On the sample rate feedback in USB audio

Christoph van Wüllen
– DL1YCF –

March 2, 2021

1 Introduction

This work models the sample rate adaption taking place in USB audio input. In the most common asynchronous mode, there is a feedback mechanism by which the sound card can tell the PC to (slightly) increase or decrease the sample rate, such that there is a perfect match and no over- or underruns occur.
Samples to not arrive and are not consumed one-by-one, but usually in blocks. In the Teensy audio module, the block size for processing the samples is currently `AUDIO_BLOCK_SIZE`=128, and the goal is that the temporal average of samples already read but not yet processed is 1.5 blocks at the time when a block is "fetched", this means one block is available and another one is being filled and is at half-filling. Half-filling ensures maximum resilience against any jitter, should the next "fetch" occur a little earlier or later.
Since there were problems with audio sample underruns with the Teensy4, I looked at the values of the buffer filling and the sample rate feedback reported to the PC and saw large-amplitude oscillations where there would ideally be a damped oscillation of the buffer filling towards the target value. These oscillations are a consequence of how the feedback is actually implemented, and this calls for a mathematical analysis of the feedback loop.

2 Feedback loop

In this simulation we ignore any buffering and assume a continuous stream of samples. Then we can treat the problem with differential equations. What

actually happens in the computer is then a discretized version thereof.
Let v_1 and v_2 be the sample rates of the Teensy and the PC, assumed to be constant. Then x_1, the number of audio samples already processed by the Teensy at time t, is simply

$$x_1(t) = v_1 t. \tag{1}$$

The PC does not send audio with a constant sample rate. It is told to vary its sample rate by a time-dependent correction $v_{corr}(t)$ sent by the Teensy. Therefore the actual sample rate of the PC is $v_2 + v_{corr}(t)$ and the number of samples received from the PC is

$$x_2(t) = x_2^0 + v_2 t + \int_{\tau=0}^{t} v_{corr}(\tau)\mathrm{d}\tau \tag{2}$$

The additive constant x_2^0 is related to the buffer filling at $t = 0$, the first call to update() which actually gets data. It is normalized such that $x_2(t) - x_1(t)$ is zero if there are exactly 1.5 buffers of data, that is, the target is that x_2 and x_1 are equal. Therefore, the dynamical variable of interest is

$$X(t) = x_2(t) - x_1(t) \tag{3}$$

and the feedback loop has the function to keep this variable small. For 128-sample blocks, it should stay within ± 63 to avoid under- and overruns.
In the "feedback loop", v_{corr} (which is directly related to the value of the variable `feedback_accumulator`) is never set to any specific value but only increased or decreased. This increment is proportional to the difference of half of the audio buffer size and the number of samples in the being-filled buffer. Thus the feedback operationally is implemented as

$$\frac{\partial v_{corr}}{\partial t} = -\alpha X(t) \tag{4}$$

where we explicitly written a minus sign such that $\alpha \geq 0$. The parameter α is controlled by the programmer, it is proportional to the factor applied to

```
AUDIO_BLOCK_SIZE/2 - count                    (5)
```

before adding this to the feedback accumulator. Since

$$\frac{\partial X}{\partial t} = (v_2 - v_1) + v_{corr}(t) \tag{6}$$

a new variable $V(t)$ is defined such that

$$V(t) = (v_2 + v_1) + v_{corr}(t) = \frac{\partial X}{\partial t} \tag{7}$$

In the feedback loop, the value of X drives the *change* of v_{corr} and thus V, so we have

$$\frac{\partial V}{\partial t} = \frac{\partial^2 X}{\partial t^2}(t) = -\alpha X(t) \tag{8}$$

This equation is well known, it is the differential equation of a simple harmonic oscillator. Its general solution is

$$X(t) = A\cos(\omega t) + B\sin(\omega t) = \sqrt{A^2 + B^2}\cos(\omega t - \phi) \tag{9}$$

and the two parameters are given by

$$\omega = \sqrt{\alpha} \tag{10}$$
$$A = X_0 = X(0) \tag{11}$$
$$B = \frac{1}{\omega}V(0) = (v_2 - v_1) + v_{corr}(0) \tag{12}$$

The phase factor ϕ is not terribly interesting, what counts is the amplitude

$$X_{max} = \sqrt{X_0^2 + (v_2 - v_1 + v_{corr}(0))^2} \tag{13}$$

The buffer filling thus oscillates with this amplitude around its optimum value, and over/underruns result if the amplitude is larger than half the audio block size (currently: 64). Of course, the correction of the sample rate sent to the PC also shows this oscillatory behaviour with amplitude ωX_{max}.

