ADC basics for TDAQ (Lab 8)

Andrea Borga (andrea.borga@nikhef.nl, tutor)

Cairo Caplan (cairo.caplan@cern.ch,lab assistance)

Diogenes Gimenez (diogenes.gimenez@usp.br, lab assistance)

Table of Contents

Concepts of this lab

Figure 1 below shows the generic signal flow of a Data AcQusition (DAQ) system to perform Analog to Digital Conversions (ADC).

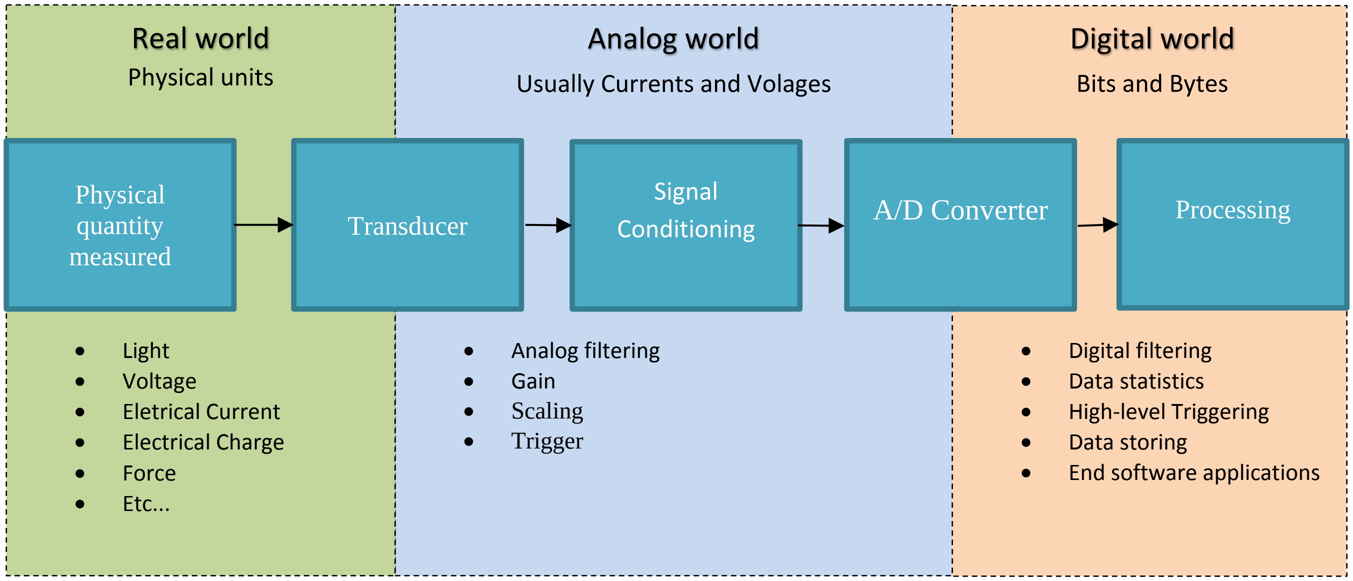


Figure 1- Basic ADC DAQ chain

The scope of this laboratory experience is to understand and experiment with the following components of a Data AcQuisition system:

1. Triggers:
    1.1. External triggers: accelerator like type of triggers
    1.2. Triggering on the signal: astroparticle like type of triggers
2. Analog to Digital Converter (ADC): we will try to understand fundamental parameters that come into play when measuring with ADC; as for example its resolution, speed, bandwidth, acquisition window, etc.

Lab setup

Figure 2 below depicts the setup for Lab8.

