ΣΔ ADCs and DACs for FPGAs

Jesús Arias Álvarez

1 Introduction

FPGAs are very interesting devices that allows the physical implementation of complex digital systems in a very convenient way, but they lack an integrated interface with the analog world. Of course, external ADCs and/or DACs can be attached to the digital pins of the FPGA when needed, but here I want to present a way to keep most of the related hardware inside the FPGA, while reducing the analog blocks to a minimum, and there are two main tricks to do it:

1. Pulse Width Modulation, PWM, where the analog information is encoded in the width of a rectangular pulse.

2. Pulse Density Modulation, also known as Sigma-Delta Modulation, where the analog information is encoded in the average number of high pulses in a bitstream.

PWM is easily accomplished in the digital domain using counters and comparators, and in the analog domain by resorting to triangular waveforms and analog comparators. This technique isn't discussed here, just let me remark its main problems when used for analog to digital or digital to analog converters: low resolution and distortion. In spite of these issues PWM is currently used in a lot of applications, including energy efficient, class-D, audio amplifiers.

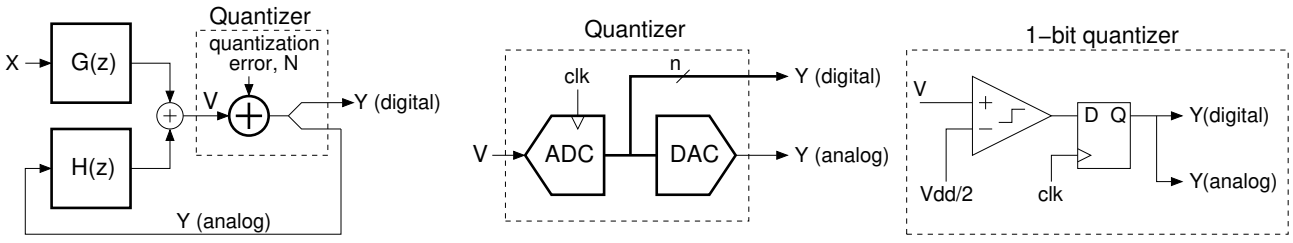
Sigma-delta modulation has proven to be a higher performance solution and it is currently used for almost all audio CODECs around. A sigma-delta converter has two main parts:

1. The ΣΔ modulator that generates the bitstream. This is a purely digital block for DACs and a mainly analog block for ADCs.

2. A low-pass filter that retrieves the average value of the signal. For a DAC this has to be an analog filter, but it can be a quite simple one, sometimes even an implicit one, while for an ADC the filter is implemented in the digital domain.

So, lets present some theory about ΣΔ modulators first.

2 ΣΔ modulation, a brief theoretical introduction.



The general diagram of a ΣΔ modulator is shown in the above figure as composed of several filter blocks, H(z), and G(z), and a quantizer. The quantizer itself is composed of an ADC followed by a DAC, so, the analog

output is the same signal as the input but restricted to a finite number of values (quantized), and a digital copy of the output is also available. A lot of $\Sigma\Delta$ ADCs have quantizers with just a single bit of resolution, and in this case the quantizer can be built with just a comparator. Quantizers also are sampling circuits and therefore a flip-flop is added after the analog comparator.

In order to analyze the modulator it is a good approximation to consider that the quantizer has at its output the same signal at the input plus a quantization error, and we can assume this error is uncorrelated with the input signal (this isn't true, is just a coarse approximation to reality). Therefore we can write:

$$Y = NTF(z) \cdot N + STF(z) \cdot X$$

Where NTF(z) is the so called Noise Transfer Function and STF(z) the Signal Transfer Function, that can be obtained by just making zero X or N, and solving the equation:

$$NTF(z) = \frac{Y}{N} = \frac{1}{1 - H(z)}$$

$$STF(z) = \frac{Y}{X} = \frac{G(z)}{1 - H(z)} = G(z)NTF(z)$$

In a $\Sigma\Delta$ modulator we want a high-pass frequency response for the NTF, so this function is designed to be a cascade of derivators:

$$NTF(z) = (1 - z^{-1})^L$$

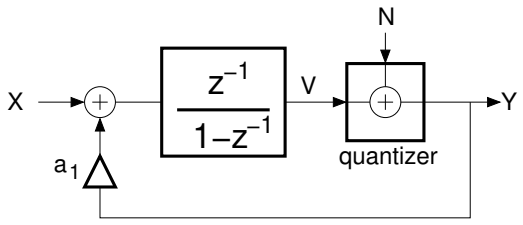
Where L is 1 for first order modulators or 2 for second order modulators. Higher values of L will result in unstable modulators. With these NTFs the quantization error, N, is strongly attenuated for low frequencies, that is just where our signal of interest, X, is.

So, the NTF is the critical transfer function of the $\Sigma\Delta$ modulator, while the STF can be just an all-pass filter:

$$STF(z) = z^{-M}$$

With $M = 0$, or $M = 1$, or maybe even a low-pass filter, it doesn't matter too much.

2.1 First order modulator



A simple block diagram for a first order modulator is shown in the previous figure, were along with the quantizer we get a discrete-time integrator. And solving for the NTF we get:

$$NTF = \frac{Y}{N} = \frac{1 - z^{-1}}{1 - (1 + a_1)z^{-1}}$$

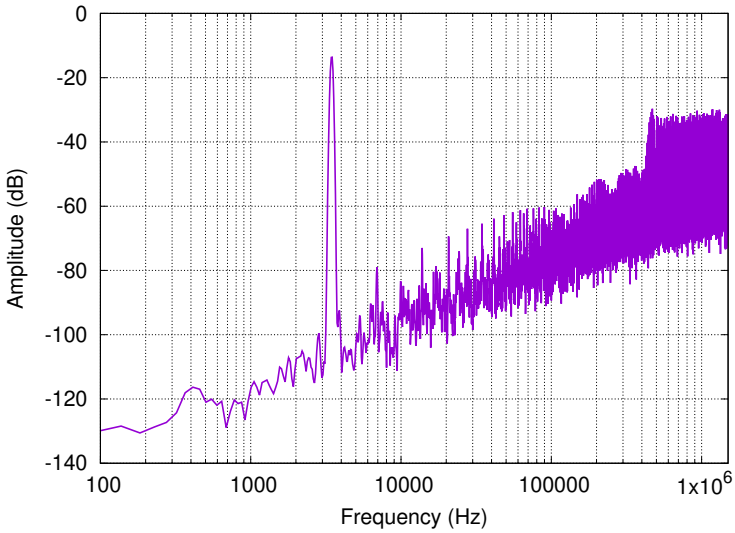
We want just: $NTF = 1 - z^{-1}$, and therefore a_1 must be -1. The resulting STF is just $STF = z^{-1}$

This modulator can be easily simulated, lets assume the quantizer generates a value of +1 if V is positive or -1 if V is negative. And the integrator recurrence equation is just:

$$v_i = v_{i-1} + (x_{i-1} - y_{i-1})$$

For x_i lets use the samples of a low frequency sine wave. The resulting simulation code written in C is:

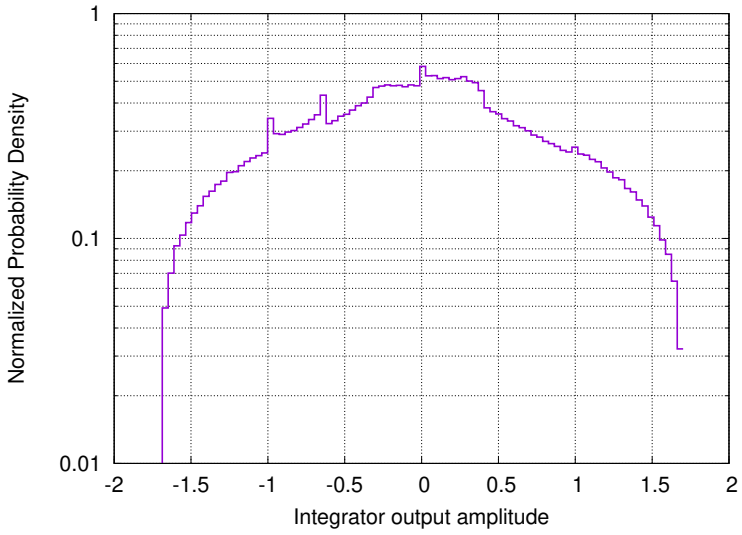
```c
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#define FS (3E6)
#define DT (1.0/FS)

#define FREQ (3456.7)
#define AMP (0.7)
#define NS (1<<18)

main()
{
    int i,o;
    double ph, x, v, y;

    ph=0; x=0; v=0;
    for (i=0;i<NS;i++) {
        // Quantizer
        o=(v>=0)? 1 : 0;    // Digital
        y=(v>=0)? 1 : -1;   // Analog
        // integrator
        v+=x-y;
        // Input sinewave
        x= AMP*sin(ph);
        ph+=FREQ*2.0*M_PI/FS;
        if (ph>=2*M_PI) ph-=2*M_PI;
        // listing
        printf("%g %g %g %d\n",i*DT*1e6, x,v,o);
    }
}
```



