

Dear Paula

1 Introduction

I've ignored my trusty Amiga for far too long. Now with the rise of the Vampire (no pun intended), I got hooked up to Amiga tinkering again. I've got a separate sound card in my A4000 (Melody MPEG) and consequently ignored the native Amiga chipset from there on. Recently, I read into a number of discussions regarding Paula's shortcomings and stumbled over the issue of aliasing. Aliasing on the DAC side of a sound chipset? Ridiculous, right? All you need is some sort of DAC plus some proper analog reconstruction filter and you are all set.

But here is the culprit: Either you apply the anti-aliasing filter (Commodore term) and cut away all frequencies beyond 7 kHz for all intents and purposes or you can switch it off and have to provide sufficiently well band-limited input signals. The Amiga features a second lowpass filter (actually first in the chain of most models), based on a simple RC circuit that will not fulfill the necessary task of proper analog signal reconstruction with a decent frequency response in mind.

I wrote down this document to get a better grasp on approaches to get the best possible sound out of the Amiga chipset, in an OS friendly way. I don't go down the route of manual Paula register banging.

One major milestone in the endeavor towards higher audio quality was the rise of 14 Bit sound in the mid 90's, culminating in the 14 Bit calibration approach by Christian Buchner [1]. The Amiga features four separate 8 Bit audio channels, where two of the channels are panned left and the other two right. The audio channels may be assigned a volume level between 0 and 64. When two of the channels operate at maximum volume and the other two at minimum volume, then you obtain a total dynamic range of $8+6=14$ Bit. Sadly, the DACs usually exhibit some distortions (non linearities) of their own. Therefore, the actual number of steps at minimum volume usually depends on the amplitude of the high volume channel.

Of course, there were previous approaches to analyze Paula's behavior, where the most notable (to my knowledge) were published by Antti S. Lankila [2] and Toni Wilen [3]¹. In both cases, the main intent was to reproduce the shortcomings of the Amiga chipset as close as possible in an emulated environment. Nevertheless, I consider their work as highly valuable input towards a better understanding of how to improve the sound reproduction, this time the other way around.

The rest of the document in a nutshell is as follows: Confirmation of DAC artifacts, along with propositions to avoid distortions in sound output with reasonable computational requirements, amended by some thoughts towards stereo calibration.

Some warning is necessary, though. What I propose to apply will be targeted at a sound reproduction with a more or less flat frequency response, which in turn will sound totally different from what one is used to when listening to Amiga music. If you are fond of the Amiga sound and don't want to have it any other way, then this stuff is not for you.

¹There is quite a number of interesting threads on EAB, like [4].

Contents

1	Introduction	1
2	Signal Theory	3
2.1	Parameters	3
2.2	DAC in impulse mode	5
2.3	DMA to DAC	6
2.4	Volume Control	9
2.5	Lowpass Filters	10
3	Signal processing considerations	11
3.1	Band limiting	11
3.2	Lowpass compensation	12
4	14 Bit Mode and Calibration	16
4.1	14 Bit Calibration	16
4.2	Revised 14 Bit Calibration	17
5	Measurements	18
6	Conclusion	21
A	Test Code	22
A.1	Intermodulation test	22
A.2	SNR Calculation in Matlab	24

2 Signal Theory

2.1 Parameters

The system timing of the Amigas was different, depending on the region where they were sold. The American models were using the NTSC video system, while the European models were customized for PAL. So what? What does the video system have to do with audio output? In conjunction with the Amiga chipset: everything. First of all, the master system clock is 14.31818 MHz for NTSC Amigas and 14.18758 MHz for PAL Amigas. All other clocks are derived from that, including the 68000 CPU clock². So one of the answers is the base clock.

Another matter is the fact that audio DMA slots are tied to the scan lines. The Amiga chipset provided a rate of two samples per channel per scan line. A rough estimate yields (example from HRM):

$$2 \text{ samples/line} \cdot 262.5 \text{ lines/frame} \cdot 59.94 \text{ frames/second} = 31469 \text{ samples/second}$$

The practical maximum with 15 kHz scan line frequency is actually 28867 Hz for Amigas with OCS chipset³. The ECS chipset⁴, introduced with the Amiga 3000 allows for double scan VGA compatible screen modes. In these double scan screen modes, twice the number of scan lines are shown and hence, twice the number of samples can be written to the DMA. Most more capable Amigas these days either feature a graphics card or at least make use of the ECS/AGA screen modes for more real estate on the monitor. The same applies for UAE, which includes the double scan considerations in the emulation core⁵.

The DAC base clock is the master clock divided by four, 3579545 ticks per second for NTSC and 3546895 ticks per second for PAL. This clock plays an important role in the understanding of the available Amiga sampling rates and of course, the purportedly present aliasing problem. Sampling rates are specified indirectly, by the period audio registers. The period is calculated as follows:

$$\text{period} = \frac{\text{clock constant}}{\text{samples per second}}$$

The minimum period for NTSC systems is 124 and for PAL systems 123, respectively in single-scan screen modes⁶. The period is actually the number of ticks at DAC base clock between each sample.

Each channel has its own volume registers. The volume levels 0...63 will force the DAC into a periodic approximation mode in order to achieve the respective relative signal amplitude. In this mode, the DAC works in a raster of pulse sequences being 64 ticks or $64 \cdot 1/3546895$ seconds apart. The maximum volume level 64 disables the fixed DAC sampling raster. With volume 64, the actual desired sampling rate is maintained without further resampling. Please note that I will concentrate on PAL timings further on, although the considerations apply to NTSC Amigas as well.

Wait a bit. Why did I write 64 ticks instead of *period* ticks? Well, here is another catch. There is a difference between the *period*, which controls the DMA rate and the actual DAC output

²European users got 7.09 MHz, while the Americans had 7.14 MHz CPU clock

³Actually, the practical constraints are different for the four channels, but in these considerations we need all four in sync.

⁴Super Denise 8373, coupled with Agnus 8372 or later

⁵Setting from Amiga Shell with Picasso96, both on native Amigas and UAE: `SetEnv EnvArc:Picasso96/AmigaVideo 31kHz`

⁶DMA slot assignment yields different limits per channel, which is not discussed further here.

speed. Essentially, the DAC performs an internal resampling of the digital input signals to its own 64 tick clock period (55420 Hz PAL) when the volume setting is below 64. This resampling in itself is already prone to some intermodulation distortions towards the spectrum. As the internal schematics of Paula are still not yet publicly available, one must conclude from signal observations what actually happens. One test that indirectly confirms my observations is shown in the appendix (Sec. A.1). That test just outputs a square wave on two channels, one at full volume and the other at volume 63. The latter channel exhibits the expected intermodulation distortions. In any case, the discussion in [4] also points out the implicit resampling at lower volume settings.

Furthermore, it must be noted that the resampling from DMA rate (period) to the DAC rate is not performing filtering of any kind. Hence, the DMA rate dictates the prominent spectral repetitions and hence, audible spectral distortions. In software applications, nearest-neighbor upsampling in audio and images implicitly repeats samples as needed. The mentioned repetition is already a kind of *anti-imaging* filter, actually the worst performing kind you can implement. Depending on the ratio between DMA rate and DAC rate, the sample-and-hold approach will in normal cases reproduce spectral repetitions that will carry over to the analog domain.

Since the Paula DAC outputs impulses as differences between adjacent samples, the nearest-neighbor analogy from the previous sentences does not hold. Instead, Paula acts like an ideal upsampler. Ideal upsampling keeps the spectral repetitions of the lower frequency digital signal and just features a higher output sampling rate. This is exactly the reason why Paula creates artifacts in the audible part of the spectrum. When looking closely at UADE's Amiga sound emulation, you can actually see that it is emulated exactly as I stated [2]. I'll get back to this topic in Sec. 2.3.

The bottom line at this point is that the optimal sampling rates for mixed sound output on PAL Amigas are 55420 Hz ($=period\ 64$) with double scan screen modes and 27710 Hz ($=period\ 128$) otherwise⁷. The mentioned frequencies avoid additional distortions as they map the DMA rate 1:1 (or 1:2, respectively) to the DAC rate⁸. In 14 Bit playback, this behaviour applies to the low volume channel only, which attenuates the modulation effects by a fair margin (IMHO).

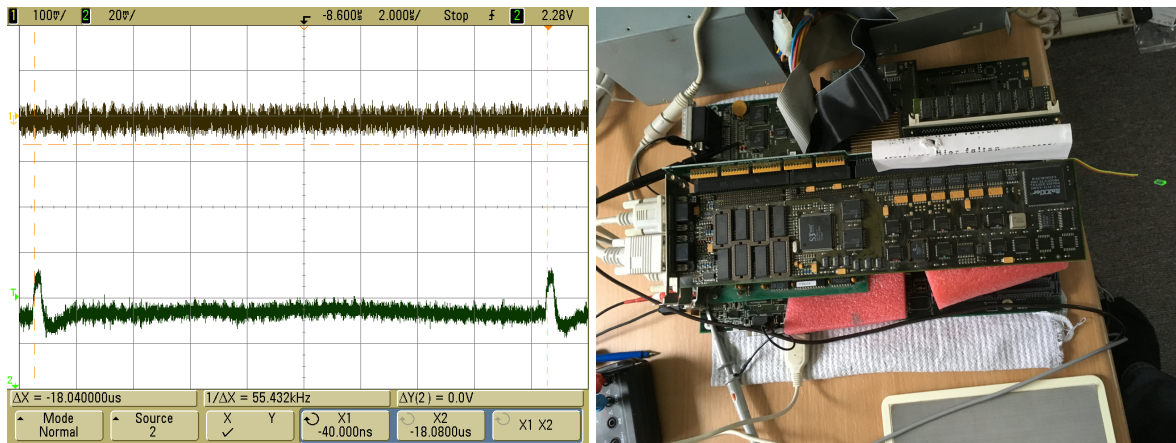


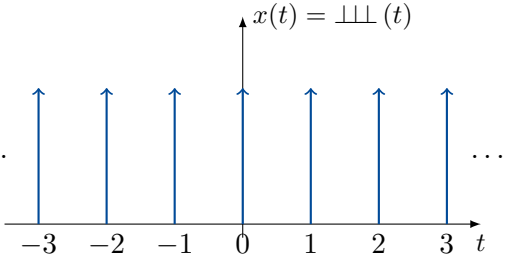
Fig. 1: Left: Oscilloscope plot depicting the DAC rate of 55420 Hz (Amiga4000, 44336 Hz DMA rate, 14 Bit mode), Right: device under test: A4000/CS060MK-I/CV64 after recapping, with connected probes

⁷NTSC: double scan 55930.4 Hz, single scan 27965.2 Hz

⁸The jitter depends on the input signal and the ratio between DMA and DAC rates. Since the hardware allows to set DMA rates equal or at least close to the DAC rate, a viable workaround is available. I'll skip the tedious math.

