Redpitaya ADC characterization,

E. Rubiola, J.-M Friedt, November 8, 2022

The objective of the laboratory session is to get familiar with the radiofrequency grade Analog to Digital Converters (ADC) on the 14-bit Redpitaya. These ADCs are LTC2145 fed by a differential signal converted from the unbalanced signal provided on each SMA connector by a LTC6403 amplifier (Fig. 1).

The outline of the laboratory session is to first develop the appropriate signal processing functions for displaying a signal (e.g. a sine wave) in the time domain (validating that the signals are properly read from a file), in the frequency domain and in the probability domain (histogram), and making sure that energy conservation is verified in all cases. Once these processing steps are validated on a known signal, the same processing steps are applied to noise (loading the ADC with a 50 Ω load) to assess the Effective Number Of Bits (ENOB) of the radiofrequency grade ADC.

Check on the datasheet the gain-bandwidth product of the amplifier and the sample-and-hold bandwidth of the analog to digital converter. How do they compare with the 125 MHz sampling rate of the Redpitaya?

1 Connecting to the Redpitaya

The IP address of the Redpitaya is 192.168.2.200. The host computer IP address is 192.168.2.1 The login and password are `root` and `root`. Either connect to the Redpitaya using `minicom` and transfer files through NFS as the home directory on the host computer can be exported to the Redpitaya, or use the secure shell commands `ssh` and `scp` to interact with the embedded system.

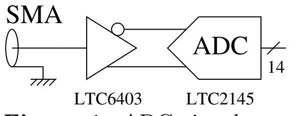


Figure 1: ADC signal conditioning

2 Data acquisition

Two bitstreams are provided in the `/home/jmfriedt/bitstreams` directory of the host computer,

- `adcChanADmaDirect_single.bit.bin` and `adcChanADmaDirect_dual.bit.bin` for single channel measurement or dual channel synchronous measurements respectively,
- the devicetree overlay requires that the bitstream is called `adcChanADmaDirect_wrapper.bit.bin` so rename (`cp`) the wanted bitstream accordingly on the Redpitaya when copying to the `/lib/firmware` directory on the embedded board.

The same directory holds the kernel module `data_dma_direct_core.ko` for communicating between the kernel handling Direct Memory Access (DMA) and userspace, as well as the devicetree overlay `adcChanADmaDirect.dtbo`. After copying these files to the Redpitaya in addition to the userspace acquisition software `adcChanADmaDirect_us` and the bash script automating the sequence for running the application `adcChanADmaDirect_us.sh`, launch once `adcChanADmaDirect_us.sh` to initialize the Redpitaya, namely by loading the kernel module, copying the bitstream to the right location in the GNU/Linux tree structure, and loading the bitstream in the FPGA. All these steps are automated thanks to the `adcChanADmaDirect.dtbo` devicetree overlay.

Once the Redpitaya has been initialized, the application `adcChanADmaDirect_us` will acquire data through the DMA, 1 Msamples for a single channel or 500 ksamples/channel interleaved for two channels. The output filename is `dump.bin` stored in the directory the executable has been launched from. This output file is to be transfered to the PC for processing.

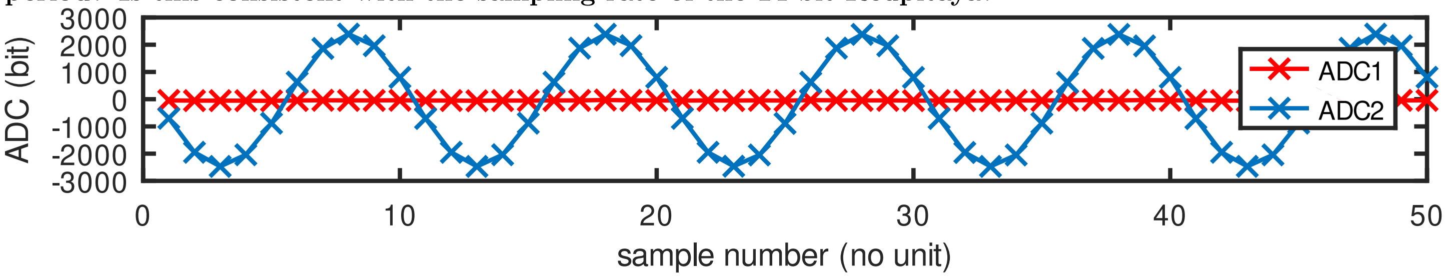
```
cd /root/bitstreams/
cp adcChanADmaDirect_dual.bit.bin adcChanADmaDirect_wrapper.bit.bin
cd /root/app/
./adcChanADmaDirect_us.sh  # loads bitstream and kernel drivers: blue LED
./adcChanADmaDirect_us
```

