Raspberry Pi Voltmeter

with color display

Just a few additional components are needed to turn a Raspberry Pi into a DC voltmeter capable of reading up to 5 V and displaying the results in color on a monitor. The entire display area can be used, which makes the project an ideal basis for demonstrations, for example in schools.

By **Hermann Nieder**
(Germany)

The program that accompanies this project is written for revision 2 of the Raspberry Pi in Python: a good three-part introduction to this language for people more used to BASIC has been published previously in *Elektor* [1]. There are two versions of the code. In both cases the Raspberry Pi communicates with an A/D converter chip using a small number of GPIO pins and displays readings on its screen.

The expansion circuit

The expansion circuit for the Raspberry Pi (**Figure 1**) consists of a Texas Instruments TLC549 analog-to-digital converter [2]. This device has a resolution of eight bits and is controlled over a serial interface. Although no spring chicken (the device has been around for more than thirty years!) it is easy to use, still readily available and, of course, comes in a DIL package. Here the input to the converter is fitted with a voltage divider comprising R1 to R3. For testing the author connected instead a 10 kΩ potentiometer and a digital multimeter so that the results could be compared.

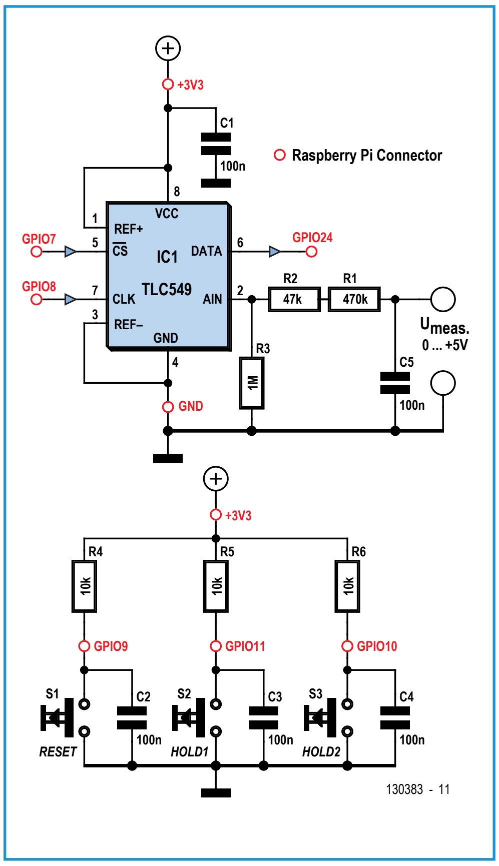


Figure 1.
Expansion circuit for the Raspberry Pi using an 'antique' A/D converter.

The two miniature pushbuttons 'HOLD1' and 'HOLD2' cause the displayed reading to be stored. The stored values are indicated by marks on the displayed scale as described below. Pressing the 'RES' button clears the stored values so that new readings can be recorded and displayed.

The circuit can easily be built on any of the prototyping boards widely available for the Raspberry Pi that fit directly to its expansion header, over which the Pi also provides power at 3.3 V. The voltage range can be adjusted by adding more resistors in series with R1 and then making suitable adjustments to the Python code. The comments in the code should help you find the places where changes are needed.

Program versions

Figure 2 shows a screenshot of one variant of the first version of the program. It shows three readings, the central one (which is larger) being the current value. The two pushbuttons can be used to store the current reading as a digital value. In additional to the digital readouts is a quasi-analog bargraph-style display. The stored readings are indicated on the bargraph by marker lines.

If the stored values will not be too close together then it is possible to use a slightly modified version of the program to show the markers are small arrowheads as shown in **Figure 3**.

The second version of the program (**Figure 4**) simulates a traditional analog voltmeter with a needle along with the digital display below. Again two values can be stored by pressing the 'HOLD1' and 'HOLD2' buttons on the expansion board, and these values are displayed as marker lines above the meter scale. The stored values and marker lines can be cleared as before. **Figure 5** shows a modification of this second version of the code to include a frame around the meter display.

