TURN ON YOUR CREATIVITY

THE FRANZIS ARDUINO™ TUTORIAL KIT

MANUAL

FRANZIS

ULLI SOMMER

TURN ON YOUR CREATIVITY

THE FRANZIS ARDUINO™ TUTORIAL KIT

ORIGINAL
ARDUINO UNO
AND 20 OTHER
COMPONENTS
FOR 65 PROJECTS

ISBN: 978-3-645-65279-7

Translation and DTP: G&U Language & Publishing Services GmbH
Layout: bora-dtp

Table of Contents

Preface

With many microcontroller systems, you have to work through countless data sheets that are incomprehensible for beginners. The programming interfaces are very complex and devised for professional developers with years of experience in programming microcontrollers. Thus, the access to the world of microcontrollers is unnecessarily made complicated.

The Arduino system, however, is an easily comprehensible open-source platform that is easy to understand. It is based on a microcontroller board with an Atmel AVR controller and a simple programming environment. For the human-machine interaction, you can attach a variety of analog and digital sensors that capture ambient quantities and pass the data to the microcontroller where they are processed. The program causes the creation of new analog or digital output data. There is no limit to the creativity of the developer. Whether you want to build a control system for your home or a beautiful LED lamp with changing colours: The Arduino allows even beginners from another background to write functional programs and to put their own ideas into practise.

The smooth cooperation of hardware and software is the basis for »physical computing« – the linking of the real world to the bits-and-bytes world of the microcontroller.

This tutorial kit conveys the basics of electronics and Arduino programming and shows in a plain way how to implement your own ideas.

Ulli Sommer

The CD in the Tutorial Kit

This tutorial kit contains a CD with several programs, tools, data sheets, and examples. The CD is intended to help you in working with this book. All examples in this book are contained on the CD as well.

The contents of the CD

The contents of the CD

- Arduino IDE (Integrated Development Environment)
- Sample program code
- Several tools
- Data sheets
- Circuit diagrams

GPL (General Public Licence)

You can share your own programs on the internet with other users. The sample programs are provided under the open-source GPL licence (General Public Licence). This means that you have the right to modify, publish, and share the programs according to the conditions of the GPL, provided that you make them available under the same licence terms.

System Requirements

- Windows XP (32- or 64-bit) or newer; or:
- Linux (32- or 64-bit); or:
- Mac OS X.
- CD drive
- Java

More information can be found on the following websites:

Further reading

- *www.arduino.cc*
- *www.fritzing.org*
- *www.processing.org*

Updates and Support

The Arduino IDE is continually developed further. You can download any updates free of charges at the following website:

hptt://arduino.cc

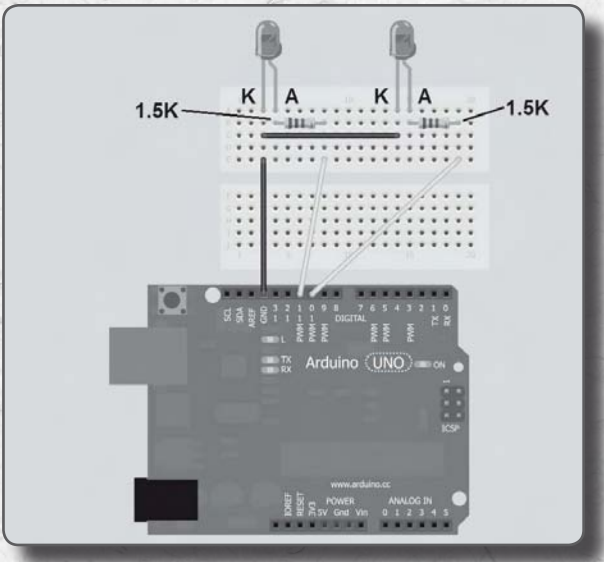
Warning! ***Eye protection in handling LEDs***

Never look directly to an LED at a short distance! This could damage your retina! This is especially true for bright LEDs in a clear housing und even more for Power LEDs. The perceived brightness of white, blue, purple, and ultraviolet LEDs gives a false impression of the real danger for your eyes. Always exercise extreme caution when using convergent lenses. Operate any LEDs according to the instructions, and never use higher currents.

TURN ON YOUR CREATIVITY

FRANZIS
ARDUINO

MORE EXPERIMENTS WITH THE ARDUINO



10

Even more experiments with the Arduino

Now that you have worked through the fundaments and made yourself familiar with programming the Arduino, you can start with hands-on experiments. The following projects build up on the basic knowledge you have gained in the previous chapter and extend it with new functions and programming options.

It is assumed that you already understand the program statements described so far, so you can implement the examples.

The basic mode of operation is given for all the examples, but there will be no further explanation of familiar statements. If you do not have a firm grasp on some the commands, you shall tackle them again.

Work with the breadboard

In most of the following experiments, you will need the breadboard and the components included in the tutorial kit. The circuits are deliberately kept simple. You can easily follow the current flow on the breadboard without a circuit diagram.

10.1 | LED dimmer

Build an LED dimmer for your living room

In the previous chapter, you have become acquainted with the analog PWM output and `analogWrite`. This allows you to build a dimmer that controls the brightness of an LED. Use a red LED at analog output 3 for the next experiment. If you want to use more powerful LEDs like those by Luxeon, you have to add a transistor to the analog output in order to increase the small current of the microcontroller to the amount needed by the LED.

The example project already uses a transistor as an amplifier and shows how to use it on a digital PWM output. In this experiment, we only use the low-current LED included in the tutorial kit, but you can apply a greater load to the collector circuit like the high-power LED mentioned above or a small lightbulb for a flashlight (max. 100 mA). The push-buttons S1 (brighter) and S2 (darker) control the duty cycle of the PWM output and thus the brightness. The transistor relieves the digital pin. Only a very small current (ca. 300 times smaller than the load) flow to the base. This current is amplified by the transistor that uses the small base current to switch the larger collector current.

