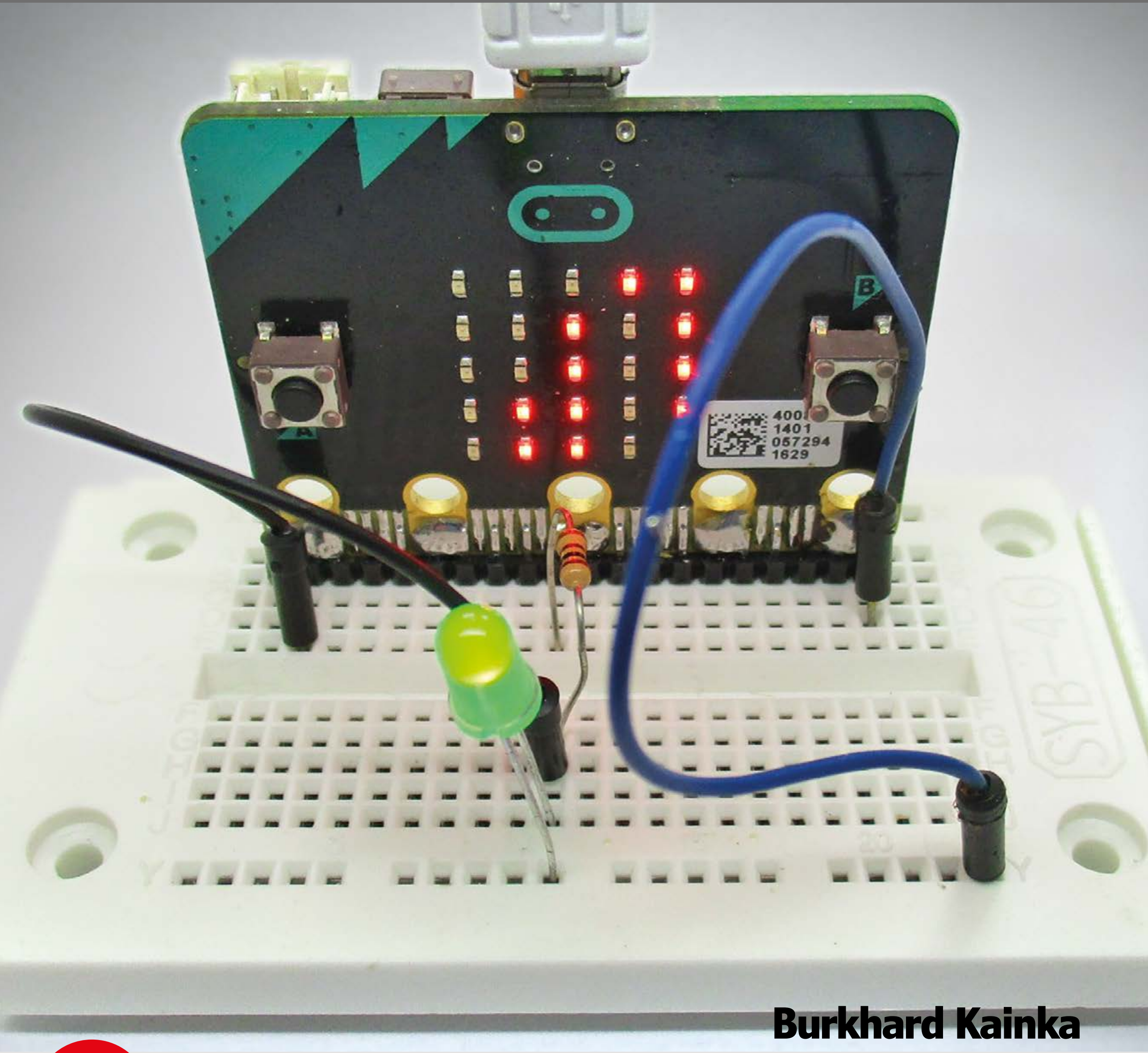
Analogue Electronics and Microcontrollers Projects

Basic Electronics for Beginners



Burkhard Kainka

elektor

LEARN DESIGN SHARE

Basic Electronics for Beginners

Analogue Circuits and Microcontroller Projects

with Burkhard Kainka

elektor LEARN DESIGN SHARE

● This is an Elektor Publication. Elektor is the media brand of
Elektor International Media B.V.
78 York Street, London W1H 1DP, UK
Phone: (+44) (0)20 7692 8344

● British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

● **ISBN 978-1-907920-84-4**

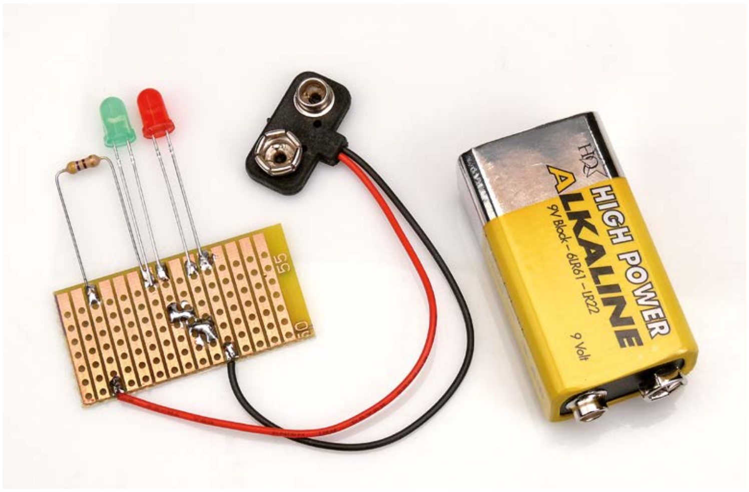
© Copyright 2020: Elektor International Media b.v.
Prepress Production: D-Vision, Julian van den Berg
First published in the United Kingdom 2020

Part 1 Analogue Electronics

Chapter 1 • Electronics for Starters (1)

Diodes and LEDs

Electronic devices are becoming more and more complex, with simple circuits and discrete transistors practically a thing of the past. This makes it increasingly difficult for beginners to get up to speed. In this series we therefore want to get back to basics, and in electronics the basics are analogue. However, we realise that many beginners are interested in digital technology, so a microcontroller circuit is also included in the course material.



Two LEDs with series resistor.

One way to approach a basics course would be to start with basic theory and concepts, including current, voltage and power, Ohm's law, parallel and series circuits and so on — in other words everything taught in 'Electronics 101', which you actually already know or should know. But that's deadly dull, so it's better to start with real circuits in small practical examples and projects.

You may be wondering what we aim to achieve with this course. Ideally it should help to bring new readers of Elektor up to the usual Elektor level. Some of our readers who follow this course may be the sons or daughters of long-term Elektor readers, who are looking for a chance to emulate their parents. It would certainly be helpful for experienced electronics enthusiasts and beginners to follow the course together. We have also established a forum for this course series at www.elektor.com/starters-forum. It would be nice if old hands could contribute something from their knowledge and experience in this forum.

There may also be a few Elektor readers who have already built lots of projects but never really understood exactly how they work. Of course, you can't expect miracles from a course such as this, but it should help clear away a few cobwebs.

The basics lie largely the realm of 'old-fashioned' analogue electronics, but the fact that so much of modern electronic is digital does not mean that the basics are no longer relevant.

Even for people who are interested in microcontrollers, there's no getting around analogue technology. This can be demonstrated using a few simple examples from the world of embedded applications. For example, microcontrollers are used to measure analogue quantities, among other things. This means that the course is unquestionably suitable for people who are getting their feet wet in the pool of pint-sized computers.

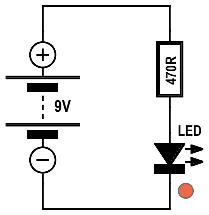


Figure 1. LED circuit.

LED with series resistor

Let's start off by putting together the circuit shown in **Figure 1**, with a LED, a resistor (470 Ω) and a battery. You can assemble it any way you wish — by simply soldering the components together on your desktop, using alligator clips or using a breadboard — although making a PCB specifically for this circuit would hardly be worth the effort. No matter how you do it, you're bound to get the LED to light up.

With LEDs you always have to pay attention to the polarity. The plus lead is the anode lead. The minus lead, which is normally the shorter lead, is the cathode lead. There is also a flat on the package next to the cathode lead. Inside the LED package you can see a throat-shaped mount for the LED chip, which is usually (but not always) fitted on the cathode side. The anode connection is provided by an extremely thin wire bonded to a contact on the top surface of the chip. If you connect the LED with reverse polarity, it won't light up. It shares this feature with every type of diode: current flows through a diode in only one direction.

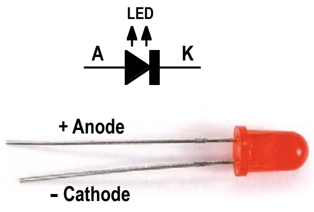


Figure 2. A LED.

LEDs should never be connected directly to a battery. If you look at a plot of the current through a LED versus the voltage on the LED, you can see why. Figure 3 shows the characteristic curves of various types of diodes. All of these curves have one thing in common: the current increases exponentially with increasing voltage. If the voltage is below what is called the forward voltage, virtually no current flows through the LED. However, if the voltage is just a bit higher than the forward voltage, the current quickly rises to a very high level and the LED may become overloaded. It's practically impossible to set the voltage to exactly the right value, in part because the curve shifts to the right with rising temperature at approximately 2 mV/K. Nevertheless, it's easy to set the current to a particular level, and all it takes is a single resistor. You simply have to choose the right value, and with this arrangement the right LED voltage is obtained automatically.

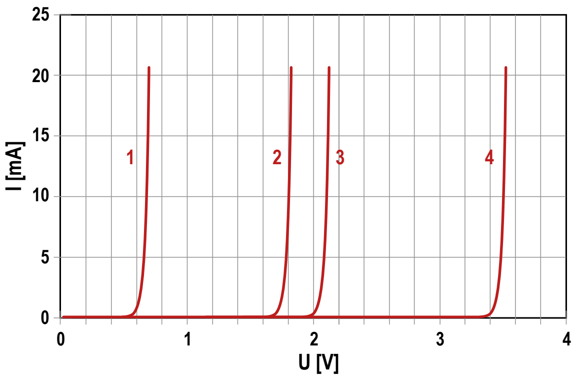


Figure 3. Characteristic curves of a silicon diode (1), a red LED (2), a green LED (3) and a white LED (4).

The forward voltages of several types of diodes at a typical current level of 20 mA are:

Silicon diode (e.g. 1N4148):	0.7 V
Red LED:	1.8 V
Green LED:	2.1 V
Blue or white LED:	3.5 V

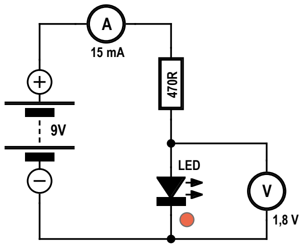


Figure 4. In-circuit measurements.

You can check this by measuring the voltages yourself (see **Figure 4**). The exact voltages may vary somewhat. For example, modern high-brightness red LEDs have a slightly higher forward voltage than older types of red LEDs.

Component dimensioning

If you have measured the diode voltage and you know the battery voltage, you don't need to measure the current. You can simply calculate it. This is because the voltage over the resistor is the difference between the battery voltage and the LED voltage (e.g. 9 V – 1.8 V = 7.2 V). With this information you can use Ohm's law to determine the current:

$$I = U / R$$
$$I = 7{,}2\ V / 470\ \Omega$$
$$I = 0{,}0153\ A = 15{,}3\ mA$$

If you instead want to calculate the value of the series resistor, you must specify the desired current value and know the values of the supply voltage and the LED voltage. For example, suppose you want to have a current of 20 mA flow through a green LED. For practical purposes, the voltage across the LED can be taken as 2.1 V. The battery voltage is 9 V, so the resistor has to produce a voltage drop of 6.9 V (9 V – 2.1 V). The calculation yields a value of 345 Ω, but this is not a standard resistor value. However, you may be able to find a 330 Ω resistor or a 390 Ω resistor in your parts box. It's a good idea to choose the higher value, since this puts you on the safe side with regard to the amount of current.

$$R = U / I$$
$$R = 6{,}9\ V / 0{,}02\ A$$
$$R = 345\ \Omega$$

You should also experiment with this circuit with various resistors having much higher resistance values. In each case, measure the LED voltage and determine the current. Generally speaking, no matter whether you operate the LED at a current of 1 mA, 5 mA or 10 mA, the voltage across the LED is nearly the same. This is due to the exponential shape of the characteristic curve.

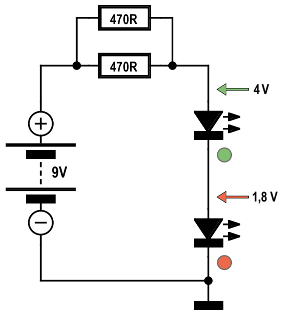


Figure 5. LEDs connected in series.

Series circuit

It's often useful to connect two or more LEDs in series with a common series resistor, as shown in **Figure 5**. In this situation the voltage across the series resistor is lower because the voltages across the LEDs add together. This means that the resistance must be reduced in order to obtain the rated 20 mA current thorough the LEDs. Suppose you are using a red LED with a forward voltage of 1.8 V and a green LED with a forward voltage of 2.2 V. This makes the voltage across the two series LEDs exactly 4 V, so the voltage across the series resistor is only 5 V. With a 470 Ω resistor you will have a current of approximately 10 mA. If you connect two such resistors in parallel, the current doubles. If you check the calculations, you should find that the current is 21 mA.

Semiconductors and depletion layers

The electrical conductivity of a typical semiconductor material, such as silicon, generally increases with rising temperature, but it is very low at room temperature. This is because all four outer electrons of the individual atoms are bound in the crystal lattice (**Figure 6**). However, they can be freed by the addition of a small amount of energy.

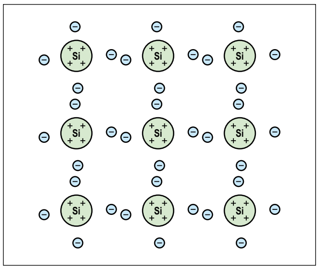


Figure 6. Crystal lattice of silicon.

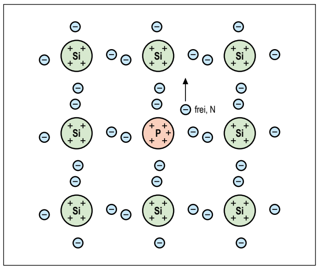


Figure 7. Silicon doped with phosphorus (n-type).

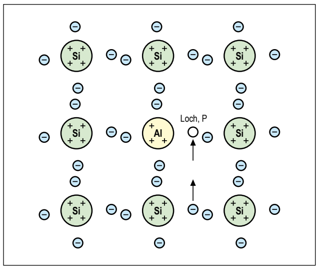


Figure 8. Silicon doped with aluminium (p-type).

Devices made from semiconductor materials, such as transistors and diodes, are commonly called semiconductors. They are made by intentionally adding foreign atoms to a material such as silicon (which is called doping the material) to obtain a defined conductivity. Doping with a Group V substance, such as phosphorous, produces free electrons and therefore n-type conductivity (Figure 7). Doping with a Group III substance produces electron holes, leading to p-type conductivity. These electron holes migrate through the crystal lattice, as though they were positive charge carriers, when the holes are filled by neighbouring electrons that leave behind new holes (**Figure 8**).

Diodes are semiconductor devices that conduct current in only one direction. They are usually made from layers of n-type and p-type silicon. A thin non-conductive depletion layer forms at the junction of these two layers. In the depletion layer free electrons fill holes in a process called recombination, with the result that practically no free charge carriers are left in the depletion layer, just as in pure silicon. In this state the diode does not conduct electricity (**Figure 9**).

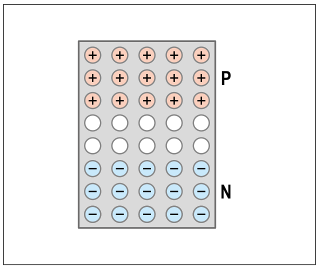


Figure 9. Layer structure of a diode.

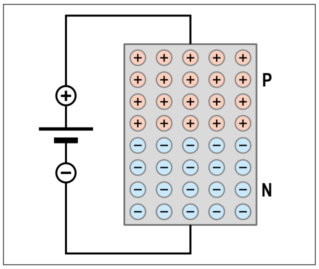


Figure 10. A diode in the forward conduction state.

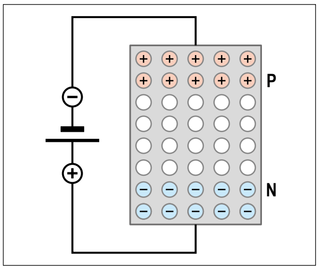


Figure 11. Depletion layer widening under reverse bias.

If a low voltage is applied to the external leads of a diode, the depletion layer becomes thinner or thicker. If the n lead is connected to the minus terminal of a battery and the p lead is connected to the positive terminal, the charges at the lead connections repel their corresponding charge carriers toward the depletion layer. Above a voltage of approximately 0.5 V, the n and p layers start to touch each other and a current starts to flow (Figure 10). Good conductivity is achieved at a voltage of approximately 0.7 V. In this state the diode is operating in the forward direction.

If the polarity of the applied voltage is reversed, the opposite effect occurs. The charge carriers are attracted to the outer connections and the depletion layer widens. This makes the depletion layer an even better insulator (**Figure 11**). A typical diode such as a 1N4148 can handle reverse voltages up to 75 V. A diode effectively allows current to pass in only one direction, so it can be used as a rectifier.

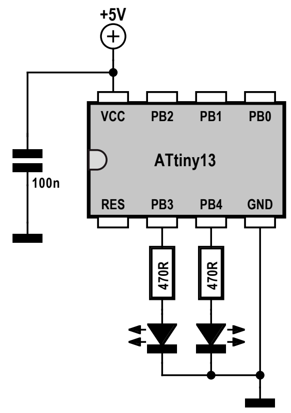


Figure 12. A microcontroller with two LEDs.

In most cases the reverse voltage should not exceed the value recommended by the manufacturer. A reverse current flows if the applied voltage is too high. This results from what is called breakdown (of the insulating layer). With some special types of diodes, such as Zener diodes, this effect is used intentionally. Zener diodes have well defined breakdown voltages and are used as voltage regulators. If you abuse a silicon diode such as a 1N4148 by applying an excessive reverse voltage to it, you will cause what is called second breakdown, which is fatal. This is because the excessive reverse current heats the junction to the point of destruction. This causes the formation of a permanent, non-repairable short circuit. LEDs, as the name indicates, are diodes and also have p-n junctions. They are made from semiconductor materials such as gallium arsenide. LEDs have higher forward voltages than silicon diodes, and electron–hole recombination in LEDs produces visible light. This effect also occurs in silicon diodes, but they produce only minute amounts of light in the infrared region.

Blinking LED

LEDs are often driven by microcontrollers. Here again series resistors are necessary. The circuit depicted in Figure 12 has two LEDs, each connected to an I/O port pin of the ATtiny13 microcontroller by a 470 Ω series resistor. The associated simple BASCOM program sets PB3 constantly high and PB4 alternately high and low, causing the connected LED to blink. Try measuring the voltage on the PB3 lead. It will be a bit less than 5 V (for example 4.9 V) because the switch transistor in the microcontroller also has a small resistance. You can determine the internal resistance of the port pin from the voltage drop. The next question is how much current is flowing through the LED. You can calculate this easily. Have a look at the ATtiny13 data sheet (www.atmel.com/dyn/resources/prod_documents/doc2535.pdf) to see how much current is allowed to be drawn from an I/O port.

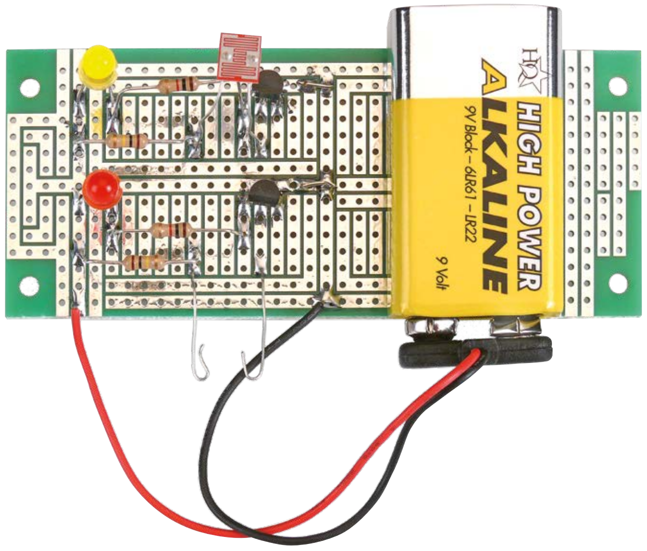
```
,ATtiny13 driving LEDs
$regfile = "attiny13.dat"
$crystal = 1200000
Config Portb = Output

Do
  Portb.3 = 1
  Toggle Portb.4
  Waitms 500
Loop

End
```

Listing 1.

Chapter 2 • Electronics for Starters (2)

Transistors in action

Electronic devices are becoming more and more complex, which makes it increasingly difficult for beginners to get up to speed. In this series we therefore aim to get back to basics. In this instalment we present some interesting experiments with transistors. We also have a quiz for you, with the chance of winning a nice prize.


Two experiments on a Elektor Elex board.

Transistors can easily be regarded as one of the most significant technological inventions ever. Many aspects of modern everyday live — including computers, mobile phones and the Internet — would be impossible without them. In the 1950s these small semiconductor components started displacing vacuum valves, which had played a dominant role up to then. Germanium transistors were the first to become popular, followed later by bipolar silicon transistors and even later by field-effect transistors. Technological progress in this area was accelerated by the invention of integrated circuits (ICs), which contain a large number of transistors in a single package. However, you can implement a wide variety of functions with a single discrete transistor, as we demonstrate in this instalment.

First experiments

Start by building the circuit shown in Figure 1, for example on an Elektor Elex board (see elektor.com/120002). This allows you to use the same board for several experiments and utilise the through tracks for power and ground rails. A 9-V battery provides a convenient source of power. It doesn't need to be fully charged – for example, a battery retired from service in a smoke detector will do nicely. A weak battery actually has the advantage that if something goes wrong, it can't supply enough current to cause anything to go up in smoke.

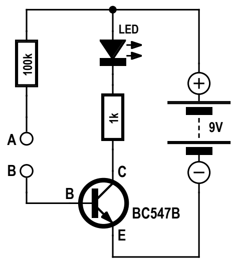


Figure 1. Our first experimental setup.

Now let's try a set of simple experiments:

1. When contacts A and B are not connected, the LED should remain dark.
2. Connect A and B together. The LED should light up brightly.
3. Bridge A and B with a wet finger. The LED should light up more or less dimly.
4. Leave A and B open, and see what happens when you short the emitter (E) and collector (C) leads of the transistor together. The LED should light up brightly.
5. Connect A and B again (the LED should be lit), and then short the base lead (B) to ground. The LED should go dark.

This set of experiments illustrates the basic operating principle of a transistor: a small base current (between the base and the emitter) controls a larger collector current (between the collector and the emitter). We say that the base current is amplified, and roughly speaking, we can regard the amplification factor (or gain) as constant. The widely used BC547B transistor has a gain of approximately 300, which means that the collector current is a factor of 300 greater than the base current (Figure 2). However, this is only true if it is not limited to a smaller value by a collector resistor (as in the circuit shown in **Figure 1**).

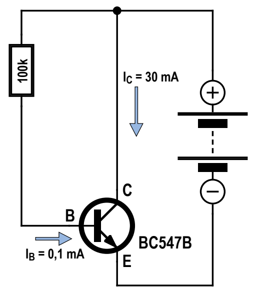


Figure 2. Basic current gain circuit.

Circuit design

In order to design a transistor circuit, you first need to know exactly what you want to achieve.

a) Should the transistor operate as a switch and be either fully off (cut off) or fully on (conducting)?

b) Or should the transistor operate as an analogue gain stage and allow more or less current to flow?

You have already tried both options in the initial set of experiments. When contacts A and B are joined together, the transistor is driven fully into conduction (switched on), although it has more internal resistance in this state than a real switch with two metallic contacts. As a result, there is always a small voltage drop between the emitter and the collector. With the wet finger experiment you were in the analogue camp, and you may have noticed that the brightness of the LED depends on how hard you press your finger against the contacts. The choice of liquid also plays a role here – for example, cola yields more current than tea, due to the acids in the cola.

One of the difficulties in designing transistor circuits is that you do not know the exact gain of the transistor. Unlike resistors, which are readily available with a tolerance of 1%, it is very difficult to manufacture transistors to tight tolerances. The gain in particular shows a considerable range of variation. In the case of the BC547, the gains of individual devices in a new fabrication batch can lie anywhere between 110 to 800. These new devices are measured by automated equipment and sorted into the three gain groups A, B and C (see the 'TUP/TUN' inset). The range of gains in these three groups is still fairly large, which is simply a fact of life for circuit designers. They must design their circuits to work properly with every transistor in the selected group. This sometimes requires a bit of calculation; in many cases just trying it out is not enough.

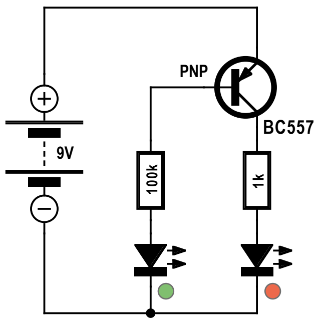


Figure 3. A PNP transistor in a common-emitter circuit.

Now let's have a look at the circuit shown in **Figure 3**. A PNP transistor operates in the same way as an NPN transistor, but it has the opposite polarity. This means that the emitter is connected to the positive terminal of the battery. This circuit has an additional LED in the base circuit. It is intended to show that the base current is much lower than the collector current, which is why the light from the green LED is very dim.

Inverter

From high to low, from on to off: inverters perform a very simple task in the world of computers and microcontrollers. However, a transistor can do this just as well. Up to now we have been using our transistor as a sort of controlled switch: if you switch on the base current, the transistor switches on the load current. But you can also reverse (invert) the switching function with a transistor. **Figure 4** shows a simple inverter circuit. Here the LED lights up when the switch is closed and goes dark when the switch is open. The reason for this is that when the switch is closed, the base circuit is closed through the LED and a current flows into the base. This causes the transistor to conduct, and it shorts out the voltage over the red LED. If you measure the voltage between the collector and the emitter, you will find that it is around 100 mV. At this low voltage the current through the LED is virtually nil, so it remains dark.

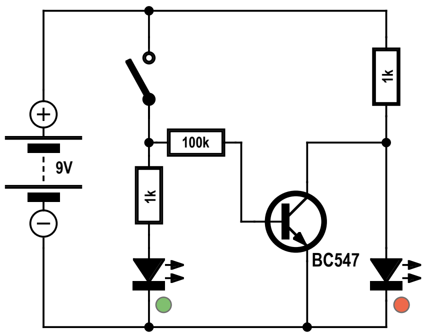


Figure 4. A transistor configured as an inverter.

Delayed switch-off circuit

The current gain of a transistor can be used to extend the discharge time of a capacitor. The circuit shown in **Figure 5** has a 100 µF electrolytic capacitor serving as a storage capacitor. It charges quickly when you press the pushbutton, and after the button is released it supplies a base current to the transistor. The high resistance of the base resistor results in a time constant of around 10 seconds. After this interval the base current is no longer strong enough to drive the transistor into full conduction.

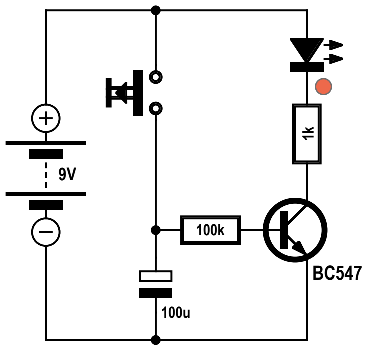


Figure 5. Delayed switch-off.

The time constant of an RC network is the time required for the capacitor to discharge to the point where its voltage is a factor of 1/e (1/2.718…) of the initial voltage (36.8%).

The time constant can be calculated using a simple formula:

Time constant = resistance × capacitance
t = R × C
t = 100 kΩ × 100 µF
t = 10 s

As it happens, you can still detect a faint light after one minute. The LED actually continues to emit light for a relatively long time, but the current drops to such a low level that the light is no longer visible.

If you prefer to implement a time switch with a microcontroller, see the 'Microcontroller time switch' inset.

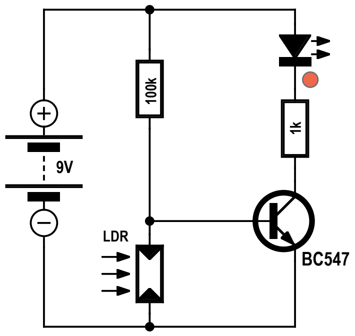


Figure 6. A twilight switch.

Twilight switch
In the circuit shown in Figure 6 we use a light dependent resistor (LDR) as a light sensor. This component has a light-sensitive resistive layer made from cadmium sulphide (CdS). Its resistance depends on the intensity of the incident light, ranging from approximately 100 Ω in full sunlight to over 1 MΩ in the dark. The resistance at an illumination level of around 1000 lux (equivalent to a well illuminated workplace) is approximately 1 kΩ.

The combination of the variable resistance of the LDR and the fixed resistance of the 100 kΩ resistor forms a voltage divider. The transistor is cut off when the voltage between the base and the emitter (UBE), which is taken from the voltage divider junction, is too low. In simplified terms, we can say that this circuit has a switching threshold of approximately 0.6 V. This value applies to all silicon transistors and results from the well known diode characteristic curve.

Try out this circuit with various light levels to see how it behaves. The LED is switched off when the light level at the sensor is high and switched on when the light level is low. You should see fairly abrupt switching at a certain threshold light level. The range of light levels for which the transistor is in the partially conducting state is small.

Darlington circuit

The gains of a pair of transistors can be multiplied by using the amplified current from the first transistor as the base current for the second transistor, where it is further amplified (see Figure 7). If each of these transistors has a gain of 300, the Darlington pair has a gain of 90,000. This circuit can be driven into full conduction with a base lead resistance of 10 MΩ, so it can be used effectively as a touch switch with two bare wires touched by two fingers. Moistening your fingers is no longer necessary; even dry skin allows enough current to flow to drive the circuit fully on. The additional 100 kΩ resistor protects the transistors against excessive base current, which would otherwise flow if the two wires were shorted together.

An extension of the Darlington circuit to three transistors (**Figure 8**) can be used for interesting experiments with static charge detection. To see this, try sliding you feet on the floor while touching the base lead of this Darlington circuit with one finger. Depending on the nature of the floor and the material of your shoe soles, this will produce more or less strong charge displacements that are made visible by flickering of the LED. In many cases simply approaching the input terminal without actually touching it is enough to cause the LED to light up.

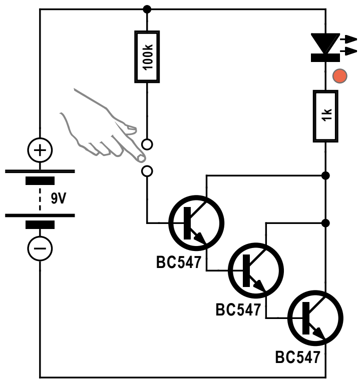


Figure 8. A triple Darlington.

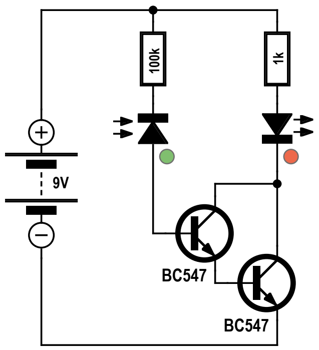


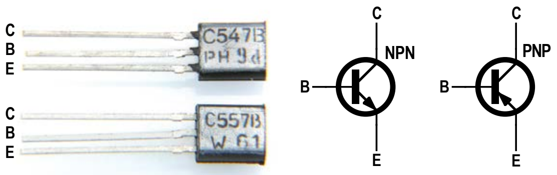
Figure 9. Amplifying the reverse current of an LED.

Using a LED as a photodiode

In addition to emitting light, LEDs can be used as sensors for ambient light. In principle no current flows through a diode when it is reverse biased, but in fact you can measure a very small reverse current in the range of a few nanoamperes, which is low enough to be ignored in most cases. However, the high gain of the Darlington circuit allows you to perform experiments with extremely low currents such as this. For instance, the reverse current of an LED depends on the light level, which means that an LED acts as a sort of photodiode. We can use our Darlington circuit to amplify the extremely small reverse current to the level needed to light up the second LED. In such experiments you should bear in mind that the rated reverse voltage of an LED is much less than that of a normal diode. The maximum reverse voltage of LEDs is usually specified as 5 V on the data sheets, but the voltage on the LED in our circuit is approximately 8 V. In fact most red, yellow and green LEDs can withstand significantly higher reverse voltages before entering the breakdown region, although the reverse breakdown voltages of white and blue LEDs are very low. In any case, the 100 kΩ resistor protects the LED against serious damage.

Glory days of TUP and TUN

There are so many different types of transistors that it can be difficult to decide which one to use. In the distant past Elektor used the designations 'TUP' (transistor universal PNP) and 'TUN' (transistor universal NPN), but in those days it was possible to buy unmarked transistors a bit cheaper than marked ones, and 'TUN' simply meant any type of general-purpose small-signal NPN transistor. Nowadays you are well advised to use the BC547B; it almost always fits and is a sort of modern TUN. You should actually have a bag of them on hand, and it won't make a big dent in your budget. For the TUP the natural choice is the BC557B.



NPN- und PNP-Transistor.

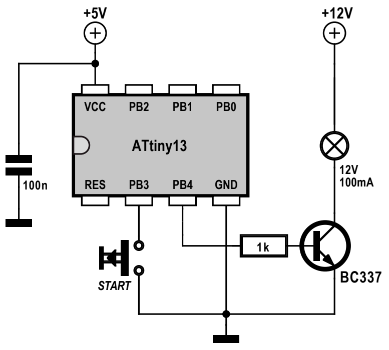
The key BC547B specs are:

- Maximum collector voltage: 45 V
- Maximum collector current: 100 mA
- Current gain: 200 to 450 (290 typical)

The BC547A has a current gain of 110 to 220 (180 typical), and the BC547C has a current gain of 420 to 800 (520 typical). If you examine the current gain curves in more detail, you will see that the current gain of a transistor is fairly constant only at moderate collector currents; it drops significantly at relatively high and low current levels.

Microcontroller time switch
Modern time switches are built around microcontrollers. This allows them to achieve high precision without calibration. RC timing circuits have evidently had their day, but there's one thing a microcontroller cannot do: switch high currents. For this you need a transistor. A simple NPN transistor makes a suitable power driver for switching external loads. It gives the relatively lightweight microcontroller port more muscle. A popular choice for this task is the BC337, which can switch up to 800 mA. The figure shows a time clock circuit where the current that must be switched by the microcontroller is less than 5 mA. The transistor amplifies the port current enough to switch an incandescent lamp. It also provides level shifting, since the microcontroller operates at 5 V and the lamp operates at 12 V.

The small BASCOM example program implements a time switch. The timeout (1 minute) starts counting down after the button is pressed. Unlike the analogue circuit in **Figure 5** of the main text, pressing the button again during the timeout interval does not prolong the timeout. How should the code be modified to enable retriggering?



A time clock circuit.

```
,Timer 60 s
$regfile = „attiny13.dat“
$crystal = 1200000
Config Portb.4 = Output
Portb.3 = 1               ,Pullup

Do
  Do
  Loop Until Pinb.3 = 0
  Portb.4 = 1
  Waitms 60000
  Portb.4 = 0
Loop

End
```

Listing 1.

Chapter 3 • Electronics for Starters (3)

Transistor measurements

Manufacturers' data sheets for transistors typically include sets of characteristic curves that show how the transistors behave in different situations. However, it's even better to use your own instruments to measure as much as possible yourself. This gives you a feeling for the component and improves your understanding of how transistors work in practice — not just in theory.

You should measure the base current IB, the collector current IC, the base–emitter voltage UBE and the collector–emitter voltage UCE. If you wish to make all these measurements with a single multimeter, it's advisable to measure only voltages and to use the same measuring range as much as possible. The currents can be calculated relatively easily from the voltages and the resistances in the circuit.

Figure 1 shows our measurement set-up. The potentiometer connected to the circuit input allows the input voltage to be adjusted in small increments over the range from zero to 5 V. For each setting, you should measure and record the four voltages U1–U4. From this you can derive the currents and the amplification factor A. Table 1 shows an example of values measured with a BC547B transistor using separate, fixed multimeters for U1, U2 and U3. U4 was calculated from U3, and the currents and amplification factor were derived from the measured voltages.

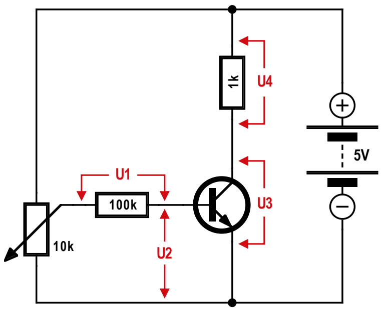


Figure 1. Measurement setup.

Practical tips

What procedure should you follow for making the measurements? A good approach is to first adjust the collector current to 0.1 mA (U4 = 0.1 V) and then double it repeatedly for each new measurement until it stops rising (set the value of U4 successively to 0.1 V, 0.2 V, 0.4 V, 0.8 V, and so on). This reveals an interesting aspect of typical transistor behaviour: each time you double the collector current, the base current is approximately doubled as well, but the base–emitter voltage increases each time by a constant amount of around 20 mV.

Table 1. Measured values for a BC547B transistor

	U1	I_B	$U2=U_{BE}$	$U3=U_{CE}$	U4	I_C	$V = I_C/I_B$
1	0 V	0 µA	0 mV	5 V	0 V	0 mA	0
2	0 V	0 µA	400 mV	5 V	0 V	0 mA	0
3	0,07 V	0,7 µA	573 mV	4,9 V	0,1 V	0,1 mA	143
4	0,15 V	1,5 µA	595 mV	4,8 V	0,2 V	0,2 mA	133
5	0,26 V	2,6 µA	612 mV	4,6 V	0,4 V	0,4 mA	153
6	0,47 V	4,7 µA	629 mV	4,2 V	0,8 V	0,8 mA	170
7	0,90 V	9,0 µA	646 mV	3,4 V	1,6 V	1,6 mA	177
8	1,77 V	17,7 µA	665 mV	1,8 V	3,2 V	3,2 mA	181
9	2,63 V	26,3 µA	679 mV	0,3 V	4,7 V	4,7 mA	179
10	3,54 V	35,4 µA	681 mV	0,15 V	4,85 V	4,85 mA	137
11	4,32 V	43,2 µA	683 mV	0,13 V	4,87 V	4,87 mA	113

You can calculate the current gain by dividing the collector current IC by the base current IB. As you can see from the table, the maximum amplification factor (current gain) was around 180 with our test transistor. This is a bit on the low side, since this device should have a gain of at least 200. However, there are several potential error sources, such as the finite internal resistance of the meter (10 MΩ). Among other things, this means that a small portion of the intended base current flows through the meter instead of the transistor when U2 is measured. Measurement errors are normal. If you take all the error sources and tolerances into account (including resistor tolerances), the gain of this transistor could actually lie right at the lower limit of 200. Try this for yourself — maybe your transistor can do better.

With such a large amount of measurement data, it's a good idea to analyse it in graphic form. You can do this with pencil and paper, or with a worksheet on your PC. This yields the following results:

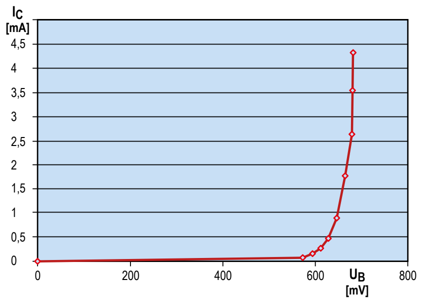


Figure 2. Base current versus base–emitter voltage.

Figure 2 (IC versus UB) shows the typical exponential curve of a silicon diode. On the linear scale you see a breakpoint at approximately 0.6 V. Above this level the slope of the current curve keeps rising (exponential curve). The measurement data shows that no measurable base current flows at a base voltage of (for example) 400 V, so there is also no collector current. Accordingly, you can note that the base voltage is usually somewhere between 0.6 V and 0.7 V.

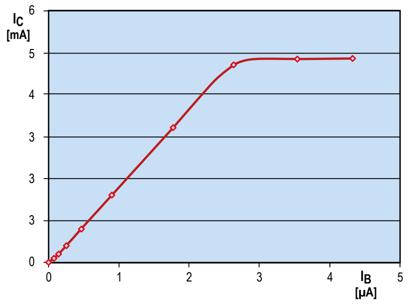


Figure 3. Collector current as a function of base current.

Figure 3 (IC versus IB) shows that the collector current increases linearly with the base current (in the first approximation) until it stops rising at just under 5 mA, which is called the saturation point of the transistor. In any case it isn't possible to measure more than 5 mA with this circuit because the collector resistor limits the current to 5 mA (5 V ÷ 1 kΩ = 5 mA). You can also clearly see that even this 5 mA limit cannot be reached. The transistor is driven fully into conduction ('hard on'), with a residual collector–emitter voltage of slightly less than 0.1 V.

This curve also has a lower slope (lower current gain) at very low current levels. This is actually true, because the gain decreases slightly at very low and very high collector current levels, but it also reflects a measurement error that amplifies this effect. A small current flows through the meter that measures U2, and at low current levels this makes the base current appear to be greater than it actually is.

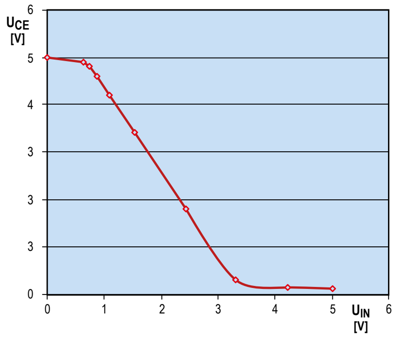


Figure 4. Output voltage as a function of input voltage.

Finally, **Figure 4** shows the output voltage (UCE) as a function of the input voltage (U1 + U2) on the wiper of the potentiometer. Here you can see right away that increasing the input voltage causes the output voltage to drop. The reason for this is simple: when the collector current rises, the voltage drop over the collector resistor increases.

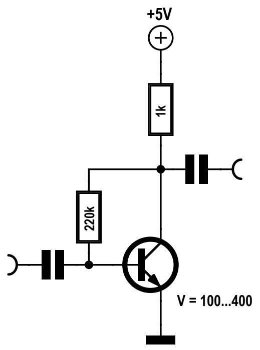


Figure 5. Using negative feedback to set the operating point.

Negative feedback

If you are designing a transistor circuit and you do not know the transistor gain precisely, you must adapt to this situation. Things are easy with a switching stage: you only need to dimension the base current such that the circuit will work properly even with the lowest possible current gain. This means that in case of doubt you should use a bit more base current, to ensure that there's enough for every transistor of the chosen type.

Things are different when you need to amplify an analogue signal. Too much base current may be exactly the wrong thing in this situation, due to the risk of saturating the transistor. As much as possible you should aim for a mid-range collector current that can be adjusted upward or downward. One way to achieve this with a variety of transistors is to use negative feedback. This can be done by connecting the base resistor to the collector instead of the supply voltage (see **Figure 5**). A transistor with especially high gain tends to produce more voltage drop across the collector resistor, which reduces the collector voltage and with it the base current. At the other end of the scale, a transistor with low current gain automatically operates with more base current. The end result is that the circuit is suitable for a wide variety of transistors.

Making measurements with an ohmmeter

Pointer-type analogue meters still have certain advantages for component testing and troubleshooting. They allow results to be read faster than with digital multimeters, at least roughly. However, digital multimeters are indispensable when you need high precision.

Most simple analogue multimeters have one or more resistance ranges, and with a bit of practice you can use a simple ohmmeter to check not only resistors, but also transistors, diodes, capacitors and many other types of components. A multimeter needs a battery to measure resistance, although this battery is often not used for any other purpose. Resistance is determined by measuring the amount of current that flows with a constant supply voltage. As a result, the resistance scale is not linear. The full-scale indication at zero ohms must be adjusted using a potentiometer in order to compensate for variations in the battery voltage (see **Figure 6**). The other end of the scale corresponds to infinite resistance, regardless of the range.

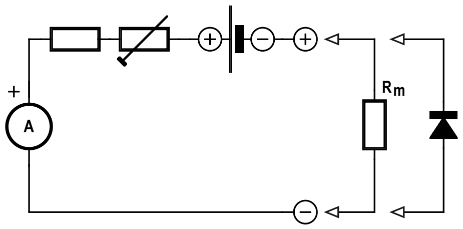


Figure 6. Basic circuit of an analogue ohmmeter.

A consequence of the usual circuit configuration used in simple analogue multimeters is that the polarity of the voltage on the terminals for the resistance ranges is opposite what you would expect from the markings for the current and voltage ranges. Consequently when a resistance range is selected, the negative terminal of the multimeter is positive and the positive terminal is negative. This must be taken into account if you want to use a simple ohmmeter to check diodes or transistors.

When measuring diode junctions, you should bear in mind that it's impossible to specify a fixed resistance value for a semiconductor junction. The indicated value is strongly dependent on the measurement current, and therefore on the selected measuring range.

Nevertheless, it is possible to draw meaningful conclusions. If the pointer deflection is around half of full scale with an internal voltage of 1.5 V, the voltage drop over the object being measured must be approximately 0.75 V. Due to the exponential shape of the diode characteristic curve, the pointer deflection changes only slightly when you switch to a different range. Although a different resistance is indicated in each measuring range, the pointer deflection is nearly the same because the voltage drop is always approximately 0.6 V. The forward voltage of the diode can also be used to judge the diode type. For example, the diode in **Figure 7** is most likely a silicon diode.

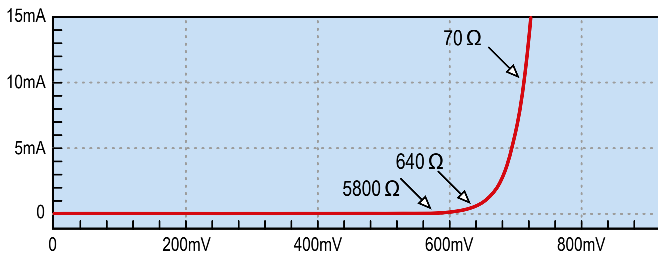


Figure 7. DC resistance of a silicon diode at various measurement currents.

Transistor testing
sing just a simple ohmmeter for transistor testing, you can draw several conclusions about the transistor type and its general condition. Even with a fully unknown transistor type, you can at least determine which lead is which.

A transistor can be fully tested with the aid of three typical measurements. First you measure the base–emitter junction and the base–collector junction (see the 'Basic transistor operation' inset), which allows you to distinguish between silicon and germanium transistors and identify any short-circuits (**Figure 8**, parts 1 & 2). After this you measure the resistance between the emitter and the collector with and without base current (3). A transistor in good working order will show infinite resistance (no current) with the base open. With the base connected to the collector, the indicated current should be somewhat greater than the current through the base–emitter junction alone.

The final test (4) should be made with the a small base current flowing through a resistor connected between the collector and the base. The base current can be provided by touching the collector and base leads with a wet finger. The resulting deflection of the ohmmeter pointer gives a rough indication of the current gain of the transistor. A small current gain can be seen even with the emitter and collector leads reversed, so in case of doubt you should try reversing the transistor leads if the lead assignments are not known reliably.

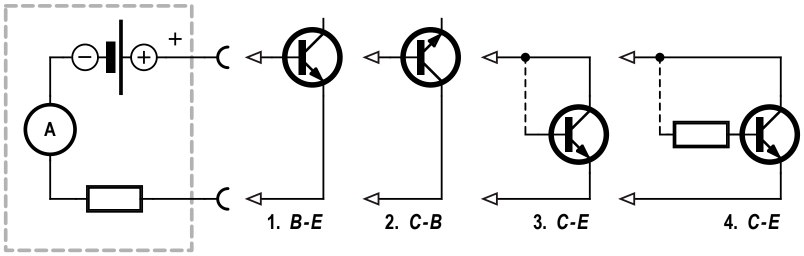


Figure 8. Transistor measurements.

Most digital voltmeters use an entirely different internal circuit configuration for resistance ranges. In this case the resistance measurement is based on measuring the voltage drop with a constant current. This results in a linear scale and a clearly defined measuring range, and it eliminates the need for adjusting the zero point. Another difference with respect to pointer instruments is that the voltage, current and resistance ranges have the same terminal polarity.

In theory, the resistance range of a digital multimeter can be used to make the same component tests as with an analogue meter. Many such meters also have a separate measuring range for diode junctions. Although this is based on the same measuring principle as the resistance range, it displays the voltage drop in millivolts or a reading that is proportional to the forward voltage.

Basic transistor operation made from p-doped and n-doped semiconductor materials, but they have three regions separated by junctions. These regions may be arranged as N-P-N or as P-N-P.

First let's consider the NPN transistor, whose structure and equivalent circuit diagram are shown in **Figure 9**. The individual regions of the transistor are called the emitter (E), the base (B) and the collector (C). For proper operation the base region must be very thin. First consider the situation with the transistor connected to a voltage source, with the emitter connected to the negative terminal and the base lead unconnected (**Figure 10**). No current flows in this situation because the base–collector junction is reverse biased.

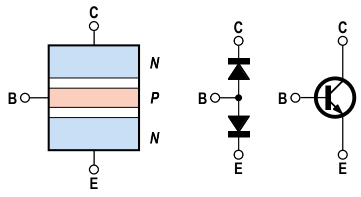


Figure 9. NPN transistor.

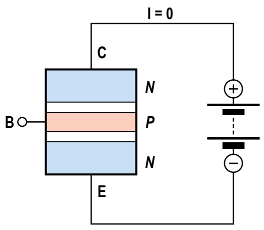


Figure 10. No current flows in transistor.

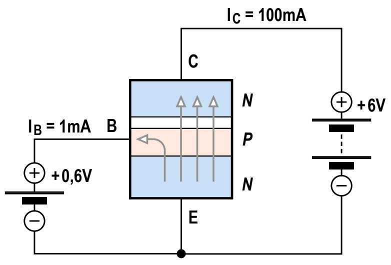


Figure 11. Current flows in transistor

Now suppose a second voltage source is connected between the base and the emitter, with the positive terminal connected to the base and the voltage so low (approximately 0.6 V) that only a small current flows through the base–emitter junction. Here you will see a much larger current flowing between the emitter and the collector. This results from the fact that the base region is very thin. Negative charge carriers that enter the base region are exposed to a strong electric field across the collector–base junction, and most of them are drawn into the collector region. Only around 1 per cent of the charge carriers originating from the emitter reach the base lead (**Figure 11**). Put differently, this means that the collector current is around 100 times greater than the base current. The collector current is controlled by the base–emitter voltage or the base current.

Although the electrons travel from the emitter to the collector, due to the convention that current flows from the positive terminal to the negative terminal, here we say that the current flows from the collector to the emitter.

Transistor tester
A microcontroller with an internal A/D (analogue-to-digital) converter can be put to good use as a measuring instrument, such as (how did you guess?) a transistor tester. Here all we're interested in is measuring the current gain. An ATtiny13 microcontroller is sufficient for this task if the resulting measurement data is sent in serial form to a PC and displayed on the PC monitor using a terminal emulator program.

The circuit for this is very simple (see **Figure 12**). Only the collector voltage is measured. The transistor is operated with negative feedback, which allows a wide range of gain values to be measured. This requires a bit more computation from the program, but that's what microcontrollers are for.

```
,Transistor tester
$regfile = „attiny13.dat“
$crystal = 1200000
$hwstack = 8
$swstack = 4             ' 16
$framesize = 4


Dim UC As Word
Dim U1 As Word
Dim U2 As Word
Dim I1 As Word
Dim I2 As Word
Dim V As Word

Config Adc = Single , Prescaler = Auto
Start Adc

Open „comb.1:9600,8,n,1,INVERTED“ For Output As #1

Do
  UC = Getadc(3)         ' PB3=ADC3 -> UC = 0..1023

  UC = UC * 50           ' max 51150 -> 5115 mV
  U2 = UC - 6000         ' 6000 <- U_BE = 600 mV
  U1 = 51150 - UC
  I1 = U1                ' 1 k
  I2 = U2 / 100          ' 100 k
```

```
     V = I1 / I2
     Print #1 , V            ' --> RXD
     Waitms 1000
   Loop

   End
```

Listing 1.

The program calculates the voltage drop U1 over the collector resistor and the voltage drop U2 over the base resistor. From these values it derives the collector current I1 and the base current I2. The current gain A is then I1 ÷ I2.

To make all of this possible with a small microcontroller, the program is uses only integer values of type word. We also took precautions to avoid numerical overflow and to avoid reduced accuracy due to small intermediate results.

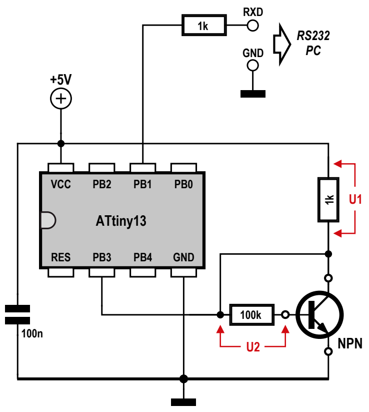


Figure 12. NPN test.

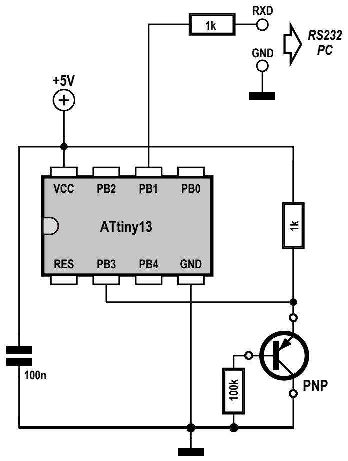


Figure 13. PNP test.

The microcontroller can also make measurements on PNP transistors with the same program. This only requires a slightly different connection scheme, as shown in Figure 13.

Chapter 4 • Electronics for Starters (4)

Constant current sources

Designing electronic circuits usually starts with studying the data sheets of complex integrated circuits. However, you shouldn't loose sight of the fact that ICs incorporate a variety of basic circuits, which in many cases you can also build with discrete transistors. Besides being educational, that's a lot of fun!

In the previous instalments of this course we have already dealt with the current gain, input characteristics and saturation behaviour of bipolar transistors. Now we expand our field of view to include field-effect transistors. The various properties of transistors can be utilised in a wide variety of circuits. In each case the interesting question is how you should design the circuit so it will reliably do what it is suppose to do. Of course, you also need to know the limits of what the circuit can handle.

A constant current source

Sometimes you need a constant current with the least possible dependence on supply voltage stability. For example, a LED driven by a constant current source will always have the same brightness, even when the battery voltage drops. In theory all you need for this is a single transistor in a common-emitter circuit. If you drive the transistor with a constant base current, the collector current will be nearly constant regardless of the voltage between the collector and the emitter. This can be seen from the output characteristics of a typical NPN transistor (**Figure 1**).

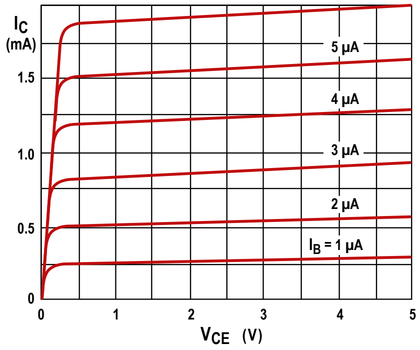


Figure 1. BC547B output characteristics.

Figure 2 shows a constant current source built around a single transistor. It can be used with a wide range of supply voltages and with different numbers of LEDs wired in series. The collector current is nearly constant. In technical terms, we say that the constant current source has a large differential internal resistance Ri, where Ri = dU/dI. In this circuit the constant base current is provided by a separate voltage source.

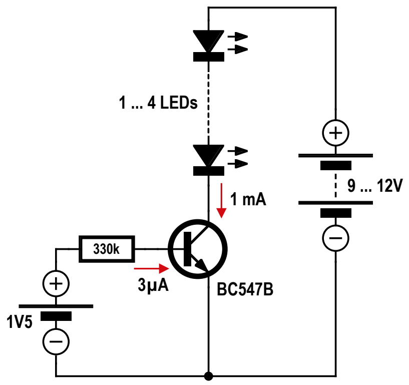


Figure 2. Constant current.

You could also build a voltage regulator that generates a stable voltage (e.g. 3 V) from an unstable supply voltage. However, you can't easily obtain a precisely defined current with this circuit. In particular, the tolerance range of the current gain makes it impossible to predict the exact collector current.

Using a BF245 JFET

A similarly imprecise constant current source can be built with a junction field-effect transistor (JFET), such as the type BF245. As explained in the inset, with this type of transistor you apply a negative voltage to the gate to obtain the desired current. **Figure 3** shows the output characteristics of the BF245B with various values of gate-source voltage VGS. The drain current ID is fairly constant as long as the drain-source voltage VDS is not too low. The range of variation of the BF245's characteristics is similar to that of the BC547, which is why the BF245 also has three gain classes (A, B and C). The BF245B has a specified drain current of approximately 10 mA with zero gate voltage.

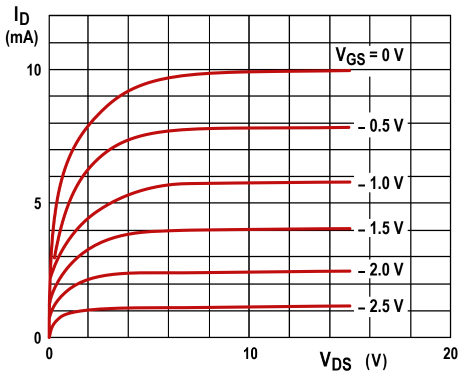


Figure 3. BF245B output characteristics (source: Philips).

The simple 10 mA current source shown in **Figure 4** is good enough for practical use, as long as you can live with an actual current somewhere between 8 mA and 12 mA. In any case, it's hard to beat this circuit for simplicity, although it should be noted that the current is somewhat dependent on the drain-source voltage because the internal resistance is not especially high at low drain-source voltages.

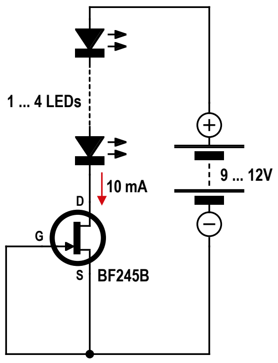


Figure 4. A simple JFET constant current source.

What you need here is some sort of closed-loop control that keeps the current constant. A simple way to implement this is to add a source resistor, as shown in **Figure 5**. This arrangement is often used to generate the gate bias voltage 'automatically'. It also improves the stability of the output current by increasing the internal resistance. This works as follows: if the output current increases, the voltage drop over the source resistor also increases, causing the gate voltage to be more negative relative to the source voltage. This acts to reduce the drain current. This circuit effectively generates a simple form of negative feedback. You can also set the value of the current within wide limits by selecting the value of the resistor. If you want to have a bit more than 1 mA, simply use a resistor with a lower value.

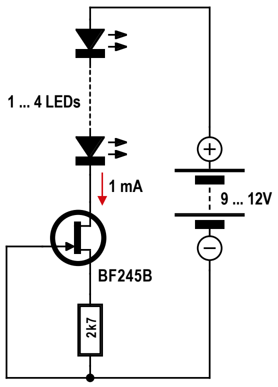


Figure 5. Using a source resistor to set the current.

Using a bipolar transistor

The circuit shown in **Figure 6** is a simple constant current source built around an NPN transistor, which converts a constant voltage into a constant current. A Zener diode at the input stabilises the base voltage at approximately 2.7 V, due to the effect of its steep characteristic curve. As the base-emitter voltage is always approximately 0.6 V, the voltage over the emitter resistor is approximately 2.1 V. This resistor therefore determines the emitter current. The collector current is only slightly less than the emitter current, which also includes the much smaller base current. With negative feedback provided by the emitter resistor, this circuit is almost directly equivalent to the FET circuit shown in **Figure 5**. The only difference is that here you need a positive voltage source for the base, which makes the component count a bit higher. However, the good news is that the BC547 is cheaper and it provides better regulation in this circuit. The negative feedback is so effective with this circuit that you can scarcely measure any difference if you first fit a BC547A, then a BC547B and finally a BC547C. In other words, you can use whatever version you happen to have in your parts tray. Another tip: if you don't have a suitable Zener diode handy, you can replace it with a forward-biased LED and still obtain good results.

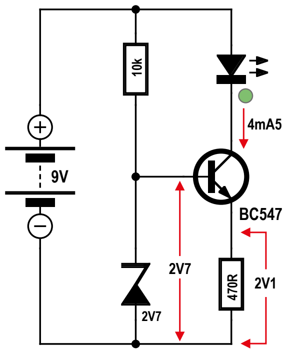


Figure 6. A constant current source using a Zener diode.

Test the results for yourself with a new battery and with a nearly discharged battery, or with an adjustable power supply. The LED brightness should remain nearly the same as long as the battery isn't completely flat, and an ammeter should show a constant collector current.

Another commonly used type of constant current source uses a second transistor in place of the LED. In this case the reference voltage is effectively the base-emitter voltage (around 0.6 V) of the left-hand transistor in **Figure 7**. If the voltage drop over the emitter resistor is too high, the left-hand transistor reduces the base current until everything is as it should be.

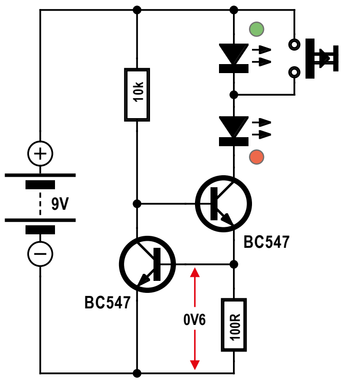


Figure 7. A constant current source with two transistors.

The current provided by a constant current source is not only independent of supply voltage variations, but also independent of the voltage drop over the load. You can use the switch in the circuit shown in **Figure 7** to drive either one or two LEDs with the constant current source. The same current flows in both cases. This current source is quite practical and provides a current of approximately 6 mA.

Soft LED blinker
This simple ATtiny13 application implements an LED driver that causes the LED brightness to slowly rise and fall repeatedly. Of course, you could achieve the same effect with a controlled current source, but standard practice in the microcontroller world is exactly the opposite:

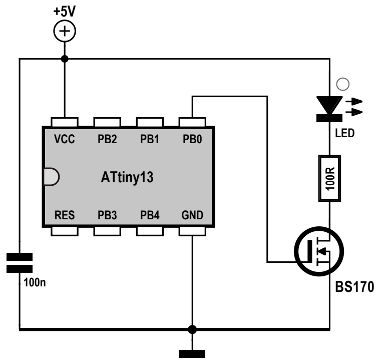


Figure 8. A soft LED blinker.

Everything is either hard on or hard off — absolutely binary. In order to achieve variable brightness in this situation, you can use pulse width modulation (PWM). This involves switching an output alternatingly on and off in rapid sequence to generate a pulse train that is so fast that you can't see the individual pulses. Here the LED brightness depends on the ratio of the On and Off times. The ATtiny13 uses its internal Timer 1 for PWM output; the PWM signal appears on the PB0 pin. Here we use a VMOS transistor type BS170 as a power driver. As you can see, the base resistor needed with a bipolar transistor is not required here.

```
$regfile = „attiny13.dat“
$crystal = 1200000
Dim I As Byte
Dim D As Integer

Config Portb = Output
Config Timer0 = Pwm , Prescale = 1 , Compare A Pwm = Clear Down

Do
   For I = 40 To 215
     If I < 128 Then
       D = I
```

```
          D = D * D
        End If
        If I > 127 Then
          D = 255 - I
          D = D * D
        End If
        D = D / 64
        Pwm0a = D
        Waitms 60
      Next I
      Waitms 800
    Loop
    End
```

Listing 1.

Field-effect transistors

The two main classes of transistors are bipolar transistors and field-effect transistors (FETs). A field-effect transistor consists of a small piece of semiconductor material with only one type of doping (either p or n). It has an isolated gate electrode, which alters the number of charge carriers in the region between the source and the drain when a voltage is applied to the gate. This changes the conductivity in this region, which is called the channel. Depending on the voltage on the gate, the concentration of charge carriers in the channel is either depleted or enriched. The advantage of field-effect transistors is that they do not need a current to control the output current, but instead a voltage.

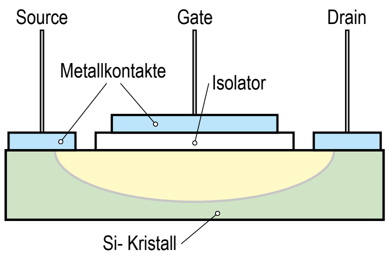


Figure 9. A MOSFET structure.

The gate (G), source (S) and drain (D) terminals of a FET correspond to the base (B), emitter (E) and collector (C) terminals of a bipolar transistor. There are numerous types of field-effect transistors. Along with the junction FET, which has an isolating diode junction between the gate and the channel, there is another type (MOSFET) that has a metallic oxide isolating layer. Like bipolar transistors, MOSFETs are available in two polarities, called n-type MOSFETs and p-type MOSFETs according to the polarity of the source and drain voltages. MOSFETs are basic building blocks of many types of integrated circuits, especially computer ICs. Complementary n-type and p-type MOSFETs are often used in the same IC, which is called CMOS technology. Power MOSFETs are usually fabricated with a vertical structure and accordingly designated VMOS. The following table provides a comparative overview of the specifications of a number of typical VMOS transistors:

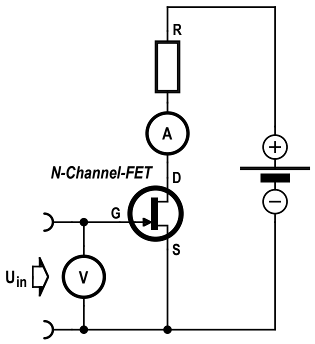


Figure 10. A FET basic circuit.

Typ	N/P channel	I_{max}	U_{max}	P_{max}	$R_{DS\text{-}ON}$	C_{GS}	C_{DG}
BS107	N	150 mA	200 V	0,8 W	28 Ω	50 pF	4 pF
BS170	N	175 mA	60 V	0,8 W	5 Ω	60 pF	5 pF
BS250	P	180 mA	45 V	0,8 W	14 Ω	60 pF	5 pF

Junction field-effect transistors use a semiconductor junction to isolate the gate from body of the transistor. This means that the gate voltage must always be negative, as otherwise the GS junction would be biased into conduction. JFETS are classified as depletion-mode FETs because charge carriers are normally present in the channel when no gate voltage is applied, and they can be depleted by applying a voltage to the gate. If an increasingly negative voltage is applied to the gate, the channel between the source and the drain is gradually pinched off until the transistor stops conducting. Incidentally, this behaviour corresponds exactly to that of a vacuum valve.

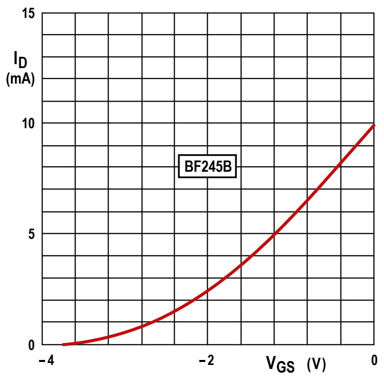


Figure 11. Characteristic curve of a JFET.

A typical example of this type of FET is the BF245, which is primarily intended to be used in high-frequency applications. It has a typical transconductance of 5 mA/V, which means that changing the gate voltage by 1 V causes the drain current to change by 5 mA. The BF245B has a typical cutoff voltage (zero drain current) of approximately –4 V and a drain current of approximately 10 mA with zero gate voltage.

Chapter 5 • Electronics for Starters (5)

Voltage stabilisation

In the previous instalment of this course series we looked at circuits for constant-current sources. Now it's time to examine ways to generate stable voltages. Of course, you can always use an integrated voltage regulator, but there are many other interesting approaches, most of which need only a few (usually discrete) components.

If you carefully inspect the many schematic diagrams published in Elektor magazine, you will repeatedly encounter voltage stabilisation circuits. Some devices are powered by batteries, and the output voltages of batteries can vary over a relatively wide range. For this reason, a voltage regulator is often used in such devices to provide a somewhat lower but stable voltage, such as 5 V for digital circuitry or a microcontroller.

Diode stabilisation

Voltage stabilisation is not a difficult issue in practice, since wonderful voltage regulator ICs such as the 7805 are readily available. Operating from an input voltage anywhere between 7 V and 30 V, it supplies an output voltage of exactly 5 V. However, this IC contains a large number of components. You can manage with a single semiconductor device instead, namely a Zener diode. The 7805 actually contains a Zener diode, along with lots of transistors. A Zener diode is a type of diode in which breakdown occurs at a well-defined reverse voltage. For instance, you can buy a Zener diode with a rated voltage of 6.8 V if you want to stabilise a supply voltage at this value. **Figure 1** shows the corresponding basic circuit.

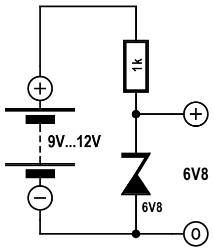


Figure 1. Voltage stabilisation with a Zener diode.

The operating principle of this circuit can be seen from the characteristic curve of a typical Zener diode (**Figure 2**). First breakdown occurs when the reverse voltage rises above a certain value (UZ), leading to a sharp increase in the reverse current. The voltage across the diode remains stable at the breakdown voltage, as long as you don't overdo it with the reverse current. Second breakdown is a frequently observed fault with Zener diodes. If the Zener diode becomes too hot, the junction shorts out, and after this the diode 'stabilises' the voltage at something close to zero volts.

Strictly speaking, the designation 'Zener diode' is not always correct, because two different phenomena are responsible for the breakdown effect with voltages over the range of 3 V to 200 V. The true Zener effect predominates at voltages below 5.6 V. It has a negative temperature coefficient, causing the Zener voltage to drop by up to 0.1% per degree. The avalanche effect, which predominates above 5.6 V, has a positive temperature coefficient. Zener diodes with a rated voltage of 5.1 V have the lowest temperature coefficient, while Zener diodes rated at 7.5 V or so have the steepest characteristic curves and therefore the lowest differential internal resistance. This means that they provide the best voltage stabilisation with variable Zener current.

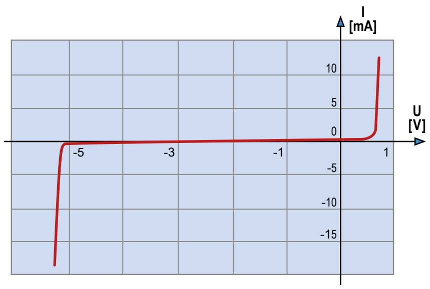


Figure 2. Characteristic curve of a Zener diode.

Quick solution
Sometimes all you need is a more or less stable voltage in the range of 2 to 3 V, with relatively little current. For example, you may want to power the RF front end stages of a simple radio circuit from a low voltage, while the output amplifier operates directly from a 9 V battery. In such cases you can use a forward-biased LED as a simple voltage stabiliser (**Figure 3**).

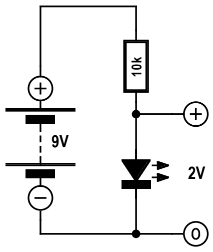


Figure 3. Voltage stabilisation with an LED.

The base-emitter junction of a perfectly ordinary NPN transistor has the same characteristics as a Zener diode. The Zener voltage is usually somewhere in the range of 7 to 12 V. The value with a BC547B is approximately 9 V, which lies in the favourable range with very low internal resistance. This type of transistor can therefore be used quite nicely as a Zener diode, although the exact Zener voltage cannot be known in advance. The manufacturers' data sheets don't say anything about this, although they do state that the reverse breakdown voltage of the base-emitter junction is at least 5 V. Here first breakdown of the base-emitter junction is a sort of useful side-effect. If you don't have a Zener diode handy, you may be able to make do with a transistor (**Figure 4**). Try it for yourself: apply reverse voltage to the base-emitter junctions of a few transistors and measure their Zener voltages.

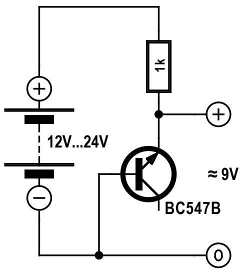


Figure 4. Using an NPN transistor as a Zener diode.

By the way, there's another little-known side effect: the 'Zener diode' of an NPN transistor emits yellow light. If you try this experiment with a transistor in a metal package (such as the BC140 in a TO5 package) with the package opened up, you can see this light if you work in absolute darkness. Let's hear it for silicon LEDs!

Efficiency

Although voltage stabilisation with a Zener diode is easy, it has some drawbacks. One of the major drawbacks is power dissipation. This results from the fact that the series resistor must be dimensioned for the lowest input voltage and the highest output current. For example, if the circuit shown in **Figure 4** has to supply a maximum current of 2 mA, the maximum output power is just 18 mW. The voltage over the series resistor is 3 V at the lowest input voltage of 12 V. This means that 1 mA flows through the Zener diode and 2 mA flows through the load. A current of less than 1 mA through the Zener diode is undesirable because it places the operating point on the knee of the characteristic curve, resulting in higher internal resistance and poorer voltage stabilisation. However, even at this current level one-third of the input current is 'wasted' in the Zener diode. With even higher load requirements, the recommenced minimum Zener current is 5 mA.

Things are even worse when the input voltage rises to 24 V. In this case the voltage drop over the series resistor is 15 V and the current is 15 mA. The resulting total input power is 360 mW. Compared with the useful power of 18 mW, this yields an efficiency of just 5%, which is terrible and is hardly tolerable in times of energy crisis. Fortunately, there is a solution to this problem.

Series regulators

Efficiency can be improved significantly if the Zener diode is followed by a transistor operating in common-collector mode, with the collector of the transistor connected directly to the positive terminal of the supply voltage (**Figure 5**). This type of circuit is also called an emitter follower because the voltage on the emitter always follows the voltage on the base, with an offset of 0.6 V. In the present case the emitter voltage is 5.6 V (6.2 V – 0.6 V).
Here the Zener circuit only has to supply the base current for the transistor. As a result, the input current is only slightly higher than the output current of the circuit over a wide range of operating conditions. Most of the power dissipation occurs in the series-pass transistor, and it depends only on the output current and the difference between the input voltage and the output voltage.

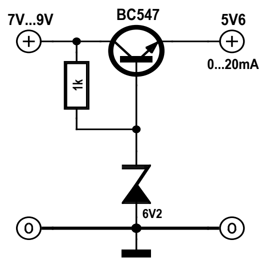


Figure 5. Using a transistor as a series regulator.

Only a small change is necessary to convert this circuit into an adjustable voltage regulator. As shown in **Figure 6**, a potentiometer acts as a voltage divider for the stabilised auxiliary voltage. The output voltage is always approximately 0.6 V lower than the voltage on the wiper of the potentiometer.

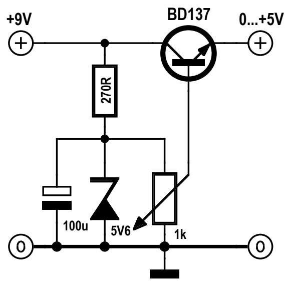


Figure 6. An adjustable voltage regulator.

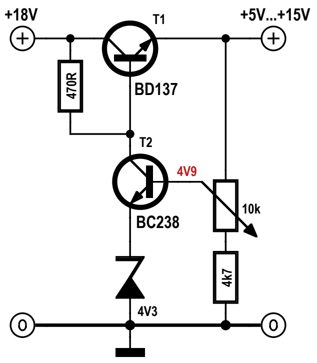


Figure 7. An improved adjustable voltage regulator.

To ensure adequate stability with variable output current, the current through the potentiometer must be greater than the maximum base current.

Even better stabilisation can be achieved by using an active output voltage follower, as shown in **Figure 7**. Here an adjustable portion of the output voltage is compared with the voltage on the Zener diode. The difference forms the error input to the control circuit, which drives the base voltage of series-pass transistor T1 via transistor T2. With this circuit it is possible to obtain an output voltage that is significantly higher than the Zener voltage, and which is close to the input voltage. This circuit can be used to build an adjustable power supply for currents up to 1 A. The actual load capacity depends on the cooling of the BD137 power transistor.

All that's missing here for a full-fledged adjustable power supply is current limiting. For this purpose, we insert a small resistance in the negative lead (**Figure 8**). The voltage drop over this resistor is proportional to the output current. The extra transistor starts conducting when this voltage drop rises above 0.6 V or so. This reduces the base voltage of the series-pass transistor. With a 1 Ω current sense resistor, the maximum possible current in the event of a short circuit is 0.6 A. However, the power dissipation of the series-pass transistor is very high in this situation. It won't be able to handle this without a large heat sink.

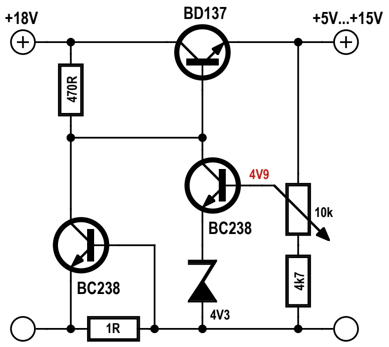


Figure 8. Adding current limiting.

Integrated voltage regulators

It's good that low-cost integrated voltage regulators are available for all common output voltages. A 7805 can deliver up to 1 A at 5 V, although a heat sink is necessary at such high current levels. In many situations the current is much lower, and in such cases the 78L05, with a maximum current rating of 100 mA, is sufficient. However, you should note that the 78L05 has a different pinout than its larger cousin. These voltage regulators require two capacitors — one at the input and the other at the output — to prevent oscillation at frequencies of several hundred kilohertz (**Figure 9**).

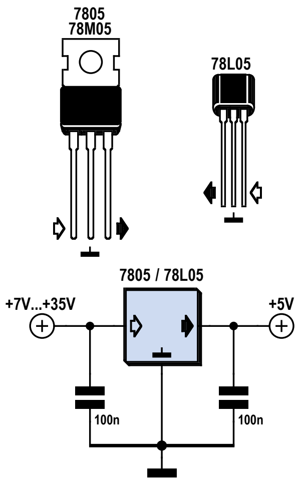


Figure 9. 780x voltage regulator circuit with bypass capacitors.

These voltage regulator ICs contain everything already described in this instalment of our basics course using discrete semiconductor devices. If you examine the internal circuit shown in Figure 10, you will see a lot of familiar things, such as the Zener diode with its series resistor. The actual control circuit is somewhat more complicated and includes a differential amplifier as well as a current mirror (see inset). The series pass transistor is implemented as a Darlington pair consisting of Q11 and Q12, with most of the power dissipation occurring in Q12.

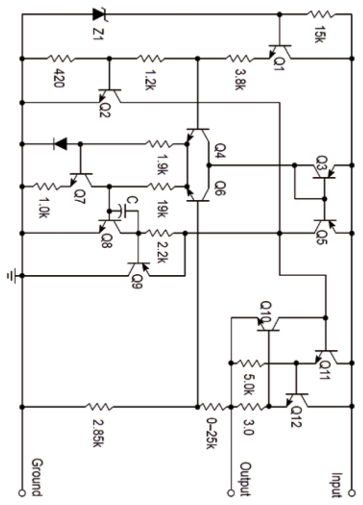


Figure 10. Internal structure of a 78Lxx (source: Motorola).

Current limiting is handled by transistor Q10, which chokes off the base current to the Darlington pair Q11/Q12 if and when necessary. As might be expected from the 3 Ω value of the current sense resistor, the cutoff current is 200 mA. However, the IC is already very hot at this point, since the voltage on the base of Q10 is less than 0.6 V. The IC is protected against overcurrent and overtemperature. The overtemperature protection circuitry is built around Q7, Q8 and Q9.

Current mirror

A current mirror, as illustrated by this circuit, is a distant cousin of a constant-current source. The (constant) current through the 1 kΩ resistor is mirrored by the two transistors, and the collector current of the right-hand transistor is nearly the same as the that of the left-hand transistor. The base and collector of the left-hand transistor are connected together, which causes the base-emitter voltage to automatically assume a value that results in the specified collector current. In theory, if the second transistor has the same characteristics it should have the same collector current at the same base-emitter voltage. In practice, the current is usually slightly different because it's difficult to obtain identical transistor characteristics. This circuit is primarily used in ICs, where a large number of transistors on the same chip have the same characteristics.

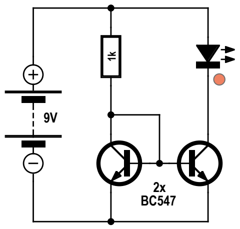


Figure 11. A current mirror with two transistors.

It's also important that both transistors have the same temperature, since the transfer characteristics are temperature dependent. A current mirror of this sort can therefore be used as a temperature sensor. Try touching one of the transistors with your finger. The resulting heating changes the output current, which can be seen from the change in the brightness of the LED. Depending on which of the two transistors you touch, you can make the LED a bit brighter or a bit darker. The temperature dependence of the current mirror is actually a drawback of this circuit. This sort of thing is often seen in electronics, where something that is an undesirable 'degrading' effect in one situation is a desirable 'useful' effect in another situation.

Voltage monitor

Many circuits require an operating voltage of 5 V and have a maximum tolerance range of ±10%. In such cases it's a good idea to monitor the actual voltage. Here we want to use a microcontroller to monitor the voltage and generate suitable indications. A green LED should light up when the voltage is within the tolerance range (4.75 to 5.25 V). A red LED should light up if the voltage is too low, and a yellow LED should light up if it is too high. The microcontroller operates from the voltage that it monitors. It compares this voltage with an internal reference voltage of 1.1 V.

The source code file for this project, Tiny13_V-V_monitor.bas, can be downloaded free of charge from www.elektormagazine.com/magazine/elektor-201205/19874.

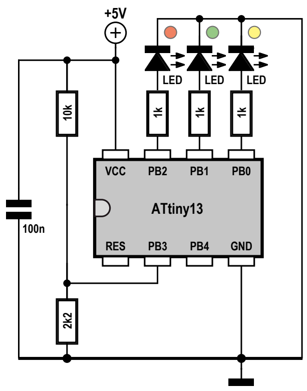


Figure12. A voltage monitor with three LEDs.

```
'Voltage Monitor
$regfile = "attiny13.dat"
$crystal = 1200000
$hwstack = 8
$swstack = 4
$framesize = 4

Dim U As Word

Config Adc = Single , Prescaler = Auto , Reference = Internal
Start Adc
Ddrb = &H07                  'B0/1/2 outputs

Do
  U = Getadc(3)              '0..6.1V

  If U < 797 Then            '4.75 V
     Portb = &H04            'red
  Else
     If U > 880 Then         '5.25 V
        Portb = &H01         'yellow
     Else
        Portb = &H02         'green
     End If
  End If
  Waitms 1000
Loop

End
```

Listing 1.

Chapter 6 • Electronics for Starters (6)

Flip-flops

To understand the basics of electronics, you have to go beyond high-level block diagrams — now and then you need to look at the details inside the blocks. For example, a microcontroller consists of a large number of functional blocks, each of which is composed of just a few transistors. The best way to learn how these subunits work is to conduct simple, hands-on experiments.

On or off, one or zero: welcome to the world of digital logic! Along with analogue tasks, transistors can also be used for digital tasks. In the past there were even full-fledged computers built entirely from discrete transistors. One of the most important basic digital circuits is the flip-flop. Even a single flip-flop can be put to good use in practical applications.

The flip-flop

A flip-flop is a circuit with two stable states. Flip-flops are important basic elements of digital computer technology, particularly for use in counters and memories.

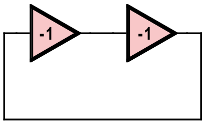


Figure 1. Principle of the feedback amplifier.

Flip-flop circuits use positive feedback of an in-phase, amplified signal. This can be implemented using two inverting amplifier stages (**Figure 1**). As each stage inverts the signal, the input and output signals of the overall circuit are in phase. The feedback from the output to the input causes the output signal to go to one of two states. A rising voltage on the output, for example, is amplified by the circuit and drives the output voltage even higher. As a result, the circuit switches to the fully 'on' state. In the other direction, a falling output voltage causes the output to be quickly switched to the 'off' state.

The amplifier can be built as a two-stage, directly coupled transistor circuit (i.e. without coupling capacitors). Each common-emitter stage (**Figure 2**) inverts its input signal. This simple circuit is effectively a bistable flip-flop, which means that the output voltage can be either low (nearly 0 V) or high (nearly the same as the supply voltage), and it remains in the given state for an indefinite length of time. The state can only be changed by some sort of external action.

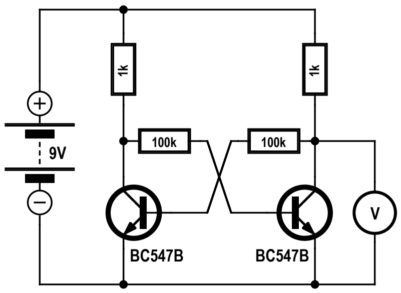


Figure 2. A bistable flip-flop.

When the output voltage is low, the left-hand transistor does not receive any base current and is therefore cut off. As a result, its collector voltage is high and the right-hand transistor receives full base current, causing it to be driven hard on. This causes the output voltage to stay low. When the output voltage is high, the situations of the two transistors are exactly reversed. It is impossible to predict which state the circuit will assume after being switched on — it's simply a matter of chance.

RS flip-flop

Now let's add a pair of LEDs our circuit, along with two pushbutton switches that can be used to change the state as desired (**Figure 3**). After switching on the power, you will see that one of the two LEDs is lit. You can't say in advance which one of the two will light up. Which way the circuit goes when it is powered on (i.e. which state it assumes) usually depends on the difference in the current gains of the two transistors. If they happen to have identical characteristics, noise (which is always present in electronic circuits) plays a role. Accordingly, it may happen that the circuit sometimes ends up in one state after being switched on, and sometimes in the other state.

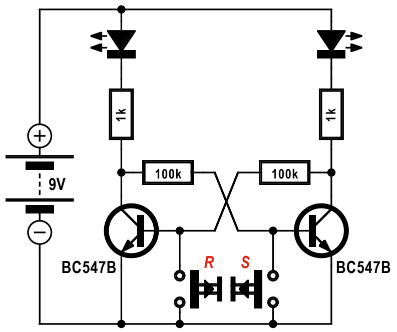


Figure 3. RS flip-flop.

However, it's possible to set the circuit to either state as desired. To do so, simply press one of the two pushbuttons, each of which shorts out the base current of the associated transistor. This circuit is called a reset/set (RS) flip-flop.

An RS flip-flop can be used as a 1-bit data memory. It can save states such as 'red' of 'green' (if you use LEDs with these colours), 'on' or 'off', 'yes' or 'no', etc. You could use this sort of device at home to leave a message, such as "Back right away" or "Out for a while". Your message remains present until it is changed.

Reset buttons and reset inputs are commonly found on computers and microcontrollers. This is hardly surprising, since complex systems of this sort contain a large number of flip-flops, some of which are used to store data. When the power is first switched on, all static flip-flops initially assume a random state. To bring order to this chaos, you need a reset function. A short reset pulse is all it takes to set everything nicely to zero. Now the work can begin.

Incidentally, modern microcontrollers use MOSFETs instead of bipolar transistors, and things are even easier with MOSFETs. An RS flip-flop using two BS170 MOSFETs (**Figure 4**) can manage without the base resistors needed by the version with bipolar transistors.

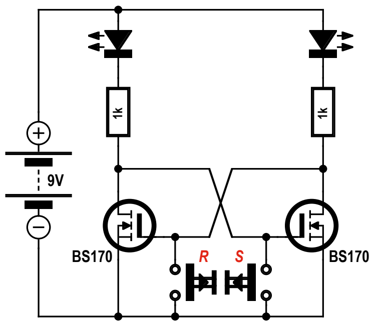


Figure 4. RS flip-flop using two MOSFETs.

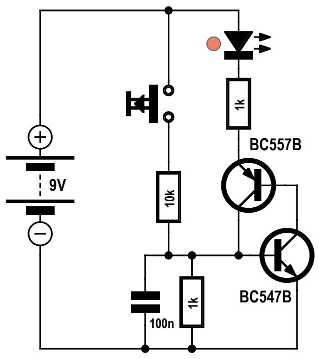


Figure 5. A flip-flop with a NPN transistor and a PNP transistor.

Triggering and clearing

A flip-flop can be built with an NPN transistor and a PNP transistor wired as shown in **Figure 5**. Here the collector current of each transistor is the base current of the other one. As a result, both transistors are either cut off or conducting. After power is switched on, the circuit is initially in the non-conducting ('off') state.

Briefly actuating the switch puts the circuit in the conducting ('on') state. The transistors remain in the conducting state until the supply voltage is switched off. This circuit therefore acts the same way as a thyristor (see inset). Incidentally, the capacitor in the circuit prevents the circuit from accidentally triggering by itself when the supply voltage is applied.

Monostable flip-flops

In many situations what is wanted is a flip-flop circuit that changes to a different state for a defined period instead of indefinitely. For example, this could be used to build a timer that is triggered by a pushbutton and switches off automatically after a certain time. This behaviour can be achieved by including a capacitor in the feedback path. The capacitor starts charging when the timer is triggered. When it is fully charged, no more current flows in the feedback path and the circuit returns to the stable quiescent state. The timeout period ('on time') of the circuit shown in **Figure 6** is approximately 10 seconds.

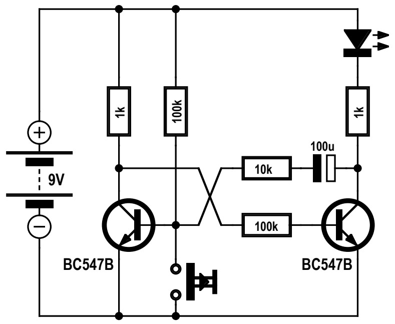


Figure 6. A monostable flip-flop.

Schmitt trigger

A Schmitt trigger is a circuit that converts a variable input signal into two distinct states (High or Low). It has two threshold levels separated by a hysteresis zone. For example, with threshold levels of 2 V and 1 V, the output changes to the 'off' state when the input signal rises above 2 V and remains in that state until the input signal drops below 1 V. The current state remains unchanged as long as the signal level remains in the hysteresis zone.

The classic Schmitt trigger circuit (**Figure 7**) employs feedback over a common emitter resistor. The thresholds and the amount of hysteresis can be set as desired by selecting suitable resistor values.

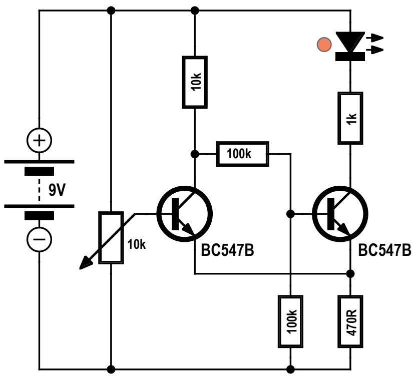


Figure 7. The classic Schmitt trigger circuit.

As a diversion, you can try a little RF experiment with this circuit. Connect a length of wire to the input to act as an antenna, and then adjust the potentiometer very close to one of the switching points. Now flip a light switch. You may hear some static on the radio, and at the same time the circuit will be triggered and its state will change. Here the Schmitt trigger effectively acts as a broadband radio receiver, with the light switch acting as the associated radio transmitter.

An illustrative example of a Schmitt trigger application is a twilight switch (**Figure 8**). Here the idea is to switch on a lamp when it gets dark outside. An important consideration is that the lamp should not flicker near the switching point. This means that the circuit should switch off only after the light level rises significantly. In other words, there must be some hysteresis between the two switching points.

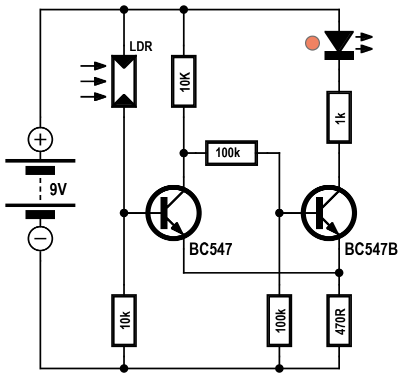


Figure 8. A twilight switch.

The voltage at the input of the Schmitt trigger here depends on the resistance of the LDR. This resistance increases with deceasing ambient light level, which causes the LED to switch on. When the ambient light level rises, the LED is switched off. There is noticeable hysteresis between the two switching points. It should be sufficient to prevent the circuit from responding to the flickering of artificial lighting.

Simplified Schmitt trigger

A simpler implementation of a Schmitt trigger is shown in **Figure 9**. Here two NPN transistors in common-emitter configuration are directly coupled. An additional resistor from the output to the input provides the necessary feedback for the switching hysteresis.

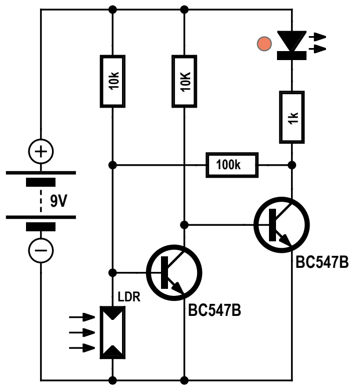


Figure 9. A simpler implementation of a Schmitt trigger.

For this experiment we again use the LDR as a variable resistor. However, in this case the LED switches on when the light level rises and switches off when the light level falls. This circuit switches cleanly, even with a slowly decreasing light level. It has the typical Schmitt trigger characteristic of converting a gradual change into a step change. The sensitivity of the circuit can be adjusted over a wide range to suit various lighting conditions by modifying the voltage divider at the input. With a 10-kΩ fixed resistor, it switches at a relatively high light level (e.g. close to a lamp). With a 100-kΩ resistor, it is suitable for sensing typical light levels in residential rooms.

Thyristors

Thyristors are bistable switching devices with three electrodes: a cathode (K), a gate (G) and an anode (A). The gate electrode is used to trigger (switch on) the thyristor, which remains conductive until the circuit is interrupted. The structure of a thyristor is similar to that of a transistor, but it has four semiconductor layers (NPNP). The gate electrode operates in a similar manner to the base of an NPN transistor. If the gate current rises above a specific threshold value, the thyristor is triggered and starts conducting. It remains conducting until the current stops flowing through the device. This can be done by switching off the supply voltage or by briefly shorting out the thyristor. Another technique that is often used is to operate the device with an AC voltage. In this situation, the thyristor stops conducting each time the voltage passes through zero.

