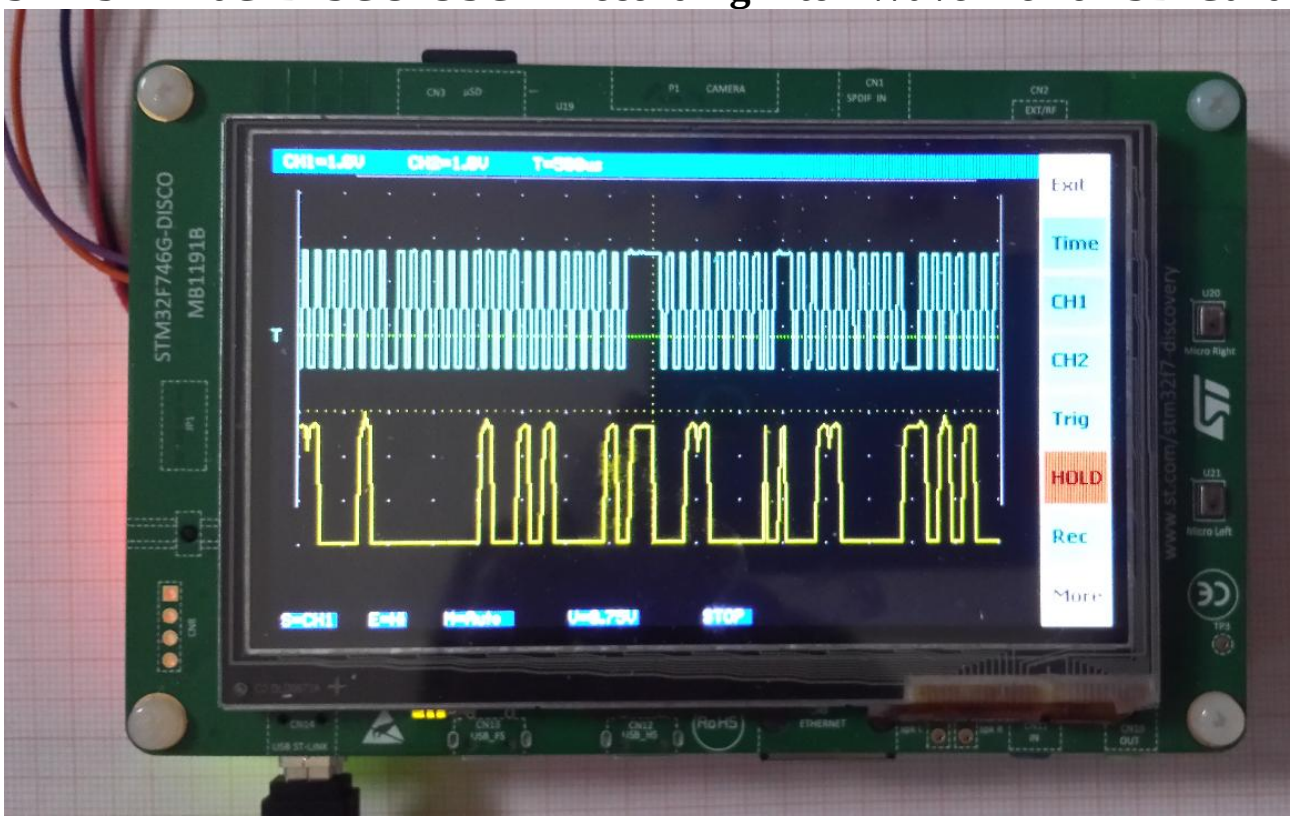
STM32F746G-DISCO-OSCI – recording into “Wave file” on SD-Card



STM32F746G-DISCO-OSCI – only an “Estimation-Iron” (free translation from “Schätzeisen” ;-) or maybe a little bit more?

The intention for this project:
- Getting in touch with the STM32F7xx device family
- Evaluation of STemWIN for GUI development

So I bought my first “STM32F746G-DISCO”...
...and nothing is better for evaluation and whirl up potential problems more, than making a meaningful project:

For the evaluation of the STemWin stuff I tried first to rebuild the “Demonstration”, that can be found in the relating folder of the CubeMX repository. Here for example the folder path for the current used Version V1.14.0:
“...../STM32Cube/Repository/STM32Cube_FW_F7_V1.14.0/Projects/STM32746G-Discovery/Demonstrations/StemWin”.
I prefer to use native Eclipse CDT, so I configured it up as project for this IDE.
This implementation is done with Eclipse Oxygen, CDT and the GNU MCU tool chain.
The „apps” meue structure of the demonstration is nice and a good base to hook in useful functionality (apps).

After getting it running, the idea was to adapt the famous “OSZI” project for the STM32F429I-DISCO from “Uwe” as such an app for the F7-DISCO.
The original source for the F4-OSZI can be found here:
http://mikrocontroller.bplaced.net/wordpress/?page_id=752
Or here:
https://github.com/noahchense/STM32F429i-Disco_OSC.

Another really useful gimmick from Uwe is his DDS frequency generator project, also for the F4-DISCO:
http://mikrocontroller.bplaced.net/wordpress/?page_id=762.

To have both as callable apps on the same board, would be nice (not jet - a future app). But exploring the schematic and sorting out the connectivity, an analog wave form output by the DACs (DDS) cannot be provided on the Arduino ports directly. Because the related ports (PA4 and PA5) are used somewhere else (DCMI_HSYNC and ULPI_CK).
But at least on the camera connector P1 – Pin 17 the "DCMI_HSYNC" can be fished out and this way the PA4 with DAC1 could be accessed. Of course that would need a very sharp solder iron or a 30 way FFC with a small breakout board...
If there is some interest, I would make a small breakout board (with additional amplifier)...!?

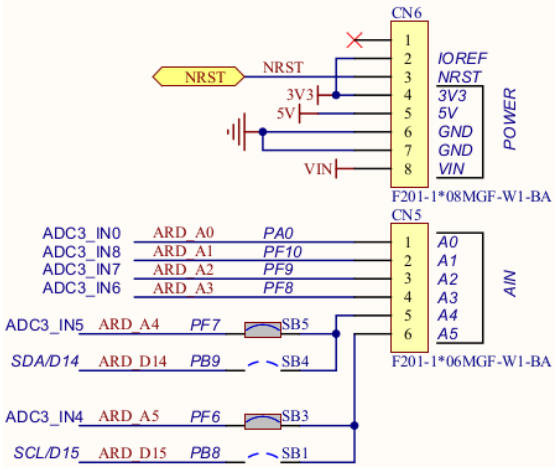
Another intention for this F7-OSCI was testing and evaluation of MEMS-IMUs (SPI and especially with the I2C). On the F4 the I2Cs is (sometimes, in harsh environment) a nightmare because of a silicon bug in the I2C filters (See the millions of posts about the „BUSY-Flag"). So I was keen on finding out, how the F7 will behave on the I2C. In a first "Quick-HACK" I've suppressed the ADC triggering and injected the I2C data from the IMU directly into the ADC sample buffer – so the OSCI graphs now showed the two selected "movement signals".
Maybe, that could become another helpful app for the future... ;-)
The data rate from these sensors is not so heavy compared to the ADCs – so displaying the "Basic 6" (3xAccel. + 3xGyro) in the OSCI should not become a huge problem.
And of course: "Recording" for the movements!

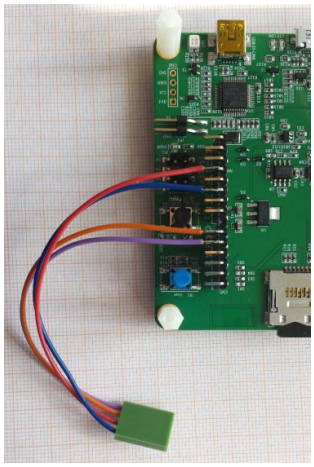
One intention also to have a closer look on the STM32F746G-DISCO was – beside the larger display – the SD Card socked on board!
Most valuable to get an oscilloscope recording functionality on a SD card. More about below...

Sadly this DISCO board has no jumper or resistor to separate the V-BAT pin. So one deficit must noted. The internal RTC cannot be supplied externally from a coin cell or accu. That's way it is not possible to provide a clock and calendar for the recording and the filesystem time. (An externally added RTC via I2C could fix that.)

Another limitation, only "A0" is connected to a port, reachable by ADC1 or ADC2. All other are only available on ADC3. But simultaneous sampling only with ADC1+2 or only with ADC2+3 is not possible. The workaround was to use all three ADCs in "Trible Mode". For more details about the used assignments see the description of the recording format below.

Connecting the both channels is easy on the Arduino Ports "A0" and "A1" - and of course we need a "GND":



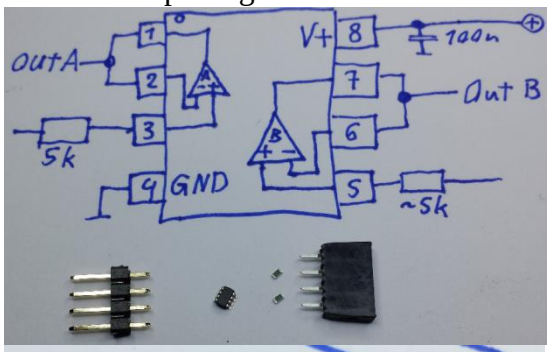


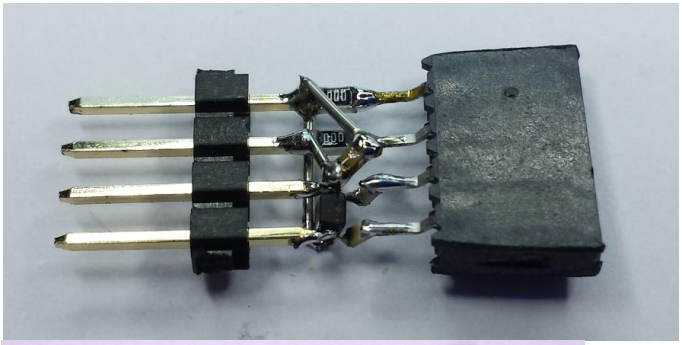
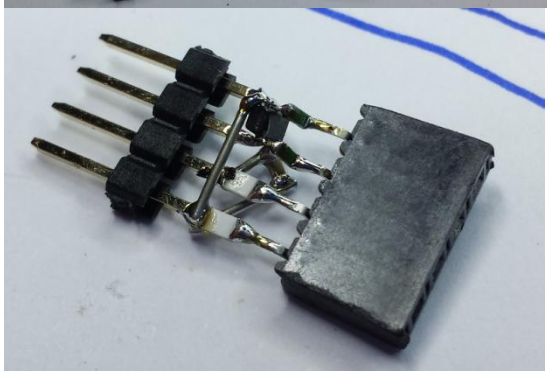
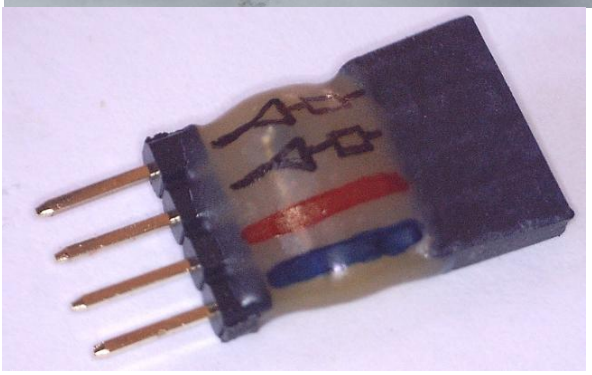
The “3V3” is basically not necessary, but I wired it out as well, to have a supply for an external OP-Amplifier. The ADCs draw a reasonably current from the measurement point, because of the switched internal sample and hold capacitor (see datasheet):

R_{AIN} (2)	External input impedance	See *Equation 1* for details	-	-	50	kΩ
R_{ADC} (2)(4)	Sampling switch resistance	-	-	-	6	kΩ
C_{ADC} (2)	Internal sample and hold capacitor	-	-	4	7	pF

Also on this page, the “Equation 1: RAIN max formula”, that show, how the conversion accuracy depend of the “external impedance”. But the question should not be the “external impedance” - the question should be the “Input impedance” of the OSCI!