This simple code generates a long bitstream whose power spectrum is shown on the right side. As we can see, the input tone at 3456.7Hz clearly stands over a noise floor that increases with the frequency at a rate of 20dB/decade. If the signal bandwidth is limited to 1/256 of the clock frequency (here assumed to be 3MHz), the resulting signal to noise ratio, SNR, is 63.25 dB. Therefore, if this modulator is followed by a filter that removes any signal content over 11.7kHz the resulting $\Sigma\Delta$ ADC will have an effective number of bits of:

$$ENOB = (SNR_{MAX} - 1.76)/6.02 = 10.2\,bits$$

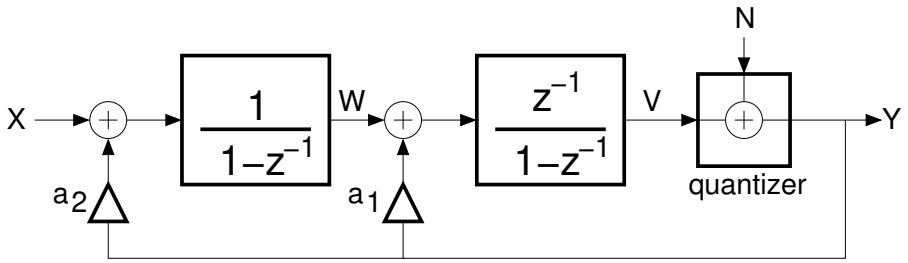
Also notice that $\Sigma\Delta$ modulators usually reach their maximum SNR when the tone amplitude is about a 70% of the quantizer output. This isn't a surprise because when the input signal is too high or too low we get only a few sparse low or high pulses at the output, and we can't get a good average.



The simulation program also gives us the simulated signal at the integrator output, V, and it is interesting to see what values it have. In the above histogram we can see the maximum output amplitude of the integrator is higher than the quantizer output (± 1.7), and for a practical circuit this would mean the integrator is going require a higher supply voltage than the logic of the quantizer. But that is not a worry because we can reduce

the gain of the integrator as far as we like and the quantizer output is going to be the same because for the comparator only the sign of the integrator output matters. So, we can use lets say, an integrator with a gain of 0.5 (just by doubling its capacitor) and the maximum amplitude would be 0.85, or we can reduce the gain a lot more and the integrator can be implemented with just a simple passive R-C circuit.

2.2 Second order modulator



The second order modulator whose block diagram is shown just above is a much better performer because it attenuates the quantization noise a lot more than the first order one. It has two nested integrators and two feedback coefficients, a_1, and a_2, that have to be selected in order to get the following NTF:

$$NTF(z) = (1-z^{-1})^2$$

First, we must solve the circuit for Y/N while making $X = 0$:

$$Y = N + Y\frac{a_1 z^{-1}}{1-z^{-1}} + Y\frac{a_2 z^{-1}}{(1-z^{-1})^2}$$

That results in:

$$NTF(z) = \frac{Y}{N} = \frac{(1-z^{-1})^2}{1+(-2-a_1-a_2)z^{-1}+(1+a_1)z^{-2}}$$

The denominator has to be 1, and therefore:

$$a_1 = -1 \quad ; \quad a_2 = -1$$

The sign of these coefficients means the two feedback loops of the modulator must have a negative feedback. Looking towards simulation, we can also write the recurrence equations for simulation, that are:

$$V(1-z^{-1})^2 = Y(-2z^{-1}+z^{-2}) + Xz^{-1}$$

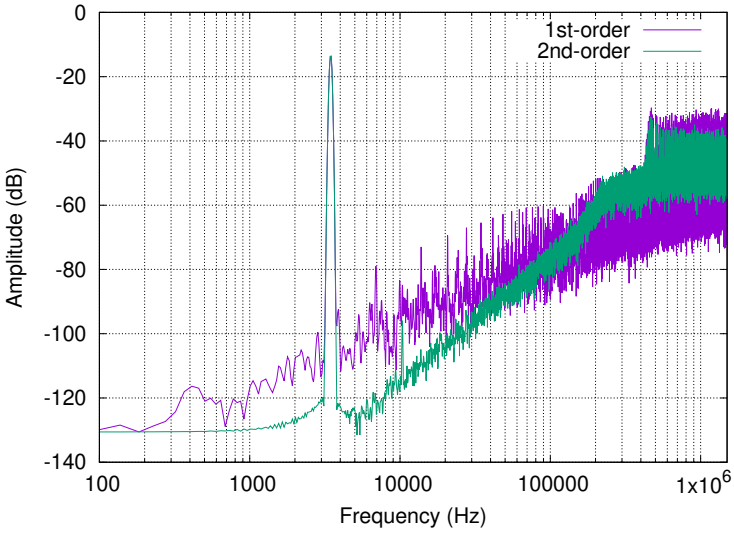
$$or: \quad V_i = 2V_{i-1} - V_{i-2} - 2Y_{i-1} + Y_{i-2} + X_{i-1}$$

And:

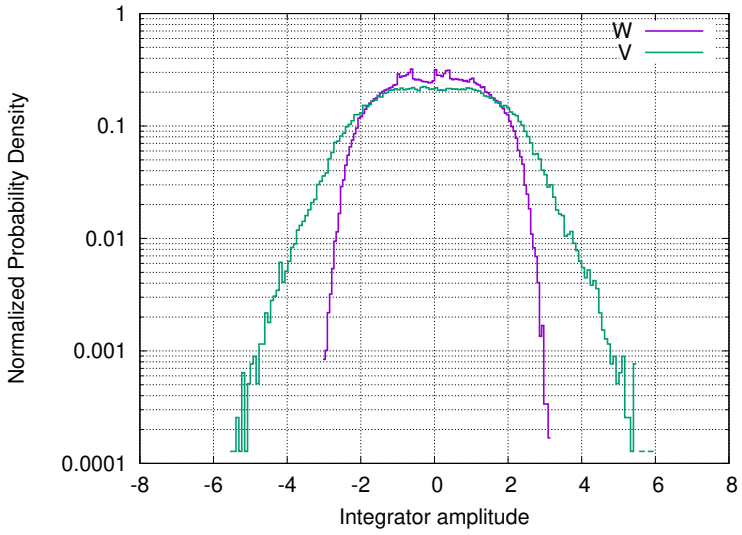
$$W_i = W_{i-1} - Y_i + X_i$$

The simulation code is now:

```c
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#define FS (3e6)
#define DT (1.0/FS)

#define FREQ (3456.7)
#define AMP (0.7)
#define NS (1<<18)

main()
{
    int i,o;
    double ph,x,x1,w,w1,v,v1,v2,y,y1,y2;

    ph=0; x1=0; v1=v2=0; y1=y2=0; w1=0;
    for (i=0;i<NS;i++) {
        // Input Sine wave
        x= AMP*sin(ph);
        ph+=FREQ*2.0*M_PI/FS;
        if (ph>=2*M_PI) ph-=2*M_PI;

        // Integrators
        v=2*v1-v2-2*y1+y2+x1;
        w=w1-y+x;

        // Quantizer
        if (v>0) y=o=1; else {o=0; y=-1;}

        // Listing
        printf("%g %g %g %g %d\n",i*DT*1e6, x,v,w,o);

        // Next cycle
        v2=v1; v1=v;
        y2=y1; y1=y;
        w1=w;
        x1=x;
    }
}
```