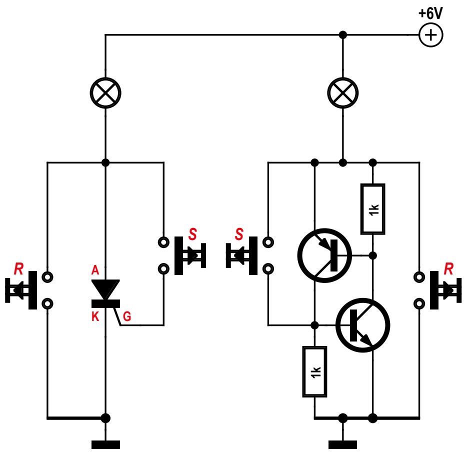


Figure 10. An equivalent circuit with bipolar transistors.

The circuit diagram here also shows an equivalent circuit with bipolar transistors. This equivalent circuit can also be switched on (triggered) or off in the same manner. Both circuits can be used in the same way as an RS flip-flop.

Twilight switch

It's quite easy to implement a Schmitt trigger function using a microcontroller. You simply measure the input voltage and compare it with pre-defined threshold values. The output is either switched on or switched off, depending on the results of these comparisons. The two switching thresholds are defined independently. This gives you complete control over the amount of hysteresis. In addition, you can build a defined delay into the program. A one-second wait loop also helps prevent undesirable switching jitter.

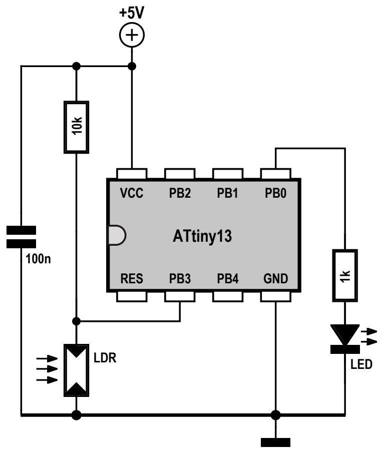


Figure 11. LDR at the analogue input.

We implemented a twilight switch using the Tiny13 and a few additional components.

```
'Dämmerungsschalter
$regfile = "attiny13.dat"
$crystal = 1200000
$hwstack = 8
$swstack = 4
$framesize = 4
Dim U As Word
Config Adc = Single , Prescaler = Auto
Start Adc
Config Portb = 1          'Output B.0
Do
 U = Getadc(3)
 If U < 400 Then Portb.0 = 0
 If U > 600 Then Portb.0 = 1
 Waitms 1000
Loop
End
```

Listing 2.

Chapter 7 • Electronics for Starters (7)

Blinkers and oscillators
In the previous instalment we worked with static flip-flops and Schmitt triggers. Now things get a bit more dynamic, with capacitors providing the necessary feedback. You will see LEDs blinking and flashing against an audio backdrop provided by signal generators. After all, making things blink and beep is one of the main functions of electronics.

A bistable circuit that automatically and continually switches states without any outside influence is called a multivibrator or an astable multivibrator. **Figure 1** shows an example of an astable multivibrator circuit, and as you can see right away, in this circuit the feedback is provided capacitors. If you use electrolytic capacitors for this, you must pay attention to the polarity because the voltage on the associated collector is (on average) higher than the voltage on the opposite base. The circuit remains in a stable state only as long as it takes for the capacitors to be charged or discharged. After this the circuit flips to the opposite state and the process starts again.

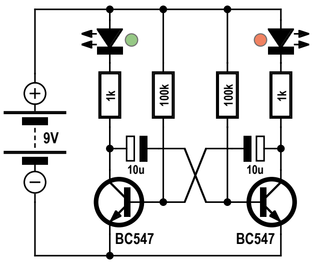


Figure 1. Multivibrator circuit.

A practical experiment with two 10 µF capacitors yielded a fairly low LED blinking rate with a period of around 1 second. You can adjust the toggle rate of the multivibrator over a wide range by changing the capacitor values. Experiments using relatively small capacitors and capacitors with differing values are also worthwhile. With a pair of capacitors having values of 100 µF and 100 nF, the circuit generates short flashes of light from one of the two LEDs. With a pair of 100 nF capacitors it produces rapidly flickering light.

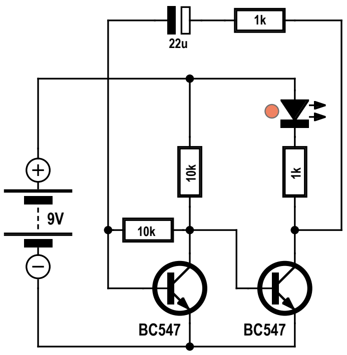


Figure 2. Simplified blinker circuit.

Simplified multivibrator

The multivibrator circuit can be modified to operate with just one capacitor. The circuit basically needs two transistors operating in common-emitter mode, each acting as an inverter that changes the phase of its input signal by 180 degrees. These two stages can be directly coupled to eliminate one of the capacitors, as shown in **Figure 2**.

To ensure reliable oscillation, this circuit must be dimensioned to have an operating point in the middle of the characteristic range in the absence of feedback. Otherwise the output transistor would be either completely cut off or driven fully on. The overall circuit would therefore not have enough gain to start oscillating. In this example, strong negative feedback on the first transistor (provided by the 10 kΩ resistor between the collector and the base) yields an operating point in the middle of the range. However, the feedback via the RC network is stronger than the negative feedback, with the end result that the output transistor is alternately cut off and driven into saturation.

It's a good idea to first put together the circuit without the feedback capacitor. The LED should light up dimly because the output transistor is not fully switched on. With the capacitor fitted, the LED will alternately be full on or full off. With a 22 µF capacitor, the LED blinks approximately once per second. The circuit also works with smaller capacitors, down to a value of 10 nF. As you reduce the capacitor value, the blinking gradually changes to rapid flickering. If you connect an acoustic transducer to the output, you will hear a clacking sound.

LED voltage converter

Red LEDs need 1.5 to 2 V, while blue or white LEDs need as much as 3 to 4 V. This is usually handled by connecting three batteries in series to provide 4.5 V. A series resistor is then used to reduce the supply voltage to the operating voltage. It would be better to be able to manage with just 1.5 V, which means we need a voltage converter.

The key component here is a small fixed inductor rated at 1.5 mH. This component looks like resistor, but it has a small ferrite core and a wire coil under the protective lacquer. If you wish, you can also make your own inductor for this purpose. Around 200 turns on a ferrite rod will do the job.

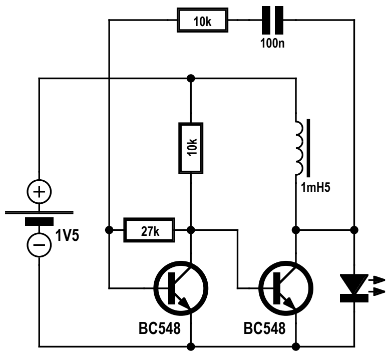


Figure 3. LED voltage converter.

The circuit shown in **Figure 3** is another simple multivibrator. The current through the inductor is switched on and off at a high rate. Here the inductor acts as a magnetic energy storage device. Each time it is switched off, it generates an induced voltage that adds to the battery voltage. The magnitude of this voltage depends on the connected load. It adapts automatically to the load, so a white LED receives a higher voltage than a red LED. Most voltage converters of this type also have a rectifier and a smoothing capacitor. Here they are not necessary because the LED acts as its own rectifier. A pulsating DC current flows through the LED. The average value of this current is somewhat less than the battery current because the voltage is higher. The overall efficiency of this circuit is higher than with the usual approach of using a higher battery voltage and a series resistor.

Audio generator
If we fit our simple multivibrator with a relatively small-value capacitor, it will generate a signal in the audio frequency range. A piezoelectric transducer (buzzer or beeper) connected to this circuit with produce an audible tone (**Figure 4**). A piezoelectric transducer has some of the characteristics of a capacitor and therefore affects the audio frequency. As a diversion, you can try a little experiment with this circuit. If you touch the transducer with your finger or with a hard object, the tone (frequency) and the sound level both change. The vibrating piezoelectric disc generates an AC voltage that affects the signal generator. To a lesser extent, acoustic echoes can also affect the signal generator.

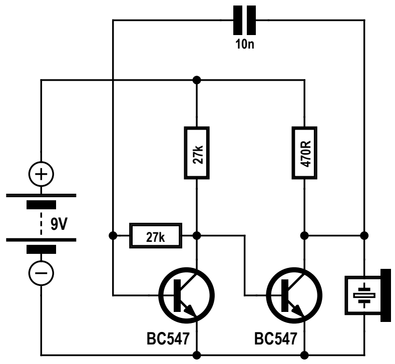


Figure 4. Driving a piezoelectric buzzer.

The frequency can also be modified by adjusting the value of the resistor in the feedback path. You could use a potentiometer for this, or you could use a light-dependent resistor (LDR). In the latter case the audio tone depends on the amount of light striking the LDR.

With the circuit shown in **Figure 5**, you can distinguish not only the brightness of the light, but also different types of light. Rapidly fluctuating artificial light modulates the frequency of the audio signal. Light from a fluorescent lamp produces a raspy tome. Light from a PC monitor also affects the sound, due to modulation of the audio signal at the frame rate.

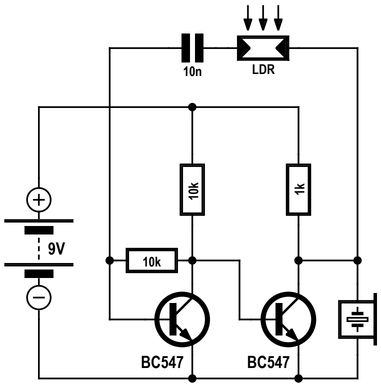


Figure 5. An adjustable audio signal generator.

Voltage to frequency converter

You can use a multivibrator to build a voltage-controlled oscillator (VCO) as illustrated in **Figure 6**. Here both base resistors are connected to a common voltage input. The capacitor charging current, and therefore the oscillator frequency, is directly dependent on the voltage applied to this input. An adjustable voltage divider can be used to set the input voltage to any desired value in order to set the frequency. This circuit can also be used as an acoustic voltmeter with a measuring range of 1 V to 9 V. Among other things, this is handy for quickly checking various batteries.

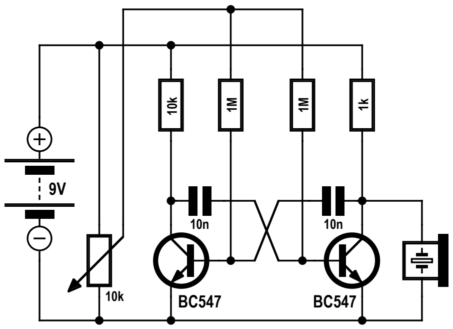


Figure 6. Using a multivibrator as a VCO.

NPN/PNP flip-flop circuit

You don't necessarily have to use a pair of NPN transistors. **Figure 7** shows a blinker with complementary transistors. As in the previously described circuit, negative feedback from the collector to the base of the NPN transistor establishes a stable operating point. The PNP transistor acts as an emitter follower. A signal with the right phase for the feedback can be taken from the resistor in series with its collector.

The impedance of the feedback path of this circuit is very high. As a result, it has a period of around 1 second with a feedback capacitance of just 1 μF.

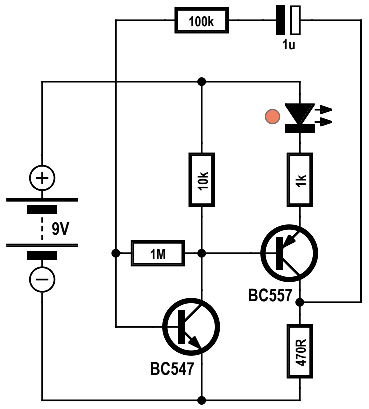


Figure 7. A blinker circuit using complementary transistors.

Energy-saving LED flasher

In shops you sometimes see advertising signs with a blinking LED that seems to work forever from a single battery. The circuit shown in **Figure 8** is an astable multivibrator with special properties. The 100 µF electrolytic capacitor is charged relatively slowly by a small current and discharged quickly by a short current pulse through the LED. This also generates the necessary voltage boost, since 1.5 V isn't enough for the LED.

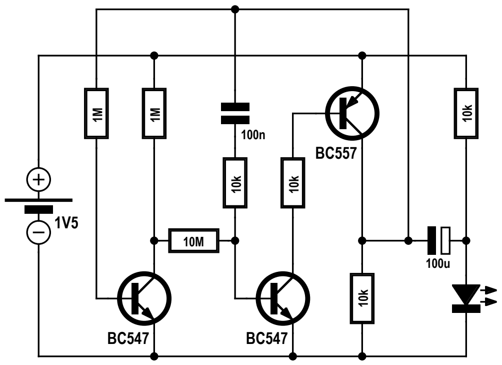


Figure 8. Low-power LED flasher.

This circuit is optimised for low-power operation, which is why the multivibrator circuit is built with a pair of complementary transistors (NPN and PNP). This avoids power losses from control currents. Both transistors conduct only briefly when the LED flashes. To ensure stable operating conditions and reliable oscillation, there is an additional stage with direct-coupled negative feedback. Here again the circuit is designed to work with very high resistance values to minimise power consumption.

The PNP transistor only conducts during the very short pulses occurring every couple of seconds. The output capacitor is charged to nearly 1.5 V between pulses. When the transistor is switched on, the voltage across the capacitor adds to the battery voltage. This produces an open-circuit voltage of nearly 3 V. A red or green LED with a forward voltage of 1.8 V to 2 V connected to the output will flash brightly.

You can use the charging current of the electrolytic capacitor to estimate the current consumption. The average voltage across the pair of charging resistors, each with a value of 10 kΩ, is 1 V. This means that the average charging current is 50 μA. Exactly the same amount of charge is additionally drawn from the battery during the LED pulse. so the average current is around 100 μA. If you assume a battery capacity of 2,000 mAh, the battery should last approximately 20,000 hours, which is over 2 years.

Sawtooth generator

Sawtooth signals, with their characteristic jagged waveforms, can be generated by periodically charging a capacitor to a specific voltage and then discharging it suddenly. This is illustrated in **Figure 9**. While the capacitor is being charged, the PNP transistor is cut off and no base current flows to the two NPN transistors. The discharge level is set to around 5.1 V (4.5 V + 0.6 V) by a voltage divider consisting of two 10 kΩ resistors; above this level the voltage on the base is 0.6 V lower than the voltage on the emitter. This means that the transistor starts to conduct when the voltage rises above 5.1 V, and this current is amplified to obtain a hefty discharge current. This causes the voltage to drop to 0.6 V, at which point the transistors are cut off and the next charging cycle begins. The circuit shown in Figure 9 has three transistors and is designed for very slow charging. It produces a sort of metronome signal: tick, tick, tick, ...

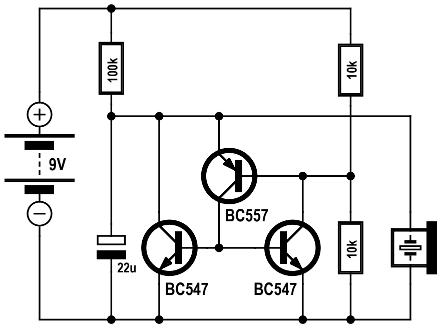


Figure 9. Sawtooth generator.

This circuit can be simplified somewhat by omitting the left-hand transistor. In many cases the resulting circuit still works properly, but sometimes there are difficulties with switching off this 'DIY SCR' (see the previous instalment for more information). If the charging current is low, the circuit may get stuck in the on state. This doesn't happen with the three-transistor version, which works reliably over a wide range of charging currents.

Another simplification is also possible. As the piezoelectric transducer is effectively a capacitor, you can omit the electrolytic capacitor. This converts the otherwise slow clock generator into a fast audio generator.

NPN sawtooth signal generator

A sawtooth generator can also be built with a single transistor operated in a highly unusual manner. Here the NPN transistor is wired the wrong way round in the circuit, with a positive voltage on the emitter, and the base lead is left open. The voltage on the capacitor gradually rises to approximately 9 V. At this point the transistor suddenly starts conducting and discharges the capacitor to approximately 7 V. The data sheet for the transistor doesn't say anything about this, and each transistor behaves somewhat differently. It's worth trying a variety of transistors in this circuit.

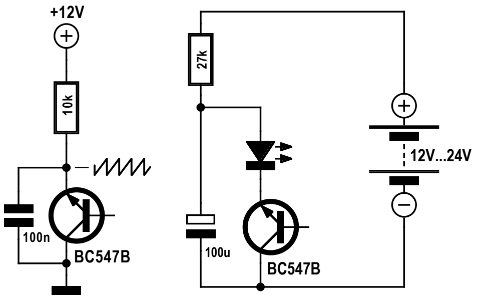


Figure 10. A sawtooth generator and LED blinker.

The discharge current pulse is strong enough to drive an LED. This requires a supply voltage greater than 12 V. The circuit works very nicely with a pair of nearly exhausted 9 V batteries. The LED keeps blinking for a long time as it sucks the batteries dry, with a gradually decreasing blink rate.

When operated this way with an 'inverted' voltage between the emitter and the collector, the transistor has a characteristic curve with a negative slope, which can easily be measured. The base-emitter junction exhibits the well-known avalanche breakdown effect at approximately 9 V. At this voltage the high electric field strength in the thin reverse-biased junction region causes the charge carriers to move so fast that they dislodge other charge carried from the crystal lattice. As result the number of charge carriers, and with it the current, rises very quickly. This is the same as what happens in a 9 V Zener diode, but a Zener diode has a positive internal resistance.

There's another factor involved with an inverted transistor. Here the emitter and collector switch roles, but due to the essentially symmetrical structure the transistor also operates in this inverted condition. The operating principle of a transistor is that some of the charge carriers that enter the base layer pass through the reverse-biased junction on the other side. In this case avalanche breakdown is occurring in the reverse-biased junction region, so there are even more charge carriers available to dislodge additional charge carries from the lattice. This results in a self-reinforcing avalanche effect.

A voltage to frequency converter with the Tiny13

The voltage to frequency converter described here is really handy because it can be used as an acoustic voltmeter. For instance, a concert A could mean that the battery voltage is OK. Your ears are also good at detecting slow changes. With the microcontroller you can implement a converter with a linear relationship between frequency and input voltage.

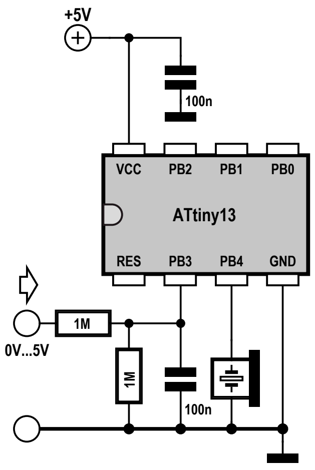


Figure 11. A voltage to frequency converter.

Here the ATtiny13 microcontroller operates with a 5 V supply voltage, which means that the measuring range of the A/D converter extends to 5 V. The range is enlarged to 10 V by a high-impedance voltage divider. A piezoelectric acoustic transducer is connected to port B4. The software implements a simple direct digital synthesis (DDS) function with square-wave output. An accumulator A is iteratively incremented by the input sample value until the most significant bit flips, at which point the state of the output signal is toggled. Here the accumulator has a width of 12 bits. At the highest input voltage the output state changes every four measurement intervals, which yields a frequency of just under 600 Hz. The source code can be downloaded from www.elektormagazine.com/magazine/elektor-201209/19963.

```
'U/f Converter  0...5 V 0...600 Hz
$regfile = "attiny13.dat"
$crystal = 1200000
$hwstack = 8
$swstack = 4
$framesize = 4

Dim U As Word
Dim A As Word

Config Adc = Single , Prescaler = Auto
Start Adc
Ddrb.4 = 1

Do
  U = Getadc(3)
  A = A + U
  A = A And &H0FFF
  If A >= &H0800 Then
    Portb.4 = 1
  Else
    Portb.4 = 0
  End If
Loop

End
```

Listing 1.

Chapter 8 • Electronics for Starters (8)

Audio Preamplifier

One of the most important applications for transistors is the amplification of audio signals. The device providing this function is the audio amplifier or audio frequency (AF) amplifier. This might be a microphone preamplifier or perhaps part of a radio circuit but the result is the same: something quiet is made louder!

The principle of a (preferably) linear amplifier is straightforward: an audio signal modulates (varies in volume and frequency) the small base current of a transistor, so that the amplified collector current produces corresponding audio signals of greater intensity. The most commonly employed amplifier circuit is the emitter circuit, the name implying that the emitter is connected to the common ground potential of the circuit. Whilst the base voltage and collector voltage can vary, the emitter voltage always remains constant, namely 0 V. In circuit diagrams we sometimes draw components with individual ground symbols and at other times a continuous ground connection, but the meaning is the same. Generally the ground connection is joined to the minus (negative) pole of the current source.

Our goal for an amplifier of this kind is to provide the greatest possible dynamic range before distortion occurs and undermines the satisfyingly faithful amplification achieved up till then. For this reason the quiescent collector current should be capped at just half the maximum possible current, with the collector resistor and supply voltage selected carefully to achieve this. In this way the current can be varied significantly in either direction to an equal extent without hitting limitations. All the same, achieving the correct quiescent current with a suitable base resistor is not entirely simple.

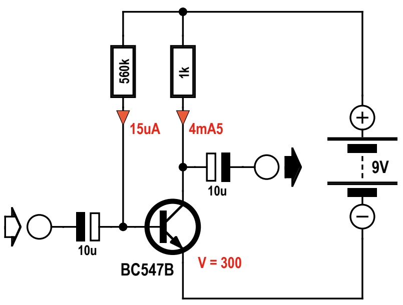


Figure 1. Setting the bias point for exactly 300-fold gain.

If we take a look at **Figure 1**, we see here some optimal values for the components, assuming that the BC547B transistor happens to exhibit current amplification of exactly 300-fold. The base current amounts to

$$IB = U / R = (9\text{ V} - 0.6\text{ V}) / 560\text{ k}\Omega = 15\ \mu\text{A}.$$

The collector current is then IC = IB × V = 15 µA × 300 = 4.5 mA. The voltage dropped across the collector resistor is U = I × R = 4.5 mA × 1 kΩ = 4.5 V. The remainder of the

9 V, also 4.5 V as it happens, is developed between the emitter and collector. This is the ideal case, in which the output can now be driven an equal extent in both directions. The voltage on the collector can therefore vary between 0 V and 9 V, meaning for example that a pure sinewave tone will be amplified at maximum output voltage without distortion. The voltage amplification is in this situation more than 100-fold.

However, if you build this circuit using a variety of transistors you will get a different result each time, because their current gain varies wildly. In the worst-case scenario, a transistor might have an amplification factor of 600, which is so great that the transistor is already switched hard-on to full saturation. Then you can forget about getting low-distortion output signals. Fortunately there is a perfect remedy: negative feedback.

Negative feedback

The simplest form of negative feedback occurs when the base resistor is connected not to the supply voltage but to the collector (**Figure 2**). The rule of thumb is this: RB = RC × V, where V is selected for the middle range of current amplification expected. For 300-fold current amplification and a collector resistance of 1 kΩ we need to use a base resistor of around 300 kΩ.

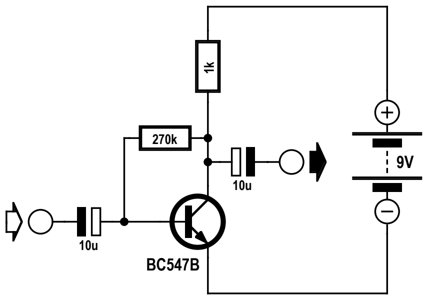


Figure 2. Setting the bias point with negative feedback.

This time our circuit reacts in a more measured manner to differing current gain situations. Greater gain leads to larger collector current and therefore to greater voltage drop across the collector resistor. The collector-emitter voltage falls at the same time and with it the voltage on the base resistor too, which in turn leads to a reduction in base current. The nett result is higher gain offset in part by smaller base current, so that the collector current rises less. We call this negative feedback because the increase in collector current is counteracted. In this situation the negative feedback leads to reduced amplification but with correspondingly less distortion. Most importantly, you can use absolutely any NPN transistor in your junk box to build this circuit and it will work each and every time. Or expressed more scientifically, the circuit can tolerate a wide variation of samples.

Achieving a still more accurate setting of the bias point calls for more effort. For this purpose we provide the transistor with an emitter resistor (**Figure 3**). By providing the base with a voltage divider we can ensure a fixed voltage on the base, in which the current flowing through the voltage divider is around 10 times larger than the base current, thus avoiding any reaction arising from variations in base current. The emitter voltage stabilises itself at a value in which the base-emitter voltage is below the base voltage. This means the emitter current remains stable and consequently the collector current too.

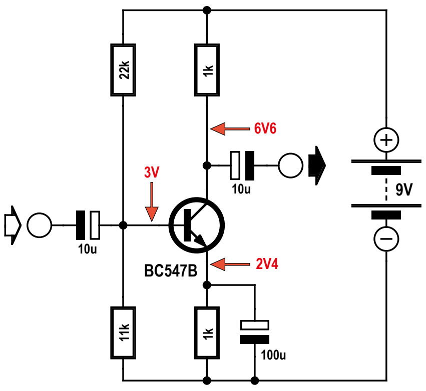


Figure 3. Stabilising the bias point.

In the sample circuit a base voltage of 3 V is given. This determines the emitter voltage of 2.4 V. An emitter resistor of 1 kΩ produces an emitter current of 2.4 mA. A collector resistor of 1 kΩ causes a voltage drop of 2.4 V. The resulting collector voltage is 6.6 V, along with a collector-emitter voltage of 4.2 V.

The circuit displays a strong relationship between voltage and negative feedback. A small variation in the emitter voltage has a direct effect on the base-emitter voltage and, on account of the steep characteristic curve of the base, leads to a significant change in the collector and emitter currents. Since only very small variations in base-emitter voltage are desired, the emitter voltage is set at a value that is always about 0.6 V below the base voltage. The function resembles that of the so-called emitter follower (see panel).

In principle a harsh level of negative feedback has the disadvantage of much reduced voltage amplification. To achieve adequate amplification of audio signals we need to boost the negative feedback for AC currents, specifically by using an emitter capacitor. Its value is determined by the lowest frequency being handled, without skimping (avoid using too small a value).

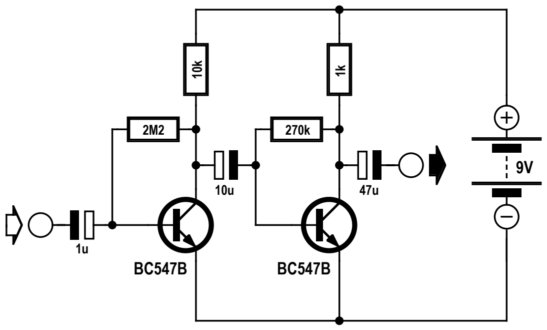


Figure 4. A two-stage audio amplifier.

Two stages

If the level of amplification provided by a single transistor is insufficient, a multi-stage amplifier will be required. **Figure 4** shows a version with two stages and R-C coupling. In most situations the simple form of negative feedback, with one resistor between base and collector, will be entirely adequate for obtaining an appropriate bias point. The circuit also demonstrates a further principle of amplifier development: from left to right, the circuit becomes successively lower in impedance. In this way we achieve high input resistance and low output resistance. The latter would not be up to connecting a loudspeaker but with headphones on the output, it would be fine for many applications.

DC-coupled stages

Multi-stage audio amplifiers can be created at reduced cost if you forego capacitor coupling between the individual stages and employ DC coupling instead. The collector resistor of the first transistor is now simultaneously the base resistor of the second one (**Figure 5**). Frequently this also simplifies setting the bias point, with the base current being appropriate for both transistors.

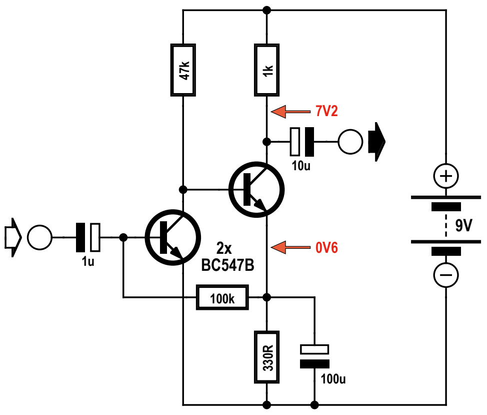


Figure 5. A two-stage audio amplifier with DC coupling.

Figure 5 shows a two-stage, directly coupled amplifier. The base current of the first transistor is branched off from the emitter of the second stage. This provides negative feedback and stabilisation of the bias point. The voltage drop across the relatively small base resistor of 100 kΩ can be disregarded and consequently the emitter voltage at the second transistor amounts to around 0.6 V. In the process the emitter resistor of 330 Ω determines the emitter and collector currents of the second stage, which are largely independent of the supply voltage.

The negative feedback should not lead to any reduction in AC voltage amplification. An additional emitter capacitor takes care of ensuring that only DC current is fed back. In practice the amplifier has a low frequency limit that can be set even lower with a larger emitter capacitor.

Three stages

The position is even simpler with a three-stage amplifier because negative feedback can be applied direct from the output to the input (**Figure 6**). Each stage reverses the signal phase by 180 degrees. With a two-stage amplifier we needed a flip-flop to achieve feedback. On the other hand, one or three stages produce negative feedback, which stabilises the bias point.

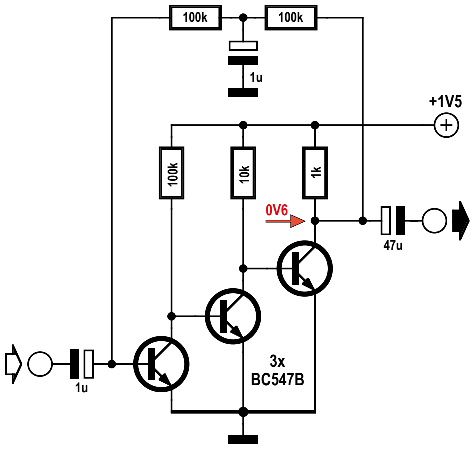


Figure 6. A three-stage, direct-coupled amplifier.

Figure 6 shows a circuit optimised for small supply voltages from 1 V upwards. Here, as in Figure 2, we see how greater amplification in the transistors leads to increased collector current flow in the third transistor and hence to larger voltage drop across its collector resistor. The collector-emitter voltage falls and consequently also the voltage dropped across the two 100 kΩ negative feedback resistors. This results finally in reduced base current, as explained previously above.

The collector-emitter voltage here amounts consistently to merely 0.6 V or so, enabling each stage to deliver relatively modest amounts of audio amplification. All the same, this is more than compensated by the high number of amplifier stages.

The negative feedback element of the circuit includes a low-pass filter with a capacitor to ground (bypass capacitor), which boosts the negative feedback for higher frequencies. It is essential for achieving high levels of audio amplification. The circuit is, for example, ideal as a sensitive microphone preamplifier or as audio amplifier in simple radio receivers. A headset can be connected directly in place of the collector resistor in the third stage. Incidentally the circuit was used in the 'Tapir' RF Sniffer described in the Elektor July & August 2012 edition.

The Emitter Follower

The emitter follower is also known as the common collector circuit, as the transistor operates with a constant collector voltage. The output of the amplifier is at the emitter. Every change of voltage on the input is reflected automatically at the output because even a very small variation in the base-emitter voltage is sufficient to alter the emitter current significantly. The emitter voltage lies always around 0.6 V below the base voltage.

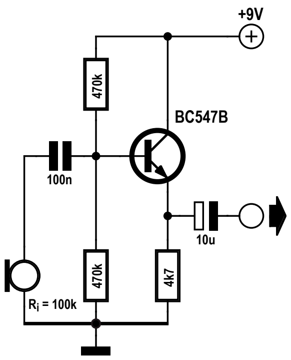


Figure 7. A emitter follower.

A change in the base voltage of 1 V is therefore reflected in one of almost 1 V in the emitter voltage as well. The precise value might be 0.99 V if a change in collector current required a variation in base-emitter voltage of 10 mV. The voltage amplification is thus almost unity (1), meaning the input voltage is not amplified even though the input current certainly is. The advantage of the emitter follower is its high input resistance or impedance. Whilst the input resistance of an emitter follower circuit is about 1 kΩ (according to its bias point), the collector circuit achieves 100 kΩ and more. For example, you can connect a crystal microphone or a crystal pick-up cartridge direct. This circuit also lets you use a simple piezo-ceramic transducer as a microphone (or for picking up sound waves in solid objects). It can be used like this for monitoring pulse beats for instance.

An audio millivoltmeter
An audio millivoltmeter would be very handy for investigating the amplifiers described here. The A-D converter in the ATtiny13 can in fact measure only DC voltages. All the same, by elevating the mean voltage to 2.5 V and making rapid measurements, AC voltages can be investigated as well. The result is sent serially to the PC and can then be displayed using a terminal program.

The principle of the measurement program is straightforward. First we determine the mean voltage by averaging. There then follows a rapid series of multiple individual measurements, from which the mean voltage is subtracted, creating an absolute value for each instance. The smallest measurable voltage step for the A-D converter is actually 5 mV. Even so, by averaging many separate readings we can get down to 1 mV. In the actual measurement loop process we carry out, process and add up 2,780 readings. Incidentally, the reading points are entirely asynchronous with regard to the test signal. However, the large number of test measurements made and the random factor combine to produce an adequate result, so long as the input signal remains between roughly 50 Hz and 50 kHz.

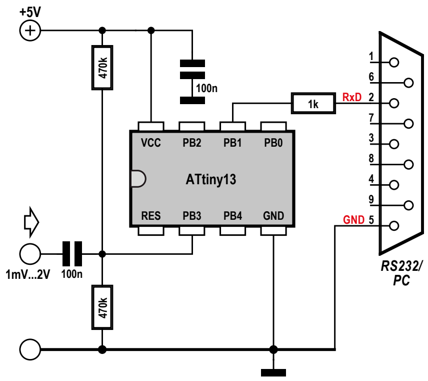


Figure 8. Analogue input and serial port.

How do we arrive at this ominous figure of 2,780? This figure is based on a reference voltage of 5 V and takes into account the difference between arithmetic averaging and true RMS measurement, so that the result in the case of a sinusoidal signal is displayed as actual effective values in mVeff. In order to determine accurate rms (root-mean-square) values, you would have to actually add up the square of the voltage and afterwards calculate the mean value from the square root. This would overwhelm the ATtiny13, however. For this reason we derive the arithmetic mean from the absolute voltage. This is around 10% too low, or more precisely by a factor of the root of 2 divided by pi / 2, i.e. 0.9003. One A-D step is 5000 mV / 1023 = 4.8876 mV. In addition the measured value is multiplied by 8 and then divided by 4096, thus effectively dividing by 512. In order to display everything correctly in mV, we need to multiply by 512 × 4.8876 / 0.9003 = 2780, and this is simplified when 2,780 measurements are totalled.

The result is remarkably accurate. Even values like 1 or 2 mVeff are displayed stably!

```
'Millivoltmeter  1 mVeff ... 2000 mVeff
$regfile = "attiny13.dat"
$crystal = 1200000
$hwstack = 8
$swstack = 4
$framesize = 4

Dim U1 As Integer
Dim U2 As Integer
Dim U3 As Long
Dim N As Integer

Config Adc = Single , Prescaler = Auto
Start Adc
Open "comb.1:9600,8,n,1,INVERTED" For Output As #1
```

```
Do
 U2 = 0
 For N = 1 To 64
   U1 = Getadc(3)
   U2 = U2 + U1
 Next N
 Shift U2 , Right , 3         ' /8
 U3 = 0                       ' Nullpunkt
 For N = 1 To 2780
   U1 = Getadc(3)
   Shift U1 , Left , 3  ' *8
   U1 = U1 - U2
   U1 = Abs(U1)
   U3 = U3 + U1
 Next N                       ' * 2780
 Shift U3 , Right , 12        ' / 4096
 Print #1 , U3
Loop

End
```

Listing 1.

Chapter 9 • Electronics for Starters (9)

Sine-wave oscillators

If we only used electronics to process existing signals, we would be missing an important aspect of electronics: generating oscillating signals, as if by magic. Oscillators are important parts of many devices and are used for a wide variety of purposes. For example, they can be used to generate audible signals or test signals for checking out circuits and modules.

RC oscillators

Everyone knows the unpleasant whistling that can occur with a public address system. It results from acoustic feedback between the loudspeaker and the microphone. The pitch of the tone varies from one situation to the next, and the effect can only be prevented by increasing the distance between the system components or reducing the gain.

In theory, any circuit or system with sufficient feedback can oscillate. The feedback path may be purely electronic, such as feedback from a signal output to an input. A necessary condition is the right phase relationship, which is present with a two-stage amplifier.

The circuit in **Figure 1** is similar to that of a multivibrator, but with adjustable feedback. A multivibrator always generates square-wave signals, but the circuit shown here can also generate sine waves or other waveforms. The feedback can be adjusted with the volume control to the point where weak oscillation just starts to occur. The waveform in this situation is usually sinusoidal.

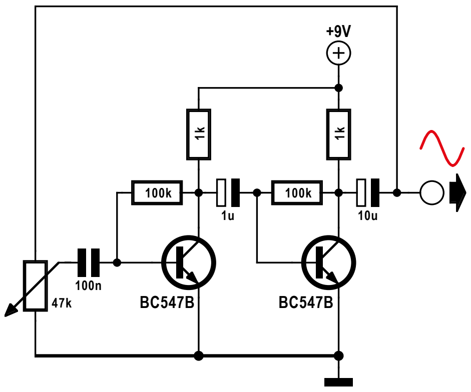


Figure 1. Oscillation caused by positive feedback.

It is also possible to generate an oscillating signal with a single transistor, even though it has a 180-degree phase shift. The required additional 180-degree phase shift can be achieved by connecting several RC networks in series. The phase-shift oscillator shown in **Figure 2** generates a sine-wave signal at approximately 800 kHz, which is ideal for purposes such as practicing your Morse code or providing a test signal for checking out audio amplifiers.

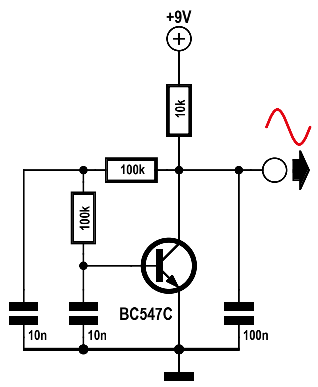


Figure 2. A phase-shift oscillator.

A working phase-shift oscillator can also be built using a BS170 field effect transistor. The circuit in **Figure 3** is designed with very high resistance values and oscillates at a frequency of 10 Hz. It draws a very low operating current of approximately 30 µA.

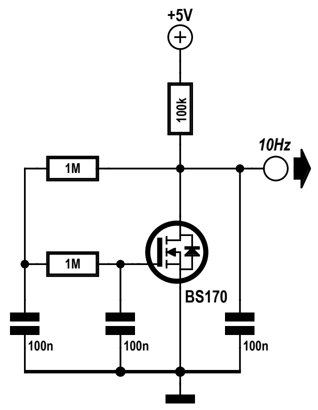


Figure 3. A phase-shift oscillator with a FET.

Ring oscillators

Up to now we have used one-stage or two-stage amplifiers to build oscillators. What happens if you have a circuit with three common-emitter stages? You would actually expect the feedback to be negative, since the overall phase shift is 180 degrees. However, in practice the circuit oscillates (**Figure 4**). The oscillating frequency rises with increasing supply voltage and can rise as high as 1 MHz.

What is happening here? We basically have a three-stage amplifier with negative feedback and very high voltage gain. However, each of the stages also causes a small time delay in addition to its gain. At a very specific frequency, the combination of these three delays results in an additional 180-degree phase shift. The negative feedback therefore turns into positive feedback at this frequency, and the result is oscillation. If you want to use a circuit of this sort as an amplifier for very low input signal levels rather than an oscillator, you must do everything possible to prevent any form of positive feedback. With such high gain it is not especially easy to prevent parasitic oscillations.

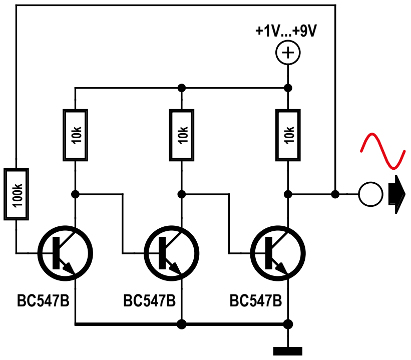


Figure 4. An oscillator with no capacitors..

It's easier to build a three-stage oscillator than a three-stage amplifier. The lower the average collector current, the higher the impedance of the circuit – and the internal capacitances of the transistors have a stronger effect with increasing impedance. This is why the time delay is greater with a lower supply voltage, resulting in a lower oscillation frequency.

A circuit of this sort consists of a ring of individual amplifier stages, which is why it is called a ring oscillator. The same effect can also be achieved with five, seven or nine stages. The only condition that has to be satisfied is that there is negative DC feedback. By contrast, with an even number of stages the result will always be a static flip-flop.

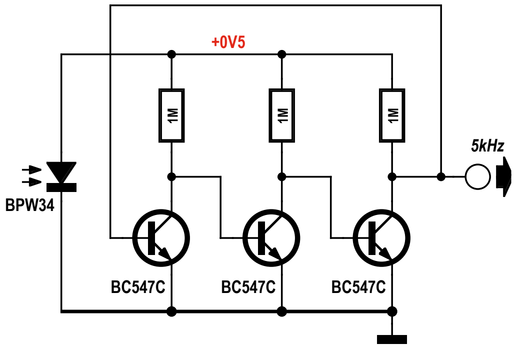


Figure 5. A ring oscillator powered by a solar cell.

A three-stage ring oscillator can be operated with very high resistance values and therefore very low power consumption. With three 1-MΩ collector resistors, the oscillator operates with a supply voltage as low as 0.5 V and consumes less than 1 µA. This means that a BPW34 photodiode in the sun, acting as a miniature solar cell, can provide enough power to operate the oscillator (**Figure 5**). The frequency of the output signal is approximately 5 kHz. The frequency rises with increasing light level, so you might be able to put the circuit to good use as a light sensor.

You may be wondering how this circuit can oscillate at just 5 kHz, entirely without capacitors. This seems strange, considering that the internal capacitance of a transistor is only a few picofarads. The answer to this puzzle is what is called the Miller effect (see inset), which causes the capacitance seen at the input to be the product of the collector–base capacitance and the voltage gain. Once you know this, you can easily connect additional capacitors between the collector and base leads to generate very low frequencies (**Figure 6**). With three 100-nF capacitors, the output frequency is approximately 1 Hz.

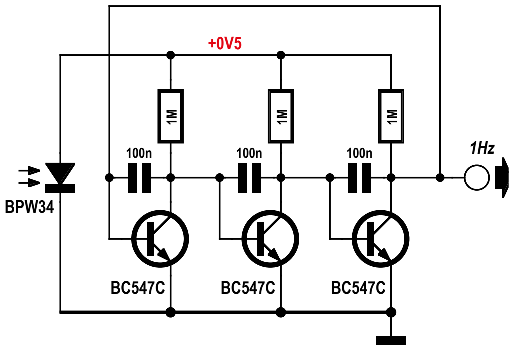


Figure 6. A lower-frequency ring oscillator with less power consumption.

Three-phase LED blinker

Attractive lighting effects can be generated with such low frequencies. The aim of the circuit shown in **Figure 7** is to use three LEDs to generate a pleasant flickering effect. This is a three-phase oscillator in which each of the three LEDs lights up in a different phase. The LED current is approximately sinusoidal, resulting in gentle transitions.

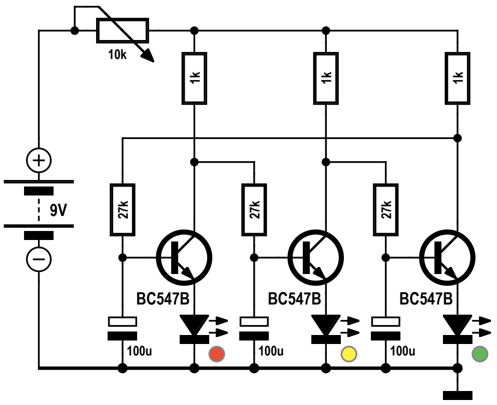


Figure 7. A three-phase lighting effect generator.

Depending on whether you connect the circuit directly to the 9 V supply voltage or use the potentiometer to reduce the operating current, the light is bright and flickers quickly or is less bright and flickers more slowly. Here again the frequency is highly dependent on the operating current.

The Miller effect

The voltage gain of a common-emitter amplifier stage is typically around 100. This holds true up to fairly high frequencies, but sometimes not as high as you might wish. Although the unity gain frequency of the BC547 is approximately 300 MHz (the current gain drops to 1 at 300 MHz), the upper limit frequency of this amplifier circuit is much lower, especially if the circuit is designed with fairly high resistance values. The culprit here is the internal junction capacitances of the transistor.

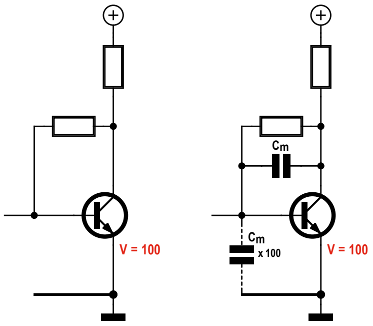


Figure 8. The Miller effect.

The base–collector capacitance Cbc has an especially strong influence, even though it is only around 5 pF with a BC547. This is due to the Miller effect. The Miller capacitance Cm (i.e. Cbc) between the input and the output of the inverting amplifier is charged and discharged from two sides. For example, if the base voltage rises by 1 mV, the collector voltage simultaneously drops by 100 mV. This means that 100 times as much charge must be supplied. The net effect is that there appears to be a capacitor connected to the input with a value equal to the Miller capacitance multiplied by the voltage gain, which in this case would be around 500 pF. The combination of this capacitance and the internal resistance of the connected signal source forms a low-pass filter that drastically reduces the upper limit of the amplifier bandwidth.

For an amplifier this means that if wide bandwidth is important, you should keep the circuit resistances as low as possible. In addition, in some cases it can be worthwhile to work with lower voltage gain, for example by reducing the output impedance. Another good option is to use special HF transistors with much lower junction capacitance.

In the case of oscillators, the Miller capacitance allows us to build oscillators without using capacitors to determine the frequency, since the transistor itself provides the necessary capacitance.

Three-phase blinker

Three-phase signals can also be generated very easily with a microcontroller. This requires a total of six switching points with the same time spacing. The result with three LEDs is similar to that with a three-phase ring oscillator, but distinctly more digital.

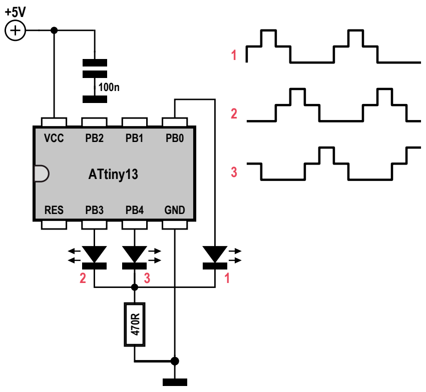


Figure 9. Three-phase blinker.

A special feature of this circuit is that the three LEDs share a single series resistor. As a result, each LED operates at two brightness levels. There are three On states for each individual LED. In the first state its lights up together with the LED to its left and shares the operating current with the other LED. In the third state it does the same, but with the LED to its right. By contrast, in the middle state it lights up by itself and therefore receives the full current. Each LED accordingly passes through the following sequence of states: half on, full on, half on, off, off, off, and in each case with a phase difference of 120 degrees relative to its neighbour LED.
If you are new to AVR Basic, get a copy of Elementary Course BASCOM-AVR, see the online book store at www.elektor.com

```
'Dreiphasen-Blinker 1500 ms, 0,67 Hz
$regfile = "attiny13.dat"
$crystal = 1200000
Config Portb = Output

Do
  Portb.0 = 1
  Waitms 250
  Portb.3 = 0
  Waitms 250
  Portb.4 = 1
  Waitms 250
  Portb.0 = 0
  Waitms 250
  Portb.3 = 1
  Waitms 250
  Portb.4 = 0
  Waitms 250
Loop

End
```

Listing 1.

Chapter 10 • Electronics for Starters (10)

Radio Frequency (RF)

It all started with high frequencies: radio technology was the driving force of electronics. At first there were electron tubes and AM radio, later followed by VHF and FM. Now we work with semiconductor devices, and the frequencies just keep on rising. Nevertheless, RF projects for the medium- and short-wave bands are still extremely interesting, especially for starters.

Building you own radio is still a good way to get started in electronics. However, your first radio project should be very simple, with no special components, and as foolproof as possible. The wideband regenerative receiver shown in **Figure 1** fits the bill. A regenerative receiver is a circuit that amplifies HF signals and demodulates them at the same time. With this extremely simple receiver you can listen to everything on the ether, whether it be a medium-wave broadcasting station or a very distant short-wave transmitter – and of course all possible types of HF noise in the near vicinity. The only prerequisite is a sufficiently long wire antenna. Around 30 feet of wire in the open air is ideal, but even shorter antennas can still yield good results. You also need an earth connection. A sensitive headset can be connected to the audio output, or better yet, a loudspeaker amplifier.

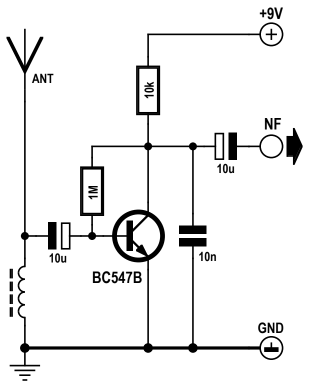


Figure 1. A wideband regenerative receiver.

The wideband regenerative receiver picks up everything at the same time. You will usually hear several transmitter signals, although most of the time one of them will be distinctly louder. The frequency range of the receiver is partially determined by the coil you use. If you wish, you can also use a fixed inductor. For medium wave reception the coil should have an inductance of around 300 µH, and for short wave reception approximately 10 µH, or as little as 3 µH if you want to listen to the higher-frequency bands in the 15 to 17 MHz range. A DIY coil for medium wave reception could consist of 100 turns on a ferrite rod. For short wave reception a coil with 10 to 30 turns and a diameter of 10 mm, without a ferrite core, is enough. However, the results also depend on the antenna.

It's a good idea to try several types of coil or inductor. A wide variety of stations can be heard, depending on the receiving site, the time of day and the coil that you use. For example, you may sometimes be able to listen to Radio China with this simple receiver, since the range is especially great in the higher-frequency short wave bands.

If you take a good look at the circuit, you will see that it consists of a single amplifier stage. This means that the high frequency signal from the antenna is amplified. With a truly ideal amplifier you wouldn't be able to hear anything, since the signal frequencies are far above the audible range. The reason that you do hear something is that the transistor has a non-linear input characteristic. As a result, the positive half-waves of the HF signal increase the collector current more than the negative half-waves reduce the current. This means that the average current varies according to the amplitude of the HF input signal. And because the transmitter modulates the amplitude of the HF signal at the frequency of the audio signal, the demodulated audio signal appears at the output of the receiver.

The capacitors in the circuit are also important. The relatively large capacitor at the input shorts audio signals on the base of the transistor to ground via the input coil, so that only HF signals are present at the base. The current in the collector circuit consists of a combination of the amplified HF signal and the audio signal. The 10 nF capacitor at the output shorts the HF signal to ground, leaving the audible AC signal in the audio range. The end result is a simple, low-cost receiver with fairly good sensitivity.

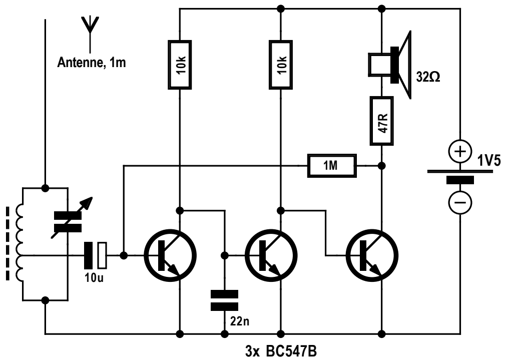


Figure 2. A regenerative receiver with higher gain and better selectivity.

Figure 2 shows a simple radio with the same sort of receiver stage, but with more gain and better selectivity. It has a tunable resonant circuit at the input. Depending on the coil characteristics, this radio can receive signals over a range extending from the long-wave band up to the upper short-wave region. The receiver coil is tapped at approximately 10% of its length. This keeps the damping factor of the resonant circuit relatively low, which means it has a good Q (quality) factor (see the 'Resonant Circuits' inset). This tuned circuit allows a specific broadcast signal to be selected. With a total of three transistors and a supply voltage of just 1.5 V, this circuit works well as a headphone receiver. Incidentally, this circuit has a certain similarity to the TAPIR HF Sniffer published in the July & August 2012 issue of Elektor, which also has three transistors but operates as a wideband regenerative receiver.

Generating HF signals

The world's first transmitter was a spark-gap device. To see how it works, all you need is an ordinary light switch. When you flip the switch, you can hear a bit of static on the radio — at least if you're listening to the medium wave band and you aren't receiving a signal from a strong AM transmitter. A pulse waveform with steep edges always has signal components in the RF range. In fact, in many places switch-mode power supplies and other electrical devices are a major source of radio interference, especially in the medium wave band. This means that the difficulty is not generating RF signals, but instead preventing the radiation of undesirable RF signals.

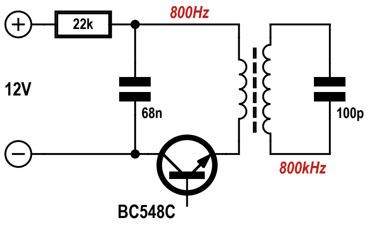


Figure 3. A tuned 'spark gap' transmitter.

You don't really need a high-power transmitter suitable for a pirate radio station, but a signal generator that produces just a wee bit of RF power can come in handy now and then, if only for testing your own receivers. **Figure 3** shows a tiny transmitter circuit that works on the same principle as the old-fashioned spark-gap transmitter. In this case the periodic pulses at a frequency of around 800 kHz result from switching an NPN transistor in an astable multivibrator circuit (see instalment #7 of this series in the September 2012 issue of Elektor). Each steep pulse edge excites oscillations in the resonant circuit, which then quickly decay as damped oscillations. The transmit frequency is determined by the resonant circuit. For example, if you use a ferrite rod antenna from an old radio, the frequency will be somewhere in the medium wave band, such as 800 kHz. You will hear a humming sound if you hold a radio close to the ferrite rod. All you need now for a full-fledged transmitter is a Morse key. The shipboard transmitters of days gone by worked in a similar manner, but with considerably more power and at much lower frequencies.

LC oscillators

Spark-gap oscillators could only transmit damped oscillations. The first real progress came with the invention of electron tubes, which were able to generate undamped oscillations. In the previous instalment of this series, which dealt with sine wave oscillators, we described what you need for this: enough gain and suitable feedback.

The grandfather of all RF oscillators is the Armstrong or Meissner oscillator, in which a coupling coil provides feedback with the right phase (**Figure 4**). The first radio transmitters were also built this way: take a tube and a resonant circuit and add a bit of feedback, and you've got an RF power oscillator. Modulation was handled very simply by fitting a carbon microphone in the antenna lead (!).

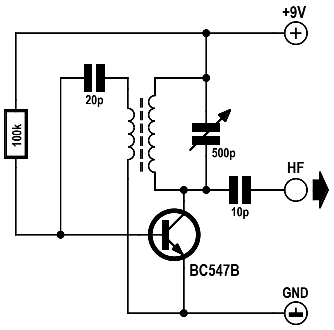


Figure 4. A sine wave oscillator with inductive coupling.

The circuit diagram in **Figure 4** shows a single-transistor Armstrong oscillator. It's not difficult to get this circuit to oscillate. If you have problems, the phase may be wrong, in which case all you need to do is to reverse leads of the coupling coil.

The frequency of this simple oscillator can be adjusted with the variable capacitor. The amplitude depends on a lot of factors, including the coil damping, the feedback winding, and the coupling capacitor. Sometimes it takes a bit of experimentation before you get stable oscillation, since this circuit has a peculiar problem: it is prone to intermittent oscillation if the amplitude is too great. This is caused by rectification of the HF signal by the base–emitter diode, which drives the base negative until the transistor is cut off and oscillation suddenly stops. After a few microseconds the base resistor restores the normal operating point and the cycle starts over again.

This problem does not occur with the oscillator shown in **Figure 5,** since it has automatic amplitude stabilization. Furthermore, this circuit only needs a simple coil without a coupling winding or a tap, and it the coil doesn't even need to have an especially high Q factor, thanks to the high gain. This circuit is therefore a good choice for DIY construction, and it always oscillates reliably. Here the feedback is provided by a pair of transistors. Both transistors also operate at a very low voltage (approximately 0.6 V). The circuit can operate with a supply voltage as low as 1 V and can also operate with very low current consumption. This can be advantageous for some applications.

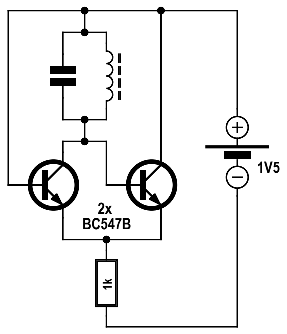


Figure 5. An oscillator for use with low supply voltage.

The basic circuit can operate over a wide frequency range extending from audio to UHF, depending on the resonant circuit, making it ideal for building small test oscillators or the like with little effort or expense. However, it does have a drawback: the internal capacitances of the transistors affect the frequency, and they also depend on the operating voltage. This means that although the circuit is very simple, it's not exactly a shining example of frequency stability.

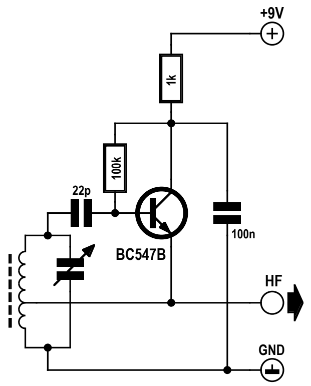


Figure 6. An oscillator with a tapped coil.

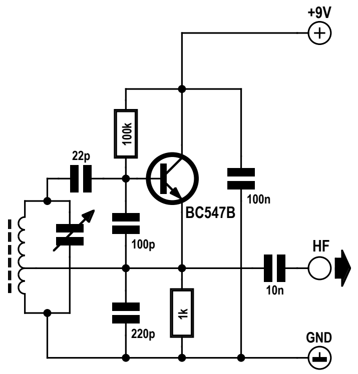


Figure 7. An oscillator with a capacitive voltage divider.

However, direct coupling to the resonant circuit affects the frequency in this circuit as well, since the voltage divider capacitances vary with the operating voltage. This simple oscillator also highlights a stability problem that may be encountered with amplification stages using a common-collector configuration. Although your intention may simply be to use an emitter follower as an impedance converter, it can inadvertently turn out to be an oscillator. Whether the circuit acts as an amplifier or as an oscillator depends on several factors, including the operating point and the damping of the resonant circuit.

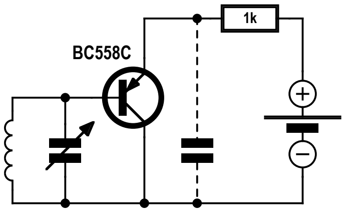


Figure 8. A DC coupled oscillator in common-collector configuration.

Crystal oscillators
If you need a really stable frequency, you should use a crystal oscillator. Quartz crystals have essentially the same characteristics as a resonant circuit with an extremely high Q factor. They can therefore be used to build relatively simple oscillators with good frequency stability. A typical circuit is shown in **Figure 9**.

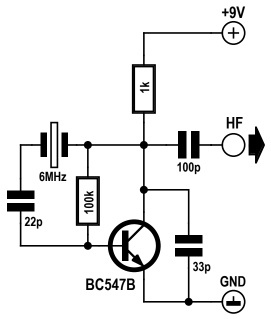


Figure 9. A crystal oscillator.

If you like to experiment with DIY medium-wave radios, you may wish to have a modulated oscillator for test purposes. A test transmitter of this sort is not connected to an antenna and therefore does not disturb your neighbours. Instead, the RF energy is transferred directly from one coil to the other.

However, the oscillator needs to be stable, as otherwise there’s little point to such a modulated signal generator for medium-wave frequencies, but crystals for these relatively low frequencies are very expensive. For this reason, ceramic resonators are often used in such cases. Resonators with rated frequencies of 500 kHz or 2 MHz are commonly available, but these frequencies are outside the medium wave band. However, resonators are also available with frequencies that fit better. For example, we found a ceramic resonator with a frequency of 986 kHz in a discarded television remote control.

Figure 10 shows the circuit of a complete AM test oscillator. The trimmer can be used for fine tuning. If a broadcast station (most likely weak) can still be heard in the background, you can simply tune the oscillator to a beat frequency that neutralizes the station, such as 981 kHz. The small ferrite coil in the transmitter couples directly to the ferrite rod in the receiver. Incidentally, with relatively minor alterations you can also use this circuit in the short wave region, for example at a frequency of 13.56 MHz, which can be used for experimental purposes without a license.

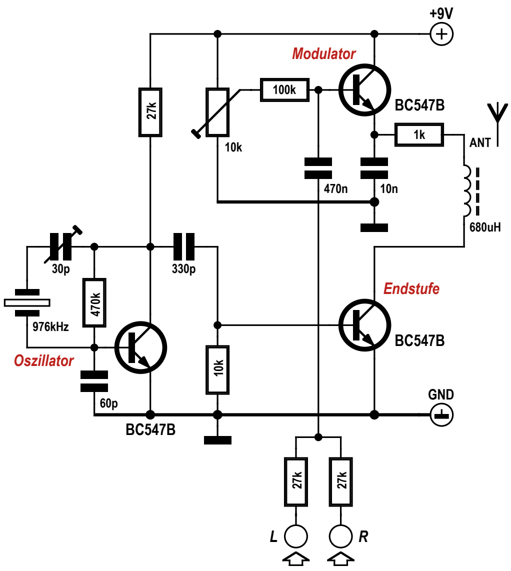


Figure 10. A modulated medium-wave AM signal generator.

The modulator stage is designed as an emitter follower that modulates the supply voltage of the output amplifier stage. As monaural transmission is still standard in the medium- and short-wave bands, the signals from the left and right stereo channels are combined at the audio input. The potentiometer should be adjusted for the lowest distortion and the best sound quality. The RF amplifier stage has been intentionally kept modest, since high power is not the objective here.

Short wave regenerative receiver with feedback

The three-stage short wave receiver shown in **Figure 11** has an additional control for the feedback. The first stage is essentially an oscillator corresponding to the circuit in Figure 8. However, its operating point can be adjusted as desired to vary the gain. Here the trick is to adjust the gain to just balance out the losses of the resonant circuit, so that the circuit is just on the edge of oscillation. At this point the receiver has the highest gain and the best selectivity. The PNP oscillator stage in the common-collector configuration is also the regenerative stage that demodulates the HF signal. The two following audio stages generate enough power to allow a small loudspeaker to be used.

Whether the regenerative stage can be adjusted to the edge of oscillation depends in part on how much the input circuit is damped by the connected antenna. For this reason there are two options for connecting the antenna. Connection through a small coupling capacitor results in loose coupling and low damping. By contrast, direct connection is suitable for very short antennas, since a relatively long antenna also radiates HF energy, thereby damping the resonant circuit. With the right setting, extremely high sensitivity can be achieved with a regenerative receiver. Such receivers were standard equipment in the early days of radio engineering. Even with weak transmitters, they achieved ranges of several thousand miles. You can still experience the fascination of this circuit today, since it allows outstanding receivers to be built with very low cost and effort.

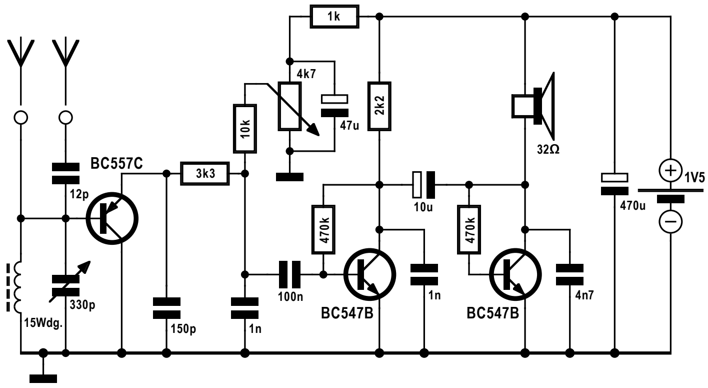


Figure 11. A regenerative receiver with feedback.

This brings us to the end of the *Electronics for Starters series*. However, we intend to continue presenting articles on basic aspects of electronics at irregular intervals in upcoming issues of the magazine. Some of the topics on the agenda are operational amplifiers and commonly used digital ICs. Stay tuned for more!

Resonant Circuits

If you combine an inductance (L) with a capacitance (C) you get a resonant circuit. In a manner similar to mechanical energy in a pendulum, electrical energy can flow back and forth between the inductor and the capacitor at a specific frequency (f), which is called the resonant frequency. A resonant circuit oscillates after being excited by any short pulse of electricity. An ideal resonant circuit would have no losses and would therefore oscillate forever after being excited.

However, in practice the oscillation is damped because energy is dissipated in the form of heat due to the resistance of coil winding, magnetic losses in the core of the inductor, and electromagnetic radiation. For the sake of simplicity all of these losses can be collectively represented by a resistance R in parallel with the circuit.

The resonant frequency f is given by the formula:

f = 1 / (2 p √ LC)

A quality factor (Q factor) can be specified for every resonant circuit. Q is determined by the ratio of the parallel damping resistance R to the inductive impedance ZL = 2 √ fL or the capacitive impedance ZC = 1 / (2 √ fC) at the resonant frequency, at which the impedance is resistive:

Q = R / ZL or Q = R / ZC

If a resonant circuit is excited by a variable-frequency signal from an AC signal generator with high internal impedance, the voltage across the resonant circuit is the highest at the resonant frequency. The amplitude of the voltage at the resonant frequency is directly proportional to the Q factor, or in other words, the lower the damping due to energy dissipation of any form, the higher the resonant voltage. At either side of the resonant frequency it is possible to find points on the resonance curve where the voltage is reduced by a factor of 0.707 (1 / √2), or 3 dB down from the maximum. The frequency difference between these two points is designated the bandwidth b of the resonant circuit. The relationship between the resonant frequency f, the bandwidth b and the Q factor of the circuit is given by the formula b = f / Q.

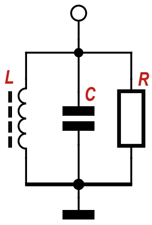


Figure 12. A damped resonant circuit.

With careful coil construction it is usually possible to attain Q factors up 100 or so. However, resonant circuits are also damped by the connected circuit or connected antenna. This damping can be countered by using a small coupling winding, a coil tap or a suitable capacitor to achieve loose coupling. If the resonant circuit is connected directly to a gain stage, this stage should have high internal impedance to keep the damping small.

An AM signal generator

Anyone who designs or builds radio receivers can make good use of a small AM signal generator. With a suitable signal waveform, you don't even need to be able to adjust the frequency. This works if the generator operates at an adequately low frequency and generates enough harmonics. The ATtiny13 AM signal generator described here produces short pulses at a frequency of 70 kHz. This results in strong harmonics covering the entire long-wave and medium-wave spectrum. The pulse train is also briefly interrupted at regular intervals to produce amplitude modulation at a frequency of 750 Hz. A radio can therefore receive an AM test signal at 70 kHz, 140 kHz, 210 kHz, etc. and demodulate it to produce a humming tone.

A wire loop with a diameter of about 4 inches (10 cm) or so makes a suitable antenna. It generates an AC magnetic field that couples directly to the ferrite rod of the receiver. The signal generator can also be used for comparative sensitive measurements by testing how far away from the generator the signal can still be heard. A good receiver can receive a clear signal at distances up to 3 feet.

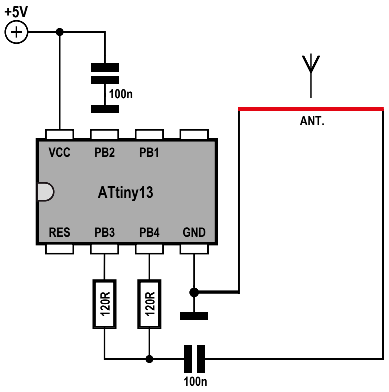


Figure 13. An AM signal generator.

The source code can be downloaded from the web page for this article.

```
'ATtiny13 AM Generator
$regfile = "attiny13.dat"
$crystal = 1200000
$hwstack = 8
$swstack = 4
$framesize = 4

Config Portb = Output
Dim N As Byte

Do
  For N = 1 To 50   '70 kHz
    Portb = 255
    Portb = 0
  Next N

  For N = 1 To 50   'AM 750 Hz
    nop
    nop
  Next N
Loop

End
```

Listing 1.

Internet Links

[1] www.elektormagazine.com/magazine/elektor-201212/20017

Chapter 11 • Operational Amplifiers in Practice

Part 1: Introduction and basics

Operational amplifiers are ubiquitous in electronics, being used in a wide range of applications in all sorts of configurations. There are numerous device families and types available, each with their own strengths and weaknesses. So here we will take a closer look at the details.

Operational amplifiers, or 'opamps', are integrated circuits built from bipolar transistors, JFETs or MOS transistors. The name comes from their original application in analogue computers, where they act as amplifiers in circuits carrying out operations such as addition, multiplication and so on.

Basic opamp circuits

An operational amplifier amplifies the voltage difference between its two inputs. An ideal operational amplifier has an infinite voltage gain; real devices, on the other hand, achieve gains of up to around 100 000. In most cases negative feedback is applied, and it is the nature of this feedback that determines the behaviour of the circuit. **Figure 1** shows an opamp with direct negative feedback, and the voltage gain of this circuit is exactly 1 (unity). However, there is considerable current gain: the input to the circuit has a high impedance, but the output can drive a relatively low-impedance load. The circuit thus acts as a buffer amplifier.

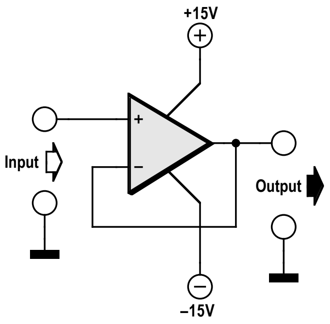


Figure 1. An opamp as a buffer amplifier.

Voltage follower

The operation of the circuit can be thought of as a control loop. The opamp continuously compares the 'set point' (at the non-inverting, or '+', input) with the actual voltage (at both the output and at the inverting, or '–', input), compensating for even a tiny difference. In practice there is usually a small residual constant difference between the inputs, called the 'offset voltage', typically of the order of 1 mV. Some devices, including the LM741, allow the offset to be adjusted to zero.

In the past it was usual to power opamps from a symmetrical supply, frequently +15 V and –15 V. The range of allowable input voltages might then be from at least –10 V to +10 V. This convention also dates from the time of analogue computers, and these days a lower supply voltage is normally used.

Non-inverting amplifier
An opamp can also be used to amplify an input voltage by a precise factor. To do this, a voltage divider is used in the feedback network: see **Figure 2**. Now, the device automatically sets its output voltage so that the voltages at the inverting input and the non-inverting input are practically the same; any small deviation from this will cause a suitably large change in the output voltage so that the negative feedback exactly cancels it out. The residual difference between the input voltages (the offset) does not change significantly because of the high current gain.

When designing a circuit the first step is to consider an ideal opamp with zero input offset voltage. This ideal component also has an infinite input impedance, zero output impedance, and an infinite bandwidth. For many applications a standard opamp will approximate this ideal remarkably well.

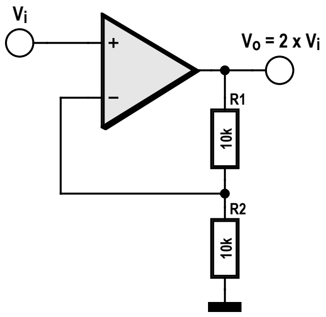


Figure 2. A voltage gain of 2.

In theory any desired gain can be achieved by a suitable choice of resistors in the feedback network. The voltage gain is equal to the division ratio produced by the voltage divider, and so the configuration of Figure 2 provides a gain of 2. In analogue computers, an opamp circuit like this was used to achieve precise values of gain, which corresponds to the mathematical operation of multiplication by a constant factor G, where

$$G = (R1 + R2) / R2.$$

It is worth noting, particularly at high gain values, that the offset voltage is also multiplied by the gain factor. In some devices (for example the trusty LM741) dedicated connections are provided to which an offset adjustment circuit can be connected. Other types, including the OP07, are trimmed during manufacture to achieve an offset error of just a few microvolts.

Inverting amplifier
An opamp can be configured to invert its input voltage exactly: see **Figure 3**. The non-inverting input of the opamp is connected to ground, which means that the voltage on the inverting input will also be zero. We therefore connect two equal resistors such that when the input voltage to the circuit is +1 V and the output voltage is –1 V, the voltage at the inverting input to the opamp is zero. By changing the ratio of the two resistors we can achieve any desired (negative) gain value: in general,

G = –R2 / R1.

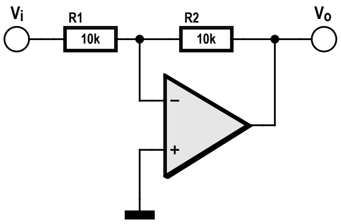


Figure 3. An inverting amplifier.

Adder

Often we want to add several voltages together. This can be achieved using the inverting amplifier configuration with more than one input resistor: see **Figure 4**. The inverting action of the amplifier can be undone by following it with a second inverting amplifier circuit.

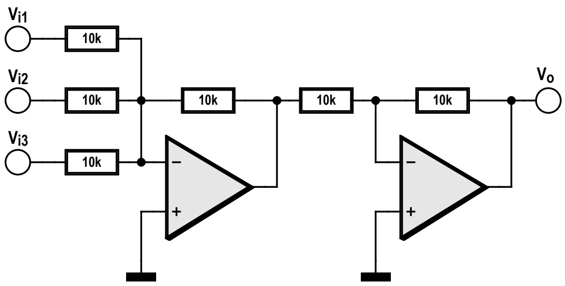


Figure 4. A three-input adder circuit.

Oscillator

In a circuit designed to act as an amplifier we only find negative feedback. Oscillator circuits, on the other hand, also employ positive feedback. **Figure 5** shows a squarewave generator with feedback into the non-inverting input; there is also negative feedback but this is slowed down using an RC network, whose values determine the output frequency.

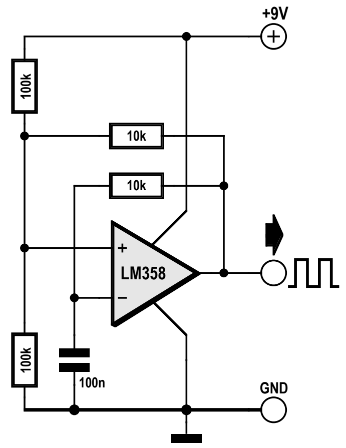


Figure 5. A squarewave generator.

Under the hood

Inside, an opamp consists of a differential amplifier and an output stage. A comparable circuit can be realised using discrete transistors, as shown in **Figure 6**, and this gives us a good way to help understand the typical characteristics, strengths and weaknesses of an opamp.

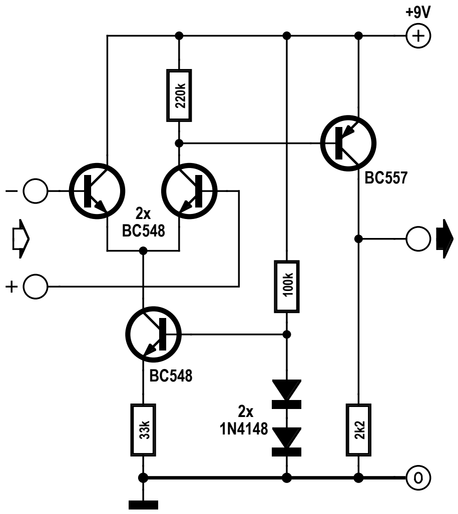


Figure 6. An opamp constructed from discrete components.

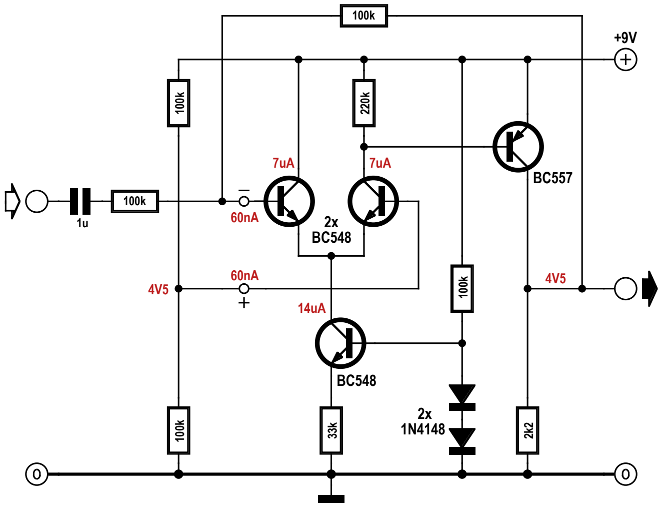


Figure 7. An inverting amplifier under test.

We can make measurements of internal and external voltages in a real application using negative feedback to collect data about the operation of our discrete opamp (**Figure 7**). The current source for the two input transistors delivers a current of 14 µA, which is approximately equally split between the them. The quiescent input current is 60 nA; in the LM741, for comparison, this figure is 10 nA. The offset voltage is 5 mV (LM741: 1 mV). We can also measure the open-loop gain of the circuit. To do this we apply an input voltage of 2 VPP; then we obtain 2 VPP at the output, but with a phase shift, and we can measure 50 mVPP at the inverting input. This means that the open-loop gain is 40: nothing to write home about, as even the humble LM741 manages an open-loop gain of 100 000. But there is one respect in which our discrete opamp performs better: its output can swing from rail to rail, delivering up to 9 VPP.

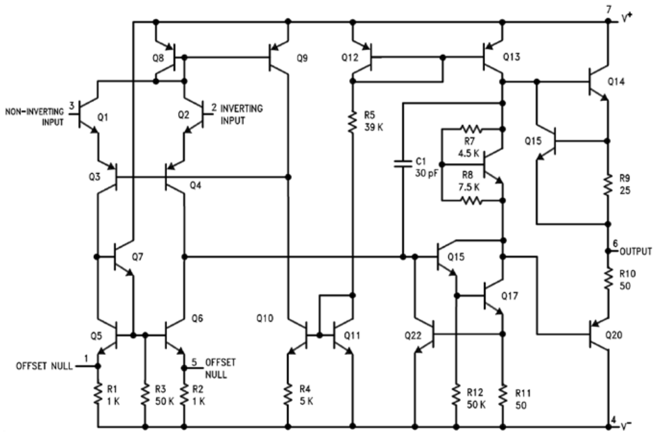


Figure 8. Internal circuit diagram of the venerable LM741 (source: Texas Instruments).

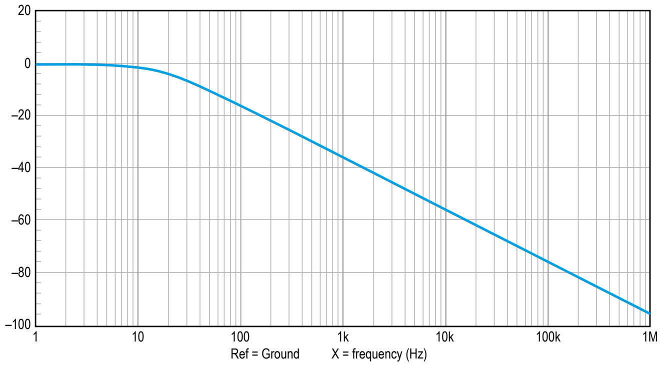


Figure 9. Open-loop gain as a function of frequency.

Most opamp manufacturers present a simplified internal circuit diagram in their datasheets, giving only important details relevant to the use of the device. **Figure 8** shows the internal circuit of the LM741, and the similarities to our discrete design, in particular around the NPN input transistors, are clear. These transistors form a differential amplifier, whose emitter currents can be externally adjusted to a small extent to achieve optimum symmetry, and hence reduce the offset voltage to zero. The signal is taken via a current mirror to an intermediate amplifier stage, which in turn drives a push-pull output stage. A single capacitor serves to reduce the internal gain-bandwidth product to 1 MHz. Although the open-loop gain is around 100 000 at frequencies below 10 Hz, it falls off to unity at 1 MHz. **Figure 9** plots the relationship between frequency and open-loop gain on logarithmic axes. This fall-off in gain with increasing frequency is necessary to maintain adequate stability under all operating conditions. However, it also means that a jellybean part like this is not suitable for use at very high frequencies.

In the second instalment of this mini-series we will look at a different kind of opamp that uses field-effect transistors in the input stage, and some of the applications where its properties come in handy.

Chapter 12 • Operational Amplifiers in Practice

Part 2: FET-input opamps and wide-bandwidth applications
In many applications it is essential to use an opamp with extremely high impedance inputs. In these cases a standard bipolar device will not do the job, and instead we must use an opamp with FET (field-effect transistor) inputs.

The input leakage current of a typical LM358-type opamp is around 30 nA. If an input resistor of 1 MΩ is used, then this will have a voltage drop of 30 mV across it, which in many circuits is not acceptable. Opamps with FET inputs, on the other hand, have practically zero input leakage current: for example, a JFET-input opamp might have an input current some 1000 times lower than that of its bipolar brother. Typical examples of JFET-input opamps are the TL071 (single), TL072 (dual) and TL074 (quad). These devices use JFETs in the input stage and bipolar transistors for the rest of their circuitry, as **Figure 1** illustrates.

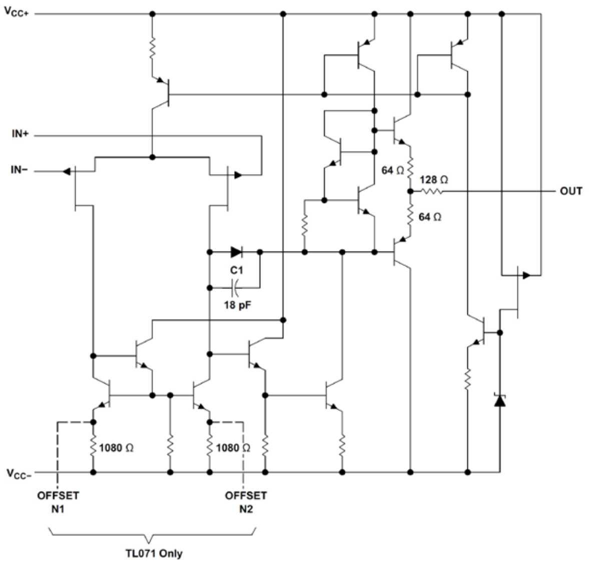


Figure 1. Circuit diagram of the TL071 (Source: Texas Instruments).

Even better input isolation can be obtained by using MOSFETs in the input stages. One example of this technology is the CA3140 (**Figure 2**). This device can run from a single or symmetrical supply of at least 4 V, and the input voltages can swing down to 0.5 V below the negative supply rail. The MOSFETs are very sensitive to excessively high voltages, and so Zener diodes are included in the circuit for protection. Input currents can be as low as 2 pA. The output stage of the CA3140 is conventionally constructed using bipolar transistors, and so the manufacturer calls the device a ‘BiMOS’ amplifier. As well as the single version, there is also a dual version, the CA3240.

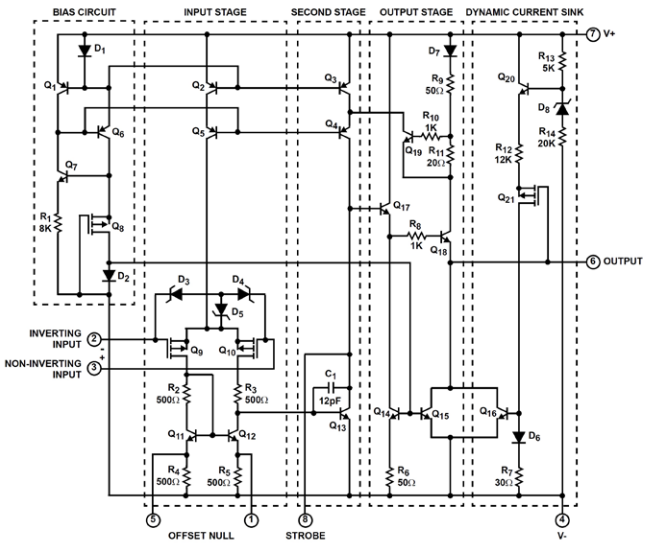


Figure 2. Circuit diagram of the CA3140 (Source: Renesas)..

Yet further improvement is brought by the CA3160 which uses CMOS transistors also in its output stage. This allows output voltage swings to within 10 mV of the negative and positive supplies ('rail-to-rail operation').

Voltage ramps

A FET-input opamp is a good choice when it is desired to generate slowly-varying waveforms such as voltage ramps. **Figure 3** shows a typical circuit of a ramp generator, which can, for example, be used to help plot characteristic curves automatically. The opamp is wired as an integrator, and the slope of the ramp is determined by the low charging current that flows towards the inverting input. The high input impedance of the opamp means that its input leakage current does not affect the operation of the circuit. The switch across the integration capacitor allows it to be discharged to begin a new ramp.

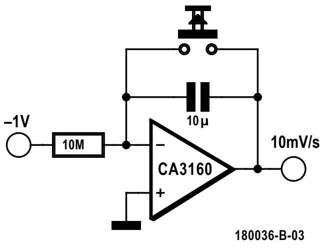


Figure 3. A ramp generator.

In some situations an exponential ramp is required rather than a linear one. An example of how to achieve this is shown in **Figure 4**: this circuit could be used, for example, to drive a tone generator for audio testing purposes. The charging current for the capacitor is generated here from a voltage which is in turn proportional to the capacitor voltage. The result is exponential growth of the output voltage (see **Figure 5**). To initialise the circuit the capacitor must be charged to a non-zero voltage, say 50 mV. Then the gain of 2 provided by the amplifier will result in 100 mV at its output. The high input impedance of the BiMOS opamp allows the component values in the circuit to be varied over a wide range.

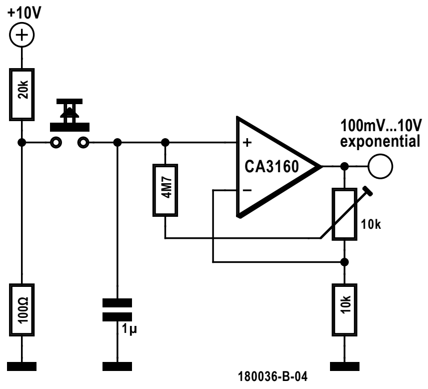


Figure 4. An exponential ramp generator.

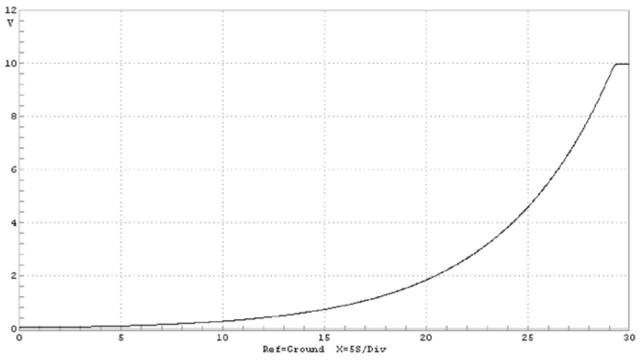


Figure 5. Output trace from the exponential ramp generator.

Many modern opamps are constructed entirely in CMOS technology and use no bipolar transistors at all. An example of this kind of device is the TLC272 CMOS opamp, which offers input leakage currents of less than a picoamp! **Figure 6** shows its internal circuit. Some CMOS opamps offer operation with both input and output voltages swinging all the way to the supply rails.

New opamps with ever better characteristics continue to be developed. Sometimes these improvements take the form of higher-impedance inputs or lower input offset voltages, or in true rail-to-rail operation at the inputs and outputs, or in the ability to drive low-impedance loads at the output. In many respects, therefore, we are getting closer and closer to the ideal opamp. However, it is often the case that obtaining particularly good characteristics in one area entails compromises on other points. So, for example, CMOS opamps with a very high input impedance often have a large input offset voltage, and parts optimised for low offset voltage only have a low bandwidth.

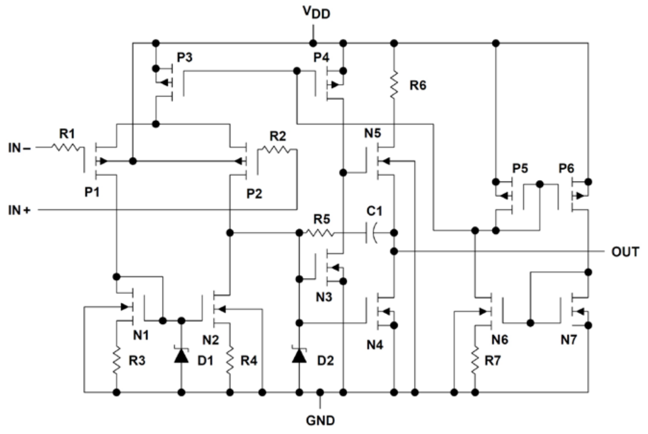


Figure 6. Circuit diagram of the TLC272 (Source: Texas Instruments).

An example from the new generation of universal opamps is the TS914. This offers rail-to-rail operation on both inputs and outputs. The circuit diagram (**Figure 7**) shows that the device has a complementary differential amplifier stage at the input and a push-pull output employing MOSFETs. The part is a good choice in many applications and is used, for example, in the Elektor SDR (software defined radio). It has an input leakage current of just 1 pA and an input offset voltage of 5 mV. Its bandwidth is 800 kHz and it can drive loads with impedances down to less than 100 Ω.

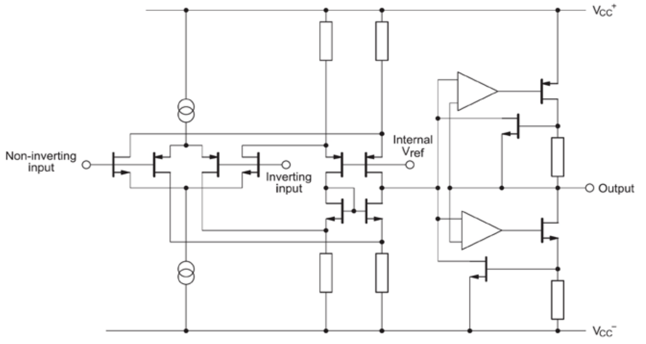


Figure 7. Circuit diagram of the TS914 (Source: STMicroelectronics).

Extremely high input-impedance opamps are often used in instrumentation amplifiers. As **Figure 8** shows, an instrumentation amplifier has a differential input and a single-ended output. It can therefore be used to make measurements on circuits without having to connect one side of the input to ground, although of course the voltage relative to ground on the inputs must lie within the permissible input voltage range of the opamp. The circuit comprises an adjustable differential amplifier with two impedance converters connected before it: this is how it achieves its extremely high input impedance. The differential amplifier should be adjusted to optimise the stability and common-mode rejection performance of the circuit.

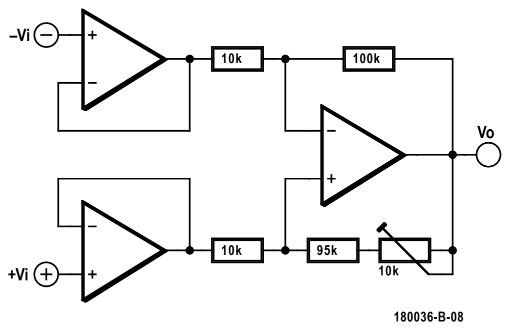


Figure 8. An instrumentation amplifier.

Wide-bandwidth applications
Although an opamp can basically be thought of as a DC amplifier, it is of course also suitable for use as an AC amplifier. One example is the microphone amplifier circuit of **Figure 9**. With a single supply voltage it is necessary to create an artificial supply between the rails, in this case at half the supply voltage. The circuit then behaves as if it had both a negative and a positive supply voltage relative to this intermediate rail.

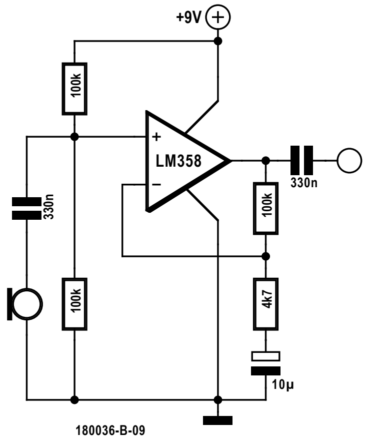


Figure 9. A microphone amplifier using an opamp.

Amplifier circuits like this are capable of operation over a broad frequency range, although sometimes it can come as quite a surprise how quickly a device can bump into its frequency limit. Suppose, for example, that we want an amplifier with a gain of 100 that can operate at up to 20 kHz. The gain-bandwidth product is thus 100 × 20 kHz, or 2 MHz. However, many devices have a gain-bandwidth product of only 1 MHz or even lower. That means that at 20 kHz we already do not have enough gain available, and the negative feedback network will not operate as hoped. The result is distortion as internal stages in the device are driven into saturation. The oscilloscope will show a sinewave signal being distorted into a triangle wave, because inside the device the saturated stages are generating square waves, whose fast edges are integrated in the output stages to uniform ramps.

The Elektor SDR shield uses a type TS914 quad opamp which offers a gain-bandwidth product of 0.8 MHz (see **Figure 10**). In order to increase the overall bandwidth of the circuit it is divided into two stages each with a gain of 10. These stages will thus each have a bandwidth of 80 kHz, which is sufficient for this application, where the signals are taken to the stereo inputs of a sound card for further processing by the SDR software. Depending on the sound card, these inputs will have a maximum bandwidth of 24 kHz or 48 kHz; by processing the two channels separately we can achieve an effective total reception bandwidth of 48 kHz or 96 kHz.

