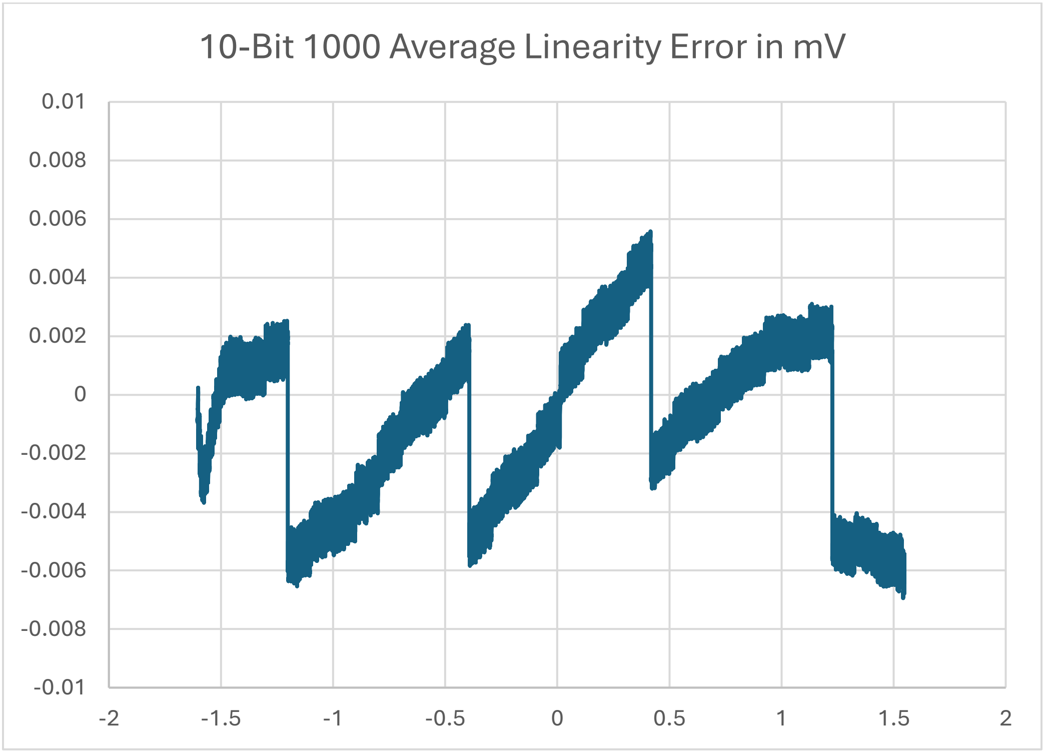
Breaking ADC Linearity Limitations

by Simple-Circuit 2024

The ADCs found in SOC (system on a chip) processors are convenient but often have limitations with respect to linearity and accuracy. Averaging can reduce noise and add resolution, but differential linearity ultimately limits achievable accuracy. We will show you how to improve linearity while adding resolution. One drawback is that the sampling frequency will suffer. However, this is not as important when measuring static parameters.

The method constructs a single-supply Delta-Sigma converter using a quad operational amplifier and a few resistors and capacitors. The input range is approximately ±1.55V referenced to a 1.65V level. In addition, it is possible to perform True RMS conversion on the input signal.



The above graph shows linearity for a Raspberry Pi Pico converter read with 10-bits of resolution. The data has been averaged 1000 times. For sections of the range, resolution is improved to about ±1mV. This about ±0.03% of full scale. However, differential non-linearity affects accuracy at several locations. Linearity over the full range is closer to ±0.16% of full scale. A true 10-bits of accuracy is ±0.049% of full scale.

What You Need:

Software:

- Arduino IDE 1.8
- Earle Philhower's Arduino-pico https://github.com/earlephilhower/arduino-pico
- GTKTerm (or any serial terminal with data logging)
- GNU Octave (for calculating linearity)

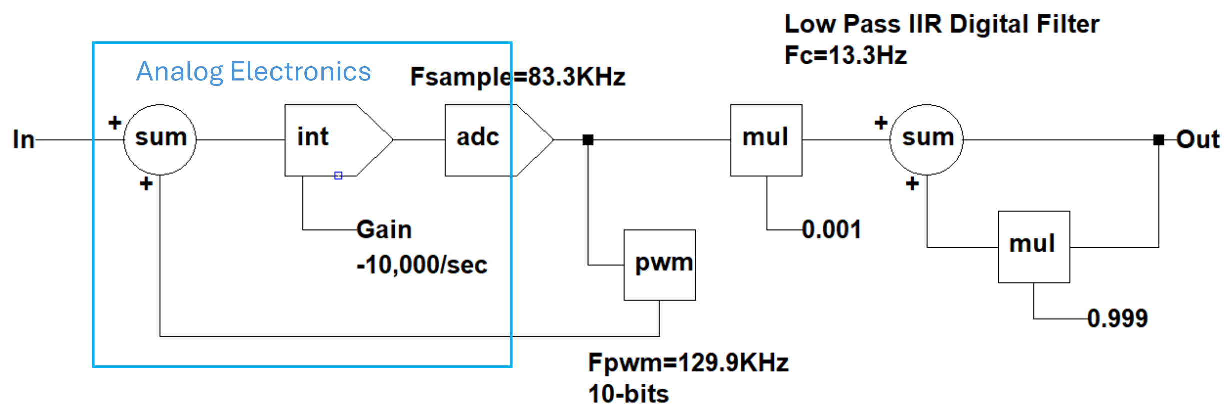
Hardware:

- Raspberry Pi Pico with pins
- USB cable
- Solderless breadboard
- Resistors Carbon film 5%: 15, 390, 30K, 4 – 100K, 10Meg
- Capacitors Polyester Film: 0.47uF, 0.22uF, 0.001uF
- Electrolytic Capacitor: 220uF
- Capacitor Polypropylene Film: 0.47uF
- Jumper Wires: about 11
- Quad Op Amp: MCP6004

Test Equipment:

- Floating voltage source: e.g. 1.5V battery
- Optional: Signal Generator: Output set to 1.65V ±1.65V max swing
    - (Use ground as negative input)
- Digital Multimeter