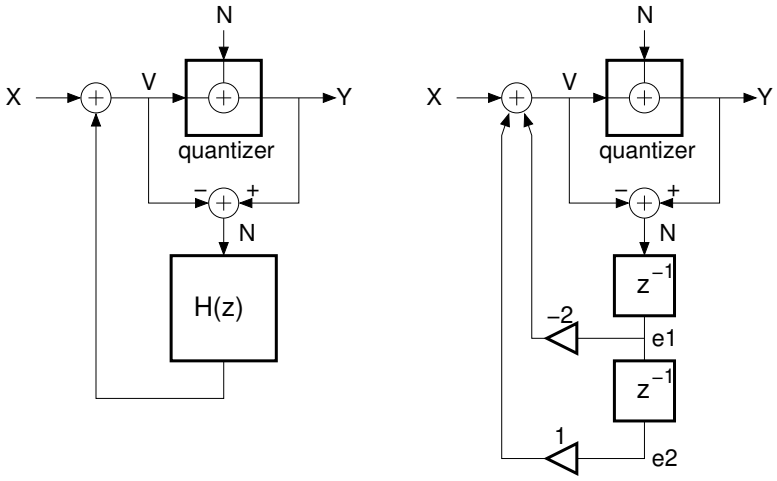


Here we can see how the noise floor of the 2nd-order modulator is much lower than that of the first order, and in fact, the flat spectrum below 1000Hz is an artifact due to the truncated gaussian window used for the FFT, the noise keeps decreasing at a rate of 40dB/decade. With this data we can double the signal bandwidth, up to 23kHz, and the SNR is 72dB resulting in 11.7 effective bits.



But now lets take a look at the amplitude at the integrators. In the above curves we see a lot of amplitude that have to be reduced by lowering the integrators gain. But we must be careful with the first integrator: If the amplitude of W is scaled by 0.25, this output must be multiplied by 4 before attaching it to the second integrator or the NTF will be ruined. The V integrator is only connected to the quantizer and we have no problems with its amplitude as only the sign of its output matters.

2.3 Error-feedback modulator (ΣΔ DACs)



A different block diagram can be used with ΣΔ modulators. In the previous figure a general error-feedback topology is presented, where its easy to see:

$$Y = N(H(z)+1)+X$$

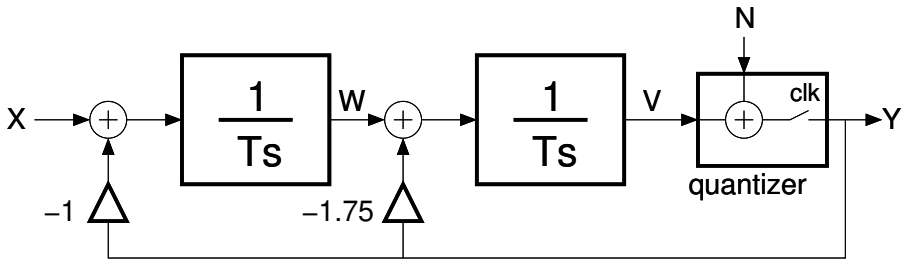
and therefore $H(z) = NTF(z) - 1$, and for a second order modulator $H(z) = -2z^{-1} + z^{-2}$

The resulting second order modulator is displayed on the right. These topologies are ill suited for analog implementations because any inaccuracy in the calculation of N will result in a noise excess at the output. But they are quite simple to build in the digital domain, and thus this is the usual way the ΣΔ modulators for DACs are built.

2.4 Continuous-time ΣΔ modulators

All the modulators presented in this document so far are assumed to be sampled systems, and they are analyzed by means of the Z-transform where a variable multiplied by z^{-n} is equivalent to the same variable n samples before. These analog blocks are usually integrated using switched-capacitor circuits.

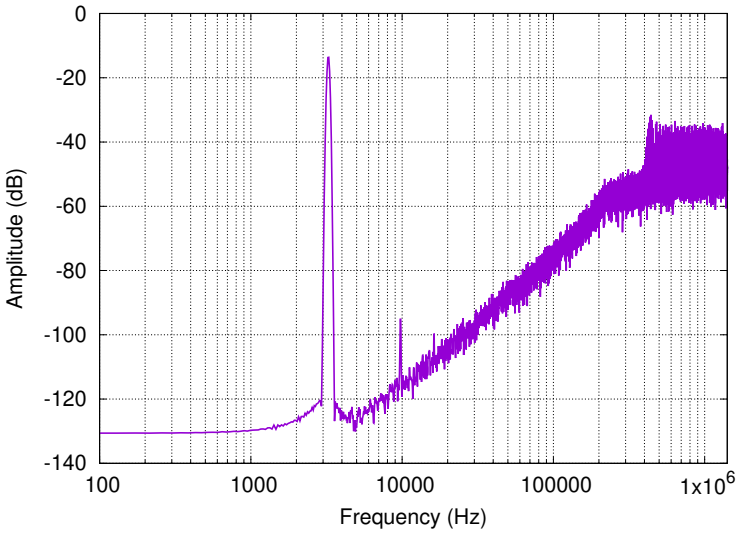
But it is also possible to restrict the sampling to the quantizer and to use the habitual continuous-time integrators for filtering. The analysis is a lot more complex but the resulting block diagrams are almost the same as before, only the feedback coefficients are different. Let me present a second order modulator of the CT type:



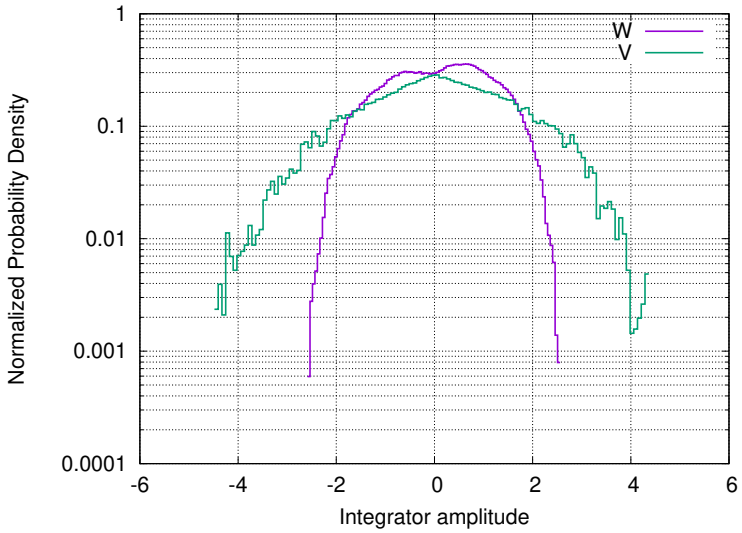
Here T is the period of the sampling clock and we use the Laplace transform to describe the transfer functions of the integrators ($1/s \equiv \int dt$).

These modulators are a bit harder to simulate because each sample now implies many time steps, but they are still affordable. Let me show a code example where each clock cycle is divided into 32 time steps and a simple Euler integration is used during calculations:

```c
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#define FS (3e6)
#define SPS (32)        // Steps per sample
#define T  (1.0/FS)
#define DT (1.0/FS/SPS)

#define FREQ (3456.7)
#define AMP (0.7)
#define NS  (1<<18)

#define A1 (-1.75)
#define A2 (-1.0)

main()
{
    int i,j,o;
    double ph, t, x, vo,vi,wo,wi, y;

    ph=0; vo=wo=0; o=0; t=0;
    for (i=0;i<NS;i++) {
        // quantizer
        if (vo>0) {o=1; y=1; }
        else {o=0; y=-1; }
        // one cycle simulation
        for (j=0;j<SPS;j++) {
            // Input sinewave
            x= AMP*sin(ph);
            ph+=FREQ*2.0*M_PI*DT;
            if (ph>=2*M_PI) ph-=2*M_PI;
            // Integrator inputs
            wi= x + A2*y;
            vi=wo + A1*y;
            // Integration
            wo+=wi*DT/T;
            vo+=vi*DT/T;
            // list values for histograms
            //printf("%g %g %g %g %g\n",
            //          t*1e6,x, wo,vo,y);
            t+=DT;
        }
        // List output
        printf("%d\n",o);
    }
    return 0;
}
```



As we can see, the resulting spectrum is the same as the obtained with the discrete-time modulator. We can also take a look to the W and V amplitudes. In the following histograms these amplitudes are shown, and they can be much higher than 1, so the integrator gains also have to be scaled down:



The physical implementation of these integrators can be made with amplifiers, resistors, and capacitors as usual.