So the idea was simply to try out a naked OP-Amp as a “Voltage follower”. I had some OPA2374 in SOT23-8 package in stock and – because no PCB on hand – made a “3D-SOLDERED”:



That tiny and quick “Hardware-Hack” – that works surprisingly better than first expected…!
The Rail-to-Rail capabilities of the OPA2374 are really good, as you can see later in the graphs.
And now having an “Input impedance” in the range of “Tera-Ohms” and “Input Bias Currents” of some “Pico-Amps” it should be also sufficient to measure “low power” signals, e.g. in solar- or other energy harvesting applications...

Some words about the discovered problems, which I got so far on the way for this project:

Chip related within the STM32F746G device:

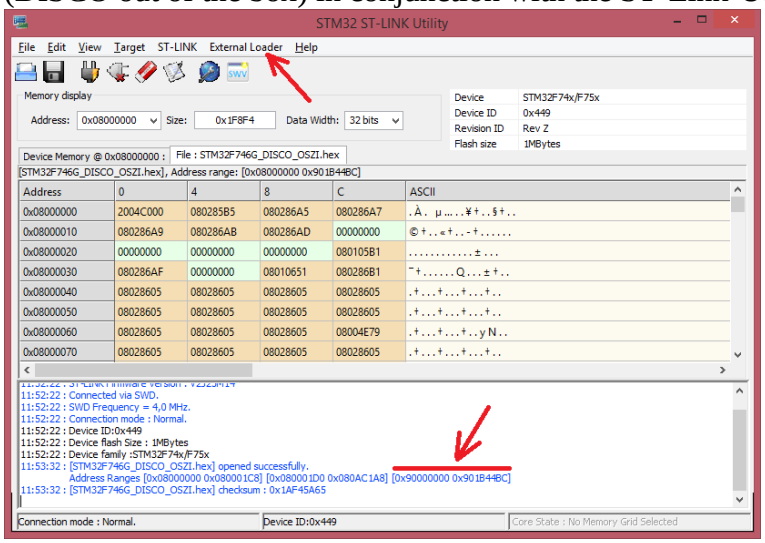
- ADC not triggered from a timer event (neither the used TIM4, nor some tried out other timers).
The timer was running, because it triggered his interrupt – just activated for a test.
ADC-Triple including DMA transfer worked also, if triggered by software flag from the timer interrupt.
Finally – also after losing also some hair – found the answer here: (see "mollyandchloe1")
https://community.st.com/s/question/0D50X00009XkY6vSAF/dacadc-conversion-timer-triggering

- The STM32F746 has another silicon bug, that can cause frustration and head-(to-table)-banging while debugging. I was wondering that I got trapped always into an interrupt handler, on single debug steps (OpenODC+ST-Link).
Then I realized the relating OpenOCD info output:
>> Warn : Silicon bug: single stepping will enter pending exception handler! <<

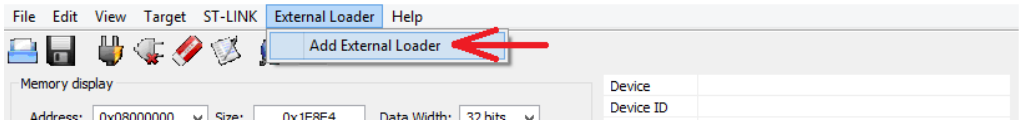
An internet research brought up this link, describing exactly the issue:
https://community.st.com/s/question/0D50X0000A4pIcQSQU/stm32f746-cortexm7-silicon-bug-singlestep-lands-in-interrupt-handler
The first link inside lead to a more detailed description:
http://www.keil.com/support/docs/3778.htm
Also in this thread, the hint to use „J-Link On-board":
>> (Piranha):
SEGGER have implemented a workaround for this hardware core bug in their J-Link debuggers.
On-board ST-LINK can be converted to J-Link OB.
https://www.segger.com/products/debug-probes/j-link/models/other-j-links/st-link-on-board/
<<

AND - a welcome surprise by trying out "SEGGERs J-Link OB" was the fact, that it also program the QSPI-Flash device in one rush. Into that memory the graphics content (icons, images, …) is mapped by the linker. The ST-Link in conjunction with OpenODC is not able to do it in this comfortable manner.
(Maybe it could be possible, but I had no closer look to find it out up to now. Replies are heavily welcome, if someone knows more...)

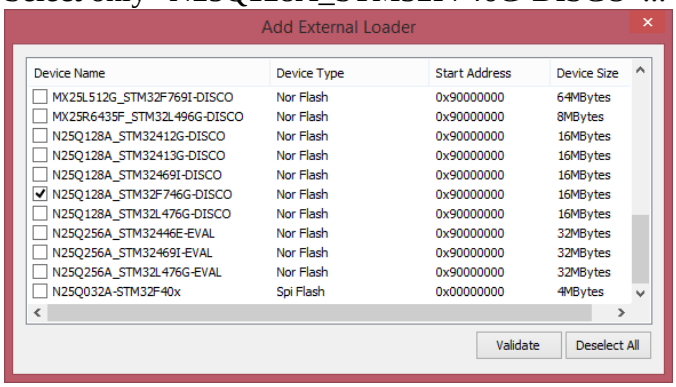
By the way, just to download a binary or hex (no debugging) work fine with the original ST-Link (DISCO out of the box) in conjunction with the ST-Link-Utility (Windows):



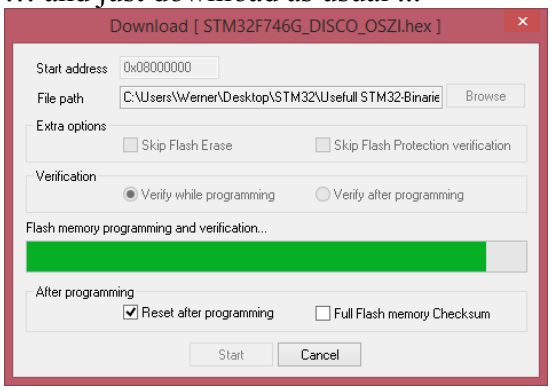
The important point hereby is, to select the relating “External Loader”, to get the QSPI-Flash also flashed. Otherwise – even the program will run – the Graphics show just mess:



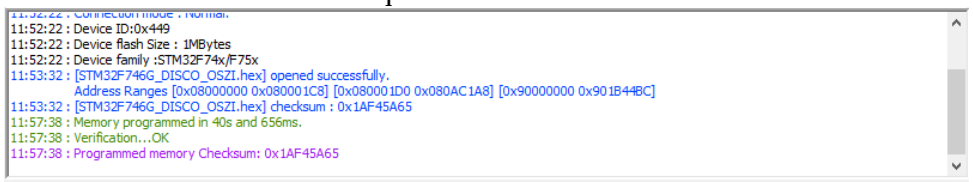
Select only “N25Q128A_STM32F746G-DISCO”...



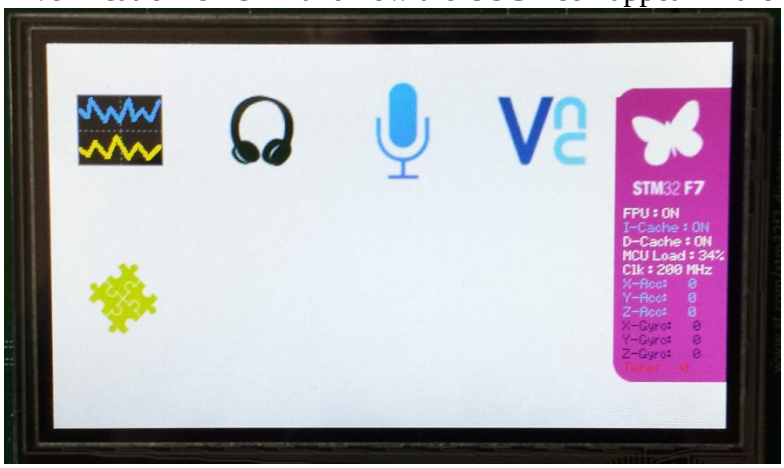
… and just download as usual ...



That should take about three quarters of a minute...



If verification is “OK” and now the OSCI icon appear in the menu, everything went well:



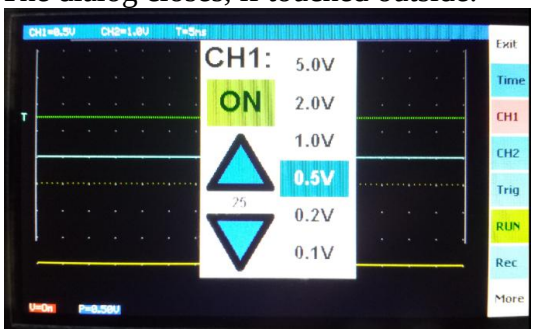
The OSCI menu structure:

On the right side is a column of 8 permanent buttons, calling the main functions of the OSCI:

Exit: Leaf the app and turn back to the start screen above.

Time: Open the time selection dialog. The currently active setting is colored in blue.
To select another one, just touch on the time number you wish. The dialog closes automatically after touching.
For no change, touch the still active setting, touch outside the dialog or touch the "Time" button again.

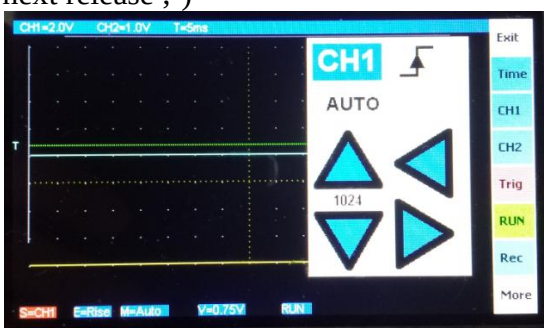
CH1 + CH2: Scale selection is done by touching the desired voltage. The active one colored in blue.
The "ON" – "OFF" button toggles on touching and change the color to red, if off.
With the arrows, the channel offset on the screen can be moved up and down.
The dialog closes, if touched outside.



