ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Version 1.0.0
Manual Number: 88

Physikpraktikum für Vorgerückte (VP)
vp.phys.ethz.ch

Instruction Manual

Advanced Physics Lab
Electronics A+D

Modern Aspect of Data Taking and Processing with a Microcontroller

Pirmin Berger & Michael Reichmann

Abstract

The experiment "Digital Electronic" provides an introduction into modern data taking by operating simple digital circuits utilising an Arduino board. This manual will inform you about the Arduino board, the installation of the required software and the electrical components you will have to use. Basic knowledge on electronics, how to use oscilloscopes, bread boards and power supplies is recommended.

During the experiment you will learn how to build a circuit that measures the temperature, how to operate it using the Arduino board and to modify and improve it using more components.

In case you should already have previous knowledge we will provide many more material and own ideas on implementation are very welcome and can be built consulting the assistants.

ETH Zürich, Switzerland, October 8, 2018

Contents

1 Introduction

Arduino is a computer company, project and user community based on easy-to-use hardware and software, that designs and manufactures single-board microcontrollers and microcontroller kits for building digital devices and interactive objects that can sense and control objects in the physical and digital world. All products are distributed as open-source hardware and software, and it's licences permit the manufacture of Arduino boards and software distribution by anyone. The boards are commercially available in preassembled form, or as do-it-yourself (DIY) kits.

The Arduino project started in 2003 as a program for students without a background in electronics and programming at the Interaction Design Institute in Ivrea (Italy). The aim was to provide a low-cost and easy way for novices and professionals to create devices that interact with their environment using sensors and actuators. The actual name Arduino comes from a bar in Ivrea, where some of the founders of the project used to meet. The bar was named after Arduin of Ivrea, who was the margrave of the March of Ivrea and King of Italy from 1002 to 1014 [**?**].

In order to work with the Arduino Boards the Arduino programming language, based on Wiring, and the Arduino Software (IDE), based on the Processing are used [**?**]. Both Wiring and Processing are programming languages using a simplified dialect of features from the programming languages C and C++.

1.1 Arduino Board

The original boards were produced by the Italian company Smart Projects but as of 2018, 22 versions of the Arduino hardware have been commercially produced. The information and specifications of these boards can be found on this website. During this Lab you will work the Arduino Uno shown in Figure 1.

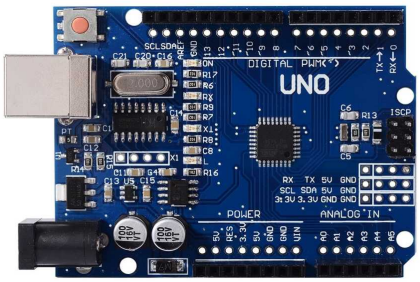


Figure 1: Arduino Uno.

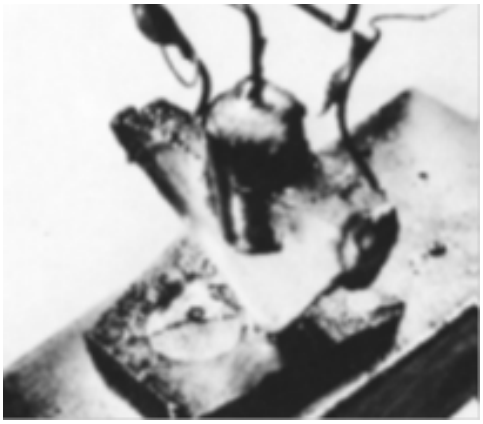
The Arduino Boards use a variety of microprocessors and controllers and are equipped with sets of digital and analogue input/output (I/O) pins that may be interfaced to various expansion boards or Breadboards (shields) and other circuits. The boards feature serial communications interfaces, Universal Serial Bus (USB) on some models, which are also used for loading programs from personal computers.

Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell the board what to do by sending a set of instructions to the microcontroller.

1.2 Transistor

The invention of the transistor was announced in 1948 by the American physicists, J. Bardeen and W. H. Brattain as a new type of amplifying device made from semiconducting crystals. At that time almost no one could have foreseen the revolutionary developments that were to follow, developments so important and far-reaching as to change the whole outlook of the

science and technology of electronics. The physical principles of a transistor had been worked out in conjunction with their colleague, W. Shockley. In recognition of their work the three physicists were awarded jointly the Nobel Prize for Physics in 1956.



(a) Point-contact transistor.

(b) Bardeen, Brattain and Shockley.

The term "transistor" is a combination from the words *trans*former and res*istor*, since the device is made from resistor material and transformer action is involved in the operation. In the beginning only point-contact transistors existed, but due to their vulnerability to mechanical shock they were soon replaced by junction transistors which are firmly established now [**?**].

The transistor is the key active component in practically all modern electronics. It is considered as one of the greatest inventions of the 20th century. Its importance in today's society rests on its ability to be mass-produced using a highly automated process that achieves astonishingly low per-transistor costs (10 femto$/transistor) [**?**].

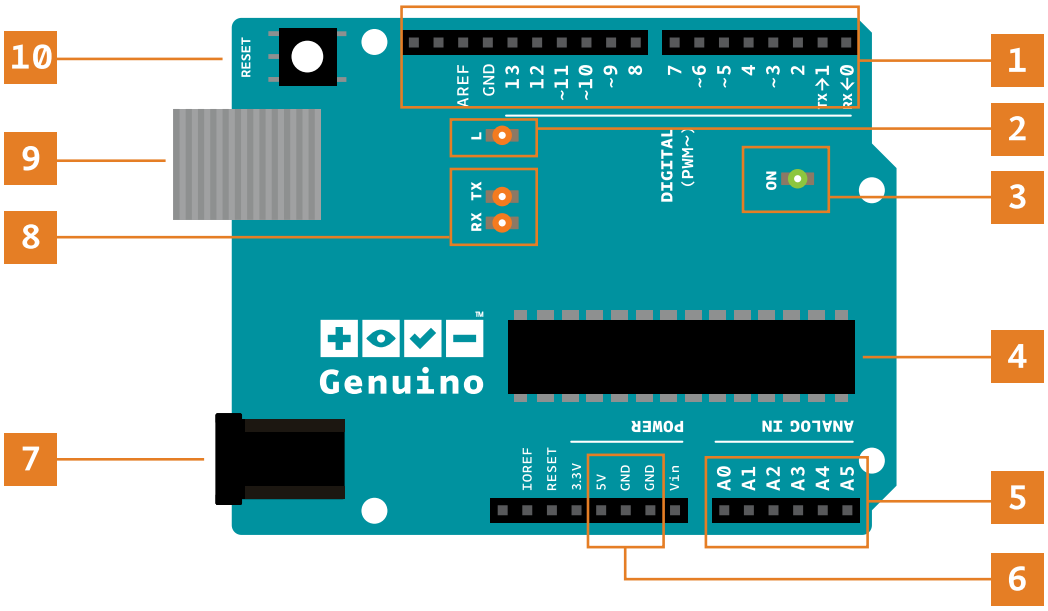
Although billions of individually packaged (discrete) transistors are produced every year the vast majority of transistors are now produced in integrated circuits (ICs). A logic gate consists of up to about twenty transistors whereas an advanced microprocessor, as of 2009, can use as many as 3 billion transistors. In 2014, about 10 billion transistors were built for each single person on Earth [**?**].