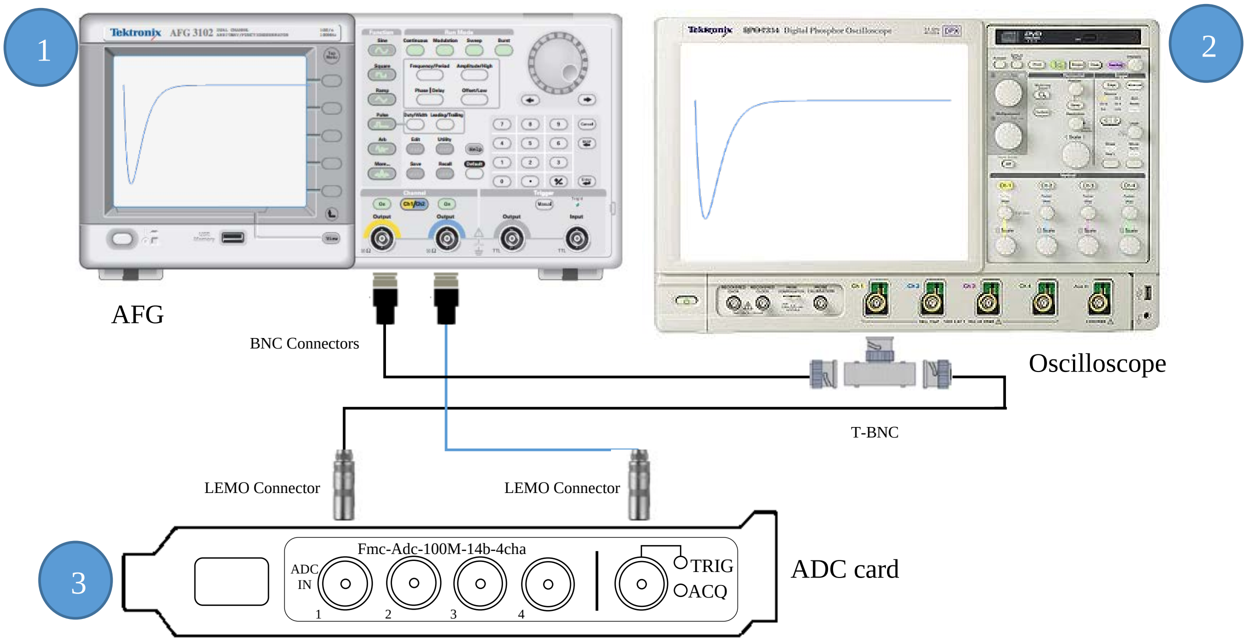


Figure 2 - Equipment setup

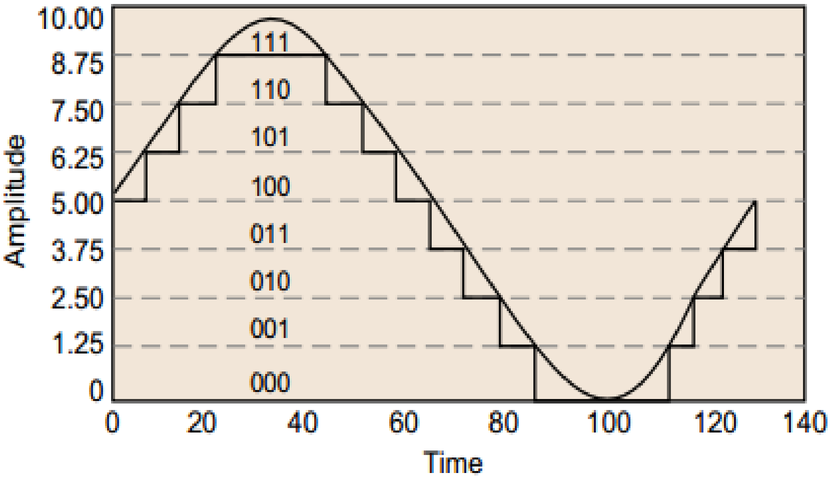
1. Tektronix AFG3252: Arbitrary Function Generator, is the **input** of our system
2. Tektronix DPO70404C: Oscilloscope, it the **monitor** of our system (to cross-check that the signals fed to the system are truly what intended)
3. SPEC+FMC ADC card plus host PC: this is the **core ADC DAQ**
    3.1. FPGA Mezzanine Card (FMC) ADC: http://www.ohwr.org/projects/fmc-adc-100m14b4cha
    3.2. Simple PCI Express Carrier (SPEC) card: http://www.ohwr.org/projects/spec/wiki
    3.3. Linux based host PC

Introduction to Analog to Digital Conversion (ADC)

An Analog to Digital Converter (also known as "digitizer") is a device which converts the level of an analog signal into an integer number which is closest to the real value of the signal in terms of ratio. There is a limit for the number of choices of this integer which is determined by the number of bits used for the digitization.