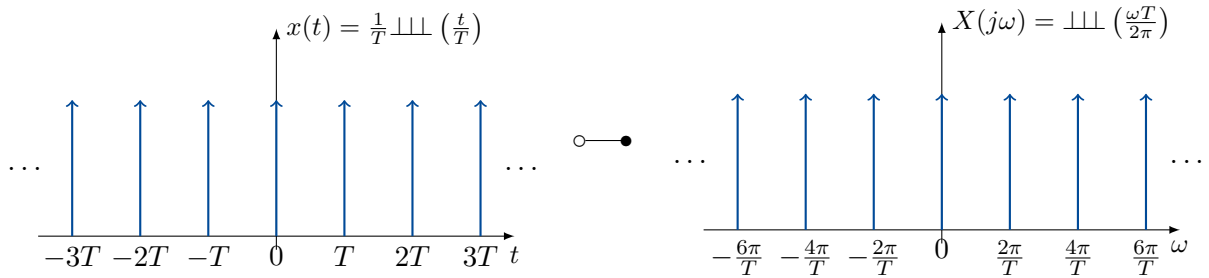
2.2 DAC in impulse mode

With a fixed volume level of 64, the DAC will output impulse trains with a frequency of 55.4 kHz and a pulse width of $1/3546895 \text{ s} = 0.279 \mu\text{s}$. For the first part, let's ignore that the impulses have a width and assume infinitely short pulses. We come back to the actual pulse width in a later step by straightforward relationships. Let's first consider an impulse train at fixed time instances $\dots, -2, -1, 0, 1, 2, \dots$.

$$x(t) = \text{III}(t) = \sum_{\mu=-\infty}^{\infty} \delta(t - \mu) \quad , \quad \mu \in \mathbb{Z} \quad (1) \quad \dots$$


The simplified symbol for a delta or Dirac impulse train is $\text{III}(t)$. With the correspondence $\delta(t - \mu) \circ \bullet e^{-j\mu\omega}$ follows for the impulse train in frequency domain $X(j\omega) = \sum_{\mu=-\infty}^{\infty} e^{-j\mu\omega}$, which in turn is an impulse train $X(j\omega) = \text{III}(\omega/2\pi)$ again [5, 6, 7]. Now the application of the time-scaling rule gives the following relationship:

$$\frac{1}{T} \text{III}\left(\frac{t}{T}\right) \circ \bullet \text{III}\left(\frac{\omega T}{2\pi}\right) \quad (2)$$



In case of good old Paula, $T = 64/3579545$. The repetitions of the impulses in the spectrum are 55 kHz apart. This is actually a good property. If (hypothetically) Paula was generating pulses at the frequency dictated by *period*, the level would have to be scaled along with the changing rate to provide the necessary energy for a constant audio output level, regardless of the desired sampling rate. In addition, there is the catch of the two channels per panning (right/left). So this resampling to 55 kHz actually makes sense, since it simplifies a lot of secondary considerations. For typical low frequency and/or relatively well band-limited digital signals, it won't make much of a difference to the casual listener.

Next, let's include the width of the impulses. Each impulse is a rectangle of $T_r = 1/3546895 \text{ s} = 0.279 \mu\text{s}$. We obtain the idealized DAC output signal by positioning a rectangle around each pulse. This is easily accomplished by convolving the impulse train with a rectangular pulse:

$$x_{DAC}(t) = \frac{1}{T} \text{III}\left(\frac{t}{T}\right) * \text{rect}\left(\frac{t}{T_r}\right)$$

When we want to observe the spectral properties $X_{DAC}(j\omega)$ of the convoluted signal $x_{DAC}(t)$, we just need to multiply the spectra of the two signals which were convoluted in time domain:

$$X_{DAC}(j\omega) = \text{III}\left(\frac{\omega T}{2\pi}\right) \cdot T_r \text{si}\left(\frac{T_r \omega}{2}\right)$$

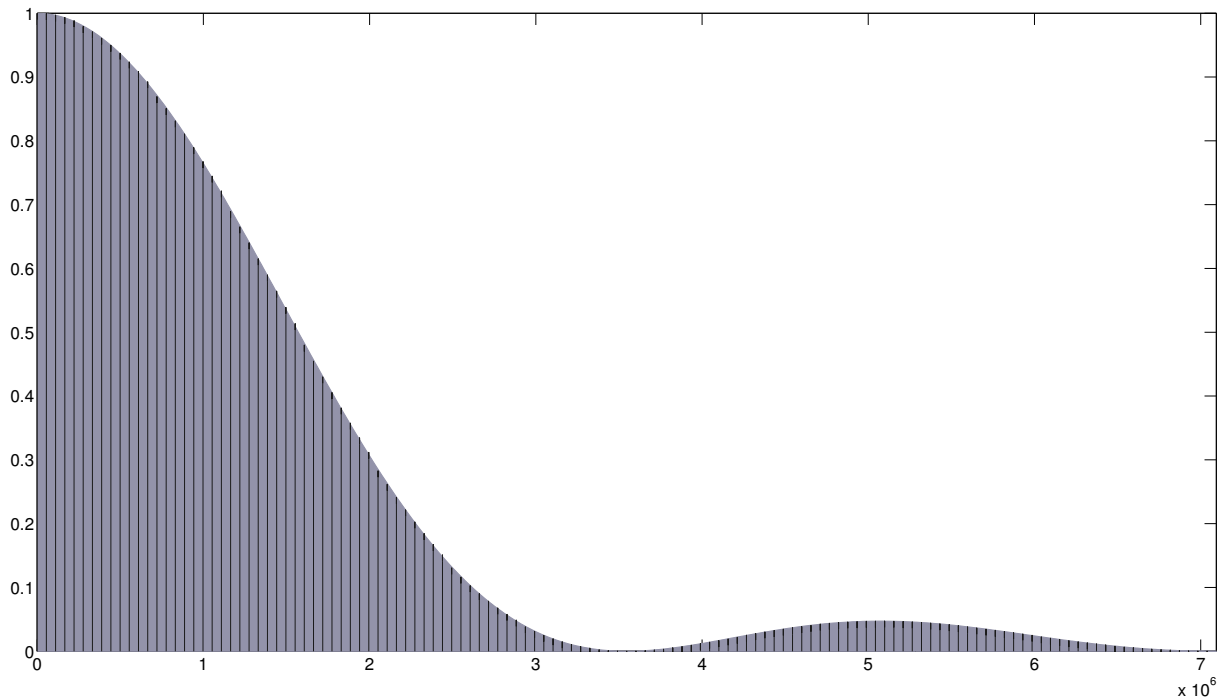


Fig. 2: Power density spectrum of Paula pulse output (before low-pass filters), normalized to 1 and shown in linear scale. The envelope due to the pulse width is drawn solid in the background and the spectral repetitions of the DAC frequency are shown as lines. The spectrum is shown from 0 to 7 MHz.

What does this mean when we look at the practical parameters? We do have a time interval T , which is 64 times the time interval T_r . In other words, the first zero crossing of the rectangular pulse is in the spectrum at 3.5 MHz. Hence, we have 64 repetitions of the pulse train in frequency domain before the pulse shape yields a zero-contribution to the spectrum for the first time. This is visualized in Fig. 2.

One can conclude at this point that the big difference between the 55 kHz impulse train frequency and the 3.6 MHz pulse shape is suitable for mathematical simplifications. The first few repetitions in the spectrum are almost at the same level so that they can be treated as near constant in amplitude. In addition, considerations toward output of square wave signals can also be conveniently handled as ideal rectangles.

Nevertheless, the DAC frequency is not the issue we are looking for (at least when the DMA rate is ≥ 40 kHz).

2.3 DMA to DAC

Now that we have established that the main culprit is not to be found in the DAC frequency, let's take a closer look at the signals coming out of the DAC. You can observe the result of a worst-case experiment in Fig. 3. The input signal is a 2.6 kHz square wave, output with a DMA frequency of 11 kHz and 50 kHz. Apart from an apparent resampling accuracy issue (in software), one can see that the signal coming out of the DAC is identical, otherwise.

What does this mean for the resulting audio signal? Simply put, it doesn't matter how many samples you collect for a square wave. In any case, the spectral repetition issues are bound to the square wave frequency, **not** the sampling frequency. So from my perspective, this is an

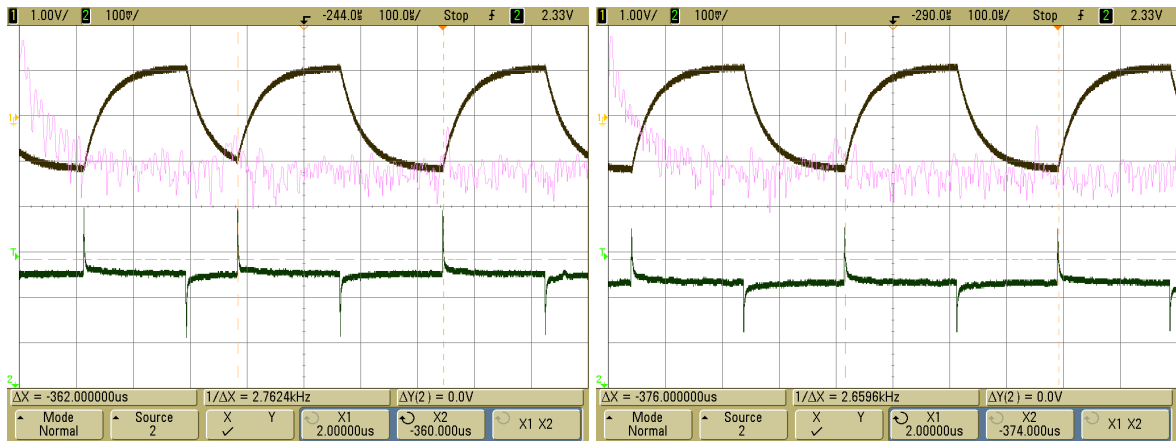


Fig. 3: Oscilloscope plots of 2.6 kHz square waves, at 11 kHz and 50 kHz DMA rate, upper traces are after the RC lowpass filter, lower traces show the DAC output

interesting observation.

Let's put the observations into math. We can re-use the approach from the previous section. We do have an impulse train again, which is then multiplied by the square wave. This time, the effective frequency $1/T$ of the impulse train is exactly the frequency of the square wave times two. The square wave can be constructed again from the convolution between a single rectangular pulse and an impulse train. That signal will be multiplied with an impulse train again to obtain the signal we know from the scope. But then again, it gets even simpler (that's why I left out formulas until now). We can see already in Fig. 3 that we just have a positive impulse train of frequency $1/T$ and a negative impulse train of the same frequency, shifted by $T/2$. So in total we get the output signal $x_{DAC}(t)$ for a square wave of frequency $1/T$:

$$x_{DAC}(t) = \frac{1}{T} \text{III} \left(\frac{t}{T} \right) * \text{rect} \left(\frac{t}{T_r} \right) - \frac{1}{T} \text{III} \left(\frac{t}{T} - \frac{T}{2} \right) * \text{rect} \left(\frac{t}{T_r} \right)$$

As the two parts of the expression are only additive, we can use the same correspondence we had in the previous section, now incorporating a shift in time for the second term by $-T/2$, which maps to a phase shift by $e^{-j\omega \frac{T}{2}}$ in the frequency domain.

$$X_{DAC}(j\omega) = \text{III} \left(\frac{\omega T}{2\pi} \right) \cdot T_r \text{si} \left(\frac{T_r \omega}{2} \right) - \text{III} \left(\frac{\omega T}{2\pi} \right) \cdot T_r \text{si} \left(\frac{T_r \omega}{2} \right) \cdot e^{-j\omega \frac{T}{2}}$$

We know from the previous example that the impulse train in frequency domain decays way more slowly than wished for. The relevant constant T_r is present here in the $\text{si}()$ term as well. For all intents and purposes in the frequency ranges relevant to man, there is almost no amplitude or signal power decay in the spectral repetitions⁹.