The working principle of the transistor will be covered in subsection 2.9.

2 Basics

This section will give you the basic information about the components we are using in this lab.

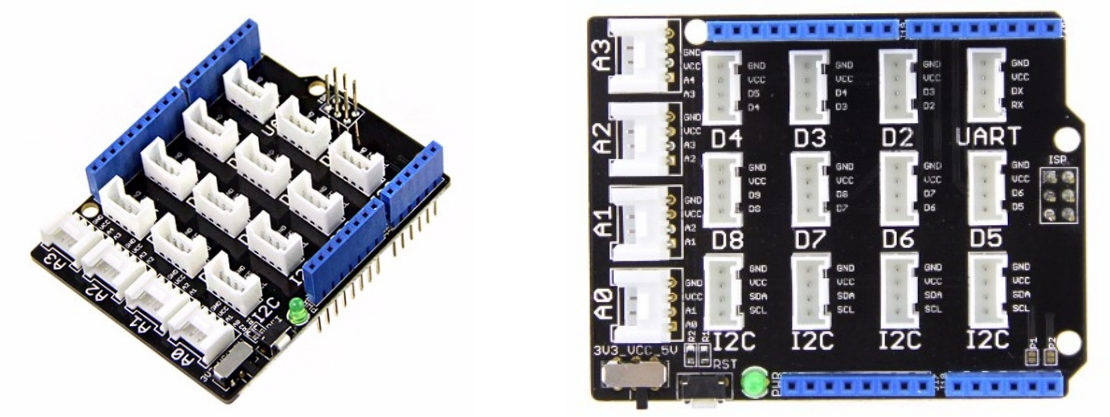
2.1 Arduino Uno



1. **Digital pins:** used with digitalRead(), digitalWrite(), and analogWrite() methods, analogWrite() only works on pins with the pulse width modulation (PWM) symbol
2. **Pin 13 LED:** only built-in actuator
3. **Power LED**
4. **ATmega microcontroller**
5. **Analogue in:** used with analogWrite() method
6. **GND and 5V pins:** provide 5 V power and ground (GND) to the circuits
7. **Power connector:** additional power supply, accepted voltages: $7 \sim 12$ V
8. **TX and RX LEDs:** indicate communication between Arduino and computer
9. **USB port:** used for powering and communication with computer
10. **Reset button:** resets the ATmega microcontroller

2.2 Grove Base Shield

The so called shields are printed circuit expansion boards, which plug into the normally supplied Arduino pin headers. The Grove Base Shield is one example that simplifies projects that require a lot of sensors or LEDs. With the Grove connectors on the base board, one can add all the Grove modules to the Arduino Uno very conveniently.



(a) Side view. (b) Top view.

Figure 3: Grove shield.

There are 16 Grove connectors on the Base Shield which are shown in Table 3. Apart from the connectors the board also consists of a reset (RST) button, a green LED to indicating power status, a toggle switch and four rows of pinouts, which is equivalent to the pinout of the Arduino.

Specification	Name	Quantity
Analog	A0/A1/A2/A3	4
Digital	D2/D3/D4/D5/D6/D7/D8	7
UART	UART	1
I2C	I2C	4

Table 1: Base Shield connectors.

Every Grove connector has four wires, one of which is the voltage common collector (VCC). Since some micro-controller main boards need need different supply voltages the power toggle switch allows you to select the suitable voltage. In the case of the Arduino Uno a voltage of 5 V is required [**?**].

2.3 The Software

2.3.1 Arduino IDE

All you require to write programs and upload them to your board is the Arduino Software (IDE). There are two options how to use it:

1. Online IDE

- no installation required
  - needs plugin if you want to upload sketches from Linux
- requires you to create account with e-mail verification
- save sketches in cloud (available from all devices)
- always most up-to-date version
- instructions on the website

2. Desktop IDE

- if you want to work offline
- installation usually very straightforward and in has general no dependencies
- if you need help, follow installation instructions depending on your operating system (OS)
  - Linux
  - Mac OS X
  - Windows

2.3.2 Board Drivers

First the Arduino board has to be connected to the computer via the USB cable which will power the board indicated by the green power (PWR) LED. The board drivers should then install automatically in Linux, Mac OS X and Windows. If the board was not properly recognised, follow these instructions.

2.4 Programming

In order to get a feeling for the programming language it is recommended to have a look at the examples first which can be found under: `File > Examples`
A list of the most common methods is shown in Table 2. For more information look at the detailed description.

Category	Method Syntax	Description
Sketch	setup()	called once at the start of the sketch, used to initialise variables, pin modes, etc.
	loop()	loops consecutively after setup was called
Digital I/O	digitalRead(pin)	reads the value from the digital pin (HIGH or LOW)
	digitalWrite(pin, value)	writes HIGH or LOW value to the digital pin
	pinMode(pin, mode)	configures the pin as INPUT or OUTPUT
Analogue I/O	analogRead(pin)	reads the value (0-1023) from the pin
	analogWrite(pin, value)	writes an analogue value (PWM wave) to the pin
	analogReference(type)	configures the reference voltage used for analogue input
Advanced I/O	tone(pin, f, duration)	generates a square wave of frequency f [Hz] for a duration [ms]
	pulseIn(pin, value)	returns the time [ms] of a pulse, if value is HIGH: waits until HIGH and stops when LOW
Time	delay(time)	pauses the program for a time [ms]
	micros()	returns time since starting the program [µs]
	millis()	returns time since starting the program [ms]
Math	constrain(x, a, b)	constrains a number x to be in range [a, b]
	map(x, a, b, c, d)	re-maps x from range [a, b] to range [c, d]
	random(a, b)	returns pseudo-random number in range [a, b]
	abs(value)	return the absolute value
	⋮	further general math commands
Serial	Serial.begin(speed)	initialise serial communication at speed [bit/s]
	Serial.print(value)	prints the value to the serial port
	Serial.println(value)	prints the value to the serial port with [\r\n]
	Serial.read()	reads incoming serial data

Table 2: Most common methods for controlling the Arduino board and performing computations.