Required parts for the experiment

- 1 x microcontroller board Arduino Uno
- 1 x red LED
- 2 x push-buttons
- 1 x transistor BC548C
- 1 x 1.5 kΩ resistor
- 1 x 4.7 kΩ resistor
- 5 x jump wire, ca. 10 cm
- 2 x jump wire, ca. 5 cm

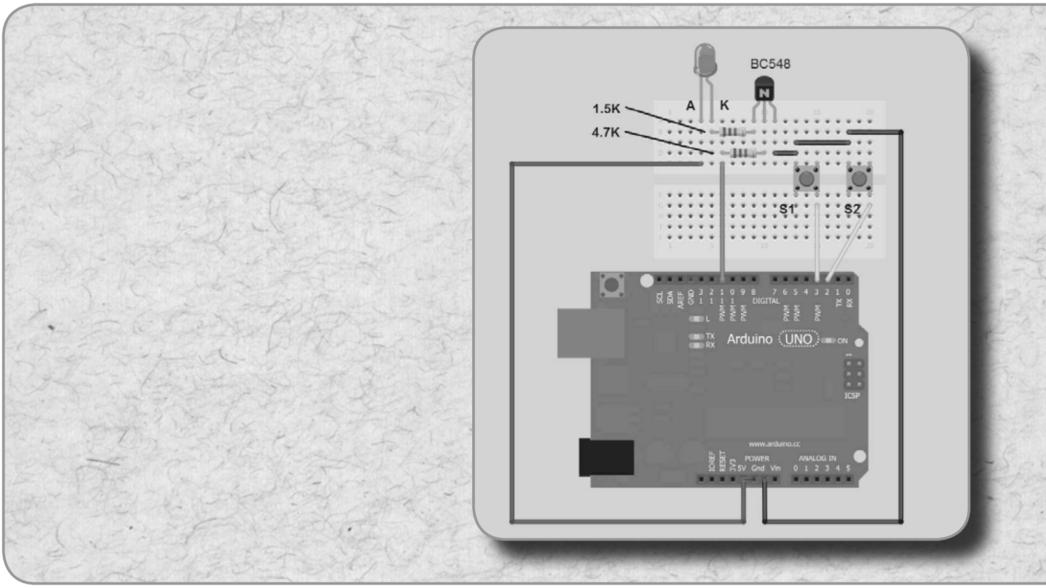


Figure 10.1: Diagram of the set-up for an LED dimmer with transistor

Example: LED dimmer

Time needed: 15 min
Difficulty: 2

```
// Franzis Arduino
// LED-Dimmer

int brightness=0;
int SW1=3;
int SW2=2;
int LED=11;

void setup()
{
   pinMode(SW1,INPUT);
   digitalWrite(SW1,HIGH);
   pinMode(SW2,INPUT);
   digitalWrite(SW2,HIGH);
}

```

```
void loop()
{
   if(!digitalRead(SW1)&&digitalRead(SW2))
   {
      if(brightness<255)brightness++;
      analogWrite(LED,brightness);
      delay(10);
   }
   else if(digitalRead(SW1)&&!digitalRead(SW2))
   {
      if(brightness>0)brightness--;
      analogWrite(LED,brightness);
      delay(10);
   }
}
```

This example also demonstrates the usage of logical operators like `!` and `&&` in an `if` query. These comparison operations cause the push-buttons to lock each other, so that nothing happens when you press both of them simultaneously.

```
if(!digitalRead(SW1)&&digitalRead(SW2))
{
   // statement 1
}
else if(digitalRead(SW1)&&!digitalRead(SW2))
{
   // statement
}
```

The preceding program snippet can be verbalized as follows: If SW1 is *low* (0 V because the button is pressed and the pull-up resistor is active and we thus have a digital value of 0) and SW2 is *high* (the button is not pressed and the pull-up resistor is active, thus the digital value is 1), then execute the first block. If SW1 is *high* (not pressed, 5 V are applied, thus the digital value is 1) and SW2 is *low* (pressed, digital value is 0) then execute the code after `else if`.

In short form:

```
If SW1 = 0 and SW2 = 1 then execute statement 1
```

If the first condition is not true, then test the following:

```
If SW1 = 1 and SW2 = 0 then execute statement 2
```

In the statements we increment (++) or decrement (`--`) the variable `brightness`. To avoid an overflow, the less-than query (<) and the greater-than query (>) provide an upper limit of 255 and a lower one of 0.

A delay makes for a more pleasant dimming experience

No matter how long you press the button, the value of the variable will never exceed 255 or drop below 0. In order to provide a more convenient way to set the brightness, a delay of 10 ms is added. Every pass takes 10 ms, which makes the adjustment of the brightness very comfortable and simple.

When you increase the delay value, the dimming process will be slower when you press a button. If you remove `delay` completely, the variable `brightness` is incremented or decremented so rapidly that you cannot observe any dimming. Instead, it looks as if the LED was turned on or off.

10.2 | Soft flasher

More lighting effects by using the sine function

With the sine function, you can coax the analog output to issue a sinusoidal signal. This provides for a smooth increase and decrease in the brightness of the LED which comes in handy for some applications. This slow variation in the brightness looks as if the board had a beating heart.

The set-up is the same as in the previous example (see Fig. 10.1). The main program runs through a loop that counts from 0 to 255. The corresponding values to the numbers are retrieved from the array with the sine function table and passed as PWM values to the analog output via `analogWrite`. Using a table is significantly faster than calculating the values at run time.