The effect on the spectrum is shown in Fig. 6. Please note that further filtering is not considered yet. What can be seen is the effect that we get higher frequency harmonics, clearly within the audible frequency range. These harmonics are expected for a square wave. With lowpass filtering, subsequent higher harmonics will be attenuated, of course. In this particular symmetric case, half of the spectral lines present in both parts of the equation are cancelled out while the other half is amplified by a factor of two (not shown in normalized graph).

Let's move on to another example, this one using the parameters from HRM (3rd edition, page 155). There it is suggested to use a 4 kHz Wave with 12 kHz sampling frequency. Ironically, if

⁹At this point, we are disregarding the 6dB/Oct. RC lowpass that follows in the analog signal processing chain

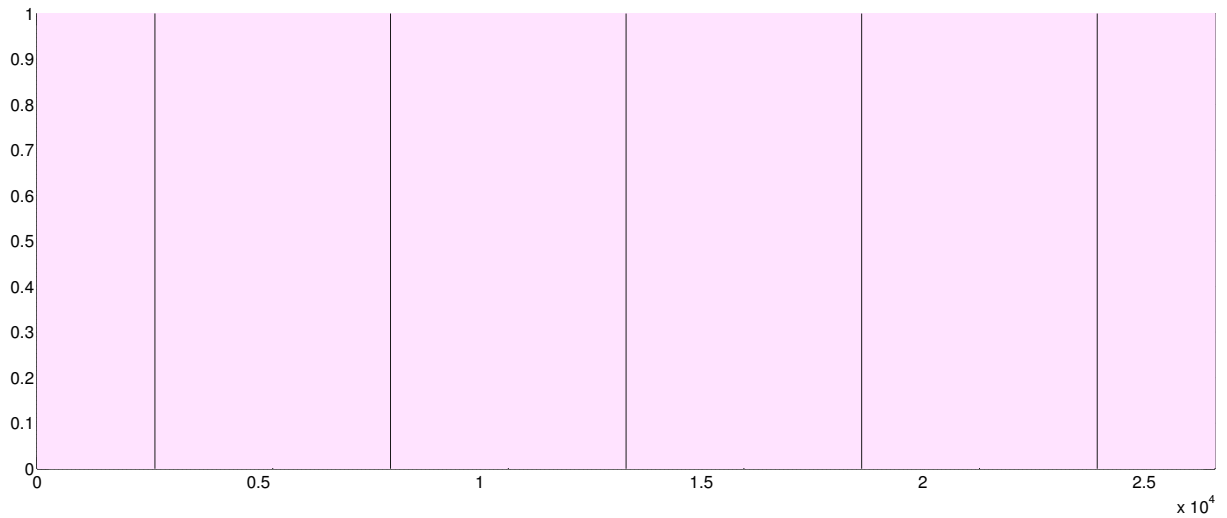


Fig. 4: Simulation of spectral effect of DAC impulse trains for a 2.66 kHz square wave, linear scale, normalized to amplitudes of one

you generate a cosine wave at initial phase 0 with the given parameters, you obtain
 127, -64, -64, 127, -64, -64, ...

Oh. We get a square wave again, but this time the positive and negative impulse trains are separated by $T/3$ in time. The parameter T in this case, corresponds to the impulse train frequency of 4 kHz. We skip the time part and just adjust the phase parameter in the Fourier domain formula that applied to the evenly distributed square wave:

$$X_{DAC}(j\omega) = \text{III}\left(\frac{\omega T}{2\pi}\right) \cdot T_r \text{si}\left(\frac{T_r \omega}{2}\right) - \text{III}\left(\frac{\omega T}{2\pi}\right) \cdot T_r \text{si}\left(\frac{T_r \omega}{2}\right) \cdot e^{-j\omega \frac{T}{3}}$$

The corresponding plot is shown in Fig. 5. When comparing the result to the example in the

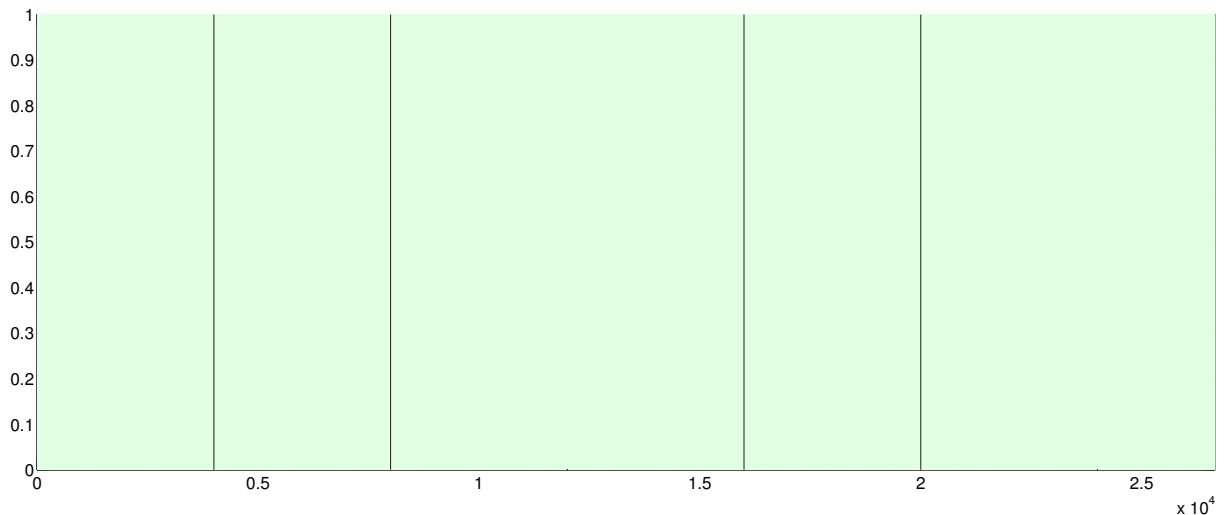


Fig. 5: Simulation of spectral effect of DAC impulse trains for a 4 kHz cosine wave with 12 kHz sampling frequency, linear scale, normalized to amplitudes of one

Amiga Hardware Reference Manual, the figures match. The spectrum repeats with the 12 kHz sampling frequency, so that we get contributions at $12+4$ kHz as well as $12-4$ kHz and on.

Now with the 7 kHz filter assumed to be off, we can immediately see the problem. This 4 kHz wave, played at 12 kHz sampling rate produces additional audible frequency contributions at 8

kHz and 16 kHz. Imagine, we modulate the frequency of the wave downwards to, say 3 kHz. What actually happens is that the first repetition moves up to $12-3=9$ kHz. Well, that effect is of course not what is typically desired in sound reproduction with minimal distortions. Due to the short 3.5 MHz pulses involved in the DAC step, we also observe a particularly slow decay of the spectral repetitions.

The given examples already show the obvious answer: use a sampling rate as high as possible to push the spectral repetitions as far away as one can. It is also necessary to make sure that the output signal is sufficiently well band-limited. For example, if the sampling frequency is 28 kHz and we assume that casual listeners don't hear anything above 17 kHz, the digital signal must be limited to frequencies below 11 kHz. We have seen from the roughly sampled cosine signal that this condition can be achieved.

The bottom line is: If you dislike spectral harmonics in a given sound output, don't offer the corresponding signals (triangles, squares,...) to Paula. She doesn't treat them like you would expect from a more forgiving DAC with steep reconstruction filters. The second point is that one should choose the highest available sampling rate to avoid getting the spectral repetitions into the audible range in case of the Amiga, since there is no adaptive analog filter.

2.4 Volume Control

It could be seen already in Fig. 3 how Paula will work at the full volume level of 64. The differential impulses out of Paula are amplified and filtered by the OpAmp to obtain the respective waveform.

Now, there is the question how to obtain the desired amplitude by volume levels <64 . The DAC voltages are fixed. Classic PWM does not apply as longer pulses would result in even more signal energy going into the OpAmp.

So what Paula does is to change its operation mode for volume 63 and below. It establishes the aforementioned raster of $T = 64/3579545s$ in which it will perform an operation cycle similar to a PWM. Within each cycle, an impulse is generated which will be canceled out by a negative impulse of the same amplitude after a number of time steps corresponding to the volume setting, i.e. $T_v = v/3579545s$ with $v \in [1 \dots 63]$.

These 55.4 kHz cycles are depicted in Fig. 6 for the same waveform at three different volume settings (yellow upper traces). It should be noted that the impulses at full volume have approximately twice the amplitude of the *differential PWM* signals. The main reason is the differential output of impulses at volume 64. This results in differences of $-255/+255$ towards the respective previous sample when a rectangle sequence with amplitudes $+127/-128$ is played.

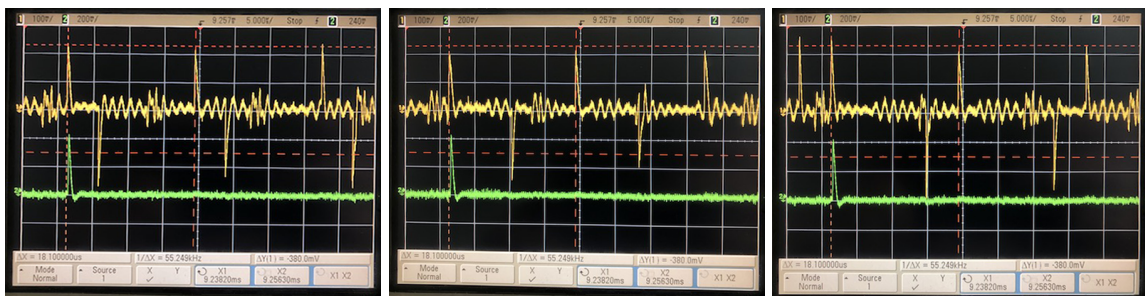


Fig. 6: Oscilloscope plots of 500 Hz square waves at Paula's output pins, volume settings 16,32 and 48 (left to right), lower traces show the same wave at volume 64

2.5 Lowpass Filters

From [2]: After leaving the chip, the sound enters a 6 dB/oct resistor+capacitor (RC) low-pass filter tuned at approximately 5 kHz. I will call this component fixed filter in this document. The purpose of this filter is probably to remove some of that aliasing which is inherent in the pulse synthesis. Additionally, it is possible to engage a low-pass 12 dB/oct Butterworth filter tuned at approximately 3.2 kHz by turning the Amiga power LED brighter with a special protracker command. I will call this component LED filter.

The Butterworth filter removes any high frequencies from the output signal, resulting in a quite muffled sound. I typically disregard (switch off) this filter. This leaves the effect of the RC lowpass.

The approximation in [2] is described as follows:

We should apply the IIR filters to the sinc function before proceeding to build the band-limited step. The 6 dB/oct filter is an IIR filter that is computed according to the following formula:

$$\text{output}(n) = b_0 * \text{input}(n) + (1 - b_0) * \text{output}(n - 1)$$

that is, the output is a linear combination of input and the value of output in the last iteration.

The b_0 determines the cutoff value and is computed as follows:

$$\begin{aligned} \omega &= 2 * \pi * 5000 / \text{sampling_rate} \\ b_0 &= 1 / (1 + 1 / \omega) \end{aligned}$$

As a reminder, the cutoff frequency would be 5.0 kHz and sampling rate is 3.55 MHz because we will operate the blep that is "sampled" at 3.55 MHz. The rest of the problem here is just applying the previous formula to filtering the BLEP.

