Analog Random Numbers

Using an Arduino and a noise source

By **Kurt Diedrich** (Germany)

It's easy enough these days to generate random numbers digitally using a computer program. But there is also an analog approach: the project described here processes the noise produced by a transistor junction using an Arduino. The random numbers so generated can be used in various interesting experiments and will even lead us a little into the world of mysticism.

It is not only mystics who wonder whether there exists some universal consciousness that connects us all together and lets us communicate subconsciously with one another. Thanks to a project run from Princeton University using a collection of random number generators spread all over the world, we may now have some signs that there may be a connection between human consciousness and the behavior of electrons in a noisy pn-junction (see text box). Since I had just finished soldering together a simple random noise source for my home-built analog synthesizer, I hit upon the idea of trying to reproduce the Princeton experiment with my own hardware and software.
Random numbers produced by a computer are just the results of mathematical calculations, and so are in fact deterministic, or 'pseudo-random'. Such numbers are not suitable for this project: instead we must use an analog method to generate the numbers. One approach is to process the noise created by the electrons in the pn-junction of a reverse-biased diode. If this noise voltage is amplified and fed into an analog-to-digital converter, we will obtain a steady stream of unpredictable random numbers.

The project

The hardware I designed consists of a simple noise source whose output signal is fed into an Arduino microcontroller. The Arduino passes the random numbers derived from the analog source to a PC over USB, where they can be displayed graphically. The PC software has no pretensions to completeness and interested readers are encouraged to extend it. Both the source code for the PC program and the Arduino sketch are available as a free download from the Elektor Magazine website [1].

Hardware

The circuit of the noise source is shown in **Figure 1**. The noise originates in the reverse-biased base-emitter junction of transistor T1. This extremely weak signal has its DC component removed by C1 and is then amplified by IC1. The noise signal, now symmetric about zero, is taken via capacitor C2 to summing amplifier IC2, where a voltage set by P2 is added to ensure that the output is always positive. This is required because the analog inputs of the Arduino can only work with positive voltages. Preset P1 can be used to adjust the amplitude of the noise signal. You can use an oscilloscope to set the average level of the signal to about +1.4 V and the amplitude to about 1 V_{PP} (an accurate measurement of these parameters of the noise signal is not possible). The output of the noise source can now be connected to the input of the Arduino.
Figure 2 shows how the noise signal is fed into the Arduino:

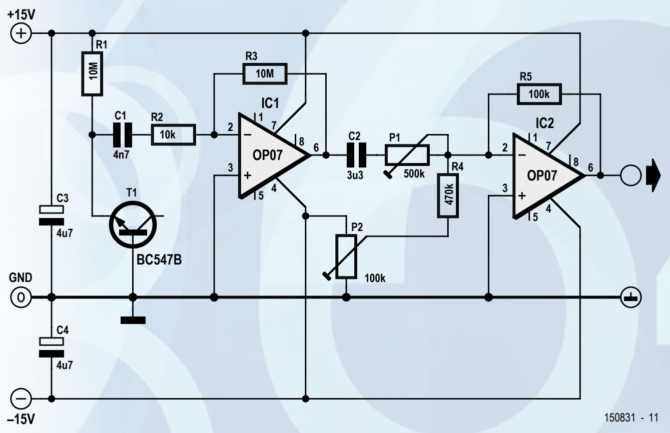


Figure 1. The circuit of the noise source uses a transistor and two operational amplifiers.

