.3V boards) for HIGH, 0V (ground) for LOW.

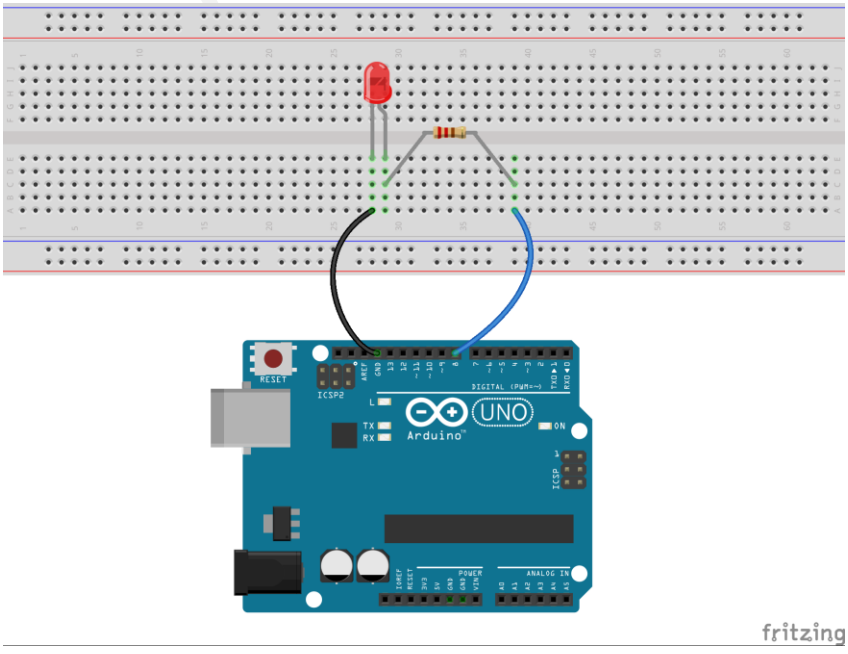
If the pin is configured as an INPUT, digitalWrite() will enable (HIGH) or disable (LOW) the internal pullup on the input pin. It is recommended to set the pinMode() to INPUT_PULLUP to enable the internal pull-up resistor.

●delay()

Pauses the program for the amount of time (in miliseconds) specified as parameter. (There are 1000 milliseconds in a second.)

Procedures

1. Build the circuit

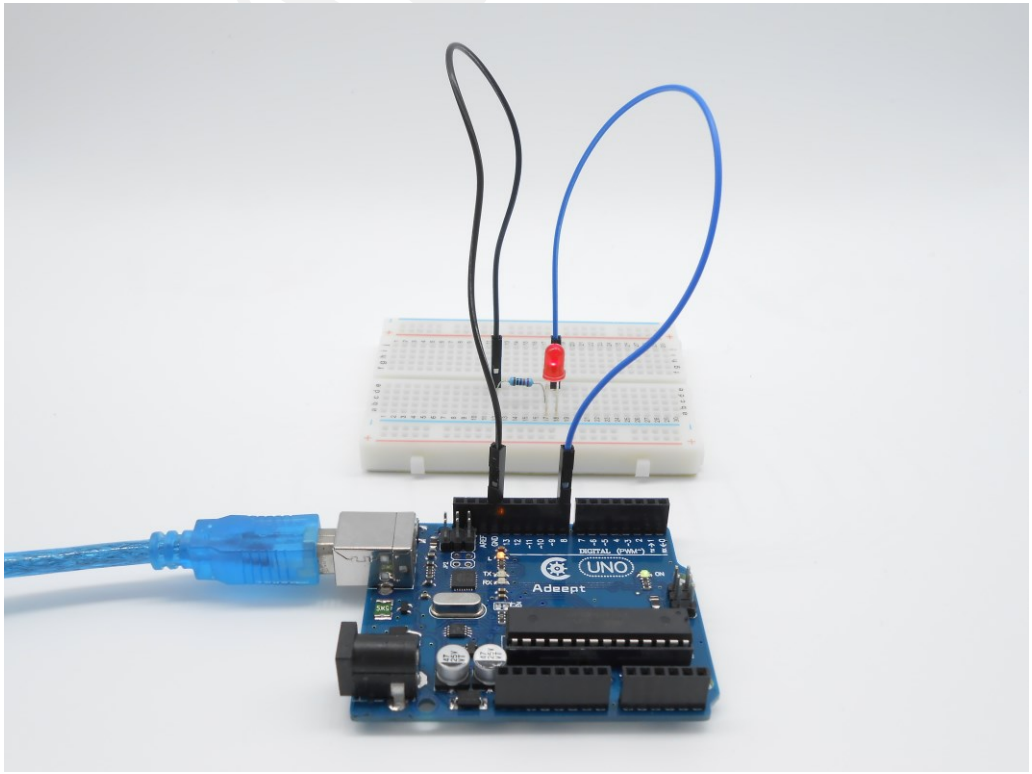


2. Program

```
/************************************************************
File name:   01_blinkingLed.ino
Description:  Lit LED, let LED blinks.
Website: www.adeept.com
E-mail: support@adeept.com
Author: Tom
Date: 2015/05/02
************************************************************/
int ledPin=8; //definition digital 8 pins as pin to control the LED
void setup()
{
    pinMode(ledPin,OUTPUT);    //Set the digital 8 port mode, OUTPUT:
Output mode
}
void loop()
{
    digitalWrite(ledPin,HIGH); //HIGH is set to about 5V PIN8
    delay(1000);               //Set the delay time, 1000 = 1S
    digitalWrite(ledPin,LOW);  //LOW is set to about 5V PIN8
    delay(1000);               //Set the delay time, 1000 = 1S
}
```

3. Compile the program and upload to Arduino UNO board

Now, you can see the LED is blinking.



Lesson 2 Buzzer

Overview

In this lesson, we will learn how to program the Arduino to make an active buzzer sound.

Requirement

- 1* Arduino UNO
- 1* USB cable
- 1* Active buzzer
- 1* 1 kΩ Resistor
- 1* NPN Transistor (S8050)
- 1* Breadboard
- Several Jumper wires

Principle

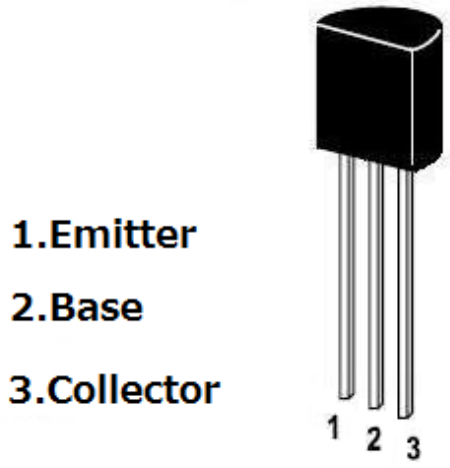
A buzzer or beeper is an audio signaling device. As a type of electronic buzzer with integrated structure, which use DC power supply, are widely used in computers, printers, photocopiers, alarms, electronic toys, automotive electronic equipments, telephones, timers and other electronic products for voice devices. Buzzers can be categorized as active and passive buzzers (See the following pictures).

When you place the pins of buzzers upward, you can see that two buzzers are different, the buzzer that green circuit board exposed is the passive buzzer.

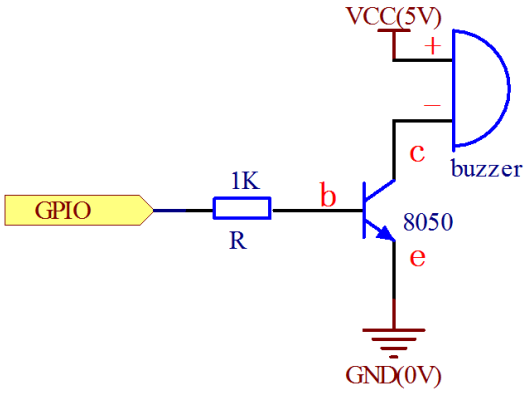
In this study, the buzzer we used is active buzzer. Active buzzer will sound as long as the power supply. We can program to make the Arduino output alternating high and low level, so that the buzzer sounds.

A slightly larger current is needed to make a buzzer sound. However, the output current of Arduino's GPIO is weak, so we need a transistor to drive the buzzer.

The main function of transistor is blowing up the voltage or current. The transistor can also be used to control the circuit conduction or deadline. And the transistor is divided into two kinds, one kind is NPN, for instance, the S8050 we provided; another kind is PNP transistor such as the S8550 we provided. The transistor we used is as shown in below:



There are two driving circuit for the buzzer:



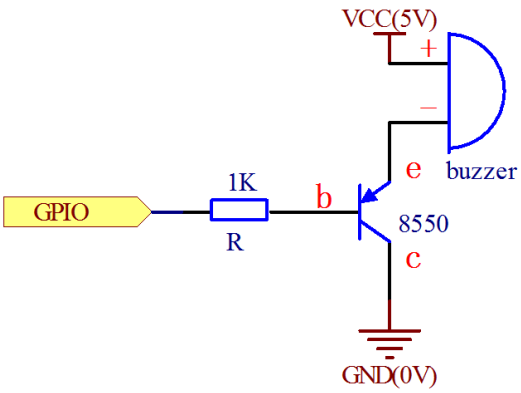


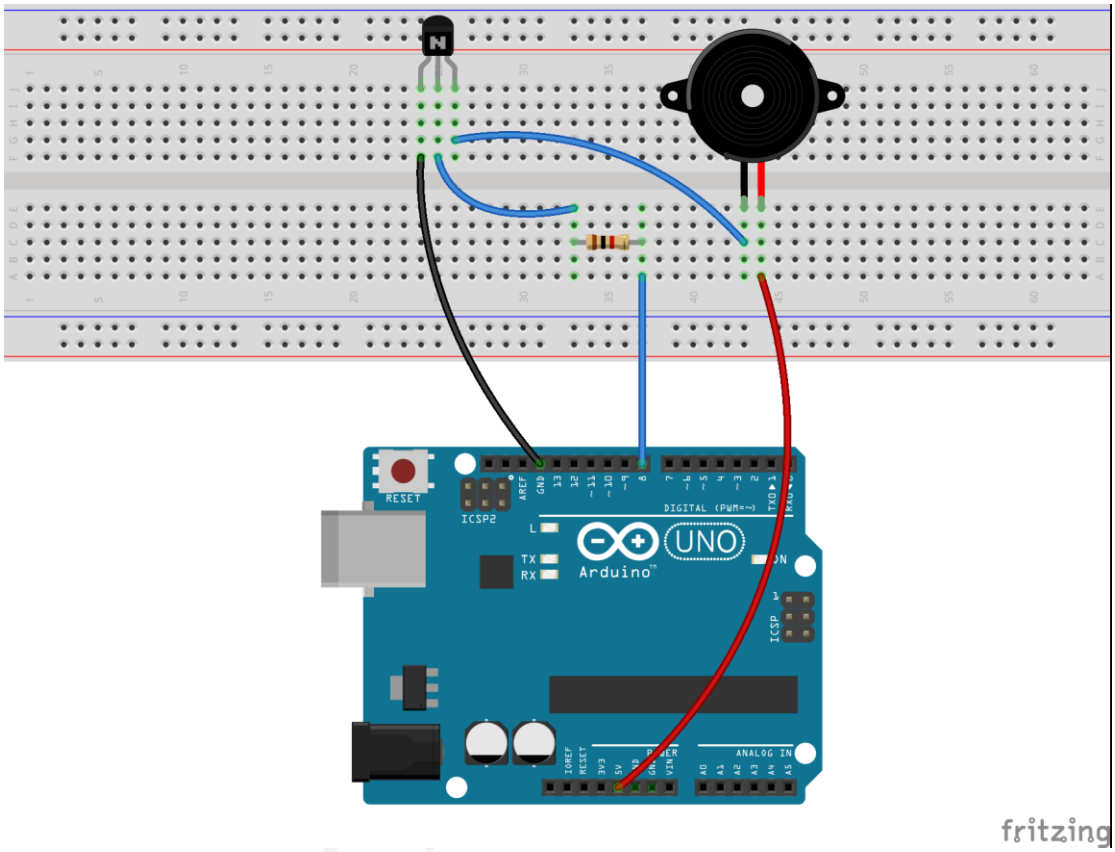
Figure1 Figure2

Figure 1: Set the Arduino GPIO as a high level, the transistor S8050 will conduct, and then the buzzer will sound; set the Arduino GPIO as low level, the transistor S8050 will cut off, then the buzzer will stop.

Figure 2: Set the Arduino GPIO as low level, the transistor S8550 will conduct, and the buzzer will sound; set the Arduino GPIO as a high level, the transistor S8550 will cut off, then the buzzer will stop.

Procedures

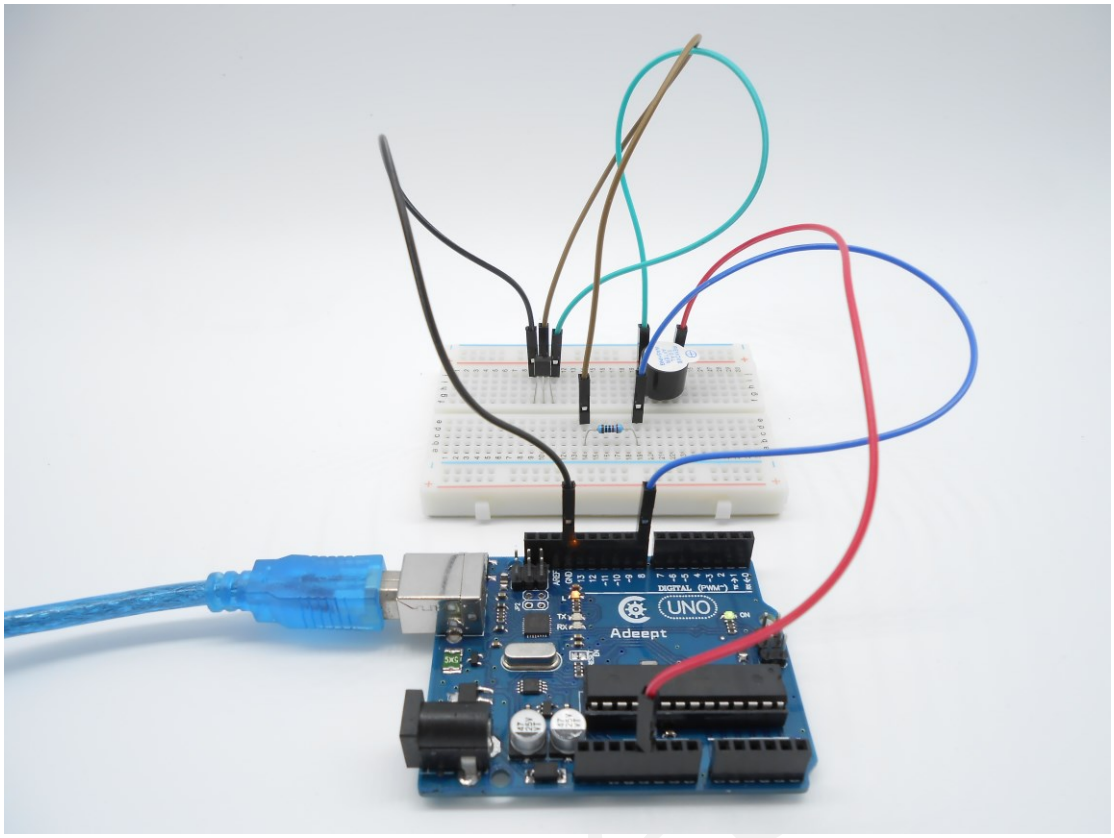
1. Build the circuit



2. Program

3. Compile the program and upload to Arduino UNO board

Now, you should be able to hear the sound of the buzzer.



Summary

By learning this lesson, we have mastered the basic principle of the buzzer and the transistor. We also learned how to program the Arduino and then control the buzzer. I hope you can use what you have learned in this lesson to do some interesting things.

Lesson 3 Controlling an LED with a button

Overview

In this lesson, we will learn how to detect the state of a button, and then toggle the state of LED based on the state of the button.

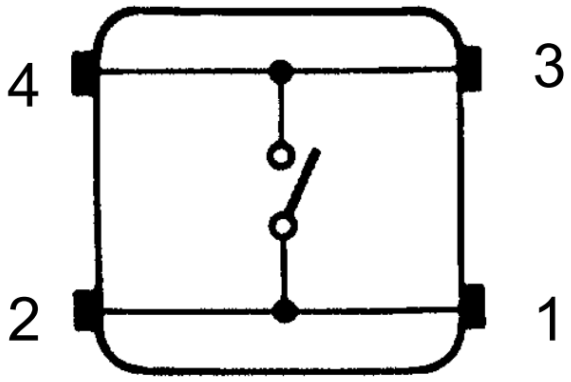
Requirement

- 1* Arduino UNO
- 1* USB cable
- 1* Button
- 1* LED
- 1* 10KΩ Resistor
- 1* 220Ω Resistor
- 1* Breadboard
- Several Jumper wires

Principle

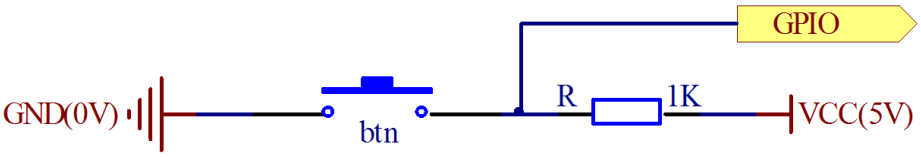
1. Button

Buttons are a common component used to control electronic devices. They are usually used as switches to connect or disconnect circuits. Although buttons come in a variety of sizes and shapes, the one used in this experiment will be a 12mm button as shown in the following pictures. Pins pointed out by the arrows of same color are meant to be connected.