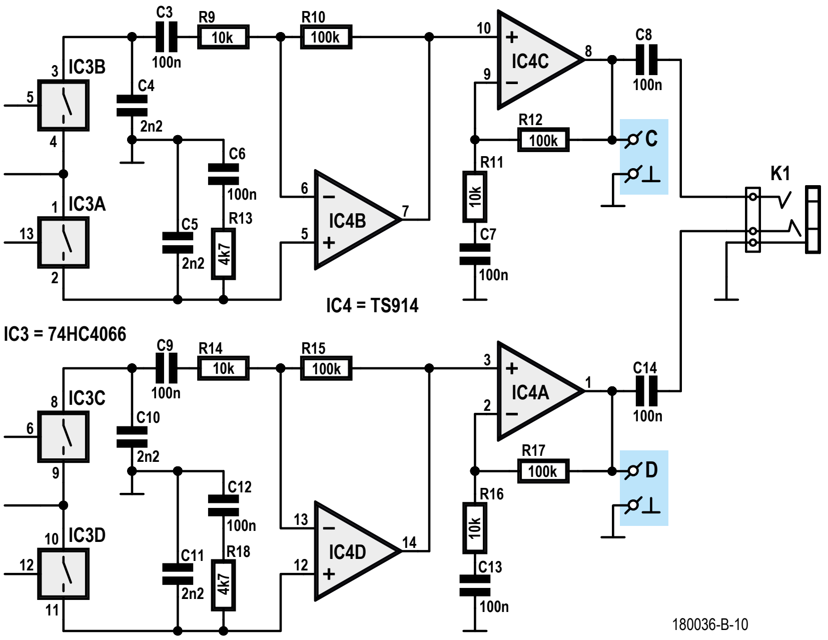


Figure 10. The TS914 as used in the Elektor SDR shield.

For standalone operation of the receiver the signal processing software running on the PC must be replaced by a purely electronic solution. One approach is to use the bandpass filter with a centre frequency of 750 Hz shown in **Figure 11**. The circuit has separate in-phase (I) and quadrature (Q) inputs: first a relative phase shift of 90 ° at 750 Hz is applied to these signals, and then they are added together. The result is passed through a two-stage 750 Hz bandpass filter. The design uses type LM348 dual opamps.

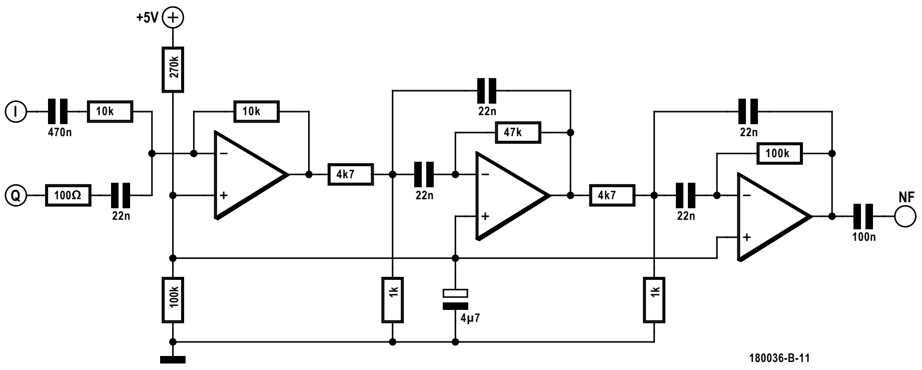


Figure 11. A bandpass filter with quadrature inputs.

The filter stages were simulated and tuned using LTspice. Be aware, however, that here again the limited bandwidth of the opamps makes a difference! It is easy to think that since all the signals involved are firmly in the audio frequency range there cannot possibly be a problem, and even the second filter stage, which has a gain of 100, demands a gain-bandwidth product of only 80 kHz for an input signal at 800 Hz. However, this overlooks the fact that many higher-frequency components will be present in the signal, and we do not want these to generate distortion. Things become even more critical if we want to construct filters with a high quality factor operating at frequencies of 10 kHz and above.

@ www.elektor.com

Chapter 13 • Operational Amplifiers in Practice

Part 3: Opamps with PNP input stage, and power types

The final instalment of the short series on opamp basics explores some of the more unusual features of operational amplifiers. Another topic is opamps with power outputs.

Most operational amplifiers operate within a limited voltage range at a certain level below the supply voltage. The voltage difference is caused by the current source, the base-emitter path and the current mirror in the input circuit. Some types are optimized by special input circuits to work right up to the negative supply voltage. These devices have an additional base-emitter path in the input differential amplifier, allowing input signals up to the negative supply voltage limit to be applied. For example, the dual opamp type LM358 (**Figure 1**) as well as the quad opamp type LM324 can be operated from a single supply voltage of +3 V; making it well suited for battery operation. By using PNP input stages, it is possible to achieve input voltages up to 300 mV below the negative supply rail. The output voltage, on the other hand, is not quite close to zero. It is therefore definitely worth knowing more about the characteristics of the inputs.

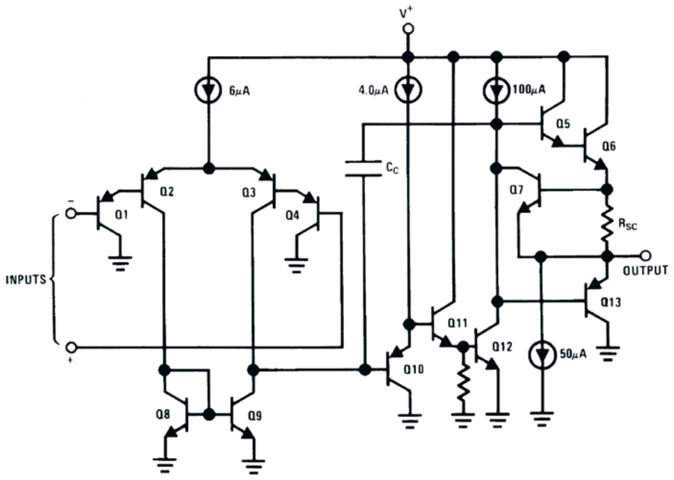


Figure 1: Internal circuit of the LM358 with PNP input stages (image: Texas Instruments).

The LM358 is something like the standard operational amplifier for all cases, it's extremely inexpensive due to mass production and adequate for most cases. If more than two opamps are needed in the circuit, the equally inexpensive LM324 should be on the shortlist.

In practice you should always dimension a circuit so that it works correctly with almost every operational amplifier. But it is also attractive to exploit special properties of a certain type in order to save components or to develop very special solutions. The circuit in **Figure 2** shows a simple light sensor with the green LED acting as a photodiode.

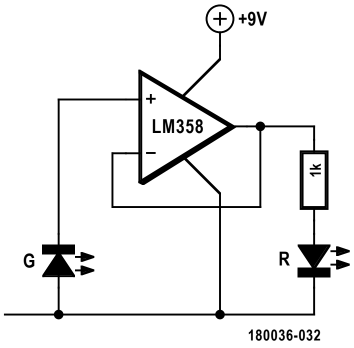


Figure 2: Using the input current.

The LED is operated in reverse direction, i.e. with the anode connected to ground. When light falls on the LED, a small photocurrent flows. The experiment does not work in this form with just any operational amplifier, but with the LM358 or an LM324 only. Due to their PNP input stages, these types deliver a small current (the basic quiescent current) of about 30 nA at each input. An open input is therefore 'raised'. When the brightness is low, the red LED lights up. As soon as sufficient light falls on the green sensor LED, it diverts the input current so that the input voltage drops. Therefore the opamp switches off the red LED if there is sufficient illumination. The input current is subject to relatively low variation and is hardly dependent on the operating voltage. Here it determines the brightness threshold at which switching takes place. If you wanted to build the circuit with a CMOS operational amplifier, you would have to provide the input current through a 330-MΩ resistor, which will be difficult if not impossible to find in an electronics store.

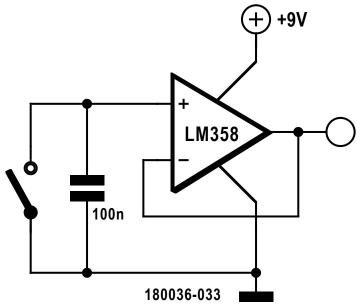


Figure 3: Simple ramp generator.

If a capacitor is connected to the input of an LM358, a very simple ramp generator is obtained (**Figure 3**). The virtually constant opamp input current slowly charges the capacitor. The exact input current can be determined from the slew rate (**Figure 4**). The ramp shows a slight bend. With a charge current of 30 nA, a voltage of 3 V is generated in the first ten seconds.

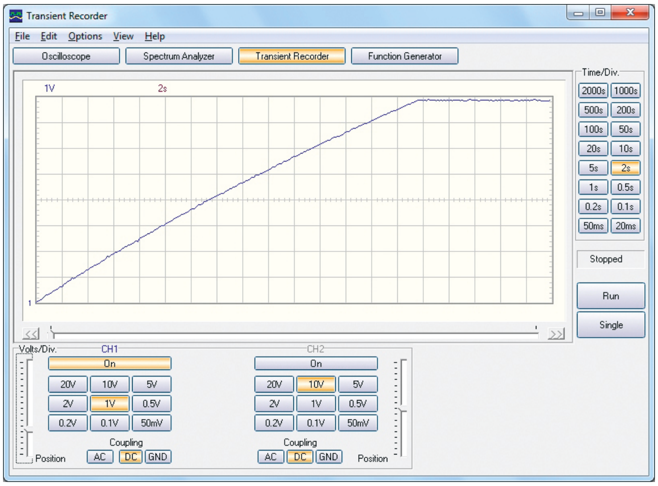


Figure 4: The voltage ramp at the output.

Comparator type LM339

The comparator type LM339 (**Figure 5**) also uses a very similar input stage. Additional diodes improve the behaviour in case of strong overload. It uses an emitter circuit with an open collector at the output, forcing an external collector resistance to be used. A further difference with a common operational amplifier is that no internal limiting of the frequency response is used. This results in faster switching, a higher operating frequency and steeper edges of the output signal. Stability problems do not occur with a comparator because it is practically always in an overdriven state in which a very small gain effectively prevails.

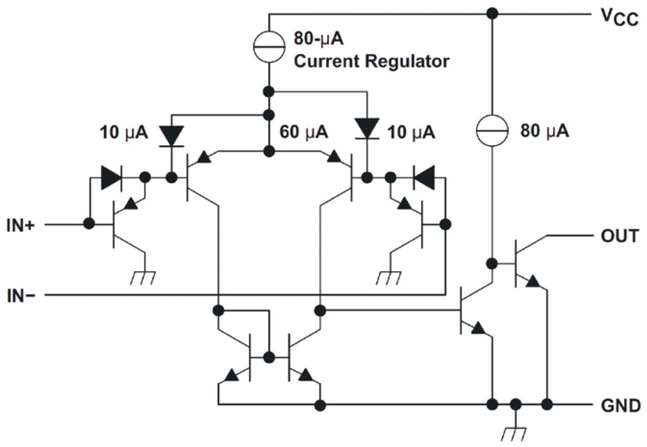


Figure 5: Internal circuit of the LM339 comparator (image: Texas Instruments).

You could attempt to use the comparator together with a load resistor against V+ as an operational amplifier. At higher gain, the comparator would behave like a normal opamp, but at full negative feedback, self-oscillations would occur. It is better if the comparator is restrained and delivers only squarewave signals at the output. **Figure 6** shows the typical case in which a sinusoidal signal or any other signal form is converted into rectangles. An LM358 could also be used for this task, but the real comparator reaches a higher cutoff frequency.

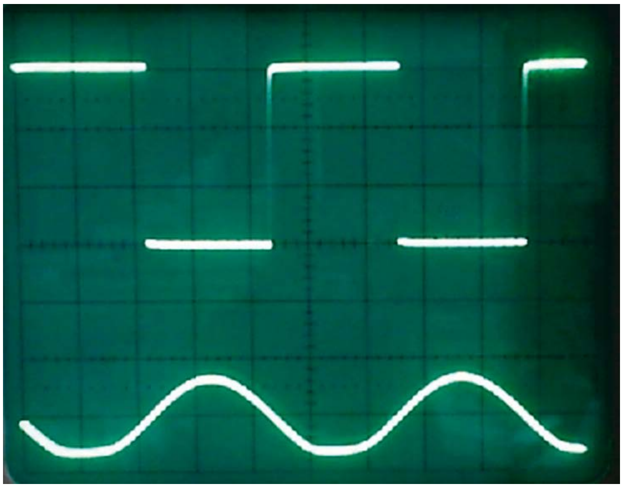


Figure 6: Rectangular wave creation.

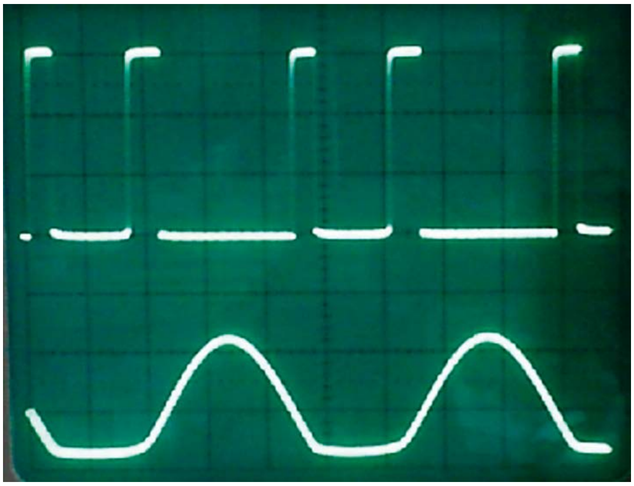


Fig. 7: Overloading in the negative range.

For 5-V operating voltage, the datasheet specifies a common mode range (i.e. span in which a signal with the same phase may be present at the inputs) from 0 V to 1.5 V. The datasheet also specifies a common-mode range (i.e. span n which a signal with the same phase may be present at the inputs) for 5-V operating voltage. Input voltages below zero are not provided, but it still works down to about –300 mV, just as with the LM358. But from –500 mV onwards there are serious problems. The input suddenly reverses its function and the result of the comparison is inverted. If the output is overloaded into the negative range, you will suddenly find signals with double the input frequency (**Figure 7**). Two Schottky diodes, placed in antiparallel at the inputs, help against overdriving. This limits the input voltage to ±0.3 V.

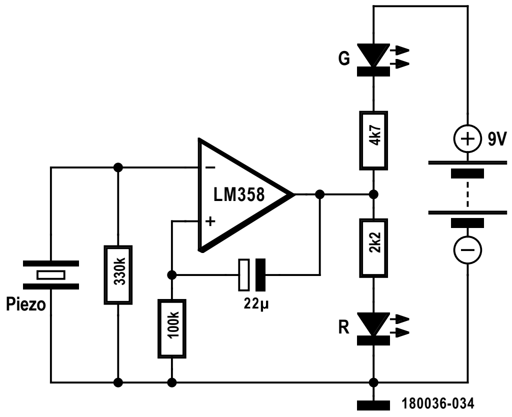


Figure 8: Monoflop and integrator with an LM358.

Clap-activated switch
The LM358 also exhibits these problems. Up to –0.3 V at the inputs everything runs normally, but if a voltage of –0.5 V is applied to the + input, it suddenly behaves as if it were a – input. But there is nothing that cannot be exploited in a positive way — with this normally unwanted side effect, a special clap switch can be built using a piezo disc for a sensor. The switch toggles the LEDs when loud noises are picked up or when the sensor is touched (**Figure 8**). The green LED lights up in the idle state. A noise or shock causes an abrupt change from green to red. Then it takes about half a minute for the circuit to switch back to the green state. So far, it looks like we're dealing with a monoflop.

But then things become unusual: the change from red to green is not abrupt, but the colours slowly fade over. An ideal operational amplifier would behave quite differently. Here, however, the special properties of the bipolar opamp with a PNP input stage are exploited. The input current is about 30 nA, so that a voltage drop of 10 mV occurs at the inverted input and only 3 mV at the non-inverted input. The difference is sufficient to create a stable rest state. The sensor must apply at least 7 mV to change the state. When toggling to the red state, the electrolytic capacitor raises the voltage at the + input and keeps this state stable through feedback. It must then charge itself to such an extent that the input voltage drops below 5 mV, which takes about half a minute. Then everything will be returned to its original state. However, this would cause the input to be a few volts below zero, which is no longer part of the normal operating range of the operational amplifier. Below –0.5 V, the function of the LM358 is reversed. Positive feedback therefore becomes negative feedback for a certain time, causing the circuit to work as an integrator. Therefore, the initial state changes slowly.

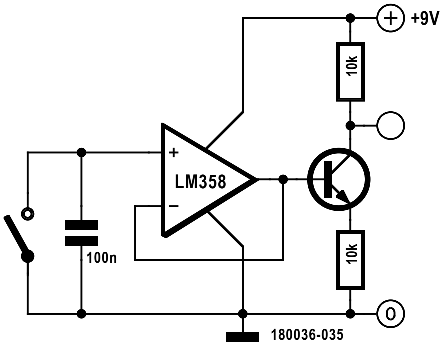


Figure 9: Experiment for inverting the input function.

If you want to understand the phenomenon better, look at the input stage. If the + input is too negative, both emitters of the Darlington transistors in the input differential amplifier are pulled down so far that the collector voltages also become more negative again. This can be illustrated with an experiment: this time an NPN transistor is driven (**Figure 9**), with the ramp generator already presented used for the control. First, the output voltage drops with increasing input voltage, as known to happen in an emitter circuit (**Figure 10**). However, as soon as the transistor is fully driven, the behaviour reverses. The output voltage now rises again.

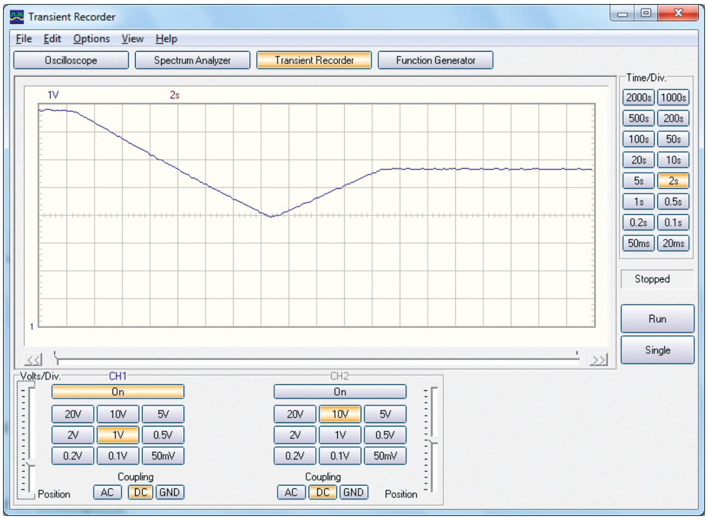


Fig. 10 Signal reversal in the event of overload.

Power amplifier

Operational amplifiers are normally used for small output currents up to a maximum of 10 mA only. However, there are also special power opamps. Sometimes even a standard opamp or a few additional transistors are enough to achieve more output current.

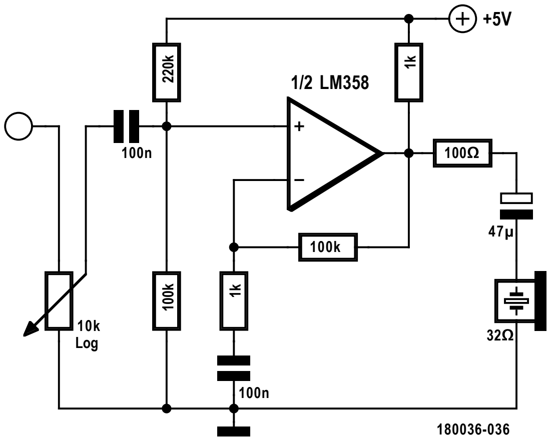


Figure 11. A headphone amplifier.

The small headphone amplifier shown in **Figure 11** was built using an LM358 dual opamp to use the Elektor SDR as a direct mixer without a PC. The LM358 tends to run into cross-over distortion when driving a relatively low impedance load at high gain. This problem is alleviated by the 1-kΩ resistor from the output to the supply voltage, because then only the lower part of the push-pull output is active at low levels. In series with the headphones a resistance of 100 Ω remains. This simplifies the task for the operational amplifier on the one hand, while on the other hand hearing damage is ruled out because the maximum performance of the headphones remains within reasonable limits. The LM358 can be driven almost to GND level but does not quite reach the supply voltage. Therefore, the open-circuit voltage was set to about one third of the operating voltage.

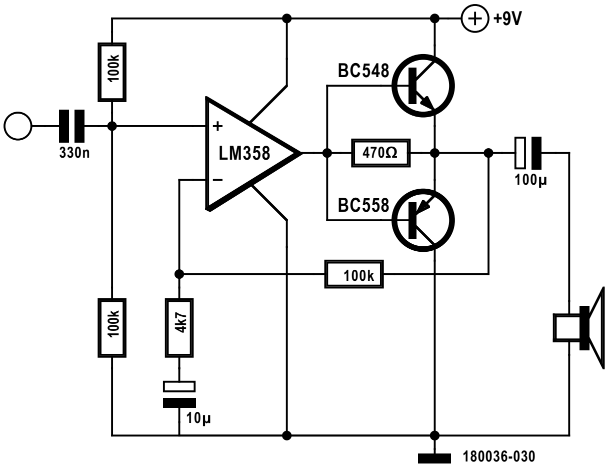


Fig. 12 Power amplifier with complementary stage.

As shown in **Figure 12**, operational amplifiers can also be used to build loudspeaker amplifiers. Although the maximum output current of approximately 10 mA is only suitable for simple headphone amplifiers, with two additional transistors a manageable setup is still feasible. The transistors form a push-pull output stage with zero quiescent current. Small signals are delivered directly by the opamp. The output stage transistors begin to amplify when the output current exceeds 10 mA. The principle is strongly reminiscent of the legendary 'Edwin' amplifier from Elektor's early days.

In principle, a push-pull amplifier without quiescent current causes high crossover distortions. However, these are largely compensated here by the strong negative feedback. Nevertheless, no Hi-fi amplifier can be built in this way. The circuit is more suitable for small experiments or simple signal horns. For serious applications it is better to use integrated power amplifiers like the LM386.

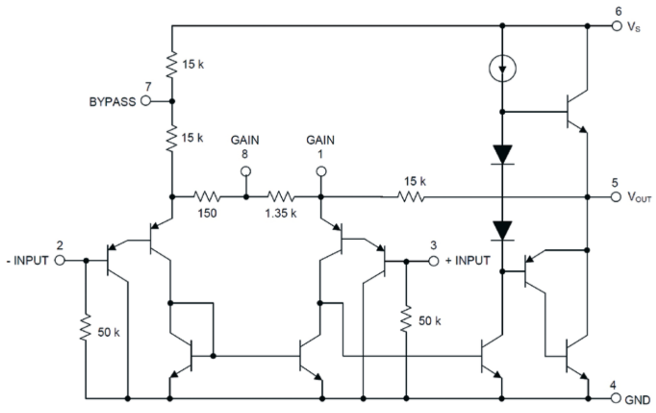


Fig. 13 Internal circuit of the LM386 loudspeaker amplifier (image: Texas Instruments).

The internal circuit of this LM386 (**Figure 13**) is very similar to that of an opamp. The LM386 however has internal feedback to set the basic amplification to about 20. The PNP input stages enable input voltages with 'drive margin' around the GND potential. The average output voltage is adjusted without further measures.

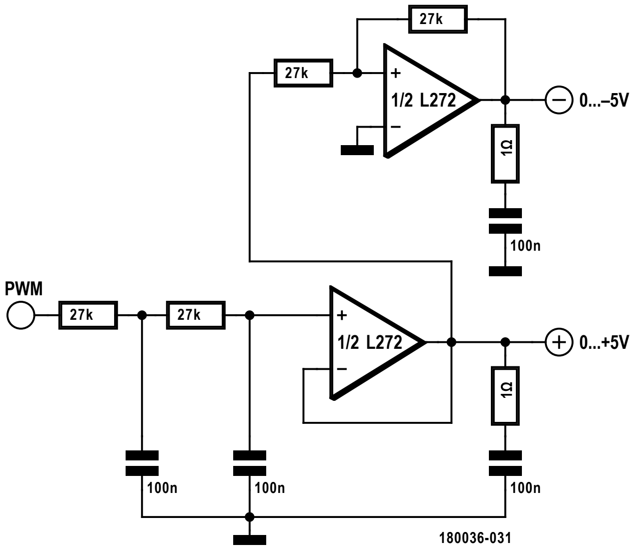


Figure 14: Using the L272 power opamp.

For general applications with high output power, the type L272 dual power opamp is available with output currents up to 1 A. **Figure 14** shows a circuit for generating an adjustable dual operating voltage. A microcontroller supplies a PWM signal which is smoothed to a DC voltage in the range 0 V to 5 V using a double low-pass filter. The L272 operates as a power buffer and inverter for a negative output voltage. However, the opamp tends to self-resonate. That's why attenuators comprising of a 100-nF capacitor and a 1-Ω resistor have been added here, as customary with some audio amplifiers. This ensures stability.

Like the LM358, the L272 has a PNP input stage. That's good to keep in mind! In a simple functional test (i.e. without connecting microcontroller-generated PWM) the input is short-circuited to ground. Both output voltages are then close to 0 V. As soon as the input is opened, the input current of the L272 slowly charges the capacitors in the input filter. Again we have a simple ramp generator and can examine the output voltages at both outputs with and without load, with total ease of mind and aiming to spot possible instability.

@ www.elektor.com

XL741 Discrete Opamp Kit:
www.elektor.com/the-xl741-discrete-op-amp-kit

Chapter 14 • EMV-EMC Limit Values and CE Declaration

Simplified measurements for private individuals and small companies
Electronics enthusiasts keen on marketing their wonderful prototypes are faced with a myriad of regulations. Two of the biggest hurdles are the CE mark and EMC testing. However, they can be mastered.

If you basically do electronics as a pastime, you are usually under the radar of the authorities. Only really big mistakes can have painful consequences. If your product unintentionally radiates a considerable amount of RF energy, you can expect to receive unpleasant questions. And if you build something that hurts someone, you can face criminal charges.

Statutory requirements
Statutory requirements apply to companies of all sizes, and there are more and more of them all the time. Here is a brief summary of the situation in Europe, and specifically Germany, with no claim to completeness.

Every product that is publicly marketed in Europe must have a CE mark as evidence of the manufacturer's declaration that it conforms to all relevant regulations [1]. For this there must be a formal declaration stating which regulations those are. Most of the requirements relate to electromagnetic interference and passive noise immunity. Now there is also RoHS conformity (including lead-free solder, etc.), as well as general requirements regarding the materials used in the product (REACH, the European chemical regulations), which are aimed at avoiding toxic substances. Many other standards can also be relevant, depending on the application. The manufacturer must be able to present this CE declaration upon request. There must also be a test report containing the measurement data used to verify compliance with the requirements.

When you see all this in concentrated form for the first time, you may be tempted to give up — and there are also many additional minor issues. Where can you find out which standard is relevant for your product? And where can you read all that? On the one hand there are the European standards, which you can find on the Internet. They are implemented as national standards, such as the DIN standards in Germany [2] or the BS standards in Great Britain. These standards are not freely available; you have to buy them — and they aren't cheap. Occasionally you might know someone who knows someone else who can loan you a copy of a relevant standard.

And then there is the electronic waste regulation WEEE. If you market products, you must also look after disposing of them so they don't end up in a landfill. A refuse bin symbol with a red line through it must be printed on the PCB or the product to indicate that it must be turned in to a public take-back point and that the manufacturer is participating in this take-back system. The details of all this have been discussed on the Elektor Forum site (in German) [3]. In any case, the whole thing involves significant costs and considerable red tape. Is that all? No — there is also a similar system for all types of batteries, and a system for packaging materials. And you always have to be on the lookout for new regulations that may affect you.

To mark or not to mark

A logical question here is who can or must comply with all this. If you want to sell a transistor, you don't have to stick a CE label on it because a transistor is not considered to be equipment. If you build a small circuit board with a voltage converter, it could be classified as a component or as a finished product. As you can see, there is a grey area.

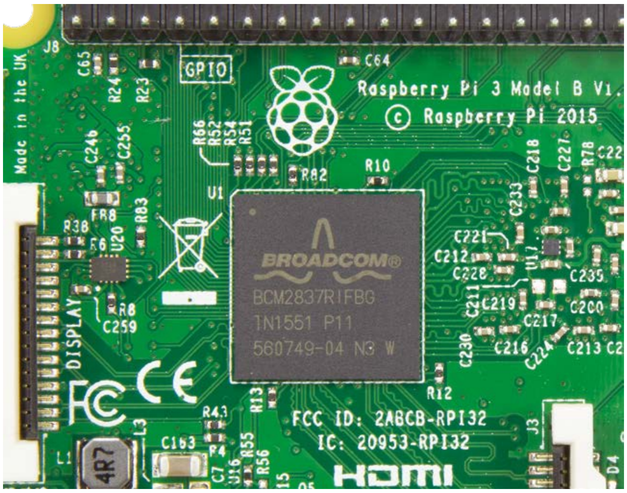


Figure 1. Markings on the Raspberry Pi.

On the back of a Raspberry Pi board (**Figure 1**) you see a CE mark, an FCC mark (the US equivalent of the CE mark), and a refuse bin symbol with a line through it. On an Arduino Uno board (**Figure 2**) there is a CE mark and an FCC mark but no refuse bin symbol; in its place there is a notice regarding RoHS conformity. Many microcontroller evaluation kits from large semiconductor companies do not have any marks at all. They probably rationalise this by saying that these products are not intended for the general public, but instead only for other companies. That's another grey area.

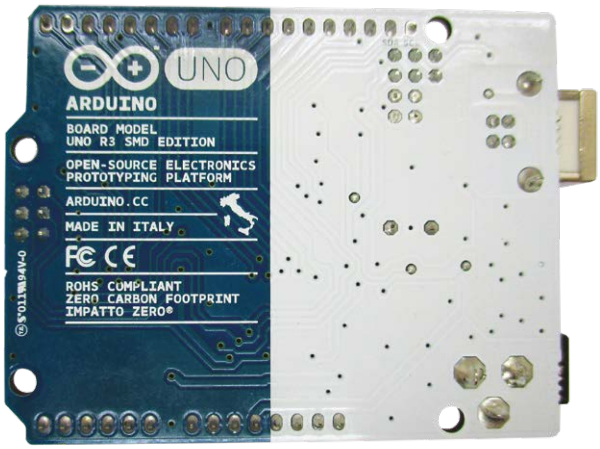


Figure 2. Markings on the Arduino Uno.

What would much smaller companies have to say about that? If you have invented a very specific measuring device for use by electronic hobbyists and want to somehow market it, and you have no idea if you will ever sell more than 50 of them, do you have to put yourself through all that stress? The official answer is “Yes”. Or suppose you have designed an experimental tube radio and want to sell it as a kit via eBay. Some would simply say that it is not a finished product, just a set of components, because the customer has to put everything together. However, it’s very unlikely that the authorities would see it that way. Another question is what is the worst that can happen to you if the authorities are alerted and decide that you are not complying with the regulations? In that case they will usually prohibit you from continuing to sell the product. Then you would either have to comply with the requirements somehow or be stuck with a bunch of unsold products. That is not pleasant if you have made a large number of them, and it can also lead to a recall campaign. However, a start-up company just getting off the ground might be tempted to take the risk. In fact the chance of getting into trouble is lower with small product volumes.

In the USA the situation is even more difficult for small companies. Doing an FCC certification yourself is not possible; it has to be done by an authorised body and it is expensive. Many small companies in the maker community cannot afford that and simply ignore it. The issue is collectively swept under the carpet and kept quiet. Everybody hopes that as long as nothing bad happens, nobody will talk about it.

Contact with the authorities
What everyone fears actually happened to me. I received a complaint from the Bundesnetzagentur (the German regulatory body for telecommunications and other public services) regarding one of my products. Their staff had taken several products off the shelf for routine checking to see if they complied with the regulations, and they weren’t happy with one of the kits. Fortunately it did not involve a violation of a limit value, but I had failed to observe an important directive. The product was a crystal oscillator with several interchangeable crystals, intended to allow simple short-wave radios to receive DRM signals. Purely as an aside, I also showed how to inductively couple a 2-MHz AM signal to a radio. I thought that with a range of less than two metres, interference would not be a problem, so there was nothing to worry about. But the authorities had a different opinion. In their view, AM at 2 MHz is not allowed. Other frequency bands have been allocated for that.

We had to go cap in hand to the authorities and jointly consider how to resolve this. The end result was a sticker with a printed notice and a change to the next edition of the manual. All in all, that was a relatively good outcome. However, another very important point was also raised. The people at the authority said to us: “Don’t think we haven’t noticed all the other products being sold without CE marks. That’s not allowed.” In other words, kits and development boards should also have CE marks. And fully assembled products or type tests are essential for verification, as described in the guidelines.

We then asked how that should work, because verification by an authorised body is not affordable for small product volumes. Then we discussed a couple of cases as examples. With that it became clear that self-declaration is certainly possible, and in most cases we could do the assessment under our own responsibility. Once a CE mark is there, nobody takes a closer look. Also, the declaration does not have to be as elaborate as is often the case in some areas. And I could send them an initial CE self-declaration for checking. That was adequate, and it could serve as a template for other products.

EMC limit values

One of the key aspects of CE declarations for electronic products is the EMC limit values. Manufacturers must ensure that their products do not cause excessive radio frequency interference, which means they must comply with statutory limit values. The usual measurements for this are often complicated and expensive. Many developers have an uneasy feeling because they are not able to properly estimate whether their products will comply with the requirements.

For many electronic and microcontroller products, one of the relevant standards is EN 55022 [4]: "Information technology equipment — Radio disturbance characteristics — Limits and methods of measurement". An interference signal in the range from 30 MHz to 300 MHz may not exceed 30 dBµV/m at a distance of 3 metres; in the range from 300 MHz to 1 GHz the limit is 37 dBµV/m. Interestingly enough, nothing is specified for the frequency range below 30 MHz — as though the traditional shortwave bands no longer matter. However, the lower frequencies certainly do matter if anything manages to penetrate into the power grid. If you have testing performed by an authorised body with suitable equipment, you typically receive test results like the chart in **Figure 3**. There you can see the usual picket fence of harmonics of the CPU clock frequency emitted at different signal levels. The test report notes which peaks were found at which frequencies and the corresponding signal levels. The measurement is made in a shielded chamber, so all the signals must originate from the device under test.

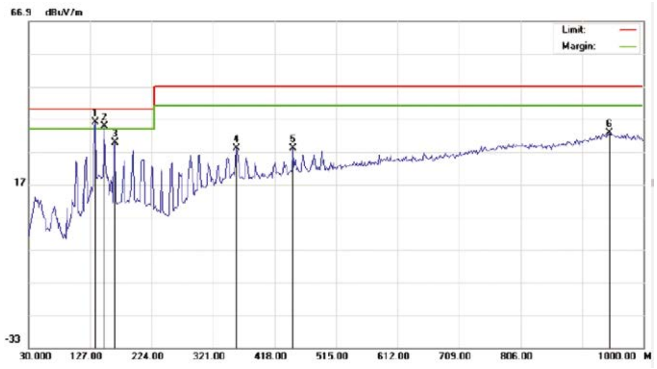


Figure 3. EMC measurement of a microcontroller.

To be able to correctly assess the situation myself, a few years ago I bought a spectrum analyser with a measuring range up to 1 GHz. The only other thing I actually need is a fully shielded test chamber. However, there is a simpler alternative. I take a very small antenna in the form of a wire loop and hold it very close to the device under test. On the spectrum analyser display I see all sorts of known signals, such as FM broadcast signals in the VHF band, DAB, DVBT and so on. Between them I also see the signals that actually matter to me: the ones that become stronger when the probe antenna gets closer to the device under test. By comparing them to the levels of the known signals, I can estimate whether the interference signals generated by my device exceed the limit values.

Of course, this all depends very much on the antenna you use. But roughly speaking you can say that if a dipole tuned to 100 MHz produces a 1-μV signal with a 50-Ω load, the electric field strength is about 3 to 4 μV/m. In many cases all local FM transmitters will have very similar signal levels, which you can note for the location concerned. They are typically well above the EMC limit levels, as otherwise noise-free reception would not be possible. For example, if the signal level of an FM transmitter at 100 MHz is 1 mV on a 50-Ω dipole, that is 60 dBμV, which corresponds to a field strength of about 70 dBμV/m.

Those signals are my reference levels. Then I can say that my interference signals must be a certain number of dB below the reference signals. If I see that the interference signals at a distance of 50 cm are sufficiently weak and they are not even detectable at a distance of 3 m, then I can be reasonably certain that my device complies with the limit values. And in the vast majority of cases, relatively small devices easily comply with the requirements.

An AVR microcontroller such as the one on the Arduino Uno board, makes it easier to build devices with low noise emissions. The RAM and ROM are integrated into the chip, so there are no external lines with fast signals. The Vcc and GND pins are next to each other. If you put a 0.1-μF ceramic capacitor right next to these pins, the supply lines will be virtually noise-free. With the right fuse settings, the crystal oscillator operates at a very modest amplitude of around 1 Vpp instead of maximum amplitude, and what's more, the waveform is virtually sinusoidal. That keeps the harmonic levels low. The only other thing you have to consider is the signals on the port pins. There you have watch out if relatively high frequencies are present and a PCB track or wire could start acting like an antenna.

A continuous ground plane on the back of the PCB also minimises interference. The Elektor SDR shield is a good example of how well that works. There you might expect that the Arduino would cause a lot of interference to shortwave reception. In fact it doesn't, because the Arduino Uno and the SDR shield both have ground planes and tight decoupling.

EMC estimation with simple resources

Sometimes you can draw conclusions even without a spectrum analyser. What matters in most cases is the harmonics of the clock frequency. With a microcontroller clocked at 16 MHz, you have to measure the harmonics at 32 MHz, 48 MHz, and so on. At 96 MHz there might be a signal in the FM broadcast band. With a signal level of 30 dBμV/m at a distance of 3 metres, you have a strong signal that distinctly suppresses the noise. However, in most cases this harmonic can only be seen very close to the microcontroller and is not detectable at a distance of 3 metres. In that case you can at least say that this harmonic is clean.

Some modern radios display the antenna voltage in units of dBμV. That includes the Elektor DSP Radio with the SI4735. Using a perfectly ordinary rod antenna, you can make measurements with that sort of radio that are comparable to measurements made with a spectrum analyser. For example, I measured a signal level of 60 dBμV from a local FM transmitter at 99.2 MHz. I could also receive a signal from the Arduino at 96 MHz. For that I had to bring the antenna to within 20 cm from the board to clearly suppress the noise floor. The indicated antenna voltage was 20 dBµV. That is roughly equivalent to the limit value of 30 dBµV/m. At a distance of 50 cm I was no longer able to determine whether the Arduino was on or off. In other words, the emitted interference was already below the limit value at about 20 cm, compared to the specified distance of 3 m. With that sort of result, it is unlikely that any of the other harmonics will exceed the limit value.

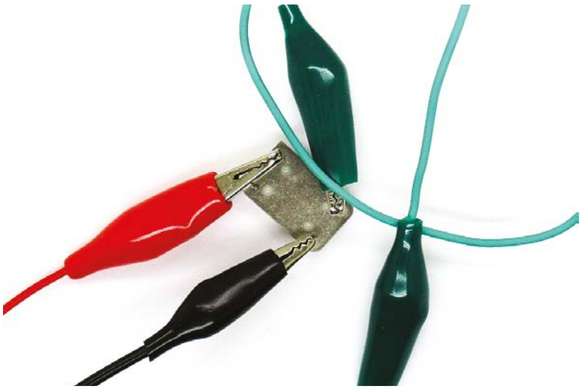


Figure 4. Here undesirable emissions are observed.

For comparison purposes, you can also occasionally do something intentionally wrong just to get a feel for the magnitudes involved. For example, I took a packaged 10-MHz crystal oscillator, connected a 30-cm wire to its output, and simply laid it on the bench (**Figure 4**). The signal levels were distinctly higher than the limit values. The steep signal edges generate lots of harmonics, and the wire is a suitable antenna. Signals were visible every 10 MHz, with the odd harmonics stronger than the even. The 90-MHz harmonic was certainly there, so I could check it with an FM radio. The range was amazingly large. An FM radio showed 40 dBµV at 3 m distance for the 90-MHz harmonic, which is much too high.

That meant I had to switch it off right away, because I could not know which of the many harmonics was already interfering with a radio service. In that situation the least interference occurs at the fundamental frequency, since the wire would have to be 7.5 m long to act as a quarter-wave antenna at 10 MHz. However, there is a good chance that one of the many harmonics will be radiated especially well because it matches the resonant frequency of the antenna.

This experiment shows that the signals usually present on typical circuit boards can certainly generate interference signals above the limit values. You might think that a port pin cannot output much power, but that is easy to underestimate. For comparison, radio amateurs have what they call the 'whisper net' (WSPR), which they use to see how much power is necessary to reach a given range. A Raspberry Pi with suitable software is all you need for a transmitter. The RF signal comes directly from a port pin. After that, you only need a good low-pass filter to get 10 mW at the antenna without any amplifier. In the shortwave bands, that is enough to reach everywhere in Europe. A range of 2,000 km is easily possible with just 10 mW. This also means that an unintentionally radiated high-frequency signal can easily reach the International Space Station at a height of 400 km.

Summary

If you develop a feel for the potential risks and take known precautions seriously, such as ground planes and tight decoupling, you should be on the safe side. Then a few simple measurements can confirm what you already know: your device complies with the limit values. It often helps to have the radio on while you're working. Sometimes you run into problematic signals, and then you hear interference on the radio. That way you never forget what matters.

Web Links

[1] Searching for directives:
https://ec.europa.eu/commission/index_en
[2] Overview of available standards:
www.beuth.de/de/themenseiten/ce-kennzeichnung
[3] WEEE discussion:
https://weee-forum.org/
[4] A bit outdated, but readily available:
http://cq-cq.eu/EN55022-2006.pdf
[5] Elektor DSP Radio:
www.elektormagazine.com/100126

Chapter 15 • LED-LDR Ring Oscillator

Can high-brightness white LEDs and light-dependent resistors (LDRs) be used to make an oscillator? I decided to have a go. For the experiment I chose type PGM5516 LDRs which have a relatively low resistance (5 kΩ to 10 kΩ at 10 lux). A ring oscillator configuration was used, with each LED placed directly next to its neighbouring LDR (see circuit).

The first test, without the capacitors, was unsuccessful. I had hoped that the natural slow response of the LDRs would be enough to start oscillation, but the oscilloscope proved me wrong.

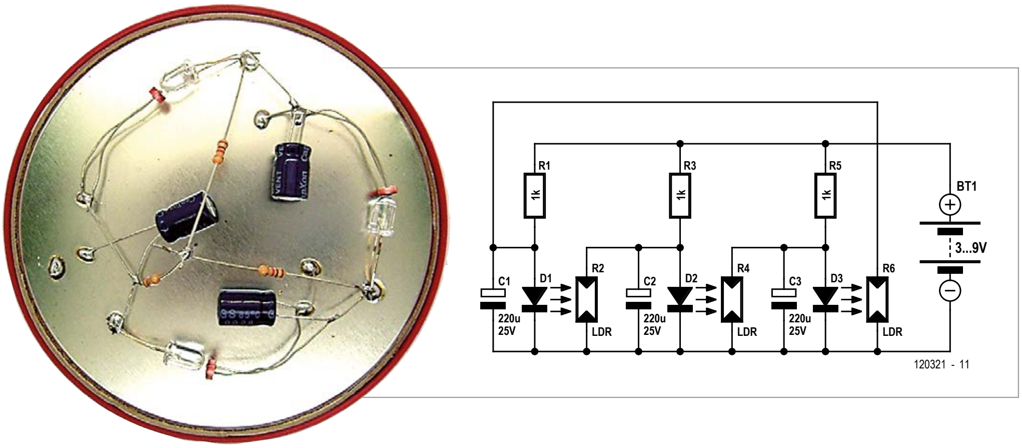


Figure1. An oscillator with passive components.

Only by adding the 220 µF electrolytic capacitors could I get the oscillation to start, using a power supply of between 3 V and 9 V. The oscillation frequency rises with supply voltage. At 3 V the circuit must be kept in darkness (and the result is a very low-energy running light!). At 2.7 V the circuit draws 0.9 mA and is disturbed by even the smallest amount of ambient light.

Chapter 16 • Picoammeter

It is often necessary to deal with tiny currents in the picoamp to microamp range when making measurements in ionisation chambers and other radiation sensors, as well as when testing insulation.

The instrument described here can be used to measure currents from around 0.1 pA to 1 µA, without the need to change range. One approach is to exploit the logarithmic characteristic curve of a silicon diode, with the diode voltage buffered by a type TLC272 CMOS opamp. For the first experiment we used a 1N4148 silicon diode (see l/h circuit). However, the lowest current measurable using this arrangement was over 10 pA, as below that value the diode's characteristics deviate from the logarithmic curve.
A particularly good diode with a very low reverse current is the gate-source diode inside a BF245 JFET (see lower circuit). Using this we can measure currents of less than 1 pA. To calibrate the circuit we use known currents at the input and measure the output voltage:

1 µA	580 mV
100 nA	510 mV
10 nA	440 mV
1 nA	370 mV
100 pA	300 mV

The above values closely follow a logarithmic curve: each increase in current by a factor of ten gives an output voltage increment of 70 mV. We have already covered four decades: the graph shows how the curve can be extended to even lower currents.

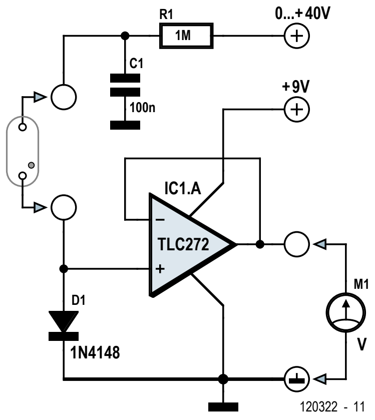


Figure1. Silicon diode as a current sensor.

Experiments show that it is even possible to get useful results with currents of less than 1 pA. In this case, however, it is essential to carefully screen the whole circuit, including the item under test, against the effects of external fields. For the prototype a metal tin was used with feed-throughs.

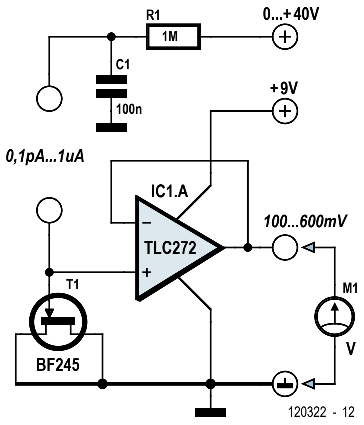


Figure2. JFET as a current sensor.

Some results obtained with the prototype:

- ionisation chamber with a sample of pitchblende: about 1 pA;
- BPW34 photodiode used as a radiation detector (in complete darkness): about 10 pA;
- burnt-out filament lamp: about 100 pA;
- burnt-out halogen lamp: about 0.1 pA.

From these last two results we can see that quartz glass is a considerably better insulator than ordinary glass.

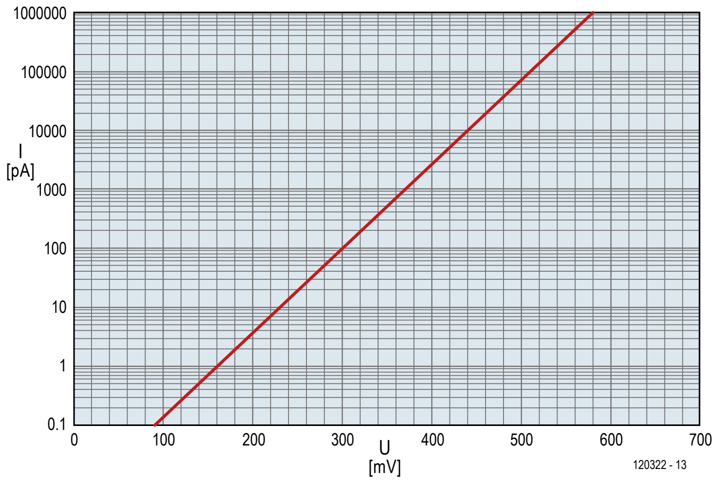


Figure 3. A characteristic curve of a JFET diode.

Chapter 17 • LC Oscillator with Pot Tuning

An LC oscillator is usually adjusted using a variable capacitor. However, for frequencies below around 100 kHz this calls for a variable capacitor with a value in the nanofarad range, which is somewhat impractical. In many situations, however, a potentiometer can be used instead.

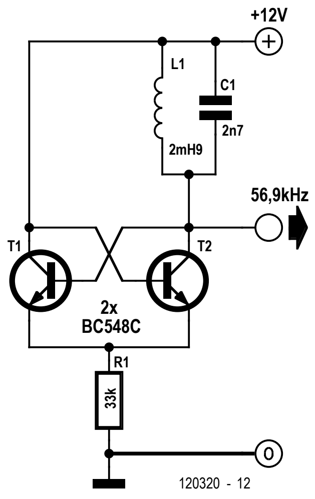


Figure 1. A basic circuit.

We start by looking at an oscillator using a 2.9 mH inductor salvaged from a low-energy lightbulb and a 2.7 nF capacitor (see upper circuit). The theoretical resonant frequency of this combination is 56.9 kHz.

The circuit operates from a supply voltage as low as 1 V, as the resonant circuit has a high Q factor. If an extra 10 nF capacitor is wired in parallel the resonant frequency falls to 26.2 kHz. The Q factor is reduced and so the gain in the circuit must be increased, and a supply voltage of at least 2 V is needed.

Using a switch it is possible to select between the two frequencies (see middle circuit).

And now the subtle bit: instead of the switch we use a 1 kΩ potentiometer (see lower circuit). In this form the frequency of the oscillator can be smoothly adjusted using the potentiometer, almost as if we were using a 10 nF variable capacitor. Experience shows that using a linear potentiometer gives a rather non-linear frequency adjustment, and so it is preferable to use a logarithmic potentiometer. A further problem is the high level of damping: the energy loss needs to be made up for with higher gain, and so with a higher emitter current. This can be achieved either by reducing the emitter resistor or by increasing the supply voltage.

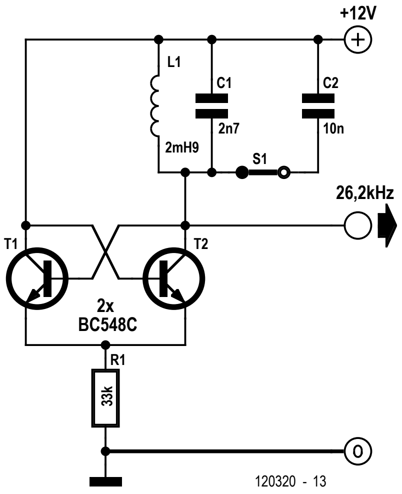


Figure 2. An extra capacitor is wired in parallel.

Experiments show that the maximum frequency coverage possible is around a 2:1 range. If the two capacitors are very different in value the damping in the middle of the frequency range is so great that oscillation stops. With the values shown in the circuit diagram the frequency can be adjusted between 34.2 kHz and 55.1 kHz.

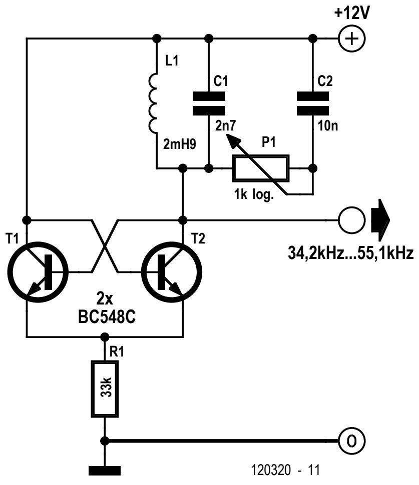


Figure 3. A adjustable oscillator.

Chapter 18 • FET Radiation Meter

What must be the simplest possible radiation meter consists of just a BF245 JFET and an ohmmeter. Ions produced as a result of the radiation charge up the gate of the FET and thus change its resistance. The FET needs to be enclosed in a metal tin to screen it from electric fields and from ions that might normally be present in the atmosphere. After taking a calibration measurement we can experiment by placing various samples inside the tin.

The device was tested using a small sample of pitchblende (a uranium ore), a 241Am source taken from a smoke alarm, and a gas mantle, still in its paper sleeve. The results were as follows.

Reference (no sample)	160.2 Ω
Pitchblende	156.3 Ω (−3.9 Ω)
241Am source	155.9 Ω (−4.3 Ω)
Gas mantle	159.0 Ω (−1.2 Ω)

The results are clear: when a sample is placed near the FET it becomes positively charged, reducing its drain-source resistance. The experiments above were repeated several times and showed good reproducibility. In practice it takes around half a minute for the FET's resistance to settle.

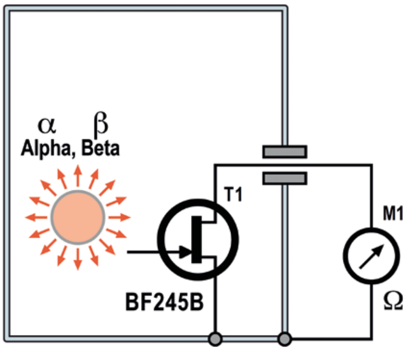


Figure 1. FET Radiation Meter.

Chapter 19 • 'Green' Solar Lamp

Energy saving is all the rage, and here is our small contribution: how much (or rather how little) current do we need to light an LED? Experiments with a super-bright 1 W green LED showed that even one microamp was enough to get some visible light from the device.

Rootling in the junk box produced a 0.47 F memory back-up capacitor with a maximum working voltage of 5.5 V. How long could this power the green LED? In other words, if discharged at one microamp, how long would the voltage take to drop by 1 V? A quick calculation gave the answer as 470 000 seconds, or about five days.

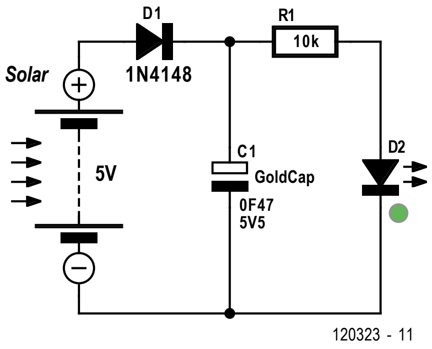


Figure 1. Solar lamp with capacitor for energy storage.

Not too bad: if we use the capacitor for energy storage in a solar-powered lamp we can probably allow a couple more microamps of current and still have the lamp on throughout the night and day. All we need to add is a suitable solar panel. The figure shows the circuit diagram of our (in every sense) green solar lamp.

Chapter 20 • Battery Maintainer

The author found that a long-neglected gel battery in a hand-held vacuum cleaner had gone high-resistance. It took some effort to make it usable again, by alternately applying voltages of opposite polarities to it. A reverse voltage can help to break down the internal non-conducting layers which can form when the battery is left idle. The battery is now back in action and charging and discharging as normal.

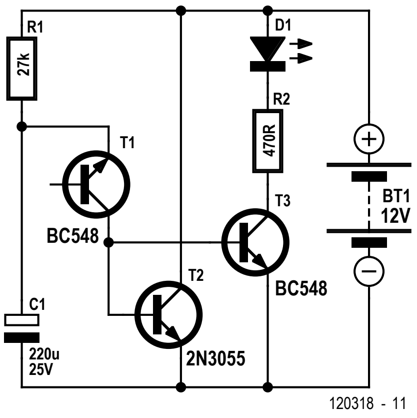


Figure 1. The less frequently the LED flashes, the lower the battery voltage.

Unfortunately, however, the battery will probably start to fail again if we once more leave the appliance lying around unused for a while. To prevent this happening the author applied a well-known technique: the battery is intermittently presented with a very brief high-current load. The circuit shown here does the job: every two seconds it draws a current of about 1 A for 2 ms. This corresponds to an average current of about 1 mA, which is of comparable magnitude to the self-discharge of the battery. Although the circuit does not consume much energy, it can keep the battery fresh.

The circuit is based on the NPN relaxation oscillator from the 2011 Project Generator Edition of Elektor (www.elektormagazine.com/magazine/elektor-201107/19699/), here delivering the base current for the power transistor. In the prototype the current was measured at around 1 A: to be on the safe side, you can add an extra load resistor to the circuit.

The LED indicates when each current pulse occurs, which also serves as an indication of the battery's charge state: the less frequently the LED flashes, the lower the battery voltage.

Chapter 21 • One-transistor Voltage Converter

Taking apart a solar-powered lamp revealed a single-transistor voltage converter circuit that allowed an LED to be driven from a 1.2 V cell. The l/h diagram shows the circuit (with slight modifications). The circuit oscillates at about 500 kHz and, at a cell voltage of 1.4 V, draws 11 mA with a respectably bright LED. The circuit works down to a supply voltage of 0.8 V.

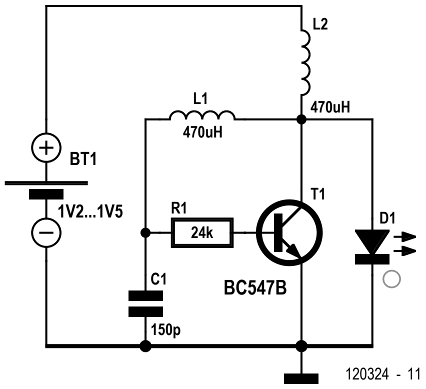


Figure 1. An LC oscillator.

The oscilloscope shows 3 Vpp at the LED, as expected. The left-hand coil and the capacitor form a series resonant circuit, excited by the collector of the transistor which alternates periodically between conducting and blocking. When the transistor is off the upper coil dumps its stored energy so that the voltage on the collector rises to about double the cell voltage.

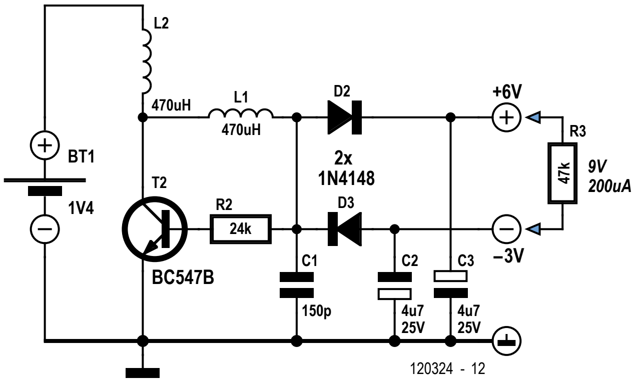


Figure 2. An additional rectifier.

A sinewave voltage of 35 Vpp (!) was measured across the capacitor in the resonant circuit. Using a two-channel oscilloscope showed the phase relationships: the resonant circuit shifts the phase by about 90 degrees. The base resistor coupled with the base capacitance and the Miller capacitance (http://en.wikipedia.org/wiki/Miller_effect) of the transistor add a further phase shift.

The voltage increase obtained using the series resonant circuit can be used to make a bipolar voltage converter, for example to power operational amplifiers (see r/h diagram). Two electrolytic capacitors and two diodes rectify the voltage. The circuit can deliver a voltage difference of 9 V at 0.2 mA, which is enough for a low-power opamp.

Chapter 22 • Analogue LED Chaser Light

The circuit shown here is formed of nine inverting transistor amplifier stages connected in series with an LED connected between emitter and ground. The output of the final stage connected to the input of the first stage.

The principle is similar to the ring oscillator described by the author elsewhere in this edition. Similar but not identical, since the stages in this circuit have additional delay elements formed of a 33 kΩ resistor and a 47 µF electrolytic. The circuit operates with any odd number of LED stages of your choice, for instance with nine (as shown here).

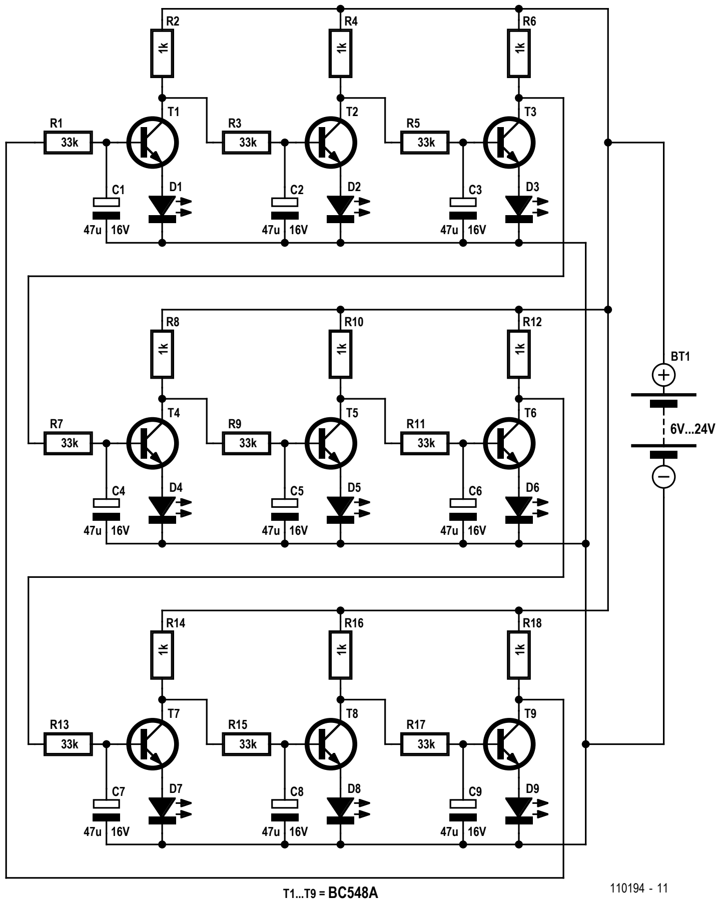


Figure 1. The ring oscillator.

The project oscillates very reliably and the way the twinkling LEDs fade in and out is quite a novelty. If you watch just two LEDs, they look like a simple blinker as there is always one lit LED next to a dark one. But with the lights circulating it looks a lot more complicated. Any disturbance will also travel round the ring. To watch the effect take a look at this You Tube Video: www.youtube.com/user/bkelektronik#p/u/1/-U_vAx_EK_M

Chapter 23 • Experimental Hall Sensor

Hall sensors can of course be purchased but making them yourself is far more interesting (and satisfying)!

According to the theory the crucial thing is to use a touch layer that's as thin as possible; the length and width are unimportant. An 'obvious' starting point for our trials would be copper, which in the form of printed circuit board material is easy to find and handle. Copperclad board may be obvious but not ideal, because it has a very weak Hall constant. Nevertheless we should be able to use it to demonstrate the Hall effect by using very powerful magnets in our sensor.

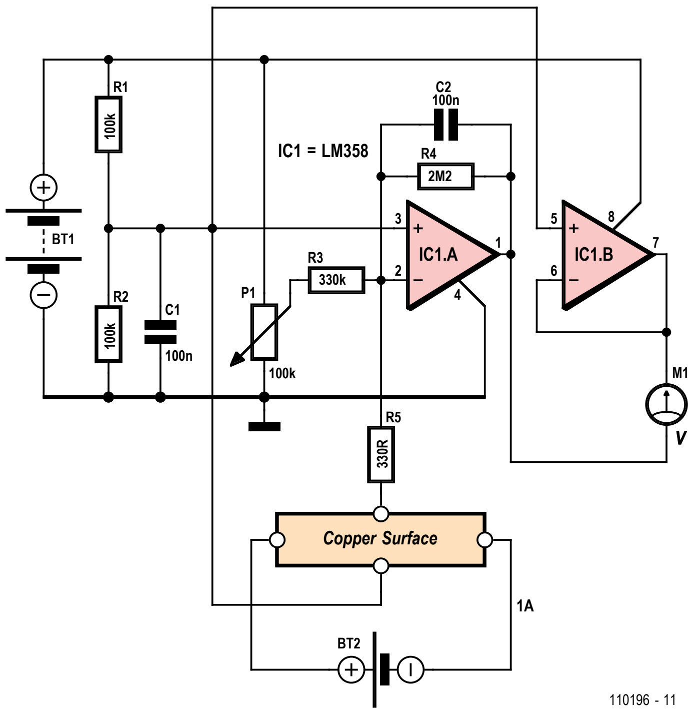


Figure 1. Hall sensor and amplifier.

To achieve detection we need the highest possible level of amplification. In the circuit shown here the voltage amplification is set by the relationship of the two feedback resistors of the first op-amp. With the values given (2.2 MΩ and 330 Ω) produce a gain of 6,667. This also creates a convenient bridge connection for taking measurements. The trimmer potentiometer allows fine adjustment. With zero setting that's accurate to within millivolts we could use this test point to measure Hall voltages of well below a microvolt. Finally in this way we could also measure the flux density of a magnet.

Copper has a Hall constant of AH = –5.3·10-11 m3/C. The thickness of the copper layer is d = 35 µm. The Hall voltage then amounts to:

$$VH = AH \times I \times B / d$$

When the field B = 1 T and current I = 1 A a Hall voltage of VH = 1.5 µV is produced. The 6,667-fold gain then achieves a figure of 10 mV. The circuit thus has a sensitivity of 10 mV per Tesla. That said, adjusting the zero point with P1 is not particularly easy. The amplifier has a separate power supply in the form of a 9 V battery (BT1). To take measurements we connect a lab power supply with adjustable output current (BT2) to the Hall sensor (the copper surface) and set the current flowing through the sensor to exactly 1 A. Then the zero point must be adjusted afresh.

Next we place a strong Neodymium magnet below the sensor. The output voltage of the circuit should now vary effectively by several millivolts. Note that there are several effects that can influence the measurements we take. Every displacement of the magnet will produce an induction voltage in the power feed wires that is significantly greater than the Hall voltage itself. Every time you move the magnet you must wait a while to give the measurements time to stabilise. With such small voltage measurements problems can also arise with thermal voltages due to temperature variations. It’s best not to move and inch — and to hold your breath as long as possible!

Chapter 24 • Minimalist Dip Meter

In days gone by a radio amateur always had a dip meter close to hand in his 'shack'. Now that people can afford oscilloscopes, the poor old dip meter has lost its importance and is frequently no longer to be seen. Actually this is a shame because many tasks are much easier to carry out with a dip meter. Anyone who's interested (perhaps the second time around) can easily build one rapidly with this very simple but adequate circuit. The interesting question is namely what do you actually need from a dip meter?

- A visual display of the dip? Nope, the 'scope can handle that task.
- A large frequency scale? Not necessary, as you can connect a frequency counter for this.
- A selection of coils? We don't need these because we can use a jumper to change range (no coils to lose any more!).

The sensor coil L1 has ten turns and is wound using an AA-size battery as a former. This coil will allow us to over the range from 6 MHz to 30 MHz. With jumper JP1 open an additional fixed inductance of 10 µH comes into circuit. The frequency measurement range is then from 2.5 MHz to 10 MHz. The switch may be replaced by a jumper.

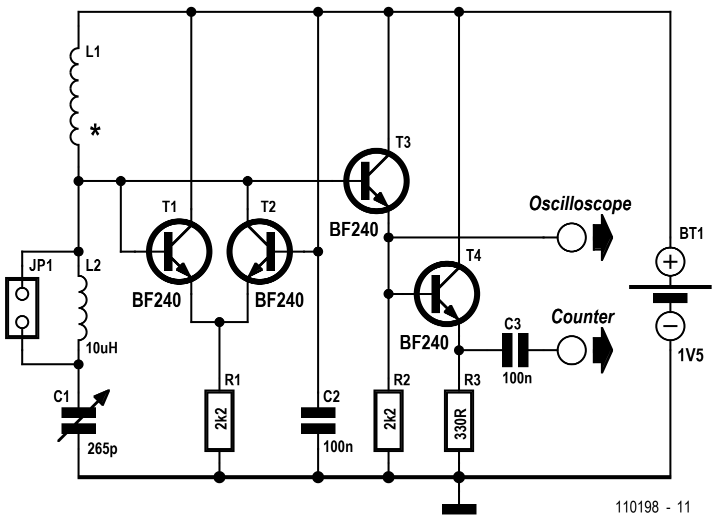


Figure 1. HF oscillator with buffer stage.

To take measurements you hold a resonant circuit close to the sensor coil. Tune the rotary capacitor C1 slowly to and fro in order to find the resonant frequency, at which the oscillator amplitude decreases somewhat. The frequency can then be read directly off the oscilloscope. To obtain a very accurate measurement you can additionally connect your frequency counter to the second output.

Chapter 25 • Wideband Receiver for Spark Transmissions

In the early years of radio technology spark transmissions ruled the (air) waves. They occupied a relatively wide bandwidth, in what came to be known as the long waveband. The receivers used had a corresponding bandwidth, as 'wide open' as the proverbial 'barn door'. Most were simple detectors without an amplifier stage.

Today when you operate an electric light switch you produce a wideband spark that's audible on some radios as a crackle from long through to short waves. The same occurs with intermittent breaks in cables, high voltage strikes, defective transformers, poorly suppressed electric motors and all kinds of contacts that open and close.

With a suitable receiver it's possible to track down the source of these problems. Tests using normal radios are largely unsuccessful, simply because they display restricted bandwidth and are too effective at suppressing short interference pulses. After some research the best results were obtained with a wideband Audion receiver.

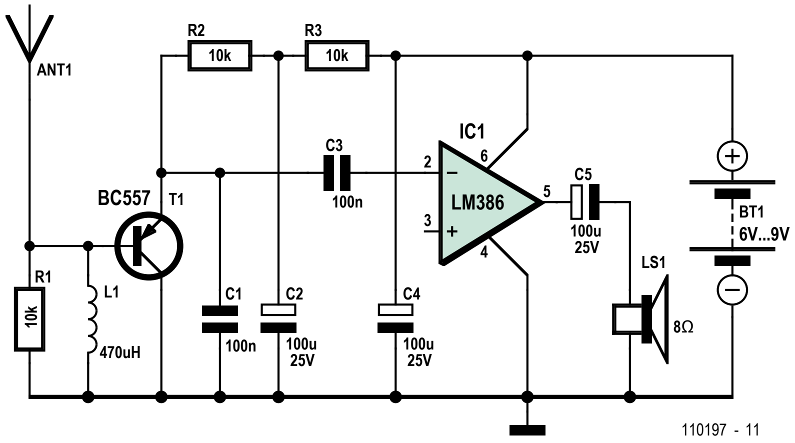


Figure 1. Wideband receiver with LF amplifier.

The requirements for this kind of receiver are totally different from normal radio reception: the receiver must have bandwidth as wide as possible, with maximum sensitivity in the long wave region. A further special request: since the wave packets of a single spark are often extremely short, the receiver should integrate them into a longer pulse whose spectrum should lie well inside the audible range.

As for the circuit, the audion stage in the collector circuitry detunes the input circuit. To prevent self-oscillation we need to add a 10 kΩ resistor. Using an oscilloscope you can see extremely short pulses on the emitter of the BC557 being broadened. The amplitude is frequently sufficient to drive the final amplifier into limiting. A 1 μs long input pulse results in a circa 1 ms long audio pulse in the loudspeaker.

Chapter 26 • Ring Oscillator

The ring oscillator comprises a number of inverting transistor amplifier stages connected in series, in which the output of the final stage is connected to the input of the first stage.

You have the choice of using three, five, seven or nine stages. The only condition is that the number must be odd, not even. A feature of this circuit is that no capacitors are required. Oscillators of this kind are used widely in integrated circuits, for instance in microcontrollers.

In principle we are dealing with an amplifier with negative feedback that reaches oscillation as a result of the high total amplification. In the circuit shown in **Figure 1** five stages are employed. To avoid affecting the ring a buffer stage is used for uncoupling the oscillator signal. All resistors in the circuit have a value of 2.2 kΩ and all transistors are type BC548A. The oscillator produces a frequency upwards of 1 MHz, which is somewhat dependant on the power supply voltage (see **Figure 2**). An average maximum frequency of 1650 kHz is produced with an operating voltage of 3 V.

The ring oscillator can be viewed in its broadest sense as a run-time oscillator. The signal run-time of all five stages amounts to half the oscillation period, in other words exactly 300 ns at 1.65 MHz. Every individual stage then has a run-time of 60 ns. At higher operating voltages the delay introduced by each stage increases somewhat, because the transistors are driven heavily into saturation.

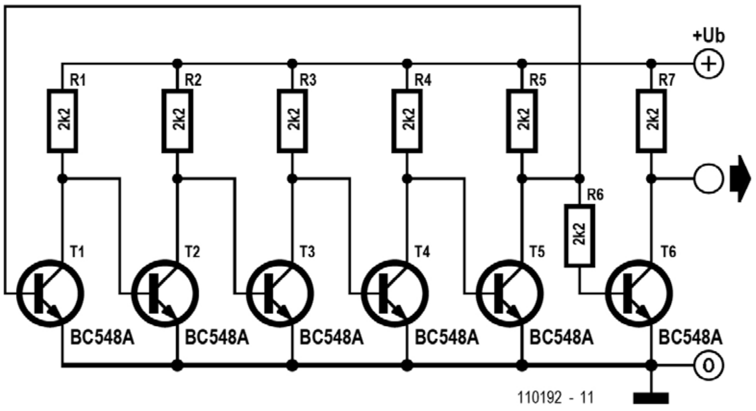


Figure 1. Five-stage ring oscillator.

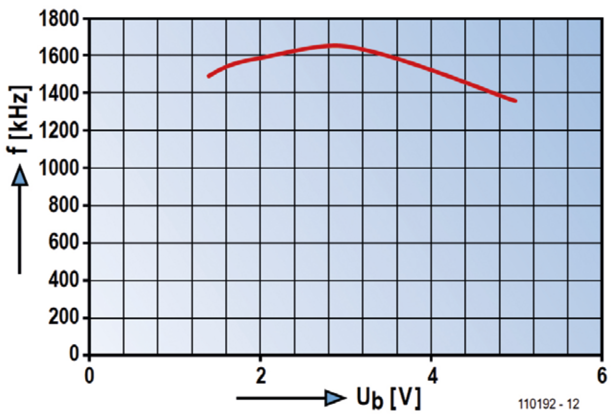


Figure 2. The frequency as a function of the power supply voltage.

Chapter 27 • LED Multi-Flasher

The first circuit in **Figure 1** shows a particularly simple LED flasher for AC power operation with six channels. All six LEDs flash entirely at random (not synchronised), producing a totally chaotic display. This must be the ultimate low energy lamp bulb, consuming a mere 0.2 watts or so. To see it in action take a look at my short video [1] on the Internet.

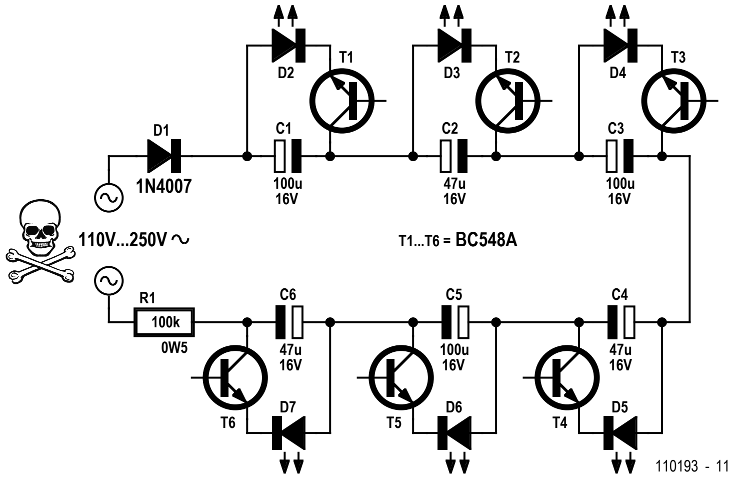


Figure 1. Six LED flashers in a row.

The project employs my NPN multivibrator circuit described elsewhere in this issue. Each of the six NPN multivibrator circuits (connected here in series) draws the same loading current. By varying the value of the electrolytic capacitors you can change the flashing frequency and brightness. You can make the circuit flash more slowly if you select a value higher than 100 kΩ for the charge resistor R1 or add an additional resistor (in the power feed).

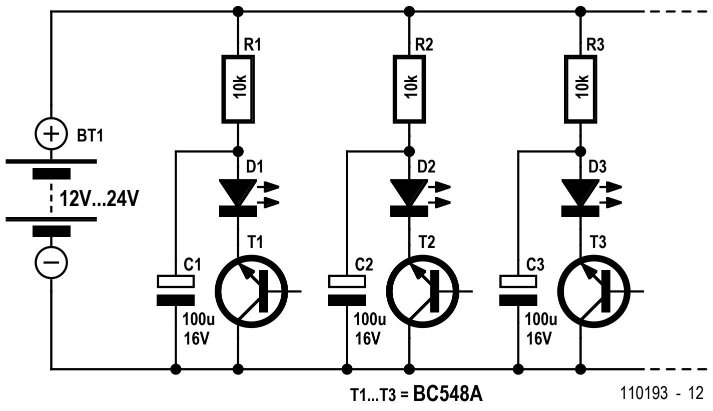


Figure 2. The 12 to 24 V version.

A disadvantage of the circuit is the risk of death that arises from the fact that the circuit is connected directly to the AC power line. Therefore it is extremely dangerous to touch components of the circuit when live. Therefore it's imperative that you construct the project in a well insulated (touch-proof) plastic case fitted with proper cable restraints.

To avoid risks of this kind **Figure 2** shows another version of the circuit designed for low voltage operation in the range from 12 to 24 V. Here the NPN multivibrators are powered in parallel (not series) with the operating voltage. Using this method you can also construct longer flasher chains.

Internet Links

[1] www.youtube.com/user/bkelektronik#p/u/6/lqr-YTf3b9U

Chapter 28 • Emitter-Follower Audion

A shortwave audion receiver using only two transistors and a single 1.5 V battery — that must be the ideal entry level into shortwave receiver technology. Just add an active PC loudspeaker for very convincing performance.

A special feature is the audion circuitry that uses a BC558C PNP transistor working in emitter follower mode. This function works thanks to the few picofarads of internal capacity between the transistor's base and emitter. This produces a capacitive voltage divider, enabling the transistor to operate as a three-point oscillator, also known as a Hartley oscillator. Only a minute amount of emitter current is required to go into oscillation. The trimpot (trimmer potentiometer) is used to adjust the audion for AM reception so that it does not quite oscillate (immediately before oscillation sets in), for CW (telegraphy with keyed carrier) and SSB (single-sideband) reception it is set slightly higher.

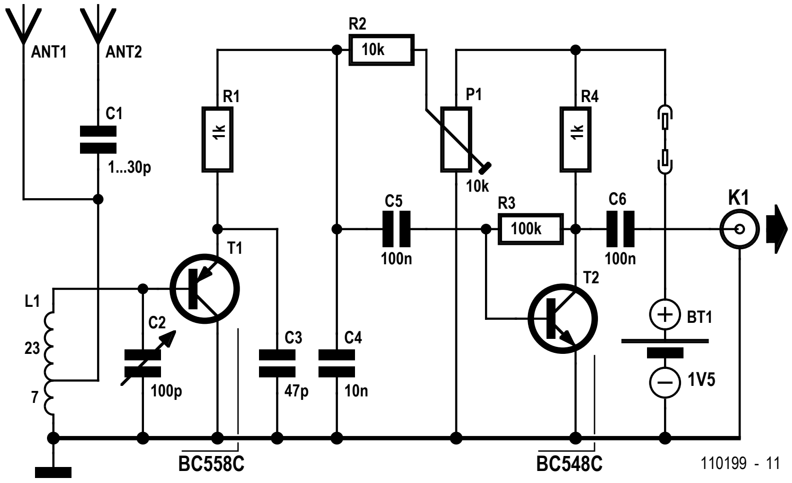


Figure 1. Audion with feedback control.

Decoupling and amplification of the audio signal is handled by the second transistor. The signal on the output connector K1 is at line level, with an output impedance of about 1 kΩ.

Either of the two antenna connections ANT1 and ANT2 can be used. A good ground (earth) connection is essential for this circuit, in which case a short indoor wire antenna of less than a meter in length connected to ANT1 will be sufficient to pull in countless broadcast stations. For DX (long distance) reception an external antenna is better, for example an aerial 'long wire' of around ten meters (30 ft.) length. In this case the ANT2 connection must be used. The coupling to this input is slightly weaker in order to reduce resonance and offset any reaction (feedback). As a general rule, the longer the antenna, the smaller the value of coupling capacitor C1.

Chapter 29 • NPN Relaxation Oscillators

If you've read old textbooks on electronics basics you may recall how it's possible to create a multivibrator from just a neon lamp and a capacitor. The circuit of the simple multivibrator shown in **Figure 1** works in exactly the same way but using an NPN transistor instead of a neon lamp and at a much lower voltage. Anyone can check this out because the function is so basic. But why?

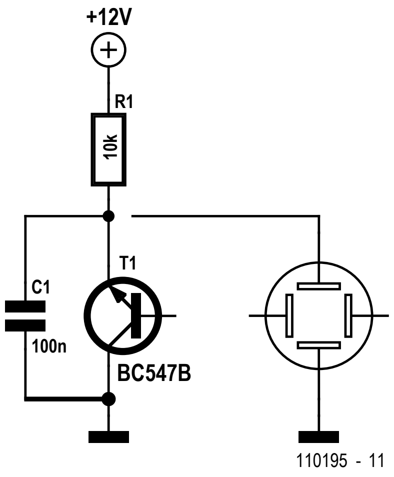


Figure 1. The principle.

The author explains the circuit function like this:

In inverse operation (emitter positive with respect to the collector) the NPN transistor has a negative characteristic (which can easily be checked) between its emitter and collector. At around 9 V the base-emitter diode displays the well known avalanche effect. When this occurs the charge carriers in the junction (barrier) layer are so thick and fast that they release further charge carriers. The number of charge carriers grows just like an avalanche and with them so does the current. This corresponds exactly to the same effect in a 9 V Zener diode. The internal resistance of this diode remains positive, however.

The inverse transistor now adds to this effect. The emitter and collector do indeed exchange roles but the symmetrical principle of its construction means that the transistor functions equally in inverse operation. We can measure a slight current gain from about 3 to 10. The transistor still functions due to fact that the charge carriers pass through the thin base layer to reach the junction barrier. And now comes the salient point: it's precisely in this barrier layer that the avalanche effect takes place. There are still more charge carriers, which liberate yet more of them, producing an avalanche squared (so to speak). Once this avalanche is triggered, a weaker voltage is all that's necessary to maintain the effect. The collector current thus amplifies the avalanche effect and assures the negative characteristic.

The strength of the discharge current is sufficient to drive an LED (see **Figure 2**). For this we need nevertheless a voltage greater than 9 V. The circuit functions adequately with two almost dead (discharged) 9 V batteries. The LED will still flash for a long time, right until the very last drop of energy in the batteries. The flashing frequency will slow down as the battery runs down.

For mechanical reasons and to simplify construction, the charge resistor is fitted between the batteries.

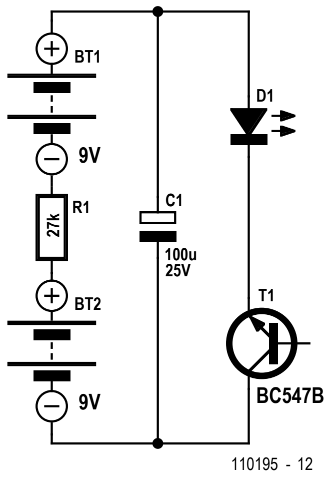


Figure 2. The application.

Chapter 30 • Measure Gamma Rays with a Photodiode

Radiation detector using a BPW34

The first device that springs to mind when thinking about measuring radioactivity is the Geiger-Müller tube. However, these counter tubes are getting hard to find and expensive, and even if you do manage to get hold of one, you will still need to find a way to generate its operating voltage of several hundred volts. It is less well known that even a humble photodiode such as the BPW34 can be used to detect X-rays and gamma radiation.

Ionising radiation is potentially harmful to health, and it is important to minimise one's exposure to it as far as possible. A simple Geiger counter with a small glass mantle tube will not usually be adequate to detect possibly harmful radiation. The semiconductor sensor we describe below also has a relatively low sensitivity, only being able to detect fairly intense sources of radiation, but it is nevertheless an interesting device for carrying out experiments and measurements.

An advantage of using a photodiode is its small sensitive area. The background rate due to cosmic rays is very low and signals from small samples are easier to detect than with a counter tube.

Radiation

When considering protection from radiation it is gamma rays that are the most important. They can penetrate walls and it is difficult to block them. Hard gamma rays are present in the environment all around us and are also not stopped even by a thick wall. Alpha particles, on the other hand, only have a short range and generally cannot even penetrate a sheet of paper: this is the reason that many counter tubes cannot detect them, unless they have a very thin mica window. Beta particles have a longer range and can penetrate thin sheets of metal. Most counter tubes are mainly designed for detecting gamma rays while, within certain limitations, also being sensitive to beta particles.

Diode as detector

The behaviour of a type BPW34 PIN photodiode is similar to that of a low-cost counter tube. Alpha particles will be stopped by the plastic enclosure of the device, whereas gamma rays pass through without problem and create many electron-hole pairs in the diode's depletion layer. If the diode is reverse-biased, almost all of the charge carriers will be drawn away: this corresponds to a small current pulse which can be amplified and processed. Beta particles can also generate such a signal if they are sufficiently energetic to reach the depletion layer.

The amplitude of the signal produced by the photodiode is considerably smaller than that normally obtained from a counter tube, and so a very low-noise instrumentation amplifier circuit is needed.

Another requirement when using a photodiode as a beta and gamma radiation detector is that light must be completely excluded, as otherwise the photocurrent will overwhelm the signal we are looking for. In our prototype we used ordinary aluminium kitchen foil as a screen.

The difference between PIN diodes and PN diodes is that the former include an extra very lightly N-doped region called the 'intrinsic', or 'i' region. This high-resistance region lies between the 'n' and 'p' regions. The result is a wider depletion layer in the diode, and hence a greater volume of semiconductor that can interact with photons. The structure is used in a photodiode in order to obtain as many charge carriers as possible per photon, optimising the device's sensitivity.

Another way to increase sensitivity is to increase the sensitive area of the device. However, this has the disadvantage of increasing its capacitance, which reduces the (voltage) amplitude of its output signal. Commercially-available semiconductor radiation detectors have a large area and a wide intrinsic region. Simple PIN photodiodes such as the BPW34 are less sensitive than these devices, but also of course somewhat cheaper.

The BPW34 and BPX61 photodiodes are practically identical apart from their enclosures. The (cheaper) BPW34 comes in a plastic package, whereas the BPX61 comes in a TO-5 metal enclosure with a glass window. It is possible to remove this window (carefully!) to expose the chip: this will make the diode capable of detecting alpha particles.

The rays or particles must first make it through a 15 µm thick piece of aluminium (the thickness of ordinary kitchen foil). This is no obstacle to gamma rays and beta particles, and alpha particles with an energy of 4 MeV or more will also pass through. When the particle enters the plastic of the photodiode package, deceleration radiation (German: 'bremsstrahlung') will be produced in the form of brief flashes of light, which can also sometimes be detected by the sensor. It is therefore not impossible for even the BPW34 to have some sensitivity to alpha particles.

In principle any semiconductor is sensitive to ionising radiation. It is perhaps less surprising, then, that a photodiode is sensitive to radiation than that the effect has not been widely remarked on before. The effect is however well known in dynamic RAMs, whose stored data can be corrupted by incident radiation. The problem of building electronics to withstand the higher levels of radiation found in space is becoming increasingly difficult, because as structures get smaller it becomes increasingly likely that a single energetic particle can interfere with the operation of a circuit.

Amplifier

In the literature charge amplifiers are usually constructed using a low-noise FET-input opamp as the input stage. Here we take an alternative approach: **Figure 1** shows the circuit of the sensor amplifier. Two transistors are used to amplify the signal from the photodiode. The direct-coupled amplifier automatically sets itself to a mid-range operating point, which gets a good signal-to-noise ratio from the low-noise BC549C transistors.

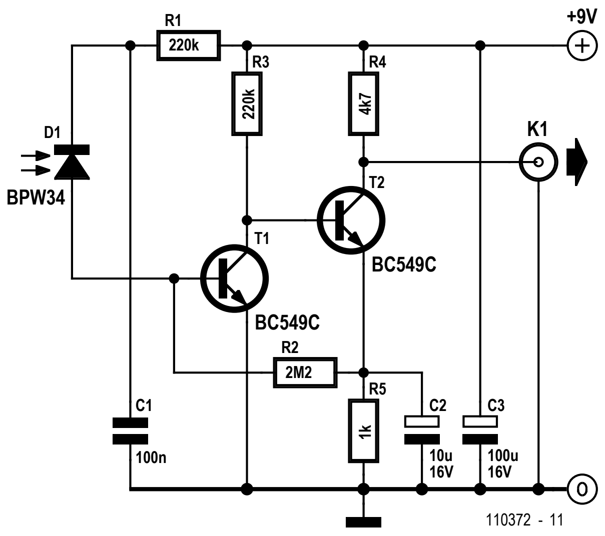


Figure 1. The amplifier circuit.

The transistor input of the amplifier has a comparatively low impedance, which gives good noise matching. As a result of its base-collector capacitance the first stage also operates as an integrator: this turns the brief pulses from the photodiode into longer pulses which can then more easily be amplified.

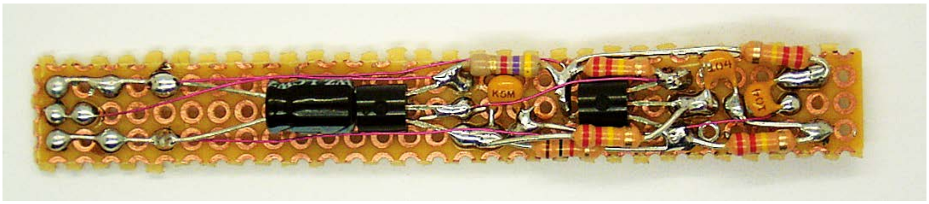


Figure 2. Prototype of the sensor amplifier.

Sensitivity can also be increased by increasing the reverse voltage on the diode. This reduces the capacitance of the diode and increases the size of the depletion layer. The voltage can be as high as 32 V, although the optimum value probably lies somewhat lower: the diode already operates well at 9 V. It is also possible to wire two or more photodiodes in parallel, and that way it is possible to achieve a sensitivity on a par with that of a small counter tube such as the ZP1310.

An oscilloscope can be connected to the output of the circuit to view the signal. Readers who yearn for the clicking sound of a 'real' Geiger counter should consult the text box 'From radiation to sound' for a suitable solution.

Construction

The circuit can be built on a piece of breadboard (see **Figure 2**), with the photodiode on the underside (**Figure 3**). To keep light out of the sensor the whole circuit is wrapped in aluminium foil (**Figure 4**). As mentioned above, ordinary kitchen foil is ideal for this as it is thin enough to let beta particles through. The foil also functions as electrical screening.

To avoid the foil causing short circuits, wrap the board first in insulating tape, leaving a gap for the window of the photodiode. Then wrap the assembly in foil, not forgetting to connect the foil to ground.

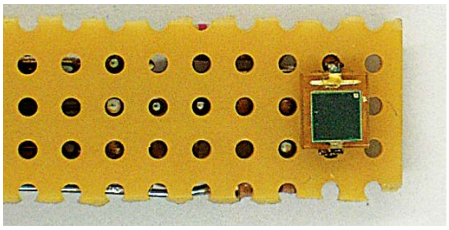


Figure 3. The sensor is on the underside of the board.

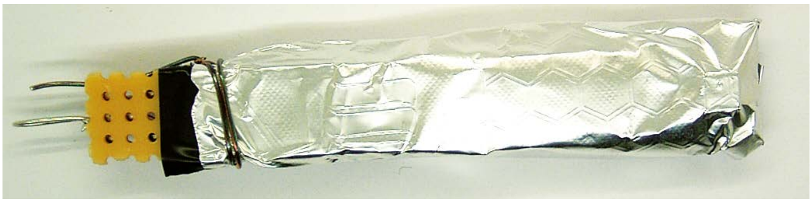


Figure 4. The whole thing is wrapped in aluminium foil.

Experiments and results

The best way to evaluate the results is to use a digital oscilloscope, in AC-coupled mode. A good place to start is with a vertical sensitivity of 50 mV per division and a timebase of 0.2 ms per division. Some oscilloscopes have a persistence mode which allows the results to be accumulated on the display. It is of course also possible to use an analogue oscilloscope.

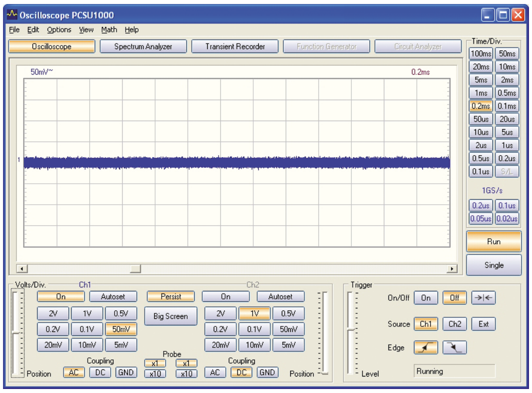


Figure 5. Circuit output in the quiescent state.

In the quiescent state a band of amplified noise about 30 mVpp will be seen (**Figure 5**). A gamma ray hitting the sensor will be seen as a positive pulse with a small negative undershoot following it. If strong negative-going pulses are seen it is a sign that the circuit is not screened well enough and is reacting to RF signals: the radiation we are trying to detect only causes positive-going pulses. **Figure 6** shows the signal accumulated over a period of 30 s with the sensor pointing at an old pocket watch with luminous hands.

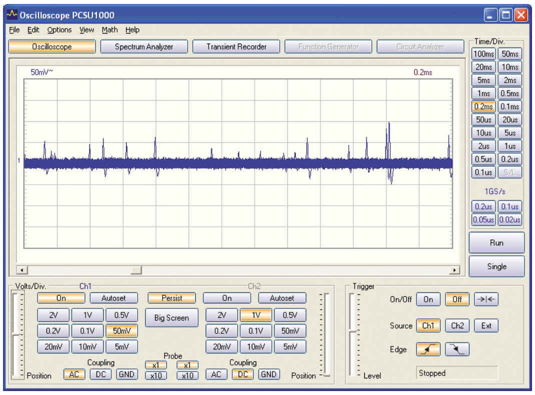


Figure 6. Readings over 30 seconds from an old watch with luminous hands.