Example: Sine wave blinker

Time needed: 10 min
Difficulty: 2

```
// Franzis Arduino
// Sine wave blinker

byte i=0;
int LED=11;

byte Data[] = {128,131,134,137,140,144,147,150,153,
156,159,162,165,168,171,174,177,180,182,185,188,191,
194,196,199,201,204,206,209,211,214,216,218,220,222,
224,226,228,230,232,234,236,237,239,240,242,243,244,
246,247,248,249,250,251,251,252,253,253,254,254,254,
255,255,255,255,255,255,255,254,254,253,253,252,252,
251,250,249,248,247,246,245,244,242,241,240,238,236,
235,233,231,229,227,225,223,221,219,217,215,212,210,
208,205,203,200,197,195,192,189,187,184,181,178,175,
172,169,167,164,160,157,154,151,148,145,142,139,136,
133,130,126,123,120,117,114,111,108,105,102,99,96,
14,12,11,10,9,8,7,6,5,4,4,3,3,2,2,1,1,1,1,1,1,1,2,
```

```
2,2,3,3,4,5,5,6,7,8,9,10,12,13,14,16,17,19,20,22,
24,26,28,30,32,34,36,38,40,42,45,47,50,52,55,57,60,
62,65,68,71,74,76,79,82,85,88,91,94,97,100,103,106,
109,112,116,119,122,125,128};

void setup()
{
   // This time, we do not have to do anything
   // in here …
}

void loop()
{
   for(i=0;i<255;i++)
   {
      analogWrite(LED,Data[i]);
      delay(5);
   }
}
```

The program uses the dynamic byte Array `Data[]` whose values are assigned in the braces. It is called a dynamic array because its size is determined by the number of values defined in the braces. As we have put 256 values into the braces, the array has a size of 256 bytes. You can access the single values in the array by the index in the range from 0 to 255.

How to change the duty cycle of the PWM output

The code fetches one value (`Data[i]`) at a time and writes it to the hardware using `analogWrite`, thereby changing the duty cycle of the PWM output.

By the way: This is only a quasi-analog output. The statement is called `analogWrite`, but we only change the duty cycle of the output. Without a filter at the output, we just get a simple PWM signal and not a true analog signal as it is issued by a real digital/analog converter (DAC). The next experiment will show how you can generate a real analog signal out of a PWM signal.

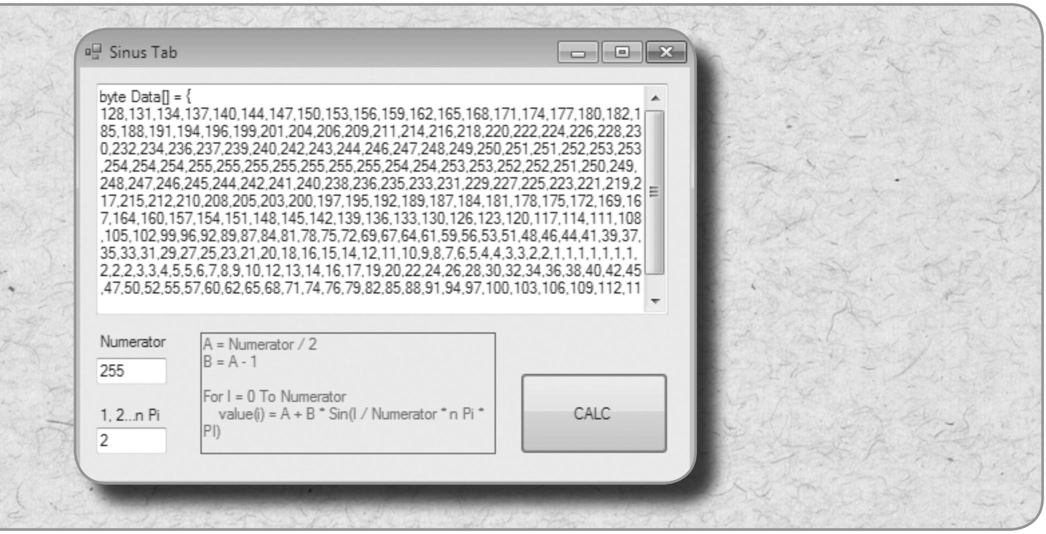


*Figure 10.2: **The program Sinus Tab that calculates the values of the sine wave table***

The little Visual Basic .NET program Sinus Tab calculates the sine wave table that you can directly insert into your programs. You can find the program on the CD included in the tutorial kit.

Variants with an oscilloscope

If you happen to own an oscilloscope you can attach an RC circuit to the analog output (instead of the LED) and view the sinusoidal progress on the monitor of the device. An RC circuit with a 10 kΩ resistor and a 1 μF capacitor will suffice.

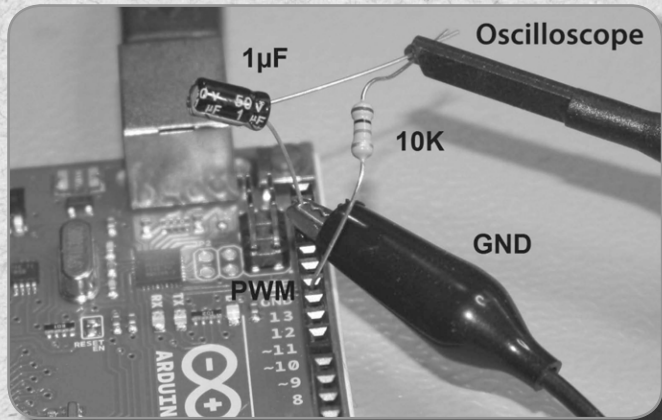


Figure 10.3: Set-up of the RC circuit. The resistor is connected to analog output 11 on the Arduino board. The negative terminal of the capacitor is attached to ground. This circuit filters the PWM signal so that only the envelope of the sine function is visible on the oscilloscope.