At start-up both versions of the program allow a choice from a range of standard colors for the display background and foreground, and the selection of a scaling factor to determine the size of the display. A title for the display can also be entered.

Testing

The author's Python programs can be downloaded from the *Elektor* website [3], ideally by connecting the Raspberry Pi itself directly to the Internet.

Create a new directory under the /home/pi directory, copy the downloaded archive file into it and unpack the file.

Before proceeding further it is necessary to install the Pygame library, which is used in all the author's program examples. If you are using a Debian-based operating system, this is a matter of typing the following line into a terminal window:

```
sudo apt-get install python-game
```

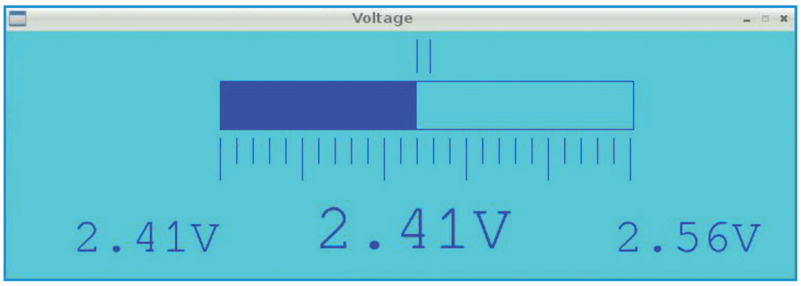


Figure 2. Digital voltmeter with quasi-analog display.

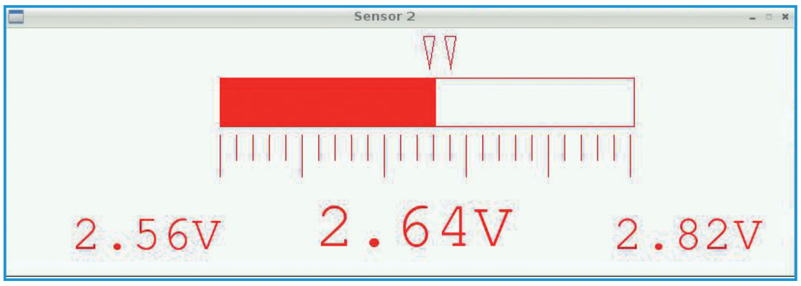


Figure 3. Color variation including two small arrowheads to indicate stored values.

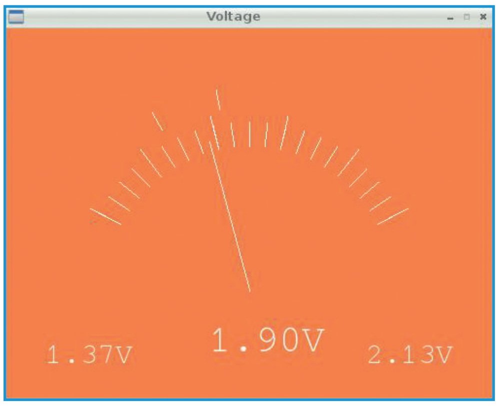


Figure 4.
Voltmeter with needle indication and digital display.

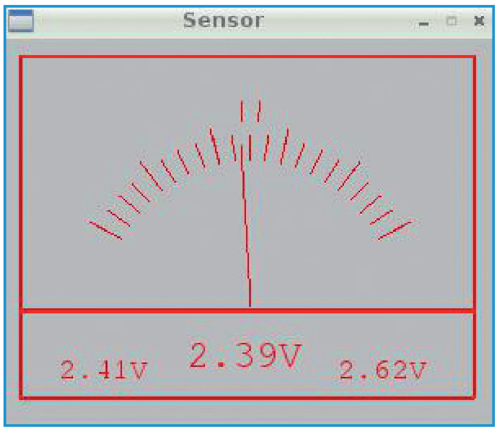


Figure 5.
The *de luxe* model includes a frame around the meter.