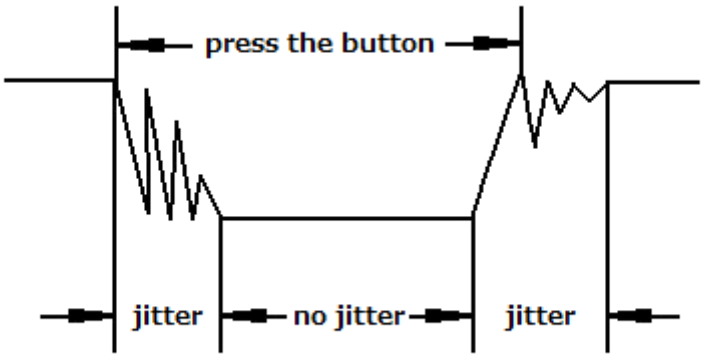


The button we used is a normally open type button. The two contacts of a button is in the off state under the normal conditions, only when the button is pressed they are closed.

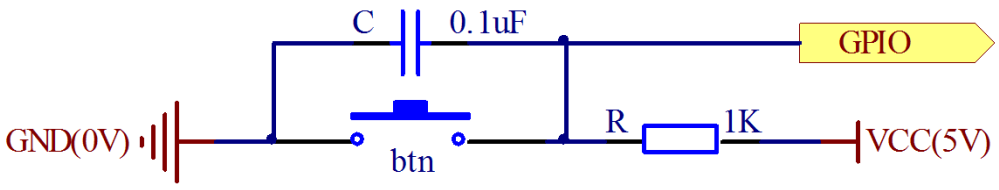
The schematic diagram we used is as follows:



The button jitter must be happen in the process of using. The jitter waveform is as the flowing picture:



Each time you press the button, the Arduino will think you have pressed the button many times due to the jitter of the button.We must to deal with the jitter of buttons before we use the button. We can through the software programming method to remove the jitter of buttons, and you can use a capacitance to remove the jitter of buttons. We introduce the software method. First, we detect whether the level of button interface is low level or high level.When the level we detected is low level, 5~10 MS delay is needed, and then detect whether the level of button interface is low or high. If the signal is low, we can confirm that the button is pressed once. You can also use a 0.1 uF capacitance to clean up the jitter of buttons. The schematic diagram is shown in below:



2. interrupt

Hardware interrupts were introduced as a way to reduce wasting the processor's valuable time in polling loops, waiting for external events. They

may be implemented in hardware as a distinct system with control lines, or they may be integrated into the memory subsystem.

3. Key functions:

●attachInterrupt(interrupt, ISR, mode)

Specifies a named Interrupt Service Routine (ISR) to call when an interrupt occurs. Replaces any previous function that was attached to the interrupt. Most Arduino boards have two external interrupts: numbers 0 (on digital pin 2) and 1 (on digital pin 3).

Generally, an ISR should be as short and fast as possible. If your sketch uses multiple ISRs, only one can run at a time, other interrupts will be ignored (turned off) until the current one is finished. as delay() and millis() both rely on interrupts, they will not work while an ISR is running. delayMicroseconds(), which does not rely on interrupts, will work as expected.

Syntax

attachInterrupt(pin, ISR, mode)

Parameters

pin: the pin number

ISR: the ISR will be called when the interrupt occurs; this function must take no parameters and return nothing. This function is sometimes referred to as an interrupt service routine.

mode: defines when the interrupt should be triggered. Four contstants are predefined as valid values:

-LOW to trigger the interrupt whenever the pin is low,

-CHANGE to trigger the interrupt whenever the pin changes value

-RISING to trigger when the pin goes from low to high,

-FALLING for when the pin goes from high to low.

●digitalRead()

Reads the value from a specified digital pin, either HIGH or LOW.

Syntax

digitalRead(pin)

Parameters

pin: the number of the digital pin you want to read (int)

Returns

HIGH or LOW

●delayMicroseconds(us)

Pauses the program for the amount of time (in microseconds) specified as parameter. There are a thousand microseconds in a millisecond, and a million microseconds in a second.

Currently, the largest value that will produce an accurate delay is 16383. This could change in future Arduino releases. For delays longer than a few thousand microseconds, you should use delay() instead.

Syntax

delayMicroseconds(us)

Parameters

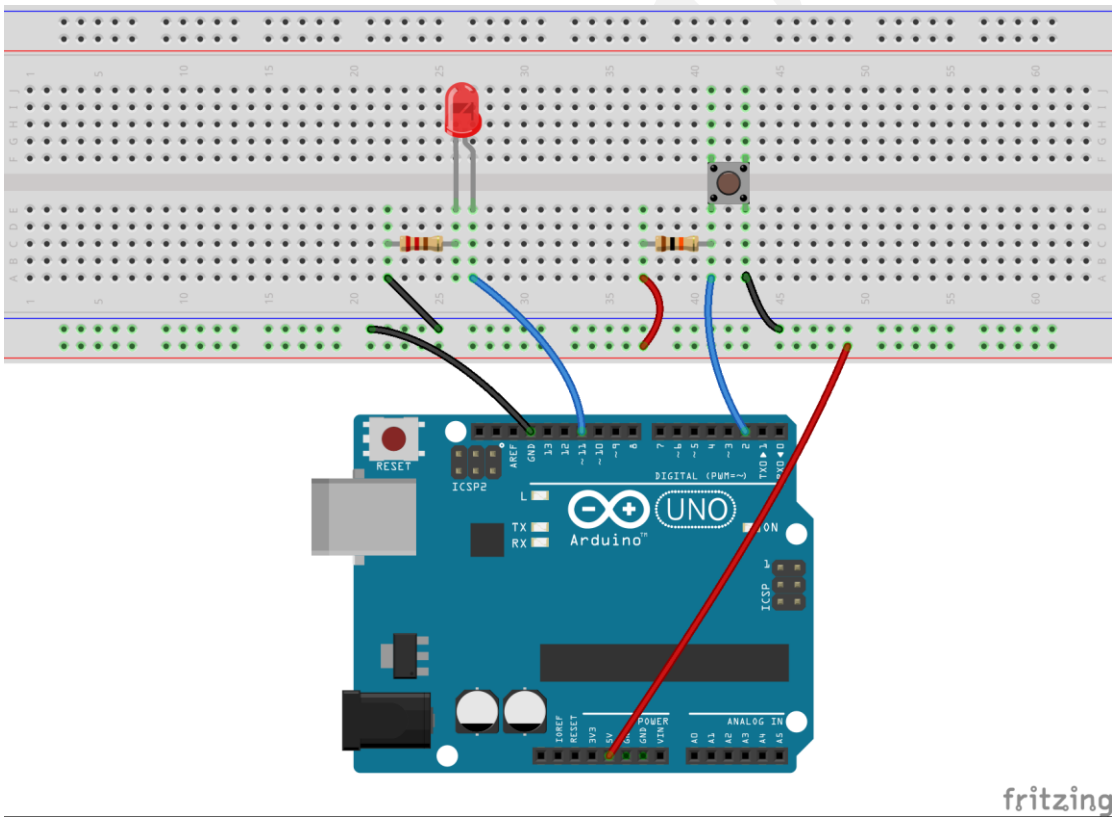
us: the number of microseconds to pause (unsigned int)

Returns

None

Procedures

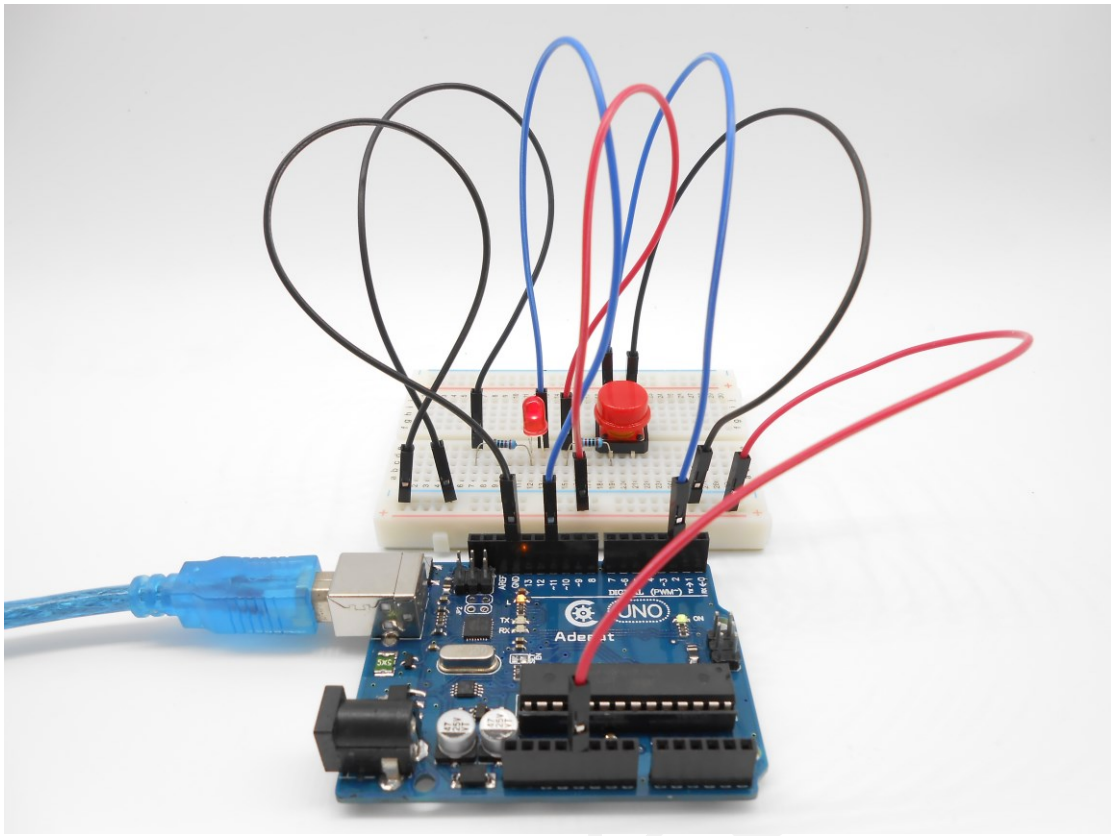
1. Build the circuit



2. Program

3. Compile the program and upload to Arduino UNO board

When you press the button, you can see the state of the LED will be toggled. (ON->OFF, OFF->ON).



Summary

Through this lesson, you should have learned how to use the Arduino UNO detects an external button state, and then toggle the state of LED relying on the state of the button detected before.

Lesson 4 Tilt Switch

Overview

In this lesson, we will learn how to use the tilt switch and change the state of an LED by changing the angle of tilt switch.

Requirement

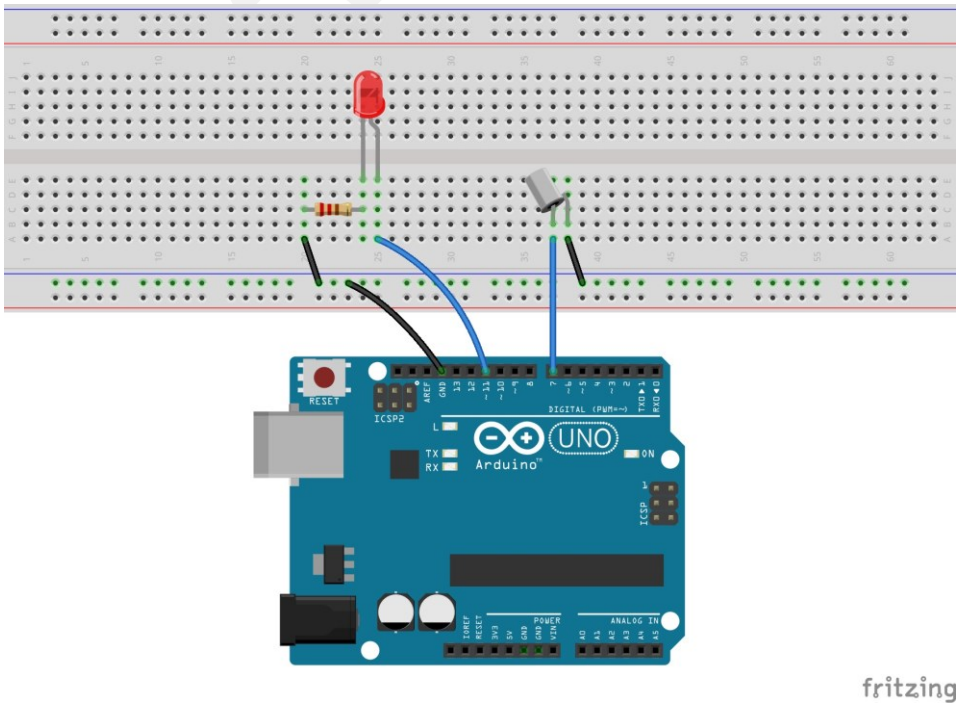
- 1* Arduino UNO
- 1* USB cable
- 1* Tilt switch
- 1* LED
- 1* 220Ω Resistor
- 1* Breadboard
- Several Jumper wires

Principle

The tilt switch is also called the ball switch. When the switch is tilted in the appropriate direction, the contacts will be connected, tilting the switch the opposite direction causes the metallic ball to move away from that set of contacts, thus breaking that circuit.

Procedures

1. Build the circuit

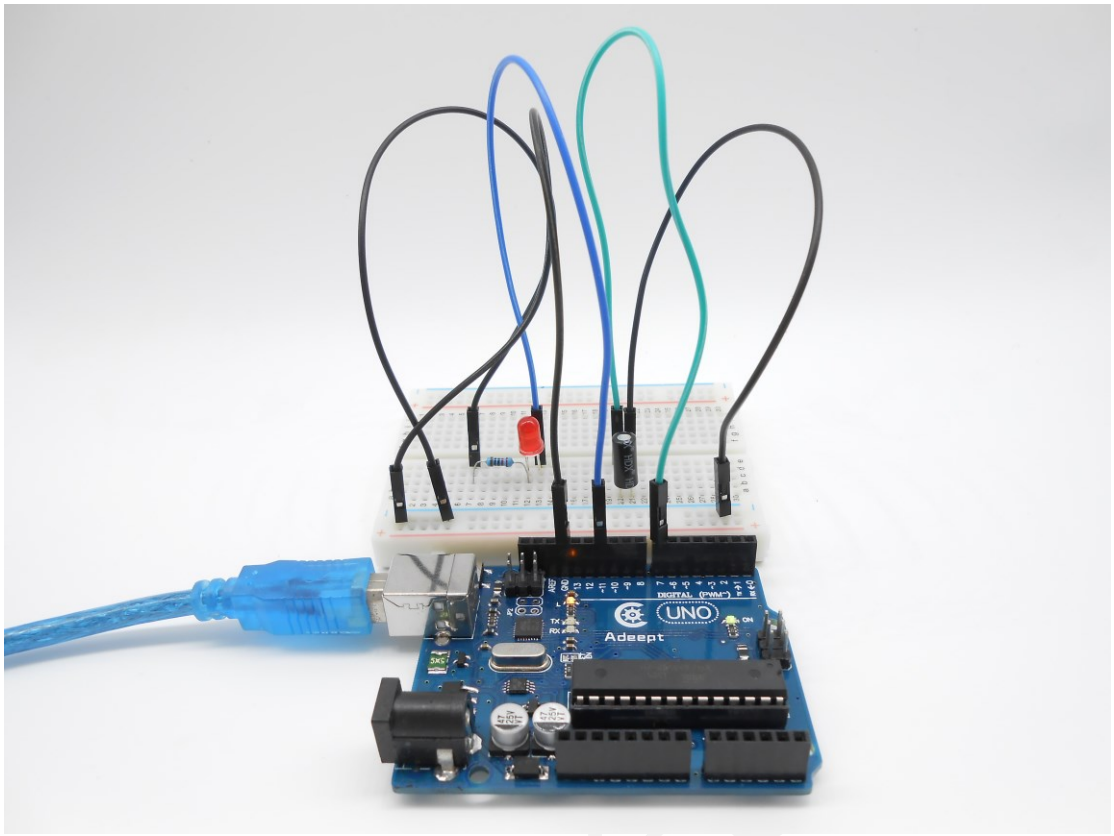


2. Program

```
/************************************************************
File name: 04_tiltSwitch.ino
Description: Tilt switches to control the LED light on or off
Website: www.adeept.com
E-mail: support@adeept.com
Author: Tom
Date: 2015/05/02
************************************************************/

int ledpin=11;          //definition digital 11 pins as pin to control the
                        //LED
int tiltSwitchpin=7;  //Set the digital 7 to tilt switch interface
int val;              //Define variable val

void setup()
{
  pinMode(ledpin,OUTPUT);     //Define small lights interface for the
                              //output interface
  pinMode(tiltSwitchpin,INPUT_PULLUP);//define the tilt switch
                              //interface for input interface
}

void loop()
{
  val=digitalRead(tiltSwitchpin);//Read the number seven level value is
                              //assigned to val
  if(val==LOW)                //Detect tilt switch is disconnected, the
                              //tilt switch when small lights go out
  { digitalWrite(ledpin,LOW);} //Output low, LED OFF
  else                       //Detection of tilt switch is conduction,
//tilt the little lights up when the switch conduction
  { digitalWrite(ledpin,HIGH);}  //Output high, LE  ON
}
```

3. Compile the program and upload to Arduino UNO board

Now, when you lean the breadboard at an certain angle, you will see the state of LED is changed.



Summary

In this lesson, we have learned the principle and application of the tilt switch. Tilt switch is a very simple electronic component, but simple device can often make something interesting.

Lesson 5 LED Flowing Lights

Overview

In the first class, we have learned how to make an LED blink by programming the Arduino. Today, we will use the Arduino to control 8 LEDs, so that 8 LEDs showing the result of flowing.

Requirement

- 1* Arduino UNO
- 1* USB cable
- 8* LED
- 8* 220Ω Resistor
- 1* Breadboard
- Several Jumper wires

Principle

The principle of this experiment is very simple. It is very similar with the first class.