2.5 Project Management with git

Git is a version control system for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for source code management in software development, but it can be used to keep track of changes in any set of files [**?**].

2.6 Voltage Divider

A voltage divider is a passive linear circuit that produces an output voltage V_{out} that is a fraction of its input voltage V_{in}. Voltage division is the result of distributing the input voltage among the components of the divider. A simple example of a voltage divider is two resistors connected in series, with V_{in} across the resistor pair and V_{out} emerging from the connection between them as shown in Figure 4.

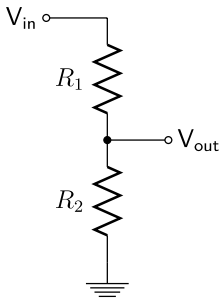


Figure 4: Voltage divider.

Resistor voltage dividers are commonly used to create reference voltages, or to reduce the magnitude of a voltage so it can be measured. Using Ohm's law one can easily derive the formula for V_{out}:

$$V_{\text{out}} = \frac{R_2}{R_1 + R_2} \cdot V_{\text{in}} \tag{1}$$

2.7 Thermistor

If, for a given temperature, the current is directly proportional to the applied voltage the electrical component is said to obey Ohm's law. Such components are called linear resistors. If a component does not meet this requirement it is termed a non-linear resistor, which falls into two classes – the temperature-sensitive type and the voltage-sensitive type. The temperature-sensitive types are often known as thermistors and change the resistance very reproducible. The word is a portmanteau of *THERM*ally-sensitive and res*ISTOR*.

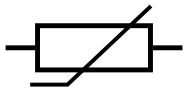


Figure 5: Electronic symbol of the thermistor

They consist of the sintered oxides of manganese and nickel with small amounts of copper, cobalt or iron added to vary the properties and the physical shape is usually a bead, rod or

a disc. The electronic symbol is shown in Figure 5. The resistance is given by

$$R = R_0 \cdot \mathrm{e}^{-B\left(\frac{1}{T_0}-\frac{1}{T}\right)} \tag{2}$$

where B is a constant depending upon the composition and physical size, T is the temperature in °K, and R_0 the resistance at ambient room temperature T_0 (25 °C = 298.15 K). Thermistors can be classified into two types, depending on the classification of B. If B is positive, the resistance increases with increasing temperature, and the device is called a positive temperature coefficient (PTC) thermistor, or posistor. If B is negative, the resistance decreases with increasing temperature, and the device is called a negative temperature coefficient (NTC) thermistor.

2.8 Temperature Sensor

Thermistors have very widespread applications as thermometers. We know that a NTC thermistor varies its resistance as a function of the temperature, but resistance is not the easiest parameter to measure. In our case we want to feed a signal into an analogue-to-digital converter (ADC) to treat it numerically and compute the actual temperature. Now, the ADC of the Arduino requires a voltage at it's input.

An easy solution is to install the NTC in a voltage divider as shown in subsection 2.6. It requires only one additional fixed resistor R_0. V_{in} is the reference voltage used of the ADC and V_{out} the measured voltage. So what you will measure in the end is just a digital 10 bit value corresponding to the divided voltage where the maximum of $2^{10}-1 = 1023$ corresponds to V_{in}. In order to convert it into the resistance of the thermistor we have to use Equation 1:

$$R = \left(\frac{V_{\mathrm{in}}}{V_{\mathrm{out}}} - 1\right) \cdot R_0 = \left(\frac{1023}{V_{\mathrm{meas}}} - 1\right) \cdot R_0 \tag{3}$$

Now we need to convert the resistance into the temperature using Equation 2:

$$\frac{1}{T} = \frac{\ln\left(\frac{R}{R_0}\right)}{B} + \frac{1}{T_0} \tag{4}$$

Note that this temperature will be in Kelvin!

2.9 Bipolar Junction Transistor

A bipolar junction transistor (BJT) is a type of transistor that uses both electron and hole charge carriers. For their operation, BJTs use two junctions between two semiconductor types, n-type and p-type and thus can be manufactured in two types, NPN and PNP. The basic function of a BJT is to amplify current which allows it to be used as amplifiers or switches, giving them wide applicability in electronic equipment.

2.9.1 Working Principle

Since a transistor consists of two pn junctions within a single crystal, transistor action can be explained with Figure 8. For diagrammatic purposes the base region is shown fairly thick,

but in fact the pn junctions are very closely spaced and the active portion of the base is very thin.

The charge flow in a BJT is due to diffusion of charge carriers across a junction between two regions of different charge concentrations. The regions of a BJT are called emitter, collector, and base. Typically, the emitter region is heavily doped compared to the other two layers, whereas the majority charge carrier concentrations in base and collector layers are about the same.

In the absence of any external applied voltages the collector and emitter depletion layers are about the same thickness, the widths depending upon the relative doping of the collector, emitter and base regions. During normal transistor operation the emitter-base junction is forward biased so that current flows easily in the input or signal circuit. The bias voltage V_{BE} is about 200 mV for germanium transistors and about 400 mV for silicon devices. The collector-base junction is reverse-biased by the main supply voltage V_{CB} typically with 4.5 V, 6 V and 9 V. The collector junction is therefore heavily reversed-biased and the depletion layer there is quite thick.

The injection of a hole into the base region by a signal source will now be considered. Once in the base, the hole attracts an electron from the emitter region. The recombination of the hole and electron is not likely to occur however, since the base region is lightly doped compared with the emitter region and so the lifetime of the electron in the base region is quite long. In addition the base is extremely thin so the electron, instead of combining with the signal hole or with a hole of the p-type base material, diffuses into the collector-base junction. The electron then comes under the influence of the strong field there and is swept into the collector and hence into the load circuit. In a good transistor many electrons pass into the collector region before eventually the signal hole is eliminated by combination with an electron. A small signal current can thus give rise to a large load current i_C, and so current amplification has taken place. In practical transistors for every hole injected into the base 50 to 250 electrons may be influenced to flow into the collector region. The current gain or amplification is therefore 50 to 250. It is usually given by the symbol β.

