Calibrating the Labmaster

Carolyne Pickler, Neil Edelman

February 5, 2007

Abstract

The two channels of the DAC (Digital to Analog Converters) and the four of the ADC (Analog to Digital Converters) of the Labmaster unit were calibrated. These calibrations yielded a linear relationship between the voltage outputed by the DAC and the inputed DAC number. The same was true of the relationship between the outputed ADC number and the inputed voltage. With these calibrations, a software function generator was written using Labmaster for output that could create sine, square, sawtooth, and triangle waves dependent on user selected amplitude and frequency.

1 Introduction

The DACs and ADCs were calibrated. This calibration determined a relationship between the DAC or ADC number and the voltage. The ADC and DAC numbers are 12 bit, which means the numbers they deal with range from -2048 to 2047. The voltages are limited to a range of -10 V to 10 V[1]. Linear regression was used to calculate the relationship of the DAC or ADC number with its voltage[2]. In regression, the relationship between two variables is analyzed statistically. This analysis considers one variable dependent on the other. These dependent variables are thought to be of random variation while the independent ones were assigned. Using this knowledge, one tries to determine a relationship between the two variables. In the case of linear regression, one analyzes the linear relationship between the two variables. To accomplish this, one must first plot the dependent variables as a function of the independent ones. By examining this graph one can determine what type of regression is suitable. If the relation between the two variables appears to be linear then it can be described by the following equation.

$$y = a + bx \tag{1}$$

where y represents the dependent variable, x represents the independent variable, a is the y intercept and b is the slope of line that is representative of the relationship between x and y. With the following equation, one can calculate the slope.

$$b = \frac{y_2 - y_1}{x_2 - x_1} \tag{2}$$

In equation 2, (x_1, y_1) and (x_2, y_2) represent points on the line describing the relation of y to x. Since a and b are constant, if we know one of the variables, x or y, then we can determine the other[3]. This linear regression allowed us to calibrate the DACs and ADCs. To produce various waveforms, such as sine, square, sawtooth, and triangle, with our software function generator, we need to understand how these functions are produced. A sine wave is described by the following equation.

$$y = a\sin((\omega t) + \theta) \tag{3}$$

where a is the amplitude, ω is the angular frequency, and θ is the phase angle. It takes one period, the angular frequency divided by two π, to complete one cycle[4]. The square wave alternates periodically and instantly between two levels [5]. The sawtooth wave increases linearly and then drops sharply [6]. For the triangle wave, it increases linearly for sometime and then decreases in the same linear fashion for the same amount of time[7]. For our software function generator, the amplitude and frequency of each waveform is determined by the user. The amplitude is the maximum displacement from equilibrium while the frequency is the number of oscillations per second[8]. The importance of calibrating the Labmaster is evident when trying to create a software function using the Labmaster as an input. The calibration allows us to compare with values taken from another unit because they are all calibrated with respect to the same standard. It also allows us to know the error on our measurements and be able to use these values for othe measurements. Finally, it allows us to be able to determine what value we must enter to obtain a specific desired result.

2 Experimental Methods

To calibrate each channel of the DAC, the desired channel to calibrate was attached to the multimetre. Between -2048 and 2047 and in steps of 16, DAC numbers were inputed to the DAC and the outputed voltage was measured on the multimetre and recorded. When calibrating each channel of the ADC, we connected the channel to be calibrated to one of the already calibrated DAC channels. By connected these two, we would be able to know what the inputed voltage to the ADC was as we measured its ADC number while we ran through the same 16 steps we did to calibrate the DAC. To produce the waveforms, the DAC channel 0 was attached to the oscilloscope's source to act as a timing trigger while the first channel was attached to the oscilloscope's second channel to act as the input.

