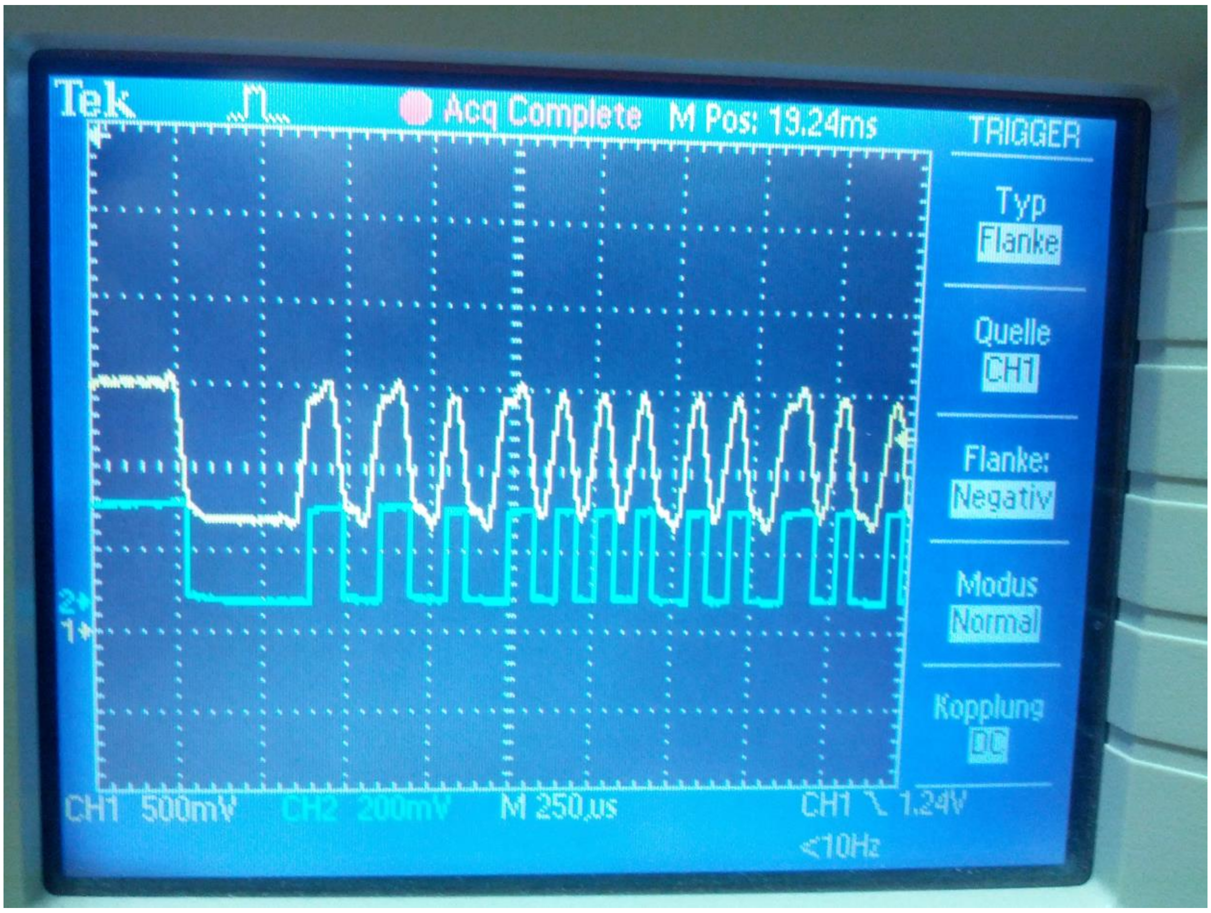
Blinkenrocket Debugging, V1 – 10/2016

Background: After assembling my 2 Blinkenrockets, I had the problem of neither text nor animations could be uploaded via the web interface. Also the Python script did not work. I've tried various devices to send, including an Android Nexus smartphone, an iPhone 5, a Lenovo Yoga Pro laptop, and a Lenovo Thinkpad laptop. Only on the Thinkpad laptop I could upload scrolltext but still it did not work every time. (The volume control of all devices was fully turned on)