Trig: Here the trigger sensitive channel can selected, the trigger edge and the mode.
Trigger edge cycles through the possibilities "Rising", "Falling" and "Both":

.

Trigger Mode cycles through "Normal" (the normal trigger mode), "AUTO" (the free running without triggering) and "Single" (taking only one shot).
The trigger level can be moved with the up and down arrows. And the edge position on the screen with the left and right arrows. Closing also by touching outside.
This functionality is currently "UNDER CONSTRUCTION" and will work hopefully in the next release ;-)



RUN/HOLD/WAIT: That is a toggle button, that hold (red) or release (green) the ADC data acquisition. If waiting for a trigger it is yellow. This button also is used to get the next "Single" shot.
The interaction/collision with the recording is not designed or explored up to now (maybe writing additional infos into V-Ref's track 1)... More details in future releases...

Rec: This is also a toggle button, starting and stopping the recording. If the recording starts, a new wave file with the next number is created. The button color changes to red. The native ADC data streamed into the wave file, as long as the button is touched again.

More: That switches to the second button page with the additional (not so often needed) stuff.
It covers some of the menu items from Uwe's former implementation for the F4:
"Back" - switch back to the first button page.
"Curs" - Showing and moving a cursor for one of the channels or time on the screen.
"FFT" - showing FFT transformation for one of the channels.
"Save" - Write a screenshot as BMP picture to SD card.
"Send" - Write ADC data to a serial connection. (USART on ST-Link, USART on Arduino port or maybe in future a VCP on a USB-Device).
"Vers" - Show the version string in the lower line.
"Help" - Show the used pin connections (ADC, USART).
"Conf" - Maybe for additional configurations (File size, display options,…)
Some of these items will change and maybe supplied by additional dialogs (e.g. Vers and Help merged together in a Info dialog).

TODOs and future ideas:

- The design of the GUI, now with permanent available buttons (instead of the single arrow GUI for the F4), is a little bit a challenge. Sorting out the functionality and designing the GUI for usability and a minimum of needed touch interaction is still under construction. And it shows my current state of learning and exploring the "STemWIN". So some "trial and error" hacks can be found in the source code ;-)
  Clean-up and more efficient message handling are in progress... I'm still learning...
- For ADC acquisition above 25us/Div. it should be possible to increase the sample time to 15 or more ADC clock cycles (increases accuracy, decreases noise).
- Config file on SD card, which conserve the last used settings and states.
- Configurable file size limitation for the recording, automatically start the next one or just stop the recording at size or time limit.
- Using the rest of the "audio buffer" for better ADC/SDIO decoupling (switching DMA mem address to next free one, ring buffer like), four more complete buffers should minimize SKIPs dramatically.
- Using both, SD-Card and USBH+Stick side on side!?
- Analog-Input-Shield with protection, range switching (digital controlled by Arduino Dx) and OP-Amps on board.

If someone has the intention to join this project - **Welcome!!!**

Recording:

In the “Demonstration” the file system for the audio-player and the audio-recorder run on an external USB-Stick via the USB-Host-Driver.

So the first challenge was to get the SD card transfer working for that purpose. I used in the past, SD cards for recording in some projects – but always only interfaced with “SPI”. So now I wont to do the venture for the upgrade to “SDIO”. Started with the “sd_diskio_dma_rtos_template.c” from the related CubeMX “*/Middlewares/Third_Party/FatFs/src/drivers” folder. It compiled and linked without problems, but running fired a “Hard_Fault”. Then I tried the “sd_diskio_dma_template.c” driver, same result. Finally the simplest “sd_diskio_template.c” gave me a first running access to the SD card.
It was a moment of melting into the chair if Madonna’s Evita sounds out of the headphones, streamed from the SD card.

Even if the audio player work with this “polling” driver on the SD card, the audio-recorder or just to try out, writing some OSCI data, always failed with a transmit FIFO underrun (HAL_SD_ERROR_TX_UNDERRUN). The CPU cannot serve the SDIO fast enough, if it has to do the “OSCI work” also. So the conclusion: DMA is a must have!

Switching back to the “sd_diskio_dma_rtos_template.c” driver and “now” providing the four additional needed interrupt handlers (BSP_SDMMC_DMA_Tx_IRQHandler, BSP_SDMMC_DMA_Rx_IRQHandler, BSP_SDMMC_IRQHandler and the EXTI15_10_IRQHandler for the SD_DETECT_PIN) – and oh wonder – the Hard_Faults were gone...
Also the SDIO FIFO unterruns.

Current state 19.03.2019:

Driver	Osci Recorder	Audio Player	Audio Recorder
sd_diskio.c	HAL_SD_ERROR_TX_UNDERRUN	OK	No Fault / No Recording
sd_diskio_dma.c	No Fault / No Recording	No Fault / No files found	No Fault / No Recording
sd_diskio_dma_rtos.c	OK	Hard_Fault	Hard_Fault

For one of the next steps, debugging must be done on this issue (Checking DMA and IRQ resources, priorities,...).

My primary focus is the OSCI-Recording, therefore the “sd_diskio_dma_rtos.c” driver is in use. But I kept the other drivers and the „Audio P/R“ apps in the project for further debugging...
Unused source parts are simply excluded from the build process.

Now the big question was, how to write the OSCI data to the disk?

Why not trying the WAVE format, before developing an own exotic one?!
Byte order would fit, value format (12 bit right aligned) would fit and DMA sequential transfer from the three ADCs to memory would also fit for direct block transfers into the wave file format.
But ONLY IF it would be possible to handle three tracks, instead of one (mono) or two (stereo)!
And doing the transfers only and direct with DMA – from ADCs to MEM, and from MEM further to SDIO – without CPU copy processing, would be fast enough for high speed recording.

So I simply tried it out! Cloned the „audio_recorder_app.*“ files into „osci_recorder_app.*“ files and modified the header generation for three tracks in function:

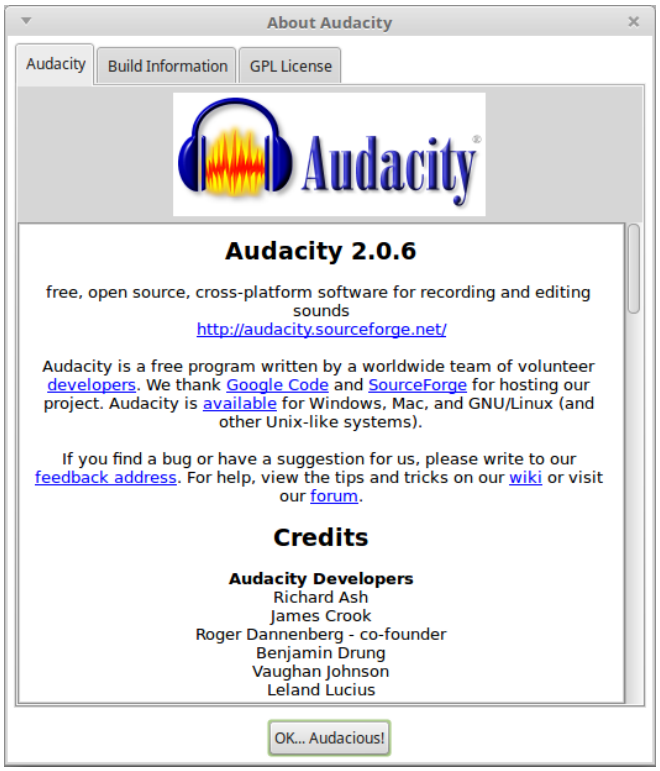
```
static uint32_t WavProcess_EncInit(uint32_t Freq, uint8_t *pHeader)
```

So generating and writing out such a “special” wave file was more or less trivial...

Viewing the OSCI data:

Using the wave format, of course, the idea was to use an “existing” audio processing program for a first access. Many of them will fail, because of the oblique sampling frequencies and the odd amount of tracks.

But “Audacity” does the job:



ATTENTION!!! Watch your EARS!!!

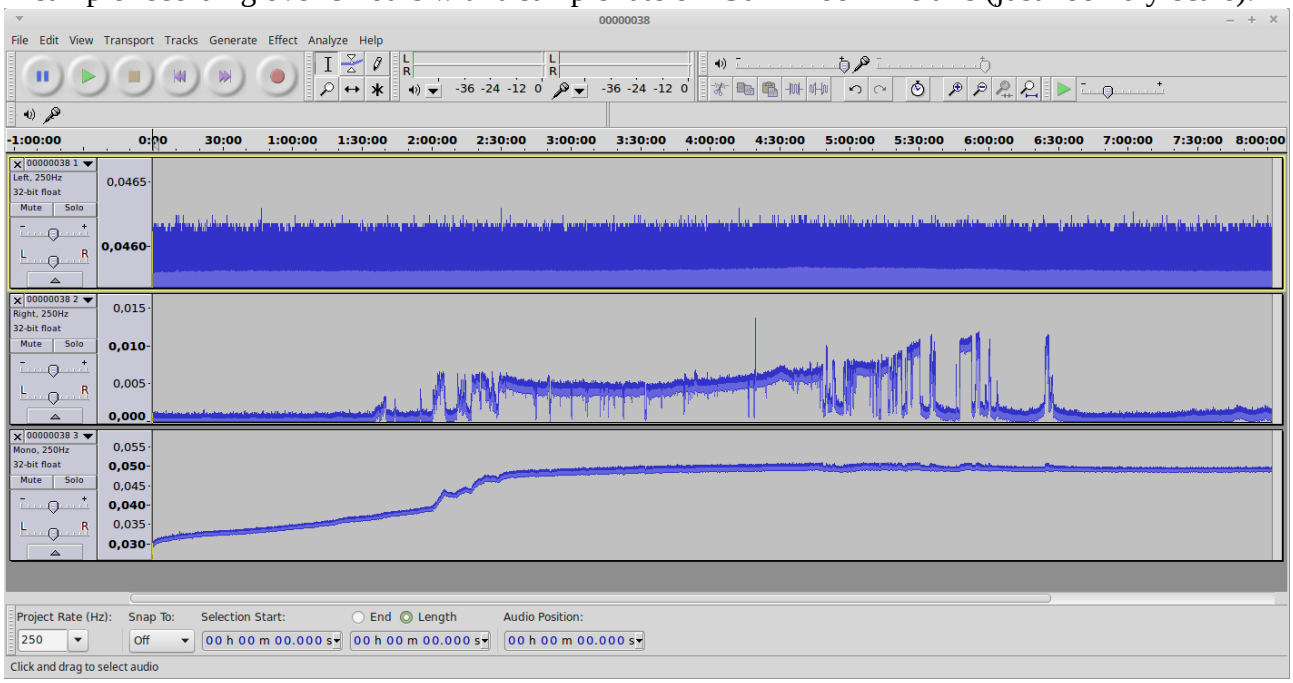
With the wave format it is possible to “listen” to the “signals”.