3 Results

To be able to calibrate the DAC channels, zero and one, we plotted the DAC input number as a function of its output voltage. The inputed DAC numbers ranged from -2048 to 2047 [1]. We divided them up into steps of 16 for increased accuracy. For DAC channel zero,

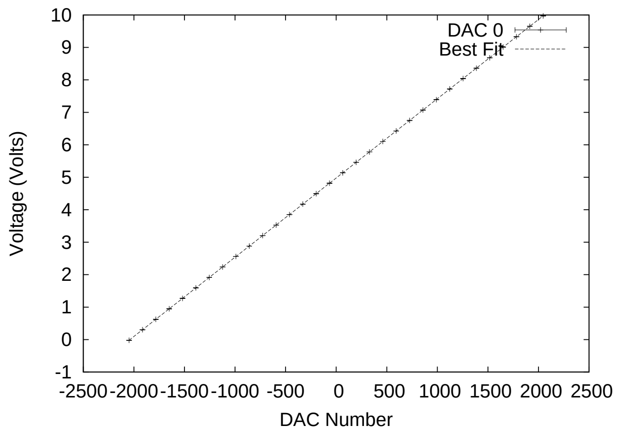

Figure 1: The DAC 0 input numbers vs. the Voltage read from the DAC.

Figure 1 shows that there appears to be a linear relationship between the DAC number and the voltage. It also allows us to see that the DAC is limited to voltages of 0 to 10 V. To verify this assumption of linearity, we plotted the residual voltage as a function of the DAC number.

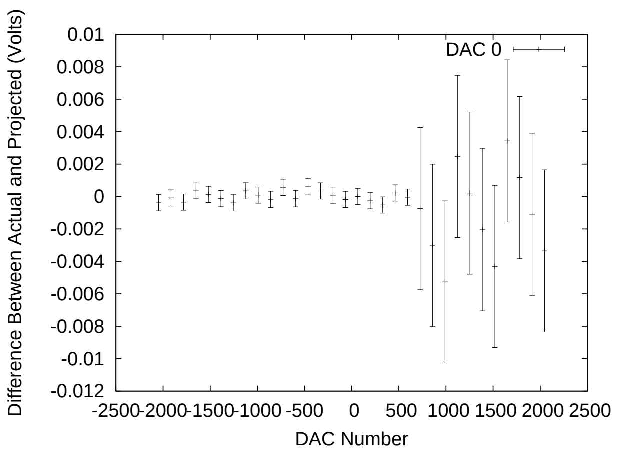

Figure 2: The residual DAC 0 when the best fit is applied.

Figure 2 shows that the residual remains around the zero of the y-axis, which confirms the supposed linear relationship between the voltage and DAC number. When the voltage and DAC numbers were inputed into Linfit, a programme calculating linear regression, we got the following equation.

$$V_{DAC} = a + bN_{DAC} \tag{4}$$

where V_{DAC} represents the voltage outputed by the DAC and N_{DAC} is the inputed DAC number; $a = 4.9783 \pm 0.0001$ V; $b = 0.0024414 \pm 1.3 \times 10^-7$ V. For the other DAC channel, the following figure was obtained.

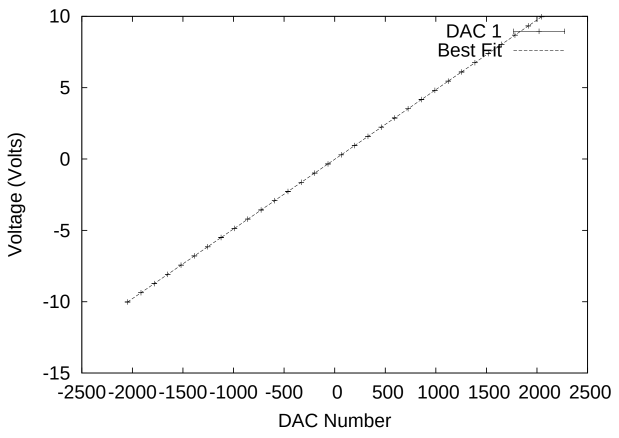

Figure 3: The DAC 1 input numbers vs. the Voltage read from the DAC.

As with Figure 1, it appears that the relationship between the voltage and the DAC number is linear but to ensure this, we again ploted the residual as a function of the DAC number as seen in the figure below. The above figure also shows us that this DAC is limited to voltages between -10 V and 10 V.