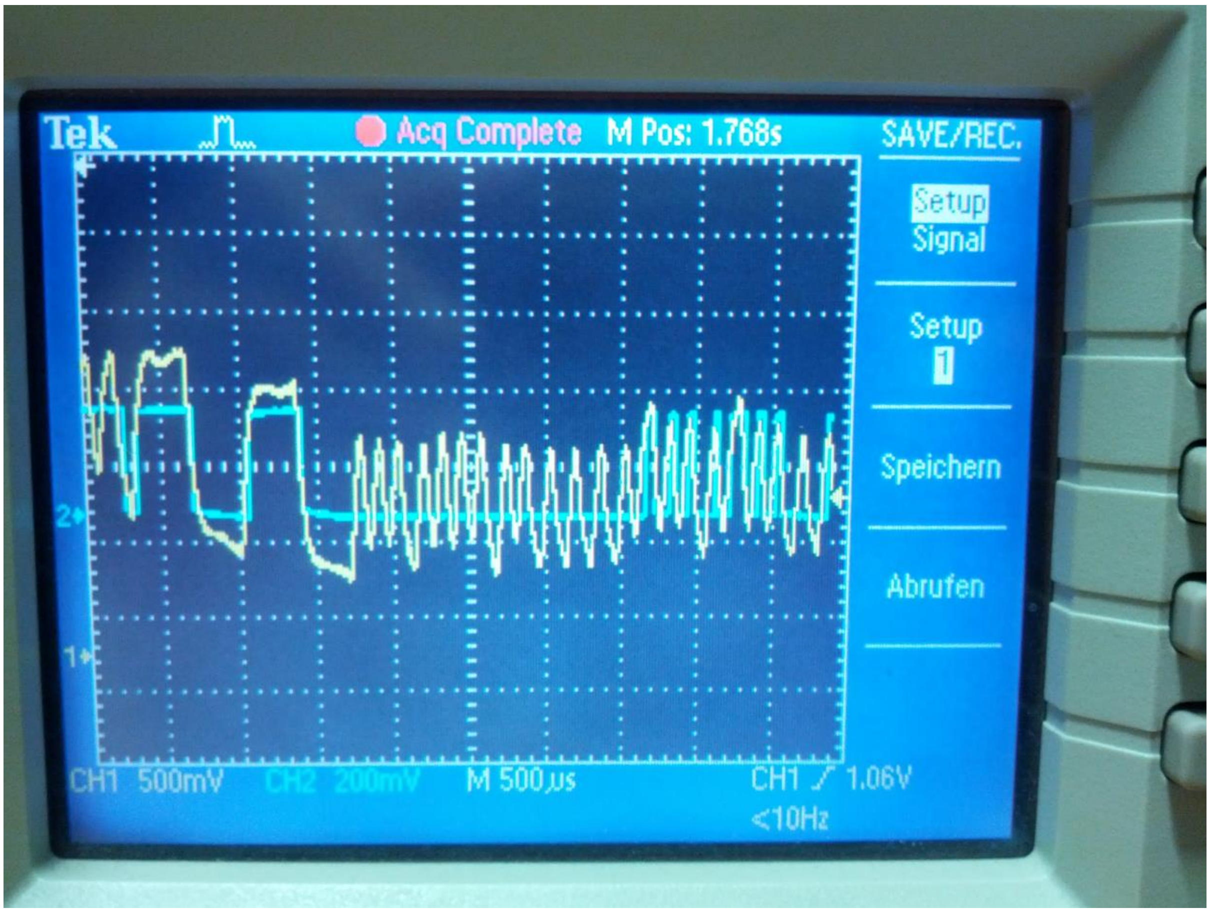
<u>Analysis of the audio input signal at the GPIO Input Pin of the ATTiny88 microcontroller</u>

A measurement with the oscilloscope probe connected after the input circuitry of the audio signal on the Blinkenrocket PCB shows the following signal trace (audio from the working Lenovo Thinkpad laptop):



The yellow channel is the analog input signal, the blue channel shows the activity of the pin change interrupt (ie the calls of the ISR for the input pin) - output to PC2 (pin E2).
The intervals (bit times) can be reconstructed correctly here!