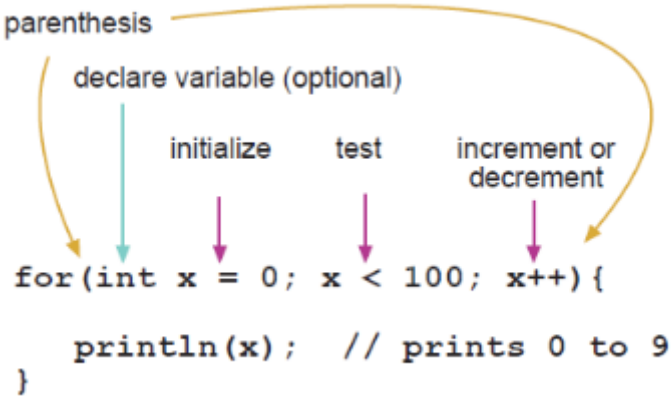
Key function:

●for statements

The for statement is used to repeat a block of statements enclosed in curly braces. An increment counter is usually used to increment and terminate the loop. The for statement is useful for any repetitive operation, and is often used in combination with arrays to operate on collections of data/pins.

There are three parts to the for loop header:

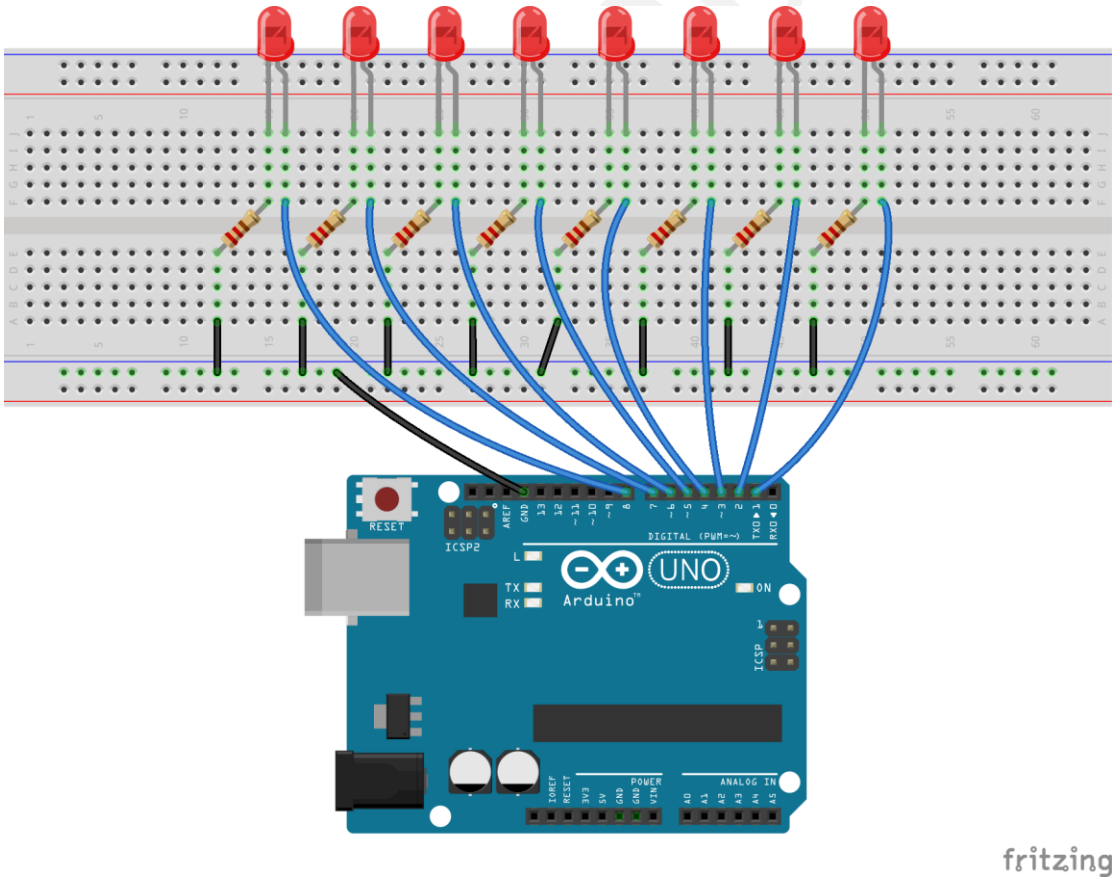
```
for (initialization; condition; increment) {
//statement(s);
}
```



The initialization happens first and exactly once. Each time through the loop, the condition is tested; if it's true, the statement block, and the increment is executed, then the condition is tested again. When the condition becomes false, the loop ends.

Procedures

1. Build the circuit

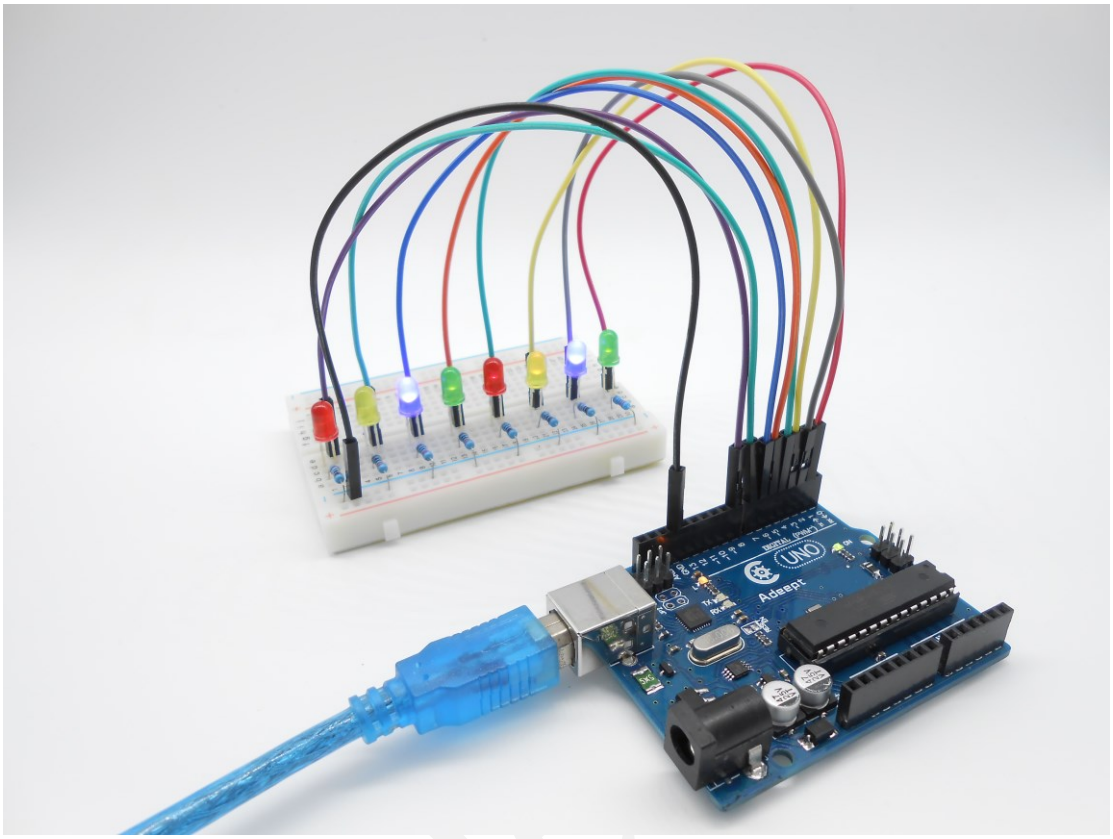


2. Program

3. Compile the program and upload to Arduino UNO board

Now, you should see 8 LEDs are lit in sequence from the right green one to the

left, next from the left to the right one. And then repeat the above phenomenon.



Summary

Through this simple and fun experiment, we have learned more skilled programming about the Arduino. In addition, you can also modify the circuit and code we provided to achieve even more dazzling effect.

Lesson 6 Breathing LED

Overview

In this lesson, we will learn how to program the Arduino to generate PWM signal. And use the PWM square-wave signal control an LED gradually becomes brighter and then gradually becomes dark like the animal's breathing.

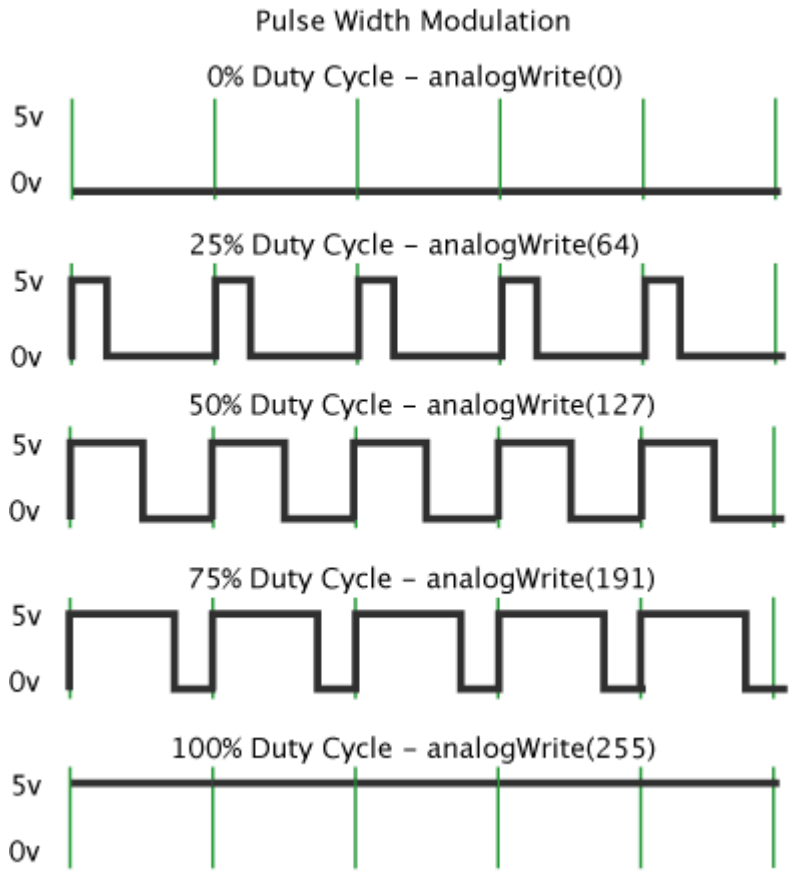
Requirement

- 1* Arduino UNO
- 1* USB cable
- 1* LED
- 1* 220Ω Resistor
- 1* Breadboard
- Several Jumper wires

Principle

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5v controlling the brightness of the LED.

In the graphic below, the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with Arduino's PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each. A call to analogWrite() is on a scale of 0 - 255, such that analogWrite(255) requests a 100% duty cycle (always on), and analogWrite(127) is a 50% duty cycle (on half the time) for example.



Key function:

●analogWrite()

Writes an analog value (PWM wave) to a pin. Can be used to light an LED at varying brightnesses or drive a motor at various speeds. After a call to analogWrite(), the pin will generate a steady square wave of the specified duty cycle until the next call to analogWrite() (or a call to digitalRead() or digitalWrite() on the same pin). You do not need to call pinMode() to set the pin as an output before calling analogWrite().

Syntax

analogWrite(pin, value)

Parameters

pin: the pin to write to.

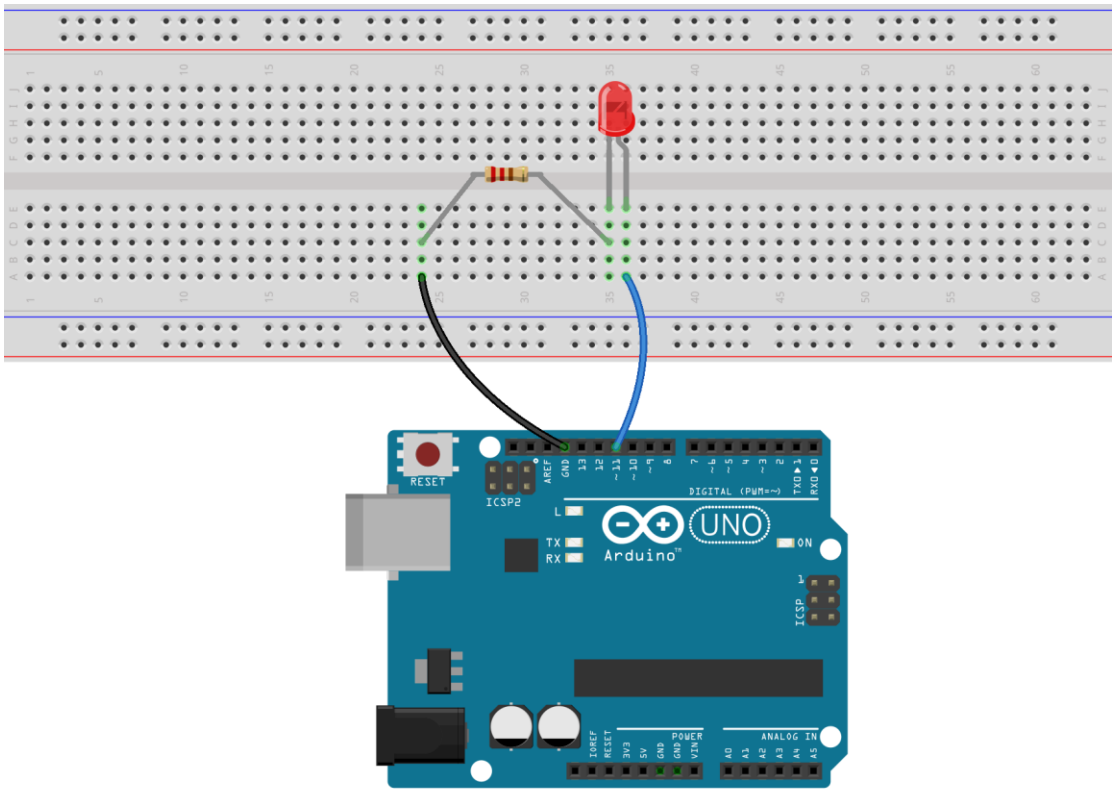
value: the duty cycle: between 0 (always off) and 255 (always on).

Returns

nothing

Procedures

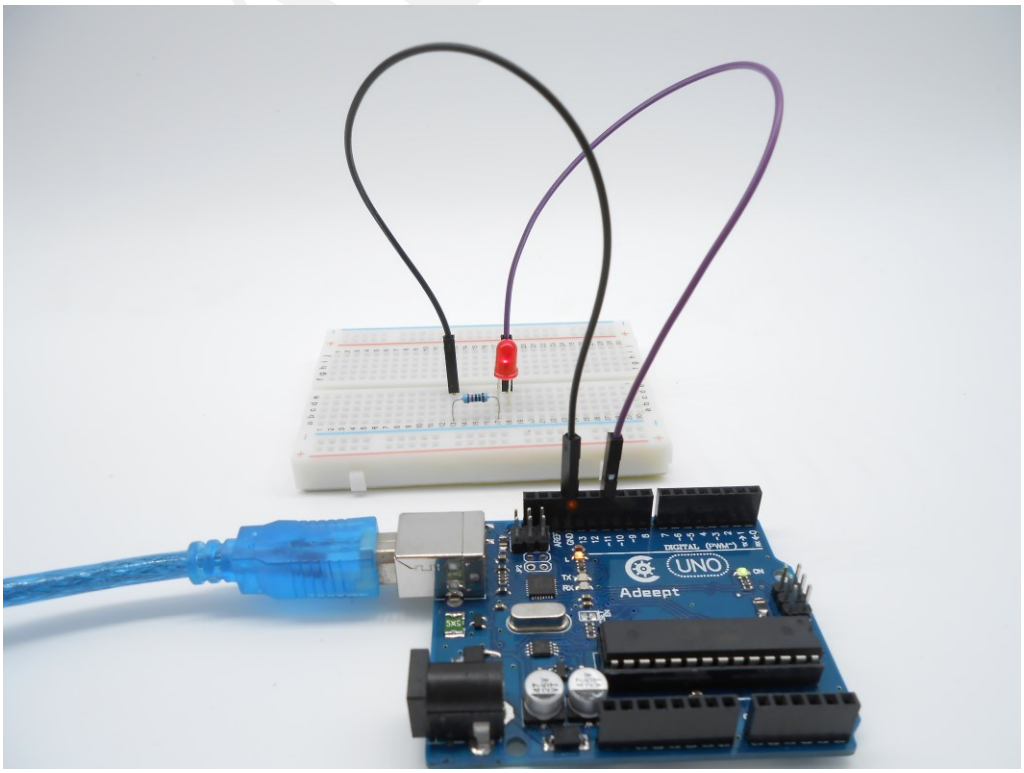
1. Build the circuit



2. Program

3. Compile the program and upload to Arduino UNO board.

Now, you should see the LED gradually from dark to brighter, and then from brighter to dark, continuing to repeat the process, its rhythm like the animal's breathing.



Summary

By learning this lesson, I believe that you have understood the basic principles of the PWM, and mastered the PWM programming on the Arduino platform.

Lesson 7 Controlling a RGB LED by PWM

Overview

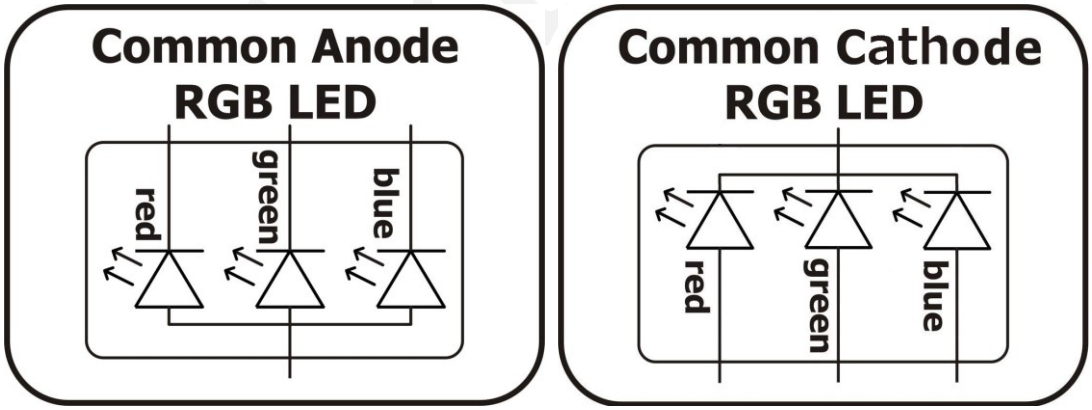
In this lesson, we will program the Arduino for RGB LED control, and make RGB LED emits a various of colors of light.