Now you are in a position to try out the author's programs.
In the terminal window switch to the directory where the Python files are stored and run the program with:

```
sudo python ADW_PTR_E.py
```

(Use sudo python ADW_PTR_D.py to try the German-language version of the code.) Now, from the terminal window, select one of the available background colors and press ENTER. Then select a foreground color, a title for the display and a scale factor to specify the desired size of display. The display window will then immediately open.

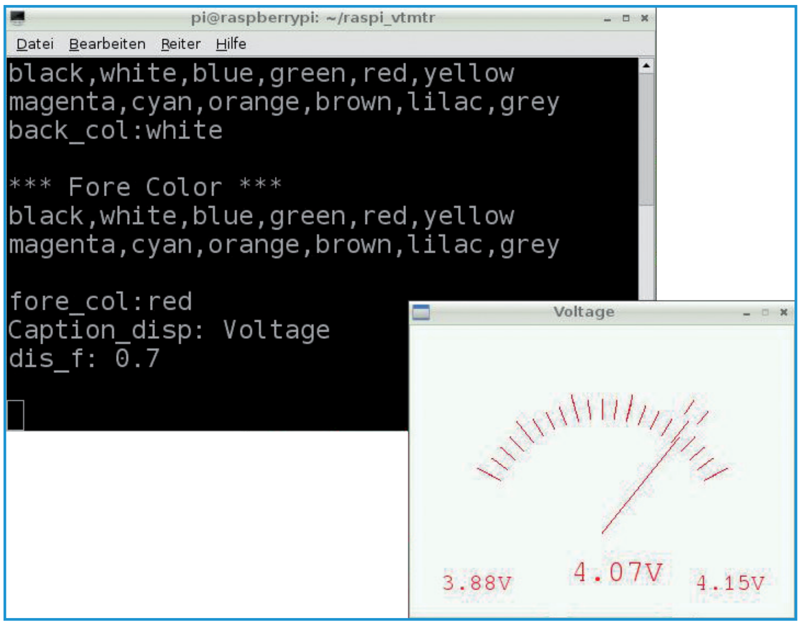
If the size of the window implied by the scale factor is too small or too large the program can be stopped by pressing control-C in the terminal window and then restarted. This will give you the opportunity to choose a new size. The other versions of the program are operated in a similar fashion.

In the example in **Figure 6** the background color chosen was 'white', while the foreground color for the needle, the scale and the marker lines is 'red'. The scale factor selected was 0.7. **Figure 7** shows the entries made in the terminal window to display readings from the TLC549 in blue on a yellow background.

Python listing

The TLC549 A/D converter is controlled using GPIO pins 7, 8 and 24 of the Raspberry Pi. To make the code easier to read, GPIO 7 is defined in the software as AD_CS and GPIO 8 as AD_Clk. The data sheet for the A/D converter [2] shows how the various signals have to be controlled to read a sample from the device. The eight bits of the result are shifted out of its DATA pin one after another, whereupon they can be reassembled for further processing. The DATA pin is connected to GPIO 24 on the Raspberry Pi, defined in the software as AD_Dat.

In the first version of the program, when the bargraph and digital reading is to be updated the previously-displayed colored bars and the previous reading must first be erased. This is done by plotting an 'invisible' (that is, background color) filled rectangle. Then