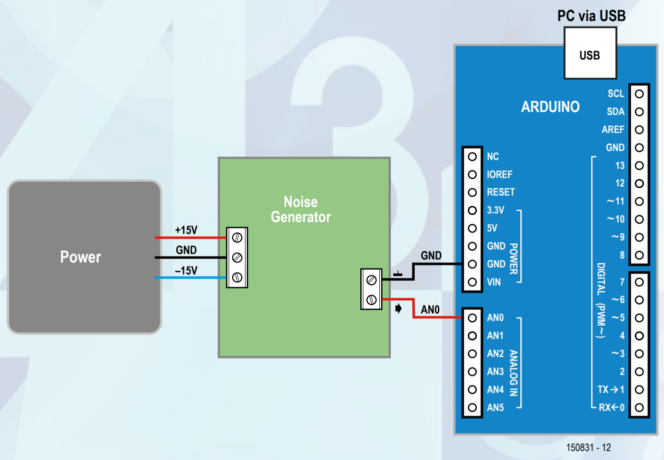
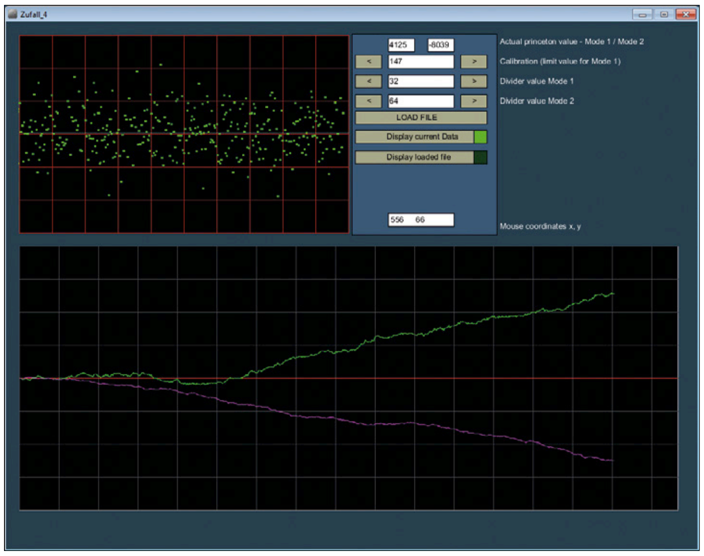


Figure 2. Connections between the power supply, the Arduino and the noise source.



4125 -8039 Actual princeton value - Mode 1 / Mode 2
147 Calibration (limit value for Mode 1)
32 Divider value Mode 1
64 Divider value Mode 2
556 66 Mouse coordinates x, y

Figure 3. User interface presented by the program, which automatically starts collecting data when it is launched.

an Arduino Uno is used in this case. There are two connections: first, the signal connection, which is taken to analog input A0 of the Arduino; and second, the ground connection. The noise source can be powered from any suitable external 2 x 15 VDC power supply. The Arduino is powered from the PC over the USB cable.

Software

As already mentioned, two separate pieces of software are needed in this project: one running on the Arduino and one on the PC [1]. The program running on the PC is written in Processing. The Processing programming environment (which so far I have only used under Windows XP and Windows 7) can be downloaded from the Internet free of charge [2]. A Processing file must always be placed in a directory that has the same name as the program file itself.

It is very important to ensure that you enter the correct COM port number in the setup part of the Processing code so that the Arduino is recognized by Windows. You can find the correct number using the Device Manager. For example:

```
serport = new Serial(this, “COM3”, 115200);
```

Unfortunately it can happen that the port number changes by itself while you are experimenting with the system.
After using the Arduino IDE to flash the sketch into the Arduino, it can be connected to the PC and then the Processing program can be launched. After a brief delay the signal should appear on the screen in the upper graph window (see **Figure 3**, above), and it should be symmetric about the horizontal center line. If it is shifted or if the amplitude is not correct, P1 and P2 must be adjusted. Below we will describe how you can fine-tune these settings using the software.

Operation

Upper graph window

The points shown in the upper graph window represent the random sample values received from the noise source. Every second approximately thirty of these values are received from the Arduino. Filling the window takes about ten seconds, corresponding to about 300 samples per pass that are available for further processing.

Lower graph window

Two curves slowly build up in the lower window, one green and one purple. Each time the upper window is filled each of these curves is extended by one pixel. The curves are produced from the random sample values as follows.

Purple curve (mode 2)
The program checks whether the current sample value of the noise signal is even or odd, and respectively increments or decrements a variable. Then the value of this variable is plotted.

Green curve (mode 1)
If a sample is above the horizontal center line a variable is incremented; if it is below the center line, it is decremented. It is not easy to set the offset potentiometer in the hardware sufficiently accurately, and in any case the offset will vary slightly with temperature, and so a calibration process is required. The calibration buttons help with this.