END OF QUOTE

Now, while most Amigas share the same analog reconstruction filter structure, the choices of resistors and capacitors ¹⁰ differ a little. The fixed filter's designed corner frequencies of some Amiga models are:

- Amiga 500/2000: 4.42 kHz (100 nF, 360 Ω)
- A600: 4.42 kHz (100 nF, 360 Ω)¹¹
- A1200 Rev1d: 27.7 kHz (3.9 nF, 1.5 k Ω)
- A1200 Rev2: 34.4 kHz (6.8 nF, 680 Ω)
- A4000: 4.52 kHz (47 nF, 750 Ω)

Oh. A1200 is very different from the others. Following the filter design of the other Amiga models, I'd expected something around 68 nF for the A1200 Rev2.

But there's more. On A3000, the C430/C440 combo is unused/reserved. Instead, the lowpass is placed directly in front of the RCA jacks. It consists of a 1 k Ω resistor, a 100 nF cap towards GND with 390 Ω parallel to it. That filter is designed for a cutoff frequency around 5.7 kHz, assuming that the A3000 is supposedly connected to a high impedance sink. Effectively, the A3000 is pretty much in line with most other Amigas concerning the frequency response with just about 3 dB less attenuation around 18 kHz than the A4000.

¹⁰A500/600/1200: R331/C331, R321/C321, A4000: R430/C430, R440/C440

¹¹a previous version of this text had wrong parts listed, these here I've checked

3 Signal processing considerations

After establishing the reasons for the Amiga sound output behavior, it's time to move on to the signal processing part. Typically, one takes an appropriate filter design tool these days and lets it do the job. Current hardware is easily fast enough for long filter impulse responses. Even the formerly challenging Fourier domain signal processing has become common in the meantime¹².

But then again, we are talking about the Amiga here. When implementing higher-order audio filters with non power-of-two coefficients, even 68040 and 68060 CPUs will be stressed considerably by the old mantra of costly multiplications. That's why the filters I'm going to propose are at best suboptimal by modern standards.

We know from the observations in Sec. 2.3 that the number of constant samples is irrelevant for the actual reproduction, as long as the total duration of the cycle in time stays the same. This observation and subsequent theoretical analysis prove that nearest-neighbor mixing will keep the harmonics of the Amiga sound intact when played by the Amiga chipset.

Once we start filtering the processed samples, the spectral repetition components of the signals will be strongly attenuated. The result is a sudden loss of perceived medium and high frequency content. These missing medium and high frequencies were only the result of the way the Amiga reproduces sound at low sampling frequencies. Does this now mean that the DAC of the Amiga is bad? It is certainly not high-end. But it can serve one purpose well: synthesize waves with multiple harmonics and variable frequency at very low expense in terms of memory. So IMHO, we discuss properties of a chipset here that tried to bridge the gap between 80s era computer synthesized sound and digital sample playback.

So again, if you like the Amiga sound as it was, use direct output or the straightforward nearest-neighbor mixing.

3.1 Band limiting

Band limiting of audio signals can be done in various places of the signal processing chain. On the Amiga, several precautions must be taken, however. The most prominent type of Amiga sound is Tracker music (Protracker, OctaMED, ..., <insert houndreds of others here>). In such tunes, the pitch (=sampling frequency) is variable. When one upsamples the signal to the target mixing rate and filters afterwards, the filter would have to be adapted to the original sampling rate in order to be able to successfully attenuate the spectral repetition distortions discussed in Sec. 2.3.

Instead, it makes sense to include the filter directly into the mixing stage. This way, only one set of precomputed filter coefficients tuned for the mixing rate is needed.

A simple kind of filtering is linear interpolation, which can be achieved by different means. One option would be to mix at 2, 3, 4, ... times the target mixing rate and perform averaging when scaling the rate down to the target mixing rate again. This approach by oversampling is in effect nothing else than linear interpolation, where the oversampling factor determines the accuracy of the interpolation. In practical matters, it is in many cases more efficient to compute the linear interpolation directly, especially when an accuracy $\geq \frac{1}{4}$ is demanded. For such linear interpolation, the effective filter impulse response is a triangle with a squared sinc as frequency response.

¹²OFDM is a prime example for that statement.

Nearest-neighbor resampling uses a rectangle as defining function, leading to a slowly decaying sinc in frequency domain. Linear interpolation is basically a triangular shaped function. Its frequency domain correspondence is a squared sinc, which decays considerably faster compared to the sinc in nearest-neighbor resampling.

It should be fairly obvious right now that I was aiming higher. From the myriad of approaches towards resampling I chose a classic: the windowed sinc. Why? In theory it's optimal. Of course, there are compromises when truncating the sinc function, leaving room for other approaches.

Mixing > 8 channels with an internal filter on 68k CPUs is quite a burden for them. So I restricted my initial consideration to a sinc in the range of $[-2, 2]$. This range includes the main and two side lobes. The sinc windowed sinc filter kernel function $h(t)$ is computed as follows:

$$h(t) = \begin{cases} \text{si}(\pi \cdot t) \text{si}(\pi \cdot t/r) & \text{with } r > 0, \text{ typical } r \in 2, 3 \\ 0 & |t| > r \end{cases} \quad (3)$$

Another approach to implement an effectively similar type of filter would be the Hermite spline.

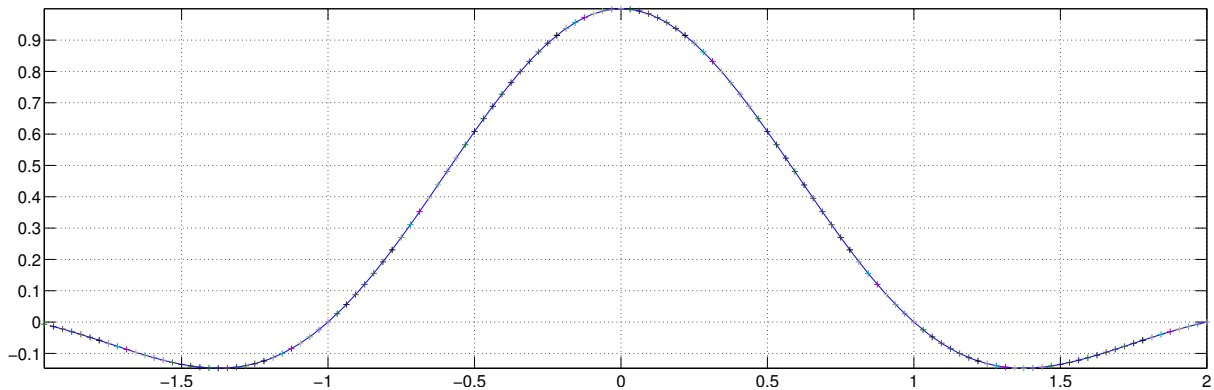


Fig. 7: Plot of two-lobe sinc function, along with the polyphase coefficients at 1/32 sample resolution.

For the upsampling case, the chosen range corresponds to four input samples at the positions $-1, 0, 1, 2$. Now the future positions would require for a causal implementation to delay the input accordingly. The problem is: when the inputs are at varying sampling rates, you actually get different delays. In case of Amiga sound mixing, this issue can be solved by a lookahead approach.

Sound reproduction on the Amiga commonly uses double-buffered DMA with chunks of data. Hence, when the DMA is supposed to start, a larger number of samples is directly available for a filtered mixing approach. This way, the phase between the mixed channels can be kept intact.

For an actual realtime implementation, only the precomputed table remains to be discussed. For these precomputed values we have now 4 coefficients per position. The position range is between two samples, an offset in the range $[0, 1)$. Thus, a useful approximation is to define a granularity (e.g. 1/32 sample position) and tabellize 32 steps for 4 coefficients per step.

3.2 Lowpass compensation

With band limiting filters in place, it's time to do something about the issue with missing/attenuated high frequencies. In order to get more high frequencies in the resulting output, an

appropriate compensation for the lowpass filter (see Sec. 2.5) is needed. I was interested how my own Amiga would fare. I generated a linear chirp signal in Matlab, played it with the Amiga 4000, captured the result and computed the transfer function as quotient between the measured PSD and the original PSD (Fig. 8).

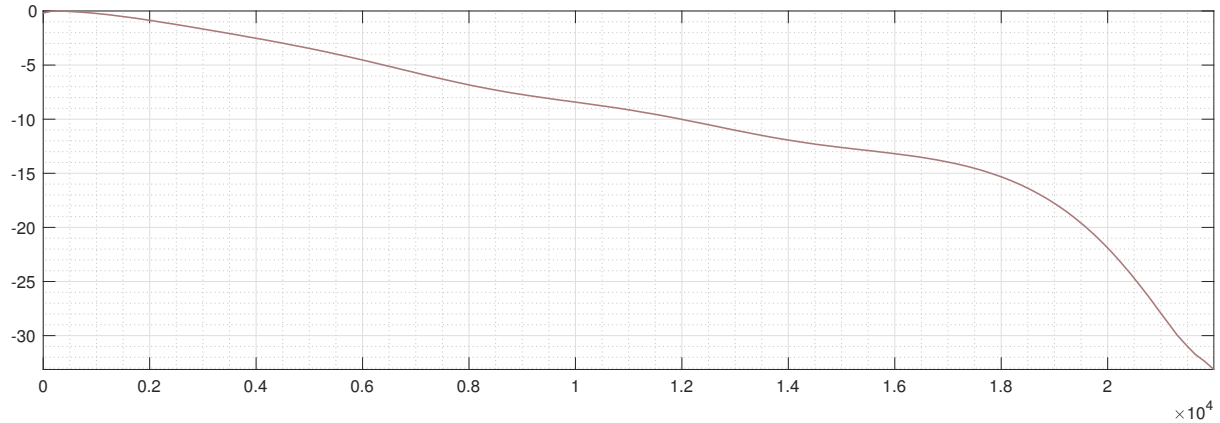


Fig. 8: Measured Amiga 4000 transfer function (10Hz-22.5 kHz chirp at 44336 Hz sampling rate over 20 seconds), with 7 kHz filter switched off

The transfer function I obtained is quite similar to the observations in [2], where a steep attenuation occurs beyond 19 kHz¹³.