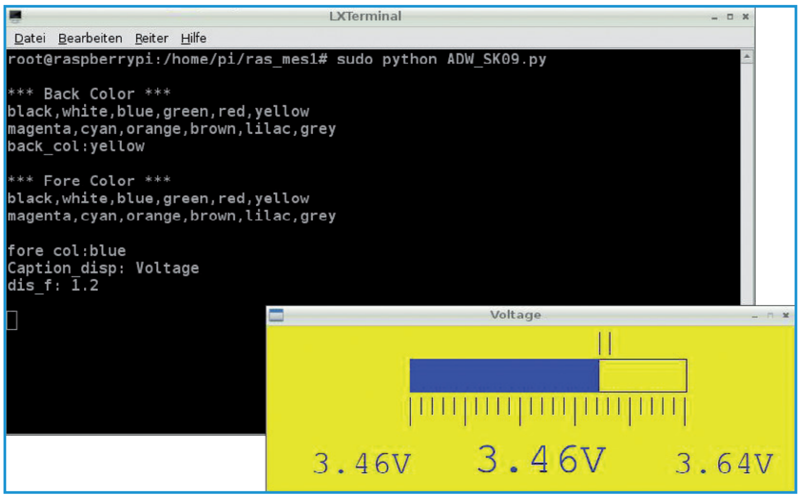


```
black,white,blue,green,red,yellow
magenta,cyan,orange,brown,lilac,grey
back_col:white

*** Fore Color ***
black,white,blue,green,red,yellow
magenta,cyan,orange,brown,lilac,grey

fore_col:red
Caption_disp: Voltage
dis_f: 0.7
```

Figure 6. Meter simulation and the corresponding set-up parameters, entered using LXTerminal on the Raspberry Pi.



```
root@raspberrypi:/home/pi/ras_mes1# sudo python ADW_SK09.py

*** Back Color ***
black,white,blue,green,red,yellow
magenta,cyan,orange,brown,lilac,grey
back_col:yellow

*** Fore Color ***
black,white,blue,green,red,yellow
magenta,cyan,orange,brown,lilac,grey

fore_col:blue
Caption_disp: Voltage
dis_f: 1.2
```

Figure 7. Different set-up parameters result in the simulation shown.

the foreground color is selected and a bar corresponding to the new value, and the new digital value itself, are drawn.

The second version of the program works in a similar way but the details are more intricate than the first version. The coordinates of the start- and end-points of the lines for the meter scale are calculated using sine and cosine functions. The deflection angle of the simulated voltmeter's needle corresponds to the current reading, and again the sine and cosine functions are used to calculate the required coordinates so that, as in a real instrument, the needle is aligned with the right point on the scale. The marker lines are plotted in a similar way, copying the instantaneous orientation of the needle when a pushbutton is pressed.

Before the needle is plotted its previous position must be erased. As in the case of the first version of the program, this is done by selecting the background color and then plotting its entire area of motion: in this case, a sector of a circle. Further details of how the graphics are drawn can be found by looking at the comments in the program listings.

(130383)

Internet Links

[1] From BASIC to Python, *Elektor*, May, June and September 2013; http://www.elektor.com/110483

[2] http://www.ti.com/lit/ds/symlink/tlc549.pdf

[3] http://www.elektor.com/130383

[4] http://lxde.org

The following fragment of code (common to both versions) is responsible for driving the TLC549.

```
def ADin():
    GPIO.output(AD_Clk,GPIO.LOW)# set AD_Clk to 0
    AD_res=0
    for n in range(10):
        time.sleep(0.05)
        GPIO.output(AD_CS,GPIO.HIGH)# set AD_CS to 1
        MSB=128
        time.sleep(0.001)
        GPIO.output(AD_CS,GPIO.LOW)# set AD_CS to 0
        time.sleep(0.0005)
        AD_value=0
        for z in range(8):
            if (GPIO.input(AD_Dat)):
                AD_value=AD_value+MSB
            GPIO.output(AD_Clk,GPIO.HIGH)# set AD_Clk to 1
            time.sleep(0.0005)
            GPIO.output(AD_Clk,GPIO.LOW)# set AD_Clk to 0
            MSB=MSB>>1
            time.sleep(0.0005)
            result=AD_value
        GPIO.output(AD_CS,GPIO.HIGH)# set AD_CS to 1
        AD_res=AD_res+AD_value
    AD_res=AD_res/10
    result=AD_res
    return result
```

For digital and graphical display the following conversion routine needs to be called.

```
def conversion(value0):
    value=value0
    value=value*1960 # for 5V
    value=value/1000
    pic_value=value/2 # for bargraph display
    one=value/100
    rest_z=value % 100
    tenth=rest_z/10
    hundredth=rest_z%10
    return one, tenth, hundredth, pic_value,value
...
```