Physics Lab

ETH zürich

D PHYS

Version 2.0.0
Manual Number: 88

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 "Electronic Analog+Digital" provides an introduction into modern data taking by reading out analog sensors and control devices with an ATmega328P micro-controller using the Arduino Uno platform. This manual will introduce you to the Arduino board and programming, 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 required for measuring the temperature with an NTC, including an operational amplifier. Furthermore you will control a fans speed via pulse width modulation and can build a regulator to keep the temperature of an object constant. Eventually you will use the Arduino to build a PID controller, which is a widely used technique.

In case you should already have previous knowledge we will provide additional components you can use in your experiment. Own ideas for other applications of the Arduino are very welcome and can be built after consulting the assistants.

ETH Zürich, Switzerland, February 23, 2021

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 [12].

In order to work with the Arduino Boards the Arduino programming language, based on Wiring, and the Arduino Software (integrated development environment (IDE)), based on the Processing are used [1]. 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 course you will work the Arduino Uno shown in Figure 1.

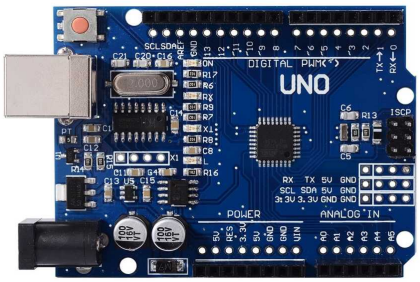


Figure 1: Arduino Uno.

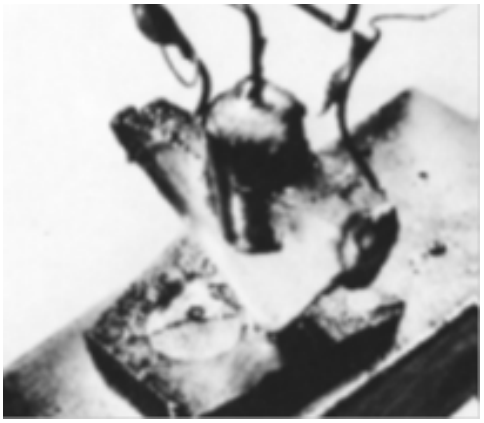
The Arduino Boards use a variety of microprocessors and controllers and are equipped with sets of digital and analogue input/output (I/O) pins that may be interfaced to various expansion boards or Breadboards (shields) and other circuits. The boards feature serial communications interfaces, Universal 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 [8].

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) [3].

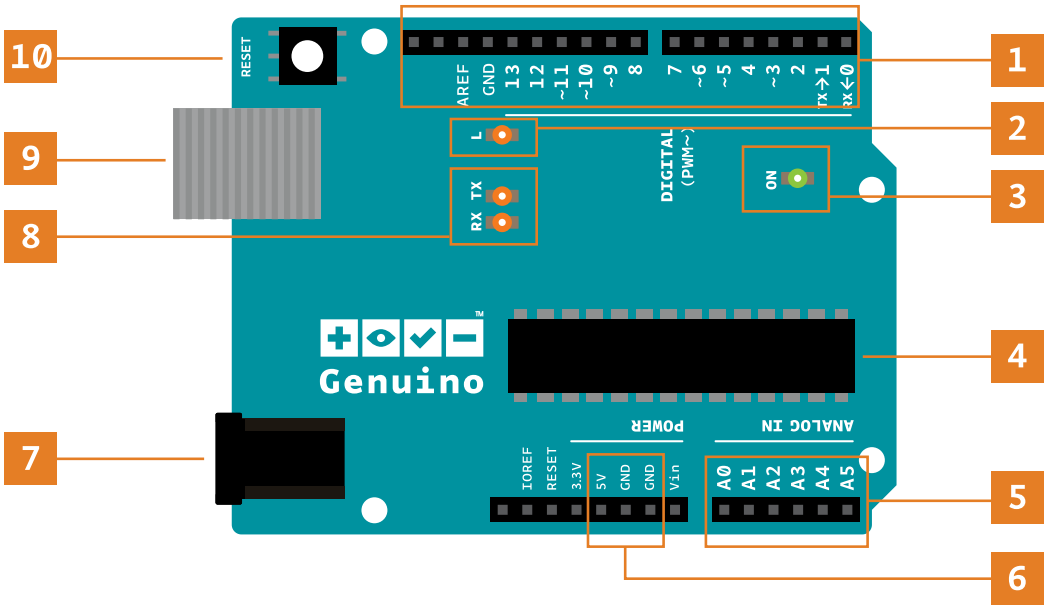
Although billions of individually packaged (discrete) transistors are produced every year the vast majority of transistors are now produced in integrated circuits (ICs). A logic gate consists of up to about twenty transistors whereas an advanced microprocessor, as of 2009, can use as many as 3 billion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

The working principle of the transistor will be covered in subsection 2.10.

2 Basics

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

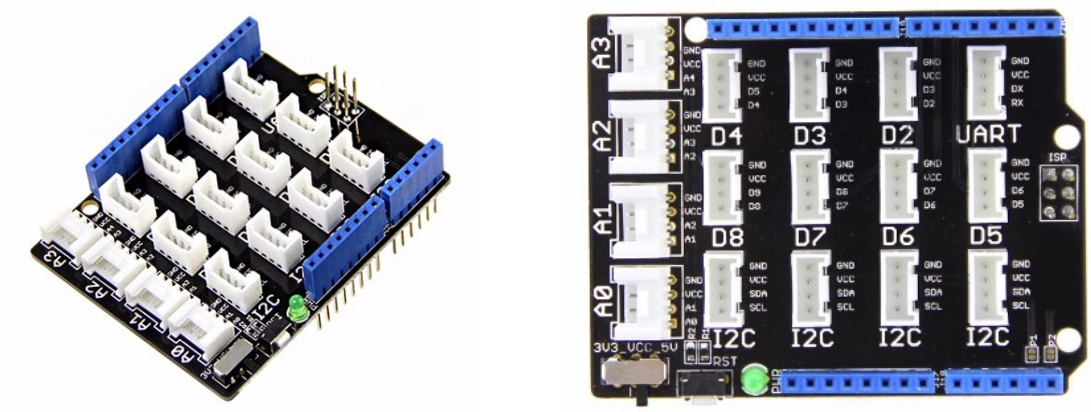
2.1 Arduino Uno



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 1. 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
Analogue	A0/A1/A2/A3	4
Digital	D2/D3/D4/D5/D6/D7/D8	7
UART	UART	1
I^2C	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 [10].