Example: We have a voltage signal whose range is [0 - 10] Volt, and we have a digitizer which has just 3 bits. We can have 8 different integer numbers with 3 bits. Each of these numbers will correspond to a voltage. See table below.

Voltage	Digitizer bits	Integer
0.00 - 1.25	000	0
1.25 – 2.50	001	1
2.50 - 3.75	010	2
3.75 – 5.00	011	3
5.00 - 6.25	100	4
6.25 – 7.50	101	5
7.50 – 8.75	110	6
8.75 – 10.00	111	7



This kind of conversion is performed at regular intervals (**samples**), and the repetition (frequency, Hz) is called **sampling frequency** (normally expressed in samples per seconds). Thus, digitization is not only performed in the signal domain, but also in the time domain.

Also note that to improve the precision of the measurement one should:

- Increase the number of bits, so each corresponding range is small with respect to the signal to be sampled
- Sampling should be done at the proper frequency (see Nyquist frequency); and even more importantly, a precise sampling clock should be used, such that the deviation between consecutive samples is kept as constant as possible.

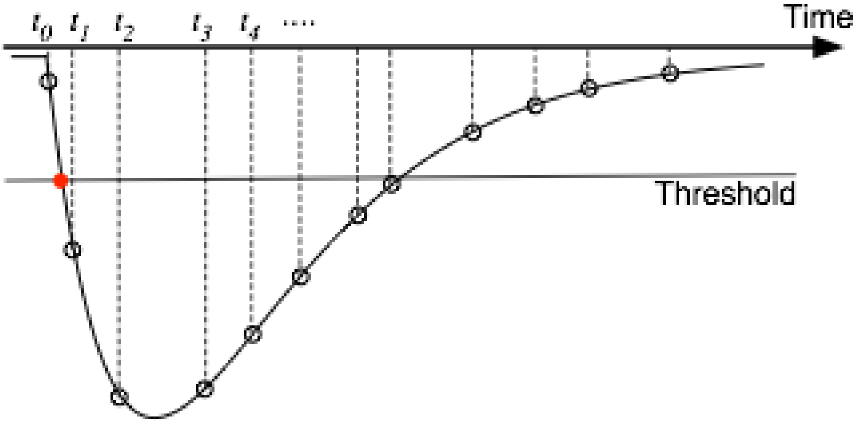
If you are interested in more details about ADC parameters, please check: http://www.analog.com/en/analog-to-digital-converters/products/index.html

If are interested in understanding more in detail the importance of clock stability (jitter), please check: http://anlage.umd.edu/Microwave%20Measurements%20for%20Personal%20Web%20Site/Tek%20Intro%20to%20Jitter%2061W_18897_1.pdf

The ADC card we will be using has 14 bit voltage resolution and a typical conversion time of 10 nsec (100 MS/s). In this exercise, we will operate the ADC in order to understand and push its limits.

The Measurement

Qualify in the best possible way a set of signals mimicking real experimental equipment. The tutors will provide some pre-cooked ones (sine waves, positive pulses, scintillator like pulses) but you can try to think of your real world input.



In general terms, a digitization process is characterized by the following:

- Signal range: how much the signal can vary to be correctly interpreted by the device (voltage)
- Sampling frequency: the rate at which it can convert an analog value to a discrete signal (bits)
- Latency: How long does the device takes to finalize the acquisition process in the chain (time)
- Noises: physical quantities responsible for the signal deterioration (e.g. added noise, intrinsic noise, quantization errors, etc.)

The best measurement is achieved by understanding and controlling those parameters. **The designers (so YOU) have to decide how to setup your TDAQ:** choose a trigger type, define the acquisition windows, find the signal in the window, maximize the scaling for increasing the accuracy of the measurement, etc…

Architecture of a Linux Device Driver for the PCIe Card

Before running the DAQ system, an **overview on the communication between the software and the hardware layers and an introduction to the role of device drivers**, is fundamental to understand. *Figure 3* shows a simplified diagram of the Layers of a Linux Operating System (OS).