2.5 Non idealities

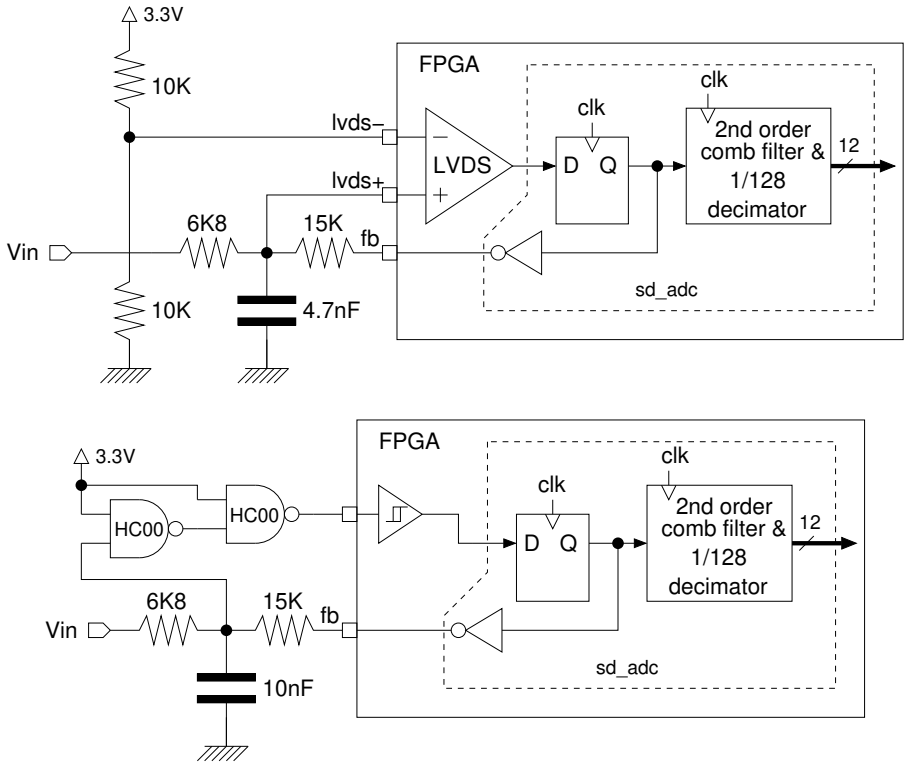
There are quite a lot of real world effects that can degrade the performance of $\Sigma\Delta$ modulators, let me present a few:

- Finite gain in integrators. An ideal integrator should have an infinite gain at 0Hz. This is never the case and the finite DC gain also means the real integrator has a pole in its frequency response at some non zero frequency (it is a first-order low-pass filter, not an ideal integrator). If this frequency is higher than the signal bandwidth the SNR of the modulator can be severely degraded.
- Comparator hysteresis. This is a very undesirable characteristic for a comparator intended for a $\Sigma\Delta$ modulator because it will turn the modulator into a sort of relaxation oscillator with much lower SNR.
- Clock jitter. This is an specific problem for continuous-time modulators because a variable clock period also means a variable integration time that gets translated into excess noise. So, an stable clock with a quartz crystal is advised.
- Electrical noise. Of course noise is always present in real circuits, both in the signals themselves and in the supply rails. This last noise is quite notable when we have digital circuits switching around.

3 Simple sigma-delta ADCs

Lets present the schematics of some $\Sigma\Delta$ ADCs built into an FPGA. All these converters are intended to digitize audio signals and have a signal bandwidth ranging from 10kHz to 20kHz. Of course, they can also be used for the measurement of DC signals, but in that case they may require calibration. In addition to the FPGA, that includes all the digital circuitry of the converter, some analog electronics is also required. Lets start with the simplest ones:

3.1 First-order sigma-delta ADCs



In these circuits the "integrator" was built using just two resistors and one capacitor. This isn't an integrator but a low-pass filter with a pole frequency at 7.2kHz or 3.4kHz. This frequency is comparable with the signal bandwidth, so, it would be desirable to lower that frequency by means of using a bigger capacitor or resistors. But this will also result in less integrator gain, and the electrical noise will be made dominant if further reduced, so, these values are a compromise between quantization noise and electrical noise and were chosen after some experimentation.

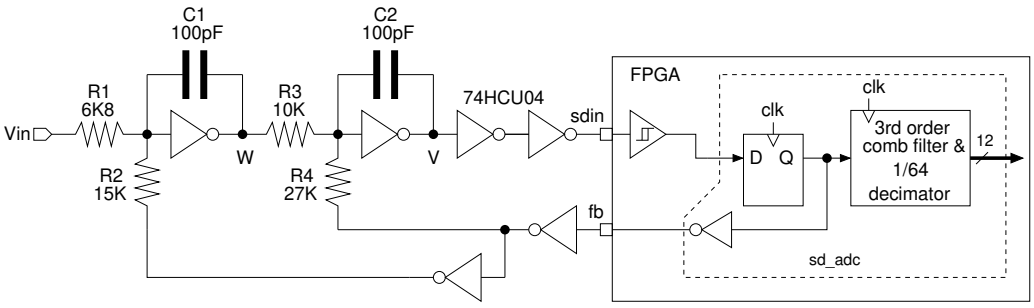
There are two versions of the modulator. The first uses a comparator built into the FPGA that was originally intended for LVDS digital signals and here compares the voltage of the capacitor with a constant reference of 1.65V. This circuit is the preferred one because it would present a low input offset thanks to the accurate voltage comparison, and doesn't require any active component outside the FPGA. But on the other hand it requires two pins for the input and these pins must have to be configured as an LVDS input (not all pins are capable of this).

The second circuit gets rid of the LVDS comparator and uses a conventional CMOS input. But that pin can't be connected to the capacitor directly because these input pins all have a built-in hysteresis. So, a simple external gate is used as a comparator without hysteresis (other gates can be used, not just NANDs). The gate is in fact comparing the capacitor voltage with its internal input threshold, and therefore we can expect a quite big offset that varies from gate to gate. Please don't use here 74HCT parts because these gates have threshold values much lower than Vdd/2.

In terms of SNR and effective number of bits both circuits should be more or less the same, achieving little more than 8 effective bits of resolution, that is a very decent figure for ADCs with so little analog circuitry. And definitely, the preferred circuit is the one with the LVDS comparator because if some external gates are to be included we can build a second order modulator by just adding another capacitor and two resistors, resulting in a much better performance.

Of course, the $\Sigma\Delta$ modulator is only the input block of the converter and a digital filter also has to be included in the FPGA logic. We will talk about these filters later, but now I want to remark that a first-order low-pass filter isn't enough because the quantization noise of the modulator increases at a 20dB/decade rate, so, we need at least a second-order low-pass filter or a lot of quantization noise could end up folded into the signal band during sample decimation (one output sample every 128 modulator cycles).

3.2 Second-order sigma-delta ADC



In this circuit a continuous-time second-order modulator is built around two integrators that are using CMOS inverters as operational amplifiers. Well, an opamp usually have a DC gain over 100000 while the measured gain of these inverters was only -18.5, so, the finite DC gain of these integrators could really be a problem, but lets see. It is very important to use the 74**HCU**04 part because the regular HC04 gates are "buffered", meaning they really are 3 inverters connected in series: first an small inverter for a low capacitance input, then a second inverter for the logic, and finally a big inverter for driving the output load. That kind of inverters can't be used in circuits with feedback loops because they will be unstable and oscillations are guaranteed. On the contrary, the HCU04 chip has single inverters that can be used as amplifiers (that chip is usually found in crystal oscillators).

But let me show who to obtain the component values. First, starting with a clock rate of 2.82MHz the period is $T = 355ns$. This has to be the value of all the RC products, so, if we chose a value for the capacitors as 100pF, resistors will be $R1 = R2 = R3 = 3546\Omega$ and $R4 = R/1.75 = 2026\Omega$.