Calibration buttons

These are the two buttons towards the top of the display, which can be used to make vertical adjustments to the horizontal line in the upper graph window that divides the positive samples from the negative ones. Depending on this setting, the green curve in the lower graph window will either run horizontally or will trend up or down. The offset potentiometer in the noise source should be adjusted so that the green curve runs as horizontally as possible when the threshold position is set to 150 using the buttons. In this case the random points displayed in the upper graph window should appear to be perfectly symmetric with respect to the center line. The threshold value of 150 appears among the variable declarations in the Processing code, as follows:

```
int limit = 150 ;
```

If a different value suits your hardware better, you can modify the code.

Divider buttons

The next two buttons down allow for the expansion and compression of the curves in the Y direction, to allow them to be analyzed in greater detail. The two numbers shown at the top of the display are the most recent values plotted on the curves.

Files and ‘LOAD FILE’

Normally every thirty minutes a file is created storing the data represented by the curves in the lower graph. These files are saved in the same directory as the Processing program. The file name is determined when the program is started and consists of the time and the date when the program was started. The time interval between files being saved can be adjusted by modifying the variable declarations in the code: details on

Spooky action at a distance

Since the end of the 1990s a research group at Princeton University has been conducting a study, called the Global Consciousness Project, or GCP, that collects and evaluates data generated by seventy random number generators located all around the world.
In theory the distributions of numbers from these generators should all be identical. For example, if the plot of noise sample values is shifted so that they are symmetrical about the zero line then occurrences of positive and negative values will balance when analyzed over a sufficiently long time period. The consequence of this is that the plot obtained by accumulating a fixed number of these values should wobble about the zero line; if the shift is not exactly correct then the plot will drift at an approximately constant rate in one direction or the other. In rare cases the researchers at Princeton observed significant disturbances to this delicate balance when an event of global significance occurred that had an emotional effect on a large fraction of human beings: the events of 11 September 2001, the death of Princess Diana, or the Madrid attacks. In all these cases the readings collected showed deviations from their expected properties.

Further information on the background of this research, as well as criticism of it, can be found at the following sites:
http://global-mind.org
http://global-mind.org/results.html#alldata
http://global-mind.org/control.distribution.html
http://noosphere.princeton.edu/story.html
http://subroutine.jimdo.com (author's website, partly in English, mostly in German)

how to do this are given in the program.
Clicking on 'LOAD FILE' brings up a file selection dialog where you can navigate to any previously-saved file and reload it.

Display current data / Display loaded file
The data plotted in the lower graph can be changed by clicking on the two buttons 'Display current data' or 'Display loaded file'. When the plot is changed to 'Display current data' it is not updated until the plot in the upper graph reaches its right-hand edge.

The figures below the buttons show the coordinates of the mouse pointer when the mouse button is pressed. This is a useful tool to have when adding new elements to the user interface design.

Interpretation
Of course the plotted curves are open to various interpretations. When you launch the program you will notice that, although there are many deviations of small magnitude, the overall trend of each curve is firmly in one direction. The tiny jagged deviations are a natural consequence of the random nature of the noise generator; they can also be caused by temperature changes. However, if the curve should suddenly show a clear change in direction from its previous average trend, perhaps something is afoot! Perhaps you might want to take a quick look at your favorite news website to see what is going on...

On the other hand, you might prefer to spend your time modifying the processing program to display and analyze other types of data more obviously influenced by the outside world, such as readings from a temperature sensor!

(150831)

Web Links

[1] www.elektormagazine.com/150831
[2] https://processing.org/download/

Listing 1. Arduino sketch (excerpt).

```
void setup()
{
    Serial.begin(115200);
}

void loop()   // Endlosschleife
{
    int sensorValue = analogRead(A0);
    Serial.println(sensorValue);
```

Notes on the Processing code

The code might at first sight seem rather complicated. This is because Processing does not have a built-in way to construct graphical user interfaces for programs. All buttons and text labels, and all the mouse events that connect together to form such an interface, must be written as 'normal' source code. However, the result works well and the payback for the complexity of the code is that graphics handling in Processing is astonishingly quick. I have added thorough comments to the source code wherever required. Readers interested in programming in Processing will find many interesting ideas in the listing, such as how buttons and text labels are programmed, how to display the contents of an array as a plot in a graphics window, store them to a file and read them back again, how to determine the time and how to create a file selection dialog.