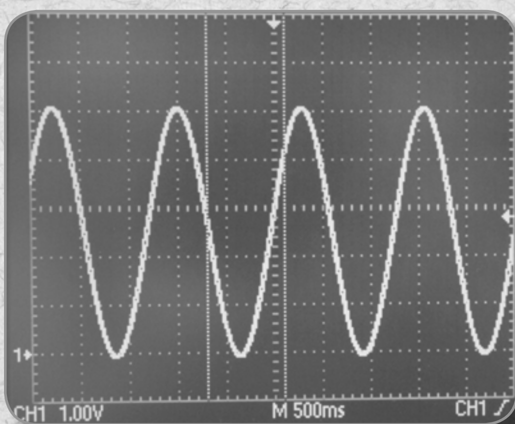


Figure 10.4: The result on the oscilloscope after attaching the RC circuit

Tip More information about envelopes of signals and RC circuits can be found at:

http://en.wikipedia.org/wiki/Envelope_detector
http://en.wikipedia.org/wiki/RC_circuit

Calculating the sine function in the program

The second example shows how you can calculate the sine function for the sine wave blinker in the program. In the previous example, we have used a table, now we will carry out the calculation directly on the microcontroller. The program is much smaller, but the calculation puts a lot of stress on the microcontroller so that the run time is significantly increased.

The `sin()` function accepts a value in radians. First, you have to convert angular degrees to radians, which is done by `x*(pi/180)`. When you multiply the result with 255 (PWM range from 0 to 255) you scale the sine function to the range from 0 to 255.

Example: Sine wave blinker with sine function

Time needed: 10 min
Difficulty: 2

■ ■ □ □ □

```
// Franzis Arduino
// Soft blinker with sine function

int ledPin = 11;
float Val;
int led;

void setup()
{
   pinMode(ledPin, OUTPUT);
}

```

```
void loop()
{
   for (int x=0; x<180; x++)
   {
      Val = (sin(x*(3.1416/180)));
      led = int(Val*255);
      analogWrite(ledPin, led);
      delay(10);
   }
}
```

10.3 | Debouncing buttons

How to debounce push-buttons via the software

Due to their mechanical composition, push-buttons exhibit the characteristic trait of »bouncing«. Whenever you press or release the button, the signal does not change immediately to *high* or *low*, but issues short impulses that give the impression of someone rapidly operating the button.

As these impulses are too short, you cannot see this effect when you use the button to switch on or off a light bulb. However, the controller retrieves the button state so fast that he gains the impression of a button that is rapidly pressed and released. In order to determine a steady state, the button has to be debounced by means of the software.

Figure 10.5: This is the impression the microcontroller gains at the digital input due to the bouncing effect of a button

The problem can be avoided by retrieving the state two times with a short delay between the readings. Only when the signal level at the second reading is identical to the level at the first try, you can act on the assumption that the button was actually pressed (or depressed) and the current digital value on the input is correct. The delay should be in the range from 20 to 100 ms.

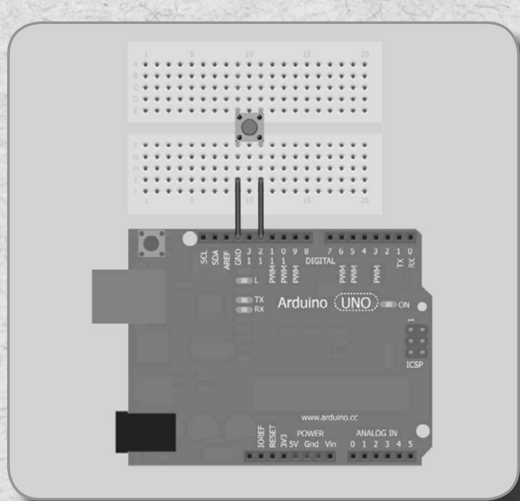


Figure 10.6: ***Diagram of the set-up for an LED dimmer with transistor***

Required parts for the experiment

- 1 x microcontroller board Arduino Uno
- 1 x breadboard
- 1 x push-buttons
- 2 x jump wire, ca. 5 cm

Example: Debouncing a push-button V1

Time needed: 10 min
Difficulty: 2

```
// Franzis Arduino
// Debouncing a push-button V1

int SW1=12;

void setup()
{
   Serial.begin(9600);
   pinMode(SW1,INPUT);
   digitalWrite(SW1,HIGH);
   Serial.println("Debouncing a push-button V1");
}

void loop()
{
   if(!digitalRead(SW1))
   {
      delay(50);
      if(!digitalRead(SW1))
      {
         Serial.println("Button SW1 has been
                                 pressed");
      }
   }
}
```

In this code, text is output to Terminal when the button is pressed. The state of the button is read (query for a `low` state), and after a short delay (50 ms) it is read again. If it is still `low`, the text is printed to Terminal.

Simple debouncing

The drawback of this method is, that the programs is called so often until the button is released. Another possibility is to execute the program code and then wait for the button to be released. The program runs through the `while(!digitalRead(SW1));` loop until the button is pressed no longer.

Example: Debouncing a push-button V2

Time needed: 10 min
Difficulty: 2

```
// Franzis Arduino
// Debouncing a push-button V2

byte i=0;
int SW1=12;

void setup()
{
   Serial.begin(9600);
   pinMode(SW1,INPUT);
   digitalWrite(SW1,HIGH);
   Serial.println("Debouncing a push-button V2");
}

void loop()
{
   if(!digitalRead(SW1))
   {
      delay(50);
      if(!digitalRead(SW1))
      {
         i++;
         Serial.print("Button SW1 was pressed ");
         Serial.print(i,DEC);
         Serial.println("x times");
         do{
         }while(!digitalRead(SW1));
      }
   }
}
```

The reverse behaviour can be obtained by placing the `do while` loop at the beginning. Now the code will be executed only after the button is released.