3 Demonstrative solutions of the original feedback loop

Having set up the mathematical model, we look at solutions for some initial conditions. First we determine the update of the frequency accumulator that has to be performed upon each call to update(). To this end we first note

$$v_{corr} = \frac{1000}{2^{24}} * (f - f_0) \tag{14}$$

where f is the current value of the frequency accumulator and f_0 its nominal value which is (rounded to the nearest integer)

$$f_0 = 44.1 * 2^{24} = 739875226 \tag{15}$$

For a given buffer filling X, we have to increase/decrease v_{corr} at each call to update() such that

$$\frac{\Delta v_{corr}}{\Delta t} = -\alpha X \tag{16}$$

$$\Delta v_{corr} = -\alpha(\Delta t)X \tag{17}$$

Assuming a block size of 128 and a nominal sample rate of 44100,

$$\Delta t = 128/44100 \tag{18}$$

such that we finally have determined the change of the frequency accumulator in each call to update()

$$\Delta f = -\frac{2^{24}}{1000}\frac{128}{44100}\alpha X \approx -48.696\ \alpha X \tag{19}$$

In the current implementation, $\Delta f = -X$ which means $\alpha \approx 0.0205$ such that $\omega \approx 0.1433$ and the oscillation frequency is 0.023 Hz. This matches nicely the observed oscillation frequency of about one cycle every 50 seconds.
The most important figure is not the oscillation frequency, but the amplitudes both for V and X. The absolute value of the latter one must stay below 64, otherwise underruns or overruns will result. We therefore discuss X_{max} for different scenarios.

The first scenario is trivial. If $V_0 = V(0)$ (sample rates of Teensy and PC perfectly match, possibly after adusting the PC's sample rate) and $X_0 = 0$ (exactly 1.5 sample buffers available at $t = 0$), then V will be zero and stay zero, since the buffer remains half-filled. This is the ideal situation the feedback loop should achieve sooner or later.

The second scenario assumes matching sample rates ($V_0 = 0$, but at the beginning the buffer is not exactly half-filled, that is, X_0 is somewhere between -64 and +64. From equation (13) we see that $X_{max} = |X_0|$ and the buffer filling will oscillate between $\pm X_0$. V also oscillates with amplitude ωX_0. This matches the experimental observation: in my hardware setup, where $V \approx 0$, I observed the the buffer filling oscillated between 30 and 90 (in one particular experiment, this corresponds to $X_{max} \approx 30$), and that the feedback accumulator oscillated around its reference value with an amplitude of about 70000 which corresponds to about 4 Hz, which is close to 30ω.

The third scenario assumes that the PC is too slow or too fast ($V_0 \neq 0$) but that accidentially one starts with an ideally-filled buffer ($X_0 = 0$). In this case the buffer filling will oscillate with an amplitude

$$\frac{|V_0|}{\omega} \approx 7|V_0| \tag{20}$$

If the maximum permissible amplitude is 64, this means the feedback loop can only cure sample rate differences up to 9 Hz. The maximum allowed difference would be larger if one chooses a larger ω, that is, a larger factor in the update of the feedback accumulator. Than the feedback is tighter and the PC is corrected before over/underrun occurs. However too large values of ω probably won't help much since there is a delay in the PC sample rate adaptation which is not taken into account here.