Requirement

- 1* Arduino UNO
- 1* USB cable
- 1* RGB LED
- 3* 220Ω Resistor
- 1* Breadboard
- Several Jumper wires

Principle

RGB LEDs consist of three LEDs. Each LED actually has one red, one green and one blue light. These three colored LEDs are capable of producing any color. Tri-color LEDs with red, green, and blue emitters, in general using a four-wire connection with one common lead (anode or cathode). These LEDs can have either common anode or common cathode leads.

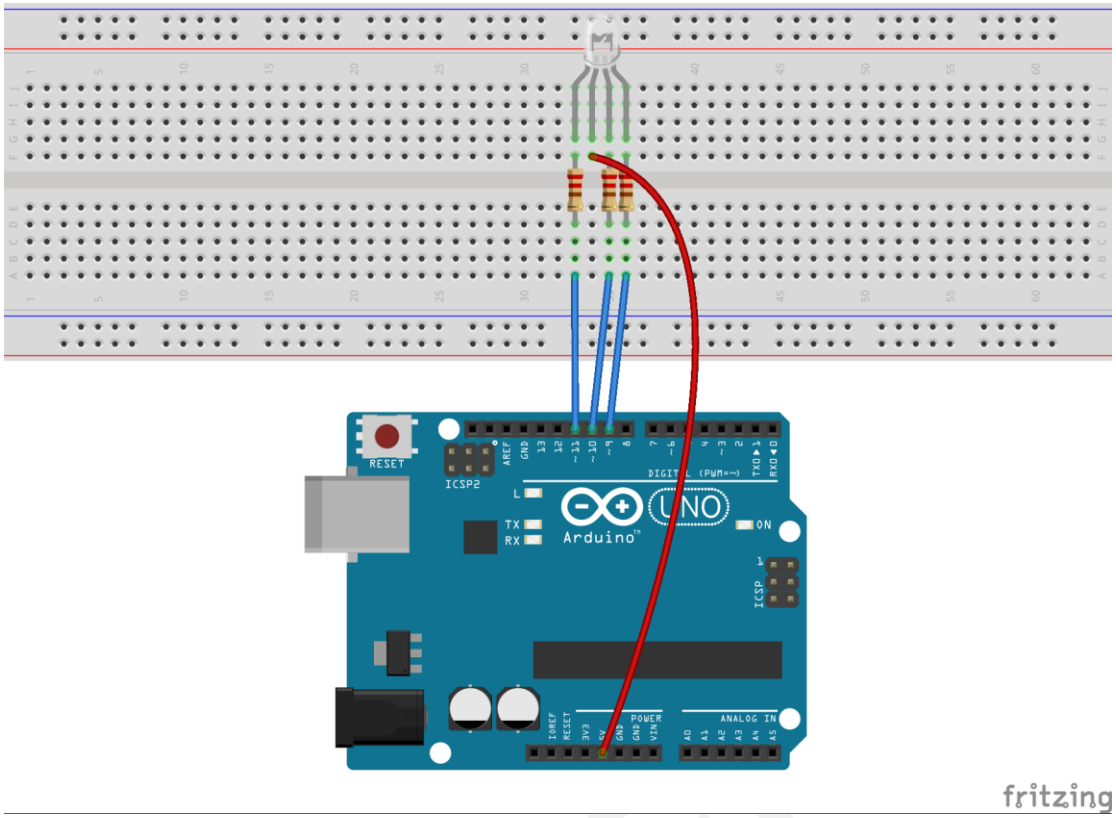


What we used in this experiment is the common anode RGB LED. The longest pin is the common anode of three LEDs. The pin is connected to the +5V pin of the Arduino, and the three remaining pins are connected to the Arduino's D9, D10, D11 pins through a current limiting resistor.

In this way, we can control the color of RGB LED by 3-channel PWM signal.

Procedures

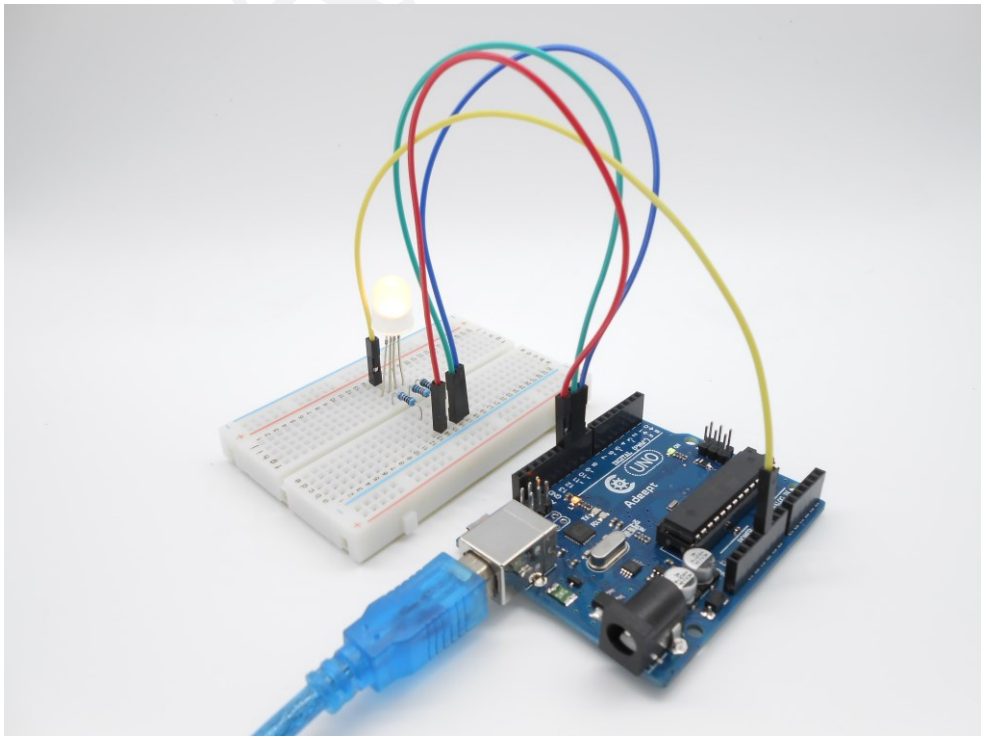
1. Build the circuit



2. Program

3. Compile the program and upload to Arduino UNO board

Now, you can see the RGB LED emitting red, green, blue, yellow, white and purple light, then the RGB LED will be off, each state continues 1s, after repeating the above procedure.



Summary

By learning this lesson, I believe you have already known the principle and the programming of RGB LED. I hope you can use your imagination to achieve even more cool ideas based on this lesson.

Lesson 8 7-segment display

Overview

In this lesson, we will program the Arduino to achieve the controlling of segment display.

Requirement

- 1* Arduino UNO
- 1* USB cable
- 1* 220Ω Resistor
- 1* 7-Segment display
- 1* Breadboard
- Several Jumper wires

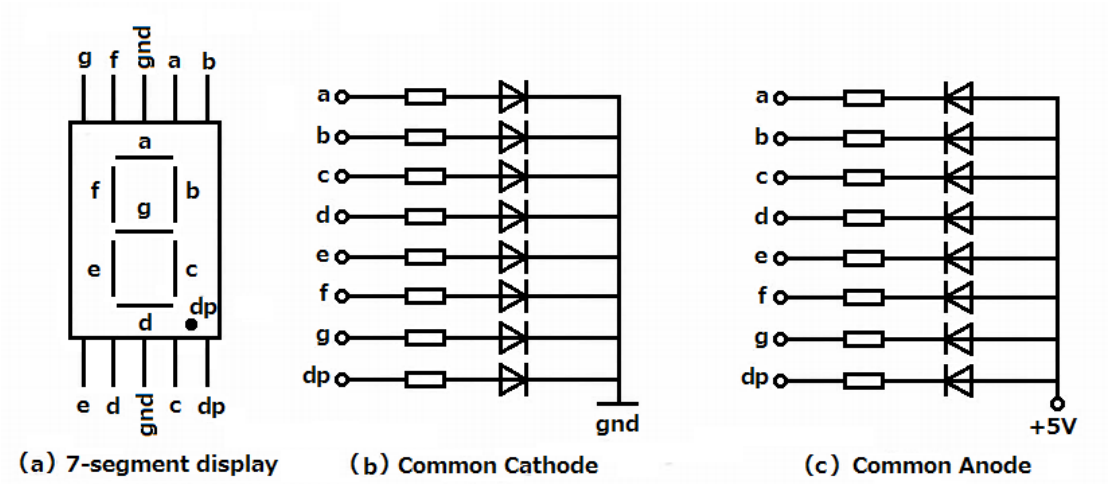
Principle

The seven-segment display is a form of electronic display device for displaying decimal numerals that is an alternative to the more complex dot matrix displays.

Seven-segment displays are widely used in digital clocks, electronic meters, basic calculators, and other electronic devices that display numerical information.

The seven-segment display is an 8-shaped LED display device composed of eight LEDs (including a decimal point), these segments respectively named a, b, c, d, e, f, g, dp.

The segment display can be divided into common anode and common cathode segment display by internal connections.

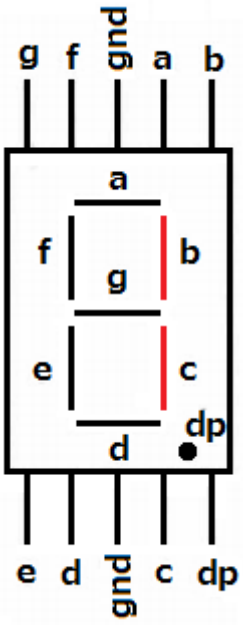


(a) 7-segment display (b) Common Cathode (c) Common Anode

When using a common anode LED, the common anode should to be connected to the power supply (VCC); when using a common cathode LED, the common cathode should be connected to the ground (GND).

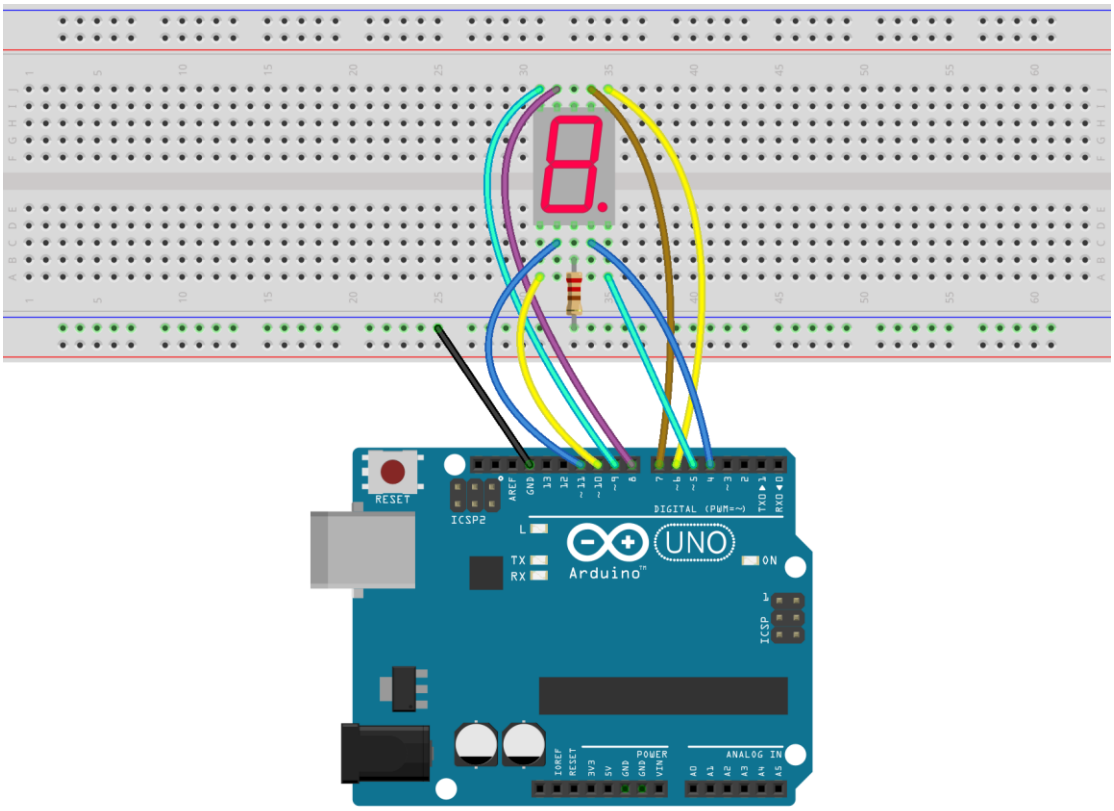
Each segment of a segment display is composed of LED, so a resistor is needed for protecting the LED.

A 7-segment display has seven segments for displaying a figure and a segment for displaying a decimal point. If you want to display a number '1', you should only light the segment b and c.



Procedures

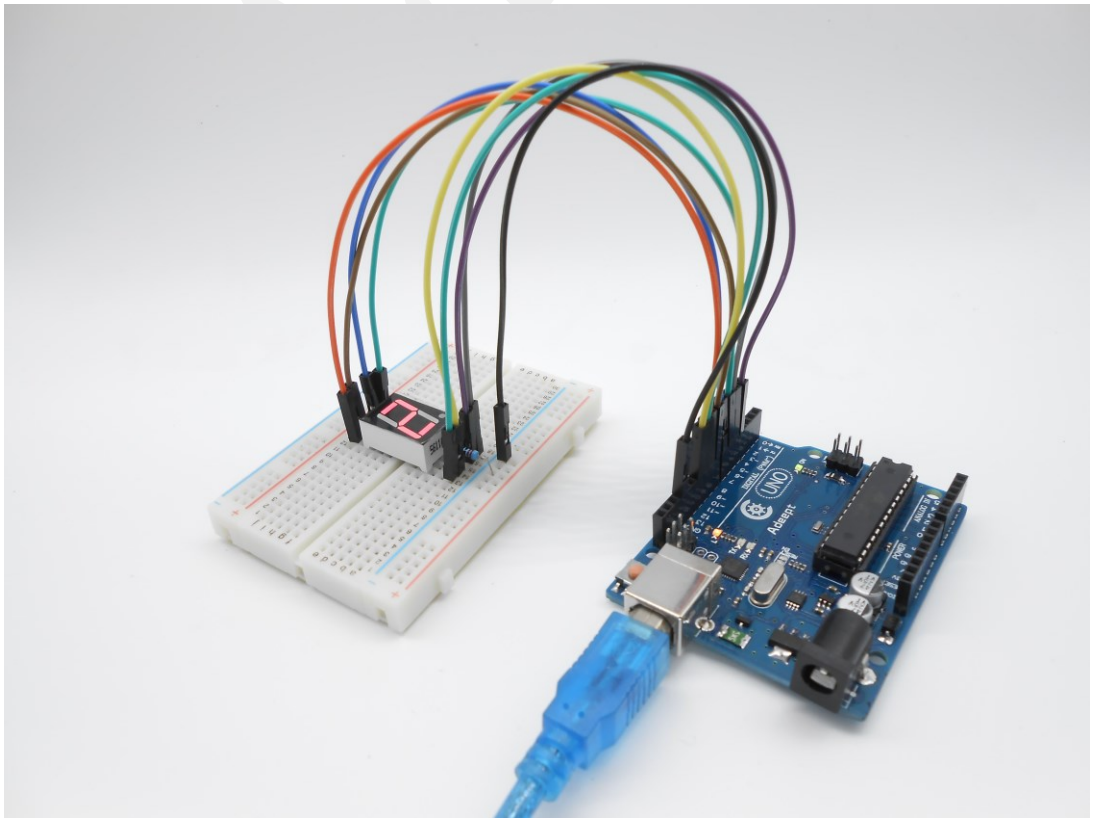
1. Build the circuit



2. Program

3. Compile the program and upload to Arduino UNO board

Now, you should see the number 0~9 are displayed on the segment display.



Summary

Through this lesson, we have learned the principle and programming of segment display. I hope you can combine the former course to modify the code we provided in this lesson to achieve cooler originality.

Lesson 9 Dot-matrix display

Overview

In this lesson, we will program to control a 8*8 dot-matrix to realize the display of graphical and digital we want.

Requirement

- 1* Arduino UNO
- 1* USB cable
- 1* 8*8 Dot-matrix
- 2* 74HC595
- 1* Breadboard
- Several Jumper wires

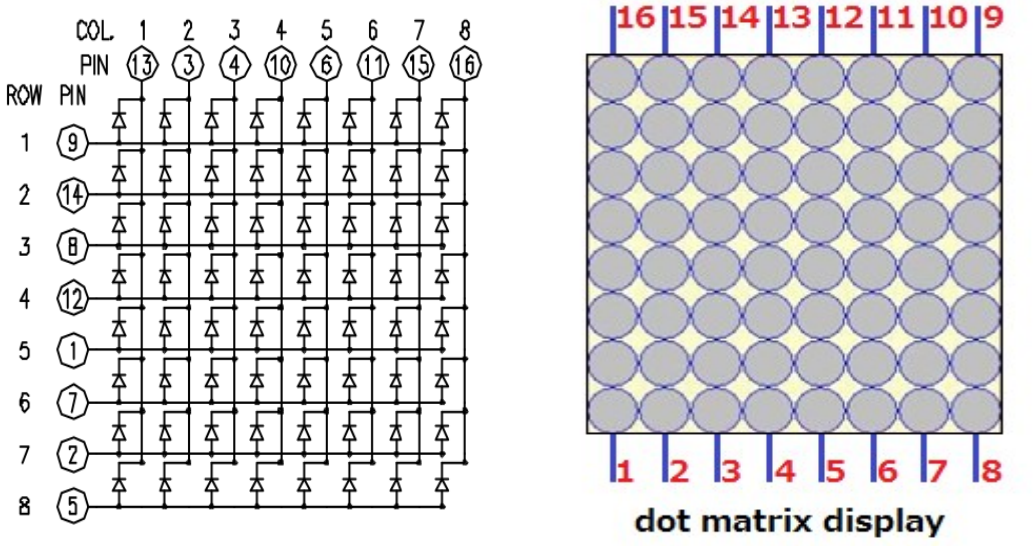
Principle

1. Dot-matrix display

A dot-matrix display is a display device used to display information on machines, clocks, railway departure indicators and many other devices requiring a simple display device of limited resolution.