Example: Debouncing a push-button V3

Time needed: 10 min
Difficulty: 2

```
// Franzis Arduino
// Debouncing a push-button V3

byte i=0;
int SW1=12;

void setup()
{
   Serial.begin(9600);
   pinMode(SW1,INPUT);
   digitalWrite(SW1,HIGH);
   Serial.println("Debouncing a push-button V3");
}

void loop()
{
   if(!digitalRead(SW1))
   {
      delay(50);
      if(!digitalRead(SW1))
      {
         do{
         }while(!digitalRead(SW1));
         i++;
         Serial.print("Button SW1 was pressed ");
         Serial.print(i,DEC);
         Serial.println("x times");
      }
   }
}
```

Debouncing for advanced learners

The following code provides an even better (and nearly perfect) solution. It is an amalgamation of the previous example. Furthermore, the results are not only retrieved twice but also compared. The value of `digitalRead` has to be identical at two points in a given time period in order to run the code. As a further addition, we turn on or off LED L on the Arduino board.

*Example: **Debouncing a push-button V4***

Time needed: 10 min
Difficulty: 3

■■■□□

```
// Franzis Arduino
// Debouncing a push-button V4

byte i=0;
int SW1=12;
int LED=13;
int TOG=0;
byte value_1, value_2=0;

void setup()
{
   Serial.begin(9600);
   pinMode(SW1,INPUT);
   digitalWrite(SW1,HIGH);
   pinMode(LED,OUTPUT);
   Serial.println("Debouncing a push-button V4");
}

void loop()
{
   value_1=digitalRead(SW1);
   if(!value_1)
   {
      delay(50);
      value_2=digitalRead(SW1);
```

```
      if(!value_2)
      {
         i++;
         Serial.print("Button SW1 was pressed ");
         Serial.print(i,DEC);
         Serial.println("x times");
         if(TOG!=0)TOG=0;else TOG=1;
         digitalWrite(LED,TOG);
         do{
         }while(!digitalRead(SW1));
      }
   }
}
```

10.4 | A simple switch-on delay

As the name implies, a switch-on delay switches on a consumer (in our case, the LED L) with a delay after pressing the button. In our example, the delay is implemented by the `delay()` command and a counting loop. When you press the button, a flag stores the state und increments the variable `i`. When `i` exceeds the preset amount of milliseconds (in this case 3000 ms or 3 s), LED L is turned on and the program gets »trapped« in the `while(1)` loop.

A simple switch-on delay switches on a consumer with a delay after pressing the button

As in the examples about the debouncing of push-buttons, the button is attached to digital pin 12 and ground. Now you have to press the push-button once and then release it in order to leave `do{}while(!digitalRead(SW1));`.

In querying the button state, the flag is set to `1`. Now the incrementation of the variable `i` begins. When it exceeds 3000, the LED is turned on. Due to `delay(1)`, the increase of `i` by 1 only happens every millisecond.

Time needed: 10 min
Difficulty: 2

Example: Switch-on delay

```
// Franzis Arduino
// Switch-on delay

int SW1=12;
int value_1, value_2=0;
int LED=13;
byte Flag=0;
int i=0;

void setup()
{
   pinMode(SW1,INPUT);
   digitalWrite(SW1,HIGH);
   pinMode(LED,OUTPUT);
}

void loop()
{
   value_1=digitalRead(SW1);

   if(!value_1)
   {
      delay(50);
      value_2=digitalRead(SW1);
      if(!value_2)
      {
         Flag=1;
         do{
         }while(!digitalRead(SW1));
      }
   }

```

```
    if(Flag==1)i++;
    if(i>3000)
    {
       digitalWrite(LED,HIGH);
       while(1);
    }
    delay(1);
}
```

10.5 | A simple switch-off delay

The counterpart of the switch-on delay is the switch-off delay. With this, a consumer is turned off with a preset delay after pressing the button. The procedure is identical to that of the switch-on delay, but here the variable `i` is decremented instead of incremented.

A switch-off delay switches off a consumer after a preset time

Example: Switch-off delay

Time needed: 10 min
Difficulty: 2

```
// Franzis Arduino
// Switch-off delay

int SW1=12;
int value_1, value_2=0;
int LED=13;
byte Flag=0;
int i=3000;

void setup()
{
    pinMode(SW1,INPUT);
    digitalWrite(SW1,HIGH);
```

```
    pinMode(LED,OUTPUT);
    digitalWrite(LED,HIGH);
}

void loop()
{
    value_1=digitalRead(SW1);

    if(!value_1)
    {
        delay(50);
        value_2=digitalRead(SW1);
        if(!value_2)
        {
            Flag=1;
            do{
            }while(!digitalRead(SW1));
        }
    }

    if(Flag==1)i--;
    if(i==0)
    {
        digitalWrite(LED,LOW);
        while(1);
    }
    delay(1);
}
```

10.6 | LEDs

Calculating the series resistors for LEDs

In most of the previously described applications, one or more LEDs were used as output to test the software. You may have asked yourself how you have to calculate the series resistor in cases like these.

An LED is very much like a normal silicon diode, but it is operated in conducting direction (anode to the positive pole and cathode to the negative). There is a voltage drop along the LED, the amount of which depends on the colour (between 1.6 and 4 V).

The exact voltage is given in the data sheet for the LED and is called $V_{forward}$. The LED also needs some current so that it can light up. This current is called $I_{forward}$ in the data sheets. In this tutorial kit, we only use low-current LEDs with a maximum operating current of 2 mA.

An example for the calculation:

$I_{forward}$ = 2 mA (low-current LED)

$V_{forward}$ = 2.2 V

Operating voltage of the Arduino Vcc = 5 V

R = x Ω (the quantity we want to determine)

$$R = \frac{VCC - V_{forward}}{I_{forward}} = \frac{5V - 2.2V}{2mA} = 1\,400\,\Omega$$

Use a series resistor of the E12 series with a little higher value, namely 1.5 kΩ, to make sure the LED will not be damaged.

