

Calibration of the PVR has two components. One component is to calibrate the voltage reference chip. The other component is to set the offset for each PVR value. Calibrating the reference has to be done first.

To set the reference measure the voltage at the VREF test point and adjust via the trim pot. The more accurately you can set VREF the better, so you need to use a good quality voltmeter, one with at least 100  $\mu\text{V}$  accuracy in the 4V range. 10  $\mu\text{V}$  accuracy and resolution would be even better.

There are a number of things that will temporarily and/or permanently change the value of the voltage reference besides the trim pot. These include temperature extremes, mechanical stress, power supply variation, load and time. You stressed the reference both in temperature and mechanically when you soldered it to the board. The first few hours of operation the reference will "age" by changing value more quickly for than the rest of its life. To allow these stresses and their effects to stabilize, as well as to get past this early aging it is best to "burn in" the reference before it is calibrated. Burn in is nothing more than turning the PVR on and letting it sit. A burn in of 250 hours (about two weeks) is recommended. Obviously this is more conveniently done from a mains power connection rather than from batteries, though batteries will certainly work. After burn in the reference should be reasonably stable and the trim pot should be adjusted to set VREF to as close to 4.09600 volts as you can get.

You may be tempted to continue to periodically adjust VREF to maintain accuracy. Calibration experts would argue that it is better to NOT adjust the trim pot but rather to check VREF periodically and plot drift over time. This allows you to see how stable (or NOT) the reference is behaving and perhaps even to predict how it will drift in the future. It is of course your instrument and you can do as you like.

The remainder of the calibration procedure is relatively simple, but tedious. Even with a perfect reference voltage, the DAC is not perfectly accurate. I will not go into detail describing the different kinds of inaccuracies. If you want you can read up on offsets, INL and DNL. DAC data sheets often have write-ups on each of these characteristics. Suffice it to say that when you program the DAC for 1.000 volts per the data sheet you won't necessarily get 1.000 volts. You certainly won't get 1.000000 volts except as a fluke (pun intended). For the PVR you can set any value from 1 (0.001 V) to 4095 (4.095 V). This is represented inside the PVR as a 12 bit value. The DAC however is 16 bits. The 12 bit value is used to specify the upper 12 bits for the DAC. The lower 4 bits would normally be set to zero. Setting the lower 4 bits to other than zero adds a fraction more voltage so that the DAC can be adjusted much more finely than the 1 mV steps the 12 bit value would indicate. In fact the extra 4 bits allows the DAC to be adjusted in 1/16 mV or 62.5  $\mu\text{V}$  steps.

For example, let's say that the PVR is set for 1.000 volts. At the output however the measured voltage is only 0.9996 volts. The extra bits in the DAC can be used to add a little to the voltage. In this case we would choose to add six 62.5  $\mu\text{V}$  steps. This would make the output 0.999975 which is closer to perfectly accurate. In the PVR software there is the capability of setting an offset that gets added to the DAC value. The software effectively says "if the PVR is set for 1.000 V then add another 6 to the 16 bit DAC value". The offset is maintained as a signed 8-bit number so we can subtract a few fractions if the DAC is too high or add a few fractions if the DAC is low.

The goal is to make the PVR accurate to within 100 uV of the set value. In our example above, the value was set to 1.000 V. Any value between 0.9999 V and 1.0001 V would meet the specified accuracy. The measured 0.9996 value does not meet the spec so we add the offset to compensate. You can calculate the offset value by taking the programmed value subtracting the measured value and then dividing that result by 0.0000625. In the example the calculation would be

$$\text{Offset} = ( 1.000 - 0.9996 ) / 0.0000625 = 6.4 \text{ (round to nearest whole number)} = 6$$

More generally:

$$\text{Offset} = \text{round} ( ( \text{programmed\_value} - \text{measured\_value} ) / 0.0000625 )$$

The program remembers that offset and anytime the PVR is set to 1.000 that particular offset is added in. There needs to be an offset value for each possible PVR setting. The PVR has 4095 potential settings so we need to create the offset for each one. The calibration procedure is as follows.

1. for each possible PVR setting, set the PVR
2. measure the actual output voltage
3. calculate the offset
4. remember the offset for that setting

As I said earlier, it's not too complicated but it is tedious. If you can automate the process it will make things easier. What I did was write a simple program to set the value of the PVR over its USB interface, then read the PVR output voltage from my DMM over GPIB (thru a USB adapter) and then calculated the offset. The program writes the two numbers, programmed value and offset, into a text file on one line separated by a semicolon and terminated by a carriage return. Then the program loops through all possible set values for the PVR. The result is a CSV text file of all 4095 setting and offset values. Here is an excerpt from the file.

```
...
0995;-0002
0996;-0002
0997;-0002
0998;-0003
0999;-0003
1000;-0002
1001;-0002
1002;-0002
1003;-0002
1004;-0002
1005;-0002
...
```

To set the PVR value over USB you need to connect the PVR to a computer with an FTDI adapter and communicate via a virtual serial port (115200-8-N-1). Send the PVR a string of the form "!XXXX<cr>".