Now, the transfer function of the A4000 is quite usable as a reference for most Amiga models. Then again, there is still the A1200. We can already assume a different transfer function from the corner frequency calculations in Sec. 2.5. Sadly, I've sold my A1200 back in the day when switching to the A4000. That changed recently, tanks to John "Chucky" Hertell's ReAmiga1200 project (and the a1k.org group order for PCB and associated parts). The measurement of the

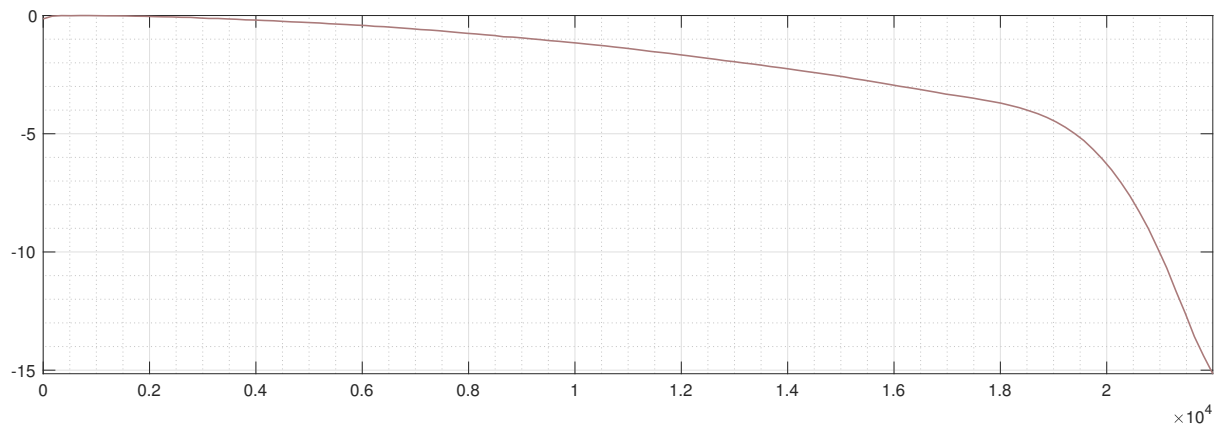


Fig. 9: Measured Amiga 1200 transfer function (10Hz-22.5 kHz chirp at 44336 Hz sampling rate over 20 seconds), with 7 kHz filter switched off

A1200 transfer function by example of my newly built A1200 computer¹⁴ is shown in Fig. 9. We can immediately see the difference to the A4000. The effective 3dB corner frequency is around 15.5 kHz instead of the usual 5 kHz. Please disregard the drop-out below 50 Hz. That's the input bandpass of the sampling interface I've used for the quick test. Frankly, I didn't expect such a huge difference between A1200 and the other Amigas. To ensure that my observation

¹³I recapped the A4000 just a week ago, with quality parts. Therefore, I expect my filters to work as designed.

¹⁴Audio filter specs like A1200Rev.2, decoupling caps Panasonic SVPF polymer electrolytes

was not isolated, I asked a friend (many thanks, Flype) to capture the chirp from his Rev. 2B A1200. The results matched. Indeed, the A1200 does need special treatment.

The best compensation filter for most Amiga models I could find so far is a fair trade-off between computational complexity and impulse response performance.

$$H(z) = \frac{4 - 2z^{-1}}{1 + 0.5z^{-1}} \quad (4)$$

This first-order recursive filter (a shelving high-boost filter design, converted to the standard representation of a first order IIR) can be implemented completely in registers and doesn't require any multiplications. It does have a 2.5 dB DC amplification, though. Let's see how it fares when compared to the Amiga 4000 transfer function. It can be seen in Fig. 10 that the curves match

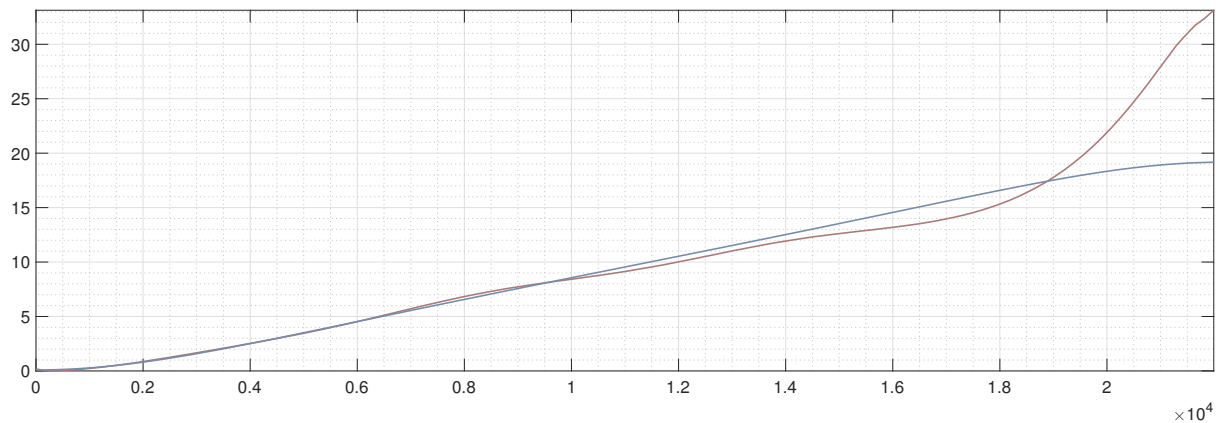


Fig. 10: Comparison between measured Amiga 4000 transfer function (inverted, red line) and designed filter (attenuated by a constant 2.5 dB for comparison purposes, blue line)

up quite well. It must be noted that in the respective figure, the constant 2.5 dB amplification was cancelled out to match the curves better. We'll come back later to that topic.

Instead of just overlaying the curves, the proper way of simulating the system performance is the combination of both transfer functions. The expected total performance is depicted in Fig. 11, along with the measured transfer curve. It can be seen that the implemented filter lines up quite

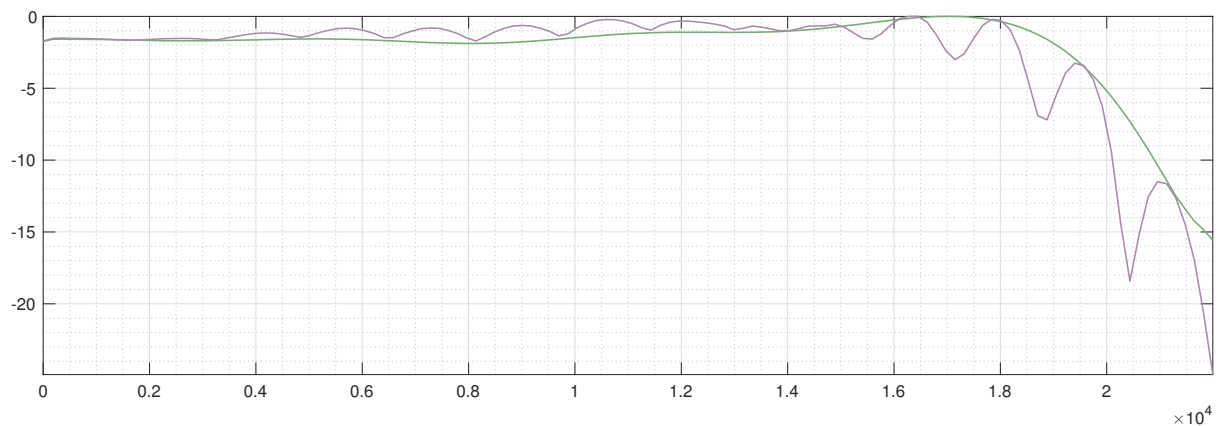


Fig. 11: Simulated transfer function of an Amiga 4000 with applied compensation filter (green) versus actual behavior of implemented compensation filter (red)

nicely with the projected results. The ripple which is increasing beyond 16 kHz is mainly caused by the transfer function of the sinc based band limiting filter.

The simple multiplier-less filter is quite adequate to provide a near-linear sound reproduction out of a stock Amiga. Yes, I've ignored the DC amplification in that statement. The point is: This filter does way more amplification (19 dB) for high frequency components. In order to avoid clipping in an integer implementation, the input must be attenuated anyway. One option is to include these considerations directly into the volume scaling constants of the mixing step. This way, no further instructions are spent in the filtering loop.

But then again, bits of dynamic range are precious in fixed-point implementations. Thus, a second option is to scale the forward coefficients from $[4, -2]$ to $[1, -0.5]$. The frequency response curve basically the same, only the amplification changes. The 0dB point is now exactly in the middle. On the lower and upper end of the spectrum the attenuation/amplification are -9.5dB and 9.5dB, respectively.

These parameters were tuned for 44 kHz playback. A useful low-complexity approximation of the compensation filter for 28.687 kHz DMA rate would be to change the $0.5z^{-1}$ in the denominator to $0.25z^{-1}$, again for most Amiga models.

Finally, there is the A1200 to consider. The filters proposed above are not appropriate for this Amiga model and will lead to overly amplified high frequencies. Thankfully, a filter design that approximates a linear frequency response with a tolerance of less than 0.2 dB in the range between DC and 18 kHz is also feasible with shift-based computations. For 44 kHz playback we get:

$$H(z) = \frac{1 - 0.125z^{-1}}{1 + 0.125z^{-1}} \quad (5)$$

The DC adjustment (factor 1.25 in this case) can be incorporated again in conversion tables. This leaves two shifts by 3 bits for the main computation loop in filtering. A suitable approximation for 28 kHz playback would be to change the -0.125 in the numerator to -0.0625.

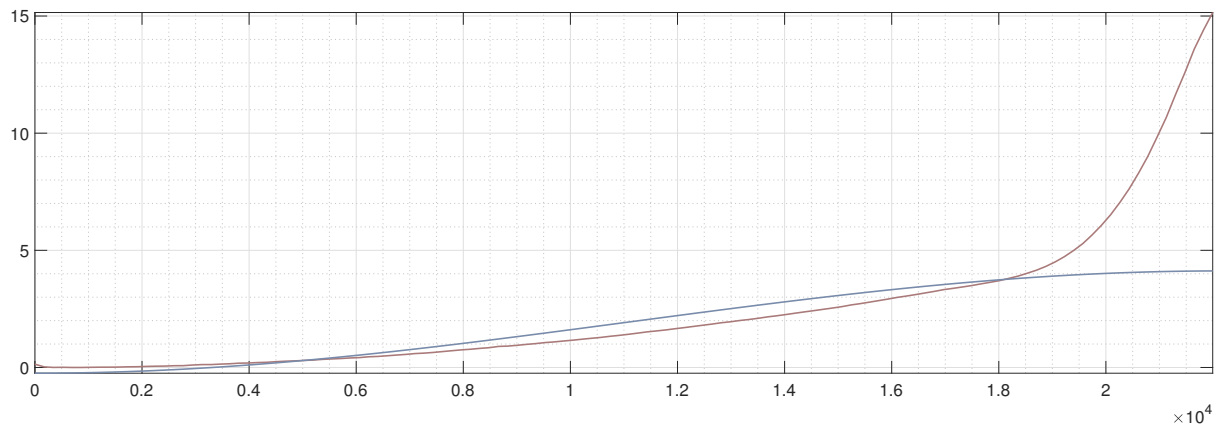


Fig. 12: Comparison between measured Amiga 1200 transfer function (inverted, red line) and designed compensation filter (44 kHz variant, blue line)

To be honest, the A1200 doesn't really require much of a frequency response correction filter. At least according to my listening habits, not much difference can be made out in typical cases between uncorrected and high-boosted playback.

4 14 Bit Mode and Calibration

The famous Amiga 14 Bit mode works by stacking two 8 Bit audio channels of the same panning. Since Paula contains a 65 steps hardware volume control, one of the stacked channels is assigned the maximum volume 64 (henceforth called HI) while the second channel is set to the minimum useful volume 1 (LO). This way, the coarse quantization steps of the HI channel can be augmented by the respective levels of the LO channel. Channels 0 and 1 are typically the HI channels for left and right panning, respectively. Channel 2 is the right LO channel and channel 3 the left LO channel. Other configurations are also possible but might require a slightly different calibration. That's why it is recommended to stick with the semi-standard assignment.

Assuming an idealized DAC behaviour, one can put 64 steps of the LO channel between each step of the HI channel. Sadly, the underlying assumption does not hold for practical purposes very well.

To understand some of the difficulties of getting 14 Bit to actually work nicely, it is necessary to have a look at Paula's workings again. Please recall figure 3 and section 2.2. At maximum volume, Paula will act as a near ideal comb filter and output 3.5 MHz impulses only.