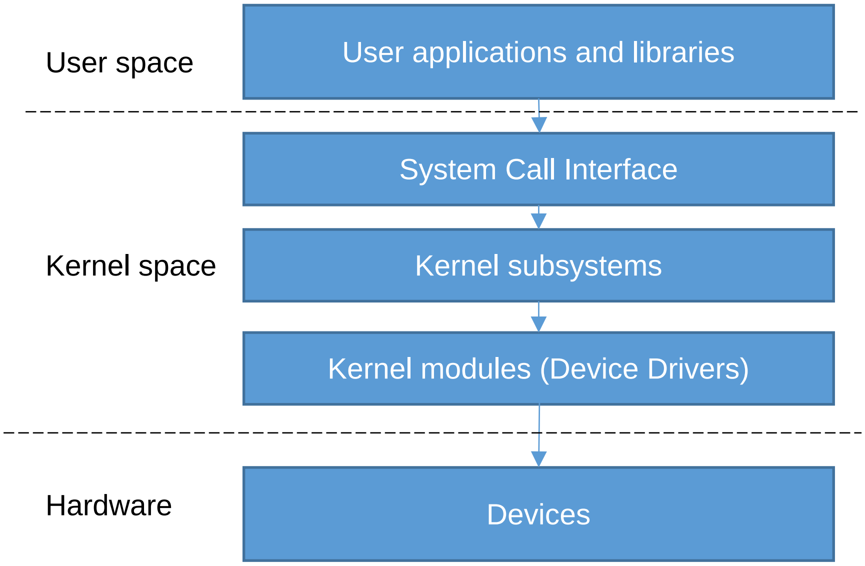


Figure 3- Layers of a Linux OS

The basic control is on the hardware peripheral itself. The lowest level software for this system resides in the kernel of the OS as a “device driver.” There are certain control/status registers on the ADC card. These registers can be accessed just like a regular memory access in a C program.

As seen earlier in this document, in our context, the ADC DAQ hardware used is composed of two electronic boards: the SPEC carrier board and the FMC ADC module. The first board is attached directly to a PCIe slot connector on the PC, and is the host for the ADC module. The SPEC card acts as a bridge for the electrical signals of the ADC FMC to be interfaced and converted to PCIe. For the scope of this lab, the “bridging” is done by some “black box” electronics. Thus from a software perspective, one kernel module is instantiated to use the SPEC card, another one for the FMC ADC module.

For simple sensors and devices vendors provide information about their operation and use; which can be seen as a map of internal registers and/or procedures required to perform operations or change their current states (more about in Exercise 12 " DAQ Online Software”). For the sake of simplicity and reusability of software code, software engineers created the concept of frameworks. Among others, this concept was used at CERN for creating a flexible interface for the development of input and output drivers. The target is for very-high-bandwidth devices, with high-definition time stamps, and a uniform metadata representation. Such framework is named the ZIO (with Z standing for “The Ultimate I/O” Framework).

In short, the way ZIO provides to talk to our hardware is through two different channels, one for control and one for data. *Figure 4* shows the two data flows.