Double-flash LED signal

As a hands-on example, we will build an LED double flasher: The LEDs attached to the digital pins 10 and 11 blink alternately three times each. This simulates the light effect of the beacon light on an ambulance car.

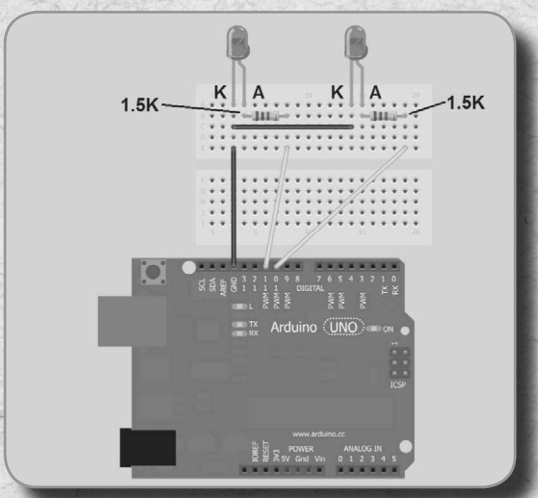


Figure 10.7: ***Diagram of the circuit***

Required parts for the experiment

- 1 x microcontroller board Arduino Uno
- 1 x breadboard
- 2 x red LED
- 2 x 1.5 kΩ resistors
- 4 x jump wire, ca. 10 cm

Example: Double flasher

Time needed: 10 min
Difficulty: 3

```
// Franzis Arduino
// Double flasher

int LED_1=10;
int LED_2=11;
int i=0;
int TOG=0;

void setup()
{
   pinMode(LED_1,OUTPUT);
   pinMode(LED_2,OUTPUT);
}

void loop()
{
   for(i=0;i<3;i++)
   {
      if(TOG==0)TOG=HIGH;else TOG=LOW;
      digitalWrite(LED_1,TOG);
      delay(40);
   }
   TOG=0
   digitalWrite(LED_1,LOW);
   delay(100);

   for(i=0;i<3;i++)
   {
      if(TOG==0)TOG=HIGH;else TOG=LOW;
      digitalWrite(LED_2,TOG);
      delay(40);
   }
   digitalWrite(LED_2,LOW);
   delay(100);

```

```
    for(i=0;i<3;i++)
    {
       if(TOG==0)TOG=HIGH;else TOG=LOW;
       digitalWrite(LED_1,TOG);
       delay(40);
    }

    digitalWrite(LED_1,LOW);
    delay(500);
}
```

The program runs through the first `for` loop and lets the LED at digital pin 10 blink three times. Then it enters the second `for` loop and causes the second LED to blink three times. After that, it waits for 500 ms and starts again.

10.7 | Switching large consumers

Operating mode of a transistor

If you need more current than our port can provide (max. ±40 mA), you will have to amplify it by a transistor as you did in the dimmer project. Let us have a more detailed look at transistors and their properties.

In a transistor, a small current (I_B) flows to the base and provides for a larger collector current (I_C). The amplification (expressed as the so called h_{FE} value) of small-signal transistors amounts to a factor of 100 to 1000, depending on the model. The transistor BC548C that we use in our experiments has an average amplification factor of about 300. A base current of 0.1 mA will therefore result in a collector current of 30 mA. The collector current for our transistor must not exceed 100 mA. Again, we will use an LED with a series resistor for demonstration purposes.

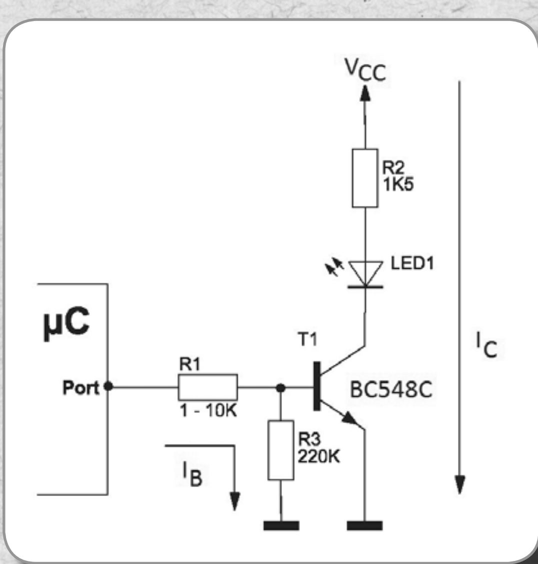


*Figure 10.8: **A transistor at the digital output of the Arduino microcontroller (µC); I_B = base current, I_C = collector current. Both base and collector current flow through the emitter. The tutorial kit contains low-current LEDs ($I_{forward}$ = 2 mA), hence the large 1.5 kΩ series resistor R2.***

For the R1 resistor, we choose a value between 1 and 10 kΩ, depending on the application. With a BC548C transistor a 10 kΩ resistor is sufficient to fully illuminate an LED.

The resistor R3 serves to protect the base against interference. When you switch on the Arduino, the digital pins have a high resistance because they are initialized as inputs. The base would be »up in the air«. To avoid this, we attach a 220 to 470 kΩ resistor directly to the base and against ground. This makes sure that the transistor connects through only when a larger current flows to the base.

The resistor serves to protect the base against interference

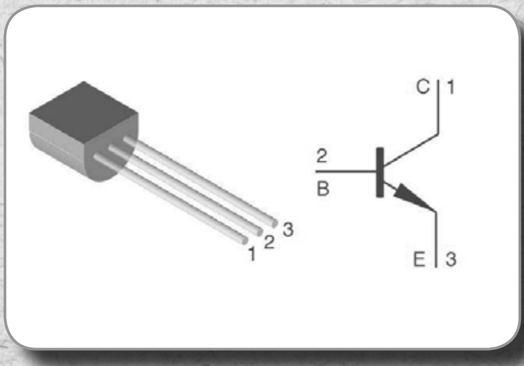


*Figure 10.9: **Pin configuration of the BC548 transistor (Source: Vishay data sheet)***

The more current the consumer needs, the more current must flow to the base so that a larger collector current is possible.