An PNP transistor behaves in a similar fashion except that electrons are injected into the base and holes flow from the emitter into the collector. To maintain the correct bias conditions the polarity of the external voltages must be reversed.

Figure 9 shows the three basic transistor arrangements. The common emitter mode is the most commonly used arrangement for voltage amplification because very little current is required from the signal source.

The common base mode of operation is also capable of voltage amplification. This is achieved by the use of high values of load resistor. The transistor is able to maintain the current through the load because the device is a good constant current generator [**?**].

2.9.2 Common Collector

The common collector circuit shown in Figure 9c has a voltage gain of a little less than unity and so is useless as a voltage amplifier. However this circuit has very important impedance matching properties and is typically used as a voltage buffer. In this circuit the base serves as the input, the emitter is the output, and the collector is common to both.

The voltage gain is just a little less than one since the emitter voltage is constrained at the voltage drop over the diode of about 0.6 V (for silicon) below the base. The transistor

continuously monitors V_D and adjusts its emitter voltage almost equal (less V_{BEO}) to the input voltage by passing the according collector current through the emitter resistor R_E. As a result, the output voltage follows the input voltage variations from V_{BEO} up to V_+; hence also the name, emitter follower. This circuit is useful because it has a large input impedance, so it will not load down the previous circuit and a small output impedance, so it can drive low-resistance loads

2.10 Operational Amplifier

Should be covered in AP next semester

2.11 PID Controller

A proportional–integral–derivative (PID) controller is a control loop feedback mechanism widely used in industrial control systems and a variety of other applications requiring continuously modulated control. A PID controller continuously calculates an error value et as the difference between a desired setpoint (SP) and a measured process variable (PV) and applies a correction based on proportional, integral, and derivative terms (denoted P, I, and D respectively) which give the controller its name.

In practical terms it automatically applies accurate and responsive correction to a control function. An everyday example is the cruise control on a road vehicle; where external influences such as gradients would cause speed changes, and the driver has the ability to alter the desired set speed. The PID algorithm restores the actual speed to the desired speed in the optimum way, without delay or overshoot, by controlling the power output of the vehicle's engine [**?**].

3 Setup and Experimental Procedure

The Digital Electronics Lab is meant to be very open in implementation. The aim will be to build an automated cooling system. We will guide you through this process step by step, but you are also encouraged to bring in your own ideas of other possible implementations!

3.1 Experimental Material

You will find the following objects on your test stand:

- 1 Arduino Uno Board
- 1 Oscilloscope
- 2 Probes
- 1 Multimeter
- 1 USB Type B cable
- 1 Breadboard
- 1 Grove Starter Kit for Arduino

  - 1 Base Shield
  - 1 LCD RGB Backlight
  - 1 Smart Relay
  - 1 Buzzer
  - 1 Sound Sensor
  - 1 Touch Sensor
  - 1 Rotary Angle Sensor
  - 1 Temperature Sensor
  - 1 LED Module
  - 3 Dip LEDs (red, green, blue)
  - 1 Light Sensor
  - 1 Button
  - 1 Mini Servo
  - 10 Grove Cables
- 1 3-pin fan
- 1 operational amplifier (LM358)
- 1 NPN transistor (BC547)
- 1 NTC thermistor (B57164-K104-J)

In addition you should bring a lab book to note down everything what you do and measure (immediately after the execution). Please write clearly and meticulously since it will greatly help you to find mistakes and to write your report!

3.2 Setting up the Arduino

3.2.1 Software installation

We highly recommend you to install the Arduino IDE on your machine so that you can access it any time. If you should have trouble with the installation or you prefer to not use your computer for the experiment, you can use one of the Windows XP machines provided to you, which have the Arduino IDE and all necessary drivers pre-installed. These machines do not have internet connection, so you have to bring a portable USB drive to copy the data.

- install IDE following subsection 2.3

3.2.2 Connect the Arduino

- connect the Arduino to your computer and check if it is recognised by the IDE
  - select the correct port: `Tools > Port`
  - get board info: `Tools > Get Board Info`

3.2.3 Your First Sketch

- (optional) set your sketchbook location
  - you can link it to your IDE: **`File > Preferences > Sketchbook Location`**
- create a new sketch with **`File > New`**, it will look like this:

```
1 void setup() {
2   // put your setup code here, to run once:
3
4 }
5
6 void loop() {
7   // put your main code here, to run repeatedly:
8
9 }
```

Now we want to turn on the LED on the Arduino board, which is connected to the special pin LED_BUILTIN. Since use this pin as an output, we have to set the pin mode during the setup() function to OUTPUT. Afterwards we can use the method digitalWrite() to set the pin to HIGH, (corresponding to 5V), which will turn on the LED on the board. We don't need to use the loop() method for this simple sketch, but we will use it later.

```
1 void setup() {
2   pinMode(LED_BUILTIN, OUTPUT);
3   digitalWrite(LED_BUILTIN, HIGH);
4 }
5
6 void loop() {
7   // put your main code here, to run repeatedly:
8
9 }
```

That's it. All the methods and constants we use in this script are already defined and don't have to be imported explicitly. There is a long list of such pre-defined constants, make sure to not redefine them when adding new constants or variables! We are now ready to test the communication of the Arduino board with the computer and test if flashing the code works. Make sure to select the correct Port and choose "Arduino/Genuino Uno" as board type if not already selected.

- compile the code
- upload the code to the Arduino
- add the sketch folder to a git repository

Important: Do not use the pins A4 and A5! The Arduino Uno uses these pins for the I2C communication, which will later be used for the display! Even if it looks like they are separate pins on the shield board, they are shared internally!

3.3 Blinking LED on Bread Board

That was easy (but boring...). As next step we want to have a blinking external LED on the breadboard. Since it should continue blinking forever, we will use the loop() function now.

- supply the Breadboard with 5 V from the Arduino
- connect a LED to the breadboard with appropriate resistor (220 Ohm) and to a digital pin
- write a sketch (`Blink.ino`) that makes the LED blink with a frequency of 1 Hz using the delay(int nMilliSeconds) method, which pauses the execution for nMilliSeconds milliseconds.

Upload your sketch now to the Arduino and test it.

- connect a second LED on the breadboard to a different pin
- write a sketch (`Blink2.ino`) that makes the second LED blink in a different pattern than the first one
  - why is delay() not the optimal method for this task?
  - use a timer with micros() which returns a micro-seconds counter as unsigned long.
  - remember to use unsigned long to save the results of micros()
  - after some time (roughly 70 minutes), the 32-bit counter will overflow and start from 0 again! Make sure to handle this case well!