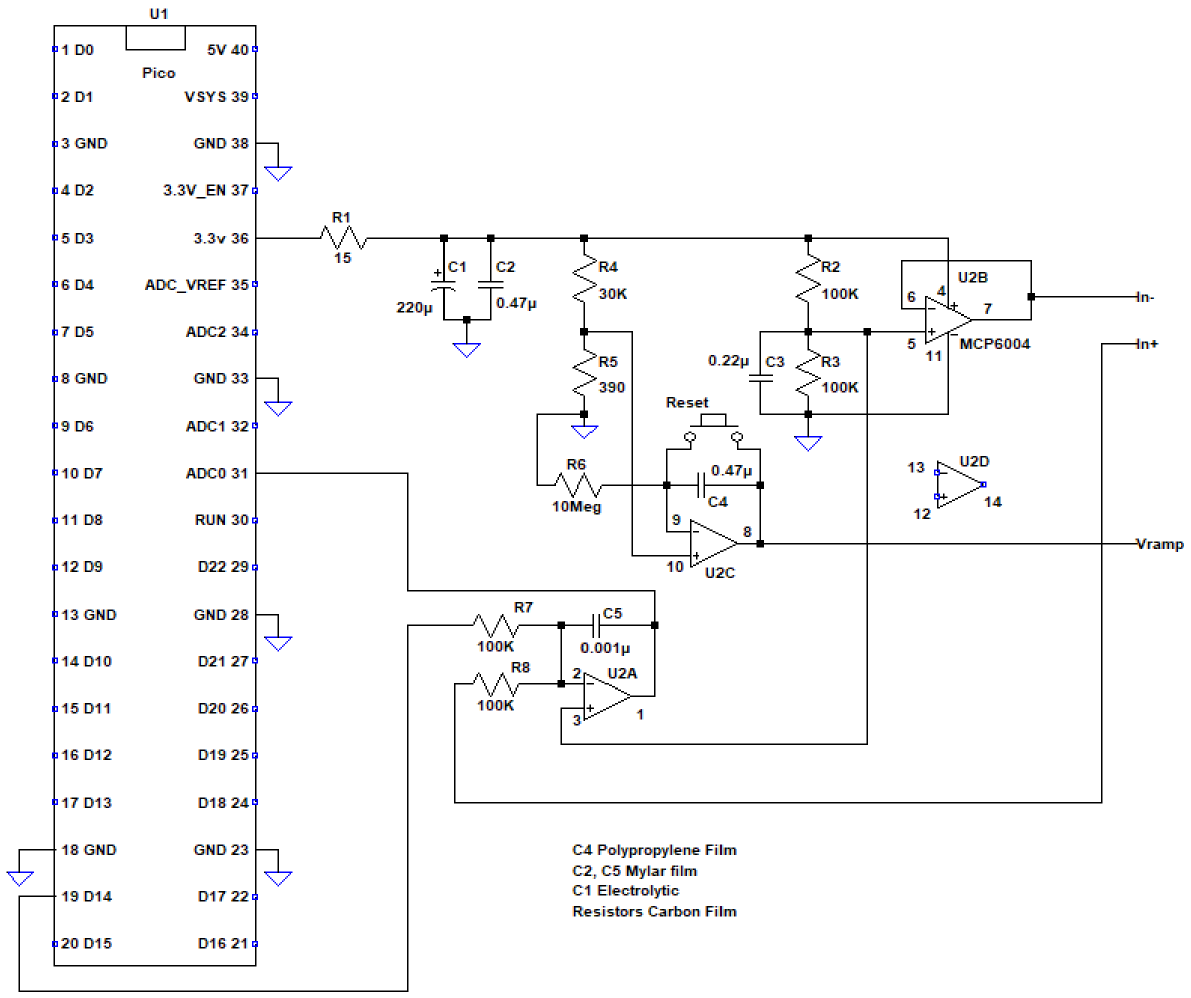
Delta-Sigma Converter Block Diagram:



The front end to the Delta-Sigma converter is an analog inverting summing integrator. The analog to digital converter samples the integrator output and sends the value to a pulse width modulated output bit. The PWM signal is used as negative feedback to form a voltage follower. The PWM value dithers around the input level. The ADC (PWM) value is the input to an Infinite Impulse Response Low Pass Filter. The averaged value has reduced noise and increased resolution.

The IIR output filter time constant is 12ms. For step changes, the output settles to 0.1% in about seven-time constants or 84ms.

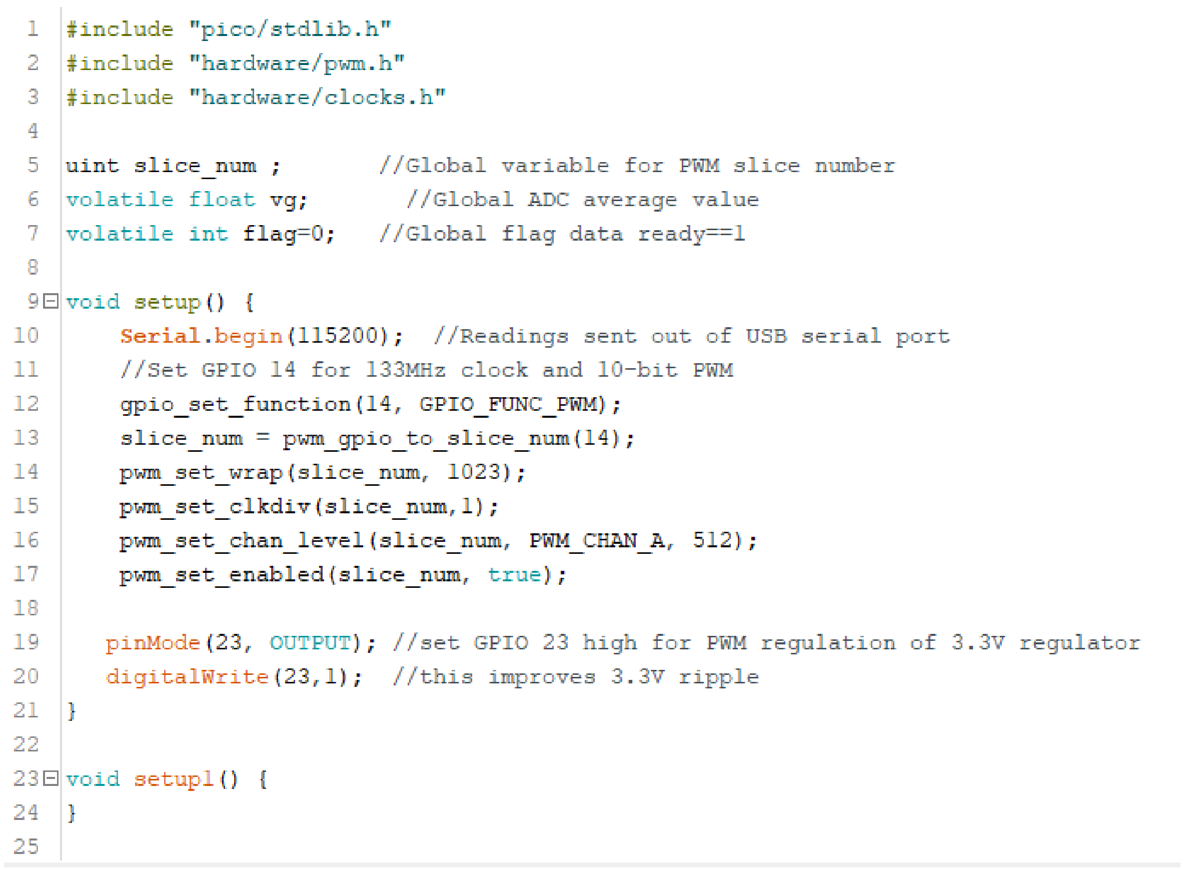
Schematic Diagram:



Operational amplifier U2A is the summing integrator. One input is from the D14 PWM output. The second input is positive voltage in. U2B is a voltage follower connected to the 3.3V midpoint 1.65V reference voltage. This output is used as the input voltage negative input.

A secondary integrator U2C is connected to a voltage divider set at 42mV. By pressing then releasing the reset button, a linear voltage ramp is generated. This ramp can be used to test linearity. Capacitor C4 must be a polypropylene film type. We will discuss linearity testing at a later time.

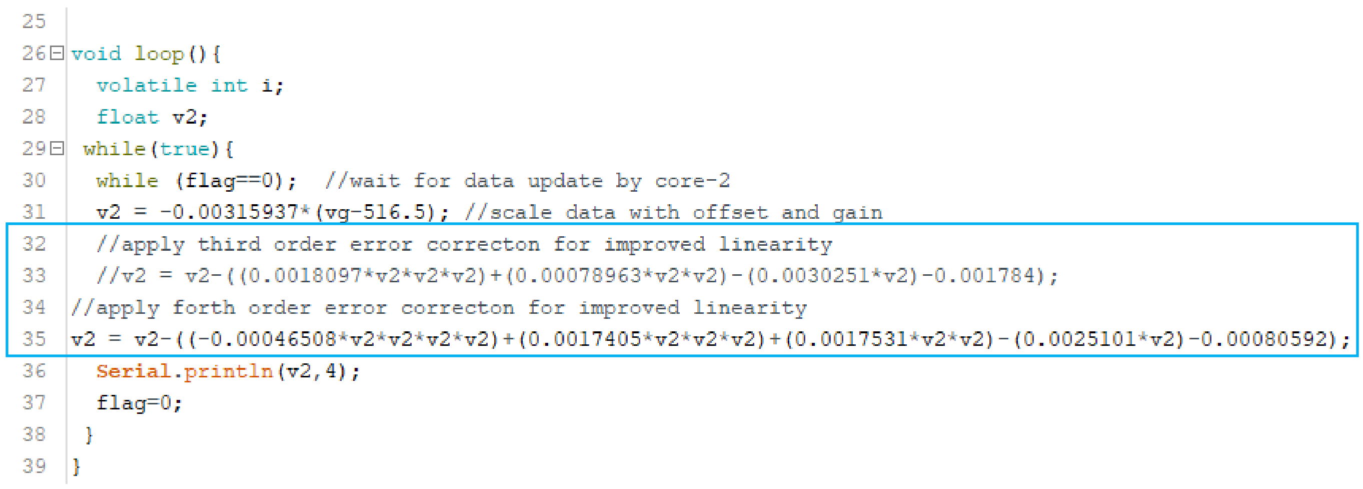
<u>Arduino IDE Program Code:</u>



```
#include "pico/stdlib.h"
#include "hardware/pwm.h"
#include "hardware/clocks.h"

uint slice_num ;       //Global variable for PWM slice number
volatile float vg;       //Global ADC average value
volatile int flag=0;   //Global flag data ready==1

void setup() {
    Serial.begin(115200);  //Readings sent out of USB serial port
    //Set GPIO 14 for 133MHz clock and 10-bit PWM
    gpio_set_function(14, GPIO_FUNC_PWM);
    slice_num = pwm_gpio_to_slice_num(14);
    pwm_set_wrap(slice_num, 1023);
    pwm_set_clkdiv(slice_num,1);
    pwm_set_chan_level(slice_num, PWM_CHAN_A, 512);
    pwm_set_enabled(slice_num, true);

   pinMode(23, OUTPUT); //set GPIO 23 high for PWM regulation of 3.3V regulator
   digitalWrite(23,1);  //this improves 3.3V ripple
}

void setup1() {
}

```

Core-1 is used for serial output while Core-2 performs continuous conversions.

GPIO-14 is set to output 10-bits of PWM signal at 133MHz. The PWM signal has a 3.3V amplitude. The PWM ripple out of the analog integrator is around 90mV. The IIR output filter should reduce this by 1000 to 0.09mV. However, it is generaly around three times this due to system noise.