Figure 7 shows measurements taken from another radioactive sample, in this case a small piece of pitchblende (uranite), a naturally-occurring ore of uranium. Again the measurements are taken over 30 s. It is easy to see that this sample is more radioactive; but it is also possible to see a difference in the energy distribution. There are more pulses with an amplitude of over 100 mV than in the case of the luminous watch.

This shows that, unlike a Geiger tube, this detector can determine the energy of the individual particles. This in turn lets us make deductions about the types of nuclei that are disintegrating. In the case of pitchblende these will be elements in the uranium decay series; in the case of the luminous watch the decaying nucleus is likely to be radium.

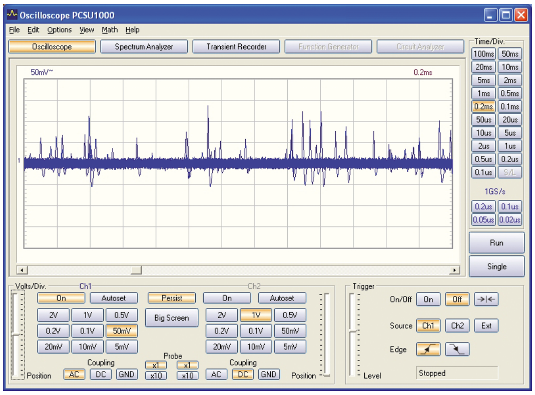


Figure 7. Readings over 30 seconds from a sample of a mineral containing uranium.

The possibility of accumulating readings over a long time period lets us examine samples where we would expect little or no radioactivity. Here the photodiode works better than the counter tube as the background rate is practically zero. With a Geiger counter there are almost always pulses arising from cosmic rays: these hard gamma rays also affect the photodiode sensor, but because of its much smaller sensitive area these events are much rarer, and so it is easier to separate the wanted signal from the background. **Figure 8** shows readings from a sample of galena, a mineral that we would expect not to be radioactive at all. After half an hour, however, we see two clear peaks. We obtained a similar result from a sample of granite, which is known to be slightly radioactive.

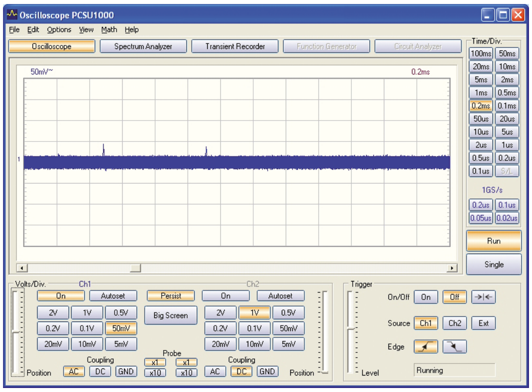


Figure 8. Readings over 30 minutes from a sample of galena.

Certain components and pieces of apparatus manufactured before stricter modern controls were in place can turn out to be radioactive sources. A well-known example is that certain gas discharge lamps and voltage regulator tubes rated under 100 V contain radioactive substances. The author had already had suspicions about an old Russian gas discharge lamp rated at 75 V/3 mA (**Figure 9**). There is a small metal cap welded on to the outer cover, beneath which a strange pill is visible. Beneath this is a tiny hole.

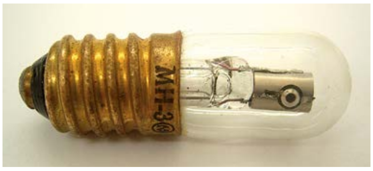


Figure 9. A gas discharge tube with a radioactive ioniser to aid starting.

Taking readings over half an hour (**Figure 10**) revealed impulses with a particularly high energy: all the more surprising given that the radiation had had to penetrate the glass envelope of the tube.

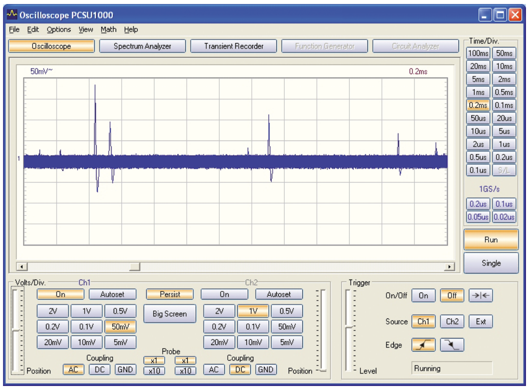


Figure 10. Readings taken from the gas discharge tube shown in Figure 9.

Outlook

We have described the sensor and a simple amplifier for it. If the circuit is built into an enclosure, along with the comparator circuit and loudspeaker described in the text box, the result is a device that can be used in the field, for example to test minerals in a quarry. Combine the comparator with a digital counter, and the overall level of radioactivity can be measured. A sample-and-hold circuit could be added to record energy levels, and the results could be displayed as a kind of energy spectrogram.

Another possibility is to look at taking measurements from other samples over a long period. For example, potassium chloride is a very weak beta emitter. It would be interesting to see if the photodiode can detect it.

Luminous dials

An old watch with a luminous dial is ideal for testing radiation detectors. Alternatively, a suitable alarm clock or compass might be found at a car boot sale.

Radioactive luminous paints were used until about 1965 and watch and clock faces from that time will have lost almost all of their luminosity by now. If you are not sure whether your watch is radioactive, it is possible to carry out a simple test without any electronics: all you need is a magnifying glass. In complete darkness, with your eyes fully accommodated, look at the hands under the glass. If the paint is a radioactive mixture, you will see faint flashing and flickering: you are actually witnessing individual decays. The alpha particles produced excite the luminous paint. If you see no light or an absolutely uniform light, then there is no radioactivity present.

This test is probably only possible with luminous paints based on radioactive materials that have aged considerably, as when the paints are new there may be too many decays happening to see them individually.

From radiation to sound

A ‘real’ Geiger counter makes a pleasant ticking sound. Our diode sensor, on the other hand, is completely silent.

We can remedy the situation with the help of a comparator and a circuit to stretch the pulses so that we can drive a loudspeaker to make clicks. The tested circuit shown here uses a type LM311 comparator which produces a pulse on its output when the amplitude of a pulse on its input exceeds a threshold set by the trimmer. The transistor at the output stretches the pulse to make it audible. The final output can be used to drive headphones, an audio amplifier and a loudspeaker, or a PC-style active speaker.

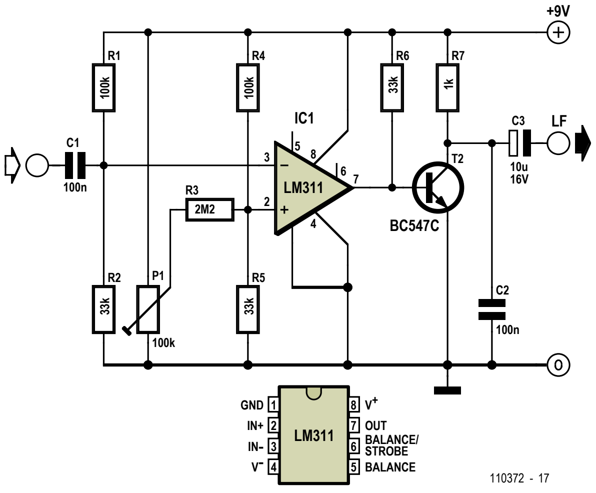


Figure 11. The LM311 comparator which produces a pulse.

Radon decay products

Radioactive samples for testing can be obtained directly from the environment: we are constantly surrounded by radioactive materials. For example, radon is continuously escaping from the ground. This radioactive gas decays (with a moderate half-life) producing further radionuclides, which can be collected.

Take a thin (0.2 mm diameter) piece of enamelled copper wire and stretch it out indoors. Apply a negative potential of between –5 kV and –10 kV to the wire and leave it for ten minutes. Disconnect the high voltage and then run a strip of paper along the length of the wire: you will pick up a dark line of dust that was attracted to the wire by the high voltage. These dust particles are particularly rich in the radioactive decay products of radon. The reason for this is interesting: when the radon decays the new nuclei are moving rapidly, and are therefore stripped of some of their electrons. This means that they have a positive charge and so are attracted to the negatively charged wire.

When this ‘dirty’ strip of paper is held up to the radiation detector, a high level of activity will be recorded. There is no danger, however: if the isotopes had not been picked up by the wire, you would probably have breathed them in instead.

This method will let you determine which rooms in your house have more radon in them. Normally levels will be higher in a cellar or basement as the source of the radon is the earth below.

A suitable high-voltage generator was described as the ‘Air Ioniser’ mini project in the June 2009 edition of Elektor (www.elektormagazine.com/magazine/elektor-200906/19056). It is necessary to extend the design by a further two stages (two capacitors and two diodes) to obtain an output voltage of 5 kV.

References and Internet Links

- Maxim Application Note 2236: Gamma-Photon Radiation Detector
  http://pdfserv.maxim-ic.com/en/an/AN2236.pdf

- Erhan Emirhan and Cenap S. Özben: PIN photodiode-based X- and gamma-ray detectors
  http://thm.ankara.edu.tr/tac/YAZOKULU/yazokulu6/dersler/06-09-2010/erhan-emirhan-cenap-ozben-pin-photodiode.pdf

- C. W. Thiel: An Introduction to Semiconductor Radiation Detectors
  www.sciencemadness.org/talk/files.php?pid=313842&aid=28436

Chapter 31 • Short-Wave Regenerative Receiver

for AM and DRM

Is it possible to make a short-wave regenerative valve receiver so stable that it is even suitable for DRM? And is it possible to do this using a 6-V supply, so only a single voltage is necessary for the filament and anode supply? It looks like it might be possible with an EL95, which has good transconductance even at low anode voltages, although it is actually an outputstage pentode instead of an RF valve.

Besides that, it draws only a modest 200 mA of heater current. Everything can be operated from a small battery, which eliminates any problems with 50-Hz hum. The stability depends entirely on the tuned circuit. Consequently, a robust coil with 20 turns of 1.5-mm wire was wound on a PVC pipe with a diameter of 18 mm. With short leads to the air-dielectric variable capacitor, this yields an unloaded Q factor considerably larger than 300.

The schematic shows a regenerative receiver with feedback via the cathode. The amount of feedback is adjusted using the screen grid voltage. The audio signal across the anode resistor is coupled out via a capacitor. No additional gain is necessary, since the voltage level is suffi- cient for a direct connection to the Line input of a PC sound card. A screened cable should be used for this connection. A two-turn antenna coil is located at the bottom end of the tuned circuit. It provides very loose coupling to the antenna, which is important for good stability.

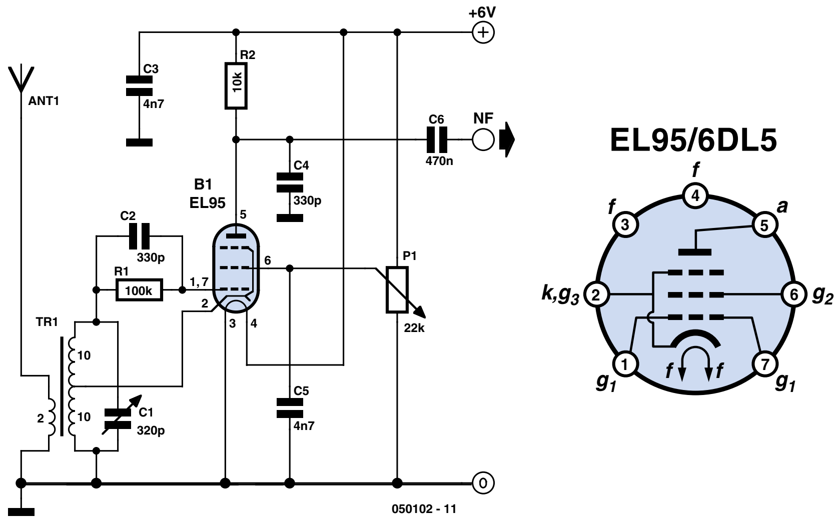


Figure 1. Circuit of a short-wave regenerative valve receiver.

Now it's time to see how it works in practice. Despite the open construction, the frequency drift is less than 1 Hz per minute. That's the way it should be if you want to receive DRM. Quite strong feedback should be used, so the regenerative circuit acts like a direct mixer or a self-oscillating mixer stage.

Every strong DRM signal could be seen using DREAM and tuned to 12 kHz. A total of six different DRM frequencies could be received in the 40-m and 41-m bands. If no good DRM stations are available, the receiver can also pick up AM transmitters. In this case, the amount of feedback should be reduced. The PC can be set aside for AM reception, since all you need is a direct connection to an active PC speaker.

Chapter 32 • DRM Double Superhet Receiver

Using an EF95/6AK5

This receiver arose from the desire to demonstrate that valves are fully capable of holding their own against modern semiconductor devices. Valves often have better large-signal characteristics and less noise. The decisive difference is that the circuit must be designed with higher input and output impedances.

This circuit is built using four EF95 (US equivalent: 6AK5) valves, since this type of valve is small and has proved to operate well with low anode voltages. All four heaters are connected in series and oper ated from a 24-V supply. That makes it attractive to use the same supply for the anode voltage. The achievable gain is fully adequate.

The receiver is designed for the RTL DRM channel at 6095 kHz. It consists of two mixer stages with two crystal oscillators. A steep-edged ceramic filter (type CFW455F) with a bandwidth of 12 kHz provides good IF selectivity. Thanks to the high-impedance design of the circuit, the valves achieve a good overall gain level. The receiver performance is comparable to that of the Elektor Electronics DRM receiver design published in the March 2004 issue, and it can even surpass the performance of the latter circuit with a short antenna, since the tuned input circuit allows better matching to be obtained.

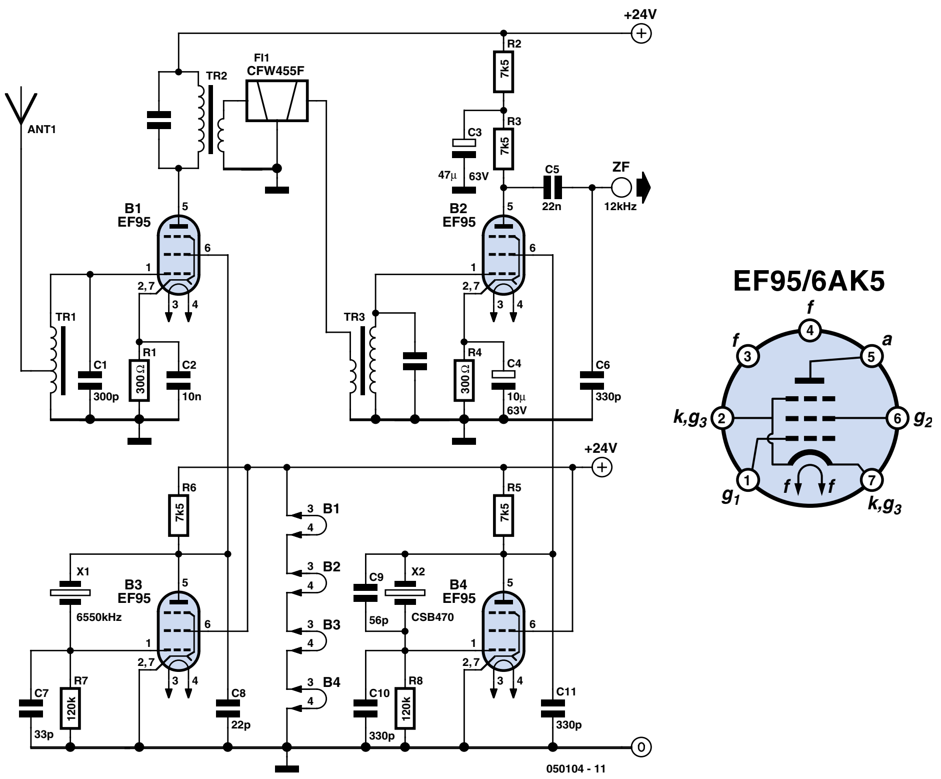


Figure 1. A Valve receiver designed for the RTL DRM channel at 6095 kHz.

The key difficulty is obtaining a suitable crystal with a frequency of 6550 kHz. Old-style FT243 crystals with exactly this frequency are available from American army radio units. However, it takes a bit of luck to obtain a crystal with exactly the right frequency. It's also possible to use a standard crystal with a frequency of 6553.6 kHz, which yields an IF that is 3.6 kHz too high. However, that shouldn't be a problem if a relatively wide-band-width ceramic filter is used. One possibility is the CFW455C, which has a band- width of 25 kHz. If the frequency of the second crystal oscillator remains unchanged, the DRM baseband signal will appear at around 9 kHz, approximately 3 kHz below the nominal value. Nevertheless, the signal can easily be decoded by the DREAM software, since it does not depend on the signal being at 12 kHz. Another possibility would be to use the programmable crystal oscillator design published in the March 2005 issue of Elektor Electronics.

Chapter 33 • Transistor Dip Meter

The dip meter consists of a tunable RF oscillator whose resonant circuit is held in the vicinity of a resonant circuit to be checked. If the frequencies of the two circuits match, the circuit being measured draws energy from the oscillator circuit. This can be measured. This type of meter is also called a 'grid dip meter', since it was originally built using a valve. The amplitude of the voltage on the tuned circuit could be measured from the grid leakage current. Such meters typically have a set of interchangeable coils and several frequency scales.

A meter that can manage with a low voltage of only 1.5 V can be built using a circuit with two transistors. In addition, a coil tap is not required in this design. That makes it easy to connect many different coils to cover a large number of frequency bands.

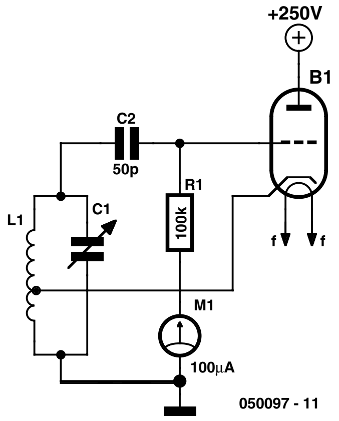


Figure 1. Dip meter using a valve.

If a sufficiently sensitive moving-coil meter is not available, an acoustic signal can be used instead of a pointer display. This involves a sound generator whose frequency increases when its input voltage rises. A resonance dip is then indicated by a falling tone. This acoustic indicator draws less current from the measurement rectifier than a moving-coil instrument. That allows the amplitude of the oscillation to be decreased slightly by reducing the emitter current. This increases the sensitivity, so the dip meter can measure resonant cuits at greater distances.

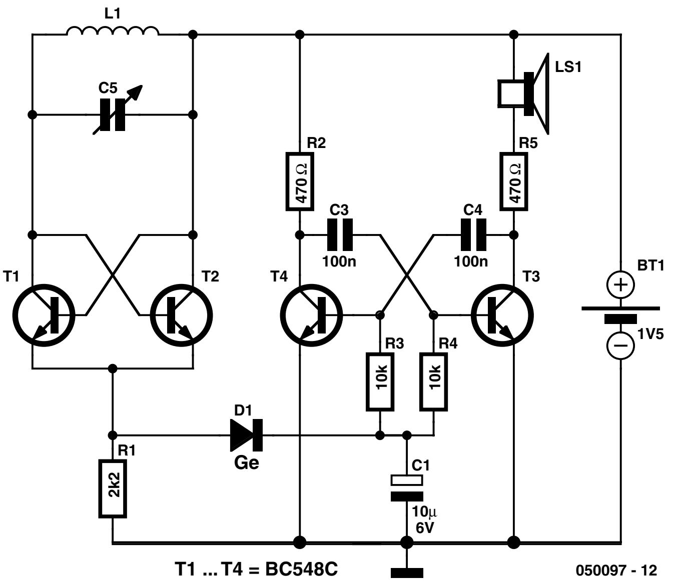


Figure 2. Dip meter with acoustic output.

Chapter 34 • DRM Direct Mixer Using an EF95/6AK5

This hybrid DRM receiver with a single valve and a single transistor features good large-signal stability. The EP95 (US equivalent: 6AK5) acts as a mixer, with the oscillator signal being injected via the screen grid. The crystal oscillator is built around a single transistor. The entire circuit operates from a 6-V supply. The receiver achieves a signal-to-noise ratio of up to 24 dB for DRM signals. That means the valve can hold its own against an NE612 IC mixer.

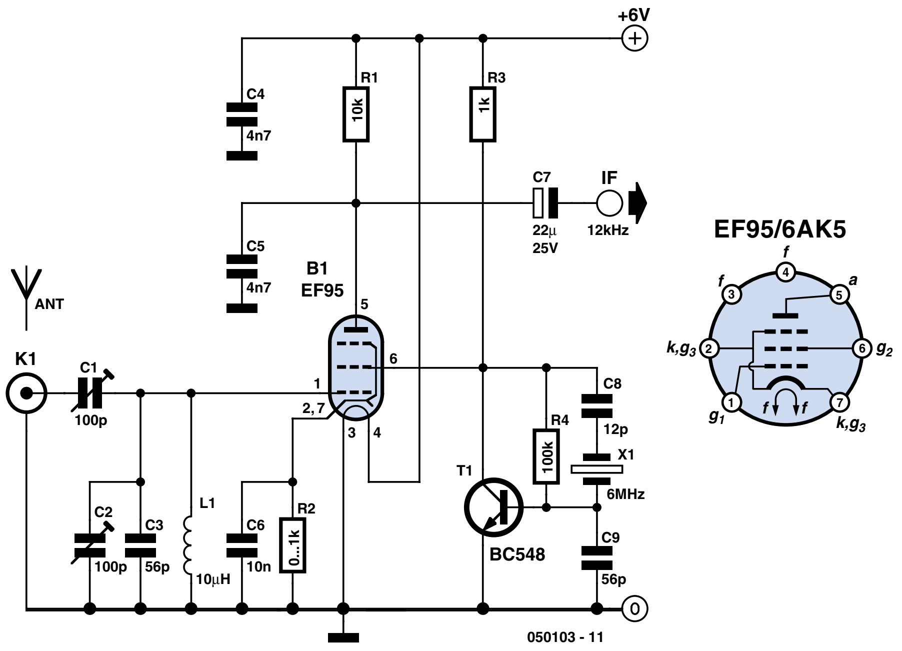


Figure 1. Hybrid DRM receiver with a single valve and a single transistor.

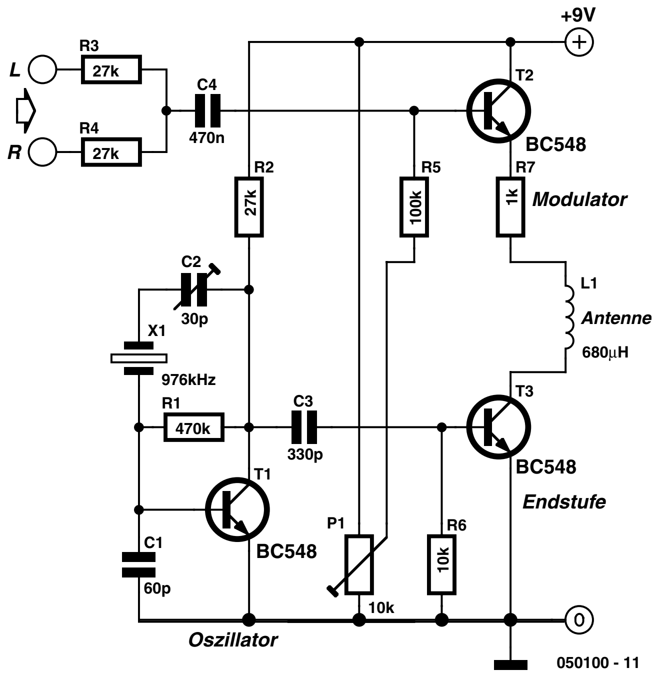
The component values shown in the schematic have been selected for the RTL2 DRM channel at 5990 kHz. That allows an inexpensive 6-MHz crystal to be used. The input circuit is built using a fixed inductor. Two trimmer capacitors allow the antenna matching to be optimised. The operating point is set by the value of the cathode resistor. The grid bias and input impedance can be increased by increasing the value of the cathode resistor. However, good results can also be achieved with the cathode connected directly to ground.

Chapter 35 • Medium-Wave Modulator

If you insist on using a valve radio and listening to medium-wave stations, you have a problem: the existing broadcasters have only a limited number of records. Here there's only one remedy, which is to build your own medium-wave transmitter. After that, you can play your own CDs via the radio.

The transmitter frequency is stabilised using a 976-kHz ceramic resonator taken from a TV remote control unit. Fine tuning is provided by the trimmer capacitor. If there's another station in the background, which will probably be weak, you can tune it to a heterodyne null, such as 981 kHz. As an operator of a medium-wave transmitter, that's your obligation with respect to the frequency allocations. And that's despite the fact that the range of the transmitter is quite modest. The small ferrite coil in the transmitter couples directly into the ferrite rod antenna in the radio.

The modulator is designed as an emitter follower that modulates the supply voltage of the output amplifier. As the medium-wave band is still mono, the two Medium-Wave Modulator input channels are merged. The potentiometer can be adjusted to obtain the least distortion and the best sound. The RF amplifier stage has intentionally been kept modest to prevent any undesired radiation. The quality of the output signal can also be checked using an oscilloscope. Clean amplitude modulation should be clearly visible.



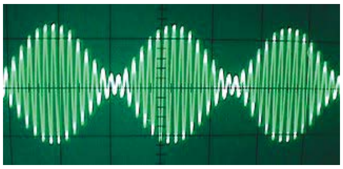


Figure 1. Oscillator and modulation stage.

The medium-wave modulator can simply be placed on top of the radio. A signal from a CD player or other source can be fed in via a cable. Now you have a new, strong station on the radio in the medium- wave band, which is distinguished by good sound quality and the fact that it always plays what you want to hear.

Chapter 36 • EE-ternal Blinker

You occasionally see advertising signs in shops with a blinking LED that seems to blink forever while operating from a single battery cell. That's naturally an irresistible challenge for a true electronics hobbyist…

And here's the circuit. It consists of an astable multivibrator with special properties. A 100-µF electrolytic capacitor is charged relatively slowly at a low current and then discharged via the LED with a short pulse. The circuit also provides the necessary voltage boosting, since 1.5 V is certainly too low for an LED.

The two oscillograms demonstrate how the circuit works. The voltage on the collector of the PNP transistor jumps to approximately 1.5 V after the electrolytic capacitor has been discharged to close to 0.3V at this point via a 10-kΩ resistor. It is charged to approximately 1.2 V on the other side. The difference voltage across the electrolytic capacitor is thus 0.9 V when the blink pulse appears. This voltage adds to the battery voltage of 1.5 V to enable the amplitude of the pulse on the LED to be as high as 2.4 V. However, the voltage is actually limited to approximately 1.8 V by the LED, as shown by the second oscillogram. The voltage across the LED automatically matches the voltage of the LED that is used. It can theoretically be as high as 3 V.

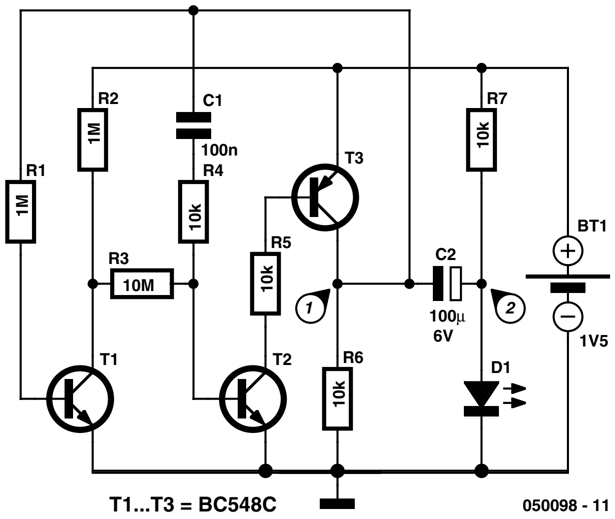
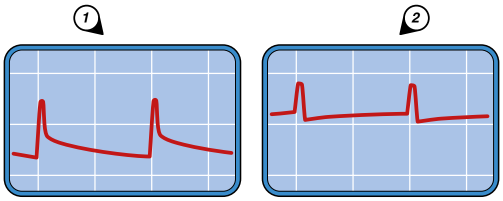


Figure 1. Pulse generator with voltage doubling.



The circuit has been optimised for low-power operation. That is why the actual flip-flop is built using an NPN transistor and a PNP transistor, which avoids wasting control current. The two transistors only conduct during the brief interval when the LED blinks. To ensure stable operating conditions and reliable oscillation, an additional stage with negative DC feedback is included. Here again, especially high resistance values are used to minimise current consumption.

The current consumption can be estimated based on the charging current of the electrolytic capacitor. The average voltage across the two 10-kΩcharging resistors is 1 V in total. That means that the average charging current is 50 µA. Exactly the same amount of charge is also drawn from the battery during the LED pulse. The average current is thus around 100 µA. If we assume a battery capacity of 2500 mAh, the battery should last for around 25,000 hours. That is more than two years, which is nearly an eternity. As the current decreases slightly as the batter voltage drops, causing the LED to blink less brightly, the actual useful life could be even longer. That makes it more than (almost) eternal.

Chapter 37 • Short-Wave Superregenerative Receiver

Superregenerative receivers are characterised by their high sensitivity. The purpose of this experiment is to determine whether they are also suitable for short-wave radio.

Superregenerative receivers are relatively easy to build. You start by building a RF oscillator for the desired frequency. The only difference between a superregenerative receiver and an oscillator is in the base circuit. Instead of using a voltage divider, here we use a single, relatively high-resistance base resistor (100 kΩ to 1 MΩ). Superregenerative oscillation occurs when the amplitude of the oscillation is sufficient to cause a strong negative charge to be applied repeatedly to the base. If the regeneration frequency is audible, adjust the values of the resistors and capacitors until it lies somewhere above 20 kHz. The optimum setting is when you hear a strong hissing sound. The subsequent audio amplifier should have a low upper cutoff frequency to strongly attenuate the regeneration signal at its output while allowing signals in the audio band to pass through.

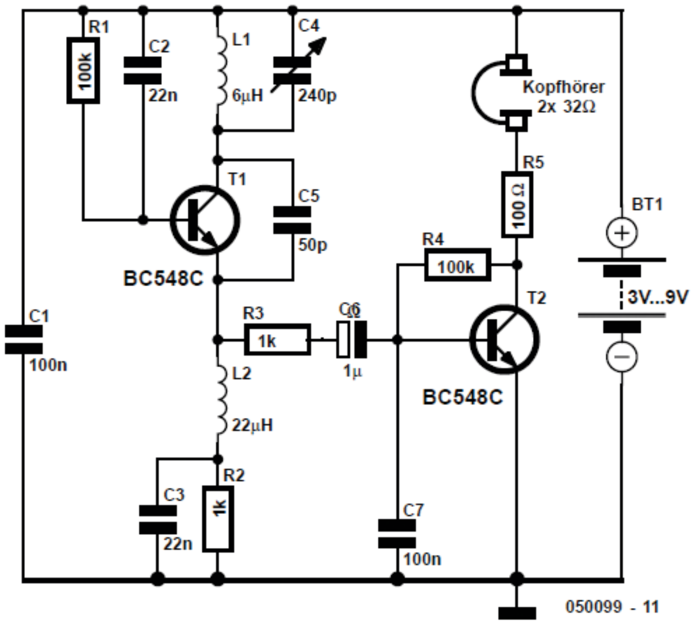


Figure 1. This experimental receiver uses two transistors.

This experimental circuit uses two transistors. A Walkman headphone with two 32-Ω earphones forms a suitable output device. The component values shown in the schematic diagram have proven to be suitable for the 10–20 MHz region. The coil consists of 27 turns wound on an AA battery serving as a winding form. The circuit produces a strong hissing sound, which diminishes when a station is received. The radio is so sensitive that it does not require any antenna to be connected. The tuned circuit by itself is enough to receive a large number of European stations. The circuit is usable with a supply voltage of 3 V or more, although the audio volume is greater at 9 V. One of the major advantages of a superregenerative receiver is that weak and strong stations generate the same audio level, with the only difference being in the signal to noise ratio. That makes a volume control entirely unnecessary. However, there is also a specific drawback in the short-wave bands: interference occurs fairly often if there is an adjacent station separated from the desired station by something close to the regeneration frequency. The sound quality is often worse than with a simple regenerative receiver. However, this is offset by the absence of the need for manual feedback adjustment, which can be difficult.

Chapter 38 • Short-Wave Converter

This short-wave converter, which doesn't have a single coil requiring alignment, is intended to enable simple medium-wave receivers to be used to listen to short-wave signals. The converter transforms the 49-m short-wave band to the medium-wave frequency of 1.6 MHz. At the upper end of the medium-wave band, select an unoccupied frequency that you want to use for listening to the converted short-wave signals. Good reception performance can be obtained using a wire antenna with a length of one to two metres.

The converter contains a free-running oscillator with a frequency of around 4.4 MHz, which is tuned using two LEDs (which act as variable-capacitance diodes!) and a normal potentiometer. The frequency range is set by adjusting Short-Wave Converter the emitter current using a 1-kΩ trimpot. The oscillator frequency depends strongly on the operating point. This is due to the combination of using an audio transistor and the extremely low supply voltage. Under these conditions, the transistor capacitances are relatively large and strongly dependent on the operating point.

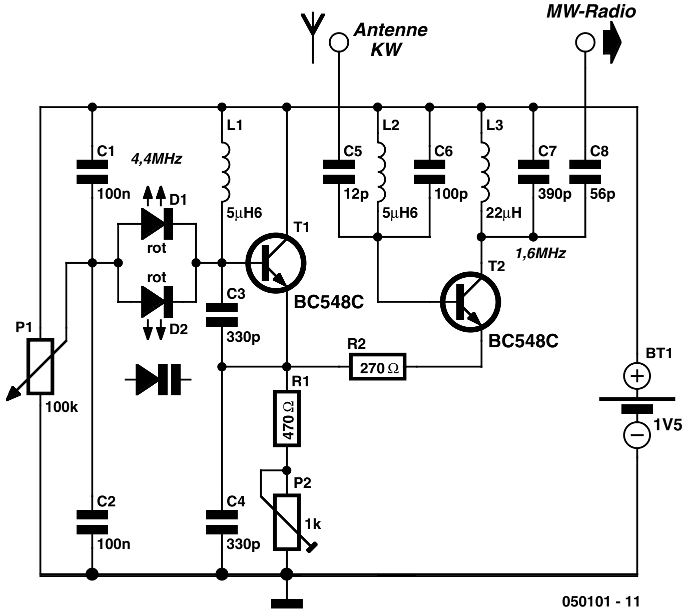


Figure 1. Oscillator and mixer.

The second transistor forms the mixer stage. If you calculate the resonant frequencies of the tuned circuits, you will obtain 6.7 MHz for the antenna circuit and 1.7 MHz for the output circuit. Additional transistor capacitances and the effects of the coupling capacitors shift each of the resonant frequencies downward. The tuned circuits are relatively heavily damped to obtain bandwidths that are large enough to allow the circuit to be used without any specific alignment. The results are good despite the low collector–emitter voltage of around only 0.6 V, due to the fact that only a modest amount of mixer gain is necessary. The entire circuit also draws less than 1 mA.

Part 2 Microcontroller

Chapter 39 • Microcontroller BootCamp (1)

Arduino and Bascom

In the Editorial Office we receive a fair number of requests from electronics enthusiasts who are looking for an easy way to get started with microcontrollers. It's been a good while since we last ran a large series of articles on this subject in Elektor, and the associated hardware is vintage, so it's high time to run a new series. This series is aimed at our readers who already have some experience with analog electronics and now want to start using microcontrollers in their own circuits.

You might ask why you should use a microcontroller when it's possible to do so much with ordinary analog electronics. At first glance, this looks like a good question. New microcontrollers with more features, higher performance, higher clock rates and even more memory are appearing all the time, but the first demo program for every one of them invariably makes a LED blink. This leads to the justified criticism that you could get the same result by simply taking an NE555 timer IC and adding a couple of resistors and a capacitor. That's absolutely right, and the comparison is better than you might think because a lot of the elements of an NE555 timer IC can also be found in a microcontroller.

For comparison: the NE555 timer IC

First of all there's the output. When you look at the internal block diagram of the NE555 on the data sheet (**Figure 1**), you can see that it has a push-pull output stage that can actively switch high and low. Microcontrollers have exactly the same kind of outputs. They are called ports, where the name "port" can stand for a set of outputs or for pins that can be configured either as inputs or as outputs. The circuitry connected to the output for a LED blinker application is also the same: a LED with a series resistor, connected either to ground (GND) or to the supply voltage (Vcc). The NE555 also has a second output driven by an open-collector transistor, which can only switch something to ground. Many microcontrollers can also emulate this function. Actually the only difference is that microcontrollers usually have several outputs but the NE555 has only one, since the push-pull output and the open-collector output are not independent.

Next we have the inputs of the NE555, which consist two inputs to a comparator that controls an internal flip-flop. A typical application for this is an astable multivibrator, which hobbyists often call a blinker circuit. You can put all this together in almost no time. You just plug the numbers into a couple of formulas to determine the right component values, and then the NE555 does exactly what you want. Determining the component values for an NE555 circuit and programming a microcontroller are actually comparable tasks. There's another thing that's very similar: both devices (NE555 and microcontroller) have a Reset input that you can use to set everything back to the starting point. And with both devices the Reset input is usually high in the quiescent state and must be actively pulled to ground.

Figure 2 shows a simple square-wave generator in the form typically used as an LED blinker. The NE555 data sheet also shows several other basic circuits, including a monostable and a pulse-width modulator—all of which are typical tasks for microcontrollers. If you further consider the countless NE555 applications you can find somewhere on the

Internet, you certainly have to agree that in many cases all you need is a 555 timer IC and a microcontroller is overkill. For everything from light curtains to servo controls or processing analog sensor signals, there are many things that can be done with this IC and similar devices. And you can be sure that there are still lots of potential applications that haven't been worked out yet.

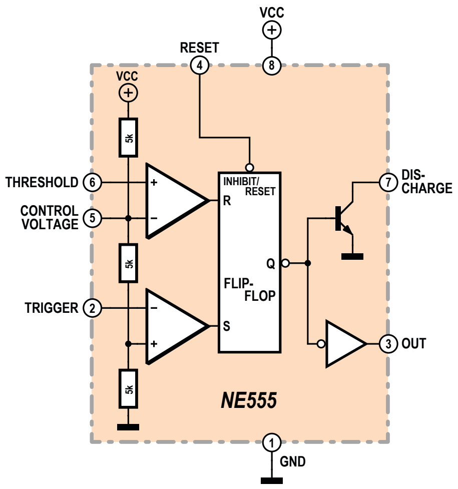


Figure 1. Block diagram of the NE555.

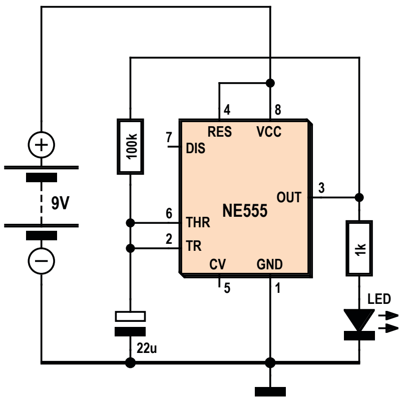


Figure 2. Blinker circuit with the NE555

Reducing development time

The similarities between the NE555 and a microcontroller are summarized in **Table 1**. You could formulate the result of a fair comparison as follows: The microcontroller has a bit more of everything and is therefore generally the better choice for complex tasks. For example, a single microcontroller could probably handle a task that would take ten NE555s. Above a certain level of complexity, the microcontroller solution is also smaller and cheaper than the analog solution. On the other hand, an analog electronics solution is a better and more economical choice for quite a few simple tasks.

Table 1: Comparison of NE555 and microcontroller.

NE555	Mikrocontroller ATmega
Switching output	Port outputs
Inputs	Port inputs
Timing control by RC networks	Internal timer driven by crystal controlled clock
Reset input	Reset input
Comparator inputs	Comparator; analog inputs
PWM function	PWM outputs
Analoge Signalverarbeitung	Analog signal processing
Flip-flop	Memory cells

Anyone with a bit of experience in microcontroller development can also mention another advantage of microcontrollers: once the circuit is complete, you don't need to touch the soldering iron again. From that point on, all you do is write and test code. Changes to device functions can be implemented and tested very quickly. Microcontrollers are general-purpose and versatile computation workhorses. The learning curve is worth the effort, since you ultimately save time.

Your first exposure to a microcontroller data sheet may put you off, since it can easily amount to 300 pages or more. Fortunately, there are approaches to the world of microcontrollers that do not require you to understand everything right up front. In many programming languages the first steps are very easy. You just need to have enough confidence to try something even when you don't entirely understand how it works. The main thing is to have a couple of positive experiences at the beginning, and after that it just goes automatically. That's doubtless the reason why the first demo task for every microcontroller is a LED blinker. Here as well we remain true to this tradition, if only to maintain the comparison with the NE555.

Arduino and Bascom

Microcontrollers actually come in all sorts and sizes. At the start of a series such as this we are faced with the choice of which system to use, and there are many different options. We spent a long time talking about which microcontroller, which circuit board (existing or new), and which programming language we could specifically recommend for beginners. Our discussions led to the following proposal: hardware: Arduino Uno; software: Bascom.

Arduino has become the most popular system in the hobby environment. The programs are developed on a PC using a simple programming language, and they can be downloaded directly to the microcontroller over a USB connection. A large variety of boards and extension boards (Arduino shields) are also available at amazingly low prices. The entire system is based on the open-source concept, so the software and the hardware are fully documented. The low-cost Arduino Uno board [1] is a good choice for our course because it is equipped with a widely used microcontroller. The ATmega328 is an AVR microcontroller made by Atmel. Programming this device is quick and quite easy. The microcontroller has enough memory to allow relatively large programs to be executed later on (see **Table 2**). There is a dedicated development environment available for the Arduino. A development environment in this context, also called an integrated development environment (IDE), is a PC program that is used to develop programs for a microcontroller. In the development environment, the microcontroller programs are typed in using an editor and then converted into bytes by the compiler, after which they are downloaded to the microcontroller.

To ensure that the compiler understands what the microcontroller is supposed to do, you must adhere to the syntax rules of a particular programming language when you write the program. The Arduino IDE uses a simple version of the C programming language, which is the language used by most professional programmers. It also has several special commands that make program development easier. Many users have become familiar with this programming language and have no trouble working with it. Nevertheless, for this course we chose the Basic programming language. There are many reasons for this choice. One is the Bascom Basic compiler for AVR microcontrollers, which is widely used and popular. The learning curve is especially easy, in part because Bascom includes many special commands for sending characters from a microcontroller to a PC, for showing letters and numbers on a display, and much more. However, perhaps the most important reason for using Bascom is that it makes you fit for all AVR controllers. The selected Arduino board (**Figure 3**) is used here as a learning platform, but in the end you can use any desired AVR microcontroller on other commercially available boards or on your own boards. You also don't have to limit yourself to ATmega devices. For example, you may be able to manage with the compact ATtiny13, which has only eight pins.

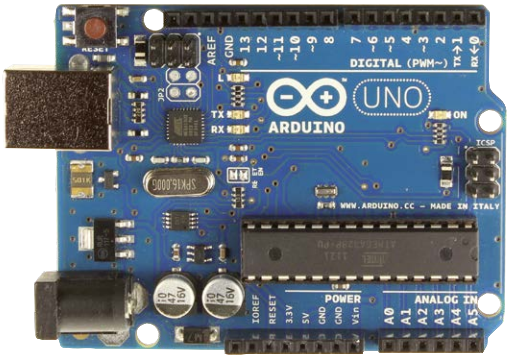


Figure 3. The Arduino Uno board.

A special feature of the Arduino board is the USB port. In addition to downloading programs to the microcontroller as described below, you can use it to send characters back and forth between the PC and the ATmega328. Among other things, you can use this to control the microcontroller remotely from a PC program or to send measurement values captured from sensors by the microcontroller to the PC and display them on the PC. Another thing is that the board can be powered over USB if you so wish. All you need is a USB cable, and you're ready to go.

Your first program

One option for putting together a blinker circuit with an ATmega328 is shown in **Figure 4**. To make it easier to see what matters here, the circuit diagram shows the microcontroller without all the peripheral circuitry of the Arduino board. The LED is already present on the Arduino board, so you don't have to put anything together.

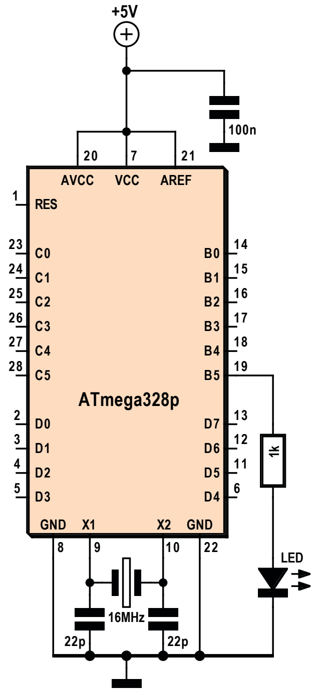


Figure 4. linker circuit with a microcontroller.

Similar simplified circuit diagrams are also used for the other example applications described later on. The idea behind this is that you can also try out all of these examples using a bare microcontroller, for example on a breadboard or on a piece of prototyping board. This means that if you wish, you can follow the entire course without the Arduino board. However, it's more convenient and easier for beginners to use the Arduino Uno board.

```
'------------------------------------
'Uno_LED1.BAS

'------------------------------------
$regfile = "m328pdef.dat" 'ATmega328p
$crystal = 16000000       '16 MHz
'------------------------------------

Config Portb = Output
```

```
Do
  Portb.5 = 1      'LED on
  Waitms 500       '500 ms
  Portb.5 = 0      'LED off
  Waitms 500       '500 ms
Loop
```

Listing 1. LED blinker

Listing 1 shows the Bascom program for the LED blinker. It starts off with two directives for the compiler, which define what is called the environment. This is necessary because the Bascom compiler needs to know the target microcontroller for compiling the program. The Arduino Uno is equipped with an ATmega328P, so the directive is:

```
$regfile = "m328pdef.dat"
```

You also have to tell the compiler what clock speed the microcontroller runs at. The higher the clock speed, the faster the microcontroller executes the instructions. This can be important when dealing with things that require exact timing. An example is sending characters to the PC at a particular data rate. The board has a 16-MHz crystal that sets the clock speed of the controller.

The corresponding directive for the compiler is:

```
$crystal = 16000000
```

Before you get to the actual program, there's another important configuration issue: the port pins of the microcontroller can be used as inputs or as outputs. All port pins are initially configured as inputs after a restart. Here we need an output, so the corresponding port must be configured accordingly:

```
Config Portb = Output
```

This instruction configures all six pins belonging to port B as outputs. The program actually only uses port pin PB5; all of the other pins remain unused. Why did we choose PB5 in particular? Because a yellow LED is already connected to this pin on the Arduino Uno board.

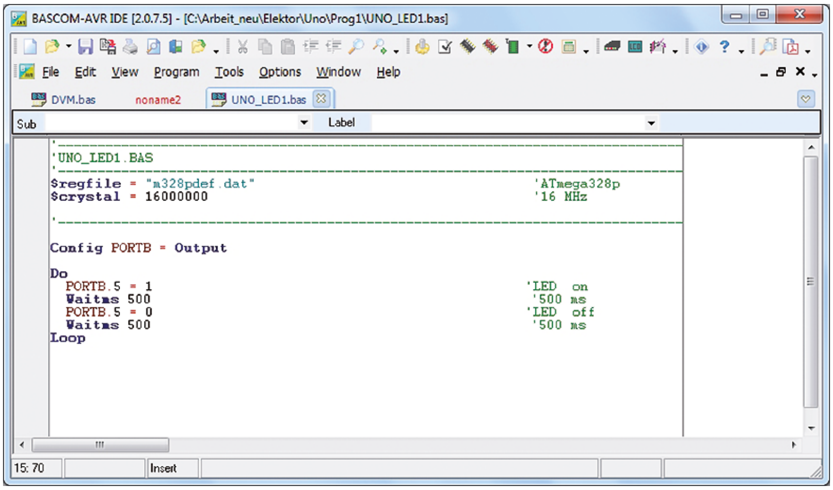
```
Portb.5 = 1
Waitms 500
Portb.5 = 0
Waitms 500
```

The individual line to the LED is switched to the "high" voltage level (close to the supply voltage of the microcontroller) by the instruction *Portb.5 = 1*. This causes a current to flow through the LED. The port pin is switched back to the "low" voltage level (close to the ground level) by the instruction *Portb.5 = 0*. There is also a wait instruction to delay program execution. *Waitms 500* is self-explanatory; it causes a time delay of 500 ms. All of this is built into a loop structure. Everything between “Do” and “Loop” will be repeated indefinitely. The only way to stop the program is to switch off the supply voltage or press the Reset button.

Software: the compiler

So far, so good. But how is the program converted into an executable form, and how do you get it into the microcontroller? The Bascom Basic compiler was developed by Mark Alberts for the 8051 family and for AVR microcontrollers. First you have to get a copy of this PC software. It is available from the website of MSC Electronics [2]. You can opt for the paid full version or the free demo version. The demo version is fully adequate for trying out the software and for getting starting with programming. It is limited to programs that occupy up to 4 KB (4,096 bytes) after compilation. By comparison, the Arduino has room for up to 32 KB. However, 4 KB is not peanuts; it takes a fair amount of programming effort to fill it up. Most of the examples in this series will be much smaller.

Installing Bascom is very easy. After you launch the program, the first thing you see is an empty Editor window where you can type in your own program. Of course, you can also import an existing program into the Editor (see **Figure 5**). These program files (with the suffix .bas) are plain text files that can also be viewed with Windows Notepad or another editor program. This is called “source code” because the compiler uses the contents of these text files as the source for compilation. The compiled program is called “hex code” because the bytes are often shown in hexadecimal notation, always with two hex characters in the range 0–9 and A–F per byte. Numbers in the form of bits and bytes and the various notations used for them will be a frequent topic in this series on microcontroller programming.



```
'-----------------------------------------------------------------
'UNO_LED1.BAS
'-----------------------------------------------------------------
$regfile = "m328pdef.dat"                        'ATmega328p
$crystal = 16000000                              '16 MHz
'-----------------------------------------------------------------

Config PORTB = Output

Do
  PORTB.5 = 1                                   'LED on
  Waitms 500                                     '500 ms
  PORTB.5 = 0                                   'LED off
  Waitms 500                                     '500 ms
Loop
```

Figure 5. The first program in the Bascom editor.

You can type in the program in Listing 1 yourself or download it from the Elektor page. The file UNO_LED1.bas is located in the zip folder [3]. Compiling the program is very easy: simply click Program/Compile, click the corresponding icon on the toolbar (a black IC), or press the F7 key. If there is any sort of error in the source text, an error message will be displayed. If there are no problems, a pop-up window shows what percentage of the memory is occupied. In this case it is less than 1%, which is rounded down to 0%.

What matters is the resulting hex file UNO_LED1.hex or the binary file UNO_LED1.bin, which are two different file formats with the same content. They contain the executable code for the microcontroller. Now you have to load this code into the microcontroller's flash memory. There are many ways to do this, and for now we only describe the simplest way. Other options will be described in subsequent instalments.

The simplest way: use the boot loader

This program must be located in the microcontroller's flash ROM in order to run on the microcontroller. Flash ROM is a special type of memory, similar to EEPROM, in which electrical charges ensure that the memory contents are retained reliably for many decades. Flash ROM can be rewritten repeatedly ("flash" refers to very fast writing), so programs stored in flash memory can always be altered at a later date. Special programming devices are available for programming flash memory. They are connected to specific pins of the microcontroller, and the program bytes are received from the development environment on the PC via the USB link.

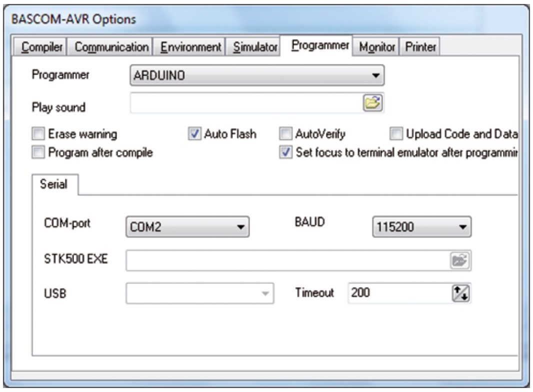
However, this can also be done without a programming device. When the Arduino board has a USB connection to the PC as mentioned above, the program bytes can be sent to the microcontroller over the USB link as long as there is a small program running in the microcontroller that receives the data and writes it to the microcontroller's flash memory. This small program is called a boot loader, which is related to the expression "booting up" for starting up a computer. This term originates from the word "bootstrap" in the saying "pull yourself up by your bootstraps". Of course, pulling yourself out of the mud by tugging on your bootstraps doesn't work in practice, and likewise a microcontroller with nothing in its program memory cannot program itself. However, this is possible if a boot loader is already present in memory, and the Arduino comes with a built-in boot loader.

The development environment on the PC also has to be able to download program code using the Arduino boot loader. Fortunately, Bascom developer Mark Alberts already guessed that someone would want to program an Arduino board in Bascom at some point in time, so this capability is already incorporated. After you configure the right settings, everything is quite easy. Here we describe how this works with the demo version, since there is a small difference with the full version.

First you have to install the original Arduino software available from [4], which includes the USB driver for the Arduino Uno board. Then you connect the Arduino Uno over a USB cable. The driver is loaded automatically, after which the Arduino Uno should be visible in the Windows Device Manager window. There you can see which COM port number (e.g. COM2 or COM3) has been assigned to the Arduino Uno. If you wish you can change the COM port number in Device Manager, but this is usually not necessary. However, you should note or write down the COM number. If you wish, you can also open the Arduino IDE and try out a couple of sample programs. However, here we want continue straightaway with Bascom.

In Bascom you can choose from a large variety of programming devices and boot loaders under the menu item Options/Programmer. The right setting is “ARDUINO” (not “Arduino STK500/2”), as shown in **Figure 6**. It’s also important to configure the right COM port (in this example COM2), and it’s essential to configure the right baud rate (115,200). This is because both parties must agree on the data transmission rate in bits per second (baud). We’ll come back to this subject later on in the course when we talk about sending messages and measurement data to the PC.

If you only go by the Bascom help and the examples included with Bascom, some of your settings will be wrong here because the Arduino Uno is relatively new and the Arduino camp has only recently changed to the highest standard transmission rate of 115,200 baud. All in the interest of fast data transfer and correspondingly super-fast device programming. Time is scarce in our fast-paced era.



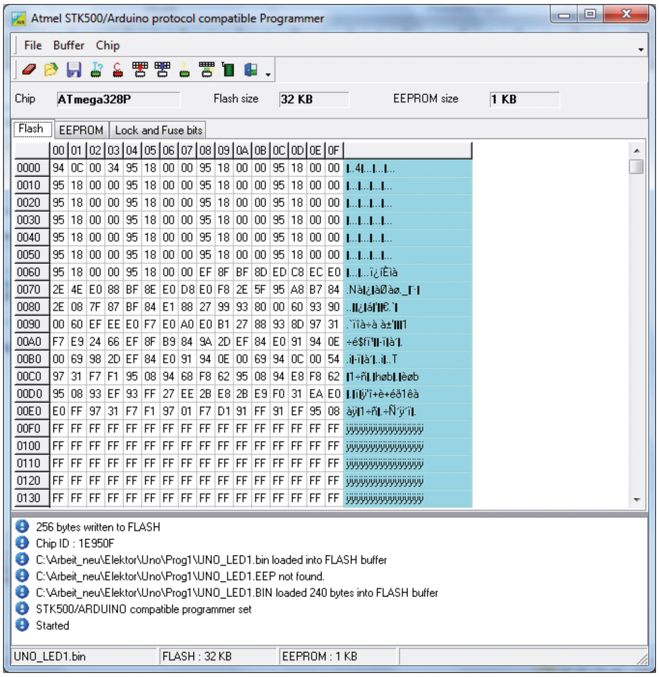
Programmer: ARDUINO
Play sound:
[ ] Erase warning
[✓] Auto Flash
[ ] AutoVerify
[ ] Upload Code and Data
[ ] Program after compile
[✓] Set focus to terminal emulator after programmir

COM-port: COM2
BAUD: 115200
STK500 EXE:
USB:
Timeout: 200

Figure 6. The settings for the Arduino boot loader.

In **Figure 6** you can see that the option “Auto Flash” has also been ticked. This saves a mouse click later on, and it reduces the risk of doing something wrong. The “Terminal Emulator” option is also very helpful if you want to have programs send messages and data to the PC later on. Once the right settings have been selected, close the window with ESC. That’s the previously mentioned difference between the demo version and the full version, which has an “OK” button. If you don’t know about using the ESC key, it looks like the program is stuck. Incidentally, ESC also works with the full version.

Now you’re done, and it’s time for action. As previously mentioned, press F7 to compile the source code. Then press F4 to start the programmer. Alternatively you can click the small green PCB symbol “Program Chip”; the result is the same. In any case, a new Programmer window opens. There the compiled program (UNO_LED1.bin) is already nicely loaded and displayed in the form of hexadecimal numbers (**Figure 7**).


Atmel STK500/Arduino protocol compatible Programmer

File Buffer Chip

Chip ATmega328P Flash size 32 KB EEPROM size 1 KB

Flash | EEPROM | Lock and Fuse bits

```
      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000  94 0C 00 34 95 18 00 00 95 18 00 00 95 18 00 00  ▌..4▌...▌...▌...
0010  95 18 00 00 95 18 00 00 95 18 00 00 95 18 00 00  ▌...▌...▌...▌...
0020  95 18 00 00 95 18 00 00 95 18 00 00 95 18 00 00  ▌...▌...▌...▌...
0030  95 18 00 00 95 18 00 00 95 18 00 00 95 18 00 00  ▌...▌...▌...▌...
0040  95 18 00 00 95 18 00 00 95 18 00 00 95 18 00 00  ▌...▌...▌...▌...
0050  95 18 00 00 95 18 00 00 95 18 00 00 95 18 00 00  ▌...▌...▌...▌...
0060  95 18 00 00 95 18 00 00 EF 8F BF 8D ED C8 EC E0  ▌...▌...ï¿íÈìà
0070  2E 4E E0 88 BF 8E E0 D8 E0 F8 2E 5F 95 A8 B7 84  .Nà▌¿▌àØàø._▌¨·▌
0080  2E 08 7F 87 BF 84 E1 88 27 99 93 80 00 60 93 90  ..▌▌¿▌á▌'▌▌€.`▌
0090  00 60 EF EE E0 F7 E0 A0 E0 B1 27 88 93 8D 97 31  .`ïîà÷à àà±'▌▌▌1
00A0  F7 E9 24 66 EF 8F B9 84 9A 2D EF 84 E0 91 94 0E  ÷é$fï▌¹▌▌-ï▌à'▌.
00B0  00 69 98 2D EF 84 E0 91 94 0E 00 69 94 0C 00 54  .i▌-ï▌à'▌..i▌..T
00C0  97 31 F7 F1 95 08 94 68 F8 62 95 08 94 E8 F8 62  ▌1÷ñ▌.▌høb▌.▌èøb
00D0  95 08 93 EF 93 FF 27 EE 2B E8 2B E9 F0 31 EA E0  ▌.▌ï▌ÿ'î+è+éð1êà
00E0  E0 FF 97 31 F7 F1 97 01 F7 D1 91 FF 91 EF 95 08  àÿ▌1÷ñ▌.÷Ñ'ÿ'ï▌.
00F0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0100  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0110  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0120  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
0130  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF  ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
```

```
256 bytes written to FLASH
Chip ID : 1E950F
C:\Arbeit_neu\Elektor\Uno\Prog1\UNO_LED1.bin loaded into FLASH buffer
C:\Arbeit_neu\Elektor\Uno\Prog1\UNO_LED1.EEP not found.
C:\Arbeit_neu\Elektor\Uno\Prog1\UNO_LED1.BIN loaded 240 bytes into FLASH buffer
STK500/ARDUINO compatible programmer set
Started
```

UNO_LED1.bin | FLASH : 32 KB | EEPROM : 1 KB

Figure 7. The compiled program in the form of hex numbers.

Good job: it works!

After this you can sit back and relax; the rest is automatic. However, you can also do everything yourself. If you opt for this, you must be very careful because there is a function here that can completely erase the microcontroller memory, which means that the Arduino boot loader is also gone. Stay well clear of anything with the word "erase" in it. The only right option is "Chip/Write Buffer into Chip".

Now you will see a long chain of messages in the programmer window, with the lovely word "Started" at the end. During the programming process you can see from the activity of the Tx and Rx LEDs on the Uno board that a lot of data traffic is going on. At the end the window closes again.

The downloaded program starts running immediately after the end of programming, and the yellow LED blinks. It may be a tiny program, but it's a big step for you as the programmer. If you're not quite sure about all this, try making some small changes to the program. For example, you can change the blinking rate. For really fast blinking, change *Waitms 500* to *Waitms 100*, or for very slow blinking change it to *Waitms 2000*. After editing the code, recompile it and reprogram the microcontroller, and then check the result. If the LED does exactly what you programmed it to do, there's no room for doubt: it really works! In the next instalment we turn our attention to inputs.

Web Links

[1] www.elektor.com/arduino
[2] www.mcselec.com
[3] www.elektor-magazine.com/120574
[4] http://arduino.cc/en/Main/Software

Chapter 40 • Microcontroller BootCamp (2)

Digital inputs

When a microcontroller has to do more than just blindly follow a predefined process, it needs additional information while the program is running. Inputs enable microcontrollers to detect external events and respond accordingly.

Microcontrollers, including the ATmega328 on the Arduino board, have many different types of input. They include analog inputs, which can be used to measure voltages, and digital inputs, which can only detect whether a signal level is high or low. This can be used to read the state of a button or switch, among other things.

Digital inputs

Each port pin of the ATmega328 can be configured as an input or as an output, as mentioned in the previous instalment of this series [1]. Here we want to use port pin PC5 as a digital input (see **Figure 1**). Incidentally, all pins of port C can also be used as analog inputs for measuring voltages. However, to do this you have to use specific instructions to configure the microcontroller accordingly. The microcontroller is configured by writing specific values to separate memory locations inside the device, which are called registers. There are also program instructions to read the contents of the registers that hold the current states of the microcontroller. We have more to say about this later on.

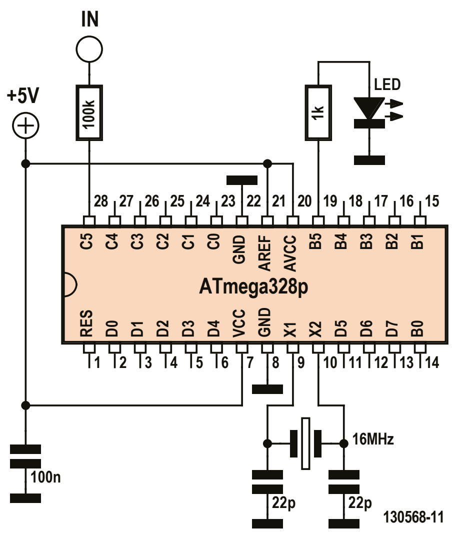


Figure 1.A digital input.

The developers of the Arduino board assumed that port pins PC0 to PC5 would be used as analog inputs, so the corresponding terminals on the Uno board are labelled A0 to A5. Port pin PC5 is connected to A5 at the edge of the board diagonally opposite the USB connector, which makes it easy to find. That's actually why we chose this particular port pin for our first program. **Figure 2** shows all the pins of the ATmega328 with their official pin names (PC6, PD0, etc.), the special functions of individual pins (which we will discuss in due course), and the abbreviated Arduino pin names (at the outside edges).

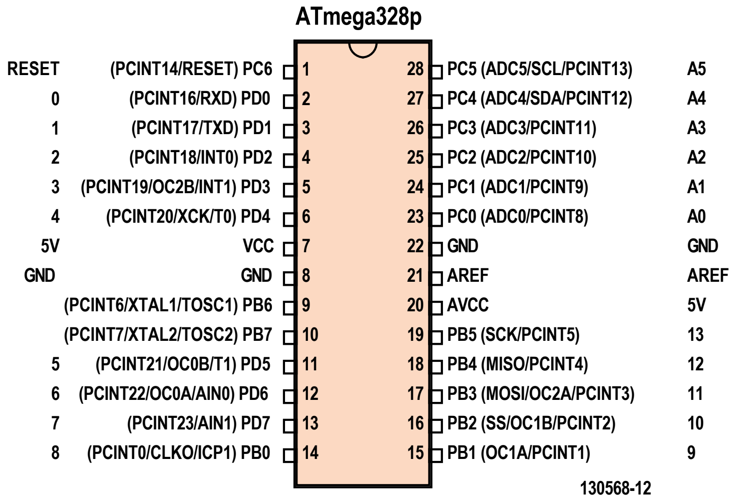


Figure 2. The ATmega328 pinout and the corresponding Arduino names.

After a restart or reset, every pin of port C is configured as a high-impedance CMOS input with the same properties as the familiar 4000 family of CMOS ICs. Accordingly, it's not a bad idea to use 100 kΩ resistors here for protection against excessive voltages and static discharges.

Protection diodes

All modern CMOS devices have a pair of protection diodes on each input, with the one connected to ground and the other to Vcc. Their job is to limit excessive voltages. The first CMOS ICs in the 4000 family, which came on the market a long time ago, did not have input protection diodes. Premature IC failures due to improper handling were therefore fairly common at that time. Static discharges from people to grounded objects at voltage levels from 100 to 1000 V happen all the time, but MOSFET gates don't like them at all and may break down at voltages as low as 50 V. Now that protection diodes are integrated in all ICs, failure due to static discharge breakdown has become rare. You can detect these protection diodes with an ordinary ohmmeter. This is often very useful because it is an easy way to check the connections to the pins of an IC in a circuit to see whether or not they are okay.

The protection diodes are what make it safe to touch an open CMOS input with your finger through a 100 kΩ resistor, despite the fact that your body may have static charges and carry induced AC voltages up to 100 V under no-load conditions. The human body is a sort of node with numerous capacitive links to all sorts of cables and power outlets. As audio designers and troubleshooters know, a finger is an excellent source of a hum signal. The protection diodes (Figure 3) limit the high-amplitude sinusoidal signal to a low-amplitude square wave (5 V peak to peak). You can see this for yourself by measuring the signal with an oscilloscope. Actually the amplitude of the square wave is 6.2 V peak to peak, since the protection diodes only start conducting at –0.6 V and +5.6 V (assuming a supply voltage of 5 V).

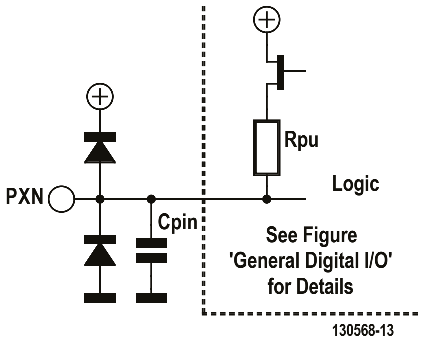


Figure 3. Input protection diodes.

Reading input states

The program UNO_Input1.Bas in Listing 1 (the code can be downloaded from the Elektor website [2]) reads the state of the signal on port pin PC5 and outputs it to the LED on port pin PB5. First all pins of port C (PC0 to PC5) are initialized as digital inputs. In the interest of readability, the instruction Config Portc = Input is used for this purpose. Actually you could omit this instruction, since as previously mentioned port C is automatically configured with all pins as inputs after power-up.

```
'--------------------------------
'UNO_Input1.BAS
'-------------------------------
$regfile = "m328pdef.d    'ATmega328p
$crystal = 16000000       '16 MHz

'-------------------------------


Config Portb = Output
Config Portc = Input


Do
  Portb.5 = Pinc.5        'Copy Input to Output
  Waitms 19               '19 ms
Loop
```

Listing 1. Copying an input to an output.

ATmega microcontrollers have output registers that determine the states of the output port pins. When you program the software to write a particular value to an output register, the corresponding port pins are set to 0 or 1 according to the content of the register. There is a separate input register that holds the actual states of the pins. This register can be read by the software to determine whether the pins are high or low. Each register holds one byte (eight bits) with a value range from 0 to 255 (see the Bits and Bytes inset). Each bit shows the state of a single port pin.

In the case of port B, the output register (write register) is called PORTB and the input register (read register) is called PINB. The software sees these registers as memory locations that can be read and written. However, in hardware terms they are physical circuits with external connections. There is another register for each port called the Data Direction Register, which allows the port pins to be configured as inputs or as outputs. For port B, this register is called DDRB. There is no obvious way to write data to this register in a Bascom program, since this operation is performed indirectly by the Config instruction. For example, Config Portb = Output sets all bits of DDRB to 1, which causes all pins of port B to act as outputs.

If you run the program and connect the input to ground, the LED will be off (dark). If you connect the input to +5 V, the LED will light up. As you can see, the input state is copied to the output. Now try something different: first make sure that you are standing or sitting or standing completely insulated from any conductive objects, and then touch the input with your finger. Remarkably enough, you will see that the LED blinks. At first this may seem very strange, but when you think about it the reason becomes clear: a stroboscope effect. Your finger couples a signal into the input – the well-known 50-Hz AC hum signal with a period of 20 ms, or 16.6 ms for 60 Hz. By contrast, the wait time in the program is 19 ms. The 5% difference between the two times yields an output signal with a frequency of 2.5 Hz (50 Hz x 0.05). If you live in an area where the AC line frequency is 60 Hz, the output frequency—and thus the blinking rate—is correspondingly higher.

What makes a high level high?

Like all digital circuits, microcontroller output pins have only two states, which are variously called on and off, high and low, 1 and 0, or whatever. Furthermore, microcontrollers can only detect these two signal states on their input pins. However, in practice any voltage from 0 V (ground) to the supply voltage (Vcc) can be applied to an input pin. You might think that anything above 2.5 V is a high level and anything below 2.5 V is a low level. However, the ATMega328 data sheet states somewhat circumspectly that the minimum value where the pin is guaranteed to be read as high is 0.7 Vcc (corresponding to 3.5 V) and the maximum value where the pin is guaranteed to read as low is 0.3 Vcc (corresponding to 1.5 V). That sounds a bit like a disclaimer intended to prevent unjustified complaints about supposedly incorrect operation: if you connect a signal between 1.5 and 3.5 V to an input, whatever happens is your responsibility. But what actually happens in that case? The only way to find out is to simply do it and observe the results.

Go ahead, try it! Use an external power supply for this so that Vcc is exactly 5 V, since the voltage available from a USB port is usually only 4.5 V or so. Connect a potentiometer as shown in Figure 4 and vary the input voltage over the range of 0 to 5 V. Here’s what you should measure with the microcontroller: If you gradually raise the voltage from 0 V, the output state changes from low to high at 2.7 V and the yellow LED lights up. If you then slowly reduce the voltage, the output state changes back to low at 2.2 V. You can repeat this process as often as you like. The switching points are 0.5 V apart, and no change occurs in the range between 2.2 and 2.7 V. This means that there is a hysteresis of 0.5 V. The data sheet doesn't actually say this, perhaps because nobody at Atmel wants to claim that you can rely on this property. However, it's a useful property because it means that the inputs act like Schmitt triggers. As you may know, the purpose of Schmitt triggers is to convert analog signals with variable voltage levels into digital signals.

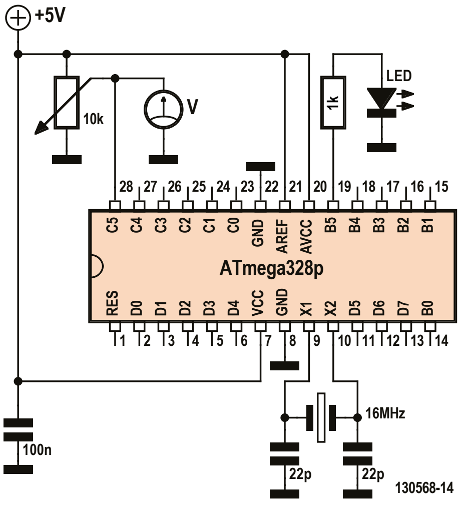


Figure 4. Measuring the input voltage.

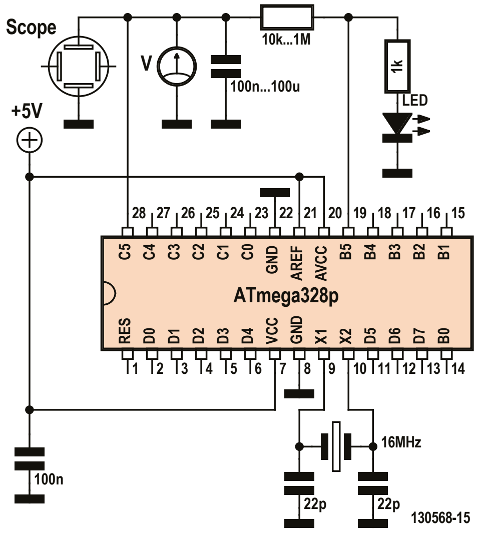


Figure 5. A control loop.

Switching back and forth

You can also automate the above process and dispense with the potentiometer. You already have the necessary hardware in the form of the microcontroller, and the idea is to have it measure its own switching points. In the circuit in **Figure 5**, the capacitor connected to the input pin is charged by the output pin through a series resistor. The output should go to 0 when a 1 is read at the input, and it should go to 1 when a 0 is read at the input. This circuit is effectively a sort of on/off controller with hysteresis. You need to change the program slightly for this purpose, as shown in **Listing 2**. The Not function inverts a digital state, in the same way as an inverter gate in hardware logic. The wait time is not necessary because the program should respond immediately when a new state is detected. In this case, "immediately" means within a microsecond or so.

```
'--------------------------------
'UNO_Input2.BAS
'--------------------------------
$regfile = "m328pdef.dat"  'ATmega328p
$crystal = 16000000        '16 MHz

'-------------------------------


Config Portb = Output
Config Portc = Input

Do
  Portb.5 = Not Pinc.5     'Inverted Input to Output
Loop
```

Listing 2. Inverting the input signal.

The potentiometer of the previous circuit is replaced here by an RC network. The component values of this network (and the corresponding time constant) can be chosen freely over a wide range. For instance, you could use 100 nF and 10 kΩ if you have an oscilloscope available. In that case you can view the triangular waveform and see the hysteresis directly (**Figure 6**). If you use a digital multimeter with high input impedance to make your measurements, it's better to use 100 µF and 1 MΩ. In that case the voltage changes so slowly that you can easily read the minimum and maximum levels from the meter.

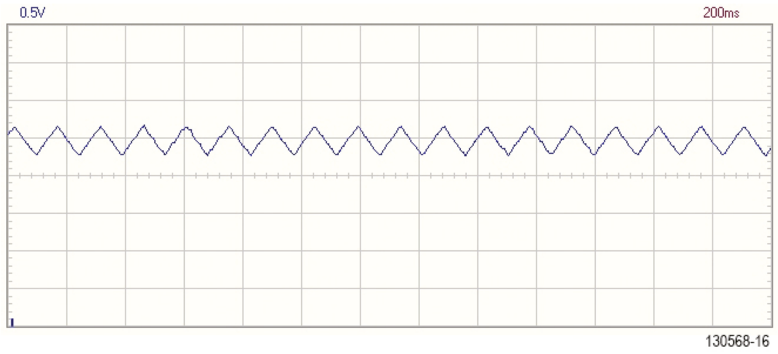


Figure 6. Voltage waveform at the input.

Actually you don't need a measuring device for this process, since you can use the analog inputs of the Arduino board to measure the voltage. Any voltage in the range of 0 to 5 V can be measured digitally with a resolution of approximately 5 mV. We'll focus on how to do this in the next part of this series.

Incidentally, if you want to examine the response time of the program in more detail, you can simply omit the RC network and connect pin PB5 directly to pin PC5. If you then measure the signal with an oscilloscope, you will see a rectangular waveform with a frequency of approximately 500 kHz. This means that the circuit responds to changes in the input state roughly once per microsecond.

There's only one strange thing about this circuit: with a symmetrical 500-kHz square-wave signal, the yellow LED on the Arduino board should be lit at half its normal brightness, but it remains dark. This reason for this can only be determined by examining the circuit diagram of the Arduino Uno in more detail. The built-in LED on the Uno board is connected to PB5 through a buffer consisting of one half of an LM358 dual opamp. This is unusual on a digital circuit board, and it comes from the fact that the other half of the LM358 is used as a comparator for the supply voltage. If you connect an external power supply, the Uno board automatically disconnects the USB supply voltage—very user-friendly and reliable. The other opamp is redundant and actually shouldn't be used, but the designer took advantage of its high input impedance to use it as buffer that allows port pin PB5 (Arduino pin 13) to also be used as an input. However, a 500-kHz signal is simply outside the frequency range of an ordinary opamp, so the LED on the board remains dark. If you want to actually see something, you have to connect your own LED to the board as shown in **Figure 1**.

Branching

Microcontrollers can make decisions by following program branches. In practical terms, this means that parts of a program are either executed or not executed, depending on specific conditions. The program structure used for this in Bascom is the If block, which takes the *form If* <condition> *Then* <instructions> *End If.* The instructions inside the If block are executed if the condition is fulfilled, and otherwise these instructions are not executed and the program jumps directly to *End If*.

In the program in **Listing 3** you can also see another new instruction called Toggle, which means "switch to the opposite state" and causes the output to change state each time the instruction is executed. The blinker runs as long as the signal level on port pin PC5 is logic 1, which means 5 V or at least something higher than 2.7 V. When the input pin is connected to ground, which corresponds to logic 0, the LED remains in its current state – either constantly lit or constantly dark.

```
'--------------------------------
'UNO_Input3.BAS
'--------------------------------
$regfile = "m328pdef.dat"  'ATmega328p
$crystal = 16000000        '16 MHz
'-------------------------------


Config Portb = Output
Config Portc = Input


Do
  If Pinc.5 = 1 Then
    Toggle Portb.5
    Waitms 250
  End If
Loop
```

Listing 3. An If block.

The circuit is the same as in Figure 1. Here as well, you can test the circuit by connecting the input to ground or +5 V, or by touching it with your finger. An interesting aspect in this regard is that your finger acts like a high (logic 1) signal. This means that if you touch the input pin when nothing else is connected to it, the LED will blink at a steady rate. This results from the Wait instruction in the If block. When the program sees a 0 level on PC5, it immediately jumps to End If and then back to the start of the loop. This means that a 0 level causes the If condition to be tested repeatedly in rapid succession. If you apply a 50-Hz signal to the input by touching it with your finger, after at most 10 ms a 1 level is detected at the input. Then the LED changes state and the program waits 250 ms before testing the If condition again.

This aspect of the program also shows you what you shouldn't do. Open inputs are bad news and cause nasty problems, especially if you overlook them. In the case of this simple program, it's totally unpredictable what will actually happen when nothing is connected to the input. The input may be at the high level, in which case the will LED blink, or it may be at a low level, in which case the LED will not blink – but you can't say in advance whether or not the LED will blink. If you hold your hand close to the input and move your feet, the static charge on your body can cause the port to change states as a result of capacitive coupling through the air. An outside observer who doesn't know all the details might start looking for a pesky intermittent contact, and perhaps start to doubt the reliability of microcontrollers in general. Many experienced designers have learned about this the hard way by spending hours looking for a suspected software bug or hardware error, when the actual problem was an open input.

Reading switch states with a pull-up resistor

From the above, we can conclude that open inputs cause problems. Suppose you want to read the state of a switch. This means that the input must have a defined state when the switch is open. The usual way to achieve this is to use a pull-up resistor, which is a resistor connected to Vcc to pull the input high when it is open. If the microcontroller sees a low level at the input, it knows that the switch connected to the input is closed.

The pull-up resistor in **Figure 7** is shown connected by dashed lines, which means that the resistor is optional. This is because the microcontroller has built-in pull-up resistors; all you have to do is connect them. Many microcontrollers have a separate register for this purpose. However, AVR microcontrollers manage to handle everything with a total of three registers: PORTC, PINC and DDRC. In this case you set all bits of DDRC to 0, which configures all pins of port C as inputs. You also set all bits of PORTC to 1, as though you wanted to set all the pins to high outputs. What actually happens in this case is that each port pin is connected to Vcc by an internal pull-up resistor (marked “Rpu” in Figure 3) with a resistance of approximately 30 kΩ. This causes the input pins to have a high signal level and a relatively low input impedance, instead of a high impedance. As a result, all bits in PINB will be read as 1 unless they the corresponding pin is connected to ground by an external switch.

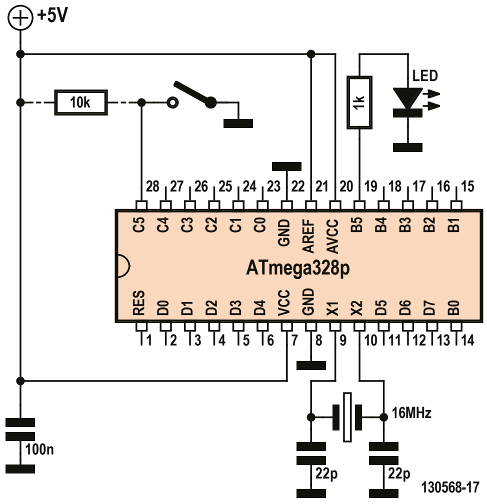


Figure 7. A pull-up resistor.

In **Listing 4** you only need the instruction *Portc.5 = 1*, which connects the internal pull-up resistor to pin PC5. The other inputs of port C are left in the high-impedance state without a pull-up resistor. That doesn’t matter here because they are not used in the program. With this change, the program response is unambiguous. An open input is always read as a high level (logic 1), and the LED blinks. Another change is necessary to ensure that there is no ambiguity when the switch is closed. This consists of adding an *Else* section to the If block. The instructions between *Then* and *Else* define what has to be done when the switch is open, and the instructions between *Else* and *End If* define what has to be done when the switch is closed. In this case, the LED should be off then.

By the way, you might want to measure the actual value of the internal pull-up resistor. The data sheet says that it is somewhere between 20 kΩ and 50 kΩ. To check this, connect an ammeter to the input pin instead of the switch, or in parallel with the switch. In our case, we measured 140 µA. This corresponds to a resistance of 35.7 kΩ (5 V ÷ 0.14 mA), which is exactly in the middle of the range specified by Atmel.

```
'--------------------------------
'UNO_Input4.BAS
'--------------------------------
$regfile = "m328pdef.dat"  'ATmega328p
$crystal = 16000000        '16 MHz
'-------------------------------

Config Portb = Output
Config Portc = Input
Portc.5 = 1                'Pullup

Do
  If Pinc.5 = 1 Then
    Toggle Portb.5
    Waitms 250
  Else
    Portb.5 = 0
  End If
Loop
```

Listing 4: Port query with pull-up.

By the way, you might want to measure the actual value of the internal pull-up resistor. The data sheet says that it is somewhere between 20 kΩ and 50 kΩ. To check this, connect an ammeter to the input pin instead of the switch, or in parallel with the switch. In our case, we measured 140 µA. This corresponds to a resistance of 35.7 kΩ (5 V ÷ 0.14 mA), which is exactly in the middle of the range specified by Atmel.

Bits and Bytes

Each register in the AVR microcontroller holds eight bits, or one byte. A bit can only have the value 1 or 0. A byte (in a register or otherwise) consists of eight bits, so a byte register can hold values from 0 to 255 in decimal notation. If you write the values in binary notation instead, you can see the individual bit values directly. For example: 00000000, 11111111 or 100110010.

In many registers each bit is directly associated with a specific pin. One example is the PORTD register. Bit 7 (at the left end) determines whether port pin PD7 is high or low. Similarly, bit 0 (at the right end) determines the state of port pin PD0.

Many programming languages have instructions for setting or clearing individual bits in a register, which are accordingly called single-bit instructions. There are also instructions that can write a specific value to a register and therefore affect all of the bits at the same time.

To take an example from Bascom, Portb.5 =1 sets bit 5 of the eight-bit register PORTB to 1, which means that the signal level on PB5 is high. None of the other bits is affected, since this is a single-bit instruction. However, you can also access all bits of the PORTB register at the same time with the Bascom instruction Portb = &B00100000, where &B is the Bascom notation for a binary number. This has the same effect on bit 5 as the previous instruction: in both cases the yellow LED on the Arduino board lights up. However, the difference can certainly be significant for the other seven port pins. In the first case, any of these bits that were previously set to 1 remain in that state, but in the second case they are all set to 0.

To take another example, in Listing 1 you could replace the line Portb.5 = Pinc.5 with Portb = Pinc. If you only consider bit 5, you have the same circuit and the same result, but there's a big difference because now you're moving eight bits at a time instead of just one. As a result, any change on input PC0 will affect output PB0.
Along with binary notation (such as &B00100000), Bascom supports two other commonly used notations. Instead of &B00100000, you can write the value in hexadecimal notation, in which case the instruction takes the form Portb =&H20. The other alternative is decimal notation: Portb = 32.

The best way to understand how these different ways to represent numbers work is to consider the example of a four-bit number. The smallest possible value is 0000 (all bits off), and the largest possible value is 1111 (all bits on). In mathematical terms, this is called a base-2 number system. Each bit has a specific weight, which increases from right to left: 1, 2, 4, 8. Bit 0 has the weight 2^0 = 1, bit 1 the weight 2^1 = 2, bit 2 the weight 2^1 = 4, and bit 3 the weight 2^3 = 8. As you can see, the value doubles with each bit position. You can obtain the corresponding decimal number by adding up the values of the individual bits. In hexadecimal notation, the numbers 10 to 15 are represented by the letters A to F.

„8“	„4“	„2“	„1“	Decimal	Hexadecimal
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	2	2
0	0	1	1	3	3
0	1	0	0	4	4
0	1	0	1	5	5
0	1	1	0	6	6
0	1	1	1	7	7

1	0	0	0	8	8
1	0	0	1	9	9
1	0	1	0	10	A
1	0	1	1	11	B
1	1	0	0	12	C
1	1	0	1	13	D
1	1	1	0	14	E
1	1	1	1	15	F

For comparison (and for those of you who have forgotten your first-grade math): our usual number system with base 10 comes from the more or less random biological circumstance that we have a total of ten fingers and thumbs. The weights of the individual digits are 1. 10, 100, 1000 and so on.

The eight bits of a byte can be divided into two sets of four bits. Each set of four bits can then be expressed directly as a hexadecimal number. For example, the binary number &B00100000 (0010 0000) is the same as &H20 in hexadecimal notation.

To convert a hexadecimal number into a digital number, you only need to know that the hexadecimal system is based on 16 as the name indicates (of course, in that regard it helps if you know a bit of classical Greek and Latin). The weights of the digits from right to left are therefore 1, 16, 256, 4096 and so on. For example, the number &H20 corresponds to 2 x 16 = 32.

If you often need to convert binary numbers into decimal numbers, you should memorize the weights of the eight binary digits of a byte. Armed with that knowledge, all you have to do is to add up the individual bit values.

Position	7	6	5	4	3	2	1	0
Value	128	64	32	16	8	4	2	1

Now for a question: what is the largest number that can be represented by a byte? If you said ‘255’, you’re right. Once you know this, the world of numbers is an open book for you and it is no longer a mystery why you can replace the instruction Portb = Output in Bascom with the simple instruction Ddrb = 255. This says that all bits in the data direction register for port B (DDRB) are set to 1 and all pins from PB0 to PB7 are outputs.

Latch-up

You can use the input protection diodes of the port pins intentionally in your circuits to limit input voltages higher than Vcc. To take an RS232 interface as an example: the signal voltage on the output line of a PC serial port is often −12 V and +12 V, or in some cases somewhat less. This is a complete mismatch to CMOS inputs with 5 V signal levels. The quick and dirty solution is to connect the signal line directly to the input through a protective series resistor (value 10 kΩ or 100 kΩ) as illustrated in Figure 1. Here the protection diodes take care of the rest.

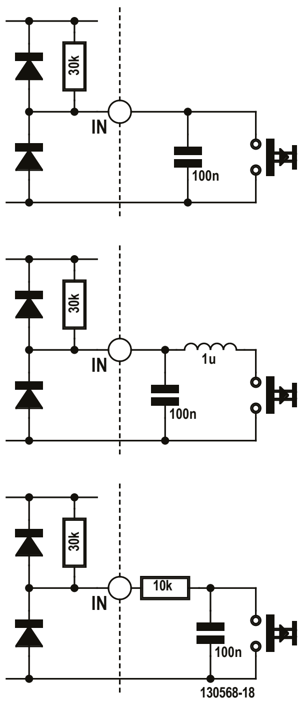


Figure 8. Capacitor, parasitic inductance and damping resistance.

However, there's one thing you have to watch out for: the current through the protection diodes should never exceed a few milliamperes. This is because every CMOS IC has a parasitic thyristor that can be triggered unintentionally. It basically consists of an NPN transistor and a PNP transistor, which are a sort of undesirable side effect of the CMOS structure. This thyristor can be triggered by the current through the protection diodes if it rises above a certain value, which might be 100 mA or as high as 1 A —but you never know the precise value. The worst thing about this is that even very short current spikes can trigger the thyristor. After the device enters the latch-up state, what happens next depends on how much current the power supply can deliver. The latched CMOS IC draws as much current as it can and gets blistering hot. So if you get your fingers get burned, switch off the power right away and wait a few minutes. There's still a small chance that the IC will have survived, but in most cases it's game over. If you measure the resistance between the Vcc and ground pins, you'll only read a few ohms.

If your body has a static charge and you touch an IC input, the current pulse may be large enough to trigger latch-up. Many modern ICs can withstand static discharges up to 15 kV, based on a human body model with a contact resistance of 1 kΩ. In that case the ignition threshold is about 15 A, and you can certainly feel static discharges of that magnitude. However, no IC can survive contact with a 12 V power lead lying loose on the bench, especially if the power supply has a hefty output capacitor. If the power lead touches an IC pin, the microcontroller is guaranteed to be dead.

Have a look at the circuit diagram in the above figure. Do you see anything to be worried about? To all appearances, the only thing the capacitor on the input does is to debounce the switch contacts. This is common practice to ensure that the microcontroller only sees a single level change when the switch contacts bounce back and forth a few times. What the circuit diagram doesn't show is the length of the wires and their inductance. If you imagine an inductor in the circuit as shown in the middle diagram, you have something that looks a lot like Marconi's spark-gap transmitter. When the switch is closed, there is an excited resonant circuit that oscillates at 500 kHz, assuming a capacitance of 100 nF and an inductance of 1 µH There's also enough energy available, since the capacitor has previously been charged through the pull-up resistor. The first peak of the oscillation waveform hits the protection diode with full force, and under unfavorable conditions it can trigger that nasty thyristor. The circuit diagram at the bottom shows how you can prevent this. All you need is a current limiting resistor that soaks up the excess energy.

Web Links

[1] www.elektor-magazine.com/120574
[2] www.elektor-magazine.com/130568

Chapter 41 • Microcontroller BootCamp (3)

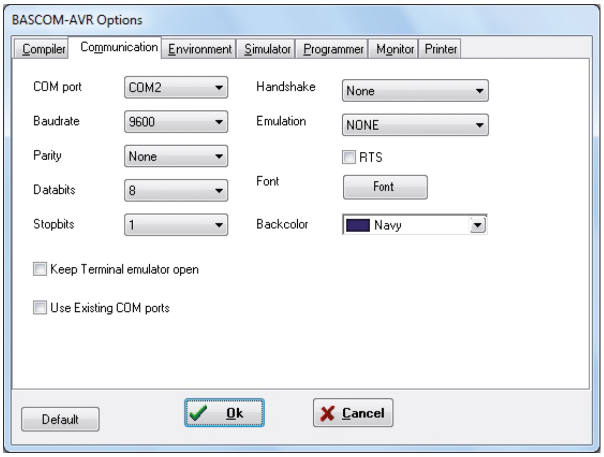
Serial interface and A/D converter

This instalment focuses on the analog to digital converter, which is effectively a voltmeter built into the microcontroller. However, it does not have a pointer or a display, so you have to do a bit of work to be able to read the measurements. The A/D converter simply supplies data. In order to be displayed somewhere, this data has to be transmitted. That's where the serial interface comes into play.

Although most PCs nowadays have dispensed with the traditional serial interface, it is still widely used in the microcontroller world. This interface was originally developed to transfer serial data over a conductor. The individual bits obediently travel down the wire one after the other. The bit timing is precisely defined so that the receiver can make sense of the serial bit stream. A commonly used data rate is 9600 bits per second, which is also called 9600 baud. In addition to the eight data bits of each byte, there is a start bit and one or two stop bits. Each data bit is present on the wire for an interval of approximately 100 µs (at 9600 baud).

Print output

Every ATmega microcontroller has a serial interface (UART) with RXD (PD0) and TXD (PD1) lines. These signals are TTL compatible, which means that the signal level is 5 V in the quiescent state and 0 V in the active state. By contrast, a COM port on a PC operates with RS232 signal levels. They are -12 V in the quiescent state and +12 V in the active state. For this reason, a peripheral interface IC such as the MAX232 is often necessary to invert and adapt the signal levels. Instead of this, the Arduino board has a serial to USB converter that sends the data over USB. This means that you need a driver for a virtual serial port (such as COM2) on the PC. For programs running on the PC, that makes the data transported over USB look like it entered the PC through a conventional serial port. This data can therefore be displayed using a standard terminal emulator program.



COM port: COM2
Baudrate: 9600
Parity: None
Databits: 8
Stopbits: 1
Handshake: None
Emulation: NONE
RTS: ☐
Backcolor: Navy
☐ Keep Terminal emulator open
☐ Use Existing COM ports

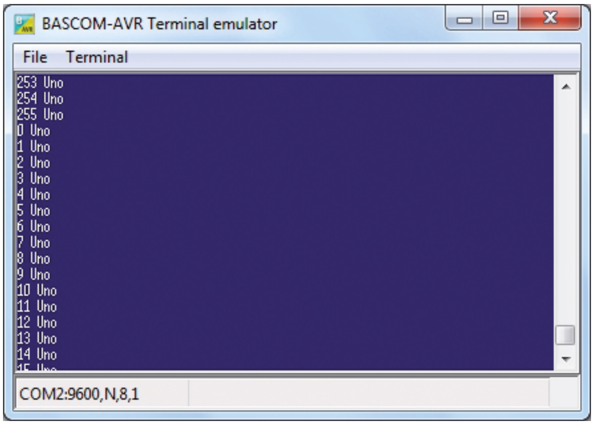
Figure 1. Terminal emulator settings.

