# DAC0 to ADC Rescale and Offset

Alex Striff

July 21, 2019

## 1 Offset and gain calculations for DAC0 to ADC opamps

### 1.1 Voltages that determine required offset and gain

```
In [1]: V_DL   = 3.3 * 1/6; # V. Due DAC0 minimum voltage
        V_DH   = 3.3 * 5/6; # V. Due DAC0 maximum voltage
        V_AL   = -2.5;      # V. ADC input minimum voltage
        V_AH   = 2.5;       # V. ADC input maximum voltage
        offset = ((V_AH + V_AL) - (V_DH + V_DL)) / 2.; # V
        gain   = (V_AH - V_AL) / (V_DH - V_DL); # V/V
        println("Offset:\t", offset, " V\nGain:\t", gain, " V");
```

```
Offset:        -1.65 V
Gain:        2.2727272727272725 V
```

### 1.2 Choice of resistor tolerance range (the E-series to use)

```
In [2]: E3  = [1.0, 2.2, 4.7]; # Very common
        E24 = [1.0, 1.1, 1.2, 1.3, 1.5, 1.6, 1.8, 2.0, 2.2, 2.4, 2.7, 3.0, 3.3, 3.6, 3.9, 4.3, 4.7, 5.1, 5.6, 6.2, 6.8, 7.5, 8.2, 9.1]; #
```

```
In [3]: E_series = E24;
```

Evaluates $f$ on all possible combinations of resistors in a given E-series.

```
In [4]: combinations(E, f) = [[f(E[i], E[j]), E[i], E[j]] for i in 1:length(E) for j in 1:length(E)]; # [[f(x, y), x, y]]
```

Finds the nearest E-series value to the value given.

```
In [5]: function nearest(E, x)
            p = floor(log10(x));
            y = x / 10^p;
            best = first(sort(E, lt = (a, b) -> abs(y - a) < abs(y - b)));
            best * 10^p;
        end
```

```
Out[5]: nearest (generic function with 1 method)
```

### 1.3 Offset component selection calculations

Improved calculation of the output voltage of the `LM4041-ADJ` where $V_{\text{ref}}$ depends upon $V_O$ (`V_Onom` here).

```
In [6]: V_Onom = abs(offset); # V
        V_Y = 1.240; # V
        ΔV_O = V_Onom - V_Y; # V
        ΔV_ref_ΔV_O = -1.55e-3; # V/V. Absolute worst case: -3 mV/V
        V_ref = V_Y + ΔV_O * ΔV_ref_ΔV_O; # V
        R2_to_R1 = (V_Onom / V_ref) - 1.; # Ω/Ω
        VO(ratio) = V_ref * (ratio + 1.);
```

Selection parameters.

```
In [7]: ratio_tolerance = 1.; # %
        RV_off = 1.0e3; # Ω
        println("We want a resistor ratio of ", R2_to_R1, " Ω/Ω and ", ratio_tolerance, "% nominal tolerance.");
```

```
We want a resistor ratio of 0.33132746661696366 Ω/Ω and 1.0% nominal tolerance.
```

Calculate the best resistors to choose to give the best nominal ratio.

```
In [8]: offset_ratio(x, y, off) = (x + 0.5 * off) / (y + 0.5 * off);
        # Divide RV_off by 1e3 so that it is at the same scale as the E-series values.
        off_ratios     = combinations(E_series, (x, y) -> offset_ratio(x, y, RV_off / 1e3)); # [[Ω/Ω, Ω, Ω]]. [[ratio, R2, R1]]
        off_ratios_tol = filter(x -> ratio_tolerance / 100. > abs(1. - x[1] / R2_to_R1), off_ratios); # [[Ω/Ω, Ω, Ω]]
        off_ratios_s   = sort(off_ratios_tol, lt = (x, y) -> isless(abs(x[1] - R2_to_R1), abs(y[1] - R2_to_R1))); # [[Ω/Ω, Ω, Ω]]
        if (length(off_ratios_s) > 0)
            R1 = off_ratios_s[1][3] * 1e3; # Ω
            R2 = off_ratios_s[1][2] * 1e3; # Ω
```

```
            display(off_ratios_s);
            println("Best offset resistor values:\n",
                "R2: ", R2 / 1e3, " kΩ\n",
                "R1: ", R1 / 1e3, " kΩ");
        else
            println("No combinations give a ", ratio_tolerance, "% offset tolerance.");
        end
```

```
3-element Array{Array{Float64,1},1}:
 [0.333333, 1.1, 4.3]
 [0.333333, 2.4, 8.2]
 [0.333333, 2.7, 9.1]


Best offset resistor values:
R2: 1.1 kΩ
R1: 4.3 kΩ
```

Calculate the margins of adjustability when a potentiometer is inserted between the two resistors.