But these integrators will have too much amplitude, so lets divide the gain of V by about 13, that can be accomplished by multiplying R3 and R4 by 13: $R3 = 47k\Omega$, $R4 = 27k\Omega$.

And then lets divide the gain of the W integrator by about 4, this time multiplying R1 and R2: $R1 = R2 = 15k\Omega$. R1 can be further changed to adjust the input range of the converter, $15k\Omega$ gives an input range of 3.3V.

But now, in order to preserve the gain at the V output, R3 also has to be divided by the same factor, resulting in $R3 = 10k\Omega$. All these values are approximate.

And what about the finite gain of inverters, well, the low frequency pole is located at $\omega_p \approx 1/(A_v RC)$, where A_v is the DC gain of the inverter (18.5), so, for the integrator W the pole is at a frequency $\omega_p = 1/(18.5 \cdot 15k\Omega/2 \cdot 100pF) = 11.4kHz$, still below the signal bandwidth, now 22kHz, and for the integrator V we have: $\omega_p = 1/(18.5 \cdot 10k\Omega || 27k\Omega \cdot 100pF) = 11.8kHz$. So, the small gain of the inverters can result in a SNR loss, but not too much.

In this case the digital filter has to be a third-order one because now the quantization noise increases with frequency at a rate of 40dB/decade instead of 20dB/decade, and the decimation is now one sample every 64 cycles.

With this circuit we get more than 10 effective bits of resolution for a 22kHz bandwidth.

4 Digital filters

Most of the logic in a $\Sigma\Delta$ ADC is found in its low-pass filter. The modulator, in contrast, only requires one flip-flop. There are basically two types of digital filters: Finite Impulse Response, FIR, or Infinite Impulse Response, IIR, and in both cases their more general form is hardware expensive as there are multiplications involved. A FIR filter only has zeroes in its transfer function (not counting poles at z=0) while the IIR filter in addition to zeroes, also has poles.

We want a hardware efficient filter without multiplications, and this kind of filter exist, it is called “Comb filter” or also “moving Average”. A first order comb filter is simply the addition of the last N samples. We can also multiply the resulting value by 1/N if we want to have a truly average, but this isn’t strictly needed: the frequency response is the same but the low frequency gain would be N instead of 1. The filter calculation is:

$$y_i = \sum_{j=0}^{N} x_{i-j} \quad \frac{Y}{X} = (1 + z^{-1} + z^{-2} + ... + z^N)$$

This is the equation of a FIR filter where all its coefficients are 1.

But there is another way to compute this filter: For each new sample add the input to the output and subtract the input N samples before:

$$y_i = y_{i-1} + x_i - x_{i-N} \quad \frac{Y}{X} = \frac{1 - z^{-N}}{1 - z^{-1}}$$

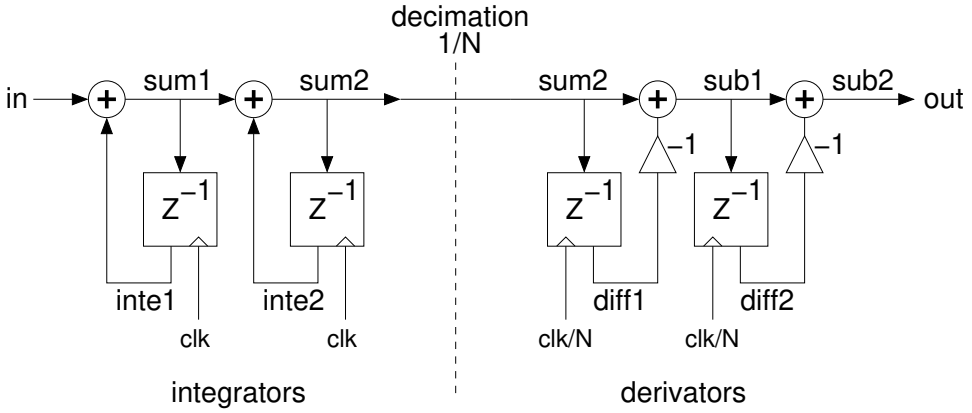
So, here we have the equation of an IIR filter, because it has a pole for $z = 1$. But there is also a zero for $z = 1$, so, the pole and that zero cancels and we get the FIR filter again. Anyway, we can cascade L of these filters, resulting in the general transfer function for a comb filter of order L:

$$\frac{Y}{X} = \frac{1}{(1 - z^{-1})^L} \cdot (1 - z^{-N})^L$$

Notice that this function has two parts: first a cascade of L integrators and then a cascade of L derivator-like functions $(1 - z^{-N})$.

Then, after the filtering comes the decimation, that is just to discard N-1 samples every N. This divides the sampling rate by N. And here comes a trick: the decimation can be done before the last part of the filtering, and if the sampling rate is divided by N this is equivalent to change z^{-N} by z^{-1} in the last part of the filter, so, this part is really made with derivators, but running at the divided sampling rate.

Let me show the resulting block diagram for a second-order comb decimator-filter :



Looking at this diagram we can see the filter requires 4 adders and 4 registers (and maybe a 5th register for holding the output if we want to have a valid data during the N cycles an output sample takes). And now the question is how many bits these registers should have. Well, if N is a power of 2 ($N = 2^m$) we know N additions will result in a value with m more bits than the data, so we can write:

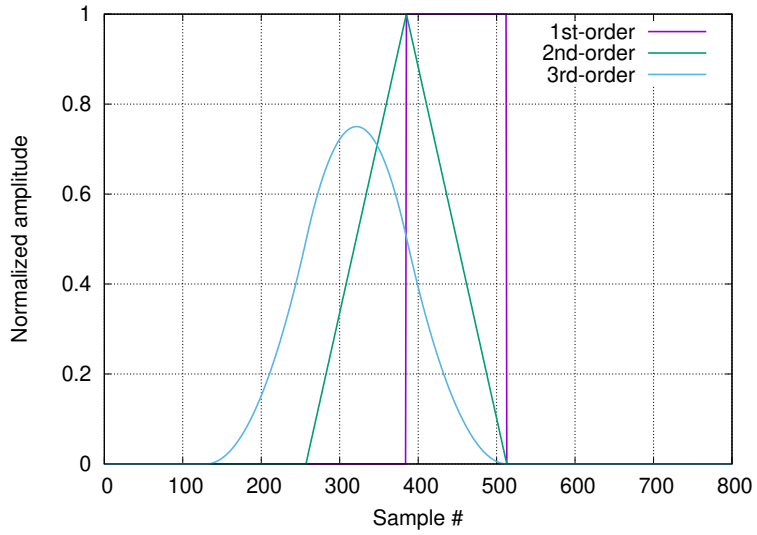
$$nbit = nbit_{input} + L * m$$

In our first-order sigma-delta ADC the input sample has 1 bit, L is 2, and N is 128, so we get $nbit = 1 + 2 * 7 = 15$ bits, and consequently our registers and adders must have this data width in order to avoid overflows.

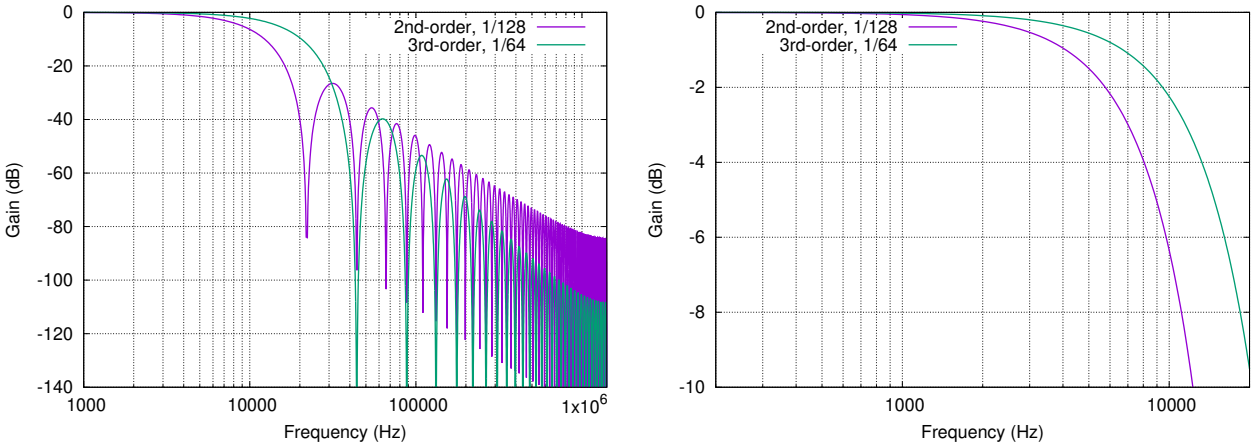
For the second-order ADC L is 3, meaning we have 3 integrator and 3 derivators, totaling 6 registers and 6 adders, but N is 64. this gives: $nbit = 1 + 3 * 6 = 19$ bits.