Add the code for the 2 blinking LEDs, implemented with timers, to your report.

3.4 General tips for writing Code for the Arduino

3.4.1 Data-types

Data-types on Arduino are defined differently than for 32-bit or 64-bit x86 processors, e.g. int is only 16-bit, which can store values from -32768 to 32767. In most cases it's better to use long instead of int to avoid overflows. If no negative numbers are required, use unsigned versions. Note: On Arduino Uno, double and float are exactly the same.

3.4.2 Avoid hard-coded values

Instead of a hard-coded number somewhere in the middle of the program, like

```
1 ...
2 if (now > lastPid + 100000) {
3 ...
```

define a constant (with const or as macro with #define) in the beginning of the program. Give it a systematic name, e.g. all uppercase for constants, and e.g. also include the unit (micro-seconds) here and add a comment:

type	length	range
char	8 bit	-128 to 127
unsigned char	8-bit	0 to 255
int	16-bit	-32768 to 32767
unsigned int	16-bit	0 to 65535
long	32-bit	-2147483648 to 2147483647
unsigned long	32-bit	0 to 4294967295
float	32-bit	floating point
double	32-bit	floating point

Table 3: Basic numeric data-types for Arduino Uno.

```
1 #define INTERVAL_MICROS_PID             100000      // update
    interval of PID in micro-seconds
2 ...
3 if (now > lastPid + INTERVAL_MICROS_PID) {
4 ...
```

and later use this constant INTERVAL_MICROS_PID in the if clause. This makes the code easier to read. Macros with #define don't use space for variables (in contrast to declaration with int ...), which reduces memory footprint of your program. Keep in mind that the total RAM is only 2 kB, which corresponds to only around 500 32-bit integers you can store and part of this is already used by libraries. Using constants instead of hard-coded numbers is especially important for pin numbers.

3.4.3 Code documentation

Add a short description on the functionality of your program at the top of the code. Also include the date and author name. Also comment any non-trivial steps in your code. (remember: it's generally better to write code in a self-explaining way such that no comments are necessary. Sometimes though comments are unavoidable.)
Your final code has to be added to the report. A reasonable documentation is therefore required for the report to be accepted by the assistant.

3.5 Grove Temperature Sensor

In this experiment you will work with the Grove - Temperature Sensor V1.2. It uses a NTC thermistor to detect the ambient temperature. The specifications are shown in Table 4.

Specification	Value
Operating voltage	$3.3 \sim 5.0\,\mathrm{V}$
Zero power resistance	$(100 \pm 1)\,\mathrm{k\Omega}$
Operating temperature range	$-40 \sim +125\,^{\circ}\mathrm{C}$
Nominal B-constant	$4250 \sim 4299\,\mathrm{K}$

Table 4: Grove-Temperature Sensor V1.2 specifications.

- connect the Shield to your Arduino
- connect the Grove Temperature Sensor to the shield
  - figure out which connector to choose on the shield
- write a sketch (`GroveTemp.ino`) that measures the ambient temperature
  - read the voltage value from the sensor
  - convert the voltage into the corresponding resistance
  - convert the resistance into a temperature in °C
  - write the temperature to the serial interface every second
  - look at the data with the Arduino Serial Monitor:
    Tools > Serial Monitor (Ctrl+Shift+M)
  - look at the data with the Arduino Serial Plotter:
    Tools > Serial Plotter (Ctrl+Shift+P)

The serial connection has to be enabled during the setup() function with Serial.begin(9600); where 9600 is the baud rate. It is 9600 by default in the Serial Plotter and Serial monitor utilities, so it is convenient to use this.

3.6 Grove Display and Potentiometer

As a next step, we will add two more components from the Grove kit: a display to output the current temperature read by the sensor and a potentiometer to set the threshold for our two-point temperature control later.

- add the display to your project
  - install the libraries from http://wiki.seeedstudio.com/Grove-LCD_RGB_Backlight/
  - look at the example code on the above website, especially on how to include the libraries in your project and how to initialize it during the setup() routine
  - make sure the "3V3_VCC_5V" switch on the Grove shield is set to 5V, otherwise the display will not work correctly
  - connect the Grove LCD RGB Backlight display to your Grove shield
  - modify your project such that the measured temperature is printed on the display every second
- define a fixed threshold in your code, e.g. 30 degrees above which a LED is turned on. (As alternative you can change the background color of your display). Test if your project is working.
- now we want to make the threshold controllable without recompiling our project. For this purpose, connect the potentiometer (Grove Rotary Angle Sensor) to the Grove shield (you can also use the sliding potentiometer instead if you prefer)
  - read the description at http://wiki.seeedstudio.com/GroveRotary_Angle_Sensor/

  - read-out the sensor with `analogRead()`
  - to map the ADCs which range from 0 to 1023 to a reasonable temperature range (a, b), e.g. 20 to 40 degrees, the function `map(degrees, 0, 1023, a, b)` can be used for convenience

- indicate the status below/above threshold also in the output to the serial interface, e.g. add a second number (0 = below threshold, 50 = above threshold) separated by a space
- test your code by varying the threshold below/above the room temperature
  - the serial plotter should now draw a second line indicating whether below or above threshold. Vary your threshold slowly below and above the room temperature and make a screenshot of the resulting graph.
- optional: while turning the potentiometer, you can also print the threshold temperature on the display in the second row below the measured temperature

Don't forget to add the code for this exercise to your report!

3.7 Data Handling

Even though one can use the Arduino IDE to look at the values from the temperature sensor, there is no way to save them. That is why we encourage you to write a short program to save the data to file so that you can use it afterwards. We recommend you to use python for this purpose, but feel free to use whatever you have the most experience with. The easiest solution for Windows is the "type" command, which is explained below.

Once a sketch is uploaded to the microprocessor the Arduino will perform the loop until it is disconnected from power or overwritten by a new sketch. Note down the port number or name, which can be seen under "Tools" in the Arduino IDE. In order to read the data externally from the serial port you have to close the Arduino serial tools (Serial Monitor, Serial Plotter, ...).