We can check the content of the binary file with `cat dump.bin | xxd | head` which should display, here in interleaved mode with only one channel connected to a synthesizer and the other left floating

```
00000000: 42fd d3ff 62f8 cbff 5bf6 cbff 09f8 c9ff  B...b...[.......
00000010: 8dfc c4ff 7302 d1ff 4607 d8ff 5609 d4ff  ....s...F...V...
00000020: ab07 d4ff 1e03 d8ff 3cfd cfff 61f8 ceff  ........<...a...
00000030: 5bf6 c5ff 09f8 cfff 8ffc cdff 7102 d2ff  [...........q...
```

which is well representative of signed **16-bit data** with one channel close to 0 and the other fluctuating widely.

Demonstrate understanding of data collection and processing by setting a frequency sythesizer to 12.5 MHz, amplitude 0.5 V_{pp} or +6 dBm, and collecting a dataset (named by default `dump.bin` in the current directory). Transfer the dataset to the PC and display the content of the file: how many samples can you find for each period? Is this consistent with the sampling rate of the 14-bit Redpitaya?



3 Data processing

The data are stored in binary format, signed 16-bit integers. With GNU/Octave: for a single channel acquisition

```
fs=125e6;f=fopen('dump.bin');d=fread(f,'int16');
f=linspace(-fs/2,fs/2,length(d));
plot(f,abs(fftshift(fft(d-mean(d)))))
```

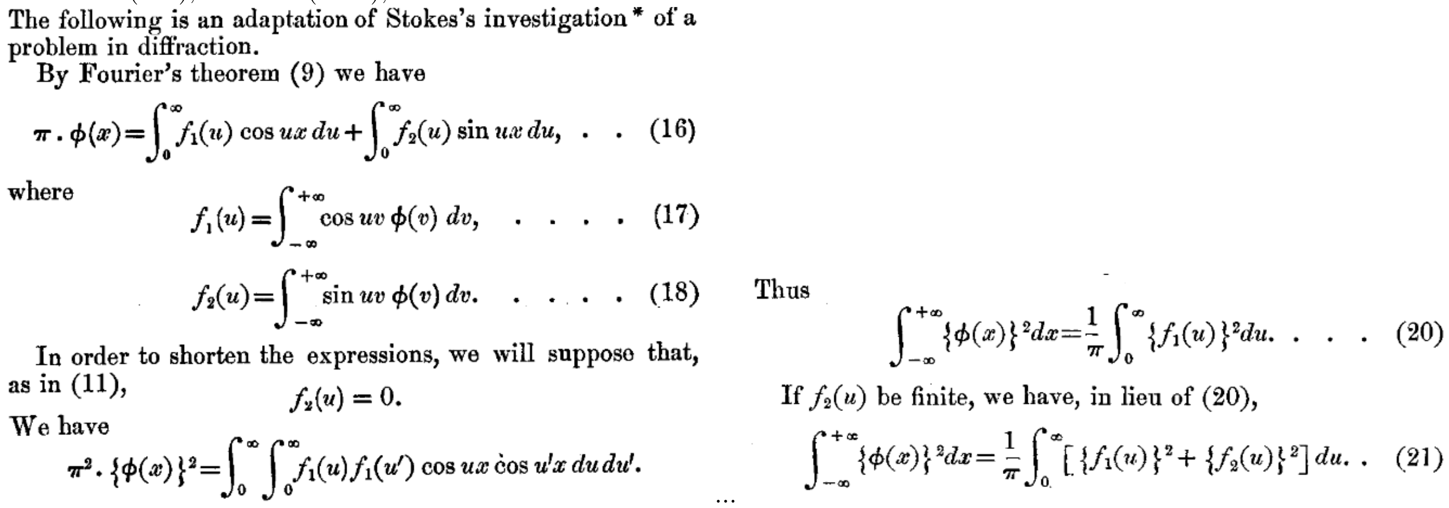
and for a dual channel acquisition

```
fs=125e6;f=fopen('dump.bin');d=fread(f,'int16');
d1=d(1:2:end);  % deinterleave
d2=d(2:2:end);
f=linspace(-fs/2,fs/2,length(d1));
plot(f,abs(fftshift(fft(d1-mean(d1)))))
```

Check that time-domain and frequency-domain representations of the signal match your expectations. Check your understanding of the time and frequency domain representation by changing the frequency to 6.25 MHz.