Also notice that integrator registers actually overflows, but this isn't a problem because the difference between values after N samples is going to be correct, even if an overflow happens, if the width of the variables is right.

Lets present finally the performance of such filters, I mean without decimation, starting with their impulse response in the time domain:



In this example the filters were averaging 128 samples. As we can see the first-order moving average has a simple rectangular impulse response and the second-order a triangular one. The Fourier transform of these pulses gives us the response in the frequency domain:



Here the gain of the filters is plotted as a function of the frequency and we must remark two things: first, outside the pass-band the gain has a peak envelope that decreases at 40dB/decade for the second-order filter or at 60dB/decade for the third-order one. And second, the gain has notches at frequencies that are multiples of the final sampling rate. These notches are very convenient for decimation because they are located just in the middle of the alias frequency bands, and they are also responsible for the "Comb" name because this shape actually looks like a comb.

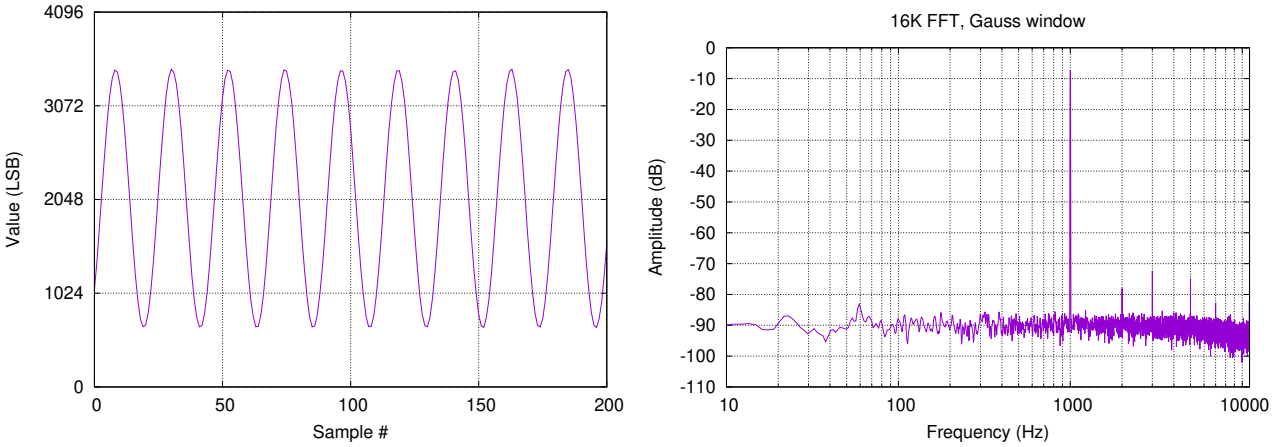
A detail of the pass-band frequency range is also displayed on the right plot where we can see the attenuation the filters have for those frequencies. I don't think this attenuation near the Nyquist frequency is a big problem, but commercial ADCs can include an additional FIR filter for the equalization of this frequency band.

5 Experimental results

The proposed converters, both ADCs or DACs, have the same word width, 12 bits, that in all cases is more than the expected effective resolution. Thus, 1 LSB is 1/4096 of the full signal range, or 0.8mV. The performance of ADCs was measured by means of digitizing a good quality analog sine wave, dumping the samples to a PC through a fast serial port, and analyzing the data. Some results are presented next:

5.1 First-order ADC

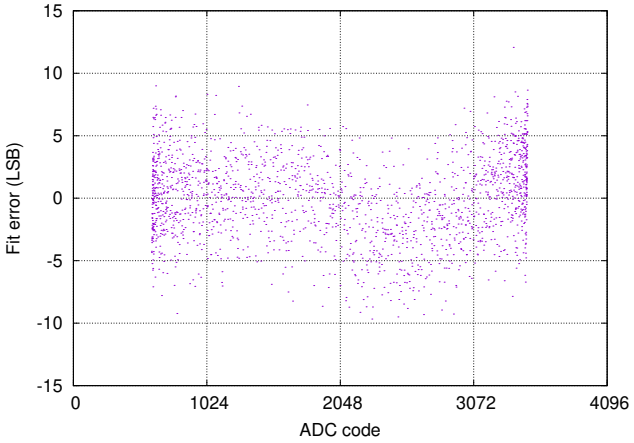
LVDS version



In the above graphs the data digitized using the first-order ΣΔ ADC with a LVDS comparator is presented along its power spectrum. The Input signal was a $1kHz$ sine wave being played from a WAV file using an MP3 USB-stick. The graphs show a low DC offset and little distortion. A deeper analysis of this data gives the following figures:

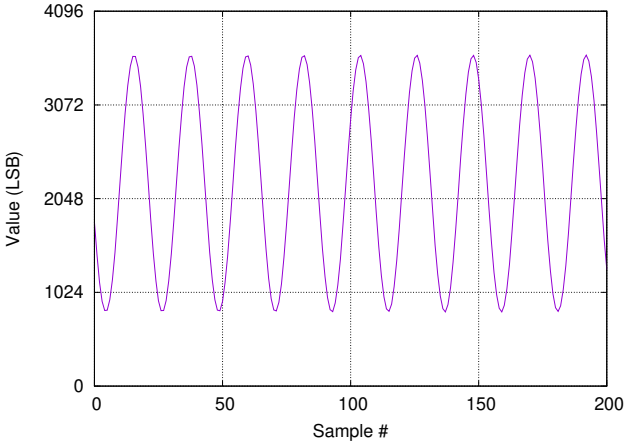
Parameter	Value
DC offset	-8.1mV
	-10.1 LSB
Amplitude	70% of full scale
	577 mV
SNR	50.4 dB
ENOB	8.08 bits
Distortion	-56.4 dBc
	4.3 LSB

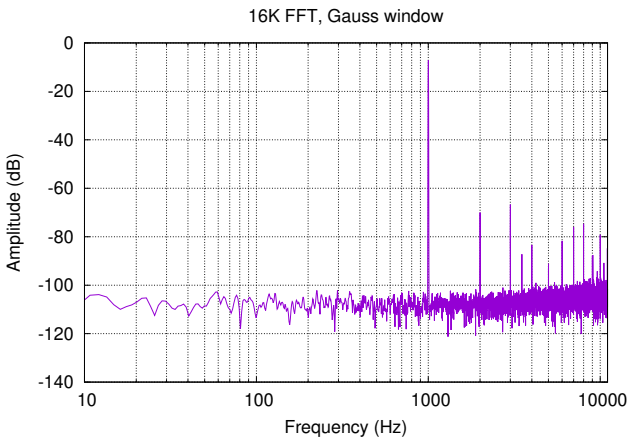
Also, the best curve fit to a sine wave was subtracted from the ADC samples and the resulting error is plotted in the following graph:



This error band is about 15 LSBs wide, meaning we are losing about 4 bits of effective resolution, and it is also curved, meaning there is also some distortion.