We recommend that you use the terminal emulator program integrated into Bascom. You can open it under Tools > Terminal Emulator or with the key combination Ctrl-T, but you have to configure the right settings to make it work properly. In particular this means selecting the right COM port (the same one you configured for the boot loader) and setting the right baud rate (in this case 9600). For the rest you can use the default settings: 8 data bits, 1 stop bit and no parity (**Figure 1**).

```
'------------------------------------
'UNO_Print.BAS
'------------------------------------
$regfile = "m328pdef.dat"
$crystal =
$baud = 9600

Dim N As Byte

Do
  Print N;
  Print " Uno"
  Waitms 200
  N = N + 1
Loop
```

Listing 1: Print output

Now you have to persuade the Arduino Uno to transmit something. That's easy in Bascom, thanks to the Print command. See **Listing 1** for an example of how to use the Print command (note that all code files can be downloaded at [1]). This command can be used to send output to a printer, which explains its name. However, to avoid wasting paper it's better to send text and data to the monitor. The result is shown in **Figure 2**.



```
253 Uno
254 Uno
255 Uno
0 Uno
1 Uno
2 Uno
3 Uno
4 Uno
5 Uno
6 Uno
7 Uno
8 Uno
9 Uno
10 Uno
11 Uno
12 Uno
13 Uno
14 Uno
```

COM2:9600,N,8,1

Figure 2. Print output transmitted over a serial line.

There are two Print commands in the program. The second one sends a message in quotation marks: "Uno". But first a number is sent – more particularly, the number N. If it's been a while since you had anything to do with math, you may still have a vague memory that unknown quantities in algebra are often designated as x. You could do the same in Bascom, but in situations where numbers are simply incremented (1, 2, 3 and so on), the symbols N, M, I and J are commonly used in source code. However, you are free to choose your own symbols for variables of this sort. The only thing that really matters is that you first tell Bascom what type of number each variable represents. This is because Bascom has to know how many bytes of memory the number needs and how computations should be performed with the number in the code. For this reason, the variables must be declared (dimensioned) before they are used. That's the purpose of the following instruction: Dim N as Byte. This means that N is a number of type Byte, which clearly indicates that this number consists of eight bits (recall the previous instalment) and has a number range from 0 to 255.

Assignments
Now you can perform calculations with the number variable N. The variable is assigned the value 0 at the start of program execution. Although you might think this is only logical, it is actually a particular feature of Basic. In other languages you have to write 'N = 0', but in Bascom this is done automatically. The value of N is subsequently incremented by 1 each time the loop is executed. My old math teacher would have boxed my ears if I wrote 'N = N + 1', because it is not valid mathematical equation. However, in many programming languages the equal sign is used as an assignment operator. The intention here is that the variable N is assigned a new value by adding 1 to the existing value.

You can see the results of this ongoing incrementing in the terminal window: the value of N increases by 1 after each output. You can also see something else: the value stops rising after it reaches 255, since we defined N as a single-byte number. This means that 255 + 1 is not 256, but instead 0. This is called number overflow. Here this is not a problem, but in other places it can be a problem – namely when you have inadvertently chosen a number type that is too small and the overflow is not detected.

There's another detail that should be noted here: why does the display always go to a new line after "Uno", but not after the number? The answer is simple: by default, Bascom sends the control character CR (which stands for 'carriage return', like an old-fashioned typewriter) and LF (line feed) at the end of each Print command. If this is not what you want, you have to put a semicolon at the end of the instruction (Print N;). That's why the text is on the same line as the number. A space is also inserted to separate the number from the text. As you can see, it's all very simple and straightforward; you could use the same approach to display '200 mV' on the monitor. After all, our original intention here was to display measurements from the A/D converter, and now we're ready to do so.

The serial bits output by the Atmega328, which are sent to the serial to USB converter on the Uno board, can easily be seen with an oscilloscope. Simply touch the scope probe to the TX pin of the microcontroller, which is routed to the socket header at the top right on the Arduino board. Then you can see the data stream. The RX pin is next to the TX pin, and if you type some characters on the terminal you can see them on the RX line. This shows that the PC can also send data to the microcontroller. At this point the received data does not do anything because the program ignores it.

The A/D converter

The analog to digital converter (ADC) in the microcontroller is a sort of measuring device. As we all know, measuring involves comparing. First you need some sort of unit to serve as a reference quantity. Before you can measure a distance in meters, you have to know how long a meter is. Likewise, you have to know how big a volt is before you can say how many volts are lurking in your wall outlet. The situation here is similar. The microcontroller needs a reference voltage so that it can compare the voltage on a selected analog input (A0 to A5 on the Uno board) to the reference voltage. The reference voltage is connected to the AREF pin. However, you can also use a register to select the reference voltage. For example, you could apply an external reference voltage of 1 V or connect a 5 V reference voltage from inside the microcontroller, but of course you shouldn't do both at the same time.

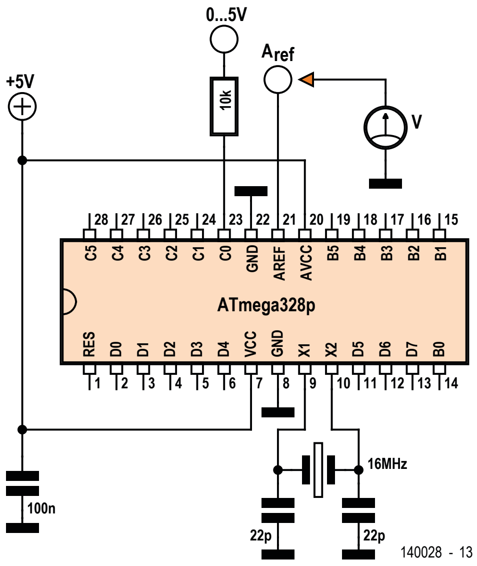


Figure 3. Using the A/D converter.

Now try measuring the voltage between AREF and GND (see **Figure 3**). Since the A/D converter hasn't been configured yet, the voltage is zero. If you load the program in **Listing 2** into the microcontroller and make this measurement again, you will see a voltage of 5 V. This is due to the following configuration statement:

```
Config Adc = Single , Prescaler = 64 , Reference = Avcc
```

```
'----------------------------------------
'UNO_AD1.BAS  ADC
'----------------------------------------
$regfile = "m328pdef.dat"
$crystal = 16000000
$baud = 9600

Dim D As Word

Config Adc = Single , Prescaler = 64 , Reference = Avcc  '5 V

Do
  D = Getadc(0)
  Print D
  Waitms 200
Loop
```

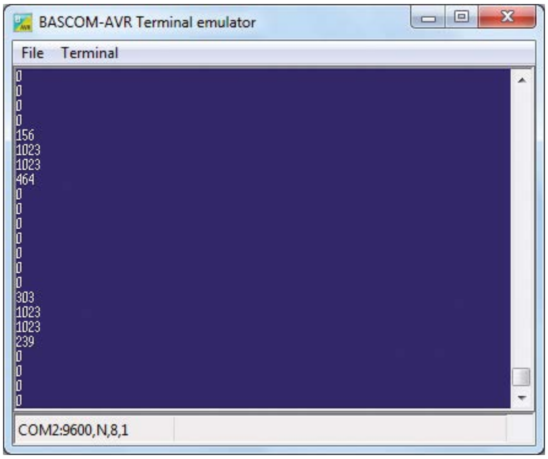
Listing 2: Using the A/D converter

This initializes the ADC, which means it enables the ADC with specific attributes. Here Single mode means that only one measurement is made for each request. The other option is Free (short for free-running mode), in which measurements are made repeatedly without interruption. The prescaler generates a clock signal for the A/D converter by dividing the processor clock signal, which is 16 MHz on the Uno board. In this case the A/D converter runs at 250 kHz (16 MHz divided by 64). It can also run faster, but then it is not quite as accurate. The setting ‘Reference = Avcc’ connects the Aref pin to Avcc, which the supply voltage of approximately 5 V. Incidentally, it’s worthwhile taking a look at the data sheet for the microcontroller – the section on the A/D converter is practically a data sheet in its own right. If after that you still don’t have any idea of what settings you can or should use, the Bascom Help is a good source of information. Simply point the mouse to ‘Config’ and press the F1 key, and you will be rewarded with all sorts of useful information and a bit of sample source code.

The ‘instruction D = Getadc(0)’ initiates a measurement of the voltage on A0 and copies the result to the variable D. The A/D converter provides readings with a resolution of 10 bits, which corresponds to a number range of 0 to 1023. The variable D must therefore never be dimensioned as a byte variable. You should use either the type Word (range 0 to 65535) or the type Integer (range -32768 to +32767) to ensure that the readings will fit.

The measurements are made using a successive approximation algorithm with ten steps. First the converter compares the input voltage to half the reference voltage. If it is higher than the reference voltage, the next comparison is to three-quarters of the reference voltage. If it is still higher, the next comparison is seven-eighths, and so on. The resolution doubles with each step. At the end you have number that indicates which step on a ladder with 1024 steps between zero and the reference voltage corresponds to the measured voltage. Each step is approximately 5 mV (5 V divided by 1023) if you use a reference voltage of 5 V. For comparison, the resolution of a digital voltmeter with a 3-½ digit display is roughly twice as good (2000 steps). This makes the microcontroller a fairly good measuring device, although you shouldn't confuse resolution with accuracy – the accuracy can be seriously compromised by an inaccurate reference voltage.

The analog inputs are subject to the same considerations as the digital inputs: an input voltage well below zero or well above the supply voltage can lead to undesirable side effects, extending as far as latchup (as described in the previous part of this series). However, here again a 10-kΩ resistor in series with the analog input can prevent serious problems. With that precaution in place, it's even okay to touch the input with your finger. If you do so, you will see varying readings in the terminal window (**Figure 4**), including many limit values (0 and 1023) because the hum voltage exceeds the measuring range. The overall effect is similar to the waveform from a half-wave rectifier as seen on a oscilloscope. If you compare the measurement cycle time, consisting of the programmed wait interval of 200 ms plus the actual measurement time, with the AC line voltage period of 20 ms (at 50 Hz), you can see that here again there is a stroboscope effect. This is called undersampling, and it produces a low-frequency alias signal because the measurement samples do not all lie within the same period of the measured signal. In fact, there are several periods of the input signal between each pair of samples.



```
0
0
0
0
156
1023
1023
464
0
0
0
0
0
0
0
303
1023
1023
239
0
0
0
0
```

Figure 4. Measurement data from a 50 Hz signal.

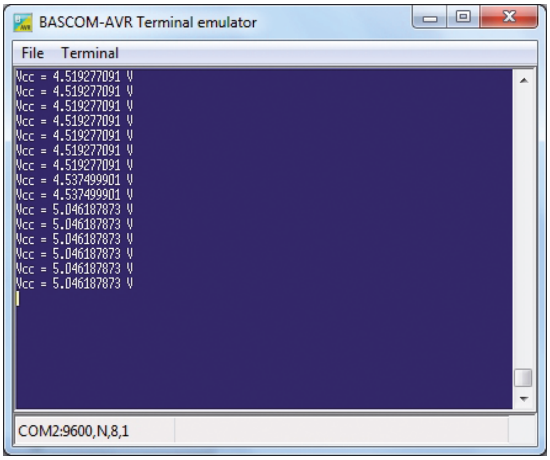
A bit of math

Our first simple measurement program only generates raw data, so you have to calculate the signal voltage yourself. Consider the following example: You connect analog input A0 to the 3.3 V output of the Arduino board. Now you see a reading of 676, or something close to that. You grab your pocket calculator and calculate: 676 ÷ 1023 × 5 V = 3.305 V (okay). Another option is to modify the program and have the microcontroller do the math (**Listing 3**). For this you need a variable that can hold more than just integer values. You can use a variable U of type Single (which means a single-precision floating point / real number variable), which is fully sufficient. Now you can convert the raw data in two steps. You can only perform one calculation in each instruction (that's a Bascom rule). The long chained equations you see in C or Pascal are not possible here.

```
U = D * 5.0
U = U / 1023

'----------------------------------------
'UNO_AD2.BAS    0...5 V
'----------------------------------------
$regfile = "m328pdef.dat"
$crystal = 16000000
$baud = 9600

Dim D As Word
Dim U As Single

Config Adc = Single , Prescaler = 64 , Reference = Avcc  '5 V

Do
  D = Getadc(0)
  U = D * 5.0
  U = U / 1023
  Print U ; " V"
  Waitms 200
Loop
```

Listing 3: Conversion to volts

Now you can see the result in the terminal window. Of course, you should treat the large number of decimal places with caution, since they give a false impression of the actual resolution. Later we will show you how to properly round off readings of this sort, but for the time being it's nothing to worry about.



```
Vcc = 4.519277091 V
Vcc = 4.519277091 V
Vcc = 4.519277091 V
Vcc = 4.519277091 V
Vcc = 4.519277091 V
Vcc = 4.519277091 V
Vcc = 4.519277091 V
Vcc = 4.537499901 V
Vcc = 4.537499901 V
Vcc = 5.046187873 V
Vcc = 5.046187873 V
Vcc = 5.046187873 V
Vcc = 5.046187873 V
Vcc = 5.046187873 V
Vcc = 5.046187873 V
```

COM2:9600,N,8,1

Figure 5. Measuring the microcontroller's own supply voltage.

At this point you can see that it makes a difference whether you power the Uno board over USB or from an external power supply. The supply voltage on the USB port can easily differ from 5 V by 10% or more (see **Figure 5**), and the accuracy of the measurement results will vary accordingly. By contrast, the 5 V regulator on the Uno board has a relatively narrow tolerance, typically around 1%. Try both options for yourself. Also measure the actual voltage on the 3.3 V terminal and the voltage on the AREF pin in each case. Armed with that information, you can significantly improve the accuracy. Suppose you measure a reference voltage of 5.03 V with an external power supply (using a high-precision DVM). Enter this value in the program in place of the nominal value of 5.0 V, and your readings will be nearly spot on.

You should also try other reference settings. If you configure 'Reference = Aref', at first you do not have any reference voltage at all. You have to provide your own reference voltage, with a free choice anywhere in the range of 0 to 5 V. For example, you can connect the Aref pin to the 3.3 V terminal of the Uno board. In this case the measuring range extends to 3.3 V. Another option is 'Reference = Internal'. In that case the A/D converter uses an internal 1.1 V reference voltage, and you can measure this voltage on the Aref pin. The data sheet very modestly states a tolerance of approximately 10%, which means 1.0 V to 1.2 V. In our case we measured 1.08 V, which is only 2% away from the nominal value. However, you should measure the actual value yourself. There are two things worth noting about this relatively low reference voltage. The first is that the accuracy is independent of the supply voltage; and the other is that it gives you a relatively high resolution of approximately 1 mV.

```
‘----------------------------------------
‘UNO_AD3.BAS  Vcc / 1.1 V
‘----------------------------------------
$regfile = “m328pdef.dat”
$crystal = 16000000
$baud = 9600

Dim D As Word
Dim U As Single

Config Adc = Single , Prescaler = 64 , Reference = Avcc

Do
  D = Getadc(14)                     ‘Ref 1.1 V
  U = 1023 / D
  U = U * 1.1
  Print “Vcc = “;
  Print U ; “ V”
  Waitms 1000
Loop
```

Listing 4: Measuring V_{CC}.

Measuring the temperature

In addition to measuring its own supply voltage, the ATmega328 can measure its own temperature thanks to an internal temperature sensor connected to channel 8 of the A/D converter. The output voltage of this sensor is proportional to the temperature, with a factor of approximately 1 mV per degree. The accuracy is not especially high – the specified tolerance is ten degrees. However, you can always calibrate it yourself.

According to the data sheet, the sensor output is 242 mV at −45 °C, 314 mV at 25 °C and 380 mV at 85 °C. To obtain meaningful readings with voltages in this range, you have to use the relatively low internal reference voltage (1.1 V). However, the same program is supposed to measure higher voltages as well, so it automatically switches to the higher reference voltage (Avcc) as necessary.

For internal temperature measurement the A/D converter operates with the internal 1.1 V reference voltage and a resolution of about 1 mV, which roughly corresponds to 1 degree at the temperature sensor output. At 25 degrees C the sensor output is approximately 314 mV. We decided to take a quick-and-dirty empirical approach, and we determined that the temperature in degrees C could be found by subtracting 338 from the measured value (see **Listing 5**). However, even better results could be obtained with a bit more computation effort. In any case, you should adjust the results according to your actual situation to obtain the best possible match with the temperature shown by an ordinary thermometer. Next we held a finger on the microcontroller to see whether the temperature rose.

```
'-------------------------------------------
'UNO_AD4.BAS   Temp
'-------------------------------------------
$regfile = "m328pdef.dat"
$crystal = 16000000
$baud = 9600

Dim D As Word
Dim N As Word
Dim U As Single

Config Portb = Output
Portb.5 = 1
Portb.4 = 0

Do
  Config Adc = Single , Prescaler = Auto , Reference = Internal  '1,1 V
  Waitms 200
  D = Getadc(8)                                            'Temperature
  D = D - 338
  Print D ; " deg"
  Waitms 500
  Config Adc = Single , Prescaler = Auto , Reference = Avcc '5 V
```

```
    Waitms 200
    D = Getadc(0)
    U = D * 5.0
    U = U / 1023
    Print U ; " V"
    D = Getadc(1)
    U = D * 5.0
    U = U / 1023
    Print U ; " V"
  Loop
```

Listing 5: Temperature and voltage drops

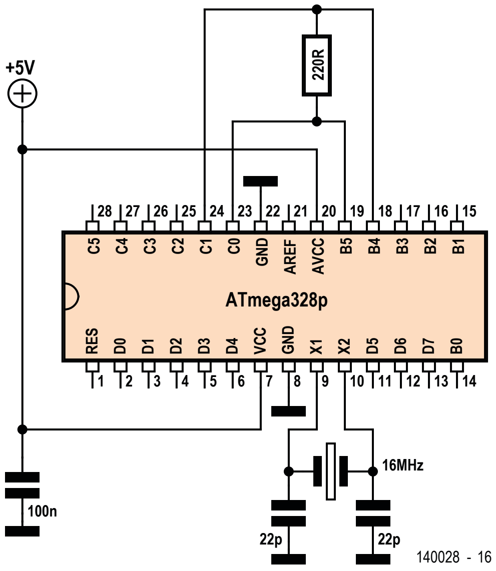
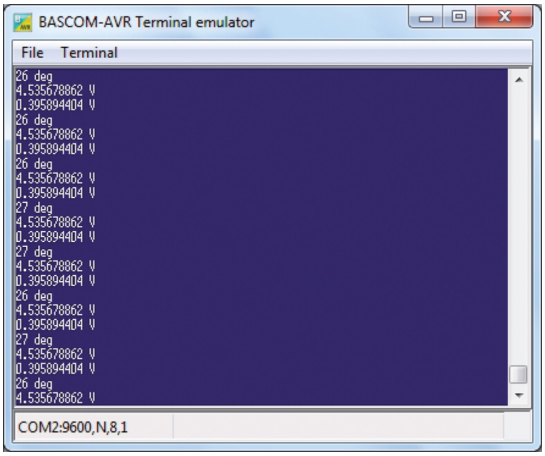


Figure 6. Temperature measurement under load.

Then we wanted to see whether the temperature of the microcontroller increases when it is driving a load. **Figure 6** shows a 220 Ω resistor connected between two port pins. If one output is set high and the other is set low, the voltage on the load resistor is close to 5 V. This results in a load current of roughly 20 mA. We wanted to see whether this causes the microcontroller to warm up. The program also measures the voltage drops on the port pins. For this purpose, the reference voltage must be briefly switched to 5 V. Although it is somewhat unusual for a program to work with two different reference voltages, switching voltages while the program was running does not cause any problems.

The results: with an output current of 20 mA, the voltage drops are just 0.4 V at the lower output and 0.5 V at the upper output. This leaves 4.1 V across the load resistor, yielding a current of 19 mA. The additional power dissipation in the microcontroller is 17 mW (0.9 V x 19 mA). Significant warming cannot be expected from this amount of power, and that's exactly what we saw from the temperature measurement (**Figure 7**).



```
26 deg
4.535678862 V
0.395894404 V
26 deg
4.535678862 V
0.395894404 V
26 deg
4.535678862 V
0.395894404 V
27 deg
4.535678862 V
0.395894404 V
27 deg
4.535678862 V
0.395894404 V
26 deg
4.535678862 V
0.395894404 V
27 deg
4.535678862 V
0.395894404 V
26 deg
4.535678862 V
```

COM2:9600,N,8,1

Figure 7. Temperatures and voltage drop.

According to the data sheet, the maximum current at any port pin should not exceed 40 mA and the total current should not exceed 200 mA. In our experiments we found that the ports could handle currents up to 100 mA without damage, which means that even small DC motors could be operated directly from the ports without driver ICs. However, if you want to be on the safe side the rule is to never exceed 40 mA.

Measuring the input hysteresis

In the previous instalment of this series we used external test equipment to determine the switching thresholds of a digital input. Now we want to the same thing entirely automatically with the built-in A/D converter (see **Figure 8**). For this we use two new variables (Dmin and Dmax) to hold the values of the lower and upper switching thresholds (see **Listing 6**). The measurement loop is preceded by a control loop which ensures that the input voltage is already within the desired range before the actual measurement starts. At first the maximum possible value is stored in the Dmin variable and zero is stored in the Dmax variable. The program then looks for the actual limit values. This is done using a random-sample approach. The longer the measurement session lasts, the more accurate the determined limit values are.

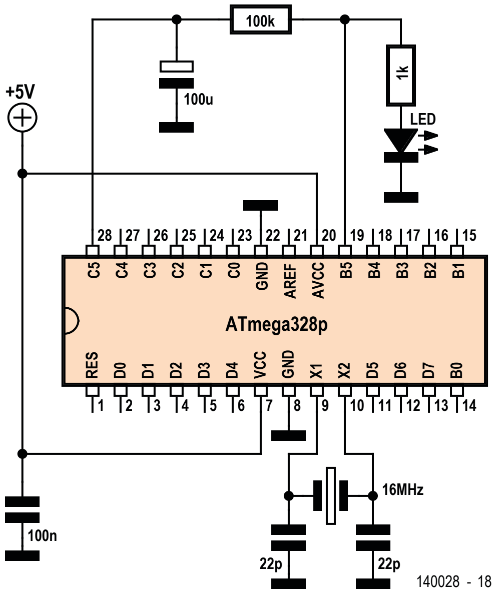

100k
100u
1k
LED
+5V
ATmega328p
C5 C4 C3 C2 C1 C0 GND AREF AVCC B5 B4 B3 B2 B1
RES D0 D1 D2 D3 D4 VCC GND X1 X2 D5 D6 D7 B0
16MHz
100n
22p
22p
140028 - 18

Figure 8. Measuring the input hysteresis.

```
'----------------------------------------------------
'UNO_AD5.BAS   Hi/Lo Input Threshold
'----------------------------------------------------
$regfile = "m328pdef.dat"
$crystal = 16000000
$baud = 9600

Dim D As Word
Dim Dmin As Word
Dim Dmax As Word
Dim N As Word
Dim U As Single
Config Portb = Output
Config Adc = Single , Prescaler = 64 , Reference = Avcc '5 V

For N = 1 To 10000
     Portb.5 = Not Pinc.5
Next N

Dmin = 1023
Dmax = 0

Do
  For N = 1 To 10000
     Portb.5 = Not Pinc.5              '13 100k A5
```

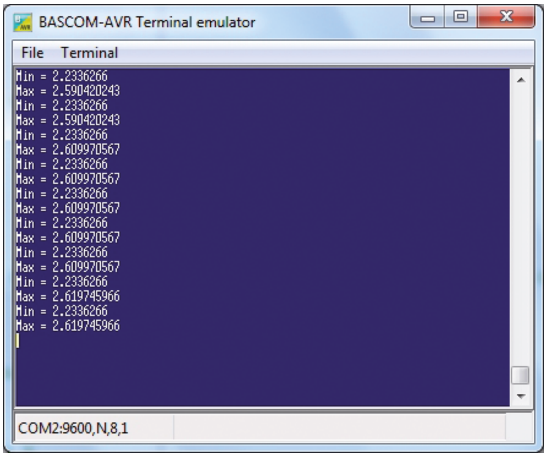
```
        D = Getadc(5)                     ‘A5 100µF GND
        If D < Dmin Then Dmin = D
        If D > Dmax Then Dmax = D
     Next N
     U = Dmin * 5.0
     U = U / 1023
     Print “Min = “;
     Print U , “ V”
     U = Dmax * 5.0
     U = U / 1023
     Print “Max = “;
     Print U , “ V”
   Loop
```

Listing 6: Measuring the input hysteresis.

This program shows how to use loop counters. The For-Next loop counts down from N = 1000 to 0 and executes the instructions inside the loop exactly 1000 times. Although one thousand iterations may sound like a lot, the entire process takes only a fraction of a second. The large number of measurements increases the probability that the measurement results include the true limit values. This means that you have a good chance of finding the correct values after just one round. This can be seen from the fact that the subsequent rounds do not yield any significant changes.

```
For N = 1 To 10000
     … Anweisungen …
Next N
```

Figure 9 shows the result of this experiment. The switching thresholds are 2.23 V and 2.62 V. These are approximately the same as the values determined using the external test equipment..



```
Min = 2.2336266
Max = 2.590420243
Min = 2.2336266
Max = 2.590420243
Min = 2.2336266
Max = 2.609970567
Min = 2.2336266
Max = 2.609970567
Min = 2.2336266
Max = 2.609970567
Min = 2.2336266
Max = 2.609970567
Min = 2.2336266
Max = 2.609970567
Min = 2.2336266
Max = 2.619745966
Min = 2.2336266
Max = 2.619745966
```

COM2:9600,N,8,1

Figure 9. The upper and lower switching thresholds of an input.

External programmer
When you want to use a microcontroller, you usually need some sort of programming device. This is not necessary with the Arduino because it has a built-in boot loader, as described in the first part of this series. However, you do not have a boot loader when you buy a new AVR microcontroller and fit it on your own PCB, so you have to use a programmer. One low-cost option is the Atmel ISP mkII. It can be used with the free AVR Studio 6 development environment. However, there are lots of other programmers that can do the same job.

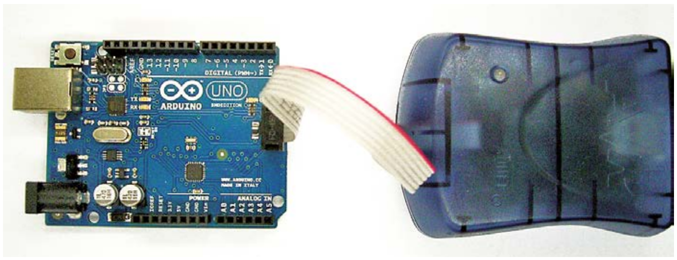
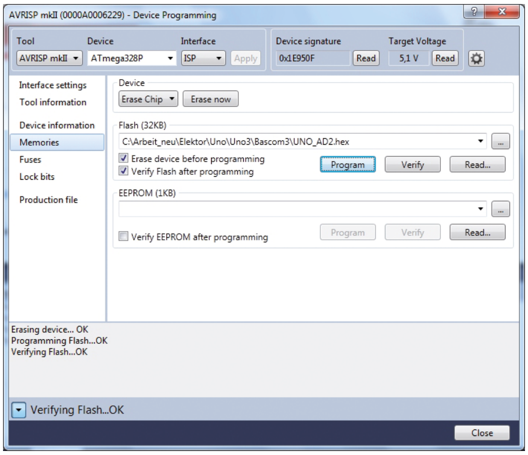


Figure 10. Arduino and a programmer device.

A six-pin ISP connector is normally used for in-system programming (ISP) of AVR microcontrollers. Many microcontroller boards, including the Arduino, have a suitable 6-pin header for this. The SPI interface, which consists of serial data streams (MOSI to the microcontroller and MISO back to the programmer) and a clock signal (SCK), is often used to transmit the programming data. This interface will be described in more detail in a later instalment. The pinout of the ATmega328 is:

1 MISO, PB4
2 Vcc
3 SCK, PB5
4 MOSI, PB3
5 Reset
6 GND

The Arduino Uno board has two ISP ports for external programmers. The one close to the USB connector is best ignored, since it is used to program the USB to serial converter described in the body of this article. The ISP port for the ATmega328 is located at the edge of the board. It's nice to know that it's possible to connect an external programmer, because it means you can always restore everything to the original state if you do something really wrong, such as accidentally deleting the boot loader.



AVRISP mkII (0000A0006229) - Device Programming

Tool: AVRISP mkII
Device: ATmega328P
Interface: ISP
Device signature: 0x1E950F
Target Voltage: 5,1 V

Device: Erase Chip

Flash (32KB): C:\Arbeit_neu\Elektor\Uno\Uno3\Bascom3\UNO_AD2.hex
☑ Erase device before programming
☑ Verify Flash after programming

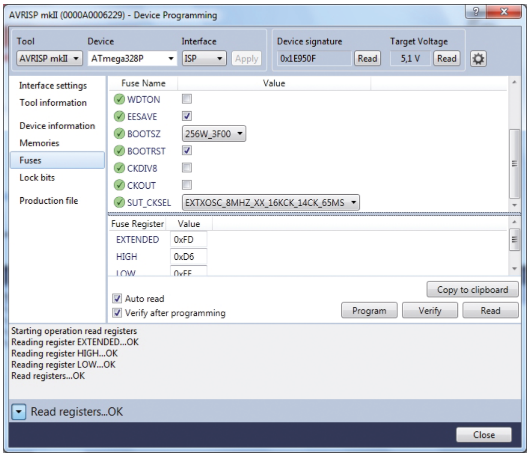
EEPROM (1KB)
☐ Verify EEPROM after programming

```
Erasing device... OK
Programming Flash...OK
Verifying Flash...OK
```

Verifying Flash...OK

Figure 11. Programming Flash.

You can also use the programmer to read out the content of the flash memory, for example in order to make a backup copy (including the boot loader) that you can use to restore the Arduino if necessary. A backup copy (UnoBoot.hex) [1] is also available in the Elektor archive in case you need it.



AVRISP mkII (0000A0006229) - Device Programming

Tool: AVRISP mkII
Device: ATmega328P
Interface: ISP
Device signature: 0x1E950F
Target Voltage: 5,1 V

Fuse Name	Value
WDTON	☐
EESAVE	☑
BOOTSZ	256W_3F00
BOOTRST	☑
CKDIV8	☐
CKOUT	☐
SUT_CKSEL	EXTXOSC_8MHZ_XX_16KCK_14CK_65MS

Fuse Register	Value
EXTENDED	0xFD
HIGH	0xD6
LOW	0xFF

☑ Auto read
☑ Verify after programming

```
Starting operation read registers
Reading register EXTENDED...OK
Reading register HIGH...OK
Reading register LOW...OK
Read registers...OK
```

Read registers...OK

Figure 12. Programming Fuses.

In order to load your own program into the microcontroller, you first have to load the hex file. In this case you must tick the option 'Erase chip before programming' to ensure that the boot loader is deleted. This means you have to decide whether or not you want to use the boot loader. You can also opt to use another boot loader, such as the MCS boot loader. We'll describe how that works in one of the upcoming instalments.

You can also view the microcontroller fuse settings with Bascom or in Atmel Studio 6. By the way, the term 'fuse' comes from the early days of microcontroller technology when there was program memory in which links were literally burnt through to program the memory once and for all. The modern fuses in ATmega microcontrollers are actually flash memory cells that must be configured to define specific basic settings. Among other things, they allow you to switch back and forth between the internal clock and an external clock source. The screenshot shows the factory default fuse settings of the Arduino Uno board. You can see that the boot loader is enabled (BOOTRST) and how much memory is reserved for it.

Web Link
[1] www.elektormagazine.com/magazine/elektor-201405/26437

Chapter 42 • Microcontroller BootCamp (4)

User interfaces

Serious microcontroller applications usually have some sort of interface between humans and the machine. You can do a lot with a just a small display, a few buttons, a potentiometer and a couple of LEDs. What's more, you can hit the ground running with the Arduino shield developed by Elektor, which is described elsewhere in this edition.

First let's see what it takes to drive a display. Various types are available, including graphic and text displays. Text displays range from tiny with a single line of 8 characters, to large with four lines of 20 characters. As the saying goes, small is beautiful, and in many cases you don't have to display a lot of characters. That's why the new Elektor shield has a small two-line text display with eight characters per line. In any case, that's enough room for messages such as "V=3.3V" or "I=8.2mA". What more do you need on your electronics bench?

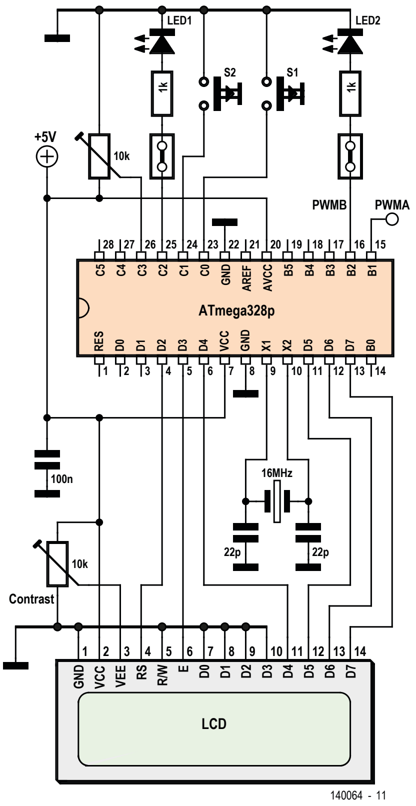

LED1
LED2
S2
S1
1k
1k
+5V
10k
PWMB
PWMA
C5 C4 C3 C2 C1 C0 GND AREF AVCC B5 B4 B3 B2 B1
ATmega328p
RES D0 D1 D2 D3 D4 VCC GND X1 X2 D5 D6 D7 B0
100n
16MHz
22p
22p
10k
Contrast
GND VCC VEE RS R/W E D0 D1 D2 D3 D4 D5 D6 D7
LCD
140064 - 11

Figure 1. Basic diagram: how the LEDs, buttons and display on the Elektor shield are connected to the ATmega328 on the Arduino Uno board.

LCD connection

Figure 1 shows how the display module on the shield is connected to the ATmega328 on the Arduino board. You can also see the other peripheral components there: two LEDs, two pushbuttons and a potentiometer. If you wish, you can build all this yourself on a piece of prototyping board. You don't even have to use exactly the same type of display module. One with two lines of 16 characters would work just as well. If you like to keep things simple, you can order the new shield on the web page for this article [1], where as usual you can also download a file containing the code for the programs described here.

The display occupies six pins on port D: D2 to D7. That's handy, because it leaves exactly two spare for the serial interface: D0 (RXD) and D1 (TXD). Ports B and C remain free for whatever strikes your fancy. All standard display modules of this sort can optionally be operated in 8-bit mode, which means that the data is sent to the display over eight lines in parallel. However, 4-bit mode has become customary to reduce the number of port pins used on the microcontroller. In this mode each data byte is sent to the display in two steps. For example, if you want to write an upper-case A you have to send the hex value &H41 (binary &B01000001, or 65 in decimal notation), as specified in the ASCII code table. In fact you first send &B0001 and then &B0100. However, you don't actually need to know all these details or how to go about initializing the display, since Bascom does all the hard work for you. If you simply write Lcd "A" in your program, the letter A will appear on the display.

```
'------------------------------------
'UNO_LCD1.BAS  Text Output
'------------------------------------
$regfile = "m328pdef.dat" 'ATmega328p
$crystal = 16000000       '16 MHz
$baud = 9600

Dim N As Word

Config Lcdpin = Pin , Db4 = Portd.4 , Db5 = Portd.5 , Db6 = Portd.6 , Db7 =
Portd.7 , E = Portd.3 , Rs = Portd.2
Config Lcd = 16 * 2

Cls
Lcd "Elektor"
'Cursor Off
Do
  Locate 2 , 1
  Lcd N
  N = N + 1
  Waitms 1000
Loop
```

Listing 1: Text and number output.

Our first example program (**Listing 1**) shows how it all works. First you need a Config instruction to tell the compiler how the display pins are connected to the port pins. Next you have to initialize the display with the Config Lcd = 16 * 2 instruction. You can choose from several formats here. The format 8 * 2 is not supported because the display controller in the LCD module is designed for two lines of 16 characters (16 * 2), even with this small display. That's not a problem in practice because the program keeps on running without an error if you accidentally write up to 16 characters in a line. However, when you see the truncated output on the display you will have to consider how you can shave a few more characters off the message.

After you initialize the display, you should clear everything with the Cls (Clear Screen) command. Everything written after this lands on the first line, starting at the left end. Here it is the word "Elektor" – seven characters, short and sweet. If you send another character to the LCD now, it will be written to position 8 on the first line, but what you actually want is to start with the second line. For this you use the Locate command. Here Locate 2 , 1 causes the next character to be written to the leftmost position of the second line. The program outputs an incrementing number at this position. The count variable N is incremented once per second, so you can use the program to measure time. For example, it's taken me 1200 seconds to write these lines since I started the program. That's 20 minutes, which means I'm pretty slow today.

There are two Lcd instructions in this example program, and it's important to understand that they do two completely different things. The pure text output instruction simply copies the characters set in quotation marks by the programmer. By contrast, the instruction Lcd N converts the numerical value in the variable N into text and then sends this text to the display. In this case N is an integer, but in other cases it could be a real number with decimal places. Bascom decides on its own what has to be done, which is very convenient for the user. In other languages the programmer has to do more in these situations.

On the display you will see an underscore after the number that was output, which represents a cursor. The cursor shows where the next character will be written if and when it comes. This may be very helpful when you're busy typing something in, but in our case the cursor is irritating. Fortunately, we can do something about this with the Cursor Off command, which is initially commented out in the code. If you delete the comment character, recompile the program and download it to the microcontroller, you will see that the cursor has vanished.

This is a good place to remind you how easy it is to display more information by using the Bascom help function. Simply move the cursor (here we mean the one on your PC monitor) to the keyword concerned and press the F1 key, and you're already a lot wiser. For instance, you can learn how to make the cursor blink.

A two-channel voltmeter

Happen to need a two-channel voltmeter? **Listing 2** shows one of many possible solutions. In the previous instalment we told you how to convert the data provided by the A/D converter into a voltage reading. Now let's see how you can show it on the display. First we write a string at the start of each line that identifies the input. The is followed by a space character to separate it from the reading. We also tack on a couple of spaces after the reading (in this case a real number) has been output. Without them, the following situation could occur: Your last reading was 4.859 V, and the new reading is 5.0 V. Bascom outputs “5.0” just as it should, but the rest of the previous output is still there on the display, so you see “5.059 V” and wonder what’s going on. To prevent this from happening, we always delete anything that might be present after the current output. Here we accept the risk that this may involve characters intended for a larger display that are not visible on the actual display.

```
'-----------------------------------
'UNO_LCD2.BAS  Voltage A0, A3
'-----------------------------------
$regfile = "m328pdef.dat" 'ATmega328p
$crystal = 16000000       '16 MHz
$baud = 9600

Dim D As Word
Dim U As Single

Config Adc = Single , Prescaler = 64 , Reference = Avcc     '5 V

Config Lcdpin = Pin , Db4 = Portd.4 , Db5 = Portd.5 , Db6 = Portd.6 , Db7 =
Portd.7 , E = Portd.3 , Rs = Portd.2
Config Lcd = 16 * 2

Cls
Cursor Off
Do
  Locate 1 , 1
  Lcd "A0 "
  D = Getadc(1)         'A0
  U = D * 5.0
  U = U / 1023
  Lcd U
  Lcd "    "

  Locate 2 , 1          'Pot
  Lcd "A3 "
  D = Getadc(3)
  U = D * 5.0
  U = U / 1023
  Lcd U
  Lcd "    "
  Waitms 1000
Loop
```

Listing 2: Anzeige von Spannungen.

Incidentally, this program uses the ADC0 and ADC3 channels for its measurements. You can connect a signal source to ADC0, preferably with a protective series resistor as shown in **Figure 2**. There's already something connected to ADC3: the potentiometer on the Elektor shield. You can adjust it to provide a voltage from 0 to 5 V.

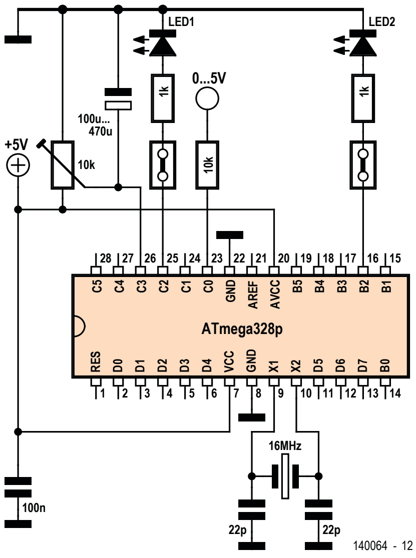


Figure 2. Using the analog inputs.

All of the Arduino pins are also fed out to socket headers on the shield. If you connect a hefty electrolytic capacitor (you surely have an electrolytic somewhere on your bench – but don't exceed 470 µF) between the GND and ADC3 pins, which puts it in parallel with the potentiometer, you have a low-pass filter and you can see on the display how the voltage gradually adjusts after you change the potentiometer setting. The reason for the 470 µF limit is that you might allow the capacitor to charge slowly to 5 V and then quickly turn the potentiometer towards zero. If the capacitor is too large, the resulting discharge current could fry the potentiometer.

```
Do
  Locate 1 , 1
  Lcd "A2 "
  D = Getadc(2)          'LED1
  U = D * 5.0
  U = U / 1023
  Lcd U
  Lcd "    "

  Locate 2 , 1
  Lcd "A3 "
  D = Getadc(3)          'Pot
```

```
    U = D * 5.0
    U = U / 1023
    Lcd U
    Lcd "    "
    Waitms 1000
  Loop
```

Listing 3: Measuring the LED voltage and the potentiometer voltage.

Another thing you might notice is that at low light levels the voltage measured on ADC2 is affected to a certain extent by the potentiometer voltage on ADC3. This is due to the sample-and-hold capacitor at the input of the A/D converter. This capacitor is first charged to the voltage present on the measurement input, and then its charge is measured. If the capacitor last saw 3 V from the potentiometer during the previous measurement, some of its charge will still be left because it cannot discharge quickly enough through the extremely high impedance of the LED. For this reason, the microcontroller data sheet recommends that the internal resistance of the signal source should not exceed 10 kΩ. Here we are dealing with many megohms instead, so it's better to set the potentiometer to zero. Then you will see that there is still a measurable voltage at low light levels. This circuit shows that the A/D converter has very high input impedance. A typical digital voltmeter with an input impedance of 10 MΩ would not be able to measure the LED voltage, but the ATmega can do so.

With very high impedance signal sources, you can connect a 10 nF bypass capacitor in parallel as shown in **Figure 3**. This makes the measurements much more reliable. Even with very dim light, you can now measure a voltage of roughly 1 V. For all PN junctions with their exponential characteristic curves, the rule of thumb is that the voltage rises by approximately 70 mV when the current is increased by a factor of 10. This means that the span between 1.0 V and 1.5 V corresponds to seven decades of current range, and thus seven decades of brightness. That should be more than enough for measuring from 10 lux to 100,000 lux, and we can see the makings of a light meter here. Maybe that's an attractive job for an enthusiastic reader? There's nothing wrong with a nice collection of Bascom programs with contributions from many people.

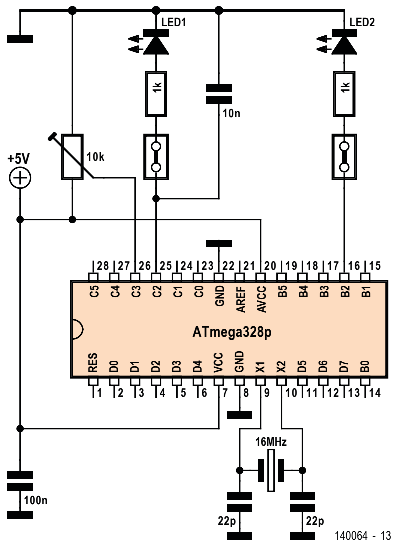


Figure 3. Using an LED as a light sensor.

PWM outputs

When precise timing matters, timers come to the fore. Most microcontrollers have several timers. Timer1 of the ATmega328 has a resolution of 16 bits. You can regarded it as a sixteen-stage counter, similar to the well-known CD4040 (which has only twelve stages). A clock signal (or other pulse signal) entering at one end comes out at the other end with a lower frequency. You can use the CD4040 to build a clock generator or a frequency divider, or as the basis for a binary pulse counter. Timer1 can also handle all of these functions; you just have to initialize it properly. There are also two 8-bit timers on board (Timer0 and Timer2).

One of the many operating modes of Timer1 is PWM mode. It can generate two pulse width modulated signals, which means rectangular pulse signals with adjustable mark/space (on/off) ratio. Among other things, they can be used to control the brightness of a lamp, much in the same way as a dimmer on the AC line which simply switches the voltage on and off at a high rate. Here we want to do the same thing by using the PWM output to control the brightness of an LED.

The PWM output has a maximum adjustment range of 10 bits, but it can also be configured in 8-bit mode. Here 10 bits is an attractive choice because it matches the resolution of the A/D converter. In both cases the counter runs from 0 to 1023. You could for example program the output to switch off when the counter reaches 511, which results in a PWM signal with a 50% duty factor. This is achieved by repeatedly comparing the counter value with the programmed value in a compare unit. There are two compare units in the microcontroller, so you can generate two independent PWM signals with the same timer. PWM1A is output on port pin PB1, while PWM1B is output on pin PB2. That reminds us of something. You guessed it: LED 2 is connected to PB2 through a 1 kΩ series resistor. Coincidence? Not at all, because now you can use the PWM signal to control the brightness of the LED.

The program in **Listing 4** uses the PWM1A output together with the A/D converter input ADC3, which is connected to the potentiometer. The measured value of the potentiometer setting is used as the comparison value for the PWM signal. If you set the potentiometer to mid-range, the PWM output is a symmetrical square wave. Everything from 0 to 1023 is possible. The actual output value is shown on the LCD. If you want to look at the signal, you can connect a scope probe to PB1 (Arduino pin 9).

```
'-------------------------------------
'UNO_LCD3.BAS  PWM
'-------------------------------------
$regfile = "m328pdef.dat" 'ATmega328p
$crystal = 16000000       '16 MHz
$baud = 9600


S1 Alias Pinc.0
S2 Alias Pinc.1

Dim D As Word
Dim U As Single

Config Adc = Single , Prescaler = 64 , Reference = Avcc     '5 V

Config Timer1 = Pwm , Prescale = 1 , Pwm = 10 , Compare A Pwm = Clear Up ,
Compare B
Pwm = Clear Up

Config Lcdpin = Pin , Db4 = Portd.4 , Db5 = Portd.5 , Db6 = Portd.6 , Db7 =
Portd.7 , E = Portd.3 , Rs = Portd.2
Config Lcd = 16 * 2

Portc.0 = 1                    'Pullup
Portc.1 = 1                    'Pullup
Config Portb = Output
Cls
Cursor Off
Do
  Locate 1 , 1
  Lcd "PWMA="
  A = Getadc(3)
  Pwm1a = A
  Lcd A
  Lcd "    "

  If S1 = 0 Then
    If B > 0 Then B = B - 1
  End If
  If S2 = 0 Then
    If B < 1023 Then B = B + 1
  End If
  Pwm1b = B

  Locate 2 , 1
  Lcd "PWMB="
  Lcd B
  Lcd "    "
  Waitms 100
Loop
```

Listing 4: PWM control

What is the frequency of the PWM signal? That depends on the frequency of the clock input to the timer. It can be the clock signal from the Arduino board (16 MHz), but it can also be a lower-frequency signal derived from the clock signal by passing it through a prescaler. Here the initialization instruction `Prescale = 1` means that the full clock frequency is used. You might think that this yields a PWM frequency of 15.625 kHz (16 MHz divided by 1024), but in fact it is only half of this (approximately 7.8 kHz). This come from the fact that Bascom uses the "Phase Correct PWM" mode instead of the "Fast PWM" mode. You can learn more about this from the microcontroller data sheet, but that's more like a book than a data sheet. Here Bascom saves developers time and effort by always choosing a reasonable option for each setting. This means you can obtain working results with Bascom even if you don't fully understand all the details.

LED 2 is connected to the second PWM output (PB2). In our program this output is controlled by the two pushbuttons S1 and S2. To make the program easy to read, the `Alias` instruction is used to assign the button names to the port inputs. This means that when you write If `S1 = 0 Then` in your code, Bascom knows that what you actually mean is `If Pinc.1 = 0`, so the state of port pin PC1 has to be polled. Another important consideration is that the buttons are connected to ground and therefore require pullup resistors. As a result, the program normally sees the quiescent state S1 = 1. This only changes to S1 = 0 when someone presses the button. That makes things very simple. When you press S1 you reduce the PWM output level (assuming it is greater than 0), and when you press S2 you raise the PWM output level (assuming it is less than 1023). This proceeds in single steps if you press the button briefly, but if you press and hold the button it changes continuously at ten steps per second. The current output value is always shown on the LCD.

Button polling

The program shows a very simple example of how to use buttons and corresponding program branching. Of course, there are lots of other ways to obtain the desired result. Developing the ideal user interface for a given task is a fascinating job. You can configure a wide variety of things with two buttons and a potentiometer. A lot can be achieved with just a few buttons.

The program uses everything that the Elektor shield has to offer. The LCD shows the two current PWM values, the potentiometer controls the PWMA output, and the two buttons control the PWMB output and thereby LED 2. Is that really everything? Well, not quite—LED 1 isn't doing anything. So let's plug a wire into PB1 (Arduino pin 9) and ADC2 (Arduino A2) to connect PWMA to LED 1 through a 1 kΩ resistor and a jumper (**Figure 4**). Now you can control the brightness of the LED with the potentiometer.

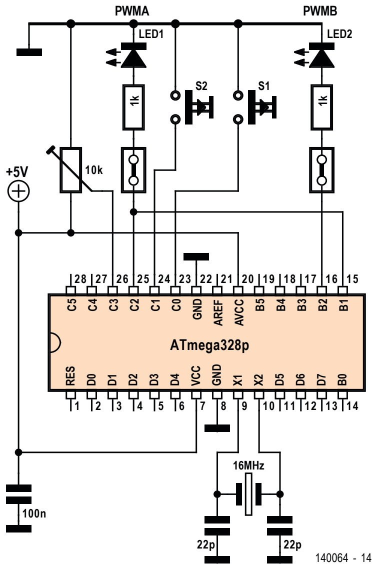


Figure 4. A pair of LEDs connected to the PWM outputs.

With two LEDs whose brightness can be set independently, you have the makings of a little skill-testing game. The first player sets the brightness of LED 2 to some arbitrary value using the buttons. Then this LED is covered, and the second player tries to set LED 1 to the same brightness using the potentiometer. The LCD reveals how well this went. Then the two players swap roles. All differences between the two values are recorded. The player with the lowest score at the end is the winner and is designated "Hawkeye".

Liquid Crystal Displays

Liquid crystal displays (LCDs) are very popular because they are easy to read and don't need much power. They can be operated without a backlight, although the backlight on the Elektor shield is always on. This still results in much less power consumption than an LED display with the same number of characters.

The reason LCDs are so energy efficient is that they do not have to generate light. Instead, they control whether ambient light is transmitted or blocked. The liquid crystal material, which is located between two sheets of glass, rotates the polarization plane of the incident light depending on the applied voltage. A polarization film is always located on one of the glass sheets. (Incidentally, if you have a defective LCD module you should sometime pull off this film, because you can use it to perform interesting experiments with light [2][3].)

The display element of an LCD module is driven by square-wave voltages with no DC component, which are applied across the individual segments. There are dedicated CMOS ICs for this purpose, but some microcontrollers can also drive LCDs directly. However, standard LCD modules have special built-in controllers that handle the actual drive tasks.

Figure 5. Liquid crystal displays (LCDs).

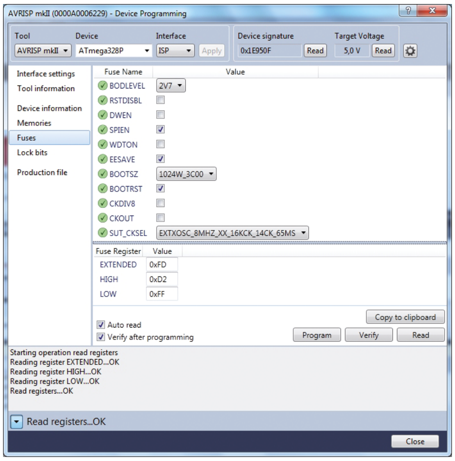
The great-grandfather of all LCD controllers was the HD44780. Even today, most controllers are HD44780 compatible. This involves several standardized control commands as well as the connections to the display module. There are usually 14 lines, and in some cases two more for the backlight. The display on the Elektor shield need only 14 lines because the backlight is hardwired.

About the individual lines: GND and Vcc are obviously ground and +5 V. Vee is connected to a contrast trimpot. With many LCD modules the optimal setting is approximately 4.5 V between Vee and Vcc. It is therefore often sufficient to connect a fixed resistor with a value of 470 Ω to 1 kΩ between Vee and ground, which saves the cost of a trimpot.

RS is an input that allows the display be set to receive either data commands (RS = 1) or control commands (RS = 0). Data is necessary to display text, while control commands are necessary for initialization and positioning the cursor. R/W switches between reading and writing. Reading is actually only necessary to determine whether the display is ready to receive the next data. This can also be ensured by a short wait time, so the read direction is often not used and R/W is tied to ground. E is the Enable input. It is used to indicate to the display controller that valid data is present on the data bus (lines D0 to D7) and should be read in by the controller. The controller always reads in the current data after a falling edge. In 8-bit mode all eight data bits are put on the bus and then a falling edge is generated on the E input. In 4-bit mode this process occurs twice, with only the upper four data lines (D4 to D7) being used for data transfer. The lower nibble (least significant bits) is sent first, followed by the upper nibble. The display must be initialized to operate in either 4-bit mode or 8-bit mode before it is used.

MCS Boot Loader

Up to now we have described two options for programming the Arduino: using the Arduino boot loader or using the ISP interface, which is also brought out on the Elektor shield. With an external programmer, you can also use the ISP interface to load a different boot loader.


AVRISP mkII (0000A0006229) - Device Programming

Tool: AVRISP mkII | Device: ATmega328P | Interface: ISP | Device signature: 0x1E950F | Target Voltage: 5,0 V

Fuse Name	Value
BODLEVEL	2V7
RSTDISBL	☐
DWEN	☐
SPIEN	✓
WDTON	☐
EESAVE	✓
BOOTSZ	1024W_3C00
BOOTRST	✓
CKDIV8	☐
CKOUT	☐
SUT_CKSEL	EXTXOSC_8MHZ_XX_16KCK_14CK_65MS

Fuse Register	Value
EXTENDED	0xFD
HIGH	0xD2
LOW	0xFF

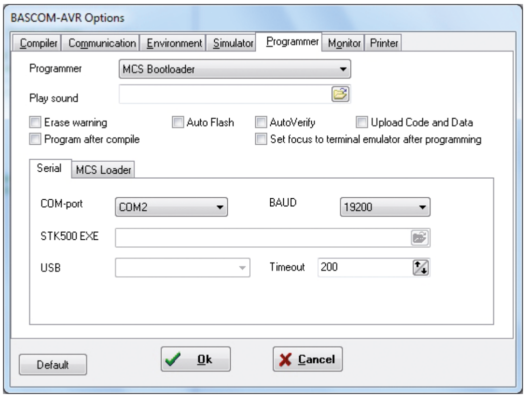
✓ Auto read
✓ Verify after programming

```
Starting operation read registers
Reading register EXTENDED...OK
Reading register HIGH...OK
Reading register LOW...OK
Read registers...OK
```

Read registers...OK

Figure 6. Fuse settings.

Bascom has the MCS boot loader, which can be adapted to various microcontrollers. That's attractive because it allows you to equip different microcontroller boards with the same boot loader. The source code is included with the Bascom example programs. All that is necessary for adaptation is to configure the microcontroller type, the clock frequency and the desired baud rate. The configuration for the ATmega328P was not yet there, but the adaptation was easy. The fully adapted boot loader, including the source code and hex file, can be downloaded from the Elektor website [1]. The files are called BootLoaderUno_16.bas (16 stands for 16 MHz) and BootLoaderUno_16.hex. Note that the fuse settings must be slightly different (see the first AVR Studio screenshot) than for the Arduino boot loader because more memory space is needed. The boot area is 1024 words with the MSC boot loader and starts at &H3C00. If you intend to use the MSC boot loader, you must configure Bascom accordingly. Here again it is important to set the right COM port and the right baud rate, in this case 19,200 (see the second screenshot).



BASCOM-AVR Options

Compiler | Communication | Environment | Simulator | Programmer | Monitor | Printer

Programmer: MCS Bootloader

Play sound:

☐ Erase warning ☐ Auto Flash ☐ AutoVerify ☐ Upload Code and Data

☐ Program after compile ☐ Set focus to terminal emulator after programming

Serial | MCS Loader

COM-port: COM2 BAUD: 19200

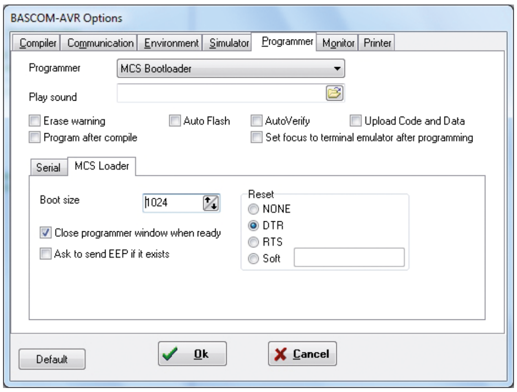
STK500 EXE:

USB: Timeout: 200

Default | Ok | Cancel

Figure 7. Choice of the Bootloader.

There is another important setting on the MCS Loader tab. As you may recall, data is transmitted to the Arduino board over the USB bus, but it arrives through a simulated RS232 port. All Arduino systems use the RS232 DTR line as a Reset line; a falling edge on this line triggers a reset. That's what starts the boot loader, because the microcontroller runs the program code stored at the boot address (&H3C00) when it starts up. This code waits for a specific action from the PC. Either new program comes down the pipe or it doesn't, and then execution branches to the start of program memory at address zero. This process occurs with every reset, which is what allows Bascom to force a jump to the boot loader by changing the level on the DTR line. Users have it easy because they don't have to press the reset button themselves.



BASCOM-AVR Options

Compiler | Communication | Environment | Simulator | Programmer | Monitor | Printer

Programmer: MCS Bootloader

Play sound:

☐ Erase warning ☐ Auto Flash ☐ AutoVerify ☐ Upload Code and Data

☐ Program after compile ☐ Set focus to terminal emulator after programming

Serial | MCS Loader

Boot size: 1024

☑ Close programmer window when ready

☐ Ask to send EEP if it exists

Reset: ○ NONE [● DTR] ○ RTS ○ Soft

Default | Ok | Cancel

Figure 8. Setting of the Bootloader.

Web Links

[1] www.elektor-magazine.com/140064

[2] http://b-kainka.de/bastel49.htm (in German)

[3] http://en.wikipedia.org/wiki/Polarizer

Chapter 43 • Microcontroller BootCamp (5)

Using timers

Flip-flops, dividers and counters are fundamental components in digital circuits. They are also found in microcontrollers, where they have a wide range of uses: here we look at a few.

One of the longest chapters in the ATmega328 data sheet is the one that describes its three timers. The timers can be used in so many different ways that we only have space here to look at a small fraction of the possibilities. The main application areas are in measuring time intervals and frequencies, and in generating various signals including PWM output.

Measure those microseconds

The need to measure time intervals often crops up in electronics. For example, we might want to know how long it takes to output some data to an LCD module: is it milliseconds or microseconds? Without some inside knowledge, the only way is to measure it. The ATmega328 has a suitable module already on board in the form of Timer1, which has a resolution of 16 bits. Now, all the timer module does is simply act as a counter of regular events. In this case, with a 16-bit counter, the maximum count is 65535, after which the counter returns to zero: this is called 'overflow'. If things are arranged so that the counter increments once a microsecond we can measure time intervals of up to 65535 µs. Between the crystal oscillator and the clock input of the counter there is a programmable prescaler (similar to the arrangement for the A/D converter) that divides down the clock frequency. If a 16 MHz crystal is used then the required division ratio to obtain a 1 MHz clock for the timer is 16. So our first attempt reads

```
        Config Timer1 = Timer , Prescale = 16
```

but unfortunately this does not work. The error message we get is 'Prescale value must be 1, 8, 64, 256 or 1024', and the data sheet of the ATmega328 confirms the problem. So, we shall set the prescaler ratio to 8 and obtain an input clock to the timer of 2 MHz and a maximum interval measurement of 32767.5 µs. At any time the 16-bit register that holds Timer1's value can be read or a new value can be written to it. This makes our example very straightforward. Before outputting the data to the LCD module we set the counter to zero ('Timer1 = 0'). After outputting the data we read the counter value ('D = Timer1') and we have the result of the measurement in units of 0.5 µs. To convert to microseconds we simply divide the result by 2. Listing 1 shows the complete program.

As in all the programs we shall look at here, the result is displayed on the Arduino shield LCD module we described in the previous installment in this series, and simultaneously the result is output to the terminal using a 'Print' command. The program will work even if the display is not connected: Bascom does not check whether the display has actually received the data it sends to it. The first result we see is 'Timer 1 = 0 us': this happens because at this point, before the first measurement, the variable D has not been set. The second result looks rather more interesting, giving the time taken to output the first result to the LCD:

'Timer 1 = 17105 us'. Now in this case four additional digits have had to be sent to the display, and so we would expect the process to take a little longer. And indeed it does, the third result being 21933 µs. The results now stabilize, with variations between successive readings of at most one microsecond.

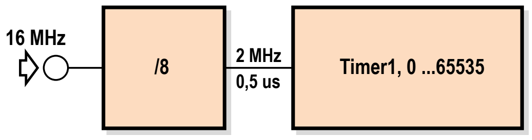


Figure 1. Time measurement using Timer1.

```
'-----------------------------------
'UNO_Timer1.BAS  Timer1 0.5 us
'-----------------------------------
...

Dim D As Word
Config Timer1 = Timer , Prescale = 8  'Clock 2 MHz

Do
  Print "Timer1 = ";
  Print D;
  Print " us"
  Timer1 = 0
  Locate 1 , 1
  Lcd "Timer1 ="
  Locate 2 , 1
  Lcd D
  Lcd " us"
  D = Timer1
  D = D / 2
  Waitms 1000
Loop
```

Listing 1. Measuring the time taken to output to the LCD.

So now we know exactly how long a complex process like writing a string to the LCD takes. The precision and repeatability we have seen would be unimaginable on a Windows or Linux machine, since these complex operating systems do not handle real-time functions well: there is always some process running in the background that makes it impossible to predict exactly how long some action will take. Under Bascom things are different: the processor executes exactly the code you tell it to execute and nothing more. Moreover, being so close to the hardware, some commands such as setting an output bit execute in less than a microsecond. Outputting to the LCD takes around a millisecond per character because Bascom allows plenty of time for the controller in the module to accept the data. If you take a look at the E signal on the LCD (port pin PD3, Arduino pin 3) using an oscilloscope you will see the 1 ms delays.

Measuring the period of a signal

A small modification to our program (see **Listing 2**) allows it to measure the length or period of a pulse. We use input PC0 (Arduino A0: see **Figure 2**), and we will measure the time between two rising edges of the input signal. We need two sampling loops to detect an edge reliably: first we wait until the input reads as a zero, and then we wait until it reads as a one. This will detect the first rising edge, and we set the timer register to zero. We now wait for the second edge, and read the result from the timer.

```
'------------------------------------------
'UNO_Timer2.BAS  Timer1,  0.5 us
'------------------------------------------
...

Dim D As Word

Config Timer1 = Timer , Prescale = 8  'Clock 2 MHz
Config Timer2 = Pwm , Prescale = 256 ,
    Compare A Pwm = Clear Up
Ddrb = 255
Pwm2a = 128

Do
  Do
  Loop Until Pinc.0 = 0
  Do
  Loop Until Pinc.0 = 1
  Timer1 = 0
  Do
  Loop Until Pinc.0 = 0
  Do
  Loop Until Pinc.0 = 1
  D = Timer1
  D = D / 2
  Locate 1 , 1
  Lcd "Timer1 ="
  Locate 2 , 1
  Lcd D
  Lcd " us    "
  Print "Timer1 = ";
  Print D;
  Print " us"
  Waitms 1000
Loop
```

Listing 2. Measuring the period of a signal in microseconds.

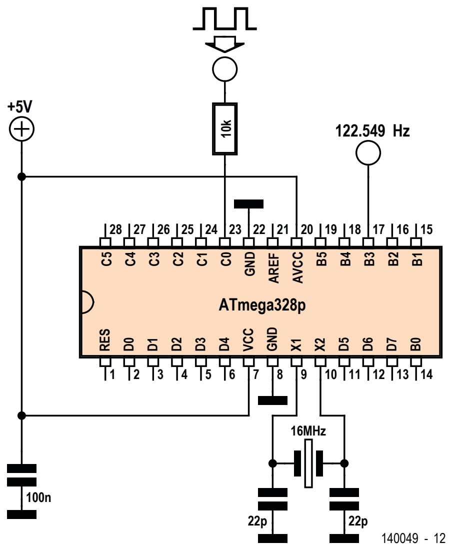


Figure 2. Measuring the period of a signal.

If a finger is touched on the input pin, the microcontroller will receive a signal picked up from the mains supply, which will be at 60 Hz in the USA and 50 Hz in most other countries. We would expect a result of around 16667 µs or 20000 µs respectively. In a real experiment (carried out in Europe) a reading of 20030 µs was obtained, a little on the slow side. Since the mains supply in most of mainland Europe is phase-synchronous, we can only conclude that perhaps too many power stations had been turned off or the sun was not shining brightly enough or the wind not blowing strongly enough. Perhaps it might be worth turning a couple of lights off and trying again...

In order that we have a reliable signal to measure, the program includes its own clock source, taking advantage of the PWM outputs. However, there is a potential problem. PW-M1A and PWM1B cannot be used because we have already committed Timer1 for making the measurement itself. That leaves us with Timer0 and Timer2, each of which can also drive two PWM outputs. However, these extra outputs mostly appear on Port D and would therefore interfere with the LCD interface. An exception is PWM2A which is on port pin PB3 (Arduino pin 11), and so this is the one we use to output our test signal. Timer2 has only an 8-bit counter but if we set its prescaler ratio to 256, we can obtain an output frequency of 16 MHz / 256 / 255 / 2 = 122.549 Hz and hence an output period of 8.16 ms. Connecting PB3 to PC0 lets us measure this signal, and the reading we get is 8160 µs. Result!

Square wave generator, 125 Hz to 4 MHz

A variable-frequency signal generator is often a useful tool to have. At 1 MHz it might be used to test a frequency counter, or at 440 Hz it might be used to tune a violin. The program shown in **Listing 3** covers the whole range from 125 Hz to 4 MHz. Timer1 is used as a frequency divider and the signal is output on the PWM1A pin (B1, Arduino pin 9: see **Figure 3**). This is an example of an application where we are going a little beyond the standard uses of Bascom, and in particular its built-in initialization functions are not suitable for our purposes: we have to program a few registers directly. I borrowed some ideas from Roger Leifert's DCF simulator [1], and he in turn borrowed from the code in Martin Ossman's SDR course [2], which is written in C. We use a variant of PWM output mode where we adjust the frequency while trying to maintain a duty cycle of approximately 50%. Register LCR1 is loaded with the desired division ratio. If, for example, this value is 100, then the output frequency will be 16 MHz / 100 / 2 = 80 kHz. To ensure that the output is a symmetric square wave we need to load register OCR1A with the value 50.

```
'-----------------------------------------
'UNO_Timer3.BAS  B1 Fout 250 Hz...4 MHz
'-----------------------------------------
...

Dim D As Long
Dim F As Long
Dim N As Byte

Config Timer1 = Pwm , Prescale = 1 , Pwm = 10 ,
    Compare A Pwm = Clear Up

Tccr1a = &B10000010    'Phase-correct PWM, Top=ICR1
Tccr1b = &B00010001    'Prescaler=1

D = 8000
Do
  N = Ischarwaiting()
  If N = 1 Then
    Input F
    D = 8000000 / F
    If D > 64000 Then D = 64000
    If D < 2 Then D = 2
  End If
  If S1 = 0 Then
    D = D + 1
    If D > 100 Then D = D + 1
    If D > 1000 Then D = D + 100
    If D > 10000 Then D = D + 1000
    If D > 64000 Then D = 64000
  End If
  If S2 = 0 Then
    If D > 2 Then D = D - 1
    If D > 100 Then D = D - 10
    If D > 1000 Then D = D - 100
    If D > 10000 Then D = D - 1000
    If D > 64000 Then D = 64000
  End If
```

```
    Locate 1 , 1
    F = 16000000 / D
    F = F / 2
    Lcd F
    Lcd " Hz      "
    Icr1 = D
    Ocr1a = D / 2
    Waitms 50
  Loop
```

Listing 3. Timer1 used as a square wave generator.

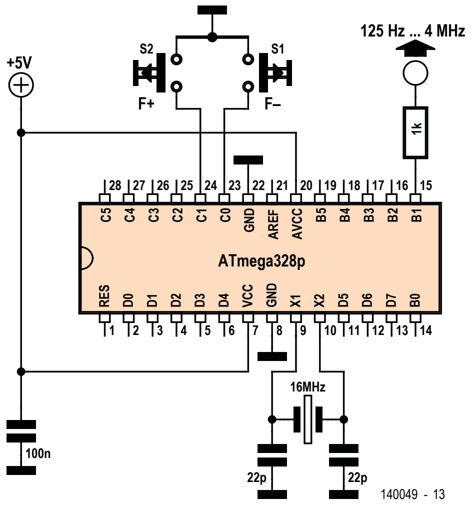


Figure 3. Adjustable square wave generator.

The program can be controlled (simultaneously) in two different ways: over the serial port using a terminal emulator program or using buttons. When pressed, the buttons increase or decrease the division ratio, and the program calculates the resulting frequency and displays it. A brief press of a button changes the ratio in small steps, while a longer press causes the ratio to change (almost) continuously. Since it would be rather tedious to step through all the possible ratios in this way (there are more than 65000 of them), the amount of increment changes depending on the frequency range. This means that many frequencies will only be approximated rather than generated exactly. The output frequency is displayed in Hertz and the numbers involved are such that the calculations must be done not using word variables but with ‘long’ (32-bit) quantities: this applies both to the frequency F and to the division ratio D.

It is perhaps not completely obvious how the program manages to respond both to button presses and to input on the serial port. If we write simply ‘Input F’ then the program will wait at this point until a value is entered, and will not respond to the buttons. We therefore have to check first whether there is a character available on the serial interface: the function IsCharWaiting() returns a value of 1 if there is at least one character available and zero otherwise. In our case, if one character arrives on the serial interface then we know that there are more to come, and we can safely read in a new value for F. From F we can calculate the required division ratio, in many cases obtaining an exact result. At 440 Hz there is only negligible error, although at 549 kHz, for example, only a rough approximation is available: the program chooses a division ratio of 14, which results in an output frequency of 571428 Hz. The smallest division ratios give rise to the highest frequencies, namely 4 MHz, 2.666666 MHz, 2 MHz, 1.6 MHz, 1.333333 MHz and 1 MHz. If these are useful to you, then you could build the design as a self-contained unit.

The steep edges on the signal on output PB1 mean that it contains harmonics well into the VHF range. This means it is all too easy for the signal generator to become a source of radio interference. For example, if you connect the oscilloscope probe to the output but forget to connect the ground clip close by then a large and rather effective VHF loop antenna can be created via the ground connection to the USB port and then through the PC's power supply and the mains. Unsurprisingly this can disrupt reception on nearby FM radios and hence relations with your neighbors! To keep on the right side of the EMC regulations (and your neighbors) it is necessary to use either a screened cable, or a resistor close to the signal generator's output to reduce the amplitude of the higher harmonics. A resistor of 1 kΩ in conjunction with a cable capacitance of 30 pF forms a low-pass filter with a cutoff frequency of 5 MHz. The edges of the signal are smoothed considerably, and VHF interference is reduced by around 20 dB.

Timer interrupts
In the previous installment of this series we looked at an example program that generates a one-second clock. The timing used a simple 'Waitms 1000' command. Now this approach does not give an exact result, as the other parts of the program such as the infinite loop and the code that generates the output all take time. The problem can be avoided using a timer interrupt: that is, allowing a certain part of the code to execute exclusively under control of a timer. It works as follows. A hardware timer counts away, completely independent of other activity in the microcontroller. When it reaches its maximum count and resets to zero ('overflow') the program executing in the foreground is interrupted and a special piece of code called an 'interrupt service routine', or ISR, is called. Within this piece of code we can carry out any actions that need to happen at exactly-specified time intervals. It makes no difference how long the ISR itself takes to execute as the hardware timer is continuing to count: the only thing that matters is that the routine completes before the next time the timer overflows and triggers the interrupt again.

For the job of generating a precise one-second clock Timer1 is overkill: the eight-bit resolution of Timer0 is plenty for this application. We will arrange for the timer to overflow and hence generate an interrupt every 1000 µs. The interrupt service routine, which, as it is triggered from Timer0, we have called 'Tim0_isr:', will thus be called every millisecond. The colon means that 'Tim0_isr:' will be interpreted as a label, marking a point in the code which can be jumped to. The command 'On Ovf0 Tim0_isr' tells the microcontroller to jump to this label whenever there is an overflow ('Ovf') on Timer0. The interrupt service routine must finish with a 'Return' command: after that point the interrupt routine can be called again.

The example program shown in **Listing 4** initializes Timer0 with a prescale ratio of 64, which means it increments at 250 kHz (see **Figure 4**). If we did not take any further steps the timer would overflow on every 256th clock, as the counter is eight bits wide. To arrange for an overflow every 250 clocks, the first thing the interrupt service routine does is load the counter with the value 6. The result is that the routine is called exactly every millisecond. The word variable 'Ticks' is incremented by one on each timer overflow. When it reaches 1000 the variable 'Seconds' is incremented. The variables 'Seconds' and 'Ticks' can be read from the main program code. In this example the program outputs the number of seconds since it was started, both to the LCD and to the terminal.

```
'------------------------------------
'UNO_Timer4.BAS
'Timer0-Interrupt, Seconds
'------------------------------------
...

Dim Ticks As Word
Dim Seconds As Word
Dim Seconds_old As Word

Config Timer0 = Timer , Prescale = 64
On Ovf0 Tim0_isr
Enable Timer0
Enable Interrupts

Do
  If Seconds <> Seconds_old Then
    Print Seconds
    Seconds_old = Seconds
    Locate 1 , 1
    Lcd Seconds
  End If
Loop

Tim0_isr:
  '1000 µs
  Timer0 = 6
  Ticks = Ticks + 1
  If Ticks = 1000 Then
    Ticks = 0
    Seconds = Seconds + 1
  End If
Return

End
```

Listing 4. Counting seconds exactly using an interrupt.

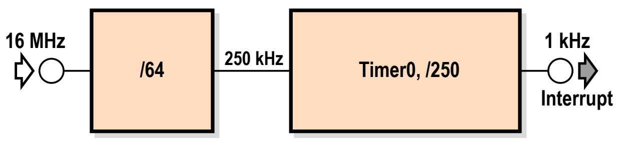


Figure 4. Producing 1 kHz from 16 MHz.

To ensure that the interrupt service routine is actually called, the corresponding interrupt (the Timer0 overflow interrupt) must be enabled. This is done with the line 'Enable Timer0'. Interrupts must also be globally enabled, using the command 'Enable Interrupts'. The complementary command 'Disable Interrupts' prevents all interrupts from occurring.

Averaging analog readings

Analog readings often have mains hum, at 50 Hz or 60 Hz depending on the local frequency, superimposed on them. One way to try to remove this is to take the average of a number of consecutive samples: see **Listing 5**. If readings are averaged over an integer number of mains cycles (that is, over a multiple of 20 ms or 16.667 ms respectively), the effect is to create a null at that frequency. In other words, the hum will be significantly attenuated.

```
'------------------------------------------------
' UNO_Timer5.BAS  Timer1-Interrupt, ADC0 average
'------------------------------------------------
...

Dim Ticks As Word
Dim Ad As Word
Dim Ad0 As Long
Dim Ad0_mean As Long

Config Adc = Single , Prescaler = 32 , Reference = Internal
Config Portb.2 = Output

Config Timer2 = Timer , Prescale = 64
On Ovf2 Tim2_isr
Enable Timer2
Enable Interrupts

Do
  Disable Interrupts
  Ad0_mean = Ad0_mean * 2443       'AC
  'Ad0_mean = Ad0_mean * 1100      'DC
  Ad0_mean = Ad0_mean / 1023
  Ad0_mean = Ad0_mean / 500
  Print Ad0_mean
  Locate 1 , 1
  Lcd Ad0_mean
  Lcd " mV     "
  Enable Interrupts
  Waitms 1000
Loop

Tim2_isr:
  '800 µs
  Timer2 = 56
  Portb.2 = 1
  Ticks = Ticks + 1
  Ad = Getadc(0)
  Ad0 = Ad0 + Ad
  If Ticks >= 500 Then
    Ticks = 0
    Ad0_mean = Ad0
    Ad0 = 0
  End If
  Portb.2 = 0
Return

End
```

Listing 5. Averaging within a timer interrupt.

Again in this example we use a timer interrupt to ensure accurate timings. We will take the average of 500 consecutive readings from ADC0 (see **Figure 5**), which a total of 400 ms. Now this is exactly 20 periods at 50 Hz and 24 periods at 60 Hz and so in either case we are averaging over an integer number of mains cycles. The period between conversions is 800 µs, and to generate this timing we use Timer2, again with a prescale ratio of 64. Each time Timer2 overflows it is reloaded with the value 56, and so the next overflow occurs exactly 200 clocks later.

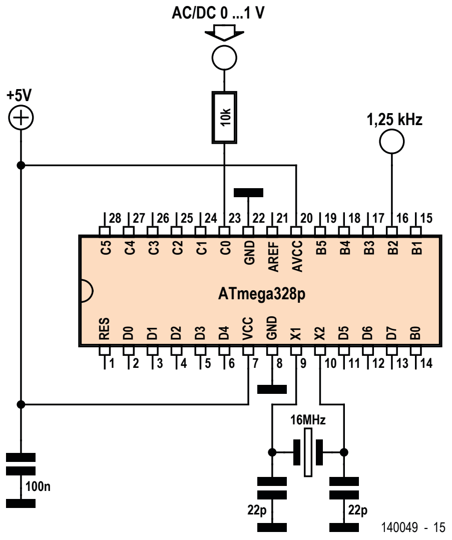


Figure 5. AC and DC voltage measurement.

Eight hundred microseconds is long enough to carry out one conversion (or even several) and accumulate the results. The calls to the interrupt routine and hence the individual readings are counted in the variable 'Ticks', and after 500 readings have been accumulated the sum in variable 'AD0' is copied to the variable 'AD0_mean'. The foreground code can read this variable and send the result to the terminal.

It is sound practice to use an oscilloscope to check that the interrupt routine is running as expected. Is it really being executed every 800 µs, that is, at 1.25 kHz? How long does it take to service the interrupt? A simple trick helps to see what is going on: at the start of the interrupt service routine set a port pin high, and at the end take it low again. In this example we use port PB2, which is connected to LED 2. The oscilloscope does indeed show a pulse every 800 µs lasting about 60 µs, and so there is nothing to worry about. In other cases, however, it can happen that so much is packed into the interrupt service routine that the main program never gets executed, and it is not always obvious what is happening. Furthermore, when interrupts are used, the timing of the main program is no longer so easily predictable; a good rule of thumb is to ensure that no more the 50 % of the microcontroller's time is spent in interrupt service routines.

The averaging process is so good at removing the mains hum component of the signal that it can be used to measure AC voltages using half-wave rectification. The A/D converter already performs half-wave rectification, in that it only measures positive voltages and all negative voltages read as zero. If an AC voltage is applied to the A/D converter input via a 10 kΩ series protection resistor then the converter will only see the positive half-cycles. The program will then average these readings, with the result that, in the case of a square wave input, the average of these half cycles is half the DC voltage with the same effective (RMS) value as the input. In the case of a sinusoidal input there is a further factor of pi/2 = 1.571: in other words, the calculated average is 90.03 % of the true RMS value. These factors can be taken into account in calculating the displayed reading in millivolts. For pure DC measurements the fact that the reference voltage is 1100 mV means the correction factor is 1100. For AC voltage measurements the factor should be 2443, and readings will be correct up to a peak input voltage of 1.1 V. The procedure also works at other frequencies, and in fact voltages at any frequency from 50 Hz to 50 kHz can be measured, meaning that the set-up can be used as a wide-bandwidth millivoltmeter in audio and other applications.

Frequency measurement

In the examples we have looked at so far the timer has received its clock pulses from within the processor, either directly from the processor's clock or via a divider. However, it is possible to clock the timer using an external signal, in which case the timer behaves as a pulse counter. Timer1 in an ATmega328 clocked at 16 MHz can reliably count external pulses at a frequency of up to 4 MHz. Unfortunately, the input lies on port pin PD5 (see **Figure 6**) and so we cannot use the LCD module as well. We could alternatively use an external LCD module controlled over a serial interface, but for now we will simply send our results to a terminal emulator running on a PC.

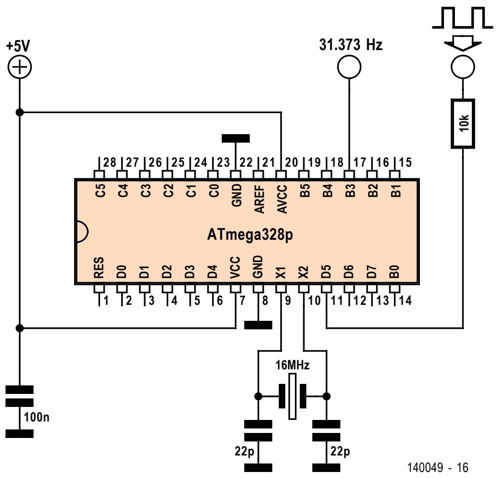

+5V
31.373 Hz
10k
ATmega328p
C5 C4 C3 C2 C1 C0 GND AREF AVCC B5 B4 B3 B2 B1
RES D0 D1 D2 D3 D4 VCC GND X1 X2 D5 D6 D7 B0
16MHz
100n
22p
22p
140049 - 16

Figure 6. Frequency meter with test output.

To use Timer1 as part of a frequency counter (see **Listing 6**) we also need to measure the gate time accurately. To do this we use a second timer and two interrupts. So that we can measure frequencies above 65535 Hz, the interrupt service routine Tim1_isr is called whenever it overflows to increment the variable 'Highword'.

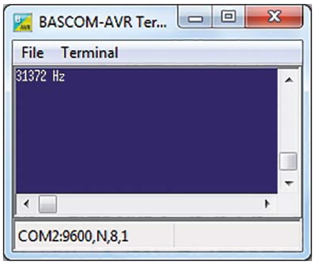
```
'-------------------------------------
'UNO_Timer6.BAS  Frequency 0...4 MHz
'-------------------------------------
...

Dim Lowword As Word
Dim Highword As Word
Dim Ticks As Word
Dim Freq As Long

Config Timer0 = Timer , Prescale = 64
On Ovf0 Tim0_isr
Enable Timer0

Config Timer1 = Counter , Edge = Falling , Prescale = 1
On Ovf1 Tim1_isr
Enable Timer1

Config Timer2 = Pwm , Prescale = 1 , Compare A Pwm = Clear Up
Pwm2a = 128        'B3: 31373 Hz
Enable Interrupts

Do
  Print Freq;
  Print " Hz        ";
  Print Chr(13);
  Waitms 1000
Loop

Tim0_isr:
  '1000 µs
  Timer0 = 6
  Ticks = Ticks + 1
  If Ticks = 1 Then
    Timer1 = 0
    Highword = 0
  End If
  If Ticks = 1001 Then
    Lowword = Timer1
    Freq = Highword * 65536
    Freq = Freq + Lowword
    Ticks = 0
  End If
Return

Tim1_isr:
  Highword = Highword + 1
Return
End
```

Listing 6. Frequency measurement up to 4 MHz.

Timer0 is responsible for providing a gate time of exactly one second. When the variable 'Ticks' is equal to one Timer1 is reset and counting starts. Exactly 1000 ms later the current counter value in Timer1 is read into the variable 'Lowword' and then the frequency is calculated from this and the value in 'Highword'. The foreground code can now output the result as a frequency in Hertz.

If we configure Timer1 as an ordinary timer with a 16 MHz clock (using the command 'Config Timer1 = Timer , Prescale = 1') then we should see a reported frequency of exactly 16000000 Hz. With the timer configured as a counter, however, the maximum increment rate is limited to a quarter of the processor clock, because the state of the input pin is only sampled at a limited rate. If we try an input frequency of 6 MHz we find that approximately every other pulse is lost, giving a reading of around 3 MHz. At frequencies up to a little over 4 MHz, however, the counter is very accurate.

Observe one special aspect of the 'Print' commands. Normally Bascom terminates each print command by emitting Chr(13) and Chr(10), which makes it easy to send the output of one Bascom program to the input of another. However, in the receive direction only the character Chr(13) is expected, which means that we can suppress the line feed character (the Chr(10)) and send just the carriage return (the Chr(13)). To display the results we can use Bascom's own terminal emulator: the effect is that each new reading overwrites the old on the same line rather than beginning a new line for each: see **Figure 7**.



```
31372 Hz
```

COM2:9600,N,8,1

Figure 7. The frequency displayed using the terminal emulator.

External display

A simple solution to the problem of not being able to use the display directly is to use another Arduino. One, with an Elektor shield fitted, functions as the display module, while the other acts as the frequency counter, sending its results over a serial interface to the first. **Listing 7** shows the code for a simple display with scrolling output. The last two lines are always shown.

```
'-----------------------------------
'UNO_Display.BAS  COM input B0
'----------------------------------
...

Dim Text1 As String * 16
Dim Text2 As String * 16

Open "comb.0:9600,8,n,1" For Input As #2
'Software COM input at B0

Do
  'Input Text1
  Input #2 , Text1
  Locate 1 , 1
  Lcd Text2
  Text2 = Text1 + "        "
  Locate 2 , 1
  Lcd Text2
Loop

End
```

Listing 7. Text output on the LCD.

The program offers two possibilities for receiving serial data. The command ‘Input Text1’ (commented out) uses the normal serial RX input on D0. This input is connected to the USB interface on the Uno board via a 1 kΩ series resistor. This signal can be overridden using a low-impedance drive, as for example happens when the RX pin on the display unit is connected directly to the TX pin of the transmitting Uno (the one carrying out the frequency measurement). The disadvantage is that communication during the next program upload using the bootloader will be disturbed. It is easy to forget this, which can lead to a lot of head-scratching when you next make a change to the program!

The second possibility takes advantage of Bascom’s ability to implement a serial port in software using any desired port pin. Here we have chosen PB0 (Arduino pin 9: see **Figure 8**) and use the command ‘Input #2, Text1’. The text is received just as reliably, and there is no interference with program upload.

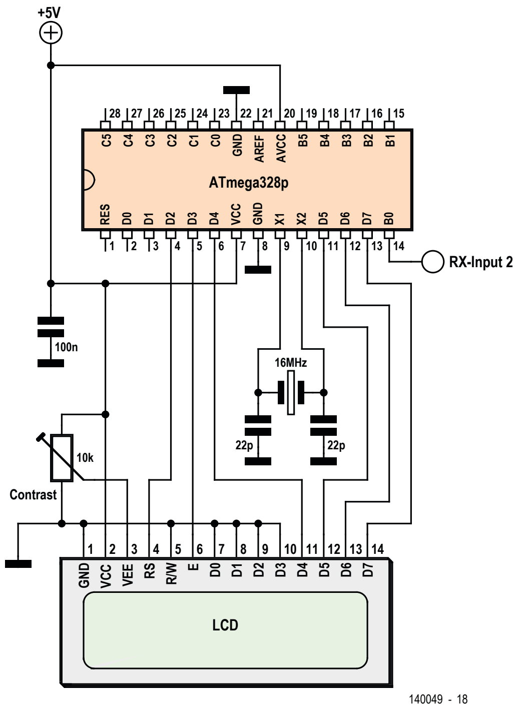


Figure 8. The LCD terminal.

Web Links

[1] Roger Leifert, 'DCF Tester', Elektor June 2014, www.elektor-magazine.com/130571

[2] Martin Ossmann, 'AVR Software Defined Radio', Elektor April 2014, www.elektor-magazine.com/100181

[3] www.elektormagazine.com/magazine/elektor-201409/27024

Chapter 44 • Microcontroller BootCamp (6)

The SPI interface

Serial communication with each unit of information following the previous one is actually the normal situation. That's how we talk to each other in person or by phone, read and write text—or in the case of Retronics: send telegrams.

In many cases all you need is a single line to transmit data. However, adding a clock line makes things more reliable. Let's find out.

On the Serial Peripheral Interface (SPI) bus, the actual data travels bit by bit over one line—for example from a microcontroller to a display, an EEPROM or an SD card. In most cases it's also desirable to be able to read data, so you need a second line to provide a return channel for the data. There's yet another part to the picture: a clock line. The clock signal always clearly indicates when the next bit is available on the data line. That eliminates the need for precise agreement on data timing, which both parties have to supervise with timers. With these three lines, data transmission is fairly bombproof.

Port extension with a shift register

The first thing we want to try out is actually not an SPI interface, but instead something entirely different. Shift registers have been around for a long time (before microcontrollers were even invented) and are good for understanding how serial data transfer works. They can also be put to good use in combination with a microcontroller. That's because port lines are always scarce, especially on Arduino boards. A port extension with a shift register can help ease the scarcity. With a type 4094 8-bit shift register, you need three lines to talk to it and you end up with eight new outputs. You can increase this to 16 by connecting a second shift register, or even 80 if you connect ten shift register ICs in series. If you need a lot of outputs, that's an especially low-cost way to meet the requirement.

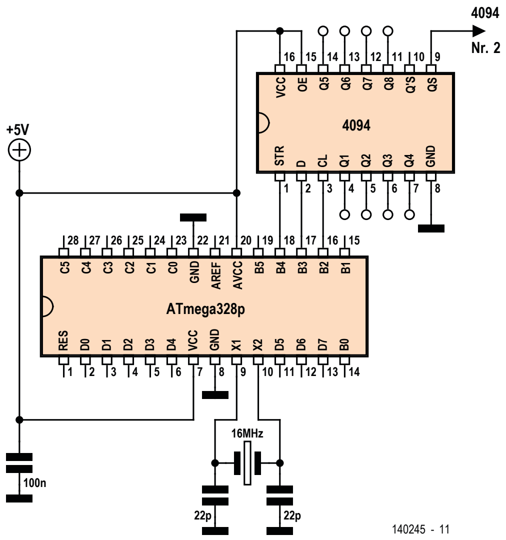


Figure 1. Connecting a type 4094 shift register.

Figure 1 shows the connections to the Uno board. The 8-bit shift register has a clock input (CL) and a data input (D). The data is applied to the D input one bit at a time, starting with the most significant bit, and a rising edge is applied to the clock input CL for each bit. The data is shifted through the individual flip-flops of the register step by step with each clock pulse. There is also the strobe input STR. When a pulse is applied to the strobe input, all the data present in the shift register is transferred to the type-D flip-flops connected to the output pins. You could also tie the strobe input to the supply voltage Vcc, but then all the intermediate results of each shift operation would appear on the outputs. By contrast, if you apply a strobe pulse after all the bits have been shifted in, you only see the final result on the outputs.

The software for all this (Listing 1) is simple. To output a byte D, the code first copies the most significant bit to the bit variable B (`B = D.7`) and puts it on the corresponding port pin. It then generates a positive clock pulse on CL with a length of 1 millisecond. A microsecond would also be sufficient, but the slower output is easier to see on an oscilloscope. After the clock pulse, a shift instruction (`Shift D , Left`) causes all bits of D to be shifted left by one position. This puts what used to be bit 6 on the output. This process is repeated until all eight bits have been shifted out. At the end comes the strobe pulse, and then all eight bits are present at the outputs of the 4094.

```
'-------------------------------------
'UNO_shift.BAS  Shift Register 4094
'-------------------------------------
$regfile = "m328pdef.dat"
$crystal = 16000000
$baud = 9600
```

```
Dim Dat As Byte
Dim D As Byte
Dim N As Byte
Dim B As Bit


Sr Alias Portb.4     '4094 pin 1
Da Alias Portb.3     '4094 pin 2
Cl Alias Portb.2     '4094 pin 3
Config Portb = Output

...

Dat = 0
Do
  Cls
  Lcd Dat
  Lcd "  "
  Print Dat
  D = Dat
  For N = 1 To 8
    B = D.7
    Da = B
    Waitms 1
    Cl = 1
    Waitms 1
    Cl = 0
    Waitms 1
    Shift D , Left
  Next N
  Sr = 1
  Waitms 1
  Sr = 0
  Waitms 1
  Dat = Dat + 1
  Waitms 500
Loop
End
```

Listing 1. Output using a shift register.

The code continuously increments the data byte to be transferred so the outputs of the shift register change while the program is running. The current value is shown on the LCD if the Elektor Extension shield is fitted, and it is output to the terminal emulator. This makes it easy to compare the output levels of the shift register to the digital value being output.