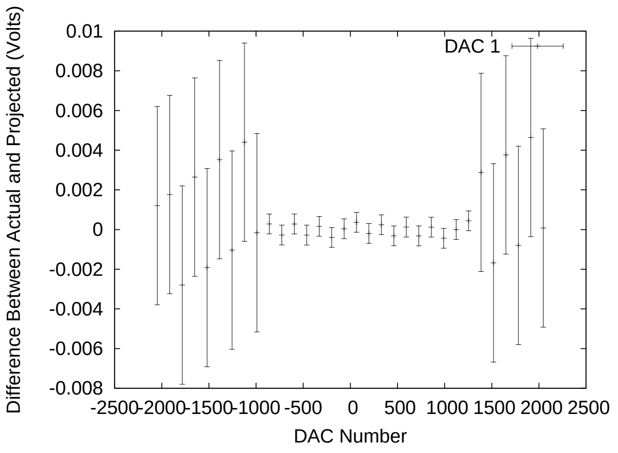

Figure 4: The residual DAC 1 when the best fit is applied.

Figure 4 shows that the residual voltage is centred around the zero of the y-axis. This supports the assumption that the voltage and the DAC number are linearly related. One can also see in figures 2 and 4 that the large error bars are inclined up while the smaller ones are inclined down. This shows the inconsistency of the thing we are measuring with, in this case the multimetre; there is no way to improve this result, as it will always occur. We can also see in these figures that the error bars changed size, small to large, when we went from 3 significant figures to only two. As with the zero channel, when the voltage and DAC numbers were inputed into Linfit, we got the following equation.

$$V_{DAC} = a + bN_{DAC} \tag{5}$$

$a = -0.0208 \pm 0.0001$ V; $b = 0.005 \pm 0.005$ V. To calibrate the ADC channels, zero through three, each channel was attached to each channel of the DAC. Going through the same steps of 16, as with the DAC calibration, we plotted the voltage outputed from the DAC, the input to the ADC, as a function of the ADC number outputed. For the ADC and DAC zero channels, we got the following figure.

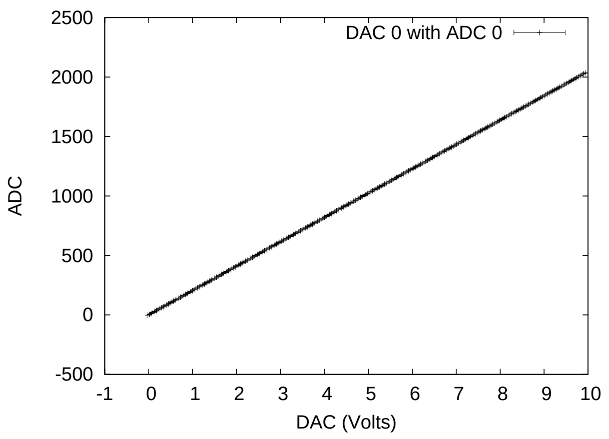

Figure 5: Connecting the DAC 0 with the best fit applied to the ADC 0 giving a unitless ADC number.

As seen in this graph, the relationship between the voltage and the ADC number is linear. When these values were inputed into Linfit, we got the following relationship.

$$N_{ADC} = 0.70 \pm 0.03 + ((204.6920 \pm 0.0006)V^{-1})V_{ADC} \tag{6}$$

where V_{ADC} is the voltage inputed to the ADC and N_{ADC} is the outputed ADC number. The same DAC channel was then connected to channel one of the ADC. This produced the following figure.

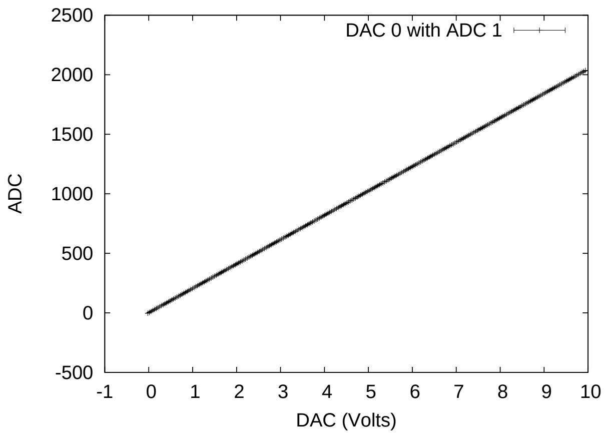

Figure 6: Connecting the DAC 0 with the best fit applied to the ADC 1 giving a unitless ADC number.