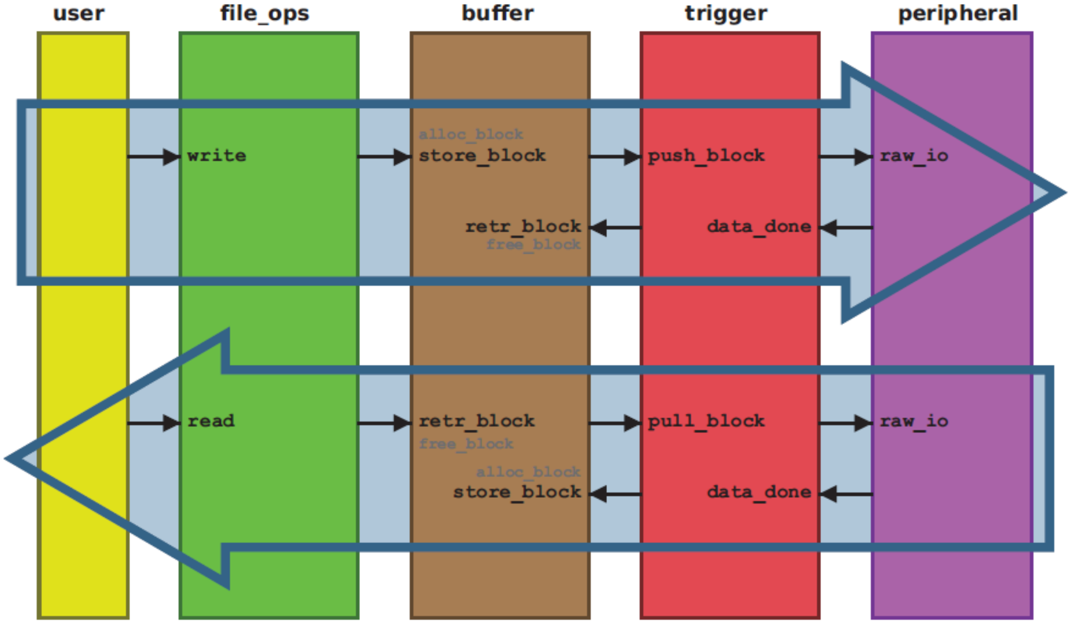


Figure 4 - ZIO data pipeline

When designing a piece of low-level software to communicate to an acquisition device, it is important to choose the way data streams should be passed back and forth the hardware device. To illustrate this they can be qualified in three types: **Polling mode, Interrupt based, and DMA (Direct Memory Access) transfers**.

- **Polling mode:** the CPU checks the state of the device's registers every time it can, this has the advantage of providing a fast and low latency communication and data transfer between them, but at the cost of high CPU load.
- **Interrupt based**: this type of transfers eases the CPU load time as it only have to care about the hardware when it tells the CPU to do so; it has the advantage of a lower impact on CPU loads and fast transfer, but with variable latency.
- **DMA transfers**: relies on a shared memory area the CPU provides to the DMA controller to work on the transfer with the hardware device, this frees completely the CPU from controlling the hardware transfer representing a true concurrent system. Highest of all transfer rates allied to an acceptable latency.

Operating the setup

1) Construct/confirm the experimental setup according to the sketch you find at the beginning of this document. **Check that both outputs of the AFG are switched OFF.**

   The signal generated by the Channel-1 (**source input**) of the AFG unit is supplied to the oscilloscope, and to the Channel-1 input of the ADC card. Use a T-BNC to split the signal at the AFG output. Also check if the Channel-2 output (**trigger input**) of the AFG is connected to the Trigger input of the ADC card. Again you can monitor the trigger on the scope by spitting with a T-BNC on the AFG output.

2) Using the AFG, set Channel-1 to generate a sine waveform with a frequency of 1 MHz and amplitude of 1 Vpp. Completed this operation, set Channel-2 to generate a pulse waveform with a frequency of 1 KHz and amplitude of 4 V. **Don't turn them ON yet, and check if they have not an amplitude above 5 Vpp**.

3) Login to the PC and reset the lab8 directories, so all the work/changes done by the previous group are removed and a fresh copy of the files are installed

4) In order to load the drivers, open the `~/adc` directory and execute the script for loading the drivers:

   ```
   $ cd ~/adc
   $ ./load_drivers.sh
   ```

   You need the root password in order to execute the command above. Ask your tutor for it.
   Spying the content of our script shows the correct order for loading the different kernel modules and set the user permission to talk with the ADC board. It is also possible to check the current kernel modules loaded by issuing the command `lsmod`.

5) Now it is time to test our ADC, turn ON only Channel-1 of the AFG and check if the signal is correctly displayed by the oscilloscope. If yes go to the folder `libtools` and run the acquisition program which will subsequently show an acquisition plot:

   ```
   $ cd Trigger_ext
   $ ./fald-acq -a 1000 -b 0 -n 1 -l 1 -g 1 –r 10 -e -X 0100
   ```

   Where acq-program has the following parameters:

   ```
   --before|-b <num> number of pre samples
    --after|-a <num> n. of post samples (default: 16)
   --nshots|-n <num> number of trigger shots
    --delay|-d <num> delay sample after trigger
   --under-sample|-u|-D <num> pick 1 sample every <num>
   --external|-e use external trigger
   --threshold|-t <num> internal trigger threshold
   --channel|-c <num> channel used as trigger (1..4)
   --range|-r <num> channel input range: 100(100mv) 1(1v) 10(10v)
   --negative-edge internal trigger is falling edge
   ```

```
--loop|-l <num> number of loop before exiting
--graph|-g <chnum> plot the desired channel
--X11|-X Gnuplot will use X connection
```

Did the program made the acquisition?