BUT – BE CAREFULL WITH THE LOUDNESS!!!
Like listen to music, don't blow out your ears!!!

Using only 1/8 of the positive half (1/16 overall) the loudness in a OSCI record should not be dangerous or hazardous for the ears, compared to music files. Never the less, start with moderate loudness and increase it step by step. A “I2C” for example is like a “drill in the brain”, if listened too loud!!! And use Autacity's processing functions, especially “Normalize”, with care and – so ever prayed – on your own risk!

Now – just press “Play” and dive into the “sound of electronics”...

It's TOP for a first view of the recorded data! And allow quick loading, zooming and time scrolling. A sample recording over 8 hours with a sample rate of 250 Hz look like this (just zoomd y-scale):

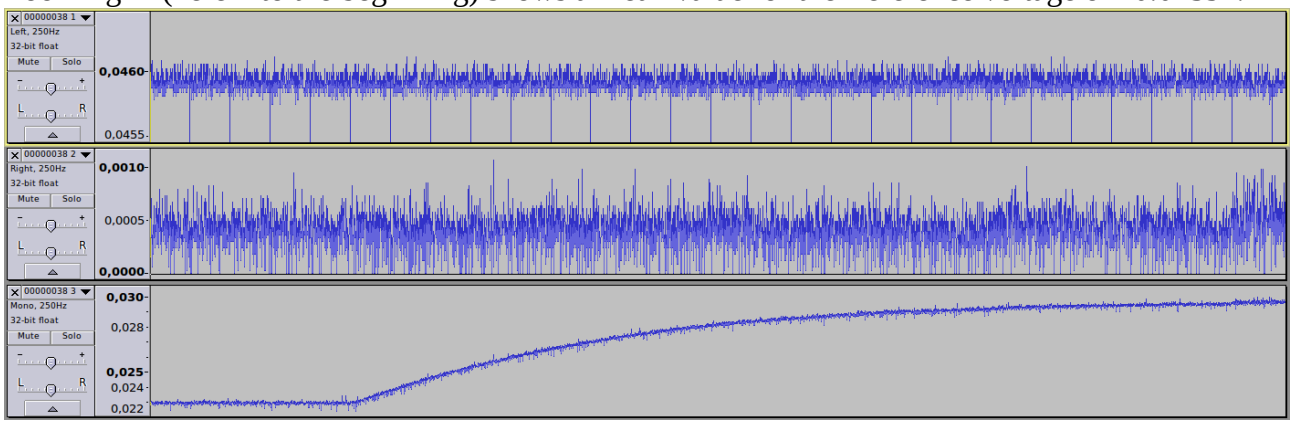


Filename here is “00000038.wav”.
Track 1 contains the sampled Ref-Voltage from ADC1.

Track 2 contains the OSCI channel 1, which is Arduino “A0” on port “PA0”, sampled by ADC2.
Track 3 contains the OSCI channel 2, which is Arduino “A1” on port “PF10”, sampled by ADC3.

Why this assignment?

No analog channel – except “A0” – can be routed to ADC1 or ADC2. But a “Dual Simultaneous Mode” only with ADC1+3 or with 2+3 is not possible. So the “Triple Simultaneous Mode” is the trick!
And to let ADC1 do something useful, the internal Ref-Voltage is sampled, maybe it can be used for a more precise post-processing (e.g. for correction of the other channels).
Zooming in (here into the beginning) shows a mean value for the Reference voltage of “0.0459”:



What does it tell?
Autacity show the recording normalized in the range from -1 to +1.
The recording data type is 16 bit. That's a “signed int” in the wave format, representing values from -32768 to +32767. ADC conversion is 12 bit unsigned, representing a voltage range from 0V to 3.3V (U) by codes from 0 to 4095 (x). That is simply the 8th part of the positive half in the wave data type and therefore shown in Autacity in the range from 0 to 0.125.

The formula to get the sampled voltage value:

$$U = 3.3V * (32767 / 4095) * x$$
$$= 3.3V * 8 * x$$
$$= 26.4V * x$$

For our V-Ref we get:

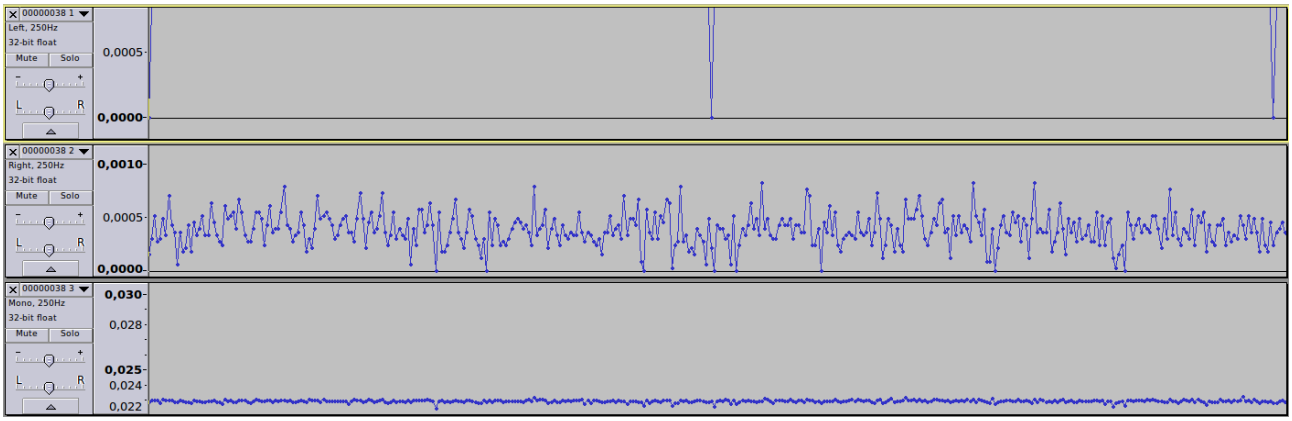
$$U = 26.4V * 0.0459 = 1.212V$$

Which match pretty well the data sheet specification (even if the sampling time is much shorter than the recommended 10us):

Table 71. internal reference voltage

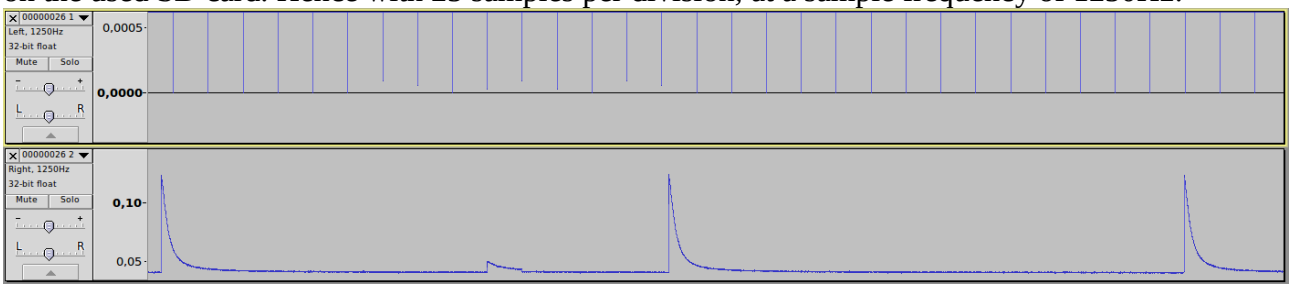
Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{REFINT}	Internal reference voltage	–40 °C < T_A < +105 °C	1.18	1.21	1.24	V
$T_{S\_vrefint}$(1)	ADC sampling time when reading the internal reference voltage	-	10	-	-	µs
V_{RERINT_s}(2)	Internal reference voltage spread over the temperature range	V_{DD} = 3V ± 10mV	-	3	5	mV

BUT – what are the ZEROs every 200 samples in the V-Ref track?



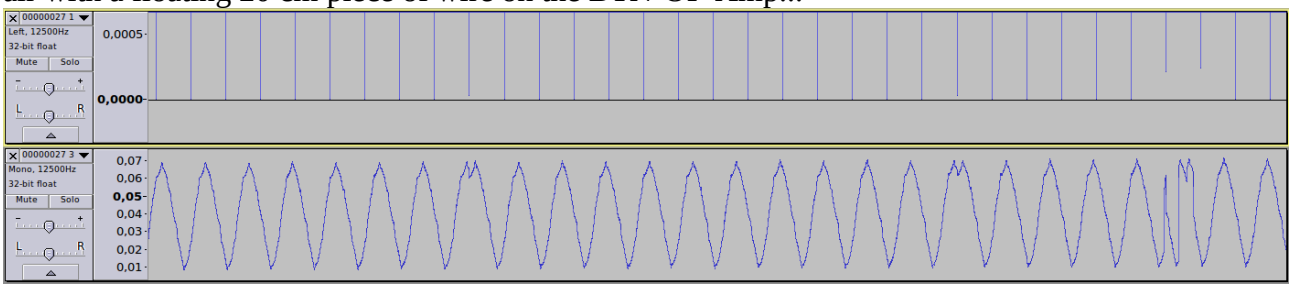
That's not a bug – it's a feature!

I've overridden every first value of a ADC-DMA-(Half)-Transfer-Buffer by a counter for “Lost” sampling buffers. If the file system management takes too long (e.g. cluster or chain allocation), so the recording is not fast enough to handle all buffers just in time.
The value “Zero” is a good one – meaning no skipped buffers before.
The boarder where the skips appear sporadically is at a time scale of 20ms/div, depending heavily on the used SD card. Hence with 25 samples per division, at a sample frequency of 1250Hz:



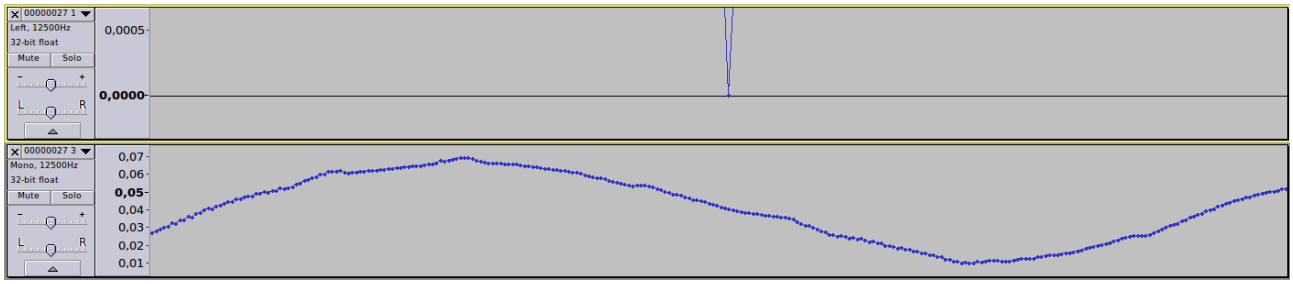
Here for example, the “five“ pulses in track 2 should be equidistant and all with the same shape.
But because of the skips one is canceled out completely and on appear as a fragment.
Later in the Matlab/Octave data processing, I'll show, how this “leaks” can be handled with “NaN”s to reconstruct a real time scale.