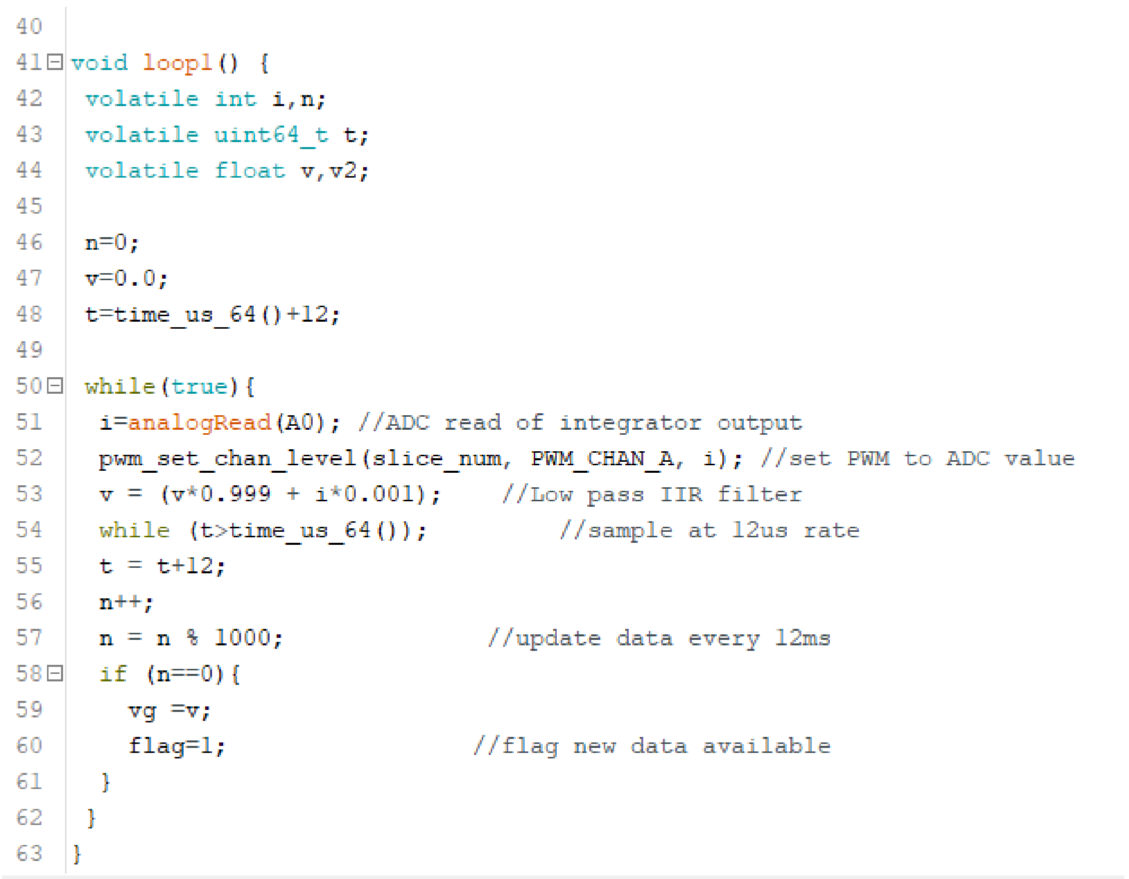
```
25
26 void loop(){
27   volatile int i;
28   float v2;
29  while(true){
30   while (flag==0);  //wait for data update by core-2
31   v2 = -0.00315937*(vg-516.5); //scale data with offset and gain
32   //apply third order error correcton for improved linearity
33   //v2 = v2-((0.0018097*v2*v2*v2)+(0.00078963*v2*v2)-(0.0030251*v2)-0.001784);
34 //apply forth order error correcton for improved linearity
35 v2 = v2-((-0.00046508*v2*v2*v2*v2)+(0.0017405*v2*v2*v2)+(0.0017531*v2*v2)-(0.0025101*v2)-0.00080592);
36   Serial.println(v2,4);
37   flag=0;
38  }
39 }
```

The Delta-Sigma conversion value is held in Global variable vg which is valid when flag is equal to one. The midpoint value is around 512. This offset is subtracted for the zero point. A negative scale factor is applied to adjust the range of around±1.55V. These values will vary with each build and calibration to a known source is required for accuracy.

After scaling, the value is sent to the USB serial port and flag is reset.

Notes:

- The scale factor and offset values for line 31 will vary with your resistor tolerances.
- Do not include line 33 or 35 for the initial test. These error correction equations are least square fits to the measured linearity error.



```
void loop1() {
 volatile int i,n;
 volatile uint64_t t;
 volatile float v,v2;

 n=0;
 v=0.0;
 t=time_us_64()+12;

 while(true){
  i=analogRead(A0); //ADC read of integrator output
  pwm_set_chan_level(slice_num, PWM_CHAN_A, i); //set PWM to ADC value
  v = (v*0.999 + i*0.001);    //Low pass IIR filter
  while (t>time_us_64());         //sample at 12us rate
  t = t+12;
  n++;
  n = n % 1000;              //update data every 12ms
  if (n==0){
    vg =v;
    flag=1;                //flag new data available
  }
 }
}
```

A 12us loop is set up to read the ADC, update the PWM value and filter the data. This is enough time to process the three operations. After 1000 loops, Global variable vg is assigned the filter output value. The variable flag is set to one indicating new data is available.

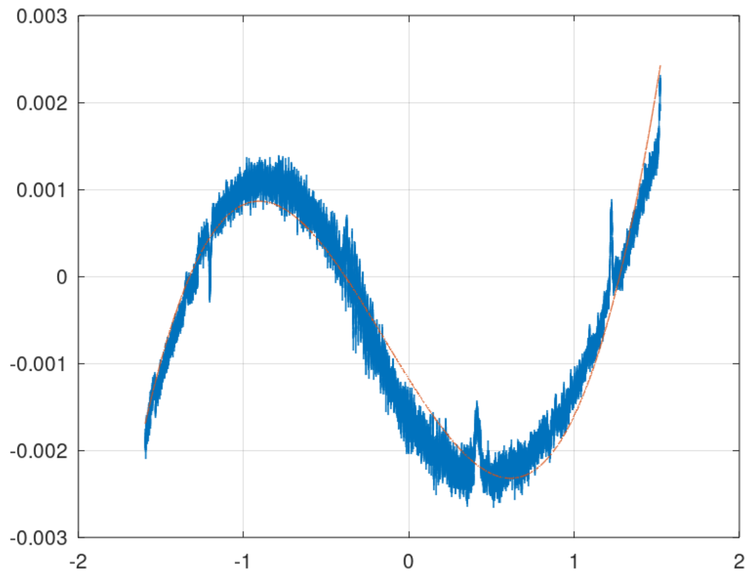
<u>Testing and Improving Linearity:</u>

To test linearity, use the following process:

- Connect the output Vramp to In+
- Use a serial terminal program that can log data (e.g. GTKTerm)
- Connect the terminal to the Pico
- Start data logging to the file “adcdata”
- Press, hold the reset button on U2C, then release it
- Data will drop to about -1.6V then slowly increase to about 1.6V
- When data is around 1.6V and not changing, stop data logging
- Open the “adcdata” file in a text editor and trim the ends of the data
    - Remove data below about -1.55V that is not increasing
    - Remove data above about 1.55V that is not increasing
    - Save the file
- Use the supplied GNU Octave script to calculate the linearity
    - The script will also generate a third order error polynominal for data corection

<u>GNU Octave Script Program 1:</u>

```
clear
a=csvread('adcdata');
m=length(a);
n=linspace(1,m,m);
x=a';
f=polyfit(n,x,1)
# f =f + [ 0.0005e-4, -0.0001 ] #use to adjust polyfit
y=polyval(f,n);
e=x-y;
p=polyfit(x,e,3)
yerr=polyval(p,x);
plot(x,e,x,yerr)
axis([-2,2,-0.003,0.003]);
grid on
```