When a channel is set to a lower volume, the Paula DAC operates differently. The volume setting does not change the voltage levels of the DAC itself. Rather it will enter a periodic PWM-style addition/subtraction DAC cycle, as outlined in section 2.4. This happens in the aforementioned resampling raster of 64 Paula ticks. Since two Paula channels of the same panning share one DAC, this process is carried out in sequence, with a phase difference of 45 degrees (measurements by Toni Wilen et al.).

One immediate consequence is that the stacking of the two channels is not synchronized to an exactly common point in time. Phrased differently, a step on the HI channel induces an immediate wideband pulse that is averaged out throughout the 55 kHz approximation raster by the LO channel. This relationship may be the reason why the calibration often requires levels bigger than decimal 64 in order to cancel audible differences between a HI level and its successor (speculative on my part).

Another consequence of the 55 kHz resampling at low volumes is that modulation byproducts will occur on the LO channel.

4.1 14 Bit Calibration

The Paula calibration was proposed by Christian Buchner and adopted by most 14 Bit capable audio solutions on the Amiga. Christian Buchner discovered that the relationship between adjacent HI levels and associated LO levels is not constant across the 8 Bit range. His calibration utility allows measure the correct number of LO steps between each pair of HI channel levels in a user-assisted way. The outcome is a calibration array for positive and negative amplitude ranges.

The calibration array `LOSTEPS[]` has a length of 256 Bytes and is organized as follows:

- 128 Bytes for negative range from -128...-1
- 128 Bytes for positive range from 0...127

Each Byte contains the number of LO steps between the level indicated by the position and the next higher level. Christian Buchner's reference code provides a Bresenham based stretch algorithm that will output a 16 Bit to 14 Bit conversion table.

It may be noted that this reference code is entirely based on positive LO steps, even in the negative amplitude range. While this approach looks consistent in the calibration utility, there are some drawbacks to be noted.

To approximate the largest negative 14 Bit number closest to 0, the assignment of amplitudes is $HI = -1$ and $LO = LOSTEPS[127]$. Now a low volume signal or even just some noise in the LSBs of the input signal will cause HI to vary between 0 and -1. That process produces additional noise on uncalibrated Amigas where the optimal HI/LO relationship is unknown.

16 Bit to 14 Bit conversion is in and out itself a quantization. Depending on application, either midthread or midrise quantization is performed. The former method places a so called dead zone around 0 such that this zero range is typically $\frac{1}{2}$ of the quantization step size in either direction. The second method excludes 0 as valid quantizer index and assumes that only nonzero negative or positive values are of interest.

It came as quite a surprise to me that the 16 Bit to 14 Bit conversion table example code produces neither type of quantizer. Zero is a valid output but any 16 Bit value < 0 directly maps to the aforementioned maximum negative number in 14 Bit. This amplified the 16 Bit to 14 Bit quantization error amplitude by a factor of 4 ($\approx 12\text{dB}$).

4.2 Revised 14 Bit Calibration

The fixes to the issues mentioned in section 4.1 are quite straightforward. Only the conversion table setup routine needs modest modifications (`_CreateTable`). The actual code converting 16 Bit to 14 Bit at runtime is unaffected.

The only modifications needed are the sign change of the correction level in the LO channel and 0 as the starting point on the HI channel. The outcome is a midthread quantization table. A mathematical expression of this process could be formulated as follows:

$$x_q = \frac{|x + 0.5 \cdot \Delta|}{\Delta} \cdot \text{sgn}(x) \quad \text{with } \Delta \approx 4$$

The quantization step size Δ is mentioned as approximation as the actual local step size depends on the calibration outcome. My personal experience is that a larger dead zone ($0.25 \cdot \Delta$) appears to sound slightly better. YMMV.

One important property of this symmetric quantization table is that only the LO channel is responsible for the first 64 steps in either polarity. This minimizes HI/LO channel interaction inaccuracies for low amplitudes where noise influence is most noticeable, especially for uncalibrated systems.

I've also noticed subtle differences in optimal calibration values between the left and right channel pairs. The updated calibration utility I've bundled with the fixed Paula AHI driver may be used to adjust left and right calibration independently.

The storage of this additional piece of information is done in my proposal by an extension of the file format. The first 256 bytes correspond to the usual common calibration information of a convenient point that minimizes the distortion on both left and right channel. This can be written as:

$$cal_{common} = ((cal_{left} + cal_{right}) \& 0x1FE) >> 1$$

With $\&$ as bitwise AND operation and \gg as bit shift¹⁵. All applications that rely on the traditional 256 Byte calibration array will still work as intended.

The second set of 256 bytes corresponds to the differences between left and right channel:

$$cal_{diff} = ((cal_{left} + cal_{right}) \& 1) + (cal_{left} - cal_{right}) \cdot 2$$

The storage of the LSB from cal_{common} in the differences array limits the available range of the differences to $-63 \dots 64$. I've tested several Paula chips from early 8364R7 to later 8364R7PD and 8364R7PL. None of them exceeded a difference of 10 between left and right channel calibration.

Since the LSB of the first equation is kept in the differences array, exact recovery of cal_{left} and cal_{right} is possible.

$$cal_{left} = 0.5 (2 cal_{common} + cal_{diff} \& 1 + (cal_{diff} \gg 1))$$

$$cal_{right} = 0.5 (2 cal_{common} + cal_{diff} \& 1 - (cal_{diff} \gg 1))$$

The presence of stereo calibration data is simply detected by a calibration file of length 512 on disk. Otherwise, the second half of the array just needs to be initialized with zeroes, denoting equal left and right calibration.

Sample code utilizing my proposed changes to the calibration process and the quantization table setup are incorporated into the updated AHI Paula driver on Aminet [8].

The calibration process is somewhat tedious. Therefore, I added some presets as possible starting points to the utility (Fig. 13). It's worth noting that the ECS revision of Paula got significant improvements wrt. DAC accuracy. The ECS Paula can be found in SMD Amigas as standard configuration (8364R7PL). A500+ usually also has the ECS Paula equipped (8364R7PD). By my experience, the ECS Paula is less commonly found in A3000 systems. If you can get a hold of an ECS Paula for A500/2000/3000, I'd recommend to install it.

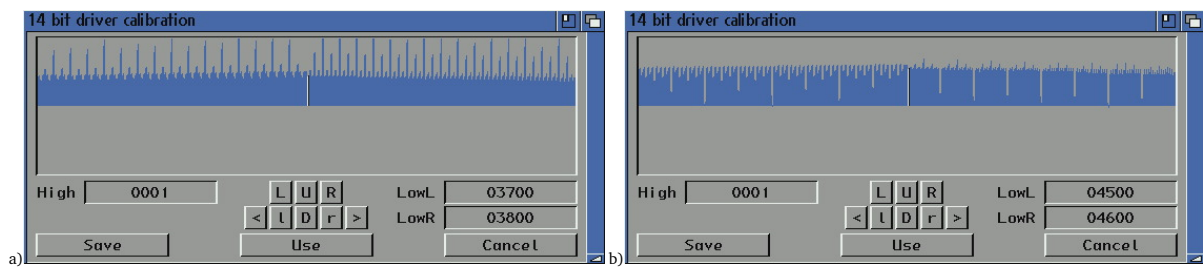


Fig. 13: a) Preset for OCS Paula 8364R7, b) Preset for ECS Paula 8364R7PL, 8364R7PD

5 Measurements

As noted in the introduction towards 14 Bit on Amiga in section 4, DMA driven high volume and low volume channels of the 14 bit stack won't be in perfect sync. By also taking Paula's comb filter and it's differential PWM approach into consideration, a final result matching a true 14 Bit quantizer cannot be expected.

¹⁵My apologies for mixing math expressions with C expressions.

I've used a regular PC for the measurements, equipped with an M-Audio Transit 24 Bit ADC and ASIO drivers. The test file consists of a 1 kHz 0 dBFS cosine wave with a sampling frequency of 44336 Hz, i.e. closest match of 44.1 kHz to PAL Amigas¹⁶. The inputs were recorded with Audacity (ASIO build) in 48 kHz/24 Bit and saved as 32 Bit floating point AIFF prior to the analysis. Devices under test were my A1200 and my A4000, both equipped with new Aluminium Polymer capacitors¹⁷ in the audio circuit.

The respective calculations for the analysis were done in Matlab. You find the script in the appendix A.2. In a nutshell: the script trims silence from the input file, then searches for a zero crossing on both ends (alternative: Hanning window) to avoid the leakage and calculates the PSD¹⁸ $S_{xx}[\Omega]$ over the whole remaining signal. Under the assumption of a single sinusoid, the dominant frequency is detected within the PSD¹⁹. The SNR of a perfect sinusoid $S_{xx}[\Omega_1]$ would then be:

$$SNR = 10 \log_{10} \left(\frac{S_{xx}[\Omega_1]}{\sum S_{xx}[\Omega] - S_{xx}[\Omega_1]} \right) \text{ [dB]} \quad (6)$$

Now, imperfections of the measured sine (i.e. harmonics) are indeed a kind of distortion. Yet, these are part of the quantized sinusoid signal itself and not supposed to be included in the noise term for ADC and DAC SNR computation. Hence, the equation becomes

$$SNR = 10 \log_{10} \left(\frac{S_{xx}[\Omega_1]}{\sum S_{xx}[\Omega] - \sum_k S_{xx}[k\Omega_1]} \right) \text{ [dB]} \quad (7)$$

Now, the first set of measurements are supposed to provide a baseline. Fig. 14 depicts the power spectral results of playing the 16 Bit file over a regular 8 Bit output channel. In order to compare the effect of the harmonics on SNR calculation (see eq. 6 vs. eq. 7), the first two plots are shown with and without harmonics of the measured signal. The blue plot on the right is the direct result of an 8 bit quantizer on the test signal, i.e. no analog path. Please note that the SNR calculated by eq. 7 matches the common shorthand equation $SNR = 6.02B + 1.76 \text{ dB}$ for 0 dBFS sinusoids. Nevertheless, the harmonics are shown in the subplot c).

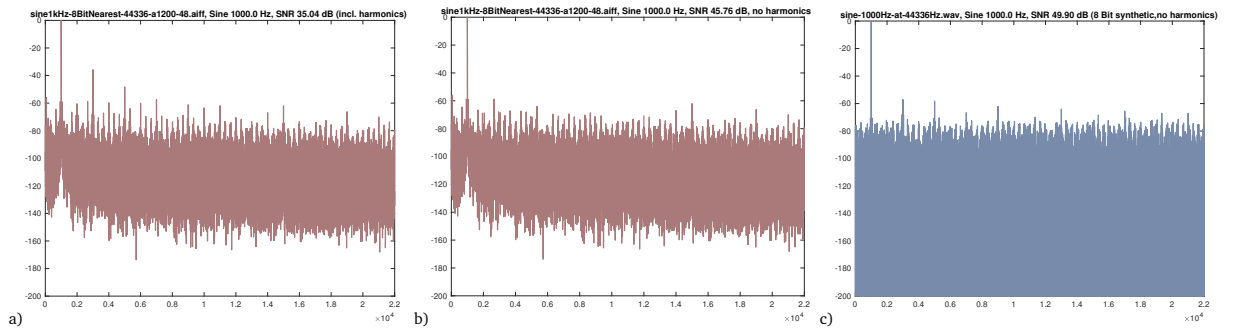


Fig. 14: Measurement results in regular 8 Bit mode: (left to right) a) full PSD with calculations including harmonics (SINAD), b) PSD without $k \cdot 1 \text{ kHz}$ harmonics (SNR), c) synthetic results for an 8 bit quantizer including harmonics (SNR)

It can be observed in Fig. 14b) and c) that the overall noise floor of Paula is in the expected range for an 8 Bit Chipset. On the other hand, Fig. 14a) reveals one of the culprits why Paula will likely not reach true 14 Bit capability. The DAC distortions on the sine wave (2,3,4,5,... kHz) reveal limitations with respect to DAC linearity (compare Fig. 13). It might be of further interest how

¹⁶see A.2, bottom of script

¹⁷Panasonic SVPF

¹⁸PSD=power spectral density

¹⁹alternatives for mixed signals: MUSIC, ESPRIT etc.

the obtained ECS Paula SNR figure of 45.76 dB compares to the results of the OCS Paula. The SNR of the latter was estimated in [9] to a value around 42 dB.