This effect can be better observed on a sine wave. Here for example a “50Hz” noise, captured from air with a floating 20 cm piece of wire on the BTN-OP-Amp...

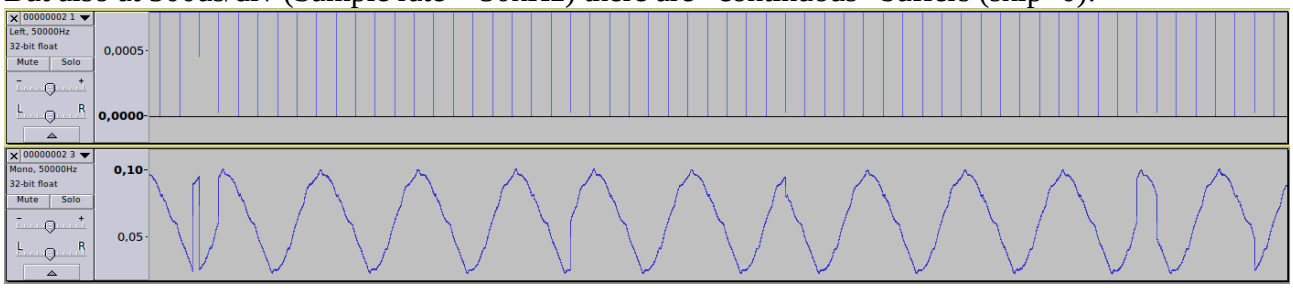


Easy to see where the wave form is corrupted…

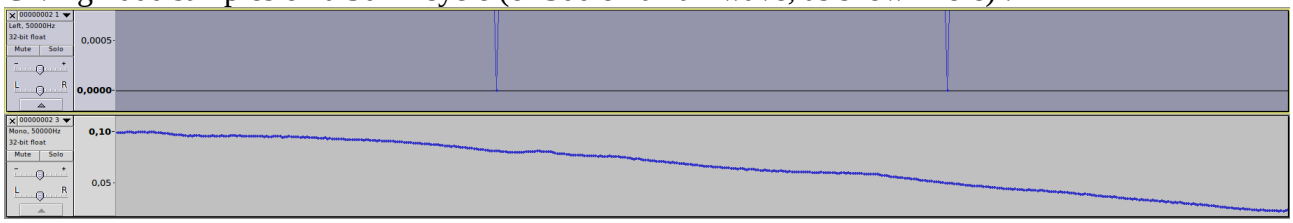
But there exist also some continuous recording phases (skip=0), in which e.g. a 50Hz cycle can be shown completely with 250 samples (Time scale setting = 2ms/div):



At higher sampling frequencies the leaks occur more often and they are longer!
But also at 500us/div (Sample rate = 50kHz) there are “continuous” buffers (skip=0):



Giving 1000 samples on a 50Hz cycle (or 500 on a half wave, as shown here) :



Lower timescales do not work (maybe not up to now)! Could be, that the SDIO and the ADCs running in conflict, because they both use the DMA2. That's also a topic for future development and exploration – I'm sure, that something more can be wangled out here...

Working with the OSCI data:

To “work” with the OSCI data, I’ve tried out two other “Standard” programs for this purpose – and “YES”:

“MATLAB” and “Octave” – both can read-in this “exotic” wave files with their build-in “**audio***xyz*” functions:

```
>> audioinfo ('Test - 50Hz-Noise - 50k-SPS.wav')
ans =

  scalar structure containing the fields:

    Filename =  Test - 50Hz-Noise - 50k-SPS.wav
    CompressionMethod =
    NumChannels =  3
    SampleRate =  50000
    TotalSamples =  366000
    Duration =  7.3200
    BitsPerSample =  16
    BitRate = -1
    Title =
    Artist =
    Comment =
```

Loading the file is simply done by:

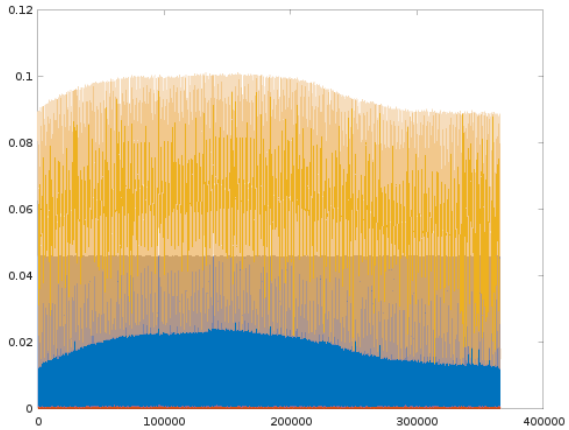
```
>> [Y, FS] = audioread ('Test - 50Hz-Noise - 50k-SPS.wav');
```

The record is now in the workspace:

Bezeichner	Klasse	Dimension	Wert	Attribut
FS	double	1x1	50000	
Y	double	366000x3	[0, 3.0518e-05, 0.080566; 0....	
ans	struct	1x1	...	

To get a first view:

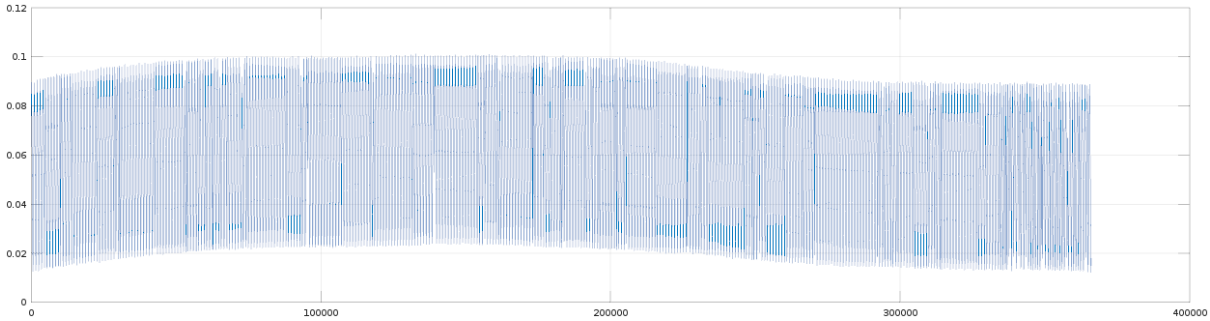
```
>> plot(Y);
```



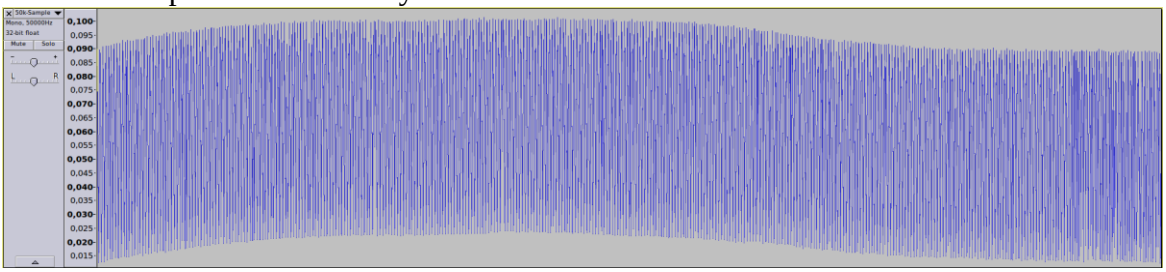
Blue: ADC1=V-Ref / Red: ADC2=Osci-CH1 / Yellow: ADC3=Osci-CH2

Or to show “Osci-CH2” alone:

```
>> plot(Y(:,3));
```



A direct comparison to Autacity:



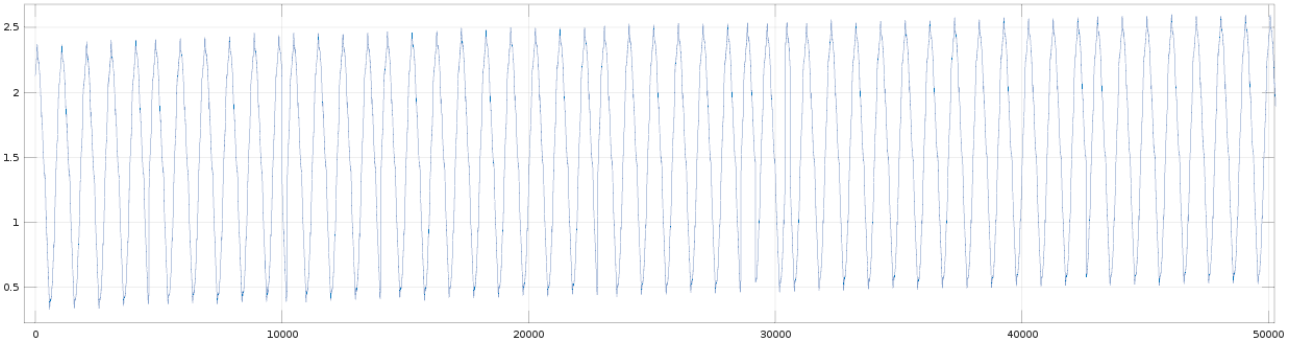
The representation is the same like in Autacity within a range from [-1, +1] or for our 12 bits from [0 to 0.125]. But her we now have the possibility of simple data manipulation...

```
>> ch2_buffs = Y(:,3)*26.4;
```

...to get the real messured voltage and...