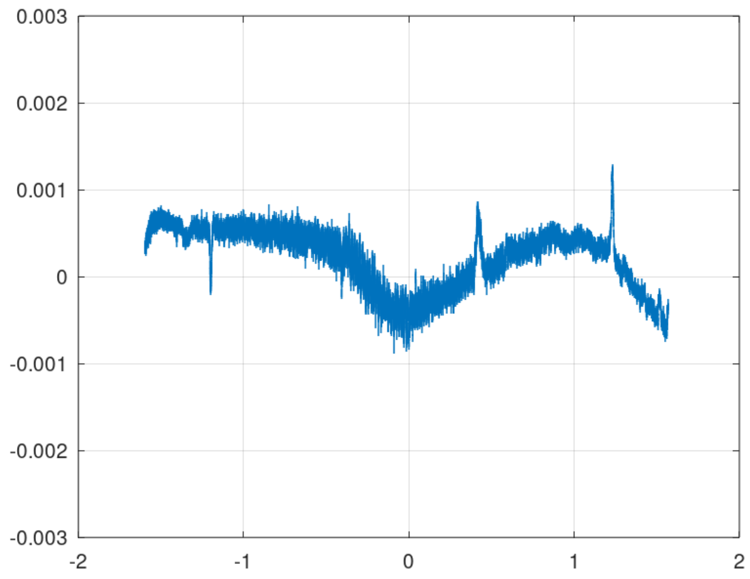
```
p =  1.8097e-03   7.8963e-04  -3.0251e-03  -1.1784e-03
```

The linearity plot shows the Delta-Sigma converter is linear to around ±2mV. This is about ±0.063% of full scale. The since the error shape is a curve, the third order fit can be used to correct some of the linearity.

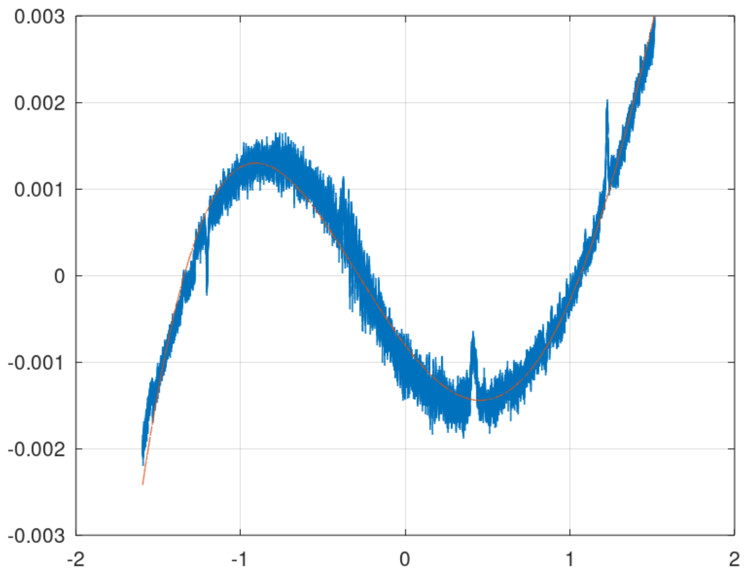
GNU Octave Script Program 2:

After adding the polynomial correction to line 33 in the Arduino code, retesting linearity can be performed with this simpler script.

```
clear
a=csvread('adcdata');
m=length(a);
n=linspace(1,m,m);
x=a';
f=polyfit(n,x,1)
y=polyval(f,n);
e=x-y;
plot(x,e)
axis([-2,2,-0.003,0.003]);
grid on
```

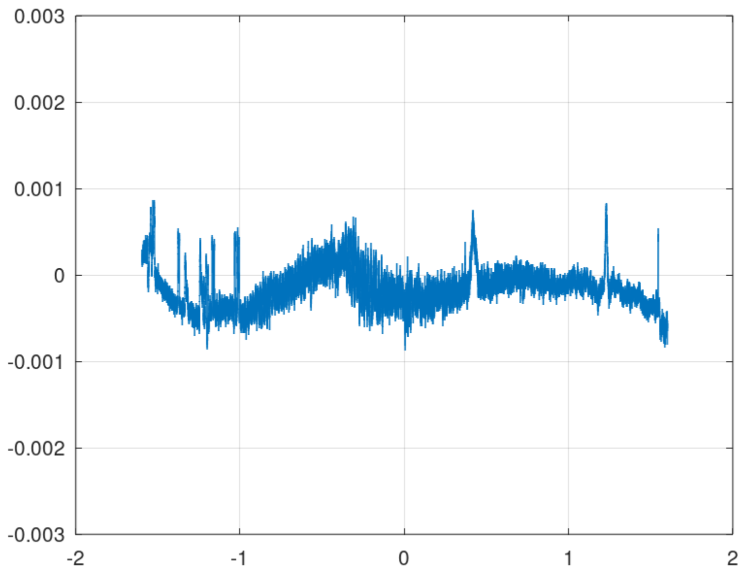


The linearity plot shows the Delta-Sigma converter with error corrections is linear to around ±1mV. This is about ±0.031% of full scale.



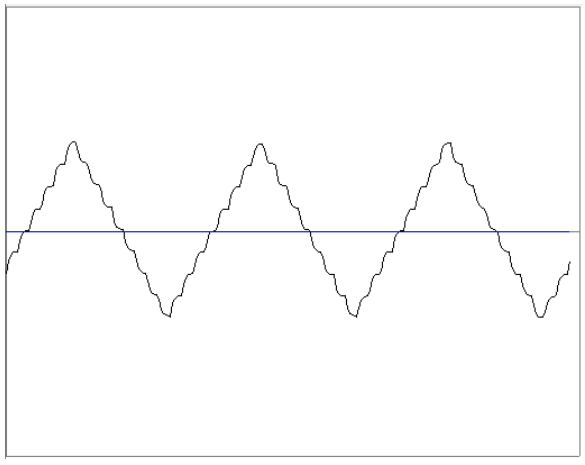
p = -4.6508e-04   1.7405e-03   1.7531e-03 -2.5101e-03 -8.0592e-04

A better error curve match is possible with a fourth order polynomial.