6) Turn ON Channel-2 of the AFG. Does it run now?

7) Now that we know that the ADC is working properly we can go to some real time data analysis. Bearing this in mind, there is a small program called `V_t_continous.C` which uses the ROOT framework functionalities and runs on top of `fald-acq`. To run `V_t_continous.C` issue the command:

```
$ root V_t_continous.C
```

8) Try now to modify Channel-1 frequency in steps of Hz while checking if the acquired waveform remains true to it. Check also if the measured amplitude is the same as set on the AFG or, say, twice its value. **Make sure you are changing the frequency NOT the amplitude.**

9) One issue you may get while changing the frequency is the ‘non-stopping’ graph, meaning it is continuously sweeping horizontally. What causes this?

   Since we are using Channel-2 of the AFG as an external trigger, its triggering frequency dictates how, or at which point in time, the acquisition of Channel-1 signal starts. In practice: if the frequency of our sine wave is not a harmonic of Channel-2 pulse, i.e. not an integer multiple, the ADC doesn’t capture the signal at the same phase.

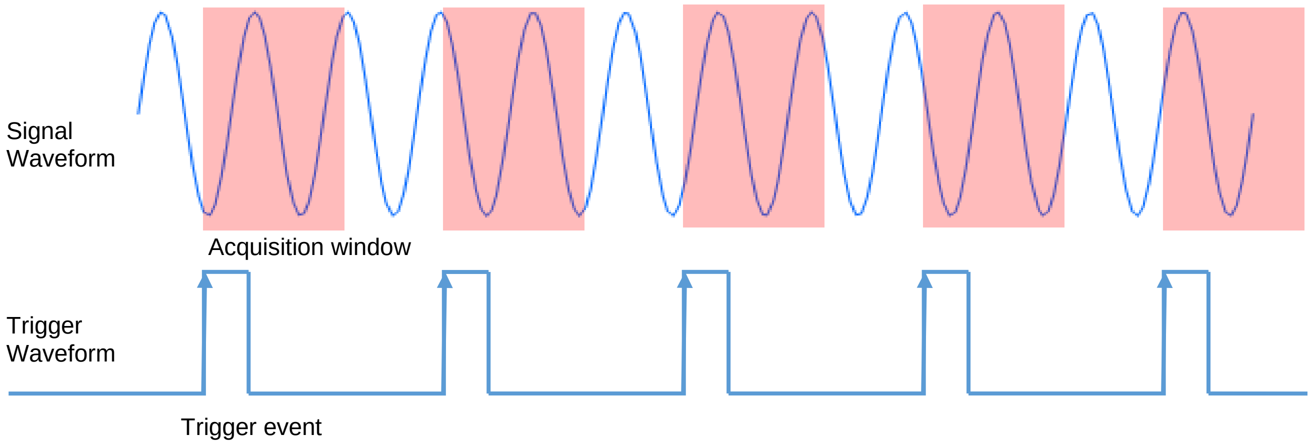


Figure 5- Signal frequency non multiple of Trigger frequency

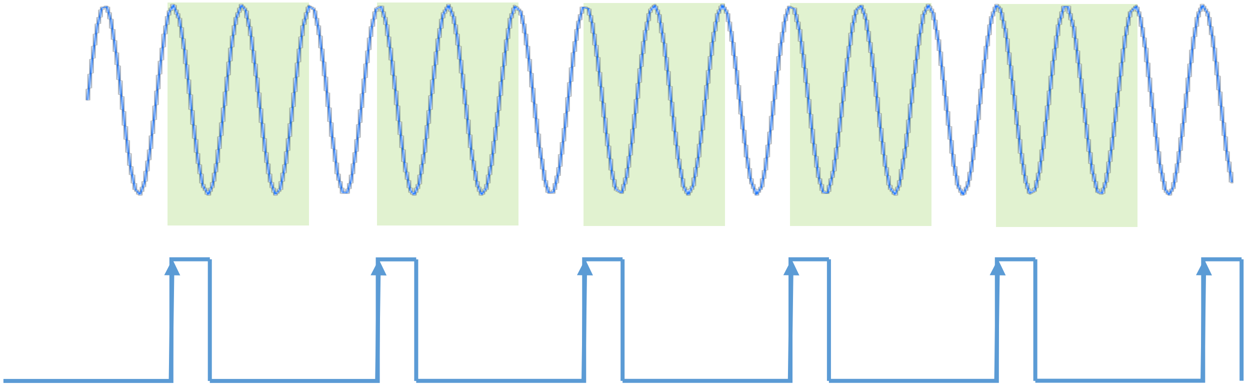


Figure 6 - Signal frequency multiple of Trigger frequency

10) Going further, let's push the frequency of our signal to the limits of the ADC capabilities. Recalling that our ADC specifications say that the default sampling rate is 100 Million Samples/s try to increase the signal frequency in steps of MHz. *Please note: it is recommended to decrease the number of samples in our acquisition window*; otherwise it would become hard to analyze the signal in a cloud of points.

    For this, open the `V_t_continous.C` file with your favorite editor and change the number of post samples (-a parameter) to 100 or less, it is located on the line which calls `fald-acq`.