As with the other ADC channel, the relationship is linear and is represented by the following relationship, which was derived using Linfit.

$$N_{ADC} = 0.767 \pm 0.003 + ((204.6880 \pm 0.0006)V^{-1})V_{ADC} \tag{7}$$

For channel 2 of the ADC and the same DAC channel, we got the following figure.

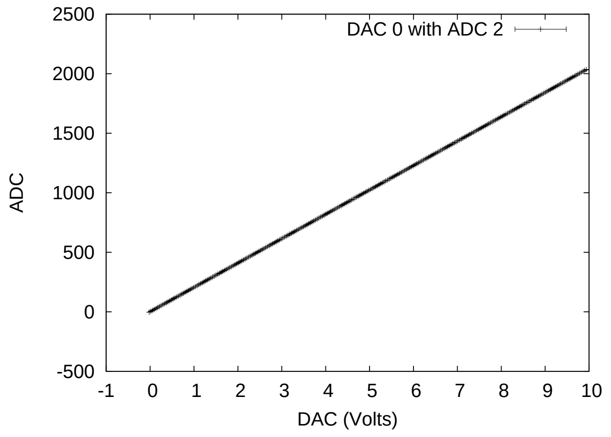

Figure 7: Connecting the DAC 0 with the best fit applied to the ADC 2 giving a unitless ADC number.

The linear relationship describing this, as determined by Linfit, can be seen in the equation below.

$$N_{ADC} = 0.686 \pm 0.003 + ((204.7000 \pm 0.0006)V^{-1})V_{ADC} \tag{8}$$

Finally, the zero channel of the DAC was connected to the third channel of the ADC.

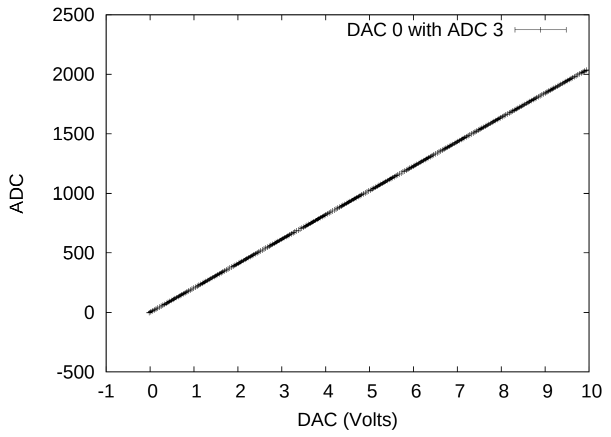

Figure 8: Connecting the DAC 0 with the best fit applied to the ADC 3 giving a unitless ADC number.

This figure shows a linear relationship between the input and output which can be described by the following equation.

$$N_{ADC} = 0.782 \pm 0.003 + ((204.6910 \pm 0.0006)V^{-1})V_{ADC} \tag{9}$$

For ADC channel zero and DAC channel one, we got the following figure.

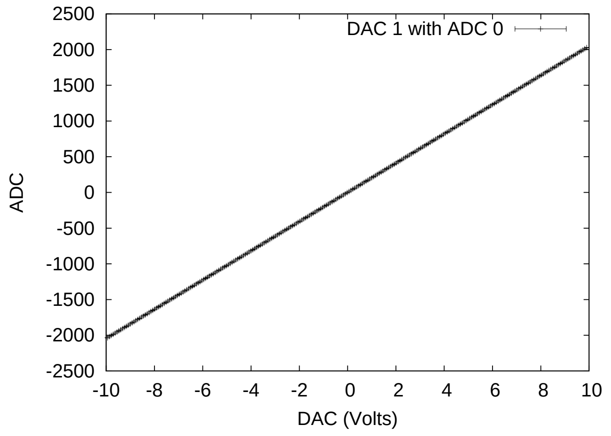

Figure 9: Connecting the DAC 1 with the best fit applied to the ADC 0 giving a unitless ADC number.