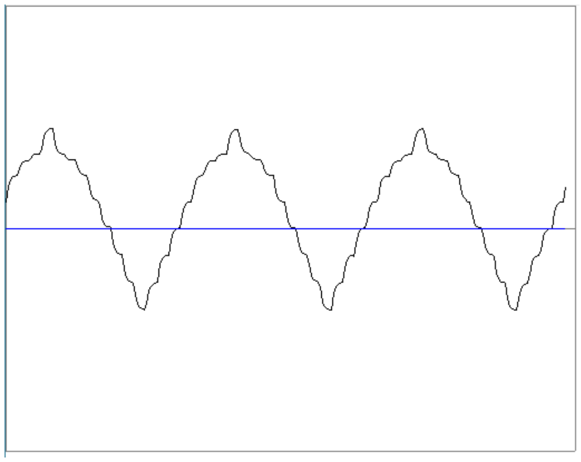


The linearity plot shows the Delta-Sigma converter with a fourth order polynomial correction is linear to around ±0.8mV. This is about ±0.025% of full scale.

Comparison of Delta-Sigma Sampling to Averaging:



Delta-Sigma Sampling



1000X average Sampling

A 40mVpp waveform was sampled using two techniques. The waveform generator has a DAC step of about 5mV. Both sampling techniques reduce noise and improve resolution. As the right plot shows, averaging does not improve linearity.

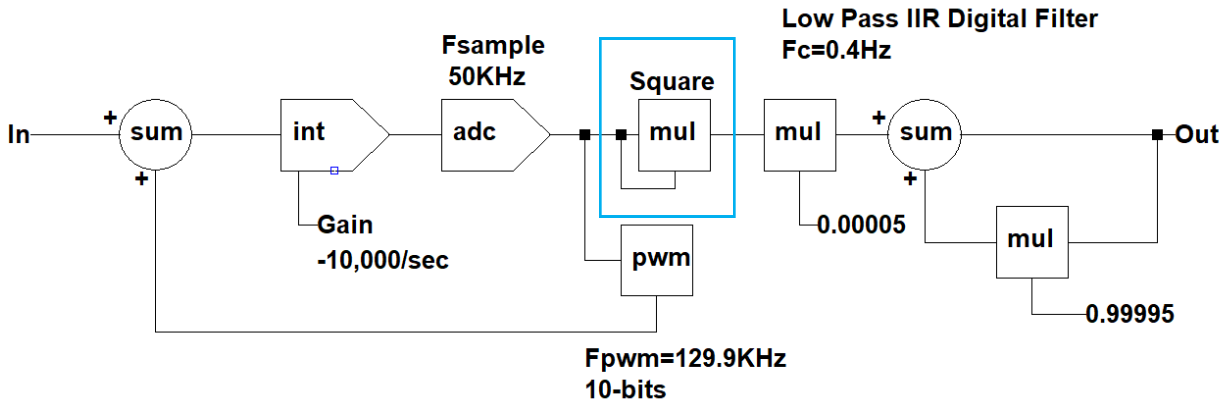
True RMS Measurement:

Root Mean Square or RMS measurement is a way to calculate the equivalent direct current heating value of an arbitrary voltage. You probably are familiar with some of the formulas for several waveform types:

Sine wave RMS = Vpeak * 0.707
Triangle wave RMS = Vpeak * 0.577
Square wave RMS = Vpeak * 1.0

The following formula will calculate the RMS voltage of any waveform. To perform this calculation, only a few extra program lines are needed for the Delta-Sigma converter.