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

For the programming of the micro-controller we use the **Arduino Language**, which is mostly the same as **C/C+ +** . 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 the following. For more information look at the detailed description.

Sketch

```
void setup() {}  // called once at the start of the sketch, used to
↪  initialise variables, pin modes, etc.
```

```
void 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 [mode]
```

Analogue I/O

```
analogRead(pin)  //  reads the value (0-1023) from the [pin]
```

```
analogWrite(pin, value)  // writes an analogue [value] to the [pin] (~)
```

```
analogReference(type)  // configures the reference voltage [type] 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] in ms
```

```
micros()  // :returns: time in us since starting the program
```

```
millis()  // :returns: time in ms since starting the program
```

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)  // :returns: the absolute [value]
```

Serial

```
Serial.begin(speed)  // initialises serial communication at [speed] in 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()  // :returns: incoming serial data
```

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].

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.

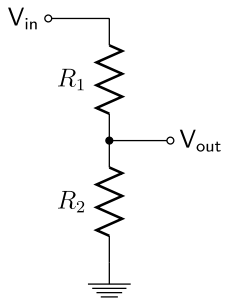


Figure 4: Voltage divider.

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 Low-Pass Filter

In general, an electronic filter is a device, that either passes or attenuates a signal depending on the frequency of the signal. All filters have a range of frequencies, which they completely pass or completely stop. The frequency at which the filter switches from pass to no pass (or vice versa) is called cutoff frequency ω_c [5].

A low-pass filter (LPF) is designed to pass all frequencies below the cutoff frequency and stop all frequencies above. A simple LPF implemented with an RC circuit is shown in Figure 5. Using Kirchhoff's, the following relation between the input voltage v_{in} and the output voltage v_{out} can be derived:

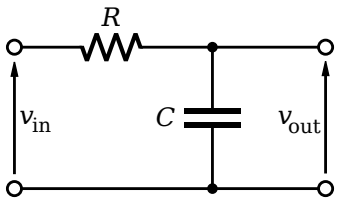


Figure 5: Simple low-pass RC filter.

$$v_{\text{in}}(t) - v_{\text{out}}(t) = RC\frac{dv_{\text{out}}}{dt} = \frac{1}{\omega_c}\frac{dv_{\text{out}}}{dt} = \tau\frac{dv_{\text{out}}}{dt} \tag{2}$$

where time constant τ of the LPF is the inverse of the cutoff frequency.

The frequency response of a LPF is in general characterised be the Laplace transform of the transfer function with follows from Equation 2:

$$H(s) = \frac{V_{\text{out}}(s)}{V_{\text{in}}(s)} = \frac{\omega_c}{s + \omega_c} \tag{3}$$

Also digital filters can be designed to give low-pass characteristics. Assuming that the samples at the input and the output are taken at a constant sampling time Δt, it is simple to discretise Equation 2:

$$x_i - y_i = \tau\frac{y_i - y_{i-1}}{\Delta t} \tag{4}$$

where v_{in} is represented by the sequence $(x_0, x_1, \ldots, x_n)$ and v_{out} represented by $(y_0, y_1, \ldots, y_n)$. By rearranging the terms an equation for the current output can be derived:

$$y_i = \underbrace{x_i\left(\frac{\Delta t}{\tau + \Delta t}\right)}_{\text{input}} + \underbrace{y_{i-1}\left(\frac{\tau}{\tau + \Delta t}\right)}_{\text{previous output}} \tag{5}$$

This shows that the discrete implementation of a simple LPF is equivalent to the the exponentially weighted average:

$$y_i = \alpha x_i + (1 - \alpha)y_{i-1}, \quad \text{where} \quad \alpha := \frac{\Delta t}{\tau + \Delta t}. \tag{6}$$

By definition the smoothing factor α lies in the interval $0 \leq \alpha \leq 1$. If $\alpha = 0.5$, the time constant τ is equal to the sampling period. Setting $\alpha = 1$ makes the system infinitely fast,

so the output is equal to the input, whereas if $\alpha = 0$, the response is infinitely slow, so the output would stay constant at its initial value.

From Equation 6 follows, that the cutoff frequency of the LPF may be adjusted by choosing appropriate values for α and Δt.

2.8 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*.

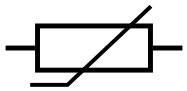


Figure 6: Electronic symbol of the thermistor

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 6. The resistance is given by

$$R = R_0 \cdot \mathrm{e}^{-B\left(\frac{1}{T_0}-\frac{1}{T}\right)} \,, \tag{7}$$

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 ($25\,^{\circ}\mathrm{C} = 298.15\,\mathrm{K}$). 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.

2.9 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 its 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

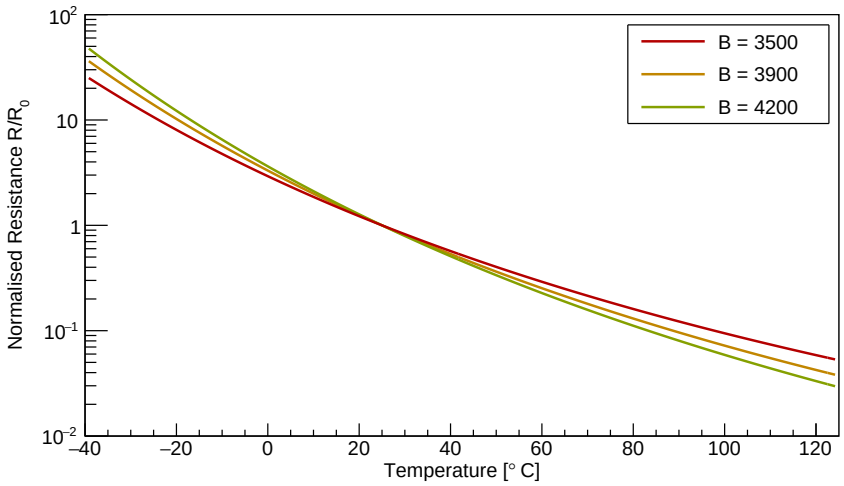


Figure 7: Resistance/temperature characteristic of a NTC thermistor.

to V_{in}. In order to convert it into the resistance of the thermistor we have to use Equation 1:

$$R = \left(\frac{V_{\text{in}}}{V_{\text{out}}} - 1\right) \cdot R_0 = \left(\frac{1023}{V_{\text{meas}}} - 1\right) \cdot R_0 \tag{8}$$

Now we need to convert the resistance into the temperature using Equation 7:

$$\frac{1}{T} = \frac{\ln\left(\frac{R}{R_0}\right)}{B} + \frac{1}{T_0} \tag{9}$$

Note that this temperature will be in K!

2.10 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 electronic symbols of theses two types are shown in Figure 8. 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.

(a) NPN (b) PNP

Figure 8: Electronic symbols of the BJT.

2.10.1 Working Principle

Since a transistor consists of two pn junctions within a single crystal, transistor action can be explained with Figure 9. 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.

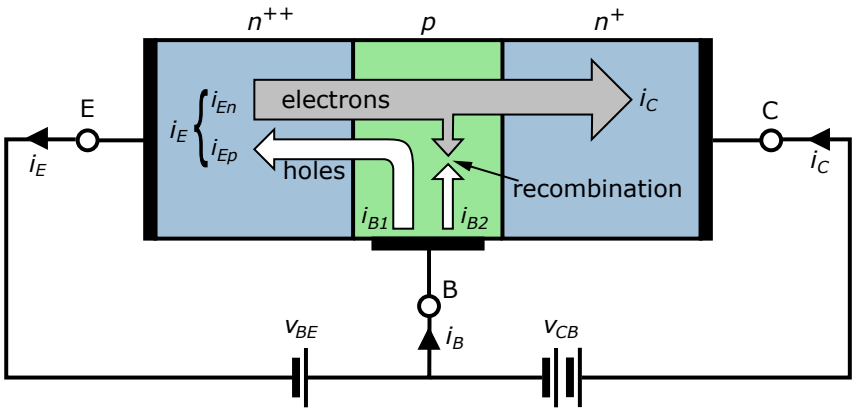


Figure 9: Diagrammatic representation of the amplifying action of a transistor.

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.

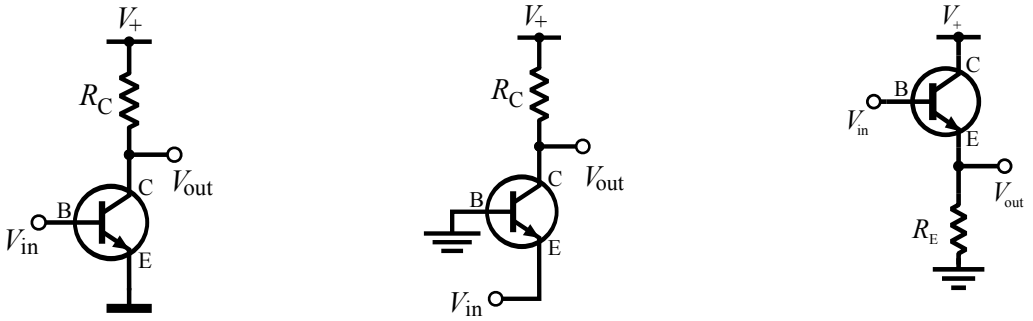
The injection of a hole into the base region by a signal source will now be considered. Once in the base, the hole attracts an electron from the emitter region. The recombination of the hole and electron is not likely to occur however, since the base region is lightly doped compared with the emitter 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 10 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 [8].



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

Figure 10: The three basic amplifier arrangements.

2.10.2 Common Collector

The common collector circuit shown in Figure 10c 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

2.11 Operational Amplifier

Operational amplifiers (op-amps) are used for various purposes in electronics. The amplifier takes a differential input voltage ($V_{\mathrm{in+}}$ and $V_{\mathrm{in\text{-}}}$) and usually amplifies it to a single-ended output voltage (V_{out}). A typical example is shown in Figure 11.

Its operating mode is determined by the way the output (right side) is connected to the inputs (left side). This connection between input and output is called *feedback*. We will use an op-amp as *voltage follower* (or also called buffer), which is a special case of a non-inverting amplifier with a gain (voltage amplification) of 1. It is used to decouple the two parts of the circuit, the measurement side and the read-out (digitisation) side. Without such decoupling, the current drawn from the Arduino analogue input pin could influence the voltage drop over the NTC and therefore deteriorate the measurement.

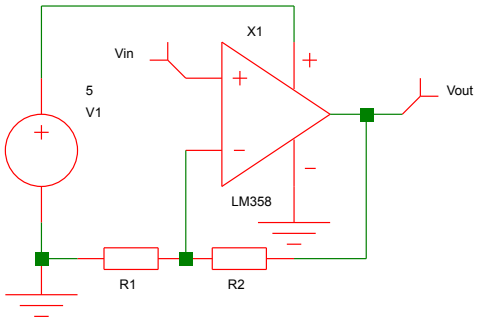


Figure 11: op-amp as non-inverting amplifier.

Task 1: Calculate V_{out} of op-amp

Calculate V_{out} based on V_{in} in the non-inverting amplifier circuit shown in figure Figure 11. Show that for $R_2 = 0$ you obtain the voltage follower: $V_{out} = V_{in}$!

For calculations with op-amps you can use the following rules:

- no current flows into the two **inputs** (left side) of the op-amp. It gets all its current from the **supply** (top and bottom)
- the op-amp puts both inputs at the same potential ("virtual-short")

With these two rules and Kirchhoff's law, you can then calculate the output voltage V_{out}.

This type of voltage follower circuit has a very high input impedance and therefore only draws very little current from the circuit with the NTC resistor. This allows us to use the formulas for an unloaded voltage-divider in the calculation and it also limits the current through the NTC, which could heat it up.

2.12 Pulse Width Modulation

PWM is a technique, to transmit information (or power) using a square-wave signal with fixed period and amplitude, but varying pulse-width. The fraction of time during which the signal is at the high level is called *duty cycle*. PWM signals with different duty cycles are shown in figure Figure 12. The frequency of the PWM signal, which is created with the `analogWrite()` command, is fixed to ~490 Hz. This is fast enough for our purpose, however it is also possible to change this frequency, see e.g. in [9].

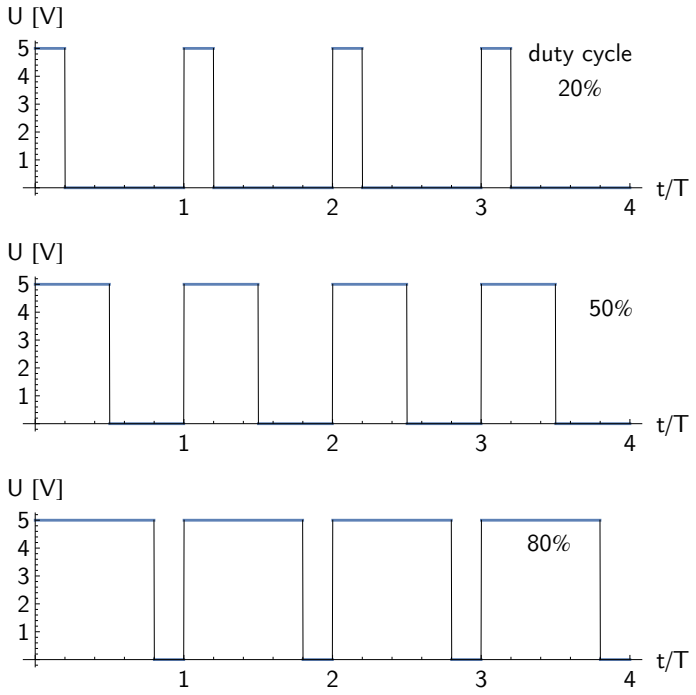


Figure 12: PWM signals with different duty cycles 20 %, 50 % and 80 %.

2.13 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 $e(t)$ as the difference between a desired setpoint and a measured process variable 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 [13].

3 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

The full setup is contained in the box shown in Figure 13. You will find this box on your assigned lab space in room H 41 in the HPP building. The box contains the items shown in Figure 14, and the contents of the Grove starter kit are shown in Figure 15. Manuals for several devices may be found in Table 2:

Figure 13: Utz box containing the full setup.

Item	Type	Manual
multimeter	UT61C	Link
4-pin fan	Noctua NF-A12x25 5V PWM	Link
op-amp	LM358-N	Link
NTC thermistor	B57164-K104-J	Link

Table 2: Item manuals.

Note 1: Lab notes

In addition you should bring a lab book (or any other form of notepad) 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!

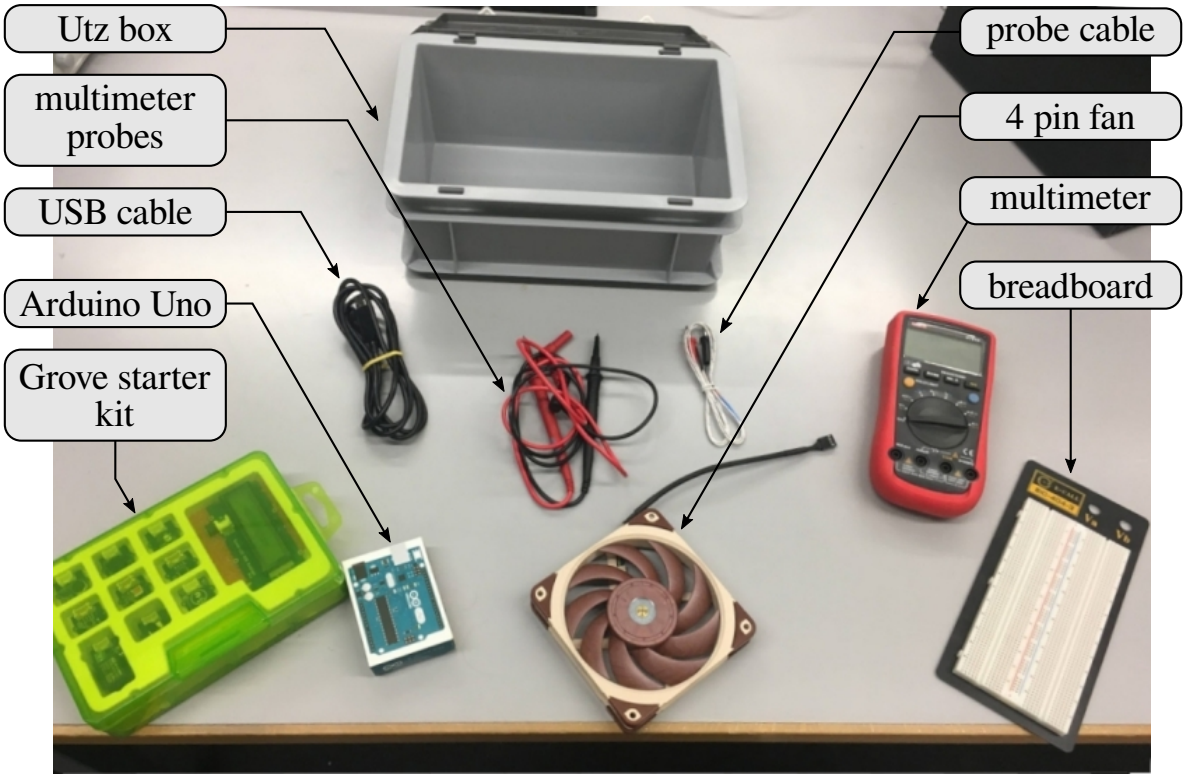


Figure 14: Content of the Utz box.