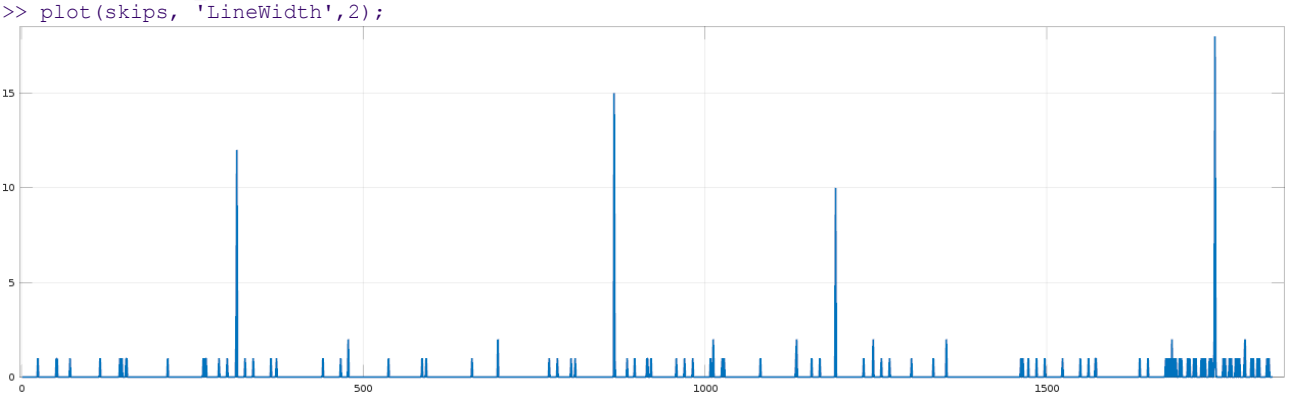
```
>> plot(ch2_buffs(1:50000));
```

...to show only the first 50 thousand samples for better details:



But that is NOT a real second of recording time – because we got some “SKIPs”!!!
To find out the SKIPs, we need every first value of a DMA transfer buffer – that one, that is overwritten in the V_Ref track. A transfer buffer contains 200 values in each track:

```
>> vref_buffs = Y(:,1);
>> skips = vref_buffs(1:200:end)*32768;
>> plot(skips, 'LineWidth',2);
```



That looks not to bad for such a sampling speed. Most time we have only one buffer lost, but in the worst case up to 18 on a piece.

The multiplication with “32768” normalize the “range” back to whole “integers”.
Over the whole recording time we lost:

```
>> losts = sum(skips)
losts =  168
```

And got out:

```
>> recs = length(vref)/200
recs =  1830
```

The whole recording time was therefore:

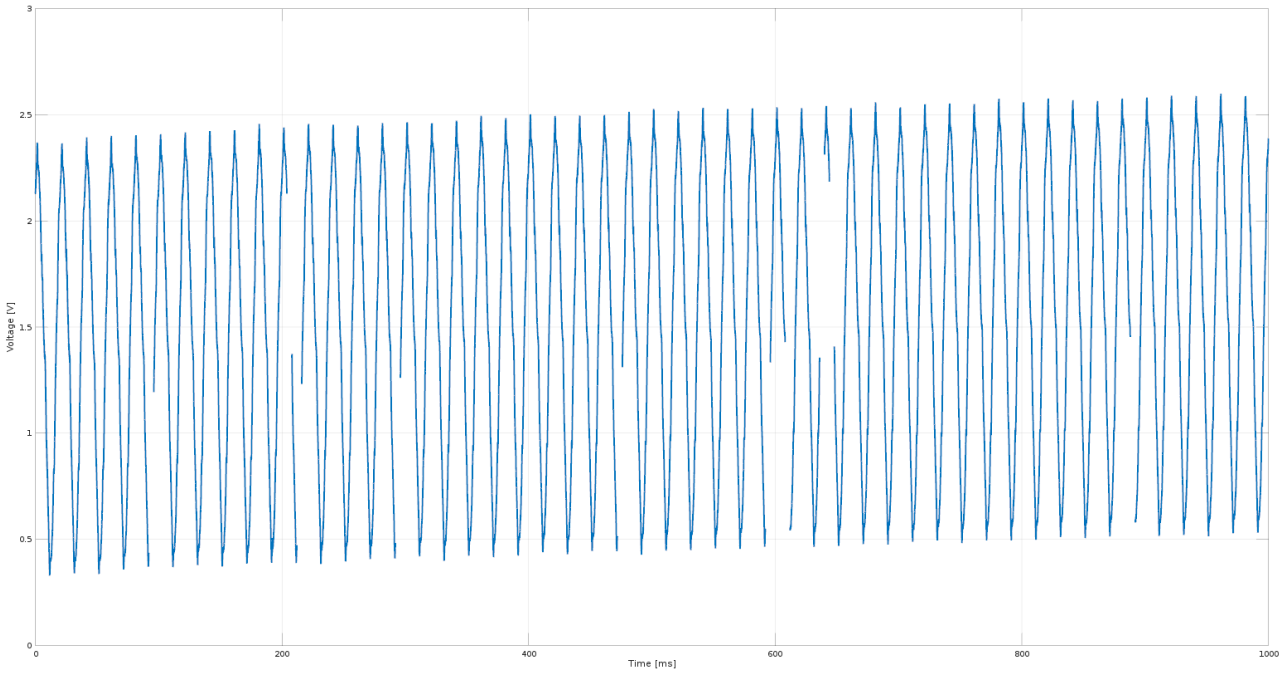
```
>> t_total = (recs+losts)*200/FS
t_total =  7.9920
```

And the percentage of losts in this example:

```
>> lost_pers = losts/(recs+losts)*100
lost_pers =  8.4084
```

The outcome here of course depend in a high manner also on the used SD card speed and card size!!!

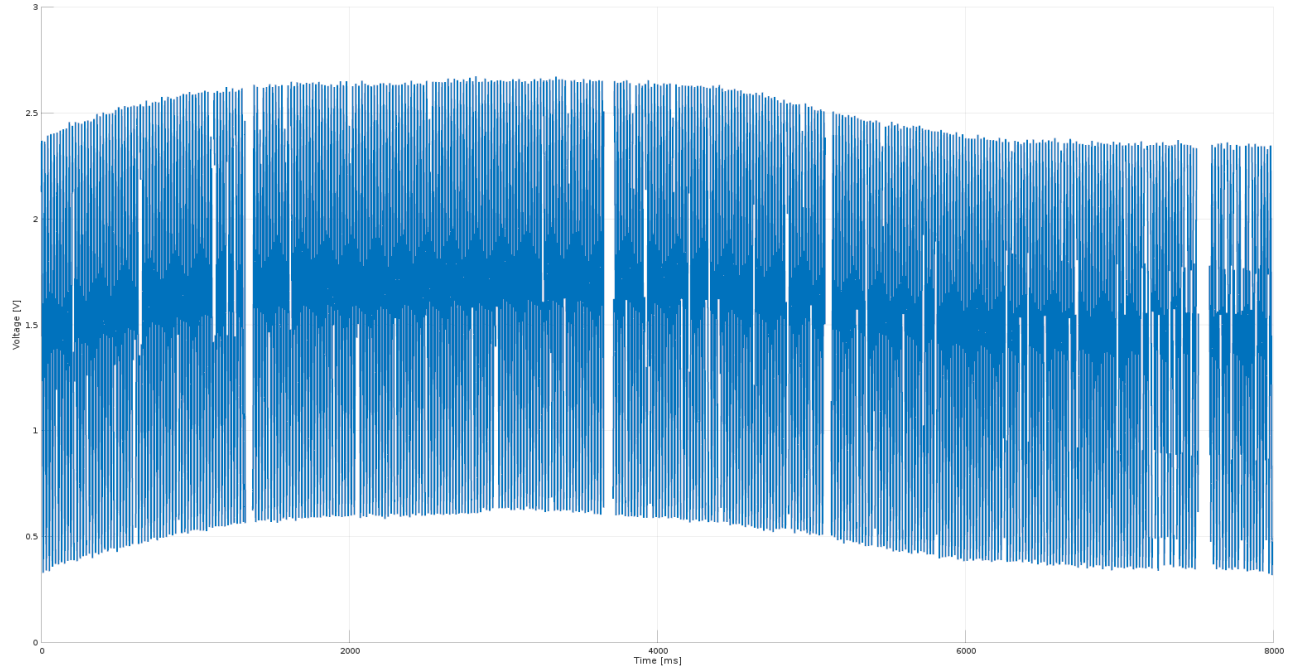
To get the real timing we need to insert the “losts” virtually as NaNs in a whole array of the right length. See script file “osci.m” how this could be done. It run in Matlab as well as in Octave.
Now the result for 1 second or 50000 samples show exact the 50 stable oscillations of the European power supply:



Even with the leaks, the overall signal structure can be seen.

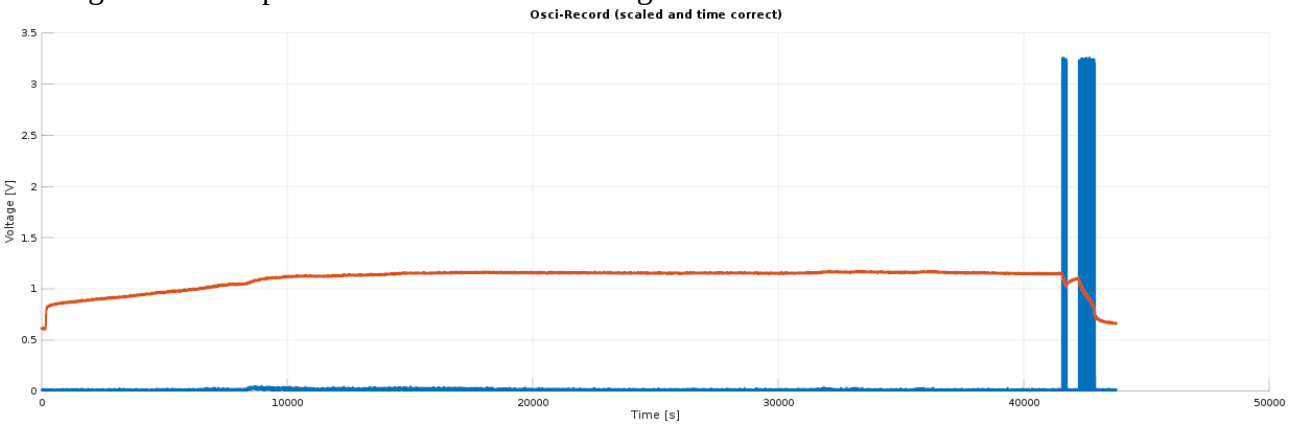
If not, two consecutive transfer buffers can be seen as a “screen shot” from the OSCI.
So – at least – having a chain of “screen shots” should allow some analysis, that is hard to do without recording…

The whole recorded example:

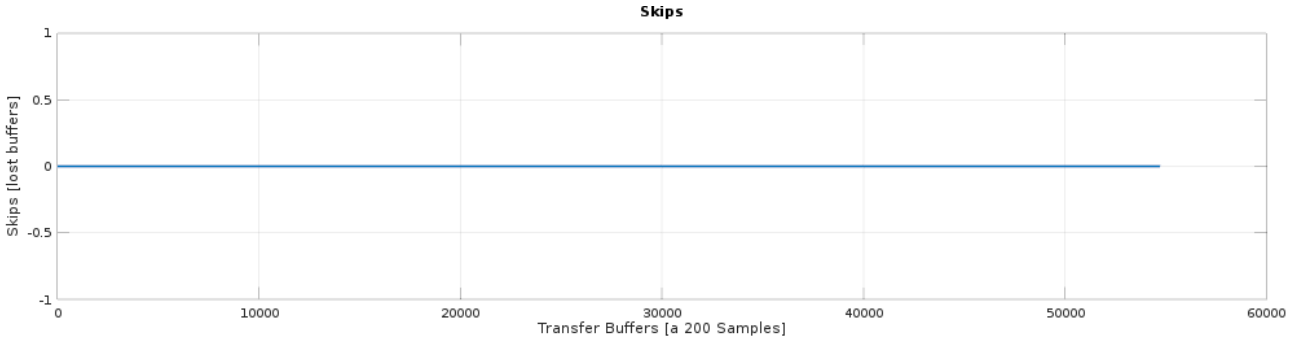


Compare especially the larger gaps with the position of the higher “Skip” peaks.