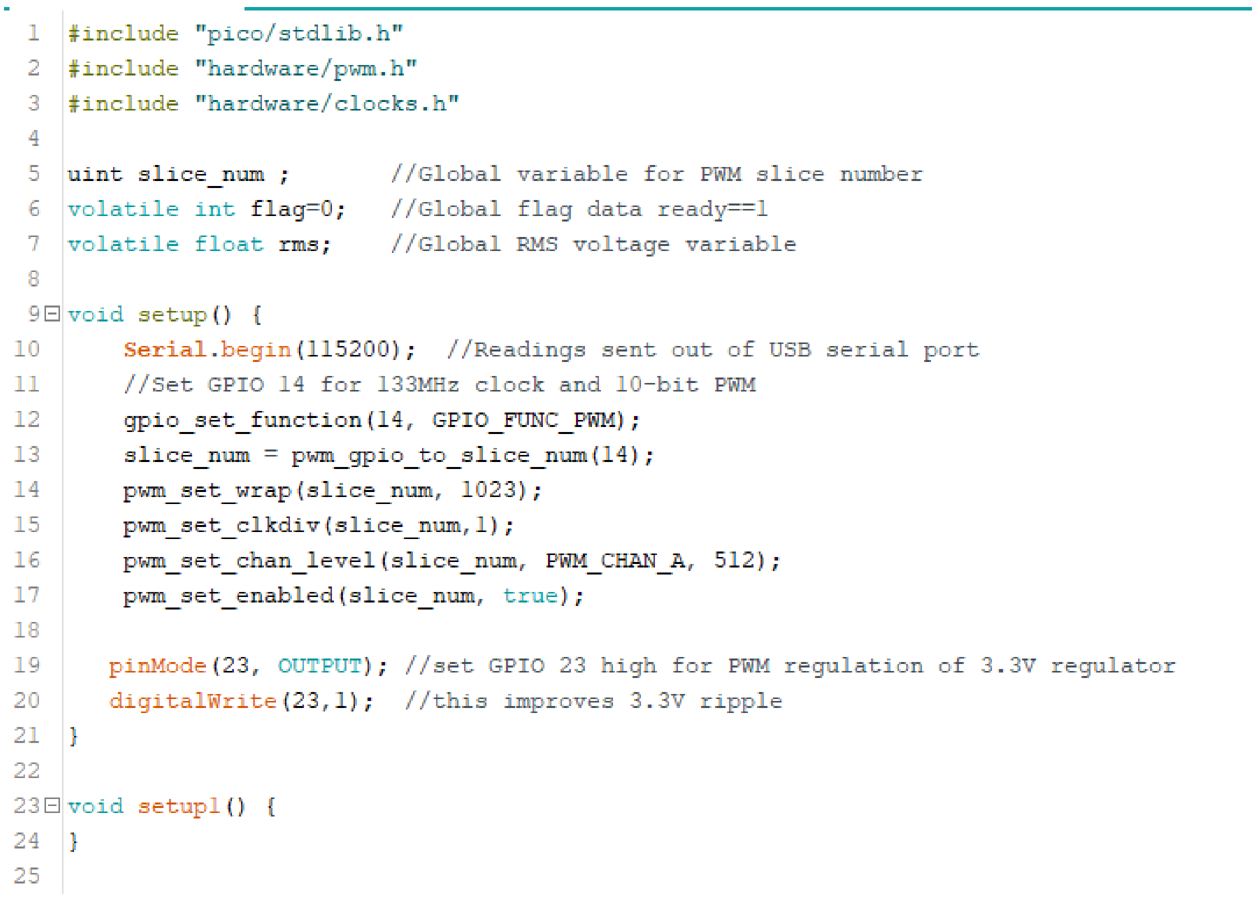
$$V_{RMS} = \sqrt{\int Vin^2 dt}$$



Squaring the ADC/PWM level allows root mean square (RMS) measurement. The IIR low pass filter acts as an integrator. To keep ripple less than 1%, the cutoff frequency is set two orders of magnitude lower than twice the frequency (squaring doubles the frequency). A 0.4Hz cutoff was selected for 30 to 120Hz measurements.

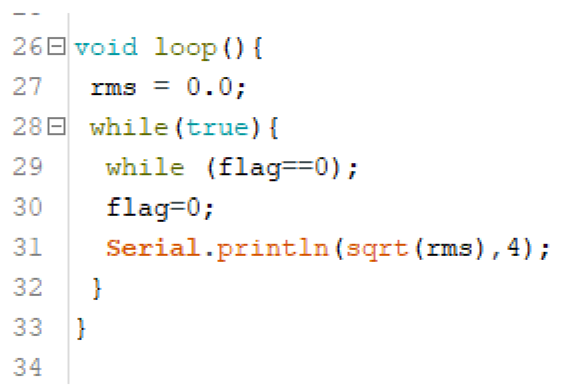
Taking the square root of the filtered output completes the RMS calculation.

Arduino IDE Program Code:



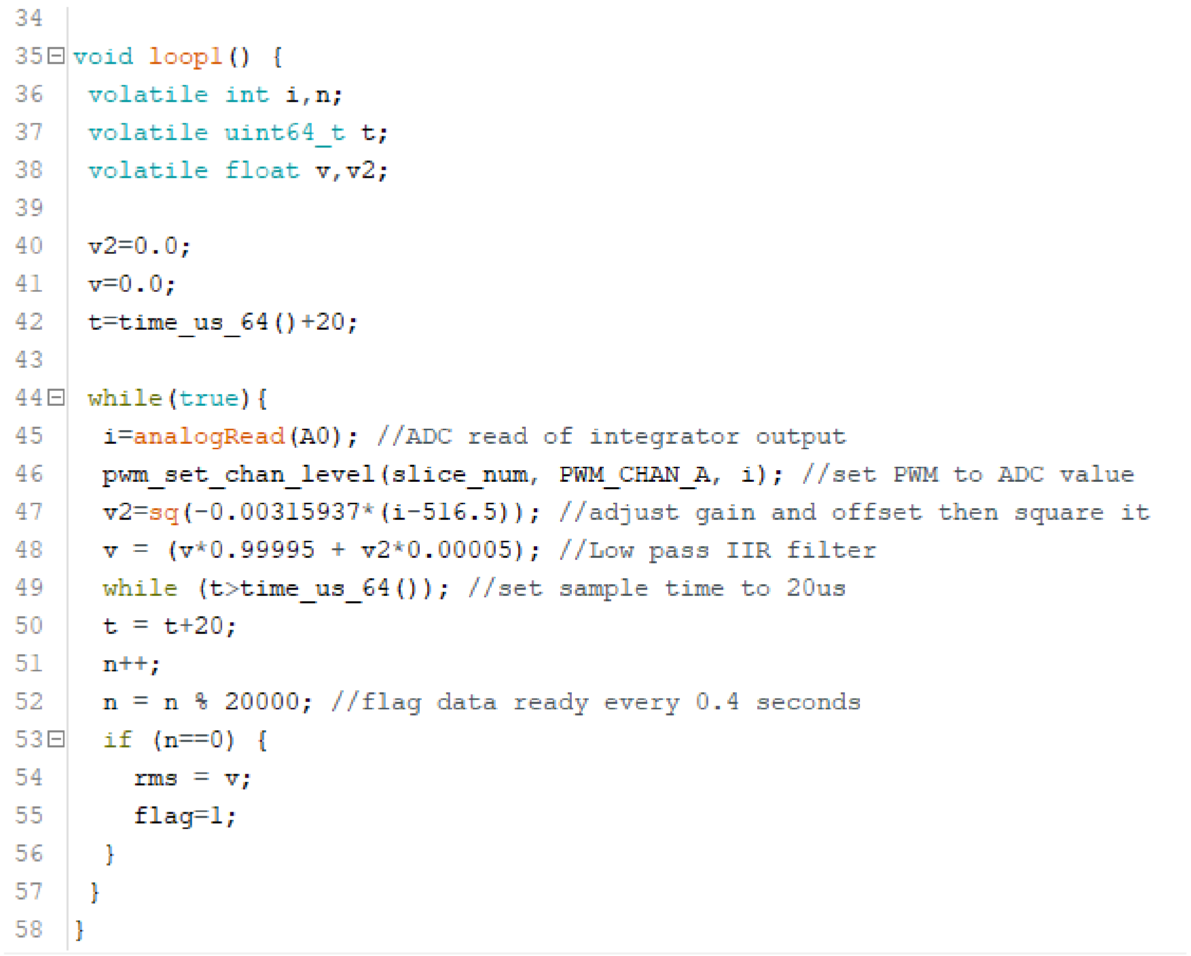
```
#include "pico/stdlib.h"
#include "hardware/pwm.h"
#include "hardware/clocks.h"

uint slice_num ;        //Global variable for PWM slice number
volatile int flag=0;    //Global flag data ready==1
volatile float rms;     //Global RMS voltage variable

void setup() {
    Serial.begin(115200);  //Readings sent out of USB serial port
    //Set GPIO 14 for 133MHz clock and 10-bit PWM
    gpio_set_function(14, GPIO_FUNC_PWM);
    slice_num = pwm_gpio_to_slice_num(14);
    pwm_set_wrap(slice_num, 1023);
    pwm_set_clkdiv(slice_num,1);
    pwm_set_chan_level(slice_num, PWM_CHAN_A, 512);
    pwm_set_enabled(slice_num, true);

   pinMode(23, OUTPUT); //set GPIO 23 high for PWM regulation of 3.3V regulator
   digitalWrite(23,1);  //this improves 3.3V ripple
}

void setup1() {
}

```

Setup is identical to the Delta-Sigma converter. The Global variable name was changed to rms.



```
void loop(){
 rms = 0.0;
 while(true){
  while (flag==0);
  flag=0;
  Serial.println(sqrt(rms),4);
 }
}

```

When data ready is flagged, Core-1 takes the square root and prints the value. Since the design is for around 1% calculation error, linearity correction is not needed.