    Run `root V_t_continous.C` again.

    Can you see the relation between the acquisition window and the signal speed?

11) You can see that the closer you get to 100 MHz the worst the acquisition signal looks like. Can you define the maximum AFG signal frequency where our sine wave keeps its shape in a single acquisition shot (i.e. you can still see a sine wave with the same frequency as the original signal)?

    This value is called the **Nyquist frequency**. The **Nyquist theorem** states that: *the minimum sampling rate of an acquisition device must be at least two times the maximum frequency of the original signal, otherwise information would be lost in the process*. See *Figure 7* below: the original signal is in blue, the sampling points are pointed by the black arrows and the acquired data is represented in orange. When this criteria is not respected an effect called aliasing appears.

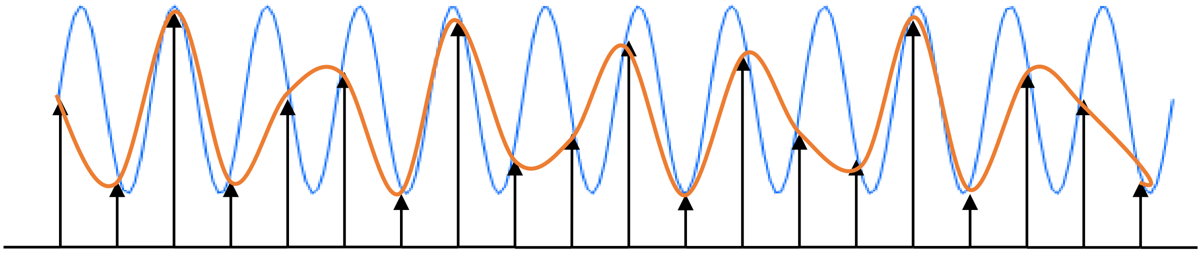


Figure 7 - Aliasing effect

12) The maximum Nyquist frequency is computed on a signal ideally composed of a single sinusoid. In fact any real function of a complex signal is mathematically described as the sum of a series of trigonometric functions in the **frequency domain** (called *signal spectrum*) instead of time. **Fourier series and transforms** base their analyses on this concept. The spectral representation of a sine wave is rather simple: it is only a single line centered around the frequency it coverts from the time domain. The spectral content is more complex for different waveforms signals such as square, saw-tooth and other more complex signals.

In order to check how our acquisition systems behaves with complex signals go back to the frequency of 1 MHz but change the type from sine-wave to square signal. Increase the frequency to values below Nyquist criteria.

13) Ask the tutor to present a real time Fast Fourier Transform (FFT) and **ask further questions!**