A “long term” example: Record of a solar flash-light with NiMH accu over 12 hours with 250Hz:

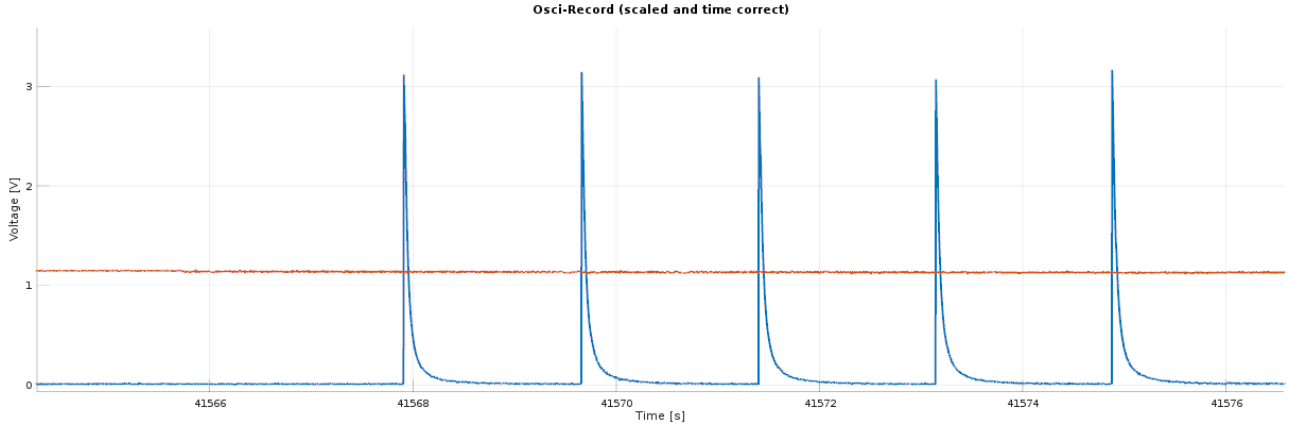


Blue: ADC2=Osci-CH1=Flash-Light output (photo resistor) / Red: ADC3=Osci-CH2=Accu-Voltage

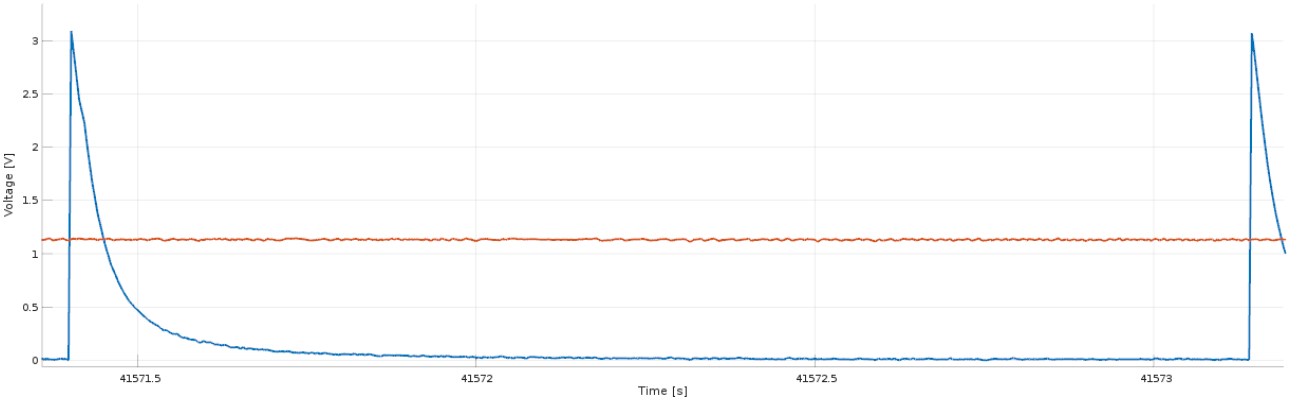


No skips!

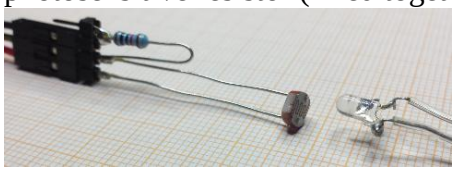
Zoomed into the five first flashes:



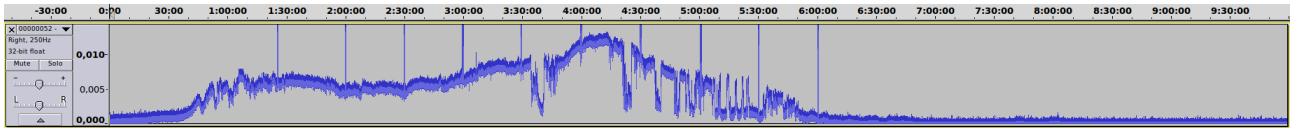
And into the third flash:



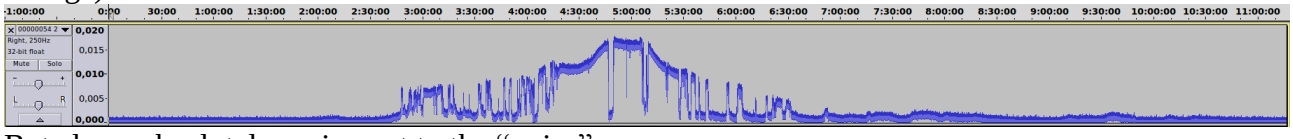
The blue graph in the first picture of this example show some “noise” (beginning around 8000s)! But what could be the source? That's the light output of the Flash-LEDs, measured with a photosensitive resistor (fixed together and covered with more than one layer of black tape):



So my first conclusion was a leakage in the solar harvesting circuit, that let the LEDs “glow” a little bit. So in one of the following days I made an experiment by covering the solar cell every half hour for 30 seconds:



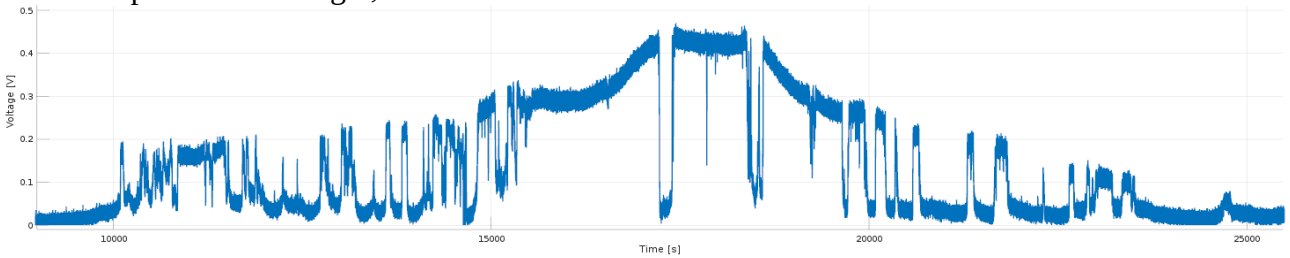
But that showed no effect to the “noise”. Another record one day later, but now disconnecting the whole blinking circuit for some time and finally only the LEDs also some times (10 min per change):



But also – absolutely no impact to the “noise”.

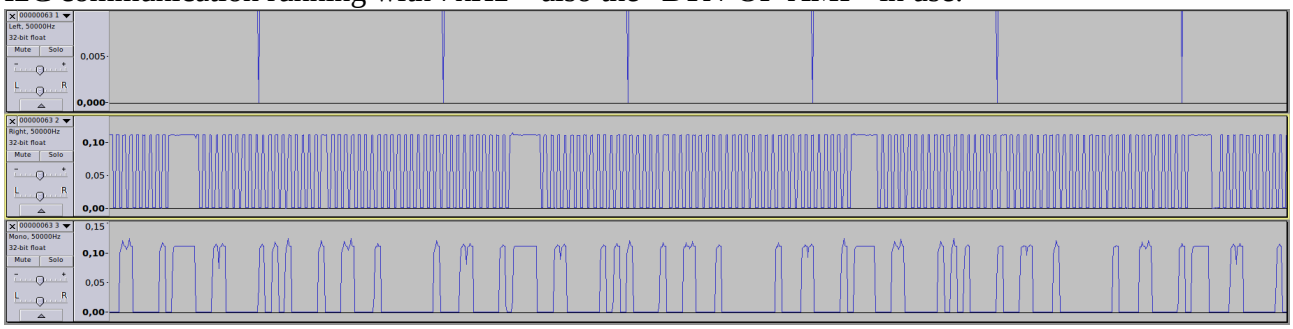
The final result: It was the “stray light“, that creeped into the small opening and the transparent isolation and this way reached the photo resistor:

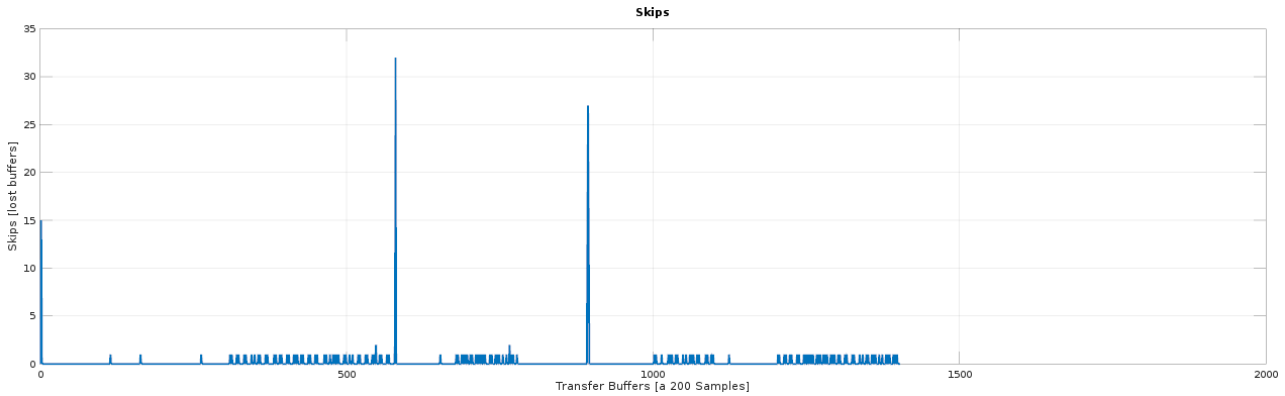
Now the “noise profile” makes sense! I had the assembly on the window (ESE direction) in my kitchen. The first peak (at 10500) was my lamp, while making coffee and preparing breakfast. The “main shape” is the sun light, which is scattered around in the kitchen from the different surfaces.