The display consists of a dot-matrix of lights or mechanical indicators arranged in a rectangular configuration (other shapes are also possible, although not common) such that by switching on or off selected lights, text or graphics can be displayed. A dot-matrix controller converts instructions from a processor into signals which turns on or off lights in the matrix so that the required display is produced.

The internal structure and appearance of the dot-matrix display is as shown in below:



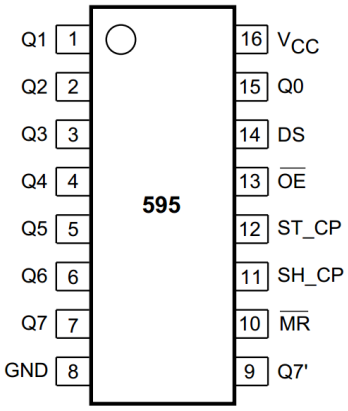
A 8*8 dot-matrix display consists of 64 LEDs, and each LED is placed at the intersection of the lines and columns. When the corresponding row is set as high level and the column is set as low level, then the LED will be lit.

A certain drive current is required for the dot-matrix display. In addition, more pins are needed for connecting dot-matrix display with controller. Thus, to save the Arduino's GPIO, driver IC 74HC595 is used in the experiment.

2. 74HC595

The 74HC595 is an 8-stage serial shift register with a storage register and 3-state outputs. The shift register and storage register have separate clocks. Data is shifted on the positive-going transitions of the SH_CP input. The data in each register is transferred to the storage register on a positive-going transition of the ST_CP input. The shift register has a serial input (DS) and a serial standard output (**Q7'**) for cascading. It is also provided with asynchronous reset (active LOW) for all 8 shift register stages. The storage register has 8 parallel 3-state bus driver outputs. Data in the storage register appears at the output whenever the output enable input (OE) is LOW.

In this experiment, only 3 pins of Arduino are used for controlling a dot-matrix display due to the existence of 74HC595.



The flowing is the function of each pin:

DS: Serial data input

Q0-Q7: 8-bit parallel data output

Q7': Series data output pin, always connected to DS pin of the next 74HC595

OE: Output enable pin, effective at low level, connected to the ground directly

MR: Reset pin, effective at low level, directly connected to 5V high level in practical applications

SH_CP: Shift register clock input

ST_CP: storage register clock input

3. Key function:

●shiftOut()

Shifts out a byte of data one bit at a time. Starts from either the most (i.e. the leftmost) or least (rightmost) significant bit. Each bit is written in turn to a data pin, after which a clock pin is pulsed (taken high, then low) to indicate that the bit is available.

Syntax

shiftOut(dataPin, clockPin, bitOrder, value)

Parameters

dataPin: the pin on which to output each bit (int).

clockPin: the pin to toggle once the dataPin has been set to the correct value.

bitOrder: which order to shift out the bits; either MSBFIRST or LSBFIRST. (Most Significant Bit First, or, Least Significant Bit First)

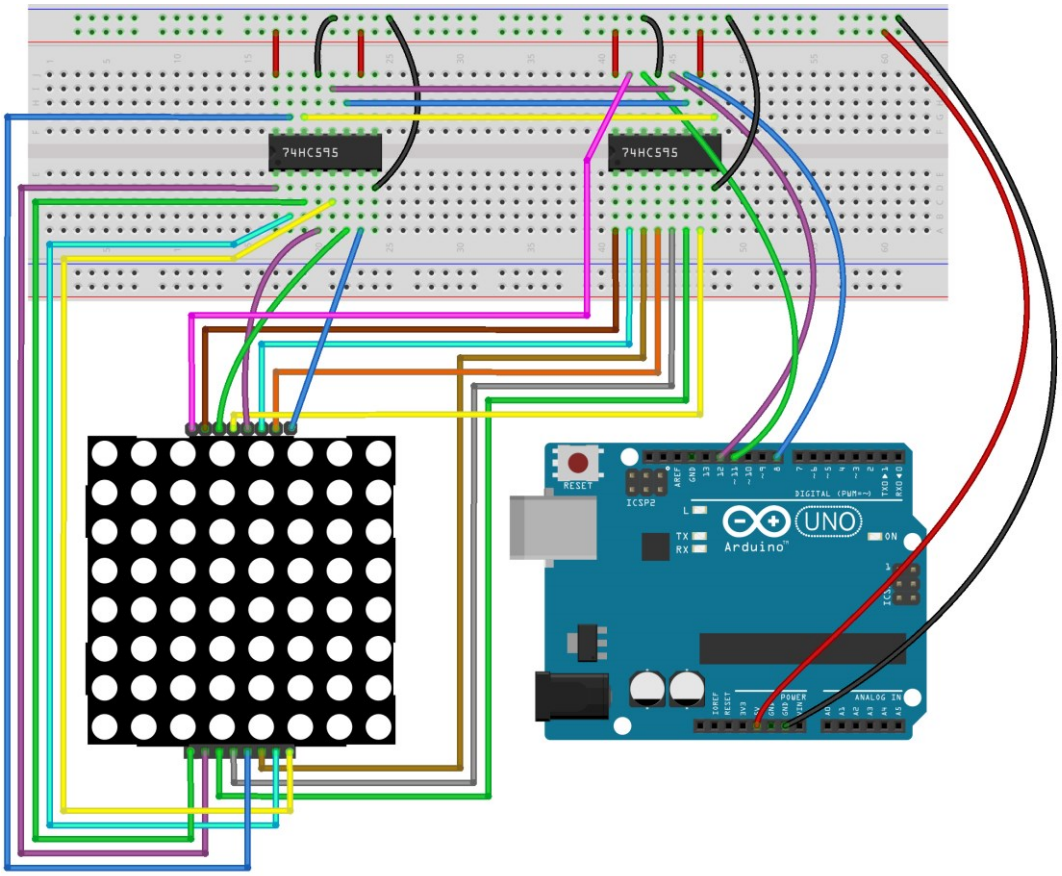
value: the data to shift out. (byte)

Returns

None

Procedures

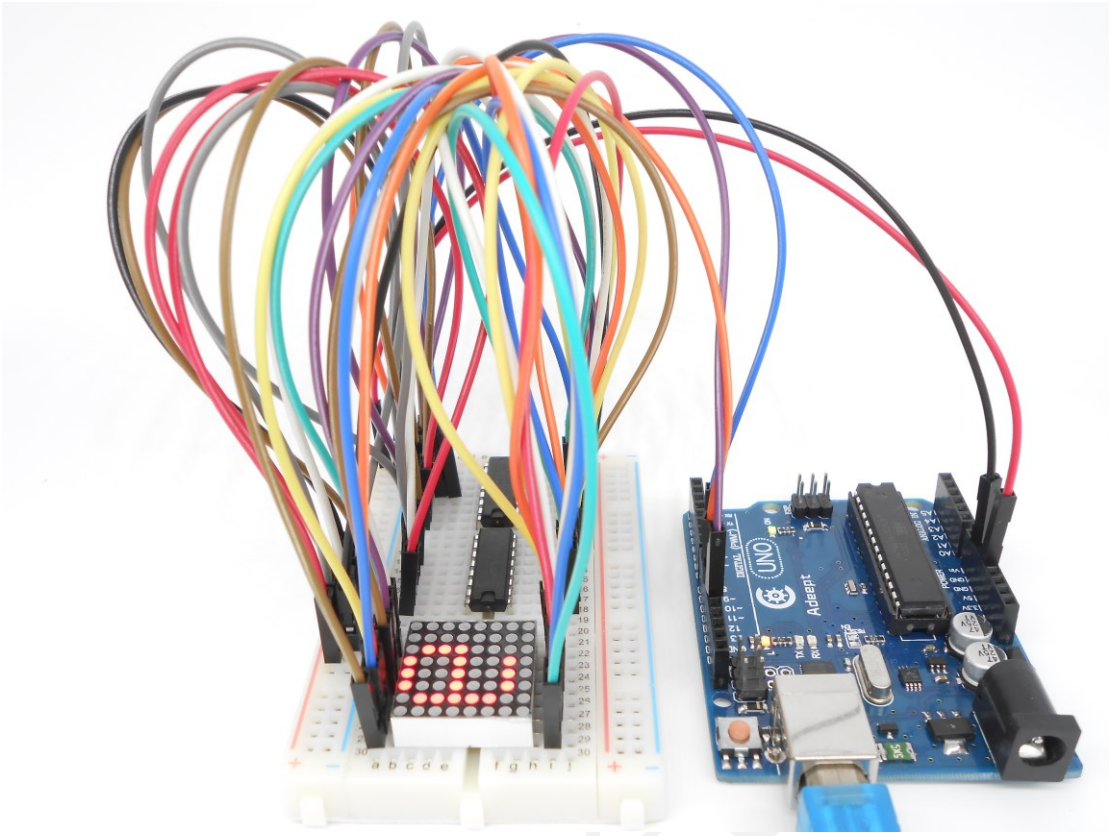
1. Build the circuit (Make sure that the circuit connection is correct and then power, otherwise it may cause the chips to burn.)



2. Program

3. Compile the program and upload to Arduino UNO board

Now, you can see a rolling “Adeept” should be displayed on the dot-matrix display.



Summary

In this experiment, we have not only learned how to operate a dot-matrix display to display numbers and letters, but also learned the basic usage of 74HC595, then you can try operating the dot-matrix display to show other images.

Lesson 10 LCD1602

Overview

In this lesson, we will learn how to use a character display device—LCD1602 on the Arduino platform. First, we make the LCD1602 display a string "Hello Geeks!" scrolling, then display“Adeept”and“www.adeept.com”static.

Requirement

- 1* Arduino UNO
- 1* USB cable
- 1* LCD1602
- 1* 10KΩ Potentiometer
- 1* Breadboard
- Several Jumper wires

Principle

LCD1602 is a kind of character LCD display. The LCD has a parallel interface, meaning that the microcontroller has to manipulate several interface pins at once to control the display. The interface consists of the following pins:

● A register select (RS) pin that controls where in the LCD's memory you're writing data to. You can select either the data register, which holds what goes on the screen, or an instruction register, which is where the LCD's controller looks for instructions on what to do next.

● A Read/Write (R/W) pin that selects reading mode or writing mode

● An Enable pin that enables writing to the registers

● 8 data pins (D0-D7). The state of these pins (high or low) are the bits that you're writing to a register when you write, or the values when you read.

● There's also a display contrast pin (Vo), power supply pins (+5V and Gnd) and LED Backlight (Bklt+ and BKlt-) pins that you can use to power the LCD, control the display contrast, and turn on or off the LED backlight respectively.

The process of controlling the display involves putting the data that form the image of what you want to display into the data registers, then putting instructions in the instruction register. The LiquidCrystal Library simplifies this for you so you don't need to know the low-level instructions.

The Hitachi-compatible LCDs can be controlled in two modes: 4-bit or 8-bit.

The 4-bit mode requires seven I/O pins from the Arduino, while the 8-bit mode requires 11 pins. For displaying text on the screen, you can do most everything in 4-bit mode, so example shows how to control a 2x16 LCD in 4-bit mode.

A potentiometer , informally a pot, is a three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider. If only two terminals are used, one end and the wiper, it acts as a variable resistor or rheostat.

Key function:

●begin()

Specifies the dimensions (width and height) of the display.

Syntax

lcd.begin(cols, rows)

Parameters

lcd: a variable of type LiquidCrystal

cols: the number of columns that the display has

rows: the number of rows that the display has

●setCursor()

Position the LCD cursor; that is, set the location at which subsequent text written to the LCD will be displayed.

Syntax

lcd.setCursor(col, row)

Parameters

lcd: a variable of type LiquidCrystal

col: the column at which to position the cursor (with 0 being the first column)

row: the row at which to position the cursor (with 0 being the first row)

●scrollDisplayLeft()

Scrolls the contents of the display (text and cursor) one space to the left.

Syntax

lcd.scrollDisplayLeft()

Parameters

lcd: a variable of type LiquidCrystal

Example

scrollDisplayLeft() and scrollDisplayRight()

See also

scrollDisplayRight()

●print()

Prints text to the LCD.

Syntax

lcd.print(data)

lcd.print(data, BASE)

Parameters

lcd: a variable of type LiquidCrystal

data: the data to print (char, byte, int, long, or string)

BASE (optional): the base in which to print numbers: BIN for binary (base 2), DEC for decimal (base 10), OCT for octal (base 8), HEX for hexadecimal (base 16).

Returns

byte

print() will return the number of bytes written, though reading that number is optional

●clear()

Clears the LCD screen and positions the cursor in the upper-left corner.

Syntax

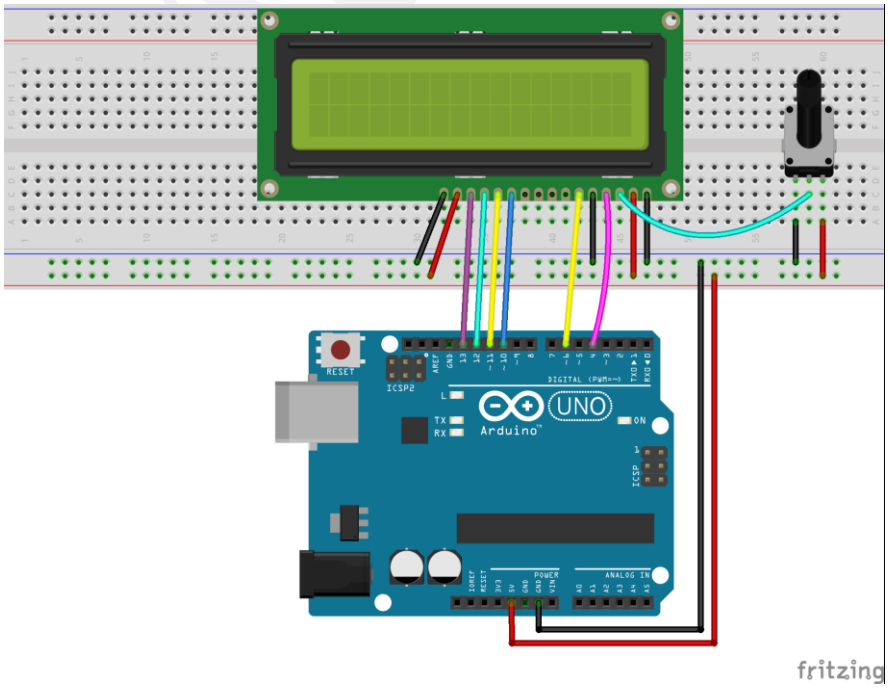
lcd.clear()

Parameters

lcd: a variable of type LiquidCrystal

Procedures

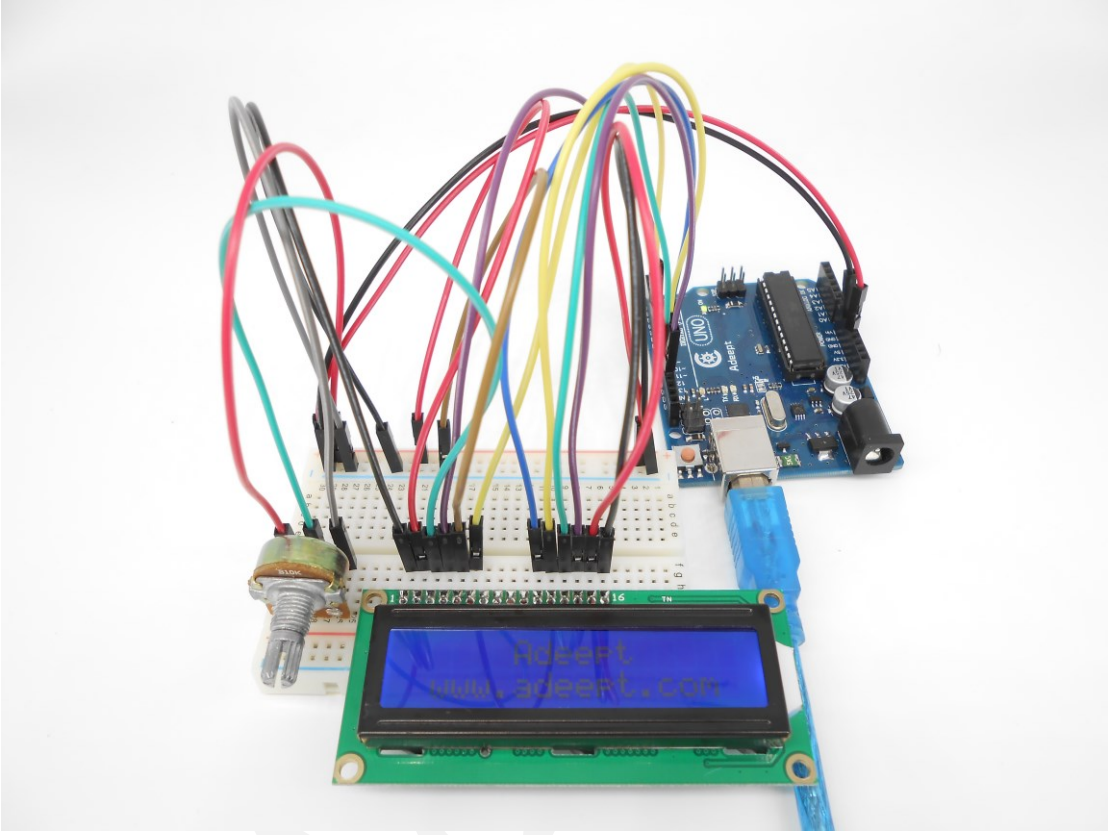
1. Build the circuit



2. Program

3. Compile the program and upload to Arduino UNO board

Now, you can see the string "Hello Geeks!" is shown on the LCD1602 scrolling, and then the string "Adeept" and "www.adeept.com" is displayed on the LCD1602 static.



Summary

I believe that you have already mastered the driver of LCD1602 through this lesson. I hope you can make something more interesting base on this lesson and the previous lesson learned.