3.7.1 With Python

Python needs the **pyserial** package to be able to read from the serial interface.
If you are using **anaconda**, type the following command in your anaconda shell:

```
conda install -c anaconda pyserial
```

If you are using **pip**:

```
pip install pyserial
```

You can test if you have installed the **pyserial** package correctly by typing "import serial" into a Python shell. After this works, you can start writing you Python code. An outline of what the code should do is below:

- open the serial port
  - `from serial import Serial`

  - arduino = Serial('/dev/<arduino_portname>')
  - you can find the name of the port in the Arduino IDE
  - read a single line using the serial method arduino.readline().decode('utf-8')
- print the line in the terminal using print
- write the data to a text file (`data.txt`), every value on a single line
  - if you have trouble handling files in python follow this guide
- your program should repeat this until you terminate it e.g. by pressing CTRL+C.
- optional: add timestamps to every line in your log-file. This makes it easier later analyze the data.

On Windows, the Arduino is typically called "COM3", so do arduino = Serial("COM3") to open it.

3.7.2 On Windows XP lab computers

If your port is COM3, you can type the following simple command into a command shell (Start > Run > cmd):

```
type com3: > output.log
```

There is a small delay in writing as data is written in blocks. It can also happen that the last line is incomplete, take this into account in the analysis of the file. You can stop the program by pressing CTRL+break in the command shell. You can use a USB stick to transfer the resulting file to your personal computer to proceed with the analysis.

3.7.3 bash (Linux, Mac OSX

If you use Linux or Mac, you can do all of the above with a single line of bash. Instead of "/dev/cu.usbmodem14141" put your device name (which is listed as port in the Arduino IDE), which can be different from this.

```
while read -r line; do echo $line; done < /dev/cu.usbmodem14141 | tee "output.txt"
```

3.8 Building Your Own Temperature Sensor

In this task you will build your own temperature sensor using discrete components. The B57164-K104-J thermistor has the specifications listed in Table 5.

Specification	Value
Operating voltage	$3.3 \sim 5.0\,\mathrm{V}$
Zero power resistance	$(100 \pm 5)\,\mathrm{k\Omega}$
Operating temperature range	$-55 \sim +125\,^{\circ}\mathrm{C}$
Nominal B-constant	$(4600 \pm 1340)\,\mathrm{K}$

Table 5: B57164-K104-J specifications.

- build a voltage divider circuit to convert the resistance into a measurable voltage
- use the LM358 operational amplifier (with gain 1) to amplify the sensor signal.
- reproduce the temperature measurements from subsection 3.6

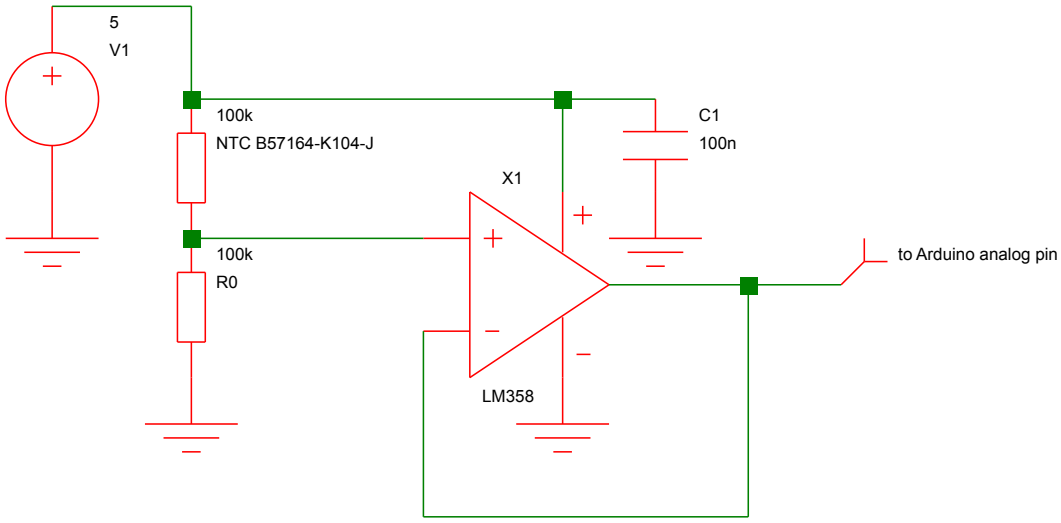


Figure 10: Circuit for reading the NTC resistance with Op-amp.

Figure 10 shows the simplest possible circuit you can use for this purpose. The high input impedance of the op-amp ensures that there is basically no load on the voltage divider and we can use the simple formula which is only valid without load. There is then also virtually no current flowing through the NTC resistor which would heat it up.

3.9 Building a Heating System

Since it is boring to measure a constant temperature, we build a simple heat load of 0.2-0.25 W with a couple of resistors.

- calculate the resistors needed to have the correct power dissipation.
- bring the heating resistor and the temperature sensor close together to have as good thermal contact as possible.
- create a plot of the temperature vs. time (starting from room temperature) and measure the time constant of the temperature rise and estimate the equilibrium temperature.

3.10 Building a Cooling System

Many electrical components will produce heat under load and may break at critical temperatures. That is why many of complex systems like computers require cooling. You will now build a system that controls a fan and can regulate it's rotation speed.

- connect the fan to the breadboard, for the 4 pin header, the following conventions are used:
  - black: ground
  - yellow: 5V
  - green: tacho read-out (will be used later)
  - blue: PWM control
- keep in mind that only a few of the digital pins (marked by "~" on the board) are able to use PWM, e.g. use digital pin 11

Now we can finally build the full two-point controller.

- define a low and high threshold, e.g. use the potentiometer for the high threshold and set the lower threshold a few degrees lower than the high one.
- set the duty cycle of the fan to 100% when above the high threshold
- set the duty cycle of the fan to 0% when below the low threshold
- setting the PWM duty-cycle can be done using the `analogWrite()` function
- if thresholds are set reasonably, you will see an oscillatory behavior. **Create a plot of temperature vs. time which shows multiple periods.**

Experiment with different set-points, what is the advantage of having two setpoints instead of a single threshold above we turn cooling on? What are the draw-backs of the two-point controller?

3.11 Read Out the Fan Speed

We now wan to use the built-in Hall Effect Sensor (HES) of the fan to measure its rotation speed. In every rotation this sensor will produce two pulses we can count and then convert to revolutions per minute (**RPM**). The maximum (for 100% duty cycle) according to the datasheet is 1900 RPM, with a tolerance of 10%, the minimum is around 230 RPM.