Here is the same situation when playing back from the Lenovo Yoga laptop:

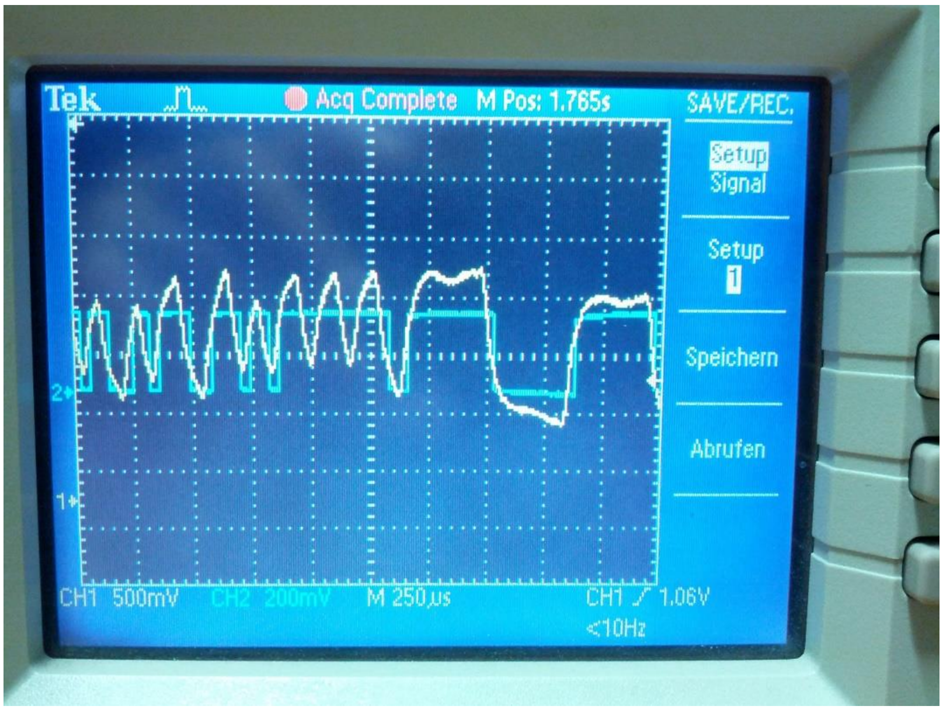

Tek
Acq Complete M Pos: 1.768s
SAVE/REC.
Setup Signal
Setup 1
Speichern
Abrufen
CH1 500mV CH2 200mV M 500µs CH1 ⁄ 1.06V
<10Hz

It becomes clear that by the drifting of the analog signal a reconstruction of the high / low changes by the pin change interrupt becomes impossible. The measuring principle can only work if the signal is stably at about the middle of the high / low thresholds of the input pin, otherwise the pin-change interrupt does not trigger or triggers at the wrong time ...

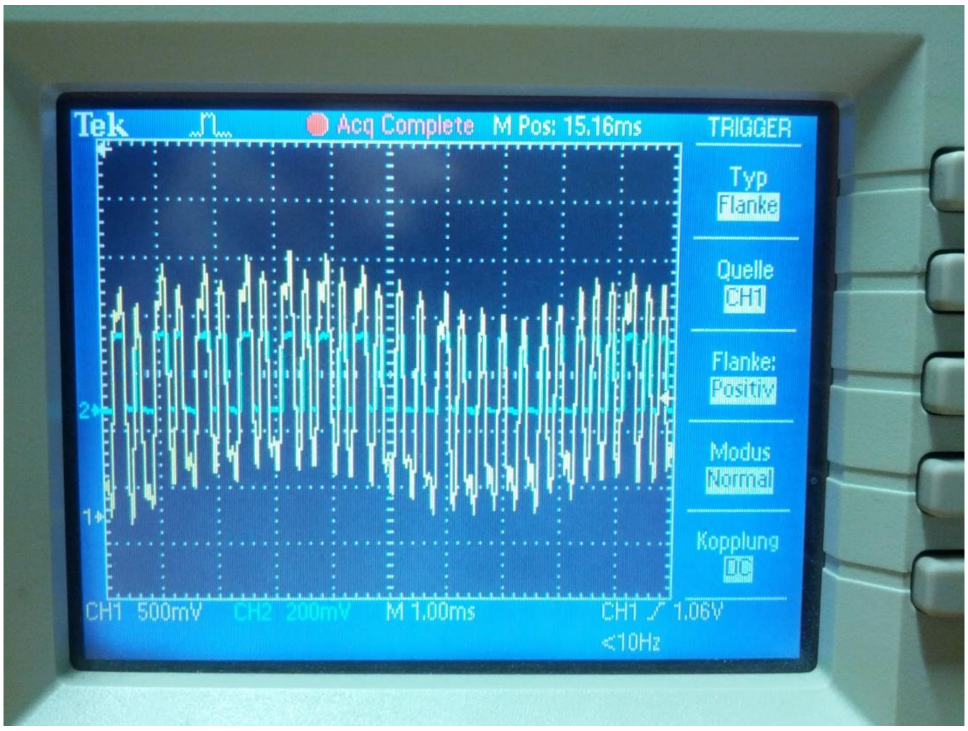
The input of the Blinkenrocket is connected with Clamping diodes (limited to approx. +/- 0.7V), then the signal is decoupled with a capacitor / highpass and raised by means of a voltage divider to the desired level between the thresholdLow and the thresholdHigh the GPIO input pin. The high-pass filter does not cut "sharp" enough to completely remove the slow drift of the signal.