Note 2: Tidy return

If you finish your experiment, please return the setup as shown in Figure 13, Figure 14 and Figure 15. If anything broke during the experiment, please inform the assistant, so that it can be replaced.

3.2 Working from home

Due to the compactness of the setup it is possible for you to bring the setup to your place and do the measurements at home. In order to do so, please fill the form and send it signed to the assistant. It is, of course, also possible to do the experiment at the designated lab space at ETH.

The setup should contain all required items for the experiment. If something is missing or you require additional parts, please contact the technical assistants at the help-desk in HPP J 14. If you want to take additional parts home, please also contact the assistant.

3.3 Setting up the Arduino

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 not to use your computer for the experiment, you can use one of the Windows XP machines provided to you at the lab place, which have the Arduino IDE and all necessary drivers pre-installed. These

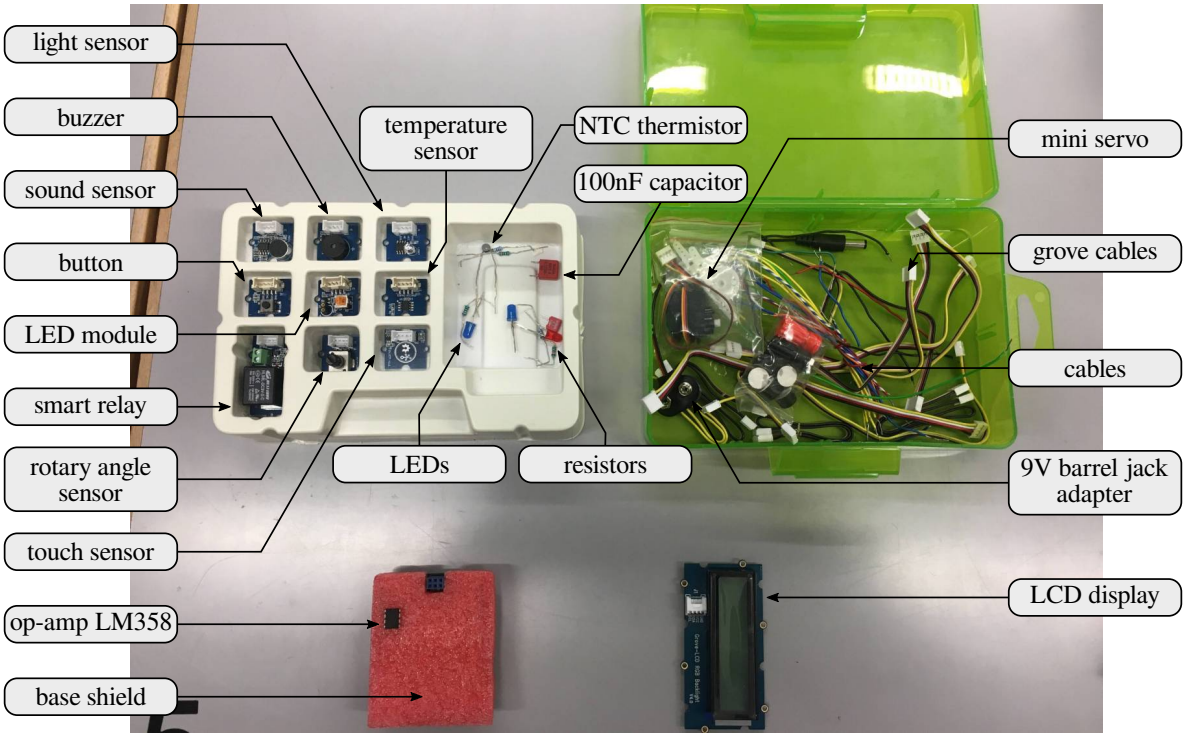


Figure 15: Content of the Grove starter kit. The base shield is under the red protective foam.

machines do not have internet connection, so you have to bring a portable USB drive to copy the data.

Task 2: IDE installation

Install the Arduino IDE following subsection 2.3!

Task 3: Connecting the Arduino

Connect the Arduino to your computer via the USB cable and check if it is recognised by the IDE!

- **select the correct port:** `Tools > Port`
- **get board info:** `Tools > Get Board Info`

3.4 Your First Sketch

In order to have a better overview of your own sketches, it is recommended to set up your own sketchbook location: `File > Preferences > Sketchbook Location`.

Task 4: Creating the first sketch

Create a new sketch with `File > New`!

A new sketch will look like this:

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Now we want to turn on the LED on the Arduino board, which is connected to the special pin `LED_BUILTIN`. Since this pin is used as an output, we have to set the `pinMode` during the `setup()` function to `OUTPUT`. Afterwards we can use the `digitalWrite()` method to set the pin to `HIGH` (corresponding to 5 V), which will turn on the LED on the board. There is no need to use the `loop()` function for this simple sketch, but it will be important later.

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, HIGH);
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

That's it. All the methods and constants used in this script are already defined and do not have to be imported explicitly. There is a long list of such predefined 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 it is not already selected.

Task 5: Uploading the first sketch

First compile the code and then upload it to the Arduino. If everything is working, create a git repository in your sketch folder and commit your first sketch!

Note 3: Do not use pins A4 & A5

The Arduino Uno uses pins A4 and A5 for the inter-integrated circuit (I^2C) 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!

3.5 Blinking LED on Bread Board

That was easy (but boring ...). As the 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.

Task 6: Writing a blinking LED sketch

Fist supply the breadboard with a voltage of 5 V from the Arduino. Then connect a LED to the breadboard with an appropriate series resistor (220 Ω) and to a digital pin of the Arduino. Finally 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`. Upload your sketch now to the Arduino and test it!

Task 7: Adding a second LED

Now 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, upload it and test it.

Why is `delay()` not the optimal method for this task? Try to use a timer with `micros()`, which returns a micro-seconds counter as `unsigned long` to solve this problem. In order to save the result from `micros` you have to save it to a `unsiged long` variable. After ~70 min, the 32-bit counter will overflow and start again from 0! Make sure to handle this case well!

Task 8: Adding code to the report

If you succeeded to implement the two blinking LEDs with timers, please add the code to your report and to your git repository.

3.6 General tips for writing code for the Arduino

3.6.1 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 is 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.

Type	Size	Range
`char`	8-bit	$-128 \sim 127$
`unsigned char`	8-bit	$0 \sim 255$
`int`	16-bit	$-32\,768 \sim 32\,767$
`unsigned int`	16-bit	$0 \sim 65\,535$
`long`	32-bit	$-2\,147\,483\,648 \sim 2\,147\,483\,647$
`unsigned long`	32-bit	$0 \sim 4\,294\,967\,295$
`float`	32-bit	floating point
`double`	32-bit	floating point

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

Note 4: Datatypes

Be sure to use `unsigned long` to store the output of microsecond counters like `micros()`!

3.6.2 Avoid hard-coded values

Avoid of a "hard-coded" numbers somewhere in the middle of your program, like e. g.