When it comes to the 14 Bit mode, there are two questions to be answered:

- SNR compared to 8 Bit output
- Effect of calibration

Beside the question how much actual SNR improvement can be gained by the 14 Bit mode, there is also the question how much - if any - measurable improvement can be obtained by Paula calibration. To answer the latter, the 14 Bit tests were run twice: once with uncalibrated defaults (flat $0x40 = 0 \dots 63$) and once with a calibrated setup. The points noted in section 4.2 concerning 16 Bit - 14 Bit quantization and calibration were part of the software. Since the test file was already at a native Amiga sampling rate, no further filtering was performed.

The first test results for the 14 Bit output were obtained on the A1200. An expected synthetic result for a 10 Bit quantizer is shown on the right hand side in Fig. 15. It can be seen that the

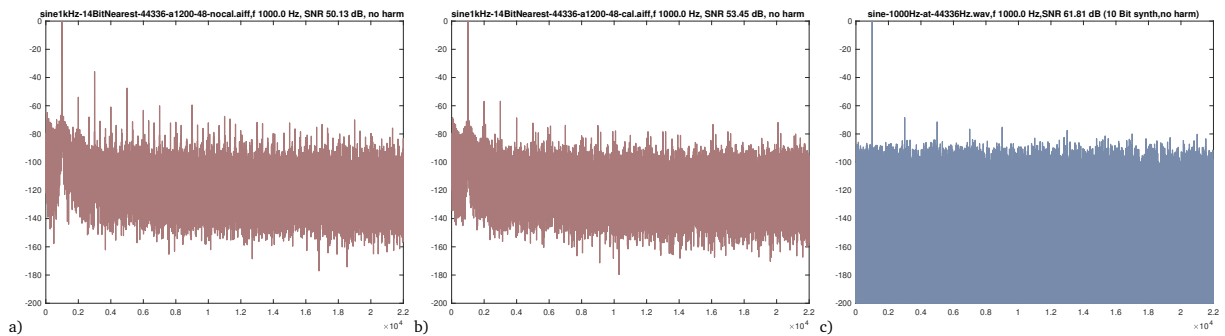


Fig. 15: 14 Bit results by PSD: a) A1200 uncalibrated (50.13 dB) b) A1200 calibrated (53.45 dB) c) synthetic results for a 10 Bit quantizer (61.81 dB)

noise floor is roughly consistent with the noise figure of the 10 Bit uniform quantizer test. Also noteworthy is the effect of calibration on this system which improves the SNR from 50.13 dB to 53.45 dB.

Further details of interest in the resulting PSDs reveal why the Amiga 14 Bit mode does not achieve the intended resolution. Since the low volume channel is subject to Paula's internal re-sampling (from 44.3 kHz to 55.4 kHz in this case), the intermodulation result does not augment the high volume channel sufficiently. Even in a calibrated setup, this effect is visible within the PSD, especially around 2 kHz in Fig. 15b).

Another 14 Bit test was performed on the A4000. Compared to the A1200, the A4000 features a more restrictive reconstruction filter. While it's downside is an attenuation of higher frequencies, it shows a clear benefit towards the SNR on the other hand. As shown in Fig. 16, the calibrated A4000 scored about 6 dB better than the A1200 amounting to 1 Bit more dynamic range.

Now, the blue plots that have been inserted as a reference to the figures are simulations only that haven't gone through a true cycle of DAC-Analog channel-ADC. A commonly mentioned rule of thumb suggests a typical loss of 1 LSB for it. With that in mind and some generosity towards rounding, an equivalent to true 12 Bit amplitude resolution out of Paula's 14 Bit mode could well be reasoned, at least in case of A500/600/2000/3000/4000. In a strict sense, the numbers accumulated in here suggest about 10 Bit measurable resolution.

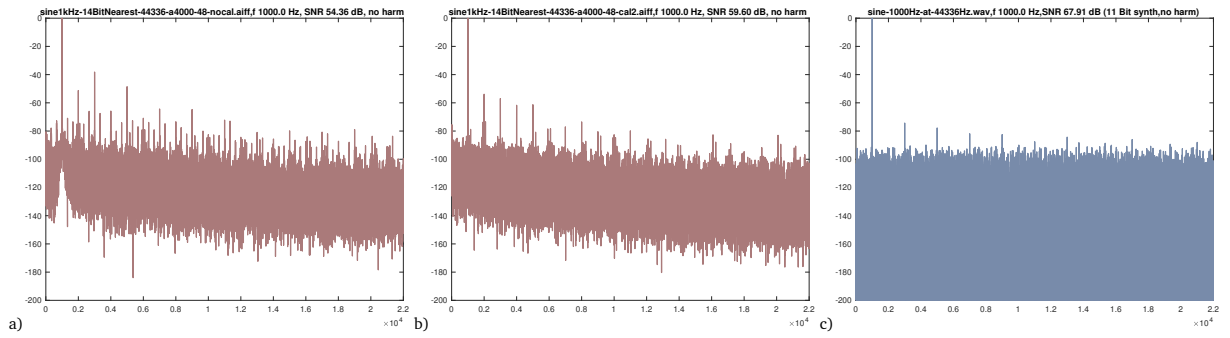


Fig. 16: 14 Bit results by PSD: a) A4000 uncalibrated (54.36 dB) b) A4000 calibrated (59.60 dB) c) synthetic results for a 11 Bit quantizer (67.91 dB)

There is a property of the measured PSD that might provide further insight as to why the quality seems perceptively higher at times. A considerable amount of energy contributing to the noise figure is present around the harmonics of the sine (see Fig. 16b) as a consequence of unsynchronised compensation signal energy from of the low volume channel. Whether this property of the PSD applies in a psychoacoustic sense remains to be determined and is beyond the scope of this document.

6 Conclusion

Writing down the theoretical models helped me to understand Paula's behavior in the usual operating modes a lot better. Of course, much in this text just verifies information that in some cases has been known for quite a while. As there were more pressing issues, I didn't dive into resampling distortions due to Paula's mapping of DMA samples onto it's fixed 54 kHz raster.

You might've been wondering why I chose to avoid one term throughout the text: Aliasing. Well, while doing my stuff and having a look at some papers I got aware of a branch in the signal processing community that refers to the spectral periodicity of digital signals as Alias Spectra. I'm none too happy with it. That term sure confused me when I started digging into this topic and I still don't like it. I've actually grabbed some books that are quite often seen as standard literature to see what the commonly cited scholars wrote up²⁰.

My personal opinion to the matter is: Aliasing on the ADC side is an effect that destroys the signal by overconvoluting the excess bandwidth into the base interval, where the higher frequencies alias the lower frequencies, making them indistinguishable. This is different on a DAC. Insufficient filtering will sure leave some of the spectral periodicity in the analog output but these (annoying) spectral contributions don't destroy the signal. You can continue with a better filter at any time and place in the processing chain to sufficiently suppress these. And that is IMHO different enough to the effects on the ADC side to merit distinguishing terminology.

There are some issues I couldn't resolve yet. In theory, the minimum period value with 31 kHz screen modes is 64. I never got that 55420 Hz rate to work properly with my PAL A4000. One channel always produced clicking sounds in regular intervals, pointing to DMA starvation issues. Period 65 on the other hand, works just fine. I guess, I have to stick with the 54567 Hz maximum output rate.

²⁰Girod clearly positioned himself against the Aliasing term for DAC effects, Oppenheim politely avoided to get dragged into terminology discussions

In total, this was an interesting retro-style experience for me. I've implemented the outlined filters into my music player but kept them optional. Some music really sounds much better but some other relying on the habits of good old Paula, doesn't.

If one would like to implement this approach, please keep in mind that the filters are meant to go along with each other. The short sinc interpolator by itself attenuates higher frequencies more than the Hermite spline, for example. In combination with the high pass filter however, they are a somewhat adequately matched pair (at least given the computational performance constraints of the classic Amiga). It also doesn't make much sense to combine nearest neighbor resampling with the high pass filter (unless you can count on high sampling rate input like CDDA, FLAC or MPEG Layer 3).

References

- [1] Christian Buchner, "Ver1.1 CDPlayer for Toshiba, 14bit output," 1997, http://aminet.net/package/disk/cdrom/14Bit_CDPlayer.
- [2] Antti S. Lankila, "Amiga sound playback note on interpolation," 2006, <https://bel.fi/alankila/modguide/>.
- [3] Toni Wilen, "Comments on 15 bit 44 khz audio idea," 2010, <http://eab.abime.net/showthread.php?p=677012>.
- [4] Various authors, "Low-level workings of Paula," 2012, <http://eab.abime.net/showthread.php?t=63227>.
- [5] B. Girod, R. Rabenstein, and A. Stenger, Einführung in die Systemtheorie, 1st ed. Stuttgart, Germany: B.G. Teubner, 1997.
- [6] T. Frey and M. Bossert, Signal- und Systemtheorie. Wiesbaden: Vieweg und Teubner, 2008.
- [7] A. V. Oppenheim, A. S. Willsky, and S. H. Nawab, Signals & Systems (2nd Ed.). Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996.
- [8] Blom, M. and Buchner, C. and Richter, H., "updated AHI Paula Driver," 2018, <https://aminet.net/package/driver/audio/AHI-Paula>.
- [9] A. Cohen and M. Mieszkowski, "Frequency synthesis with the commodore amiga for research on perception and memory of pitch," Behavior Research Methods, Instruments, & Computers, vol. 21, pp. 623–626, 11 1989.