4 Time v.s frequency energy distribution

Energy must be conserved, whether data are displayed in the time domain or the frequency domain (Parseval theorem analyzed from its energy conservation perspective by Rayleigh). Indeed, as found in Lord Rayleigh, *On the character of the complete radiation at a given temperature*, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science **27**(169), 460–469 (1889),



The following is an adaptation of Stokes's investigation* of a problem in diffraction.

By Fourier's theorem (9) we have

$$\pi \cdot \phi(x) = \int_0^\infty f_1(u)\cos ux\,du + \int_0^\infty f_2(u)\sin ux\,du, \quad . \quad . \quad (16)$$

where

$$f_1(u) = \int_{-\infty}^{+\infty} \cos uv\,\phi(v)\,dv, \quad . \quad . \quad . \quad . \quad (17)$$

$$f_2(u) = \int_{-\infty}^{+\infty} \sin uv\,\phi(v)\,dv. \quad . \quad . \quad . \quad . \quad (18)$$

In order to shorten the expressions, we will suppose that, as in (11),

$$f_2(u) = 0.$$

We have

$$\pi^2 \cdot \{\phi(x)\}^2 = \int_0^\infty \int_0^\infty f_1(u) f_1(u') \cos ux \cos u'x\,du\,du'.$$

Thus

$$\int_{-\infty}^{+\infty} \{\phi(x)\}^2 dx = \frac{1}{\pi}\int_0^\infty \{f_1(u)\}^2 du. \quad . \quad . \quad . \quad (20)$$

If $f_2(u)$ be finite, we have, in lieu of (20),

$$\int_{-\infty}^{+\infty} \{\phi(x)\}^2 dx = \frac{1}{\pi}\int_0^\infty [\{f_1(u)\}^2 + \{f_2(u)\}^2]\,du. \quad . \quad (21)$$

...

The energy of the signal in the time domain is $\sum x_k^2$ computed in GNU/Octave as `sum(x.^2)`

The challenge in the frequency domain lies in properly normalizing the spectra. Many presentations on the web explain how to correctly normalize power spectra [1], so we will use the readily signal processing tool `pwelch` [2] here

```
pkg load signal

N=32768;
fs=1;
x=rand(N,1);x=x-mean(x);
window=rectwin(N);
% default is for pwelch to REMOVE the mean value
[pxx,f]=pwelch(x,window,0,N,fs,[],'no-strip');   % W/Hz
subplot(121);plot(f,10*log10(pxx));        ylabel('dBW/Hz')
subplot(122);plot(f,10*log10(pxx*fs/N));  ylabel('dBW/bin')
sum(pxx)
```

demonstrates that indeed `sum(pxx)` is equal to `sum(x.^2)` after making sure to compensate for the sampling frequency in the former quantity. Notice that `pwelch` is also available for SciPy as `scipy.signal.pwelch`

Another useful representation of a signal is neither in the time or frequency domain but by displaying the probability of each value. For plotting such a histogram, the `hist()` function from GNU/Octave can be used (Fig. 2).

1. **Display the histogram of the collected data, and check energy conservation, namely that the sum of energy in each bin matches the time-domain and frequency-domain energy calculation completed above. Doing so might require to know which bin is associated with each sample, which is the second argument `v` in `[u,v]=hist(x,N);`**

2. **Analyze and interpret the observed histogram of the collected sine wave. For a better understanding, plot the histograms of the synthetic signals for**

   (a) **a sine wave**

   (b) **a sine wave with the addition of 1/100th of gaussian noise (`randn()` function in GNU/Octave)**

[1] e.g. S. Hageman, *Real spectrum analysis with Octave and MATLAB*, EDN (2015) at `https://www.edn.com/real-spectrum-analysis-with-octave-and-matlab/`

[2] O.M. Solomon, *PSD Computations Using Welch's Method*, Sandia Report (1991) at `https://www.osti.gov/servlets/purl/5688766/`

(c) a sine wave with the addition of 1/10th of gaussian noise (`randn()` function in GNU/Octave)

(d) a triangle wave (`sawtooth()` function in GNU/Octave)