Where "XXXX" is the four digit millivolt value (number in the range of 1 to 4095) with no decimal. "<cr>" is a carriage return (linefeed also works). The "!" prefix tells the PVR to set the value without an offset (raw DAC setting). If you want to set the value with the stored offset included use a string of the form "#XXXX<cr>". And if you are doing this by hand (ouch) the way to set a program value without an offset is to power the PVR up while holding down the rotary button.

Time to take a little detour. Notice that with the offset the PVR is much more accurate than the spec. In fact if you contemplate the math a bit you will agree that the DAC now has a worst case accuracy of plus or minus 31.25 uV. Not too shabby. That assumes of course that all measurements are perfect, noise free and our DMM is perfectly accurate. To test that theory, you can run the program again. All things being equal it should generate exactly the same table right? But if you look, you will see that not every offset value comes out the same. The good news is that many do and even the ones that are off are only off by a count. Since our accuracy is plus or minus 31.25 uV and our goal is 100 uV the device is within spec even if these measurements are off by one count.

Another little detour. We have a problem. There are 4095 bytes of offset data we need to store somewhere. Unfortunately the microcontroller only has 1024 bytes of non-volatile memory (EEPROM). There is room in program memory but that would mean recompiling the code every time a PVR is calibrated. Looking at the data we can find another solution. If you look through the table of offsets you will see that many offset values are repeated. We can therefore compress the data to fit into the EEPROM memory. The compression algorithm does not need to be very sophisticated. It's better if it's simple to program and fast to read the offset value we want.

The algorithm is to scan up from one and note the program-value where the offset value changes. We record that last value. In the example calibration table the first six offsets are -2, then the offset changes to -3. So the first entry in our compressed data table is 0006,-2. Note that each entry now takes up three bytes, two for the program value and one for the offset. After the first six the offset values stay at -3 all the way up to 1058 where the offset changes back to -2. So the second value in our compressed data table is 1058,-3. The value then stays at -2 until program value 1154, so 1154,-2. We keep scanning up the table in this way until the end. Since the offset of program value 4095 is zero, the last entry in the compressed data table would be 4095,0.

Now let's combine the two detours to get back on track. Take a look at the list of program values (0995 to 1005) above. Note that the values are all -2 except for 0998 and 0999 which are -3. There are many times in the table where the offset changes value for just a couple of entries then returns to the more common value. It could be that these offsets are completely accurate, but our first detour hints that the anomalous values might also be noise. The first detour also argues that even if the offset is perfectly accurate, one more step away would still be within our spec of plus or minus 100 uV. The result is that we can discard these infrequent anomalies and thereby smooth the overall table data. Smoothing the overall data reduces the number of entries in the compressed table which makes the lookup at runtime faster. The entire compressed data table for the example is given below.

0006;-2  
1058;-3  
1154;-2  
1376;-3  
1395;-2  
1396;0  
1533;-2  
1557;-3  
1574;-2  
1878;-3  
1885;-2  
1969;-3  
2414;-2  
2493;0  
2626;-1  
2704;0  
2892;-2  
2942;0  
3255;-1  
3294;0  
3989;-1  
4095;0

Store this table starting at EEPROM address zero. Each table entry is three bytes. The 12-bit program value is broken into a high byte with the top 4 bits and a low byte with the bottom 8 bits. Then the offset is stored as a signed 8-bit value in the third byte. For example the first entry, "0006;-2", is stored in EEPROM as the following.

addr 0000 = 00  
addr 0001 = 06  
addr 0002 = 254 ( = 0xFE = -2 )

The next entry is stored as:

addr 0003 = 04  
addr 0004 = 34 (4 \* 256 + 34 = 1058)  
addr 0005 = 253 ( = 0xFD = -3 )

EEPROM values can be written through the serial port. It is a two-step write. First set the EEPROM address by setting the reference value, "!XXXX<cr>", where XXXX is a four digit decimal number of the address you wish to write. Then send "WDDDD<cr>", where DDDD is a four digit representation of the value you wish to write. Note that although it is expressed as four digits the only valid write values are in the range of 0 ("0000") to 255 ("0255").

Once you have written the table to EEPROM the calibration procedure is complete. You can spot check calibration manually or rerun the calibration program changing "!XXXX<cr>" to "#XXXX<cr>" so that calibrated values are output.