Lesson 11 Photoresistor

Overview

In this lesson, we will learn how to measure the light intensity by photoresistor and make the measurement result displayed on the LCD1602.

Requirement

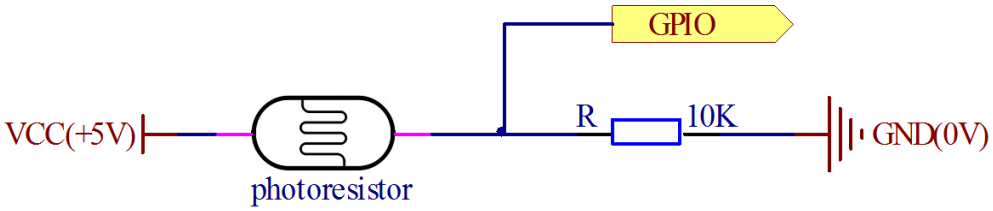
- 1* Arduino UNO
- 1* USB cable
- 1* LCD1602
- 1* Photoresistor
- 1* 10KΩ Resistor
- 1* 10KΩ Potentiometer
- 1* Breadboard
- Several Jumper wires

Principle

A photoresistor is a light-controlled variable resistor. The resistance of a photoresistor decreases with the increasing incident light intensity; in other words, it exhibits photoconductivity. A photoresistor can be applied in light-sensitive detector circuits.

A photoresistor is made of a high resistance semiconductor. In the dark, a photoresistor can have a resistance as high as a few megohms (MΩ), while in the light, a photoresistor can have a resistance as low as a few hundred ohms. If incident light on a photoresistor exceeds a certain frequency, photons absorbed by the semiconductor give bound electrons enough energy to jump into the conduction band. The resulting free electrons (and their hole partners) conduct electricity, thereby lowering resistance. The resistance range and sensitivity of a photoresistor can substantially differ among dissimilar devices. Moreover, unique photoresistors may react substantially differently to photons within certain wavelength bands.

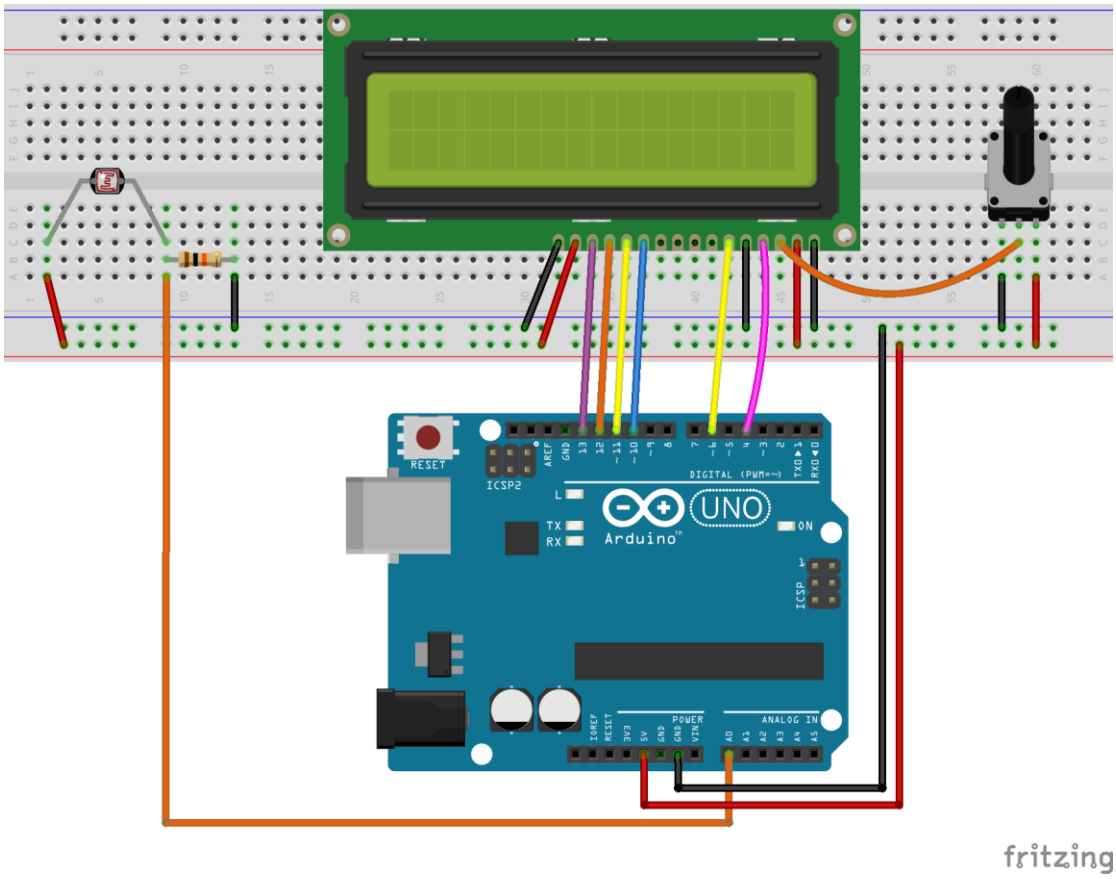
The schematic diagram of this experiment is shown below:



With the increase of the light intensity, the resistance of photoresistor will be decreased. The voltage of GPIO port in the above figure will become high.

Procedures

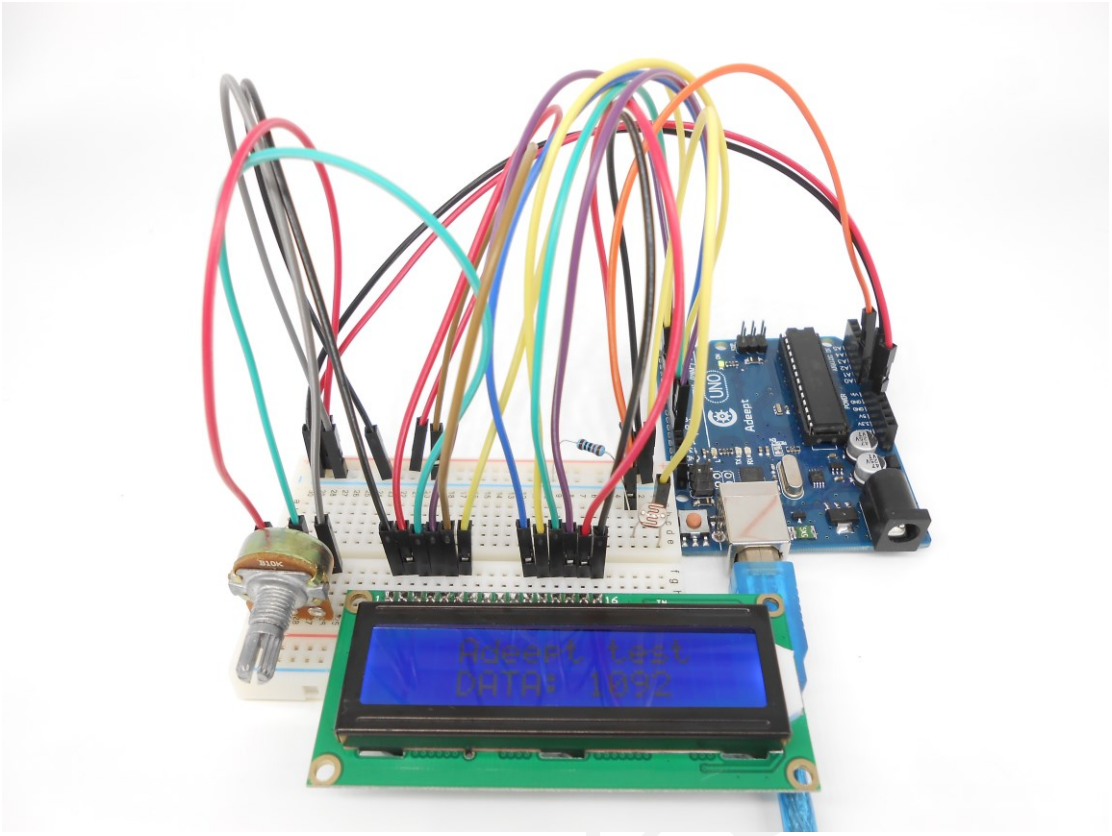
1. Build the circuit



2. Program

3. Compile the program and upload to Arduino UNO board

Now, when you try to block the light towards the photoresistor, you will find that the value displayed on the LCD1602 will be reduced. Otherwise, when you use a powerful light to irradiate the photoresistor, the value displayed on the LCD1602 will be increased.



Summary

By learning this lesson, we have learned how to detect surrounding light intensity with the photoresistor. You can play your own wisdom, and make more originality based on this experiment and the former experiment.

Lesson 12 Serial Port

Overview

In this lesson, we will program the Arduino UNO to achieve function of send and receive data through the serial port. The Arduino receiving data which send from PC, and then controlling an LED according to the received data, then return the state of LED to the PC's serial port monitor.

Requirement

- 1* Arduino UNO
- 1* USB cable
- 1* LED
- 1* 220Ω Resistor
- 1* Breadboard
- Several Jumper wires

Principle

1. Serial ports

Used for communication between the Arduino board and a computer or other devices. All Arduino boards have at least one serial port (also known as a UART or USART). It communicates on digital pins 0 (RX) and 1 (TX) as well as with the computer via USB. Thus, if you use these functions, you cannot also use pins 0 and 1 for digital input or output.

You can use the Arduino environment's built-in serial monitor to communicate with an Arduino board. Click the serial monitor button in the toolbar and select the same baud rate used in the call to begin().

To use these pins to communicate with your personal computer, you will need an additional USB-to-serial adaptor, as they are not connected to the UNO's USB-to-serial adaptor. To use them to communicate with an external TTL serial device, connect the TX pin to your device's RX pin, the RX to your device's TX pin, and the ground of your UNO to your device's ground. (Don't connect these pins directly to an RS232 serial port; they operate at +/- 12V and can damage your Arduino board.)

2. Key function

●begin()

Sets the data rate in bits per second (baud) for serial data transmission. For

communicating with the computer, use one of these rates: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. You can, however, specify other rates - for example, to communicate over pins 0 and 1 with a component that requires a particular baud rate.

Syntax

Serial.begin(speed)

Parameters

speed: in bits per second (baud) - long

Returns

nothing

● print()

Prints data to the serial port as human-readable ASCII text. This command can take many forms. Numbers are printed using an ASCII character for each digit. Floats are similarly printed as ASCII digits, defaulting to two decimal places. Bytes are sent as a single character. Characters and strings are sent as is. For example:

Serial.print(78) gives “78”

Serial.print(1.23456) gives “1.23”

Serial.print('N') gives “N”

Serial.print(“Hello world.”) gives “Hello world.”

An optional second parameter specifies the base (format) to use; permitted values are BIN (binary, or base 2), OCT (octal, or base 8), DEC (decimal, or base 10), HEX (hexadecimal, or base 16). For floating point numbers, this parameter specifies the number of decimal places to use. For example:

Serial.print(78, BIN) gives “1001110”

Serial.print(78, OCT) gives “116”

Serial.print(78, DEC) gives “78”

Serial.print(78, HEX) gives “4E”

Serial.println(1.23456, 0) gives “1”

Serial.println(1.23456, 2) gives “1.23”

Serial.println(1.23456, 4) gives “1.2346”

You can pass flash-memory based strings to Serial.print() by wrapping them with F(). For example :

Serial.print(F(“Hello World”))

To send a single byte, use Serial.write().

Syntax

Serial.print(val)

Serial.print(val, format)

Parameters

val: the value to print - any data type format: specifies the number base (for integral data types) or number of decimal places (for floating point types)

Returns

byte print() will return the number of bytes written, though reading that number is optional

● println()

Prints data to the serial port as human-readable ASCII text followed by a carriage return character (ASCII 13, or '\r') and a newline character (ASCII 10, or '\n'). This command takes the same forms as Serial.print().

Syntax

Serial.println(val)

Serial.println(val, format)

Parameters

val: the value to print - any data type

format: specifies the number base (for integral data types) or number of decimal places (for floating point types)

Returns

byte

println() will return the number of bytes written, though reading that number is optional

● read()

Reads incoming serial data. read() inherits from the Stream utility class.

Syntax

Serial.read()

Parameters

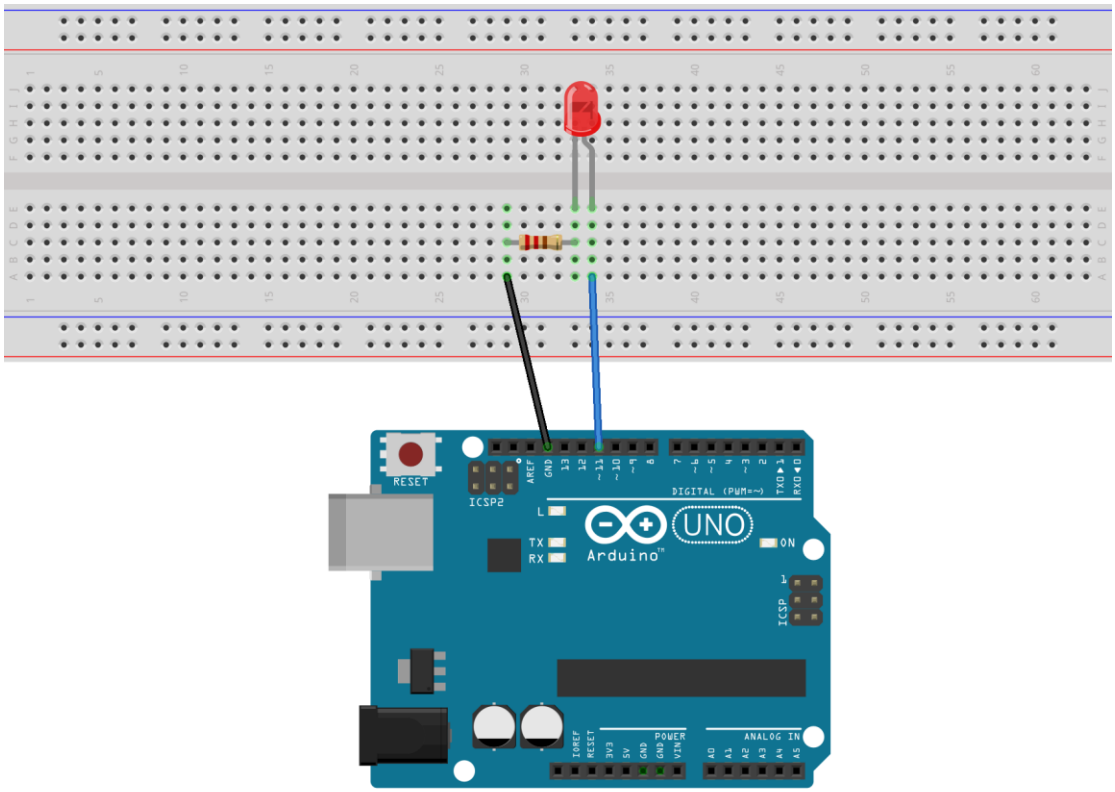
None

Returns

the first byte of incoming serial data available (or -1 if no data is available) - int

Procedures

1. Build the circuit

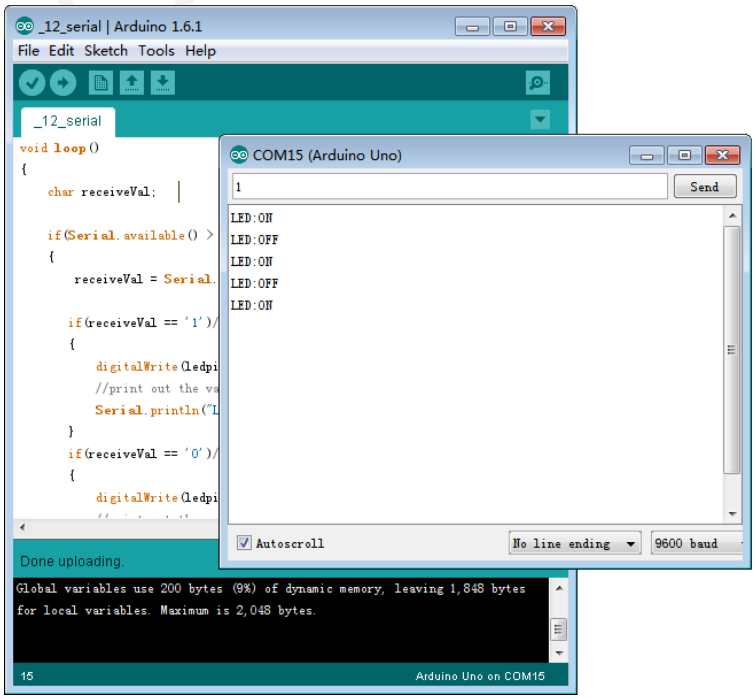


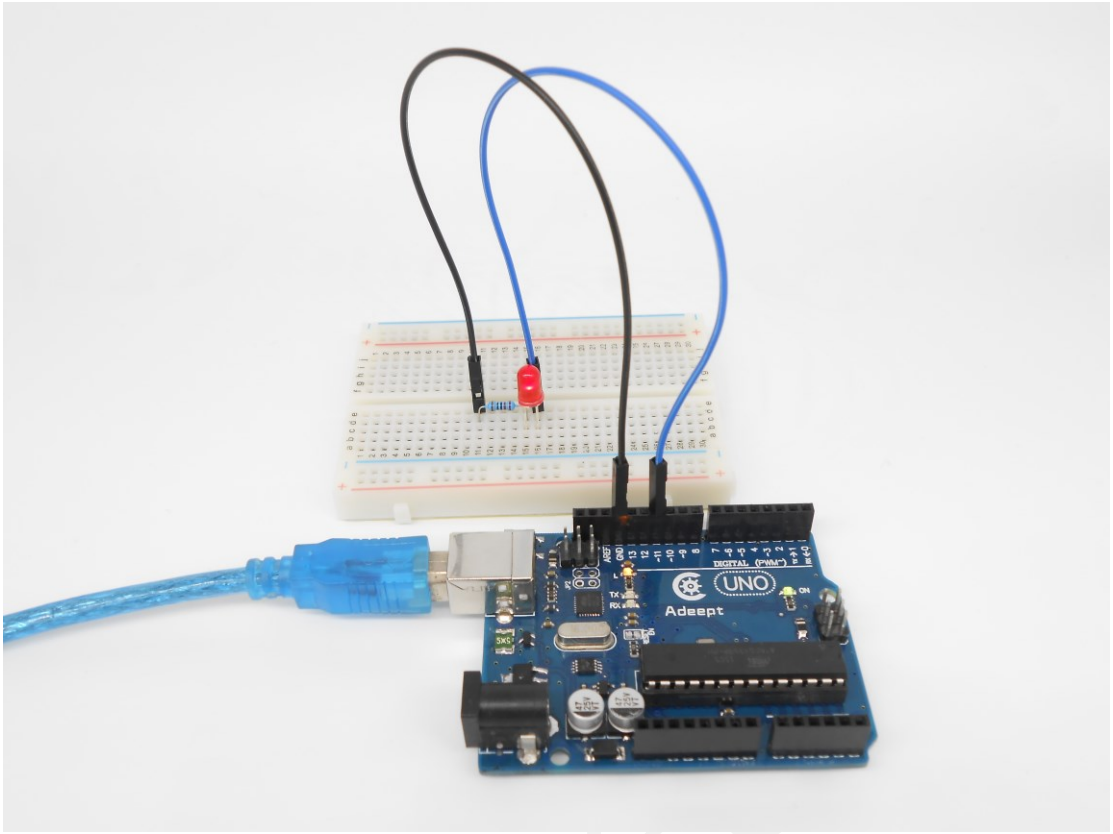
2. Program

3. Compile the program and upload to Arduino UNO board

Open the port monitor, and then select the appropriate baud rate according to the program.