We can see in the above figure a linear relationship between the voltage and ADC number. This can be described by the following equation.

$$N_{ADC} = 1.047 \pm 0.002 + ((204.6700 \pm 0.0003)V^{-1})V_{ADC} \tag{10}$$

For the same DAC channel attached to channel one of the ADC,

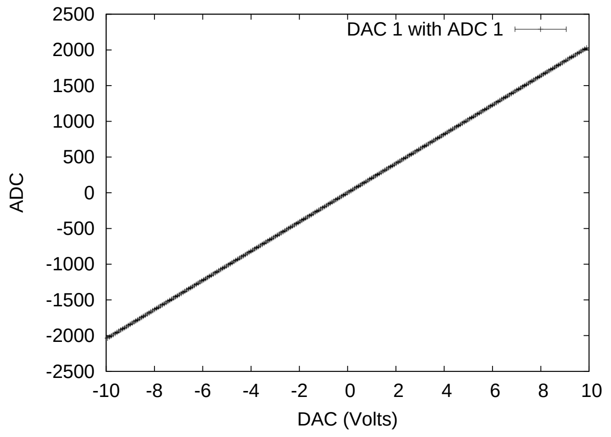

Figure 10: Connecting the DAC 1 with the best fit applied to the ADC 1 giving a unitless ADC number.

The linear relationship seen in the above figure can be described by the following equation.

$$N_{ADC} = 1.117 \pm 0.002 + ((204.6640 \pm 0.0003)V^{-1})V_{ADC} \tag{11}$$

The linear relationship between channel 2 of the ADC and channel 1 of the DAC is seen in the following figure.

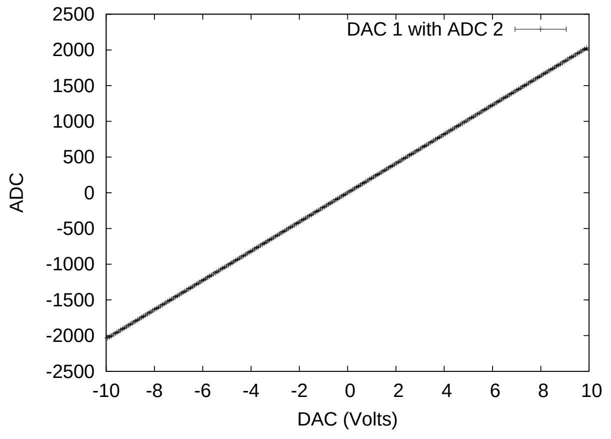

Figure 11: Connecting the DAC 1 with the best fit applied to the ADC 2 giving a unitless ADC number.

The observed relationship is described in the following equation.

$$N_{ADC} = 1.096{\pm}0.002{+}((204.6640{\pm}0.0003)V^{-1})V_{ADC} \quad (12)$$

Finally, the DAC was attached to the third channel of the ADC.

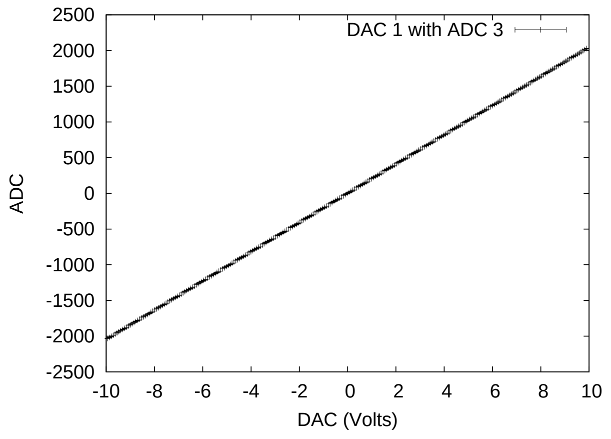

Figure 12: Connecting the DAC 1 with the best fit applied to the ADC 3 giving a unitless ADC number.

The relationship seen in the above figure is linear and can be explained by the following equation.

$$N_{ADC} = 1.223{\pm}0.002{+}((204.6550{\pm}0.0003)V^{-1})V_{ADC} \quad (13)$$

For each ADC calibration, 256 values were taken to get the uncertainty of the measurement.