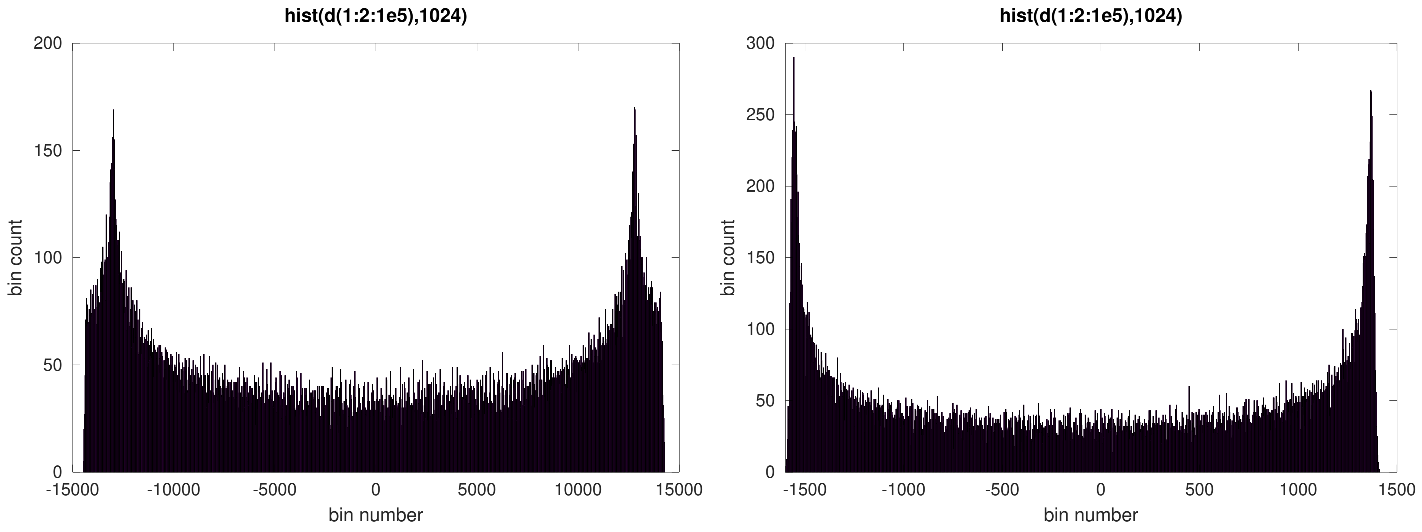


Figure 2: Histogram of an undersampled 250 MHz (left) and 850 MHz (right) sine wave sampled at 122.88 MS/s with the 16-bit Redpitaya allowing for aliasing.

5 Effective number of bits

While the physical number of bits of the LTC2145 is 14, not all are relevant. The effective number of bits is an experimental measurement of the number of relevant bits. Such a measurement is achieved by loading the ADC with a 50 Ω resistor (to avoid floating potential fluctuations) and recording the signal. The distribution of the recorded data is expected to follow a Gaussian noise distibution.

1. Display the histogram of the collected signal(s)
2. Display the histogram of a synthetic gaussian noise
3. The width at half height of the gaussian distribution is 2.4σ with σ the standard deviation, one of the two parameters (the other being the mean value) characterizing a Gaussian distribution. What is the standard deviation (in bits) of the recorded signal? in volts considering the full scale range is ± 1 V=2 V?

A gaussian distribution is only defined by two quantities, the mean value and the standard deviation. Assuming the mean value has been removed, demonstrate how a manual fit is easily achieved based on the standard deviation of the dataset and the maximum value of the histogram.

Fig. 1 displays an example of such a processing on experimental data with the ADC loaded by a 50 Ω resistor. The case of a sine wave is shown in Fig. 2

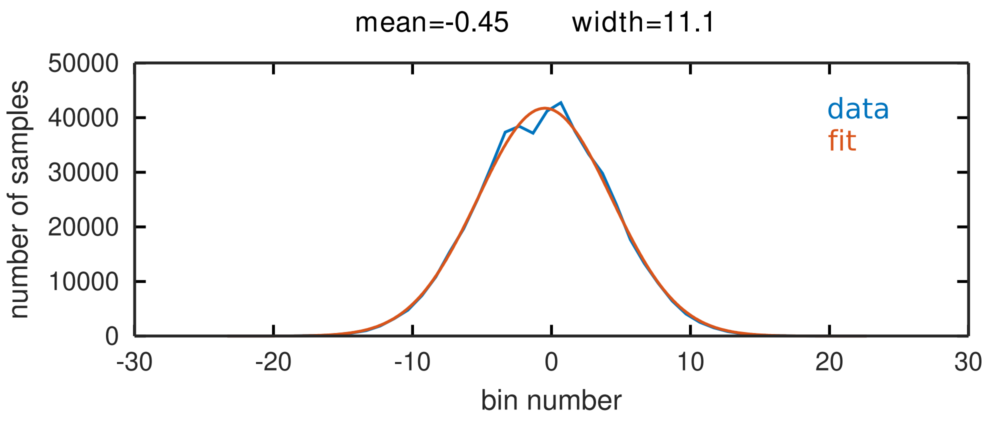


Figure 3: Fit of the bit value distribution of the ADC loaded on a 50 Ω resistor (noise).