Now, if you send a character'1'or'0'on the serial monitor, the state of LED will be lit or gone out.





Summary

Through this lesson, you should have understood that the computer can send data to Arduino UNO via the serial port, and then control the state of LED. I hope you can use your head to make more interesting things based on this lesson.

Lesson 13 Frequency meter

Overview

In this lesson, we will make a simple frequency meter with the Arduino UNO. We will acquire the frequency of square wave which is generated by 555 timer, and then send the result to serial monitor through USB port.

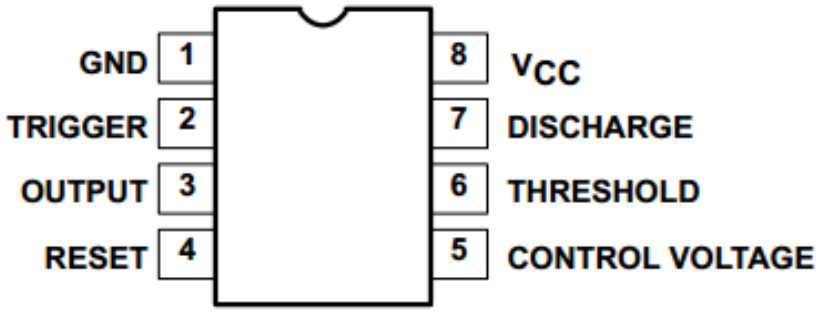
Requirement

- 1* Arduino UNO
- 1* USB cable
- 1* NE555
- 1* 10KΩ Resistor
- 1* 10KΩ Potentiometer
- 2* 104 Capacitor
- 1* Breadboard
- Several Jumper wires

Principle

1. NE555

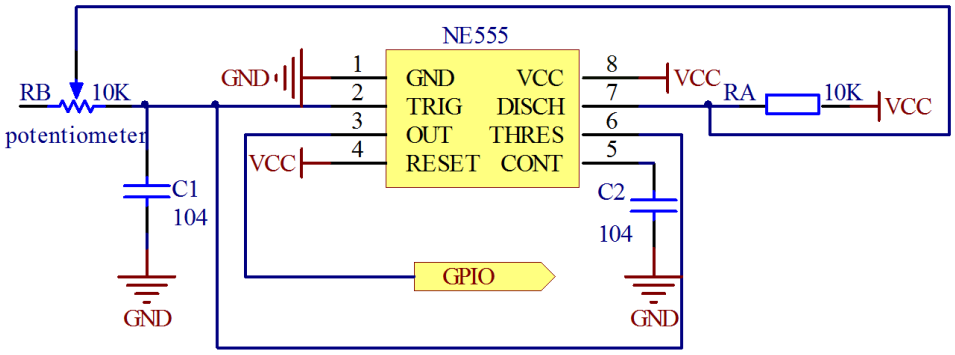
The 555 integrated circuit is originally used as a timer, and that is why it is called 555 timer or 555 time-base circuit. It is widely used in various electronic products because of its reliability, convenience and low price. There are dozens of components in the 555 integrated circuit, such as divider, comparator, basic R-S trigger, discharge tube, buffer and so on. It is a complex circuit and a hybrid composed of analog and digital circuit.



As shown in the above picture, the 555 integrated circuit is dual in-line with 8 pins package(DIP). Thereinto:

Pin 6 is the THRESHOLD for the input of upper comparator;
Pin 2 is TRIGGER for the input of lower comparator;
Pin 3 is the OUTPUT having two states of 0 and 1 decided by the input electrical level;
Pin 7, the DISCHARGE which has two states of suspension and ground connection also decided by input, is the output of the internal discharge tube;
Pin 4 is the RESET that outputs low level when supplied low voltage level;
Pin 5 is the CONTROL VOLTAGE that can change the upper and lower level trigger value;
Pin 8 (Vcc) is the power supply;
Pin 1(GND) is the ground.

The circuit schematic diagram used in the experiment is shown in below:



The circuit can generate a square wave signal that the frequency is adjustable. The frequency can be calculated by the formula:

$$\text{Frequency} \approx \frac{1.44}{(R_A + 2R_B) * C}$$

2. Key functions **:**

●pulseIn()

Reads a pulse (either HIGH or LOW) on a pin. For example, if value is HIGH, pulseIn() waits for the pin to go HIGH, starts timing, then waits for the pin to go LOW and stops timing. Returns the length of the pulse in microseconds. Gives up and returns 0 if no pulse starts within a specified time out.

The timing of this function has been determined empirically and will probably show errors in longer pulses. Works on pulses from 10 microseconds to 3 minutes in length.

Syntax

pulseIn(pin, value)

pulseIn(pin, value, timeout)

Parameters

pin: the number of the pin on which you want to read the pulse. (int)

value: type of pulse to read: either HIGH or LOW. (int)

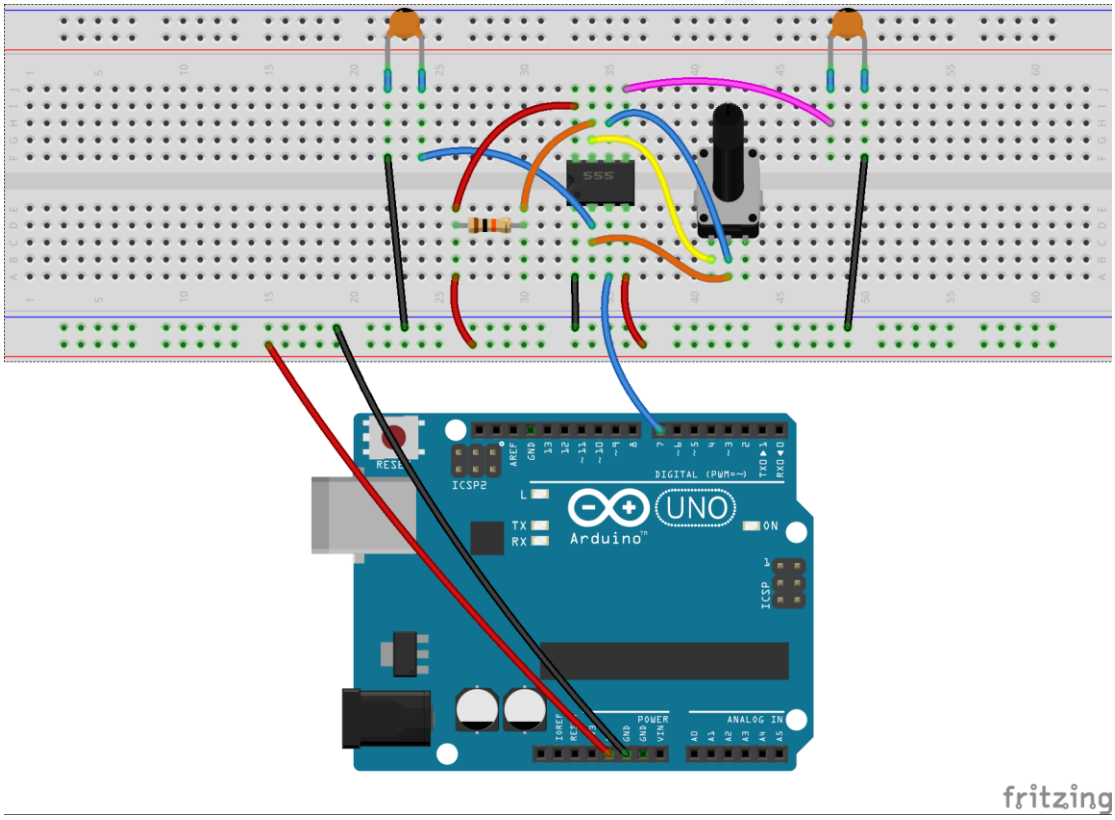
timeout (optional): the number of microseconds to wait for the pulse to start; default is one second (unsigned long)

Returns

the length of the pulse (in microseconds) or 0 if no pulse started before the timeout (unsigned long)

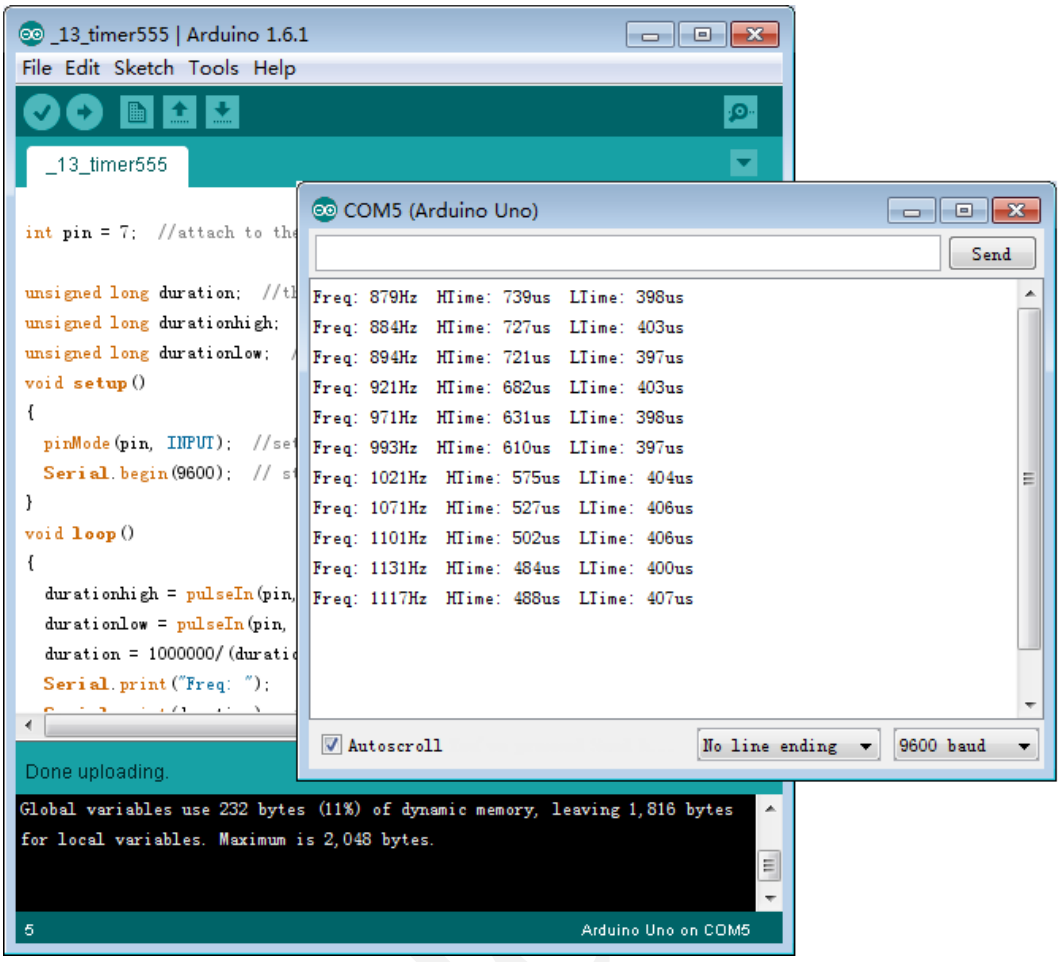
Procedures

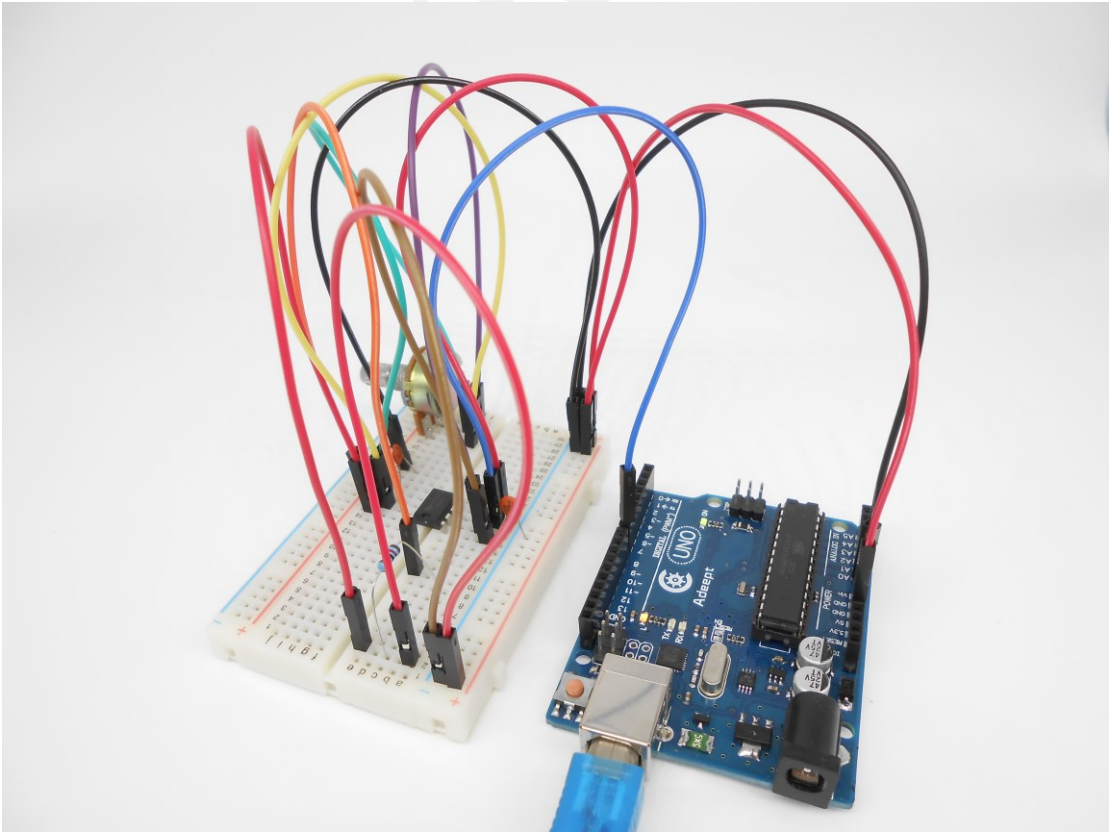
1. Build the circuit (Make sure that the circuit connection is correct and then power, otherwise it may cause the chips to burn.)



2. Program
3. Compile the program and upload to Arduino UNO board

Now, when you rotating the potentiometer knob, the value of square wave's frequency printed on the serial monitor will be changed.


_13_timer555 | Arduino 1.6.1
File Edit Sketch Tools Help
_13_timer555
int pin = 7; //attach to the
unsigned long duration; //t
unsigned long durationhigh;
unsigned long durationlow;
void setup()
{
pinMode(pin, INPUT); //se
Serial.begin(9600); // s
}
void loop()
{
durationhigh = pulseIn(pin,
durationlow = pulseIn(pin,
duration = 1000000/(durati
Serial.print("Freq: ");
COM5 (Arduino Uno)
Send
Freq: 879Hz HTime: 739us LTime: 398us
Freq: 884Hz HTime: 727us LTime: 403us
Freq: 894Hz HTime: 721us LTime: 397us
Freq: 921Hz HTime: 682us LTime: 403us
Freq: 971Hz HTime: 631us LTime: 398us
Freq: 993Hz HTime: 610us LTime: 397us
Freq: 1021Hz HTime: 575us LTime: 404us
Freq: 1071Hz HTime: 527us LTime: 406us
Freq: 1101Hz HTime: 502us LTime: 406us
Freq: 1131Hz HTime: 484us LTime: 400us
Freq: 1117Hz HTime: 488us LTime: 407us
Autoscroll
No line ending
9600 baud
Done uploading.
Global variables use 232 bytes (11%) of dynamic memory, leaving 1,816 bytes
for local variables. Maximum is 2,048 bytes.
5
Arduino Uno on COM5



Summary

By learning this lesson, I believe that you have mastered the principle of timer and you can make a simple frequency meter by yourself. You can display the value of square frequency to a LCD1602 by modifying the code we provided.

Lesson 14 A Simple Voltmeter

Overview

In this lesson, we will make a simple voltmeter with Arduino UNO and LCD1602, the range of this voltmeter is 0~5V. Then, we will measure the voltage of the potentiometer's adjustment end with the simple voltmeter and display it on the LCD1602.