How to calculate the collector current

The collector current is calculated as follows:

$$I_C = I_B \times h_{FE} \text{ (transistor amplification factor)}$$

The following circuit diagram shows a transistor controlling a small relay. The resistance value of resistor R5 may be 1 to 22 kΩ, depending on the coil current. In general you can use a 4.7 kΩ resistor because the transistor works as a simple switch.

There is no interference-suppression resistor in this circuit, but you can add one in case you experience any problems. As in the previous example, you can use a 220 kΩ resistor between base and ground.

Button S1 uses R4 as an external pull-up resistor, where R4 should have a resistance between 10 and 22 kΩ. Diode D1 prevents the inductive voltage in the relay coil from damaging the transistor when switching off. The inductive voltage is polarised in the opposite direction of the source. Thus the diode has to be inserted in a way that short-circuits the inductive voltage. In this example, the relay turns on lamp La1 when the digital pin is *high* (5 V).

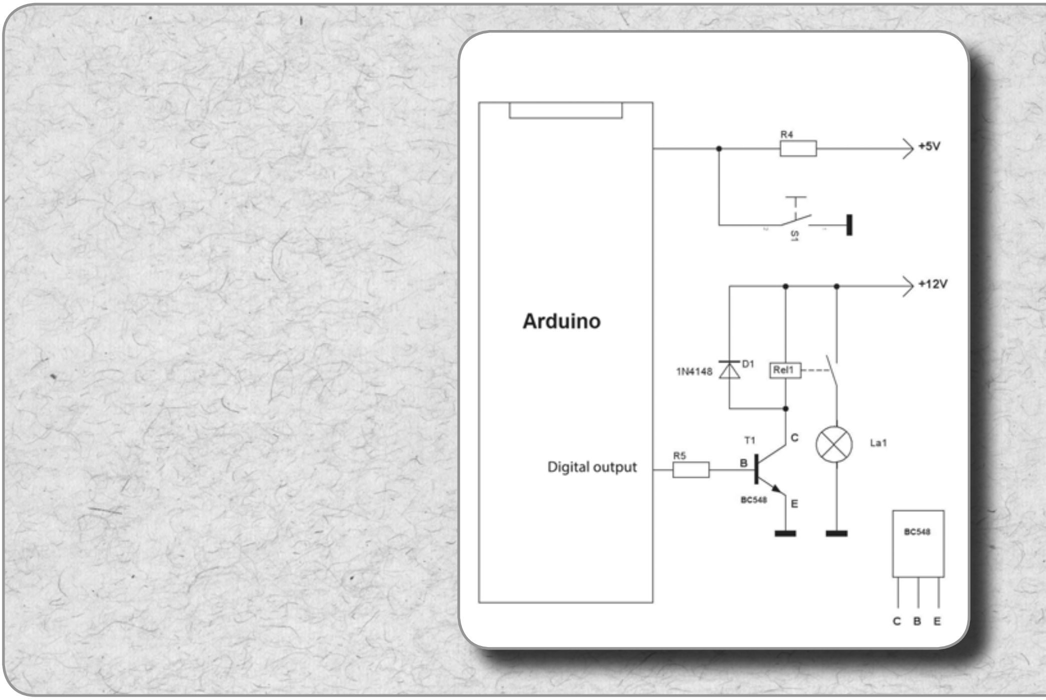


Figure 10.10: ***Relay at the digital pin of the Arduino***

There are many different models of relays. They all have potential-free contacts, i.e., the contact has no connection to the microcontroller circuit whatsoever.

Relays have a potential-free contact

TURN ON YOUR CREATIVITY

FRANZIS

ARDUINO

APPENDIX

... ABBREVIATIONS, QUANTITIES, AND UNITS

13

On the following pages, you will find some useful tables for abbreviations, electrical quantities, units of measurements and symbols.

13.1 | Electrical quantities

You have to differentiate between quantities like voltage, current, and resistance and the units of measurement for these quantities (volt, ampere, and ohm). Every quantity and every unit of measurement has its own abbreviation that is used in formulas. This provides for a neat and clear notation. For instance, you simply write »I = 1 A« instead of »current = 1 ampere«.

In this book, the following abbreviations are used:

Quantity	Abbreviation	Unit	Abbreviation
Voltage	V or U	Volt	V
Current	I	Ampere	A
Resistance	R	Ohm	Ω
Power	P	Watt	W
Frequency	f	Hertz	Hz
Time	t	Second	s
Wave length	λ (lambda)	Meter	m
Inductance	L	Henry	H
Capacitance	C	Farad	F
Area	A	Square meter	m^2

13.2 | ASCII Table

Symbol	Decimal	Hexa-decimal	Binary	Description
NUL	000	000	00000000	Null character
SOH	001	001	00000001	Start of header
STX	002	002	00000010	Start of text
ETX	003	003	00000011	End of text
EOT	004	004	00000100	End of transmission
ENQ	005	005	00000101	Enquiry
ACK	006	006	00000110	Acknowledgment
BEL	007	007	00000111	Bell
BS	008	008	00001000	Backspace
HAT	009	009	00001001	Horizontal tab
LF	010	00A	00001010	Line feed
VT	011	00B	00001011	Vertical tab
FF	012	00C	00001100	Form feed
CR	013	00D	00001101	Carriage return
SO	014	00E	00001110	Shift out
SI	015	00F	00001111	Shift in
DLE	016	010	00010000	Data link escape
DC1	017	011	00010001	Device control 1
DC2	018	012	00010010	Device control 2
DC3	019	013	00010011	Device control 3
DC4	020	014	00010100	Device control 4
NAK	021	015	00010101	Negative acknowledgment
SYN	022	016	00010110	Synchronous idle
ETB	023	017	00010111	End of transmission block
CAN	024	018	00011000	Cancel