This problem is apparently dependent on the player/audio device and possibly also on the played waveform itself. It seems that playing back square-wave signals leads to particularly strong problems because the output stage of the (some) audio hardware is not designed for rectangles.

Here is another example of a distorted signal that is not properly recognized by the pin change interrupt:

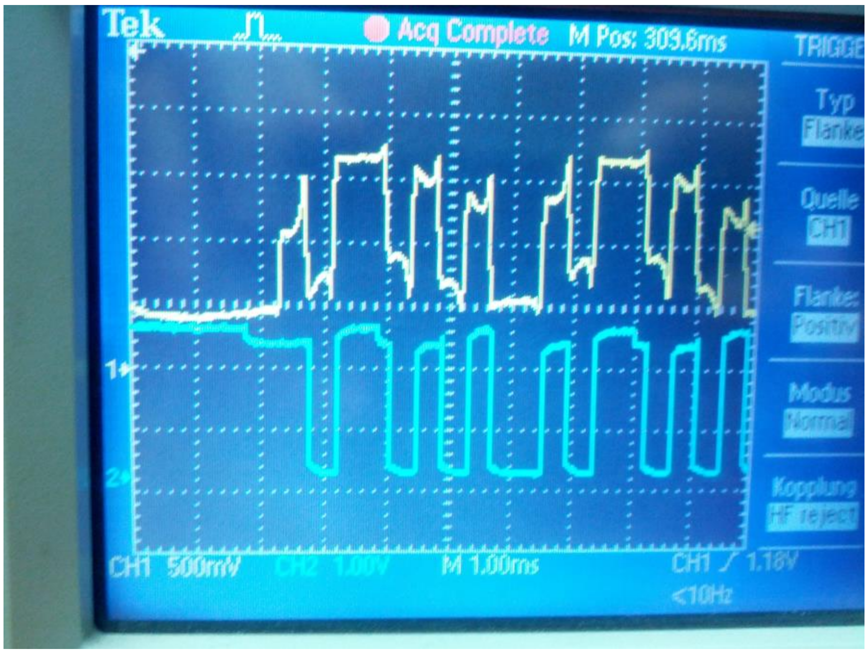

Tek
Acq Complete M Pos: 1.765s
SAVE/REC.
Setup Signal
Setup 1
Speichern
Abrufen
CH1 500mV CH2 200mV M 250µs CH1 ∕ 1.06V
<10Hz

And here the drifting in coarser time resolution - under these circumstances, a decoding of the signal is not possible:



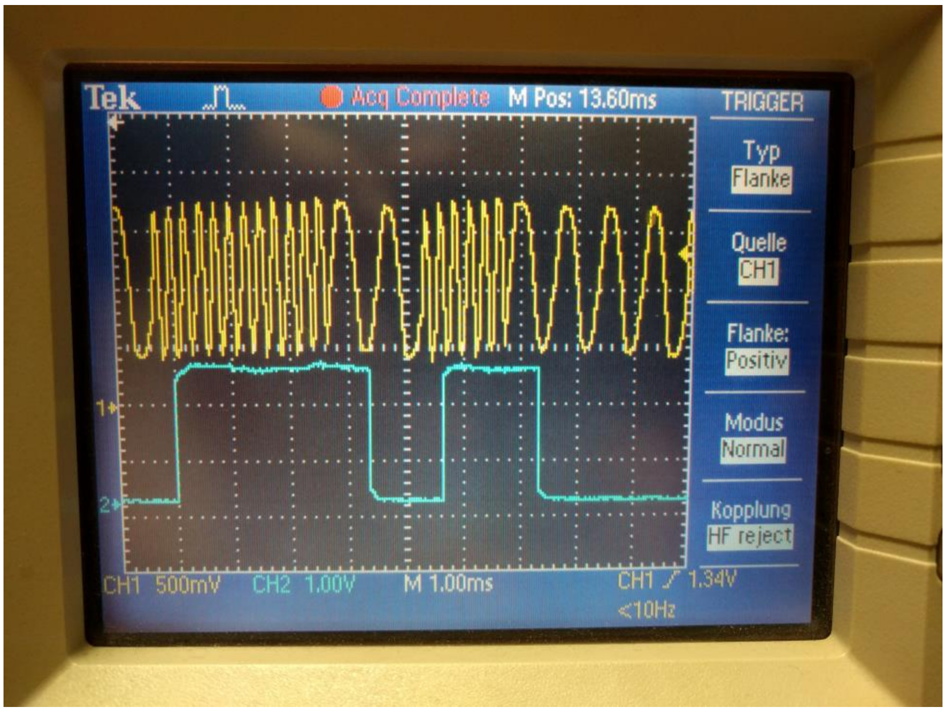
Strategy: Use the ADCs for sampling

Luckily (or because of the prudent design of the Blinkenrocket HW-creators), the input pin is also ADC-capable ☺ - The first strategy was to extended the duration of the high / low-phases of the rectangular signal in the Python script (the ADC would otherwise be too slow). The Oszi image shows a good detection of the short / long coding despite a significant drift of the signal (Lenovo Yoga Laptop)::



Nevertheless, the measurement was not reliable enough to receive longer scrolling texts or animations when the audio wave was played with the Lenovo Yoga laptop of via Smartphone. Obviously, the errors could still lead to incorrect bit patterns which could not be corrected by the Hamming method.

In course of the next trials, the square-wave signals generated by the Python script were replaced by different faster and slower sine waves to encode the low / high phases. Here (at least when played via my Yoga laptop) the output signal drifts much less strongly. After a certain "settling time" - which also depends on the sampling rate of the ADC - the two frequencies could be distinguished quite reliably (detection see blue signal):



Through this procedure, I was able to transfer a longer scroll text as well as animations over the yoga laptop to the Blinkenrocket via the Blinkenrocket.py script

(However, there were still problems when paying back the sine waves from the webedit – react / javascript context in the browser)

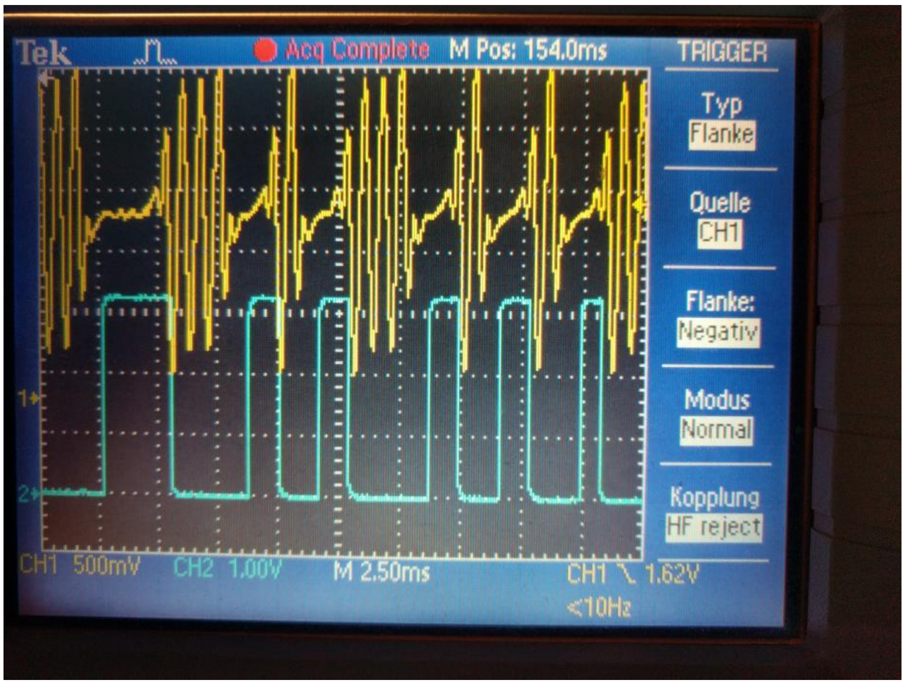
Blinkenrocket Debugging, update – 19.12.2016

After several more trials and failures, I found a method which seems to work from all my audio playback devices (using browser and/or python script).