- calculate a rough estimate for the minimum frequency needed for reading the voltage on the tacho pin and be able to count the pulses? (Use Nyquist theorem) In practice, a much higher frequency should be used than the limit calculated above
- check the Arduino reference of `analogRead()` for the maximum frequency. Since we also have to do other work in the `loop()` function, the frequency should be also much less than the maximum. Make a reasonable choice.
- what is the expected uncertainty on the minimum and maximum RPM if we count pulses for 1 second?
- connect the tacho (green wire) of the fan to an analog pin of the Arduino, using a 10k pull-up resistor to 5V. (optional: figure out how to use internal pull-up resistor of the Arduino instead of a discrete component)

- what is the role of the pull-up resistor?
- count the pulses of the HES and convert the result to RPM. Is it within the range expected by the datasheet numbers?
- write the revolutions per minute (RPM) value to the serial output

3.12 Final measurements

Now that our device has all basic functionalities, we can perform some measurements with it.

- plot the fan RPM vs. the duty cycle. Which is the minimum duty cycle above which the fan starts to spin, and at which RPM? What is the maximum RPM for full duty-cycle?
- plot the equilibrium temperature vs. the duty cycle and the equilibrium temperature vs. the fan RPM. Make sure you measure long enough for each data point and give a reasonable estimate for the uncertainty that you obtain on the equilibrium value.

Scanning the points for the different duty cycles should be done without flashing the device in between. Instead, the measurement programme (e.g. number of steps, seconds per step etc.) should be programmed into the Arduino or, alternatively, controlled by a Python program running on the computer which changes the parameters via the serial interface (see "Bi-directional communication with the computer" section below).

After you are done with above measurement, pick one of the "Advanced" topics below or come up with your own idea. It can be a new measurement, involve new hardware, or be an improved analysis of the data measured already.

3.13 Building a PID Controller (Advanced)

We have seen that with the two-point controller above, we could not exactly stabilize to a constant temperature and were suffering from under- and overshoot. Using a PID controller can overcome both issues.

- implement a PID controller
- regulate cooling by the PWM duty-cycle of your fan
- tune your three parameters to have reasonably stable operation. (Perfect tuning is very complicated and does not have to done here.)
- create some plots with temperature vs. time, starting from different initial temperatures, which show converging temperature.

The output of the PID controller consists of three different terms, each having a tuneable coefficient.

- proportional: proportional to the error, which is `error = temperature - setpoint`
- integral: proportional to the sum of all errors of previous steps
- derivative: proportional to the difference in error compared to last step

In practice, there are few more things to consider

- using a timer, perform the PID calculations between every 100 ms and every second. You should not use longer values to be fast enough in response and not too much shorter values to be not susceptible to noise.
- use a discrete digital low-pass filter on your measured temperature if it is noisy.
- convert your PID response to a duty_cycle between 0 and 255 to be used in analogWrite(PIN_FAN_PWM, duty_cycle);

A low-pass filter can be used to filter out high frequency fluctuations (noise) on the measured temperature. It can be implemented in it's simplest form with

```
temperature = (1-LOWPASS_ALPHA) * temperature + LOWPASS_ALPHA * (
    temperature_current - temperature);
```

where temperature_current is the actual reading from the sensor and temperature (which has to be properly initialized once to be the sensor reading!) the output of the filter. LOWPASS_ALPHA controls the cut-off frequency of the lowpass filter and obviously depends on how often per seconds we measure. Set it to a reasonable value which suppresses the noise and is still fast enough for the timescale of temperature changes we expect (roughly few degrees per second). It can also be computed analytically:

$$\alpha = \frac{\Delta t}{\tau + \Delta t} \tag{5}$$

where τ is the characteristic time constant of the low-pass filter (corresponds to RC time of a RC filter). Setting $\alpha = 0$ makes it infinitely fast, so as consequence disables the filter, where $\alpha = 1$ would make it infinitely slow, so the output would stay constant.

3.14 Fit heating/cooling(Advanced)

Take the two-point controller and set the two setpoints far from each other, but still in achievable range. For heating and cooling process, find a suitable function to describe the data and perform a fit of the model to data. List the fit parameters (with uncertainties!) and repeat this for several cycles. Are the uncertainties obtained by repeating the experiment consistent with the uncertainties from the fit? If not discuss why.

3.15 Bi-directional communication with the computer (Advanced)

Until now we only read values from the device, but we can also write commands from the computer to the Arduino using the serial interface.

- instead of using the potentiometer to set the threshold, implement a way to set the threshold via serial commands
- one can use the SCPI protocol as a reference, e.g. set the threshold to 32 degrees with the command "SET:THR 32"

- which other commands are necessary to run the measurements from above section without hardcoding the measurement programme (e.g. number of steps, seconds per step etc.)?
- write a Python script to perform the measurements from above section via sending SCPI style commands to the Arduino

3.16 Use of other components (Advanced)

There is also a bunch of other components available which can be used to extend your Arduino project. Among them:

- ultra-sonic distance sensor
- electric current sensor
- infrared leds and sensor
- electro magnet
- Piezzo vibration sensor
- 96x96 pixels OLED display
- Servo motor
- ...

Ask the assistants for more information if you are interested in using them. For example you could use the distance sensor and servo motor to keep an object at a fixed position relative to another moving one.

3.17 I2C protocoll debugging (Advanced)

Many of the Grove components communicate via I2C bus protocol. Using a digital oscilloscope, it is possible to look at the I2C signal and manually decode it on the scope. Use the Grove RBG LCD display, initialize it and periodically write text to the display and set the color. Spy on the commuication, find out the the device IDs of the 2 devices connected to the bus. What do they correspond to? Compare with the library of the Grove display. Record a short communication of a few bytes and decode it.

4 Analysis / Protocol

The analysis and the following protocol should be done such that the reader is able to reconstruct the described circuit and is able to proof that it is working correctly. That includes besides the exact description of all the used parts:

- A link to the repository with all the used sketches. The code should be written in a readable fashion according to C++ coding guidelines with meaningful denotations and sufficient comments.

- Circuit diagrams of **ALL** circuits. A circuit diagram consists of international standardised symbols for all parts in the circuit.

Besides you should answer all questions posed in section 3 describing the approach. Do not forget the results and a conclusion!