A Test Code

A.1 Intermodulation test

```
;simple Paula intermodulation test
;assumes an active DBLPAL/DBLNTSC/Productivity screen mode
;written in 2019, Henryk Richter
;note: test code only, don't use for deployment
MINPER EQU      64      ;124 when not in double scan
```

```

MAXPER EQU 1120 ;

    lea    $dff0a0 , a0
    lea    tune , a1

    move.l a1 , (a0)
    move    #1,4(a0)
    move    #1280,6(a0)
    move    #63,8(a0)      ; left vol 63
    move    #$7f80,10(a0)

    move.l a1,$10(a0)
    move    #1,$14(a0)
    move    #1280,$16(a0)
    move    #64,$18(a0)    ; right vol 64
    move    #$7f80,$1a(a0)

    ; this loop scales periods up and down
    move    #$8003,$dff096
    move    #MINPER,d2
    moveq    #1,d1
mainloop:
    btst    #6,$bfe001
    beq     end

    ; sweep from 64 to 1120 up/down
    add.w    d1,d2
    cmp      #MINPER,d2
    ble.s    changedir
    cmp      #MAXPER,d2
    ble.s    nochangedir
changedir:
    eor.w    #$fffe,d1
nochangedir:
    bsr      delay

    move     d2,6(a0)
    move     d2,$16(a0)

    bra.s    mainloop
end:
    move     #$1,$dff096
    rts

;
; waste some time (not exactly correct but sufficient)
; in: A0=$DFF0A0
delay:
delay_a:
    btst    #6,$bfe001
    beq.s    delay_q

```

```

        move    $dff006-$dff0a0(a0),d0
        lsr     #8,d0
        cmp.w   #20,d0
        blt.s   delay_a
delay_b:
        btst    #6,$bfe001
        beq.s   delay_q
        move    $dff006-$dff0a0(a0),d0
        lsr     #8,d0
        cmp.w   #20,d0
        bgt.s   delay_a
delay_q:
        rts

```

```

        section bla,data_c
tune:    dc.b    $7f,$80

```

A.2 SNR Calculation in Matlab

```

%
% SNR from Audio Files , specifically recorded sine waves
% where f_sine is assumed to be much smaller than f_s ,
% i.e. something like 1kHz vs 44 kHz
%
zcross=1; % if(1), search for 0-crossing , else apply hanning window
fjitter=10; % in Hz, assume some jitter of replayer/recorder (total)
qbits=11; % see at bottom: quantizer test for comparison purposes
          % expected: 6.02*qbits+1.76 dB for a sine :-)
noharmonics=1; % delete harmonics from SNR computation ? 1 = SNR / 0
    ↪ = SINAD

fig1=12; % figure numbers
fig2=13;

% reference file: this script assumes that no trim is necessary
reffile='sine-1000Hz-at-44336Hz.wav';
%reffile='Sine_tone_1000Hz_at_44336Hz.wav';

% A4000
% calibration
%recfile='sine1kHz-14BitNearest-44336-a4000-48-cal.aiff';
%recfile='sine1kHz-14BitNearest-44336-a4000-48-cal2.aiff';
% no calibration
%recfile='sine1kHz-14BitNearest-44336-a4000-48-nocal.aiff';
%recfile='sine1kHz-14BitPolyphase-44336-a4000.aiff';
%recfile='sine1kHz-14BitNearest-44336-a4000.aiff';
% A3000
% no calibration
%recfile='sine1kHz-14BitNearest-88200-a3000.aiff';

```

```

% A1200
% with calibration
recfile='sine1kHz-14BitNearest-44336-a1200-48-cal.aiff';
% no calibration
%recfile='sine1kHz-14BitNearest-44336-a1200-48-nocal.aiff';
%recfile='sine1kHz-8BitNearest-44336-a1200-48.aiff';
% old tests
%recfile='sine1kHz-14BitNearest-44336-a1200-48.aiff';
%recfile='sine1kHz-15BitNearest-44336-a1200-48-2.aiff'; % bad
%recfile='sine1kHz-15BitNearest-44336-a1200-48.aiff';

colors=[
    .999 .888 .999;
    .578 .578 .666;
    .888 .999 .888;
    .780 .733 .545;
    % spare colors
    .470 .545 .666;
    .555 .555 .555;
    .666 .478 .478;
    .666 .545 .666;
    .470 .666 .470;
];

[wref, refrate]=audioread(reffile, 'native');
[wrec, recrate]=audioread(recfile);
wref=wref(:,1); % use left channel only
wrec=[zeros(10000,1);wrec(:,1);zeros(10000,1)]; % alternative:
    ↪ averaging, max. power etc.

% parameter check of loaded file
if( recrate < 44000 )
    fprintf('some_sampling_rate_>=44kHz_would_be_nice\n');
    return
end

if( length(wrec) <= 2*recrate )
    fprintf('please_record_more_than_two_seconds\n');
    return
end

if noharmonics
    noharm='no_harmonics';
else
    noharm='';
end
%
% eliminate silence at start and/or end, if needed (auto trim)
%
l=4000; % accumulate signal energy over 4000 samples (~1/10s@44kHz)

```

```

segs=fix(length(wrec)/l);
segpow=zeros(1,segs);
for i=0:segs-1
    snip=wrec(1+l*i:l*(i+1)); % segment
    segpow(1+i)=sum(snip.^2); % segment power
end
msempow=median(segpow);
stdsempow=sqrt(var(segpow));

% beginning
stdmul=2;
idx1=1;
for i=1:fix(segs/2)
    if( segpow(i) < ( msempow - stdmul*stdsempow) )
        idx1=i+1;
    else
        break;
    end
end
% end
idx2=segs;
for i=segs:-1:fix(segs/2)
    if( segpow(i) < ( msempow - stdmul*stdsempow) )
        idx2=i-1;
    else
        break;
    end
end
%fprintf('idx1 %d idx2 %d\n',idx1,idx2);
% trim recorded file
wrec=wrec(idx1*l:idx2*l);

% next: either search for zero crossing at start/end (single sine
    ↪ only)
%      or apply hanning window
if( zcross == 1 )
    idx1=1;
    for i=1:fix(length(wrec)/2)
        if( (wrec(i)<0) && (wrec(i+1)>=0) )
            idx1=i+1;
            break;
        end
    end
    idx2=length(wrec);
    for(i=length(wrec)-1:-1:fix(length(wrec)/2))
        if( (wrec(i-1)<0) && (wrec(i)>=0) )
            idx2=i-1;
            break;
        end
    end
end
end

```

```

% fprintf('idx1 %d idx2 %d\n',idx1,idx2);figure;plot(wrec(1:100));
    ↪ figure;plot(wrec(end-100:end))
wrec = wrec(idx1:idx2);
else
    wrec = wrec.*hann(length(wrec));
end

%
% compute PSD from recorded file
%
Wrec=fft(wrec);
sldrec=abs(Wrec(1:fix(length(Wrec)/2))).^2;

%
% plot PSD
%
[lev,pos]=max(sldrec);
Hl2=10.*log10(sldrec./lev);
scalel2=0:(recreate/2)/length(Hl2):recreate/2;scalel2=scalel2(1:end-1);
f=figure(fig1);plot(scalel2,Hl2,'color',colors(7,:), 'linewidth',1.5);

%
% dominant frequency
%
f1=scalel2(pos);
fperpos=(recreate/2)/length(sldrec); % Hz per bin

%
% SNR(total), not considering harmonics
%
sigpower=sum(sldrec(pos-round(fjitter/2/fperpos) : pos+round(
    ↪ fjitter/2/fperpos)));
noisepower=sum(sldrec)-sigpower;

snr=10*log10(sigpower/noisepower);
figure(f);title(sprintf('%s, Sine %3.1f Hz, SNR %2.2f dB (incl.
    ↪ harmonics)',recfile,f1,snr));
ylim([-200 0]);xlim([0 22000]);

%
% common variant: separate signal and also it's harmonics from
    ↪ background
% noise, i.e. harmonics of a sinusoid are also part of the "signal"
%
if noharmonics
sigpower=sum(sldrec(pos-round(fjitter/2/fperpos) : pos+round(
    ↪ fjitter/2/fperpos)));
%noisepower=sum(sldrec(1:round(20000/fperpos)))-sigpower; % optional:
    ↪ max. 20 kHz

```

```

sldn=sldrec;
if noharmonics
    idx = 2*pos;
    while( idx < length(sldn) )
        sldn(idx-10:idx+10) = 0; % +-1 Hz
        idx = idx+pos; % multiple
    end
end
noisepower=sum(sldn)-sigpower;

[lev3 , pos3]=max(sldn);
Hl3=10.*log10(sldn./lev3);
f=figure(fig2); plot(scalel2 ,Hl3, 'color', colors(7,:), 'linewidth', 1.5);
ylim([-200 0]); xlim([0 22000]);

snr=10*log10(sigpower/noisepower);
figure(f); title(sprintf('%s, Sine_%3.1f_Hz, SNR_%2.2f_dB, %s', recfile
    ↪ , f1, snr, noharm));
ylim([-200 0]); xlim([0 22000]);

end

%
%
%-----
% Quantization test
%-----
% compute PSD of reference file
% see above for inputs: qbits, wref, refrate, colors, fjitter (or run
    ↪ script)
%
if isfloat(wref)
% -1..1 normalized input
wrefq=fix(abs( wref.*(2^(qbits-1))-0.5 )).*sign(wref);
else
% integer input
wrefq=fix(abs(double(wref)/2^(16-qbits))).*sign(double(wref));
%figure; hist(wrefq,[-2^(qbits-1):2^(qbits-1)])
end

Wref=fft(double(wrefq));
sldref=abs(Wref(1:fix(length(Wref)/2))).^2;

[m1, pos1]=max(sldref);
Hl=10.*log10(sldref./m1); Hl(Hl<-200)=-200;
scalel=0:(refrate/2)/length(Hl):refrate/2; scalel=scalel(1:end-1);

% dominant frequency
flr=scalel(pos1);
fperpos=(refrate/2)/length(sldref); % Hz per bin

```

```

rsigpower=sum( sldref( pos1-round(fjitter/2/fperpos) : pos1+round(
    ↪ fjitter/2/fperpos) ));
%noisepower=sum(sldref(1:round(20000/fperpos)))-sigpower; % optional:
    ↪ max. 20 kHz
rsldn=sldref;
if noharmonics
    idx = 2*pos1;
    while( idx < length(rsldn) )
        rsldn(idx-10:idx+10) = 0; % +-1 Hz
        idx = idx+pos1-1; % multiple
    end
end
rnoisepower=sum(rsldn)-rsigpower;

[lev3 ,pos3]=max(rsldn);
Hl4=10.*log10(rsldn ./ lev3);
f=figure(7);plot(scale1 ,Hl4, 'color', colors(7,:), 'linewidth',1.5);
ylim([-200 0]);xlim([0 22000]);

rsnr=10*log10(rsigpower/rnoisepower);
figure(f);title(sprintf(' %s, _Sine_ %3.1f_Hz, _SNR_ %2.2f_dB_(%d_Bit_
    ↪ synthetic,%s)',reffile ,flr ,rsnr ,qbits ,noharm));

%
%
%-----
% cosine generator for various sampling rates
% (optional)
%-----
%
%
%
if 0
%
% cosine tone at frequency f in Hz
% (len = fs*seconds)
% -> write WAV
%
% Paula clocks, OCS 15 kHz screenmodes (old Denise)
% NTSC 3579545/124 = 28867.3 Hz
% PAL 3546895/124 = 28604 Hz
%
sec=10;
f=1000; % 1 kHz

paulaclock=14187580/4; % PAL Amiga system clock / 4 = Paula clock
periods=[63 74 80 124 161 223 321 443 886 1419 1773]; % Paula periods
%periods=[80]; % 44.3 kHz

```

```

paulafreq= repmat(paulaclock, size( periods))./ periods; % corresponding
    ↪ sampling frequencies

for j=periods

fs=int32(paulaclock/j); % round down

len=fs*sec;
t=double([0:len-1]);

oa=int16(32767*(cos( 2 * pi * double(f) * t./double(fs) )));

oname=sprintf('sine-%dHz-at-%dHz.wav',int32(f),fs);

audiowrite(oname,oa,fs,'BitsPerSample',16); % wavwrite(oa,fs,16,oname
    ↪ );

end

end

```