CMOS gate version

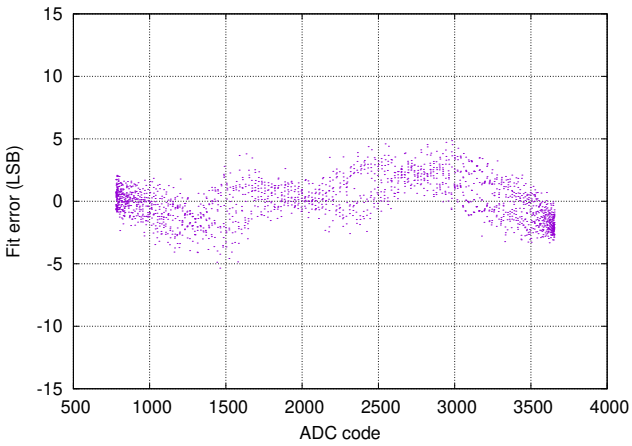




In the above graphs the data digitized using the first-order ΣΔ ADC with the external 74HC00 NAND gate as a comparator is presented along its power spectrum. Here a DC offset is easy to see while in the spectrum plot harmonics looks higher than before, but in fact they are revealing a lower noise floor. The analysis of this data gives the following figures:

Parameter	Value
DC offset	134 mV
	166 LSB
Amplitude	68.6% of full scale
	513 mV
SNR	62.4 dB
ENOB	10.07 bits
Distortion	-56.5 dBc
	4.2 LSB

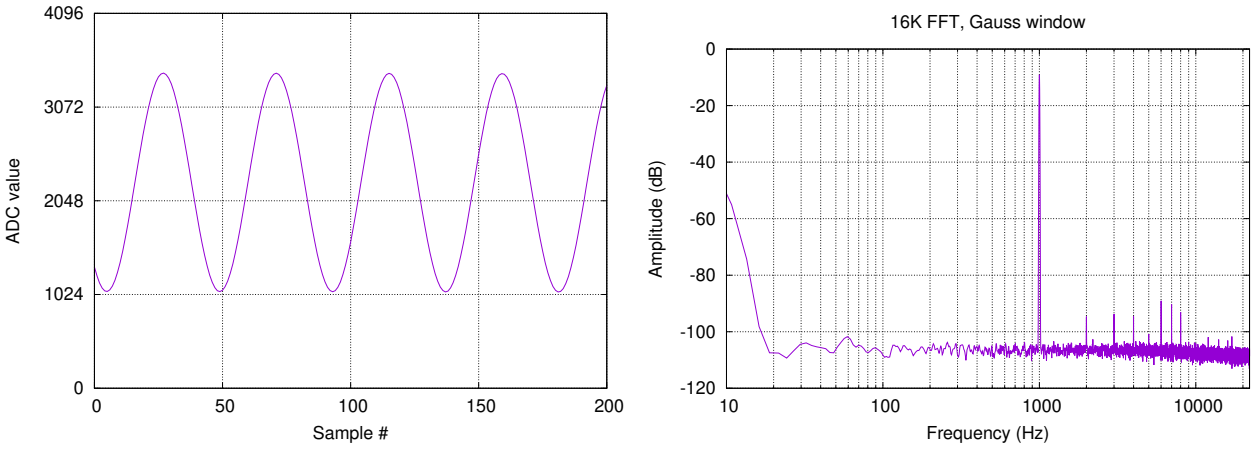
Thanks to the lower noise floor we get two more bits of effective resolution than in the LVDS ADC. In order to display the distortion the best curve fit to a sine wave was subtracted from the ADC samples and the resulting error is plotted in the following graph:



As we can see the error band is narrower than in the LVDS converter, according with the increased resolution, and it is also curved due to distortion.

It is interesting to see how the ADC with the external gate achieves no less than two extra bits of resolution than the one with the internal LVDS comparator, almost the theoretical maximum, and the only explanation I can find is the probably noisy 3.3V supply inside the FPGA.

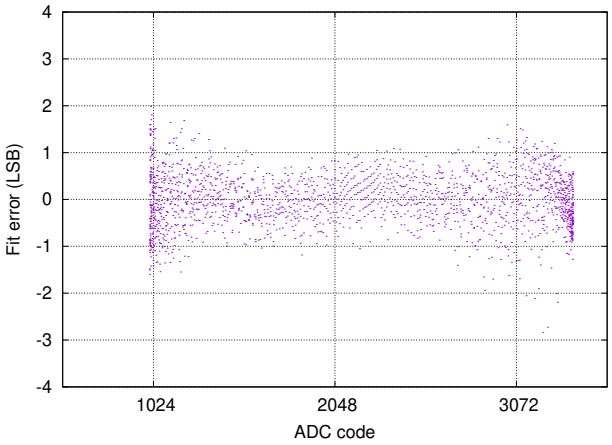
5.2 Second-order ADC



In the above graphs the data digitized using the second-order $\Sigma\Delta$ ADC is presented along its power spectrum for the same $1kHz$ input sine wave. The analysis of this data gives the following figures:

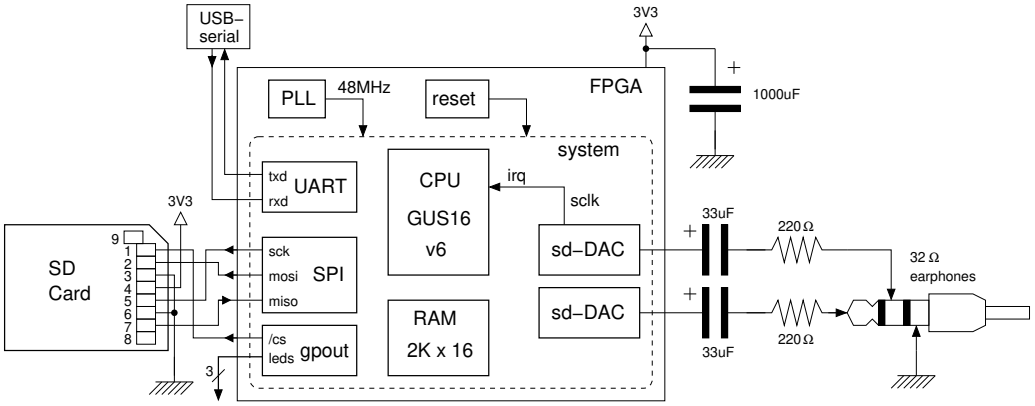
Parameter	Value
DC offset	119 mV
	148 LSB
Amplitude	58.3% of full scale
	0.48 V
SNR	63.45 dB
ENOB	10.25 bits
Distortion	-72.94 dBc
	0.54 LSB

The distortion is quite low, but anyway, in order to display it the best curve fit to a sine wave was subtracted from the ADC samples and the resulting error is plotted in the following graph:

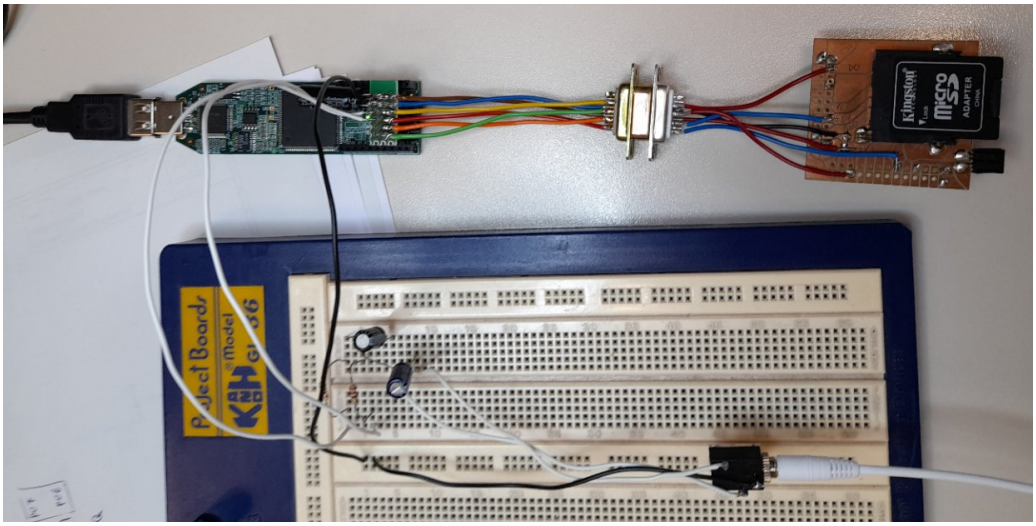


As we can see, the error band lies over a straight line meaning no visible distortion nor idle tones, and the error magnitude is what we can expect from the ENOB figure. (ENOB=12 would mean an error bounded to $\pm 0.5 LSB$)

5.3 WAV player (DAC)



In order to test the design of the ΣΔ DAC the WAV player whose diagram is shown above was designed and built. This system includes an small computer with several interfaces, including two DACs for stereo sound, an SPI controller for the attachment of an SD card, and a UART for controlling it. The CPU core is a 16-bit processor of my own design. There are two tasks in the firmware: an interrupt driven audio playback with a double buffer, and a file selection and reading thread that feeds the buffer.