Considering that the quantization noise of an ADC is $\sigma^2 = V_q^2/12$ with V_q the effective quantization level, what is the effective number of bits[3] $ENOB = \log_2(1 + V_{FSR}/V_q)$ if we equate σ and the standard deviation of

[3]see `https://pico-adc.markomo.me/ENOB/` (Feb. 2021)

the observed gaussian distribution to deduce the effective V_q?
Since the nominal number of bit is 14 or a full-scale range (in bits) of $V_{FSR} = 2^{14}$, then $\log_2(V_q) = \log_2(\sigma \cdot \sqrt{12})$ will tell us how many bits have been lost with respect to the ideal behaviour (1-bin width of the Gaussian distribution would mean 0-bit being lost).

Otherwise, the excellent `peakfit.m` at `https://terpconnect.umd.edu/~toh/spectrum/InteractivePeakFitter.htm` is used for automated Gaussian peak fitting. After recording the bin index and number of samples per bin, peak fitting is achieved with

```
pkg load signal
f=fopen('dump.bin');d=fread(f,'int16');
d1=d(1:2:end);d1=d1-mean(d1);
d2=d(2:2:end);d2=d2-mean(d2);
f=linspace(-125/2,125/2,length(d1));
[hh,xx]=hist(d1,floor(max(d1)-min(d1)));
signal=[xx' hh'];
[FitRes,GOF,bl,coeff,res,xi,yi,BootRes]=peakfit(signal,0,0,1,1,0,0,[0 max(xx)],0,0,0);
```

where `FitRes` holds the mean value and width of the peak in its second and fourth attributes respectively, and the result of the fit is in `(xi,yi)`.

A more accurate approach to estimating the standard deviation of the gaussian distribution is to use the fit: the width at half height will be returned as the fourth item of the first returned argument (`FitRes` in the example above). **Compare the width at half height from the fit with your manual measurement**.

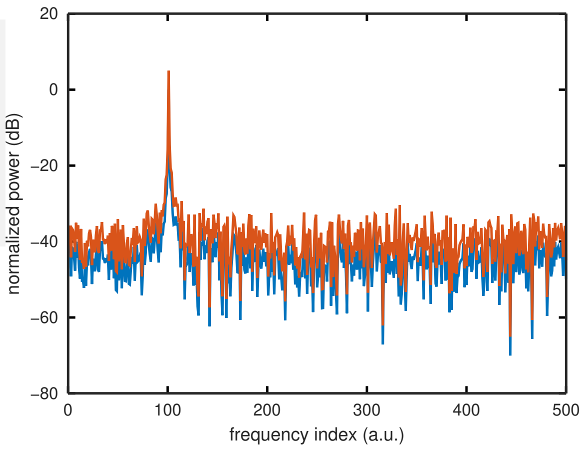
Computing the ENOB from a sine wave acquisition is discussed in [4]

A `pwelch` v.s. `fft`

While `pwelch` will properly normalize the output by accounting for the datalength and sampling frequency while `fft` will either normalize the forward Fourier transfrom, inverse Fourier transform or both depending on the implementation, the normalized spectra are similar as shown with

```
pkg load signal
x=sin([0:.1:100]*2*pi);x=x+randn(1,length(x))/10;plot(x)
y=abs(fft(x));y=y/max(y);
plot(20*log10(y(1:length(y)/2)))

[z,f]=pwelch(x,rectwin(length(x)),0,length(x),1,[],'no-strip');
z=z/max(z)
hold on
plot(10*log10(z)+5)
```



B Demonstration of energy conservation

This short GNU/Octave demonstration program illustrates how time domain (variance), frequency domain (power spectra) and probabilistic (histogram) representations of a same signal conserve energy:

```
pkg load signal
N=32768;
fs=100;
x=randn(N,1);x=x-mean(x);
x=x+0.5;
window=rectwin(N);
% default is for pwelch to REMOVE the mean value
[pxx,f]=pwelch(x,window,0,N,fs,[],'no-strip');   % W/Hz
subplot(121);plot(f,10*log10(pxx));         ylabel('dBW/Hz')
subplot(122);plot(f,10*log10(pxx*fs/N));  ylabel('dBW/bin')
[u,v]=hist(x,1024);
sum(pxx)*fs
sum(x.^2)
sum(u.*v.^2)
```

[4] S. Weaver & al., *ENOB Calculation for ADCs with Input-Correlated Quantization Error Using a Sine-Wave Test*, 22nd International Conference on Microelectronics (ICM 2010) at `http://www.benjamin.hershberg.com/wp-content/papercite-data/papers/2010-icm-enob-calculation-for-stochastic-adcs.pdf`