The fourth and most complex scenario assumes that both the sample rates do not match exactly ($V_0 \neq 0$) and that the initial buffer filling is not at exact half-filling of the being-filled buffer ($X_0 \neq 0$). One sees that the feedback loop can prevent overflows provided that

$$64 > \sqrt{(X_0)^2 + \left(\frac{V}{\omega}\right)^2} \tag{21}$$

$$|X_0| < \sqrt{4096 - \left(\frac{V_0}{\omega}\right)^2} \tag{22}$$

(assuming an audio block size of 128). This means that the inital buffer filling must be closer and closer to the ideal value the larger the sample rate difference is. For sample rate differences of 1, 2, 4, and 8 Hz, the maximum permissible deviation of the initial buffer filling from its optimum value is 63, 62, 57, and 31. So at least theoretically, the feedback loop works well for sample rate differences up to 4 Hz, since there is a very large probability that $|X_0|$ will be smaller than 57 (this translates to an 128-sample buffer filling between 7 and 121).

4 Achieving damped oscillations

Of course, increasing ω will expand the "island of stability" at the expense of being less resilient to some real-world disturbances such as occasional drop-outs, delays in the sample rate adaptation, discretization errors and so forth. It would be clearly preferable if the feedback loop would produced damped oscillations, such that the buffer filling approaches the optimal value (half-filling) and stays there, which implies that v_{corr} is not updated any more (stays constant) and then of course has the correct value $(v_1 - v_2)$
Thus we will investigate a correction of the form

$$\frac{\partial v_{corr}}{\partial t} = -\alpha X(t) - \beta V(t) \tag{23}$$

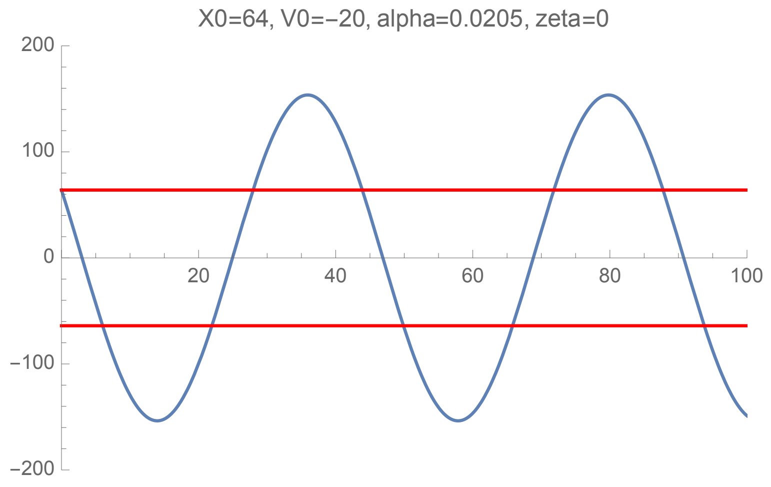
The first term tries keep the buffer at half-filling, while the second term tries to nail down the feedback sample rate correction to the actual sample rate difference.
We thus have

$$\frac{\partial v_{corr}}{\partial t}(t) = \frac{\partial^2 X}{\partial t^2}(t) = -\alpha X(t) - \beta \frac{\partial X}{\partial t}(t) \tag{24}$$

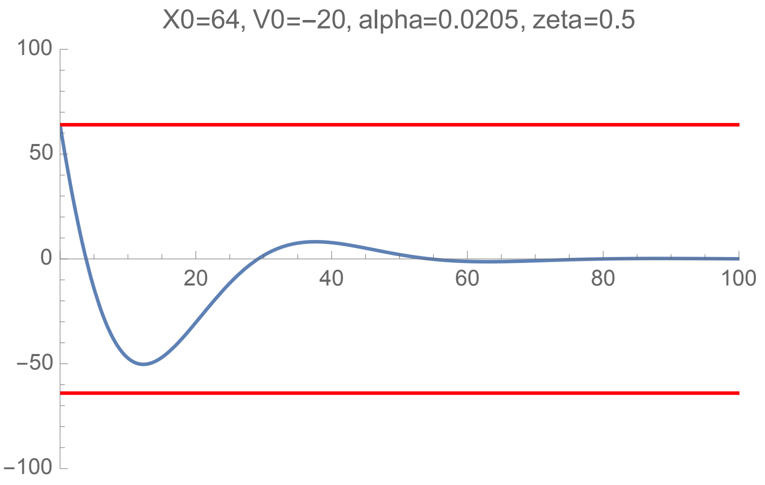
This differential equation is also well known, it is the differential equation of a damped harmonic oscillator. Again, $\sqrt{\alpha}$ is the angular frequency of the oscillator and the so-called damping ratio is defined as