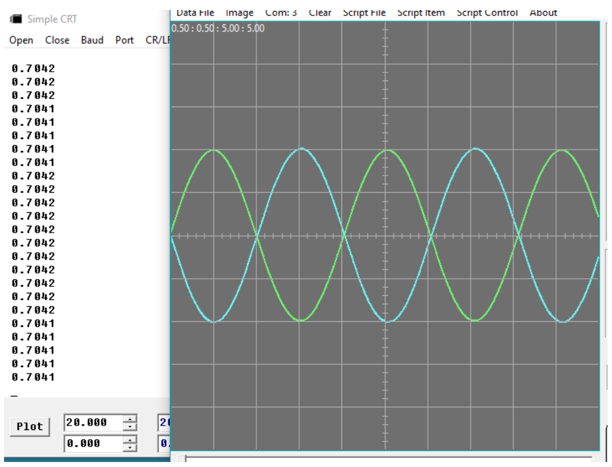


```

void loop1() {
 volatile int i,n;
 volatile uint64_t t;
 volatile float v,v2;

 v2=0.0;
 v=0.0;
 t=time_us_64()+20;

 while(true){
  i=analogRead(A0); //ADC read of integrator output
  pwm_set_chan_level(slice_num, PWM_CHAN_A, i); //set PWM to ADC value
  v2=sq(-0.00315937*(i-516.5)); //adjust gain and offset then square it
  v = (v*0.99995 + v2*0.00005); //Low pass IIR filter
  while (t>time_us_64()); //set sample time to 20us
  t = t+20;
  n++;
  n = n % 20000; //flag data ready every 0.4 seconds
  if (n==0) {
    rms = v;
    flag=1;
  }
 }
}
```

ADC sampling time was increased to 20us allowing for the added squaring operation. The IIR filter constants were changed for the cutoff of 0.4Hz. Also, data is updated at a slower 0.4 seconds rate.

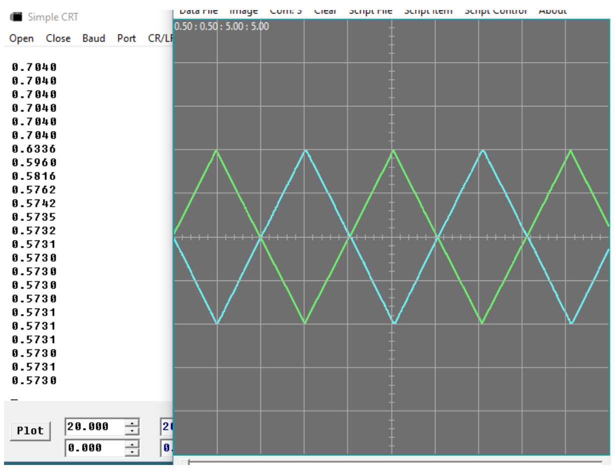
Testing of different 60Hz waveforms shows only the square wave does not meet the 1% error criteria. This was due to the rise time of the system. The full square wave data cannot be captured at the rising and falling edges.

The plots show the input waveform in green and the output of the analog integrator in blue.


Simple CRT
Open Close Baud Port CR/LF
Data File Image Com: 3 Clear Script File Script Item Script Control About
0.50 : 0.50 : 5.00 : 5.00
Plot
20.000
0.000

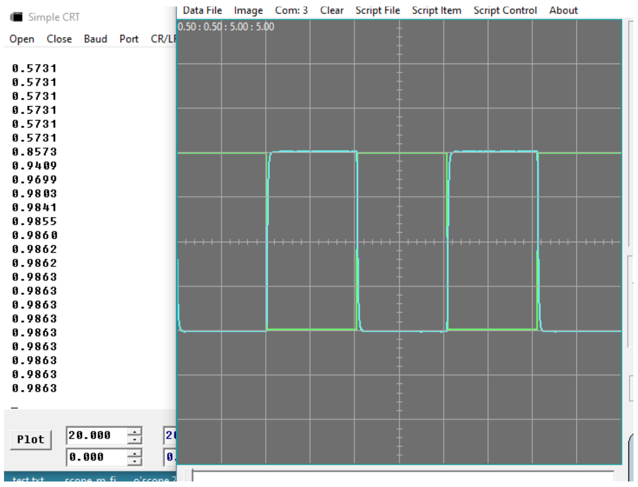
```
0.7042
0.7042
0.7042
0.7041
0.7041
0.7041
0.7041
0.7041
0.7042
0.7042
0.7042
0.7042
0.7042
0.7042
0.7042
0.7042
0.7042
0.7042
0.7042
0.7041
0.7041
0.7041
0.7041
0.7041
```

Error -0.14%


Simple CRT
Open Close Baud Port CR/LF
Data File Image Com: 3 Clear Script File Script Item Script Control About
0.50 : 0.50 : 5.00 : 5.00
Plot
20.000
0.000

```
0.7040
0.7040
0.7040
0.7040
0.7040
0.7040
0.6336
0.5960
0.5816
0.5762
0.5742
0.5735
0.5732
0.5731
0.5730
0.5730
0.5730
0.5730
0.5731
0.5731
0.5731
0.5730
0.5731
0.5730
```

Error -0.75%


Simple CRT
Open Close Baud Port CR/LF
Data File Image Com: 3 Clear Script File Script Item Script Control About
0.50 : 0.50 : 5.00 : 5.00
Plot
20.000
0.000

```
0.5731
0.5731
0.5731
0.5731
0.5731
0.5731
0.8573
0.9409
0.9699
0.9803
0.9841
0.9855
0.9860
0.9862
0.9862
0.9863
0.9863
0.9863
0.9863
0.9863
0.9863
0.9863
0.9863
0.9863
```

Rise time = 120us Error 1.37%

<u>Conclusion:</u>

Delta-Sigma conversion can improve ADC resolution and accuracy. Keep in mind that these experiments were not long-term stability tests. The stability of the 3.3V regulator will have a large effect on gain error. Temperature tolerances of the resistors will affect gain and offset.