The resulting system was small enough to fit into an ICESTICK board. In the previous snapshot that board is shown along the crappy connections to the SD card and earphones. Notice the earphones are connected to the FPGA pins almost directly, with a series capacitor for DC removal and a resistor for volume attenuation. There is no filtering at all, but an implicit low-pass filter is already in place at the listener ears, and also the frequency response of the earphones will drop when moving into ultrasonic frequencies.

In the preceding terminal captures we can see a directory listing during WAV file selection, and then the playing screen.

There was no trace of quantization noise on the earphones (hiss background), but other noises were clearly audible, all of them with an electrical origin. With that kind of DAC any ripple in the 3.3V rail will also be present in the earphones, and I tried to attenuate it by means of a big decoupling capacitor that removed most of the noises. But some small artifacts still remain, and they seem to be related with SD card activity.

6 Conclusions

Three ADCs were built and analyzed. A DAC was also built but its performance is awaiting measurement. Let me summarize the ADCs performance in the following table:

	ICs	Res	Cap	Logic cells	Bandwidth	ENOB	DC offset	Distortion
1st-order, LVDS	0	4	1	160	11 kHz	8 bits	good	fair
1st-order, HC00	1	2	1		11 kHz	10 bits	poor	fair
2nd-order, HCU04	1	4	2	289	22 kHz	10 bits	poor	good

Notice that, while for audio applications the DC offset and gain errors are irrelevant, that couldn't be the case if these ADCs are used for other applications. In these cases a calibration would be needed in order to have accurate measurements.

7 Sources

The Verilog sources for the $\Sigma\Delta$ ADCs, and also for a DAC are listed next. The DAC module only includes the modulator since low-pass filtering has to be done in the analog domain. The ADCs modules have one flip-flop for the quantizer of the $\Sigma\Delta$ modulator and the appropriate comb filter for the converter.

7.1 First-order sigma-delta ADC

```
/*-----------------------------------------------------------------------
Digital filter for a 1st-order Sigma-Delta modulator

J. Arias (2023) Public domain sources

This is a 2nd-order Comb filter with a 1/128 decimation factor
that is equivalent to a cascade of 2 moving averages of 128 samples each
followed by a sample output every 128
  input is 1-bit (unsigned)
  output is 12-bit (unsigned)
  Internal variables must have a minimum number of bits:
     Nbit = Nbit_input + Filter_Order * log2(decimation)
     Nbit = 1 + 2 * 7 = 15 bits

-----------------------------------------------------------------------*/
module sd_ADC_1st(
    input clk,          // 128 x Fs clock (falling edge)
    input sdin,         // input from analog comparator
    output fb,          // Feedback to integrator
    output newsample,   // New output sample next clock pulse
    output [11:0]out    // Output data
);

reg fb=0;
reg [14:0]inte1=0;      // Integrator registers
reg [14:0]inte2=0;
reg [14:0]diff1=0;      // differenciator registers
reg [14:0]diff2=0;
wire [14:0]add1 = (~fb) + inte1;    // Integrator adders
wire [14:0]add2 = add1 + inte2;
wire [14:0]sub1 = add2 - diff1;     // Differenciator subs
wire [14:0]sub2 = sub1 - diff2;

reg [6:0]deci=0;        // Decimator counter
wire newsample = &deci; // new sample after maximal count

reg [11:0]out=0;        // Output register (12 bits, but only 10.4 effective bits)

always @(negedge clk) begin
    fb <= ~sdin;        // Negative feedback loop
    deci <= deci + 1;   // decimator counter
    inte1 <= add1;      // Integrators
    inte2 <= add2;
    if (newsample) begin    // every 128 cycles
        diff1 <= add2;  // Differenciators
        diff2 <= sub1;
        // Output with saturation
        out <= sub2[14] ? 12'hfff : sub2[13:2];
    end
end

endmodule
```

7.2 Second-order sigma-delta ADC

```
/*-------------------------------------------------------------------------
Digital filter for a 2nd-order Sigma-Delta modulator

J. Arias (2023) Public domain sources

This is a 3rd-order Comb filter with a 1/64 decimation factor
that is equivalent to a cascade of 3 moving averages of 64 samples each
followed by a sample output every 64
  input is 1-bit (unsigned)
  output is 12-bit (unsigned)
  Internal variables must have a minimum number of bits:
     Nbit = Nbit_input + Filter_Order * log2(decimation)
     Nbit = 1 + 3 * 6 = 19 bits

-------------------------------------------------------------------------*/
module sd_ADC_2nd(
    input clk,          // 64 x Fs clock (falling edge)
    input sdin,         // input from analog comparator
    output fb,          // Feedback to integrator
    output newsample,   // New output sample next clock pulse
    output [11:0]out    // Output data
);

reg fb=0;
reg [18:0]inte1=0;      // Integrator registers
reg [18:0]inte2=0;
reg [18:0]inte3=0;
reg [18:0]diff1=0;      // differenciator registers
reg [18:0]diff2=0;
reg [18:0]diff3=0;
wire [18:0]add1 = (~fb) + inte1;    // Integrator adders
wire [18:0]add2 = add1 + inte2;
wire [18:0]add3 = add2 + inte3;

wire [18:0]sub1 = add3 - diff1;     // Differenciator subs
wire [18:0]sub2 = sub1 - diff2;
wire [18:0]sub3 = sub2 - diff3;

reg [5:0]deci=0;        // Decimator counter
wire newsample = &deci; // new sample after maximal count

reg [11:0]out=0;        // Output register (12 bits, but only 10.4 effective bits)

always @(negedge clk) begin
    fb <= ~sdin;        // Negative feedback loop
    deci <= deci + 1;   // decimator counter
    inte1 <= add1;      // Integrators
    inte2 <= add2;
    inte3 <= add3;
    if (newsample) begin    // every 64 cycles
        diff1 <= add3;  // Differenciators
        diff2 <= sub1;
        diff3 <= sub2;
        // Output with saturation
        out <= sub3[18] ? 12'hfff : sub3[17:6];
    end
end

endmodule
```

7.3 Second-order sigma-delta DAC

```
//-------------------------------------------------------------------------
//-- 2nd-order Sigma-Delta DAC (error-feedback topology)
//   J. Arias (2023) Public domain source
//-------------------------------------------------------------------------

module sd_DAC(
    input clk,          // around 3MHz, (2.82MHz for 44100/2^n)
    input [NBIT-1:0]in, // signed value
    output out,         // output bitstream
    output sclk         // sampling clock (clk/OSR)
);

parameter NBIT=12;      // default input sample resolution
parameter OSR=64;       // default oversampling Ratio
localparam NDIV=$clog2(OSR);

// Clock divider for sampling rate generation
reg [NDIV-1:0]sdiv=0;
always @(posedge clk) sdiv<= (sdiv==(OSR-1)) ? 0 : sdiv+1;
assign sclk=sdiv[NDIV-1];

// sampled input
reg [NBIT-1:0]inreg;
always @(negedge sclk) inreg<=in;
// The SD modulator input, x, requires sign extension
wire [NBIT+2:0]x={{3{inreg[NBIT-1]}},inreg};

//////////////////// SD MODULATOR (DAC) ////////////////////////
// 3 more bits for variables in order to avoid overflows
wire [NBIT+2:0]err;     // Quantization error
reg  [NBIT+2:0]e1=0;    // error 1 cycle before
reg  [NBIT+2:0]e2=0;    // error 2 cycle before
wire [NBIT+2:0]y;       // filter output
wire outp;              // output bit
reg out=0;              // registerred output (to avoid glitches)

assign y=x+{e1[NBIT+1:0],1'b0}+(~e2)+1'b1;  // y=x+2*e1-e2
assign outp=y[NBIT+2];                      // output = sign bit

localparam FULLSC=19'h3<<(NBIT-2);          // Full scale at +-150%
assign err=y+(outp ? -FULLSC : FULLSC);     // Quantization error

always @(posedge clk) begin     // Register update
    e1<=err;
    e2<=e1;
    out<=outp;
end

endmodule
```