If you need more than eight outputs, you can use the Qs output of the 4094 IC. Each bit that is clocked into the shift register appears at the Qs output eight clock pulses later. The D input of the next shift register can be connected to this output. In this way you can connect as many 4094 ICs in series as desired, with the clock and strobe lines connected to all of them in parallel. Of course, the software will have to be modified accordingly. First it shifts out all of the bits necessary to fill the chain of shift registers (e.g. 16 with two ICs or 80 with ten), and then it outputs the common strobe pulse.

Manual data transmission
Although you only need one line for the data, you also need a clock line when you use a clocked serial interface, as in the above example with a shift register or with the SPI bus. If you compare this with Morse telegraphy, for example, you can see the difference. There both parties have to agree on the transmission rate, and no breaks are allowed within an information unit. For example, if a radio operator wants to send an “X” (dash dot dot dash) and stops in the middle to scratch his head, the two characters “N” (dash dot) and “A” (dot dash) are sent instead. The situation is exactly the same with an asynchronous serial data interface, where both parties have to agree on the baud rate. After the transmission of a byte has started, all of the bits must be sent within a precise time frame. By contrast, with SPI the timing is not critical and any desired delays are allowed. The additional clock signal makes the transfer entirely independent of the speed. No matter whether the data rate is just one bit per minute or a million bits per second, the data will be transferred properly. On an SPI bus there is always an SPI master and an SPI slave. Data can travel in both directions, but the master ways generates the clock signal.

You can try this for yourself manually, where you (as the user) assume the role of master. You can transmit a byte by pushing the two buttons S1 and S2 (**Figure 2**). This is not the usual way of doing things, but it helps you understand exactly how it works. One of the buttons is for the data, and the other is for the clock. Aside from that there’s nothing new you have to learn, since you already know how a byte is put together. Here’s how it works: First you send bit 7. If it is a ‘1’, you press and hold button S2; otherwise you don’t. Then you press button S1 briefly without changing the state of S2. The receiving end (the slave, which in this case is the Arduino board) then knows when it should read the bit from the data line. Now you repeat the process for bit 6, bit 5, and so on until bit 0.

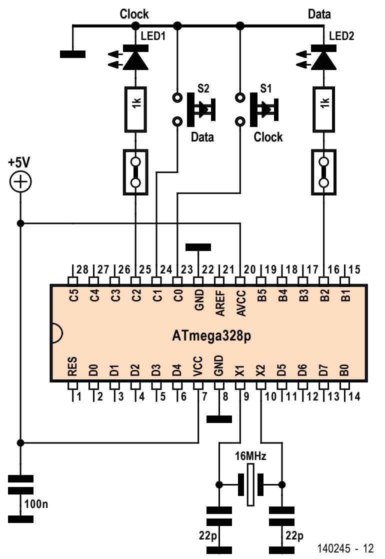


Figure 2. Manual input and output.

```
'--------------------------------------
'UNO_spi1.BAS  Shift in/out
'--------------------------------------
$regfile = "m328pdef.dat"
$crystal = 16000000
$baud = 9600

Dim D As Byte
Dim B As Bit
Dim N As Byte

S1 Alias Pinc.0
Portc.0 = 1
S2 Alias Pinc.1
Portc.1 = 1
Led1 Alias Portc.2
Ddrc.2 = 1
Led2 Alias Portb.2
Ddrb.2 = 1

...

Do
  D = Rnd(255)
  Cls
  Lcd D
  Print D
  Locate 2 , 1
```

```
    For N = 0 To 7
       B = D.7
       Lcd B
       Print B;
       Led2 = B
       Waitms 300
       Led1 = 1
       Waitms 200
       Led1 = 0
       Waitms 500
       Shift D , Left
    Next N
    Led1 = 0
    Led2 = 0
    Waitms 2000
    Cls
    Print
    D = 0
    Do
    Loop Until S1 = 1
    For N = 0 To 7
       Shift D , Left
       Do
       Loop Until S1 = 0
       If S2 = 0 Then B = 1 Else B = 0
       D = D + B
       Lcd B
       Print B;
       Waitms 100
       Do
       Loop Until S1 = 1
       Waitms 100
    Next N
    Print
    Locate 2 , 1
    Lcd D
    Print D
    Waitms 2000
  Loop
  End
```

Listing 2. SPI master and slave (very slow).

The program in **Listing 2** displays the data in both directions. At first the microcontroller is the master and you are the slave. A byte with a random value is sent, with the clock signal indicated by LED1 and the data indicated by LED2. If you watch carefully, you can read the transmitted byte. However, that's not easy, so the byte is also shown on the LCD and sent to the terminal emulator. The individual bits also appear one after the other on the LCD and the terminal emulator screen:

```
        83
        01010011
```

Then the roles change. Now you are the master, and your job is to send exactly the same byte back to the microcontroller. Here you can see that the ability to send data at any desired speed is a big advantage, since you can take all the time you want to decide which bit value to send next. For example, suppose you want to send the decimal number 100. Bit 7 corresponds to decimal 128, which is more than 100, so it is '0'. Next comes bit 6 with a value of 64, so it's a '1', and you're left with 36 still to send. This means that bit 5 (32) is a '1', which leaves 4. The next two bits (bit 4 = 16 and bit 3 = 8) are '0', bit 3 (4) is a '1', and the last two bits are '0'. Now you have sent the binary number 01100100, with each bit marked by a clock pulse. It may not have been all that easy, but the microcontroller had no problem reading the data. You can regard this as a test of your concentration, and if the LCD shows the right result, you pass the test.

If you look closely at **Listing 2**, you will see that the bits are inverted when they are read. That's because pressing the data button yields a zero bit value. This is not especially intuitive, so the result is inverted when the bit is read to make things easier for you. Another interesting aspect is using the instruction **D = Rnd (255)** to generate a pseudo-random number. In fact, this always generates the same sequence of numbers, but the Bascom Help gives some suggestions for what you can do about this.

From microcontroller to microcontroller

In this example, data is sent over the SPI bus from one microcontroller to another. The data is this case consists of 10-bit readings from the A/D converter. This shows another advantage of SPI, which is that the data width is not fixed. No matter whether you send 8, 10, 12 or 16 bits, the procedure is always the same. If the only objective were to connect two microcontrollers together, it would actually be less effort to use an asynchronous serial interface with the TXD and RXD lines. The SPI bus, by contrast, is better for controlling and communicating with external hardware. Here the main purpose of the exercise is to illustrate the transmission protocol.

As previously with the 4094 shift register, a third line is involved here—in this case the chip select line /CS. The slash (/) means that the signal on this line is active Low. The chip select line allows you to connect several slave devices to a single master. In that case they share the data and clock lines, but each one has its own chip select line. When that line is low, the corresponding slave knows that it is selected. There's also another benefit from using a chip select line. If there is any delay in enabling the slave, there may be some confusion about which bits have already been transferred. However, if the slave waits until it sees a falling edge on its CS input (high to low signal transition), it knows that the transfer is starting. And if a noise pulse is read as a clock signal, the rest of the data for that transfer is trash, but on the next access everything is again as it should be.

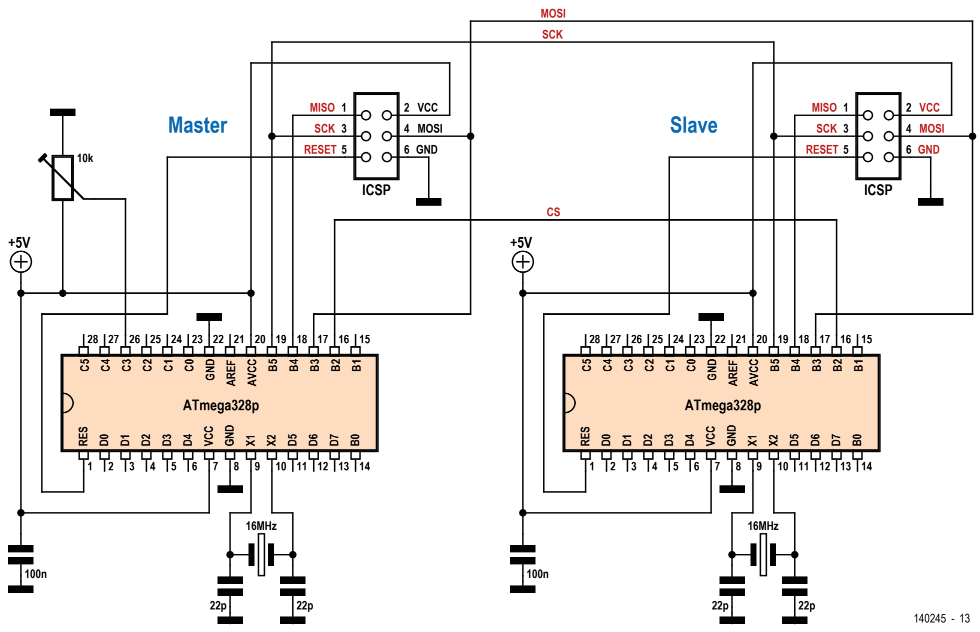


Figure 3. An SPI connection between two microcontrollers.

The ATmega328 also uses the SPI bus for program download from an external programming device. The following lines are therefore available on the six-pin programming connector on the Arduino board and on the Elektor Extension shield (ICSP in **Figure 3**): the clock line Serial Clock (SCK) on B5, the write data line Master Out Slave In (MOSI) on B3, and the read data line Master In Slave Out (MISO) on B4. There is no chip select line, but the Reset line has the same effect because programming takes place with the Reset line pulled low. Now we want to use these lines exactly as intended. This has the advantage that we can use the hardware SPI unit of the microcontroller, if it has one. With hardware SPI we do not have use program code to put each bit individually on the data line as in the previous examples, and everything is a lot faster. However, we still need a chip select line, and in this case we use the B2 line for this purpose.

The master uses the MOSI line as the output and generates the clock and chip select signals (**Listing 3**). The process is slowed down a bit by three 1-millisecond delays so that all the signals can easily be seen on the oscilloscope. Besides, we don't want to make things too difficult for the slave. If you wish, you can test the boundaries by reducing the delays until transmission errors start to occur.

```
'------------------------------
'UNO_spi2.BAS  SPI Master
'------------------------------
$regfile = "m328pdef.dat"
$crystal = 16000000
```

```
$baud = 9600
Dim B As Bit
Dim Dout As Word
Dim N As Byte
Dim I As Byte

Sck Alias Portb.5
Ddrb.5 = 1
Mosi Alias Portb.3
Ddrb.3 = 1
Cs Alias Portb.2
Ddrb.2 = 1

Cs = 1
Mosi = 0
Sck = 0

Config Adc = Single , Prescaler = 32 ,
Reference = Avcc
Start Adc

Cls
Cursor Off

Waitms 200
Do
  Dout = Getadc(3)    'Pot
  Locate 1 , 1
  Lcd Dout
  Lcd "    "
  Cs = 0
  Waitms 20
  For N = 1 To 10
    Mosi = Dout.9
    Waitms 1
    Sck = 1
    Waitms 1
    Sck = 0
    Waitms 1
    Shift Dout , Left
  Next N
  Cs = 1
  Waitms 100
Loop
End
```

Listing 3. SPI master.

The three lines are inputs for the slave device (**Listing 4**). It constantly waits for specific signal edges on the /CS and SCK lines and then reads in a bit from the MOSI line. Since everything is handled by software here, the code must wait for each edge in a Do loop. This takes a bit of time, so data transmission must be slower than with a hardware SPI implementation. The received data is shown on the display and on the terminal emulator. When you turn the potentiometer on the master board, the change is visible on the slave.

```
'-----------------------------
'UNO_spi3.BAS  SPI Slave
'-----------------------------
$regfile = "m328pdef.dat"
$crystal = 16000000
$baud = 9600

Dim Addr As Byte
Dim B As Bit
Dim Dout As Word
Dim Din As Word
Dim N As Byte
Dim I As Byte


S1 Alias Pinc.0
Portc.0 = 1
S2 Alias Pinc.1
Portc.1 = 1
Sck Alias Pinb.5
Portb.5 = 1
Mosi Alias Pinb.3
Portb.3 = 1
Cs Alias Pinb.2
Portb.2 = 1
...

Do
  Do
  Loop Until Cs = 0
  Din = 0
  For N = 1 To 10
    Shift Din , Left
    Do
    Loop Until Sck = 1
    Din = Din + Mosi
    Do
    Loop Until Sck = 0
  Next N
```

```
    Do
    Loop Until Cs = 1
    Locate 1 , 1
    Lcd Din
    Lcd "    "
    Print Din
  Loop
  End
```

Listing 4. SPI slave.

SPI EEPROM 25LC512

There is a wide range of ICs available with an SPI interface, including A/D converters, memory devices and display drivers. Serial EEPROMs from Microchip and other companies are available at low cost and are widely used. The 25LC512 (not to be confused with the 24C512, which has a I²C bus interface) has a capacity of 64 KB, and it is a good solution when the 1-kilobyte capacity of the ATmega328's internal EEPROM is not sufficient. I²C EEPROMs are more widely used in the hobby realm, but SPI types are generally preferred in the professional realm because they offer especially high operational reliability.

Figure 4 shows the connections to the Uno board. The pins of the original SPI interface (2x3 pin header) are again used here. That makes it easy to build a convenient plug-in memory module by fitting the IC in a socket soldered to a small 6-pin socket header. You only have to connect one additional line. This is the chip select line, which is again assigned to B2 because the Reset line present on the connector cannot be used for this purpose.

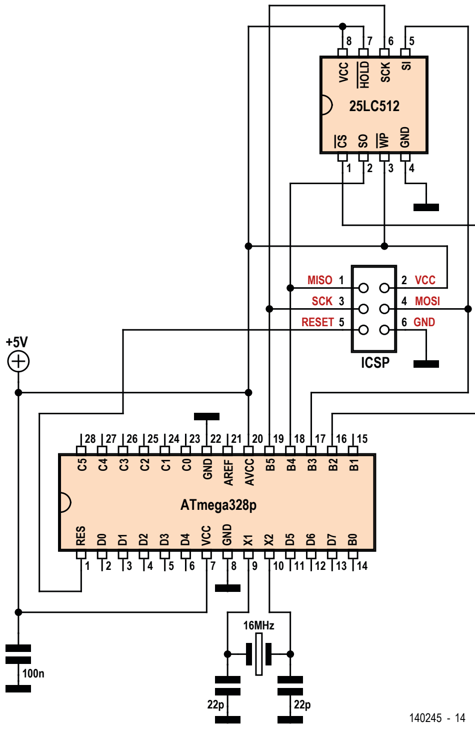


Figure 4. Connecting a serial EEPROM.

Here there are two data lines. MOSI (Master Out Slave In) is the data output line and is connected to the Serial Input (SI) pin of the EEPROM, while MISO (Master In Slave Out) is used to read data from the Serial Output (SO) pin. The microcontroller is always the master, and it generates the clock on the SCK line. To go with this exercise, we have written a subroutine (subroutines are called “Sub” in Bascom; see the **inset**) that transfers data in both directions in a single go (**Listing 5**).

```
Sub Spioutin
  Din = 0
  For N = 0 To 7
    Shift Din , Left
    Din = Din + Miso
    If Dout.7 = 1 Then Mosi = 1 Else Mosi = 0
    Waitus 3
    Sck = 1
    Waitus 2
    Sck = 0
    Waitus 2
    Shift Dout , Left
  Next N
End Sub
```

Listing 5. Reading and writing data over MOSI and MISO.

Before the subroutine is called, the data to be sent must be placed in the global variable **`Dout`**, and after the call the received data is located in the variable **`Din`**. Of course there are situations in which data is only written, but in that case zero bits are usually sent in the other direction.

The relatively complex data sheet for the 25LC512 tells you what has to be sent to the device, as well as when and how. After the chip select line has been pulled low, the memory chip first receives a simple byte command that specifies what action is to be performed. To read data from the memory, you have to send a '3' command followed by two address bytes forming the high-byte and low-byte portions of the address. After that as many data bytes as desired can be read out with automatic address incrementing (see **Listing 6**).

```
 Cs = 0
 Dout = 3          'read
 Spioutin
 Dout = 0          'A8...A15
 Spioutin
 Dout = 0          'A0...A7
 Spioutin
 I = 0
 Do
   Locate 1 , 1
   Lcd I
   Print I;
   Print "  ";
   I = I + 1
   Spioutin
   Locate 2 , 1
   Lcd Din
   Print Din
   Waitms 200
Loop
```

Listing 6. EEPROM readout (excerpt).

The program displays the sequential addresses and the data bytes that are read out. A brand-new or fully erased EEPROM always delivers only the value 255. Now let's try to program some data. For that we use the byte command '2'. However, a bit of preparation is necessary first. Writing must be enabled by sending the command '6' (**Listing 7**). To check whether writing is enabled you can read the EEPROM status register, which requires sending the command '5'. Each action is only effective if you pull /CS low at the start and then return it to the high level at the end. If the value in the status register is 2, the IC is enabled for write operations. Complicated? Yes, but that makes it especially foolproof.

```
Cs = 0
Dout = 6          'write enable
Spioutin
Cs = 1
Waitms 20
Cs = 0
Dout = 5          'read status
Spioutin
Spioutin
Cs = 1
Locate 1 , 1      'status
Lcd Din
Print Din
```

Listing 7: Enabling write (excerpt).

Now we are allowed to write data to the memory, but even then there's something we have to consider. The memory space is divided into pages, which in the case of the 25LC512 have a size of 128 bytes. Each transfer is limited to a maximum of 128 bytes of data, or as many bytes as it takes to reach the next page boundary. After this you switch /CS high and then give the EEPROM enough time to actually store the data. According to the data sheet, 5 ms is enough time for storing the data. If you exceed the page boundary (I tried it), the result is chaos. Then the data you find in memory is totally different from what you wanted to write to memory. For this reason, the example program carefully obeys the rules and writes 128 bytes to the first page from address 0 to address 127, with the data values in ascending order from 0 to 127. When the data is read back, that's exactly what you see.

As usual, the entire program code (**`UNO_spiEE1.bas`**) can be downloaded from the Elektor website [1]. It performs the following actions in sequence:

- Command 6, write enable
- Command 5, read status register; display for 1 second
- Command 2, write 128 bytes starting at address 0
- Command 3, read memory starting at address 0; endless loop

Data logger

One practical application for the 64-KB memory is a data logger. The objective here is to acquire measurement data from the ADC4 analog input once per second and store the data. The memory will be full in approximately 18 hours.

You don't always have to program everything yourself, since Bascom has a lot of ready-made functions for many situations. In this case you have the option of configuring an SPI interface as a software interface using any desired port pins or as a hardware interface using the microcontroller pins designated for this purpose. The hardware SPI is especially fast and is commonly used for tasks such as driving graphical displays. However, this involves a whole lot of parameters that must be configured for each specific application, which requires a detailed study of the ATmega328 data sheet. Things are a bit easier with the software SPI function, and it provides a reasonably high transmission rate. Although writing your own SPI procedure from the ground up is not a bad idea because it allows you to implement the timing diagrams in the data sheets very clearly, the ready-made software SPI is more convenient and faster, which is why we use it here.

The interface configuration specifies which lines are to be used. For Din, Dout and Clock we use the familiar MISO, MOSI and SCK lines, which are already available on the ICSP connector. The SS line corresponds to the /CS line. In this case this line should not be operated automatically by Bascom because it is usually necessary to transfer a lot of data during a single active chip select phase. Consequently, this line (on port pin B2) will still be operated “manually”. The **`Mode = 0`** setting is also important, because there are four different SPI modes.

The program excerpt in **Listing 8** shows how the software SPI is used to read data from the serial EEPROM. The instruction **`Spiout Dout , 1`** sends exactly one byte, which is transferred in the variable **`Dout`**. In the other direction, the instruction **`Spiin Din , 1`** reads one byte, which is then available in the variable **`Din`**. The entire program reads all the data from the EEPROM and shows the contents on the display and on the terminal emulator screen.

```
Config Spi = Soft , Din = Pinb.4 , Dout = Portb.3 ,
Ss = None , Clock = Portb.5 , Mode = 0
Spiinit

Cs Alias Portb.2
Ddrb.2 = 1
Cs = 1

     Cs = 0
     Dout = 4          'write disable
     Spiout Dout , 1
     Cs = 1
     Waitms 1
     Cs = 0
     Dout = 3          'read
     Spiout Dout , 1
     Dout = 0          'A8...A15
     Spiout Dout , 1
     Dout = 0          'A0...A7
     Spiout Dout , 1
     I = 0
     Do
       Locate 1 , 1
       Lcd I
       Lcd "  "
       Spiin Din , 1
       Locate 2 , 1
```

```
    Lcd Din
    Lcd "  "
    Print I ;
    Print "  ";
    Print Din
    Waitms 200
    I = I + 1
    If I >= 65535 Then Exit Do
  Loop
```

Listing 8. Using the software SPI (excerpt).

As usual, the entire program code (**`UNO_spiLogger.bas`**) can be downloaded from the Elektor website [1]. It is too large to be listed fully here. Pressing S1 starts a measurement run. It can be stopped at any time by pressing S2, after which the stored data can be read out.

A timer interrupt routine (excerpt in **Listing 9**) is used to control the timing during data acquisition. The voltage on ADC4 is measured and stored once per second. The 128-byte block size of the EEPROM is taken into account. At the start of each block, a write access is started and the current address is transferred, followed by 128 bytes of data. At the end of the block, the /CS line is pulled high to allow the EEPROM to store the entire block. Since the /CS line is connected to port pin PB2, LED2 on the shield is lit when the line is high. The LED therefore flashes each time a block of data has been transferred to memory.

```
Tim0_isr:
  '4000 µs
  Timer0 = 6
  Ticks = Ticks + 1
  If Ticks = 250 Then
    Ticks = 0
    U = Getadc(4)
    U = U / 4
    Addr = Seconds
    Addr = Addr And 127
    If Addr = 0 Then             'start of page
      Cs = 0
      Dout = 6                   'write enable
      Spiout Dout , 1
      Cs = 1
      Waitus 100
      Cs = 0
      Dout = 2                   'write
      Spiout Dout , 1
      Waitus 100
      Cs = 0
      Dout = 3
      Addr = High(seconds)
```

```
          Dout = Addr              'A8...A15
          Spiout Dout , 1
          Addr = Low(seconds)
          Dout = Addr              'A0...A7
          Spiout Dout , 1
        End If
        Dout = U
        Spiout Dout , 1
        Addr = Seconds
        Addr = Addr And 127
        If Addr = 127 Then         'End of page
          Cs = 1
        End If
        Print Seconds
        Seconds = Seconds + 1
        If Seconds = 0 Then Seconds = 65535
   End If
  Return
```

Listing 9. Data storage in the timer interrupt routine (excerpt).

Subroutines

Subroutines are called "Sub" in Bascom, and they are used to allow a block of code to be called repeatedly from different locations in a program. To make this possible, each subroutine must be declared at the start of the program. This way the name of the subroutine is known to the compiler, so the it can be called using this name in the same way as Bascom functions.

```
Declare Sub Spioutin()
…
Dout = 6
Spioutin
…

Sub Spioutin
  Din = 0
…
  Shift Dout , Left
End Sub
```

You always have to be very careful with the variables used by a subroutine. In the case of the Spioutin subroutine, all of the variables it uses are "global" variables, which are dimensioned at the start of the main routine and are therefore valid in the entire program. However, you could do things differently and transfer the data to the subroutine when it is called:

```
Declare Sub Spioutin (Byteout as Byte)
```

In that case the variable Byteout would not be valid globally, but only within the subroutine. The subroutine call would then take the form:

```
        Spioutin 6
```

This saves one line compared to the previously shown call.

You can also transfer a group of several variables to a subroutine in a subroutine call, as can be seen from the example of the Bascom Spiout subroutine:

```
        Spiout Dout , 1
```

For novice programmers, it's generally safer to use only global variables in your own subroutines. For advanced programmers, on the other hand, selecting the optimal form of data transfer is a major consideration.

Tip for using the Arduino programmer in Bascom

Many readers who are using the original Arduino boot loader have encountered the following problem: if a program performs serial output, the Arduino programmer in Bascom does not work properly the next time and hangs. Some readers have discovered that this problem can be resolved by using the Arduino IDE before using the programmer again. Simply transferring the Blink program, for example, is sufficient.

However, there's an easier solution. After you launch the programmer in Bascom, briefly press the Reset button on the Uno board three times (or more) at roughly 1-second intervals. After this the programming function will work again, even when the previous program included a Print output.

If you use the MCS boot loader on the Uno board, this problem does not occur. Another alternative is to use an external programmer.

Web Link

[1] www.elektor-magazine.com/140245

Chapter 45 • Microcontroller BootCamp (7)

The I²C-Bus

If you are running short of I/O lines on an Arduino Uno board, a remedy is available. The I²C bus needs only two port pins and can address up to 127 external ICs. There are countless devices available with I²C interfaces, including simple port expanders, EEPROMs and a wide variety of sensors. In the final instalment of our Bascom course series, we show you how the I²C bus works. As usual, the main focus is on interesting demo applications.

The Inter-IC bus or I²C bus (barring code, also sloppily written as I2C) is a two-wire data bus consisting of a data line and a clock line, originally developed by Philips (not: Phillips) for consumer electronics applications. Most television sets or video recorders have a central processor that controls a large number of modules. With a data bus consisting of just two lines, connecting all these modules to the processor is easy. The processor is the bus master, as with the SPI bus, and the peripheral devices are the slaves. A particular feature of the I²C bus is that every slave device has a 7-bit address. That allows a large number of ICs to be connected to the bus without interfering with each other. Along with RAMs, EEPROMs, port expanders, real-time clocks, A/D and D/A converters, there are a large number of special-purpose I²C devices such as display drivers, PLL ICs and many others. An excellent reference book on I²C is available from Elektor, see **Further Reading** [a].

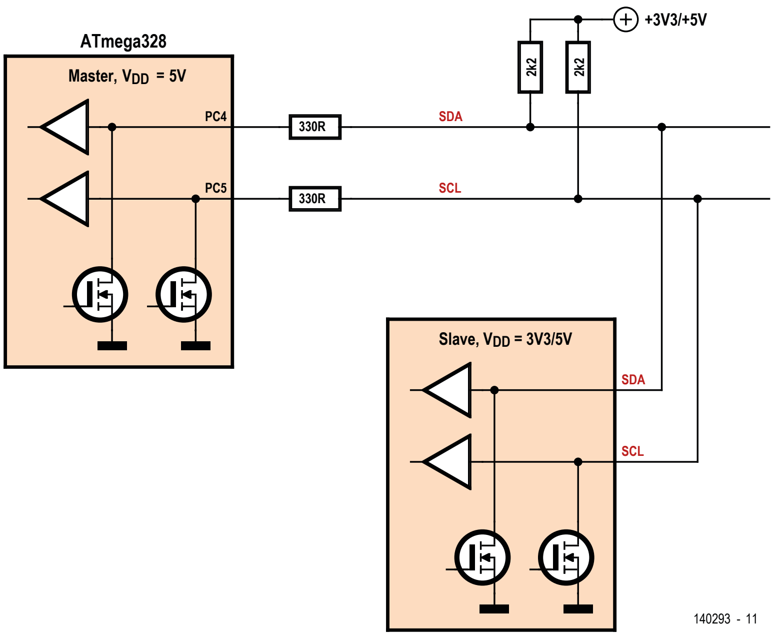


Figure 1. I²C bus connections between master and slave devices.

Data transfer and addressing

The I²C bus consists of a serial data line (SDA) and a clock line (SCL). One bit per clock pulse (as in a shift register) is transferred on the data line. Usually the address bits are sent first, followed by the data bits. Each line has a pull-up resistor, and each line can be pulled low by any device on the bus. **Figure 1** shows the basic bus architecture. The master generates the clock signal. The data can come from the master or from the slave.

The I²C bus can work with 5 V microcontrollers and ICs as well as 3.3 V devices. You can even connect both types to the same bus. The two pull-up resistors, which typically have a value of 2.2 kΩ, hold the bus lines at 3.3 V or 5 V (logic High level) when the lines are not pulled low by any of the pull-down transistors in the devices connected to the bus. Any 5 V devices on the bus also see 3.3 V as a High level because it is significantly higher than 2.5 V (half of the supply voltage), and of course 0 V is logic Low in any system. This means that you can easily connect the Arduino Uno board to a 3.3 V slave device. That's handy because many recent ICs are only designed to operate at 3.3 V.

The ATmega328 on the Uno board has an integrated hardware I²C interface connected to pins PC4 and PC5. However, Bascom also has special commands that can be used to implement a software I²C interface using any desired port pins, and of course you can also write your own functions to set the lines High or Low using individual code lines. Here we only use the Bascom software I²C interface, but with the same port pins as used by the hardware I²C interface integrated in the microcontroller.

These port pins are also used on the Elektor Extension shield [1] for the Arduino Uno. There they are routed to the EEC/Gnublin connector K2, which can be used to connect Gnublin modules with an I²C interface over a flat cable, such as a module with eight relays [2]. The bus lines also have 330 Ω series resistors, which can be omitted if desired. However, they provide protection against false signals resulting from reflections on long bus lines, and they can help avoid problems that may occur on buses with devices operating at different supply voltages. For example, the input currents of 3.3 V peripheral devices could exceed the maximum rated value if one of the bus lines is accidentally set to a 5 V high level for a prolonged period. The series resistors limit the input current to a safe level.

The I²C bus protocol defines several specific states that allow every device on the bus to de tect the start and end of a transfer:

- Quiescent state: SDA and SCL are high and therefore inactive. The Bascom instruction **`I2cinit`** puts both lines in the quiescent state but without internal pull-up resistors, since they are located externally.

- Start condition SDA is pulled low by the master while SCL remains high (Bascom instruction: **`I2cstart`**).

- Stop condition: SDA changes from low to high while SCL remains high (Bascom instruction: **`I2cstop`**).

- Data transfer: The current sender places eight data bits on the data line SDA, which are shifted out by clock pulses on the clock line SCL generated by the master. The transfer starts with the most significant bit (Bascom instruction: **`I2cwbyte Data`**).

- Acknowledge (Ack): The currently addressed receiver acknowledges reception of a byte by holding the SDA line low until the master has generated a new clock pulse on the SCL line. This acknowledgement also means that another byte is expected to be received. If the device wishes to end the transfer, it must indicate this by omitting the acknowledgement (Nack). The transfer is terminated by the stop condition (Bascom instruction: **`i2crbyte Data, Ack`** or **`i2crbyte Data, Nack`**).

Addresses are transferred and acknowledged in the same way as data. In the simplest case of a data transfer from the master to a slave, such as an output port, the following procedure is used. The master generates the start condition and then transfers the address of the port IC in bits 1 to 7 and the desired direction of the data transfer in bit 0 – in this case, 0 for writing to the device. The address is acknowledged by the addressed slave. Then the master sends the data byte, which is also acknowledged. The connection can be ended now by generating the stop condition, or another byte can be sent to the same slave.

I²C bus address with data direction							
A6	A5	A4	A3	A2	A1	A0	R/W

f the master want to read data from a slave, the address must be sent with the data direction bit set to 1. The master then generates eight clock pulses and receives eight data bits. If reception is confirmed by an acknowledgement on the ninth clock pulse, it can receive another data byte. At the end the master terminate the transfer with a stop condition when no acknowledgement is received.

Every I²C device has a fixed address. Part of the address is specific to the device type, and the rest can be configured by the user with the address lines A0, A1, etc. fed out from the device. These address lines are tied high or low in the circuit to set the address bits. If the device has three address lines fed out, such as the PCF8574 port expander, up to eight different addresses can be set. This means that up to eight devices of the same type can be connected to one bus. This port expander provides eight digital outputs, and the signal levels on the outputs are determined by the eight bits in the data byte sent to the IC. The base address of the port expander is 40_{hex} (decimal 64). The base addresses of some typical first-generation Philips I²C devices are:

• PCF8591 A/D converter:	90_{hex} (decimal 144)
• RAMs and EEPROMs:	$A0_{hex}$ (decimal 160)
• PCF8583 real-time clock:	$A0_{hex}$ (decimal 160)

Incidentally, there are two different notations for the address, which sometimes cause confusion and laborious troubleshooting. Some data sheets only give the 7-bit address without the read/write bit. In that notation the base address of the PCF8574 would be 20_{hex} (decimal 32) By contrast, in Bascom this IC has a write address of 64 (40_{hex} = **&H40**) and a read address of 65 (**&H41**).

```
'-----------------------------------
'UNO_I2C1.BAS Test for valid Adresses
'-----------------------------------
$regfile = "m328pdef.dat"
$crystal = 16000000
$baud = 9600

Dim Addr As Byte
...
Config Scl = Portc.5
Config Sda = Portc.4
I2cinit

For Addr = 2 To 254 Step 2
  I2cstart
  I2cwbyte Addr
  If Err = 0 Then
    Print Addr
    Locate 2 , 1
    Lcd Addr
    Waitms 1000
  End If
  I2cstop
  Next Addr
End
```

Listing 1. Polling for active I²C addresses.

To learn the addresses of the devices connected to the bus, you can use the small Bascom program shown in **Listing 1** (downloadable from [3]). It polls every possible bus address to see whether a device responds. After a device address is output, the Bascom system variable ERR (which does not have to be declared with **Dim** because Bascom has already done this for you) is set to 1 if no acknowledgement is received or to 0 if an Ack signal is received. The latter case means that the address is valid. All even addresses from 2 to 254 are tested, since the odd addresses are the corresponding read addresses of the same devices. For the circuit shown in **Figure 2**, the program reports the addresses 64, 144, 160 and 162 just as expected.

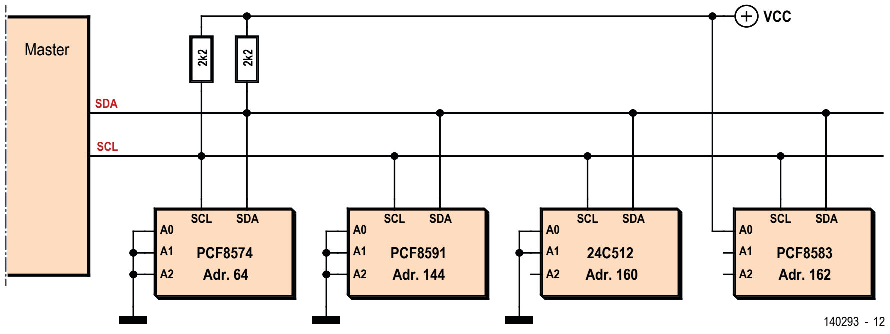


Figure 2. A master and four slaves.

There's another special feature of the I²C bus protocol: every device on the bus can halt the master for a while if it needs a bit more time. To do so it pulls the clock line Low, which forces the master to wait until the line is released again. Bascom follows this convention faithfully. However, this means that a program that uses the I²C bus will hang if no pull-up resistors are connected. From other projects you may be used to the idea that you can sometimes test software without connecting the associated hardware. As a result, you might find yourself staring at an oscilloscope while checking out your software to see whether there are any signals at all on the SCL and SDA lines, while the cable to the Gnublin board is not yet connected. What you have overlooked is that the I²C bus pull-up resistors are on the Gnublin board. Since no pull-ups are present on the host board, everything remains in suspended animation and there are no signals on the I²C bus.

The PCF8574 port expander
The PCF8574 is a port expander IC with eight bidirectional port pins. It does not have a data direction control function. Instead, the port pins have internal pull-up resistors that cause a high level to be present in the quiescent state. This means that after booting you will initially read the port status 255 (binary 11111111). You will only see low values (logic 0) if the port pins are actively pulled to ground by an external device. It is possible to use some of the port pins as outputs and the others as inputs. This requires first configuring all of the input pins in the high state, the same as the output pins.

Figure 3 shows the bus connections and a potential application of the port expander as a digital tester. The port expander can be powered from 3.3 V or 5 V according to the operating voltage of the item under test. Any desired digital circuit with four inputs can be driven using the output port pins P0 to P3. For example, you could apply an incrementing digital value to these four pins to obtain all possible combinations of signal levels on the circuit inputs. Cables and connectors can also be tested the same way. Every open circuit and short circuit can be detected.

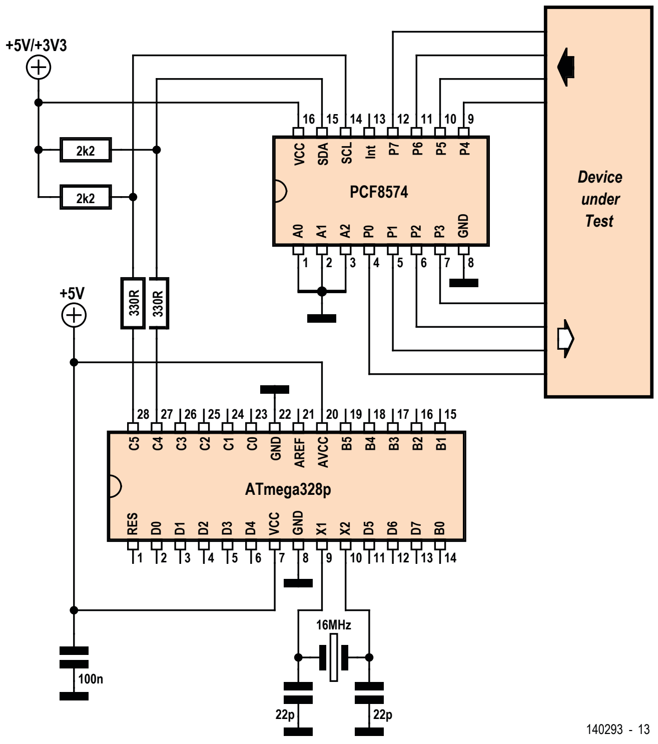


Figure 3. Using the PCF8574 port expander.

Listing 2 shows an example of how to use a mixed set of inputs and outputs. Here pins P0 to P3 output a continuously incrementing digital value. Pins P4 to P7 are used as inputs and must therefore be set high in the write operation (**`Or &B11110000`**) The IC is addressed twice: first in the write direction (address 64) and then in the read direction (address 65). Incrementing the value on the four outputs is the simplest possible type of test stimulus. Depending on the item under test, a completely different test sequence may be necessary, which means you will need different test software.

```
'------------------------------------
'UNO_I2C2.BAS input/output PCF8574
'------------------------------------
$regfile = "m328pdef.dat"
$crystal = 16000000
$baud = 9600

Dim N As Byte
Dim D As Byte
...
Do
  For N = 0 To 15
    I2cstart
```

```
      I2cwbyte 64         'write
      D = N Or &B11110000
      I2cwbyte D
      I2cstop
      Print N;
      Print "  ";
      Locate 2 , 1
      Lcd N
      Lcd "  "

      I2cstart
      I2cwbyte 65         'read
      I2crbyte D , Nack
      I2cstop
      Shift D , Right , 4
      Print D
      Locate 2 , 5
      Lcd D
      Lcd "  "
      Waitms 100
    Next N
  Loop
```

Listing 2. Accessing the PCF8574 ports.

PCA9555 16-bit I/O port
Sometimes eight more lines are not enough. The Gnublin port expander module [2] provides 16 I/O lines using an NXP PCA9555 IC. The board can be plugged directly into the EEC connector on the Elektor Extension shield. There is also a Gnublin relay board that uses the same IC.

NXP is the successor to Philips, and the PCA9555 is the rightful successor to the PCF8574. That's why the two devices have the same bus address: 64 (**`&H40`**). This sort of address recycling makes sense because the address space is limited. In any case, why would you need a PCF8574 when you have a PCA9555? Along with twice as many I/O pins, the new IC has additional functions such as data direction control and inversion of the input data.

Figure 4 shows how the IC can be connected to the Elektor Extension shield using the EEC connector. The I²C bus lines and supply voltage lines are connected using a flat cable. The two pull-up resistors on the Gnublin board are connected to the bus lines by a pair of jumpers. These jumpers must be fitted if only one board is connected. If you use several boards, perhaps connected using the Gnublin distributor board [2], make sure that the pull-up resistors are only enabled on one board. It is possible to use several port expander boards because each board has jumpers for configuring address lines A0 to A2. In the default state, all three lines are tied to ground. That yields a bus address of 64. A total of eight boards can be connected, with addresses from 64 to 72. With 16 I/O pins per board, that gives you a grand total of 128 I/O lines.

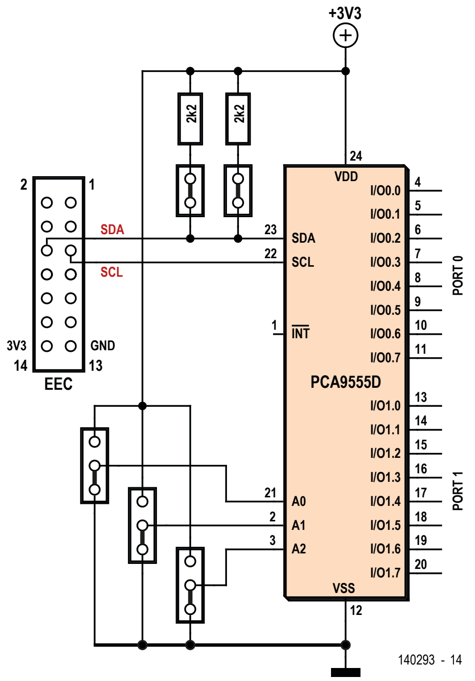


Figure 4. The PCA9555 on the Gnublin board.

Listing 3 shows an application for the PCA9555, which can also be used for testing other circuits. As in the previous example, the port pins are divided up with half of them used for outputs and the other half used for inputs. Since things are a bit more complicated here, a command byte has to be sent after each address [4]. The command selects a register address in the IC. After that you can send or receive one or two bytes of data. For example, if you want to configure port 0 (with eight pins) and port 1 (with the other eight pins), you first send the command '6' after the address and then write two data bytes, which are placed in registers 6 and 7. Zero bits in these bytes mean that you want to configure the corresponding pins as outputs. One bits stand for inputs. In our example, all pins of port 0 are configured as outputs and all pins of port 1 are configured as inputs. Incidentally, pins configured as inputs have integrated high-impedance pull-up resistors (approximately 100 kΩ), so open inputs are read as logic 1. You can use the following command byte values:

0	Input Port 0
1	Input Port 1
2	Output Port 0
3	Output Port 1
4	Polarity Inversion Port 0
5	Polarity Inversion Port 1
6	Configuration Port 0
7	Configuration Port 1

```
'------------------------------------
'UNO_LCD1.BAS input/output PCA9555
'------------------------------------
$regfile = "m328pdef.dat"
$crystal = 16000000
$baud = 9600

Dim N As Byte
Dim D As Byte
...
I2cstart
I2cwbyte 64     'Gnublin port expander
I2cwbyte 6
I2cwbyte 0      'Port 0 output
I2cwbyte 255    'Port 1 input
I2cstop

Do
  For N = 0 To 255
    I2cstart
    I2cwbyte 64      'write
    I2cwbyte 2
    I2cwbyte N
    I2cstop
    Print N;
    Print "  ";
    Locate 2 , 1
    Lcd N
    Lcd "  "

    I2cstart
    I2cwbyte 64      'write
    I2cwbyte 1
    I2cstart
    I2cwbyte 65      'read
    I2crbyte D , Nack
    I2cstop
    Print D
    Locate 2 , 5
    Lcd D
    Lcd "  "
    Waitms 100
  Next N
Loop
```

Listing 3. Using the PCA9555 port expander.

The commands 2 (Output Port 0) and 1 (Input Port 1) are used iteratively in the data loop of the sample program. The IC must be addressed anew for each command. In order to read a port, the IC must also be addressed again with the read bit set (address 65). If you examine the code closely, you may wonder why there is no **`I2cstop`** instruction. That's because the code implements a 'repeated start' without a previous stop condition, since the two accesses always belong together: writing the command to select the register to be read and reading the register contents.

Analog I/O with the PCF8591

The PCF8591 contains an 8-bit A/D converter with four inputs along with an 8-bit D/A converter in the same package. You probably won't need the A/D converter by itself because the Arduino Uno already has enough analog inputs and higher resolution for A/D conversion. However, a real A/D converter can come in handy. Unlike a PWM output, it delivers a true DC voltage. You can use this to build a simple diode tester that measures the forward voltage at a defined current level (see **Figure 5**). The circuit operates at 5 V so that it can also be used to measure the relatively high forward voltage of LEDs.

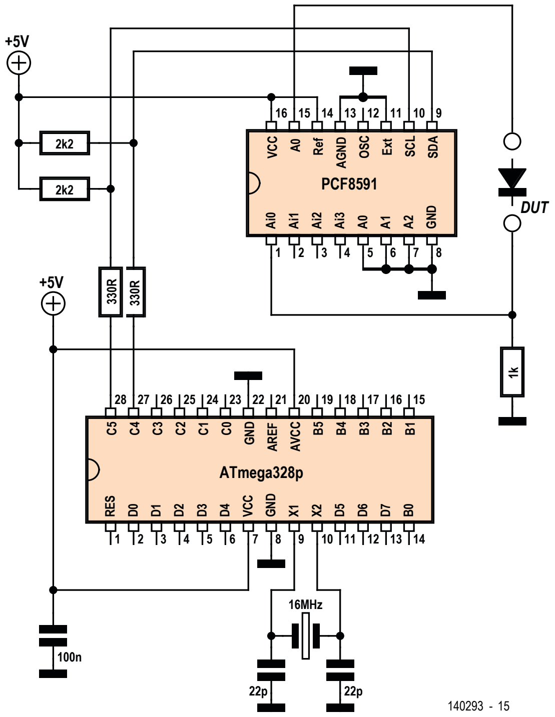

+5V
2k2
2k2
330R
330R
PCF8591
VCC A0 Ref AGND OSC Ext SCL SDA
Ai0 Ai1 Ai2 Ai3 A0 A1 A2 GND
DUT
1k
ATmega328p
C5 C4 C3 C2 C1 C0 GND AREF AVCC B5 B4 B3 B2 B1
RES D0 D1 D2 D3 D4 VCC GND X1 X2 D5 D6 D7 B0
16MHz
100n
22p
22p
140293 - 15

Figure 5. A diode tester using the PCF8591.

The IC has a control register that must be written right after the bus address is sent. A control byte value of 64 enables the D/A converter and selects input channel 0. The following byte is put into the D/A register and results in an output voltage in the range of 0 to 5 V, corresponding to a data range of 0 to 255. With this 8-bit resolution, the output voltage increment is approximately 20 mV. The demo program in **Listing 4** generates a rising voltage ramp and measures the voltage across the 1 kΩ sense resistor at the same time. The PCF8591 has to be addressed again in the read direction (address 145) to read the measured voltage. The read byte value represents the voltage on input Ai0. A reading of 50 indicates a voltage of 1 V, which corresponds to a diode current of 1 mA. At this point the ramp is stopped. Now the forward voltage of the diode can be calculated from the difference between the output voltage and the input voltage. It is displayed in millivolts. With a sample blue-green LED, the following results were displayed at the end of the measurement cycle:

```
1 mA
2920 mV
```

```
'-------------------------------------
'UNO_I2C4.BAS AD/DA PCF8591
'-------------------------------------
$regfile = "m328pdef.dat"
$crystal = 16000000
$baud = 9600

Dim N As Byte
Dim D As Byte
Dim U As Word
...
N = 0
Do
  I2cstart
  I2cwbyte 144            'write
  I2cwbyte 64             'DA enable
  I2cwbyte N
  Print N;
  Print "  ";
  Locate 2 , 1
  Lcd N
  Lcd "   "
  I2cstart
  I2cwbyte 145            'read
  I2crbyte D , Nack       'Ain0
  I2cstop
  Print D
  Locate 2 , 5
  Lcd D
  Lcd "  "
  Waitms 100
  N = N + 1
  If D >= 50 Then Exit Do
Loop
```

```
Cls
Locate 1 , 1
Lcd " 1 mA"
U = N - D
U = U * 20
Locate 2 , 2
Lcd U
Lcd " mV"
End
```

Listing 4. A diode tester using the PCF8591.

Future prospects

This is the last installment of our Microcontroller BootCamp series, but there will be other articles on Bascom applications for the Arduino Uno and the Elektor Extension shield from time to time.

We hope we have aroused your enthusiasm for developing your own programs. If so, we encourage you to share the results of your efforts with other members of the Elektor community at www.elektor-labs.com. Relatively small projects or applications still in the development stage are especially welcome.

Other interesting I²C components

Anyone who reads Elektor regularly is always running into interesting ICs with an I²C interface. They often form the inspiration for new projects. Some particularly significant types are:

- I²C EEPROMs up to 64 KB (for example, the 24C512). These can be used to build data loggers and lots of other things.

- The CY27EE16 is a crystal clock generator that can be programmed over the I²C bus. It is used in the Elektor Software Defined Radio project. Software control with a microcontroller opens the door to new possibilities.

- The SI4735 is a complete AM/FM receiver and has already been used in several Elektor projects along with Bascom. Any desired frequency can be set using just a few I²C commands.

- High-resolution A/D and D/A converters often have I²C ports. One example is the ADS1115 16-bit A/D converter recently described in Elektor.

If you want to delve deeper into this subject, you can even build your own I²C bus IC. A Bascom library for programming I²C slave devices is available for this purpose. That is a bit more difficult than programming a bus master because the slave device must be able to handle the bus speed set by the master. The *Mastering the I²C Bus* book has all the details.

Web Links

[1] www.elektor-magazine.com/140009
[2] www.elektor.com
[3] www.elektor-magazine.com/140293
[4] www.nxp.com/documents/data_sheet/PCA9555.pdf

Further Reading

[a] *Mastering the I^{2}C Bus,* Vincent Himpe, Elektor International Media Publishers, ISBN 978-0-905705-98-9. Also available as an E-book.
www.elektor.com/mastering-the-i2c-bus

Chapter 46 • Sensors Make Sense (1)

For Arduino et al.

For some time now Elektor has offered a kit of 35 sensors and actuators [1]. It's mighty popular because with these parts you can produce all kinds of applications in fields like test & measurement, robotics or household automation. Whether you favor the Arduino or some other platform is irrelevant; the possibilities are endless.

With so many types of sensor (from joysticks to temperature and humidity monitors) and output devices (from relays to laser lights) it's no easy task to keep up with all the possibilities. This series of articles will make it easier for you by presenting practical applications. As our base point we'll use an Arduino Uno, which can be programmed both in Arduino C and in Bascom.

Overview: sensors and actuators

Analog sensors take center stage in this first part of the series. Sensors for light, temperature and magnetic field strength, together with a joystick with two potentiometers, are best examined alongside the appropriate actuators with which they work hand in hand. Here is where relays, dual-color LEDs, RGB LEDs and laser lights come into action. By way of comparison, we've checked out some digital sensors too.

All the sensors and actuators mentioned here have their schematics and connection details shown in **Figure 1**. The designations correspond with those used in the sensor kit. The legends on the connections match those on the PCB too. In this way you can identify what you're looking at and get your bearings. On the right, next to the connection pins, you have in each case suggested connections to the Arduino Uno, as used in the sample programs.

If you take a close look at the multiplicity of sensors in the kit, you soon see that some of them use the same PCBs. This is not just cost-effective but also logical, because they share the same basic principles. A good example of this is the matching circuitry of the NTC temperature sensor (*Analog Temp*) and the LDR sensor (*Photoresistor*) [NTC = negative temperature coefficient; LDR = light-dependent resistor]. Both are sensors that vary their resistance according to the value measured (temperature or brightness). Usually they are both used in conjunction with a voltage divider or a fixed resistance.

The voltage divider is particularly simple, yet despite this, it also has the advantage of accuracy. This is because whenever the same voltage is used both for the voltage divider and as reference for the A-D converter, errors arise that can be traced back to inaccurate voltage. If you use the Arduino's 5-V connection, you have at this point a truly exact 5 V (when used with an AC power supply) but major variations can occur when you take power from a USB connection. However, the A-D converter provides the right result regardless of accurate supply voltage, as the value arises out of the relationship between the measured voltage and the reference. In this way you register the resistance ratio in the sensor circuitry direct, so to speak.

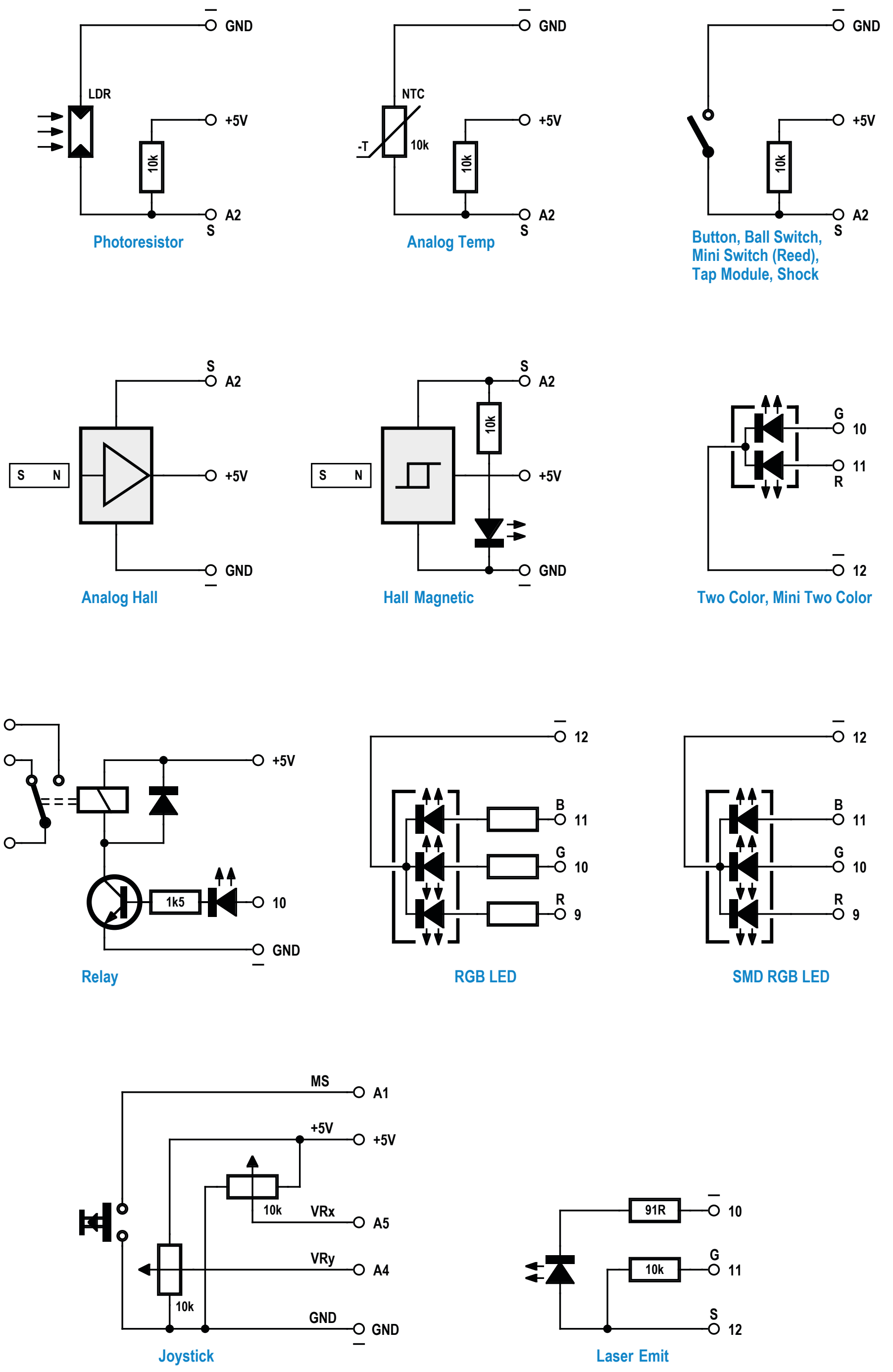


Figure 1. Schematics and connections for the sensors and actuators used.

The way in which the two sensors are connected means we always get a falling voltage with more light and higher temperature. When programming we need to take this into account. But in this case it is also possible to transpose GND and VCC, in order to achieve a rising voltage when measured values increase. All the same, perhaps you should better not do this, because other sensors cannot tolerate this and the diversity of sensors means that any potential source of errors must be avoided. For example the analog magnetic sensor (*Analog Hall*) is, in principle, connected exactly in this manner, albeit with an additional integrated measurement amplifier that requires a supply voltage of the correct polarity. Note the differing connections of the LDR and the NTC!

Looking at the digital sensors, it's readily apparent that they use the same PCB as the LDR and the NTC sensor. This holds good for the pressbutton switch (*Button*) and the reed switch (*Mini Switch*). Here again we find a sort of 'voltage divider' made up of a switch and a 10-kΩ resistor. Because this switch, according to its state, is either an infinite resistance or none at all, the voltage measured is either 5 V or 0 V. Consequently you can also interrogate this sensor to obtain a voltage measurement on an analog input, when in this case a digital input would also suffice.

Using analog sensors is generally quite straightforward: you measure a voltage with the A-D converter and then analyze or display it. For doing this you will find plenty of points of reference in our Bascom teach-in course from 2014 [2]. The way you use the Arduino Uno with Bascom is described in detail there. You can either control the Arduino Bootloader direct or flash the customized MCS Bootloader into the controller. Incidentally, during the preparation of this series of articles we used two Unos. One of them was left untouched, whilst the other was equipped with the MCS Bootloader, which worked trouble-free with Bascom.

Once again we set the Elektor Extension Shield [3] to work, exploiting its display and control functions (such as switches, LEDs and text display) to further expand our range of possibilities. This time all of the sample exercises are presented in Arduino C. This enables everybody to work effortlessly in their own preferred environment. You do nevertheless need to keep an eye on the connector assignment (configuration) of the Extension Shield, which is shown again in **Figure 2** as a reminder:

- Port B (Arduino pins 8 to 13): unassigned;
- Port B.2 (Arduino pin 10) LED2 via jumper, also used for PWM output;
- Port B.5 (Arduino pin 13) LED on the Arduino;
- Port D (Arduino pins 0 to 7): fully assigned to the UART and LCD;
- AD0 and AD1: Switches S1 and S2 versus GND, also usable as analog inputs;
- AD2: LED1 via jumper, otherwise unassigned (preferred input for analog sensors);
- AD3: Pot on the LCD shield;
- AD4 and AD5: Unassigned, but frequently used for the I^{2}C interface.

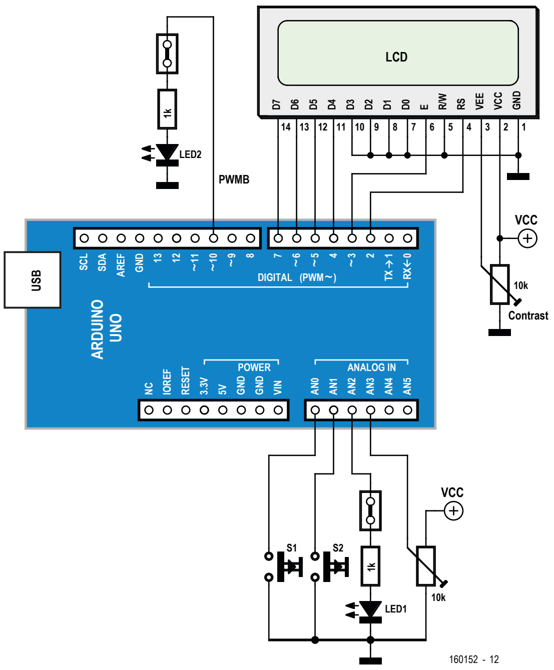


Figure 2. The Extension Shield supplements the Arduino Uno with LEDs, pushbuttons, a pot and a display.

Do take care when using AD0 and AD1! Because the pressbuttons here are integral to the board and cannot be separated from it, it is possible to accidentally short-circuit the voltage being measured. Many sensors just ignore this but some react sulkily. Particularly at risk is the joystick fitted with two pots. If you connect the wiper of a pot (in this case via one of the pressbuttons) to GND at the same time as an external contact is at +5V potential, and you then rotate the wiper in the direction of this pin, the resistance becomes ever smaller and the current flow ever greater. A full short circuit would be only half as bad, as the Arduino has a self-resetting fuse. But there is a point near the end stop of the pot where it will get extremely hot and in this situation many a pot has given up the ghost in a puff of smoke. Consequently you will do better to steer clear of AD0 and AD1, also AD3 too, because there's an internal pot on the Shield in parallel and in the worst-case scenario you could end up destroying two pots. You'll do better to connect the joystick to AD4 and AD5, assuming you can manage without I^{2}C.

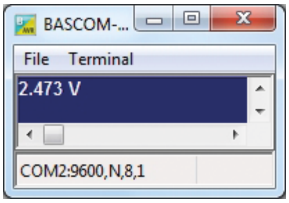
Displaying voltages in Bascom

Now comes our first sample program (all of these programs can be downloaded at [4]). A simple voltage display is handy for testing different sensors. Here we will use analog input ADC2. The jumper contact to LED1 on the Shield needs to be open (i.e. disconnected) for this. The raw data in the range 0 to 1023 provided by the A-D converter is converted into voltages, output in serial format and displayed on the LCD.

```
'-----------------------------------------
'ADvolt.BAS    0...5 V
'-----------------------------------------
$regfile = "m328pdef.dat"
$crystal = 16000000
$baud = 9600
$hwstack = 16
$swstack = 16
$framesize = 64


Dim D As Word
Dim U As Single
Dim S As String * 10

S1 Alias Pinc.0
S2 Alias Pinc.1

Config Adc = Single , Prescaler = 64 , Reference = Avcc
Config Lcdpin = Pin , Db4 = Portd.4 , Db5 = Portd.5 , Db6
  = Portd.6 , Db7 = Portd.7 , E = Portd.3 , Rs = Portd.2
Config Lcd = 16 * 2
Waitms 50
Portc.0 = 1                     'Pullup
Portc.1 = 1                     'Pullup
Config Portb = Output


Cls
Cursor Off

Do
  D = Getadc(2)
  U = D * 5.0
  U = U / 1023
  U = U * 1000
  D = Int(u)
  'Print D
  S = Str(d)
  S = Format(s , "0.000")
  Print Chr(13);
  Print S ; " V     ";
  Locate 1 , 1
  Lcd S ; " V   "
  Waitms 500
Loop
```

Listing 1. Voltage readout.

A special feature of the Bascom program (**Listing 1**) is its serial output, which is customized specially for the Bascom Terminal. A frequent distraction on the Terminal is new lines of output constantly appearing one after another, caused by every print instruction being ended with Carriage Return (CR, ASCII 13) and Line Feed (LF, ASCII 10). In fact you can suppress these control characters with a semicolon at the end of the print instruction and by sending just a CHR(13) before the instruction. Each new output then appears on the same line, resulting in a static display (**Figure 3**).



```
2.473 V
```

COM2:9600,N,8,1

Figure 3. Voltage readout in Bascom Terminal.

Another special feature is the use of the Format function. An integer D is first transformed into a string S. Then we establish an edit mask ‘0.000’, so that the last three places are placed behind the decimal point. To ensure everything works properly, a voltage like 1.234 V needs first to be multiplied by 1000, so that you end up outputting 1234 mV.

The string prepared in this manner is sent serially to the Terminal and also displayed on the LCD. If you wish to use the program without using the Shield, you can do this without any modification, as outputs to the LCD do not wait for any acknowledgment and simply disappear into the void if nothing is connected. Conversely, once you have loaded the program, you can also use it without needing the PC any more and just look at the LCD. If you feel like testing the program first without a sensor, simply bridge a wire between AD2 and AD3. Then it’s up to the pot on the Shield, so that you can set any voltage of your choice between 0 V and 5 V and have it displayed.

The universal voltage measurement device can now be used with a variety of sensors.

- The NTC sensor indicates a voltage of 2.5 V at 25 °C and a falling voltage with rising temperature. A still more exact analysis is displayed.

- The LDR sensor outputs a broad voltage range between almost zero (very bright) and almost 5 V (very dim).

- The pressbutton and the reed switch indicate 5 V in idle state. When you press the button or hold a strong magnet next to the reed switch, the voltage drops to 0 V.

- The analog Hall sensor delivers around 2.5 V in standby mode. Rotating it spatially makes no definitive change caused by the Earth’s magnetic field. Bringing powerful magnets close to it cause readings of between around 1 V and 4 V, according to their direction.

- The digital Hall sensor is connected in the same way but includes a Schmitt trigger. In idle mode it measures over 3 V but when activated by a magnet approaching on the correct direction, the reading is almost zero.

If you need to process the measured data further, you can open a Log file in the Bascom Terminal and close this again at the end of measurement process. In this case it is better to use the commented output in the program 'Print D'. D employs whole integers and indicates the voltage in mV, so that no errors can arise with decimal points. All data collected is then stored and remains available for subsequent processing, for instance in a spreadsheet program. You can also produce charts and diagrams in this way (**Figure 4**).

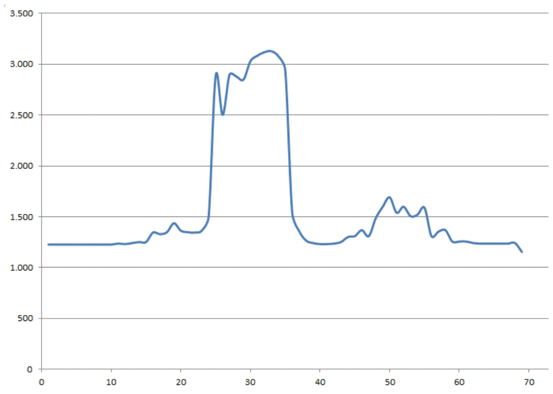


Figure 4. Excel graphic of a brightness curve.

Measuring voltages with Arduino

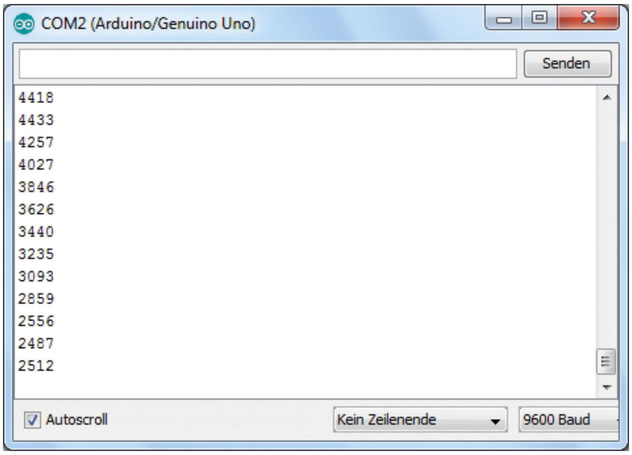
Programming in C with the Arduino IDE is not much different from programming in BASIC with Bascom. The universal-use program *VoltageAD2.ino* (**Listing 2**) again converts the raw output (value) from the A-D converter into the voltage in mV. At the same time this sample program shows how the LCD on the Elektor Shield is controlled to output the voltage in volts on the LCD. Without any additional formatting, the result is shown as two places after the decimal point.

```
//VoltageAD2 0...5000 mV at AD2
#include <LiquidCrystal.h>
int sensorPin = 2;
int value = 0;
float voltage;
LiquidCrystal lcd(2,3,4,5,6,7);

void setup() {
  Serial.begin(9600);
  lcd.begin(16, 2);
}
void loop() {
  value = analogRead(sensorPin);
```

```
      voltage = value;
      voltage = voltage * 5000 / 1023;
      value = voltage;
      Serial.println(value);
      lcd.setCursor(0, 0);
      lcd.print(voltage/1000);
      lcd.print (" V   ");
      delay(500);
    }
```

Listing 2. Voltage measurement and LCD readout using the Arduino.

You can see the measurement readings sent (voltages in mV) on the serial monitor (**Figure 5**). Starting with version 1.6.8, the Arduino software has been enhanced to handle serial plotters (**Figure 6**), enabling them also to receive and display simple sequences of numbers. In this way you can create a continuous diagram like on a plotter with an endless roll of paper. The range of measurement adjusts itself continually to the data sent. The smallest measurement range spans from –10.0 to +10.0.



```
4418
4433
4257
4027
3846
3626
3440
3235
3093
2859
2556
2487
2512
```

Autoscroll [✓] · Kein Zeilenende · 9600 Baud

Figure 5. Measured values on a serial monitor.

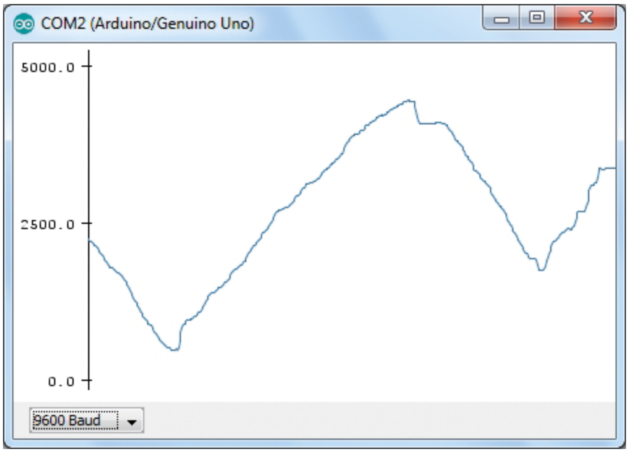


Figure 6. Measured values on a serial plotter.

Temperature measurement using NTC sensors
The nominal resistance of NTC sensors is quoted for a temperature of 25 °C. Additionally a so-called ‘B value’ is stated, which defines the gradient of the sensor’s characteristic curve. The variation in resistance proceeds to a close degree exponentially according to the change in temperature. This means that the voltage change is sufficiently large as to provide very good resolution without the need for a measurement amplifier.

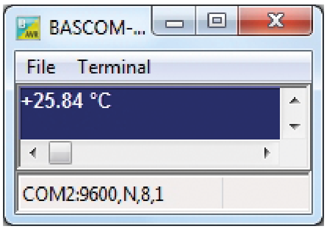
Precision sensors are manufactured mainly with a nominal resistance of 10 kΩ and a *B* value of 3900. With this information we can evaluate temperatures (**Listing 3**). This is a two-step process. First we calculate the resistance *R* of the sensor from the measured value *D*, i.e. raw data in the range 0 to 1023. In the second step we determine from this the temperature in degrees using logarithms and further stages of calculation.

```
Dim D As Integer
Dim B As Integer
Dim R As Single
Dim T As Single
Dim S As String * 10
…
Do
  D = Getadc(2)
  B = 1023 - D
  R = D / B           'R = 1 @ 10k
  T = Log(r)
  T = T / 3900        'B25/85=3900
  T = T + 0.0033557   '1(273+25)
  T = 1 / T
  T = T - 273
  T = T * 100
  D = Int(t)
```

```
    S = Str(d)
    S = Format(s , "+0.00")
    Print Chr(13);
    Print S ; " " ; Chr(248) ; "C   ";
    Locate 1 , 1
    Lcd S ; " C   "
    Waitms 500
  Loop
```

Listing 3. Analyzing NTC temperatures.

The result of the calculation also provides a remarkably accurate temperature display without calibration (**Figure 7**). These sensors normally have an accuracy of 1%, meaning the maximum variation is less than 1 K. To round off the project, we'll again create a static display in the Terminal with optional simultaneous output to the LCD. No deception is intended in matters of accuracy; we are not boasting to display hundredths of a degree but to provide a better view of small temperature variations. One A-D step stands for around 0.1 K at average temperatures. At very low or very high temperatures the resolution becomes coarser.



```
+25.84 °C
```

COM2:9600,N,8,1

Figure 7. Temperature display using an NTC sensor.

In Arduino C the conversion process (**Listing 4**) functions, in principle, exactly as in Bascom. The most obvious difference is that in C you can write the entire calculation on one line, whereas Bascom and other dialects of BASIC observe the restriction that only one arithmetic step can appear on a given line.

```
//NTCtempAD2 10 k NTC at AD2
#include <LiquidCrystal.h>
int sensorPin = 2;
int relaisPin = 10;
int value;
float fvalue;
float resist;
float temp;
LiquidCrystal lcd(2,3,4,5,6,7);

void setup() {
  Serial.begin(9600);
  lcd.begin(16, 2);
  pinMode(relaisPin, OUTPUT);
}

void loop() {
  value = analogRead(sensorPin);
  fvalue = value;
  resist = fvalue / (1023-fvalue);
  temp = 1/(log (resist) / 3900 + 0.0033557) - 273;
  Serial.println(temp);
  lcd.setCursor(0, 0);
  lcd.print(temp);
  lcd.print (" C   ");
  if (temp > 25 ) digitalWrite (relaisPin, HIGH);
  if (temp < 20 ) digitalWrite (relaisPin, LOW);
  delay(500);
}
```

Listing 4. Measuring temperatures with the Arduino.

The Arduino program includes yet another useful enhancement with control for the relay module in the sensor kit. The relay operates with 5 V and is switched via a driver transistor. The base resistance of 1.5 kΩ along with the series-connected status LED is applied direct to a Portpin. Here Pin 10 (B.2) is selected as output, as this enables LED2 on the Shield to be controlled at the same time. The relay is now used in conjunction with the NTC sensor for operating a ventilator fan. When the temperature rises above 25 °C, the relay switches on the fan. If the temperature drops below 20 °C, it is switched off again. This gives us a two-setpoint controller with a hysteresis of 5 degrees.

RGB LED with joystick control

The sensor kit contains two RGB LEDs, one in standard LED form with three dropper resistors (150 Ω each), and one SMD version without any dropping resistors whatsoever. You often hear the claim that it's fine to connect LEDs to batteries or Ports without using dropper resistors ("No problem. I checked and nothing nasty happened.") But doing this clearly exceeds the threshold values of the LEDs and the microcontrollers, and overloads of this kind reduce the LEDs' life span. It's far better using dropper resistors of 150 Ω, which at 5 V will limit the current to 20 mA. If you want to avoid every risk, you will also connect these series resistors externally to the SMD LED.

But there's yet another compromise, which will certainly not win any design prize, that still needs addressing. In this we connect the common (shared) cathode of the RGB LED not to GND but to a Portpin that is switched low. The current is then limited by the Port's internal resistance. Each Portpin has an internal resistance of around 20 Ω. Because an LED is connected to two Portpins, we have effectively a series resistance of 40 Ω. With an LED voltage of 3 V and an operating voltage of 5 V the LED current is 50 mA. Oh well, it still works. And if all three LEDs are switched on, there's a greater voltage drop to be borne by the Portpin at the cathode. A voltage of 1.2 V is measured here, which points to a total current of 60 mA. In fact only 40 mA is permitted for an individual Portpin and 60 mA is more than this, although not enough to expect any harm to occur. The total load for all Ports together is 200 mA.

The Arduino program *JoystickRGB* (**Listing 5**) uses the PWM outputs 9, 10 and 11 (B1, B2, B3) for controlling individual LEDs. The common cathode goes to Pin 12 (B4), configured as an output.

```
//JoystickRGB
#include <LiquidCrystal.h>
int joyX = A4;
int joyY = A5;
int red = 9;
int green = 10;
int blue = 11;
int cathode = 12;
int value;
int midX;
int midY;

LiquidCrystal lcd(2,3,4,5,6,7);

void setup() {
  Serial.begin(9600);
  lcd.begin(16, 2);
  midX = analogRead(joyX);
  midY = analogRead(joyY);
  pinMode(cathode, OUTPUT);
  digitalWrite(cathode, LOW);
}

void loop() {
  value = analogRead(joyX);
  value = constrain(value, 35, 980);
  lcd.setCursor(0, 0);
  lcd.print(value);
  lcd.print("     ");
  if (value >(midX+200)) {
    analogWrite(red , 0);
    analogWrite(green , 0);
    analogWrite(blue , 0);
  }
  if (value <(midX-2)) {
    analogWrite(blue, (midX - value)/2);
  }
  value = analogRead(joyY);
  value = constrain(value, 35, 980);
```

```
    lcd.setCursor(0, 1);
    lcd.print(value);
    lcd.print("     ");
    if (value > midY) {
      analogWrite(red, value -midY);
    }
    if (value > (midY+2)) {
      analogWrite(red, (value - midY)/2);
    }
    if (value < (midY-2)) {
      analogWrite(green, (midY - value )/2);
    }
    delay(500);
  }
```

Listing 5. RGB control with the joystick.

The intensity or brightness of each individual color can now be controlled with the joystick. For the reasons mentioned above, inputs A4 and A5 are used for polling the two pots. The control function uses the joystick in a way in which each axis is divided in two halves. Right, left, up, down together make four functions, so that as well as the three colors, we can also control the total brightness. In this case winding back to zero means every light goes out. In use it works like this: each color is adjusted for the desired intensity. If you release the control knob rapidly back to its rest position, the last value remains in force, because a new measurement takes place only every 500 ms. The fourth channel (down) switches all the LEDs off.

At program start-up the neutral center positions *midX* and *midY* are measured. A certain variation up to 30 steps from the center of the measurement range (512 = 2.5 V) is normal and must be taken into account in the program. Helpful in this respect is the constrain(value, 35, 980) function, which prunes the range of values within 35 up to 980. Another thing to note in the control process is that each PWM output is controlled within the range 0 to 255, with half of the pot's range covering double the value range. Finally you need to allow some tolerance when comparing whether you are measuring above or below the center position.

Laser light control
The program also lends itself to testing dual-color LEDs. You can then adjust red and green separately and mix them together appropriately. Even the 'laser' can be controlled without difficulty using the same software. You can make adjustment to any desired degree of brightness and because the PWM signal produces rapid switching on and off, it is remarkably easy to 'paint' the walls with a kind of oscillogram effect. By varying the beam with suitably rapid motion you end up seeing red lines and gaps that demonstrate the PWM ratio. By default the Arduino software uses a low frequency to enable the switching operation to dissolve without difficulty. Contrastingly, the Bascom example uses the maximum PWM frequency, so that you would need an oscilloscope to recognize or distinguish the PWM signals. The equivalent program in Bascom is called *JoystickRGB.bas* (**Listing 6**) and differs in its

method of operation. The joystick movements assigned to colors drive the relevant LED only in the 'brighter' direction. Accordingly the maximum value is used for the actual value and setting in every case. As a result the measurement loop runs very quickly, because it no longer has to worry about returning back to the zero position. The control thus becomes more fluid and more delicate. At the same time the 'darken' function is now infinitely variable. Altogether this produces excellent adjustability of any desired color mix.

```
Do
  D = Getadc(4)
  Locate 1 , 1
  Lcd D ; "  "
  If D > 980 Then D = 980
  If D < 35 Then D = 35
  D = D - 2
  If D > Midx Then
     Pwm = D - Midx
     Pwm = Pwm / 2
     Print Pwm
     Pwm = 200 - Pwm
     Print Pwm
     If Pwm < 0 Then Pwm = 0
     If Pwm < Red Then Red = Pwm
     If Pwm < Green Then Green = Pwm
     If Pwm < Blue Then Blue = Pwm
  End If
  D = D + 4
  If D < Midx Then
     Pwm = Midx - D
     Pwm = Pwm / 2
     If Pwm > Red Then Red = Pwm
  End If

  D = Getadc(5)
  Locate 2 , 1
  Lcd D ; "  "
  If D > 980 Then D = 980
  If D < 35 Then D = 35
  D = D - 2
  If D > Midy Then
     Pwm = D - Midy
     Pwm = Pwm / 2
     If Pwm > Blue Then Blue = Pwm
  End If
  D = D + 4
  If D < Midy Then
     Pwm = Midy - D
```

```
        Pwm = Pwm / 2
        If Pwm > Green Then Green = Pwm
     End If

     Pwm1a = Red
     Pwm1b = Green
     Pwm2a = Blue
     Waitms 10
   Loop
```

Listing 6. Excerpt from the program JoystickRGB.bas.

Web Links

[1] www.elektor.com/arduino-sensor-kit
[2] www.elektormagazine.com/120574
[3] www.elektormagazine.com/140009
[4] www.elektormagazine.com/160152

Chapter 47 • Sensors make Sense (2)

For Arduino and more

Sensors are either analog or digital. Analog values are read in via an A-D input, whereas for digital signals a simple Port connection frequently adequate. In some cases, however, sensors have dual outputs, one analog and one digital.

A glance at the sensor combo kit (available from Elektor [1]) reveals seven superficially identical PCBs equipped with differing sensors. All of them have one analog output AO and one digital output DO. As you know, to create a digital signal from an analog one, we use a comparator. There is one of these included on each of these boards, or more precisely an LM393 dual differential comparator.

Sensors equipped with comparators

When we examine the circuit of the board in **Figure 1** more closely, its function becomes clear. Once more we find the actual sensor in a voltage divider that this time is preset with a 25-turn precision trimmer pot. A second voltage divider made up from two 100 kΩ resistors provides a reference voltage of 2.5 V for the comparator. The temperature sensor (*Digital Temp*) can now be adjusted so that the sensor voltage at the desired temperature will likewise be exactly 2.5 V. A rising temperature at the NTC sensor makes the comparator switch on the digital output DO, whilst a falling temperature switches it off. The second comparator connected downstream operates only the status LED. Thanks to these LEDs we can test all seven sensors without any need for software.

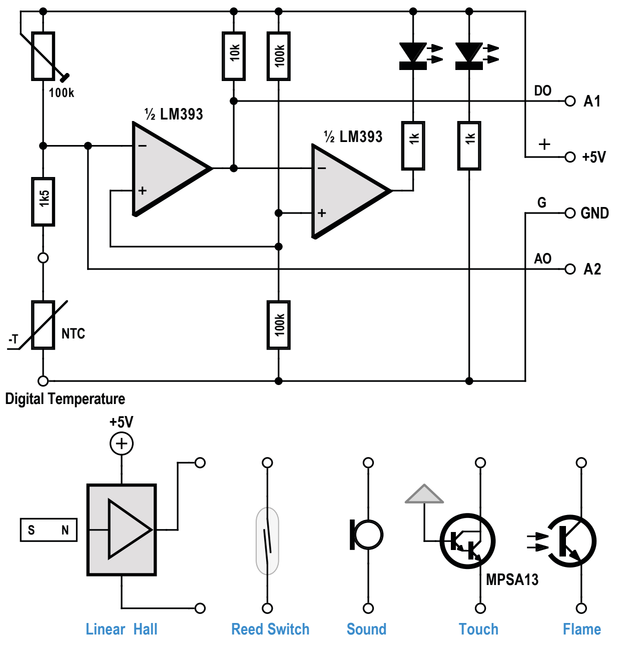


Figure 1. Sensors with comparators.

The circuit does not include a bypass capacitor (see **inset**), meaning that any variations or spikes on the supply voltage may affect the operation. The length of the wiring and other random factors can also have some influence. In actual fact with the temperature sensor and the heat rising, we find a narrow region in which the output oscillates, which is possibly caused by parasitic capacitance on the PCB. If we fit a small capacitor between the input and the second comparator output, we can see an oscillator in action. This is easy to recognize with the oscilloscope. The status LED on the sensor PCB shows it as well. When things warm up slowly, it lights up initially at half brightness (state of oscillation) and only after this does it reach full brilliance (stable condition). If the software behaves strangely, bear this point in mind.

The other sensors using this circuit behave in more or less the same way, even if the changes in the measured value are usually not so gradual. Most comparable with the NTC sensor is the phototransistor in the flame sensor. The dark housing allows the longer wavelengths to pass through preferentially, in order for flames to be detected. The Hall sensor differs in that it expects an operating voltage on its third pin. The fact that the reed switch is connected as an analog sensor may surprise you. But there are two out-of-phase outputs. When a magnet is brought closer, one output goes on and the other goes off.

Sound and touch behave somewhat differently; normally they deliver a squarewave signal with rapid transitions. The two sound sensors employ one large and one small electret microphone. With the pot adjusted correctly, half waves of loud sound signals appear at the output as square waves. This must be taken into account during your evaluation testing. In the same way the touch sensor generally provides a 50 Hz squarewave signal (60 Hz in North America) when activated.

The comparator circuit delivers a sharp switchover point, without any hysteresis. This means that with a slow change of temperature, a region can occur in which the output flutters a bit. In a program you can take account of this and poll the digital output only at longer intervals. Or you could use the analog output signal AO direct from the voltage divider and evaluate this, something that we'll describe in greater detail below. Unlike the analog NTC sensor already described, you will then have the advantage that a desired temperature can be set externally using the pot.

It's also interesting to note that the digital output of this board can control an actuator directly, without any assistance from a microcontroller. Output DO is connected to the input of the relay PCB. That's it — the temperature controller or thermostat is complete. Admittedly without hysteresis, which in some applications could be problematic. If you feel like connecting one of the LEDs directly to the comparator, you need to bear in mind that a comparator has an open collector output. Therefore it switches down actively, so that in the High state only the pull-up of 10 kΩ supplies current, meaning that the LED does not illuminate very brightly. Nevertheless, applications are conceivable in which three separate sensors directly drive the three colors of an RGB LED (**Figure 2**), not with great intensity but clearly visible nevertheless. The resulting color mixture of pink perhaps then stands for "hot and loud".

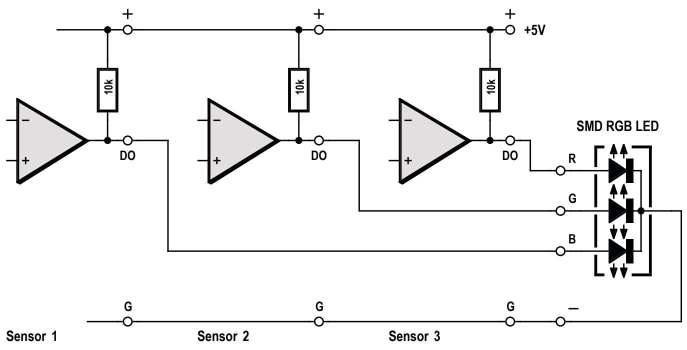


Figure 2. Direct connection to RGB LED.

If you wish to control laser lights directly, you'll need to hook these up between DO and +5 V (**Figure 3**). On/Off reverses the situation, but you can now switch larger currents up to about 20 mA. With this arrangement an interesting experiment can be carried out: opto-thermal feedback. The laser is pointed precisely towards the NTC sensor (see header photograph). The switching temperature is then set exactly at the changeover point using the potentiometer. Once the correct position is found, the laser starts to flash. Here's how: when switched on, it warms the sensor slightly. The digital output switches on, which turns off the laser because of the inverted connection. This gives the sensor the chance to cool down again until the comparator flips and the sequence begins once more.

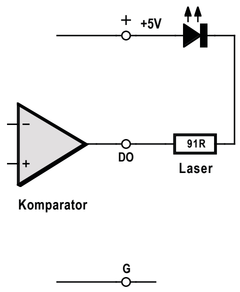


Figure 3. The laser on the comparator output.

Software-based Schmitt trigger

When you have a transition region a comparator can generate highly volatile output states. A Schmitt trigger helps mitigate this by displacing the transition points that set the switch-on and switch-off points. An example is switching on at 25 degrees Celsius and off at 20 C, as discussed regarding the NTC sensor (*Analog Temp*) in first part of this series.

All the sensors on the comparator board also have an analog output, as you know. There is nothing to stop you making a comparison in the software, and in so doing, building in some appropriate hysteresis. Because all the sensors are adjusted using the trimpot to a switching point of 2.5 V, the same software can be used for the differing sensors.

In the software the two outputs B.2 (LED 2 on the Elektor Extension Shield, see Part 1 [2]) and B.5 (LED on the Arduino board) are used as outputs. There's a specific reason for switching them in antiphase. You might connect the dual-color LED to both outputs. Or you might hook up actuators of this kind between both outputs, which in reality still require a series (dropper) resistor but in this situation would at least see two internal resistances in series. This class of actuators also includes the SMD RGB LED and the infrared LED. The color-changing LED discussed later on belongs in this category too, but it includes a reverse-voltage protection diode, which will not tolerate the inverted voltage level cheerfully. In this case you are better off using a genuine 'real' dropper resistor.

from the A-D converter (as always, all of the code samples can be downloaded from the Elektor Magazine website [3]). The center of the measurement range represents a value of 512. The switching threshold points in this example lie at 509 and 515. With around 5 mV per A-D step we then have a hysteresis of 30 mV. The program also controls LED2 on the Extension Shield. This means you can compare the switchover with that on the sensor board yourself. The software offers two slightly displaced switchover points as opposed to the sharply defined switching point of the sensor-comparator. The complete program additionally displays the analog sensor voltage, both on the LCD of the Shield and on the terminal screen. This helps you find the correct trimmer setting.

```
Do
  D = Getadc(2)
  If D > 514 Then Portb.2 = 0
  If D > 514 Then Portb.5 = 1
  If D < 510 Then Portb.2 = 1
  If D < 510 Then Portb.5 = 0
...
  Waitms 500
Loop
```

Listing 1. A comparator with hysteresis (Comparator.bas).

The Arduino version of the program (**Listing 2**) differs little from the BASIC example. The analog signals can be displayed without additional overheads using a serial plotter. **Figure 4** shows signals from the microphone sensor. This program can be used again for all seven sensors on the red comparator PCB.

```
//Comparator AD1
...
void loop() {
  value = analogRead(sensorPin);
  if (value > 514) {
    digitalWrite (output1, 1);
    digitalWrite (output2, 0);
  }
  if (value < 510) {
    digitalWrite (output1, 0);
```

```
        digitalWrite (output2, 1);
      }
      Serial.println(value);
      lcd.setCursor(0, 0);
      lcd.print(value);
      lcd.print ("   ");
      delay(50);
    }
```

Listing 2. The Arduino comparator.

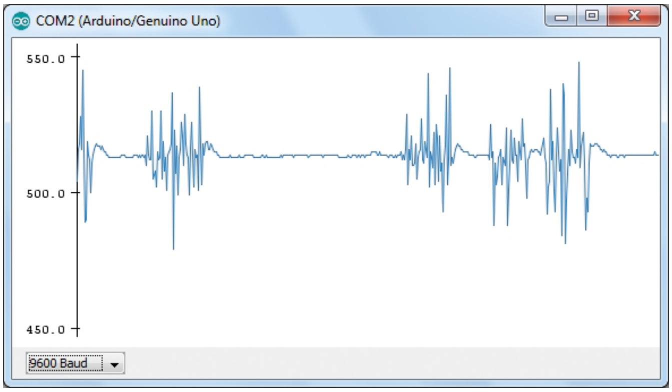


Figure 4. Microphone signals.

Polling contact sensors

The touch sensor does not produce a slowly varying signal, but rather a (normally) square-wave signal of 50 or 60 Hz. Were you to use this for controlling a relay, the result would be wavering, noisy and inelegant. However, the signal can be improved in software (**Listing 3** and **Listing 4**). The digital output in this case is connected to AD1. Yes, this is an analog input but you can also use it as a digital input Port. The result of polling it will be 1 or 0 and can be copied direct to an output Port. Here we chose B2 (Arduino pin 10), because it can also control the LED2 on the Shield. Additionally B5 is also controlled, as that's where the Arduino-internal LED is located. Externally you can also hook up another LED or the laser light.

```
Dim D As Boolean
Config Portb = Output

Do
  Portb.2 = Pinc.1
  D = Pinc.1 Xor 1
  Portb.5 = D
  Waitms 21
Loop
```

Listing 3. Input and output Port in Bascom.

```
//Touch1  A2 > 10, 13
#include <LiquidCrystal.h>
int input = A2;
int output1 = 10;
int output2 = 13;

void setup() {
  pinMode(output1, OUTPUT);
  pinMode(output2, OUTPUT);
}
void loop() {
  digitalWrite (output1, digitalRead(input));
  digitalWrite (output2, 1-digitalRead(input));
  delay(21);
}
```

Listing 4. Input and output Port using Arduino-C.

The program consists of a simple loop that has a waiting time of 21 ms built in. If you now tap the touch sensor, the output flashes. What actually happens is that touching it applies a 50 Hz interference signal to the base of the Darlington transistor. Therefore we get a 50 Hz squarewave signal on the digital output, i.e. a cycle duration of 20 ms. If we now sample this signal using a slightly different frequency, you obtain a significantly smaller frequency. The cycle duration 21 ms indicates a sampling frequency of around 48 Hz. The difference 50 Hz – 48 Hz = 2 Hz appears in antiphase at outputs B2 and B5.

It may not be self-evident that we can control a transistor in this manner. Usually we specify a base current and determine the bias point in this way. With small signals, things would not work without base current. The person touching the sensor is seen as one plate of a capacitor, with the second plate formed by all the wires and cables located in the environment. A base current would charge this capacitor so far negative that the transistor would block completely. It works only because every bipolar transistor contains a zener diode in the range 7 V to 10 V between base and emitter (see **Figure 5**). The idle state mains hum voltage of a person is generally significantly higher than 10 V_{pp}. Therefore an alternating current flows, with its positive half-wave driving the transistor.

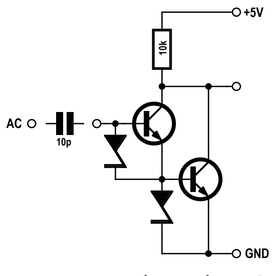


Figure 5. Contact sensor and touch switch (Touch2.bas).

You might expect that on the collector of the Darlington transistor we would also find an approximately equal squarewave signal. So let's also try coupling the analog output AO to the digital input A1. That's right; now the output flashes when the base is touched. However, the difference is that this time the output in idle mode is 'on', because the comparator on the sensor board inverts the signal.

It even works without the sensor board. At input A1 we have just a piece of insulated wire connected (**Figure 6**). If you then touch the outside of this insulation, it still flashes! This happens because the input of the controller has extremely high impedance. The wire, insulation and finger form a small coupling capacitor of a few picofarads in value. If an AC voltage is applied to the input, it is limited to the input voltage range by the internal protection diodes. The open input does admittedly have a disadvantage compared to the correct touch sensor. The idle state is not unambiguous, you see, which makes evaluation tricky.

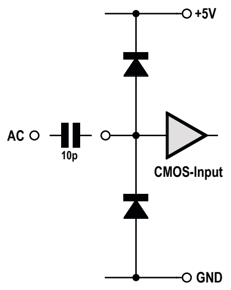


Figure 6. Input Port used as contact sensor.

Processing switching signals

Up to now all we have done with our touch sensors is achieve flashing on the output. In reality we'd expect a contact switch to do rather more, namely switching something on. This is entirely feasible if we write our program somewhat differently. To manage this, all we need do is make sure that the switching-on process takes precedence and switching-off takes place after a short delay. Our *Touch* programs (**Listing 5** and **Listing 6**) additionally employ a delay period of 10 µs and consequently interrogate the Port very frequently. Once a High state is detected, the program sets counter T to a value of 50000 and switches on the output. The counter decrements slowly and only after about 500 ms without any further pulse does the output switch off. Any further impulse that occurs will reset the counter High again and the timeout period is extended accordingly. Now everything functions as desired. The output goes on immediately if someone touches the sensor and goes off again, not when the contact ceases but after some delay. The function corresponds to a retriggerable monostable multivibrator.