$$\zeta = \frac{\beta}{2\sqrt{\alpha}}, \qquad \beta = 2\zeta\sqrt{\alpha} \tag{25}$$

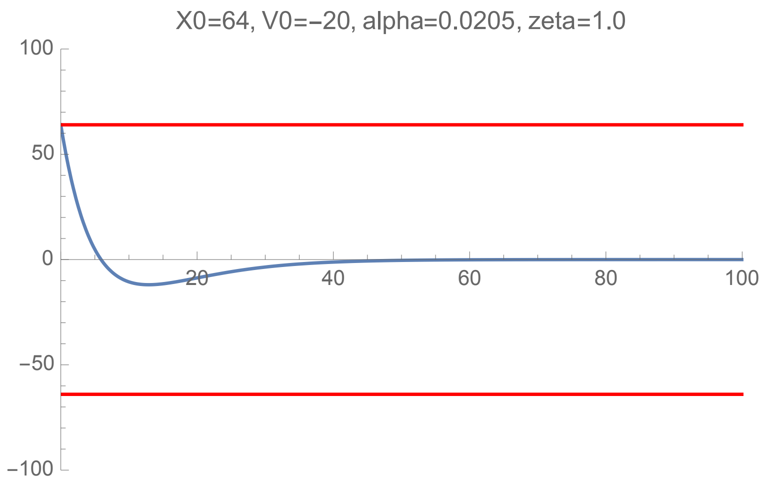
A damping ratio of about $\zeta \approx 0.5$ will lead to a strong damping which should stabilize the buffer filling in two or three oscillations. The so-called ,,critical damping" is achieved with $\zeta = 1$, here there are no oscillations but a single "over-shoot" and then a quick stabilization at the target. This can easily be investigated numerically (I use the Mathematica program). As initial values, $V = -20\ \mathrm{s}^{-1}$ and $X = 63$ were chosen, this means the sample rate of the PC is 20 Hz too slow, and initially the buffer filling is close to the upper limit. This is the typical situation encountered immediately after the first underrun, which cannot be avoided if the sample rate difference is that large. The original (undamped) feedback (with $\alpha = 0.0205$ and $\zeta = 0$) hits the next underrun condition ($X = -63$) after about 6 seconds, and at that time, V is essentially unchanged (still ≈ -20) such that the initial condition will be reinforced and there will be underruns every 6 seconds. The numerical solution is shown in this figure, where the red lines mark the buffer low- and high-water mark (horizontal axis: seconds, vertical axis: buffer filling relative to half-filled).



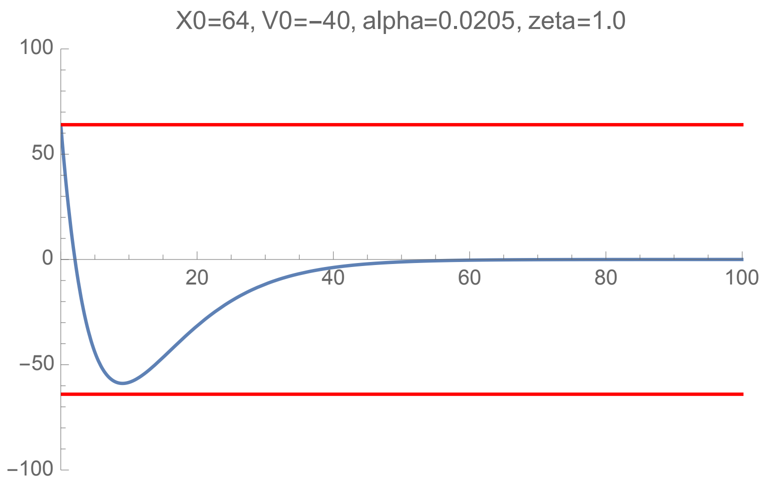
With the same initial conditions and a damping coefficient $\zeta = 0.5$ (this means that $\beta = \omega \approx 0.1431$ for $\alpha = 0.0205$ as before) the numerical solution of the damped oscillator gives