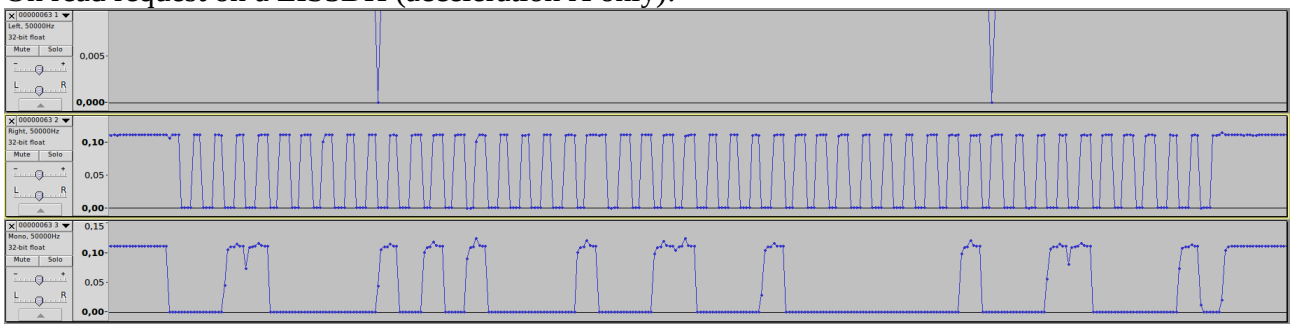
The drops in the “main shape” are simply the clouts, that passing the day over the sky. A really cloudy day in the morning and in the afternoon, two big clouts around high noon. The day before, the clouds come at the afternoon. I had to stop the 30s-shut-offs exercise, because at the last one the clouts closed up completely.

One example more:
I2C communication running with 7kHz – also the “BTN-OP-AMP” in use:

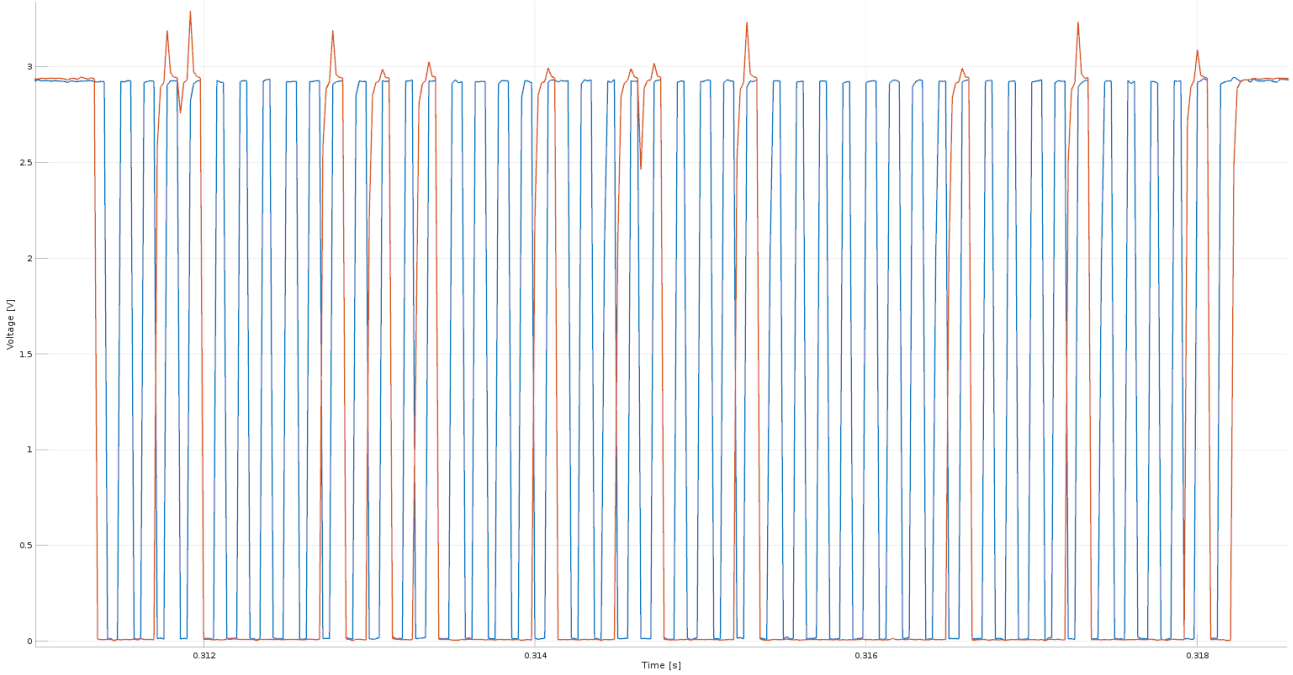




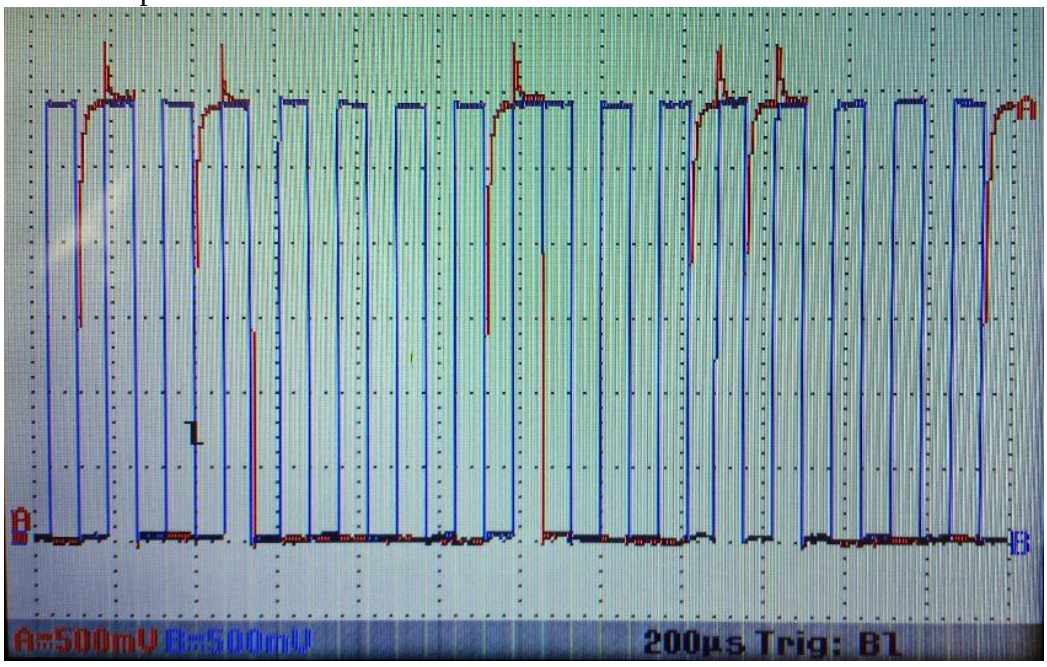
On read request on a LIS3DH (acceleration X only):



The same as an Octave graphic and in volts:

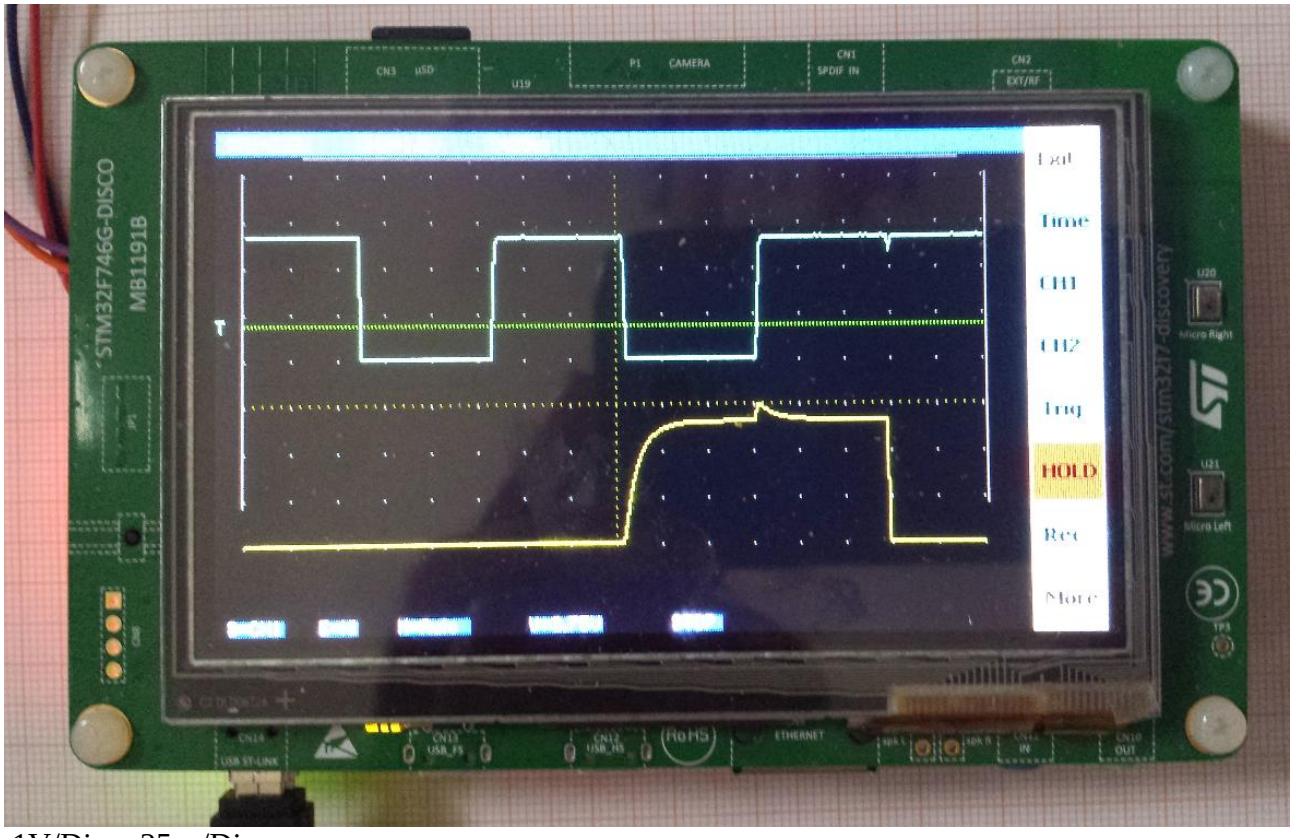


And here the same partial signal on the screen of a parallel connected “real” oscilloscope, for a direct comparison:



And how it looks like on the „Estimation-Iron“:

1V/Div – 500us/Div



1V/Div – 25us/Div

So far, so good…
Werner 21.03.2019