In [9]:
```
ratio_min = R2 / (R1 + RV_off);
ratio_max = (R2 + RV_off) / R1;
V_O_low   = VO(ratio_min) - V_Onom; # V. Negative margin away from V_Onom
V_O_high  = VO(ratio_max) - V_Onom; # V. Positive margin away from V_Onom
V_O_total = V_O_high - V_O_low;     # V. Total argin away from V_Onom
println("Offset margins **after amplification** for a ", RV_off / 1e3, " kΩ potentiometer:\n",
    "Low margin:\t",   V_O_low   * 1e3 * gain, " mV\t(", 100. * V_O_low   / V_Onom, "%)\n",
    "High margin:\t",  V_O_high  * 1e3 * gain, " mV\t(", 100. * V_O_high  / V_Onom, "%)\n",
    "Total margin:\t", V_O_total * 1e3 * gain, " mV\t(", 100. * V_O_total / V_Onom, "%)");
```

```
Offset margins **after amplification** for a 1.0 kΩ potentiometer:
Low margin:       -348.65660377358483 mV       (-9.297509433962265%)
High margin:       442.3534883720931 mV       (11.796093023255818%)
Total margin:      791.010092145678 mV       (21.093602457218083%)
```

### 1.3.1 `LM4041-ADJ` resistor selection

In [10]:
```
Vcc_ref = 5.; # V
V_R_ref = Vcc_ref - V_Onom; # V
```

```
R_load = R1 + RV_off + R2; # Ω
I_load = V_Onom / R_load; # A
I_Q = 2.9e-3; # A
I_t = I_Q + I_load;
R_ref = nearest(E_series, V_R_ref / I_t); # Ω
I_ref = V_R_ref / R_ref; # A
println("Best resistor for the LM4041-ADJ (I_load = ", I_load * 1e3, " mA):\n",
    "R_ref:\t", R_ref / 1e3, " kΩ\n",
    "I_ref:\t", I_ref * 1e3, " mA\t(", 100. * (I_ref - I_t) / I_t, "%)\n",
    "Power:\t", V_R_ref * I_ref * 1e3, " mW");
```

```
Best resistor for the LM4041-ADJ (I_load = 0.2578125 mA):
R_ref:       1.1 kΩ
I_ref:       3.045454545454546 mA        (-3.558094552651698%)
Power:       10.202272727272728 mW
```

## 1.4 Gain component selection calculations

Selection parameters.

```
In [11]: div_tolerance = 1.; # %
         RV_gain = 1.0e3 # Ω
         div_nom = 1. / gain;
         println("We want a resistor divider with a gain of ", div_nom, " V/V and ", div_tolerance, "% nominal tolerance.");
```

```
We want a resistor divider with a gain of 0.44000000000000006 V/V and 1.0% nominal tolerance.
```

Calculate the best resistors to choose to give the best nominal resistor divider.

```
In [12]: offset_divider(x, y, off) = (x + 0.5 * off) / (y + x + off);
         # Divide RV_gain by 1e3 so that it is at the same scale as the E-series values.
         gain_divs     = combinations(E_series, (x, y) -> offset_divider(x, y, RV_gain / 1e3)); # [[Ω/Ω, Ω, Ω]]. [[ratio, Rb, Ra]]
         gain_divs_tol = filter(x -> div_tolerance / 100. > abs(1. - x[1] / div_nom), gain_divs); # [[Ω/Ω, Ω, Ω]]
         gain_divs_s   = sort(gain_divs_tol, lt = (x, y) -> isless(abs(x[1] - div_nom), abs(y[1] - div_nom))); # [[Ω/Ω, Ω, Ω]]
         if (length(gain_divs_s) > 0)
             Ra = gain_divs_s[1][3] * 1e3; # Ω
             Rb = gain_divs_s[1][2] * 1e3; # Ω
             display(gain_divs_s);
```

```
            println("Best gain resistor values:\n",
                "Ra: ", Ra / 1e3, " kΩ\n",
                "Rb: ", Rb / 1e3, " kΩ");
        else
            println("No combinations give a ", div_tolerance, "% gain tolerance.");
        end
```

```
11-element Array{Array{Float64,1},1}:
 [0.44, 3.9, 5.1]
 [0.440367, 4.3, 5.6]
 [0.44086, 3.6, 4.7]
 [0.439024, 1.3, 1.8]
 [0.438596, 2.0, 2.7]
 [0.438356, 2.7, 3.6]
 [0.44186, 3.3, 4.3]
 [0.442308, 1.8, 2.4]
 [0.4375, 1.6, 2.2]
 [0.436975, 4.7, 6.2]
 [0.443038, 3.0, 3.9]


Best gain resistor values:
Ra: 5.1 kΩ
Rb: 3.9 kΩ
```

Calculate the margins of adjustability when a potentiometer is inserted between the two resistors.

```
In [13]: div_min   = Rb / (Ra + Rb + RV_gain);
         div_max   = (Rb + RV_gain) / (Ra + Rb + RV_gain);
         div_low   = div_min  - div_nom; # V. Negative margin away from V_0nom
         div_high  = div_max  - div_nom; # V. Positive margin away from V_0nom
         div_total = div_high - div_low; # V. Total argin away from V_0nom
         println("Gain margins for a ", RV_gain / 1e3, " kΩ potentiometer:\n",
             "Low margin:\t",   div_low   * 1e3, " mV\t(", 100. * div_low   / div_nom, "%)\n",
             "High margin:\t",  div_high  * 1e3, " mV\t(", 100. * div_high  / div_nom, "%)\n",
             "Total margin:\t", div_total * 1e3, " mV\t(", 100. * div_total / div_nom, "%)");
```

```
Gain margins for a 1.0 kΩ potentiometer:
Low margin:       -50.00000000000004 mV       (-11.363636363636372%)
```

```
High margin:        49.999999999999936 mV        (11.363636363636346%)
Total margin:        99.99999999999997 mV        (22.72727272727272%)
```