Finally, the "critically damped" case ($\zeta = 1$) is shown. In this case, there is only one over-shooting of the buffer filling which than quickly approaches the optimum value.



The buffer filling still moves quickly away from the ”dangerous region”, while having only a small over-shoot. Damping beyond $\zeta = 1.0$ (over-critical damping) is possibly harmful since this keeps the buffer close to the high-water mark unnecessarily long. Experimentally, it turns out that this case can handle sample rate too low by 40 Hz, as shown in the following diagram, and resilience to such large sample rate differences seems more than enough.



Implementing the damped case. Damping means that the feedback loop not only ”looks” at the buffer filling in each call to update(), but also measures how the buffer filling changes, that is, the difference between the buffer filling in the current and the previous call to update(). We thus have

$$\frac{\Delta v_{corr}}{\Delta t} = -\alpha X - \beta \frac{\Delta X}{\Delta t} \tag{26}$$

$$\Delta v_{corr} = -\alpha(\Delta t)X - \beta(\Delta X) \tag{27}$$

$$\Delta f = -\frac{2^{24}}{1000}\left\{\frac{128}{44100}\alpha X + \beta \Delta X\right\} \tag{28}$$

Here, X is the buffer filling a the time of the current update(), while ΔX is the difference between the actual buffer filling and that observed in the previous call to update(). Taking the original value of $\alpha \approx 0.0205$ and $\zeta = 0.5$ (that is, $\beta \approx 0.1431$) one arrives at

$$\Delta f = -X - 2400(\Delta X) \tag{29}$$

Note that the value of $2400(\Delta X)$ can be quite large since the samples come from the USB interface in packets and this happens asynchronously to the calls to update(). Therefore, X contains a high-frequency oscillatory component. To avoid large jumps in the feedback value, the total update from the damping part is distributed over many calls to update() and restricted in each of them to be within ± 64, but without affecting the long-term average of the feedback accumulator. So what is actually done in each call to update() is

```
$f \leftarrow f - X;$
$g \leftarrow g - 2400(\Delta X);$
**if** $(g < -64)$ **then**
    $f \leftarrow f - 64;$
    $g \leftarrow g + 64;$
**else if** $(g > 64)$ **then**
    $f = f + 64;$
    $g = g - 64;$
**endif**
```

where f is the variable controlling v_{corr}.

Handling overruns and underruns. Depending on the initial conditions and/or the amount of sample rate mismatch, a few under- or overruns cannot

be avoided until things have settled down. It turns out that no special treatment is needed in this case except that one has to take underruns and overruns into account when "measuring" the sample rate difference, that is, when determining ΔX.
An *overrun* occurs if the input buffer is full. In this case a certain number of incoming samples is just discarded, while the buffer stays at maximum filling. Without any correction, the feedback loop would "see" $\Delta X = 0$ and therefore assume a perfect match of the sample rate. Since ΔX is simply counted as the difference between the current and previous value of the buffer filling

$$\Delta X = X - X_{old} \tag{30}$$

the variable X_{old} simply has to be decreased by the number of discarded samples each time an underrun occurs. This is done in the USB audio receive callback. When an underrun occurs in update(), no samples are "consumed", instead implicitly an empty block of audio samples is transferred downstream. In the next call to update(), the buffer filling fill be higher by about the number of samples in one audio block. Therefore, X_{old} is increased by `AUDIO_BLOCK_SAMPLES` whenever an underrun occurs. As a technical note, in order to modify X_{old} in update() interrupts must be momentarily disabled, such that there can be no race conditions if update() and the USB audio receive callback want to change X_{old} at the same time.