List of Figures

List of Tables

List of Acronyms

IDE integrated development environment

DIY do-it-yourself

USB Universal Serial Bus

I/O input/output

GND ground

RST reset

PWR power

IC integrated circuit

VCC voltage common collector

OS operating system

PTC positive temperature coefficient

NTC negative temperature coefficient

ADC analogue-to-digital converter

BJT bipolar junction transistor

PWM pulse width modulation

PID proportional–integral–derivative

RPM revolutions per minute

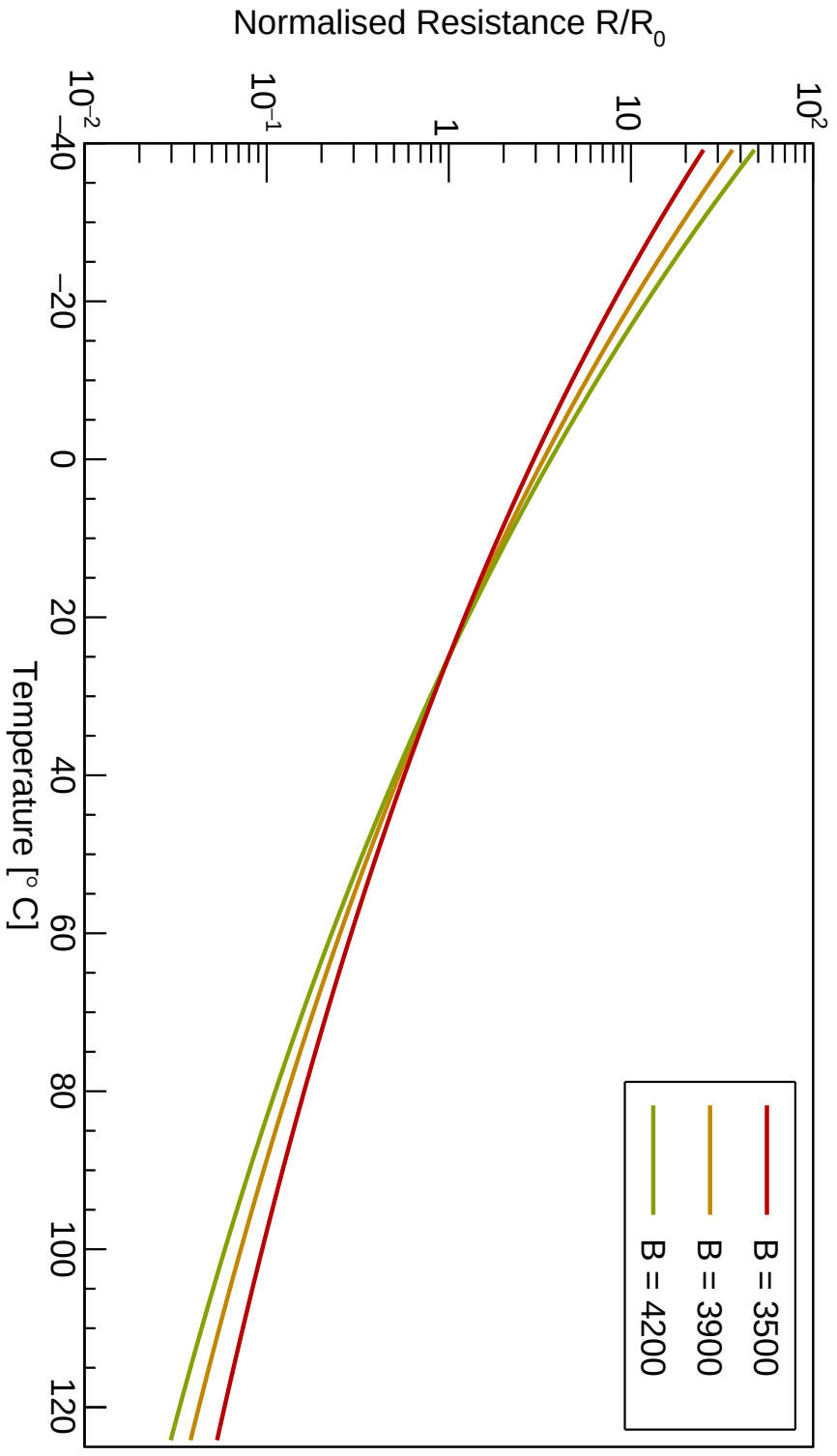


Figure 6: Resistance/temperature characteristic of a NTC thermistor.

(a) NPN (b) PNP

Figure 7: Electronic symbols of the BJT.

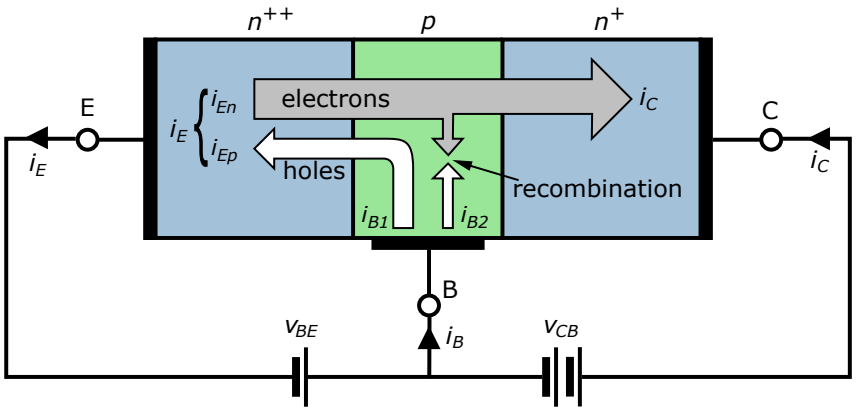
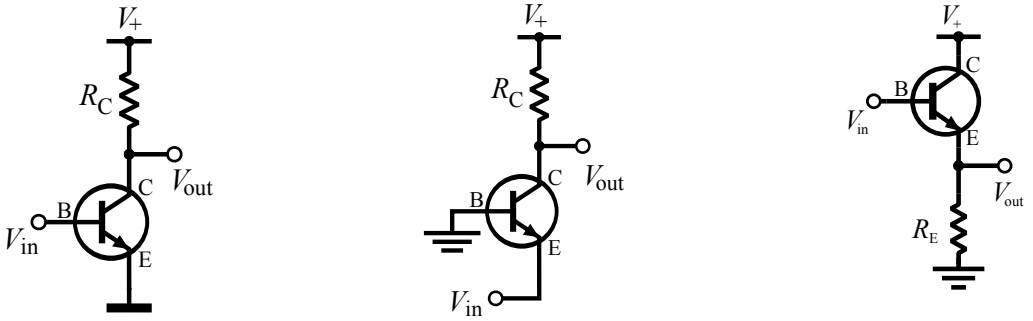


Figure 8: Diagrammatic representation of the amplifying action of a transistor.



(a) Common emitter. (b) Common base. (c) Common collector.

Figure 9: The three basic amplifier arrangements.