Now our touch sensor, with its Darlington transistor, is working the way we want it to. The squarewave signal on the digital output has been transformed into an unambiguous switching signal that will work even in environments without mains hum. Short pulses caused by static charge are sufficient to trigger the output.

```
Dim T As Word

Config Portb = Output

Do
  If Pinc.1 = 1 Then T = 50000
  'If Pinc.1 = 0 Then T = 50000
  If T > 0 Then T = T - 1
  If T > 0 Then Portb.2 = 1 Else Portb.2 = 0
  If T > 0 Then Portb.5 = 0 Else Portb.5 = 1
  Waitus 10
Loop
```

Listing 5. Berührungssensor und Klopfschalter (Touch2.bas).

```
void loop() {
  if (digitalRead(input) == 1) timeout = 50000;
  //if (digitalRead(input) == 0) timeout = 50000;
  if (timeout > 0) timeout = timeout -1;
  if (timeout > 0) {
    digitalWrite (output1 , 1);
    digitalWrite (output2 , 0);
  }
  else {
    digitalWrite (output1 , 0);
    digitalWrite (output2 , 1);
  }
  delayMicroseconds(10);
}
```

Listing 6. Touch switch in Arduino-C.

There's a special reason why we chose the loop delay of 10 µs to be much smaller than necessary for a 50 Hz signal. The format of the program makes it equally suitable for use with the two sound sensors in the Sensor Kit, with which, according to the sound frequency being sampled, up to 10 kHz or more can appear on the output. Correct adjustment of the trimpot will give you a very usable speech switch that will also react to loud whistling, hooting or other noises. Even gentle tapping your finger on the microphone will set off the switch. The sensitivity depends very much on the precise setting of the pot. You need to keep in mind that this type of microphone normally delivers less than 1 mV. This means you must maintain the sensor voltage to the switching point within a few millivolts of accuracy in order for sound signals to make the comparator change level correctly.

Shock sensor

Certain other digital sensors, such as the *Tap Module*, are appropriate for this kind of sampling. This contains a spring that can close a contact when disturbed. Two very brief pulses are produced in this process. You need to be aware that the module has a pull-up resistor, meaning that the output voltage is +5 V when not operated and the pulses are 0 V. In this situation the sampling process needs to invert the input signal, which is marked in the software with a commented line. Alternatively in this case you might also connect +5 V and GND transposed and leave the software unaltered, as the polarity is of no interest to the switch and resistor.

The *Shock Sensor* was tested in the same way, in which an internal contact is closed for a short duration each time it is disturbed. The sensitivity of the Shock Sensor is greater than that of the Tap Module. The *Tilt Switch* (ball switch) can also be sampled in this manner. At the right degree of inclination a small ball rolls downwards and closes two contacts. If you shake the sensor rapidly to and fro, you can hear the ball rattling and watch the program triggering the output.

In point of fact all switching sensors can be polled or sampled meaningfully using this method, as well as press buttons (*Button*), reed switches (*Mini Switch*) and the reed switch on the comparator board. Using this type of time delay we can achieve effective debouncing of the switches. Just about every mechanical switch (apart from mercury switches, which are now banned in many territories, however, on account of the poisonous mercury in them) bounces momentarily one or more times when operated, generating several impulses to begin with. The software turns this into a single, elongated pulse. If you alter the period somewhat you can make a time switch, for instance to illuminate a staircase at night. The light can then be activated, according to the type of sensor, by loud speech, touch, tapping or advancing a magnet. Even the digital output of the temperature sensor could be polled in this way. This would simultaneously eliminate the problem of fluttering transitional states between Low and High.

```
void loop() {
  if (digitalRead(input) == 1) timeout = 50;
  if (timeout > 0) timeout = timeout -1;
  if (timeout > 0) {
    digitalWrite (output1 , 1);
    digitalWrite (output2 , 0);
  }
  else {
    digitalWrite (output1 , 0);
    digitalWrite (output2 , 1);
  }
  Serial.println(analogRead(sensorPin)+100*digitalRead (output1));
  delay (20);
}
```

Listing 7. Additional serial output.

With a minor alteration (**Listing 7**) the function of a comparator sensor could also be made clearer using the serial monitor. For this we additionally make use of the analog signal at AD2, while the digital output signal of the comparator carries on with the task of triggering the actual switching process. To display the output signal with only one channel, we raise the output in a High state by 100. You can then observe the original analog signal as if raised on a pedestal. **Figure 7** shows the result for the sound sensor. You can see clearly that a sound signal must first exceed a certain level to switch the output state. At the end of all of the signals the software enables the laid down delay period, after which the output drops back down.

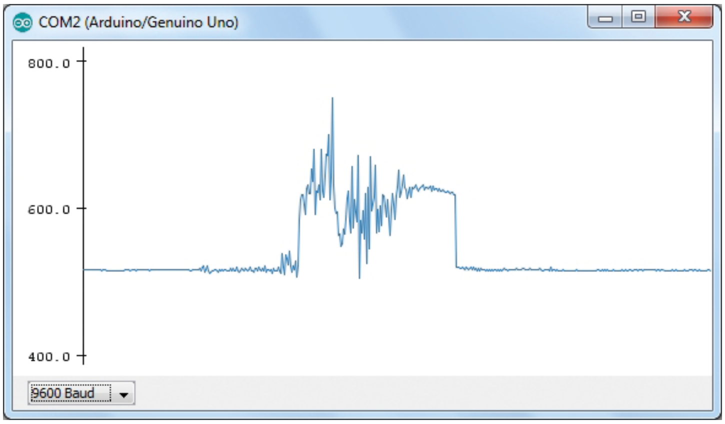


Figure 7. Switching using the sound sensor.

Figure 8 portrays the work of the temperature sensor. For this the program is slowed down to an extent by a delay of 100 ms. During the measurement time-window the sensor was twice warmed by touching with a finger. You can spot clearly the inversion performed by the comparator: a falling voltage on the actual sensor switches on the comparator's output. Equally easy to recognize is a brief phase of oscillation in the region around the switchover point. The software suppresses these oscillations effectively.

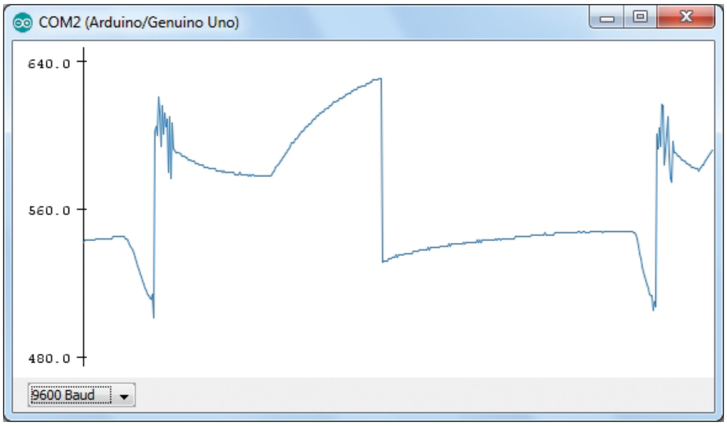


Figure 8. Temperature sensor in action.

Buzzers and other actuators

At output B2 we can again attach the relay board. But there are yet more actuators that we can also put to use. Among these belongs the *buzzer*, a small active device that is provided initially with some protective foil. At 5 V the current consumption amounts to 25 mA. The buzzer can therefore be attached direct to a Port. The polarity is admittedly not completely unambiguous and depends on how the buzzer is soldered onto the board. On the sample device the signal and minus connections were transposed in fact and the S-pin need to be connected to GND. A test using the lab power supply also indicated that the buzzer begins to work already at 0.7 V and at a different frequency if the sound opening is blocked. Without even opening the housing, our experts were able to figure what lay inside: a free-running oscillator with a bipolar silicon transistor with the normal threshold voltage of 0.5 V to 0.7 V (naturally it insists on having the correct polarity of supply voltage). At the same time take care that you do not confuse the active buzzer with the passive one, which is more comparable to a small 16-Ω loudspeaker.

Another interesting actuator is the color-changing LED (*Color Flash*), in actual fact a three-color LED package with a built-in controller. This automatic LED produces a color transition between red, green and blue. This is another case requiring a dropper resistor, since color-change LEDs of this kind are in fact designed for 3 V. Incidentally, although there is indeed a resistor of 10 kΩ on the board, it is connected in parallel (**Figure 9**). For 5 V a dropper resistor of 100 Ω would work best. Here again, however, the compromise presented in the previous article works again: you can connect the LED between two Port pins, in order to achieve a degree of current limitation using the internal resistance of the Ports. Pin 12 (B4) is recommended for the opposite pole.

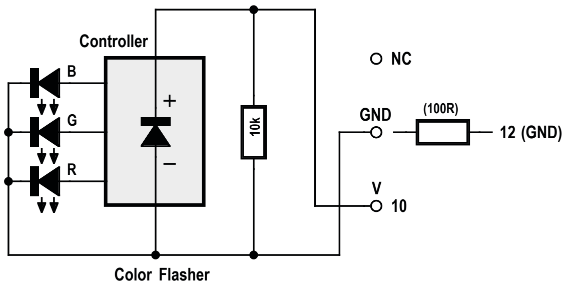


Figure 9. The controller of the color-change LED.

Incidentally, the internal controller of the color-change LED, like most ICs, includes an inverse polarity-protection diode on its supply voltage connections and therefore reacts sulkily if you reverse the supply connections. Without the dropper resistor a reversed power connection could lead to destruction. This is vitally important: GND lies in the middle of the three connections. By the way, just for fun, you can also connect the passive mini-speaker in series. Then you can not only see the switching process but also hear it.

Bypass capacitors

In many circuits we find capacitors wired between the supply voltage (V_{CC}) and ground (GND). They improve the stability of the circuitry and help prevent radio interference. Whenever electronic circuitry uses a lengthy cable to the power supply it's not just the resistance of the wires in the cable that comes into play but also the inductivity. A piece of twin-conductor cable 1 m long can have an inductance of around 0.5 µH, according to the gauge (thickness) of the wires and the distance between them. From this arises an inductive resistance of 3 Ω at 1 MHz or 30 Ω at 10 MHz. If your circuit has a varying current load, effectively you have an alternating current in the supply lead and with this some voltage drop. Typical problems with voltage drop of this kind are susceptibility to radio interference and vulnerability to failures in the passive elements of the circuit. The long (from a radio perspective) cable can also act as an antenna. Then you have radio-frequency (RF) signals radiating that potentially may exceed the permitted limits. Conversely pulses of interference may induce brief fluctuations in the supply voltage that might interfere with the correct functioning of a circuit.

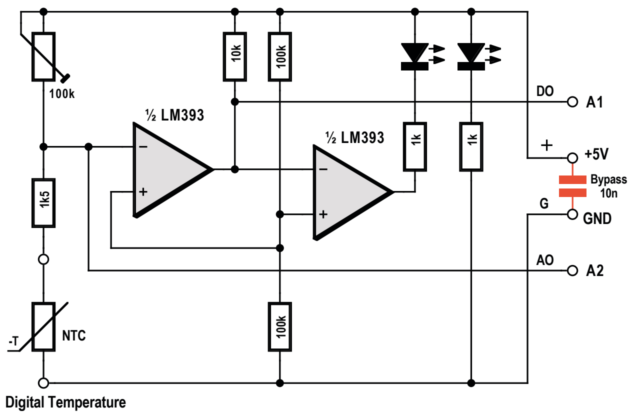


Figure 10. Color-changing LED.

and can lead to radio interference. This is why capacitors are connected direct to the supply terminals and are generally called anti-interference or suppressor capacitors. The DC motor in an RC model cannot function without these, as otherwise it would interfere with its own receiver onboard. Microcontrollers like the Arduino also require a capacitor between GND and V_{CC}, as otherwise they would not pass their test for electromagnetic compatibility. In this case they are usually called decoupling capacitors, because any radio-frequency currents can be diverted or decoupled by taking a shortcut through this capacitor mostly commonly of 100 nF. The older expression ‘blocking capacitor’ used for these components has fallen out of fashion.

Our sensors are all definitely not particularly susceptible to interference problems of this kind. If you do wish to employ a cable longer than 12 inches (30 cms), you should connect a bypass capacitor of 100 nF direct to the sensor between the supply terminals (see **image**).

Oscillators

One of Murphy’s Laws states that when you construct an oscillator it never oscillates, whereas when you make an amplifier it oscillates all the time. There’s something in this observation, which is why it’s always worth looking closely when your circuit turns into an oscillator and ‘starts to hoot’ as people say. An oscillator consists fundamentally of an amplifier and some matching feedback from the output to the input. The signal fed back must moreover have the correct phase relationship. If the voltage at the input rises directly, the same should happen at the output, only correspondingly amplified. Then it's sufficient to feed back a part of the output signal to the input using a capacitor — and now you have an oscillator.

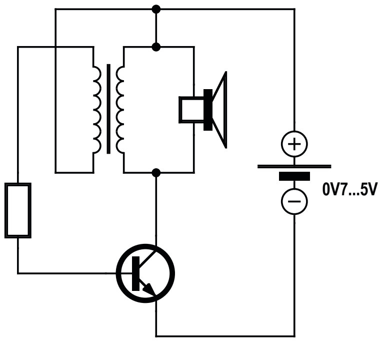


Figure 11. Oscillator with only one transistor.

An amplifier of this kind can be constructed with two transistors, in which both of them rotate the phase by 180 degrees. Or you can use an op-amp or an integrated loudspeaker amplifier.

A single amplifier stage with one transistor in grounded emitter mode turns the phase through 180 degrees. Rising input voltage will make the output voltage drop. All the same, you can build an oscillator with only one transistor. All you need do is ensure that the phase is reversed accordingly. You can do this, for instance, using several capacitors and resistors (phase-shift oscillator). Or use a transformer for the feedback (Meissner oscillator). If the oscillator still observes Murphy's Law and doesn't oscillate, all you normally need to do is reverse one of the two windings to get the phase right. So it's likely that the small buzzer is constructed in this way (**image**). The 'loudspeaker' has two windings, which at the same time make a transformer.

If you construct an amplifier with multiple stages and higher overall gain, it may actually be not that easy to prevent self-oscillation. Signals from the output can sneak back to the input via the supply voltage, which you may be able to avoid using a fairly large bypass capacitor. Or some small capacity may exist between the output and input wiring. In extreme cases the only remedy that can still solve this problem is a physical screening (shielding) plate.

Web Links
[1] www.elektor.com/arduino-sensor-kit
[2] www.elektormagazine.com/160152
[3] www.elektormagazine.com/160173

Chapter 48 • Sensors Make Sense (3)

For Arduino and more

Ever since the earliest days of telegraphy, electrical communication has been possible over a single wire. Today the 1-Wire Bus and similar protocols function using just one conductor, without the need for an additional clock line. Infrared (IR) remote controls operate on similar principles. In this session we get to grips with temperature and humidity sensors among other things, and in particular with the *IR Transmitter* and *Receiver* in Elektor's 35 Sensors Kit.

We commence with the temperature sensor *18B20 Temp* in the kit, which is available from the Elektor Store [1]. The DS18B20 under discussion may look like a regular transistor in a TO92 package but it's actually a complex IC embracing a temperature sensor and a special interface. The 1-Wire Bus was developed by the Dallas Semiconductor Corporation. One or more sensors can be handled using a single wire, if you discount the GND line. As this wire carries only data, it can also be used to power the IC. In the main, however, people use the third (V_{DD}) connection of the DS18B20 for supplying power, which then adds up to three wires in total. On the sensor PCB there is also an LED plus its dropper resistor, connected to the data line.

Arduino software for the 18B20

There are two Libraries that we need to load into the Arduino IDE: *OneWire* and *Dallas-Temperature*. Both are supplied on the Sensor Kit CD and they have to be copied into the *Libraries* folder in the *Arduino Sketchbook* directory. In each case you will find a header file **.h* and a C++ file **.cpp*. It's interesting to look closer at these files. If you manage to get all the way to the end, leaving nothing at all unread, that's pretty good going! After that, all you need do is integrate the header files into your own program and then call up a few functions. Thanks to Arduino, all this is straightforward. The program in **Listing 1** shows how to interrogate and retrieve a temperature pure and simple, repeated every 500 ms. The data line is connected to A2 (= PC2) (**Figure 1**). All the code mentioned in this article can of course be downloaded from the Elektor website [4].

```
#include <OneWire.h>
#include <DallasTemperature.h>
#include <LiquidCrystal.h>
#define ONE_WIRE_BUS A2
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

float temp;
int minTemp;
int maxTemp;
LiquidCrystal lcd(2,3,4,5,6,7);

void setup(void)
{
  Serial.begin(9600);
  sensors.begin();
  lcd.begin(16, 2);
  minTemp = 100;
  maxTemp = -100;
}

void loop(void)
{
  sensors.requestTemperatures();
  temp = sensors.getTempCByIndex(0);
  Serial.println(temp);
  lcd.setCursor(0, 0);
  lcd.print(temp);
  lcd.print (" C   ");
  if (temp < minTemp) minTemp = temp;
  if (temp > maxTemp) maxTemp = temp;
  lcd.setCursor(0, 1);
  lcd.print(minTemp);
  lcd.setCursor(5, 1);
  lcd.print(maxTemp);
  delay (500);
}
```

Listing 1. Temperature measurement using the DS18B20.

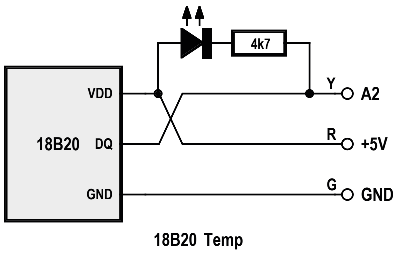
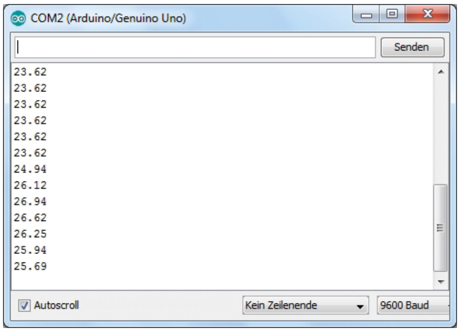


Figure 1. 18B20 temperature sensor connections.

Not only does the thermometer have a serial output but it can also indicate the temperature on the LCD screen, if you are using the Elektor Extension Shield [2] [5]. And because there is space on the second line of the display, we have additionally programmed a minimum/maximum thermometer.

The temperature is indicated as a real number to two decimal places. The actual resolution amounts to 0.06 degrees C per step. The absolute accuracy is given within 0.5 degrees. On the serial monitor (**Figure 2**) you can read the data. A touch on the sensor will demonstrate a change in temperature.



```
23.62
23.62
23.62
23.62
23.62
23.62
24.94
26.12
26.94
26.62
26.25
25.94
25.69
```

Figure 2. Temperature output in the serial monitor.

The serial plotter indicates changes in temperature over time (**Figure 3**). In the plot shown the sensor was touched (warmed) twice by finger. This shows clearly the differing time constants for heating and cooling. Also interesting is how the finger was visibly warmer on the second touch. Something must have caused the rise in temperature during the intervening time. Thanks to the high resolution of the sensor even small variations can be detected.

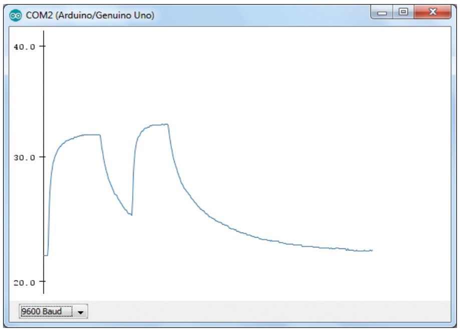


Figure 3. Temperature curve.

18B20 in Bascom

Bascom supports the 1-Wire Bus, although the Bus alone does not provide a complete solution for using the 18B20. So you need to dive a little deeper (into the data sheet) and do some programming of your own. Of course this takes some time but it does also open up some interesting opportunities. You can then make use of some special features of the sensor, for instance for altering the resolution or for reading out the unique ID number included in every IC.

The Bascom sample code (**Listing 2**) shows how we read out temperatures. Two commands need to be sent, (*Skip ROM, &HCC* and *Convert T, &H44*). Following a delay while the measurement itself is taken, we send the command *Read Scratchpad (&HBE)* and then read out the two Bytes. From these the software calculates a 16-Bit number and the temperature (in steps of .0625 degrees). That's pretty clever and even better, the program is only slightly longer than the Arduino version. Once again we have the bonus of a minimum/maximum thermometer with LCD readout.

```
'DS18B20LCD  AD2, PORTC.2
...
Do
  1wreset
  1wwrite &HCC
  1wwrite &H44
  Waitms 800
  1wreset
  1wwrite &HCC
  1wwrite &HBE
  Dat(1) = 1wread(2)
  1wreset
  Temp = 256 * Dat(2)
  Temp = Temp + Dat(1)
  Temp = Temp * 0.0625
  Print Temp
  Tempint = Round(temp)
  If Tempint > Maxtemp Then Maxtemp = Tempint
  If Tempint < Mintemp Then Mintemp = Tempint
  Locate 1 , 1
  Lcd Temp ; " C   "
  Locate 2 , 1
  Lcd Mintemp
  Locate 2 , 5
  Lcd Maxtemp
  Waitms 200
Loop

End
```

Listing 2. Temperature measurement in Bascom (excerpt).

Temperature and humidity using the DHT11

At first glance the combined humidity and temperature sensor DHT11 looks like a straightforward resistive humidity sensor. This impression is reinforced when you see that an analog Pin is recommended (**Figure 4**). In reality, however, the insignificant-looking exterior conceals a complex sensor with a digital interface.

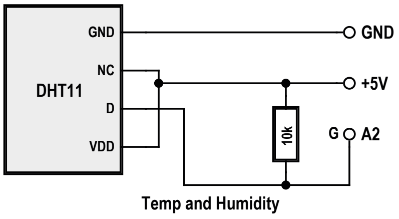


Figure 4. Temperature and humidity sensor DHT11 connections.

The Chinese company Aosong has come up here with a scheme that (only at first sight) brings to mind the 1-Wire Bus of Dallas Semiconductor. Each measurement is initiated by a Low pulse from the Master (controlling device) lasting at least 18 ms. After this a total of 40 Bits are read out, which the Master requests each time by means of an 80 µs long Low pulse. The sensor responds with High pulses, in which a period lasting 28 µs maximum stands for a zero and a period of 70 µs for a one. The 40 Bits then contain one High Byte and one Low Byte for the humidity and the temperature plus an additional parity Byte for checking correct transfer of the data. With the DHT11 the Low Bytes are always set to zero, meaning that no post-decimal point figures are transferred. However, there is also a DHT22 device using the same protocol, which does indeed handle the decimal places in addition. Consequently both types of sensor can be interrogated using the same software Library.

Once again we are fortunate that someone has already taken the trouble to re-format the complicated protocol into an Arduino Library. Using this is quite simple, once you have copied the Library directory *DHT* from the CD into the Arduino Library folder. The sample code (**Listing 3**) indicates how the two measured values are interrogated and can be displayed on the LCD screen. Once again we select Pin AD2 for connecting the data line.

```
//DHT11LCD, pin AD2

#include <dht.h>
#define dht_apin A2
#include <LiquidCrystal.h>
LiquidCrystal lcd(2,3,4,5,6,7);
float temperature;
float humidity;

dht DHT;

void setup(){
  Serial.begin(9600);
  delay(500);
  delay(1000);
  lcd.begin(16, 2);
}

void loop(){
   DHT.read11(dht_apin);
   humidity = DHT.humidity;
   temperature = DHT.temperature;
   Serial.print("Humidity = ");
   Serial.print(DHT.humidity);
   Serial.println(" %  ");
   Serial.print("Temperature = ");
   Serial.print(temperature);
   Serial.println(" C  ");
```

```
        lcd.setCursor(0, 0);
        lcd.print(temperature);
        lcd.print (" C   ");
        lcd.setCursor(0, 1);
        lcd.print(humidity);
        lcd.print (" %   ");
        delay(2000);
    }
```

Listing 3. DHT11 in Arduino-C.

On the datasheet the absolute accuracy for temperature measurements is stated modestly as within 2 degrees, that for air humidity as within 5 %. Ambient humidity is difficult to judge because all run-of-the-mill moisture meters are fairly imprecise. However, for temperature at least, the results measured on our sample device seemed really good. A digital thermometer that was to hand indicated 23.8 °C, the DHT11 showed 24 °C, and the DS18B20 read 23.37 °C, in each case leaving enough time for the sensor to settle properly..

DHT11 and Bascom

Many ready-to-use commands and functions exist in Bascom of course but the DHT11 is not supported directly as such. Consequently for most topics your most profitable approach is to search the Net to see if someone else has tackled them already. On this occasion our search delivered positive results in the Bascom Forum, where a user by the name of Grütze had written a program called *DHT11LCD.bas* that does exactly what we want. The code is easy to read and reflects more or less exactly what is said in the data sheet. Only minor adjustments are needed for using it with the Extension Shield and enabling the sensor to work using Pin AD2. Beyond this, we also incorporated a serial output, initially only for the humidity data, as it was intended that these would be mapped out using the serial plotter from the Arduino IDE (**Figure 5**).

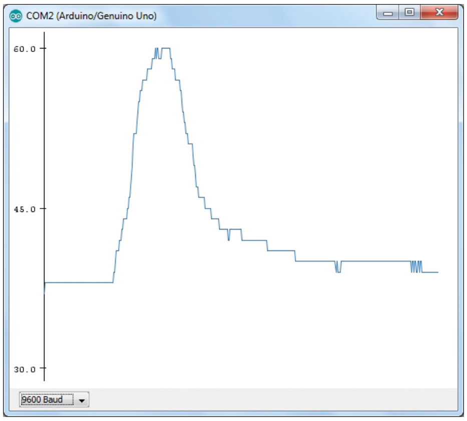


Figure 5. Measuring ambient humidity by finger touch.

The significant activity takes place in the function *Get_dht11()*. This is where we carry out exactly the same processes that exist in the corresponding Library for the Arduino. In this way we can create a complete miniature weather station equally well using Bascom (**Listing 4**).

```
'DHT11LCD an AD2, PORTC.2
...
Do
   If Get_dht11() = 1 Then
      Print Humidity
      Locate 1 , 1
      Lcd "H: " ; Humidity ; " %  "
      Locate 2 , 1
      Lcd "T: " ; Temperature ; " C  "
    End If
    Waitms 1000
Loop
End
```

Listing 4. Using the DHT11 in Bascom (excerpt).

Infrared remote control

We are all familiar with infrared remote controls or ‘zappers’ for TVs and other home entertainment equipment. Numerous different manufacturers and protocols exist that are not compatible with one another, meaning that a remote control handset must be a correct match for the equipment it commands. One feature common to all designs is that the signal sent by the infrared transmit diode is modulated with a frequency between 30 and 40 kHz. This signal is then pulsed (in one of a number of methods) so as to transmit individual packets of data. An integrated infrared receiver detects the signals with a photo diode, amplifies and filters them and demodulates them back into a digital signal. By design the internal filters are dimensioned for specific frequencies in the range 30 to 40 kHz, although the bandwidth is sufficient to also receive ‘off-frequency’ signals at shorter ranges.

You can also use an IR zapper for other, more general, remote control or switching tasks. The Sensor Kit includes an IR transmit diode or emitter (*IR emission* in the diagram) and an integrated IR Receiver (**Figure 6**). In conjunction with an Arduino you have the choice of receiving or transmitting IR signals (or both!). Here we will demonstrate a program that can do both. Two buttons are used to send commands that are evaluated in the receiver in order to switch an output. The same assignment is performed both in Bascom and in Arduino-C. Both programs are sufficiently compatible to the extent that that the Bascom controller can tell the Arduino-programmed Uno what is to be switched and vice versa.

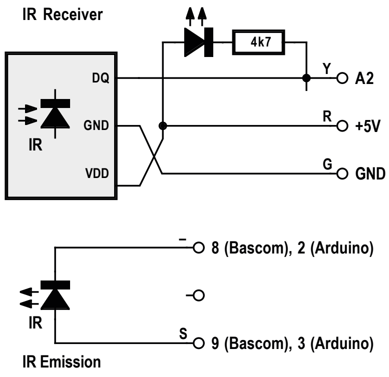


Figure 6. IR receiver and transmitter.

A standard frequently used for IR remote controls is RC-5, developed by Philips (see boxout panel). Simple commands for this are provided in Bascom, making decoding zappers a simple task. **Listing 5** provides a simple RC-5 receiver and transmitter in Bascom. All received data is displayed on the LCD screen. The command is also sent over the serial interface. At the same time a check is made for any output to switch to Port B. Given that LED2 is available at B.2 on the Extension Shield, we have designed the code to illuminate this with button 2 on the zappers and switch it off with button 0.

The RC-5 receiver can be connected to any input Pin you choose. So we have again deployed input PC3 (AD3) for this. For the command *Getrc5* Bascom uses the *Timer0* Interrupt in the background, which you must enable globally. In addition we have switched in the internal pull-up resistor for input PC3. It is then possible to use the controller for transmitting (only) without an IR receiver connected. Had we not done this, an open high-impedance input without a pull-up might otherwise assume a Low state and cause the program to hang.

```
'RC5LCD  In AD2, PORTC.2, Out OC1A, PORTB1
...
Do
  If S2 = 0 Then
    Togbit = 0 : Address = 0 : Command = 2
    Do
      Rc5send Togbit , Address , Command
      Waitms 100
    Loop Until S2 = 1
  End If
  If S1 = 0 Then
     Togbit = 0 : Address = 0 : Command = 0
     Do
      Rc5send Togbit , Address , Command
      Waitms 100
    Loop Until S1 = 1
  End If
```

```
    Getrc5(address , Command)
    If Address < 255 Then
      Locate 1 , 1
      Lcd Address ; "   "
      Locate 2 , 1
      Togbit = Command / 128
      Lcd Togbit ; " "
      Locate 2 , 5
      Command = Command And &B01111111
      Lcd Command ; "   "
      Print Command
      If Command = 2 Then Portb.2 = 1
      If Command = 0 Then Portb.2 = 0
    End If
  Loop
```

Listing 5. RC-5 transmitter and receiver in Bascom.

For the transmit output (command *Sendrc5* in Bascom) we normally use PB1 (Arduino Pin 9), because the output OC1A of Timer 1 is connected to this Pin, used for generating the 36 kHz transmit signal (warning: in the Arduino-C++ software a different output is used for this task, meaning that on this occasion we cannot keep the same hook-up allocations).

In Bascom the standby state of this Portpin must be defined in advance using a Port command; here it needs to be Low during inactive intervals. For each pulse packet the software then switches over from the Port to the timer output. Because the IR diode has no series resistor, it makes sense to switch this to a different output Port, in order to use the two internal resistors for current limiting. Here we selected PB0, which is also switched Low as an output. In that way we still have four outputs on Port B as potential switching outputs for received data.

In transmit mode two commands are supported. If you press button S1 on the Extension Shield, then the code for button 2 on the remote control is transmitted. This switches on the output at the receiver, making LED2 come on. With button S2 you send a zero and switch off the output at the receiver once more.

Arduino and IR

For programming in Arduino-C we need to install the *IRremote* Library (**Listing 6)**. This makes use of Timer2 and its output OC2B (PD3, Arduino Pin 3) for generating pulses applied to the IR LED. This is also the E wire of the LCD screen on the Extension Shield. That unfortunately means the program cannot interact simultaneously with the LCD. However, that's not totally bad news, as we still have the serial outputs for viewing the received data.

```
#include <IRremote.h>
int RECV_PIN = A2;
IRrecv irrecv(RECV_PIN);
IRsend irsend;
```

```
decode_results results;
int d;
int S1 = A0;
int S2 = A1;
int LED = 13;
int kathode =2;
void setup()
{
  Serial.begin(9600);
  irrecv.enableIRIn();
  pinMode(S1, INPUT_PULLUP);
  pinMode(S2, INPUT_PULLUP);
  pinMode(RECV_PIN, INPUT_PULLUP);
  pinMode(LED, OUTPUT);
  pinMode(kathode, OUTPUT);
}

void loop() {
  if (irrecv.decode(&results)) {
    Serial.println(results.decode_type);
    Serial.println(results.value, HEX);
    d = results.value & 15;
    Serial.println(d);
    if (d==2) digitalWrite(LED,1);
    if (d==0) digitalWrite(LED,0);
  }
  if (digitalRead(S1) == 0) irsend.sendRC5(0x382, 32);
  if (digitalRead(S2) == 0) irsend.sendRC5(0x380, 32);
  irrecv.enableIRIn();
  delay(100);
}
```

Listing 6. IR control in Arduino-C.

The input is a matter of choice and is assigned here to A2 once more. Programming an input pull-up is overwritten by the Library, rendering this ineffective. But it's not even necessary, as reception does not block the program even when an open input is in the Low state. Our goal remains achievable, to use a program without modification or reconfiguration for a choice of receiving only, transmitting only or handling both tasks.

The crucial advantage of the *IRremote* Library is that it can 'speak' not only RC-5 but also a multiplicity of other standards. Most of us have a whole collection of disparate remote controls at home. If so, it makes good sense to point each of these once at the IR Receiver. You will then get a report of the standard employed and the data received. RC-5 is shown as Type 3. If you press several times on button 2 of an RC-5-compatible remote control, the following messages appear:

```
3
382
3
B82
```

The data is output as 12-Bit numbers. The lower 5 Bits denote the key code. Next comes the 5-Bit device address, in this case the address 14 for a DVBT receiver. The highest value Bit is the Toggle Bit, which changes with every key press and does not need to be evaluated. It's worth noting down this information, so you can resend it identically. You can ignore the Toggle Bit when doing this. If you transmit 382_{hex} via the IR diode, this corresponds to button *2* in RC-5 code.

Because in this situation the Arduino has to get by without the Extension Shield, the switching output is assigned to connector 13. Doing this enables us to control the LED on the Arduino. The relay can now be connected to Portpin 13 and straightaway a load can be switched either with an RC-5 remote control or else by a second Arduino with an IR diode.

Here too the pressbutton commands 2 and 0 can be transmitted back in the reverse direction. As in the Bascom version, the relevant buttons are assigned to A0 and A1, in which the internal pull-ups have been enabled. Here you can hook up plenty of the things that the sensor kit has in store. Of course these do not have to be touch switches every time. Why not use a magnetic sensor, a position sensor or an optical sensor? You could turn an infinitely large number of ideas into reality. The TV could be turned off by sunrise at the latest, or make the position sensor on the door handle recognize when somebody enters the room and then switch on the lights and the radio to welcome them!

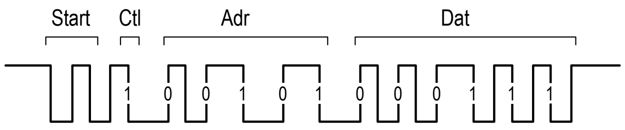
The 1-Wire protocol

With the 1-Wire Bus everything is channeled down the single wire DQ. However, we also have GND and V_{CC} wires. In standby mode the data line DQ is taken High with a pull-up resistor. The Master (Controller) can now send a Reset pulse for initiating communication with Slaves, in order to then send commands or receive data. Both partners can load data onto the Bus. A 0 Bit is represented by a 15 µs long Low pulse and a subsequent, 45 µs long High state. In contrast a 1 Bit is symbolized by a 60 µs long Low pulse. Between individual Bits there is a resting state, during which the data line is taken High by the pull-up.

The Master always sends a Reset followed by one or more commands. A sensor chip then responds with the wanted data. If you study the data sheet for the chips in detail, you will find information there not only on the general Bus protocol but also countless commands and the makeup of the data structure that is repeated back. Matters become even more complex, because multiple Slaves can be attached to the same Bus. You can hardly imagine how much work it would take to program all of this yourself. Thank goodness you don't have to reinvent the wheel every time and can refer to ready-made code almost always.

The RC-5 protocol

Infrared remote controls for TV receivers, video recorders and other home entertainment devices operate in part using the RC-5 standard defined by Philips. This employs modulated optical signals in the range 30 kHz to around 40 kHz. The remote control sends individual bursts (packets of data pulses) 0.888 ms or 1.776 ms in length. At a modulation frequency of 36 kHz a short burst contains 32 individual pulses and a long one 64. The complete data packet lasts about 25 ms and is repeated every 100 ms for as long as a button is pressed.



The protocol employs a bi-phase signal. A Bit has a length of 1.776 ms. If the 36 kHz pulse lies in the first half of this time period, it represents a logical Zero; a logical One is signaled by a pulse in the second half. The signal is introduced every time with a start sequence that never changes. There then follow three data fields:

- The Control or Check Bit (Ctl) alternates between 0 and 1 with every key press. In this way the receiver can differentiate whether a key has been pressed once for a long time or several times briefly.

- The Device or System Address Bits (Adr) comprise 5 Bits, in which the high-value Bits are transferred first. Common device addresses are 0 for TV sets and 5 for video recorders. In this way several remote controls can be deployed in the same room.

- The Data or Command Bits (Dat) comprise 6 Bits for up to 64 differing keys (pressbuttons). The number keys 0 to 9 generate codes from 0 to 9. Here too the highest value Bits are sent first.

Web Links

[1] www.elektor.com/arduino-sensor-kit
[2] www.elektormagazine.com/160152
[3] www.elektormagazine.com/160173
[4] www.elektormagazine.com/160210
[5] www.elektormagazine.com/140009

Chapter 49 • Sensors Make Sense (4)

For Arduino and more
Using optics for switching and control is a widely employed and accepted technique, with many favorable features. No mechanical movement, no wear and tear on switch contacts, modules are electrically isolated and much more.

This time our subject is optical switches and their applications, from fork sensors to reflex light barriers and pulse detectors. We'll cover the best methods of interpreting and analyzing the signals too. Once again we are using sensors from the 35 Sensors Kit, available direct from Elektor [1].

As always, all sample programs and listings can be downloaded from the web page for this article [5].

The tracking sensor
We're all familiar with little robots on two wheels that can follow a white line. These assess work by assessing the reflectivity of the surface below them. An IR transmit diode illuminates the surface beneath them and an IR phototransistor detects the reflected light. In basic principle terms this uses what is called a reflex (or reflected) light barrier. To adjust for optimum sensitivity for differing surfaces and clearances the tracking sensor uses a potentiometer (pot) and a comparator (**Figure 1**).

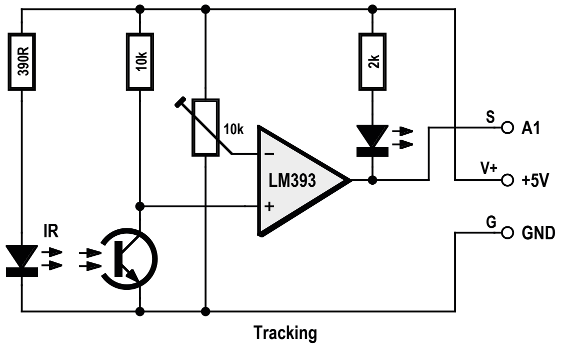


Figure 1. Adjustable reflected light barrier.

Thanks to the LED on the output of the comparator we can easily test its functionality. Your finger will provide the reflective surface if you hold it a centimeter away from the sensor. You can now adjust the pot to get the best switchover point.

A digital signal will appear on the output and this can be analyzed and processed in the usual way. You can either use the software described in part 2 of this series [3] or you can connect an actuator direct. What you do with this after that is left to your own imagination! It doesn't have to be a robot even. Perhaps you need a light or a fan that is switched on by a hand gesture, or you may have some other requirement.

Optical fork sensors

In many machines and devices (from scanners to 3-D printers) you will find limit switches for determining the position of a moveable component. Mechanical contacts or microswitches often suffer problems concerning their long-term stability and need to be replaced after fairly long periods of use. A fork sensor gives better service because the IR diode and phototransistor used display no signs of ageing, al least with moderate LED current.

A transmit diode and phototransistor (**Figure 2**) are positioned close together. The fork-shaped housing contains a slot, in or through which you can place or pass an object that is opaque to light. The on/off switching transition achieved is easily reproducible with an accuracy of less than a millimeter. Fork sensors of this kind are used also in computer mouse devices and demonstrate their reliability day in and day out.

The output signal at the collector of the phototransistor can be polled either analogly or digitally, according to your choice. Exactly what is done with the signal then depends on the particular assignment and the software. Your starting point can be with an accuracy of less than 1 mm — possibly exactly this, as required for a 3-D printer. In less critical cases a purely digital (yes or no) query will suffice: light or no light.

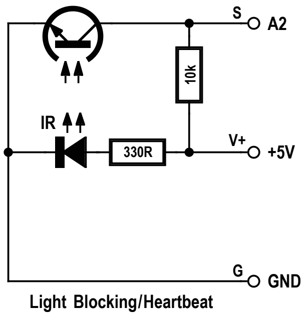


Figure 2. Fork sensor.

The pulse sensor

Heartbeat monitors and other pulse meters frequently use an optical procedure. Light of an appropriate wavelength is passed through a finger or an earlobe and then received using a phototransistor. Part of the light is absorbed in the blood inside the body. The heartbeat varies the momentary blood flow periodically and consequently modulates the light stream to a small degree. Naturally these signals must be evaluated analogly, as we are not expecting any major variations.

The heartbeat sensor is constructed around a fork sensor (see **Figure 2**), but with a greater distance between transmitter and receiver. Both the IR transmit diode and the phototransistor have long connecting leads, enabling you to adjust the optimum position. The PCB is identical to that used for the fork sensor, except that the connector and components are fixed on the other side.

In use you will need to bend the IR diode and the phototransistor so that a finger can be clamped between them both. It's also vital that the finger cannot touch any conductive area on the PCB, because additional mains (AC line) hum interference could be picked up in this way. On our example the sensors were flexed as seen in **Figure 3**. The IR LED then shines almost directly onto the phototransistor. The output voltage is now close to zero. If you next lay your finger in the gap, the light is shaded. The phototransistor conducts less, meaning that the voltage rises. For longer tests it makes sense to hold the sensor onto the finger with a rubber band (**Figure 4**). It is, you see, important that the finger is relaxed and the signal is not falsified by varying pressure.

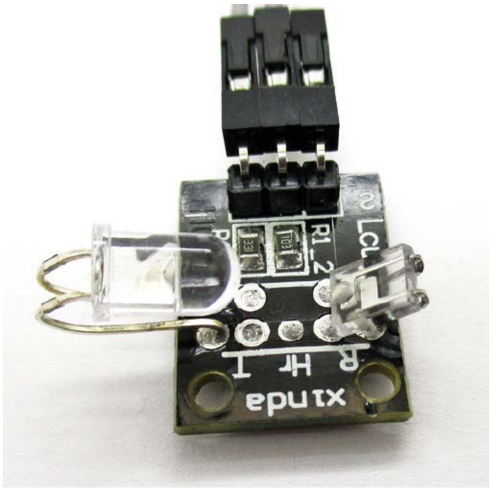


Figure 3. Assembled pulse sensor.

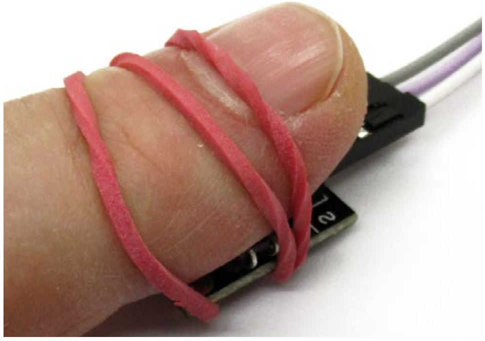


Figure 4. The pulse sensor in action.

For your first test with the Arduino IDE the program VoltageAD2 from part 1 of the series [2] will be fine. The voltage on the phototransistor can now be represented using the serial plotter (**Figure 5**). The light modulation caused by the pulse is clearly visible. Overlaid on the signal are slower variations, caused by unavoidable movement. This explains why evaluating the signals can be a challenge.

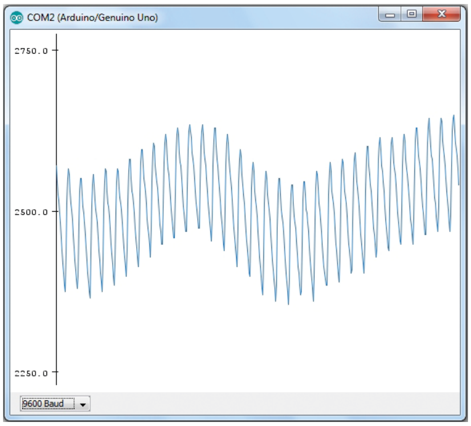


Figure 5. Voltage on the pulse sensor.

To improve the interpretation we first used a lowpass filter (**Listing 1**). This simply produces a moving average of the value measured. For this we need nothing more than a couple of lines of code. Each of the current values measured is added to a sum total of average values mean that is afterwards reduced by 1/20. Each individual measurement can then influence the average by only 5 %. The time constant of the lowpass filter is therefore the sensing period times 20, in this case 0.4 s. The sampling rate is fixed at 50 Hz in order to attenuate any interference signals from the AC line current (of course you would use 60 Hz where this is the norm). The cutoff frequency of the filter is f = 1 / (2 Pi T), in other words around 0.398 Hz. For illustration purposes the unfiltered and filtered measurements are always shown alternately.

```
`//VoltageAD2 0...1023 at AD2 filter
int sensorPin = 2;
int value;
int mean;

void setup() {
  Serial.begin(9600);
  lcd.begin(16, 2);
}
void loop() {
  value = analogRead(sensorPin);
  mean = mean - mean / 20;
  mean = mean + value;
  Serial.println(value);
  Serial.println(mean / 20);
  delay(19);
}
```

Listing 1. Generating an average.

On the serial plotter you can now see how the filter works. The moving average follows the signal with a degree of inertia (**Figure 6**). If you remove your finger rapidly from the sensor, you can observe the pulse response of the lowpass filters (**Figure 7**).

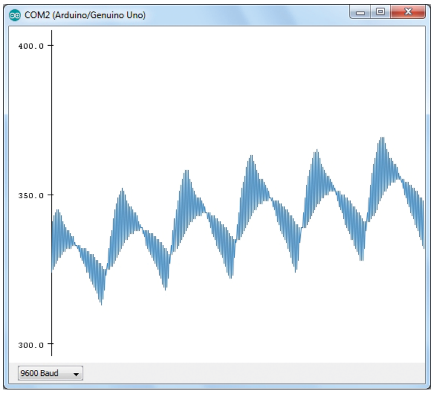


Figure 6. Original and average values.

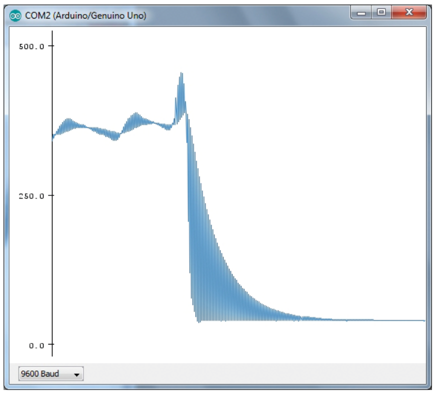


Figure 7. Pulse response.

All that remains is to generate the difference (input signal minus the average) (**Listing 2**). You do this by creating a highpass filter and then obtain a clean pulse signal that is rid of all the sluggish variations. With this done you can now set about evaluating the pulse frequency.

```
//Filter2AD2 0...1023 at AD2 filter
int sensorPin = 2;
int value;
int mean;

void setup() {
  Serial.begin(9600);
}
void loop() {
  value = analogRead(sensorPin);
  mean = mean - mean / 20;
  mean = mean + value;
  value = value - mean / 20;
  Serial.println(value);
  delay(19);
}
```

Listing 2. Highpass filter.

As well as this you can now recognize some further details. The signal always rises steeply and drops back more gradually (**Figure 8**). You can see how, with every beat, the heart pumps blood at high pressure through the arteries into the finger, from which the return flow through the veins takes longer. You can also spot easily any irregularities that may occur in the pulse rhythm.

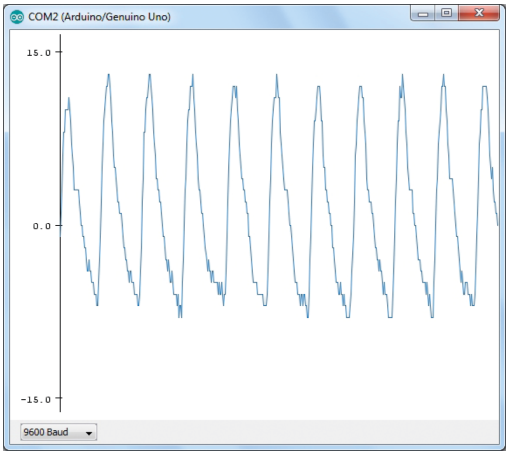


Figure 8. Filtered pulse signals.

To make a complete pulse meter device out of this now we need to measure the intervals between the positive flanks (leading edges). The Arduino has a simple timekeeping function by the name of millis(). This returns the time elapsed since the last Reset operation in milliseconds. In this way you can measure the interval between two heartbeats. The program (**Listing 3**) indicates the pulse time in ms and the pulse rate in beats per minute on the LCD of the Elektor Extension Shield [6]. The frequency is calculated from the pulse duration for each individual heartbeat, meaning that you don't have to spend long waiting. It does nevertheless happen that a pulse can go missing, even after slight movement, which can then lead to double or threefold pulse timings. This is why measurements are output only when the pulse frequency is greater than 45 beats per minute.

```
void loop() {
  value = analogRead(sensorPin);
  mean = mean - mean / 20;
  mean = mean + value;
  value = value - mean / 20;
  if ((old < 0) & (value > 0)) {
    time2 = millis();
    pulseTime = time2-time1;
    time1 = time2;
    //Serial.println(pulseTime);
    n = n + 1;
    pulseFreq = 60000 / pulseTime;
    if (pulseFreq > 45) {
      Serial.println(pulseFreq);
      lcd.setCursor(0, 0);
      lcd.print(pulseTime);
      lcd.print(" ms  ");
      lcd.setCursor(0, 1);
      lcd.print(pulseFreq);
      lcd.print(" /min  ");
    }
  }
  old = value;
  delay(20);
}
```

Listing 3. The Arduino pulse meter..

Measuring pulses with Bascom
In Bascom you have to sort out time measurement on your own because there is no permanent timekeeping process running in the background, only whatever you arrange yourself. A Timer Interrupt works well for this. Here we are using Timer 0 for the timekeeping. The Interrupt routine is invoked every 10 ms, meaning that you can assess time to this level of resolution.

With Interrupts comes precision. In this case it means that that the voltage measurements are also carried out by the Interrupt. This has the advantage that they take place in extremely accurate 10 ms tempo, eliminating any potential AC line frequency (50 or 60 Hz) interference.

For an ATmega 10 ms represent a long time and that means the entire analysis can be carried out in the (**Listing 4**). As in the Arduino model above, this involves generating the moving average, identifying the pulse edges (flanks), eliminating measurement errors caused by missing pulses and outputting to the Terminal and the LCD. The end product is a Bascom pulse meter with extremely similar characteristics to those of the Arduino project. Anyone who has battled with a computer mouse of the older design will have encountered the type of light barrier used for measuring rotation. The crucial factor is that you can detect not only impulses but also (using two light barriers) the direction of rotation. The two light barriers are arranged so that rotation produces squarewave pulses offset in phase by 90 degrees.

```
'--------------------------------
'Pulse.BAS  ADC2, LCD
'--------------------------------

...

Config Timer0 = Timer , Prescale = 1024
On Ovf0 Tim0_isr
Enable Timer0
Enable Interrupts

...

Do
Loop
```

```
Tim0_isr:
  Timer0 = 100    '10ms
  D = Getadc(2)
  Du = U / 40
  U = U - Du
  U = U + D
  Ticks = Ticks + 1
  U1 = D - Du
  If U1 > 0 And U2 < 0 Then
  T = Ticks * 10
     F = 60000 / T
     Ticks = 0
     If F > 45 Then
        Print T
        Print F
        Locate 1 , 1
        Lcd T
        Lcd " ms "
        Locate 2 , 1
        Lcd F
        Lcd " /min "
     End If
  End If
  U2 = U1
Return
```

Listing 4. Bascom pulse meter.

The same signals are produced by the rotary encoder in the Sensor Kit (**Figure 9**). However, instead of using light barriers, the encoder employs mechanical contacts. It also has clearly detectable mechanical detents, enabling you to feel, track and count each step made. A complete rotation covers 20 steps by the way. Twenty may not sound plentiful but since there is no physical end-stop, you could select, say, hundreds or thousands of steps, which would be very practical for setting a frequency or voltage precisely. A pressbutton switch is also included, although we do not use this here.

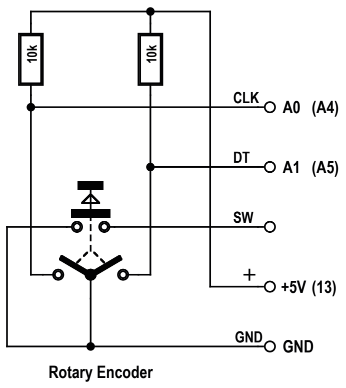


Figure 9. Rotary encoder.

Although both contacts are in fact equivalent, one of them is designated the Clock Pin (CLK) and the other is the Data Pin (DT). **Figure 10** shows the output signals for one rotation. Evaluating them is really easy: you wait for a trailing edge at CLK and then examine DT in order to decide whether to increment or decrement the counter status accordingly.

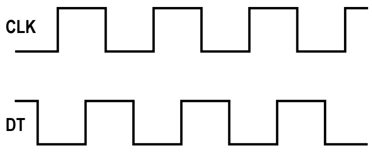


Figure 10. Switching signals during rotation.

The sample program (**Listing 5**) employs two encoders, enabling us to have two analog output signals PWM1 and PWM2 on Arduino outputs 9 and 10. With an additional lowpass filter you could then have a dual adjustable voltage source for all-purpose measurement use. You could also use this for creating color mixes using the red/green LED in the Sensor Kit with absolute sensitivity and reproducibility. The preset values can additionally be output in serial format and indicated on the LCD.

```
//Encoder  A0/A1 PMW1, A4/A5 PWM2

...

int clk1 = A0;
int dt1 = A1;
int clk2 = A4;
int dt2 = A5;
int pwm1 = 9;
int pwm2 = 10;

...
```

```
void loop() {
  new1 =  digitalRead(clk1);
  if((new1==0) & (old1==1)){
    if (digitalRead(dt1)==0)  d1++; else  d1--;
    Serial.println (d1);
    if (d1 > 250) d1 = 250;
    if (d1 < 0) d1 = 0;
    analogWrite (pwm1, d1);
    lcd.setCursor(0, 0);
    lcd.print(d1 * 20);
    lcd.print(" mV  ");
  }
  old1=new1;

  new2 =  digitalRead(clk2);
  if((new2==0) & (old2==1)){
    if (digitalRead(dt2)==0)  d2++; else  d2--;
    Serial.println (d2);
    if (d2 > 250) d2 = 250;
    if (d2 < 0) d2 = 0;
    analogWrite (pwm2, d2);
    lcd.setCursor(0, 1);
    lcd.print(d2 * 20);
    lcd.print(" mV  ");
  }
  old2=new2;
}
```

Listing 5. Encoder evaluation in Arduino C.

The rotary encoder includes its own pullup resistors. At the same time internal pullups are enabled on the Arduino. So using the same program you could use either one or two encoders. The first is connected to inputs A0 and A1. Purely by coincidence we have here pressbutton switches S1 and S2 on the Extension Shield. Because all contacts are at GND potential no conflict arises. But in an emergency you could transpose the first output value d1 with the pressbuttons. S1 would then provide the Clock signal and S2 the direction of rotation.

We connect the second encoder to A4 and A5. On the Arduino the corresponding Portpins are brought out twice, the second time diagonally opposite as SDA and SCL of the I^{2}C interface. Nearby there is also a GND Pin. What is missing, however, is an additional 5 V connection, which we need for the pullups on the encoder. No problem, however, as we switch Pin 13 HIGH, to which the Arduino LED is also connected. This is always a solution: whenever you are short of a GND or VCC connection, just program a Portpin for this purpose.

Encoding in Bascom

The Bascom version of the program is constructed almost identically. You can switch between languages by mutually accepting the source code, retaining the Variable names, and revising only the specific syntax properties. In the end result there is only one real difference: in Bascom you normally use the two PWM outputs of Timer1 with 10-bit resolution, meaning you now have 1023 steps and a step width of around 5 mV.

```
'Encoder.BAS    C0/C1 PMW1a, C4/C5 PWM1b

...

Do
  New1 = Pinc.0
  If New1 = 0 And Old1 = 1 Then
    If Pinc.1 = 0 Then D1 = D1 + 1 Else D1 = D1 -1
    If D1 > 1023 Then D1 = 1023
    If D1 < 0 Then D1 = 0
    Pwm1a = D1
    Print D1
    Locate 1 , 1
    Lcd D1
  End If
  Old1 = New1

  New2 = Pinc.4
  If New2 = 0 And Old2 = 1 Then
    If Pinc.5 = 0 Then D2 = D2 + 1 Else D2 = D2 -1
    If D2 > 1023 Then D2 = 1023
    If D2 < 0 Then D2 = 0
    Pwm1b = D2
    Print D2
    Locate 2 , 1
    Lcd D2
  End If
  Old2 = New2
Loop
```

Listing 6. Encoder evaluation in Bascom.

Two precisely adjustable PWM outputs, which almost scream out for an experiment on the theme of bridge output stages, especially since the two PWM signals originate from the same timer and have precisely matching pulses with synchronized leading edges. A green and a red LED can be connected in anti-parallel between the two outputs (**Figure 11**). If you provide a middle-ranging voltage, like 2500 mV for example, both LEDs remain dark. If one channel is out of balance, one of the LEDs starts to light up.

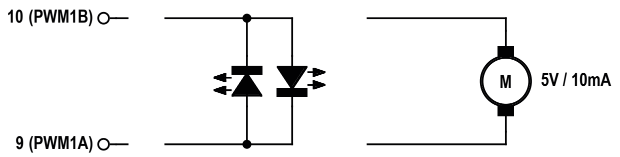


Figure 11. Bridge output stages.

Likewise one could connect an extremely frugal (low current) DC motor (cassette recorder type, starting current 10 mA) and then control the direction and rotational speed. Yes, it's true: actually you should not do this without a proper motor driver circuit. But it does work in fact. There is no danger of induced voltages in this case, because the outputs are always in a low-impedance state. Except perhaps when you reset the Arduino during operation, because then the connections are high-impedance. So you are better off making only brief tests and removing the motor again for as long as the program is still running...

Web Links

[1] www.elektor.com/arduino-sensor-kit
[2] www.elektormagazine.com/160152
[3] www.elektormagazine.com/160173
[4] www.elektormagazine.com/160210
[5] www.elektormagazine.com/160302
[6] www.elektormagazine.com/140009

Chapter 50 • A Beginner's Guide to Microcontroller Development Kits

The first step is the... easiest!

Too many electronics enthusiasts harbour a certain reluctance to enter the world of microcontrollers and programming. It really is not so difficult, as we will show by introducing four low-cost boards. You will see the setup and how some simple demo programs can be run so that early success is guaranteed. To get things going we gave a low-cost Arduino board to Burkhard Kainka who is one of our long-time contributors to Elektor and a specialist in many fields of electronics. His son Fabian will be giving us the lowdown on the NodeMCU board — a platform designed especially for IoT applications.

Too many electronics enthusiasts harbour a certain reluctance to enter the world of microcontrollers and programming. One reason is that there is just too much choice — faced with the jungle of competing development platforms — it is difficult to find the kit that's right for you. Not too complicated, well supported by third-party manufacturers in both hardware and software, and with a supportive community of users who welcome newbies.

We did the searching for you and picked out four controller-boards/kits suitable for beginners and gave them to some electronics wizards to unpack and put through their paces. You can read their impressions; it might help you to discover that your first steps into the world of microcontrollers may not be so daunting after all!

We start off in this issue with the father and son team of Burkhard and Fabian Kainka. Burkhard is already a long time contributor to Elektor, responsible for many popular articles, projects and books ranging in subject from RF technology to microcontrollers. In this issue he is taking a close look at an Arduino nano clone.

His son Fabian is following in his father's footsteps and has specialised in developments and publications on the topic of the Internet of Things. The microcontroller kit he will be looking at is not unrelated to this topic: the NodeMCU Kit.

In the next edition, an ESP32 Wi-Fi/Bluetooth board with an integrated OLED display will come under the spotlight.

JOY-iT Nano V3

The JOY-iT Nano V3 is an Arduino Nano compatible clone from JOY-iT, on sale in the Elektor-Store at just under under 12 Euro. A useful addition to this board is the Arduino Supplement Kit which includes additional components to enable a quick and easy start to microcontroller board experimenting.

These days a spare Arduino board in the lab is an essential resource to quickly prototype new ideas using its powerful processor, versatile connectivity and simple software development environment. I usually use the Standard Model Arduino Uno, the Nano is much smaller but almost the same hardware based around an ATmega328. This means there's lots of storage space for big projects and more computing power than you are likely to need.

First impressions
At first glance there does not seem to be too much difference between an original Arduino Nano and the Joy-It Nano clone (**Figure 1**), that's before you flip the board over. Underneath you will see an FT232R USB to UART converter chip on the original board and a CH340G on the clone. Both do the same job and convert ATmega serial interface signals into signals to and from the USB port. When plugged into a PC it creates a virtual serial interface so that the board can communicate with the PC. It assigns a COM port number automatically, for example, it uses COM2, COM3, or COM99, depending on how many other devices have previously used a serial interface on the PC. For a FT232R, new COM numbers are constantly being created. The CH340 is different; the next board will appear with the same COM port, which you can rename if you want to use multiple boards simultaneously. If a problem occurs with the USB driver on a computer, you will need to download and manually install the driver [1].

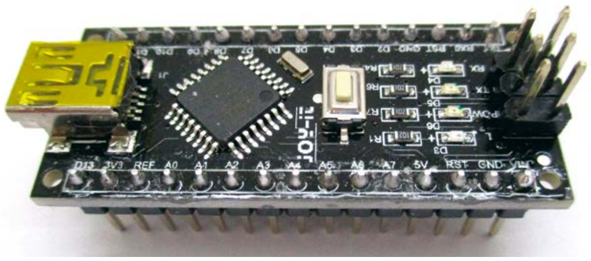


Figure 1. The Nano V3.

The **Arduino Supplement Kit** (**Figure 2**) contains a prototyping plug-board and many useful components from flying leads to pots. This gives you more LEDs, resistors, capacitors, push buttons and other stuff than you can use in a week.

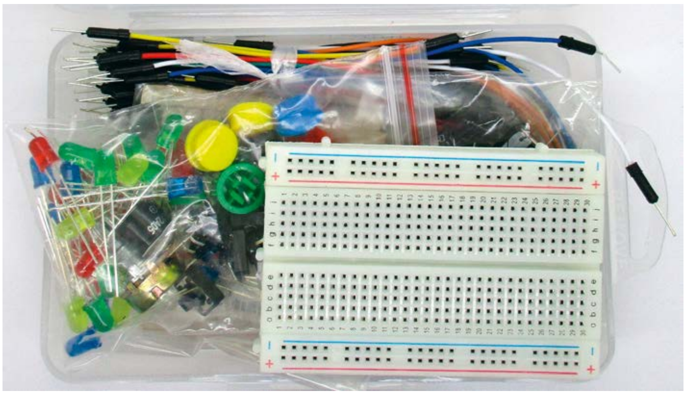


Figure 2. All the goodies in the Supplement Kit.

Now we can get to test the Nano and mount it onto the prototyping plug board (**Figure 3**) this is a stable platform that gives me confidence I can run and test a program without the risk of an accidental short-circuit. A USB cable with a mini-USB plug is all I need now to connect to a PC. The green LED on the board lights up to show the board is powered up and the yellow LED continually emits brief flashes. For Arduino this indicates that no program is loaded yet and the bootloader is waiting to do something.

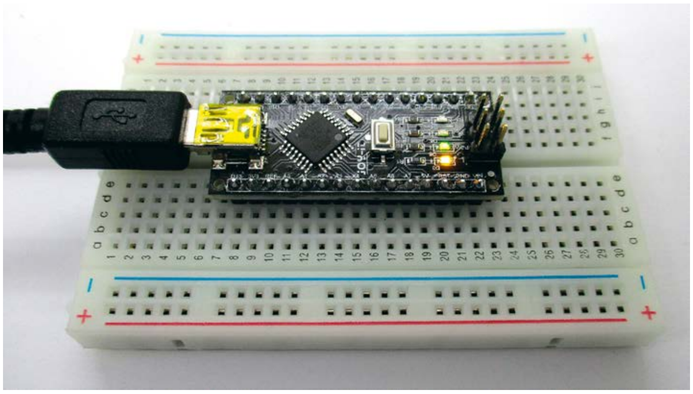
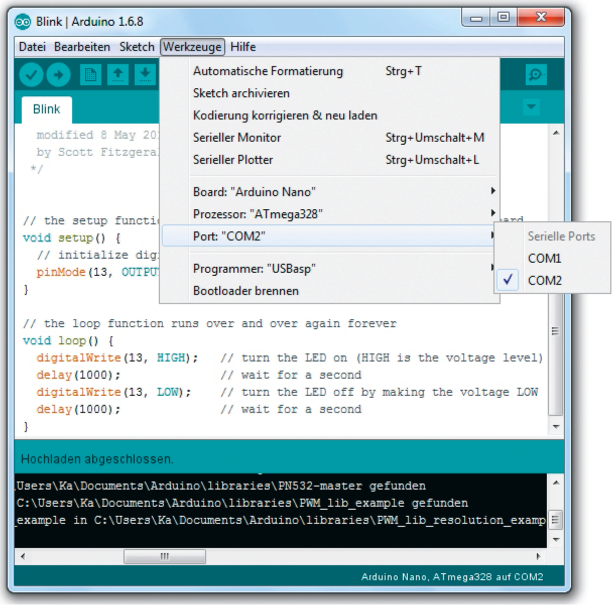


Figure 3. The Arduino Nano in use.

The Arduino IDE must now be fired up on the PC. Whenever I test a new Arduino, I always first turn to the example program `Blink.ino` (**Figure 4**), which is included in the IDE. Whenever you refer to programs that run on an Arduino they are known as 'sketches'. `Blink.ino` flashes the yellow LED at 2 s intervals. Before the program can be loaded on the Arduino, I have to select the Arduino Nano board as target in the IDE and activate the communication interface (COM2). The last controller board I used with this PC was also fitted with a CH340 and it used the COM2 port also.


Blink | Arduino 1.6.8
Datei Bearbeiten Sketch Werkzeuge Hilfe
Blink
Automatische Formatierung Strg+T
Sketch archivieren
Kodierung korrigieren & neu laden
Serieller Monitor Strg+Umschalt+M
Serieller Plotter Strg+Umschalt+L
Programmer: "USBasp"
Bootloader brennen
Serielle Ports
COM1

Board: "Arduino Nano"
Prozessor: "ATmega328"
Port: "COM2"
✓ COM2

```
  modified 8 May 20
  by Scott Fitzgera
 */


// the setup functi
void setup() {
  // initialize dig
  pinMode(13, OUTPU
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
}
```

Hochladen abgeschlossen.

```
Users\Ka\Documents\Arduino\libraries\PN532-master gefunden
C:\Users\Ka\Documents\Arduino\libraries\PWM_lib_example gefunden
example in C:\Users\Ka\Documents\Arduino\libraries\PWM_lib_resolution_examp
```

Arduino Nano, ATmega328 auf COM2

Figure 4. Setting up and loading the Nano in the IDE.

Next the upload button was activated and I waited until the upload was completed. No errors were detected and the LED starts to flash slowly. I can't resist fiddling so I just change the delay time in the blink program to 2000 ms and upload the sketch again. It flashes slower.

This all works just as I had expected… so what could we test now? I had a rummage around in the supplement kit and found a buzzer that was just crying out to be tested. It looked like the type of buzzer with internal electronics. I hook it up to the 3.3 V output of the Nano and it bleeps, next I try it at 5 V, woah… that's piercing. The buzzer comes with a label over the sound outlet which I immediately re-attach. Could we try a resistor in series with the buzzer to quieten it a bit? Why not — any value higher than 100 ohms however is too high and stops the buzzer from working. It must work connected directly to a GPIO pin of the Nano board so I connect it to pin 13, which also drives the on-board yellow LED. That works as expected — the buzzer now sounds every time the LED comes on.

I am beginning to suspect that the buzzer is not a piezo-type as I first thought but is instead an electromagnetic buzzer. I wave a bar magnet close to the buzzer and there is a noticeable pull when I get close. This indicates that inside is a magnetic speaker membrane with a solenoid coil driven by a transistor oscillator. This type of buzzer needs a lot more power than a comparable piezo buzzer and is the reason it won't work with a high value resistor in series. An ATmega328 port pin however can provide enough power — it can even drive very low-power DC motors without any additional motor driver stage.

Drive the buzzer with PWM signals
What else could we build using the components in the kit? I tested a red LED with a 330-Ω series resistor to D9 using the example sketch `Fade.ino` from the Arduino IDE. It all worked straight away with the LED continuously glowing brighter then darker. The dimming is all produced by quickly switching the port output on and off.

What would happen if I hooked up the buzzer to the PWM output signal? Actually, that worked as well although it's not really intended to be driven that way. The LED glows brighter and dimmer and the buzzer volume changes—to a lesser extent — at the buzzer frequency. During operation you can hear some additional harmonics produced when the natural buzzer frequency and multiples of the PWM frequency coincide.

What else can we fit on the breadboard? At first glance, there doesn't seem to be much space left but it is possible to squeeze a few more components on the board. Two extra-large push-button, one pot, the buzzer, the LED, the resistor and many cables, all fit well (**Figure 5**). I just connected to two of the push button pins and left the other two unused, hanging over the edge of the board.

Now the Fade example needs to be modified so that the red button stops the output and switches off the sound and the blue button switches it back on. I can use the pot value setting to set the speed. As usual, things don't always go without a hitch and I need to do some background reading to find the solution (I'd rather be working with Bascom). In the end everything worked out as planned. Although the project doesn't really perform anything useful and is a little annoying (hit S1 — the stop button) it's good just to get confidence by using the hardware and software features of Arduino.

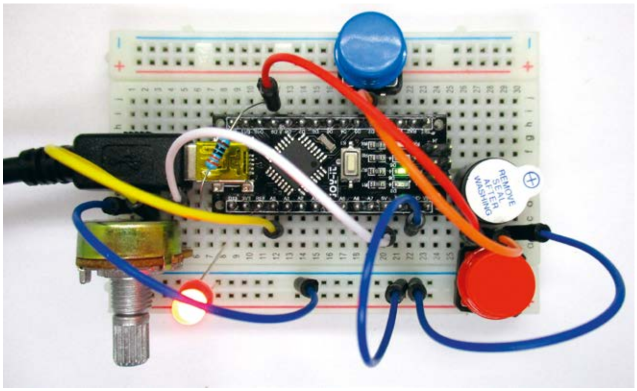


Figure 5. An experimental setup.

I did notice something peculiar when using the board; after downloading and test running a program I unplugged the board and then plugged it into a USB power bank to make the application portable. When I later plugged it back into the PC, the program was gone! The yellow LED was blinking, indicating an empty Nano. What's going on? A search on the web brought the answer: It is all to do with the version of the Arduino IDE I was using. In Arduino Version 1.6.8. the system will always put a newly plugged-in Arduino into boot mode. Why hadn't I noticed this before? Maybe it's because I usually use the slightly older (1.6.5) version of the IDE; its startup sequence is a bit faster.

Plotting analogue signals

Sometimes version 1.6.8 or higher must be used, especially if you need to make use of the serial plotter display. This allows you to plot varying signal levels. In version 1.8.2, the plotter can even display multiple channels. The program was modified again to plot the value of the variable `brightness`. In addition, a low-pass filter made up with a 100 kΩ resistor and 10 μF capacitor was used to smooth the PWM signal which is measured at the analogue input pin A2 (**Figure 6** and **Listing 1**). The result can be seen in **Figure 7**. The brightness is controlled with a triangular waveform, but the RC filter makes it look closer to a sine wave function. After some time, the pot was rotated to slow the output. The signal period varies relative to the time constant of the RC filter. The amplitude of the 'sine wave signal' increases and the signal becomes distorted. Such experimentation would normally need a bench full of test equipment but here the Nano manages everything on its own.

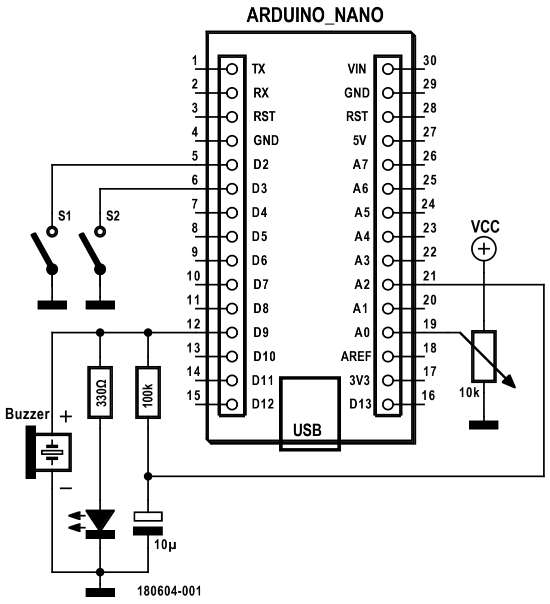


Figure 6. A low-pass filter using 100-kΩ resistor 10-μF capacitor.

```
/*
 Fade
 This example shows how to fade an LED on pin 9
 using the analogWrite() function.
 This example code is in the public domain.
 Additional: pot + s1 off + s2 on + AD + serial plot
 */

int led = 9;           // the pin that the LED is attached to
int brightness = 0;    // how bright the LED is
int fadeAmount = 5;    // how many points to fade the LED by
int s1 = 2;
int s2 = 3;

// the setup routine runs once when you press reset:
void setup() {
  // declare pin 9 to be an output:
  pinMode(led, OUTPUT);
  pinMode(s1, INPUT_PULLUP);
  pinMode(s2, INPUT_PULLUP);
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  // set the brightness of pin 9:
  analogWrite(led, brightness);
  Serial.print(brightness);
  int U = analogRead(A2);
  Serial.print( " ");
  Serial.print(U/4);
  Serial.println ( " ");
  // change the brightness for next time through the loop:
  brightness = brightness + fadeAmount;

  // reverse the direction of the fading at the ends of the fade:
  if (brightness == 0 || brightness == 255) {
    fadeAmount = -fadeAmount ;
  }
  // wait for … milliseconds to see the dimming effect
  delay(analogRead(A0)/10+10);
  if (digitalRead(s1) == LOW){
    analogWrite(led,0);
    while (digitalRead(s2) == HIGH);
  }
}
```

Listing 1. Das erweiterte Fade-Beispiel.

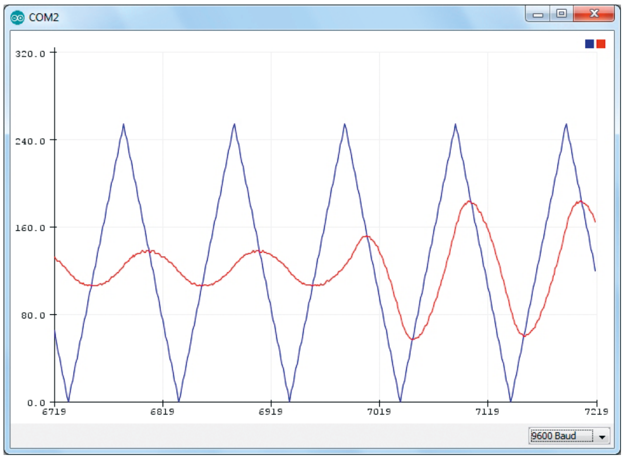


Figure 7. Output waveform and filtered signal.

ISP Programming

By the way, you are not tied to using the Arduino IDE. In principle, you can also use any other development environment for AVR controllers. The hex file can then be flashed to the Nano using the ISP plug. This makes this small board even more universal. As I mentioned earlier, I like to use Bascom. In the microcontroller series ('Microcontroller BootCamp', *Elektor* magazine from 4/2014), an Arduino Uno uses a specially adapted Bascom boot loader and this will also work with the Nano which uses the same ATmega328 controller and runs at the same 16 MHz clock frequency.

I tried to flash the bootloader into the Nano using an STK500 but there was a problem. The Nano (both original and clone) uses a particularly low value (1 kΩ) pull up resistor on the reset pin so the STK500 was not able to pull the pin low enough to trigger a reset. In contrast the Arduino Uno uses a 10 kΩ pullup at this point, which the STK500 can easily handle. For the Nano, I tried a quick and dirty fix by connecting an external 1 kΩ resistor between reset and GND. Now in normal operation the reset voltage level is lower but not low enough to trigger a reset. Now when the SKT500 issues a reset the level falls and a reset occurs. The same could happen if you use another type of programmer, if yours doesn't work, first check the voltage level at the reset input.

Web links
[1] CH340G driver: www.wch.cn/download/CH341SER_ZIP.html
[2] The author's Web site: www.b-kainka.de/

NodeMCU *(Author: Fabian Kainka)*
NodeMCU is an open-source IoT platform based on the ESP8266 32-bit Wi-Fi SoC micro-controller made by Espressif, a company who specialize in products for the Internet of Things.

As you can see from **Figure 1** the hardware uses the ESP12E Wi-Fi module. It has a total of 13 GPIO user pins available to build a wide range of applications. It uses a DP2102 chip to convert USB to serial communication. An LED and two push buttons (one for reset) are also included on the board.

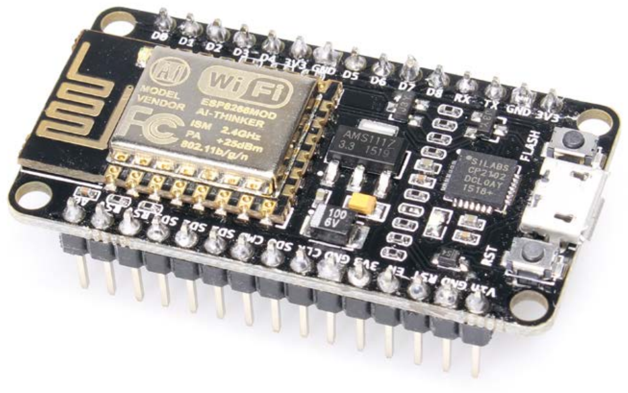


Figure 1. The NodeMCU development platform.

NodeMCU by default is actually the name of the firmware rather than the hardware kit. It uses the Lua language which is an interpreted language and means that the firmware also contains a Lua interpreter. Lua (Portuguese for 'moon') is a scripting language and is the real reason I'm interested in the NodeMCU board. An ESP8266 can be programmed (like comparable boards) fairly easily using the Arduino IDE. For me the attraction of Lua is that the program is interpreted at runtime and not translated previously by a compiler. This should make software development simpler because the program can developed and debugged on the fly.This can best be compared to the console on Linux or the Windows Power Shell, which is able to accept commands at any time, and run script files automatically. Here there is no need for a development environment, just a simple text editor. This can make your job easier as you will see later.

Unboxing
The first step was to take the board out of its box and hook it up to my PC using a micro-USB cable. To establish serial communication I used the Terminal program PuTTY, but any other serial monitor program should work just as well. After I had established the connection at 9600 baud, I was able to receive the first readable characters. They verified that the board is running firmware Version 0.9.6, dated mid 2015 — that seems quite old so I will need to set about finding a more recent version. On my quest I found a lot of other exciting stuff…

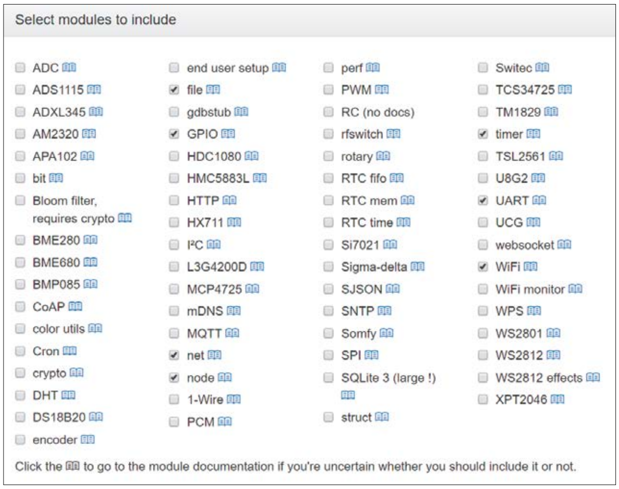
NodeMCU Technical data

- The ESP12E Module
- 32-Bit Low-Power CPU clocked at 80 MHz
- 4 MB Flash memory for storing the program and files
- Wi-Fi: 802.11 b/g/n at 72.2 MBit/s
- Printed Wi-Fi antenna
- 3.3 V operating voltage
- Programming and power via Micro-USB port.
- DP2102 USB-to-Serial converter
- 13 GPIO pins
- 10-Bit ADC (successive approximation)
- SPI, I^{2}C, I^{2}S, 2 x UART, IRDA, PWM, RTC
- Onboard LED, 2 pushbuttons

Flash the latest firmware
My journey led me to the corresponding GitHub web site [1] (GitHub is a web-based internet site for open-source code). Here all important information was linked, I was just hoping to download the latest firmware edition but to my surprise, as well as the firmware, each user can assemble their own version of the firmware. You can choose from over 65 different different modules which ones you want to integrate into the LUA firmware. There is not enough room to fit all the modules into the 4OO MB flash memory… that would be like installing all the libraries in the Arduino IDE onto an Arduino board. The user therefore needs to decide in advance which functions they are going to need.

Standard Modules such as Timer, UART, WiFi, GPIO and so on should be included in almost all firmware configurations, while modules like HTTP, MQTT, SNTP or specialized sensor modules such as DHT (Temperature and humidity), BME280 (Temperature, humidity and air pressure), HMC5883L (Triple-Axis Compass) or TCS34725 (colour/light sensor) should only be included when they are specifically required.

That all sounds like a lot of work to assemble all the individual modules and compile a running firmware. The whole process would be pretty cumbersome if it were not for the easy-to-use **Cloud Build Service** (**Figure 2**). On the website [2] you can simply 'personalize' your own personal firmware environment with all the necessary modules. There are additional options, for example, which of your branches you want to use from the GitHub project flow structure and whether you need TLS/SSL support or FatFs to read SD cards. Once you have made all your choices, you just need to enter your email address in the appropriate field and click **Start your build**. Creating the firmware takes a while (for me me it took about three minutes), then you get an email with the download link which remains valid for 24 hours.



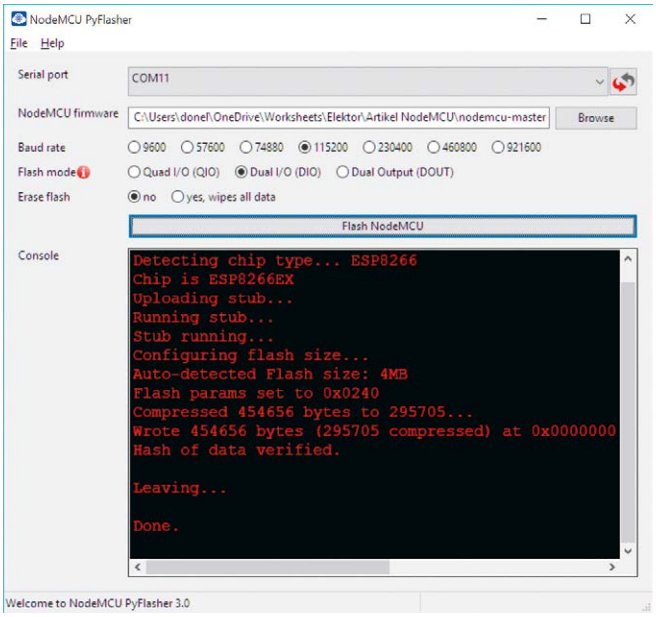
Select modules to include

☐ ADC	☐ end user setup	☐ perf	☐ Switec
☐ ADS1115	☑ file	☐ PWM	☐ TCS34725
☐ ADXL345	☐ gdbstub	☐ RC (no docs)	☐ TM1829
☐ AM2320	☑ GPIO	☐ rfswitch	☑ timer
☐ APA102	☐ HDC1080	☐ rotary	☐ TSL2561
☐ bit	☐ HMC5883L	☐ RTC fifo	☐ U8G2
☐ Bloom filter, requires crypto	☐ HTTP	☐ RTC mem	☑ UART
	☐ HX711	☐ RTC time	☐ UCG
☐ BME280	☐ I²C	☐ Si7021	☐ websocket
☐ BME680	☐ L3G4200D	☐ Sigma-delta	☑ WiFi
☐ BMP085	☐ MCP4725	☐ SJSON	☐ WiFi monitor
☐ CoAP	☐ mDNS	☐ SNTP	☐ WPS
☐ color utils	☐ MQTT	☐ Somfy	☐ WS2801
☐ Cron	☑ net	☐ SPI	☐ WS2812
☐ crypto	☑ node	☐ SQLite 3 (large !)	☐ WS2812 effects
☐ DHT	☐ 1-Wire		☐ XPT2046
☐ DS18B20	☐ PCM	☐ struct	
☐ encoder			

Click the 📖 to go to the module documentation if you're uncertain whether you should include it or not.

Figure 2. Module selection for the Firmware setup.

The fully compiled firmware now only needs to be loaded to the board. There is also a practical tool, called the **NodeMCU PyFlasher** (**Figure 3**), which can be downloaded from [3] for Windows and MacOS. Installation is not necessary; the file can be executed directly. Select the correct COM port and downloaded the firmware file and then start the upload process. That was totally unproblematic for me. A detailed description of how to create and flash the firmware and also information about all the modules and basic FAQs can be found on the excellent project documentation page [4].



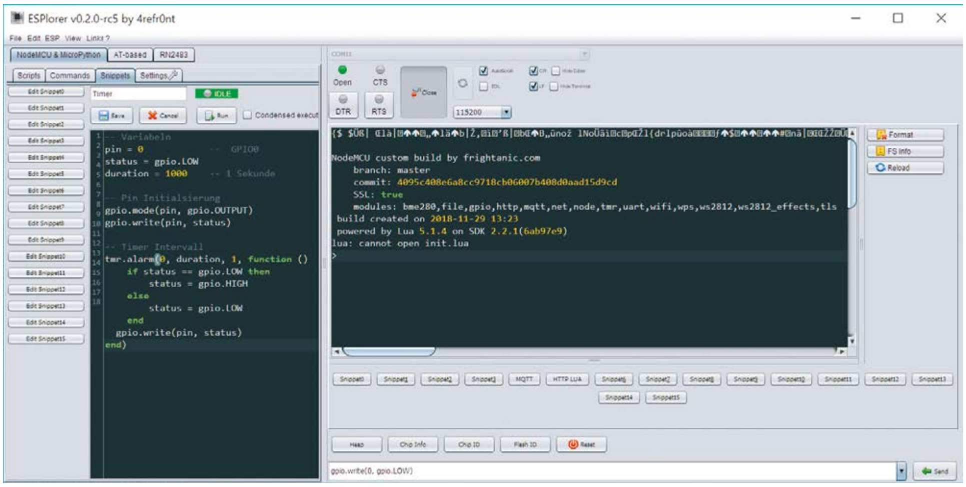
Serial port: COM11
NodeMCU firmware: C:\Users\donel\OneDrive\Worksheets\Elektor\Artikel NodeMCU\nodemcu-master
Baud rate: 9600 57600 74880 [✓ 115200] 230400 460800 921600
Flash mode: Quad I/O (QIO) [✓ Dual I/O (DIO)] Dual Output (DOUT)
Erase flash: [✓ no] yes, wipes all data

```
Detecting chip type... ESP8266
Chip is ESP8266EX
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Auto-detected Flash size: 4MB
Flash params set to 0x0240
Compressed 454656 bytes to 295705...
Wrote 454656 bytes (295705 compressed) at 0x0000000
Hash of data verified.

Leaving...

Done.
```

Figure 3. The PyFlasher tool.

Hello World – Lua-Test

On the documentation page I also found a link [5] to the very useful **ESPlorer** tool. This program (**Figure 4**) is not just a simple serial monitor but a kind of Swiss Army knife for the ESP8266 — regardless of whether you are using MicroPython, NodeMCU or an AT firmware.



Timer

```
-- Variabeln
pin = 0          --  GPIO0
status = gpio.LOW
duration = 1000    -- 1 Sekunde

-- Pin Initialsierung
gpio.mode(pin, gpio.OUTPUT)
gpio.write(pin, status)

-- Timer Intervall
tmr.alarm(0, duration, 1, function ()
    if status == gpio.LOW then
        status = gpio.HIGH
    else
        status = gpio.LOW
    end
  gpio.write(pin, status)
end)
```

COM11 115200

```
{$ $Üß| Ɑlà|[illegible]„lă b|Ž,[illegible]ß|[illegible]B„ûnož lNoÜăi[illegible]c[illegible]pƊŽl{drlpûoà[illegible]

NodeMCU custom build by frightanic.com
    branch: master
    commit: 4095c408e6a8cc9718cb06007b408d0aad15d9cd
    SSL: true
    modules: bme280,file,gpio,http,mqtt,net,node,tmr,uart,wifi,wps,ws2812,ws2812_effects,tls
 build created on 2018-11-29 13:23
 powered by Lua 5.1.4 on SDK 2.2.1(6ab97e9)
lua: cannot open init.lua
>
```

gpio.write(0, gpio.LOW)

Figure 4. The ESPlorer tool: Terminal on the right, Snippets on the left.

After downloading and starting the executable ESPlorer.jar file I established the serial connection to the board. This latest firmware, communicates at 115200 baud. The program now tried to communicate with the board automatically without success. Only when

I pressed the reset button on the board did I get the first message telling me about the firmware version and all the modules, so communication was successful.

The ESPlorer program screen shows a line of buttons along the bottom on the right which can be used to send automatic commands to the board. For example, I can use them to read the chip ID or information on the file system. There is, of course, an input window for direct commands. I first tried entering the following commands:

```
gpio.mode(0, gpio.OUTPUT)
gpio.write(0, gpio.LOW)
```

If you are already familiar with Arduino source code these should be easy to understand. First, GPIO0 is configured as an output and then set to low. A series resistor and LED with its anode connected to VCC is attached to this pin so setting the output low will make the LED light up. That was just the simplest test I could think of at the time to see if the commands really work without compiling.

This manual input window is really handy if you only want to test one or two commands. It keeps track so it makes it simple to execute a command again. Most programs that do anything useful will of course need a whole sequence of commands. I wanted to implement the classic “Hello World” and let the LED flash. These are just a few lines, but fortunately ESPlorer has the ability to create entire code snippets and even save it on the PC. This snippet containing command sequences, called Snippets here, can then, with a click be automatically sent to the board. I built the following blink program by typing 15 editable snippets and then hit run to start:

```
-- Variables
pin = 0            --  GPIO0
status = gpio.LOW
duration = 1000    -- 1 sec
-- Pin Initialization
gpio.mode(pin, gpio.OUTPUT)
gpio.write(pin, status)
-- Timer Intervall
tmr.alarm(0, duration, 1, function ()
    if status == gpio.LOW then
        status = gpio.HIGH
    else
        status = gpio.LOW
    end
  gpio.write(pin, status)
end)
```

Interestingly, variables do not need to be declared first. If you have some experience with JavaScript, you’re probably aware of that already. Also interesting is the use of the timer. In this case, the timer will simply pass an entire function defined in the call (called an anonymous function) that will run every second. The flashing LED was therefore no problem and the program was easy to understand. Here we are using an IoT board, so I created the following snippet to connect to my router:

```
wifi.setmode(wifi.STATION)
station_cfg={}
station_cfg.ssid="YourSSID"
station_cfg.pwd="YourPassword"
station_cfg.save=true
station_cfg.auto=true
wifi.sta.config(station_cfg)
wifi.sta.connect()

wifi.eventmon.register(wifi.eventmon.STA_CONNECTED, function(T)
    print("WiFi connected!")
end)
```

Once a connection has been established, the program puts out the message 'WiFi connected!'. Now I need to create an anonymous function that will be executed when the Wi-Fi connection has been established. This type of event-driven programming is typical in Lua.

The connection setup worked fine. The configuration used here also ensures that the Wi-Fi credentials are stored and maintained after a reset. In addition, `station_cfg.auto=true` ensures that the board will always attempt to connect to the network automatically. Without a corresponding program, no 'Wifi connected' message will appear, that will only occur after a board reset.

Since the connection process has worked so well, I wanted to go on and test the MQTT module. The documentation contains a good example of how to connect the board to a broker as an MQTT client; I copied it directly into a snippet with some slight modifications and then tested it 'live on the board' [6]. Here, too, everything went unusually smoothly and gave me confidence to move on to the next step.

Now I need to write a program that remains resident in the board memory, automatically executing when it reboots. This should be easy to achieve when you consider:

- The firmware provides its own Virtual File System to which files can be uploaded. These files can be script-files executed with th `dofile(file name)` command or web pages, images or other files that you want to store on the board.

- On reboot the board checks whether the *init.lua* file exists in the file system, if it does, it runs automatically.

Specifically, I simply switched from the *Snippets* tab to *Scripts* on ESPlorer and used the text editor. Here I wrote the following lines, saved to my PC as *init.lua*, and then Uploaded them to the NodeMCU:

```
wifi.eventmon.register(wifi.eventmon.STA_CONNECTED, function(T)
    print("WiFi connected!")
    http.get("https://pastebin.com/raw/k4ccGx3T",
 nil, function(code, data)
     if (code == 200) then
          if file.open("run.lua","w+") then
               file.write(data)
               file.close()
          end
      else
          print("HTTP request failed")
      end

      if file.exists("run.lua") then dofile('run.lua') else print("run.lua
not found") end
    end)
 end)
```

This program now does something that usually can only be done in scripting languages. Firstly it waits for the successful setup of a Wi-Fi connection and then in run time it downloads a program from the Internet which it stores with the filename *run.lua* and then runs it. In this example, it just downloaded a simple blink program from a Paste site (Pastebin.com). Any other resources can of course also be used here. Both the HTTP-Module and the TLS/SSL-Option were used in this firmware.