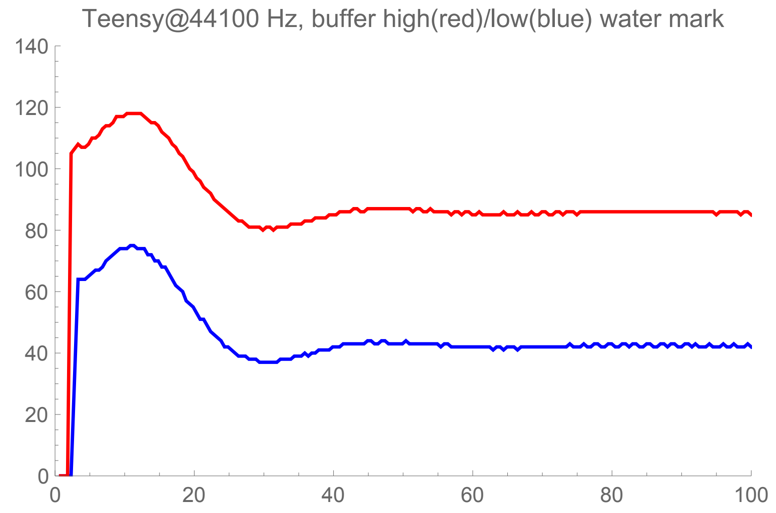
Other sample rates. Because most ist done with integer arithmetic, the α-dependent update will not change for sample rates not too different from 44100 kHz, that is, upon each call to update() the feedback accumulator will be updated by $-X$. This means that at 48000 kHz sample rate the value of α will be slightly different, namely $\alpha \approx 0.0224$ there. So the damping ratio $\zeta = 0.5$ now implies $\beta \approx 0.1495$, such that for 48000 kHz, the actual update changes to

$$\Delta f = -X - 2508(\Delta X) \tag{31}$$

The exact value of the damping coefficient ζ is, however, not overly critical. The simulations presented above indicate it should be somewhere between 0.5 and 1.0, so when increasing the "magic factor" to about 3000 will probably give a reasonable damping for both 44100 and 48000 Hz.

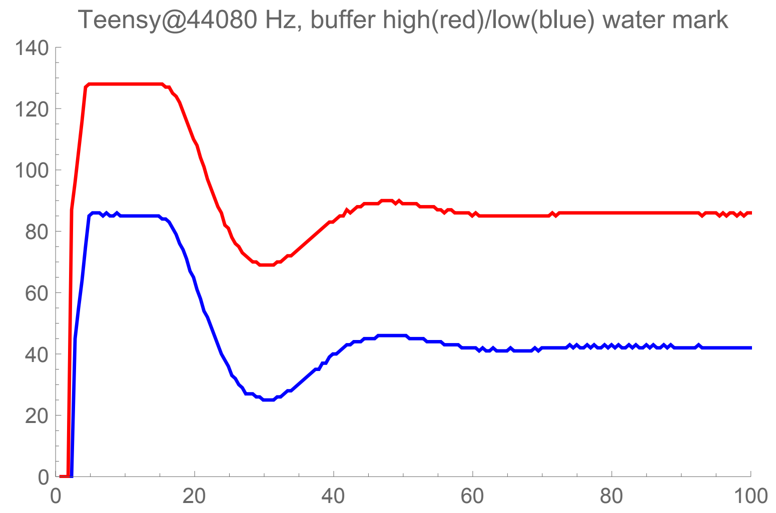
5 Measuring the performance on Teensy hardware

Of course, the proof of the pudding is in the eating, so we have to test the outcome on real hardware. To this end, we have instrumented the Teensy such that every 500 msec, the maximum and minimum buffer filling in the last 500 msec period is printed on the serial console. We plot for the first 100 seconds these two values. In the plots, the red line follows the high-water and the blue line the low-water mark. As the audio output device (which sets the pace) we used the MQS pulse-width-modulated output. The following figure shows the result:



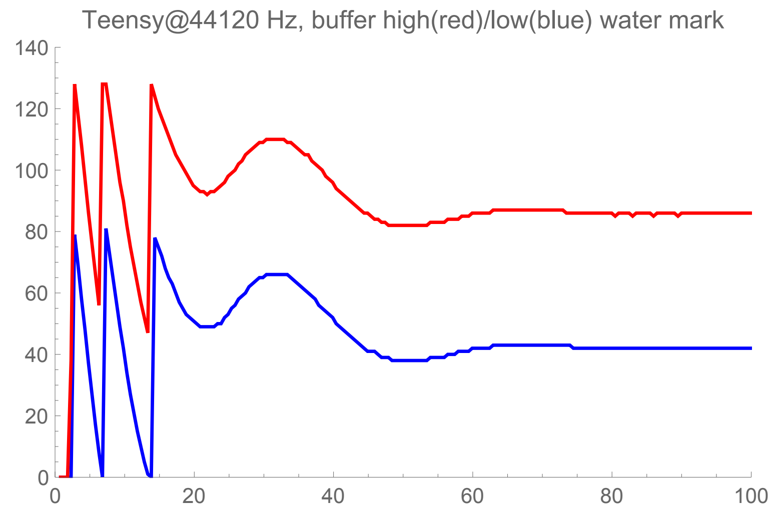
Upon startup, the buffer is empty. After about 2 seconds, the PC opens the Teensy as audio output card and starts to send the audio samples. There is always a single underrun at the beginning, because at the time the PC starts to send data, not enough samples are present to fill more than one buffer. Apart from this first underrun, no further underruns and overruns occured, and the buffer filling quickly stabilizes. The minimum buffer filling after stabilization was ~ 42 and the maximum count ~ 86, so the buffer filling is always 64 ± 22. The reason for this ”band” is that the samples from the USB interface arrive in chunks, and this happens asynchronously to the calls to update(). The experimental outcome suggests that the chunk size is somewhere around 44. The value of v_{corr} stabilizes to 0.22 Hz, which means that the sample rates of the PC and the Teensy match quite well.

This result is already a success, since the original feedback loop produced underruns, associated with noticeable audio cracks, every 10–15 seconds. The hard test for the feedback loop however is the case where there is significant mismatch in sample rates. To do the experiment, the sample rate of the Teensy has been changed from its nominal values of 44100 Hz to 44080 and 44120 Hz, such that it is either 20 Hz too slow or 20 Hz too fast. Note that the original feedback loop just cannot handle this case. To investigate the performance of the new feedback loop, the sample rate in `AudioOutputMQS::config_i2s()`, from which the parameters for setting the audio PLL are computed, has been changed. The next figure shows what happens if the Teensy is too slow:



After the PC starts delivering audio data, the buffer very quickly hits the high-water mark. Several underruns occur while the buffer stays being well filled. After about 20 seconds, the PC has adjusted to the slow speed, and the buffer filling gradually decreases and then stabilizes to the optimum value which is again 64 ± 22 samples. Of course, v_{corr} as calculated from the value of the feedback accumulator f then stabilizes at -20 Hz. Listening to music while performing this experiment, it becomes clear that the overrun situation hardly audible, obviously since every now and then a few samples are discarded. This contrasts with an underrun situation, where a whole audio block (currently: 128 samples) is missing and inserted as zeroes in the output stream.

The buffer filling curves look quite different initially if the Teensy is too fast, as documented by the next figure:



After the buffer has been filled for the first time, it is drained and the first underrun occurs already 5 seconds after the PC started sending audio. This is to be expected, because with the Teensy sample rate being 20 Hz to high, each second 20 samples more than delivered by the PC are consumed. If an underrun has happened, the next time the input buffer will be almost full, therefore both curves jump upwards. There is a second underrun at about 14 seconds, but after this the feedback loop has stabilized the buffer filling and the v_{corr} also stabilizes at 20 Hz. The experiment was repeated several times, in many cases the second underrun does not occur. Listening to music during this test, an underrun is normally accompagnied by a notable audio crack.