4 Discussion

The DAC and ADC were calibrated. In each case, the relationship between the voltage and the ADC or DAC number was found to be linear, as seen in figures 1, 3, and 5 through 12. We discovered by plotting the voltage as a function of DAC number what the voltage limits were for each DAC channel. For channel 0, those limits were 0 to 10 V while those for channel 1 are -10 V to 10 V. The precision of the DAC is to within a half of the least-significant bit. This means it is dependent on what we're using to measure. When examining the linear relation equations for each ADC channel with a specific DAC channel, one can see that they are similar. For our software function generator, there are limitations on the user inputed amplitude and frequency. The amplitude is limited by the voltage limit of the channel. In the case of our function generator, the amplitude is limited to -10 V to 10 V. For frequency, it is limited to a value less than half the sampling frequency of 115000 Hz and to a value greater than 10 Hz.

5 Concusion

Once calibrated, we were able to determine that the ADC and DAC are governed by linear relationships between the voltage and the DAC or ADC number. The DAC channels also have voltage limits, channel 0 is between 0 and 10 V while channel 1 is between -10 V and 10 V. The inputed voltage values of the ADC dependent on the limitations of the connected DAC channel. The limitations on the ADC output are -2048 to 2047, which are those for a 12 bit device. This calibration proved useful when creating the software function generator. This function generator uses the Labmaster as input and having user defined amplitude and frequency. We determined that these have limitations placed upon them. The frequency was limited to being below half of the sampling frequency and above 10 Hz. The amplitude must remain within the DAC channel's voltage limit.

References

[1] *Labmaster DMA Handbook*, pages 13–3. Scientific Solutions Inc., 1987.

[2] O. Kempthorne K. Hinkelmann. *Design and Analysis of Experiments*, pages 60 – 74. John Wiley & Sons, 1994.

[3] S. Bernstein and R. Bernstein. *Elements of Statistics II: Inferential Statistics*, page 334. McGraw Hill, 1999.

[4] J. Edminister M. Nahvi. *Electric Circuits*, page 103. McGraw Hill, 2003.

[5] J. Edminister M. Nahvi. *Electric Circuits*, page 102. McGraw Hill, 2003.

[6] J. Edminister M. Nahvi. *Electric Circuits*, page 420. McGraw Hill, 2003.

[7] J. Edminister M. Nahvi. *Electric Circuits*, page 422. McGraw Hill, 2003.

[8] P. Tipler. *Physics for Scientists and Engineers*, page 404. Worth Publishers, 1999.

A DAC Values: dac.h

```
/* our eqiupment DAC, by experiment, y = a + bx where y is volts */
const static double a[] = {
  4.97831,
  -0.0207601
};
const static double aSigma[] = {
  0.000145044,
  0.0001258
};
const static double b[] = {
  0.00244137,
  0.00488303
};
const static double bSigma[] = {
  0.000000133962,
  0.000000182228
};
const static double rChiSq[] = {
  0.391955,
  0.317828
};
```

B Programme that Calculates the DAC: dac.c

```
/* this calculates the DAC number based on the DAC and the Voltage */

#include <stdio.h> /* [f]printf */
#include <stdlib.h> /* sscanf */
#include "dac.h"

/* prints off the Ndac based on the voltage */
int main(int argc, char **argv) {
  int n;
  double Vdac, Ndac;

  /* initialize arguments */
  if(argc <= 2) { fprintf(stderr, "Usage: dac <dac> <value>\n"); return 1; }
  if(sscanf(argv[1], "%d", &n) != 1) {
    fprintf(stderr, "Ca’n’t parse DAC.\n");
    return 1;
  }
  if(n < 0 || n > 1) { fprintf(stderr, "DAC out-of-bounds.\n"); return 1; }
  if(sscanf(argv[1], "%lf", &Vdac) != 1) {
    fprintf(stderr, "Ca’n’t parse input Voltage.\n");
    return 1;
  }

  /* Vdac = a + (b * Ndac); */
  Ndac = (Vdac - a[n]) / b[n];
  printf("%g\n", Ndac);
  return 0;
}
```