I was impressed with Lua and found it to be a really powerful and compact interpreted language. I imagine there are a whole series of programs on a web server that can be tweaked and improved. You can choose your program via a website or MQTT and download it to the board. It could hardly be more modular or more simple.

@ www.elektor.com
- NodeMCU Microcontroller board
  www.elektor.com/nodemcu-microcontroller-board-with-esp8266-and-lua
- Dogan and Ahmet Ibrahim: 'ESP8266 and MicroPython'
  www.elektor.com/esp8266-and-micropython

Web Links
[1] Git Repository: https://github.com/nodemcu/nodemcu-firmware
[2] Firmware: https://nodemcu-build.com/
[3] PyFlasher: https://github.com/marcelstoer/nodemcu-pyflasher/releases
[4] Documentation: https://nodemcu.readthedocs.io/
[5] ESPlorer: https://esp8266.ru/esplorer/#download

[6] MQTT example: https://nodemcu.readthedocs.io/en/master/en/modules/mqtt/#example
[7] The author’s Web site: https://fkainka.de/

Chapter 51 • BBC micro:bit for Electronicists (1)

In bed with mbed

The BBC micro:bit 'computerette' is not only for school-age users — it's also a great controller board for grown up electronicists. Crammed into its tiny footprint is almost everything that you need: digital inputs/outputs with PWM and A-D converters along with a USB interface for power supply, programming and transferring data. A variety of sensors and Bluetooth round off the configuration.

Sure, an Arduino also offers a range of inputs and outputs but the BBC micro:bit offers additionally numerous sensors and control functions. Two input buttons and a 5x5-LED display field, a compass sensor and a 3-axis accelerometer, plus light and temperature measurement. As the crowning glory you have Bluetooth Low Energy (BLE) data transfer.

Vital connections

The PCB was developed by the BBC in collaboration with the University of Lancaster as a teaching aid for schools. For this reason it is equipped with five oversized connection pads with 4 mm holes, to which crocodile clips can be linked easily (**Figure 1**). In theory these clips might also touch the smaller pins on either side but the designers took this risk into consideration. The pins adjacent to the GND and 3-V connections carry the same potential, and next to the large Port connections 0, 1 and 2 you will also find Port pins that will survive any direct cross-connection without complaining.

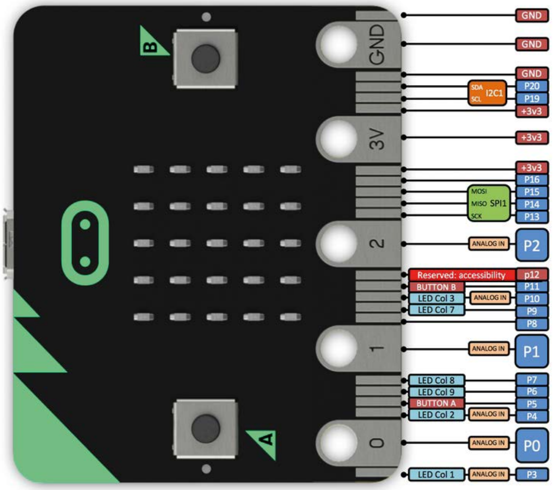


Figure 1. Connections of the BBC micro:bit.

You can even use croc-clip cables for your first simple experiments (see header photo). When things get more complex and you need more connections, you'll require some kind of connector plugs if you wish to avoid soldering wires. With its 0.05- mm pitch connections, the board happens to match the card slots of a discarded PC motherboard, so with a pair of pliers, small saw and soldering iron you could well upcycle this into a new life form. It is of course far easier to use a card connector with 2 x 40 pins that is properly made for the job. Custom-made boards are available in the Elektor Store [6][7]. **Figure 2** shows the PCB for the BBC micro:bit Weather Station project.

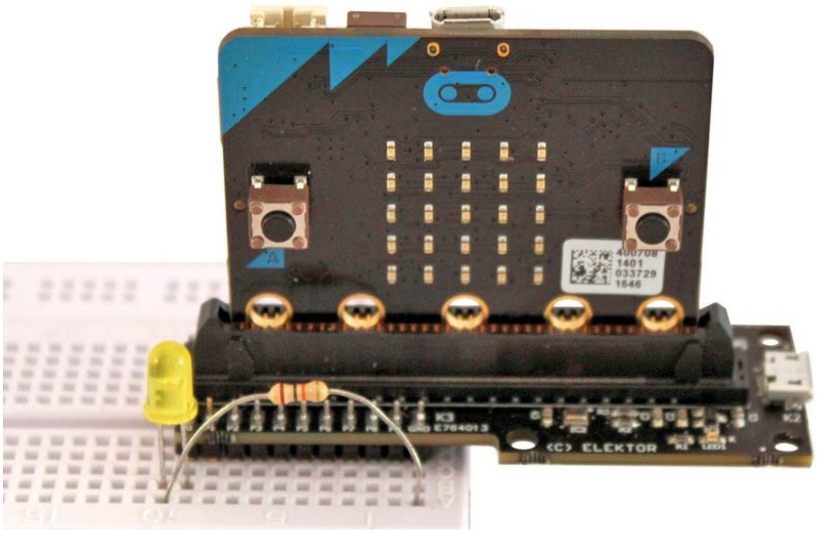


Figure 2. Elektor PCB 150652-1 used as a breadboard.

Alternatively you could use dual-row header strips with 2 x 20 pins that you simply solder to the gold contacts (**Figure 3**). The pins of the header strips are spaced at 2.54 mm, which is twice the pitch of the small contacts on the PCB. The workaround is to solder pins to every second contact on the BBC micro:bit.

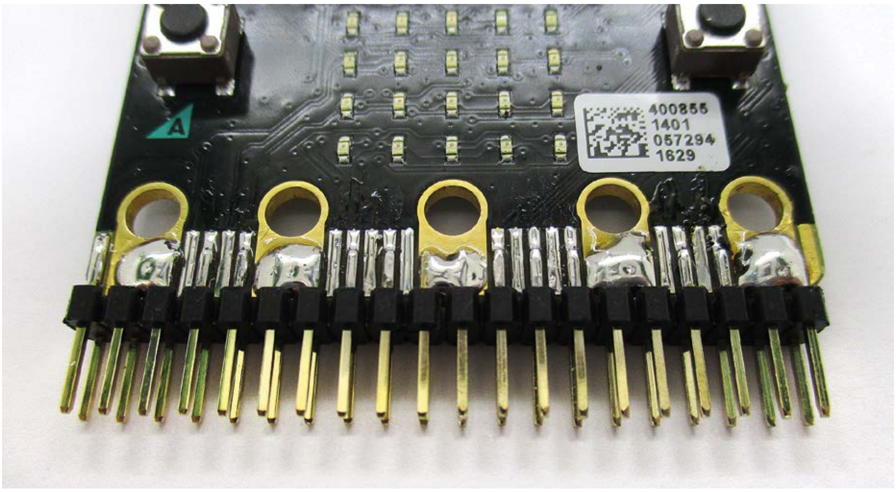


Figure 3. Header pin assignments.

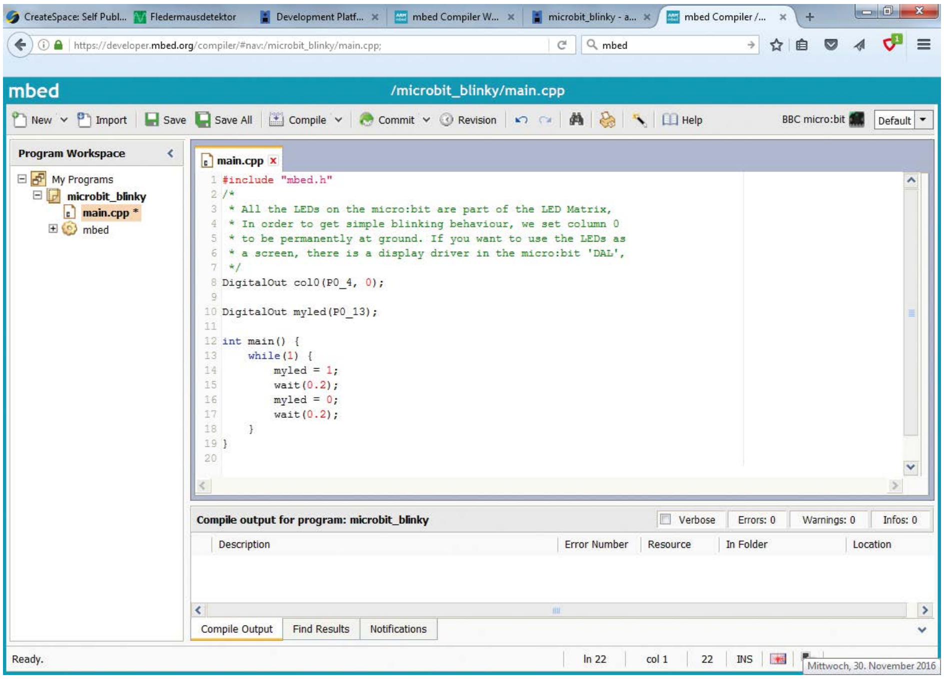
On the rear side of the board things are much the same but you need to take care that all connections on the back are totally isolated. The pins soldered on that side serve primarily only for mechanical stability. The rear-side pins do have a secondary purpose, however: if you need to make use of a signal on the micro:bit that you have not employed so far, you can connect this to one of the rear-side pins with a short piece of wire. If you intend to plug the Micro:bit into another PCB, then you will need to remove the corresponding pin in the front-side row to avoid creating a short circuit. There are of course plenty of pins that are not required, for instance the auxiliary connections for GND and VCC. What might be critical on the other hand are the I2C connections, not all of which can be handled using the header strip method. To resolve this we remove the superfluous pin at the GND connection. This is very simple: heat the pin with a hot soldering iron tip, then pull it out with pliers. You can now poke some wire through the opening previously occupied by the pin and then solder one end to the desired connection point (SDA, P20). On the rear side you solder the wire to the corresponding pin, ensuring the integrity of the I2C bus is preserved.

Our first programs
Practical applications in the realm of electronics are really simple to fix up. Several programming languages that were designed primarily for educational use in schools [1][5] are suitable. Here, however, we will work with C++, in which practically everything is possible, whereas other languages always have certain limitations. Using the mbed platform [2] is particularly straightforward, because you can work online without having to install anything.

Log into mbed and set up an area of your own, if you have not already done this for other projects. If you now attempt to load an existing example, you will be reminded to select a platform, in other words a system with which you wish to work. You will be referred to the Hardware page and here you may be amazed at how many boards are already usable, one being from Elektor by the way. In this situation the micro:bit is the ‘target system’. Here you will come across further information, such as the circuit diagram of the Micro:bit and a button to ‘add to your mbed Compiler’. There are also crucial links to the documentation of Lancaster University and some entry-level sample code.

The first of these is called *Micro:bit_Blinky* and should be downloaded now. The compiler shows the newly installed platform and the imported program. Clicking on *main.cpp* opens the uncluttered source code (**Figure 4**).

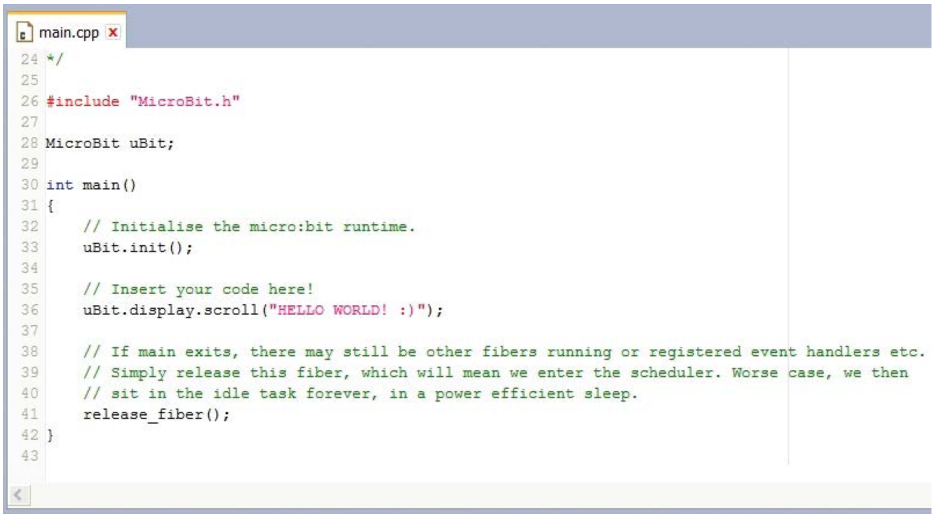
Something that will strike you immediately is the way that the Port connections are labeled P0_4 and P0_13, which conflicts with the official pin assignment. This is because when you install *mbed.h*, you are addressing the controller on the micro:bit but not the entire board with all its features and options. We can test the program now. Using *Compile* creates a hex file, which is then simply copied into the USB memory of the Micro:bit. The yellow status LED on the board begins to flash straightaway, indicating that the system is ready to start programming the controller. After one second the process is complete and you can see the result: the upper left-hand LED on the 5x5 LED display flashes.



```
#include "mbed.h"
/*
 * All the LEDs on the micro:bit are part of the LED Matrix,
 * In order to get simple blinking behaviour, we set column 0
 * to be permanently at ground. If you want to use the LEDs as
 * a screen, there is a display driver in the micro:bit 'DAL',
 */
DigitalOut col0(P0_4, 0);

DigitalOut myled(P0_13);

int main() {
    while(1) {
        myled = 1;
        wait(0.2);
        myled = 0;
        wait(0.2);
    }
}
```

Figure 4. The micro:bit Blinky project.

The second example is called *Micro:bit Hello World* and embeds *microbit.h* (**Figure 5**). This makes it possible to employ the highly specific options that the board offers. Importing the program takes a noticeably long time, which indicates that very many files are being loaded. This is the great advantage of Mbed: everything you previously had to merge and integrate with great care Mbed now handles itself. With a different IDE on your own computer you could of course do this yourself, but you might well spend a long time doing this or even give up in desperation. With mbed you can work without even a moment's thought for what's going on in the background. In the example some scrolling script is produced. The mission-critical line is self-explanatory: uBit.display.scroll ("HELLO WORLD! :)"; compile, transfer, run.



main.cpp

```
*/

#include "MicroBit.h"

MicroBit uBit;

int main()
{
    // Initialise the micro:bit runtime.
    uBit.init();

    // Insert your code here!
    uBit.display.scroll("HELLO WORLD! :)");

    // If main exits, there may still be other fibers running or registered event handlers etc.
    // Simply release this fiber, which will mean we enter the scheduler. Worse case, we then
    // sit in the idle task forever, in a power efficient sleep.
    release_fiber();
}

```

Figure 5. The micro:bit Hello World project.

You may be thinking, “So much effort for just a scrolling display?” But in reality Mbed offers much more, namely total access to all the vital hardware elements of the system, not only for display functions but also to Ports having A-D converters and PWM, to specialized sensors like compass and accelerometer functions and much more. In the Workspace you will find the *microbit* folder and within this the *microbit-dal* (Device Abstraction Layer) sub-folder with an unbelievable number of files through which you can rummage for hours in order to discover all the options.

A quicker way would be to read Lancaster University’s own documentation [3]. In this you can find all the important information on every topic, also snippets of code that you can copy and insert into your own source code. You could easily expand the existing source code *main.cpp* of the micro:bit Hello World project too and try out everything, one item after another. An alternative would be to clone the project and then continue working on the copy. The cloned project will now be called, for instance, *microbit-test* and will work the same as the original initially. Following on from this, we have presented some brief source code samples that can be copied into *main.cpp* easily and then tested in this way. All of these programs can be downloaded in .txt format from the Elektor website [8].

Measuring voltages

What we all need constantly in our personal electronics labs are A-D inputs. The micro:bit has a 10-Bit A-D converter with six potential inputs, three of which are taken out to the large connection pads. For testing its characteristics we have written a simple program (**Listing 1**). Using *uBit.io.P0.getAnalogValue()* you obtain a 10-bit value for the voltage range up to 3.3 V. This is converted into mV and then displayed.

```
//Voltage1
#include "MicroBit.h"

MicroBit uBit;

int main()
{
   uBit.init();
   MicroBitSerial serial(USBTX, USBRX);
   while (1) {
        int u = 3300 * uBit.io.P0.getAnalogValue()/ 1023;
        uBit.display.scroll(u);
        uBit.serial.printf("%d\r\n", u);
        uBit.sleep(500);
        }
}
```

Listing 1. Voltage measurement.

The micro:bit's typical method of displaying data is scrolling text on the LED display. This is highly functional but the legend displayed does require a fair degree of attentiveness to read it. Fortunately there is also a serial interface enabling you to make the data visible on a terminal screen. To use all this under Windows we need to install the mbed Windows Serial Port driver. How you do this is described in [4].

If you connect Input 0 to the 3-V connection on the board 3300 mV is displayed, as expected. Comparative measurements prove that the actual voltage is slightly smaller. That's because the board stabilizes the 5 V supply from the USB input to 3.3 V using a voltage regulator followed by a Schottky diode to produce V_{CC}.

With an open input the measurement indicates about 880 mV. If you measure the voltage simultaneously using a high-impedance oscilloscope, you will see a higher voltage that nosedives (collapses) shortly after each measurement. In point of fact the three most significant connections 0, 1 and 2 are each provided with pull-up resistors of 10 MΩ, enabling the digital Port pins to serve as touch sensors without extra effort. The true open-circuit voltage should thus lie close to 3.3 V. To measure this as well you can connect a 100 nF capacitor to the input. This will then charge itself up to the open circuit voltage and buffer this during the measurement.

Sensors galore

Our second little sample program (**Listing 2**) indicates numerous different measurement readings from the various inputs and sensors. Among these are the temperature of the microcontroller and the system time in milliseconds since the last start. You also have the three voltages on Pins 0 to 2, accelerometer values on three axes and an XYZ-combined magnetic field strength. The fact that two complex sensors are read via the I2C-bus is something that we can simply accept, as others have already done the difficult part of the work. Only if you insist on knowing exactly how this works do you have to work through the deeper layers of countless embedded files.

```
//Sensors
#include "MicroBit.h"
MicroBit uBit;
int main()
{
   uBit.init();
   MicroBitSerial serial(USBTX, USBRX);
   while (1) {
        uBit.serial.printf("Time: %d ms \r\n", uBit.systemTime());
        uBit.serial.printf("Temp: %d deg \r\n", uBit.thermometer.
getTemperature());
        uBit.serial.printf("P0: %d mV  \r\n", uBit.io.P0.getAnalogValue());
        uBit.serial.printf("P1: %d mV  \r\n", uBit.io.P1.getAnalogValue());
        uBit.serial.printf("P2: %d mV  \r\n", uBit.io.P2.getAnalogValue());
        uBit.serial.printf("X: %d mG  \r\n", uBit.accelerometer.getX());
        uBit.serial.printf("Y: %d mG  \r\n", uBit.accelerometer.getY());
        uBit.serial.printf("Z: %d mG  \r\n", uBit.accelerometer.getZ());
        uBit.serial.printf("B: %d µT  \r\n", uBit.compass.
getFieldStrength());
        uBit.serial.printf("\r\n");
        uBit.sleep(1000);
        }
}
```

Listing 2. Displaying sensor values.

All of the values measured are displayed in the terminal. While you are checking all of this out some ideas may occur to you of possible applications, such as measurements with magnets. Or else how about putting an accelerometer in your car to tell drivers when they are driving a little too friskily? The readout could look like this:

```
Time: 1715018 ms
Temp: 22 deg
P0 465 mV
P1 251 mV
P2 252 mV
X 0 mG
Y 1008 mG
Z 64 mG
B: 169311 uT
```

Static numerical display

An LED display with scrolling text is hard to read; it requires a lot of concentration. If you miss even one digit you'll have to wait until it comes around again. However, with 25 LEDs available we still have some other possibilities. Of course we are used to multi-digit static displays or alternatively to analog pointer instruments.

A static display can work with binary digits or in BCD code. In both cases you need four LEDs per digit. Since the Micro:bit has five LEDs in a column, however, there's another way of presenting digits. The figures 1 through 5 are displayed from bottom upwards using one to five LEDs. For the figures 6 through 9 the light-bar dangles downwards from the top. You can visualize this by imagining there are always five dots, of which some are not always visible. Our diagram makes this clear (**Figure 6**).

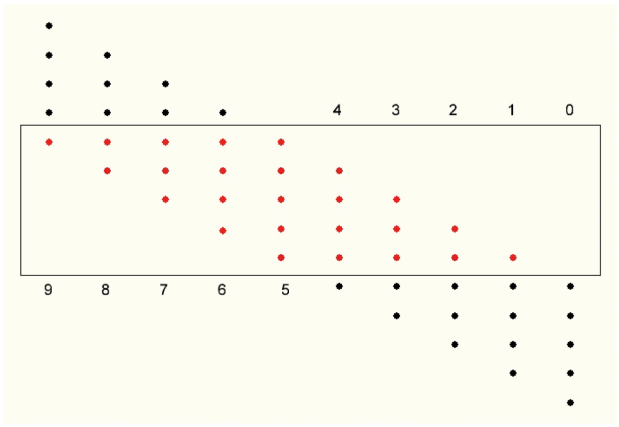


Figure 6. Numerical display.

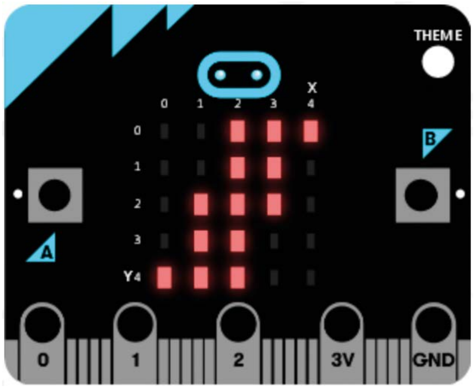


Figure 7. Displaying the figures 13579..

A display in this format (**Figure 7**) becomes readable intuitively after some practice. Incidentally this representation corresponds exactly with the structure of the Morse Code symbols for die figures 0 to 9, in which a lit LED indicates a dot and an unilluminated LED to a dash. So seven is sent as — — · · · in Morse.

For this kind of output format we have written a function in C. The figure to be output is transferred using the Integer Variables *n*. A voltage up to 3300 mV can be indicated on the display (**Listing 3**).

```
//LED-Display
#include "MicroBit.h"
MicroBit uBit;

void ledDispay (int n){
    int x;
    int y;
    int d;
    uBit.display.enable();
    MicroBitImage image(5,5);
    image.clear();
    for(int i = 0; i < 5; i++){
            d = 9 - n % 10;
            n = n / 10;
            x = 4 - i;
            for(int j = 0; j < 5; j++){
                  y = d - j;
                  image.setPixelValue(x,y,255);
            }
        }
        uBit.display.print(image);
    }


int main()
{
   uBit.init();
   uBit.io.P1.setDigitalValue(0);
   uBit.io.P2.setDigitalValue(1);

   MicroBitSerial serial(USBTX, USBRX);
   while (1) {
       int u = 3300 * uBit.io.P0.getAnalogValue()/ 1023;
       uBit.serial.printf("%d\r\n", u);
       ledDispay (u);
       uBit.sleep(500);
       }
}
```

Listing 3. Voltage measurement with static display.

In the main program two Port pins are switched additionally. P1 becomes Low, P2 is made High. You do not have to declare a changeover of data direction explicitly, as this happens automatically in the background. You can now investigate the load capability of the Port easily and use its own A-D converter for this straightaway. It's obvious that these have significantly higher impedance than the Ports of an AVR controller. Officially the Ports are capable of up to 5 mA in both directions. Test measurements indicate that a voltage drop of 300 mV arises. The 'on' resistance of the Port FETs thus amounts to around 60 Ω. Measurements taken with large loads indicate that the output current never rises above 15 mA. A current of around 10 mA flows when you connect an LED directly without a dropper resistor.

In normal situations you should use a dropper resistor, however, letting you then measure the LED voltage uniformly, compare differing LEDs and carry out further experiments (**Figure 8**).

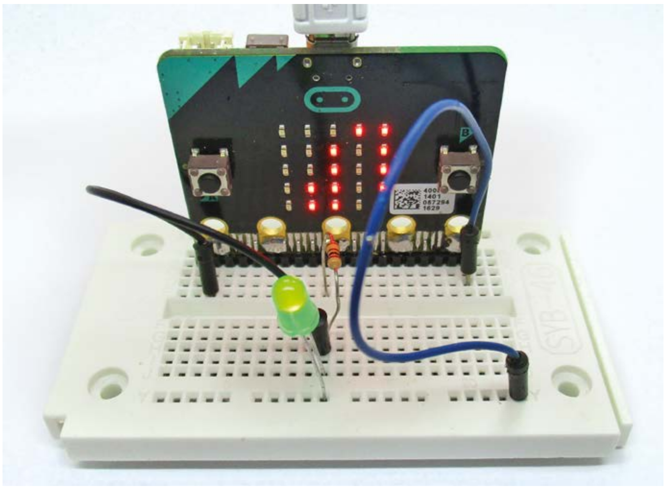


Figure 8. Measuring the LED voltage: 2.496 V.

Web Links

[1] http://microbit.org/code/
[2] https://developer.mbed.org/
[3] https://lancaster-university.github.io/microbit-docs/ubit/
[4] https://developer.mbed.org/handbook/Windows-serial-configuration
[5] B. Kainka, BBC micro:bit Tests Tricks Secrets Code, CreateSpace 2016
[6] www.elektormagazine.com/150652
[7] www.elektormagazine.com/160274
[8] www.elektormagazine.com/160273

Chapter 52 • BBC micro:bit for Electronicists (2)

Data acquisition and oscilloscope functions

Every microcontroller that features an A-to-D converter and a PC interface can be used as a logging device for data acquisition systems. But with the BBC micro:bit you get the bonus of a small LED display and a wirefree interface into the bargain. Just the job for special applications in your electronics lab!

The small footprint of the BBC micro:bit would alone make it a go-to choice for measurement tasks. Regardless of whether powered over a USB cable or by batteries, and whether it used Bluetooth or some simplified wireless protocol to communicate with the outside world, it can always be installed close to the object under examination.

A USB oscilloscope

For programming the BBC micro:bit the mbed platform has proven its worth. The basics are explained in reference [1]. All the programs mentioned in this article are available to download as text files on the Elektor website [2] and need to be copied into an existing mbed project.

When deploying the BBC micro:bit for general-purpose measurements you must ensure that the voltage range under examination cannot overstep the range between GND and V_{CC}. A protective resistor of 10 kΩ in series with the input will restrict the current flow in all situations, necessary if you accidentally exceed safe limits (**Figure 1**). As well as the analog input there's also a squarewave output, with which you can produce a handy test signal.

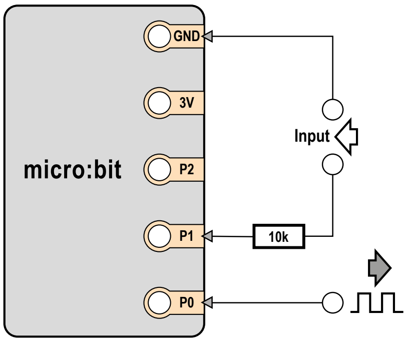


Figure 1. Measurement input and signal output.

For maximum speed the program in **Listing 1** captures data without buffering for immediate transfer of each measured value. A significant element of the cycle time arises from the serial transfer at 115,200 baud. If the measurements captured are a mixture of, say, single-digit (3 mV) and four-digit (3000 mV) amounts, these variable figures will result in uneven data flow. Consequently we raise the voltage by 1000 mV, making the possible values from 1000 mV to 4300 mV and always taking the same time to process. The program also provides its own signal source so that you can measure something without additional overheads. P1 becomes a PWM output with a PWM frequency of 10 Hz and a period of 100 ms.

For evaluating the data there are plenty of options. You could take in the data using a terminal program and then present it as a spreadsheet. A convenient alternative is the serial plotter in the Arduino IDE (version 1.6.8 onwards). This software provides a scrolling screen display, adjusted automatically to the display range, making range switching or reformatting unnecessary. With the settings shown we can measure a symmetrical square-wave signal with 10-Hz repetition rate (**Figure 2**). At the same time we now have a known time axis. The entire oscillogram clearly depicts a data acquisition duration of 300 ms.

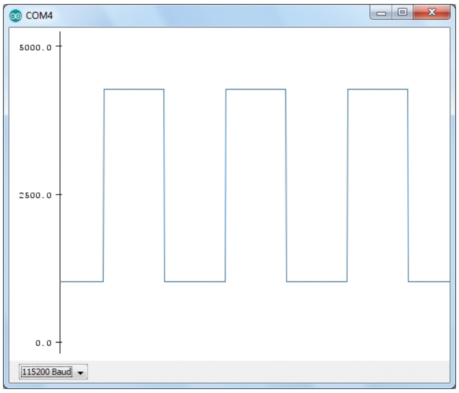


Figure 2. A squarewave signal with 10 Hz repetition rate.

The serial plotter of the Arduino IDE always plots first from left to right until the screen is completely filled. After this, the picture scrolls to the left to make old data disappear. The choice between a continuous or static display is made with the button A (`if(uBit.buttonA.isPressed()`in **Listing 1**). Just press button A for the duration of the measurement and a scrolling display is shown. As soon as you release the button, the last display is frozen, so you can examine it more closely or save a copy of it.

The additional signal output can be useful for investigating circuits or components. Using a low-pass filter with 4.7 kΩ and 22 µF a sawtooth signal is generated from the squarewave, as would be expected (**Figure 3**). The measurement range of the Arduino plotter adapts automatically to smaller voltages.

```
//Voltage Logger/Scope
#include "MicroBit.h"
MicroBit uBit;

int main()
{
   uBit.init();
   MicroBitSerial serial(USBTX, USBRX);
   uBit.io.P1.setAnalogValue(512);
   uBit.io.P1.setAnalogPeriodUs(100000);
   while (1) {
     if(uBit.buttonA.isPressed()){
```

```
                int u = 1000+3300 * uBit.io.P0.getAnalogValue()/ 1023;
                uBit.serial.printf("%d\r\n", u);
               // uBit.sleep(100);
                }
              }
        }
```

Listing 1. Rapid measurement with direct data transfer.

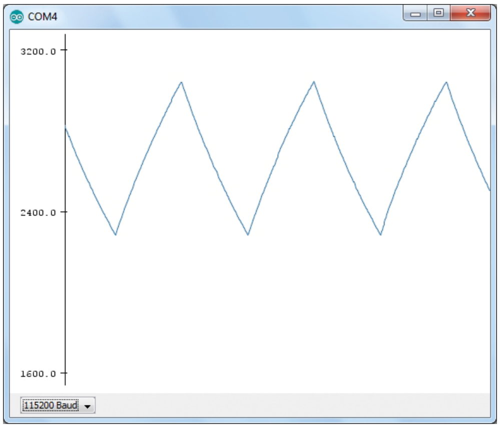


Figure 3. The filtered signal.

Faster sampling by buffering

If you are minded to reduce the time taken by serial data, it is only the acquisition time of the A-to-D converter that limits the achievable sampling rate. So you create a Data Array, fill it with data measurements and send these off to the PC. But it's exactly here that problems arise that you had not reckoned with. Although the controller is well endowed with RAM, an Array of the type `int d[100]` is pushing the limits, because the Microbit Runtime does not have much spare capacity. But if you want to use the serial plotter, there should already be 500 measured values.

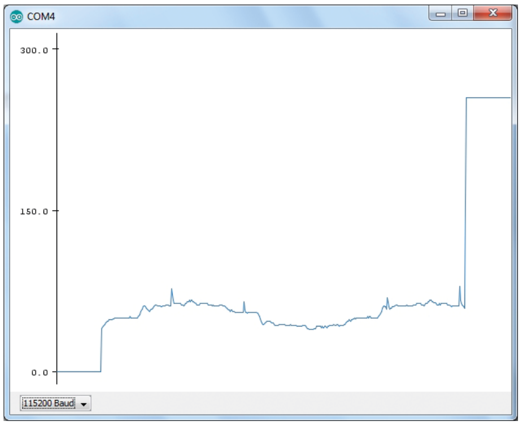


Figure 4. Measuring with a higher sampling rate.

You definitely need to be aware that the type of `int` in a 32-bit system has a size of four bytes. Accordingly we have 400 bytes at our disposal. Therefore we use a Byte Array with 400 bytes using `char d[400]`. By dividing by 4, the 10-bit data of the A-to-D converter becomes an economical 8 bits.

```
//Fast Scope
#include "MicroBit.h"
MicroBit uBit;

int main(){
  char d[400];
   uBit.init();
   MicroBitSerial serial(USBTX, USBRX);
   uBit.io.P1.setAnalogValue(512);
   uBit.io.P1.setAnalogPeriodUs(2000);
   while (1) {
     if(uBit.buttonA.isPressed()){
         for(int i = 0; i < 400; i++){
              d[i]  =  uBit.io.P0.getAnalogValue()/ 4;
              }
         for(int i = 0; i < 50; i++) uBit.serial.printf("%d\r\n", 0);
         for(int i = 0; i < 400; i++){
              uBit.serial.printf("%d\r\n", d[i]);
              }
           for(int i = 0; i < 50; i++) uBit.serial.printf("%d\r\n", 255);
      }
      uBit.sleep(500);
   }
}
```

Listing 2. Rapid saving and subsequent transfer.

We now store and transfer 400 bytes (**Listing 2**). We are still 100 bytes short for filling the plotter. But we can make a virtue out of necessity and prefix a zero-bytes header and suffix a trailer having 255-bytes. This causes the serial plotter to always display the full measurement range and provides the viewer with clear visibility of the range limits. The reading in **Figure 4** depicts a 50-Hz signal with typical interference pulses, which you pick up with an exposed measurement lead. Because around 1.5 oscillations are shown, the measurement lasts around 30 ms, which means that with 400 data points a sampling rate of approximately 13 kHz can be deduced. Because the PWM frequency in this program was raised by 500 Hz, you can easily verify this with a signal of your own. **Figure 5** shows the PWM signal at the output a low-pass filter with 4.7 kΩ and 100 nF.

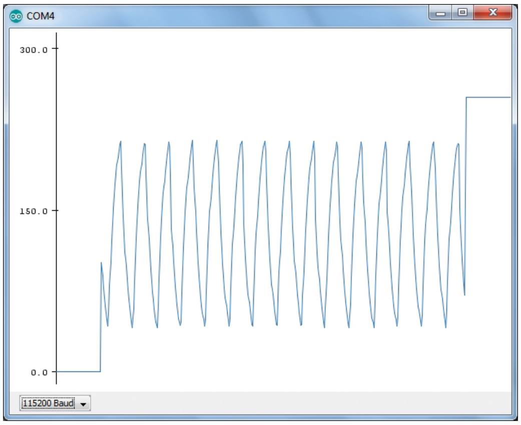


Figure 5. A 500-Hz sawtooth signal.

Wireless transfer of captured data

The BBC micro:bit is equipped with Bluetooth Low Energy (BLE). You will look in vain on the board for a dedicated chip for this, as the RF circuitry is already built into the microcontroller. The nRF51822 from Nordic Semiconductor was originally developed for applications such as wireless keyboards and mice, which did not call for extensive range but did have to use battery power economically. The BBC micro:bit takes advantage of these capabilities. You can power the board using a 3-V battery and then dispense with even the USB cable. This makes the system viable even for long-term applications powered by batteries. The shortform lineup of its main features speaks for itself:

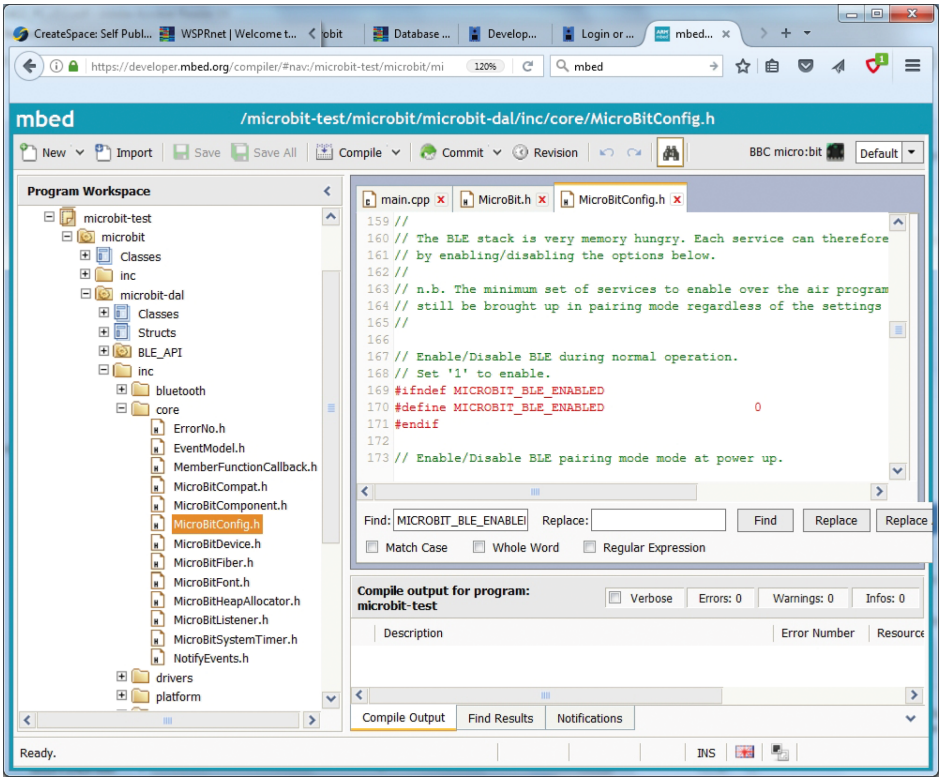
- 2.4-GHz transceiver
- –93 dBm sensitivity in Bluetooth® low energy mode
- 250 kbps, 1 Mbps, 2 Mbps supported data rates
- Tx Power –20 to +4 dBm in 4-dB steps
- Tx Power –30 dBm whisper mode
- 13 mA peak Rx, 10.5 mA peak Tx (0 dBm)
- 9.7 mA peak Rx, 8 mA peak Tx (0 dBm) with DC/DC
- RSSI (1-dB resolution)
- ARM® Cortex™-M0 32-bit processor
- 275 μA/MHz running from flash memory
- 150 μA/MHz running from RAM
- Serial Wire Debug (SWD)

Programming with mbed enables the use of Bluetooth, allowing you to transmit data direct to a smartphone or tablet. Admittedly this calls for a pretty sizeable software stack and worse, it leaves you to develop the custom apps.

But there’s a far simpler way. You see, you can address the 2.4-GHz transceiver at a lower level, dispensing altogether with the complicated Bluetooth protocol. Making this possible is the MicroBitRadio support platform [6] with simple datagrams (text messages) that the transceiver can handle unaltered. One BBC micro:bit module sends a text message and all other modules in range can receive it. To make this work a default channel and a default transmit power have been defined, so you really don't need to concern yourself with anything. That makes this one of the simplest methods of transferring data without wires. At the same time it gives you the ability to make isolated (potential-free) measurements. A typical application might be an electrocardiogram device, as the electrical (galvanic) separation eliminates any troublesome electrical hum interference.

All you need is two BBC micro:bit modules. One is employed as the measuring instrument for transmitting the data, the second one is programmed as the receiver, for displaying the data or transferring it to a PC over a USB cable. To use MicroBitRadio in mbed you do need to deactivate Bluetooth Low Energy. There's a note about this in the BBC micro:bit documentation:

It is not currently possible to run the MicroBitRadio component and Bluetooth Low Energy (BLE) at the same time. If you want to use the MicroBitRadio functionality, you need to disable the BLE stack on your micro:bit by compiling the runtime with #define MICROBIT_BLE_ENABLED 0 in your inc/MicroBitConfig.h file.



```
//
// The BLE stack is very memory hungry. Each service can therefore
// by enabling/disabling the options below.
//
// n.b. The minimum set of services to enable over the air program
// still be brought up in pairing mode regardless of the settings
//

// Enable/Disable BLE during normal operation.
// Set '1' to enable.
#ifndef MICROBIT_BLE_ENABLED
#define MICROBIT_BLE_ENABLED                0
#endif

// Enable/Disable BLE pairing mode mode at power up.
```

Find: MICROBIT_BLE_ENABLEI Replace:

Compile output for program: microbit-test

Figure 6. Deactivating BLE.

It is not entirely simple to locate the correct point in the numerous files of the runtime system. The exact path is: *microbit\microbit-dal\inc\core\MicroBitConfig.h* (see **Figure 6**). In this file is an entry `MICROBIT_BLE_ENABLED 1`, in which you need to replace the 1 with a 0. In the next compilation that you make BLE is then deactivated, enabling you to use the simplified MicroBitRadio.

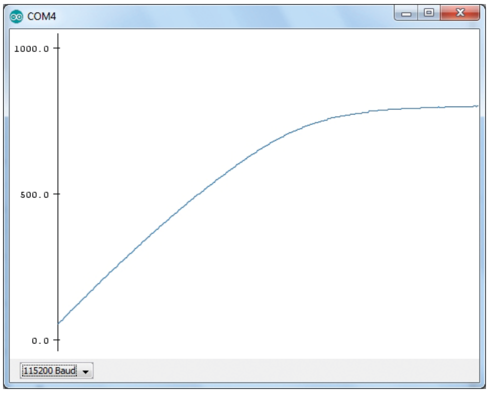


Figure 7. Data measurements sent by wireless link.

The program in **Listing 3** sends and receives datagrams of measurement data. The sender and receiver can therefore use the same program and also exchange data reciprocally. The sender executes a measurement at P1 and transmits the reading in mV. The receiver pass-es the received data forward via the USB connection. Nothing changes on the PC side. Here we can again make use of the serial plotter. **Figure 7** illustrates a measurement reading. The transmitter is battery-driven and is located three meters (10 feet) away from the re-ceiver. The link works for up to approx. 10 m (30 ft.). A 10-µF electrolytic was attached to analog input P1. This was charged up slowly using the 10-MΩ pull-up resistor provided on the PCB. You may notice some deviation from the normal charging curve, because this ca-pacitor had not been used for a long time. In a case like this just a low leakage current flows initially that increases only gradually. On the right-hand side of the diagram the measured voltage lies clearly below 1 V and rises only slowly.

```
//Radio Data
#include "MicroBit.h"
MicroBit uBit;


void onData(MicroBitEvent e)
{
    ManagedString s = uBit.radio.datagram.recv();
    uBit.serial.send (s);
    uBit.serial.send (" \r\n");
}

int main()
```

```
{
   uBit.init();
   uBit.messageBus.listen(MICROBIT_ID_RADIO, MICROBIT_RADIO_EVT_
        DATAGRAM, onData);
   uBit.radio.enable();
   char output[16];
   while (1) {
        int u = 3300 * uBit.io.P1.getAnalogValue()/ 1023;
        itoa (u, output);
        uBit.radio.datagram.send(output);
        uBit.sleep(100);
    }
}
```

Listing 3. Sending and receiving datagrams.

Mini-oscilloscope with LED display

An extremely basic oscilloscope is better than none at all and sometimes it's more important that the device is very small, standalone and easy to handle. Here we see measurement data displayed graphically on the LED display using 5×5 LEDs (**Listing 4**). Even if you are accustomed to a far more sophisticated instrument, you can definitely get results with this little alternative. It is quite remarkable, what is still discernible with such a simple 'scope.

```
//LED-Scope
#include "MicroBit.h"
MicroBit uBit;
int main()
{
   int y;
   uBit.init();
   uBit.io.P0.setAnalogValue(512);
   uBit.io.P0.setAnalogPeriodUs(500);
   uBit.display.enable();
   MicroBitImage image(5,5);
   while (1) {
        for(int x = 0; x < 5; x++){
            y =  4- (uBit.io.P1.getAnalogValue()/205);
            image.setPixelValue(x,y,255);
        }
        uBit.display.print(image);
        uBit.sleep(500);
        image.clear();
    }
}
```

Listing 4. Using the LED display.

Once again the mini-oscilloscope uses Port 1 as an analog input and additionally employs Port 0 as a PWM output. With a repetition rate of 500 µs, an output signal with a frequency of 2 kHz is generated. A direct connection to the measurement input shows the limits of the A-to-D converter (**Figure 8**). The sampling time is obviously too long to display sharp edges of the PWM signal. The limiting frequency of this simple oscilloscope is therefore somewhere below 10 kHz. This is not enough for an RF lab but probably adequate for many simple measurements and experiments.

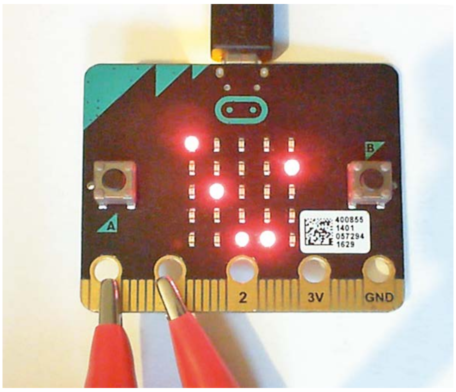


Figure 8. Measuring the internal test signal.

Web Links

[1] www.elektormagazine.com/160273
[2] www.elektormagazine.com/160384
[3] https://developer.mbed.org/
[4] https://lancaster-university.github.io/microbit-docs/ubit

Chapter 53 • RF Detector using an Arduino

Programmed in Bascom

Add an RF receiver to a microcontroller and you open up many possibilities. It doesn't need to be complicated, for most applications you can get away with just using a simple diode detector; in fact we can even use a plain old LED for the job! Just take an Arduino Uno with an extension shield and you've already got all the hardware you'll need. Now add the code, programmed in Bascom.

If you delve back into the history of RF receiver designs you are sure to find reference to the simple detector receiver using a germanium (Ge) diode. Microcontrollers have inputs able to measure analog signals so there is no reason why we can't just hook up the output of the detector to one of the A/D inputs on the microcontroller and see if we can pick up some signals.

Once you've got it connected you have already built a signal strength meter. The readings you make can be useful, for example, in the world of amateur radio to tune an aerial. The resonant frequency of the detector circuit needs to be adjusted to be in the range of the measured frequency. You can also tune to a nearby medium wave station and check the received signal strength in your area. You might be surprised to detect things you weren't expecting. In my study at home I can tell when a streetcar passes by because it noticeably affects the received signal strength of an AM station around 720 kHz.

There are two main reasons for using a germanium diode in the detector circuit shown in (**Figure 1**). First off it has a low forward-conduction voltage. This means that signals as low as 100 mV will produce an output signal. Secondly its reverse-voltage resistance is not especially high, that's useful to dissipate any charge accumulating on the output capacitor.

If you replace it with a silicon diode such as a 1N4148, for example, you will need to receive a much stronger signal before you start to see an output signal from the circuit. You will also need to add some form of output load such as a 1 MΩ resistor.

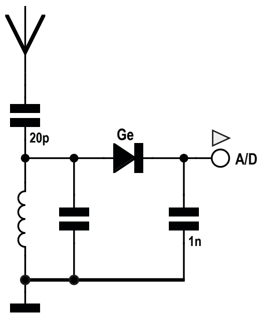


Figure 1. The classic detector receiver.

That doesn't mean that silicon diodes are all bad for this sort of application. You can make use of a bias voltage to offset the diode's conduction threshold. **Figure 2** shows such a circuit without any form of tuned circuit for frequency selection so it's got a very wide bandwidth. With no received signal you can measure a voltage of around 0.6 V at the diode. When a signal is received this voltage level drops noticeably. This circuit works well with RF signals of around 100 mV. It can be used as an RF signal monitor and works across the entire short wave band without the need for any selector.

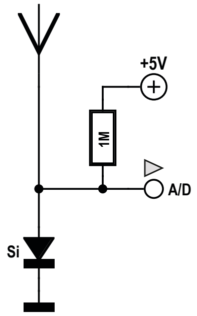


Figure 2. Si diode with bias voltage.

An LED as a detector diode?
Would it be possible to use the LED that's already fitted to the Elektor Extension-Shield? We already know that LEDs also function as a photo diode, a voltage stabilizer, limiter and a varicap, surely we can get one to work as an RF detector as well. LED1 on the shield is already connected to the analog input ADC2. There is also a 1 kΩ resistor in series with the LED but that should not give a problem. The internal 30-kΩ pullup resistor can be configured to provide a bias voltage, perfect; we really don't need anything else to build the circuit (see **Figure 3**).

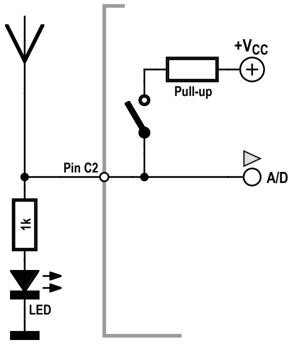


Figure 3. An LED RF detector.

And so to the software! Now we have already built our little RF test lab (**Listing 1**) and can begin programming it to generate an RF signal. A 1 MHz square wave signal is output from pin B1.

```
'----------------------------------------------------
'UNO_RX1.BAS   B1 RF out, C2 RF in
'----------------------------------------------------
$regfile = "m328pdef.dat"     ' ATmega328p
$crystal = 16000000           ' 16 MHz
$baud = 9600
$hwstack = 16
$swstack = 16
$framesize = 16

Dim D As Word

Config Lcdpin = Pin , Db4 = Portd.4 , Db5 = Portd.5 , Db6 = Portd.6 , Db7 =
Portd.7 , E = Portd.3 , Rs = Portd.2
Config Lcd = 16 * 2
Cls
Cursor Off

Config Adc = Single , Prescaler = 64 , Reference = Avcc     ' 5V

Config Timer1 = Pwm , Prescale = 1 , Pwm = 10 , Compare A Pwm = Clear Up
Tccr1a = &B10000010           ' Phase-correct PWM, Top=ICR1
Tccr1b = &B00010001           ' Prescaler=1

D = 8                         ' 1 MHz
Icr1 = D
Ocr1a = D / 2
Portc.2 = 1

Do
  D = Getadc(2)
  Print D
  Locate 1 , 1
  Lcd D
  Lcd "    "
  Waitms 500
Loop
```

Listing 1. Measuring the LED voltage [1].

The program enables the pull up resistor on port pin C2 and continually measures the voltage on ADC2. Here you can read the value 410 which corresponds to a voltage at the LED of around 2 V. Connect a 10 cm length of insulated wire to C2 to act as an antenna. Attach a bare wire at B1 and hold the other end of it. Your body is now connected to the signal and becomes an antenna for the signal which can be picked up by a normal AM radio receiver. Now take the insulated wire on C2 and couple it to the RF signal. You will see the measured value drop to below 400. It's interesting to note that the measured value remains at a constant level. The RF oscillations are not registered because the sample rate of the A/D converter is relatively low and this produces an averaging effect on the measured signal. The measurement shown is the average voltage across the LED which drops as the RF signal gets stronger.

An Integrating Detector

Maybe we could do with a bit more gain? This could be achieved in principle by using a higher value of pull up resistor. Even better would be to just switch on the pull up resistor briefly and then go into a high impedance state before the measurement is made. The charge across the LED will dissipate during the measurement period. The presence of an RF signal will increase the discharge rate.

You can think of the LED junction as having a small value of capacitance. Every peak of the received RF signal brings the LED briefly into conduction which has the effect of reducing the charge on its capacitor slightly. The value of this capacitor is only a few picofarads. That means you only need a very low level of RF current to produce a measurable effect. To give better sensitivity you can increase the delay between turning off the pull up and making a measurement. This will however make the circuit sensitive to low frequency signals which can cause interference. For this reason it's better to make the measurement quickly after the pull up has been turned off using a relatively short sample time (`Prescaler = 8`).

The program in **Listing 2** performs averaging on the measurement samples to determine the zero level `D0`. By comparing the zero level with the input we can find out if an RF signal has been received. A drop of three A/D steps of the LED voltage is recognized as the threshold to indicate a signal has been received. Tests indicate that a received RF signal of around 50 mV is necessary. To flag this event, LED2 on the Elektor Shield is lit and a tone is produced at B2. You can hook up a simple piezo loudspeaker here to make the tone audible. The signal strength is also transferred serially but not available on the LCD due to timing constraints. When an RF signal is received at the input an (almost) constant tone will be audible at the output.

```
'-------------------------------------------------------
'UNO_RX2.BAS  B1 RF out, C2 RF in
'-------------------------------------------------------
$regfile = "m328pdef.dat"       ' ATmega328p
$crystal = 16000000             ' 16 MHz
$baud = 9600
$hwstack = 16
$swstack = 16
$framesize = 16

Dim D As Word
Dim D0 As Word
Dim N As Byte

Ddrb.2 = 1
```

```
Config Lcdpin = Pin , Db4 = Portd.4 , Db5 = Portd.5 , Db6 = Portd.6 , Db7 =
Portd.7 , E = Portd.3 , Rs = Portd.2
Config Lcd = 16 * 2
Cls
Cursor Off

Config Adc = Single , Prescaler = 8 , Reference = Avcc      ' 5V

Config Timer1 = Pwm , Prescale = 1 , Pwm = 10 , Compare A Pwm = Clear Up
Tccr1a = &B10000010             ' Phase-correct PWM, Top=ICR1
Tccr1b = &B00010001             ' Prescaler=1

D = 4                           ' 2 MHz
Icr1 = D
Ocr1a = D / 2

D = 0
For N = 1 To 50
   Portc.2 = 1
   Waitus 100
   Portc.2 = 0
   D = D + Getadc(2)
Next N
D0 = D / 50

Do
  Portc.2 = 1
  Waitus 100
  Portc.2 = 0
  D = Getadc(2)
  If D < D0 Then
    D = D0 - D
    If D > 2 Then
      Print D
      'Locate 1 , 1
      'Lcd D
      'Lcd "    "
      Sound Portb.2 , 20 , 4000   ' LED2 and Piezo
    End If
  End If
Loop
```

Listing 2. RF receiver with sound output [1].

With this set up you can send and receive Morse characters. For this the output frequency was raised to 2 MHz to give increased range. For tests you can dab your finger on the output pin B1 to send Morse signals. The other hand should close enough to the receiving antenna to ensure reception of the signals. You can of course set up two Arduinos so that signals can be sent and received.

For test purposes a sine wave generator was used as a RF generator with a finger touching the output signal so the body works as an antenna, it was possible to achieve a range of around 39 inches using an output signal of 16 V_{pp} and a frequency of 1 MHz.
So what can the circuit be used for? Without any additional circuitry you can use it to track down sources of RF interference. Many switch mode power supplies are guilty of high levels of RF noise. Energy saving lamps and conventional fluorescent lamps are also culprits when it comes to unwanted RF noise especially in the medium wave band. All of these sources can be easily detected with the integrating LED detector you have built from an Arduino.

By the way, don't overlook the other useful components on the extension shield (**Figure 4**). There are two push buttons and a pot to play with. With a little ingenuity you could make use of these to provide sensitivity adjustment, a Morse key, call button, mute and standby…

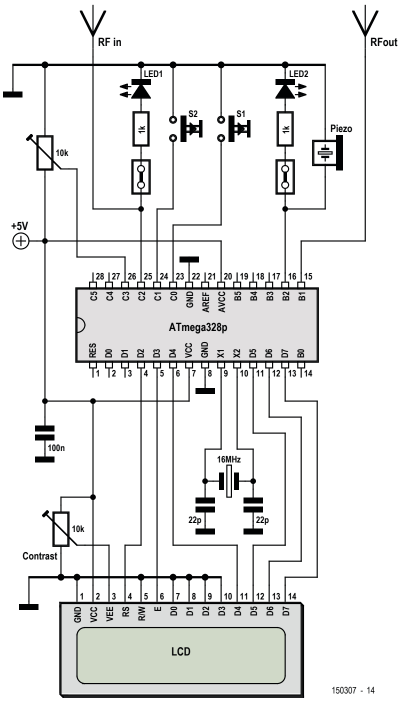

RF in
RFout
LED1
LED2
S2
S1
1k
1k
Piezo
10k
+5V
ATmega328p
C5 C4 C3 C2 C1 C0 GND AREF AVCC B5 B4 B3 B2 B1
RES D0 D1 D2 D3 D4 VCC GND X1 X2 D5 D6 D7 B0
100n
16MHz
22p
22p
10k
Contrast
GND VCC VEE RS R/W E D0 D1 D2 D3 D4 D5 D6 D7
LCD
150307 - 14

Figure 4. Except for the two antennae and the piezo beeper everything else is already on board the Extension shield.

Web Link

[1] www.elektormagazine.com/magazine/elektor-201601/28735

Chapter 54 • Resistance Measurement with the Arduino

Great for testing humidity sensors
Next to the oscilloscope, the ohmmeter is the most widely used T&M device in the electronics lab. Using this you can measure component characteristics, trace wires, find errors in circuits and evaluate many types of sensor. But a normal ohmmeter or multimeter cannot always fulfill the demands placed upon it, for example when we need to measure alternating current (AC). This is when a microcontroller can help. Sure, another opportunity to use the Arduino Uno, our Extension Shield and Bascom!

The starting point for this project was some newish resistive air humidity sensors, whose resistance varies in the range 1 kΩ to 10 MΩ. The datasheet states expressly that they must be measured using AC. And that's precisely what a regular ohm meter cannot do.

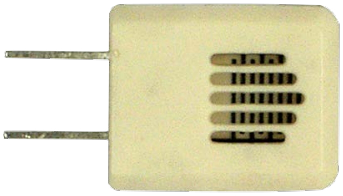


Figure 1. A resistive humidity sensor.

A representative example of these resistive sensors (**Figure 1**) is the HCZ-H8A(N), which you can obtain from Farnell, Conrad, Mantech and other suppliers. The datasheet can be found here [1]. Normally these sensors are driven using an AC voltage of 1 V_{eff}. If you connect them to a signal generator using a voltage divider, you can view the result very clearly on the oscilloscope (**Figure 2**). Put your hand close to the sensor and the humidity increases; you can watch the resistance fall and the signal voltage rise at the input of the oscilloscope.

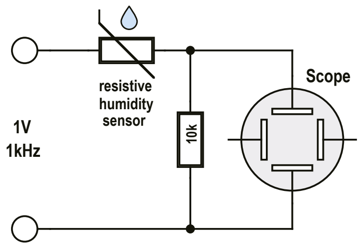


Figure 2. The first test.

Why can't you use direct current (DC) for this? The answer is all to do with polarization. Water molecules, as you probably know, have polar characteristics; in other words they are more positive on one side and more negative on the other. Gradually they will align according to the DC applied and change resistance in the process.

The same effect arises when you measure the conductivity of water; here too we must use only AC. A long time ago I even observed this effect while measuring the conductivity of wood as a function of humidity. The resistance starts off low and then rises slowly. A pair of stainless steel nails pushed into damp wood even behave something like a battery that you can charge up. In those days I was amazed. Since then, however, I found out that this is the same principle you have in dual-layer capacitors, also known as supercapacitors or GoldCaps.

Resistance measurement
How do we get round the fact that microcontrollers prefer DC? Or that handling the huge measurement range involved is no easy task for a 10-bit A-to-D converter? Consequently thoughts turned towards *R-C* elements and time measurement. Oh yes, it would be very handy if we could get away with using only one pin of the microcontroller too. The result is the simple measurement circuit using the Port pin PB3 (**Figure 3**).

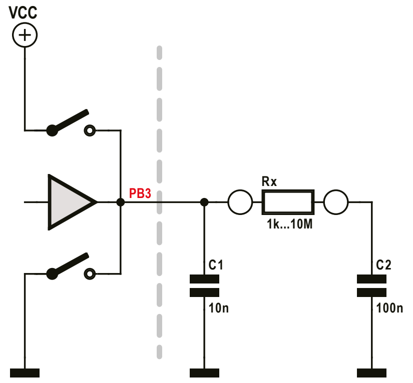


Figure 3. Resistance measurement using only one Port pin.

Rx and C1 form an *R-C* element, whose time constants need to be measured. C2 works in the background like a backup battery that provides the charging current required. All the same, because C2 is also charged via the resistance under measurement, the DC flowing through Rx in the middle is zero. The actual measurement process (**Listing 1**) proceeds in three phases:

- **Charging.** The Port is connected to V_{CC} via a low resistance, so that C1 charges immediately and C2 more sluggishly.
- **Discharging.** The Port is connected to GND very briefly, precisely long enough to discharge C1 but short enough to leave C2 retaining almost full voltage.
- **Measurement.** The Port is configured as a high-impedance input. We then measure the time taken for the input to revert to 1 (High).

```
Count = 0
Portb.3 = 1
Ddrb.3 = 1         ' Charge
Waitms 500
Portb.3 = 0         ' Discharge
Waitus 10
Ddrb.3 = 0
Do                'Counter
  Count = Count + 1
Loop Until Pinb.3 = 1 Or
Count.15 = 1
Portb.3 = 1
Ddrb.3 = 1
Print Count
Locate 1 , 1
Lcd Count
Lcd "       "
```

Listing 1. Measuring the time to charge.

The method produces counts that are within broad limits proportional to the resistance. With the input unconnected the test result is limited to 32,768 maximum.

There is still one small problem in that measurement malfunctions with resistances of significantly less than one k-ohm. The reason is evidently that with very small resistances the larger capacitor is discharged immediately during the short discharge pulse, meaning the charging source is now lacking.

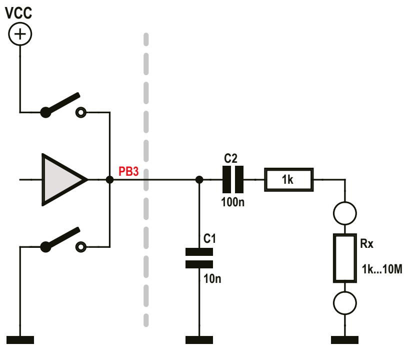


Figure 4. Optimized measurement circuit.

Circuit optimization

For this reason it is better to add an extra resistor of 1 kΩ in series, which you can easily subtract later on. Because Rx and C2 are connected in series, you can also change these around (**Figure 4**). This works better because the item under measurement is now connected at one end to ground. The 'no DC' rule still applies, so it's AC measurements only. The same method should be usable for resistive humidity sensors. The small number of components involved can be soldered to a piece of header connector strip (**Figure 5**), for connection to the corresponding Arduino socket strip. This makes our test adapter a kind of Mini Shield. For indicating the value measured we use once more the display on the Elektor Extension Shield [2], on which the Arduino connector strips are duplicated. When you plug the Arduino Uno, the Extension Shield and the test adapter all together, the whole combination looks like our heading photo.

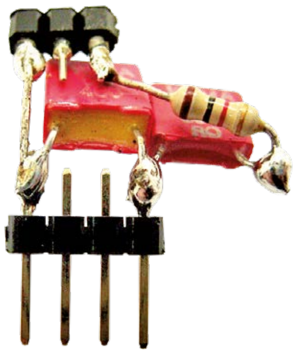


Figure 5. Mini Shield for resistance measurement.

Using linear conversion (**Listing 2**) we can output the resistance in kΩ. The result is not to be sniffed at: between 1 kΩ and 1 MΩ we achieve really good linearity of around 5 %. In the range up to 10 MΩ the variance is a bit larger. In any case, absolute accuracy is also dependant on the tolerances of the capacitors and the exact switching threshold of the input. The method is not noted primarily for great accuracy, rather for its broad measurement range and simple circuitry.

```
'Calculate Resistance
If Count > 4 Then Count = Count - 4 Else Count = 0     ' -1 kOhm
Resistance = Count * 14
Resistance = Resistance / 40
Print Resistance            '...kOhm
Locate 2 , 1
Lcd Resistance
Lcd " kOhm  "
```

Listing 2. Conversion into k-Ohms.

Logarithmic measurement

Resistive air humidity sensors possess more or less exponential characteristic curves (**Figure 6**). You need to measure the resistance and then express it logarithmically. For the Arduino this is an easy exercise. The calculation is carried out in several steps (**Listing 3**), in which we produce the natural logarithm in Bascom using the Log function. The following formula delivers the air humidity in percent from the value Count:

$$\text{Air humidity [\%]} = (103 - 8.9 \times \ln(\text{Count}))$$

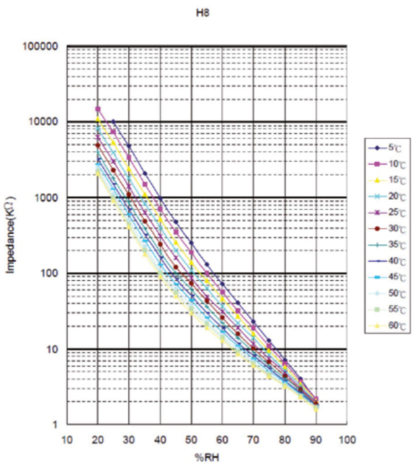


Figure 6. Sensor resistance in relation to humidity and temperature (Source: Datasheet [1]).

```
'Calculate Humidity
F = Count
F = Log(f)
F = F * 8.9
F = 103 - F
Humidity = Round(f)
Locate 1 , 6
Lcd Humidity
Lcd "%      "
```

Listing 3. Conversion into relative air humidity.

Errors arise from a certain curvature of the characteristic line in the logarithmic scale. The values in the formula are selected so that the smallest errors occur at 40 % and 80 %. The largest deviation arises from variations among different examples of sensor, however. Calibration is costs money and plenty of simple humidity measurement devices for sale are equally imprecise.

Moreover, temperature dependency is not considered; a room temperature of 20 °C is assumed instead. Nevertheless you can see variations in air humidity very clearly. Unavoidable measurement error is due least of all to any inaccuracy in resistance measurement, since in the logarithmization process these errors merge into virtually nothing.

As these lines are written, it is cold outdoors and warm inside. The air indoors is fairly dry and the sensor (**Figure 7**) is indicating 40 %. That could well be right, according to one of my hygrometers. My flowers urgently need to be watered. Once I do this, the air humidity rises immediately to 41 % and after a while to 42 %. Larger variations arise when you put your hand close to the sensor. With a finger placed either side of the sensor, it shoots up to over 80 % quite rapidly.

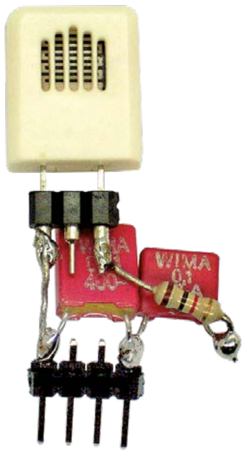


Figure 7. Mini Shield with sensor attached.

By the way, the circuit can of course be used as an ohm meter. All values between 1 kΩ and a few MΩ can be displayed reliably (**Figure 8**).

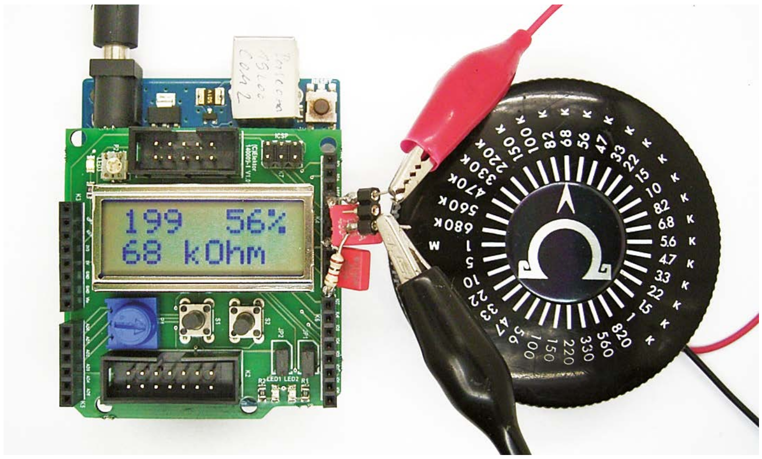


Figure 8. Ohmmeter test.

Web Links

[1] www.farnell.com/datasheets/1355478.pdf

[2] www.elektor-magazine.com/140009

Chapter 55 • Arduino-Powered AM Transmitter

Broadcast the inductive way on Medium Wave
Old tube radios have a very special charm. They look handsome, sound great and frequently arouse pleasant memories too. For this reason they have many fans, who collect, repair and lovingly restore radios of this kind. The only downside is the dwindling number of radio stations that still transmit on the classic AM frequencies.

Beside me on my workbench is an old tube radio dating back to 1957, which I still switch on regularly. And yes, on the medium wave bands. It's true that amplitude modulation on the medium wave makes do with a restricted bandwidth, but this gives reception a uniquely warm sound. And in the evening, one can pull in countless stations from across the borders. The only snag is that there are fewer and fewer of them these days. Many local stations have already closed down altogether and even the BBC has abandoned medium wave transmissions. This calls for a new use of these now-vacant frequencies. Our own private AM station is what we need! And that's just what this article describes, enabling me to receive good old BBC by Internet radio and rebroadcast it on medium wave, just as I listened in the olden days.

But is this lawful? Pirate transmitters are not favored generally but there is a perfectly legal solution. You need to broadcast the signal inductively (just as some college and hospital broadcasting systems do) over a restricted range. In this way there is no risk of upsetting your neighbors. This is permitted in my country under the 'General allocation of frequencies in the range 9 kHz to 30,000 kHz for inductive radio applications' (Vfg. 4/2010 amended by Vfg. 4/2014); other countries have comparable allocations. For the frequency range 148.50 kHz to 5,000 kHz you are confined to a limit of 15 dBµA/m at a distance of 10 meters (30 feet).

Magnetic field strength is admittedly not easy to measure but you can calculate it from the antenna current flowing in an inductive loop. If you use a wire loop of radius r = 1 m (i.e. 2 m diameter) with just one turn and restrict the RF current to 0.3 mA, then the magnetic field strength at a distance of 10 meters will amount to only –15 dBµA/m. This will keep everything legal. In practice your 'transmit antenna' will be fixed to the underside of the table or hung on the wall close to the radio. Inside the wire loop reception will be good but outside this the field strength drops off rapidly. You could also create two coils from the same length of wire, concentrating the field strength inside a smaller area, with even faster fall-off at distance.

So we're going to construct an MW AM transmitter, but how? A transmitting tube is driven hard by the carrier signal (Class C final stage), with its anode voltage modulated by the audio signal. For flea power we can do the same thing with a transistor, either bipolar or FET. All we need then is a crystal-controlled RF signal on the appropriate frequency.

Microcontroller RF source

Bring on the microcontroller. The Arduino Uno has a built-in crystal that we can use to derive frequencies in the medium wave band by programmed division factors. This elegant and straightforward solution has the minor disadvantage that we cannot transmit in the 10 kHz spacing normal in the USA (9 kHz in other territories). Despite this you'll have no difficulty in finding a suitable unoccupied frequency. A small control module with a frequency display is easy to produce with the Elektor Extension Shield [1].

Even better, the Uno already has a transmitter final stage on board! By using a little trickery, any Port pin of your choice can be used for this. Instead of addressing the Port register we use the Data Direction register. Take, for example, the PB0 connector. With PORTB.0 = 0 and DDRB.0 = 1 (output) this puts in low-ohmic (Low) state. If we now switch DDRB.0 = 0 (normally you do this when you wish to read in digital signals), the Port is set High. These two states are equivalent to a FET with an open drain. We can now apply any 'drain' voltage of our choice to PB0 (so long as the voltage is within the range 0 V to 5 V). When DDRB.0 = 1, it is grounded, in sync with the phase of the RF signal. Naturally this drain voltage can be modulated, for example with an audio signal.

Now we have everything necessary! We just need to sort out a couple of little details and do a bit of programming.

Circuit

The headphone output of a typical signal source (CD player, PC sound card, etc.) normally produces up to 1 Vrms, which is almost 3 Vpp. It definitely does not want to be much more than this, because the final stage must be driven within the range 0 to 5 V. This makes a modulation amplifier unnecessary. A couple of resistors and capacitors suffice to 'condition' the audio signal (**Figure 1**, right-hand side). The left-hand side of the schematic shows the ATmega328 of the Arduino Uno and the components located on the Extension Shield. You could also construct the whole affair on an experimenter's breadboard of course.

At the antenna output we have a simple low-pass filter for suppressing the harmonics. Theoretical considerations and actual measurements both indicate somewhat less than 0.3 mA of current should flow in the loop antenna. You will be well within the guidelines so long as the total length of wire in the loop antenna is not greater than 3 meters. For your first tests a far smaller wire loop will be fine (using just 20 cm of wire for example).

The output from a regular sound card (about 3 Vpp) will give us a modulation factor of around 50%, which conveniently rules out the risk of overmodulation. As AM signals are transmitted only in mono, we combine the left and right-hand channels. This will produce absolutely clean modulation without any kind of distortion. When heard on a tube radio you will enjoy the agreeable, warm sound of the old days.

Software

To generate the carrier frequency we simply toggle the corresponding DDRB bit on and off fairly rapidly. To perform the simple Do Loop (**Listing 1**) programmed in Bascom, the ATmega328 takes seven clock cycles. Add three NOPs and we have then divided the clock frequency of 16 MHz by 10. This means we can transmit on the upper end of the medium wave at 1.6 MHz.

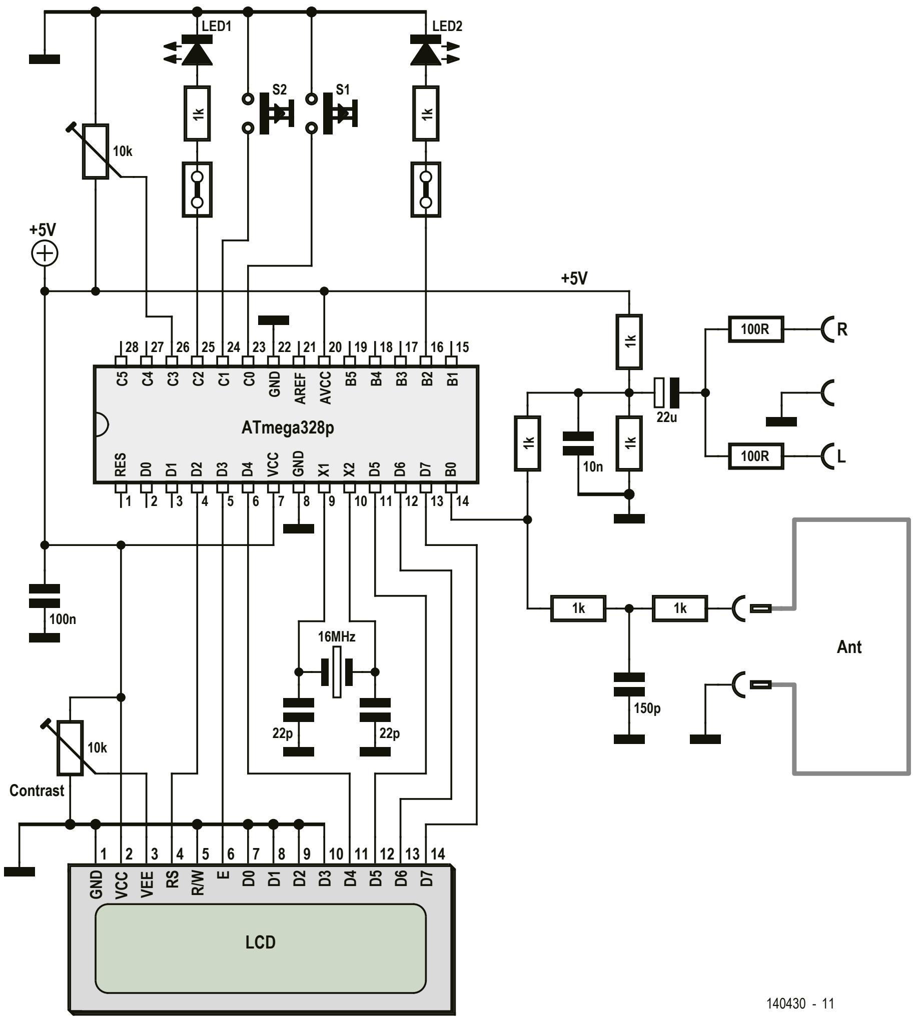


Figure 1. Circuit of this AM transmitter. All components on the left-hand side are located on the Arduino Uno and the Elektor Extension Shield.

```
D11:
Do                '/11, 1454 kHz
  Ddrb.0 = 1
  nop             '+4 nop
  nop
  nop
  Ddrb.0 = 0
  nop
Loop

D10:
Do                 '/10, 1,6 MHz
  Ddrb.0 = 1
  nop              '+3 nop
  nop
  nop
  Ddrb.0 = 0
Loop
```

Listing 1. Producing the RF signal.

If you add in another NOP command, the loop will divide by 11, producing 1454 kHz. And so on. In our program, which as always can be downloaded from the Elektor Magazine website [2], every frequency gets its own loop, each with a label stating the relevant division factor. The lowest frequency of 500 kHz is produced using D32 (16 MHz divided by 32). Incidentally, the NOPs are arranged so that the pulse/pause ratio is as close as possible to equal. By its very nature a symmetrical square wave produces the lowest number of harmonics.

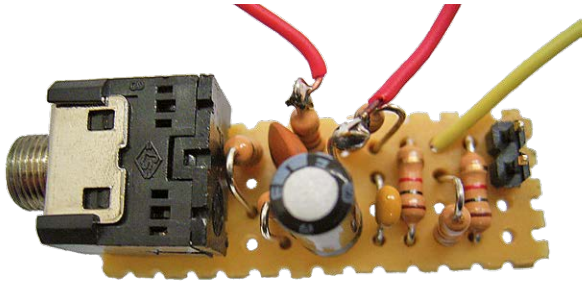


Figure 2. These additional components are located on a small piece of perf board.

A total of 22 different medium wave frequencies can be selected. At the same time the frequencies at the lower end (500 kHz, 516 kHz, 533 kHz, etc.) are closer together than at the upper end.

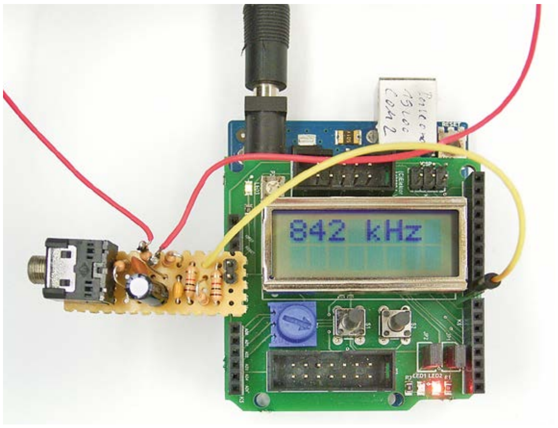


Figure 3. The transmitter with Arduino and display.

Operation

Once in the (RF) loop there is no going back. Of course you could come up with some solution involving an Interrupt but the following operating procedure is simpler (**Listing 2**). First you use the pot to select a suitable frequency, confirmed by the display. Then you press button S1 to switch on the transmitter. This also lights up the LED2, indicating 'on the air'. At all times the display shows the last frequency selected. Any operation of the pot will have no action now and the frequency remains unaltered. This is the same as with the high-power tube transmitters of the olden days, on which you also could not just change frequency once up and running. If you nevertheless wish to alter the frequency, you need to press Reset first, which stops the transmitter. Now you can set a new frequency and press S1 to activate this.

```
Do
  U = Getadc(3)
  D = U / 46
  D = D + 10
  F = 16000 / D
  Locate 1 , 1
  Lcd F
  Lcd " kHz "
  Waitms 500
  If S1 = 0 Then
    Led2 = 1
    If D = 10 Then Goto D10
    If D = 11 Then Goto D11
    If D = 12 Then Goto D12
    If D = 13 Then Goto D13
    If D = 14 Then Goto D14
    If D = 15 Then Goto D15
    If D = 16 Then Goto D16
    If D = 17 Then Goto D17
    If D = 18 Then Goto D18
```

```
            If D = 19 Then Goto D19
            If D = 20 Then Goto D20
            If D = 21 Then Goto D21
            If D = 22 Then Goto D22
            If D = 23 Then Goto D23
            If D = 24 Then Goto D24
            If D = 25 Then Goto D25
            If D = 26 Then Goto D26
            If D = 27 Then Goto D27
            If D = 28 Then Goto D28
            If D = 29 Then Goto D29
            If D = 30 Then Goto D30
            If D = 31 Then Goto D31
            If D = 32 Then Goto D32
          End If
        Loop
```

Listing 2. Selecting the transmit frequency.

Web Links

[1] www.elektor-magazine.com/140009
[2] www.elektor-magazine.com/140430

Chapter 56 • Security Labels are the Key

Door-entry system using Bascom

As we saw in our recent microcontroller course, Bascom running on an Arduino Uno together with the Elektor Extension Shield builds a powerful and versatile environment that can be used for a wide variety of applications. Here we build a low-cost contactless door-entry system and use some clever software to evaluate the resonant behavior of a standard security label.

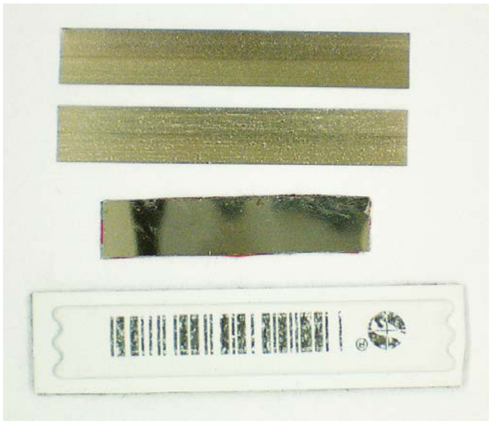


Figure 1. The AM label components, shown here are two resonators and a (shorter) magnetic strip.

It's increasingly common for shop owners to protect high value goods with security labels. When you buy an item that carries a label the cashier will rub it over a special area on the counter to deactivate it before packing it in a bag. Should the cashier forget to deactivate it, an alarm sounds when you pass through the exit. Maybe these labels can be reused for other purposes… To find out how they work The Net is, as usual a good place to start; try searching for 'acousto-magnetic strips'. This type of label is made up of two or more metal strips (**Figure 1**). A soft iron magnetic strip (sometimes two) of a defined length with a mechanical resonance of 58 kHz. One strip is made up of a special material called amorphous metal (also known as metallic glass or glassy metal). In order for the resonant strip to couple to an external alternating magnetic field the resonator needs to have a magnetic bias. This is taken care of by a second strip made of semi-hard magnetic material (**Figure 2**) which becomes magnetized. With these strip in close proximity the resonator vibrates in response to a 58 kHz alternating magnetic field and emits an alternating field that can be detected. This is the signal that triggers the alarm at the exit. Once the label has been degaussed at the checkout counter its magnetic field is neutralized and the label no longer responds to the 58 kHz field.

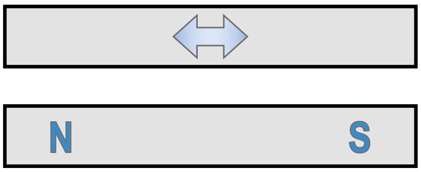


Figure 2. The AM label layout.

Reactivate

At home your purchase still carries the deactivated label. You can make it active again by stroking it with a strong magnet. This re-magnetizes it and converts it back into a resonator. A quick test rig can be made by connecting the output of a signal generator to a coil of about 100 turns and using a scope to explore the label's properties (**Figure 3**). It will typically show a resonance at 58 kHz. A deactivated label shows practically no peaking response to the signal. The mechanical resonance also produces a sound that can be detected with an ultrasonic detector.

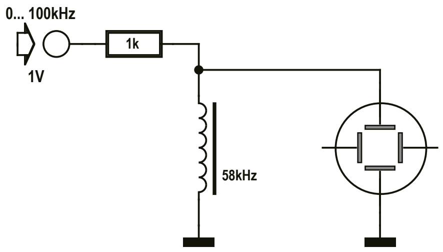


Figure 3. Measuring the resonant frequency using a signal generator and scope.

Armed with this knowledge you can begin to think of other possible applications of the technology. How about, for example, an electronic door-entry system that uses security labels as a key to gain access? The simplest way to build such a system would be to use a microcontroller like the Arduino Uno. Thinking over the hardware requirements you might assume that a DDS generator would be needed to generate the signal and then a complex detector/rectifier so that the controller can measure the signal. In practice we can simplify things. A controllable oscillator with sufficient accuracy can be made using the timer output OCC1A of the ATmega328 controller. A rectifier to measure the signal will also not be needed providing an AC signal no larger than about 0.3 V_{peak} is applied to the analog input, sampled often enough and the measured values averaged. All negative signals will have a value of zero so this gives us a signal detector with almost ideal properties. The complete circuit (**Figure 4**) is easily implemented with an Arduino Uno and an Elektor Extension-Shield [1]. This shield has the correct type of LCD to allow the display of measured values. The 'unlocked' signal of the electronic lock is output on pin PC2, indicated by the state of LED1 on the shield. In addition to this you just need one resistor and a suitable coil. The coil we use here (**Figure 5**) is a medium waveband antenna coil from an old AM radio fitted with a 10 mm diameter ferrite rod. A little bit of pressure squashed the coil slightly so that the label slides neatly inside. The same coil has also made an appearance in another Elektor article; it was used in the preselector of the Elektor SDR [2]. You can wind your own coil here using 80 to 100 turns of 0.2 mm (approx 32 AWG) enamel coated wire or salvage a winding from a junked radio. It's important to ensure that the coil has a low value of capacitance and has approximately the right value of inductance. Make sure that it doesn't exhibit self-resonance in the frequency range of interest.

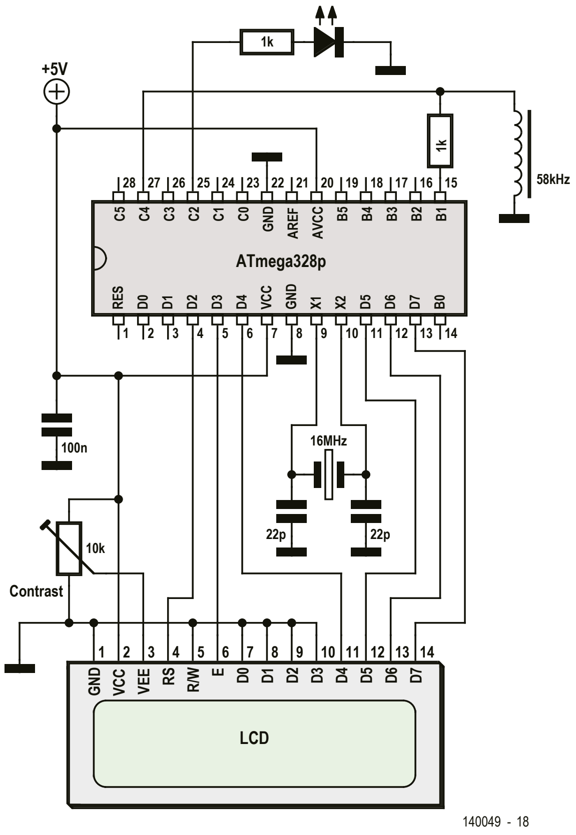

+5V
1k
1k
58kHz
ATmega328p
C5 C4 C3 C2 C1 C0 GND AREF AVCC B5 B4 B3 B2 B1
RES D0 D1 D2 D3 D4 VCC GND X1 X2 D5 D6 D7 B0
100n
16MHz
22p
22p
10k
Contrast
GND VCC VEE RS R/W E D0 D1 D2 D3 D4 D5 D6 D7
LCD
140049 - 18

Figure 4. Values displayed on the ATmega328.

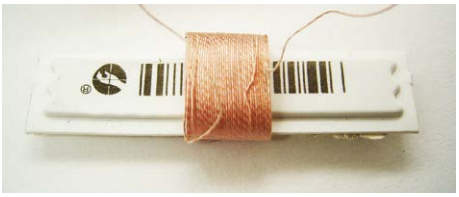


Figure 5. The medium-wave coil.

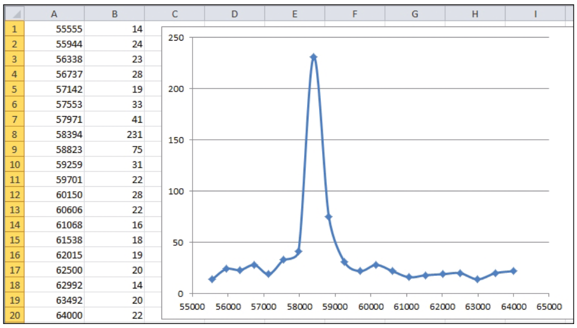
Software

The software (**Listing 1**) is written in Bascom and uses Timer 1. The program generates a rising series of 20 frequencies centered on 58 kHz. At each frequency 64 measurements are made using ADC4 and the values averaged. The resulting values are then stored in an array. During measurement the values of frequency and signal amplitude are sent to a PC using the serial interface. Here a graph of the resonance characteristics can be plotted (**Figure 6**). It is also possible to display the values on an LCD by removing the comment marks inserted in front of the LCD commands in the code listing.

```
'-----------------------------------------------
'UNO_AMetikett.BAS  58 kHz
'-----------------------------------------------
$regfile = "m328pdef.dat"             'ATmega328p
$crystal = 16000000                   '16 MHz
$baud = 9600
$hwstack = 32
$swstack = 32
$framesize = 64

Dim D As Long
Dim F As Long
Dim N As Byte
Dim U As Word
Dim Um As Word
Dim Ux(50) As Word
Dim I As Word

Led1 Alias Portc.2
Led2 Alias Portb.2
S1 Alias Pinc.0
S2 Alias Pinc.1
Portc.0 = 1                           'Pullup
Portc.1 = 1
Config Led1 = Output
Config Led2 = Output

Config Lcdpin = Pin , Db4 = Portd.4 , Db5 = Portd.5 , Db6 = Portd.6 , Db7 =
Portd.7 , E = Portd.3 , Rs = Portd.2
Config Lcd = 16 * 2
Cls
Cursor Off

Config Timer1 = Pwm , Prescale = 1 , Pwm = 10 , Compare A Pwm = Clear Up

Tccr1a = &B10000010     'Phase-correct PWM, Top=ICR1
Tccr1b = &B00010001     'Prescaler=1
```

```
Config Adc = Single , Prescaler = Auto , Reference = Internal_1.1
Start Adc

Do
  For I = 1 To 20
    D = 145 - I
    F = 16000000 / D
    F = F / 2
    Print F ;
    Print "  ";
    Print Chr(9);
   ' Locate 1 , 1
   ' Lcd F
   ' Lcd " Hz      "
    Icr1 = D
    Ocr1a = D / 2
    Waitms 10
    Um = 0
    For N = 1 To 64
      U = Getadc(4)
      Um = Um + U
    Next N
    Um = Um / 32
    Ux(i) = Um
    Print Um
   ' Locate 2 , 1
   ' Lcd Um
   ' Lcd " mV   "
   ' Waitms 500
  Next I
  Max(ux(1) , Um , I)
  D = 145 - I
  F = 16000000 / D
  F = F / 2
  Locate 1 , 1
  Lcd F
  Lcd " Hz      "
  Icr1 = D
  Ocr1a = D / 2
  Locate 2 , 1
  Lcd Um
  Lcd " mV   "
  If F > 57000 And F < 59000 And Um > 50 Then
    Waitms 50
    For N = 1 To 255
      U = Getadc(4)
      Um = Um + U
    Next N
    Um = Um / 127
    If Um > 50 Then Led1 = 1
    Lcd F
    Lcd " Hz      "
    Icr1 = D
    Ocr1a = D / 2
    Locate 2 , 1
    Lcd Um
    Lcd " mV   "
  Else
     Led1 = 0
  End If
  Waitms 1000
Loop
```

Listing 1. Resonant frequency measurement..



	A	B
1	55555	14
2	55944	24
3	56338	23
4	56737	28
5	57142	19
6	57553	33
7	57971	41
8	58394	231
9	58823	75
10	59259	31
11	59701	22
12	60150	28
13	60606	22
14	61068	16
15	61538	18
16	62015	19
17	62500	20
18	62992	14
19	63492	20
20	64000	22

Figure 6. The resonant response.

After the measurements the program searches for the maximum amplitude value stored in the array and its corresponding frequency. The resonant frequency is then transmitted again so that the waveform can be easily displayed on an oscilloscope. The LCD shows the measurement values. The software now decides if the key is recognized or not. When the resonant frequency lies between 57 kHz and 59 kHz and the signal level is above a threshold the lock will open and the red LED comes on. The measurement process will be repeated with the same result so long as the label remains in the coil.

This contactless entry system is convenient to use but will only be secure if its operating principle remains secret. If the level of security needs to be much higher, the worry is that there are so many potential keys in circulation that could be used to open the lock. You can however remedy this and make it more secure; trim the strip so that it's a bit shorter and this will change (increase) its resonant frequency. Using this trick you can make several 'keys' with different length strips and then build a multi-function lock that responds in different ways, depending which key is presented.

Web Links

[1] www.elektor-magazine.com/140009
[2] www.elektor-magazine.com/090615

Index

Analogue Electronics and Microcontrollers Projects

Basic Electronics for Beginners

Burkhard Kainka,
Born in 1953, a Radio amateur with the callsign DK7JD. He has spent many years active as a physics teacher, and from 1996 became a self-employed developer and author of electronics and microcontroller books.
He operates the websites **www.elektroniklabor.de** and **www.b-kainka.de** among other projects.

Hobbyist electronics can be a fun way to learn new skills that can be helpful to your career. Those who understand the basics of electronics can design their own circuits and projects. However, before you run, you need to learn to walk.

It all starts with analogue electronics. You should be familiar with the simple components and circuits and understand their basic behaviors and the issues you may encounter. The best way to do this is through real experiments. Theory alone is not enough. This book offers a large number of practical entry-level circuits, with which everyone can gain the basic experience.

Through the widespread introduction of microcontrollers, a new chapter in electronics has begun. Microcontrollers are now performing more and more tasks that were originally solved using discrete components and conventional ICs. Starting out has become easier and easier thanks to platforms including Bascom, Arduino, micro:bit. The book introduces numerous manageable microcontroller applications. It's now a case of less soldering and more programming.

ISBN 978-1-907920-84-4

9 781907 920844

Elektor International Media

www.elektor.com