Symbol	Decimal	Hexa-decimal	Binary	Description
EM	025	019	00011001	End of medium
SUB	026	01A	00011010	Substitute
ESC	027	01B	00011011	Escape
FS	028	01C	00011100	File separator
GS	029	01D	00011101	Group separator
RS	030	01E	00011110	Request to send, record separator
US	031	01F	00011111	Unit separator
SP	032	020	00100000	Space
!	033	021	00100001	Exclamation mark
“	034	022	00100010	Double quote
#	035	023	00100011	Number sign
$	036	024	00100100	Dollar sign
%	037	025	00100101	Percent
&	038	026	00100110	Ampersand
‘	039	027	00100111	Single quote
(	040	028	00101000	Left opening parenthesis
)	041	029	00101001	Right closing parenthesis
*	042	02A	00101010	Asterisk
+	043	02B	00101011	Plus
,	044	02C	00101100	Comma
-	045	02D	00101101	Minus or dash
.	046	02E	00101110	Dot
CHAR	DEC	HEX	BIN	Description
/	047	02F	00101111	Forward slash
0	048	030	00110000	
1	049	031	00110001	
2	050	032	00110010	

Symbol	Decimal	Hexa-decimal	Binary	Description
3	051	033	00110011	
4	052	034	00110100	
5	053	035	00110101	
6	054	036	00110110	
7	055	037	00110111	
8	056	038	00111000	
9	057	039	00111001	
:	058	03A	00111010	Colon
;	059	03B	00111011	Semicolon
<	060	03C	00111100	Less than
=	061	03D	00111101	Equal
>	062	03E	00111110	Greater than
?	063	03F	00111111	Question mark
@	064	040	01000000	At symbol
A	065	041	01000001	
B	066	042	01000010	
C	067	043	01000011	
D	068	044	01000100	
E	069	045	01000101	
F	070	046	01000110	
G	071	047	01000111	
H	072	048	01001000	
I	073	049	01001001	
J	074	04A	01001010	
K	075	04B	01001011	
L	076	04C	01001100	
M	077	04D	01001101	
N	078	04E	01001110	

Symbol	Decimal	Hexa-decimal	Binary	Description
O	079	04F	01001111	
P	080	050	01010000	
Q	081	051	01010001	
R	082	052	01010010	
S	083	053	01010011	
T	084	054	01010100	
U	085	055	01010101	
V	086	056	01010110	
W	087	057	01010111	
X	088	058	01011000	
Y	089	059	01011001	
Z	090	05A	01011010	
[	091	05B	01011011	Left opening bracket
\	092	05C	01011100	Backslash
]	093	05D	01011101	Right closing bracket
^	094	05E	01011110	Caret
CHAR	DEC	HEX	BIN	Description
_	095	05F	01011111	Underscore
`	096	060	01100000	
a	097	061	01100001	
b	098	062	01100010	
c	099	063	01100011	
d	100	064	01100100	
e	101	065	01100101	
f	102	066	01100110	
g	103	067	01100111	

Symbol	Decimal	Hexa-decimal	Binary	Description
h	104	068	01101000	
i	105	069	01101001	
j	106	06A	01101010	
k	107	06B	01101011	
l	108	06C	01101100	
m	109	06D	01101101	
n	110	06E	01101110	
o	111	06F	01101111	
p	112	070	01110000	
q	113	071	01110001	
r	114	072	01110010	
s	115	073	01110011	
t	116	074	01110100	
u	117	075	01110101	
v	118	076	01110110	
w	119	077	01110111	
x	120	078	01111000	
y	121	079	01111001	
z	122	07A	01111010	
{	123	07B	01111011	Left opening brace
|	124	07C	01111100	Vertical bar
}	125	07D	01111101	Right closing brace
~	126	07E	01111110	Tilde
DEL	127	07F	01111111	Delete

TURN ON YOUR CREATIVITY

THE FRANZIS ARDUINO™ TUTORIAL KIT

This tutorial kit includes everything you need for your first steps in programming: an original Arduino™ Uno, breadboard, components, a 282-page manual and software. With this kit you can build successful projects and bring to life your Arduino.™

ALREADY DONE YOUR EXPERIMENT FOR THE DAY?

Whether you want to build a home automation system or an LED lamp with changing colours – with the Arduino™ even beginners can successfully write their first programs and implement their very own ideas! In this tutorial kit you discover the basics of electronics and Arduino™ programming and get step-by-step instructions to put your ideas into practice.

LIST OF THE COMPONENTS:

1 Arduino Uno
1 breadboard
2 push-buttons
1 NPN transistor BC548C
1 silicon diode 1N4148
1 piezo buzzer
1 red LED
1 green LED
2 yellow LEDs
3 resistors 1.5 kΩ
1 resistor 4.7 kΩ
1 resistor 47 kΩ
1 resistor 10 kΩ
1 trim potentiometer 10 kΩ PT10
1 capacitor 1 μF
1 insulated hookup wire ca. 1 m

WITH THIS TUTORIAL KIT YOU WILL PERFORM THE FOLLOWING PROJECTS:

Programming with loops
Generating random numbers
A simple game
Stop-watch
Measuring voltages
LED dimmer
Switch-on and switch-off delay
Music with the Arduino™
Candlelight, courtesy of the microcontroller
Monitoring exits
School bell
Keypad lock
Voltage plotter
Storage oscilloscope
Temperature switch
Romantic lights
Timer clock
Composing melodies
State machines
Capacitance meter
... and many more!

Not suitable for children under 14!

In addition, you need: USB connection cable

ISBN 978-3-645-65279-7

9 783645 652797

FRANZIS