```
...
if (now > lastPid + 100000) {
...
```

Rather define a constant (with `const` or as a macro with `#define` in the beginning of the program. Give it a systematic name, like all uppercase for constants, include the unit (e. g. microseconds) and add a comment:

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

and then use the constant (`INTERVAL_MICROS_PID`) in the `if` clause. This makes the code easier to read. Macros with `#define` do not use space for variables (in contrast to a declaration with `int ...`), which reduces the 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 that you can store! Note, that part of this memory is already used by libraries. Using constants instead of hard-coded numbers is especially important for pin numbers.

3.6.3 Code documentation

Add a short description on the functionality of your program at the top of the code. Include the date and author name. Also comment any non-trivial steps in your code. (Remember: it

is in general better to write code in a self-explaining way, such that no comments are necessary. Sometimes though, comments are unavoidable.)

Task 9: Code documentation

Always add reasonable documentation to your code, so that it is easy for another person to understand each step of it!

3.7 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 of the sensor 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: Specifications of the Grove-Temperature Sensor V1.2.

Task 10: Connecting the Grove temperature sensor

First connect the Grove Base Shield to your Arduino. Then find out which connector on the shied you have to use to connect the temperature sensor.

Task 11: Measuring the ambient temperature

Create a new sketch and name it `GroveTemp.ino`! In order to measure the temperature, you have to first read the voltage value from the sensor, then convert the voltage into the corresponding resistance and then convert the resistance to a temperature in °C. Use the serial interface to write one temperature reading every second and both investigate the data with the *Arduino Serial Monitor*: `Tools > Serial Monitor` (Ctrl+Shift+M) and the *Arduino Serial Plotter*: `Tools > Serial Plotter` (Ctrl+Shift+P)

Note 5: Baud rate

The serial connection has to be enabled during the `setup()` function with `Serial.begin(9600);` where 9600 is the baud rate. 9600 is the default value for the *Arduino Serial Plotter* and *Arduino Serial Monitor* utilities, so it is convenient to use this.

3.8 Grove Display and Potentiometer

As a next step, we will add two more components from the Grove kit: an LCD display to show the current temperature reading of the sensor and a potentiometer to adjust the threshold for our two-point temperature control later.

Task 12: Adding a LCD display

First install the libraries from the Grove website. Include the libraries to your sketch and initialise the display in the `setup()` function, based on the example code on the above website. Connect the Grove LCD RGB Backlight display to your Grove Shield and modify your project, such that the measured temperature is printed on the display and updated every second.

Note 6: VCC switch

The LCD display will only work correctly, if the *3V3_VCC_5V* switch on the Grove Shield is setup to 5 V.

Task 13: Temperature threshold

1. **Define a fixed threshold in your code, e. g. 30 °C, above which a LED is turned on. As an alternative you can also change the background colour of the display. (Always test if your code is working!)**
2. **Make the threshold adjustable without recompiling the code! For this purpose, connect a potentiometer, e. g. the Grove Rotary Angle Sensor, to the Grove Shield. Using the description, read out the sensor with `analogRead()` and map the ADC values (0 ∼ 1023) to a reasonable temperature range (a ∼ b), e. g. 20 ∼ 40 °C. This can be implemented, e. g. , with the method `map(adc, 0, 1023, a, b)`.**
3. **Indicate the status (below/above threshold) also in the output to the serial interface. For example, add a second number (0 = below threshold, 50 = above threshold) separated by a space.**
4. **Test your code by setting the threshold below and above the room temperature! The serial plotter should now draw a second line indicating whether the value is below or above threshold. Vary the threshold slowly below and above the room temperature and make a screenshot of the resulting graph.**
5. **(Optional). Print the temperature threshold on the second row of the display below the measured temperature!**
6. **Add the code to your report!**

3.9 Data handling

Even though the Arduino IDE has tools to display 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 analyse it afterwards. We recommend you to use python for this purpose, but feel free to use whatever you have the most experience with.

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, ...).

3.9.1 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:

```
student@host:~$ conda install -c anaconda pyserial
```

If you are using **pip**:

```
student@host:~$ pip install pyserial
```

You can test if you have installed the `pyserial` package correctly by typing `import serial` into a python shell. If this works, you can start writing you python code. An outline of what the code should do is below (python3):

```
1  #!/usr/bin/python3
2
3  from serial import Serial
4
5  port_name = '<arduino_portname>'  # the name of the port is shown in the IDE
6  arduino = Serial(portname)  # open serial port of the Arduino
7  line = arduino.readline()  # read a single line from the serial output
8
9  def save_line(file_name):
10   """ writes a single temperature measurement to the next line of the file
     ↪ [file_name] ."""
11   pass
12
13 def save_data(file_name):
14   """ opens a file [file_name] and continuously saves the data. """
15   pass
16
17 def add_timestamp(line):  # optional
18   """ adds the current time stamp to the [line] to simplify the analysis.
     ↪ """
19   from datetime import datetime
20   t = str(datetime.now())
```

If you have trouble handling files in python follow this guide. On Windows, the Arduino's serial port is typically called `'COM3'`, so you can open it via `arduino = Serial('COM3')`.

3.9.2 On Windows XP lab computers

The easiest solution for Windows is the **`type`** command. If your port is `COM3`, you can type the following simple command into a command shell (`Start > Run > cmd`):

```
C:\Users\student> type com3: > output.log
```

There is a small delay in writing the data, since it is written in blocks. It can also happen, that the last line is incomplete. Take this into account, when you analyse the file. You can stop the program by pressing Ctrl+break in the command shell. Use a thumb drive to transfer the resulting file to your personal computer to proceed with the analysis.

3.9.3 bash (Linux, Mac OS X)

If you use Linux or Mac, you can do all of the above with a single line of bash. Instead of `/dev/arduino_portname` put your device name, which is listed as port in the Arduino IDE.