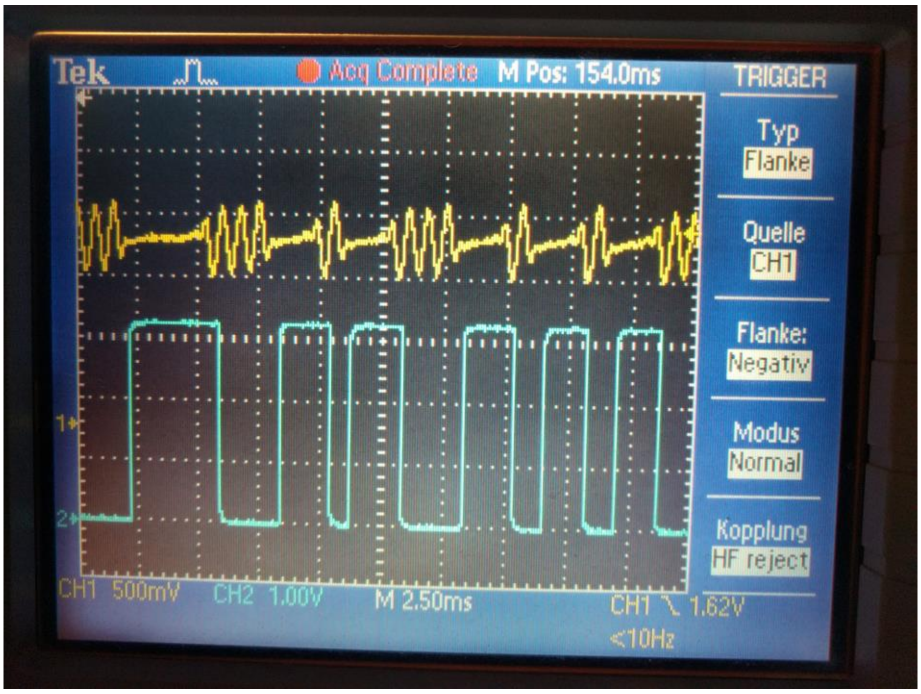
For debugging purposes I added an SPI connection to an Arduino UNO which outputs the recognized bytes to UART / serial Window so that transmission problems can be found more easily.

The information is encoded via sine waves that are faded in and out to reduce the drifting which occurs when playing back sharp signal edges on certain audio hardware. Furthermore, the slow sine wave has completely omitted and the end of a transmission is detected if the signal activity remains under the threshold of the noise level for a particular time (which is about twice the maximum valid bit time). Thus, information can be reconstructed corrected also if the playback volume setting is not at its maximum.

Example Playback from Lenovo Thinkpad (via Binkenrocket WebGui) - Volume setting 100%:

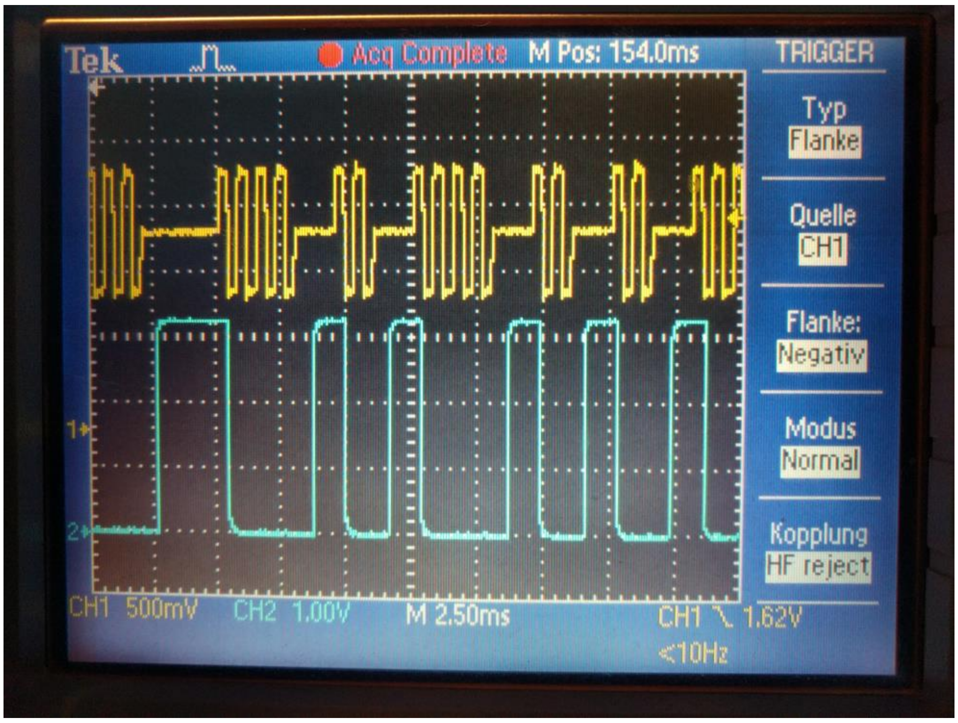


Example Playback from Lenovo Thinkpad (via Binkenrocket WebGui) - Volume setting 60%:



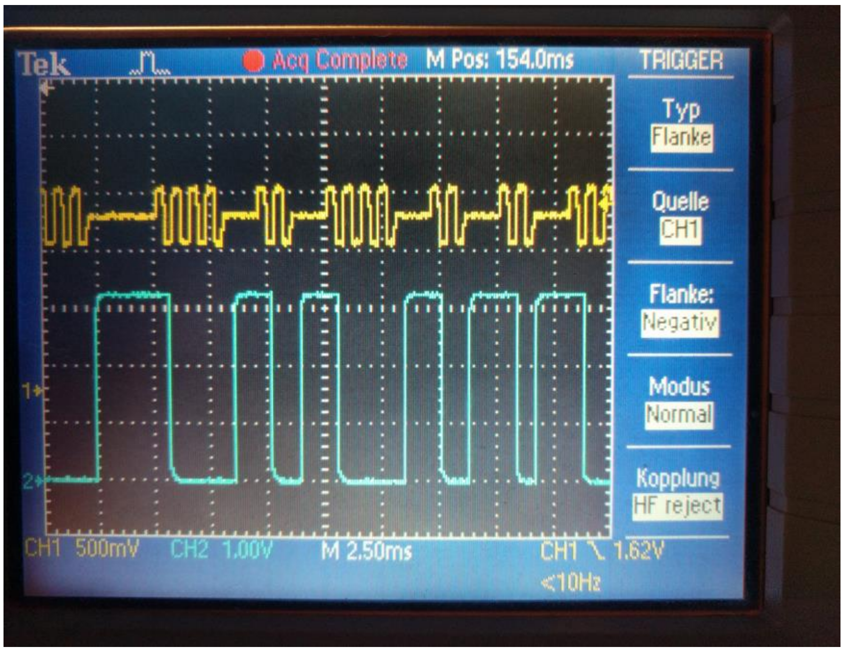
Here the remaining signal amplitude is only about 500mV…

Example Playback from Galaxy Nexus 5X Smartphone (via Binkenrocket WebGui)
 - Volume setting 100%:



Note that the waves look very different than the playback from the Laptop – not only in amplitude but also in signal shape …

Example Playback from Galaxy Nexus 5X Smartphone (via Binkenrocket WebGui)
 - Volume setting ca. 65%:



In all these examples, Scrolltext + Anims could be reconstructed and were displayed correctly on the Blinkenrocket.

Summary of most important changes

- changed transmission protocol / framing bytes / state-machine for frame recognition
- changed waveform generation in python / javascript
- added SPI debug functionality (Arduino sketch in subfolder firmware/utilities)

These changes were implemented in firmware files modem.cc / system.cc, Blinkenrocket.py & modem.js. Furthermore, some updated were implemented in the Web-gui, including activation of sliders for horizontal scrolling and an added feature for autoskipping animation patterns (number of repetitions can be set from 1-15 oder unlimited (=0).

Details see forks of Blinkenrocket-firmware and Blinkerocket-web-react at Github:

https://github.com/ChrisVeigl/blinkenrocket-firmware

https://github.com/ChrisVeigl/blinkenrocket-webedit-react

I have to admit that achievable data rate is much lower compared to the original transmission strategy - about a factor of 25 ;) the transmission of the standard texts + anime from Blinkenrocket.py takes about 12 seconds - but this should not be a big problem if the transmission is reliable ;)