Requirement

- 1* Arduino UNO
- 1* USB Cable
- 1* LCD1602
- 2* Potentiometer
- 1* Breadboard
- Several Jumper wires

Principle

The basic principle of this experiment: Converting the analog voltage that the Arduino collected to digital quantity by the ADC(analog-to-digital converter) through programming, then display the voltage on the LCD1602.

Connect the three wires from the potentiometer to your Arduino board. The first goes to ground from one of the outer pins of the potentiometer. The second goes from analog input 0 to the middle pin of the potentiometer. The third goes from 5 volts to the other outer pin of the potentiometer.

By turning the shaft of the potentiometer, you change the amount of resistance on either side of the wiper which is connected to the center pin of the potentiometer. This changes the voltage at the center pin. When the resistance between the center and the side connected to 5 volts is close to zero (and the resistance on the other side is close to 10 kilohms), the voltage at the center pin nears 5 volts. When the resistances are reversed, the voltage at the center pin nears 0 volts, or ground. This voltage is the analog voltage that you're reading as an input.

The Arduino has a circuit inside called an analog-to-digital converter that reads this changing voltage and converts it to a number between 0 and 1023. When the shaft is turned all the way in one direction, there are 0 volts going to the pin, and the input value is 0. When the shaft is turned all the way in the

opposite direction, there are 5 volts going to the pin and the input value is 1023. In between, analogRead( ) returns a number between 0 and 1023 that is proportional to the amount of voltage being applied to the pin.

Key functions:

- analogRead()

Reads the value from the specified analog pin. The Arduino board contains a 6 channel (8 channels on the Mini and Nano, 16 on the Mega), 10-bit analog to digital converter. This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023. This yields a resolution between readings of: 5 volts / 1024 units or, 0.0049 volts (4.9 mV) per unit. The input range and resolution can be changed using analogReference( ).

It takes about 100 microseconds (0.0001 s) to read an analog input, so the maximum reading rate is about 10,000 times a second.

Syntax

analogRead(pin)

Parameters

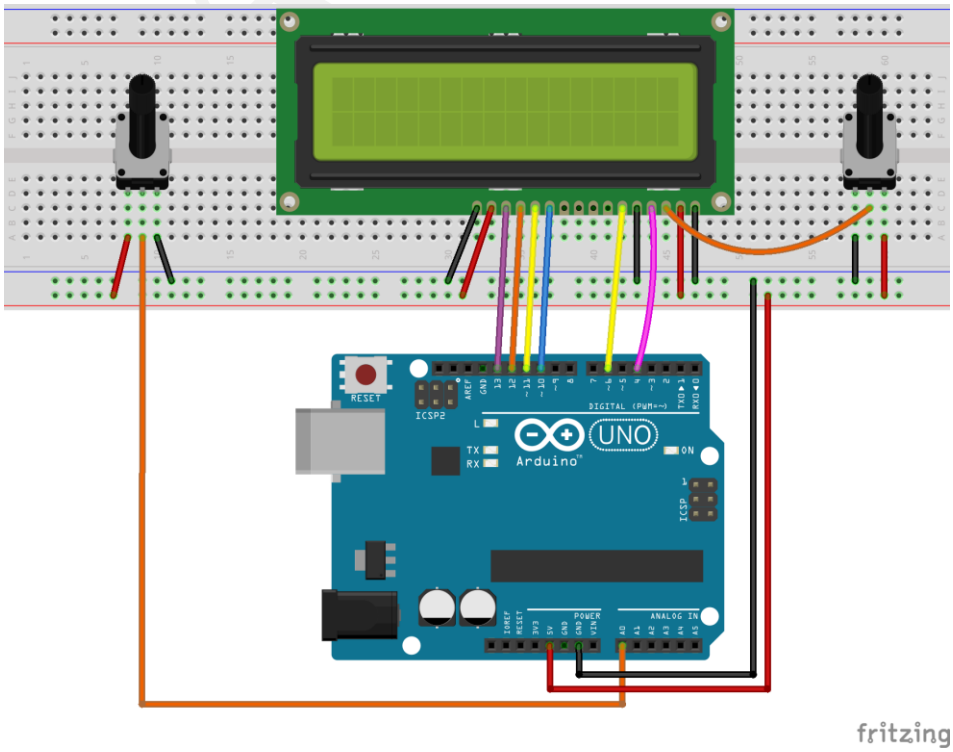
pin: the number of the analog input pin to read from (0 to 5 on most boards, 0 to 7 on the Mini and Nano, 0 to 15 on the Mega)

Returns

int (0 to 1023)

Procedures

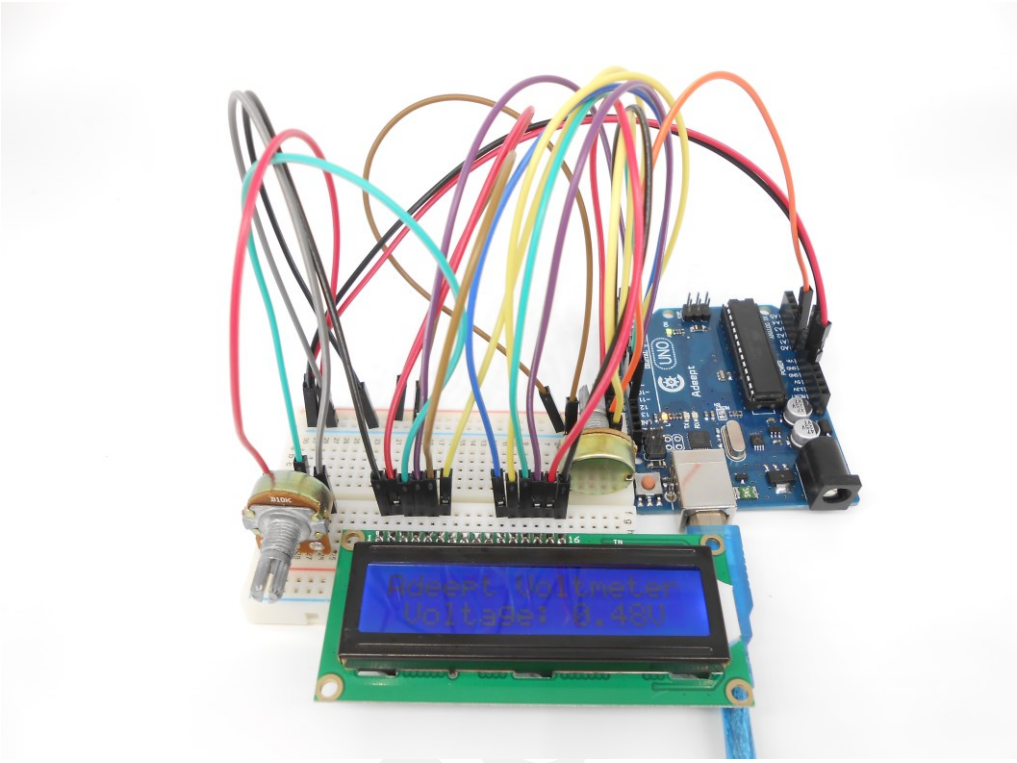
1. Build the circuit



2. Program

3. Compile the program and upload to Arduino UNO.

Now, when you turning the shaft of the potentiometer, you will see the voltage displayed on the LCD1602 will be changed.



Summary

The substance of voltmeter is reading analog voltage which input to ADC inside. Through this course, I believe that you have mastered how to read analog value and how to make a simple voltmeter with Arduino.

Lesson 15 DC motor

Overview

In this comprehensive experiment, we will learn how to control the state of DC motor with Arduino, and the state will be displayed through the LED at the same time. The state of DC motor includes its forward, reverse, acceleration, deceleration and stop.

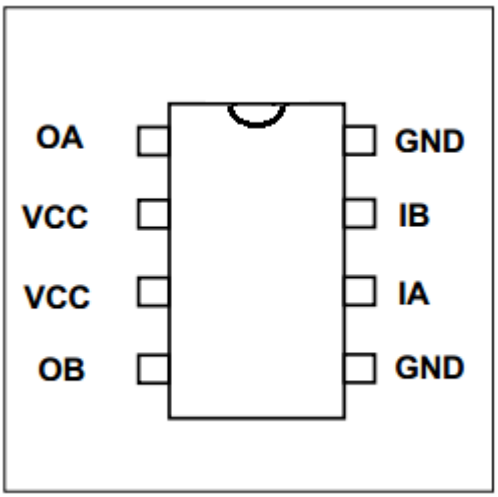
Requirement

- 1* Arduino UNO
- 1* USB Cable
- 1* L9110 DC Motor driver
- 1* DC motor
- 1* Battery holder
- 1* Breadboard
- Several Jumper wires

Principle

1. L9110

L9110 is a driver chip which is used to control and drive motor. The chip has two TTL/CMOS compatible input terminals, and possesses the property of anti-interference: it has high current driving capability, two output terminals that can directly drive DC motor, each output port can provide 750~800mA dynamic current, and its peak current can reach 1.5~2.0A; L9110 is widely applied to various motor drives, such as toy cars, stepper motor, power switches and other electric circuits.



OA, OB: These are used to connect the DC motor.

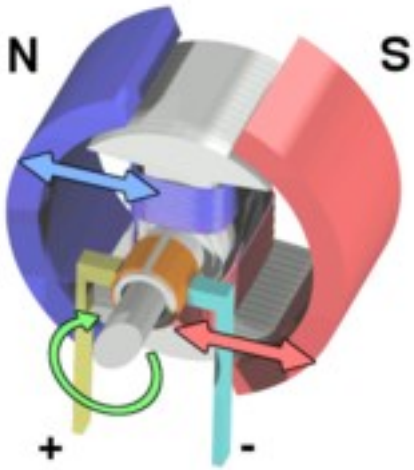
VCC: Power supply (+5V)

GND: The cathode of the power supply (Ground).

IA, IB: The input terminal of drive signal.

2. DC motor

A DC motor is any of a class of electrical machines that converts direct current electrical power into mechanical power. The most common types rely on the forces produced by magnetic fields. Nearly all types of DC motors have some internal mechanism, either electromechanical or electronic, to periodically change the direction of current flow in part of the motor. Most types produce rotary motion; a linear motor directly produces force and motion in a straight line.



DC motors were the first type widely used, since they could be powered from existing direct-current lighting power distribution systems. A DC motor's speed can be controlled over a wide range, using either a variable supply voltage or by changing the strength of current in its field windings. Small DC motors are used in tools, toys, and appliances. The universal motor can operate on direct current but is a lightweight motor used for portable power tools and appliances.

3. Key functions

● switch / case statements

Like if statements, switch···case controls the flow of programs by allowing programmers to specify different code that should be executed in various conditions. In particular, a switch statement compares the value of a variable to the values specified in case statements. When a case statement is found whose value matches that of the variable, the code in that case statement is run.

The break keyword exits the switch statement, and is typically used at the end of each case. Without a break statement, the switch statement will continue executing the following expressions ("falling-through") until a break, or the end of the switch statement is reached.

Example

```
switch (var) {
    case 1:
      //do something when var equals 1
      break;
    case 2:
      //do something when var equals 2
      break;
    default:
      // if nothing else matches, do the default
      // default is optional
  }
```

Syntax

```
switch (var) {
  case label:
```

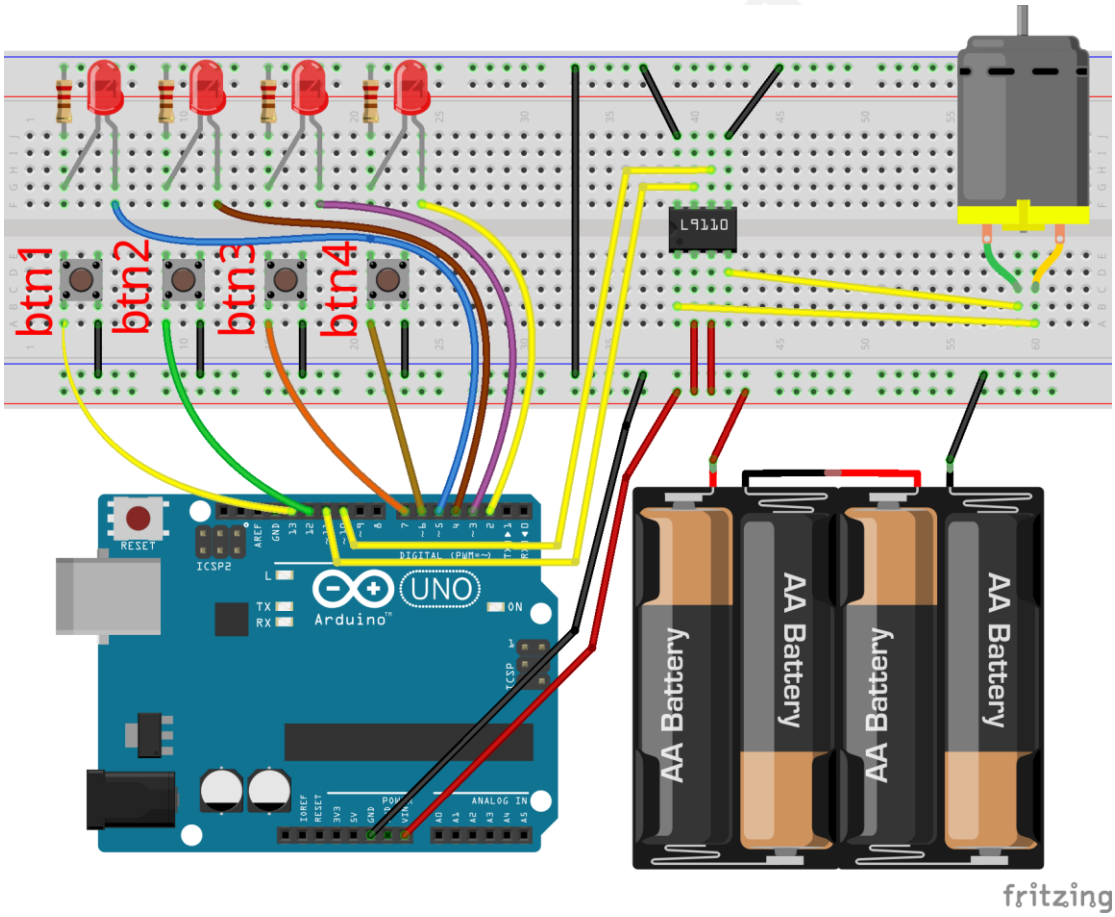
```
    // statements
    break;
  case label:
    // statements
    break;
  default:
    // statements
}
```

Parameters

var: the variable whose value to compare to the various cases label: a value to compare the variable to

Procedures

1. Build the circuit (Make sure that the circuit connection is correct and then power, otherwise it may cause the chips to burn.)

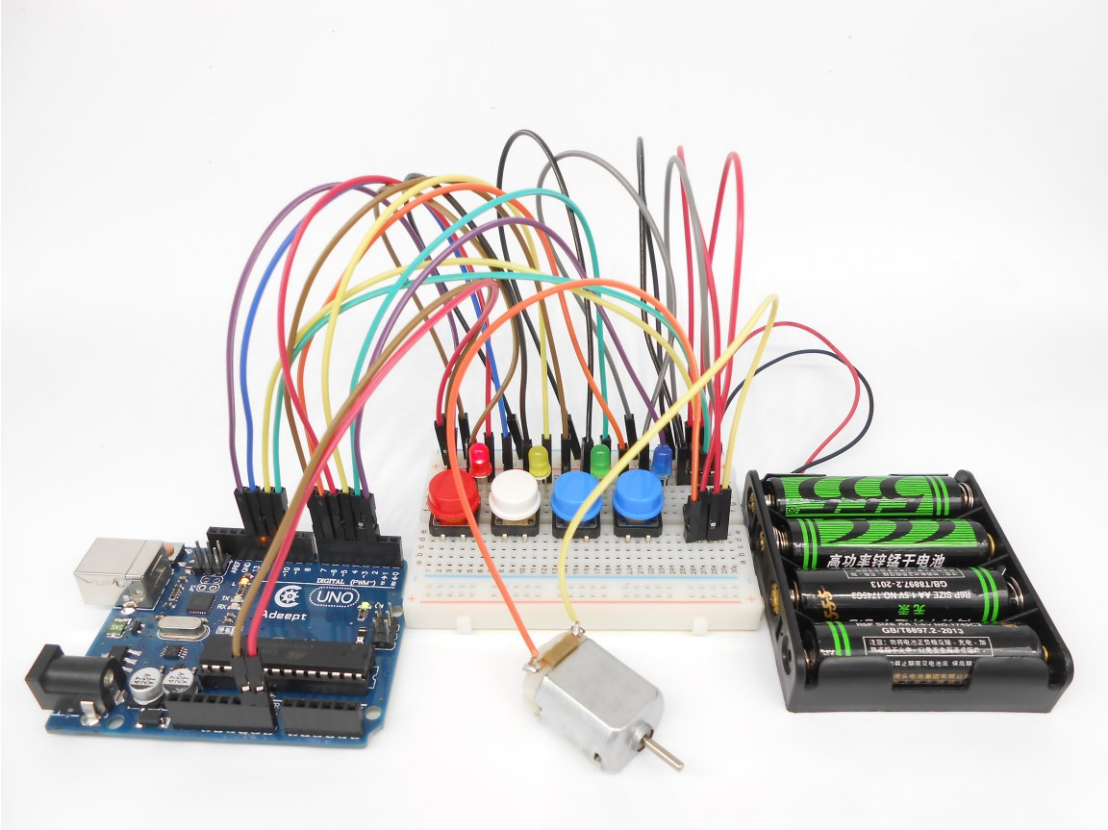


2. Program

3. Compile the program and upload to Arduino UNO board

Press the btn1 button to stop or run the DC motor; press the btn2 button to forward or reverse the DC motor; Press the btn3 button to accelerate the DC motor; Press the btn4 button to decelerate the DC motor. When one of the four

buttons is pressed, their corresponding LED will be flashing which prompts that the current button is clicked.



Summary

I think you must have grasped the basic theory and programming of the DC motor after studying this experiment. You not only can forward and reverse it, but also can regulate its speed. Besides, you can do some interesting applications with the combination of this course and your prior knowledge.

Adeept

Sharing Perfects Innovation

E-mail: support@adeept.com
website: www.adeept.com