C Programme that Connects DAC to the ADC: adc.c

```
/* connect the DAC to to ADC and call this with adc <dac> <adc> to get data
   MUST be linked with the Labmaster! eg
   gcc -Wall -O3 -ansi -pedantic -lm -llabmaster -o adc adc.c */

#include <stdio.h>     /* [f]printf */
#include <stdlib.h>    /* sscanf */
#include <math.h>      /* sqrt */
#include <labmaster.h> /* labmaster calls */
#include "dac.h"

const static int dacStep = 16;
const static int iterations = 256;

int main(int argc, char **argv) {
  int dac, adc, dacIn, iter;
  double dacOut, adcOut, adcErr, adcUnc, sum, sumSq;

  /* initialize arguments */
  if(argc < 3) { fprintf(stderr, "Usage: adc <dac> <adc>\n"); return 1; }
  if(sscanf(argv[1], "%d", &dac) != 1) {
    fprintf(stderr, "Ca’n’t parse DAC.\n");
    return 1;
  }
  if(dac < 0 || dac > 1) { fprintf(stderr, "DAC out-of-bounds.\n"); return 1; }
  if(sscanf(argv[2], "%d", &adc) != 1) {
    fprintf(stderr, "Ca’n’t parse ADC.\n");
    return 1;
  }
  if(adc < 0 || adc > 3) { fprintf(stderr, "ADC out-of-bounds.\n"); return 1; }

  /* "This step is critical," OK */
  labmaster_initialize();

  /* connect DAC to ADC */
  for(dacIn = -2048; dacIn < 2048; dacIn += dacStep) {
    /* dac output */
    out_dac(dac, dacIn);
    /* dac translated to volts */
    dacOut = a[dac] + (b[dac] * dacIn);
    /* adc out, average, error */
    for(iter = 0, sum = 0, sumSq = 0; iter < iterations; iter++) {
      adcOut = in_adc(adc);
      sum += adcOut;
      sumSq += adcOut * adcOut;
    }
    adcOut = sum / iterations;
    adcErr = sqrt(sumSq / iterations - adcOut * adcOut);
    adcUnc = adcErr / sqrt(iterations);
    /* points were getting saturated, hence if() */
    if(adcUnc) printf("%g %g %g\n", dacOut, adcOut, adcUnc);
  }

  /* I assume this step is critical too */
  labmaster_terminate();

  return 0;
}
```