```
student@host:~$ while read -r line; do echo $line; done <
↪  /dev/arduino_portname | tee "output.txt"
```

3.10 Op-Amp Circuit

Now you will build your own temperature sensor using a B57164-K104-J NTC resistor and the op-amp LM358-N. The thermistor has the specifications listed in Table 5.

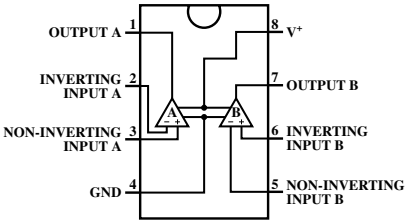
Specification	Value
Operating voltage	$3.3 \sim 5.0\,\mathrm{V}$
Zero power resistance	$(100 \pm 5)\,\mathrm{k\Omega}$
Operating temperature range	$-55 \sim +125\,^{\circ}\mathrm{C}$
Nominal B-constant	$(4600 \pm 138)\,\mathrm{K}$

Table 5: Specification of the B57164-K104-J thermistor.

Task 14: Building a temperature sensor

Build a voltage divider circuit on the breadboard to convert the resistance of the thermistor into a measurable voltage. Use the an op-amp as a voltage follower with gain 1 to amplify this voltage signal (see Figure 16). Reproduce the temperature measurements from subsection 3.7.

Figure 17 shows the simplest possible electrical circuit for this purpose. The high input impedance of the op-amp ensures that there is basically no load on the voltage divider and



(a) Layout.

PIN			TYPE	DESCRIPTION
D/P/LMC NO.	DSBGA NO.	NAME		
1	A1	OUTA	O	Output , Channel A
2	B1	-INA	I	Inverting Input, Channel A
3	C1	+INA	I	Non-Inverting Input, Channel A
4	C2	GND / V-	P	Ground for Single supply configurations. negative supply for dual supply configurations
5	C3	+INB	I	Output, Channel B
6	B3	-INB	I	Inverting Input, Channel B
7	A3	OUTB	O	Non-Inverting Input, Channel B
8	A2	V+	P	Positive Supply

(b) Description.

Figure 16: Pin layout and descriptions of the LM358 op-amp [11].

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 otherwise.

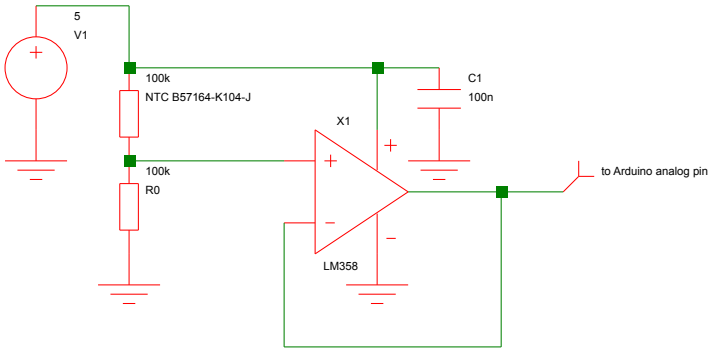


Figure 17: Circuit diagram for reading the NTC resistor with an op-amp.

3.11 Calibration

The NTC resistor we used for this experiment is not calibrated yet and therefore we obtain a relatively large error on the absolute temperature scale.

Task 15: Calculation of the temperature uncertainty

Calculate the uncertainty on the measured temperature from the following sources:

- ***B*-constant of the NTC**
- **100 kΩ resistor (use 1 % resistors if possible)**
- **resolution of the ADC**
- **any other non-negligible uncertainty you can think of**

By calibrating our device with an already calibrated reference device, we can reduce the systematic uncertainties. For this calibration you can use the Grove temperature sensor or any equivalent device.

Task 16: Temperature calibration

Calibrate your device by adjusting the B constant, such that the measured temperature matches the one measured with the reference device. Use the given uncertainty on the temperature of the reference device to calculate a new uncertainty for your B constant. How much does it improve?

3.12 Heating

Since it is boring to monitor a constant temperature, you will build a simple heat load of $0.20 \sim 0.25\,\mathrm{W}$ with some of resistors.

Task 17: Resistance calculation

Calculate the resistance, that is required to have the power dissipation of $0.20 \sim 0.25\,\mathrm{W}$. Bring the heating resistor and the temperature sensor as close as possible together, so that they have a good thermal contact.

Note 7: Temperature difference

The temperature difference between the room temperature and the equilibrium temperature without cooling should be at least 5 K, ideally ∼10 K. If improving the thermal contact is not enough to heat up the NTC by more than 5 K, use the external power supply to provide more power for the heating resistor.

Task 18: Temperature plot

Plot of the temperature versus time (starting from room temperature)! Fit the curve with an appropriate function and determine the time constant of the temperature rise and the equilibrium temperature.

3.13 Cooling

Many electrical components produce heat under load and may break above a critical temperature. That is why many complex systems like computers require cooling. You will now build a system that controls a fan and can regulate its rotation speed (revolutions per minute (RPM)).

Task 19: Connecting the fan

Connect the fan to the breadboard. Follow the conventions in Table 6 for the 4 pin header. Mind that only a few of the digital pins (marked by the ∼ symbol) are able to use PWM. You can, e. g. , use the digital pin 11.

Pin	Wire colour	Usage
1	black	GND
2	yellow	5 V
3	green	sense (tachometer read-out)
4	blue	control (PWM)

Table 6: Intel standard connector pinout for a for 4-wire fan [7].

3.14 Two-point controller

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

Task 20: Building the 2-point controller

Define a low and high temperature threshold, e. g. use the potentiometer for the high threshold and set the lower threshold a few degrees lower than the high one. Mind that both thresholds have to be within the minimum and maximum temperature of your setup. Set the duty cycle of the fan to 100 % when above the high threshold and to 0 % when below threshold.

You can set the PWM duty-cycle of the fan using the `analogWrite()` function. Experiment with different set-points, so that you get a better understanding of the system. What is the advantage of having two set-points instead of a single threshold above which the cooling is turned on? What are the draw-backs of the two-point controller?

Task 21: Temperature plot 2

Plot of temperature versus time with an active two-point controller and show multiple periods of cooling and heating.

3.15 Fan speed

Now you will use the fan's built-in hall effect sensor (HES) to measure its rotation speed. For every rotation this sensor produces two pulses, that can be converted to RPM. The maximum speed (for 100 % duty cycle) is 1900 RPM, with a tolerance of 10 %, the minimum is 230 RPM [6].

Task 22: Sampling theory

1. **Based on the Nyquist-Shannon sampling theorem, estimate the minimum frequency for reading the voltage on the tachometer pin, that is required to count the pulses! In practice, a much higher frequency should be used.**
2. **Figure out, what the maximum frequency of the `analogRead()` method is (see Arduino reference). Since you also have to do other jobs in the `loop()` function, the frequency should be also much less than the maximum. Make a reasonable choice!**

Task 23: Fan speed read-out

1. **Connect the tachometer of the fan to an analogue pin of the Arduino, using a 10 kΩ pull-up resistor to 5 V.**
2. **(Optional). Figure out how to use internal pull-up resistor of the Arduino instead of using a discrete component!**
3. **Describe the role of the pull-up resistor!**
4. **Count the pulses of the HES and convert the it to RPM. Is the measured value in accordance with the expected range from [6]?**
5. **Write the RPM value to the serial output!**

Scanning the points for the different duty cycles should be done without flashing the device in between. Instead, the measurement program (e. g. number of steps, seconds per step etc.) should be programmed into the Arduino code or, alternatively, controlled by a python program, which changes the parameters via the serial interface, as described in subsection 3.19.

Task 24: Plotting RPM vs. duty cycle

Plot the RPM of the fan versus the duty cycle. Which is the minimum duty cycle above which the fan starts to spin, and at how many RPM? What is the maximum RPM for the full duty-cycle?

3.16 Equilibrium temperature

Task 25: Equilibrium temperature

Using the 2-point controller, plot the equilibrium temperature versus the RPM of the fan. Give a reasonable estimate for the uncertainty on the equilibrium temperature.

Note 8: Equilibrium temperature

It can take several minutes to reach a stable equilibrium for each data point, so make sure to wait long enough until the temperature has reached the equilibrium.

3.17 PID Controller

We have seen, that the two-point controller takes a long time to reach to a constant temperature and it suffers from under- and overshoot. Both issues can be overcome using a PID controller.

The output of a PID controller consists of three different terms, each having a tunable coefficient:

1. proportional: proportional to the error, where the error is equal to the difference of the actual temperature and the set-point
2. integral: proportional to the sum of all errors of previous steps
3. derivative: proportional to the difference current error and the error of the last step

In practice, there are few more things to consider.

Task 26: Implementing a PID controller

Regulate cooling by the PWM duty-cycle of your fan! Adjust the three PID parameters so that the controller work reasonably stable. (Perfect tuning is very complicated and does not have to be done here.) Plot the temperature versus time, starting from at least 3 °C below or above the converging temperature.

In order to use the full capacity of the fan, it is important to convert you PID response to a `duty_cycle` between $0 \sim 255$ This way it can be used in the `analogWrite(PIN_FAN_PWM, duty_cycle)` method.

Task 27: Optimising the update interval

1. **Using a timer, the update time should be tuned, so that the response is fast enough while the measurement is not too susceptible to noise. In our case, the PID calculations should be done every $0.1 \sim 1$ s.**
2. **If you notice, that your temperature measurements gets noisy, add a discrete digital low-pass filter (see subsection 2.7) to your code.**

The digital LPF should look similar to this:

```
#define ALPHA 0.5

float read_temp() {...};  // method to make a single temperature reading

float t_out = read_temp();  // initialise the output temperature

void loop(){
...
t_out = ALPHA * ... + ...  // filter every temperature reading

}
```

Remember, that you can adjust the cutoff temperature by adjusting `ALPHA` and the update interval.

Task 28: Optimising the PID parameters

Now optimise the three parameters reasonably well to have fast convergence and low fluctuations after a certain time and plot the temperature versus time. Also detune the parameters to produce slower convergence and/or overshooting/oscillations and show both curves in a single plot.

3.18 Fit heating & cooling (Advanced)

Note 9: Advanced tasks

All of the following tasks are optional. They can be used to replace some of the other tasks if agreed on with the assistant.

Now we want to investigate the functional behaviour of the heating and cooling. For this purpose it is best to set the two temperature set-points far from each other, while still being in achievable range.

Advanced Task 1: Fit of heating & cooling

Find a suitable function to describe the data and perform a fit of the model to the data. Show the fit parameters (with uncertainties!) and repeat this for several heating and cooling cycles.

Always when you fit a model to data, it is important to check if the fit makes sense. In this case it is important to examine if there is a correlation of the input uncertainties.

Advanced Task 2: Interpreting the fit

Explain why the correlation of the input uncertainties poses an issues for this experiment. Also look at the uncertainties and correlations of the fitted parameters. Are the uncertainties obtained by repeating the experiment consistent with the uncertainties from the fit? If not discuss why!

3.19 Bi-directional communication (Advanced)

So far we only read values from the Arduino. However, it is also possible to send commands from the computer to the Arduino using the serial interface. The is usually done via Standard Commands for Programmable Instruments (SCPI) [4], which defines a standard for syntax and commands to use in controlling programmable test and measurement devices.

Sending the string `'SET:THResh 32'` to the serial interface, is a SCPI example to set the threshold to 32. This string then must be read and parsed by the Arduino.

Advanced Task 3: Bi-directional communication

Implement a way to set the threshold via serial commands, instead of using the potentiometer. Write a python script to perform the measurements from above section by sending SCPI to the Arduino.

3.20 Using other components (Advanced)

There are also many other components available, that can be used to extend your Arduino project. There are for example:

- ultra-sonic distance sensor
- electric current sensor
- infrared LEDs and sensor
- electromagnet
- piezoelectric vibration sensor
- 96x96 pixels OLED display
- ...

Advanced Task 4: Building your own PID setup

Build an experiment similar to the temperature control, which also uses a PID controller. For example you could use the distance sensor and servo motor to build a vehicle that keeps a fixed distance to another object.

If you are interested in building your own PID controller system, please first consult the assistant. This task may replace a large fraction of the official tasks above.

3.21 I^2C protocol debugging (Advanced)

Many of the Grove components communicate via I^2C bus protocol. Using an oscilloscope, it is possible to directly look at the I^2C signal and manually decode it.

Advanced Task 5: I^2C decoding

Use the Grove RBG LCD display, initialise it, periodically write text to the display and set the background colour. Spy on the communication, find out the the IDs of the 2 devices connected to the bus. What do they correspond to? Compare with the library of the Grove display. Record a 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 Tasks

List of Advanced Tasks

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

I^2C inter-integrated circuit

op-amp operational amplifier

HES hall effect sensor

LPF low-pass filter

SCPI Standard Commands for Programmable Instruments

References

[1] Arduino. Introduction to Arduino. https://www.arduino.cc/en/Guide/Introduction, 2018. [Online; accessed 15-May-2018].

[2] Scott Chacon and Ben Straub. *Pro git*. Apress, 2014.

[3] Dan Hutcheson. A look at Moore's Law. https://spectrum.ieee.org/computing/hardware/transistor-production-has-reached-astronomical-scales, 2018. [Online; accessed 15-May-2018].

[4] Scpi-1999 specification. https://www.ivifoundation.org/docs/scpi-99.pdf, 2021.

[5] Afshin Izadian. *Passive Filters*, pages 341–398. Springer International Publishing, Cham, 2019.

[6] Nf-a12x25 5v pwm. https://noctua.at/en/products/fan/nf-a12x25-5v-pwm, 2021.

[7] Pin-configuration of noctua products. https://noctua.at/en/productfaqs/productfaq/view/id/215/, 2021.

[8] G.H. Olsen. *Electronics: A General Introduction for the Non-Specialist*. Springer US, 2013.

[9] QEEWiki. Pwm on the atmega328. https://sites.google.com/site/qeewiki/books/avr-guide/pwm-on-the-atmega328, 2018. [Online; accessed 26-Nov-2018].

[10] Seeed. Base Shield V2. https://www.seeedstudio.com/Base-Shield-V2-p-1378.html, 2018. [Online; accessed 15-May-2018].

[11] Lmx58-n low-power, dual-operational amplifiers. http://www.ti.com/lit/ds/symlink/lm158-n.pdf, 2021.

[12] Wikipedia contributors. Arduino — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Arduino&oldid=839287116, 2018. [Online; accessed 15-May-2018].

[13] Wikipedia contributors. Pid controller — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=PID_controller&oldid=841844294, 2018. [Online; accessed 18-May-2018].