D Programme that Outputs to the Scope: drawing.c

```
/* programme that outputs to a scope using the "LabMaster" set of libraries;
   MUST be linked with the Labmaster! eg
   gcc -Wall -O3 -ansi -pedantic -lm -llabmaster -o drawing drawing.c */

#include <stdio.h>     /* (f)printf stderr */
#include <stdlib.h>    /* sscanf */
#include <math.h>      /* sqrt */
#include <string.h>    /* strncmp */
#include <labmaster.h> /* out_dac etc */
#include "dac.h"

/* prototypes */
int main(int argc, char **argv);
int VtoDAC(double v);
void fillSinwave(int *buffer, double amp, int length);
void fillTriangle(int *buffer, double amp, int length);
void fillSquarewave(int *buffer, double amp, int length);
void fillSawtooth(int *buffer, double amp, int length);
void fillCalibration(int *buffer, double amp, int length);

/* constants */
const static int timerDAC = 0;  /* the timer DAC */
const static int outDAC = 1;    /* the output DAC */
const static double pi = 3.14159265358979323846264338327950288419716939937510\
58209749445923078164062862089986; /* etc; since ansi apperantly does’n’t define
   M_PI */

/* main (duh?) */
int main(int argc, char **argv) {
  void (*fn)(int *, double, int);
  int i;
  int length; /* length of the buffer @= the wavelength */
  int *buffer;
  double amp, fre;

  /* initialize arguments */
  if((argc > 1) && (!strncmp(argv[1], "-h", 2) || !strncmp(argv[1], "--h", 3))) {
    fprintf(stderr, "Usage: drawing [sinewave|triangle|squarewave|sawtooth] ");
    fprintf(stderr, "[amplitude] [frequency]\n");
    return 1;
  }

  /* fill with def val */
  fn = fillSinwave;
  amp = 1;
  length = 1024;

  /* user input */
  if(argc > 1) {
    if     (strncmp(argv[1], "sawtooth",    2) == 0) fn = fillSawtooth;
    else if(strncmp(argv[1], "squarewave",  2) == 0) fn = fillSquarewave;
    else if(strncmp(argv[1], "triangle",    2) == 0) fn = fillTriangle;
    else if(strncmp(argv[1], "sinewave",    2) == 0) fn = fillSinwave;
    else if(strncmp(argv[1], "calibration", 2) == 0) fn = fillCalibration;
    else {
      fprintf(stderr, "%s?\n", argv[1]);
      return 1;
    }
  }
  if(argc > 2) {
    if(sscanf(argv[2], "%lg", &amp) != 1) {
      fprintf(stderr, "Ca’n’t parse number; use --h for help.\n");
      return 1;
    }
    fprintf(stderr, "Using amp = %g.\n", amp);
  }
  if(argc > 3) {
    if(sscanf(argv[3], "%lg", &fre) != 1) {
      fprintf(stderr, "Ca’n’t parse number; use --h for help.\n");
      return 1;
    }
    if(fre <= 10) {
      fprintf(stderr, "Frequency must be 10.\n");
      return 1;
    }
    length = 115000 / (double)fre;
    fprintf(stderr, "Using fre = %g, len = %d.\n", fre, length);
  }

  buffer = malloc(length * sizeof(int));
  if(!buffer) {
    fprintf(stderr, "Wavelength too big for memory allocation: %d.", length);
    return 0; /* mucho scecthy! */
  }

  /* call *fn to fill the buffer */
  (*fn)(buffer, amp, length);

  /* "This step is critical," OK */
  labmaster_initialize();

  /* display instuctions */
  printf("Connect DAC-%d to the timer trigger, ext.\n", timerDAC);
  printf("Connect DAC-%d to Y.\n", outDAC);
```

```
  /* system dependent */
  printf("Ctrl-C to exit?\n");

  /* display infinitaly (FIXME: timer?) */
  for( ; ; ) {
    /* for every buffer[i] */
    for(i = 0; i < length; i++) {
      /* -’ve pulse to trigger scope */
      out_dac(timerDAC, (i == 0) ? (-2048) : (2047));
      /* this is the output */
      out_dac(outDAC, buffer[i]);
    }
  }

  /* I assume this step is critical too */
  /* FIXME: but it does’n’t get executed ever since Ctrl-C is the
     only way to end */
  labmaster_terminate();

  /* the fn that’s never called */
  free(buffer);

  return 0;
}

/* volts to DAC; in dac.h */
int VtoDAC (double v) { return (v - a[outDAC]) / b[outDAC]; }

/* sinwave */
void fillSinwave(int *buffer, double amp, int length) {
  int i;
  double x;

  for(i = 0; i < length; i++) {
    x = i * 2 * pi / length;
    buffer[i] = VtoDAC(sin(x) * amp);
  }
}

/* triangle */
void fillTriangle(int *buffer, double amp, int length) {
  int i;
  double x;

  for(i = 0; i < length; i++) {
    x = i / (double)length;
    if(x < .5) buffer[i] = VtoDAC((-1 + 4 * x) * amp);
    else       buffer[i] = VtoDAC(( 3 - 4 * x) * amp);
  }
}

/* squarewave */
void fillSquarewave(int *buffer, double amp, int length) {
  int i;
  double x;

  for(i = 0; i < length; i++) {
    x = i / (double)length;
    if(x < .5) buffer[i] = VtoDAC( amp);
    else       buffer[i] = VtoDAC(-amp);
  }
}

/* sawtooth */
void fillSawtooth(int *buffer, double amp, int length) {
  int i;
  double x;

  for(i = 0; i < length; i++) {
    x = i / (double)length;
    buffer[i] = VtoDAC((-1 + 2 * x) * amp);
  }
}

/* calibration; FIXME: calibration is not implemented, the only thing to do
   is change the code, which is Bad, but suits our needs */
void fillCalibration(int *buffer, double amp, int length) {
  int i;

  fprintf(stderr,
  "Calibration . . . sampling frequency = 2 measured frequency.\n");
  for(i = 0; i < length; i++) {
    buffer[i] = (i & 1) ? (2047) : (-2048);
  }
}
```